CMPU 101 § 02 · Computer Science I

Designing Programs for Tables

26 September 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 and can provide 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

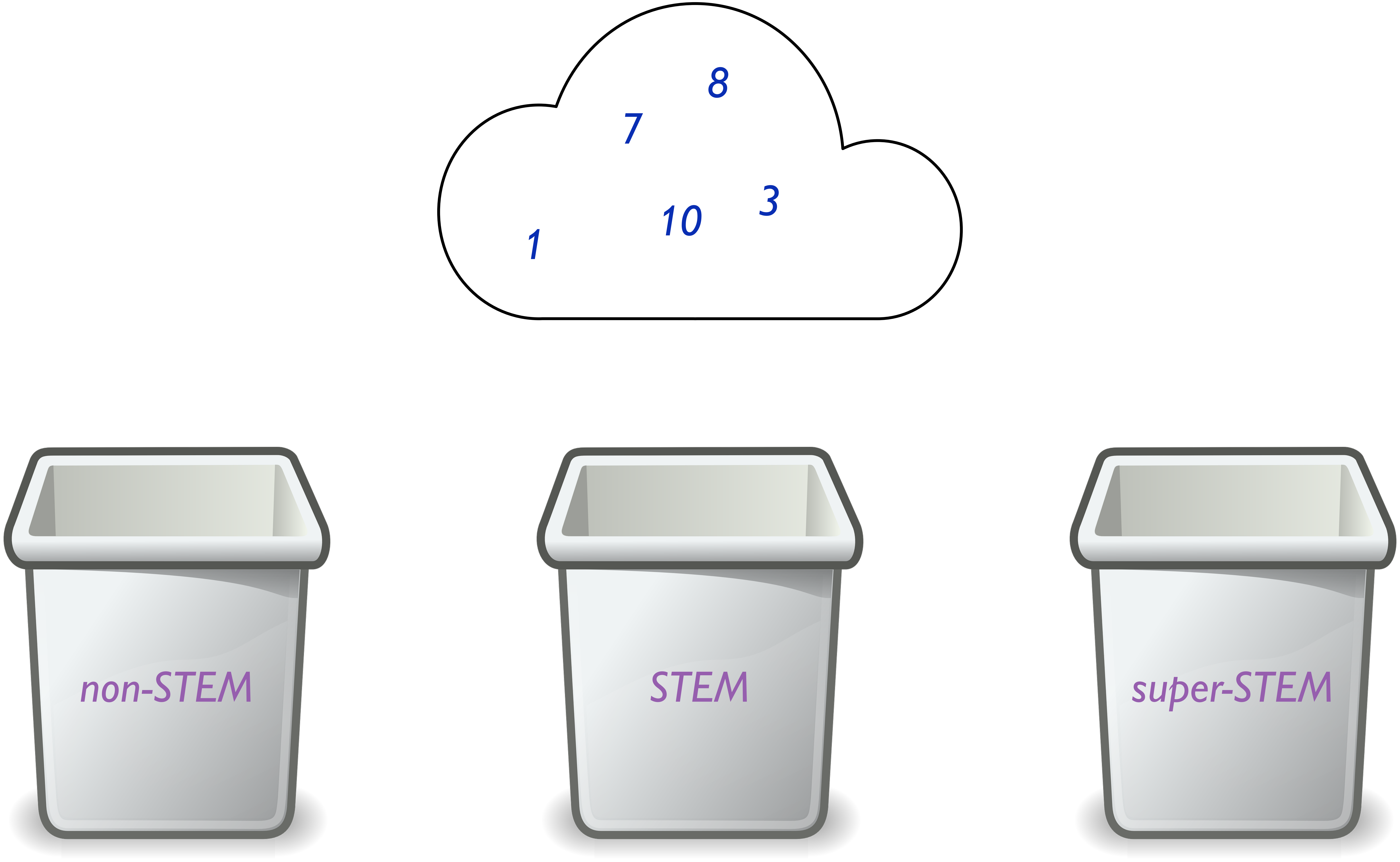
What if we want to look at the distribution of responses to the question

