



# plotID – a toolkit for connecting research data and visualization

Martin Hock <sup>1</sup>Hannes Mayr <sup>1</sup>Manuela Richter <sup>1</sup>Jan Lemmer Peter F. Pelz <sup>1</sup>

1. Chair of Fluid Systems, Technische Universität Darmstadt, Darmstadt.

**Date Submitted:**

2022-09-01

**Licenses:**This article is licensed under: **Keywords:**research data management,  
visualization, figure, plot, mapping,  
referencing, ID**Data availability:**

Data can be found here:

[matplotlib\\_example.py](#)**Software availability:**

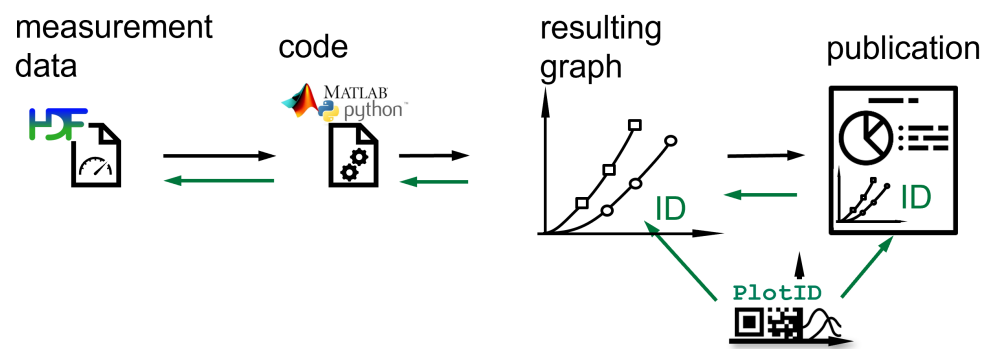
Software can be found here:

[git.rwth-aachen.de](#)[/plotid/plotid\\_python](#)

**Abstract.** While visualizations can carry a vast amount of information compared to text and are often used for validation, references to data and metadata resulting in these visualizations are not common. To provide such references, the software plotID provides two key modules that strive to seamlessly integrate into a generic, Python-based research workflow. The module *tagplot* generates or accepts a unique ID and anchors it (visibly) as a reference to a figure or picture. The module *publish* exports the figure along with the data, code and parameters used in its creation into folders named by the reference ID for later reuse. The tools work to provide aid in research data management with simple base functionality as opposed to encompassing management frameworks. Later features and improvements will expand the scope and applicability to other programming environments.

## 1 Statement of need

1 Scientific results are published in the form of hypotheses, axioms and equations as well as text  
2 and diagrams. Likewise, research software is being published more and more frequently. The  
3 comprehensibility of scientific results is indispensable for scientific discourse and reproducibility.  
4 Hypotheses, axioms and equations are usually published in text form and can be referenced  
5 accordingly. Software can be made traceable and referencable through the use of version control  
6 software. But what about diagrams? A diagram published in a paper is difficult to trace because  
7 the (raw) data is usually not available. However, the traceability of diagrams and the data they  
8 contain is not only a challenge in publication but also in everyday research. Diagrams are used for  
9 visualization and are therefore often produced for interim results. While the researcher continues  
10 the research process with investigations, experiments or simulations, volatile but important  
11 information like metadata, background information and details of the data processing are lost.  
12  
13 To be able to reconstruct the complete path, a treasure map is needed, starting from a publication,  
14 marking major landmarks of the process back to the raw data and metadata. This map needs  
15 to be provided along with the product that will be reviewed the most – the created diagram. If  
16 diagrams – regardless of whether they are published later or only serve as interim results – are



**Figure 1:** Research workflow from left to right; afterwards following the chain of references from right to left

'plotID-referencing' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

17 provided with an identifier which connects to previous steps, then traceability can be ensured.  
 18 Figure 1 shows the order in which crucial elements are created and how the reference chain  
 19 tracks back.

20 A tool designed to meet these needs must satisfy the following requirements:

- 21 • Diagrams must have a unique identifier.
- 22 • The identifier must reference the raw data, relevant metadata and the code used to process  
 23 the data.
- 24 • The method must be easy to implement into the existing research workflow.

25 To reduce the effort of organizing figures along with all necessary data and metadata for later  
 26 review and reuse, the tool plotID was developed. plotID meets all the above-mentioned require-  
 27 ments. The tool is limited to usage in an existing Python environment, but investigations on  
 28 enabling independent installation and execution or offering plotID as a web-based service are  
 29 ongoing. The software depends on multiple other Python libraries. It is currently limited to  
 30 visualizations from the Matplotlib-library [1] and general picture files such as PNG and JPG.

