CSE-544 Project

Irfan Ahmed (113166464)

Anurag Yepuri (113070893)

Sharfuddin Mohammed (112070993)

Fatemeh Aslan Beigi (113870646)

Note: Reviewing in Ipynb files would be better than in this PDF due to better formatting

Outlier Analysis:

```
import pandas as pd
import numpy as np
from scipy.stats import gamma
import matplotlib.pyplot as  plt
```

```
from google.colab import drive
drive.mount('/content/gdrive')

%cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call d
/content/gdrive/My Drive/Prob_stats_proj
```

Data Preprocessing

```
data = pd.read_csv('7.csv')
```

```
## converting date column to datetime data type ##
data['Date'] = pd.to_datetime(data['Date'])
```

Data snapshot

Following df view gives a snapshot of the columns in our data and the values in each of them

```
data
```

	Date	IA confirmed cumulative	ID confirmed cumulative	IA deaths cumulative	ID deaths cumulative	IA confirmed	ID confirmed
0	2020-01-22	0	0	0	0	0	0
1	2020-01-23	0	0	0	0	0	0
3	2020-01-25	0	0	0	0	0	0

Data features

Following list displays the list of features in our dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438 entries, 0 to 437
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date                     438 non-null    datetime64[ns]
 1   IA confirmed cumulative  438 non-null    int64
 2   ID confirmed cumulative  438 non-null    int64
 3   IA deaths cumulative     438 non-null    int64
 4   ID deaths cumulative     438 non-null    int64
 5   IA confirmed             438 non-null    int64
 6   ID confirmed             438 non-null    int64
 7   IA deaths                438 non-null    int64
 8   ID deaths                438 non-null    int64
dtypes: datetime64[ns](1), int64(8)
memory usage: 30.9 KB
```

Detecting outliers with Tukey's rule

Using Tukey's rule, we detected outliers and found the number of outliers for each feature

Negative value found at 2020-12-25 for ID state.

Outliers -

- IA confirmed - 38
- IA deaths - 35
- ID confirmed - 28
- ID deaths - 37

But when removing the outliers, most of the original data was getting deleted. So instead we used the original data to perform the asked inferences.

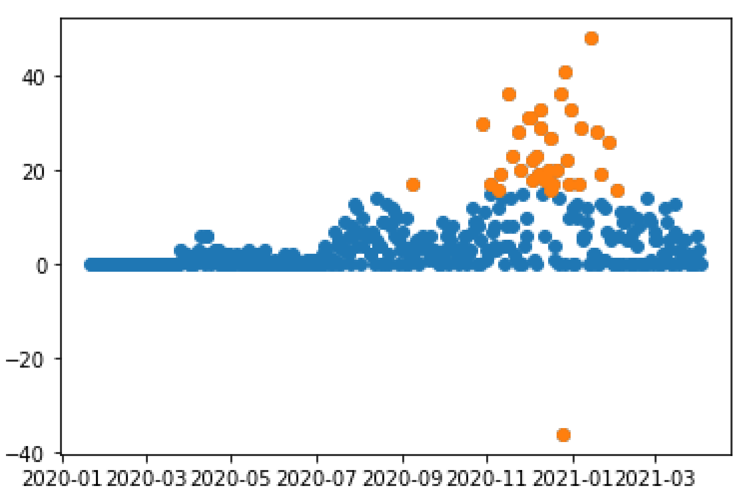
```
def IQR(X):
    X = sorted(X)
```

```
    n = len(X)
    Q1 = X[np.int(np.ceil(n/4))]
    Q3 = X[np.int(np.ceil(3*n/4))]

    return Q3-Q1, Q1, Q3

def outliers(X, handle, a, b):
    if( a < 0): a=0

    condition = (X[handle] < a) | (X[handle] > b)
    outlier = X.loc[condition]

    return outlier
```

Data plot

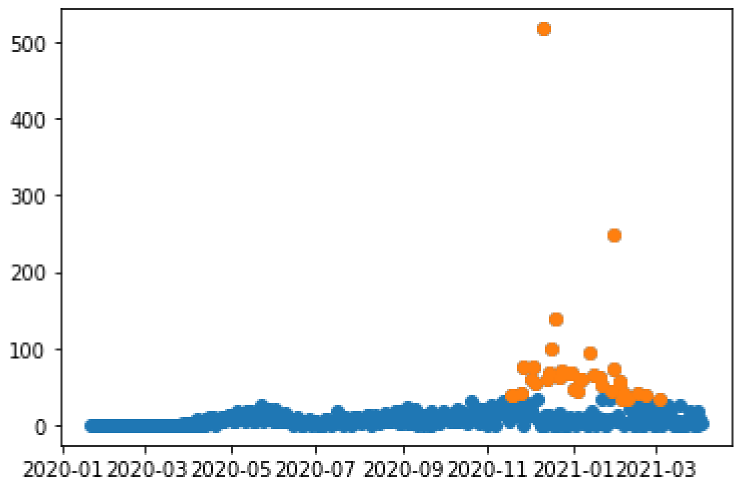
The following scatter plot represent the data for each of the features and identifies the outliers in yellow

