Data Science and Artificial Intelligence

Model deployment of a predictive machine learning model for clinical decision support

Lisanne Wallaard

Supervisors:
Prof.dr. M.R. Spruit & Dr. M.R. Haas

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

July 5, 2023

Abstract

An increasing amount of available healthcare data has led to research creating validated predictive healthcare models. However, when the research is completed, little attention is given to deploying these models. This is unfortunate as the models could be used as clinical decision support (CDS) tools for doctors and patients. This research aims to reduce the gap between developing and implementing predictive machine learning models. To do so, the study implements several predictive healthcare models in Python and R as a web application using the Streamlit library in Python. From these specific applications, a general coding framework arises for implementing medical machine learning models as web apps on a local machine. In healthcare research, models are often saved in a remote environment that is not accessible to general practitioners or patients due to privacy issues with the data. To overcome this issue, the framework only requires the model's parameters (through an export file of a trained model) instead of the data itself. The doctors and patients should enter any missing variables in the interactive web app as input to get a prediction.

Contents

1 Introduction

1.1 The Situation

In recent years, more and more data have been collected about individuals. Also, the availability of healthcare data has increased drastically (Jung et al., 2021). This enables researchers to train predictive models on tasks related to medical issues, such as determining a certain level of risk for cardiovascular diseases. For example, these models could be a valuable support system for doctors in general practice when explaining health risks to patients. If used by doctors, this is an example of non-physical Artificial Intelligence (AI) in healthcare (Ellahham et al., 2020).

In Figure 1, a data mining model is shown, called CRISP-DM, which organisations can use to guide their data mining process (Shearer, 2000). Spruit and de Vries (2021) developed the AutoCrisp tool, which should automate the data understanding, data preparation and modelling phase of the CRISP-DM model. This tool is based on R shiny, which is a package in R that enables building web applications to display the data[1]. It is intended to help clinicians without advanced coding skills to implement these phases. However, little attention has been given yet to automating the deployment phase.

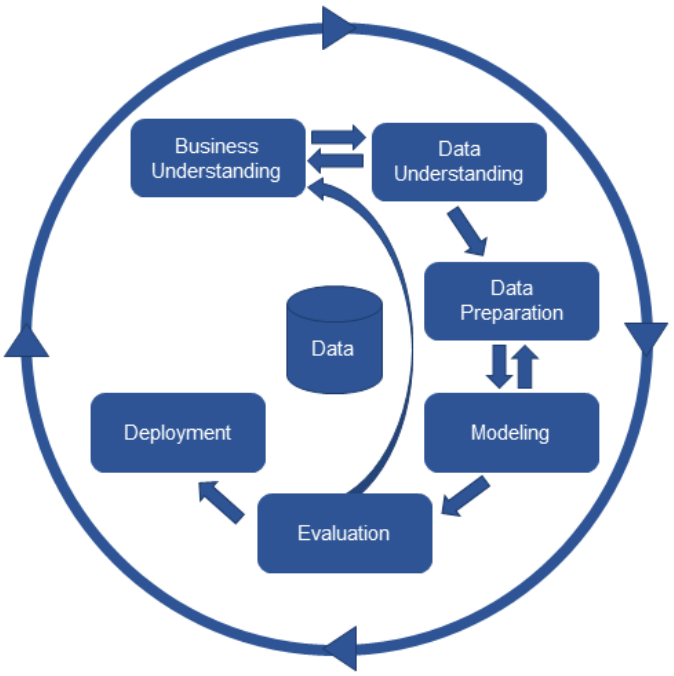


Figure 1: Phases of the CRISP-DM Model (Dsouza, 2018)

An interface for the model should be created in the deployment phase of the CRISP-DM model. Nevertheless, the phase is not always executed when researchers stop as they have their results. This

[1]Source: https://shiny.posit.co/

might be due to the fact that implementing a model can be difficult and therefore takes time. The time they may not be willing to spend as they already have everything needed for their paper. In reality, only 13% of all the models worldwide are implemented (Davenport & Malone, 2021). This is a point of criticism because the research results are not returned to clinical practice (Baier et al., 2019).

As this sixth phase is not often executed, the phase is also quite poorly defined by Shearer (2000). He mentioned the following key steps: ‘plan deployment‘, ‘plan monitoring and maintenance‘, ‘the production of the final report‘ and ‘review of the project‘. From which ‘plan deployment‘ is only described as the following:

> ”In order to deploy the data mining result(s) into the business, this task takes the evaluation results and develops a strategy for deployment.”

This needs to be done better. Spruit (2022) talked about the deployment phase in healthcare in a bit more detail. He mentioned that it is common practice for researchers to post their source code so that the end user can reconstruct the model. However, this is often not feasible for a healthcare professional. Therefore, he discussed possible model publications by mentioning exporting the model in a standard exchange format or exporting a user application of the model. However, he does not describe how this can be achieved, nor does he release any code.

Anyway, implementing predictive machine learning models in general practice is often not easy. The models are commonly built and trained in a different environment than where they are used in practice. As a use case for a remote environment, Statistics Netherlands (CBS) is explained in the box below.

CBS

CBS is the national statistical office of the Netherlands[a]. CBS provides independent and reliable statistical information about the Dutch society. Therefore, CBS has many data, including sensitive data. All this information is kept in a secure Remote Access (RA) environment. Researchers can gain access to this environment to conduct a study. If the researcher wants to export any information outside this environment, there are strict regulations. They need to put the desired output in an export folder, and CBS will perform a check such that no data will leave the environment[b].

[a]Source: https://www.cbs.nl/en-gb/about-us/organisation

[b]Source: https://www.cbs.nl/en-gb/our-services/customised-services-microdata/microdata-conducting-your-own-research/export-of-information

Since sensitive health characteristics such as ethnicity or economic status can be of interest for developing accurate predictive models in healthcare, researchers often need to use such a secure environment. However, general practitioners cannot access this environment. Thus, researchers are creating models that can not be easily used in healthcare, which is unfortunate as they may be useful for doctors. To reduce this gap between the development and the implementation of predictive machine learning models in healthcare, this research focuses on the following question:

> **How to deploy a predictive machine learning model as clinical decision support in general practices considering sensitive population health characteristics?**

1.2 Contributions

This research includes the following deliverables:

- A general coding framework for implementing medical machine learning models as web applications.
- A description of the implementation of predictive machine learning models in healthcare using the same approach as this framework.
- A manual containing the required steps for using this framework as a possible extension for the deployment phase of the CRISP-DM model.
- A UML activity diagram visualising the steps of this manual.

1.3 Thesis Overview

Section 2 gives definitions of terms used in this paper. In section 3, the methods used in this research, literature review and prototyping, are described. Section 4 discusses implementations in the past eight years of machine learning models as a CDSS in healthcare or literature related to the process of deploying such models. In Section 5, four experiments are described in which most machine learning models are implemented successfully as a web application. Section 6 provides the general coding framework and a description of the usage of this framework. In Section 7, the results are discussed in the research context, and limitations & further research are mentioned. Finally, section 8 summarises the results and subsequently answers the research question.

2 Definitions

This paper uses the following terms:

- EHR: an Electronic Health Record is an online medical record of a patient[2].
- CDSS: a Clinical Decision Support System is a system that assists a physician with information to improve healthcare[3].
- HL7: "Health Level Seven International (HL7) is a not-for-profit, ANSI-accredited standards developing organization dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing and retrieval of electronic health information that supports clinical practice and the management, delivery and evaluation of health services."[4].
- HL7 FHIR: HL7 Fast Healthcare Interoperability Resources is "a standard for health care data exchange, published by HL7"[5].

[2]Source: https://www.healthit.gov/faq/what-electronic-health-record-ehr
[3]Source: https://www.healthit.gov/topic/safety/clinical-decision-support
[4]Source: https://www.hl7.org/
[5]Source: http://hl7.org/fhir/R4/index.html

- OMOP-CDM: "The Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) is an open community data standard, designed to standardize the structure and content of observational data and to enable efficient analyses that can produce reliable evidence."[6].
- Streamlit: "Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science."[7].
- PyCaret: "PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows."[8].
- TRL: Technology Readiness Levels are specific phases in product development. The levels are between 1 and 9, where 1 is only the start of the development, and 9 is a product ready to enter the market[9].
- API: Application Programming Interface enables computer programs to communicate with each other[10].
- FHIR RESTful API: Representational State Transfer (REST) is utilised to exchange data in the FHIR API. "REST is a method of exchanging information using the World Wide Web standard transfer protocol HTTP, Hypertext Transfer Protocol, the underlying internet standard that forms the basis for all website data exchange".[11].
- CDS Hooks: Clinical Decision Support (CDS) Hooks "provides a way to embed near real-time functionality in a clinician's workflow when an EHR (Electronic Health Record) system notifies external services that a specific activity occurred within an EHR user session."[12].
- Pickle file: A pickle file is a Python object, a trained model in this study, serialised into a byte stream[13]. These files can be 'unpickled', transformed back into the object (a trained model) using the 'pickle' module in Python, and used elsewhere without the need for retraining.
- RDS file: An RDS file in R contains an R object in binary form[14]. It is roughly the same as a pickle file in Python, as it can be 'unserialised' (converted back) into a single R object.

3 Methods

3.1 Literature Review

To find literature relevant to the topic of this research, no systematic literature review was conducted as it was a major challenge to find unity in the terminology of this novel study. Eventually, the

[6]Source: https://www.ohdsi.org/data-standardization/
[7]Source: https://docs.streamlit.io/
[8]Source: https://pycaret.org/
[9]Source: https://www.rvo.nl/onderwerpen/trl
[10]Source: https://www.ibm.com/topics/api
[11]Source: https://www.healthit.gov/topic/standards-technology/standards/fhir-fact-sheets
[12]Source: https://www.trisotech.com/cds-hook/
[13]Source: https://docs.python.org/3/library/pickle.html
[14]Source: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/readRDS

search queries in Table 1 were entered on Google Scholar.

Search Query	Paper(s)
Model deployment of machine learning models in healthcare	Jackson et al. (2019)
Model deployment in healthcare	Reddy et al. (2020) & Cohen et al. (2021)
Model deployment as clinical decision support systems	Yoo et al. (2022) & Gruendner et al. (2019)
Model deployment as clinical decision support platform	Del Fiol et al. (2020)
A standards-based clinical decision support platform	Bradshaw et al. (2022)
Implement predictive healthcare model using Pycaret and Streamlit	Motaib et al. (2022) & Adewunmi et al. (2022) & Yao et al. (2022) & Zhong et al. (2023)

Table 1: Search queries during the literature review

From here, the backward snowballing method, finding new literature in the references of relevant papers (Wohlin, 2014), was used to find more papers on the topic. Gruendner et al. (2019) and Yoo et al. (2022) refer to the paper of Khalilia et al. (2015). Yoo et al. (2022) also refers to the application of Gruendner et al. (2019). On the U-Prevent website at ‘publications‘ the papers of Groenhof et al. (2019), Nijman et al. (2021) and “SCORE2 risk prediction algorithms: new models to estimate 10-year risk of cardiovascular disease in Europe” (2021) were selected. Thereby, Nijman et al. (2021) refers to the CDSS of Groenhof et al. (2019). The papers about early child development by van Buuren et al. (2015) and Lancaster et al. (2020) were searched for specifically. Furthermore, Ramamurthy et al. (2017) is only referred to by Jackson et al. (2019) for their pipeline.

