

# Wrattler

A framework for AI-assisted data wrangling

Final Report (Turing/GD006/1.2)

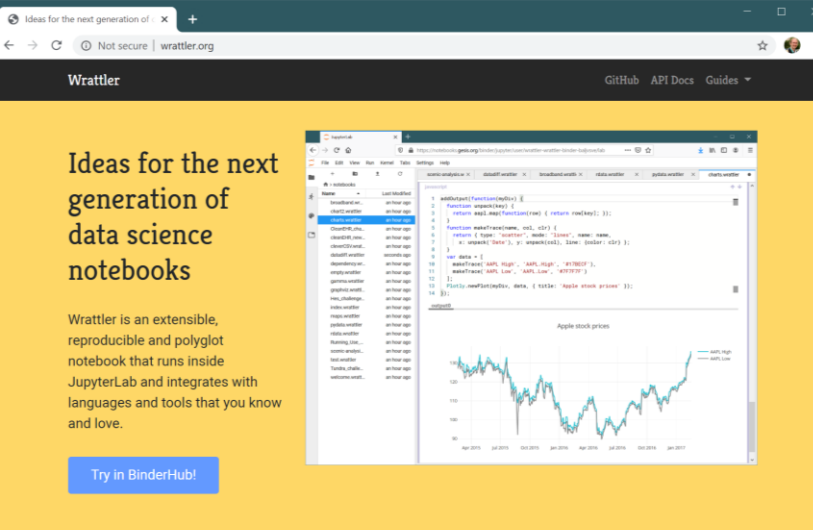
Tomas Petricek (PI), May Yong, Nick Barlow,  
Roly Perera, Anna Hadjitofi, Radka Jersakova

# Wrattler: An AI-assisted framework for data wrangling

This report provides a brief overview of the work done on the Wrattler data wrangling platform as part of the project Turing/GD006/1.2 between April 2019 and March 2020. The work on Wrattler has been divided into the following two strands:

1. Working with the Turing community on implementing concrete case studies in Wrattler and using this to guide the development
2. Implement innovative notebook features that leverage the Wrattler architecture and can significantly accelerate the data science workflow

We start with an overview of the work done before looking at selected individual aspects in more detail.



The screenshot shows the Wrattler website with a yellow background. The main heading is "Ideas for the next generation of data science notebooks". Below it, a text block describes Wrattler as an extensible, reproducible, and polyglot notebook that runs inside JupyterLab. A blue button labeled "Try in BinderHub!" is present. To the right, a notebook interface is shown with a code editor containing JavaScript code and a line chart titled "Apple stock prices" showing data from 2013 to 2017.

**</ Polyglot**  
Write notebooks that combine Python, R, JavaScript and more and easily pass data frames between cells in different languages.

**🔗 Smart**  
Simplify data wrangling tasks with AI assistants, clever but transparent tools that employ advanced machine learning techniques.

**🔄 Reproducible**  
Never get results based on out-of-date code. Wrattler tracks dependencies and automatically invalidates affected cells.

**🛠 Flexible**  
Write JavaScript extensions that create custom user interfaces and easily support languages ranging from R and Python to Racket.

**🖱 Interactive**  
Use interactive components that run fully in browser and write your own powerful JavaScript extensions that let users explore data.

**🔌 Extensible**  
Extend Wrattler with simple server-side kernel or a custom JavaScript cell type that has full access to notebook state.

# 1. Turing case studies and core system development (1/2)

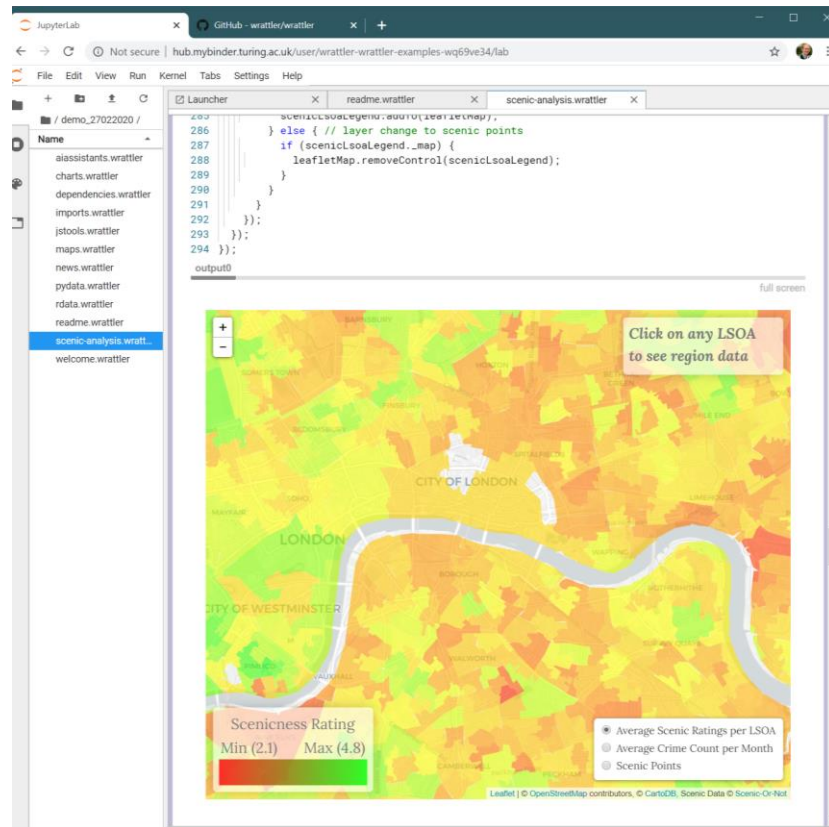
## a. Identify and work on three case studies