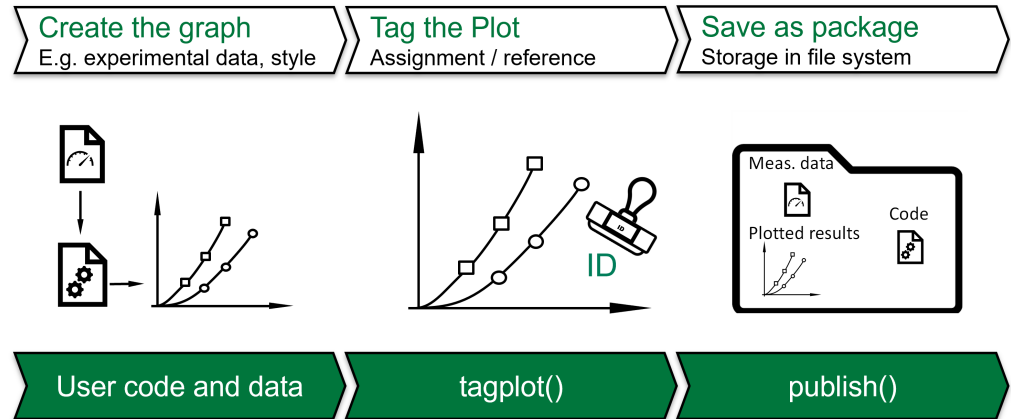
31 Researchers often tend to keep an Excel table, noting down manually which data file corresponded  
 32 with which result along with input parameters. Sometimes an ID system is used (counting up or  
 33 using the date), but interim results like visualizations – used to verify results – are usually not  
 34 included. Reviewing and understanding the environment of solutions used in and specifically  
 35 created for research data management (RDM) remains an ongoing process. The named products  
 36 in this paragraph are meant to provide some overview and examples but are by no means a  
 37 comprehensive or rated list. The reviewed solutions range from simple local scripts and libraries  
 38 (like plotID), backup and synchronization software (for filesystem like ZFS [2] and for folders  
 39 like rsync [3]), software version control (git [4], svn [5]) and software to extend on version  
 40 control (git LFS [6], git-fat [7], git-annex [8]) to better handle binary and large data up to  
 41 dedicated workflow management systems (DataLad [9], DVC [10], signac [11]). Another area  
 42 of solutions focuses on providing the working environment by integrating documentation with  
 43 code (Jupyter Notebooks [12]), providing bespoke and versioned Virtual Research Environments

44 (VREs) or offering programmable or fixed – often discipline-specific – schematics in Electronic  
45 Laboratory Notebooks (ELNs such as eLabFTW [13], RSpace [14]). With more comprehensive  
46 solutions and added functionality for sharing and exporting data, products lean more towards  
47 a client-server structure or even a fully hosted product with web and API interfaces. Many  
48 solutions are Open Source with Software as a Service (SaaS) offerings. Versioning often uses  
49 hashing algorithms for security reasons, thus providing unique identifiers for a specific state  
50 (snapshot) out of the box, although those are not always used for identification in user interfaces.  
51 Some hosted services implement filesystem-level software to equip each data resource with  
52 identifiers to track them independently of their current storage location (iRODs [15]). Structuring  
53 and organizing data is part of most RDM solutions and even rather strict ELN products offer to  
54 append files, images and comments to their organizational units (a probe or process). Export and  
55 sharing of research data along with its metadata is an integral part of most RDM solutions, and  
56 most offer more refined features and compatibility than plotID. DVC (Data Version Control) can  
57 create plots and visualizations as part of the versioning workflow as well as overlaying multiple  
58 versions to show differences between plotted results [16].

59 While the organization of data, metadata and code as much as identification, versioning and  
60 export could be found in several products, the unique feature of applying an ID **visibly** to a  
61 visual representation is not provided in any examined solution. With the big difference in scope,  
62 plotID could be implemented as part of a workflow complementing most of the above-mentioned  
63 solutions. Only some of the most restrictive ELNs or filesystem-level operations are unlikely to  
64 be compatible.

## 65 2 Methodology

66 The developed tool plotID is a software solution that covers the needs specified in section 1. The  
67 underlying concepts and methods of the software are independent of the programming language.  
68 The software aims to support the research workflow shown in Figure 2 and to enable traceability.  
69 plotID aims to help during the early research process to decrease the work of making publications  
70 reproducible later. To ensure ease of use, the tool has been designed to be integrated seamlessly  
71 into existing scripts. For this purpose, a graphical user interface (GUI) has been omitted. Instead,  
72 two main functions (building blocks) are provided, which can be inserted into existing user  
73 scripts as one-liners. They are the core of plotID. The first module creates a (unique) ID and  
74 stamps this ID onto an object containing a visualization, while the second module helps organize  
75 all relevant code, software, and data that went into creating this graphic, into one complete  
76 package. Furthermore, connectivity to existing identifiers is ensured. If a specific visualization  
77 is later chosen to be included in a publication, the ID can be replaced by a permanent identifier  
78 like a DOI and the package of code, software and data can be published at the location referenced  
79 by the DOI. The ID in the published paper will then directly reference the data, software and  
80 code used to create it, thus promoting reproducibility. In the following, plotID is presented in  
81 more detail using the Python implementation.