Would you classify your academic focus as more humanities, more STEM (science, technology, engineering, and mathematics), or in the middle? *

	1	2	3	4	5	6	7	8	9	10	
Super humanities	○	○	○	○	○	○	○	○	○	○	Super STEM

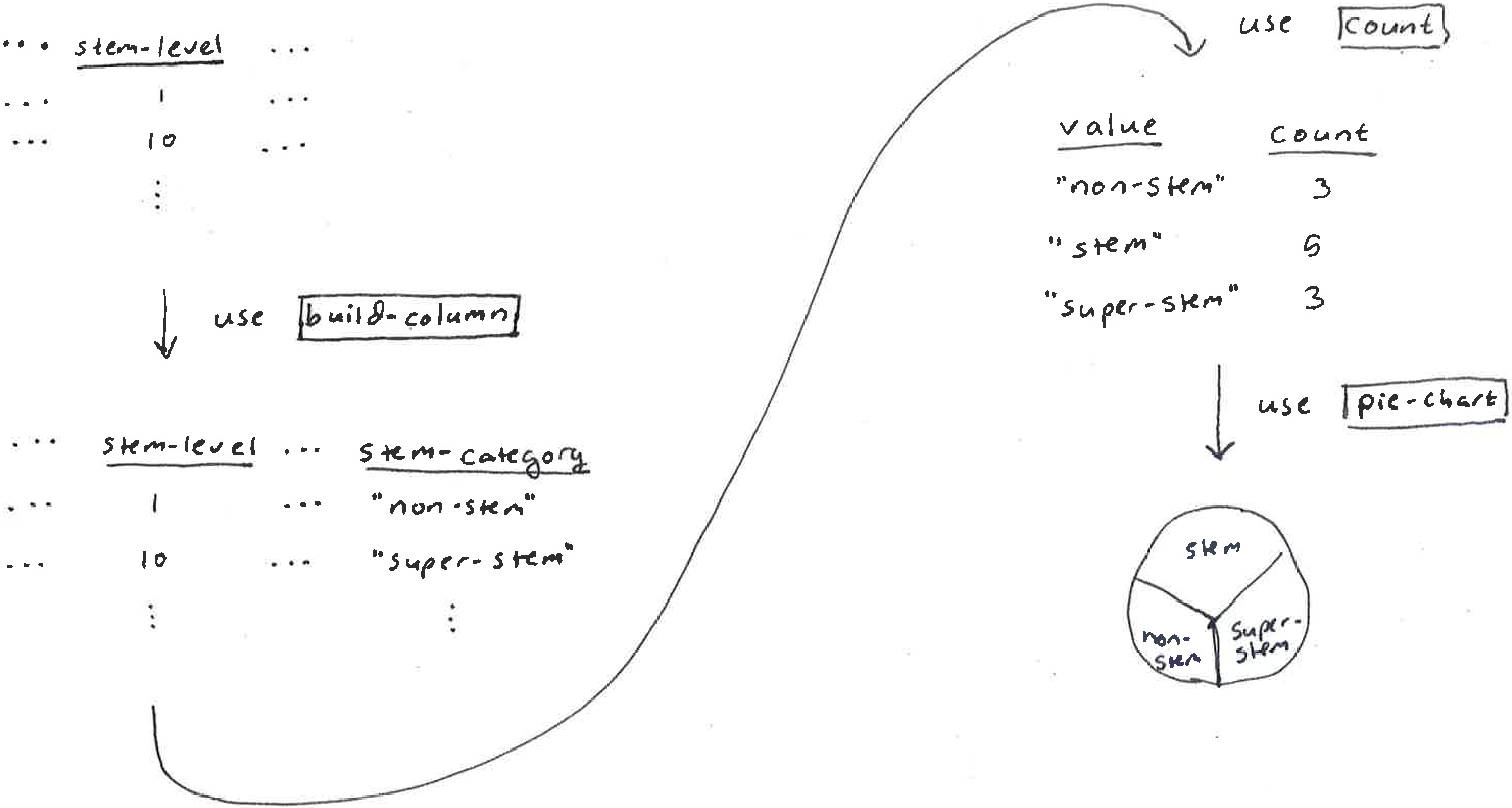
We don’t particularly care about how many students rated their STEM-iness as 2 or 8 or any particular number.

