Reproducible HTML Notebooks

Computer Science Research Methods
INFO4990
Semester 1, 2023
Assignment 2

Tanaka Chitete

Supervisor:
Dr. Rahul Gopinath

The University of Sydney
School of Computer Science

April 26, 2023

Contents

Chapter 1

Literature review

Reproducibility represents perhaps the most essential aspect in any science discipline and sharing someone's work by means of a computational notebook [source] is meaningful only to the extent which the notebook can be reproduced. With this in mind, we have identified that the high-level process of achieving reproducibility and extensibility within the context of Jupyter Notebooks consists of replicating notebook state, sharing a ready-to-run notebook, and collaborating on a single notebook. To begin, we enumerate the complexities associated with saving and restoring a notebook. Thereafter, we discuss the complications related to sharing notebooks in a lightweight manner. At last, we evaluate means for facilitating collaboration on a single notebook.

1.1 Replicating notebook state

1.1.1 Introduction

Existing literature has achieved state replication in a variety of different manners with careful consideration to specific contexts of use. The prevailing methods are either *snapshot-based* or *provenance-based*. Snapshot-based solutions focus on capturing notebook state at a specific point in time during the computational workflow. Provenance-based solutions, on the other hand, focus on capturing information pertaining to the entire computational workflow–inputs, processes, and outputs–and documenting their evolution throughout the entire experiment. Regardless, achieving replicability, irrespective of the means, represents a crucial first step towards furthering reproducible science with Jupyter Notebooks.

1.1.2 Taking snapshots or capturing the whole tale

When examined in detail, there are several challenges associated with replicating the state of a notebook. Not only must the associated code be fully replicated, so to must the underlying Python run-time environment. Concerning the environment, subtle problems arise in replicating referenced data, resolving external library dependencies, and re-initialising variable states. In this section, we shall discuss snap-shot based solutions in tandem with other provence-based works.

Introduction: In order to faithfully reproduce notebook state, computational notebook developers must devise a means to re-initialise runtime variables, resolve external library dependencies, and replicate local referenced data. Existing works concerned with achieving reproducibility with Jupyter Notebooks are either *snapshot-based* [1, 2] or *provenance-based* [3, 4]. **Approach 1**: To accomplish the objective of absolute notebook state capture, Wannipurage et. al. [1] utilised mechanisms contained within the Jupyter standard library and the IPython kernel to create a savable archive of a running notebook–capturing all of the information required to reproduce notebook state in its entirety. In a similar fashion, however, under the context of live-migrating to different environments, Juric et. al. [2] present a means to freeze an active Jupyter notebook Notebook to persistent storage, and efficiently restore it to memory on-demand. **Approach 2**: However, not all approaches to achieving reproducibility are snapshot-based. Pimentel et. al. [3, 4], employ a provenance-based approach. With their solution, noWorkflow, the authors seek to demonstrate the differing levels of specificity at which provenance can be collected. **Conclusion**: The use cases of provenance-based workflows are extensive. Such workflows have use cases which include verifying inconclusive results to managing experiment evolution. Ostensibly, supporting restoration through provenance would entail significantly more work than achieving the same end through snap-shotting. Therefore, for this project, we will focus on implementing a snapshot based approach. However, extending to provenance-based approach would be a candidate for future work.

1.1.3 Conclusion

...

1.2 Sharing ready-to-run notebooks

1.2.1 Introduction

...

1.2.2 Resolving dependencies statically or dynamically

Introduction: "Dependency hell" is a problem that plagues essentially every piece of software at some point during its life-cycle. Jupyter notebooks are no exception. Whilst each of these software packages prove to be excellent in their usage, grave problems arise when attempting to execute a notebook which uses some of these packages, on a different system/environment. More often than not, the receiver attempting to use the notebook won't have the complete set of packages installed. Resultantly, they encounter roadblock after roadblock attempting to resolve these dependency issues, if they are even able to do so. With this pain point in mind, finding a means to ameliorate the problem of dependency hell would prove to be crucial in catering to those users without a background in software engineering. Existing works concerning dependency resolution are either *static* [5] or *dynamic* [6].

