Explainable Artificial Intelligence

How to increase interpretability through object-oriented programming and dashboard technology

June 9, 2021

Andreas Work - 66712
Emil Sejer Pedersen - 66822
Frederikke Rønne Westergaard - 66426
Marie M Riskær Povlsen - 66708
Lucas Kolding - 66824

Group number: S2125625149

Supervisor: Jens Ulrik Hansen

GitHub Repository:
`https://github.com/mriskaer/BachelorS2125625149`

Abstract

This paper seeks to investigate the field of artificial intelligence, more specifically, the explainability of machine learning models. The problem we attempt to solve is how to create an object-oriented infrastructure, and how to achieve interpretability through dashboard technology. Since machine learning is a growing field within artificial intelligence, we found it interesting to explore how to understand and convey machine learning models. Methodologies used in this project to explore our problem includes CRISP-DM, Data pre-processing, UML, LIME, object-oriented programming and regressions. The analysis explores our code using the TRIN-model. Furthermore, the object-oriented approach is analyzed and lastly the explainability gained from the dashboard technology is examined. We can conclude that our solution did increase the explainability of the machine learning models. However, as our development ceased in the CRISP-DM modelling phase, we did not reach a point where we could evaluate whether or not our solution reached a sufficient level of explainability. Furthermore, we conclude that the object-oriented approach has its applications in certain situations, but like any methodology, it should be chosen carefully.

Contents

1 Introduction

The world is in a state of continuous technological development, especially computer science seems to be exceeding the expectations we have for it on a daily basis. The first electronic digital computer was developed in 1937, and since then, the computer technology have come a long way (Gregersen, n.d.). Artificial intelligence (AI), dates back to the 1950s, but it was in 1956 it really had its breakthrough. At the historical conference Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI), it became clear to many - who later became leading researchers in the field - that AI was an obtainable and possible achievement (Anyoha, 2017). This conference is said to be groundbreaking for the next 20 years of AI technology, where the goals were set high. Funding was obtained for the ambitious and expensive venture of AI through agencies such as Defense Advanced Research Projects Agency (DARPA) (ibid.).

A branch in the field of AI is machine learning (ML), which focuses on the use of statistical methods, algorithms and data, with the goal of imitating how humans learn (IBM Cloud Education, 2021). ML is widely used today, and the demand for developing technologies that can replace tasks that require a human presence, is steadily increasing.

One of the issues with AI, and the implementation of ML, is that people tend to have a hard time trusting these technologies. A classic example of this, is self-driving cars. The ethical questions called "the trolley problem", has been raging for a long time now. The trolley problem is stylized ethical dilemmas, where a person for example has to choose between sacrificing one person, to save more people. This question has been greatly discussed in connection with the talk of self-driving cars, since people do not understand how they work, and thereby does not trust their judgement (Dierker, 2019).
In order to avoid distrust and misunderstanding regarding this new technology, it is important for people to understand the AI. This is where the field *explainable AI* becomes relevant, since an interpretable and understandable explanation of the technological innovations, will help in building trust between people and technology (ibid.).

The seed for this project was planted by our own lack of understanding within the field of ML and AI, which sprouted into a mission of developing a solution that could help people better understand ML-models.

1.1 Problem field

The market of AI has gone from being non-existent in the 1950's, to a market worth 327,5 billion US dollars in 2021. While the market has expanded exponentially, it does not seem that the explainability solutions has followed. The lack of solutions, is creating an increased demand (Liu, 2021). This problem is being partially solved using explainability models. The development of explain-

ability models is still a rising field, and the methods for developing within this field is still being shaped. Based on our background in software development which has a focus on object-oriented programming (OOP), we were curious as to what would happen, if we took an object-oriented approach to developing ML explainability. Our experience in the field was also limited and we found it challenging and exciting, which fueled a desire to dive deeper into the subject. This lead to the formulation of our problem statement:

1.2 Problem statement

How can we build an object-oriented infrastructure when developing explainable machine learning models, and how can we increase interpretability, through the use of dashboard technology?

1.3 Delimitation

This study is primarily focused on the object-oriented structural approach and how to create interpretability of ML-models, through the use of dashboard technology.
The initial focus and problem we were interested in, was about visualizing IMDb (Internet Movie database) movie data (IMDb, n.d).
We quickly found out, that we wanted to change the direction, and focus on the explainability of the ML-models, and the creation of an object-oriented infrastructure instead. This led to a focus, that has resulted in a program, which will be able to take any given dataset and visualize it through a dashboard. The reason explainable AI piqued our interest, was that we found it interesting to examine the importance of understanding the outcome of a given model. This all resulted in a project concerning both explainable AI by including them in a dashboard, and an object-oriented approach.

The development process of this project - both writing this paper and the development of the code - can be seen as a learning process. Since we have been changing direction throughout this process, and had an iterative approach, we have increased our knowledge regarding ML. Therefore we chose to not have any specific work-related questions to follow.

1.3.1 Target audience

The product of this project has been to build a dashboard that offers increased interpretability, by visualizing and explaining ML-models. As will be accounted for in section 3.3.1, *trust* is a necessity to the adoption of a technology. Therefore, the program could be useful for someone who needs to understand a dataset, and the decisions an ML-model makes, in order to gain trust in that technology.

To summarize, people that would make use of our program, is people that need to understand ML-models, in context to a specific dataset, that can be cleaned using the program. The specific ML-models will be elaborated on in section 4.6.

In our project, we use two IMDb datasets, in order to predict how different characteristics of a movie will affect the ratings it will get. The first dataset is about the specific movies from the database, with information about directors, actors, and more (Leone, 2020a). The second dataset is a log of the amount of reviews, and how many stars each movie has gotten, and from who (Leone, 2020b).
In this context the program puts the IMDb data through the ML-models multiple linear regression and random forest regression. It explains how the data is used in these models based on the cleaning and prediction criteria set by a user.

2 Semester bindings

This project includes all the three HumTek dimensions; *Technological Systems and Artefacts*, *Science and Technology Studies* and *Design & Construction*. In the following section, we will account for these three dimensions, and how they individually contribute with different perspectives to the project.

2.1 Technological Systems and Artefacts

This dimension introduces theories and concepts, that can be used to analyze and understand technological innovations. A tool used to analyze innovative technologies is the "TRIN-model", which seeks to split the technology into different steps, in order to go in depth with every aspect of it (Study RUC, 2018c).

2.1.1 TRIN-model

The TRIN-model is a tool used to describe an arbitrary technology through an analytical approach. It highlights the technical and scientific aspects of the technology and focuses on the design, production, structure, and usage of the technology being analyzed. The model is split into six steps, which each focus on different parts of the technology.

Step 1: Inner mechanisms and processes of the technology
In the first step, we identify and analyze the technology's inner mechanisms and processes. The point of doing this is to identify the purpose of the technology's functionality. By defining the purpose, it provides knowledge of the problem and thereby makes it possible to fully understand and work with the technology (Jelsøe, 2018c).
All the inner mechanisms and processes within a technology, is actually the parts that defines the technology. These needs to be examined and analyzed, in order to understand the purpose of the technology (Jørgensen, 2018a).

Step 2: Technological artefacts
In the second step, we take a closer look at the technology's artefacts, to better understand the purpose of the technology, like in step 1. The difference is that in this step, we focus on the technological artefacts, as *physical anthropogenic objects*. These are man-made objects, with a technological purpose.
A technology consist of different artefacts, and when these are combined, they form the whole technology (Jelsøe, 2018b). The purpose of this step, is to go through all the artefacts, that belongs to the technology. By identifying these, we end up with a better understanding of the technology, and the intention behind it (Jørgensen, 2018b).

Step 3: Unintended effects of the technology
In this step, we identify the unintended effects caused by the technology. An unintended effect can either be positive and negative, but in this step the focus is mainly on the negative consequences of the implementation of a new technology. These unintended effects can both be a technological effect, maybe based on the inner mechanisms, or the unintended effect can be user-oriented and happen because of the way the user interacts with the technology.
It is important to have specified the intention of the technology in step 1 and 2, since the purpose of creating a new technology is very important to know, in order to find out whether or not the implementation is worth the unintended effects. The effects can be separated into two categories, the *uncertain effects* and the *lasting effects*. The uncertain effects are the ones that only occur, in case of an unfortunate event or situation, whereas the lasting effects have a constant negative effect on someone or something (Jørgensen, 2018a).

Step 4: Connections in bigger technological systems
In the fourth step, we analyze the connections of the technological artefacts, in a bigger technological system. When the artefacts get combined in a larger system, it is because they have a purpose to fulfil an overall functionality.
This larger system could for example be the society. In order for a technology to be used and recognized, it needs to be approved in the society it exists in.
The systems functionality consists of an *input*, a *process*, and an *output*, which in the end results in the transformation required for the technology to be a part of the bigger system (Jelsøe, 2018c).

Step 5: Create a model of the technology
In the fifth step, we create one or more models that represents different aspects of the technology's features, in order to understand them better. These models can be used to explain parts of the technology, or to learn more about the usage and development of the feature focused on. Before creating the model, it is important to establish which part of the technology you want to examine, since there are many different ways to build the model. A model can be in the form of a physical formation of the technology, or it can also be a UML-diagram. It depends on whether you want to visualize the appearance, the functionality, or something else (Christensen, 2018b).

Step 6: Technology as an innovation
In the last step of the TRIN-model, we analyze the driving forces and barriers of the dispersion of the innovation. We look at the circumstances that have caused the need for the specific technology, and how different barriers can prevent the technology from growing and evolving. A technology is defined as an innovation, if it is either a completely new product, or a significantly improved version of an already existing product. It is therefore important to find out whether or not the technology is an innovation, and if so, how to prevent possible barriers, or assist the dispersion, if needed (Christensen, 2018a).

2.2 Science and Technology Studies

This dimension focuses on the relation between technology and people, and how these compliment each other's development. It deals with how technologies affect the evolution of a society, and the world in general. All of this is taught with an interdisciplinary historical, social, cultural and environmental perspective. There is also a focus on different approaches to ethical analysis of technological issues, primarily in regards to how it is incorporated in the society. Additionally, this dimension has a great focus on how to gather knowledge and material from different informants through different methods (Study RUC, 2018b).

2.3 Design and Construction

This dimension is rooted in a design-scientific approach, and focuses on both the development and evaluation of different systems. It draws on design and architecture theories and concepts, as well as scientific theoretical issues within the design-field. This dimension teaches how to manage design processes, as well as how to support and organize them. The dimension includes analysis of needs, modeling, evaluation and risk analysis, as well as visual, auditory and aesthetic properties of systems and artefacts, with the purpose of influencing the users perception and cognitive processes (Study RUC, 2018a).

3 Theory

In this section we will account for the philosophy of computer science, which is our scientific theoretical standpoint. We will also account for what artificial intelligence and machine learning actually is. Furthermore, we will describe what interpretability is, and why it is important in relation to machine learning.

3.1 Philosophy of computer science

Philosophy of computer science contemplates the ontology and epistemology of computational systems. Philosophy of computer science involve the philosophical questions that might emerge, when engaging with the academic discipline

computer science. The philosophical questions emphasises the problems related to the specification, programming, methodology, design, implementation, verification and testing of systems (Angius et al., 2021). Since it can be argued that data science is a subject within the field of computer science, there also seems to be some inconsistencies within that definition as we found while writing this paper.

One approach to the philosophy of computer science is through *abstractions*. Abstraction is the process in computer science, where you only show relevant data to the user, and hide extraneous data for the user. This will be explained further in section 4.5. This approach is highly relevant to this project due to our use of object-oriented programming, which is based on building software that allows abstraction.
One way of perceiving the philosophy of computer science could be as a mathematical discipline. It could be argued that computer science differs from mathematics through its approach to abstraction. Where mathematics seeks to abstract endlessly to a point where the abstraction only takes meaning from the context of which it is part of, computer science must leave behind an implementation trace. A mathematician might abstract to a point where the formula only become relevant through its context, a specific formula is often an abstraction of something that came before it, but in mathematics we remove that trace. In computer science the trace always exist, but it is hidden (ibid.).
To summarize, abstractions in relation to computer science can be said to hide its point of abstraction where mathematics destroy its point of abstraction (ibid).

OOP can be seen as an abstraction in the same way that machine code is seen as an abstraction of a higher-level programming language. The Euclidean distance between two points in a GPS system is an abstraction from a method that calculates the Euclidean distance, and a graphical representation would be yet another abstraction. In that way, OOP can be seen as a way of building systems that allow for abstractions to a higher level (ibid).

Where does that leave data science? Data science can be seen as a field within computer science. However, often data science products, while being a result of abstraction processes, are not built for further computational abstraction. Systems are built in ways that often returns a model, or a visualization that can be extracted, but the systems themselves do not take abstraction into account through OOP (ibid.).
As an example, many resources for learning data science and exploring the field are script-based. These systems often only have one function, and are not build for abstraction in the same way that an object-oriented system would be.

We do not see our project as a mathematical discipline, because the goal does not lie in proving the mathematical correctness of the system. Our approach lies more within the field of the engineering discipline. Instead of asking the

question of 'what exists', we ask the question of 'what can exist'. Thereby we have a focus on how to produce the specific artefact, and within that artefact a focus on developing a product that is build on abstracting instead of one-time script use. In this manner, the field data science might be considered more of an engineering discipline, because the way we evaluate the system is by whether it lives up to the requirements it set out to meet (ibid.).

3.2 What is machine learning?

In this section, we will explain what artificial intelligence (AI) and machine learning (ML) is, and how these two parts of computer science are connected. In addition to this, we will discuss how an ML-model can be designed. Our knowledge in this section, is based on the book *Deep learning with Python* (Collet, 2017).

3.2.1 What is artificial intelligence and machine learning?

We have known AI since the 1950s, as that part of computer science that deals with developing algorithms that can solve intellectual and logical tasks, usually solved by human beings (Collet, 2017). AI however, is not capable of solving complex problems, that goes beyond the rules defined by a computer scientist. Another branch of AI is called machine learning. It is capable of solving more complex problems, and finding patterns, with minimum human intervention (ibid.).
ML-models are trained to learn data-processing rules by themselves, by looking at data. It is no longer a computer scientist that is crafting data-processing rules, but instead the solutions are given to the ML-models which come up with the rules itself (ibid). This concept is illustrated in **figure 1** below:

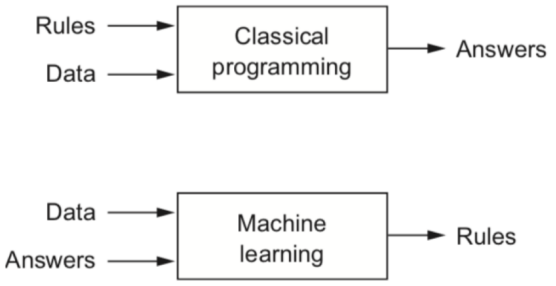

Figure 1: Classical programming vs. machine learning (ibid.)

As mentioned earlier, ML-models are trained, more than they are actually programmed to solve tasks. As **figure 1** illustrates, ML-models are exposed to data and answers, and they will find patterns and structure in these, in order to be able to solve and find the answers itself.

3.2.2 What is the purpose of machine learning?

ML has now become one of the most popular and used subfields of AI, when working with fast hardware and big datasets (Collet, 2017). There are three key aspects attached to ML, and why it is needed: *Automation*, *generality* and *effectiveness* (Lu, 2020). ML is especially relevant if you want to automate a process. ML-models are also able to deal with not only one - but various datasets, and therefore, ML is great to use in generality. Big datasets can be overwhelming, and the structure and patterns can go way beyond what we humans can derive just from looking at the data. ML-models helps with making this process more efficient.

