Python for physicists – Final Project

The **objective** of the final project is to demonstrate your Python skills in utilizing Python packages for data manipulation, analysis, and visualization, particularly focusing on object-oriented programming with classes and subclasses.

1- Dataset Selection: Options and Requirements

Obtain a dataset (you have the freedom to choose where to get the data)

- Options include:
  - Seaborn database, such as planets, brain networks, titanic, etc. (see the last task of tutorial 8 or the last slides of lecture number 10).
  - Data.gov data sets (https://catalog.data.gov/dataset/) (see the first task of tutorial 8).
  - Any data files shared or used during the lectures or tutorials such as example_file_pandas.csv.
  - From your **own project**, but ensure that the dataset is in a suitable format (e.g. CSV).
- The data must be of type 1) time series or 2) dictionaries (such as Titanic from Seaborn database where rows do not involve changes over time). If you select time series, select 2 time series. Make sure they overlap in time and that all measurements are NOT all simultaneous (i.e. there are different measurement times or missing values.)

2- Object-Oriented Data Analysis: Time Series Manipulation or Dictionary Manipulation

Define a class with the main object, for example, **data_manipulation**

a. Define functions within this class (ensure that functions are well-defined, organized, and commented) for;
   i. Reading the data as pandas **DataFrame**
   ii. Cleaning the data (handling missing values and outliers by detecting and dropping rows or lines, or filling them with interpolation, as well as adjusting data types from string to float, etc.)
   iii. Displaying the data using **Matplotlib** libraries. Try to include various types of representation such as scatter plots, time series, and histograms.
   iv. Apply mathematical operations or perform statistical analyses using **NumPy, SciPy,** or **Pandas functions**.

      <u>If the data type is a time series:</u>

      1. Detrend the data by subtracting the mean and then rescale it to fit within the interval [-1, 1].
         a. Plot the detrended time series to visually inspect the removal of trends.
         b. Calculate and plot the autocorrelation function (ACF) or partial autocorrelation function (PACF) to assess the stationarity of the detrended series.
         c. Perform a time series decomposition (e.g., using seasonal decomposition) to separate the trend, seasonal, and residual components and analyze each component separately.
      2. Perform interpolation on the 2-time series using **SciPy** to fill in the missing values and obtain simultaneous measurements.

    a. Visualize the original and interpolated time series to compare the effects of interpolation.
    b. Assess the impact of different interpolation methods (e.g., linear interpolation, cubic spline interpolation) on the data.
    c. Check the quality of the interpolation by comparing it with neighboring data points and assessing the continuity of the interpolated values.
3. Utilize the **cumsum function** for cumulative summation.
    a. Plot the cumulative sum to visualize the cumulative changes over time.
    b. Analyze the rate of change of the cumulative sum to identify periods of acceleration or deceleration.
    c. Compare the cumulative sum of the 2-time series.
4. Save the transformed data as new columns in the **DataFrame**.

Write the above steps into object-oriented coding:

Create a base class called “time_series_analyzer” which contains common methods and attributes for analyzing time series data. This base class includes methods for loading data, preprocessing, and common analysis tasks. Additionally, emphasize the importance of error handling using try-except blocks, documentation through meaningful docstrings, and logging to record program execution details.

Then implement a subclass of “time_series_analyzer” called “detrending_analyzer”. This subclass will use the detrending method inherited from the base class to perform detrending by subtracting the mean and rescaling. Additionally, it implements methods for plotting the detrended time series, calculating and plotting autocorrelation functions, and performing time series decomposition.

For the inheritance from a parent class: “interpolation” and “cumulative_summation” should be implemented as separate subclasses of “time_series_analyzer”.

Each subclass inherits common methods and attributes from the base class.

<u>If the data type is a dictionary:</u>

1. Sort these new columns in descending or ascending order (or any other columns).
    a. Analyze the sorted data to identify trends or patterns based on the sorted order.
    b. Compare the sorted data with the original data to assess the impact of sorting on the distribution or characteristics of the data.
    c. Plot histograms or density plots of the sorted data to visualize the distribution.
2. Apply Boolean conditions using if/else syntax to extract a data subset: 25% of all odd times.
    a. Visualize the extracted data using scatter plots, line plots, or bar charts to identify any patterns or outliers.
    b. Compare the extracted data with the original dataset to assess the impact of the selection on the subset of data.

    c. Compute the correlation coefficient between the original and the subset.
3. Calculate the average, mean, median, and standard deviation of all data sets in columns.
    a. Visualize the distribution of the calculated statistics using histograms, box plots, or violin plots.
    b. Compare the statistics across different subsets of the data to identify differences or similarities.
    c. Assess the variability of the statistics across different groups within the dataset.

