The Gamma: Data Exploration through Iterative Prompting

ANONYMOUS AUTHOR(S)

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.

CCS Concepts: • **Human-centered computing → Interaction paradigms**; • **Software and its engineering → Integrated and visual development environments**; *Domain specific languages.*

Additional Key Words and Phrases: data exploration; end-user programming; data journalism; programming languages; type providers

ACM Reference Format:
Anonymous Author(s). 2020. The Gamma: Data Exploration through Iterative Prompting. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 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 [11]. To reverse this trend, we need tools that let non-experts, including journalists and other information-literate citizens, produce transparent, engaging data analyses that are easy to interpret without requiring expert programming skill [18]. The design of such data exploration tool poses a unique mix of chellenges. First, the tool needs to have a very low barrier to entry. Second, it needs to support a wide range of data sources in a uniform way. Third, the resulting data analyses need to assist readers in learning how to reproduce the work and verify the claims it makes.

We contribut The Gamma, a text-based data exploration environment for non-experts. The Gamma is based on a single, easy to understand interaction principle and provides a uniform access to a range of data sources including data tables, graph databases and data cubes. The resulting analysis 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 data.

Iterative Prompting. The key idea in The Gamma, which we term the *iterative prompting* principle, is that all valid data exploration scripts can be constructed by repeatedly choosing from a list of offered options. This way, non-programmers can write entire scripts through auto-complete, without learning a programming language, but they still produce transparent and reproducible source code. In other words, iterative prompting turns auto-complete from a programmer assistance tool into a non-expert programming mechanism. A crucial feature is that iterative prompting only offer operations that are valid in a given context and that it offer all such operations; it is both correct and complete.

Manuscript submitted to ACM

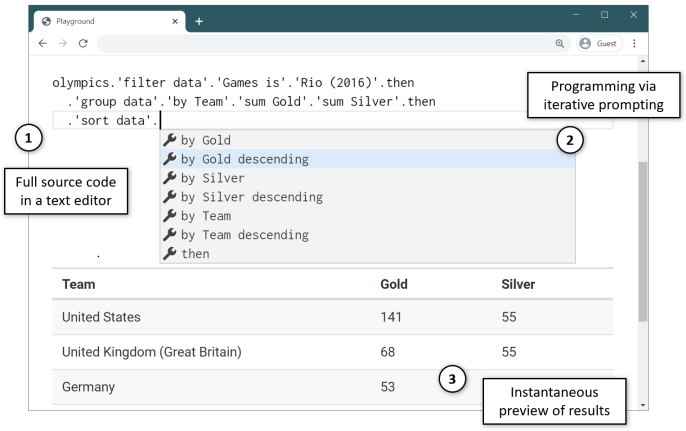


Fig. 1. Obtaining teams with the greatest number of gold medals from Rio 2016 Olympics with a reproducible The Gamma script (1), contextual iterative prompting mechainsm offering ways of sorting the data (2) and an instant preview of results (3).

Data Exploration. The Gamma focuses on data exploration of the kind illustrated in Figure 1. The user accesses data available in a structured format. They make several experiments to find an interesting way of looking at the data, e.g. by applying different aggregations or filters. They may choose to view the results as a table or a basic chart before publishing their analysis. The Gamma makes such simple programming with data simple enough for non-experts. Scraping and cleaning of messy data or building rich data visualizations is outside of the scope of this paper, but exposing those using the iterative prompting approach is an interesting future challenge.

Contributions. The Gamma is available (non-anonymously) at http://thegamma.net, both as a JavaScript library and a hosted data exploration service. In this paper, we describe and evaluate the design principles behind the project:

- We introduce the iterative prompting principle in The Gamma and show that it can be used for querying of distinct data sources including data tables, graph databases and data cubes (Section 3).
- We show that iterative prompting is a complete and correct program construction method and consider how it lowers the barrier to entry and allows learning through examples, without expert guidance (Section 4).
- We evaluate the system through a number of case studies (Section 7) and a user study (Section 8). Our study shows that non-programmers can use The Gamma to construct non-trivial data queries and ascertains the extent to which users can, (i) learn from examples and (ii) transfer knowledge between tasks.

2 RELATED WORK

The key contribution of our work is that it develops a new, fundamentally different, way of using the established auto-completion mechanism. Unlike most past work dating back to Kaiser [29], we do not view it as a programmer assistance tool, but as an interaction mechanism through which non-experts can create entire programs. We build on recent research on information-rich programming [61] and aim to make those advances available to non-programmers [42, 43], in the context of data exploration as done, most notably, by journalists [18]. Our work features a novel combination of characteristics in that the iterative prompting we develop (i) is centered around editing and understanding of program code, (ii) its conceptual complexity is reduced to a single basic kind of interaction, yet (iii) it is correct and complete in that it can be used to construct all meaningful programs.

Code Completion for Data Science. A key component in The Gamma is the use of auto-complete for offering possible operations. Our work follows type providers [48, 61], which integrate external data into a static type system of F#, allowing the use of auto-completion; for querying data tables, we utilize the theory developed by Petricek [46]. The key difference in our work is that The Gamma can be used without a programming language expertise.

