University of Zurich[UZH]

MARG: Automatic Visualization of a Data Science Notebook's Narrative

Further Development of a Prototype

Thesis October 20, 2020

Daniela Flüeli
of Zürich ZH, Switzerland

Student-ID: 07-912-744
daniela.flueeli@uzh.ch

Advisor: **Dhivyabharathi Ramasamy**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I am grateful that I was able to write this exciting and instructive thesis in the Dynamic and Distributed Information Systems Group in the IT department at the University of Zurich.

Many thanks to Abraham Bernstein, head of this group, who is driving research in the field of large-scale human-machine systems for data science with the CrowdAlytics project, and to everyone who is involved in this project.

Special thanks go to Cristina Sarasua, postdoc researcher, and Dhivyabharathi Ramasamy, PhD researcher, for the elaboration and formulation of this bachelor thesis, which fascinated me straight away. I would like to thank Dhivyabharathi Ramasamy additionally for supervising my thesis. I appreciated the weekly or bi-weekly exchange with you and I am grateful for your feedback and your inputs.

A thank you to everyone who participated in testing the tool that I developed as part of this thesis. Such voluntary work is essential for research. No less important are all those people who proofread a thesis. I am incredibly grateful to all of you for putting the finishing touches to my work.

My warmest thanks go to my boyfriend, who accompanied me with so much love during this exhausting time.

I would like to dedicate this thesis to my beloved mother, who always believed in me.

Zusammenfassung

Die hohe Flexibilität von Computational-Notebooks hinsichtlich der Organisation und Ausführung von Code unterstützt optimal die im Allgemeinen nichtlineare und iterative Arbeitsweise von Datenwissenschaftlern und wird daher häufig von diesen genutzt. Die gleiche Flexibilität hat jedoch zur Folge, dass viele Notebooks schwierig zu verstehen sind.

In dieser Bachelorarbeit wird die Jupyter-Erweiterung MARG 2.0 vorgestellt, ein Visualisierungs-Plugin, mit der die Verständlichkeit von Notebooks verbessert werden soll. Sie bietet dem Benutzer ein interaktives und dynamisches Baumdiagramm, das die Workflow-Struktur der Notebookzellen visualisiert. Dieses Diagramm zeigt zusätzliche Informationen für die einzelnen Zellen, wie z.B. deren Position in der linearen Zellsequenz, deren Platz im Workflow, die darin ausgeführten datenwissenschaftlichen Aktivitäten, deren Ausführungsnummern sowie die Begründung und Absicht deren Zellinhalts. Die Visualisierung erleichtert das Navigieren und Orientieren innerhalb eines Notebooks, während und nach dessen Erstellung. Die zusätzlichen Informationen können direkt vom Benutzer über die MARG-Benutzeroberfläche eingegeben und geändert werden, woraufhin sich das Baumdiagramm dynamisch aktualisiert. MARG umfasst zudem ein Dashboard, anhand dessen die Entwicklung eines Computer-Notebooks analysiert werden kann.

Abstract

Computational notebooks' high flexibility concerning code organization and execution optimally supports the generally non-linear and iterative way of data scientists' work and is, therefore, a tool they use frequently. However, the same flexibility makes many notebooks difficult to comprehend.

This bachelor thesis presents the Jupyter extension MARG 2.0, a visualization plugin, which aims to improve notebooks' comprehensibility. It offers the user an interactive and dynamic tree diagram that visualizes the notebook cells' workflow structure and allows them to keep track of their exploration. The tree shows additional information for the individual cells, such as their position in the linear cell sequence, their place in the workflow, the type of the data science activity performed in them, their execution numbers, and the code's rationale and intent in them. The visualization facilitates navigating and orientating oneself within a notebook during and after development. The additional information can be entered and modified directly by the user via the MARG user interface, whereupon the tree diagram is updated dynamically. MARG also includes a dashboard that can be used to analyze the development of a computer notebook.

Keywords

Marg, Jupyter Notebook, Plugin, Narrative, Workflow, Forks, Dead Ends, Data Science Steps, Rationale, Tagging, Exploration, Navigation, Visualization, Dashboard

Contents

1

Introduction

Data science is a very popular and influential field in computer science [Aparicio et al., 2019]. Among data scientist, Jupyter Notebook is a widely used tool to perform data analyses [Nature, 2018, Nbviewer, 2020, Rule, 2018]. Code can be written into cells that can be executed individually. Once a cell has been executed, the result is cached. This has the advantage that computationally intense and long-running tasks in a cell are run once and its cached results can be reused for further experimentation. This saves a lot of time that would otherwise be spent waiting for the computation to complete. Nevertheless, cells can be executed any number of times and in any order. They can also be moved in their order. This high flexibility supports the non-linear and iterative workflow of data scientists.

Each executed cell displays its results once the computation is done. It is helpful to have such interim results, numbers, tables and graphs at hand during analyses. They can be used to check assumptions, make comparisons, define the direction of further analysis, and to recall analyses already made or insights gained. To a certain extent, these interim results can also serve as a documentation of the analysis carried out. An additional means to improve documentation are markdown cells. Markdown can display nicely formatted text. The possibility to interweave code fragments, their output and prose is useful both in performing the analysis and in creating structured artifacts that document the analysis and convey the insights gained.

Despite these documentation options, the comprehensibility of data science notebooks is a major problem, as shown in section 4.3. One aspect that can negatively affect the comprehensibility is that comments might lie about the actual meaning of the described code. This can happen when code has been changed or deleted but the comment wasn't updated and thus conveys a wrong meaning of what the code does [Martin, 2008]. This danger exists with conventional programming but is even greater with programmatic data science analyzes performed in notebooks. The highly dynamic, iterative workflow of data scientists can quickly invalidate comments in code and markdown cells.

The exploratory development process to solve a specific data science problem has other negative influences on notebooks' comprehensibility. Such a process often involves pursuing different strategies and using different methods. This results in multiple, non-linear, nested workflows [Liu et al., 2019, Kery, 2018]. However, readers of the notebook will initially try to read through the notebook from top to bottom. This corresponds to our natural reading behavior. But this often contradicts the structure of data science

notebooks due to the usually non-linear process of their creation. This can massively reduce the comprehensibility of the notebook for the reader. The non-linearity can make a standard linear design of a notebook impossible. This can impair the reproducibility of analyses [Pimentel et al., 2019, Rule, 2018]. In addition, notebooks that are not cleaned up during or after completing the data science task may contain a lot of unnecessary code, which can further impair the comprehensibility of the notebook. Those that are overly tidied up may no longer be traceable, executable or reproducible due to missing intermediate steps.

Another aspect that can negatively affect comprehensibility, is the fact that different data scientists have different styles of exploratory development, whereby the specific procedure of an individual data scientist may even differ from project to project [Rule et al., 2018, Kery et al., 2018]. This results in an immense diversity in the structure and content of notebooks. This individuality of notebooks makes it even more difficult for a reader to understand a specific one.

The poor comprehensibility of data science notebooks makes it difficult for notebook readers to grasp or retrieve the key points and findings of the analysis carried out, to gain an overview of the methods used, to reuse code fragments, as well as to replicate and verify the results.

While various solutions could improve the comprehensibility of notebooks in general, described in section **??**, most of the existing approaches do not directly address the problem of the poorly identifiable forks and dead ends which arise from the explorative nature of solving a data science task. Ramasamy et al. have developed MARG 1.0, an extension for Jupyter Notebook, that visualizes the structure of a notebook in terms of forks and dead ends as a static tree diagram [2]. Data science steps implemented in code cells and a possible additional description of the cell content are also displayed in the tree diagram. Together, the auxiliary information and the workflow forms a narrative. MARG should allow a data scientist to quickly orientate and navigate within a notebook and targets to enhance the readability and comprehensibility of notebooks.

The goal of this bachelor thesis is to extend MARG 1.0. The current version of the extension produces a static visualization of the tree based on the metadata available in a JSON format that is generated after notebook creation. As a part of this bachelor thesis, the visualization will be made dynamic by adding the functionality to generate, store and update this metadata directly during notebook creation. Another goal is to provide more statistics on the evolution of notebooks, which should enlarge the body of research on how data scientists perform data science. A better understanding of how data scientists conduct their analyses and create notebooks is the basis for the development of further tools that can support them in their work.

2

Related Work

In the following sections we present some research which we consider as related to the content of this bachelor thesis in some aspects, along with a short explanation of why we consider these studies as important for this thesis and of how they differ from it.

2.1 Assessment of the Comprehensibility of Notebooks

Studies that address the comprehensibility of notebooks and that come to the conclusion that this is a problem, are essential for the legitimization of our work, which aims to improve the comprehensibility of notebooks.
However, our thorough examination of current research on the comprehensibility of notebooks suggests that there is a lack of basic research on this. In particular, there seems to be a lack of studies that investigate the comprehensibility of notebooks experimentally. Studies in which notebook readers work on comprehension tasks such as naming what the goal of the data science analysis carried out in a notebook is, which and how many methods were used in it, what results it yielded, what knowledge can be obtained, which cells are involved in a particular outcome, approach, or issue. To the best of our knowledge, there is no such study at the current level of science. Some research questions are listed in section 4.3, which we believe should be attempted to be answered by basic research.

Although such basic research on the comprehensibility of notebooks seems to be lacking, our work seems to be highly legitimized. A plethora of papers deals with certain characteristics of notebooks, which directly or indirectly influence the comprehensibility of notebooks or are influenced by them. Many of the results of these studies clearly indicate that the comprehensibility of notebooks is quite bad.
There are numerous studies on how data scientists work, which examine, for example, how they work, which tools they use, which problems they face. These studies are often carried out in the form of surveys or interviews, sometimes accompanied by questionnaires. Numerous statements from participants in such studies indicate that notebooks are difficult to comprehend.

The difficulty of getting a high-level overview of the data science analysis processed in a notebook is mentioned several times [Kery et al., 2017, Wenskovitch et al., 2019]. The fact that just capturing the basic features of the notebook content by readers is a

difficulty shows impressively that the comprehensibility of notebooks is quite poor. With MARG we would like to support the readers of notebooks in getting such a rough high-level overview of a notebook. For this purpose, we provide the user with a visualization of the different approaches to solving the data science problem as a tree, equipped with additional information about the data science tasks carried out in the individual cells and the rational behind their content.