ML is used in various applications and platforms that we use in our daily lives. An example could be tagged pictures on social media platforms, like Facebook. With the help of an ML-model, the system will learn to recognize facial-features, and it will make it possible to automate the task of tagging pictures posted to Facebook. This system will be exposed to many pictures tagged by humans, and then be able to learn the rules for when a picture should be tagged (Collet, 2017).

3.2.3 MLOps

Machine Learning Operationalization Management (MLOps) is an emerging field, and gaining momentum among data science companies (Visengeriyeva et al., n.d.).
MLOps is a process of getting ML-models into a production system. It is an end-to-end ML-process, which helps organisations, who are trying to implement AI or ML into their production system or platforms (ibid).
MLOps are designed for helping the data scientists and production team with for example minimal waste, and aiming to standardize automation for things like testing - both testing data and testing ML-models (ibid.).
In order to get to the point, where an ML-model can be put into production, you need to make sure that the model is approved to be ready. This might not only require that the model is approved in a technological sense, but also from a user perspective. This will be highlighted in the following section.

3.3 What is interpretability?

As mentioned in the beginning of the paper, we also found it relevant to take a look at ML with a user perspective, and examine what it takes, to make ML-models more understandable. In relation to this, we found it relevant to explore the term *AI-interpretability*.
The book we have used to gain knowledge about this field is *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable* by Christoph Molnar, 2021.

AI-interpretability in its essence means understanding the ML-model itself, or the outcome of the model. Interpretability is not a clearly defined concept, and several definitions exist. Among these are: *"Interpretability is the degree to which a human can understand the cause of a decision"*(Miller, 2017), or *"Interpretability is the degree to which a human can consistently predict the model's result."* (Kim et al., 2016).
A model has a high level of interpretability, if humans are able to comprehend and understand the decisions it makes (Molnar, 2021).

3.3.1 Why interpretability is important in relation to machine learning models

ML is a technology that is here to stay, and therefore, we need to be able to *explain* ML-models, so everyone can be able to understand it. We need this explanation for a few reasons. The three reasons we will emphasize are *trust*, *moral and ethics* and *debugging* (Molnar, 2021).

Trust plays a crucial role when it comes to adoption of new technologies. For people to trust the technological artefacts they use, they need to understand within a reasonable scope what that artefact does or is going to do (ibid.).
An example could be self-driving cars, as mentioned earlier. Another example could be a robot vacuum. It might decide to stop every time it reaches a specific surface, and if I as an end-user cannot recognize that it behaves that way due to the change in surface, I will not trust the robot vacuum to perform the task. However, if I am able to recognize why the vacuum behaves the way it does, I can predict its behavior when it comes to specific surfaces, and I will trust the product.

Another reason why interpretability is important in relation to ML-models lies within the area of morals and ethics. In order for a technology to be adopted by society, it needs to live up to the moral and ethical rules that are set by society. This is also essential, when working with ML-models, to avoid biased and discriminatory models (ibid.).

The third and last factor we will mention as an argument that ML interpretability is important, is debugging (ibid.).
As an example, a model could be misclassifying pictures of plants as cats. We need to understand why it is making this misclassification, in order to fix how we train the model, and to achieve interpretability. A way to fix this specific problem, could be by taking colors into account since cats are not green.

These are some example on why it is important to explain ML-models. In the next section, we want to look at some of the approaches to implement and achieve interpretability, when creating an ML-model.

3.3.2 Approaches to achieve interpretability

In this paper we will highlight two main approaches to explain an ML-model, in order to achieve interpretability. The first is *intrinsic interpretation*, which can be defined as achieving interpretability through restricting the complexity of the model. This means, that interpretability is achieved, by only looking at the model itself. The second is *post-hoc interpretation* which concerns achieving interpretability through an explanation that comes after training the model (Molnar, 2021). Post-hoc interpretability can be achieved by deploying a local proxy model (Wei & Landay, 2020).

If an ML-model uses intrinsic interpretability, it means that it has a simple structure. An example of this, could be a structure containing small decision trees. This model will always be *model specific*, because the interpretations use intrinsic values that are unique to the model. This means, that because of the simple model structure, intrinsically interpretable models are limited to specific model classes, and can not be used on any other models (ibid).
The opposite of model specific is *model-agnostic*. Model-agnostic tools are defined by the fact that they can be used on any ML-model. The interpretation in this model must be post-hoc, while a model-agnostic approach can not use intrinsic values (ibid.).

Now that we have clarified some of the different approaches to achieve interpretability, we want to account for theory about *when* it is actually relevant, to explain ML-models.

3.3.3 What is an explanation?

One way to perceive an explanation is as an answer to a question. When talking about explanations within the scope of ML we want to create 'everyday'-explanations, meaning explanations that does not require a large amount of technical knowledge, and can be used in everyday situations. Everyday situations and explanations are unique to the audience. What matters most is that the explanation fits within the scope of an everyday explanation, for the specific audience for which the explanation model is developed (ibid.).

Properties of a good explanation
Good explanations follow certain rules. In this theory, Molnar (2021) are listing five properties of what makes a good explanation. We will explore these properties, and their relation to ML, in the following section.

Human beings tend to think of questions in a *contrast* to others, which means, that everything is understood in relation to its surroundings.
In relation to an ML context, a good explanation does not tell us why the machine made the decision it did, but why it made the decision it did instead of another decision. This also helps humans understand features, and assists in

explaining how a features affected the decision. This also means that humans do not necessarily want complete explanations, humans want to know why and what features affected the decision that was made. With this in mind, it can be said that *good explanations are contrasting* (Molnar, 2021).

The *Rashomon effect* is the name of a theory that states that there are various ways by which an event can be explained. This is the same for ML. By acknowledging that a good explanation is chosen and is an abstractions of human intention, we acknowledge that there could have been millions of ways to get to this decision, so we should choose a small sample of those explanations that seem the most relevant to the model.
This also has roots in the philosophy of computer science, where a formulation of the levels of abstraction for ontology in computational systems has been devised by Primiero in 2016. The first step in this formulation is the intention, which is defined as the cognitive act that defines what computational problem is to be solved. The intention can be said to be an abstraction of that cognitive act (Angius et al., 2021). By this, it can be stated that *good explanations acknowledge that they are chosen* (Molnar, 2021).

By realizing that an explanation is an interaction between an explainer and an explainee, we acknowledge that there is an audience, and when we have an audience it is important that we are conscious of their technical knowledge. What this means for ML explanations is, that we should care about the audience we are communicating to, since *good explanations realize that they are interactions* (ibid).

A good explanation should be as truthful as possible, which means that it should focus on the important parts of the truth. It also means, that if a feature matters in one particular instance, it should also matter in others. By acknowledging that we are choosing explanations, we are also acknowledging that we are omitting part of the truth. Sometimes omitting some truth increases understandability, but truthfulness should still be what is strived for, since it is known, that *good explanations are truthful* (ibid).

The rule of explanation called *good explanations are consistent with prior belief* is hard to achieve in ML, often because the purpose of ML, can be seen as the exact opposite. They are build to find patterns and feature importance we as humans did not expect or realize could have an impact (Molnar, 2021).

Now that we have covered what makes a good explanation, we explore to what extent it is important to explain an ML-model.

3.3.4 The extent of the explanation

To what extent it is relevant to explain the ML-model, relies on a lot of factors. Molnar (2021) distinguishes between five different approaches finding the extent

of our explainability.

Algorithmic transparency explains the specifics of the algorithm. Algorithmic transparency and explanations is a topic on its own. It is rarely effective to provide algorithmic explanations when trying to understand ML-models.

Global holistic model interpretability is when the user understands the whole model. This includes the models features and weights. This is impossible to achieve for larger models, where some features might have interdependency.

Global model interpretability on a modular level attempts - like the holistic approach - to create understanding of how and why the model makes the decision it does, but this time on a modular level. It is a good way to zoom in on a model and create understanding. However, it is impossible on models such as trees due to the fact that the features are interconnected with each other.

Local interpretability for a single prediction zooms in on a single instance and examines the prediction for a specific input (ibid.).

Local interpretability for a group of predictions, is the opposite of local interpretability for a single prediction. It takes a group of instances and treats them as a full dataset using global methods on all subsets. The explanations can then be used individually or aggregated into an explanation for the entire group (ibid.).

In this project we focus on Local interpretability for a single prediction and Local interpretability for a group of predictions. We find it important to include explanations of all approaches, in order to understand the contrast there is between these approaches.

3.3.5 Properties of explanation methods

The unique thing about creating understanding in computer science is that we often rely on algorithms as our explanation methods to communicate the explanation to the end-user. In the following section we will be going through four of these explanation methods.

First and foremost there is the *expressive power*, which is either the language or structure of the explanation. This topic asks which methods and what language is used to explain the model, and furthermore whether that is the correct language or method to be used for that instance (ibid.).

Translucency describes how much the explanation we provide relies on intrinsic parameters. A linear regression is considered highly translucent since it is regarded as an intrinsic model (ibid.).

Portability describes the range of ML-models on which the explanation method can be used, whether or not the explanation method easily can be applied to other ML-models (ibid.).

Algorithmic complexity describes the computational complexity of the explanation method. It is mostly used for time sensitive ML where the complexity creates a bottleneck. An example could be having to produce animations of the model, this could cost a lot of computing power, slowing down the ML in favor of explainability (ibid.).

There are also properties to the individual explanations. Molnar (2021) accounts for multiple properties to individual explanations. We will only account for the property *accuracy* that we use in this project.

Accuracy describes how well the explanation predicts unseen data. If the model is highly accurate, then the explanation needs to be as well (ibid.).

Evaluation on the achieved interpretability
As with any technological artefact, it is necessary that we ask how the product can be evaluated.

We consider three approaches as described by Molnar (2021). The first is *application-level evaluation* in which the model would be given to an end-user to see if they can perceive the inner workings of the model. This requires an end-user, which is often an expert within the field of what the model is trying to predict. Secondly there is *human-level evaluation*, where instead of seeking out an expert, we might ask a layperson which explanation they find best for explaining the model, or even if they can understand the explanations at all. Thirdly there is *function-level evaluation*, which uses a proxy for the explanation quality. An example of this could be that the end-user understands trees, and on that basis it could be concluded that the depth of the tree can function as a proxy for how well it serves as an explanation (Molnar, 2021).

4 Methodology

The group as a whole have had limited exposure to machine learning (ML) prior to writing this project. Therefore, this project has been structured as an iterative learning process.
Throughout this project, we have been developing the explainable ML-models, and writing the report simultaneously.

In this section, we want to present some of the methodical approaches we have used in this project. Initially we want to present the CRISP-DM model, and describe how we have used this as a framework for our project process.
Afterwards, we want to introduce what data pre-processing is, and how we

used it, in order to prepare our data, so that we were able to analyse it, and create models based on it. Next we wanted to introduce the explanation-method LIME, and how it contributed to making our ML-model more interpretable. We also account for UML-diagrams, that is a modeling language used to explain the structure of our program. Finally we want to talk about what an object-oriented programming language is, and why we used this approach in our project.

4.1 CRISP-DM

CRISP-DM is short for *Cross Industry Standard Process for Data Mining*. Data mining is the process of finding correlations and patterns in large datasets, and for instance make predictions based on this. ML can be considered a data mining process (Wirth & Hipp, 2000). The length of a data mining process is divided into six phases, as illustrated in **figure 2**.
This section is based on the article *CRISP-DM: Towards a Standard Process Model for Data Mining* (ibid).

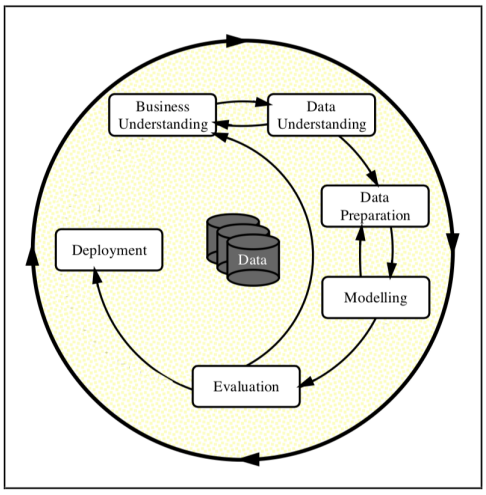

Figure 2: Phases in the CRISP-DM Process Model for Data Mining (ibid.).

It is important to note, that these phases do not need to be strictly followed, and it can vary from project to project, which phase to follow next (ibid.). In the following, we want to go through these steps, and account for how they work in a data mining process.

Business understanding
In this initial phase, the main focus is to account for what goals and requirements there are for this specific project. The project is looked at from a business perspective, in order to concretize the meaning of the project, and generate a project plan (ibid.). In this phase, you want to answer questions like; What kind of problems should the data mining model solve?

Data understanding
This step requires data collection. Here it is decided, which data to use, to create the data mining model. This step is essential in order to get to know and understand the data you decide to work with. These two first phases of the CRISP-DM model are closely related, since you cant get to know the business plan, without knowing the data (ibid.).

Data preparation
In this phase the selected data gets prepared. Datasets requires pre-processing, so it can be used to match its purpose. How we made use of data preparation, will be presented in section 4.2. (ibid).

Modeling
In this phase, the suitable model techniques that matches the intended outcome are selected and applied. There might be various kinds of model techniques for the same data mining problem, while the different types of techniques can express different aspects of the specific goal of a given project. (ibid.)

Evaluation
In this phase, there has been constructed one or more models that has an acceptable accuracy, in terms of the intended task. It is important to evaluate the models that have been constructed, to see if they match the business goals and requirement that were made in phase one and two. (ibid.)

Deployment
This part of the process is mainly seen from a user perspective. When the models have been evaluated, you need to acquaint yourself with what it takes for the model to be implemented in a bigger perspective. We will go more in detail with what this means, when we account for step 4 in the TRIN-model, in section 6.1.2.
In this phase, the models have been created, and it is now time to look at how to actually make use of the created models (ibid.).

CRISP-DM in our development process
This model works as a framework for our project process. We have used the CRISP-DM model as a method to describe and keep track of how to structure our development process. It has helped us to maintain a focus of what we should do next, in order to prevent coming to a halt in the development process.
Later in this paper, we will describe how far we have gone with the development process according to the CRISP-DM model. We also want to discuss which step would be our next (ibid.).

4.2 Data pre-processing

In this project we use the terms data cleaning and pre-processing interchangeably. Data pre-processing is the act of preparing data to be analyzed. For an

ML-algorithm to learn anything, it needs to be fed some data. Before feeding data to an ML-model, the data needs to be as "clean" as possible to get the best results. Data cleaning or pre-processing is done through multiple steps. It is important to note that the amount and methods used for data cleaning depends on the dataset that is worked on. There is no stringent methodology to follow. Therefore, the steps required to clean data differs from case to case. There are many ways of tackling data pre-processing (Techopedia, n.d). In this paper, we will only be explaining the steps that we have used in our project.

4.2.1 Data assessment

When looking at a dataset, one of the first steps is getting an overview of the data that we are working with. This means that we need to identify empty rows and decide whether we remove empty rows entirely, or we replace empty values. In our case, due to the size of our dataset, we deemed that removing rows containing empty cells was the wisest choice. Data assessment also handles wrongful entries, i.e values in a column that does not match the datatype of all other entries. In our case we check our columns and see if all values in the columns are numeric or not. If there are string values in the columns, we will drop those rows (Hutajulu, 2019).

4.2.2 Data reduction

Data reduction is the process of narrowing down the set of data to only contain relevant information. This is mainly done to increase the performance of the ML, as it does not have to deal with unused or unrelated data. The process could involve removing entire columns from a dataset (Hutajulu, 2019).
In the case of our project, we do reduce some of the data in our dataset. We do this by creating a function called keep_columns. We use this function in main, in order to make a keep_col_list .

