TU WIEN Informatics

Exploratives Visuelles System für prädiktives Machine Learning von Eventorganisationsdaten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Maximilian Sbardellati, B.Sc.
Matrikelnummer 01526262

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr.techn. Manuela Waldner, M.Sc.
Mitwirkung: Dipl.Ing. Sophie Grünbacher
           Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

Wien, 1. Oktober 2021

Maximilian Sbardellati	Manuela Waldner

TU WIEN Informatics

Exploratory Visual System for Predictive Machine Learning of Event-Organisation Data

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Maximilian Sbardellati, B.Sc.
Registration Number 01526262

to the Faculty of Informatics

at the TU Wien

Advisor: Dr.techn. Manuela Waldner, M.Sc.
Assistance: Dipl.Ing. Sophie Grünbacher
Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller

Vienna, 1st October, 2021

_____   _____
Maximilian Sbardellati   Manuela Waldner

Erklärung zur Verfassung der Arbeit

Maximilian Sbardellati, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Oktober 2021

_____
Maximilian Sbardellati

Danksagung

Ich möchte meiner Familie und meinen Freunden dafür danken, dass sie mich während meiner Studienzeit unterstützt haben. Ihr seit eine große Stütze in meinem Leben und ohne sie wäre der Abschluss dieser Arbeit nicht möglich gewesen.

Ich möchte mich auch bei meiner Betreuerin Manuela Waldner bedanken, die mir in schwierigen Situationen immer mit Rat und Tat zur Seite stand und mir half, andere Perspektiven als meine eigene zu sehen. Außerdem danke ich Meister Eduard Gröller für die Beratung in der Endphase dieser Arbeit.

Ein großes Dankeschön geht auch an meine Kollegen von der DatenVorsprung und Absolut Ticket, die mir ein Umfeld geboten haben, das die Arbeit an diesem Projekt zu einer angenehmen Erfahrung gemacht hat. Ein besonderer Dank geht an Zahra Babaiee und Sophie Grünbacher, die mit mir an diesem Projekt gearbeitet haben.

Acknowledgements

I would like to thank my family and friends for supporting me during my years of study. A special thanks goes to Luise. You all are big pillar in my life and finishing this thesis would not have been possible without you.

I also want to thank my supervisor Manuela Waldner for always giving great advice in difficult situations and helping me to see other perspectives than my own. Additionally, I am thankful to Meister Eduard Gröller for providing counselling during the final stages of this thesis.

A big thank you also to my colleagues from DatenVorsprung and Absolut Ticket for providing me with an environment that made working on this project an enjoyable experience. A special thanks goes to Zahra Babaiee and Sophie Grünbacher for working on this project with me.

Kurzfassung

In den letzten Jahren hat die Verwendung von Modellen des maschinellen Lernens (ML-Modelle) und insbesondere von tiefen neuronalen Netzen außerhalb der Forschungswelt stark zugenommen. Eine der größten Herausforderungen bei der Arbeit mit ML-Modellen ist die korrekte und effiziente Interpretation der von einem Modell gelieferten Ergebnisse. Außerdem ist das Verständnis wie das Modell zu Schlussfolgerungen gekommen ist selbst für Experten auf dem Gebiet des maschinellen Lernens eine sehr komplizierte Aufgabe. Für Laien sind ML-Modelle oft nur Black-Boxes, die nicht nachvollziehbare Ergebnisse liefern. Das mangelnde Verständnis eines Modells und seiner Argumentation führt dazu, dass Benutzer den Modellvorhersagen oft nicht vertrauen.

In dieser Arbeit arbeiten wir mit einem ML-Modell, das auf Eventorganisationsdaten trainiert wurde. Das Ziel ist es, ein exploratives visuelles Eventorganisationssystem zu erstellen, das es Event-Organisatoren ermöglicht, effizient mit dem Modell zu arbeiten. Die Hauptziele des Benutzers in diesem Szenario sind die Gewinnmaximierung und die Möglichkeit, sich auf die vorhergesagte Besucherzahl vorzubereiten. Um diese Ziele zu erreichen, müssen Nutzer in der Lage sein, Aufgaben, wie das Interpretieren der aktuellen Vorhersage und das Durchführen von **was-wäre-wenn Analysen** um die Auswirkungen von Parameteränderungen zu verstehen, zu erfüllen. Das vorgeschlagene System beinhaltet angepasste Versionen mehrerer moderner modell-agnostischer Interpretationsmethoden wie Partial Dependency Plots und Case-based Reasoning. Da modell-agnostische Methoden unabhängig vom ML-Modell sind, bieten sie eine hohe Flexibilität.

Viele State-of-the-Art Ansätze zur Erklärung von ML-Modellen sind oftmals zu komplex um von Laien verstanden zu werden. Bei unserer Zielgruppe, den Event-Organisatoren, kann nicht davon ausgegangen werden, dass sie über außreichendes technisches Wissen im Bereich des maschinellen Lernens verfügen. In dieser Arbeit wollen wir Antworten auf die folgenden Fragen finden: Wie können wir ML-Vorhersagen für Laien verständlich visualisieren? Wie können die Vorhersagen miteinander verglichen werden? Wie können wir Nutzer dabei unterstützen, Vertrauen in das ML-Modell zu gewinnen? Unser Eventorganisationssystem wird mit Hilfe eines **Human-Centred Design** Ansatzes erstellt, wobei während des gesamten Entwicklungszyklus mehrere Fallstudien mit potenziellen Nutzern durchgeführt wurden.

Abstract

In recent years, the usage of machine learning (ML) models and especially deep neural networks in many different domains has increased rapidly. One of the major challenges when working with ML models is to correctly and efficiently interpret the results given by a model. Additionally, understanding how the model came to its conclusions can be a very complicated task even for domain experts in the field of machine learning. For laypeople, ML models are often just black-boxes. The lack of understanding of a model and its reasoning often leads to users not trusting the model's predictions.

In this thesis, we work with an ML model trained on event-organisation data. The goal is to create an exploratory visual event-organisation system that enables event organisers to efficiently work with the model. The main user goals in this scenario are to maximise profits and to be able to prepare for the predicted number of visitors. To achieve these goals users need to be able to perform tasks like: interpreting the prediction of the current input and performing **what-if analyses** to understand the effects of changing parameters. The proposed system incorporates adapted versions of multiple state-of-the-art model-agnostic interpretation methods like partial dependence plots and case-based reasoning. Since model-agnostic methods are independent of the ML model, they provide high flexibility.

Many state-of-the-art approaches to explain ML models are too complex to be understood by laypeople. Our target group of event organisers cannot be expected to have a sufficient amount of technical knowledge in the field of machine learning. In this thesis, we want to find answers to the questions: How can we visualise ML predictions to laypeople in a comprehensible way? How can predictions be compared against each other? How can we support users in gaining trust in the ML model? Our event-organisation system is created using a **human-centred design** approach performing multiple case studies with potential users during the whole development circle.

Contents

Introduction

With the accessibility of data rising year by year, the usage of machine learning (ML) models in day to day business is gaining importance. In this thesis, we are focusing on predictive machine learning using event-organisation data. When organising events like concerts or theatre shows, the goal of the event organiser is to maximise profit. Profit depends on a multitude of parameters including: ticket sales, the occupancy rate of the venue, costs for renting a venue, personnel costs, the artist's salary and more. When organising an event, one has to make several decisions that can influence the profit, such as: Which venue do I choose? What is a good name for the event? At which date should the event happen? How much are the tickets? How do I advertise the event?

To optimise these decisions and set ideal event parameters is a complicated task and event organisers can often only rely on their experience and domain knowledge. This leaves a lot of room for improvement since it is not possible to know if the right parameters were chosen until the event is over. Early ticket sales can indicate trends but it could be too late to change parameters at this time. Additionally, it is often difficult to single out the parameters that lead to low ticket sales. We propose an **exploratory visual event-organisation system (EVEOS)** that allows event organisers to compare different parameter settings and guides them to optimized solutions. The EVEOS is built on top of a predictive machine learning model whose purpose is to predict the number of tickets sold for several given event parameters.

Working in an ML environment is not easy. The most important task for users in such an environment is arguably the interpretation of the ML model results [76, 80]. This raises questions like: What is the objective of the prediction of this model? What information can I gather from the prediction result? Additionally, for the user group event organisers, it is most likely the often first time that they are working with data that is the result of an intelligent model and therefore does not necessarily reflect reality. This could lead to the user exaggeratedly mistrusting the results. Or, as Lim [44] states, the following questions could arise: Why did X happen?, Why not Y?, What happens if I do Z? and

1

How do I make X happen?. Hohman et al. [32] state in their survey that visualisation is one of the best concepts to understand why and how machine learning models come to their decisions, which leads to increased trust in them.

Building **trust** in the results of predictive machine learning is a task well known to the visualisation community. Zhao et al. [80] use the example of how doctors who use the output of ML models in their diagnosis can't simply trust the model without understanding the reason for a given result. It is especially hard to design visualisation systems that are understandable to a broad audience. In recent work, Wexler et al. [76] state that they tried to achieve this but realised that their approach needs at least some knowledge in data science. Ribeiro et al. [64] discuss that the users' trust in a model depends heavily on how well they understand model predictions. So what makes gaining trust in a model and interpreting it so difficult?

As Lipton [45] states, in the real world, humans usually expect an explanation from their counterpart for why certain decisions were made. In general, we want to understand the reasoning behind them. To get such reasoning from ML models, it is necessary to be able to **interpret** them. Lipton criticises that model interpretability is an ambiguous term and is often not properly defined by authors. One definition that is often proposed is that interpretability is a means to gain trust, with trust again being a subjective feeling [36, 63]. When talking about interpreting an ML model, it is also important to differentiate between what Lipton calls **transparency** and **post-hoc explanations**. A transparent model enables us to understand how the model works while post-hoc explanations like explanation by example aim to extract additional information about the model which does not necessarily explain precisely how the model works. Lipton argues that this is similar to humans, since the process of how we make decisions can be distinct from how we explain them. One of the advantages of post-hoc explanations is that developers do not have to sacrifice model complexity or output quality just to keep the model interpretable. Goldstein et al. [27], for example, discuss interpretability of black-box models. Although such models can be very precise, it is very hard to comprehend how the model uses the input to compute the prediction. When looking for transparency in this example, the only way would be to reduce the complexity of the model which in turn would most likely result in a loss of precision.

Doshi and Kim [23] state that there is little consensus on what interpretability in machine learning is. They define it as: "*the ability to explain or to present in understandable terms to a human*". Doshi and Kim argue that problem formalisations are often incomplete because of e.g.: lack of training data, insufficient scientific understanding of the problem, or multi-objective trade-offs. Interpretability or explanations are one way to help us overcome this incompleteness. In contrast to Lipton [45], Doshi and Kim differentiate between **global** and **local** interpretability. Global interpretability is about understanding the general structure and patterns of a model, while local interpretability aims to explain why a single decision was made. They also point out that the context in which the model is used is important. Time constraints and user knowledge can be important factors in how detailed an explanation can or has to be.

One big aspect of being able to interpret ML models is to gain trust in them. Caruana et al. [21] state that for being able to trust a model, it needs to be able to explain its reasoning. If that is not the case, user acceptance is very likely to be low. They also raise the point, that modern (in 1999) models, like neural networks (NN) or large decision trees, are becoming increasingly more complex and therefore harder to comprehend. With the emergence of deep NN and convolutional NN, the complexity has only increased in recent years. To overcome this, Caruana et al. suggest not to focus on understanding the model itself but try to understand the predictions using **case-base reasoning** where one investigates which collection of inputs leads to equal or similar outputs.

In summary, the interpretability of ML models is a broadly discussed topic in the field of ML. Many different definitions are floating around in the community, but most of them boil down to the following three tasks that interpretability needs to serve:

T1 Visualise the prediction result of an ML model in a comprehensible way.

T2 Explain to users why an ML model converts a given input into an output so they understand, to a certain extent, how they relate to each other.

T3 Generating trust in the model to such a degree that the user can work with it with good conscience.

1.1 Aim of the Thesis

The goal of this thesis is to develop an exploratory visual event-organisation system (EVEOS). It allows event organisers without any prerequisite knowledge in computer science to efficiently work with the results of an ML model that predicts how many tickets will be sold given a set of event parameters, which comprise several different data types. Organisers deal with categorical data like the venue and genre as well as with ordered data of different kinds: ordinal data like event dates and quantitative data like ticket prices and venue capacities. The different types of data are described more closely in the Sections 3.1 and 4.1. Additionally, parameters can be directly linked to other parameters. For example, when selecting a date for an event, it directly dictates if the event will happen on the weekend or not. Another example would be the selection of an event venue, which determines the country and city of the event as well. Between all these different types of data, the goal of event organisers is to find a combination of parameters that maximise profit.

Therefore, we strive to implement an EVEOS that allows us to work with all these data types and visually encodes them in a way that fits their specific characteristics. The resulting system should provide possibilities for users to compare the model outputs of different parameter configurations in the sense of **what-if analysis** and **case-based reasoning**. In general, this approach should lead to users being able to interpret the output of the underlying predictive ML model and therefore enable them to build trust in using this system. We also have to always keep in mind that the target group of our

exploration system are mostly laypeople in the field of ML, which influences the set of possibilities we have when striving for model interpretability.

The design of our EVEOS was greatly influenced by the following questions which are based on finding optimal parameters for a future event and therefore maximising ticket sales:

- For a given set of input parameters, how many tickets will be sold for the event?

- What is the probability of selling a given amount of tickets? - The model predicts that 500 tickets will be sold, but what is the probability of 600 tickets being sold?

- How does changing a single parameter influence the ticket sale? - For example, how does the ticket price affect sales, or does not matter which day of the week the event is happening?

- Which parameters should be changed to increase the number of sold tickets? - Are there parameters that have a greater impact than others?

To help event organisers get answers to these questions, we developed our EVEOS accordingly. To provide a certain level of generality and to avoid tailoring our work too much towards the use case of event organisation, we defined four general questions that our system should be able to display answers to:

Q1 What is the prediction of a model with the current input?

Q2 How certain is a prediction?

Q3 Which effect does changing parameters have?

Q4 Which parameters cause the biggest change in the result?

1.2 Methodological Approach

The questions we aim to answer from an information visualisation point of view to perform the presented tasks **T1-T3** of interpretable ML models are:

V1 How can we visualise the results of predictive machine learning models so that they are useful and comprehensible to non-expert users?

V2 How can we make multiple predictions comparable against each other?

V3 How can we support users in gaining trust in the predictions using model interpretation methods?
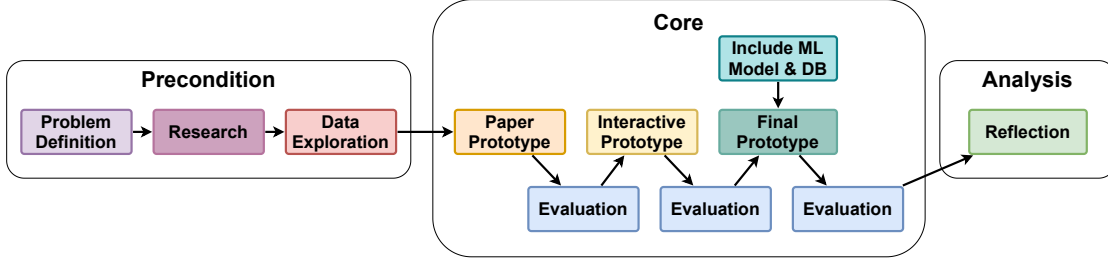
Figure 1.1: Design Timeline.

To achieve a visualisation design that is as comprehensible as possible, we employ the **what-why-how** analysis framework introduced by Munzner [52]. This framework helps us to find the appropriate visual encoding for each of the incorporated data types.

To answer the second and third questions, we incorporate interactive analysis features allowing the user to interact with the model and thereby gaining a better understanding of and more trust in the model. Specifically, the relationship between the input features and the resulting prediction is highlighted. To increase the generality of our work, we use model-agnostic methods, which fall into the category of post-hoc explanations [46], wherever we can. This way, we are independent of the model structure, which allows us to update the model rather easily. As described by multiple works [76, 66, 80, 40], the measurement of partial dependence is a model-agnostic method that is suited well for **what-if analyses**. By allowing the user to test their hypothesis by changing the input parameters in a graphical user interface, we can answer questions like: How is the result influenced if I change a certain parameter? By how much does an input value have to change to provoke a change in the prediction? This also includes providing the means to quickly identify the local influence of parameters on the current prediction. In general, we focus on providing local interpretability of the model as discussed by Doshi and Kim [23]. Additionally, showing the user historical ground truth results and corresponding input data of similar instances provides case-based reasoning. As described by Zhao et al. [80], this is a well-known concept in decision making. By connecting the predictive machine learning models with this familiar method, we aim to generate more trust in our models.

To achieve the implementation of an EVEOS that satisfies the needs of our target group, we loosely follow Sedlmair et al.'s [67] nine-stage framework for design studies. As illustrated in Figure 1.1, the nine stages are grouped into three parent-stages: **precondition**, **core** and **analysis**. The **precondition phase** deals with learning about the problem and state-of-the-art solution in the problem domain (see Section 2.3). We also perform a set of data exploration tasks to get a better understanding of the data we are working with.

In the **core phase**, the work of implementing a solution for the problem is handled. Here, we deploy the strategy of the user-centred design cycle, the methodology of which is discussed in Section 2.1. In our project, we perform three iterations of the design cycle. Each step is evaluated by multiple potential users in a case study environment. First, we

derive a paper prototype according to the given use case and data types (Chapter 4). The main focus here lies on finding the best visual encodings for different tasks at hand and finding a first layout for the system that allows users to fulfil the tasks they need to perform. Next, an interactive horizontal prototype that gives a first complete view of the EVEOS is developed (Chapter 5). This prototype is not connected to real data but only uses sample data with interactions producing mock results. The evaluation at this stage aims to verify the correct implementation of the given user stories. Finally, a vertical prototype is developed that connects the previous one with a database and the ML model to show actual predictions. In this last step, the overall performance of the system is evaluated (Chapter 6). In the **analysis stage** in Chapter 7, we reflect on the results of our work. This stage of the framework also includes writing down the findings of the design study.

The main contribution of this thesis is the conception, development and evaluation of an exploratory visual event-organisation system on top of a predictive ML model for event organisers who, until now, have not been able to perform such analyses with their domain-specific data. Within the iterative design process, we aim to learn more about the needs of laypeople in the field of machine learning in regards to model interpretability. We strive to grasp which concepts in the field of user interface design work well in the proposed use case and which do not. Additionally, our work can hopefully serve as a best practice example on how to deal with different data types during predictive machine learning and post-hoc model analyses. Special challenges that distinguish our work from other state-of-the-art approaches include the diversity of the data types at hand and the usage of directly linked parameters.

1.3   Outline

To get a better understanding of the topics relevant to this thesis, we discuss related work in Chapter 2. This includes the topics: user-centred design, interacting with user interfaces and ML model analysis. In Chapter 3, the creation process of the used DNN is described. First, we discuss data gathering and exploration, followed by the architecture and training pipeline of the model. Finally, the API that handles communication between the front-end and the model is presented. The three stages of the human-centred design process to develop the proposed exploratory visual event-organisation system are described in the Chapters 4, 5 and 6. In each of these chapters, we discuss the current state of the prototype and the hypothesis we have of the development of the particular stage. Next, we show the general layout of the prototype and, following that, take a more detailed look into its components. Additionally, for Chapter 5 and 6, the changes respective to the previous stage are discussed. As the last step, each prototype stage is evaluated by multiple users in a qualitative user study. We conclude this thesis with Chapter 7, reflecting and summarizing our approach and discussing potential future work.

Background and Related Work

In this chapter, we dive into basic concepts and related work relevant to this thesis. First, we discuss the basic concepts of user-centred design, which serve as a framework for the development process. Next, we discuss what performance requirements are common for interacting with user interfaces (UI). Finally, an overview of various works in the field of ML model analysis is given.

2.1 User-Centred Design

In this section, we describe why it is beneficial to use user-centred design approaches and what basic principles are deployed in such approaches. Our elaborations are based on the work by Kulyk et al. [41] from where we pick the most relevant points relating to our work. First, we need to define what user-centred design is. User-centred design or human-centred design evolved from the field of HCI. Stone et al. [70] define it as:

"*An approach to user interface design and development that views the knowledge about intended users of a system as a central concern, including, for example, knowledge about user's abilities and needs, their task(s), and the environment(s) within which they work. These users would also be actively involved in the design process.*" [70] page 628

In general, the main goal of user-centred design approaches is to heavily involve the user in the development process and adapt to their needs. The International Organization for Standardization (ISO) mentions among others the following elements of user-centred design in ISO 13407 [33]:

- The users, their tasks and requirements are clearly understood and they are actively involved in the project.

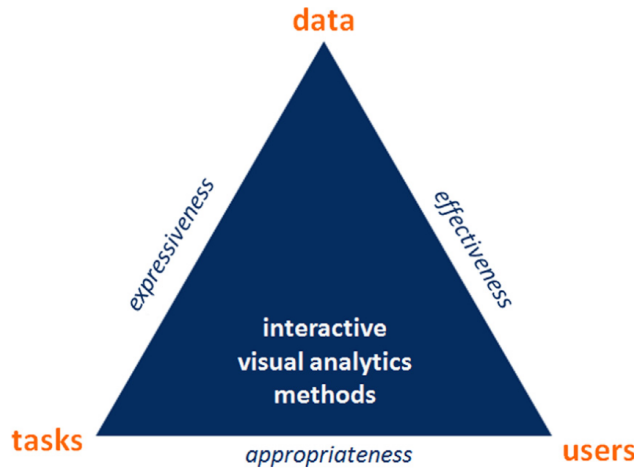- The users play a critical part in giving feedback during the iterations of the development.

Figure 2.1: Design triangle showing the major factors that need to be considered when designing interactive visual analytics methods. From Miksch and Aigner [48].

The ISO 13407 additionally lists several benefits of user-centred design when applied correctly. First and foremost, they state that the quality, aesthetics and impact of the final product, as well as the productivity of the users working with the system, can be improved. Furthermore, since the users actively contribute during development, their discomfort in transferring from an old to a new system can be drastically reduced. But when can one say that a design strategy was a success? Often the answer is that when the software project is usable. But what is **usability**?

Usability is a broadly discussed term and several works elaborate on its main aspects [62], [70]. Bennet [15], for example, mentions four points: learnability, throughput, flexibility and attitude. Nielsen [57] formulates these points a little differently and adds a fifth aspect: learnability, efficiency, memorability, errors and satisfaction. The ISO standard 9241 [34] combines all the different points into three distinct aspects: **effectiveness**, **efficiency** and **satisfaction** (see Figure 2.1). Effectiveness shows how accurate a user is in completing a given task. Efficiency describes the ratio between effectiveness and the amount of work needed to complete the task. Satisfaction is a rather ambiguous aspect that deals with the user's level of comfort and their feelings towards using a system. The importance of each axis depends on the domain of the project. For example, for some tasks, it could be of high importance that they are solved with the utmost precision and the associated workload can be neglected. In this case, efficiency is less important.

Talking to domain experts, we get the impression that, in our case, the most important axes are effectiveness followed by satisfaction. The goal of the users is to find the best parameters for an event and they need to be able to precisely evaluate the parameters. Users satisfaction also plays an important part in our system. The domain experts need to feel comfortable when working with the ML model. In our opinion, the comfort level
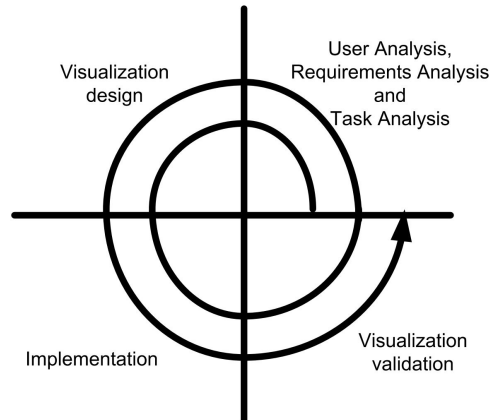
Figure 2.2: The visualisation design cycle as defined by Shneiderman and Plaisant [69]. The four stages of this iterative approach are repeated multiple times until a satisfactory sate is reached.

is greatly influenced by how well the users can interpret the model predictions and how much they trust them.

2.1.1 User-Centred Design Circle

As stated in Section 2.1, we now know that a user-centred design approach is beneficial, aims to let the users contribute during development and the goal is to have a usable product. But how can we structure the project development to achieve all of this? In the past, a lot of work has been put into developing workflows for user interface design. Notable examples of approaches are the do's and don'ts of Johnson [35] and the comprehensive strategies of Mayhew [47], Shneiderman and Plaisant [69], or Lauesen [42]. Kulyk et al. [41] state that user interface design and visualisation design are two closely related fields and therefore it stands to reason that the same strategies can be used in both fields. Our work developing an exploratory visual event-organisation system is located somewhere between those two since it combines a novel user interface with multiple visualisations.

The approach by Shneiderman and Plaisant [69] introduces the so-called **visualisation design cycle** (see Figure 2.2). It features four stages, which are repeated until the result is satisfactory:

1. A thorough analysis of the users and their aims and requirements is conducted (user analysis).

2. The method of visualisation is derived (visualisation design)

3. and implemented (prototyping).

(a) The four nested layers.

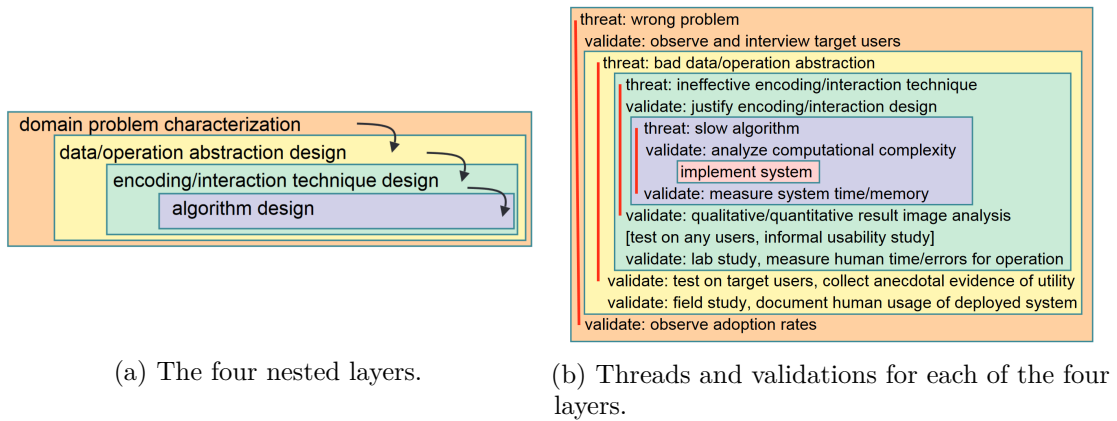(b) Threads and validations for each of the four layers.

Figure 2.3: Nested model for visualisation creation by Munzner [51].

4. The visualisation is evaluated (evaluation).

In his work, Fallman [25] groups the design and implementation steps into a one-step **synthesis**. He also argues that steps can overlap and are sometimes not clearly separable. Munzner [51], proposes a nested model for the design process of visualisations (see Figure 2.3a). She states that the output from a level above serves as the input for the level below. This also has the effect that errors made in upper levels propagate downwards. In the following sections, we will briefly discuss the main aspects of the stages defined by Shneiderman and Plaisant [69].

User Analysis

The first step in the design cycle represents a thorough user analysis. The goal is to collect as much data as possible about the users that will work with the product. This includes investigating who the users are, the goals the users have, the requirements that the system needs to fulfil, understanding the underlying data and also considering external factors like the available hardware or workplace conditions. Often these questions are summarized as [41, 42, 69]:

- Who are the users of the system?

- What kind of data are they working with?

- What are the general tasks of the users?

These questions also are the basis for the data-users-tasks triangle proposed by Miksch and Aigner [48] shown in Figure 2.1. Sometimes users already have some kind of visualisation tool to show the data they use. In this case, it can be better to enhance what they are already used to instead of coming up with new designs that are unfamiliar to the users

[72]. When establishing user requirements, it is essential to categorise them by relevance so it is evident what is most important to the users. Benyon et al. [16] suggest using the MoSCoW schema: **M**ust have, **S**ould have, **C**ould have, **W**on't have. Performing a task analysis determines what features need to be available for the users to fulfil their tasks. Zhang et al. [79] state that a system should always have exactly the features needed by the users, not more, not less. There are multiple methods to gather users tasks, including interviews or task demonstrations, both of which have their pros and cons. Interviewing, which is the most commonly used method, is very time-consuming. Since this leads to a rather small quantity of users being consulted, it is important to do a good selection of users to keep the interviews representative [69].

Answering these questions should lead to a better understanding of the problem domain and helps the developer team create a better visualisation design [24]. As stated by Carroll et al. [20], a user study is an interchanging activity between the developers and the users. For this thesis, we organised a focus group meeting with several event organisers to talk about their expectations and needs regarding the proposed EVEOS.

Visualisation Design

When creating a visualisation design, it is important to first decide what purpose it serves. Usually, visualisations can either be used for presentation or analysis. A visual presentation of data is used to communicate the properties of the data to the viewer. In our work, we strive to present the predictions of an ML model. Here, the most important thing is that the data is presented in a way that is understandable by all viewers since interaction and exploration to gain additional insight is limited. Visualisation for analysis allows the user to interact with the data more deeply. By allowing data exploration and manipulation, users gain a deeper insight into data that helps them solve the tasks they have [41].

Prototyping

Since the user-centred design cycle includes multiple iterations of analysis, design and implementation, the artefacts that resolve from one iteration can be viewed as a prototype. A prototype is an intermediate version of the product that can only present a subset of the features of the final product. Prototypes often are used in usability testing and evaluation of the current project status [41]. Depending on how advanced the project is, different types of prototypes can be used [42].

In the early stages of a project, **sketches** and **paper prototypes** can be of much use. By simply drawing on paper or creating paper models, designers can quickly try multiple ideas without having to implement anything. **Screen prototypes** can already consist of real software components but are still not functional. Interaction is in most cases not possible, but the prototype can give the users a first impression of how the final product might look. There are two types of functional prototypes: **Horizontal prototypes** focus on providing interactivity for as many components as possible, but the processing
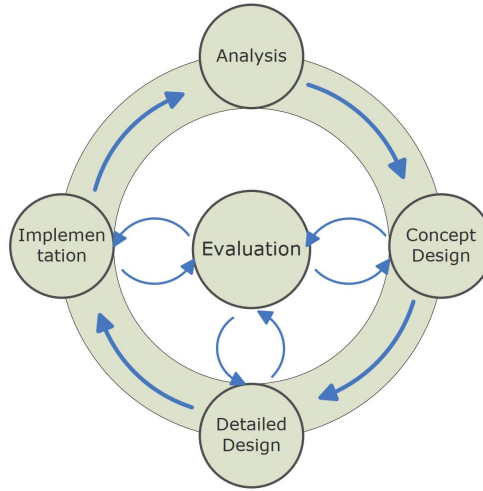
Figure 2.4: Human centred design cycle by Kulyk et al. [41]. In comparison to the design cycle proposed by Shneiderman et al. [69] in Figure 2.2, evaluation is moved to the centre of the circle and is repeated whenever necessary.

behind them might not be fully functional yet. Depending on the grade of functionality, a similarity to screen prototypes can be observed. **Vertical prototypes** on the other hand, provide full functionality for selected features.

