Expansion

Zachary Levine

08/04/2021

"Mathematical Models for Assessing Vaccination Scenarios in Several Provinces in Indonesia N. Nuraini, K. Khairudin, P. Hadisoemarto, H. Susanto, A. Hasan, and N. Sumarti"

Expanding old gradient functions from replication assignment to incorporate awarness.

This next few code chunks provide a nice framework for plotting the CN covid data by town, with plenty of options. meant for checking spacial heterogeniety condition.

```
#This chunk handles python imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
ByTown = pd.read_csv("ConCovidTowns.csv")
```

This is the main code for plotting.

```
#read in the covid data organized by town.

#These are graph settings for Seaborn. Can ignore if trying to understand code --- they mask matplotlib
rc={'lines.linewidth': 4, 'axes.labelsize': 20, 'axes.titlesize': 18}
sns.set(rc=rc)
sns.set_style("whitegrid", {'axes.grid' : False}) #remove grey grid
sns.set_context("paper") #changes the theme
plt.rcParams['figure.figsize'] = [16, 7] #changes the size


#Function for separating out towns.
def getTown(df: pd.DataFrame, n: int) -> pd.DataFrame:
    '''
    Give the dataframe and a town number.
    Returns all columns for that town.
    '''
    #Rename Date Column
    df.rename(columns = {'Last update date':'Date'}, inplace = True)
    #Make date a proper datetime object
    df['Date'] = pd.to_datetime(df.Date)
    #Sort by Date
    df.sort_values(by='Date')
    #Reset Index
    #Grab all elements of big array with this town number
    j = df.loc[df["Town number"] == n]
    #Reset index now that other towns are gone
    j.reset_index(drop=True, inplace=True)
    return(j)
```