Listing 1: Python code

```
keep_col_list = ['imdb_title_id', 'year', 'genre', 'duration',
    'country', 'total_votes', 'avg_vote']
```

4.2.3 One-hot Encoding

One-hot encoding is used to manipulate the dataset features into a language that can be fed to the ML-model. As ML-models only deal with numbers, we need to transform our columns containing string format values. In this project we use one-hot encoding to transform our data.

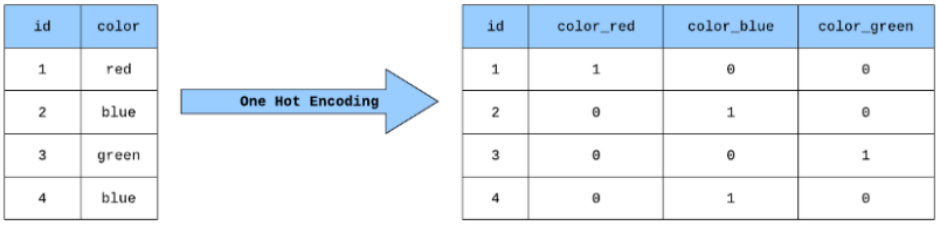

Figure 3: Example of one-hot encoding (Novak, 2020)

One-hot encoding takes a column of string values and separates all unique values into separate columns. If a value is then true, the cell will be marked 1 and if not true, the cell will be marked 0 (Gupta, 2019).

4.2.4 Data merging

Data merging can be used to fuse datasets together. All datasets, however, must have an identical column to merge on. In our case we merged two datasets together; our *ratings dataset* with our *movies dataset*, for the program to be able to handle the data more easily. The merge is done on a unique movie ID, which were present in both datasets (hutajulu, 2019).

Now that we have accounted for data pre-processing, we want to account for what kind of methods can used, to make ML-models more explainable.

4.3 Local surrogate models and LIME

When a given ML-model does not give an outcome that easily and directly can be measured, a method, such as *local surrogate models* can be used. Local surrogate models are interpretable models that are used to explain individual predictions of a model (Molnar, 2021). The LIME model does not measure the ML-model directly, but instead creates a surrogate model, which is a model that explains individual predictions. *Local Interpretable Model-agnostic Explanations* (LIME), uses a concrete implementation of surrogates (ibid.). This means, that LIME trains on the surrogate model, which is used locally to explain specific predictions (ibid.).

What LIME does is that it sees whatever model it is given as a black box. A blackbox is a system, where people can not see the internal mechanisms. LIME then generates a dataset of samples and their predictions based on the model. Based on this dataset, LIME trains an interpretable model such as a decision tree. The model is then a good local predictor, but it does not have to be a good global predictor for the model (ibid.). By focusing locally and training a more explainable model, we can get a sense of what features were important in the decision and their weight.

The process of training a local surrogate can be split into five steps: First an instance is selected where the user wants an explanation for the prediction on that instance. Secondly a dataset is generated and the predictions on this dataset is retrieved. This however, on a blackbox-level, means that only the outcome of the model is retrieved.Thirdly the samples generated in step two are weighted according to their distance to the instance that was chosen in the first step. A weighted, interpretable model is then trained on the dataset described in step three. The prediction is then explained by interpreting this local model.

When using the LIME import these are the steps that are executed, and the last step is the explanation we retrieve from LIME.

4.4 Unified Modeling Language

Unified Modeling Language (UML), is a standardized language of diagrams, meant to describe structures and processes in a system. The purpose of making UML diagrams, is to specify and explain a system to other developers, as well as making yourself and your team notice potential flaws and opportunities in the system (Visual Paradigm, n.d.). UML models can also work as an explanatory method, to give the user/receiver a look into how the program is built, and how it should be used.
This method of describing structures and processes relates directly to the dimension Design and Construction, which focuses on designing and modelling systems.
This section is also related to step 5 *Create a model of the technology* in the TRIN-model.

4.4.1 Class Diagram

A class diagram is a technique used in object-oriented systems, to describe the objects and classes, and the relationships between them. The purpose of making a class diagram, is to visualize the static view of the system (Tutorialspoint, n.d.-c). Class diagrams are a popular choice when working with object-oriented systems, since classes is one of the main focuses. It is built by creating a box for every class in the program, which consists of three rows. The first row holds the name of the class, the second contains the attributes, and the last row has the methods or operations in the designated class. Every attribute and operation needs an access modifier, showing how accessible it is.

In a UML class diagram, there is a great focus on the interactions between the different classes, and their association to each other. This is represented by different types of lines, where the default relationship is the bidirectional association. The classes have a line connecting them, and their relation to each other is written in each end with a multiplicity value, as shown in the example below (Lucidchart, n.d.).

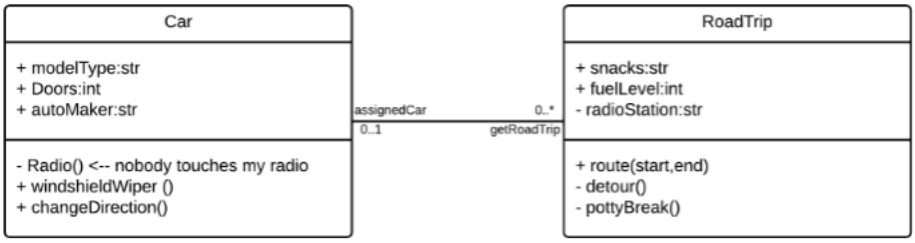

Figure 4: Example of class diagram (ibid.)

Making a class diagram makes it easier to illustrate either small or big data models, such that it is a simpler process to understand it, and to get an overview of the whole system. It can also be a great advantage to make before creating the program, to ensure that the direction and purpose is clear.

4.4.2 Sequence Diagram

The sequence diagram shows a specific scenario of a use case, in which the program works. The diagram portraits the scenario happening over time, and explains the specific actions that are taken, and the consequence of that action. It has some similarities with a flowchart, since it takes the user through how the system works. The difference is, that the sequence diagram shows a chosen case, and gives a much more elaborate and advanced explanation of the process (Visual Paradigm, n.d.). To achieve explainability is not only about understanding the model, but can also be about understanding the program itself, and how the code runs. The example below portraits how a sequence diagram looks and works.

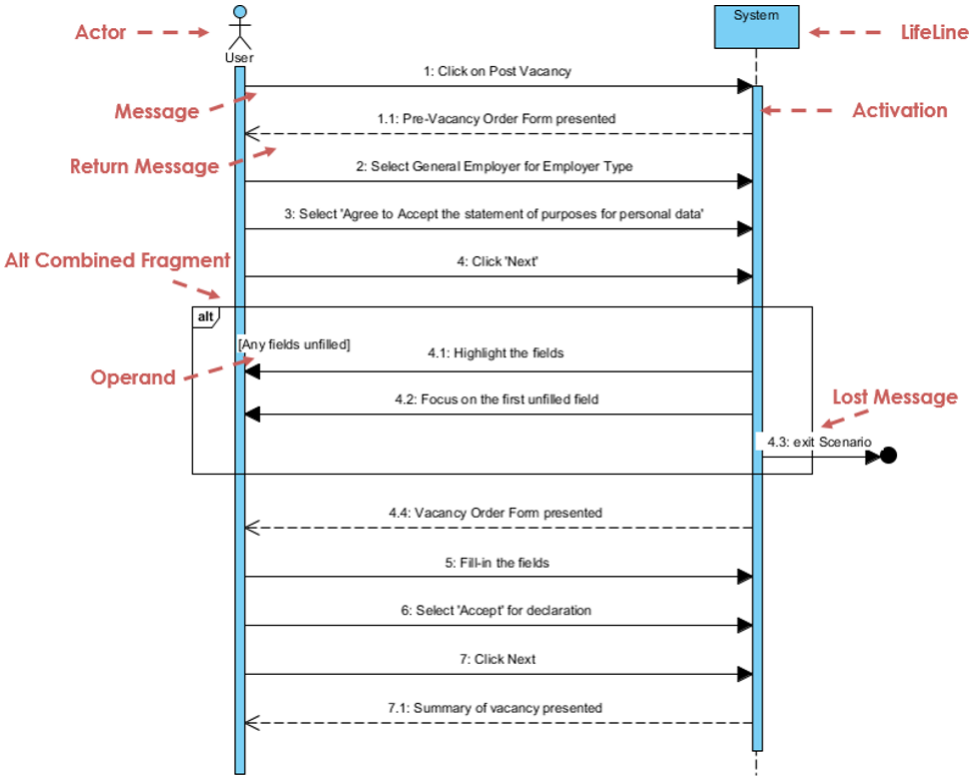

Figure 5: Example of sequence diagram (ibid.)

4.5 Object-oriented approach

To describe the object-oriented approach, the data structure is built upon objects, that has unique behaviors. The functions and the logic is attached to these different objects. In OOP you create different classes, where objects are the instances of these classes (Towards data science, 2019).
When working with an object-oriented approach - in any given programming language - there are some fundamental concepts related to the object-oriented design. Three of the main concepts are *abstraction*, *inheritance* and *polymorphism*.

Abstraction
Abstractions are the approach, where the user is not exposed to all the implementation of the functionalities (Clark, 2013).

As already mentioned in section 3.1, abstractions are a crucial component, when working with the philosophy of computer science. In OOP, abstractions allows the computer scientist to hide information in objects, and thereby reduce the complexity. According to this theory, OOP can be described as a method, to

constantly allow further abstraction. Abstraction is a relevant method in relation to this project, because of our object-oriented approach.

Inheritance
Inheritance is the process of creating new objects, based on other objects of other classes. It allows computer scientists to use and reuse logic, which makes it easier and more intuitive to understand the code (ibid.).

Polymorphism
This is the process of creating objects that respond to the same message, in their own unique way (ibid.). The same object, can have multiple functions. It allows the computer scientist to create different methods of the same object.

Beyond these three key-points an important component to OOP is Unit Tests.

Unit Tests
The idea about testing a software in general, is to evaluate whether or not the code is ready to be presented to a user. In unit testing, you take individual units of the code, and run it through various tests (Sharma, 2020).

4.6 Regression and model building

In this section we will account for what regression models is, and how we are going to apply them to the project. The book used to obtain this knowledge is *Introduction to Linear Regression Analysis, fifth edition* by Montgomery et al., 2012. This section will mainly focus on the primary regression models that are going to be used in this project.

Regression analysis is a statistical method of exploring and modelling correlations between variables. Regression analysis can be used in various fields of study such as economics, engineering, biology, social science etc. There are multiple types of regression models tied to ML. The most used and commonly known types of regression are Linear regression, Polynomial regression, and Logistic regression. For this project we will mainly focus on Linear regression, so we will not go into detail regarding the other types. A very important part of building a regression model is the data that you will be using. A good data collection can make or break the interpretation of the data and thus the analysis (Montgomery et al., 2012). In our case, as we have not collected our own data, we will have to rely on the dataset we have chosen, and clean it as described in section 5.2.1.

Regression models can be used for multiple purposes, such as describing data, prediction and controlling processes etc. Description of data can be very useful when you are trying to understand the data you have collected. Say that in our case we have collected a large sample of movie review data. This data can seem

a bit confusing due to its size, and it can be hard to understand the correlations between the different data parameters. A regression model, can act as a summary, which can help understanding this correlation. Prediction of future outcomes based data can also be a way to work with regression models. Say we want to find out which movie genres will be rated the highest based on different age groups. Using a predictive algorithm like random forest regression, we can train our model, based on the data to make this prediction. It is important to note that there are some risks when using predictive modelling, however we will come back to that later. As mentioned, regression models can also be used to control data. An example of this, is looking at whether or not a process or machinery is behaving as expected. Here we can use the data to control whether the data we have, is within expected parameters and if not, this anomaly can be fixed. (Montgomery et al., 2012)

Building a regression model is an iterative process, that requires the user to constantly adjust and check that the model is accurate and adequate. Lastly the model needs to be validated using validation data. Validation data is usually a part of the original dataset, that has been separated from the data used to train the model.

The model below visualizes the process of building a regression model, as explained above.

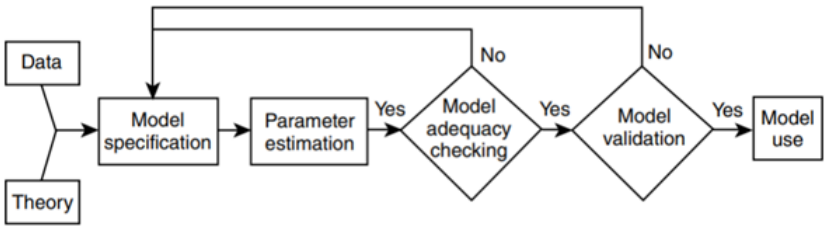

Figure 6: Regression model process (Montgomery et al., 2012: 10)

4.6.1 Simple and Multiple Linear Regression

In this section we will go more into detail with the Linear regression model, how it works and the mathematical background.

There are two types of linear regression models, *simple linear regression models* and *multiple linear regression models*. The simple linear regression model is a model where there is only one regressor variable X (also called the independent variable), which holds a relationship with the response variable Y (also called dependent variable). This regression is represented in a straight line.

Multiple linear regression is very similar to simple linear regression, however in multiple linear regression there are multiple X variables (independent vari-

ables). The goal is to understand what happens with Y when X changes. If both the X and Y value is increasing, there is a positive relationship between the two variables. However, if X increases, and Y decreases, there is a negative relationship between the two variables.

As an example, we can set "duration" as Y and "total votes" + "genre" as X in a scatter plot diagram. We want to draw a regression line through our plotted data. Once the regression line has been drawn, we will need to calculate the error. The error is the distance between our points and our regression line. The bigger distance there is between the points and the line, the greater the error. The goal is to minimize our errors.

A way to minimize errors is to improve the quality of the data or rethink whether we are using the correct independent X variables. Improving the data can be done by further data cleaning and reassessing the variables can be done by feature engineering.

4.6.2 Into the forest

In this section, we want to account for what *decision trees* and *random forest regression* is. We will also describe why these are commonly used methods in order to create different regression-models.

Decision trees
A decision tree can be seen as a sequence of decisions or choices, where a selection is made at each stage through the tree. The decisions are seen as nodes and their branches can be seen as choices. The analogy to building a forest is a large amount of these trees, and building a random forest means that these trees are randomized.
The decision tree is at the root of the random forest, and before we can grow a forest, we must know exactly what a decision tree entails (Necaise, 2011).

The random forest
Decision trees are roots in a random forest. A random forest consist of multiple decision tress. It is build as a tree-like structure, and is explainable because the random forest shows how the decision trees makes their decisions. The decisions are based on "yes" or "no" decisions, until it gets the desired outcome (JavaTPoint, n.d.).
Behind forests lies the idea of "bagging" also know as "bootstrap aggregation". Bagging in this case, refers to using ensemble learning with the same model on different parts of the data. In connection to this, we split the data randomly into different bags, so that these bags holds a random subset of the data. We then build decision trees around these bags, and when they are combined, they create a forest (Sarkar, 2021).

