

Artificial Intelligence and Machine Learning

Project Report

Semester-IV (Batch-2022)

Flat Price Prediction



Submitted By: -

Supervised By: - Shubham Singhal

Naman Bassi	2210990594	G11
Harsh Pawar	2210991614	G11
Arshad Malik	2210990580	G11
Nikhil <u>Guleria</u>	2210990610	G11

**Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab**

Predicting Flat Prices with Machine Learning Methods

REPORT BY HOUSEFED CODERS

Bacheloring in CSE
Date: 13th May, 2024
Supervisor: Shubham Singhal
Chitkara University

Abstract

In this study, the machine learning algorithms k-Nearest-Neighbours regression (k-NN), Random Forest (RF) regression, Linear Regression and Decision Trees were used to predict house prices from a set of features in the Ames housing data set. The algorithms were selected from an assessment of previous research and the intent was to compare their relative performance at this task.

Software implementations for the experiment were selected from the scikit-learn Python library and executed to calculate the error between the actual and predicted sales price using four different metrics. Hyperparameters for the algorithms used were optimally selected and the cleaned data set was split using five-fold cross-validation to reduce the risk of bias. An optimal subset of hyperparameters for the two algorithms was selected through the grid search algorithm for the best prediction.

The Random Forest was found to consistently perform better than the k-NN algorithm in terms of smaller errors and be better suited as a prediction model for the house price problem.

With a mean absolute error of about 9 % from the mean price in the best case, the practical usefulness of the prediction is rather limited to making basic valuations.

Contents

1	Introduction	1
1.1	Research Question	1
2	Background	2
2.1	Machine learning algorithms	3
2.1.1	k-Nearest neighbours regression	3
2.1.2	Random forest regression	5
2.2	The Ames Housing data set	5
3	Methods	7
3.1	Cleaning data	7
3.1.1	Normalizing data.....	7
3.1.2	Encoding categorical data	8
3.1.3	Missing values	9
3.2	Prediction and evaluation.....	10
3.2.1	Splitting the data	11
3.2.2	Algorithm hyperparameters	12
3.3	Error metrics	13
3.3.1	Mean absolute error (MAE).....	13
3.3.2	Mean squared error (MSE)	13
3.3.3	Median absolute error (MedAE)	14
3.3.4	Coefficient of determination (R^2)	14
4	Results	15
4.1	Grid search over hyperparameters.....	15
4.2	k-NN hyperparameters	15
4.3	Random forest hyperparameters.....	17
4.4	Algorithm comparison	18
5	Discussion	20

6 Conclusions	22
Bibliography	23

Chapter 1

Introduction

Accurately estimating the value of real estate is an important problem for many stakeholders including house owners, house buyers, agents, creditors, and investors. It is also a difficult one. Though it is common knowledge that factors such as the size, number of rooms and location affect the price, there are many other things at play. Additionally, prices are sensitive to changes in market demand and the peculiarities of each situation, such as when a property needs to be urgently sold.

The sales price of a property can be predicted in various ways, but is often based on regression techniques. All regression techniques essentially involve one or more predictor variables as input and a single target variable as output.

In this paper, we compare different machine learning methods performance in predicting the selling price of houses based on a number of features such as the area, the number of bed- and bathrooms and the geographical position.

1.1 Research Question

How well can house prices be predicted by using k-Nearest neighbour, Random forest regression, Linear Regression and decision trees?

Chapter 2

Background

The field of Data Science is rather young, having taken form over the last half-century as a discipline distinct from statistics. It is also rapidly growing with many interesting advancements in recent years, most notably within Machine Learning (ML). This has resulted in an increase in media attention as well as funding of AI related businesses and research projects.

In *50 years of Data Science* [1] Donoho comments on the history of Data Science and questions whether it is really different from statistics. With regards to Machine Learning, he points to a study he conducted that compared a set of highly-cited and glamorous classifier methods such as Random Forests and k-Nearest neighbour to a simple linear classifier applied on the same problem. The study found that the simpler method did not only perform similarly, but had a lower worst-case regret. This suggests that when benchmarked, more advanced ML algorithms are not necessarily better when put in practice, which highlights the need for making algorithm comparisons.

A study using similar techniques was made on predicting the sales price of used cars. [2] This problem is similar to predicting house prices and arguably simpler because it is dealing with cars, commodities that aren't geographically fixed and are often highly standardized. The methods used were Multiple Linear Regression, k-Nearest Neighbours, Naïve Bayes and Decision Trees, including Random Forest. Cross validation was used for finding the optimal hyperparameters, such as the 'k' for k-NN.

The house pricing problem was approached by Baldominos et al. [3] from the viewpoint of finding investment opportunities. They formulated the regression problem and used several Machine Learning algorithms such as k-Nearest neighbour, variations of neural networks and decision trees. Another study by Oxenstierna [4] investigated it for the purposes of valuation of houses. The

data set included 5000 entries. Again, the k-Nearest neighbour method was used as well as Artificial Neural Networks, to minimize the median absolute percentage error of the prediction. The methods performed similarly at around 8-9 % Median Absolute Percentage Error.

2.1 Machine learning algorithms

In this study two machine learning algorithms were compared against each other in order to investigate which one is more successful in predicting housing prices. As mentioned in the previous section, Baldominos et al. [3] performed a similar study in which they compare four machine learning algorithms for housing prices. In their study they found that the Random forest regression algorithm predicted with the smallest error followed by k-Nearest neighbours regression. In the study performed by Oxenstierna [4] k-Nearest neighbours regression and Artificial neural networks are suggested as methods for predicting house prices. Even though Oxenstierna finds the Artificial neural networks to perform better in many cases this study has excluded Artificial neural networks in order to limit the scope of the report and the time frame for the project. Since both reports study the performance of k-Nearest neighbours regression, the algorithm will be studied in this report as well. Since Baldominos et al. [3] finds that Random forest regression gives the least error for predicting house prices and Oxenstierna [4] mentions it as relevant for future work it will also be part of the study and compared against the k-Nearest neighbours regression algorithm.

2.1.1 k-Nearest neighbours regression

k-Nearest neighbours (k-NN) is a non-parametric algorithm that can be used for both classification and regression problems. The algorithm relies on the assumption that any item in the data set should have a similar value for the prediction target if they share similar values for other features. In our particular case, the house price is the target variable that is predicted by the k number of neighbours that are the most similar in the features.

The data can be conceptualized as points in a n-dimensional cartesian space. As an example, in the 2-dimensional case, we have two features of which the values are represented as points on a plane. Figure 2.1 illustrates the case when $k = 3$.

