

The Gamma: Data Exploration through Iterative Prompting

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

ABSTRACT

Governments, non-profit organizations and citizen initiatives publish increasing amounts of data, but extracting insights from such data and presenting them to the public is hard. First, data comes in a variety of formats that each requires a different tool. Second, many data exploration tools do not reveal how a result was obtained, making it difficult to reproduce the results and check how they were obtained. We contribute The Gamma, a novel data exploration environment for non-experts. The Gamma is based on a single interaction principle and using it results in transparent and reproducible scripts. This allows transfer of knowledge from one data source to another and learning from previously created data analyses. We evaluate the usability and learnability of The Gamma through a user study on non-technical employees of a research institute. We argue that our approach allows journalists and the public to benefit from the rise of open data, by making data exploration easier, more transparent and more reproducible.

Author Keywords

Data exploration; End-user programming; Data journalism; Programming languages; Type providers

INTRODUCTION

Data science has more capabilities to help us understand the world than ever before, yet at the same time post-truth politics and increasing public distrust in statistics makes data-driven insights increasingly less relevant in public discourse [5]. This should perhaps not be a surprise. Journalists can access increasing amounts of data, but producing engaging and transparent data-driven reports that are easy to interpret is expensive and requires expert programming skills [7].

The design of a data exploration tool for journalists poses a unique mix of challenges. First, the tool needs to be easy to learn for end-users working under tight deadlines. Second, it needs to support a wide range of data sources in a way where the expertise gained when working with one data source is relevant for other data sources. Third, the resulting data-driven insights need to be transparent, allowing the readers to verify the claims and learn how to reproduce the work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UIST'20, October 20–23, 2020, Minneapolis, MN, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-6708-0/20/04...\$15.00

DOI: <https://doi.org/10.1145/3313831.XXXXXX>

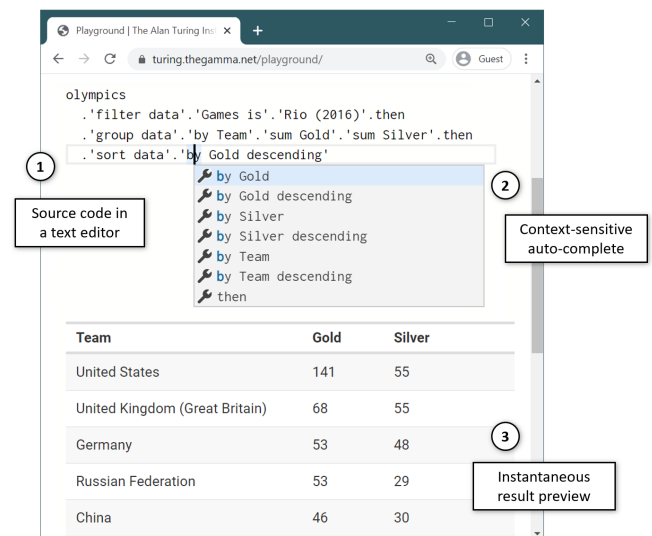


Figure 1. Teams with the greatest number of gold medals from Rio 2016 Olympics with a reproducible The Gamma script (1), an auto-complete prompt offering ways of sorting the data (2) and instant preview (3).

We present The Gamma, a text-based data exploration environment for non-experts. The Gamma is based on a single interaction principle, which provides uniform access to a range of data sources including data tables, graph databases and data cubes. An analysis created in The Gamma is a transparent script that can be followed to reproduce the result from scratch. This allows learning from existing analyses and encourages readers to engage with the results. Our key contributions are:

- We identify the design requirements for a data exploration tool for journalists (Section 3) and follow those to build a novel programming environment The Gamma (Section 4).
- We introduce *iterative prompting* (Section 5), an interaction design principle that can be used to complete a variety of programming tasks in a uniform way that allows transfer of knowledge between different tasks.
- We show how to use the iterative prompting principle for querying of distinct data sources including data tables, graph databases and data cubes (Section 6).
- We discuss a number of case studies (Section 7) and conduct a user study to evaluate the usability of The Gamma and the extent to which users can, (i) learn from examples and (ii) transfer knowledge between tasks (Section 8).

The Gamma is available as open-source at thegamma.net.

RELATED WORK

Visual tools.

Programming tools. Notebooks

Both follow from assistant tools for software development [8]. Our implementation, outlined in the previous section, implements iterative prompting through a code editor mechanism designed for code completion. Finally, the work on type providers [19, 16, 15], which we extend, also utilizes code completion.

Journalism. Idyll [4]

Type providers. PL work

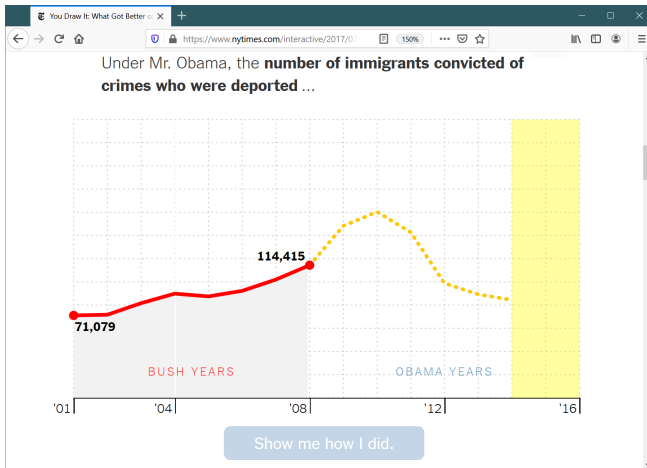


Figure 2. New York Times article on Obama’s legacy [9]. The article asks the reader to make a guess (engagement), but only lists “Immigration and Customs Enforcement” as a source of data.

