UPPSALA
UNIVERSITET

UPTEC F 24001

Examensarbete 30 hp

January 2024

Using artificial intelligence to improve time estimation for project management

Marcus Bonnedahl

Civilingenjörsprogrammet i Teknisk Fysik

UPPSALA
UNIVERSITET

Using artificial intelligence to improve time estimation for project management

Marcus Bonnedahl

Abstract

Time estimation is an important aspect in project management. Failure to make accurate estimates can lead to large consequences. Despite this, humans tend to make fairly inaccurate estimates when tasked to, often underestimating the time something will take substantially. This thesis explores using artificial intelligence and machine learning to produce time estimates for the life science company Biotage. A predictive model can be trained using previous projects as samples, including time reporting data for employees as the output variable.

A total of 12 completed projects were found that had both sufficient time reporting data and some project information. Previous projects took on average 55.1% longer to complete than estimated at the start of the project. Every project had one or more of the following: project description, work breakdown structure and/or Gantt chart. However, the level of detail in almost all of the projects was very low, making it difficult to extract useful features. A constant-time model (predicting that every project takes the same amount of time), had a Root Mean Squared Error (RMSE) of 5058 hours and a Mean Absolute Percentage Error (MAPE) of 282%. Another model that took into account whether the project was a software only, hardware only or both had a RMSE of 4269 hours and MAPE of 320%. Due to the scarcity of data, no further improvements were made. It was determined that in order to develop a predictive model that can match human estimates, at least one of the following had to be true: *Better level of detail in the data*, *bigger sample size of previous projects*, or *projects being more similar so that they share common features more often*.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Markus Östman Ämnesgranskare: Ping Wu

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

Att kunna uppskatta hur lång tid något kommer att ta är viktigt, inte minst inom arbetslivet. Företag använder sig idag utav olika tekniker för att göra det enklare att ta fram tidsestimat. Trots detta så är vi människor något bristfälliga på att ta fram noggranna estimat. Forskning har visat att när man bett en grupp studenter att uppskatta hur lång tid det kommer ta dem att skriva en uppsats, så tog det 64% längre än vad de uppskattat. Liknande forskning visar att detta stämmer in på grupparbeten också. Överlag så tenderar vi att underskatta hur lång tid något kommer att ta. Ett sätt att försöka åtgärda detta är Artificiell intelligens och maskininlärning, ett område som är hett just nu.

Biotage är ett globalt företag som är verksamma inom områden som life science och kemi, där de utvecklar innovativa produkter för att möta deras kunders behov. De har ett kontor i Uppsala som är inriktat på produktutveckling. Med över 3000 utvecklade produkter så kan man tänka sig att Biotage har bemästrat att göra bra tidsestimeringar, men det visar sig att även de underskattar tiden projekt tar att färdigställa, ungefär 55% längre tid i genomsnitt än uppskattat. I dagsläget fungerar det ungefär som så att en projektledare frågar anställda hur lång tid deras del av ett projekt kommer att ta, sedan summerar projektledaren det och får fram ett tidsestimat över hela projektet. Här skulle man kunna använda en AI istället där man matar in de olika delarna projektet består av och så får den räkna ut en tidsuppskattning. Hur ska då en AI veta hur lång tid varje del av ett projekt tar om inte ens de anställda har speciellt bra koll? Det är där maskininlärning kommer in i bilden.

Fördelen med maskininlärning är att vi kan mata in massor av gammla projekt och låta AIn lära sig av dem, och på så sätt lista ut hur lång tid varje del av ett projekt tar. För att detta ska fungera så krävs det två saker. Ett, vi måste ha tillräckligt med information om tidigare projekt så vi kan avgöra vad projekten faktiskt innehöll. Det måste gå att bryta ner projektet i mindre delar så att varje del kan tilldelas hur lång tid den delen tar. Två, vi måste veta hur lång tid ett projekt tog totalt. Om vi vet att ett projekt innehöll delarna X, Y och Z, och att projektet tog 2000 timmar att slutföra, så ger det lite information om hur lång tid de olika delarna tar. Metoden som kan användas för detta kallas Linjär Regression (Linear Regression på engelska). Detta innebär också att vi behöver ha tillräckligt många tidigare projekt att träna på (ju fler desto bättre!). Man kan sedan ta en andel av de tidigare projekten för att testa hur pricksäker AI-modellen man tränat är.

För detta kunde Biotage bidra med tidsrapportering för anställda, dvs hur många timmar jobbade den här anställda denna dagen på det här projektet. Genom att summera det för alla anställda så gick det att få fram hur många timmar tidigare projekt tog. De bidrog även med tillgång till information om tidigare projekt. Totalt hittades 12 tidigare projekt som hade både information om projektet och tidsrapportering. Med detta kunde en AI tränas med maskininlärning. Den första och mest simpla modellen var att låta varje projekt ta lika lång tid, utan att ta hänsyn till vilka delar projekten innehöll. Denna modell visade sig vara dålig, den hade ett genomsnittligt fel på 282% (jämfört med Biotages nuvarande fel på 55%). Nästa AI-modell tog hänsyn till om projektet var ett mjukvaruprojekt, hårdvaruprojekt, eller både och. Denna modell var till och med sämre,

och hade ett genomsnittligt fel på 320%.

Tyvärr gick det inte att utveckla bättre modell, då den information om gamla projekt som fanns tillgänglig var väldigt odetaljerad, och gjorde det näst intill omöjligt att dela upp projekten i mindre delar. Detta visar vikten i att ha detaljerad och väl strukturerad data att jobba med när man ska använda maskininlärning. Den information om gamla projekt som fanns var ofta så pass bristfällig att man inte ens förstod vad projektet handlade om. Även mängden gamla projekt var lite för liten, helst vill man ha mer än 12 att arbeta med. Till sist så är projekten Biotage håller på med väldigt omfattande och varierande, vilket leder till att varje projekt innehåller mycket unika moment. För att maskininlärning ska fungera så krävs det att projekt har delar som återkommer i flera olika projekt, och inte är unika till ett enda. Att använda AI till att göra tidsestimeringar är lockande, men ställer höga krav på omständigheterna.

Contents

1 Introduction

1.1 Background

Time estimation is an important aspect of project management. It plays a crucial role in successfully completing various tasks and endeavours. Being able to accurately predict how long a task or project will take is a critical factor when making informed decisions and managing resources in an efficient way. Failure to make accurate estimates can severely impact productivity and outcomes, and could be the difference between a successful company and a bankruptcy.