```
handle = 'ID deaths'

iqr, q1,q3 = IQR(data[handle])
threshold_min = q1 - 1.5 * iqr
threshold_max = q3 + 1.5 * iqr


outlier_data = outliers(data, handle, threshold_min, threshold_max)

plt.scatter(data['Date'], data[handle])
plt.scatter(outlier_data['Date'], outlier_data[handle])
plt.show()
print(len(outlier_data))
```



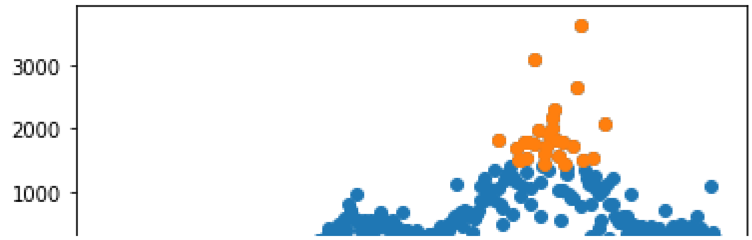

```
37
```

```
handle = 'IA deaths'

iqr, q1,q3 = IQR(data[handle])
threshold_min = q1 - 1.5 * iqr
threshold_max = q3 + 1.5 * iqr


outlier_data = outliers(data, handle, threshold_min, threshold_max)

plt.scatter(data['Date'], data[handle])
plt.scatter(outlier_data['Date'], outlier_data[handle])
plt.show()
print(len(outlier_data))
```

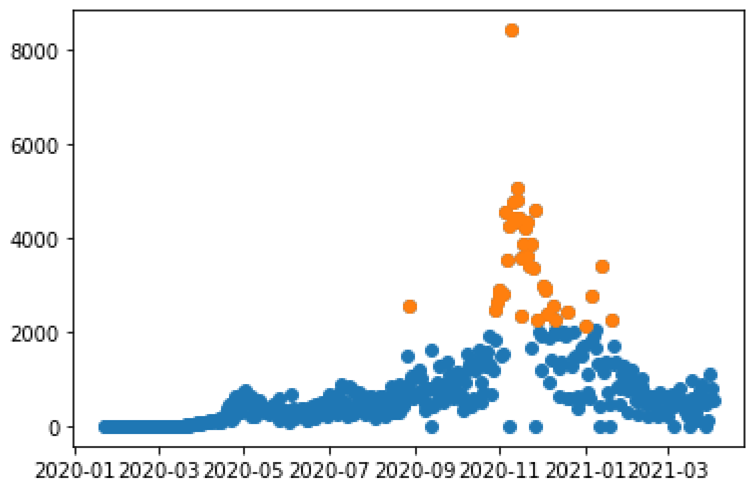



```
35
```

```
handle = 'ID confirmed'

iqr, q1,q3 = IQR(data[handle])
threshold_min = q1 - 1.5 * iqr
threshold_max = q3 + 1.5 * iqr


outlier_data = outliers(data, handle, threshold_min, threshold_max)

plt.scatter(data['Date'], data[handle])
plt.scatter(outlier_data['Date'], outlier_data[handle])
plt.show()
print(len(outlier_data))
```

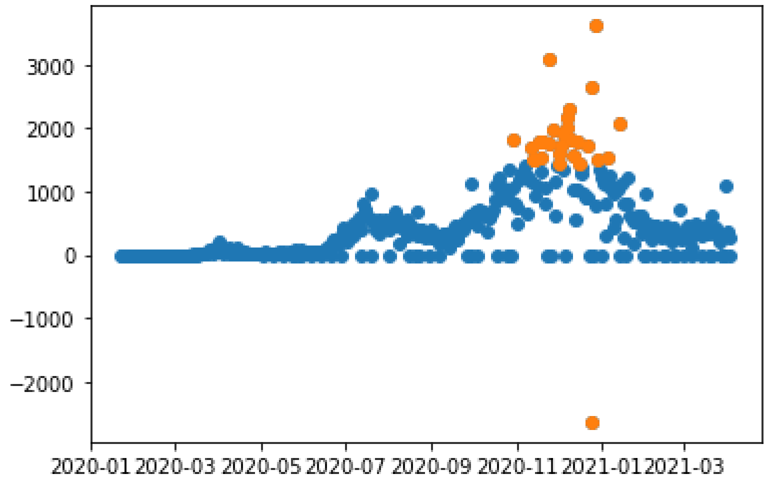



```
handle = 'IA confirmed'

iqr, q1,q3 = IQR(data[handle])
threshold_min = q1 - 1.5 * iqr
threshold_max = q3 + 1.5 * iqr


outlier_data = outliers(data, handle, threshold_min, threshold_max)

plt.scatter(data['Date'], data[handle])
plt.scatter(outlier_data['Date'], outlier_data[handle])
plt.show()
print(len(outlier_data))
```




```
38
```

```
handle = 'ID confirmed'

iqr, q1,q3 = IQR(data[handle])
threshold_min = q1 - 1.5 * iqr
threshold_max = q3 + 1.5 * iqr


outlier_data = outliers(data, handle, threshold_min, threshold_max)

plt.scatter(data['Date'], data[handle])
plt.scatter(outlier_data['Date'], outlier_data[handle])
plt.show()
print(len(outlier_data))
```


3000
2000
1000
0
−1000
−2000
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03

28

Question A)

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.

