A project on Data Analytics – Aug 2023

Python procedural steps, Databases, Query Tables from COVID-19 API -Venu Gopal Madhav Annamdas

# Project Overview:

# Description: To adopt public API to investigate covid-19 information related to specific countries and the world.

# Purpose: To retrieve information about deaths due to covid-19 related to countries and show them on the world map.

# Tools & technologies:

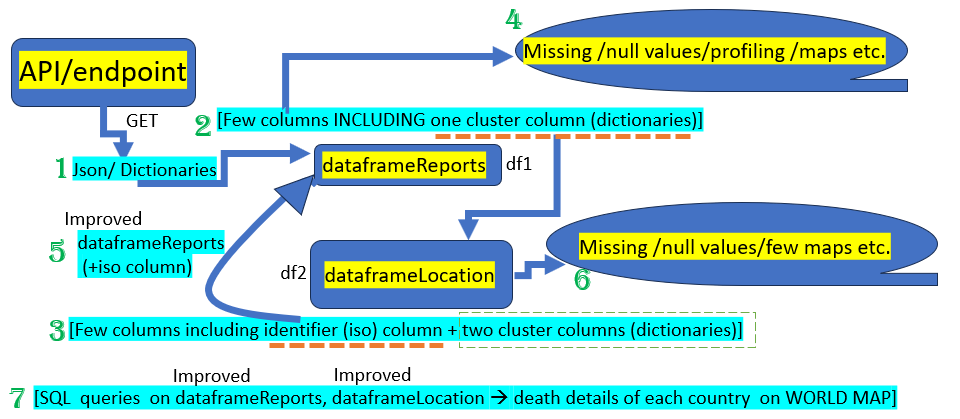
# Programming language: Python

# libraries/packages: requests, NumPy, Pandas, Matplotlib, Seaborn, Folium, Datetime, sqlite3, tabulate, pandas-SQL.

# Challenges:

# First rapid API provided limited data not sufficient for the required exploration. So, I adopted second public and key free API. The data were clustered into dictionaries, the (a) date of death, and (b) updated dates of death are not provided. The data includes confirmed cases, deaths, recovered, fatality rates, country names and country identifiers. Inconsistencies/ were handled by creating a new data frame. Thus, two data frames formed the basis for this mini-project

# Flow sequence

**

*Python procedural steps in Jypter Notebook: (file name: DataScience.ipynb)*

Program Description: Covid19 Endemic details of each country and continent in the world from a couple of public APIs (API 1 : URL <https://covid-193.p.rapidapi.com/statistics> (Rapid API). Maps and comparison plots

# # Import Libraries



|  |
| --- |
| **Data Retrieval:**  - Use Python and its libraries (e.g., requests) to build a dataset using a public API.  - Example: You can use this api : <https://covid-api.com/api/> |

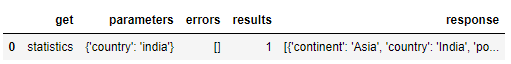
# Accessing Rapid API - COVID-19 data



The output: data is in dictionary form

# # Dataframe (df) from the dictionary (data)

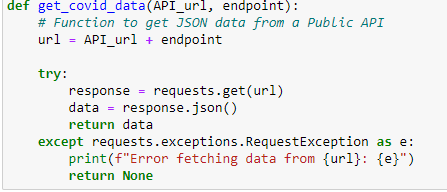


The dataframe (output) accesses country-by-country info on covid-19 cases. The data is smaller than I required. 

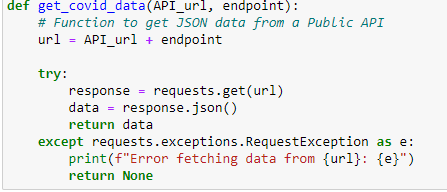
Thus, I moved to another API location where I can get more information for this data analytics project.

(API 2 : URL <https://covid-api.com/api/> (Key free Public API)

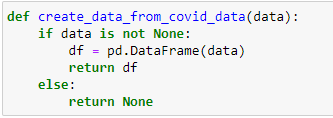
# # Function to get json data from Public API

.

# # 1. Function to get json data from Public API

.

