**A short(?) description of the data pipeline**

1. **Reading** the text file as **JavaRDD<String> rdd.**
2. Apply **mapToPair** on **rdd** to get **JavaPairRDD<LocalDate, String> commits.**
3. **Cache** the **commits** rdd (we will use it once again when we search for the winner)
4. Apply **mapToPair** on **commits** to get **JavaPairRDD<LocalDate, Long> commitsCount.**  
   The long value is set to “1” which I will later aggregate since each row here represents a commit.
5. Apply **aggregateByKey** on **commitsCount** to get **JavaPairRDD<LocalDate, Long> dailyCommitsCountByDay.**  
   Now we have the commit-count for each day (not day-of-week).  
   The size of this rdd is bounded by the total number of days since GitHub was founded so for this specific granularity it might be faster to collect to driver ~3K values and calculate in memory {avg/std/pct} per day-of-week but for the more general case of granularity we will distribute.
6. **Cache** the **dailyCommitsCountByDay** rdd (we will need it later when we search for the winner )
7. Apply **mapToPair** on **dailyCommitsCountByDay** to get **JavaPairRDD<DayOfWeek, Long>** **dailyCommitCountByDOW.**Basically, I only change the key from LocalDate to DayOfWeek to make it easier to work with later on.  
   We still have the same set of values (commit count per day; not day-of-week)
8. Before handing the problem to SparkSQL we can consider use custom partitioner to have all elements of the same DayOfWeek on the same partition.  
   In minimal test I made it was a bit faster (2 sec) but sometimes just the same.  
   So I decided to keep it simple but this is something that can be tuned further.
9. Use **SparkSQL** to get **Dataset<Row>** **dataset** with the 3 required values.  
   The dataset’s size is 7 (day-of-week) so I collect it to the driver and build 3 Maps with DayOfWeek as key and { avg / std / pct } as value.
10. Apply **filter** on **dailyCommitsCountByDay** (cached) to get **JavaPairRDD<LocalDate, Long>** **anomalous.**  
    The logic of the filter is fairly simple because we know the {avg & std} of each day-of-week.   
    These 2 maps are part of the **AnomalousFilter** and are serialized to the executors. I find it more convenient than broadcast.
11. Apply **max** on **anomalous** to “collect” **to the driver** the maximum element from the anomalous.   
    This is the crazy-day: the anomalous day with maximum commits.  
    We use a **comparator** that compares according to the commit-count value.
12. Apply **filter** on **commits** (cached; hopefully; partially…) to get **JavaPairRDD<LocalDate, String>** **commitsOnCrazyDay.**  
    The **filter** logic is to retain only commits where the date equals the CrazyDay (compare date not day-of-week).
13. Apply **mapToPair** on **commitsOnCrazyDay** to get **JavaPairRDD<String, Long> userCommits.**The map puts the author as key and set the value of 1 for all items (to be aggregated later on).
14. Apply **aggregateByKey** on **userCommits** to get **JavaPairRDD<String, Long>** **commitsCountByUser.**  
    Simple “sum aggregation” to accumulate commits per key (=author)  
    So now I have all to authors that committed on the CrazyDay along with the commit-count for each of them.
15. Apply **max** on **commitsCountByUser** to “collect” **to the driver** the maximum element.  
    This is the winner: the author with the most commits on the CrazyDay.  
    I use the same (#11) **comparator** that compares according to the commit-count value.
16. From here on it is “JavaJava” to print it all.

Thanks You :-)