Most similar to our approach are tools that recommend scripts when users begin interacting with data. Those based on machine learning code completion for domain-specific languages [20, 22] differ in that they do not guarantee completeness, i.e. the user cannot create all possible scripts. Approaches based on natural language can effectively support data exploration or visualization [53, 58], but hide the structure of the underlying language and require more than just selecting options. Conversational agents [15] improve on such work in that they can provide more guidance. Code completion based on machine learning or statistical methods [6, 51] also exists for general-purpose programming languages used by data scientists such as Python [60], providing assistance to expert programmers. Finally, DS.js [67] is interesting in that it enables querying of data on the web; it uses JavaScript with rich contextual code completion.

Notebooks and Business Intelligence Tools. Notebooks such as Jupyter [33], which allow combining source code with commentary and visual outputs, are widely used by data scientists, but require expert programming skills. The Gamma targets non-experts, but could be easily integrated with a multi-language notebook system such as Wrattler [47].

Spreadsheets and business intelligence tools [41, 66] do not involve programming, but require mastering a complex GUI. This is also the case for other visual data analytics tools [10, 23]. In contrast, The Gamma is based on a single kind of interaction. Several visual systems [30, 50, 55] record interactions with the GUI as a script that can be seen and modified by the user. Unlike in The Gamma, the source code does not guide the user in learning how to use the system.

Easier Programming Tools. We aim to build an easy to use and learn programming system. Many approaches to this goal have been tried. Victor [63, 64] introduced design principles that inspired many to build live programming systems [17, 34, 52] that give immediate feedback to help programmers understand how code relates to output and exploratory systems [31, 32] that assist with completing open-ended tasks. A system combining textual language with visualization also exists for graph querying [2]. To avoid difficulties with editing code as text, some systems use structured editors [38, 45, 62]. In Subtext [13, 14] the language itself is co-designed with the editor to make the interactions with code more natural. The Gamma is live in that our editor gives an instant preview of the results.

Many systems simplify programming by designing high-level declarative abstractions, e.g. for interactive news articles [9], statistical analyses [28] or interactive data visualization [56, 57]. The Gamma uses high-level abstractions for data querying, but designing high-level iterative prompting abstractions for other tasks remains future work.

Programming without Writing Code. There are two main approaches to programming where the user does not write code. In programming by example [37], the user gives examples of desired results. This has been used, e.g. for specifying data transformations in spreadsheets and data extraction [19, 36]. In direct manipulation [26], a program is specified by directly interacting with the output. This has been used in the visual domain [24], but also for data querying [5, 59]. The VQE language [12] 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].

Guestures and Data Entry. Although our focus is on program construction, our work can be positioned in the broader context of input methods. Akin to Dasher [65], our system provides a way of navigating through a complete space of options, while on-screen feedforward [3] allows efficient selection in guesture-based interfaces. Those provide compelling alternatives to auto-completion menus, although the efficiency of input methods is not an issue in programming.

3 OVERVIEW

The review of related work suggests that there is an unexplored point in the design space of tools for data exploration. Although various efforts make text-based programming easier, e.g. by providing high-level declarative abstractions, most systems that target non-experts shy away from code, by using either programming by example, natural language or graphical user interface. As spelled out in Section 4, text-based programming has notable benefits for public-facing data analyses such as learnability and transparency. Our work is thus motivated primarily by the question whether we can make text-based programming easy enough for non-experts? The study presented in Section 8 shows that this is, indeed, possible at least for typical data querying tasks. The Gamma consists of a programming language, a web-based coding environment and a number of type providers that enable access to various kinds of data sources. It allows non-experts to create programs using the *iterative prompting* interaction principle – by repeatedly selecting an item from an auto-complete list. We start with a walkthrough of the system (Section 3.1), before looking at details of The Gamma language (Section 3.2) and individual type providers (Section 3.3).