During the design process of the EVEOS, we first develop a digital paper prototype. Next, we implement a horizontal prototype that shows all components but is not connected to the database or ML model. Finally, we connect the back-end to provide a fully functioning prototype. Each of these stages is evaluated by several domain experts in a round of interviews.

Evaluation

One of the key features of user-centred design is to involve users and evaluate completed steps as early as possible. For example, once the first paper prototypes are developed, they should already be subject to evaluation. This way, mistakes and misunderstandings can be eliminated early on and the relevance of the project can be ensured [30]. To highlight the importance of evaluation, Kulyk et al. [41] changes the design cycle proposed by Shneiderman et al. [69] and moves the point of evaluation into the centre, with it being connected to all other stages as shown in Figure 2.4.

There are multiple ways to conduct evaluations. **Analytical methods**, for example, are used mostly in the early stages of development and are often conducted by usability experts and not the users. Techniques like cognitive walkthroughs, early concept evaluations with focus group meetings or interviews are used to validate the first concepts [50]. The most commonly used types of evaluation are **empirical methods**. They are often performed

as experiments that try to measure usability data (see Section 2.1). Kulyk et al. [41] criticise that such experiments are often conducted in artificial settings and the tasks performed by the users are highly engineered. This leads to the results being more focused on the visual representation and less on the user experience when working with the provided interface.

In her nested model, Munzner [51] suggests that each of the layers has its threats that need to be considered in its validation. As illustrated by red lines in Figure 2.3b, some threats of outer levels are connected to validations in inner layers, meaning that they can only be properly evaluated once the inner layer is reached. Munzner also states that, usually, in a single work one focuses only on a subset of the layers. We see our work to be in the orange and yellow layer of the nested model. Munzner [51] suggests employing more qualitative validation methods for these layers rather than empirical experiments.

When conducting evaluations, Kulyk et al. [41] state that it is important to plan them. On the one hand, the content of the evaluation needs to be defined, so it is evident what the questions are that are supposed to be answered. According to the selected questions, the developer team then needs to decide what to show the users. Do they get to see a preliminary prototype, just sketches, or a combination of multiple artefacts? Sometimes artefacts are not in a state where they can be used for proper evaluation and need to be adjusted before the experiment. Additionally, the time frame and workload need to be discussed to allow the team to create a budget and decide if the evaluation is feasible.

During a validation round, interviews or questionnaires can be used to collect feedback. Questionnaires are easy to evaluate, but often do not go into much detail. Performing interviews allows the gathering of more qualitative data since they can be in a semi-structured format. The downside of this is that the answers can be hard to analyse and interpret [41].

2.2 Interacting with User Interfaces

In this section, we discuss the impact that waiting time (or response time) has on user satisfaction when working with interactive UIs. Waiting time is defined as the time between a user action like clicking on a button and the result of the action being shown e.g. a graphic is computed and then displayed. Research on acceptable response times in HCI has been done for over 50 years, with first studies being published as soon as 1968 [49].

Depending on the type of ML algorithm used and the number of predictions that need to be made simultaneously, the time needed to compute results can vary. Especially when using post-hoc explanation approaches, like partial dependence [26] or feature importance [71], where measurements are computed by comparing multiple predictions, the response time often is a few seconds. Therefore, we investigate the influence that waiting time has on the user experience and how we can deal with it properly.
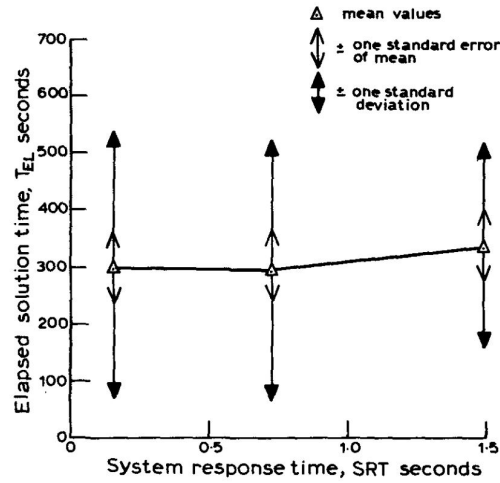
Figure 2.5: Elapsed Solution Time (T_{EL}) against the SRT. The tasks were solved with set SRTs of 0.16, 0.72 and 1.49 seconds. [28]

In their work in 1978, Goodman and Spence [28] discuss how system response time (SRT) affects users productivity. They performed experiments in the field of computer-aided engineering where enhancing productivity is one of the driving factors of using computers. Users had to adjust multiple parameters to solve a task. When performing these actions, the SRT was fixed to either 0.16, 0.72 or 1.49 seconds and the total time of solving the task was used to measure the impact of the SRT. To get genuine results, the total waiting time was subtracted from the solution time. As shown in Figure 2.5, the results of the experiment suggest that users take relatively longer to solve a task when the individual SRT for actions during the task is longer.

In his work, Shneiderman [68] asks the question: **How long will users wait for the computer to respond before they become annoyed?** He argues that the time until users are frustrated waiting for the system response is highly dependent on multiple factors. One important factor is if the user expects to be waiting for a result after acting. The expected waiting time for the action "go to next page" is likely to be much lower than for "download 100GB of data". If the user has experience performing a certain action, established expectations can also be a big factor in how long it takes for the user to be annoyed. Shneiderman reviewed several studies on this topic [28, 75, 31, 74, 17] and they all support the statement by Goodman and Spence [28] that, in general, productivity increases with lower response times.

Nielsen talks about the importance of response time in user interface design in various works [56, 58]. He also raises the argument that for some actions the response time should be artificially slowed down because the users cannot cope with the speed. When scrolling through a list, for example, the scrolling speed should be slow enough for the user to be able to recognise when the wanted element is found. In general, Nielsen defined three

stages of waiting time [56]:

- **0.1 seconds**: If the response for an action is available in 0.1 seconds or faster, the user feels he is directly manipulating objects in the UI.

- **1 second**: The user realizes that there is a delay. There is no feedback necessary because the user still feels in control. Navigation actions in an interface should always take a maximum of 1 second.

- **10 seconds**: In a UI, no common action should take longer than this, because the users start to feel bored.

Nielsen also states that when waiting for more than 10 seconds, a percent-done indicator [53] should be used. If it is not possible to compute a percentage of work already done (e.g. 20 out of 100 files loaded), one should at least try to show the total amount of work (e.g. 20 files loaded . . .). If that information is also not available, Nielsen suggests using repetitive loading symbols like spinning balls. He states that loading symbols give users something to look at, which is a factor that should not be underestimated when talking about perceived waiting time. Additionally, when waiting between 2 and 10 seconds it is often not necessary to show a true percent-done indicator, because the user cannot read the information in the time available.

Loading symbols are the main focus of the work of Kim et al. [37], where they investigate how different symbols and progress functions affect the perceived waiting time. First, they state that the best and obvious way is to reduce the actual waiting time. In their experiments they tested 48 different symbols options derived from (see Figure 2.6): 3 durations (5, 10, 20 seconds) $\times$ 4 progress functions (repetitive, linear, power, inverse power) $\times$ 2 shapes (bar, circle) $\times$ 2 embellishments (none, bike with flag). Waiting durations below 5 seconds were not tested since they are considered too short, which aligns with Nielsen's work [56]. The results of their work indicate that the duration and progress functions have the highest impact on perceived response time. Using a repetitive progress function that does not give any indication about the progress being made performs the worst, followed by a linear progress function. The power (start slow, then fast) and inverse power (start fast, then slow) functions resulted in the lowest perceived waiting time. The shape of the symbol and the use of embellishments did not have a significant effect. Kim et al. state that the repetitive circle that is used in a vast majority of online video players performed the worst of all options.

All operations in our work that require the usage of a loading symbol are calls to external APIs where we cannot get any information about how long the response time is. We also cannot get intermediate results so we could display how much of the total work is already done. Therefore, we cannot employ percent-done indicators as suggested by Nielsen [56, 58] or the power functions used by Kim et al. [37]. In this case, Nielsen advises using repetitive loading symbols. Even though they are not the optimal choice in general, users at least know that the system is waiting for a response and did not

Progress Function (4 levels)	Shape (2 levels)	Embellishment (2 levels)	
		Unembellished	Embellished
Repetitive	Bar		
	Circle		
Linear	Bar		
	Circle		
Power	Bar		
	Circle		
Inverse Power	Bar		
	Circle		

Figure 2.6: The different loading symbols tested by Kim et al. [37].

crash. As discussed in Chapter 6, we employ repetitive circles when we are waiting for the response of an API call.

2.3   Machine Learning Model Analysis and Interpretability

In recent years, a lot of research has been done in the area of visually analysing machine learning models and parameters as shown in the surveys by Hohman et al. [32] and Sedlmair et al. [66]. Hohman et al. focus on answering the questions Why?, Who?, What?, How?, When? and Where? in the context of visual analytics in deep learning. A prominent answer to the question Why? is **Interpretability & Explainability**.

The work by Sedlmair et al. [66] concentrates on answering the questions What? and How? regarding visual parameter space analysis. The tasks they describe include:

interactive parameter optimisation, output uncertainty and parameter sensitivity. A notable point they present is that the visualisation of uncertainty often overwhelms users and is therefore only shown when explicitly requested. They propose several techniques to enable the user to perform these tasks, including **local-to-global, global-to-local** and **informed trial and error**. Informed trial and error allows the user to refine the input parameters until a satisfactory result is achieved.

When talking about machine learning model analyses and making them interpretable, we need to differentiate between two types: interpretable models and model-agnostic explanations. **Interpretable models** are models that are constructed in a way that one can understand how the model derives an output from a given input by simply looking at its structure. A good example of that are small decision trees, where investigating the decision rules at the nodes is usually enough to understand the model. **Model-agnostic explanations** on the other hand, treat the ML model as a black box and try to achieve interpretability by showing diverse measurements only computed by the model output. In their work, Ribeiro et al. [64] discuss the benefits and drawbacks of these types. An evident benefit of model-agnostic approaches is the separation between the model structure and the visualisation. This makes it possible to use different visualisations for the same model. Additionally, treating the model as a black-box makes changing the model trivial. Another benefit of model-agnostic approaches is that the complexity of the model does not influence its explainability as strongly as for interpretable models. For example, if a decision tree grows larger, it gradually gets harder to interpret it. If one relies on model-agnostic measurements, the size of the tree is irrelevant for its degree of interpretability. Therefore, model designers do not have to sacrifice complexity and accuracy for the sake of interpretability. On the other hand, there is still a trade-off between model complexity and interpretability, so it may be hard to gain a global understanding of the model using model-agnostic methods if the model is complex. Interpretable models are preferred when accuracy is not as important as interpretability or when designers can create a small interpretable model that has the same accuracy as a complex black-box model.

The following approaches use a model-agnostic method called **partial dependence** (PD) plot, which was first introduced by Friedman [26] in 2001. Partial dependence plots show how changing the value of a single feature x changes the prediction on average. To compute this, a representative sample s_x of the values of feature x and a set of samples s_y of the complete feature space without x are taken. Then, for each s_x all samples of s_y are completed with the current value of x and fed to the model. The average of the results is the partial dependence value of the current x. Krause et al. [40] introduce the measurement of **local partial dependence**. Here, instead of getting multiple samples s_y, only the currently selected parameter combination is used and combined with all values of s_x. This way, the partial dependence of x is computed only for the current selection.
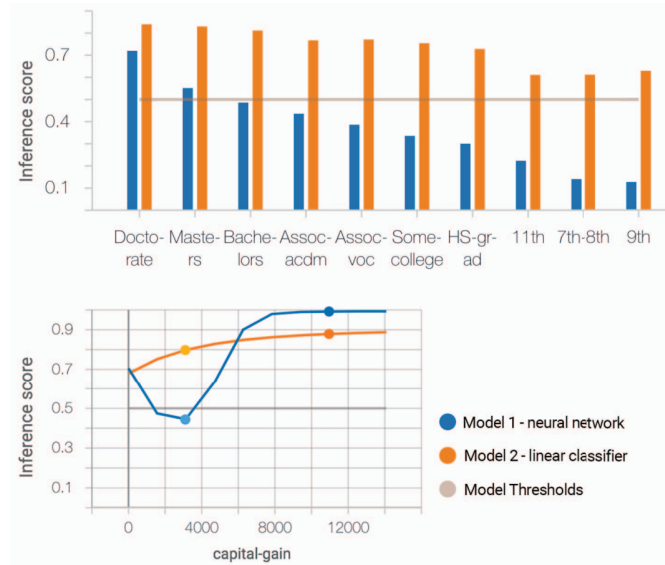
Figure 2.7: Partial Dependence (PD) plots comparing two ML models. Top: categorical PD plot. Bottom: numerical PD plot. [76]

Wexler et al. [76] present an approach called the **What-if Tool**, used to interactively test multiple hypotheses that users might have. **Data point editing** allows users to test how the model output changes by making changes in the input. To allow for a better comparison, the user can duplicate the original input and output. They also introduce **counterfactual reasoning**, which answers questions like: What ground truth input is most similar to the current input that achieves the wanted result? They also point out that to answer this question, numerical features and categorical features need to be treated slightly differently. To show how a single input parameter affects the result, they use **partial dependence plots** as shown in Figure 2.7. The presented task solutions by Wexler et al. can be applied to our tasks in some ways, but the visual representation does not align with the data we are using. Also, as stated by themselves, the incorporated visualisation interface requires at least some knowledge in working with machine learning models, which our users do not have.

In their tool called **IForest**, Zhao et al. [80] define similar tasks as Wexler et al. [76] but focus on analysing random forest models. They also describe the problem of revealing the influence of input parameters on the resulting prediction, which they too solve using **partial dependence** information. Additionally, they compute and show so-called **split points** to show how much an input parameter needs to change to lead to a change in the prediction. Using **case-based reasoning**, they show a collection of ground truth data that corresponds to the current input and the corresponding prediction, giving the user a better understanding of why the model made the current prediction. Since their computation and visualisation is tailored towards random forests, we can only incorporate abstract solution ideas into our work.
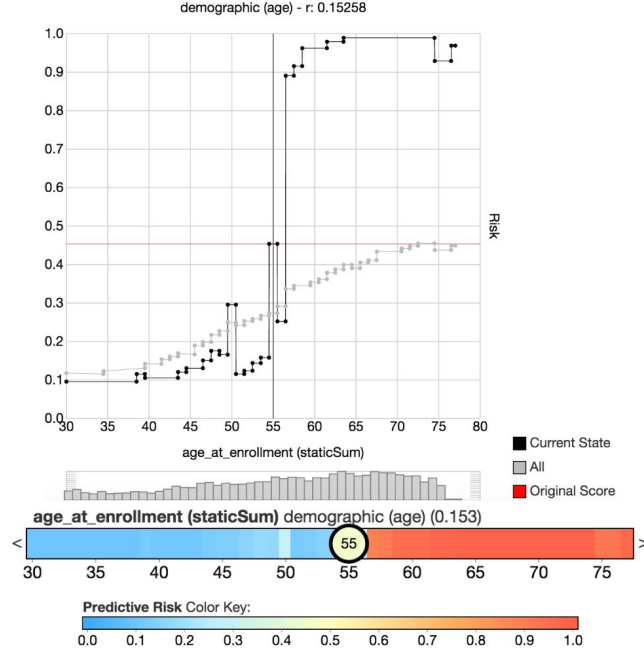
Figure 2.8: A partial dependence plot encoded as partial dependence bar. The values are colour-coded [40].

Krause et al. [40] present a novel encoding for **partial dependence plots** in their tool **Prospector**. First, they describe how they optimise the sampling of the partial dependence measurement by adapting the sampling to the used machine learning approach and the characteristics of the input data. Then, they introduce local partial dependence LPD encoded as **partial dependence bars**, which are interactive sliders showing the range of the input value on the x-axis and the corresponding change in the prediction using colour coding on the slider as shown in Figure 2.8. This encoding allows users to investigate which change in a feature leads to which change in the prediction, while all other features are fixated. Their approach only works if the prediction is a 1D value. Krause et al. only use numerical input values in the partial dependence bars, whereas we want to extend the usage to also incorporate categorical data.

Encoding LPD information on an partial dependence bar as suggested by Krause et al. [40] can also be seen as a type of **scented widgets** as proposed by Willet et al. [77]. They define scented widgets to be: "*graphical user interface controls enhanced with embedded visualisations that facilitate navigation in information spaces*". Adding visual cues to user interface controls can help users in exploration tasks, according to the authors. In their work, Willet et al. state that to add visual cues one has to first select which data should be shown in the cue. In our case, this data would be, for example, LPD information of a feature value. Additionally, it is important to use the correct visual variables to encode the given information. Since interface controls cannot
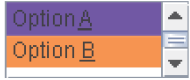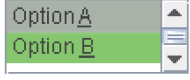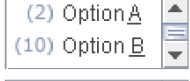
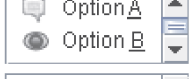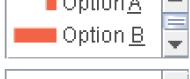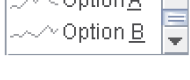| Name	Description	Example
Hue	Varies the hue of the widget (or of a visualization embedded in it)	
Saturation	Varies the saturation of the widget (or of a visualization embedded in it)	
Opacity	Varies the saturation of the widget (or of a visualization embedded in it)	
Text	Inserts one or more small text figures into the widget	
Icon	Inserts one or more small icons into the widget.	
Bar Chart	Inserts one or more small bar chart visualizations into the widget	
Line Chart	Inserts one or more small line chart visualizations into the widget	

Figure 2.9: Scent encodings supported by the work by Willet et al. [77].

be manipulated arbitrarily, one can only apply the following visual variables directly on a widget: hue, saturation, lightness and texture. On the other hand, these are not the best visual variables to encode quantitative data. To enable the usage of variables like position and length, Willet et al. suggest adding small visualisations to the widgets, as shown in Figure 2.9. This approach is similar to how we visualise LPD information for categorical variables, with our small visualisations being more sophisticated than a simple bar or line charts. Willet et al. also propose several guidelines when working with scented widgets. One of them mentions the usage of identifiers like icons, tooltips or legends to add additional information to the cues. We adopt this idea by showing tooltips for LPD information that give details information about the underlying ML model prediction.

A different approach on how to investigate local properties of features is presented by Strumbelj and Kononenko [71]. They propose the measurement of global and local **feature importance**. This is also a model-agnostic approach, which limits all interactions to changing the input and evaluating the result. To define feature importance, they use an adapted version of the Shapely value [78] usually used in the field of game theory. The idea behind feature importance is to compute a value between -1 and 1 that indicates the impact a feature has on the prediction. Global feature importance (see Algorithm 2 of Figure 2.10) computes this value for a value j of feature i by first taking m random samples from the input space. Next, two predictions are made for each sample: One with the original sample and one with the value of feature i being set to j. These predictions are then subtracted and accumulated to finally compute the average of the differences.

Algorithm 1. Approximating $\varphi_i(x)$, the importance of the i-th feature's value for instance x and model f. Take m samples.

$\varphi_i(x) \leftarrow 0$
for $k = 1$ to m **do**
    select (at random) permutation $\mathcal{O} \in \pi(n)$ and instance $y \in \mathcal{A}$

    $\varphi_i(x) \leftarrow \varphi_i(x) + f(x_1) - f(x_2)$

end for
$\varphi_i(x) \leftarrow \frac{\varphi_i(x)}{m}$

Algorithm 2. Approximating $\psi_{i,j}$, the global importance of the i-th feature's value j for model f. Take m samples.

$\psi_{i,j} \leftarrow 0$
for $k = 1$ to m **do**
    select (at random) instance $y \in \mathcal{A}$
    $x_1 \leftarrow$ set i-th feature to j, take other values from y
    $\psi_{i,j} \leftarrow \psi_{i,j} + f(x_1) - f(y)$
end for
$\psi_{i,j} \leftarrow \frac{\psi_{i,j}}{m}$

Figure 2.10: Algorithms by Strumbelj and Kononenko [71] to compute local and global feature importance.

Computing the local feature importance of the i-th feature value of the current input x is done in the following way. The general idea is the same as for the global case: take m samples, make two predictions, subtract them and compute the average of the differences. How the input for the predictions is created makes the difference here. First, a random permutation O of the features is created to then check which features precede i in O. According to this (see Algorithm 1 of Figure 2.10), the input for the two predictions is formed by combining the values of x with the value of the current random sample. The complexity of computing local feature importance is $O(m \times T(f(x))$ with $T(x)$ being the time it takes the model to compute one prediction. Local feature importance is a measurement that fits our task very well. We could use it to show users which parameters they should change because they have a negative impact on the prediction.

WeightLifter is an approach by Pajer et al. [59] to help users in complicated multi-criteria decision-making processes. It differs from the previously mentioned approaches and this thesis in the sense that it does not incorporate standard machine learning models. Instead, it uses weighted criteria to filter the result set and presents the user with the best solutions. The authors state that the main goal of their approach is to show how a change in criteria weight affects the result set, which is a similar task as described by Wexler et al. [76], Zhao et al. [80] and Krause et al. [40]. To achieve this goal, they colour code the region in which a change in weight does not influence the result a technique also used by Zhao et al. and their **split points** [80]. Additionally, Pajer et al. allow the user to **constrain the weight space**, which helps filter results even further. The major drawback of their visualisation is that it is limited to three trade-off parameters. They state that it is possible to combine several parameters into one trade-off. This is not applicable in our case, as we want to investigate the importance and influence of single parameters.

Goldstein et al. [27] criticise the usage of PD plots. They argue that by taking the average of all the sample predictions, important information can get lost if the dependence of the feature x to the other features is strong. In their work, they propose **individual conditional expectation** (ICE) plots. Instead of plotting the average of the predictions, each prediction curve is plotted separately. An example of such behaviour is shown in the Figures 2.11a, 2.11b and 2.11c. An ICE plot can also be viewed as a collection of all local PD plots. Goldstein et al. also introduce multiple variations of ICE. The centred ICE plot (see Figure 2.11d), for example, picks a curve x^* and centres the plot around it by showing only the difference to x^* for all other curves. According to Goldstein et al., this helps in the detection of outlier curves.

Case-based reasoning is another model-agnostic approach to make models more understandable. The general idea is to show the user input combinations that are similar to the current one. This way, the user can compare predictions between these inputs and derive information about the model behaviour. Caruana et al. [21] apply this concept to NN. They argue that, instead of computing the similarities in the input space, it is better to investigate what the NN thinks are similar inputs. During the training phase of an NN, an activation pattern for each training input is created. Caruana et al. save
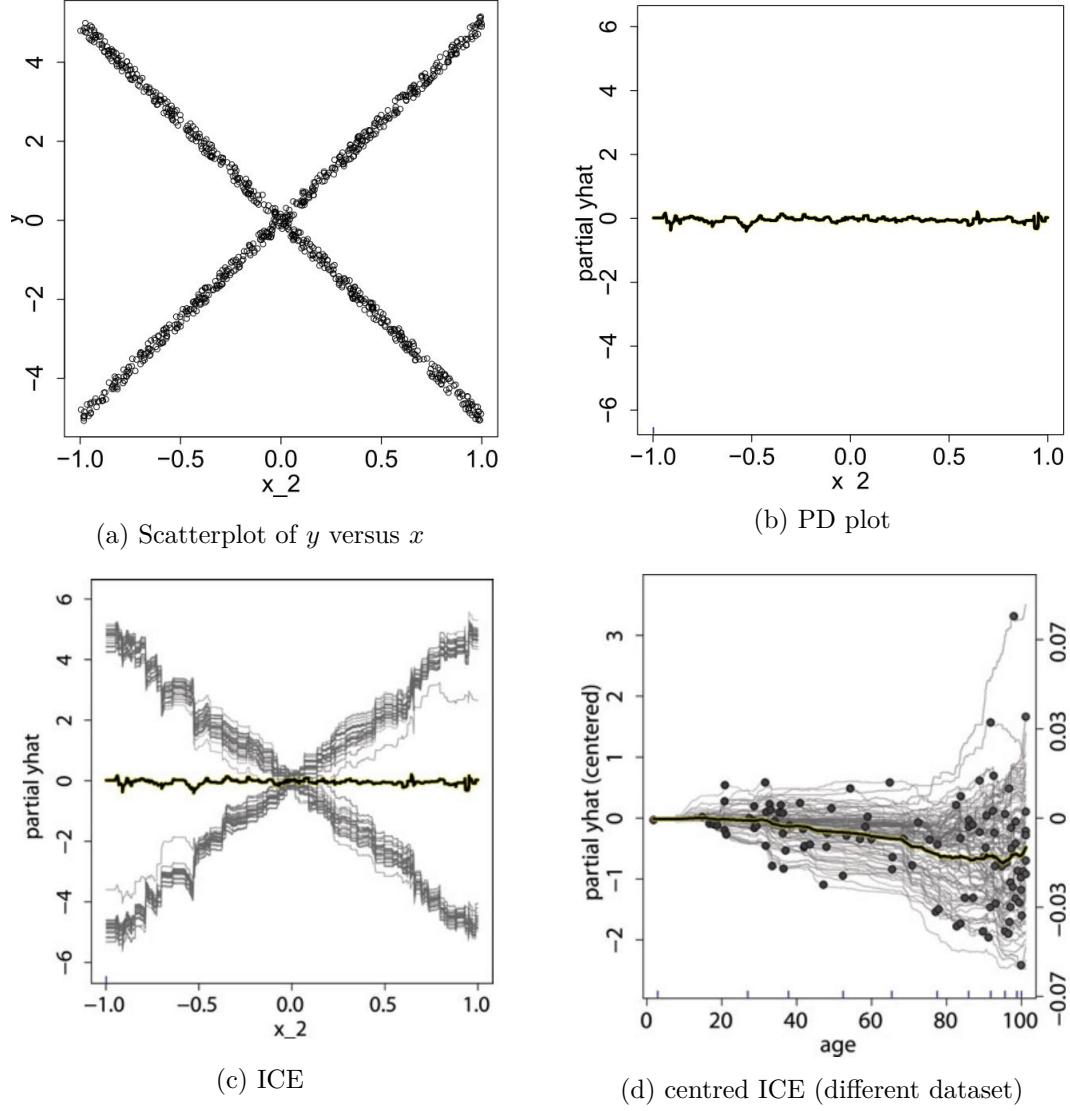
(a) Scatterplot of y versus x

(b) PD plot

(c) ICE

(d) centred ICE (different dataset)

Figure 2.11: The PD plot (b) suggests, that there is no meaningful relationship between feature x_2 and the predicted value y. When looking at the scatterplot of the predictions (a), it is obvious that this is not a correct statement. By using ICE plots (c), the data loss by approximation is omitted. To better find an outlier, a centred ICE plot (d) can be used. From Goldstein et al. [27]

these patterns. When a new input is fed to the NN, its activation pattern is also saved. Afterwards, the k-nearest neighbours (k-NNs) [61] of the training patterns to the current pattern are fetched. The k-NNs represent the most similar inputs as regarded by the NN. Caruana et al. argue that this approach explains the predictions better than computing similarities in the input space because when using the input space, only the training dataset and not the NN's behaviour is explained.

Predictive Machine Learning

The proposed EVEOS is developed as a front-end on top of a predictive machine learning model. The used model is a deep neural network (DNN) that was developed by our colleague Zahra Babaiee [1]. In this chapter, the machine learning process is discussed roughly, so that the methodology of the underlying data analysis can be understood. We first discuss the dataset used for machine learning and perform some data exploration tasks to get a better feeling of the data and remove potential outliers. Next, we take a look at the model training pipeline before discussing the API we develop for communication with the front-end. The programs used in the machine learning process are implemented in Python using the Keras [6] framework for developing neural networks.

3.1 Data

To feed the models, we use historical sales data provided by AbsolutTicket [1]. The dataset contains 1.5 million bookings of 300000 events. To avoid working with data from the production database, it is copied into a separate database that contains only the needed data. We use a service that retrieves new data from the production database every day to keep our dataset up to date.

When working with data, it is important to know which type of data the features belong to. First, we need to understand what types of data there are in general and what their differences are. According to Munzner [52], data can usually be assigned to one of two main types: categorical or ordered. **Categorical data** is grouped into categories according to its characteristics. It is important to state that these categories are mutually exclusive, so each element can only belong to a single category. For example, if we had a dataset of colour names "*blue, blue, red, red-blue*", it would feature the three categories "*blue(2), red(1), red-blue(1)*". Categorical data can also include numerical values if they

---

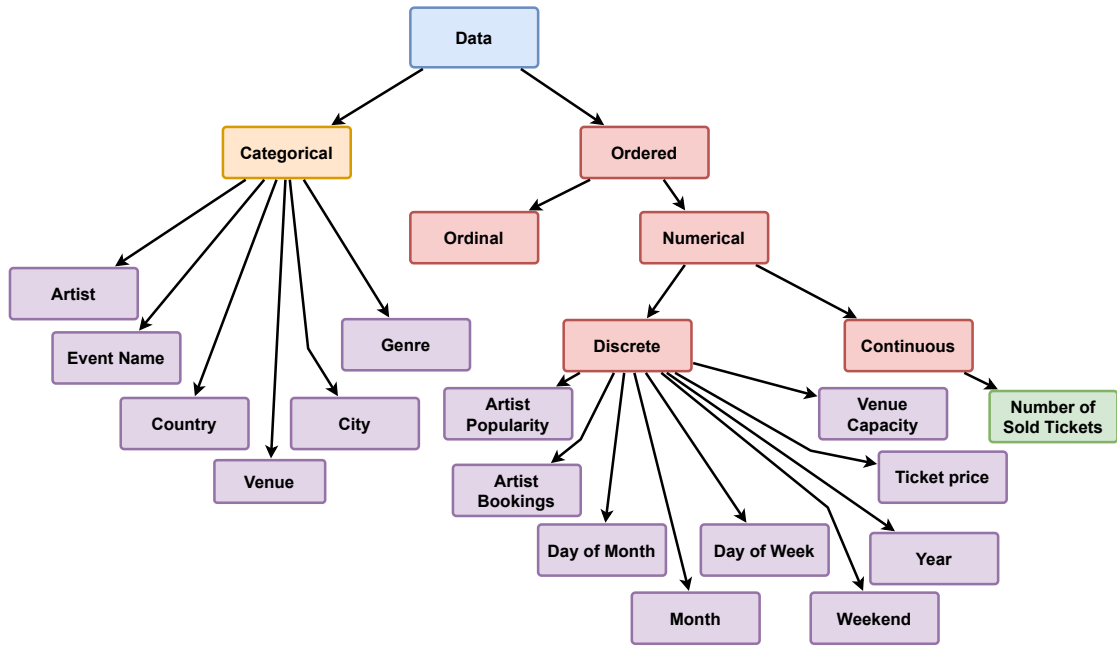[1]Zahra Babaiee, TU Wien, https://informatics.tuwien.ac.at/people/zahra-babaiee

Figure 3.1: Categorisation of the used data types. Purple data is used as the ML model input. The *Number of Sold Tickets* (green) is the ground truth for model predictions. The ground truth consists of discrete values but we categorise it as continuous because the prediction of the model is continuous.

are not measured and sortable but simply used as a category label. The values assigned to a category do not provide any additional information.

