

The Gamma: Programmatic Data Exploration for Non-programmers

Tomas Petricek

University of Kent, Canterbury, UK &
Charles University, Prague, Czech Republic

Abstract—Data exploration tools based on code can access any data source, result in reproducible scripts and encourage users to verify, reuse and modify existing code. Unfortunately, they are hard to use and require expert coding skills. Can we make data exploration tools based on code accessible to non-experts?

We present The Gamma, a novel text-based data exploration environment that answers the question in the affirmative. The Gamma takes the idea of code completion to the limit. Users create transparent and reproducible scripts without writing code, by repeatedly choosing from offered code completions.

The Gamma is motivated by the needs of data journalists and shows that we may not need to shy away from code for building accessible, reproducible and transparent tools that allow a broad public to benefit from the rise of open data.

Index Terms—data exploration, data journalism

I. INTRODUCTION

Despite the advances on visual tooling, programmatic data exploration remains the choice of expert analysts. It is flexible, offers greater reusability and leads to transparent analyses. The design of a programmatic data exploration tool that would be accessible to data journalists poses a number of design challenges. First, the tool needs to have a low barrier to entry to support first-time users without training. Second, it needs to support multiple data sources in a uniform way to allow transfer of knowledge across domains. Finally, users need to be able to learn by looking at existing data analyses.

We present The Gamma, a text-based data exploration tool for non-experts that is based on a single, easy to understand interaction principle. It provides a uniform access to data tables, graph databases and data cubes and leads to transparent analyses that can be easily reproduced, encouraging learning and critical engagement with data.

The Gamma is based on *iterative prompting*, which turns code completion from a programmer assistance tool into a non-expert programming mechanism that allows users to construct all valid data exploration scripts just by repeatedly choosing an item from a list of offered options. The design favors *recognition over recall* and allows non-programmers to construct entire scripts without first learning to code. Yet, the result remains a transparent and reproducible script. A crucial feature is that iterative prompting only offers operations that are valid in a given context and that it offers all such operations; it is both correct and complete.

The Gamma focuses on tasks that a data journalist may want to complete (Figure 1). The user accesses data available in a

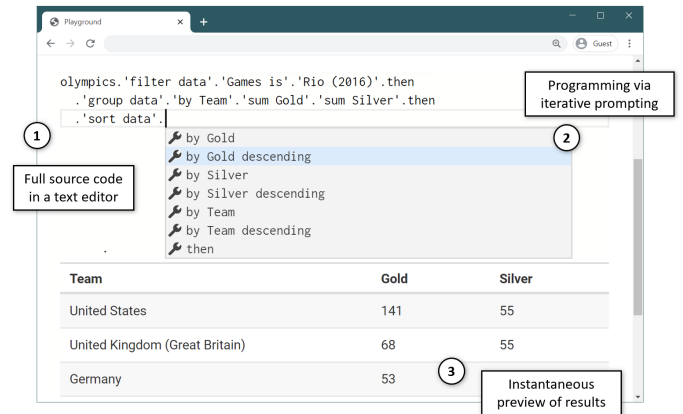


Fig. 1: Obtaining teams with the greatest number of gold medals from Rio 2016 Olympics: (1) Reproducible The Gamma script; (2) contextual iterative prompting offering ways of sorting the data; (3) an instant preview of results.

structured format. They make several experiments to find an interesting insight, e.g. by applying different aggregations or filters. They visualize the results using a table or a chart before publishing their analysis. The Gamma makes such programmatic data exploration simple enough for non-programmers. Scraping and cleaning of messy data or building custom data visualizations is outside of the scope of our work, but exposing such functionality using iterative prompting is an interesting and worthwhile future challenge.