Another difficulty seems to be that if you could get a high-level overview of a notebook, the problem still exists of connecting the individual cells to it [Kery et al., 2017, Wenskovitch et al., 2019]. With MARG we also try to address and minimize this problem, for example through the possibility of highlighting the corresponding notebook cells of individual nodes as well as entire branches of the tree diagram provided.

The comprehensibility of notebook seems to be so miserable that even its authors repeatedly have problems understanding the self-written code, finding their way around it and finding individual passages, analysis steps or results [Kery et al., 2018, Kery et al., 2017].

Many participants in quantitative studies refer to their own code as well as someone else's code as messy [Kery et al., 2018, Rule et al., 2018]. They mainly refer to the presence of numerous very small cells in which often only a single command is carried out. The cells are often loosely coupled, redundant or their order reversed. In addition, cells are sometimes far apart due to the output displayed in between. They speak of the not or insufficiently used opportunity to use the cells to divide the code into coherent, meaningful blocks. We do not assume that MARG will change this practice, as it seems to be very much conditioned by the exploratory development process for solving a certain data science problem. However, through the tree diagram provided, MARG can offer visual help to capture the dependencies of the individual cells. The display of which data science steps are carried out in the individual cells also enables the individual cells to be perceived as larger units that together represent a data science task. Notebooks are also often referred to as mess, as they often seem to have a generally poor code quality. [Wang et al., 2020] have examined almost 2000 Jupyter notebooks with "high quality" for their code quality, including compliance with recommended coding practices, the presence of dead codes or deprecated functions. Their results show that the code quality of many notebooks is extremely poor. A total absence or a lack of comments describing the code was found in another large-scale study [Rule et al., 2018].

Notebook authors seem to justify or legitimize this poor code quality with the nature of the exploratory development process and the goal of data analysis. The goal of the analysis is often to get results and insights from data as quickly as possible. The process used here is exploratory, iterative, non-linear in nature. This tempts data scientists to neglect the code quality in favor of a quick achievement of the goal [Kery et al., 2017, Rule et al., 2018]. It can be assumed that the poor code quality of notebooks impairs their comprehensibility. Thus, all of these hints and references to a general bad code quality of notebooks can be seen as implicit references to bad notebook comprehensibility, through which we legitimize our work.

Another aspect that is often mentioned and which can have a negative influence on the comprehensibility of notebooks is the non-linear execution order of its individual cells. Studies have shown that around 30 - 40% of the notebooks examined

have a non-linear execution order [Pimentel et al., 2019, Rule, 2018]. Since this non-linear structure of notebooks contradicts our natural top-to-bottom reading behavior, the comprehensibility of notebooks is massively impaired. This non-linear structure of notebooks is also one of the factors that makes many notebooks not reproducible [Pimentel et al., 2019, Koop and Patel, 2017, Rule et al., 2018]. That the non-reproducibility is a major problem of notebooks is shown by the results of a study, which states that only about 4% of almost one million notebooks could be reproduced [Pimentel et al., 2019]. This non-linear structure of the notebooks also means that the reader often has to scroll back and forth within the notebook in order to follow the analysis workflow [Wenskovitch et al., 2019].

All these indications that notebooks are often difficult to understand show the importance of our work, through which we try to increase the comprehensibility of notebooks.

2.2 MARG 1.0

MARG is a Jupyter Notebook extension prototype, which aims to improve notebooks comprehensibility by visualizing a Jupyter notebook's narrative using workflow information. The meaning of the word MARG is âpathâ in Sanskrit [1] and is intended to express that our tool aims to make the user aware of the forks and dead ends that are not directly visible in a notebook.

A prototype, MARG 1.0, has been developed by Ramasamy et al. The MARG 1.0 only offers a static visualization of this information as a tree diagram. The visualization does not dynamically adapt to subsequent notebook changes, so adding, deleting or moving cells has no effect on the tree diagram shown. It is also not possible to add or change the information later using the MARG or Jupyter interface. This would have to be done directly in the source code.

MARG allows the user to explore the notebook by navigating through a visualization. It adds an exploration and navigation tool to Jupyter Notebook. This tool is primarily intended to make notebooks easier to document the analysis workflow and thus understanding.

MARG facilitates the retrieval of important results and the used methods. In addition, improved comprehensibility of notebooks can help to reuse code fragments and to replicate and verify results.

The main component of MARG is a tree-based visualization of the notebook. Each node in the tree represents a cell in the notebook. In the visualisation, nodes are differentiated by shape depending on whether they refer to code cells or markdown cells. Code cells are represented by anthracite circles with the number on the node corresponding to the position of the mapped cell within the notebook, counting from the top. Markdown cells are represented by white rectangles showing the first few characters of the respective markdown cell text. The edges of the tree diagram reflect the flow in the notebook workflow. If based on previous data, knowledge, and results, different approaches are

[1] https://en.wiktionary.org/wiki/%E0%A4%AE%E0%A4%BE%E0%A4%B0%E0%A5%8D%E0%A4%97

pursued in parallel, this is reflected in a fork. Dead-ends and forks are decision points (see section 3.2.2. Forks are represented by dashed edges, while the edges of singular approach chains are represented by solid lines. In the tree visualization, additional information about each corresponding cell is displayed. To the right of a node, the current execution count of the cell is displayed in square brackets. Above a node, data science tasks (see 3.2.2) are listed that are carried out in the cell. On hovering over a node the rationale behind implementation or procedure decisions for the cell are displayed. Nodes that are declared as dead end have a red border.

Currently, the data about the decision points (forks/dead-ends) and description (rationale) for the visualization is manually curated and represented in a JSON format

As a result, the MARG prototype only offers a static tree visualization of this information. It does not dynamically adapt to subsequent notebook changes, i.e. deleting or moving cells has no effect on the tree visualization shown. It is also not possible to add or change the information later using the MARG or Jupyter interface, the input JSON data must be manually edited.

MARG 1.0 has various functionalities for navigating and exploring a notebook. By clicking on a node in the tree diagram, the corresponding cell in the notebook is selected, highlighted and scrolled into view. Similarly, by clicking on a notebook cell, the corresponding node of the tree diagram is selected, highlighted and scrolled into the view. By double-clicking on a node of the tree visualization and choosing fork or all, either all related nodes can be highlighted in one color or all individual outgoing forks in distinct colors. The corresponding notebook cells are also highlighted here in both cases. Additionally, MARG has a data science step picker feature. With this, individual or multiple data science steps can be color highlighted in the tree visualization. The dehighlighting of all data science step labels in the tree visualization can be done quickly and easily via the reset button. Individual subgraphs of the tree diagram can be collapsed and expanded using the - or + buttons.

The exploration and navigation tool has a fixed size and position but can be scrolled vertically if necessary. The tree diagram itself also has a fixed size and position and can be scrolled both horizontally and vertically to make the whole tree explorable. Below the tree diagram, a small legend provides information on how decision points, i.e. forks and dead ends are marked in the tree diagram.

Figure 3.4 shows how MARG is integrated in Jupyter to the left of the actual notebook area.

Reference to paper Usability study done

The MARG prototype is written in javascript and CSS, using the libraries d3 for the generation of the tree diagram and jQuery for easy HTML DOM tree traversal and manipulation as well as for event handling.

2.3 Data Science Step Prediction Tool

The automatic annotation algorithm from the "Automatic Annotation of Data Science Notebook"[Ramasamy, 2019] uses supervised machine learning and implements gradient

boosting classifier to predict data science step labels for code snippets. The classifier is trained on a dataset of 100 manually curated data science Jupyter notebooks written in Python 3.0+. The algorithm takes as input the code content of a notebook cell, which itself consists of a list of the lines of code. The output consists of a label that is one of these data science steps: load data, data preprocessing, data exploration, data visualization, modeling, prediction, evaluation, helper functions, result visualization, save results. In this thesis, I integrate this automatic annotation algorithm into MARG 2.0 as described in section 3.1 and 3.2.5.

3

Methodology

The goal of my bachelor thesis is to further develop the Jupyter Notebook extension, MARG. The purpose of MARG and this bachelor thesis is to enhance the comprehensibility and traceability of data science notebooks, since this can help notebook readers to grasp or retrieve the key points and findings of the analysis carried out, to gain an overview of the methods used, to reuse code fragments, as well as to replicate and verify results. In this section I provide an overview of the different components that I add to it, in order to reach this goal. Afterwards I describe my approach to design and implement these components. The following describes how I evaluate the implemented components that extend MARG. And the closing of this chapter is the description of possible limitations on the validity of the results of this evaluation.

3.1 Task Overview

The goal of this thesis is to extend MARG and to enable the user to change the JSON dynamically through the MARG UI. In order to accomplish this, four tasks were defined. The goal and purpose of each of these tasks are briefly described below.

Task 1: Add a tagging tool

The input data (explained in 2.2) for the current version of MARG has been generated manually for notebooks that have already been developed. This task is to integrate the creation of this input data directly into the notebook development process. The data on which the tree diagram is based should be able to be entered and changed using a tagging tool developed for this purpose. Tagging parallel to coding should make the tagging process easier, faster and more precise due to the mental presence of the target, the purpose and the embedding of the code.

Task 2: Integrate an existing stand-alone tool

This task is to integrate the data science step prediction tool (see section 2.3) into MARG. The data science step labels predicted for a cell by this tool can be used to simplify and accelerate tagging for the user. At the same time, the data science step prediction tool can be continuously improved through the increasing amount of labeled input data.

Task 3: Make the exploration and navigation tool dynamic

The current version of MARG produces a static visualization of the exploration and navigation tree based on the metadata available in a JSON format [explained in 2.2]. This task is to make the visualization dynamic. It should adapt to new or changed additional information added by the user via the tagging tool [see task 1]. It should also adapt to the addition and deletion of cells, their order and their execution counts.

MARG allows a user to explore notebooks and to analyze the workflow that was used to write it. The purpose of this task is that a user can make use of the exploration and navigation tools that MARG offers in parallel to the creation and editing of a notebook.

