MandatoryInference

May 14, 2021

1 Task 1

Given Task : Clean your dataset (remove missing values, sanitize data, etc.). Remove any outliers (except 0s) using the Tukey's rule from class using the default values as in class. Report what you found (number of outliers). Comment on your findings both for data cleaning (what issues you found, how you dealt with them) and outlier detection.

In this task, we have checked for missing/null values. Then we detected the outliers using Tukey's rule, removed them and reported the number of outliers and details of the findings.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import collections
from scipy.stats import poisson, binom, geom, gamma
```

[2]:
```
states_df = pd.read_csv("States Data/22.csv")
```

[3]:
```
#Checking for null values in dataset
print(states_df.isnull().values.any())
```

```
False
```

There are **no missing values** in the dataset.

[4]:
```
#Calculating the difference between each row as cumulative data is given in the␣
 ↪states dataset.
states_df_inter = states_df.set_index('Date').diff()
states_df_inter.fillna(0, inplace=True)
```

[5]:
```
#Calculating the Q1, Q3 and IQR
Q1 = states_df_inter.quantile(0.25)
Q3 = states_df_inter.quantile(0.75)
IQR = Q3 - Q1
print("---- Lower range for Tukey's rule ----\n\n")
print("Column\t\tValue\n")
print(Q1 - 1.5 * IQR)
print("\n\n---- Upper range for Tukey's rule ----\n\n")
print("Column\t\tValue\n")
print(Q3 + 1.5 * IQR)
```

```
---- Lower range for Tukey's rule ----


Column          Value

TN confirmed    -2696.750
TX confirmed   -10068.125
TN deaths         -45.000
TX deaths        -213.000
dtype: float64


---- Upper range for Tukey's rule ----


Column          Value

TN confirmed     5275.250
TX confirmed    19458.875
TN deaths          83.000
TX deaths         401.000
dtype: float64
```

[6]:
```
states_df_inter.shape
```

[6]:
```
(438, 4)
```

[7]:
```
states_df_inter.to_csv("States Data/22_inter.csv")
```

[8]:
```
#Getting the rows which are outliers (detected by Tukey's rule with alpha = 1.5)
outliers = states_df_inter[((states_df_inter < (Q1 - 1.5 * IQR))␣
 ↪|(states_df_inter > (Q3 + 1.5 * IQR))).any(axis=1)]
outliers
```

[8]:
```
            TN confirmed  TX confirmed  TN deaths  TX deaths
Date
2020-07-27        2547.0        4412.0       11.0      673.0
2020-07-31        5793.0        8360.0       27.0      295.0
2020-11-09        5860.0        4193.0       15.0       22.0
2020-11-11        3630.0       11071.0       89.0      141.0
2020-11-15        5817.0        6008.0       16.0       89.0
…                    …             …         …          …
2021-02-03        1856.0       17372.0      133.0      418.0
2021-02-04        3154.0       15131.0      169.0      439.0
```

```
2021-02-05        2661.0       15357.0       203.0       401.0
2021-02-08        1203.0        5987.0        96.0        57.0
2021-02-10        3652.0       12372.0       119.0       386.0

[62 rows x 4 columns]
```

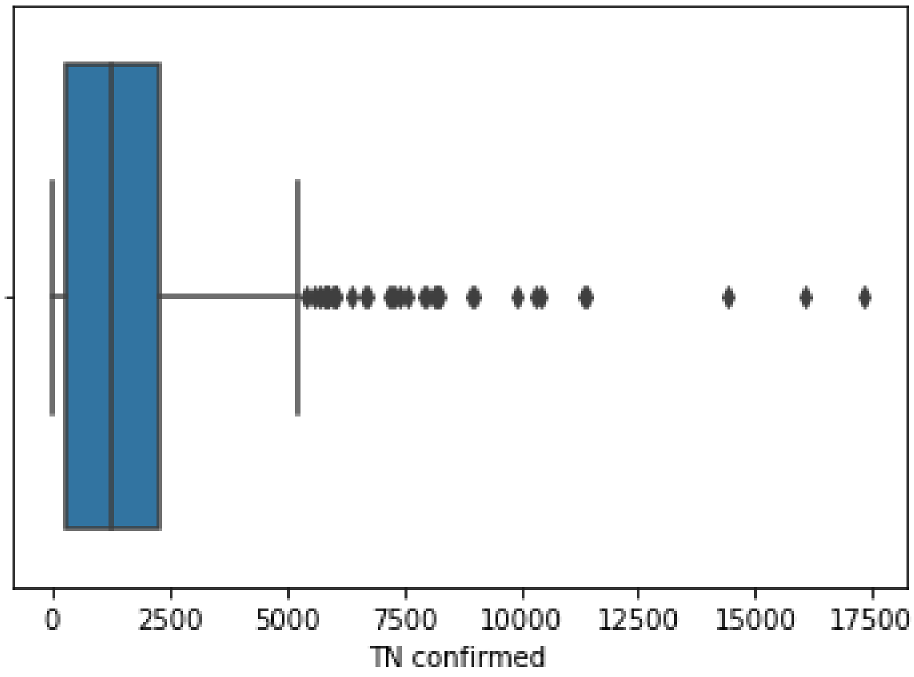
[9]:
```
outliers.to_csv("States Data/outliers.csv")
outliers_date = outliers.reset_index().iloc[:,0].tolist()
```

There were **62 outliers**. The outlier values can be found in outliers.csv

We can confirm the outliers from the box plots for the 4 columns

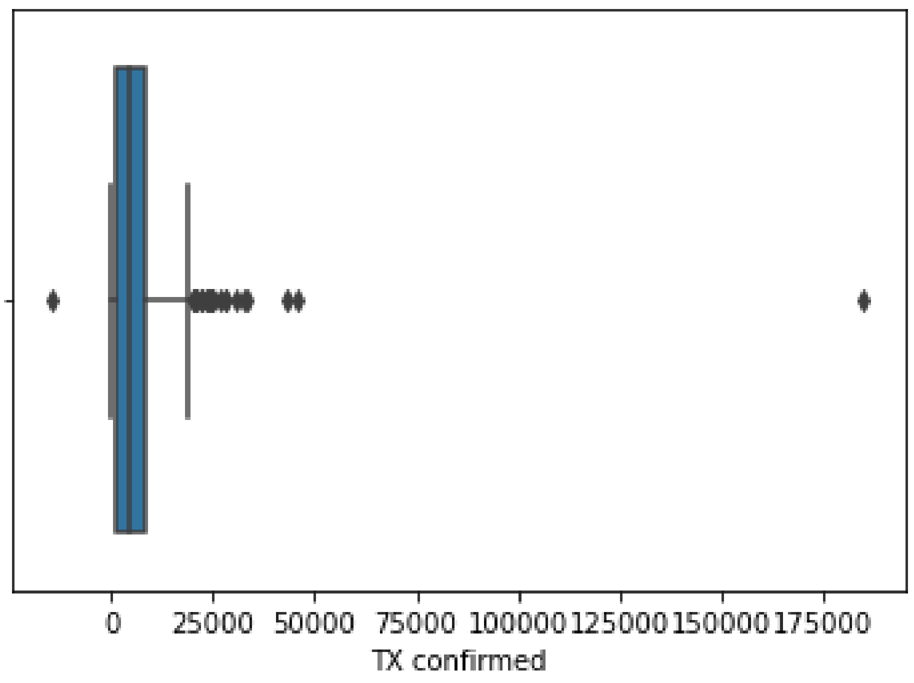
[10]:
```
sns.boxplot(x=states_df_inter['TN confirmed'])
```

[10]:
```
<AxesSubplot:xlabel='TN confirmed'>
```



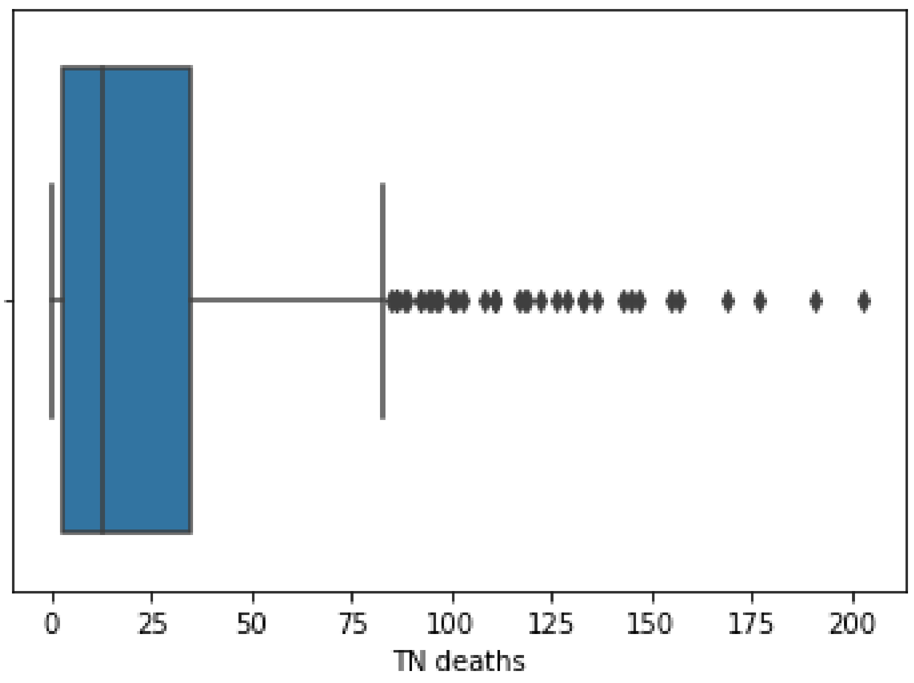
[11]:
```
sns.boxplot(x=states_df_inter['TX confirmed'])
```

[11]:
```
<AxesSubplot:xlabel='TX confirmed'>
```



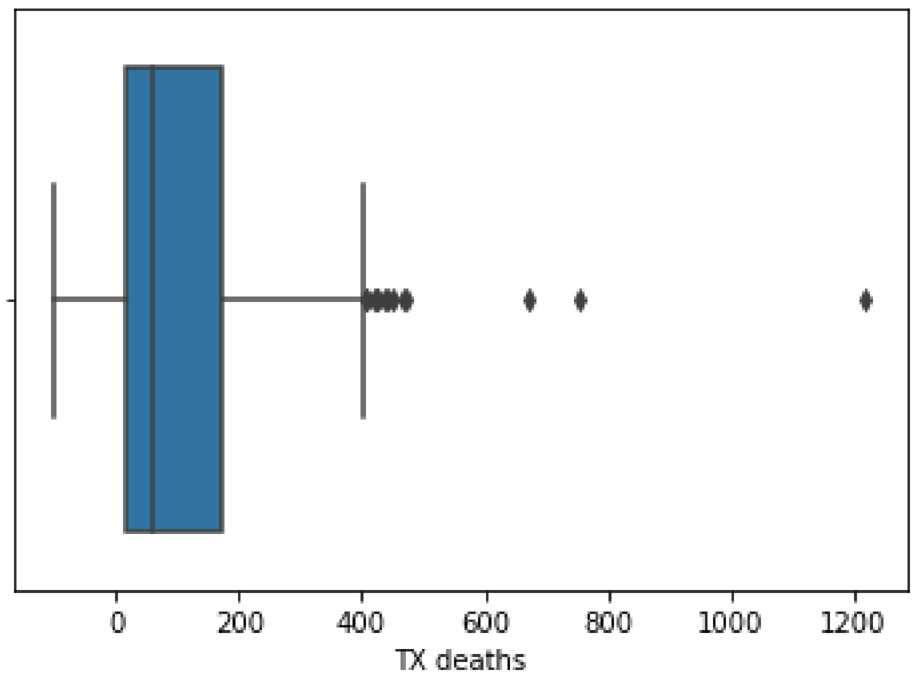
```
[12]: sns.boxplot(x=states_df_inter['TN deaths'])
```

```
[12]: <AxesSubplot:xlabel='TN deaths'>
```



[13]:
```
sns.boxplot(x=states_df_inter['TX deaths'])
```

[13]:
```
<AxesSubplot:xlabel='TX deaths'>
```



[14]:
```
#Removing the outliers from data
states_df_inter = states_df_inter.reset_index()
states_df_out = states_df_inter[~states_df_inter['Date'].isin(outliers_date)]
states_df_out.shape
```

[14]:
```
(376, 5)
```

[15]:
```
states_df_out.to_csv("States Data/22_cleaned.csv")
```

The 62 outliers detected are dropped and the remaining daily data values are stored in 22__cleaned.csv

2 Task 2

3 Required Inference 1 - AR and EWMA

Given Task : In this task, we want to predict COVID19 stats for each state. Use the COVID19 dataset to predict the COVID19 fatality and #cases for the fourth week in August 2020 using data from the first three weeks of August 2020. Do this separately for each of the two states. Use the following four prediction techniques: (i) AR(3), (ii) AR(5), (iii) EWMA with alpha = 0.5, and (iv) EWMA with alpha = 0.8. Report the accuracy (MAPE as a % and MSE) of your predictions using the actual fourth week data.

```
[16]: states_df_out.shape
```

```
[16]: (376, 5)
```

Filtering data to contain only the rows for the month of August

```
[17]: states_df_2a = states_df_out[((states_df_out['Date'] >= '2020-08-01') &␣
       ↪(states_df_out['Date'] <= '2020-08-31'))]
      states_df_2a.shape
```

```
[17]: (31, 5)
```

We are creating two dataframes - test (last week of August) and train (first 3 weeks of August)

```
[18]: train, test = states_df_2a[0:-7],states_df_2a[-7:]
```

3.1 AR with p = 3 and 5

```
[19]: #Helper function for MSE and MAPE
      def getMSEandMAPE(actual, prediction):
          mse = 0
          mape = 0
          for x in range(len(actual)):
              mse += np.square(prediction[x] - actual[x])
              mape += abs(actual[x] - prediction[x])/actual[x]
          mse = mse/len(actual)
          mape = (mape/len(actual))*100
          return mse, mape
```

```
[20]: #Helper function for plotting graph
      def plotGraph(date, actual, prediction):
          plt.plot(date,actual,label="Actual")
          plt.plot(date,prediction,label="Prediction")
          plt.xlabel('X')
          plt.ylabel('Y')
          plt.legend(loc='upper left')
          plt.xticks(rotation=30)
          plt.show()
```

For Autoregression, we have written 3 methods : trainAR, getBetaValues and predictAR. We retrain the model as new information comes in before prediction (as discussed in class).

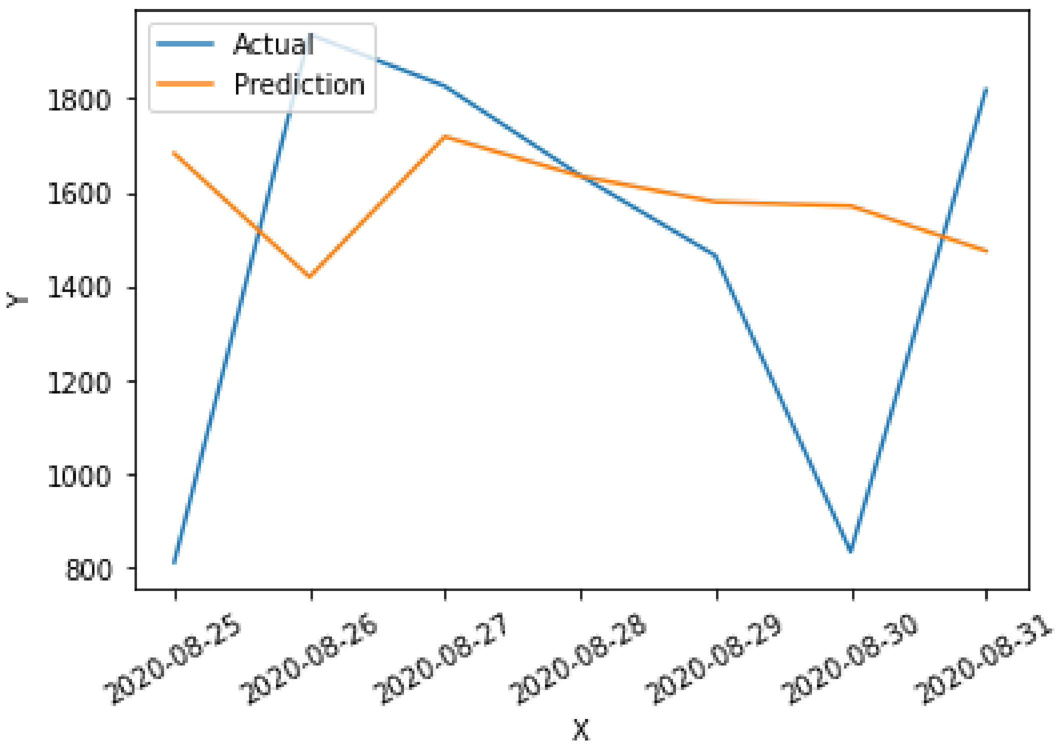
```
[21]: def trainAR(data,p,curLen):
          X = []
          Y = []
          for i in range(curLen):
              if i+p < curLen :
                  X.append([1] + list(data[i:i+p]))
                  Y.append(data[i+p])
              else:
                  break
          return X, Y

      def getBetaValues(X,Y):
          beta=np.matmul(np.linalg.inv(np.matmul(np.transpose(X),X)),np.matmul(np.
       ↪transpose(X),Y))
          return beta

      def predictAR(train, test, p):
          data = np.hstack([train, test])
          trainLen = data.shape[0] - test.shape[0]
          predictions = np.zeros(test.shape[0])
          for i in range(trainLen,data.shape[0]):
              dat = np.hstack([[1], data[i-p:i]])
              X,Y = trainAR(data, p, i)
              beta = getBetaValues(X,Y)
              predictions[i-trainLen] = np.matmul(dat,beta)