Despite the importance of time estimation, humans are not particularly good at it. In general, there is a tendency to underestimate the amount of time needed to complete a task. This is known as the *planning fallacy* [1]. A study in 1994 had students estimating the time it took for them to write their senior theses. The average estimate was 33.9 days, but the actual average time to complete it was 55.5 days, 63.7% longer than estimated [2].

Biotage is a global company specializing in providing innovative solutions for various application in the field of life science and chemistry [3]. At the moment of writing this thesis, they have close to 550 employees and over 3000 developed products. Biotage continuously develops new products and improves their existing ones, often tailored to suit the specific customer's needs. It is therefore easy to see why it is important to them (and their customers) to be able to give accurate time estimates. However, thus far, Biotage has no systematic way to produce time estimates. Most often, individual employees are given a list of tasks to complete, and are asked how long they believe the task will take to complete. Some examples of tasks would be to design a component, implement a feature or to test the product. The project managers then collect the estimates from the various departments and create a final estimate of project length. As mentioned previously, humans are rather poor at time estimates. This could lead to two individuals giving two completely different estimates for the same task. In addition, the planning fallacy means that most estimates are too optimistic, and this seemingly holds true for Biotage as well, but to what extent is unknown.

1.2 Artificial intelligence

Seeing as humans tend to produce inaccurate time estimates, it may be worth considering artificial intelligence (AI) as a tool to lessen this issue. The core idea would be to let a human input a list of tasks to the AI, and then use a predictive model to calculate an estimate. There are two potential benefits to this idea. First, we eliminate the problem of different employees making wildly different estimates, thus getting a result that is more standardized. It is not unreasonable to expect it to make the estimates more consistent. It also makes it easy to fine-tune the predictive model as more data becomes available. The second reason is that it opens up the possibility to use machine learning to train the model. This means employees no longer have to guess how long a typical task takes, and instead let the AI use real data to estimate it, which would in turn hopefully lead to more accurate estimates.

1.3 Purpose and goals

The purpose of this thesis is to explore the various aspects of using artificial intelligence to produce time estimates. While there will be some discussion on this topic in general, most focus will be on time estimation in the context of the company Biotage. The main question the thesis sets out to answer is whether it is possible, given the available data provided by Biotage, to produce time estimates that can match those currently made by employees at the company. This means it is necessary to investigate how accurate Biotage's current estimates are. Then, using any available data to, as well as possible, train a model to produce time estimates and compare them to the human made estimates. Finally, the results should be used to determine what steps can be taken to improve future models.

The main goal is to produce a model using machine learning that is equally good or better at producing time estimates for projects at Biotage than what employees are currently able to produce. A secondary goal is to determine what steps should be taken to improve future time estimation models.

1.4 Tasks and scope

The scope of the thesis will largely depend on the available data. If the data happen to be plentiful, then a large portion of the time will be spent examining it and attempt to train the best possible AI using machine learning. On the other hand, if the data is scarce, then the focus is likely to shift towards determining what can be done to improve future models instead. The workflow of the thesis can be broken down into the following tasks:

1. Familiarization on what Biotage's projects work like, and learning about their current methods of estimating time.
2. Examining the available data from previous projects and sorting out the data that is usable, and from this data select a list of viable previous projects work with.
3. Comparing reported time with estimated time for the selected projects to obtain an estimate of Biotage's current accuracy for time estimates.
4. Attempt to extract features from the selected projects, use these to train the AI using machine learning and measure its accuracy.
5. Determine various shortcomings of the model and data provided, and suggest steps that can be taken to improve future time estimation models.

2 Theory

2.1 What is a project?

Before it is possible to investigate time estimation for projects, one must define what a project is. This definition can be borrowed from Kissflow, which states that *A project is a set of interdependent tasks that have a common goal* [4]. The idea that a project can be broken down into tasks is very useful, and will be central to this thesis. An extension to this idea is that each task can then be broken down into even smaller tasks, if a higher level of detail is desired.

To illustrate this idea with an example, say we are interested in developing a new laboratory instrument (something Biotage frequently does). The whole project is broken down into various tasks, which could include things such as hardware development, software development, and testing&validation. If greater level of detail is desired, each of these tasks could be divided into smaller tasks, i.e. software development might include developing a graphical user interface (GUI) and writing the code that communicates with the hardware. This step of dividing a task into multiple smaller can be repeated as many times as desired. Of course, there is nothing stopping a given task from occurring multiple times in a project.

2.2 Time estimation of a project

With this definition of a project, it is time to start thinking of how to make a time estimation for said project. The most intuitive way is to simply assign an expected completion time for each task in the project. Since a project is merely the sum of its tasks, one can simply add the time estimate of all tasks in the project to obtain an estimate for the entire project.

There is one very important thing to consider, and that is what unit of time to use for the value. There are two options:

- Man-hours (or man-days). This is the amount of time in hours that it would take for one employee to finish the task.
- Calendar days. The expected number of days to finish the task.

The most intuitive choice here is man-hours, for one major reason: there is no need for additional information to be able to assign a time value to a task. In order to use calendar days, we would need to know how many employees are assigned to a specific task (and how much of their time is assigned to that task), since the time to complete the task heavily depend on how many are working on it. Instead, if we use man-hours, we can convert it to calendar time when necessary once we know who is assigned to the task. Another issue with calendar time is that many tasks can not be worked on in parallel, i.e. you can not have the testing&validation team work before there's actually a near-finished product to test. This means that sometimes a delay in one task will delay the entire project in calendar days (because other tasks had to wait for this task to finish), other times the project is not delayed at all (because even though this task took longer, it could work in parallel with other tasks). Both measures end up having their uses. Man-hours helps

with understanding the total workload of the project (and thus things such as how much salary needs to be paid). Calendar days helps us make a prediction around what date the project is finished. This thesis however, will use man-hours only.

2.3 Time estimation model

The general idea for a time estimation model is to divide the project into tasks (or *features*, as is often the term used in machine learning contexts) and let each task have a value (in man-hours). Selecting the proper features for the model is crucial. If the selected features are too specific, i.e. they rarely appear in more than one project, then it will not be possible to generalize the model for future project. On the other hand, if the features are too few and/or generic (i.e. same features show up in most projects), then the accuracy of the model will suffer.

