Designing Programs for Tables

14 February 2022
Where are we?
We can represent complex data as tables.

These can be encoded directly in a program or loaded from an external source.

Real data may need clean-up, which can be manual or automatic.
Automatic data clean-up includes using *sanitizers*, which ensure all data in a column is of the desired type, providing default values for empty cells.
We can modify table data later using `transform-column`, and we can remove (apparent) bad data using `filter-with`.
We saw how this clean-up process works last week in lab by looking at the student data from the form (some of) you filled out.
Task plans
If you aren’t sure how to approach a problem, don’t start by trying to write code!

Plan until you understand the problem.
1 Develop a concrete example of desired output
   Typically a table with 4–6 rows

2 Identify functions useful to transform data
   Functions you already know or look up in the documentation

3 Develop a sequence of steps to transform data
   Draw as pictures, use textual descriptions, or a combination of the two
   Use functions from previous step

4 Repeat Step 3 to further break down steps until you can write
   expressions/functions to perform each step
Example: Binning
What if we want to look at the distribution of responses to the question

Would you classify your academic focus as humanities, STEM, or somewhere in the middle? *
We don’t particularly care about how many students rated their STEM-iness as 2 or 8 or any particular number.

Instead, we might want to *bin* the responses into a few categories.
Let’s come up with a task plan to count the number of students in these three categories.
... stem-level ...
... 1 ...
... 10 ...

\[ \text{use } \text{build-column} \]

\[ \begin{array}{c|c}
\text{value} & \text{count} \\
\text{``non-stem''} & 3 \\
\text{``stem''} & 5 \\
\text{``super-stem''} & 3 \\
\end{array} \]

\[ \text{use } \text{pie-chart} \]
This needs a helper function! Call it `stem-category`.
Task plan:

1. Write `stem-category`.
2. Add stem category to table using `build-column`.
3. Summarize results using `count`.
4. Visualize the results using `pie-chart`. 
test-table =
  table: stem-level
    row: 1
    row: 3
    row: 4
    row: 7
    row: 8
    row: 10
  end

fun stem-category(r :: Row) -> String:
  doc: "Return a stem category (non-stem, stem, or super-stem) for a given stem-level"
    ...
  where:
    stem-category(test-table.row-n(0)) is "non-stem"
    stem-category(test-table.row-n(1)) is "non-stem"
    stem-category(test-table.row-n(2)) is "stem"
    stem-category(test-table.row-n(3)) is "stem"
    stem-category(test-table.row-n(4)) is "super-stem"
    stem-category(test-table.row-n(5)) is "super-stem"
  end

The test table can omit the columns we’re not using!

If the survey data changes, our tests will still pass!
fun stem-category(r :: Row) -> String:
  doc: "Return a stem category (non-stem, stem, or super-stem) for a given stem-level"
  s = r["stem-level"]
  if stem-level < 4:
    "non-stem"
  else if stem-level < 8:
    "stem"
  else:
    "super-stem"
end

where:
  ...
end
\textit{data-stem-category} =
\hspace{1em} build-column(student-data-cleaned, "stem-category", stem-category)

\textit{counts} =
\hspace{1em} count(data-stem-category, "stem-category")

pie-chart(counts, "value", "count")
Nested functions
(Optional) Part 3: Going further

Congratulations! You have reached the end of lab. Here is an optional exercise in case you are looking for a challenge:

**TASK:** Write a function `percent-true` that takes a table and column name as input and returns the percent of rows that are `true` for the column specified.

```haskell
fun percent-true(t :: Table, col :: String) -> Number:
  "Returns the percentage of rows that are true in column 'col"
  ...
end
```

What's neat about this function is it will work on any table that has a column of type `Boolean`!

**TASK:** Use this helper function to find the percentage of survey responders who are student-athletes. Check to see if it's the same answer you got for Part 2.3.
fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col"
  ...
end
fun percent-true(t :: Table, col :: String) -> Number:
    doc: "Return the percentage of rows that are true in column 'col"
    filter-with(t, ...).length() / t.length()
end
fun true-filter(r :: Row) -> Boolean:
  doc: "Return true if 'col' is true in this row"
  r[col]
end

fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col'"
  filter-with(t, true-filter).length() / t.length()
fun true-filter(r :: Row) -> Boolean:
  doc: "Return true if 'col' is true in this row"
  r[col] \not= true
end

fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col'"
  filter-with(t, true-filter).length() / t.length()
fun true-filter(r :: Row) -> Boolean:
  doc: "Return true if 'col' is true in this row"
  r[col]
end

fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col'"
  filter-with(t, true-filter).length() / t.length()
col is undefined in true-filter.

