# Bachelorarbeit

## Converting computational notebooks to production-ready code


Verfasserin ODER Verfasser

Lukas Kleinl


angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2022

| | |
|---|---|
| Studienkennzahl lt. Studienblatt: | A 033 521 |
| Fachrichtung: | Allgemeine Informatik |
| Betreuerin / Betreuer: | Stephen John Warnett, BSc (Hons) MSc |

**Abstract**

Machine learning is becoming more and more important in today's world. It is used in almost all areas, for instance such as finance, healthcare or automotive driving. In addition, the rising importance of cloud computing and big data management increases this development. Of course, this also makes it increasingly important how these decisions are made. Therefore, it is important that the code is traceable and reproducible. Machine learning brings new challenges for the architecture of systems based on this technology. The goal of this thesis is to show problems of the current architecture of machine learning based systems and to propose a solution to tackle these problems.

# Contents

# List of Figures

# 1 Introduction

Machine learning (ML) is becoming more and more important in today's world. It is used in almost all areas, for instance, finance, healthcare, or automotive driving. In addition, the rising importance of cloud computing and big data management increases this development. This development can also be seen in the number of publications. The number has risen drastically since 2016 and therefore underlines the growing importance of this topic.[25]

## 1.1 Problem statement

With the growing importance of machine learning, as shown above, the aspect of how to develop such systems is getting more attention. In data science, computational notebooks are the go-to tool for developing machine learning models. Computational notebooks offer many possibilities to bring scientific computing to a new level. Computational notebooks let their user document their thought process and display it. When someone reads the code later, the program is easier to understand, and it is easier to see what the programmer was trying to solve. Furthermore, computational notebooks can be used to create tutorials (e.g. by lecturers for students), since principles and concepts can be presented. One of the largest areas where computational notebooks are used is data science. The most popular notebook in this field is Jupyter, because they offer a way for a scientist to complement their code with analysis, hypotheses, and conjecture.[31] While the use of computational notebooks offers us many advantages, there are also some disadvantages. In a study by Pimente et al. [32], the authors explored a large corpus of computational notebooks to gain insight into the reproducibility of these notebooks. They found that out of 863,878 attempted executions, only 24.11% were executed without error and only 4.03% produced the same results.[32] This shows one of the problems of computational notebooks. A lot of times the data scientists just start programming and explore the data. Thereby they completely neglect the aspect of architecture and coding practices. The lack of code quality often leads to messes in the notebooks. Based on this mess in the notebook, some cells may get neglected because all the cells of the notebook are not always executed. Hidden states can therefore occur.[32] Getting such codes into production is quite difficult and needs a lot of time and work to set up an infrastructure for deploying these models. In some cases, companies that can afford to do so are trying to integrate notebooks into their production process. One example is Netflix. The infrastructure is visible in Figure 1

However, there is no existing solution that combines the best of both worlds, the classical approach with Python[36] scripts and the new approach of integrating computational notebooks into the production workflow. Therefore there are two sides to this. One side tries to extend the notebooks and research new concepts to eliminate the problems, the other side says that notebooks are not suitable for production and write Python scripts based on the results. An example of this process is the guideline of Microsoft Azure.[23] However, a lot of

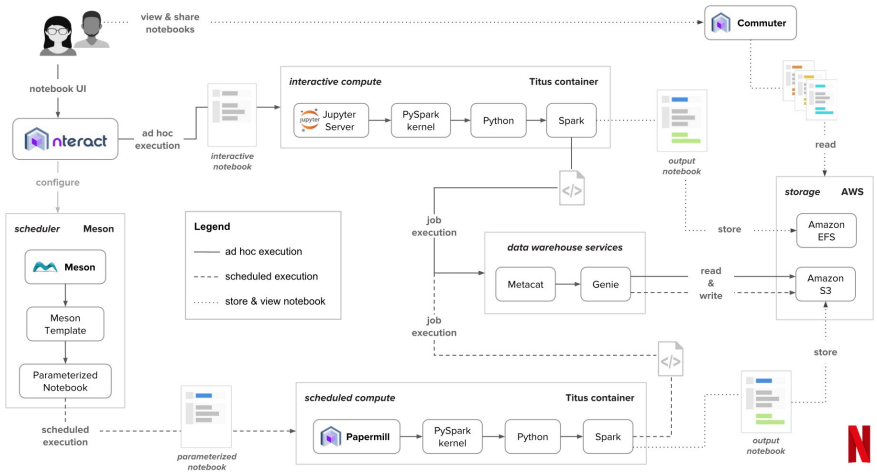


Figure 1: Notebook Infrastructure at Netflix [22]

work has to be done by hand. Alternatives to this process are very rare. For this reason the goal of this work is to create a way to simplify this process.

## 1.2 Goals of the work

As shown before, Jupyter Notebook brings up many problems. Especially in terms of good coding practices and software architecture. Therefore, this thesis tries an approach to convert various notebooks into python scripts and thus provide a solid infrastructure for the development and operation of machine learning models. The tool should therefore be a modular solution that is stable, robust, and easy to follow. With this approach, it should also be possible to automate testing, perform continuous integration, and offers a possibility to track the metrics of the machine learning models. To evaluate the proposed solution, a short quantitative analysis is performed first, where the results of the prototype are compared with the results in the respective Jupyter Notebooks. After that, a qualitative analysis follows, where different points, such as functionality, maintainability, or reliability are examined. Furthermore, the ease with which notebooks can be integrated or converted will be investigated, as well as a comparison with existing solutions.

## 1.3 Methodology

The research method used in this work is called "design science".[3] In this method, knowledge is created with the help of the systematic creation of artifacts. Hevner and Chatterjee define design science research in their book "Design Research in Information Systems: Theory and Practice"[3] as follows: "De-

sign science research is a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem."[3] Thus, in this research method, a solution is sought on the basis of a real problem, which is of interest for practice. To arrive at a solution, one goes through a cycle during the project which is mostly composed of the following phases [35], which can also be seen in Figure 2

- **Awareness of Problem:** The perception of the problem is based on different resources, such as previous research or new developments in the industry. The output of this step is a Proposal [35]
- **Suggestion:** "is a creative step wherein new functionality is envisioned based on a novel configuration of either existing or new and existing elements."[35] The output of this step is a tentative design, which is closely linked to the previous output, the proposal. [35]
- **Development:** In this step the output from the previous step, the tentative design is further developed and implemented and thus the artifact is created as output. [35]
- **Evaluation:** The developed artifact from the development phase is getting evaluated based on the implicit or explicit mentioned criteria. [35]
- **Conclusion:** This step can either be the end of a research cycle or the finale of a research effort. Therefore, the results are assessed as sufficient or insufficient, as well as written down. [35]

## 1.4 Contributions

The main contributions of this bachelor thesis are the following:

- It analyzes existing solutions to convert Jupyter Notebooks and point out problems of computational notebooks with a focus on Jupyter Notebooks.
- Based on these results, a tool will be created that addresses these problems and presents a solution approach. With this tool, it should be possible to convert existing Jupyter Notebooks to production-ready code. Thus, an important contribution to the creation of machine learning projects shall be delivered.
- Furthermore, the quality of the solution approach is evaluated and attempts are made to convert various Jupyter Notebooks.

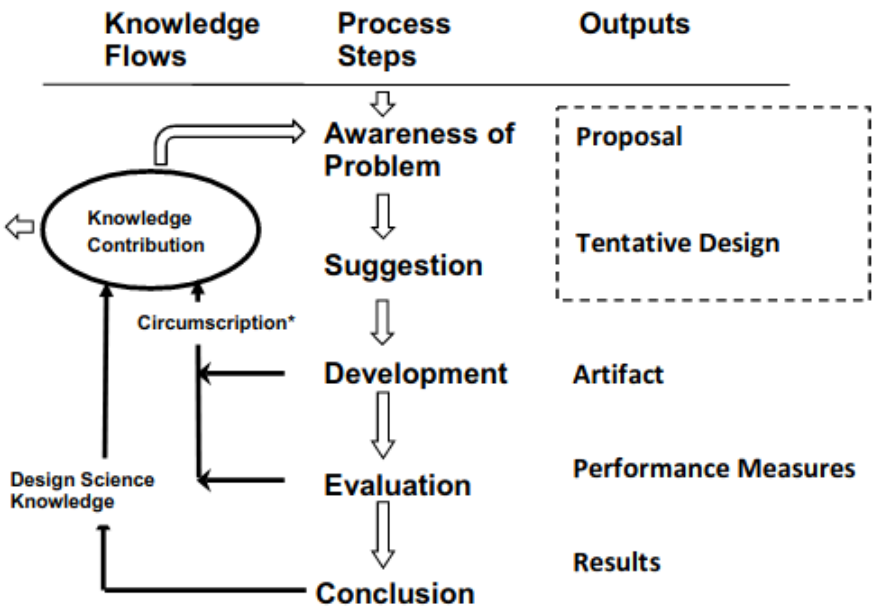


Figure 2: Research Design Cycle from Vaishnavi and Kuechler, 2004[35]

## 1.5 Structure of the work

The subsequent chapter 2 consists of a description of the essential concepts and tools, as well as a literature review, which includes the state of the art and an overview of existing tools.

