JULIEN GODFROY S427183

ANALYSIS OF THE COVID-19 GLOBAL DATASET USING HADOOP/SPARK

MACHINE LEARNING & BIG DATA

CSTE CIDA Option

Table of contents

I - Introduction

During the Covid-19 pandemic, the global community has faced vital challenges that have necessitated a comprehensive response from the scientific, medical, and data science communities. The use of Big Data technologies has made it possible to monitor and analyse the spread of the virus, inform public health decisions and facilitate resource allocation. This report describes the methodologies and results of an in-depth analysis of Covid-19 data using a data processing framework, namely Apache Spark.

The project at hand aims to distil meaningful insights from the "*time_series_covid19_confirmed_global*" dataset, a comprehensive aggregation of the daily confirmed cases of Covid-19 across various global coordinates, spanning from January 22nd, 2020 to the 10th March 2023. This dataset was meticulously updated on a daily basis and reflects cumulative figures, providing a fertile ground for time series analysis and epidemiological study.

The primary objective of this study is to execute a series of data-driven queries to reveal the pandemic's trajectory. These queries are threefold: first, to calculate the mean number of daily confirmed cases per country for each month; second, to compute descriptive statistics for the continents' weekly confirmed cases, narrowing the focus to the 100 most impacted states; and third, to employ K-means clustering to categorize the 50 states most affected on a monthly basis, based on the progression of daily cases.

Through the execution of these queries, the report will offer a view of the pandemic's progression, provide statistical summaries, and group similar trajectories using clustering techniques. The report will further present a comparative analysis of algorithmic efficiency and processing times, offering insights into the practical application of big data tools in real-world scenarios.

Beyond the technical and analytical achievements, the report will also consider the ethical dimensions of data usage and privacy considerations in pandemic data handling. The concluding section will encapsulate the findings, highlight the implications for data science in public health, and suggest avenues for future research.

II - Background & Methodology

The Covid-19 pandemic has triggered an unprecedented global health crisis, affecting nations on every continent. In response, data scientists and healthcare professionals have turned to the mass of data generated as a crucial tool for understanding and mitigating its spread. At the heart of this data-driven approach is the *'time_series_covid19_confirmed_global'* dataset, which forms the basis of our analysis.

Dataset Description

The dataset for this study is a CSV file titled "*time_series_covid19_confirmed_global*"[1], which is composed of a series of records representing the cumulative number of confirmed Covid-19 cases across the world. Each record contains fields for the province or state, country or region, latitude, and longitude, followed by a time series of dates starting from the 22nd of January 2020 to the 10th of March 2023. For each date, the dataset reports the total number of confirmed cases up to that day. This structured format allows for the data to be parsed and analysed over temporal and spatial dimensions, offering a wide view of the pandemic.

The header of the dataset outlines the structure, with columns designated for geographical identifiers and each subsequent column representing a single day's total confirmed case count.

Methodology

Our methodology uses the PySpark library, chosen for its robust data processing capabilities, which are particularly well suited to handling large datasets. PySpark allows us to perform complex data transformations and analyses, which are crucial to answering the queries posed in our study.

The analyses will be run on a single-node cluster in a Docker container, which runs on Windows Subsystem for Linux (WSL2) [2]. The computer running WSL2 is a Windows laptop with 16GB of RAM and an Intel Core i5-1135G7 (2.4GHz). This configuration, practical for educational and development purposes, has its limitations. In terms of performance, the single-node cluster is slower than a multi-node environment and may not take full advantage of distributed computing, which is a key benefit of using frameworks such as Hadoop and Spark.

In addition, memory consumption is an important consideration. PySpark applications can be memory intensive, and running in a Docker container could add an additional layer of resource constraints due to the memory limits of the container. To mitigate these challenges, we need to ensure that allocated resources are managed efficiently and that the Docker container is configured with enough memory to handle the workload.

Throughout the data processing tasks, we will follow a sequence of steps: data ingestion, data pre-processing, analysis and results extraction. Each step is designed to systematically address the queries set by the project guidelines while optimising the constraints of our processing environment.

III – Application & Results

Data ingestion and consolidation

Although the dataset studied has not been updated since March 2023, an update system has been implemented. To do this, a python script *update_clean_data.ipynb* was created. It is used to retrieve the latest updated data from GitHub using the *requests* library, which enables HTTP GET requests to be made.

Once the raw data has been retrieved, it is stored in a CSV file with the current date in the */data/raw folder*. If an old dataset backup is already present, it is deleted.

Once the raw data has been stored, the same script cleans and consolidates the data.

Although the dataset is well maintained overall, 2 lines with missing data are still present. Considering that these 2 lines are not at all representative, the decision was made to delete them directly.

