PYSPARK ASSIGNMENT

PEER LEARNING DOCUMENT

Problem Statement -

Use Python to make a request to https://rapidapi.com/Gramzivi/api/covid-19-data/ for at least 20 countries using '/getLatestCountryDataByName' API, fill up the csv file, create a dataFrame using Spark. Using the Above dataFrame, find out the following:

2.1) Most affected country among all the countries (total death/total covid cases).
2.2) Least affected country among all the countries (total death/total covid cases).
2.3) Country with highest covid cases.
2.4) Country with minimum covid cases.
2.5) Total cases.
2.6) Country that handled the covid most efficiently(total recovery/ total covid cases).
2.7) Country that handled the covid least efficiently(total recovery/ total covid cases).
2.8) Country least suffering from covid (least critical cases).
2.9) Country still suffering from covid (highest critical cases).

Create a RestFul API to show datas collected in question 1. Create each RestFul API to show the result of every sub question in question 2. Please pay attention to API's naming conventions.

Aayush Sinha

Approach -

He had created two files named

- Api.py
- Process.py

Process.py

He created Python class called Process that processes COVID-19 data from an API and performs given queries on the data using the PySpark library. Here is a brief overview of the methods:

- __sanitise(self, state): A private method that removes any asterisks from the given state name.
- __load_dataset(self, key): A private method that loads the COVID-19 data from an API using a given API key, and returns the data as a list of lists.
- __create_dataframe(self, dataList): A private method that creates a PySpark DataFrame from the given list of lists.
- __load_dataframe(self): A private method that loads the COVID-19 data from the API and creates a PySpark DataFrame from it.
- __create_affected_df(self): A private method that creates the affectedDF DataFrame.
- __create_handled_df(self): A private method that creates the handleDF DataFrame.
- get_most_affected(self): A public method that returns the state with the highest ratio of deaths to total cases.
- get_least_affected(self): A public method that returns the state with the lowest ratio of deaths to total cases.
- get_most_total(self): A public method that returns the state with the highest number of total cases.
- get_least_total(self): A public method that returns the state with the lowest number of total cases.
- get_total(self): A public method that returns the total number of COVID-19 cases across all states.
- get_most_handled(self): A public method that returns the state with the highest ratio of cured cases to total cases.
- get_least_handled(self): A public method that returns the state with the lowest ratio of cured cases to total cases.
- get_data(self): A public method that returns the entire COVID-19 data as a list of dictionaries, where each dictionary represents a row in the PySpark DataFrame.

Api.py

He defined a Flask app with several routes that return COVID-19 related data for different states in India. The data is obtained from an instance of the Process class, which is imported from a module named process.

The available routes are:

- /affected/most: returns the state with the most number of active COVID-19 cases.
- /affected/least: returns the state with the least number of active COVID-19 cases.
- /total/most: returns the state with the most number of total COVID-19 cases.
- /total/least: returns the state with the least number of total COVID-19 cases.
- /total: returns the total number of COVID-19 cases in India.
- /handled/most: returns the state that has handled the most number of COVID-19 cases.
- /handled/least: returns the state that has handled the least number of COVID-19 cases.
- /data: returns all the COVID-19 related data for all states in India.

Rohith Boodireddy

Approach -

He had created two files named

- App.py
- dataframe.py

Dataframe.py

1. The getData() function sends a GET request to the RapidAPI endpoint for COVID-19 data and returns the response. The cleanData() function performs several cleaning steps on the resulting dataframe, such as dropping a _corrupt_record column, removing rows where the state column is null or empty, casting the confirm, cured, and death columns to the Long datatype, stripping state names that end in *, and selecting only specific columns.
2. He then created a Spark session and context, sets the log level to ERROR, and reads the JSON data from the response into a PySpark dataframe. Finally, the cleanData() function is called to clean the dataframe, and the resulting cleaned dataframe is stored in the covidData variable.

App.py

He wrote a Python script that creates a Flask application with various endpoints to extract insights from a Spark dataframe containing data about COVID cases in India. The endpoints include:

- /: provides a list of available endpoints
- /get_csvfile: exports the Spark dataframe to a CSV file stored on the desktop
- /most_affected_state: returns the state with the highest death to confirmed cases ratio
- /least_affected_state: returns the state with the lowest death to confirmed cases ratio
- /highest_covid_cases: returns the state with the highest number of confirmed COVID cases
- /least_covid_cases: returns the state with the lowest number of confirmed COVID cases
- /total_cases: returns the total number of confirmed COVID cases in India
- /most_efficient_state: returns the state with the highest cured to confirmed cases ratio
- /least_efficient_state: returns the state with the lowest cured to confirmed cases ratio

The various endpoints use PySpark to calculate the required insights and return them as JSON objects using Flask's jsonify function. The /get_csvfile endpoint exports the Spark dataframe to a CSV file using the com.databricks.spark.csv format and the repartition method to ensure that the output is a single file.