Exercise: The Impact of Covid-19 on NYS Students

Summary

This exercise uses "school report card" data from the New York State Department of Education (NYSED) to assess the impact of the Covid-19 pandemic on English and math proficiency by elementary and high school students in New York State.

It compares district-level average proficiency rates on standardized English and math exams before and after the acute phase of the pandemic. The years 2018 and 2019 are used for the pre-pandemic era. Most standardized exams were not given during the onset of the pandemic in 2020, and few were given in 2021. Regular testing resumed in 2022, so 2022 and 2023 are used as the post-pandemic era.

Proficiency is measured by the percent of students whose scores are rated as "proficient" on two sets of exams: English and math exams given to all fourth grade students, and two exams taken by high school students to demonstrate competence in English and geometry known as Regents Exams. Results are reported for all students as a group as well as for several subgroups: students identifying as Black or African American; students classified as economically disadvantaged; and students classified as having a disability.

A key goal of the exercise is to give you experience using Python scripts to work with SQL databases directly and via Pandas.

Input Data

The input data is contained in a zip file called **nysed.zip** that will need to be downloaded from the course Google Drive. It contains four SQLite databases, one for each year, which contain tables selected from the full report card databases posted by NYSED. After downloading the zip file, unzip it in the repository for the assignment. You can then delete the zip file to save space. Also, the end of the demo script optionally uses a database of electric grid data from the Energy Information Administration, **eia860.db**, that can be downloaded from the class Google Drive.

Deliverables

There are two deliverables: **filtered.py** and **analyze.py**. The first reads the input databases, selects the appropriate data, and builds a new database called `filtered.db`. The second reads `filtered.db` and generates several figures that show the impact of the pandemic.

Instructions

A. Script filtered.py

1. import `sqlite3`
2. Create a variable called `out_name` that contains the string `filtered.db`, which will be the name of a new database the script creates.
3. Set variable `years` to a list containing the years 2018, 2019, 2022 and 2023. Note that the list does not contain 2020 and 2021 because little or no testing was done in those years.
4. Connect to the new database by setting variable `out_con` to the result of calling `sqlite3.connect()` with argument `out_name`. The `.connect()` call will automatically create an empty database if the file doesn't exist.
5. Start a `with` block using `out_con` as the expression following the `with`. The effect of this will be to start a SQL transaction that will automatically be committed if the `with` block terminates normally and automatically rolled back if an error occurs. We'll use a `with` block each time data is written by the script.
    1. Within the `with` block, add a statement that sets `cur` to the result of calling the `.executescript()` method on `out_con` using a triple-quoted string of SQL commands as an argument. The string

should contain the two commands below. Remember to end each of the statements with a semicolon (`;`).

1. A `DROP TABLE IF EXISTS` statement on table `exams`;
2. A `CREATE TABLE` statement for table `exams` that creates a table with seven columns: `code` with type `VARCHAR` (which will store the district's code number); `name` with type `VARCHAR` (the district's name); `year` with type `INT`; `subgroup` with type `VARCHAR` (which will identify the group of students whose score is being reported); `exam` with type `VARCHAR` (will store the name of the exam); `per_prof` with type VARCHAR (the percent of students achieving proficiency); and `level` with type VARCHAR (which will indicate whether the data is for elementary and middle school, "EM", or high school, "HS"). At the end, add a `UNIQUE` constraint on the combination of `code`, `year`, `subgroup`, `exam`, and `level` to prevent duplicate records from being inserted into the table.

