Scratch Community Blocks: Supporting Children as Data Scientists

Sayamindu Dasgupta[*][†]
[*]MIT Media Lab
Cambridge, MA 02142
sayamindu@media.mit.edu

Benjamin Mako Hill[†]
[†]University of Washington
Seattle, WA, 98195
{makohill, sdg1}@uw.edu

ABSTRACT
In this paper, we present *Scratch Community Blocks*, a new system that enables children to programmatically access, analyze, and visualize data about their participation in *Scratch*, an online community for learning computer programming. At its core, our approach involves a shift in who analyzes data: from adult data scientists to young learners themselves. We first introduce the goals and design of the system and then demonstrate it by describing example projects that illustrate its functionality. Next, we show through a series of case studies how the system engages children in not only representing data and answering questions with data but also in self-reflection about their own learning and participation.

Author Keywords
data science; learning; computers and children; creativity support tools; social computing and social navigation; block-based programming

ACM Classification Keywords
K.3.2 Computers and Education: Computer and Information Science Education; H.5.2 Information Interfaces and Presentation (e.g., HCI): User Interfaces

INTRODUCTION
Widespread use of social media and digital learning platforms by children has led to the creation of massive sets of observational data that describe the ways that young people interact, socialize, and learn [8]. In hundreds of studies using data collected from a variety of platforms and contexts, researchers have answered a wide variety of research questions about youth, influencing educational policy and instructional methods [47]. Although very few common threads can be drawn across the large and diverse body of "data science" that studies uses of sociotechnical systems by children, one common trait is that the collecting, displaying, and visualizing of data is done by adult analysts, designers, and policymakers. In most cases, this group is also charged with making sense of and acting upon analyzed data. In this process, children are the object of analysis; their role is to generate data by using the system.

CHI 2017, May 06–11, 2017, Denver, CO, USA
ACM 978-1-4503-4655-9/17/05.
http://dx.doi.org/10.1145/3025453.3025847

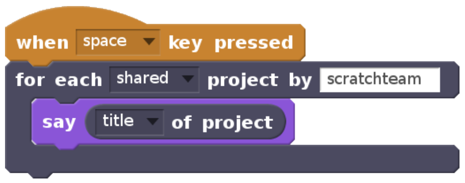


Figure 1: A Scratch script, consisting of blocks from the *Scratch Community Blocks* system. When the "space" key is pressed, this script iterates through all the shared Scratch projects by the user "scratchteam," and during each iteration, a graphical object on the screen says the title of the currently selected project through a visual speech bubble.

We imagine a different approach to data science and education in which young people use data to ask and answer their own questions. Our approach is inspired by constructionism [30], a theory of learning put forward by Seymour Papert and others where learners are understood to be driven by their own interests to *construct* knowledge instead of passively acquiring it from a teacher. Building on Piaget's theory of constructivism [33], constructionism theorizes the learner as an active builder of knowledge and adds the idea that knowledge building occurs "especially felicitously in a context where the learner is consciously engaged in constructing a public entity" [31].

In this paper, we present a system called *Scratch Community Blocks* that is designed to give the 15 million users of the *Scratch* online community the ability to programmatically analyze data from the community itself—an ability that has previously been the exclusive domain of data scientists and engineers. The system enables community members to create and share their own visualizations and analytics tools. We begin with a brief description of a number of related systems and of Scratch. Next, we describe the motivation behind and the design of *Scratch Community Blocks* and provide several short sample projects as illustrative examples. To demonstrate the range of possibilities introduced by the system, we describe a series of projects created by children using the system as part of a beta test. We discuss the technical implementation of our systems as well as several limitations of our design. We conclude by connecting back our work to some of the core aspirations of constructionist theory.

RELATED WORK

The design of *Scratch Community Blocks* is informed by a number of previous systems designed to support personal analytics, social media analytics, data science, and learning. A number of systems for data-mediated reflection and personal analytics fall into the genre of "quantified self" tools [29]. These include commercially available systems (e.g., systems that visualize data from wearable fitness trackers) as well as systems produced by peer production [6] (e.g., blood glucose level analysis [4]). Commercial systems are typically preprogrammed with limited customizability. Peer-produced tools are generally "hackable" but tend to be designed for experienced programmers. There is growing interest in engaging the quantified self movement in educational contexts [25]. Additionally, a number of tools exist to help end users analyze online social media systems and social networks. For example, there are widely used systems that visualize users' social connections using data from Facebook [48] and email archives [46, 21]. Email archives have also been used as a source of reflection with the MUSE system by Hangal et al. [19].

In learning technologies, the term "dashboards" is often used to describe systems that support the visualization and analysis of learning data. Verbert et al. [44] review 15 dashboards across a wide range of characteristics including target audience and type of data tracked. In some of these systems, data is used to inform educators about student progress with information visualization on who is progressing as expected and who is stuck (e.g., [35]). In some cases, dashboards are presented to learners so that they can reflect. For example, the dashboard provided by the Khan Academy platform gives learners a sense of how much time they have spent on a given course module [23]. Beyond dashboards, Rivera-Pelayo et al. [39] presented a framework that supports reflection in informal learning through mechanisms derived from the quantified self community. Within Scratch, *Jots* was an experimental system that supported Scratch users in creating brief updates or "jots" as they worked through their projects [40].

There is also growing interest in data science tools for youth. A comparatively early tool to support learning with computation and data was *Tinkerplots*, a visualization and modeling system that can be used for developing statistical reasoning skills [17]. *iSense* is a hardware toolkit and an associated web-based collaboration system that allows young learners to collect and visualize data [27]. In a more specialized context, Van Wart and Parikh created *Local Ground*, a system that enables youth to gain fluency with geographical information systems [43]. *BlockyTalky* [42] is a block-based visual language that enables young programmers to use sensing data from the physical world to build interactive programs. *DataSnap* is an extension to the block-based language *Snap*, similar in some respects to Scratch, which can fetch and analyze data from online sources [20]. *DataBasic.io* is a suite of data-literacy tools, designed to used by novices in a variety of contexts [7].

In addition to these systems, educators have frequently turned to more traditional systems to introduce data science to novices. These systems include spreadsheets such as *Microsoft Excel*, visualization tools such as *ManyEyes* [45], data management web-services such as *Google Fusion Tables* [18]), and mainstream programming languages (e.g., [13], [12]).

Compared to these systems and approaches, *Scratch Community Blocks* provides a unique combination of affordances; it opens up a significant amount of space for open-ended, user-driven exploration within data science, while ensuring a comparatively low barrier to entry. We know of no other system that is specifically designed for children to engage in interest-driven programmatic analysis and visualization of their own learning and social data. Like many previous systems, our work is informed by research in the learning sciences and statistics education research that has examined how children and youth establish relationships between data and context [5] and how they interpret and reason about complex information visualizations [24].

SCRATCH

Scratch Community Blocks is a new system built within the *Scratch* programming language [37]. Scratch is a visual, block-based programming language designed for children and youth aged 8–16. Programs in Scratch (called "projects") are constructed by dragging and dropping visual blocks together. Each visual block can be thought of as a programming primitive that determines the behavior of on-screen graphical objects called "sprites." For example, a script constructed out of Scratch programming blocks in Figure 1 will make a cat sprite sequentially say the title of all the Scratch projects shared by the "scratchteam" user in response to a key press.