Approach 1: With *SnifferDog*, Wang et. al. [5] expertly achieve dependency resolution. The solution, SnifferDog, upon receipt of a Jupyter notebook as input, determines and installs the necessary packages, and ultimately, verifies if the notebook is executable and reproductive of the original results. **Approach 2**: However, Zhu et. al. [6] employed an alternative, dynamic approach. Employing an iterative approach driven by the occurrence of runtime errors, RELANCER identifies and fixes deprecation issues one package at a time. **Conclusion**: Through associated performance analyses, it is made quite apparent that RELANCER is lowly-performant in comparison to SnifferDog. Whilst both solutions are fundamentally different, the time overhead associated with catching run-time errors and resolving them is far more costly than a dynamic approach. As such, we shall aim to achieve dependency resolution through dynamic means.

1.2.3 Conclusion

...

1.3 Writing notebooks in collaborative settings

1.3.1 Introduction

...

1.3.2 Supporting version-control or synchronous editing

Introduction: Facilitating collaboration represents a natural next step from achieving faithful state capture and replication. Through reviewed literature, we uncovered that existing solutions implement either *version-control* or *synchronous-editing*. **Approach 1**: To accomplish the objective of enabling collaboration, Kery et. al. [7] utilised implemented both automatic and absolute versioning with their notebooks. It should be noted, however, Verdant has only been provisioned for use by a single notebook user. Thus, it cannot be used with versioning software such as Git, right out of the box. **Approach 2**: Conversely, synchronous editing has been viewed as a viable means to provision for collaboration. Tools like Google Colab source have demonstrated the possibility for synchronous editing — multiple users are able to edit the same notebook and changes are updated in real-time, which may revolutionize the ways data scientists collaborate [8]. Curious to observe the implications on data science, the authors proceeded to conduct studies. They found that synced notebooks can result in interference, lack of awareness, and privacy concerns. These issues could ultimately dissuade users, and, in turn, computational notebook developers, from developing and/or supporting synchronous editing, at all. In wake of these findings, the authors then go on to highlight that perhaps local version control could be a potential solution in assisting collaborators to track each others' edits. In addition, the complexity involved with enabling synchronous editing would be far too great for the scope of an honours project. **Conclusion**: That said, imple-

menting version-control would be the preferred means for collaboration. However, synchronous-editing presents a very interesting avenue for future work.

1.3.3 Conclusion

...

In terms of versioning, where does data science programming diverge from any other form of code development? Typically in regular code development, the primary artifact that a programmer works with is code [8]. Data science programming relies on working with a broader range of artifacts: the code itself, important details within the code [4], parameters or data used to run the code [9], visualizations, tables, and text output from the code, as well as notes the data scientist jots down during their experimentation [10].

Chapter 2

Research approach

2.1 Contributions

I intend to make the following contributions:

1. Saving and restoring partially computed Jupyter Notebooks.
2. Lightweight sharing of Jupyter Notebooks–inclusive of all dependencies.
3. Provisioning of multi-user collaboration through the supporting of version-control.

2.2 Methods

...

2.3 Evaluation

...

2.4 Conclusion

...

Bibliography

(1) D. Wannipurage, S. Marru and M. Pierce, "A Framework to capture and reproduce the Absolute State of Jupyter Notebooks", *PEARC 2022 Conference Series - Practice and Experience in Advanced Research Computing 2022 - Revolutionary: Computing, Connections, You*, 2022, DOI: 10.1145/3491418.3530296.

(2) M. Juric, S. Stetzler and C. T. Slater, "Checkpoint, Restore, and Live Migration for Science Platforms", 2021.

(3) J. F. Pimentel, L. Murta, V. Braganholo and J. Freire, "noWorkflow: a tool for collecting, analyzing, and managing provenance from Python scripts", *Proceedings of the VLDB Endowment*, 2017, **10**, 1841–1844.

(4) J. Felipe, N. Pimentel, V. Braganholo, L. Murta and J. Freire, "Collecting and Analyzing Provenance on Interactive Notebooks: when IPython meets noWorkflow".

(5) J. Wang, L. Li and A. Zeller, "Restoring Execution Environments of Jupyter Notebooks", *Proceedings - International Conference on Software Engineering*, 2021, 1622–1633.

(6) C. Zhu, R. K. Saha, M. R. Prasad and S. Khurshid, "Restoring the Executability of Jupyter Notebooks by Automatic Upgrade of Deprecated APIs", *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, 2021, 240–252.

(7) M. B. Kery and B. A. Myers, "Interactions for untangling messy history in a computational notebook", *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2018, **2018-October**, 147–155.

(8) A. Y. Wang, A. Mittal, C. Brooks and S. Oney, "How data scientists use computational notebooks for real-time collaboration", *Proceedings of the ACM on Human-Computer Interaction*, 2019, **3**, DOI: 10.1145/3359141.