**Disease Outbreaks Prediction Using Time Series Analysis and Classification**

**Overview:**

This project focuses on predicting disease outbreaks with a particular emphasis on Zika and Hepatitis E as a case study. We utilized time-series analysis techniques such as ARIMA, Prophet, and LSTM, alongside classification methods including decision trees, random forests, and neural networks to forecast and categorize outbreak risks.

**Team Members:**

* Rhythm Gupta – 500101921
* Shlok Chawla - 500105057

**Abstract:**

We developed predictive models to classify countries into low, moderate, and high-risk categories based on their healthcare infrastructure, socio-economic metrics and parsed news headlines, using machine learning algorithms. The methodologies are adaptable to other infectious diseases, making this a versatile tool in global health crisis management.

**Data Collection:**

Data is collected from various sources :

* Outbreak data from WHO's Disease Outbreak News and the Coronavirus Dashboard.
* Medical parameter data from various sources including Statista covering hospital bed density, ICU availability, health expenditure, and more.
* Google Scrapper (For scrapping headlines)

**Tools:**

* We are using Jupyter notebook to conduct this analysis
* **Libraries used :** Pandas, Numpy, Geonamescache, RE, UNICODE

**Methods :**

* We employed a variety of forecasting and classification techniques:
  + **Time Series Forecasting:** ARIMA, Prophet, and LSTM.
  + **Classification:** Random Forest, Decision Trees, and Neural Networks.

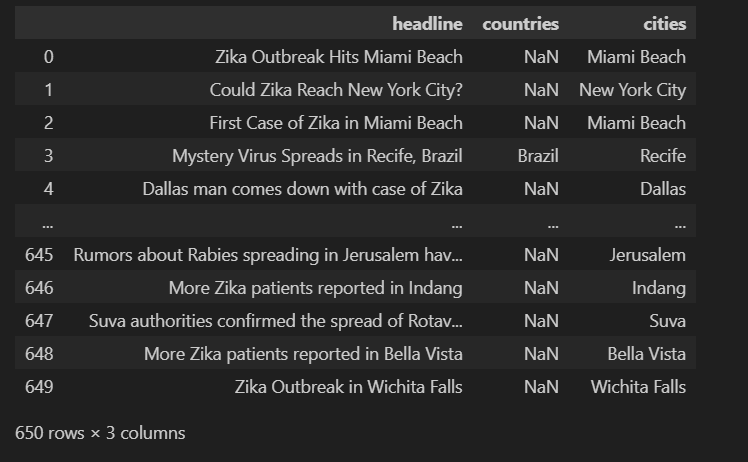
PART-1

Parsing News headlines

The code aims to extract and analyze news headlines from a text file named "headlines.txt." It identifies countries and cities mentioned in the headlines using the GeonamesCache library. The extracted data is then organized into a pandas DataFrame for further analysis.

**Key Steps:**

1. **Import Libraries:**
   * pandas: For data manipulation and analysis.
   * geonamescache: To access geographical data from the GeoNames database.
   * re: For regular expressions (if needed for more complex text processing).
   * unidecode: To convert Unicode characters to ASCII equivalents (for better compatibility with the GeoNames database).
2. **Read Headlines:**
   * Open the "headlines.txt" file and read its contents line by line, storing each headline in the arr list.
3. **Initialize Dictionary:**
   * Create a dictionary to store the extracted headlines, countries, and cities.
4. **Iterate Through Headlines:**
   * Iterate over each headline in the arr list.
   * **Normalize Text:** Convert Unicode characters to ASCII using unidecode.unidecode() to ensure compatibility with the GeoNames database.
   * **Tokenization:** Split the headline into individual words using i.split().
   * **City Extraction:**
     + Iterate over each word in the headline.
     + If the word is a city name (according to GeoNames), add it to the poss\_cities list.
     + If no city is found, try combining the current word with subsequent words to form longer city names.
     + If a city is found, append the last matching city to the dictionary['cities'] list.
   * **Country Extraction:**
     + Iterate over each word in the headline.
     + If the word is a country name (according to GeoNames), add it to the poss\_countries list.
     + If no country is found, try combining the current word with subsequent words to form longer country names.
     + If a country is found, append the last matching country to the dictionary['countries'] list.
   * **Handle Missing Values:** If a headline doesn't have a matching country or city, append 'NaN' to the corresponding list in the dictionary.
5. **Create DataFrame:**
   * Create a pandas DataFrame from the dictionary to organize the extracted data in a structured format.



