ANALYZING **COVID19** DATASETS BY USING THE CORRELATION COEFFICIENT FACTOR

Introduction

This project tries to analyze **COVID19** virus spread data around the globe and the US separately. It seeks to find some meaningful relationships between different regions' timely trends. By this measure, compare them to each other and find out how their behavior is different or similar to the other areas. Also, by finding similar behavioral trends predict the future behavior of the regions that the virus spread starts later. Feel free to use the application for the research purpose, and if it is useful, please reference this repository as the source.

The development environment

This application is developed based on [**Microsoft window 10**](https://www.microsoft.com/en-us/windows/get-windows-10) x64 base processor and [**python 3.7.5**](https://www.python.org/downloads/release/python-375/). It uses libraries such as [os, DateTime, sys, numpy, matplotlib.pyplot, datetime, matplotlib.dates, minepy, and geopandas.](https://www.python.org/downloads/release/python-375/) The ***matplotlib*** is the main library for viewing the figures and plots, as well as the geographical mapping. The geographic map implementation has been done through a powerful geo data frame tools well implemented in [**geopandas**](https://geopandas.org/gallery/cartopy_convert.html#sphx-glr-gallery-cartopy-convert-py). The library has several functions and methods for building, importing, and working with different geospatial data and especially standard shapefiles. It easily coordinated with the **matplot** library for viewing purposes. It is as easy as calling the **matplotlib** plot method with the **geodata** frame. It displays the maps with different properties and a wide variety of the coloring well enough for the project needs. Geopandas ver. 0.6.1 use for this project. Thanks for a good example and implementation of [Geraint Ian Palmer](http://www.geraintianpalmer.org.uk/2017/09/22/plotting-geopandas/). It is convenient.

External data sources

For the geographical data implementation, two sources of data used:

* The world country border as shapefile imported from [“Made with Natural Earth. Free vector and raster map data @ naturalearthdata.com”](https://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m-admin-0-countries/). It must be copied in a directory named *gis* for furthermore references.

shapefile=r'gis\ne\_110m\_admin\_0\_countries.shp'

* The states and the other territories border shapefile of the US have imported from the [US Census Bureau, Department of Commerce](https://catalog.data.gov/dataset/tiger-line-shapefile-2017-nation-u-s-current-state-and-equivalent-national). It must copied in a directory *gis*

shapefile\_states =r'gis\tl\_2017\_us\_state.shp'

* the COVID-19 time series data directly load from the source file in *CSV* format form a [Github repository maintained and updated daily by the Johns Hopkins University Center for Systems Science and Engineering (JHU CSSE). Also, Supported by ESRI Living Atlas Team and the Johns Hopkins University Applied Physics Lab (JHU APL).](https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series)

Four different files daily update which is used by this application:

* Daily global confirmed cases:

url\_confirmed\_global = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/csse\_covid\_19\_time\_series/time\_series\_covid19\_confirmed\_global.csv'

* Daily global death confirmed cases:

url\_death\_global = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/csse\_covid\_19\_time\_series/time\_series\_covid19\_deaths\_global.csv'

* Daily US death cases:

url\_death\_US ='https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/csse\_covid\_19\_time\_series/time\_series\_covid19\_deaths\_US.csv'

* Daily the US territory confirmed cases:

url\_confirmed\_US ='https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/csse\_covid\_19\_time\_series/time\_series\_covid19\_confirmed\_US.csv'

All the data are in a comma-separated ***CSV*** format and updated daily around midnight (UTC). Every row in those tables represents a territory, and each day a new data column is added. The data series are in cumulative format. It means for every new day data is the summation of all the previous days’ data.

There is also another reference table in this repository. It represents the coding of geographical notions. The coding for the names of countries and states are listed up there. However, matching the right geographical position with names is no easy task. Some names are not matched with any popular databases at all. The [ISO3](https://unstats.un.org/unsd/tradekb/knowledgebase/country-code) coding is the best one, but for subcategories, there is the lake of the matching. Also, for some reason, there is an unmatched **ISO3** code for small countries and territories. It needs to improve this part of the data. This application mainly uses **ISO3** for matching purposes. Some small unmatched ones and errors are updated in the application internally.