Papers about model implementation written before 2015 were omitted as it is a relatively new and fast-moving field, which makes older articles less relevant. Thereby it prevents the literature review for this paper from becoming too large. Searching for new literature was stopped after no new relevant papers were found in the references and by the queries on Google Scholar. The queries were combinations of the following keywords: *model*, *deployment*, *implementation*, *machine learning*, *predictive*, *healthcare* and *clinical decision support system/platform*. Thus accessible and relevant papers since 2015 will be discussed in the literature review.

In total, 18 papers are discussed in the related work, from which 14 describe a successful implementation until a certain TRL level.

3.2 Prototyping

This research will develop a general coding framework to simplify and automate the deployment phase of CRISP-DM using the method of prototyping. Prototyping is an approach where primitive versions of the product are developed instead of developing the final product instantly. This way of development allows for easier adjustments to the framework. This study will generate several implementations for Python and R medical machine learning models as they are commonly used programming languages in data science[15]. These models will be developed using code available on Kaggle. Kaggle is a platform where people can share code, datasets and other related data science material[16].

Out of these implementations, the coding framework will be created that provides an easier implementation of machine learning models in healthcare and enhances the deployment phase of the CRISP-DM model. To prevent privacy issues, this framework should not need access to the original training data but only access to a file of the model and information about the preprocessing & naming of the data. As there is no access to the sensitive health characteristics of patients, such as financial status or ethnicity, it is the intention that doctors enter the variables themselves when speaking to the patient.

This deployment approach of the framework has similarities with the organisational structure *decentralised translation* (Kashyap et al., 2021). In this organisational structure, the model is built & trained by researchers solely. Only once they enter the deployment phase do they collaborate with the IT department of a healthcare instance for a smooth integration of the model into their workflow.

4 Related Work

In this section, a literature review will be given. Papers are clustered in subsections because they have the same approach or purpose. Firstly, subsection 4.1 discusses applications trying to provide real-time interaction between the EHR data, predictive model and user via healthcare standards. Secondly, in subsection 4.2, a hybrid approach combining a machine learning model's development and deployment phase is briefly covered. Then, subsection 4.3 talks about predictive models deployed as a web page. Furthermore, in subsection 4.4, multiple studies using Pycaret and Streamlit to develop and deploy their model are addressed. Next, subsection 4.5 will briefly touch upon the ethical and legal issues of AI implementations. Finally, a summary of the different applications is given in subsection 4.6.

4.1 Deployment with Healthcare Standards

In 2015, Khalilia et al. (2015) wrote a paper about real-time development and deployment of clinical predictive models using the web services of HL7 FHIR. Figure 2 shows the architecture they proposed. The EHR data are saved in an OMOP-CDM database and used to train predictive models. The training phase can handle all sorts of data as long as they are saved in an OMOP-CDM. OMOP-CDM, also briefly stated in Section 2, provides a way to convert heterogeneous data like EHR data into a standardised format for analytical purposes. Important aspects are the OHDSI vocabularies that provide harmonising the healthcare data into standardised medical terminologies. However, it is the intention that, over time, the information is directly retrieved from the EHR with the use of FHIR resources. As briefly mentioned in Section 2, HL7 FHIR web services offer a standardised format for data exchange in healthcare, allowing for easier medical data interoperability. For this, FHIR uses certain 'Resources' that represent healthcare components

[15]Source: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022
[16]Source: https://www.kaggle.com/

like patients or medication in XML[17], JSON[18] or RDF[19]. These resources can be exchanged across systems.

Regarding the architecture, the best models are uploaded to the FHIR server. The paper shows this for a Random Forest, SVM and KNN model. Users, physicians, can ask for this predictive model within their own systems, as FHIR web services will take care of the integration. The integration works abstractly as follows. FHIR resource on the doctor's device wraps the patient's needed information and sends it via the FHIR RESTful API to the FHIR server and finally to the predictive model. The predictive model calculates a risk score and returns this to the user. The score is also stored in a resource database. This can be sensitive information, which is why the system of our research will not be saving the scores.

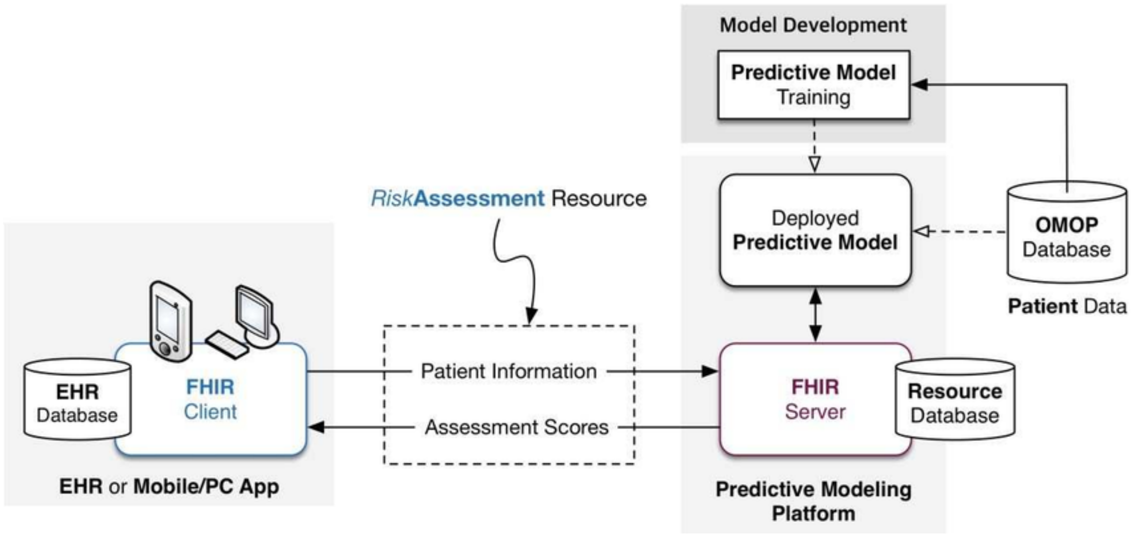


Figure 2: Architecture of the system developed by Khalilia et al. (2015)

In Oktober 2019 Gruendner et al. (2019) created KETOS, an environment designed for researchers to train and implement decision support and machine learning models in the ICT environment of a hospital. KETOS comes from $\kappa\tilde{\eta}\tau o\zeta$, which means sea monster in Greek. This name is chosen because the application depends on Docker, which has a whale on its logo. Figure 3 shows that with the use of a graphical user interface (GUI), scientists can make a virtual development environment using Docker containers. This environment provides basic resources such as R, Python and DataSHIELD. They can access their environment with Jupyter Notebook, where data can be retrieved via a preprocessing service, and a model can be trained and, in the end, deployed.

The preprocessing service tries to supply data that is structured and of high quality. This service gets the data from an OMOP-CDM or other database using an FHIR web service and then converts it to a CSV file. However, it is also possible for the developer to load their own data via the notebook. During the development of a model, changes can be saved and later used to continue the

[17]Source: https://www.hl7.org/fhir/xml.html
[18]Source: https://www.hl7.org/fhir/json.html
[19]Source: https://www.hl7.org/fhir/rdf.html

work. For a smooth integration of the model into the app used by the doctor, a certain function specified by the researcher should be called in this app. In the end, the dashboard of this platform provides the development, training and deployment of machine learning models on sensitive data in a secure way.

Overall, this architecture may still be complicated for a user without much computer science background. Therefore, our study focuses on simplifying the implementation of predictive models as much as possible.

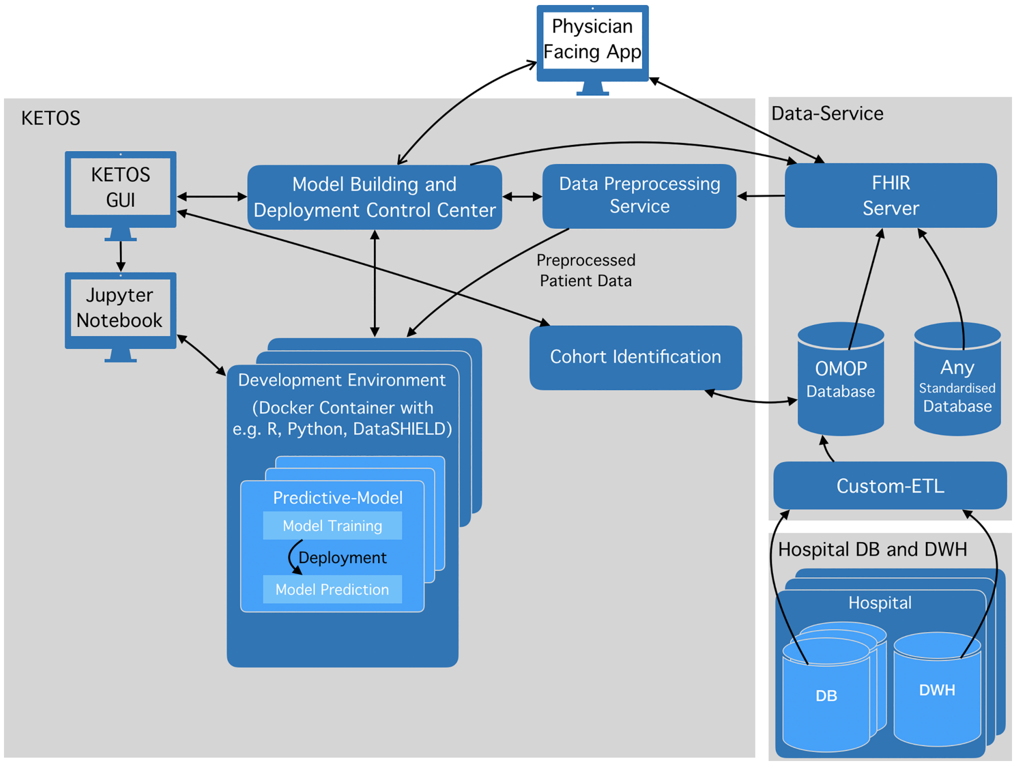


Figure 3: The architecture of KETOS[20]

At the beginning of 2020, Del Fiol et al. (2020) implemented a rule-based algorithm using the following open-source CDS software, including a population health management (PHM) platform: http://www.opencds.org/. The algorithm identifies patients meeting the criteria for genetic evaluation of familial cancer. These criteria rely on the patient's family health history (FFH) recorded in an EHR. During the development of this standard-based CDS platform, it was intended to be implemented at several healthcare organisations with the use of standard APIs. This wasn't entirely successful as custom adaptation needed to be made, partly because HL7 FHIR is not universally used but also because EHR systems allow for different terms of, for example, the same diseases. In Figure 4, the architecture of this standard-based CDS platform is shown. Bradshaw et al. (2022) continued working on the application after this first pilot, later called Genetic Cancer Risk Detector (GARDE). In the architecture, CDS Hooks are used. CDS Hooks, also briefly mentioned in Section 2, provides a systematic way for CDS services to communicate with EHR systems. With

[20]Source: https://doi.org/10.1371/journal.pone.0223010.g002

this help, recommendations to healthcare professionals based on EHRs can be given.

