




Imagining the future of statistical education software

Amelia McNamara (@AmeliaMN)

Assistant Professor, Department of Computer & Information Sciences
University of St Thomas, St Paul, MN USA



Key Attributes of a Modern Statistical Computing Tool

Amelia McNamara 

Statistical and Data Sciences, Smith College, Northampton, MA

ABSTRACT

In the 1990s, statisticians began thinking in a principled way about how computation could better support the learning and doing of statistics. Since then, the pace of software development has accelerated, advancements in computing and data science have moved the goalposts, and it is time to reassess. Software continues to be developed to help do and learn statistics, but there is little critical evaluation of the resulting tools, and no accepted framework with which to critique them. This article presents a set of attributes necessary for a modern statistical computing tool. The framework was designed to be broadly applicable to both novice and expert users, with a particular focus on making more supportive statistical computing environments. A modern statistical computing tool should be accessible, provide easy entry, privilege data as a first-order object, support exploratory and confirmatory analysis, allow for flexible plot creation, support randomization, be interactive, include inherent documentation, support narrative, publishing, and reproducibility, and be flexible to extensions. Ideally, all these attributes could be incorporated into one tool, supporting users at all levels, but a more reasonable goal is for tools designed for novices and professionals to “reach across the gap,” taking inspiration from each others’ strengths.

ARTICLE HISTORY

Received September 2016
Revised May 2018

KEYWORDS

Bootstrap; Data visualization;
Exploratory data analysis;
Randomization;
Reproducibility; Software
design; Software evaluation

1. Introduction

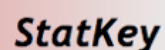
Tools shape the way we see the world, and statistical comput- tools are starting to blur, and we believe this lowers the barrier

tools designed for **learning** statistics are typically:

- graphical
- interactive
- intuitive
- supportive of EDA

but:

- don't support reproducibility
- can't handle real data

The logo for Fathom, featuring the word "Fathom" in a bold, blue, sans-serif font. The letter "o" is replaced by a 3D-rendered yellow sphere with a grid pattern.The logo for TinkerPlots, with "Tinker" in orange and "Plots" in blue. A dotted line arches over the text, and a small 3D sphere with blue dots is positioned between the two words.The logo for StatKey, consisting of the word "StatKey" in a bold, black, sans-serif font inside a light-colored rectangular box.

to accompany [Statistics: Unlocking the Power of Data](#)
by Lock, Lock, Lock, Lock, and Lock

Rossman/Chance Applet Collection

tools designed for **doing** statistics are typically:

- powerful
- flexible
- reproducible
- supportive of extensions

but:

- hard to get started using
- not interactive

The logo for Stata, featuring the word "Stata" in a white, bold, sans-serif font on a dark teal rectangular background.The logo for R, consisting of a stylized blue letter "R" inside a grey, metallic-looking ring.The logo for SPSS, with "SPSS" in white, bold, sans-serif font on a red square background. Below it, the text "AN IBM COMPANY" is written in a smaller white font.The logo for SAS, featuring a blue stylized "S" followed by "sas" in a bold, black, sans-serif font. Below the logo is the tagline "THE POWER TO KNOW" in a smaller, grey font.The logo for MATLAB, with the word "MATLAB" in a bold, black, serif font. Below it is the tagline "The Language of Technical Computing" in a smaller, italicized font.



We need a bridge between the two

**Could be software, or curriculum. Today, I'm
focused on software.**

Software for Learning and for Doing Statistics

Rolf Biehler

*Institut für Didaktik der Mathematik (IDM), Universität Bielefeld, Postfach 10 01 31, D-33501
Bielefeld, Germany.*

e-mail: rolf.biehler@post.uni-bielefeld.de

Summary

The community of statisticians and statistics educators should take responsibility for the evaluation and improvement of software quality from the perspective of education. The paper will develop a perspective, an ideal system of requirements to critically evaluate existing software and to produce future software more adequate both for learning and doing statistics in introductory courses. Different kinds of tools and microworlds are needed. After discussing general requirements for such programs, a prototypical ideal software system will be presented in detail. It will be illustrated how such a system could be used to construct learning environments and to support elementary data analysis with exploratory working style.

Key words: Statistics education; Statistical software design; Evaluation of statistical software; Exploratory data analysis; Simulation.

Table 1. Summary of attributes.

1. Accessibility
 2. Easy entry for novice users
 3. Data as a first-order persistent object
 4. Support for a cycle of exploratory and confirmatory analysis
 5. Flexible plot creation
 6. Support for randomization throughout
 7. Interactivity at every level
 8. Inherent documentation
 9. Simple support for narrative, publishing, and reproducibility
 10. Flexibility to build extensions
-

	Accessibility	Easy entry	Data as first-order object	EDA/CDA	Flexible plotting	Randomization	Interactivity	Inherent documentation	Narrative, publishing, reproducibility	Flexibility for extensions
Graphing calculators	*	✓								
Excel	*	✓					✓			
applets	*	✓				✓	✓			
TinkerPlots	*	✓		✓	✓	✓	✓	✓		
Fathom	*	✓		✓	✓	✓	✓	✓		
R	✓		✓	✓	✓	*	*		✓	✓
Python	✓		✓		*	*	*		✓	✓
SAS software			✓	✓		*			*	✓
Stata software			✓	✓		*			*	✓

Table 1: A summary of many currently-available tools for learning and doing statistics, and how they satisfy the attributes outlined in this paper. Asterisks indicate partial satisfaction of the attribute. For example, most tools are not accessible, either because of prohibitive cost or because they do not support disabled users. R and Python are free and can be used with adaptive technology. R, Python, SAS software, and Stata software get an asterisk for randomization because it is possible within the system, but difficult for novices. Similarly, R and Python can be used to create interactive graphics, but it is difficult, and SAS software and Stata software can be used to create reproducible reports, although it is difficult.

On the State of Computing in Statistics Education: Tools for Learning and for Doing. pre-print <http://bit.ly/StateOfComputingPreprint>

Accessibility

- free or inexpensive
- available on many platforms
- compatible with accessibility features

Easy entry for novice users

- the "complexity of tool problem" from Biehler
- known as "low threshold" in CS ed literature

TinkerPlots/Fathom

The screenshot displays the TinkerPlots software interface with the following components:

- Menu Bar:** Apple logo, TinkerPlots, File, Edit, Object, Data, Window, Help.
- Toolbar:** Cards, Table, Plot, Sampler, Text, Separate, Order, Stack, Ref. Line, Divider, Ruler, Hats, Line, Courts, Averages, Meter, Label, Key.
- Snack.csv Table View:**

Attribute	Value	Unit	Form...
SnackCos...	Less tha...		<input type="radio"/>
SnackCos...	0.5		<input type="radio"/>
SnackImage	SKIPPED		<input type="radio"/>
SnackLoc...	5		<input type="radio"/>
SnackLoc...	Vehicle		<input type="radio"/>
SnackPer...	0		<input type="radio"/>
SnackPer...	Mid-mor...		<input type="radio"/>
WhatSnack	Pb & j		<input type="radio"/>
WhoYouS...	0		<input type="radio"/>
WhoYouS	Alone		<input type="radio"/>
- Dot Plot View:** A dot plot showing the distribution of 'SnackCostlabel' values. The y-axis categories are: NOT_DISPLAYED, More than \$10.00, Less than \$1.00, \$7.00-\$10.00, \$5.00-\$7.00, \$3.00-\$5.00, and \$1.00-\$3.00. The x-axis categories are: Friends' houses, Home, NOT_DISPLAYED, Other, Party, Restaurant, School, Vehicle, and Work. A legend on the right shows the size of the circles representing the count of data points.
- Stacked Dot Plot View:** A stacked dot plot showing the distribution of 'SnackCostlabel' values. The y-axis categories are: \$1.00-\$3.00, \$3.00-\$5.00, \$5.00-\$7.00, \$7.00-\$10.00, Less than \$1.00, More than \$10.00, and NOT_DISPLAYED. The x-axis categories are: Friends' houses, Home, NOT_DISPLAYED, Other, Party, Restaurant, School, Vehicle, and Work. Percentages are shown above each stack of dots.

R Syntax Comparison :: CHEAT SHEET <http://bit.ly/R-syntax-sheet>

Dollar sign syntax

```
goal(data$x, data$y)
```

SUMMARY STATISTICS:

one continuous variable:
`mean(mtcars$mpg)`

one categorical variable:
`table(mtcars$cyl)`

two categorical variables:
`table(mtcars$cyl, mtcars$am)`

one continuous, one categorical:
`mean(mtcars$mpg[mtcars$cyl==4])`
`mean(mtcars$mpg[mtcars$cyl==6])`
`mean(mtcars$mpg[mtcars$cyl==8])`

PLOTTING:

one continuous variable:
`hist(mtcars$disp)`

`boxplot(mtcars$disp)`

one categorical variable:
`barplot(table(mtcars$cyl))`

two continuous variables:
`plot(mtcars$disp, mtcars$mpg)`

two categorical variables:
`mosaicplot(table(mtcars$am, mtcars$cyl))`

one continuous, one categorical:
`histogram(mtcars$disp[mtcars$cyl==4])`
`histogram(mtcars$disp[mtcars$cyl==6])`
`histogram(mtcars$disp[mtcars$cyl==8])`

`boxplot(mtcars$disp[mtcars$cyl==4])`
`boxplot(mtcars$disp[mtcars$cyl==6])`
`boxplot(mtcars$disp[mtcars$cyl==8])`

WRANGLING:

subsetting:
`mtcars[mtcars$mpg>30,]`

making a new variable:
`mtcars$efficient[mtcars$mpg>30] <- TRUE`
`mtcars$efficient[mtcars$mpg<30] <- FALSE`

Formula syntax

```
goal(y~x|z, data=data, group=w)
```

SUMMARY STATISTICS:

one continuous variable:
`mosaic::mean(~mpg, data=mtcars)`

one categorical variable:
`mosaic::tally(~cyl, data=mtcars)`

two categorical variables:
`mosaic::tally(cyl~am, data=mtcars)`

one continuous, one categorical:
`mosaic::mean(mpg~cyl, data=mtcars)`

tilde

PLOTTING:

one continuous variable:
`lattice::histogram(~disp, data=mtcars)`

`lattice::bwplot(~disp, data=mtcars)`

one categorical variable:
`mosaic::bargraph(~cyl, data=mtcars)`

two continuous variables:
`lattice::xyplot(mpg~disp, data=mtcars)`

two categorical variables:
`mosaic::bargraph(~am, data=mtcars, group=cyl)`

one continuous, one categorical:
`lattice::histogram(~disp|cyl, data=mtcars)`

`lattice::bwplot(cyl~disp, data=mtcars)`

The variety of R syntaxes give you many ways to “say” the same thing

read across the cheatsheet to see how different syntaxes approach the same problem

Tidyverse syntax

```
data %>% goal(x)
```

SUMMARY STATISTICS:

one continuous variable:
`mtcars %>% dplyr::summarize(mean(mpg))`

one categorical variable:
`mtcars %>% dplyr::group_by(cyl) %>% dplyr::summarize(n())`

two categorical variables:
`mtcars %>% dplyr::group_by(cyl, am) %>% dplyr::summarize(n())`

one continuous, one categorical:
`mtcars %>% dplyr::group_by(cyl) %>% dplyr::summarize(mean(mpg))`

PLOTTING:

one continuous variable:
`ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")`

`ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")`

one categorical variable:
`ggplot2::qplot(x=cyl, data=mtcars, geom="bar")`

two continuous variables:
`ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")`

two categorical variables:
`ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") + facet_grid(~am)`

one continuous, one categorical:
`ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") + facet_grid(~cyl)`

`ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars, geom="boxplot")`

WRANGLING:

subsetting:
`mtcars %>% dplyr::filter(mpg>30)`

making a new variable:
`mtcars <- mtcars %>% dplyr::mutate(efficient = if_else(mpg>30, TRUE, FALSE))`

the pipe

Scratch

The image shows a Scratch project window titled "Factorial (progopedia.com)". The project name is "Factorial" and it is locked. The stage shows a green dragon character with coordinates x: -167, y: -115, and direction: 90. The Scripts area contains the following code:

```
delete all of strs
set i to 0
set f to 1
repeat 17
  add join i join != f to strs
  set i to i + 1
  set f to f * i
hide
```

The right side of the window displays a list titled "Factorial strs" with 17 entries:

Index	Factorial
1	0! = 1
2	1! = 1
3	2! = 2
4	3! = 6
5	4! = 24
6	5! = 120
7	6! = 720
8	7! = 5040
9	8! = 40320
10	9! = 362880
11	10! = 3628800
12	11! = 39916800
13	12! = 479001600
14	13! = 6227020800
15	14! = 87178291200
16	15! = 1307674368000
17	16! = 20922789888000