Chapter 3 sheds light on the design for the possible solution, including a brief requirement analysis.

Chapter 4 deals with a detailed description of the prototype, as well as the respective design decisions.

Chapter 5 covers an internal and an external evaluation of the proposed solution.

The remaining chapter 6 covers the conclusion including limitations and open challenges of the work, as well as a prospect for future work.

# 2 Related work

In this chapter, the central concepts and principles are briefly explained and an overview of the state of the art is given. Finally, similar approaches are discussed

## 2.1 Computational Notebooks

Donald Knuth's work "Literate Programming" introduces the world of computer science to the connection between computer-readable code and human-readable code. In comparison to the existing approach of writing the code for the machine so that it can be easily interpreted by the computer, to a view that understands the program as literature.[18] Thus writes the author: "I believe that the time is ripe for significantly better documentation of programs and that we can best achieve this by considering programs to be works of literature." [18] Since 1984, when Donald Knuth's work appeared, this approach has continued to develop. Especially in the last 20 years, this topic has received an enormous boost with the emergence of computational notebooks, particularly in the field of data science. In 2014 Millman and Perez proposed their approach to "literate computing" with their work "Developing open-source scientific practice".[24] Their view is not about writing code, but about the complete environment. They propose an interactive environment where code can be executed immediately, results are immediately visible, and code can be continued to build upon.[24] Back in 2011, the web-based notebook IPython was developed, which eventually became Project Jupyter. This computing environment is one of the most widely used computational notebooks. Other widely used notebooks are for example Databricks Notebook [9], R Notebook [40] or Mathematica. [39]

Computational notebooks are so widely used for a reason. They can put code into a form that is easy for humans to understand. In the conclusion of "Exploration and Explanation in Computational Notebooks" by Rule, Tabard, and Hollan, the authors write: "Computational notebooks address many fundamental challenges with performing, documenting, and sharing data analyses. They support incremental and iterative analyses, enabling users to edit, arrange, and execute small blocks of code in any order. They enable the explanation of thought processes by allowing analysts to intersperse code with richly formatted textual explanations. They facilitate sharing by combining code, visualizations, and text in a single document that can be posted online or emailed. Some computational notebooks are truly remarkable in the way they elegantly explain complex analyses."[34]

## 2.2 Machine Learning

Machine learning belongs to the intersection of cybernetics and computer science. It has recently attracted overwhelming interest, both from experts and the general public.[12] Machine learning is also of particular importance in the field of data science.

Especially in recent years, the literature on machine learning is piling up, which can also be seen in the number of publications. From 2016 to 2019, the number of publications has increased sixfold.[25] In the year 2019, Nascimento et al. [25] conducted a literature review where they examined how the development of Artificial Intelligence and Machine learning software was structured. Based on this obtained data, they identified challenges and practices, which they finally evaluated. They found out that most of the problems were in the area of testing, artificial intelligence software quality, and data management.

The problem with testing machine learning models is that in computational notebooks it is only possible to test code in a roundabout way. In 2020, Riccio et al. [33] conducted a systematic mapping that analyzes existing solutions for testing machine learning-based systems and identifies open challenges. They focus on three perspectives:

- Context of the problem
- Their features
- Empirical evaluation

The biggest open questions are in the area of realistic inputs and reliable evaluation metrics and benchmarks.

Another important area of research is how to get code ready for production and deployment. Lanubile et al. [21] and Baier et al. [4] researched in this area and present the challenges organizations are facing by developing machine learning-based systems.

Uchida and O'Leary[26] worked with over 100 participants from diverse subfields of data science to identify issues in creating machine learning pipelines. They were able to identify three major problems:

- The environment for prototyping ML models should be designed to prevent the need to re-implement from scratch for production. [26]
- ML pipelines should provide a framework of pre-defined canonical units of operations as components such that ML code can follow ML engineering best practices, as opposed to free-form flexibility. [26]
- Interfaces between components—both code and data—should be made explicit and simple enough so that implementing such an interface is easy to use for ML code authors. [26]

## 2.3 Similar approaches

Geoffrey Hung [16] developed a little tool for an article on "towards data science"[15] where he converts a Jupyter Notebook to a Python script and in the end, he can also make predictions with it. With his article, he wants to show that Jupyter Notebook is not the ideal tool for developing machine learning models. Especially for production scripts are way better. The problem with

this approach is, that you need to do a lot of manual work and it can only be reused for projects with a similar setup.

Another guideline to convert Jupyter Notebooks to production-ready code is written by Microsoft Azure. [23] In this guideline the Jupyter Notebook has to be refactored until it has a satisfactory structure and after that, a tool named nbconvert, is used to convert the code from Jupyter Notebook to Python scripts. If you have to do this for more notebooks, this can become quite exhausting.

The tool that comes closest to this problem is Soorgeon from Ploomber. This tool converts existing notebooks into maintainable Ploomber pipelines. [6] Soorgeon is a part of the larger Ploomber project. This project seeks to turn hard-to-maintain notebooks into collaborative, production-ready pipelines. This is intended to overcome the error-prone process of maintenance. [5]

# 3 Design

To address the aforementioned problems, this chapter discusses how the issues which arise through the use of computational notebooks can be overcome. First, a brief overview of the workflow of machine learning systems and Jupyter Notebooks will be explained. Subsequently, the use case for the implementation is outlined. After that, the decisions for converting Jupyter Notebooks are discussed. Based on these decisions a system was designed. UML (Unified modeling language)[27] diagrams were created to document and illustrate these considerations. To describe this system, Kruchten's 4+1 model was used. Therefore, in the following section first, this architecture model is explained, and then the created UML diagrams.

## 3.1 Introduction machine learning and Jupyter Notebooks

In Figure 3 you can see a rather high-level flowchart of a typical life cycle to create a machine learning model. It starts with getting the data from an external source. Subsequently, this data set is modified so that we can make statements about these data as accurately as possible with the help of an algorithm. Typical steps, which can vary depending on different problems, are: Cleaning the data, extracting features, and transforming the data. After the preprocessing of the data is completed, the model training is continued. The goal of the training is to find a set of weights and biases that have a low average loss across all examples and thus predict the data as well as possible. After training is complete, based on the previously created training and test data sets, the model is evaluated. For this purpose, different metrics can be used, depending on the use case. Frequently used metrics are for example accuracy, F1-score, or the confusion matrix. If the result of the predictions based on the test data sets is satisfactory, the model can be used to predict real data. Depending on the architecture of the system, the model is integrated differently. For example, it could be integrated into a program or a separate API could be created to which data is sent and predictions are made based on this data. Depending on the architecture of the system, the point serve model can therefore vary. The last point in the flowchart is predicted from input data. For this purpose, the model is fed with a data set and makes predictions about this data set based on the previously learned data. Depending on how good the model is, the statements about the given data set are correct.