In practice, the algorithm calculates the distance to all other samples from the training data and selects the data item from the training data that is the

closest for prediction. However, predicting the value of a sample by the single closest neighbor from the training data does not capture much information, thus multiple data items is used for prediction in training. If the prediction is based on the k nearest neighboring items to the one being predicted (thus the name of the algorithm) more information underlies the estimation and thus a more accurate prediction could be obtained. Figure 2.1 illustrates the case when $k = 3$.

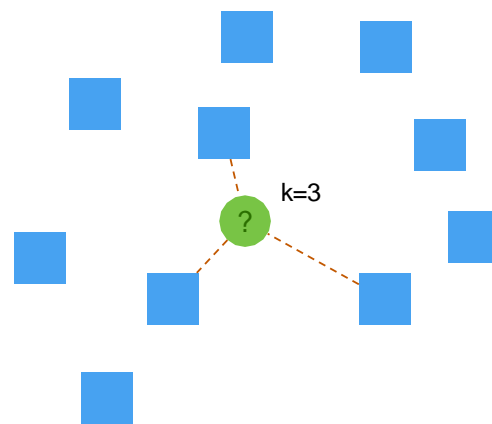


Figure 2.1: k -nearest neighbor algorithm with $k=3$. The three nearest samples are identified by lines from the test sample.

Since we are dealing with regression, we are concerned with determining a numerical value for the unknown variable. Taking the mean of the nearest neighbors is a simple approach. However some of those k points could be much more distant than others. To combat that, weights can be added to each neighbor according to some function dependent on the distance. One approach is to weigh them inversely proportional to the distance. In many cases the latter approach, inverse distance weighted average, performs better than uniform weights, i.e. no weights.

The number of neighbors to include in the calculation (i.e. the size of k) is in practice determined by trial, comparing prediction errors for different values of k . [5] [6]

2.1.2 Random forest regression

Random forest is an algorithm which can be used both for classification and regression. Random forest models are constructed by using a collection of decision trees based on the training data. Instead of taking the target value from a single tree, the Random forest algorithm makes a prediction on the average prediction of a collection of trees. The decision trees themselves are constructed by fitting to randomly drawn groups of rows and columns in the training data. This method is called bagging, and results in a reduction of bias as each tree is built on different parts of the input at random. The method of averaging the predictions of decision trees reduces the overfitting that can occur when using single decision trees. [7, pp. 587-588] [8]

The number of trees in the Random forest is an important hyperparameter of the algorithm called 'n_estimators' and the more trees used the more will overfitting be prevented. The tradeoff however, is an increase in the computation time needed. 'n_estimators' will be tested with different values in this study. Another hyperparameter of the Random forest algorithm is the 'criterion' hyperparameter which determines what error metric to use for measuring the quality of splits in the trees in the Random forest. The value can be either mean squared error or mean absolute error. A third hyperparameter that is particularly interesting for this study since there are a lot of features in the data set is 'max_features' which controls the number of features to consider when building the trees.

2.1.3 Linear Regression

Linear Regression is a supervised machine learning model that attempts to model a linear relationship between dependent variables (Y) and independent variables (X). Every evaluated observation with a model, the target (Y)'s actual value is compared to the target (Y)'s predicted value, and the major differences in these values are called residuals. The Linear Regression model aims to minimize the sum of all squared residuals. Here is the mathematical representation of the linear regression:

$$Y = a_0 + a_1X + \epsilon$$

The values of X and Y variables are training datasets for the model representation of linear regression. When a user implements a linear regression, algorithms start to find the best fit line using a_0 and a_1 . In such a way, it becomes more accurate to actual data points; since we recognize the value of a_0 and a_1 , we can use a model for predicting the response

2.2 The Ames Housing data set

In order to train a reliable machine learning model a sufficiently large data set was required. The Ames Housing data set, compiled by Cock [10], is a fairly complete data set consisting of almost 3000 entries for houses in the city of Ames, Iowa with as much as 80 different variables describing various property details. Because of the number of entries and features, the Ames Housing data set was selected as a good fit for this project with enough data for both training

and testing the machine learning model. Any details about the data set can be found in the Data Documentation [11].

The data set includes a large number of variables describing almost every aspect of the properties that might be of interest when evaluating a house. There are 23 nominal, 23 ordinal, 13 discrete and 20 continuous variables for each property. The nominal variables are for example the roof type or the material on the exterior facade of the property. The nominal variables have

about 5 to 10 possible values although three of the variables have about 20 different possible values. These are being "one-hot" encoded for this study, which is described in more detail in section 3.1. The 23 ordinal variables refer to various types of qualities of the properties which includes the overall quality of the property, quality of the basement, etc. Some of the ordinal variables take on numerical values while others use categorical values. The categorical values have been "one-hot" encoded while the numerical values have been left as is. Almost all of the 13 discrete variables describe a count of some kind. Examples are number of fireplaces or bathrooms. Since these variables are numerical by default, they have only been treated for missing values. The 20 continuous variables are mostly used for describing areas of various attributes of the house or of the entire lot. Like the discrete variables these do not require any further encoding to use them in a prediction model.

Chapter 3

Methods

In order to answer the question about what machine learning method is better to use for the house price problem the algorithms k-NN and Random Forest, as motivated in section 2.1, have been compared in terms of their prediction accuracy. Instead of implementing the algorithms from scratch for this study, algorithms from the *scikit-learn* library have been used. It is a state-of-the-art library part of the scikit suite of scientific toolkits for Python. We have also used the “Our Python” data analysis library Pandas. Prior to comparing the algorithms, the data set has been pre-processed and cleaned in order for the algorithms to take the data as input. Moreover, a method for evaluating the data has been established and finally the machine learning algorithms have been executed with the cleaned data set and tested with different values for relevant hyperparameters for prediction.

3.1 Cleaning data

Machine learning algorithms are largely implemented to only take data that is in a numeric format as input. More than half of the columns in the Ames Housing data set are non-numerical and need to be encoded, in this case using one-hot encoding and labeling. Additionally, various columns contain some empty values that have been dealt with in different ways as described in section 3.1.3.

3.1.1 Normalizing data

The numerical variables of the data set take on a large range of values and depending on the column these ranges can be quite different. In order for this

not to introduce bias normalization has been applied, scaling the numbers to the range $[0, 1]$. The Euclidean norm has been used for normalizing data in this study, although there are other options. The Euclidean norm is denoted

$\|x\|_2$ and is calculated with the following formula $\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}$ where $x_1 \dots x_n$ are all of the values of a feature in the data set. Each value of the feature is then normalized by dividing it by the Euclidean norm. [12] [13] [14] Normalization with the Euclidean norm has been used for all features in the data set since all non-numeric features are encoded into numeric as is explained in the following section.