Scratch is situated within an online community where members can share projects [28]. Users are encouraged to socialize around projects through commenting, bookmarking, showing appreciation, etc. Another affordance of the Scratch community is the ability to remix existing projects. Every project shared in the community is associated with a view source ("See Inside") button, and community members are encouraged to build on and extend their peers' work. Seen through the lens of constructionism, Scratch projects are instances of "public entities" that are both individually and collaboratively created and shared.

CHILDREN AS DATA SCIENTISTS

The central design goal of *Scratch Community Blocks* is the idea of *children as data scientists*. That said, it would be incorrect to describe the goal of our approach as merely shifting the ability to analyze data about youth activity online from adults to children. Just as the design of Logo by Papert and others [30, 1] was fundamentally shaped to support details of the constructionist theory of learning—rather than merely part of an attempt to turn children into programmers—the shift in the locus of data analysis from adults to children is only one visible feature of our approach as designers. Beyond this obvious shift lie two broad goals that parallel key features of successful constructionist learning toolkits in general. Those features are the ability to support learning through making in a social context and the ability to support self-reflection and learning about learning.

To achieve the first goal, our system aims to foster *new pathways to learning data science* by enabling data science learning experiences that are discovery-driven, built around building, personally meaningful, and engaging for participants. For the second goal, the system strives to enable learners to *reflect with data about their own behavior and thought processes*. The most effective constructionist systems can encourage young users to engage in self-reflection in ways that prompt them to "embark on an exploration about how they themselves think" [30]. Our goals represent our pathway, as designers, toward this central aspiration.

New pathways to learning data science

Although data science is cited as an increasingly important skill or literacy, there is no general consensus on what it constitutes. We adopt one common definition that uses the term to describe a set of practices at the intersection of substantive question asking, mathematical analysis, and computing skills [14, 22]. Critically, we treat programming as a core component of data science. In contrast to most existing tools aimed at introducing youth to data analysis where visualization is central, we focus on programming as the primary way to engage with data. In our vision, programming is important because it lets the learner ask questions or conduct explorations that we as designers may not have thought of. In their discussion of the design of construction kits for children, Resnick and Silverman state that "a little bit of programming goes a long way" in that it opens the possibilities of a vast potential design space [38].

Self-driven question setting also aligns well with the constructionist approach to learning that frames our approach. The core feature of any constructionist system is the ability to support learning that is personally relevant and meaningful. As a design criterion for toolkits in constructionism, this quality is referred to as "appropriability," [30] which is described as constituting three core principles: *continuity*, *power*, and *cultural resonance*. The continuity principle states that the topic being learned should be continuous with the prior personal knowledge of the learner. The power principle suggests that learners should be empowered to achieve new creative possibilities. The principle of cultural resonance seeks to ensure that the topic is seen as valuable in a larger social and cultural context.

Scratch Community Blocks is designed to satisfy these principles. First, it is situated in the context of the Scratch community, and information accessed through the system is data that the users of the system and their peers have created. For the system's users (participants in the Scratch community), this creates *continuity* between what users already know about the community and about programming with Scratch and the data and tools they have access through *Scratch Community Blocks*. Additionally, the system builds on the visual drag-and-drop programming paradigm that Scratch users are familiar with. The system is designed to be *powerful* in the sense that it provides programmatic access to data, allowing users to analyze and visualize information in ways that were unanticipated by the designers and are independent of the affordances provided by the Scratch website. Finally, given the significant amount of excitement around the emerging discipline of data science [34], it is not unfair to say that the topic is seen as valuable and relevant in present society. Within the Scratch community itself, there is significant enthusiasm and excitement about the possibility of doing data analysis using Scratch data. In both senses, the system can be described as *culturally resonant*.

Reflection on learning and social participation

Apart from constructionism, a number of theories of education highlight reflection as an important component of learning. Bruner [11], for example, theorizes that with reflection, learners "distance" themselves to reach a "higher ground" of abstraction, thereby gaining new perspectives on what has been learned. Boud, Keogh and Walker [9] propose a model of reflection in which emotion is a component and where behavior change is an important outcome of reflection. In this model, reflection requires individuals to "recapture their experience, think about it, mull it over and evaluate it."

Many models of reflection focus on relatively short temporal scales. For example, in Resnick's creative learning spiral model [36], reflection is seen as the final step in one iteration of a creative activity. Schön [41] differentiates between "reflection-on-action" and "reflection-in-action." In reflection-on-action, the process of reflection happens after the action has been performed, while in reflection-in-action, the reflective process occurs hand-in-hand with the activity. The *Scratch Community Blocks* system seeks to allow children to revisit and analyze data about themselves in ways that encourages reflection after the fact, but at both longer and short time scales.

DESIGN

The *Scratch Community Blocks* system is tightly integrated with the Scratch interface and community, extends the list of programming primitives in Scratch, and uses the same visual block-based drag-and-drop editing paradigm as the core Scratch language. In addition to using a block-base editing paradigm, we tried to make *Scratch Community Blocks* consistent with the rest of the Scratch programming environment by (i) using programming constructs already known to Scratch users (e.g., "accessor" blocks, which we describe below) and by (ii) avoiding a large number of blocks or block parameters, as Scratch has traditionally avoided having complex blocks, or a large number of blocks, to avoid intimidating novice users [16]. This approach prevented us from considering alternate paradigms of interacting with data programmatically, such the read-eval-print loop (REPL) that is often seen in data-focused programming languages. One design that we considered involved the use of the list data structure in Scratch. We ended up not using this approach because research has shown that list are used infrequently in Scratch [2] and also because lists are not "first class" (i.e., they cannot be used as input to an outer block or inserted into another list).

Scratch Community Blocks was designed as an extension to the Scratch language and was made available directly inside the Scratch editor. To use the systems, a user clicks on an "Add Extension" button in the "More Blocks" category. This shows a dialog box listing several available extensions that the user can pick from including the *Community Blocks* system. When

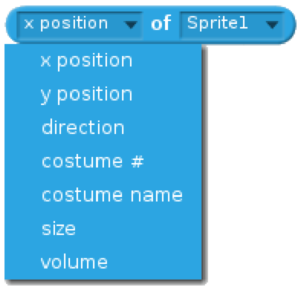


Figure 2: Accessor method block for Scratch sprites, with a dropdown to select the appropriate property. The shape of the block indicates it is a "reporter" block, and that it can be used as an input to another blocks (e.g., the "say" block).

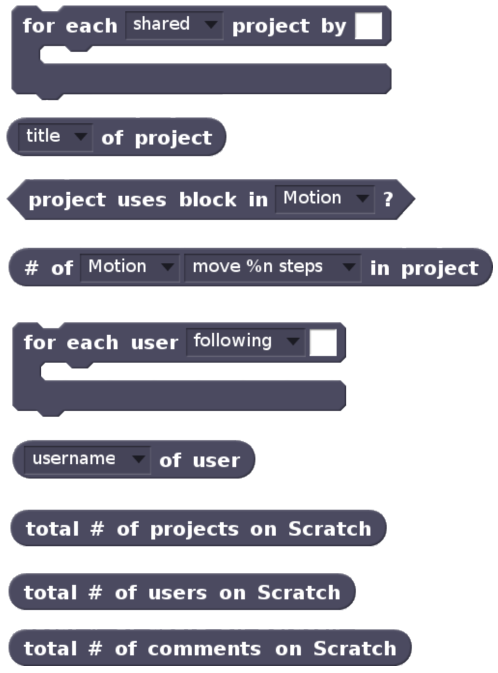
users choose the extension, a new palette of programming blocks, shown in Figure 3, appears in the programming editor.

