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, 18 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 [12]. 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 [19]. 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 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 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

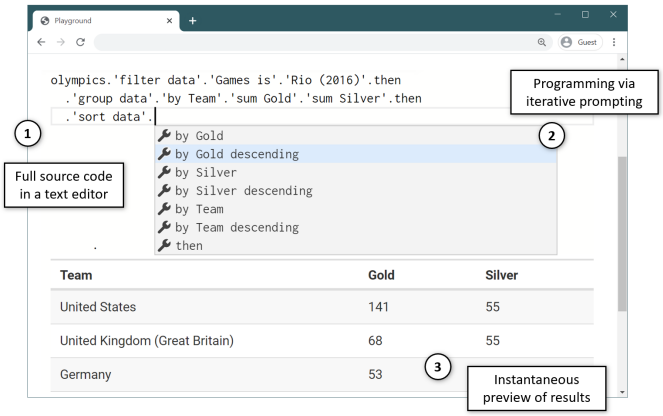


Fig. 1. Obtaining teams with the greatest number of gold medals from Rio 2016 Olympics with a reproducible The Gamma script (1), contextual iterative prompting mechainsm offering ways of sorting the data (2) and an instant preview of results (3).

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 reflect how our design addresses challenges faced by journalists (Section 4.1) and analyse the design theoretically. We argue that our design lowers barriers to entry (Section 4.2), supports learning without experts (Section 4.3) and offers a complete and correct program construction method (Section 4.4).
- We evaluate the system through a number of case studies (Section 5) and a user study (Section 6). 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 [30], 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 [43, 44], in the context of data exploration as done, most notably, by journalists [19]. 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 [49, 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 [47]. 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 [21, 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 [16] 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 [66] 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 [34], 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 [48].

Spreadsheets and business intelligence tools [42, 65] 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 [31, 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] introduced design principles that inspired many to build live programming systems [18, 35, 52] that give immediate feedback to help programmers understand how code relates to output and exploratory systems [32, 33] 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 [39, 46, 62]. In Subtext [14, 15] 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 [29] 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 [38], the user gives examples of desired results. This has been used, e.g. for specifying data transformations in spreadsheets and data extraction [20, 37]. In direct manipulation [27], 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 [13] 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 [64], 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 6 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).