```
import pandas as pd
import numpy as np
```

```
%cd D:\StonyBrook\Study\Prob&Stats CSE544\Project

# from google.colab import drive
# drive.mount('/content/gdrive')

# %cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Mounted at /content/gdrive
/content/gdrive/My Drive/Prob_stats_proj
```

Data Preprocessing

```
data = pd.read_csv('7.csv')
```

```
## converting date column to datetime data type ##
data['Date'] = pd.to_datetime(data['Date'])
```

```
data
```

	Date	IA confirmed cumulative	ID confirmed cumulative	IA deaths cumulative	ID deaths cumulative	IA confirmed	ID confirmed
0	2020-01-22	0	0	0	0	0	0
1	2020-01-23	0	0	0	0	0	0
2	2020-01-24	0	0	0	0	0	0

Data features

Following list displays the features in our data

01-26

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438 entries, 0 to 437
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date                     438 non-null    datetime64[ns]
 1   IA confirmed cumulative  438 non-null    int64
 2   ID confirmed cumulative  438 non-null    int64
 3   IA deaths cumulative     438 non-null    int64
 4   ID deaths cumulative     438 non-null    int64
 5   IA confirmed             438 non-null    int64
 6   ID confirmed             438 non-null    int64
 7   IA deaths                438 non-null    int64
 8   ID deaths                438 non-null    int64
dtypes: datetime64[ns](1), int64(8)
memory usage: 30.9 KB
```

```
def generate_data(p,data):
    X, Y = [], []
    for i in range(len(data) - p):
        X.append([1] + data[i:i+p].tolist())  # appending 1 for beta_0

    Y = data[p:].tolist()

    return np.array(X),np.array(Y)


def SSE(y_pred, y_true):
    A = y_true - y_pred
    A = A*A
    return sum(A)

def MSE(y_pred,y_true):
    return SSE(y_pred,y_true)/len(y_pred)
```

```
def MAPE(y_pred, y_true):
    A = np.abs(y_true - y_pred)
    A = np.sum(np.divide(A, y_true))*100
    return A/len(y_pred)


def AutoRegression(p, data):
    X, Y = generate_data(p, data)
    A = np.linalg.inv(np.matmul(np.transpose(X), X))
    B = np.matmul(np.transpose(X), Y)
    beta = np.matmul(A,B)

    return beta

def Predict_AR(data, beta):
    p = len(beta) - 1
    data = data.to_numpy()
    for i in range(7):
        y_pred = beta[0] + np.matmul(data[-p:].T, beta[1:])
        data = np.append(data,y_pred)

    return data


###### Outlier Detection Using Tukey's Rule ########
## Plot data ####


###### Getting August 2020 data ######
start_date, end_date = '2020-08-01', '2020-08-29'
condition = (data['Date'] >= start_date) & (data['Date'] <= end_date)
august_data = data.loc[condition]
august_first_3_week_data = august_data[:-7]
print(august_first_3_week_data)
```

```
           Date  IA confirmed cumulative  ...  IA deaths  ID deaths
192 2020-08-01                     44936  ...          7          7
193 2020-08-02                     45481  ...          2          0
194 2020-08-03                     45802  ...          5          4
195 2020-08-04                     45981  ...          6         10
196 2020-08-05                     46501  ...          8          7
197 2020-08-06                     47137  ...         13          6
198 2020-08-07                     47728  ...          6          6
199 2020-08-08                     48112  ...         13          6
200 2020-08-09                     48732  ...          5          2
201 2020-08-10                     49000  ...          1          2
202 2020-08-11                     49208  ...          6          7
203 2020-08-12                     49702  ...         12          0
204 2020-08-13                     50167  ...          5          5
205 2020-08-14                     50808  ...         10         14
206 2020-08-15                     51640  ...          9          4
207 2020-08-16                     52306  ...          2          0
```

```
208 2020-08-17                      52617  ...         4          4
209 2020-08-18                      52930  ...         8          9
210 2020-08-19                      53538  ...        16          9
211 2020-08-20                      53831  ...         9          0
212 2020-08-21                      54709  ...         5         13
213 2020-08-22                      55496  ...        13          2

[22 rows x 9 columns]
```

AutoRegression

Autoregression(3)

The following displays the MSE and MAPE prediction scores for each of the following features using AutoRegression(3)

```
##### AutoRegression(3) for IA confirmed cases #####
handle = 'IA confirmed'

beta_coefficients = AutoRegression(3,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [ 5.66635313e+02 -1.34695294e-01 -2.23357517e-01  2.3140849
[513.18017973 391.34473344 436.56796269 511.12810621 534.6916701
 517.39955435 498.09201224] 214     661
215     428
216     571
217     863
218    1477
219    2535
220    1081
Name: IA confirmed, dtype: int64

MSE :  780502.625601559

MAPE :  41.793620039545836
```

```
##### AutoRegression(3) for IA deaths cases #####
handle = 'IA deaths'

beta_coefficients = AutoRegression(3,august_first_3_week_data[handle])
```

```
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [10.36353614 -0.03504037 -0.26036512 -0.03231118]
[8.32630186 6.53455507 7.52899358 8.12713856 7.91167815 7.72805835
 7.76913048] 214     5
215     5
216     9
217    13
218    18
219    11
220    17
Name: IA deaths, dtype: int64

MSE :  33.85951053875955

MAPE :  41.590798986318184
```

```
##### AutoRegression(3) for ID confirmed cases #####
handle = 'ID confirmed'

beta_coefficients = AutoRegression(3,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [ 4.80033946e+02 -1.19496137e-01  2.29419760e-02 -4.3096158
[482.77789562 385.88766059 439.4671811  412.25753072 426.23739101
 418.60812386 422.5090897 ] 214      0
215    403
216    409
217    309
218    342
219    262
220    293
Name: ID confirmed, dtype: int64

MSE :  41907.47746495576

MAPE :  inf
```

```
##### AutoRegression(3) for ID deaths cases #####
handle = 'ID deaths'
```

```
handle = 'ID deaths'

beta_coefficients = AutoRegression(3,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [10.37464008 -0.09420523 -0.44076285 -0.36700701]
[3.91070902 6.83318872 5.95470621 4.80899618 5.34136936 5.73372874
 5.46301166] 214     0
215     8
216    12
217    11
218     6
219    10
220     5
Name: ID deaths, dtype: int64

MSE :  15.768349069585089

MAPE :  inf
```