Ordered data on the other hand is always sortable. It can be further divided into two types: ordinal and numerical. **Ordinal data** is similar to categorical data but introduces a ranking to the categories. School grades for example can feature six categories from A to F. If we know that grade A is better than grade F, we can rank the categories accordingly. It is important to mention that even though ordinal data can contain numerical values, they are still ordinal since they cannot be mathematically measured. **Numerical data**, on the other hand, can always be measured. As the name suggests, it can only represent numbers and one can carry out mathematical operations on it. Numerical data can be divided into two groups: discrete and continuous. **Discrete data** consists of countable elements meaning a mapping with the natural numbers exists. **Continuous data** is uncountable and in general, represents a set of real numbers. Both discrete and continuous data can be either finite or infinite.

The dataset we are using contains categorical as well as numerical features. In Figure 3.1 we present a categorisation of the available features. The features that are used as inputs for the ML model are either categorical or discrete. Categorical features include: artist name, event name, genre, city, etc. Discrete features contain: artist popularity, venue

(a) Artist Popularity
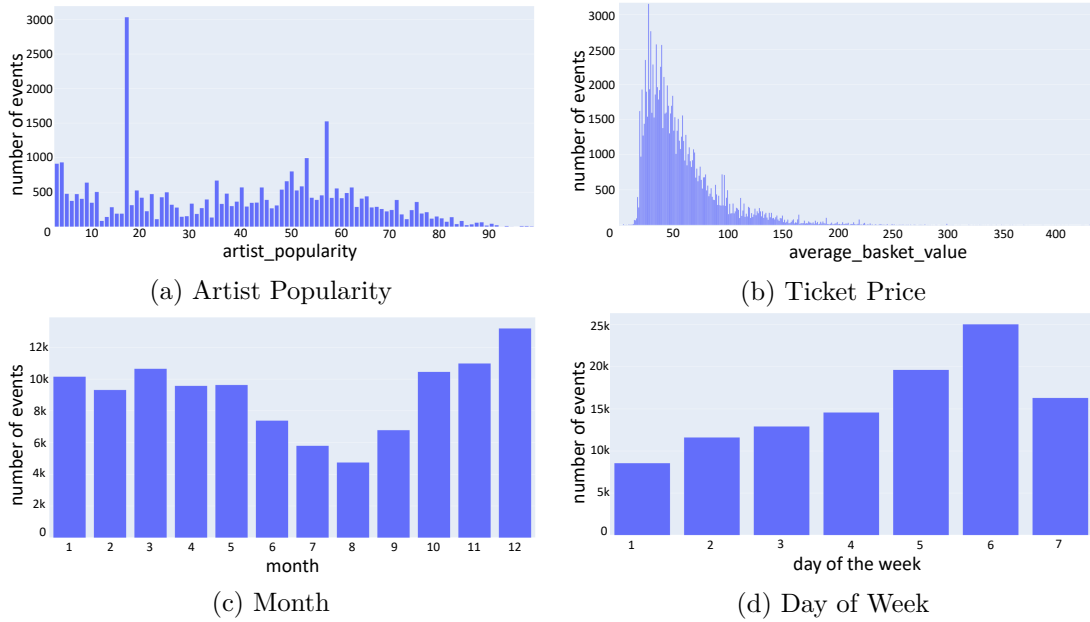
(b) Ticket Price

(c) Month

(d) Day of Week

Figure 3.2: Selection of training data histograms grouped by events.

capacity, ticket price, etc. This differentiation is important to make because it affects how the data needs to be encoded for the model. The number of sold tickets is the value we want to predict with the model. Since one can not sell half a ticket, for example, this is a discrete value. The employed ML model, on the other hand, returns continuous values when predicting the number of sold tickets, which is why we categorise it as such. A convenient circumstance is that the dataset is complete and does not feature any *Null* or *NaN* values.

3.2 Data Exploration

Before using a dataset for machine learning, it is always important to know what data one is using. In this section, we discuss what data exploration tasks we perform to become familiar with the data at hand. The aim is to find criteria for filtering outliers that would reduce the quality of the ML model. The graphics presented in this section are computed using the Plotly [8] package for Python.

As a first step, we compute histograms and scatter plot matrices of all features. This helps us understand how the data is distributed and if any irregularities cannot be explained with the domain knowledge we gained until now. A row of data in our dataset represents an event with all its parameters, which is why the histograms are computed by counting the number of events that fulfil a given characteristic. As shown in Figure 3.2, investigating the histograms gives us some interesting information. Grouping the events by month, we see that more events happen during the winter season than in summer. We

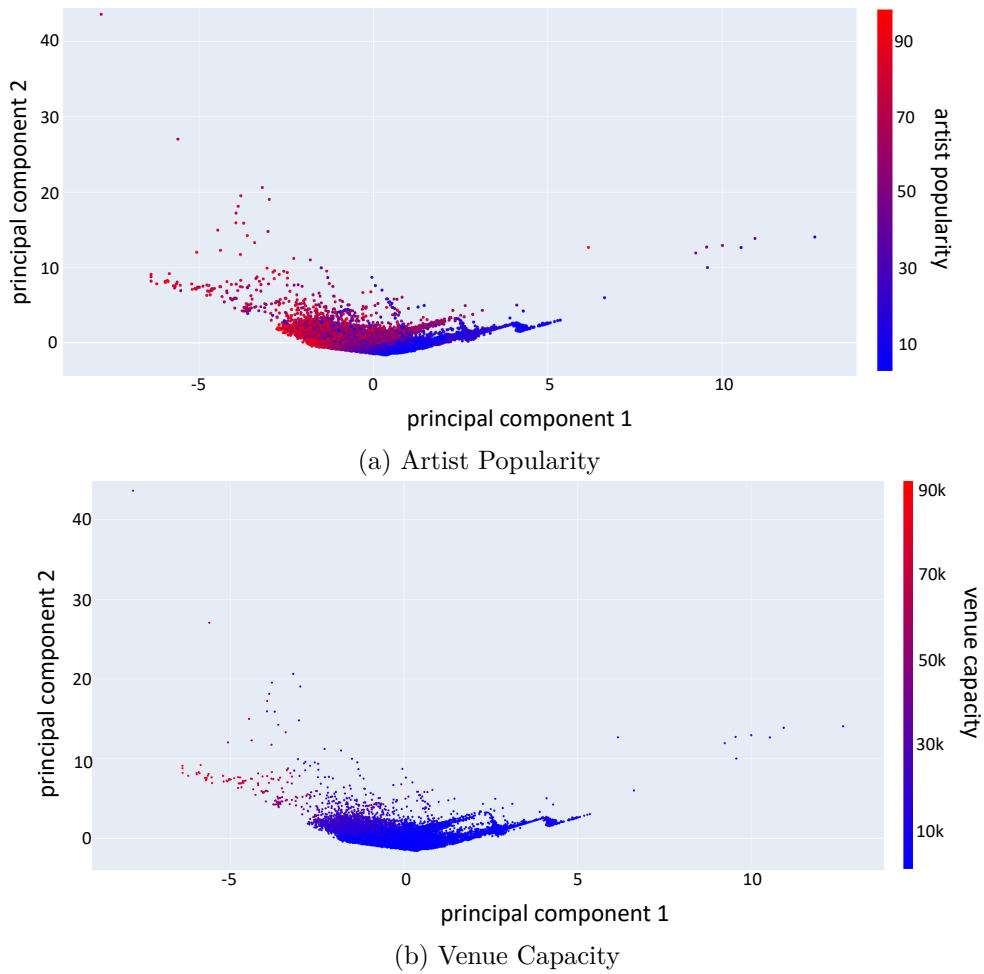(a) Artist Popularity

(b) Venue Capacity

Figure 3.3: PCA of numerical features *a)* colour coding according to the artist popularity and *b)* according to the venue capacity.

also find that, when looking at separate days, most events are scheduled on Saturdays. Grouping the events by weekend and working days, there are fewer events on weekends than on all working days combined. Other findings include that the ticket price is usually between €15 and €150. Additionally, the artist popularity seems to be evenly distributed with two outliers and only a few artists with a popularity of over 90, with 0 being the minimum and 100 the maximum popularity.

To further explore our dataset, we use the data dimensionality reduction methods employing the Scikit-learn framework [60]: **PCA** (principal component analysis), **t-SNE** (t-distributed stochastic neighbour embedding) and **MDS** (multidimensional scaling). The goal of all these methods is to reduce the dimensionality of a dataset to achieve a simpler representation of the data, such as creating a 2D version of a 10D dataset. This can help to visualise the data for data analysis purposes. Sometimes in machine learning,
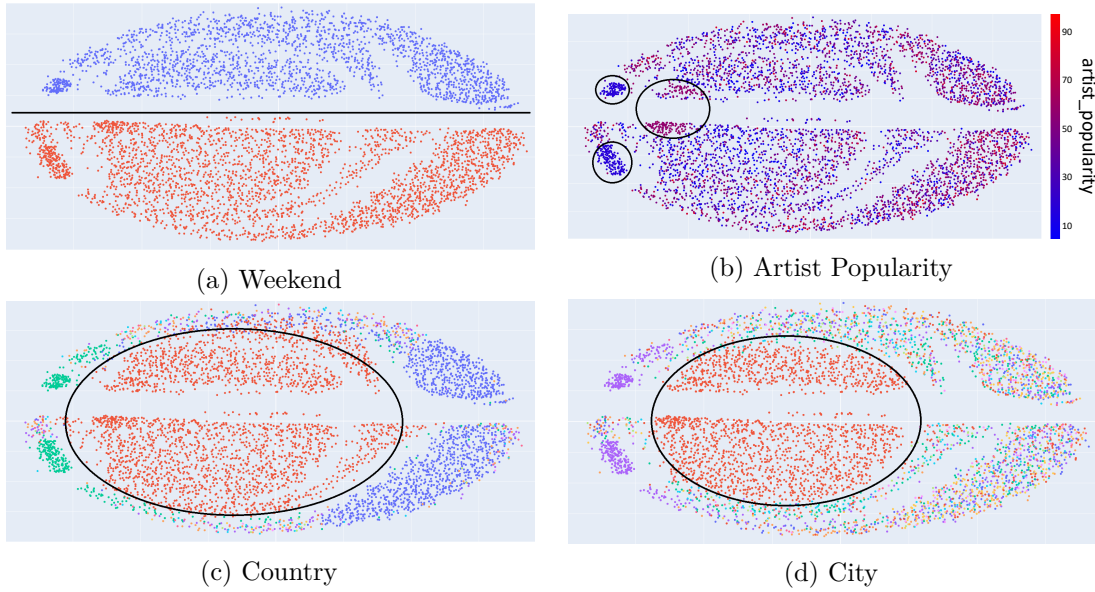
(a) Weekend

(b) Artist Popularity

(c) Country

(d) City

Figure 3.4: MDS using Gower's distance with different colour codings. *a), c), d)* encode categorical features. *b)* encodes a numerical feature. The line in a) and the ellipses in b), c), d) show clusters of interest.

better results can be achieved when using a lower dimensionality representation of the data. Showing these visualisations to domain experts helped us define rules for filtering data in the training pipeline described in Section 3.3.

To perform PCA [14], the principal components need to be derived first. These are defined as the eigenvectors of the data covariance matrix. The first principal component represents the direction in which the dataset shows the highest variance, the second one the direction with the second-highest variance while being orthogonal to the first component, etc. The principal components can then be used to perform a change of basis on the data. One can choose how many components are used for this.
In our case, we decided to transform our data into 2D, using the first and second principal components. It is noteworthy that PCA only works with numerical values, which is why we only transformed the numerical features of the dataset (see Figure 3.1). As shown in Figure 3.3, the PCA reveals a few outlier points but does not provide us with a lot of new findings otherwise. One interesting insight we gained when colour coding the plot according to different features is that in the provided dataset it seems that large venues with a capacity of more than 40.000 are almost always booked by popular artists.

Both t-SNE [73] and MDS [18] aim to reduce dimensionality while preserving the distances between data points, meaning that if points are close in x-dimensional space, they should also be close in 2D space for example. To achieve this, we need to compute a distance matrix that includes the distances between the points in x-dimensional space before we can deploy these dimensionality reduction methods. Since we also want to be able to
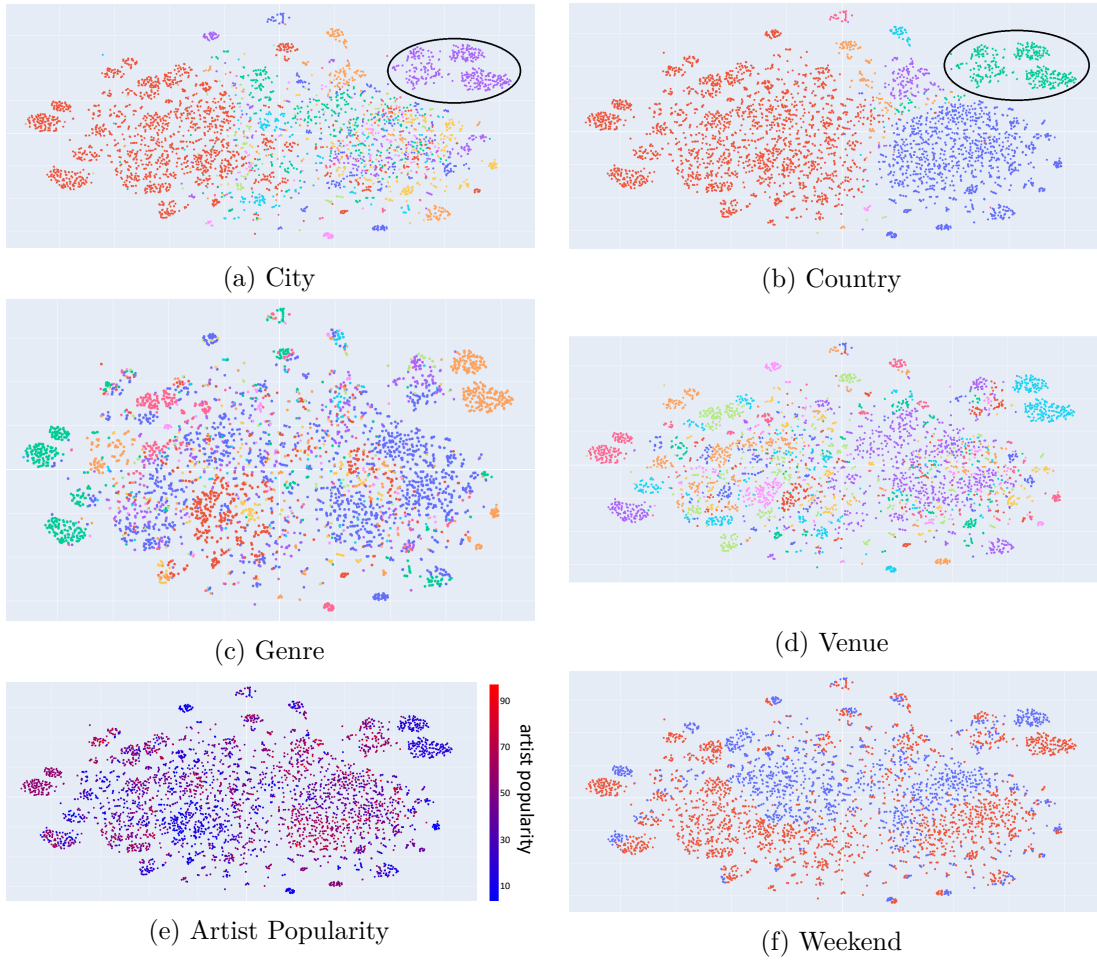
(a) City

(b) Country

(c) Genre

(d) Venue

(e) Artist Popularity

(f) Weekend

Figure 3.5: t-SNE using Gower's distance with different colour codings. *a), b), c), d), f)* encode categorical features, *e)* encodes a numerical feature. The ellipses in a), b) show clusters of interest.

explore categorical data, we decided to employ the Gower distance [29, 22], which works for categorical and numerical data, as the metric for the matrix. Since the distance matrix would be too large for the RAM of our machine and t-SNE and MDS are both computationally very expensive, we use a random sample of 5000 data points for the analysis. Our goal in using these methods was to find clusters in our dataset and gain insight into which features seem to influence the creation of these clusters.

Starting with MDS (see Figure 3.4), we see that the dataset is split horizontally into two clusters. Additionally, we see three elliptical-shaped clusters. Colouring the plots according to the features, we found that the horizontally divided clusters stem from events being either on the weekend or not. The elliptical clusters are related to the city and country of the event. The two inner ellipses all belong to the same country, with
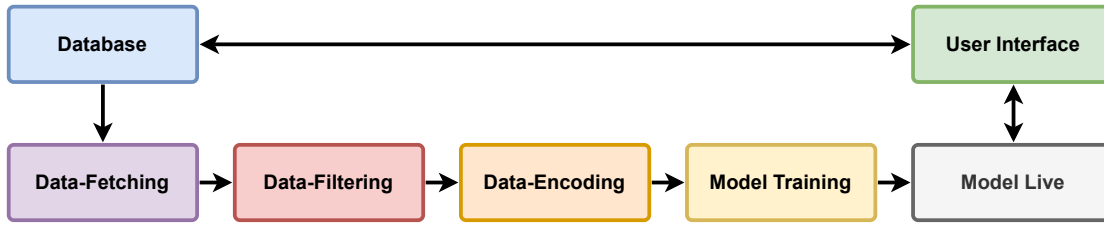
Figure 3.6: Stages of the model training pipeline.

the innermost one representing a single city in this country. The outer ellipse is divided roughly into two other countries, one right and one left, with the left one also containing a city cluster. This city cluster can also be found when looking at the artist popularity. There we find an additional cluster that relates to a single venue.

Looking at the result of the t-SNE in Figure 3.5, we can see that there is what seems to be a randomly distributed and rather sparse cluster in the middle of the plot and some distinct clusters around that. The distinct clusters relate to events that happen in the same venue, in the same genre and the same artist popularity. This suggests that the dataset contains a few event clusters that could be some kind of event series where the same artist performs multiple times in the same venue. The three country clusters we see in the MDS are also present here, but not as separated. The cluster of the green country and the purple city on the top right of the plots shows us that this combination belongs to a distinct group of events. In contrast to the MDS visualisation, not the whole dataset but only distinct clusters are separated by weekend and weekday.

3.3 Model Training Pipeline

In this section, we discuss the steps of the model training pipeline shown in Figure 3.6. As described in Section 3.1, the data provided by Absolut Ticket [1] is stored in a database. The first step of the pipeline is to **fetch** the data from the database. The training set has the same cardinality as the number of events in the dataset. To avoid overfitting, we simply do not use all events but fetch the training data in numerous batches, each batch containing 128 random events. Next, the data is **filtered** according to our findings described in Section 3.2 and domain knowledge provided by colleagues from Absolut Ticket. For example, we remove events where the ticket price is over €1200 or events in rural cities where almost no tickets were sold online. These are outliers as found in the data exploration and confirmed by domain experts. The filtered dataset is then saved for later processing.

Before the model can be trained, the data needs to be **encoded** accordingly. Depending on the data type of a feature, it is encoded using commonly known encoding techniques like min-max encoding or binary encoding. The metadata of the encoding is saved so we can use it later to correctly encode the input provided by the user interface. As mentioned before, the ML model we are using was developed by our colleague Zahra

Request Data   **Encoded Data**

```
{
 numberOfPredictions: 3,
 data:{
  artistPopularity:[80,55,95],
  month:[11,11,0],
  date:[30,31,1]
 }
}
```

```
[
 [0.8,0.91,0.96],
 [0.55,0.91,1],
 [0.95,0,0.03]
]
```

Figure 3.7: Data-transformation of the encoding process using dummy data.

Babiee using the Keras [6] framework. It is a DNN consisting of more than 20 layers. The model is **trained** to predict the number of sold tickets for an input representing a new event. The last layer of the network provides us with nine strictly monotonically increasing values for the predicted number of sold tickets. According to the probability density function used in the model setup, the probabilities for each of these values can then be computed. Finally, the trained model is **saved** locally so it can be loaded and used in the model API described in Section 3.4.

3.4   Model API

After the ML model is trained as described in Section 3.3, it needs to be hosted on a server to make it accessible for a web front-end. In this section, we discuss the structure of the API that serves this purpose. The API is developed using the Flask [5] framework for Python.

When starting the API, the trained model and the encoding metadata are loaded. The API provides a single endpoint that accepts a POST request containing the data needed to make one or more predictions. The data contains the number of predictions that need to be made and the raw input data entered by the user in the front-end. Additionally, the data is expected to be structured in the right way. As shown in Figure 3.7, each feature has to be represented by an array with index i of the arrays being the combined input for prediction i. By structuring the data this way, we can encode the values of each feature all at once. This minimizes the time needed for encoding because we can exploit the parallelisation capabilities of Numpy [7]. As described in Section 3.3, each feature is encoded according to its data type using the metadata saved during the training process. Finally, the encoded matrix is transposed so each row represents a complete event for which we want to predict the number of sold tickets (see Figure 3.7).

Once the raw data is encoded, the resulting matrix can be passed to the DNN at once, which provides a big boost in performance in comparison to performing each prediction separately. As described in Section 3.3, the model returns nine values per prediction. Using the probability density function that was also used when training the model, we can now map these values to their probabilities. To do so, we compute the intervals
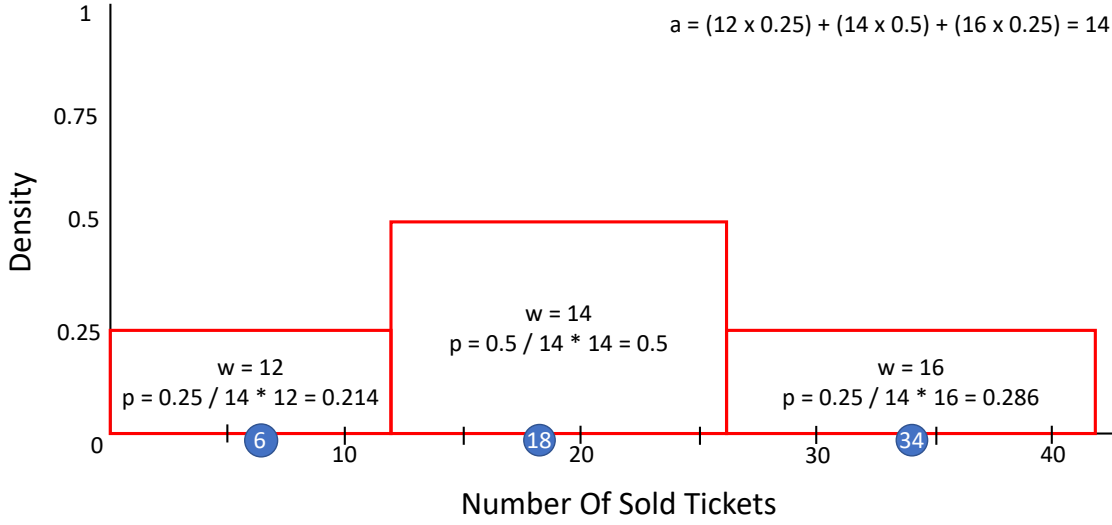
Figure 3.8: Computation of value probabilities.

between the values including their start, mid and endpoints. Given this information, we can derive the width w_i of each interval i. The height h_i of each interval is given by the density function. Next, we compute the area a of all intervals combined, summing up all nine individual areas: $a = \sum_{i=0}^{9}(w_i \times h_i)$. To then get the probability p_i of each interval we use the equation: $p_i = \frac{h_i}{a} \times w_i$. The resulting probabilities are returned to the caller of the API request together with the predicted values.

An example of this computation is illustrated in Figure 3.8. Assuming that the model returns the three values $[6, 18, 34]$ and a discrete probability density function with the values $[0.25, 0.5, 0.25]$, first the interval widths are computed: $[12, 14, 16]$. Following the proposed formula, a is computed to be 14. Using this information, we now can derive the probabilities for each of the values: $[0.214, 0.5, 0.286]$

Formatting the result so that it can be used in the visualisations of the front-end is not done on the server because it can be done much more efficiently in JavaScript than in Python. We describe the formatting in more detail in Section 6.2.

To optimize the performance of the server, we exploited the parallelisation capabilities of Numpy [7] wherever possible to avoid the use of standard Python loops, which are rather slow. There are five steps from sending a request to the API and having the complete result that could potentially be performance bottlenecks: encoding the data, making the prediction, computing the probabilities, decoding in the front-end and sending the data back and forth. As shown in Table 3.1, the total time t_{total} of an API call only rises slowly with the number of predictions being requested. As expected, the time needed for the single steps also increases accordingly, with the time needed for communication being stable until the amount of data sent reaches significant levels.

#predictions	$t_{encoding}$	$t_{prediction}$	$t_{probabilities}$	$t_{decoding}$	$t_{communication}$	t_{total}
95	24	53	1	2	334	414
327	51	74	2	12	361	500
790	68	87	1	11	380	547
1477	133	138	1	37	373	682
3809	275	583	1	75	677	1611

Table 3.1: Performance measurements of the ML API in milliseconds.

As we discuss in Section 5.4.3, we make 50 predictions for each numerical value, which is only the ticket price as shown in Figure 4.1. The other feature that has a strong impact on the number of predictions is the event date. We need to make a prediction for each date in the range set by the user. Talking to domain experts, they stated that they would usually not search for a time span larger than 3-6 months. Including the categorical samples, users request between 50 and 300 predictions at once under normal use. For this amount and more, t_{total} is below 1 second most of the time, which, as discussed in Section 2.2, is long enough for users to realise that they are waiting for something, but is fast enough for them to not feel frustrated by this, even without the use of any kind of loading symbol.

Paper Prototype

When creating the proposed exploratory visual event-organisation system (EVEOS), our main focus was on developing a system that is usable by our target group of event organisers. We opted to deploy a user-centred design process as described in Section 2.1. Involving the users regularly and early helps us to better understand their needs and also provides us with domain knowledge that would otherwise not have been acquired. As described in Section 1.1, we aim to develop an EVEOS that helps organisers optimise their choices regarding event parameters. The EVEOS should provide them with possibilities to compare different parameter settings and make the underlying ML model understandable in order to build trust in the system.

Knowing about the users, their tasks and requirements, the next step in the user-centred design process is the development of a first conceptual design. In this chapter, we discuss the creation of a paper prototype that evolves from simple sketches to a complete view of the system with multiple alternatives for layout and visualisations. First, the data that needs to be shown is discussed. We then consider potential overall layouts of the interface before going into detail on the separate components. For each component, we discuss its purpose and multiple layouts and visualisations that could be fitting. In the end, we evaluate the prototype with two domain experts, which allows us to filter inappropriate ideas of ours and to come one step closer to a final product that fulfils the needs of our users.

4.1 Data Types

In Section 3.1 we discussed the dataset the employed DNN is using. To generate valid inputs from the user interface, users need to be able to select all features present in this dataset. As illustrated in Figure 4.1, some input features are grouped together. The reason for this is that they are directly dependent on each other. The event date, for example, is used by the model as distinct features for the year, month, day of month
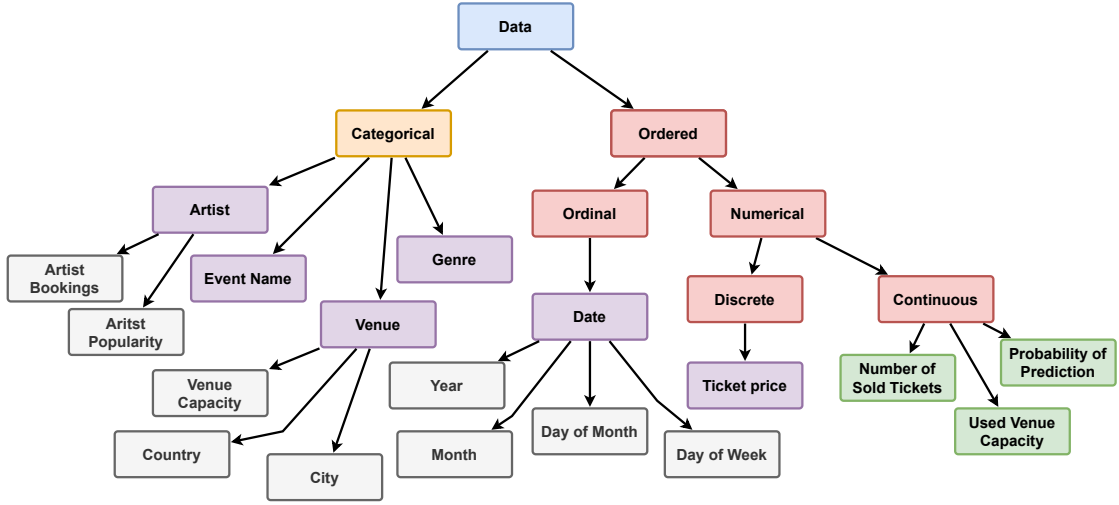
Figure 4.1: Data Categorisation by input parameters. Purple data needs to be selected by the user as input parameters for the model. Green data results from the model prediction. White data is directly dependent on its parent parameter and is not categorised.

and day of the week. For the users, it does not make sense to select all of these values separately. They simply select a date and the features are directly defined by this date. This is echoed by the venue feature with the city and country of the venue being distinct features but still defined by the venue. This circumstance poses a unique challenge for dealing with the data.

The nature of the different data types also suggests that the visual encoding of their input fields can and sometimes has to differ. Our goal is to show the inputs together with local partial dependence (LPD) information to allow comparisons between input values. For **numerical** inputs Krause et al. [40] solve this problem by introducing LPD bars as shown in Figure 2.8. The bar represents successive samples of the numerical input with its colour showing the prediction value. Our data only contains a single numerical feature, the ticket price, for which we can adopt this solution.

Ordinal inputs like dates could also be encoded using this method. One could either show the discrete numerical values of a date as separate LPD bars or since a date can be viewed as an ordinal feature, use the whole date as values on the bar. As we argue more closely in Section 4.4, dates already have a standard input format, date-pickers in calendar form. Users are accustomed to date-pickers and they can be enhanced with LPD information. In general, we need to decide if for the grouped features we always show the LPD information of the whole group or its sub-features separately.

For strictly **categorical** features it is more difficult to show their inputs and LPD information in a compact way. The first problem is that the number of possible values can be very high, depending on the feature. In our case, the selection of venues has thousands of options and the possibilities to choose an event name are infinite. Numerical

data has a vast amount of options as well, but for numerical data, it is easy to take meaningful samples, which is not the case for categorical data where the only way of selecting samples is that the user selects them. Additionally, categorical data inputs cannot be encoded on an LPD bar since one needs to have some kind of drop-down selection or free text input to select samples.

