ReVISitPy: Python Bindings for the reVISit Study Framework

H. Shrestha[1], J. Wilburn[2], B. Bollen[2], A. M. McNutt[2], A. Lex[2], and L. Harrison[1]

[1]Worcester Polytechnic Institute, Worcester, Massachusets, USA
[2]University of Utah, Salt Lake City, Utah, USA

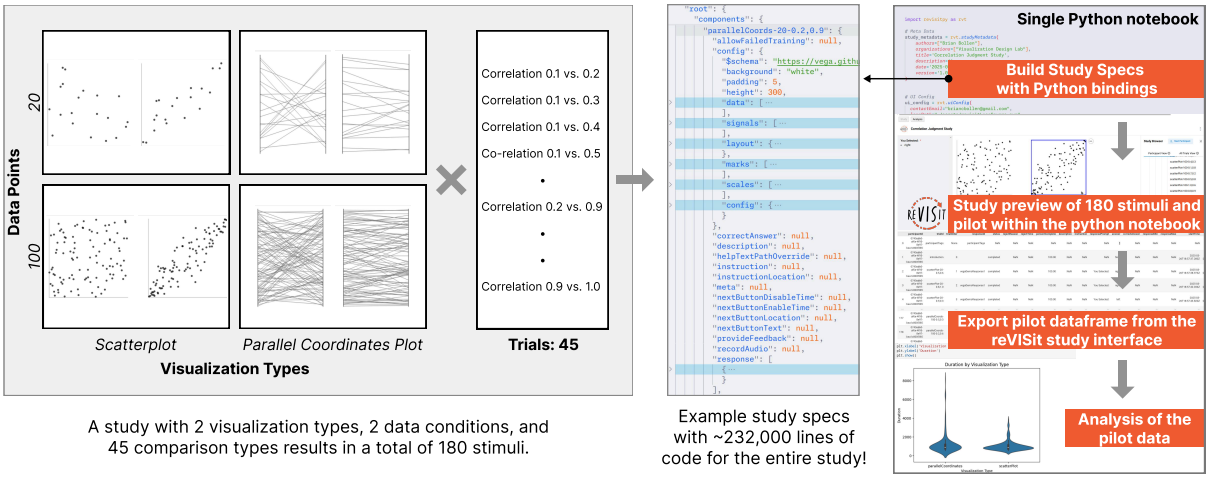

Figure 1: *ReVISitPy provides easy-to-use functions for developing, testing and performing preliminary analysis of an entire study from within a single Python notebook. ReVISitPy builds on reVISit, a web-based study framework supporting complex visualization research experiments.*

Abstract

User experiments are an important part of visualization research, yet they remain costly, time-consuming to create, and difficult to prototype and pilot. The process of prototyping a study—from initial design to data collection and analysis—often requires the use of multiple systems (e.g. webservers and databases), adding complexity. We present reVISitPy, a Python library that enables visualization researchers to design, pilot deployments, and analyze pilot data entirely within a Jupyter notebook. ReVISitPy provides a higher-level Python interface for the reVISit Domain-Specific Language (DSL) and study framework, which traditionally relies on manually authoring complex JSON configuration files. As study configurations grow larger, editing raw JSON becomes increasingly tedious and error-prone. By streamlining the configuration, testing, and preliminary analysis workflows, reVISitPy reduces the overhead of study prototyping and helps researchers quickly iterate on study designs before full deployment through the reVISit framework.

CCS Concepts

• ***Human-centered computing*** → ***Visualization toolkits; User studies;***

1. Introduction

Visualization researchers often design and run user studies, requiring them to preview, test, and pilot approaches for data collection and analysis. Typically, this workflow involves generating study data in environments like Python, designing stimuli, testing the study in a browser-based platform, collecting pilot data, and then importing it back to Python for analysis. This fragmented process is time consuming and requires heavy back-and-forth movement of data across tools and languages.