Once every part has a time value assigned to them, the final time estimation can be obtained by adding the time value of all parts. This however places the assumption that the time value of each part is independent from other parts. This is a reasonable assumptions when working with man-hours. One part taking longer than expected may cause another part to be delayed, but it should not affect the number of man hour it takes in any significant way (even though it does affect the time estimation in calendar days).

2.4 Time value of features

Once the proper features has been selected, a time value in hours must be assigned to it. The challenge is knowing what value to pick. There are a few ways to go about it:

- Let one or multiple employees pick a number for each task
- Use data from previous projects to calculate an estimate.
- A combination of the above.

The first option resembles how time estimates are done already, letting an employee come up with a reasonable number. This option is better if there are multiple experts deciding on the values. The benefit would be that the time estimates becomes more standardized, as the same tasks/features are reused. If the time value of a feature turns out to be inaccurate, the employee(s) can decide to change the values of the feature to improve accuracy. There are some pros and cons of this approach. There should most often be one or more experts that can come up with an initial value for a given feature, so it is unlikely there will be a situation where nobody has any idea on how long a feature should take. The downside is that we are still dealing with the fact that humans are not very good at estimating time. There is no guarantee the initial time value for the feature will be accurate. Multiple people deciding together could improve the accuracy, but the planning fallacy means that it is likely the value will be too optimistic. And while the initial time value can be be changed later on, it will requires some effort from the experts, who you would expect to rather spend their valuable time on something else. This method will not be used in this thesis.

The second option is to use data from previous projects to calculate approximate values

for each feature. This way we completely avoid relying on human estimates and instead base our value on how long a feature actually has taken to complete in previous projects. In theory, this approach means that the only thing needed to know is how much time a feature took in a few previous projects to get a fairly good estimate for how long a feature is going to take in this project. In addition, this approach would lead to a good idea of how high risk a given feature is, since it is possible to use a metric such as the standard deviation. In practice however, it is unlikely that this granular data exists. This is the downside of using this data driven approach: no matter how clever we are, in the end, the result will not be better than the data we have available. It is unreasonable to expect a company to have data on how much time was spent on each and every feature. Often though, there will be data available to show how much time was spent in total on previous project, as well as what tasks were included. There are ways to make use of this data to produce time values for each tasks, such as linear regression, which will be explored further into this report. One upside with using a data-driven approach is that the model will improve over time as long as the data from newly finished projects are added to it.

The third option is to mix multiple approaches to cover the weaknesses of each. For example, employees could make an initial guess for a feature and then improve it using previous data where it is available. If there is very little data available for a feature, the model could weigh it so that it mostly uses the estimation made by the employees. If there is a lot of previous data, then that data is likely to be more accurate than the estimation made by the employees, so the previous data can instead be weighted heavily. Though, this approach would still not address the issue of humans making poor estimates. While intriguing, this idea will not be explored in this thesis.

2.5 Machine learning

Machine learning is a branch of artificial intelligence that works with making AI models learn and improve from data, as opposed to explicitly programming its behaviour. There are various approaches in machine learning, each tailored to different scenarios. Supervised learning involves training models on labeled data (data where the outcome is known) to make the model able to predict the value of continuous variables or to make it able to properly classify discrete variables, given some input data. Unsupervised learning instead explores unlabeled data, looking for patterns or underlying structures in the data, such as trying to divide the data samples in to distinct groups (a technique called Clustering). Another approach is reinforcement learning, where models are trained to make sequences of decision, based on the environment they are in. In reinforcement learning, the model is trained by rewarding decisions leading to a good outcome, or penalizing decisions leading to a bad outcome.

2.5.1 Regression

Regression is a supervised learning technique commonly used in machine learning. It is used to establish the relationship between the input variables (features) and the output variable, which in the case of regression is continuous. It can be represented as:

$$Y = f(X, \beta) + e \tag{2.1}$$

where Y is the output variable, X the input variables, β some constants to be determined and e an error term. What is important here is that e is going to be the difference between the real output and the output predicted by the regression model, given some X. Thus the goal is to come up with a function $f(X, \beta)$ and coefficients β that minimizes the error. More specifically, the thing that should be minimized is the squared sum of the error terms from all samples in the training data, also known as the residual sum of squares (RSS), which is expressed as:

$$RSS = \sum_i e_i^2 = \sum_i (Y_i - f(X_i, \beta_i))^2 \tag{2.2}$$

where Y_i refers to the actual output variables, and index i refers to the i-th sample of the training data. The first step is to determine what the function $f(X, \beta)$ is, and it is often done using already existing knowledge about the relationship between X and Y. The machine learning part is then determining the coefficients β that minimizes the RSS.

A widely used and simple form of $f(X, \beta)$ is of the following form:

$$f(X, \beta) = \beta_0 + \beta X \tag{2.3}$$

which can be written as

$$Y = \beta_0 + \beta X + e \tag{2.4}$$

This is known as linear regression, as the output variable Y depends only linearly on X. Here X and β are vectors with the same number of elements as there are features, and β_0 is a constant. A typical example is illustrated in Figure 2.1.

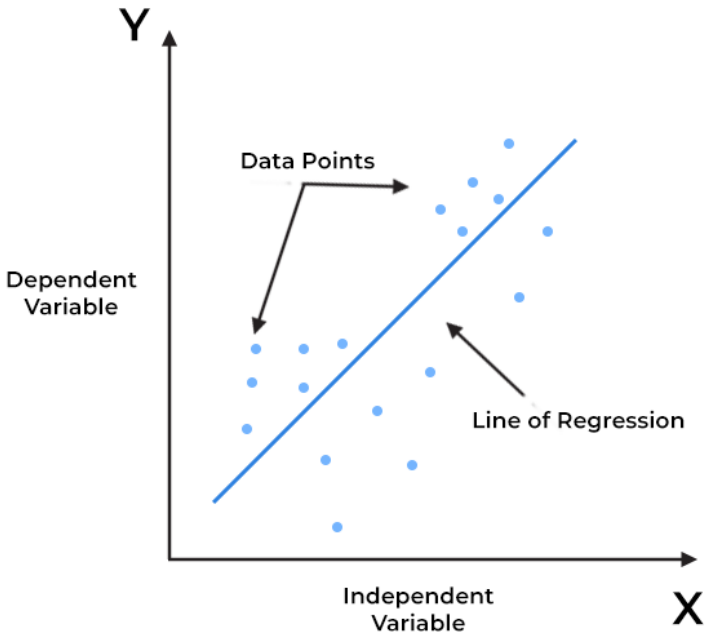

Figure 2.1: Linear regression, the blue line is the best fit given the data points [5].