3.1.2 Encoding categorical data

Many of the variables of the data set are categorical, and take on a limited set of values. One example is the nominal variable "Street" which represents the type of road access to the property and takes on the values "Grvl" for gravel and "Pave" for paved. Such categorical values can not be interpreted by conventional machine learning algorithms without preprocessing them to a numerical format. There are two types of categorical variables in the data set; ordinal and nominal. The difference is that the ordinal variables carry some kind of natural ordering between them. For example, the "LandSlope" variable, categorizes the slope of the property using one of three values - gentle, moderate or severe - that are intuitively ordered. Labeling is the simple process of pairing a numerical value to each of the ordinal values so that the ordering is preserved. This can be done by simply assigning integer values starting from 1 for the lowest order value, 2 for the next and so on. [15] Figure 3.1 is an example with four arbitrary entries out of the data set and how they might be encoded.

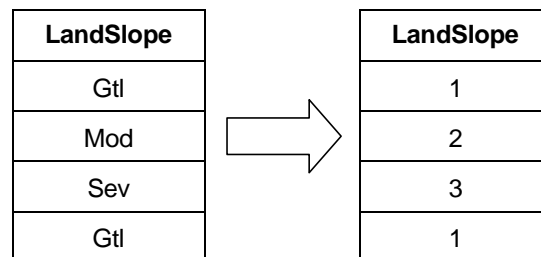
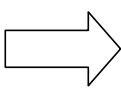


Figure 3.1: Labeling of ordinal variables.

For encoding nominal variables, however, labeling would suggest that there

is an order between the values when there actually is none. Therefore, a technique called one-hot encoding is better suited. Essentially, a column is made for each of the categorical values. Then a binary value is inserted signifying whether that entry has the categorical value or not. If the entry had the categorical value, 1 is inserted and 0 otherwise. [15] The illustration in figure 3.2 shows an example of how the three categorical values for "RoofStyle" are one-hot encoded.

A potential problem with one-hot encoding is for variables with a high cardinality i.e. many possible values. The issue is that it could generate far too many columns to be practical. In the Ames data set, most nominal variables take on less than 20 values, and most of them between five and ten. Thus, one-hot encoding have been used for all.



RoofStyle
Flat
Gable
Shed
Flat

Flat	Gable	Shed
1	0	0
0	1	0
0	0	1
1	0	0

Figure 3.2: One-hot encoding of nominal variables.

3.1.3 Missing values

Values for entries in the data set that are empty are not useful for the model and thus have been handled in the pre-processing. Fortunately, the data set is fairly complete, containing only a few missing values. These have been processed differently depending on the column. In the nominal and ordinal columns there are a lot of "NA" values. Pandas interprets this as an empty cell through coercion, however according to the Data Documentation the value "NA" represents that a feature is not present rather than that it is unknown. For example, the columns "PoolQC", which is an ordinal variable describing the quality of the pool, and "PoolArea" has the value "NA" for most properties, indicating that there is no pool rather than the information being unknown. Thus, for these columns the value "NA" is not interpreted as an empty value.

However, there are some entries that are empty because of missing values in nine of the ordinal and nominal columns. Each of them except one have between one and four missing values and one of them has 23 missing values.

Since there are quite few rows with missing values, 37 in total, for the ordinal and nominal columns, these rows have been removed from the data set. This does not impact on the reliability of the model as it is such a small fraction of the total data [16].

For the columns representing continuous variables like areas the value "NA" represents an actual missing value. Since there are quite few rows with missing values for areas, 28 in total, these have without affecting the output of the model been removed. However, a continuous-valued column that stands out is "LotFrontage" with 490 missing values. It represents the street length connected to the property in linear feet. 490 rows is quite a significant portion of the rows in the data set (17 %). Removing that many rows could impact the results. Instead, the missing values have been imputed meaning that the values of cells with missing values are set to some statistic like the mean or median. In this case the value has been set to the mean of the values present in the column. Although the "correct" value for the cell cannot be identified, the column as a whole will be reliable enough to support our model and facilitate a better analysis than if the rows were removed entirely [16].

Finally the column "MSZoning" needed cleaning. The column represents a nominal variable that describes the general zoning classification of the sale with values like "RH" for "Residential High Density" or "I" for "Industrial". For 29 rows the value of the column is abbreviated with "(all)", for example "I (all)". As it is not specified in the Data Documentation [11] what this represents we've chosen to remove those 29 rows that have such a value from the data set. They represent less than 1 % of the data set.

3.2 Prediction and evaluation

In order to compare the machine learning models they have been tested on data where the price of the properties are known. Therefore, the data set is split into training data and testing data. Training data is, as the name suggests, used to train the machine learning model and constitutes the larger share of the split data set. When the model is fit with the training data the sales price of the properties are known to the model in order for it to learn from the data. The testing data is used to get a measurement of how well the machine learning model predicts house prices. The machine learning model is given the test data but without the price of the properties in order to predict the price for them given the various features for the properties. The predicted price is then compared to the actual price in the test data. In order to measure the error of a model various error metrics are used as described in section 3.3.

3.2.1 Splitting the data

The data set is used in two ways. First to train the algorithm, and then to test it, and for these intents we have split the set in two. The ratio between the number of rows in the training data and the test data needs to be carefully selected. If the test data is too small the result is less convincing since it is not tested on a large variety of rows. Increasing the test data size improves reliability but reduces the number of rows in the training data which causes the model to predict worse. A way of mitigating the effects of a larger test set is to use cross-validation, which is used for this experiment. [17]

Cross-validation means running the model on the data set multiple times, alternating the part of the data that is used as test data. Using five-fold cross validation is common practice, meaning 20 % of the data set is used as testing data and alternated for five runs. On the first run the first 20 % of the data set is used as testing data and the remaining 80 % is used as training data. On the second run the subsequent 20 % is used as testing data and so on. This method of splitting data is illustrated in figure 3.3.

For each run the error is calculated, and to obtain a single measurement of the error of the model we aggregate the error, using the mean error of the runs. This measurement of error is then used to compare the machine learning models. By using cross-validation all parts of the data will be used as testing data at some point which removes the risk that the particular part of the data set used as test data gives a small error because it happens work well with the model. Since the Ames Housing data set is relatively small cross-validation is an important part in ensuring that a small error for a model is not the effect of the test data working particularly well with the model while another set of testing data might not [17].

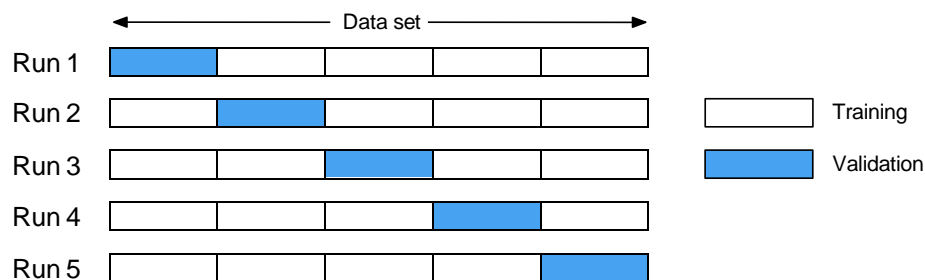


Figure 3.3: Cross-validation.