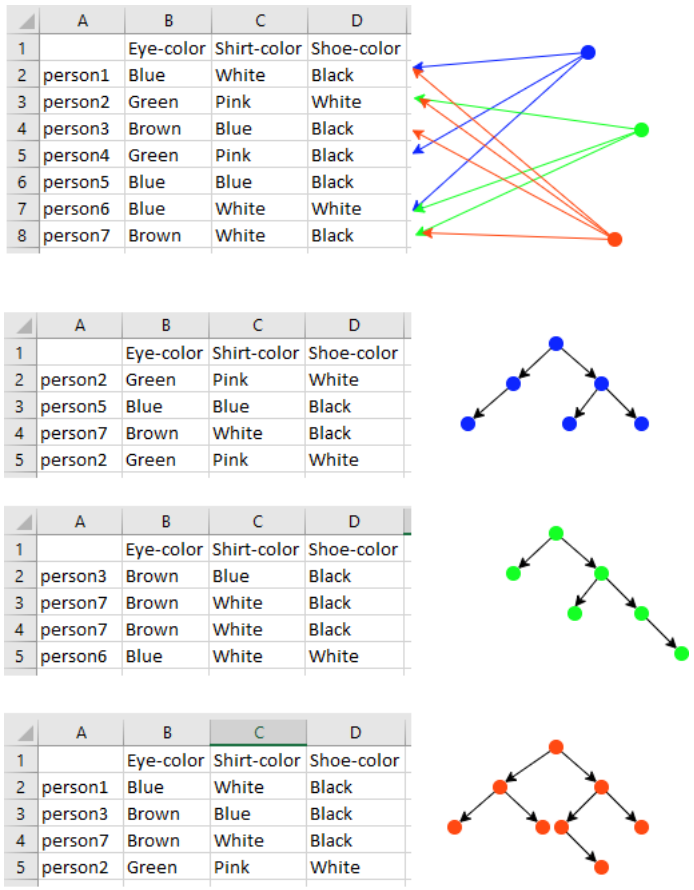

Figure 7: Example of random forest

Figure 7 serves as a rough visualization of the random forest process, in this specific example it would be random forest classification.
Decision trees or random forests are popular and often used methods for classification in ML.
Another commonly used method in ML is regression, which can be used to show the correlation between values in a dataset.

5 The program

In this section we want to account for the program we have developed, with all the previously mentioned theory and methodology in mind. We want to describe the Python packages we have included in order for our program to work. Next, we want to involve the classes we have creates, and display some code examples. Lastly we want to display our sequence diagram.

In order to run the actual program, we refer to the README file at our GitHub Repository. In the file we will disclose which programs and downloads are needed, to be able to run the code. The link to our repository can be found on the front page. To see the program being run, we refer to the video in **appendix 3**.

5.1 Imports

In this section, we want to list all the python packages we have imported, in order to create our program. The imports are as following:

- Numpy
- Pandas
- Plotly
- Dash
- Json
- Matplotlib
- Seaborn
- Scikit-learn
- Graphviz
- SciPy
- OS
- RANDOM
- LIME
- PIL

In **Appendix 1**, we have extended definitions of these imports.

5.2 Classes and their internal mechanisms

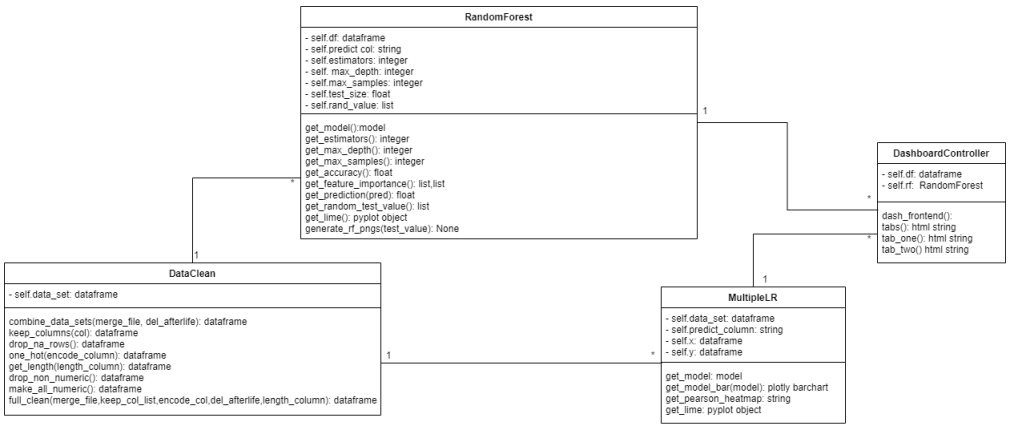

Figure 8: Full class diagram

We will now go in depth with our classes an their internal mechanisms.

This section has been written on the basis of the theory from the TRIN-model, and the first two steps. We will identify and analyze the inner mechanisms and processes of the technology and its artefacts.
This section will describe how the program is built, and we will define the purpose of each aspect of the code, in order to explain the functionality of the program.

5.2.1 DataClean

In this project we refer to data cleaning and data pre-processing interchangeably. The class DataClean has the function of cleaning the data, before it is used further in the program. It is a crucial component to make sure that the DataFrame that we used can be further processed by our other classes. What is absolutely necessary in our DataClean is that the DataFrame is fully numerical, meaning that all values are either integer data types or float data types.

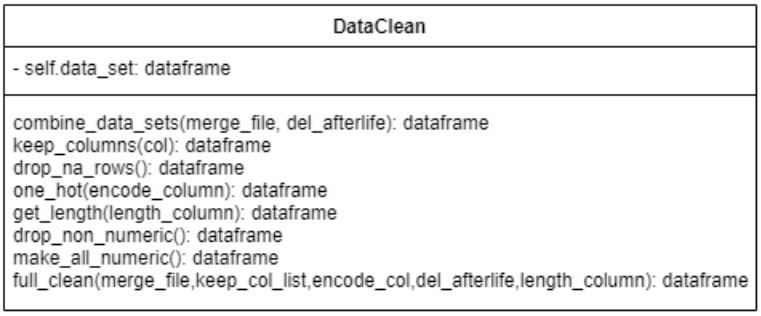

Figure 9: DataClean class diagram

In the section below, we will explain every method that DataClean contains, and what they do. With these explanations, we will also display the actual method from the code. We will go through them from beginning to end, excluding imports, which we have already accounted for.

Listing 2: Python code

```
def __init__(self, data_set):
    self.data_set = data_set
```

__init__ initializes the class and takes a Pandas DataFrame as an argument. The DataFrame that is added on the initialization of the class, is the DataFrame that will be manipulated throughout the methods of this class.

Listing 3: Python code

```
def combine_data_sets(self, merge_file, del_afterlife):
    self.data_set = pd.merge(self.data_set, merge_file)
    del self.data_set[del_afterlife]
    return self.data_set
```

combine_data_sets merges the Pandas DataFrame that has been passed as an argument, to the DataFrame that has been passed on the __init__ function of the class. It takes two arguments; the Pandas DataFrame that is to be merged, and a string value of a row that is to be deleted after the merge has happened.

The Pandas merge function can identify which column should be used as a key for merging the lists. In the case of our program, it finds that we have a imdb_title_id column which exists in both DataFrames. We use the del_after to tell the method to delete this column afterwards, since it is not relevant for our ML-models.

Listing 4: Python code

```
def keep_columns(self, col):
    for i in self.data_set.head():
        if i not in col:
            del self.data_set[i]
    return self.data_set
```

The keep_columns method takes an argument called col. The col argument is supposed to be a list of the columns in the Pandas DataFrame that the user wants to keep in the DataFrame. The method uses a **for** loop, to loop through the column names of the Pandas DataFrame. If the column name is not in the col-list it is dropped from the DataFrame.

Listing 5: Python code

```
def drop_na_rows(self):
    self.data_set = self.data_set.dropna()
    return self.data_set
```

The drop_na_rows uses the function dropna() which is built into Pandas. The dropna() function goes through the DataFrame and deletes rows that contains missing values.

Listing 6: Python code

```
def one_hot(self, encode_column):
    data_split = self.data_set[encode_column].str.split(",")
        .apply(pd.Series)
    df = (pd.get_dummies(data_split[0])
        .add(pd.get_dummies(data_split[1]), fill_value=0)
        .astype(int))
    self.data_set=self.data_set.join(df)
    del self.data_set[encode_column]
    return self.data_set
```

The one_hot method takes the argument of encode_column, which is a string value that contains the name of the column, that is to be one-hot encoded. It splits the values in each row in the column into multiple rows. These values then get comma-separated, and pd.Series gets applied, which transforms them into a Pandas Series object. The one_hot method separates these different genres by splitting them, and creates a one-hot encoded matrix. This is done by using the Pandas get_dummies function. We merge the matrix onto the initial DataFrame, and delete the old column.

Listing 7: Python code

```
def get_length(self,length_column):
    data_split = self.data_set[length_column].str.split(",")
    df = data_split.str.len()
    del self.data_set[length_column]
    self.data_set = self.data_set.join(df)
    return self.data_set
```

The get_length method takes the argument of length_column which is a string value, that represents the column, that the method is to be used on. The purpose of the method is to get the length of a list that contains a single cell in a DataFrame. As an example each film has a column called countries that contains a list of the countries that has participated in producing the movie i.e.: Denmark, Norway, Sweden all these are however contained within one cell, so we split this cell into a list. We get the length of that list, and merge those numbers back into the DataFrame. We then have a column of the amount of countries that participated in producing the movie and delete the column that was there before.

Listing 8: Python code

```
def drop_non_numeric(self):
    for i in self.data_set.head(1):
        self.data_set = self.data_set[self.data_set[i]
            .apply(lambda x: x.isnumeric())]
        return self.data_set
```

The drop_non_numeric method loops through the DataFrames columns using a **for** loop. It then uses a **lambda** function to check if all values are numeric, if they are not, it deletes the row.

Listing 9: Python code

```
def make_all_numeric(self):
    for i in self.data_set.columns:
        self.data_set[i] = pd.to_numeric(self.data_set[i])
```

The make_all_numeric function loops through data columns and converts all values to numeric values. This is necessary, since the DataFrame will label any row that contains a string, as all string values. This means that even though we have dropped non-numeric values, the data type of string can still exist, if that string contains only numbers. This method then makes sure all of the values in all rows are numeric data types such as integers or floats.

Listing 10: Python code

```
def full_clean(self, merge_file, keep_col_list, encode_col,
    del_afterlife,length_column):
    self.combine_data_sets(merge_file,del_afterlife)
    self.keep_columns(keep_col_list)
    self.drop_na_rows()
    self.one_hot(encode_col)
    self.get_length(length_column)
    self.drop_non_numeric()
    self.make_all_numeric()
    return self.data_set
```

The full_clean method combines and uses all the methods in succession. It does not serve any other purpose than keeping our main class clean and short. This method has been adapted to our specific DataFrame.

5.2.2 RandomForest

Listing 11: Python code

```
def __init__(self, df, predict_col, estimators, max_depth,
    max_samples, test_size):
    self.df = df
    self.predict_col = predict_col
    self.estimators = estimators
    self.max_depth = max_depth
    self.max_samples = max_samples
    self.test_size = test_size
    self.get_model()
    self.rand_value = self.x_test[random.randrange
        (len(self.x_test))]
    self.generate_rf_pngs(self.rand_value)
```

The RandomForest class is very extensive, and there are a lot of initializations. The reason we are taking all these arguments and saving them in our __init__, is because we want to be able to call these in our dashboard. This means that they need to be initialized to a local value.

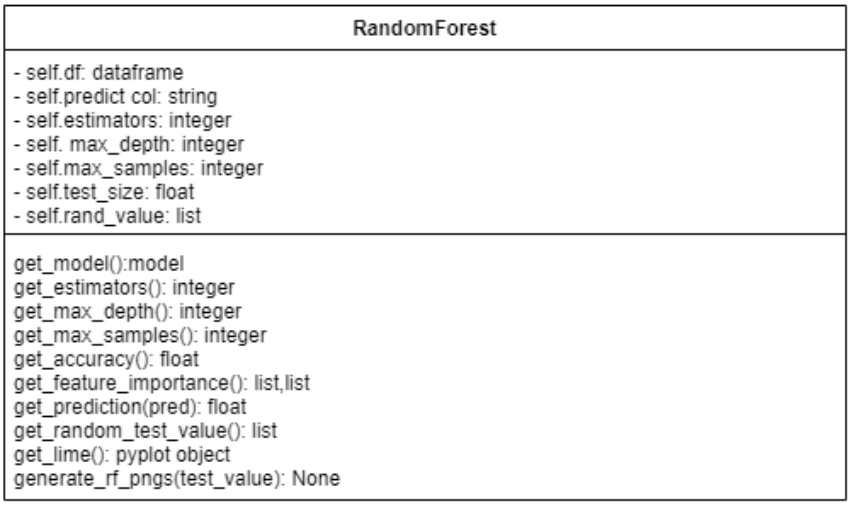

Figure 10: Random Forest class diagram

The class, as all our other classes, takes the DataFrame as an argument. It then takes a lot of arguments, that are related to creating the model from scikit-learn. Most notably is the fact that it initializes the model in our `__init__`. This is done because we want to generate the png's of the trees upon initialization, since it is too expensive to keep doing this every time we switch to the 'Random Forest'-tab in the dashboard. The `self.rand_value` gets random test-data from our defined test-dataset that we use later on in the class.

Listing 12: Python code

```
def get_model(self):
    self.train, self.test = train_test_split(self.df,
        test_size=self.test_size)
    self.y_train = np.array(self.train[self.predict_col])
        .tolist()
    self.x_train = np.array(self.train.drop([self.predict_col],
        axis=1)).tolist()
    self.x_names = self.train.drop([self.predict_col], axis=1)
        .columns.tolist()
    self.x_test = np.array(self.test.drop([self.predict_col],
        axis=1))
    self.y_test = np.array(self.test[self.predict_col]).tolist()
    self.model = RandomForestRegressor(n_estimators=self.
        estimators, bootstrap=True, max_depth=self.max_depth,
        max_samples=self.max_samples)
    self.model = self.model.fit(self.x_train, self.y_train)
    return self.model
```

The `get_model` method produces the random forest model that is used. It starts by splitting the data into training and test data based on the arguments given in the class' `__init__` method. Then it splits these into labels, and changes them into the needed datatypes to be able to run the `RandomForestRegressor` from the

scikit-learn library. At last it produces the model based on the training data. It does not use the test data in its model creation.

The following seven methods are just get methods. They get values from the class to be used in the dashboard.

Listing 13: Python code

```
    def get_estimators(self):
        return self.estimators

    def get_max_depth(self):
        return self.max_depth

    def get_max_samples(self):
        return self.max_samples

    def get_accuracy(self):
        return self.model.score(self.x_test, self.y_test)

    def get_feature_importance(self):
        importances = self.model.feature_importances_
        x = self.test.drop([self.predict_col], axis=1).columns.values
        y = importances
        return x,y

    def get_prediction(self,pred):
        return self.model.predict(pred.reshape(1,-1))

    def get_random_test_value(self):
        return self.rand_value
```

The get_feature_importance uses the feature_importances_ method that comes with the scikit-learn random forest library. The x value shows us the column names of the features, and the y value returns the weights that the feature importances has been given in this specific model.

The get_prediction is passed a Python list called pred, which is the Python list that the method makes the prediction on.

The get_random_test_value returns a random test value that is fetched from the test data, defined in the get_model method.

Listing 14: Python code

```
def get_lime(self):
    x_test_local = self.train.drop([self.predict_col], axis=1)
    explainer = lime.lime_tabular.LimeTabularExplainer
        (training_data=np.array(x_test_local),
        mode = 'regression',
        feature_names=self.x_names,
        categorical_features=[0]
        )
    exp = explainer.explain_instance
        (data_row= np.array(x_test_local.iloc[5]),
        predict_fn=self.model.predict)
    exp.as_pyplot_figure(label=1)
    return plt
```

The get_lime method produces a plotly.express figure object of LIME for the specific prediction it has produced. It uses a prediction of the model that is initialized in the __init__ of the class and data fetched by the get_main_test_value method.

The class generate_rf_pngs is a large piece of code, so we have chosen not to include the full code directly in this text section, so we refer to our GitHub repository or **appendix 2** if a full walkthrough of the code is wanted.