          return predictions
```

We are calculating AR(3) and AR(5) for all the 4 columns : TN confirmed, TX confirmed, TN deaths and TX deaths. We are displaying the results along with the MSE and MAPE% in a table, and also plotting the graph between actual and prediction results

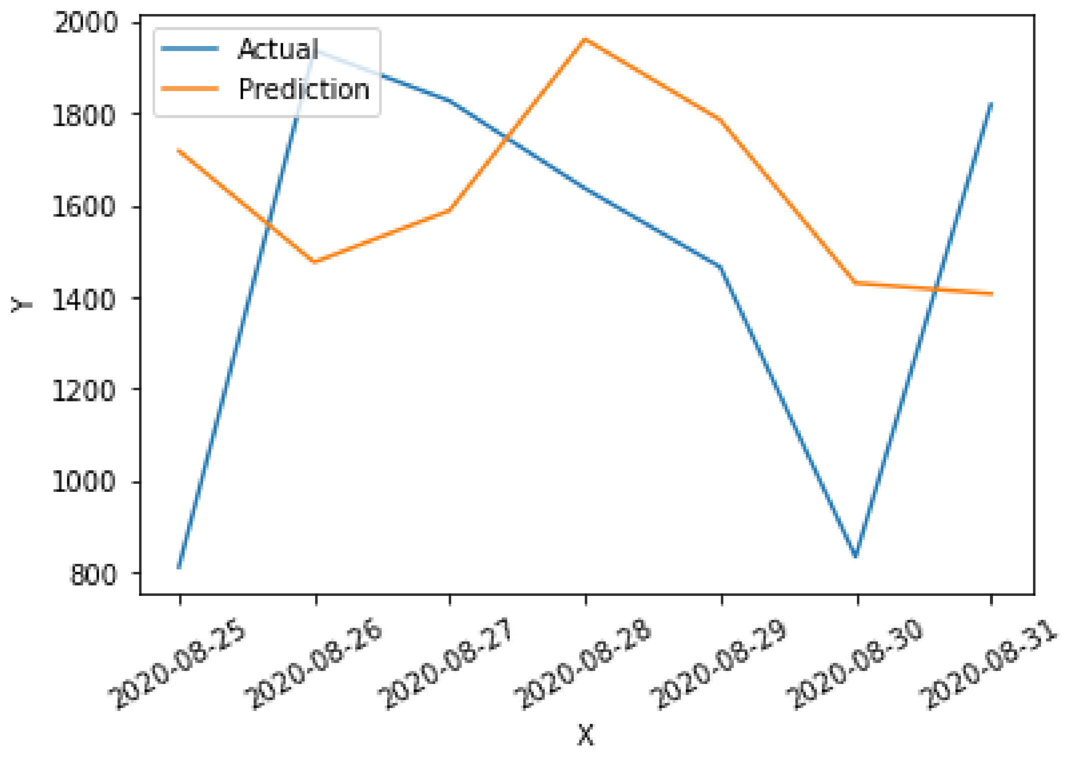
```
[22]: cols = ['TN confirmed', 'TX confirmed', 'TN deaths', 'TX deaths']
      date = np.array(test['Date'])
      for col in cols:

          actual = np.array(test[col])
          prediction_3 = predictAR(np.array(train[col]),np.array(test[col]), 3)
          prediction_5 = predictAR(np.array(train[col]),np.array(test[col]), 5)


          df_ar1 = pd.DataFrame()
          df_ar1['Date'] = date
          df_ar1['Actual'] = actual
          df_ar1['Predicted AR(3)'] = prediction_3
          df_ar1['Predicted AR(5)'] = prediction_5
```

```
    print('\033[1m --------------- ',col,' ---------------------- \033[0m \n')
    print(df_ar1, '\n')

    mse_3, mape_3 = getMSEandMAPE(actual,prediction_3)
    mse_5, mape_5 = getMSEandMAPE(actual,prediction_5)


    df_ar2 = pd.DataFrame()
    df_ar2['p'] = [3,5]
    df_ar2['MSE'] = mse_3, mse_5
    df_ar2['MAPE(%)'] = mape_3, mape_5
    print(df_ar2, '\n\n')


    print("\t\tp = 3")
    plotGraph(date, actual, prediction_3)
    print('\n\n')

    print("\t\tp = 5")
    plotGraph(date, actual, prediction_5)
    print('\n\n')
```

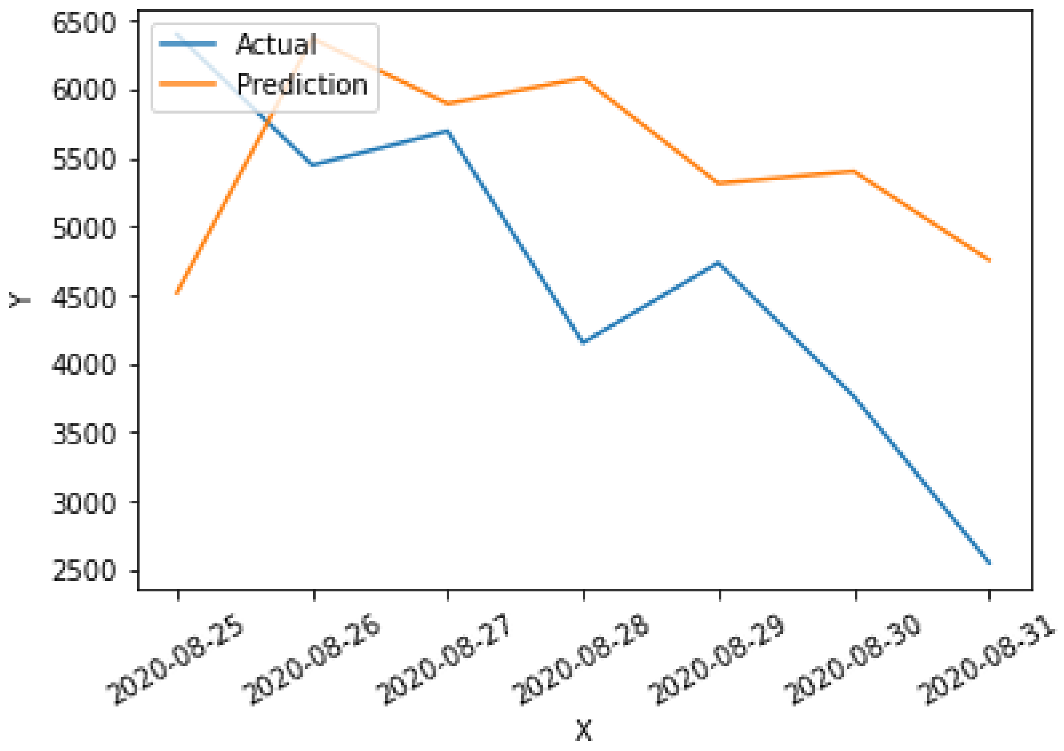
```
 ---------------  TN confirmed  ----------------------

         Date  Actual  Predicted AR(3)  Predicted AR(5)
0  2020-08-25   813.0      1682.735460      1717.652240
1  2020-08-26  1936.0      1419.627093      1475.290305
2  2020-08-27  1826.0      1718.582469      1587.893367
3  2020-08-28  1636.0      1634.727556      1960.396517
4  2020-08-29  1465.0      1579.627186      1784.786379
5  2020-08-30   835.0      1571.120433      1430.050688
6  2020-08-31  1818.0      1475.304503      1407.024957

   p            MSE    MAPE(%)
0  3  243867.683117  36.349112
1  5  259689.443258  40.519518


                p = 3
```


Actual
Prediction
1800
1600
1400
1200
1000
800
Y
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31
X

```
p = 5
```

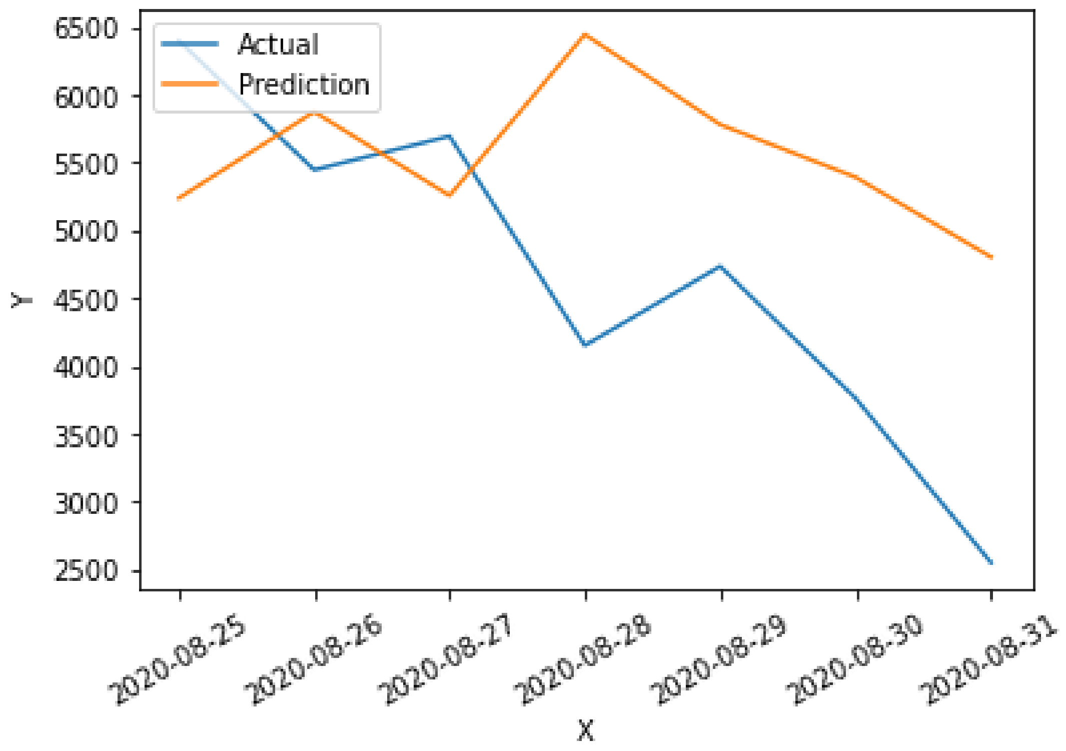


```
 ---------------  TX confirmed  ----------------------

          Date  Actual  Predicted AR(3)  Predicted AR(5)
0   2020-08-25  6397.0      4514.706349      5236.600036
1   2020-08-26  5445.0      6367.400933      5873.624214
2   2020-08-27  5694.0      5893.878577      5255.526762
3   2020-08-28  4150.0      6077.947372      6443.795974
4   2020-08-29  4733.0      5312.722264      5779.291486
5   2020-08-30  3761.0      5401.093170      5391.265881
6   2020-08-31  2550.0      4753.796890      4802.147329

   p           MSE    MAPE(%)
0  3  2.290499e+06  34.087384
1  5  2.258381e+06  34.679552


                p = 3
```

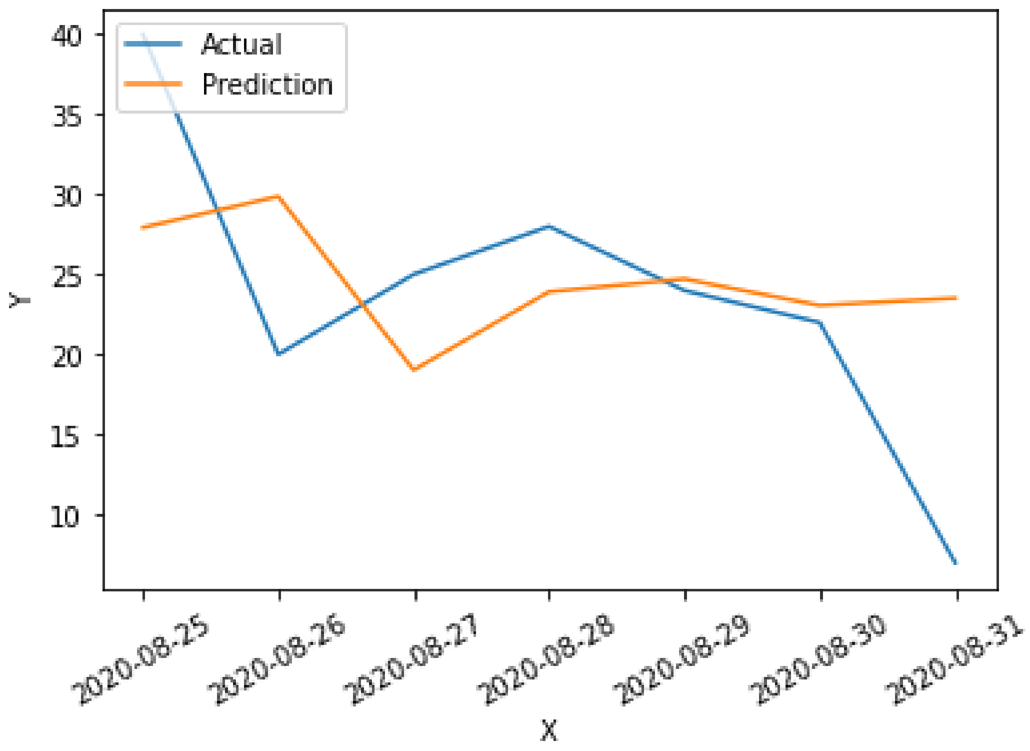


Actual
Prediction
6500
6000
5500
5000
4500
4000
3500
3000
2500
Y
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31
X

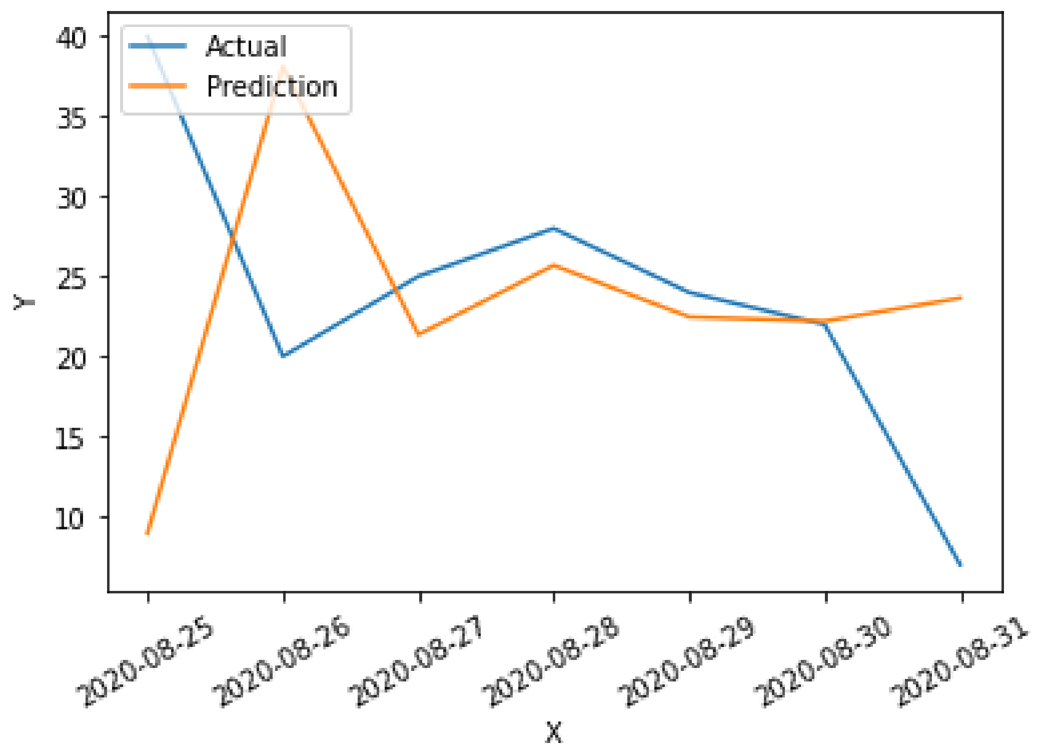
```
p = 5
```



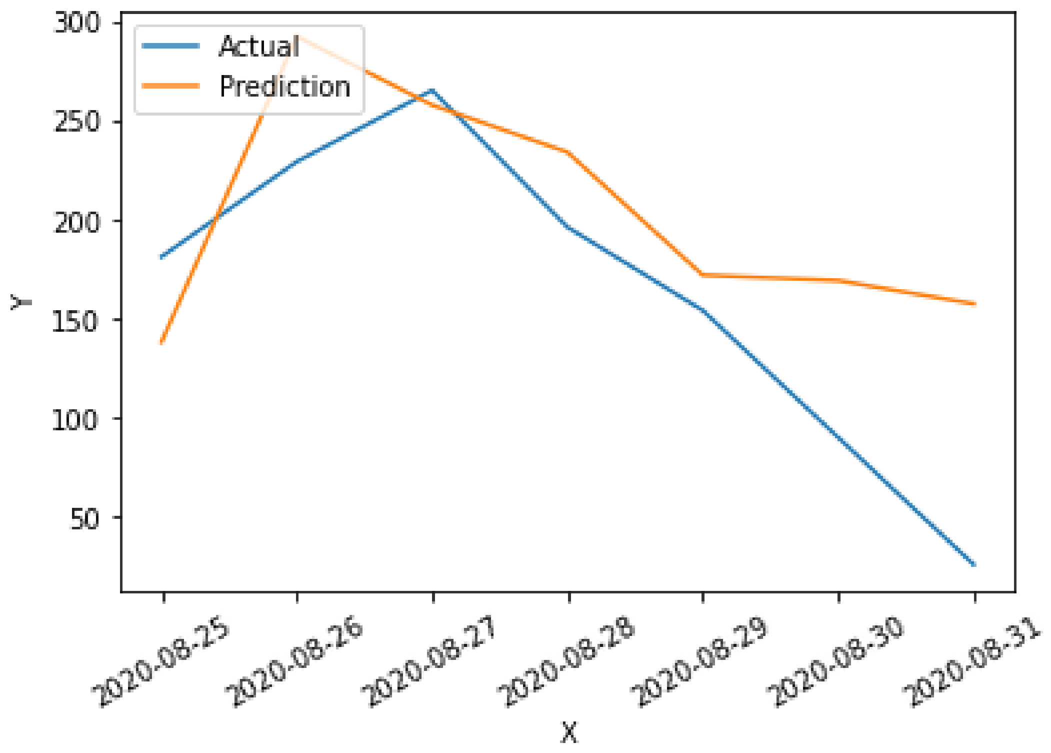
```
 ---------------  TN deaths  ----------------------