As discussed in Section 3.3, the DNN gives us a prediction on values for the **number of sold tickets** (NT) and a **prediction probability** (PP) for each of those values. In preliminary talks with domain experts, they mentioned that often they do not work with the absolute number of tickets, but rather with a percentage of **venue occupancy rate** (OR). The occupancy rate can be computed by dividing the predicted number of sold tickets by the venue capacity of the chosen venue, which makes it dependent on these two values: $OR = \frac{NT}{venueCapacity}$. This leaves us with three **continuous** values that should be shown to the user for each prediction.

4.2 General Layout

The first step in designing the proposed EVEOS, is to define which components are needed so that the users can fulfil the following five tasks:

1. Create inputs for the ML model,

2. filter input values,

3. read and interpret the prediction,

4. compare predictions of different input combinations and

5. compare predictions to ground truth data.

Since all of the tasks are connected, we want to show all components on a single page. On the other hand, there is a lot of information to show, which poses the risk of information overload if the components are not structured well. It is also important to consider the available screen space. In this work, we are focussing on finding a suitable layout for standard PC screens with a resolution of 1920×1080 pixels. In the end, we are settling on the following component structure.

As shown in Figure 4.2, the interface consists of five components. On the left side, we have the **input** component, which allows users to manipulate the input data for the model. Each input feature has its small area and is connected to a measurement of **feature influence**, which should indicate the influence the selected value of the feature has on the current prediction. Additionally, each feature value is connected to LPD information. We discuss the chosen encodings in greater detail in Section 4.4. This component contains functionality to perform the tasks **1** and **4**.
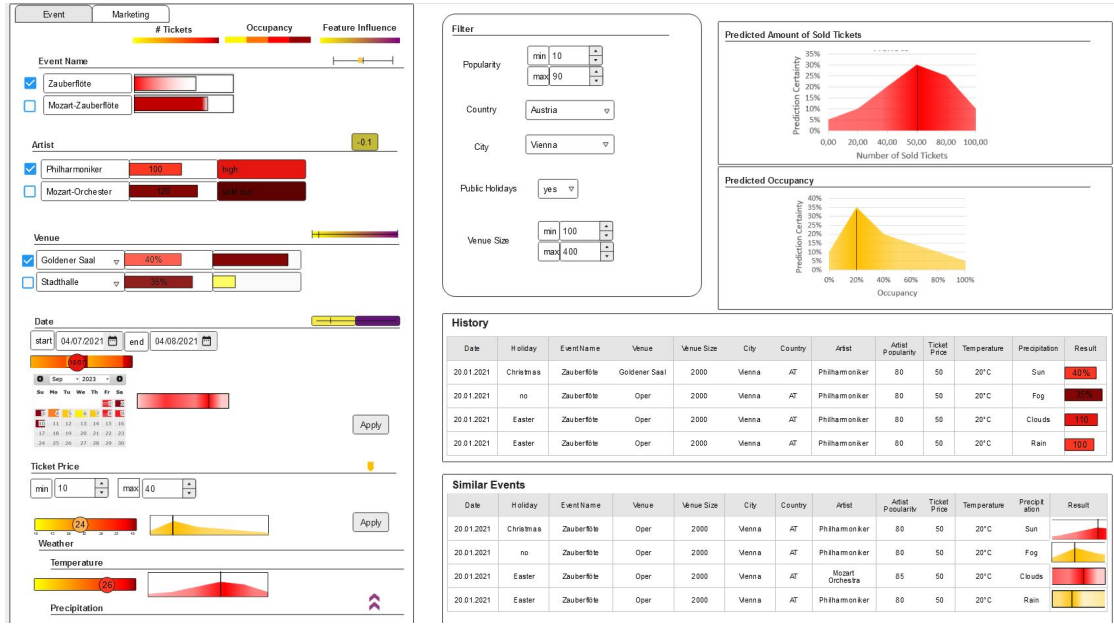
Figure 4.2: First general layout.

In the top-middle part of the interface, we place the **filter** component to fulfil task **2**. As discussed in Section 4.1, for categorical data, users have to select the samples they want to compare themselves. For some features, the domain experts suggested that they would like to have the possibility to filter them, for example, filtering venues by location or capacity.

The **result** of the prediction of the current input is shown in the top-right corner. Aiding task **3**, this component should present the result in a comprehensible way and give detailed information about it. The idea is that this is a static component for reading only. The view only gets updated when a new prediction is made and interaction is limited to showing details on demand. Different solutions for encoding the result are discussed in Section 4.3.

In the bottom-right corner of the system, we find two tables. The one on top is the **input history** table. The idea behind this table is that every time the user changes an input value, a new row is added to the table representing the current input values. This way, a change history of the input selection is created. Each row of the table also features an abstracted version of the prediction result of its values. To have a consistent design, this abstracted result should have the same encoding as the one used in the input view for partial dependence information. Different encodings are discussed in Section 4.4. Together with the information presented in the input view, this component should help the users to compare different input combinations. In the input view, users can compare different values for a given feature, whereas in the history table they can easily see the difference between multiple complete input rows. Additionally, one can select a

configuration from the table and take over its parameters into the input view.

The second table component contains data that allows **case-based reasoning** (task **5**). It shows training data that is similar to the current input, which are in our case similar historical events. By inspecting the predictions of historical events and seeing how many tickets were actually sold, users can validate the prediction for their new event. In this prototype, the prediction and actual value of the ground truth data is all shown in the table using again an abstract prediction encoding. We think that it could be helpful to also superimpose the ground truth prediction with the current one, to allow more detailed comparison and discussed this idea with domain experts in Section 5.7. In preliminary talks with domain experts, we realised that for data-privacy reasons it is not allowed to show the ground truth of an event of organiser A to organiser B. Therefore, the data available for case-based reasoning always has to be restricted to historical events of the organiser currently using the system.

4.3 Result View

The goal of the result view is to present the user with the prediction result. As discussed in Section 4.1, there are three continuous features that comprise a prediction: NT, OR and PP. At this stage of the design process, the ML model is not developed yet so we do not know the format in which the result is sent from the model. As shown in the timeline in Figure 1.1, the model will only be included in the final prototype discussed in Chapter 6. What we know is that the goal of the model is not to predict a single value of how many tickets are sold, but multiple values leaving us with multiple 3D data points. An example output could be: $[\{NT : 100, OR : 0.5, PP : 0.45\}, \{NT : 150, OR : 0.75, PP : 0.55\}]$. We, therefore, thought about visual encodings to show a collection of 3D data points.

In Figure 4.3a.a a first sketch of a possible result view is shown. The idea here is to show the values of NT and PP in some kind of area chart by interpolating between the values. PP additionally is encoded by a coloured bar above the area chart. The occupancy rate would be shown separately as seen in Figure 4.3a.b. We show OR with the highest PP in a categorical pyramid with the categories: low, middle, high, sold out.

In the next iteration of sketches, we disregard the idea of having two separate graphs and try to combine all information in one graph. We achieve this by using an area graph, encoding NT and PP on the x- and y-axis. OR is given by the colour beneath the curve. In Figure 4.3b, we see two sketches, one using a sequential single-hue colour scale and one using a multi-hue colour scale.

The proposed graphs properly encode the three available features, but we want to highlight the data point with the highest probability PP even more. The idea is to add visual cues around this point. In Figure 4.3c we present four more graphs that showcase attempts to achieve this. All of the presented graphs feature a vertical line reaching from the point with the highest PP to the x-axis. We also try using the opacity of the area to add an encoding layer to the probability feature. In Figure 4.3c.a the area is coloured in a single
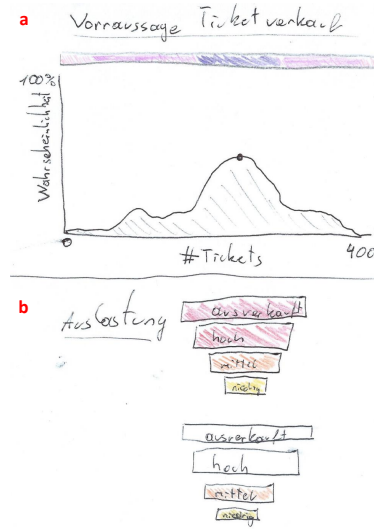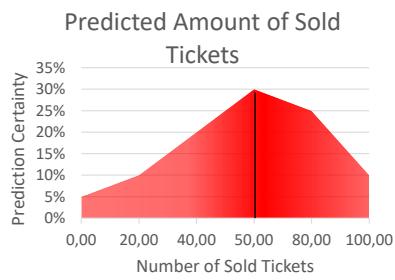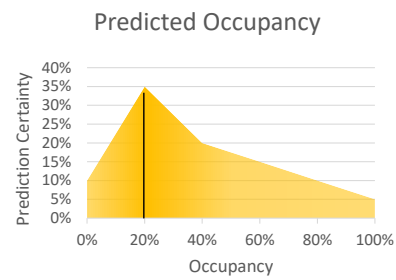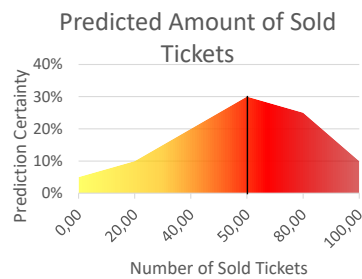
(a) Number of tickets and occupancy rate separated.
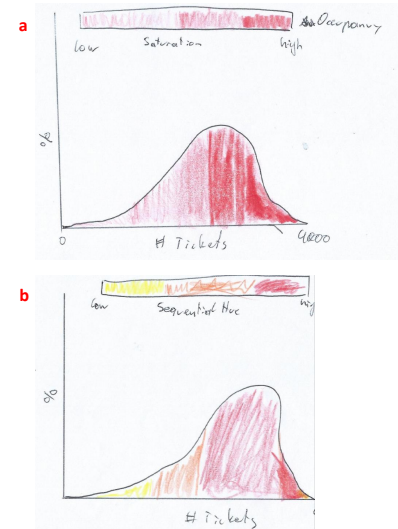
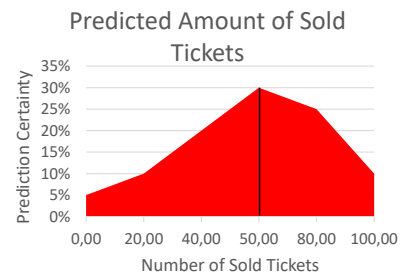(b) Number of tickets and occupancy in one graph, using colour to encode the occupancy.

(c) Screen prototype. Using colour to encode occupancy and opacity to encode probability.

Figure 4.3: First prototypes of prediction result views.

colour value representing NT and the opacity of the area is given by PP. Same goes for Figure 4.3c.b, which shows the occupancy rate. Figure 4.3c.d shows again the number of sold tickets, but without incorporating opacity. A combination of NT and OR results in the graph is shown in Figure 4.3c.c. OR is again encoded using the colour of the area and we also employ opacity.

Our hypothesis is that these additions help users to be faster in finding the point with the highest PP and in general guide their focus to important regions of the graph. We later have to realise that manipulating the opacity of the area when using sequential colour scales changes the perception of the colours in a way that destroys the sequential properties of the scales as discussed in Section 5.3.

4.4 Input View

The input view represents the point of interaction between the user and the interface. Here users can set the input parameters for their event and predictions are made accordingly. For each parameter, multiple options should be selectable and the corresponding predictions comparable using LPD information. Users should also see how the chosen value affects the prediction using the measurement of feature importance (FI). In the following sections, we describe the different encodings we develop for input selections, LPD and FI. A global view of how the input component is structured and the encodings are placed can be seen in Figure 4.2.

### 4.4.1 Data Input and Local Partial Dependence

Local partial dependence (LPD) of a feature can be described as showing the prediction results of multiple values of this feature, assuming that the other features do not change. It is a great tool to perform **what-if analysis** on ML inputs. In our case, the prediction contains multiple values as discussed in Section 3.4. Therefore, we need to decide if we want to show the complete prediction for LPD or just the value with the highest PP. We also need to find encodings that allow us to show LPD information while also providing input fields for all the different data types. Figure 4.4 shows first sketches of how this could be achieved, which we discuss in this section.

For **numerical features**, we can use the LPD bar proposed by Krause et al. [40]. As shown in Figure 4.4a.b and c, the LPD bar allows us to combine input selection and local partial dependence information into a single encoding. The different colours encode the prediction value with the highest PP with a sequential multi-hue color scheme (see Figure 4.4a.a). The range of the bar can be changed using input fields for minimum and maximum bounds. Since they can be ordered, it is possible to also encode **ordinal data** using the LPD bar. We are thinking about employing it for date features in particular. As shown in Figure 4.4a.b, we also think it could be helpful to add details on demand to the bar, by showing tooltips giving information about samples on the bar if there is any additional information to show.
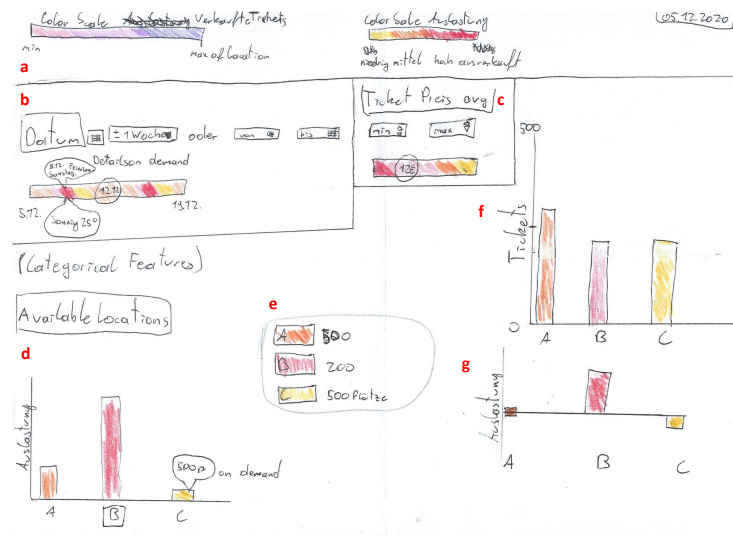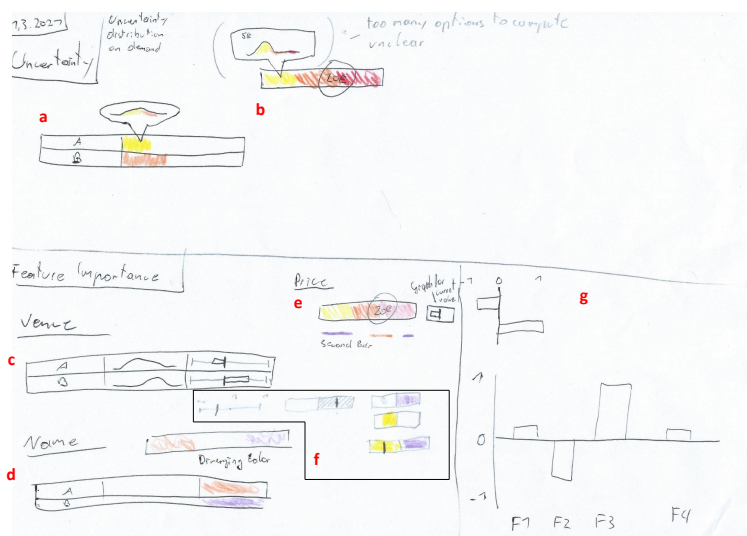
(a) LPD encodings for numerical and categorical data.
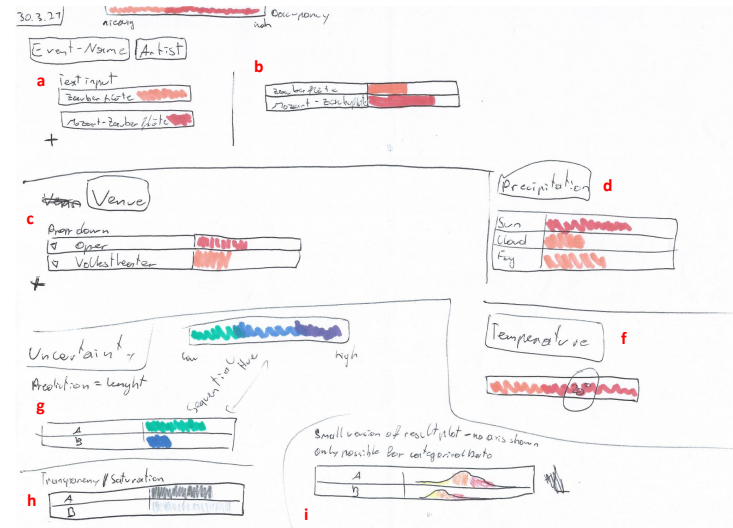
(b) LPD, probability (uncertainty) and input encodings for categorical data.

(c) Feature importance encodings.

Figure 4.4: Sketches of visual encodings for different data types.

Categorical features are more difficult to combine with LPD information since they do not have an order in which they could be placed on a LPD bar. Since they cannot be ordered, it would be necessary to add labels for each sample to the bar, which is difficult for more than a few labels because of the limited space. Additionally, it would be complicated to add or change input values. We, therefore, decide to think of alternative solutions for categorical features.

In Figure 4.4a.d, e, f and g we present different types of bar charts that show LPD information. Each bar encodes the prediction value with the highest PP. Using a coloured bar we can encode two values at the same time. Figure 4.4a.f, for example, encodes NT as height and OR as colour of the bars with Figure 4.4a.g being similar but centring the graph around a selected bar focussing on the difference to the other bars. We think adding details on demand as shown in Figure 4.4a.d is helpful for the user to understand the prediction. Thinking about these encodings and showing them to domain experts, we realise that bar charts like this are not suitable because we cannot have flexible input fields there. Therefore, we try to define the input fields first and then think about how we can add LPD information to them. The standard input encodings for categorical variables are free text inputs or dropdown selections. As shown in Figure 4.4b.a, we added a button (+) that provides the possibility to add additional input fields. This way users can add multiple values they want to explore to a given feature. Staying true to our first idea of using bar charts, in the Figures 4.4b.a, b, c and d we show different versions of bars beside each input value.

All discussed encodings show the prediction value with the highest PP. What they do not show is the probability of this prediction value and they also omit the other predicted values. One way to fix this is shown in Figure 4.4b.g where either NT or OR are encoded using the bar length and the colour showing the PP. Using this approach we would lose information about NT or OR. Other ideas we have include showing the PP with transparency as seen in Figure 4.4b.h. Adding this third visual encoding layer we can show all three values at once. If we want to show all prediction values and not just the one with the highest PP, we can show a small scale version of the area graph presented in Section 4.3 instead of bars. The downside of this is that this is not possible for the LPD bar. What we can do for all input types is to show the graph as a tooltip on demand as shown in Figure 4.4c.a and b.

Inspired by these sketches, we develop multiple encodings for the screen prototype as shown in Figure 4.5a, which we then show to the domain experts for evaluation. In all the alternative graphs, NT is encoded by bar length and OR by the colour. Figure 4.5a.a is the most basic encoding, simply showing a coloured bar. To make it easier to read the value of different bars in addition to comparing their relative length and colour, we add the value of NT as text in Figure 4.5a.b. Alternatively, in Figure 4.5a.c we disregard the length and show only a coloured area with OR defining the colour and NT being added as text. Since length is a better encoding for numerical values than colour as stated by Munzner et al. [52], we think the users will find this encoding to be the hardest to interpret.
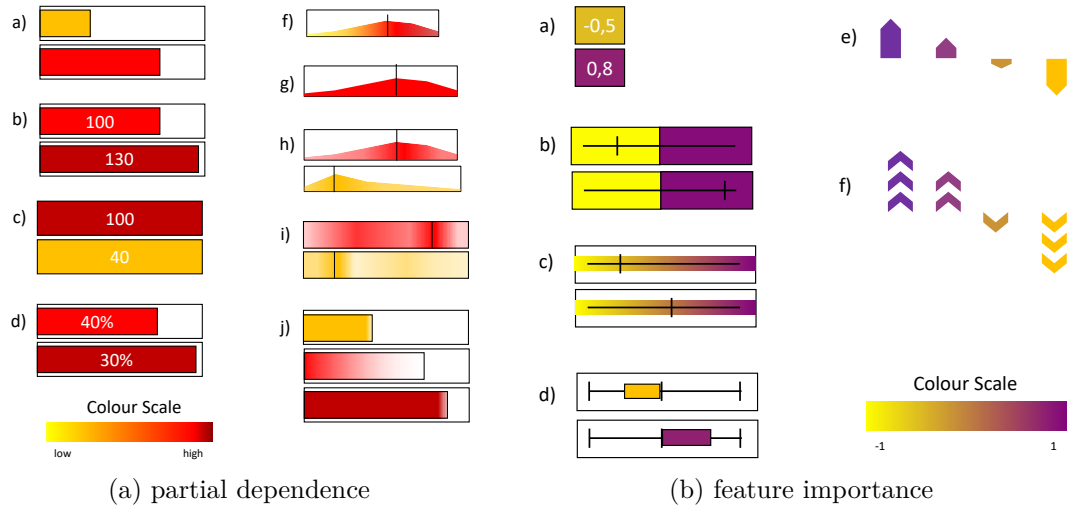
(a) partial dependence   (b) feature importance

Figure 4.5: Alternative visualisations for categorical partial dependence information and feature importance.

Trying to add information about PP to the bars, we show it as text in Figure 4.5a.d. We are not sure though if the users will understand what this text represents without an explanation. Since OR is also a percentage, it could be encoded likewise. As discussed before, we propose to show small versions of the area chart to convey LPD information for categorical values. In the Figure 4.5a.f, g and h we show multiple versions that we already discussed in Section 4.3. We think that this type of encoding will be the preferred choice by the users. It gives the most detailed information and in our opinion, it is still easily comparable since the vertical line gives the position of the prediction value with the highest PP. On the other hand, this encoding could also lead to some information overflow because there is a lot of information packed into a small area. We, therefore, try to abstract the area chart by removing the curve. In Figure 4.5a.i we show a chart that encodes the PP values using transparency instead. Reverting back to a bar chart in Figure 4.5a.j, PP is shown using by fading out the bar. The later the fade starts, the higher is PP. In our opinion, the last two encodings are probably the hardest to interpret without a previous description of their meaning. On the other hand, they provide additional information about PP that is not shown in the bar charts. We are excited to see what the domain experts think about all these alternatives.

As mentioned before, we can encode the date feature as an LPD bar. Showing the sketches of this to domain experts, they asked us why we do not use a calendar based date-picker. This comment is of course valid because date-pickers are the standard input component for dates and the users are used to them. As shown in Figure 4.6, we find that a calendar can be used in the same way a LPD bar can by just colouring the fields of the calendar using the prediction value.
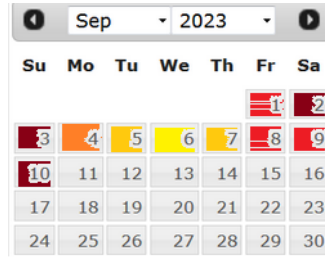
Figure 4.6: Date input using a calendar and colouring the fields according to the prediction value.

4.4.2 Feature Importance

Additionally to showing the users LPD information to make different feature values comparable, we also want to provide some guidance towards the quality of the currently selected value. This is not so much about giving detailed measurements, but rather about indicating if a value is good or bad. Inspired by the local feature importance presented by Strumbelj and Kononenko [71], we image our feature importance (FI) to be a value between -1 and 1 giving us information about the influence of a value on the prediction. How this value is computed is discussed in Section 6.4.1 since at this stage of the design process we have not decided on the approach we want to employ yet. The FI is a diverging value centred around 0, it seems natural to use a diverging colour scale in the following encodings.

We start by developing some sketches of FI encodings shown in Figure 4.4c. Adding the FI information to categorical data inputs, we use bars centred around 0 in the examples in Figure 4.4c.c and f. In Figure 4.4c.d we try using only a coloured area. Trying to add FI values to all samples of a LPD bar, in Figure 4.4c.e we aim to show a second bar below the LPD bar. In retrospect, this would just be a duplicate encoding of the bar itself. It also suggests only showing a single graph for the currently selected value instead of showing FI for all samples. Untying the graph from the input fields, in Figure 4.4c.g we show a separate bar chart comparing the feature importance of the different features using the FI of the currently selected values.

Preparing the screen prototype for user evaluation, we propose multiple FI encodings in Figure 4.5b. Figure 4.5b.a simply encodes the FI value as text on a coloured background. The graph in Figure 4.5b.b consists of a binary coloured bar indicating if the value is good (yellow) or bad (purple) with the exact value shown using a line. Enhancing this encoding, we use the complete colour scale instead of the binary encoding in Figure 4.5b.c. Trying to keep the graphs consistent with the LPD encodings presented in Section 4.4.1, we use bars extending either left or right from the center and using the colour of the bar continuously with the colour scale in Figure 4.5b.d. We think that option **d** is the easiest to understand since it gives detailed information in a way where it is easy to read the current value and allows good comparison.

We also propose the usage of **glyphs** to encode feature importance. In Figure 4.5b.e a closed arrow is used where the height and the colour give the value. If the FI is negative the arrow is pointing downwards, else upwards. Abstracting this encoding, we propose a separated arrow in Figure 4.5b.f. Here the colour encodes the value continuously, but the number of arrow parts is discrete. This is not as exact as the closed arrow, but we think it is visually more pleasing and makes it easier to compare different values since for this measurement we believe that it does not matter if the value is 0.4 or 0.44, but rather if it is good or bad. We think that using three arrow parts for positive and negative values respectively is a reasonable granularity for this task.

4.5 Evaluation

After developing the paper and screen prototype, we now need to evaluate them. We want to know if the choices we made are good and which of the alternative solutions we presented in the previous sections are the right ones. Therefore, we perform interviews with two domain experts. *User 1* is working in a managing position and therefore does not work on creating events in event-administration tools every day. *User 2* is dealing with the task we want to solve in day to day business. The interviews roughly follow the following structure.

First, we give a short introduction to the topic and talk about the main goal we pursue with this project: introducing machine learning into the event-organisation business. Next, we discuss the nature of the predictions that the proposed ML model makes. We explain to the domain experts, that predictions are not 100% accurate and we show multiple prediction values with corresponding probabilities. Following that we show them the visual encodings we developed for showing the prediction result as discussed in Section 4.3 and ask them to interpret them. We then show them a preview of the input view so they get the context before discussing the alternative encodings we propose for handling input parameters and displaying local partial dependence information LPD and feature importance FI as discussed in Section 4.4. In the end, we present them with the general layout and talk about the input history table and the case-based reasoning table.

Talking about the **result view**, both users were able to interpret the proposed area charts after being explained about the model returning multiple values with corresponding probabilities. Especially *user 2* was fast to explain this even before the introduction was finished. They both stated that the line highlighting the prediction value with the highest PP helps them to focus on this area. On the other hand, they suggested that it would be even more helpful to have a textual legend that gives them the prediction value and the probability of that point. Discussing the usage of colour, *user 1* stated that they prefer the version shown in Figure 4.3c.c where the colour scale fills the area. They suggested that this helps them in gaining a better perspective on the result, meaning that it is easier to grasp if the value with the highest PP is good or bad relative to what is possible. In contrast the graphs of Figure 4.3c.a and b do not provide them with this perspective. *User 2* found the colours more irritating than helpful, contrary to our

hypothesis. They suggested showing only a line, which indicates to us that they did not fully understand that the colours encode a different value than the line height. Using transparency to additionally encode the prediction probability was confusing to both of the users. After explaining this concept in more detail, *user 1* stated that now they think adding the transparency to Figure 4.3c.c would result in their preferred version. This aligns with our hypothesis because we think that this combination would present the most detailed data while highlighting the area around the value with the highest PP.

Discussing the encodings for **LPD information**, we asked the domain experts which graph allows them to compare multiple predictions the fastest and where they can read the most information. *User 1* repeatedly stated that they like the small area chart in Figure 4.5a the most, saying that it gives them the most information and they find it easy to compare because of the position of the line. As we thought when designing the graphs, *user 1* criticised that the bar charts do not show the uncertainty of the prediction in any way. The bar chart using transparency and fade-outs proposed in Figure 4.5a.i and j would solve this, but their semantics were not at all obvious to the users. *User 2* on the other and preferred the most simple of the encodings (see Figure 4.5a.a) using only colour and text to encode OR and NT. They stated that adding text would strongly suggest that the number of tickets is predicted with a probability of 100%. They suggested that showing a small version of the area chart is not necessary because, on this small scale, it would be hard to interpret. We asked *user 2* if they think it is enough information if we only show the simple bar chart and when selecting the corresponding value, the area chart is shown in the result view in full scale. They replied that they think that this is a good solution and even though it would be nice to see information about the probability already in the LPD information, it is enough to see it in the result view after selecting the value.

We then discussed the usage of LPD bars for numerical features. For both users, it was immediately apparent how to use them and that the colour on the bar encodes the prediction of the corresponding value. When talking about the different encodings for the date feature, both users stated that they highly prefer a calendar view for selecting and comparing dates over the LPD bar because they are used to working with this input component. This confirms our thoughts about this topic.

Next, we discussed the encodings for the feature importance (**FI**) value. Here the focus was if the users can explain approximately if the current value is good or if it is bad and therefore changes should be made to the input selection. *User 1* stated that, not including the glyphs, the encoding using only text and a coloured background (see Figure 4.5b.a) is the easiest to understand because it has the value written in text. On the other hand, they stated that the bars are visually more pleasing but the semantics of the encodings are not evident enough. When then showing the glyphs to *user 1*, they said that they prefer them over the other graphs. The glyphs are easy to understand and one can see if the value is good or bad very fast. The version with the separated arrow parts (see Figure 4.5b.f) is the best encoding according to *user 1*. *User 2* also stated that the version using only text with coloured background is the easiest to understand.

47

They still preferred Figure 4.5b.d over it, because it is faster to read once the concept is understood because one does not have to read a text there. They additionally implied that using the colour scale and showing the value as a line is confusing because they did not realise that they have to read the line. Talking about the glyphs, *user 2* said that they still prefer the bar chart over the glyphs because it encodes the FI value more exact.

Wrapping up discussing the alternative encodings, we then showed the users the first version of the **general layout** presented in Figure 4.2. Both users stated, that the provided components would allow them to effectively solve the tasks they want to perform. *User 2* added that they expect the final version to look more "modern" than what we showed them at this stage. They also both agreed that the **interaction history table** helps in keeping an overview of the different parameter combinations that are tried. In the end, we discussed the **case-based reasoning table**, which is named "*Similar Events*" in this prototype. At the start, *user 1* was confused about what data is shown in this table and after some discussion suggested adding the word "*Past*" to the table title to clarify that historical ground truth data is shown. Both users stated that this feature helps them in gaining trust in the system, assuming that the predictions for the ground truth events match the actual number of sold tickets of these events.