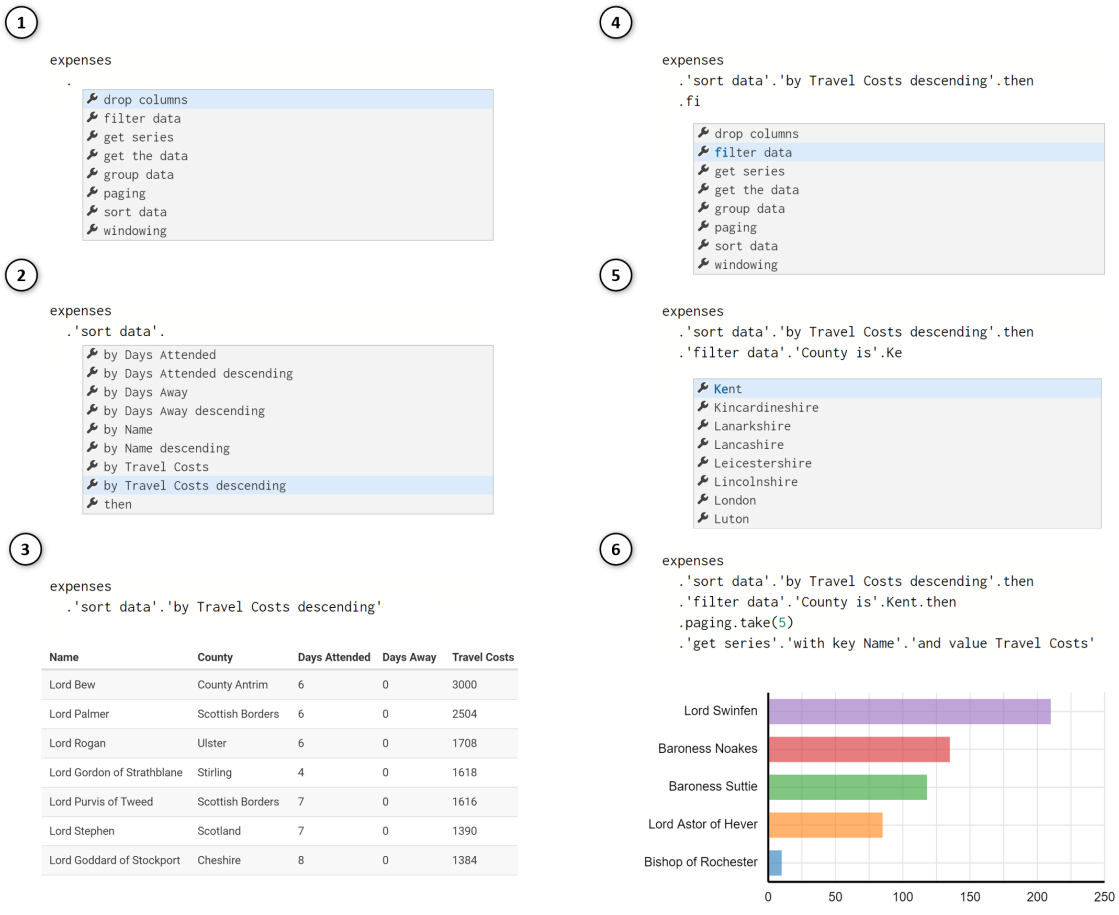


Fig. 2. Constructing a script that charts the top 5 members of the House of Lords for Kent, based on their travel costs.

3.1 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. We want to find out which of the members representing the Kent county spend the most on travel. The following shows a subset of the data:[1]

```
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
```

The data is available as a CSV file. After the analyst imports the file through a web interface, the environment is initialized with code that refers to the imported data as `expenses` and she starts exploring the data using the type provider for tabular data (Section 3.3), following the steps illustrated in Figure 2:

(1) The analyst types '.' (dot) to trigger auto-completion on `expenses`. The type provider offers a list of operations that the analyst might want to perform such as grouping, filtering and sorting.
(2) To find the House of Lords with the largest spending, the analyst chooses the `sort data` operation. Next, she is offered a list of possible arguments based on the schema of the tabular data and chooses the desired one.
(3) The Gamma evaluates the source code on-the-fly and shows a preview of results. After choosing the sorting key, the analyst sees a table with House of Lords members from more remote counties of the UK.
(4) To finish specifying the (possibly compound) sorting key, the analyst chooses `then` and is offered the same list of querying operations as in the first step. To obtain House members from Kent, she chooses `filter data`. To navigate through the offered list more efficiently, she types first two characters of the name.
(5) After selecting `County is` to specify the desired type of condition, the analyst types '.' and is offered a list of options based on the values of the `County` column in the source data set. She types `Ke` and selects `Kent`.
(6) The analyst chooses `then` and is, again, offered the list of querying operation. She uses `paging` to get top `5` records, which requires typing `5` as the argument. She then uses the `get series` operation to obtain a data series associating travel expenses with a name, which is automatically visualized using a bar chart.

The constructed code is not unlike an SQL query, except that the whole script is constructed using iterative prompting, by repeatedly selecting one of the offered members. Those represent both operations, such as `sort by` and arguments, such as `Kent`. The only exception is when the analyst needs to type the number `5` to specify the number of items to take.

3.2 The Gamma Programming Environment

The Gamma consists of a text-based programming language with a web-based coding environment, based on the Monaco editor[2], and type providers that provide access to data tables, graph databases and data cubes.

The Gamma Language. A program in The Gamma 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 such as `paging` or an operation which takes a list of parameters enclosed in parentheses as in `take(5)`. When the member name contains non-alphanumerical characters it is written in quotes such as `'sort by'`.

[1]Full data set has been obtained from https://www.parliament.uk/mps-lords-and-offices/members-allowances/house-of-lords/holallowances/
[2]For more information, see https://microsoft.github.io/monaco-editor/

The Gamma uses a type system to infer what members are available at a given point in a chain. Each expression has a type with a list of members that, in turn, have their own types. The types are not built-in, but are generated by type providers for individual data sources. The programming environment for The Gamma is based on a text editor. When the user types '.' the editor triggers auto-completion and retrives a list of available members based on the type information. The programming environment evaluates scripts on-the-fly and shows a preview as illustrated in Figure 1.

Making Complex Things Possible. As illustrated by the `take(5)` operation, there is a handful of situations where The Gamma does not yet fully support the iterative prompting principle. The language supports a small number of other features that can be used by more advanced users through text editing:

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

First, The Gamma allows operations with parameters such as `take(5)` 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. The `let` construct can be used to define (immutable) variables and The Gamma also supports lists written as `[1,2,3]`. Advanced language features are currently used when building custom charts, but we expect that a charting library compatible with iterative prompting would alleviate the need for most of those.