3.2.2 Algorithm hyperparameters

Both of the k-Nearest neighbour and Random forest algorithms have a variety of hyperparameters directly affecting their functioning and by extension, their predictions. A hyperparameter is a parameter to the algorithm that is set prior to computation, as opposed to parameters that are derived through training. This study has focused on the particular hyperparameters introduced in section 2.1.

Grid search

Multiple hyperparameters have been tested with various values for both of the two algorithms. A question that arose was what values the other hyperparameters should have when the values of one hyperparameter is varied. A method which solves this problem is grid search which exhaustively tests all combinations of hyperparameter values for a set of specified parameters and values. This is in essence a way of verifying that no more optimal parameters can be found. When running grid search and testing different values for the hyperparameters, cross-validation is used on the data set on which the machine learning algorithms run with the given hyperparameters. The errors of the predictions by the algorithms are then compared for different hyperparameter values and the set of hyperparameter values that results in the lowest prediction error is the best set of hyperparameters. These values are used when testing values of the various hyperparameters. [18] [19]

k-Nearest neighbour regression

For the k-Nearest neighbour algorithm the hyperparameter 'k' and 'weights' have been tested in order to investigate which produce predictions with the least error. Recall from section 2.1.1 that the 'k' hyperparameter determines how many neighbours to include in the target value calculation. k-values from 1 to 10 are tested in this study. The 'weights' hyperparameter is either 'distance' or 'uniform'. With the value 'distance' the target values of the neighbours (the sale price) are weighted by distance when calculating the target value for the sample being predicted. The value 'uniform' indicates that the values of the neighbours are not weighted when calculating the target value. Both values have been tested and the prediction error measured as described in section 3.3.

Random forest regression

For the Random forest algorithm the tested hyperparameters are 'n_estimators', 'max_features' and 'criterion', all introduced in section 2.1.2.

The 'n_estimators' hyperparameter determines how many trees to use in the model. We test values up to 50 trees. The 'max_features' controls the number of features to use when building the trees of the Random forest and will be tested with values from 1 to the total number of features in the dataset (221 after cleaning and including one-hot encoded features). The 'criterion' hyperparameter is used to set the function that measures the quality of splits when constructing the trees in the Random forest. Two values are available for this parameter in the library; 'mse' and 'mae', abbreviations for *mean squared error* and *mean absolute error* respectively. Both values have been tested and compared in section 4.3.

Linear Regression

Linear Regression is a supervised machine learning model that attempts to model a linear relationship between dependent variables (Y) and independent variables (X). Every evaluated observation with a model, the target (Y)'s actual value is compared to the target (Y)'s predicted value, and the major differences in these values are called residuals. The Linear Regression model aims to minimize the sum of all squared residuals. Here is the mathematical representation of the linear regression:

$$Y = a_0 + a_1X + \varepsilon$$

The values of X and Y variables are training datasets for the model representation of linear regression. When a user implements a linear regression, algorithms start to find the best fit line using a_0 and a_1 . In such a way, it becomes more accurate to actual data points; since we recognize the value of a_0 and a_1 , we can use a model for predicting the response

3.3 Error metrics

For measuring how good predictions the model makes, four error metrics have been used. Mean absolute error (MAE), Mean squared error (MSE), Median absolute error (MedAE) and Coefficient of determination (R^2). They are all defined below.

3.3.1 Mean absolute error (MAE)

Mean absolute error measures the prediction error by taking the mean of all absolute values of all errors, that is:

$$MAE = \frac{\sum_{i=0}^n |y_i - \hat{y}_i|}{n}$$

Where n is the number of samples, y are the target values and \hat{y} are the predicted values. A MAE closer to 0 means that the model predicts with lower error and that the prediction is better the closer the MAE is to 0. [20]

3.3.2 Mean squared error (MSE)

Mean squared error is similar to MAE, but the impact of a term is quadratically proportional to its size. It measures the prediction error by taking the mean of all squared absolute values of all errors, that is:

$$MSE = \frac{\sum_{i=0}^n (y_i - \hat{y}_i)^2}{n}$$

As for MAE, values close to 0 are better. [21]

3.3.3 Median absolute error (MedAE)

The median absolute error (MedAE) is the median of all absolute differences between the predicted value and the target value. In difference to MAE and MSE, the median absolute error is more robust to outliers by virtue of using the median instead of the mean.

$$MedAE = median(|y_1 - \hat{y}_1| \dots |y_n - \hat{y}_n|)$$

A low MedAE means little error and a good prediction. [22] [23]

3.3.4 Coefficient of determination (R^2)

The coefficient of determination, R^2 , is the ratio between the explained variance and the total variance. Another, perhaps more intuitive way of understanding it is that it is the fraction of the variance that is predictable by (or explained by) by the model. It is frequently used to evaluate how well a regression model fits the data. The coefficient value ranges from 0 to 1 where 1 is better, meaning perfect determination and 0 meaning no determination. In this study, it is calculated using the scikit-learn `r2_score` function [24] as follows [25].

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{samples}-1} (y_i - \bar{y})^2}$$

Where \hat{y}_i is the predicted value of the i th sample, y is the corresponding actual value over $n_{samples}$ and \bar{y} is the arithmetic mean as such:

$$\bar{y} = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} y_i$$

Chapter 4

Results

By following the method stated in the previous section, the following results have been obtained. To begin with, the two methods have been tested with different values for the selected hyperparameters as described in the previous section. The results of those tests are presented in the graphs and tables in section 4.2 and 4.3. The values of the parameters for each method that yields the least prediction error has then been used to compare the two methods and the results of that comparison are presented in section 4.4. All tests are performed on the cleaned data, described in section 3.1.

4.1 Grid search over hyperparameters

The grid search algorithm was used to find the best set of values for the selected hyperparameters of each algorithm, presented in this section. In short, the following was found:

For the k-Nearest neighbour algorithm the best value for the 'k' hyperparameter is 9 neighbors and for the 'weights' hyperparameter the best value was 'distance' (inversely proportional). For the Random forest algorithm the best value for the hyperparameter 'n_estimators' was 41, for 'max_features' 63 and for 'criterion' the best option was 'mse'.

4.2 k-NN hyperparameters

Two hyperparameters have been tested for k-NN with different values. The value of the k number of neighbours to use for the algorithm has been tested for the values 1 through 10 comparing by MAE, displayed in figure 4.1. As

can be seen in the figure, the MAE is least when k is equal to 9 which provides the best prediction with the current setup for the k -NN algorithm when k is ranging between 1 and 10. The value for the 'weights' hyperparameter was set to 'distance' when testing the different values for the k parameter since this value produced the lowest error in the grid search over the hyperparameters.