We collaborated with the Turing community and Turing data scientists used Wrattler for three projects:

- I. Testing the Broken Window hypothesis (see fig.)
- II. NLP analysis of Turing publications
- III. Detecting patterns in neonatal intensive care

## b. Improve the ability to handle large data frames

Wrattler data store now uses Apache Arrow to store and transmit data in an efficient binary format. This makes it possible to work with larger files (100MB – 1GB). Due to the lack of concrete use cases, we have not investigated support for Spark, which would be the next logical step towards “Big Data” support in Wrattler.



# 1. Turing case studies and core system development (2/2)

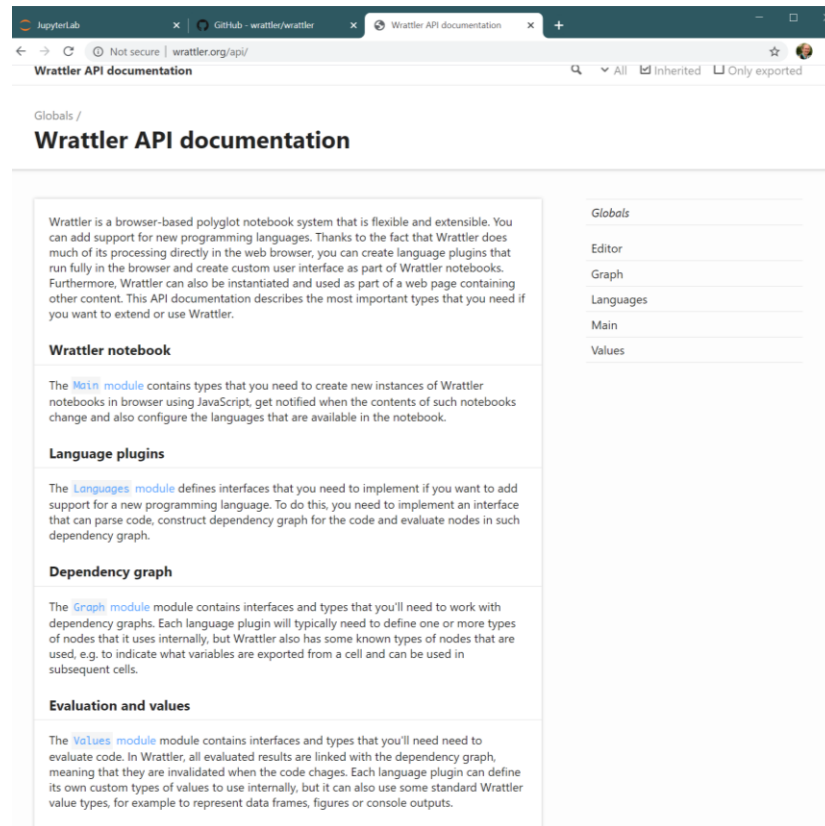
**c. Integrate Wrattler into Jupyter Lab and/or Binder**  
Wrattler adds a new type of notebook to Jupyter Lab (figure on the previous slide). It can be hosted in the cloud using Binder, which lets anyone try it using the “Try in Binder” button on the homepage.

## **d. Document the public interface of Wrattler**

The Wrattler web page now hosts a comprehensive documentation (see figure). We also published a guide on “Creating language plugins” that has been used by a developer joining the team later.

## **e. Specification of imports, efficiency of previews**

Imports are now specified via the standard Binder mechanism and previews no longer download the whole data set into the browser (only first N rows).



## 2. Implementing innovative notebook features (1/2)

We explored the feasibility of the following features and implemented four (two more than expected):

### a. Integration of AI assistants developed by AIDA

We implemented support for interactive AI assistants as a new cell type with Wrattler. We integrated three AIDA assistants (datadiff, ptype and CleverCSV) as well as one new (Outlier detection). The ColNet assistants remains a prototype.

### b. An API for the development of new AI assistants

The API is documented in the project repository in a readme file. A comprehensive report has been submitted as a paper to the KDD 2020 conference.

## AI assistants: A framework for semi-automated, accountable, and tooling-rich data wrangling

Anonymous Author(s)

### ABSTRACT

Data wrangling tasks such as obtaining and linking data from various sources, transforming data formats, and correcting erroneous records, constitute 80% of typical data engineering work. Despite the rise of machine learning and artificial intelligence, data wrangling remains a tedious and manual task for data scientists and engineers.