Write the above three options into object-oriented coding:

Create a base class called “dictionary_analyzer” which contains common methods and attributes for analyzing dictionary-type data. This base class includes methods for loading data, preprocessing, and common analysis tasks applicable to dictionary-type data. Additionally, emphasize the importance of error handling using try-except blocks, documentation through meaningful docstrings, and logging to record program execution details.

Implement a subclass of “dictionary_analyzer” called “sort_analyzer”. This subclass will override the sorting method inherited from the base class to sort the data in ascending or descending order. Additionally, it can implement methods for analyzing the sorted data, comparing it with the original data, and visualizing the distribution of sorted data.

Implement another subclass of “dictionary_analyzer” called “Boolean_condition_analyzer”. This subclass will implement methods to apply boolean conditions to extract specific data satisfying certain criteria. Additionally, it should implement methods for visualizing the extracted data, analyzing its frequency or proportion, and comparing it with the original dataset.

Implement a third subclass of “dictionary_analyzer” called “statistics_analyzer”. This subclass will implement methods to calculate statistics such as average, mean, median, and standard deviation for the columns of the dictionary data. Additionally, it should implement methods for visualizing the distribution of calculated statistics, comparing statistics across different subsets of the data, and assessing their variability over time or across different groups within the dataset.

Each subclass inherits common methods and attributes from the base class.

v. Display the original and transformed data you obtained.
vi. Add necessary information to the plot: grid, labels, title, axis names, xlim, etc.
vii. Save the plot as **figure1.pdf**, **figure2.pdf**, **figure3.pdf**, etc. (depending on what you do in your project, it can be 1D, 2D, 3D, or histogram …).

3- **Code Integration, Module Import, and Performance Evaluation:**

   a. Combine all code outlined in Step 2 into a module named **main.py.**

   b. Import the **main.py** module into another script called **presentation.py**, and use it to utilize all your classes and functions in order to demonstrate the properties of your code such as printing results, presenting plots, etc.

   c. Evaluate the performance of the code by measuring the execution time using the **time.time()** function in the **main.py** and printing the result in **presentation.py**. If you think you cannot improve it anymore, explain why.

Alternatively, feel free to exercise creativity by demonstrating how the project output can be both useful and impressive. **You will be graded for your creativity.**

If you encounter any difficulty completing the final project, please review the lectures, tutorials, and homework exercises, as you have sufficient time. If nothing works, feel free to send me an email for help.

Submission instructions - please read carefully:

• To be submitted by **5th of May 2024** in the moodle (Lemida) system.
• *** files with **py, txt, pdf** extensions must be submitted - named exactly as detailed below in the project. **In addition, submit all files in one .pdf file as FP_+studentID+.pdf**
That is to say that:
• Do not submit libraries, zip files, etc., but separate files with the names listed above and the whole project as one .pdf file.
• Make sure that the files run on a recent version of Python, 3.5 or higher. At the beginning of the script called **presentation.py**, print the version of each package that you used.
• Use the commands that are in the context of what we have learned in this course. If you want to use external packages, motivate your choice by explaining why you could not do it with what you learned during the lectures.
• Note that the project will be written in object-oriented coding with classes and subclasses by utilizing the inheritance properties.
• Implement **try/except** syntax and **raising errors** in your code, preferably during file reading. But if you use it elsewhere in the code, ensure clarity and explain why you want to use this syntax in another part of the code. At the end create a **log.txt** file using the logging function.
• For each step in the codes both in module **main.py** and script **presentation.py**, provide clear explanations by using hashtags (#), and single or double quotation marks (‘’) to enhance understanding and clarity. If necessary provide a README.txt file where you explain what the code is doing and what each file contains such as for an overview of the code structure, explanations of each file's purpose, instructions for running the code, and any other relevant information.

Lecturer: Dr. Husne Dereli-Bégué
Email: husnedereli@gmail.com