Task Summary: Write a script/program that periodically calls rippled's `server_info` command and records the sequence number of the latest *validated ledger* along with the current time. Record this data in a file. Then, use this data to construct a plot (time on the x-axis, sequence number on the y-axis) that visualizes how frequently the ledger sequence is incremented over time (i.e. how often new ledgers are validated). Choose a time span and polling interval that can effectively capture and depict this information.

- **How does your script work?**
  - I have written the script in Python, the script uses "requests" libraries to call API, the POST function along with the API method name is called on the provided URL to get the response. The response is then stored in String and parsed to get Time and Seq value as was asked in the task summary. The script recalls the URL to get the new response and when there is a new Seq ID in response, data is stored in a file. The loop is written to run in such a way to get 500 records.

- **How did you decide on your polling interval?**
  - While understanding the API, I ran a test run with a polling interval of 100 milli second and for the data of 1000 distinct ledger results I figure out that the average time a ledger is taking to update is around 2 with minimum value coming around 1.6 seconds. With these findings I decided the polling interval to be 800 milli seconds.

- **What do the results tell you?**
  - The result can be categorized in following cases
    - Below 3.0 seconds
      - Few cases falling under this category
    - 3.0 to 4.5 seconds
      - Majority of the cases are in this range with slight variation
    - Above 4.5 seconds
      - Some cases falling under this range, which shows that system took extra time in closing the ledger

- **What might explain the variation in time between new ledgers?**
  - As we can see from documentation there are some scenario in the system which can cause ledger to acquire more time than usual, like the scenario of "Double Spend Problem", where the system waits for consensus to complete before closing the ledger. The spikes seems in the result might be the cause of these cases.

Submitted by: Syed Talha Ahmed

Bonus question #1: Enhance your script to calculate the min, max, and average time that it took for a new ledger to be validated during the span of time captured.

- A Separate result set is attached where individual time of each ledger and minimum, maximum and average time is also stored.

Bonus question #2: There are some other (better) ways that you could use the rippled API to find how long each ledger took to close/validate. Using the API documentation, find and describe one of these methods (you don't need to actually implement it).

- Following API calls can be considered as well...
    - ledger_current: The ledger value can be used along with current system time to manage the task. The benefit is the response is only a "ledger" id value not much parsing required
    - ledger: This api responds with closing time along with ledger id. Which means no separate logic is required to calculate closure time of each ledger, as the "server_status" response, the time is of system time.

Assignment experience:

I identified that I need to break the problem into following parts...

1. Understanding the protocol and API specs
2. Establishing the connection with the Server
3. Finalizing the polling interval and time span for the overall execution
4. Parsing the response
5. Writing the file with correct formatting
6. Identifying the Minimum, Maximum and Average value
7. Understanding the GNUPlot
8. Feeding the file to GNUPlot

I will briefly explain what challenges I faced in each problem and how I moved forward.

1. **Understanding the protocol and API specs:**
   I studied the specs present at https://xrpl.org/server_state.html. They were precisely explaining what they do and the little sample provided clears out all the confusions. Lastly I tried with the Web socket tool present under "Developer Tools" to call the API and see all the fields received in response

2. **Establishing the connection with the Server**
   I logged in an online tool present to check the API status. I selected POST method and was able to receive a response. This gave me confidence that there is no issue with connection and now I can start the development.
   I used "requests" package of python and called "request.post" to make the connection and call the API, after few attempts I was successful in establishing the connection and getting a successful response.

3. **Finalizing the polling interval and time span for the overall execution**
   While understanding the API and making the test calls, I realized the ledgers are getting updated in around 2 second on average going up to 4 seconds at max. With this I thought that data with 500 ledger calls should suffice. For this I put the polling interval at 800 milli second, and made the loop to complete when I hit the 500th record.

4. **Parsing the response**
   With this understanding and logic I started the coding. Since I am not very much familiar with Python I didn't used any library for parsing and did the parsing for "Time" and "Seq" by making substring of overall response.
   It is to be noted that the length of ledger id is not mentioned anywhere, so I coded by searching the closing and next tab to find the ending index of the value.

Submitted by: Syed Talha Ahmed

5. **Writing the file with correct formatting**
   Now I parsed the time and sequence for each new record. With this I formatted the string line to store in the File. And started writing the file, one record at a time.
   The logic of writing each record is also used so as to make the real time graph using same file if there is any requirement.

6. **Identifying the Minimum, Maximum and Average value**
   While formatting the output file, I kept 3 variables to calculate min, max and avg time taken and kept comparing these variables for each record and updated them if condition is satisfied. A separate Data file is maintained for time taken of each Ledger Seq.

7. **Understanding the GNUPlot**
   This is the first time I have used GNUPlot, I watched a tutorial video on YouTube and it explained the working very precisely. I tried with dummy files and I got the understanding of using it.

8. **Feeding the file to GNUPlot**
   The formatted file generated by the script is fed in the GNUPlot using "plot" command, I faced many problems in achieving the result, but after providing the range, updating the formatting for reading the data and trying multiple times, I finally achieve the result.

Assignment explanation:

The folder contains following files and following are the details...

1. **code.py**: Python script for communication with API
2. **Task_seq_duration.txt**: Data file for records of ledger with Seq ID and current Time
3. **Task_seq_n_time.txt**: Data file for records of ledger with Seq ID and its duration
4. **Plot.dat**: Script file for GNUPlot
5. **Graph.png**: Graph of Seq against system time

Submitted by: Syed Talha Ahmed