Regression and time estimation in the manufacturing industry

Walter Bjernulf
Abstract

In this thesis an analysis is performed on operation times for different sized products in a manufacturing company. The thesis will introduce and summarise most of the theory needed to perform regression and also cover a worked example where three different regression models are learned, evaluated and analysed. Conformal prediction, which at the moment is a hot topic in machine learning, will also be introduced and will be used in the worked example.

Keywords: regression, linear regression, regression trees, random forests, cross validation, variable selection, conformal prediction
## Contents

1 Introduction  
   1.1 Background  
   1.2 Limitation  

2 Notation  
   2.1 General notation  
   2.2 Typical notation for regression  

3 Theory  
   3.1 Introduction  
   3.2 Basic theory and definitions  
      3.2.1 Data set  
      3.2.2 Response variable  
      3.2.3 Explanatory variables  
      3.2.4 Dummy variables  
      3.2.5 The regression problem  
      3.2.6 Predictions  
      3.2.7 Residual  
      3.2.8 Coefficient of determination  
      3.2.9 Outliers  
      3.2.10 Bootstrapping  
      3.2.11 Overfitting  
   3.3 Linear regression  
      3.3.1 Assumptions  
      3.3.2 The linear regression model  
      3.3.3 Simple linear regression  
      3.3.4 Multiple linear regression  
      3.3.5 Normal equations  
   3.4 Loss functions and cost functions  
      3.4.1 Loss functions  
      3.4.2 The cost function  
   3.5 Regression trees  
      3.5.1 Introduction  
      3.5.2 The splitting points  
   3.6 Ensemble methods  
      3.6.1 The bias-variance trade off  
      3.6.2 Bagging  
      3.6.3 Random Forests  
   3.7 Evaluation  
      3.7.1 The error function  
      3.7.2 Train and test split  
      3.7.3 Cross validation  
   3.8 Transforming variables  
      3.8.1 Polynomial transformations  

2
3.8.2 Power transformations .................................................. 23
3.8.3 Non-linear transformations ............................................. 23
3.9 Variable selection ............................................................ 23
  3.9.1 Forward selection ...................................................... 23
  3.9.2 Backward selection .................................................... 24
  3.9.3 Lasso regularisation ................................................... 24
3.10 Hyperparameter optimisation ............................................ 24
  3.10.1 Grid search .......................................................... 25
  3.10.2 Random search ....................................................... 25
3.11 Prediction intervals ....................................................... 25
  3.11.1 Standard deviation ................................................... 26
  3.11.2 Conformal prediction ................................................ 26

4 Methodology and worked example ........................................ 27
  4.1 The problem .............................................................. 28
  4.2 Cleaning the data set .................................................... 28
  4.3 Exploring, visualizing and transformations .......................... 29
  4.4 Models ................................................................. 31
  4.5 Hyperparameter optimization .......................................... 31
  4.5.1 Optimization results ................................................ 31
  4.6 Comparing the models ................................................... 31
  4.6.1 Visualizing the performance ....................................... 31
  4.6.2 Cross validation ...................................................... 32
  4.7 Variable selection ....................................................... 36
  4.8 Predictions .............................................................. 36
  4.8.1 Point predictions ..................................................... 36
  4.8.2 Prediction intervals with conformal prediction .................. 37

5 Discussion and conclusions ................................................ 39

References .............................................................................. 40

A Decomposition of the MSE ................................................... 41

B Python code ....................................................................... 42
  B.1 Import statements ......................................................... 42
  B.2 Loading the data .......................................................... 42
  B.3 Exploring and visualizing the data ..................................... 42
  B.4 Transforming the variables .............................................. 43
  B.5 Hyperparameter optimization .......................................... 43
  B.6 Visualizing performance on test data ................................ 44
  B.7 Cross validation .......................................................... 45
  B.8 Variable selection ......................................................... 46
  B.9 Point predictions .......................................................... 47
  B.10 Conformal prediction .................................................... 48
1 Introduction

In this thesis an analysis is performed on operation times for different sized products in a manufacturing company. The objective is to develop a new model for estimating the operation times for new products. The analysis is performed on one work center, but the same methodology and ideas that are presented can be followed when developing models for other work centers. The manufacturing company where this analysis is performed, is interested in a linear function which can easily be implemented, therefore the linear regression method will be the main focus in this thesis.

The thesis is supposed to serve both as an introduction to and summary of regression. It is aimed at people with little to no background in mathematics with the intention of introducing them to regression and at people with a mathematical background which are in need of a refresher of the theory or might want to see a worked example. Even though some parts might be more difficult to grasp with no knowledge of mathematics, most of the ideas presented should still be comprehensible.

The aim of this chapter is to cover the background and limitations of the thesis.

1.1 Background

Pricing is an important part in any manufacturing company and being able to offer a price, which will both be profitable for the company and at the same time not making the customer choose a competing manufacturer is crucial.

What price a customer is willing to pay for a product is typically unknown, but even if a price were given, the manufacturer can not know if the price is profitable unless they know all the costs that the company have related to that product. This motivates the manufacturer to have a way of calculating these costs. One part in this calculation is knowing all of the operation times related to the product, which leads us to the objective of the thesis.

1.2 Limitation

Operation times can be analysed in many different ways and result in various different models. The operations which are being analysed are manual operations, meaning that there are many uncontrollable variables affecting the operation times in the data set. For instance, who performs the operation will affect the operation time a lot and all discrepancies, even things which are a part of a normal workday for the operator, will affect the reported operation time more or less. It is important to remember that these unpredictable variables will limit the performance of the model.
2 Notation

This section introduce some useful notation.

2.1 General notation

\( b \) - a scalar

\( \mathbf{b} \) - a vector

\( \mathbf{B} \) - a matrix

\( \mathbf{b}^T \) - transpose of \( \mathbf{b} \)

\( \| \mathbf{b} \|_1 \) - \( L^1 \) norm of \( \mathbf{b} \), \( \| \mathbf{b} \|_1 = |b_1| + \ldots + |b_n| \)

\( \| \mathbf{b} \|_2 \) - \( L^2 \) norm of \( \mathbf{b} \), \( \| \mathbf{b} \|_2 = \sqrt{b_1^2 + \ldots + b_n^2} \)

2.2 Typical notation for regression

\( \mathbf{x} \) - the input

\( \mathbf{y} \) - the output

\( \hat{\mathbf{y}} \) - an estimation of \( \mathbf{y} \)

\( \epsilon \) - irreducible error

\( L \) - loss function

\( J \) - cost function

\( \beta \) - the parameter vector

\( \hat{\beta} \) - an estimation of \( \beta \)
3 Theory

3.1 Introduction

This chapter will cover some basic theory needed for regression and will also introduce linear regression, loss and cost functions, regression trees, bagging, random forests, transformation of variables, variable selection, evaluation of models, cross validation, hyperparameter optimisation and prediction intervals. The references in this chapter are mostly to the book Machine learning - A First Course for Engineers and Scientists [2] (Lindholm et al., 2022) which I got familiar with in the course Statistical Machine Learning. There are also some references to the book Applied Linear Regression [3] (Weisberg, 2014) which I got familiar with in the course Regression Analysis.

3.2 Basic theory and definitions

3.2.1 Data set

A data set is a collection of data and usually consist of several rows where each row represents a given observation. Moreover, the data set usually consist of several columns, where each column represents a given variable.

3.2.2 Response variable

The response variable, or dependent variable, is the variable which we are trying to predict and which depends on the other variables. This variable is unknown when we are making estimations. This could be the reported time of nailing pallets or sawing sheets of plywood.

In regression the response variable needs to be numerical and in classification the response variable is categorical.

3.2.3 Explanatory variables

The explanatory variables, predictors or independent variables, are the variables which the response variable depends on. This could be the length, width and type of a pallet.

Explanatory variables can be numerical or categorical, but if a variable is categorical we need to give them some attention before they can be used to train a model.

3.2.4 Dummy variables

Dummy variables are needed when one or more of the predictors are categorical. Categorical variables could for example be type of pallet or material type. Assume that we have an input variable "type of pallet" which can take 2 different values, "2-way" and "4-way". To be able to use this variable in regression we
need to create a dummy variable, suppose we call it \( x \), which can be defined in the following way

\[
x = \begin{cases} 
0 & \text{if 2-way}, \\
1 & \text{if 4-way}.
\end{cases}
\] (1)

The above dummy variable \( x \) can now be treated as any other numerical variable in regression.