Although the Scratch language does not meet all the criteria necessary to be an object-oriented language (e.g., there is no inheritance mechanism), the model of programming in Scratch can be described as object-centered. A sprite in Scratch can be thought of as an object with a number of properties that might include its position and orientation on the Scratch stage or a set of graphical "costumes." Each sprite is also associated with a set of programmatic scripts that defines its behavior. *Scratch Community Blocks* were modeled upon the *accessor* or *getter* method block for a Scratch sprite (shown in Figure 2), which is a widely-used part of the core Scratch language. In an accessor block, the first dropdown menu lets the programmer select the property they are interested in; the second dropdown menu lets them choose a sprite from within the project. An accessor block can be embedded inside other blocks and will return data as determined by the parameters set through the two dropdown menus when executed.

Instead of returning data on sprites within projects, *Scratch Community Blocks* is designed to report data from across the Scratch online community. This model is also similar to the Object/Relationship Mapping (ORM) abstraction layer used to bridge an object-oriented language or development framework with an underlying relational database [3]. *Scratch Community Blocks* provides data on the two most salient entities within the Scratch website and community: projects and users. Projects are the central site for interaction in Scratch and receive about 24% of website traffic. User profiles are also seen as an important site for interaction and are the second most important locus with 7% of website traffic.[1] One design challenge was that while Scratch usernames are unique and widely known, projects are uniquely identified only by numeric IDs that are not surfaced in the Scratch user interface.

As a result, usernames are the only visible primary "key" in *Scratch Community Blocks*. The user profile page in Scratch consists of data that the user shares about themself (e.g., "about me," "what I'm working on"), as well as lists of projects shared, projects "favorited" or bookmarked, users followed

[1]Data for first quarter of 2016, from Google Analytics. Remaining traffic is to the front page, studios, and variety of other pages.


for each shared project by
title of project
project uses block in Motion ?
# of Motion move %n steps in project
for each user following
username of user
total # of projects on Scratch
total # of users on Scratch
total # of comments on Scratch
Figure 3: Full palette of programming blocks made available through the *Scratch Community Blocks* system.

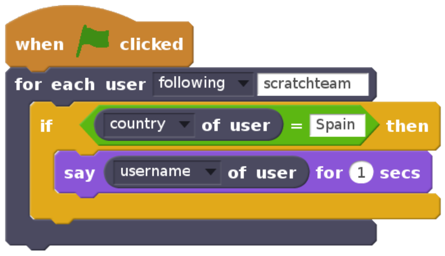
("followees"), and users who follow the user ("followers"). Using *Scratch Community Blocks*, these lists are represented as a combination of "for each" loop blocks and context-sensitive accessor blocks that are intended to be used inside the loops. For example, to get the title of each project shared by a given user, the "`title▼ of project`" block needs to be placed inside the "`for each shared▼ project by` ___" as shown in the script in Figure 1.

In *Scratch Community Blocks*, the "for each" loop blocks represent queries to fetch:

- all shared projects by a user
- all favorited (bookmarked) project by a user
- all users who follow a given user
- all users who are followed by a given user

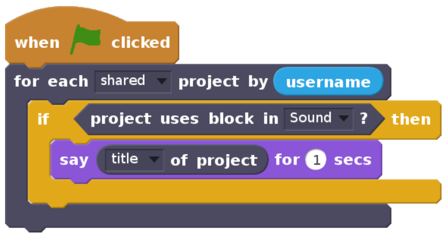
For the project object type, three distinct context-sensitive accessor blocks are made available in the system. These blocks provide not only social metadata (e.g., number of comments on the project) but also code metadata (e.g., number of blocks of a certain type used in a given project, or whether a project uses blocks from a certain category). Specifically, the system includes a predicate block that returns a boolean `true` or `false` value depending on whether a project uses blocks from a certain category in Scratch (e.g., control, sound), a block that returns the number of instances of a specific block (e.g., "`wait ___ seconds`"), and another block that can return the following social metadata on the project:

- title of the project
- description of the project
- number of "love-its" received by the project



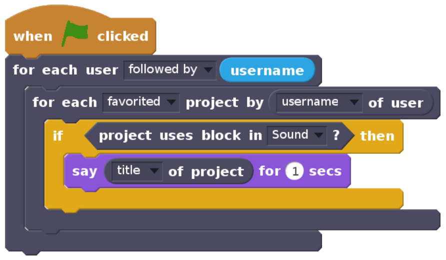
```
when clicked
for each user following scratchteam
    if country of user = Spain then
        say username of user for 1 secs
```

(a) Script to filter followers by country.



```
when clicked
for each shared project by username
    if project uses block in Sound ? then
        say title of project for 1 secs
```

(b) Script to filter projects by use of of blocks from the "Sound" category.



```
when clicked
for each user followed by username
    for each favorited project by username of user
        if project uses block in Sound ? then
            say title of project for 1 secs
```

(c) Script to find curated projects that use blocks from the Sound category.

Figure 4: Simple code examples using *Scratch Community Blocks* to show the system's design and functionality.

- number of users who have bookmarked/favorited the project
- number of comments on the project

For the user object type, there is only one accessor block, through which the following properties are made available:

- username of the user
- short public biography shared by the user ("about me")
- country the user is from (publicly shared and self-reported)

Although blocks reporting data on projects and users constitute most of the system, *Scratch Community Blocks* also includes three reporter blocks, shown at the bottom of Figure 3, that return community-wide statistics: the total number of projects, users, and comments.

An open question for us was whether to add a set of primitives for drawing visualizations (e.g., bar charts or scatter plots). In the end, we did not include visualization primitives. Instead, we provided users with sample projects that used the "pen" primitives in Scratch to render several standard visualizations. As we show in the next section, children creating visualizations were frequently driven by their own artistic tastes and sensibilities and eschewed the more canonical approach that we showcased in the samples.

ILLUSTRATIVE EXAMPLES

The best way to illustrate *Scratch Community Blocks* is to show how it can be used. Toward that end, we present three simple examples that demonstrate the possibilities with the system. Figure 4 shows the complete code of three small projects that demonstrate the operation of the system. The first sample project, in Figure 4a, demonstrates how one can filter, or search inside, the list of followers of a given user: the script iterates through the followers of the "scratchteam" user and "says" (in a speech bubble) the username of the followers who list Spain in the country field on their profile page.

Scratch has a number of blocks capable of producing sound, including blocks to play a recorded sound file, a musical note, or a drum. Within the Scratch system, these blocks are categorized as "Sound" blocks. The script shown in Figure 4b will iterate over all the projects shared by a user and say the title of the projects that use blocks from the sound category. An important difference between this script and the one shown in Figure 4a is the use of the "`username`" block that returns the username of the user currently viewing the project. This effectively provides customized results for the Scratch user viewing the project.

