Facet
A recursive approach to visualization

Vadim Ogievetsky
Frontend Developer
@metamarkets
The problem: Datastores are designed to answer single specific queries, not drive visualizations.
For the top four countries (by revenue), what are the top three venues?

You can not answer this question with a single query.
Visualizing data and extracting it from a data store are two sides of the same coin.

The solution I'm exploring is a single declarative language for visualization and data querying.
Hadley Wickham popularized a concept called **split-apply-combine** as a way of thinking about data querying.

http://www.jstatsoft.org/v40/i01/paper
The basics of **split-apply-combine**

- **split** by country
- **apply**: Sum Revenue
- **combine**: sort descending by Sum Revenue, limit 4

### Data

<table>
<thead>
<tr>
<th>Country</th>
<th>Sum Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>$83</td>
</tr>
<tr>
<td>France</td>
<td>$42</td>
</tr>
<tr>
<td>Canada</td>
<td>$36</td>
</tr>
<tr>
<td>Japan</td>
<td>$18</td>
</tr>
</tbody>
</table>
The basics of **split-apply-combine**

<table>
<thead>
<tr>
<th>Country</th>
<th>Venue type</th>
<th>Sum Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>fastfood</td>
<td>$16</td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>$10</td>
</tr>
<tr>
<td></td>
<td>restaurant</td>
<td>$9</td>
</tr>
<tr>
<td>France</td>
<td>cafe</td>
<td>$18</td>
</tr>
<tr>
<td></td>
<td>pub</td>
<td>$12</td>
</tr>
<tr>
<td></td>
<td>restaurant</td>
<td>$2</td>
</tr>
<tr>
<td>Canada</td>
<td>cafe</td>
<td>$10</td>
</tr>
<tr>
<td></td>
<td>fastfood</td>
<td>$4</td>
</tr>
<tr>
<td></td>
<td>park</td>
<td>$3</td>
</tr>
<tr>
<td>Japan</td>
<td>street</td>
<td>$5</td>
</tr>
<tr>
<td></td>
<td>fastfood</td>
<td>$4</td>
</tr>
<tr>
<td></td>
<td>pub</td>
<td>$1</td>
</tr>
</tbody>
</table>
Operations equivalent to **split-apply-combine** are expressible in any reasonable data query language

```sql
SELECT
  `country` AS "Country",
  SUM(`revenue`) AS "Sum Revenue" -- Apply: sum Revenue
FROM `myDataTable`
GROUP BY `country`                -- Split by country
ORDER BY `Sum Revenue` DESC       -- Combine by sorting on
LIMIT 4;                          -- Sum Revenue and limiting
```

Similarly for Excel (Pivot Table), Hadoop (MapReduce), Druid, e.t.c.
Similar concepts can be recognized in data visualization

**split** by country, **combine** by sorting desc. on Sum Revenue, map to the **vertical axis** using an **ordinal scale**.

Define plot of size X by Y

Use `Country` as label

**add labels**

**apply**: sum revenue, call it Sum Revenue, plot rectangles and map **length** to the **horizontal axis** using a **linear scale**, Color with **#45808E**.
Nested application of split-apply-combine can be used to produce more complex visualizations

1. **split** on state
   - **apply** sum population
   - **combine**: sort desc. by population; limit 6

2. **split** on age (bin by 5 year)
   - **combine**: sort by age
   - **apply** sum population
With **facet** a visualization can be used to make the data query

```
SELECT `country` AS "Country",
       SUM(`revenue`) AS "Sum Revenue"
FROM `myDataTable`
GROUP BY `country`
ORDER BY `Sum Revenue` DESC
LIMIT 4;
```
The data driver

split-apply-combine query

nested JSON result

DB
Facet $\Rightarrow$ JavaScript driver

<table>
<thead>
<tr>
<th>Country</th>
<th>Sum Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td></td>
</tr>
</tbody>
</table>
SELECT `country` AS "Country",
    SUM(`revenue`) AS "Sum Revenue"
FROM `myDataTable`
GROUP BY `country`
ORDER BY `Sum Revenue` DESC
LIMIT 4;
Facet ➔ Druid

<table>
<thead>
<tr>
<th>Country</th>
<th>Sum Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td></td>
</tr>
</tbody>
</table>

Facet Visualization Description

Druid query $\sum$
Aside,

Facet ♥ Druid

(Checkout the Druid presentation: Tomorrow 4pm)

http://metamarkets.com/druid
DEMO
Conclusion

- Querying data takes focus away from the visualization.
- A single declarative language for data visualization and data querying would allow for faster iteration on visualizations.
- A visualization description language can be built on top of the split-apply-combine principle of data transformation.
Thank you. Questions?

Strata office hours today at 2:20pm
@Vogiievetsky / vadim@metamarkets.com