Now suppose that our input variable "material type" can take the values A, B and C. This can be solved by creating a three-dimensional vector \( x \) in the following way

\[
x = [x_A \ x_B \ x_C]^T
\] (2)

where only one entry in \( x \) is 1 and the others are 0. For example if the material type is B, we would get the following vector

\[
x = [0 \ 1 \ 0]^T.
\] (3)

This last method is called one-hot encoding. Both methods can easily be extended to deal with cases where the categorical variable can take multiple values.

### 3.2.5 The regression problem

The regression problem is about learning a model \( f \), such that

\[
y = f(x) + \epsilon,
\] (4)

where \( y \) is the response variable, \( x \) are the explanatory variables and \( \epsilon \) is the irreducible error term, which is considered to be a random variable (Lindholm et al., 2022, p. 37). In figure 1 there is an example showing some observed \( y \)-values and the function \( f(x) \). Note that when faced with a real-world regression problem the function \( f \) is unknown. Therefore the goal is to approximate the function \( f \) by trying to learn the input-output relationship between the response variable \( y \) and the explanatory variables \( x \).

![Figure 1: A graph showing an example of some observed \( y \)-values and the function \( f(x) \).](image)
3.2.6 Predictions

One of the primary reasons why we are interested in the input-output relationship is because we want to be able to make predictions for new, unseen data (Lindholm et al., 2022, p. 18). We make predictions by using a mathematical method which have generalised the relationship between the response variable and the explanatory variables.

3.2.7 Residual

The difference between the observed value $y_i$ and the predicted value $\hat{y}_i$ is called a residual $r_i$, where $i$ denotes the $i$th observation,

$$ r_i = y_i - \hat{y}_i. \tag{5} $$

3.2.8 Coefficient of determination

The coefficient of determination, $R^2$, is a scale-free one-number summary of the strength of the relationship between the response variable and the explanatory variables (Weisberg, 2014, p. 36).

Suppose that we are given a data set consisting of $n$ observations, the response variable vector $y$ and the explanatory variable vector $x$. If we would ignore $x$, a good prediction for a new response $y$ would be to just compute the average of $y$, which we denote as $\bar{y}$. Assume that we use $\bar{y}$ as our model, which would correspond to predicting $\bar{y}$ for every observation. In this case the residual sum of squares would be the same as the observed total variation of the response, which for this section we will denote as $SYY$,

$$ SYY = \sum_i^n (y_i - \bar{y})^2. \tag{6} $$

If instead we would use the explanatory variable $x$ when training the model, we would get different (and hopefully better) predictions and the residual sum of squares would be as follows

$$ RSS = \sum_i^n (y_i - \hat{y}_i)^2. \tag{7} $$

Given the above sums we can now compute the coefficient of determination

$$ R^2 = \frac{SYY - RSS}{SYY}. \tag{8} $$

If there is no relationship between the response variable and the explanatory variables, $RSS$ should be close to $SYY$ and we would therefore get $R^2 \approx 0$. Moreover, suppose there is a strong relationship instead, in this case $RSS$ will be small which would make $R^2 \approx 1$. Note that $R^2$ will never be $> 1$ and that $R^2 = 1$ would suggest a perfect relationship between the response and the explanatory variables. It is possible to have $R^2 < 0$ if the predictions are worse than the average.
3.2.9 Outliers

In some data sets, it is possible that some of the observations are not following the fitted model in the same way as the rest of the data, these data points are called outliers and identifying these might be a good idea, since a single outlier can change the trained model by a lot (Weisberg, 2014, p. 214). An example of some observed $y$-values including a potential outlier is shown in figure 2. If we only have one explanatory variable, outliers can usually be easily identifiable by visualizing the data in a scatter plot.

The Cook’s distance $D_i$ can be used to analyse the data and find potential outliers, which essentially is defined as the sum of changes the regression model make when the $i$th data point is removed.

$$D_i = \frac{\sum_{j=1}^{n}(\hat{y}_j - \hat{y}_{j(i)})^2}{p\sigma^2}$$

where $\hat{y}_{j(i)}$ is the model trained without the $i$th data point, $p$ is the number of explanatory variables and $\sigma^2$ is the estimated variance.

Locating potential outliers can be done with the help of Cook’s distance, but it does not state anything about what should be done with the data point. Usually, the potential outliers are analysed and if we suspect some kind of error in the measuring process, the points are deleted. If we do not find anything surprising about the points, they can either be kept in the data set or we could train two different models, one with and another without the potential outliers, analyse the differences and evaluate how the two models generalize.

![Figure 2: A graph showing an example of some observed $y$-values including a potential outlier.](image)

3.2.10 Bootstrapping

Bootstrapping is a sampling method, where we use random sampling with replacement to produce new, slightly different versions of the given data set. The idea is that sampling with replacement is sampling from the empirical distribution, which is an estimate of the distribution which generated the observations.
Sampling with replacement means that the new data set may contain more than one copy of the same data point from the original data set.

### 3.2.11 Overfitting

Overfitting is when the model fits the training data too much, which leads to poor generalization. Since the regression problem is about learning a model \( f \), such that

\[
y = f(x) + \epsilon,
\]

where \( \epsilon \) is a random variable and \( f \) is unknown and can only be approximated, the regression model should not be trained with the purpose of making zero error on the training data. Since we want the regression model to approximate \( f \) and not \( f + \epsilon \), we should expect an irreducible error \( |\epsilon| > 0 \). Note that \( \epsilon \) is a random variable and that the irreducible error can be small for some points and large for others. The more flexible a model is, the higher the risk is for the model to fit too much to the training data. Figure 3 is showing two models fitted to some training data, where the more flexible model suffers from overfitting. Note that even though the more flexible model fits the training data perfectly, it would give very poor results when evaluated on new unseen data.

![Figure 3: A graph showing two models fitted to some training data, where the more flexible model clearly suffers from overfitting.](image)

### 3.3 Linear regression

The linear regression model might be the most popular method for solving regression problems, at least historically (Lindholm et al., 2022, p. 37). This section will first cover the assumptions that typically are assumed when learning a linear regression model. Thereafter it will introduce the linear regression model, briefly cover simple linear regression and lastly multiple linear regression will be covered.
3.3.1 Assumptions

Below are the most common assumptions related to linear regression. If some of these are violated, we need to take it into account when evaluating or analysing the trained models.

- There exists a linear relationship between the explanatory variables and the response variable. Note that this includes transformed input variables.

- There is no multicollinearity between the explanatory variables. If there exists a linear relationship between the explanatory variables we have multicollinearity, which violates this assumption. This assumption is really important if we want to make any statements about the relationship between some of the explanatory variables and the response. If we are only interested in creating a model that make good predictions, and less interested in how each of the explanatory variables affect the dependent variable, this assumption can typically be dropped.

- The error terms have mean zero and they are independent (Weisberg, 2014, p. 21-22). Independence means that having information about one error term would give no information about another error term.

- The error terms follow a normal distribution. This assumption is typically needed if we want to make tests and confidence statements, but if we are only interested in producing a model with the primary purpose of making predictions, this assumption is usually stronger than we need (Weisberg, 2014, p. 22).

- The error terms are homoskedastic. If the error terms in the regression model have the same finite variance they are homoskedastic, but if the variance of the error terms varies for different inputs we have heteroskedasticity. When this assumption does not hold, we can not make any strong conclusions between the explanatory variables and the response, as the statistical tests assumes a constant variance.

3.3.2 The linear regression model

As stated in 3.2.5, the regression problem is about approximating the unknown function $f$, which describes the true relationship between the response and the explanatory variables.

In linear regression we assume that there exist a linear relationship between the response variable $y$ and the $p$ explanatory variables $x_1, x_2, ..., x_p$. Therefore, the model takes the following form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon.$$  \hfill (11)
3.3.3 Simple linear regression

In simple linear regression we only have one explanatory variable and we typically assume that all of the assumptions stated in section 3.3.1 hold. Therefore we can write the simple linear regression model in the following way

\[ y = \beta_0 + \beta_1 x + \epsilon. \]  

The parameter \( \beta_0 \) is usually called the intercept or offset and \( \beta_1 \) is usually called the slope.

For simple linear regression, the calculations needed to find the parameters \( \beta_0 \) and \( \beta_1 \) which minimize the residual sum of squares (RSS), only depend on averages of the variables, sums of squares and sums of cross products (Weisberg, 2014, p. 23). Another way of finding the parameter estimates that minimizes the RSS is by using the normal equations, which will be covered in section 3.3.5.

