4. Review of Python and pandas

Unlike most chapters, there are no slides corresponding to this chapter, because they consist mostly of in-class exercises. They aim to help you remember the Python and pandas you learned in CS230 and be sure they're refreshed and at the front of your mind, so that we can build on them in future weeks.

4.1. Python review 1: Remembering pandas

This first set of exercises works with a database from the U.S. Consumer Financial Protection Bureau. The dataset recorded all mortgage applications in the U.S. in 2018, over 15,000,000 of them. Here we will work with a sample of about 0.1% of that data, just over 15,000 entries. These 15,000 entries are randomly sampled from just those applications that were for a conventional loan for a typical home purchase of a principal residence (i.e., not a rental property, not an office building, etc., just standard house or condo for an individual or family).

Download the dataset as a CSV file here. If you have questions about the meanings of any column in the dataset, they are fully documented on the government website from which I got the original (much larger) dataset.

Exercise 1

In class, we will work independently to perform the following tasks, using a cloud Jupyter provider such as Deepnote or Colab.

1. Create a new project and name it something sensible, such as "MA346 Practice Project 1."
2. Upload the data file into the project.
3. Start a new Jupyter notebook in the same project and load the data file into a pandas DataFrame in that notebook.
4. Explore the data using pandas's built-in `info` and/or `head` methods.
5. The dataset has many columns we won't use. Drop all columns except for `loan_amount`, `interest_rate`, `property_value`, `state_code`, `tract_minority_population_percent`, `derived_race`, `derived_sex`, and `applicant_age`.
6. Reading a CSV file does not always ensure that columns are assigned the correct data type. Use pandas's built-in `astype` function to correct any columns that have the wrong data type.
7. Practice selecting just a subset of the DataFrame by trying each of these things:
    - Define a new variable `female_applicants` that contains just the rows of the dataset containing mortgage applications from females. How many are there? What are the mean and median loan amounts for that group?
    - Repeat the previous bullet point, but for Asian applicants, stored in a variable named `asian_applicants`.
    - Repeat the previous bullet point, but for applicants whose age is 75 or over, stored in a variable `applicants_75_and_older`.
8. Make your notebook presentable, using appropriate Markdown comments between cells to explain your code. (Chapter 5 will cover best practices for how to write such comments, but do what you think is best for now.)
9. Use Deepnote or Colab's publishing feature to create a shareable link to your notebook. Paste that link into our class's Microsoft Teams chat, so that we can share our work with one another and learn from each other's work.

Learning on Your Own - Basic pandas work in Excel

Investigate the following questions. A report on this topic would give complete answers to each.

- Which of the tasks in Exercise 1 are possible to do in Excel and which are not?
- For those that are possible in Excel, what steps does the user take to do them?
- Will the resulting Excel workbook continue to function correctly if the original data changes?
- Which steps are more convenient in Excel and which are more convenient in Python and pandas, and why?

Contents

4.2. Adding a new column

As you may recall from CS230, you can add new columns to a pandas DataFrame using code like the example below. This example calculates how much interest the loan would accrue in the first year. (This is not fully accurate, since of course the borrower would make some payments that year, but it's just an example.)

```
df['interest_first_year'] = df['property_value'] * df['interest_rate'] / 100
```

Running this code in the notebook you've created would work just fine, and would create that new column. It would have missing values for any rows that had missing property values or interest rates, naturally, but it would compute correct numerical values in all other rows.

But what happens if you try to run the same code, but just on the `female_applicants` DataFrame (or `asian_applicants` or `applicants_75_and_older`)?

Big Picture - Writing to a slice of a DataFrame

The warning message you see when you attempt to run the code described above is an important one! It relates to the difference between a DataFrame and a *view* of that DataFrame. You can add columns to a DataFrame, but if you add to just a view, you'll receive a warning. We will discuss the details of this in class.

4.3. What if you don't remember CS230 very well?

I have several recommendations of resources you can use:

4.3.1. DataCamp

I will regularly assign you exercises from DataCamp, some of which will review CS230 materials. If you remember everything from CS230, the first few weeks of these exercises should be easy and quick for you. If not, you will need to put in more time, but it will help you catch up.

4.3.2. Your instructor

I'm glad to meet with students who need help catching up on material from CS230 they may not remember. Please feel free to come to office hours!

4.3.3. Stack Overflow

The premiere question and answer website for technical subjects is Stack Overflow. You don't need to visit the site, though; if you do a good Google search for any specific Python or pandas question, one of the top hits will almost alway be from Stack Overflow. Here are a few tips to using it well:

- When you do a search, put as many specific words related to your question as possible.
  - Be sure to mention Python, pandas, or whatever other libraries your question might touch upon.
  - If your question is about an error message, put the specific key words from the error message in your search.
- When viewing questions and answers on Stack Overflow, don't read only the top answer. A lower-ranked answer might actually be more suited to your specific needs.