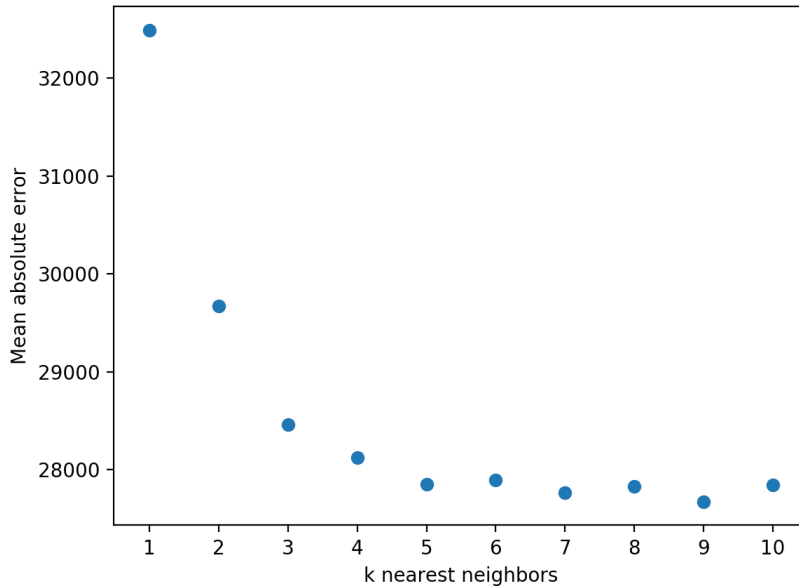


Figure 4.1: MAE for k -NN with values for k from 1 to 10.

Secondly, the hyperparameter 'weights', which determines what weighting function to use for the prediction, is tested with the values 'distance' and 'uniform'. Details of these weighting functions are described in section 2.1.1. The results of running the algorithm with the two values for the 'weights' hyperparameter is presented in table 4.1 where the error metrics presented in section 3.3 are all included. The 'k' hyperparameter was set to 9 when performing these tests since it performed best in the grid search. The distance weight function proves to give better results than the uniform weight function, consistently in all error metrics used.

Weights	MAE	R^2	MedAE	MSE
distance	27670.7	0.695988	17663.9	1896058151.2
uniform	28331.7	0.685283	18425.0	1963543948.5

Table 4.1: Errors for 9-NN with different values for the weight hyperparameter.

4.3 Random forest hyperparameters

The random forest algorithm has been tested with three different hyperparameters; 'n_estimators', 'max_features' and 'criterion'. 'n_estimators', which represents the number of decision trees in the random forest, has been tested with values ranging between 1 and 50 and the results are presented in figure 4.2 where the values are compared by MAE. When testing the 'n_estimators' parameter, the 'criterion' hyperparameter was set to 'mse' and the 'max_features' hyperparameter was set to 63 since those values performed best in the grid search. The 'n_estimators' value that yields the lowest MAE (of 16208.5) is 41.

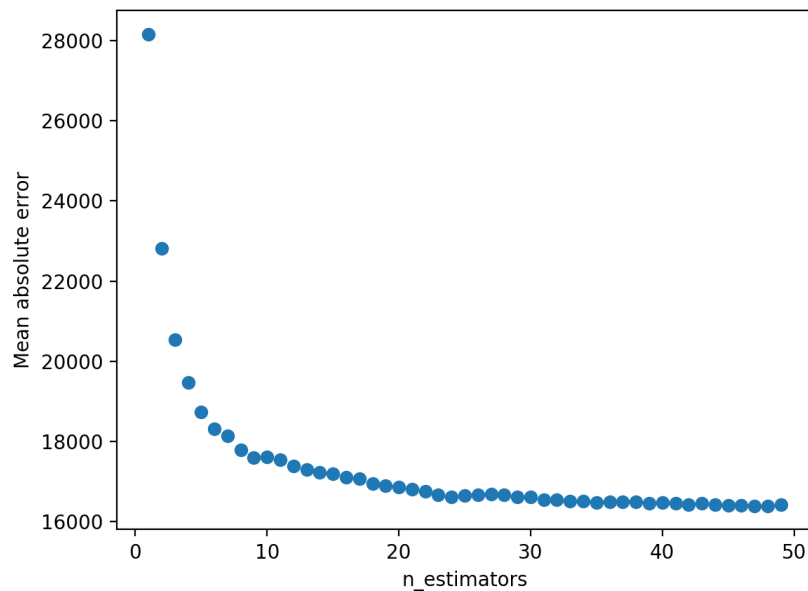


Figure 4.2: MAE for Random forests with n_estimators from 1 to 50.

Secondly, the 'max_features' hyperparameter was tested with values from 1 to 221 which is the total number of features in the cleaned data set (including one-hot encoded features). The results are presented in figure 4.2 where values of 'max_features' are compared by MAE. The value with the least MAE is 63.

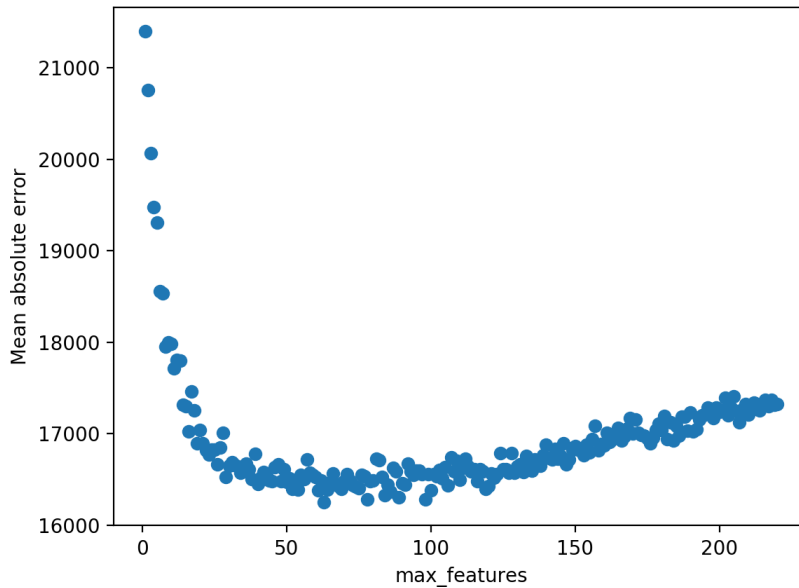


Figure 4.3: MAE for Random forests with `max_features` from 1 to 221.

The third hyperparameter tested for Random forest is the 'criterion' hyperparameter which determines what error measurement to use to measure the quality of a split in the trees. The values tested are 'mae' which uses mean absolute error and 'mse' which uses mean squared error. The hyperparameter 'n_estimators' was set to 41 when performing the tests and 'max_features' was set to 63, values determined from the grid search.