3.3.4 Multiple linear regression

In multiple linear regression we have more than one explanatory variable and we typically assume less in comparison to simple linear regression. We always assume that there exists a linear relationship between the response variable and the explanatory variables, but depending on the regression problem we can allow some of the assumptions in section 3.3.1 to be violated. To get more compact notation, we define the parameter vector \( \beta \)

\[ \beta = [\beta_0 \ \beta_1 \ldots \beta_p]^T. \]

We also define the vector \( x \) to be

\[ x = [1 \ x_1 \ x_2 \ldots x_p]^T, \]

where we add a constant 1 additionally to the explanatory variables. Note by doing this, now both vectors have \( p + 1 \) entries, one for the intercept and one for each explanatory variable. We can now rewrite the linear regression model in the following way

\[ y = \beta_0 \cdot 1 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon \]
\[ = \beta^T x + \epsilon. \]  

Suppose now that we have a data set consisting of \( n \) observations, the inputs \( x_i \), and the outputs \( y_i, i = 1, \ldots, n \). We define the matrix \( X \)

\[ X = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix} \]
and the vectors $y$ and $\epsilon$ in the following way

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \quad (17)$$

We can now express the linear regression model for all $n$ data points,

$$y = X\beta + \epsilon \quad (18)$$

Moreover, if we define the vector for the predicted outputs as $\hat{y} = [\hat{y}_1 \ \hat{y}_2 \ \ldots \ \hat{y}_n]^T$, we get the following equation

$$\hat{y} = X\beta \quad (19)$$

### 3.3.5 Normal equations

If we use the squared error loss to learn a linear regression model, i.e. if we want to find $\hat{\beta}$ such that we minimize the residual sum of squares, we need to solve the following problem (Lindholm et al., 2022, p.41)

$$\hat{\beta} = \arg\min_\beta \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = \arg\min_\beta \frac{1}{n} \|X\beta - y\|^2_2 \quad (20)$$

A solution to this problem can be found by using linear algebra. We want to find the vector that is closest to $y$, allowing ourselves only to look in the subspace that is spanned by the columns of $X$. The solution is the orthogonal projection of $y$ into this subspace (Lindholm et al., 2022, p.41),

$$X^TX\hat{\beta} = X^Ty \quad (21)$$

The above matrix equation is called the normal equations and given that $X$ have full rank, such that the matrix $X^TX$ is invertible, $\hat{\beta}$ is the unique parameter vector that minimizes the RSS. Here $X$ having full rank means that there is at least as many linearly independent rows as there are columns in the matrix. Given that $X^TX$ is invertible we get the following expression for $\hat{\beta}$,

$$\hat{\beta} = (X^TX)^{-1}X^Ty \quad (22)$$

### 3.4 Loss functions and cost functions

In the earlier sections we have mentioned the squared error loss and talked about minimizing the residual sum of squares. A loss function $L(y, \hat{y})$ is a function which gives a value to how close the estimation is from the observed response, typically assigning lower values to close estimations and larger values to worse estimations. The squared error loss is typically the go to loss function when learning regression models, and especially linear regression models. The reason
for this is that we often want to penalize large errors more and small errors less and for linear regression we also get the closed form solutions for $\hat{\beta}$ in the form of the normal equations.

There are many loss functions to choose from when training a model and which one to choose can be a difficult task since there is usually no right or wrong loss function for a given problem (Lindholm et al., 2022, p.97). It is important to remember that the chosen loss function will give $\hat{\beta}$ different characteristics. The difference between different loss functions is typically the robustness, which is strongly connected to outliers. Outliers are data points which does not describe the true input-output relationship, see section 3.2.9 A loss function is called robust if outliers have a minor impact on the fitted model. When the data set consist of more outliers, it becomes more important to choose a robust loss function and vice versa. The following sections will cover some of the most common loss functions.

### 3.4.1 Loss functions

The squared error loss is the most common loss function and the default choice for linear regression

$$L(y, \hat{y}) = (\hat{y} - y)^2$$

The squared error loss grows fast and penalizes large errors by a lot, whereas small errors does not get penalized by much. This will lead to outliers affecting the trained model by a lot and therefore the squared error loss is typically not said to be robust.

The absolute error loss is another common loss function

$$L(y, \hat{y}) = |\hat{y} - y|$$

With the absolute error loss, large errors does not grow as fast in comparison to the squared error loss. Therefore the absolute error loss is more robust, compared to the squared error loss. One disadvantage with the absolute error loss is that small errors gets penalized more than desired, in comparison to the squared error loss which penalizes small errors $\epsilon < 1$ less.

A natural creation, considering both the advantages and disadvantages of the squared and the absolute error loss, is the Huber loss which is a combination of the squared and absolute error loss functions.

$$L(y, \hat{y}) = \begin{cases} 
\frac{1}{2}(\hat{y} - y)^2 & \text{if } |\hat{y} - y| \leq \epsilon, \\
\epsilon(|\hat{y} - y| - \frac{1}{2}\epsilon) & \text{otherwise} 
\end{cases}$$

where the value of $\epsilon$ control the robustness.

The behaviour of the three loss functions are visualized in figure [4](#).
3.4.2 The cost function

Given a chosen loss function, the goal when training a regression model is to find which $\beta$ that minimize the average loss over all $n$ training data points. This can be summarised in the following expression

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i).$$  \hspace{1cm} (26)

Note that $\hat{y}_i$ is a function of $\beta$. The average loss is called the cost function and is usually denoted as $J$ (Lindholm et al., 2022, p. 41).

3.5 Regression trees

3.5.1 Introduction

Regression trees is a method in which regions are created in the space spanned by the explanatory variables. The creation of the regions are based on rules with the intention of minimizing the loss. Once the regions have been created, there will be a number of responses $y$ in each region, such that the mean value for each region can be calculated. To make a prediction for a new observation $x$, we just look at which region $x$ belongs to and predict $\hat{y}$ to be the corresponding regions mean value.
3.5.2 The splitting points

Binary trees are used to define the regions, and also help with making the predictions. Suppose that we have two explanatory variables, $x_1$ and $x_2$, the first splitting point $x_1 < 2$ and the second splitting point $x_2 < 3$, for the points that fulfil the first condition. In figures 5 and 6, both the corresponding binary tree and the three regions are visualized.

How to decide the splitting points and the shape of the tree, in a way that
the predictions match the data, will now be covered. First, note that the computational power needed to cover all possible splitting points and tree shapes is very large. Therefore, searching through all of the potential shapes is not a reasonable procedure. Instead, a method called recursive binary splitting is used. This method is greedy, which means that it does not look into the future when deciding the next splitting point and will instead look at the current shape and decide the new splitting point based on what is best at the given moment.

Now follows a step in the process of deciding a splitting point. Suppose we have a data set with \( p \) explanatory variables, \( x_1, \ldots, x_p \). We want to pick the \( j \)th explanatory variable and a value for the cut point \( s \), which will split the data set into two regions \( R_1 \) and \( R_2 \) in a way that minimizes the average loss. We define \( R_1 \) and \( R_2 \),

\[
R_1(j, s) = \{x \mid x_j < s\} \quad \text{and} \quad R_2(j, s) = \{x \mid x_j \geq s\}.
\]

(27)

To be able to write the cost function, which we want to minimize, we define \( \hat{y}_1 \) and \( \hat{y}_2 \),

\[
\hat{y}_1(j, s) = \text{Average}\{y_i \mid x_i \in R_1(j, s)\} \quad \text{and} \quad \hat{y}_2(j, s) = \text{Average}\{y_i \mid x_i \in R_2(j, s)\}
\]

(28)

The cost function can be written in the following way

\[
J(j, s) = \sum_{i \mid x_i \in R_1(j, s)} L(y_i, \hat{y}_1(j, s)) + \sum_{i \mid x_i \in R_2(j, s)} L(y_i, \hat{y}_2(j, s)).
\]

(29)

Now, to find the optimal split, we just loop through \( j = 1, \ldots, p \). For each \( j \), there is a finite number of cut points, since we have a finite number of data points. For each pair, \((j, s)\) we compute the cost function and then we just pick the pair with the lowest average loss (Lindholm et al., 2022, p. 29). To find the next optimal split point, we repeat the same procedure for the left and the right branch independently. This continues until we reach a stopping condition.

### 3.6 Ensemble methods

An ensemble method uses an ensemble of base regression models to produce one strong model. This section will first introduce the bias-variance trade off, which is one of the reasons to use ensemble methods, before we will cover bagging and random forests.