Autoregression(5)

The following displays the MSE and MAPE prediction scores for each of the features using AutoRegression(5)

```
##### AutoRegression(5) for IA confirmed cases #####
handle = 'IA confirmed'

beta_coefficients = AutoRegression(5,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [ 1.23688915e+03 -3.78362287e-01 -5.05795367e-01 -1.4116118
 -3.52351814e-01 -1.07904834e-01]
[375.29195354 416.91063269 393.62535071 264.27616256 523.2342253
 678.87906026 585.12962034] 214     661
215     428
216     571
217     863
218    1477
```

```
219    2535
220    1081
Name: IA confirmed, dtype: int64

MSE :  724632.2272832438

MAPE :  47.13160284598954
```

```
##### AutoRegression(5) for ID confirmed cases #####
handle = 'ID confirmed'

beta_coefficients = AutoRegression(5,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [ 5.01137897e+02 -2.48285335e-01  1.26179862e-01 -1.5863076
  1.12680461e-01 -3.82969515e-02]
[516.75777886 316.2148367  585.31802984 302.99907648 497.78328109
 334.9635085  491.67913059] 214      0
215    403
216    409
217    309
218    342
219    262
220    293
Name: ID confirmed, dtype: int64

MSE :  53537.11806932668

MAPE :  inf
```

```
##### AutoRegression(5) for IA deaths cases #####
handle = 'IA deaths'

beta_coefficients = AutoRegression(5,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [24.00162144 -0.49331075 -0.48065216 -0.27876814 -0.5619020
[ 2.23814104  2.25642108 11.44240974  9.16272835  6.0653865  11.23131005
  7.27228123] 214     5
215     5
216     9
```

```
217    13
218    18
219    11
220    17
Name: IA deaths, dtype: int64

MSE :  38.994589362148744

MAPE :  41.770317133024434
```

```
##### AutoRegression(5) for ID deaths cases #####
handle = 'ID deaths'

beta_coefficients = AutoRegression(5,august_first_3_week_data[handle])
print('Beta Coefficients : ' , beta_coefficients)
predicted_values = Predict_AR(august_first_3_week_data[handle],beta_coefficients)[-7:]
print(predicted_values, august_data[handle][-7:])


print('\nMSE : ' , MSE(predicted_values, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(predicted_values, august_data[handle][-7:] ))
```

```
Beta Coefficients :  [13.64619059 -0.29595428 -0.17363244 -0.23096157 -0.4929253
[2.23751629 6.12792431 7.45149344 3.02904424 6.40463944 6.22611731
 4.27153991] 214     0
215     8
216    12
217    11
218     6
219    10
220     5
Name: ID deaths, dtype: int64

MSE :  15.38182456175087

MAPE :  inf
```

EWMA

```
def EWMA(alpha, Y):
    y_hat = 0.0
    Y = Y.to_numpy()
    for i in range(len(Y)-1):
        y_hat = (1-alpha)*(y_hat + Y[i])

    y_hat+= Y[-1]
    y_hat = y_hat*alpha

    return y_hat
```

```
def Predict_EWMA(alpha, Y, y_hat):
    Y = Y.to_numpy()
    pred_values = [y_hat]
    for i in range(1,7):
        y_hat = alpha*Y[i] + (1-alpha)*y_hat
        pred_values.append(y_hat)

    return pred_values
```

EWMA (alpha =0.5)

The following displays the MSE and MAPE prediction scores for each of the following features using EWMA, with alpha=0.5

```
##### EWMA(0.5) for IA confirmed cases #####
handle = 'IA confirmed'
alpha = 0.5

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[712.815943479538, 570.407971739769, 570.7039858698845, 716.8519929349422, 1096.
215     428
216     571
217     863
218    1477
219    2535
220    1081
Name: IA confirmed, dtype: int64

MSE :  120119.62289580102

MAPE :  20.884348345351857
```

```
##### EWMA(0.5) for IA deaths cases #####
handle = 'IA deaths'
alpha = 0.5

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[10.27108359336853, 7.635541796684265, 8.317770898342133, 10.658885449171066, 14
```

```
215     5
216     9
217    13
218    18
219    11
220    17
Name: IA deaths, dtype: int64

MSE :  8.802800691487516

MAPE :  33.14257708187212
```

```
##### EWMA(0.5) for ID confirmed cases #####
handle = 'ID confirmed'
alpha = 0.5

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[359.60629892349243, 381.3031494617462, 395.1515747308731, 352.07578736543655, 3
215    403
216    409
217    309
218    342
219    262
220    293
Name: ID confirmed, dtype: int64

MSE :  19100.165519379807

MAPE :  inf
```

```
##### EWMA(0.5) for ID deaths cases #####
handle = 'ID deaths'
alpha = 0.5

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[5.206548452377319, 6.60327422618866, 9.30163711309433, 10.150818556547165, 8.07
215     8
216    12
217    11
218     6
219    10
```

```
220     5
Name: ID deaths, dtype: int64

MSE :  6.624337398517957

MAPE :  inf
```