To narrow down the steps of this whole process even further, it can basically be broken down into five points. These are as follows:

- Loading the data
- Preprocessing the data
- Creating the model
- Evaluating the model

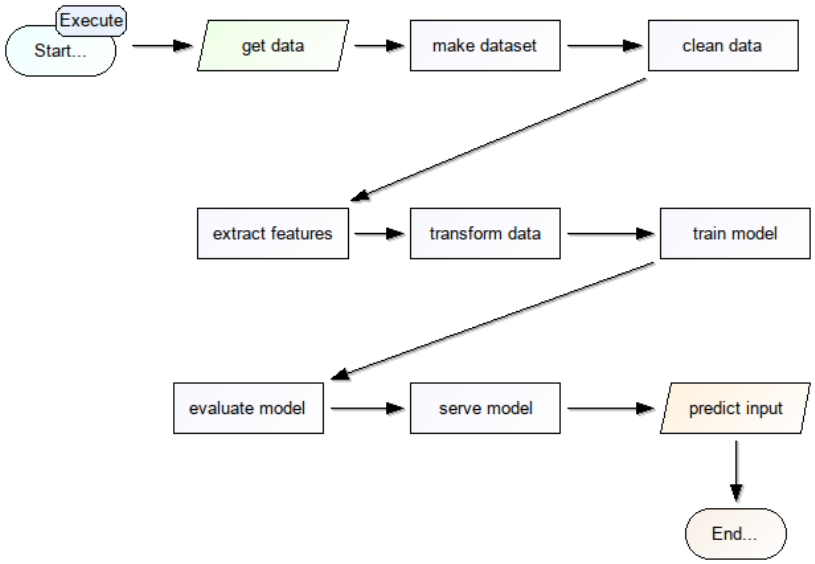


Figure 3: High-level flow chart of typical machine learning workflow

- Serving the model and making predictions

Typically, a data scientist deals with the first 4 points. This means that the task of the data scientist is to analyze the data and to develop a model based on this data, which makes it possible to represent the data as accurately as possible. These steps can also be found in Jupyter Notebooks and therefore the focus of this work is on these points.

## 3.2 Use case for the implementation

Extraction and conversion of computational notebooks could be used in many fields. In this work, however, the area of data science, in particular machine learning, is of great interest. For this purpose, a use case was created. The use case is now briefly described here and can also be seen as the goal of this work.

A Data Scientist is working in a university and researching in his field with machine learning. The scientist runs all his experiments in Jupyter Notebook. Since no infrastructure in his working environment allows him to integrate Jupyter Notebooks into the production process, he has to create a new project from nothing and incorporate his results from the experiments. Therefore, a tool should be developed, which makes it possible to convert these notebooks automatically and to integrate them into a solid infrastructure. It is also important that the project includes automatic testing which ensures the quality of the implementation. Furthermore, it should be possible for several scientists

to work on the project. Thus, continuous integration should be an integral part of the project.

## 3.3 Structure of Jupyter Notebooks

Before proceeding to the design decisions, it is necessary to explain how Jupyter Notebooks are built. In Jupyter Notebook, it is possible to store content in so-called "cells".[17] A cell can have the following formats:[17]

- **Markdown** - here the desired text can be saved. The contents will be rendered as HTML (HyperText Markup Language). [8]
- **Code cells** - content in these cells will be sent to the kernel and will be executed.
- **Raw cells** - content in these cells will pass through nbconvert unmodified.

In Figure 4 an example for these cells is provided. At the top of the image, you can see markdown text. This text contains the name of the algorithm used in this example. In the next two cells, the content is Python code. At the bottom of the picture, the output of the previous cell can be seen. There is also the possibility, as in the second code cell, to tag the cell.

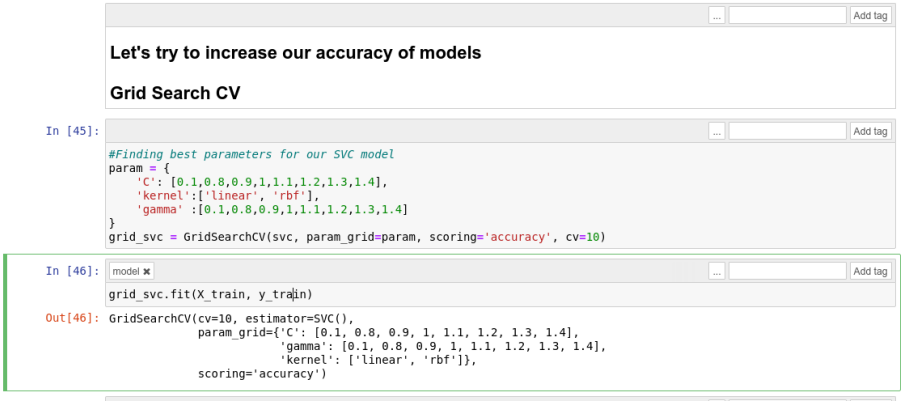


Figure 4: Excerpt from wine prediction notebook[37]

Regarding the conversion of Jupyter Notebook to Python scripts, it is still important to know what the representation of a Jupyter Notebook looks like. Jupyter Notebooks are basically simple JSON (JavaScript Object Notation) documents. JSON "is a lightweight data-interchange format".[28] This is also one of the reasons why version control works so hard with Jupyter, because changes are more difficult to track. To extract information, the most important thing is how the cells are represented in this JSON document. The important contents for the conversion are in "cell type", where the type of the cell is stored, and in "source", where the content of the cell is stored.[17] A sample representation can be seen in Figure 5:

```
{
  "cell_type" : "code",
  "execution_count": 1, # integer or null
  "metadata" : {
      "collapsed" : True, # whether the output of the cell is
collapsed
      "autoscroll": False, # any of true, false or "auto"
  },
  "source" : ["some code"],
  "outputs": [{
      # list of output dicts (described below)
      "output_type": "stream",
      ...
  }],
}
```

Figure 5: Sample representation of a code cell [17]

## 3.4 Conversion of Jupyter Notebooks

In this section, the different possibilities of how to convert Jupyter Notebooks to production-ready code are discussed. To narrow down the scope the focus was solely on notebooks, which implement supervised learning algorithms.

### 3.4.1 Design decisions

The goal of this prototype is, as explained before, to design a system that allows converting experiments from Jupyter Notebooks into code, that follows principles of software engineering. The first question that arises in this task is, "How should the process of converting the notebook be implemented?" After analysis of various approaches, two processes could be filtered out, which sounded promising.

The first consideration was to convert the entire notebook and then test and execute the code. This process could be compared to creating a pipeline. This would solve a problem of Jupyter, namely the problem of hidden states because the complete run of the code prevents this state. A structure of the individual steps could be created with the help of the markdown cells. Each Markdown cell breaks the previous structure, grouping the cells into a logical unit. The problem with this variant is, to identify which code cells belong together and how to finally ensure that the code is executable.

The second variant is based on a different approach. Here the required cells, which are to be converted, are marked. These marked cells are finally converted to Python scripts. The problem with this variant is that in many notebooks no coding best practices are followed and parts of code are not grouped into functions. Therefore some parts of code can spread over several lines of code.

After weighing the pros and cons, the second variant was selected. This variant has the advantage that a modular structure can be developed, which is

making it possible to run various Jupyter Notebooks. Furthermore, this way can also offer a possibility that in the future not only Jupyter Notebooks can be integrated but also simpler and well-structured notebooks can be created based on this implementation.

Subsequently, the next question arises: "Which cells should be converted?" This question can be answered with the previously explained function "tag a cell". The data scientist marks the required cells with a tag and these cells are finally integrated into the program.

## 3.5 4+1 Model

To describe software architecture, author Philippe Kruchten proposes the 4+1 model in his work "Architectural Blueprints-The '4+1' View Model of Software Architecture".[20] This model is intended to support software developers in describing software architectures. The software architecture deals with the design and implementation of programs at a rather high level of abstraction. With the help of these drafts, the architectural elements of the system are to be represented as exactly as possible. Therefore, they show the most important functional and performance requirements of the system. In order to represent the challenge of the software architecture as exact as possible, the author proposes a view, which is based on the following five layers.[20]

- **The process architecture**
  The process architecture takes into account some non-functional requirements, such as load and availability. Furthermore, this view raises the question of how concurrency and distribution can be solved, for example, or how the integrity of the system can be ensured. In this view, the architecture can be described on several levels. At the highest level, the communication between the programs can be described. Several logical networks can share a physical resource. The software is mostly divided into independent tasks. These tasks are located on separate threads, which can be individually scheduled on a processing node. [20]

- **The logical architecture**
  The logical view is mainly there to represent the functional requirements. That means, it should present the services offered to the user with a high degree of abstraction. These abstractions are usually assumed in the form of objects of the problem domain and can be represented using various principles, such as inheritance and encapsulation.[20]

- **The development architecture**
  The development architecture focuses primarily on the actual organization of the software modules in the program. The software is packed into small packages, which can be provided finally by the developers. The development view puts above all value on the internal requirement of the software and pays attention thereby to aspects such as reuse or commonality. Fur-