MOTIVATION

The Gamma aims to adapt the recent innovations in programming language research, especially the work on type providers, into a form where it could be used in practice by journalists and other non-expert interested in data exploration. We start with a careful consideration of our target application domain, i.e. data analyses produced by journalists and citizen data scientists that are published online. We look at both practical requirements for such programming environment and requirements arising from our focus on journalism. This analysis is based on the author’s experience of collaborating with journalists¹, review of literature on data journalism, e.g. [7, 17, 1] and more general trends in journalism.

Open Journalism

Journalism continually develops and responds to the many challenges it faces [6]. Two recent challenges are relevant to our work. The first is building trust in media. One way of establishing trust in the age of fake news is to be more transparent about editorial decisions, process and original sources. Many journalists believe that opening up the process shows the quality and trustworthiness of their work [12]. The second challenge is reader engagement. To develop a relationship with readers, journalists are increasingly looking for meaningful ways of engagement. This includes reader comments, involvement of citizen journalists [11, 3] and the development of new interactive formats [9]. To address the above challenges, a tool for data exploration should satisfy the following three requirements.

Trust Through Transparency

To support trustworthiness, data analyses should be transparent. The reader should be able to determine what is the source of analysed data and how has the data been transformed. As much as possible, these capabilities should also be accessible to non-expert readers.

¹Citations removed to preserve anonymity.

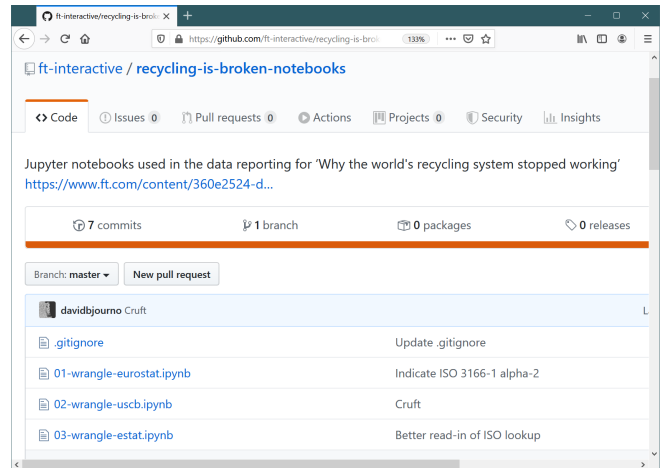


Figure 3. Financial Times analysis of recyclable waste. Full source is provided as Jupyter Notebooks on GitHub [2], but re-running the analysis is difficult, even for an expert.

Reproducibility for Fact Checking

It should be possible to re-run the analysis to verify that it produces the presented results. However, running an opaque script is not enough. A reader should be able to recreate the analysis by following the necessary steps from the original data source to the end result.

Encouraging Meaningful Engagement

The tool should support a mechanism through which readers can engage in a meaningful discussion. For example, it should allow modifying of parameters of a data visualization in order to show how different choices affect the final result.

End-user Data Exploration

Our aim is to make programmatic data exploration accessible to journalists, but we want to keep the desirable properties of text-based programming. In particular, source code of a data exploration should provide a full reproducible record of how the data analysis has been done. As end-users, journalists have a number of interesting characteristics. They work under tight deadline and data exploration is only a complementary skill. They also need to work with a wide range of data sources, including big data tables (e.g. Iraq War documents leak) or graph databases (e.g. Panama Papers). This leads to a number of practical requirements on the programming environment.

Conceptual Simplicity

We target end-users who cannot dedicate much time to learning about a tool prior to using it. Consequently, using the tool should require understanding of only a small number of concepts. Once the user understand a small number of concepts, they should be able to complete basic data exploration tasks.

Uniformity across Data Sources

The users should be able to navigate through large databases, query relational databases and query graph databases through the same mechanism. Ideally, expertise gained with one data source should also be transferable to working with another data source.

Learning without Experts

Sarkar [18] reports that users learn how to use Excel either by talking to experts, or by seeing a feature in a spreadsheet received from a colleague. In our circumstances, experts are unlikely to be available, so the tool should support learning from examples. When looking at a work done and published by another person, the user needs to see (and be able to understand) how a task was completed.

OVERVIEW

The Gamma is a text-based programming environment that allows non-experts create simple data exploration scripts using a single interaction principle – choosing an item from an auto-complete list. It supports a range of data sources including tabular data, graph data and data cubes.

