Cranfield University

Carried out by:

YASSER El KARKOURI

ID : **S450915**

Machine learning & Big Data Assignment

Analysis of the COVID-19 global dataset using **Hadoop/Spark**

Module supervisor teacher:

Dr Stuart Barnes

Cranfield University

Table of Contents

Illustrations

I. Abstract :

In the current era of data-driven decision-making, the importance of big data, data analysis, and machine learning cannot be overstated. The data analysis process of a large set of data presents several challenges that need to be addressed to ensure the effectiveness . Finding an efficient and sustainable process is the key to handle that big volume . Big Data often comes in multiples formats , including unstructured data , what rases the challenge of data cleansing . In the same context , our assignment would be the best opportunity to get familiar with how to handle those challenges since we'll be working on a huge set of data representing the worldwide trend of confirmed cases of Covid-19.

II. Introduction :

The exponential explosion of data nowadays has ensured that data analysis and statistical modeling are now central to solving hard problems in this current age of data-driven decision making. This is exemplified by our growing digital footprint which has seen that data rapidly increase, both volumously and diversely. Big Data is normally characterized by datasets that are so massive to be managed by the standard hardware in an effective manner. Consequently, traditional data processing tools often do not manage such big datasets with appropriate performance quite easily. [1]

Our Machine Learning & Big Data course broke down the complexities of Big Data and provided solutions that it may requires. A significant emphasis is placed on Apache Spark with PySpark, an advanced tool that enhances the Hadoop MapReduce model. By simplifying intricate computations across numerous nodes, this technology facilitates the analysis of large-scale data, making it more accessible and efficient. Through automated data partitioning and distribution, users can now direct their attention towards analysis, significantly reducing the complexities associated with data management. [2]

In this project, we harness the power of PySpark ,which is the Python API for Apache Spark, a unified analytics engine for large-scale data processing , to delve deep into the comprehensive global COVID-19 dataset. Our primary goal is to uncover invaluable insights, identify emerging trends, and unearth hidden patterns. These analytical endeavours hold significant academic value and possess the potential to shape decision-making processes in regions that have been severely impacted by the ongoing pandemic. By leveraging PySpark, we can ensure a meticulous and efficient analysis that drives impactful results. [3]

III. Objectives:

This report offers a detailed examination of the global impact of the COVID-19 pandemic, utilizing the advanced data processing tool **Apache Spark**. Our analysis is based on a comprehensive **CSV** file, which is supposed to be updated at **23:59 UTC** each day but in our case the data frame stopped being updated in **09/03/2023** , providing in-depth insights into the progression of confirmed **COVID-19** cases worldwide. By leveraging these powerful frameworks, we aim to deliver accurate and up-to-date information on the global pandemic, helping individuals from all walks of life stay informed and make informed decisions. [4]

We are going to conduct various analysis on different levels :

- **Country-Level Analysis:** For each country, we aim to calculate the average number of confirmed **COVID-19** cases daily for each month throughout the dataset.

- **Continent-Level Analysis:** Our goal here is to assess the pandemic's impact on each continent. We will calculate key statistical measures - mean, standard deviation, minimum, and maximum - of daily confirmed cases for each week.

- **Cluster Analysis Using K-means:** Focusing on the **50 states** most affected by COVID-19 monthly, we will apply the **K-means** clustering algorithm with **K=4**. [4]

IV. Methodology:

Before digging deeper in the tools , software and libraries we used in our analyse we will start first by describing the dataset importation .It was mentioned that the dataset is normally getting updated at **23:59 UTC** each day since data frame stopped being updated in **09/03/2023** we decided to export the dataset in **csv** format from the provided link once and then implement our analysis approach based on that **csv** , this decision was made for performances purpose because there is no need to pull the data from the URL - raw data - every day if there is no changes made .

1. Importation process(overview) :

The data analysis project embarked upon a comprehensive workflow, starting with the extraction of raw data in **CSV** format from a designated **URL**. **Jupyter Notebook**, a primary framework, was harnessed in conjunction with **Python** as the programming language and **PySpark** as the essential analytical tool. To ensure efficient data processing, a suite of libraries including P**ySpark**, **Pandas**, **Datetime**, **Numpy**, and **Time** were utilized, with an additional installation of **Pycountry**. For visualization purposes, a list of libraries such as **Geopandas**, **Matplotlib**, **Seaborn**, **Datetime**, **Geoplot**, and **Plotly** were employed to address the project's visualization requirements. The culmination of this meticulous process involved exporting the analysed data back into a **CSV** format, effectively completing the analysis cycle. Employing this approach, we combine robust data processing with versatile visualization techniques, facilitated a thorough and efficient analysis of the dataset. We can resume this process in the diagram in (***Figure 1***).

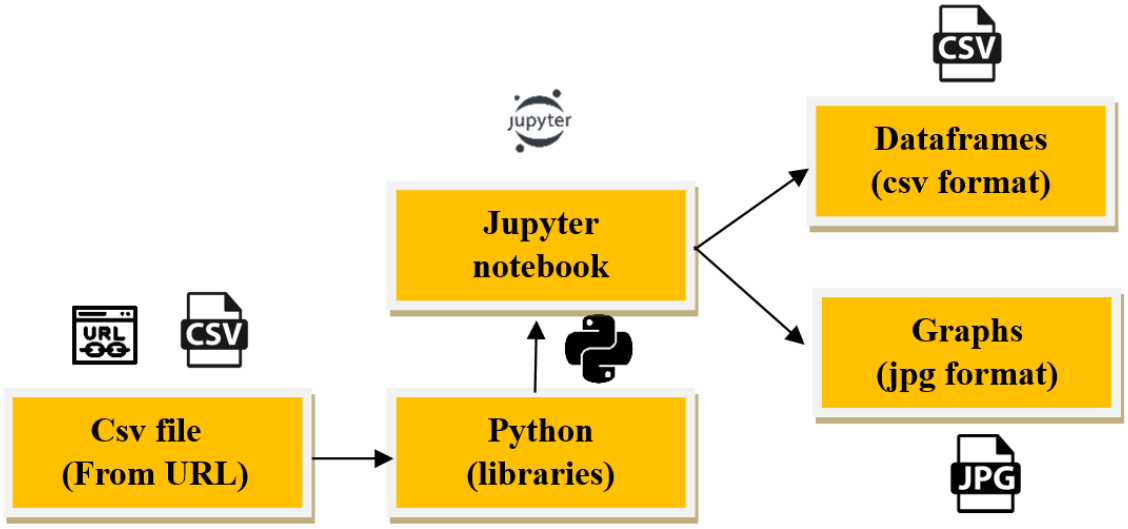


Figure 1:*Process Architecture*

2. Data cleansing (overview) :

The dataset was imported in **October 25** we will use a Pyspark data frame to identify and rectify errors, duplicates, and irrelevant data . We will make good use of the aggregation function in PySpark to do operations on rows and columns. After data cleansing, we export the cleaned data in csv file to further use .

3. Analysis Process:

3.1. Task 1 :

After data cleansing, we were obliged to change the data from a wide format to a long format to facilitate the operations on columns using the aggregation function on pyspark and also for performance purposes since we should avoid loops , which was also the reason we will use the **SQL** function **stack()** to pivot the table instead of loops and then we are going to use .**selectExpr()** function from pyspark along with it .

To calculate the mean number of confirmed cases daily for each month in the dataset we were facing a challenge because the data is cumulative. So, to perform the calculation we had to

- **Determine Monthly Change in Cases:** (Last Month Day cases – First day cases)
- **Calculate the number of days in a month.**
- **Computing the average number of daily cases in that month:**

$$\text{Average Daily Cases} = \frac{\text{Total Monthly Change in Cases}}{\text{Number of Days in Month}}$$

3.2. Task 2 :

After rotating the dataframe from a wide to a long format , we will perform a window partition using pyspark **. Window** functions are used to define a window, and then a separate function or set of functions is selected to operate within that window. They are particularly useful for examining relationships within groups of data and are more beneficial than using **groupBy()** for this purpose.

In this task window functions will be used to calculate the Trendline coefficients . We have chosen to perform a slope calculation using window function and avoid using the predefined function **slope** to optimize time execution of the program .[5]

The typical equation of a line is **y=mx+b** , where **m** is the slope and b are the y-intercept. Slope Calculation: In the context of **linear regression**, the slope (**m**) is calculated to understand how much the dependent variable in our case ("Daily_Cases") changes for a one-unit change in the independent variable for our case ("Day_Index").[6]

The slope calculation will be using the next equation , this formula is derived from the least squares method, which minimizes the sum of the squares of the differences between the observed and estimated values:

$$\text{Slope} = \frac{n \cdot \text{sum\_xy} - \text{sum\_x} \cdot \text{sum\_y}}{n \cdot \text{sum\_x2} - (\text{sum\_x})^2}$$

n: Count of days (or data points).

sum_xy: Sum of the product of "**Day_Index**" and "**Daily_Cases**".

sum_x: Sum of "**Day_Index**".

sum_y: Sum of "**Daily_Cases**".

sum_x2: Sum of the square of "**Day_Index**".

3.3. Task 3 :

In this task we will perform an other method to calculate the trendline coefficients .We are using the linear regression library to define a function that returns the linear coefficients .Then

we will use a lambda function to to apply the custom function to each group and then filter the top 50 ranked states .

In this task we are also going to use a KMeans clustering algorithm which s an unsupervised machine learning algorithm used for clustering data points into a predetermined number of clusters, 4 in our case. The goal is to partition the states into clusters in which each data point belongs to the cluster with the nearest mean.[7]

The mathematical algorithm followed to perform the data clustering is the following :

- **Distance Measure:** The most common distance measure used in **KMeans** is the Euclidean distance, calculated as: $D(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ where **x** and **y** are two points in an n-dimensional space.[8]
- **Centroid Calculation:** The centroid of a cluster is the arithmetic mean of all the points belonging to the cluster. If **C** is a set of points in a cluster. .[8] the centroid **c** is given by: $c = \frac{1}{|C|}\sum_{x \in C} x$
- **Objective Function:** KMeans aims to minimize the within-cluster variance. The objective function, often called the 'inertia', is the sum of squared distances between each point and its centroid**:** $J = \sum_{i=1}^{k}\sum_{x \in C_i} ||x - \mu_i||^2$ where **μi** is the centroid of cluster **Ci.** .[8]

3.4. Visualization process:

We will use variety of graphs and diagrams to Identify the patterns and to extract valuable insights about each query .This process will help us in our analyse what will result in better understanding and interpretation of the results .

V. Covid-19 data analysis:

1. Data Importation :

The dataset of covid-19 was downloaded from the link on October 25 in csv format and then it was saved in a folder “data” .As we have already mentioned the code editor, we have are using is Jupyter Notebook for several reasons including the possibility to run segments of code independently .

Before further analysis we should make sure first that all the libraries we will use are already installed. (***Figure 2***).

```
# ! pip install pycountry pandas
# ! pip install plotly
# ! pip install pyspark
# ! pip install seaborn
# ! pip install datetime
```

Figure 2:Package Installation

Once we have finished the installation, we should start a pyspark session because it will enable us to perform large scale data processing using various features such as Spark SQL, Dataframes (***Figure 3***). [9]

```
ss = SparkSession.builder\
    .master("local")\
      .appName("test")\
       .config("spark.sql.legacy.timeParserPolicy", "LEGACY")\
          .getOrCreate()
```

Figure 3: Pyspark session

Then we import our dataset in a spark dataframe to get the next scheme structure **(*Figure 4*).**

```
root
 |-- Province/State: string (nullable = true)
 |-- Country/Region: string (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- 1/22/20: integer (nullable = true)
...
 |-- 3/7/23: integer (nullable = true)
 |-- 3/8/23: integer (nullable = true)
 |-- 3/9/23: integer (nullable = true)
```

Figure 4:dataset scheme

Our dataset is composed of **289** rows and **1147** columns the first 4 are for "**Province/State**" , "**Country/Region**" , "**L**at" and "**Long**" and the other columns are dates from **1/22/20** to **3/9/23.**

2. Data Cleansing and preprocessing :

To conduct a successful data analysis procedure data preprocessing is a crucial stage because most of raw data usually contains noise, mistakes, empty rows , invalid elements and so on .

