Table Sanitizing and Processing

21 September 2022
Where are we?
Last class, we downloaded a real data set – the London Fire Brigade’s animal rescues – as a CSV file.

We imported it to Google Sheets and pruned it to just the columns we were interested in.

Then we loaded the data in Pyret and began to investigate what we had and what issues we’d need to fix.
<table>
<thead>
<tr>
<th>ward</th>
<th>borough</th>
</tr>
</thead>
<tbody>
<tr>
<td>some(&quot;Crystal Palace &amp; Upper Norwood&quot;)</td>
<td>some(&quot;Croydon&quot;)</td>
</tr>
<tr>
<td>some(&quot;Woodside&quot;)</td>
<td>some(&quot;Croydon&quot;)</td>
</tr>
<tr>
<td>some(&quot;Carshalton Central&quot;)</td>
<td>some(&quot;Sutton&quot;)</td>
</tr>
<tr>
<td>some(&quot;Harefield&quot;)</td>
<td>some(&quot;Hillingdon&quot;)</td>
</tr>
<tr>
<td>some(&quot;Gooshays&quot;)</td>
<td>some(&quot;Havering&quot;)</td>
</tr>
<tr>
<td>some(&quot;Alibon&quot;)</td>
<td>some(&quot;Barking and Dagenham&quot;)</td>
</tr>
<tr>
<td>some(&quot;Cathall&quot;)</td>
<td>some(&quot;Waltham Forest&quot;)</td>
</tr>
<tr>
<td>some(&quot;Wanstead&quot;)</td>
<td>some(&quot;Redbridge&quot;)</td>
</tr>
<tr>
<td>some(&quot;New Addington North&quot;)</td>
<td>some(&quot;Croydon&quot;)</td>
</tr>
<tr>
<td>some(&quot;Lea Bridge&quot;)</td>
<td>some(&quot;Hackney&quot;)</td>
</tr>
</tbody>
</table>
We’ll sanitize the “borough” and “ward” fields, which contain blanks:

```perl
include gdrive-sheets
include data-source
This provides the sanitizer functions

include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")

ssid = "1iohS5voB–Y–CezClAphyIdc90t_Me1z3tA0TACem0a4"
spreadsheet = load-spreadsheet(ssid)

rescue-data =
  load-table:
    datetime, year, description, animal-group,
    property-type, property-category, service-kind,
    service-type, ward, borough, station-name, street
  source: spreadsheet.sheet-by-name("rescues-simple", true)

sanitize borough using string-sanitizer
sanitize ward using string-sanitizer
end
```
<table>
<thead>
<tr>
<th>ward</th>
<th>borough</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Crystal Palace &amp; Upper Norwood&quot;</td>
<td>&quot;Croydon&quot;</td>
</tr>
<tr>
<td>&quot;Woodside&quot;</td>
<td>&quot;Croydon&quot;</td>
</tr>
<tr>
<td>&quot;Carshalton Central&quot;</td>
<td>&quot;Sutton&quot;</td>
</tr>
<tr>
<td>&quot;Harefield&quot;</td>
<td>&quot;Hillingdon&quot;</td>
</tr>
<tr>
<td>&quot;Gooshays&quot;</td>
<td>&quot;Havering&quot;</td>
</tr>
<tr>
<td>&quot;Alibon&quot;</td>
<td>&quot;Barking and Dagenham&quot;</td>
</tr>
<tr>
<td>&quot;Cathall&quot;</td>
<td>&quot;Waltham Forest&quot;</td>
</tr>
<tr>
<td>&quot;Wanstead&quot;</td>
<td>&quot;Redbridge&quot;</td>
</tr>
<tr>
<td>&quot;New Addington North&quot;</td>
<td>&quot;Croydon&quot;</td>
</tr>
<tr>
<td>&quot;Lea Bridge&quot;</td>
<td>&quot;Hackney&quot;</td>
</tr>
</tbody>
</table>
Looking for problems
```python
count(rescue_data_clean, "borough")
```