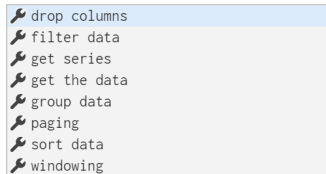
Querying Travel Expenses

To introduce The Gamma, we walk through a simple problem that a local journalist might want to solve. The UK government publishes travel expense claims by members of the House of Lords. A journalist wants to find out which of the members representing the Kent county spend the most on travel. The following shows a subset of the data:²

```
1 Name, County, Days Attended, Days Away, Travel Costs
2 Lord Adonis, London, 8, 0, 504
3 Baroness Afshar, Yorkshire, 2, 0, 0
4 Lord Alderdice, Oxfordshire, 3, 0, 114
5 Lord Alli, London, 5, 0, 0
6 Baroness Amos, London, 3, 0, 0
```

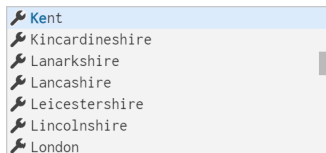
After the analyst imports the CSV file (through a web interface), the environment is initialized with code that refers to the imported variable `expenses`. The analyst then types `.'` (dot):

```
1 expenses.
```



The type provider for tabular data allows analysts to construct simple queries. It first offers a list of operations that the analyst might want to perform such as grouping, filtering and sorting. To find members of the House of Lords from Kent, the analyst chooses `filter data`, types `.'` and then chooses `County is` from the offered list, types `.'` and starts typing `Kent`:

```
1 expenses
2 . 'filter data'. 'County is'. Ke
```



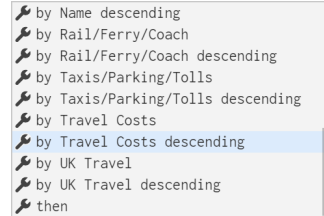
The completion list is generated from the values in the `County` column of the dataset. After selecting `Kent`, the live preview is updated to only show records according to the specified filter:

Name	County	Days Attended	Days Away	Travel Costs
Lord Astor of Hever	Kent	7	0	85
Lord Freud	Kent	1	0	0
Lord Harris of Peckham	Kent	3	0	0
Baroness Noakes	Kent	6	0	135

²<https://www.parliament.uk/mps-lords-and-offices/members-allowances/house-of-lords/holallowances/>

To finish specifying filtering conditions, the analyst chooses `then` and is offered the same list of querying operations as in the first step. To sort House of Lords members by their travel costs, she now chooses `sort data` and types `.'` (dot):

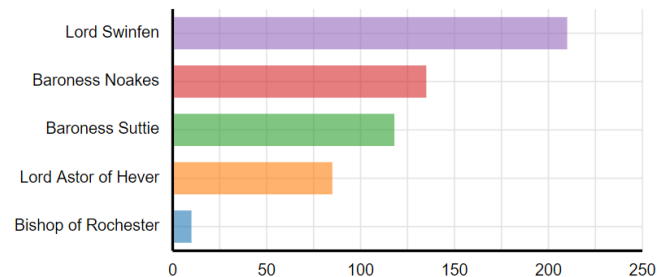
```
1 expenses
2 . 'filter data'. 'County is'. Kent. then
3 . 'sort data'.
```



The auto-complete offers a list of columns that can be used for sorting, each with ascending (default) and descending order option. After choosing one or more sort keys, the analyst selects the `then` member and is, again, offered the list of querying operations. They use `paging` to get top 5 records and `get series` to obtain a data series with just the House of Lords member name and their travel expenses.

```
1 expenses
2 . 'filter data'. 'County is'. Kent. then
3 . 'sort data'. 'by Travel Costs descending'. then
4 . paging. take(5)
5 . 'get series'. 'with key Name'. 'and value Travel Costs'
```

When the code evaluates to a data series with a categorical (textual) key and a numerical value, The Gamma switches from displaying the result as a table to a column chart:



Querying via Iterative Prompting

In the example, the journalist constructs a query that filters and sorts a data table. This is not unlike writing a query in the SQL language. However, the whole query is constructed through a single interaction mechanism that we refer to as *iterative prompting*. The user triggers the interaction by typing `.'` at the end of the query constructed so far. She is then offered a list of available operations in the current context. This may include a choice of top-level operations (e.g. at the start and then again after choosing `then`) or possible parameters for an operation (such as possible sorting keys and directions after choosing `sort data`). The user then merely needs to choose one of the available options, rather than learning the syntax and semantics of SQL in order to know what constructs can be used in the given context. We discuss the important aspects of The Gamma design in the next section and then discuss the integration of specific data sources in detail, revisiting the type provider for querying data tables.

DESIGN

We identified a number of requirements for a data exploration tool for journalists earlier. First, using the tool should result in transparent and reproducible data analyses that can be understood and modified by non-experts. Second, the tool should minimize the number of concepts that the user needs to understand and, subsequently, allow them to learn through exploration and from examples. In this section, we discuss how our design aims to satisfy those requirements.

Lowering the Barrier to Entry

Data exploration tasks, such as querying tables, have a certain irreducible complexity. Regardless of the interface, the user will be exposed to concepts such as filtering, sorting and grouping. Our design aims to lower the barrier to entry by stratifying the concepts into first-level *iterative prompting* principle and second-level *domain specific languages* for each kind of data source. The user needs to master the iterative prompting principle in order to start working with the system. However, using individual data sources can be mastered gradually.

