Lab 6 — TypeScript, HTML, and Representation in Computing

In this lab we'll use web technologies to critically examine representation in K-12 CS education in California.

Pre-reading

Computing is considered a "21st-century literacy", with many arguing compellingly that all students should learn *some* computing, whether or not they intend to pursue computing as a career. While progress has been made toward making this vision a reality, much yet remains to be done. In California, **only 5%** of high-school students take CS courses.[1] When one looks at individual demographics, the situation is even more dire for slices of society. Specifically, Hispanic/Latino students and Black students still take CS courses at lower rates than White or Asian students, and girls still take CS courses at lower rates than boys. Crucially, this is not due to a lack of interest in studying computing.

Please read the paper **Diversity Barriers in K-12 Computer Science Education: Structural and Social**. The paper explores various reasons behind these disparities through large-scale survey data collection (not limited to just California).

As you work through the lab, answer the **Reflection questions** at the end of Parts 1, 2, and 3, drawing on what you have read in the paper.

Objectives

- To use TypeScript to embed Vega-lite figures into HTML pages.
- To transform an existing dataset to prepare it for use in Vega-lite visualizations.
- To use TypeScript features like variables, collections, conditions, loops, functions, and problem-solving patterns like map, reduce, filter, and sort to perform the above transformations.
- To critically examine representation in K-12 CS education in the state of California.
- To identify and answer *original* questions regarding representation in K-12 CS education in California.

BPC learning objectives

- To reflect on the which community groups might be served by this type of computational artifact/program and which groups were left out of the development process
- To reflect on the ways that computing can offer opportunities for achieving communal goals (and be able to define the ways computing can be used to reach these goals)

Introduction

In this lab, we'll put together everything we've learned so far. That is, we'll use the following pieces:

- HTML (and some CSS)

- TypeScript
- Vega-lite

Over the past 7 weeks, you've come *really far* in terms of your programming knowledge and abilities. 7 weeks ago, many of you were seeing HTML and CSS for the first time, discussing different kinds of data and making charts using Vega-lite, and dipping your toes into general-purpose programming using TypeScript.

Today, you're reasonably familiar with:

- Data types and variables
- Arrays
- Functions
- If conditions and control flow
- Loops
- Problem-solving patterns (map, filter, reduce)
- Functions as values

That's a lot! Congratulations on being awesome and learning so much so fast!

Overview

In this lab, we're going to continue to interrogate the state of K-12 CS Education in California. In the previous lab, we broke down enrollment trends in CS based on gender and the county's rural/urban status. This time, the dataset includes another dimension of information—**race**.

The dataset

The data is in the `dataset.json` file.

In addition to the inclusion of `Race` as a field in the dataset, you'll notice that the dataset is structured a little differently.

The first thing you should notice is that each county appears in several records in the dataset.

This is because each record no longer represents an entire county. **Each record now represents a *slice* of the students in that county.**

For example, the first record is:

```
{
  "countyName":"Alameda",
  "Race":"African American",
  "Gender":"F",
  "courseType":"AP CS",
  "totalStudents":19,
  "isRural":false
},
```

This record says that there are **19 female African American** students taking **AP CS** courses in **Alameda county**.

The next record is:

```
{
  "countyName":"Alameda",
  "Race":"African American",
  "Gender":"F",
  "courseType":"Non-AP CS",
  "totalStudents":292,
  "isRural":false
},
```

This says that there are **292 female African American** students taking **non-AP CS** courses in **Alameda county**.

For each county, there's an individual record for *each slice of students*, based on: * Race * Gender * Type of enrollment (AP CS, non-AP CS, or overall enrollment)

So if I wanted to know the *total number* of female African American students taking CS courses in each , I would *filter* the list to only include records where `Gender` is `"F"` , `Race` is `"African American"` , and `courseType` is `"AP CS" || "Non-AP CS"` , and then sum up the `totalStudents` value.

So why have I provided the data in this format?

Having the original data sliced up in this way allows us to perform arbitrary transformations on the data to get it into the format we want. This enables us to answer a broader variety of questions from our dataset.

We're going to be doing *visual analysis* in this lab. Instead of (only) printing out averages or percentages, you'll instead create informative figures that can communicate a lot more information at once.

Separating the data based on nominal fields like `Race` , `courseType` , `Sex` , and `isRural` allows us to easily use those different categories in our visual encodings in Vega-lite. This similar to how we used `isUpForAdoption` and `Sex` of Cats in the Cal Poly Cat Program dataset.

First let's take a look at what's already given to us in `script.ts` .

