The Gamma: Data Exploration through Iterative Prompting

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 the 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 [8]. 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 [14].

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.

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

The Gamma brings together recent programming language innovations for working with data [54], work on making programming accessible to non-programmers [37, 38] and work on tools for journalists such as Idyll [6]. Related work includes a range of visual and programmatic tools used by data scientists, work that aims to simplify programming or explore alternative paradigms and various predictive tools that assist users in completing programming or data exploration tasks.

Data science tools

Data exploration is typically done using spreadsheets, comprehensive visual tools or via code using notebooks. Notebooks such as Jupyter [29] are also popular with technically skilled journalists as they allow combining code with commentary and visual outputs. However, notebooks can lead to messy hard to reproduce code [17, 42].

Visual data analytics tools [19, 7] such as Tableau [59] do not require programming skills, but require mastering a range of different interactions. They also make it hard to see what data transformations have been done. Several systems for data exploration, e.g. Potter’s Wheel and Wrangler [45, 26], and visualization, e.g. Lyra [49], provide a GUI, but record a transparent script that can be seen and modified by the user.

Easier programming tools

We aim to design a programming environment that is easy to use and learn. Many approaches to this goal have been tried. Victor [57, 56] introduced design principles that inspired many to build live programming systems [46, 30, 13] that give immediate feedback to help programmers understand how code relates to output and exploratory systems [27, 28] that assist with completing open-ended tasks.

To avoid difficulties with editing code as text, some systems use structured editors [55, 40, 34]. In Subtext [10, 11] the language itself is co-designed with the editor to make the interactions with code more natural. Finally, programming can be made easier by designing high-level declarative abstractions. Tea does so for statistical analyses [24] and Satyanarayanan et al. [51, 50] does so for interactive data visualizations.

The Gamma is live in that it gives an instant preview of the results. It is text-based, although it is syntactically simple. The code written in The Gamma is high-level and so work on declarative abstractions is a valuable inspiration.

Code completion and assistants

A key component in The Gamma design is the use of auto-complete for offering possible operations. Code completion has been pioneered by Kaiser [25], but our work is more directly inspired by the work on type providers [54, 43, 41], which utilize code completion to integrate external data into a statically typed programming language.

When using type providers, completions are offered as the user types code. However, completion tools can also recommend scripts when users begin interacting with data [18, 16]. Alternatively, they can also be guided by natural language as for example Eviza [52], which uses natural language for data exploration. CodeMend [47] uses natural language for more general code edits, but uses data visualization as a case study. Finally, data entry interfaces based on completion such as Dasher [58] utilize an interaction principle similar to our iterative prompting.

Programming without writing code

A number of tools for data exploration use alternative approaches to programming where the user does not write code. In programming by example [33], the user gives examples of desired results. In data science, this approach has been used for example for specifying data transformations in spreadsheets and data extraction [15, 32].

In direct manipulation [22], a program is specified by directly interacting with the output. This has been used in the visual domain [20], but also for data querying [53, 3]. The VQE language [9] also considers how to allow code reuse and modification in this context. Direct manipulation can also support data exploration by letting users partially edit queries, e.g. by changing quantifiers as in DataPlay [1].

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 journalists, review of literature on data journalism, e.g. [14, 44, 21] and more general trends in journalism.

Open Journalism

Journalism continually develops and responds to the many challenges it faces [12]. 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 [36]. 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 [35, 4] and the development of new interactive formats [31]. 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.
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 [48] 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

<table>
<thead>
<tr>
<th>Name, County, Days Attended, Days Away, Travel Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lord Adonis, London, 8, 0, 504</td>
</tr>
<tr>
<td>Baroness Afshar, Yorkshire, 2, 0, 0</td>
</tr>
<tr>
<td>Lord Alderdice, Oxfordshire, 3, 0, 114</td>
</tr>
<tr>
<td>Lord Alli, London, 5, 0, 0</td>
</tr>
<tr>
<td>Baroness Amos, London, 3, 0, 0</td>
</tr>
</tbody>
</table>

