Lab 8: Pandas and Plotting[1]

Part 0: Submission/Reference Materials

Remember to read (and write!) good documentation and use the internet to find code examples. Finding and using appropriate/accurate reference materials is hard to teach directly, but tends to be what separates bad scientific programmers from good ones!

If you get stuck:

- Review the lecture material
- Check the docstrings of functions you are trying to use (hint: use ?function in jupyter)
- The pandas official documentation includes all of the commands you need to complete this week's practical https://pandas.pydata.org/docs/user_guide/index.html#user-guide
- Use online resources like stackoverflow, w3schools, realpython etc. (but make sure you don't blindly copy code without working out HOW it works).
- Looking up materials is totally fine but remember if you copy code (or autocomplete it) directly from any source, you MUST cite where you got it from in a comment next to the code.

Submit this assignment as a formatted notebook - include an explanation of your answers and make sure every function has a clear docstring that explains what it does, its arguments, and what it returns. This will be part of the grading of each of your answers.

Part 1: Set-Up and Data Wrangling

In today's assignment you are going to parse, clean, explore, and visualise a dataset from a study in January 2018 that measured the average number of steps taken that month compared to their age and income. This is an open simulated dataset so there are no restrictions on how it is used (despite it containing sensitive personal information which would normally require research ethics and/or restrictions). This sort of dataset could be collected by a gerontology researcher looking at activity levels as we age or an insurance company investigating whether step-trackers could be used to incentivise discounts on premiums.

However, before we play with that data we are going to make sure everything works and make some basic DataFrames from scratch.

```
%matplotlib inline
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[1] Modified from UCSD COGS108 Assignment 2

```
# Round decimals when displaying DataFrames
pd.set_option('display.precision', 2)
```

Q1 [3 points]. Create a DataFrame called `fruits` that looks like this using at least 2 different approaches (e.g., with and without creating `pd.Series` first).

	Apples	Bananas
2017 Sales	35	21
2018 Sales	41	34

Q2 [4 points]: Create a DataFrame entirely within Python that represents measurements from an experiment with:

- 5 samples (sample_id: A1, A2, A3, A4, A5)
- 3 conditions (control, treatment1, treatment2)
- Measurement values should be random values between 10 and 50 (hint: look at the documentation for np.random)

Now that that is out of the way, for the rest of this assignment you will be working with two data files that contain information on people's step counts. The two files and their columns (also called "fields") are:

- age_steps.csv: Contains one row for each person.
    - id: Unique identifier for the person.
    - age: Age of the person.
    - steps: Number of steps the person took on average in January 2018.

- incomes.json: Contains one record for each person.
    - id: Unique identifier for the person. Two records with the same ID between age_steps.csv and incomes.json correspond to the same person.
    - last_name: Last name of the person.
    - first_name: First name of the person.
    - income: Income of the person in 2018.

Download this data using the following URL:
https://maguire-lab.github.io/scientific_computing/static_files/practicals/lab8_data.zip

Unzip the lab8_data.zip into the folder containing your lab practical notebook for this week.

Q3 [2 points]: Load the age_steps.csv file into a pandas DataFrame named df_steps. Print out the number of rows and columns in `df_steps`

Q4 [2 points]: Load the incomes.json file into a pandas DataFrame called df_income. Print out the number of rows and columns in `df_income`

Q5 [2 points]: Drop the first_name and last_name columns from the df_income DataFrame. The resulting DataFrame should only have two columns. Remember you can use the keyword-arg `axis` with most `pd.DataFrame` and `pd.Series` class methods to control whether it works on the rows or the columns.

Q6 [6 points]: Merge the df_steps and df_income DataFrames into a single combined DataFrame called df using the id column to match rows together. The final DataFrame should have 10,135 rows and 4 columns: id, income, age, and steps. Using `assert` statements write code that checks your merged DataFrame contains the correct number of rows/columns and that the column names are correct. This (and all future questions can generally be done with 1 or 2 built-in pandas methods); try to avoid writing for loops as they are quite inefficient for DataFrame operations.

Q7 [2 points]: Reorder the columns of df so that they appear in the order: id, age, steps, then income. (hint: you can select several columns by passing a list of names to the DataFrame).

Q8 [3 points]: You may have noticed something strange: the merged `df` DataFrame has fewer rows than either of `df_steps` and `df_income`. Why did this happen? (If you're unsure, check out the documentation for the `pandas` method you used to merge these two datasets. Take note of the default values set for this method's parameters.)

Please select the **one** correct explanation below and explain it in a comment along with code that justifies your answer?

1. Some steps were recorded inaccurately in `df_steps`.
2. Some incomes were recorded inaccurately in `df_income`.
3. There are fewer rows in `df_steps` than in `df_income`.
4. There are fewer columns in `df_steps` than in `df_income`.
5. Some `id` values in either `df_steps` and `df_income` were missing in the other DataFrame.
6. Some `id` values were repeated in `df_steps` and in `df_income`.

Part 2: Data Cleaning

Once you've wrangled your dataset into a single DataFrame object then comes the second major challenge: data cleaning. This dataset is already relatively clean but there are still some issues for you to fix.

The most common issue to deal with is missing values. There are many reasons data might contain missing values. Here are two common ones:

- **Nonresponse.** For example, people might have left a field blank when responding to a survey, or left the entire survey blank.
- **Lost in entry.** Data might have been lost after initial recording. For example, a disk cleanup might accidentally wipe older entries of a database.

In general, it is **not** appropriate to simply drop missing values from the dataset or pretend that if filled in they would not change your results. Systematic biases in which data is missing (this is a type of selection bias called ascertainment or sampling bias) are a very common reason why analyses are misleading. For example, psephologists have mispredicted election results by not correcting for the fact that supporters of certain parties are less likely to respond to polls. In this particular dataset, however, the **missing values occur completely at random**. This criteria allows us to drop missing values without significantly affecting our conclusions.

Q10 [2 points]: How many values are missing in the `income` column of `df`?

Q11 [2 points]: Create a new dataframe `clean_df` in which all rows from `df` that have missing values have been removed.

Q12 [2 points]: We can now compute the average income. If your `clean_df` contains the right values, `clean_df['income'].mean()` should produce the value `25508.84`. Write an assert statement that makes sure this is true. Suppose that we didn't drop the missing incomes. Compare the `clean_df['income'].mean()` and `df['income'].mean()` outputs and explain which of the following is occurring?

1. No change; `df['income'].mean()` will ignore the missing values and output `25508.84`.
2. `df['income'].mean()` will produce an error.
3. `df['income'].mean()` will output `0`.
4. `df['income'].mean()` will output `nan` (not a number).
5. `df['income'].mean()` will fill in the missing values with the average income, then compute the average.
6. `df['income'].mean()` will fill in the missing values with `0`, then compute the average.

Q13 [2 points]: Suppose that missing incomes did not occur at random, and that individuals with incomes below $10000 a year are less likely to report their incomes. If so, which of the following statements below would be true?

1. `clean_df['income'].mean()` will likely output a value that is the same as the population's average income
2. `clean_df['income'].mean()` will likely output a value that is smaller than the population's average income.
3. `clean_df['income'].mean()` will likely output a value that is larger than the population's average income.
4. `clean_df['income'].mean()` will raise an error.

Part 3: Data Visualization

Although `pandas` only displays a few rows of a DataFrame at a time, we can use data visualizations to quickly determine the **distributions** of values within our data.

Helpfully `pandas` comes with some plotting capabilities built-in accessed via DataFrame.plot (look at [https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html for examples). This functionality is built on top of `matplotlib`. We will use some matplotlib functions to tweak our plots.

Q14 [6 points]: Plot a histogram of the `age` column with 25 bins and set appropriate axis labels and title using the following `matplotlib.pyplot` functions.