PART-2

Finding Geographic Locations

The code helps to effectively extracts latitude, longitude, and country codes from city names mentioned in news headlines. Key steps:

1. **Import necessary libraries:**
   * pandas for data manipulation.
   * geonamescache for accessing geographical data.
   * re for regular expressions (if needed).
   * unidecode for converting Unicode characters to ASCII.
2. **Load data:**
   * Read the CSV file containing news headlines into a pandas DataFrame.
3. **Initialize lists:**
   * Create empty lists to store latitude, longitude, and country code values.
4. **Iterate through headlines:**
   * Iterate over each row in the DataFrame.
   * If the city name is not found in the GeoNames database or is "NaN," append "NaN" to the corresponding lists.
   * If the city is found, extract its latitude, longitude, and country code using the geonamescache library.
5. **Create new columns:**
   * Add new columns to the DataFrame for latitude, longitude, and country codes, and assign the extracted values to these columns.
6. **Drop the original "countries" column:**
   * Remove the "countries" column from the DataFrame, as it is no longer needed.



PART-3:

Clustering Headlines:

The code effectively performs K-Means and DBSCAN clustering on geographical coordinates from a CSV file and visualizes the results using Basemap. Key Steps are:

**Data Loading and Cleaning:**

1. **Load CSV:** The code reads the CSV file containing latitude, longitude, city names, and country codes.
2. **Remove Unnamed Column:** It removes the unnecessary "Unnamed: 0" column.
3. **Create DataFrames:** Two DataFrames (new\_df and new\_df\_with\_cities) are created, one containing only latitude and longitude, and the other including all columns.
4. **Handle Missing Values:** The code removes rows with missing values in both DataFrames using dropna(thresh=2). This removes rows with more than 2 missing values. Consider adjusting the threshold based on your data quality.

**K-Means Clustering:**

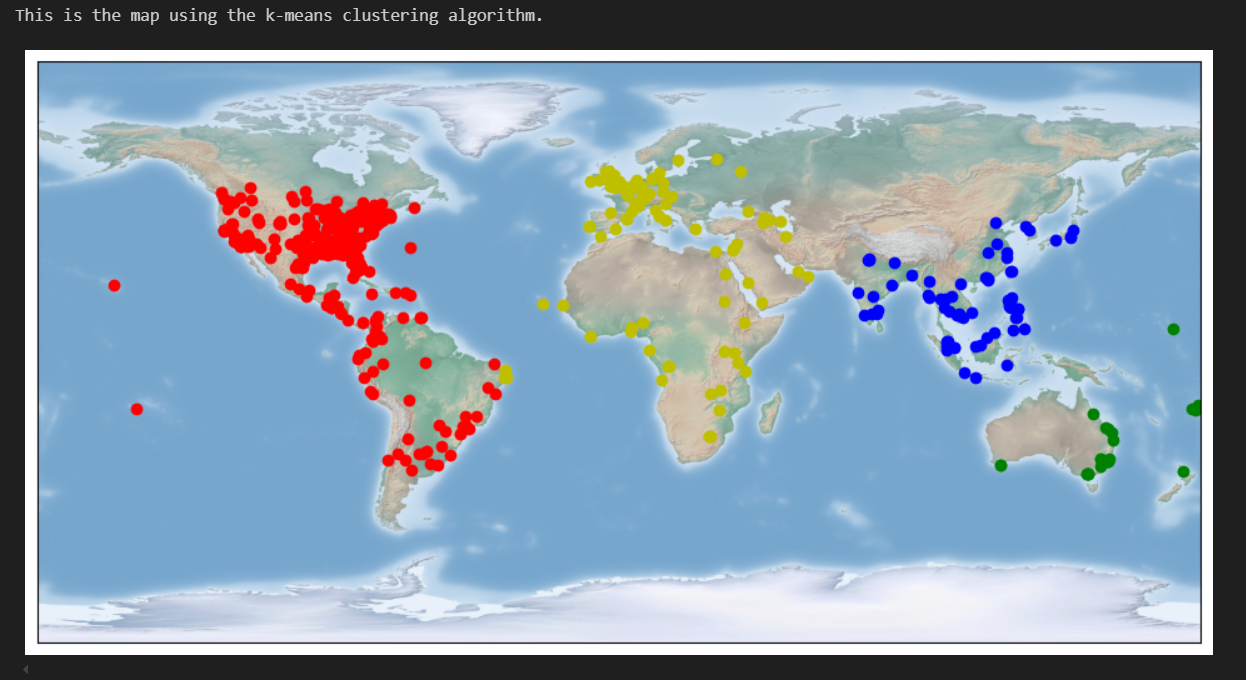
1. **Elbow Method (Commented Out):** While the code includes commented-out sections for the Elbow Method, it directly assigns 4 clusters for K-Means. You can uncomment the code to find the optimal number of clusters for your data using this method.
2. **K-Means Model:** A K-Means model with 4 clusters is fitted to the new\_df DataFrame, and the cluster labels are assigned to the DataFrames.
3. **Visualization:** The code is commented out for plotting the clusters on a global map using K-Means. You can uncomment this section to visualize the clusters.