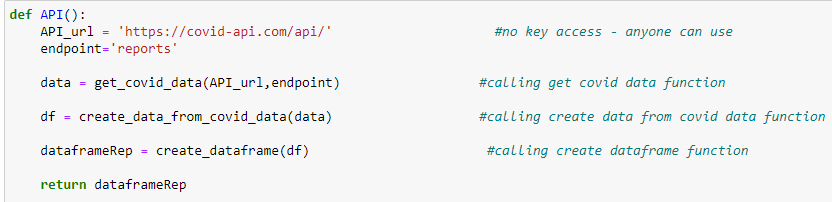
# # 2.Function to get single column data dictionary

.

# # 3.Data dictionary conversion to multi column dataframe

.

# # Call to use the above 3 functions (1., 2., 3)

.

These calls result in one dataframe as below.

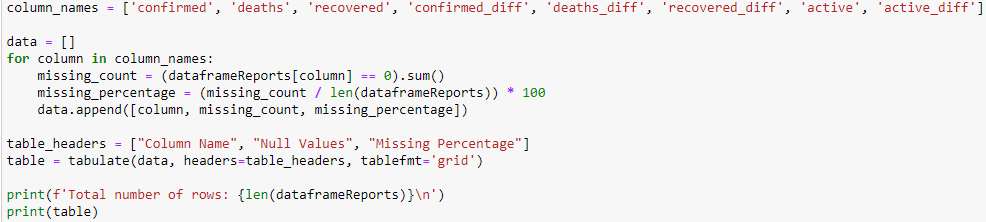


|  |
| --- |
| **Data Preprocessing of ‘dataframeReports’:**  - Generate the summary on missing values  - Document any inconsistencies and potential outliers  - Clean and preprocess the retrieved data to handle missing values, inconsistencies and potential outliers. |

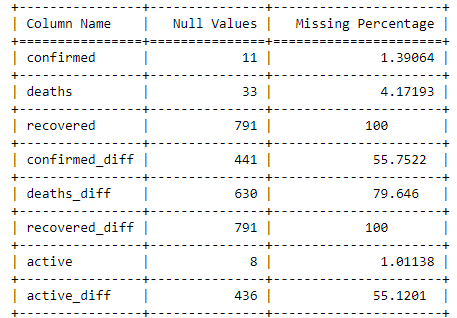
Steps used to study dataframeReports are

1. dataframeReports.head(5)
2. dataframeReports.tail(5)
3. dataframeReports.shape
4. dataframeReports.dtypes
5. dataframeReports.describe()

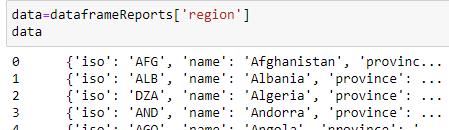
*it was observed that the data inside dataframeReports contains 9 columns with date objects, integer and float objects. It also contains dictionaries in one of the columns (‘region).*  



The following table summarises the ZERO values in all the columns of dataframeReports.



# # outliers : column 'region' is filled with dictionaries as shown below



As data is related to human loss, active covid cases etc, there is no specific outlier except clustered data inside the ‘region’ column. This column is converted to a new dataframe (named dataframeLocation) and removed from dataframeReports.

# # Code to covert the column 'region' contents to another dataframe



This contains new columns as shown below (where ‘iso’ is the identifier column)



|  |
| --- |
| **Data Preprocessing of ‘dataframeLocation’:**  - Generate the summary of missing values  - Document any inconsistencies and potential outliers  - Clean and preprocess the retrieved data to handle missing values, inconsistencies and potential outliers. |

Steps used to study dataframeLocation are

1. dataframeLocation.info()
2. dataframeLocation.dtypes()
3. dataframeLocation.value\_counts
4. dataframeLocation.describe()

in this investigation, I found that columns ‘cities’ and ‘province’ are clustered. I dropped them as I don’t want to create more dataframes. I used the following codes to find values of columns ‘cities’, ‘province’ and drop them.