In this paper, we describe and evaluate the design principles behind The Gamma project:

- We introduce the iterative prompting principle in The Gamma (Section III) and show how it can be used for querying of distinct data sources including data tables, graph databases and data cubes (Section IV).
- We illustrate the expressiveness of the system through a case study (Section V) and evaluate it through a user study (Section VI), confirming that non-programmers can use The Gamma to construct non-trivial data queries.
- We reflect how our design lowers barriers to entry, supports learning without experts and offers a complete and correct program construction method (Section VII).

The Gamma is available at <http://thegamma.net>, both as an open-source library and a hosted data exploration service.

II. RELATED WORK

We aim to make recent advances on information-rich programming [1] available to non-programmers [2], [3], in the context of data journalism [4]. Our work features a novel combination of characteristics in that our iterative prompting interaction principle is centered around code, but reduces the conceptual complexity of coding to a single basic kind of interaction.

Code Completion for Data Science: The Gamma utilizes type providers [1], [5], which integrate external data into a static type system. This enables auto-completion [6], which we turn into a tool for non-programmers. Similar systems based on machine learning and domain specific languages [7], [8] do not guarantee completeness, i.e. it is unclear whether the user can create all possible scripts. Approaches based on natural language are effective [9], [10], [11], but hide the underlying structure and do not help users understand the exact operations performed. Code completion based on machine learning [12], [13] also exists for general-purpose programming languages used by data scientists such as Python [14], but this focuses on providing assistance to expert programmers.

Notebooks and Business Intelligence Tools: Notebooks such as Jupyter [15] are widely used data exploration environments for programmers. The Gamma targets non-experts, but could be integrated with a multi-language notebook system [16]. Spreadsheets, business intelligence [17], [18] and other visual data analytics tools [19], [20] do not involve programming, but require mastering a complex GUI. In contrast, in The Gamma, all operations can be completed through a single kind of interaction. Several systems [21], [22], [23] record interactions with the GUI as a script that can be modified by the user. Unlike in The Gamma, the generated code does not guide the user in learning how to use the system.

Easier Programming Tools: Many systems aim to make programming easier. Victor [24] inspired work on live environments environments [25], [26], [27] that help programmers understand how code relates to output; exploratory systems [28], [29] assist with completing open-ended tasks; and system combining code with visualization also exists for graph querying [30]. The Gamma is live in that our editor gives an instant preview of the results. To avoid difficulties with editing code as text, some systems use structured editors [31], [32], [33], [34], [35]. Many systems simplify programming by offering high-level abstractions, e.g. for interactive news articles [36], statistical analyses [37], data visualization [38], [39]. The Gamma provides high-level abstractions for data querying, but supporting other tasks remains future work.

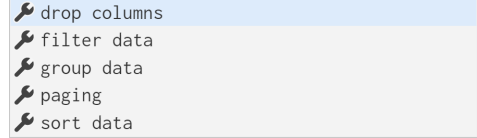
Programming without Writing Code: In programming by example [40], used for example in spreadsheets [41], [42], the user gives examples of desired results. In direct manipulation [43], a program is specified by interacting with the output. This has been used in the visual domain [44], but also for data querying [45], [46], [47]. Direct manipulation can also support data exploration by letting users partially edit queries, e.g. by changing quantifiers as in DataPlay [48].

III. OVERVIEW

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

We introduce The Gamma by walking through a typical data exploration task. A data journalist from Kent is exploring travel expense claims by members of the House of Lords published by the UK government [49]. After importing the CSV file through a web interface, the environment is initialized with code that refers to the imported data as expenses using the type provider for tabular data (Section IV). The journalist types ‘.’ (dot) to start exploring the data:

```
1 expenses.
```



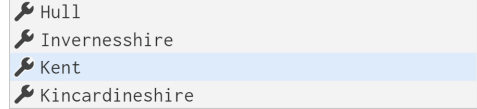
```

✎ drop columns
✎ filter data
✎ group data
✎ paging
✎ sort data

```

The type provider offers a list of operations that the journalist can perform. To find House of Lords members from Kent, the journalist chooses filter data. She is then offered a list of columns based on the schema of the CSV file and chooses County is. The completion lists counties in the data set:

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



```

✎ Hull
✎ Invernesshire
✎ Kent
✎ Kincardineshire

```

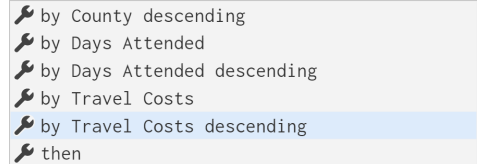
The journalist chooses Kent. The Gamma evaluates the code on-the-fly and shows a preview of results.