**Figure 2:** Workflow integrating the plotID core functions

'plotID-workflow' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

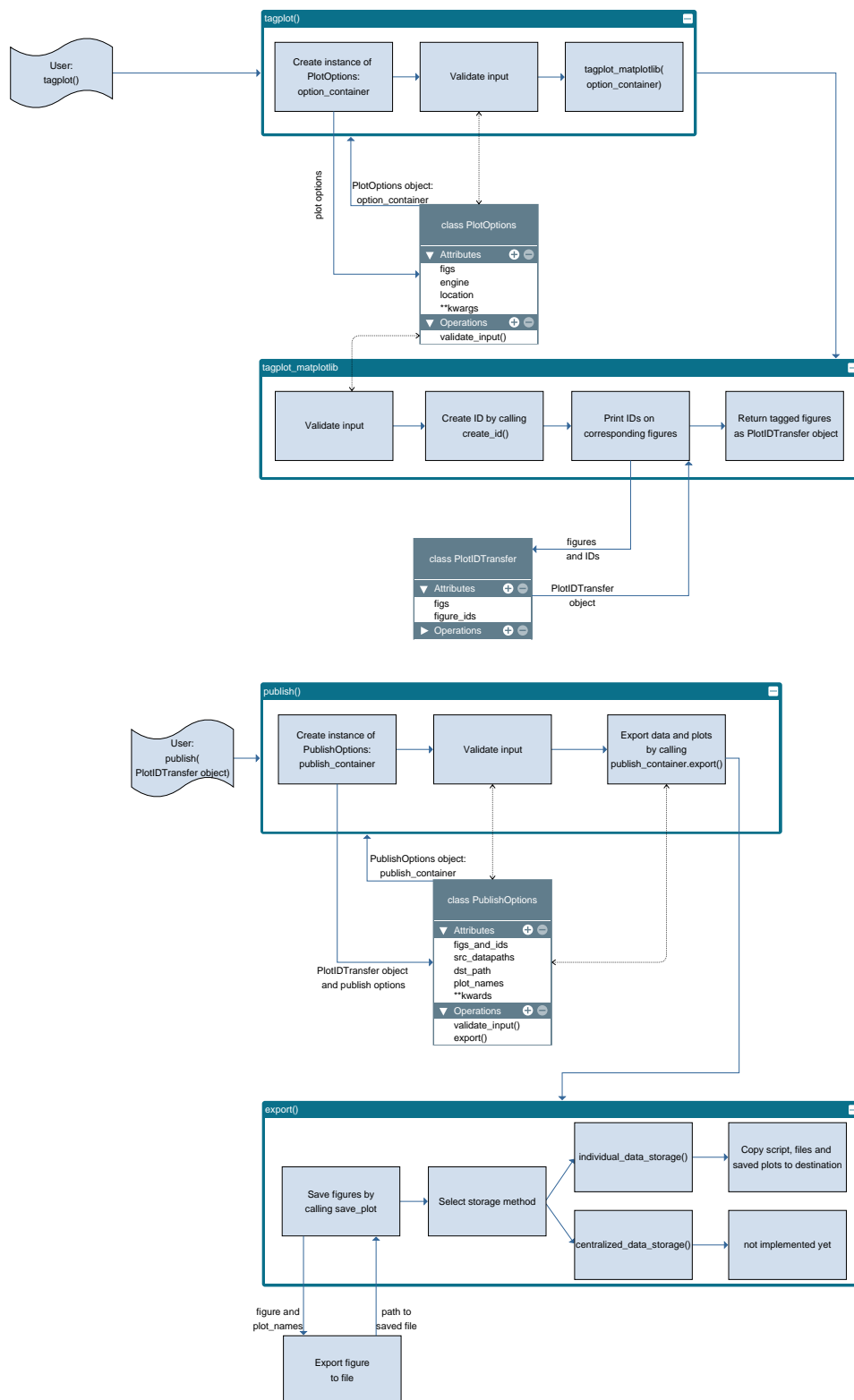
## 82 3 Python – Implementation

83 The first version of plotID was implemented in MATLAB since this is the most widely used  
 84 programming language in the local working environment and the language the authors had the  
 85 highest familiarity with. After reaching a usable state, the focus shifted to rewriting the tool in  
 86 Python, the second most used programming language (locally). The goal was to make plotID  
 87 accessible to a broader audience. Globally, Python is a lot more popular than MATLAB with a  
 88 currently 15 times higher rating on the TIOBE index [17]. Moreover, in contrast to MATLAB,  
 89 Python better fulfils the requirements for reusable software in the sense of the FAIR<sup>1</sup> principles  
 90 as defined by the Force11 group [18], described for software specifically by Chue Hong et  
 91 al [19]. Although MATLAB code can be (and often is) Open Source as well this proprietary,  
 92 commercial software needs to be paid for. This diminishes its accessibility even if older versions  
 93 are archived properly and provided by the company. In addition to being widely used in the  
 94 engineering and research community, Python is non-proprietary, Open Source, easy to install  
 95 and even shipped along many operating systems. Python also offers a package index (*PyPI* [20]  
 96 and installer (*pip* [21]) for easy distribution of software packages.

## 97 4 Core functions

98 The core functions of plotID are *tagplot()* and *publish()*. *tagplot()* generates an ID and adds this  
 99 ID to the figure object creating a new container object. *publish()* takes this container object to  
 100 bundle the figure, the script file, which plotID was called from and the processed data files and  
 101 stores them together in a folder named with the ID. In addition, the script is parsed and a list of  
 102 required dependencies is generated and added as a text file compatible with the Python package  
 103 installer [21]. New features might bring additional steps with the further development of plotID  
 104 and a widening of its scope.

1. FAIR: Findable, Accessible, Interoperable, Reusable



**Figure 3:** System architecture diagram 'plotID-system-architecture' by Hannes Mayr, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

## 105 4.1 tagplot()

106 The *tagplot()* function creates an ID and tags the figure object with this ID.

### 107 4.1.1 ID

108 *tagplot()* creates a unique ID (unique in a local system), that consists of a static prefix and a  
109 generated part. The prefix parameter is meant to be used to identify a project or organizational  
110 unit to which the figure is assigned. The generated part is by default created from the UNIX  
111 timestamp in hexadecimal form. As an alternative option, a random number generator can be  
112 used. As an alternative to generated IDs, full IDs can be input as a keyword argument, replacing  