Province/State	Country/Region	Lat	Long	1/22/20	1/2
Repatriated Travellers	Canada			0	
Unknown	China			0	

Figure 1 : Two lines with missing data

Next, to perform all the queries requested, we had to determine the continent of each country. To do this, two steps were necessary:

- Determine the country from the geographical coordinates (using the *reverse_geocode* library) [3]
- Determine the continent from the country [4].

If the *Province/State* column is empty, it's filled with the *Country/Region* column

A once the clean and consolidated with the column continent, a backup is made in the file */data/cleaned/**date_of_the_day***. If a previous backup exists, it is deleted. This file will be the source for each analysis.

Query 1: Mean number of cases daily for each month for each country

For each country, the mean number of confirmed cases daily for each month in the dataset has been computed. To do so, after loading the dataset as PySpark dataframe, it has been pivoted, so that there is one column with all the dates, as shown in Figure 2.

Country/Region	date	cases	year	month
Afghanistan	2020-01-22	0	2020	1
Afghanistan	2020-01-23	0	2020	1
Afghanistan	2020-01-24	0	2020	1
Afghanistan	2020-01-25	0	2020	1
Afghanistan	2020-01-26	0	2020	1

Figure 2 : Pivoted PySpark dataframe

Once the dates have been converted from text format to date format, the month and year can be extracted to group the data by month and calculate the average number of confirmed cases. After these steps, the final result dataframe is saved inside a CSV file in the folder */query_results/**date_of_the_day**_query1*. If another save of the results exists, it's removed.

It is interesting to check the calculated mean with public records. After validating with the data available from the World Health Organization website [5] (figure 3 & 4), it seems like the script is working properly.

However, it's interesting to see that from the studied datased, the China doesn't show up as one of the most affected countries by Covid. This may raise questions about the accuracy of the data we are analysing, or even political questions about the reporting of cases in a context of global difficulties. This will be discussed later in this report.

Country/Reg	year	month_name	mean_cases
US	2023	March	103,663,894.77777
US	2023	February	102,988,048.32142
US	2023	January	101,693,794.83870
India	2023	March	44,689,348.44444
India	2023	February	44,685,755.07142
India	2023	January	44,682,345.45161
France	2023	March	39,845,947.66666
France	2023	February	39,777,771.42857
France	2023	January	39,648,640.48387
Germany	2023	March	38,217,536.44444
Germany	2023	February	37,963,204.39285
Germany	2023	January	37,612,321.90322
Brazil	2023	March	37,074,974.11111
Brazil	2023	February	36,948,450.75000
Brazil	2023	January	36,621,050.77419
Japan	2023	March	33,281,061.88888
Japan	2023	February	32,989,579.10714
Japan	2023	January	31,277,941.87096
Korea, South	2023	March	30,572,608.66666
Korea, South	2023	February	30,381,115.53571
Korea, South	2023	January	29,798,656.93548
Italy	2023	March	25,597,586.00000

Global	771,679,618
United States of A...	103 436 829
China	99 317 967
India	45 001 245
France	38 997 490
Germany	38 437 756
Brazil	37 721 749
Republic of Korea	34 571 873
Japan	33 803 572
Italy	26 230 177

Figure 3 : Mean cases for each month, for each country query result on the 7th of November 2023, compared with the WHO public data

