1. Download the data from the given URL : https://www.kaggle.com/datasets/kimjihoo/coronavirusdataset

2. Create a producer with a python connector in confluent kafka and stream your data.

3. Consume your data through the python connector and dump it in mongodb atlas.

   Note: Here in the dataset you will be finding a multiple files you need to use all file for the kafka and mongodb

4. Collect your data as a pyspark dataframe and perform different operations.

   Note: Consider only three files for creating a dataframe among all **case**, **region** and **TimeProvince**

   a. Read the data, show it and Count the number of records.
   b. Describe the data with a describe function.
   c. If there is any duplicate value drop it.
   d. Use limit function for showcasing a limited number of records.
   e. If you find the column name is not suitable, change the column name.[optional]
   f. Select the subset of the columns.
   g. If there is any null value, fill it with any random value or drop it.
   h. Filter the data based on different columns or variables and do the best analysis.

      For example: We can filter a data frame using multiple conditions using AND(&), OR(|) and NOT(~) conditions. For example, we may want to find out all the different

infection_case in Daegu Province with more than 10 confirmed cases.

i. Sort the number of confirmed cases. Confirmed column is there in the dataset. Check with descending sort also.
j. In case of any wrong data type, cast that data type from integer to string or string to integer.
k. Use group by on top of province and city column and agg it with sum of confirmed cases. **For example** `df.groupBy(["province","city"]).agg(function.sum("confirmed")`
l. For joins we will need one more file.you can use region file. User different different join methods.**for example** `cases.join(regions, ['province','city'],how='left')` `You can do your best analysis.`

5. If you want, you can also use SQL with data frames. Let us try to run some SQL on the cases table.

   For example:

```
cases.registerTempTable('cases_table')
newDF = sqlContext.sql('select * from cases_table where confirmed>100')
newDF.show()
```

   Here is a example how you can use df for sql now you can perform various operations with **GROUP BY, HAVING, AND ORDER BY**

6. **Create Spark UDFs**

   Create function **casehighlow()**

   If case is less than 50 return low else return high

   convert into a UDF Function and mention the return type of function.

   Note: You can create as many as udf based on analysis.