3.3 Type Providers for Data Querying

The Gamma can be extended to support any kind of data source by implementing a *type provider*. Conceptually, a type provider defines a domain-specific language for exploring data of a particular kind. Techincally, it generates object types with members (such as `sort by` or `Count is`) that are accessed via iterative prompting. We describe type provider for exploring data cubes (inspired by an example from Syme et al. [61]), tabular data (based on theory developed by Petricek [46]), and a novel type provider for exploring graph databases.

Data Cube Type Provider. Our first type provider allows users to explore data form a data cube. For example, the World Bank[3] 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 uses the provider to access CO_2 emission data for United States:

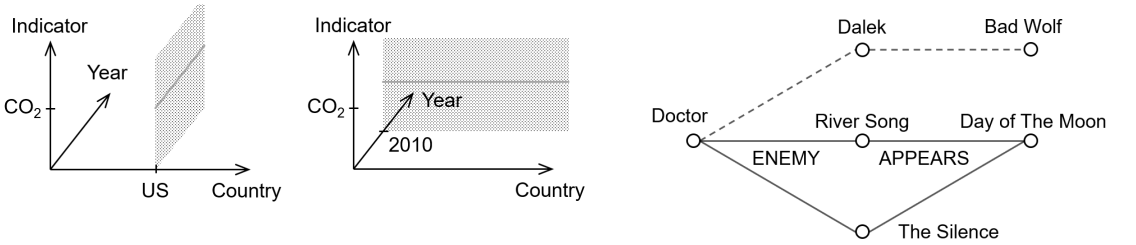
```
worldbank.byCountry.'United States'.'Climate Change'.'CO2 emissions (kt)'
```

As illustrated in Figure 3a, the provider allows users to select a data series from the data cube. Choosing `byCountry.'United States'`, restricts the cube to a two-dimensional plane. We then choose an indicator category and a specific indicator `'CO2 emissions (kt)'`, obtaining a time series with years as keys and emission data as values. Similarly, we could first filter the data by a year or an indicator. The same mechanism can be used for exploring the UK government expenditure:

```
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. As there is no widely used standard format for data cubes, adding another data cube to The Gamma currently requires implementing a new type provider.

[3]The data is retrived using an API available at: https://data.worldbank.org/



(a) Exploring World Bank data using the data cube type provider, users choose values from two dimensions to obtain a data series.

(b) To query graph data, the user specifies a path through the data, possibly with placeholders to select multiple nodes.

Fig. 3. Design of type providers for exploring data cubes and graph databases.

Tabular Data Type Provider. Our second type provider allows users to construct queries to explore data in tabular formats such as CSV files. 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:

```
olympics.'filter data'.'Games is'.'Rio (2016)'.then
  .'group data'.'by Team'.'sum Gold'.'sum Silver'.then
  .'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 choose 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 is sufficient to construct a wide range of practical queries.

Graph Database Type Provider. Our third type provider allows users to explore data from graph databases, which store 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 [44]. 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:

```
drwho.Character.Doctor
  .'ENEMY OF'.'[any]'
  .'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 select only enemies that appear in a specific episode. The resulting query is illustrated in Figure 3b.

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 illustrated by an example discussed in Section 7, the returned table can be further queried using the tabular data type provider.

4 DESIGN

4.1 Journalism

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[4], review of literature on data journalism, e.g. [18, 25, 49] and more general trends in journalism.

Journalism continually develops and responds to the many challenges it faces [16]. 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 [40]. 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 [7, 39] and the development of new interactive formats [35]. 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.

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.

4.2 Lowering barriers

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.

[4]Citations removed to preserve anonymity.

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.

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. 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.

4.3 Learnign without experts

Sarkar [54] 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.

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 [54] 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.

4.4 Correct and complete

any program can be created by autocomplete, all programs created by autocomplete are correct

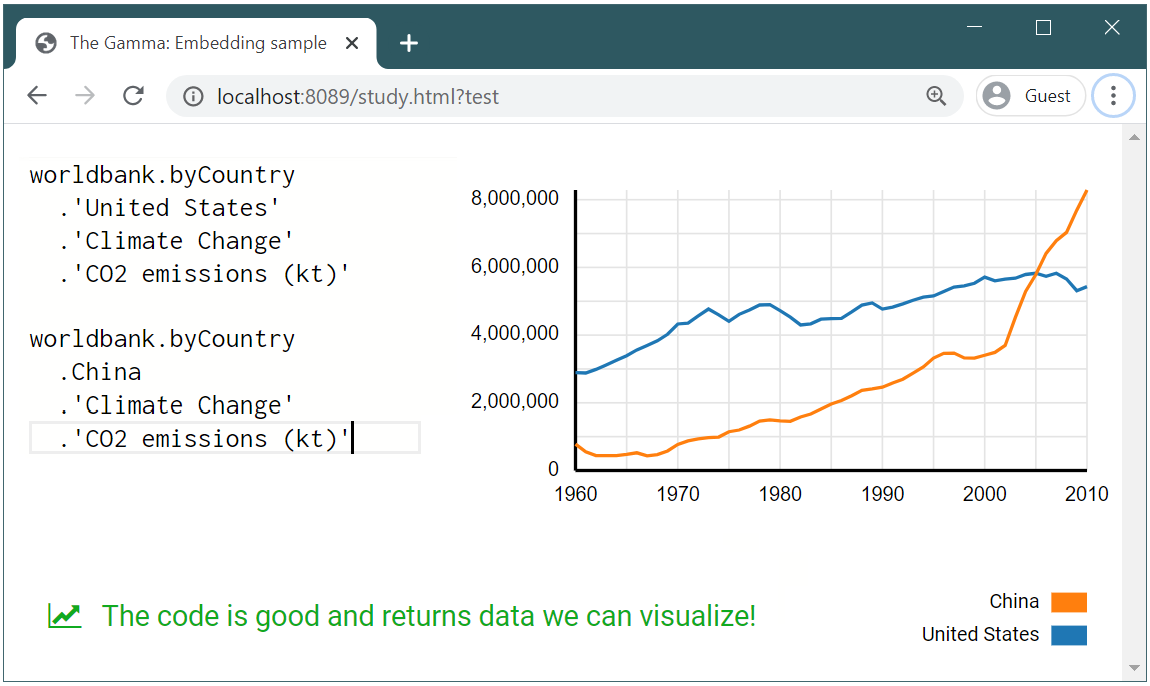


Fig. 4. The Gamma environment with an automatically generated chart comparing CO2 emissions of China and United States.

5 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[5]. In this section, we consider two interesting larger examples.

5.1 If Michael Phelps were a Country

Michael Phelps has won so many medals that media compared him to countries.[6] Using the tabular data provider, we construct a chart shown in Figure 5 that produces a country league table including Michael Phelps:[7]

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

let phelps = olympics
  .'filter data'.'Athlete is'.'Michael Phelps'.then
```