We introduce *AI assistants*, a class of semi-automatic interactive tools to streamline data wrangling. An AI assistant guides the analyst through a specific data wrangling task by recommending a suitable data transformation that reflects the constraints obtained through interaction with the analyst.

We formally define the structure of AI assistants and describe two ways of creating them, one based on optimization and one based on Bayesian inference. We use this framework to implement AI assistants for five common data wrangling tasks. Leveraging the common structure, we make AI assistants easily accessible to analysts in a notebook environment for data science. We evaluate our AI assistants both empirically and through several case studies, and show that the interactivity makes it easy to perform a task that would be difficult to do manually or with a fully automatic tool.

### ACM Reference Format:

Anonymous Author(s). 2018. AI assistants: A framework for semi-automated, accountable, and tooling-rich data wrangling. In *KDD '20: The 20th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, August 22–27, 2020, San Diego, CA, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3222443.3222456>

### 1 INTRODUCTION

Surveys among data scientists [8, 21] indicate that up to 80% of data engineering is spent on tedious, manual, and time-consuming data wrangling tasks such as obtaining data, making sense of encodings, merging datasets, and correcting erroneous records.

Data wrangling is hard to automate. First, datasets used in machine learning applications are often large, complex, and can hide special cases that require human insight. Second, data is heterogeneous and automatic methods developed for one dataset may fail when more data is added. Third, tuning of parameters needs to be interleaved with manual inspection of the results to evaluate whether a wrangling task was successful. Moreover, an automatic tool can easily confuse interesting outliers for uninteresting noise

in cases where a human would immediately spot the difference. This makes incorporating human insight into the data wrangling process crucial.

A major advance in the practice of data wrangling is this best achieved by developing *semi-automatic* methods that allow us to effectively combine the automation and scalability of machine learning methods with human insight. In this paper, we present semi-automated tools that guide a data analyst through a data wrangling task. To effectively streamline data wrangling while giving the analyst the necessary control, we require these tools to be:

*Interactive.* The tool should repeatedly ask the analyst for input and use machine learning techniques to find the best solution, respecting the analyst's insights, until the analyst accepts the solution.

*Unified.* A range of tools will need to be built for the many specific data wrangling tasks. To facilitate their integration into data science workflow, the tools need to share a common structure.

*Accountable.* Data wrangling tools should not opaquely transform input data into output data, but rather, produce scripts that can be reviewed, modified, and executed by a human.

*Tooling-rich.* Data wrangling tools need to integrate with modern data science programs, such as Jupyter, and use standard mechanisms such as code completion and live previews.

We introduce *AI assistants*, a class of tools have the above properties. We define the class formally, develop five concrete examples and evaluate them. Specifically, our contributions are as follows:

- We give a definition of an *AI assistant*, which formally captures the interaction between data analyst and a semi-automatic data wrangling tool, and we present two common patterns for creating AI assistants (Section 3).
- We adapt four data wrangling tools (for data parsing, data integration, type inference, and annotation with knowledge base concepts) to make them interactive and structure them as AI assistants, and we present a minimal AI assistant for column standardization in the presence of outliers.
- We extend the industry-standard JupyterLab system [25] with support for AI assistants, making it easy to use any AI assistant as part of a typical data wrangling workflow (Section 2).
- We evaluate our approach both quantitatively and qualitatively. The quantitative evaluation measures how many interactions are needed to achieve a data wrangling task, whereas the qualitative evaluation presents several case studies that illustrate how AI assistants allow data analysts to resolve cases where automatic tools would fail (Section 5).

While we may not eliminate the 80% of time data scientists spend on data wrangling entirely, our framework provides a direction that will allow data analysts to finally leverage the advances in AI for the most tedious and time-consuming aspect of their job.