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; &quot;</td>
<td>11</td>
</tr>
<tr>
<td>&quot;WESTMINSTER&quot;</td>
<td>152</td>
</tr>
<tr>
<td>&quot;HAMMERSMITH AND FULHAM&quot;</td>
<td>119</td>
</tr>
<tr>
<td>&quot;CITY OF LONDON&quot;</td>
<td>5</td>
</tr>
<tr>
<td>&quot;RICHMOND UPON THAMES&quot;</td>
<td>130</td>
</tr>
<tr>
<td>&quot;HARROW&quot;</td>
<td>101</td>
</tr>
<tr>
<td>&quot;MERTON&quot;</td>
<td>93</td>
</tr>
<tr>
<td>&quot;SOUTHWARK&quot;</td>
<td>187</td>
</tr>
</tbody>
</table>

What are these rows? What should we do with them?
fun no-borough(r :: Row) -> Boolean:
    doc: "Return true if the borough column is empty for a given row"
    r["borough"] == ""
end

>>> filter-with(rescue-data, no-borough)
fun no-borough(r :: Row) -> Boolean:
    doc: "Return true if the borough column is empty for a given row"
    r["borough"] == ""
end

>>> filter-with(rescue-data, no-borough)

<table>
<thead>
<tr>
<th>ward</th>
<th>borough</th>
<th>station-name</th>
<th>street</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Essex&quot;</td>
<td>&quot;COOLGARDIE AVENUE&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Essex&quot;</td>
<td>&quot;PALMERSTON ROAD&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Essex&quot;</td>
<td>&quot;BRADWELL ROAD&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Chingford&quot;</td>
<td>&quot;RANGERS ROAD&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Kent&quot;</td>
<td>&quot;MEAD WALL&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Buckinghamshire&quot;</td>
<td>&quot;GEORGE GREEN ROAD&quot;</td>
</tr>
</tbody>
</table>
If we make a visualization that shows a count for "", people will rightly think it’s a bug. We should give a more meaningful value for these rows.
The **transform-column** function is used to clean up or otherwise alter the data in a single column of a table.

It returns a new table by applying its function argument to each value of the given column.
fun empty-to-other(s :: String) -> String:
    doc: "Return the string or 'Other' if the string is empty."
    if s == "":
        "Other"
    else:
        s
    end
end

rescue-data-clean =
    transform-column(
        transform-column(rescue-data,
            "borough", empty-to-other),
        "ward", empty-to-other)
order-by(
count(rescue-data-clean, "borough"), "value", true)

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BARKING AND DAGENHAM&quot;</td>
<td>98</td>
</tr>
<tr>
<td>&quot;BARNET&quot;</td>
<td>198</td>
</tr>
<tr>
<td>&quot;BEXLEY&quot;</td>
<td>130</td>
</tr>
<tr>
<td>&quot;BRENT&quot;</td>
<td>142</td>
</tr>
<tr>
<td>&quot;BROMLEY&quot;</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Barking and Dagenham&quot;</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Barnet&quot;</td>
<td>151</td>
</tr>
</tbody>
</table>
order-by(
    count(rescue-data-clean, "borough"),
    "value", true)

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BARKING AND DAGENHAM&quot;</td>
<td>98</td>
</tr>
<tr>
<td>&quot;BARNET&quot;</td>
<td>198</td>
</tr>
<tr>
<td>&quot;BEXLEY&quot;</td>
<td>130</td>
</tr>
<tr>
<td>&quot;BRENT&quot;</td>
<td>142</td>
</tr>
<tr>
<td>&quot;BROMLEY&quot;</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Barking and Dagenham&quot;</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Barnet&quot;</td>
<td>151</td>
</tr>
</tbody>
</table>
Some of the boroughs are ALL UPPERCASE and others are Title Case.