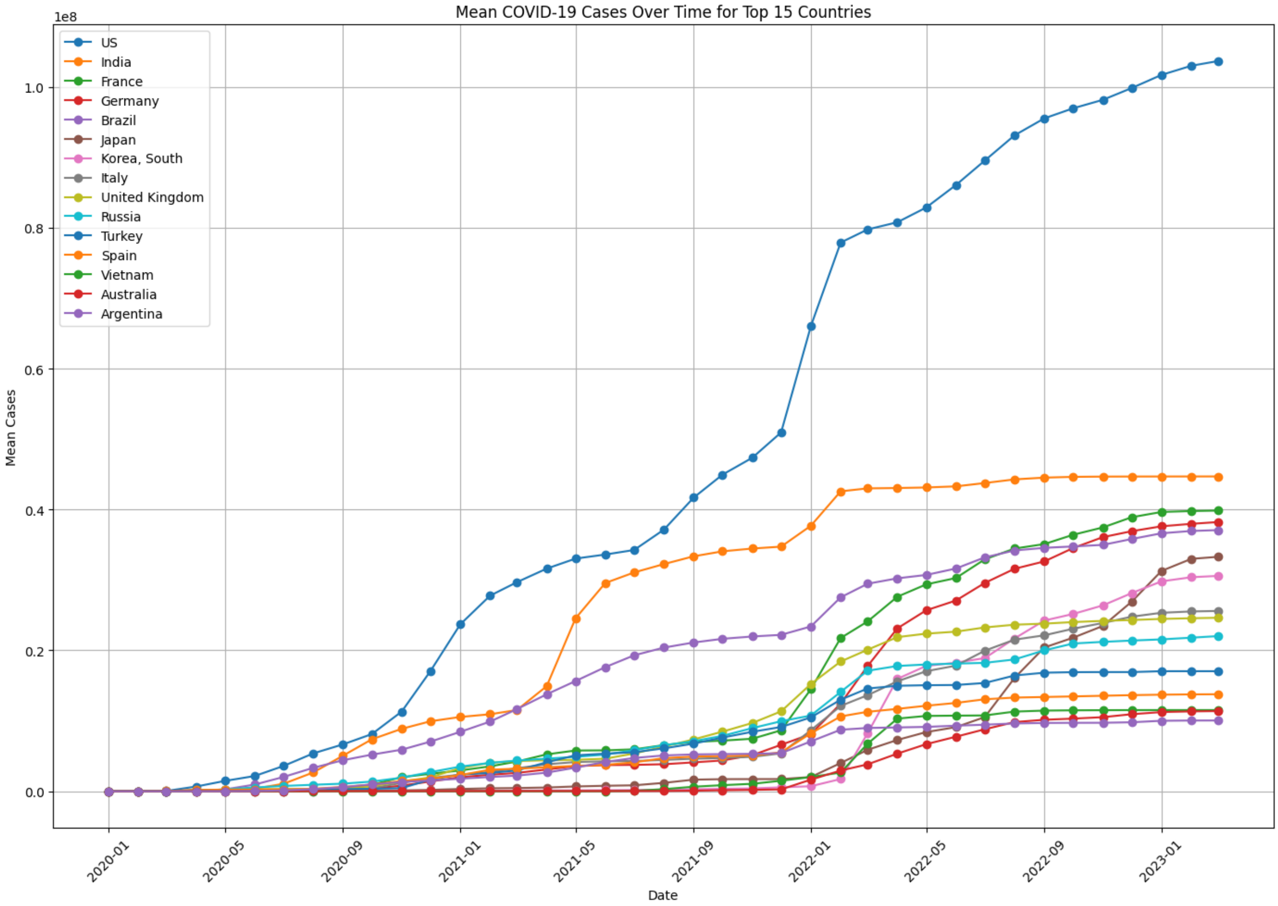

Figure 4 : Mean COVID-19 for the 15 most infected countries.

Query 2: Mean, Standard deviation, min, max each for each continent, each week

The second request in this project is to analyse the Covid-19 pandemic by focusing on weekly trends in confirmed cases by continent. The aim of this analysis is to identify and characterise variations in the evolution of the pandemic by calculating the descriptive statistics - mean, standard deviation, minimum and maximum of new cases confirmed daily for each week.

The first step was to transform the dataset, which was structured in wide format, into long format for easier handling, in particular with a date column. The data was then filtered to retain only the hundred most affected states or provinces, using the coefficient of the regression line for daily increases in cases as a selection criterion. This coefficient represents the slope of the trend in confirmed cases, providing a quantitative means of determining the most affected areas.

With these states identified, the data was then grouped by continent and week, where we calculated the desired statistics. Grouping by week allowed us to capture the temporal evolution of the pandemic, while classifying by continent provided a global perspective.

The results show significant variations between continents and within the weeks analysed. Some continents showed high peaks in confirmed cases, while others maintained relatively low levels, reflecting the diversity of responses to the pandemic and the varying impact of the virus around the world. These differences are reflected not only in the calculated averages, but also in the standard deviations, which indicate the degree of variation in the daily increases in cases.

The average of confirmed cases gives an indication of the overall disease burden in a given week, while the standard deviation reflects the constancy or variability of cases on a day-to-day basis. The minimum and maximum values, meanwhile, capture the extent of daily fluctuations in each continent, providing an insight into the volatility of epidemic trends.

It's interesting to note that in *minimum* column, few negative values can be found (figure 5). It can be explained by the fact that some countries apply correction after having declared to many confirmed cases the previous days.

The figure 6 shows the mean confirmed cases trend each week. Here again, some doubts about the Asian figures appear.

The final result dataframe is saved inside a CSV file in the folder */query_results/**date_of_the_day**_query2*. If another save of the results exists for this query, it's removed.

Continent	Week	Mean	Standard_	Min	Max
Africa	1	690.3619048	1438.82123	0	9358
America	1	18309.26923	91579.7953	-1029	1042792
Asia	1	6807.956522	23992.7791	0	245542
Europe	1	13303.65714	40434.4427	0	372766
Oceania	1	2918.995671	8493.25429	-137	51089
Africa	2	926.3904762	2316.79732	0	16256
America	2	19978.51832	100107.82	0	1354505
Asia	2	6883.190476	20277.432	0	198873
Europe	2	13317.88163	39668.4389	0	359550
Oceania	2	3461.082251	11214.6449	-105	92264
Africa	3	1278.095238	4127.36105	0	31401
America	3	19115.84615	89142.5298	0	1132846
Asia	3	6566.915114	19320.3362	0	243295
Europe	3	15156.57687	45513.0035	0	461517
Oceania	3	2046.220779	6084.15183	-266	32297
Africa	4	1250.646154	5243.3517	0	46914
America	4	12878.77071	62723.483	-1524	908747
Asia	4	5912.071906	17659.9507	0	212234
Europe	4	13389.82088	41184.2975	0	501635
Oceania	4	1496.79021	4986.57089	-129	50258
Africa	5	848.9071429	4128.0338	0	45474
America	5	8612.646978	38468.4569	-44	535312
Asia	5	5520.212733	16379.7052	-13	111157
Europe	5	12308.27245	43978.2588	-781	847371
Oceania	5	556.8896104	3925.05323	-53304	14553