In order for this model to be accurate, Y needs to depend linearly on all variables X. Given the time estimation model defined in Section 2.3, linear regression is a good choice to use. It is clear that the relationship between the input variables (time taken per task) and output (estimated time for the entire project) is linear.

2.5.2 Model Validation

One or more metrics are needed to determine the accuracy of the model, along with a method to calculate these. A common method to use is k-fold cross-validation. The data set is divided into a number of smaller sets (called folds) equal to whichever value was selected for k. The model is then trained k times, each time with a different fold used to validate the data, and with the remaining $k - 1$ folds used as training data. The error is then the average of all folds, as shown in Figure 2.2.

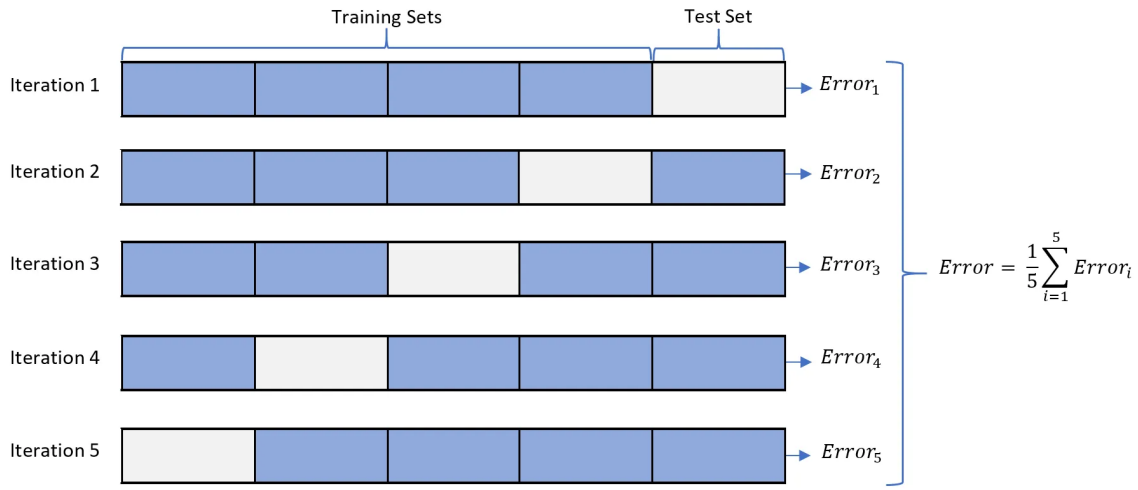

Figure 2.2: A visual representation of k-fold cross-validation [6].

A special case of k-fold cross-validation is when $k = 1$, and this is called Leave-one-out cross-validation. Leave-one-out was selected as the method to validate the model, with the motivation being that it tends to performs better than k-fold [7]. Leave-one-out is more computationally expensive as it has to train the model n times (where n is the number of samples) instead of k, but it is not an issue when the sample size is small (as it ends up being in this case).

Mean Squared Error (MSE) is a commonly used metric for evaluating the accuracy of predictive models. It calculates the average of the squared differences between the predicted values and the actual values. The formula for calculating MSE is

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

where y_i is the real value and $\hat{y}_i$ is the predicted, for sample i, and N the number of samples. A benefit of this metric is that the error $y_i - \hat{y}_i$ is squared, which gives it a higher weight to large errors. This is reasonable, as in a real world situation it is the large errors in time estimation that impacts us, the small errors are likely not that relevant. A downside of using MSE is that the metric it is not very intuitive, as the unit of MSE will be squared time, which lacks a real world interpretation. However, it is possible to instead use the Root Mean Squared Error, which is simply the square root of the MSE. In this case, the unit will be time, and the result will be easier to interpret.

Another metric is the Mean Absolute Percentage Error (MAPE) which is written as:

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{|y_i|} \tag{2.6}$$

which is a quantity in percentage. Unlike MSE, this metric does not give a higher weight to larger errors, which may or may not be a good thing. In addition, it is sensitive to certain outliers, such as when y_i is very small or 0, in which case the error can become very large or even infinite.

Another potential downside that is true for both MSE and MAPE is that they do not differentiate between underestimated and overestimated errors. In the context of time estimation, overestimating how long a project takes is probably not as bad as underestimating it. Thus, it may be advantageous to come up with a metric that gives less weight to over-estimations.

2.6 Project management tools

2.6.1 Project charter

A project charter is a concise document that contains the fundamental details, objectives, scope, and key stakeholders involved in the project. Its purpose is to serve as a quick reference to the purpose and goals of a project, as well as show the constraints. It is often created early on to serve as guide throughout the project lifecycle, and to make sure all the stakeholders have proper information about the project.

2.6.2 Work Breakdown Structure

A Work Breakdown Structure (WBS) is a hierarchical representation of a project, as shown in Figure 2.3. By systematically breaking down the project into smaller elements, it becomes easier to get an overview and plan [8]. It represents the project as a tree structure, with the top level representing the entire project, with each subdivision representing smaller and more detailed parts. This allows employees and project managers to get a good understanding of the whole structure of the project.

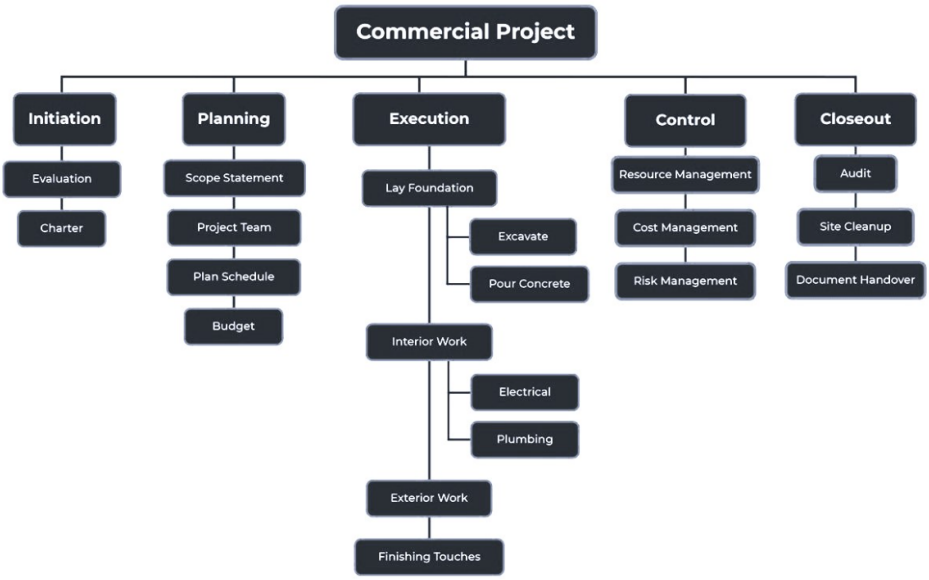

Figure 2.3: A work breakdown structure of a fictional project [8].