* Lookup table :

Look = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/UID\_ISO\_FIPS\_LookUp\_Table.csv'

Application modules

This application consist of three python files that must be copied into root directory:

* ***Trend\_Similarity.py***
* ***HeatMap.py***
* ***input1.py***
* ***Multi\_variance\_ts.py***
* ***Multivariate\_LSTM.py***

The ***input1.py*** is the functions library that is used by the other programs.

Trend\_Similarity.py

This is the main part of the application. It preprocesses the input data and represents some basic analysis based on the user input. It also builds a matrix of similarity for each region for the selected dataset and saves it in the ***CORR*** directory for later references. Here describe in more depth the application.

The program gets some basic information from the user by calling the function ***input\_dat()***:

Consider,dataset\_type,flatten,similarity,NAME,Y,last\_date,first\_date =input\_data()

It has done by calling above function that is in input1.py and read data from the data sources and return some basic variable to the main program as below:

***dataset\_type*:** by asking the user to select a dataset type and return it. This can be a ***dataset of Global confirmed cases, Global Death cases, US confirmed cases, or US Death cases*.**

***flatten*:** by asking the user to select a flattening factor to use for calculating the moving average. It is between 1 and 7.

***similarity*:** by asking the user to select a percentage factor between 0-1. It is a threshold parameter that will use for finding the most similar region with this percentage and higher. The higher number near 1 is mean more similar ones.

***consider*:** after showing the list of the region in data set to the user, asks for selecting a region by rank to show the result about this region. This is a number that represents the row in data set for that region

***NAME*:** is the list of the regions name extracted from the dataset.

***last\_date,first\_date***: they are the first and the last date of that there is data in between them in data set.

***dataset:***the output dataset accordingly whom it reads from the sources and preprocesses. The data format is something like this:

|  |
| --- |
|  |

Every row represents one region in that category. Because the data record is in the cumulative form, it must change to daily cases first. As an example, let take a look at the one record.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | 4/7/2020 | 4/8/2020 | 4/9/2020 | 4/10/2020 | 4/11/2020 | 4/12/2020 | 4/13/2020 | 4/14/2020 | 4/15/2020 |
| Country X | 1 | 1 | 39 | 96 | 139 | 170 | 199 | 215 | 215 |

It is in cumulative form and must be changed to daily new cases format

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | 4/7/2020 | 4/8/2020 | 4/9/2020 | 4/10/2020 | 4/11/2020 | 4/12/2020 | 4/13/2020 | 4/14/2020 | 4/15/2020 |
| Country X | 1 | 0 | 2 | 93 | 43 | 131 | 700 | 60 | 2 |

After changing to a daily format, it needs to make it flatten to have a smoother dataset. The ***flatten*** parameter to use for calculating the **Moving Average** as a flattening method. The data is discrete, and the source of data in different regions are collecting them by various tools. It is logical to use the moving average as a tool for helping time series be more realistic in terms of real daily cases measure. A moving average of up to a week can be practical. A coefficient between 1 and 7 asked from the user and calculated the moving average based on it. For example, use the flatten factor of two and will have:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | 4/7/2020 | 4/8/2020 | 4/9/2020 | 4/10/2020 | 4/11/2020 | 4/12/2020 | 4/13/2020 | 4/14/2020 | 4/15/2020 |
| Country X | 0 | 0.5 | 1 | 47.5 | 68 | 87 | 415.5 | 380 | 31 |

For comparing two datasets, it needs them to be normalized. For the datasets of daily new cases, a division to a maximum of the dataset for all items makes a new dataset with maximum and minimum between zero and one.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | 4/7/2020 | 4/8/2020 | 4/9/2020 | 4/10/2020 | 4/11/2020 | 4/12/2020 | 4/13/2020 | 4/14/2020 | 4/15/2020 |
| Country X | 0.000 | 0.001 | 0.001 | 0.049 | 0.070 | 0.090 | 0.428 | 0.392 | 0.032 |

For adjusting the trend curve of the data sets, it needed to delete the trial zeroes from starting and ending .there is a need to ignore tiny numbers by a threshold limit. This threshold set by default in the program to 0.01 as ***epsilon***. The remaining data after enforcing the trimming will be:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | 4/10/2020 | 4/11/2020 | 4/12/2020 | 4/13/2020 | 4/14/2020 | 4/15/2020 |
| Country X | 0.049 | 0.070 | 0.090 | 0.428 | 0.392 | 0.032 |

Here some factors define according to the behavior of the dataset curve.