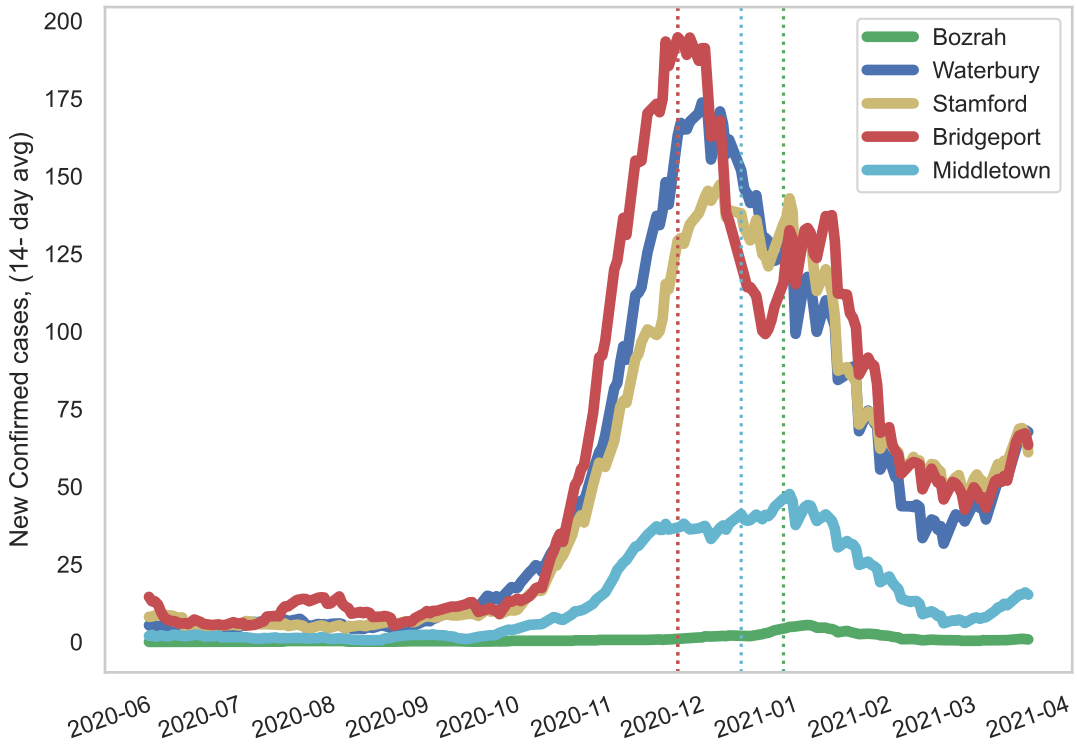
```
def getPeak(df:pd.DataFrame, n:int, metric:str) -> pd.DataFrame:
    '''
    given original dataframe df, town number n, and metric str,
    return the day at which the most NEW (metric) were recorded.
    For plotting lines on graphs :D
    '''
    maximum = getTown(ByTown, n)[metric].diff().max()
    q = getTown(ByTown, n).loc[getTown(ByTown, n)[metric].diff() == maximum]
    q.reset_index(drop=True, inplace=True)

    #Return the first time the max comes up --- one or two towns reported the
    #same max cases twice, always within a week. So this removes the ambiguity.

    return(q.head(1))


def getFullPeakDF(df:pd.DataFrame, metric:str) -> pd.DataFrame:
    '''
    Given original dataframe df, and a metric (eg Confirmed cases), return a
    dataframe containing the row in which the most new people were registered
    as having [metric] eg, testing postitive for Confirmed cases.
    '''
    MaxDF = getPeak(ByTown, 1, metric)
    for i in range(2,ByTown["Town number"].max()):
        MaxDF = pd.concat([MaxDF, getPeak(ByTown, i, metric)])

    MaxDF['Date'] = pd.to_datetime(MaxDF.Date)
    MaxDF = MaxDF.sort_values(by='Date')
    MaxDF.reset_index(drop=True, inplace=True)

    return(MaxDF)



def plotTownCases(df: pd.DataFrame,
                  n:int,
                  metric: str,
                  col:str,
                  bins:int = 14,
                  maxcaseline = False,
                  IncludeLegend = True):
    '''
    Give df(big dataframe) and town number (n) and return a plot
    of rolling average over (bins) days for daily change in column (metric)
    '''

    UnC = getTown(ByTown, n)[metric].diff()

    plt.ylabel("New {}, ({}- day avg)".format(metric, bins))
    plt.xticks(rotation=19)
    if IncludeLegend == True:
```

```
        plt.plot(getTown(ByTown, n)["Date"],
         UnC.rolling(bins).mean(),
         label = getTown(ByTown, n)["Town"][0],
         color = col,linewidth= 4)
        plt.legend()
    else:
        plt.plot(getTown(ByTown, n)["Date"],
         UnC.rolling(bins).mean(),
         color = col)


    if maxcaseline == True:
        plt.axvline(x=getPeak(df,n,metric)._get_value(0,"Date"),
                    ls =':',
                    color= col)

def plotAll(df:pd.DataFrame,
            n:int,
            metric:str,
            col:str = 'b',
            bins:int = 14,
            mcl = True):
    '''
    Plot all the towns and highlight town (n) in (col) with rolling avg over
    9bins) and maybe a line where there are the most new cases (not avgd)
    '''
    for i in range(1, ByTown["Town number"].max()):
        if i == n:
            plotTownCases(ByTown, i, metric, col, maxcaseline = mcl,
                          IncludeLegend = True)

        else:
            plotTownCases(ByTown,
                          i,
                          metric,
                          'grey',
                          maxcaseline = False,
                          IncludeLegend = False)

#makes raw data into nice data for optimizer :)
def npTownForLM(df:pd.DataFrame, n:int, metric: str) -> np.array:
    #Want np.ndarray with
    q = getTown(df, n)[metric].tail(len(getTown(df, n)[metric])-70)
    q = q.to_numpy()
    return(np.nan_to_num(q))
```

Next, produce a plot to check the above chunks:

```
#Plots
plotTownCases(ByTown, 13, "Confirmed cases", 'g', maxcaseline = True)
plotTownCases(ByTown, 151, "Confirmed cases",'b', maxcaseline = True)
plotTownCases(ByTown, 135, "Confirmed cases", 'y', maxcaseline = True)
plotTownCases(ByTown, 15, "Confirmed cases", 'r', maxcaseline = True)
#plotAll(ByTown, 15, "Confirmed cases")
```

```
plotTownCases(ByTown, 83, "Confirmed cases", 'c', maxcaseline = True)
plt.show()
```




```
#Define gradients for big model
```