As can be seen in table 4.2, the 'mse' parameter option performs marginally better for all error metrics except MedAE.

Criterion	MAE	R^2	MedAE	MSE
mae	16412.8	0.875408	10121.8	768458634.6
mse	16208.5	0.877811	10135.9	754362031.6

Table 4.2: Errors for Random forests different values for the criterion hyperparameter.

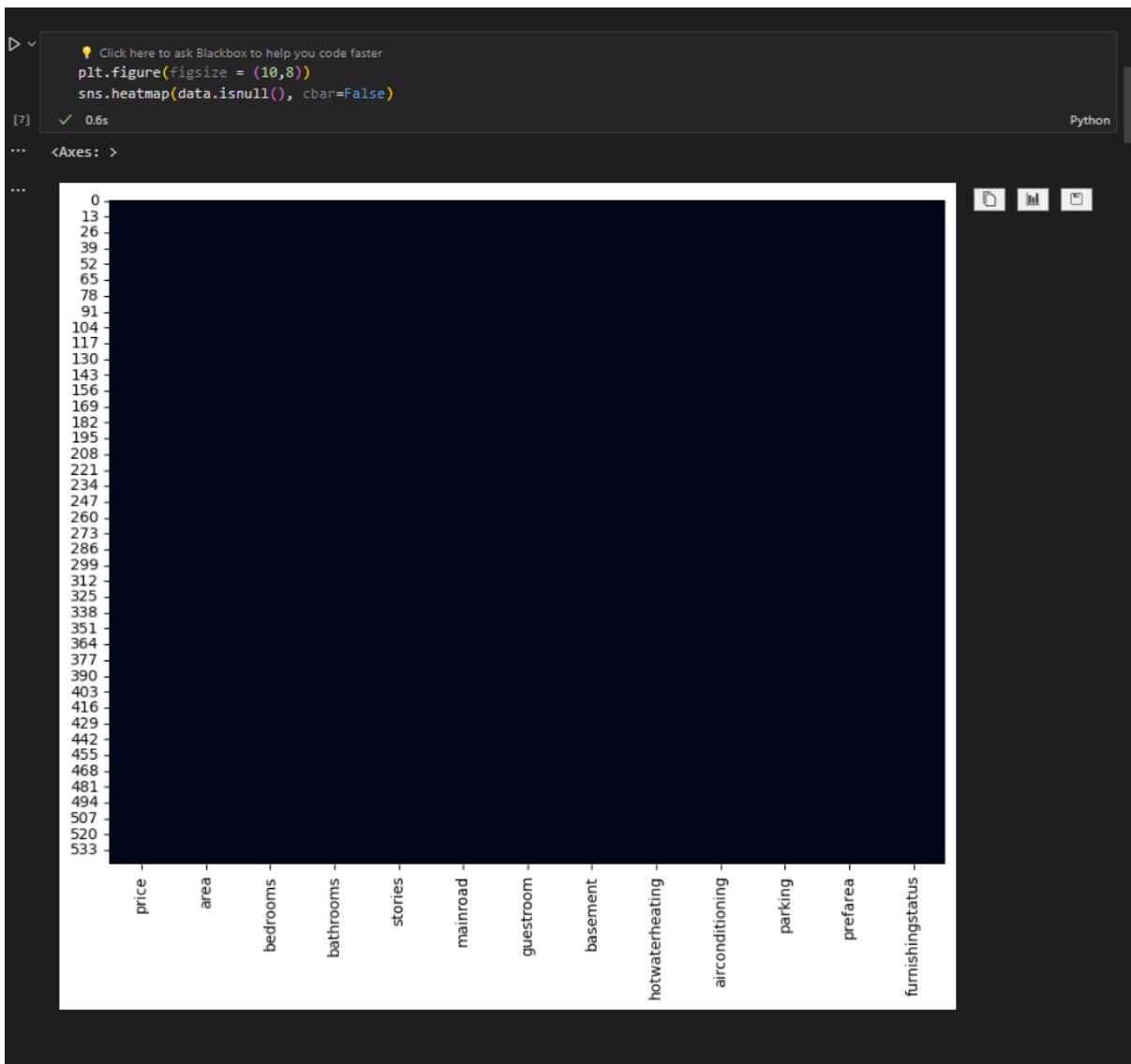
4.4 Algorithm comparison

Finally, errors of the k-Nearest neighbour algorithm and the Random forest algorithm are compared, which gets to the focus of this study. Presented in

table 4.4 are the errors for the respective algorithms running on their selected optimal hyperparameters. On all four error metrics investigated, it is clear that the Random forest algorithm performs considerably better than the k-Nearest neighbour algorithm, with regards to predicting with the smallest error.

Algorithm	Hyperparameters	MAE	R ²	MedAE	MSE
k-Nearest neighbours	k=9, weights=distance	27670.7	0.695988	17663.9	1896058151.2
Random forest	n_estimators=41, max_features=63, criterion=mse	16208.5	0.877811	10135.9	754362031.6

Table 4.3: Errors of the two methods.



Click here to ask Blackbox to help you code faster

data.describe()

[9] ✓ 0.0s Python

...

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

Click here to ask Blackbox to help you code faster

for column in data.columns:
 unique_values = data[column].unique()
 print(f"Unique values for {column}: {unique_values}")

[10] ✓ 0.0s Python

...

Unique values for price: [13300000 12250000 12215000 11410000 10850000 10150000 9870000 9800000
9681000 9310000 9240000 9100000 8960000 8890000 8855000 8750000
8680000 8645000 8575000 8540000 8463000 8400000 8295000 8190000
8120000 8080940 8043000 7980000 7962500 7910000 7875000 7840000
7700000 7560000 7525000 7490000 7455000 7420000 7350000 7343000
7245000 7210000 7140000 7070000 7035000 7000000 6930000 6895000
6860000 6790000 6755000 6720000 6685000 6650000 6629000 6615000
6580000 6510000 6475000 6440000 6419000 6405000 6300000 6293000
6265000 6230000 6195000 6160000 6125000 6107500 6090000 6083000
6020000 5950000 5943000 5880000 5873000 5866000 5810000 5803000
5775000 5740000 5652500 5600000 5565000 5530000 5523000 5495000
5460000 5425000 5390000 5383000 5320000 5285000 5250000 5243000
5229000 5215000 5145000 5110000 5075000 5040000 5033000 5005000
4970000 4956000 4935000 4907000 4900000 4893000 4865000 4830000
4795000 4767000 4760000 4753000 4690000 4655000 4620000 4613000
4585000 4550000 4543000 4515000 4480000 4473000 4445000 4410000
4403000 4382000 4375000 4340000 4319000 4305000 4277000 4270000
4235000 4200000 4193000 4165000 4130000 4123000 4098500 4095000
4060000 4025000 4007500 3990000 3920000 3885000 3850000 3836000
3815000 3780000 3773000 3745000 3710000 3703000 3675000 3640000
3633000 3605000 3570000 3535000 3500000 3493000 3465000 3430000
3423000 3395000 3360000 3353000 3332000 3325000 3290000 3255000
3234000 3220000 3150000 3143000 3129000 3118850 3115000 3087000
3080000 3045000 3010000 3003000 2975000 2961000 2940000 2870000
2852500 2835000 2800000 2730000 2695000 2660000 2653000 2604000
...
Unique values for airconditioning: ['yes' 'no']
Unique values for parking: [2 3 0 1]
Unique values for prefarea: ['yes' 'no']
Unique values for furnishingstatus: ['furnished' 'semi-furnished' 'unfurnished']
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Data Vizualization, Storytelling & Experimenting with charts :

