Enabling Data Science for the Majority

Aditya Parameswaran
Assistant Professor
University of Illinois

http://data-people.cs.illinois.edu
Many many contributors!

- **PIs**: Kevin Chang, Amol Deshpande, Karrie Karahalios, Aaron Elmore, Sam Madden (Spanning Illinois, Chicago, MIT, UMD)
- **PhD Students**: Mangesh Bendre, Akash Das Sarma, Yihan Gao, Silu Huang, Doris Lee, Stephen Macke, Sajjadur Rahman, Tarique Siddiqui, Tana Wattanawaroong, Doris Xin, Liqi Xu
- **MS Students**: Ayush Jain, Vipul Venkataraman, Chao Wang, Ed Xue, Paul Zhou, ...
- **Many Undergrads!**
It was the year 2013 ...

Many of us (the database community) were doing the exact same thing!
The “99%” of Data Analytics Needs

So far, focused on the data analytics needs of the 1%
• Companies w/ **massive data, resources & know-how**

Ignoring the 99%:
• **scientists**
• small business owners
• **statistical analysts**
• **journalists**
• consultants, ...

Our research has been focused on easing the burden of data analytics for the 99%

**So what were their frustrations?**
What about the Needs of the 99%?

The bottleneck is not one of *scale*...
but is actually the “*humans-in-the-loop*”

From “Big data and its Technical Challenges“, CACM 2014

*For big data to fully reach its potential, we need to consider scale not just for the system but also from the perspective of humans. We have to make sure that the end points—humans—can properly “absorb” the results of the analysis and not get lost in a sea of data.*
Need of the hour: Human-In-the-Loop Data Analytics Tools

HILDA tools:
- treat both humans and data as first-class citizens
- reduce human labor
- minimize complexity
A Maslow’s Hierarchy for HILDA

Background: Maslow developed a theory for what motivates individuals in 1943; highly influential
A Maslow’s Hierarchy for HILDA
Browse & Explore: **DataSpread**

DataSpread is a **spreadsheet-database hybrid**:

*Goal: Marrying the flexibility and ease of use of spreadsheets with the scalability and power of databases*

Enables the “99%” with large datasets but limited prog. skills to open, touch, and examine their datasets

[http://dataspread.github.io](http://dataspread.github.io)

[Vldb’15,Vldb’15,ICDE’16]
Play and View: Zenvisage

Zenvisage is **effortless visual exploration tool.**

**Goal: “fast-forward” to visual patterns, trends, without having analyst step through each one individually**

Enables individuals to play with, and extract insights from large datasets at a fraction of the time.

http://zenvisage.github.io
OrpheusDB is a tool for managing dataset versions with a database.

**Goal:** building a versioned database system to reduce the burden of recording datasets in various stages of analysis.

Enables individuals to collaborate on data analysis, and share, keep track of, and retrieve dataset versions.

[http://orpheus-db.github.io](http://orpheus-db.github.io)

(also part of : a collab. analysis system w/ MIT & UMD)

[VLDB’17, SIGMOD’17, VLDB’16,VLDB’15 x 2, TAPP’15, CIDR’15]
This talk

About 10 minutes per system:
overview + architecture + one key technical challenge

Common theme: if you torture databases enough, you can get them to do what you want!
Motivation

Most of the people doing ad-hoc data manipulation and analysis use spreadsheets, e.g., Excel

Why?

• *Easy to use*: direct manipulation
• *Built-in visualization capabilities*
• *Flexible*: schema-free
But Spreadsheets are Terrible!

- **Slow**
  - single change ➔ wait minutes on a 10,000 x 10 spreadsheet
  - can’t even open a spreadsheet with >1M cells
  - speed by itself can prevent analysis

- **Tedious + not Powerful**
  - filters via copy-paste
  - only FK joins via VLOOKUPs; others impossible
  - even simple operations are cumbersome

- **Brittle**
  - sharing excel sheets around, no collab/recovery
  - using spreadsheets for collaboration is painful and error-prone
Let’s turn to Databases

Databases are:
• Slow Scalable
• Tedious + not Powerful Powerful and expressive (SQL)
• Brittle Collaboration, recovery, succinct

So why not use databases?
Well, for the same reason why spreadsheets are so useful:

• Easy to use Not easy to use
• Built-in visualization No built-in visualization
• Flexible Not flexible
Combining the benefits of spreadsheets and databases

Spreadsheet as a frontend interface
Databases as a backend engine

Result: retain the benefits of both!