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

Name	County	Days Attended	Travel Costs
Lord Astor of Hever	Kent	7	85
Lord Freud	Kent	1	0
Lord Harris of Peckham	Kent	3	0

The journalist decides to compare travel costs. She finishes specifying the filtering condition by choosing then and is offered the same list of querying operations as in the first step. She selects sort data and is offered a list of sorting options:

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



```

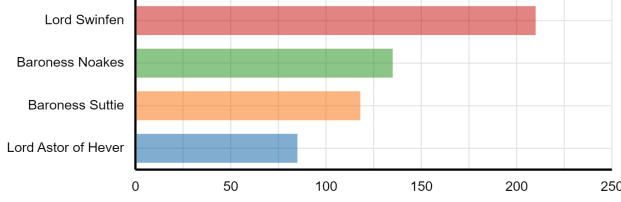
✎ by County descending
✎ by Days Attended
✎ by Days Attended descending
✎ by Travel Costs
✎ by Travel Costs descending
✎ then

```

The journalist chooses then and is, again, offered the list of querying operation. She uses paging to get the top 4 records, which requires typing 4 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:

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



The 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 4 to specify the number of items to take.

IV. SYSTEM DESCRIPTION

A program in The Gamma is a sequence of commands that can be either a variable declarations or an expression that evaluates to a value. An expression is a reference to a data source followed by a chain of member accesses. Each expression has a type that is used to generate options in auto-completion. A type defines 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 syntax and semantics of the language has been described elsewhere [50].

A new data source can be supported 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 outline type providers for exploring data cubes (inspired by Syme et al. [1]), tabular data (formalized elsewhere [52]), and graph databases.

Data Cube Provider: Data cubes 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:

```
1 worldbank.byCountry.'United States'.
2   'Climate Change'.CO2 emissions (kt)'
3
4 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 2a illustrates how the provider allows users to slice the data cube. Choosing byCountry.'United States', restricts the cube to a plane and '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.

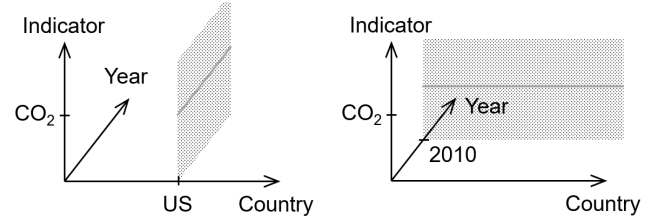
Graph Database Type Provider: Graph databases store nodes representing entities and relationships between them. The following example explores a database of Doctor Who characters and episodes. It retrieves all enemies of the Doctor that appear in the Day of the Moon episode:

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

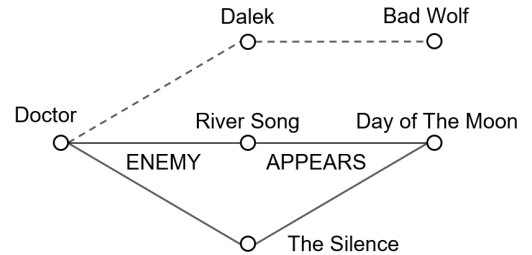
We start from the Doctor node and then follow two relationships. We use 'ENEMY OF'.'[any]' to follow links to all enemies of the Doctor and then specify 'APPEARED IN' to select only enemies that appear in a specific episode. The query is illustrated in Figure 2b. The members are generated from the data; 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, which can be further queried and visualized.

Tabular Data Provider: Unlike the graph and data cube providers, the type 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 of querying expense claims from Section III, which filters and then sorts the data.

When using the provider, the user specifies a sequence of operations. Members such as 'filter data' or 'sort data' determine the operation type. Those are followed by members that specify operation parameters. For example, when filtering data, we first select the column and then choose a desired value. Unlike SQL, the provider only allows users to choose from pre-defined filtering conditions, but this is sufficient for constructing a range of practical queries.



(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. 2: Type providers for exploring cube and graph data.

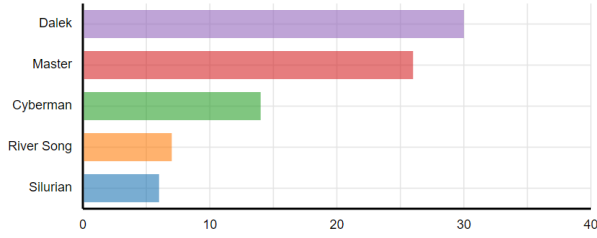


Fig. 3: Who does the Dr Who fight most frequently?

V. CASE STUDY

The Gamma aims to simplify programmatic data exploration while keeping enough expressive power to allow users to create interesting data explorations. To show what can be achieved by interactive prompting, we present a case study that explores a graph database with Dr Who series data.¹

The following constructs a chart (Figure 3) of top Dr Who villains by the number of episodes in which they appear. This case is interesting as it combines the graph database provider for fetching the data with the tabular data provider:

```

1 drWho.Character.Doctor.'ENEMY OF'. '[any]'
2   '.APPEARED IN'. '[any]'. explore
3   '.group data'. 'by Character name'
4   '.count distinct Episode name'. then
5   '.sort data'. 'by Episode name descending'. then
6   '.paging.take(8)'. 'get series'
7   '.with key Character name'
8   '.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 explore. For each character (the villain) we count the number of distinct episodes. The result is shown in Figure 3. 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.

VI. USER STUDY

Data exploration environments are complex systems that do not yield to simple controlled experimentation [54]. Rather than comparing our work with other tools, we evaluate whether The Gamma can be successfully used by non-programmers.

We performed a between-subjects study to assess whether non-programmers are able to complete a simple data exploration task using The Gamma. We recruited 13 participants (5 male, 8 female) from a business team of a research institute working in non-technical roles (project management, communications). Only one participant (#12) had prior programming experience. We split participants into 4 groups and asked each group to complete a different task. We gave participants a brief overview of The Gamma. The participants then worked for 30

minutes, after which we conducted a semi-structured group interview. We offered guidance if participants were unable to progress for more than 5 minutes. The four tasks were:

- *Expenditure*. Participants were shown the *worldbank* data cube and were asked to compare UK spending on ‘Public order and safety’ and ‘Defence’ using another data cube.
- *Lords*. Participants were shown *worldbank* and were asked to use the *expenses* data table provider to sort London House of Lords members by their travel costs.
- *Worldbank*. Participants were given a minimal iterative prompting demo and a code sample using *worldbank*. They were asked to solve another *worldbank* task.
- *Olympics*. Participants were given a demo using *olympics* that did not involve grouping. They were asked to solve a problem involving grouping and aggregation.

Our primary hypothesis was that non-programmers will be able to use The Gamma to explore data. This was tested by all four tasks for one of the supported data sources.

The tasks *expenditure* and *lords* further test if knowledge can be transferred between different data sources by using one sources in the introduction and another in the task; *worldbank* explores whether users can learn how to use a data source from just code samples; and *lords* lets us study to what extent participants form a correct mental model of the more complex query language used in the tabular data source.

Can non-programmers explore data with The Gamma? All participants were able to complete, at least partially, a non-trivial data exploration task and only half of them required further guidance. Participants spent 10–25 minutes (average 17) working with The Gamma and 12 out of 13 completed the task; 6 required assistance, but 3 of those faced the same issue related to operations taking arguments (discussed later).