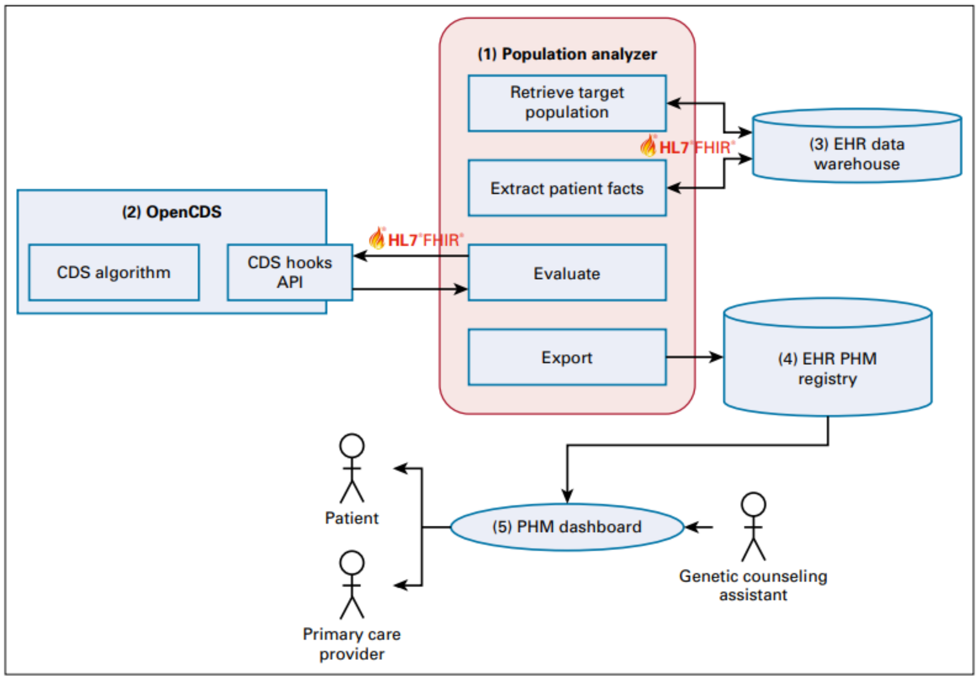


Figure 4: The architecture of a standard-based CDS platform embedded with an EHR system (Del Fiol et al., 2020)

In 2022, Yoo et al. (2022) developed CANE as a platform to support the implementation of CDS systems across different EHR products. Models on this platform are usually trained using data from an OMOP-CDM database, but training on data in another format is also allowed. The developed models, often R or Python objects, are translated into C#, a programming language. This translation allows the programs to run on any operating system. The entire process of an operation is shown in Figure 5.

On the user, physician, interface (UI), there is a 'CANE' button. After clicking this button, the input data are transformed into the right JavaScript Object Notation (JSON) format with the help of HL7 FHIR resources. This input is preprocessed on the CANE server to run the model and receive a calculated output. The integration module provides supplementary data by interaction with the CANE systems of other institutions. To protect the privacy of this data, a population-level summary of it is given. Finally, a UI is created and sent back to the EHR using the additional data and calculated risk score.

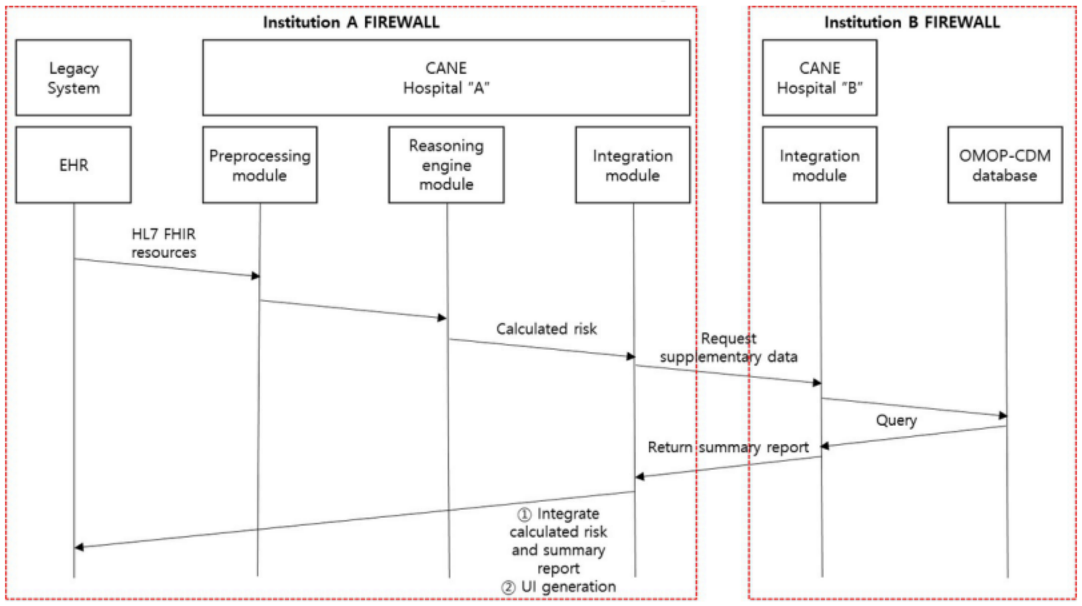


Figure 5: Operation process of a CDSS using CANE (Yoo et al., 2022)

In the previous applications, three different standards (HL7 FHIR, OMOP-CDM, CDS Hooks) have been used and discussed. They all try to increase the interoperability of healthcare data but have slightly different priorities. Where FHIR focuses on exchanging medical data between applications, OMOP-CDM standardises the data for observational use. CDS Hooks, on the other hand, provide real-time interaction between a CDSS & the data. Anyway, they can be used together, as shown in the implementations above. Almost all of these implementations reach TRL 6 as a pilot has taken place except for the first one, which has a TRL level of 4/5. These studies try to increase the usability of models in hospitals by providing real-time interaction with EHR data in their implementations. However, in this study, the focus is on implementing without direct access to data so security regulations do not obstruct the research. Therefore, the integration of the development and deployment phase of the models will not be included in this research. Nevertheless, it is good to consider the possibilities of the discussed resources in healthcare.

4.2 Agile Deployment

At the start of 2019, a more general approach was proposed by Jackson et al. (2019). They proposed an agile deployment of machine learning models in healthcare to improve the deployment process. In contrast to other subsections, where the emphasis is mainly on the technical aspect of the implementations, here, the emphasis is on the agile process, which is a way of working together. This hybrid approach includes an incremental product release allowing continuous feedback from the application domain. Therefore, the focus is on incorporating the developing and deploying stage of models to provide better maintenance of the application, amongst other things. This research could be considered an addition to MLOPs. MLOps is an approach for maintaining and deploying machine learning models by providing a set of tools. As a use case, they have implemented an improved version of a pipeline developed by Ramamurthy et al. (2017). This pipeline contains a multi-stage regression model and concerns healthcare cost prediction. The application is deployed end-to-end but not used in practice yet, so it has a TRL of 6. It seems logical that healthcare insurance companies will use this application. An agile deployment is a good approach when focusing more on implementing and maintaining evolving models. However, in our paper, the focus lies on deploying models without having access to a development phase.

4.3 Deployment as Web Page

A different concept is developing an online risk calculator available for everyone on the internet. On http://www.u-prevent.com/, users are able to calculate their personal cardiovascular risk (CVR) profile and more. Several models are implemented on this website, but the most recent ones, SCORE2, are survival analysis models ("SCORE2 risk prediction algorithms: new models to estimate 10-year risk of cardiovascular disease in Europe", 2021). It is intended that the implementation, created with the general coding framework of this research, should look approximately like the CVR calculator.

In the summer of 2019, Groenhof et al. (2019) implemented a different model, a decision tree (with more prediction rules of varying time frames), concerning the same health issue. However, instead of deploying the model as a CVR calculator on the web, it is deployed as a CDSS for cardiovascular risk management (CVRM) embedded in the EHR environment. As this paper is mentioned in the publications on the website of U-prevent, it is discussed in this subsection after U-prevent itself, even though the implementation isn't a web page. The architecture behind the dashboard of this application for clinicians can be seen in Figure 6.

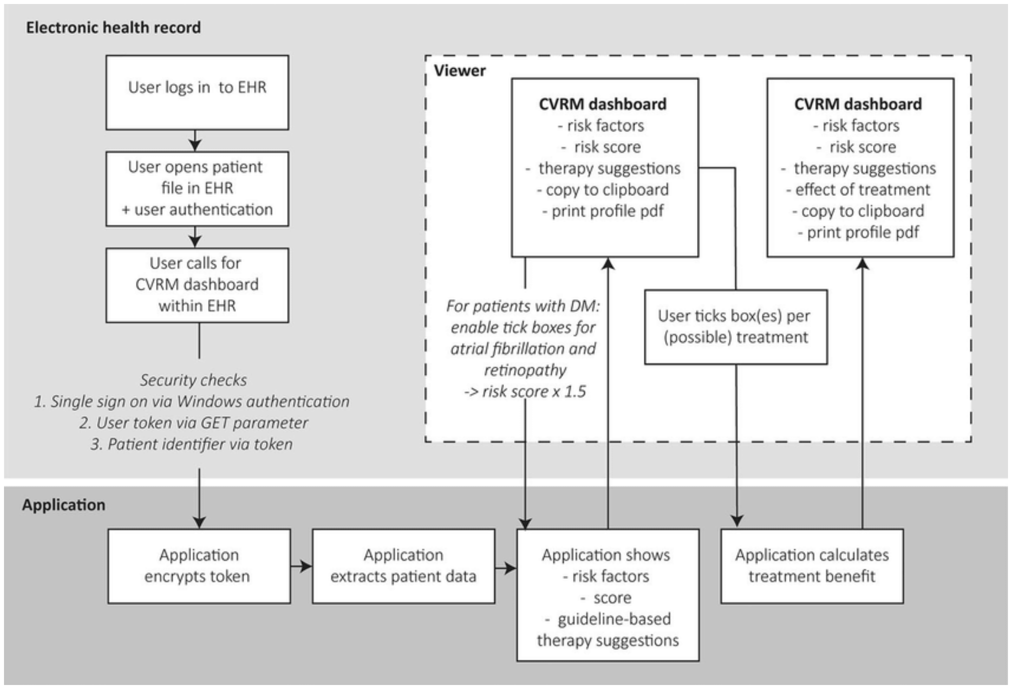

Electronic health record
User logs in to EHR
User opens patient file in EHR + user authentication
User calls for CVRM dashboard within EHR
Security checks
1. Single sign on via Windows authentication
2. User token via GET parameter
3. Patient identifier via token
Viewer
CVRM dashboard
- risk factors
- risk score
- therapy suggestions
- copy to clipboard
- print profile pdf
CVRM dashboard
- risk factors
- risk score
- therapy suggestions
- effect of treatment
- copy to clipboard
- print profile pdf
For patients with DM: enable tick boxes for atrial fibrillation and retinopathy -> risk score x 1.5
User ticks box(es) per (possible) treatment
Application
Application encrypts token
Application extracts patient data
Application shows
- risk factors
- score
- guideline-based therapy suggestions
Application calculates treatment benefit

Figure 6: The architecture of a CDSS for CVRM in an EHR environment (Groenhof et al., 2019)

A challenge for this CDSS is missing data. Without these values, no good predictions can be made.