"Islington" and "ISLINGTON" will be treated by Pyret as distinct values, which will throw off our counts!
rescue-data-clean =
    transform-column(
        transform-column(
            transform-column(
                transform-column(rescue-data,
                    "borough", empty-to-other),
                    "ward", empty-to-other),
                "borough", string-to-upper),
            "ward", string-to-upper))
fun clean-table(t :: Table) -> Table:
  doc: "Fix various columns from the animal rescue spreadsheet"

  # Change the blank strings to "Other"
  t1 =
  transform-column(
    transform-column(t,
      "borough", empty-to-other),
    "ward", empty-to-other)

  # Change strings to all uppercase
  t2 =
  transform-column(
    transform-column(t1,
      "borough", string-to-upper),
    "ward", string-to-upper)

  t2
end

rescue-data-clean = clean-table(rescue-data)
Now we’ll want to use `rescue-data-clean` rather than `rescue-data` for the rest of our work.

We usually maintain separate names for the initially-loaded table, the cleaned table, and for significant variations for analysis purposes.
Data analysis
```
>>> count(rescue_data_clean, "borough")

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;OTHER&quot;</td>
<td>11</td>
</tr>
<tr>
<td>&quot;TANDRIDGE&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;BROXBOURNE&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;BRENTWOOD&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;CITY OF LONDON&quot;</td>
<td>13</td>
</tr>
<tr>
<td>&quot;ENFIELD&quot;</td>
<td>394</td>
</tr>
<tr>
<td>&quot;MERTON&quot;</td>
<td>172</td>
</tr>
<tr>
<td>&quot;BRENT&quot;</td>
<td>246</td>
</tr>
</tbody>
</table>
```
```python
>>> order_by(
    count(rescue_data_clean, "borough"),
    "value", true)
```

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Barking and Dagenham&quot;</td>
<td>214</td>
</tr>
<tr>
<td>&quot;Barnet&quot;</td>
<td>349</td>
</tr>
<tr>
<td>&quot;Bexley&quot;</td>
<td>236</td>
</tr>
<tr>
<td>&quot;Brent&quot;</td>
<td>246</td>
</tr>
<tr>
<td>&quot;Brentwood&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Bromley&quot;</td>
<td>309</td>
</tr>
<tr>
<td>&quot;Broxbourne&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>
```sql
>>> order-by(
    count(rescue-data-clean, "borough"),
    "name", false)

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ENFIELD&quot;</td>
<td>394</td>
</tr>
<tr>
<td>&quot;CROYDON&quot;</td>
<td>358</td>
</tr>
<tr>
<td>&quot;BARNET&quot;</td>
<td>349</td>
</tr>
<tr>
<td>&quot;SOUTHWARK&quot;</td>
<td>347</td>
</tr>
<tr>
<td>&quot;TOWER HAMLETS&quot;</td>
<td>336</td>
</tr>
<tr>
<td>&quot;HARINGEY&quot;</td>
<td>322</td>
</tr>
<tr>
<td>&quot;NEWHAM&quot;</td>
<td>316</td>
</tr>
</tbody>
</table>
```
borough-counts = order-by(
    count(rescue-data-clean, "borough"),
    "value", true)

>>> pie-chart(borough-counts, "value", "count")
What about getting a pie chart for the types of animals? Well, we could do

```
animal-counts =
    order-by(
        count(rescue-data-clean, "animal-group"),
        "value", true)
```

and then

```
pie-chart(animal-counts, "value", "count")
```
What about getting a pie chart for the types of animals? Well, we could do

```
animal-counts =
    order-by(
        count(rescue-data-clean, "animal-group"),
        "value", true)
```

and then

```
pie-chart(animal-counts, "value", "count")
```
Rather than copy and paste long expressions, we should define a function:

```plaintext
fun pie-chart-counts(t :: Table, col :: String) -> Image:
  doc: "Make a pie chart of the number of times each value occurs in the specified table column"
  counts = order-by(count(t, col), "value", true)
  pie-chart(counts, "value", "count")
end
```
pie-chart-counts(rescue-data-clean, "animal-group")
The cliché may be true!
But we see a familiar problem – we have a count for "Cat" and for "cat".