Iterative Prompting

Iterative prompting is an interaction principle where the user repeatedly invokes an auto-complete prompt and makes a selection from the offered options. The principle is closely related to both code completion and the use of command line. Iterative prompting is novel as an overarching interaction principle for program construction.

In code completion, the auto-complete prompt is invoked only in certain contexts, e.g. when accessing a member of an object through a variable defined earlier in code. It requires the user to be sufficiently familiar with the programming language in order to get to a point where they are offered a list of members. In contrast, our system only requires choosing the initial data source. The rest of the programming is done via iterative prompting. The iterative nature of iterative prompting makes it similar to using the command line or REPL (read-eval-print-loop) tools. However, those repeatedly ask the user to type code or commands.

We argue that iterative prompting is easier than other forms of interaction, because it follows the *recognition over recall* usability heuristic. The users are only required to choose from an offered list of options, rather than having to recall a possible command (to type in a command line) or a syntax in a text-based programming environment.

Domain Specific Languages

The access to individual data sources in The Gamma is facilitated through a domain specific language, which defines the primitives that are offered to the user in the auto-complete prompts invoked through iterative prompting. The domain specific languages are embedded in The Gamma – they define merely the available members (and specify how to evaluate a chain of members), but they cannot define any custom syntax.

We discuss the individual domain specific languages for querying data tables, graph databases and data cubes in a later section. The complexity of those languages differs. The previously discussed language for querying tabular data is the most

complex one. However, The Gamma makes it easy for the user to start exploring and learning new languages, because they are all accessible via iterative prompting.

Building the Magic Escalator of Knowledge

As discussed earlier, The Gamma is designed for users who work under tight deadlines and only analyse data as their secondary task. The Gamma aims to support such users by having a low barrier for entry and making it easy to learn independently.

Design for Percolation

When analysing how Excel users learn Sarkar [18] points out that users learn new features opportunistically when the usage of a feature is apparent in a spreadsheet. For example, users can learn different functions to use in formulas, because those are visible in the cell. Learning how to use a wizard for creating charts in this way is not possible because it leaves no trace in the spreadsheet – only the final result. Sarkar's recommendation is to *design for percolation*, i.e. in a way where looking at the final result makes it apparent what feature has been used and how.

Text-based Source Code

In The Gamma, each step in the iterative prompting process results in an identifier that is added to the source code. This means that a program constructed solely through iterative prompting keeps a full trace of how it was created. Seeing the resulting source code provides the user all information that they need to recreate the program, not just by copying it, but also by using the iterative prompting mechanism. The Gamma represents code as text and allows the user to edit it freely, so not all interactions leave a trace. For example, deleting the most recently selected option and choosing a different one is not apparent from the result.

Making Complex Things Possible

Although the focus on The Gamma is simple data exploration and almost all the use cases we envision can be achieved using the iterative prompting interaction principle, there remain a few cases where more flexibility is needed. We aim to *make simple things easy and complex things possible*. The language thus supports a small number of additional constructs that can only be used through text editing. The following example illustrates three additional constructs:

```
1 let topTravel =
2   expenses
3   . 'sort data'. 'by Travel Costs descending'. then
4   . paging. take(5). 'get series'
5   . 'with key Name'. 'and value Travel Costs'
6
7   charts.column(topTravel)
8   . setTitle("House of Lords members by travel expenses")
9   . setColors(["red", "blue", "green"])
```

First, The Gamma allows operations with parameters such as `take(3)` or `setTitle("...")`. This is currently needed when writing a query that skips or takes first N elements from a table. The remaining features are not needed for basic data exploration, but allow other scenarios, such as manual creation of charts. The `let` construct can be used to define (immutable) variables and The Gamma also supports lists written as `[1, 2, 3]`.

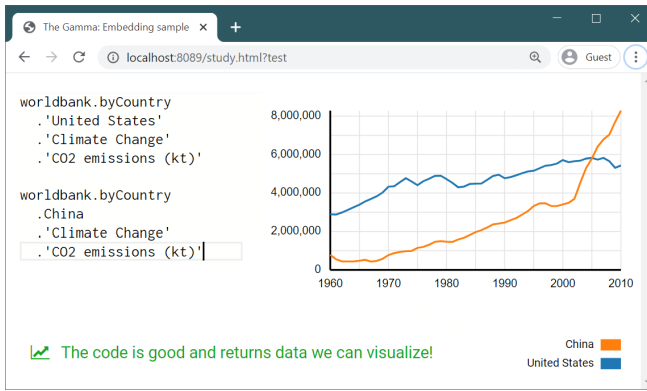


Figure 4. The Gamma environment with an automatically generated chart comparing CO2 emissions of China and United States.

IMPLEMENTATION

The Gamma consists of a programming language with a web-based coding environment and a number of type providers that provide access to various kinds of data sources. The system is implemented in F# and compiled to JavaScript using Fable.

