The Gamma: Data Exploration through Iterative Prompting

ANONYMOUS AUTHOR(S)

Data exploration tools based on code have many desirable characteristics. They can easily access a wide range of different data sources, result in reproducible scripts and encourage users to reuse and modify existing data analyses. Unfortunately, most programming tools require expert coding skills. Can we make data exploration based on code accessible to non-experts? We present The Gamma, a novel text-based data exploration environment. The Gamma is based on a single interaction principle and using it results in transparent and reproducible scripts. This lowers the barrier to entry and allows 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. Our work shows that we may not need to shy away from code in order to build accessible, reproducible and transparent tools that will allow a broad audience to benefit from the rise of open data.

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

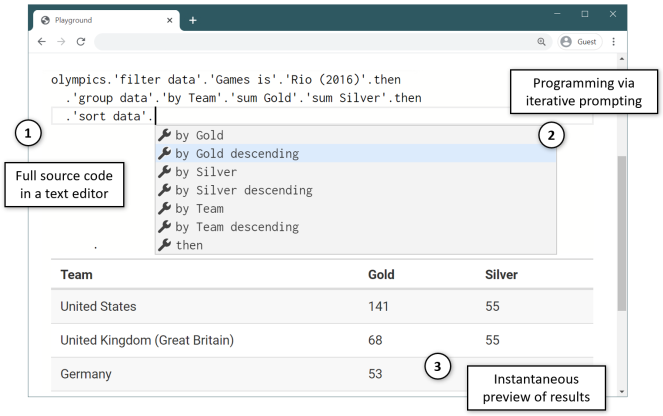
ACM Reference Format:
Anonymous Author(s). 2020. The Gamma: Data Exploration through Iterative Prompting. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 [9]. 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 [15]. The design of such data exploration tool poses a unique mix of challenges. 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 contribute 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 main contribution of our work is the *iterative prompting* interaction principle, which makes it possible to construct all valid data exploration scripts 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 first, but still produce transparent and reproducible 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.


Playground
Guest
1
Full source code in a text editor
2
Programming via iterative prompting
3
Instantaneous preview of results

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

```
by Gold
by Gold descending
by Silver
by Silver descending
by Team
by Team descending
then
```

Team	Gold	Silver
United States	141	55
United Kingdom (Great Britain)	68	55
Germany	53	

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 mechanism 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 programmatic data exploration simple enough for non-experts, but scraping and cleaning of messy data or building custom data visualizations is currently outside of the scope of our work. Exposing those using iterative prompting remains an interesting future challenge.

Paper Overview. 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[1]:

- We introduce the iterative prompting principle in The Gamma and show how it can be used for querying of distinct data sources including data tables, graph databases and data cubes (Section 3).
- We reflect how our design lowers barriers to entry (Section 6.1), supports learning without experts (Section 6.2) and offers a complete and correct program construction method (Section 6.3).
- We evaluate the system through a number of case studies (Section 4) and a user study (Section 5), which confirms that non-programmers can use The Gamma to construct non-trivial data queries.

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 [24], we do not view it as a programmer assistance tool. Instead, we turn it into an interaction mechanism through which non-experts can create entire programs. We build on recent research on information-rich programming [53] and aim to make those advances available to non-programmers [35, 37], in the context of data exploration as done, for example, by journalists [15]. Our work features a novel combination of characteristics in that our iterative prompting interaction principle (i) is centered around editing and understanding of code, (ii) reduces conceptual complexity to a single basic kind of interaction, yet (iii) it is correct and complete in that it can be used to construct all meaningful queries for a variety of data sources.

[1]This paper is X pages, without citations, to be commensurate with the size of the contribution.

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 [41, 53], 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 [39]. 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 make recommendations when users begin interacting with data. Those based on machine learning-based code completion for domain specific languages [17, 18] differ in that they do not guarantee completeness, i.e. it is unclear whether the user can create all possible scripts. Approaches based on natural language are effective [45, 50], but hide the underlying structure and do not help the user understand it. Conversational agents [13] share similar characteristics, except that the construction process is iterative.

Code completion based on machine learning or statistical methods [5, 43] also exists for general-purpose programming languages used by data scientists such as Python [52], providing assistance to expert programmers. Finally, DS.js [59] is interesting in that it enables querying of data on the web. It uses JavaScript, but with rich contextual code completion.