Figure 5 : Mean, Std_dev, Min and Maximum for each week, for each continent on the 7th of November.

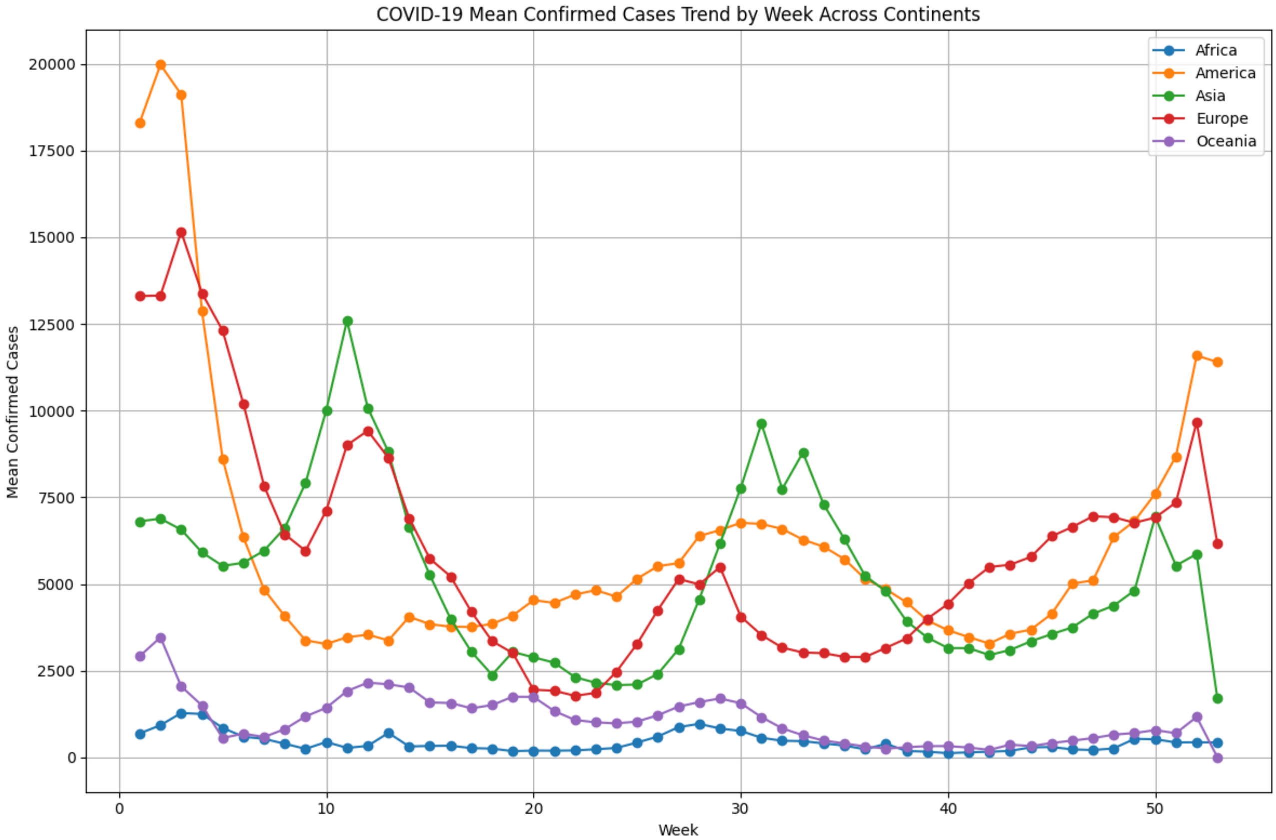

Figure 6 : Mean confirmed cases trend by week for each continent

Query 3: K-means clustering

The third request of the project is to examine the 50 states or nations most affected by the COVID-19 pandemic on a monthly basis, determined by the trend coefficient [6] of confirmed cases. The study uses the K-means algorithm [7] with K=4 to group these states according to their monthly epidemiological profiles.

In this study, a transformation of the initial data was carried out to prepare them for the application of the K-means algorithm. The data were restructured again in long format, and the dates were converted into the appropriate format. For each month, the analysis grouped the data by state or nation, then calculated the linear trend coefficients for the confirmed cases.