113 the generated ID. The implementation of the ID is modular, easing the integration of individual  
114 needs or sources for IDs. Optionally the ID can be encoded into a QR code for improved machine  
115 readability.

### 116 4.1.2 Tagging

117 In Python, there are multiple available packages that can produce visualizations from data.  
118 Adding an ID needs to be implemented for many of these engines separately. For now, plotID  
119 supports figures created with *Matplotlib* and raw image files. The ID is added as an attribute to  
120 the Python object and the graphical, visible item.

### 121 4.1.3 Arguments

122 Necessary input arguments for *tagplot(figs, engine [, \*\*kwargs])*:

- 123 • *figs*: the figure object or a list of objects, that is to be tagged
- 124 • *engine*: the plot/image engine to be used (currently only 'matplotlib' and 'image' (for plain  
125 image files) are supported)

126 Important optional input arguments are:

- 127 • *figure\_ids*: IDs that will be printed on the plot. If empty, IDs will be generated for each  
128 plot. If this option is used, an ID for each plot has to be specified. Default: empty list. Type:  
129 list of strings.
- 130 • *location*: Location for ID to be displayed on the plot. The default is 'east'. Type: string.
- 131 • *rotation*: Rotation of the printed ID in degree. Overwrites the value defined by location.  
132 Type: float or int.
- 133 • *position*: Position of the ID given as (x,y). x and y are relative coordinates with respect to  
134 the figure size and must be in the interval [0,1]. Overwrites the value defined by location.  
135 Type: tuple of float.
- 136 • *prefix*: Will be added as prefix to the ID. Type: string.
- 137 • *id\_method*: Creating an ID by Unix time is referenced as 'time', creating a random ID  
138 with *id\_method='random'*. The default is 'time'. Type: str, optional
- 139 • *qrcode*: Encode the ID in a QR code on the exported plot. Default: False. Type: boolean.

140 • *id\_on\_plot*: Print ID on the plot. Default: True. Type: boolean.

141 There are additional arguments for custom positioning and font of the ID and adjusting  
142 the size and position of the QR code.

143 The function's output is a *PlotIDTransfer Object*) which provides a compatible method to transfer  
144 the output of all plot engines with additional information, most importantly the ID.

145 At this point the figure object inside can still be modified, for example, to adjust colours and  
146 positioning or to recreate the full plot before exporting a final version.

## 147 4.2 publish()

148 This function starts the export process. The source files of the processed data, the visualization  
149 (including the tagged ID), and the script hosting the call to the *publish* function are copied  
150 together into a destination folder.

### 151 4.2.1 Script

152 A function in Python has access to the file path of the script which it was called from. With this,  
153 the code for calculations can easily be collected. For this reason, *publish()* cannot be called from  
154 the command line or from within a script that has been started with the 'python -m' flag.