Notebooks and Business Intelligence Tools. Notebooks such as Jupyter [28], 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 also be integrated with a multi-language notebook system [40]. Spreadsheets, business intelligence tools [33, 58] and other visual data analytics tools [8, 19] do not involve programming, but require mastering a complex GUI. In contrast, The Gamma is based on a single kind of interaction, through which all available operations can be completed. Several systems [25, 42, 47] record interactions with the GUI as a script that can be 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 [56] introduced design principles that inspired many to build live programming systems [14, 29, 44] that give immediate feedback to help programmers understand how code relates to output and exploratory systems [26, 27] 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 [32, 38, 54]. In Subtext [11, 12] 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 offering high-level abstractions, e.g. for interactive news articles [7], statistical analyses [23] or interactive data visualization [48, 49]. The Gamma exposes a number of data sources through high-level abstractions that support iterative prompting, but support for tasks other than querying remains furture work.

Programming without Writing Code. There are two main approaches to programming where the user does not write code. In programming by example [31], the user gives examples of desired results. This has been used, e.g. for specifying data transformations in spreadsheets and data extraction [16, 30]. In direct manipulation [21], a program is specified by directly interacting with the output. This has been used in the visual domain [20], but also for data querying [4, 51]. The VQE language [10] 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].

Gestures 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 [57], our system provides a way of navigating through a complete space of options, while on-screen feedforward [3] allows efficient selection in gesture-based interfaces. Those provide compelling alternatives to auto-completion menus, although the efficiency of input methods is typically not an issue in programming.

