Project 3

Corrections

- Feb 6: fixed tests for q20 (please re-download test.py)
- Feb 8: clarify that extrapolation is linear (see new example in function 5 description)

Description

The City of Madison has many different agencies providing a variety of services. In this project, you'll analyze real spending data from 2015 to 2018 for five of the largest agencies: police, fire, streets, library, and parks. You'll get practice calling functions from a `project` module, which we'll provide, and practice writing your own functions.

Start by downloading `project.py`, `test.py` and `madison.csv`. Double check that these files don't get renamed by your browser (by running `ls` in the terminal from your `p3` project directory). You'll do all your work in a new `main.ipynb` notebook that you'll create and hand in when you're done. You'll test as usual by running `python test.py`.

We won't explain how to use the `project` module here (the code in the `project.py` file). The lab this week is designed to teach you how it works, so be sure to do the lab from home (if you missed it) before starting the project.

This project consists of writing code to answer 20 questions. If you're answering a particular question in a cell in your notebook, you need to put a comment in the cell so we know what you're answering. For example, if you're answering question 13, the first line of your cell should contain `#q13`.

Dataset

The data looks like this:

agency_id	agency	2015	2016	2017	2018
11	police	68.06346877	71.32575615000002	73.24794765999998	77.87553504
22	fire	49.73757877	51.96834048	53.14405332	55.215007260000014
33	library	16.96543425	18.12552139	19.13634773	19.845065799999997
44	parks	18.371421039999998	19.159243279999995	19.316837019999994	19.7607100000000
55	streets	25.368879940000006	28.2286218	26.655754419999994	27.798933740000003

The dataset is in the `madison.csv` file. We'll learn about CSV files later in the semester. For now, you should know this about them:

- it's easy to create them by exporting from Excel
- it's easy to use them in Python programs
- we'll give you a `project` module to help you extract data from CSV files until we teach you to do it directly yourself

All the numbers in the dataset are in millions of dollars. Answer questions in millions of dollars unless we specify otherwise.

Requirements

You may not hardcode agency IDs in your code. For example, if we ask how much was spent on streets in 2015, you could obtain the answer with this code: `get_spending(get_id("streets"), 2015)`. If you don't use `get_id` and instead use `get_spending(55, 2015)`, we'll deduct points.

For some of the questions, we'll ask you to write (then use) a function to compute the answer. If you compute the answer without creating the function we ask you to, we'll manually deduct points from the `test.py` score when recording your final grade, even if the way you did it produced the correct answer.

Questions and Functions

Q1: What is the agency ID of the library agency?

Hint: use `project.get_id("library")`

Q2: How much did the agency with ID 44 spend in 2018?

It is OK to hardcode `44` in this case since we asked directly about agency 44 (instead of about "parks").

Q3: How much did "streets" spend in 2017?

Hint: instead of repeatedly calling `project.get_id("streets")` (or similar) for each function, you may wish to make these calls once at the beginning of your notebook and save the results in variables, something like this:

```
streets_id = project.get_id("streets")
police_id = project.get_id("police")
fire_id = project.get_id("fire")
...
```

Function 1: `year_max(year)`

This function will compute the maximum spending of any one agency in a given year. We'll give this one to you directly (you'll have to wriet the code for the subsequent functions yourself). Copy/paste this into a cell in your notebook:

```
def year_max(year):
    # grab the spending by each agency in the given year
    police_spending = project.get_spending(project.get_id("police"), year)
    fire_spending = project.get_spending(project.get_id("fire"), year)
    library_spending = project.get_spending(project.get_id("library"), year)
    parks_spending = project.get_spending(project.get_id("parks"), year)
    streets_spending = project.get_spending(project.get_id("streets"), year)

    # use builtin max function to get the largest of the five values
    return max(police_spending, fire_spending, library_spending, parks_spending,
streets_spending)
```

Q4: What was the most spent by a single agency in 2015?

Use `year_max` to answer this.