The Gamma Language

In The Gamma, a program is a sequence of commands. A command can be either a variable declaration or an expression that evaluates to a value such as a data table or a chart. An expression is a reference to a data source followed by a chain of member accesses. A member can be either an ordinary member or an operation which takes a list of parameters enclosed in parentheses. When the member name contains non-alphanumeric characters such as whitespace, it needs to be written in quotes. Our implementation supports a few other features, but those are not used in this paper.

The Gamma uses a type system to infer what members are available at a given point in a chain. A type is an object with a list of members that, in turn, have their own types. The types are not built-in, but are generated by a type providers for individual data sources. The types are generated lazily and are only computed when a member appears in code.

The web-based coding environment uses the Monaco³ editor and implements auto-completion feature that is triggered when the user types `'` and relies on information from the type system. The can display the value of a current command inline as in Figure 1, or values of all commands on the side as in Figure 4. In the latter case, it can also combine one or more data series into a single chart.

Data Cube Type Provider

Our first type provider allows users explore data form a data cube. For example, the World Bank⁴ collects a range of indicators about many countries in the world each year. The data set is a three-dimensional cube with dimensions corresponding to countries, indicators and years. The following example, also shown in Figure 4, uses the provider to access CO2 emission data for United States:

³<https://microsoft.github.io/monaco-editor/>

⁴<https://data.worldbank.org/>

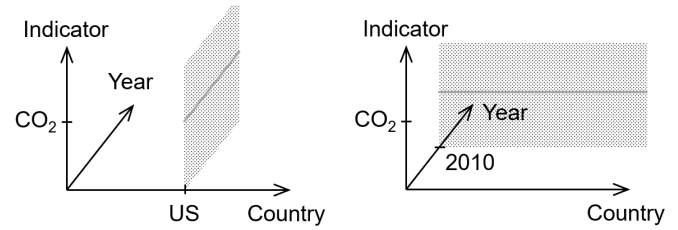


Figure 5. Exploring data cubes. The World Bank provider allows users to first choose a country or a year and then an indicator.

```
1 worldbank.byCountry.'United States'
2   .'Climate Change'. 'CO2 emissions (kt)'
```

As illustrated in Figure 5, the provider allows users to select a data series from the data cube. By choosing `byCountry.'United States'`, we restrict the cube to a two-dimensional plane. We then choose an indicator category `'Climate Change'` and a specific indicator `'CO2 emissions (kt)'`, obtaining a time series with years as keys and emission data as values. The World Bank provider also supports filtering by year and indicator.

Another type provider available in The Gamma that is based on the data cube structure allows the user to explore UK government expenditure:

```
1 expenditure.byService.Defence.inTermsOf.GDP
```

The dimensions of the cube are government services, years and value type (adjusted, nominal, per GDP). Here, we select the Defence service and GDP value type. Our implementation does not yet use a standardized data cube storage and so adding another data cube source currently requires implementing a new type provider.

Tabular Data Type Provider

Our second type provider allows users to construct queries to explore data in tabular formats such as CSV files. It is based on the theory developed Petricek [15]. Unlike the data cube provider, the provider for tabular data does not just allow selecting a subset of the data, but it can be used to construct SQL-like query. Consider the example from Figure 1:

```
1 olympics.'filter data'. 'Games is'. 'Rio (2016)'. then
2   .'group data'. 'by Team'. 'sum Gold'. 'sum Silver'. then
3   .'sort data'. 'by Gold descending'
```

The example works with a CSV file that records individual medals awarded in Olympic games. The chain constructs a query that selects rows corresponding to the Rio 2016 Olympics and then calculates total number of gold and silver medals for each team (country) before sorting the data.

When using the provider, the user specifies a sequence of operations. Members such as `'filter data'` or `'group data'` determine the operation type. Those are followed by operation parameters. For example, when grouping data, we first select the key and then add a number of aggregations to calculate over the group. Unlike SQL, the provider only allows users to choose from pre-defined aggregations such as calculating the sum, average or the number of distinct values. As illustrated in Section 7, this still allows us to construct a wide range of practical queries.

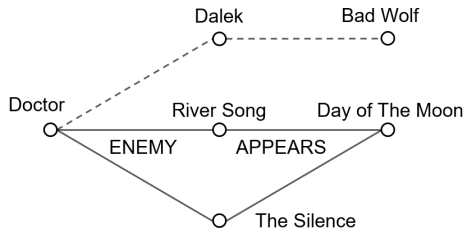


Figure 6. Exploring graph databases. The user specifies a path through the data, possibly with placeholders to select multiple nodes.

Graph Database Type Provider

Our third type provider allows users explore data from graph databases. A graph database consists of nodes representing entities and relationships between them. In the context of data journalism, graph databases have been used for example in The Panama Papers reporting [13].

The following example explores a database of Doctor Who characters and episodes. It retrieves all enemies of the Doctor that appear in the Day of the Moon episode:

```

1 drwho.Character.Doctor
2   . 'ENEMY OF' . '[any]'
3   . 'APPEARED IN' . 'Day of the Moon'

```

We start from the Doctor node and then follow two relationships. We use 'ENEMY OF' . '[any]' to follow links to all enemies of the Doctor and then specify 'APPEARED IN' . 'Day of the Moon' to only select only such enemies that appear in a specific episode. The resulting query is illustrated in Figure 6.