EWMA (alpha =0.8)

The following displays the MSE and MAPE prediction scores for each of the following features using EWMA, with alpha=0.8

```
##### EWMA(0.8) for IA confirmed cases #####
handle = 'IA confirmed'
alpha = 0.8

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[783.871623342618, 499.1743246685236, 556.6348649337046, 801.726972986741, 1341.
215     428
216     571
217     863
218    1477
219    2535
220    1081
Name: IA confirmed, dtype: int64

MSE :  22626.52912647783

MAPE :  12.268134517127049
```

```
##### EWMA(0.8) for IA deaths cases #####
handle = 'IA deaths'
alpha = 0.8

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[11.601882177836865, 6.320376435567372, 8.464075287113474, 12.092815057422694, 1
215     5
216     9
```

```
217    13
218    18
219    11
220    17
Name: IA deaths, dtype: int64

MSE :  7.160578492838215

MAPE :  27.74439017035049
```

```
##### EWMA(0.8) for ID confirmed cases #####
handle = 'ID confirmed'
alpha = 0.8

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[344.5830454747838, 391.3166090949568, 405.4633218189914, 328.2926643637983, 339
215    403
216    409
217    309
218    342
219    262
220    293
Name: ID confirmed, dtype: int64

MSE :  17073.518956820095

MAPE :  inf
```

```
##### EWMA(0.8) for ID deaths cases #####
handle = 'ID deaths'
alpha = 0.8

y_hat_t = EWMA(alpha,august_first_3_week_data[handle])
y_pred = Predict_EWMA(alpha,august_data[handle][-7:], y_hat_t)
print(y_pred, august_data[handle][-7:] )

print('\nMSE : ' , MSE(y_pred, august_data[handle][-7:] ))
print('\nMAPE : ', MAPE(y_pred, august_data[handle][-7:] ))
```

```
[3.7502158035371127, 7.150043160707423, 11.030008632141486, 11.006001726428298,
215     8
216    12
217    11
218     6
219    10
220     5
Name: ID deaths, dtype: int64
```

```
MSE :  2.5520090127111494

MAPE :  inf
```

Question B)

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 $\alpha=0.05$ for all. For t-test, the threshold to check against is $t_{n-1,\alpha/2}$ 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.

```
import pandas as pd
import numpy as np


#%cd D:\StonyBrook\Study\Prob&Stats CSE544\Project

from google.colab import drive
drive.mount('/content/gdrive')

%cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Mounted at /content/gdrive
/content/gdrive/My Drive/Prob_stats_proj
```

Data processing

```
data = pd.read_csv('7.csv')


## converting date column to datetime data type ##
data['Date'] = pd.to_datetime(data['Date'])
```

```
data['Date'] = pd.to_datetime(data['Date'])
```

```
data
```

	Date	IA confirmed cumulative	ID confirmed cumulative	IA deaths cumulative	ID deaths cumulative	IA confirmed	ID confirmed
0	2020-01-22	0	0	0	0	0	0
1	2020-01-23	0	0	0	0	0	0
2	2020-01-24	0	0	0	0	0	0
3	2020-01-25	0	0	0	0	0	0
4	2020-01-26	0	0	0	0	0	0
...	...	...	...	...	...	...	...
433	2021-03-30	349742	179429	5726	1956	141	0
434	2021-	350840	180536	5744	1962	1098	1107

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438 entries, 0 to 437
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date                     438 non-null    datetime64[ns]
 1   IA confirmed cumulative  438 non-null    int64
 2   ID confirmed cumulative  438 non-null    int64
 3   IA deaths cumulative     438 non-null    int64
 4   ID deaths cumulative     438 non-null    int64
 5   IA confirmed             438 non-null    int64
 6   ID confirmed             438 non-null    int64
 7   IA deaths                438 non-null    int64
 8   ID deaths                438 non-null    int64
dtypes: datetime64[ns](1), int64(8)
memory usage: 30.9 KB
```

```
###### Feb and March data #######
feb_start_date , feb_end_date = '2021-02-01', '2021-02-28'
march_start_date, march_end_date = '2021-03-01', '2021-03-31'
condition = (data['Date'] >= feb_start_date) & (data['Date'] <= feb_end_date)
feb_data = data.loc[condition]

condition = (data['Date'] >= march_start_date) & (data['Date'] <= march_end_date)
```

```
march_data = data.loc[condition]

print(len(march_data))
```

    31

Hypothesis tests for IA confirmed cases

The below hypothesis tests are on IA confirmed cases

```
########################### Tests for IA confirmed cases handle #######################
```

1. Wald's Test for IA_confirmed

The null hypothesis for the Wald's test below is:

average_IA_confirmed_in_feb = average_IA_confirmed_in_march

The wald's statistic obtained is less than the critical value, hence we accept the above hypothesis

Is Wald's test applicable? Since the number of samples is high, using CLT the mean is Asymptotically normal. Hence wald's test is applicable