But it’s not that simple...
## Different Ideologies

<table>
<thead>
<tr>
<th>Feature</th>
<th>Databases</th>
<th>Spreadsheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>Schema-first</td>
<td>Dynamic/No Schema</td>
</tr>
<tr>
<td>Addressing</td>
<td>Tuples with PK</td>
<td>Cells, using Row/Col</td>
</tr>
<tr>
<td>Presentation</td>
<td>Set-oriented, no</td>
<td>Notion of current window, order</td>
</tr>
<tr>
<td></td>
<td>such notion</td>
<td></td>
</tr>
<tr>
<td>Modifications</td>
<td>Must equal queries</td>
<td>Can be done at any granularity</td>
</tr>
<tr>
<td>Computation</td>
<td>Query at a time</td>
<td>Value at a time</td>
</tr>
</tbody>
</table>

Due to this, the integration is not trivial...
First Problem: Representation

Q: how do we represent spreadsheet data?

Dense spreadsheets: represent as tables (Row #, Col1 val, Col2 val, ...)

Sparse spreadsheets: represent as triples (Row #, Column #, Value)
First Problem: Representation

Q: how do we represent spreadsheet data?

Can we do even better than the two extremes? Yes!

Carve out dense areas ➔ store as tables, sparse areas ➔ store as triples
First Problem: Representation

However, even if we only use “tables”, carving out the ideal # partitions (min. storage, modif., access) is \textbf{NP-Hard}

\textbf{\textnormal{\Large \rightarrow Reduction from min. edge-length partition of rectilinear polygons}}

Thankfully, we have a way out...
Solution: Constrain the Problem

A new class of partitionings: recursive decomp.

A very natural class of partitionings!
Solution: Constrain the Problem

The optimal recursive decomposable partitioning can be found in PTIME using DP

- Still quadratic in # rows, columns 😞

- Merge rows/columns with identical signatures
  ~ the time for a single scan
One Sample Result

Up to 30% reduction in storage, 40% reduction in eval time
Initial Progress and Architecture

Hopefully bring spreadsheets to the big data age!

DataSpread

Other Applications
- Sally
- Bob
- Sue

Vanilla SQL
Spreadsheet Formulae
Spreadsheet SQL
New Interface Algebra

Interface Query Processor

Interface Storage Manager
Interface Transaction Manager

Underlying Data
Interface-Embedded Queries
Interface-Aware Indexes

Other Applications

Spreadsheet

SQL

Spreadsheet Formulae

New Interface Algebra

Interface Query Processor

Interface Storage Manager
Interface Transaction Manager

Underlying Data
Interface-Embedded Queries
Interface-Aware Indexes

Top Supplier
- PARAMOUNT: 1800.0
- UNICOM: 3000.0

Supplier Totals

Invoice Report
Sheet2
**Standard Data Visualization Recipe:**

1. **Load** dataset into data viz tool
2. **Start** with a desired hypothesis/pattern
3. **Select** viz to be generated
4. **See** if it matches desired pattern
5. **Repeat** 3-4 until you find a match
Laborious and Time-consuming!

**Key Issue:**
Visualizations can be generated by varying
- data subsets
- visualized attributes

Too many visualizations to look at to find desired visual patterns!
Broadly Applicable

• find keywords with similar CTRs to a specific one
• find solvents with desired properties
• find aspects on which two sets of genes differ
• find supernovae with specific patterns

Common theme: **manual labor** for finding desired patterns to test hypotheses, derive insights
Key Insight: Automation

We can automate that!