***Complete\_form***: it is 0 if the curve does not pass its maximum bell curve format in about the middle of the time length. If the maximum value of the data series is after 30% and before 80% of the time length of the data series, then ***Complete\_form*** will be equal to one. It means that the data series already passed the peak. If it has the above situation and the last data in the data series is less than 20% of the maximum value, it shows that it is in the near end of the bell curve format and this ***complete\_form*** set to 2. This curve, with a good probability, has passed the peak and is in the final stage of maturity.

***Precious*:** There is another measure that set to 1 if the total dataset amount of cases is more than 1% of the total amount of all datasets series. It means it has a precious amount of data to be concerned. The datasets with PRECIOUS data can be used for comparison. Then only the datasets with the ***precious*** equal to one use.

In the next step, a ***Correlation Coefficient Factor Matrix*** calculated for all regions in the dataset. This matrix uses later for finding the similarity between two regions in terms of the correlation coefficient factor. This matrix has an asymmetric format. For calculating this factor, depending on the length of the two data set, the second one trimmed or filled with zero to make it in equal length with the first dataset. This concept explains in detail in the project definition.

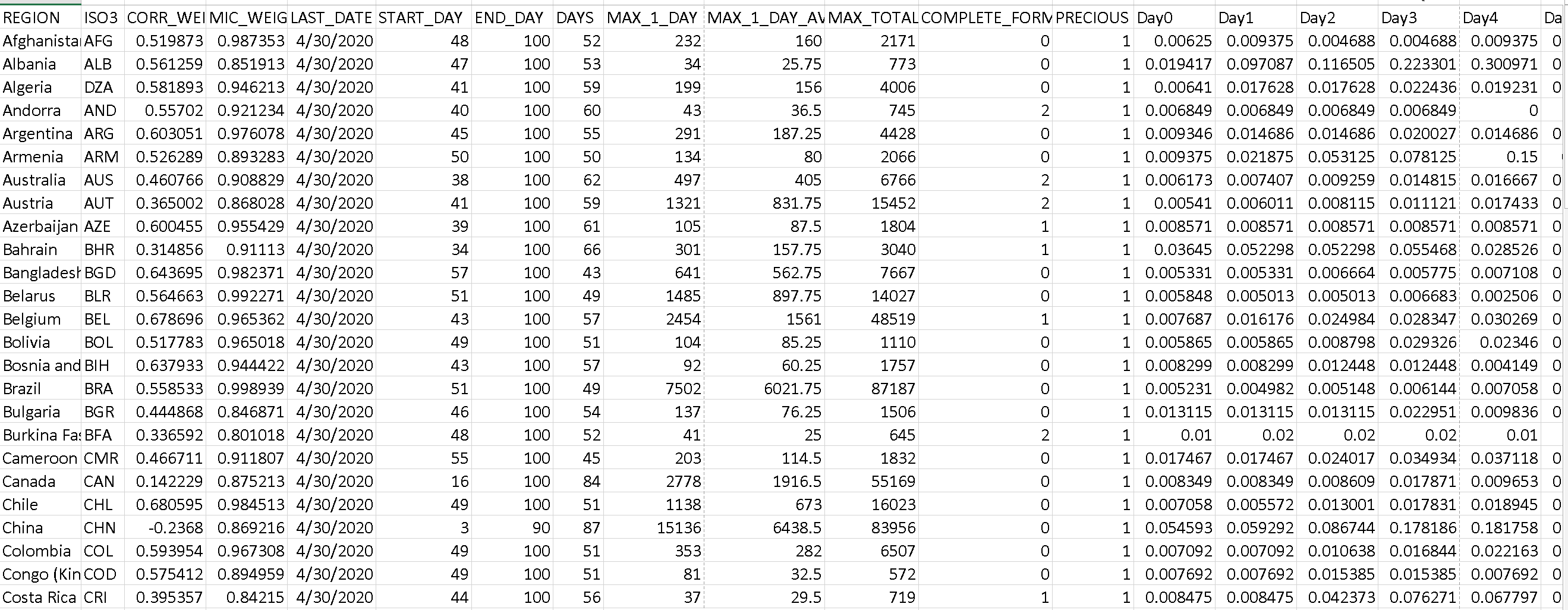
Next part, a weighted average for columns and rows in the correlation coefficient matrix is calculated.