## 2. Implementing innovative notebook features (2/2)

### c. Create refactoring tools

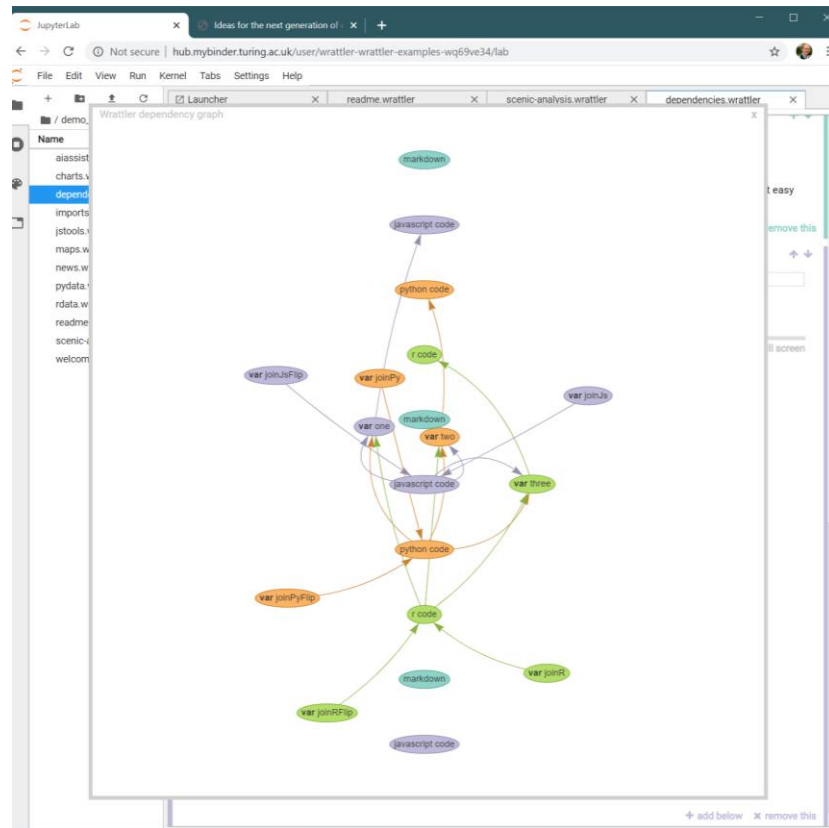
We completed preliminary work (making it possible to import functions from .py, .R and .js files), but did not focus on this area further.

### d. Support interesting programming languages

We added support for a research language for building transparent data visualization that has been developed at the Turing – called project Fluid.

### e. Better exposure of the dependency graph

The dependency graph is now exposed to the user who can access it programmatically. We used this feature to build a simple visualization tool (see figure), although deeper integration is an interesting future work.



---

## Detailed overview of selected work done

In the rest of the report, we provide more details on some of the interesting aspects of the work.

### We now look at the following components

Web page and documentation (1d)

Case studies using Wrattler (1a)

Integration with Binder and JupyterLab (1c)

Support for AI assistants (2a)

Data visualization language (2d)



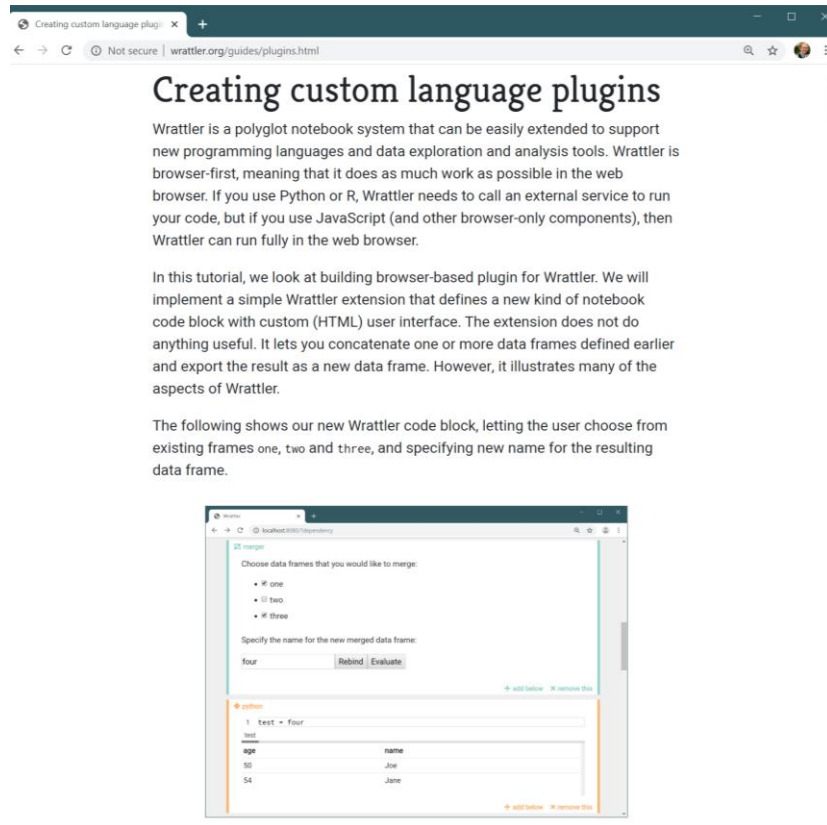
# Launching Wrattler web page with documentation (1d)

Wrattler is open-source and is available under the permissive MIT license. To enable others to contribute, we launched three new resources:

- <http://wrattler.org> is the new Wrattler homepage providing high-level project overview.
- <http://wrattler.org/guides/plugins.html> documents how to build a custom language plugin
- <http://wrattler.org/api/> provides a detailed API documentation for Wrattler components

More information about developing and contributing to Wrattler can be found in our open-source repositories:

- <https://github.com/wrattler/wrattler>
- <https://github.com/wrattler/wrattler-examples>