```
handle = 'IA confirmed'
mean_feb = np.mean(feb_data[handle])

## Using mean_feb as guess for mean_march ##

# Wald's test #
# w = (theta_hat - theta_0)/ se_hat(theta_hat)
# theta_hat is estimator of theta

# Null Hypothesis : mean_march = mean_feb
# Alternate Hypothesis : mean_march != mean_feb

# Assuming the distribution of march data to be poisson. MLE_mean = Sample_mean

sample_mean_march = np.mean(march_data[handle])

standard_error_estimate = np.sqrt(sample_mean_march/len(march_data))

walds_statistic = np.abs((sample_mean_march - mean_feb)/standard_error_estimate)

print('MLE for March data ',sample_mean_march )
print('Guess mean ', mean_feb)
print('Standard Error ', standard_error_estimate)
print('Walds Statistic', walds_statistic )
```

```
# Accept Null Hypothesis(less than 1.96)
```

```
MLE for March data  468.51612903225805
Guess mean  611.1785714285714
Standard Error  3.887598682627827
Walds Statistic 36.69680284485552
```

2. Z-Test for IA_confirmed

The null hypothesis for the z-test is same as the wald's test above

We accept the null hypothesis in this case as it is less than the critical value.

Is Z-test applicable? Yes, since we estimate true variance using the entire data, and number of samples is large

```
## Z-test ##

# z_statistic = (sample_mean - guess)/ root(true_variance/n)
# true variance = corrected sample standard deviation

sample_mean_full_data = np.mean(data[handle])

true_variance = np.sum(np.square(data[handle] - sample_mean_full_data))/(len(data)-1)

z_statistic = np.abs((sample_mean_march - mean_feb)/(np.sqrt(true_variance)/np.sqrt(le

print(' True Variance ', true_variance)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' z_statistic ', z_statistic)


# Accept Null Hypothesis(less than 1.96)
```

```
True Variance  1040931.0330971861
Sample Mean  468.51612903225805
Guess  611.1785714285714
z_statistic  0.7785374776325806
```

3. T- Test for IA_confirmed

The null hypothesis for the T-test is same as the wald's test above

We accept the null hypothesis under the T-test as the T-statistic is less than the critical value.

Is T-test applicable? Since the number of samples is high, using CLT, the mean is Asymptotically normal. Hence T-test is applicable

```
## t-test ##

# t_statistic = (sample_mean - guess)/ corrected_sample_standard_deviation/root(n)
corrected_sample_SD = np.sqrt(np.sum(np.square(march_data[handle] - sample_mean_march)

t_statistic = np.abs((sample_mean_march - mean_feb)/(corrected_sample_SD/np.sqrt(len(m

print(' Corrected Sample Standard Deviation', corrected_sample_SD)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' t_statistic ', t_statistic)

# degrees of freedom : 28 + 31 -2 = 57
# threshold = 2.002465

# Accept Null Hypothesis
```

```
Corrected Sample Standard Deviation 273.86065446594574
Sample Mean  468.51612903225805
Guess  611.1785714285714
t_statistic  2.900419792822982
```

4. Wald's 2 sample test

The null hypothesis for the this is same as the wald's test above

We accept the null hypothesis under Wald's 2 test as the statistic is less than the critical value.

Is Wald's 2 sample test applicable? Since the number of samples is high, using CLT the mean is Asymptotically normal. Hence wald's 2 sample test is applicable

```
## Walds - 2 sample test ##

# delta = mean_march - mean_feb

# w_stat = delta_hat/ SE_hat(delta_hat)

# Assumption : Data is poisson distributed

# Null Hypothesis : delta = 0
# Alternate Hypothesis : delta != 0


sample_mean_march = np.mean(march_data[handle])
sample_mean_feb = np.mean(feb_data[handle])

delta_hat = np.abs(sample_mean_march - sample_mean_feb)

SE_hat = np.sqrt(sample_mean_march/len(march_data) + sample_mean_feb/len(feb_data))
```

```
w_stat = np.abs(delta_hat/SE_hat)

print(' Standard Error Estimate', SE_hat)
print(' Delta Estimate ',delta_hat )
print(' w_stat ', w_stat)

# Accept Null Hypothesis
```

```
Standard Error Estimate 6.0779297165744355
Delta Estimate  142.6624423963134
w_stat  23.472209954530204
```

5. Unpaired 2-sample T-test

We accept the null hypothesis under this test as is less than the critical value

Is 2 sample T-test applicable? Since the number of samples is high, using CLT, the mean is Asymptotically normal. Hence 2-sample T-test is applicable

```
## 2 sample unpaired t-test ##

# D_bar = X_bar - Y_bar
# t_stat = D_bar/root(corrected_var_x/n + corrected_var_y/m)

corrected_variance_march = np.sum(np.square(march_data[handle] - sample_mean_march))/(
corrected_variance_feb =  np.sum(np.square(feb_data[handle] - sample_mean_feb))/(len(f

D_bar = np.abs(sample_mean_march - sample_mean_feb)

t_stat = D_bar/np.sqrt(corrected_variance_march/len(march_data) + corrected_variance_f

print(' Corrected Variance March', corrected_variance_march)
print(' Corrected Variance Feb', corrected_variance_feb)
print(' D bar',D_bar )
print(' t_statistic ', t_stat)