2.6.3 Gantt Chart

A Gantt chart is a visual project management tool used both to schedule employees on tasks, and to track the progress of tasks over time [9]. It displays a horizontal timeline which represents the projects duration, with individual tasks represented as a bar. The bars starting position on the horizontal axis determines when the task is supposed to begin, and its ending position when the task is supposed to end. This visual representation of a tasks makes it easy to get an overview of what tasks needs to be done, when they're supposed being, and when they're supposed to be finished. An example of a Gantt chart is shown in Figure 2.4.

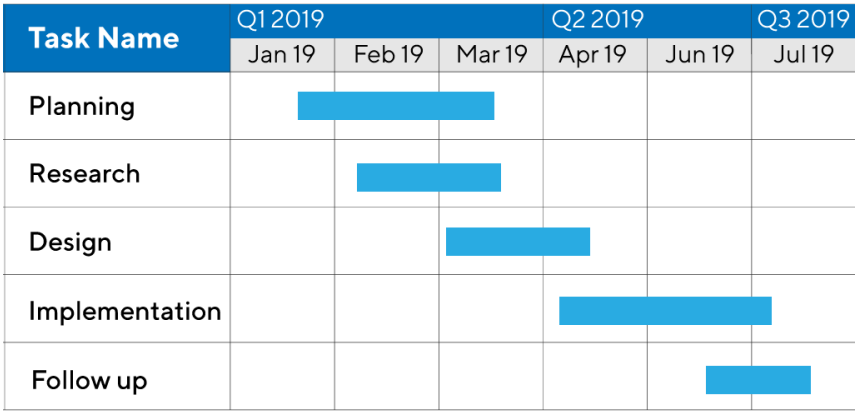

Figure 2.4: Example of a Gantt chart. The blue bars represent the time frame of the various parts of the project [10].

3 Data and Method

Two main sets of data were provided as follows

- Hours logged for previous projects, per employee, per day.
- Various files related to previous projects, often including project and/or time plans.

The first of the two, consisted of one big excel file, which each row being the number of hours logged for a specific employee, for a specific day, for a specific project. The data spanned the time period Jan 1, 2018 to April 29, 2022.

The second set of data is all files located on the company's server for closed projects. This data ranges all the way from 2006 to January 2023. The relevant files for this thesis were anything to do with project planning and time planning, everything else were disregarded. There were no standard format for these files, most commonly they were in the format of PDF (.pdf), Excel (.xlsx), Text document (.docx) or Microsoft project (.mpp). It may be possible to extract a list of tasks from these files, along with an estimation of how long each task takes.

3.1 Data mining

The folder containing the data for all previous projects contained hundreds of thousands of files. To not have to search through it manually, a python script searched through all of them and returned a list of all the files containing any of the following keywords (underscores (_) in the file names were replace with space) : *project plan*, *projektplan*, *projectplan*, *projekt plan*, *time plan*, *time plan*, *tidsplan*, *tidplan*, *time line*, *timeline* or *estimat*. This returned a list of 981 files. A total of 447 files last modified before 2016 were excluded. 28 files were excluded for having having a file format other than *.doc*, *.docx*, *.pdf*, *.mpp*, *.pptx*, *.xls*, *.xlsm*, or *.xlsx*.

From these remaining files, 44 unique projects could be identified. Cross-referencing these with those found in the time reporting data, 15 projects were found that had both types of data. One issue that arises is that a project might have hours logged before or after given time period. Those should not be included in the analysis, but it is not known what those projects are. One way to fix this issue is to exclude projects with time logged near the start or end of the time period, as there is a chance these were active outside of the given time period, and including these projects would underestimate the hours taken for the project. In total, 2 projects were excluded for having hours logged within 2 months of the lower cut-off date (2018-01-01) and 1 project were excluded for having hours logged within 2 months of the upper cut-off date (2022-04-29), leading to a total of 12 projects remaining for further analysis.

3.2 Validity of the data

An important question to ask is whether the given data is valid or not. Another data set for reported time did exist, however access to this data was not granted. Nevertheless, it was possible to ask someone with access to that data set to provide the total time reported for 5 different projects that was arbitrarily selected. Out of these projects, 4 had the same

reported time in both of the data sets. One project had a much higher number of hours in the other data set compared to the one accessible (approximately 12000 hours vs 7700). The higher number was confirmed to be more accurate by a project manager, and also from looking at the estimated time to complete the project (around 9000 hours). For this project, the inaccurate value was replaced with the more accurate one from the other data set.

Two other entries were suspected to be invalid due to being highly implausible. The estimated time for one of the projects according to the project plan was approximately 5600 hours, but the number of hours logged was only 575. A project being completed in almost a tenth of the expected time is not possible. The other project only had a total of 24 hours logged, far too little for what is expected from a project. Thus, the data set provided may contain multiple entries with a high amount of error, and this fact must be taken into account before drawing any conclusions.

3.3 Typical project structure

Once the cross-referencing was finished, a list of all employees found in any of the selected projects were generated. An employee at the company helped match each employee in the list to which department they worked at. Unfortunately, the department for some of the employees were not known, and these were instead listed as 'Unknown'. The following categorizations were used:

- Software
- Hardware &A rchitecture
- Chemistry
- Project Management
- Verification & Validation
- Unknown

The general project structure tend to look the same, even though the details and scope may differ a lot. Each project has a software and/or hardware part. Often it has both of them, but there exists projects which are mostly focused on software or mostly on hardware. In addition, some project contains a chemistry part, however it tends to be only a small proportion of the whole project. These parts can be considered the "*core*" of the project, they are what defines the projects. However, there are always two other main parts to a project: project management and verification & validation. Every project has managers coordinating the work, and every project has a team tasked with testing and making sure the finished product or feature works as intended.