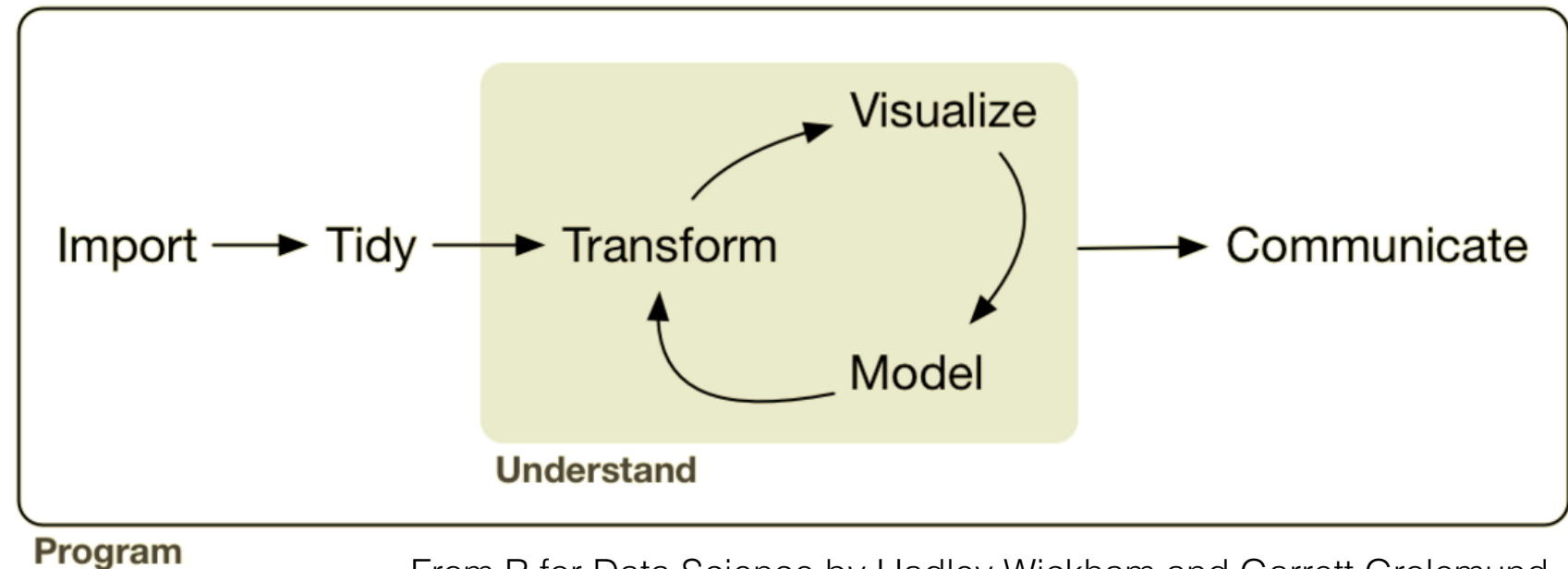
At the top of the right panel, there are two input fields: "Factorial i" with the value 17, and "Factorial f" with the value 355687428096000.

<https://scratch.mit.edu/>

Data as a first-order object

- focused on data
- data should be easily **human-readable**
- support many data types
- difficult to overwrite original data
- affordances for reproducibility

Support for a Cycle of Exploratory and Confirmatory Analysis

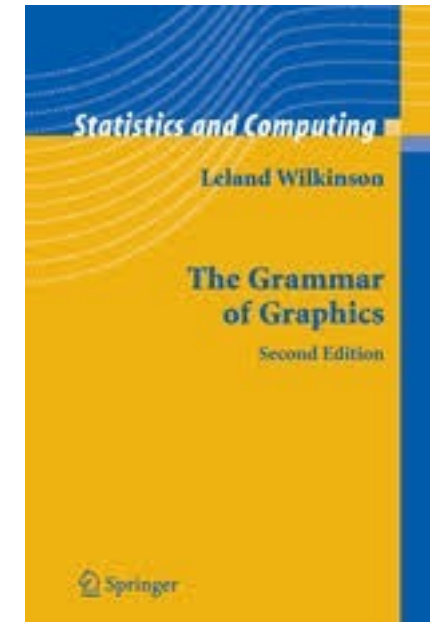


- Users need "scratch paper"

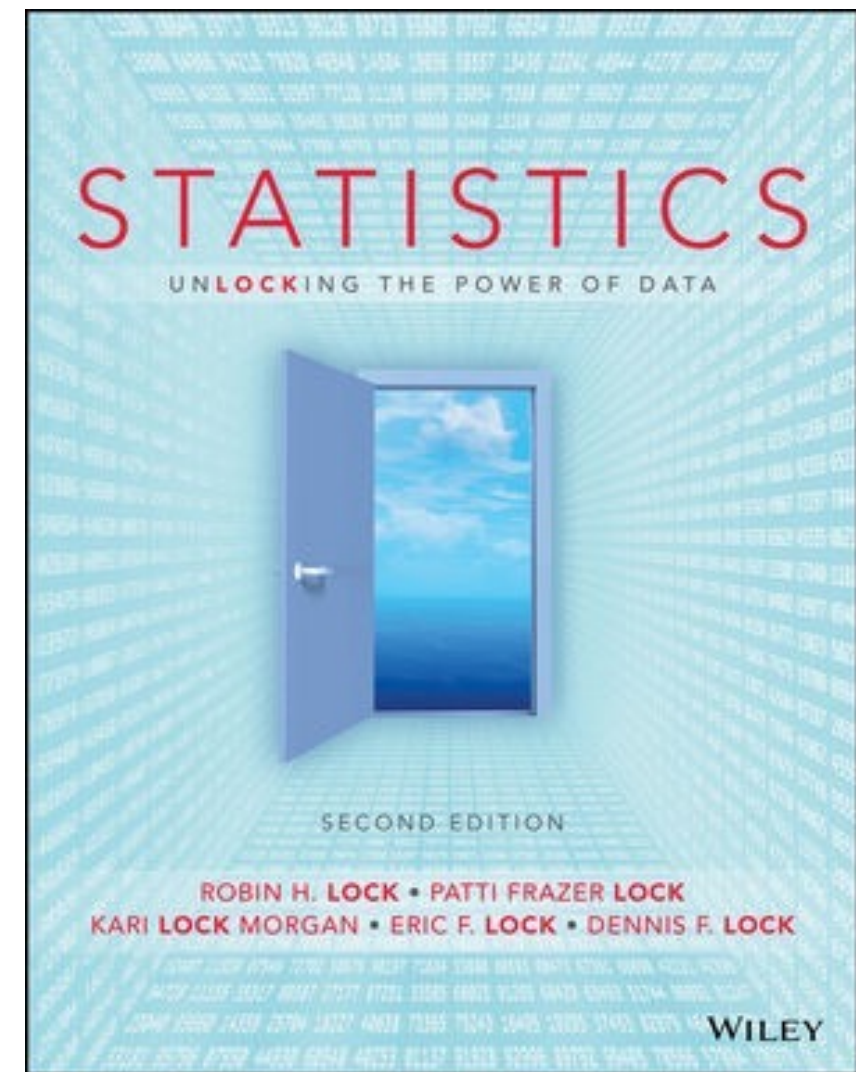
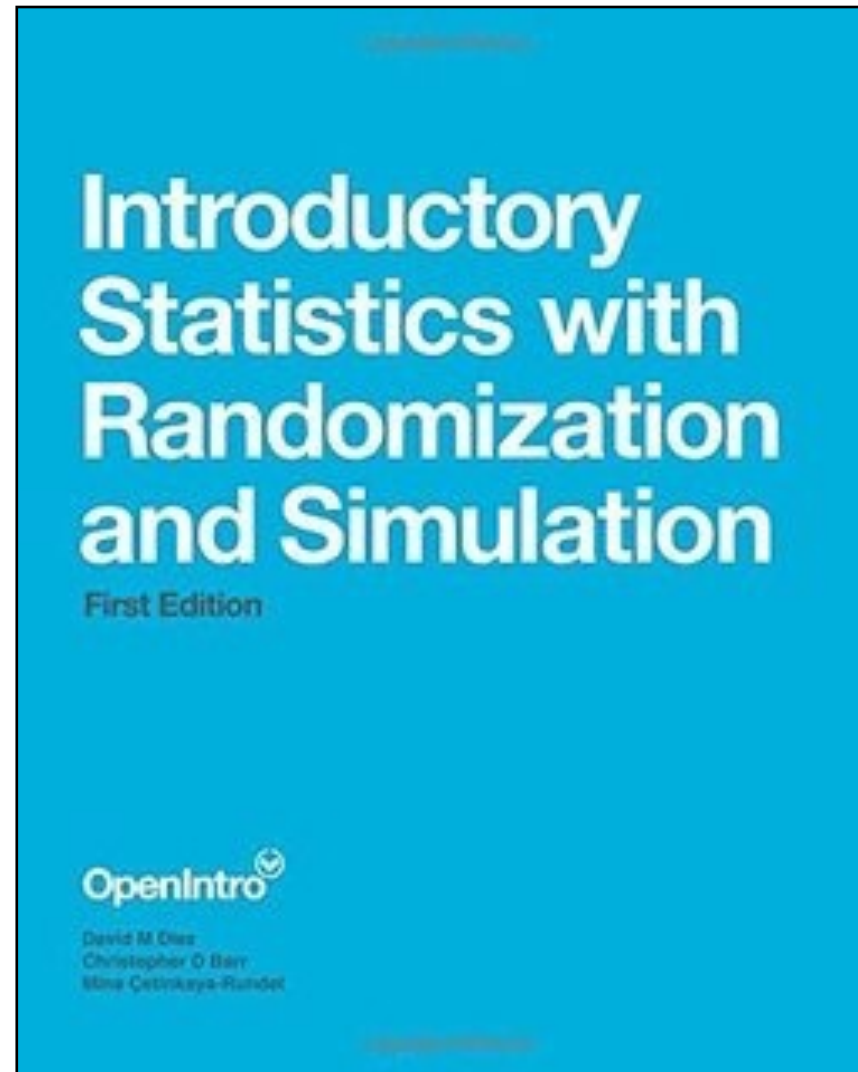
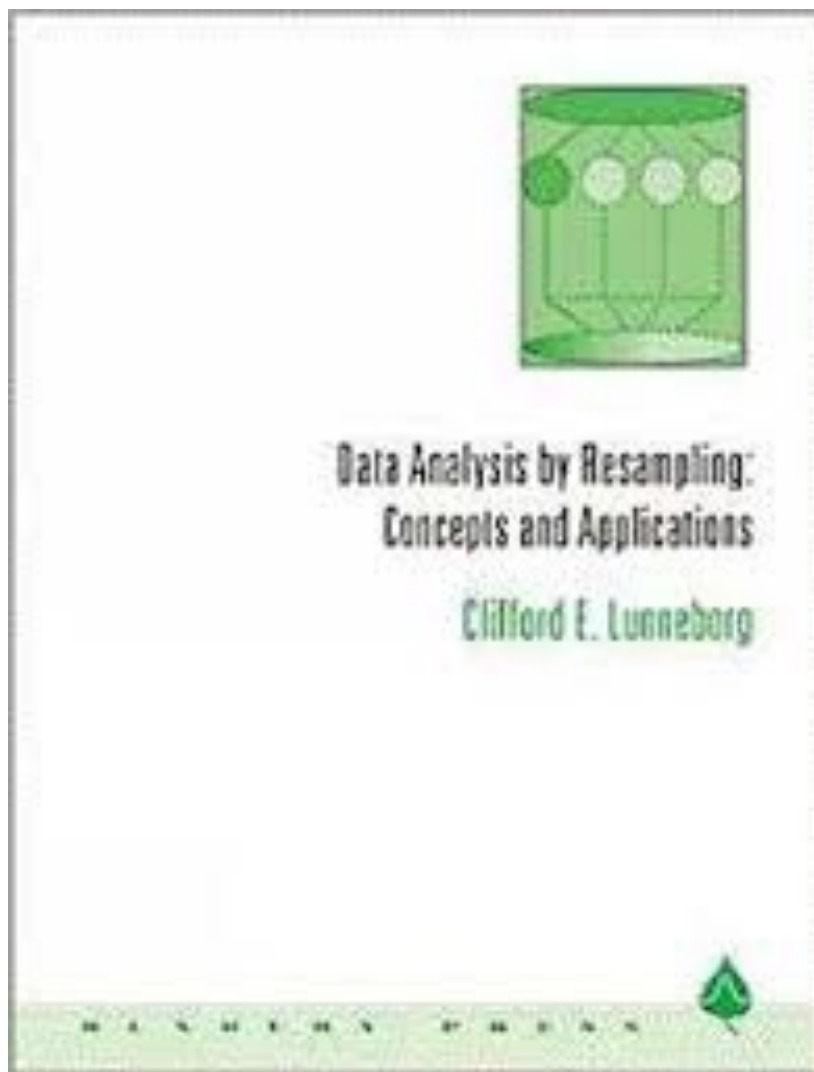


