Knowledge of cricket?

On a scale of 0-5, I would rate myself a 3. A seldom watch cricket and know almost all the rules

If the use case was extended to required incrementally loading new match data on a go-forward basis, how would your solution change?

I would schedule to run the pipeline regularly using **Airflow**. Airflow is very helpful in automation management. I would use the python operator and call the class from the main file. If the dataset increases **Kafka** can also be used for streaming and batch transfers.

Can you provide an example of when, during a project or analysis, you learned about (or created) a new technique, method, or tool that you hadn't known about previously? What inspired you to learn about this and how were you able to apply it?

During my previous pyspark project, I had to built a customisation file in which a user would write the rules that should be applied to the data that is downloaded i.e. the pre-processing would be user-specified. So, a set of rules had to be built which the user could use to specify the constraints. While some of the rules were easy to build but others were difficult. One such rule was specifying one-to-one relation betweent two columns i.e. take only those values from the first column

which have only one matching value in the second column. At first, I tried to look for ways to unique pairs between the columns but that didn't suffice. Then, I tried to find ways to count the occurrence of pairs and after a lot of searching I devised a window function code as follows

```
def one_one(df, col1, col2):
     window_spec = Window.partitionBy(col1)
     return df.withColumn(f'cnt_{col2}',
                size(collect_set(col2).over(window_spec))) \
              .filter((col(f'cnt_{col2}') == 1) | (col(col2).isNotNull())) \
              .drop(col(f'cnt_{col2}'))
```

But still this was a bit confusing and the window function was giving some issues so I devised a simpler way of doing this and here is the final piece of code

```
def one_one(df, col1, col2):
    tdf = df.groupBy(col1, col2).count().groupBy(col1).count()
    return df.join(tdf, on = col1, how = 'left')\
             .filter((col('count') == 1) | (col(col2).isNotNull())).drop('count')
```

Apart from that while trying different options of transferring data to mongodb, I was facing an issue with an option. After a lot of searching, I eventually realised it was a **bug** and reported it **here**. It has been resolved in the latest version of mongo-spark connector 10.2.0.