3.4 Relation between the different parts

Since the amount of hours spent per person per project is known, and also which department (and thus which part of the project) the person is associated with, calculating the proportion of time each part takes is trivial. This information can provide additional

insight and may help in identifying certain patterns. In particular, there are two questions which could be of great use if answered:

- Is there any part that takes roughly the same proportion of time no matter the project?
- Is there any part whose time largely depend on how long other parts take?

It is expected that the core parts of the project (software, hardware, chemistry) to differ a lot between projects, but what about project management and V&V? It is not unreasonable that the answer is yes to either question for these project. If so, it would make estimating them much easier. If a part takes a similar proportion of time no matter the project, then all that needs to be done is to estimate it to that proportion of time in future projects. On the other hand, if a part heavily depends on other parts, then the relationship can be identified and used to estimate that part.

3.5 Feature selection

All 12 project had a project plan or time plan. Often there were multiple versions of the same project plan, and in this case, the latest one were used when selecting features. Almost every project plan was saved as an Excel file, a template used between all projects. The first page in that file was a project charter. The fields that may contain relevant information in the project charter are *Background / Idea* and *Project objectives*.

The project plan excel template also had a work breakdown structure. Since the goal of a WBS is to break down a project into smaller parts, it may prove useful in extracting useful features, assuming of course that the WBS is detailed enough.

Finally, the template also contained a Gantt chart. Since the Gantt chart is supposed to list the various parts of a project, some of the listed parts may be possible to use as features.

3.6 Linear regression

As mentioned in Section 2.5.1, linear regression is good candidate for time estimation, since the time to finish a project is simply the sum of the time it takes for each part. Mathematically, the predictive model is expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... \tag{3.1}$$

Where y is estimated time for the entire project, β_0 some constant, x_i are the parts of the project and β_i are the coefficient associated with each part. The value of x_i is the amount of times the i-th task occur in the project, which would be 0 if the project does not include said task. The interpretation of this model becomes very simple, where β_i is simply the amount of hours the i-th task adds to the project. Aside from providing a time estimation for the entire project, it makes it trivial to obtain an estimate for how long each task in a project takes, which could offer valuable insight to the company.

3.7 Model training

The model was trained in Python using the *scikit-learn* library (sklearn). The input variables X and output variables Y were input into a *numpy* array (the low sample size made it faster to just manually input the values). The variable x here is 1 if a specific feature is present in the project (in this case the project has two possible features). The variable y is the reported time a project took to complete in hours.

```
X = np.array([
    [1,0],
    [1,0],
    [1,1],
    [1,0],
    [1,0],
    [1,0],
    [1,0],
    [0,1],
    [1,1],
    [0,1]])
y = np.array([3698.53, 9773.8, 575.11, 2349.3, 693.3, 7741.59, 568.49, 1634.42, 10270.95, 6744.87])
```

Sklearn can perform linear regression, and it has to be set up as such:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
mse_scores = []  # List to store mean squared errors
mape_scores = []  # List to store mean absolute percentage errors

```

Here the linear regression model is initialized, in addition to two lists that store the error metrics (MSE and MAPE) for each iteration. Sklearn can also perform Leave-one-out cross validation as such:

```
from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()

for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

For each split of the data, the model is trained on the training data. The trained model then predicts the test sample and calculates the error metrics by comparing the predicted value to the real value.

```
    # Fit the model on training data
    model.fit(X_train, y_train)

    # Predict on the test sample
    y_pred = model.predict(X_test)

    # Calculate mean squared error
    mse = mean_squared_error(y_test, y_pred)
    mape = mean_absolute_percentage_error(y_test, y_pred)
    mse_scores.append(mse)
    mape_scores.append(mape)
```

Finally, once all splits of the data has been tested, the average of the error metrics are calculated.

```
average_mse = np.mean(mse_scores)
average_mape = np.mean(mape_scores)
```

4 Results and Discussion

4.1 Comparison of human time estimates

Of all included projects, 10 had time estimates in man-hours, but one was excluded for being improbable (taking approximately only 10% of the estimated time). Using these values and comparing them to the actual logged hours for each project, it is possible get an idea on how reliable human time estimates are in this context. The total hours logged per project was 155.1%(±70.8%) of the estimated time. The best case took 79.5% of the estimated time and the worst case took 316.0%. There was no correlation between the number of hours in the initial estimate and the accuracy of the estimate, which can be interpreted as the company being equally bad at estimating large and small projects.

4.2 Proportion of time per part

The projects were divided into 5 parts: 1) Project management, 2) Software, 3) Hardware & Architecture, 4) Chemistry and 5) Verification & Validation (V&V) based on hours logged per employee. The data for the projects are shown in Table 4.1 in which values in *Percentage of hours* are presented as mean (± standard deviation). Note that 4 of the projects had employees without a known department, their time is presented under *Unknown*. For 2 of these projects, the unknown proportion was small (0.9% and 0.1%), but for the other two it was higher (9.8% and 10.9%).

Part	Percentage of hours	Lowest	Highest
Project Management	11.6%(±4.1%)	5.4%	19.6%
Software	44.6%(±31.2%)	0.5%	82.5%
Hardware	24.3%(±26.4%)	0%	73.8%
Chemistry	2.8%(±3.8%)	0%	10.4%
V&V	16.6%(±10.9%)	1.6%	33.7%
Unknown	1.8%(±4.0%)	0%	10.8%

Table 4.1: Proportion of man-hours on each part of a project

Out of the 12 selected projects, Software took the largest proportion of time, 44.6%, but it differed a lot between the various projects. Project management remained within a fairly narrow range, 5.4% to 19.5% of the total time spent, with the mean being 11.6%. It may thus be possible to model time spent on project management by assuming that the time spent will always be 11.6%. This would simplify the model by removing one variable, and may provide some insight for the company in regards to how long project management usually takes. On the other hand, the proportion of time spent on project management did still vary quite a bit, just not nearly as much as the other parts of the project. An alternative approach of modeling this is to assume project management time depends on what other parts are included in the project. It is not unreasonable to think that it would take more time to manage the hardware part of the project than V&V, for example. This approach can be modelled using linear regression:

$$y = a_1x_1 + a_2x_2 + ... + a_nx_n \tag{4.1}$$

Where y is hours spent on project management, $x_{1...n}$ are time spent on the various other

parts of the project (Software, Hardware etc), and $a_{1...n}$ their coefficient. So for example, if x_1 is the number of hours spent on the software part of the project, a_1 ends up being the ratio of project management hours per software hours, i.e. $a_1 = 0.1$ means 10 hours of software leads to 1 hours of project management. This approach can offer additional insight compared to the simpler model above.