Nijman et al. (2021) discussed different methods for imputing the missing values: mean imputation, joint modelling imputation (JMI) and conditional modelling imputation (CMI). JMI uses the training data to calculate the mean and covariance. These statistics are used, in combination with the known values of the predictive factors, to produce an imputation. CMI, on the other hand, uses the training sample to create a model for each predictor in order to estimate an imputation value. In the end, the model imputation methods JMI and CMI performed best overall when predictor values were missing. In our study's framework, there is no need to impute missing values, as the user fills in those values themselves.

Anyhow, the U-Prevent website reaches a TRL of 9 as the online CVR calculator is on the market, and multiple healthcare models are available for everyone. The TRL level for the CDSS by Groenhof et al. (2019) is six because validation and evaluation with healthcare professionals have been done, but testing the application in an operational environment isn't executed yet.

A growth predictor is a different kind of online calculator than U-prevent. Stef van Buuren and Arjan Huizing started in 2019 with the development of a web service called JAMES, https://james.groeidiagrammen.nl/, which is short for Joint Automatic Measurement and Evaluation System. This web service is used for the implementation of an early child growth predictor that can be found on https://tnochildhealthstatistics.shinyapps.io/Groeicalculator/. This application, which is based on just a linear regression, helps the healthcare professional detect an unusual growth in the height or size for a child (van Buuren et al., 2015).

This web page is another example of how an implementation should roughly look using the intended framework in this study. JAMES ensures that the growth predictor can be used by anyone online and therefore has a TRL of 9.

In 2020 another tool for early child development was made. Lancaster et al. (2020) developed an instrument that gives a standardised measurement for child development between the ages of 0 and 3 using logistic regression. The website of this Infant and Young Child Development (IYCD) tool can be found at the following link: https://ezcollab.who.int/iycd. They even implemented the tool on a tablet using images and audio to make it easier.

Nevertheless, the website has a different format than is planned for the implementation with the general framework of our research, as you first need to log in and download files. Our study aims to generate an interface that is easier to use. The IYCD tool is not complete yet but has been tested in a real-life setting, so it reaches a TRL of 7.

4.4 Deployment with Pycaret & Streamlit

In 2022 and 2023, another easy approach for implementing machine learning models as web applications was found. Models related to healthcare have been created with a pipeline in PyCaret and are deployed on a local web server via Streamlit.

Motaib et al. (2022) developed a machine learning model (Extra tree classifier) and deployed this model with Streamlit as an online calculator predicting poor glycated control for patients who do not fast during Ramadan. Streamlit is also used to create a web app that visualises various variables relating to different cancer treatments across patients (Adewunmi et al., 2022). When uploading relevant data as input, the web app displays the results of two models created by the libraries PyCaret and BERTopic. Another example of a website generated using the Streamlit library is an online calculator that predicts the chance of a positive outcome of treatment with mechanical thrombectomy (MT) for patients with acute ischemic stroke (AIS) (Yao et al., 2022). The implementation of one of the developed machine learning models in the paper is available on https://zhelvyao-123-60-sial5s.streamlit.app/. Zhong et al. (2023) also developed and deployed a machine learning model (Random Forest) that predicts the short-term clinical outcome of patients with a certain autoimmune disease called Myasthenia gravis using both PyCaret and Streamlit. The interactive implementation can be found on http://47.99.108.154:8501/ .

Even though the implementations provide a product, they have a TRL of around four because they are not tested in a relevant environment. Depending on the purpose of the interactive web app, it could be used by doctors or patients. In these papers, Streamlit is shown to be a great tool for making web apps for machine learning models and can also be useful for developing the general framework.

4.5 Deployment Constraints

With the upcoming applications of AI in healthcare, as outlined in the previous subsections, Reddy et al. (2020) created a governance model, taking into account the ethical and legal issues. This is important to consider when implementing a machine learning model into the healthcare system. Is it, for example, ethical for a doctor to make their treatment partially based on a black-box model that they can not fully understand? Or how do we ensure that the patient's privacy is guaranteed? What if, for instance, the model does not perform as well as expected when the training sample does not match the real data (Cohen et al., 2021)? This governance model is the first step in practically addressing these problems. Figure 7 shows four main aspects: Fairness, Transparency, Trustworthiness & Accountability. Fairness is more related to the development of models. Transparency and trustworthiness, on the other hand, are components to consider when deploying models in general practice. By explaining the model in the implementation, transparency and also trustworthiness will increase. As the implemented models are only tools for the physicians, the healthcare professionals will stay accountable for the decision they make.

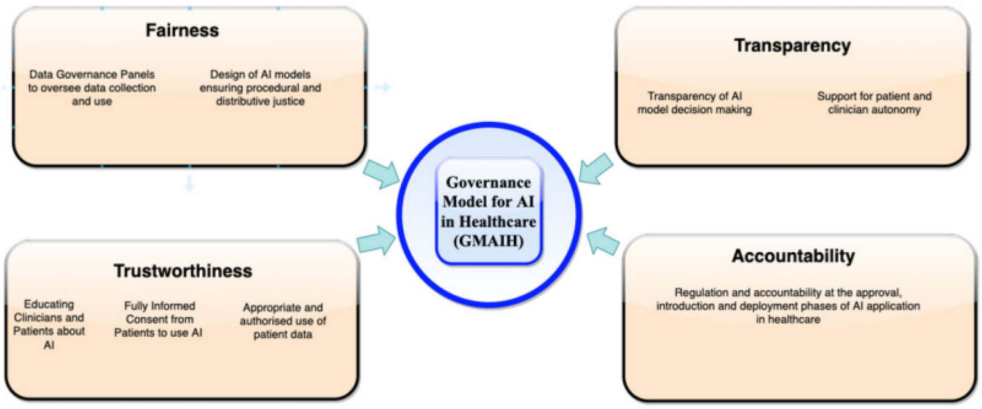

Fairness
Data Governance Panels to oversee data collection and use
Design of AI models ensuring procedural and distributive justice
Transparency
Transparency of AI model decision making
Support for patient and clinician autonomy
Governance Model for AI in Healthcare (GMAIH)
Trustworthiness
Educating Clinicians and Patients about AI
Fully Informed Consent from Patients to use AI
Appropriate and authorised use of patient data
Accountability
Regulation and accountability at the approval, introduction and deployment phases of AI application in healthcare

Figure 7: "A governance model for the application of AI in healthcare" (Reddy et al., 2020)

4.6 Summary

Thus, multiple implementations of models related to healthcare in the past few years have been discussed. One is more broadly applicable than the other. In Table 2, the different implementations are compared. Note that in this table, some have *machine learning* in the column 'Model type', meaning that multiple machine learning models are implemented by the application(s).

	Model type	Target group	TRL	Embedded in an EHR system	Publication year
Khalilia et al. (2015)	random rorest, SVM and KNN	physician	4/5	yes	2015
KETOS	machine learning and decision support	physician	6	yes	2020
Pilot GARDE	rule-based	physician	6	yes	2020
CANE	deep-learning and rule-based	physician	6	yes	2022
Healthcare cost prediction	multi-stage regression	healthcare insurance company	6	no	2019
U-Prevent	survival analysis models	physician/patient	9	no	2021
A CDSS for CVRM by Groenhof et al. (2019)	decision tree	physician	6	yes	2019
Growth calculator	linear regression	physician/patient	9	no	2015
IYCD tool	logistic regression	physician/patient	7	no	2020
Pycaret & Streamlit	machine learning	physician/patient	4	no	2022-2023

Table 2: Comparison of discussed model implementations

Noticeable is that recently there has been an increase in the use of Streamlit in combination with PyCaret for the development and deployment of machine learning models as web apps. In this paper, it is the intention to come up with a general coding framework for the implementation of such healthcare models. Streamlit would be an excellent tool for this framework as it allows for relatively easy implementation of predictive models as risk calculators on a local device. The target groups of this interactive prototype web app are patients and physicians. This research should reach a TRL of 4 as it is the intention to test the idea on several models and create a prototype version of the framework. Furthermore, the risk calculator should not be embedded within an EHR system to prevent problems with data privacy. Therefore, patients and doctors can fill in the data themselves. Unlike the Streamlit implementations in subsection 4.4 that show a specific model, this study will implement a diverse collection of predictive models to create a general framework.

5 Experiments

As mentioned in Section 3, the general coding framework results from multiple implementations of machine learning models in R and Python. The experiments work towards a manual for implementing machine learning models as interactive web apps. Every experiment adds steps to this manual. These experiments can be found in the folder ‘experiments‘ on the `GitHub repository` and will be discussed in this section. For these experiments, the performance of the models does not matter.

5.1 Heart Condition Checker

This `project` on Kaggle is a great example of implementing a machine learning model. Based on a dataset containing ‘Personal Key Indicators of Heart Disease‘, a logistic regression was developed in Python, saved as a pickle file and implemented using Streamlit. With the help of Heroku[21], which is a platform as a service, the application was uploaded to the web[22]. On the web app there is a link to the `GitHub repository` of the project made by Kamil Pytlak. As the approach is pretty similar to the approach of this research, the repository is cloned and, once the necessary libraries are installed, run on a local machine to gain a better insight into this Streamlit implementation.

The application still uses the dataset for preprocessing the input data. This is changed, and an application that works without this dataset has been made. The input possibilities (see Appendix A for a further explanation with example) are manually added to the input data frame so that the same preprocessing can occur. Now, the data is not necessary anymore for the implementation, which this research wants to achieve. Only a pickle file of the model and information about the preprocessing & naming of the data are needed. Also, the text was adjusted a bit in the application by adding my name to it.

During this first experiment, the following steps for implementing a predictive model were executed:

1. **Prepare for new environment**: *install required packages*

2. **Adjust application**: *import a pickle file, change name & text of application, get the user input & input possibilities into a pandas data frame, preprocess the user input*

3. **Execute model**: *get the prediction & corresponding probabilities out of the pickle file, print the prediction*

4. **Deploy application on machine**: *run & stop the application via the terminal*

5.2 Implementation of heart failure prediction models in Python

For this research, it is essential to explore if other machine learning models developed in Python could be implemented in approximately the same way. Therefore, this `dataset` and `notebook` on Kaggle have been used to develop five different models in a new notebook. After preprocessing, the notebook builds a logistic regression, a support vector classifier, a decision tree classifier, a random forest classifier and a k-nearest neighbors classifier. It is important to note that it is necessary to

[21]Source: `https://www.heroku.com/`

[22]Source: `https://kamilpytlak-heart-condition-checker-app-2r42q4.streamlit.app/`

set the probability in the support vector classifier to true; otherwise, it is impossible to get the probability of the predictions from the model later.

In the end, these models are saved as pickle files. The approach for implementing these models is, except for the preprocessing, pretty much the same as Kamil Pytlak's. This can vary greatly as it depends on the developers preprocessing of the training data given to the model. In contrast to the previous subsection, the categorical variables are encoded manually to experiment with different preprocessing methods. As mistakes are easily made, the resulting data frame is often printed while coding this part. Also, the layout of the code is different as it is split up into more functions.

In an attempt to give explanations with the models as well, the feature importances of variables of the random forest model and decision tree are plotted after the prediction and corresponding probability. This feature importance can be extracted directly from the imported model. However, this is not possible for every model. Therefore, there is also an option to plot SHapley Additive exPlanations (SHAP) values, an example of explainable AI. SHAP values explain the influence of the value of a feature on the prediction of the model[23]. In Python there is a package named 'shap', which enables easy calculation of SHAP values.