The class essentially creates PNG's of five of the trees that has been generated in the random forest model. However, to show the specific path for a value it was necessary to do some workaround.

The class creates .dot files of the trees, and then gets the specific path for a prediction made by the random forest regression model. It then transforms the .dot file to a json file. By using the Python dict, it finds the nodes that corresponds to the prediction, and changes the color values of these specific nodes. Then the method transform these json files back into .dot files, which the graphviz import then can convert into PNG files that are saved in a local assets folder, which is accessible by the dashboard.

5.2.3 DashboardController

The DashboardController is the class that creates and launches the dashboard solution. It deals with the visual components using the dash library, which mainly uses HTML. The Dashboard controller uses a tab menu that has two tabs. Each tab is then defined as a new function with HTML code, that we call using the callback function, which is a built in part of the dash library. The callback function in its essence is what allows us to communicate with the dashboard.

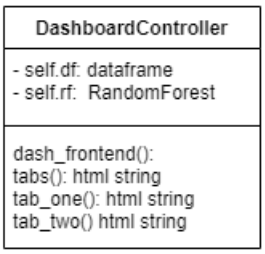

Figure 11: DashboardContoller class diagram

for multiple linear regression, we use tab_one. It gets the Pearson heatmap by the html.Img-function, which gets the image from a specific path. Further down it uses the dash core components library's Graph function, this function allows for the dashboard to call for a plotly figure, which we parse to it by using the get_model_bar() method from our MultipleLR class.

Our biggest tab that holds information and descriptions for the Random Forest model, is tab_two. It calls on the RandomForest class's get methods to display text that is directly in relation to the random forest model, that has been generated on the dashboards __init__ function. It uses the dash core components Graph function to display the feature importance from the model. It creates the plotly.express bar chart locally, but gets the x and y values from the get_feature_importance method from the RandomForest class. The method uses alot of the get functions from the RandomForest class to display variables, that are directly related to the model that was specifically generated for this dashboard. What is most noticeable here, is that by taking an object-oriented approach we can keep the text description local to the model. The rest of the visuals that are included in the dashboard comes from PNG files that RandomForest has generated and placed in the assets folder. The methods that generate these PNG's return a string value with the path to the specific PNG's. The dash import can only retrieve objects, used in the dashboard that are not native to plotly or dash if they are placed in an assets folder.

5.2.4 MultipleLR

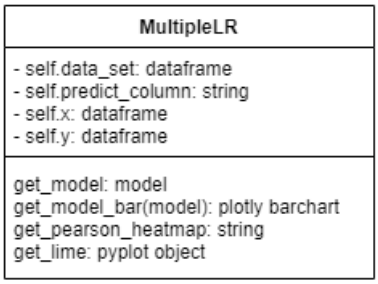

Figure 12: MultipleLR class diagram

The MultipleLR class has four methods in its __init__ method. It takes two arguments, data_set is supposed to be a Pandas DataFrame and predict_column is supposed to be a string value equivalent to the column from the Pandas DataFrame. predict_column is the values that the user wants the multiple linear regression to predict. The initialization then splits the dataset into self.X and self.y. self.y is the column, which is removed from the self.X Pandas DataFrame.

Listing 15: Python code

```
def __init__(self,data_set,predict_column):
    self.data_set = data_set
    self.predict_column = predict_column
    self.X = self.data_set.drop(columns=[self.predict_column])
    self.y = self.data_set[self.predict_column]
```

The method get_model defines the variable model as the LinearRegression imported from the sklearn.linear_model library. The method then fits the model using the native .fit() function from the LinearRegression import. The method then returns this model.

Listing 16: Python code

```
def get_model(self):
    model = LinearRegression()
    model.fit(self.X, self.y)
    return model
```

The method get_model_bar returns a plotly.express figure. The plotly library works directly with the dash import, which means that this figure can easily be called in the DashboardController. The method starts by labeling the features 'Positive' or 'Negative' using a python list comprehension. It bases the labels on whether the values are above or below zero. The method then generates the figure using the .bar() method from the plotly.express library. Its x-values are the self.X Pandas DataFrame we defined on the initialization of the class. The y-values are the coefficient values of the model, which is retrieved by calling the .coef_ function from the sklearn.linear_model import.

Listing 17: Python code

```
def get_model_bar(self,model):
    colors = ['Positive' if c > 0 else 'Negative' for c in
        model.coef_]
    fig = px.bar(
        x=self.X.columns, y=self.get_model().coef_,
            color=colors,
        color_discrete_sequence=['red', 'blue'],
        labels=dict(x='Feature', y='Linear coefficient'), title
            ='Weight of each feature for predicting movie rating'
    )
    return fig
```

The method get_pearson_heatmap creates a matplotlib.pyplot figures. These cannot be imported by the dash library which means it is necessary that we save it as a picture, and access the model as a picture in the dashboard. It is generated by using the .corr() function from the Pandas library. This function gets the pairwise correlation of the columns in the DataFrame (The pandas development team, n.d.-b). It then uses the import seaborn to generate a heatmap of these correlations and saves that figure to the assets folder. The method returns a string value equivalent to the path to the picture.

Listing 18: Python code

```
    def get_pearson_heatmap(self):
        plt.figure(figsize=(20, 10))
        cor = self.data_set.corr()
        sns.heatmap(cor, annot=True, cmap=plt.cm.Greens)
        plt.savefig('assets/parson_heatmap.png')
        return 'assets/parson_heatmap.png'
```

5.2.5 UnitTests

The UnitTests class is unique in the sense that it uses the unittest import. It also takes an argument in its class definition unittest.Testcase which is unusual. This is necessary to let the program know that this class is specifically meant for unit testing the program.

We acknowledge that the amount of tests and the tests that are currently written in the UnitTests do not cover what is necessary to call this a functional unit test. We include this part of the program because it highlights one of the strengths in object-oriented programming. It currently holds four unit tests, which tests whether there exist anything at the path Datasets/IMDb_movies.csv:

Listing 19: Python code

```
        def test_df_import(self):
            self.assertTrue(path.exists('Datasets/IMDb_movies.csv'))
```

It tests if there are missing values in the DataFrame 'Datasets/IMDb_movies.csv, which in this case might fail because we are not currently resaving the cleaned dataset to this file.

Listing 20: Python code

```
        def test_if_nan_exist(self):
            df = pd.read_csv(r"Datasets/IMDb_movies.csv",
                low_memory=False)
            for i in df.columns:
                self.assertFalse(df[i].isnull().values.any())
```

It tests if there are non-numeric values in the dataset 'Datasets/IMDb_movies.csv meaning if there are values that are not of the type 'int64' and 'float64'. This test would also fail, because we are not saving the cleaned dataset to this location yet.

Listing 21: Python code

```
def test_if_non_numeric_exists(self):
    df = pd.read_csv(r"Datasets/IMDb_movies.csv",
        low_memory=False)
    for i in df.columns:
        self.assertEqual(str(df[i].dtypes), 'int64' or
            'float64')
```

Lastly the UnitTests class tests if there is anything at the location 'assets/pearson_heatmap.png' which would imply a Pearson heatmap has been generated and saved to that location.

Listing 22: Python code

```
def test_mlr(self):
    self.assertTrue(path.exists(
        'assets/pearson_heatmap.png'))
```

5.3 Sequence diagram of the program

In this section, we will be using theory from the TRIN-model, step 5 *create a model of the technology.*

In order to visualize the technology, we have made a sequence diagram, portraying how the program we have created, should be used. This model shows, how the program runs, which means that it is a walkthrough of exactly what the program does, step by step. As earlier mentioned, the sequence diagram is meant to give a broader understanding of the program, which is why we have chosen to use this specific diagram. Our program sends our datasets through a lot of preprocessing, which could be difficult to understand, if not visualized. By showing the specific steps taken, and all of the things the datasets goes through, before actually getting used, we attempt to make the code and program itself, more explainable. The goal is, to show the sequence in which the classes are executed in our program. Our program is separated into six classes, *main*, *DataClean*, *MultipleLR*, *UnitTests*, *RandomForest*, and *DashboardController*, as shown in **figure 13** below.

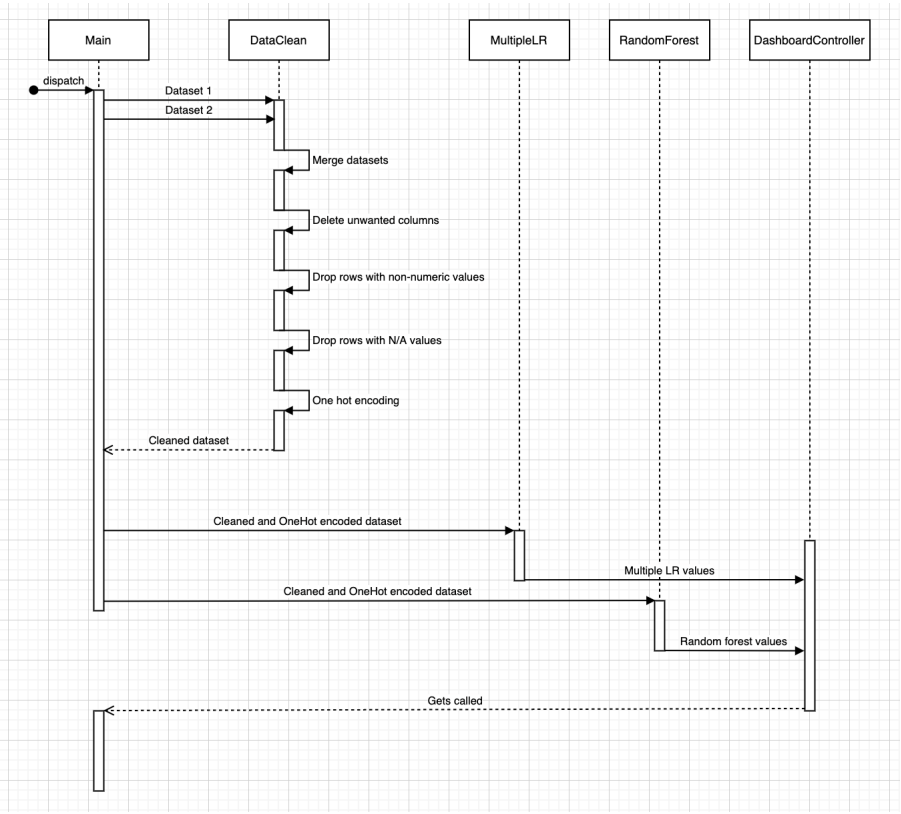

Figure 13: Our sequence diagram.

The goal of our sequence diagram is to make it simple and easy to follow, even though one is not necessarily familiar with the type of diagram beforehand. Our main sends the two datasets to the DataClean, in which they will get merged. Then the program removes the unneeded columns, drops the rows that holds values that are not numeric, drops the rows with empty cells, and lastly it one-hot encodes the data. Then the cleaned and merged dataset gets send to MultipleLR, where the multiple linear regression gets initialized, and sent directly to the DasboardController. The RandomForest goes directly to the DashboardController too, where it is represented as a bar chart. The same approach is taken in the RandomForest class, where we create the tree-models and the LIME-charts.

6 Analysis

In this section we will analyze our program using the TRIN-model. Furthermore we will examine how the object-oriented approach has contributed to this project. Lastly we will analyse our dashboard and the explainability it provides.

6.1 TRIN-model (TSA)

In this section, we want to include step 3, 4 and 6 from the TRIN-model, in order to analyze our program, and our project in general.

6.1.1 Step 3: The unintended effects of the technology

In this step we would like to address some of the unintended and unpredictable effects of the technology. These can be both positive and negative effects, however the focus, as the theory describes, will be on the negative effects.

Unintended effects in this case, will be defined as actions or consequences based on our program that we did not intend to happen. As already described the unintended effects can both be in relation to the technology, for example flaws in the code, but it can also be due to user interaction with the program.

As we cannot predict all usages of our technology, some usages may cause other outcomes than what we intended. One could argue that when we try to predict unintended effects, we should also try to prevent these effects in some way. This is done in our README file that comes with the program. In the README file we will state what the intended usages of the program are, so that hopefully we can prevent the happenings of unintended effects.

The following section will be divided in *certain effects* and *lasting effect*, as described in the theory from the TRIN-Model.

Misinterpretation
One lasting effect that might occur, is if our ML-model are interpreted wrongly and therefore causes bad decisions based on the prediction. To be more specific, this might occur if our code is simply not good enough, to actually make the right prediction. It could also occur if our explainability is not good enough, then the user might not understand the predictions.
This effect is one we can affect directly, with two steps. First of all, by optimizing our code, so it does not make wrong predictions. Secondly, by making sure that the explainability of our ML-model, is as accurate as possible, in order to prevent misinterpretation.

One of the actions we take to combat misinterpretation is that we explain each step our ML-model takes. This will result in a written record of the results that it predicts, so that the user does not have to derive the basis of the prediction themselves.

As mentioned earlier, there can be unintended effects caused by the creation of the code, and possible flaws in it, or unintended effects related to the user interaction. We consider that the misinterpretation, would be an uncertain effect caused by the technology itself.

Unintended effects of using the technology within the scope of explainability
One of the certain effects that generally lie within the field of explanations, is the fear of creating a false sense of security for people, as well as not having highlighted the Rashomon effect. If we return to the example of the robot vacuum, the user might feel they understand why the vacuum stopped working, when it arrived at a different surface area. If we pretend that the user likes their vacuum a lot, they might decide to change out their entire flooring. However, they might have falsely believed that surface area was the problem and find that the vacuum still stops in the same location.
ML-models explanations can be false, and the decisions humans then make based on that false sense of security can be severe. To recap: The Rashomon effect is the concept that multiple explanations can exist for a given outcome. This can lead to humans making faulty decisions based on the belief, that the explanation they received, included the full scope of every variable used in that specific decision.

As with any technology, unintended effects will arise throughout its use, and even unintended effects that fall far outside of the scope of what the technology tried to solve. We have found that the above effects fall within the scope of the project and covers the field of unintended effects as broadly as the project allows for, without testing the product.

Blind usage
One possibility for unintended certain effects, might be if the user blindly trusts the result of our prediction model.
As our program is generic, and therefore can be used for any type of dataset, there is no limit to what data the user can make predictions based on. The program will simply tell the user which data features has the biggest coherence, but that will not mean that they are correlated by any means. This can result in predictions that can be wrong, if the user trusts a prediction without being critical. Depending on what the user does with the prediction, there will be a possibility that the user creates dangerous or possibly harmful situations for either themselves or others.
There are countless examples of people blindly following their GPS systems for example. This is primarily due to the trust in the system, that it will always find the correct route. However, in some cases, unintended effects may happen, when a road has been closed, or the GPS faulty has mistaken a forest trail for a regular road, which might be dangerous. Therefore, the driver must always be critical about where the GPS leads you if it looks wrong (Maxwell, 2015).

Security, personal data and vulnerability
The last potential unintended certain effect that we want to highlight is the security aspect of our program. The user must always be aware of what is being downloaded or uploaded to the internet. When using the program, the user

must always be aware if they are downloading a dataset from the internet, as there is always the possibility of downloading something unintended, such as malicious software, along with the dataset. There is also the scenario where the user implements a self-elected dataset in the program. If the user then uploads a local version to Github etc. the user need to be aware whether their dataset includes sensitive personal data or other information that is not intended to be public.

To summarize, we have considered these three unintended effects as being uncertain effects, while they are not a result of the actual program, but the usage of it.
It is very important, that the user realizes how to interact with the program, or else, different kind of uncertain effects might appear.