2.1. Drop duplicates :

First thing we have noticed is that the table may contain some duplicates this is why as a precautionary step we will `.dropDuplicates()` to drop duplicates.

2.2. Restructure dataframe :

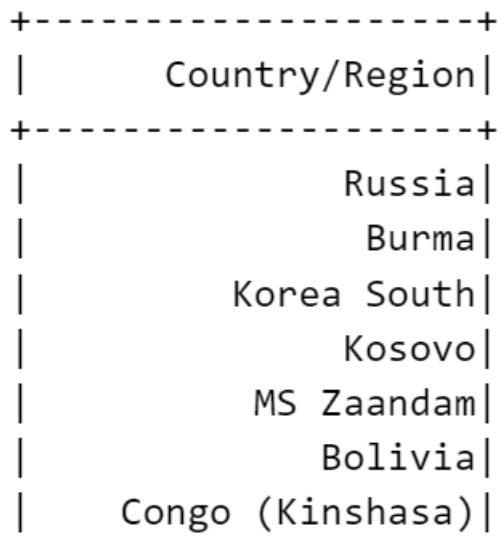
We have noticed that the date format in the original dataset is sometimes “MM/DD/YY” and others "MM/DD/YYYY", we need to convert all the dates to the same format "MM/DD/YY".

2.3. Filter nan and empty values :

We have noticed that the dataset includes some nan and empty values, so we have decided to replace nan values in the column "Province/State" with “no state” and drop the rows of the nan values in the columns “Lat” , “Long” .

2.4. Filter mismatched data:

The dataset contains some elements which is not established to “Country/Region” , this is why we have generated a csv file which has all the countries of the world based on the Pycountry library . Then we will import this csv file in our pyspark session as a new dataframe called “Countries” , then we will use a “Left Anti” join filter none country values in our dataset **(*Figure 5*).**



Country/Region
Russia
Burma
Korea South
Kosovo
MS Zaandam
Bolivia
Congo (Kinshasa)

Figure 5 : mismatched values

As you may notice some of the values represents real countries but they are not written the good way so we will try to correct the spelling of words that are not spelled correctly .For this process we use a dictionary that contains the mismatched countries and their correct names .And then we will replace the mismatched countries with the correct ones **(*Figure 6*).**

```
# Now we will replace the mismatched countries with the correct ones
df = df.replace(Corrected_country, subset=["Country/Region"])
```

Figure 6:Correct misspelled countries

After correcting the countries spelling, we filter the noncountries values. We get a cleaned dataframe ***.(see Figure 7)***

Province/State	Country/Region	lat	Long	1/22/20	1/23/20	1/24/20	1/25/20
no state	Libya	26.3351	17.228331	0	0	0	0
no state	Guinea	9.9456	-9.6966	0	0	0	0
no state	Saint Lucia	13.9094	-60.9789	0	0	0	0
Guernsey	United Kingdom	49.448196	-2.58949	0	0	0	0
South Australia	Australia	-34.9285	138.6007	0	0	0	0
no state	Grenada	12.1165	-61.679	0	0	0	0
no state	Kazakhstan	48.0196	66.9237	0	0	0	0
no state	Philippines	12.879721	121.774017	0	0	0	0
Prince Edward Island	Canada	46.5107	-63.4168	0	0	0	0
Saint Pierre and ...	France	46.8852	-56.3159	0	0	0	0
no state	Guyana	4.860416	-58.93018	0	0	0	0
Northern Territory	Australia	-12.4634	130.8456	0	0	0	0

Figure 7: Cleaned dataframe

➡ After the data processing and cleansing we finally export data in a csv file **'data/Cleaned_df.csv'** for further analysis .We have used this cleaned data frame and we have used the library **Geoplot** to plot a map and then we use we Plot the points based on latitude and longitude of the countries to get the graph in (***Figure 8).***

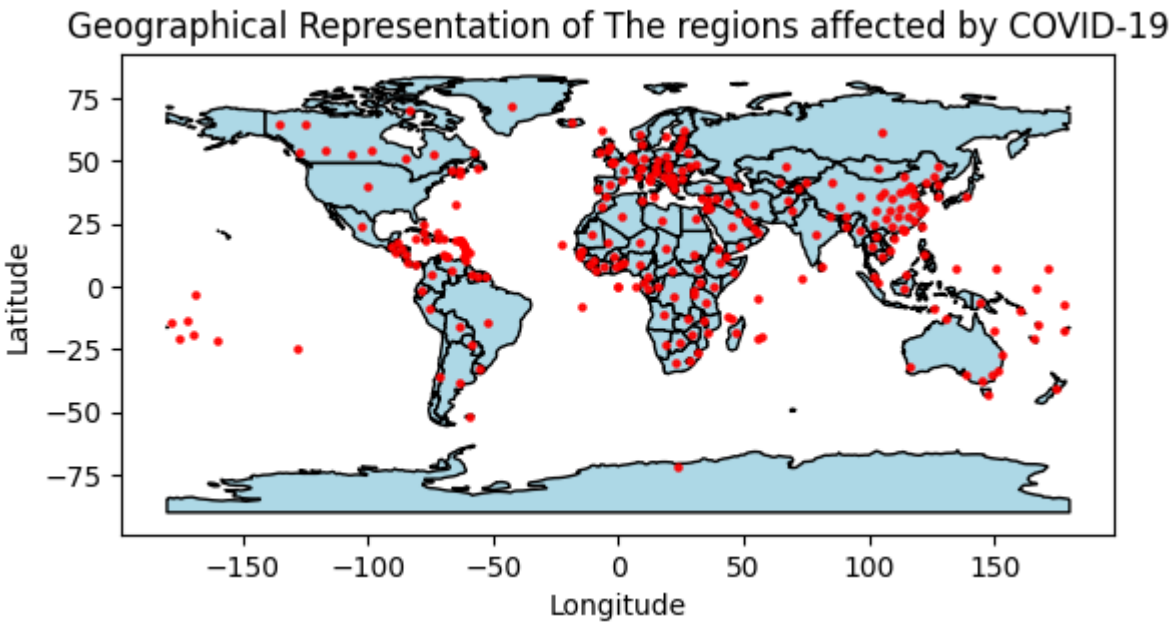


Figure 8:Countries Affected By Covid-19

VI. Results and interpretations:

All the results obtained from the queries are provided in CSV format in the folder "data" in the materials submitted along with the program .

1. Task 1 :

The first query is a **Country-Level Analysis** in which we aim to calculate the average number of confirmed **COVID-19** cases daily for each month throughout the dataset , for each country .

After the different coding phases , we get the "**Task1_df**" dataframe which represents the average number of confirmed **COVID-19** cases daily for each month , For all the countries in the dataset from the month **January 2020** to **Mars 2023** .(***see Figure 9)***

Country/Region	2020-01	2020-02	2020-03	2020-04
Afghanistan	0.0	0.1724137931034483	5.193548387096774	54.5
Albania	0.0	0.0	7.838709677419355	17.133333333333333
Algeria	0.0	0.034482758620689655	23.06451612903226	105.3
Andorra	0.0	0.0	12.129032258064516	11.833333333333334
Angola	0.0	0.0	0.22580645161290322	0.6333333333333333
Antarctica	0.0	0.0	0.0	0.0
Antigua and Barbuda	0.0	0.0	0.22580645161290322	0.5666666666666667
Argentina	0.0	0.0	34.0	112.46666666666667
Armenia	0.0	0.0	17.129032258064516	49.833333333333336
Australia	0.2903225806451613	0.4482758620689655	146.19354838709677	63.46666666666667

Figure 9 : Monthly average cases for each country

To have a better understanding of our results e will try to define a function to plot a composite bar chart that evaluate daily average of **COVID-19** cases each month over time for a chosen list of countries. The **Figure 10** is an exemple for selection of countries ["**Morocco**", "**China**", "**Australia**", "**United States of America**", "**Brazil**", "**Russian Federation**"] over the period (**2020-01**) - (**2023-03**).

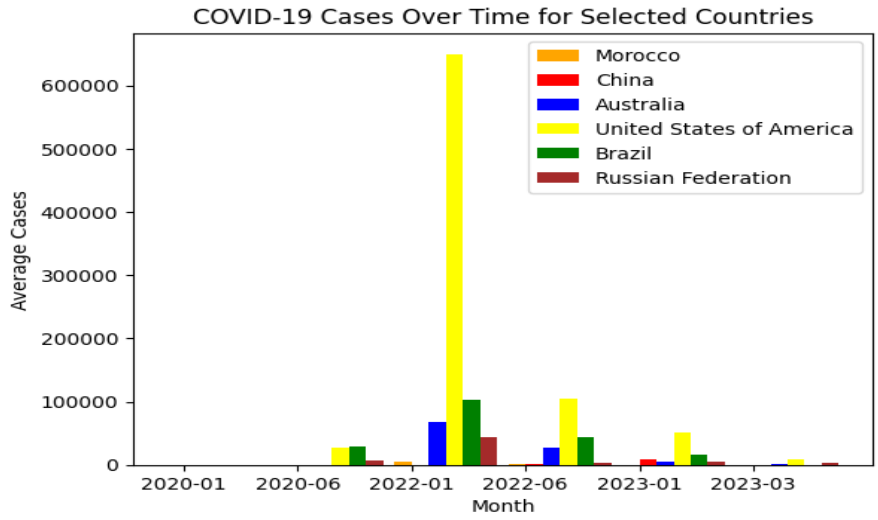


Figure 10:Covid-19 Cases over time

- The graph shows that for most countries, there's a significant peak in the number of cases around January **2022**. This could indicate a global surge in cases, possibly due to a new variant, less adherence to safety measures, increased testing, or a combination of factors.
- **Morocco** is having good consistency in low covid cases average number and on the other side The **US** are the most affected by the pandemic .

- By **March 2023**, all countries show a reduction in daily cases, suggesting control over the spread of the virus, possibly due to vaccination campaigns, natural immunity, or sustained public health measures.