Understand the relationships between variables

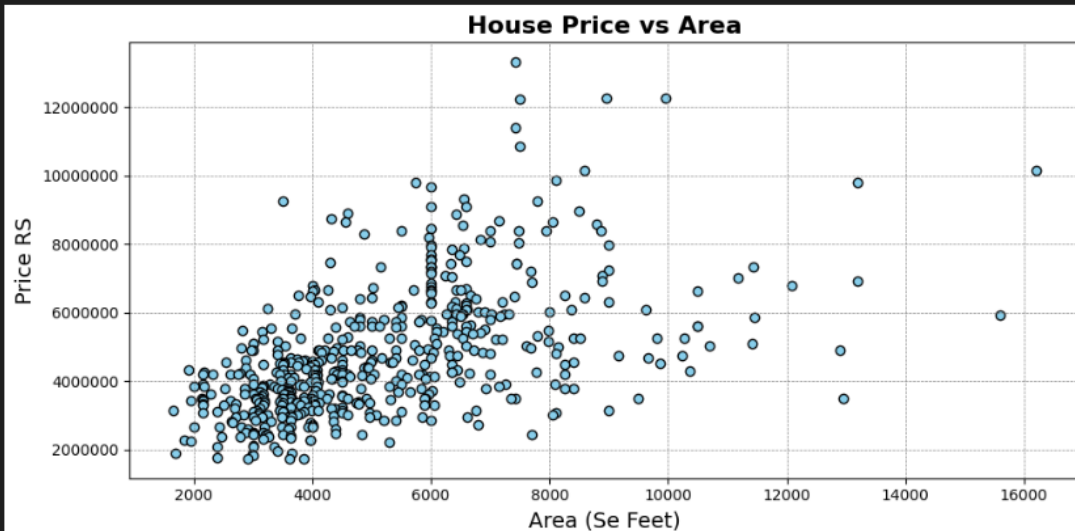
Click here to ask Blackbox to help you code faster

```
plt.figure(figsize=(10, 5))
plt.scatter(data['area'], data['price'], color='skyblue', edgecolor='black')
plt.title('House Price vs Area', fontsize=16, fontweight='bold')
plt.xlabel('Area (Se Feet)', fontsize=14)
plt.ylabel('Price RS', fontsize=14)
plt.grid(True, linestyle='--', linewidth=0.5, color='gray', axis='both')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout()
plt.show()
```

[16]

✓ 0.3s

Python



Click here to ask Blackbox to help you code faster

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='bedrooms', y='price', data=data, palette='Set1')
plt.title('House Price Distribution by Number of Bedrooms', fontsize=18, fontweight='bold')
plt.xlabel('Number of Bedrooms', fontsize=14, fontweight='bold')
plt.ylabel('Price ', fontsize=14, fontweight='bold')
plt.ticklabel_format(style='plain', axis='y')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
plt.show()
```

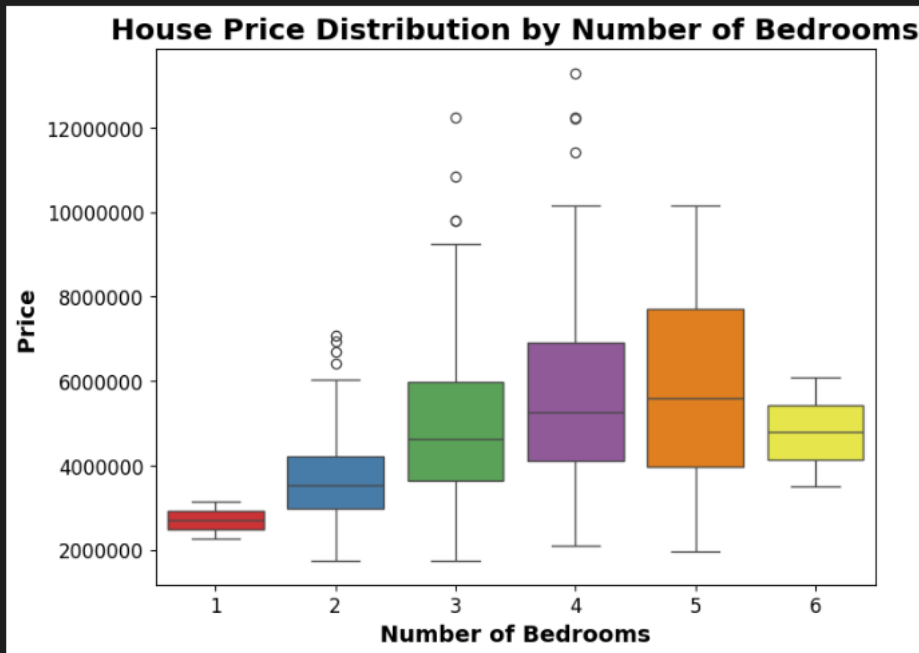
0.3s

Python

C:\Users\335ar\AppData\Local\Temp\ipykernel_26288\1233864766.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

```
sns.boxplot(x='bedrooms', y='price', data=data, palette='Set1')
```



Click here to ask Blackbox to help you code faster

```
# Chart - 3 visualization code
```


Click here to ask Blackbox to help you code faster