#### 3.6.1 The bias-variance trade off

The mean squared error (MSE) can be used to describe how a model generalizes and can be decomposed into the squared bias, the variance and the irreducible error. The models bias can be described as the error from the models simplifying assumptions and the models variance is related to the sensitivity to small changes in the training data. Let \( y = f(x) + \epsilon \), where \( f \) describes the true relationship between \( y \) and \( x \) and \( \epsilon \) is the irreducible error term with mean zero and
Assume that we have approximated \( f \) with the function \( \hat{f} \), where \( \hat{f} \) is considered to be random since it has been approximated using random data. We can then define the \( MSE \) on a new random observation \((x, y)\) in the following way

\[
MSE = E[(y - \hat{f}(x))^2] \\
= E[y^2] - 2E[y\hat{f}(x)] + E[\hat{f}(x)^2] \\
= ... \\
= (f(x) - E[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2 \\
= Bias[\hat{f}(x)]^2 + Var[\hat{f}(x)] + \sigma^2
\]

A full derivation of equation (30) can be found in appendix A.

Typically, a regression model is more or less flexible. A model that is flexible is able to describe complex relationships between the dependent variable and the explanatory variables. The more flexible a model is, the more it will fit to the training data, which in some cases can lead to overfitting. Flexible models will adapt more to the training data, compared to less flexible models, and will therefore have a higher variance. Since the goal in regression is to make predictions for new unseen data points, we need the model to generalize well.

Suppose that we are given two data sets with observations drawn from the same population. If we would fit a deep regression tree (high flexibility) on the two data sets separately, we would get two fitted models. If we then compare the fitted models we expect some differences between the two models, since the deep regression tree will fit to the training data very well. If we would do the same with a shallow regression tree (low flexibility), we do not expect to notice any large differences. If a fitted model changes a lot, depending on the training data, we say that the model have a high variance. A flexible model typically have low bias and high variance, whereas a not so flexible model have high bias and low variance, this is visualized in figure 7.

An important thing to remember is that we can create a very flexible model, with high variance and low bias, that will have zero error on the training data, which in reality does not say anything about how it generalizes. We can also create a model that is not very flexible, with high bias and low variance, but with a poor ability to fit to the training data. The goal is therefore to find something in between, a model that both have low bias and low variance. Typically, when choosing a more flexible model to get a lower bias, we do it at a cost of higher variance and vice versa, this is called the bias-variance trade off.
3.6.2 Bagging

The base models in bagging are typically highly flexible models, often deeply grown regression trees, which have high variance and low bias. The reason for this is because bagging lowers the variance without increasing the bias (Lindholm et al., 2022, p. 164).

Suppose that \( B \) data sets are bootstrapped from the original training data and for each data set, a deeply grown regression tree is fitted. The strong model that is created using bagging, is simply the average of the ensemble of base models that is trained on the \( B \) bootstrapped data sets. A prediction of the response of \( x \) can be described mathematically as

\[
\hat{y}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{y}^{(b)}(x)
\]  

where \( \hat{y}_{bag} \) is the bagged prediction and \( \hat{y}^{(b)} \) is the prediction from the regression tree trained on the \( b \)th bootstrapped data set.

3.6.3 Random Forests

Random forests is a method which can be described almost in the same way as bagging but with a modification, which reduces the variance even further. In random forests, the base models are regression trees, which is not the case in bagging where we can use any regression model as the base model. Moreover the idea is to add some randomness into the training of the regression trees which reduces the correlation between them even further (Lindholm et al., 2022, p. 171).

Suppose that we have a data set with \( p \) explanatory variables \( x_1, \ldots, x_p \) and that we want to train a regression tree. Normally, when deciding the next splitting point in a regression tree, we consider all \( p \) input variables. In random
forests, we instead only consider $q$ out of the $p$ inputs for each split, where $q \leq p$. Which $q$ inputs to consider is randomly chosen for each split. For regression problems $q$ is usually set to $p/3$ but it is usually a better choice to use cross-validation to decide which $q$ minimizes the expected new error (Lindholm et al., 2022, p. 173).

To understand why this would create a better model, recall the recursive binary splitting. Recursive binary splitting was the method which decided the next splitting point in the creation of the binary tree. The greediness in this method leaves the possibility of making early splits which might be unfavourable and suboptimal in the long run (Lindholm et al., 2022, p. 172). By adding some randomness into the deciding of splits, we leave the possibility of creating trees that are more optimal. For instance in bagging, if one of the $p$ explanatory variables are usually chosen when making the first split, the trees will get a similar structure. By adding the randomness in random forests, some trees will be trained in a different way which lowers the correlation between the ensemble members and thus reduces the variance even further.

### 3.7 Evaluation

Since there exist many different regression methods, where each method have lots of hyperparameters, the amount of potential models for any given problem is huge. Therefore, a method for evaluating how different models generalize is needed. Cross validation is a method which can be used for this. Below we will first introduce the error function and the train and test split before explaining the cross validation procedure.

#### 3.7.1 The error function

The error function have many similarities with the loss function. The difference is that the loss function is used in the training of a model, for estimating the elements of the parameter vector, whereas the error function is used in the evaluation of a model. The most common error function is the squared error

$$E(y, \hat{y}) = (\hat{y} - y)^2, \quad (32)$$

but just as we can choose different loss function, we are able to choose different error functions.

The most important reason, for why we need to introduce the error function, is because the loss and the error function does not have to be the same for a given problem. For instance, the absolute error can be used for training a model and the squared error can be used for evaluating a model (Lindholm et al., 2022, p. 63).

#### 3.7.2 Train and test split

When faced with a regression problem, we usually have access to a data set consisting of $n$ observations. Since it might both cost money and take a lot of
time to gather new data, these $n$ observations might be the only observations available to solve the regression problem. As mentioned earlier, the goal of the regression model is not to minimize training data, instead we want the regression model to generalize well on new observations. Since new data might not be available we therefore need to split the accessible data set into two parts.

The train and test split divides the data set into a train set and a test set, where the train set is used when training the model and the test set is used when evaluating the model. This is a good method for acquiring an expectation for the generalized performance of the trained model, since we evaluate it on new data from the models point of view. But what if we are lucky or unlucky when making the splits. Is it really a good idea to let only a part of the whole data set evaluate the performance of the model? This is why we want to use cross validation, which can be seen as an extension to the train and test split.

3.7.3 Cross validation

Cross validation is a method which let us use the whole data set for evaluation of the model, while still only evaluating on new data from the models point of view. To perform cross validation, we start by splitting the data set into $k$ parts, which we label $1, \ldots, k$. We then let the first part be the test set and the other $k - 1$ parts be the train set. The model is trained using the train set before being evaluated on the part which we set aside as the test set. The procedure is then repeated until all of the $k$ parts have been the test set, and the model evaluation is simply the average of the $k$ evaluations. This procedure is also called $k$-fold cross validation (Lindholm, et al., 2022, p. 69). Typically $k$ is chosen to be either 5 or 10, but we can choose $k$ to be larger or smaller, mostly depending on how long we want the the cross validation to run and how precise we want our results to be.

3.8 Transforming variables

Transformation of variables is a useful tool and can be explored if the given explanatory variables can not fully describe the input-output relationship or if we suspect that a transformed variable might describe the input-output relationship in a better way. Since there might exist relationships which we do not know about, transforming variables can also be used to find them. Transformations are non-linear functions and can be used on both the explanatory variables and the response.

A difficult question to answer, especially when the data set consist of many explanatory variables, is what transformations should be tried. Scatter plots with the explanatory variable and the response are usually made, which can then be analysed. In a two-dimensional scatter plot we can easily see if there is for example a linear or quadratic relationship. The next sections will cover some common transformations.

An example where the explanatory variable $x$ should be transformed into $x^2$ can be found in figure \[\text{S}\] where there seems to be a squared relationship between
$x$ and $y$. In figure 9 the same observations are plotted against the transformed variable $x^2$.

![Figure 8: A graph visualizing a squared relationship between $x$ and $y$.](image1)

![Figure 9: A graph visualizing a linear relationship between the transformed variable $x^2$ and $y$. Note that the observations are the same as in figure 8.](image2)

3.8.1 Polynomial transformations

Suppose that we are given a data set with a single explanatory variable $x$ and the response $y$. If we would learn a simple linear regression model for this data set we would want to estimate the parameters $\beta_0$ and $\beta_1$ in the following model

$$y = \beta_0 + \beta_1 x + \epsilon. \quad (33)$$

We could also easily raise $x$ to different powers and see each power as an explanatory variable, creating the following linear regression model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \epsilon. \quad (34)$$
The linear regression model as seen in equation 34 is called polynomial regression.