155 For dependent packages, the python compiler can report imported modules with the installed  
156 version. Those are then written into a file named "required\_imports.txt". This file can be directly  
157 read by the Python package installer to install the code's dependencies. The import for the plotID  
158 package is removed from this list, and in the script file calls to plotID functions are changed to  
159 comments, to avoid unnecessary exports. Furthermore, the user has to take care of the necessary  
160 Python version (if restrictions apply) and of including additional function files as data paths, if  
161 they have not been imported but are still accessed by the executed script.

### 162 4.2.2 Data files

163 Data files are handed over as a list of file or folder paths. Ideally, the script already manages  
164 a list of all files that are read during the execution of the script. It is up to the user to control  
165 this. By default, the data files are copied to each exported package. For large data files, the  
166 'centralized' flag is intended. It is up to the user to decide which resources to add to the export,  
167 by placing them in the list of data paths or not.

168 By default, the data files are copied into each export. The 'centralized' selection is optional. With  
169 this, the data is copied to a central folder, relative to the export packages. For further exports,  
170 the data files are compared to the ones already present and only copied if new data files are  
171 selected. With this, a publication on a data repository could encompass the data files in addition  
172 to multiple "satellite" folders containing the specific script, parameters and graphics. For HDF5  
173 files, each package can contain an empty HDF5 file that only contains a link to the "real" central  
174 data file. While this has proven to be useful in the MATLAB implementation, the Python version  
175 aims to include the 'centralized' option in a future release.

176 Remote files on network drives are handled just like local files and while HTTP(s) URLs currently  
177 lead to a FileNotFound error, they will be supported in a future release. It is recommended  
178 to not add large data or data available from acknowledged repositories into packages meant  
179 for publication. plotID will not support additional data or transfer protocols. The exported  
180 script should suffice to reference and showcase the usage of remote data sources. Additional  
181 commentary or documentation can be added to the export as a data file.

### 182 4.2.3 Arguments

183 Necessary input arguments for `publish(figs_and_ids, src_datapath, dst_path [, **kwargs])` are:

- 184 • `src_datapath`: This can be a single or a list of file or folder paths for source data and  
185 additional function files. The type is a string or a list of strings.
- 186 • `dst_path`: This is the destination folder path. If it does not exist, the folder will be created.  
187 The type is a string.
- 188 • `figure`: This is a figure object, the exact class depends on the plot engine used. This object  
189 will be turned into an image file.

190 Optional input arguments:

- 191 • `data_storage`: Currently only 'individual' and 'centralized' are available. 'Individual' will  
192 store all data in each exported package, while 'centralized' stores the data files in a central  
193 folder separate from the packages containing script and image files. To be implemented.  
194 Type: string or file path.
- 195 • `plot_name`: This is the name for the graphics objects. The type is a string or list of strings.  
196 If a single name is passed for multiple objects, a raising number will be added. If no name  
197 is passed, the ID will be used as the file name. Type: string or a list of strings.

## 198 5 Example script

199 The following script shows how plotID is used.

```
200 10 # %% Import modules
201 11 import numpy as np
202 12 import matplotlib.pyplot as plt
203 13 from plotid.tagplot import tagplot
204 14 from plotid.publish import publish
205 15
206 16 # %% Set Project ID
207 17 PROJECT_ID = "MR05_"
208 18
209 19 # %% Create sample data
210 20 x = np.linspace(0, 10, 100)
211 21 y = np.random.rand(100) + 2
212 22 y_2 = np.sin(x) + 2
213 23
```



```
214 24 # %% Create sample figures
215 25
216 26 # 1. figure
217 27 FIG1 = plt.figure()
218 28 plt.plot(x, y, color="black")
219 29 plt.plot(x, y_2, color="yellow")
220 30
221 31 # 2. figure
222 32 FIG2 = plt.figure()
223 33 plt.plot(x, y, color="blue")
224 34 plt.plot(x, y_2, color="red")

225 38 # If multiple figures should be tagged, figures must be provided as
226     list.
227 39 FIGS_AS_LIST = [FIG1, FIG2]
```

228 In this part, the plotID modules and those necessary to create figures and images are imported.  
229 The variable *PROJECT\_ID* is set to provide the static part of the ID. Random data is used to  
230 create two figures with Matplotlib.