6.1.2 Step 4: Connections in bigger technological systems

In order for an ML-model - or another AI-technology - to be useful in a society, it is essential that it is explainable enough, to be trustworthy. We can look at our society as being one big technological system, in which every small part, must work together with the others, to perform sufficiently.
Regarding the science and technology studies approach, we will analyze how our program works in a bigger technological system.
In the case of our program, it will not be useful to people, if the role it takes on as a part of the bigger system, does not contribute to the society.

Trust as referred to in theory section 3.3.1 plays a big role in the adoption of new technologies. The decision to accept and adopt it, is based upon many aspects of the technology's trustworthiness. In general, a technology needs to be explainable and understood, especially if it is something as complex as ML. Our program seeks to live up to these standards, by reaching a level of interpretability, that makes the data and models understandable for people, who does not necessarily have a background in the field, or any experience with it.

One of the reasons why it is so important to understand an ML-model, is to ensure that it will not have biases, or at least making the bias known to the users. It is necessary to detect bias as an ethical precaution, to avoid ML-models making unethical and discriminatory decisions.
The society in which the technology performs, has rules regarding how a new technology needs to behave in the bigger technological system. When a model is able to explain why it made the predictions or decisions it did, it becomes more transparent what the model does to and with the data, which creates trust in the technology.

As already mentioned ML-models have the possibility to contribute to society. An example could be cancer-detection which is beneficial to our society. However, if we do not understand, and are able to explain AI, there will not be

enough social acceptance of the technology. This could result in slowing down the adoption of potentially life-saving technologies. This could have happened with the previously mentioned example with the robot vacuum. If the users did not trust or understand the robot, they would not necessarily use it, since they would not be able to trust that it works or not. Furthermore, we must make sure not to leave society behind, since not having explainable AI will leave people sceptic and frightened by this technology, that can have huge beneficial impact on society.
To find out whether the program is explainable enough, we therefore need to get further knowledge about these social rules, to assess the level of interpretability needed to achieve this. The bias of the program needs to be identified and transparent, in order for it to be trusted.

If our ML-model should be used in the bigger technological system that is society, we first need to make tests, to assure that people understand the program enough, to actually trust it, and understand the benefits of using it. By producing our dashboard, we attempt to achieve this. However, without testing a product, we can never really be sure as to how well the program will work, and how it will be received. In this context, we can look at the programs input and output, regarding the implementation in the society.

As mentioned in section 2.1.1, about the 4th step in the TRIN-model, the functionality of a given system consist of an *input*, a *process* and an *output*, in order for this new technology to be called a system, and for the technology to fit to society.

When a new system is presented, the *input* will be the introduction to the society since it is being added to an already existing system. The *process* that then begins, is the above-mentioned approval-process, where the society in its whole, will judge whether or not the program lives up to the rules of trust and application. If the system gets implemented, the *output* will then be a society as a bigger technological system, in which the program is now a part of. This procedure is a recurring one, with every system that is to be adopted in a society.

Our program specifically will not necessarily get received and applied to the bigger system as it is now, since we have not tested the product on real users yet.
Our program is currently still in the development phase of our CRISP-DM model. This will be further examined in the next section.

Our project in relation to the CRISP-DM model
If we take a look on **figure 2** again, we can argue that our ML-model is currently at the modeling stage. We have gone through the first phase, concerning the *business understanding*. Our business understanding can be defined as being our problem statement. We wish to create an ML-model based on an object-oriented structure, and we want to increase the explainability of the models that

we create.
We have considered the second phase in the CRISP-DM model not to be that relevant in relation to our project, while our focus is not on what kind of data we are working with. We want to create an ML-model, that ideally could take any kind of data into use.
The third phase on the other hand, concerning *data preparation* is very important for our project. Even though we do not focus on the specific kind of data we use, the data still needs to be pre-processed.

If we move to the third phase in the CRISP-DM model, *modeling*, We have created a Pearson Heatmap, a Random Forest, a bar chart and LIME as our explainable models. As it is right now, we have not evaluated on our explainability solution. And there is a lot more potential in using the dashboard technology, than what we have managed to accomplish. Currently our process is located around the *modeling* phase.

6.1.3 Step 6: Technology as an innovation

In this section we will account for some of the drivers and barriers for implementing our technology.

Drivers for implementing machine learning models, in general
In ML, the main purpose is to generate a model, which has the ability to *generalize* and be able to predict and perform well, on unseen data (Collet, 2017). ML is a technology that is created to automate and digitalize tasks, normally done by humans. You can argue that the development and implementation of ML-models, is a product of the "high-speed society" (Rosa & Scheuerman, 2010) that we live in, where everything needs to be optimized, so that we humans can handle more and more tasks at the same time. A growing number of companies and sectors seek to use and implement AI/ML-systems, to improve production and the working processes.

As already mentioned in the report, ML and AI is about to be a more used and commonly recognized concept, and therefore *explanations* are becoming a very important tool, if we humans are supposed to trust and understand this technology (Turek, n.d.).

Barriers, and how to prevent them
To prevent the barriers that exist in the adoption of ML, explainability is a very essential concept. It is important in order to ensure that the end-user understands the function of a given ML-model, in order to prevent things such as blinded usages, as mentioned in section 6.1.1 about unintended effects.

Explainable AI is a widely discussed concept. The agency DARPA (Defense Advanced Research Projects Agency) seeks to improve Explainable Artificial Intelligence (XAI). DARPA is an agency that aims to; *"make pivotal invest-*

ments in breakthrough technologies for national security.“ (Turek, n.d.). They emphasizes the importance of assigning an extra dimension, on how to implement and work with ML. This is illustrated on **figure 11** below:

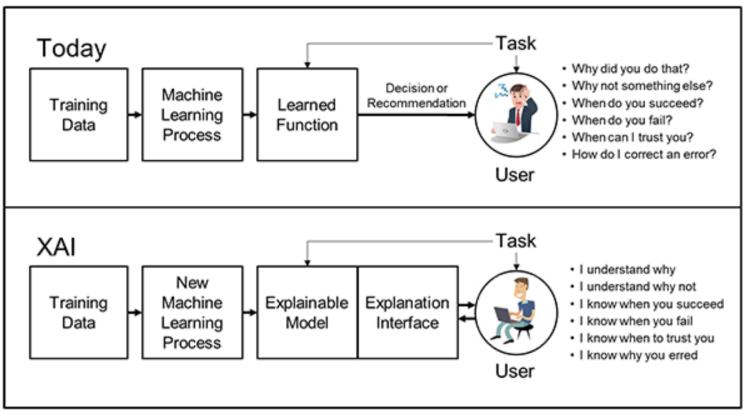

Figure 14: The concept of XAI (ibid).

Creating this other dimension will work as an optimization on the already existing ML-models, as we see them today. If we look at **figure 14**, which is the XAI concept developed by DARPA – then instead of just transferring the *learned function* to the user, which can cause a lot of barriers like distrust and uncertainty, there could be added an *explainable model*, and a *explanation interface*. If this dimension were to be implementing on how we work with ML DARPA argues, that it will improve ML-models, and make sure that the user understands what the models does, and prevent distrust between the outcome of the ML-models, and the end-user.

6.2 How has object-oriented programming benefited this project?

In this project we have an object-oriented approach and use abstractions all throughout the program which is not usually what you would see in a data scientific solution. How this affects the product will be mentioned in this section. The way it affects us regarding the philosophy of computer science is through our view of the product we produce.

As we knew it from data science, a common way of using Python, were to do procedural or notebook/script-based programming, where single lines of code are executed sequentially. As we progressed in the program it became clear, that we wanted to make a more generic code base, that was not limited to our chosen dataset, but could be used on any dataset. On that basis we wanted to take an object-oriented approach. Inside the community of programmers, it has long

been discussed, whether Python is an OOP language, or not. The creator of Python, Guido van Rossum wanted to keep the language simple, and by simple he means, not being able to 'hide' data, which is one of the things abstraction works with, which is one of the main concepts of OOP (upGrade.blog, 2019). As described, one of the properties of an explanation is portability. Portability refers to the amount of ML-models that can be explained from the program. At the current iteration, the only ML-models, that we have implemented is the mutliple linear regression and the random forest regression models. Because of our object-oriented approach, we can easily add another ML model to the program and display it in our dashboard. Another positive factor, is that LIME is model-agnostic, which means that it can be applied to any ML-model.

Even though it was not originally the intent, Python does support OOP because it allows the programmer to both create many objects of a class, and work with inheritance and polymorphism, which are very essential concepts of OOP. As we have already accounted for, the script-based approach did not support abstraction in the same way that OOP does. This was also a big factor in choosing to do OOP, because that we through abstractions can hide the extraneous data from the user. As we are creating a more generic type of explainable AI program, the user does not necessarily need to see the inner workings of the program. We can choose to abstract and thus hide, whatever we want to. This is one of the big factors behind choosing OOP.

Another big factor is the readability and the general understanding of the code. Had we used a script-based method, it would be quite hard to understand due to the scale of the program. The object-oriented approach allow us to easily navigate the program, and see what the different classes are able to do. The object-oriented approach also allows us to make changes in the program, either adding or removing objects. It could also be argued that the object-oriented approach increases the explainability of our ML-model, as we have a separate class entirely for our random forest ML-model.

6.3 How have our dashboard contributed to making the AI more explainable?

When evaluating the dashboard solution, we also must state that the product in its current form is, just one iteration, to which we think improvements can be made. There are multiple perspectives that can be taken when evaluating an explainability solution in ML-modelling. We first suggest looking at whether the solution developed meets the criteria of what makes explanations important.

We suggested three main reasons for why interpretability is important in section 3.3.1. These are *trust*, *moral and ethics* and *debugging*. In this section we will not be discussing whether the models included in the dashboard meet these criteria, but rather what our dashboard component adds to these models regarding trust, moral and ethics and debugging. The dashboard mainly expands on the

models it presents, using text and structure of the dashboard. We are adding expressive power, by including these components to the model. We propose that it increases the trust in the model, to hold the users hand and go through the model from a holistic global example to a local surrogate model, and offer explanations throughout. However, this way of going through the model is far from useful in every situation where interpretability is needed. Surgeons that need split second decision making and rely on ML-models, would not find any use in the slow-paced multi-perspective approach we take in our dashboard. We would argue, however, that it does make the explanations more understandable for non-technical people, because we include this text, and multiple perspectives. Perhaps the product could have value in situations where ML-models are being introduced to people who do not need to make split second decisions, and people that might be considering adoption of the technology. While LIME and multiple linear regression could be used for debugging or highlighting moral and ethical problems in the model, the approach which we have taken is better used in a situation, where we want to establish trust in the technology through explanations and information.

6.3.1 Dashboard

A dashboard can be described as a collection of various different images and interactive graphs, displayed with a certain layout(ibid.). There is a lot of ways to display correlation between data through dashboards such as; diagrams, maps, models ect.
As mentioned in **appendix 1**, dash is a library build on plotly, and it is meant for interactive web application and other visualizations (Plotly, n.d.-b). When working with data science and big data, it can be almost impossible for the human mind, to see and understand the correlation of the data, just by looking at the dataset. Visualizations is therefore a very important tool, in order to ensure interpretability.

In the following section, we will describe how we visualized our data correlation, and how it contribute to making our ML-model more explainable.

Regression and feature selection
We have created a multiple linear regression model, based on our dataset. In our dashboard, we have two tabs, with different visualizations. Tab one contains a *Pearson heatmap* and a *bar chart*. Tab two contains our *Random Forests* and *LIME* models.

6.3.2 Feature importance

Even if the program does not have any bugs, the dataset can still contain buggy data. *"'Buggy data' is unrepresentative or biased data."* (Stringer, 2018) which means that the dataset can have flaws, that needs to be resolved in the prepossessing of the data.

Feature importance are used for deciding which features are important for model predictions and for illustration of feature correlation.

Pearson Heatmap
Heatmap is a graphic representation of a correlation matrix (Kumar, 2020). The correlations are represented in any values from negative correlation (-1) to positive correlation (1). Negative correlation is when one variable increase and the other variable decreases. At a positive correlation is when one variable increase, the other variable increases too. If the correlation 0, there is no correlation between the variables (ibid.). Our heatmap contains the correlation between all the features in our dataset.

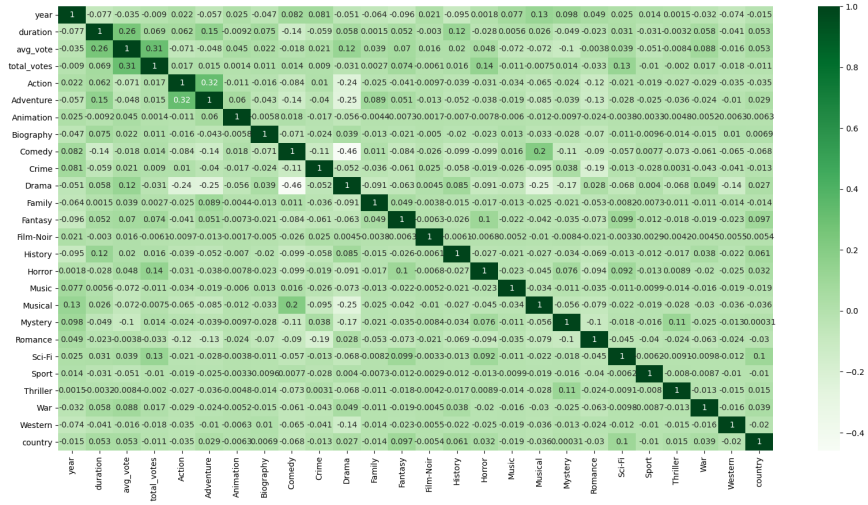

Figure 15: Pearson Heatmap

If you had to look directly at the dataset to find a correlation, it would be very difficult and uninterpretable. By making a graphical visualisation like a heatmap, the process of finding the correlations, is much more understandable.

Pearson heatmap - and heatmaps in general - is a commonly used method, for feature selection in ML (Okoh, 2019). In our program, we have implemented a Pearson heatmap through the Python library; Seaborn.
When making a heatmap visualization, it completely ignores non-numeric columns, which we handled in our data pre-processing phase (ibid.).

Our heatmap is color-coded, which is a way to enhance the explainability, and make it interpretable for the viewer of the visualization.
The correlation shown, is found through our multiple linear regression, and is

visualized by showing that the darker the color is, the more positive the correlation is, and visa versa.
Furthermore, when the correlation numbers are boiled down into small size numbers (-1 to 1) it is easier to interpret.

Bar chart
Another way of illustrating the correlations for the values can be through a bar chart. A bar chart visualizes the values in vertical bars. Our bar chart, can be seen at **figure 16**. The simplicity of the bar chart makes it explainable, and easy to understand. The bar chart shows the feature importance for all chosen feature, contrary to the heatmap, which shows for all features in our dataset.

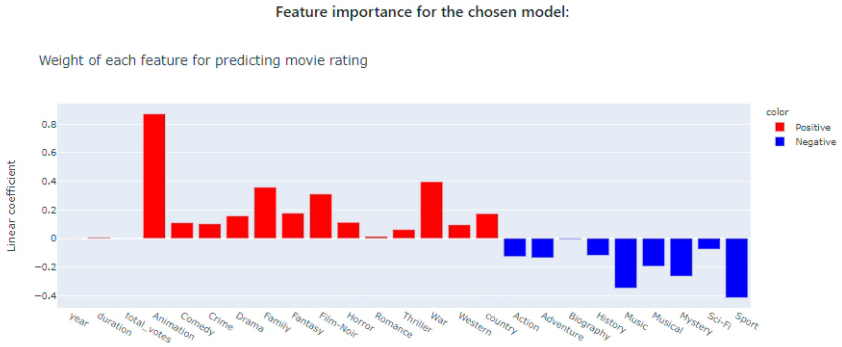

Figure 16: Bar chart

6.3.3 Random forest

The random forest model produces a pre-set amount of trees, forming a forest. The tree reaches a depth of five, which means that it makes a prediction after five decisions. Every node holds some variables, representing for example the most weighted feature for the decision. The maximum depth of the tree increases the explainability of the model as a smaller tree is easier to comprehend and display in a dashboard. This limit to the maximum depth, sacrifices some prediction accuracy, which will be discussed further in section 7.2.