The first line is importing the `vegaEmbed` function, which we use to draw Vega-lite figures in HTML, and the `VisualizationSpec` interface, which we can use to get type hints while creating Vega charts.

The next statement (on line 3) creates the `County` interface. We use this to describe the data that we see in `dataset.json` and in the examples above.

Take a second to confirm that the field names in the interface match the field names in the records in the dataset. If any fields don't match, this can lead to unpredictable and hard-to-find errors, as some of you saw in Lab 5.

Next, we've got these two lines:

```
const dataset: string = await (await fetch('dataset.json')).text();
const countyData: County[] = JSON.parse(dataset);
```

The first line reads the data into one giant `string`.[2] The second line turns that giant `string` into an array of `County` objects and assigns that to the `countyData` variable.

`countyData` now contains all of the data from the `dataset.json` file. Since the data is separated based on a number of categories, we can "slice and dice" the data any way we want to help with our analysis.

Part 1

Task 1: Overall enrollments by Race

First, we'd like to know how many students of each race (according to the US census categories) are enrolled in California public high schools.

That is, we'd like to only look at records whose `courseType` is `"Overall Enrollment"`.

This part of the lab walks you through the steps to do this.

Step 1: Filter the `countyData` array to only include records where `courseType === "Overall Enrollment"`.

Create a new array called `overallEnrollment` that only includes records showing the overall enrollment in the county (i.e., as opposed to CS or AP CS).

```
const overallEnrollment: County[] = countyData.filter(function(c) {
  return c['courseType'] === 'Overall Enrollment';
});
```

See the Replit in the "Functions as values" module on "map and filter" for a reminder of how this works.

The `filter` above takes the function that given to it as an input, and runs that function on each item in `countyData`. If the result is `true`, that item is included in the resulting list. After the dust settles, the `overallEnrollment` array includes all records where `courseType` is `"Overall Enrollment"`.

Step 2: Create a Vega-lite bar chart showing overall enrollment for individual races in California.

Using the new array, create a new Vega-lite chart specification. We want the chart to show the following:

- On the `x` axis, show a **bar** for each `Race`
- On the `y` axis, the height of each **bar** should be the **sum** of the `totalStudents` for that `Race`

Use the following as a starting point. Fill in the blanks.

```
const overallEnrollmentChart: VisualizationSpec = {
  title: "Overall enrollment of high school students.",
  data: {
    values: ______
  },
  mark: {
    type: ______
  },
  encoding: {
    x: {
      field: _______,
      type: 'nominal'
    },
    y: {
      field: ________,
      aggregate: 'sum',
      type: 'quantitative',
      title: 'Number of students'
    }
  }
}
```

Notice that we have added a `title` *to the chart as well as to the* `y` *encoding.* Since this document is going to have many charts, it's important that you communicate clearly what each chart depicts. If you omit the titles, Vega-lite does its "best" based on the data it's plotting.

So for the x-axis, that worked fine, because `Race` is a meaningful title for that axis. For the y-axis, it would have labelled it `totalStudents`, which may be meaningful but is not super professional.

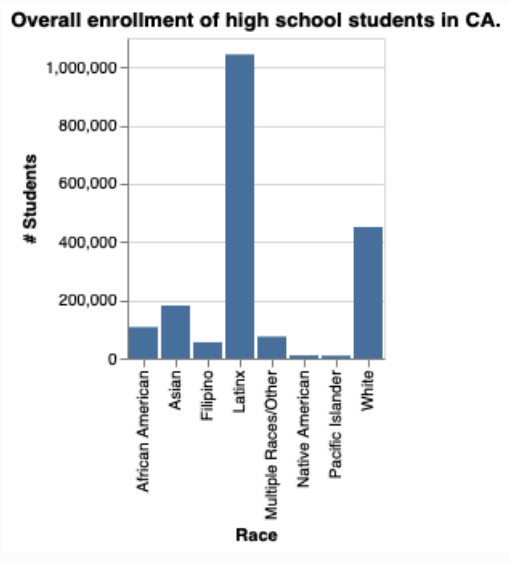
Step 3: Display the chart in the HTML document.

Your `index.html` already has an empty `<div>` container with the id `"overall-chart"`. Use `vegaEmbed` to add your new chart to the HTML page.

```
vegaEmbed('#overall-chart', overallEnrollmentChart);
```

So go ahead and hit the **Run** button.

You should see the following chart showing the total number of students of each race enrolled in public high schools in California.



Num students in California.