The provider works with any graph database and generates members automatically, based on the data in the database. In the above example, ENEMY OF and APPEARED IN are labels of relations and Doctor and Day of the Moon are labels of nodes. The [any] member defines a placeholder that can be filled with any node with the specified relationships. The results returned by the provider is a table of properties of all nodes along the specified path. As discussed in the next section, such table can be further queried using the tabular data type provider.

USE CASES

USER STUDY

Our goal is to develop an easy-to-learn tool that journalists and other non-programmers can use for producing transparent data analyses. To evaluate the extent to which The Gamma achieves this, we conduct a user study. We recruit participants among the operations and business team of a non-profit research organization and investigate three research questions.

RQ1: Can non-programmers explore data with The Gamma?

We give participants one of four simple data exploration tasks and see if they can complete the task using The Gamma and, possibly, how much assistance they need.

RQ2: Can knowledge transfer between data sources?

Our first hypothesis is that users familiar with the iterative prompting principle will be able to use an unfamiliar data source. In two of the tasks, participants are shown one data source and then asked to work with a different one.

RQ3: Can users learn from just code samples?

Our second hypothesis is that users can learn through precolation, i.e. by looking at the source code of published analyses. In one of our tasks, participants are given a code sample, but only a minimal explanation of The Gamma environment.

Study design

We perform between-subjects study to evaluate the first experience of using The Gamma. We recruit 13 participants (5 male, 8 female) from a business team of a research lab working in various non-technical roles (project management, partnerships, communications) including one former journalist.

We split participants into 4 groups. We first gave a brief 5 minute demonstration of The Gamma and then asked participants to complete a task. Finally, we conducted a short semi-structured group interview and later sent participants a follow-up questionnaire. The four tasks were:

- *Expenditure*. Participants were given a demo using *worldbank*. They were asked to use the *expenditure* data source to compare the UK government spending on “Public order and safety” and “Defence” in terms of GDP.
- *Lords*. Participants were given a demo using *worldbank*. They were asked to use the *lords* data source (a table with House of Lords members expenses) to find a members representing London with the highest travel costs.
- *Worldbank*. Participants were given a minimal explanation of The Gamma environment and a code sample using *worldbank*. They were asked to solve a different task using the *worldbank* data source.
- *Olympics*. Participants were given a demo using *olympics*. They were asked to solve a more complex problem using the same data source.

We let participants work independently, but offered guidance if they got stuck for longer period of time. Tasks *expenditure* and *lords* aim to answer the questions RQ1 and RQ2; the task *worldbank* aims to answer RQ1 and RQ3. In *olympics*, we test RQ1 using a more complex data source and we ask further questions to explore their understanding of the data source.

	Task	Kind	Done	Notes
#1	expenditure	cube	●	Obtained one of two data series
#2	expenditure	cube	●	Explored further data series
#3	expenditure	cube	●	Explored further data series
#4	expenditure	cube	●	With hint to use another member
#5	expenditure	cube	●	Explored further data series
#6	worldbank	cube	●	With general syntax hint
#7	worldbank	cube	●	Completed very quickly
#8	worldbank	cube	●	Extra time to find data
#9	lords	table	●	Struggled with composition
#10	lords	table	●	Completed very quickly
#11	lords	table	●	With a hint to avoid operations
#12	olympics	table	●	With a hint to avoid operations
#13	olympics	table	●	Hints about ‘then’ and operations

Table 1. Overview of the work completed by individual participants. The marks denote: ● = required some guidance, ● = partially completed.

Results

Table 1 summarizes the work done by the study participants. For each participant, we record the task, the kind of data source used in the task and the level of completion. For participants who needed assistance, the notes section details the help given. We analyse the hints in the next section. Table 2 shows the results of the follow-up questionnaire.

RQ1: Can non-programmers explore data with The Gamma?

Three results allow us to answer RQ1 in the affirmative. First, 7 out of 9 participants agree or strongly agree that they “found the system easy to use”. Second, participants spent between 10 and 25 minutes (average 17 minutes) working with The Gamma and all 12 out of 13 participants completed the task; 6 participants required some assistance, but 3 of those faced one issue (discussed later) that could be covered in the introductory presentation. Third, a number of participants shared positive comments in the semi-structured interview.

Participant #3 found the system simple, but also points out an issue about data provenance, which we discuss later:

“This is actually pretty simple to use. You think about the logic of what you’re actually asking and then you try to get it into the format you can. But knowing where it comes from would tell you how to trust it.”

Similarly, participant #2, notes that The Gamma alleviated their unease about code:

“For somebody who does not do coding or programming, this does not feel that daunting. It’s not like you’re giving me large screen full of code, which is reassuring.”

Finally, participant #5 suggested the system could be used as an educational tool for teaching critical thinking with data. They answer a follow-up question about what training materials would the students need as follows:

“I don’t think they’d need more than 5 minute video (..) this is the data source, this is what’s in there.”

RQ2: Can knowledge transfer between data sources?

Our study does not conclusively answer RQ2. There is some evidence in favor of a positive answer. In the practical part,

two of the tasks (*expenditure* and *lords*) used a different data source in the introductory presentation than the one that the participants were asked to use. Participants were able to complete those tasks, although *lords* has been more challenging as it involves a more complex data source. In the interview, participant #2 also gives a positive answer:

“I found it quite easy to translate what you showed us in the demo to the new dataset. I thought it was quite easy to just get how that works.”