[5]Links removed to preserve anonymity.
[6]**npr.org/sections/thetorch/2016/08/14/489832779/**
[7]Available at: http://gallery.thegamma.net/86/

```
    .'group data'.'by Athlete'.'sum Gold'.then
    .'get series'.'with key Athlete'.'and value Gold'

charts.bar(data.append(phelps))
    .setColors(["#aec7e8","#aec7e8","#1f77b4"])
```

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 group. 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.

5.2 The Most Frequent Doctor Who Villains

In the second case study, we list Doctor Who villains by the number of episodes in which they appear.[8] This is interesting as it combines the graph database provider for fetching the data with the tabular data provider for summarization:

```
1  let topEnemies = drWho.Character.Doctor
2    .'ENEMY OF'.'[any]'.'APPEARED IN'.'[any]'.explore
3    .'group data'.'by Character name'
4      .'count distinct Episode name'.then
5    .'sort data'.'by Episode name descending'.then
6    .paging.take(8).'get series'
7      .'with key Character name'.'and value Episode name'
8
9  chart.bar(topEnemies)
```

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 5.

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 prviews support the user while doing so.

6 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.

[8] Available at: http://gallery.thegamma.net/87/

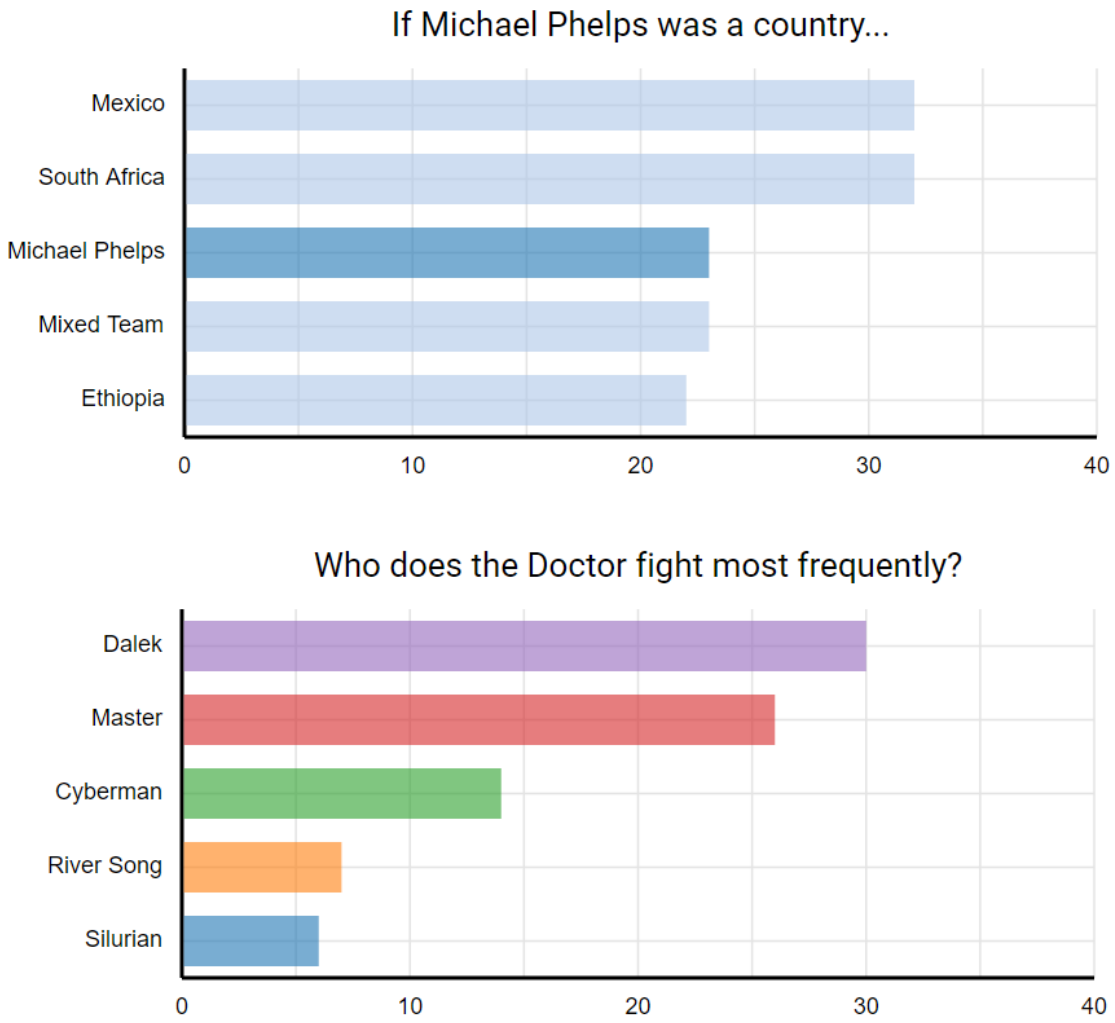


Fig. 5. Charts produced by two The Gamma case studies.

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 percolation, 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.

	Task	*Kind*	*Done*	*Notes*
#1	expenditure	cube	50	Obtained one of two data series
#2	expenditure	cube	100	Explored furhter data series
#3	expenditure	cube	100	Explored further data series
#4	expenditure	cube	75	With hint to use another member
#5	expenditure	cube	100	Explored further data series
#6	worldbank	cube	75	With general syntax hint
#7	worldbank	cube	100	Completed very quickly
#8	worldbank	cube	100	Extra time to find data
#9	lords	table	75	Struggled with composition
#10	lords	table	100	Completed very quickly
#11	lords	table	75	With a hint to avoid operations
#12	olympics	table	75	With a hint to avoid operations
#13	olympics	table	75	Hints about 'then' and operations

Table 1. Overview of the work completed by individual participants. The marks denote: 75 = required some guidance, 50 = partially completed.

6.1 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.
- *Lodrs.* 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.* Participnts 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.

6.2 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 reassurring."*

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 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 diagree 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 wher 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.

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.

6.3 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.