Back to cleaning data!
fun clean-table(t :: Table) -> Table:
  doc: "Fix various columns from the animal rescue spreadsheet"

  # Change the blank strings to "Other"
  t1 =
  transform-column(
    transform-column(t,
      "borough", empty-to-other),
    "ward", empty-to-other)

  # Change strings to all uppercase
  t2 =
  transform-column(
    transform-column(
      transform-column(t1,
        "borough", string-to-upper),
      "ward", string-to-upper),
    "animal-group", string-to-upper)

  t2
end

rescue-data-clean = clean-table(rescue-data)
pie-chart-counts(rescue-data-clean, "animal-group")

- CAT: 4,252 (48.6%)
- BIRD: 20.6%
- BUDGIE: 15.3%
- BULL: 5.2%
- CAT: -
- COW: -
- DEER: -
- DOG: -
- FERRET: -
- FISH: -
- FOX: -
- GOAT: -
- HAMSTER: -
- HEDGEHOG: -
- HORSE: -
- LAMB: -
- LIZARD: -
Ok, I *have* to know: How many of those cats are actually stuck up trees?
About 15% of the time, the "description" column is "Redacted", but when it isn’t, this gives us a way to approximate the question.
Plan

Get just the rows about cats

Figure out for each row if it involves a tree

Make a pie chart

Need a column to count values in, so we’d better put the up-a-tree-or-not answer into a table.
fun is-cat(r :: Row) -> Boolean:
    doc: "Return true if the row is about rescuing a cat"
    r["animal-group"] == "CAT"
end

just-cats = filter-with(rescue-data-clean, is-cat)
Plan

✓ Get just the rows about cats

Figure out for each row if it involves a tree

Make a pie chart

   Need a column to count values in, so we’d better put the up-a-tree-or-not answer into a table.
fun up-tree(r :: Row) -> String:
    doc: "Return string indicating whether or not cat is up a tree"

    desc = string-to-lower(r["description"])

    if string-contains(desc, "tree"): 
        "up a tree!"
    else:
        "not up a tree"
end
end

cats-who-may-be-up-trees =
    build-column(just-cats, "up-tree", up-tree)
fun up-tree(r :: Row) -> String:
    doc: "Return string indicating whether or not cat is up a tree"

desc = string-to-lower(r["description"])

if string-contains(desc, "tree"):
    "up a tree!"
else:
    "not up a tree"
end
end

cats-who-may-be-up-trees =
    build-column(just-cats, "up-tree", up-tree)

This handles descriptions with "TREE" or "Tree"
Plan

- Get just the rows about cats
- Figure out for each row if it involves a tree

Make a pie chart

Need a column to count values in, so we’d better put the up-a-tree-or-not answer into a table.
pie-chart-counts(cats-who-may-be-up-trees, "up-tree")
Plan

✔ Get just the rows about cats

✔ Figure out for each row if it involves a tree

✔ Make a pie chart

Need a column to count values in, so we’d better put the up-a-tree-or-not answer into a table.
Slander!
Changes
That’s enough pie for now.

Let’s make a bar chart to see the number of animal rescues per year.
`freq-bar-chart(rescue-data, "year")`
```
freq-bar-chart(rescue-data, "year")
```
We can split the data into pre-2020 and 2020–2022 to explore this surge.

```haskell
fun plague-year(r :: Row) -> Boolean:
  doc: "Return true if the row takes place during the COVID pandemic (rounding up to full years)"
  (r["year"] >= 2020) and (r["year"] <= 2022)
end

rescue-data-plague = filter-with(rescue-data-clean, plague-year)

fun not-plague(r :: Row) -> Boolean:
  doc: "Return true if the row is from a year entirely before or after the COVID pandemic"
  not(plague-year(r))
end

rescue-data-not-plague = filter-with(rescue-data-clean, not-plague)
```
What else could we compare for these time periods?
Class code:

https://tinyurl.com/101-2022-09-21
Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University