**Approach Followed**

Looking at the problem statement, it seemed an easy task and I estimated that it wouldn’t take much time, however as I started working on it, errors or internal hidden details started unfolding.

Initially tested it locally on my windows machine using Visual studio and Anaconda with packages such as pandas, numpy.

Other approach was to use EMR which is also fine or AWS Sagemaker notebook instances, however dependency management can be sometimes painful and needs more time in the beginning.

Later switched to ‘Databricks community cloud’ for working. This involves attaching python3 notebook to single node instance cluster.

Looking at data types being object and ‘cookTime’ & ‘prepTime’ column values it first occurred to me to strip string values using lambda functions, create new columns and then compute times in minutes. Later it occurred using ‘timedelta’ would be an easier approach, however based on past experience I was reluctant to go for it as I knew, it would then involve numpy and there are some known issues with numpy and pyspark (especially when it comes to time/date formats).

Anyways finally I choose ‘timedelta’ and proceeded with python code to solve problem. While working on problem statement, I was already thinking of converting data types for mainly cookTime, prepTime columns, however using timedelta avoids that. In Addition while working on pandas/numpy series, computation is easier based on indexes.

After testing the code in just python, I moved to customizing the same in pyspark to run over spark. Spark does lazy evaluation and that is good, however while building application we have to think about this as only on invocation of actions, some errors show up, such as ‘numpy being involved with timedelta’,’ data types casting issues’ if not taken care beforehand.

Other approaches could be choosing more partitions for spark to process and benefit from parallelism, caching or persisting at appropriate transformations, creating schema and applying datatypes instead of casting.

After whole customization, testing of code and reviewing final result, I then started finalizing the deliverable by making it more modular, separating config handling, error/log handling, managing packages and dependencies and a structured flow for main application.

Now was the main task to have it run on local cluster (standalone setup on windows/stand alone cluster setup on linux). Successfully did both setups and tested run of job on both.

Note\*\* Stand alone cluster with 2 worker nodes, spark-2.4.3-built for Hadoop 2.6.5 (apache core) was used on ubuntu machines(VMBOX).

Noted issues are spark dependencies, python dependencies with version mismatches, files/packages to be available on all worker nodes and memory configuration had to be faced on the way.

Forgot to attach the logs of job runs or screenshot from webUI, however that can re regenerated.

Finally tested the code on spark stand alone setup on windows machine. (pyspark & spark-submit).

**Could haves:**

* Main application could be written better with UDFs being managed in a more micro approach/modular approach.
* Partitioning and caching options should/could have been used for more parallelism or fast run.
* Spark’s pipelining approach could be used in a better way.
* Config.json can be populated with spark properties to allow choosing/customizing ‘no of executors, no of cores, memory to be used’.
* Dataset API instead of DF API could also be used.