         Date  Actual  Predicted AR(3)  Predicted AR(5)
0  2020-08-25    40.0        27.932749         8.964733
1  2020-08-26    20.0        29.877670        38.125933
2  2020-08-27    25.0        19.024509        21.368313
3  2020-08-28    28.0        23.921775        25.686135
4  2020-08-29    24.0        24.715985        22.474346
5  2020-08-30    22.0        23.070699        22.202311
6  2020-08-31     7.0        23.523434        23.651808

   p         MSE    MAPE(%)
0  3   81.458318  51.703239
1  5  227.133090  62.309692


                p = 3
```


Actual
Prediction
Y
X
40
35
30
25
20
15
10
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31

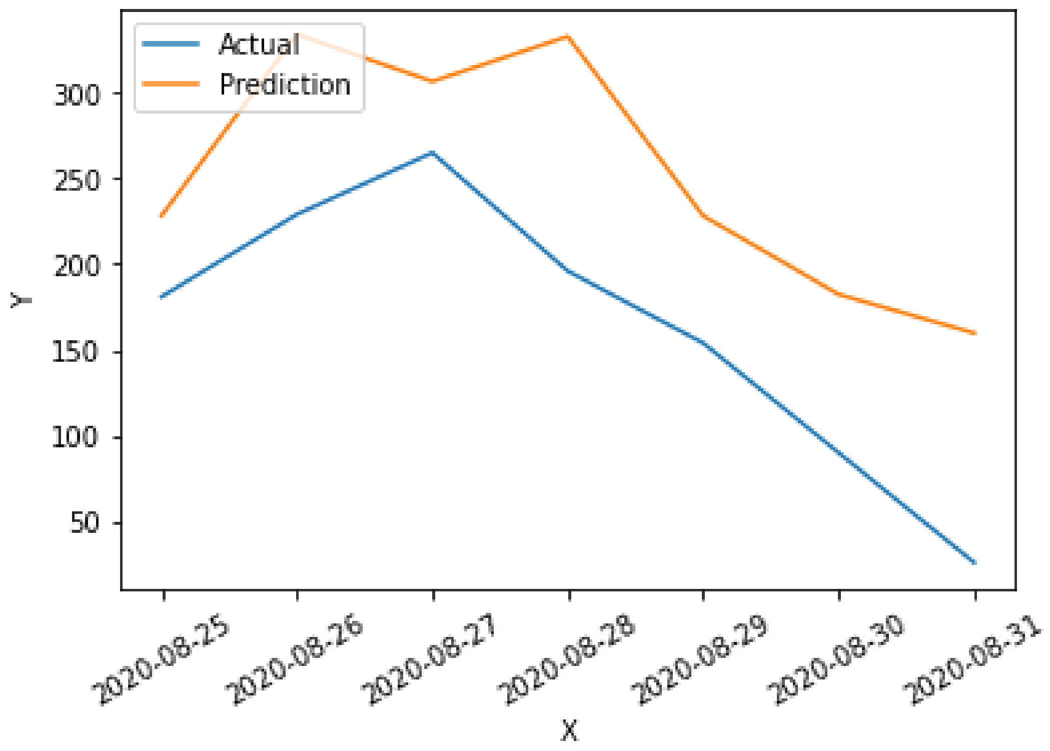
```
p = 5
```



```
 ---------------  TX deaths  ----------------------

        Date  Actual  Predicted AR(3)  Predicted AR(5)
0  2020-08-25   181.0       138.099671       228.313805
1  2020-08-26   229.0       292.103796       333.991673
2  2020-08-27   265.0       257.573662       306.541542
3  2020-08-28   196.0       233.817839       332.813780
4  2020-08-29   154.0       171.750801       228.116348
5  2020-08-30    90.0       169.063178       182.387426
6  2020-08-31    26.0       157.386058       159.839522

   p          MSE     MAPE(%)
0  3  4448.034264   96.865821
1  5  9378.177823  117.573529


                p = 3
```


Actual
Prediction
300
250
200
150
100
50
Y
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31
X

```
p = 5
```



3.2 EWMA with alpha = 0.5 and 0.8

For EWMA, I have one method to calculate the predictions based on the input alpha and data.

[23]:
```
def EWMA(data, alpha):

    y_pred = []
    y_pred.append(data[0])
    for i in range(1,len(data)):
        y_pred.append(alpha * data[i-1] + (1 - alpha) * y_pred[i-1])

    y_actual = data
    #Computing MSE and MAPE for test set
    MSE, MAPE = getMSEandMAPE(y_pred[-7:], y_actual[-7:])
    return y_pred,MSE,MAPE
```

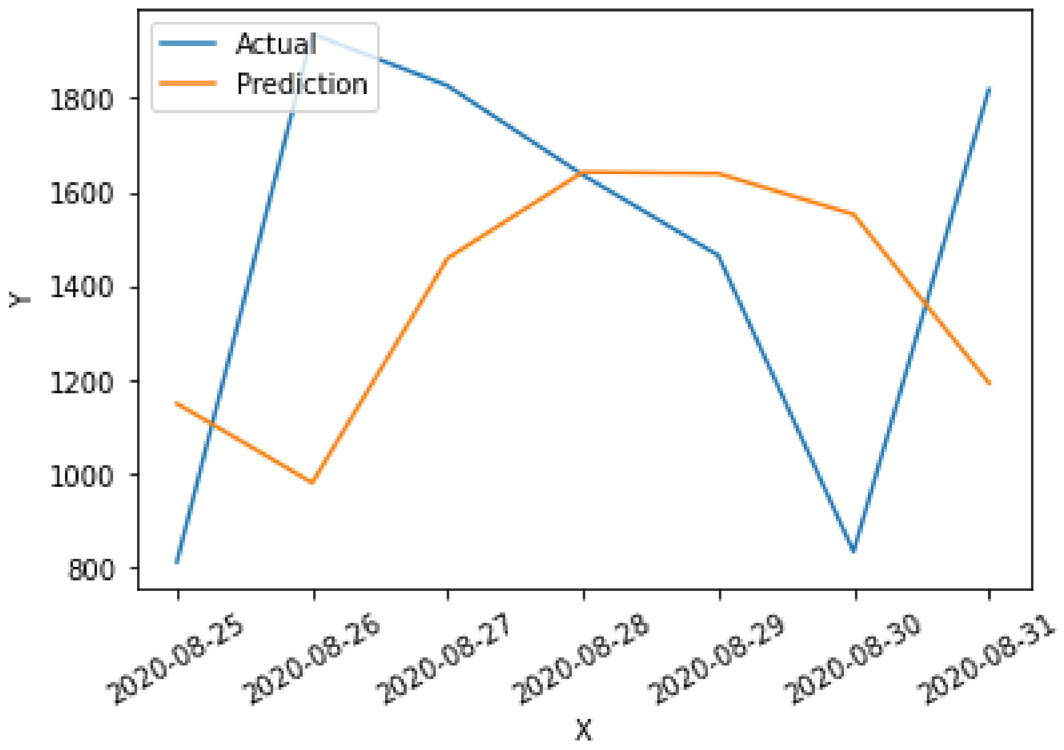
We are calculating EWMA with alpha = 0.5 and EWMA with alpha = 0.8 for all the 4 columns : TN confirmed, TX confirmed, TN deaths and TX deaths. We are displaying the results along with the MSE and MAPE% in a table, and also plotting the graph between actual and prediction results

```
[24]: cols = ['TN confirmed', 'TX confirmed', 'TN deaths', 'TX deaths']
      date = np.array(test['Date'])
      for col in cols:

          y_predicted_point5, MSE_predicted_point5, MAPE_predicted_point5 =␣
       ↪EWMA((states_df_2a[col]).tolist(),0.5)
          y_predicted_point8,MSE_predicted_point8, MAPE_predicted_point8  =␣
       ↪EWMA((states_df_2a[col]).tolist(),0.8)

          df_ar1 = pd.DataFrame()
          actual = np.array(test[col])
          df_ar1['Date'] = date
          df_ar1['Actual'] = actual
          df_ar1['Predicted_EWMA(0.5)'] = y_predicted_point5[-7:]
          df_ar1['Predicted_EWMA(0.8)'] = y_predicted_point8[-7:]
          print('\033[1m --------------- ',col,' ---------------------- \033[0m \n')
          print(df_ar1, '\n\n')

          df_ar2 = pd.DataFrame()
          df_ar2['Alpha'] = [0.5,0.8]
          df_ar2['MSE'] = MSE_predicted_point5, MSE_predicted_point8
          df_ar2['MAPE(%)'] = MAPE_predicted_point5, MAPE_predicted_point8
          print(df_ar2, '\n\n')

          print("\t\tAlpha = 0.5")
          plotGraph(date, actual, y_predicted_point5[-7:])
          print('\n\n')

          print("\t\tAlpha = 0.8")
          plotGraph(date, actual, y_predicted_point8[-7:])
          print('\n\n')
```

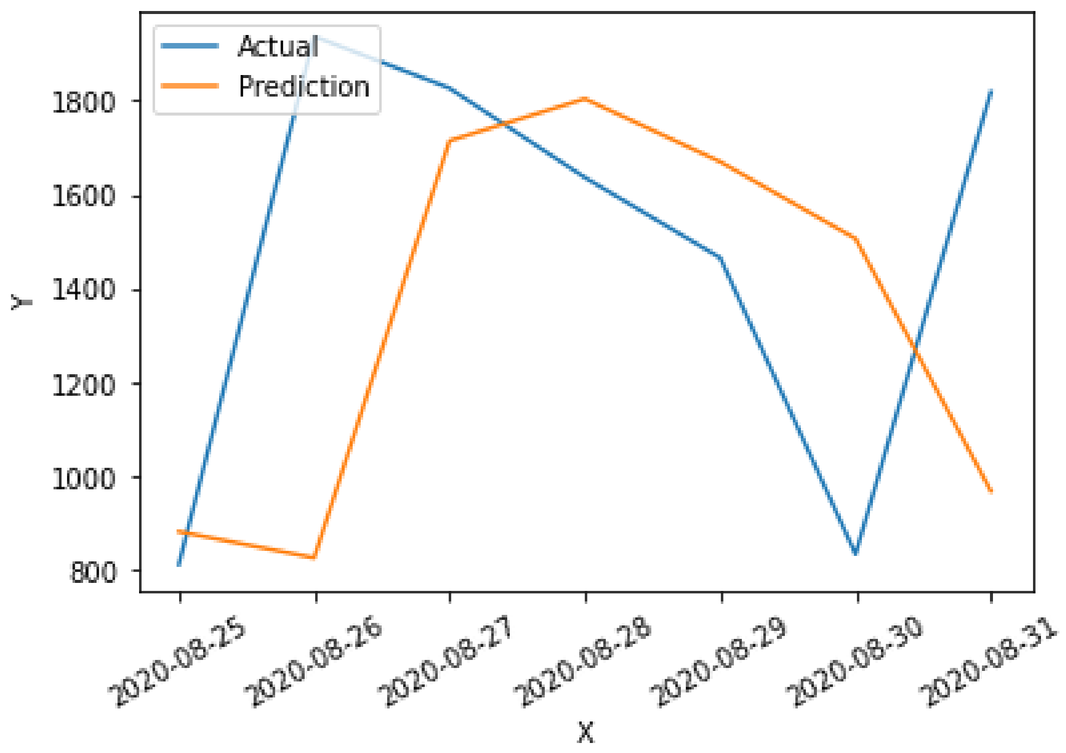
```
 ---------------  TN confirmed  ----------------------

         Date  Actual  Predicted_EWMA(0.5)  Predicted_EWMA(0.8)
0  2020-08-25   813.0          1149.772742           882.915525
1  2020-08-26  1936.0           981.386371           826.983105
2  2020-08-27  1826.0          1458.693185          1714.196621
3  2020-08-28  1636.0          1642.346593          1803.639324
4  2020-08-29  1465.0          1639.173296          1669.527865
5  2020-08-30   835.0          1552.086648          1505.905573
6  2020-08-31  1818.0          1193.543324           969.181115



   Alpha            MSE    MAPE(%)
```

```
0    0.5  299164.767727  37.325142
1    0.8  355407.004003  43.174667
```

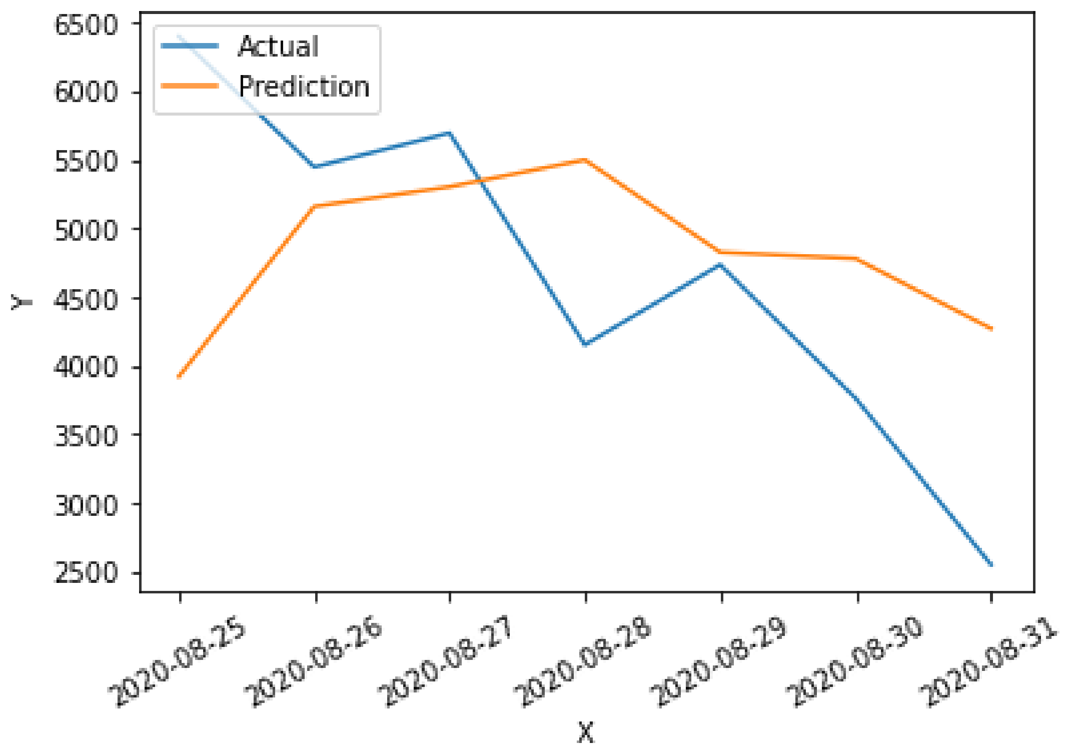
```
Alpha = 0.5
```



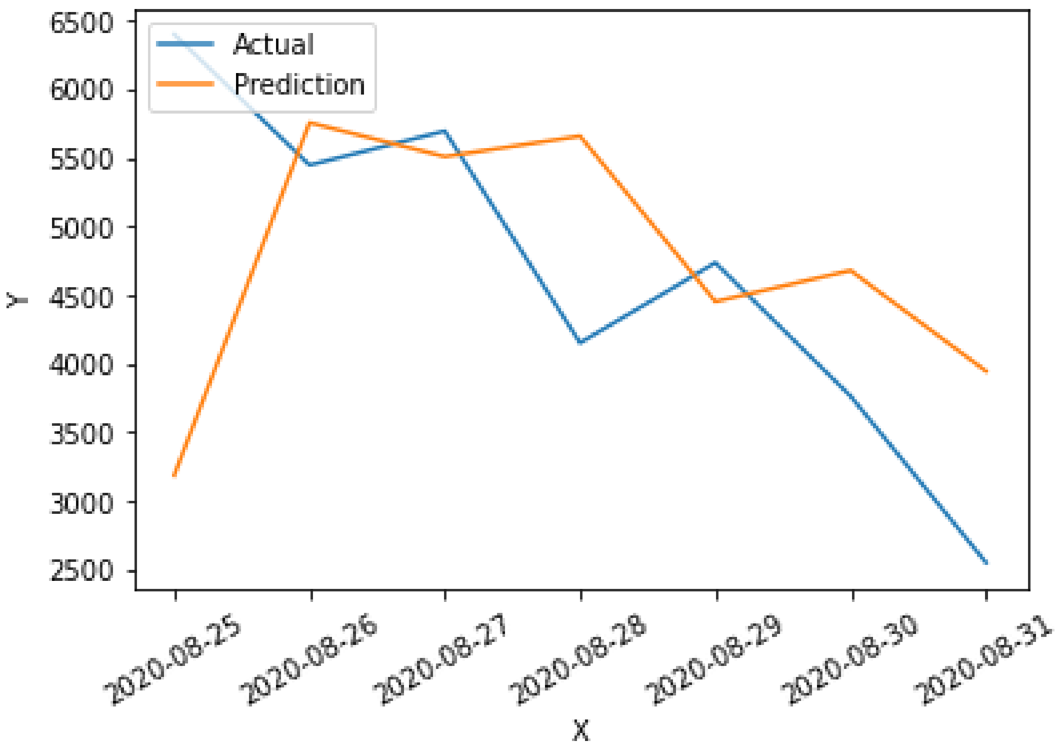
```
Alpha = 0.8
```



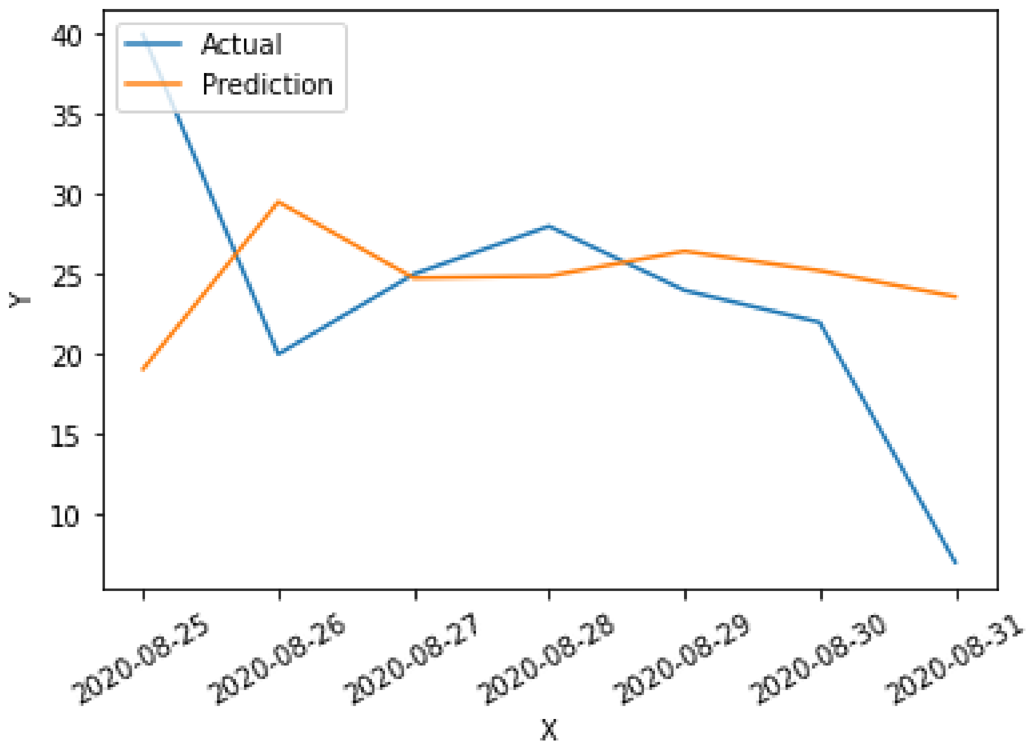
```
 ---------------  TX confirmed  ----------------------