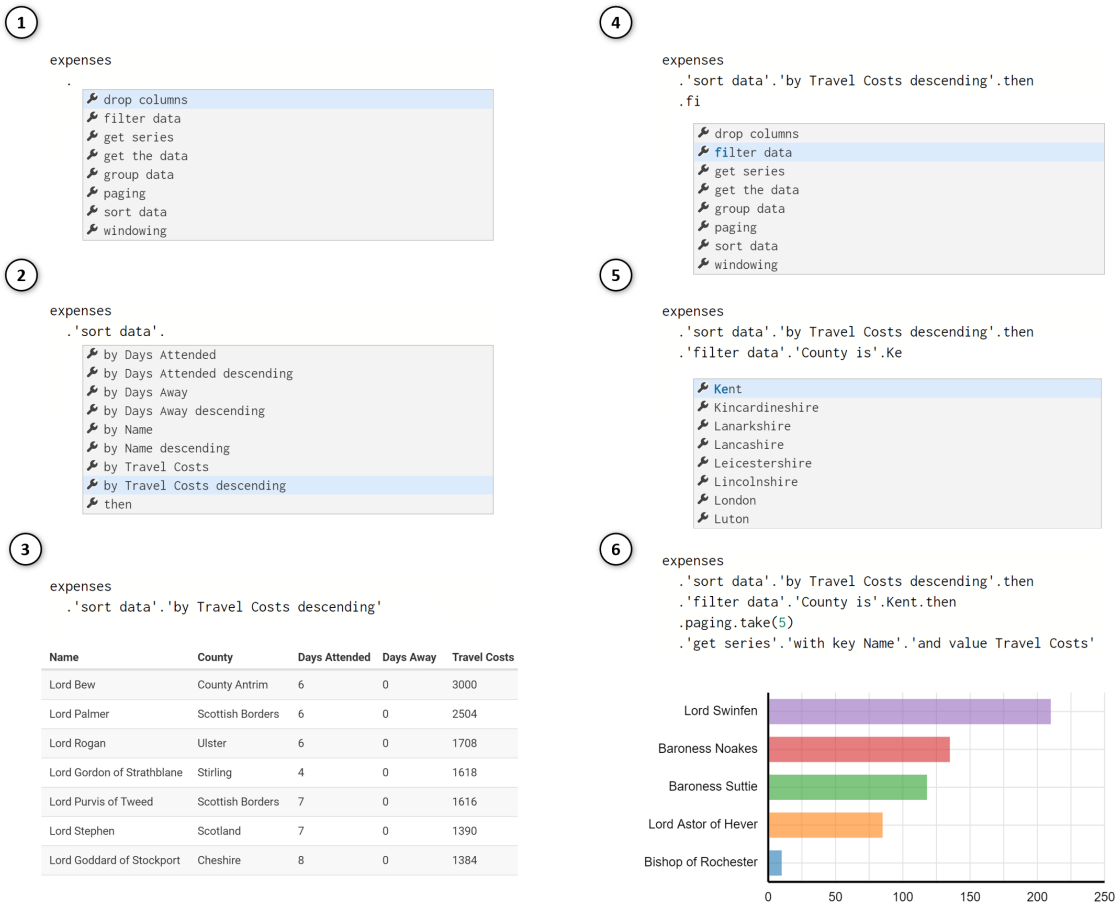


Fig. 2. Constructing a script that charts the top 5 members of the House of Lords for Kent, based on their travel costs.

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 [47]), 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:

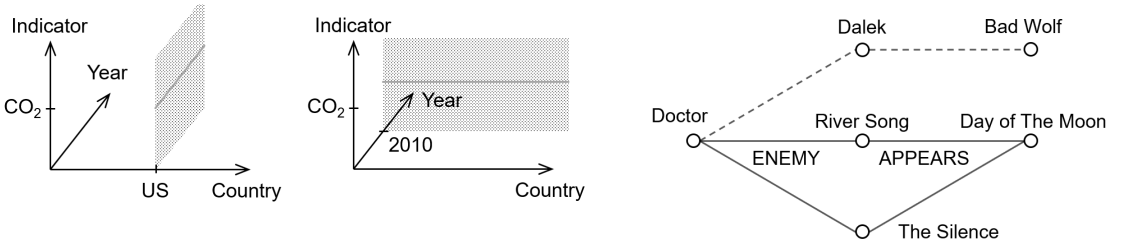
```
worldbank.byCountry.'United States'.'Climate Change'.'CO2 emissions (kt)'
```

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. A prominent example of tabular data used by journalists is, e.g. the Iraq War documents leak [11]. 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 5, 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 [45]. 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 5, the returned table can be further queried using the tabular data type provider.

4 REFLECTIONS ON DESIGN

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.

4.1 Data Journalism Perspective

Although our work targets non-expert data exploration in general, data journalism is an important specific domain with interesting design requirements. We reflect those on the basis of relevant literature, e.g. [19, 25, 26] and past experience of collaborating with journalists. Two challenges faced by journalists [17] are particularly relevant to The Gamma.

Trust Through Transparency. The first issue is trust in media. Many practitioners believe that transparency about the editorial process and information sources can serve as a proof of quality and trustworthiness [41]. This applies to data analyses too. For example, the Financial Times increasingly share source code of their analyses on GitHub, e.g. [4], but reproducing such analyses is difficult even for an expert. The Gamma has a potential to improve on the status quo in that a part of the analysis can be directly embedded in an article. The source code provides full account of what the data sources are and how are they used. The Gamma makes this code, to a greater extent, accessible to non-expert readers.

Encouraging Meaningful Engagement. The second challenge is developing relationship with readers. Journalists are looking for meaningful ways of engagement through reader comments, involvement of citizen journalists [7, 40] and the development of new interactive formats [36]. The Gamma can support this aim by providing a data exploration environment that can engage non-expert readers in a meaningful discussion. As argued below, non-experts can learn to use the system and can, for example, modify parameters or change filtering criteria to get a new perspective on a story.