Moreover, visualization studies can quickly grow complex and long, especially when involving multiple conditions and data combinations. For example, a user study with two visualizations, each tested across two conditions with 45 different data combinations results in a total of $2 \times 2 \times 45 = 180$ unique stimuli. When each stimulus is specified using fairly common approaches of Domain-Specific Languages (DSLs) such as Vega [SRHH16] and Vega-Lite [SMWH17], the combined configuration of all stimuli–such as one used in a reVISit study framework [DWS*23]–can become

DOI: 10.2312/evp.20251133

overwhelmingly large, especially when incorporating components like interactions and data. Such specifications often consist of tens of thousands of lines of repetitive JSON code, making them difficult to maintain, extend, or debug. Manually creating or managing such specifications becomes nearly impossible at scale.

However, a key strength of JSON DSLs—and their JSON Schema based specifications [McN23]—is their portability, which enables integration with environments like Python. Frameworks such as Vega-Altair [VGH*18] illustrate how Python can be used to generate DSL based visualizations offering a more flexible and user-friendly experience for designers.

To address these challenges, we developed reVISitPy (pypi.org/project/revisitpy), a Python wrapper for the ReVISit DSL. ReVISitPy simplifies the creation and management of complex visualization studies by streamlining the configuration process within the familiar Python ecosystem. We designed reVISitPy to reduce the burden of generating and managing large specs and to allow designers to rapidly prototype studies within a single notebook UI—capturing the configuration of studies, their preview, and early-stage data analysis. This pipeline aims to lower to barrier of conducting complex visualization studies quickly.

2. ReVISitPy

ReVISitPy is a Python package which wraps the standard items of the reVISit configuration file with readable, easy-to-use functions. It provides a factory function for each top-level item in the configuration, covering elements such as the study interface, metadata, stimuli, and the sequencing of stimuli shown to participants.

Consider designing a just-noticeable difference (JND) study (e.g., [HYFC14]) that varies across two visualization types–scatter plots and parallel coordinates plots–along with different data sizes (e.g., 20 and 100 points) and multiple JND values. With reVISitPy, a study designer first defines a dataset representing all possible combinations of JND values. Next, they create a sequence by permuting the three factors: visualization type, data size, and correlation (see Figure 2). Finally, the *permute* method is chained with a function that maps each condition set generated from the permutation to a specification for rendering the study component.

The resulting configuration file is large, 232,783 lines of code. This is too large to feasibly create by hand without introducing errors from copying and pasting. ReVISitPy affords editing config files such that a study designer can modify pieces of the study without having to search through the resulting file—supporting experimental transparency long-term. This example is available at (github.com/revisit-studies/revisitpy-examples).

A crucial design alternative would be to embed different forms of abstraction within the specification itself—for instance variables (à la Canis [GZL*20]) or loops (à la ColorBuddy [MSH24]) might reduce some of the repetition latent to this context. However, we found that nuances in experiment design were difficult to capture through the design of the language itself—for example generating data for stimuli or managing the combinatorics of more complex study designs—which may be better suited for widely used languages (e.g., Python) which already support these types of computation well. Instead, ReVISitPy adopts a binding style similar to Vega-Altair [SMWH17] in the context of Python's general purpose programming environment, enabling experiment designers to leverage the full expressiveness of general-purpose programming languages (e.g., variables, complex control flow (including loops)). To ensure the visualization developers (who are often also the experiment designers) can easily get familiar with the library, we borrowed Altair's structured method-chaining approach [VGH*18]. We focus on this design, because it allows us to keep our focus on doing one thing well (specification and deployment of crowd work user studies).

```
dataSet = [{"corrValues": [x / 10, y / 10]}
           for x, y in itertools.combinations(range(1, 11), 2)]

main_sequence = rvt.sequence(order="fixed")
main_sequence.permute(
    factors=[{"vis": "scatterPlot"}, {"vis": "parallelCoords"}],
    order="latinSquare",
).permute(factors=[{"points": 20}, {"points": 100}], order="fixed"
).permute(factors=dataSet, order="random"
).component(component_function)

sequence = rvt.sequence(order="fixed",
                        components=[introduction]) + main_sequence

study = rvt.studyConfig(
    schema="...", uiConfig=ui_config,
    studyMetadata=study_metadata, sequence=sequence)
```