*list\_av\_to,list\_av\_from = average\_corr(sd\_matrix,NAME,dataset\_type,list\_max1,last\_date)*

***average\_corr***: This function is in ***input1.py*** library as well as other functions, and also save this two item for every region with the last date of the data in which generated accordingly to a ***CSV*** file in director named ***CCORR***. The file name picks from the name list according to the ***dataset\_type****. Also, two more* ***CVS*** *files created in the same director. The first one is the correlation matrix for precious regions with an extension of* ***\_SD,*** *and the other one is a maximal information coefficient (MIC)* for the same regions with the extension of **\_*MIC***. This recent one will explain more later.

*dataset\_typem=['CCORR/US\_DEATH.csv','CCORR/US\_CONFIRMED.csv','CCORR/GLOBAL\_DEATH.csv','CCORR/GLOBAL\_CONFRIRMED.csv']*

This file is the core data for further presentation and analysis. The output format is like this:



This ***CSV*** file can be used for any further research. The ***CORR\_WEIGHTED*** column is the weighted average of each row in the correlation coefficient factor matrix, and the **MIC\_WEIGHTED** is the weighted average of [maximal information coefficient (MIC).](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3325791/) This is a measure between 0 and 1 and shows if the two data set has any nonlinear relation to each other. The concept of it is a little complex but easy to use. I used **MINE** class form ***minepy*** package. And calculate a MIC matrix for the whole datasets like the Correlation matrix. It shows for the data sets which do not have linear relation according to their low or zero value of correlation coefficient factor; they have a nonlinear relationship with a significant value of MIC. For example, China, with an average **of -.23 of correlation factor**, has a **.86 MIC** factor. It means the china dataset does not have or even a negative correlation with other countries' data. However, with MIC .86, it has a robust nonlinear relationship with others. The **LAST\_DATE** column is the date the last data is available in the input dataset. Also **COMPLETE\_FORM,** and **PRECIOUS** columns are the same amount calculated for each region according to the parameter ***complete\_form*** and ***precious.***

From here, the program uses the user-selected region for studying. It is the region in row ***consider***.

Then, it tries to find the regions that have ***CORR\_WEIGHTED***factor higher than the ***similarity*** factor if there is any. Of course, the regions which have less length than the *consider* region not listed as they do not have enough dataset width to be useful for prediction for the ***consider*** region. After finding proper regions, it makes a list of them and by showing to the user asks for selecting one for further processing. If nothing found requesting the user to start again with a lower ***similarity*** factor.

Then, based on these data, two figures present to the user. This figure is plotted through using ***matplotlib*** library function ***plot***. ***Fig1*** showing the trend of the *consider* region beside who is similar to that. This figure aligns all data series with being started together at their day one. It means the region's data can be compared to each other based on their trends. A sample figure of ***Fig1*** shows here.

|  |
| --- |
|  |

***Fig2*** presents the actual trend of the ***consider*** region in real-time from the starting date of the first case beside a regression of the same region based on the fact that if it follows the regions who selected from most similar ones in the last part by the user. This figure shows a prediction according to the data in different regions and tries to show what will happen to this considered region in the future compare to the other similar trend to it. A sample for this figure is as follows.

|  |
| --- |
|  |

It saves the figures in a directory name *fig* in the root directory in .***png*** format for further more use as well.

