NDAB21003E Information and Exam

Final Version 7 Jan 21, 2023

Information

How do I hand-in?

The final version of your exam must be submitted to the Digital Exam system (https://eksamen.ku.dk). The submission to the digital exam should consist of two files: `exam.py` and `exam_test.py`. While the exam is ongoing, you can hand in as many times as you like, so feel free to submit preliminary versions during the exam.

Is the auto-correction server available during the exam?

Different from the hand-ins throughout the course, you will (of course) **not** be able to get detailed feedback on your solution by submitting it to CodeGrade. However, we have set up the system to allow you to test whether the basic structure of your files is correct. It works exactly like the weekly hand-ins: in Absalon, you will find an assignment called "Exam" - under this assignment, you can submit your code to CodeGrade and get feedback as usual.

Please note that even if all tests pass, it says nothing about the correctness of your code, so you cannot use it to validate your solution, but if a test fails, it means that this particular exercise does not run and will, therefore not be assessed for the exam. This service is optional, but we highly recommend it to rule out any silly mistakes, such as spelling errors in function names.

Please note that submitting to CodeGrade is not considered "handing in" - you must submit to the digital exam for your exam to be registered as submitted.

Format:

You should create the following two python files for the exam:

1. A Python file called `exam.py`, containing the functions and packages.
2. A Python file called `exam_test.py`, containing test code for the individual questions and packages.

The details about which code should go where will be provided in the questions below.

Content:

Remember to use meaningful variable names for the Python part, include docstrings for each function/method/class, and add comments when the code is not self-explanatory. Many of the questions will instruct you to solve the exercise using specific tools (e.g., only for-loops) - please pay attention to these instructions. Also, **please use external modules only when explicitly mentioned in the exercise**. Please do all of these to ensure we deduct points even for code that works.

Can we look up things on the internet?

Yes. You can search for online documentation and use websites like StackOverflow during the exam, but please try to avoid copying large blocks of code verbatim from online examples. If you, for some reason, find it necessary to copy code directly (without rewriting it), then please add a comment that specifies the source of this code (so that we can rule out plagiarism if two of you copied the same block). You are **not** allowed to communicate with anyone during the exam, also not use online chat services, mail, etc.

How should I structure my time?

Note that Q1, Q2, Q3, and Q4, are independent of one another - in the sense that you can solve any of them without solving the others. Some exercises will also be easier than others. If you are stuck on something, we strongly encourage you to move on to one of the exercises following it to ensure you get all the exercises you could have solved.

What do I do if I find an error in the exam?

If you find an error or ambiguity in the exam, please email ds@di.ku.dk. If we agree that something should be clarified, we will post an Absalon announcement with a clarification.

EXAM

1 Question Random and Modules (20 points: 6+7+7)

In this task, we will create a python program that simulates tossing twenty-sided dice to determine a winner between 2 players. You can imagine this as a part of a digital game where a player can play a bonus card to increase their chance of winning.

a. Create a function called `dice_roller(sides, bonus)` that rolls the dice that takes 2 parameters, the number of sides, and if the team has a bonus. Be sure to place this and the following functions in `exam.py`

b. Create a second function called `game_play()` for the gameplay that calls the `dice_roller` function for players with the following:

- `monster` rolls a 20-sided dice and has a bonus of +10
- `team` has 3 rolls
  - First roll with an 8-sided dice and no bonus card
  - Second roll with a 12-sided dice and a bonus card of +5
  - Third roll with a dice of 6 and no bonus card
- Inside `gameplay()` create a conditional statement (if ... then) to return the winners with the score and the losers with the score. In `game_play()`, you need to return the following for one part of the conditional statement, and for the second part of the statement, the team wins.
  - `return ('monster wins', monster, 'team loses', team)`
  - The output in the terminal should look like this `('monster wins', 25, 'team loses', 11)`

c. From `exam_test.py` call the `game_play()` and print the result.

2 Question List of List / For Loops (25 points: 5+10+10)

In this task, we need to transpose (switch the rows and the columns if thinking about a table) the `dia_list`. The transposition of a list of lists (matrix) is a new list of lists (matrix) where the rows of the original become the columns, and the columns become the rows.