The SHAP values for training data of the logistic regression and support vector classifier are calculated in the notebook and exported as a pickle file. The 'shap' library could have been used in the application code, but calculating the values requires the original data, which this research does not want to use in the framework. The SHAP values of the logistic regression are shown instead of the coefficients because SHAP values are applicable to other models like the support vector classifier as well. It is also possible to calculate the SHAP values for the test data. SHAP values for training data say something about feature importance during model training, and SHAP values for test data try to explain the impact of the features on the prediction of new data. For the k-nearest neighbors classifier, extracting feature importances or SHAP values was not possible in the same way as for the other models. Therefore, no plot is generated for this model.

This second experiment enriches the steps for implementing a machine learning model. These enrichments are underlined:

1. **Prepare for new environment**: *install required packages*, *make sure the model provides probabilities along with the prediction*, *export model as pickle file out of data environment*

2. **Adjust application**: *import a pickle file*, *change name & text of application*, *get the user input & input possibilities into a pandas data frame*, *preprocess the user input exactly the same as the training data*

3. **Execute model**: *check the input values in the data frame*, *get the prediction & corresponding probabilities out of the pickle file*, *print the prediction*, *explain the model by feature importances or SHAP values*

4. **Deploy application on machine**: *run & stop the application via the terminal*

5.3 Implementation of stroke prediction models in Python

As it is the intention to implement models that concern different types of medical issues, another dataset about stroke prediction is used for making machine learning models. Again code from

[23]Source: https://www.kaggle.com/code/dansbecker/shap-values

a notebook on Kaggle was used for developing the models with the new dataset. After data preparation, three different pipelines are made: a random forest classifier, a support vector classifier and a logistic regression. As also mentioned earlier, it is necessary to set the probability in the support vector classifier to true in order to receive the probability of the model's predictions. After these pipelines are fitted and optimised, they are exported as pickle files.

These pickle files are imported and used to predict. Less preprocessing needs to be done since the pipelines include automatic data scaling. Only label encoding of categorical input variables in the same way as the notebook is required. Also, there is in this implementation an option for the random forest model to plot the feature importances and for the logistic regression to plot the SHAP values of the variables. This file has the same structure as the one in subsection 5.2. Thus, the same approach is successfully used for a stroke risk predictor. Furthermore, experimenting is done with different pictures and layouts as it concerns another health issue than before. This resulted in using no images in the web app as they added no valuable information. Only a bar plot can be displayed, providing explainability to the model. To emphasise the prediction class, a good outcome is shown in green and a bad outcome in red. Also, the predict button was removed from the main part to the sidebar to keep the interactive part of the web app together on one side.

This experiment with a stroke dataset includes only one addition to the steps mentioned in the previous subsections:

1. **Prepare for new environment**: *install required packages*, *make sure the model provides probabilities along with the prediction*, *export model as pickle file out of data environment*

2. **Adjust application**: *import a pickle file*, *change name & text of application*, <u>*change layout & pictures*</u>, *get the user input & input possibilities into a pandas data frame*, *preprocess the user input exactly the same as the training data*

3. **Execute model**: *check the input values in the data frame*, *get the prediction & corresponding probabilities out of the pickle file*, *print the prediction*, *explain the model by feature importances or SHAP values*

4. **Deploy application on machine**: *run & stop the application via the terminal*

5.4 Implementation of heart failure prediction models in R

Machine learning models are not only created in Python; R is also a widely used programming language for this topic. Therefore, the implementation of R models should be possible as well. From a notebook on Kaggle, code for the implementation of the six models with the highest accuracy of the 14 models is used. These models are trained on the same dataset as in subsection 5.2, but this data is preprocessed differently. After preparation of the data, workflows are made for a random forest algorithm, extreme gradient boosting, multivariate adaptive regression splines (mars), ensembles of mars models, k-nearest neighbors algorithm, naive bayes classifier and logistic regression. The models are extracted from the workflows and saved as RDS files.

To import these models in the implementation code, the library 'rpy2' is used. This package provides an interface for R in Python[24]. Since using R in Python produces the warning that the application can not be stopped with 'Ctrl-C' in the terminal anymore, a 'stop' button is added

[24]Source: https://rpy2.github.io/

to the web app so that the application can be stopped in another way. This button is also added to the other implementations. With the help of the ‘rpy2‘ library, it is also possible to make a prediction about the input data and get the related probabilities. Here, the input possibilities are added manually to be able to prepare the input data the same as the training data. In order to check the values of the data frame easily, the order of the columns is changed to the order of the original data frame.

In some workflows, the feature importance could be turned on for the model. Therefore, there is an option for the random forest algorithm, extreme gradient boosting and logistic regression to plot their feature importances. Importing the ‘vip‘ library from R with ‘rpy2‘ is necessary for this plot. Overall this method is pretty similar to the previous ones. Only the syntax is slightly different for importing the files, making a prediction and getting the feature importances.

This experiment with models developed in R includes a few additions to the implementation steps:

1. **Prepare for new environment**: *install required packages, make sure the model provides probabilities along with the prediction, turn feature importance on for the model if possible, export model as pickle file or RDS file out of data environment*

2. **Adjust application**: *import a pickle file or RDS file, change name & text of application, change layout & pictures, get the user input & input possibilities into a pandas data frame, preprocess the user input exactly the same as the training data, reorder the data frame*

3. **Execute model**: *check the input values in the data frame, get the prediction & corresponding probabilities out of the pickle file or RDS file, print the prediction, explain the model by feature importances or SHAP values*

4. **Deploy application on machine**: *run & stop the application via the terminal, stop the application in the web app itself*

5.5 Implementation of a Cox-PH model in R

A Cox Proportional-Hazards model (Cox-PH) is a kind of regression model used to analyse the relationship between predictor variables and the time until the event of interest happens (survival time)[25]. As this survival analysis technique is often used in medical research, it would be valuable to implement such a model as well. On Kaggle there is this `script` using another heart failure prediction `dataset` to develop a Cox-PH model. Code from this script has been used to build and save a Cox-PH model as an RDS file.

However, it failed to get the survival probability into the web application successfully. Survival probability estimates the likelihood that a subject survives beyond a given time ‘t‘ (Baek et al., 2021). When calling the survival prediction function of the Cox-PH model (see Listing 1), an error occurs, stating that the object `DEATH_EVENT` is not found. This is the event of interest for the model. The function seems to search for an object called `DEATH_EVENT`, while `DEATH_EVENT` is just a column of the data frame in the code. This is quite surprising as during the development of the model in R, almost the same syntax does give a survival probability (see Listing 2). Therefore, the error probably lies somewhere in integrating this R functionality in Python.

[25]Source: `http://www.sthda.com/english/wiki/cox-proportional-hazards-model`

```python
survival = survival.predict_coxph(model_ml , newdata=df_input , type="survival")
```

Listing 1: Python code

```r
surv_prob_num <- predict(cox_num , newdata = X_num , type = "survival")
```

Listing 2: R code

A Cox-PH model is different from the other models as the response of the model's formula is a 'survival' object and not a single variable[26]. The following function call returns this object: `Surv(time, DEATH_EVENT)`. It is noticeable that `DEATH_EVENT`, mentioned in the error message, also appears in this function call. Perhaps the response object is incorrectly handled in the Python code resulting in the error. However, even after consulting an experienced data scientist, no way has been found yet to fix this.

Although the experiment with the Cox-PH model isn't successful, the error analysis provides an add-on, considering the problems with the prediction function of this model, to the four steps:

1. **Prepare for new environment**: *check if the model has a class and probability prediction function*, *install required packages*, *make sure the model provides probabilities along with the prediction*, *turn feature importance on for the model if possible*, *export model as pickle file or RDS file out of data environment*

2. **Adjust application**: *import a pickle file or RDS file*, *change name & text of application*, *change layout & pictures*, *get the user input & input possibilities into a pandas data frame*, *preprocess the user input exactly the same as the training data*, *reorder the data frame*

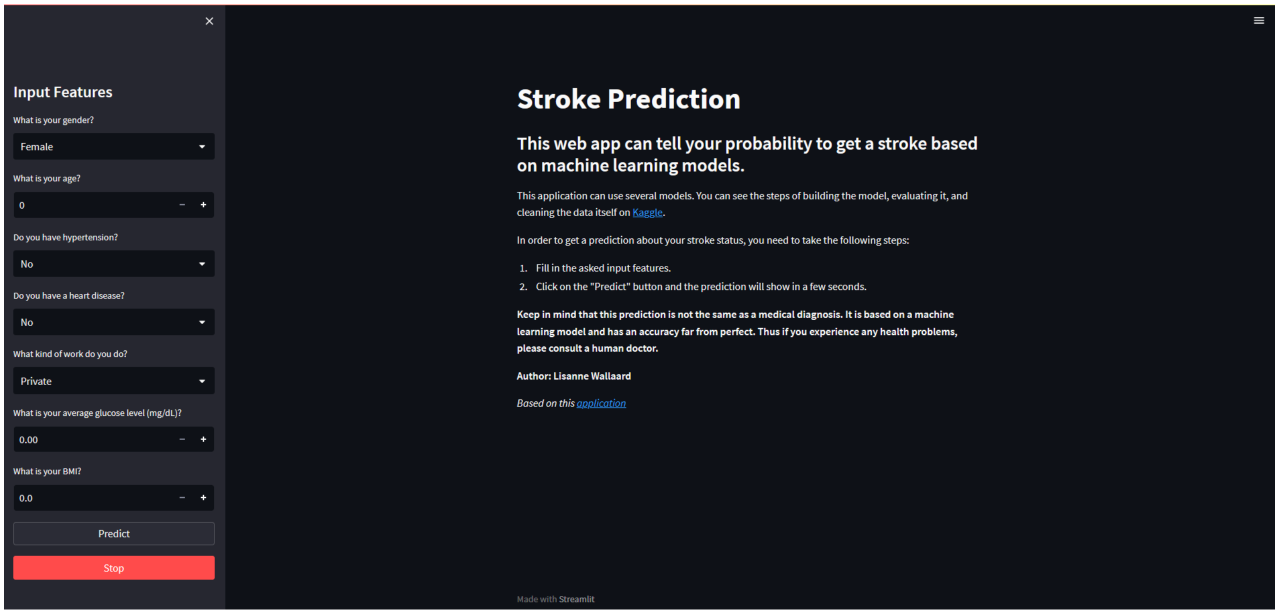
3. **Execute model**: *check the input values in the data frame*, *get the prediction & corresponding probabilities out of the pickle file or RDS file*, *print the prediction*, *explain the model by feature importances or SHAP values*

4. **Deploy application on machine**: *run & stop the application via the terminal*, *stop the application in the web app itself*

6 Results

Out of the experiments discussed in Section 5, a general coding framework is made. In this framework, the steps for implementing a machine learning model as a web app on your device are provided by example code and comments with explanations. Figure 8 shows an interface provided by this framework. In the middle of the picture is an explanation of the web app; the input features can be entered on the left side.

[26]Source: https://www.rdocumentation.org/packages/survival/versions/3.5-5/topics/coxph


Input Features

What is your gender?

Female

What is your age?

0

Do you have hypertension?

No

Do you have a heart disease?

No

What kind of work do you do?

Private

What is your average glucose level (mg/dL)?

0.00

What is your BMI?