### 3.8.2 Power transformations

The power family is a collection of transformations which include the square root, cube root, the inverse and the log transformations and is defined only for strictly positive variables (Weisberg, 2014, p. 188). A power transformation $\Psi$ is basically raising the variable $u$ to the power $\lambda$,

$$\Psi(u, \lambda) = u^\lambda$$

where $\lambda = 0$ is defined as the log transform (Weisberg, 2014, p. 188).

The log transform, which belongs to the power family, is very useful and should be considered especially when the values of a variable range over more than one order of magnitude (Weisberg, 2014, p. 188).

### 3.8.3 Non-linear transformations

Any non-linear function can be used to make transformations, including functions which use multiple explanatory variables. Suppose we have the explanatory variables $x_1$ and $x_2$, the following non-linear function is an example of a possible transformation and potential new variable,

$$x_3 = \sqrt{x_1^2 + x_2^2}.$$  

### 3.9 Variable selection

If the purpose of training a regression model is for it to be used in practice, one criteria is that the model should not contain any explanatory variables that are not needed to give an acceptable estimation. The primary reason for this can be that it both cost money and time to collect the variables which in the end might not be needed to make a prediction. Another important reason is that the more variables that are included in the model, the more difficult it might be to use or interpret, which can be more or less important, depending on the problem and the purpose of the model. In this section we will cover three methods which are used for variable selection.

#### 3.9.1 Forward selection

Forward selection is a greedy method for deciding which explanatory variables to use in a model. Suppose that we have $p$ explanatory variables and we want to decide which to be used. Forward selection typically starts with zero explanatory variables and will identify which of the $p$ variables, when added to the current model, will contribute most to the performance. The identified variable will be added to the current model, before the process is repeated with the $p-1$ remaining variables. The process is is run iteratively until a stopping condition
is met, which might be that we have added \( q \) variables to the model or that the performance have reached a threshold.

### 3.9.2 Backward selection

Backward selection is a method that works almost in the same way as forward selection, but backwards. It is a greedy method for deciding which explanatory variables to keep in the final model. Suppose that we have \( p \) explanatory variables in a model and we want to remove the ones that we do not need. Backward selection will identify which of the \( p \) variables that contribute the least to the performance of the model. The identified variable will be removed and we will have \( p - 1 \) variables left. The process is then repeated until a stopping condition is met, which might be that we have \( q \) variables left or that the model performance have dropped below a certain threshold.

### 3.9.3 Lasso regularisation

Regularisation is a method which typically is used to prevent overfitting when training highly flexible models, by adding a penalty term to the cost function. Regularisation typically penalizes large values in the parameter vector. In lasso regularisation (Least Absolute Shrinkage and Selection Operator) the added penalty term is the \( L^1 \) norm of the parameter vector (Lindholm et al., 2022, p. 110). The cost function using lasso regularisation becomes

\[
\hat{\beta} = \arg \min_{\beta} J(\beta) + \lambda \|\beta\|_1,
\]

where \( \lambda \) is called the regularisation parameter, which should be tuned and optimized for the given problem. If we would use the squared error loss, which is normally used in linear regression, we would get the following cost function

\[
\hat{\beta} = \arg \min_{\beta} \frac{1}{n} \|X\beta - y\|_2^2 + \lambda \|\beta\|_1.
\]

The reason for why lasso regularisation can be used for variable selection is because it will give a sparse solution to the parameter vector \( \hat{\beta} \). This means that most of the elements in \( \hat{\beta} \) will be zero and we can simply choose the variables corresponding to the non-zero elements to be included in the final model (Lindholm et al., 2022, p. 110).

### 3.10 Hyperparameter optimisation

When solving a regression problem, apart from training the parameters of a model, we also want to find the most optimal hyperparameters for the model. Examples of hyperparameters can be the value of \( \lambda \) in lasso regression, how deep we should let the binary trees be grown when using regression trees or how many variables should be considered when deciding the splitting points in random forests. A model usually have more than one hyperparameter and some of them can affect others, therefore we need a way of optimizing all parameters at once. This can be performed with grid search or random search.
3.10.1 Grid search

Grid search is a method which can be used to optimize hyperparameters. It is in fact a simple method, since what it is basically doing is just to train and evaluate the model with different hyperparameter combinations. Once this is done, it is not a difficult task to pick the combination that works best. What combinations to try is given by the analyst, and should cover the reasonable possible values for each of the hyperparameters that are optimized. For instance if we would state 10 different values for each of two hyperparameters, we would create a two dimensional grid with 100 grid points, and thus 100 evaluations needs to be performed in the grid search.

This method works well as long as we do not have too many hyperparameters or state too many values for each hyperparameter. Suppose that we want to optimize 5 hyperparameters, where we state 10 values for each of the hyperparameters. This would lead to $10^5 = 100000$ evaluations, which will probably not be calculated in a reasonable amount of time (Lindholm et al., 2022, p. 129).

In the case when the grid search will lead to too many evaluations, one can try to lower the amount of grid points per hyperparameter and then perform the grid search one time. Look at the result and then make a new grid with new grid points close to the result of the first grid search. Another solution to this problem is to use random search instead.

3.10.2 Random search

Random search works almost in the same way as grid search, but instead of evaluating all of the grid points, only a subset of them will be evaluated. Which grid points that are chosen are picked randomly. How many combinations that will be evaluated is chosen by the analyst. The analyst can therefore decide how long the process should run. Once the random search is performed, the analyst can look at the top candidates to get an idea of which combinations work best, before doing a normal grid search based on the result of the random search.

3.11 Prediction intervals

When we are using mathematical models to make predictions, we often want an assurance of how good the prediction is. For example, when using models for medical diagnostics it is helpful to know the strength of a prediction before making any decisions. The width of a prediction interval can be used to draw conclusions about the strength of a prediction, where a narrow interval would mean that the prediction is strong and a wide interval would correspond to a weak prediction. This section will cover two methods for producing prediction intervals to accompany the prediction.
3.11.1 Standard deviation

Assuming that the error term is normally distributed and estimating the corresponding standard deviation might be the most common method for producing a prediction interval, which works good if the data Gaussian. Let the estimated standard deviation be $\hat{\sigma}$ which can be estimated by dividing the $RSS$ by its degrees of freedom ($df$), $df = n - p$, where $n$ is the number of data points and $p$ is the number of estimated parameters (Weisberg, 2014, p. 26),

$$\hat{\sigma}^2 = \frac{RSS}{n - p}. \quad (39)$$

Now suppose that we have learned a model and estimated $\hat{\sigma}$. Let $x$ be an unknown observation and that our model predicts the response $\hat{y}$. A prediction interval for $\hat{y}$ would be

$$[\hat{y}(x) - z\hat{\sigma}, \ \hat{y}(x) + z\hat{\sigma}] \quad (40)$$

for some $z$. For instance, if we would be interested in a 95% prediction interval we would let $z \approx 2$.

3.11.2 Conformal prediction

Conformal prediction is an alternative method for producing prediction intervals. This method allows the prediction interval to be larger for some predictions and smaller for others, which is usually the case. Conformal prediction can be implemented in various different ways, but the idea is to convert a heuristic notion of uncertainty from any model into a rigorous one (Angelopoulos and Bates, 2022, p. 5) \[1\]. This section briefly explains the idea and how conformal prediction will be implemented in the worked example.

In the same way as we learn a regression model $\hat{y}(x)$ for the response, a regression model $\hat{r}(x)$ can be learned for the residual which will try to predict $|y - \hat{y}(x)|$. Using the regression model $\hat{r}(x)$ to make prediction intervals is often poor in practice, but conformal prediction can be used to make it more rigorous (Angelopoulos and Bates, 2022, p. 9). Given our regression models $\hat{y}(x)$ and $\hat{r}(x)$, we can define a score function $s(x, y)$ in the following way

$$s(x, y) = \frac{|y - \hat{y}(x)|}{\hat{r}(x)}, \quad (41)$$

which we will use to calculate calibration scores. If we then let $\hat{q}$ be the $(1 - \alpha)(n + 1)/n$ quantile of the calibration scores, where $\alpha$ is the significance level, we can make prediction intervals for $\hat{y}$ as follows

$$[\hat{y}(x) - \hat{q}\hat{r}(x), \ \hat{y}(x) + \hat{q}\hat{r}(x)]. \quad (42)$$

As Angelopoulos and Bates (2022, p. 10) state, when we use conformal prediction and the score function defined as above we guarantee that

$$P[s(x_{new}, y_{new}) \leq \hat{q}] \geq 1 - \alpha \implies P[|y_{new} - \hat{y}(x_{new})| \leq \hat{q}\hat{r}(x_{new})] \geq 1 - \alpha. \quad (43)$$

26
Note that in equation 43 the probability is over the randomness not only in $x_{new}$ but also in the training data and the calibration data.