         Date  Actual  Predicted_EWMA(0.5)  Predicted_EWMA(0.8)
0  2020-08-25  6397.0          3918.991362          3185.316248
1  2020-08-26  5445.0          5157.995681          5754.663250
2  2020-08-27  5694.0          5301.497840          5506.932650
3  2020-08-28  4150.0          5497.748920          5656.586530
4  2020-08-29  4733.0          4823.874460          4451.317306
5  2020-08-30  3761.0          4778.437230          4676.663461
6  2020-08-31  2550.0          4269.718615          3944.132692


   Alpha           MSE    MAPE(%)
0    0.5  1.742036e+06  23.452344
1    0.8  2.225285e+06  28.213501


                Alpha = 0.5
```


Actual
Prediction
6500
6000
5500
5000
4500
4000
3500
3000
2500
Y
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31
X

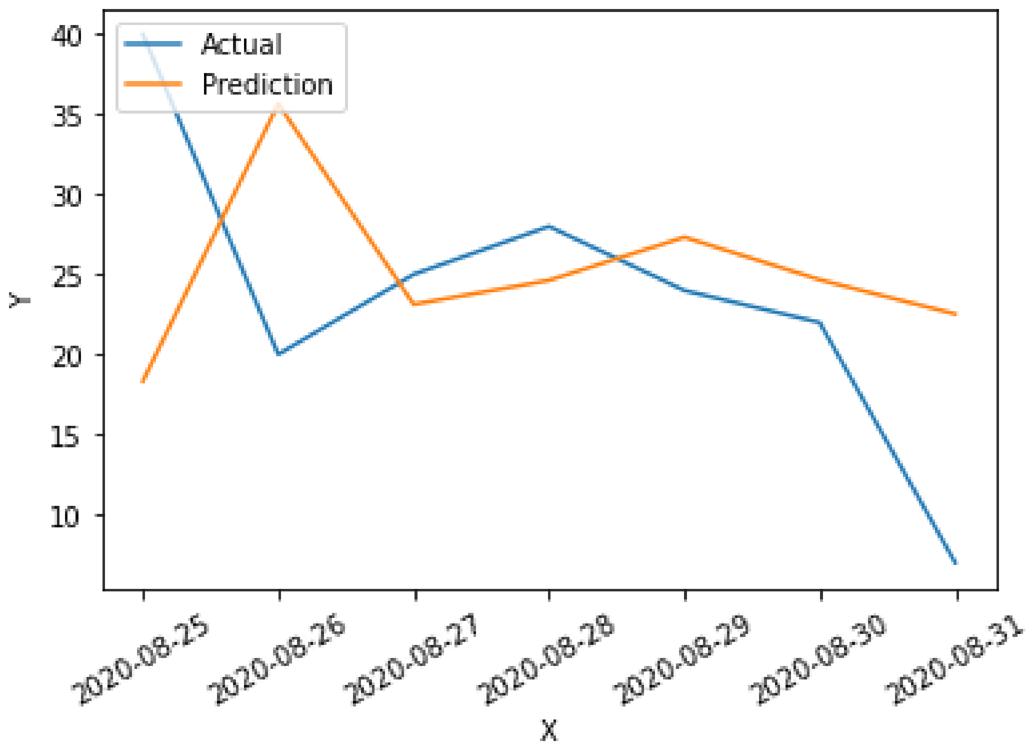
```
Alpha = 0.8
```



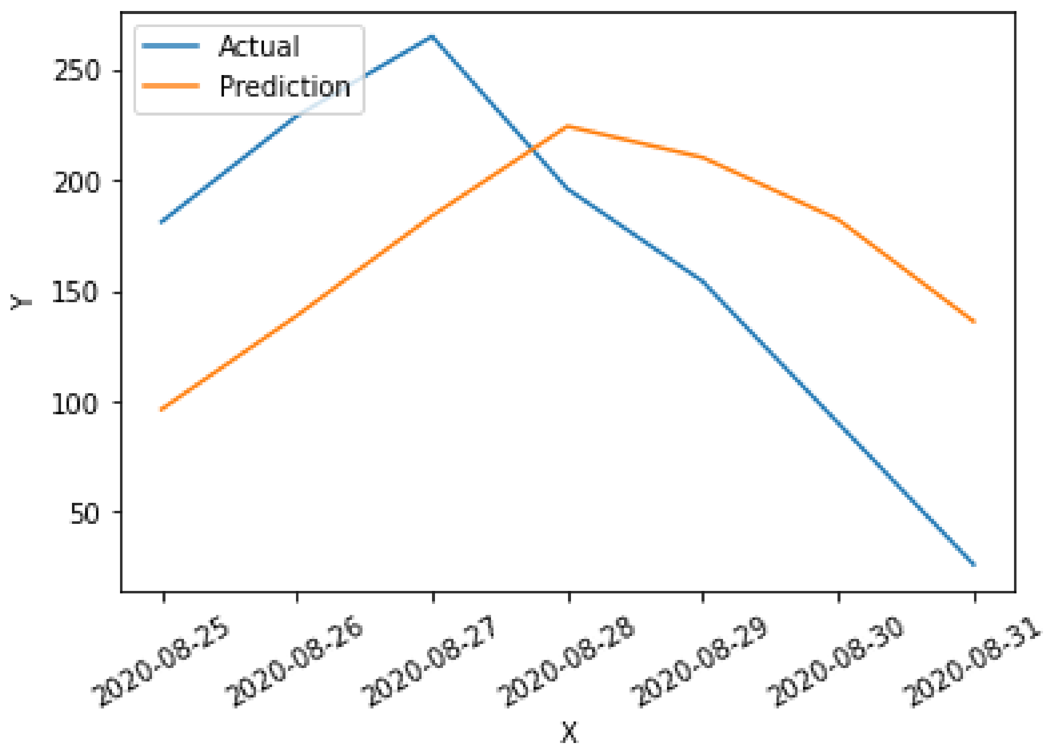
```
 ---------------  TN deaths  ----------------------

         Date  Actual  Predicted_EWMA(0.5)  Predicted_EWMA(0.8)
0  2020-08-25    40.0            19.057459            18.333400
1  2020-08-26    20.0            29.528729            35.666680
2  2020-08-27    25.0            24.764365            23.133336
3  2020-08-28    28.0            24.882182            24.626667
4  2020-08-29    24.0            26.441091            27.325333
5  2020-08-30    22.0            25.220546            24.665067
6  2020-08-31     7.0            23.610273            22.533013


   Alpha         MSE    MAPE(%)
0    0.5  118.770719  35.428052
1    0.8  141.312164  39.397461


               Alpha = 0.5
```


Actual
Prediction
Y
X
40
35
30
25
20
15
10
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31

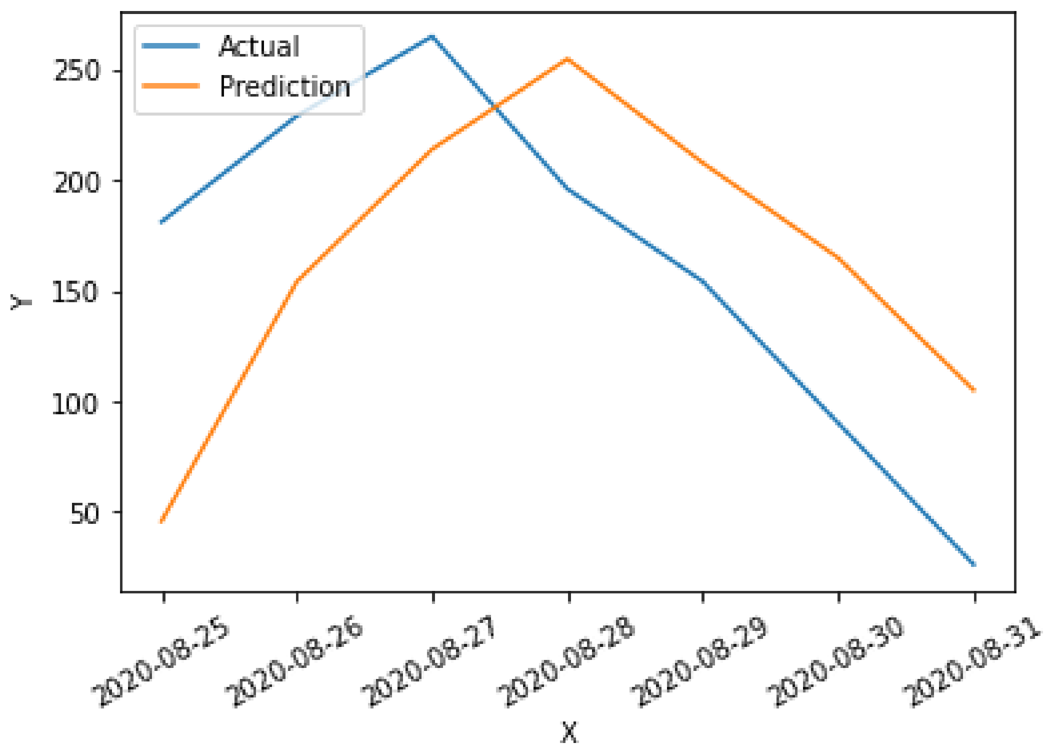
```
Alpha = 0.8
```



```
 ---------------  TX deaths  ----------------------

         Date  Actual  Predicted_EWMA(0.5)  Predicted_EWMA(0.8)
0  2020-08-25   181.0            96.157597            45.529315
1  2020-08-26   229.0           138.578799           153.905863
2  2020-08-27   265.0           183.789399           213.981173
3  2020-08-28   196.0           224.394700           254.796235
4  2020-08-29   154.0           210.197350           207.759247
5  2020-08-30    90.0           182.098675           164.751849
6  2020-08-31    26.0           136.049337           104.950370


   Alpha          MSE   MAPE(%)
0    0.5  6646.687472  52.64619
1    0.8  6394.630060  77.10447


               Alpha = 0.5
```


Actual
Prediction
250
200
150
100
50
Y
2020-08-25
2020-08-26
2020-08-27
2020-08-28
2020-08-29
2020-08-30
2020-08-31
X

```
Alpha = 0.8
```



3.3 Required Inference 2 - Wald's test, Z-test, and t-test

Given Task :In this step, we want to check, for each state, how the mean of monthly COVID19 stats has changed between Feb 2021 and March 2021. Apply the Wald's test, Z-test, and t-test (assume all are applicable) to check whether the mean of COVID19 deaths and #cases are different for Feb'21 and March'21 in the two states. That is, we are checking, for each state separately, whether the mean of daily cases and the mean of daily deaths for Feb'21 is different from the corresponding mean of daily values for March'21. Use MLE for Wald's test as the estimator; assume for Wald's estimator purposes that daily data is Poisson distributed. Note, you have to report results for deaths and #cases in both states separately. After running the test and reporting the numbers, check and comment on whether the tests are applicable or not. First use one-sample tests for Wald's, Z-test, and t-test by computing the sample mean of daily values from Feb'21 and using that as a guess for mean of daily values for March'21; here, your sample data for computing sample mean will be the 28 daily values in Feb'21 whereas your sample data for running the test will be the 31 daily values of March'21. Then, repeat with the two-sample version of Wald's and two-sample unpaired t-test (here, your two samples will be the 28 values of Feb'21 and the 31 values of March'21). Use =0.05 for all. For t-test, the threshold to check against is tn-1, /2 for two-tailed, where n is the number of data points. You can find these values in online t tables, similar to z tables. For Z-test, use the corrected sample standard deviation of the entire COVID19 dataset you have for each state as the true sigma value.

NOTE

For all the tests below, we set the below hypothesis

Null Hypothesis H0: Mean number of deaths/cases in March,'21 is equal to mean number of deaths/cases in Feb,'21

Alternative Hypothesis H1: Mean number of deaths/cases in March,'21 is not equal to mean number of deaths/cases in Feb,'21

3.4 One sample Wald's Test¶

[25]:
```
feb_month_data = states_df_out[(states_df_out['Date'] >= '2021-02-01') &␣
 ↪(states_df_out['Date'] <= '2021-02-28')]
feb_month_data.shape
```

[25]:
```
(21, 5)
```

Note :

If we observe the feb data, no of days is 21 instead of 28.This is because of removal of outliers.For all the tests below, we are considering the updated feb data (21 days)

[26]:
```
march_month_data = states_df_out[(states_df_out['Date'] >= '2021-03-01') &␣
 ↪(states_df_out['Date'] <= '2021-03-31')]
march_month_data.shape
```

[26]:
```
(31, 5)
```

[27]:
```
#State 1 : TN
feb_month_mean_death_TN = feb_month_data['TN deaths'].mean()
feb_month_mean_cases_TN = feb_month_data['TN confirmed'].mean()

march_month_mean_death_TN = march_month_data['TN deaths'].mean()
march_month_mean_cases_TN = march_month_data['TN confirmed'].mean()

print("TN State Feb Death Mean : " + str(feb_month_mean_death_TN))
print("TN State Feb No of Cases Mean : " + str(feb_month_mean_cases_TN))
print("TN State March Death Mean : " + str(march_month_mean_death_TN))
print("TN State March No of Cases Mean : " + str(march_month_mean_cases_TN))
```

```
TN State Feb Death Mean : 38.23809523809524
TN State Feb No of Cases Mean : 1483.2857142857142
TN State March Death Mean : 15.709677419354838
TN State March No of Cases Mean : 1187.4193548387098
```

[28]:
```
#State 2 : TX
feb_month_mean_death_TX = feb_month_data['TX deaths'].mean()
feb_month_mean_cases_TX = feb_month_data['TX confirmed'].mean()

march_month_mean_death_TX = march_month_data['TX deaths'].mean()
march_month_mean_cases_TX = march_month_data['TX confirmed'].mean()

print("TX State Feb Death Mean : " + str(feb_month_mean_death_TX))
print("TX State Feb No of Cases Mean : " + str(feb_month_mean_cases_TX))
print("TX State March Death Mean : " + str(march_month_mean_death_TX))
print("TX State March No of Cases Mean : " + str(march_month_mean_cases_TX))
```

```
TX State Feb Death Mean : 207.85714285714286
TX State Feb No of Cases Mean : 7725.857142857143
TX State March Death Mean : 139.96774193548387
TX State March No of Cases Mean : 4377.129032258064
```

[29]:
```
def walds_one_testing(march_data_mean,feb_data_mean,march_data):
    # W = (theta^- guess) / se_hat(theta^) = (theta^ - guess) / (root(lambda_MLE␣
 ↪/ n))
    guess = feb_data_mean
    theta_hat = march_data_mean  # Since for Poisson-distributed data, MLE␣
 ↪estimator is lamda^ which is equal to sample mean
    n = len(march_data)
    num = theta_hat - guess
    den = np.sqrt(march_data_mean / float(n))
    w_stats = num / den
    return np.abs(w_stats);
```

[30]:
```
#Walds Test Result for State : TN
print("___Walds Test Result for State : TN___")
print()

#Category : Death
walds_one_result_death_TN =␣
 ↪walds_one_testing(march_month_mean_death_TN,feb_month_mean_death_TN,march_month_data)
print("Walds one test Result of State TN under death category is : " +␣
 ↪str(walds_one_result_death_TN))
if(walds_one_result_death_TN > 1.962):
    print("Walds One Sample Testing for mean of death W1 = " +␣
 ↪str(walds_one_result_death_TN) +" which is greater than z_alpha/2 = 1.962 so␣
 ↪reject the NULL hypothesis")
else:
    print("Walds One Sample Testing  for mean of death is  W1 = " +␣
 ↪str(walds_one_result_death_TN)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")
```