Flexible plot creation

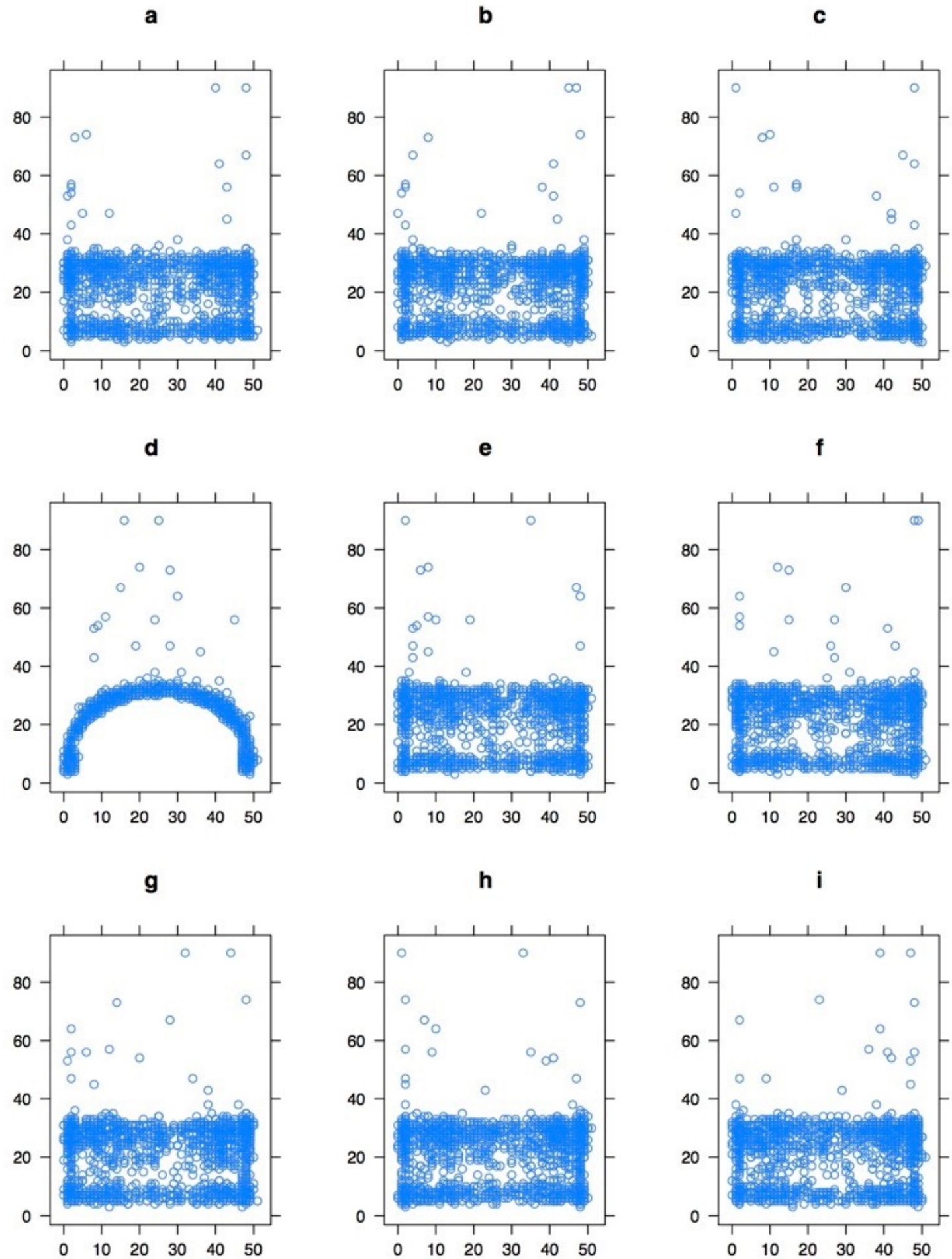
- rather than a few out-of-the-box visualizations, a user should be able to build their own
- perhaps following the Grammar of Graphics (like ggplot2 and d3.js)

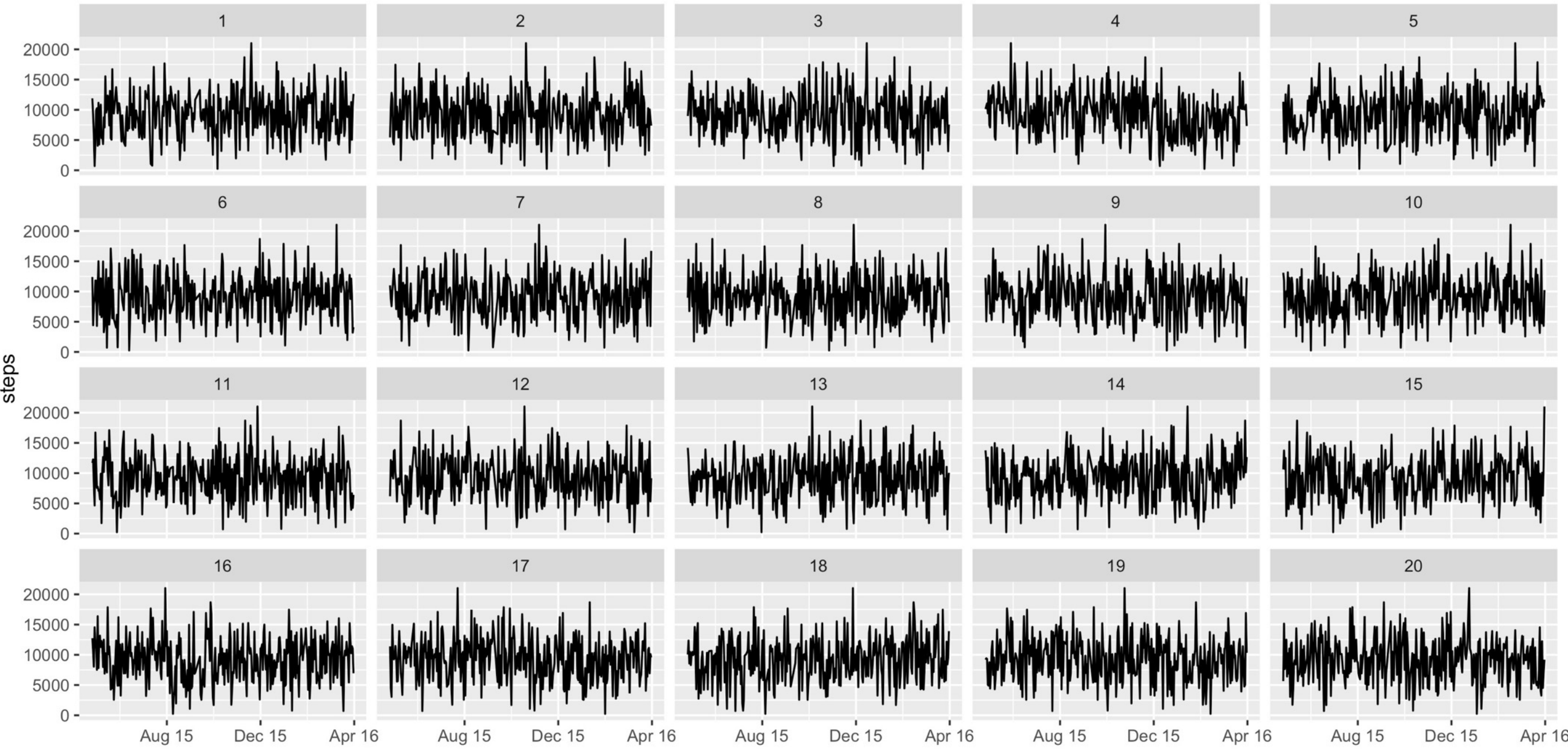


Support for randomization and the bootstrap

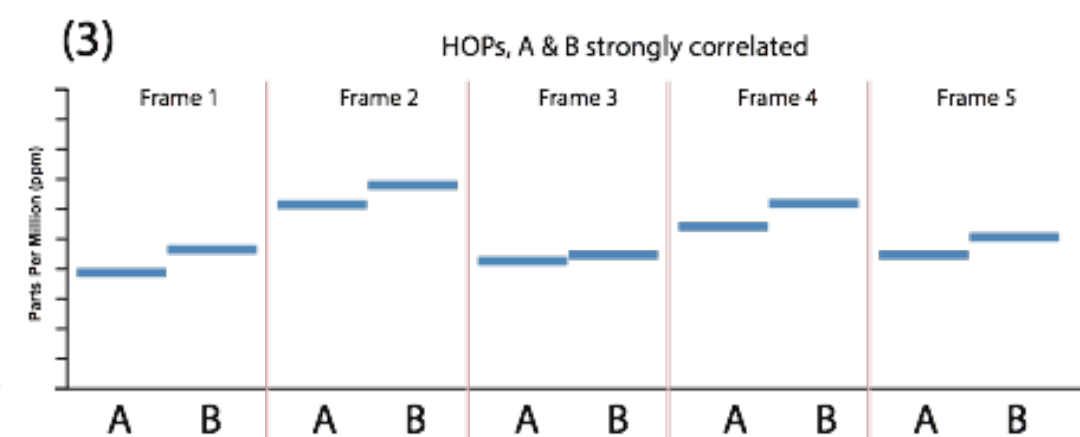
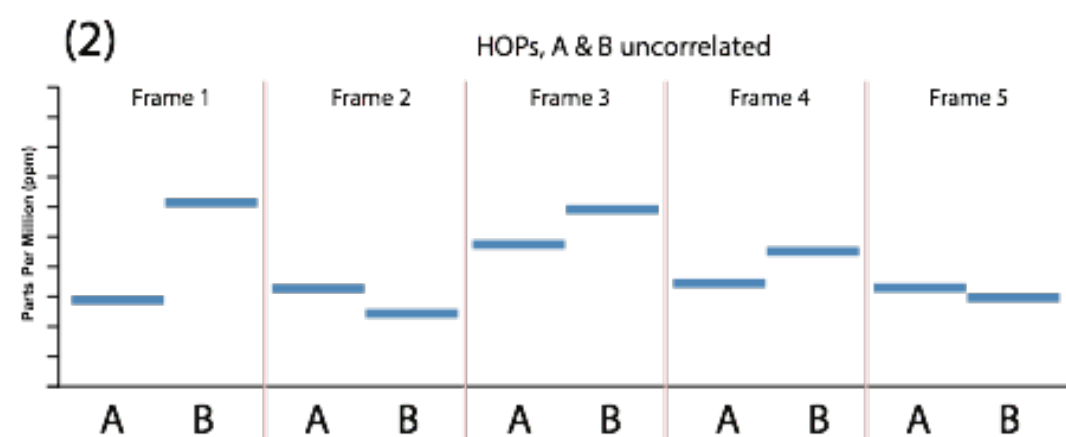
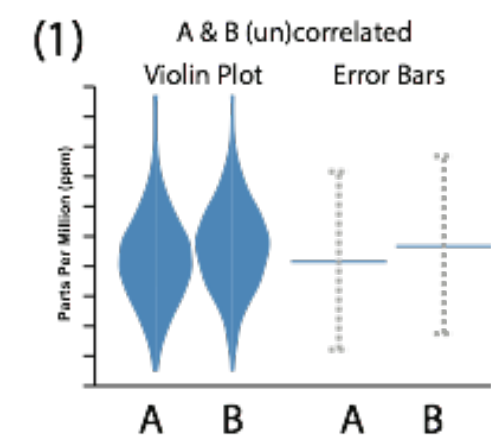
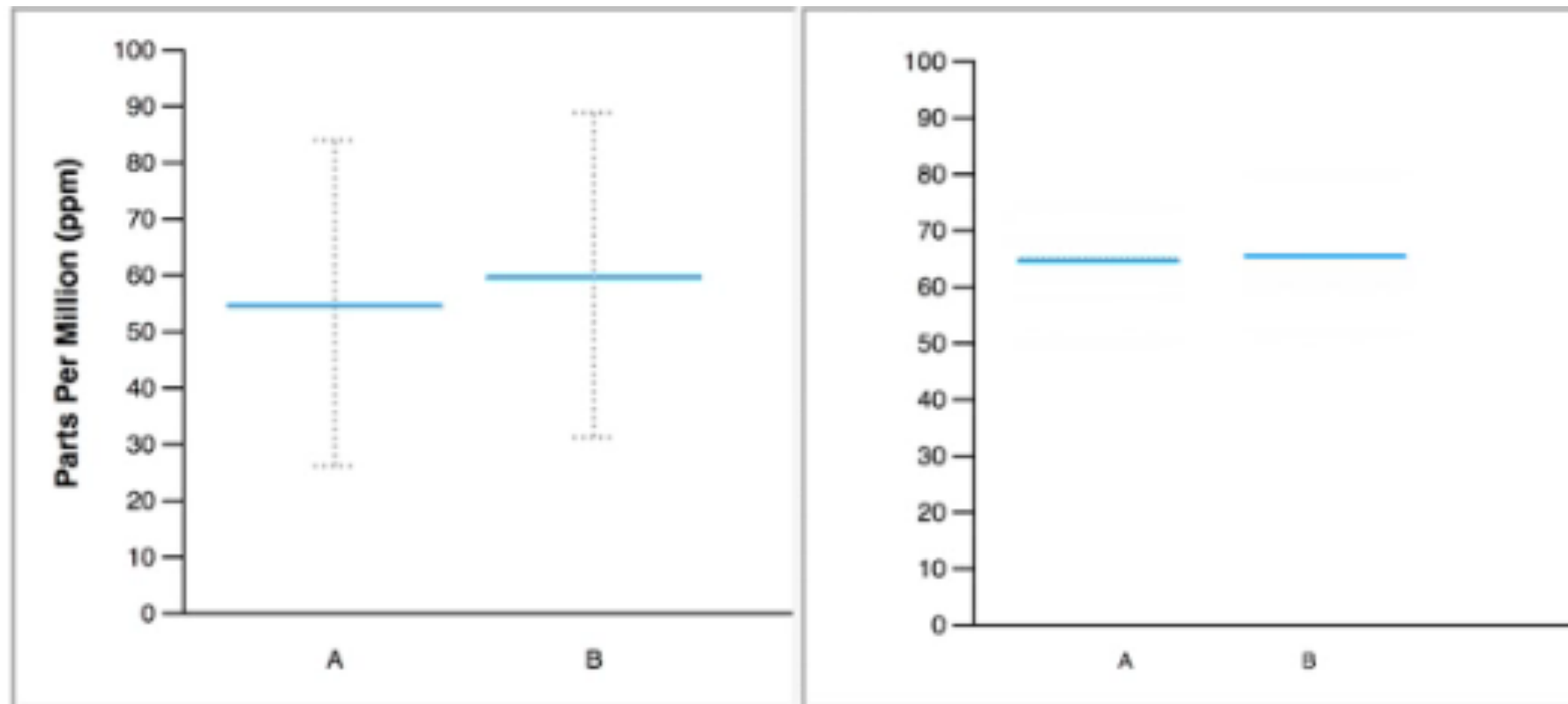