Figure 10: A diagram visualizing the typical workflow when solving a regression problem.

4 Methodology and worked example

This chapter will cover the workflow which were used in the given regression problem. The workflow is visualized in figure 10.

Gathered data needs to be cleaned, explored and prepared before a model should be learned from it. Data cleaning involve removal of irrelevant parts and the detection and removal of observations with inaccurate or missing data. Exploration involve visualizing and analysing the data. Preprocessing also involve transformation of the variables, normalization, creation of dummy variables, one hot encoding and so on. How the data is preprocessed will affect the final model.

For this thesis Microsoft Excel were used for cleaning the data set. The data set were then converted into a csv-file and the rest of the regression problem were solved in Jupyter Notebook by using python code. The problem, the cleaning using Microsoft Excel and the steps that were taken using python will be described in the following sections, the python code can be found in appendix B.
4.1 The problem

The regression problem in this worked example is to estimate the operation time given the size of a product. Time refers to time per piece and is measured in seconds. The product size is described by the product's length, width, and height and are given in millimeters.

4.2 Cleaning the data set

Data were retrieved from a manufacturing company's reporting system and loaded into Microsoft Excel. The data were then cleansed from inaccurate and irrelevant observations. The data set consisted originally of roughly 3200 observations, which were reduced to around 800 after the cleaning process. Some examples from the cleaning steps are explained below.

An example of inaccurate data was that in some observations the operator had forgotten to pause the operation when the shift was ending, resulting in the time until the next shift got added to the reported time, which usually was until the next morning. Another example of inaccurate data was when the operator start the operation but does not switch operation in the reporting system, resulting in too little time being reported for the operation which is being investigated. This resulted in that times which were unreasonable high (above 30 seconds per piece) or low (below 0.36 seconds per piece) where investigated and removed. Note that this decision in a way force estimations for new observations to be in this interval.

An example of an anomaly or outlier is when a product contain more than one piece, since what is included in the reported time is the production time for more than one product size and how the time is distributed between the different sizes is unknown. Moreover, the finished product could contain i.e. two pieces of one size and three pieces of another size which complicate things even further.

Another example of something that were decided to be seen as anomalies were the very few observations which had adhesive tape applied, which seemed to affect the way that the product was being manufactured by a lot. The decision was made to investigate these products separately, instead of creating a dummy variable, since there were very few observations which had adhesive tape applied.

The response variable in the regression problem, time per piece, is a transformed variable which was created by dividing the reported time with the order quantity. The length, width, and height of the product were not given in different columns but were instead stated in the product name and could therefore be retrieved by using some simple Microsoft Excel functions.

Some discussions were also held with the operators which all said that the material type, which could be grouped into three categories, affect the production time in different ways. Since it seemed like there were some differences in how the different material groups were being handled by the operators, the decision was made to separate the three groups into three different data sets,
instead of just creating dummy variables. Note that this lead to the learning of three models instead of one. In the thesis we will only consider the most common material group from this point, and let the other material groups be analysed separately.

After the above steps the filtered data set were converted to a csv-file.

![Figure 11: A table showing a summary of the data set after the cleaning process.](image)

### 4.3 Exploring, visualizing and transformations

In Jupyter Notebook the relevant packages from matplotlib, scikit-learn, pandas and numpy were imported. Matplotlib\(^1\) is a plotting library which is used for visualizing data. Scikit-learn\(^2\) is a machine learning library with tools for predictive data analysis. Pandas\(^3\) is also used for data analysis and manipulating data frames. Numpy\(^4\) is a fundamental package for scientific computing and for creating multi-dimensional arrays in python.

The data set were loaded into Jupyter Notebook using pandas. A data summary was then created using pandas and various scatter plots were created using matplotlib for trying to understand the relationships between the variables. The data summary and the scatter plots are shown in the two figures 11 and 12.

We start by analysing the data summary. In this data set, the mean time (seconds per piece) is roughly 10 and the median is around 8. The standard deviation is around 7. We also note that around 50 percent of the observations lie in the interval \([4, 14]\). Length (L), width (W) and height (H) describe the shape of the product. In the data set we have observations in the interval \([30, 2000]\) for length, \([30, 780]\) for width and \([3, 125]\) for height. We also note that most observations lie in the interval \([235, 388]\) for length, \([130, 295]\) for width and \([10, 45]\) for height. This is important to remember if the model is put into production, since the performance might be worse outside of these intervals.

\(^1\)https://matplotlib.org/stable/index.html
\(^3\)https://pandas.pydata.org/
In figure 12 there are various scatter plots showing the relationships between time and the explanatory variables. Area (L x W) and Volume (L x W x H) are two of the transformed variables, the idea behind these transformations were that both the area and volume of a product should affect the production time. After that some power transformations were tried, where the logarithmic transformation and raising the variables to the power 1/3 both seem like good alternatives. The reason for the why there are vertical lines in the plots in figure 12 is because the products are measured in millimeters, and some product dimensions are more common than others.

Figure 12: Scatter plots visualizing the relationship between the response variable and the explanatory variables.
4.4 Models

Four different models will be learned and evaluated. The first one is the old model which is used to estimate new operation times today. The second model will be a linear regression model which will be trained using the squared error loss, this model will be called "LinearRegression" in the figures. The third model will also be a linear regression model, but will be trained using the huber loss and will be called "HuberRegressor" in the figures. The fourth model will be a random forest model, which will be called "RandomForest".

4.5 Hyperparameter optimization

Before learning the models the optimal hyperparameters need to be found, this will be done by using the grid search method. For the first linear regression model, which is trained using the squared error loss, we will not optimize any hyperparameters. For the linear model trained using the huber loss we will optimize the hyperparameter epsilon, which decides the robustness of the model.

In neither of the linear regression models the hyperparameter fit intercept will be set to false, which would force the regression line to pass through the origin. The reason for not doing this is that some of the transformed variables seem to have a linear relationship with the response variable that does not pass through the origin, this can be seen in figure 12.

The parameters that will be optimized for the random forest models are criterion, max depth, min samples split and max features. Criterion is what function should be used when measuring the quality of a split, we will choose between squared error and absolute error. Max depth is the maximum depth of the tree. Min samples split state how many samples there have to be in a region for the next splitting point to be considered in that region. Max features states how many features are considered when searching for the next splitting point.

4.5.1 Optimization results

Grid search were performed for the two models and for the linear model trained using the huber loss the hyperparameter epsilon were optimized to be equal to 1. For the random forest model the hyperparameters were optimized simultaneously with the following results, criterion were optimized to use absolute error, the max depth were optimized to be 65, max features were optimized to be 3 and min samples split were optimized to be 2. A max depth of 65 is very large for trees, which should be taken into account if we later would suspect the random forest model to suffer from overfitting.

4.6 Comparing the models

4.6.1 Visualizing the performance

Various plots were made to help us understand how each of the models perform. In figure 13 we can see the relationship between the estimated $y$, the residuals
and the observed $y$ on the testing data, i.e., observations which were not used when training the model. For the three learned regression models we can clearly see a heteroskedastic relationship between the fitted values and the residuals, since the residuals seem to grow with the estimations. For the old model there seems to be a different relationship, the reason for this seem to be that it typically estimate low values, and as observed times can not be negative, most residuals will be positive. From figure [13] we can conclude that the old model does not describe the reality in a good way. We can also see in figure [13] that both the linear model trained using the huber loss and the random forest model seem to have fit the data the best.

In figure [13] we can also see that the linear model trained using the squared error typically overestimates $y$ for low values. We also note that it have not estimated any values below $\approx 4$ on the testing set. A reason for this could be that the intercept is too high, if the model should be implemented we should therefore analyse the performance when the intercept is forced to be zero.

In figure [14] the relationship between the observed $y$, the estimated $y$ and three of the explanatory variables, length, width and height are visualized. For the two linear regression models we clearly see that the estimations grow with the size of the product. It also looks like the variable length have a big impact on the estimations, since the estimations are not as spread out in those graphs. For the random forest model we can suspect a similar relationship, but the estimations seem to be more spread out, which means that those variables does not have a big impact on the estimations. Note that similar graphs can be made and analysed for all the other explanatory variables as well.

Figure [14] clearly show that the two variables length and width have a big impact on the estimation and that the variable height does not have the same impact. We can also see that the old model seem to have a really strange relationship between the size of the product and the estimation, since the estimations grow when the size becomes smaller, especially for $L, W < 250$.