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 [38, 45, 62]. 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:

```
expneses.'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]."*

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.

6.4 Study 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.

7 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.

7.1 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 [27].

- *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.

7.2 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 [8].

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 currated online database such as Enigma Public[9]. Providing access to open government data repositories such as **`data.gov.uk`** is more appealing, but very challenging due to their unstructured nature.

7.3 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 [66]. 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.

[9] https://docs.enigma.com/public

8 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.

REFERENCES

[1] Azza Abouzied, Joseph M. Hellerstein, and Avi Silberschatz. 2012. DataPlay: interactive tweaking and example-driven correction of graphical database queries. In *The 25th Annual ACM Symposium on User Interface Software and Technology, UIST '12, Cambridge, MA, USA, October 7-10, 2012*, Rob Miller, Hrvoje Benko, and Celine Latulipe (Eds.). ACM, 207–218. https://doi.org/10.1145/2380116.2380144

[2] Eytan Adar. 2006. GUESS: a language and interface for graph exploration. In *Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, Rebecca E. Grinter, Tom Rodden, Paul M. Aoki, Edward Cutrell, Robin Jeffries, and Gary M. Olson (Eds.). ACM, 791–800. https://doi.org/10.1145/1124772.1124889

[3] Olivier Bau and Wendy E. Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology, Monterey, CA, USA, October 19-22, 2008*, Steve B. Cousins and Michel Beaudouin-Lafon (Eds.). ACM, 37–46. https://doi.org/10.1145/1449715.1449724

[4] David Blood. 2018. *Recycling is broken – notebooks.* https://github.com/ft-interactive/recycling-is-broken-notebooks

[5] Ivan Bretan, Robert Nilsson, and Kent Saxin Hammarstrom. 1994. V: a visual query language for a multimodal environment. In *Conference on Human Factors in Computing Systems, CHI 1994, Boston, Massachusetts, USA, April 24-28, 1994, Conference Companion*, Catherine Plaisant (Ed.). ACM, 145–147. https://doi.org/10.1145/259963.260174

[6] Marcel Bruch, Martin Monperrus, and Mira Mezini. 2009. Learning from examples to improve code completion systems. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*, Hans van Vliet and Valérie Issarny (Eds.). ACM, 213–222. https://doi.org/10.1145/1595696.1595728

[7] 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.

[8] James Cheney, Stephen Chong, Nate Foster, Margo I. Seltzer, and Stijn Vansummeren. 2009. Provenance: a future history. In *Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA*, Shail Arora and Gary T. Leavens (Eds.). ACM, 957–964. https://doi.org/10.1145/1639950.1640064

[9] 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. https://doi.org/10.1145/3242587.3242600

[10] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: Interactive Analytics Through Pen and Touch. *Proceedings of the VLDB Endownment* 8, 12 (Aug. 2015), 2024–2027. https://doi.org/10.14778/2824032.2824127

[11] William Davies. 2017. How statistics lost their power - and why we should fear what comes next. The Guardian. Retrieved March 6, 2020 from https://www.theguardian.com/politics/2017/jan/19/crisis-of-statistics-big-data-democracy.

[12] Mark Derthick, John Kolojejchick, and Steven F. Roth. 1997. An Interactive Visual Query Environment for Exploring Data. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST 1997, Banff, Alberta, Canada, October 14-17, 1997*, George G. Robertson and Chris Schmandt (Eds.). ACM, 189–198. https://doi.org/10.1145/263407.263545

[13] Jonathan Edwards. 2005. Subtext: uncovering the simplicity of programming. *ACM SIGPLAN Notices* 40, 10 (2005), 505–518. https://doi.org/10.1145/1103845.1094851

[14] Jonathan Edwards. 2018. *Direct Programming.* https://vimeo.com/274771188

[15] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S. Bernstein. 2018. Iris: A Conversational Agent for Complex Tasks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). ACM, 473. https://doi.org/10.1145/3173574.3174047

[16] Bob Franklin. 2012. THE FUTURE OF JOURNALISM. *Journalism Studies* 13, 5-6 (2012), 663–681. https://doi.org/10.1080/1461670X.2012.712301