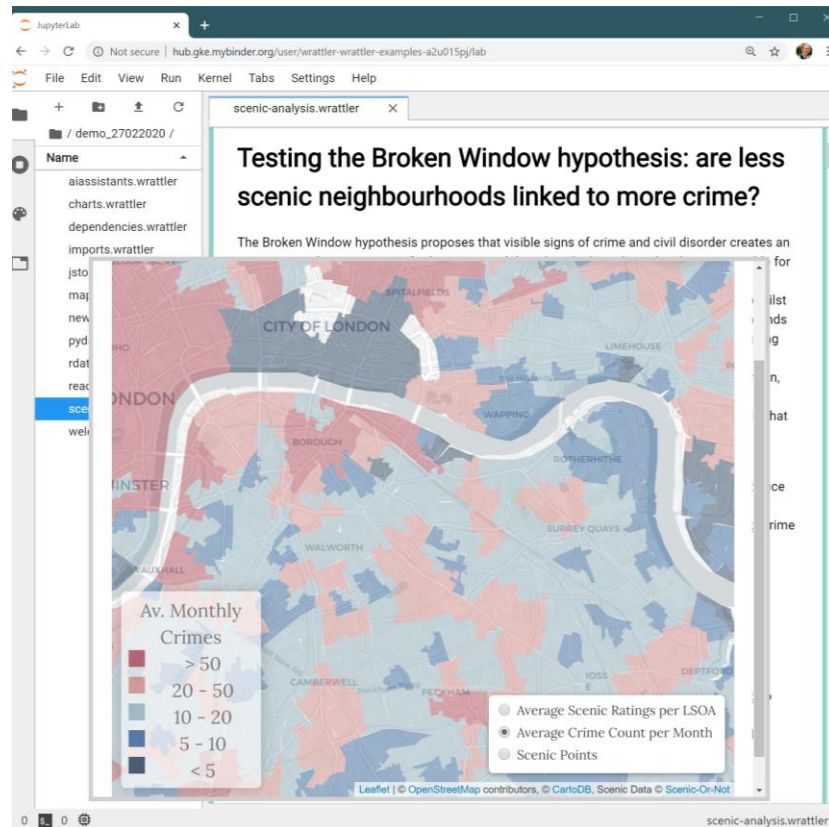
# Case study I: Testing the Broken Window Hypothesis (1a)

The Broken Window hypothesis proposes that visible signs of crime and civil disorder create an environment that encourages further crime and disorder.

There are several technically interesting aspects:

- The analysis combines multiple languages: Python for core analysis, R for statistical functions and JavaScript for data visualization.
- The JavaScript code uses a third-party mapping library (Leaflet) to visualize geographical data.

This work has been done by Turing data scientist **Anna Hadjitofi** together with **Chanuki Seresinhe**, who is a visiting researcher at the Turing. The code is available from: <https://github.com/wrattler/wrattler-examples>

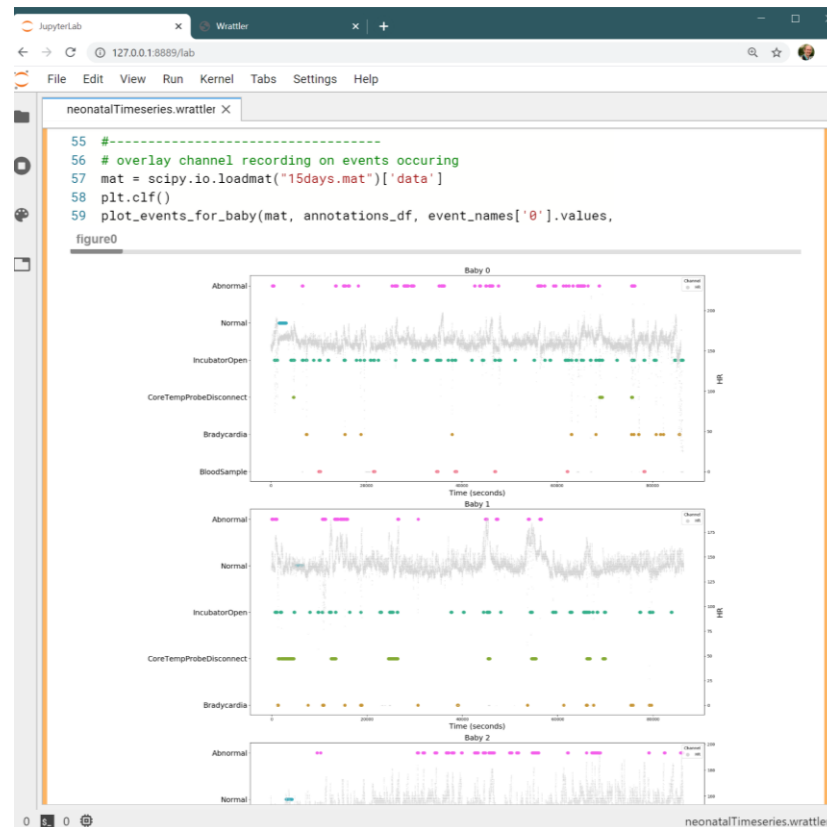


# Case study II: Detecting patterns in neonatal intensive care (1a)

In this analysis, we investigate whether we can detect patterns in the monitoring traces of premature babies in intensive care. The data used consists of several physiological measurements on fifteen patients in neonatal intensive care over 24 hours. There are several technically interesting aspects:

- The analysis uses Python for data analysis and popular JavaScript D3 package for visualization.
- The notebook serves tests the viability of the Wrattler system on a large and complex analysis.