Linear regression was performed on the selected projects, however the 4 project with a proportion of time being unknown were excluded to not skew the results. The coefficients obtained using the remaining 8 projects are shown in Table 4.2.

Part	Coefficient	Standard Error	95% Confidence Interval
Software	0.0995	0.008	$0.078 - 0.121$
Hardware	0.3711	0.065	$0.192 - 0.550$
Chemistry	-0.5073	0.320	$-1.396 - 0.381$
V&V	0.0568	0.014	$0.017 - 0.096$

Table 4.2: Coefficients for project management as dependent variable

What immediately stands out is that the coefficient for chemistry is negative, meaning more hours spent on chemistry equals less hours on project management, which intuitively does not make sense at all. It is not statistically significant though, as evidenced by the large confidence interval, so it is likely a result of the low sample size. A crude way to resolve this issue was to merge hardware and chemistry into one category, given that the two are somewhat related. The new regression was then repeated with the new variables. The new coefficients are given in Table 4.3.

Part	Coefficient	Standard Error	95% Confidence Interval
Software	0.1097	0.009	$0.087 - 0.132$
Hardware & Chemistry	0.2305	0.030	$0.154 - 0.307$
V&V	0.0510	0.019	$0.002 - 0.100$

Table 4.3: Coefficients for project management as dependent variable, with hardware and chemistry merged

These coefficients look better, and they all end up statistically significant. The model ends up having $R^2 = 0.997$, which signifies a very good fit. From this regression, it looks like Hardware & Chemistry takes roughly twice as much management time as software, and V&V takes roughly half the time.

The question is now whether or not it is possible to do the same for V&V. Unfortunately, V&V spans a much larger range, 1.6% to 33.7% of the total time, so it is expected that the model will not work as well. Indeed, running linear regression on Software and Hardware & Chemistry as independent variables yields the coefficients given in Table 4.4:

Part	Coefficient	Standard Error	95% Confidence Interval
Software	0.3977	0.090	$0.179 - 0.617$
Hardware & Chemistry	0.0908	0.630	$-1.450 - 1.632$

Table 4.4: Coefficients for V&V as dependent variable

The standard error ends up being very large for Hardware & Chemistry, suggesting that this approach is a poor way to model hours spent on V&V. Instead, it ought to be treated as an independent part of the project.

4.3 Project data and feature selection

4.3.1 Project charter

Out of the 12 projects, 10 of them had a project charter. However, only 3 of them had a description that made it clear what the project was about, the remaining 7 only had a list of goals (often features to be implemented). Most goals listed were unique for each project, which would make them ill-suited to use as features. In order to create a good model, features needs to appear in multiple projects, otherwise the model will not generalize well to new projects.

4.3.2 Work Breakdown Structure

A work breakdown structure (WBS) was found for 8 of the projects. None of them were very detailed, most often it was just one node for each major feature to be made (often similar to the goals listed in the project preview). In some other cases, the WBS consisted of one node per major part of the project, such as Software, Firmware, Electrical Engineering, Mechanical Engineering, etc. In 1 project, the WBS had very little detail, consisting only of two nodes: Software and Documentation.

Similar to the project description, extracting features to use in the model from the WBS proves difficult. If the WBS only lists things specific to that project, using these as features means that features will generally be unique to one project, and the model will not generalize well to new projects. On the other hand, if trying to extract features from the WBS that contains the major parts of the project instead, the features may end up too general. That is to say, many features may end up in most projects. Also, having one feature for something like *Software* means that the model is going to consider *Software* to always take the same amount of time, which is clearly not the case. Nevertheless, this crude model will be investigated later in the report.

4.3.3 Gantt Chart

All 12 projects had a Gantt chart of varying level of detail. Unfortunately, 6 had very low level of detail, often just the various phases of the project (i.e. *Sprint 1*, *Sprint 2* and so on). This information adds nothing of value as it does not describe anything unique to the project. The other 6 projects typically had a lot of information, however it ended up having the same problem as the project description and work breakdown structure with most features being unique to that project.

4.4 Predictions in different application cases

4.4.1 Application case 1

An easy way to approach modeling in general is to start simple and then add complexity with every iteration. Thus, this application of the linear regression model is simplest possible and is expected to be the least accurate. It serves as a good starting point and a sort of sanity check. No features were used, and the constant term β_0 is set to the mean completion time of the sample projects. Simply put, it always estimates that the project is going to take the same amount of time no matter what features it has, and the estimated

time is simply equal to the mean of all projects in the training set.

This gives the following equation:

$$y = t_{mean} \tag{4.2}$$

where t_{mean} is the mean time of the whole project among all projects used in the training set.

This model has a Root Mean Squared Error(RMSE) of 5058 hours, and a Mean Absolute Percentage Error (MAPE) of 282%. Compared to human estimates which had a mean percentage error of 55.1%, this model is very inaccurate, but can serve as a starting point, because any future model should be an improvement of this one.

4.4.2 Application case 2

The the next application of the linear regression model is to use the parts as defined in Section 4.2 as features i.e *Software, Hardware & Chemistry, Project Management* and *V&V*. However, every project has a V&V part, so it can not be used as a feature. Similarly, it was determined in that section that project management can be removed as a feature since it is largely dependent on the other parts of the project. This gives the following equation to fit:

$$y = \beta_{sw} x_{sw} + \beta_{hw} x_{hw} + \beta_0 \tag{4.3}$$

Applying linear regression to fit the data to the model gives the following coefficients:

$$y = 1233.39 x_{sw,1} + 1285.5283 x_{hw} + 2904.12 \tag{4.4}$$

The interpretation of this equation is that if the project has a software part, that part adds 1233.39 hours to the project, and a hardware part adds 1285.53 hours. The constant term of 2904.12 hours would include V&V, but also include some hours spent on software and hardware, as even project that did not have those parts still had some hours logged on SW/HW for various reasons.

Using Leave-One-Out Cross-Validation to test this model, the RMSE is 5873.7h and the MAPE is 374.9%. It has $R^2 = 0.019$, i.e it captures only about 1.9% of the variance. That it ends up being very inaccurate is not too surprising, given that it assumes every part takes a constant time to complete, which clearly is not the case. What is surprising is that it is actually performing worse than the simpler case.