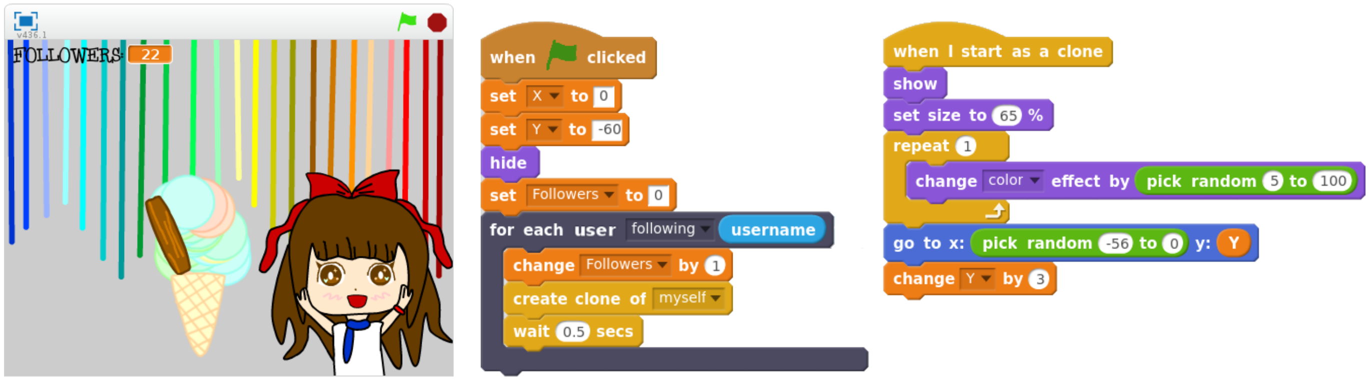
In the final example in Figure 4c, all the "favorited" projects of community members followed by the viewer are analyzed, and the title of the projects that contain blocks from the Sound category are presented. When run, this project will recommend projects that include sound or music to the viewer. These recommendations will have been socially curated by community members who are followed by the viewer.

FIELD EVALUATION WITH CHILDREN

We believe that the most compelling illustrations of the system are projects created by real users in a realistic setting. In this section, we describe a series of projects, created by children, that serve as illustrations of how *Scratch Community Blocks* can enable new pathways to learning data science and how it can make it possible for Scratch users to self-reflect on their learning and social participation.

In February 2016, we invited users to beta test the *Scratch Community Blocks*. Invitees were selected from a pool of active users on Scratch who had been in the community for at least 6 months and had shared at least 4 projects in the month preceding the selection. We looked only at project creation activity—we did not attempt to analyze the programming expertise or sophistication of these users. From this pool, 2,500 users were randomly selected and given access to the system in three phases.

To help users understand how to program with the blocks, we provided users with a set of example projects, most similar to those shown in Figure 4, that were highlighted prominently in the landing page of the beta-test website. Additionally, the Scratch editor was modified to show a sidebar with documentation on the functionality of the blocks. As is always the case in Scratch, users had the ability to view and interact with other users' projects, including the corresponding source code, which is made available through the "See Inside" functionality in Scratch. Throughout the duration of the beta program, we also actively curated the website, placing projects that made interesting use of *Scratch Community Blocks* on the landing page. Although the beta test took place on a separate website; the data accessed through *Scratch Community Blocks* reflected what was there on the main Scratch website.



```
when ⚑ clicked
set X to 0
set Y to -60
hide
set Followers to 0
for each user following username
    change Followers by 1
    create clone of myself
    wait 0.5 secs
```

```
when I start as a clone
show
set size to 65 %
repeat 1
    change color effect by pick random 5 to 100
go to x: pick random -56 to 0 y: Y
change Y by 3
```

(a) Visualization

(b) Code for the "scoop" sprite that creates the visualization through cloning the sprite.

Figure 5: Ice cream visualization project where the number of scoops on the cone is determined by the number of followers.

Self-reported gender data show that among the 2,500 users, 1,187 (47.5%) were male, 1,184 (47.4%) were female, 114 (4.6%) identified themselves as other, and for 15 users, data was not available. The median age was 12, and the interquartile range was between 11 and 15. Over an approximately 4 month period, 721 of the invited users created 1,660 projects using the new blocks. These users created between 1 and 19 projects, with a mean of 2.3, median of 1, and standard deviation of 2.3. Toward the end of the beta test, we sent out a survey with 2 multiple choice and 8 open-ended questions to all the 2,500 invited users. 499 users responded. Additionally, we hosted a 3-hour workshop with 12 Boston-area Scratch users where we took detailed field notes and interviewed two participants. Finally, we interviewed 3 users from the online community over voice/video chat.

An analysis of the relative usage frequencies of the new blocks indicate that the most commonly used block was the one to get social metadata of projects (`title▼ of project`), followed by the one to iterate through projects (`for each shared▼ project by ___`). 86% of the survey respondents rated the system as "excellent" or "very good", and 58% reported that they spent at least an hour using it. A histogram showing the relative usage frequencies of the new blocks, and a full breakdown of the quantitative survey responses are included in the supplementary material for this paper.

Using a qualitative, open-ended coding process on the projects, interview transcripts, and comments on projects, we identified a variety of ways in which users were engaging in data science and reflection. We describe exemplars of these activities below. The projects presented are of higher quality than many of the other projects shared on the site, but they are not exceptional, and each is representative of a broader "theme" or group of projects and comments. Throughout the rest of this section, we use pseudonyms and altered usernames to refer to users.

New pathways to data science

Scratch users were able to use *Scratch Community Blocks* to understand their world through computation and data. To that end, they created visualizations and representations of data as well as projects that tried to answer questions about social behaviors and learning activities of themselves and their peers.

Representing and visualizing data

Jondroidous (13 years old) created a visualization project that prompted for a username and then visualized the distribution of block categories in all the projects shared by that user as a doughnut chart. An example of the visualization is shown in Figure 6. When Jondroidous first created the project, there was a small bug in the code. As an illustration of the power of the social context provided by Scratch for learning and iteration, another user, Chewie184 (12 years old), remixed the project to fix the bug and improved the project by creating a progress bar.

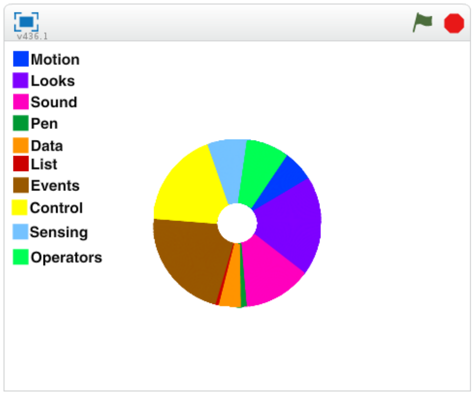


Figure 6: Doughnut chart project showing distribution of all blocks ever used by a user over block categories.

As an illustration of how data analysis done by children may be more in line with the interests and desires of children than information visualizations created by adults, dashboards made using *Scratch Community Blocks* were not restricted to "standard" visualization types such as pie charts, line graphs, etc. For example, AwesomeNemo (12 years old) made an "ice cream visualization" of a user's Scratch activity (shown in Figure 5) where the number of scoops on the ice cream cone is determined by the number of followers. This project, as well as several other non-canonical visualization projects, made extensive use of Scratch's `clone` block that allows the creator of a project to programmatically create copies of sprites on the Scratch stage. In this particular project, a single sprite representing the scoop of ice cream was cloned repeatedly (number of repetitions = number of followers) to achieve the desired effect (Figure 5b).