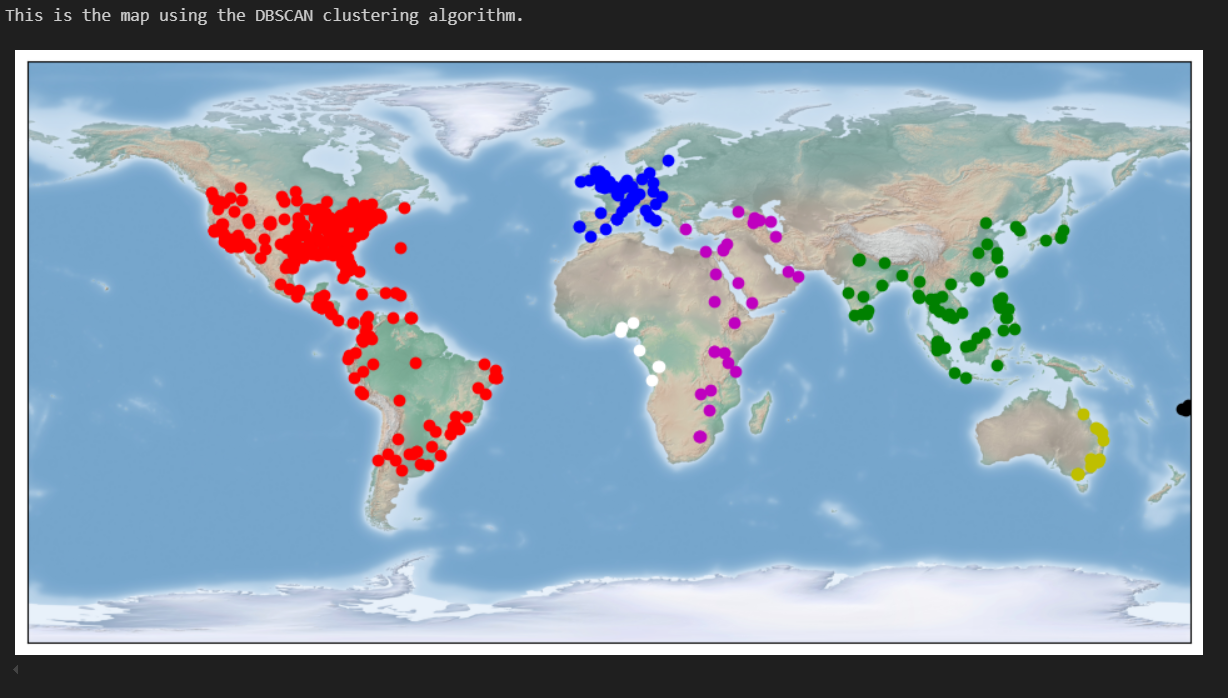
**DBSCAN Clustering:**

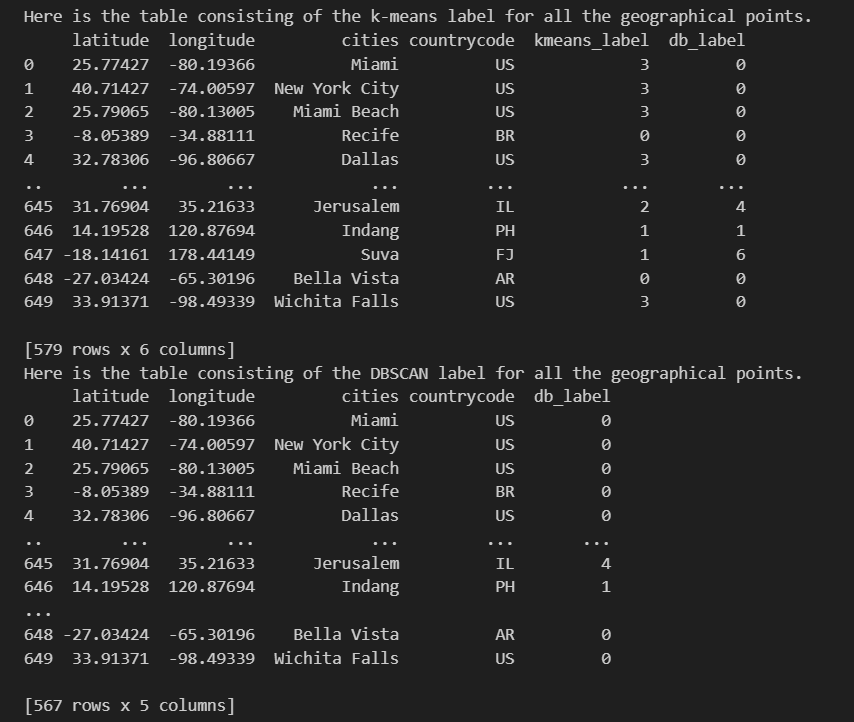
1. **Distance Function:** The code defines the haversine function to calculate the great-circle distance between two geographical coordinates, which is a more accurate measurement for distances on a sphere like Earth.
2. **DBSCAN Model:** A DBSCAN model with eps=1000 (epsilon, a distance threshold) and min\_samples=3 (minimum samples in a cluster) is fitted to the list of points extracted from the new\_df DataFrame.
3. **Outlier Removal:** The code removes rows with a DBSCAN label of -1 (considered outliers) from the new\_df\_with\_cities\_db\_scan DataFrame.
4. **Visualization:** The clusters identified by DBSCAN are plotted on a Basemap using different colors.

**Printing Results:**

* The code prints the updated DataFrames (new\_df\_with\_cities and new\_df\_with\_cities\_db\_scan) containing the city names, country codes, and K-Means/DBSCAN cluster labels.







PART-4:

**Clustering Disease Headlines:**

This code clusters news headlines based on geographic locations (latitude, longitude) and visualizes clusters on a map using K-Means clustering. It also provides options for keyword-based search and filters the news based on keywords like "zika" and "hepatitis."

**Key Steps:**

**1. Data Loading and Cleaning:**

* **Load CSV File**: The CSV file containing news headlines, geographic coordinates (latitude, longitude), city names, and country codes is loaded.
* **Remove Unnecessary Column**: The column 'Unnamed: 0' is deleted as it's not required.