Task 4: Add a dashboard

This task is to design, implement and integrate a dashboard, showing the evolution of a notebook. The history of a notebook can contribute to the comprehensibility of the notebook: The dashboard shows different metrics like lines of code over time. In addition, the dashboard could form the basis for studies on how data scientists perform data science and enlarge the body of research in this field. A better understanding of how data scientists conduct their analyses and create notebooks may have implications on existing tools that can support those in their work as well as on the development of new such tools. Even if these components are considered separately in the following sections, they are not to be regarded as independent of one another; some of them complement one another, others require one another.

3.2 Design and Implementation Consideration

This section provides the implementation details of the tasks listed in 3.1. I also provide the design decisions, specifications and the scope of the system. In **??**, a high-level view of the entire MARG system is given, all following subsections are dedicated to the individual components of MARG. For each component, the elicited requirements, the identified constraints as well as the design decisions and their rationale are provided.

3.2.1 Data Storage

A Jupyter notebook is a simple JSON document (see figure 3.2). Each cell has, among other things, the cell type as a character string, a dictionary for the cell metadata and the cell content as a string or list of strings. The metadata is of particular interest to us. A metadata property exists for the notebook as well for each given cell. MARG uses this metadata property to persist its data. An advantage of this approach is that the Jupyter data and the MARG data is combined in a single JSON document. This saves the subsequent merging of this data which simplifies further processing steps of the data.

```
{
  "metadata": {
    "kernel_info": {
      "name": "the name of the kernel"
    },
    "language_info": {
      "name": "the programming language of the kernel",
      "version": "the version of the language",
      "codemirror_mode": "The name of the codemirror mode to use [optional]"
    }
  },
  "nbformat": 4,
  "nbformat_minor": 0,
  "cells": [
    {
      "cell_type": "code|markdown|...",
      "execution_count": 1,
      "metadata": {
        "collapsed": true,
        "scrolled": false
      },
      "source": "[some multi-line code]",
      "outputs": [
        {
          "output_type": "stream"
        }
      ]
    }
  ]
}
```

Figure 3.1: The Rough Structure of the Jupyter Notebook File Format

3.2.2 Tagging Tool

The tagging tool has been developed to allow tagging of individual notebooks cells with additional information which has been described in 2.2. I determined and recorded the requirements for the tagging tool in the form of user stories (see table A.1).

```
{
  "MARG_cell_id": 35,
  "MARG_node_id": 1,
  "MARG_name": [
    "modelling",
    "prediction",
    "load_data",
    "data_preprocessing",
    "data_exploration",
    "data_visualization",
    "evaluation",
    "helper_functions"
  ],
  "MARG_decision": [
    "dead_end",
    "fork"
  ],
  "MARG_parent_node_id": 2,
  "MARG_parent_cell_id": 30,
  "MARG_rationale": "example rationale"
}
```


Figure 3.2: MARG Metadata

Tagging Tool Design

A Single (Floating) Tagging Window One Tagging Window Per Cell

Table 3.3 shows two draft designs of the tagging tool. Put simply, the main difference between the two variants is the positioning of the tool within the user interface of Jupyter notebook. While in one variant the tool is positioned outside the actual notebook area, in the other variant the tool is strongly embedded in it. In the first variant, the tagging tool consists of a single window. As a floating window, this can be freely positioned in the browser window. In addition, the window can be docked above or below MARGâs navigation and exploration tree. This combination of a fixed, extremely compact and a highly flexible arrangement could accommodate different user needs and preferences as well as different browser window sizes. This design is strongly inspired by that of the Table of Contents (2) extension of the jupyter_contrib_nbextensions package. In the second variant, the tagging tool consists of many individual windows. One of these is integrated in every notebook cell, directly below the input area of the cell and thus above possible cell outputs. In this design, particular importance is attached to the proximity of the tagging information to the content of the cell. Both variants have both advantages and disadvantages, which are summarized in Table 3.3.

Weighing up the respective advantages and disadvantages suggests rather one window per cell. The decisive factor for the decision to implement the tagging tool as one window per cell, however, was ultimately the preference of the original author/developer of the MARG prototype. This solution was more like what she vaguely imagined.

	A Single (Floating) Tagging Window	**One Tagging Window Per Cell**
Advantages	- little redundancy - takes up little space - position customizable - unchanged appearance of the notebook area - do not have to scroll up and down in very large cells	- quick overview of entire notebook - information within cells - short distances to referenced cell - possibly less mistakes and misunderstandings due to short distance to referenced cell
Disadvantages	- no quick overview of entire notebook - no information within cells (a cell internal view mode could however be added) - possibly large distances to referenced cell - possibly more mistakes and misunderstandings due to possibly large distance to referenced cell	- lots of redundancy that clutters the notebook up - takes up a lot of space - position fix, not customizable - changed appearance of the notebook area - does have to scroll up and down in very large cells to capture the contents of the entire cell and to tag accordingly

Figure 3.3: Tagging Tool Design Variants: Advantages and Disadvantages

Data Science Steps

The user can choose from a fixed set of possible data science step labels for tagging notebook cells. We used the labels and algorithm from [Ramasamy, 2019]. Based on this source we chose the following data science step labels:

- load data
- data preprocessing
- data exploration
- data visualization
- modeling
- prediction
- evaluation
- helper functions

Whether the current selection of available data science step labels is reasonable and useful requires further clarification and research.

Decision Point (Fork)

By default, a new cell is considered to be a child of its predecessor cell. Thus, the user only needs to specify a parent for a cell if a fork happened.

While Jupyter Notebook internally uses an indexing of the cells started from zero, MARG should use one that starts from one, as this corresponds more to a natural numbering of objects, here cells. This inconsistency is not a problem insofar as the indices used internally by Jupyter Notebook are nowhere visible in its user interface.

Cycle detection

A user needs to be able to specify the parent for any given cell. MARG's validation mechanism must prevent that a user can enter data that would lead to a cyclic dependency. To avoid cyclic dependencies, we only allow mouse input to the numeric input HTML element, e.g. clicks on the spinners (the small up and down arrows next to the input field) or via mouse wheel.

We disabled keyboard input for two reasons. First, when a user wants to type 31, he first types 3 and the after a delay 1. How long should we wait for the user input until starting the validation? Second, how do we clearly signal to the user, that his input value has been rejected and reset? The detailed process of this variant is shown in Figure 3.5 as a flowchart. After each click on a spinner, MARG computes which number forms the next possible valid input. If the internal test shows that this new value is a valid input, it is set and displayed in the input field. If not, the next valid input is searched for in the identified direction. If the internal test shows that this new value is a valid input, it is displayed in the input field. If not, the next valid input is searched for in the identified direction. For the edge case that each number within the range would form a cycle, the input field remains empty, clicks on the spinners do not change anything. Figure 3.4 shows such a case.

Figure A.4: In this example, it is not possible to set a parent in the first cell, since every possible reference to a cell of the notebook would cause a cyclic dependency structure.

3.2.3 Exploration and Navigation Tool

The exploration and navigation tool extends the MARG prototype (see section 2.2). The main contribution is the introduction of a dynamic version of the tree visualization. The exploration and navigation tool is now linked to the tagging tool, described in Section 3.2.2. Thus, when an user makes changes to the tags or adds, deletes, moves or merges notebook cells, this is immediately represented in the tree. The purpose of this adaptation is that a user can make use of this exploration and navigation tool in parallel to the creation and editing of a notebook. The existing code base of the prototype was optimized towards static visualization, therefore we decided to reimplement the

exploration tool from scratch. Below there is a list of all changes compared to the existing prototype. In addition to the main change that the tree diagram now adapts dynamically to additional or changed input data, important changes have been made to improve the visualization quality of the tree diagram and the usability of the exploration and navigation tool. The former primarily addresses the reduction of overplotting. There are problems of overlapping nodes in the tree diagram of the prototype. This problem results from a fixed maximum tree width, which in this case is specified by the width of the svg HTML element. Since an aesthetically pleasing tree that requires minimal drawing space is usually not characterized by fixed distances between nodes, these are not an option to solve the overplotting problem. The problem is finally solved or at least minimized by a dynamically determined base area available to the tree. The method of the d3 library used to generate the tree then automatically positions the nodes and links on the specified area according to best aesthetic and space considerations. Defining the height of this dynamic base area is not a problem, just multiply the height of the tree by a fixed distance. The fixed distance is chosen so that there is enough space between the nodes to display in extreme cases all data science step labels above the other. Defining the width of this dynamic base area is more difficult, since the width is neither directly proportional to the total number of nodes, nor to the maximum number of sibling nodes of all levels. However, a relatively respectable result that minimizes overplotting can be achieved by multiplying a fixed distance by the number of leave nodes. In Figure 3.4 you can clearly see that this dynamically defined base area in the tree means that no nodes overlap, in contrast to the same tree in the prototype, Figure 3.4. Figure 3.4 shows how another problem of overplotting, this time that of overlapping data science step labels, is addressed in the further development of MARG. The labels are now arranged one above the other instead of next to one another. Since the distances between the nodes cannot be defined directly due to the selected tree layout and these vary greatly, this solution can reduce overlaps, but not entirely avoid them. The important changes with regard to an improved usability of the exploration and navigation tool concern an optimal and flexible use of the available space for the representation of the tree diagram. In the MARG prototype, the exploration and navigation tool window takes up a fixed amount of space, regardless of tree size and browser window size. The tree diagram window contained therein can be scrolled, but the size of the area visible at once is also fixed. In the version of MARG developed as part of this bachelor thesis, the size of the tagging tool window can be changed manually, whereby the tree diagram window contained therein adapts dynamically to the new size. The tree diagram window can also be given more space vertically by collapsing the data science step picker below. If the whole tree is not visible at once in the tree diagram window, then the window is still scroll-able as in the prototype. In order to further improve and simplify the comprehension of the entire tree structure and the navigation within it, the entire tree diagram window can be zoomed in this further developed version of MARG. In addition, the exploration and navigation tool window can be entirely collapsed and expanded. This should take into account the various use cases of MARG, where the visibility of the exploration and navigation tool is not necessary or even disadvantageous in all cases.