### 4.6.2 Cross validation

K-fold cross validation were performed two times for all four models, and the results are summarised in the box plots in figure [15]. One used the squared error loss as error function and the other one used the absolute error loss. We can conclude that the old model performs much worse than the three regression models. Moreover, the random forest model seem to perform better than both of the linear regression models.

If we only compare the two linear models, we need to remember that one have been trained using the huber loss and the other have been trained using the squared error loss. That is the reason for why one performs better than the other depending on which box plot we analyse. We should also remember that the random forest model have been trained using the mean absolute error, despite this it still outperforms the other models when evaluated using the mean squared error.
Figure 13: Scatter plots visualizing the relationship between the response, the fitted values and the residuals on testing data for each of the four different models.
Figure 14: Scatter plots visualizing the relationship between the observed $y$, the estimated $y$ and three of the explanatory variables, length, width and height on testing data for each of the four different models.
Figure 15: Box plots visualizing the results from the cross validation. The first box plot evaluated the models using the mean absolute error and the second box plot evaluated the performance using the mean squared error.
4.7 Variable selection

If some of the models will be put into production, variable selection should be performed. Both forward and backward selection where performed for the linear regression model trained using the squared error loss and the results can be seen in figure 16. Since both methods are greedy, one reasoning can be to use the results from forward selection if we want to use few parameters and backward selection if we want to keep most of the parameters. Note that the selection procedure was based on a 10-fold cross validation and the $R^2$’s in figure 16 were calculated as in section 3.2.8 on training data.

<table>
<thead>
<tr>
<th>LinearRegression</th>
<th>Forward selection</th>
<th>Backward selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Selected</td>
<td>$R^2$ with # of variables</td>
</tr>
<tr>
<td>1</td>
<td>Volume*1/3</td>
<td>0.22</td>
</tr>
<tr>
<td>2</td>
<td>W</td>
<td>0.24</td>
</tr>
<tr>
<td>3</td>
<td>Area*1/3</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>H</td>
<td>0.25</td>
</tr>
<tr>
<td>6</td>
<td>$H^1/3$</td>
<td>0.26</td>
</tr>
<tr>
<td>7</td>
<td>logW</td>
<td>0.20</td>
</tr>
<tr>
<td>8</td>
<td>$L^1/3$</td>
<td>0.26</td>
</tr>
<tr>
<td>9</td>
<td>logL</td>
<td>0.26</td>
</tr>
<tr>
<td>10</td>
<td>$W^1/3$</td>
<td>0.26</td>
</tr>
<tr>
<td>11</td>
<td>logArea</td>
<td>0.26</td>
</tr>
<tr>
<td>12</td>
<td>logVolume</td>
<td>0.26</td>
</tr>
<tr>
<td>13</td>
<td>logW</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Figure 16: A table showing the results of the variable selection procedure for the linear regression model trained using the squared error loss.

4.8 Predictions

4.8.1 Point predictions

Once a model is learned, it is quite easy to make predictions. In python this can be done with just a few lines of code. The linear regression model learned using python can easily be exported and implemented in Microsoft Excel, since it is just a linear function. If another regression model is learned it might make it more difficult to implement in excel, but if some assumptions are made it should still be possible to implement most models in a simplified fashion. For example, one could make a lot of point predictions for different $x$-values and export those to excel. Then by assuming that $x$-values close to each other should have roughly the same response, a prediction for a new observation can be made by finding the already made prediction for any $x$-value close to the new observation. If a decision tree or random forest model have been learned, it can be exported to excel using the package sklearn2excel\footnote{https://pypi.org/project/sklearn2excel/}, which will convert all of
the model’s splitting points into a single table.

In figure 17 a couple of point predictions using the different models can be seen.

![Figure 17: A table showing a couple of predictions made by the models. This clearly shows that the old model is not working as intended.](image)

### 4.8.2 Prediction intervals with conformal prediction

Conformal prediction intervals were learned for the three regression models, these are visualized in figure 18. Since the prediction intervals in figure 18 were made with a significance level of 0.5, roughly 50% of the observations should land inside the interval. Since the intervals are wider for larger estimations and narrower for smaller estimations, we can conclude that the predictions are stronger when the models predict low values and more uncertain on higher values.
Figure 18: Graphs showing prediction intervals made with conformal prediction.
5 Discussion and conclusions

In this thesis 4 different models have been analysed, three of them were regression models which all outperformed the old model. According to both cross validations plots in figure 15 the random forest model had the best performance out of the three regression models, but in relation to the old model they all seem to have similar performance. Since the random forest model is a bit more difficult to implement, we conclude that either of the linear regression models should be put into production.

In figure 13, 14 and 17 we can clearly see that the old model is not working as intended. A reason for this is that the model is not being used in the way that it was designed. This part could be elaborated, but it would serve no purpose for the thesis. Instead we conclude that the way the old model is being used today gives very poor performance.

Since each tree in the random forest model are allowed to be grown extremely deep, there is a suspicion of the random forest model being overfitted, even though the random forest model performs well in the cross validations, figure 15. One explanation of the good performance in the cross validations is that some product sizes occur several times in the data set, which means that the test set may include already seen \( x \) values. This was investigated and it turned out that only around one tenth of the product sizes in the observations occurred more than once.

Even though the linear regression model trained using the squared error loss have good performance, it seem to struggle with estimating low values and usually overestimate operation times, according to figure 13, therefore this model should be adjusted and analysed once again. One idea would be to log transform the time variable since this would make the smaller times have a larger impact when training the model.

There seem to be a strong relationship between operation times and the size of the product, but still neither of the three regression models seem capable of fully explaining the variance in the data. In figure 16 we can see that the coefficient of determination is roughly 0.25, which suggests that the variance in the data can not be explained by the explanatory variables. Since there are many uncontrollable variables affecting the operation times, which was stated in the introduction, we conclude that the performance of the models might still be acceptable.
References


A Decomposition of the MSE

The mean squared error (MSE) can be used to describe how a model generalizes and can be decomposed into the squared bias, the variance and the irreducible error. Let \( y = f(x) + \epsilon \), where \( f \) describes the true relationship between \( y \) and \( x \) and \( \epsilon \) is the irreducible error term with mean zero and variance \( \sigma^2 \). Assume that we have approximated \( f \) with the function \( \hat{f} \), where \( \hat{f} \) is considered to be random since it have been approximated using random data. We can then define the MSE on a new random observation \((x, y)\) in the following way

\[
MSE = E[(y - \hat{f}(x))^2]
\]

\[
= E[y^2] - 2E[y\hat{f}(x)] + E[\hat{f}(x)^2]
\]

\[
= f(x)^2 + \sigma^2 - 2f(x)E[\hat{f}(x)] + Var[\hat{f}(x)] + E[\hat{f}]^2
\]

\[
= (f(x) - E[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2
\]

\[
= Bias[\hat{f}(x)]^2 + Var[\hat{f}(x)] + \sigma^2
\]

(44)

since the following equations hold

\[
E[y^2] = E[(f(x) + \epsilon)^2]
\]

\[
= E[f(x)^2] + 2E[f(x)\epsilon] + E[\epsilon^2]
\]

\[
= f(x)^2 + 2f(x)E[\epsilon] + \sigma^2
\]

\[
= f(x)^2 + \sigma^2
\]

(45)

\[
E[y\hat{f}(x)] = E[(f(x) + \epsilon)\hat{f}(x)]
\]

\[
= E[f(x)\hat{f}(x)] + E[\epsilon\hat{f}(x)]
\]

\[
= f(x)E[\hat{f}(x)] + E[\epsilon]E[\hat{f}(x)]
\]

\[
= f(x)E[\hat{f}(x)]
\]

\[
E[\hat{f}(x)^2] = Var[\hat{f}(x)] + E[\hat{f}]^2
\]
B Python code

This appendix includes a subset of the code which were used during the thesis. The reason for not including all of the code is because a lot is reused with minor modifications, i.e. the code used for the figures.

B.1 Import statements

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import HuberRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import sklearn.model_selection
```

B.2 Loading the data

```python
data = pd.read_csv("Group1.csv")
```

B.3 Exploring and visualizing the data

```python
data["Time","L","W","H"].describe()

y = np.array(data["Time"])
X = np.array(data["L","W","H"])

plt.scatter(X[:,0], y, alpha=0.3)
plt.xlabel("Length")
plt.ylabel("Time")
```

Various scatter plots can be visualized with only small modifications to the cell above, below are some examples.