The next step was to sort these coefficients to identify the 50 regions most affected each month. Once these states had been selected, the feature vector only based on the trend coefficient was constructed to serve as the basis for the clustering algorithm.

The K-means algorithm was applied separately (figure 7) for each monthly group to form four clusters, each grouping together states showing similar patterns in the daily increase in confirmed cases. Predictions from the model were used to determine which states belonged to each cluster. From the result dataframe, each of the cluster has been placed on a map to ease to visulisation of the data (figure 8).

Finally, the result dataframe is saved inside a CSV file in the folder */query_results/**date_of_the_day**_query3*. If another save of the results exists for this query, it's removed.

Province_State	Country_Re	Month	TrendlineCoef	Cluster
Afghanistan	Afghanistan	5	435.6725769	1
Afghanistan	Afghanistan	6	1634.108398	1
Afghanistan	Afghanistan	6	553.868103	0
Albania	Albania	2	1053.830078	1
Alberta	Canada	5	1134.825806	3
Alberta	Canada	4	156.5103455	1
Alberta	Canada	9	1548.696289	1
Alberta	Canada	5	519.6116943	1
Algeria	Algeria	5	180.2378998	1
Algeria	Algeria	7	545.3237915	1
Algeria	Algeria	8	451.4903259	1
Anhui	China	1	26.40606117	3
Anhui	China	2	22.7857151	3
Argentina	Argentina	7	4096.895996	3
Argentina	Argentina	10	13738.21973	3
Argentina	Argentina	1	10394.36719	3
Argentina	Argentina	4	22153.9082	3
Argentina	Argentina	1	100408.9688	3
Argentina	Argentina	8	7228.54248	2
Argentina	Argentina	6	21699.28906	2

Figure 7 : Clusterisation of Province/State into 4 groups using the "TrendlineCoef" feature.

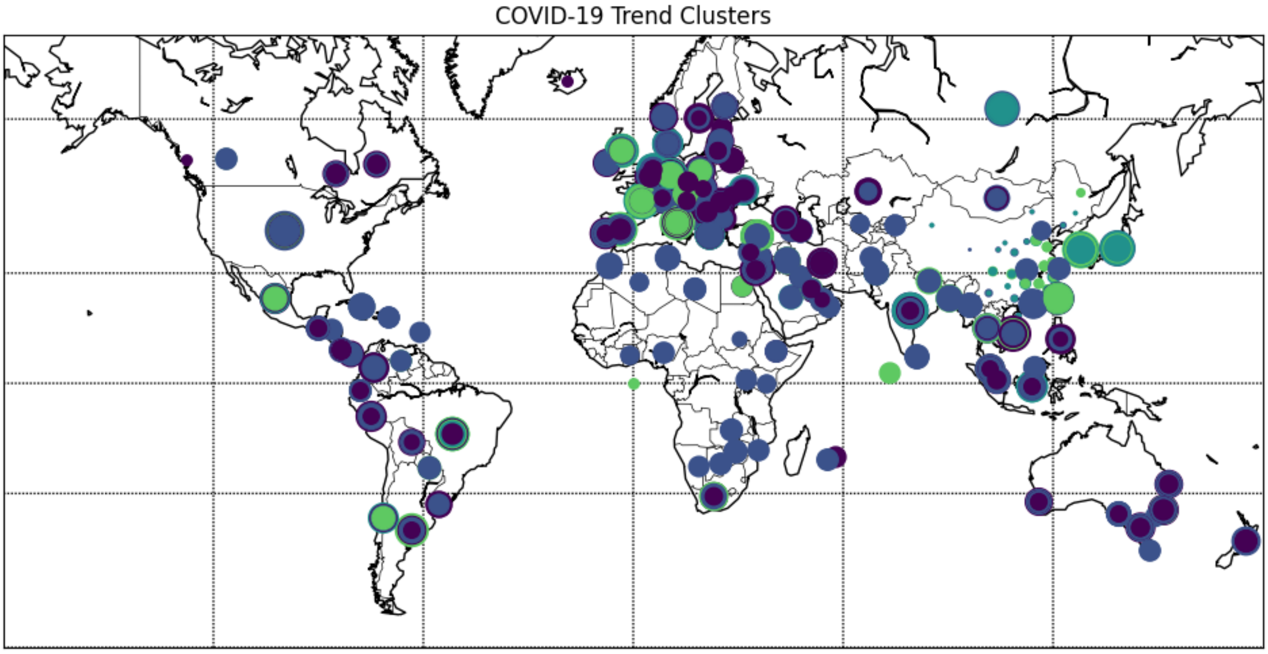

Figure 8 : Visualisation of the 4 clusters on a map.

Processing time & resource consumption

query_nb	exec_time(s)
query1	18.8
query2	17.7
query3	90.6

Figure 9 : Processing time for each query.