* **Create DataFrames**: Three dictionaries (dict\_us, dict\_other, and dict\_no\_world) are created to separate data based on US and non-US countries. DataFrames df\_us, df\_other, df\_no\_us, df\_no\_other, and df\_no\_world are created from these dictionaries.
* **Handle Missing Values**: Rows with missing values are removed from the datasets using the dropna() function to ensure clean data for clustering.

**2. Elbow Curve Method for Optimal K-Means Clusters:**

* **Elbow Curve Function**: The elbow\_curve() function plots the Elbow curve to help find the optimal number of clusters. It uses K-Means clustering for values of clusters ranging from 1 to 20. The score for each K-Means model is plotted to visualize the "elbow," where the curve bends, indicating the optimal cluster number.

**3. K-Means Clustering:**

* **K-Means Cluster Assignment**: The function print\_k\_means() applies K-Means clustering to the given data (new\_df) for the specified number of clusters (num\_cluster). It assigns cluster labels to each data point and stores them in a new column 'kmeans\_label'.
* **Geographical Map Visualization**: The function run\_k\_means() visualizes the clustering results on a geographical map. Depending on whether the type is "US" or "world," it uses Basemap to plot clusters in different colors based on their latitude and longitude.

**4. Cluster Headlines Based on K-Means Labels:**

* **Counting Headlines per Cluster**: The code counts the number of headlines per cluster for both US (headline\_count\_us) and world (headline\_count\_world) clusters.
* **Cluster Dictionary Creation**: For each cluster, the code creates a dictionary that stores headlines and their corresponding latitude, longitude, and K-Means labels. This is done for both US and world datasets.