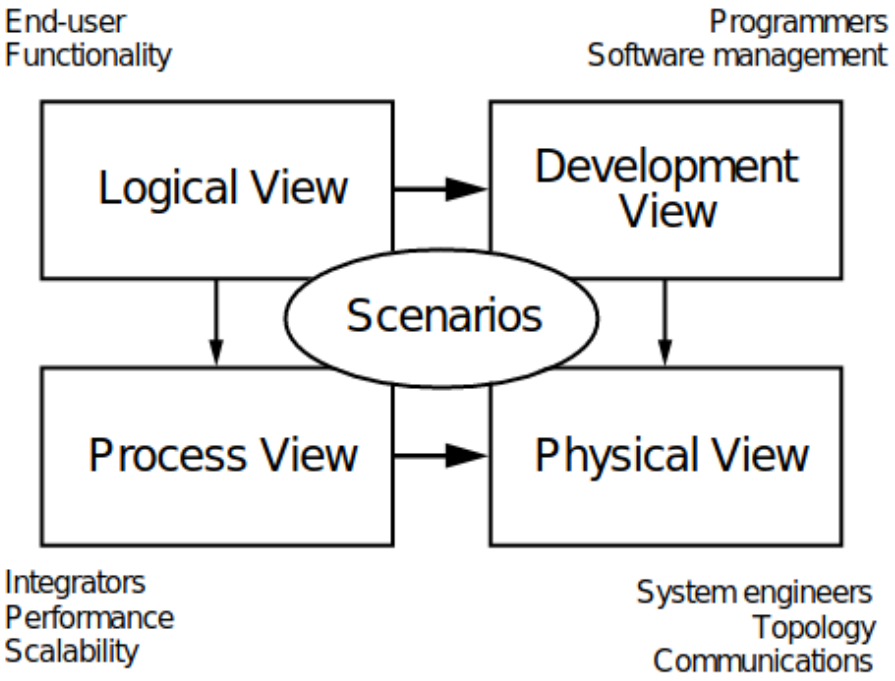


Figure 6: The 4+1 View Model of Architecture from Kruchten Philippe [20]

thermore the development architecture can serve for the division of tasks for the different development teams. [20]

- **The physical architecture**
  The physical view primarily considers the non-functional requirements of a system. Points of particular interest for this view are availability, reliability, scalability, or performance. The identified elements of a system, such as networks, processes, or tasks, are mapped to the different nodes. However, it is important to note that the mapping of the system to the nodes is minimal and has little impact on the source code.[20]

- **Scenarios**
  This view shows how the four different layers work together. This is shown in selected scenarios. In a way, the scenarios are also an abstraction of the requirements for the system. This view is redundant to the other views, which is the reason for the name "4 + 1". [20]

## 3.6 System Design

With the knowledge of the previously explained method for creating the different views of software architecture, the figures shown in the following chapters were created.

### 3.6.1 Process Architecture

An overview of the basic communication flow is shown in Figure 7. As can be

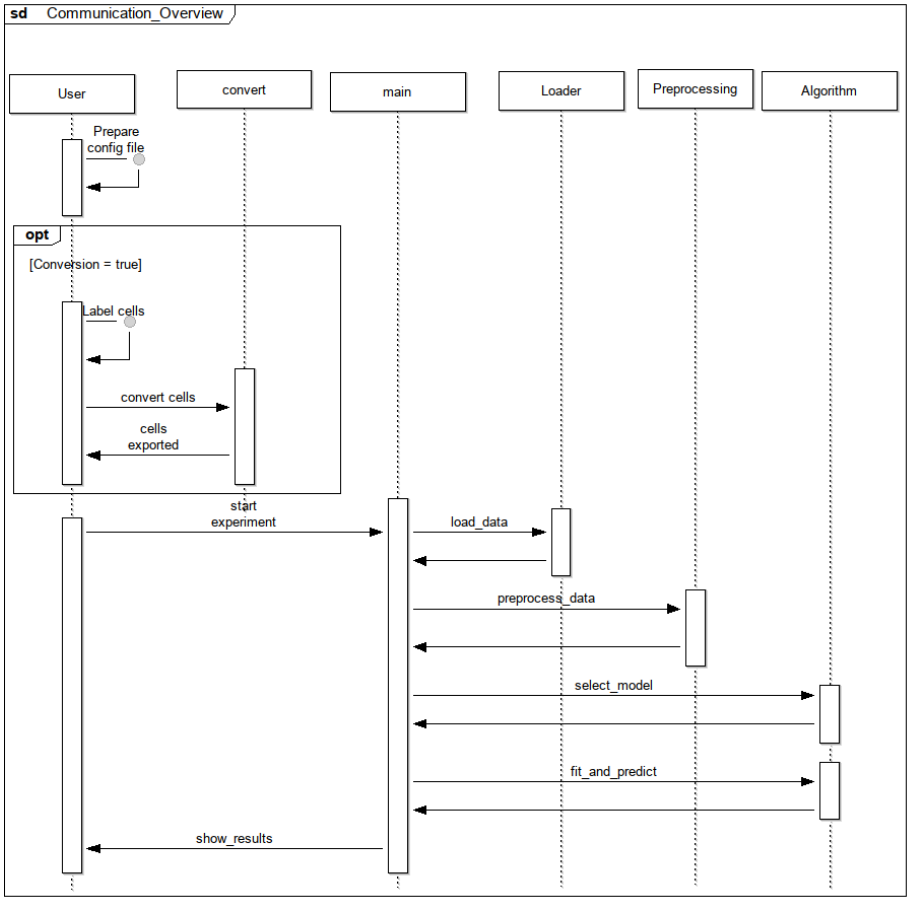


Figure 7: Communication between User and Tool

seen in the diagram, the user first creates a configuration file. Here he already decides if a conversion from a Jupyter Notebook should be performed. If no conversion is to be performed, all parameters must be specified in the configuration file. After completion of the conversion, the program can be started. Afterward, the program communicates with the 3 main components of the prototype, the

loader, the preprocessor, and the algorithm selector. After completion of the process, the results are returned.

### 3.6.2 Logical Architecture

In this chapter, the entity-relationship diagram is displayed, which shows the basic structure of the proposed solution, which can be seen in Figure 8. Further-

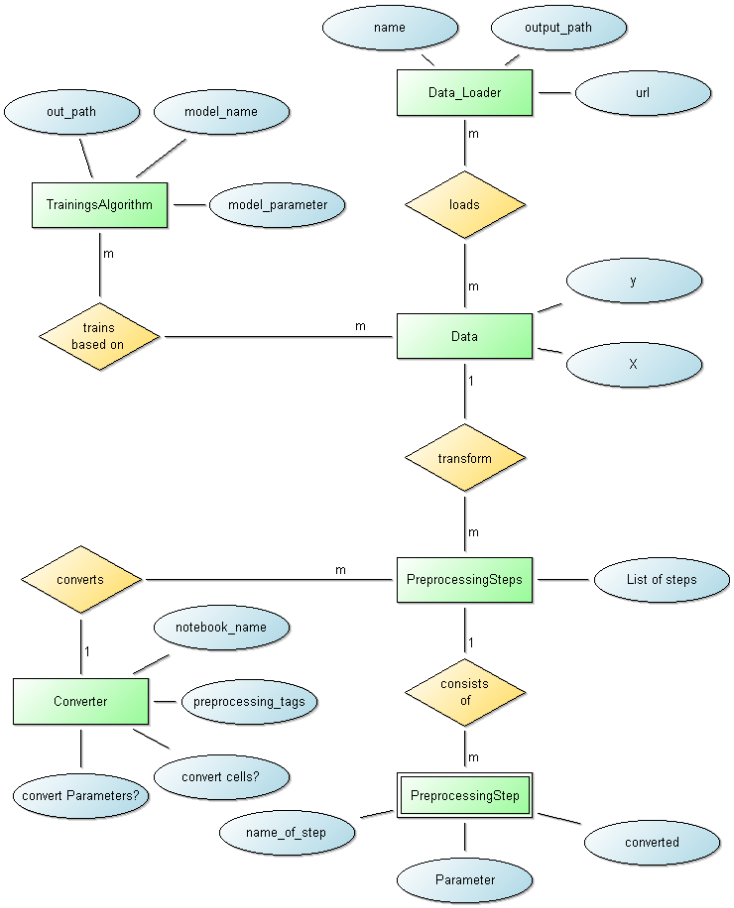


Figure 8: Logical View

more you can see how the different entities relate to each other. Six entities can be identified here. The entity data is in the center. Machine learning projects therefore also represent a new challenge for the system architecture, because in addition to typical problems, the great relevance of data is also added. This data is loaded with the help of the entity loader. The data is modified on the

basis of preprocessing steps. The preprocessing steps consist of several individual steps. The dependency of a preprocessing step on the entity preprocessing steps, which contains all individual steps, is also evident from the fact that the entity preprocessing step is a weak entity. The converter can convert one or more preprocessing steps. Furthermore, it is also possible to convert a loader and a training algorithm. Finally, the algorithm is trained based on the data.