After reviewing the feedback we got during this first evaluation round, we are now able to make a lot of design choices which we elaborate on in Chapter 5 where we discuss the next iteration of our system. In some cases, the users do not agree on visual encodings, in which cases we decide to continue using both suggested versions and making them easy to interchange so we can evaluate them again in the next round. Our hypotheses about which encoding would work best were mostly confirmed by the choices of the users, but they also made comments that we did not expect at all. For example, we did not think that the LPD encoding using only a simple coloured bar (see Figure 4.5a.a) would be liked by the users since one loses a lot of information in comparison to the small area charts. This helps us in fixing mistakes we made in a very early stage of development and confirms that using this user-centred approach helps us in designing a better system.

CHAPTER 5
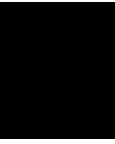
Interactive Prototype

After receiving user feedback for the developed paper prototype (see Section 4.5), we continue with the next iteration of the user-centred design process, which is to develop a horizontal interactive prototype. This prototype gives a broad view of the final exploratory visual event-organisation system (EVEOS) but the functionality is not necessarily complete and the data used is only mocked and not provided by the database. The machine learning model is also not connected to the system at this stage of development. This prototype should, to a certain extent, allow users to perform their tasks and get a first feeling of how the complete system could work. The user feedback from this stage can help us in finding potholes regarding the user's workflow. Additionally, we see if users can interpret the used visual encodings in a more realistic setting than the early paper and screen prototypes. To test if our approach is also applicable to other topics related to event organisation, we also develop a prototype that could be used to create marketing campaigns for events.

In this chapter, we first discuss the general layout of the interactive prototype, showing that it applies to both the tasks of event organisation and marketing campaign creation. We also go into detail about the technical implementation of the project, discussing the component architecture and how the data handling process is set up. Similar to Chapter 4, we then discuss the current state of the separate components. For each component, we elaborate on how users can interact with it and how the feedback of Section 4.5 influences our design choices. Finally, we conduct an evaluation with four domain experts, which provides us with essential information about the usability of the prototype.

5.1   General Layout

In this section, we discuss the general layout of the system in the interactive prototype shown in Figure 5.1a. Overall, the layout is the same as presented in the paper prototype (see Section 4.2). One noteworthy change is that that **filter component** does not exist
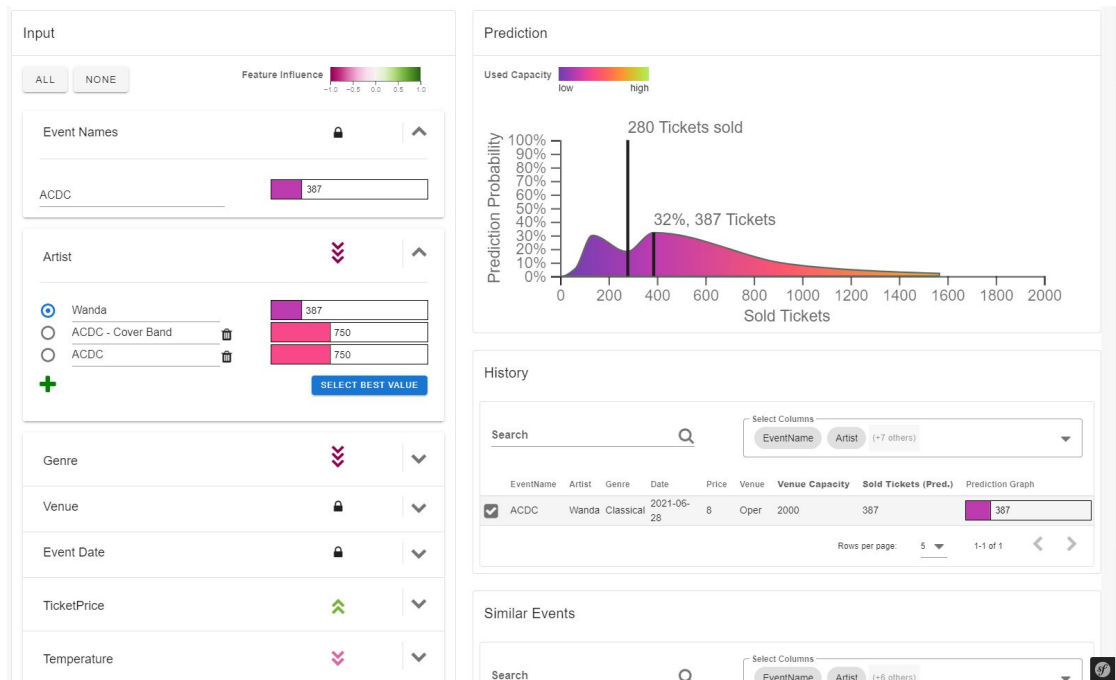
(a) Event Update

(b) Marketing

Figure 5.1: Layout Overviews

as a single component anymore. During the implementation, we realised that it makes more sense to move the filter into the input component for the feature that needs to be filtered as shown in Figure 5.10 and Figure 5.8. Also, since we now encode all three prediction values (NT, OR, PP) in a single graph (see Section5.3) we only show a single result view compared to Figure 4.2 where two views are used.

Talking to the domain experts, they stated that they also can imagine using the system to check the predictions for already set events where they then could **update parameters** to optimise the remaining ticket sales. Making only a few changed, we can solve this task sufficiently. As shown in Figure 5.1a, the parameters that cannot be changed anymore for an event where ticket sales already started are marked with a *lock* glyph and their input field is disabled. Additionally, in the result view, we add a line that shows how many tickets were already sold. We think that this can help users in getting a feeling of how well the prediction will match the final ticket sales.

As stated before, we want to try to apply the developed structure and input encodings to **multiple use cases**. Therefore, in addition to event organisation, we develop a prototype for creating marketing campaigns, which is a task our target group is also performing regularly. As shown in Figure 5.1b, the layout and the encodings fit this task as well. We only change the input features to the ones needed to create a marketing campaign and adopt the legend of the result view. To make this prototype functional, we would have to implement a machine learning model that predicts how well the marketing campaign will perform, which was not done in the scope of this thesis.

The user feedback given in Section 4.5 led us to disregard most of the proposed encodings for LPD and FI. As shown in Figure 5.1, we selected two encodings each and integrated them into the interactive prototype. We can switch between the encodings using URL parameters, which is useful in testing and presenting the prototype. The chosen encodings are discussed in detail in Section 5.4.

5.2   Implementation

Our proposed EVEOS is integrated into the ticketing system of Absolut Ticket [1]. It uses a PHP Symphony [9] back-end for communicating with the database. The front-end is developed using the JavaScript framework Vue.js [10]. To implement the proposed visualisations, we use D3.js [4]. Following, we give an overview of the front-end architecture of the EVEOS focusing on data and event handling as well as important implementation details of the components.

To achieve a high degree of generalisation, we split the application into several components. Especially for the visual encodings of LPD, FI and the data type-specific input components (IC), this is important so they can be used in other interfaces as well. As described in Section 5.1, this allows us to use these components for developing interfaces for creating events as well as marketing campaigns. In Figure 5.2, we show a diagram of the component architecture. The blue components are generalised, while the green (event)
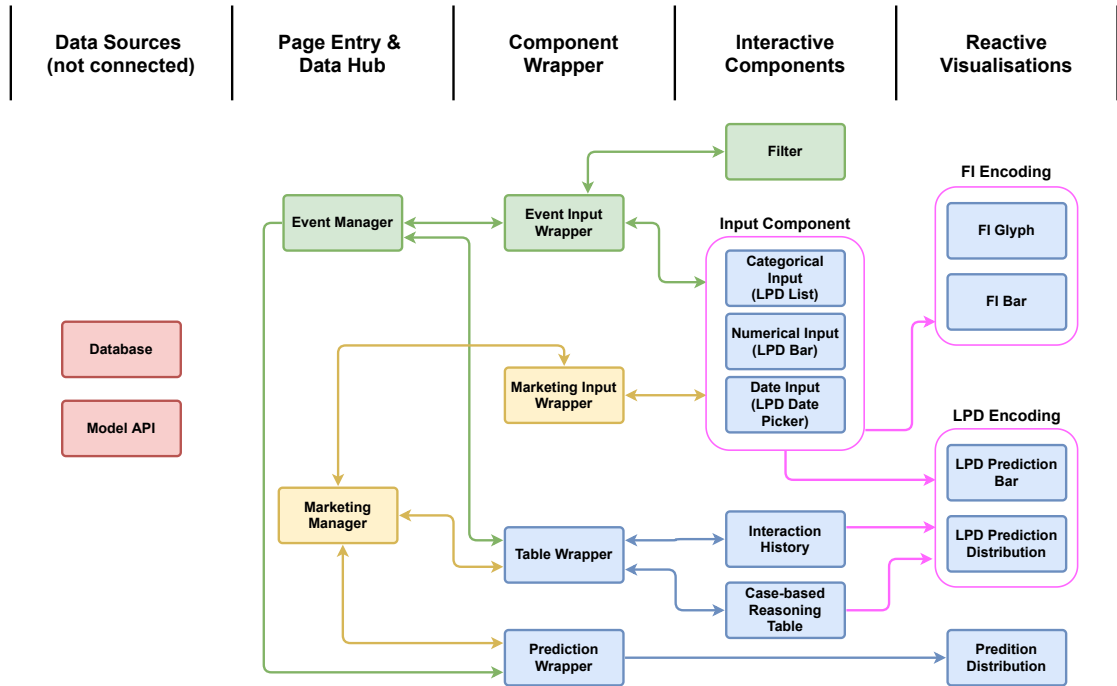
Figure 5.2: Diagram of front-end architecture. The direction of the arrows indicates data and event flow. Blue: general components. Green: event-specific components. Yellow: marketing specific components. Pink: component groups. One or more components of a group can be used at the same time.

and yellow (marketing) components are specific for their given use case and handle the corresponding data. The pink boxes indicate that the included components belong to the same group and can be used interchangeably depending on data type in case of the input components or depending on user preferences for the LPD and FI encodings.

The **manager component** is the entry point to the system and is the highest point in the components hierarchy. It is responsible for hosting all other components and provides a central data hub. The **main data object** that contains all ML model input data is created in the manager together with all other data needed in the system like the data object for the interaction history table. Exploiting the reactivity system of Vue.js (see Figure 5.3) we can manage the main data object by passing it down from the manager to the other components without having to actively send changes in the data back to the manager. Each component receives the parts of the main data object it needs and when data is changed somewhere in a component it automatically changes in all other components as well. The only thing we need to do is to handle the change event if we want to perform some additional computation on a change, like making new predictions. The event flow of this case is illustrated in Figure 5.4. When the model input data is changed in an IC or a row is selected in the interaction history table, first, the manager component is notified. The manager component then triggers the computation of a new
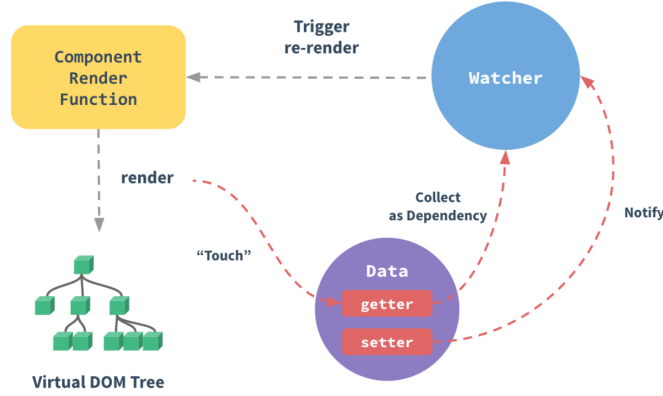
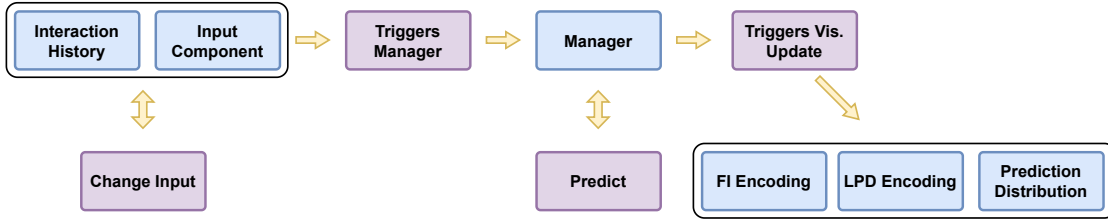Figure 5.3: Vue.js data watcher pipeline. Components are re-rendered on change. [11]

Figure 5.4: Abstract event flow on input change.

prediction. Once the ML model is connected to the EVEOS, an API call will be sent to the model to make the prediction. At this prototype stage, we simply compute random values. The change of the prediction result triggers the re-render process in the reactive visualisations At no point during this event flow, we actively send data. We always only update the reference object and trigger events.

The components that are directly connected to the manager are diverse **component wrappers**. They can host multiple other components and have the job of generalising data flow. The **event and marketing input wrappers** host all the input components for the specific features. Here the main data object is split and only the references to the corresponding features are passed down to the input components. The event triggers of the different input components are also collected here and only the general change event *dataChanged* is emitted to the manager. The input wrapper additionally hosts the **filters** of the ICs if there are any. By not adding feature specific filters to them, we can keep the ICs general. The filters are implemented in separate components and are plugged into the ICs using the Vue.js slot API [12]. Applying the filter to the main data object is also handled in the input wrappers.

Other wrapper components are the **prediction wrapper** that provides an interface for the result view and the **table wrapper** that hosts both the interaction history table and the case-based reasoning table.

The next level in the component hierarchy is the layer of **interactive components**. These are the components with which the users are directly interacting, for example, by setting input parameters, filtering data or making selections in the interaction history. As the name suggests, the **input components** (ICs) are responsible for handling the manipulation of ML model input. According to the data types we discussed in Section 4.1, we propose three distinct ICs for: categorical inputs, numerical inputs and date inputs. For each feature in the corresponding data set, the input wrapper hosts an IC of the correct type. The ICs all contain the functionality to add and remove data from the main data object. When adding data, we need to be careful and provide all necessary data since adding samples is immediately propagated to all other components.

Categorical input components feature the possibility to show one of three input encodings: free, selectable, fixed. They are elaborated on in Section 5.4.2. The **numerical** input components implement the LPD bar proposed by Krause et al. [40]. An LPD bar consists of a rectangle, an x-axis and a draggable circle. The colours of the rectangle are set according to the prediction value at the samples along the axis. The circle is also coloured according to the sample it currently shows. Clicking on a position on the rectangle, the circle is set to this position. Additionally, we listen to value changes from the outside to reactively update the circle position on change. Numerical data objects also always contain a minimum and maximum value to properly scale the x-axis. To implement the **date input** component, we use the customisable date picker by the Broj 42 group [2]. We enhance the date picker by adding the functionality of colouring the date fields in the calendar view. The colours are derived using the prediction value for the given date. Same as for numerical inputs, we listen to changes of the selected date from the outside to reactively update the component.

The **reactive visualisations** are at the bottom of the component hierarchy and represent diverse visualisations of the current predictions, all of them using D3.js [4]. As the name states, the users cannot directly manipulate them, they only change when the prediction changes. The **prediction distribution** component represents the main result view showing the prediction of the currently selected ML model input. It is implemented as an area chart, taking the points provided by the prediction and interpolating them using a D3.js curve function. We provide multiple parameter settings, which are elaborated on in Section 5.3, to be able to test different configurations and figure out what the domain experts like the most. The colour of the area is computed by interpolating the colours of the prediction values according to the used colour scale using an HTML linear gradient. We develop two potential **LPD components**. One is a small version of the prediction distribution component and the other a prediction bar. They, and the two **FI encodings**, are further discussed in Section 5.4.
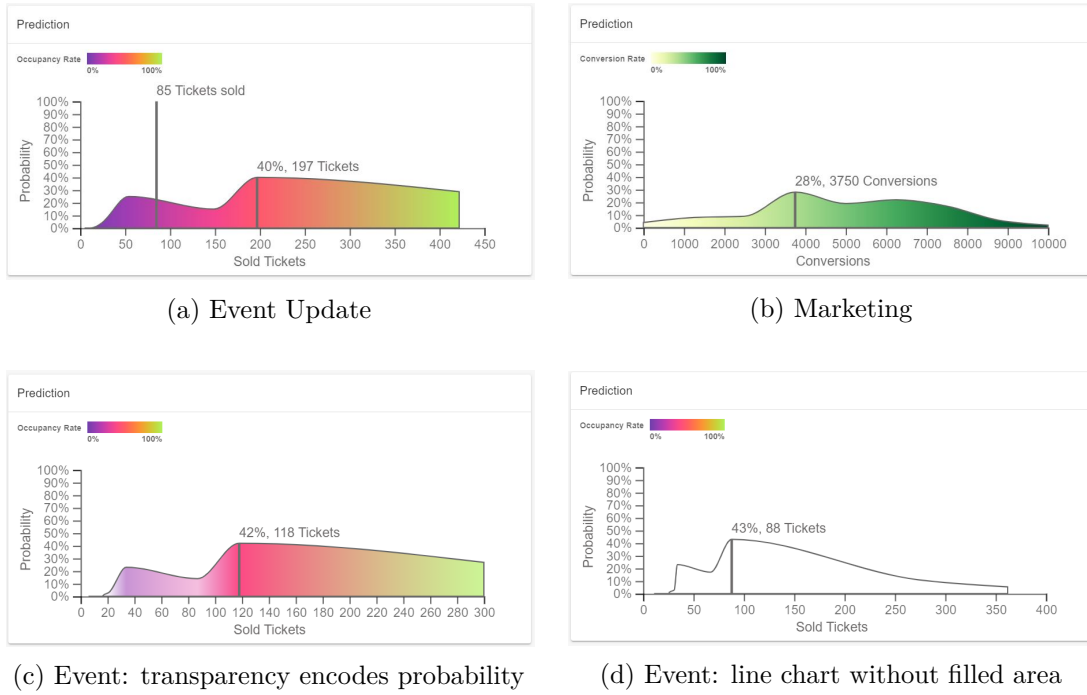
(a) Event Update

(b) Marketing

(c) Event: transparency encodes probability

(d) Event: line chart without filled area

Figure 5.5: Main prediction result views

5.3 Result View

As discussed in Section 4.3 the goal of the result view is to present the user with the prediction result. The feedback provided by the users in Section 4.5 still leaves a lot of areas open, which is why we implement four different versions of the proposed area chart as shown in Figure 5.5. We elaborate on them using the previously presented use case of predicting ticket sales for events but they also can be used to encode the results of a marketing campaign prediction as shown in Figure 5.5b. For all of them, we encode the prediction probability PP on the y-axis, the number of sold tickets NT on the x-axis and the occupancy rate OR using the colour of the area. To interpolate the line between the prediction points, we choose the D3.js curve *MonotoneX* [3], which produces a cubic spline that preserves monotonicity in y, assuming monotonicity in x. This fits our data because the prediction points are strictly monotonically increasing along the x-axis.

Taking a closer look at the four examples in Figure 5.5, it is apparent that the maximum value of the x-axis is different for each of them. The computation of the maximum value heavily depends on the use case, the result view is used for. When predicting ticket sales for an event, the maximum number of tickets that can be sold depends solely on the capacity of the venue. We decided to set always set the maximum value to the venue capacity of the currently selected venue. When working with marketing campaigns, the maximum value of conversions depends on the customer reach that can be derived from a combination of target group features.
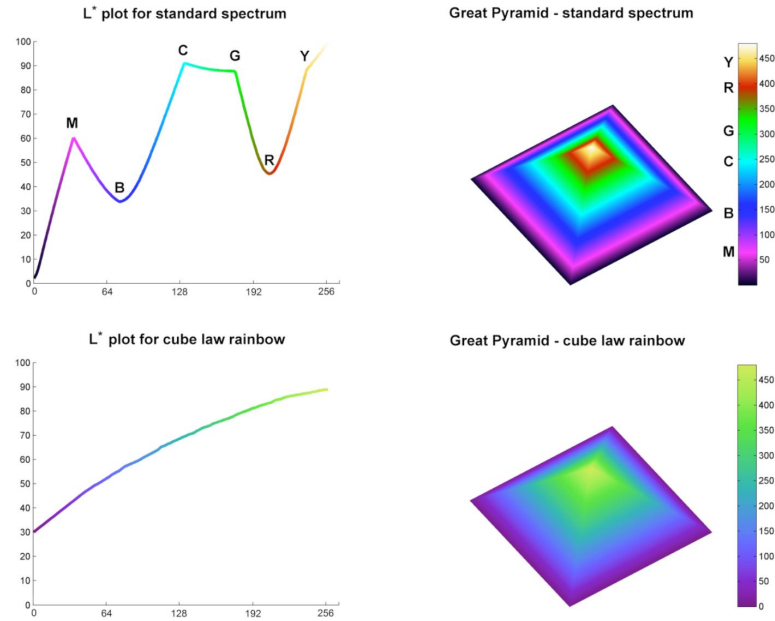
Figure 5.6: Perceived lightness of the rainbow (top) and the cube law rainbow (bottom) [54].

Since OR lies between 0% and 100%, the **colour** represents a sequential value suggesting the usage of a sequential colour scale for encoding the values. We create versions with single-hue (see Figure 5.5b) and multi-hue (see Figure 5.5a) scales. The proposed single-hue scale uses saturation of the colour green. The multi-hue scale we use is the inverse scale of the cube law rainbow scale proposed by Niccoli [54]. It is a novel mapping of the rainbow colour scale that is **perceptually sequential**.

We are aware of the criticism that the rainbow colour map faces when being used for encoding sequential data. Numerous works describe that its main problem is that it cannot be perceptually ordered [19, 65, 43, 52]. Niccoli himself states that, even though the rainbow map is ordered according to the wavelengths of the colour spectrum, humans do not perceive it as such [55]. According to Niccoli, humans, when ordering colours, focus on the perceived lightness of colours. This is echoed by Kovesi [39], who describes in his work that to achieve uniform perceptual contrast in a colour map, the perceived lightness of the colours is the most important factor. As shown in Figure 5.6 (top), the lightness of the rainbow scale is not uniform at all. Inspired by Kindlmann et al.'s [38] approach on luminance controlled interpolation, Niccoli defines his cube law [54] that he applies to the rainbow colour scale to generate a perceptually sequential version of it. The resulting lightness scale is shown in Figure 5.6 (bottom).

We can imagine that using Niccoli's scale, it could be easier to distinguish between good and bad results since they are more distinct on this scale than on a single-hue scale. The scale goes from purple over pink and orange to green, with purple representing low and green representing high values (see legend in Figure 5.5a). We think that this allows the user to quickly categorise in which of these areas the prediction lies. On the other hand, we believe that perceptually ordering colours is still easier on single-hue scales. This leads us to think that the single-hue scale could be better suited for our data. In the following elaborations, we use Niccoli's scale [54] for colour encodings.

As shown in the examples in Figure 5.5, we add a legend of the colour scale to the result view. We think that this helps the users to understand what the colours are used for and it is also necessary to understand the LPD encoding discussed in Section 5.4. In the previous feedback round the users stated that filling the area chart with the colour scale was the most apparent solution to them. As shown in Figure 5.5a, this provides an implicit second x-axis for OR. We think that using this approach it is easier to see where on the scale the value lies, rather than having to compare the colour of the area to the scale legend as one would have to do using the version in Figure 4.3c.d.

The users also stated that the line highlighting the prediction value with the highest PP is helpful but they additionally requested a textual legend, which we add in this iteration. The combination of line and text provides a connection to LPD prediction bar that we discuss in Section 5.4. As shown in Figure 5.5a, we also add a line showing how many tickets were already sold when working with events where the ticket sales already started.

In the paper prototype we propose to use transparency to highlight the important regions even more (see Section 4.3). *User 1* stated in their feedback that they first did not understand why transparency was used, but after explaining the concept said that it was their favourite encoding, which is why we decided to implement this version as shown in Figure 5.5c. To get the correct transparency values t we perform a linear interpolation between the highest PP and 0. According to t, we set the transparency of the area beneath the curve. When creating the paper prototype we thought that this approach would be the best because it guides the focus to important regions of the graph. After seeing the results produced by using this method, we think that this is still true but also see the downsides of using transparency. The colours in regions with high transparency are distorted and in our opinion, it is hard to interpret them.

In their feedback, *user 2* stated that the colours were confusing to them and that they would like to have only a curve. As shown in Figure 5.5d, this completely removes the information about OR, which is why we disregard this option. We think that *user 2* did not understand the meaning of the colour and that with the added colour scale and the other mentioned improvements, it will now be more apparent to them.

5.4   Input View

The input view is placed on the left side of the interface as shown in Figure 5.1. In this area, the users can interact with the ML model input parameters. We show all the features corresponding to the model in separate input components according to their data type, which we discuss in the following sections. The users can select feature values, add them or remove them. This is also the area where multiple feature values can be compared using local partial dependence (LPD) information. Additionally, feature importance (FI) information is shown to guide the user towards features that still can be optimized as discussed in Section 5.4.1.

Comparing the layout of the input view with the layout of the paper prototype in Figure 4.2, they are almost the same. In the evaluation of the paper prototype in Section 4.5, the users stated that they are satisfied with the proposed layout and did not have any change requests. We make two changes that result from realisations during the development. First, when implementing the input view, we realised that showing all input components underneath each other needs a lot more space than we thought. To make it more compact, the components are now implemented as expandable cards that open and close on clicking on the card header. Since the FI information should always be visible, it is included in the card header. Additionally, we provide buttons to open and close all cards at once. The second change concerns the LPD information in the input components. In the paper prototype, we proposed separate LPD encodings for *NT* and *OR*, which now are shown together in a single graph.

5.4.1   Feature Importance

With the measurement of feature importance (FI), we want to show to users the quality of the current value selection of a given feature. FI is a value between -1 and 1, with negative values suggesting that the selected value is bad compared to other possible values. To visualise this measurement, we use a diverging colour scale that goes from pink for negative values, over white at zero to green for positive values. We choose this scale instead of a red, green scale, which are the signal colours for bad and good in the cultural region of central Europe, to make the interface more accessible for people with red, green colour blindness. As shown in Figure 5.7, we place a legend of the colour scale above the input components. Each component then has a graph in its header that is always visible.

In Section 4.4.2, we propose multiple encodings for FI. The evaluation with the users in Section 4.5 leads us to implement the two versions shown in Figure4.5b.d and f. Figure 5.7a shows the implementation of version f using **arrow glyphs**. Depending on the FI value, the glyph consists of one, two or three arrows either pointing up for positive values or down for negative values. The threshold for adding arrows are at $\pm\frac{1}{3}$ and $\pm\frac{2}{3}$ while the colour of the arrows is set continuously according to the FI value. Version d was not completely apparent to the users at first glance, but *user 2* stated it is their favourite version because it is more exact than the arrow-glyph. As shown in Figure 5.7b, FI

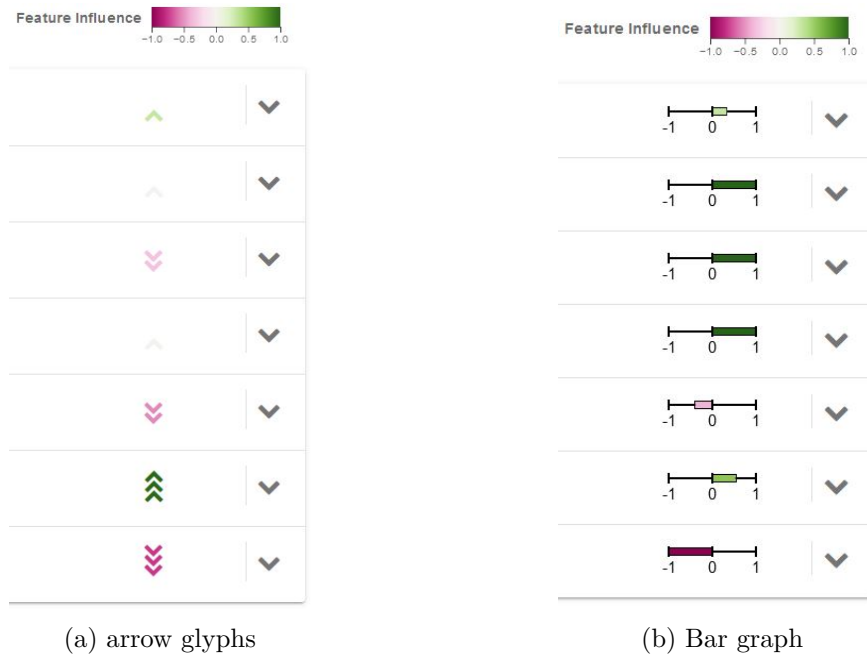(a) arrow glyphs

(b) Bar graph

Figure 5.7: Feature importance encodings.

is encoded using a **bar** on a scale from -1 to 1 with the bar being centred at value 0. The presented value is shown using the bar length and colour. In an effort to make this encoding more apparent, we add axis labels to the graph.

Comparing these two versions, the bar graph is more exact because it features value labels and the bar length directly represents the current value. The arrow glyphs abstract the value a little by only adding new arrows on the given thresholds. Also, they do not provide labels and even though the colour is exact, colour saturation and colour hue are worse encodings for quantitative data than length, as stated by Munzner et al. [52]. On the other hand, we think that for the measurement of FI it is not important to be exact. It should serve more as a rough indicator of the feature value quality. We think, that using the arrow glyphs it is easier to quickly spot if the quality is good or bad and also to compare the quality of different features. If a feature has an arrow more or less than another, we immediately see this difference. Also, we think the glyphs look visually more pleasing than the bar graph. On the other hand, we can imagine that users could also prefer the bar graphs if they like working with numbers and are interested in the exact values.