```
print()

#Category : Cases
walds_one_result_cases_TN =␣
↪walds_one_testing(march_month_mean_cases_TN,feb_month_mean_cases_TN,march_month_data)
print("Walds one test Result of State TN under Cases category is : " +␣
↪str(walds_one_result_cases_TN))
if(walds_one_result_cases_TN > 1.962):
    print("Walds One Sample Testing for mean of death W1 = " +␣
↪str(walds_one_result_cases_TN) +" which is greater than z_alpha/2 = 1.962 so␣
↪reject the NULL hypothesis");
else:
    print("Walds One Sample Testing for mean of death W1 = " +␣
↪str(walds_one_result_cases_TN)+ " which is less than z_alpha/2 = 1.962 so␣
↪accept the NULL hypothesis")
```

```
___Walds Test Result for State : TN___

Walds one test Result of State TN under death category is : 31.646661759343917
Walds One Sample Testing for mean of death W1 = 31.646661759343917 which is
greater than z_alpha/2 = 1.962 so reject the NULL hypothesis

Walds one test Result of State TN under Cases category is : 47.805115629869654
Walds One Sample Testing for mean of death W1 = 47.805115629869654 which is
greater than z_alpha/2 = 1.962 so reject the NULL hypothesis
```

[31]:
```
#Walds Test Result for State : TX
print("___Walds Test Result for State : TX___")
print()

#Category : Death
walds_one_result_death_TX =␣
↪walds_one_testing(march_month_mean_death_TX,feb_month_mean_death_TX,march_month_data)
print("Walds one test Result of State TX under death category is : " +␣
↪str(walds_one_result_death_TX))
if(walds_one_result_death_TX > 1.962):
    print("Walds One Sample Testing for mean of death W1 = " +␣
↪str(walds_one_result_death_TX) +  " which is greater than z_alpha/2 = 1.962␣
↪so reject the NULL hypothesis")
else:
    print("Walds One Sample Testing for mean of death W1 = " +␣
↪str(walds_one_result_death_TX) + " which is less than z_alpha/2 = 1.962 so␣
↪accept the NULL hypothesis")

print()

#Category : Cases
```

```
walds_one_result_cases_TX =␣
 ↪walds_one_testing(march_month_mean_cases_TX,feb_month_mean_cases_TX,march_month_data)
print("Walds one test Result of State TX under Cases category is : " +␣
 ↪str(walds_one_result_cases_TX))
if(walds_one_result_cases_TX > 1.962):
    print("Walds One Sample Testing for mean of death W1 = "␣
 ↪+str(walds_one_result_cases_TX) +" which is greater than z_alpha/2 = 1.962 so␣
 ↪reject the NULL hypothesis");
else:
    print("Walds One Sample Testing for mean of death W1 = "␣
 ↪+str(walds_one_result_cases_TX)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")
```

```
___Walds Test Result for State : TX___

Walds one test Result of State TX under death category is : 31.949851585786913
Walds One Sample Testing for mean of death W1 = 31.949851585786913 which is
greater than z_alpha/2 = 1.962 so reject the NULL hypothesis

Walds one test Result of State TX under Cases category is : 281.8162684575669
Walds One Sample Testing for mean of death W1 = 281.8162684575669 which is
greater than z_alpha/2 = 1.962 so reject the NULL hypothesis
```

3.5 Z Test

[32]:
```
#Z Test
def compute_corrected_std(data):
    sum = 0
    x_bar = data.mean()
    n = len(data)
    for x in data:
        sum = sum + (np.square((x-x_bar)))
    return np.sqrt(sum/(n-1))
```

[33]:
```
def z_testing(march_data_mean,feb_data_mean,total_data,march_data):
    # Computing the z statistic z = (x_bar - guess) / (corrected_std / root of␣
 ↪(n))
    guess = feb_data_mean
    x_bar = march_data_mean
    corrected_std = compute_corrected_std(total_data)
    num = (x_bar - guess)
    den = corrected_std / np.sqrt(len(march_data))
    z_stats = num / den
    return np.abs(z_stats);
```

[34]:
```
#Z Test Result for State : TN
print("___Z Test Result for State : TN___")
print()

#Category : Death
z_test_result_death_TN =␣
 ↪z_testing(march_month_mean_death_TN,feb_month_mean_death_TN,states_df_out[['TN␣
 ↪deaths']].values,march_month_data)
print("Z test Result of State TN under death category is : " +␣
 ↪str(z_test_result_death_TN))
if(z_test_result_death_TN > 1.962):
    print("Z test Result is Z =" + str(z_test_result_death_TN) + " which is␣
 ↪greater than z_alpha/2 = 1.962 so reject the NULL hypothesis")
else:
    print("Z test Result is Z =" + str(z_test_result_death_TN) + " which is less␣
 ↪than z_alpha/2 = 1.962 so accept the NULL hypothesis")

print()

#Category : Cases
z_test_result_cases_TN =␣
 ↪z_testing(march_month_mean_cases_TN,feb_month_mean_cases_TN,states_df_out[['TN␣
 ↪confirmed']].values,march_month_data)
print("Z test Result of State TN under Cases category is : " +␣
 ↪str(z_test_result_cases_TN))
if(z_test_result_cases_TN > 1.962):
    print("Z test Result is Z =" + str(z_test_result_cases_TN) + " which is␣
 ↪greater than z_alpha/2 = 1.962 so reject the NULL hypothesis");
else:
    print("Z test Result is Z =" + str(z_test_result_cases_TN) + " which is less␣
 ↪than z_alpha/2 = 1.962 so accept the NULL hypothesis")
```

```
___Z Test Result for State : TN___

Z test Result of State TN under death category is : [6.51395867]
Z test Result is Z =[6.51395867] which is greater than z_alpha/2 = 1.962 so
reject the NULL hypothesis

Z test Result of State TN under Cases category is : [1.44253169]
Z test Result is Z =[1.44253169] which is less than z_alpha/2 = 1.962 so accept
the NULL hypothesis
```

[35]:
```
#Z Test Result for State : TX
print("___Z Test Result for State : TX___")
print()

#Category : Death
```

```
z_test_result_death_TX =␣
 ↪z_testing(march_month_mean_death_TX,feb_month_mean_death_TX,states_df_out[['TX␣
 ↪deaths']].values,march_month_data)
print("Z test Result of State TX under death category is : " +␣
 ↪str(z_test_result_death_TX))
if(z_test_result_death_TX > 1.962):
    print("Z test Result is Z =" + str(z_test_result_death_TX) + " which is␣
 ↪greater than z_alpha/2 = 1.962 so reject the NULL hypothesis")
else:
    print("Z test Result is Z =" + str(z_test_result_death_TX) + " which is less␣
 ↪than z_alpha/2 = 1.962 so accept the NULL hypothesis")

print()

#Category : Cases
z_test_result_cases_TX =␣
 ↪z_testing(march_month_mean_cases_TX,feb_month_mean_cases_TX,states_df_out[['TX␣
 ↪confirmed']].values,march_month_data)
print("Z test Result of State TX under Cases category is : " +␣
 ↪str(z_test_result_cases_TX))
if(z_test_result_cases_TX > 1.962):
    print("Z test Result is Z = " + str(z_test_result_cases_TX) + " which is␣
 ↪greater than z_alpha/2 = 1.962 so reject the NULL hypothesis");
else:
    print("Z test Result is Z = " + str(z_test_result_cases_TX) + " which is␣
 ↪less than z_alpha/2 = 1.962 so accept the NULL hypothesis")
```

```
___Z Test Result for State : TX___

Z test Result of State TX under death category is : [4.16203793]
Z test Result is Z =[4.16203793] which is greater than z_alpha/2 = 1.962 so
reject the NULL hypothesis

Z test Result of State TX under Cases category is : [4.58422569]
Z test Result is Z = [4.58422569] which is greater than z_alpha/2 = 1.962 so
reject the NULL hypothesis
```

3.6 One Sample T test

[36]:
```
#One Sample T test
def t_one_testing(march_data_mean,feb_data_mean,march_data):
    # Computing t statistic = (x_bar - guess) / (corrected_std / root of (n))
    guess = feb_data_mean
    x_bar = march_data_mean
    corrected_std = compute_corrected_std(march_data)
    num = (x_bar - guess)
    den = corrected_std / np.sqrt(len(march_data))
    t_stats = num / den
    return np.abs(t_stats);
```

[37]:
```
#One Sample T Test Result for State : TN
print("___One Sample T Test Result for State : TN___")
print()

#Category : Death
t_one_test_result_death_TN =␣
 ↪t_one_testing(march_month_mean_death_TN,feb_month_mean_death_TN,march_month_data[['TN␣
 ↪deaths']].values)
print("One Sample T Test Result of State TN under death category is : " +␣
 ↪str(t_one_test_result_death_TN))
if(t_one_test_result_death_TN > 2.042):
    print("One Sample T Test Result is T =" + str(t_one_test_result_death_TN) +␣
 ↪" which is greater than t(30,0.025) = 2.042 so reject the NULL hypothesis")
else:
    print("One Sample T Test Result is T =" + str(t_one_test_result_death_TN) +␣
 ↪" which is less than t(30,0.025) = 2.042 so accept the NULL hypothesis")

print()

#Category : Cases
t_one_test_result_cases_TN =␣
 ↪t_one_testing(march_month_mean_cases_TN,feb_month_mean_cases_TN,march_month_data[['TN␣
 ↪confirmed']].values)
print("One Sample T Test Result of State TN under Cases category is : " +␣
 ↪str(t_one_test_result_cases_TN))
if(t_one_test_result_cases_TN > 2.042):
    print("One Sample T Test Result is T =" + str(t_one_test_result_cases_TN) +␣
 ↪" which is greater than t(30,0.025) = 2.042 so reject the NULL hypothesis");
else:
    print("One Sample T Test Result is T =" + str(t_one_test_result_cases_TN) +␣
 ↪" which is less than t(30,0.025) = 2.042 so accept the NULL hypothesis")
```

```
___One Sample T Test Result for State : TN___

One Sample T Test Result of State TN under death category is : [9.16571475]
One Sample T Test Result is T =[9.16571475] which is greater than t(30,0.025) =
2.042 so reject the NULL hypothesis

One Sample T Test Result of State TN under Cases category is : [1.91472211]
One Sample T Test Result is T =[1.91472211] which is less than t(30,0.025) =
2.042 so accept the NULL hypothesis
```

[38]:
```
#One Sample T Test Result for State : TX
print("___One Sample T Test Result for State : TX___")
print()

#Category : Death
t_one_test_result_death_TX =␣
↪t_one_testing(march_month_mean_death_TX,feb_month_mean_death_TX,march_month_data[['TX␣
↪deaths']].values)
print("One Sample T Test Result of State TX under death category is : " +␣
↪str(t_one_test_result_death_TX))
if(t_one_test_result_death_TX > 2.042):
    print("One Sample T Test Result is T =" + str(t_one_test_result_death_TX) +␣
↪" which is greater than t(30,0.025) = 2.042 so reject the NULL hypothesis")
else:
    print("One Sample T Test Result is T =" + str(t_one_test_result_death_TX) +␣
↪" which is less than t(30,0.025) = 2.042 so accept the NULL hypothesis")

print()

#Category : Cases
t_one_test_result_cases_TX =␣
↪t_one_testing(march_month_mean_cases_TX,feb_month_mean_cases_TX,march_month_data[['TX␣
↪confirmed']].values)
print("One Sample T Test Result of State TX under Cases category is : " +␣
↪str(t_one_test_result_cases_TX))
if(t_one_test_result_cases_TX > 2.042):
    print("One Sample T Test Result is T =" + str(t_one_test_result_cases_TX) +␣
↪" which is greater than t(30,0.025) = 2.042 so reject the NULL hypothesis");
else:
    print("One Sample T Test Result is T =" + str(t_one_test_result_cases_TX) +␣
↪" which is less than t(30,0.025) = 2.042 so accept the NULL hypothesis")
```

```
___One Sample T Test Result for State : TX___

One Sample T Test Result of State TX under death category is : [4.38835132]
One Sample T Test Result is T =[4.38835132] which is greater than t(30,0.025) =
2.042 so reject the NULL hypothesis

One Sample T Test Result of State TX under Cases category is : [9.807289]
One Sample T Test Result is T =[9.807289] which is greater than t(30,0.025) =
2.042 so reject the NULL hypothesis
```

3.7 Two sample Walds test

[39]:
```
##Two sample Walds test
def walds_two_sample_testing(march_data_mean,feb_data_mean,march_data,feb_data):
    mu_y = feb_data_mean
    mu_x = march_data_mean
    std_error = np.sqrt((march_data_mean / len(march_data)) + (feb_data_mean /␣
 ↪len(feb_data)))
    num = mu_x - mu_y
    w_two_sample_stats = num / std_error
    return np.abs(w_two_sample_stats)
```

[40]:
```
#Walds Two Sample Test Result for State : TN
print("___Walds Two Sample Test Result for State : TN___")
print()

#Category : Death
walds_two_result_death_TN =␣
 ↪walds_two_sample_testing(march_month_mean_death_TN,feb_month_mean_death_TN,march_month_data
print("Walds Two Sample Test Result of State TN under death category is : " +␣
 ↪str(walds_two_result_death_TN))
if(walds_two_result_death_TN > 1.962):
    print("Walds Two Sample Test Result for mean of death is W2 =␣
 ↪"+str(walds_two_result_death_TN) +" which is greater than z_alpha/2 = 1.962␣
 ↪so reject the NULL hypothesis")
else:
    print("Walds Two Sample Test Result for mean of death is  W2 =␣
 ↪"+str(walds_two_result_death_TN)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")

print()

#Category : Cases
walds_two_result_cases_TN =␣
 ↪walds_two_sample_testing(march_month_mean_cases_TN,feb_month_mean_cases_TN,march_month_data
print("Walds Two Sample Test Result of State TN under Cases category is : " +␣
 ↪str(walds_two_result_cases_TN))
if(walds_two_result_cases_TN > 1.962):
    print("Walds Two Sample Test Result for mean of cases is W2 = "␣
 ↪+str(walds_two_result_cases_TN) +" which is greater than z_alpha/2 = 1.962 so␣
 ↪reject the NULL hypothesis");
else:
    print("Walds Two Sample Test Result for mean of cases is  W2 = "␣
 ↪+str(walds_two_result_cases_TN)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")
```

```
___Walds Two Sample Test Result for State : TN___
```

```
Walds Two Sample Test Result of State TN under death category is :
14.766383456665165
Walds Two Sample Test Result for mean of death is W2 = 14.766383456665165 which
is greater than z_alpha/2 = 1.962 so reject the NULL hypothesis

Walds Two Sample Test Result of State TN under Cases category is :
28.34711545617562
Walds Two Sample Test Result for mean of cases is W2 = 28.34711545617562 which
is greater than z_alpha/2 = 1.962 so reject the NULL hypothesis
```

[41]:
```
#Walds Two Sample Test Result for State : TX
print("___Walds Two Sample Test Result for State : TX___")
print()

#Category : Death
walds_two_result_death_TX =␣
 ↪walds_two_sample_testing(march_month_mean_death_TX,feb_month_mean_death_TX,march_month_data
print("Walds Two Sample Test Result of State TX under death category is : " +␣
 ↪str(walds_two_result_death_TX))
if(walds_two_result_death_TX > 1.962):
    print("Walds Two Sample Test Result for mean of death is w =␣
 ↪"+str(walds_two_result_death_TX) +" which is greater than z_alpha/2 = 1.962␣
 ↪so reject the NULL hypothesis")
else:
    print("Walds Two Sample Test Result for mean of death is  w =␣
 ↪"+str(walds_two_result_death_TX)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")

print()

#Category : Cases
walds_two_result_cases_TX =␣
 ↪walds_two_sample_testing(march_month_mean_cases_TX,feb_month_mean_cases_TX,march_month_data
print("Walds Two Sample Test Result of State TX under Cases category is : " +␣
 ↪str(walds_two_result_cases_TX))
if(walds_two_result_cases_TX > 1.962):
    print("Walds Two Sample Test Result for mean of cases is w = "␣
 ↪+str(walds_two_result_cases_TX) +" which is greater than z_alpha/2 = 1.962 so␣
 ↪reject the NULL hypothesis");
else:
    print("Walds Two Sample Test Result for mean of cases is  w = "␣
 ↪+str(walds_two_result_cases_TX)+ " which is less than z_alpha/2 = 1.962 so␣
 ↪accept the NULL hypothesis")
```

```
___Walds Two Sample Test Result for State : TX___
```

```
Walds Two Sample Test Result of State TX under death category is :
17.88232836525732
Walds Two Sample Test Result for mean of death is w = 17.88232836525732 which is
greater than z_alpha/2 = 1.962 so reject the NULL hypothesis

Walds Two Sample Test Result of State TX under Cases category is :
148.41581668839095
Walds Two Sample Test Result for mean of cases is w = 148.41581668839095 which
is greater than z_alpha/2 = 1.962 so reject the NULL hypothesis
```

3.8 Two Sample T unpaired testing

Note :

total degrees of freedom = 21+31-1 = 50

[42]:
```
#Two Sample t unpaired testing:
def␣
↪unpaired_t_test_two_sample_testing(march_data_mean,feb_data_mean,march_data,feb_data):
↪
    y_bar = feb_data_mean
    x_bar = march_data_mean
    d_bar = x_bar - y_bar
    x_var = np.square(compute_corrected_std(march_data))
    y_var = np.square(compute_corrected_std(feb_data))
    std_dev = np.sqrt((x_var / len(march_data)) + (y_var / len(feb_data)))
    t_stats_unpaired = d_bar / std_dev
    return abs(t_stats_unpaired)
```

[43]:
```
#Two Sample t unpaired Test Result for State : TN
print("___Two Sample t unpaired Test Result for State : TN___")
print()