**5. Distance Calculation Using Haversine Formula:**

* **Cluster Center Calculation**: The center of each K-Means cluster is computed by averaging the latitude and longitude values of the data points in the cluster. These centers are stored in dict\_center\_us and dict\_center\_world for US and world clusters respectively.
* **Great Circle Distance (Haversine)**: The haversine() function calculates the great-circle distance between two points (latitude, longitude) on Earth. This formula is used to measure the distance of each headline from its cluster center.

**6. Sorting Headlines by Proximity to Cluster Centers:**

* **Distance Calculation for Headlines**: The distance of each headline from the center of its respective cluster is computed and stored in the 'distance' column. This process is repeated for both US and world clusters.
* **Sorting Headlines**: The code sorts the headlines in each cluster based on their distance to the cluster center, prioritizing headlines that are geographically closest to the center.

**7. Keyword-Based Search Function:**

* **Search Function**: The function search\_dataframe() filters headlines that contain a specific keyword (e.g., "zika" or "hepatitis"). It creates a new DataFrame of headlines that match the keyword, along with their geographic coordinates and K-Means labels.
* **Disease-Specific Filtering**: The code uses search\_dataframe() to filter headlines for specific diseases, creating filtered DataFrames for Zika and Hepatitis outbreaks for both the US and the world.