5.4.2 Categorical Features

As discussed before, we propose the usage of input encodings that fit the data type of the feature at hand. In this section, we discuss possible input versions for categorical features and how we can show their LPD information. Since in the paper prototype we

(a) Locked input

(b) Fixed input

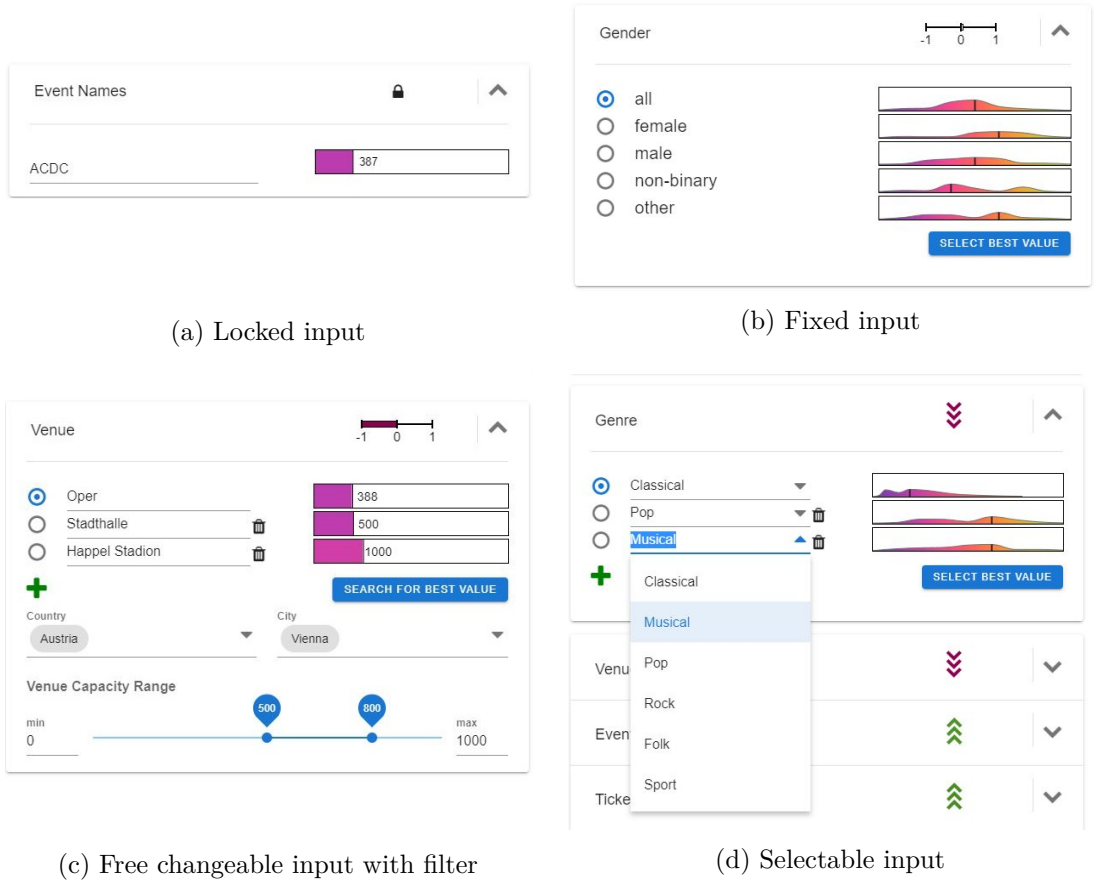(c) Free changeable input with filter

(d) Selectable input

Figure 5.8: Different input types for categorical features.

focussed on the LPD encodings and not so much on the layout of the input fields, we do not have any feedback from the users regarding this topic.

Categorical data is strictly categorical and does not have any other underlying properties. The only thing that is important to keep in mind, is the **quantity of values** a categorical feature can have. We distinguish between three types of quantity. A feature can have a small ($<= 10$) number of possible values, a large (> 10) number of values, or they can be an open set with an infinite amount of values. Also, a feature can have only a single value, which in our case happens when users work with events where the ticket sale already started and therefore a feature is locked to a value that cannot be changed anymore.

When developing the categorical input components, we realised that these quantity categories need distinct input selection fields. As shown in Figure 5.8a, when there is only one locked value, there is nothing else to show, but this value with its LPD information. For features that have less than 10 possible input values, it is feasible to show all of them

at the same time. An example for that is the gender feature when creating marketing campaigns as shown in Figure 5.8b. We define this feature to have five possible values that are all shown and can be selected using radio buttons. When the number of possible values gets too big, it becomes impractical to show all of them. We propose the usage of a drop-down selection as illustrated in Figure 5.8d. Features like the event name have an infinite amount of possible values since one can define whatever name they want. To allow the user to properly select such feature values, we provide a text input field that can be filled with any value the user desires (see Figure 5.8c). Additionally, for features with open sets and more than 10 values, it is necessary to provide opportunities to manually add and remove sample input fields. We include an add (+) button that inserts a new empty selection field and a remove (trash can) button for each field except the currently selected one. We also add a *Select Best Value* button that on click selects the best of the visible samples. This allows users to quickly get the best sample without having to compare all of them.

As discussed in Section 4.4.1, we suggest showing **LPD information** besides the selected samples. The LPD information should enable the users to compare the prediction outcome of multiple samples of feature values. We think that putting the LPD graph beside the sample and having them all underneath each other is a good layout to perform that task. The two encodings that the users suggested to be used in the evaluation of the paper prototype are the ones shown in Figure 4.5a.a and f. The version in Figure 4.5a.a is an abstraction of the prediction result using only the prediction value with the highest probability PP, As shown in Figure 5.8c, we use a **bar graph** where the length of the bar encodes NT and its colour encodes OR. Even though *user 2* suggested that adding text labels for NT would suggest that PP of this prediction is 100%, we believe that having a label helps to connect this abstracted view to the main result view. In Figure 5.8b we present the implementation of version of Figure 4.5a.f. It is a small variant of the prediction distribution used in the result view (see Section 5.3). Since there is not a lot of space available in this small scale view, we refrain from using axis and labels in this view. The line highlighting the point with the highest PP is crucial to make the area graphs comparable to each other. Without it, it would be difficult to see where each curve has its highest point on this small scale.

We think that since both versions use the visual variable of position to encode the prediction value with the highest PP, it is similarly easy for both of them to compare the prediction values. Still, they are completely different visualisations that focus on particular aspects of the prediction. The prediction distribution provides complete information about the prediction including the probability. Users could see in this preview, that for example there exists a second local maximum in the prediction probability that maybe would suggest a better prediction than the highest one. The bar does not provide this information. It focuses completely on the prediction value with the highest PP. This way, it can provide more information about this specific value. It features the text label representing NT and the information about OR is more prominent since the whole bar is coloured accordingly. We hypothesise that users will find the small scale prediction
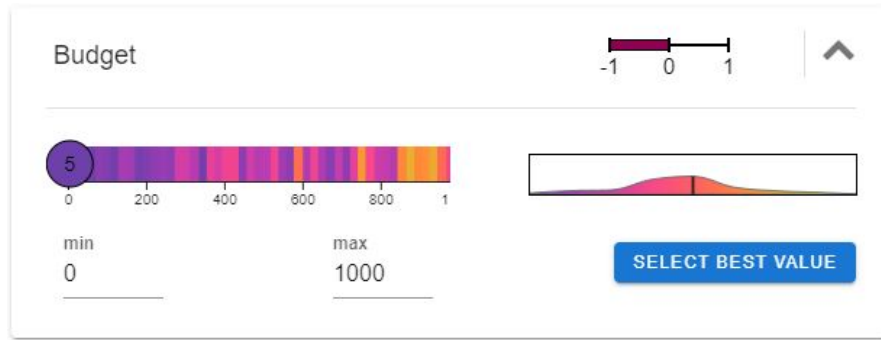
Figure 5.9: Numerical Input. Local partial dependence bar for budget

distribution superior to the bar graph because it provides more information and they are already familiar with it from the main result view.

As stated in Section 5.1, the filter components are moved to their corresponding features. As shown in Figure 5.8c, the filter is now placed underneath the sample selection of the feature. In general, we try to use data-type specific input fields for the filter variables. For numerical range data, for example, we employ a range slider with the possibility to adjust the slider range using minimum and maximum value fields.

5.4.3   Numerical Features

As mentioned before, we propose the usage of the LPD bar of Krause et al. [40] for the value selection of numerical features. As shown in Figure 5.9, it is a 1D bar graph where each value sample is coloured according to its prediction value. In our case, this value is the occupancy rate OR. The currently selected value is shown in a circle over the bar, that can be dragged to change the value. When showing the screen prototype of the LPD bar to the users, they understood it well and did not have any further suggestions towards it. To make the range of the bar changeable, we add input fields for minimum and maximum values. Same as for categorical feature inputs, there is a button for selecting the best value on the scale.

During the implementation of the LPD bar, we had to realise that it is not possible to get predictions for every single value that is shown in the scale for two reasons. The first one is screen resolution. When the value range gets too big, one pixel on the screen represents more than one sample. If these samples have different prediction values, we cannot show these values accordingly. Secondly, thinking ahead to the incorporation of the ML model, increasing the value range increases the number of predictions that need to be made. So when the range becomes too big, this could lead to problems in the performance of the model. To avoid these problems, we decide to take equidistant samples from the scale. After a trial and error approach on a screen with a resolution of 1920×1080 pixels, we decide to take 50 samples if the selected range has more than 50 values.

Figure 5.10: Filterable Date Input.

One of the key features of the LPD bar is that it can be used for input selection and displaying LPD information at the same time. This makes this encoding very compact and easy to use. On the other hand, because our predictions include three distinct values, we cannot show all of the values encoded as colours on a single bar. To be consistent with how we handle LPD information at categorical features, we show a detailed LPD encoding of the currently selected value beside the LPD bar as illustrated in Figure 5.9. In the paper prototype we also thought about showing this as a tooltip when hovering over a sample on the bar (see Figure 4.4b.b), but since the proposed solution is almost the same with the users only having to select the value they are interested in with a click and it maintains consistency we choose not to use a tooltip.

5.4.4 Date Features

When discussing with domain experts how date variables should be set, they suggested using a date picker since it is the standard input component for such data. The other option we proposed in the paper prototype was to use the LPD bar for dates as well since we can represent ordinal data like dates on this bar. But the users preferred the date picker over that solution.

As shown in Figure 5.10, we enhance a standard date picker by making the date fields in the calendar tintable. This allows us to show LPD information for each date in a similar fashion to the LPD bar. We hypothesise that since the domain experts understood the

63

Figure 5.11: Interaction History Table.

LPD bar for numerical features without problem, they will also find this encoding easy to interpret. To show more detailed information about a date, we again add an LPD encoding besides the selected date field. The date picker shows a single month and the user can switch between months. Users therefore can compare the dates within a month, but not over several months combined.

When organising an event, organisers usually have a rough time range in mind in which the event should take place. Therefore, we make the date input filterable using a start and end date as well as a selection of days. When the filter is applied to the date picker, only the dates fulfilling the filter criteria are filled with their LPD colour, the others stay white.

5.5 Interaction History

The goal of the interaction history is to enable the comparison of parameter combinations and allow users to transfer previous settings to the input view. In their feedback, the users were satisfied with the proposed table shown in Figure 4.2, which is why we do not make any significant changes to its layout. As shown in Figure 5.11, the proposed table provides a search field and a column selection where users can choose which fields they want to see. On the right side of each row, we show a LPD information encoding, in this case the small version of the prediction distribution graph, using the Vue.js slot API [12] to integrate the graph into the table.

Whenever the user changes the parameter selection in the input view, a new row is created in the interaction history. As described in Section 5.2, we add the new row to the history data object in the manager component and the table is automatically re-rendered with the new data. The new row is added as the new first row and also is selected because it represents a collection of the currently selected values. We believe that having multiple parameter combinations including their prediction underneath each other allows the users to easily compare them and find the combination that gives the best prediction. We have to differentiate between the comparisons that can be made in the interaction

Figure 5.12: Similar Events Table.

history and the input view. In the history table, the focus lies on comparing complete parameter combinations, while in the input view one can compare different value settings of a single feature. Filtering and sorting the table additionally supports the users in this task, if the interaction history has many entries.

When a history entry is selected, its parameters are selected in the input view. Since users can add am remove feature samples for categorical variables, the sample that is transferred from the history table may be no longer available in the input component. In this case, the sample is added to the selection. Similarly, for numerical features, the users can change the value range they want to investigate. If the value from the history table is outside of the currently selected range, the range settings are automatically changed to fit the value inside.

5.6 Case-based Reasoning Table

The case-based reasoning (CBR) component should present users with information about ground truth data that is similar to the current ML model input. In the case of event organisation, the ground truth data are past events where we know how many tickets were sold. Same as for the interaction history, the users were satisfied with the table view we proposed in the paper prototype. The layout of the table is similar to the interaction history (see Figure 5.11). The idea behind the CBR tables functionality is that users can fetch input combinations that are similar to the current input using a button. The button click triggers a call to the database or to a clustering API that then returns the most similar input combinations. Since this prototype is not fully functional, right now random data is filled in the table when the button is clicked.

We want to highlight the comparison between the prediction the model makes for the ground truth data input and the known result. Therefore, we make the corresponding column names bold and also try to add information to the LPD encoding. As shown in Figure 5.12, when using the small prediction distribution chart, it now features two lines. One of them highlights the prediction value with the highest probability and the other

shows, in our case, how many tickets were sold for this similar historic event.

CBR should allow users to get a feeling if the current prediction is reasonable when compared to ground truth data. If this is the case, past works [21, 80] show that this leads to increased trust of the users in the system. In the proposed prototype, the prediction of the current input and the ground truth data are shown juxtaposed, one in the main result view, the other in the CBR table. We believe that this type of comparison is sufficient to let the users perform CBR. Another solution that we propose to the domain experts in the evaluation of this prototype (see Section 5.7) is to create a superimposed view where users can select one or more rows in the CBR table and their predictions are added to the main result view to create a single visualisation for comparison. This would provide more information in a single spot but we hypothesise that it could lead to a cramped view that has an information overload.

5.7   Evaluation

Following the user-centred design cycle (see Figure 2.4), we perform an evaluation after each prototyping iteration. The goal of this evaluation is to finalise the choice of visual encodings used where there are still multiple options. Likewise, we want to investigate if users can interpret the proposed visualisations correctly. To test the usability of the prototype, we ask the users to perform small subtasks without explaining the functionality of interface components beforehand.

For this evaluation, we perform semi-structured interviews with four domain experts. The *users 1* and *2* are the same as in the evaluation of the paper prototype and both younger than 35 years of age. To get a broader perspective, we searched for older domain experts. *Users 3* and *4* are both older than 50 years of age and describe themselves as "not so affine when working with computers". Due to COVID-19 restrictions, we only could perform eye-to-eye interviews with *users 2* and *3*. With *users 1* and *4*, the interviews were done via video calls and using a remote desktop application, so the users could interact with the interface. During the interviews, we first discuss the general concept of the proposed exploratory visual event-organisation system EVEOS with each user, explaining that there will be artificial intelligence in the background making predictions about ticket sales. Then, we go through the separate parts of the EVEOS for both the event organisation and the marketing use case. We first discuss the result view, talking about the proposed colour schemes and how it is interpreted in general. We transition to the input view where we go through the input components for the features. There, the users are asked to interpret the two proposed LPD and FI encodings. Additionally, they need to perform simple input and filtering tasks to evaluate their usability. Afterwards, we discuss the interaction history and ask users to compare different parameter settings before talking about the case-based reasoning table. Finally, we ask users about their overall impression of the system and if they have any questions about parts that we did not mention before.

Talking about the **result view** presented in Figure 5.5 with *user 1*, who already knew about the encoding from the screen prototype, they were able to interpret the graph without problems. They realized that the x-axis depends on the venue capacity from the given context. Nonetheless, *user 1* suggested that a more detailed legend giving textual information about all the information encoded in the graph would be useful to them. *User 2* was also able to interpret the view but first thought that the colour represents the probability. After taking a closer look and reading the colour scale legend, they realised that it is about the occupancy rate OR and interpreted it correctly. *User 3* was initially confused about the y-axis of the chart, thinking it represents the percentage of OR. They stated, that the provided text legend for the point with the highest PP mislead them. After reading the label of the y-axis, they then interpreted the values correctly. Same as *user 1* they suggested a more detailed legend to avoid these confusions. For *user 4* the context of the prediction was not evident when talking about the result view and we had to explain it a bit more in detail before they realised what they were seeing. After that, they also stated that it is crucial to have a better legend.

Discussing the proposed **variations of the result view**, all users favoured the version shown in Figure 5.5a. Using transparency, *user 1 and 3* stated that it could help them in focussing on the areas with high probabilities but they also agreed with the other users that the distortion of colours was more confusing than helpful, confirming our hypothesis about that problem. They all stated that using no colour at all as shown in Figure 5.5d, they would not be able to read any information about the occupancy rate. Talking about the usage of the single-hue and multi-hue **colour scales**, opinions were more divided. *User 3* preferred the single-hue scale because it gives them an obvious sequential encoding and they were confused about the usage of multiple colour hues. The other users were also sceptical about the multi-hue scheme at first. When discussing this topic more closely and also showing the LPD encodings in the input view using the two colour schemes, focussing on the colour and not so much on the visual variable of position in the graphs, they stated that using the multi-hue scheme it was easier for them to quickly categorise the prediction into four groups according to the four prevalent colours on the scale. With the single-hue scheme, they were only able to distinguish between two categories, good when the colour was dark and bad when it was bright. This confirms our hypothesis about the benefits of the multi-hue colour scale. We believe that this is a very interesting outcome since most of the state-of-the art works on colour scales advises against the usage of rainbow colour maps. It seems to us that Niccoli's approach for creating a perceptually sequential rainbow map [54] is a success.

Going through the input components in the input view (see Section 5.4), we started with discussing **categorical inputs**. The users were asked to perform selections and to add, remove and manipulate samples, which they were able to do without problems. *User 4* pointed out that for features that have additional information connected to them, like the city and country a venue is placed in, they would like to see this information somewhere. To evaluate the two proposed **LPD encodings** for categorical inputs, *users 1* and *3* were shown the small prediction distribution first, while *users 2* and *4* got to

	LPD Prediction Bar	LPD Prediction Distribution	FI Glyph	FI Bar
User 1	yes	no	yes	no
User 2	no	yes	yes	no
User 3	no	yes	yes	no
User 4	no	yes	no	yes
Total	1	3	3	1

Table 5.1: Evaluation Results

initially see the bar graph encoding. *Users 1* and *3* immediately realised that the small prediction distribution is supposed to be a preview of the prediction result for the feature samples. Using the bar graph, this connection was not made without a short explanation first. Afterwards, we showed the second encoding to the users and discussed the positives and negatives. *User 1* stated that they prefer the distribution graph because it provides more information than the bar graph. They also think that it does not matter that there is no legend provided because one will see it when selecting the sample anyway. *Users 2* and *3* find the bar graph easier to interpret and they state that they can faster compare the main value of interest, which is the one with the highest probability. While *user 3* also articulated that the prediction view is more coherent with the result view and looks nice, the bar graph is still more useful to them. *User 4* deduced by themselves that when using the bar graph, they cannot see the probability of the result which they can in the prediction distribution. They still think the bars are better because they are easier to read and compare. Summing up these findings in Table 5.1, we see that the bar graph is liked by more domain experts than the distribution graph, proving our hypothesis wrong. We thought that since the prediction distribution graph provides more information it is more useful than the bar graph, which seems not to be the case.

The interaction with the date picker for **date inputs** was immediately apparent for all users. Also, the encoding of LPD information on the date fields was interpreted correctly by all of them, except for *user 2* who forgot about the colour scale presented in the result view and was confused about the colours. At this stage of the evaluation, the multi-hue colour scale was used. They suggested adding the colour legend to all input components so it is always visible when needed. For **numerical input components** the users had to answer the following questions: *What is the selected value? How can you change the value? What do the colours on the bar show? How can you change the range of the bar?* For *user 1, 2* and *3* the tasks were solved without problems. *User 4* did not initially realise that the value in the circle represents the currently selected value. They proposed adding a field where the value is shown explicitly also with additional information about the semantics of the value, for example, if the value represents a value in € or *kg*. We also asked the users to select the best value at some input components. When it was easy to see, for example, at categorical inputs with only two or three samples, they selected it directly, else they search for a few seconds and then saw the *Select Best Value* button and used it like intended.

The presented prototype features two input filters, one for venues and one for event dates. The **venue filter** presented in Figure 5.8c, provides filter inputs for the venues country, city and a capacity range. To apply the filter, there is a button called *Search For Best Value*. We asked users to search for locations in Berlin, Germany with a capacity between 2000 and 5000. The selection of the country and city was performed without problems by all users. To get to the correct selection of the capacity, users also had to change the range of the proposed slider. *Users 1* and *2* struggled with this because they were confused by the additional minimum and maximum fields, stating that it is confusing to have to adjust a scale using these fields so one can then select the filter range on the slider. They rather would like to have just a minimum and maximum field where they can directly enter the search values. All of the users struggled in applying the selected filter input. None of them connected the provided button to the filter and stated that its label does not fit the task and since it is placed at the same spot where else the *Select Best Value* button is this destroys continuity. *User 2* suggested moving the button below the filter, so the connection to it is more evident. The same problem was apparent at the **date filter**. Else the interaction with this filter was apparent to the users. *User 2* suggested that when selecting a time range that spans multiple months, it would be nice to see the months all together without having to switch between them in the date picker. When mentioning this to the other users, they agreed that this would be helpful to them as well.

After manipulating some feature values, we drew the attention of the users towards the **FI encodings** presented in Figure 5.7. Again *users 1* and *3* were shown the arrow glyphs first and *users 2* and *4* the bar graphs. Showing the glyphs first, the semantics of this visualisation was interpreted correctly by the users. The bar graph was not that immediately apparent to them. Especially *user 4* struggled with the provided legend stating that they do not understand the meaning of the values -1 and 1 on the scale. After explaining the concept and also showing them the glyph encoding *user 4* stated that the bar graphs are their preferred version because they are more exact and they like working with numbers. The other users preferred the arrow glyphs but also were not content with the provided legend. *User 1* suggested adding the word *current* to the legend title to clarify that the value is connected to the currently selected value of a feature, while *user 2* stated that when using arrow glyphs it would be helpful to have a legend based on glyphs and not just a colour scale. The evaluation summary in Table 5.1 shows that the glyphs were overall the preferred encoding of FI.

Since the users changed a lot of inputs during the evaluation process, the **interaction history** (see Figure 5.11) was filled with a lot of entries. When pointing out this table, the users were asked to find the best parameter combination of this session. All of them were able to perform this task. *User 2* pointed out that it would be helpful to somehow highlight the feature that changed in each row of the table. Overall, the domain experts stated that this table helps them in comparing different event parameter options.

We then discussed the **case-based reasoning** table with the users, which shows historical events with their prediction and actual value of sold tickets (see Figure 5.12). *User 1* and *4* immediately suggested adding the word *Past* to the component title, to clarify that these are historical events. The usage of the table was apparent to the users, with only *user 3* struggling to find the button to fill the table with data. Talking about if a superimposed view would be beneficial for comparing these ground truth events with the current prediction other than having a juxtaposed view like now, only *user 4* stated that a superimposed view would be better because it would show all information at one place. The other users thought that a superimposed view would lead to too much information being displayed in the result view, confirming our hypothesis on this topic.

In the end, we asked users for comments about the proposed EVEOS. Noteworthy comments from *user 1* include that the possibility to generate an initial input is necessary because if users would see the interface without having set any data first it would be very confusing to work with. Also, users could try to find a perfect value for a given feature without having set any meaningful values for other features. *User 4* stated that when using the system they were a little confused by all the dummy data that sometimes did not behave like expected, but they think that when using real data and real predictions, this probably will not be a problem anymore. They also confirmed our user story again by stating that the most important thing they want to get out of the proposed system would be to find event parameters that optimize ticket sales while also filling a venue to its maximum capacity. It seemed to us, that especially *user 4* began to understand more and more about how the EVEOS can be used the longer the talk lasted. They started to deduce connections between fields and projected them to user stories that they were thinking about during the interview.

This round of gathering domain expert feedback helps us in making a lot of final design choices. Especially regarding the visual encodings for LPD and FI information, we now have a well-founded basis for choosing which one to use in the final prototype. The users also pointed out a lot of details that need changing to end up with a highly usable interface. Some of our hypotheses of what would be good design choices were confirmed while in other cases the users reasoning for opposing our theories were not expected, highlighting the importance of including them in the design process.

CHAPTER 6

Final Prototype

The prototype we present in the chapter represents the final iteration of our user-centred design process. Incorporating the feedback from the domain experts (see Section 5.7), we aim to create a prototype that is close to a final version of the EVEOS and provides the complete functionality. This is done for the use case of event organisation since for the creation of marketing campaigns we do not have a back-end that we could connect to complete the functionality of the interface. We connect the database and the ML model API with the front-end. Evaluating this prototype, we want to eliminate final faults in the system. Additionally, we are interested in investigating if the usage of real data increases the usability of the EVEOS and how the longer system response times that arise from communicating with external services influence the user experience.

The structure of this chapter is similar to the previous chapter. We first discuss the changes in the general layout of the EVEOS and elaborate on how the database and model API are added to the system architecture. Next, we go over the changes to the separate components. The input filters are now also connected to the database, so we describe the data fetching process. The same goes for the case-based reasoning table that now fetches similar events from the database. We then perform a round of evaluation with domain experts and adopt their feedback into the final version of the EVEOS.

6.1 General Layout

In Figure 6.1 we present an overview of the final version of the proposed EVEOS for the task of event organisation. The layout of the interface does not have any significant changes when compared to the previous iteration presented in Figure 5.1. It can be seen that we add a missing input component: the artist feature. Additionally, after reviewing the feedback of the domain experts (see Table 5.1), we commit to the use of the FI arrows glyphs for encoding feature importance information and to the use of the LPD prediction bar for LPD encoding information. Using the LPD prediction bar makes the
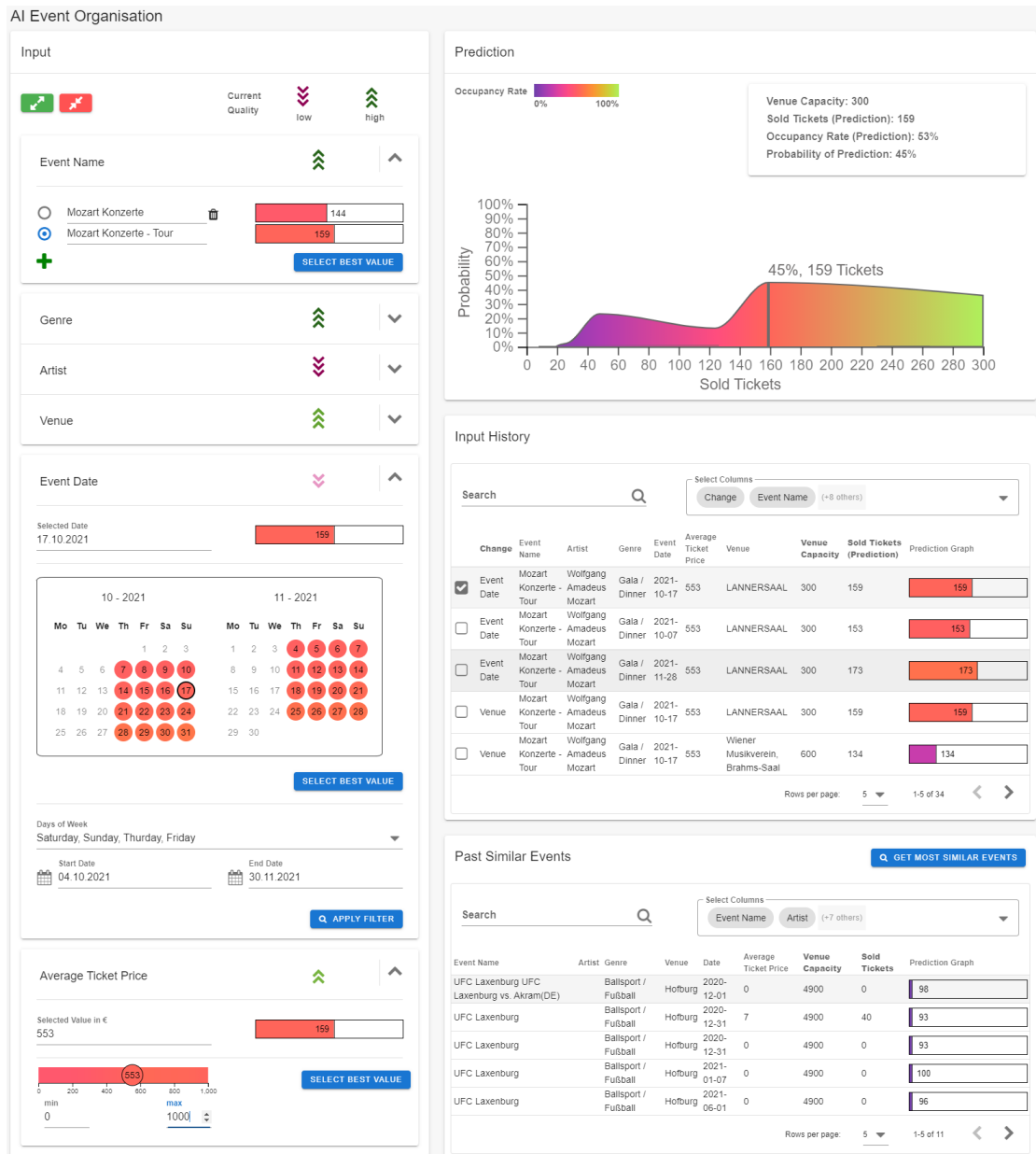
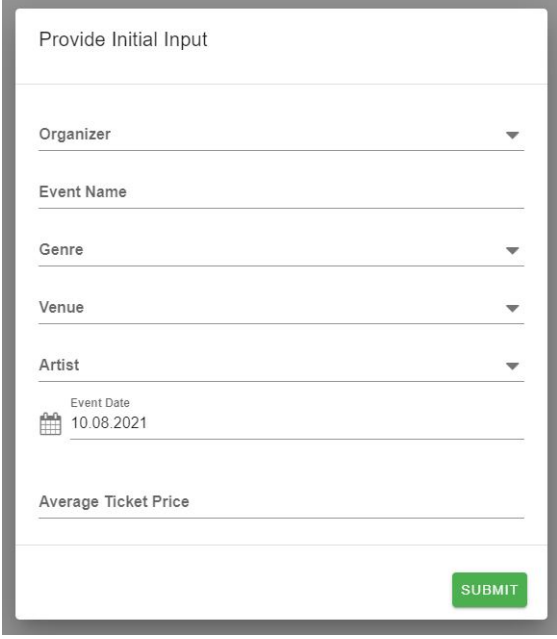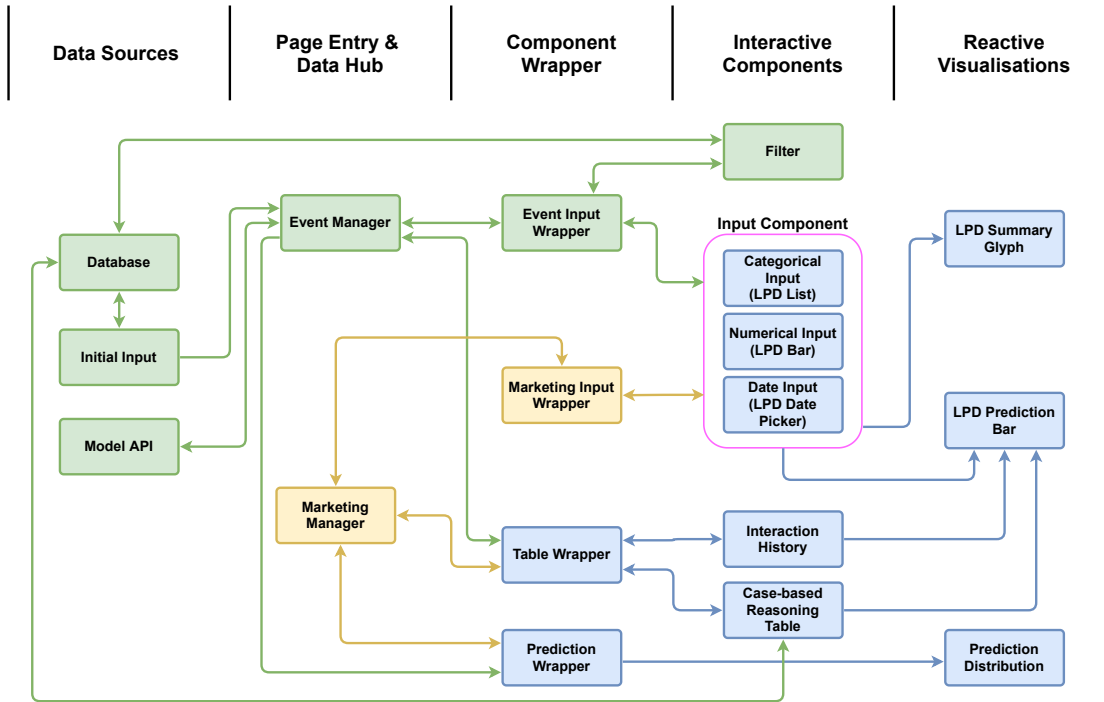Figure 6.1: Overview EVEOS for event organisation.