#Category : Death
t_two_unpaired_test_result_death_TN =␣
↪unpaired_t_test_two_sample_testing(march_month_mean_death_TN,feb_month_mean_death_TN,march_
↪deaths']].values,feb_month_data[['TN deaths']].values)
print("Two Sample t unpaired Test Result of State TN under death category is : "␣
↪+ str(t_two_unpaired_test_result_death_TN))
if(t_two_unpaired_test_result_death_TN > 2.01):
    print("Two Sample t unpaired Test Result is T2 =" +␣
↪str(t_two_unpaired_test_result_death_TN) + " which is greater than t(51,0.
↪025) = 2.01 so reject the NULL hypothesis")
else:
    print("Two Sample t unpaired Test Result is T2 =" +␣
↪str(t_two_unpaired_test_result_death_TN) + " which is less than t(51,0.025) =␣
↪2.01 so accept the NULL hypothesis")
```

```
print()

#Category : Cases
t_two_unpaired_test_result_cases_TN=␣
↪unpaired_t_test_two_sample_testing(march_month_mean_cases_TN,feb_month_mean_cases_TN,march_
↪confirmed']].values,feb_month_data[['TN confirmed']].values)
print("Two Sample t unpaired Test Result of State TN under Cases category is : "␣
↪+ str(t_two_unpaired_test_result_cases_TN))
if(t_two_unpaired_test_result_cases_TN > 2.01):
    print("Two Sample t unpaired Test Result is T2 =" +␣
↪str(t_two_unpaired_test_result_cases_TN) + " which is greater than t(51,0.
↪025) = 2.01 so reject the NULL hypothesis");
else:
    print("Two Sample t unpaired Test Result is T2 =" +␣
↪str(t_two_unpaired_test_result_cases_TN) + " which is less than t(51,0.025) =␣
↪2.01 so accept the NULL hypothesis")
```

```
___Two Sample t unpaired Test Result for State : TN___

Two Sample t unpaired Test Result of State TN under death category is :
[3.55075078]
Two Sample t unpaired Test Result is T2 =[3.55075078] which is greater than
t(51,0.025) = 2.01 so reject the NULL hypothesis

Two Sample t unpaired Test Result of State TN under Cases category is :
[1.49354506]
Two Sample t unpaired Test Result is T2 =[1.49354506] which is less than
t(51,0.025) = 2.01 so accept the NULL hypothesis
```

[44]:
```
#Two Sample t unpaired Test Result for State : TX
print("___Two Sample t unpaired Test Result for State : TX___")
print()

#Category : Death
t_two_unpaired_test_result_death_TX =␣
↪unpaired_t_test_two_sample_testing(march_month_mean_death_TX,feb_month_mean_death_TX,march_
↪deaths']].values,feb_month_data[['TX deaths']].values)
print("Two Sample t unpaired Test Result of State TX under death category is : "␣
↪+ str(t_two_unpaired_test_result_death_TX))
if(t_two_unpaired_test_result_death_TX > 2.01):
    print("Two Sample t unpaired Test Result is T =" +␣
↪str(t_two_unpaired_test_result_death_TX) + " which is greater than t(51,0.
↪025) = 2.01 so reject the NULL hypothesis")
else:
    print("Two Sample t unpaired Test Result is T =" +␣
 ↪str(t_two_unpaired_test_result_death_TX) + " which is less than t(51,0.025) =␣
 ↪2.01 so accept the NULL hypothesis")

print()

#Category : Cases
t_two_unpaired_test_result_cases_TX =␣
 ↪unpaired_t_test_two_sample_testing(march_month_mean_cases_TX,feb_month_mean_cases_TX,march_
 ↪confirmed']].values,feb_month_data[['TX confirmed']].values)
print("Two Sample t unpaired Test Result of State TX under Cases category is : "␣
 ↪+ str(t_two_unpaired_test_result_cases_TX))
if(t_two_unpaired_test_result_cases_TX > 2.01):
    print("Two Sample t unpaired Test Result is T =" +␣
 ↪str(t_two_unpaired_test_result_cases_TX) + " which is greater than t(51,0.
 ↪025) = 2.01 so reject the NULL hypothesis");
else:
    print("Two Sample t unpaired Test Result is T =" +␣
 ↪str(t_two_unpaired_test_result_cases_TX) + " which is less than t(51,0.025) =␣
 ↪2.01 so accept the NULL hypothesis")
```

```
___Two Sample t unpaired Test Result for State : TX___

Two Sample t unpaired Test Result of State TX under death category is :
[2.44514278]
Two Sample t unpaired Test Result is T =[2.44514278] which is greater than
t(51,0.025) = 2.01 so reject the NULL hypothesis

Two Sample t unpaired Test Result of State TX under Cases category is :
[3.73003452]
Two Sample t unpaired Test Result is T =[3.73003452] which is greater than
t(51,0.025) = 2.01 so reject the NULL hypothesis
```

3.9 Which Tests are applicable ??

One Sampling T test, Two sample Unpaired T tests are not applicable for the above. Its given that daily data is Poisson distribution. But we know that T tests is only applicable on Normal Distribution. Hence One Sampling T test, Two sample Unpaired T tests are not applicable here

Wald's Test(one sample and two sample) and Z test are applicable here. Based on CLT, when $n > 30$, sample mean is Asymptotically Normal. In our data set, n is greater than 30 which imples mean is Asymptotically Normal. We know that Wald's Test and Z test are applicable for Asymptotically Normal data set Hence Wald's Test(one sample and two sample) and Z test are applicable here.

4 Required Inference 3 - KS and Permutation Test

Given Task : Inference the equality of distributions in the two states (distribution of daily #cases and daily #deaths) for the last three months of 2020 (Oct, Nov, Dec) of your dataset using K-S test and Permutation test. For the K-S test, use both 1-sample and 2-sample tests. For the 1-sample test, try Poisson, Geometric, and Binomial. To obtain parameters of these distributions to check against in 1-sample KS, use MME on the Oct-Dec 2020 data of the first state in your dataset to obtain parameters of the distribution, and then check whether the Oct-Dec 2020 data for the second state in your dataset has the distribution with the obtained MME parameters. For the permutation test, use 1000 permutations. Use a threshold of 0.05 for both K-S test and Permutation test.

[45]:
```
#Obtaining the data for TN and TX confirmed cases on a daily basis dated from 10/
↪1/2020 to 12/28/2020
TN_cases = np.array(states_df_out['TN confirmed'])[251:312]
TX_cases = np.array(states_df_out['TX confirmed'])[251:312]
```

[46]:
```
#Obtaining the data for TN and TX confirmed cases on a daily basis dated from 10/
↪1/2020 to 12/28/2020
TN_deaths = np.array(states_df_out['TN deaths'])[251:312]
TX_deaths = np.array(states_df_out['TX deaths'])[251:312]
```

4.1 Permutation Test

For TN confirmed cases and TX confirmed cases:

Null hypothesis (H0):

Distribution of TN's daily confirmed cases equals distribution of TX's daily confirmed cases from Oct-Dec 2020.

Alternate hypothesis(H1):

Distribution of TN's daily confirmed cases not equals distribution of TX's daily confirmed cases from Oct-Dec 2020.

For TN deaths and TX deaths:

Null hypothesis (H0):

Distribution of TN's daily deaths equals distribution of TX's daily deaths from Oct-Dec 2020.

Alternate hypothesis(H1):

Distribution of TN's daily deaths not equals distribution of TX's daily deaths from Oct-Dec 2020.

[47]:
```
def permutation_test(TN, TX):

    c = 0.05

    def permutation_sample(data1, data2):
        data = np.concatenate((data1, data2))
        permuted_data = np.random.permutation(data)
        perm_sample_1 = permuted_data[:len(data1)]
        perm_sample_2 = permuted_data[len(data1):]
        return perm_sample_1, perm_sample_2

    def draw_perm_reps(data_1, data_2, size):
        perm_replicates = np.empty(size)
        for i in range(size):
            perm_sample_1, perm_sample_2 = permutation_sample(data_1, data_2)
            perm_replicates[i] = abs(np.mean(perm_sample_1) - np.
↪mean(perm_sample_2))
        return perm_replicates


    T_obs_b = abs(np.mean(TN) - np.mean(TX))
    T_obs_b

    print("T-Observed is:", T_obs_b)

    #Using 1000 permutations
    for n in [1000]:
        perm_replicates = draw_perm_reps(TN,TX,n)
        p_b = np.sum(perm_replicates >= T_obs_b)/len(perm_replicates)
        print("n:",n," p-value is:", p_b)

    if(p_b <= c):
          print("Hence, we REJECT the Null Hypothesis")
    else:
          print("Hence, we ACCEPT the Null Hypothesis")
```

Permutation Test - TN confirmed cases and TX confirmed cases

[48]:
```
permutation_test(TN_cases,TX_cases)
```

```
T-Observed is: 4556.377049180328
n: 1000  p-value is: 0.0
Hence, we REJECT the Null Hypothesis
```

Since the obtained p-value is less than the threshold value of 0.05, we REJECT the Null Hypothesis.

Permutation Test - TN confirmed deaths and TX confirmed deaths

[49]:
```
permutation_test(TN_deaths,TX_deaths)
```

```
T-Observed is: 66.19672131147541
n: 1000  p-value is: 0.0
Hence, we REJECT the Null Hypothesis
```

Since the obtained p-value is less than the threshold value of 0.05, we REJECT the Null Hypothesis.

4.2 2 SAMPLE K-S TEST

[50]:
```
# Getting the e-CDF
def eCDF(A):
    n = len(A)
    Sort = sorted(A)
    delta = .1
    A = []
    B = [0]
    for i in range(0,n):
        A = A + [Sort[i]]
        B = B + [B[len(B)-1]+(1/n)]
    B = B + [1]

    return A,B
```

[51]:
```
def KS_2_sample(A1,B1, A2,B2):
    matrix = np.zeros((len(A1),6))
    max_total = -1
    for i in range(len(matrix)):
        matrix[i,0] = B1[i]
        matrix[i,1] = B1[i+1]
        index1 = [idx for idx, val in enumerate(A2) if val >= A1[i]]
        index2 = [idx for idx, val in enumerate(A2) if val < A1[i]]
        if index1 == []:
            matrix[i,3] = 1
        else :
            matrix[i,3] = B2[index1[0]]
        if index2 == []:
            matrix[i,2] = 0
        else:
            matrix[i,2] = B2[index2[-1]]

        matrix[i,4] = abs( matrix[i,0] - matrix[i,2])
        matrix[i,5] = abs(matrix[i,1] - matrix[i,3])
        cmax = max(matrix[i,4], matrix[i,5])
        if cmax > max_total:
            max_total = cmax
            a1_max = A1[i]
            b1_max = matrix[i,0]
            b2_max = matrix[i,2]

    return max_total
```

2 Sample K-S Test - TN confirmed cases and TX confirmed cases

```
[52]: TN_cases = np.array(states_df_out['TN confirmed'])[251:312]
      TX_cases = np.array(states_df_out['TX confirmed'])[251:312]

      #Computing eCDF for TN confirmed cases and TX confirmed cases
      A1, B1 = eCDF(TN_cases)
      A2, B2 = eCDF(TX_cases)

      KS_val = KS_2_sample(A1,B1, A2,B2)

      print('KS statistic : ', KS_val)

      c=0.05

      if(KS_val > c):
          print("Hence, we REJECT the Null Hypothesis")
      else:
          print("Hence, we ACCEPT the Null Hypothesis")
```

```
KS statistic :  0.7049180327868858
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

2 Sample K-S Test - TN deaths and TX deaths

```
[53]: TN_deaths = np.array(states_df_out['TN deaths'])[251:312]
      TX_deaths = np.array(states_df_out['TX deaths'])[251:312]

      #Computing eCDF for TN deaths and TX deaths
      A1, B1 = eCDF(TN_deaths)
      A2, B2 = eCDF(TX_deaths)

      KS_val = KS_2_sample(A1,B1, A2,B2)

      print('KS statistic : ', KS_val)

      c=0.05

      if(KS_val > c):
          print("Hence, we REJECT the Null Hypothesis")
      else:
          print("Hence, we ACCEPT the Null Hypothesis")
```

```
KS statistic :  0.6557377049180335
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

4.3 1 SAMPLE K-S TEST

[54]:
```
def KS_1_sample(A1,B1, CDF, parameter):
    max_total = -1

    matrix = np.zeros((len(A1),4))
    for i in range(len(matrix)):
        matrix[i,0] = B1[i]
        matrix[i,1] = B1[i+1]
        Fx = CDF(parameter, A1[i])
        matrix[i,2] = abs(matrix[i,0] - Fx)
        matrix[i,3] = abs(matrix[i,1] - Fx)
        cmax = max(matrix[i,2], matrix[i,3])
        if cmax > max_total:
            max_total = cmax


    return max_total
```

[55]:
```
# Obtaining eCDF for TX confirmed cases
test_A, test_B = eCDF(TX_cases)
```

[56]:
```
# Obtaining eCDF for TX deaths
test_P, test_Q = eCDF(TX_deaths)
```

1 SAMPLE K-S TEST - POISSON DISTRIBUTION

Obtaining parameters for Poisson distribution

[57]:
```
def Poisson_MME(X):
    poiss_mme = np.mean(X)
    return poiss_mme

def Poisson_CDF(lambda_, x):
    poiss_cdf = poisson.cdf(x, lambda_)
    return poiss_cdf
```

TN confirmed cases and TX confirmed cases

[58]:
```
# Obtaining MME parametersfor TN confirmed cases
MME_TNconfirmedcases = Poisson_MME(TN_cases)
print('Poisson parameter(lambda) : ', MME_TNconfirmedcases)

# 1 sample KS-test on TX confirmed cases

KS_val = KS_1_sample(test_A, test_B, Poisson_CDF, MME_TNconfirmedcases )

print('KS statistic : ', KS_val)
```

```
c=0.05

if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Poisson parameter(lambda) :  2664.27868852459
KS statistic :  0.9344262295080024
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

TN deaths and TX deaths

[59]:
```
# Obtaining MME parameters for TN deaths
MME_TNdeaths = Poisson_MME(TN_deaths)
print('Poisson parameter(lambda) : ', MME_TNdeaths)

# 1 sample KS-test on TX deaths

KS_val = KS_1_sample(test_P, test_Q, Poisson_CDF, MME_TNdeaths )

print('KS statistic : ', KS_val)

c=0.05

if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Poisson parameter(lambda) :  35.49180327868852
KS statistic :  0.7851347244191668
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

1 SAMPLE K-S TEST - BINOMIAL DISTRIBUTION

Obtaining parameters for Binomial distribution

[60]:
```
def Binomial_param(X):
    mean = np.mean(X)
    n_estimate = np.square(mean)/(mean-np.var(X))
    p_estimate = mean/n_estimate
    return n_estimate,p_estimate

def Binomial_CDF(params,x):
    prob = binom.cdf(x, params[0], params[1])
    return prob
```

TN confirmed cases and TX confirmed cases

[61]:
```
# Obtaining MME parameters for TN confirmed cases
n,p = Binomial_param(TN_cases)

print('Parameters of Binomial Distribution (n,p) : ', n,p)

# 1 sample KS-test on TX confirmed cases
KS_val = KS_1_sample(test_A, test_B, Binomial_CDF, [n,p] )

print('KS statistic : ', KS_val)

c=0.05

if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Parameters of Binomial Distribution (n,p) :  -4.833191310504729
-551.2462713267521
KS statistic :  1.0
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

TN deaths and TX deaths

[62]:
```
# Obtaining MME parameters for TN deaths

n,p = Binomial_param(TN_deaths)

print('Parameters of Binomial Distribution (n,p) : ', n,p)

# 1 sample KS-test on TX deaths
KS_val = KS_1_sample(test_P, test_Q, Binomial_CDF, [n,p] )

print('KS statistic : ', KS_val)

c=0.05
```

```python
if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Parameters of Binomial Distribution (n,p) :  -2.675828670695916
-13.263854919925794
KS statistic :  1.0
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

1 SAMPLE K-S TEST - GEOMETRIC DISTRIBUTION

Obtaining parameters for Geometric distribution

[63]:
```python
def Geometric_MME(X):
    sample_mean = np.mean(X)
    geo_mme = 1/sample_mean
    return geo_mme

def Geometric_CDF(p,x):
    geo_cdf = geom.cdf(x, p)
    return geo_cdf
```

TN confirmed cases and TX confirmed cases

[64]:
```python
# Obtaining MME parameters for TN confirmed cases

p = Geometric_MME(TN_cases)

print('Geometric parameter : ', p)

# 1 sample KS-test on TX confirmed cases
KS_val = KS_1_sample(test_A, test_B, Geometric_CDF, p )

print('KS statistic : ', KS_val)

c=0.05

if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Geometric parameter :  0.000375336110410347
```

```
KS statistic :  0.6395840847515359
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

TN deaths and TX deaths

[65]:
```python
# Obtaining MME parameters for TN deaths

p = Geometric_MME(TN_deaths)

print('Geometric parameter : ', p)

# 1 sample KS-test on TX deaths
KS_val = KS_1_sample(test_P, test_Q, Geometric_CDF, p )

print('KS statistic : ', KS_val)

c=0.05

if(KS_val > c):

    print("Hence, we REJECT the Null Hypothesis")
else:
    print("Hence, we ACCEPT the Null Hypothesis")
```

```
Geometric parameter :  0.028175519630484991
KS statistic :  0.5645215235717812
Hence, we REJECT the Null Hypothesis
```

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

5 Required Inference 4 - Bayesian Inference

Given Task :For this task, sum up the daily stats (cases and deaths) from both states. Assume day 1 is June 1st 2020. Assume the combined daily deaths are Poisson distributed with parameter . Assume an Exponential prior (with mean) on . Assume = MME where the MME is found using the first four weeks data (so the first 28 days of June 2020) as the sample data. Now, use the fifth week's data (June 29 to July 5) to obtain the posterior for via Bayesian inference. Then, use sixth week's data to obtain the new posterior, using prior as posterior after week 5. Repeat till the end of week 8 (that is, repeat till you have posterior after using 8th week's data). Plot all posterior distributions on one graph. Report the MAP for all posteriors

Since the obtained KS statistic value is greater than the critical value of 0.05, we REJECT the Null Hypothesis.