4.2 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 6, is whether the expertise gained with one data source be transferred to working with another data source.

4.3 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 [54] 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.

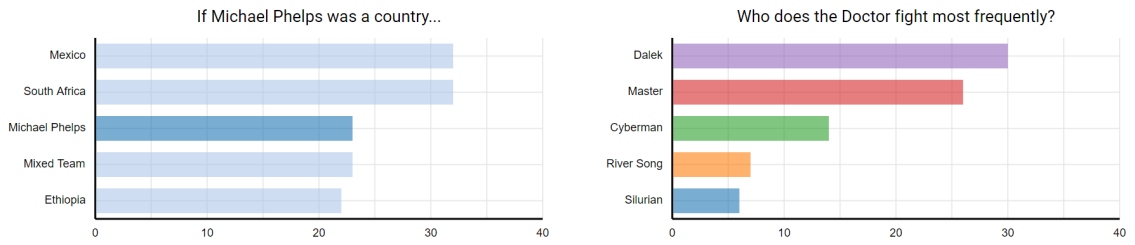
In The Gamma, each step of iterative prompting results in an identifier that is added to the source code. This means that a program constructed solely through iterative prompting keeps a full trace of how it was created. The resulting source code provides the user all information that they need to recreate 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 6.

4.4 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 contructed 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.



(a) Exploring Olympic medalists using tabular data type provider (b) Exploring Dr Who graph database by composing type providers

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

5 CASE STUDIES

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

If Michael Phelps were a Country. Michael Phelps has won so many medals that media compared the number to countries.[6] Media illustrated the story with a chart showing a country league table including Michael Phelps. We reproduce the chart, shown in Figure 4a, 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 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. The case study illustrates 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. In practice, this would likely be done with the help of an expert or by copying code from an example.

The Most Frequent Doctor Who Villains. In the second case study, use a graph database on the Dr Who series and list 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 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 4b. The example performs a fairly sophisticated data analysis that involves a graph database query followed by an SQL-like data aggregation. The code can be fully constructed using iterative prompting (with the exception of the numbers in paging). Another important feature is that instantaneous previews support the user during the construction, showing the intermediate result at each step of the construction.

[4]The analyses are available (non-anonymously) at http://gallery.thegamma.net, http://turing.thegamma.net and http://rio2016.thegamma.net
[5]Available (non-anonymously) at: http://gallery.thegamma.net/86/ and http://gallery.thegamma.net/87/, respectively.
[6]e.g. "If Michael Phelps Were A Country, Where Would His Gold Medal Tally Rank?" https://www.npr.org/sections/thetorch/2016/08/14/489832779/

6 USER STUDY

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

RQ1: Can non-programmers use The Gamma? Our first hypothesis is that non-programmers will be able to use The Gamma to explore data. To test this, we gave participants one of four simple data exploration tasks and assess whether they were able to complete the task, as well as how much assistance, if any, they needed.

RQ2: Can knowledge transfer between data sources? Our second hypothesis is that users familiar with the iterative prompting principle will be able to use an unfamiliar data source. We designed two of the four tasks to test this. Participants were shown a demo using one data source and then asked to complete a task using a different one.

RQ3: Can users learn from just code samples? Our third 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 were given a minimal explanation of The Gamma (showing how to initiate the iterative prompting process) together with an extensive code sample.

6.1 Study Design

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 four tasks were:

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

We let participants work independently, but offered guidance if they got stuck. 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 the understanding of the data source. Following the experiment, we conducted a short semi-structured group interview and later sent participants a follow-up questionnaire.

6.2 Study Results

Figure 5 shows the results of the study. 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. Table 5a 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 discuss possible design improvements based on the experience in Section 6.3.