We have noticed that this data would be more informative with a more explicit graph ,this why we have decided to plot an individual separate bar graph for each of the countries because the scale of the cases is very different.(***see Figure 11***)

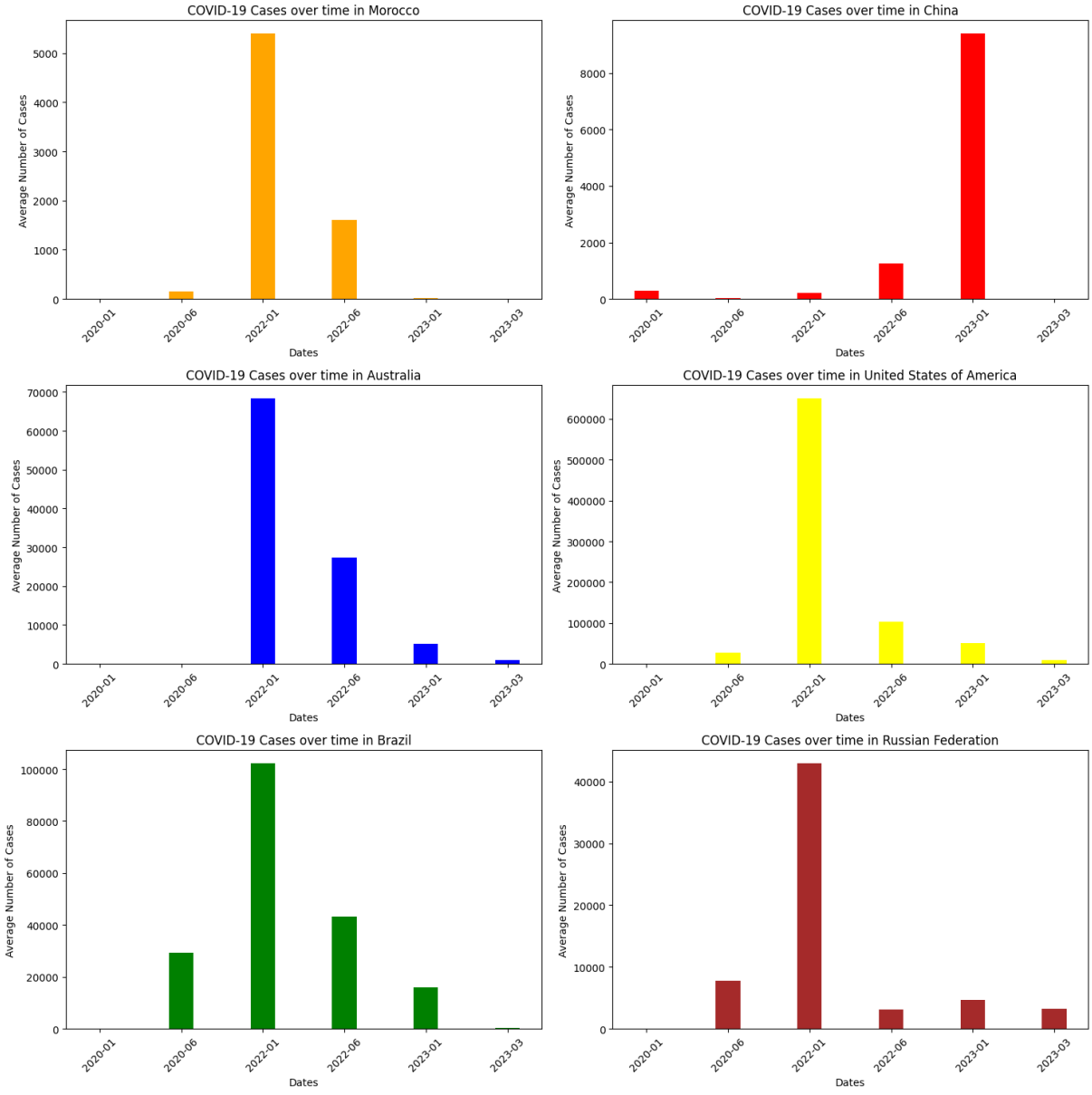


Figure 11: Seperated Bar Graphs to evaluate Covid-19 cases over time

These graphs provide detailed information on the fluctuations of COVID-19 cases in each country on specific dates. This level of detail is particularly useful for analysing the response to outbreaks and the impact of interventions on a country-by-country basis.

2. Task 2 :

The second query is a **Continent-Level Analysis** in which we aim to calculate the mean, standard deviation, minimum and maximum of the number of confirmed cases daily for each week.

After ranking the most 100 states affected by the pandemic based on the **trendline coefficients**, we then group them by continent to get the table that we export as a csv file "**Task2-df.csv**" and we save in data folder along with the code source ***(Figure 12).***

Continent	Year	Week	Mean_Cases	Stddev_Cases	Min_Cases	Max_Cases
Asia	2020	4	2.1630434782608696	5.857796859062707	0	33
Asia	2020	5	5.366459627329193	16.42394952538586	0	99
Asia	2020	6	5.173913043478261	15.535268430812236	0	93
Asia	2020	7	2.4161490683229814	6.053469053166105	0	42
Asia	2020	8	4.6894409937888195	24.30900965301875	0	229
Asia	2020	9	21.024844720496894	102.22572757814928	0	813
Asia	2020	10	25.565217391304348	109.76820934409338	0	851
Asia	2020	11	14.8125	33.058778889051844	0	242
Asia	2020	12	39.962732919254655	67.39156922408377	0	566
Asia	2020	13	117.75155279503106	290.6761735818722	0	2069
Asia	2020	14	226.72670807453417	548.5403014772207	0	3135
Asia	2020	15	335.35403726708074	912.7365885753804	0	5138
Asia	2020	16	452.63125	1152.7440470507781	0	6060

Figure 12: Mean, Standard deviation, Minimum and Maximum of continent Cases

For better understanding and interpretation, we will start by visualizing the trend of 4 of the top 100 state ranked by their **Trendline coefficients** in **line graphs.** ***(Figure 13)***

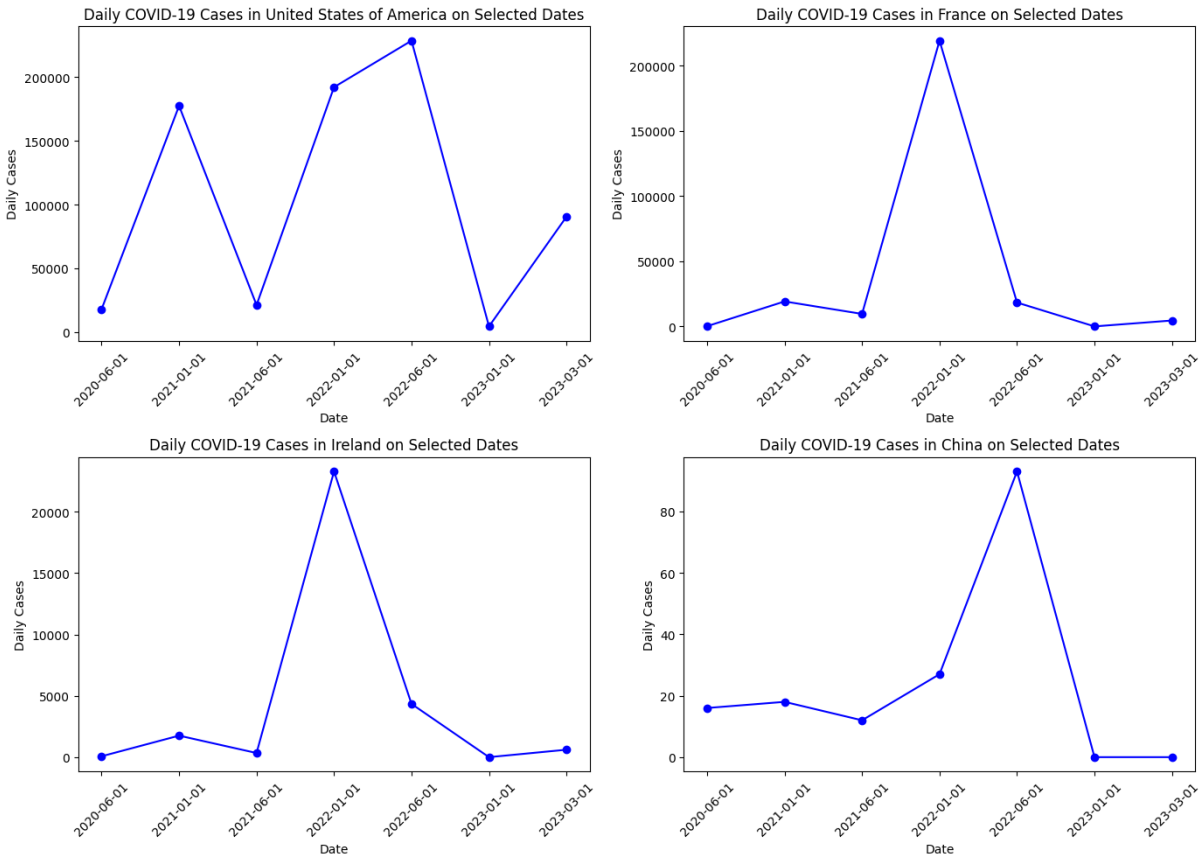


Figure 13:Line Graphs Of The Daily COVID-19 in 4 of the top 100 countries

- There is a common pattern of a significant surge in cases around **January 2021** and **January 2022** across all countries.
- The **United States** and **France** show the highest case counts among the four countries, with sharp peaks indicating intense waves of infection.
- By **March 2023**, all countries show a reduction in daily cases.

For further understanding we will evaluate the mean cases trend in a stacked area chart that shows the trend of mean **COVID-19** cases broken down by continent. Each line represents a different continent. ***(Figure 14)***

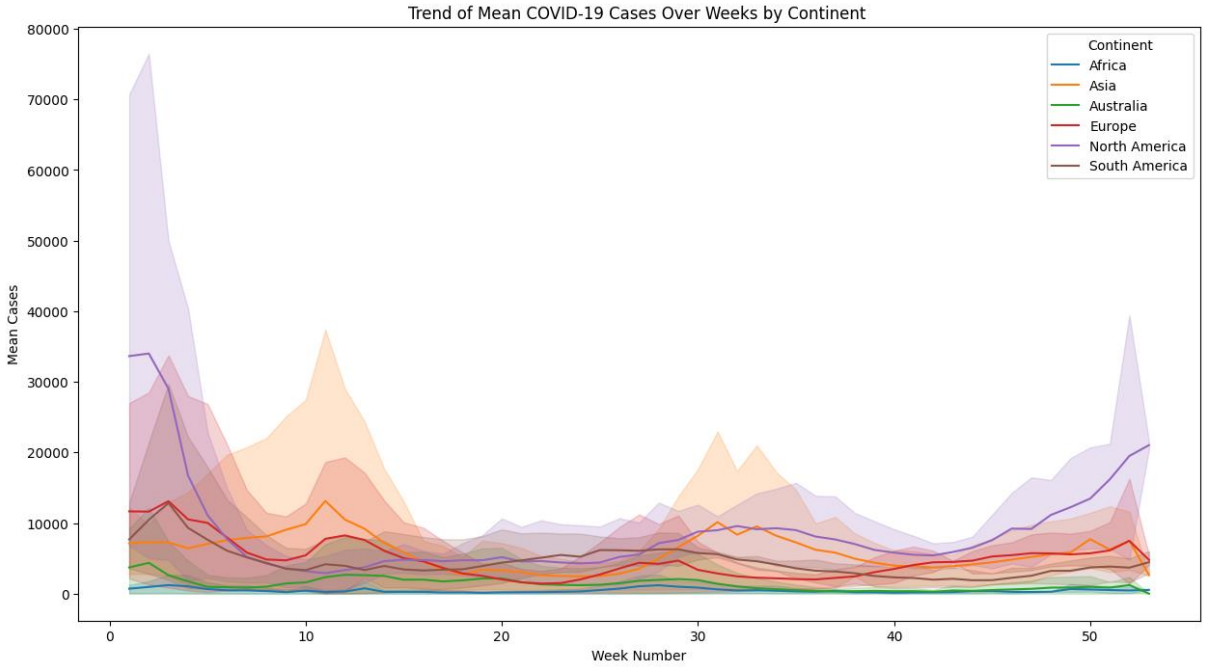


Figure 14:Stacked Area of mean COVID-19 Cases Over Weeks by continent

- **Europe** and **North America** are the continents with the highest mean cases throughout the period, suggesting a larger impact of the virus or more reporting of cases.
- There is a general upward trend in the number of cases towards the end of the period for several continents, possibly indicating a new wave or variant.
- The graph highlights the differences in **COVID-19** impact across continents. Some continents, like **Africa** and **Australia**, have consistently lower mean cases, while others like **Europe** and **North America** experience higher mean cases and more pronounced waves.

