**Basic design:**

- We need to calculate mean and median at the same time in parallel.
- So a TimerTask is created and scheduled to run every hour as the data is updated every hour in BigQuery.
- In order to execute mean and median query concurrently, an ExecutorService with fixed thread pool size of 2 is created within the TimerTask.
- Within each callable, the respective query is configured, built and executed.
- The callables are executed concurrently by using "invokeAll()" of ExecutorService.

**Logic of adjusted mean:**

- Initially global mean is zero.
- With each execution of the mean query, the PushEvents are filtered and the mean length of all the commit messages for each record is calculated. From this, the aggregated mean is returned for that hour.
- Global mean is adjusted with this aggregated mean at each run.

**Logic of adjusted median:**

- With each execution of the median query, the PushEvents are filtered and the median of all the commit messages for each record is calculated. From this, the aggregated median is computed for that hour with 0.1% error using NTH and QUANTILES function of BigQuery. The explanation and working of it can be found at this link https://cloud.google.com/bigquery/docs/reference/legacy-sql#quantiles.
- Global Median is computed as follows:
  A list of medians is maintained which stores the medians computed at each hour.

The global median is adjusted at each run by sorting this list of medians and taking the mid element of the list, if list size is odd or taking the average of mid element and its previous element, if list size is even. This method takes O(n logn) time to sort the list and compute the adjusted median from it. The median of medians algorithm can be used to improve the time complexity to O(n) time (using quick select logic) to get the adjusted median.

**Query details and scheduling:**

- The table to be queried at each run is resolved while configuring the query before execution.
- The current UTC timestamp is taken and the table name is formatted from this. The edge case of computing the mean and median at time 00:00:00 of a day is handled separately. In that case the date is subtracted by 1 day to get the required table name. In 00:00:00 execution of a day, the 23:00:00 to 23:59:59 data of the previous day is required to compute the mean and median. Hence this logic is required.
- This job is currently scheduled to run every hour, but it can be configured to run every 6 hours or every 24 hours to get a daily mean and median.
- Since the mean and median is scheduled to be adjusted every hour, it is enough to look at the day dataset instead of month and year. Even if the mean and median needs to be computed for each day, the same program can be used by scheduling it to run every 24 hours. To compute the monthly metrics, we can compute the daily metrics by running this

job every 24 hours and persist the daily metrics into a dataset. Then we can use the daily metrics dataset to get the monthly metrics. Similarly, yearly metrics can be derived from monthly metrics. This would ensure cost, performance and scalability rather than querying month and year datasets.

**UDFs used in the query:**
Two JavaScript UDFs are created for calculating mean and median each. They are hosted at Google Cloud Storage and can be accessed at:

- https://storage.googleapis.com/mybucket2209/udf_mean.js
- https://storage.googleapis.com/mybucket2209/udf_median.js

**Running the program:**

- Execute the attached jar using the above command with the right path to the jar file.

**OR**

- Unzip the coding_exercise.zip
- `mvn clean install –DskipTests`

```
 java -jar
/Users/SuryaSelvaraj/Documents/coding_exercise/target/bigquery-data-
1.0-SNAPSHOT-jar-with-dependencies.jar
```

Give the path of the jar in the above command.