Negative evidence is offered by the follow-up questionnaire. When asked whether they would know how to approach a task using a new data source, 5 out of 9 participants disagree or strongly disagree that they would know how to approach it without any further guidance. Participants did not believe that knowledge can easily transfer to another data source.

RQ3: Can users learn from just code samples?

A positive answer to RQ3 is supported by the *worldbank* task results, the follow-up questionnaire and interview comments. In the task, participants were given only a minimal demo of the iterative prompting principle together with print-out of 2 code samples. All three were able to complete a related task using the same data source. In the follow-up questionnaire, only 2 out of 9 participants disagree or strongly disagree that they would know how to approach a task using an unfamiliar data source when given “a number of code samples”.

In the semi-structured interview, participants were asked what would be the most useful format of educational material about The Gamma (code samples, video tutorials, etc.). Participant #7 noted that “a video would just be this [i.e. a code sample] anyway”, while participant #13 (former journalist) answered:

“I think providing one video of one example is good and maybe a couple of little examples of code where people can see the kind of things you can do.”

This is aligned with our design goal. Once the user understands the iterative prompting principle (which can be done through a video tutorial), they can learn how to use any specific data source just from code samples.

Further observations

In this section, we briefly discuss a number of observations about The Gamma design that emerged from the study, some of which suggest ways of improving the system.

Making complex things possible may hurt

As discussed earlier, The Gamma language supports operations such as `take(5)`. Most type providers never generate those, but the provider for working with tabular data is an exception. When filtering data, the provider allows specifying a condition on numerical attributes such as `olympics.filter data'.Year is greater than'(2004)`.

Three participants (#11, #12, #13) struggled to complete a task, because they initially attempted to use those operations. However, those violate the iterative prompting principle as one cannot type `'` after `'Year is greater than'`. This suggests that we should avoid operations in type providers for data access, even if it limits the expressive power of the type provider.

Questionnaire item	Avg	Sdv
Considering your experience & possible use of The Gamma:		
I found the system easy to use.	2.11	0.87
Journalists will be able to use it to analyse data.	2.67	1.05
Readers will be able to critically examine the analyses.	2.44	0.68
Thinking about the usability of the system:		
The resulting programs are easy to understand.	2.33	0.82
The text editor for creating them was easy to use.	2.33	0.94
Finding the right item in the auto-complete was easy.	2.56	1.42
I would know how to approach a task using a new data source		
If given a comprehensive video tutorial.	2.00	1.24
If given a number of code samples.	2.78	1.13
Without any further guidance.	3.56	1.17

Table 2. Summary of follow-up questionnaire responses using a 5-point Likert scale (1 = strongly agree, 5 = strongly disagree) over 9 subjects.

Benefits and drawbacks of text

The Gamma is based on text to aid transparency. Text hints that there is no hidden state and the reader sees the full code. The study suggests that using a text editor has both benefits and drawbacks compared to alternatives such as structured editors [20, 14, 10]. Most participants had no difficulty navigating around source code, making edits or deleting code fragments, which is arguably harder in an unfamiliar structured editor.

We observed two issues in the study. Participant #2 struggled with correct indentation, starting a second command with more indentation than needed and participant #6 had a syntax error in an unrelated command, which prevents charts from rendering. Some participants used the text editor effectively, e.g. participant #5, who used copy-and-paste to fetch the same data series for multiple countries.

How users understand the ‘then’ member

When interviewing participants who worked with a tabular data source, we asked about their understanding of the *then* member. This is a regular member generated by the type provider (i.e. not a keyword), but it has a special meaning. Consider the following example which calculates the average travel expenses and total number of representatives per county using the UK House of Lords expenses data:

```
1 expenses.'group data'.'by County'
2   .'average Travel Costs'.'count distinct Name'.then
```

The *then* member is used to complete a part of a query where the user can continue adding items to build a list. Here, we select two aggregations to be calculated for each county before choosing *then* and applying other operations such as sorting.

Two participants (#12 and #13) initially thought that *then* is used to split a command over multiple lines, but rejected the idea after experimenting and noting that they can insert line breaks elsewhere. One correctly concluded that it “allows us to chain together the operations” of the query. Participant #13 needed a hint about using the *then* member and later reflected:

“When you explained about the ‘dot then’ that was a really useful thing to know. When I found that, I was like this is fine, this is doable. If I knew this from the start, it would [have been easier].”

This comment summarizes an important fact. Although iterative prompting allows the users to start exploring new data sources, the domain specific languages used by more complex data sources have their own design principles that users need to understand to use the data source effectively.

Conclusions

The motivation behind The Gamma is to make simple programmatic data exploration accessible to journalists and other non-experts. Main-stream programming languages have the benefit of reproducibility and transparency, but are difficult to use. Our study shows that non-experts are able to solve basic data exploration tasks using The Gamma. When asked whether The Gamma is something that journalists could learn how to use, participant #13 (a former journalist) answered:

“Yeah, I think so. There’s a lot of effort going into data journalism that programming could make much quicker, but I was always nervous about code. (...) Something like this would really simplify things.”