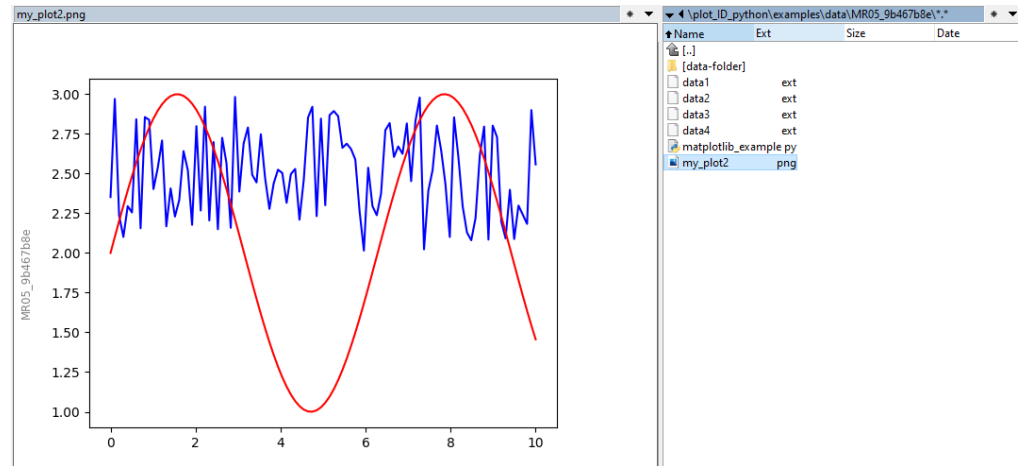
```
231 42 FIGS_AND_IDS = tagplot(
232 43     FIGS_AS_LIST,
233 44     "matplotlib",
234 45     location="west",
235 46     id_method="random",
236 47     prefix=PROJECT_ID,
237 48     qrcode=True,
238 49 )
```

239 Both Matplotlib objects are tagged with a generated ID. Using default options this call fits into a  
240 single line.

```
241 54 publish(FIGS_AND_IDS, ["../README.md", "../docs", "../LICENSE"], "
242     data")
```

243 Files (README.md and LICENSE) and a folder from the code repository are used in place of  
244 research data files. The string "data" is the relative path to the destination folder. This also shows  
245 that the workflow does not depend on any kind of file format or pre-organized structures. Any  
246 kind of data can be used. If the library used for creating the visualization is not (yet) supported,  
247 the resulting image file can still be tagged.

248 Figure 4 shows the resulting export folder with (renamed) data files, the script file "matplotlib\_ex-  
249 ample.py" and the tagged plot.



**Figure 4:** Example export folder and tagged plot  
'plotID-example-export' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

## 250 6 Distribution

251 Providing easy ways to acquire and use the software is important for adoption. The code is  
 252 Open Source under the Apache-v2.0 license. plotID requires a Python version  $\geq 3.10$  and is  
 253 OS-independent. The current release version is v0.3.1. Following Semantic Versioning [22], this  
 254 indicates that the public API is not considered stable yet.

255 At this time, the following distribution methods are available and described in the repository's [23]  
 256 README file.

### 257 6.1 Source Code

258 The plain source code is publicly available on a GitLab repository located under [git.rwth-](https://git.rwth-aachen.de/plotID/plotID_python/)  
 259 [aachen.de/plotID/plotID\\_python/](https://git.rwth-aachen.de/plotID/plotID_python/) [23] and can be directly downloaded or cloned with git.

```
260 1 git clone https://git.rwth-aachen.de/plotid/plotid_python.git
261 2 cd plotid_python
262 3 pip install -r requirements.txt
263 4 pip install .
```

### 264 6.2 Python Package

265 plotID is listed in the official Python Package Index (PyPI) [20]. The installation is done with  
 266 the following command:

```
267 pip install plotid
```

268 Distributing plotID independently from an existing Python installation is one of the aims of  
 269 later versions. Possible ways to achieve this are providing compiled executable files or a central  
 270 web-hosted service.

## 271 7 Ensuring good software quality

272 To ensure continuous good software quality, we adhere to best practices and the style guide  
273 PEP-8 [24]. This includes comments, docstrings and code formatting. To ensure adherence to  
274 these guidelines, automated tests on the code are implemented.

### 275 7.1 Unit tests

276 Python offers various libraries for unit testing. plotID, uses the *unittest* module [25], which is  
277 part of the Python standard library. Tests for each function are defined in the *tests* folder, along  
278 with the *runner\_test.py* script which organizes the execution of the tests, by discovering the test  
279 files based on their location. The *coverage* module measures how much of the code is covered  
280 by the tests, and total coverage of less than 95% is considered a failure. The tests are executed  
281 by a GitLab CI/CD pipeline [26] with every commit and merge request. Additional Jobs in the  
282 pipeline execute Pylint [27] and Flake8 [28] to check against coding style, programming errors  
283 and cyclomatic complexity. Commits that fail the pipeline tests cannot be merged into the main  
284 branch and will not make it into a release version. In the future, additional tests e.g. against  
285 security risks introduced by dependencies and more detailed reports are planned.