A number of participants shared positive comments in the group interviews. Participant #3 noted that “*this is actually pretty simple to use,*” participant #2 said that The Gamma alleviated their unease about code: “*for somebody who does not do coding or programming, this does not feel that daunting.*” and participant #5 suggested that the system could be used as an educational tool for teaching critical thinking with data.

How users learn The Gamma? There is some evidence that knowledge can be transferred between different data sources. In *expenditure* and *lords*, participants were able to complete tasks after seeing a demo using another data source. Participant #2 “*found it quite easy to translate what you showed us in the demo to the new dataset.*”. However, the *lords* task has been more challenging as it involves a more complex data source.

There is also some evidence that, once a user understands iterative prompting, they can learn from just code samples. All three participants were able to complete the *worldbank* task, where they were given printed code samples, but no demo using any data source. When discussing suitable educational materials for The Gamma, participant #7 also confirmed that “*a video would just be this [i.e. a code sample] anyway*”.

¹See: <http://gallery.thegamma.net/87/>. We also used The Gamma for projects exploring the UK government expenditure, activities of a research institute and Olympic medal winners, available at <http://turing.thegamma.net> and <http://rio2016.thegamma.net>

How users understand complex query languages? The tabular type provider uses a member then to complete the specification of a current operation, for example when specifying a list of aggregation operations. 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. While iterative prompting allows users to start exploring new data sources, the structures exposed by more complex data sources have their own further design principles that the users need to understand.

What would make The Gamma easier to use? Three participants (#11, #12, #13) struggled to complete a task using the tabular data source, because they attempted to use operation that takes a numerical parameter and thus violates the iterative prompting principle. This could be avoided by removing such operations or by hiding them under an “advanced” tab.

The Gamma uses an ordinary text editor and most participants had no difficulty navigating around code, making edits or deleting fragments, which is harder in a structure editor. Some participants used the text editor effectively, e.g. leveraging copy-and-paste. However, two participants struggled with indentation and a syntax error in an unrelated command. This could likely be alleviated through better error reporting.

VII. DISCUSSION

As a text-based programming environment for non-programmers, The Gamma examines an unexplored point in the design space of tools for data exploration. It has been particularly motivated by the use of data in journalism. The Gamma has the potential to enable journalists to make factual claims backed by data more commonplace and enable wider audience to engage with such claims, satisfying the *importance* criteria [54] for advancing the state of the art. It also satisfies a number of design goals important in the data journalism context.

Learning without experts: Our design aims to make The Gamma suitable for users who cannot dedicate significant amount of time to learning it in advance and may not have access to experts, satisfying the *empowering new participants* criteria [54]. This is supported in two ways.

First, the iterative prompting principle makes it easy for users to start experimenting. The user needs to select an initial data source and then repeatedly choose an item from a list of choices. This is easier to use than a command line or a REPL (read-eval-print-loop) interface, 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.

Second, the resulting code serves as a trace of how the analysis was created. It provides the user with all information that they need to recreate the program, not just by copying it, but also by using iterative prompting. Such design has been called *design for percolation* [55] and it supports learnability. In Excel, studied by Sarkar [55], users learn new features when their usage is apparent in a spreadsheet, e.g. different functions

in formulas, but learning how to use a wizard for creating charts is hard because the operation does not leave a full trace in the spreadsheet.

Lowering barriers to entry: Data exploration has a certain irreducible essential complexity [56]. To make a system usable, this complexity needs to be carefully stratified. The Gamma uses a two level structure. The first level consists of the language itself with the iterative prompting mechanism. The second level consists of the individual members generated by a type provider. This can be seen as a *domain specific language*, embedded in The Gamma language. Although the complexity of individual domain specific languages differs, the user can always start exploring through iterative prompting, even when faced with an unfamiliar data source.