Figure 2: *Example of 3 permutations chained together to build a total of 180 stimuli. ReVISitPy supports straightforward specification of experiments using a familiar notebook environment.*

3. Preview and Analysis

We built reVISitPy on top of the anyWidget toolkit [MAG24] to enable study designers not only to create specifications but also to preview and test studies directly within Jupyter notebooks. After testing the study preview, collected data can be exported back to the notebook as a *Pandas Dataframe*, allowing designers to inspect the results and perform preliminary analysis. For example, in Figure 1, the study designer can from a single notebook 1) change the number of correlation levels tested or the correlated data generation function, 2) preview and take the experiment themselves to test the user experiment or data collection, and 3) immediately conduct analyses on data as it would be collected in a deployed study.

4. Future Work

This work is the first step in supporting experiment designers in the rapid prototyping stage. We plan to extend the library to support deployment of studies directly from within the notebook, extending prototyping from local evaluators to crowd sourced pilots. We aim to incorporate the ability to simulate study runs based on predefined models or hypotheses. These simulations can help verify whether the analysis pipeline behaves as expected and assist in identifying potential issues before actual data collection. It could be used to test counterbalancing mechanism—such as ensuring that reVISit's various randomization methods align with expecting segmentation. By making the process of experimental iteration smoother, we hope to enable better experiment designs in general.

References

[DWS*23] DING Y., WILBURN J., SHRESTHA H., NDLOVU A., GADHAVE K., NOBRE C., LEX A., HARRISON L.: revisit: Supporting scalable evaluation of interactive visualizations. In *2023 IEEE Visualization and Visual Analytics (VIS)* (2023), pp. 31–35. doi:10.1109/VIS54172.2023.00015. 1

[GZL*20] GE T., ZHAO Y., LEE B., REN D., CHEN B., WANG Y.: Canis: A high-level language for data-driven chart animations. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 607–617. 2

[HYFC14] HARRISON L., YANG F., FRANCONERI S., CHANG R.: Ranking visualizations of correlation using weber's law. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 1943–1952. doi:10.1109/TVCG.2014.2346979. 2

[MAG24] MANZ T., ABDENNUR N., GEHLENBORG N.: anywidget: reusable widgets for interactive analysis and visualization in computational notebooks. *Journal of Open Source Software 9*, 102 (2024), 6939. Publisher: The Open Journal. URL: https://doi.org/10.21105/joss.06939, doi:10.21105/joss.06939. 2

[McN23] MCNUTT A. M.: No Grammar to Rule Them All: A Survey of JSON-style DSLs for Visualization. *IEEE Transactions on Visualization and Computer Graphics 29*, 1 (Jan. 2023), 160–170. doi:10.1109/TVCG.2022.3209460. 2

[MSH24] MCNUTT A., STONE M. C., HEER J.: Mixing linters with guis: A color palette design probe. *IEEE Transactions on Visualization and Computer Graphics* (2024). 2

[SMWH17] SATYANARAYAN A., MORITZ D., WONGSUPHASAWAT K., HEER J.: Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan. 2017), 341–350. doi:10.1109/TVCG.2016.2599030. 1, 2

[SRHH16] SATYANARAYAN A., RUSSELL R., HOFFSWELL J., HEER J.: Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan. 2016), 659–668. doi:10.1109/TVCG.2015.2467091. 1

[VGH*18] VANDERPLAS J., GRANGER B. E., HEER J., MORITZ D., WONGSUPHASAWAT K., SATYANARAYAN A., LEES E., TIMOFEEV I., WELSH B., SIEVERT S.: Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software 3*, 32 (Dec. 2018), 1057. doi:10.21105/joss.01057. 2