- the input data are subjected to a validity check
- the root of the tree diagram can be focused
- tree diagram is made zoom-able
- the tree diagram updates dynamically
- the size of the tagging tool window can now be changed manually
- the tagging tool window is made collapsible
- data science step picker is made collapsible
- the arrangement of the data science step picker adapts dynamically to the window size of the exploration and navigation tool
- the window size of the tree diagram adapts dynamically to the window size of the exploration and navigation tool
- the tree diagram window size dynamically uses the area that is freed when the data science step selection is reduced
- the tree width is no longer fixed, but adapts dynamically to the number of nodes in order to reduce node overlapping
- leave nodes no longer have a collapse sign
- node events are now also registered on the node number
- coloring of selected node and the corresponding cell is now uniform
- coloring of related node and the corresponding cells is now uniform
- node numbers are now centered
- data science step labels are arranged one below the other instead of side by side
- changed style of nodes of markdown cells
- changed position and style of the rationale shown on node hover
- changed position and style of the path menu shown by double-clicking on a node
- path menu option renamed from 'all' to 'relatives'

3.2.4 Tree Generation

This section describes the tree generation process. Since the hierarchical structure of the tree is defined by the explicit and implicit parents of the cells of a notebook, this terminology is first explained and made more tangible using an example. Then the algorithms for setting these explicit and implicit parents are described in detail.

Definition of Explicit and Implicit parents

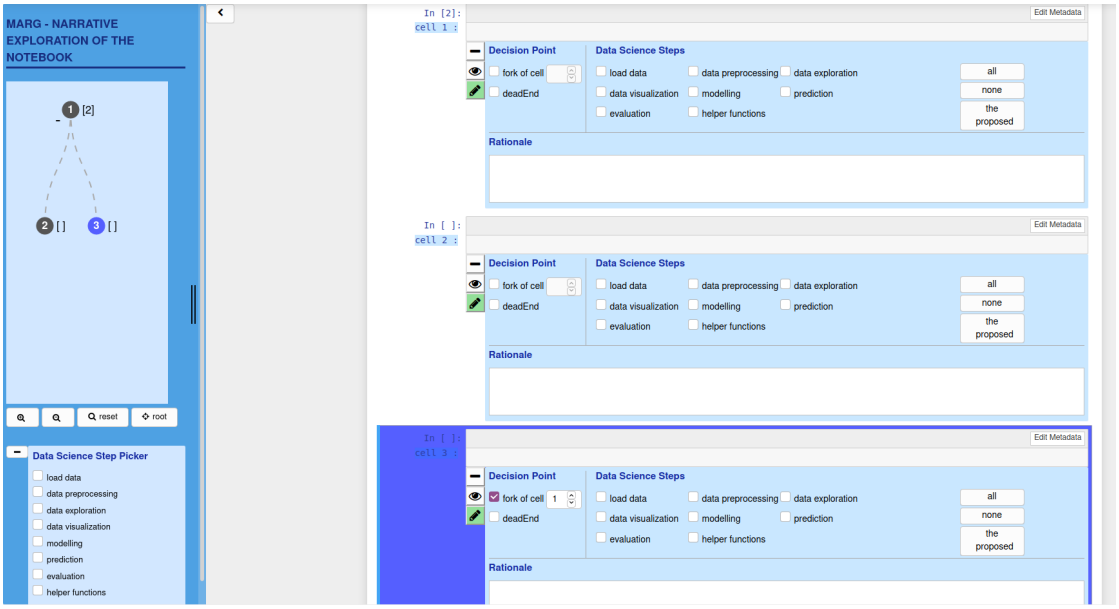

Figure 3.4: Explicit and Implicit Parents Overview. This shows a Jupyter notebook with the extension MARG running. The right part shows the actual notebook, which comprises three cells. Each cell has a MARG tagging tool (described in section X). The left part shows the MARG exploration and navigation tool. Its tree diagram visualizes the dependency structure of the notebook cells.

Explicit parents refer to parents that are explicitly set by the MARG user using the tagging tool. In the example shown in figure 3.4, only cell number 3 has an explicit parent, i.e. cell number 1.

Implicit parents refer to parents that are automatically set by MARG for all cells that do not have an explicit parent and that are not the root of the tree diagram. Implicit parents are not displayed in the tagging tool window itself. However, they are shown indirectly in the location of the cells within the tree diagram of the tagging and exploration tool. In the example shown in figure 3.4 cell number 2 is the only cell that has an implicit parent, since cell number 1 is the root of the tree diagram and cell number 3 has already an explicit parent. From the tree diagram of the tagging tool it can be deduced that cell number 2 has as implicit parent cell number 1.

This differentiation into explicit and implicit parents is a necessity for the efficient use of MARG. It would be very time-consuming and tedious if the user had to specify for each cell explicitly which cell it depends on. To prevent this, a default cell dependency is defined, based on which the implicit parents are set. Thus, the user only has to explicitly specify from this default deviating dependencies. The default cell dependency is defined in section 3.2.4.

Setting implicit parents

In general, the implicit parent is the direct predecessor cell of the current cell. However, if setting the predecessor cell as parent would result in a cyclic dependency structure in the tree, an invalid state would have been reached. To detect this, we use the same validation algorithm to detect cycles as described for the explicit parents.

As a result, the implicit parent isn't always the direct predecessor cell. However, it is always a preceding cell within the notebook. The algorithm check, whether the previous cell would lead to a cycle. If not, it's selected as parent. Otherwise, the preceding cell is checked and so on. However, this alone doesn't lead to unique results. See figure X which shows that with this definition, two different trees can result. Hence, we need to add to the definition that we start the algorithm from the top of the tree.

By default, a new cell is considered to be a child of its predecessor cell. Thus, the user only needs to specify a parent for a cell if a fork happened. The detection of the predecessor cell is not straight forward.

Figure 3.5 shows a flow chart that visualizes the algorithm to set implicit parents. It contains two loops. The outer loop iterates in ascending order through all cells in the notebook and filters nodes that are forks and the root. For each remaining cell its implicit parent is determined by an inner loop. The inner loop tests each preceding cell in descending order till one is found that can be set as parent that does not create a cyclic dependency structure. The implementation of this algorithm can be found in Appendix A.3.

Root Detection

Figure 3.7 shows a flow chart that visualizes the algorithm to determine the root. It contains two evaluation points. First: if the first cell in the notebook is not a fork, i.e. if it has no explicit parent, then this cell is taken directly as the root. Second: Otherwise the smallest given explicit parent that is not a fork itself, i.e. if it has no explicit parent, is taken as the root. The implementation of this algorithm can be found in Appendix A.3.

A concrete example based on an example notebook is shown in Figure 3.6. The alphabetically arranged sub-figures A-F describe the step-by-step procedure for determining the root. Each sub-figure is a representation of the dependency structure of the cells in a sample notebook, in which elements that are of interest in the respective step are highlighted in yellow. The cells of this example notebook are represented by blue circles. The number on the circles correspond to the position of the cells within the notebook. Blue arrows represent explicit parents. Sub-figure A shows the initial state at the time of root detection: Two notebook cells have an explicit parent. There are no implicit parents. The first step is to select the first notebook cell, sub-figure B. This cell is not a fork. As can be seen in sub-figure C, it has cell number 4 as an explicit parent. That is why it is not an option as root. Sub-figure D shows that 3 and 4 are the only cell numbers that are used as parents. Listed in ascending order: [3,4]. In the next step the first element of this list, i.e. cell number 3, is taken, sub-figure E. Since this cell does

not have an explicit parent, it is defined as the root, sub-figure F.

Cycle Detection

The implementation of this algorithm can be found in Appendix A.3.

3.2.5 Data Science Step Prediction Tool

The data science step prediction tool is a stand-alone tool described in section 2.3. The integration of this tool into MARG aims to simplify and accelerate tagging for the user. This tool is intended to suggest a set of suitable data science step labels to the MARG user for each notebook cell based on its content.