	Task	*Kind*	*Done*	*Notes*
#1	expenditure	cube	◑	Obtained one data series
#2	expenditure	cube	●	Explored furhter data
#3	expenditure	cube	●	Explored further data
#4	expenditure	cube	◕	Correct member hint
#5	expenditure	cube	●	Explored further data
#6	worldbank	cube	◕	With general syntax hint
#7	worldbank	cube	●	Completed very quickly
#8	worldbank	cube	●	Extra time to find data
#9	lords	table	◕	Composition issues
#10	lords	table	●	Completed very quickly
#11	lords	table	◕	Hint to avoid operations
#12	olympics	table	◕	Hint to avoid operations
#13	olympics	table	◕	Two operations hints

(a) Work completed by individual participants; ● = completed, ◕ = required some guidance, ◑ = partially completed

Questionnaire item	*Avg*	*Sdv*
Considering your experience and possible use of The Gamma in the context of data journalism:		
I found the system easy to use.	4.00	0.89
Journalists will be able to use it to analyse data.	3.45	1.04
Readers will be able to examine the analyses.	3.73	0.79
Thinking about the usability of the system:		
Resulting programs are easy to understand.	3.73	0.79
Text editor for creating them was easy to use.	3.73	0.90
Finding items in the auto-complete was easy.	3.45	1.37
I would know how to explore a new data source		
If given a comprehensive video tutorial.	4.00	1.26
If given a number of code samples.	3.36	1.12
Without any further guidance.	2.45	1.13

(b) Follow-up questionnaire responses using a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree) over 11 subjects.

Fig. 5. Summary of the work completed by individual participants and responses to the follow-up questionnaire.

The follow-up questionnaire was sent to all participants a week after the experiment. It was completed all but two participants (#8 and #12). We asked general questions about the usability of the system together with questions about learning designed to answer RQ2 and RQ3. A summary of the answers is shown in Table 5b.

RQ1: 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. Participant #3 found the system simple, but also points out an issue about data provenance, which we revisit 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 explore. Participants were able to complete those tasks, although *lords* has been more challenging as it involves a more complex data source. In the interview, participant #2 also gives a positive answer:

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

Negative evidence is offered by the follow-up questionnaire. When asked whether they would know how to approach a task using a new data source, 6 out of 11 participants disagree or strongly disagree that they would know how to approach it without any further guidance. The nature of RQ2 makes it a more challenging question to study and so finding a conclusive answer arguably requires further research.

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 11 participants disagree or strongly disagree that they would know how to approach a task using an unfamiliar data source when given "a number of code samples".

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

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

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

6.3 Further Observations

In this section, we briefly discuss a number of observations about The Gamma design that emerged from the experiments and follow-up interviews, some of which suggest ways of improving the system.

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. They 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 [39, 46, 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.

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.

Understanding the 'then' Member. We asked participants who worked with tabular data about their understanding of the `then` member. This is a regular member (not a keyword), but it has a special meaning in the domain specific language. Consider the calculation of average travel costs and number of representatives per county in the House of Lords:

```
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 repeatedly add 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) 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. Following a hint, participant #13 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.

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 [65]. 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 [28] 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 anlyses 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 [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[7]. 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 anlysis. 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.

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.

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

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] 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 '94*, 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 International Symposium on Foundations of Software Engineering*. ACM.

[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 '09*. ACM, 957–964.

[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 '18*. 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] Nick Davies, Jonathan Steele, and David Leigh. 2010. *Iraq war logs.* https://www.theguardian.com/world/2010/oct/22/iraq-war-logs-military-leaks

[12] 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.

[13] 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

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

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

[16] 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.

[17] Bob Franklin. 2012. The Future of Journalism. *Journalism Studies* 13, 5-6 (2012), 663–681. https://doi.org/10.1080/1461670X.2012.712301

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

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

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

[21] 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.

[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 '19*. 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 EDCJC Conference.* University College Dublin.

[26] Bahareh R. Heravi and et al. 2017. *Proceedings of the European Data and Computational Journalism Conference.* University College Dublin.

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

[28] 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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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

[34] 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

[35] 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.

[36] 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.

[37] 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.

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

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

[40] 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.

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

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

[43] 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

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

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

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

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

[48] 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.).

[49] 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

[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*. 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 '16*. ACM, 247–258.

[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. 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*. 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 '16*. 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 '92*. 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 '19*. 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*. 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] 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.

[65] 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

[66] 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.