Take a second to study the chart specification you just created. The figure uses two visual channels: the horizontal position (`x`) and the vertical position (`y`). These are mapped ("encoded") to two fields in the dataset:

- `x` is mapped to `Race` , so the horizontal position of the bars correspond to the Race
- `y` is mapped to `totalStudents` , so the height of the bars represents the number of students of that `Race`

We can also gather some useful insight from this chart. For example, that Latinx-identifying students are by far the largest population of students in California, followed by White-identifying students.

What's the `aggregate: 'sum'` part doing?

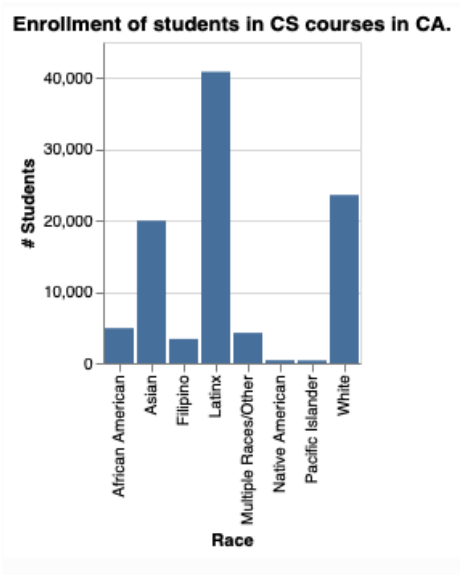
Since we're only separating the data based on the `Race` field, we want to aggregate the `totalStudents` value for all other categories (i.e., in this figure, we're not displaying counts based on `Gender` or `courseType`). So in the `y` encoding we also include an aggregation: we `sum` up the `totalStudents` across all categories except `Race` .

Task 2: CS Enrollments by Race

Next, we'd like to plot data about students enrolled in CS courses.

Do the following, looking at the previous part as a reference as needed: 1. Create a new array called `csEnrollments` containing only the records from `countyData` where `courseType` is `AP CS` or `Non-AP CS` . Use a `filter` to do this. Recall that the boolean **OR** operator is `||` in TypeScript. 2. Make a new chart that looks the same, except it shows data from your new array `csEnrollments` . That is, you can copy the old chart specification and only change the `values` field for `data` . 3. Add a an empty `div` tag to your HTML document below the `overall-chart` tag, and use `vegaEmbed` to embed your new chart in the HTML page. Recall that the div is initially empty because we are going to use `vegaEmbed` to populate it with the figure.

After you do these steps, hit the **Run** button. You should see **two charts** in the web page. The new chart should look like this:



Race and CS Course type

Task 3: Reflect and answer questions

Study the two charts that are in your HTML page. What are some differences you notice? Together what "story" do the two charts tell about who's taking (or not taking) CS courses in California high schools?

Add a `<p>` tag (paragraph) to your HTML page below the two charts and answer the question above in your own words and in a few sentences.

Part 2

Add an `<h2>` tag with the text "Part 2" below your reflection. This helps to organise the content in your HTML page.

The two charts above tell a story. When looking at *all* enrollments, we can see that Latinx students form the highest proportion of students in California high schools, but the "gap" between Latinx students and the next two most frequent categories (White and Asian) students, seems to reduce when one looks at CS enrollments in particular.

Let's dig into this a bit deeper.

Right now, we can roughly glean some insight from a combination of the two figures, but really what we would like is a figure that tells us about *rates* (i.e., percentages) instead of absolute counts.

That is, we would like to draw a chart where each bar represents a *percentage of students taking CS courses*.

For example, just eyeballing the two existing charts, we can see that

- Roughly 1 million (1000000) Latinx students are enrolled in California high schools
- Roughly 40 thousand (40000) Latinx students are enrolled in CS courses

We can therefore say that roughly 4% of Latinx students are taking CS courses (since `40000 / 1000000 = 0.04`).

This involves a number of sub-steps. Drawing a barchart containing percentages involves some steps of data preparation.

Task 1: Data preparation

At the end of this series of steps, we'd like to have an array of objects that look like this:

```
{
  Race: string,
  totalOverallEnrollment: number,
  totalCSEnrollment: number,
  percentCSEnrollment: number
}
```

So for example, one of the objects in the array would be:

```
{
  Race: 'Latinx',
  totalOverallEnrollment: 1000000,
  totalCSEnrollment: 40000,
  percentCSEnrollment: 0.04
}
```

Note that the numbers above are just approximations.

Our resulting array should have exactly 8 objects, one for each race in our dataset.

Step 1. Write a reusable function to compute the total enrollments for a given race.