### 3.6.3 Development Architecture

The system overview depicted in Figure 9 presents the structure of the proposed solution. This diagram is particularly important for the developers because the

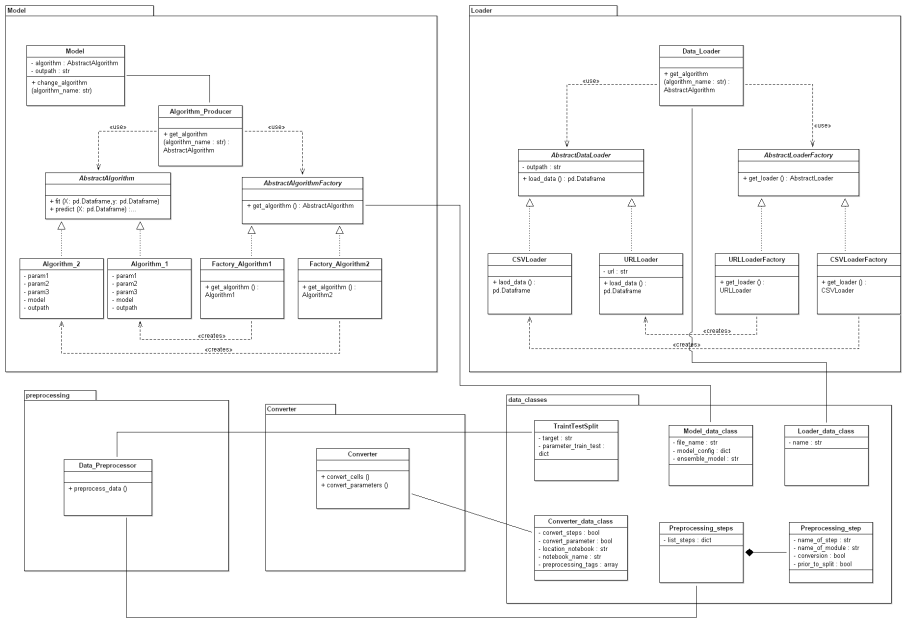


Figure 9: Development View

project is structured based on this diagram. In this diagram, you can see that the project is composed of four main components. These are the "Loader", "Preprocessor", "Algorithm" and "Observer" packages. The Loader and Algorithm packages are each composed of their data class, the realization, and a factory. In the Loader package, two different loaders have been implemented, namely, a CSV (comma separated values) loader, which loads the data from a CSV file, and a URL loader, which allows loading data from a specified URL. Furthermore, it is possible to create a loader from the factory "ConvertedFactory" or from a specified Jupyter Notebook. Similarly, it behaves with the package Algorithm. In this package, however, you will find "Algorithm1" and "Algorithm2" among the provided classes. The reason for this is that any number and variety of algorithms can be implemented here. Due to the focus on

supervised learning, a few example algorithms are implemented here, which can be extended in the future. Here also a factory class was created, which makes it possible to integrate further algorithms into the process. For both packages, the pattern "Abstract Factory" was used.[38] This abstract factory pattern is a creational design pattern that enables the creation of different objects without specifying their concrete classes.[38] For the package "Observer" the "Observer Pattern" was used.[38] With the help of the Observer it is possible to easily log things, update the metrics or output the visualization. The "Preprocessing" package mainly consists of the data class "PreprocessingSteps", where all the unique steps are saved. Based on these steps the data frame is preprocessed.

### 3.6.4 Physical Architecture

In this section, the process view is displayed. This view shows the different nodes, where you can see here that on the basis of the configuration file, the data class components are created. 10

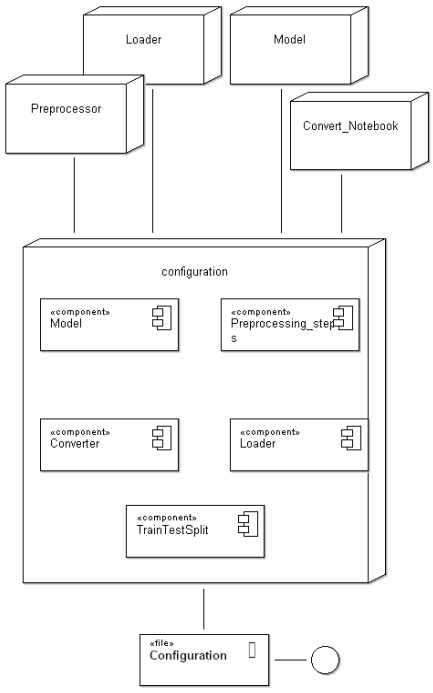


Figure 10: Physical View

### 3.6.5 Scenarios

The scenario view was created based on the use-cases presented in chapter 3.2. Three main use-cases were identified and can also be seen in Figure 11, the abstract factory pattern seemed suitable for this case. Therefore the loader was implemented in the specified way.

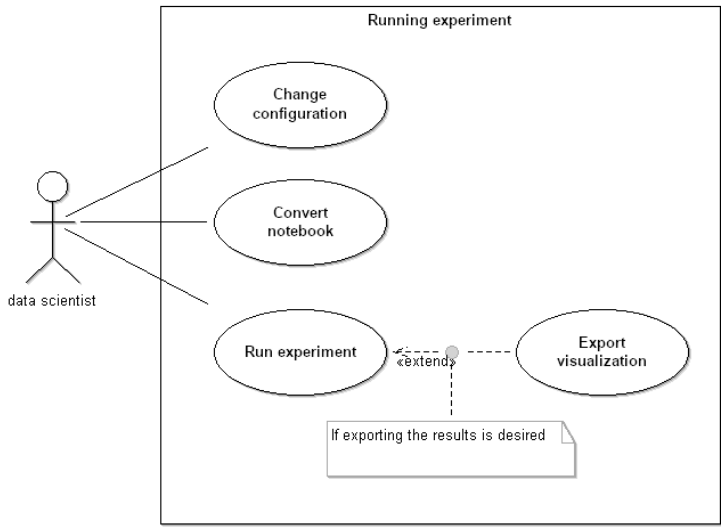


Figure 11: Scenario

- Changing the configuration - It is easily possible to select a different configuration and hence run another experiment or convert another notebook.
- Converting the notebook - The data scientist selects the Jupyter Notebook which should be converted.
- Run an experiment - After the conversion is completed the data scientist runs the project. If the export of the visualization is desired and specified the program outputs the key diagrams.

# 4 Implementation

In section3 the approaches to convert Jupyter Notebooks to production-ready code were discussed. Based on these findings the design, which can be seen in the UML diagrams, was created. This section covers the implementation of the prototype created within this thesis. The prototype has been created with PyCharm 2021.3.2 (Professional Edition). The code was tested using "pytest".[19] The documentation of the different packages of the tool was created using "Sphinx"[7], which can be found in the folder docs. To look at the documentation open the "index.html" file within your browser. The requirements of the system are specified in "requirements.txt" within the project. The code of the prototype is available at: "https://github.com/lukkleinl/BSC_Project". GitHub is a code version control and colaboration tool. [13]

## 4.1 Converting Notebooks to scripts

In order to extract cells from Jupyter Notebook, the metadata of the notebook is used. The implementation within the prototype is shown in Figure 12

```
def convert_cells(path_notebook: str, name_of_tag: str, output_path: str):
    """
    Converts the cells from the given notebook and saves them to the specified location

    :param name_of_tag: Name of the tag which was used within the notebook
    :param path_notebook: location of the notebook
    :param output_path: where to output the converted files
    :return:
    """
    notebook = nbformat.read(path_notebook, as_version=4)
    python_string_to_convert = ""
    for cell in notebook.cells:
        tags = cell["metadata"].get("tags", None)
        if tags == [name_of_tag]:
            python_string_to_convert += cell["source"]
            python_string_to_convert += "\n"

    if not os.path.exists(output_path):
        os.mkdir(output_path)

    f = open(output_path + name_of_tag + ".py", "w")
    f.write(python_string_to_convert)
    f.close()
```

Figure 12: Function for converting cells to python scripts

Therefore, the package nbformat[14] was used to get the data from the specified notebook. Furthermore, tags, which were used to mark cells that should be converted, are needed. Based on the given tags the marked cells are saved within a string and outputted to the specified location. It is possible to convert multiple cells like it is shown in Figure 13. The downside to this approach is that these multiple cells have to be put into one function. Henceforth, these

converted cells have to be refactored to satisfy the necessary format of the preprocessing function. A preprocessing function has to accept the data frame as well as a dictionary, where all the necessary data for this preprocessing step is contained. The refactored function could look like Figure 14. This step could either be done prior to or after the conversion step. If it is done within the notebook, only the refactored function should be tagged.