Desiderata for automation:

• **Expressive** – specify what you want
• **Interactive** – interact with results, cater to non-programmers
• **Scalable** – get interesting results quickly

Enter Zenvisage:
(zen + envisage: to effortlessly visualize)
Overview
Zenvisage: Two Modes

- **First Mode**: Interactions, drawing, drag-and-drop
  - Simple needs
  - Starting point / context

- **Second Mode**: the Zenvisage Query Language (ZQL)
  - Sophisticated needs
  - Multiple steps

*Can switch back and forth, as user needs evolve*

Both modes developed after many discussions with potential users
ZQL: High Level Overview

ZQL is a viz exploration language

- Captures four key operations on viz collections
  - Compose
  - Filter
  - Compare
  - Sort

- Incorporates data mining primitives

Powerful; formally demonstrated “completeness”
ZQL: A Bird’s Eye View

Output spec and identifiers

Composition of visualizations, often using values from previous steps

Sorting, comparing, and filtering visualizations
Example 1: Comparisons

Find the states where the *soldprice* trend is most similar to (or most different from) the *soldpricepersqft* trend.

⇒ *Comparing a pair of y-axes for different “z”*
Example 1: Comparisons

ZQL Table

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Constraints</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>x1</td>
<td>y1&lt;-'soldprice'</td>
<td>z1&lt;-'state'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>x1</td>
<td>y2&lt;-'soldprice':</td>
<td>z1</td>
<td>v1&lt;-'argmin_{z1}[k=3]DEuclidean(f1,f2)</td>
<td></td>
</tr>
<tr>
<td>f3</td>
<td>x1</td>
<td>y3&lt;-'soldprice', 'sc'</td>
<td>v1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results

- Metro: Prescott (1.00)
- Metro: Miami-Fort Lauderdale (0.92)
- Metro: Ocala (0.91)
- Metro: Medford (0.92)
- Metro: Lake (0.91)

Representative patterns

- Prescott (217)
- Pendleton (71)
- Lynchburg (7)

Outliers

- Monroe
- Lancaster
- Elmira
Example 2: Drill-downs

Find *cities in NY* where the trend for *soldprice* is most different from (or most similar to) the *overall NY trend*.

→ *Comparing across different granularities of “z”*
Example 2: Drill-downs

ZQL Table

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Constraints</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>x1&lt;-'year'</td>
<td>y1&lt;-'soldprice'</td>
<td>z1&lt;-'state'</td>
<td>state='NY'</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>x1</td>
<td>y1</td>
<td>z2&lt;-'city'</td>
<td>state='NY'</td>
<td>v2&lt;argmin_(z2)</td>
</tr>
<tr>
<td>f3</td>
<td>x1</td>
<td>y1</td>
<td>v2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results

soldprice by year

state: NY (1.00)

state: NJ (0.90)

state: FL (0.83)

state: IL (0.79)

state: CT (0.79)

Representative patterns

soldprice by year

AR (19)

WI (17)

GA (2)

Outliers

soldprice by year

NE

MI

IN
Example 3: Explanations/Diffs

Find visualizations on which the *states of CA* and *NY* are most different (or most similar).

→ *Comparing across different “x”, “y” for two “z”*
Example 3: Explanations/Diffs

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Constraints</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>x1&lt;=&quot;</td>
<td>y1&lt;=&quot;</td>
<td>'state'.&quot;CA&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>x1</td>
<td>y1</td>
<td>'state'.&quot;NV&quot;</td>
<td>x2,y2&lt;-argmin_{x1,y1}[k=1]DEuclidean(f1,f2)</td>
<td></td>
</tr>
<tr>
<td>f3</td>
<td>x2</td>
<td>y2</td>
<td>'state'.&quot;CA&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results

- **state: NY (1.00)**
- **state: NJ (0.90)**
- **state: AZ (0.87)**
- **state: FL (0.83)**
- **state: IL (0.79)**
- **state: CT (0.79)**

Representative patterns

- **AR (19)**
- **WI (17)**
- **GA (2)**

Outliers

- **NE**
- **MI**
- **IN**
ZQL Query Execution

Let’s use a relational database as a backend

Naïve translation approach:

For each line of ZQL:
    Issue one SQL query for each combination of X, Y, Z;
    Apply further processing on result

Often 1000s of SQL queries issued per ZQL query!

⇒ wasteful, extremely high latency
SmartFuse: Intelligent Query Optimizer

NP-Hard!