Another explainable feature of our model is that our random forest trees has their trace path marked, so that the user can see exactly which path the ML model has taken.

Lastly it describes the random data points that the model has selected, so that the user can quickly understand, which data points the random forest model is generating its prediction from.

Figure 17: Random Forest Regression

6.3.4 LIME

LIME is an explanation technique. It explains a models prediction, and the user is allowed to interpret these predictions, and maybe even take action based on them (Hulstaert, 2018).
As mentioned in section 3.3.4, one of the extents of the explanation is *Local interpretability for a single prediction*, which is what LIME is. LIME change one single data, by tweaking the feature values, and then observe the result and the impact on the specific output (ibid.).

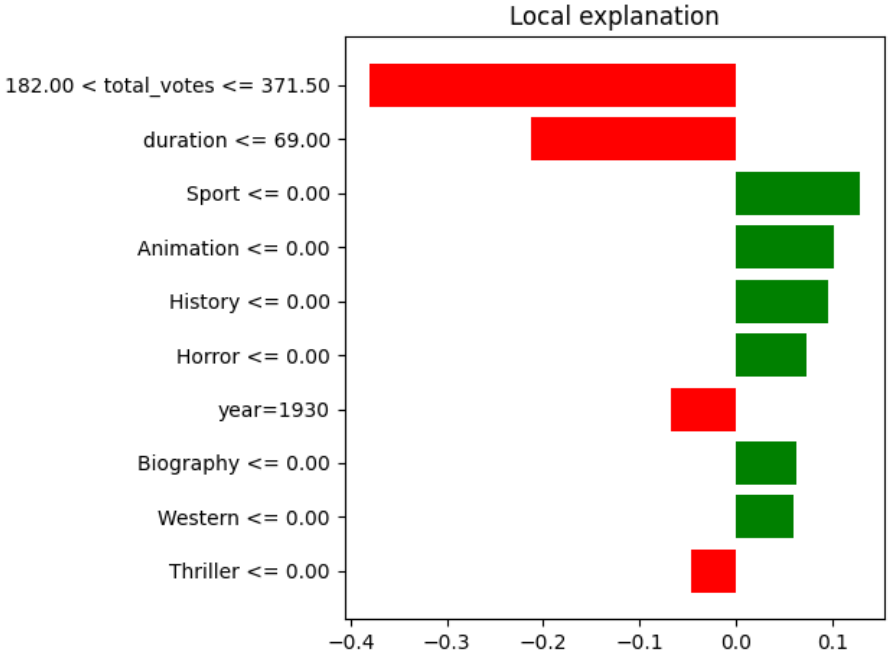

Figure 18: LIME

We can use this model for explaining what ML-models are doing. LIME is model-agnostic, and can be implemented on any ML-model. Like with any other interpretable model; it is very important to interpret the output correctly (ibid.).

6.3.5 Evaluation

When evaluating if LIME is a good explanation model, we first and foremost rely on the evidence that already exist in the field. In section 3.3.3 we have mentioned five rules as to what makes a good explanation. Besides these, in relation to the LIME model, the following point has been stated:

"If you take non-machine learning experts who have access to explanations, and you ask them to do feature engineering to chuck features that they think are not important and maybe engineer features that may be important they're able to improve the model much better than machine learning experts that do not have access to those explanations." (H20.ai, 2017: 22.33-22.57).

This in and of itself should serve a good enough statement that using the LIME model gives us the interpretability that we have sought to achieve, keeping in mind that our ML-model itself is not very precise.

However, we still wish to highlight through analysis, what we see as the most important features that make LIME serve as a good explanation.

LIME itself is contrastive due to the visualization of feature importance. It shows the features in contrast to each other, and we can understand the importance of one feature to another. Furthermore, it can be contrastive to put several LIME visualizations up next to each other, to show the contrast between predictions.

LIME acknowledges it is a chosen explanation by being local to only one prediction and only highlighting the features that are relevant to that specific prediction. It removes the noise by keeping the focus on the features of that specific instance. What could have made it even more acknowledging of the fact that the specific explanation is chosen to other predictions, would be by showing the other features of the model, which would highlight the features that were important to the specific instance. This is what we have done in our dashboard by first showing general feature importance of the model at the top of the page, and later displaying the LIME-model.

LIME does seem to be specifically targeted towards experts within the field on which the ML-model is being applied, which does require its audience to have some understanding of what LIME does and what the features mean. It is hard to define whether LIME realizes that it is an interaction, and perhaps LIME should be seen more as a model that can be included in an interaction with an audience, and not the interaction itself. Therefore, the job of understanding what interactions are necessary to the specific audience, using LIME, lies with the developer of the explanation.

The truthfulness and consistency with prior beliefs are hard points to achieve for ML-explanation models, and we would argue that LIME is truthful to the specific instance of prediction that it is applied to. LIME does leave out features as explained prior, and this is one of the weak points of the LIME model. This weak point is easily fixed by including a full feature summary with the model to display what was left out.

Consistency with prior belief is not necessarily what LIME strive for. It could be argued that if consistency lies within the prediction, LIME would be a good way of highlighting this consistency. LIME will highlight the models consistency in repeated use of the model over many predictions.

7 Discussion

Throughout this report, we have stumbled upon many things we could have done differently, and many things we can discuss, regarding the outcome of this project.
One of the things we want to discuss, is our choice of model, and our way of using LIME and dashboards as visualisation methods. Next we will discuss sacrificing model accuracy in favor of explainability. Lastly we will discuss a few unintended effects that can occur when developing with an object-oriented approach.

7.1 Selection of interpretable models

There are many factors that play a role when choosing what approach should be used for interpretation. Firstly the approach must be established and whether an intrinsic or a post-hoc interpretation should be developed. Both have their values and have been tested in this project. At first the intrinsic model was developed as a multiple linear regression model. This project found that there were little to no linearity in the dataset that was used. What is relevant is that this 'little to no linearity' was found quite early in the process due to the intrinsic explainability that lies in linear regression. The heatmap based on Pearson made it evident that linear regression was not the appropriate model, which led us to the random forest regression model.

The first experiment that was tested on the random forest regression was displaying trees from the forest, and the specific route for one instance of prediction through these trees. This experiment originated from curiosity, and was not based on an already tested or used methodology. While it came to make sense afterwards why this is not how prediction-explanations are usually done, it did serve as a learning experience that we include in the analysis, because it highlights some crucial aspects of interpretability in ML-models. Displaying the route through the tree for a specific dataset is a post-hoc method that still

restricts the model much like an intrinsic model would, which leaves the model with the downsides to intrinsic models while still being post-hoc. Another aspect of this method is that while it is supposed to visualize local interpretability in a way that is visually appealing, the model does not explain the weight of the features. We concluded that this methodology was untruthful because it offers the illusion of interpretability while not displaying the crucial internals that exist to tree models.

The LIME method is model agnostic which was ideal since we had two different ML-models; both random forest regression and multiple linear regression. LIME uses feature visualizations and statistics, which are crucial points to understanding random forest regression models. It helps the audience understand what steps were taken by the model, and what the weight of those decisions were. Furthermore, LIME offers local interpretability for a single prediction, which is what we strived for in the first experiment visualizing the route traveled of a single prediction. It is important to note that random forest is not an intrinsic model, which is why it is necessary to apply an interpretability algorithm to understand it.

Choosing a pre-existing model that offers high explainability already, also allowed the project to keep a focus on the object-oriented infrastructure of the program. This is one of the boundaries we had to put on the projects to keep a technical focus. The project simply did not allow time for application-level evaluation or human-level evaluation. Function-level evaluation was discussed but was ultimately decided against due to the focus on the infrastructure of the program over the output and front-end of the program.

LIME is, if anything, a reliance on an algorithm to create an explanation. The expressive power of the model relies mostly on feature names and model colors. In the projects use of the model, we chose to keep the built-in representation that came with the LIME import. This displays both feature names as well as their values which helps when several LIME-models are displayed side by side to display the contrast between them. The model is not as translucent as a linear regression might be, but considering how little translucency that lies intrinsically in the random forest, LIME is highly relevant because it highlights these intrinsic parameters of the model and how they affect single local predictions. LIME is highly portable because it treats any model it is given as a black box algorithm. In this project it could then be applied to any model we decided to test. We have not ourselves discovered any problems with algorithmic complexity using LIME, and we will leave it at that, because one of the boundaries of this paper is that it will not go into time complexity of algorithms.

In our dashboard we show multiple LIME models to offer contrastive explanations. Another aspect that lies within the strength of the dashboard is its ability to allow the developer to acknowledge that the explanation is chosen and can communicate this directly to an end-user. Dashboards also have the strength of

making the interaction between developer and audience stronger. An industry standard in data science is using what is called a notebook, where both code and explanations are displayed and the user can run the code themselves. What is so crucial to dashboard development is exactly that it leaves out technicality and allows the developer to communicate directly with an audience through text and models.

The type of models chosen to analyze and visualize the data, are crucial to what the outcome will be. The interpretability is therefore highly dependent on which model is used. LIME is model-agnostic, so we could have chosen some different models, and still have evaluated it by using LIME. It is important to note, that there are other alternative ML-models besides random forest regression and multiple linear regression.
This concludes the discussion of what led to the focus on the LIME models and what laid the groundwork for that development. It also highlights the strengths of the LIME model and using a dashboard in relation to the theoretical chapter on interpretability in section 3.3.

7.2 Sacrifice of model accuracy in favor of explainability

When we started visualizing our random forest trees, we realized that the trees would become very large, and we would have a lot of them. This posed a problem for our code efficiency, as the trees would become large to a point where they slowed down the computational power, and also were difficult to visualize. However, the issue is that the larger our trees are, the more precise and effective their predictions are. As already explained, function-level evaluation serves as a proxy for the quality of the explainability. Therefore, we needed to decide on a maximum depth of our trees, where the efficiency of our predictions were still acceptable, and where it would be possible to visualize the models. How do we decide the maximum depths of our random forest model? As the focus of this project is our object-oriented structure, we wanted to create the most efficient code for the users. The goal of our structure is however to create an explainable AI. Therefore, we could not focus solely on one aspect. The current max depth is five. We decided on this maximum depth of our random forest tree, due to the visualization aspect of our explainability. This depth allowed us to create a meaningful dashboard solution that can be easily interpreted by a user, while keeping an efficient code base.

7.3 Unintented effects in developing with an object-oriented approach

Looking at the unintended effect of our program, seems to naturally lead to a discussion of what the weaknesses of an object-oriented infrastructure are, when developing explainable ML-models. While the object-oriented approach has its strengths when developing explainable ML-models, there are also weaknesses. In the following segment we will try to discuss some of the unintended effects

of the technology we built. That discussion will lead to sub-conclusions on the weaknesses of taking an object-oriented approach to developing explanation models within the field of ML.

The object-oriented approach has allowed for a lot of agility in the development process. The approach allowed for quick changes to different datasets, and quick changes in what ML-models we used, as well as keeping these aspects separate and neatly defined in their own classes. The question we did seem to return to continuously is whether this kind of agility is even necessary when developing explainable ML-models. Usually, the strength of this agility lies in the fact that changes to the program are easy, but ML does not seem to have the same kind of demand for these program changes. ML development is more about allowing that kind of agility, when working with the models themselves. The structure of the program is, in some use-cases, not that important to the ML-model, because tweaking the models themselves and testing on these is so agile in itself.

Another aspect is the explainability of the code itself, splitting the code into classes and objects, is an act of abstraction. This is usually what you want, but when developing ML-models and explanations, the act of abstraction can be the wrong approach. The audience might get a deeper understanding by being exposed to the original code instead of it being split into classes. This is also somewhat evident in the notebook approach that many data scientists take to developing and explaining their models, as is evident by websites such as Kaggle.com where most explanations are presented in notebooks.

Notebooks allows for code and text being used interchangeably and allows for the audience to run and understand the code in segments. If the audience are experts within the field of computer technology, this might very well be a much better way of going about explanations, on top of that models such as LIME can be used. Since the goal of most ML-models might be abstracting to a point where the prediction or value can be translated into statistics, that are understood by non-technical people, the internal understanding of our code and ML-models, might be easier achieved by using tools such, as notebooks instead of taking an object-oriented approach.

Furthermore ML-models are often built using a lot of libraries which means that the program is already based on a lot of abstractions, which means that the object-oriented method results on building abstractions on abstractions. While it is not uncommon to use libraries in development, because the problems of ML usually use the same models, there seem to be a larger number of base libraries in use in ML-development. The problems that arise does not lie in developing the ML-algorithms themselves, but rather working with the models, tweaking its weights and features to create results. To summarize, the unintended effects of the object-oriented approach is reducing interpretability for the developer, as well as over-complicating ML-model code with further abstractions.

Part of the discovery made, was that a developer might not put a lot of thought into why they are not using an object-oriented approach when creating an ML-model. The option of structuring code in other ways might just seem 'easier'. By taking an object-oriented approach to ML, the project highlight some of the strengths that lie within the norms of data science. These being that existing methodologies, such as notebooks, offer a higher interpretability for the developer. The high number of libraries available to the developer does not always need further abstractions - in fact abstracting further can result in less explainable code.

As described earlier in this paper, the object-oriented approach can supply with robustness to the code. Robustness can be a requirement, when using ML in real life implementations. When ML are used in bigger perspective and real life productions at companies, it is not enough to hand in a notebook.
MLOps can be said to be an important process, when working with ML in real life. When looking at a workplace, it is not necessarily enough for coworkers and other developers to receive a notebook, since it is not a very clear or elaborate version of the program. Therefore it can be said that the program often has to be object-oriented in a business-related situation. In a business-case, it is also always an advantage to be as efficient and well-organized as possible. MLOps can help with this, since the process focuses on the requirements, without sacrificing the increased automation, or the quality of the models being produced.

To summarize, it can be discussed if OOP is the right approach in relation to build explainable ML-models. This is one of the *lasting unintended effects*, that we discovered and discussed, both while creating the program, and after our iteration was done.

8 Conclusion

One of the biggest takeaways from this project is our discovery of the sheer complexity regarding the field of explainable AI. As the development in the field is constantly evolving, it is increasingly important that the explainability of the ML-models mirrors the technological advances.