The Gamma is based on the iterative prompting principle which lowers the barrier to entry, but it is worth noting that it does not fully eliminate complexity involved in data querying. Instead, it provides a way of structuring the complexity. Iterative prompting makes it easy to get started, but complex data sources will inevitably require additional learning. The Gamma makes this easier by allowing users to learn from published code samples.

DISCUSSION

Data provenance (where data comes from) What are the data sources available (how to start)?

Study limitations

exploratory in nature so we do not make any quantitative claims about effects

not comparing against other systems

Design principles

How well did we do wrt design principles?

Design issues

future challenges and limitations of the model - such as issues when modifying code in the middle of the call chain

CONCLUSIONS

ACKNOWLEDGMENTS

Yo

REFERENCES

- [1] 2018. *Proceedings of the 2nd European Data and Computational Journalism Conference*. University College Dublin.
- [2] David Blood. 2018. Recycling is broken – notebooks. (11 2018). <https://github.com/ft-interactive/recycling-is-broken-notebooks>
- [3] Axel Bruns, Tim Highfield, and Rebecca Ann Lind. 2012. Blogs, Twitter, and breaking news: The produsage of citizen journalism. *Produsing theory in a digital world: The intersection of audiences and production in contemporary theory* 80, 2012 (2012), 15–32.
- [4] Matthew Conlen and Jeffrey Heer. 2018. Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web. In *The 31st Annual ACM Symposium on User Interface Software and Technology, UIST 2018, Berlin, Germany, October 14-17, 2018*, Patrick Baudisch, Albrecht Schmidt, and Andy Wilson (Eds.). ACM, 977–989. DOI: <http://dx.doi.org/10.1145/3242587.3242600>
- [5] William Davies. 2017. How statistics lost their power - and why we should fear what comes next. The Guardian. (19 Jan 2017). Retrieved March 6, 2020 from <https://www.theguardian.com/politics/2017/jan/19/crisis-of-statistics-big-data-democracy>.
- [6] Bob Franklin. 2012. THE FUTURE OF JOURNALISM. *Journalism Studies* 13, 5-6 (2012), 663–681. DOI: <http://dx.doi.org/10.1080/1461670X.2012.712301>
- [7] Jonathan Gray, Lucy Chambers, and Liliana Bounegru. 2012. *The data journalism handbook: how journalists can use data to improve the news*. O'Reilly Media, Inc.
- [8] G. E. Kaiser and P. H. Feiler. 1987. An Architecture for Intelligent Assistance in Software Development. In *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*. IEEE Computer Society Press, Washington, DC, USA, 180–188.
- [9] Haeyoun Park Larry Buchanan and Adam Pearce. 2017. You Draw It: What Got Better or Worse During Obama's Presidency. New York Times. (15 Jan 2017). Retrieved March 3, 2020 from <https://www.nytimes.com/interactive/2017/01/15/us/politics/you-draw-obama-legacy.html>.
- [10] Eyal Lotem and Yair Chuchem. 2018. Lamdu Project. (2018). <https://github.com/lamdu/lamdu>
- [11] Edith Manosevitch and Dana Walker. 2009. Reader comments to online opinion journalism: A space of public deliberation. In *International Symposium on Online Journalism*, Vol. 10. 1–30.
- [12] Klaus Meier. 2009. Transparency in Journalism. Credibility and trustworthiness in the digital future. In *Actas II Congreso The Future of Journalism*.
- [13] Bastian Obermayer and Frederik Obermaier. 2016. *The Panama Papers: Breaking the story of how the rich and powerful hide their money*. Oneworld Publications.
- [14] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live Functional Programming with Typed Holes. *PACMPL* 3, POPL (2019). DOI: <http://dx.doi.org/10.1145/3291622>
- [15] Tomas Petricek. 2017. Data exploration through dot-driven development. In *31st European Conference on Object-Oriented Programming (ECOOP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [16] Tomas Petricek, Gustavo Guerra, and Don Syme. 2016. Types from Data: Making Structured Data First-class Citizens in F#. In *Proceedings of Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, 477–490. DOI: <http://dx.doi.org/10.1145/2908080.2908115>
- [17] Tomas Petricek, Bahareh R. Heravi, Jennifer A. Stark, and et al. 2017. *Proceedings of the European Data and Computational Journalism Conference*. University College Dublin.
- [18] Advait Sarkar and Andrew Donald Gordon. 2018. How do people learn to use spreadsheets? (Work in progress). In *Proceedings of the 29th Annual Conference of the Psychology of Programming Interest Group (PPIG 2018)*. 28–35.
- [19] Don Syme, Keith Battocchi, Kenji Takeda, Donna Malayeri, and Tomas Petricek. 2013. Themes in Information-rich Functional Programming for Internet-scale Data Sources. In *Proceedings of Workshop on Data Driven Functional Programming (DDFP '13)*. ACM, 1–4. DOI: <http://dx.doi.org/10.1145/2429376.2429378>
- [20] Gerd Szwillus and Lisa Neal. 1996. *Structure-based editors and environments*. Academic Press, Inc.