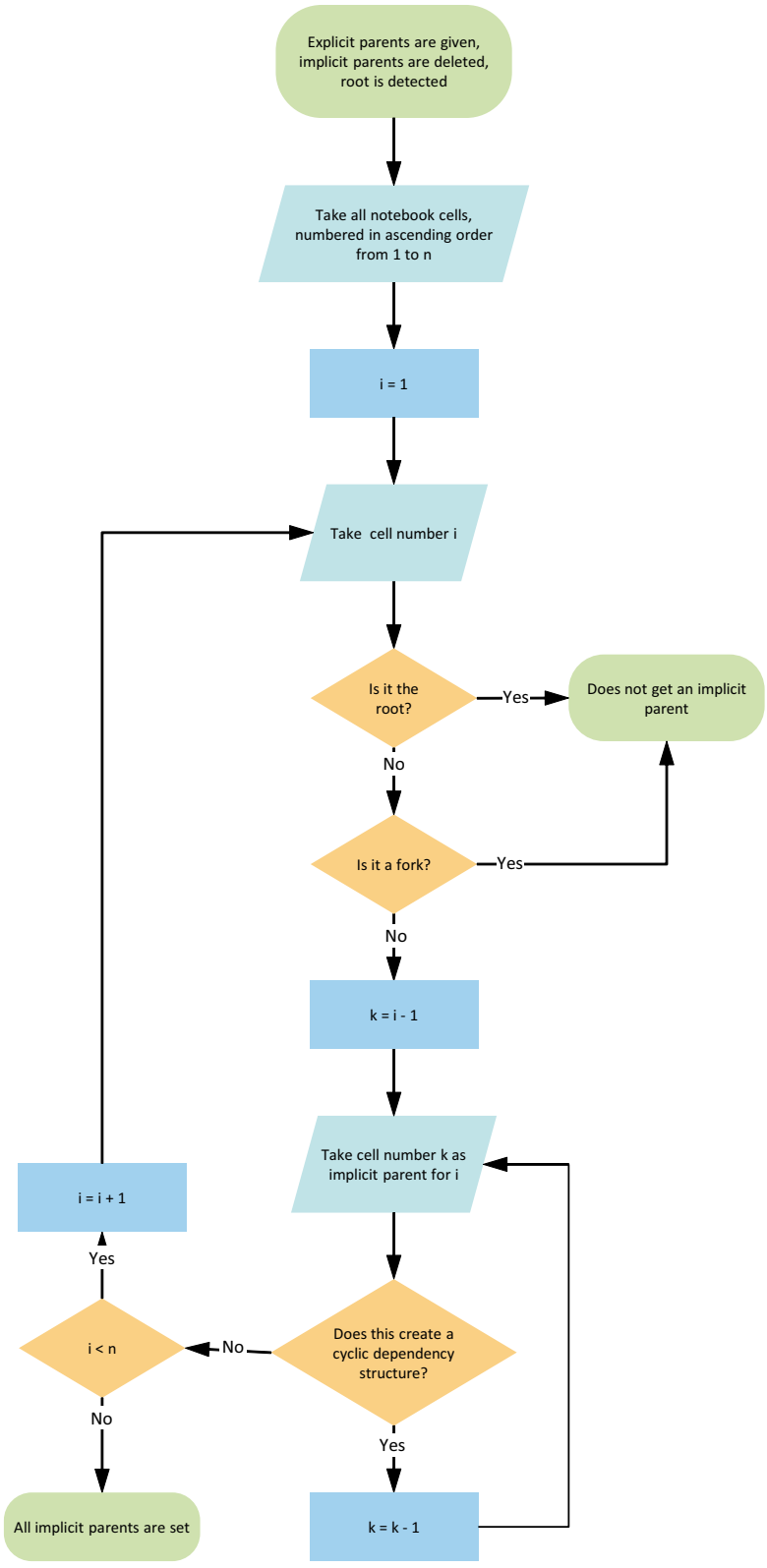

Figure 3.5: Algorithm to Set Implicit Parents as Flowchart

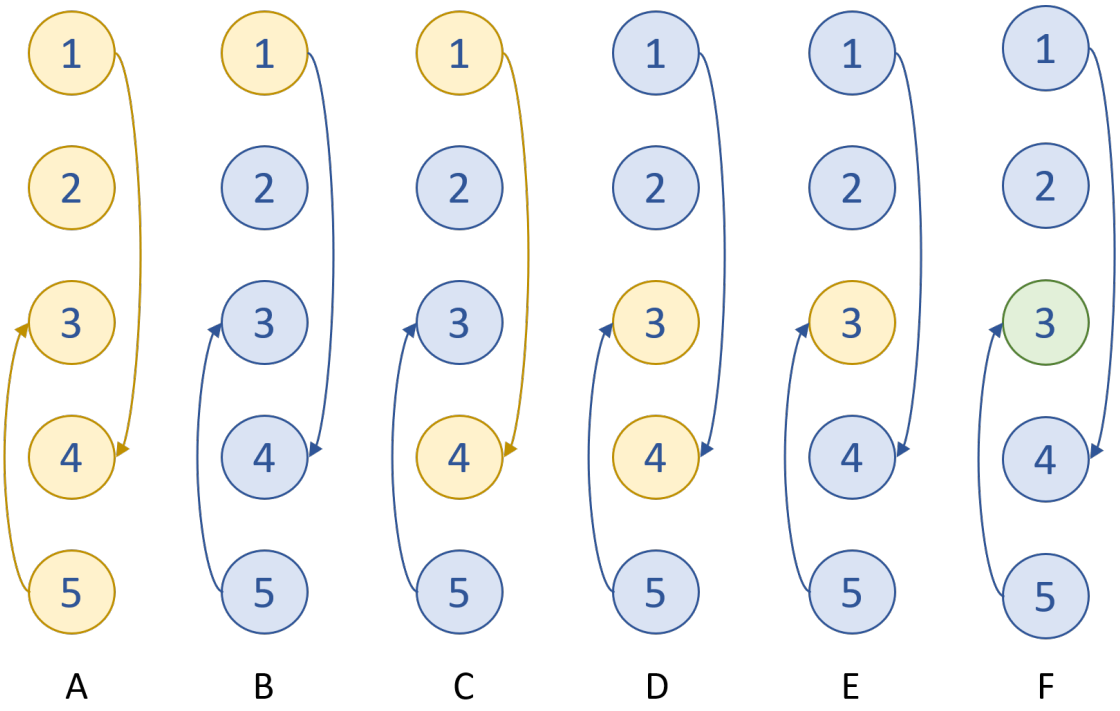

Figure 3.6: Example of the Root Detection Process

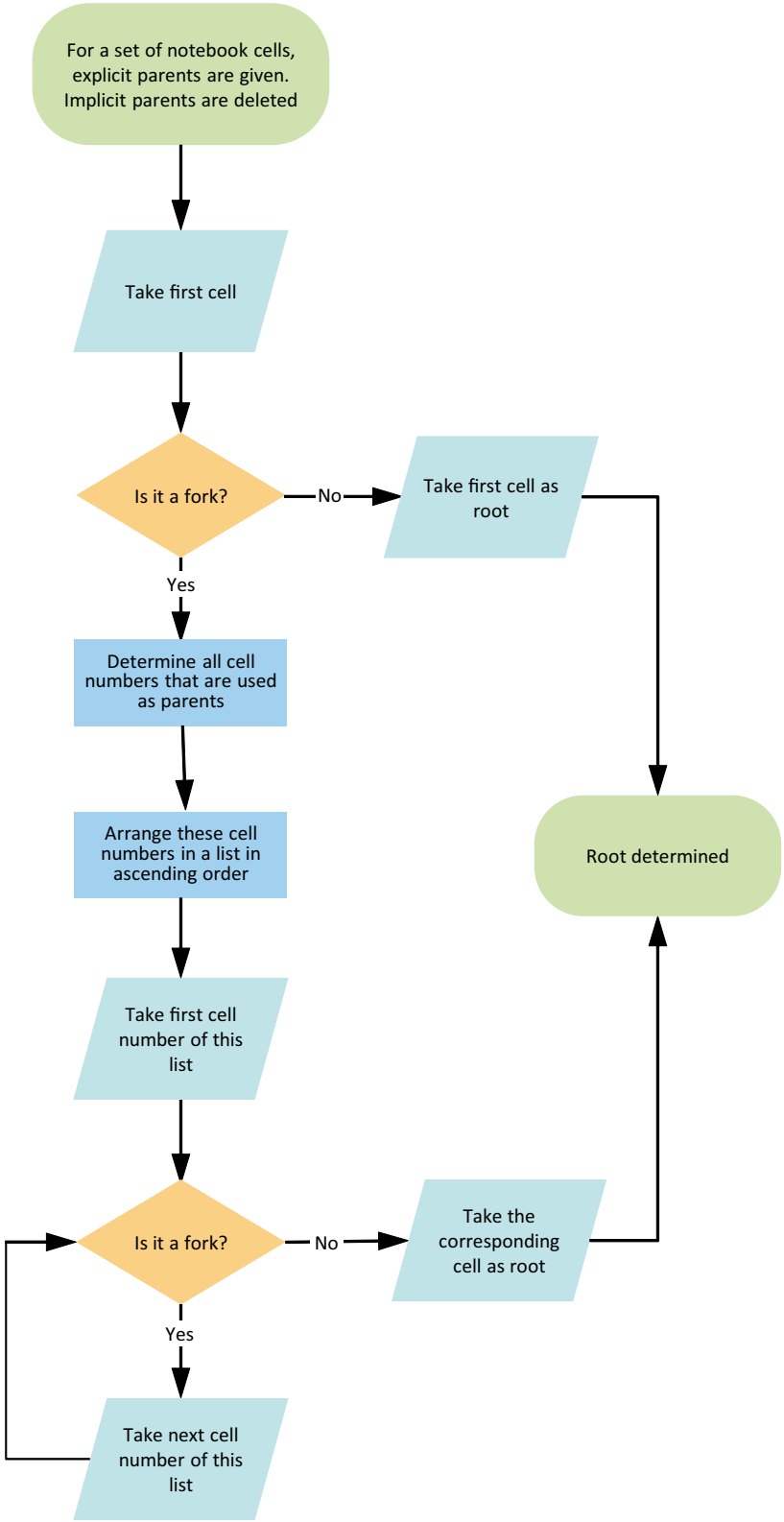

Figure 3.7: Root Detection Algorithm as Flowchart

4

Discussion

4.1 Implementation of MARG

4.1.1 Requirement Validation

The fact that all requirements (see table A.1) for the tasks 1, 2 and 3, described in section 3.1, that had been prioritized with "major" could have been implemented (see section A.1) means that this bachelor thesis brought MARG to the desired level of development. The open requirements classified as "minor" show a detailed and comprehensive planning of the project and can accelerate or inspire the continuation of the project. Unfortunately, due to time constraints, not all requirements for task 4 could be met. The intended dashboard was carefully planned and designed. However, due to time constraints, this could only be implemented as a raw version and not yet integrated into MARG.

4.1.2 Defectiveness

MARG 2.0 still has minor bugs that could not be fixed for reasons of time, but is already fully operational. The bugs currently still present can be found in the test report.

4.1.3 Limitations

- **Adequacy of Tree Diagram:** The data science notebook's narrative is visualized as a tree, which is by definition acyclic. Since the data scientist's workflow may well contain cycles there is some kind of discrepancy.
- **Adequacy of the Default Parents:** If a cell is declared as a fork by the user, its current implicit parent is automatically set by MARG as its default explicit parent, as described in section 3.2.2. It cannot be ruled out that there might be a more appropriate default for the explicit parent. It must be examined how often default parents are adopted by the users and whether a better algorithm for determining the default parents can be found on the basis of the parents that were ultimately set.

- **No Keyboard Input to Set Parent:** The current version of MARG does not allow input via keyboard to set the parent of a cell in the tagging tool (see section 3.2.2).