Pyret only knows the value for col when you’re inside percent-true.

This means we need to define true-filter inside percent-true!
fun `percent-true`(t :: Table, col :: String) -> Number:
    doc: "Return the percentage of rows that are true in column 'col'"

    fun `true-filter`(r :: Row) -> Boolean:
        r[col]
    end

    filter-with(t, true-filter).length() / t.length() 
end
As usual, we should test our function using a simple test table:

```plaintext
test-table-student-athlete =
  table:
    student-athlete
    row: true
    row: false
end

fun percent-true(t :: Table, col :: String) -> Number:
  ...
where:
  percent-true(test-table-student-athlete, "student-athlete") is 0.5
end
```
The only time you *need* to use a nested function is if that function needs data that can’t be passed in directly to the function.
Introducing $\lambda$
fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col"

fun true-filter(r :: Row) -> Boolean:
  r[col]
end

filter-with(t, true-filter).length() / t.length()
end

This is a really simple function, which we only use in one place. Instead of defining it like normal, we can write it in line where it’s used.
fun `percent-true`(t :: Table, col :: String) -> Number:
    doc: "Return the percentage of rows that are true in column 'col'"
    filter-with(t, lam(r): r[col] end).length() / t.length()
A *lambda expression* defines an anonymous function – a function that can be passed as an argument but doesn’t have an associated name.
Lambda expressions can be convenient for giving to higher-order functions `filter-with`, `build-column`, and `transform-column`.

We’ll use them more after Exam 1!
Preview: Lists
We’ve seen that when you want a row of a table, you use `.row-n` and get a Row.

What about getting a column?
<table>
<thead>
<tr>
<th>timestamp</th>
<th>house</th>
<th>stem-level</th>
<th>sleep-hours</th>
<th>schoolwork-hours</th>
<th>student-athlete</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2/09/2022 19:03:33&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/09/2022 20:00:52&quot;</td>
<td>&quot;Main&quot;</td>
<td>10</td>
<td>4</td>
<td>7</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/09/2022 20:36:00&quot;</td>
<td>&quot;Main&quot;</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 00:15:17&quot;</td>
<td>&quot;Strong&quot;</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:49:27&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:53:12&quot;</td>
<td>&quot;Davison&quot;</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:05:47&quot;</td>
<td>&quot;Josselyn&quot;</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:06:22&quot;</td>
<td>&quot;Strong&quot;</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:26:46&quot;</td>
<td>&quot;Jewett&quot;</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:35:15&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>true</td>
</tr>
</tbody>
</table>

Click to show the remaining 23 rows...
>>> student-data-cleaned.get-column("house")
["OTHER", "Main", "Main", "Strong", ...]
Lists can be very convenient!
fun normalize-house(house :: String) -> String:
    doc: "Return one of the nine Vassar houses or 'Other'"
    if (house == "Main") or
        (house == "Strong") or
        (house == "Raymond") or
        (house == "Davison") or
        (house == "Lathrop") or
        (house == "Jewett") or
        (house == "Josselyn") or
        (house == "Cushing") or
        (house == "Noyes"):
        house
    else:
        "Other"
    end
where:
    normalize-house("Main") is "Main"
    normalize-house("Offcampus") is "Other"
houses = ["Main", "Strong", "Raymond", 
           "Davison", "Lathrop", "Jewett", "Josselyn", 
           "Cushing", "Noyes"]

fun normalize-house(house :: String) -> String:
    doc: "Return one of the nine Vassar houses or 'Other'"
    if member(houses, house):
        house
    else:
        "Other"
end

where:
    normalize-house("Main") is "Main"
    normalize-house("Offcampus") is "Other"
end
Just like we did when we introduced tables, we’re separating our data from our computation!
Class code:

https://code.pyret.org/editor#share=18c8-mSQYWIF_OKVH3jQQ5DH_4Vjrw&v=1904b2c
Acknowledgments

This lecture incorporates material from:

Kathi Fisler, Brown University
Jason Waterman, Vassar College