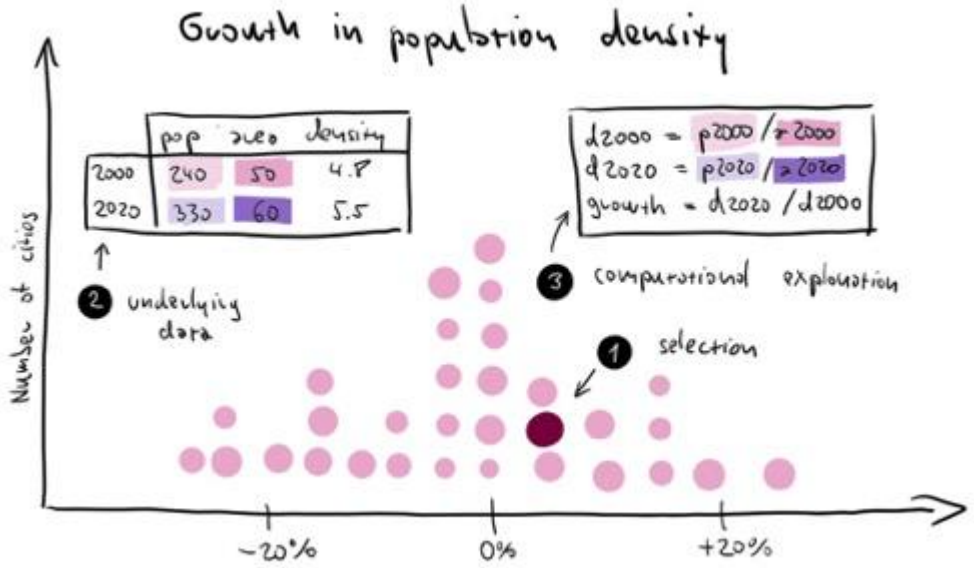
Fluid: A Programming Language for Open, Explorable Research Outputs

Conducting science in the open and making source code and data freely available is an essential part of modern scientific practice, but does not address the challenge of how to **explore** and **understand** the complex relationships between data and computational outcomes. **Fluid** [1, 2] is a new kind of programming language that makes it easy to create data-driven software artefacts with **built-in transparency features**. For example (Figure 1) a data visualisation might automatically highlight relevant parts of an underlying dataset as visual elements are selected, and (optionally) produce computational "explanations" of how those data elements were aggregated or otherwise operated on during the computation of the selected visual elements. This is achieved using dependency-tracking techniques based on **Galois connections**, a mathematical abstraction able to represent fine-grained relationships between inputs and outputs.



Fluid is being developed at the **Institute of Computing for Climate Science**, Cambridge, with collaborators at University of Bristol. Over the next year or so we plan to evolve Fluid into a mature platform for communicating science in a way that allows an interested reader to interact with outputs such as charts to better understand what they **represent**: how specific visual elements relate to the data they were computed from.

The following two extensions to Fluid would make for a suitable Part II project. Your project could involve formal/theoretical elements, interpreter implementation, UI work, data visualisation/analysis, and case studies, in any combination to suit your interests and background.

Project 1: Macros and quasiquotation for authoring interactive papers

Macros are special user-defined functions that produce code rather than data, and are are useful for creating embedded **domain-specific languages** which allow solutions to be

(efficiently) expressed in a language close to a chosen problem domain. In Fluid, we will use macros to provide convenient surface notations for use by data scientists and data journalists, which can then be translated into the core language for efficient execution. Your project will involve formally defining a macro mechanism, extending Fluid's dependency-tracking infrastructure to the **macro expansion** phase (when any macros in the user's program are expanded), and developing interesting case studies that show the feature in action. You could extend this to a Scheme-style **quasiquotation** [3] mechanism that would allow literals (such as Markdown documents) to contain embedded Fluid code which, when evaluated, would splice charts and tables into the containing document.

Project 2: Notebook-style reversible debugger with dependency tracking

For dependency-tracking, Fluid must retain all the information about program execution that is normally thrown away. This presents an interesting opportunity: a debugger which allows the user to work backwards as well as forwards, and interact with intermediate values to see how individual data elements relate to other values downsteam and upstream in the computation. Taking inspiration from data science notebooks like **Jupyter**, you could design and implement a "data pipeline explorer" which expands a complex computational pipeline into a notebook-like interface on demand, presenting the user with nested "code cells" which show the expressions used to compute intermediate values. The **Nile viewer** [4] is a hand-crafted mockup which nicely illustrates what this might look like. Your project would be to leverage Fluid's persistent execution and dependency-tracking runtime to design and implement such a system for real.

References

[1] https://f.luid.org/

[2] Roly Perera, Minh Nguyen, Tomas Petricek, Meng Wang. Linked Visualisations Via Galois Dependencies. PACMPL (POPL) 2022.

[3] https://en.wikipedia.org/wiki/Quasi-quotation

[4] https://tinlizzie.org/dbjr/high_contrast.html