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

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

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:

```
expenses
.'filter data'. 'County is' .Kent.
```

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>County</th>
<th>Days Attended</th>
<th>Days Away</th>
<th>Travel Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lord Astor of Hever</td>
<td>Kent</td>
<td>7</td>
<td>0</td>
<td>85</td>
</tr>
<tr>
<td>Lord Freud</td>
<td>Kent</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lord Harris of Peckham</td>
<td>Kent</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Baroness Noakes</td>
<td>Kent</td>
<td>6</td>
<td>0</td>
<td>135</td>
</tr>
</tbody>
</table>

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):

```
expenses
.'filter data'. 'County is' .Kent.then
.'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.

```
expenses
.'filter data'. 'County is' .Kent.then
.'sort data'. by Travel Costs descending'.then
```

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 journalists 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 [48] 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:

```plaintext
let topTravel =
  expenses
  .sortBy("Travel Costs descending")
  .take(5)
  .getSeries()
  .withKey "Travel Costs"
  .andValue "Travel Costs"
charts.column(topTravel)
  .setTitle("House of Lords members by travel expenses")
  .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]`.

**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.
As illustrated in Figure 5, the provider allows users to first choose a country or a year and then an indicator. Figure 6. Exploring graph databases. The user specifies a path through the data, possibly with placeholders to select multiple nodes.

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-alphanumerical 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 Monaco 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 Bank 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:

```plaintext
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:

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

The dimensions of the cube are government services, years and 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. 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:

```plaintext
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.

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 [39].

---

1. https://microsoft.github.io/monaco-editor/
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:

```javascript
let data = olympics
  .group data'.by Team'.sum Gold'.then
  .sort data'.by Gold descending'.then
  .paging.skip(43).take(4)
  .explore
  .get series'.with key Team'.and value Gold'

let phelps = olympics
  .filter data'.Athlete is'.Michael Phelps'.then
  .get series'.with key Athlete'.and value Gold'

charts.bar(data.append(phelps))
.setColors(["#aec7e8","#aec7e8","#f1f7b4"])```

The core of the data analysis is done in two commands. The first counts gold medals by countries and uses paging to fetch 4 countries with suitable number of medals. In the second, we abuse the grouping operation to aggregate data for just a single episode. The two data series are then assigned to local variables (for readability) and passed to the chart.columns function. In this case study, the core data exploration can be constructed via iterative prompting, but building a chart requires invoking the append operation and specifying colors using a list.

**USE CASES**

The Gamma aims to simplify programmatic data exploration while keeping enough expressive power to allow users to create interesting data visualizations. We consider the expressive power in this section and assess simplicity in the next one. We used The Gamma to analyse the UK government expenditure, activities of a research institute, Olympic medals and information about the Doctor Who series. In this section, we consider two interesting larger examples.

**If Michael Phelps was a Country**

Michael Phelps has won so many medals that media compared him to countries. Using the tabular data provider, we construct a chart shown in Figure 7 that produces a country league table including Michael Phelps:

```javascript
let data = olympics
  .group data'.by Team'.sum Gold'.then
  .sort data'.by Gold descending'.then
  .paging.skip(43).take(4)
  .explore
  .get series'.with key Team'.and value Gold'

let phelps = olympics
  .filter data'.Athlete is'.Michael Phelps'.then
  .get series'.with key Athlete'.and value Gold'

charts.bar(data.append(phelps))
.setColors(["#aec7e8","#aec7e8","#f1f7b4"])```

Lines 1-2 use the graph database provider to find all paths linking the Doctor node with any character linked via ENEMY OF, followed by any episode linked by APPEARED IN. This produces a table that can be analysed using the tabular data provider by choosing the explore member. For each character (the villain) we count the total number of distinct episodes in the table. The result is shown in Figure 7.