The records of the processing time for each query starts after loading the spark session and the CSV in spark dataframe (figure 6). The goal here is to analyse the time to realise the Spark operations.

Analysis of the query execution times reveals a significant disparity in performance on the given infrastructure, a Windows laptop equipped with a 2.4GHz Intel Core i5-1135G7 processor and 16GB of RAM running Spark on a single cluster in a Docker container. Queries 1 and 2, with execution times of 18.8 seconds and 17.7 seconds respectively, suggest that the operations performed (such as statistical calculations and data transformations) are managed efficiently, taking advantage of the computer's memory management and processing capabilities.

On the other hand, query 3, with an execution time of 90.6 seconds, is much more resource-intensive. This difference can be explained by the increased complexity of the operations, including the calculation of trend coefficients, classification and above all the application of the K-means algorithm, which is iterative and can be computationally intensive when large quantities of data are involved.

IV - Conclusion

In conclusion, the present study carried out an in-depth exploration of trends in confirmed cases of COVID-19, focusing on different spatial and temporal analyses on a global scale.

The use of the Pyspark library on a single cluster, while sufficient for basic tasks, revealed its limitations when performing more complex processes such as the application of clustering algorithms. This underlines the importance of rigorous assessment of computing resource requirements and code optimisation for the efficient management of large volumes of data [8].

This research also highlights the potential of massive data analysis to inform the response to global epidemiological events, and argues for the more systematic use of such methodologies for real-time surveillance and disease prevention.

V – Ethical challenges

The ethical issues and challenges associated with the use of Machine Learning and Big Data have become increasingly worrying, especially in critical scenarios such as the COVID-19 pandemic. The veracity of data provided by governments represents a major ethical issue, particularly when it comes to data that influences political and public health decisions. The example of China, which according to WHO figures is the second most affected country in terms of the number of cases, but which does not appear in the TOP 15 in the dataset analysed, raises questions about the reliability and transparency of the information provided, or the way in which our dataset is put together.

It is therefore essential to maintain a critical and analytical eye when examining the data, recognising that data can be manipulated. Data-driven studies must always be conducted with an ethical conscience, checking and questioning the provenance and integrity of the data used. This is imperative not only to ensure the accuracy of analyses, but also to preserve public trust and fairness in responses to crises such as pandemics.

References

[1] CSSE at Johns Hopkins University, time_series_covid19_confirmed_global.csv [Internet], Baltimor: 2023 Mar 10 [cited 2023 Nov 7]. Available from: https://github.com/CSSEGISandData/COVID-19/blob/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv

[2] Josh Lane, pyspark-devcontainer [Internet], 2023 Jul 11 [cited 2023 Nov 7]. Available from: https://github.com/jplane/pyspark-devcontainer

[3] Jeff Tanner, pycountry-convert [Internet], 2017 Mar 12 [cited 2023 Nov 7]. Available from: https://github.com/jefftune/pycountry-convert

[4] Benedict Neo, Get Continent Names from Coordinates Using Python [Internet], 2022 Nov 8 [cited 2023 Nov 7]. Available from: https://medium.com/bitgrit-data-science-publication/get-continent-names-from-coordinates-using-python-8560cdcfdfbb

[5] World Health Organization, WHO Coronavirus (COVID-19) Dashboard [Internet], [cited 2023 Nov 7]. Available from: https://covid19.who.int/table

[6] Jagdeesh, PySpark Linear Regression – How to Build and Evaluate Linear Regression Models using PySpark MLlib [Internet], [cited 2023 Nov 7]. Available from: https://www.machinelearningplus.com/pyspark/pyspark-linear-regression/

[7] Angel Das, K Means Clustering using PySpark on Big Data [Internet], 2021 Feb 11 [cited 2023 Nov 7]. Available from: https://towardsdatascience.com/k-means-clustering-using-pyspark-on-big-data-6214beacdc8b

[8] Alina Khay, Mastering PySpark Resource Management: Strategies for Maximising Efficiency & Performance, 2023 May 7, [cited 2023 Nov 7]. Available from: https://medium.com/@alinakhay/mastering-pyspark-resource-management-strategies-for-maximising-efficiency-performance-44da24a7a5b

Appendix – Source Code

QUERY 1:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr
from utils.load_data import *
from utils.remove_folder import *
from pyspark.sql import functions as F
from pyspark.sql.types import DateType
import datetime
```

```
#Create SparkSession
spark = SparkSession.builder \
      .master("local[1]") \
      .appName("PySpark project") \
      .getOrCreate()

spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY") #to avoid any date conversion issue
```

```
#Store the csv into df
data_path = "data"
df = load_csv_from_cleaned_folder(spark, data_path)
df.show(5)
```

```
#get the list of date columns (assuming the 7th column onwards are date columns)
date_columns = df.columns[6:]

