Python Programming Assignment
Using pandas

Swapnil Singh

Instruction

Ensure your code is well-commented and follows proper style conventions. You will be able to do this practice by using tools which were discussed during the class.

In this assignment you will work with two data files (look at the solution file):

- `students.txt`
- `publications.txt`

Part A: Loading and Inspecting the Data

1. Import pandas and check the installed version.
   (a) Import pandas with the usual alias.
   (b) Print the value of `pd.__version__`.
2. Load both data files into DataFrames.
   (a) Use `pd.read_csv` to read `students.txt` into a DataFrame called `students`. (Hint: the default separator is a comma, so you do not need to specify `sep`.)
   (b) Use `pd.read_csv` to read `publications.txt` into a DataFrame called `pubs`. (Hint: this file uses semicolons, so you must specify the correct separator.)
3. Inspect the structure of the DataFrames.
   (a) Use `students.info()` and `pubs.info()` to view basic information.
   (b) Display the first two rows and last two rows of each DataFrame using `.head()` and `.tail()`.
   (c) Print the `index`, `dtypes` and `values` attributes of `students`.

Part B: Cleaning and Transforming the Data

4. Clean up name formatting in the `students` DataFrame.
   (a) Select the `name` column as a Series using `students['name']`.
   (b) Use `Series.str.replace` to:
      - Replace any double spaces `"  "` with a single space.

(c) Use `Series.str.capitalize` to convert each name to "capitalized" form (first letter uppercase, the rest lowercase).

(d) Assign the cleaned Series back to the `students['name']` column.

(e) Verify the change by printing `students.head()`.

5. Handle missing GPA values.

   (a) Use `students['gpa']` and `Series.dtype` to check its current data type.

   (b) Convert the `gpa` column to numeric using `pd.to_numeric`, and assign back.

   (c) Use `students.isnull()` to identify which rows have missing GPA values.

   (d) Compute the mean GPA over all non-missing values using `students['gpa'].mean()`.

   (e) Use `DataFrame.fillna` *on the* `gpa` column only to fill missing GPAs with the mean GPA.

   (f) Confirm there are no remaining missing values in `gpa`.

6. Standardize text in the `department` column (optional mini-cleaning).

   (a) Examine `students['department'].unique()`.

   (b) Use `Series.str.replace` or `Series.str.capitalize` as needed to ensure department names are consistently formatted.

Part C: Selecting, Indexing and Filtering

7. Work with indexes.

   (a) Use `students.set_index('student_id')` to create a new DataFrame `students_by_id` where `student_id` is the index.

   (b) Display `students_by_id.head()` and inspect `students_by_id.index`.

   (c) Access the row for student `'S02'` using `.loc`.

8. Practice column selection.

   (a) From `students`, select only the `name` and `gpa` columns using `students[['name', 'gpa']]` and store in a new DataFrame `name_gpa`.

   (b) Display `name_gpa.head()`.

9. Filter rows based on conditions.

   (a) Using `.loc` and Boolean conditions, select all active students in the `Economics` department.

   (b) From these students, select only the `name`, `start_year` and `gpa` columns.

   (c) Similarly, select all students with GPA strictly greater than 8.5.

Part D: Working with the Publications Data

10. Basic inspection and typing.
    (a) Use `pubs.head()` and `pubs.tail()` to inspect the data.
    (b) Check `pubs.dtypes`.
    (c) Ensure that the `year` column is of an appropriate numeric type using `astype` or `pd.to_numeric`.
11. Filtering and string methods.
    (a) Select only the journal publications using `Series.str.contains` on the `venue_type` column.
    (b) Create a Series containing only the titles of journal publications.
    (c) Use `Series.str.capitalize` on the titles and inspect the result.
12. Counting publications per student.
    (a) Use `pubs['student_id']` to get the student ID Series.
    (b) Create a pivot table counting the number of publications per student using `DataFrame.pivot_table` with `values='pub_id'`, `index='student_id'`, and an appropriate aggregation function (e.g. counting).
    (c) Reset the index of this pivot table using `.reset_index()` so that `student_id` becomes a column again.

Part E: Combining Information

13. Compare students by publication counts and GPA.
    (a) Starting from the pivot table created in Part D, rename the column containing the counts to something meaningful like `'pub_count'` using `df.rename`.
    (b) Create a copy of the `students` DataFrame using `students.copy()`.
    (c) Set `student_id` as the index of both DataFrames (the students copy and the pivot table) using `.set_index('student_id')`.
    (d) Use `.loc` and `DataFrame.T` (transpose) if helpful to manually align and inspect publication counts and GPAs for selected student IDs (e.g. `'S01'`, `'S02'`).
    (e) Compute the mean GPA of students who have at least one publication (you may need to select only those student IDs present in the pivot table).
14. Correlation exploration.
    (a) Construct a DataFrame that has, for each student present in the publications pivot table:
        - their GPA,
        - their number of publications.
    (b) Use `df.corr()` on this DataFrame to compute the correlation matrix between GPA and publication count.
    (c) Briefly interpret the resulting correlation value.

Part F: Plotting and Exporting

15. Simple plots.

    (a) Use `students['gpa'].plot()` to create a simple plot of GPA values (index on the x-axis).

    (b) Use `students.plot()` or `DataFrame.plot` to create a plot that shows GPA by start year (you may wish to select only the relevant columns before plotting).

    (c) Use the publication count DataFrame to create a bar plot (e.g. publication count by student ID).

16. Exporting results.

    (a) Save the cleaned `students` DataFrame to a CSV file called `students_clean.csv` using `df.to_csv`.

    (b) Save the publication count per student to an Excel file called `pub_counts.xlsx` using `df.to_excel`.

Part G: Additional Short Tasks

17. Using `Series.unique`, list all distinct supervisor names.

18. Using `DataFrame.iterrows`, print (or construct a small Series) that maps each student name to their status (e.g. `"alice smith" -> "active"`).

19. Choose two Series (for example, two manually constructed Series of publication counts) and compare them using `Series.equals`.