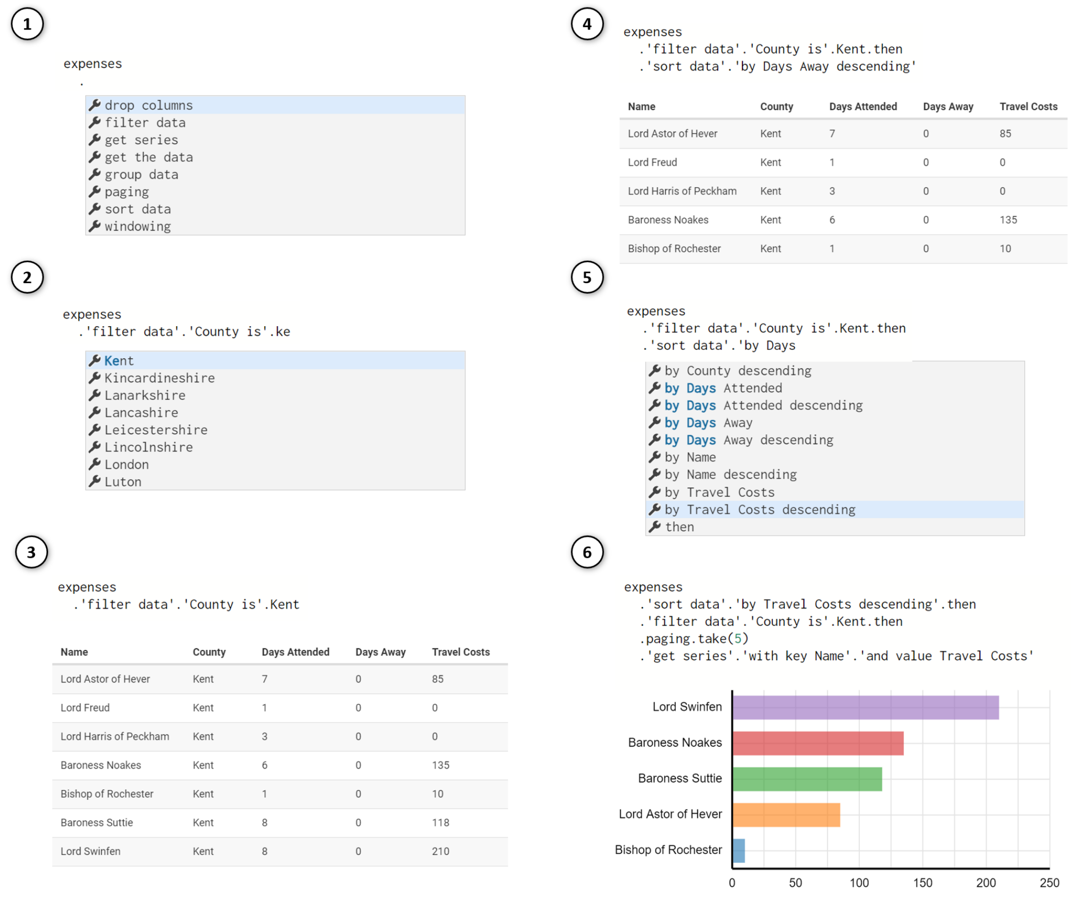
3 OVERVIEW

The Gamma aims to make text-based data exploration easy enough for non-experts. The aim is motivated, in part, by the desirable properties of text-based data exploration tools such as transparency, reproducibility and learnability and, in part, by an aim to explore an unexplored point in the design space of data exploration tools. Although various efforts make text-based programming easier, most systems that target non-experts shy away from code.

The Gamma is a text-based data exploration environment that allows non-experts explore data using iterative prompting – by repeatedly selecting an item from an auto-complete list. The study presented in Section 5 confirms that the kind of data exploration shown in the next section can, indeed, be successfully done by non-experts.

3.1 Querying Travel Expenses

The walkthough in Figure 2 shows a typical task completed using The Gamma. A data analyst from Kent is exploring travel expense claims by members of the House of Lords published by the UK government [55]. The following shows a subset of the data in the CSV format:



1

```
expenses
  .
```

- drop columns
- filter data
- get series
- get the data
- group data
- paging
- sort data
- windowing

2

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

- Kent
- Kincardineshire
- Lanarkshire
- Lancashire
- Leicestershire
- Lincolnshire
- London
- Luton

3

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

Name	County	Days Attended	Days Away	Travel Costs
Lord Astor of Hever	Kent	7	0	85
Lord Freud	Kent	1	0	0
Lord Harris of Peckham	Kent	3	0	0
Baroness Noakes	Kent	6	0	135
Bishop of Rochester	Kent	1	0	10
Baroness Suttie	Kent	8	0	118
Lord Swinfen	Kent	8	0	210

4

```
expenses
  .'filter data'.'County is'.Kent.then
  .'sort data'.'by Days Away descending'
```

Name	County	Days Attended	Days Away	Travel Costs
Lord Astor of Hever	Kent	7	0	85
Lord Freud	Kent	1	0	0
Lord Harris of Peckham	Kent	3	0	0
Baroness Noakes	Kent	6	0	135
Bishop of Rochester	Kent	1	0	10

5

```
expenses
  .'filter data'.'County is'.Kent.then
  .'sort data'.'by Days
```

- by County descending
- by Days Attended
- by Days Attended descending
- by Days Away
- by Days Away descending
- by Name
- by Name descending
- by Travel Costs
- by Travel Costs descending
- then