0.0

Predict

Stop

Stroke Prediction

This web app can tell your probability to get a stroke based on machine learning models.

This application can use several models. You can see the steps of building the model, evaluating it, and cleaning the data itself on Kaggle.

In order to get a prediction about your stroke status, you need to take the following steps:

1. Fill in the asked input features.
2. Click on the "Predict" button and the prediction will show in a few seconds.

Keep in mind that this prediction is not the same as a medical diagnosis. It is based on a machine learning model and has an accuracy far from perfect. Thus if you experience any health problems, please consult a human doctor.

Author: Lisanne Wallaard

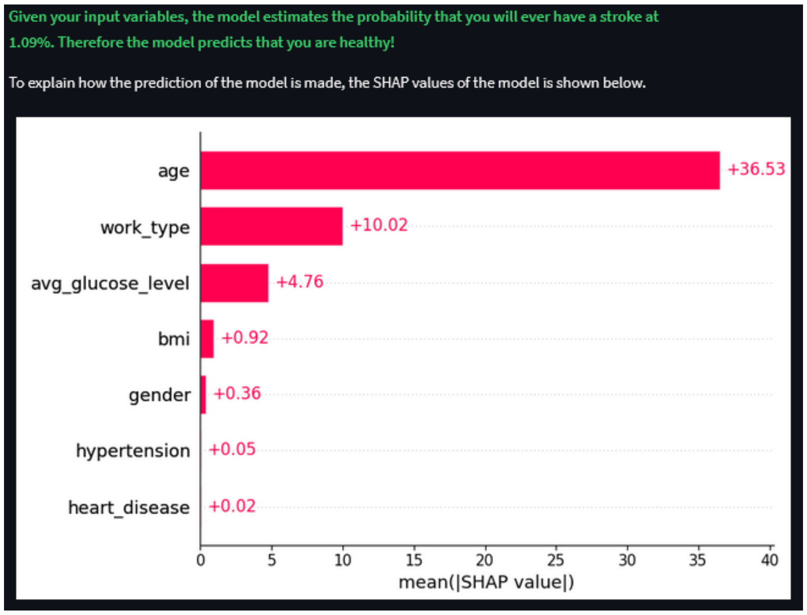
Based on this application

Made with Streamlit

Figure 8: Interface of a stroke prediction web app using the same approach as the framework

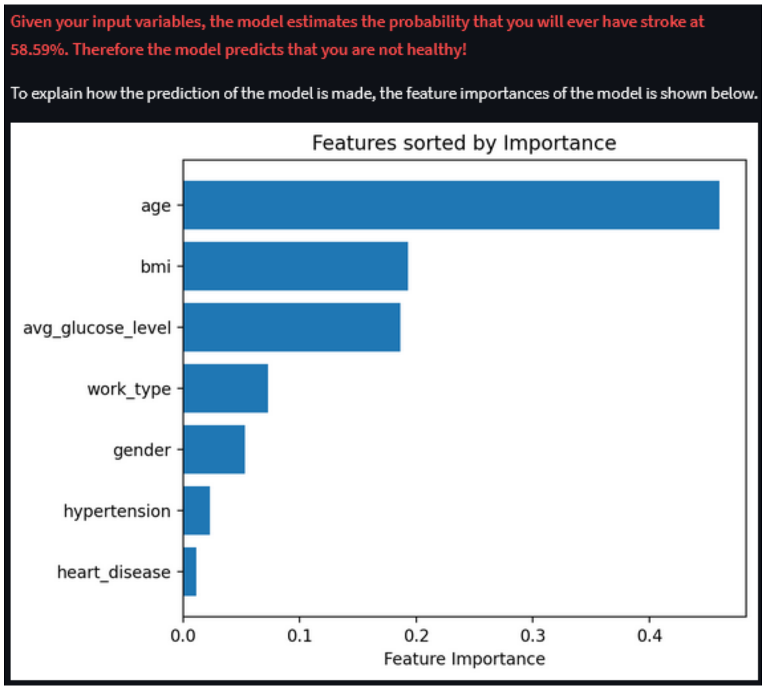
Figure 9 shows the framework's output for a model attached with SHAP values after the user has pressed the predict button. As the probability of a stroke is very low, the model concludes that the user seems healthy, so the prediction is coloured green. The SHAP values show that `age` has the most critical role in this model, which is something the doctor or patients can't change. However, other things in the bar plot can be changed. For example, it may not be wise for older people to do a certain type of work, including physical labour.

In Figure 10, the output of the framework for a model that enables feature importances is shown. This output only appears after the user has entered their input variables and pressed the predict button. As the probability of a stroke is too high, the model concludes that the user might not be healthy, and the prediction is therefore written down in red. The feature importances show that `age` is again the most crucial variable for this model. Age can't be changed, but a physician can give tips to change other important variables like the Body Mass Index (BMI) or the average glucose level of the patient if those variables are not normal.


Given your input variables, the model estimates the probability that you will ever have a stroke at 1.09%. Therefore the model predicts that you are healthy!

To explain how the prediction of the model is made, the SHAP values of the model is shown below.

Figure 9: A stroke prediction of a logistic regression attached with SHAP values


Given your input variables, the model estimates the probability that you will ever have stroke at 58.59%. Therefore the model predicts that you are not healthy!

To explain how the prediction of the model is made, the feature importances of the model is shown below.

Figure 10: A stroke prediction of a random forest classifier where feature importances can be shown

This framework works for the models mentioned in the experiments except for the Cox-PH model. Likely, the framework is also applicable to more machine learning models, but that has not been tested. In Python, different models were built using the 'scikit-learn' library, and each worked pretty easily. So, a not-yet-used model from this same library should most likely work as well. Even better, different models developed in R were implemented roughly the same as long as they had a class and probability prediction function. The Cox-PH model lacked these functions as this survival analysis model has a different approach. Thus, the framework probably works for a model exported as a pickle or RDS file that includes a class and probability prediction function.

Text explaining the steps in the framework should be added to fit the framework into the CRISP-DM key step, 'plan deployment', the relevant step for this research. In the following box, this text is written down. The text is also saved as a pdf called 'manual' and added to the repository. Furthermore, the steps of the manual are visualised in a UML activity diagram (see Appendix B).

Note that the framework provides a way to implement a predictive model on a local device. If it is the intention to upload this application to the web, additional steps should be taken, for example, by using Heroku.

Manual Framework

Introduction

To deploy a medical machine learning model with a class and probability prediction function as a web app on your local machine, you can use this `framework`. You just need to download the file from the `repository` on GitHub and start adjusting it to your model. In this repository, there is also a `ReadMe` explaining the project, giving information about the used packages and providing the installation steps. Many comments explain the code in this coding framework, but this document will also walk you through the steps.

Code Walktrough

A model can be imported only as a pickle file (Python) or RDS file (R). Enter the path to this model on your device in `PATH_MODEL`.
If your model is developed in R and exported as an RDS file, you should also fill in your R path in `PATH_R`. You can get your path to R in RStudio by entering `R.home()`. In the main function, you should use the line of code with `model_rds` for importing the model, and the other lines of code concerning a pickle file can be deleted.
If your model is developed and exported as a pickle file, you can delete the lines of code related to `PATH_R`. Also, the library 'rpy2' is not needed as it is only necessary for the use of R models; the same applies to the variable `r`. In the main function, you should import your model using the code line with `model_pkl` and delete the line of code related to an RDS file.
The text and naming of the application can be changed at the beginning of the file's main function. The comments here are pretty straightforward, so no further explanation is needed.
To add an image as a page icon, just download the one in the repository and put it in the same folder as 'framework.py'. Downloading another image and changing the path of `page_icon` is also possible.
The input data and input possibilities are loaded using the function `input_user()`. In this

function, a few examples are given showing how to get different sorts of input from the user and the corresponding input possibilities in two data frames. The two data frames can be merged, but if you are not planning to use the input possibilities during preprocessing, it is also possible to continue with the input data frame only. Ensure that after the merge, the input row is the first row in the data frame.
Data preprocessing is done by the function `preprocess_input()`. This function gives a few examples of labelling, one hot encoding and scaling. However, it depends very much on the preparation of the model's training data since the input data should be preprocessed in precisely the same way. After the preprocessing, you might want to reorder the data frame to your preference. It is advised to check the values in the resulting data frame by `st.dataframe(df)`.
Two buttons are added to the application: `submission` and `stop`. The application will stop running when pressing the `stop` button. After pressing the `submission` button, the application will make a prediction. Again, you must choose the right line of codes using either `model_pkl` or `model_rds`. After `prediction` and `prediction_prob` are calculated, the function call corresponding to your file type should be selected to print the values.
Finally, there is the option to explain the model with a bar plot showing feature importances or SHAP values. To display the feature importances of a model, the `plot` variable should be set equal to `"feature_importance"`. Nevertheless, please note that this is not possible for all R or Python models, and an error will appear if not. Also, different code lines should be selected for the pickle file and RDS file. SHAP values can only be plotted if calculated beforehand and exported as a pickle file. The path to this pickle file should be entered in `PATH_SHAP`, and `plot` should be equal to `"shap"`. If it is not possible or desirable to show a plot, just enter any other string in `plot`.
It is already mentioned a couple of times, but when libraries or lines of code are irrelevant to your implementation, they can be removed.

Conclusion

Hopefully, everything is clear with this code walkthrough and the comments in the framework file. If you, somehow, still experience difficulties implementing your model as a web app on your machine, it might be handy to take a look at the `experiments` folder in the repository. There, multiple implementations of machine learning models in both Python and R can be found using the same approach as this framework.

7 Discussion

7.1 Results in Context

In this research paper, it has been noted that there is a gap between the implementation and development of machine learning models in healthcare. Therefore, a general coding framework has been developed that provides a web interface for machine learning models without the use of sensitive data. This framework allows such models to be used as an online risk calculator by either a physician or a patient. The approach simplifies the implementation of predictive models and thus encourages their use in practice.

Considering Table 2 again, this research handles most model types containing a class and probability prediction function. The final application can be used by both doctors and patients. As the use of medical data is avoided, the application is not embedded in EHR systems. Moreover, a TRL of 4 is reached as the approach of the framework is tested on multiple types of models but not validated in a relevant environment. Although these elements match most of those in the last row of Table 2, this research has a different purpose. Where the Streamlit implementations in subsection 4.4 are made to show a specific model, here, the applications are made to provide a general framework for an interactive web app.

7.2 Limitations and Further Research

Taking into account the governance model discussed in Section 4, it is noticeable that the framework tries to add transparency to the model by showing the feature importances or SHAP values. However, the framework does not check the quality of the model's predictions considering the fairness aspect. The output shown in the interface solely depends on the given model. So, when a poor model is given, the web application will also give bad predictions. It could be valuable to somehow check the quality of the model in the framework and output the performance on the web app such that the user is aware of the quality. A simpler solution could be to manually add the performance metrics calculated in the data environment to the web page.

Secondly, no experimenting is done with SHAP values in R. That is because quite a few models provided feature importance values, so there was less need for SHAP values. However, exploring the application of SHAP values for predictive models in R that do not support feature importances is still useful.