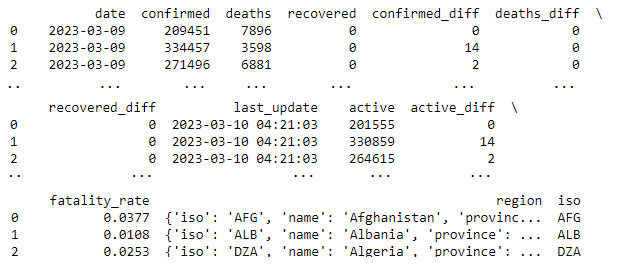


|  |
| --- |
| **Data preprocessing to handle inconsistencies for SQL requirements:**  - In general, the data frames need one unique/primary identifier (key)  -for dataframeLocation, let us make ‘iso’ column, as a primary key.  -for dataframeReports, let us add ‘iso’ column so that both can have the same primary keys for simplicity.  -I dropped ‘region’ in dataframeReports, which is clustered (note: we already used it to make dataframeLocation). So two dataframes are considered for analysis. see the following steps for details. |

# # Code to add ‘iso’ column from dataframeLocation to dataframeReports



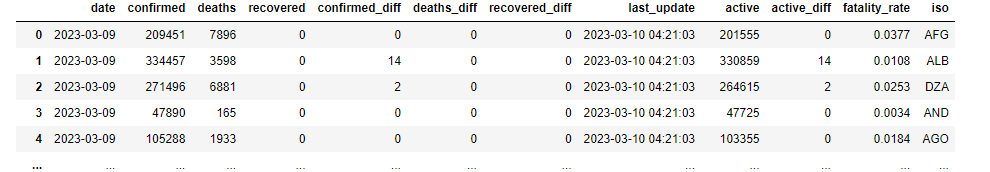
Output:



# Code to drop the column 'region'



Output:

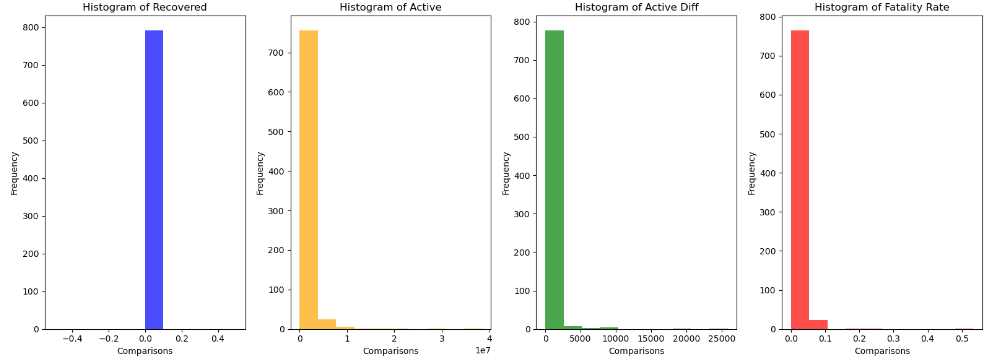


Now we have TWO data frames (1) dataframeReports and (2) dataframeLocation for analysis.

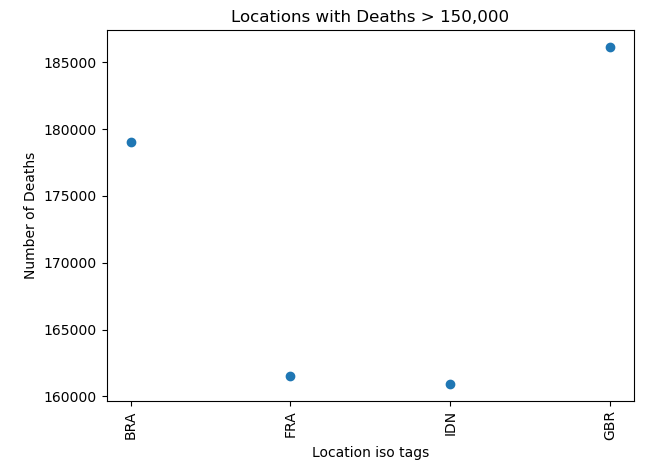
Graphical investigation**:**

# # Let us look at some chats of dataframeReports

1. histogram plots of some columns



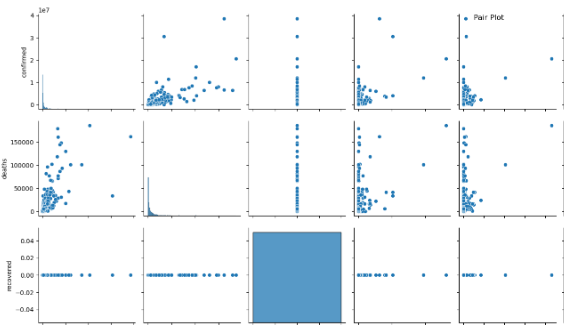
2. scatter plots showing deaths > 150,000 in countries COMPARED with ‘iso’