Once you have transposed the matrix, we create a new matrix with the `PatientNumber`, `BMI`, and `Outcome`. However, we also need to change the `1` and `0` integers to strings `positive` and `negative`.

a. Start with creating and copying the `dia_list` in the `exam.py` (see below)

b. In `exam_test.py` create a function called `transpose_list()` that uses a for loop (comprehension not allowed) to transpose items and the subitems from a "9x19" matrix to a "19x9" matrix.

- Name the new list `transposed_new`
- Remember to make a nested loop for the items and subitems. Use the following names:
  - Create a `for loop` inside another `for loop`
  - For the first loop, the `item` is the variable.
  - For the second loop, use `for subitem in...`
- From `exam_test.py,` print out the third, fourth, and fifth items of the `transposed_new` list.

c. Create a second function, `get_patients()`, that takes the `transposed_new` list and extracts `PatientNumber`, `BMI`, and `Outcome`.

- Name this new list `age_bmi`
- Create the loop with `for rows in...` for the extraction
- Use a conditional statement that tests if `Outcome` is an integer of `1` and `0` and converts to `positive` and `negative` with `1` being `positive`.

d. From `exam_test.py` call `get_patients()` and print the result.

```
dia_list=
[['PatientNumber',14568,22368,30168,37968,45768,53568,61368,69168,76968,84768,92568,503
68,18168,15968,33768,41315,39368,14168,54968],
['Glucose',148,85,183,89,137,116,78,115,197,125,110,168,139,189,166,100,118,107,103],
['BloodPressure',72,66,64,66,40,74,50,0,70,96,92,74,80,60,72,0,84,74,30],
['SkinThickness',35,29,0,23,35,0,32,0,45,0,0,0,0,23,19,0,47,0,38],
['Insulin',0,0,0,94,168,0,88,0,543,0,0,0,0,846,175,0,230,0,83],
['BMI',33.6,26.6,23.3,28.1,43.1,25.6,31,35.3,30.5,0,37.6,38,27.1,30.1,25.8,30,45.8,29.6
,43.3],
['DiabetesPedigreeFunction',0.627,0.351,0.672,0.167,2.288,0.201,0.248,0.134,0.158,0.232
,0.191,0.537,1.441,0.398,0.587,0.484,0.551,0.254,0.183],
['Age',50,31,32,21,33,30,26,29,53,54,30,34,57,59,51,32,31,31,33],
['Outcome',1,0,1,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0]]
```

3 Question Numpy + Plots (25 points: 5+5+5+5+5)

In this task, we will work with a dataset of child obesity that contains `age`, `exercise`, `height`, and `obesity`. We will use NumPy and Matplotlib to investigate the relationships between them.

a. Use `import numpy as np` and `import matplotlib.pyplot as plt` in the `exam_test.py`.

- In `exam_test.py` load the health dataset into a NumPy array using the `genfromtxt` function with the `skip_header` parameter enabled.
- Print the `shape` of the dataset (number of rows and columns) from `exam_test.py`

b. Extract columns into variables for `age`, `weight`, `height`, `exercise`, and `obesity` columns.

c. Create a scatter plot from `exam_test.py` of weight vs. age, with exercise level represented by color.

- X-axis label "Age (years)"
- Y-axis label "Weight (lbs)"
- Title "Weight vs. Age with Exercise Level"
- Colorbar "Exercise Level (hours per week)"

d. Create a bar chart of children aged 10 for obesity vs. normal from `exam_test.py`

- Use the variable `age_counts` to perform the np operations to get the age group and to plot the obesity. Obesity is 1, and Not Obese is 0.
- The bar legend should read "Not Obese", "Obese"
- X-axis "Obesity"
- Y-axis "Age 10 count"

e. Using NumPy `print` in the terminal the Correlation between weight and exercise from `exam_test.py`

4 Data Story with Pandas: 30 pts 10+5+5+5+5

You will mainly use Pandas to work with the COVID-19 confirmed cases dataset in this problem. The dataset contains the country, province, coordinate and **cumulative** count of confirmed cases each day.