Q5: What was the most spent by a single agency in 2018?

Function 2: `agency_min(agency)`

We'll help you start this one, but you need to fill in the rest yourself.

```
def agency_min(agency):
    agency_id = project.get_id(agency)
    y15 = project.get_spending(agency_id, 2015)
    y16 = project.get_spending(agency_id, 2016)
    # grab the other years

    # use the min function (similar to the max function)
    # to get the minimum across the four years, and return
    # that value
```

This function will compute the minimum the given agency ever spent over the course of a year.

Q6: What was the least the police ever spent in a year?

Use your `agency_min` function.

Q7: What was the least that fire ever spent in a year?

Q8: What was the least that library ever spent in a year?

Function 3: `agency_avg(agency)`

This function will compute the average (over the four datapoints) that the given agency spends per year.

Hint: start by copy/pasting `agency_avg` and renaming your copy to `agency_avg`. Instead of computing the minimum of `y15`, `y16`, etc., compute the average of these by adding, then dividing by 4.

Q9: How much is spent per year on streets, on average?

Use your `agency_avg` function.

Q10: How much is spent per year on parks, on average?

Q11: How much did the police spend above their average in 2018?

You should answer by giving a percent between 0 and 100, with no percent sign. In this case, your code should produce a number close to `7.2249619343519145`.

Function 4: `change_per_year(agency, start_year, end_year)`

This function returns the average increase in spending (could be negative if there's a decrease) over the period from `start_year` to `end_year` for the specified `agency`.

You can start from the following code:

```
def change_per_year(agency, start_year=2015, end_year=2018):
    pass # TODO: replace this line with your code
```

Python requires all functions to have at least one line of code. When you don't have some code, yet, it's common for that line to be `pass`, which does nothing. Note the default arguments above.

Q12: how much has spending increased per year (on average) for police from 2015 to 2018?

Use the default arguments (your call to `change_per_year` should only pass one argument explicitly).

Q13: how much has spending increased per year (on average) for police from 2017 to 2018?

Use the default argument for the `end_year` parameter (your call to `change_per_year` should only pass two arguments explicitly).

Q14: how much has spending increased per year (on average) for streets from 2016 to 2018?

Note that a negative answer is expected (a negative increase represents a decrease).

Function 5: `extrapolate(agency, year1, year2, year3)`

This function computes the average change per year from the data from `year1` to `year2` for `agency`. It then returns the predicted spending in `year3`, assuming spending continues changing by the same constant amount each year. We don't have anything for you to copy for

this one (you need to write it from scratch).

As an example, suppose spending in 2016 (year1) is 100 and spending in 2018 (year2) is 120. The average increase is 10 per year. So we would extrapolate to 130 for 2019, 140 for 2020, etc. This kind of prediction is a simple *linear extrapolation*.

Q15: how much will library spend in 2019?

Extrapolate to 2019 from the data between 2015 and 2018.

Q16: how much will library spend in 2100?

Extrapolate from the data between 2015 and 2018.

Q17: how much will library spend in 2100?

Extrapolate from the data between 2017 and 2018.

Function 6: `extrapolate_error`

We can't know how well our simple extrapolations will perform in the future (unless we wait 80 years), but we can do shorter extrapolations to years for which we DO know the result. For example, we can extrapolate to 2018 from the 2015-to-2017 data, then compare our extrapolation to the actual spending in 2018.

Write a function named `extrapolate_error` that does an extrapolation using the `extrapolate` function and compares the extrapolation to the actual result, returning the error (i.e., how much `extrapolate` overestimated). For example, if the extrapolation is 105 and the actual is 110, then the function should return -5.

What parameters should `extrapolate_error` have? That's your decision!

Q18: what is the error if we extrapolate to 2018 from the 2015-to-2017 data for police?

Q19: what is the error if we extrapolate to 2018 from the 2015-to-2016 data for streets?

Q20: what is the error if we extrapolate to 2018 from the 2015-to-2017 data for streets?