The logic is not trivial, but it performs a fairly sophisticated data analysis that involves a graph database query followed by an SQL-like data aggregation. The code can be constructed using iterative prompting (with the exception of the numbers in paging). It can also be constructed gradually and instantaneous previews support the user while doing so.

**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

---

Links removed to preserve anonymity.

[1] npr.org/sections/thetorch/2016/08/14/489832779/

[2] Available at: http://gallery.thegamma.net/86/
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 lords. 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.

**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

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

Table 1. Overview of the work completed by individual participants. The marks denote: 🌋 = required some guidance. 🌋 = partially completed.
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 though 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 design that emerged from the study, some of which suggest ways of improving the system.

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.

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

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 [55, 40, 34]. 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 the wrong indentation. 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:

```
expenses.'group data'.by County'
'.'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].”
We design a programming environment that can make 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**

We design a programming environment that can make programmatic data exploration accessible to journalists. This is aligned with recent developments in journalism, which aims to build trust through transparency and greater reproducibility as well as provide meaningful ways of engagement.

**Evaluating The Gamma**

Data exploration environments are complex systems that do not yield to simple controlled experimentation. We evaluate The Gamma using criteria inspired by Olsen [23].

- **Importance.** Open journalism has the potential to make factual claims backed by data more commonplace and enable wider audience engage with such claims. As such, we contribute towards solving an important societal issue.

- **Expressive leverage.** The Gamma reduces the number of concepts that the user needs to master to just the iterative prompting principle, although complex data sources may add more concepts through their domain specific languages.

- **Empowering new participants.** As demonstrated by our user study, The Gamma allows non-experts, including those not comfortable with code, to perform basic programmatic data exploration tasks.

- **Generality.** Finally, our case studies show that The Gamma can be used to solve a wide range of data manipulation tasks involving many data sources including tabular data, data cubes and graph databases.

**Remaining design issues**

There remain a number of aspects of data exploration in the context of journalism that The Gamma does not address. Two of those, data provenance and data availability were also observed by the participants in our study.

**Data provenance**

Data sources such as olympics or worldbank are defined when initializing The Gamma programming environment, but the system does not currently show where such data comes from. For some data sources, the source is e.g. a CSV file published by the government. In this case, we can easily show the source. However, other type providers may pre-process data. Displaying data source in such cases would require more sophisticated provenance tracking [5].

**Data availability**

In the current version, The Gamma does not have a way of informing the user what data sources are available. In other words, the user needs to know the first identifier, such as olympics, to get started. We could address this by choosing a data source as the first step of iterative prompting and perhaps typing .olympics. However, a more fundamental issue is finding the data source in the first place. This could be partly addressed by a type provider for a curated online database such as Enigma Public⁹. Providing access to open government data repositories such as data.gov.uk is more appealing, but very challenging due to their unstructured nature.

**Evaluation limitations**

The evaluation conducted in this paper is qualitative and exploratory in nature. In particular, we do not make any quantitative claims about the usability of The Gamma and its learning curve. Our study also focused on the core iterative prompting interaction, but our case studies often required using other features such as operations with parameters.

Although The Gamma is open-source, it has not been deployed in production in a newsroom so far. This would lead to valuable insights, but it requires finding a suitable fortuitous opportunity. Finally, we also do not compare the usability of The Gamma with the usability of other popular systems such as Tableau [59]. It would be possible to set tasks that can be completed in both systems, but the systems have very different aims making such comparison problematic.

**CONCLUSIONS**

We present The Gamma, a simple data exploration environment. The Gamma is motivated by the needs of journalists working with data, who need to produce transparent and easy-to-understand reports backed by data that are transparent and engaging, all the while facing tight deadlines.

The Gamma is based on a single interaction principle, iterative prompting, can be used to complete a wide range of data exploration tasks using a variety of data sources including tabular data, data cubes and graph databases. The design lowers the barrier to entry for programmatic data exploration and it makes it easy to learn the system independently from examples and by experimentation.

⁹https://docs.enigma.com/public
REFERENCES