As mentioned earlier in this paper, this project can be said to have two overall goals to create an object-oriented infrastructure, and to achieve interpretability through ML-models.
One conclusion we can draw from our work in this project is that the object-oriented approach both has its advantages and disadvantages. One disadvantage might be that the libraries that are available when working with ML, already are highly abstracted. This can have the consequence of making the object-oriented process more irrelevant, and time consuming. In relation to MLOps, an advantage of using an object-oriented structure, is that it might be easier to put the ML-model into a production environment, so that it can become a part

of the bigger technological system.
In the process of developing interpretable ML-models, we have used the CRISP-DM model as a framework, which has been used to keep track of the data mining process. We have concluded that we reached the modeling phase, according to the CRISP-DM model. We did not reach a point where we could evaluate on an application-level or human-level. We did however consider function-level evaluation, by using a proxy for increasing the explanation quality.
We have been using the dashboard technology as a visualization method, to make our ML-models more interpretable, and to include user interaction. After creating our dashboard, we concluded that we could have elaborated more on the dashboard technology, in order to make our dashboard more interactive and explainable. That is why we still consider our ML-models, to be in the modeling phase.
As we ceased our development process in the CRISP-DM modelling phase, we did not reach any of the three evaluation levels about interpretability. Therefore we can not decisively conclude whether or not we have reached a satisfactory level of explainability. We can however, still conclude that our dashboard solution has increased the interpretability of the ML-models to some extent.
We also reached the conclusion that the post-hoc methodology of displaying the routes in our random forest regression trees, was not the most suitable way of increasing interpretability, as it did not display the crucial inner workings of the ML-model.
Likewise the accuracy of our ML-models took a backseat as we developed the dashboard, therefore the ML-models that we did produce did not achieve accuracy of any significance at this point.
In summary, we conclude that an object-oriented approach to developing ML-explainability through a dashboard is a viable development option that comes with its own strengths and weaknesses. We found that the strengths that already lie within OOP remain relevant when developing ML-explainability solutions. We also found that developing within this field can have different requirements than software development, which is where OOP is usually applied. Therefore OOP has its place in developing these solutions, but careful thought should be applied when choosing this methodology, since we found that its application is probably better for larger systems.

9 Bibliography

Anaconda. (n.d.). *Dash Bootstrap Components.* Anaconda. Last accessed on 05/17/2021 from https://anaconda.org/conda-forge/dash-bootstrap-components

Angius, N., Primiero, G. and Turner, R. (2021). *The Philosophy of Computer Science.* The Stanford Encyclopedia of Philosophy (Spring 2021 Edition). Last accessed on 06/03/2021 from https://plato.stanford.edu/entries/computer-science/?fbclid=IwAR3c1l3DyUu65VuQuv__xbt_X4GsgIM-bNf6JgK58TqR6ZEyJI9ZPeHx1kM#MethLeveAbst

Anurag, D. (2020). *A guide to underrated machine learning algorithms - Alternatives to decision trees and random forest classifiers.* Medium. Last accessed on 05/27/2021 from https://medium.com/analytics-vidhya/a-guide-to-underrated-machine-learning-algorithms-alternatives-to-decision-tree-and-random-forest-6e2f8336d4d5

Anyoha, R. (2017). *The History of Artificial Intelligence.* Science in the News. Last accessed on 05/27/2021 from https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

Alex Clark and Contributors Revision. (n.d.-a). *Image Module.* Pillow. Last accessed on 06/04/2021 from https://pillow.readthedocs.io/en/stable/reference/Image.html

Alex Clark and Contributors Revision. (n.d.-b). *Pillow.* Pillow. Last accessed on 06/04/2021 from https://pillow.readthedocs.io/en/stable

Bank, S. (2020). *graphviz 0.16.* Pypy. Last accessed on 06/04/2021 from https://pypi.org/project/

Christensen, T. B. (2018a). *Drivkræfter og barrierer for udbredelse af teknologier.* PDF from lecture in TSA I, Hum-Tek, 11/22/2018

Christensen, T. B. (2018b). *Model af en teknologi.* PDF from lecture in TSA I, Hum-Tek, 11/19/2018

Clark, D. (2013). *Beginning C Object- Oriented Programming.* England: Springer Science and Business Media.http://bedford-computing.co.uk/learning/wp-content/uploads/2015/10/Beginning-C-Object-Oriented-Programming-2nd-Edition.pdf

CoderzColumn. (2020). *How to Use LIME to Understand sklearn Models Predictions?* CoderzColumn. Last accessed on 06/04/2021 from https://coderzcolumn.com/tutorials/machine-learning/how-to-use-lime-to-understand-sklearn-models-predictions#lime_tabular

Dierker, B. R. (2019). *The Trolley Problem and Self-Driving Cars.* Foundation for Economic Education. Last accessed on 05/28/2021 from https://fee.org/articles/the-trolley-problem-and-self-driving-cars

GeeksforGeeks. (2021). *Python Plotly tutorial.* GeeksforGeeks. Last accessed on 05/17/2021 from https://www.geeksforgeeks.org/python-plotly-tutorial/

Gregersen, E. (n.d). *History of Technology Timeline.* Encyclopedia Britannica. Last accessed on 05/27/2021 from https://www.britannica.com/story/history-of-technology-timeline

Gupta, A. (2019). *One Hot Encoding – Method of Feature Engineering.* Medium. Last accessed 05/17/2021 from https://medium.com/analytics-vidhya/one-hot-encoding-method-of-feature-engineering-11cc76c4b627

H20.ai. (2017). *Interpretable Machine Learning Using LIME Framework - Kasia Kulma (PhD), Data Scientist, Aviva.* Youtube. Last accessed on 06/02/2021 from https://www.youtube.com/watch?v=CY3t11vuuOM&t=1606s&ab_channel=loudavloudav

Hulstaert, L. (2018). *Understanding model predictions with LIME.* Towards data science. Last accessed on 05/27/2021 from https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b

Hutajulu, O. (2019). *Basics of Data Preprocessing.* Medium. Last accessed on 05/17/2021 from https://medium.com/easyread/basics-of-data-preprocessing-71c314bc7188

IBM Cloud Education. (2021). *Machine Learning.* IBM. Last accessed on 05/28/2021 from https://www.ibm.com/cloud/learn/machine-learning

IMDb. (n.d.). *Internet Movie Database.* IMDb. Last accessed on 06/05/2021 from https://www.imdb.com/

JavaTPoint. (n.d.). *Random Forest Algorithm.* JavaTPoint. Last accessed on 06/06/2021 from https://www.javatpoint.com/machine-learning-random-forest-algorithm

Jelsøe, E. (2018a). *Teknologiers utilsigtede effekter.* PDF from lecture in TSA I, Hum-Tek, 11/08/2018.

Jelsøe, E. (2018b). *Teknologiske artefakter og teknologibegrebet.* PDF from lecture in TSA I, Hum-Tek, 11/05/2018.

Jelsøe, E. (2018c). *Teknologiske systemer.* PDF from lecture in TSA I, Hum-Tek, 11/12/2018.

Jørgensen, N. (2018a). *Teknologiers indre mekanismer og processer.* PDF from lecture in TSA I, Hum-Tek, 11/02/2018.

Jørgensen, N. (2018b). *Digital Signatur. En Eksemplarisk analyse af en teknologis indre mekanismer og processer.* http://webhotel4.ruc.dk/~nielsj/research/publications/indre-mekanismer.pdf

Kim, D., Khanna, R. & Koyejo, O. (2016). *Examples are not Enough, Learn to Critisize! Criticism for Interpretability.* Advances in Neural Information Processing Systems. https://beenkim.github.io/papers/KIM2016NIPS_MMD.pdf

Kumar, A. (2020). *Correlation Concepts, Matrix Heatmap using Seaborn: Data Analytics.* Data Analytics. Last accessed on 05/27/2021 from https://vitalflux.com/correlation-heatmap-with-seaborn-pandas/

Leone, S. (2020a). *IMDb movies.csv.* Kaggle. Last accessed on 06/01/2021 from https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset?select=IMDb+movies.csv

Leone, S. (2020b). *IMDB ratings.csv.* Kaggle. Last accessed on 06/01/2021 from https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset?select=IMDb+ratings.csv

Liu, S. (2021). *Artificial Intelligence(AI) worldwide - Statistics & Facts.* Statista. Last accessed on 06/05/2021 from https://www.statista.com/topics/3104/artificial-intelligence-ai-worldwide/

Lucidchart. (n.d.). *UML Class Diagram Tutorial.* Lucidchart. Last accessed on 05/17/2021 from https://www.lucidchart.com/pages/uml-class-diagram

Lu, H. (2020). *Machine learning essential.* PDF from lecture in Datascience and Visualization, 03/11/2020

MATLAB (2020). *Machine Learning with MATLAB.* MathWorks. https://se.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf

Maxwell, R. (2015). *Why do we blindly follow our gps?* Geographyrealm. Last accessed on 05/22/2021 from https://www.geographyrealm.com/why-do-we-blindly-follow-our-gps/

Miller, T. (2017). *Explanation in Artificial Intelligence: Insights from the Social Sciences.* Artificial Intelligence 267. Last accessed on 06/03/2021 from https://www.researchgate.net/publication/317821828_Explanation_in_Artificial_Intelligence_Insights_from_the_Social_Sciences

Mirko S. (n.d.). *Linear Regression in Python* Last accessed on 05/13/2021 from https://realpython.com/linear-regression-in-python/

Molnar, C. (2021). *Interpretable Machine Learning.* GitHub. Last accessed on 06/03/2021 from https://christophm.github.io/interpretable-ml-book/

Montgomery, D., Peck, E. & Vinning, G. (2012). *Introduction to Linear Regression Analysis, fifth edition*

Necaise, R. D. (2011). *Data Structures and Algorithms Using Python.* http://home.ustc.edu.cn/~huang83/ds/Data%20Structures%20and%20Algorithms%20Using%20Python.pdf

Novak, G. (2020). *Building a OneHot Encoding layer with Tensorflow.* Towards Datascience. Last accessed on 05/27/2021 from https://towardsdatascience.com/building-a-one-hot-encoding-layer-with-tensorflow-f907d686bf39

Okoh, A. (2019). *Seaborn Heatmaps: 13 Ways to Customize Correlation Matrix Visualizations.* Heartbeat. Last accessed on 05/27/2021 from https://heartbeat.fritz.ai/seaborn-heatmaps-13-ways-to-customize-correlation-matrix-visualizations-f1c49c816f07

Plotly. (n.d.-a). *Basic Dash Callbacks.* Plotly. Last accessed on 05/17/2021 from https://dash.plotly.com/basic-callbacks

Plotly (n.d.-b). *Dashboard API in Python/v3.* Plotly - graphing libaries. Last accessed on 05/27/2021 from https://plotly.com/python/v3/create-online-dashboard-legacy/

Plotly. (n.d.-c). *Dash Layout.* Plotly. Last accessed on 05/17/2021 from https://dash.plotly.com/layout

Plotly. (n.d.-d). *Introduction to Dash* Last accessed on 05/17/2021 from https://dash.plotly.com/introduction

Plotly. (n.d.-e). *Plotly Express.* Plotly. Last accessed on 05/16/2021 from https://plotly.com/python/plotly-express/

Python Software Foundation. (2021a). *os - Miscellaneous operating system interfaces.* Python. https://docs.python.org/3/library/os.html

Python Software Foundation. (2021b). *pandas.* PyPI. Last accessed on 05/19/2021 from https://pypi.org/project/pandas/

Rosa, H. Scheuerman, W. (2010) *High-Speed Society.* Pennsylvania State University Press

Rungta, K. (n.d.). *SciPy in Python Tutorial.* GURU99. Last accessed on 06/04/2021 from https://www.guru99.com/scipy-tutorial.html#1

Sarkar, P. (2021). *Bagging and Random Forest in Machine Learning.* Knowledge Hut. Last accessed on 06/06/2021 from https://www.knowledgehut.com/blog/data-science/bagging-and-random-forest-in-machine-learning

Scikit-learn developers. (n.d.-a). *sklearn.metrics.accuracy_score.* Scikit-Learn. Last accessed on 05/19/2021 from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Scikit-learn developers. (n.d.-b). *Scikit-Learn.* Last accessed on 05/17/2021 from https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html

SciPy. (2021). *Sparse matrices(spicy.sparse).* SciPy. Last accessed on 06/04/2021 from https://docs.scipy.org/doc/scipy/reference/

Seaborn. (n.d.). *An introduction to seaborn.* Seaborn. Last accessed on 05/17/2021 from https://seaborn.pydata.org/introduction.html

Sharma, A. (2020). *Unit Testing in Python* Datacamp. Last accessed on 05/27/2021 from https://www.datacamp.com/community/tutorials/unit-testing-python

Statisticsfun. (2012). *Introduction to Linear Regression Analysis.* Youtube. Last accessed on 05/13/2021 from https://www.youtube.com/watch?v=zPG4NjIkCjc&ab_channel=statisticsfun

Stojiljkovic, M. (2020). *Split Your Dataset With scimitar-learn's train_test_split.* Real Python. Last accessed on 06/04/2021 from https://realpython.com/train-test-split-python-data/

Stringer, Sven (2018) *Feature importance - What's in a name?*. Last accessed on 06.04.2021 from https://medium.com/bigdatarepublic/feature-importance-whats-in-a-name-79532e59eea3

Study RUC (2018a). *Basiskursus 1: Design og konskrutktion I*. Study RUC. Last accessed on 05/15/2021 from https://study.ruc.dk/class/view/15049

Study RUC (2018b). *Basiskursus 2: Subjektivitet, teknologi og samfund I*. Study RUC. Last accessed on 05/15/2021 from https://study.ruc.dk/class/view/15051

Study RUC (2018c). *Basiskursus 3: Teknologiske systemer og artefakter I*. Study RUC. Last accessed on 05/15/2021 from https://study.ruc.dk/class/view/14418

Techopedia. (n.d.). *Data preprocessing*. Techopedia. Last accessed on 05/17/2021 from https://www.techopedia.com/definition/14650/data-preprocessing

The pandas development team. (n.d.-a). *pandas.DataFrame*. Pandas.Pydata. Last accessed on 05/16/2021 from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

The pandas development team. (n.d.-b). *pandas.DataFrameCorr*. Pandas. Last accessed on 06/06/2021 from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

The pandas development team. (n.d.-c). *Series*. Pandas.Pydata. Last accessed on 05/16/2021 from https://pandas.pydata.org/docs/reference/series.html

The SciPy community. (2021). *NumPy: the absolute basics for beginners*. Numpy. Last accessed on 06/01/2021 from https://numpy.org/doc/stable/user/absolute_beginners.html

Turek, M. (n.d.). *Explainable Artificial Intelligence (XAI)*. Darpa. Last accessed on 06/02/2021 from https://www.darpa.mil/program/explainable-artificial-intelligence

Tutorialspoint. (n.d.-a). *NumPy - Matplotlib*. Tutorialspoint. Last accessed on 05/17/2021 from https://www.tutorialspoint.com/numpy/numpy_matplotlib.htm

Tutorialspoint. (n.d.-b). *Scikit Learn Tutorial.* Tutorialspoint Last accessed on 05/17/2021 from https://www.tutorialspoint.com/scikit_learn/index.htm

Tutorialspoint. (n.d.-c). *UML – Class Diagram.* Tutorialspoint. Last accessed on 05/17/2021 from https://www.tutorialspoint.com/uml/uml_class_diagram.htm

TutorialsTeacher. (n.d.). *Python - Random Module.* TotoruialsTeacher. Last accessed on 06/04/2021 from https://www.tutorialsteacher.com/python/random-module

Vickery, R. (2019). *Python Libearies for Interpretable Machine Learning.* KDnuggets. https://www.kdnuggets.com/2019/09/python-libraries-interpretable-machine-learning.html?fbclid=IwAR1ls48GNxg-FxSgaPCUlCUoa2GMhprpot_arEiWquzIwM72mA1IAqCWsEk

Visengeriyeva, L., Kammer, A., Bär. I., Kniest, A. & Plöd, M. (n.d.). *Machine Learning Operations.* MLOps. Last accessed on 06/03/2021 from https://ml-ops.org/

Visual Paradigm. (n.d.). *What is Unified Modeling Language (UML)?* Visual Paradigm. Last accessed on 05/17/2021 from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Wei, W., & Landay, J. (2020). *ML Interpretability and Intrinsic Models.* https://hci.stanford.edu/courses/cs335/2020/sp/lec3.pdf

Wirth, R. Hipp, J. (2000). *CRISP-DM: Towards a Standard Process Model for Data Mining.* http://cs.unibo.it/~montesi/CBD/Beatriz/10.1.1.198.5133.pdf