Permuted graphics





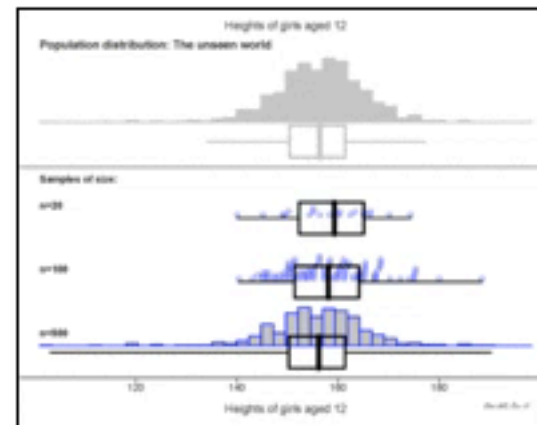
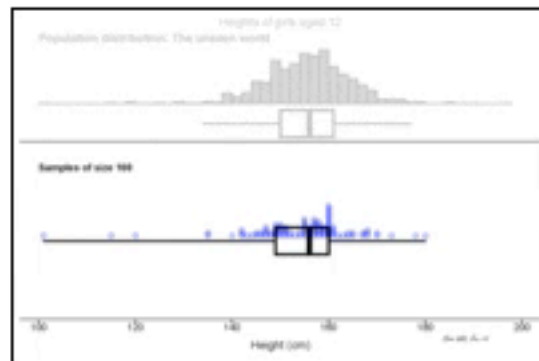
Jessica Hullman, Paul Resnick, Eytan Adar. (2015). Hypothetical Outcome Plots Outperform Error Bars and Violin Plots for Inferences About Reliability of Variable Ordering. *PLOS ONE*, 10(11).
<http://bit.ly/HypotheticalOutcomePlots>



Making the Call

Chris Wild, Nick Horton, Maxine Pfannkuch, Matt Regan

One Population



Animated Gifs

Click for: [n=30](#) [n=100](#) [n=300](#)

[View Full Size GIF](#)

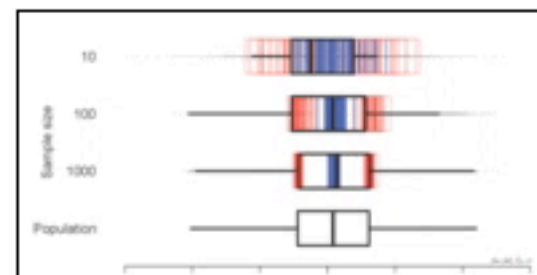
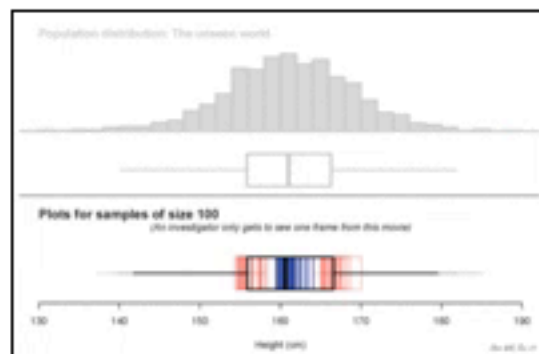
PDFs

Click for: [n=30](#) [n=100](#) [n=300](#)

[View Full size PDF](#)

1(a)

1(b)



Animated Gifs

Click for: [n=30](#) [n=100](#) [n=300](#)

[View full size GIF](#)

PDFs

Click for: [n=30](#) [n=100](#) [n=300](#)

[View full size PDF](#)

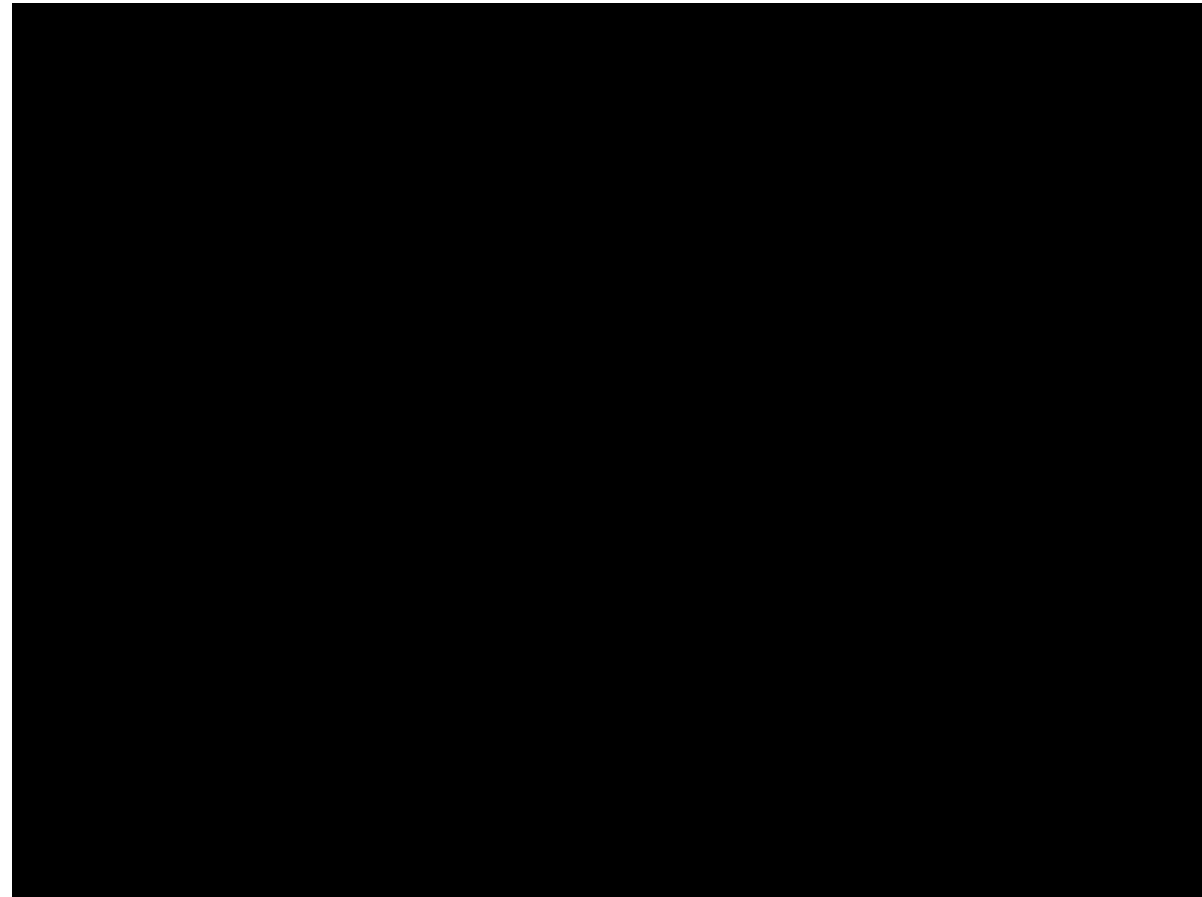
2(a)

2(b)

Interactivity at every level

- when developing an analysis
- when adjusting parameters in an analysis
- when a reader is exploring the analysis