4.3.4. O'Reilly books

You have free access to O'Reilly Online Learning through the Bentley Library. They are one of the top publishers of high-quality tutorial books on technical subjects. To get started, visit this page and at the bottom choose to download a mobile app for your phone or tablet.

Then browse their book catalog and see what looks like it might be good for you. I recommend starting here:

- Python Data Science Handbook by Jake VanderPlas, chapter 3 (or perhaps start earlier if you need to)
- Python for Data Analysis by Wes McKinney, chapter 5 (or perhaps start earlier if you need to)

4.3.5. Official documentation

Official documentation is used mostly for reference. It does not make a good tutorial or lesson. But it is the definitive reference, so I mention it here.

- Python official documentation
- pandas official documentation

4.4. Python review 2: mathematical exercises

As before, do these exercises in a new notebook in Deepnote or Colab, and when you're done, share the link to the published version into our class's Teams chat.

Exercise 2

If r is the annual interest rate and P is the principal, we're all familiar with the standard formula for the present value after n periods, $P(1+r)^n$. Write this as a Python function. Also consider:

1. How many inputs does it take and what are their data types?
2. What is the data type of its output?
3. Evaluate your function on $P = 1,000$, $r = 0.01$, and $n = 7$. Ensure you get approximately $1,072.14.

Exercise 3

Create a pandas DataFrame with two columns. The first column should be entitled F for Fahrenheit, and should contain the numbers from 0 to 100, counting by fives. The next column should be entitled C for Celsius, and contain the corresponding temperature in degrees Celsius for the number in the first column. Display the resulting table in the notebook.

Now try changing your work so that the result is a single pandas Series whose *index* is the Fahrenheit temperatures, and whose *values* are the Celsius temperatures.

Exercise 4

The NumPy function `np.random.randint(a,b)` picks a random integer between a and $b-1$. Use that to create a function that behaves as follows:

- Your function takes as input a positive integer n, how many times to "roll the dice."
- Each roll of the dice simulates two dice being rolled (each with a number from 1 to 6) and adds the results together (thus generating a number between 2 and 12).
- After all n rolls, return a pandas DataFrame with three columns:
    1. the numbers 2 through 12
    2. the number of times that number showed up
    3. the percentage of the time that number showed up
- Ensure the resulting DataFrame is sorted by its first column.

4.5. Functional-style code vs. imperative-style code

As you wrote the functions above, you might have found yourself falling into one of two styles. To see examples of each style, let's consider the definition of the statistical concept of *variance*. The variance of a list of data $x_1, \ldots, x_n$ is defined to be

$$\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1},$$

where we write $\bar{x}$ to mean the mean of the data, and we pronounce it "x bar." If we take that function and convert it directly into Python, we might write it as follows.

```
import numpy as np

# Coding style #1, "functional"
def variance ( data ):
    return sum( [ ( x - np.mean(data) )**2 for x in data ] ) / ( len(data) - 1 )

test_data = [ 5, 10, 3, 9, -1, 5, 3, 1 ]
variance( test_data )
```

```
13.982142857142858
```

Although this function computes the variance of a list of data correctly, it piles up a lot of parentheses and brackets that some readers find unnecessarily confusing when reading code. We can make the function less compact and more explanatory by breaking the nested parentheses into several different lines of code, each storing its result in a variable. Here is an example.

```
# Coding style #2, "imperative"
def variance ( data ):
    n = len(data)
    xbar = np.mean( data )
    squared_differences = [ ( x - xbar )**2 for x in data ]
    return sum( squared_differences ) / ( n - 1 )

variance( test_data )
```

```
13.982142857142858
```

I call the first one *functional style* because we're composing a lot of functions, each inside another. I call the second one *imperative style* because the term "imperative" is used in programming to refer to lines of code that give the computer a command; here we've broken the formula out into three separate commands to create variables, followed by the final formula.

Neither of these two styles is always better than the other. For a short formula, you probably just want to use functional style. But for a long formula, imperative style has these advantages:

- You can use good, descriptive variable names to clarify for the reader of your code what it's computing in each step.
- If the code you're writing isn't inside a function, you can split imperative-style code over multiple cells, and put explanations in between.
- If you know the reader of your code is new to coding (such as a new teammate in your organization) then imperative style gives them small pieces of code to digest one at a time, rather than a big pile of code they must understand all at once.

So consider using each style for those situations that it fits best.