In tackling complexity, The Gamma satisfies two criteria proposed by Olsen [54]: *generality* in that it can be used uniformly with a wide range of data sources, and *expressive leverage* in that it factors out common aspects of different data queries into the core language (first level) and leaves the specifics of each data source to the second level.

Correctness and completeness: An important characteristic of our design is that the iterative prompting mechanism is both *correct* and *complete* with respect to possible data exploration scripts. The two properties are a consequence of the fact that a program is formed by a chain of operations and that the auto-completion leverages a static type system. When invoking iterative prompting at the end of a well-typed script, a selected option, which is a valid object member, is added to the end of the script, resulting in another well-typed script. This distinguishes our system from auto-completion based on machine learning, which may offer members not valid in a given context. Auto-completion lists offered via iterative prompting contain all available members and so the user can construct all possible scripts. Two exceptions to completeness in our current design are the let binding and specifying numerical parameters as in `take(5)`.

VIII. 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.

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.

ACKNOWLEDGMENTS

We thank to May Yong and Nour Boulahcen for their contributions to The Gamma type providers. The author is also grateful to Don Syme, James Geddes, Jonathan Edwards and Roly Perera for numerous discussions about data science tooling and type providers, as well as Luke Church for discussions about human-computer interaction and Clemens Klokmoose for numerous suggestions on framing of this paper. Anonymous reviewers of this and earlier versions of the paper also provided valuable feedback. This work was partly supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1 and by a Google Digital News Initiative grant.