This graph is useful for observing the dynamics of the pandemic across continents over time and can help us understand the temporal patterns of the virus's spread. However, further context is needed to draw more detailed conclusions from this data. This is why we have plotted a scatter plot and a graph bar that shows the relationship between the mean number of **COVID-19** cases and the standard deviation of cases by continent. ***(Figure 15,16)***

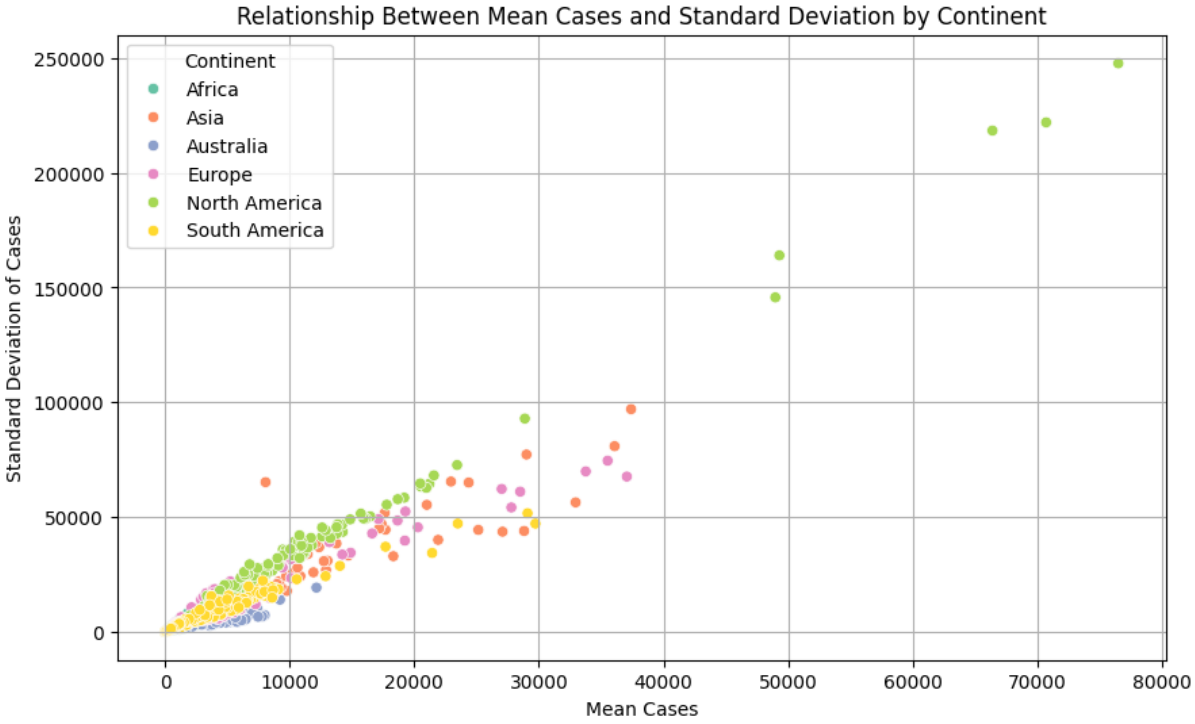


Figure 15: Link between standard deviation and mean Covid-19 by continent

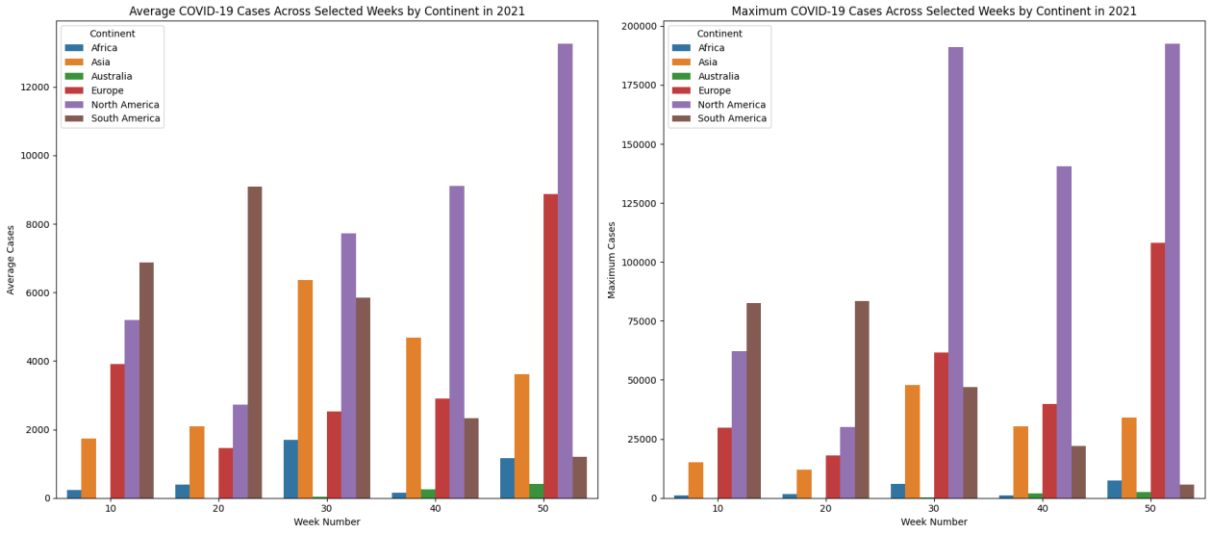


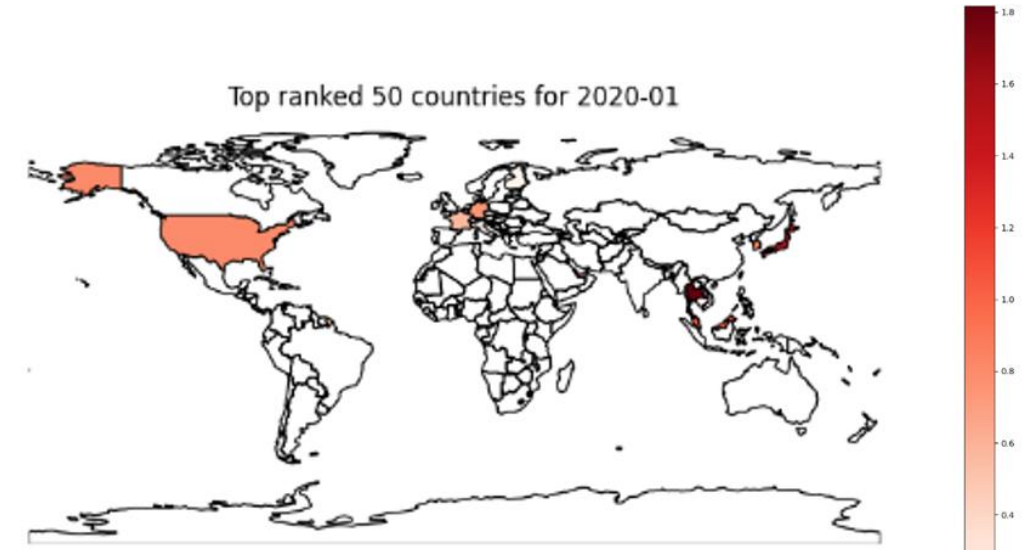
Figure 16 : Mean and Max Cases Bar Graphs weekly by continent

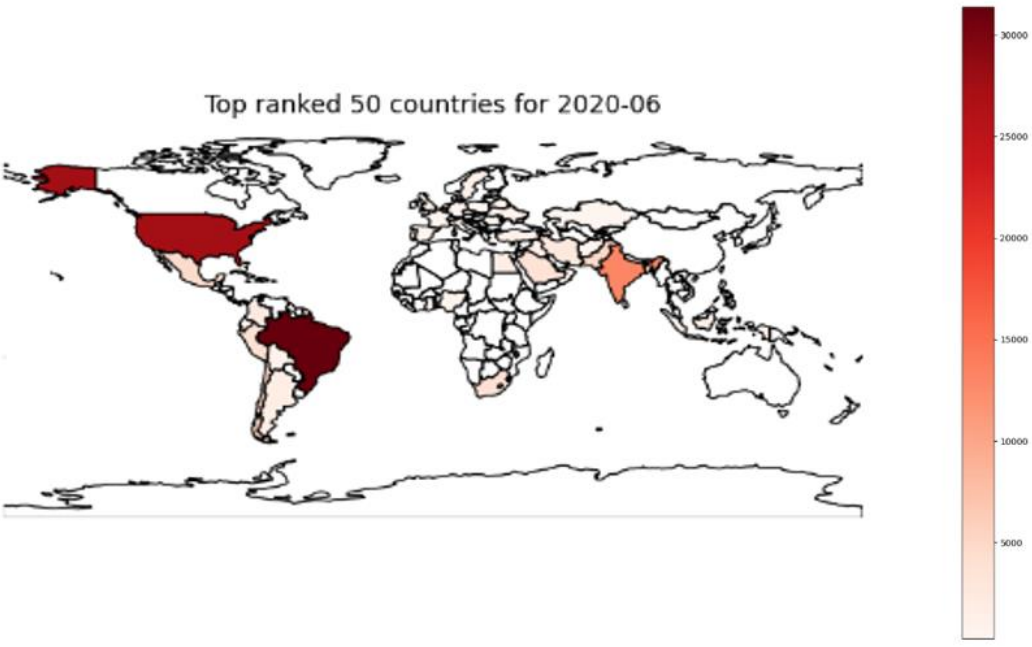
- There's a visible positive correlation between the mean number of cases and the standard deviation on the graph. This suggests that as the average number of cases increases, the variability of the cases increases. This is expected, as a higher number of cases typically leads to greater variance.

3. Task 3:

For the **Task 3** we will do conduct a **Cluster Analysis Using K-means algorithm** Focusing on the **50 states** most affected by **COVID**-19 monthly, we will apply the process with **K=4**.

After the Processing we obtain the dataframe that list the 50 states the most affected by the pandemic for each month based on the scope coefficient . We will give an example of the top 50 countries affected in the months ['**2020-01**', '**2020-06', '2021-01', '2021-06', '2022-01', '2022-06', '2023-01', '2023-03**'] and to visualize it we are going to use a **geographical heatmap *(Figure 17)*.**