### 286 7.2 Documentation

287 To ensure easy access and understanding of the code, Python docstrings [29] have been imple-  
288 mented in the source code from the beginning. The docstrings are compiled into HTML using  
289 the Sphinx [30] Python package and GitLab CI-CD [26] creating an automatically generated  
290 API reference. The documents are hosted using GitLab Pages [31]. This documentation [32]  
291 will be improved by adding the readme, example code, example use cases and an introductory  
292 text until version 1.0.

## 293 8 Conclusion

294 The idea of plotID is a simple one: creating snapshots of work. As with many research data  
295 management operations, the benefit created through additional effort presents itself only at a  
296 later point. Benefits might be harvested by the creators of visualizations themselves by making  
297 access to their previous work easier for their own reuse.

298 The code and open-source implementation are still work-in-progress, but the core functionality is  
299 present. There are many ideas to improve and add features reported already and progress in early  
300 development happens fast, so many changes should be expected. This paper should be taken  
301 as an introduction to the tool and its principles - not as up-to-date documentation. Bug reports,  
302 merge requests with code, ideas for features and all feedback are welcome and best voiced in the  
303 GitLab repository [23].

## 304 9 Acknowledgements

305 The authors would like to thank the Federal Government and the Heads of Government of the  
306 Länder, as well as the Joint Science Conference (GWK), for their funding and support within the

307 framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) -  
 308 project number 442146713.

## 309 10 Roles and contributions

310 **Martin Hock:** Conceptualization, Methodology, Coding, Tests, Writing – original draft

311 **Hannes Mayr:** Coding, Tests, Methodology

312 **Manuela Richter:** Conceptualization, Methodology, Coding

313 **Jan Lemmer:** Conceptualization, Methodology

314 **Peter F. Pelz:** Project administration, Supervision, Funding Acquisition

## 315 References

- 316 [1] “Matplotlib - visualizations with python.” (2022), [Online]. Available: <https://matplotlib.org/> (visited on 10/10/2022).  
 317
- 318 [2] O. Rodeh and A. Teperman, “Zfs - a scalable distributed file system using object disks,” in  
 319 *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies,*  
 320 *2003. (MSST 2003). Proceedings.*, 2003, pp. 207–218. DOI: [10.1109/MASS.2003.119](https://doi.org/10.1109/MASS.2003.1194858)  
 321 [4858](https://doi.org/10.1109/MASS.2003.1194858).
- 322 [3] “Rsync.” (2022), [Online]. Available: <https://rsync.samba.org/> (visited on  
 323 11/15/2022).
- 324 [4] “Git.” (2022), [Online]. Available: <https://git-scm.com/> (visited on 11/15/2022).
- 325 [5] “Apache subversion.” (2022), [Online]. Available: <https://subversion.apache.org/>  
 326 (visited on 11/15/2022).
- 327 [6] “Git large file storage | an open source git extension for versioning large files.” (2022),  
 328 [Online]. Available: <https://git-lfs.github.com/> (visited on 11/15/2022).
- 329 [7] “Jedbrown/git-fat: Simple way to handle fat files without committing them to git, supports  
 330 synchronization using rsync.” (2022), [Online]. Available: [https://github.com/jedb](https://github.com/jedbrown/git-fat)  
 331 [rown/git-fat](https://github.com/jedbrown/git-fat) (visited on 11/15/2022).
- 332 [8] “Git-annex.” (2022), [Online]. Available: <https://git-annex.branchable.com/>  
 333 (visited on 11/15/2022).
- 334 [9] Y. O. Halchenko, K. Meyer, B. Poldrack, *et al.*, “Datalad: Distributed system for joint  
 335 management of code, data, and their relationship,” *Journal of Open Source Software*,  
 336 vol. 6, no. 63, p. 3262, 2021. DOI: [10.21105/joss.03262](https://doi.org/10.21105/joss.03262). [Online]. Available: <https://doi.org/10.21105/joss.03262>.  
 337
- 338 [10] “Data version control - dvc.” (2022), [Online]. Available: <https://dvc.org/> (visited  
 339 on 11/15/2022).
- 340 [11] “Signac - simple data management - signac.” (2022), [Online]. Available: [https://sig](https://signac.io/)  
 341 [nac.io/](https://signac.io/) (visited on 11/15/2022).