prim9



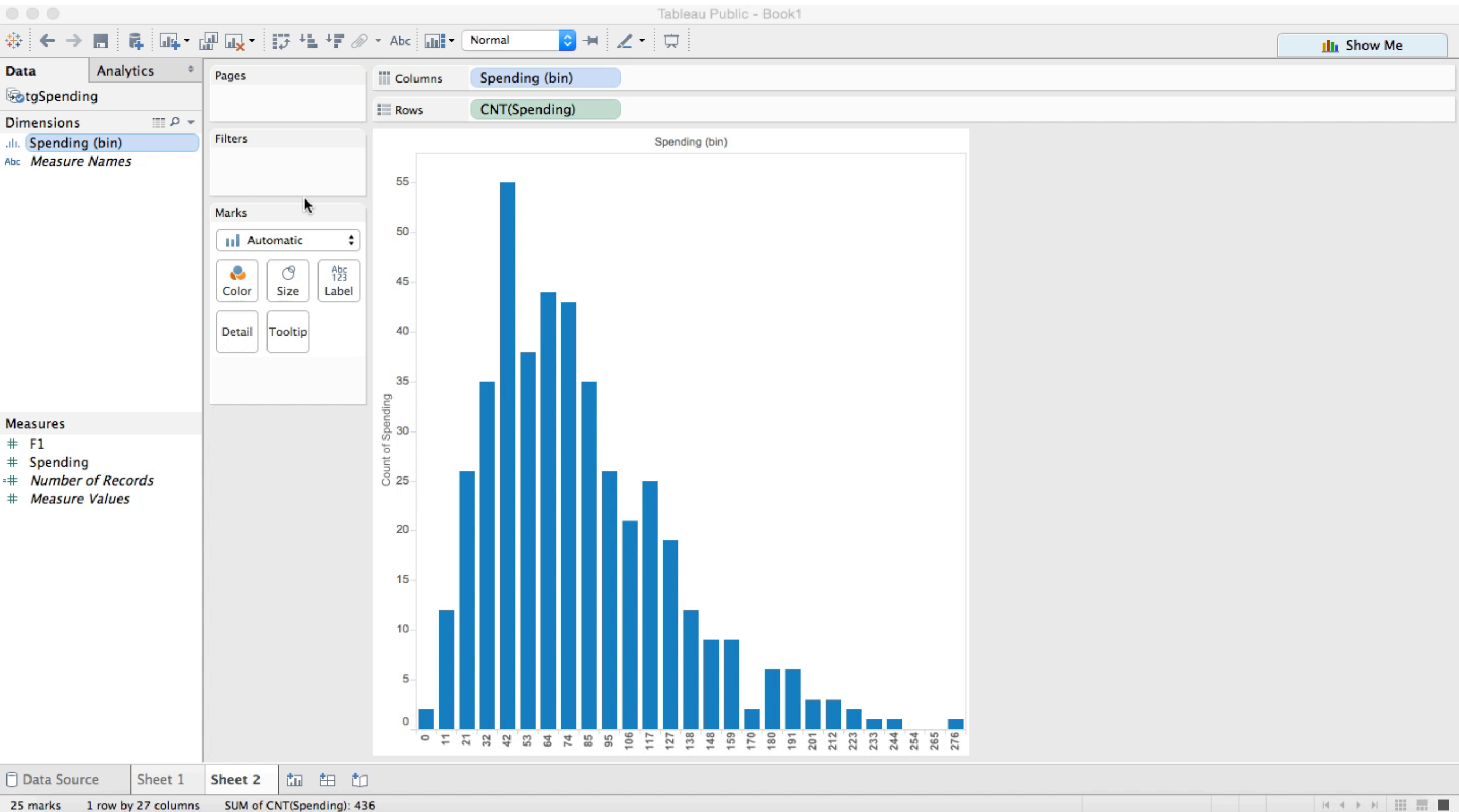
http://bit.ly/prim_9

Fathom

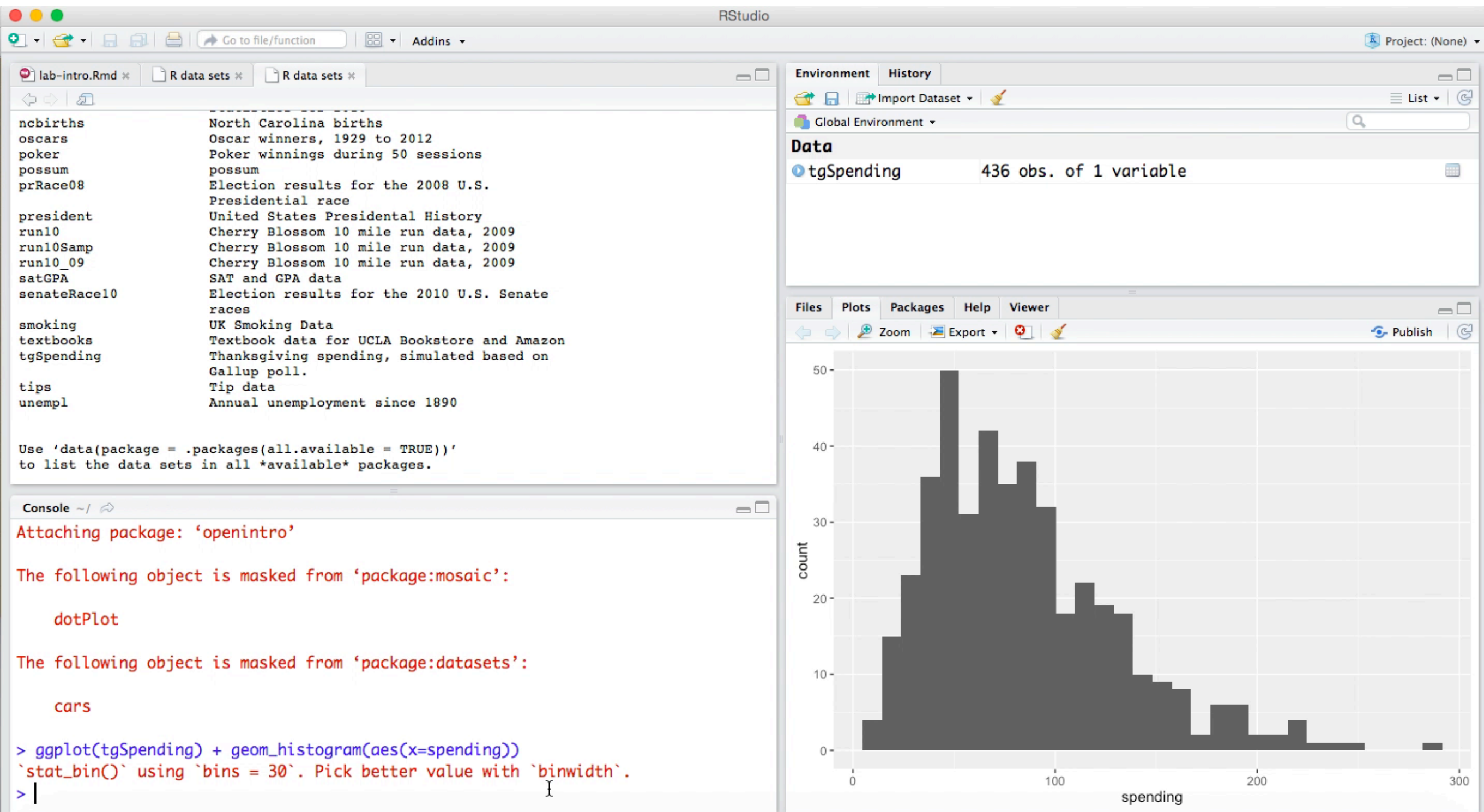
The screenshot displays the Fathom software interface. At the top, there is a toolbar with icons for various data manipulation tools: Collection, Table, Graph, Summary, Estimate, Test, Model, Slider, Meter, and Text. The main workspace shows a file named 'tgSpending.csv' open as a table. The table has four columns: an unlabeled column, 'Attr1', 'Attr2', and '<new>'. The data rows are numbered 429 through 436. The 'Attr2' column contains numerical values, and the '<new>' column is currently empty.

	Attr1	Attr2	<new>
429	429	50.1943	
430	430	41.7233	
431	431	203.553	
432	432	92.1924	
433	433	52.1701	
434	434	21.2527	
435	435	66.7804	
436	436	149.905	

Tableau



R/ggplot2



The screenshot displays the RStudio interface. The top-left pane shows a list of available data sets with their descriptions. The bottom-left pane shows the R console with the following text:

```
Attaching package: 'openintro'

The following object is masked from 'package:mosaic':

  dotPlot

The following object is masked from 'package:datasets':

  cars

> ggplot(tgSpending) + geom_histogram(aes(x=spending))
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> |
```

The top-right pane shows the Environment and History tabs, with the Data tab displaying 'tgSpending' (436 obs. of 1 variable). The bottom-right pane shows a histogram of the 'tgSpending' variable, with the x-axis labeled 'spending' and the y-axis labeled 'count'. The histogram shows a right-skewed distribution of spending values, with a peak count of 50 for a spending value between 40 and 60.

manipulate

The screenshot displays the RStudio interface with three main panels: the source editor, the environment/history pane, and the viewer pane.

Source Editor: Contains R code for creating interactive histograms using the `manipulate` function. The code is as follows:

```
1 manipulate(  
2   ggplot(tgSpending) + geom_histogram(aes(x=spending), binwidth=x),  
3   x = slider(0,100, initial=10)  
4 )  
5  
6  
7 manipulate(  
8   histogram(tgSpending$spending, breaks=slider(0,20, initial=10))  
9 )  
10  
11  
12 manipulate(  
13   histogram(tgSpending$spending, breaks=x),  
14   x = slider(4,20)  
15 )
```

Environment/History Pane: Shows the `Global Environment` with a variable `tgSpending` containing 436 observations of 1 variable.

Viewer Pane: Displays a histogram of the `spending` variable. The x-axis is labeled `spending` and ranges from 0 to approximately 300. The y-axis is labeled `count` and ranges from 0 to 50. The histogram shows a right-skewed distribution with a peak count of approximately 50 for spending values between 40 and 60.

Console: Shows the execution of the code from the source editor, with the following output:

```
> manipulate(  
+   histogram(tgSpending$spending, breaks=x),  
+   x = slider(4,20)  
+ )  
> manipulate(  
+   ggplot(tgSpending) + geom_histogram(aes(x=spending), binwidth=x),  
+   x = slider(0,100, initial=10)  
+ )  
> manipulate(  
+   ggplot(tgSpending) + geom_histogram(aes(x=spending), binwidth=x),  
+   x = slider(0,100, initial=10)  
+ )  
>
```

Gather your data

A histogram is based on a collection of data about a numeric variable. Our first step is to gather some values for that variable. The initial dataset we will consider consists of fuel consumption (in miles per gallon) from a sample of car models available in 1974 (yes, rather out of date). We can visualize the dataset as a pool of items, with each item identified by its value—which in theory lets us "see" all the items, but makes it hard to get the gestalt of the variable. What are some common values? Is there a lot of variation?

Sort into an ordered list

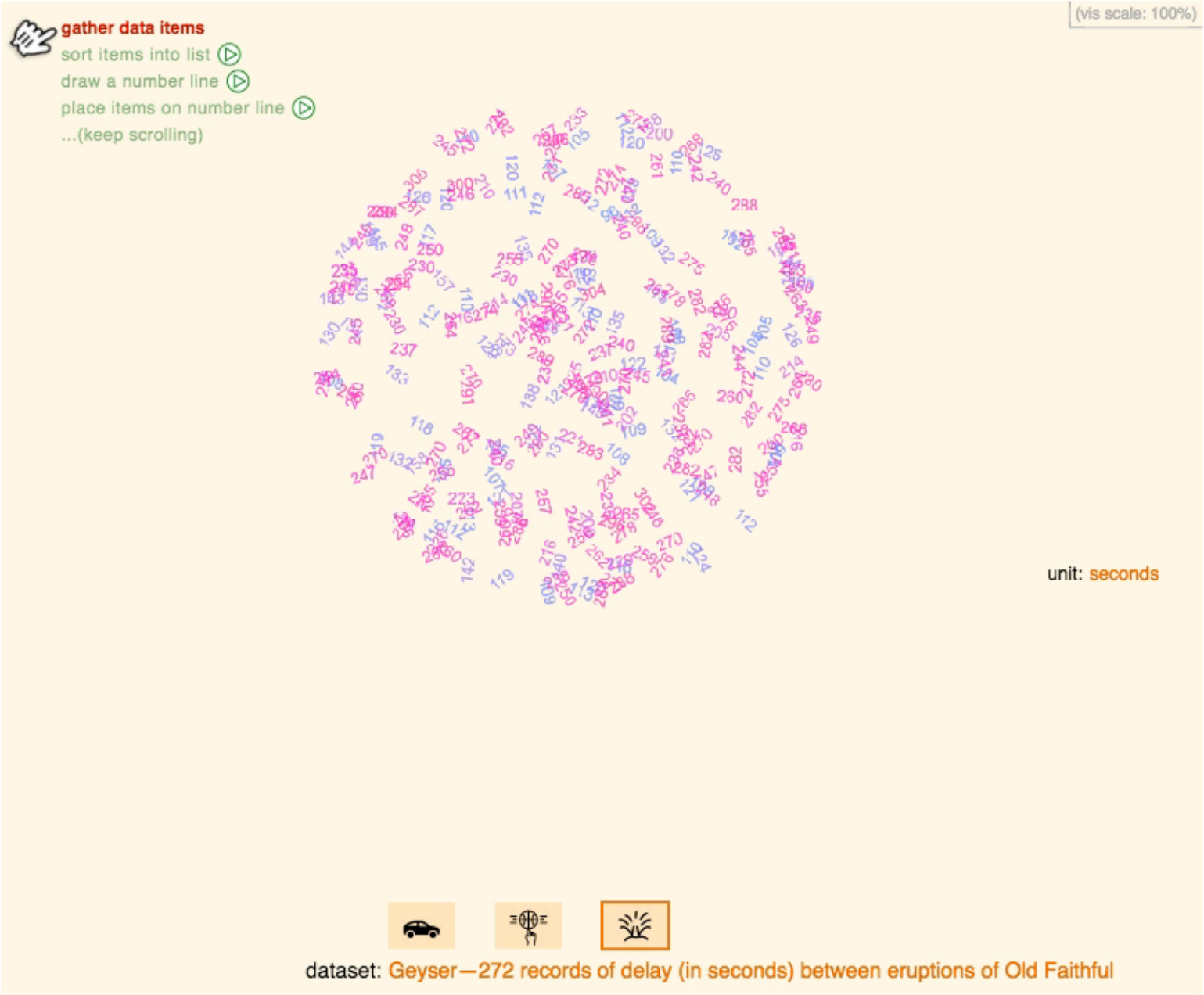
A useful first step towards describing the variable's distribution is to sort the items into a list. Now we can see the maximum value and the minimum value. Beyond that, it is hard to say much about the center, shape, and spread of the distribution. Part of the problem is that the list is completely filled; the space between any two items is the same, no matter how dissimilar their values may be. We need a way to see how the items relate to each other. Are they clustered around a few specific values? Is there one lonely item, with a value far removed from all the others?