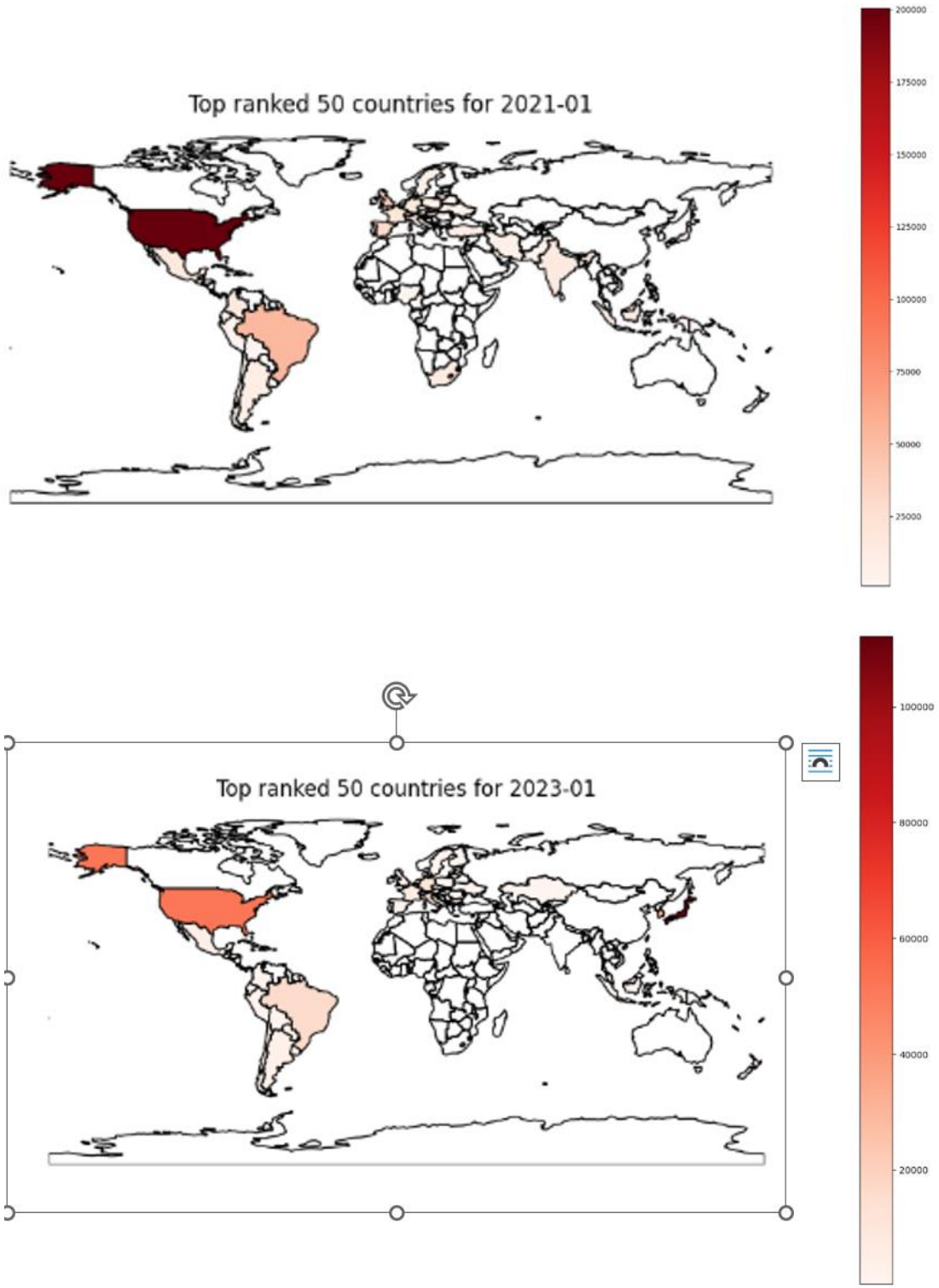


Figure 17: geographical heatmap

Once we have filtered the top 50 States/Countries for each month we will perform the K-mean clustering algorithm for with K=4 .The result we get is the table exported in "**Task3_df.csv**". ***(Figure 18)***

State/Region	Year-Month	Trendline_Coefficient	prediction
Hubei	2020-1	709.8666666666664	0
United Arab Emirates	2020-1	0.5999999999999999	1
Gansu	2020-1	3.866666666666666	1
Shanxi	2020-1	5.233333333333332	1
Tianjin	2020-1	3.7499999999999996	1
Jilin	2020-1	1.6499999999999997	1
New South Wales	2020-1	0.6166666666666665	1
Japan	2020-1	1.5333333333333328	1
Shaanxi	2020-1	10.266666666666664	1
Singapore	2020-1	1.333333333333333	1
Guangxi	2020-1	9.633333333333331	1
Hong Kong	2020-1	1.2333333333333332	1
Hainan	2020-1	6.133333333333332	1
Qinghai	2020-1	1.183333333333333	1
Inner Mongolia	2020-1	2.666666666666666	1
Taiwan	2020-1	1.133333333333333	1
Guizhou	2020-1	2.416666666666666	1

Figure 18: clustering algorithm for with K=4

To visualize and interpret the results and the predictions we got we will plot a a **Scatter Heat Plot** to visualize the density of the countries around the clusters for the top 50 **countries/states** for each month. ***(Figure 19)***

Clustering: Each colour represents a cluster, which groups states or regions with similar trends. For example, points that are the same colour across different plots may indicate states or regions that have consistently similar trends over time.

Trendline Coefficient: On the x-axis, the trendline coefficient may represent the slope of the trendline fitted to the case data over time. A higher value could indicate an increasing trend in cases, while a lower or negative value could indicate a stable or decreasing trend.

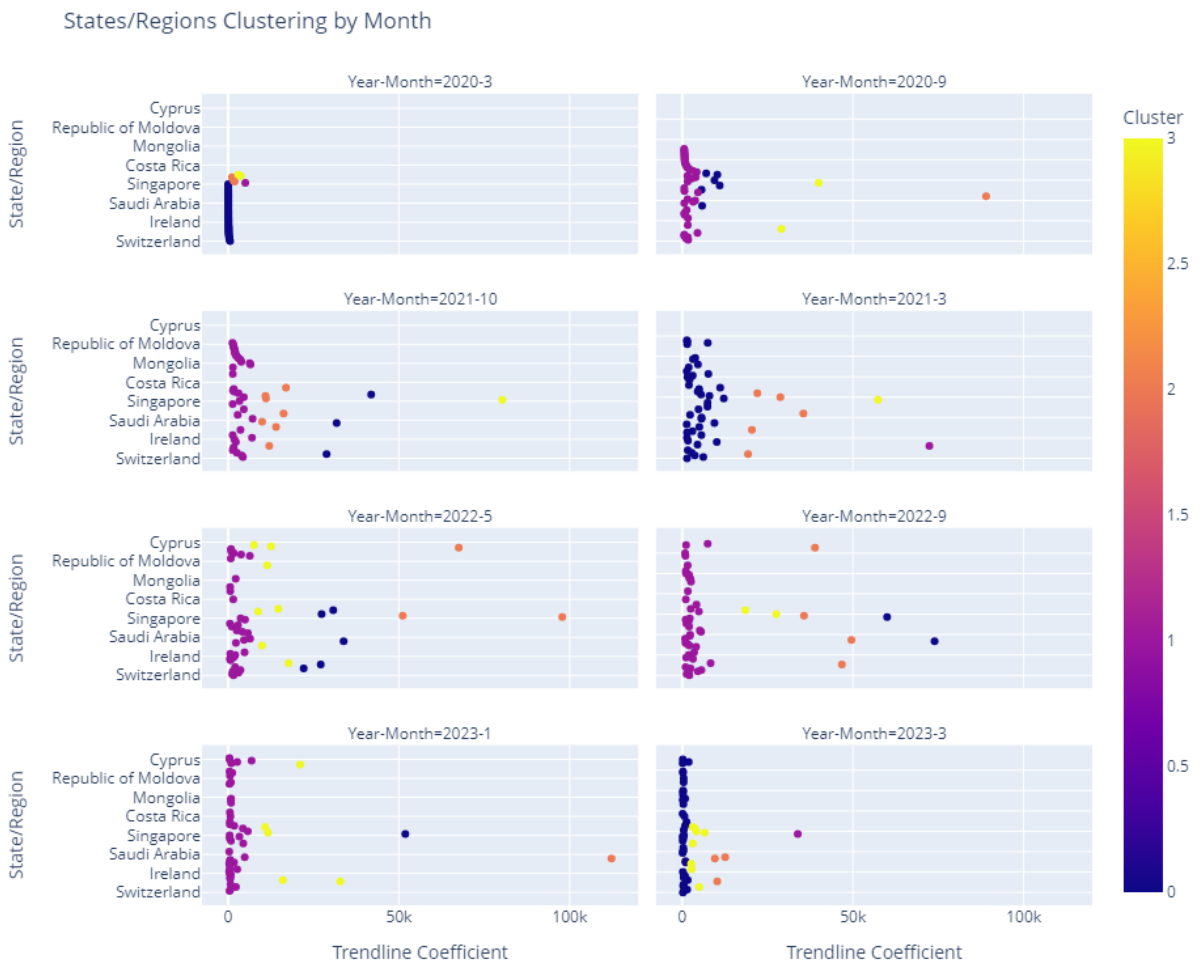


Figure 19: Scatter Heat Plot showing density of countries around the clusters.

From **March 2020** to **March 2023**, the trajectory of **COVID-19** cases in various regions has displayed significant variations. Initially, the trendline coefficients revealed a mix of rapidly increasing trends, which eventually transitioned into a general stabilization or slower growth by **September 2020**. As we progressed into **March 2021**, the evolving trends highlighted distinct regional responses, with some areas effectively controlling the cases while others faced the potential for surges. However, **October 2021** witnessed a notable decrease or stabilization in cases, offering a glimmer of hope. Unfortunately, a cluster with higher coefficients emerged in **May 2022**, signalling the potential for another wave of infections. Despite this, by **September 2022**, most regions displayed low coefficients, indicating an overall decline or stabilization in cases, while a few outliers faced challenges. This encouraging trend continued into **January 2023**, with a general stabilization observed across

regions. However, **by March 2023**, a minority of regions experienced an uptick in case trends, denoted by higher trendline coefficients. The dynamic nature of **COVID-19** cases emphasizes the importance of remaining vigilant and adhering to appropriate measures to combat the virus effectively.

VII. Efficiency analysis :

One of the crucial elements that a data analyst should be aware of and take in count while working on big data and large-scale datasets since it can significantly improve the performance of data processing and analysis tasks and minimize the resources (Ram memory for example) .

I personally had some issues when I wanted to apply the usual processing operations such us “loops / conditions …” on our dataset , because the program used to take a long time to go out of the loop and sometimes it runs out of memory .

I tried to use Docker since it already has images where the API’s and the environment is already set but I was obliged not to work with it cause on a windows laptop running a Linux virtual machine allocate an amount of memory which reached 2GB RAM in my case, so it was not optimal .

I made a program that calculates the time each block of code takes and we have done a comparison that you may see in **Figure 20** .

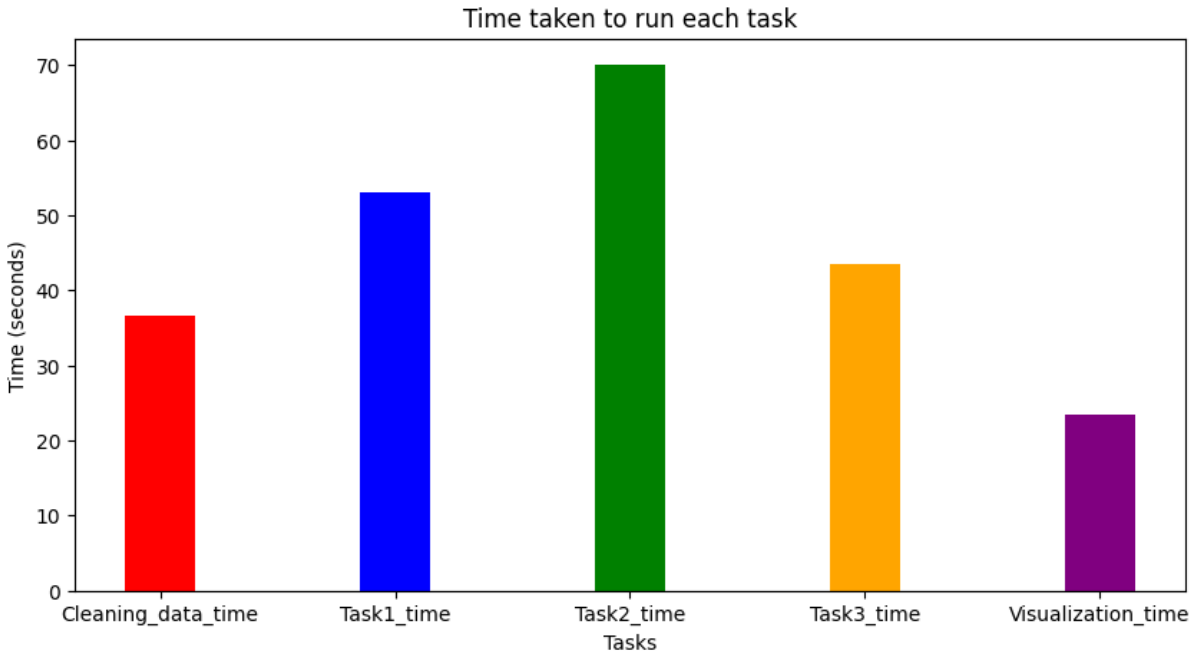


Figure 20: Time performance for each task

Another conclusion to be made is that the tasks that uses pandas library to manage Dataframes or for the visualization process seems to be faster than the tasks handled in pyspark and that maybe because of the scale of our data (sometimes spark functions loses more time in data distribution than it will gain in the process)

VIII. Conclusion :

This project was an exceptional opportunity for me to learn how data analysis, big data and machine learning can be very useful during large scale crisis such us COVID-19 pandemic. It provides a futuristic insight that can help us understand the pattern of the virus spraying that we can anticipate precautions.