Figure 13: Logical function spread over more code cells

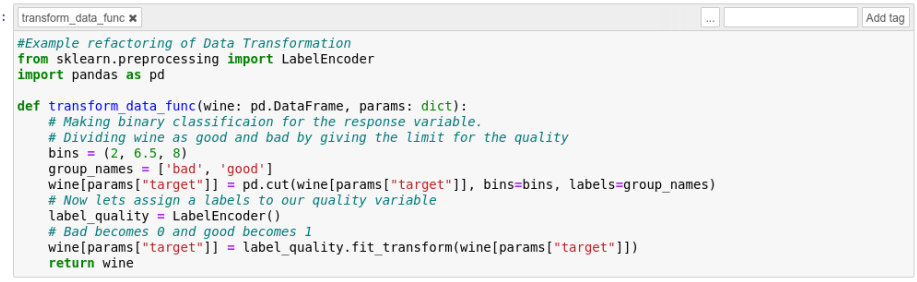


Figure 14: Refactored function to fit the implementation

## 4.2 The architecture of the main workflow

In this section, the main components of the workflow are described.

### 4.2.1 Configuration

The principle behind the configuration is, that an attempt is made to control the entire experiment via the configuration file. Therefore, several configuration files can be created, each of them a new experiment in itself. The configuration files are provided to the program via Click.[29] Click is a Python package, which can create easy-to-use command-line interfaces with just a little portion of code.[29] The usage of Click within the project can be seen in Figure 15.

Here it is possible to see that a configuration file has to be passed as argument to the main function. It is also viable to specify a default value within the click

```
@click.command()
@click.argument("config_path", type=str, default="../data/config_files/config_sgd_convert_functions_loader_CSV_Loader"
                                                 ".yaml")
def main(config_path):
```

Figure 15: Example use of Click in the prototype

configuration, which in this example is the location of a configuration file. A sample configuration file can look like Figure 16.

```
Converter:
  location_of_notebook: "../data/notebooks/Wine_Quality/"
  convert_preprocessing_steps: True
  parameter_conversion: False
  notebook_name: "prediction-of-quality-of-wine.ipynb"
  preprocessing_tags:
    - "transform_data_func"

Preprocessing_Steps:
  transform_data:
    conversion: True
    prior_to_split: True
    name_of_step: "transform_data_func"
    name_of_module: "transform_data_func"
    params:
      target: "quality"

  scale_data:
    conversion: False
    prior_to_split: False
    name_of_step: "scale_data"
    name_of_module: "transform_data"
    params:
      scaler_name: "StandardScaler"

Loader:
  name: "CSVLoader"
  path_to_csv: "../data/Wine_data/raw_data.csv"

TrainTestSplit:
  target: "quality"
  parameter_train_test_split:
    "test_size": 0.2
    "random_state": 42

Model:
  file_name: "rf_model_new.obj"
  ensemble_model: "RandomForestClassifier"
  model_config: {n_estimators: 100}
```

Figure 16: Sample configuration of an experiment

### 4.2.2 Data loader

After the configuration process is complete, the program proceeds to load the data. As already described in chapter 3.6.3, the data loader and the model selection are based on an abstract factory pattern. Figure 17 shows how to create the data, which shall be loaded. First, a loader factory is created. It receives the name of the desired loader, such as "CSVLoader" for instance, which loads the data from the specified path to the CSV file, where the data is stored. If the name is found in the implementations, the desired factory is created. The object of the desired loader is then created with the get_loader function. To do this, the previously created data class loader must be passed

to the function. In this data class all information is stored, which is needed to load the data. Afterwards, the raw version of the data can be loaded with the function create_raw_data. At no point does the program need to know exactly, which implementation of the loader is being used. Therefore, it is easy to extend the implementation with other types of loaders.

```
loader_factory = create_loader(loader.name)
load = loader_factory.get_loader(loader)
try:
    load.create_raw_data()
except AttributeError as err:
    sys.exit(err)
```

Figure 17: Workflow to create loader and load data

### 4.2.3 Data preprocessing

When the raw data is loaded, the preprocessing process can be started. For this purpose, all preprocessing steps are loaded first. Then the steps are executed in the order in which they were defined. The defined preprocessing functions receive a data frame and a dictionary as parameters, in which all parameters required for this step are stored. The functions finally return the new data frame. This process is shown in Figure 18.

```
"""Preprocess Data prior to seperation of features"""
for step in preprocessing_steps:
    if step.prior_to_split:
        df = import_and_load_preprocessing_function(step, df)
        df.to_csv(output_path_preprocessing + step.name_of_step + ".csv")
```

Figure 18: Workflow for preprocessing data

The import and load function shown in this figure, dynamically loads the specified functions. This means that the preprocessing steps specified in the configuration file are automatically loaded and executed by the respective packages. This dynamic loading is pictured in Figure 19.

A CSV file of the modified data frame is created for each preprocessing step. This allows easier tracking of the data state. After the completion of the preprocessing, the final data frame is saved separately.

### 4.2.4 Model selection

When selecting the desired model, the procedure is the same as in the previously explained procedure for creating a new loader. Here, too, the name of the desired model is passed to the factory creator and the new factory is created. With the help of this factory, the desired model is finally created. The program only recognizes the abstract class "BaseAlgorithm", which has the abstract methods

```
"""load module of preprocessing step"""
module = importlib.import_module(location_of_step + name_of_module)

"""tries to load function with given name and execute it"""
try:
    func = module.__getattribute__(name_of_step)
    df = func(df, params)
except AttributeError as err:
    sys.exit(f"specified {err}")

return df
```

Figure 19: Loading specified preprocessing steps dynamically

fit and predict. The exact implementation of the individual models is not clear. If the desired algorithm is not implemented, it is also possible to load a converted or self-made class. This possibility is shown in Figure 20.

```
class ConvertedAlgorithmFactory(AlgorithmFactory):
    """Factory to load a class from conversion dynamically"""

    def get_algorithm(self, model: Model, out_path: str):
        """load module of the specified class"""
        module = importlib.import_module("conversion." + model.name_of_module)

        """tries to load class with given name and execute it"""
        try:
            algorithm_class = module.__getattribute__(model.name_of_class)
            algorithm_class = algorithm_class(model, out_path)
        except AttributeError as err:
            sys.exit(f"specified {err}")

        return algorithm_class
```

Figure 20: Implementation of the factory to dynamically load class

## 4.3 How to use the implemented solution

In order for the program to run at all, there are a few prerequisites. First of all, an environment must be set up where all requirements are installed. Following the description below, it is possible to create an environment with Conda.[2] Conda is an open-source package management and environment system that runs on Windows, macOS, and Linux. With Conda, it is easy to manage the requirements needed by the program.[2] First, a new environment must be created. This can be done with the command shown in Figure 21.

Then, the newly created environment must be activated, as depicted in Figure 22.

Finally, pip[1], which is the Python package installer, must be installed. The

```
conda create --name your_env_name
```

Figure 21: Create Conda environment

```
conda activate your_env_name
```

Figure 22: Activate Conda environment

requirements made by the software are installed with pip. These commands are pictured in Figure 23.

```
python -m pip install --upgrade pip
pip install -r requirements.txt
```

Figure 23: How to install requirements of the prototype

To start a new or existing experiment, the src folder must be the current working directory. If the src folder is your current working directory, you can either convert code from a specific Jupyter Notebook or, if all functions are already implemented, start the experiment immediately.

When converting from a notebook, it is important that the lines of code to be converted are tagged. Then the Python convert.py command can be executed. If a configuration file other than the default configuration file is to be used, this must be passed with the command. The same applies to the implementation of the experiment. This is started with the command main.py. Example commands are shown in Figure 24.

### 4.3.1 Testing and Continious Integration

For testing the code, as already mentioned in the beginning of chapter 4 pytest was used to run the tests. To run the tests, you only need to enter the command pytest in the command line. Testing is a very important task. They can help minimize errors, or at least ensure correctness for the implemented cases.