Write a function called `getTotalEnrollmentForRace`.

- It should have two inputs:
  - a `string` representing the race for which you're computing total enrollment
  - an array of records (`County[]`) representing the dataset to work with.
- It should output a `number` representing the total number of students of the specified race.

So if you call `getTotalEnrollmentForRace('Latinx', overallEnrollment)`, you should get a number close to `100000` in return. If you call `getTotalEnrollmentForRace('Latinx', csEnrollment)`, you should get a number `40000` in return.

You should be able to accomplish this task using the following the substeps:

- a `filter` to filter based on the input `race` variable.

- a `map` to go from each remaining record to its `totalStudents` value.
- a `reduce` to sum up the `totalStudents` values.

If you're using `for` loops for the above, you may be able to do the `map` and `reduce` steps in a single loop.

Notice that we're using the `overallEnrollment` and `csEnrollment` arrays you created in Part 1. If we hadn't done those steps, you would have had to use an additional filter step to filter by `courseType`.

Step 2. Prepare data for visualisation

Now that we have our `getTotalEnrollmentForRace` function, let's use it to compute the following for each race: * total number of students overall * total number of students in CS courses

Then we can use *those* numbers to compute the percentage of students within each group that are enrolled in CS courses.

Recall that we eventually want to end up with an array of objects that look like this:

```
{
  "Race": "Latinx",
  "totalOverallEnrollment": 100000,
  "totalCSEnrollment": 40000,
  "percentCSEnrollment": 0.04
}
```

Define an `interface` to describe the data above. Call it `EnrollmentPercentByRace` (or whatever makes the most sense to you). Then, use this new interface as the type for a *new empty array* called `enrollmentPercentByRace`.

We will store our results in this array.

Step 4. Compute totals and percentages

We need to compute enrollment percentages *for each* of the items in the following list:

```
const races: string[] = ['African American', 'Asian', 'Filipino', 'Latinx', 'Mu
```

Sounds like a job for a `for-of` loop! We're going to loop over the list of race above, and for each one, compute the total overall enrollment, the total CS enrollment, and the percentage of CS enrollment.

Here are the steps for you complete this task (written in plain English):

Loop over the list of `races`. For each item in the list, do the folowing: * Compute the total overall enrollment. * Compute the total CS enrollment. * Compute the percentage of students enrolled in CS (total CS enrollment / total overall enrollment) * Construct a new object with the fields `Race`, `totalOverallEnrollment`, `totalCSEnrollment`, `percentCSEnrollment`, and give those

fields the appropriate values that you just computed. * `push` the new object into the array you had prepared earlier (`enrollmentPercentByRace`).

After this `for-of` loop, you should have an array of `EnrollmentPercentByRace` objects. The array should contain exactly 8 items (one for each unique value of `Race`).

Step 5. Prepare and draw the chart.

We want to draw a bar chart just like before, but this time instead of plotting the total number of students, we want to plot the *percentage* within each race. Go ahead and prepare the chart.

Again, you can use the previous charts as a starting point. Be sure to update the following:

- The `values` field for the `data` . You should now use your new array (`enrollmentPercentByRace`).
- The `y` encoding should be based on the `percentCSEnrollment` field.
- There should not be an `aggregate` in the `y` encoding any longer—think about why this is and discuss with your tablemates.

The `x` encoding can stay the same as the previous charts, since we're plotting by `Race` .

Make sure to set chart and axis titles appropriately.

Step 6. Reflect

Add a `<p>` tag below the newest chart. Then reflect on and answer the questions below. Write 150–250 words.

- Based on these figures, do you think that students of different backgrounds are well-represented in high school CS courses?
- Which groups do you think are under-represented or over-represented?
- What are the implications of this on representation in college CS courses, and in the information technology workforce?

Part 3

So far, we've explored slices of the data based on a single demographic category (race). But the intersections of different demographic categories is often where the more nuanced analysis can take place.

For this last part, produce a third figure including *at least two* demographic categories (choosing from race, gender, course type, or county rural/urban status), and include a `<p>` tag reflecting on your interpretation of the figure.

You can discuss and collaborate with your classmates for this, but each student must submit individually.

---

1. https://csforca.org↩

2. Notice that this uses `fetch` instead of `fs.readFileSync` like the code in Lab 5 did. Remember

what I said about code running in a web page not having direct access to the filesystem? `fetch` accesses files “over the internet”. This means that only files that are made “visible” by their owners can be accessed. In this case, the `dataset.json` file is “visible” to this webpage, so it can be accessed through `fetch`.↩