```python
plt.scatter(X[:,0]**(1/3), y, alpha=0.3)
plt.xlabel("(Length)^1/3")
plt.ylabel("Time")

plt.scatter(np.log(X[:,0]*X[:,1]*X[:,2]), y, alpha=0.3)
plt.xlabel("log(Volume)")
plt.ylabel("Time")
```
B.4 Transforming the variables

```python
data["logL"] = np.log(data["L"])
data["logW"] = np.log(data["W"])
data["logH"] = np.log(data["H"])
data["logArea"] = np.log(data["L"]*data["W"])
data["logVolume"] = np.log(data["L"]*data["W"]*data["H"])
data["L-1/3"] = data["L"]**(1/3)
data["W-1/3"] = data["W"]**(1/3)
data["H-1/3"] = data["H"]**(1/3)
data["Area-1/3"] = (data["L"]*data["W"])**(1/3)
data["Volume-1/3"] = (data["L"]*data["W"]*data["H"])**(1/3)
```

B.5 Hyperparameter optimization

```python
y = data["Time"]
X = data[["L","W","H","logL","logW","logH","logArea","logVolume","L-1/3","W-1/3","H-1/3","Area-1/3","Volume-1/3"]]

epsilon = [1,1.1,1.3,1.5,2,5]
param_grid = {"epsilon": epsilon}
model = HuberRegressor(max_iter=1000)
model_grid = GridSearchCV(estimator = model,
                          param_grid=param_grid,
n_jobs = -1)
model_grid.fit(X,y)
print(model_grid.best_params_)
```
B.6 Visualizing performance on test data

```python
import numpy as np
import matplotlib.pyplot as plt

# Prepare data

# Split data
X, y = np.array(data["Time"]), np.array(data["Time"])  # This should be corrected
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

# Build model
lm1 = LinearRegression()
lm1.fit(X_train, y_train)
y_pred_lm1 = lm1.predict(X_test)

# Plot results
plt.scatter(y_pred, residuals, alpha=0.7)
plt.plot([0, 30], [0, 0], "r")
plt.xlabel("Estimated y")
plt.ylabel("Residuals")
plt.title("LinearRegression")
plt.scatter(y_pred, y_test, alpha=0.5, label="observed y")
plt.plot(y_pred, y_pred, "r", label="estimated y")
plt.xlabel("Estimated y")
plt.ylabel("y")
plt.title("LinearRegression")
plt.legend()
plt.scatter(X_test[:, 0], y_test, label="observed y", alpha=0.7)
plt.scatter(X_test[:, 0], y_pred, label="estimated y", alpha=1)
plt.xlabel("Length")
plt.ylabel("Time")
plt.legend()
plt.title("LinearRegression")
```

44
B.7 Cross validation

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, HuberRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold

y = data["Time"]
old_model = data["Old_model_time"]
X = data[["L","W","H",
          "logL","logW","logH","logArea","logVolume",
          "L^1/3","W^1/3","H^1/3","Area^1/3","Volume^1/3"]]
models = [old_model,
          LinearRegression(),
          HuberRegressor(epsilon=1,max_iter=500),
          RandomForestRegressor(criterion = "absolute_error",
                                 max_depth=65,
                                 max_features = "sqrt",
                                 min_samples_split=2)]

n_fold = 10
tf = sklearn.model_selection.KFold(n_splits=n_fold,shuffle=True)

for i, (train_index, val_index) in enumerate(cv.split(X)):
    for j, model in enumerate(models):
        if j == 0: # if old model
            y_val = y.iloc[val_index]
y_pred = old_model[val_index]
error_matrix[i,j] = np.mean(np.abs(y_val - y_pred))

        else:
            X_train,X_val = X.iloc[train_index],X.iloc[val_index]
y_train,y_val = y.iloc[train_index],y.iloc[val_index]
model.fit(X_train,y_train)
y_pred = model.predict(X_val)
error_matrix[i,j] = np.mean(np.abs(y_val - y_pred))

plt.boxplot(error_matrix)
plt.xticks(np.arange(len(models))+1,("Old Model",
                                      "LinearRegression",
                                      "HuberRegressor",
                                      "RandomForest"))
plt.title("Cross validation error for different methods")
plt.ylabel("Mean absolute error")
plt.show()
print(np.mean(error_matrix,axis=0))
```
B.8 Variable selection

```python
linear_model = LinearRegression()

y = data["Time"]
X = data["L","W","H",
        "logL","logW","logH","logArea","logVolume",
        "L^1/3","W^1/3","H^1/3","Area^1/3","Volume^1/3"]
print("Selected variables using forward selection (in order)")
prev = np.zeros(X.shape[-1])
for i in range(1,X.shape[-1]):
    sfs = SequentialFeatureSelector(linear_model,
                           n_features_to_select=i,
                           direction = 'forward',cv=10)
    sfs.fit(X,y)
    for j,k in enumerate(sfs.get_support()*1-prev):
        if k == 1:
            print(X.columns[j])
    prev = sfs.get_support()*1

# Calculating the coefficient of determination using the top 5 variables
y = data["Time"]
X = data["Volume^1/3","W","Area^1/3","L","H"]

linear_model = LinearRegression()
linear_model.fit(X,y)
print(f"{linear_model.score(X,y):.5f}")
```

46
B.9 Point predictions

```python
# Define the models
lm1 = LinearRegression()
lm2 = HuberRegressor(epsilon=1, max_iter=500)
rf = RandomForestRegressor(criterion = "absolute_error",
                          max_depth=65,
                          max_features = "sqrt",
                          min_samples_split=2)

## Train the models
y = np.array(data["Time"])  
X = np.array(data[['L', 'W', 'H','logL', 'logW', 'logH',
                  'logArea', 'logVolume',
                  'L^1/3', 'W^1/3', 'H^1/3',
                  'Area^1/3', 'Volume^1/3']])
lm1.fit(X,y)
lm2.fit(X,y)
rf.fit(X,y)

## Define x_new using L, W, H
L = 400
W = 300
H = 30

# -------------------
x4 = np.log(L)  
x5 = np.log(W)  
x6 = np.log(H)  
x7 = np.log(L*W)  
x8 = np.log(L*W*H)  
x9 = L**(1/3)  
x10 = W**(1/3)  
x11 = H**(1/3)  
x12 = (L-W)**(1/3)  
x13 = (L+W+H)**(1/3)
# -------------------
x_new = np.array([L,W,H,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13]).reshape(1,-1)

## Point prediction for x_new
print(f"LinearRegression: {lm1.predict(x_new).item():.2f}"
print(f"HuberRegressor: {lm2.predict(x_new).item():.2f}"
print(f"RandomForest: {rf.predict(x_new).item():.2f}")
```

```
B.10 Conformal prediction

```python
import numpy as np
import torch
from sklearn.linear_model import LinearRegression

y = np.array(data["Time"])
X = np.array(data[["L","W","H", "logL","logW","logH", "logArea","logVolume", "L^1/3","W^1/3","H^1/3", "Area^1/3","Volume^1/3"]])
X_train, X_cal, y_train, y_cal = train_test_split(X,y, test_size = 0.3, shuffle=True)

# Learn y_hat and r_hat
y_hat = LinearRegression()
r_hat = LinearRegression()
y_hat.fit(X_train, y_train)
y_pred = y_hat.predict(X_train)
r_hat.fit(X_train, np.abs(y_train - y_pred))

# Calculate calibration scores
scores = np.abs(y_hat.predict(X_cal)- y_cal)/ r_hat.predict(X_cal)

# get the score quantile
n = len(X_cal)
alpha = 0.5
tensor_scores = torch.Tensor(scores)
qhat = torch.quantile(tensor_scores, np.ceil((n+1)*(1-alpha))/n).item()

# create the prediction sets
y_pred = y_hat.predict(X_cal)
r_pred = r_hat.predict(X_cal)
prediction_sets = np.array((y_pred - qhat * r_pred, y_pred + qhat * r_pred))

# plot the prediction interval
plt.xlabel("Estimated y")
plt.ylabel("y")
plt.scatter(y_hat.predict(X),y, alpha=0.3, label = "observed y")
order = np.argsort(y_pred)
plt.plot(y_pred[order], prediction_sets[1, :][order], color="C1", ls="--", label = f"significance level: {alpha}")
plt.plot(y_pred[order], prediction_sets[0, :][order], color="C1", ls="--")
plt.fill_between(y_pred[order].ravel(), prediction_sets[0, :][order].ravel(), prediction_sets[1, :][order].ravel(), alpha=0.2)
plt.legend()
plt.title("LinearRegression")
```

48