- **Unfavorable Implementation of Fork Special Case:** When tagging, there can be an edge case where a certain cell cannot be made dependent on any other cell without creating a cyclical dependency structure. In this case it should not be possible to declare this cell as a fork in the Tagging Tool Window. Nevertheless, this is possible in the current implementation. In order to declare and save a dependency, however, a cell must be specified on which the cell is to be made dependent. The corresponding input field for this is not disabled, but unlike usual, it does not display a standard value. Clicks on the spinners and the use of the mouse wheel to change the content of the input field have no effect other than usual. This means that no dependency is saved for this cell. This is to prevent a cell from being declared as a fork without actually being one. Such incorrect information could confuse the user and falsify analyzes of the occurrence of forks.

- **No Guaranteed Data Integrity:** The decision to save MARG's tagging information directly in the cell metadata (section 3.2.1) comes with some disadvantages. One disadvantage of this decision is, that MARG's tagging information can no longer be changed exclusively via the MARG user interface. Because the metadata of the cells can be accessed and edited directly via the Jupyter Notebook interface or programmatically using the Jupyter Notebook API.
  MARG expects the tagging information to have the JSON structure shown in Figure 3.2. In addition, MARG expects only specific values for the individual keys. This can only be guaranteed if the tagging information is set and changed via the MARG user interface. Otherwise there is a risk that MARG will be put into an invalid state and crash.
  Another disadvantage of this decision is the risk of involuntary deletion of all MARG information by deleting the metadata, be it through the carelessness of the user or through side effects of other notebook extensions.
  The current version of MARG does not respond to this problem.

- **Disadvantages of the Tagging Tool Design:** The selected design (section 3.2.2) for the tagging tool has some disadvantages. Because the individual tagging windows are integrated directly into the cells of the notebook, this changes the familiar appearance of the notebook area, which can potentially be perceived as annoying. Since every cell has a tagging tool window, there is a lot of visual redundancy, which can also be perceived as annoying. It also makes the notebook longer, which means more scrolling, which can be seen as a reduction in user-friendliness. The position of the Tagging Tool windows is fixed and cannot be changed by the user according to his needs and preferences. Since the Tagging Tool Window is located below the corresponding cell, it may be necessary to scroll up and down for very large cells to capture their entire contents in order to tag the cell accordingly.

- **Limited Suggestions Through the Data Science Step Prediction Tool:** It can be assumed that some notebook cells embody the execution of multiple data science steps. MARG takes this assumption into account insofar as the user can assign several data science steps to a cell using the Tagging Tool. However, the data science step prediction tool integrated in MARG (see section 2.3) suggests a maximum of one data science step label per cell to the user. It is therefore questionable how often the user effectively uses the quick selection option of this suggestion in a cell.
- **Help for Users:** There is no guided tour or documentation available to the user yet.
- **Data Science Step Labels in Tree Diagram:** There is still cases of overplotting that have to be addressed.
- **Further testing:** More testing is needed.

4.2 Implications of MARG

- **Improved Understanding:** MARG provides enhanced comprehensibility of notebooks. It helps to facilitate exploration, investigation & navigation of notebooks.
- **Extension:** As a Jupyter notebook extension, MARG is easy to incorporate into a workflow and many people are reached who can benefit from the functionalities that MARG offers.
- **Facilitate Key Point Retrieval:** With MARG it's easier to grasp or retrieve the key points and findings of the analysis carried out, to gain an overview of the methods used, to reuse code fragments, and to replicate and verify results.
- **Creation & Editing:** The user can make use of the exploration, navigation and investigation tools that MARG offers in parallel to facilitate the creation and editing of a notebook.
- **Behavioural Impact:** Nudge users to keep code that ended in dead end. Instead of being ashamed of dead ends, they might perceive dead ends as normal and important for comprehensibility and that they aren't worthless but give insight in what does not work. Fork code instead of overwriting/deleting code. Add tagging step while coding as defacto standard due to the given possibilities. Framing: write more often the rationale behind code.
- **Corpus for Research:** The usage of MARG helps to build a corpus of tagged notebooks
- **Incorporating Tagging into Daily Workflow:** integrate the creation of this input data directly into the notebook creation process. Tagging parallel to coding

should make the tagging process easier, faster and more precise due to the mental presence of the target, the purpose and the embedding of the code.

- **Accelerated Tagging:** The data science step labels proposed for a cell by this tool can be used to simplify and accelerate tagging for the user.
- **Self-learning system:** The data science step prediction tool can be continuously improved through self-feeding the the labeled input data.
- **Learn from History:** The history of a notebook can contribute to the comprehensibility of a particular version of the notebook: Discarded items are made accessible again and the original workflow for creating the notebook or processing the data science task becomes more apparent. In addition, the dashboard could form the basis for studies on how data scientists perform data science and enlarge the body of research in this field. A better understanding of how data scientists conduct their analyses and create notebooks may have implications on existing tools that can support those in their work as well as on the development of new such tools.
- **Manage Flexibility:** Reduces the drawbacks that go along with the high flexibility that Jupyter offers and which are the actual strength of it
- **Plethora of Use Cases:** Use cases: â¢ Tagging: Research â¢ Exploration: Understanding â¢ Dashboard: Research â¢ Exploration & Dashboard: Understanding â¢ Exploration & Tagging: Production â¢ Dashboard & Tagging: Research â¢ Tagging & Exploration & Dashboard: Reflection

4.3 Future Work

- **Fix Bugs:** For the continued use of the current version of MARG, for studies, for effective use or for further development, it is of imminent importance that the existing bugs are fixed. A list of the bugs known to date can be found in section 4.1.2.
- **Address Limitations:** One of the first steps in the further development of MARG could be the elimination or reduction of the limitations of the current MARG version reported in section 4.1.3.
- **Improve the Data Science Step Prediction Tool:** The integration of the Data Science Step Prediction Tool into MARG revealed new optimization potential. MARG generates the same type of data as that used to train the tool, i.e code blocks labeled with data science steps. These MARG data can be used to enlarge the training data set of the Data Science Step Prediction Tool. The interaction between MARG and the Data Science Step Prediction Tool can be reprogrammed so that at certain time intervals or after a certain amount of new data, the model

inherent to the latter is re-trained using this enlarged training data set. This could continuously improve the accuracy of the Data Science Step Prediction Tool. This should result in the tool suggesting to the MARG user data science step labels that are more appropriate for a cell. The resulting Data Science Step suggestions, which are more appropriate for a cell, should make tagging easier and faster for the MARG user.
The interaction between MARG and the Data Science Step Prediction Tool could also be reprogrammed in such a way that the latter is rewarded positively if the suggested label and the label set by the user match, and negative if they do not match.

- **Extend the Tagging Tool:**
  - **Assign Labels for Data Science Methods:** Analogous to the visualization of the data science steps that are carried out in a cell, the visualization of the data science methods used, e.g. linear regression, PCA, hierarchical clustering, is desirable. This information, like that of the data science steps, could be added to the cells by the user using the tagging tool, or it could be automatically detected and saved by an algorithm integrated into MARG.
- **Extend the Exploration and Navigation Tool:**
  - **Highlight Results:** Analogous to the highlighting of dead ends, it would be conceivable to highlight in the tree diagram those nodes in whose corresponding cells important results or written insights can be found. This could help a reader or user of a notebook to find or retrieve the most important results and findings quickly and easily. This would be a significant improvement in the comprehensibility of notebooks.
  - **Enable Path Naming:** The Exploration and Navigation Tool could be equipped with the additional functionality of providing entire paths of the tree diagram with a name and a description.
  - **Enable Notebook Slicing:** So far, certain paths can be highlighted using the Path Selection Menu in the tree diagrams of the Exploration and Navigation Tool. The corresponding cells are then highlighted in color in the notebook. An extension of this could be that a new, separate notebook can be created in the future, which only includes these highlighted cells.
  - **Make the Tree Diagram Editable:** So far, the dependency structure of the cells can only be changed by setting explicit parents in the tagging tool or by changing the cell sequence within the notebook. In the future, however, this could also be made possible by moving individual nodes or subtrees directly in the tree diagram of the Exploration and Navigation Tool. Whereupon the data displayed in the tagging tool window and stored in the cell metadata should automatically update. Since the tree diagram can provide a good overview of the entire structure of the notebook, it could be ideal for such changes.

  - **Make the Cell Execution Order Experienceable:** The Exploration and Navigation Tool could be extended by an animation of the tree diagram, which shows the cell execution order by highlighting the corresponding nodes one after the other. It should be considered that the order of the cells as well as the structure of the tree may have changed in the time span shown. One possibility would be to show the animation in the current tree structure and to identify the corresponding nodes using the unchangeable MARG cell id. For cells that do not exist in the current notebook, and thus in the tree diagram, a suitable animation would have to be developed.
    Another conceivable option would be the new functionality that notebook cells can be executed again from a certain execution count or within a certain execution count range.
    This ability to experience the cell execution order is intended to address the problem of reduced reproducibility due to out-of-order executions.

- **Extend the Dashboard:** As part of this bachelor thesis, a dashboard was carefully planned and designed. However, due to time constraints, this could only be implemented as a raw version. The user interface should be implemented according to the existing design template. The missing visualizations should subsequently be displayed on the dashboard. Most of the code to create it is already in place. The dashboard is also to be integrated directly into MARG.

- **Conduct Studies:**
  - **Basic Research on Notebook Comprehensibility:** Our thorough review of current research on notebook comprehensibility revealed that there seems to be a glaring lack of studies that experimentally investigate notebook comprehensibility. Here we list some research questions that, in our opinion, should definitely be tried to answer.
    * **RQ1:** How well are notebook readers able to identify the goal of the data science analyzes performed?
    * **RQ2:** How well can notebook readers able determine how many different approaches have been used to solve the data science problem?
    * **RQ3:** How well can notebook readers determine which methods were used for the analysis performed in the notebook?
    * **RQ4:** How well are notebook readers able to recognize the results of the analysis carried out in the notebook?
    * **RQ5:** How well are notebook readers able to grasp what knowledge can be drawn from the notebook?
    * **RQ6:** How well are notebook readers able to track down which cells are all involved in a particular result, approach or output?
  - **MARG Comprehensibility Studies:** To research the effect of MARG on the comprehensibility of note books, controlled experiments should be

carried out. Half of the participants should have MARG available, the others not. The study participants should work on specific comprehension tasks. Tasks such as finding out which data science task is being processed in a notebook, which and how many different approaches were used to solve the data science task, which methods were used, what the results of the analysis are. The studies should be evaluated both with regard to the correctness of the individual answers and with regard to the time required.