4.4.3 Application case 3

An alternative approach is to use linear regression to determine the coefficients separately. Since the number of hour each part took, it is possible to use that as the output variable to determine how long a part takes. For example:

$$t_{sw} = \beta_s w x_{sw} + \beta_{sw,0} \tag{4.5}$$

As in the previous application, $\beta_s w$ is the time in hours that software adds to the project, and x_{sw} is 1 if the project has a software part, otherwise 0. $\beta_{sw,0}$ is the number of software hours that will be present in the project no matter if the project has a software part or not.

The time for the entire project becomes:

$$= m_{sw}(\beta_{sw} x_{sw} + \beta_{sw,0}) + m_{hw}(\beta_{hw} x_{hw} + \beta_{hw,0}) + m_{vv}\beta_{vv} \tag{4.6}$$

Here m is a multiplier that forces the model to take into account the extra time from project management. The values for m can be obtained from Table 4.3 in Section 4.2. Since V&V is always present in a project, it is included in the constant $\beta_v v$, which is simply the mean time V&V adds to a project.

Out of the 12 projects, 4 of them had time spent on unknown parts, which may skew the results if included. However, two of them had a fairly small proportion of time spent on unknown parts (0.9% and 0.1%) and were included anyway (the unknown time was divided into the other parts at the same ratio as for the known time).

Linear regression on each part separately grants $\beta_{sw} = 2561.93$, $\beta_{sw,0} = 343.31$, $\beta_{hw} = 1334.98$ and $\beta_{hw,0} = 147.68$. Taking the mean value of hours logged on V&V gives $\beta_{vv} = 1143.46$. For project management, $m_{sw} = 1.1097$, $m_{hw} = 1.2305$ and $m_{vv} = 1.0510$. Thus, the final equation looks like:

$$y = 2842.97 x_{sw} + 1642.69 x_{hw} + 1764.47 \tag{4.7}$$

Of course, the end result is similar to case 2, but with different coefficients. In fact, the difference is surprisingly big for software coefficient and the constant. But does this application end up being more accurate? RMSE ends up at the RMSE is 4269h and the MAPE is 320%. Compared to case 2 it is straight up better. Compared to case 1 the RMSE is better but the MAPE is worse.

4.4.4 Future application cases

Due to the scarcity of useful data, no further model was tested. Since neither of application case 2 and 3 managed to perform better than the simple model where $y = t_{mean}$, it is unlikely that it is possible to come up with a better model than those presented here without access to more data. Some tweaking may make it produce somewhat better results, but it is highly improbable that it will ever produce results that are good enough to be of any use.

5 Conclusions and future work

In order to make a good predictive model, it must contain features that appear multiple times in the various samples, but not in every sample. In this case it proved very difficult to find features that appeared more than once in total. This is not surprising given the nature of the type of projects that Biotage works with, which tends to vary widely. This issue was made worse by the fact that the sample size ended up being very small with only 12 projects to work with, which is not a lot given how large each project is.

During the time at Biotage, a glimpse of how they did time estimation on the software side of projects was shown. In general, they broke down the project into fairly small parts of what needed to be done, such as *make a GUI window* or *make a database table*. From experience, they would have an idea on how long each of these parts would take. In addition, they would add parts such as bug fixing and code refactoring, knowing that these almost always occurred. In fact, the software department was being praised for delivering accurate time estimates (at the very least compared to other departments). Unfortunately, this kind of detailed information about each project was not available for this thesis. Had it been, it is very possible that a much more accurate model could have been developed.

One could imagine scenarios where this type of AI assisted time estimation could work much better. For example, a restaurant might make easier use of it. A "project" in the case of a restaurant would simply be one order from a customer. It would be very easy to divide the order into parts, as each part would simply be an item on the menu. Since a restaurant typically serves many customers per day, it would not take long to build up a large sample size to train the model on.

To summarize, there were three main issues preventing the creation of an accurate AI for time estimation. The first one (and likely the biggest one) being the level of detail available in the sample projects. It must be possible to divide every project into sufficiently small parts, and with the data that was provided, it simply was not for most projects. The second reason is that the projects were not similar enough. It is not possible to find useful features when most features shows up in one project at most. The third and final reason is that the sample size was very small, containing only 12 projects. It is suspected that the first issue alone would've made it impossible to fulfill the goal of this thesis. It is possible however that the second and the third issue may only be problematic when they both occur at the same time. Projects being very dissimilar may not be as big of a problem if the sample size of previous projects is large, as useful features are bound to appear multiple times anyway. Conversely, if the sample size is small but project are very similar, useful features are likely to appear due to the similarity of the projects despite the low sample size.

6 References

[1] M. R. R. Buehler, D. Griffin, "Exploring the 'planning fallacy': Why people underestimate their task completion times," *APA Psycnet*, 1994. [Online]. Available: https://psycnet.apa.org/doiLanding?doi=10.1037%2F0022-3514.67.3.366

[2] D. G. Roger Buehler and M. Ross, "It's about time: Optimistic predictions in work and love," *European Review of Social Psychology*, vol. 6, no. 1, pp. 1–32, 1995. [Online]. Available: https://doi.org/10.1080/14792779343000112

[3] Biotage. About biotage. Accessed on December 22, 2023. [Online]. Available: https://www.biotage.com/this-is-biotage

[4] K. Inc., "What is a project in project management?" [Online]. Available: https://kissflow.com/project/what-is-a-project/

[5] V. Kanade. (2022) What is linear regression? types, equation, examples, and best practices for 2022. Accessed on December 22, 2023. [Online]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/

[6] R. Patro. (2021) Cross-validation: K fold vs monte carlo. Accessed on December 22, 2023. [Online]. Available: https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b

[7] A. M. Molinaro, R. Simon, and R. M. Pfeiffer, "Prediction error estimation: a comparison of resampling methods," *Bioinformatics*, vol. 21, no. 15, pp. 3301–3307, 05 2005. [Online]. Available: https://doi.org/10.1093/bioinformatics/bti499

[8] ProjectManager. Work breakdown structure (wbs). Accessed on December 22, 2023. [Online]. Available: https://www.projectmanager.com/guides/work-breakdown-structure

[9] P. M. Institute, *A guide to the project management body of knowledge (PMBOK guide)*, 2021.

[10] ProductPlan. Gantt chart. Accessed on December 22, 2023. [Online]. Available: https://www.productplan.com/glossary/gantt-chart/