(Brazil, France, Indonesia, Great Britain)

# #pair plot investigation of dataframeReports

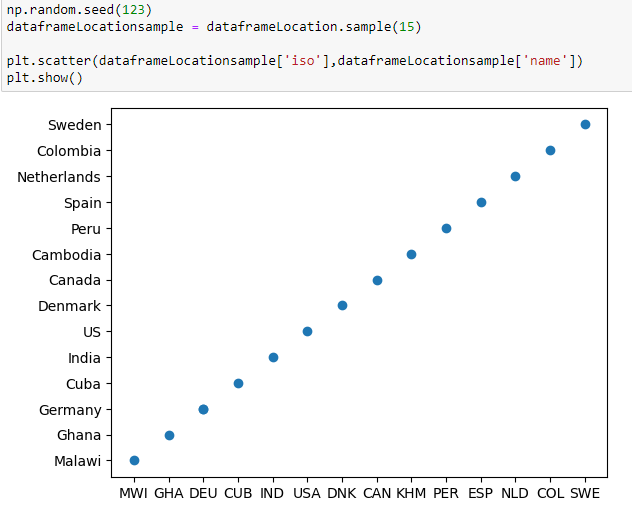




(Sample pair plots)

the horizontal and vertical blue lines indicate ZEROS of one of the column. Mostly the values are concentrated near the origin or one of the columns.

# #look at some chats of dataframeLocation



# let us further investigate dataframeReports using profiling (we dont investigate dataframeLocation as actual data of covid-19 is inside dataframeReports)

|  |
| --- |
| '''  as my jupyter notbook is unable to install profiling or sweetviz packages,  i will transfer dataframeReports to csv file and store in GITHUB,  which can be used in colab to do profiling of the dataframe  ''' |

# #code to convert dataframeReports to csv



|  |
| --- |
| Note: ‘forprofilingpurpose.csv’ is in the folder, it will be placed in the repository (github) |

PROFILING REPORT:

link to find information on dataframeReports

https://github.com/venuannamdas/DataAnalyticsProjectTrial1/blob/ver1/DataAnlytics\_profiling\_dtaframeReports.ipynb

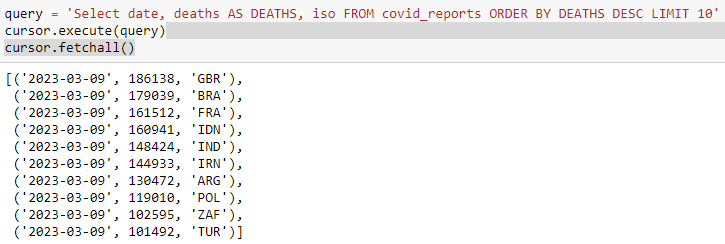
note: please use the above link and run in colab to get the REPORT.

|  |
| --- |
| **SQL:**  - SQL Database Integration:  - Set up a SQL database (e.g., SQLite or MySQL) using Python and store the preprocessed data into the database.. |

A database is created [DataAnalytics2023.db"].

A table ‘covid\_reports’ for dataframeReports is created in the database, and the following queries are executed.