Draw the number line

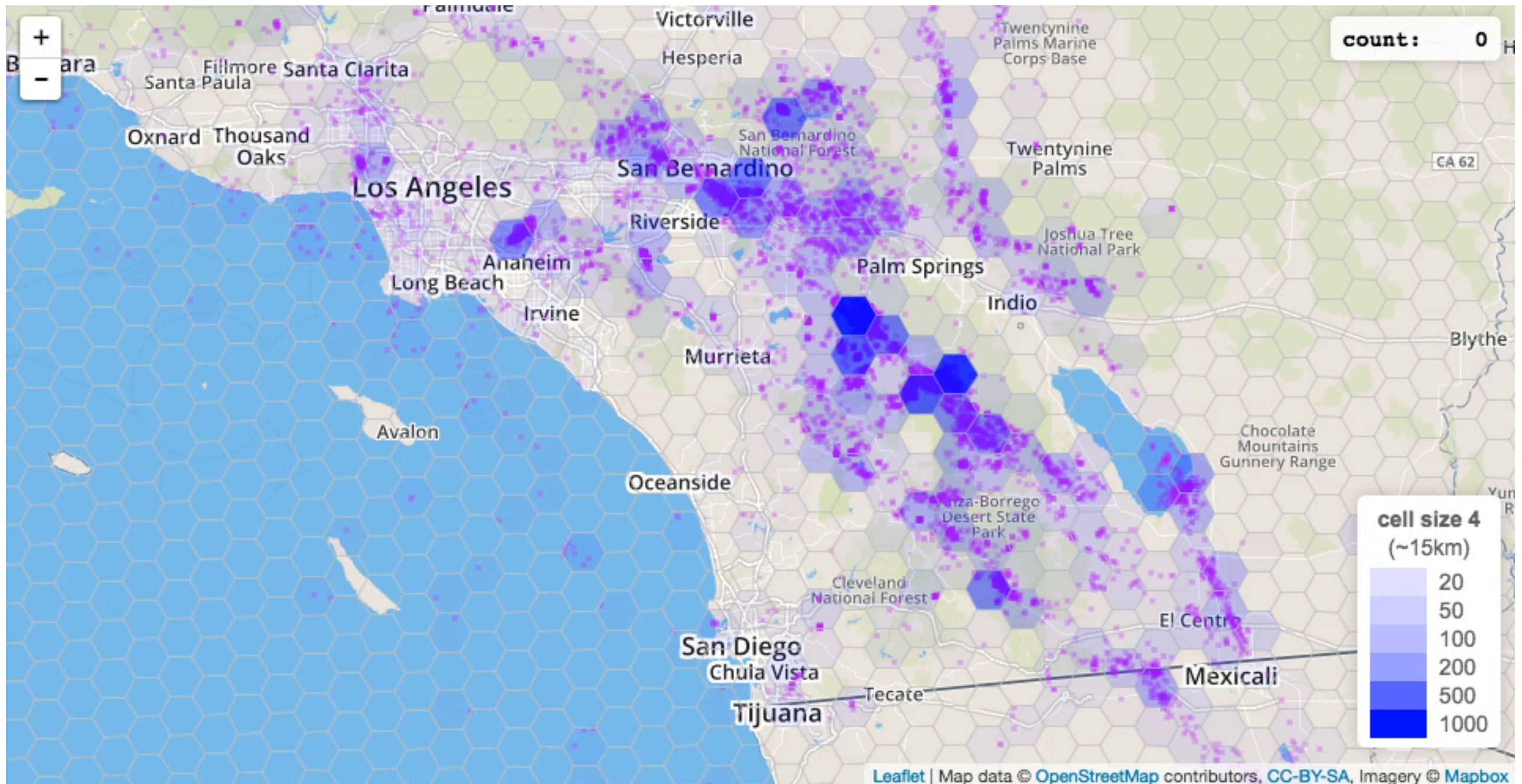
A common convention is to use a number line, on which higher values are displayed to the right and smaller (or negative) values to the left. We can draw a line representing all possible numbers between the minimum and maximum data values.

Add data to the number line

Now, we map each item to a dot at the appropriate point along the number line. In our visualization we draw the path followed by each item on its way from the list to the line, helping to reveal how adjacent list items end up close or far apart on the number line



Spatial aggregation toy



http://bit.ly/spatial_agg

Auditable products

Choose a Ranking (choose a weighting or make your own)

IEEE Spectrum

Trending

Jobs

Open

Custom

Edit Ranking

Add a Comparison

Language Types (click to hide)



Web



Mobile



Enterprise



Embedded

Language Rank

Types

Spectrum Ranking

1. Java



100.0

2. C



99.2

3. C++



95.5

4. Python



93.4

5. C#



92.2

6. PHP



84.6

7. Javascript



84.3

8. Ruby



78.6

9. R



74.0

10. MATLAB



72.6

Show Extended Ranking

Auditable products

Choose a Ranking (choose a weighting or make your own)

IEEE Spectrum

Trending

Jobs

Open

Custom

Edit Ranking

Add a Comparison

Language Types (click to hide)

Web

Mobile

Enterprise

Embedded

The ranking is calculated using 12 weighted data sources. Click a data source to toggle its inclusion in the ranking and drag its slider to reweight it.

Google (search)

50

Google (trends)

50

Github (active)

50

Github (created)

30

Stack Overflow (?s)

30

Stack Overflow (views)

30

Reddit

20

Hacker News

20

Career Builder

5

Dice

5

Topsy

20

IEEE Xplore

100

Cancel

Save as Custom

Inherent documentation

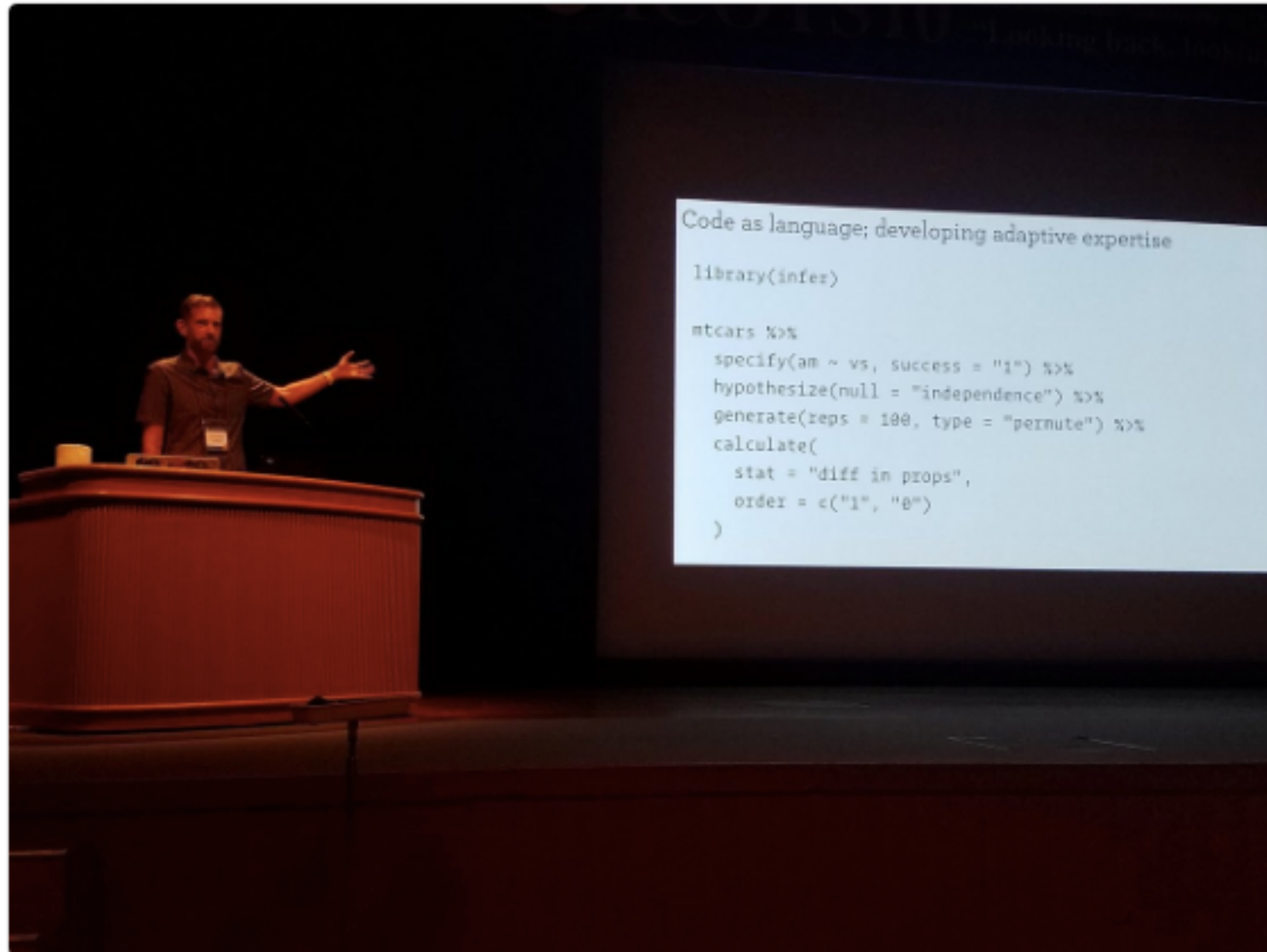
- The system or language should show or tell you what each function does
- Computing tools should “highlight the logic of what is going on” (Kaplan).



Amelia McNamara
@AmeliaMN



"Code as language" -@hadleywickham
#icots10 🥰🥰



12:26 AM - 10 Jul 2018

30 Retweets 129 Likes



Support for narrative, publishing and reproducibility

- narrative and code should be intermixed
- publishing should support others reading your work
- reproducibility should be encouraged



Amelia McNamara

@AmeliaMN



.@xieyihui has an evil exercise that seems torn from @kwbroman's emails: students do analysis, then send them the updated data.
#JSM2016

9:15 AM - 3 Aug 2016

1 Retweet 4 Likes



RMarkdown

The screenshot displays the RStudio interface with a document titled "1-example.Rmd". The editor shows the following RMarkdown code:

```
1 ---
2 title: "Viridis Demo"
3 output: html_document
4 ---
5
6 ```{r include = FALSE}
7 library(viridis)
8 ```
9
10 The code below demonstrates two color palettes in the
11 [viridis](https://github.com/sjmgarnier/viridis) package. Each
12 plot displays a contour map of the Maunga Whau volcano in
13 Auckland, New Zealand.
14
15 ```{r}
16 image(volcano, col = viridis(200))
17 ```
18
19 ## Magma colors
20
21 ```{r}
22 image(volcano, col = viridis(200, option = "A"))
23 ```
```

The rendered output on the right shows a document titled "Viridis Demo". It contains the following text:

The code below demonstrates two color palettes in the [viridis](https://github.com/sjmgarnier/viridis) package. Each plot displays a contour map of the Maunga Whau volcano in Auckland, New Zealand.