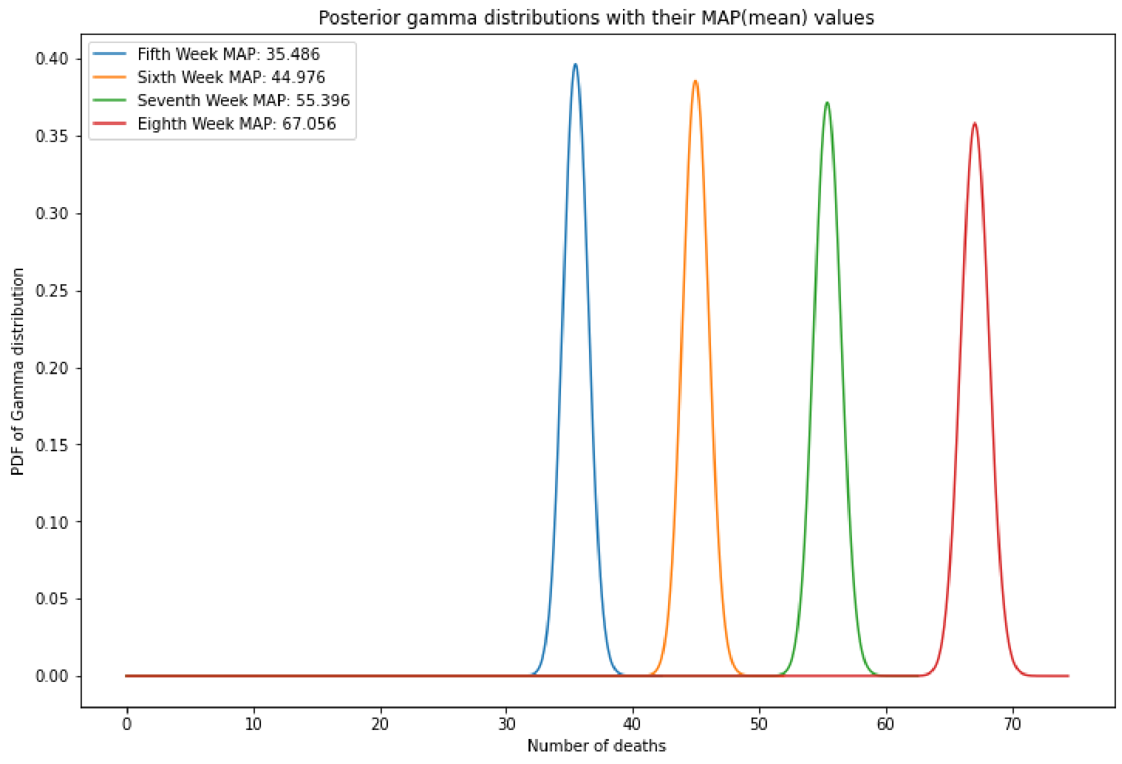
[66]:
```
pd.options.mode.chained_assignment = None  # default='warn'

states_df_out.loc[:,'total_death'] = states_df_out.loc[:,'TN deaths'] +␣
↪states_df_out.loc[:,'TX deaths']
states_df_out
```

[66]:
```
           Date  TN confirmed  TX confirmed  TN deaths  TX deaths  total_death
0    2020-01-22           0.0           0.0        0.0        0.0          0.0
1    2020-01-23           0.0           0.0        0.0        0.0          0.0
2    2020-01-24           0.0           0.0        0.0        0.0          0.0
3    2020-01-25           0.0           0.0        0.0        0.0          0.0
4    2020-01-26           0.0           0.0        0.0        0.0          0.0
..          …             …             …          …          …            …
433  2021-03-30         840.0        3672.0       29.0      109.0        138.0
434  2021-03-31        1313.0         122.0       10.0        0.0         10.0
435  2021-04-01        1770.0        7424.0       14.0      243.0        257.0
436  2021-04-02           0.0        3275.0        0.0      115.0        115.0
437  2021-04-03           0.0        1689.0        0.0       86.0         86.0

[376 rows x 6 columns]
```

[67]:
```
#Extracting the Data for month of June and July

week8_data = states_df_out[states_df_out['Date']>='2020-06-01']
week8_data = week8_data[week8_data['Date']<='2020-07-31']
week8_data = week8_data[:7*8]
```

[68]:
```
date=list(week8_data['Date'])
deaths=list(week8_data['total_death'])

def plot_gamma(alpha=1, beta=1, label="0"):
    #print((alpha-1)/beta)
    x = np.linspace(gamma.ppf(0, alpha, scale=1/beta),gamma.ppf(0.9999999999,␣
↪alpha, scale=1/beta), 1000)

    plt.title("Posterior gamma distributions with their MAP(mean) values")
    label= label + "MAP: " + str(round(((alpha-1)/beta),3))
    plt.plot(x, gamma.pdf(x, alpha, scale=1/beta), label=label)
    plt.xlabel("Number of deaths")
    plt.ylabel("PDF of Gamma distribution")
    plt.legend()

plt.figure(figsize=(12,8))
plot_gamma(sum(deaths[:35])+1,len(deaths[:35]) + len(deaths[:35])/sum(deaths[:
↪35]),'Fifth Week ')
plot_gamma(sum(deaths[:42])+1,len(deaths[:42]) + len(deaths[:42])/sum(deaths[:
↪42]),'Sixth Week ')
```

```
plot_gamma(sum(deaths[:48])+1,len(deaths[:48]) + len(deaths[:48])/sum(deaths[:
↪48]),'Seventh Week ')
plot_gamma(sum(deaths[:54])+1,len(deaths[:54]) + len(deaths[:54])/sum(deaths[:
↪54]),'Eighth Week ')
plt.show()
```



From the above plot we can find the MAP values for the Lambda for the Fifth to Eighth week. So we can see that as we add more data to the model, the MAP values are increasing as the total number of deaths are increasing.

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

exploratory-3

May 14, 2021

1 Exploratory Task 3

In this task, we are initially checking for missing/null values in the US-All datasets and the X dataset.

There is a date format difference between US-All datasets and our X (Homelessness) dataset. Our X dataset had the format mm/dd/yyyy whereas the US-All dataset has yyyy-mm-dd. We have converted X dataset format to yyyy-mm-dd.

Note:

X Data Set is about NYC Department of Homeless Services Daily Report

X DataSet Link : https://data.world/ian/nyc-department-of-homeless-services-daily-report

[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
us_confirmed = pd.read_csv("US-all/US_confirmed.csv")
us_deaths = pd.read_csv("US-all/US_deaths.csv")
homeless = pd.read_csv("US-all/DHS_Daily_Report.csv")
```

[3]:
```
#Checking for null values in dataset

print(us_confirmed.isnull().values.any())
print(us_deaths.isnull().values.any())
print(homeless.isnull().values.any())
```

```
False
False
False
```

There are **no missing values** in any of the datasets.

[4]:
```
#Converting date in homeless dataset to consistent format - yyyy-mm-dd, and␣
 ↪sorting it in descending order

homeless['Date of Census'] = pd.to_datetime(homeless['Date of Census'])
```

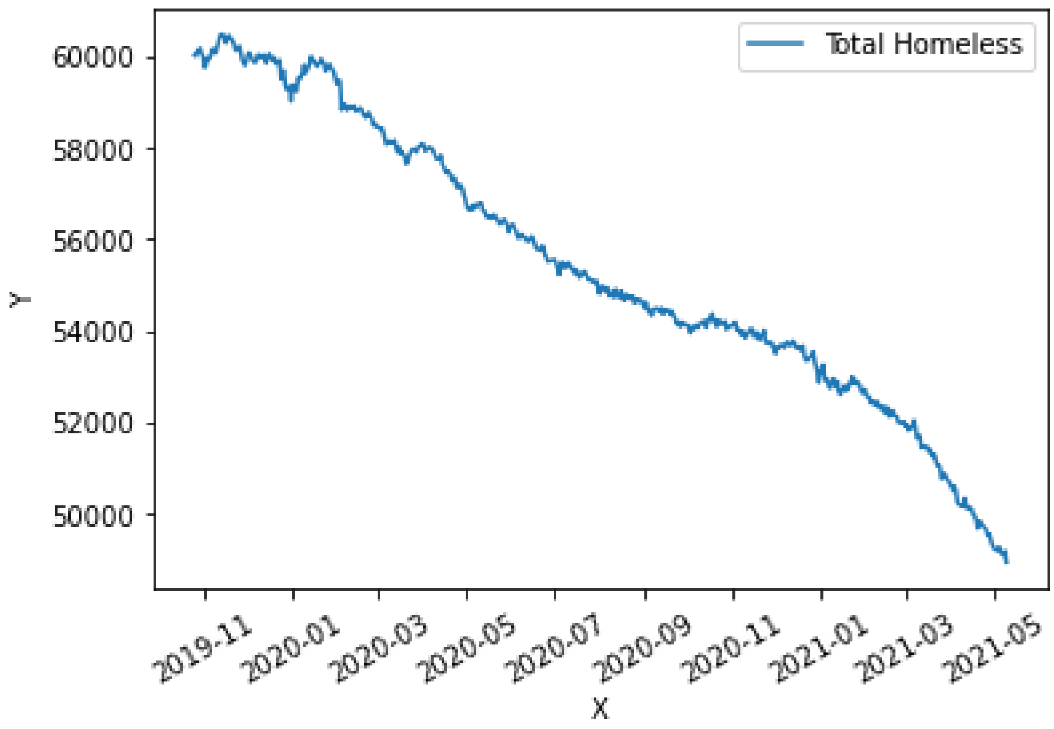
```
homeless.sort_values(by=['Date of Census'], inplace=True, ascending=False)
```

```
[5]: homeless.head()
```

```
[5]:      Date of Census  Total Adults in Shelter  Total Children in Shelter  \
     2792     2021-05-10                    33430                      15504
     2791     2021-05-09                    33565                      15619
     2790     2021-05-08                    33466                      15630
     2789     2021-05-07                    33511                      15606
     2788     2021-05-06                    33560                      15587

           Total Individuals in Shelter  Single Adult Men in Shelter  \
     2792                         48934                        13565
     2791                         49184                        13614
     2790                         49096                        13479
     2789                         49117                        13490
     2788                         49147                        13551

           Single Adult Women in Shelter  Total Single Adults in Shelter  \
     2792                           4361                           17926
     2791                           4346                           17960
     2790                           4360                           17839
     2789                           4377                           17867
     2788                           4389                           17940

           Families with Children in Shelter  \
     2792                               8920
     2791                               8981
     2790                               8994
     2789                               8999
     2788                               8985

           Adults in Families with Children in Shelter  \
     2792                                        11728
     2791                                        11814
     2790                                        11833
     2789                                        11844
     2788                                        11828

           Children in Families with Children in Shelter  \
     2792                                          15504
     2791                                          15619
     2790                                          15630
     2789                                          15606
     2788                                          15587

           Total Individuals in Families with Children in Shelter   \
     2792                                                 27232
     2791                                                 27433
     2790                                                 27463
     2789                                                 27450
     2788                                                 27415

           Adult Families in Shelter  Individuals in Adult Families in Shelter
     2792                       1795                                      3776
     2791                       1802                                      3791
     2790                       1804                                      3794
     2789                       1807                                      3800
     2788                       1803                                      3792
```

[6]:
```python
def plotGraph(date,data):
    plt.plot(date,data,label="Total Homeless")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend(loc='upper right')
    plt.xticks(rotation=30)
    plt.show()
```

[7]:
```python
date = homeless['Date of Census'][:600]
individualsInShelter = homeless['Total Individuals in Shelter'][:600]
plotGraph(date,individualsInShelter)
```



As we are using entire US data, and don't need individual state level data for our X dataset, we have added the values in US_deaths datasets to get the total deaths and added it as a new column.

```
[8]: us_deaths_transposed = us_deaths.T
     new_header = us_deaths_transposed.iloc[0] #grab the first row for the header
     us_deaths_transposed= us_deaths_transposed[1:] #take the data less the header row
     us_deaths_transposed.columns = new_header #set the header row as the df header
     us_deaths_transposed['total_death'] = us_deaths_transposed.sum(axis =1)
```

```
[9]: date = us_deaths_transposed.index
     us_deaths_transposed['date'] = date
```

The data given in US_deaths is cumulative, so we have converted it into per day stats.

```
[10]: us_deaths_transposed = us_deaths_transposed.set_index('date').diff()
```

```
[11]: us_deaths_transposed = us_deaths_transposed.reset_index()
```

As we are using entire US data, and don't need individual state level data for our X dataset, we have added the values in US_confirmed datasets to get the total confirmed cases and added it as a new column.

```
[12]: us_confirmed_transposed = us_confirmed.T
      new_header = us_confirmed_transposed.iloc[0] #grab the first row for the header
      us_confirmed_transposed= us_confirmed_transposed[1:] #take the data less the␣
       ↪header row
      us_confirmed_transposed.columns = new_header #set the header row as the df header
      us_confirmed_transposed['total_confirmed_cases'] = us_confirmed_transposed.
       ↪sum(axis =1)
```

```
[13]: date = us_confirmed_transposed.index
      us_confirmed_transposed['date'] = date
```

The data given in US_deaths is cumulative, so we have converted it into per day stats.

```
[14]: us_confirmed_transposed = us_confirmed_transposed.set_index('date').diff()
```

```
[15]: us_confirmed_transposed = us_confirmed_transposed.reset_index()
```

1.1 Inference 1 : Pearson Correlation Coefficient

If we observe the below reference , "March 11" was the date when WHO Declares COVID-19 as a Pandemic. Covid cases have drastically raised all around the world in March 2020 (https://www.ajmc.com/view/a-timeline-of-covid19-developments-in-2020). Hence, we have decided to consider the data for March 2020 and analyse the impact of Covid on Homeless Shelter

In this inference, we are calculating the PEARSON CORRELATION COEFFICIENT for CONFIRMED CASES v/s TOTAL INVIDUALS IN HOMELESS SHELTER in the Month of March, 2020

[16]:
```
def person_correlation_coefficient(x, y):
    cov_matrix = np.cov(x,y)
    r = cov_matrix[0][1]/np.sqrt((cov_matrix[0][0]*cov_matrix[1][1]))
    print("Pearson Correlation Coefficient Value is: " + "{:5.2f}".format(r))
    return r
```

[17]:
```
us_confirmed_march = us_confirmed_transposed[((us_confirmed_transposed['date']␣
 ↪>= '2020-03-01'))][:30]
homeless_march = homeless[((homeless['Date of Census'].dt.strftime('%Y-%m-%d')␣
 ↪>= '2020-03-01'))][-30:]
```

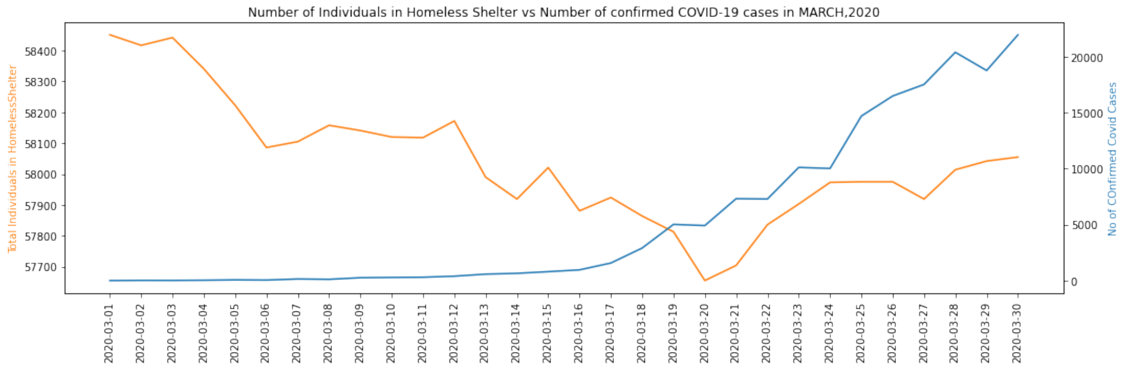
[18]:
```
homeless_march.rename(columns={'Date of Census': 'date'}, inplace=True)
homeless_march.date = homeless_march.date.astype(str)
us_confirmed_march.date = us_confirmed_march.date.astype(str)
df = pd.merge(us_confirmed_march, homeless_march, on=['date'])
```

[19]:
```
person_correlation_coefficient(np.
 ↪array(us_confirmed_march['total_confirmed_cases']), np.
 ↪array(homeless_march['Total Individuals in Shelter']))
```

