The result for Device 1054530426 on November 4, 2021, at 11:
There is no data for this Device after cleaning and filters that were applied to the datasets. Please look at the variable All listOfDevice1 in LogOfResults.pdf file that presents there is no data with these conditions.

The result for Device 3258837907 on November 4, 2021, at 11:
There is no data for this Device after cleaning and filters that were applied to the datasets. Please look at the variable All listOfDevice2 in LogOfResults.pdf file that presents there is no data with these conditions.

It is suggested to abandon operations on raw RDDs and use structured DataFrames (or Datasets if using Java or Scala) from the Spark SQL module. The creators made it very easy to create *custom partitioners* in this case.

PySpark partition is a way to split a large dataset into smaller datasets based on one or more partition keys. When you create a DataFrame from a file/table, based on specific parameters PySpark creates the DataFrame with a certain number of partitions in memory. This is one of the main advantages of PySpark DataFrame over Pandas DataFrame. Transformations on partitioned data run faster as they execute transformations parallelly for each partition. Dataframes, in the context of Spark SQL allow you to perform SQL like queries in order to play on your data.

PySpark supports partition in two ways; partition in memory (DataFrame) and partition on the disk (File system). Partitioning the data on the file system is a way to improve the performance of the query when dealing with a large dataset in the Data lake.

repartition() creates a specified number of partitions in memory. The partitionBy() will write files to disk for each memory partition and partition column. PySpark partitionBy() and repartition() help you partition the data and eliminate the Data Skew on your large datasets.