6

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

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

```
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 analyst imports the file through a web interface, the environment is initialised 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):

(1) The analyst types '.' (dot) to trigger auto-completion on `expenses`. The type provider offers a list of operations that the analyst can perform. To find House of Lords members from Kent, the analyst chooses `filter data`.
(2) The analyst is offered a list of columns based on the schema of the tabular data and choses `County is`. She is then offered a list of counties in the data set and types `ke` to search for Kent and she selects `Kent`.
(3) The Gamma evaluates the code on-the-fly and shows a preview of results. The analyst now sees a table with House of Lords members from Kent. She wants to see if there are any members who missed any House sessions.
(4) The analyst finishes specifying the (possibly compound) sorting key by choosing `then` and is offered the same list of querying operations as in the first step. She selects `sort data` followed by `by Days Away descending`.
(5) The analyst sees that there are no reported "days away" and decides to compare travel costs. She hits the backspace key a number of times, is offered the list of keys again and selects `by Travel Costs descending`.
(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.

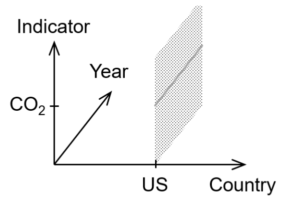
The constructed code is not unlike an SQL query, except that the whole script is constructed using iterative prompting, by repeatedly selecting one of the offered members. Those represent both operations, such as `sort by` and arguments, such as `Kent`. The only exception is when the analyst needs to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

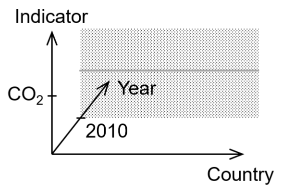
3.2 The Gamma Programming Environment

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)`. Names with non-alphanumerical characters are escaped using quotes.

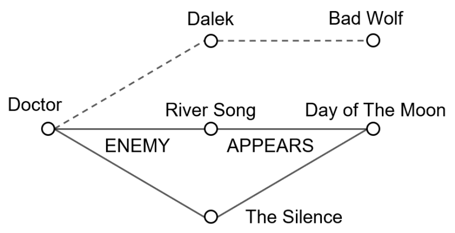
The Gamma uses a type system to infer what members are available at a given point in a chain. Each expression has a type with a list of members that, in turn, have their own types. The types are not built-in, but are generated by type providers for individual data sources. The programming environment for The Gamma is based on the Monaco editor [34]. When the user types '.' the editor triggers auto-completion and retrieves a list of available members based on the type information. The Gamma evaluates scripts on-the-fly and shows a preview as illustrated in Figure 1.

There is a handful of situations where The Gamma does not yet fully support the iterative prompting principle. First, it allows operations with parameters such as `take(5)`. This is currently needed when writing a query that skips or takes the first N elements from a table. Second, The Gamma allows the user to declare (immutable) variables using `let`. This is not needed for basic data exploration, but allows advanced users to better structure more complex code.





(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.

3.3 Type Providers for Data Querying

The Gamma can be extended to support any data source by implementing a *type provider*, which defines a domain specific language for exploring data of a particular kind. A type provider generates object types with members (such as `paging` or `Kent`) that are accessed via iterative prompting. We describe type provider for exploring data cubes (inspired by Syme et al. [53]), tabular data (based on theory developed by Petricek [39]), and graph databases.

Data Cube Type Provider. Our first type provider allows users to explore data cubes, which are multi-dimensional arrays of values. For example, the World Bank collects a range of indicators about many countries each year while the UK government expenditure records spending for different government services, over time, with different adjustments:

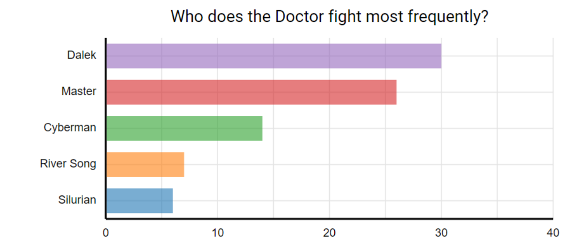
```
worldbank.byCountry.'United States'.'Climate Change'.'CO2 emissions (kt)'
expenditure.byService.Defence.inTermsOf.GDP
```

The dimensions of the `worldbank` cube are countries, years and indicators, whereas the dimensions of `expenditure` are government services, years and value type (adjusted, nominal, per GDP). Figure 3a how the provider allows users to slice the data cube. Choosing `byCountry.'United States'`, restricts the cube to a plane and selecting `'CO2 emissions (kt)'` then gives a 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 is used to select UK government spending on defence in terms of GDP.

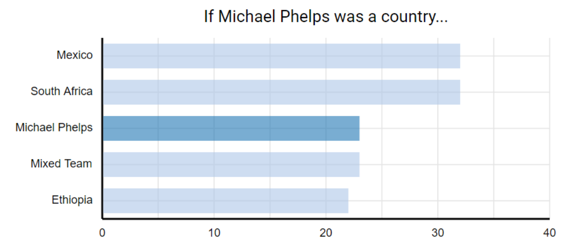
Tabular Data Type Provider. Our second type provider allows users to construct queries to explore data in tabular formats. Unlike the data cube provider, the provider for tabular data does not just allow selecting a subset of the data, but it can be used to construct SQL-like query. Consider the example 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 queries a table 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. Section 4 shows that this is sufficient to construct a range of practical queries.



(a) Exploring Dr Who graph database by composing type providers



(b) Exploring Olympic medallists using tabular data type provider

Fig. 4. Charts produced by two case studies of using The Gamma.

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. 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 4, the returned table can be further queried using the tabular data type provider.

4 CASE STUDIES

The Gamma aims to simplify programmatic data exploration while keeping enough expressive power to allow users to create interesting data explorations. We assess the simplicity of The Gamma in Section 5. Here, we consider two case studies that evaluate expressivity and show what can be achieved using the simple iterative prompting principle[2]. We used The Gamma for larger projects that analyse the UK government expenditure, present activities of a research institute, analyse Olympic medals and explore information about the Doctor Who series[3].

The Most Frequent Doctor Who Villains. Our first case study uses a graph database with data from the Dr Who series. It lists Dr Who villains by the number of episodes in which they appear. This case study is interesting as it combines the graph database provider for fetching the data with the tabular data provider for summarization:

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

Line 1 use the graph provider to find all paths linking the Doctor 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 selecting

[2]Available (non-anonymously) at: http://gallery.thegamma.net/86/ and http://gallery.thegamma.net/87/, respectively.
[3]The analyses are available (non-anonymously) at http://gallery.thegamma.net, http://turing.thegamma.net and http://rio2016.thegamma.net

`explore`. For each character (the villain) we count the number of distinct episodes. The result is shown in Figure 4a. Despite performing a 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.

If Michael Phelps were a Country. Michael Phelps has won so many medals that media compared the number to countries [36], often using a chart that shows a country league table including Michael Phelps as an additional data point. We reproduce the chart, shown in Figure 4b, using the tabular data type provider:

```
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
  .'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 data analysis is done in three commands. The first counts gold medals by countries and uses paging to fetch 4 countries with suitable number of medals. In the second, we use 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. The example illustrates a case when more advanced language features are necessary. The data exploration itself has been completed via iterative prompting, but producing the final chart currently requires some manual programming.

5 USER STUDY

Our objective is to develop a text-based data exploration tool that can be used by non-programmers. Using the characterization by Olsen [22], data exploration environments are complex systems that do not yield to simple controlled experimentation. We discuss the limits of evaluating The Gamma in Section 7.2 and do not attempt to make make quantitative comparison e.g. with SQL or Tabelau [58]. Consequently, our goals are more modest.

5.1 Study Design

We evaluate the extent to which non-programmers can use The Gamma to conduct text-based data exploration (*RQ1*). To test this, we gave volunteers one of four data exploration tasks and assessed whether they were able to complete the task, as well as how much assistance, if any, they needed. Some aspects of the study aim to shed light on three questions concerning learnability of The Gamma, in particular whether knowledge can be trasferred between different data sources (*RQ2a*), whether users can learn from code samples without a detailed guidance (*RQ2b*) and to what extent they form a correct mental model of a more complex query language used in the tabular data source (*RQ2c*).

We performed a between-subjects study to evaluate the first experience of using The Gamma. We recruited 13 participants (5 male, 8 female) from a business team of a research lab working in non-technical roles (project management, partnerships, communications) including one former journalist. Only one participant (#12) had prior programming experience. We split participants into 4 groups. We gave participants a brief overview of The Gamma (with content depending on the task) and then asked participants to complete a task. The participants worked on the tasks for 30 minutes, after which we conducted a 30 minute semi-structured group interview. We let participants work independently, but offered guidance if they got stuck and asked for help. The four tasks were:

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

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

- *Expenditure.* Participants were given a demo using *worldbank*. They were asked to use the *expenditure* data source to compare the UK government spending on "Public order and safety" and "Defence" in terms of GDP.
- *Lords.* Participants were given a demo using *worldbank*. They were asked to use the *lords* data source (a table with House of Lords members expenses) to find a members representing London with the highest travel costs.
- *Worldbank.* Participants were given a minimal demo of iterative prompting 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, involving grouping and aggregation, using the same data source.

All of the tasks aim to answer the primary question RQ1 for one of the supported data sources. In addition, the tasks *expenditure* and *lords* study RQ2a because they use different data source in introduction and taks; *worldbank* study RQ2b in that it comes with only brief introduction and *olumpics* studies RQ2c using our most complex data source.