Instead, we might want to ***bin*** the responses into a few categories.

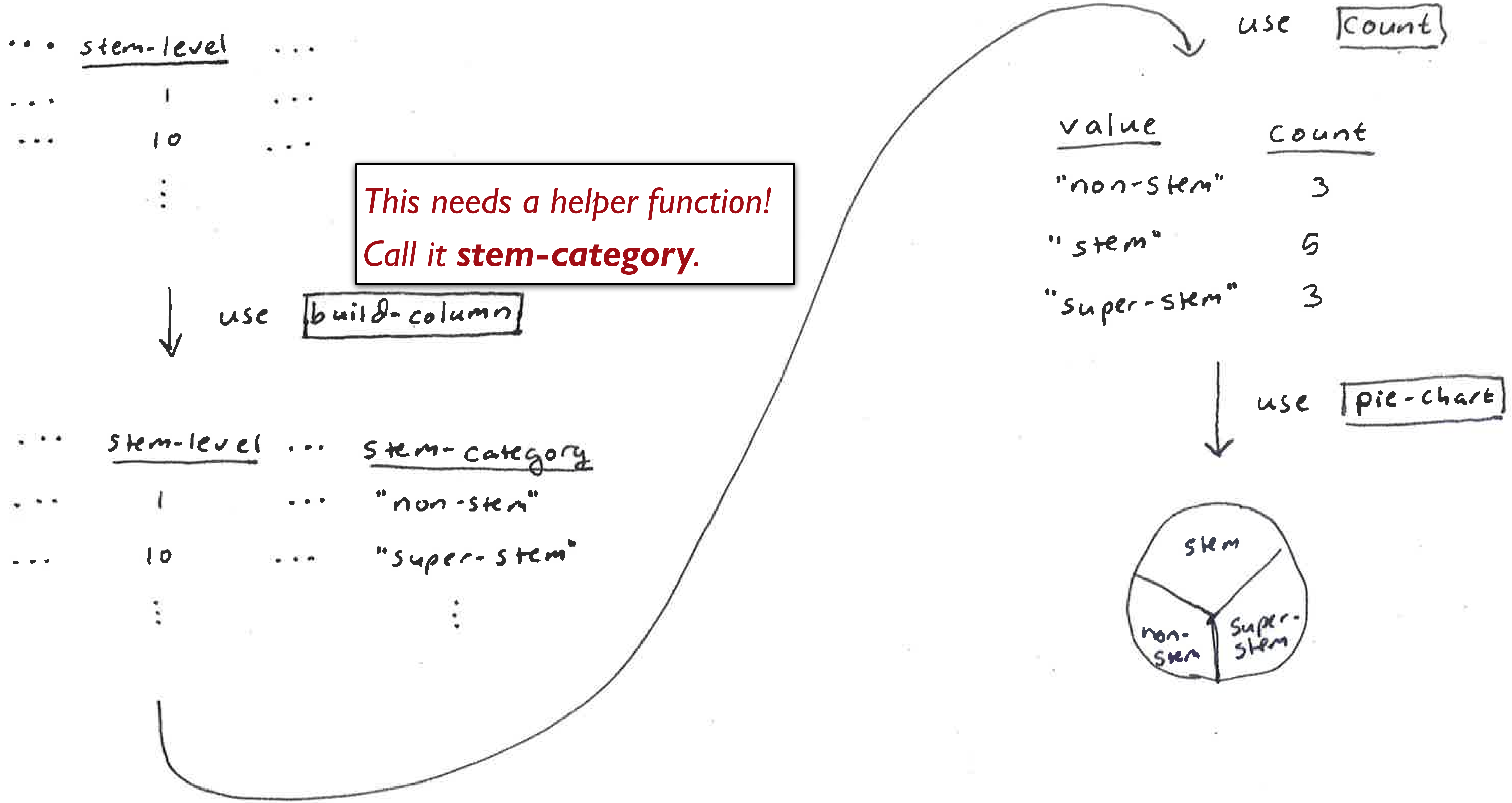

8
7
3
10
1
non-STEM
STEM
super-STEM


1
3
non-STEM
7
STEM
8
10
super-STEM

Let’s come up with a task plan to count the number of students in these three categories.



... stem-level ...
... 1 ...
... 10 ...
use build-column
... stem-level ... stem-category
... 1 ... "non-stem"
... 10 ... "super-stem"
use count
value count
"non-stem" 3
"stem" 5
"super-stem" 3
use pie-chart
stem
non-stem
super-stem



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
```

```
data-stem-category =
  build-column(student-data-cleaned,
    "stem-category", stem-category)

counts =
  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.

```
fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return 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 Exercise 2.1.

Submitting the lab

- When you've completed the exercises, show your code to your instructor or one of the coaches.

```
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()
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()
end
```

Not r[col] == true!

```
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()
end
```

Why doesn’t this work?

`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:

```
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 λ

```
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
```

```
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.
end
```

This is a really simple function, which we only use in one place. Instead of defining it like normal, we can write it inline 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()
end
```

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?

timestamp	house	stem-level	sleep-hours	schoolwork-hours	student-athlete	extracurricular-hours
"9/21/2022 21:05:52"	"OTHER"	10	6	8	false	0