Sequential (99.99%)
Grouped (45%)
Parallel (20%)
Speculation (20%)
SmartFuse
User Study Takeaways (20 Participants)

Faster  \[ \mu = 115s, \sigma = 51.6 \] vs. \[ \mu = 172.5s, \sigma = 50.5 \]

More accurate  \[ \mu = 96.3\%, \sigma = 5.82 \] vs. \[ \mu = 69.9\%, \sigma = 13.3 \]

“In Tableau, there is no pattern searching. If I see some pattern in Tableau, such as a decreasing pattern, and I want to see if any other variable is decreasing in that month, I have to go one by one to find this trend. But here I can find this through the query table.”

“you can just [edit] and draw to find out similar patterns. You'll need to do a lot more through Matlab to do the same thing.”

“The obvious good thing is that you can do complicated queries, and you don't have to write SQL queries... I can imagine a non-cs student [doing] this.”
Real Usage Stories (1-year long dev)

• **Confirmed** gene expression profiles in recent publication

• Unknown dip in an astro light curve was **caused due to saturated image equipment**

• **Relationship** between viscosity and lithium solvation energy is independent of whether a solvent is a high or low V solvent
Effortless Visual Exploration of Large Datasets with Ingredients

- Drag-and-drop and sketch based interactions
  - to find specific patterns
- Sophisticated visual exploration language, ZQL
  - to ask more elaborate questions
- Scalable visualization generation engine
  - preprocess, batch and parallel eval. for interactive results
- Rapid pattern matching algorithms
  - sampling-based techniques
Motivation

Collaborative data science is ubiquitous

• Many users, many versions of the same dataset stored at many stages of analysis

• Status quo:
  – Stored in a file system, relationships unknown

Challenge: can we build a versioned data store?

  – Support efficient access, retrieval, querying, and modification of versions
Motivation: Starting Points

• **VCS:** Git/svn is inefficient and unsuitable
  – Ordered semantics
  – No data manipulation API
  – No efficient multi-version queries
  – Poor support for massive files

• **DBMS:** Relational databases don’t support versioning, but are efficient and scalable
OrpheusDB: A Bolt-On Approach

- Retrieve the first version that contains this tuple
- Find versions where the average(salary) is greater than 1000
- Find all pairs of versions where over 100 new tuples were added
- Show the history of the tuple with record id 34.
Representing Versions in a DB: Take 1

<table>
<thead>
<tr>
<th>badgeID</th>
<th>age</th>
<th>gender</th>
<th>salary</th>
<th>vid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>6500</td>
<td>v_1</td>
</tr>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>7500</td>
<td>v_3</td>
</tr>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>7500</td>
<td>v_4</td>
</tr>
<tr>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
<td>v_1</td>
</tr>
<tr>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
<td>v_2</td>
</tr>
<tr>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
<td>v_4</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>v_1</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>v_2</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>v_3</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>v_4</td>
</tr>
<tr>
<td>0004</td>
<td>40</td>
<td>M</td>
<td>9000</td>
<td>v_2</td>
</tr>
<tr>
<td>0004</td>
<td>40</td>
<td>M</td>
<td>9000</td>
<td>v_4</td>
</tr>
<tr>
<td>0005</td>
<td>35</td>
<td>F</td>
<td>6500</td>
<td>v_3</td>
</tr>
<tr>
<td>0005</td>
<td>35</td>
<td>F</td>
<td>6500</td>
<td>v_4</td>
</tr>
<tr>
<td>0006</td>
<td>32</td>
<td>M</td>
<td>7000</td>
<td>v_3</td>
</tr>
<tr>
<td>0006</td>
<td>32</td>
<td>M</td>
<td>7000</td>
<td>v_4</td>
</tr>
</tbody>
</table>

- **Table with Versioned Records**

<table>
<thead>
<tr>
<th>badgeID</th>
<th>age</th>
<th>gender</th>
<th>salary</th>
<th>vlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>6500</td>
<td>{ v_1 }</td>
</tr>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>7500</td>
<td>{ v_3, v_4 }</td>
</tr>
<tr>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
<td>{ v_1, v_2, v_4 }</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>{ v_1, v_2, v_3, v_4 }</td>
</tr>
<tr>
<td>0004</td>
<td>40</td>
<td>M</td>
<td>9000</td>
<td>{ v_2, v_4 }</td>
</tr>
<tr>
<td>0005</td>
<td>35</td>
<td>F</td>
<td>6500</td>
<td>{ v_3, v_4 }</td>
</tr>
<tr>
<td>0006</td>
<td>32</td>
<td>M</td>
<td>7000</td>
<td>{ v_3, v_4 }</td>
</tr>
</tbody>
</table>