```
# Chart - 3 visualization code
import matplotlib.pyplot as plt
```

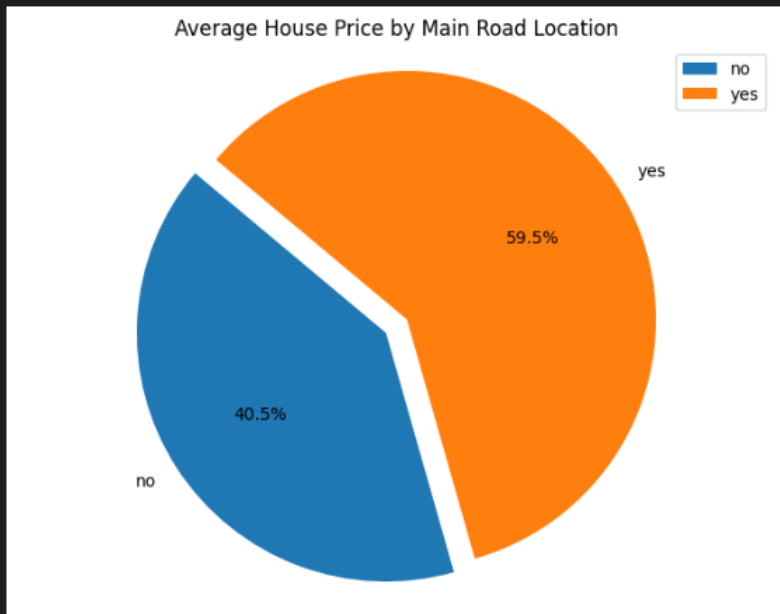
```
# Assuming you already have the 'average_price_mainroad' DataFrame and 'data' DataFrame defined
```

```
average_price_mainroad = data.groupby('mainroad')['price'].mean()
plt.figure(figsize=(8, 6))
explode = (0.1, 0)
plt.pie(average_price_mainroad, labels=average_price_mainroad.index, autopct='%1.1f%%', startangle=140, explode=explode)
plt.title('Average House Price by Main Road Location')
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
```

```
# Add legend
```

```
plt.legend(average_price_mainroad.index, loc="best")
```

```
plt.show()
```



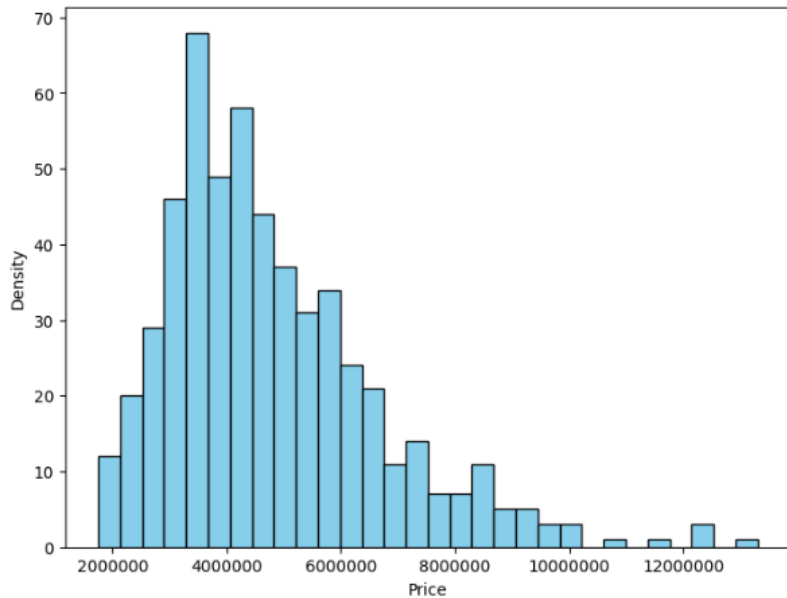
Click here to ask Blackbox to help you code faster

Chart - 4 visualization code

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8, 6))
plt.hist(data['price'], bins=30, color='skyblue', edgecolor='black') # Set density to True for KDE
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Density') # Change ylabel to reflect density
plt.ticklabel_format(style='plain', axis='x')
plt.show()
```

Distribution of House Prices



Chapter 5

Discussion

In this study the two machine learning regression algorithms k-Nearest neighbour and Random forest have been compared when trained and tested with the Ames housing data set. This has been done in order to study how accurately they, as machine learning methods, predict the prices for the house pricing problem.

Our primary finding was that the Random Forest algorithm performed best with regards to all of the error metrics. Both its MAE and MedAE are about half of the respective errors for the k-Nearest neighbour algorithm. Additionally, the R^2 value for Random forest is also better than k-NN which indicates that the Random forest algorithm will make better predictions on future data.

Since a R^2 value close to 1 is better in terms of how well the models fits the data the R^2 value for Random forest is quite good. This is consistent with the results obtained by Baldominos et al. [3] where Random forest performs better than k-NN even though our models performed better in terms of error metrics.

The values of the parameters tested seem to impact how the models perform. For k-NN the optimal value for the 'k' parameter was 9 even though the values between 7-10 gave similar MAE. Low values of 'k' gave a much higher MAE than higher values for 'k'. The two values for the 'weights' parameter did not affect the prediction errors significantly even though the value 'distance' gave lower errors. This is reasonable since the model then weighs the distances to the nearest neighbour and thus avoiding that the value of a sample potentially far away is treated the same as a sample very near. For the Random forest algorithm the 'n_estimators' parameter exhibits a similar results as the 'k' parameter for k-NN; at low values the model performs poorly and for higher values it performs better. However, higher values of 'n_estimators' comes with the cost that the algorithm takes longer time to run. The optimal

value for 'n_estimators' was 41. The 'max_features' parameter had its optimal value at 63 out of 211 features in total. 211 is a quite high number of features and even before pre-processing the data the number of features were 80 which is also quite high. An improvement to the models and to the method used in this study would have been to calculate the feature importance for various features and only use the features that affected the price of houses most. This is particularly relevant for the k-NN algorithm which perform poorly on high-dimensional data [5]. Finally the 'criterion' parameter was tested with two values, 'mae' and 'mse' and there was no significant difference in the error metrics. The model performed marginally better with 'mse' than 'mae'. A few of the available parameters for the algorithms have been tested in this study and if more parameters or other values would have been tested the results could potentially have been improved.

The Ames housing data set used in this study consists of about 3000 rows and is thus a quite small data set. A larger data set could have improved our results by training the models on a larger and potentially less biased data set.

Even though the prediction errors are quite low with the Random forest algorithm a MAE of 16208.5 dollars (9 % of the mean price) is still quite significant and might be too high to use for predictions by for example brokers or investors. This is comparable to the 8-9 % achieved by Oxenstierna for similar methods. Another aspect to consider when using these models is that housing prices usually changes over time and the current market might not correspond to the market data that the models were trained with which would cause inaccurate predictions.

Both Baldominos et al. [3] and Oxenstierna [4] mentions the use of Artificial neural networks which performs quite well for predicting house prices in both studies and thus it could be of interest for future research to study how well Artificial neural networks can predict house prices.

Chapter 6

Conclusions

The research question for this study is to study how well house prices can be predicted by using k-Nearest neighbour and Random forest regression. In this study we have found that the Random forest regression algorithm performs better at predicting house prices than the k-Nearest neighbour algorithm. However, there is still a difference between the actual prices in our testing data and the prices predicted by the Random forest regression algorithm. The lowest error achieved was for the Random forest with an MAE of 16208.5 dollars, about 9 % of the mean price.