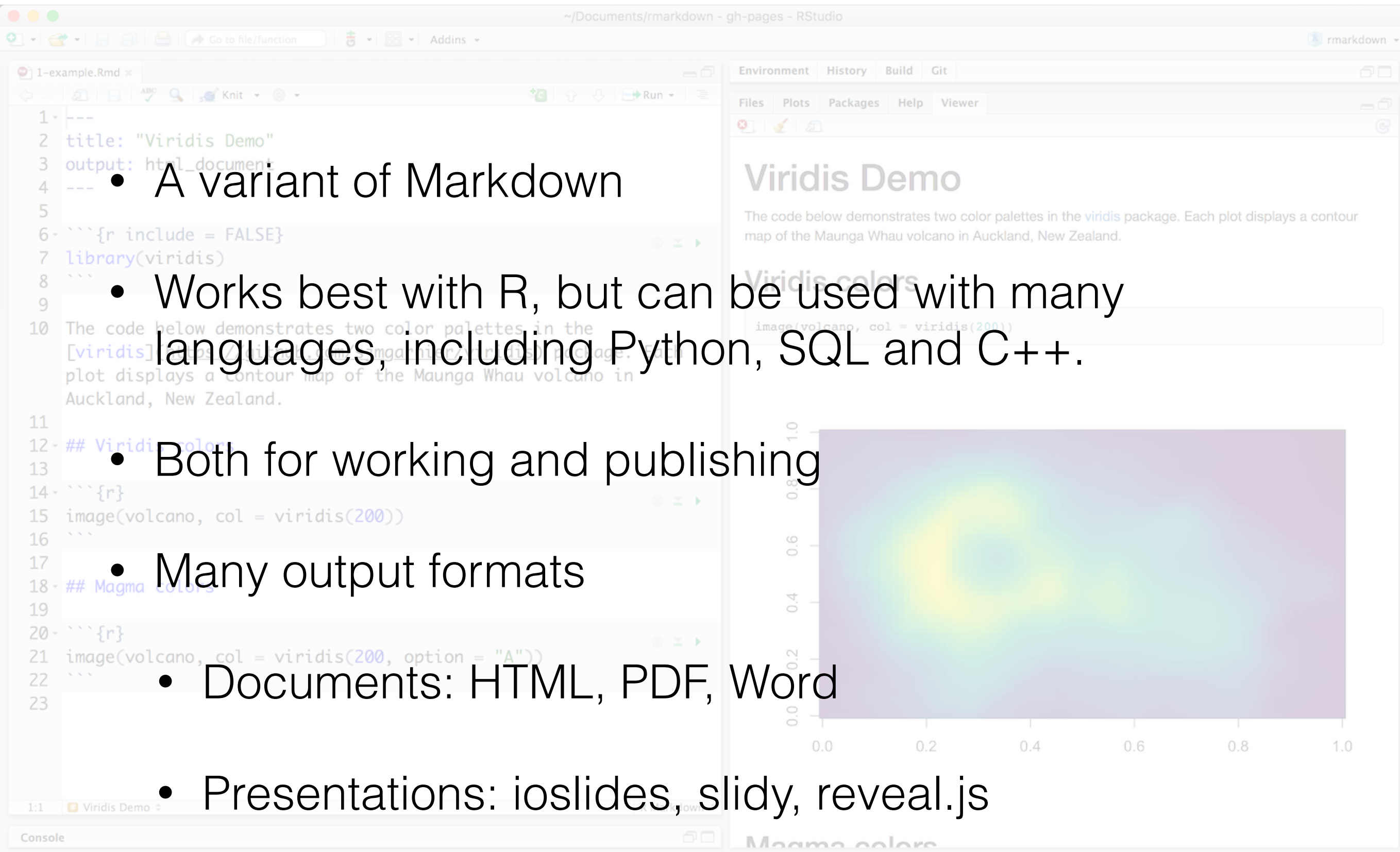
Viridis colors

```
image(volcano, col = viridis(200))
```

The plot shows a contour map of the Maunga Whau volcano, rendered using the viridis color palette. The x and y axes both range from 0.0 to 1.0. The plot displays a central peak (yellow) surrounded by concentric rings of green and blue, indicating elevation contours.

Magma colors

RMarkdown

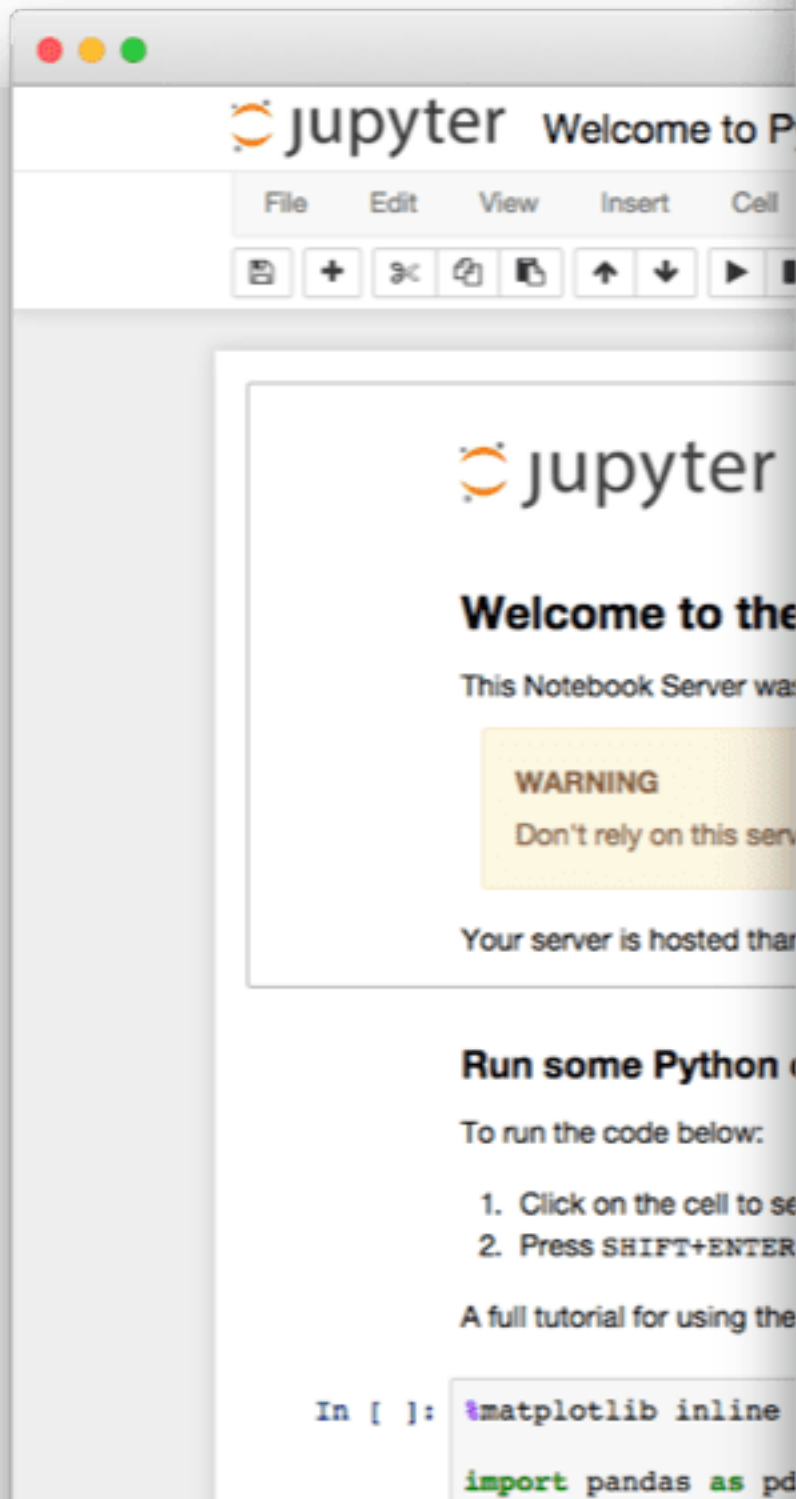


The screenshot shows the RStudio interface with a file named '1-example.Rmd' open. The editor contains R code for loading the 'viridis' package and plotting a contour map of the Maunga Whau volcano. The rendered output on the right shows the title 'Viridis Demo', a paragraph of text, and a contour plot using the viridis color palette. The plot has x and y axes ranging from 0.0 to 1.0. The code in the editor includes comments for 'Viridis colors' and 'Magma colors'.

```
1 ---  
2 title: "Viridis Demo"  
3 output: html_document  
4 ---  
5  
6 ```{r include = FALSE}  
7 library(viridis)  
8 ```  
9  
10 The code below demonstrates two color palettes in the  
11 [viridis] package. Each plot displays a contour  
12 plot displays a contour map of the Maunga Whau volcano in  
13 Auckland, New Zealand.  
14  
15 ## Viridis colors  
16 ```{r}  
17 image(volcano, col = viridis(200))  
18 ```  
19  
20 ## Magma colors  
21 ```{r}  
22 image(volcano, col = viridis(200, option = "A"))  
23 ```
```

- A variant of Markdown
- Works best with R, but can be used with many languages, including Python, SQL and C++.
- Both for working and publishing
- Many output formats
 - Documents: HTML, PDF, Word
 - Presentations: ioslides, slidy, reveal.js

Jupyter notebooks



A screenshot of a Jupyter notebook titled "Exploring the Lorenz System". The notebook is running Python 3. The title bar shows "jupyter Lorenz Differential Equations (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. The toolbar shows various icons for adding, deleting, and running cells. The main content of the notebook is as follows:

Exploring the Lorenz System

In this Notebook we explore the [Lorenz system](#) of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0., 360.),
                 sigma=(0.0, 50.0), beta=(0., 5), rho=(0.0, 50.0))
```

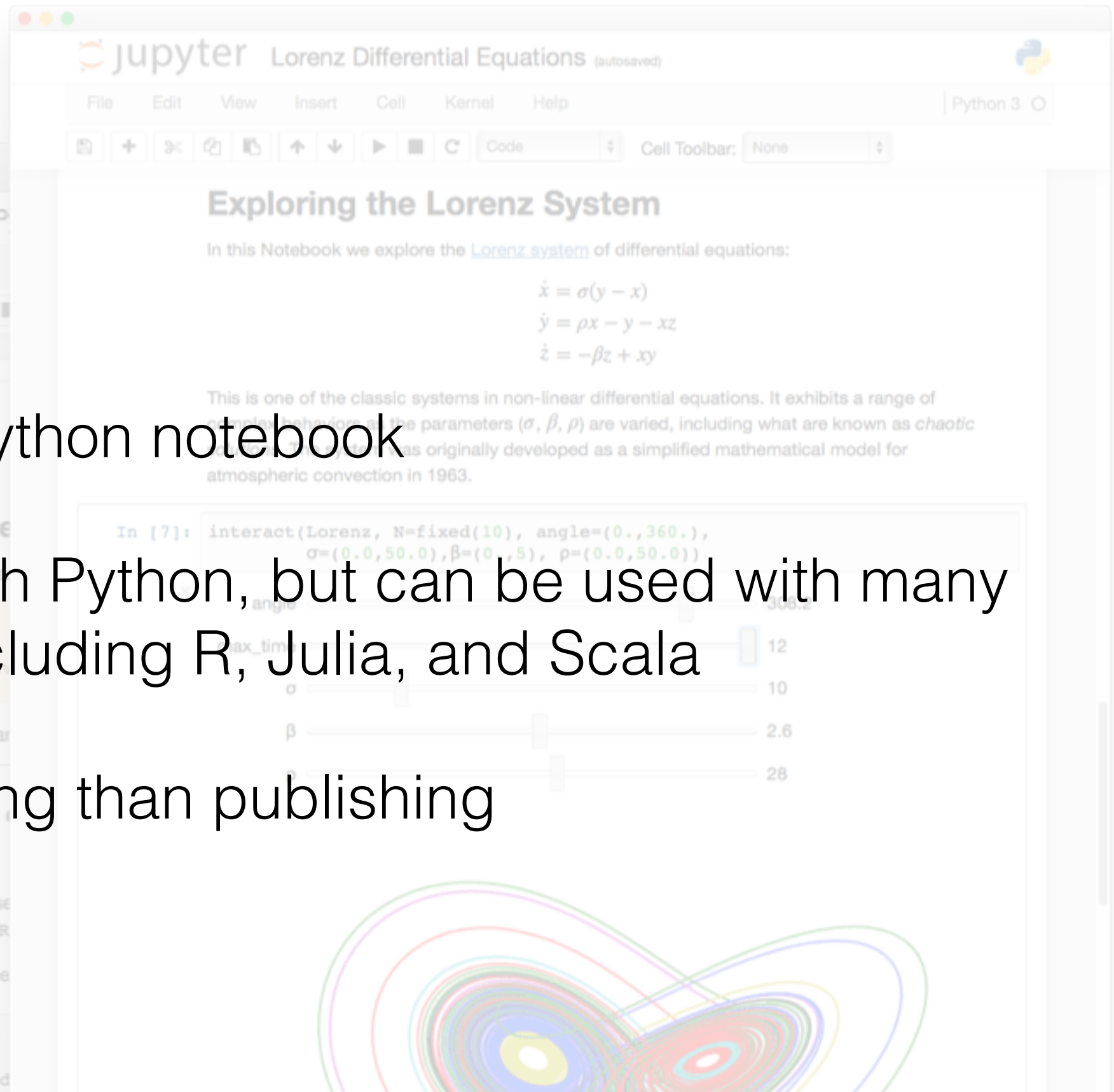
The notebook displays a set of interactive sliders for the parameters σ , β , ρ , and angle . The current values are:

- angle: 308.2
- max_time: 12
- σ : 10
- β : 2.6
- ρ : 28

Below the sliders, a 3D plot shows the Lorenz attractor, a complex, chaotic trajectory in a 3D space, rendered with multiple overlapping colored lines.

Jupyter notebooks

- Grew out of iPython notebook
- Works best with Python, but can be used with many languages, including R, Julia, and Scala
- More for working than publishing





Our path to better science in less time using open data science tools

Julia Stewart Lowndes, et al. Nature Ecology & Evolution v1.
<https://www.nature.com/articles/s41559-017-0160>

We thought we were doing reproducible science. For the first global OHI assessment in 2012 we employed an approach to reproducibility that is standard to our field, which focused on scientific methods, not data science methods. Data from nearly one hundred sources were prepared manually—that is, without coding, typically in Microsoft Excel—which included organizing, transforming, rescaling, gap-filling and formatting data. Processing decisions were documented primarily within the Excel files themselves, e-mails, and Microsoft Word documents. We programmatically coded models and meticulously documented their development, (resulting in the 130-page supplemental materials), and upon publication we also made the model inputs (that is, prepared data and metadata) freely available to download. This level of documentation and transparency is beyond the norm for environmental science.

Our path to better science in less time using open data science tools. Julia Stewart Lowndes, et al. Nature Ecology & Evolution v1. <https://www.nature.com/articles/s41559-017-0160>

We decided to base our work in R and RStudio for coding and visualization, Git for version control, GitHub for collaboration, and a combination of GitHub and RStudio for organization, documentation, project management, online publishing, distribution and communication.

Data preparation: coding and documenting. Our first priority was to code all data preparation, create a standard format for final data layers, and do so using a single programmatic language, R. Code enables us to reproduce the full process of data preparation, from data download to final model inputs, and a single language makes it more practical for our team to learn and contribute collaboratively. We code in R and use RStudio to power our workflow because it has a user-friendly interface and built-in tools useful for coders of all skill levels, and, importantly, it can be configured with Git to directly sync with GitHub online (See ‘Collaboration’).

Sharing methods and instruction. We use R Markdown not only for data preparation but also for broader communication. R Markdown files can be generated into a wide variety of formatted outputs, including PDFs, slides, Microsoft Word documents, HTML files, books or full websites.

Our path to better science in less time using open data science tools. Julia Stewart Lowndes, et al. Nature Ecology & Evolution v1. <https://www.nature.com/articles/s41559-017-0160>

Flexibility to build extensions

- Biehler called this the "closed microworld problem"
- CS ed literature calls this a "high ceiling"

Extensible—



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

features 12744 available packages.

Available Packages

Currently, the CRAN package repository features 12744 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual [R Installation and Administration](#) (also contained in the R base sources) explains the process in detail.

[CRAN Task Views](#) allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 36 views are available.

Package Check Results

All packages are tested regularly on machines running [Debian GNU/Linux](#), [Fedora](#), OS X, Solaris and Windows.

The results are summarized in the [check summary](#) (some [timings](#) are also available). Additional details for Windows checking and building can be found in the [Windows check summary](#).

Writing Your Own Packages

The manual [Writing R Extensions](#) (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Repository Policies

The manual [CRAN Repository Policy \[PDF\]](#) describes the policies in place for the CRAN package repository.

Related Directories

[Archive](#)

Previous versions of the packages listed above, and other packages formerly available.

[Orphaned](#)

Packages with no active maintainer, see the corresponding [README](#).

<https://cran.r-project.org/>

Extensible— GP



About



Frequently Asked Questions

Team and Credits

GP is a free, general-purpose blocks language that is powerful yet easy to learn.

GP can:

- generate high-quality graphics computationally
- manipulate images and sounds
- analyze text files or CSV data sets
- simulate physical, biological, or economic systems
- access the web and use cloud data
- connect to hardware via the serial port
- deploy projects on the web or as stand-alone apps

About

Gallery

Run GP!

Resources

Docs

Forum

Download

<https://gpblocks.org/about/>



Looking into the future

Trifacta

The screenshot shows the Trifacta interface with a data grid. The grid has 24 columns and 345 rows. The columns are labeled with city names: AZ_Phoenix, CA_Los_Angeles, CA_San_Diego, and CA_San_Francisco. Each column has a histogram and a range of values. A blue modal titled 'The Transformer' is open in the center, providing instructions on how to use the transformer tool. The modal includes a 'Next' button and a 'Don't show me any helpers' link.

24 Columns 345 Rows 2 Data Types Grid Filter in grid Generate

##	AZ_Phoenix	##	CA_Los_Angeles	##	CA_San_Diego	##	CA_San_Francisco	##
65 - 228	PHXR-SA	59 - 273	LXXR-SA	47 - 219	SFXR-SA	47 - 16	DNXR	
		59.43		46.96		50.3		
		59.89		47.3		50.1		
		60.4		47.84		50.3		
		61.32		47.98		50.7		
		62.03		48.31		50.7		
		62.78		48.61		50.4		
		63.46		49.08		50.2		
		64.13		49.54		50.1		

The Transformer 1 of 5

This is **the Transformer**, where you can transform your messy data into clean data.

The Transformer is populated with a sample of your dataset. Let's discover what's in your data!

[Learn more](#)

[Don't show me any helpers](#) [Next](#)

<https://www.trifacta.com/>

Exploratory

EXPLORATORY

Features Pricing Community Download Resources

日本語



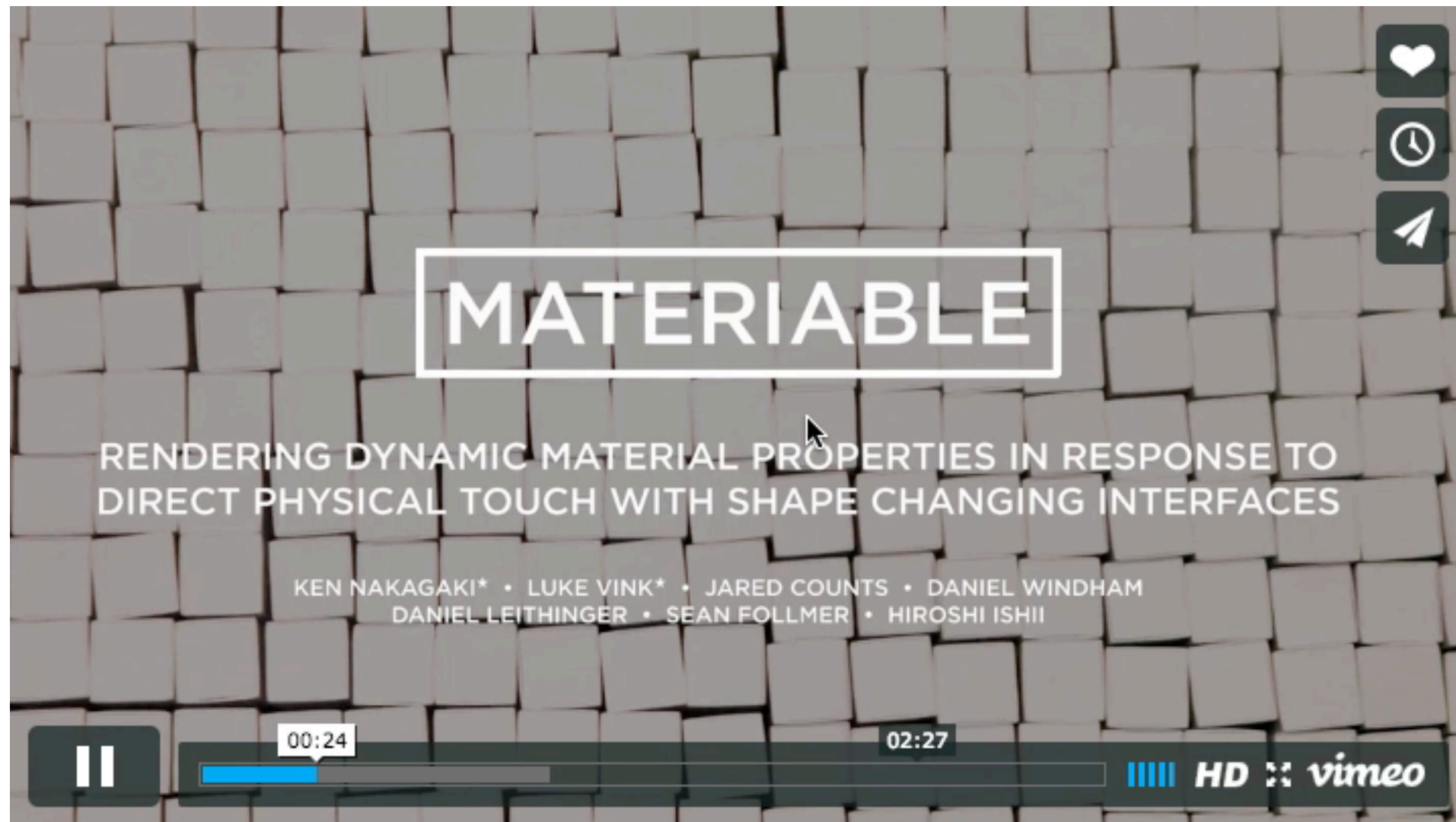
The screenshot displays the Exploratory web application interface. The main window shows a data set named 'airline_delay_2016_09' with 23 columns and 452,558 rows. The interface is divided into several sections:

- Left Sidebar (Analytics):** Lists various data frames including 'ab_test_data_for_prior...', 'ab_test_result', 'Activities', 'activities_dau_mau', 'activities_log', 'airline_delay_2016_09', 'airline_delay_2016_0..', 'airline_delay_2016_0..py', 'airline_delay_2016_0..y2', 'banking', 'cabinet_support', 'GA', 'sales', 'stocks', and 'survival_users'.
- Top Bar:** Shows the current data set name 'airline_delay_2016_09', the last data import date '7/31/2017', and a 'Re-import' button.
- Query Editor:** Contains two mutate operations: `FL_DATE_wday = wday(FL_DATE, label = TRUE, abbr = TRUE)` and `FL_DATE_day = day(FL_DATE)`.
- Summary View:** Displays a grid of charts and summary statistics for columns: FL_DATE (Date), CARRIER (character), FL_NUM (integer), ORIGIN (character), ORIGIN_CITY_NAME (character), ORIGIN_STATE_ABR (factor), DEST (factor), and DEST_CITY_NAME (character). Each chart includes a bar plot and a summary table with statistics like NA, Min, Max, Median, and Average.
- Right Panel (Select, Filter, Mutate, Separate):** Contains a 'Select' section with 'Source' (Local - CSV - airline-delay-20...), a 'Filter' section with `filter(!is.na(ARR_DELAY))`, a 'Mutate' section with `mutate(DEST = fct_lump(DEST, n = 20), is_delayed = ARR_DELAY > 0, ORIGIN_STATE_ABR = fct_lump(ORIGIN_STATE_ABR, n = 20))`, and a 'Separate' section with `separate(DEP_TIME, into = c("dep_hour", "dep_minute"), sep = c(2), remove = TRUE, convert = TRUE)`.

Trusted By

<https://exploratory.io/>





Ken Nakagaki, Luke Vink, Jared Counts, Daniel Windham, Daniel Leithinger, Sean Folder and Hiroshi Ishii. Materiable: Rendering Dynamic Material Properties in Response to Direct Physical Touch with Shape Changing Interfaces CHI 2016. <http://tangible.media.mit.edu/project/materiable/>

WHY A SEEING SPACE?

SOFTWARE-BASED TOOLS ARE TRAPPED IN TINY RECTANGLES.

MODERN PROJECTS HAVE COMPLEX BEHAVIOR.

REAL-WORLD TOOLS ARE IN ROOMS, WHERE WORKERS THINK WITH THEM. BODIES.

THE CHALLENGE IS NOT BUILDING THESE PROJECTS, BUT UNDERSTANDING THEM.

UNDERSTANDING PROVIDES SEEING, AND THE BEST SEEING TOOLS ARE ROOMS.

TODAY'S MAKER SPACES PROVIDE TOOLS FOR BUILDING.

WHAT IS A SEEING SPACE?

1 SEEING INSIDE

2 SEEING ACROSS TIME

3 SEEING ACROSS POSSIBILITIES

COLLECTING DATA

DISPLAYING DATA

CONTROLLING TIME

SEEING ACROSS TIME

AUTOMATIC INTERVIEW

AUTOMATIC EXPERIMENTATION

WHY IS SEEING SO IMPORTANT?

TAKING ENGINEERING SCIENCE

SEEING ACROSS POSSIBILITIES

AUTOMATIC EXPERIMENTATION

Bret Victor. Seeing Spaces.
<http://worrydream.com/#!/SeeingSpaces>



Thank you

Amelia McNamara (@AmeliaMN)

Assistant Professor, Department of Computer & Information Sciences
University of St Thomas, St Paul, MN USA