6. After the end of the `with` block, create a variable called `ntot` and set it equal to 0. It will be used to count records added to the database.
7. Proficiency data is stored in three different tables in the input databases: one for EM English (known as ELA for English Language Arts), one for EM math, and one for HS Regents Exams. Also, the EM and HS tables use different column names for the exams being reported. To manage all this, set `exam_tables` to a list consisting of three tuples with the level of the exam, the name of the table where it is found, and the column containing the exam name. The first should consist of `"EM"`, `"Annual EM MATH"`, and `"ASSESSMENT_NAME"`; the second should consist of `"EM"`, `"Annual EM ELA"`, and `"ASSESSMENT_NAME"`; and the third should consist of `"HS"`, `"Annual Regents Exams"`, and `"SUBJECT"`.
8. Start a `for` loop that uses `yr` as the running variable to loop over `years`.
    1. Set variable `infile` to `f"nysed{yr}.db"`.
    2. Set `in_con` to the result of calling `sqlite3.connect()` with argument `infile` to connect to the input database.
    3. Start a `for` that has the tuple `(level,table,col)` as the running variable and loops over `exam_tables`. Within the loop, do the following:
        1. Set `sql` equal to a triple-quoted f-string containing a SQL `SELECT` command as its argument. The command should select columns `ENTITY_CD`, `ENTITY_NAME`, `YEAR`, `SUBGROUP_NAME`, and `{col} AS EXAM`, and `PER_PROF` from table `'{table}'`. Include a `WHERE` clause that specifies that `YEAR` should equal `{yr}` and `ENTITY_CD` should be like `'%0000'` (selects districts or counties) and also not like `'0000%'` (eliminates the counties leaving just the districts).
        2. Print `sql` and check that it looks correct.
        3. Set `cur` to the result of calling the `.execute()` method on `in_con` with `sql` as its argument.
        4. Set variable `rows` to the result of calling the `.fetchall()` method on `cur` to retrieve all the input rows that match the select command.
        5. Start a `with` block with `out_con` as its expression.
            1. Inside the `with` block, set `cur` to the result of calling the `.executemany()` method on `out_con` with two arguments. The first should be a triple-quoted f-string with a SQL `INSERT` command that inserts data into `exams` with a `VALUES` clause that contains `(?,?,?,?,?,?,'{level}')`. There should be 6 question marks, which are placeholders for

the values of `code`, `name`, `year`, `subgroup`, `exam` and `per_prof`. The final part adds the value of the `level` column. The second argument should be `rows`. The call executes the `INSERT` statement repeatedly, once for each row in the input data.

2. Set `nrows` equal to `cur.rowcount`, the count of rows modified by the `.executemany()` call.
3. Print an f-string with a message indicating the year, table and number of rows inserted.
4. Add `nrows` to `ntot` to keep a running count of rows added.

9. After the end of both `for` loops, set `cur` to the value of calling `.execute()` on `out_con` with a SQL statement to count the number of records in `exams`.
10. Create a variable called `check` that is equal to the result of calling `.fetchone()` on `cur`. That will return the row produced by the previous call.
11. The `check` variable is a tuple whose first entry is the row count. Add an `assert` statement that checks to make sure that `ntot` is equal to `check[0]`.
12. Next we'll convert NYSED's marker for suppressed data, an "s", into NULLs. While we're at it, we'll also convert blanks (also missing data) to NULLs. Start another `with out_con` block. Inside it do the following:
    1. Set `cur` to the result of calling `.execute()` on `out_con` with a triple-quoted string giving a SQL `UPDATE` statement for table `exams` that sets `per_prof` to `NULL` where `per_prof` is `'s'` or `per_prof` is `''` (an empty string).
13. After the `with` block, add a print statement that prints an appropriate heading and then the value of `ntot`.
14. Add another print statement with a heading indicating that it is reporting the number of `'s'` and blank values converted to NULL and the prints `cur.rowcount`.
15. Call the `.close()` method on `in_con`.
16. Call the `.close()` method on `out_con`.

B. Script analyze.py

1. Import `pandas`, `sqlite3`, `matplotlib.pyplot`, and `seaborn`.
2. Set `pd.options.mode.copy_on_write` to `True` to avoid copy/view problems.

B.1 Function get_data

1. Define a function called `get_data` that takes 3 parameters: `table`, `exam_type`, and `con`. The `table` parameter will be the name of the table to read, `exam_type` will be the type of exam to extract (English or math), and `con` will be a database connection.
    1. Begin the function with an `if` statement that checks whether `exam_type` is `"english"`.
        1. Within the block set `exam_filter` to a string containing three quoted exam names: `'ELA4'`, `'REG_COMENG'`, and `'Regents Common Core English Language Art'`. The second two are for the HS Regents exam: its code changed during the period. It will be easiest to read if you use a triple-quoted string and put the exam names on separate lines. Remember that each needs to be quoted.