HeatMap.py

The ***HeatMap.py*** is mainly a reporting program. It asks the user for selecting a category data set from the list of ***Global confirmed cases, Global Death cases, US confirmed cases, or US Death cases***. After that, based on this input, it read the ***CSV*** corresponding correlation coefficient database form the ***CCORR*** directory. This file has to be already created by the main program, ***Trend\_Similarity.py,*** as described. Ultimately, the user must run the ***Trend\_Similarity.py*** for every category in advance if it is needed to use in ***HeatMap.py***. In other words, for the ***HeatMap.py*** to work properly, it has to run the ***Trend\_Similarity.py*** at least one time for the specific category before that. The ***HeatMap.py*** exit and ask the user to do that or show outdated data from previous data processing. Do not forget the correlation coefficient factor matrix is build based on the up-to-date data and also the flattening factor which entered in the ***Trend\_Similarity.py***. Flattening factor affects the output of the matrix then must be recalculated for a new flattening factor each time. The flattening factor has a key role in smoothing the curves and improve the task of finding out the similarity between the trends. Then, do not forget to start from ***Trend\_Similarity.py*** in advance. The correlation coefficient factor matrix reads to a dataset by calling a function according to the dataset type that already selected by the user.

dataset = read\_csv\_corr(inp)

The ***HeatMap.py*** works with the map for presenting geographical data. There are two map data set that use for the world and the US states. The source data are in .***shp*** file format. Thanks to the powerful geo library of the python. It is *geopandas*, an easy to use, and full of the methods and functions for working with geospatial data. This program uses it for viewing a heat with the similarity parameters of them. The map shapefile read from the sources files in ***gis*** director accordingly. The files already downloaded from the sources and saved in this location.

if inp<2:

world1= gpd.read\_file(shapefile\_states)

else:

world1 = gpd.read\_file(shapefile)

For a better presentation of the maps, some edits have been done on the map data. For the world, the 'Antarctica' has been deleted from the world map. Also, on the US map, some territories deleted for better viewing. However, the data in the tables are not touched. The dataset, which has read, sorted based on the ***CORR\_WEIGHTED*** columns. This parameter used to determine how a region data behaves similarly to others. For adjusting the data with relevant regions shapefile, use mainly the **ISO3** code for the world and the state's name for the US. There are small missing in those that are fixed by some correction! By merging two data sets, a new one consisting of the data of the ***CORR\_WEIGHTED*** and geodata is created. Next by using plot form ***matplotlib*** three bar chart relatively to the 15th least, and 15th highest similarity factor show in two different figures with a 3rd ones show them all in a bar chart together. The final figure is a heat map of the whole region in data sets. Accordingly, it is the world or the US. For showing regions without data, a black map first plot and after that, a colored one of the regions with data is the plot on that. The heat map plot is a powerful method of combining the plot form matplotlib and pandas data frame.

mix.plot(ax=ax, column='SIMILAR\_WEIGHTED', cmap=colormap, \

vmin=min(mix.SIMILAR\_WEIGHTED), vmax=max(mix.SIMILAR\_WEIGHTED), \

legend=True , legend\_kwds={'label': \

"SIMILARITY BY COLOR lef color less similarity. right colors is more similar trends to others", \

'orientation': "horizontal"})

This implemented easily by one plot call. There several attributes and the parameters that can enhance the viewing. But for this research, this one is enough. The output files are shown here by some samples ones.

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |
|  |

The output of the 3 bar chart figures and the heat map also save as ***.png*** file in fig director. The naming is categorized by the dataset type and the last date of the data together with a proper name:

fig1.savefig("fig/%s\_%s\_Most\_similar.png"%(datatype[inp],last\_date),dpi=300)

fig2.savefig("fig/%s\_%s\_Least\_similar.png"%(datatype[inp],last\_date),dpi=300)

fig3.savefig("fig/%s\_%s\_similarity\_trend.png"%(datatype[inp],last\_date),dpi=300)

fig.savefig("fig/%s\_%s\_Heat\_Map.png"%(datatype[inp],last\_date),dpi=300)

Director structure for the program

For the proper execution of the program, the structure is simple. Just set a root directory as a working directory for python and make the following structure in there.

Root Directory

├── input1.py

├── Trend\_Similarity.py

├── HeatMap.py

├── gis

│   ├── ne\_110m\_admin\_0\_countries.cpg

│   ├── ne\_110m\_admin\_0\_countries.dbf

│   ├── ne\_110m\_admin\_0\_countries.prj

│   ├── ne\_110m\_admin\_0\_countries.README.html

│   ├── ne\_110m\_admin\_0\_countries.shp

│   ├── ne\_110m\_admin\_0\_countries.shx

│   ├── ne\_110m\_admin\_0\_countries.VERSION.txt

│   ├── tl\_2017\_us\_state.dbf

│   ├── tl\_2017\_us\_state.shp

│   └── tl\_2017\_us\_state.shx

This the only directory structure that is needed for running the program. During the execution of the program, two new directories also make automatically for saving the figures in the **fig** directory and saving the correlation coefficient factor matrix in the **CORR** directory. The user can use the files in these two directories for reporting or further research.