To visualize one's creative and social activity on the site, AwesomeNemo made another project that consisted of a computationally generated island where various characteristics of the island were based on the viewer's activity in the community. She wrote in the description of the project on the website:

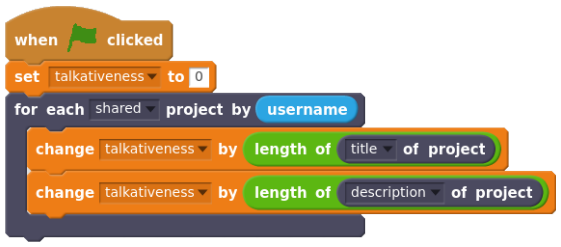
> Your island will be generated on a basis of how many followers, shared projects, and favourited projects you have!: Beauty is how many stars your island has in the sky (favourited projects). Habitability is how many houses your island has (followers). Quality of life is how many trees your island has (shared projects).

When asked about her motivation behind the island visualization during an interview with us, AwesomeNemo replied that she did not want to create a canonical visualization like a bar chart—rather, she wanted to explore more creative possibilities:

> I was thinking about how I can make a project using more of the blocks, because [. . .] I only used one in the ice cream one. So I was thinking about how I could use it to show statistics of how many followers and things you have, like not just in a bar chart.

Answering questions with data

Many projects created with *Scratch Community Blocks* answered questions on the activity of the user viewing these projects. In a sense, these were data science "apps" created by Scratch users for their peers.



```
when ⚑ clicked
set talkativeness to 0
for each shared project by username
    change talkativeness by length of title of project
    change talkativeness by length of description of project
```

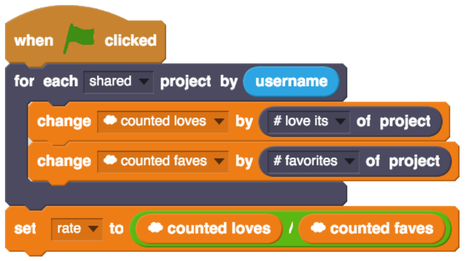
Figure 7: Part of the code to calculate Scratch user's "talkativeness" (slightly simplified).

A project by a 13-year-old titled "How talkative are you?" calculated the cumulative total length of the titles and descriptions of all the projects shared by the viewer of the project. It then added to this total the length of the username of the viewer and the length of the "about me" field on the the viewer's profile. The total was then presented to the viewer via a variable monitor widget on the Scratch stage (Figure 7 shows part of the code that did this analysis). Other projects analyzed if any of the project viewer's followers mentioned them in their "about me" text, while others calculated the total number of followers the viewer's followers had.

At the conclusion of the workshop, when we asked one of our participants (Sylvia) what other projects she would want to make with the blocks. She responded that before going on a vacation, she would like to make a project to find anyone in her social network on Scratch who lives in the place she's going, and then ask them for travel tips:

> Let's say you want to go traveling, and you want to ask a Scratcher who's from there – so you could just search through all your followers, and you can ask, "Hey! How's Paris?"

Embedded in this quote is evidence of Sylvia connecting her use of the new system with her existing knowledge (constructionism's continuity principle). Sylvia was, in essence, thinking about using *Scratch Community Blocks* to express in a formal way what she already knew about the Scratch community. She knew that someone in her social network might be from France. She realized that, with *Scratch Community Blocks*, she could access this information computationally to find those followers and connect with them.



```
when ⚑ clicked
for each shared project by username
    change ☁ counted loves by # love its of project
    change ☁ counted faves by # favorites of project
set rate to (☁ counted loves / ☁ counted faves)
```

Figure 8: Code to store and analyze data from multiple user trajectories using persistent Cloud Variables (the ☁ icon beside the variable names indicates that it is a Cloud Variable).

There were also a few projects that attempted to answer questions about the Scratch community in general. Scratch supports persistent data structures through the *Scratch Cloud Data* system created by Dasgupta [15]. Using this system, a variable in Scratch can be declared as persistent beyond the runtime of the project. Values set in the cloud variable are stored in a server, and the stored value is shared with all users accessing the project in question. Combining this system with *Scratch Community Blocks*, TheCoder486 (12 years old) created a project that collected data about users who ran the project and stored metadata about the cumulative number of love-its and favorites for each set of projects analyzed. The numbers were stored in persistent variables, as shown in the code from the project in Figure 8. With this simple approach, the project slowly built up a large sample dataset as more users accessed it. The question TheCoder486 was interested in was whether users tend to use the "love it" feature in Scratch more than the "favorite" feature. With the collected data as of date, his project suggests that love-its are 1.2 times more common than favorites.

Incorporating data in game mechanisms

Within the larger Scratch culture, games constitute a popular genre of projects created by community members. As the children using the *Scratch Community Blocks* system were active members of the larger community and were already deeply embedded within the culture of Scratch, it was not unexpected to see a number of games (e.g., quizzes that asked questions about the viewer's past projects) created with *Scratch Community Blocks* that incorporated data into their mechanisms.

The 13-year-old who made the "How talkative are you?" project described earlier also made a "data-driven" doll dress-up game in which the viewer of the project has to pick and choose clothes and accessories for a virtual doll. Although there are a number of similar games on the Scratch website, what made this game different was the use of data about the player's participation in the Scratch community to determine the purchasing power of the player.

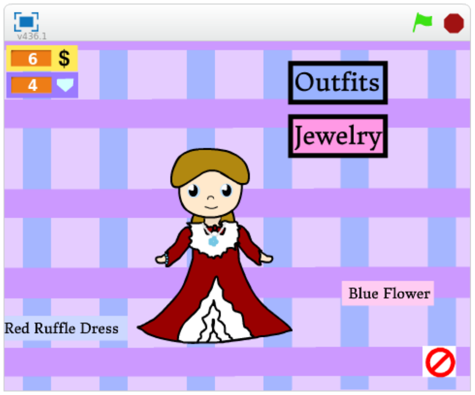


Figure 9: Data-driven dress-up game in which social metrics are used to determine the project viewer's purchasing power.

For each project shared, the player got one virtual dollar, and for each follower, a virtual diamond. The dollars and the diamonds could then be used to purchase clothes and jewelry for the virtual doll in the project (Figure 9). When asked in an interview about how she came up with the idea for this project, the creator of the project responded by saying that she wanted to do something unique that incorporated her interest in making art on Scratch:

> I was trying to think of something that somebody hadn't done yet, and I didn't see that. And also I really like to do art on Scratch and that was a good opportunity to use that and mix the two together.

While games may not necessarily fall in the category of traditional data science tools, the children creating these projects were demonstrating fluency and creativity in ways in which they could put social and behavioral data to use. Additionally, as the example above shows, as children created projects, they connected to their own interests, identities, and aesthetic sensibilities.

Self-reflection on learning and social participation

Access to data and to data analytics tools created by their peers enabled Scratch users to self-reflect upon their own learning and social participation in Scratch. This self-reflection often happened in a public setting (in project comments), and in some cases, during interviews that we conducted. For example, on seeing the results of a pie-chart visualization of the relative proportion of block categories in their shared projects, a 15-year-old user commented, "epic! looks like we need to use more pen blocks. :D"

In another example, during an interview with us, 14-year-old Alec said he realized that he usually focused on a certain type of block and did not really use others. We asked him if this realization meant that he would use the less frequently used blocks more in the future, and he replied affirmatively.