Furthermore, the SHAP values and feature importances explain the model but not specific predictions in the web app. It might be worthwhile to look into that issue. During the experiments, it has been tried to export 'explainer' objects from the 'shap' library as pickle files, but without success. Therefore, this research exported calculated SHAP values as pickle files since the training data can't be used in the framework. Nevertheless, it would empower the framework if single predictions in the web app could also be explained. It may be an option to include SHAP values for all the instances in the training data and report the SHAP values of the nearest training point for a new data point (prediction in the web app).

Although the framework works for many models, not every model was successfully implemented. The framework did not succeed in implementing a Cox-PH model. Therefore, looking further into that issue and making the framework work for such a survival analysis technique would be helpful. The wider the framework is applicable, the better.

Also, the framework can only handle pickle and RDS files since Python and R are often used as programming languages for model development. However, it could be extended to more export files from other languages.

Moreover, in this research, an attempt was made to get access to a validated predictive model in healthcare. This concerns a model from academia that is scientifically validated and thus proven to be meaningful. Unfortunately, there was no such model available due to regulatory issues. This is a shame, as no data privacy is violated when only a model is exported. Nevertheless, it does demonstrate that implementing these models doesn't have priority, which is one of the problems this research addresses. Making an application for such a predictive model enables performing a pilot with doctors and patients. Eventually, if the implementation, after some adjustments, pleases the users and meets rules concerning application constraints in healthcare, uploading it to the web would make sense. These steps would bring the TRL of this research to a higher level.

8 Conclusion

As the description of the deployment phase in the CRISP-DM model isn't often performed and therefore not very extensive, this research has focused on the following question to help execute this phase:

> **How to deploy a predictive machine learning model as clinical decision support in general practices considering sensitive population health characteristics?**

To answer this question, multiple experiments have been done with a Streamlit application of several machine learning models in Python, and R. Different types of models trained on different datasets have been tried. The implementations were successful if the model provided a class and probability prediction function.

The focus is on making these web applications work without using sensitive data to avoid privacy-related issues. Therefore, the web apps have an interactive interface where the user should fill in the input features to get a prediction. Access to a pickle or RDS file of the model and information about the preprocessing & naming of the data is only needed. Out of these experiments, a general coding framework emerged. With code and comments, this framework implements a machine learning model as a risk calculator on a local device. Moreover, the framework also gives the option to show a bar plot that provides explainability of the web app's predictive model.

In addition to the framework, a manual is written offering a code walkthrough of the code in the framework. This manual could be a valuable addition to the sixth phase of the CRISP-DM key step, 'plan deployment'. Furthermore, a UML activity diagram matching the steps in the manual is made.

However, although this framework can potentially reduce the gap between implementing and developing machine learning models, we are not quite there yet. Companies involved with data management and researchers should be willing to export their models as a pickle or RDS file. If certain rules get in the way, it may be time to change those, as exporting just the models does not breach the privacy of the patient's data. Otherwise, it is difficult for this framework to make a real difference. Nevertheless, the new approach is a step towards implementing more validated predictive research models quickly into general practices and therefore enabling AI easier in healthcare, which is important because AI can be used as a CDS tool to improve preventive healthcare.

References

Adewunmi, M., Sharma, S. K., Sharma, N., Sushma, N. S., Mounmo, B., et al. (2022). Cancer health disparities drivers with bertopic modelling and pycaret evaluation. *Cancer Health Disparities*, *6*.

Baek, E.-T., Yang, H. J., Kim, S. H., Lee, G. S., Oh, I.-J., Kang, S.-R., & Min, J.-J. (2021). Survival time prediction by integrating cox proportional hazards network and distribution function network. *BMC bioinformatics*, *22*(1), 1–15.

Baier, L., Jöhren, F., & Seebacher, S. (2019). Challenges in the deployment and operation of machine learning in practice. *ECIS*, *1*.

Bradshaw, R. L., Kawamoto, K., Kaphingst, K. A., Kohlmann, W. K., Hess, R., Flynn, M. C., Nanjo, C. J., Warner, P. B., Shi, J., Morgan, K., et al. (2022). Garde: A standards-based clinical decision support platform for identifying population health management cohorts. *Journal of the American Medical Informatics Association*, *29*(5), 928–936.

Cohen, J. P., Cao, T., Viviano, J. D., Huang, C.-W., Fralick, M., Ghassemi, M., Mamdani, M., Greiner, R., & Bengio, Y. (2021). Problems in the deployment of machine-learned models in health care. *CMAJ*, *193*(35), E1391–E1394.

Davenport, T., & Malone, K. (2021). Deployment as a critical business data science discipline.

Del Fiol, G., Kohlmann, W., Bradshaw, R. L., Weir, C. R., Flynn, M., Hess, R., Schiffman, J. D., Nanjo, C., & Kawamoto, K. (2020). Standards-based clinical decision support platform to manage patients who meet guideline-based criteria for genetic evaluation of familial cancer. *JCO Clinical Cancer Informatics*, *4*, 1–9.

Dsouza, V. (2018). *An analysis of housing rental sector in ireland* (Doctoral dissertation).

Ellahham, S., Ellahham, N., & Simsekler, M. C. E. (2020). Application of artificial intelligence in the health care safety context: Opportunities and challenges. *American Journal of Medical Quality*, *35*(4), 341–348.

Groenhof, T., Rittersma, Z., Bots, M., Brandjes, M., Jacobs, J., Grobbee, D., van Solinge, W., Visseren, F., Haitjema, S., Asselbergs, F., et al. (2019). A computerised decision support system for cardiovascular risk management 'live'in the electronic health record environment: Development, validation and implementation—the utrecht cardiovascular cohort initiative. *Netherlands Heart Journal*, *27*, 435–442.

Gruendner, J., Schwachhofer, T., Sippl, P., Wolf, N., Erpenbeck, M., Gulden, C., Kapsner, L. A., Zierk, J., Mate, S., Stürzl, M., et al. (2019). Ketos: Clinical decision support and machine learning as a service–a training and deployment platform based on docker, omop-cdm, and fhir web services. *PloS one*, *14*(10), e0223010.

Jackson, S., Yaqub, M., & Li, C.-X. (2019). The agile deployment of machine learning models in healthcare. *Frontiers in big Data*, 7.

Jung, K., Kashyap, S., Avati, A., Harman, S., Shaw, H., Li, R., Smith, M., Shum, K., Javitz, J., Vetteth, Y., et al. (2021). A framework for making predictive models useful in practice. *Journal of the American Medical Informatics Association*, *28*(6), 1149–1158.

Kashyap, S., Morse, K. E., Patel, B., & Shah, N. H. (2021). A survey of extant organizational and computational setups for deploying predictive models in health systems. *Journal of the American Medical Informatics Association*, *28*(11), 2445–2450.

Khalilia, M., Choi, M., Henderson, A., Iyengar, S., Braunstein, M., & Sun, J. (2015). Clinical predictive modeling development and deployment through fhir web services. *AMIA Annual Symposium Proceedings*, *2015*, 717.

Lancaster, G., Kariger, P., McCray, G., Janus, M., Gladstone, M., Cavallera, V., Dua, T., Alves, C. R., Rasheed, M., & Senganimalunje, L. (2020). Conducting a feasibility study in a global health setting for constructing a caregiver-reported measurement tool: An example in infant and young child development.

Motaib, I., Aitlahbib, F., Fadil, A., Tlemcani, F. Z. R., Elamari, S., Laidi, S., & Chadli, A. (2022). Predicting poor glycemic control during ramadan among non-fasting patients with diabetes using artificial intelligence based machine learning models. *Diabetes Research and Clinical Practice*, *190*, 109982.

Nijman, S. W. J., Groenhof, T. K. J., Hoogland, J., Bots, M. L., Brandjes, M., Jacobs, J. J., Asselbergs, F. W., Moons, K. G., & Debray, T. P. (2021). Real-time imputation of missing predictor values improved the application of prediction models in daily practice. *Journal of clinical epidemiology*, *134*, 22–34.

Ramamurthy, K. N., Wei, D., Ray, E., Singh, M., Iyengar, V., Katz-Rogozhnikov, D., Yang, J., Tran, K. N., & Yuen-Reed, G. (2017). A configurable, big data system for on-demand healthcare cost prediction. *2017 IEEE International Conference on Big Data (Big Data)*, 1524–1533.

Reddy, S., Allan, S., Coghlan, S., & Cooper, P. (2020). A governance model for the application of ai in health care. *Journal of the American Medical Informatics Association*, *27*(3), 491–497.

Score2 risk prediction algorithms: New models to estimate 10-year risk of cardiovascular disease in europe. (2021). *European Heart Journal*, *42*(25), 2439–2454.

Shearer, C. (2000). The crisp-dm model: The new blueprint for data mining. *Journal of data warehousing*, *5*(4), 13–22.

Spruit, M., & de Vries, N. (2021). Self-service data science for adverse event prediction in electronic healthcare records. *Research and Innovation Forum 2020: Disruptive Technologies in Times of Change*, 517–535.

Spruit, M. (2022). Translationele datawetenschap in populatiegerichte zorg (inaugural lecture).

van Buuren, S., Bezemer, R. A., Reijneveld, S. A., et al. (2015). Predictie van groei vanaf jonge leeftijd:'curve matching'met de tno groeivoorspelle. *Nederlands Tijdschrift voor Geneeskunde*, *159*.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 1–10.

Yao, Z., Mao, C., Ke, Z., & Xu, Y. (2022). An explainable machine learning model for predicting the outcome of ischemic stroke after mechanical thrombectomy. *Journal of NeuroInterventional Surgery*.

Yoo, J., Lee, J., Min, J. Y., Choi, S. W., Kwon, J.-m., Cho, I., Lim, C., Choi, M. Y., & Cha, W. C. (2022). Development of an interoperable and easily transferable clinical decision support system deployment platform: System design and development study. *Journal of Medical Internet Research*, *24*(7), e37928.

Zhong, H., Ruan, Z., Yan, C., Lv, Z., Zheng, X., Goh, L.-Y., Xi, J., Song, J., Luo, L., Chu, L., et al. (2023). Short-term outcome prediction for myasthenia gravis: An explainable machine learning model. *Therapeutic Advances in Neurological Disorders*, *16*, 17562864231154976.

A Input Possibilities

Input possibilities are the input options a user can enter for a variable. Figure 11 shows the four input possibilities of the variable 'BMI category'. Since retrieving these options from the original dataset is not possible, they should be manually added. Furthermore, the input possibilities can be put into a data frame and merged with the input data frame. In Figure 12, part of a merged data frame is shown. The first row contains a user's input, and the other rows contain the input possibilities of the variables. Merging the data frames can be helpful when using a label encoder from the 'scikit-learn' library in Python. The label encoder transforms the values in the column of a data frame to numbers between 0 and (input possibilities in the column $-$ 1)[27]. Thus, for the input possibilities in Figure 11 this would be between 0 and $(4-1)=3$.

However, only the input data frame is insufficient to encode this correctly. In that case, the label encoder sees only one value in the column and will convert it to 0 regardless of its label. This is unwanted because every input will be transformed into 0, and there will be no difference between the various input options. For example, every input possibility from the 'BMI category' should have the same unique number every time: Normal weight$\rightarrow$0, Overweight$\rightarrow$1, Obese$\rightarrow$2 & Underweight$\rightarrow$3. This is only possible if the label encoder sees all the input possibilities in the columns of the data frame.