- 342 [12] “Project jupyter | home.” (2022), [Online]. Available: <https://jupyter.org/> (visited  
343 on 11/15/2022).
- 344 [13] N. CARPi, A. Minges, and M. Piel, “Elabftw: An open source laboratory notebook for  
345 research labs,” *Journal of Open Source Software*, vol. 2, no. 12, p. 146, 2017. DOI: [10.2](https://doi.org/10.21105/joss.00146)  
346 [1105/joss.00146](https://doi.org/10.21105/joss.00146). [Online]. Available: <https://doi.org/10.21105/joss.00146>.
- 347 [14] “Rspace eln & inventory.” (2022), [Online]. Available: [https://www.researchspace](https://www.researchspace.com/)  
348 [.com/](https://www.researchspace.com/) (visited on 11/15/2022).
- 349 [15] M. Hedges, A. Hasan, and T. Blanke, “Management and preservation of research data with  
350 irods,” in *Proceedings of the ACM First Workshop on CyberInfrastructure: Information*  
351 *Management in EScience*, ser. CIMS ’07, Lisbon, Portugal: Association for Computing  
352 Machinery, 2007, pp. 17–22, ISBN: 9781595938312. DOI: [10.1145/1317353.1317358](https://doi.org/10.1145/1317353.1317358).  
353 [Online]. Available: <https://doi.org/10.1145/1317353.1317358>.
- 354 [16] “Visualizing plots | data version control - dvc.” (2022), [Online]. Available: [https://d](https://dvc.org/doc/user-guide/experiment-management/visualizing-plots)  
355 [vc.org/doc/user-guide/experiment-management/visualizing-plots](https://dvc.org/doc/user-guide/experiment-management/visualizing-plots) (visited  
356 on 11/15/2022).
- 357 [17] “Tiobe index - tiobe.” (2023), [Online]. Available: [https://www.tiobe.com/tiobe-i](https://www.tiobe.com/tiobe-index/)  
358 [ndex/](https://www.tiobe.com/tiobe-index/) (visited on 02/24/2023).
- 359 [18] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, “The fair guiding principles for  
360 scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, p. 160 018,  
361 2016, ISSN: 2052-4463. DOI: [10.1038/sdata.2016.18](https://doi.org/10.1038/sdata.2016.18).
- 362 [19] N. P. Chue Hong, D. S. Katz, M. Barker, *et al.*, *FAIR Principles for Research Software*  
363 *(FAIR4RS Principles)*, version 1.0, May 2022. DOI: [10.15497/RDA00068](https://doi.org/10.15497/RDA00068). [Online].  
364 Available: <https://doi.org/10.15497/RDA00068>.
- 365 [20] PyPI. “Pypi · the python package index.” (2022), [Online]. Available: [https://pypi.o](https://pypi.org/)  
366 [rg/](https://pypi.org/) (visited on 08/19/2022).
- 367 [21] T. pip developers. “Pip · pypi.” (2023), [Online]. Available: [https://pypi.org/proje](https://pypi.org/project/pip/)  
368 [ct/pip/](https://pypi.org/project/pip/) (visited on 03/08/2023).
- 369 [22] T. Preston-Werner. “Semantic versioning 2.0.0.” (2023), [Online]. Available: [https://s](https://semver.org/)  
370 [emver.org/](https://semver.org/) (visited on 03/08/2023).
- 371 [23] GitLab RWTH Aachen. “Plotid / plotid\_python · gitlab.” (2022), [Online]. Available:  
372 [https://git.rwth-aachen.de/plotid/plotid\\_python](https://git.rwth-aachen.de/plotid/plotid_python) (visited on 08/19/2022).
- 373 [24] “Pep 8 – style guide for python code | peps.python.org.” (2022), [Online]. Available:  
374 <https://peps.python.org/pep-0008/> (visited on 10/11/2022).
- 375 [25] “Unittest — unit testing framework — python 3.10.6 documentation.” (2022), [Online].  
376 Available: <https://docs.python.org/3/library/unittest.html> (visited on  
377 08/29/2022).
- 378 [26] “Gitlab ci/cd | gitlab.” (2022), [Online]. Available: <https://docs.gitlab.com/ee/ci/>  
379 (visited on 08/19/2022).
- 380 [27] GitHub. “Pycqa/pylint: It’s not just a linter that annoys you!” (2022), [Online]. Available:  
381 <https://github.com/PyCQA/pylint> (visited on 08/29/2022).

- 382 [28] GitHub. “Flake8/index.rst at main · pycqa/flake8.” (2022), [Online]. Available: <https://github.com/PyCQA/flake8> (visited on 08/29/2022).
- 383
- 384 [29] “Pep 257 – docstring conventions | peps.python.org.” (2022), [Online]. Available: <https://peps.python.org/pep-0257/> (visited on 08/29/2022).
- 385
- 386 [30] “Welcome — sphinx documentation.” (2022), [Online]. Available: <https://www.sphinx-doc.org/en/master/> (visited on 08/29/2022).
- 387
- 388 [31] “Gitlab pages | gitlab.” (2022), [Online]. Available: <https://docs.gitlab.com/ee/user/project/pages/> (visited on 08/29/2022).
- 389
- 390 [32] “Welcome to plotid’s documentation! — plotid 0.2.3 documentation.” (2022), [Online].
- 391 Available: [https://plotid.pages.rwth-aachen.de/plotid\\_python/](https://plotid.pages.rwth-aachen.de/plotid_python/) (visited on
- 392 12/21/2022).