```
plt.xlabel("STRING WITH X-AXIS LABEL")
plt.ylabel("STRING WITH Y-AXIS LABEL")
plt.title("STRING WITH PLOT TITLE")
```

By looking at the plot, what is the approximate average (mean) age of the individuals in `clean_df`? How different is your visually estimated value from the actual mean age calculated directly from the `clean_df?`

Q15 [6 points]: Plot a histogram of the `steps` column with 25 bins with clear labels. By looking at the plot you just generated, approximately how many people in `clean_df` do the data suggest took no steps? Calculate and compare your estimate to the actual value in the DataFrame.

Q16 [6 points]: Plot a histogram of the `income` column with 25 bins. Which of the following statements is true about the income of the individuals included in `clean_df`? Include code that justifies your answer. (*Note*: Be sure to consider the bin size of the x-axis when interpreting the plot.)

- A) Most people in `clean_df` had no income in 2018
- B) Most people in `clean_df` made a six figure salary in 2018
- C) Most people in `clean_df` made a salary of more than 200000 USD in 2018
- D) A few people in `clean_df` made a lot more money than the typical person in `clean_df`

Q17 [6 points]: Plot the data in the age, steps, and income columns using the `pandas scatter_matrix` function. Explain what this type of plot shows and determine the approximate age of the wealthiest person in `clean_df` and compare it to the real value calculated directly from the DataFrame.

Part 4: Data Pre-Processing

In the above sections, we performed basic data cleaning and visualization. In practice, these two components of an analysis pipeline are often done iteratively. We go back and forth between looking at the data, checking for issues, and cleaning the data.

Let's continue with an iterative procedure of data cleaning and visualization, addressing some issues that we noticed after visualizing the data.

Q18 [3 points]: In the visualization of the `steps` column, we notice a large number of `-1` values. Count how many rows in `clean_df` have `-1` in their `steps` column. Since it's impossible to walk a negative number of steps, we will treat the negative values as missing data. Drop the rows with negative steps from `clean_df`. Your answer should modify `clean_df` itself.

You may have noticed that the values in `income` are not normally distributed which can hurt prediction ability in some scenarios. To address this, we will perform a log transformation on the `income` values. First though, we will have to deal with any income values that are 0. Note that these values are not impossible values — they may, for example, represent people who are unemployed. So, we shouldn't remove these individuals; however, when we go to log-transform these data, we can't (mathematically) have any zero values. We'll replace the zeroes with ones, to allow for log transformation, while retaining the fact that these individuals' income was lower than others in the dataset.

Q19 [6 points]: Add a new column to `clean_df` called `income10`. It should contain the same values as `income` with all `0` values replaced with `1`. Now, transform the `income10` column using a log-base-10 transform using the `np.log10` function. Finally, plot a histogram for `income10` data after the data transformation.

Part 5: Basic Analyses

Now that we have wrangled and cleaned our data, we can start doing some simple analyses. Here we will explore some basic descriptive summaries of our data, look into the inter-relations (correlations) between variables, and ask some simple questions about potentially interesting subsets of our data.

Q20 [8 points]: Perform the following analyses:

- Use the `describe` `pandas` method to check and print a descriptive summary of the data.
- Using the `quantile` method, how many steps would you have to walk to be in the top 10% of walkers?
- What is the average income for people over the age of 65?
- Using a built-in `pandas` method (check the documentation), calculate the pairwise correlations between all variables. Which variable is most correlated with age (aside from `age` itself)? Which variable is most correlated with income (aside from `income` and `income10`)?