> **Alec:** It made me think that I actually realized that sometimes I usually focus on a certain type of block. I usually make pen projects and I realized that I don't actually use sound or motion blocks that much.
> **Interviewer:** Would you like to use them more, now that you know?
> **Alec:** Yes, I probably would be.

Not all self-reflection ended in positive feelings. After seeing the average number of "love its" on his projects, the 13-year-old author of the project that calculated this statistic left a sarcastic comment on his project: "Average no. of loves—four. Well, that's not depressing at all :'(."

However, by and large, most of the self-reflection we saw had a positive tenor. In a survey response, one user wrote:

> I thought maybe I should expand my range of blocks that I would use commonly. I don't know what about it gave me this thought, but it did.

In these projects, comments, and survey responses, we see children looking back at their own activities within the Scratch community, through data tools made by themselves and their peers to reflect on their participation in Scratch. By creating and sharing projects with *Scratch Community Blocks*, Scratch users not only got a chance to reflect on their own activity, but also they enabled other members of the community—whoever viewed the project—to engage in the same reflective exercise.

IMPLEMENTATION

Though *Scratch Community Blocks* is presented as an extension to the Scratch programmer, it is not written in JavaScript like standard Scratch extensions. Instead, it is implemented internally within the core Scratch code-base in ActionScript (the language Scratch is written in). This is due to the fact that the standard Scratch extension system does not allow for additions to the grammar of the language [16], and the design of *Scratch Community Blocks* necessitates the new types of loops that are described in the previous section.

Internally, loop blocks in the *Scratch Community Blocks* system send queries as HTTPS GET requests to the Scratch website API server. The API server parses the request, translates the request into an appropriate SQL statement, and then uses this SQL statement to query a database server. Results from the database are transformed into paginated JSON data and sent back to the interpreter as in response to the original HTTPS request. These responses are then parsed and stored for use by the query result accessor blocks. A diagram of this process is shown in Figure 10. For code metadata result accessor blocks, an extra HTTPS request is sent to an online parser web service by the result accessor block itself. This online parser web service fetches the project being requested, parses the project

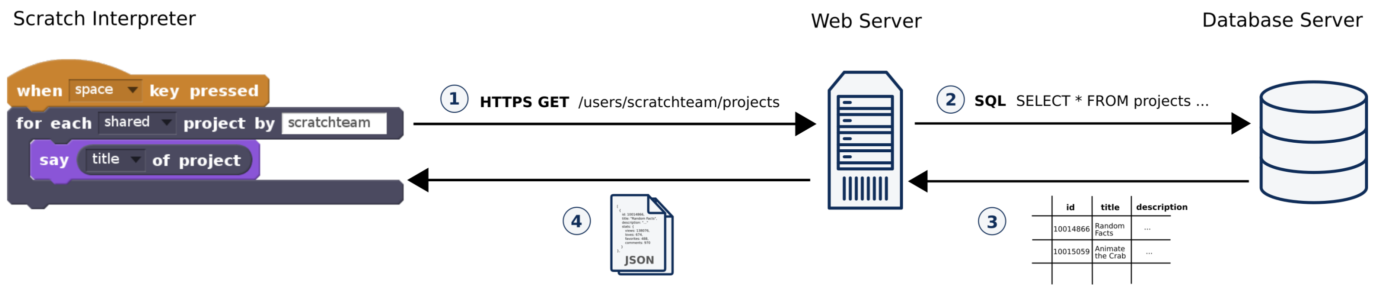


Figure 10: System diagram for *Scratch Community Blocks*. In step (1), the Scratch interpreter sends a HTTPS GET request to the Scratch API web server. In step (2), the API server translates the request URL into a SQL query and sends it to the database. The database sends back the data in step (3), which is then transformed into JSON and sent to the interpreter by the web server as a response to the original HTTPS request from step (1). Once the complete data is received, the interpreter starts the loop.

code, and sends back the result as a response to the HTTPS request.

The Scratch interpreter, designed by Maloney et al. [26], implements a cooperative round-robin scheduler where every loop block in Scratch has a built-in yield. Thus, all runnable scripts get a chance to run every display cycle, providing the illusion of fine-grained concurrency. Blocks that request data from the server simply wait (i.e., pause execution of their script) while all other scripts (e.g. those which update the display) continue to run. When data arrives, execution proceeds in that script. This behavior is consistent with other blocks in Scratch that wait for something to complete before proceeding, such as the "`wait ___ seconds`" or "`play sound ___ until done`" blocks.

A challenge with this approach is that Scratch was designed to be tinkerable and support "live" coding [26] with a minimal distinction between "live" and "edit" modes. This distinction is blurred to allow users to get a sense of what blocks or code can be used for. In *Scratch Community Blocks*, the time taken by the website API to send back the full results over multiple pages can be non-negligible (i.e., on the order of a few minutes for the largest conceivable requests), and this lag can affect the perceived tinkerability of the system. As a partial workaround, we designed the system to cache results of queries and reuse the cached results when a given script runs a second time. This means that scripts will run quickly after an initial execution if its input parameters are unchanged. The trade-off is the possibility that a program written with *Scratch Community Blocks* may operate with stale data. We felt that this trade-off was worth preserving the tinkerability that characterizes the Scratch language.

LIMITATIONS

Although *Scratch Community Blocks* supported youth exploration of data, it was not without limitations. Some of its shortcomings were highlighted by our users through feedback in the forums and the survey. Common requests for enhancements included calls for more support materials (documentation and sample projects), as well as for a faster and more responsive system. Additionally, we identified several difficulties faced by users that point toward shortcomings and limitations of the current design.

One common misconception (observed in project shared by 19% of users) involved the use of the context-sensitive accessor blocks. Context-sensitive blocks can be thought of as being similar to variables that have limited scope. In the case of the *Scratch Community Blocks*, most accessor blocks are not "valid" outside of their associated loops. This pattern of usage was more common with the accessor block for user data. In most of these cases, as in the example shown in Figure 11, it appears that the creator of the project expected the block to return data about the user currently viewing the project.

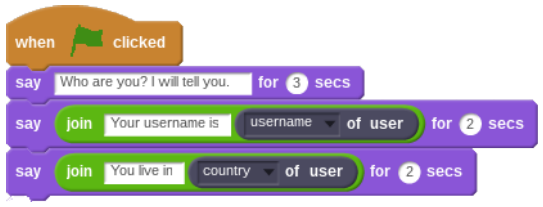

when clicked
say Who are you? I will tell you. for 3 secs
say join Your username is username of user for 2 secs
say join You live in country of user for 2 secs

Figure 11: Code from user-created project that illustrates how the context-sensitive accessor block is incorrectly used.

Context-sensitive blocks are a general problem with blocks-based languages. For example, in Scratch, text input from a user is stored in an "`answer`" block that is valid only if the programmer had previously prompted the user for input. The current design of *Scratch Community Blocks* uses these context-sensitive blocks prominently, making this problem particularly salient. Currently, other than showing an error message—an approach Scratch has explicitly avoided supporting [26]—there is no obvious or easy workaround to this problem.