**8. Visualizing Disease Outbreaks on a Map:**

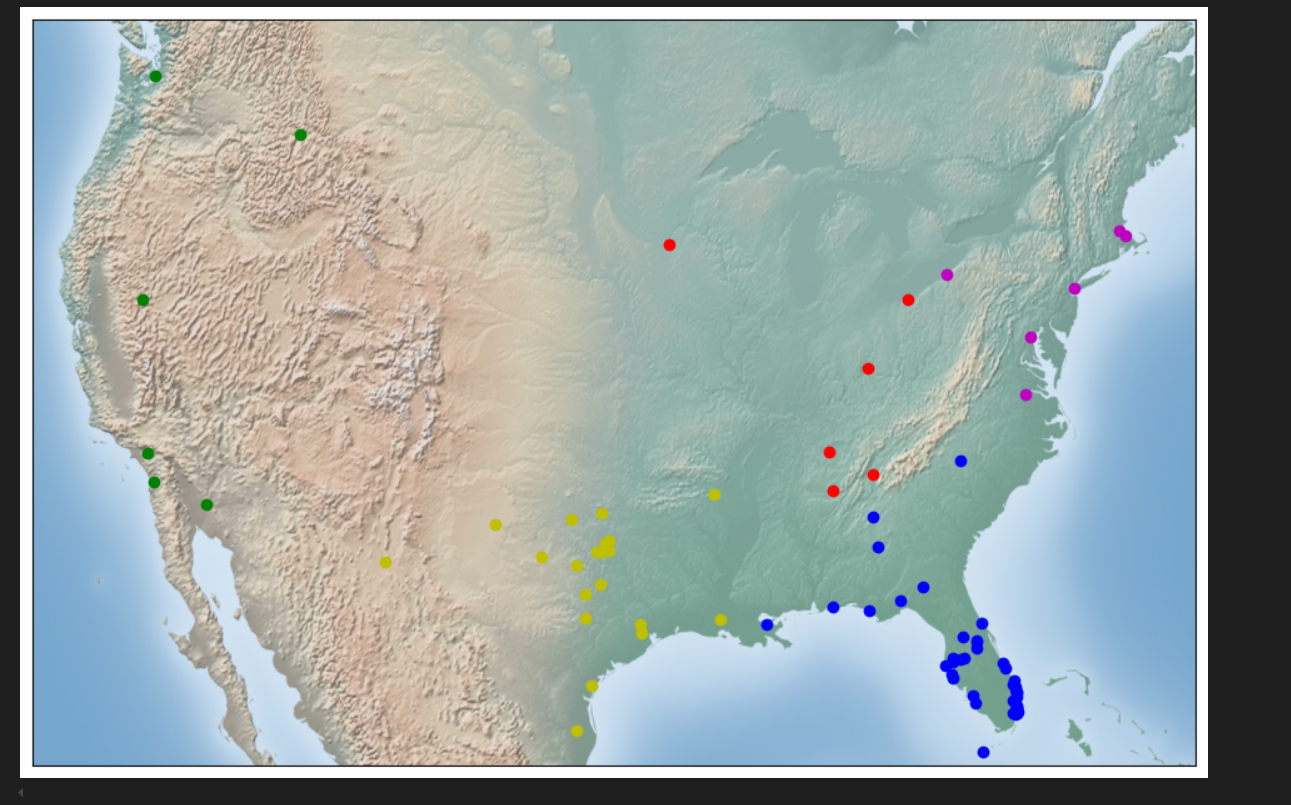
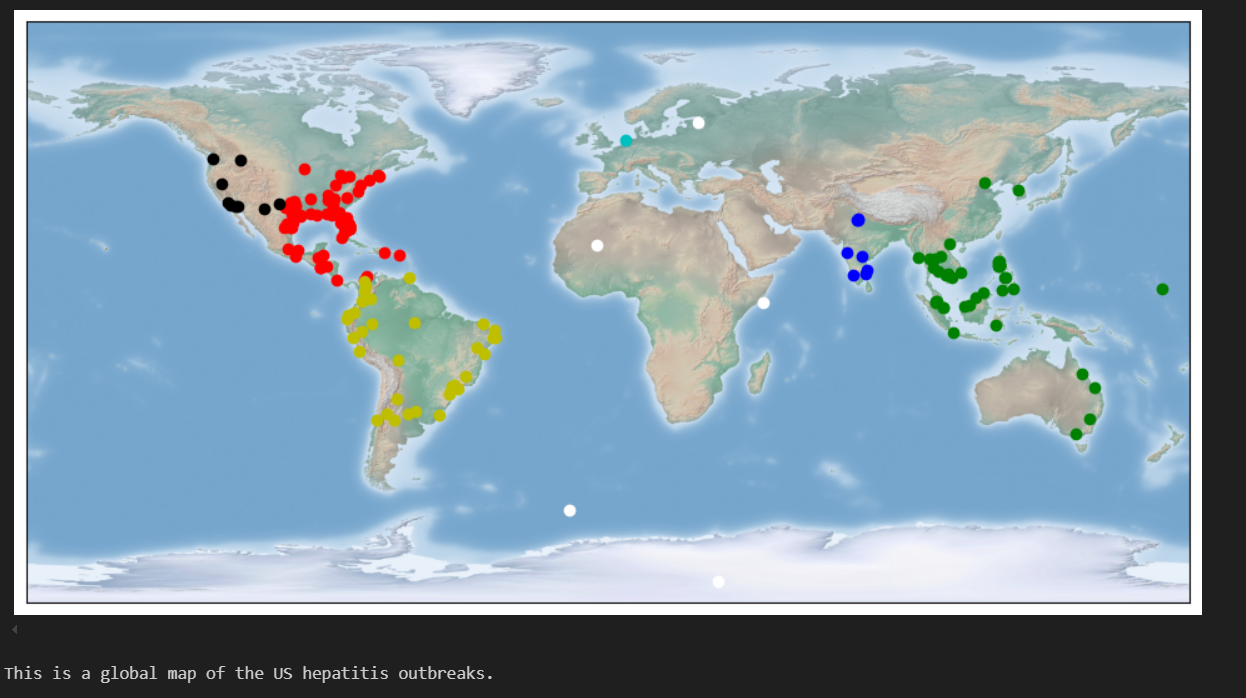
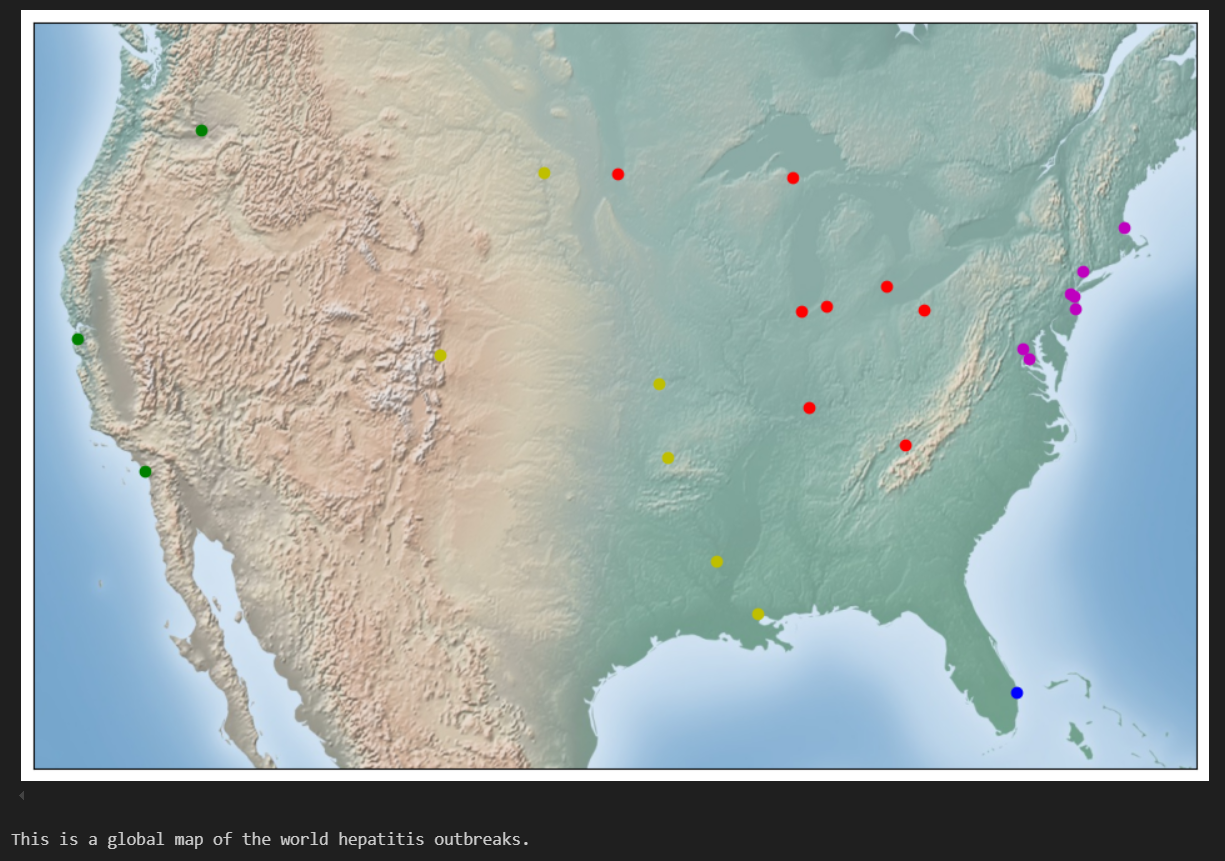
* **Disease Outbreak Maps**: The run\_k\_means() function is used to plot the geographic distribution of disease outbreaks (Zika and Hepatitis) for both US and world regions. The clustered headlines are displayed on a map, showing the disease hotspots.

