Class Project – Weather Processing App

This project will be marked out of 100 points, and is worth 50% of your final grade.

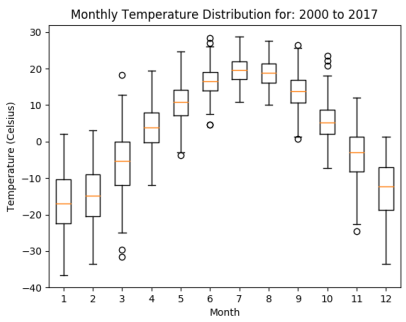
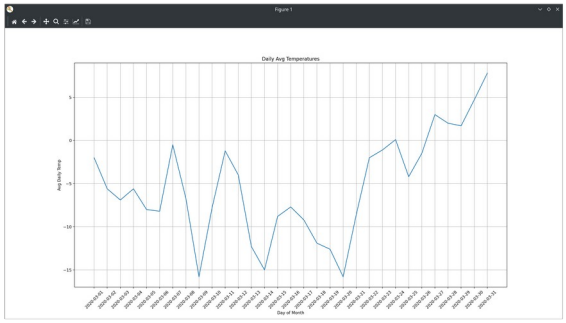
Develop an application with the following features:

Part 1 – Scraping	
Tasks	• Create a **scrape_weather.py** module with a **WeatherScraper** class inside. • Use the Python HTMLParser class to scrape Winnipeg weather data (min, max & mean temperatures) from the Environment Canada website, from the current date, as far back in time as is available. ◦ http://climate.weather.gc.ca/climate_data/daily_data_e.html?StationID=27174&timeframe=2&StartYear=1840&EndYear=2018&Day=1&Year=2018&Month=5# ◦ Notice the year and month is encoded directly in the URL. • Your code must automatically detect when no more weather data is available for scraping. In other words, you are not allowed to hard code the last available date into your program. You are also not allowed to fetch the last date from any dropdown menus on the site. ◦ You can try using a web browser to go back to the earliest available weather url. Then modify the date in the url to go back earlier, and see what happens. Use that knowledge to write your code in a way that detects when it can't go back any further in time. • All scraping code should be self-contained inside the WeatherScraper class. There should be no scraping code anywhere else in the program.
Input	The starting URL to scrape, encoded with today's date.
Output	A dictionary of dictionaries. For example: • daily_temps = {"Max": 12.0, "Min": 5.6, "Mean": 7.1} • weather = {"2018-06-01": daily_temps, "2018-06-02": daily_temps}
Grading	30 points

Part 2 - Database	
Tasks	• Create a **db_operations.py** module with a **DBOperations** class inside. • Use the Python sqlite3 module to store the weather data in an SQLite database in the specified format. SQL queries to create and query the DB can be provided if required. The DB format for your reference: ◦ id -> integer, primary key, autoincrement ◦ sample_date -> text, unique in combination with location ◦ location -> text, unique in combination with sample_date ◦ min_temp -> real ◦ max_temp -> real ◦ avg_temp -> real • Create a method called **fetch_data** that will return the requested data for plotting. The data it returns gets used in the plotting class. • Create a method called **save_data** that will save new data to the DB, if it doesn't already exist (i.e. don't duplicate data). There are a number of ways to prevent data duplication. Explore your options! • Create a method called **initialize_db** to initialize the DB if it doesn't already exist. This should be called every time the program runs. • Create a method called **purge_data** to purge all the data from the DB for when the program fetches all new weather data. This should not delete the database, it should just delete the data. • Create a context manager module called **dbcm.py** with a **DBCM** class inside to manage the database connections. The DBCM class should return a cursor, NOT a connection. Changes should be committed and all connections closed in the __exit__ method of DBCM. • All database operations should be self contained in the DBOperations class. There should be no database code anywhere else in the program.
Input	Dictionary from WeatherScraper class.
Output	A rows tuple containing DB records.
Grading	20 points

Part 3 - Plotting	

Tasks	• Create a **plot_operations.py** module with a **PlotOperations** class inside.
	• Use Python matplotlib to create a basic boxplot of mean temperatures in a date range (year to year, ex. 2000 to 2020) supplied by the user:
	  ○ https://matplotlib.org/examples/pylab_examples/boxplot_demo.html
	• In addition to the above box plot, display a line plot of a particular months mean temperature data, based on user input. For example, display all the daily mean temperatures from January 2020, with the x axis being the day, and the y axis being temperature.
	  ○ https://matplotlib.org/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py
	• All plotting code should be self contained in the PlotOperations class. There should be no plotting code anywhere else in the program.

Input	Be creative. One way is a dictionary of lists. For example: • weather_data = {1: [1.1, 5.5, 6.2, 7.1], 2: [8.1, 5.4, 9.6, 4.7]} • The dictionary key is the month: January = 1, February = 2 etc... • The data is all the mean temperatures for each day of that month, for every year desired (box plot), or just for a specific year (line plot). • You'll need to do some data shuffling and organizing for this step to put the data in a format ready for plotting.
Output	A boxplot displaying one box per month, so it shows all 12 months of the year on one plot. Labels are automatically created from user input. In addition, a line plot which shows the mean daily temp of a particular month and year. Example:  
Grading	15 points

Part 4 – User Interaction	
Tasks	• Create a **weather_processor.py** module with a **WeatherProcessor** class inside. • When the program starts, present the user with a menu of choices. • Allow the user to download a full set of weather data, or to update it. ◦ When updating, the program should check today's date and the latest date of weather available in the DB, and download what's missing between those two points, without duplicating any data. • Allow the user to enter a year range of interest (from year, to year) to generate the box plot. • Allow the user to enter a month and a year to generate the line plot. • Use this class to launch and manage all the other tasks. • All user interaction should be self contained in the WeatherProcessor class. There should be no user prompt type code anywhere else in the program.
Input	User supplies input.
Output	Call the correct class methods to accomplish the tasks.
Grading	20 points

Part 5 - Packaging	
Tasks	• Create a Windows package installer using Inno Setup, that allows a user to install your weather app on a Windows 10 computer. • Include your own icon logo and license agreement as part of the installation process. Any license agreement is fine. Try the GPL or Berkeley license.
Input	Binary distribution created with the Python pyinstaller module.
Output	Standalone exe installer package for Windows 10, clearly labeled and located so it's easy to find. Don't submit your entire exe build folder please.
Grading	10 points

Part 6 – Additional Requirements	
Tasks	• Code must adhere to the PEP8 standard, and will be checked with pylint. ◦ To install pylint: pip install pylint ◦ To use pylint: pylint myfile.py or python3 -m pylint myfile.py ◦ You must achieve a score of 8 or higher • Code must be documented well for easy review and grading. At minimum: ◦ Module level docstring ◦ Class level docstring ◦ Function/method level docstring • Every function/method needs to implement error handling. • Errors should be logged to a log file using the python logging module.
Grading	5 points

Part 7 – Bonus – Optional	
Tasks	• Create a nice user interface for all user interaction. • Label and align everything properly so it looks nice. • The widgets should scale properly when the window is resized. • You should be proud to show this to an employer. In other words, it should be functional, complete and aesthetically pleasing, not half-baked. • The matplotlib charts can open in their own window, they don't need to be integrated into the UI. • Implement threading in your application to speed up scraping.
Bonus	15 points (10 points for UI, 5 points for threading)

Total		
Points	Bonus	Maximum Possible Points
100	15	115/100