These analytics have enabled the modelling of infection trends throughout different graphs and diagrams that may help with the optimization of resource allocation and facilitates the development of health applications. It is very important for us to go forward in building new technologies that ensures the capability to handle various public health scenarios.

This project played a key role in the practical application of skills essential to data science and analytics. By conducting in-depth analysis of COVID-19 trends using advanced tools such as machine learning and big data, I improved my skills in data processing, model selection and interpretation of complex datasets. Additionally, it strengthened my understanding of the ethical aspects of data science, especially when dealing with sensitive health data.

IX. Ethical Issues and Challenges :

The COVID-19 crisis has shed light on several ethical concerns surrounding the use of Machine Learning (ML) and Big Data. As organizations gather and analyse large amounts of personal health data, privacy becomes a crucial issue. The varying levels of security measures and privacy policies across entities raise concerns about the protection of personal information. Additionally, ML algorithms can unintentionally exacerbate existing inequalities

by favouring individuals with more digital footprints or better access to healthcare resources, which leads to skewed distribution of resources. The complex nature of ML algorithms also poses challenges in terms of accountability, making it difficult to attribute responsibility for any mistakes or harmful outcomes. Furthermore, the usage of personal data for public health purposes raises questions about consent and the balance between individual autonomy and collective interest. To ensure the ethical deployment of these technologies, a multidisciplinary approach is necessary, encompassing strong legal frameworks, ethical guidelines, and public engagement. This approach aims to strike a fine balance between fostering innovation and safeguarding individual rights.

X. References :

[1] DiRenzo, V. (2022). Beginner's Guide to Machine Learning with Big Data. Towards Data Science.

[2] Gandomi, A.H., Chen, F., & Abualigah, L. (2022). Machine Learning Technologies for Big Data Analytics. MDPI.

[3] Databricks. "What is PySpark?" [Online]. Available: https://www.databricks.com/glossary/pyspark

[4] MLBD_ASSIGNEMENT 2023-2024 provided by Dr Jun Li

[5] **Sefidian, A. M. (2022, February 17). A guide on PySpark Window Functions with Partition By. Retrieved. Available on**: http://www.sefidian.com/2022/02/17/pyspark-window-functions/

[6] Investopedia. (2023, October 30). Least Squares Method: What It Means, How to Use It, With Examples. Retrieved from: https://www.investopedia.com/terms/l/least-squares-method.asp.

[7] **Stanford University. "K-Means Clustering algorithm - CS221." [Online]. Available:** https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

[8] Dabbura, I. (2018). "K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks.". Avalable: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

[9] PyPI. (2023). "pyspark · PyPI." Retrieved from: https://pypi.org/project/pyspark/.

XI. Appendix:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.functions import *
from pyspark.sql.window import Window
from pyspark.ml.regression import LinearRegression
from pyspark.sql.types import StructType, StructField, StringType, DoubleType,
IntegerType
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
import pandas as pd
import numpy as np
import time
import pycountry
from IPython.display import display, HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))



ss = SparkSession.builder\
     .master("local")\
        .appName("test")\
         .config("spark.sql.legacy.timeParserPolicy", "LEGACY")\
            .getOrCreate()

Debute = time.time()
ss
"""
Since the date format in the original dataframe is sometimes;
"MM/DD/YY" and sometimes "MM/DD/YYYY", we need to convert all
the dates to the same format "MM/DD/YY"
"""

# Define the new column names and data types
new_column_names = [
    "Province/State",  # String
    "Country/Region",  # String
    "lat",             # Double
    "Long",            # Double
] + [col(date).cast(IntegerType()).alias(date) for date in df.columns[4:]]

# Rename the columns and cast data types
df = df.select(*new_column_names)
# Replace NaN values in "Province/State" with "no state"
df = df.na.fill("no state", subset=["Province/State"])
"""This block of code was meant to be used to create a csv file
```

```
        which has all the countries of the world based on the pycountry library

    # Generating a list of country names using pycountry library
        countries = [country.name for country in pycountry.countries]

    # Creating a DataFrame and saving it as a CSV file
         Countries = pd.DataFrame(countries, columns=["Countries"])

    Countries .to_csv('data/Countries.csv', index=False)"""

# Import the world map  Countries  into a DataFrame in pyspark
Countries = ss.read.csv('data/Countries.csv', header=True, inferSchema=True)

# Select distinct country names from your dataset
df_countries = df.select("Country/Region").distinct()

# Select distinct country names from the world map data
world_countries = Countries.select("Countries").distinct()


# we will use "Left anti" join to return rows from our DataFrame that doesn't
represent a country
Weird_countries = df_countries.join(world_countries,
df_countries["Country/Region"] == world_countries["Countries"], "left_anti")

# we will display the mismatched countries
Weird_countries.show()

# Now we are going to create a dictionary that contains the mismatched
countries and their correct names
Corrected_country= {
    'US': 'United States of America',
    'Burma': 'Myanmar',
    'Korea North': 'North Korea',
    'Korea South': 'South Korea',
    'Kosovo': 'Republic of Kosovo',
    'Congo(Brazzaville)': 'Republic of the Congo',
    'Congo(Kinshasa)': 'Democratic Republic of the Congo',
    'Laos': "Lao People's Democratic Republic",
    'Venezuela': 'Venezuela (Bolivarian Republic)',
    'Holy See': 'Holy See (Vatican City State)',
    'Taiwan*': 'Taiwan',
    "Cote d'Ivoire": "Côte d'Ivoire",
    'Syria': 'Syrian Arab Republic',
    'Brunei': 'Brunei Darussalam',
    'West Bank and Gaza':'Palestine',
    'Tanazania': 'United Republic of Tanzania',
    'Moldova': 'Republic of Moldova',
    'Vietnam': 'Viet Nam',
```

```
    'Russia': 'Russian Federation',
    'São Tomé and Príncipe': 'Sao Tome and Principe',

}


# Now we will replace the mismatched countries with the correct ones
df = df.replace(Corrected_country, subset=["Country/Region"])
""" df.filter((df["Country/Region"] == "United States of America") |
          (df["Country/Region"] == "Taiwan")|
          (df["Country/Region"] == "Sao Tome and Principe")|
          (df["Country/Region"] == "Diamond Princess")|
          (df["Country/Region"] == "Palestine")).show()"""

# Now we are going to create a list that contains the elements in the
"Country/Region" not corresponding to contries

None_contries = ['MS Zaandam', 'Diamond Princess', 'Summer Olympics 2020',
'Winter Olympics 2022']

df = df.filter(~df['Country/Region'].isin(None_contries))
df.show()
# df.write.csv('data/Cleaned_df.csv', header=True, mode='overwrite')

# Now we are going to create a list that contains the elements in the
"Country/Region" not corresponding to contries

None_contries = ['MS Zaandam', 'Diamond Princess', 'Summer Olympics 2020',
'Winter Olympics 2022']

df = df.filter(~df['Country/Region'].isin(None_contries))
df.show()
# df.write.csv('data/Cleaned_df.csv', header=True, mode='overwrite')

# We start by Grouping the data by "Country/Region"
group_df = df.groupBy("Country/Region")

# We store the date columns in a list
date_columns = df.columns[4:]

""" We iterate over the columns starting from the 5th column (index 4)
    and then we apply an aggregation function to each column that has
    the same "country/region" Value
"""
aggregated_columns = [sum(col(date)).alias(date) for date in date_columns]

# We Apply the aggregation to the grouped DataFrame
sum_df = group_df.agg(*aggregated_columns)
```

```
""" since the Dataframe sum_df has a wide format and this format is not
compatible
    with the opertions we want to perform on the data , we will need to
transform
    the DataFrame from wide to long format
"""

# We will extract the date columns from the DataFrame sum_df
date_columns = sum_df.columns[1:]

"""the following  code is constructing a dynamic stack expression that we will
use
   for pivoting date columns in sum-df into a date-wise row format, where
   each row represents a date and its associated case count.
"""

stack_df = "stack({0}, {1}) as (Date, Cases)".format(
    len(date_columns),', '.join(["'{0}', `{0}`".format(c) for c in
date_columns])
)

"""We will proceed to the final transpormation by using the function
selectExpr
    which will allow us to select the columns we want to keep -
"Country/Region"- and
    apply the stack expression to the date columns
 """
# we diddn't use the function select because we want to keep the column
"Country/Region"
long_df = sum_df.selectExpr("`Country/Region`", stack_df)

# We should Convert the 'Date' string we got from the stack function to an
actual date type
long_df = long_df.withColumn("Date", to_date(col("Date"), "MM/dd/yy"))

# We will add two new columns to the DataFrame long_df where to specify the
month
# and the last day of each month because we will need them later
long_df = long_df .withColumn("YearMonth", date_format(col("Date"), "yyyy-
MM")) \
         .withColumn("MonthLastDay", last_day(col("Date")))\
         .withColumn("MonthFirstDay", expr("to_date(concat(year(Date), '-',
month(Date), '-01'))"))

# We will use window functions to get the last day of each month
windowSpecLast = Window.partitionBy("Country/Region",
"YearMonth").orderBy(col("Date").desc())
last_day_cases = long_df.withColumn("rank", rank().over(windowSpecLast)) \
```

```
                    .filter(col("rank") == 1).select("Country/Region",
"YearMonth", col("Cases").alias("LastDayCases"), "MonthLastDay")


# We will use window functions to get the First day of each month
windowSpecFirst = Window.partitionBy("Country/Region",
"YearMonth").orderBy(col("Date"))
first_day_cases = long_df.withColumn("rank", rank().over(windowSpecFirst)) \
                    .filter(col("rank") == 1).select("Country/Region",
"YearMonth", col("Cases").alias("FirstDayCases"), "MonthFirstDay")
# Join the datasets
combined_cases = last_day_cases.join(first_day_cases, ["Country/Region",
"YearMonth"])
#combined_cases.show()
# we will Calculate the average cases per day for each month and store them in
monthly_data = combined_cases.withColumn("DaysInMonth",
dayofmonth(col("MonthLastDay"))) \
                             .select("Country/Region", "YearMonth",
((col("LastDayCases") - col("FirstDayCases")) /
col("DaysInMonth")).alias("AverageDailyCases"))

# print(monthly_data.show())
# Show the result
# Pivot the DataFrame to get one column for each month with the average cases
per day
Task1_df =
monthly_data.groupBy("`Country/Region`").pivot("YearMonth").agg(first("Average
DailyCases"))

Task1_df.show(10,truncate=False)
# Task1_df.write.csv('data/Task1_df.csv', header=True, mode='overwrite')
# calculate the time it took to run the code of Task1
Task1_time = time.time() - Cleaning_data_time- Debute
# print the duration
print("Task1_time: ", Task1_time , "seconds")
# Replace Unknown values with None
df2 = df.na.replace("Unknown", None)

# Selecting non-date columns and creating an array of date columns
Head = ['Province/State', 'Country/Region', 'lat', 'Long']
date_columns = [col for col in df2.columns if col not in Head]

# Creating an array of date columns and their corresponding values
array_cols = [struct(lit(c).alias("Date"), col(c).alias("Cases")) for c in
date_columns]
stacked_cols = explode(array(array_cols)).alias("stacked_cols")