Figure 6.2: Diagram of final architecture. The direction of the arrows indicates data flow. Blue: general components. Green: event-specific components. Yellow: marketing specific components. Pink: component groups. One or more components of a group can be used at the same time.

EVEOS more model-agnostic since it does not depend on a model that provides multiple prediction values with corresponding probabilities like the LPD prediction distribution does.

From the example illustrated in Figure 6.1, we can extract, among other things, the following information: According to the result view, the organiser would probably sell around 159 tickets for this event. The probability that more tickets will be sold is also rather high. Since the selected venue only has a capacity of 300 people, the prediction is cut off at this value, as discussed in Section 5.3, even though it looks like more tickets could be sold. According to the LPD information about the event name, the name *Mozart Konzerte - Tour* seems to be slightly better than *Mozart Konzerte*. To investigate the detailed prediction for the value *Mozart Konzerte*, one has to select this value or hover over the LPD prediction bar to see the tooltip showing the detailed prediction information as presented in Figure 6.7b. The event date in the example is filtered to show the days *Thursday* to *Sunday* for two months. Checking the FI information for the feature, we see that the currently selected date is suboptimal.

Figure 6.3: Initial Input Modal Event

Fixating the components used for LPD and FI information is also reflected in the diagram of the system architecture presented in Figure 6.2 where we now see that the component groups for reactive visualisations are removed. We now have definite visualisations for these two cases. As detailed in Section 6.4.1, we redefine the term **feature importance (FI)** to **local partial dependence (LPD) summary** because of how it is computed.

The diagram in Figure 6.2 also shows how the database and the model API are integrated into the system. The feature input filters are now connected to the database for features where this is necessary. Also, the case-based reasoning table communicates with the database to fetch ground truth data similar to the current input. To get an initial selection of feature values, we propose the use of an **initial input modal**. The necessity of this modal was presented to us in the previous evaluation by a domain expert. As shown in Figure 6.3, it is a simple form that allows users to select values which they can continue to work with while creating an event. This modal is also connected to the database so that users can select available values for the features genre, venue and artist. Once a user has selected their initial inputs, the data is passed to the manager component and formatted to be the initial state of the **main data object**.

```
Main Data Object          Request Data
{
 price:{
  value:5,
  min:0,
  max:100
 },
 artists:{                 {
  [id:1,                    numberOfPredictions: 53,
   label:Queen,             data:{
   popularity: 0.95,         ticket_price:      [5,   5,   0,   2,   ...,100],
   bookings: 756,            artist_bookings:   [756, 683, 756, 756, ...,756],
   selected: true           artist_popularity:[0.95,0.85,0.95,0.95,...,0.95]
  ],                        }
  [id:2,                   }
   label:ACDC,
   popularity: 0.85,
   bookings: 683,
   selected: false
  ]
 }
}
```

Figure 6.4: Data-mapping of the main data object to structure expected by model API using dummy data. Orange values represent the current input selection. Green values represent LPD input data.

6.2 Model API

As described in Section 3.4, the model API provides a single endpoint accepting POST requests containing the data needed to make one or more predictions. The EVEOS calls this API at two distinct events, when an input is changed by the users to compute all LPD information and when the user fetches similar events for the case-based reasoning table to get the predictions for these events. The API expects the data to be formatted as presented in Figure 3.7, with each model input feature being represented by an array with index i of all arrays being the combined input for prediction i. Since the data structure of the main data object in the interface is optimised for front-end data handling, we need to perform a pre-processing step to format the data correctly.

In Figure 6.4, the result of this data-mapping for getting LPD information after an **input change** is presented. The first step in this mapping process is to set the currently selected input values, which are marked in orange in Figure 6.4, to index $i = 0$ in the corresponding arrays. Then, we need to prepare model inputs to compute the LPD information for all other samples that are currently available in the diverse input components. This means that for each value v that is visible but not selected, we set v as the value of the corresponding model input feature and use the selected values for all other model input features. In the example in Figure 6.4 at $i = 1$, the input for artists with $id = 2$ is set. We add the corresponding values to the input features *artist_bookings* and *artist_popularity* and use the currently selected *ticket_price* to complete the model

Prediction Result   **Visualisation Data**

```
{
 graphData:[
  {x:10,y:0.01},{x:12,y:0.04},{x:15,y:0.1},
  {x:20,y:0.15},{x:25,y:0.25},{x:30,y:0.3},
  {x:35,y:0.1}, {x:38,y:0.04},{x:40,y:0.01},
 ],
 maxPredictionValue: 30,
 maxPredictionProbability: 0.3
}
```

```
{
 predictions:[10,12,15,20,25,30,35,38,40],
 densities:[0.01,0.04,0.1,0.15,0.25,0.3,0.1,0.4,0.01]
}
```

Figure 6.5: Data-mapping of the prediction result of a single prediction to the format used in the visualisations using dummy data.

input data. As discussed in Section 5.4.3, to get the LPD predictions of the *price*, which is a numerical feature, we take 50 equidistant samples between its minimum and maximum value. The indices $2 \leqq i \leqq 53$ of the *ticket_price* array are filled with these samples while the artist feature values are chosen from the selected artist with $id = 1$. In this small example, we end up with a matrix representing the input for 53 predictions that need to be made. For each feature in the main data object, we save the indices corresponding to its LPD predictions so we can then correctly match the prediction results.

When the prediction result is returned from the model API, we first need to format it so it can be used in the visualisations. As shown in Figure 6.5, we extract the value with the highest probability. The result values are grouped in an array of x and y values, which is the format expected by D3.js [4] graphs. Next, the result of the prediction with $i = 0$ is saved as the main prediction result which is then shown in the result view of the EVEOS. The other results are then added to their corresponding feature values using the previously saved indices. For values that are marked as *selected*, we do not generate separate LPD predictions because this would result in the same input as the one with $i = 0$. Therefore, these values get assigned the main prediction result as their LPD prediction result. When assigning a prediction to a value, we always save the prediction value with the highest probability (PP) separately, so we can use it for the LPD prediction bar encoding.

The pre and post-processing to get predictions for the **case-based reasoning** table are similar but simpler because we do not need to compute LPD predictions. Once similar events are fetched from the database, the manager component is triggered to send a request to get their predictions. To prepare the request data, we map the feature values to model input values for each event. When receiving the results of the predictions, we assign them to the corresponding events and mirroring what we described before, we save the prediction value with the highest PP from each result so we can show the LPD prediction bar for each event.

6.3 Result View

During the evaluation of the result view presented in Section 5.7, the most common feedback of the domain experts was that they would like to have a more **detailed legend** to make the main prediction graph easier to interpret. As shown in Figure 6.6, we propose a legend that presents the three prediction values of the point with the highest PP: number of sold tickets NT, occupancy rate OR and probability PP. Additionally, it displays the maximum value of the x-axis with a label that gives context about its semantics. In our case, the x-axis range is defined by the capacity of the currently selected venue.

In the evaluation interviews, we discussed the usage of single-hue and multi-hue **colour scales** with the users. Most of them were able to interpret the single-hue scale faster at first. The interpretation of the multi-hue scale was not obvious to some users when first seeing it. When working with it a little longer, they realised that it helps them evaluate the LPD information more precisely since it provides four distinct hues that allow them to categorise the result. An example for that is shown in Figure 6.7a where three LPD predications bars are shown. Using the hue, we can quickly categorise the quality of the prediction value. We chose to use the multi-hue scale because even though it might be harder to interpret initially, the benefits in long term usage outweigh this downside. The details of the chosen colour scale are described in more detail in Section 5.3.

During the development of this final prototype, we realised that even though the domain experts seem to understand the D3.js [3] **curve interpolation** that we use on the prediction points, it distorts the result and is mathematically incorrect. We used *curveMonotoneX* which produces a cubic spline that preserves monotonicity in y, assuming monotonicity in x. This shows a distribution, but the area under the curve does not sum up to 100%. That is because the probabilities of the prediction values already sum up to 100%, so adding values in between these points automatically adds up to more than that. We also realised that we do not give any information about prediction values that do not have the highest probability, as shown in Figure 6.6a. We therefore present alternative solutions in Figure 6.6 and investigate how they influence the users understanding of the prediction (see Section 6.7).

First, we add **markers** in the form of circles for all prediction values as shown in Figure 6.6b. In doing so, we realised that there seems to be a miscalculation happening in the ML model we employ because the probabilities of the prediction values sometimes exceed 100% which should not be the case. We hypothesise that adding the markers makes it more evident that the prediction provides multiple values and the curve visualises an interpolation between those points.

We also propose two more **curve interpolations**. In Figure 6.6c, we use *curveLinear*, which creates a linear interpolation between the points. This solution is still mathematically incorrect for the same reasons as the *curveMonotoneX* but we think it could be easier to understand because it is just a straight line. In our opinion, this does not imply as strongly that the values between the points are computed using sophisticated methods,

(a) *curveMonotoneX*

(b) *curveMonotoneX* with value markers

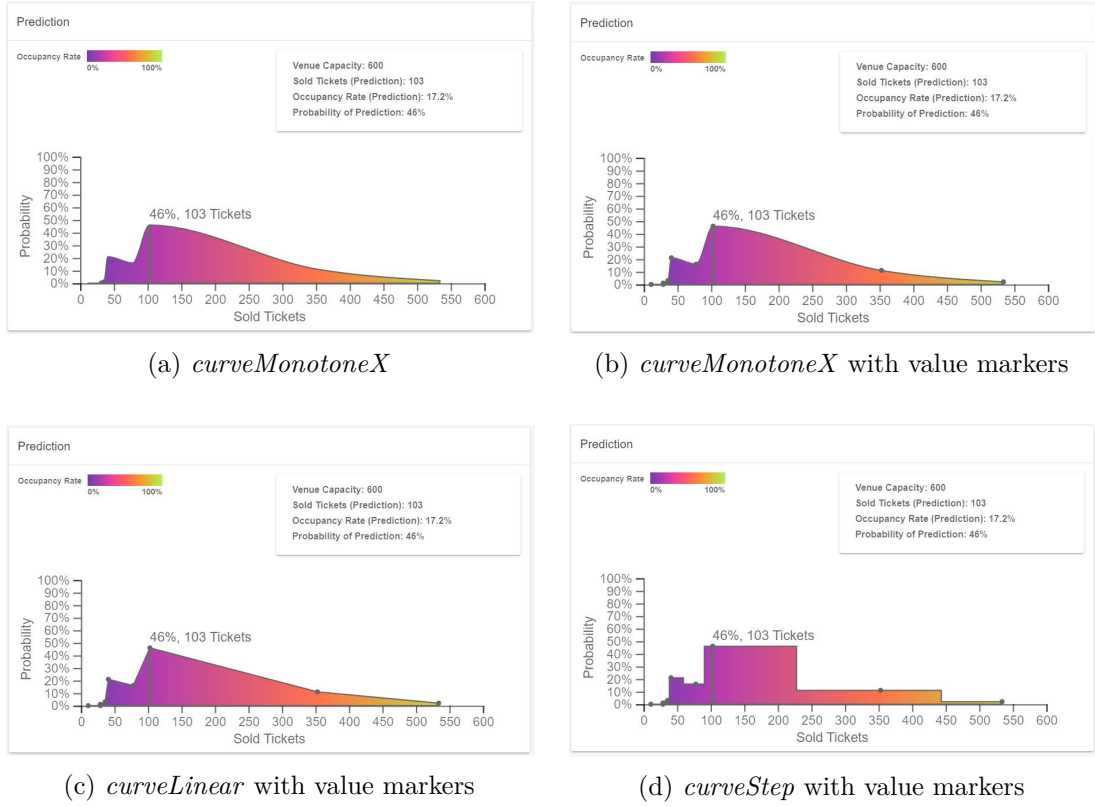(c) *curveLinear* with value markers

(d) *curveStep* with value markers

Figure 6.6: Main prediction result views using different curve interpolations of D3.js [3].

suggesting that they are important. It rather indicates that there are some points with given values and the curve just connects them. The curve that is most consistent with the prediction result is the *curveStep* shown in Figure 6.6d. It is a step-curve with the steps centred between two corresponding points. It shows intervals between the points with constant probability in each interval. These probabilities sum up to 100% which allows users to interpret this curve in a mathematically correct way. For this reason, we think that this version should be used in our system but are keen to see how the users interpret it in comparison to the other proposed solutions.

6.4   Input View

The overall structure of the input view is the same as presented in Section 5.4. We only change the design of the buttons to open and close all input component cards using glyphs indicating their functionality as shown in the top left corner of Figure 6.1.

Talking with domain experts about the distinct input components of the input view, we received the feedback that for some features there is not enough visible information about the feature value. For the venue feature, for example, there is implicit information

(a) Tooltip input value. NT as bar value.

(b) Tooltip LPD. OR as bar value.

Figure 6.7: Tooltips and comparison between using NT or OR as primary value in the LPD prediction bar.

about the venues location and its capacity that is not shown in the input component. Therefore, as shown in Figure 6.7a, we now display this information as a tooltip when hovering over the corresponding value. Extending this idea, we also add a tooltip for the LPD prediction bar as presented in Figure 6.7b. It contains the same information as the legend of the result view. We think that this is an important addition to this encoding because it provides information about the probability of the prediction, which was missing in this visualisation before.

During the evaluation, a domain expert stated that they would rather like to work with the value of the occupancy rate instead of the number of sold tickets when comparing LPD information. As shown in Figure 6.7, we implement both versions. We think that using the OR as the text value is redundant information because it is already given by the colour and length of the bar. The number of sold tickets on the other hand cannot be seen without the text value. We think that using NT is the better approach for these

reasons and if the occupancy rate needs to be investigated, this is now possible with the provided tooltips. We are curious to see what the domain experts think about this in the evaluation.

6.4.1   Local Partial Dependence Summary

As mentioned in Section 6.1, we redefine the term **feature importance (FI)** to be **local partial dependence (LPD) summary** for the following reasons. To complete the functionality of the final prototype, we have to decide how to compute the value for the FI graphs. The goal of this value is to indicate to the users how good the quality, in terms of the resulting prediction, of the currently selected value is.

First, we thought of using the approach presented by Strumbelj and Kononenko [71]. They compute **local feature importance**, by comparing the prediction results of several random samples and using the selected value of the investigated feature as described in Section 2.3. When implementing the algorithm they propose for this, we had to realise that it is computationally slow when using a reasonable amount of samples, taking up to 10 seconds for computing the FI value for a single feature when using 500 random samples. Additionally, when employing this method, the FI value depends on the random samples, which are not visible to the users. We think that the value changing for reasons that cannot be observed by users is highly confusing.

To avoid the mentioned problems, we propose to only **compare the predictions of the currently visible feature samples**. In Figure 6.8, there are two samples shown for the event name feature. Since the prediction value shown in the LPD prediction graph of the selected sample is higher than that of the other sample, the arrow-glyph shows that out of the visible samples the best one is chosen. Similarly, for the ticket price, the arrow-glyph indicates that the selected value is on the lower end when comparing the LPD values of the samples shown on the LPD bar. Using this approach, we in general compare the LPD information of the visible feature samples and show how good the quality of the currently selected value is. The quality q is computed by getting the best p_b and worst predictions p_w of the current samples and, using these values, mapping the prediction of the selected sample p to the interval from -1 to 1: $q = \frac{p - p_w}{p_b - p_w} \times 2 - 1$.

The name LPD summary better represents the resulting value and therefore we decide to rename it. Since the LPD summary only depends on samples visible to the user and its computation is straightforward, we think that it is easy to interpret. What this approach cannot do is to guide users out of local minima or maxima in regards to feature values. Since we only compare the quality of the currently visible values, we cannot infer if there are values available outside of these that would result in better or worse predictions. This would be possible using Strumbelj and Kononenko's [71] approach because they compare the current selection to a random sample set of the training data.

To further enhance the usability of the LPD summary graphs, we change their **legend**. As shown in Figure 6.8 and suggested by users during the last round of interviews, we

Figure 6.8: Arrow-glyph legend for LPD summary encoding.

use annotated arrow glyphs to show how this encoding behaves. Additionally, we change the legend title to *Current Quality* to better reflect what this value is presenting.

6.4.2 Numerical Features

Evaluating input components of numerical features, the domain experts were satisfied with the presented layout and functionality. One of the users mentioned that additionally to the proposed LPD bar it would be supportive if the currently selected value is shown separately together with a definition of its semantics. As shown in Figure 6.8, we add a read-only text field to the input component that provides the requested information.

6.4.3 Date Features

For selecting dates and comparing their LPD information, we propose a colour coded date picker in Section 5.4.4. The date picker shows a single month with the possibility to switch between months. During the evaluation of this component, a user suggested

Figure 6.9: Filterable Date Input.

showing all months that contain selected samples juxtapose. They stated that this would increase the usability of the date picker because it enables them to better compare dates over a time range of multiple months. Since we think that this is a reasonable argument, we realize this idea as shown in Figure 6.9. The only downside of this approach is that when users select a time range in the filter that spans multiple months, this view takes up a lot of space and it becomes difficult to compare dates that are far from each other.

Using the filter, users had problems in applying the filter because the corresponding button was not labelled correctly and placed at a position where users were not expecting it. In the final prototype, we separate the filter from the date selection using a horizontal line. We also rename the button to *Apply Filter* and place it at the bottom of the filter section as shown in Figure 6.9. We believe that this new layout removes the previous ambiguities.

(a) Venue

(b) Artist

Figure 6.10: Categorical inputs with filters.

6.4.4   Categorical Features

As mentioned in Section 6.1, we commit to the use of the LPD prediction bar for LPD information. Similar to the result view, we also need to define the maximum value of the bar. Looking back to Section 5.3, we discussed that one cannot sell more tickets than seats are available in the event venue. For the same reason, we decide to scale the LPD prediction bar according to the capacity of the currently selected venue. The length and colour of the bar encode OR, while the displayed number shows NT. A bar that is filled, therefore, suggests that the event will be sold out. In the input component of the venue feature, on the other hand, the bar is scaled according to each venue distinctively. In the example shown in Figure 6.10a, the prediction for the venue *Hofreitschule* is only by two tickets lower than for the venue *Kammerspiele*. But since *Hofreitschule* has a higher capacity, which can be seen when hovering over the sample to enable the tooltip, the difference in OR is higher as shown by the bar length and colour.

Same as for the date feature filter, the domain experts had difficulties applying filters for categorical data. We change the design to be consistent with the new date filter separating the filter from the input fields and renaming the apply button. As shown in Figure 6.10a, we also remove the range slider from the venue capacity filter because users deemed it unnecessary and rather wanted just minimum and maximum fields. Same goes for the popularity filter for the newly added artist feature presented in Figure 6.10b.

Both the venue and the artist filter are connected to the **database** and work in the following way. After a user has set the filter set filter parameters and clicks on the *Apply Filter* button, venues or artists fulfilling the filter requirements are fetched from the database. Since loading this data takes a few seconds, we show a loading animation on the button while waiting for the result as suggested by Nielsen [56]. Once we get the response from the database, the results are added to the available samples in the drop-down selection, and we show the number of found entities in a small text field.

6.5   Interaction History

As discussed in Section 5.5, a new entry is added to the interaction history table whenever a selected value changes in the input view. During the evaluation of the interactive prototype, one of the domain experts pointed out that for them it is hard to see which feature changed for a particular row in the table. The only possibility to get this information in the previous version was to compare all the columns and look for changes. We thought of highlighting the changed value by changing its font style to *bold*, which in our opinion would be the optimal solution.

During the implementation of the final prototype, we have to realise that the Vuetfiy table component [13] that we use does not provide any possibility to manipulate the style of a single column value of a row. Since we have to use this particular table component because of the project structure we are including the EVEOS in, we have to use a different approach. As shown in Figure 6.11, we add a new column named *Change* to the start of each row. In this column, we display the name of the feature, which change triggered the creation of the corresponding row.

Since we connect the prototype to the database, the data we are handling now contains not only the feature values but also additional data like database IDs. When creating a new row in the interaction history, we also have to save this data. On selecting a row, we use this data to restore the feature value with all its needed information. When a row different to the first row is selected and afterwards the input is changed by the user, the change column shows the name of the feature that changed in comparison to the previously selected row.

6.6   Case-based Reasoning Table

The case-based reasoning table shows ground truth data that is similar to the currently selected input parameters. In our case, this allows domain experts to compare the parameters and the sales data of historical events to the one, they are currently organising. Discussing with them if the comparison should be done in a superimposed view or by seeing the data juxtapose to the main result view in the proposed table, the domain experts predominantly stated that the juxtapose view is preferred. As shown in Figure 6.12 we, therefore, keep the proposed table design. To make the context of this component more straightforward, we change its title to *Past Similar Events* as suggested by a user.

We can only display events that were created by the organiser currently working with the system. For data-privacy reasons, it is not allowed to show an organiser the sales details of events that someone else hosted. We get similar events of the current organiser by searching for events in the database that have the same genre and are located in the same city as the event defined by the current input values. This is not the most sophisticated approach and we plan to improve the method of searching for similar events in future work (see Section 7.3).

Figure 6.11: Interaction history table

Figure 6.12: Case-based reasoning table

6.7   Evaluation

As suggested by the user-centred design cycle (see Figure 2.4), a final evaluation is performed after implementing the EVEOS. We primarily focus on the parts of the system that changed since the last iteration. Additionally, we discuss with the domain experts if they think that they would use the proposed system if it has added value compared to the systems they use now and how they would define this added value.

Same as in the previous evaluation, we perform roughly structured interviews with four domain experts. The *users 1, 2* and *4* are the same as in the evaluation of the interactive prototype. *User 3* was not available, so we introduce *user 5* who works on publishing events and especially on creating marketing campaigns for events in day-to-day business. With *users 2* and *5* the interviews were conducted eye-to-eye and with *users 1* and *4* via a remote desktop application due to COVID-19 restrictions.

Going through the components of the EVEOS we ask the users to perform a set of tasks. First, we show them the new initial input form and ask them to fill it with data they seem fit. On arriving at the main view of the system we discuss the different curve interpolations for the result view presented in Section 6.3 in combination with the new legend. We continue with the input view, asking users to add feature samples at categorical features and compare their predictions using the LPD prediction bar. Next, we focus on the interaction with the provided feature filters. After the users have selected some samples, we ask them to identify the features that have the potential to be improved using the LPD summary visualisations. Switching to the interaction history table, the domain experts are challenged to find the parameter combination that provides the best prediction. At the case-based reasoning table, the task is to fetch similar events from the database and evaluate the precision of the predictions of the ground truth data. Finally, we talk with the users about any general feedback they have.

Discussing the **initial input** form, the interaction with it was obvious to all users. Only for the input field of the initial event venue, there was some confusion. When showing the form in the German language, the label is *Ort* which led users to enter city names instead of the name of the venue they wanted to select. On clarification, they stated that using a label like *Veranstaltungsstätte* would be more fitting. Additionally, *user 2* added that it could be useful to allow the selection of multiple values for categorical features, so when entering the main view there are already multiple samples there that can be compared.

Because all users except *user 5* were already familiar with the proposed **result view** we started by showing them the curve interpolation, *curveMonotoneX*, they knew from previous interviews. In combination with the new, more detailed legend presented in Figure 6.6, all of them were able to precisely describe what the prediction values with the highest probability PP are. They also correctly made the connection that the *Venue Capacity* in the legend is the reason for the range of the x-axis. *User 5*, who did not see this encoding before, called it self-explanatory.

We then directed their attention to the fact that the curve they see is not mathematically correct and asked them if they can explain why this could be the case. All of them stated that they did not realise this before, indicating to us that their focus was always centred on the highlighted prediction value with the highest PP. *Users 1* and 2 said that they interpreted the curve by assuming that where it is lower, the probability of this value is lower, but did not think that the total probability should be 100%. We then added the value markers to the curve and presented it to the users (see Figure 6.6b). Asking the users what the markers are supposed to show, *users 1, 2* and *5* realised that one of the markers coincides with the highlighted prediction value. From this, they derived that the other markers are additional prediction values that are used to build the curve and that the marked values together sum up to a probability of 100%. *User 4* was not able to draw that conclusion and we had to explain the semantics of the markers to them. All of the users then were unsure how the curve values between the markers should be interpreted. Switching to the view using *curveLinear* as the interpolation method, as shown in Figure 6.6c, this did not change. *Users 1* and *5* stated that having a straight line between the markers indicates to them that it is used just as a connection but still were confused about how to interpret it.

We then presented them with the *curveStep* interpolation as can be seen in Figure 6.6d. Based on their knowledge about the markers, the users were able to conclude that here the height of each plateau corresponds to the probability of its prediction value. *Users 1* and *2* stated that, after being made aware of the problem regarding summing up the probabilities, this version represents the data in the best way since it clarifies that the sum of the plateau heights sums up to 100%. *User 5* added that they would find this version even more understandable if the distinct intervals would be separated by a small gap. *User 4* was unsure which of the presented visualisations they prefer. They stated that on the one hand *curveStep* is mathematically correct and its interpretation is apparent. On the other hand, *curveMonotoneX* seemed more realistic to them because it shows a smooth transition between prediction values. This made more sense to them than having intervals with a constant probability with steps between them. Overall, the users seemed to find *curveStep* to be the most valuable and since it also is the version that is most consistent with the provided result from the ML model, we decided to use it in the final version of the EVEOS as shown in Figure 6.13.

We continued the interviews by switching to the **input view**. The users were asked to add some additional samples for categorical features as well as compare samples for each of the features. They were able to perform this task without problems and stated that especially the newly added tooltip legend for the LPD prediction bar helped them in the comparison. Also, *user 5* who saw the interface for the first time, solved the task correctly without help from our side. The colour encoding in the date picker and the usage of the LPD bar for numerical features was also self-explanatory to them. The changed view of the date picker that shows all selected months juxtapose was appreciated by all users and they stated it is easier to compare dates like this than before where they had to switch between the months.

We then discussed with the users if they prefer having *NT* or *OR* as the value presented as text in the LPD prediction bar. All of them stated that *NT* is the preferred version with *user 4* adding that *NT* is the more important value and having *OR* encoded using the colour of the bar and being shown in the tooltip is enough. *User 1* argued that overall it is most important to maximise the number of sold tickets and not the occupancy rate. *User 5* stated that it would be easier to interpret the bar value if it had its semantics written there as well. They suggest writing, for example, *220 Tickets* instead of just *220*.

During the task of selecting samples, the users also had to work with the provided feature **filters**. For the venue and artist filter, the interaction with the input fields was obvious to the users. As shown in Figure 6.10, we show a textual hint to how many entities were found by the filter. *User 1*, who was the first person we interviewed, needed a few seconds of thinking to then realise where they can select these entities after the filter found them. After that interview, we added another textual hint that states that the found entities are available in the value selection (see Figure 6.13). With this information, the other users did not have this problem anymore and knew where to select the entities after searching for them. *User 4* stated that changing the range slider for selecting venue capacity and now using just minimum and maximum fields, makes the selection more apparent.

Filtering dates was also obvious to the users. Only the day of the week selection was a little confusing to *users 2* and *4*. As shown in Figure 6.9, we use a multi-select drop down where the selected days look similar to buttons. Therefore, the users tried to click on them to select them, which does not work. We change the design of this selection to eliminate this confusion as shown in Figure 6.13.

After having selected values for each feature, we asked the users what the LPD summary glyphs represent. Using the improved legend shown in Figure 6.8, all of them were able to explain that they show the quality of the currently selected values of the features. The users were also able to point out the features that could be locally improved and which ones already have the best value selected.

We continued by switching to the **interaction history** table (see Figure 6.11). There, we first asked the users to point out which feature changed for some given rows. They all were able to give the correct answer, but *users 2* and *5* did not use the *Change* column but just searched for the changed value because they did not see this column. The other users used the column with *user 1* stating that they do not understand why the changed value was not just highlighted somehow. This confirms our hypothesis that the proposed solution is not the best one, but as explained in Section 6.5 we are not able to highlight row values in the table framework we are using. We then asked the users to select the parameter combination from the table that results in the best prediction. Using the LPD prediction bar, this task was performed quickly by all users. *User 5* first sorted the table by the predicted number of sold tickets, which made their performance even quicker.

Finally, the users had to fetch similar events from the database for the **case-based reasoning** table. As shown in Figure 6.12 we provide a button titled *Get Most Similar Events* to perform this task. Interestingly, all users except *user 5* did not realise that this button was supposed to be used to automatically get similar events. They rather wanted to search for events manually trying to use the search field of the table. After explaining the concept of automatically searching for events, they stated that since they usually know what events they have organised in the past and we cannot show them events of other organisers, they still would find a manual selection more useful. Once similar events were selected, the users were able to compare the number of sold tickets with the predictions made for the events. All of them stated that if the prediction and the ground truth coincide, this would greatly increase their trust in the system and animate them to try to improve the feature values even more. On the other hand, if this is generally not the case they would rather use their domain knowledge instead of relying on the predictions. *User 5* pointed out that it is very useful to be able to compare past events not only with the event they are currently creating but also against each other. This way they can get a feeling of which parameter settings worked well in the past and which did not.

After going through all the components of the EVEOS we talked about the thoughts of the domain experts on the system. Asking about the perceived waiting times, they stated that at no point they felt that they were waiting unexpectedly long for system responses. *User 1* added that for the filters they would feel annoyed after waiting for more than 10 to 15 seconds and the predictions after two to three seconds. We asked them if this would be different if while waiting for prediction there were a loading animation which they affirmed. These statements directly coincide with the research of Nielsen [56] discussed in Section 2.2. The users also stated that in general, the interaction with the system was obvious to them with *user 4* stating that they think the system is fast and easy to learn.