```
Pearson Correlation Coefficient Value is:  0.87
```

[19]:
```
0.8667049105387643
```

[20]:
```
fig, ax1 = plt.subplots(figsize = (15,5))
ax1.plot(df['date'],df['Total Individuals in Shelter'], color = 'tab:orange')
ax1.set_xticks(df.date)
ax1.set_xticklabels(ax1.get_xticks(),rotation=90)
ax1.set_ylabel('Total Individuals in HomelessShelter',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(df['date'],df['total_confirmed_cases'],color = 'tab:blue')
ax2.set_ylabel('No of COnfirmed Covid Cases',color = 'tab:blue')
plt.title('Number of Individuals in Homeless Shelter vs Number of confirmed␣
 ↪COVID-19 cases in MARCH,2020')
fig.tight_layout()
plt.show()
```



1.2 Observation :

The below is the analysis for Number of individuals in Homeless Shelter vs Number of confirmed COVID-19 cases (in MARCH'20)

Pearson Correlation Coefficient Value is: 0.87

This value shows a strong postive correlation between Number of individuals in Homeless Shelter and Number of confirmed COVID-19 cases.This implies that Covid pandemic has an impact on people's lives causing lot of individuals to lose their job and income due to the ongoing pandemic. Due to loss of income and quality of life, many people have gone homeless, and had to move into a shelter. Hence we can infer that as the impact of Covid increases(/cases increased),total Individuals in Homeless Shelter increases.

1.3 Inference 2 : Chi-Square Test

In this inference, we are checking if the covid vaccination had an impact on the homeless people in the shelter. We have chosen 60 days before and after the vaccination drive started in the USA (2020-12-14). From the US-all datasets, we are taking the total covid deaths and total covid confirmed cases. From the X (Homeless) dataset, we are taking the total homeless men and the total homeless women in the shelter.

Our null hypothesis is that the underlying distributions both datasets are independent, i.e, the vaccination drive did not have an impact on the distributions, and the distribution of US cases and deaths is independent of distribution of the number of single men and single women in the shelter.

We are taking threshold as alpha = 0.05

[21]:
```
#homeless dataset split into 60 days before/after the covid vaccination drive of␣
↪2020-12-14

homeless_after_vaccine = homeless[((homeless['Date of Census'].dt.
↪strftime('%Y-%m-%d') >= '2020-12-14'))][-60:]
```

```
homeless_before_vaccine = homeless[(homeless['Date of Census'].dt.
↪strftime('%Y-%m-%d') <= '2020-12-14')][:60]
```

[22]:
```
#us_deaths dataset split into 60 days before/after the covid vaccination drive␣
↪of 2020-12-14

us_deaths_after_vaccine = us_deaths_transposed[((us_deaths_transposed['date']␣
↪>= '2020-12-14'))][:60]
us_deaths_before_vaccine = us_deaths_transposed[((us_deaths_transposed['date']␣
↪<= '2020-12-14'))][-60:]
```

[23]:
```
#us_confirmed dataset split into 60 days before/after the covid vaccination␣
↪drive of 2020-12-14

us_confirmed_after_vaccine =␣
↪us_confirmed_transposed[((us_confirmed_transposed['date'] >=␣
↪'2020-12-14'))][:60]
us_confirmed_before_vaccine =␣
↪us_confirmed_transposed[((us_confirmed_transposed['date'] <=␣
↪'2020-12-14'))][-60:]
```

[24]:
```
# observed values for the chi-squared test - The rows are before/after vaccine,␣
↪and columns are confirmed covid cases,
# covid deaths,homeless women in shelter, homeless men in shelter.

observed_values = np.zeros([2,4],int)
observed_values[0][0] = us_confirmed_before_vaccine['total_confirmed_cases'].
↪sum()
observed_values[1][0] = us_confirmed_after_vaccine['total_confirmed_cases'].
↪sum()
observed_values[0][1] = us_deaths_before_vaccine['total_death'].sum()
observed_values[1][1] = us_deaths_after_vaccine['total_death'].sum()
observed_values[0][2] = homeless_before_vaccine['Single Adult Men in Shelter'].
↪sum()
observed_values[1][2] = homeless_after_vaccine['Single Adult Men in Shelter'].
↪sum()
observed_values[0][3] = homeless_before_vaccine['Single Adult Women in␣
↪Shelter'].sum()
observed_values[1][3] = homeless_after_vaccine['Single Adult Women in Shelter'].
↪sum()

rows = 2
cols = 4

df = (rows-1)*(cols-1)
```

```
total_row1,total_row2= np.sum(observed_values,axis=1)
total_col1,total_col2,total_col3,total_col4  = np.sum(observed_values,axis=0)


total = total_row1+total_row2

#expected values for the chi-squared test
expected_values = np.zeros([2,4])
expected_values[0][0] = (float(total_col1)*total_row1)/(total)
expected_values[1][0] = (float(total_col1)*total_row2)/(total)
expected_values[0][1] = (float(total_col2)*total_row1)/(total)
expected_values[1][1] = (float(total_col2)*total_row2)/(total)
expected_values[0][2] = (float(total_col3)*total_row1)/(total)
expected_values[1][2] = (float(total_col3)*total_row2)/(total)
expected_values[0][3] = (float(total_col4)*total_row1)/(total)
expected_values[1][3] = (float(total_col4)*total_row2)/(total)

#calculating q_expected value
q_expected = 0.0
for i in range(rows):
    for j in range(cols):
        q_expected += ((expected_values[i][j] - observed_values[i][j])**2)/
 ↪float(expected_values[i][j])


#Displaying chi-square table as a dataframe
df_ar1 = pd.DataFrame()
df_ar1['Date'] = ['Before Vaccine','After Vaccine']
df_ar1['Observed_Covid_Cases'] = observed_values[:,0]
df_ar1['Expected_Covid_Cases'] = expected_values[:,0]
df_ar1['Observed_Covid_Deaths'] = observed_values[:,1]
df_ar1['Expected_Covid_Deaths'] = expected_values[:,1]
df_ar1['Observed_Men_In_Shelter'] = observed_values[:,2]
df_ar1['Expected_Men_In_Shelter'] = expected_values[:,2]
df_ar1['Observed_Women_In_Shelter'] = observed_values[:,3]
df_ar1['Expected_Women_In_Shelter'] = expected_values[:,3]

print("\nChi squared table")
df_ar1
```

```
Chi squared table
```

```
[24]:              Date  Observed_Covid_Cases  Expected_Covid_Cases  \
      0  Before Vaccine               8295574          8.396023e+06
      1   After Vaccine              10938538          1.083809e+07
```

```
   Observed_Covid_Deaths  Expected_Covid_Deaths  Observed_Men_In_Shelter  \
0                  80520          110338.476274                   817863
1                 172250          142431.523726                   835120

   Expected_Men_In_Shelter  Observed_Women_In_Shelter  \
0            721555.665338                     272230
1            931427.334662                     273614

   Expected_Women_In_Shelter
0              238270.345545
1              307573.654455
```

[25]:
```
print("Q_expected : ", q_expected)
print("Degrees of freedom : ",df)
```

```
Q_expected :  47835.53178643774
Degrees of freedom :  3
```

1.4 Observation :

Q_expected : 47835.53178643774 Degrees of freedom : 3

We are taking alpha = 0.05 Since Q statistic is 47835(really large), with degrees of freedom = 3, we got from the p value calculator that the p-value will be < 0.00001 (really small).

The P-Value is < .00001. The result is significant at alpha < .05.

Hence, we reject the null hypothesis, and the initial covid vaccination drive of 14th December, 2020 had an impact, i.e, distribution of X = Men/Women in Shelters is dependent on the distribution of Y = US all cases/deaths.

1.5 Inference 3 : Multi-variate Linear regression

In this inference, we are trying to predict the total deaths in the United States with the help of the data of homeless people in the United States.

[26]:
```
#function to calculate the parameters and the sse error
def parameter_search(x,y):
    w = np.zeros((np.shape(x)[1],0))
    A = np.dot(x.T,x)
    b = np.dot(x.T,y)
    beta = np.dot(np.linalg.pinv(A),b)
    #beta = np.linalg.solve(A,b)
    return beta

def sse_error(x,y_true,beta):
    y_pred = np.dot(x,beta)
    error = 0
    if len(y_true)!= len(y_pred):
        print("invalid")
        return
    for i in range(len(y_pred)):
        error+= np.power((y_true[i] - y_pred[i]),2)
    return error
```

[27]:
```
#manipulation of all US death data to get the total deaths in the US

us_all_data = us_deaths.T
new_header = us_all_data.iloc[0] #grab the first row for the header
us_all_data = us_all_data[1:] #take the data less the header row
us_all_data.columns = new_header #set the header row as the df header
us_all_data['total_death'] = us_all_data.sum(axis =1)
date = us_all_data.index
us_all_data['date'] = date
us_all_data = us_all_data.set_index('date').diff()
us_all_data = us_all_data.reset_index()
us_all_data = us_all_data[us_all_data['date']>'2020-01-01']
temp1 = us_all_data[['date','total_death']][1:]
temp1['date'] = pd.to_datetime(temp1['date'])
```

[28]:
```
#manipulation of the homeless so as to start is from 1st January 2020 so we can␣
 ↪have overlap in our data

homeless['Date of Census'] = pd.DataFrame(pd.to_datetime(homeless['Date of␣
 ↪Census']))
homeless = homeless.sort_values(by = 'Date of Census')

small_data = homeless[homeless['Date of Census']>'2020-01-01']
small_data = small_data[small_data['Date of Census']<='2021-04-03']
# temp2 = small_data[['Date of Census', 'Total Individuals in Shelter']]
temp2 = small_data
```

[29]:
```
#Merging both the homeless and the death data

temp = temp1.merge(temp2, how='left', left_on=['date'], right_on=['Date of␣
 ↪Census'])
temp = temp.dropna()
```

[30]:
```
#find parameters values for all the features

x = np.array(temp.drop(['total_death','date','Date of Census'], axis = 1))
y =np.reshape(np.array(temp['total_death']),(-1,1))
beta = parameter_search(x[:400],y[:400])
error = sse_error(x[400:],y[400:],beta)
print("The Error is:", error)
```

```python
k = temp.drop(['total_death','date','Date of Census'], axis = 1)
for i in range(len(k.columns)):
    print("column name:", k.columns[i], " parameter value:", beta[i])
```

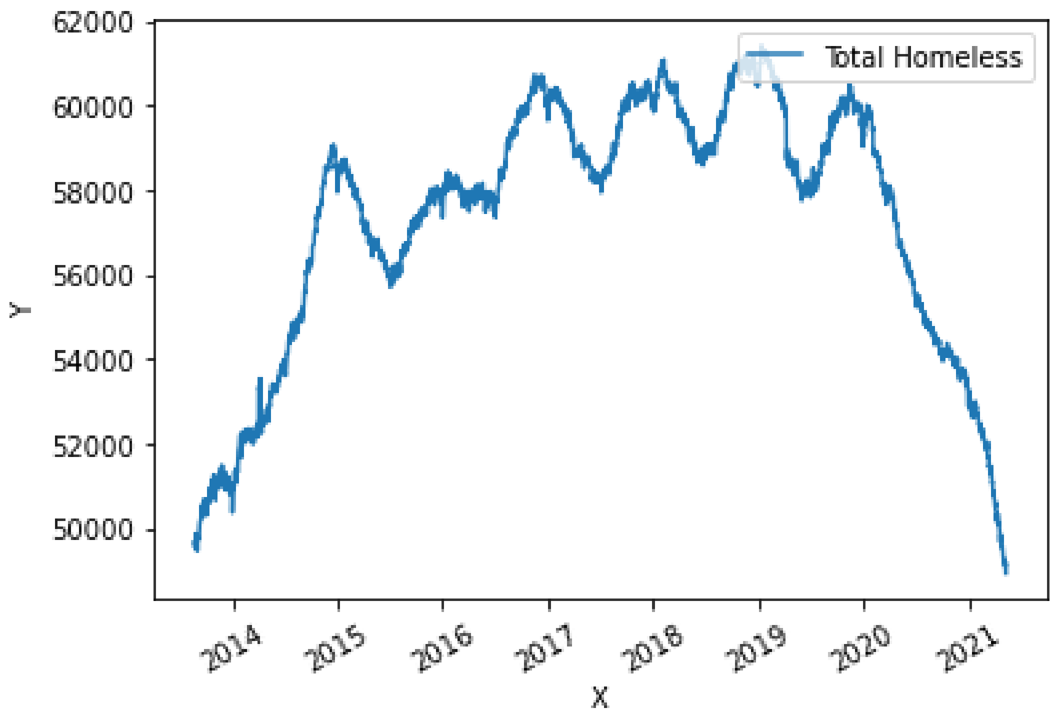
```
The Error is: [2.67610599e+08]
column name: Total Adults in Shelter  parameter value: [-9.83937027]
column name: Total Children in Shelter  parameter value: [-2.3932409]
column name: Total Individuals in Shelter  parameter value: [-12.23261672]
column name: Single Adult Men in Shelter  parameter value: [7.00603291]
column name: Single Adult Women in Shelter  parameter value: [8.72519398]
column name: Total Single Adults in Shelter  parameter value: [15.73122612]
column name: Families with Children in Shelter  parameter value: [-4.8161054]
column name: Adults in Families with Children in Shelter  parameter value:
[15.64150686]
column name: Children in Families with Children in Shelter  parameter value:
[-2.3932409]
column name: Total Individuals in Families with Children in Shelter   parameter
value: [13.24826596]
column name: Adult Families in Shelter  parameter value: [137.0168218]
column name: Individuals in Adult Families in Shelter  parameter value:
[-41.21210469]
```

1.6 Observation :

From the above we can see that the sum of squared error is too high, and hence we can't predict the total deaths based on the homeless people data.

1.7 Additional Data Analysis

```python
[31]: date = homeless['Date of Census']
individualsInShelter = homeless['Total Individuals in Shelter']
plotGraph(date,individualsInShelter)
```



1.8 Basic Interpretation

From the above plot, we can see that the individual in the homeless centers were increasing or remaining the same from year 2014 to start of 2020. But after, the start of the pandemic the people in the shelter are decreasing. We can say that an important factor of this could be people fearing to contract covid in these homeless shelters due to staying in close quarters with many people. So, we can infer that the covid has impacted the number of individual staying in the shelter home.

So, this was something interesting and therefore, we tried another inference for this.

1.9 Inference 4 : Chi-Square Test

In this inference, we are checking if the start of covid pandemic had an impact on the homeless people in the shelter. We have chosen 60 days before and after the start of covid in the USA (2020-03-01). From the US-all datasets, we are taking the total covid confirmed cases. From the X (Homeless) dataset, we are taking the total homeless individual in the shelters.

Our null hypothesis is that the underlying distributions both datasets are independent, i.e, the covid did not have an impact on the distributions, and the distribution of US cases is independent of distribution of the number of individuals in the shelter.

We are taking threshold as alpha = 0.05

```
[32]: #homeless dataset split into 60 days before/after the covid pandemic start of␣
       ↪2020-03-01

      homeless_after_covid = homeless[((homeless['Date of Census'].dt.
       ↪strftime('%Y-%m-%d') >= '2020-03-01'))][-70:-10]
      homeless_before_covid = homeless[(homeless['Date of Census'].dt.
       ↪strftime('%Y-%m-%d') <= '2020-03-01')][100:160]
```

```
[33]: #us confirmed dataset split into 60 days before/after the covid pandemic start␣
       ↪of 2020-03-01

      us_confirmed_after_covid =␣
       ↪us_confirmed_transposed[((us_confirmed_transposed['date'] >=␣
       ↪'2020-03-01'))][100:160]
      us_confirmed_before_covid =␣
       ↪us_confirmed_transposed[((us_confirmed_transposed['date'] <=␣
       ↪'2020-03-01'))][-70:-10]
```

```
[34]: #observed values for the chi-squared test
      observed_values = np.zeros([2,2],int)
      observed_values[0][0] = us_confirmed_before_covid['total_confirmed_cases'].sum()
      observed_values[1][0] = us_confirmed_after_covid['total_confirmed_cases'].sum()
      observed_values[0][1] = homeless_before_covid['Total Individuals in Shelter'].
       ↪sum()
      observed_values[1][1] = homeless_after_covid['Total Individuals in Shelter'].
       ↪sum()

      rows = 2
      cols = 2

      df = (rows-1)*(cols-1)

      total_row1,total_row2 = np.sum(observed_values,axis=1)
      total_col1,total_col2 = np.sum(observed_values,axis=0)


      total = total_row1+total_row2

      #expected values for the chi-squared test
      expected_values = np.zeros([2,4])
      expected_values[0][0] = (float(total_col1)*total_row1)/(total)
      expected_values[1][0] = (float(total_col1)*total_row2)/(total)
      expected_values[0][1] = (float(total_col2)*total_row1)/(total)
      expected_values[1][1] = (float(total_col2)*total_row2)/(total)

      q_expected = 0.0
```

```
for i in range(rows):
    for j in range(cols):
        q_expected += ((expected_values[i][j] - observed_values[i][j])**2)/
 ↪float(expected_values[i][j])


#Displaying chi-square table as a dataframe
print("\nChi squared table")

df_ar1 = pd.DataFrame()
df_ar1['Date'] = ['Before Covid','After Covid']
df_ar1['Observed_Covid_Cases'] = observed_values[:,0]
df_ar1['Expected_Covid_Cases'] = expected_values[:,0]
df_ar1['Observed_Individual_In_Shelter'] = observed_values[:,1]
df_ar1['Expected_Individual_In_Shelter'] = expected_values[:,1]

df_ar1
```

```
Chi squared table
```

[34]:
```
          Date  Observed_Covid_Cases  Expected_Covid_Cases  \
0  Before Covid                    13          1.002015e+06
1   After Covid               2954706          1.952704e+06

   Observed_Individual_In_Shelter  Expected_Individual_In_Shelter
0                         3077201                    2.075199e+06
1                         3042103                    4.044105e+06
```

[35]:
```
print("Q_expected : ", q_expected)
print("Degrees of freedom : ",df)
```

```
Q_expected :  2248227.447271685
Degrees of freedom :  1
```

1.10 Observation :

Q_expected : 2248227.447271685 Degrees of freedom : 1 We are taking alpha = 0.05 Since Q statistic is 2248227.447271685(really large), with degrees of freedom = 1, we got from the p value calculator that the p-value will be < 0.00001 (really small).

The P-Value is $< .00001$. The result is significant at alpha $< .05$.

Hence, we reject the null hypothesis, and the number of covid cases had and impact on the Total number of individual in the shelter, i.e, distribution of X = Total Individual in Shelters is dependent on the distribution of Y = US all cases. From, the result we can say that due to covid the people in the shelter home have started leaving the shelter home. This may not be the only factor but this is an important factor for people leaving the shelter homes.