"9/21/2022 21:08:21"	"Strong House (1893)"	10	7	8	false	2
"9/21/2022 21:09:01"	"Lathrop House (1901)"	6	7	4	true	4
"9/21/2022 21:09:43"	"Lathrop House (1901)"	9	8	4	false	0
"9/21/2022 21:29:32"	"Lathrop House (1901)"	8	6	7	false	2
"9/21/2022 21:33:00"	"Cushing House (1927)"	7	8	5	false	2
"9/21/2022 21:38:40"	"Josselyn House (1912)"	8	7	4	false	1.5
"9/21/2022 21:41:36"	"Jewett House (1907)"	9	6	3	true	2
"9/21/2022 21:43:53"	"Main Building (1861)"	8	7	4	true	3
"9/21/2022 22:22:48"	"Davison House (1902)"	8	7	9	false	4

Click to show the remaining 40 rows...

```
>>> student-data-cleaned.get-column("house")
[list: "OTHER", "Strong House (1893)", "Lathrop House
(1901)", "Lathrop House (1901)", ...]
```

Lists can be very convenient!

```
fun normalize-house(house :: String) -> String:
  doc: "Return one of the nine Vassar houses or 'OTHER'"
  if (house == "Main Building (1861)") or
    (house == "Strong House (1893)") or
    (house == "Raymond House (1897)") or
    (house == "Lathrop House (1901)") or
    (house == "Davison House (1902)") or
    (house == "Jewett House (1907)") or
    (house == "Josselyn House (1912)") or
    (house == "Cushing House (1927)") or
    (house == "Noyes House (1958)"):
    house
  else:
    "OTHER"
  end
where:
  normalize-house("Main Building (1861)") is "Main Building (1861)"
  normalize-house("Offcampus") is "OTHER"
end
```

```
houses = [list:
  "Main Building (1861)",
  "Strong House (1893)",
  "Raymond House (1897)",
  "Lathrop House (1901)",
  "Davison House (1902)",
  "Jewett House (1907)",
  "Josselyn House (1912)",
  "Cushing House (1927)",
  "Noyes House (1958)"
]

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 Building (1861)"
  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://tinyurl.com/101-2022-09-26`

Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University

Jason Waterman, Vassar College

Jonathan Gordon, Vassar College