Another source of difficulty among users was in correctly scaling visualizations. For example, a very active user created a "cupcake visualization" where the size of the cupcake depended on the number of projects shared by the viewer. If the project was viewed by another user with a small number of shared projects, the cupcake image would be scaled to a size that made it barely visible. The bug alluded to earlier in the doughnut chart visualization example in Figure 6 was caused by the fact that the sections of the doughnut were scaled by the number of blocks used by the project's creator. As a result, if a user who had used fewer blocks viewed the project, the visual did not form a complete circle. If a user who had used more blocks viewed it, the sections on the chart would overflow and draw over previously drawn elements. This issue can be seen as a consequence of our decision not to include visualization primitives. If we had included visualization primitives with inbuilt mechanisms for scaling, our users would not have encountered this particular difficulty.

Additionally, it was clear in several cases that a project creator expected data accessed using the blocks to be updated in real time. This was commonly seen with the overall statistics blocks, as they were embedded within loops in some cases, indicating that the author of the project expected new values in different iterations of the loop. Unfortunately, real-time updating is not supported by the underlying architecture of the system.

Finally, while we found many examples of successful uses of the system, we have little data about invited users who were unsuccessful in using the system. In this sense, our evaluation may be biased toward users who were successful in using the system on their own, at least in the bulk of the data drawn from outside of the workshop. This is a common challenge in studies of informal learning online. Though a significant portion of the users that we invited were never active on the website, we do not know if this inactivity was due to difficulties in using the system or if they were busy or uninterested. Future studies in more controlled contexts (e.g., in workshops with researchers present) may shed more light on the overall usability of the system.

CONCLUSION

In this paper, we presented the motivation and design of *Scratch Community Blocks*. We explained how the system is designed to allow children to engage directly in processes of data analysis normally practiced only by adults in order to satisfy our twin goals of promoting new pathways to learning about data analysis and promoting reflection. We demonstrated, through a series of examples and case studies, how *Scratch Community Blocks* both engages children in visualizing, representing, and answering questions with data in new ways and also supports self-reflection on learning and social participation. Finally, we presented a discussion of the limitations of the current design based on problems that users encountered.

As a system designed to support a constructionist approach to data science, *Scratch Community Blocks* enables learners to learn by designing and building projects that help them answer questions they are interested in and to represent and make use of data in ways that speak to their styles and identities. Learners can also share their work with their peers. These shared creations are not only viewable by others but can also be used by peers to reflect on their experiences and learning.

Among the related systems that informed our design, *Scratch Community Blocks* is a unique design in several senses. The system not only fosters data-mediated reflection on learning, it also enables users to construct their own visualizations and analyses in ways that are often very different from canonical forms of data representation and analysis. We believe that this unique combination of affordances is potentially useful in a wide range of personal informatics contexts.

For educators, our system can be seen providing a unique pathway into data science. While many introductory toolkits for data science education focus on data sets that may or may not be of particular relevance to learners, users of *Scratch Community Blocks* analyze data that is about themselves and that has been created by themselves. Although Scratch is an informal context for learning, we believe that a system such as ours has merits and could be used in formal settings as well. As described by Brennan [10], an interest-driven and exploration-focused approach like the one supported by our system might be utilized to enable learners to see "entry points and trajectories of participation" in an existing disciplinary realm, such as data science.

Although the projects created by youth that we have presented can appear superficially different from the kinds of analysis completed by educational researchers and learning scientists, children are using *Scratch Community Blocks* to write computer programs to ask and answer question using data about their own activities and learning. In his article "Teaching Children to be Mathematicians vs. Teaching About Mathematics" [32], Papert showed how Logo could offer children a space to use and engage with mathematical ideas in creative and personally motivated ways. This, he argued, enabled children to go beyond knowing about mathematics to "doing" mathematics, as a mathematician would. In a similar fundamental sense, *Scratch Community Blocks* allows children to *do* data science, and not just know about it.

In his book *Mindstorms: Children, Computers, and Powerful Ideas*, Papert presented the idea of the "child as the epistemologist," where children, through construction and reflection, not only learn new and powerful ideas but also think about their own thinking and learn about their own learning [30]. Inspired by Papert's vision, we feel that the greatest promise of *Scratch Community Blocks* and its approach is that, in enabling children to understand and analyze their own learning and social participation, it may not only help children become data scientists and analysts but also encourage them to become epistemologists as well.

ACKNOWLEDGMENTS

This paper is based on the first author's PhD dissertation, and some of the text and images in the paper appear in the thesis. We would like to thank members of the Scratch community for their time and for inspiring us with their creative work. We would also like to acknowledge Mitchel Resnick, Natalie Rusk, Hal Abelson, John Maloney, and our anonymous reviewers for their support and thoughtful feedback. Financial support for this work came from the National Science Foundation (grants DRL-1417663 and DRL-1417952).

REFERENCES

1. Harold Abelson and Andrea diSessa. 1986. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press, Cambridge, MA.
2. Efthimia Aivaloglou and Felienne Hermans. 2016. How Kids Code and How We Know: An Exploratory Study on

the Scratch Repository. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, New York, NY, USA, 53–61. DOI:**http://dx.doi.org/10.1145/2960310.2960325**