#converting from wide to long format for the dates
df_pivot = df.select(
    "Country/Region",
    F.explode(
        F.array([
            F.struct(F.lit(c).alias("date"), F.col(c).alias("cases"))
            for c in date_columns
        ])
    ).alias("date_cases")
).select("Country/Region", "date_cases.*")

#Converting string dates to date type with the correct format
df_pivot = df_pivot.withColumn("date", F.to_date(F.col("date"), "MM/dd/yy"))


#extracting year and month number from the date for grouping
df_pivot = df_pivot.withColumn("year", F.year(F.col("date")))
df_pivot = df_pivot.withColumn("month", F.month(F.col("date")))
print(df_pivot.show(5))

#grouping by country, year, and month to calculate the mean cases
#mean should be calculated on daily data, hence count distinct dates for each group
df_daily_stats = df_pivot.groupBy("Country/Region", "year", "month").agg(
    (F.sum("cases") / F.countDistinct("date")).alias("mean_cases")
)

df_daily_stats.show(5)
```

```
#convert month number to month Name
month_df = df_pivot.select(
    F.month("date").alias("month_num"),
    F.date_format(F.col("date"), "MMMM").alias("month_name")
).distinct()


#joining to get month names
result_df = df_daily_stats.join(month_df, df_daily_stats.month == month_df.month_num)
#selecting the usefull cols
result_df = result_df.select(
    "Country/Region",
    "year",
    "month_name",
    "mean_cases"
)

#sorting the result by "Country/Region", then year, then month
result_df = result_df.orderBy("Country/Region", "year", "month_num")

result_df.show(50)
rm_dir("query_results", "*_query1")

#write result_df to csv
today = datetime.date.today().strftime("%Y%m%d")
file_path = os.path.join("query_results", f"{today}_query1")


result_df.write.format('com.databricks.spark.csv').mode('overwrite').option("header", "true").save(file_path)
print(f"Saved : {file_path}")
```

Query 2:

```
from pyspark.sql import SparkSession
from  utils.load_data import *
from utils.remove_folder import *
from pyspark.sql.functions import col, expr
from pyspark.sql import functions as F
from pyspark.sql.window import Window
import datetime

#Create SparkSession
spark = SparkSession.builder \
        .master("local[1]") \
        .appName("PySpark project") \
        .getOrCreate()

spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")

#load clean Data
data_path = "data"
df = load_csv_from_cleaned_folder(spark, data_path)
df.show(5)
```

```
#rename columns with special characters for easier handling
df = df.withColumnRenamed("Province/State", "Province_State").withColumnRenamed("Country/Region", "Country_Region")
# List of columns that are not dates
non_date_cols = ['Province_State', 'Country_Region', 'CountryCode', 'Continent', 'Lat', 'Long']

date_cols = [col for col in df.columns if col not in non_date_cols]

#stack the date columns to long format
stack_expr = "stack(" + str(len(df.columns) - len(non_date_cols)) + ", " + \
    ", ".join(["'" + c + "', `" + c + "`" for c in df.columns if c not in non_date_cols]) + \
    ") as (Date, Confirmed)"
df_pivot = df.selectExpr(*non_date_cols, stack_expr)

#convert Date column to a proper date format
df_pivot = df_pivot.withColumn("Date", F.to_date("Date", "MM/dd/yyyy"))

#calculate daily increments
partition = Window.partitionBy("Province_State").orderBy("Date")
df_pivot = df_pivot.withColumn("Daily_Increment", F.col("Confirmed") - F.lag("Confirmed", 1, 0).over(partition))

#calculate the regression reg
#but first, create an index for the Date to treat it as numeric
df_pivot = df_pivot.withColumn("DateIndex", F.row_number().over(Window.orderBy("Date")))

reg_formula = F.covar_pop("DateIndex", "Daily_Increment") / F.var_pop("DateIndex")

#calculate the regression reg for each state/province
reg_df = df_pivot.groupBy("Province_State").agg(reg_formula.alias("regression_reg"))

#get top 100 states/provinces with the highest regs
top_states = reg_df.orderBy(F.desc("regression_reg")).limit(100).select("Province_State")

#filter the original datagrame with these top states
filtered_df = df_pivot.join(top_states, "Province_State")

filtered_df.show(10)
#calculate stats
stats_df = filtered_df.groupBy("Continent", F.weekofyear("Date").alias("Week")).agg(
    F.mean("Daily_Increment").alias("Mean"),
    F.stddev("Daily_Increment").alias("Standard_Deviation"),
    F.min("Daily_Increment").alias("Min"),
    F.max("Daily_Increment").alias("Max")
)
stats_df = stats_df.orderBy("Week", "Continent")


stats_df.show()
rm_dir("query_results", "*_query2")