  - **MARG Usability Studies:** To investigate the handling of the MARG user interface, usability studies should be carried out. This should uncover problems in the design, identify opportunities for improvement, reveal the behavior and preferences of the user, draw attention to problems, irritation and misunderstandings during use, determine user satisfaction about the user experience. Qualitative and quantitative data should be collected.
  - **MARG Usage Analysis:** The type and frequency of use of the individual MARG components and functionalities should be tracked continuously while MARG is being used. This data should be visualized in the dashboard. Analyzes of this data should provide the MARG developers with information about the effective use of MARG. The insights gained from this can have a decisive influence on the further development and redevelopment.
  - **MARG Design Variants Comparative Study:** As part of this bachelor thesis, two different design variants were designed for the tagging tool, but only one of them was fully implemented and integrated into MARG (see section 3.2.2). Since the weighing of the advantages and disadvantages of the two variants did not result in a superior winner at the time, the second variant should be fully implemented and integrated into a second, parallel version of MARG. In controlled experiments, it should then be evaluated which of the two design variants better supports the MARG user in performing the tagging of the notebook cells quickly and precisely.
    Based on the results of such studies, it should be decided in which design variant the tagging tool in MARG should be available in the future. It should also be determined whether it represents added value and whether the user wants to have both variants in MARG and the user is asked to choose and switch between the two.

- **Gather Feedback:** Feedback should be obtained on an ongoing basis about how MARG can be further improved. Interviews, questionnaires and surveys or a permanent online platform are conceivable.

- **Release MARG:** The long-term goal of this project should be to make MARG publicly available. This means that the thousands of Jupyter Notebook users can be supported in carrying out data science analyzes.

5

Conclusions

There is some evidence that data science notebooks have a general problem of comprehensibility. Various tools have been developed that attempt to improve one or more aspects of the comprehensibility of notebooks.

Our Jupyter extension, MARG, extends this set of existing tools. It offers the user an interactive tree diagram that visualizes the notebook cells' workflow structure. During and after the development, this diagram should provide a good overview of the notebook as well as facilitate navigation, orientation, and exploration. In this bachelor thesis, MARG 1.0 was further developed, and MARG 2.0 was created. The main contribution is the introduction of a dynamic version of the previously static tree diagram. The data on which the tree diagram is based can now be entered and changed using a tagging tool developed for this purpose. Another contribution is the integration of the automatic annotation algorithm [Ramasamy, 2019], which aims to simplify and accelerate tagging for the user. MARG 2.0 still has minor bugs that could not be fixed for reasons of time, but is already fully operational. As part of this work, a dashboard was also implemented that can be used to analyze the development of a computer notebook. This was carefully planned and designed, but it could only be implemented as a raw version and not yet integrated into MARG as planned due to time constraints. MARG 2.0 also has limitations that have to be assessed as to whether and how these should and can be remedied. In order to investigate the handling of the MARG user interface, a usability study will soon be carried out, which unfortunately could not take place before this work was submitted.

MARG 2.0 in its current version could facilitate the daily work of countless data scientists and could be used to create a large corpus of tagged Jupyter notebooks. Maybe MARG, albeit in a later version, can become a de facto standard in the creation and consultation of data science Jupyter notebooks due to its ease of use and its versatile possibilities in terms of navigation, orientation, and exploration within a notebook. Plenty of ideas for further developing MARG are already there.

References

[Aparicio et al., 2019] Aparicio, S., Aparicio, J. T., and Costa, C. J. (2019). Data science and ai: trends analysis. In *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE.

[Kery, 2018] Kery, M. B. (2018). Towards scaffolding complex exploratory data science programming practices. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 273–274. IEEE.

[Kery et al., 2017] Kery, M. B., Horvath, A., and Myers, B. A. (2017). Variolite: Supporting exploratory programming by data scientists. In *CHI*, volume 10, pages 3025453–3025626.

[Kery et al., 2018] Kery, M. B., Radensky, M., Arya, M., John, B. E., and Myers, B. A. (2018). The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11.

[Koop and Patel, 2017] Koop, D. and Patel, J. (2017). Dataflow notebooks: encoding and tracking dependencies of cells. In *9th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 2017)*.

[Liu et al., 2019] Liu, J., Boukhelifa, N., and Eagan, J. R. (2019). Understanding the role of alternatives in data analysis practices. *IEEE transactions on visualization and computer graphics*, 26(1):66–76.

[Martin, 2008] Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson.

[Nature, 2018] Nature (2018). Why jupyter is data scientistsâ computational notebook of choice.

[Nbviewer, 2020] Nbviewer (2020). Estimate of public jupyter notebooks on github.

[Pimentel et al., 2019] Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. (2019). A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 507–517. IEEE.

[Ramasamy, 2019] Ramasamy, D. (2019). Automatic annotation of data science notebooks. In *Online: https://www.merlin.uzh.ch/publication/*.

[Rule et al., 2018] Rule, A., Tabard, A., and Hollan, J. D. (2018). Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12.

[Rule, 2018] Rule, A. C. (2018). *Design and Use of Computational Notebooks*. University of California, San Diego.

[Wang et al., 2020] Wang, J., Li, L., and Zeller, A. (2020). Better code, better sharing: on the need of analyzing jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 53–56.

[Wenskovitch et al., 2019] Wenskovitch, J., Zhao, J., Carter, S., Cooper, M., and North, C. (2019). Albireo: An interactive tool for visually summarizing computational notebook structure. In *2019 IEEE Visualization in Data Science (VDS)*, pages 1–10. IEEE.

A

Appendix

A.1 User Stories

No	Story	Priority	Status
1.	As a user, I want to add a dead end declaration to a cell so that I can enrich cells with the information that I will no longer pursue the approach I have begun in the following cells and that this information can be used to update the tree diagram	Major	Done
2.	As a user, I want to remove a dead end declaration from a cell so that I can undo the declaration as dead end if, for example, the code within the cell or my assessment regarding the follow-up of the started approach changes.	Major	Done
3.	As a user, I want to indicate in a cell whether it is declared a dead end so that I can fall back on my assessment of whether a cell is a dead end, be it to remind it or to re-evaluate it.	Major	Done
4.	As a user, I want to add a fork declaration to a cell so that I can enrich cells with the information that from this cell on I will take a new approach and that this information can be used to update the tree diagram	Major	Done
5.	As a user, I want the tree diagram to update itself automatically when I make a change to the dead end declaration of a cell so that I can benefit directly from the information added when exploring and navigating within the notebook using the tree diagram.	Major	Done
6.	As a user, I want the tree diagram to update itself automatically when I make a change to the parent declaration of a cell so that I can benefit directly from the information added when exploring and navigating within the notebook using the tree diagram.	Major	Done
7.	As a User, I want I want to be able to reset the information on fork, dead end, data science steps, description for all cells at once or for a single cell so that for example, I can make a reassessment or that a new user can start his assessment on a tabula rasa.	Minor	Open

8.	As a user, I want to have access to a description that explains the benefits, the use and the individual features of the extension so that I get an insight into what I can and cannot do with this extension and that I can look it up if I encounter problems or have any questions when using the extension.	Minor	Open
9.	As a user, I want to be able to make changes to the settings (e.g. regarding appearance, positioning, defaults values) for all my notebooks as well as for individual notebooks, whereby the notebook-specific ones are given priority so that I can optimally use the extension according to my preferences and needs, (notebook specifically).	Minor	Open
10.	As a user, I want to change defaults values for example whether all, none or the proposed data science steps should be selected initially so that I can hopefully save time due to the default values that are adjusted to my needs.	Minor	Open
11.	As a User, I want to be able to choose some of the colors used so that I can adapt them to my preferences and any visual impairments.	Minor	Open
12.	As a User, I want to be able to change the font sizes of the extension windows myself so that I can adapt it to my preferences and any visual impairments.	Minor	Open
13..	As a User, I want to be able to minimize and expand the extension window(s), individually or all at once, so that depending on my current activity, I can decide for myself how much space I want to make available to the extension window(s).	Major	Done
14.	As a user, I want the selected data science steps to be displayed in the cell so that I can fall back on my selection be it to remind it or to re-evaluate it.	Major	Done
15.	As a user, I want the tree diagram to update itself automatically when I make a change to the description of a cell so that I can benefit directly from the information added when exploring and navigating within the notebook using the tree diagram.	Major	Done
16.	As a user, I want the description to be displayed in the cell so that so that I can refer to my description, be it to remind it, to supplement it or to correct it.	Major	Done
17.	As a User, I want to be able to change the description so that so that I can continuously adapt, expand and correct the description and correct spelling mistakes.	Major	Done
18.	As a user, I want to remove a description from a cell so that I can undo the description when it no longer applies to a cell.	Major	Done
19.	As a user, I want to add a description to a cell so that I can enrich cells with my considerations that are not found in the code or comments, for example, the rationale for choosing a specific algorithm and that this information can be used to update the tree diagram	Major	Done

20.	As a user, I want the tree diagram to update itself automatically when I make a change to the data science step labels of a cell so that I can benefit directly from the information added when exploring and navigating within the notebook using the tree diagram.	Major	Done
21.	As a User, I want to get a suggestion which of the available data science steps could apply to a cell so that I am supported in the selection and I can save time when tagging using possible additional features such as a quick selection of the suggested data science steps or by setting them as the default selection.	Major	Done
22.	As a User, I want to be able to select all of the data science steps available for selection with one click so that I can save time when tagging if all or very many of these steps apply to a cell.	Major	Done
23.	As a User, I want to remove data science step labels from a cell so that so that I can continuously adapt this information if, for example, the code changes or my assessment of which data science steps a cell focuses on changes.	Major	Done
24.	As a User, I want to select data science steps labels for a cell from a predefined set of labels so that notebook internal and notebook-wide standard labeling is followed, which should increase the comprehensibility of notebooks and simplify comparative analyzes in the research of notebook workflows.	Major	Done
25.	As a User, I want to add labels to a cell that reflects the data science steps on which the cell is focusing so that I can enrich cells with this information to improve the intelligibility of the notebook and that this information can be used to update the tree diagram.	Major	Done
26.	As a User, I want to handle all cells that follow a cell declared as a fork up to the next cell that is declared a fork as belonging to the same branch so that I can save myself the manual effort for this very likely frequent case.	Major	Done
27.	As a User, I want to change the parent node currently assigned to a cell declared as a fork by the user so that so that I can continuously adapt this information if, for example, the code changes or my assessment of the hierarchy of the approaches followed.	Major	Done
28.	As a User, I want to display the parent node currently assigned to a cell declared as a fork within the cell so that I can fall back on my assessment of whether a cell is a fork, be it to remind it or to re-evaluate it.	Major	Done
29.	As a User, I want to assign as parent node to a cell, declared as a fork, by default its preceding cell so that I can save myself the manual effort for this very likely frequent case.	Major	Done