Testing is also very important for a widely used development practice called Continuous Integration(CI).[11] CI enables the members of a team to merge and integrate their work together, gives companies the possibility to have shorter release cycles and helps in improving code quality.[11] Therefore a CI workflow was integrated as well. To implement a CI flow, a new tool of GitHub called GitHub actions were used.[13] With this tool it is possible to test the code on each push to the repository.

```
python3 convert.py ../data/config_files/Wine_quality/config_sgd_convert_functions_loader_CSV_Loader.yaml
python3 main.py ../data/config_files/Wine_quality/config_sgd_convert_functions_loader_CSV_Loader.yaml
```

Figure 24: Functions to start either conversion or experiment

# 5 Evaluation

After the prototype was developed, two different kind of evaluation, quantitative and qualitative evaluation, were conducted. In the quantitative evaluation the results of the implementation of the prototype and the implementation of the notebooks were compared. The second evaluation consists of two parts, the internal evaluation, and an external evaluation. In the internal evaluation the usability of the implemented tool as well as the relevance was analyzed, whereas in the external evaluation, the prototype was compared to the solutions stated in chapter 2.3. Therefore, the quantitative analysis focuses on the outcome of the experiments and the qualitative analysis focuses on the quality aspects of the whole project.

## 5.1 Quantitative analysis

Before the comparison between the outcome could be conducted, measurements, which should be used, must be specified. For the analysis, the notebook prediction of wine quality from Kumar Vishal[37] and the classical machine learning notebooks support vector machine and random forest tree classifier from Jean de Dieu Nyandwi's GitHub repository[10] were used. These models are all using supervised learning methods. Furthermore, these are all classification problems. Therefore, to compare the results the metric classification report from sklearn[30] was used. The results of this analysis will be discussed in chapter 5.4.

### 5.1.1 Classification report

The classification report is a typical function from scikit-learn[10] for analyzing classification problems. This function builds a text report showing the main metrics of a classification problem. It depicts the metrics precision, recall, and F1 score as well as the macro average, weighted average as well as the sample average (only for multilabel classification).[10] The precision is the ratio between the true positives and the true positives plus the number of false positives. [10] The recall is the ratio between the true positives and the sum of true positives and the number of false negatives. The F1 score is the weighted average of the precision and the recall. [10]

### 5.1.2 Results notebook 1 - wine prediction

For the first experiment, the wine prediction notebook by Kumar Vishal[37] was used. Among other things, a stochastic gradient decent classifier(SGD) was used

for the analysis. This classifier was used for the comparison. When comparing the two classification reports in Figure 25 and Figure 26, it can be seen that the values are similar in some ways, but also show strong differences. This is because there is a high variance in the outcomes when running the experiment with this classifier, and they have a high variability when running the experiment with the same values.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.87 | 0.96 | 0.91 | 273 |
| 1 | 0.40 | 0.17 | 0.24 | 47 |
| accuracy | | | 0.84 | 320 |
| macro avg | 0.64 | 0.56 | 0.57 | 320 |
| weighted avg | 0.80 | 0.84 | 0.81 | 320 |

Figure 25: Classification report SGD from the notebook[10]

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.52 | 0.68 | 273 |
| 1 | 0.26 | 0.98 | 0.41 | 47 |
| accuracy | | | 0.58 | 320 |
| macro avg | 0.63 | 0.75 | 0.54 | 320 |
| weighted avg | 0.89 | 0.58 | 0.64 | 320 |

Figure 26: Classification report SGD from the prototype

### 5.1.3 Results notebook 2 - random forest classifier

For this experiment, the random forest classifier notebook from Jean de Dieu Nyandwi GitHub repository[10], within the folder classical machine learning with scikit-learn was used. In this notebook the use of the random forest classification is depicted. Figure 27 and Figure 28 are picturing the outcome of the notebook and the implemented tool. When compared to each other, it is possible to see only a slight difference between those two reports.

