Checking Your Work

Students consider the concept of trust and testing — how do we know if a particular analysis is trustworthy?

Prerequisites	None
Relevant Standards OK K12CS CSTA NGSS	*Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).*
Lesson Goals	Students will be able to... - Create a subset of data to verify that a given transformation works as-advertised, using attributes of the transformation and the dataset.
Student-facing Lesson Goals	• Let's learn how to test the trustworthiness of a data analysis.
Materials	• Lesson Slides (Google Slides) • Computer for each student (or pair), with access to the internet • Student workbook, and something to write with
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs • Make sure all students can access the Trust-but-Verify Starter File
Supplemental Resources	
Language Table	

Types	Functions	Values
Number	`num-sqrt, num-sqr, mean, median, modes`	`4, -1.2, 2/3`
String	`string-repeat, string-contains`	`"hello", "91"`
Boolean	`==, <, <=, >=, string-equal`	`true, false`
Image	`triangle, circle, star, rectangle, ellipse, square, text, overlay, bar-chart, pie-chart, bar-chart-summarized, pie-chart-summarized, histogram`	●▲◈
Table	`count, .row-n, .order-by, .filter, .build-column`	

Confirming Analysis

30 minutes

Overview

Students learn how to create a *Testing Table* , which is small enough to reason about and can be used to test whether code does the right thing.

Launch

Samples are taken in Data Science and Computer Programming for two different reasons. One of the main purposes of Data Science is to take a representative sample from a larger population, and use information from the sample to infer what's true about the whole population. In programming, we often extract a smaller Table from a larger one, for the purpose of testing that our code seems to do what it's supposed to. In this lesson, we focus on the tasks of programmers, and consider best practices for setting up a Testing Table that helps us check our code.

- Uber and Google are making self-driving cars, which use artificial intelligence to interpret sensor data and make predictions about whether a car should speed up, slow down, or slam on the brakes. This AI is trained on a lot of sample data, which it learns from. What might be the problem if the sample data only included roads in California?
- Law enforcement in many towns has started using facial-recognition software to automatically detect whether someone has a warrant out for their arrest. A lot of facial-recognition software, however, has been trained on sample data containing mostly white faces. As a result, it has gotten really good at telling white people apart, but often can't tell the difference between people who aren't white. Why might this be a problem?
- Why might it be a bad thing to only test medicines only on men (or only on women), before prescribing them to the general public?

Testing Matters!

A good Testing Table should be *representative* of the population, and *relevant* to what's being analyzed. A good Testing Table should have...

- *At least* the columns that matter — whether we'll be ordering or filtering by those columns.
- Enough rows to include different circumstances that are relevant to the task at hand. For instance, if our code is supposed to extract certain cats from the animals table, our Testing Table should include at least one animal that's not a cat.
- Rows that aren't already sorted, if our analysis is supposed to sort for us.

Data scientists usually think in terms of samples that best serve the purpose of *performing inference* : Samples should be representative of the entire population, and large enough to get us fairly close to the truth about that population. Computer programmers need to think in terms of *Testing Tables* that best serve the purpose of verifying that their code does what it's supposed to: The Tables should be designed to call attention to any imperfections in the code's instructions.

Investigate

Testing Tables can also be used to *verify* that a certain analysis is correct. A function that is supposed to filter a table and *show only the cats* can't be tested with a Testing Table that only has cats to begin with. How would we know if the function filters out non-cats?

Suppose a function takes in a table of animals and shows *only the kittens* . A Testing Table should have cats and non-cats, as well as old and young cats.

Suppose a function takes in a table of animals and shows only the kittens, sorted in ascending order by weight. Now a Testing Table should have cats and non-cats, as well as old and young cats... *and* have rows that aren't already sorted!

- Turn to "Trust, but verify ..." (Page 67) in your student workbook.
- You've been given a function called `fixed-cats` and a description of what it *claims* to do.
- List the names of the animals that you would use in a Testing Table to verify whether the function works as advertised. When you've finished, open the Trust-but-Verify Starter File. There are three versions of `fixed-cats` here. Are they all correct? If not, which ones are broken?
- Turn to "Trust, but verify..." (Page 68). Using the same Starter File, construct a Testing Table and figure out which (if any) of the functions are correct!

Synthesize

Complex analysis has more room for mistakes, so it's critical to think about a Testing Table that allows us to trust that our code really does what it's supposed to!