The dataset contains the following columns:

- Province/State: The province or state within the country in the next column (can be empty).
- Country/Region: The name of the country/region. Each unique entry is seen as a country.
- Lat: The Latitude of the location (province/state or country) in the GPS coordinates
- Long: The longitude of the location (province/state or country) in the GPS coordinates
- Date: The values are the cumulative number of confirmed COVID-19 cases of different countries on this day.

a. In `exam.py`. Write a function `data_cleaning` that i) removes all locations with missing latitude or longitude values, or both latitude and longitude values are zeros. ii) most column names are date times in string format. Convert these columns to datetime format. The function should take a DataFrame variable `df` as input and returns the cleaned dataframe. The data cleaning needs to be done in Pandas, not with for loops.
Hint: `pd.to_datetime()`

In `exam_test.py`, read the raw data into `covid_df`, and call the `data_cleaning` function to clean the data. Store the output at the `cleaned_covid_df` variable.

b. Within `exam.py`, write a function `get_countries_by_earth_quadrant`, which takes a DataFrame variable df, similar to `cleaned_covid_df`, as input. The function should return a dictionary with each of the Earth Quadrants (NW, NE, SW, SE) as keys and a list of the countries belonging to each quadrant as values. Where 'NW', 'NE', 'SW', and 'SE' are specified as:

- Longitude >= 0 -> E; Longitude < 0 -> W;
- Latitude >= 0 -> N; Latitude < 0 -> S;

The countries in each quadrant should be ordered alphabetically.

In `exam_test.py`, call the `get_countries_by_earth_quadrant` function to get the dictionary and save it to `country_dict`.

c. In `exam.py`, write a function `group_table_by_country` to derive country-wise data from the original table using `df.groupby` function. If a country has more than one location (i.e. provinces) included, its location should be the center of all locations, and its confirmed cases count should be the summation of the per-location values. The function should take a DataFrame variable `df` as input and returns the per-country counts in another dataframe. The index of the output dataframe should be reset to 0,1,...

In `exam_test.py`, call the `group_table_by_country` function and save the result `country_wise_df`.

d. Plot the total number of cumulative confirmed cases in a whole earth quadrant on all days in the table. In `exam.py`, write a function `plot_new_cases_daily` that draws a line plot of the total number of cumulative confirmed cases **in the input earth quadrant** on each day, where the x-axis is the date and y-axis is the corresponding number of cumulative confirmed cases in the whole quadrant. Choose a proper title and label for both axes.

The function should take a DataFrame variable `df` and a string variable `earth_quadrant` in {'NW, NE, SW, SE'} as input and return the axes object (plotting area) of the figure.

In `exam_test.py`, call the function by `plot_new_cases_daily(country_wise_df,'NE')` and save the result as `ax_ne`.

Hint: You may consider obtaining a Series variable to store the total number of new cases in the earth quadrant; then create a figure with `fig, ax = plt.subplots(1,1)`; and call `series.plot()` to create the line plot and add `ax` to the arguments. The `ax` object is returned after plotting.

e. Write a function `get_top_new_case_countries_by_month` to compute the top-K countries in terms of new confirmed cases in any specified year and month. If the searched year and month are illegal or not present in the table, print a message 'the input month is not available in the data!' and return None. Otherwise, output the top K countries with the highest number of new cases during the month. **We may compute the difference between the last day and the first day of the month as the number of new confirmed cases during the month.**

The function should take a DataFrame variable `df`, an integer `year`, an integer `month,` and an integer `k_val`. For legal input, a dictionary object should be returned, where the keys are top-K country names and values are their new confirmed cases, ordered by the values in descending order.

In `exam_test.py`, call the function by `get_top_new_case_countries_by_month (country_wise_df, year=2021, month=2, k_val=10)` and save the result to `top_k_countries_dict`.

Hint: you can visit the year, month, and day of a datetime object dt by dt.year, dt.month, dt.day, respectively.