5.2 Study Results

Table 1 summarizes the work done by the study participants. Most notably, all participants were able to complete, at least partially, a non-trivial data exploration task and only half of them required further guidance. 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. The experience suggests a number of possible design improvements for The Gamma, which are discussed below.

Can Non-programmers Explore Data with The Gamma? Three facts allow us to answer RQ1 in the affirmative. First, 9 out of 11 participants agree or strongly agree that they "found the system easy to use". Second, participants spent 10–25 minutes (average 17 minutes) working with The Gamma and 12 out of 13 completed the task; 6 required assistance, but 3 of those faced one issue (discussed later) that could be addressed in the introduction. Third, a number of participants shared positive comments in the interviews. For example, participant #3 noted taht *"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."* Participant #2 noted 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."*

How Users Learn The Gamma? The nature of RQ2 makes it a challenging question to study and finding conclusive answers may require further research. There is some evidence that knowledge can be transferred between different data sources (RQ2a). Two of the tasks (*expenditure* and *lords*) used one data source in the introduction and another for the task. Participants were able to complete those, although *lords* has been more challenging as it involves a complex data source. 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."*.

There is also some evidence that users can learn just from code samples (RQ2b). In the *worldbank* task, participants were given only a minimal demo of how to invoke iterative prompting together with print-out of 2 code samples. All three participants were able to complete a related task using the same data source. When discussing suitable educational materials for The Gamma, participant #7 also confirmed that having code is sufficient when they noted that *"a video would just be this [i.e. a code sample] anyway"*. This supports our hypothesis that, once a user understands the iterative prompting principle, they can learn how to use any specific data source just from code samples.

How Users Understand Complex Query Languages? Although iterative prompting itself is simple, specific data sources may expose richer structure. To ascertain their understanding of the tabular type provider (RQ2c), we asked participants who worked with tabular data (*lords* or *olympics*) about their understanding of the `then` member. This is a regular member, but it has a special meaning in the domain specific language. For example, when specifying a list of aggregations, the user can repeatedly select further aggregations. Selecting `then` completes the operation and allows the user to choose 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. Participant #12 then correctly concluded that it *"allows us to chain together the operations"* of the query; after a hint, participant #13 reflected that *"if I knew this from the start, it would [have been easier]."* This summarizes an important fact. Iterative prompting allows the users to start exploring new data sources, but the structures exposed by more complex data sources have their own further design principles that the users need to understand.

Making Complex Things Possible May Hurt. The Gamma 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 operations violate the iterative prompting principle as one cannot type '.' after `'Year is greater than'`. This suggests that we should either avoid such operations, or hide them under an "advanced operations" tab as a caution to novice users.

Benefits and Drawbacks of Text. The Gamma is based on text to aid transparency. Text implies 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 [32, 38, 54]. Most participants had no difficulty navigating around source code, making edits or deleting code fragments, which is arguably harder in an unfamiliar structured editor.

On the one hand, 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. On the other hand, 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.

6 DESIGN REFLECTIONS

The design of The Gamma has been motivated by a curiosity as to whether iterative prompting can make text-based programming with data accessible to non-experts. In this section, we theoretically assess the resulting design and position it in the context of a possible application in the context of data journalism.

6.1 Lowering Barriers to Entry

Data exploration has a certain irreducible essential complexity. To make it accessible to users who cannot dedicate much time to learning a tool prior to using it, this complexity needs to be carefully stratified. The Gamma uses a two-level structure. At the first level, the user needs to learn iterative prompting to be able to start exploring data. At the second level, users will need to learn domain specific languages defined by individual type providers.

Iterative Prompting Principle. Iterative prompting is a suitable first level principle, because it is easy to trigger. In conventional programming languages, auto-complete assistance is only available once basic code structure is written. In contrast, user of In The Gamma only needs to choose the initial data source. Iterative prompting is also easier to use than e.g. a command line or a REPL (read-eval-print-loop), because it follows the *recognition over recall* usability heuristic. The users are not required to recall and type a command. They merely need to select one from a list of options.

Stratifying Data Exploration Complexity. At the first level, any data source is accessed through the same mechanism. At the second level, each data source defines its own domain specific language, consisting of the primitives that are offered to the user in auto-complete prompts. The domain specific languages are embedded in The Gamma language – they define merely the available members and cannot extend the core language syntax. Although the complexity of individual languages differs, the users can always start exploring and learning new languages using the familiar first level iterative prompting principle. An important question, which we study in Section 5, is whether the expertise gained with one data source be transferred to working with another data source.