Asked about where they see the benefits of using our system we got the following answers from the users. *User 1* stated that, for example, for recurring events that have been organised for some time they already have so much domain knowledge that they would probably not use our system. For new events with some unknown factors, on the other hand, it can be very useful assuming that the ML model provides accurate predictions. *User 2* added that some events have rather low flexibility in their parameter settings with sometimes the ticket price being the only parameter that feasibly can be adjusted. In general, all users stated that they see the biggest benefit of the proposed EVEOS in the opportunity to compare parameter settings against each other, both locally for multiple values of a feature and globally for complete feature value combinations using the interaction history. They also state that being able to get an idea of how many people will attend an event is useful in preparing for it.
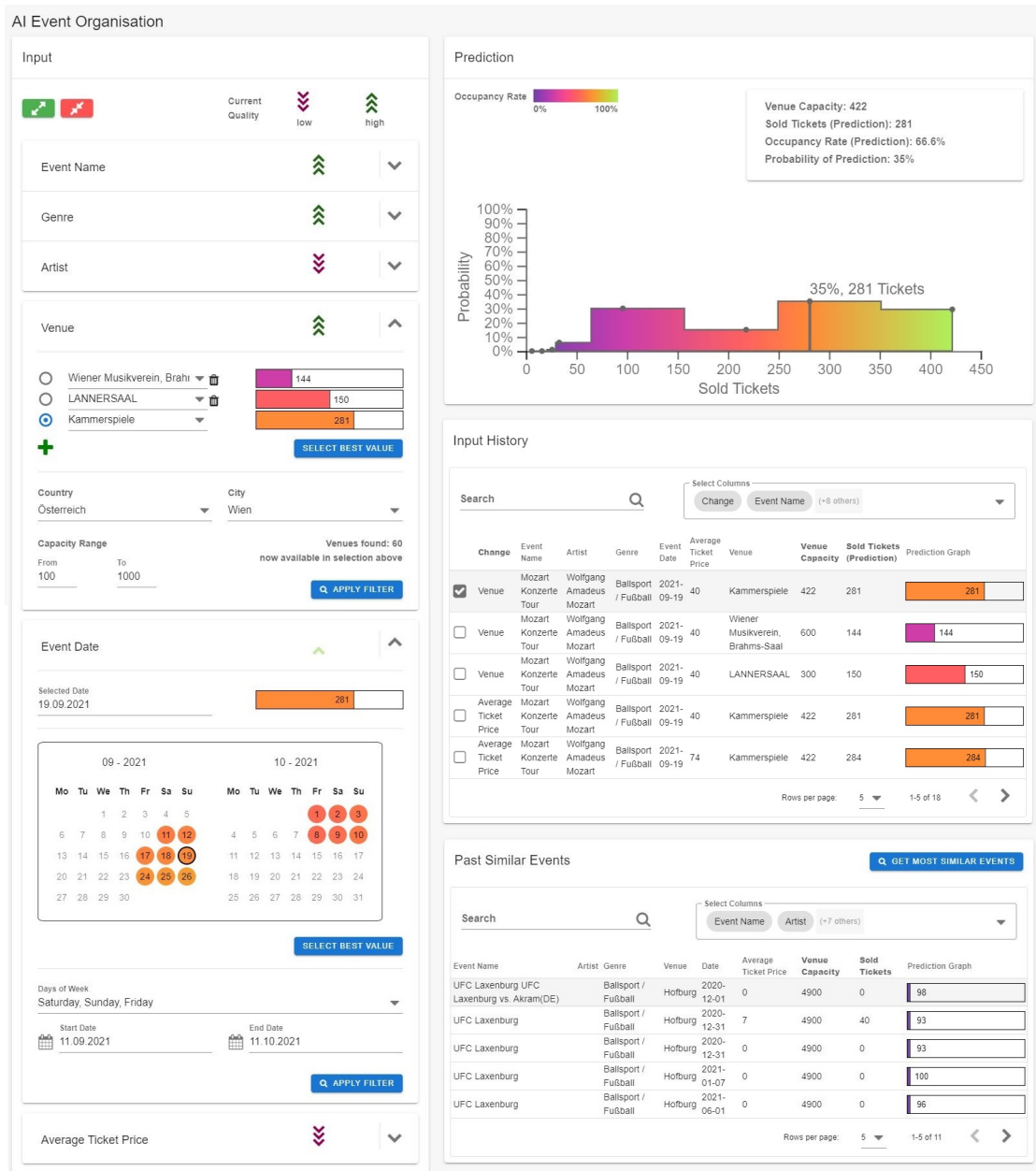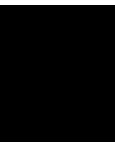
Figure 6.13: Final version of the proposed exploratory visual event-organisation system.

CHAPTER 7

Conclusion

Concluding this thesis, we summarise the results of our work and the lessons learned from it. First, we discuss the feedback we got from domain experts and the conclusions we can draw from employing a user-centred design approach. We then give a synopsis on how the research questions we presented in Chapter 1 were answered. Finally, we go through the current limitations of our work and discuss how it can be enhanced in the future.

7.1 Discussion

The results of three rounds of implementing and evaluating the proposed exploratory visual event-organisation system (EVEOS) provided us with several interesting discoveries regarding the system itself and also regarding the employed user-centred design process. While the general feedback we got from the participating domain experts was positive, we also had to realise that some parts of our work were not developed thoroughly enough.

The most apparent of these is the design of the main result view (see Figure 6.6). Here we proposed using a distribution curve in the early stages of the design process, focussing only on how this distribution can encode all the provided data in the best way possible. We did not consider other approaches thoroughly enough, thinking that we had already found a proper visual encoding. Only when incorporating the real prediction result data during the implementation of the final prototype did we realise that the proposed solution was mathematically incorrect and can lead to wrong interpretations. Since making big changes in the late stages of a project is difficult, we continued to try to improve the concept of showing the prediction in some kind of distribution chart. When discussing the proposed alternatives with the domain experts, they gravitated towards using one of the alternatives but none were completely content with it. If we had realised this oversight sooner, we could have switched to an entirely different, and more fitting approach.

Other intriguing findings while reviewing the performed interviews include that it seems that users are sometimes not interested in being provided with the most detailed information. It is often more important to them that what they are presented with is easy to understand and to read. An example of that is the different LPD encodings we presented in the early stages of the design process. The domain experts predominantly favoured visualisations they were able to compare the fastest, even if they provided less information than more sophisticated visualisations. They were mostly content with the possibility to see the details of a prediction as a tooltip or in the main result view when selecting the corresponding value. We think that this could be related to the nature of the domain of event organisation. In other domains, like medical visualisation, we believe that comparing details could be more important.

Additionally, we found that sometimes users do not like to read explanations or legends. During the evaluations, it happened a few times that users were not able to interpret a visualisation correctly. In these cases, they were so focused on the visualisation that it did not occur to them to search for a guide outside of the immediate perimeters of the visualisation. An example for that would be the FI bars proposed in Section 5.4.1. The corresponding legend was placed on top of the input view. Users were not immediately sure what this visualisation is supposed to show. Instead of looking at the legend, they tried to find cues in the immediate surroundings, which include the feature name and LPD information of feature values. Similarly, one user suggested during the evaluation of the interactive prototype in Section 5.7, that the legend for the colour scale used for encoding LPD information, should be present in every single feature input component. We only show it in the result view and the user did not find it there when working with the LPD information. In general, the feedback we got from the users leads us to the assumption that if additional information, like legends, is needed to correctly interpret a visualisation, it should be placed close to the visualisation.

We also found that as developers it is often hard to put ourselves in the position of domain experts. Many of our hypotheses along the way were correct, but users also disagreed with us, often using arguments that were not considered during the design process. It is important to remember that some concepts or specifics of implementation are only obvious to us as developers because we work with these on a day-to-day basis. The same aspects could be completely obscure to experts in other domains. This is where the user-centred design process provided us with a lot of help.

Even though user-centred design processes provide good guidelines and can help to avoid wrong design choices, they do not guarantee perfect results and are also demanding to perform correctly. We found it especially hard to be unbiased during the rounds of evaluation. When developing numerous alternative solutions to a problem, a developer often has preferences of their own. Sometimes these preferences are not related to the perceived quality of the result, but rather to the implementation. Thus it can be difficult to keep an open mind towards the opinions of the user while evaluating the proposed solutions. Especially when performing interviews, one has to be mindful to not phrase questions in a way that favours the solution that the interviewer prefers.

As developers, we also need to be careful when proposing a feature to users that they did not explicitly request but, in our opinion, can help solve their tasks. In this case, it is important to early on explain the intentions the developers pursue in introducing this feature. Otherwise, it can happen that users do not fully understand the proposed feature and then are not able to give proper feedback on it. This happened in our work with the case-based reasoning table, where only in the last round of evaluations we realised that users would rather search for past similar events on their own instead of using sophisticated methods to fetch them automatically. To avoid this pitfall, we suggest conducting focus group meetings, or other qualitative evaluations, where the developers present their idea and how they think it can beneficial for the users, maybe using a first rudimentary prototype. It is important that there is an open discussion about the proposed feature. The domain experts need to understand the idea behind the feature and the developers need to listen carefully to the provided feedback.

In general, we think that employing a user-centred design process for developing the EVEOS increased the quality of the final version of the system. It also seemed to raise the investment of the domain experts in the project and they were always eager to provide helpful and critical feedback. Because we did not have more than four participants in each round of evaluation, it was difficult to get broad and diverse views on the proposed system. We had to realise that firstly, qualitative evaluations are very time-consuming and secondly, it is hard to find domain experts willing to invest their time into such a process. This goes along with Seldmair et al.'s [67] work, where they describe the difficulties of finding the right collaborators for design studies. Nonetheless, the users provided us with constructive feedback and also criticism that helped us to gradually improve the proposed EVEOS and develop a well rounded final system.

The main advantage the domain experts saw in using the proposed EVEOS over currently used systems lies in the possibility to compare multiple parameter values and value combinations using LPD information visualisations. The final feedback we got for the data-type specific visual encodings and inputs was positive. The provided case-based reasoning was especially intriguing to the domain experts. They reported that it helps them get a new data-driven perspective on their existing domain knowledge. Furthermore, they stated that they could evaluate the quality of the predictions using case-based reasoning information. The domain experts also stated that they see great potential in the proposed system when they want to organise novel events where the existing domain knowledge is low. On the other hand, planning reoccurring events that they already hosted multiple times, they would rather trust their existing knowledge. For such events, they expressed great interest in the possibility to use an adjusted version of the proposed system to optimize their marketing strategies. Since, most of the proposed system components are model-agnostic and can be generalised, providing a system for this use case can easily be done, as shown in Chapter 5.

The accuracy of the predictions for ground truth data was the main factor to the users in deciding the usefulness of the whole system. Since their main concern for usability was the performance of the ML model, which is not the main research part of this thesis, the proposed EVEOS sufficiently satisfied the domain experts.

Except for the visualisation for the main result view, all of the proposed visual encodings were designed to be model-agnostic. We, therefore, see big potential for the developed concepts to be used in a general way. As we have shown by implementing a prototype for creating marketing campaigns, the LPD and LPD summary encodings can be used for data other than event-organisation data with only minor adjustments. We believe that especially the proposed LPD encodings for categorical and date data can serve as best practice examples for other works.

As discussed in Chapter 1, many state-of-the-art approaches to make ML models interpretable are too complex for laypeople in the field of computer science. We believe that employing a user-centred design approach, we managed to develop a system that for the most part is simple enough to be used by domain experts in their corresponding field. As discussed in this Section, for the domain of event organisation, the way to achieve this was to keep the visual encodings as simple as possible, even if this means that detailed information is lost. Using the well-known concept of *details on demand*, we enabled the users to see the lost details in tooltips. We believe that for non-technical domains, our proposed approach of keeping visual encodings simple can work well. Nonetheless, we advise always performing a proper user and task analysis before starting the design phase of a project so the needs of the users are precisely understood. Especially when developers bring new ideas to the table, it is important to have early and open discussions about the proposed ideas with the domain experts.

7.2 Summary

The main goal of this thesis was to develop an **exploratory visual event-organisation system (EVEOS)** on top of an ML model that allows event organisers to find event parameters that optimise their profits. Assuming our user group has no prerequisite knowledge in the field of machine learning, we focussed on creating a system that is highly interpretable and generates trust towards the users. During our extensive research (see Chapter 2) in the field of machine learning interpretability, we found that the concepts of what-if analysis and case-based reasoning are commonly used to solve this task. Both are based on making different input and output combinations of ML model comparable. We found that many state-of-the-art approaches have issues in making their system usable for laypeople in the field of machine learning. Taking a closer look a previous work in the field of HCI, we realised that employing a user-centred design process is often successful in creating systems that have high usability. Following this process, described in Section 1.2, we developed the proposed EVEOS in three iterations, each consisting of design, implementation and evaluation of a prototype. In Chapter 1, we defined a set of user and research specific questions as well as tasks that an interpretable machine learning system should be able to solve. In the following paragraphs, we describe how we addressed these questions and tasks in this thesis.

When working with machine learning models in general, one needs to convey information about prediction results in a comprehensive way (**T1**). The model we are employing tries to predict multiple values for how many tickets will be sold for a given event parameter's input and provides a probability for each of these values. Additionally, we compute the prediction occupancy rate from the capacity of the selected venue and the predicted number of sold tickets, leaving us with a set of multiple triples that need to be visualised. To do this, we proposed an **area chart** that uses a step curve to connect the distinct prediction points (**V1**). The x-axis of the chart represents the number of sold tickets and the y-axis the corresponding probability. The colour of the area beneath the step curve encodes the occupancy rate. To highlight the prediction value with the highest probability (**Q1, Q2**), a vertical line reaching down to the x-axis is added at its position. Additionally, a textual legend presents detailed information about this prediction value. The results of our evaluation revealed that users were able to quickly interpret the values and semantics of the highlighted prediction value. On the other hand, we had to realise that users struggled to describe the meaning of the rest of the presented data, leading to the assumption that the proposed area chart is not suited for this task as well as we thought.

We use the concept of **what-if analysis** to try to explain to users why the model converts the current input to the presented output (**T2**). What-if analysis is based on comparing predictions that result from multiple model inputs. In our research on how this comparison can be done (**V2**), we found several works that use partial dependence to solve this task [76, 66, 80]. Partial dependence is a model-agnostic approach, meaning that the underlying ML model can be changed without affecting the implementation of the comparison. Krause et al. [40] present an approach they call local partial

dependence (LPD) bar where they compare predictions resulting from changing only a single numerical feature value on a colour coded bar, keeping the other values fixed. Based on their approach, we develop LPD encodings for the distinct data types present in the used data set. For categorical data, we propose an LPD prediction bar, showing the prediction values with the highest probability using the bar size and colour to encode OR and a textual cue to show NT. For numerical data, we employ Krause et al.'s [40] LPD bar and propose a colour encoded date-picker for date-based features. To the best of our knowledge, the LPD prediction bar for numerical features and the scented date-picker are novel encodings to show LPD information.

Using these visual encodings, our evaluations showed that users were able to perform fast and precise comparisons between the predictions for multiple values of a feature (**Q3**). This way, users can achieve a better understanding of how the values of a feature influence the prediction. Additionally, we show users the local quality of the currently selected feature value by computing a summary of the LPD information and encoding it as arrow glyphs (**Q4**). Using this visualisation, users were able to detect which features should be changed and which parameters already had values that lead to a good prediction.

We found that if the comparisons provided by the LPD information in the input components coincides with the expectations and domain knowledge of the users, their level of trust in the predictions is positively influenced. Since this is not always the case, we need to employ additional approaches to try to convince users of the accuracy of the used model (**T3**). Researching this topic, we came across the concept of **case-based reasoning** [80, 21]. Case-based reasoning uses ground truth input data and shows a comparison between the prediction that is made and the known result. Employing this concept, we provide the users with historical events that are similar to the current model input in a table view. Using the proposed LPD prediction bar and the known event values, users can validate the accuracy of the model (**V3**). During our evaluations, users stated that if the predictions and the known values of historical events in general correspond, their level of trust in the model increases. If this is not the case, their trust decreases, which is reasonable and correct since this would mean that the model is not able to perform correct predictions.

By doing three rounds of qualitative evaluation at different development stages, we tried to design a system with high usability. In general, the results of these evaluation rounds were positive. Especially in the early development stages, the feedback provided by the domain experts guided us towards better results. In the final evaluation round, users were satisfied with the proposed EVEOS stating that it provides them with opportunities they do not have in their currently used systems.

7.3 Limitations and Future Work

During the evaluation of the proposed EVEOS, the interviewed domain experts pointed out a few areas where improvements can be made. Additionally, they made suggestions on how to increase the generality of the system.

When discussing multiple alternative approaches on how information can be encoded with users, they never voted unanimously for the same approach. In general, there were mostly two groups of users. One group preferred visualisations that provide a rich level of detail and were often more complex to interpret. The other group favoured visualisations that were fast and easy to interpret but had a reduced level of detail. In our work, we decided to follow the preferences of the second group because they represented the majority of users. To satisfy both groups, one could provide the possibility to switch between a *basic* and an *expert* view of the system, providing both types of visualisations.

As discussed in Section 7.1, the final version of the proposed area chart for the main result view did not completely satisfy the users. Even though they were able to interpret the prediction value with the highest probability using the provided legend, the rest of the chart was not understood completely. An alternative visual encoding that could be used for this graph would be a bar chart. Each bar could represent an interval of values that share the same probability, with the height of all bars summing up to 100%. The width of the bar could be dependent on the interval size. This way, it would be more readable than several value ranges are predicted instead of suggesting a continuous probability distribution as we did in our proposed solution.

As discussed in Section 4.1, there are a lot of input parameters for the ML model that directly depend on the user input for their parent parameters. In Figure 6.7a, we show information about these dependent parameters using tooltips. A feature that could enhance the interpretability of the LPD information of a parent parameter could be to show the influence, the dependent parameters have on the prediction. An example for this would be to compute what influence a date being on the weekend has on the prediction. To do this, one could employ the feature importance approach by Strumbelj and Kononenko [71]. The feature importance value could be added to the proposed tooltips.

In Section 5.4.2, we present a *Select Best Value* button for categorical feature input components. It selects the best of the currently visible samples. When implementing this functionality, we did not consider that for features that are connected to a filter, there usually are more available options than the currently visible ones. In this case, we think it would be helpful to the users to be able to select the best option that fits the current filter input instead of just the best visible option. This would also help users in escaping the local maximum of the currently visible options, especially when the filter returns a large number of options.

The domain experts also pointed out that they would like to be able to manipulate the selection of similar events in the course of the case-based reasoning analysis. Right

now, they are only able to press a button that automatically fetches events from the database according to a pre-set filter. Future work to improve the handling of this process could include providing a filter where users can select criteria to choose similar events. Alternatively, a simple search by event name could be implemented to search for specific events that the users would want to evaluate. Additionally, we think it could be helpful to the users to show a measurement of similarity between the fetched events and the current input. According to this measurement, similar events could also be initially sorted when they are fetched from the database.

Since the work of this thesis was focussed primarily on the front-end implementation of the EVEOS, there are some areas of the back-end and ML model implementation that can be further improved. One additional feature that was suggested by domain experts was that the model should consider how many tickets were already sold in their prediction when analysing events where the ticket sales already started.

The mentioned suggestions and current limitations show that is still room for improvement in our work. Nonetheless, the results of our evaluations show that we successfully provided our target user group with a system that provides them with possibilities they did not have before. The proposed exploratory visual event-organisation system could be a further step in introducing machine learning into various other domains. Evaluations with experts from these domains would be necessary to confirm this hypothesis.

List of Figures

List of Tables

Glossary

API application programming interface. 6, 15, 16, 25, 32–34, 53, 64, 65, 71, 74–76, 101

CBR case-based reasoning. 65, 66

DNN deep neural network. 6, 25, 32, 35, 37

EVEOS exploratory visual event organisation system. 1, 3–6, 11, 12, 25, 35, 37, 49, 51, 53, 66, 70–73, 75, 76, 84, 86, 87, 89, 91, 93–98, 100

FI feature importance or influence. 41, 45–48, 51, 52, 54, 58, 59, 66, 68–71, 73, 74, 80, 92

HCI human-computer interaction. 13, 95

IC input component. 51–54

ICE individual conditional expectation. 22, 23, 99

ISO international organization for standardization. 7

k-NN k-nearest neighbours. 24

LPD local partial dependence. 19, 20, 36, 37, 41–48, 51, 52, 54, 57–68, 70, 71, 73–77, 79–81, 83, 86–88, 92–94, 96, 97, 100

MDS multidimensional scaling. 28–31, 99

ML machine learning. xi, xiii, 1–4, 6–8, 11–13, 17, 18, 20, 26, 27, 31, 32, 34, 35, 37, 39, 41, 46, 52–54, 58, 62, 65, 71, 77, 87, 89, 94, 95, 97–99, 101

NN neural network. 3, 22, 24

NT predicted number of sold tickets. 37, 39, 41, 43, 47, 51, 55, 58, 61, 77, 79, 83, 88, 96, 100

OR predicted venue occupancy rate. 37, 39, 41, 43, 44, 47, 51, 55–58, 61, 62, 67, 77, 79, 83, 88, 96, 100

PC personal computer. 37

PCA principal component analysis. 28, 29, 99

PD partial dependence. 17, 18, 22, 23, 99

PP probability of predicted value. 37, 39, 41, 43, 44, 46, 47, 51, 55, 57, 61, 67, 76, 77, 86, 87

RAM random access memory. 30

SRT system response time. 14, 99

t-SNE t-distributed stochastic neighbour embedding. 28–31, 99

UI user interface. 7, 15

Bibliography

[1] Absolut Ticket GmbH. `https://www.absolut-ticket.at/`. Accessed: 2021-08-22.

[2] Broj 42 gitlab group: Vue Customizable Datepicker. `https://gitlab.com/broj42/vue-customizable-datepicker`. Accessed: 2021-08-22.

[3] D3 Curves. `https://github.com/d3/d3-shape#curves`. Accessed: 2021-08-22.

[4] D3.js. `https://d3js.org/`. Accessed: 2020-10-26.

[5] Flask. `https://flask.palletsprojects.com/en/2.0.x/`. Accessed: 2021-08-22.

[6] Keras. `https://keras.io/`. Accessed: 2021-08-22.

[7] Numpy. `https://numpy.org/`. Accessed: 2021-08-22.

[8] Plotly. `https://plotly.com/`. Accessed: 2021-08-22.

[9] Symfony. `https://symfony.com/`. Accessed: 2020-10-26.

[10] Vue.js. `https://vuejs.org/`. Accessed: 2021-08-22.

[11] Vue.js: Reactivity in Depth. `https://vuejs.org/v2/guide/reactivity.html`. Accessed: 2021-08-22.

[12] Vue.js: Slots. `https://vuejs.org/v2/guide/components-slots.html`. Accessed: 2021-08-22.

[13] Vuetify: Data tables. `https://vuetifyjs.com/en/components/data-tables/`. Accessed: 2021-08-22.

[14] H. Abdi and L. J. Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[15] J. L. Bennett. Managing to meet usability requirements: establishing and meeting software development goals. *Visual Display Terminals, Prentice-Hall*, pages 161–184, 1984.

[16] D. Benyon, P. Turner, and S. Turner. *Designing interactive systems: People, activities, contexts, technologies.* Pearson Education, 2005.

[17] H. Bergman, A. Brinkman, and H. S. Koelega. System response time and problem solving behavior. In *Proceedings of the Human Factors Society Annual Meeting*, volume 25, pages 749–753. SAGE Publications Sage CA: Los Angeles, CA, 1981.

[18] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications.* Springer Science & Business Media, 2005.

[19] D. Borland and R. M. Taylor II. Rainbow color map (still) considered harmful. *IEEE Computer Architecture Letters*, 27(02):14–17, 2007.

[20] J. M. Carroll and M. B. Rosson. Usability specifications as tool in iterative development. Technical report, IBM Thomas J Watson Research Center Yorktown Heights NY, 1984.

[21] R. Caruana, H. Kangarloo, J. D. Dionisio, U. Sinha, and D. Johnson. Case-based explanation of non-case-based learning methods. *Proceedings / AMIA ... Annual Symposium. AMIA Symposium*, pages 212–215, 1999.

[22] A. Divyanshu. Gower's Distance. `https://medium.com/analytics-vidhya/gowers-distance-899f9c4bd553`. Accessed: 2021-08-22.

[23] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[24] O. J. Espinosa, C. Hendrickson, and J. Garrett. Domain analysis: a technique to design a user-centered visualization framework. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99)*, pages 44–52. IEEE, 1999.

[25] D. Fallman. Design-oriented human-computer interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 225–232, 2003.

[26] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[27] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[28] T. Goodman and R. Spence. The effect of system response time on interactive computer aided problem solving. *ACM SIGGRAPH Computer Graphics*, 12(3):100–104, 1978.

[29] J. Gower. Illustration of a new technique for comparing different distance analyses. In *American Journal of Physical Anthropology*, volume 35, page 280. Wiley-Liss div John Wiley & Sons Inc, 605 Third Ave, New York, NY 10158-0012, 1971.

106

[30] M. Graham, J. Kennedy, and D. Benyon. Towards a methodology for developing visualizations. *International Journal of Human-Computer Studies*, 53(5):789–807, 2000.

[31] M. Grossberg, R. A. Wiesen, and D. B. Yntema. An experiment on problem solving with delayed computer responses. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):219–222, 1976.

[32] F. Hohman and M. Kahng. Visual Analytics in Deep Learning : An Interrogative Survey for the Next Frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2674–2693, 2019.

[33] ISO. Iso 13407: Human-centered design processes for interactive systems, 1998.

[34] ISO. Iso 9241: Ergonomic requirements for office work with visual display terminals (vdts) - part 11: Guidance on usability, 1998.

[35] J. Johnson and J. Jeff. *GUI bloopers: don'ts and do's for software developers and Web designers.* Morgan Kaufmann, 2000.

[36] B. Kim. *Interactive and interpretable machine learning models for human machine collaboration.* PhD thesis, Massachusetts Institute of Technology, 2015.

[37] W. Kim, S. Xiong, and Z. Liang. Effect of loading symbol of online video on perception of waiting time. *International Journal of Human–Computer Interaction*, 33(12):1001–1009, 2017.

[38] G. Kindlmann, E. Reinhard, and S. Creem. Face-based luminance matching for perceptual colormap generation. In *IEEE Visualization, 2002. VIS 2002.*, pages 299–306. IEEE, 2002.

[39] P. Kovesi. Good colour maps: How to design them. *arXiv preprint arXiv:1509.03700*, 2015.

[40] J. Krause, A. Perer, and K. Ng. Interacting with predictions: Visual inspection of black-box machine learning models. *Conference on Human Factors in Computing Systems - Proceedings*, pages 5686–5697, 2016.

[41] O. Kulyk, R. Kosara, J. Urquiza, and I. Wassink. *Human-Centered Aspects*, pages 13–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[42] S. Lauesen. *User interface design: a software engineering perspective.* Pearson Education, 2005.

[43] A. Light and P. J. Bartlein. The end of the rainbow? color schemes for improved data graphics. *Eos, Transactions American Geophysical Union*, 85(40):385–391, 2004.

[44] B. Y. Lim. *Improving understanding and trust with intelligibility in context-aware applications.* PhD thesis, Carnegie Mellon University, 2012.

[45] Z. C. Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

[46] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):1–28, 2018.

[47] D. J. Mayhew. The usability engineering lifecycle. In *CHI'99 Extended Abstracts on Human Factors in Computing Systems*, pages 147–148, 1999.

[48] S. Miksch and W. Aigner. A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, 38:286–290, 2014.

[49] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.

[50] D. L. Morgan. *Focus groups as qualitative research*, volume 16. Sage publications, 1996.

[51] T. Munzner. A nested model for visualization design and validation. *IEEE transactions on visualization and computer graphics*, 15(6):921–928, 2009.

[52] T. Munzner. *Visualization analysis and design.* CRC press, 2014.

[53] B. A. Myers. The importance of percent-done progress indicators for computer-human interfaces. *ACM SIGCHI Bulletin*, 16(4):11–17, 1985.

[54] M. Niccoli. Perceptual rainbow palette – the method. `https://mycarta.wordpress.com/2013/02/21/perceptual-rainbow-palette-the-method/`. Accessed: 2021-08-22.

[55] M. Niccoli. The rainbow is dead. . . long live the rainbow! – The rainbow is dead. . . long live the rainbow! – Perceptual palettes, part 3. `https://mycarta.wordpress.com/2012/10/06/the-rainbow-is-deadlong-live-the-rainbow-part-3/`. Accessed: 2021-08-22.

[56] J. Nielsen. Response times: The 3 important limits. `https://www.nngroup.com/articles/response-times-3-important-limits/`, 1993. Accessed: 2021-08-19.

[57] J. Nielsen. *Usability engineering.* Morgan Kaufmann, 1994.

[58] J. Nielsen. Website response times. `https://www.nngroup.com/articles/website-response-times/`, 2010. Accessed: 2021-08-19.

108

[59] S. Pajer, M. Streit, T. Torsney-Weir, F. Spechtenhauser, T. Möller, and H. Piringer. WeightLifter: Visual Weight Space Exploration for Multi-Criteria Decision Making. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):611–620, 2017.

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

[61] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[62] W. Quesenbery. The five dimensions of usability. In *Content and complexity*, pages 93–114. Routledge, 2014.

[63] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[64] M. T. Ribeiro, S. Singh, and C. Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[65] B. E. Rogowitz and L. A. Treinish. Data visualization: the end of the rainbow. *IEEE spectrum*, 35(12):52–59, 1998.

[66] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Moller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161–2170, 2014.

[67] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE transactions on visualization and computer graphics*, 18(12):2431–2440, 2012.

[68] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[69] B. Shneiderman, C. Plaisant, M. S. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction.* Pearson, 2016.

[70] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha. *User interface design and evaluation.* Elsevier, 2005.

[71] E. Štrumbelj and I. Kononenko. A general method for visualizing and explaining black-box regression models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6594 LNCS(PART 2):21–30, 2011.

[72] M. Tory and T. Moller. Rethinking visualization: A high-level taxonomy. In *IEEE symposium on information visualization*, pages 151–158. IEEE, 2004.

[73] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[74] S. Weinberg. Learning effectiveness: The impact of response time. In *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems.(Part-II): Human Interface and the User Interface-Volume 1981*, page 140, 1981.

[75] S. M. Weiss, G. Boggs, M. Lehto, S. Shodja, and D. J. Martin. Computer system response time and psychophysiological stress ii. In *Proceedings of the Human Factors Society Annual Meeting*, volume 26, pages 698–702. SAGE Publications Sage CA: Los Angeles, CA, 1982.

[76] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):56–65, 2020.

[77] W. Willett, J. Heer, and M. Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, 2007.

[78] E. Winter. The shapley value. *Handbook of game theory with economic applications*, 3:2025–2054, 2002.

[79] J. Zhang, K. A. Johnson, J. T. Malin, and J. W. Smith. Human-centered information visualization. In *International workshop on dynamic visualizations and learning, Tubingen, Germany*. Citeseer, 2002.

[80] Q. Zhao and T. Hastie. Causal Interpretations of Black-Box Models. *Journal of business & economic statistics: a publication of the American Statistical Association: Duplicate, marked for deletion*, 2019.