# threshold = 2.002465

# Accept Null Hypothesis
```

```
Corrected Variance March 74999.65806451613
Corrected Variance Feb 58015.92989417989
D bar 142.6624423963134
t_statistic  2.128735120645114
```

Hypothesis tests for IA deaths

The below results show the value of the statistic calculated for each of the following hypotheis tests:

- Wald's test
- Z-test
- T-test
- Wald's 2 sample test
- Unpaired 2-sample T-test

The Null hypothesis is rejected by all the above tests as the statistic calculated for each test is greater than the critical value for each of the above tests.

These tests are applicable as the number of samples can be assumed to be large. Thus, the mean is asymptotically normal using CLT. We estimate the true variance for the Z-test using the entire

```
########################### Tests for IA death cases handle ##########################


handle = 'IA deaths'
mean_feb = np.mean(feb_data[handle])

## Using mean_feb as guess for mean_march ##

# Wald's test #
# w = (theta_hat - theta_0)/ se_hat(theta_hat)
# theta_hat is estimator of theta

# Null Hypothesis : mean_march = mean_feb
# Alternate Hypothesis : mean_march != mean_feb

# Assuming the distribution of march data to be poisson. MLE_mean = Sample_mean

sample_mean_march = np.mean(march_data[handle])

standard_error_estimate = np.sqrt(sample_mean_march/len(march_data))

walds_statistic = np.abs((sample_mean_march - mean_feb)/standard_error_estimate)

print('Walds test')
print('MLE for March data ',sample_mean_march )
print('Guess mean ', mean_feb)
print('Standard Error ', standard_error_estimate)
print('Walds Statistic', walds_statistic )

# Reject Null Hypothesis(greater than 1.96)
print('########################################')


## Z-test ##
```

```
# z_statistic = (sample_mean - guess)/ root(true_variance/n)
# true variance = corrected sample standard deviation

sample_mean_full_data = np.mean(data[handle])

true_variance = np.sum(np.square(data[handle] - sample_mean_full_data))/(len(data)-1)

z_statistic = np.abs((sample_mean_march - mean_feb)/(np.sqrt(true_variance)/np.sqrt(le

print('Z test')
print(' True Variance ', true_variance)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' z_statistic ', z_statistic)


# Reject Null Hypothesis(greater than 1.96)

print('########################################')


## t-test ##

# t_statistic = (sample_mean - guess)/ corrected_sample_standard_deviation/root(n)
corrected_sample_SD = np.sqrt(np.sum(np.square(march_data[handle] - sample_mean_march)

t_statistic = np.abs((sample_mean_march - mean_feb)/(corrected_sample_SD/np.sqrt(len(m

print('T-test')
print(' Corrected Sample Standard Deviation', corrected_sample_SD)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' t_statistic ', t_statistic)

# degrees of freedom : 28 + 31 -2 = 57
# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')

## Walds - 2 sample test ##

# delta = mean_march - mean_feb

# w_stat = delta_hat/ SE_hat(delta_hat)

# Assumption : Data is poisson distributed

# Null Hypothesis : delta = 0
# Alternate Hypothesis : delta != 0
```

```
sample_mean_march = np.mean(march_data[handle])
sample_mean_feb = np.mean(feb_data[handle])

delta_hat = np.abs(sample_mean_march - sample_mean_feb)

SE_hat = np.sqrt(sample_mean_march/len(march_data) + sample_mean_feb/len(feb_data))

w_stat = np.abs(delta_hat/SE_hat)

print('Walds - 2 sample test')
print(' Standard Error Estimate', SE_hat)
print(' Delta Estimate ',delta_hat )
print(' w_stat ', w_stat)

# Reject Null Hypothesis

print('########################################')

## 2 sample unpaired t-test ##

# D_bar = X_bar - Y_bar
# t_stat = D_bar/root(corrected_var_x/n + corrected_var_y/m)

corrected_variance_march = np.sum(np.square(march_data[handle] - sample_mean_march))/(
corrected_variance_feb =  np.sum(np.square(feb_data[handle] - sample_mean_feb))/(len(f

D_bar = np.abs(sample_mean_march - sample_mean_feb)

t_stat = D_bar/np.sqrt(corrected_variance_march/len(march_data) + corrected_variance_f

print('2 sample unpaired t-test ')
print(' Corrected Variance March', corrected_variance_march)
print(' Corrected Variance Feb', corrected_variance_feb)
print(' D bar',D_bar )
print(' t_statistic ', t_stat)

# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')
```

```
Walds test
MLE for March data  8.774193548387096
Guess mean  20.392857142857142
Standard Error  0.5320136291119562
Walds Statistic 21.83903373653052
########################################
Z test
 True Variance  995.7540150256522
 Sample Mean  8.774193548387096
```

```
 Guess  20.392857142857142
 z_statistic  2.050033656586363
########################################
T-test
 Corrected Sample Standard Deviation 9.67370896612516
 Sample Mean  8.774193548387096
 Guess  20.392857142857142
 t_statistic  6.687195297225558
########################################
Walds - 2 sample test
 Standard Error Estimate 1.0056613883865118
 Delta Estimate  11.618663594470046
 w_stat  11.553256124420853
########################################
2 sample unpaired t-test
 Corrected Variance March 93.5806451612903
 Corrected Variance Feb 293.3584656084656
 D bar 11.618663594470046
 t_statistic  3.1626895393292975
########################################
```

Hypothesis tests for ID confirmed

The below results show the value of the statistic calculated for each of the following hypotheis tests:

- Wald's test
- Z-test
- T-test
- Wald's 2 sample test
- Unpaired 2-sample T-test

The Null hypothesis is rejected by all the above tests as the statistic calculated for each test is greater than the critical value for each of the above tests.

These tests are applicable as the number of samples can be assumed to be large. Thus, the mean is asymptotically normal using CLT. We estimate the true variance for the Z-test using the entire data.