6.2 Learning without Experts

A typical user of The Gamma will not dedicate significant time to learning it in advance and they will not have access to experts. Most learning thus needs to happen from examples. When analysing how Excel users learn, Sarkar [46] points out that users learn new features when the usage of a feature is apparent in a spreadsheet. For example, users can learn different functions in formulas, because those are visible in the cell. Learning how to use a wizard for creating charts is not possible because the operation leaves no full trace in the spreadsheet. 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.

In The Gamma, each step of iterative prompting 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. 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. We assess the viability of this way of learning in the study in Section 5.

6.3 Correctness and Completeness

An important characteristic of our design is that, barring a few exceptions discussed below, the iterative prompting mechanism is both correct and complete with respect to possible data exploration scripts. This means that (i) auto-complete lists offered by iterative prompting contain only options that are valid in a given context and that (ii) any script that can be written in The Gamma can be constructed via iterative prompting.

Correctness. The Gamma uses an object-based type system for error-checking and for generating auto-complete lists. A well-typed script can always be executed. When using iterative prompting, a selected option, which is a valid object member, is added to the end of a script, leading to a well-typed script. This distinguishes our system from auto-completion based on machine learning or dictionary-based methods, which may offer members not valid in a given context. Advanced users of The Gamma, modifying scripts as text, can still violate correctness – for example, if the user uses auto-complete to replace a member in a middle of a member access chain with a member of another type.

Completeness. Auto-completion lists offered via iterative prompting contain all available members and so the user can construct all possible scripts. The range of such scripts is determined by type providers, which support limited set of options (e.g. group aggregations in the tabular data type provider). Two exceptions to completeness in our current design is specifying numerical parameters as in `take(5)` and advanced language features such as let binding. We believe that alleviating the need for those poses an interesting further research challenge.

7 DISCUSSION

We examine an unexplored point in the design space of tools for data exploration. The Gamma is a text-based programming environment, but targets non-programmers such as data journalists. We conclude with a more general discussion of evaluation of the system and the possible applicability of The Gamma in data journalism.

7.1 Evaluating Exploratory Research

The research presented 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 investigation focused on the core iterative prompting principle, but some our case studies also required using 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 other popular systems such as Tableau [58]. 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.

7.2 Evaluating Complex Systems

Data exploration environments are complex systems that do not yield to simple controlled experimentation. Olsen [22] proposes criteria for judging whether a system advances the state of the art. A number of those apply to The Gamma:

- *Importance.* Data journalism can make factual claims backed by data more commonplace and enable wider audience to engage with such claims. As such, we contribute towards solving an important societal issue.
- *Expressive leverage.* Iterative prompting stratifies the complexity of data exploration such that different data sources are accessed through the same unified iterative prompting interaction principle.

- *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.* The Gamma can be used to query data from a wide range of data sources including tabular data, data cubes and graph databases. The range of possible tasks is illustrated by case studies presented earlier.

7.3 Applications to Data Journalism

Although The Gamma targets a broad audience of non-programmers, some of our work has been particularly motivated by the use of data in journalism. The Gamma is aligned with recent challenges faced by journalists in that it can help build trust through transparency as well as provide meaningful ways of reader engagement.

Our study shows that non-experts with background similar to journalists 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, a former journalist who participated in our study (#13) 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 answer suggests that iterative prompting, does indeed, lower the barrier to entry. Although it does not fully eliminate complexity involved in data querying, it provides a way of stratifying it. Iterative prompting makes it easy to get started with data exploration, addressing the initial "nervousness about code". By making the source code of data analyses visible, The Gamma then enables further learning through percolation.

7.4 Further 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, but the system does not currently show where such data comes from. For some tabular data sources, the source is a CSV file published, e.g. by the government. In this case, we can easily show the source URL. However, other type providers may pre-process data. Displaying data source in such cases would require more sophisticated provenance tracking [6].

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 Public[4]. Providing access to open government data repositories such as http://data.gov and http://data.gov.uk is more appealing, but challenging due to their unstructured nature.