30.	As a user, I want the tree diagram to update itself automatically when I make a change to the fork declaration of a cell so that I can benefit directly from the information added when exploring and navigating within the notebook using the tree diagram.	Major	Done
31.	As a user, I want to indicate in a cell whether it is declared a fork so that I can fall back on my assessment of whether a cell is a fork, be it to remind it or to re-evaluate it.	Major	Done
32.	As a user, I want to remove a fork declaration from a cell so that I can undo the declaration as fork if, for example, the code in the cell or my assessment regarding the independence of the approach followed in the following code block changes.	Major	Done

Table A.1: User Stories

A.2 Dashboard

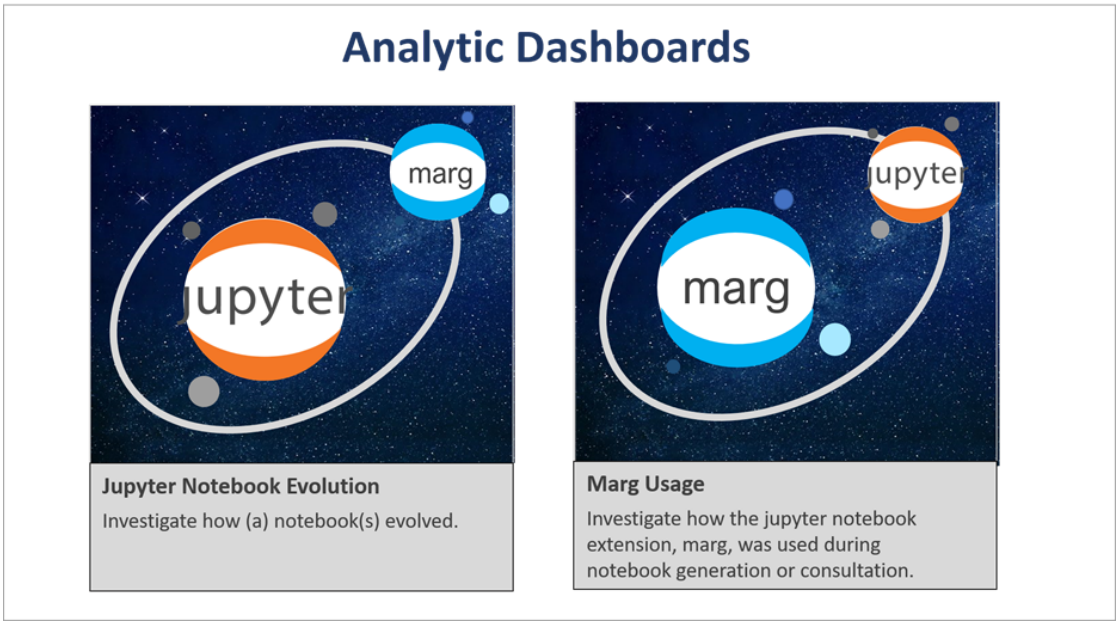

Figure A.1: Dashboard Goal Selection

A.3 Code Snippets

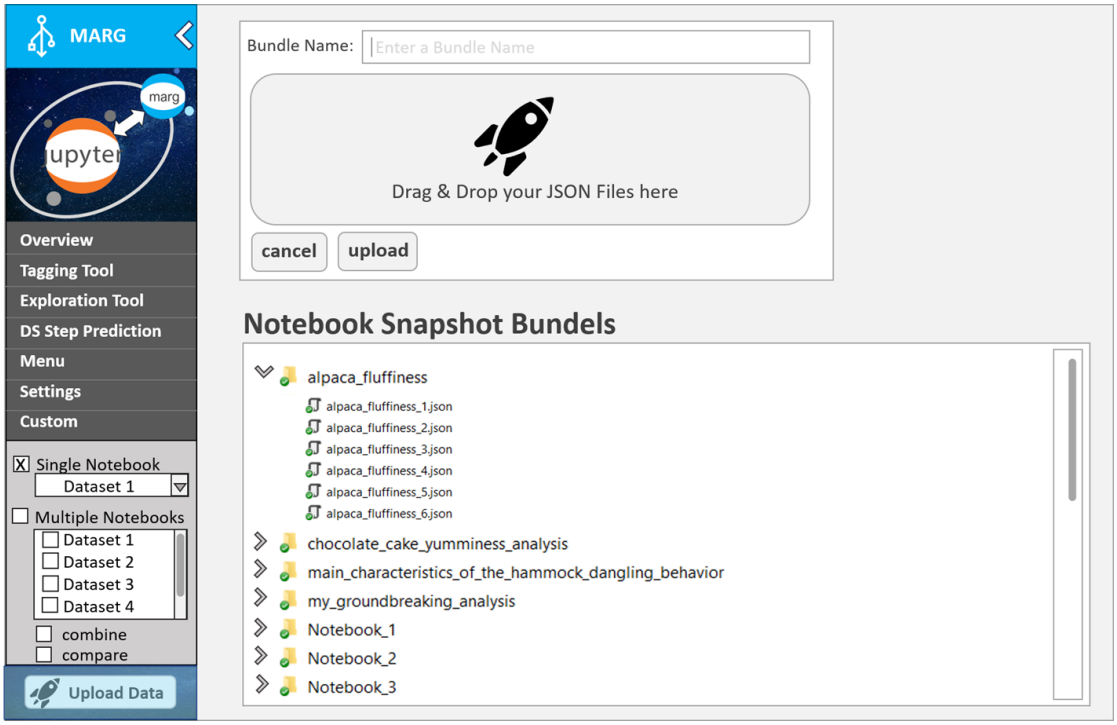

Figure A.2: Dashboard Data Upload

```
// for each affected node set the closest preceding cell that does not create a circle as the parent
for (let i = 0; i < non_fork_non_root_nodes.length; i++) {
    // test each preceding node in descending order
    // adjust to array enumeration starting from 0
    for (let j = non_fork_non_root_nodes[i] - 1; j > 0; j--) {
        if (has_no_cycle(non_fork_non_root_nodes[i], j)) {
            cells[non_fork_non_root_nodes[i] - 1].metadata["MARG_parent_node_id"] = j
            // adjust to array enumeration starting from 0
            cells[non_fork_non_root_nodes[i] - 1].metadata["MARG_parent_cell_id"] = cell_metadata[j - 1]["MARG_cell_id"]
            break
        }
    }
}
```

Figure A.3: Implementation of the Algorithm to Set Implicit Parents

```
function detect_root() {
    console.log(globals.log_prefix + "detect root");

    // get metadata from all cells of the notebook
    let cells = Jupyter.notebook.get_cells();
    let cell_metadata = cells.map(({_metadata}: any) => (_metadata));
    // @ts-ignore
    let cell_parent_nodes = cell_metadata.map(({MARG_parent_node_id}) => (MARG_parent_node_id));
    let root_node: number;
    // take first cell as parent if it is not a fork, i.e. if it has no parent
    if (!cell_parent_nodes[0]) {
        root_node = 1;
    }
    // take smallest given parent that is not a fork itself as parent, i.e. if it has no parent
    else {
        // determine given parents and cells that have no parent
        let non_fork_nodes = cell_parent_nodes.reduce((arr: any, e: any, i: any) => ((!e) && arr.push(i + 1), arr), []) //
    adjust to Marg enumeration starting from 1, not 0 to be in line with parent enumeration
        let parent_node = cell_parent_nodes.reduce((arr: any, e: any, i: any) => ((e) && arr.push(e), arr), [])
        // sort the cells in ascending order so that, when queried later, the first element that meets a condition matches
    the smallest parent that meets that condition.
        parent_node.sort((a: any, b: any) => a - b);
        // find smallest given parent that is not a fork itself
        root_node = parent_node.find((parent: any) => non_fork_nodes.includes(parent));
    }
    return root_node
}
```

Figure A.4: Implementation of the Root Detection Algorithm

```
function prevent_from_loops(node_to_set_parent: number, possible_parent_node_id: number, current_parent: number, is_parent_being_increased: boolean) {
    /*
    prevent a cell's parent from being set to a value that would create a graph containing a circle
    case 1 (self as parent): Don't allow to set the parent of cell x to cell x
    case 2 (direct fork as parent): cell x has parent y. Don't allow to set the parent of cell y to cell x
    case 3 (any degree of fork's fork as parent): cell a has parent b, cell b has parent c, cell c has parent d. Don't allow to set the parent of cell d to
cell a, b or c
    */
    // set the old parent, still stored in the metadata, as the approved parent
    let approved_parent: number = current_parent;

    // prevent creating a graph containing a circle
    if (!has_no_cycle(node_to_set_parent, possible_parent_node_id)) {
        if (is_parent_being_increased) {
            for (let i = possible_parent_node_id + 1; i < Jupyter.notebook.ncells() + 1; i++) {
                if (has_no_cycle(node_to_set_parent, i)) {
                    approved_parent = i;
                    break
                }
            }
        } else {
            for (let j = possible_parent_node_id - 1; j > 0; --j) {
                if (has_no_cycle(node_to_set_parent, j)) {
                    approved_parent = j;
                    break
                }
            }
        }
    } else {
        approved_parent = possible_parent_node_id;
    }
    return approved_parent
}
```

Figure A.5: Implementation of the Cycle Detection Algorithm

List of Figures

List of Tables