[17] Chris Granger. 2012. *LightTable: A new IDE concept.* http://www.chris-granger.com/2012/04/12/light-table-a-new-ide-concept/

[18] 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.

[19] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.

[20] Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 65–74.

[21] Andrew Head, Fred Hohman, Titus Barik, Steven Mark Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 270. https://doi.org/10.1145/3290605.3300500

[22] Jeffrey Heer, Joseph M Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation. In *CIDR*.

[23] Joseph M. Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston, Vijayshankar Raman, Tali Roth, and Peter J. Haas. 1999. Interactive data analysis: the Control project. *Computer* 32, 8 (8 1999), 51–59. https://doi.org/10.1109/2.781635

[24] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST 2019, New Orleans, LA, USA, October 20-23, 2019*, François Guimbretière, Michael Bernstein, and Katharina Reinecke (Eds.). ACM, 281–292. https://doi.org/10.1145/3332165.3347925

[25] Bahareh R. Heravi, Martin Chorley, and Glyn Mottershead (Eds.). 2018. *Proceedings of the 2nd European Data and Computational Journalism Conference*. University College Dublin.

[26] Edwin L Hutchins, James D Hollan, and Donald A Norman. 1985. Direct manipulation interfaces. *Human–Computer Interaction* 1, 4 (1985), 311–338.

[27] Dan R. Olsen Jr. 2007. Evaluating user interface systems research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Newport, Rhode Island, USA, October 7-10, 2007*, Chia Shen, Robert J. K. Jacob, and Ravin Balakrishnan (Eds.). ACM, 251–258. https://doi.org/10.1145/1294211.1294256

[28] Eunice Jun, Maureen Daum, Jared Roesch, Sarah Chasins, Emery Berger, René Just, and Katharina Reinecke. 2019. Tea: A High-level Language and Runtime System for Automating Statistical Analysis. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST 2019, New Orleans, LA, USA, October 20-23, 2019*, François Guimbretière, Michael Bernstein, and Katharina Reinecke (Eds.). ACM, 591–603. https://doi.org/10.1145/3332165.3347940

[29] 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* (Monterey, California, USA) *(ICSE '87)*. IEEE Computer Society Press, Washington, DC, USA, 180–188.

[30] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*, Desney Tan, Geraldine Fitzpatrick, Carl Gutwin, Bo Begole, and Wendy Kellogg (Eds.). ACM, 3363–3372. https://doi.org/10.1145/1978942.1979444

[31] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*, Gloria Mark, Susan Fussell, Cliff Lampe, Monica Schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor (Eds.). Association for Computing Machinery, New York, NY, USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

[32] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Austin Henley, Peter Rogers, and Anita Sarma (Eds.). IEEE, 25–29. https://doi.org/10.1109/VLHCC.2017.8103446

[33] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows. In *20th International Conference on Electronic Publishing*, Fernando Loizides and Birgit Schmidt (Eds.). 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[34] Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2018. The Road to Live Programming: Insights from the Practice. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*, Andrea Zisman and Sven Apel (Eds.). ACM, New York, NY, USA, 1090–1101. https://doi.org/10.1145/3180155.3180200

[35] Haeyoun Park Larry Buchanan and Adam Pearce. 2017. You Draw It: What Got Better or Worse During Obama's Presidency. New York Times. Retrieved March 3, 2020 from https://www.nytimes.com/interactive/2017/01/15/us/politics/you-draw-obama-legacy.html.

[36] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 542–553.

[37] Henry Lieberman. 2001. *Your wish is my command: Programming by example*. Morgan Kaufmann.

[38] Eyal Lotem and Yair Chuchem. 2018. *Lamdu Project*. https://github.com/lamdu/lamdu

[39] 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.

[40] Klaus Meier. 2009. Transparency in Journalism. Credibility and trustworthiness in the digital future. In *Actas II Congreso The Future of Journalism*.

[41] Microsoft Corporation. 2020. *Microsoft Power BI*. https://powerbi.microsoft.com/en-us/

[42] Brad A. Myers, A. J. Ko, and Margaret M. Burnett. 2006. Invited research overview: end-user programming. In *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, Gary M. Olson and Robin Jeffries (Eds.). ACM, 75–80. https://doi.org/10.1145/1125451.1125472

[43] Bonnie A Nardi. 1993. *A small matter of programming: perspectives on end user computing*. MIT press.