### 5.1.4 Results notebook 3 - support vector machine

The used notebook in this experiment was also created by Jean de Dieu Nyandwi.[10] The focus in this notebook is support vector machines. This algorithm can be used for regression, classification and detecting outliers. Figure 29 and Figure 30 are showing the classification reports again. To predict the outcome, an additional grid search was performed based on top of the support vector classifier.

```
class_report(X_train, forest_best, y_train_prepared)
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.79 | 0.87 | 0.83 | 19541 |
| 1 | 0.80 | 0.68 | 0.74 | 14443 |
| accuracy | | | 0.79 | 33984 |
| macro avg | 0.79 | 0.78 | 0.78 | 33984 |
| weighted avg | 0.79 | 0.79 | 0.79 | 33984 |

Figure 27: Classification report from random forest classification notebook[10]

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.77 | 0.90 | 0.83 | 19541 |
| 1 | 0.83 | 0.63 | 0.71 | 14443 |
| accuracy | | | 0.79 | 33984 |
| macro avg | 0.80 | 0.77 | 0.77 | 33984 |
| weighted avg | 0.79 | 0.79 | 0.78 | 33984 |

Figure 28: Classification report from the prototype

Compared to each other, it is possible to see, that the experiment, which was run with the implemented tool, did overfit the model. This means that this model does not generalize well to new data.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.85 | 1.00 | 0.92 | 11 |
| 2 | 1.00 | 0.82 | 0.90 | 11 |
| accuracy | | | 0.93 | 30 |
| macro avg | 0.95 | 0.94 | 0.94 | 30 |
| weighted avg | 0.94 | 0.93 | 0.93 | 30 |

Figure 29: Classification report from random forest classification notebook[10]

## 5.2 Qualitative analysis

This chapter consists of two parts. First the analysis of the tool in terms of usability and relevance, following a comparison to the existing tools.

### 5.2.1 Internal evaluation

This evaluation consists of two main questions:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0.0 | 1.00 | 1.00 | 1.00 | 8 |
| 1.0 | 1.00 | 1.00 | 1.00 | 11 |
| 2.0 | 1.00 | 1.00 | 1.00 | 11 |
| accuracy | | | 1.00 | 30 |
| macro avg | 1.00 | 1.00 | 1.00 | 30 |
| weighted avg | 1.00 | 1.00 | 1.00 | 30 |

Figure 30: Classification report from the prototype

- To what extent does the conversion made from Jupyter Notebooks to Python scripts make sense?
- How easy is it to create a new experiment?

When working with Juypter Notebooks, it becomes clear that the monolithic structure of the notebook makes it difficult to keep track of the entire experiment. This is especially true when you work with multiple notebooks. Also, when a notebook is to be taken over and modified by someone else. This cannot be done quickly but takes up a large amount of time. With this tool it can be easier to implement solutions from notebooks, which were implemented by someone else. Another possibility that this access offers is that it makes running multiple experiments much easier, since you simply create multiple configuration files and run the experiments based on them. On the other hand, searching for parameters, especially if they are not defined in a cell, can be very difficult in notebooks. This becomes an even bigger factor when you take over the notebook of someone else.

How easy is it to create a new experiment? In the experiments that were conducted with the tool, there were good as well as bad experiences. In particular, experiments using functions and classes that had already been implemented could be carried out very quickly and easily. Above all, the conversion of new loader or model classes was more time-consuming since they must correspond to the class structure. However, changing the preprocessing functions could be carried out more quickly, since some of these are combined in functions.

## 5.3 External evaluation

The main question in this chapter is:

- How well is the tool performing to existing tools?

Geoffrey Hung [16] tool for converting Jupyter Notebooks to Python scripts makes it possible to run similar projects very easily. The problem with this implementation is that it is very hard to adapt to other problems, because

one needs to change the preprocessing steps as well as the data loading and the model selection by hand. Therefore, it is not very reusable. Whereas the approach of the implemented tool is very modular, allowing the project to be easily extended with additional classes and functions without changing the flow of the implementation.

The Microsoft Azure guideline[23] suggests to refactor the code within the notebook first and then convert it to scripts to make it automation and testing friendly. This approach means a lot of manual work for the user, since the entire refactoring of the notebook must be done by hand. However, with the implementation of the tool, this process can be done semi-automatically since the desired cells to be converted can be simply selected. If the converted code is to be implemented in the tool, only tests for the converted functions must be written.

The tool that comes closest to this problem is Soorgeon[6] from Ploomber.[5] This tool converts existing notebooks into maintainable Ploomber pipelines. Compared to Soorgeon, the prototype does not perform as well as before. The advantage of Soorgeon is that this tool is part of the large project Ploomber, which is a tool that allows to quickly create data pipelines. However, Soorgeon also makes demands on the user. For example, users are expected to separate logically related sections with H2 headings in Jupyter. Furthermore, it is not possible to convert notebooks that use globally defined variables. Converting notebooks of this kind can be very tiring. This is also the advantage of the implemented solution. With the help of this tool it is possible to convert cells that are not in the same cell, but still belong together.

## 5.4 Discussion of the results

In the quantitative evaluation it was possible to see that we could reach similar results on the experiments, at least on the notebooks which were tested. Therefore, further evaluation needs to be done in the future. Furthermore, it was obvious that the refactoring of code to fit the specification can be quite cumbersome. It is not possible to do this conversion automatically, especially because steps could already be in a function, but not in the way the solution needs it to be. Therefore, more research must be done on how to accept a bigger variety of functions or to convert them, so they fit the specification. Compared to the other tools, the implemented solution performed fine, but still a lot of work has to be done, like enabling more possibilities for analyzing experiments.

## 5.5 Lessons learned

The biggest problem in this project was to find a way to transform code from the Jupyter Notebooks to meet the requirements set by the implementation. No solution could be found to this problem and therefore parts of code to be converted must be refactored to meet the requirements. This led to the realization that the better approach is not to transform Jupyter to Python scripts, but

rather to create an environment with Python based on which one can eventually create simple notebooks for demonstration and experimentation.

# 6 Conclusion and future work

## 6.1 Summary of the work

This thesis presents a way to convert Jupyter Notebooks to Python scripts so that eventually experiments can be performed based on these scripts. This process is important because best practices are usually neglected in Jupyter Notebooks. Further approaches to this problem are to extend Jupyter in such a way that these problems do not arise as well as to create enlightenment, especially among data scientists, to create well-structured code. Based on this knowledge, a prototype was finally developed, which should make it possible to convert Jupyter Notebooks into modular, well-structured code. After creating the proposed solution, it was evaluated and compared to existing solutions. Based on this analysis, it can be said that this prototype offers a good approach to transform Jupyter Notebooks into well-structured code but it still requires further research, evaluation and development.

## 6.2 Limitations

This work is limited in that it only examined the extent to which supervised learning notebooks using classification algorithms can be converted. Furthermore, the quantitative analysis was only performed on three notebooks. Therefore, the tool needs to be further evaluated.

## 6.3 Future work

As mentioned before, more research is needed in the field of Jupyter Notebook conversion. In particular, how diverse notebooks that rely on different algorithms can be converted and whether the proposed solution can also be used in other machine learning algorithms. This requires further development of the existing solution. If this approach is pursued, further tests by experts from the field of data science are required.

# References


[1] Python package index - pypi.

[2] Anaconda software distribution, 2020.

[3] Alan Hevner, S. C. *Design Research in Information Systems : Theory and Practice*. Integrated Series in Information Systems 22. Springer Science+Business Media, LLC, Boston, MA, 2010.

[4] Baier, L., Jöhren, F., and Seebacher, S. Challenges in the deployment and operation of machine learning in practice.

[5] Blancas, E. Ploomber. GitHub repository: `https://github.com/ploomber/ploomber`, , 22.01.2022, 2022.

[6] Blancas, E. Soorgeon. GitHub repository: `https://github.com/ploomber/soorgeon`, , 22.01.2022, 2022.

[7] Brandl, G. Sphinx documentation. accessed from: `https://www.sphinx-doc.org/en/master/index.html`, 30.01.2021.

[8] community, W. Html standard. accessed from: `https://click.palletsprojects.com/en/8.0.x/`, , 30.01.2022.

[9] Datbricks. Notebook. accessed from: `https://docs.databricks.com/notebooks/index.html`, , 22.01.2022.

[10] de Dieu Nyandwi, J. Classical machine learning notebooks. GitHub repository: `https://github.com/Nyandwi/machine_learning_complete/tree/main/6_classical_machine_learning_with_scikit-learn`, , 22.01.2022, 2021.

[11] Fitzgerald, B., and Stol, K.-J. Continuous software engineering: A roadmap and agenda. *The Journal of systems and software 123* (2017), 176–189.

[12] Fradkov, A. L. Early history of machine learning. *IFAC-PapersOnLine 53*, 2 (2020), 1385–1390. 21st IFAC World Congress.

[13] github. Github, 2020.

[14] Holdgraf, C., and other. Nbformat. GitHub Repository: `https://github.com/jupyter/nbformat`, 22.01.2021, 2021.

[15] Hung, G. From scripts to prediction api. accessed from: `https://towardsdatascience.com/from-scripts-to-prediction-api-2372c95fb7c7`, 22.01.2021, 2020.

[16] Hung, G. model-productization article. GitHub repository: `https://github.com/G-Hung/model-productization_article`, 22.01.2022, 2021.

[17] Jupyter. Project jupyter documentation. accessed from: https://ipython.org/ipython-doc/3/notebook/nbformat.html, 22.01.2021, 2014.

[18] KNUTH, D. E. Literate programming. *Computer journal 27*, 2 (1984), 97–111.

[19] Krekel, H., et al. Pytest documentation. accessed from: https://www.sphinx-doc.org/en/master/index.html, 30.01.2021, 2004.

[20] Kruchten, P. The 4+1 view model of architecture. *IEEE Software 12* (11 1995), 45–50.

[21] Lanubile, F., Calefato, F., Quaranta, L., Amoruso, M., Fumarola, F., and Filannino, M. Towards productizing ai/ml models: An industry perspective from data scientists.

[22] Michelle Ufford, M Pacer, M. S. K. K. Beyond interactive: Notebook innovation at netflix. accessed from: https://netflixtechblog.com/notebook-innovation-591ee3221233, , 23.01.2022, 2018.

[23] Microsoft. Tutorial: Convert ml experiments to production python code. accessed from: https://docs.microsoft.com/en-us/azure/machine-learning/tutorial-convert-ml-experiment-to-production, , 22.01.2021, 2021.

[24] Millman, K., and Perez, F. Developing open source scientific practice. In *Implementing Reproducible Research*, F. L. Victoria Stodden and R. D. Peng, Eds. 2014.

[25] Nascimento, E., Nguyen-Duc, A., Sundbø, I., and Conte, T. Software engineering for artificial intelligence and machine learning software: A systematic literature review.

[26] O'Leary, K., and Uchida, M. Common problems with creating machine learning pipelines from existing code.

[27] OMG. Unified modeling language. accessed from: https://www.uml.org/, , 22.01.2021, 1989.

[28] org, J. Introducing json. accessed from: https://www.json.org/json-en.html, 30.01.2021, 1999.

[29] Pallets. Click documentation. accessed from: https://html.spec.whatwg.org/multipage/, , 22.01.2022.

[30] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research 12*, Oct (2011), 2825–2830.

[31] Perkel, J. M. Why jupyter is data scientists' computational notebook of choice. *Nature (London) 563*, 7729 (2018), 145–146.

[32] Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. Understanding and improving the quality and reproducibility of jupyter notebooks. *Empirical software engineering : an international journal 26*, 4 (2021), 65–65.

[33] Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., and Tonella, P. Testing machine learning based systems: a systematic mapping. *Empirical software engineering : an international journal 25*, 6 (2020), 5193–5254.

[34] Rule, A., Tabard, A., and Hollan, J. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on human factors in computing systems* (2018), no. 32 in CHI '18, ACM, pp. 1–12.

[35] Vaishnavi, V., and Kuechler, W. Design science research in information systems. accessed from: `http://www.desrist.org/design-research-in-information-systems/`, 22.01.2022, 2004 (updated in 2017 and 2019 by Vaishnavi, V. and Stacey, P.).

[36] Van Rossum, G., and Drake, F. L. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[37] Vishal, K. Prediction of quality of wine. accessed from: `https://www.kaggle.com/vishalyo990/prediction-of-quality-of-wine`, 22.01.2021, 2018.

[38] w3sDesign. The gof design patterns reference. accessed from: `http://w3sdesign.com/?gr=c01&ugr=proble`, 30.01.2021, 2018.

[39] Wolfram. Mathematica. accessed from: `https://www.wolfram.com/mathematica/`, , 22.01.2022.

[40] Yihui Xie, J. J. Allaire, G. G. R markdown: The definitive guide. accessed from: `https://bookdown.org/yihui/rmarkdown/`, , 22.01.2022.