#write result_df to csv
today = datetime.date.today().strftime("%Y%m%d")
file_path = os.path.join("query_results", f"{today}_query2")


stats_df.write.format('com.databricks.spark.csv').mode('overwrite').option("header", "true").save(file_path)
print(f"Saved : {file_path}")
```

Query 3:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark.sql.types import IntegerType, DateType, FloatType
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from scipy.stats import linregress
from functools import reduce
from pyspark.sql.functions import udf, to_date, rank, month, year, dayofmonth, collect_list
from pyspark.ml.linalg import Vectors
from utils.load_data import *
from utils.remove_folder import *
import datetime
from pyspark.sql.functions import col, lit
from pyspark.sql import DataFrame

# SparkSession
spark = SparkSession.builder \
        .master("local[1]") \
        .appName("PySpark project") \
        .getOrCreate()
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
```

```
#fit the KMeans model and make predictions for each group
def fit_kmeans(df):
    model = kmeans.fit(df)
    return model.transform(df)

#this method groups the data by year and month, applies KMeans, and returns a pyspark dataframe
def get_month_clust(df_features):
    clusters_list = []
    for (year, month), group_df in df_features.toPandas().groupby(['Year', 'Month']):

        group_spark_df = spark.createDataFrame(group_df)
        #apply KMeans
        predictions_df = fit_kmeans(group_spark_df)
        #add the 'Year' and 'Month' columns with the corresponding group values
        predictions_df = predictions_df.withColumn("Year", lit(year))
        predictions_df = predictions_df.withColumn("Month", lit(month))
        clusters_list.append(predictions_df)

    # Combine all the cluster dataframes together
    return reduce(DataFrame.unionAll, clusters_list)

#udf to calculate the trendline coefficient
def trendline_coef(dates, cases):
    if not dates or not cases or len(dates) < 2:
        return None
    # Perform linear regression
    reg, intercept, r_value, p_value, std_err = linregress(dates, cases)
    return float(reg)

trendline_coef_udf = udf(trendline_coef, FloatType())
#load clean Data
data_path = "data"
df = load_csv_from_cleaned_folder(spark, data_path)
df.show(5)


#remove the / in the non date col
df = df.withColumnRenamed("Province/State", "Province_State").withColumnRenamed("Country/Region", "Country_Region")

#list of columns that are not dates
non_date_cols = ['Province_State', 'Country_Region', 'CountryCode', 'Continent', 'Lat', 'Long']

#list of date columns
date_cols = [col for col in df.columns if col not in non_date_cols]

#stack the date columns to long format
stack_expr = "stack(" + str(len(date_cols)) + ", " + \
    ", ".join(["'" + c + "', `" + c + "`" for c in date_cols]) + ") as (Date, Confirmed)"
df_pivot = df.selectExpr(*non_date_cols, stack_expr)

#convert Date column to a proper date format
df_pivot = df_pivot.withColumn("Date", to_date("Date", "MM/dd/yy"))

df_pivot.show(5)
```

```
#group by Province/State, Country/Region, and month, and calculate the trendline coef
df_grouped = df_pivot.withColumn('Month', month('Date'))\
                     .withColumn('Year', year('Date'))\
                     .withColumn('Day', dayofmonth('Date'))\
                     .groupBy('Province_State', 'Country_Region', 'CountryCode', 'Continent', 'Lat', 'Long', 'Year', 'Month')\
                     .agg(collect_list('Day').alias('Days'), collect_list('Confirmed').alias('Cases'))


df_trendline = df_grouped.withColumn('TrendlineCoef', trendline_coef_udf('Days', 'Cases'))

windowSpec = Window.partitionBy('Year', 'Month').orderBy(col('TrendlineCoef').desc())

#rank based on the trendline coef to get the top 50
df_ranked = df_trendline.withColumn('Rank', rank().over(windowSpec))
df_top50 = df_ranked.filter(col('Rank') <= 50)

df_top50.show(100)
```

```
#ensure trendline coef is the correct data type
df_top50 = df_top50.withColumn("TrendlineCoef", df_top50["TrendlineCoef"].cast("float"))

#assemble features into a vector
vecAssembler = VectorAssembler(inputCols=["TrendlineCoef"], outputCol="features")
df_features = vecAssembler.transform(df_top50)

#create a UDF to assign clusters
kmeans = KMeans(k=4, seed=1, featuresCol="features", predictionCol="Cluster")


#compute
df_with_clusters = get_month_clust(df_features)

df_with_clusters.show()
```

```
#select the wanted columns
df_with_clusters = df_with_clusters.select("Province_State", "Country_Region", "Month", "TrendlineCoef", "Cluster")
df_with_clusters.show(1000)


#Save the result
rm_dir("query_results", "*_query3")
#write result_df to csv
today = datetime.date.today().strftime("%Y%m%d")
file_path = os.path.join("query_results", f"{today}_query3")
df_with_clusters.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').option("header", "true").save(file_path)

print(f"Saved : {file_path}")
```