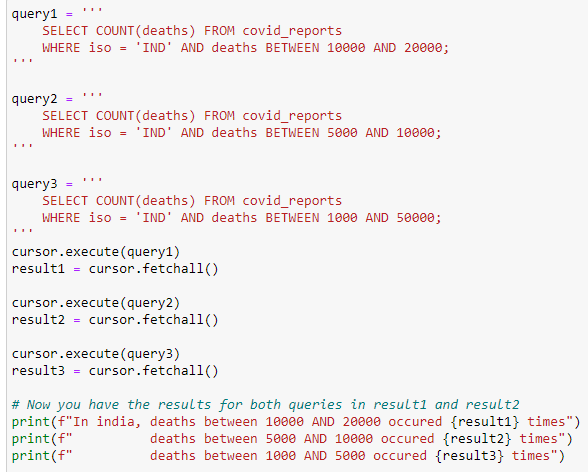




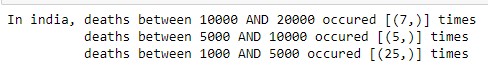
The ‘date’ column is fixed at 2023-03-09, this is the intrinsic date used in the API.

However, several interesting SQL queries are executed. Which are self-explanatory in the python code.

A few interesting investigations like the following are performed to get details of country’s iso code ‘IND’. For example, to know the occurrences of the number of deaths between a certain range are executed.

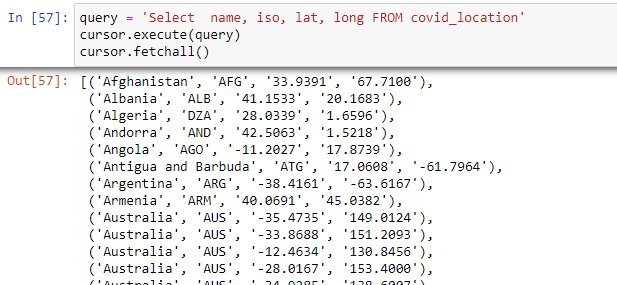


Output:

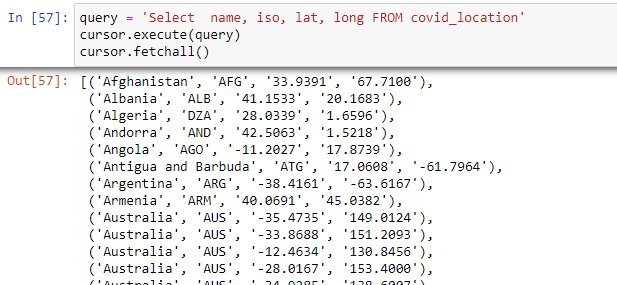


Another table ‘covid\_location’ is created for databaseLocation is created, and the following queries are executed.





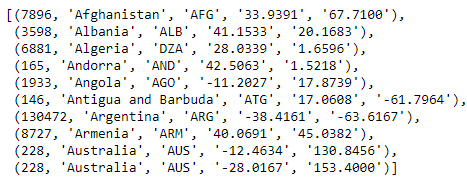
Output:



SQL query to join as shown below

|  |
| --- |
| query = 'Select covid\_reports.deaths, covid\_location.name, covid\_location.iso, covid\_location.lat, covid\_location.long FROM covid\_reports join covid\_location ON covid\_reports.iso=covid\_location.iso LIMIT 10'  cursor.execute(query)  cursor.fetchall() |

Output:



Finally, we can query one country and send it to the following code to get the country details on the map.

|  |
| --- |
| # Execute the SQL query to fetch location data  query = "SELECT covid\_location.name AS COUNTRY\_NAME, covid\_location.lat AS LATITUDE, covid\_location.long AS LONGITUDE, covid\_reports.deaths AS DEATHS FROM covid\_location JOIN covid\_reports ON covid\_location.iso = covid\_reports.iso LIMIT 1"  cursor.execute(query)  data = cursor.fetchone()  if data:  country\_name = data[0]  latitude = data[1]  longitude = data[2]  deaths = data[3]  # Create the map with the fetched location data  m = folium.Map(location=[latitude, longitude], zoom\_start=3, tiles="cartodb positron")  # Create an HTML string for the popup content  popup\_content = f"""  Location: {country\_name},  Latitude: {latitude},  Longitude: {longitude},  Deaths: {deaths}  """  # Add a red marker at the specific location with popup information  marker = folium.Marker(  location=[latitude, longitude],  icon=folium.Icon(color="red"),  popup=folium.Popup(html=popup\_content, parse\_html=True)  )  marker.add\_to(m)  # Save the map as an HTML file  m.save("map\_with\_marker.html")  else:  print("No data found") |

Map : located in github