REFERENCES

- [1] D. Syme, K. Battocchi, K. Takeda, D. Malayeri, and T. Petricek, “Themes in information-rich functional programming for internet-scale data sources,” in *Proceedings of Workshop on Data Driven Functional Programming*. ACM, 2013, pp. 1–4.
- [2] B. A. Myers, A. J. Ko, and M. M. Burnett, “Invited research overview: end-user programming,” in *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI '06*. ACM, 2006, pp. 75–80. [Online]. Available: <https://doi.org/10.1145/1125451.1125472>
- [3] B. A. Nardi, *A small matter of programming: perspectives on end user computing*. MIT press, 1993.
- [4] J. Gray, L. Chambers, and L. Bounegru, *The data journalism handbook: how journalists can use data to improve the news*. O'Reilly, 2012.
- [5] T. Petricek, G. Guerra, and D. Syme, “Types from data: Making structured data first-class citizens in F#,” in *Proceedings of Conference on Programming Language Design and Implementation*, ser. PLDI '16. ACM, 2016, pp. 477–490.
- [6] G. E. Kaiser and P. H. Feiler, “An architecture for intelligent assistance in software development,” in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87. Washington, DC, USA: IEEE Computer Society Press, 1987, p. 180–188.
- [7] J. Heer, J. M. Hellerstein, and S. Kandel, “Predictive interaction for data transformation,” in *CIDR*, 2015.
- [8] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer, “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*, 2011, pp. 65–74.
- [9] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang, “Eviza: A natural language interface for visual analysis,” in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST '16*. ACM, 2016, pp. 365–377. [Online]. Available: <https://doi.org/10.1145/2984511.2984588>
- [10] X. Rong, S. Yan, S. Oney, M. Dontcheva, and E. Adar, “Codemend: Assisting interactive programming with bimodal embedding,” in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST '16*. ACM, 2016, pp. 247–258.
- [11] E. Fast, B. Chen, J. Mendelsohn, J. Bassen, and M. S. Bernstein, “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, 2018, p. 473.
- [12] M. Bruch, M. Monperrus, and M. Mezini, “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, 2009.
- [13] V. Raychev, M. T. Vechev, and E. Yahav, “Code completion with statistical language models,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14*. ACM, 2014, pp. 419–428. [Online]. Available: <https://doi.org/10.1145/2594291.2594321>
- [14] A. Svyatkovskiy, Y. Zhao, S. Fu, and N. Sundaresan, “Pythia: AI-assisted code completion system,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19*. ACM, 2019, pp. 2727–2735. [Online]. Available: <https://doi.org/10.1145/3292500.3330699>
- [15] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, “Jupyter notebooks—a publishing format for reproducible computational workflows,” in *20th International Conference on Electronic Publishing*, F. Loizides and B. Schmidt, Eds., 2016, pp. 87–90.
- [16] T. Petricek, J. Geddes, and C. A. Sutton, “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*, M. Herschel, Ed., 2018.
- [17] R. Wesley, M. Eldridge, and P. T. Terlecki, “An analytic data engine for visualization in tableau,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. ACM, 2011, pp. 1185–1194.
- [18] Microsoft Corporation. (2020) Microsoft power bi. [Online]. Available: <https://powerbi.microsoft.com/en-us/>
- [19] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas, “Interactive data analysis: the control project,” *Computer*, vol. 32, no. 8, pp. 51–59, 8 1999.
- [20] A. Crotty, A. Galakatos, E. Zraggen, C. Binnig, and T. Kraska, “Vizdom: Interactive analytics through pen and touch,” *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 2024–2027, Aug. 2015.
- [21] V. Raman and J. M. Hellerstein, “Potter’s wheel: An interactive data cleaning system,” in *VLDB*, vol. 1, 2001, pp. 381–390.
- [22] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, “Wrangler: Interactive visual specification of data transformation scripts,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3363–3372.
- [23] A. Satyanarayan and J. Heer, “Lyra: An interactive visualization design environment,” in *Computer Graphics Forum*, vol. 33, no. 3, 2014, pp. 351–360.
- [24] B. Victor. (2012) Inventing on principle. [Online]. Available: <http://worrydream.com/InventingOnPrinciple>
- [25] P. Rein, S. Ramson, J. Lincke, R. Hirschfeld, and T. Pape, “Exploratory and live, programming and coding,” *The Art, Science, and Engineering of Programming*, vol. 3, no. 1, 2019.
- [26] J. Kubelka, R. Robbes, and A. Bergel, “The road to live programming: Insights from the practice,” in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 1090–1101.
- [27] C. Granger. (2012) Lighttable: A new IDE concept. [Online]. Available: <http://www.chris-granger.com/2012/04/12/light-table-a-new-ide-concept/>
- [28] M. B. Kery, A. Horvath, and B. Myers, “Variolite: Supporting exploratory programming by data scientists,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017, p. 1265–1276.
- [29] M. B. Kery and B. A. Myers, “Exploring exploratory programming,” in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, A. Henley, P. Rogers, and A. Sarma, Eds. IEEE, 2017, pp. 25–29.
- [30] E. Adar, “GUESS: a language and interface for graph exploration,” in *Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006*. ACM, 2006, pp. 791–800. [Online]. Available: <https://doi.org/10.1145/1124772.1124889>
- [31] G. Szwillus and L. Neal, *Structure-based editors and environments*. Academic Press, Inc., 1996.
- [32] C. Omar, I. Voysey, R. Chugh, and M. A. Hammer, “Live functional programming with typed holes,” *PACMPL*, vol. 3, no. POPL, 2019.
- [33] E. Lotem and Y. Chuchem. (2018) Lamdu project. [Online]. Available: <https://github.com/lamdu/lamdu>
- [34] J. Edwards, “Subtext: uncovering the simplicity of programming,” *ACM SIGPLAN Notices*, vol. 40, no. 10, pp. 505–518, 2005.
- [35] —. (2018, 6) Direct programming. [Online]. Available: <https://vimeo.com/274771188>
- [36] M. Conlen and J. Heer, “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, 2018, pp. 977–989. [Online]. Available: <https://doi.org/10.1145/3242587.3242600>
- [37] E. Jun, M. Daum, J. Roesch, S. Chasins, E. Berger, R. Just, and K. Reinecke, “Tea: A high-level language and runtime system for automating statistical analysis,” in *Proceedings of the 32nd Annual ACM UIST Symposium 2019*. ACM, 2019, pp. 591–603.