This work has been done by Turing Research Engineering members **Anna Hadjitofi** with **May Yong** and our project partners. The code is available from: <https://github.com/wrattler/wrattler-examples>

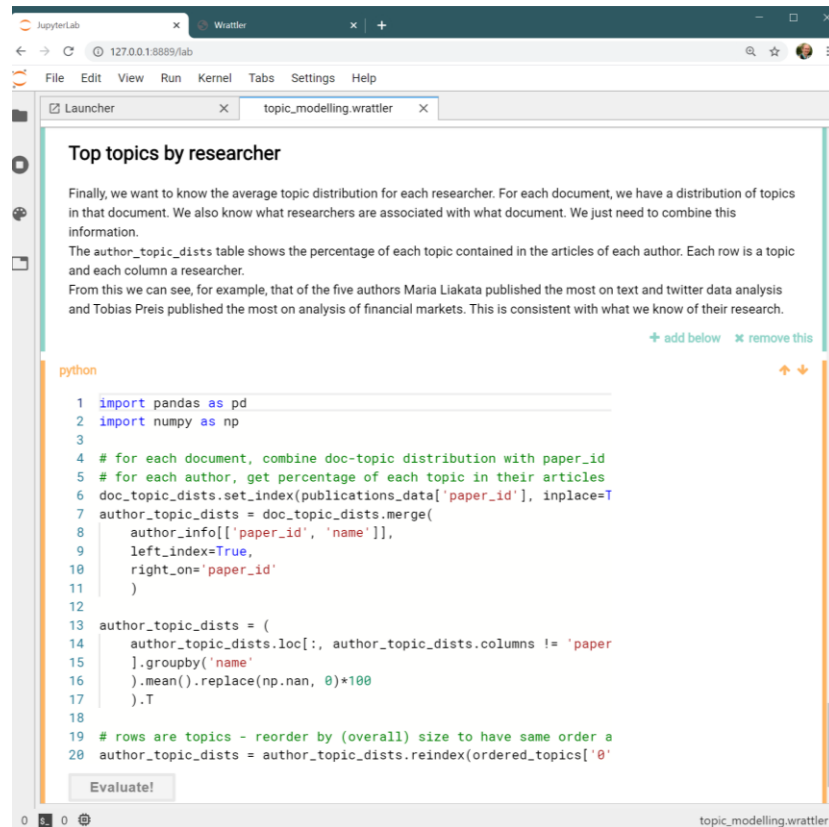


# Case study III: Topic modelling of Turing publications (1a)

We adapted parts of an analysis that illustrates the breadth of topics covered by publications by Turing researchers to run in Wrattler.

- The analysis uses interactive data visualization libraries in Python (pyLDAvis). To support those, we extended Wrattler with a mechanism for displaying interactive outputs produced by Python and R.
- The analysis motivated two potential future projects for Wrattler: support for passing files between blocks and support for interactive data cleaning. Those were not pursued as part of this project.

This work has been done by Turing data scientist **Radka Jersakova**. The code is available from: <https://github.com/wrattler/wrattler-examples>



The screenshot shows the Wrattler JupyterLab interface. At the top, there's a browser window with the URL 127.0.0.1:8889/lab. Below the browser, there's a menu bar with File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The main area has a tab labeled 'topic\_modelling.wrattler'. The content of the tab is titled 'Top topics by researcher' and includes a text block explaining the goal of the analysis: to find the average topic distribution for each researcher. Below the text, there's a code block with Python code that uses pandas and numpy to process topic distributions. The code includes comments and uses methods like set\_index, merge, groupby, and mean to calculate the average topic distribution for each researcher. The code is as follows:

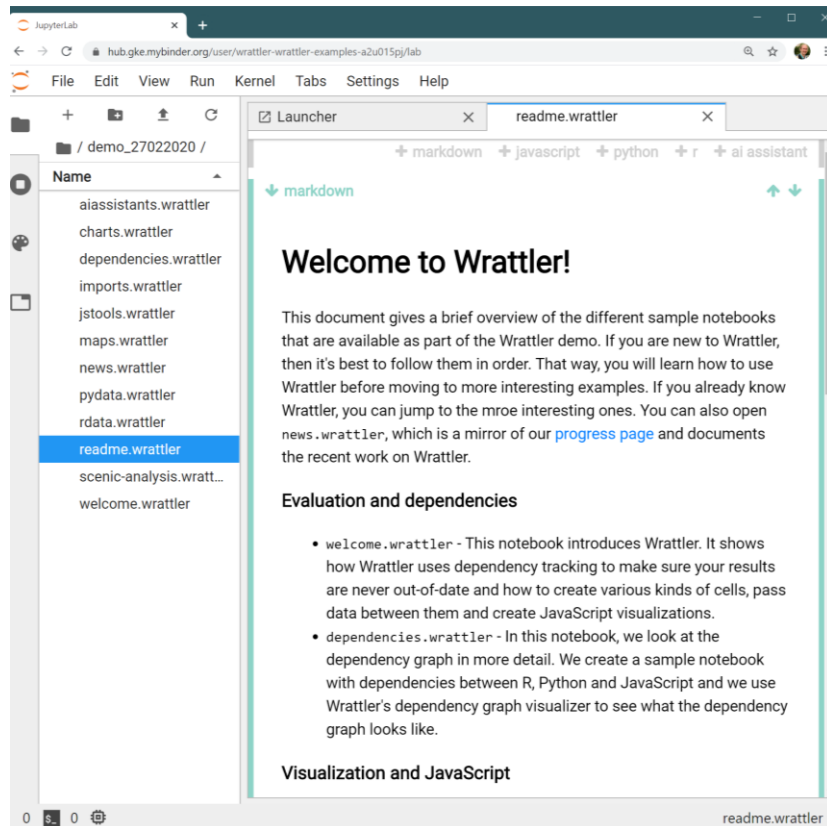
```
python
1 import pandas as pd
2 import numpy as np
3
4 # for each document, combine doc-topic distribution with paper_id
5 # for each author, get percentage of each topic in their articles
6 doc_topic_dists.set_index(publications_data['paper_id'], inplace=True)
7 author_topic_dists = doc_topic_dists.merge(
8     author_info[['paper_id', 'name']],
9     left_index=True,
10    right_on='paper_id'
11 )
12
13 author_topic_dists = (
14     author_topic_dists.loc[:, author_topic_dists.columns != 'paper_id']
15     .groupby('name')
16     .mean().replace(np.nan, 0)*100
17 )
18
19 # rows are topics - reorder by (overall) size to have same order as
20 author_topic_dists = author_topic_dists.reindex(ordered_topics['0'
```

At the bottom of the code block, there's an 'Evaluate!' button. The bottom status bar shows '0' and 'topic\_modelling.wrattler'.

# Integration with Binder and Jupyter Lab (1c)

Wrattler is now fully integrated with Jupyter Lab and Binder, which makes it a part of a growing data science ecosystem with long-term sustainability.

- **Binder** provides an easy way to launch Wrattler in the cloud using a Kubernetes cluster. Binder Hub can be hosted on a variety of cloud providers including Google Cloud, AWS and Azure.
- **JupyterLab** makes it possible to use Wrattler as part of larger data science projects, alongside existing tools such as IPython notebooks.
- Using the two tools also made it possible to create “Try in BinderHub” button for the homepage which starts Wrattler directly in the web browser.



# Interactive AI assistants (2a)

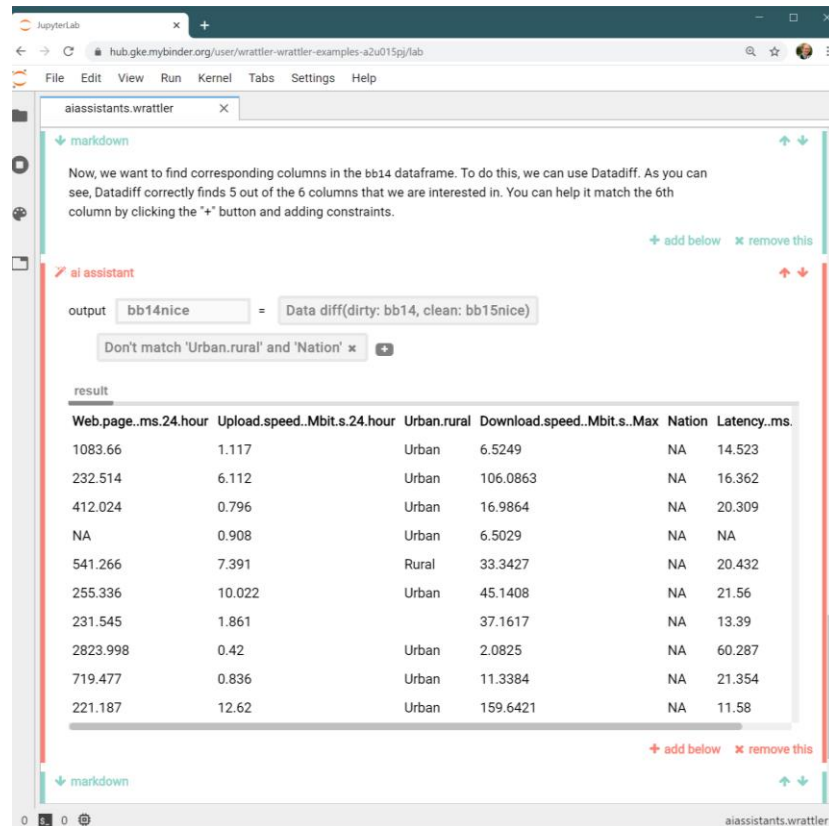
AI assistants are semi-automatic tools that assist the analyst when solving data wrangling challenges. We developed interactive versions of three AIDA assistants.