3. Scott W Ambler. 2000. Mapping objects to relational databases: What you need to know and why. (July 2000). **https://www.ibm.com/developerworks/library/ws-mapping-to-rdb/**
4. Jana Beck. 2013. iPancreas. (2013). **https://github.com/jebeck/iPancreas-archive**
5. Dani Ben-Zvi and Keren Aridor-Berger. 2016. Children's Wonder How to Wander Between Data and Context. In *The Teaching and Learning of Statistics: International Perspectives*, Dani Ben-Zvi and Katie Makar (Eds.). Springer International Publishing, Cham, 25–36. **http://dx.doi.org/10.1007/978-3-319-23470-0_3**
6. Yochai Benkler. 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, CT.
7. Rahul Bhargava and Catherine D'Ignazio. 2015. Designing Tools and Activities for Data Literacy Learners. (2015).
8. Marie Bienkowski, Minguy Feng, and Barbara Means. 2012. *Enhancing teaching and learning through educational data mining and learning analytics: An issue brief*. Technical Report. US Department of Education, Office of Educational Technology.
9. David Boud, Rosemary Keogh, and David Walker. 1985. *Reflection: Turning Experience Into Learning*. Routledge, New York, NY.
10. Karen Brennan. 2013. *Best of both worlds: Issues of structure and agency in computational creation, in and out of school*. Ph.D. Dissertation. Massachusetts Institute of Technology.
11. Jerome S. Bruner. 1986. *Actual Minds, Possible Worlds*. Harvard University Press, Cambridge, MA.
12. Richard Catrambone and Mark Guzdial. 2012. Answering Questions with Internet Data: Computational Tools for Social Studies Analysis. (2012). **http://swiki.cc.gatech.edu/compfreak**
13. Community Data Science Collective. 2015. Community Data Science Collective. (2015). **http://wiki.communitydata.cc/**
14. Drew Conway. 2010. The data science Venn diagram. (2010). **http://www.dataists.com/2010/09/the-data-science-venn-diagram/**
15. Sayamindu Dasgupta. 2013. From Surveys to Collaborative Art: Enabling Children to Program with Online Data. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 28–35. DOI: **http://dx.doi.org/10.1145/2485760.2485784**
16. Sayamindu Dasgupta, Shane M. Clements, Abdulrahman Y. Idlbi, Chris Willis-Ford, and Mitch Resnick. 2015. Extending Scratch: New pathways into programming. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Atlanta, GA, 165–169. DOI: **http://dx.doi.org/10.1109/VLHCC.2015.7357212**
17. Noleine Fitzallen and Jane Watson. 2010. Developing statistical reasoning facilitated by TinkerPlots. In *Data and context in statistics education: Towards an evidence-based society. Proceedings of the Eighth International Conference on Teaching Statistics (ICOTS8, July, 2010), Ljubljana, Slovenia.*, Chris Reading (Ed.). International Statistical Institute, Voorburg, The Netherlands.
18. Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. 2010. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10)*. ACM, New York, NY, USA, 1061–1066. DOI:**http://dx.doi.org/10.1145/1807167.1807286**
19. Sudheendra Hangal, Monica S. Lam, and Jeffrey Heer. 2011. MUSE: Reviving Memories Using Email Archives. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 75–84. DOI: **http://dx.doi.org/10.1145/2047196.2047206**
20. Jonathon David Hellmann. 2015. *DataSnap: Enabling Domain Experts and Introductory Programmers to Process Big Data in a Block-Based Programming Language*. Thesis. Virginia Tech. **https://vtechworks.lib.vt.edu//handle/10919/54544**
21. Deepak Jagdish. 2014. *IMMERSION : a platform for visualization and temporal analysis of email data*. Thesis. Massachusetts Institute of Technology. **http://dspace.mit.edu/handle/1721.1/95606**
22. Iain Johnstone and Fred Roberts. 2014. *Data Science at NSF*. Technical Report. National Science Foundation. **http://www.nsf.gov/attachments/129788/public/Final_StatSNSFJan14.pdf**
23. Khan Academy. 2013. Introducing...the Learning Dashboard. (Aug. 2013). **https://www.khanacademy.org/about/blog/post/58354379257/introducingthe-learning-dashboard**
24. Vasiliki Laina and Michelle Wilkerson. 2016. Distributions, Trends, and Contradictions: A Case Study in Sensemaking with Interactive Data Visualizations. In *Proceedings of the 12th International Conference of the Learning Sciences*. The International Society of the Learning Sciences, Singapore.
25. Victor Lee. 2013. The Quantified Self (QS) Movement and Some Emerging Opportunities for the Educational Technology Field. *Educational Technology* November-December 2013 (Oct. 2013), 39–42. **http://digitalcommons.usu.edu/itls_facpub/480**

26. John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.* 10, 4 (Nov. 2010), 16:1–16:15. DOI: **http://dx.doi.org/10.1145/1868358.1868363**

27. Fred Martin, Sarah Kuhn, Michelle Scribner-MacLean, Christopher Corcoran, James Dalphond, John Fertitta, Michael McGuiness, Sam Christy, and Ivan Rudnicki. 2010. iSENSE: A Web Environment and Hardware Platform for Data Sharing and Citizen Science. *AAAI Spring Symposium Series; 2010 AAAI Spring Symposium Series* (2010). **http://www.aaai.org/ocs/index.php/SSS/SSS10/paper/view/1099**

28. Andrés Monroy Hernández. 2007. ScratchR: sharing user-generated programmable media. In *Proceedings of the 6th international conference on Interaction design and children (IDC '07)*. ACM, New York, NY, USA, 167–168. DOI: **http://dx.doi.org/10.1145/1297277.1297315**

29. Dawn Nafus. 2006. *Quantified: Biosensing Technologies in Everyday Life*. The MIT Press, Cambridge, Massachusetts.

30. Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

31. Seymour Papert and Idit Harel. 1991. Situating constructionism. In *Constructionism*. Ablex Publishing, New York, NY, US, 1–11.

32. Seymour A. Papert. 1971. *Teaching Children to be Mathematicians vs. Teaching About Mathematics*. Technical Report 249. Massachusetts Institute of Technology, Cambridge, MA. **http://dspace.mit.edu/handle/1721.1/5837**

33. Jean Piaget. 1970. *Genetic epistemology. Trans. E. Duckworth*. Columbia University Press, New York, NY, US.

34. John Podesta. 2014. *Big Data: Seizing Opportunities, Preserving Values*. Technical Report. Executive Office of the President, United States of America.

35. Hans Põldoja. 2010. EduFeedr - following and supporting learners in open blog-based courses. In *Open ED 2010 Proceedings*. Barcelona. **http://openaccess.uoc.edu/webapps/o2/handle/10609/4861**

36. Mitchel Resnick. 2008. Sowing the Seeds for a More Creative Society. *Learning & Leading with Technology* 35, 4 (2008), 18–22.

37. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. DOI: **http://dx.doi.org/10.1145/1592761.1592779**

38. Mitchel Resnick and Brian Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children (IDC '05)*. ACM, New York, NY, USA, 117–122. DOI: **http://dx.doi.org/10.1145/1109540.1109556**

39. Verónica Rivera-Pelayo, Valentin Zacharias, Lars Müller, and Simone Braun. 2012. Applying Quantified Self Approaches to Support Reflective Learning. In *Proceedings of the 2Nd International Conference on Learning Analytics and Knowledge (LAK '12)*. ACM, New York, NY, USA, 111–114. DOI: **http://dx.doi.org/10.1145/2330601.2330631**

40. Eric (Eric Ross) Rosenbaum. 2009. *Jots : Cultivating reflective learning in Scratch*. Thesis. Massachusetts Institute of Technology. **http://dspace.mit.edu/handle/1721.1/55197**

41. Donald A Schön. 1983. *The reflective practitioner: How professionals think in action*. Basic books, New York, NY, USA.

42. R. Benjamin Shapiro, Annie Kelly, Matthew Ahrens, and Rebecca Fiebrink. 2016. BlockyTalky: A Physical and Distributed Computer Music Toolkit for Kids. In *Proceedings of the 16th International Conference on New Interfaces for Musical Expression*. Brisbane, Australia. **http://research.gold.ac.uk/18635/**

43. Sarah Van Wart and Tapan S Parikh. 2013. Increasing Youth and Community Agency in GIS. In *GeoHCI Workshop at CHI 2013*.

44. Katrien Verbert, Erik Duval, Joris Klerkx, Sten Govaerts, and José Luis Santos. 2013. Learning Analytics Dashboard Applications. *American Behavioral Scientist* 57, 10 (Oct. 2013), 1500–1509. DOI: **http://dx.doi.org/10.1177/0002764213479363**

45. Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. 2007. ManyEyes: a Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1121–1128. DOI: **http://dx.doi.org/10.1109/TVCG.2007.70577**

46. Fernanda B. Viégas, Scott Golder, and Judith Donath. 2006. Visualizing Email Content: Portraying Relationships from Conversational Histories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 979–988. DOI: **http://dx.doi.org/10.1145/1124772.1124919**

47. Karen E Willcox, Sanjay Sarma, and Philip H Lippel. 2016. *Online Education: A Catalyst for Higher Education Reforms*. Technical Report. Massachusetts Institute of Technology.

48. Wolfram Research. 2015. Wolfram|Alpha Personal Analytics for Facebook: Last Chance to Analyze Your Friend Network. (April 2015). **http://company.wolfram.com/news/2015/wolframalpha-personal-analytics-for-facebook-last-chance-to-analyze-your-friend-network/**