2. Go back to the level of the `if` statement and add an `elif` block that tests whether `exam_type` is `"math"`. Within the block do the following:
    1. Set `exam_filter` to a string containing the following three quoted exam names: `'MATH4'`, `'REG_COMGEOM'`, and `'Regents Common Core Geometry'`. As with English, the HS Regents code for geometry changed during the period.
3. Go back to the level of the `if` statement and add an `else` block. Within the block do the following:
    1. Add an `assert False` statement. This will cause the script to stop with an error if `exam_type` is not one of the expected strings.
4. After the `if` block, set `sql` to an f-string giving a SQL command to select all columns from `{table}` where `subgroup` is in `"All Students"`, `"Economically Disadvantaged"`, `"Students with Disabilities"`, or `"Black or African American"` and `exam` is in `{exam_filter}`.
5. Print `sql`. Look it over to make sure it's correct.
6. Set `data` to the result of calling `pd.read_sql()` with parameters `sql` and `con`. This will extract the desired rows from the table and return the results as a DataFrame.
7. Use the `.astype(float)` method to convert `data['per_prof']` from a string to a 'numeric variable.
8. Now create a dictionary called `eras` that contains the following key:value pairs: `2018:'pre'`, `2019:'pre'`, `2022:'post'`, and `2023:'post'`. This defines which years are pre- and post-pandemic.
9. Set `data['era']` equal to the result of calling the `.replace()` method on `data['year']` with argument `eras`.
10. Use the `.to_csv()` method of `data` to write it out to a file with a name given by `f"{exam_type}.csv"` and using `index=False` to suppress writing the index since it's not useful.
11. Return `data`.

B.2 Function compare

1. Define a function called `compare` that will plot comparison graphs. It should take 3 arguments: `data`, a data frame, `level`, which will be either `'EM'` or `'HS'`, and `title`, a string giving the graph title.
2. Within the function, set `grouped` to be the result of calling the `.groupby()` method on `data` using four columns to define the groups, `name`, `subgroup`, `level`, and `era`.
3. Set `means` equal to the value of applying the `.mean()` method to column `"per_prof"` of `grouped`. This calculates the mean score in each era for each district.
4. Set `stack` equal to the result of calling `.reset_index()` on `means`.
5. Set `stack` equal to the result of filtering the rows of slack to include only those where `stack['level']` is equal to `level`.
6. Set variable `order` equal to the result of calling the `sorted()` function on the unique elements of `stack['subgroup']`. This will be used to insure that the groups in graphs always appear in the same order.

7. Set `fig,ax` to the result of calling `plt.subplots()`.
8. Use the `.suptitle()` method of `fig` to set the title to an f-string containing the title and level as follows: `f"{title}: {level}"`
9. Draw a set of paired horizontal boxen plots for each group in each era by calling `sns.boxenplot()` with the following arguments: `data=stack`, `x="per_prof"`, `y="subgroup"`, `hue="era"`, `orient="h"`, `order=order`, and `ax=ax`.
10. Call `ax.legend()` with the following two arguments to place the legend outside the main plotting area: `loc="upper left"` and `bbox_to_anchor=(1,1)`. Together, the arguments say to put the upper left corner of the legend at the upper right corner of the main plotting area.
11. Call `ax.set_xlabel()` with an empty string (`''`) as an argument to turn off the X axis label.
12. Call `ax.set_ylabel()` with an empty string as an argument to turn off the Y axis label.
13. Call `ax.set_xlim()` with two parameters: `left=-5` and `right=105`. This ensures that all of the graphs will have the same scale for easy comparison. The -5 and 105 create small margins at the left and right since the actual data runs from 0 to 100.
14. Call `fig.tight_layout()`.
15. Set `filename` to following f-string `f"fig-{title}-{level}.png"`.
16. Clean up the name by setting `filename` to the result of calling the `.lower()` and `.replace(' ','_')` methods on it. The result should be a lower case filename with underscores and no spaces.
17. Call `fig.savefig()` with `filename` as the argument.

B.3 Main analysis

1. After the end of the `compare` function, add a line setting `con` to the value of calling `sqlite3.connect()` with `"filtered.db"` as the argument.
2. Get the English data by setting `eng` equal to the result of calling `get_data()` with three arguments: `"exams"`, `"english"`, and `con`.
3. Get the math data by setting `math` equal to the result of calling `get_data()` with three arguments: `"exams"`, `"math"`, and `con`.
4. Analyze it by adding a `for` loop using `level` as the running variable and looping over a list consisting of `"EM"` and `"HS"`.
    1. Call `compare()` with arguments `eng`, `level`, and `'English proficiency'`.
    2. Call `compare()` again but with arguments `math`, `level`, and `'Math proficiency'`.

If all has gone well, you should see that English and math proficiency was roughly steady at the EM level (little change in the median scores or overall position of the boxen plot), that HS English proficiency declined slightly, and that HS math proficiency declined dramatically, particularly for the three subgroups.

Submitting

Once you're happy with everything and have committed all of the changes to your local repository, please push the changes to GitHub. At that point, you're done: you have submitted your answer.

Notes

- When no columns are specified in the `INSERT` statement, it loads variables into the table by their **order** in the `CREATE` statement and doesn't use column names. That allows output names to be different from the input names: e.g., `code` instead of `ENTITY_CD`.