# Multi\_variance\_ts.py

This program uses the multivariate time series analysis method for predicting the future data value of them. the main library that is used for this purpose is:

**from statsmodels.tsa.vector\_ar.var\_model import VAR**

this library is a for statistical modeling purpose: (<https://www.statsmodels.org/stable/generated/statsmodels.tsa.vector_ar.var_model.VAR.html>)

the detail of the methodology and parameters explained briefly on their website.

In the program, first, read the data files to the data frame and change their format to proper for modeling. After that, for making the datasets stationary, use the differentiation methods. This has been done through ***dif\_m(input\_m).*** It gets a matrix as an input a calculate the differentiated of the columns data and return a matrix as a result. Through it, all datasets differentiate up to level 7 and save to a tensor, respectively d[0] to d[6]. By calling ***adf\_test(d\_m[j,:,i],Alpha)*** a statistical test has been done on each level of differentiation and save respectively. Then It asked the user to choose a level of differentiation with the lowest number of un stationary datasets. The best selection is the lowest number of differentiation and the number of un stationary datasets. After the selection of the targeted differentiated dataset, the nonstationary datasets delete fro that and prepper for the next step. With function, weights() the weights for the scoring the result calculated based on the population of each region. This scoring model adapted from the competition in Kaggle(<https://www.kaggle.com/c/covid19-global-forecasting-week-5/overview/evaluation>.) VAR model takes a lag parameter that has a very important role in prediction accuracy. For finding the best lag parameter through a loop, all the possible lag check against the dataset—the dataset divide to a train and test set. The test set is the last seven days of data. The prediction for those seven days compare to the real test set and calculate for every step the RMSE and score. The best lag based on this scoring select for the final prediction model. Moreover, the error calculates for the last seven days. In the next part, the best lag and full dataset use for predicting a user input number of days in the future. And the prediction fo a focused region with the lag error graph view to the user.

# Multivariate\_LSTM.py

This program uses CNN-LSTM modeling for predicting the future values for the datasets. Thank Jason Brownlee, Ph.D., for a very brief and straight forward explanation of the model implementation. <https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/>

Based on the modeling theory, the main part is preparing the time-series for the modeling in this format. The model needs to set a part of the dataset sequence as the property of each step. For a short description look at univariate dataset as this:

[2,7,9,14,13,7,9]

For predicting the next value of the above time series the datasets must form as two dimensional as below:

[[2,7,9],

[7,9,14],

[9,14,13],

[14,13,7],

[13,7,9]]

This means a lag of the passed data used for every current data. It means, for example, 2,7 gives 9. This methods gives the opportunity of using the series for forecasting the future data. there are several options for lag and future data prediction.

**series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True)** function used for preparing the data as above. The **n\_in, n\_out** used to say how many passed data (**n\_in**) used for some further data (**n\_out**). Finding the best value for every timeseries must check these two parameters for the best output value. **dropnan** parameter used for droping the nan values. After this preparation, a train, test divided has been done for error testing. Finally, a sequential model builds on an LSTM model. The Keras library used for this purpose.

**model = Sequential()**

**model.add(LSTM(256, activation='relu', \**

**input\_shape=(train\_X.shape[1], train\_X.shape[2])))**

**model.add(RepeatVector(n\_out))**

**model.add(LSTM(128, activation='relu', return\_sequences=False))**

**model.add(Dense(n\_region))**

**model.compile(optimizer='adam', loss='mse')**

this sequence gives the best performance. The model compiles and runs with 100 epochs.

**history=model.fit(X, y, epochs=100, verbose=2,validation\_data=(test\_X, test\_y), shuffle=False)**

finally, the error\_loss and the score calculated for this model. It gives a good accuracy of about 95%.the output of the model also visualizes for the user.

After all, the best prediction option is the statistical model VAR with more than 97% accuracy, and, the CNN-LSTM. More sophisticated models can use for this purpose, but the simple ones give an excellent performance.