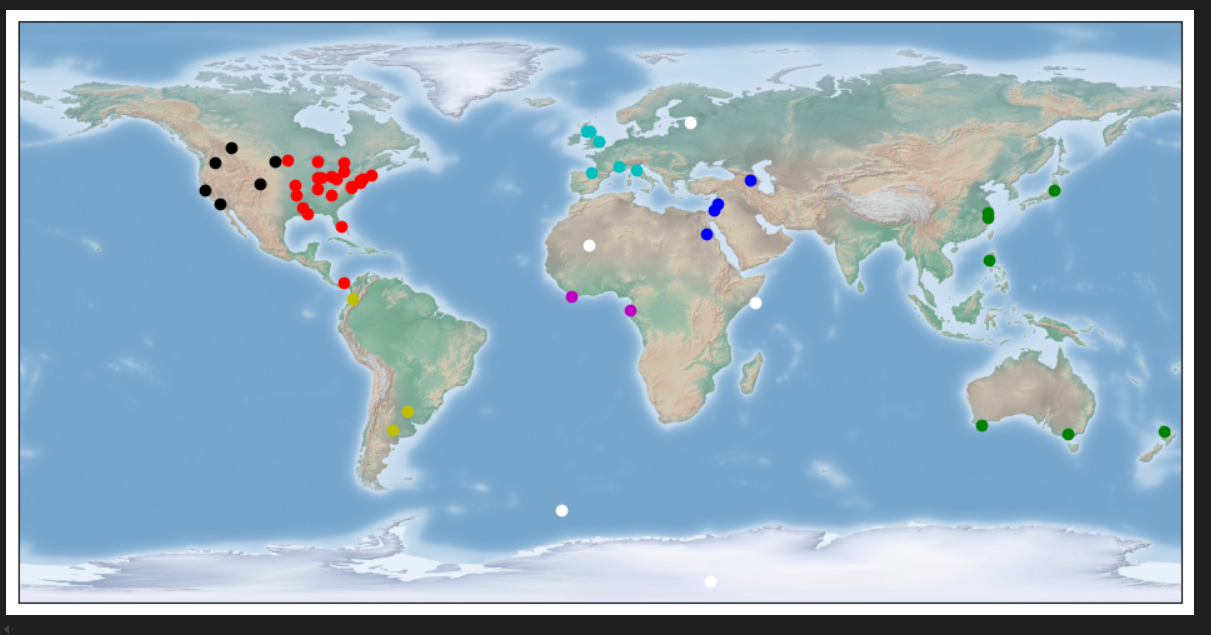
**9. Saving DataFrames:**

* **Saving Results to CSV**: The filtered DataFrames (df\_no\_us\_zika, df\_no\_world\_zika, df\_no\_us\_hepatitis, df\_no\_world\_hepatitis) are saved as CSV files for further analysis or reporting purposes.

**Key Sections of the Code:**

* **K-Means Clustering**: The code clusters the geographic data and visualizes it on a map, helping to identify patterns and disease hotspots.
* **Keyword-Based Filtering**: The ability to search for specific keywords like "zika" or "hepatitis" helps in focusing on particular diseases.
* **Great Circle Distance (Haversine Formula)**: The use of the Haversine formula ensures accurate distance calculations over the Earth's curved surface.



Part-5:

This script is designed to extract and filter real-time headlines from the top 100 news websites, focusing on coronavirus-related content. Below is a detailed step-by-step explanation of how the code works:

**Step 1: Import Libraries**

Several libraries are imported, each playing a key role in different functionalities of the code:

* BeautifulSoup from bs4: Used to parse and extract data from HTML.
* urllib.request: Opens and reads web URLs.
* URLValidator from django.core.validators: Validates URLs to ensure proper formatting.
* time and datetime: Used for handling time-based operations, such as limiting the script runtime.

**Step 2: Function to Extract Websites**

A function named extractWebsites(url, exceptionURL) is defined to fetch and extract hyperlinks from a given URL. Here's the process:

* The URL is opened using urllib.
* The HTML data is parsed using BeautifulSoup.
* A list of all links (<a> tags) is extracted.
* The extracted URLs are validated using Django's URLValidator. Invalid URLs or URLs containing the exceptionURL (like "interactive") are skipped. Valid URLs are added to an array and returned.

**Step 3: Function to Search for Specific Keywords in URLs**

A function search\_array(url, arrOfPoss) is implemented to search for a set of keywords (related to COVID-19) in the extracted URLs. It:

* Iterates through an array of keywords.
* Returns True if any keyword is found in the URL (case insensitive); otherwise, returns False.

**Step 4: Extracting and Filtering Headlines**

The actual headline extraction and filtering process unfolds in multiple phases:

1. **Initial Extraction**:
   * The code uses extractWebsites on Feedspot’s top 100 news websites.
   * The extracted URLs are then filtered to remove URLs containing "interactive" and checked against a set of keywords (like 'coronavirus', 'COVID', 'pandemic', etc.) using search\_array. These filtered URLs are stored in the covid\_urls array.
2. **Refinement with Nested Loops**:
   * The script then refines this set of URLs further by running through covid\_urls, extracting even more relevant URLs, storing them in new\_covid\_urls, and repeating the filtering process.
   * The inner loop repeats one more time to generate the most refined set of URLs (newest\_covid\_urls) that are highly relevant to COVID-19.
3. **Time and Count Control**:
   * A counter (count) is implemented to stop the execution if too many URLs are processed.
   * Additionally, a time limit (600 seconds) is imposed to prevent the loops from running indefinitely. If the process exceeds this time, it is terminated.
4. **Headline Extraction**:
   * The extracted URLs are converted back into HTML, and the <title> of the webpage is extracted using BeautifulSoup.
   * These titles are checked for duplication and against the keywords before being stored in a text file (headline.txt).
   * The file writing is handled iteratively: each valid and non-repetitive headline is appended to the file.