# Transforming the DataFrame from wide to long format
long_df2 = df2.select(*Head, stacked_cols)\
```

```
            .select(*Head, "stacked_cols.Date", "stacked_cols.Cases")

""" Creating a new column 'State/Region' and filling it with the values of
'Province/State' column
 if the value is not 'no state' and with the values of 'Country/Region' column
otherwise"""

long_df2 = long_df2.withColumn('State/Region',
                               when(col('Province/State') != 'no state',
col('Province/State'))
                               .otherwise(col('Country/Region')))

# Selecting the required columns and Then converting 'Date' to a proper date
format
long_df2 = long_df2.select('State/Region', 'Date', 'Cases')
long_df2 = long_df2.withColumn('Date', to_date(col('Date'), 'MM/dd/yy'))

# Display and save
# long_df2.show(10, truncate=False)
# We will sort long_df2 first by the "State/Region" column and then by the
"Date" column
long_df2 = long_df2.orderBy("State/Region", "Date")

# we Define a window  partitioned by 'State/Region' and ordered by 'Date'
window = Window.partitionBy("State/Region").orderBy("Date")

"""We will add new column "Day_Index" to long_df2,
     representing the sequential day number within each "State/Region"
          starting from 0"""
long_df2 = long_df2.withColumn("Day_Index", row_number().over(window) - 1)

# we calulate the daily cases based on the fact that te data is cumulative
window_lag = window.rowsBetween(-1, 0)
long_df2 = long_df2.withColumn("Previous_Cases", lag("Cases").over(window))
long_df2 = long_df2.withColumn("Daily_Cases", col("Cases") -
col("Previous_Cases"))

# Filter out rows where Daily_Cases is negative
long_df2 = long_df2.filter(col("Daily_Cases") >= 0)


# Calculate Aggregations preparing  for Linear Regression
aggregates = long_df2.groupBy("State/Region").agg(
    count("Day_Index").alias("n"),
    sum(col("Day_Index") * col("Daily_Cases")).alias("sum_xy"),
    sum("Day_Index").alias("sum_x"),
    sum("Daily_Cases").alias("sum_y"),
    sum(col("Day_Index") * col("Day_Index")).alias("sum_x2")
)
```

```
"""we will calculate the slope of the linear regression line for each
"State/Region"
         using the formula for the slope in a linear regression."""

slope_df = aggregates.withColumn("Trendline Coefficient",
                                 (col("n") * col("sum_xy") - col("sum_x") *
col("sum_y")) /
                                 (col("n") * col("sum_x2") - col("sum_x") *
col("sum_x"))
                                )
# slope_df.show(10,truncate=False)

# we will sort the DataFrame by the trendline coefficient value in descending
order
ranked_df = slope_df.orderBy(col("Trendline Coefficient").desc())
#ranked_df.show()
#ranked_df.write.format("csv").option("header",
"true").mode("overwrite").save("ranked_df.csv")

# Selecting  the top 100 most affected states/regions.
Most_Affected_States = ranked_df.select("State/Region", "Trendline
Coefficient").orderBy(col("Trendline Coefficient").desc()).limit(100)

# Now perform the join
Joined_long_df = long_df2.join(Most_Affected_States, on="State/Region",
how="inner")
Joined_long_df = Joined_long_df.orderBy(col("Trendline Coefficient").desc())
# Joined_long_df.filter(col("State/Region") == "Queensland").show()
Joined_long_df = Joined_long_df.drop("Day_Index")

# Joined_long_df.show()

"""we will start by Defining  a mapping from "State/Region" to "Continent"
      using when().otherwise() statements for each continent"""
continent_mapping = (
    when(col("State/Region").isin(['Barbados','United States of
America','Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine',
'Maryland', 'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi',
'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire', 'New Jersey',
'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio',
'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina', 'South
Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
'West Virginia', 'Wisconsin', 'Wyoming',
```

```
                                    'Alberta', 'British Columbia',
'Manitoba', 'New Brunswick', 'Newfoundland and Labrador', 'Northwest
Territories', 'Nova Scotia', 'Nunavut', 'Ontario', 'Prince Edward Island',
'Quebec', 'Saskatchewan', 'Yukon']), 'North America'
    ).when(col("State/Region").isin(['New Zealand','Australian Capital
Territory', 'New South Wales', 'Northern Territory', 'Queensland', 'South
Australia', 'Tasmania', 'Victoria', 'Western Australia']), 'Australia'
    ).when(col("State/Region").isin(['Republic of Kosovo','Albania','North
Macedonia','Bosnia and Herzegovina','Republic of
Moldova','Belarus','Montenegro','Gibraltar','Guernsey','Isle of
Man','Jersey','Montserrat','Pitcairn Islands','Saint Barthelemy','Saint Pierre
and Miquelon','French Guiana','Martinique','Mayotte' ,'St Martin','Wallis and
Futuna','New Caledonia','French Polynesia','Guadeloupe','French
Guiana','Mayotte','Martinique','Reunion','Serbia','Czechia','Austria',
'Belgium', 'Bulgaria', 'Croatia', 'Cyprus', 'Czech Republic', 'Denmark',
'Estonia', 'Finland', 'France', 'Germany', 'Greece', 'Hungary', 'Iceland',
'Ireland', 'Italy', 'Latvia', 'Liechtenstein', 'Lithuania', 'Luxembourg',
'Malta', 'Netherlands', 'Norway', 'Poland', 'Portugal', 'Romania', 'Slovakia',
'Slovenia', 'Spain', 'Sweden', 'Switzerland', 'United Kingdom','Ukraine']),
'Europe'
    ).when(col("State/Region").isin(['Burma','West Bank and
Gaza','Micronesia','Guizhou','Hainan','Hebei','Heilongjiang','Henan','Hong
Kong','Hubei','Hunan','Inner
Mongolia','Jiangsu','Jiangxi','Jilin','Liaoning','Macau','Ningxia','Qinghai','
Shaanxi','Shandong','Shanghai','Shanxi','Sichuan','Tianjin','Tibet','Xinjiang'
,'Yunnan','Zhejiang','Anhui','Beijing','French Guiana','Chongqing','Fujian'
,'Gansu','Guangdong','Hong Kong','Afghanistan', 'Armenia', 'Azerbaijan',
'Bahrain', 'Bangladesh', 'Bhutan',"Brunei Darussalam", 'Cambodia', 'China',
'Georgia', 'India', 'Indonesia', 'Iran', 'Iraq', 'Israel', 'Japan', 'Jordan',
'Kazakhstan', 'Kuwait', 'Kyrgyzstan', "Lao People's Democratic Republic",
'Lebanon', 'Malaysia', 'Maldives', 'Mongolia', 'Myanmar', 'Nepal', 'South
Korea', 'Oman', 'Pakistan', 'Palestine', 'Philippines', 'Qatar', 'Russian
Federation', 'Saudi Arabia', 'Singapore', 'Korea South', 'Sri Lanka', 'Syria',
'Taiwan', 'Tajikistan', 'Thailand', 'Timor-Leste', 'Turkey', 'Turkmenistan',
'United Arab Emirates', 'Uzbekistan', 'Viet Nam', 'Yemen']), 'Asia'
    ).when(col("State/Region").isin(['Congo (Kinshasa)','Algeria', 'Angola',
'Benin', 'Botswana', 'Burkina Faso', 'Burundi', 'Cabo Verde', 'Cameroon',
'Central African Republic', 'Chad', 'Comoros', 'Democratic Republic of the
Congo', 'Republic of the Congo', 'Cote d Ivoire', 'Djibouti', 'Egypt',
'Equatorial Guinea', 'Eritrea', 'Eswatini', 'Ethiopia', 'Gabon', 'Gambia',
'Ghana', 'Guinea', 'Guinea-Bissau', 'Kenya', 'Lesotho', 'Liberia', 'Libya',
'Madagascar', 'Malawi', 'Mali', 'Mauritania', 'Mauritius', 'Morocco',
'Mozambique', 'Namibia', 'Niger', 'Nigeria', 'Rwanda', 'Sao Tome and
Principe', 'Senegal', 'Seychelles', 'Sierra Leone', 'Somalia', 'South Africa',
'South Sudan', 'Sudan', 'Tanzania', 'Togo', 'Tunisia', 'Uganda', 'Zambia',
'Zimbabwe']), 'Africa'
    ).when(col("State/Region").isin(['Argentina', 'Bolivia', 'Brazil',
'Chile', 'Colombia', 'Costa Rica', 'Cuba', 'Dominican Republic', 'Ecuador',
'El Salvador', 'Guatemala', 'Haiti', 'Honduras', 'Jamaica', 'Mexico',
```

```
'Nicaragua', 'Panama', 'Paraguay', 'Peru', 'Puerto Rico', 'Trinidad and
Tobago', 'United States Virgin Islands', 'Uruguay', 'Venezuela (Bolivarian
Republic)']), 'South America'
    ).otherwise('Unknown')
)

# We will then apply this mapping to our dataframe to create a new column
"Continent"
Joined_long_df = Joined_long_df.withColumn("Continent", continent_mapping)
Joined_long_df = Joined_long_df.select("Continent", *[col for col in
Joined_long_df.columns if col != "Continent"])

# we start by extracting week number from 'Date' column to defien a new column
'Week'
Joined_long_df = Joined_long_df.withColumn("Year", year("Date"))
Joined_long_df = Joined_long_df.withColumn("Week", weekofyear("Date"))


# we will Group data by 'Continent' and 'Week', and calculate the metrics
result_df = Joined_long_df.groupBy("Continent", "Year","Week").agg(
    mean("Daily_Cases").alias("Mean_Cases"),
    stddev("Daily_Cases").alias("Stddev_Cases"),
    min("Daily_Cases").alias("Min_Cases"),
    max("Daily_Cases").alias("Max_Cases")
)

# we Sort the DataFrame by 'Continent' , 'year' and  'Week'
Task2_df = result_df.orderBy("Continent", "Year","Week")

# count the number of rows
num_rows = Task2_df.count()

# count the number of columns
num_cols = len(Task2_df.columns)

# print "Number of rows" , "Number of columns"
print("Number of rows:", num_rows)
print("Number of columns:", num_cols)

# Show the result
Task2_df.show(10, truncate=False)
# Task2_df.write.format("csv").option("header",
"true").mode("overwrite").save("Task2_df.csv")
# calculate the time it took to run the code of task2
Task2_time = time.time() - Task1_time - Cleaning_data_time- Debute
# print the duration
print("Task2_time: ", Task2_time , "seconds")
# we will start by importing the previous DataFrame in the long format
file_path = "data/long_df2.csv"
```

```
data = pd.read_csv(file_path)

# we convert the 'Date' column to datetime and  then extract year and month
data['Date'] = pd.to_datetime(data['Date'])
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month

# we sort the data by 'State/Region', 'Year' and 'Month'
grouped_data = data.groupby(['State/Region', 'Year', 'Month'])

# Now, proceed with the aggregation
Organized_data = grouped_data.agg({'Daily_Cases': 'sum'}).reset_index()
Organized_data = Organized_data.rename(columns={'Daily_Cases': 'Cases'})
# Organized_data.head(10)

# define a function to calculate the trendline coefficient for a given x, y
series
def trendline_coef(x, y):
    linear_coeffs = np.polyfit(x, y, 1)
    return linear_coeffs[0]  # Coefficient of the linear term

# define an other function to apply trendline_coef() to each "State/Region"
group
def state_trendline(group):
    # we Generate an array of X values from 0 to the length of the group
    X = np.arange(len(group))

    # We Use the 'Cases' column as Y values
    Y = group['Cases']

    # We Calculate the trendline coefficient using our function
    coefficient = trendline_coef(X, Y)

    return coefficient

# We then apply this function to each column in 'grouped_data' and reset the
index
# The lambda function here is used to apply the custom function to each group
trendline_coefs = grouped_data.apply(lambda g:
state_trendline(g)).reset_index()