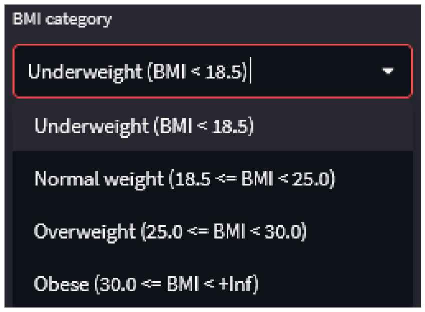
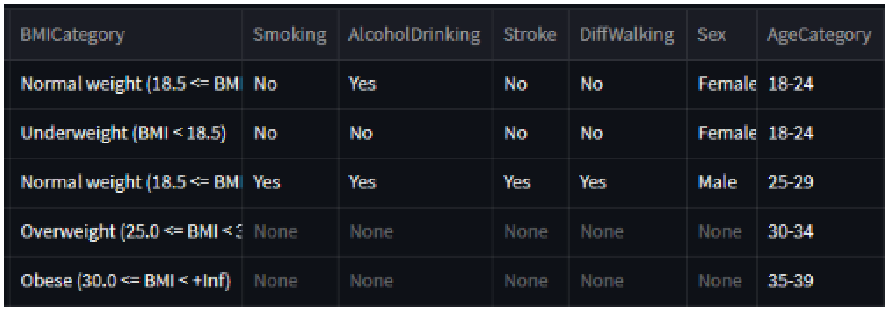

BMI category
Underweight (BMI < 18.5)
Underweight (BMI < 18.5)
Normal weight (18.5 <= BMI < 25.0)
Overweight (25.0 <= BMI < 30.0)
Obese (30.0 <= BMI < +Inf)

Figure 11: Input possibilities of the variable 'BMI category' in the first experiment



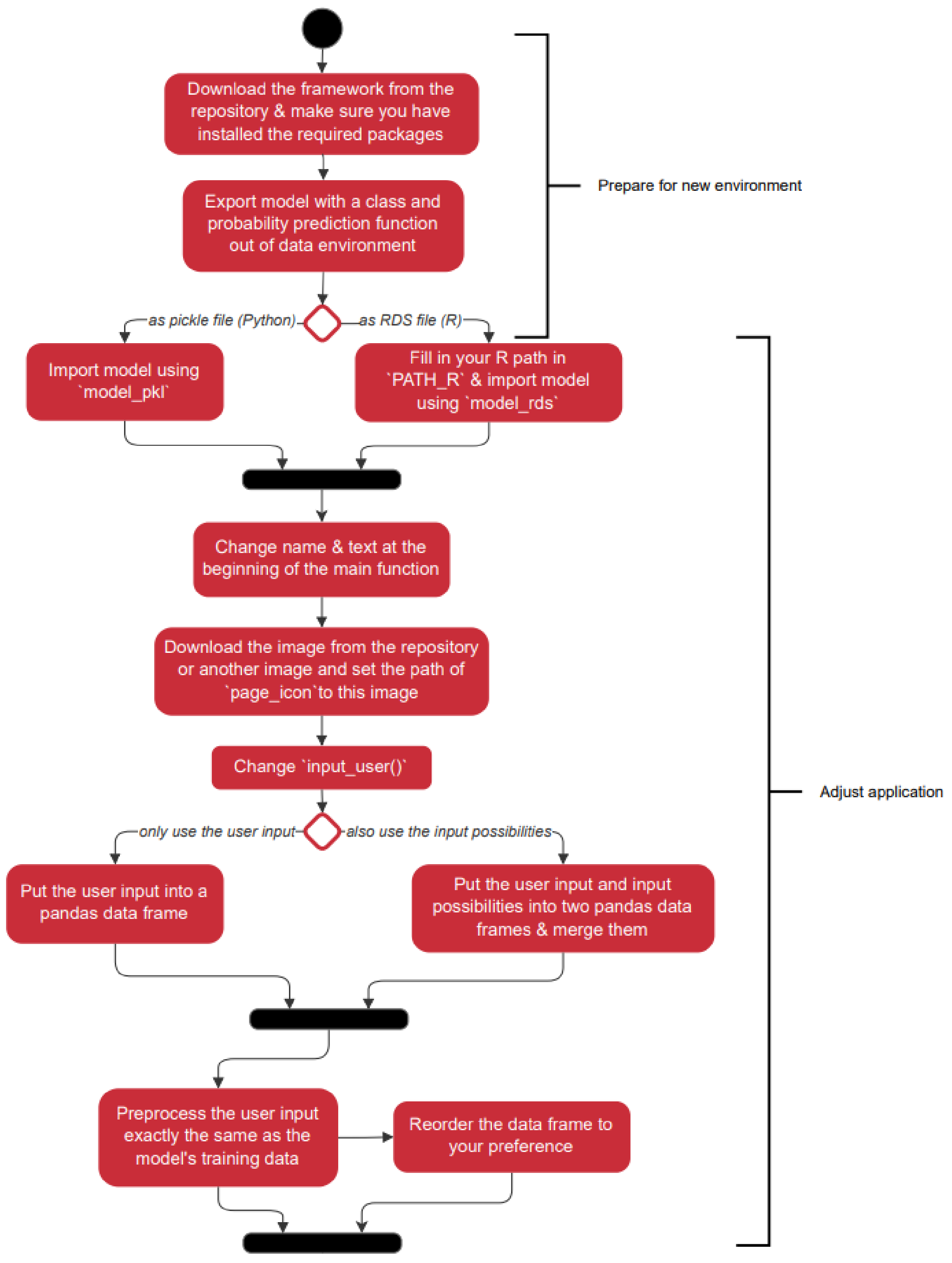
BMICategory	Smoking	AlcoholDrinking	Stroke	DiffWalking	Sex	AgeCategory
Normal weight (18.5 <= BMI	No	Yes	No	No	Female	18-24
Underweight (BMI < 18.5)	No	No	No	No	Female	18-24
Normal weight (18.5 <= BMI	Yes	Yes	Yes	Yes	Male	25-29
Overweight (25.0 <= BMI < 3	None	None	None	None	None	30-34
Obese (30.0 <= BMI < +Inf)	None	None	None	None	None	35-39

Figure 12: Part of a data frame containing the input of a user and the input possibilities

[27]Source: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

B UML Activity Diagram

Figure 13 & Figure 14 show a UML activity diagram that visualises the process of deploying a machine learning model with the framework.



Download the framework from the repository & make sure you have installed the required packages
Export model with a class and probability prediction function out of data environment
Prepare for new environment
as pickle file (Python)
as RDS file (R)
Import model using `model_pkl`
Fill in your R path in `PATH_R` & import model using `model_rds`
Change name & text at the beginning of the main function
Download the image from the repository or another image and set the path of `page_icon`to this image
Change `input_user()`
Adjust application
only use the user input
also use the input possibilities
Put the user input into a pandas data frame
Put the user input and input possibilities into two pandas data frames & merge them
Preprocess the user input exactly the same as the model's training data
Reorder the data frame to your preference

Figure 13: First part of the diagram

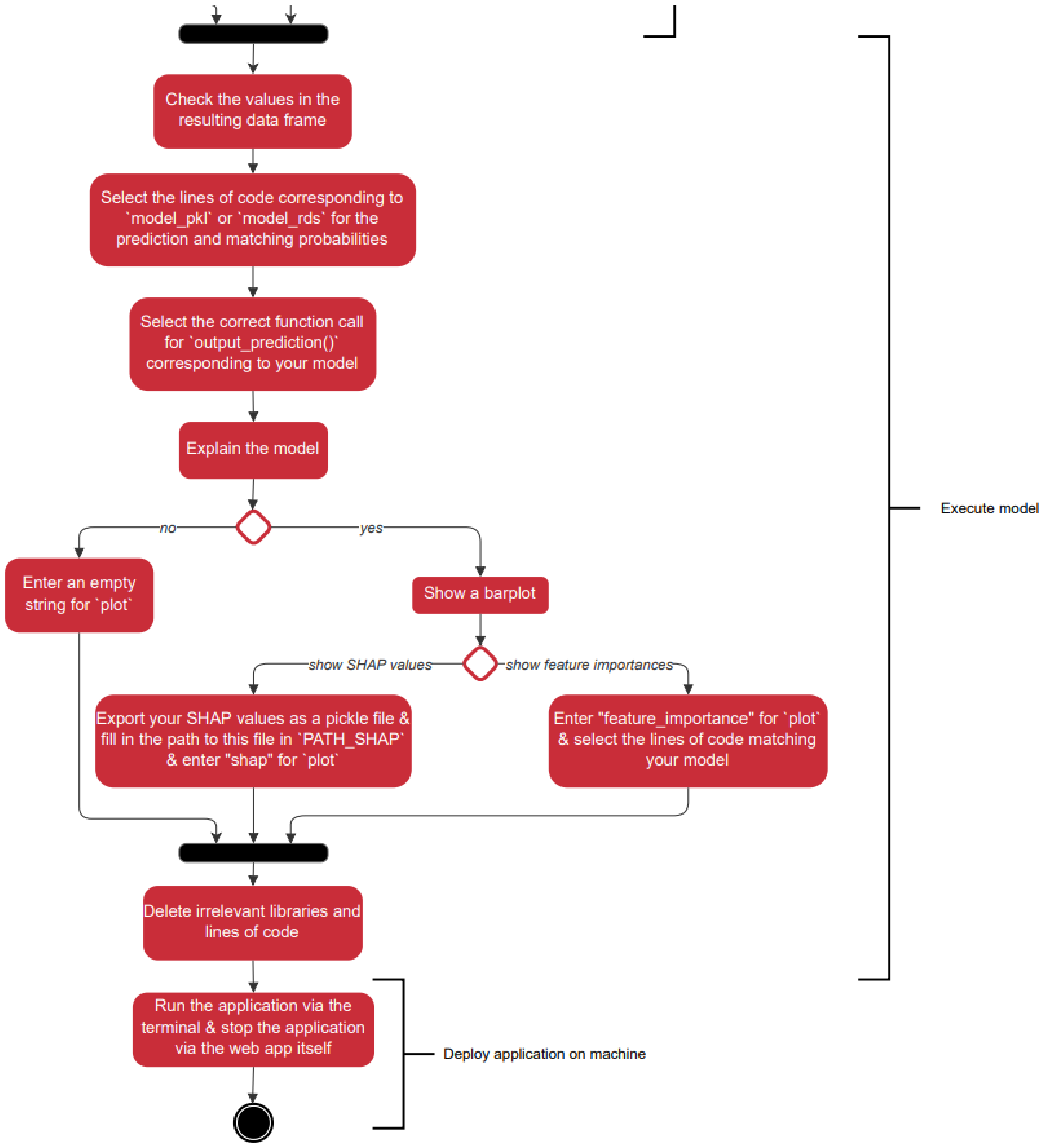

Check the values in the resulting data frame
Select the lines of code corresponding to `model_pkl` or `model_rds` for the prediction and matching probabilities
Select the correct function call for `output_prediction()` corresponding to your model
Explain the model
no
yes
Enter an empty string for `plot`
Show a barplot
show SHAP values
show feature importances
Export your SHAP values as a pickle file & fill in the path to this file in `PATH_SHAP` & enter "shap" for `plot`
Enter "feature_importance" for `plot` & select the lines of code matching your model
Delete irrelevant libraries and lines of code
Run the application via the terminal & stop the application via the web app itself
Execute model
Deploy application on machine

Figure 14: Second part of the diagram