AI assistants are available as a new type of block. They are used through an interactive user interface: (i) the assistant attempts to find a solution, (ii) user clicks the “+” button to specify constraints and (iii) the assistant refines the previous solution (see figure).

We developed a new assistant for outlier detection and adapted three AIDA assistants (DataDiff, ptype and CleverCSV) into the new interactive framework. How to create a new assistant is documented online:

<https://github.com/wrattler/wrattler/tree/master/aiassistants>

The work has been done by Turing researchers  
**Gerrit van den Burg, Alfredo Nazabal and Taha Ceritli**



Now, we want to find corresponding columns in the bb14 dataframe. To do this, we can use Datadiff. As you can see, Datadiff correctly finds 5 out of the 6 columns that we are interested in. You can help it match the 6th column by clicking the “+” button and adding constraints.

add below remove this

ai assistant

output `bb14nice` = `Data diff(dirty: bb14, clean: bb15nice)`

Don't match 'Urban.rural' and 'Nation' ✕

result

Web.page.ms.24.hour	Upload.speed.Mbit.s.24.hour	Urban.rural	Download.speed.Mbit.s.Max	Nation	Latency.ms.
1083.66	1.117	Urban	6.5249	NA	14.523
232.514	6.112	Urban	106.0863	NA	16.362
412.024	0.796	Urban	16.9864	NA	20.309
NA	0.908	Urban	6.5029	NA	NA
541.266	7.391	Rural	33.3427	NA	20.432
255.336	10.022	Urban	45.1408	NA	21.56
231.545	1.861		37.1617	NA	13.39
2823.998	0.42	Urban	2.0825	NA	60.287
719.477	0.836	Urban	11.3384	NA	21.354
221.187	12.62	Urban	159.6421	NA	11.58

add below remove this

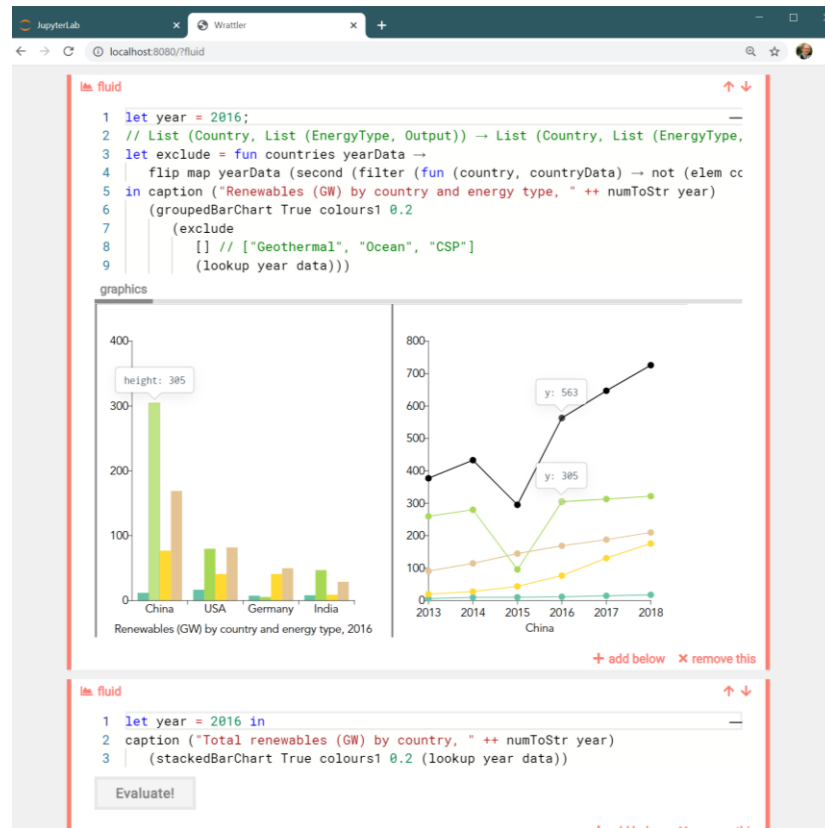
# Fluid – Integration for a data visualization language (2d)

We added support for Fluid, a research programming language for building transparent data visualizations.

For example, linked charts (see figure) make it easier to see how input data contribute to different aspects of a chart by highlighting how a data point contributes to two different charts. Building linked charts is difficult and typically requires manual coding of a specific chart.

In Fluid, linking can be used with any chart created by the analyst. Fluid uses provenance tracking to understand what inputs and what code contribute to a particular visual feature of a chart and visualizes those connections.

Fluid has been developed by **Roly Perera** and is available at: <https://github.com/rolyp/fluid>



$$= \frac{1}{C_0} h_0(x)$$

$$\int \frac{h_0(x)}{h_\psi(x)} p_\psi(x) dx$$

$$-\frac{1}{h} \sum \frac{h_0(x_i)}{h_\psi(x_i)}$$

turing.ac.uk  
@turinginst