# We then rename the column that contains the calculated coefficients for
clarity
trendline_coefs.rename(columns={0: 'Trendline_Coefficient'}, inplace=True)


# Sorting and selecting the top 50 states/regions for each month
top_50_State = trendline_coefs.sort_values(by=['Year', 'Month',
'Trendline_Coefficient'], ascending=[True, True, False]) \
```

```
                                    .groupby(['Year', 'Month']).head(50)

# top_50_State.head()

# top_50_State.to_csv('data/top_50_State.csv', index=False)
TOP50_STATE = ss.read.csv('data/top_50_State.csv' , header=True,
inferSchema=True)

#   We will Combine "Year" and "Month" into a single "Year-Month" column
TOP50_STATE = TOP50_STATE.withColumn("Year-Month", concat_ws("-",
TOP50_STATE.Year, TOP50_STATE.Month))

# We then Convert the 'Trendline_Coefficient' column to a vector column befor
applying the KMeans algorithm
vec_assembler = VectorAssembler(inputCols=["Trendline_Coefficient"],
outputCol="features")
TOP50_STATE_kmeans = vec_assembler.transform(TOP50_STATE)

# we group TOP50_STATE  by year and month and then  apply KMeans for each
Month
k = 4  # Number of clusters
results_df = None

# Apply KMeans for each "Year-Month" group
unique_year_months = TOP50_STATE.select("Year-Month").distinct().collect()
for row in unique_year_months:
    year_month = row["Year-Month"]
    df_month = TOP50_STATE_kmeans.filter(col("Year-Month") == year_month)
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
    model = kmeans.fit(df_month)

    # We will Predict the cluster for each data point
    predictions = model.transform(df_month)

    # Update the results DataFrame
    if results_df is None:
        results_df = predictions
    else:
        results_df = results_df.union(predictions)

# Sort the results DataFrame by "Year-Month" and "prediction"
results_df = results_df.orderBy("Year-Month", "prediction")

Task3_df = results_df.drop("Year", "Month", "features").select("State/Region",
"Year-Month", "Trendline_Coefficient", "prediction")
Task3_df.show()
# Task3_df.write.csv("data/Task3_df.csv", header=True)
# calculate the time it took to run the code of task3
```

```
Task3_time = time.time() - Task2_time - Task1_time - Cleaning_data_time-
Debute
# print the duration
print("Task3_time: ", Task3_time , "seconds")
```

Visualization of The results

```
import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import geoplot as gplt
from matplotlib.colors import Normalize
import plotly.express as px

# Loading the dataset
V_data = pd.read_csv('data/Cleaned_df.csv')

# Define a function to plot the map Based on the "lat" and "long" columns of a
DataFrame
def plot_map(DataFrame):

    # Plotting the world map
    plt.figure(figsize=(5, 3))
    world_Map = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
    world_Map.plot(color='lightblue', edgecolor='black')

    # Plotting the points based on latitude and longitude
    plt.scatter(DataFrame['Long'], DataFrame['lat'], color='red', alpha=0.9,
marker='o', s=5)

    plt.title('Geographical Representation of The regions affected by COVID-
19')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.show()

# Apply the function to the our covid dataframe
plot_map(V_data)

def plot_BarGraph(df, countries, dates, colors, xlabel, ylabel, title):
    bar_width = 0.15  # the width of each bar

    # Plot the average cases for each country in the list
    for idx, country in enumerate(countries):
        # Filter the data for the country and the specified dates
        country_data = df[(df['Country/Region'] == country) &
(df['YearMonth'].isin(dates))]
```

```
            # Extract the average cases for the given dates for each country
            avg_cases = country_data['AverageDailyCases'].tolist()

            # Position of bars on x-axis
            bar_positions = np.arange(len(dates)) + idx * bar_width

            # Draw the bars for this country in the chart
            plt.bar(bar_positions, avg_cases, color=colors[idx], width=bar_width,
label=country)

    # Add the provided labels and title
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)

    # Set the position of labels on the x-axis
    plt.xticks(np.arange(len(dates)) + bar_width / 2, dates)

    # Add a legend to the chart
    plt.legend()

    # Finally, display the chart
    plt.show()


# Adjusting the country names and colors as per the new request
countries = ["Morocco", "China", "Australia", "United States of America",
"Brazil", "Russian Federation"]
colors = ["Orange", "Red", "Blue", "Yellow", "Green", "Brown"]
dates = ["2020-01", "2020-06", "2022-01", "2022-06", "2023-01", "2023-03"]
df_Task1_long = pd.read_csv('data/long_Task1_df.csv')
# Plotting the graph with the updated country list and colors
plot_BarGraph(df_Task1_long, countries, dates, colors, 'Month', 'Average
Cases', 'COVID-19 Cases Over Time for Selected Countries')

df_Task1 = pd.read_csv('data/Task1_df.csv')

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 15))
axes = axes.flatten()

for X, country, color in zip(axes, countries, colors):
    country_data = df_Task1[df_Task1['Country/Region'] ==
country][dates].mean()
    X.bar(dates, country_data, color=color,width= 0.3)
    X.set_title(f'COVID-19 Cases over time in {country}')
    X.set_xlabel('Dates')
    X.set_ylabel('Average Number of Cases')
    X.tick_params(axis='x', rotation=45)
```

```
plt.tight_layout()
plt.show()
Top_100 = pd.read_csv('data/Countries_Trendline.csv')

# Select random dates from the list of dates and Countries from the list of
"State/Region"
dates = ["2020-01-22", "2020-06-01", "2021-01-01", "2021-06-01", "2022-01-01",
"2022-06-01", "2023-01-01", "2023-03-01"]
countries = ["United States of America", "France", "Ireland", "China"]

# Select the "State/Region" that matches the countries in the list and the
dates in the list
filtered_data = Top_100[(Top_100['State/Region'].isin(countries)) &
(Top_100['Date'].isin(dates))]

# Determine the layout of subplots based on the number of countries
Sub_Plots = len(countries) // 2 + len(countries) % 2
fig, axs = plt.subplots(Sub_Plots, 2, figsize=(14, 5 * Sub_Plots),
squeeze=False)

# Plotting each country's data
for idx, country in enumerate(countries):
    axes = axs[idx // 2, idx % 2]
    country_data = filtered_data[filtered_data['State/Region'] ==
country].sort_values(by='Date')

    # Plotting with line
    axes.plot(country_data['Date'], country_data['Daily_Cases'], marker='o',
linestyle='-', color='b')
    axes.set_title(f'Daily COVID-19 Cases in {country} on Selected Dates')
    axes.set_xlabel('Date')
    axes.set_ylabel('Daily Cases')
    axes.tick_params(axis='x', rotation=45)

# Adjusting layout
plt.tight_layout()
plt.show()

task2_df = pd.read_csv('data/Task2_df.csv')

# We will start by ploting a time line series for each continent
continents = task2_df['Continent'].unique()

plt.figure(figsize=(15, 8))
for continent in continents:
    continent_data = task2_df[task2_df['Continent'] == continent]
    sns.lineplot(data=continent_data, x='Week', y='Mean_Cases',
label=continent)
```

```
plt.title('Trend of Mean COVID-19 Cases Over Weeks by Continent ')
plt.xlabel('Week Number')
plt.ylabel('Mean Cases')
plt.legend(title='Continent')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(data=task2_df, x='Mean_Cases', y='Stddev_Cases',
hue='Continent', palette='Set2', marker='o')
plt.title('Relationship Between Mean Cases and Standard Deviation by
Continent')
plt.xlabel('Mean Cases')
plt.ylabel('Standard Deviation of Cases')
plt.legend(title='Continent')
plt.grid(True)
plt.show()
# Box Plot for Weekly Cases in the year 2020
Year_2021 = task2_df[task2_df['Year'] == 2021]

# Selecting a few weeks for demonstration
weeks = [10, 20, 30, 40, 50]
weeks_data = Year_2021[Year_2021['Week'].isin(weeks)]

# Bar Graphs for Min and Max Cases
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))

# Minimum Cases Bar Graph
sns.barplot(data=weeks_data, x='Week', y='Mean_Cases', hue='Continent',
ax=ax1)
ax1.set_title('Average COVID-19 Cases Across Selected Weeks by Continent in
2021')
ax1.set_xlabel('Week Number')
ax1.set_ylabel('Average Cases')

# Maximum Cases Bar Graph
sns.barplot(data=weeks_data, x='Week', y='Max_Cases', hue='Continent', ax=ax2)
ax2.set_title('Maximum COVID-19 Cases Across Selected Weeks by Continent in
2021')
ax2.set_xlabel('Week Number')
ax2.set_ylabel('Maximum Cases')

plt.tight_layout()
plt.show()


# Import the data
Top_50  = pd.read_csv('data/top_50_State.csv')
```

```
# Create a new 'Year-Month' column to respect the format of the dataframe in
the file 'top_50_State.csv'
Top_50['Year-Month'] = Top_50['Year'].astype(str) + '-' +
Top_50['Month'].astype(str).str.zfill(2)

# Load world map geometries into a GeoDataFrame
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world = world.explode()

# List of random chosen months to analyze
months = ['2020-01', '2020-06', '2021-01', '2021-06', '2022-01', '2022-06',
'2023-01', '2023-03']

# Create a figure for the plots
fig, axs = plt.subplots(nrows=4, ncols=2, figsize=(15, 40))

for i, month in enumerate(months):
    # Determine the subplot row and column
    row, col = divmod(i, 2)

    # Select the top 50 countries/States for the given month
    month_df = Top_50[Top_50['Year-Month'] == month].head(50)

    # Select the countries/States common between the world map and the month
DataFrame
    merged_data =
world.set_index('name').join(month_df.set_index('State/Region'))

    # Plotting the choropleth map with a legend for the month
    ax = gplt.choropleth(
        merged_data, hue='Trendline_Coefficient',
        cmap='Reds', ax=axs[row, col],
        legend=True,
        norm=Normalize(vmin=merged_data['Trendline_Coefficient'].min(),
vmax=merged_data['Trendline_Coefficient'].max())
    )
    ax.set_title(f'Top ranked 50 countries for {month}')

# Show the plot
plt.tight_layout()
plt.show()


task3_df = pd.read_csv('data/Task3_df.csv')

# Select the months to analyze randomly
months_selected = ['2020-3', '2020-9', '2021-3', '2021-10', '2022-5', '2022-
9', '2023-1', '2023-3']
```

```
# Filtering the data for the updated months to analyze
filtered_data = task3_df[task3_df['Year-Month'].isin(months_selected)]

# Creating an interactive scatter plot
fig = px.scatter(filtered_data, x="Trendline_Coefficient", y="State/Region",
                 color="prediction", hover_name="State/Region",
                 facet_col="Year-Month", facet_col_wrap=2,
                 title="States/Regions Clustering by Month",
                 labels={"prediction": "Cluster", "Trendline_Coefficient":
"Trendline Coefficient"})

# Adjust layout
fig.update_layout(height=800, hovermode='closest')

# Show the plot
fig.show()
# calculate the time it took to run the code of the visualization of the
results
Visualization_time = time.time() - Task3_time - Task2_time - Task1_time -
Cleaning_data_time- Debute
# print the duration
print("Visualization_time: ", Visualization_time , "seconds")

# plot the time it took to run each task in a bar chart
plt.figure(figsize=(10, 5))
plt.bar(['Cleaning_data_time', 'Task1_time', 'Task2_time', 'Task3_time',
'Visualization_time'], [Cleaning_data_time, Task1_time, Task2_time,
Task3_time, Visualization_time], color=['red', 'blue', 'green', 'orange',
'purple'], width=0.3)
plt.xlabel('Tasks')
plt.ylabel('Time (seconds)')
plt.title('Time taken to run each task')
plt.show()
```