8 CONCLUSIONS

Exploring data in a programming environment that makes the full source code available increases transparency, reproducibility and empowers users to ask critical questions about the data analysis. But can we make those features accessible to non-programmers? In this paper, we presented The Gamma, a simple data exploration environment for non-programmers that answers this question in the affirmative.

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

The Gamma is based on a single interaction principle, *iterative prompting*. It can be used to complete a range of data exploration tasks using tabular data, data cubes and graph databases. The design lowers the barrier to entry for programmatic data exploration and makes it easy to learn the system independently through examples and by experimentation. We implemented The Gamma, make it available as open source and conducted a user study, which lets us conclude that The Gamma can be used by non-programmers to construct non-trivial data exploration scripts.

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*. ACM, 207–218.

[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*. 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*. ACM, 37–46. https://doi.org/10.1145/1449715.1449724

[4] 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 '94*, Catherine Plaisant (Ed.). ACM, 145–147. https://doi.org/10.1145/259963.260174

[5] 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 International Symposium on Foundations of Software Engineering*. ACM.

[6] 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 '09*. ACM, 957–964.

[7] 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 '18*. ACM, 977–989. https://doi.org/10.1145/3242587.3242600

[8] 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

[9] 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.

[10] 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 '97*. ACM, 189–198. https://doi.org/10.1145/263407.263545

[11] Jonathan Edwards. 2005. Subtext: uncovering the simplicity of programming. *ACM SIGPLAN Notices* 40, 10 (2005), 505–518.

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

[13] 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*. ACM, 473.

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

[15] Jonathan Gray, Lucy Chambers, and Liliana Bounegru. 2012. *The data journalism handbook: how journalists can use data to improve the news*. O'Reilly.

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

[17] 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.

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

[19] 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

[20] 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 '19*. ACM, 281–292. https://doi.org/10.1145/3332165.3347925

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

[22] Dan R. Olsen Jr. 2007. Evaluating user interface systems research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, UIST '07*. ACM, 251–258. https://doi.org/10.1145/1294211.1294256

[23] 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 UIST Symposium 2019*. ACM, 591–603.

[24] 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.

[25] 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*. ACM, 3363–3372.

[26] 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). ACM, 1265–1276.

[27] 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

[28] 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

[29] 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)*. ACM, New York, NY, USA, 1090–1101.

[30] 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.

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

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

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

[34] Microsoft Corporation. 2021. *Monaco Editor.* https://microsoft.github.io/monaco-editor/

[35] 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 '06*. ACM, 75–80. https://doi.org/10.1145/1125451.1125472

[36] Greg Myre. 2021. *If Michael Phelps Were A Country, Where Would His Gold Medal Tally Rank?* https://www.npr.org/sections/thetorch/2016/08/14/489832779/

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

[38] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live Functional Programming with Typed Holes. *PACMPL* 3, POPL (2019).

[39] Tomas Petricek. 2017. Data exploration through dot-driven development. In *31st European Conference on Object-Oriented Programming.*

[40] 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.).

[41] 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

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

[43] 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*. ACM, 419–428. https://doi.org/10.1145/2594291.2594321

[44] 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

[45] 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 '16*. ACM, 247–258.

[46] 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.

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

[48] 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.

[49] 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*. ACM, 669–678. https://doi.org/10.1145/2642918.2647360

[50] 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 '16*. ACM, 365–377. https://doi.org/10.1145/2984511.2984588

[51] Ben Shneiderman, Christopher Williamson, and Christopher Ahlberg. 1992. Dynamic Queries: Database Searching by Direct Manipulation. In *Conference on Human Factors in Computing Systems, CHI '92*. ACM, 669–670. https://doi.org/10.1145/142750.143082

[52] 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 '19*. ACM, 2727–2735. https://doi.org/10.1145/3292500.3330699

[53] 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*. ACM, 1–4. https://doi.org/10.1145/2429376.2429378

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

[55] UK Parliment. 2021. *Members' allowances and expenses.* https://www.parliament.uk/mps-lords-and-offices/members-allowances/house-of-lords/holallowances/

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

[57] 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 '00*. ACM, 129–137.

[58] 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*. ACM, 1185–1194. https://doi.org/10.1145/1989323.1989449

[59] 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 '17*. ACM, 691–702.