- **List: List of versions per record**
Representing Versions in a DB: Take 2

<table>
<thead>
<tr>
<th>badgeID</th>
<th>age</th>
<th>gender</th>
<th>salary</th>
<th>vlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>6500</td>
<td>{v_1}</td>
</tr>
<tr>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>7500</td>
<td>{v_3, v_4}</td>
</tr>
<tr>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
<td>{v_1, v_2, v_4}</td>
</tr>
<tr>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
<td>{v_1, v_2, v_3, v_4}</td>
</tr>
<tr>
<td>0004</td>
<td>40</td>
<td>M</td>
<td>9000</td>
<td>{v_2, v_4}</td>
</tr>
<tr>
<td>0005</td>
<td>35</td>
<td>F</td>
<td>6500</td>
<td>{v_3, v_4}</td>
</tr>
<tr>
<td>0006</td>
<td>32</td>
<td>M</td>
<td>7000</td>
<td>{v_3, v_4}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rid</th>
<th>badgeID</th>
<th>age</th>
<th>gender</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_1</td>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>6500</td>
</tr>
<tr>
<td>r_2</td>
<td>0002</td>
<td>30</td>
<td>F</td>
<td>7500</td>
</tr>
<tr>
<td>r_3</td>
<td>0003</td>
<td>28</td>
<td>M</td>
<td>7000</td>
</tr>
<tr>
<td>r_4</td>
<td>0004</td>
<td>40</td>
<td>M</td>
<td>9000</td>
</tr>
<tr>
<td>r_5</td>
<td>0001</td>
<td>25</td>
<td>F</td>
<td>7500</td>
</tr>
<tr>
<td>r_6</td>
<td>0005</td>
<td>35</td>
<td>F</td>
<td>6500</td>
</tr>
<tr>
<td>r_7</td>
<td>0006</td>
<td>32</td>
<td>M</td>
<td>7000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vid</th>
<th>rlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1</td>
<td>{r_1, r_2, r_3}</td>
</tr>
<tr>
<td>v_2</td>
<td>{r_2, r_3, r_4}</td>
</tr>
<tr>
<td>v_3</td>
<td>{r_3, r_5, r_6, r_7}</td>
</tr>
<tr>
<td>v_4</td>
<td>{r_2, r_3, r_4, r_5, r_6, r_7}</td>
</tr>
</tbody>
</table>
Still slow... Apply partitioning!

Representing Versions in a DB: Take 3

Optimally partitioning minimizing storage and retrieval: NP-Hard!
OrpheusDB

Command Input

Please enter either the SQL or the version control command below:

```sql
SELECT * FROM VERSION 1.2 OF CVD.Interaction
WHERE coexpression > 80
LIMIT 50;
```

Output Results

<table>
<thead>
<tr>
<th>protein1</th>
<th>protein2</th>
<th>neighborhood</th>
<th>cooccurrence</th>
<th>coexpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSP273047</td>
<td>ENSP261890</td>
<td>0</td>
<td>53</td>
<td>83</td>
</tr>
<tr>
<td>ENSP273047</td>
<td>ENSP261890</td>
<td>0</td>
<td>53</td>
<td>83</td>
</tr>
<tr>
<td>ENSP300413</td>
<td>ENSP274242</td>
<td>426</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>ENSP300413</td>
<td>ENSP274242</td>
<td>426</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>ENSP300413</td>
<td>ENSP274242</td>
<td>426</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>ENSP300413</td>
<td>ENSP274242</td>
<td>426</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>ENSP300334</td>
<td>ENSP346022</td>
<td>0</td>
<td>227</td>
<td>975</td>
</tr>
</tbody>
</table>

Version Visualization

Version Graph of CVD:
Some Takeaways…

1. Many underserved communities: *why only focus on the needs of the 1%?*

2. Working with consumers from the get go: *keeps you honest; avoid the non-problems*

3. The “Human-in-the-loop” is crucial: *the interfaces are as important as the algorithms*
Summary: Takeaways

My website: http://data-people.cs.illinois.edu
Twitter: @adityagp