- [38] A. Satyanarayan, K. Wongsuphasawat, and J. Heer, “Declarative interaction design for data visualization,” in *The 27th Annual ACM Symposium on User Interface Software and Technology, UIST '14*. ACM, 2014, pp. 669–678. [Online]. Available: <https://doi.org/10.1145/2642918.2647360>
- [39] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, “Vega-lite: A grammar of interactive graphics,” *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 341–350, 2016.
- [40] H. Lieberman, *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.
- [41] S. Gulwani, W. R. Harris, and R. Singh, “Spreadsheet data manipulation using examples,” *Communications of the ACM*, vol. 55, no. 8, pp. 97–105, 2012.
- [42] V. Le and S. Gulwani, “Flashextract: a framework for data extraction by examples,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 542–553.
- [43] E. L. Hutchins, J. D. Hollan, and D. A. Norman, “Direct manipulation interfaces,” *Human–Computer Interaction*, vol. 1, no. 4, pp. 311–338, 1985.
- [44] B. Hempel, J. Lubin, and R. Chugh, “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, 2019, pp. 281–292. [Online]. Available: <https://doi.org/10.1145/3332165.3347925>
- [45] B. Shneiderman, C. Williamson, and C. Ahlberg, “Dynamic queries: Database searching by direct manipulation,” in *Conference on Human Factors in Computing Systems, CHI '92*. ACM, 1992, pp. 669–670. [Online]. Available: <https://doi.org/10.1145/142750.143082>
- [46] I. Bretan, R. Nilsson, and K. S. Hammarstrom, “V: a visual query language for a multimodal environment,” in *Conference on Human Factors in Computing Systems, CHI '94*, C. Plaisant, Ed. ACM, 1994, pp. 145–147. [Online]. Available: <https://doi.org/10.1145/259963.260174>
- [47] M. Derthick, J. Kolojechick, and S. F. Roth, “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, 1997, pp. 189–198. [Online]. Available: <https://doi.org/10.1145/263407.263545>
- [48] A. Abouzied, J. M. Hellerstein, and A. Silberschatz, “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, 2012, pp. 207–218.
- [49] UK Parliament. (2021) Members’ allowances and expenses. [Online]. Available: <https://www.parliament.uk/mps-lords-and-offices/members-allowances/house-of-lords/holallowances/>
- [50] T. Petricek, “Foundations of a live data exploration environment,” *Art Sci. Eng. Program.*, vol. 4, no. 3, p. 8, 2020. [Online]. Available: <https://doi.org/10.22152/programming-journal.org/2020/4/8>
- [51] Microsoft Corporation. (2021) Monaco editor. [Online]. Available: <https://microsoft.github.io/monaco-editor/>
- [52] T. Petricek, “Data exploration through dot-driven development,” in *31st European Conference on Object-Oriented Programming*, 2017.
- [53] G. Myre. (2021) If michael phelps were a country, where would his gold medal tally rank? [Online]. Available: <https://www.npr.org/sections/thetorch/2016/08/14/489832779/>
- [54] D. R. O. Jr., “Evaluating user interface systems research,” in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, UIST '07*. ACM, 2007, pp. 251–258. [Online]. Available: <https://doi.org/10.1145/1294211.1294256>
- [55] A. Sarkar and A. D. Gordon, “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)*, Sep. 2018, pp. 28–35.
- [56] J. Brooks, F.P., “No silver bullet essence and accidents of software engineering,” *Computer*, vol. 20, no. 4, pp. 10–19, april 1987.
- [57] J. Cheney, S. Chong, N. Foster, M. I. Seltzer, and S. Vansummeren, “Provenance: a future history,” in *Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA '09*. ACM, 2009, pp. 957–964.