[44] Bastian Obermayer and Frederik Obermaier. 2016. *The Panama Papers: Breaking the story of how the rich and powerful hide their money*. Oneworld Publications.

[45] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live Functional Programming with Typed Holes. *PACMPL* 3, POPL (2019). https://doi.org/10.1145/3291622

[46] 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.

[47] Tomas Petricek, James Geddes, and Charles A. Sutton. 2018. Wrattler: Reproducible, live and polyglot notebooks. In *10th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2018, London, UK, July 11-12, 2018.*, Melanie Herschel (Ed.).

[48] 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* (Santa Barbara, CA, USA) *(PLDI '16)*. ACM, 477–490. https://doi.org/10.1145/2908080.2908115

[49] Tomas Petricek, Bahareh R. Heravi, Jennifer A. Stark, and et al. 2017. *Proceedings of the European Data and Computational Journalism Conference*. University College Dublin.

[50] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.

[51] Veselin Raychev, Martin T. Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 419–428. https://doi.org/10.1145/2594291.2594321

[52] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2019. Exploratory and Live, Programming and Coding. *The Art, Science, and Engineering of Programming* 3, 1 (2019). https://doi.org/10.22152/programming-journal.org/2019/3/1

[53] Xin Rong, Shiyan Yan, Stephen Oney, Mira Dontcheva, and Eytan Adar. 2016. CodeMend: Assisting Interactive Programming with Bimodal Embedding. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*, Jun Rekimoto, Takeo Igarashi, Jacob O. Wobbrock, and Daniel Avrahami (Eds.). ACM, 247–258. https://doi.org/10.1145/2984511.2984544

[54] 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.

[55] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An interactive visualization design environment. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 351–360.

[56] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[57] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative interaction design for data visualization. In *The 27th Annual ACM Symposium on User Interface Software and Technology, UIST '14, Honolulu, HI, USA, October 5-8, 2014*, Hrvoje Benko, Mira Dontcheva, and Daniel Wigdor (Eds.). ACM, 669–678. https://doi.org/10.1145/2642918.2647360

[58] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*, Jun Rekimoto, Takeo Igarashi, Jacob O. Wobbrock, and Daniel Avrahami (Eds.). ACM, 365–377. https://doi.org/10.1145/2984511.2984588

[59] Ben Shneiderman, Christopher Williamson, and Christopher Ahlberg. 1992. Dynamic Queries: Database Searching by Direct Manipulation. In *Conference on Human Factors in Computing Systems, CHI 1992, Monterey, CA, USA, May 3-7, 1992, Proceedings*, Penny Bauersfeld, John Bennett, and Gene Lynch (Eds.). ACM, 669–670. https://doi.org/10.1145/142750.143082

[60] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia: AI-assisted Code Completion System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2727–2735. https://doi.org/10.1145/3292500.3330699

[61] 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* (Rome, Italy) *(DDFP '13)*. ACM, 1–4. https://doi.org/10.1145/2429376.2429378

[62] Gerd Szwillus and Lisa Neal. 1996. *Structure-based editors and environments*. Academic Press, Inc.

[63] Bret Victor. 2012. *Inventing on Principle*. http://worrydream.com/InventingOnPrinciple

[64] Bret Victor. 2012. *Learnable programming: Designing a programming system for understanding programs*. http://worrydream.com/LearnableProgramming

[65] David J. Ward, Alan F. Blackwell, and David J. C. MacKay. 2000. Dasher - a data entry interface using continuous gestures and language models. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, UIST 2000, San Diego, California, USA, November 6-8, 2000*, Mark S. Ackerman and W. Keith Edwards (Eds.). ACM, 129–137. https://doi.org/10.1145/354401.354427

[66] Richard Wesley, Matthew Eldridge, and Pawel T. Terlecki. 2011. An Analytic Data Engine for Visualization in Tableau. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*, Timos Sellis, Renée Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). ACM, 1185–1194. https://doi.org/10.1145/1989323.1989449

[67] Xiong Zhang and Philip J. Guo. 2017. DS.js: Turn Any Webpage into an Example-Centric Live Programming Environment for Learning Data Science. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017, Quebec City, QC, Canada, October 22 - 25, 2017*, Krzysztof Gajos, Jennifer Mankoff, and Chris Harrison (Eds.). ACM, 691–702. https://doi.org/10.1145/3126594.3126663