```
########################### Tests for ID confirmed cases handle #####################

handle = 'ID confirmed'
mean_feb = np.mean(feb_data[handle])

## Using mean_feb as guess for mean_march ##

# Wald's test #
# w = (theta_hat - theta_0)/ se_hat(theta_hat)
# theta_hat is estimator of theta
```

```
# theta_hat is estimator of theta

# Null Hypothesis : mean_march = mean_feb
# Alternate Hypothesis : mean_march != mean_feb

# Assuming the distribution of march data to be poisson. MLE_mean = Sample_mean

sample_mean_march = np.mean(march_data[handle])

standard_error_estimate = np.sqrt(sample_mean_march/len(march_data))

walds_statistic = np.abs((sample_mean_march - mean_feb)/standard_error_estimate)

print('Walds test')
print('MLE for March data ',sample_mean_march )
print('Guess mean ', mean_feb)
print('Standard Error ', standard_error_estimate)
print('Walds Statistic', walds_statistic )

# Reject Null Hypothesis(greater than 1.96)
print('########################################')


## Z-test ##

# z_statistic = (sample_mean - guess)/ root(true_variance/n)
# true variance = corrected sample standard deviation

sample_mean_full_data = np.mean(data[handle])

true_variance = np.sum(np.square(data[handle] - sample_mean_full_data))/(len(data)-1)

z_statistic = np.abs((sample_mean_march - mean_feb)/(np.sqrt(true_variance)/np.sqrt(le

print('Z test')
print(' True Variance ', true_variance)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' z_statistic ', z_statistic)


# Reject Null Hypothesis(greater than 1.96)

print('########################################')


## t-test ##

# t_statistic = (sample_mean - guess)/ corrected_sample_standard_deviation/root(n)
corrected_sample_SD = np.sqrt(np.sum(np.square(march_data[handle] - sample_mean_march)

t_statistic = np.abs((sample_mean_march - mean_feb)/(corrected_sample_SD/np.sqrt(len(m
```

```
print('T-test')
print(' Corrected Sample Standard Deviation', corrected_sample_SD)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' t_statistic ', t_statistic)

# degrees of freedom : 28 + 31 -2 = 57
# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')

## Walds - 2 sample test ##

# delta = mean_march - mean_feb

# w_stat = delta_hat/ SE_hat(delta_hat)

# Assumption : Data is poisson distributed

# Null Hypothesis : delta = 0
# Alternate Hypothesis : delta != 0


sample_mean_march = np.mean(march_data[handle])
sample_mean_feb = np.mean(feb_data[handle])

delta_hat = np.abs(sample_mean_march - sample_mean_feb)

SE_hat = np.sqrt(sample_mean_march/len(march_data) + sample_mean_feb/len(feb_data))

w_stat = np.abs(delta_hat/SE_hat)

print('Walds - 2 sample test')
print(' Standard Error Estimate', SE_hat)
print(' Delta Estimate ',delta_hat )
print(' w_stat ', w_stat)

# Reject Null Hypothesis

print('########################################')

## 2 sample unpaired t-test ##

# D_bar = X_bar - Y_bar
# t_stat = D_bar/root(corrected_var_x/n + corrected_var_y/m)

corrected_variance_march = np.sum(np.square(march_data[handle] - sample_mean_march))/(
corrected_variance_feb =  np.sum(np.square(feb_data[handle] - sample_mean_feb))/(len(f
```

```
D_bar = np.abs(sample_mean_march - sample_mean_feb)

t_stat = D_bar/np.sqrt(corrected_variance_march/len(march_data) + corrected_variance_f

print('2 sample unpaired t-test ')
print(' Corrected Variance March', corrected_variance_march)
print(' Corrected Variance Feb', corrected_variance_feb)
print(' D bar',D_bar )
print(' t_statistic ', t_stat)

# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')
```

```
Walds test
MLE for March data  302.96774193548384
Guess mean  302.17857142857144
Standard Error  3.1262042424737335
Walds Statistic 0.2524372835883352
########################################
Z test
 True Variance  309319.95639112673
 Sample Mean  302.96774193548384
 Guess  302.17857142857144
 z_statistic  0.007900375310135137
########################################
T-test
 Corrected Sample Standard Deviation 230.31174291540407
 Sample Mean  302.96774193548384
 Guess  302.17857142857144
 t_statistic  0.01907812154501079
########################################
Walds - 2 sample test
 Standard Error Estimate 4.534891928414112
 Delta Estimate  0.7891705069123987
 w_stat  0.17402189939030763
########################################
2 sample unpaired t-test
 Corrected Variance March 53043.498924731175
 Corrected Variance Feb 54679.6335978836
 D bar 0.7891705069123987
 t_statistic  0.013037597775749851
########################################
```

Hypothesis tests for ID deaths

The below results show the value of the statistic calculated for each of the following hypotheis tests:

- Wald's test

- Z-test
- T-test
- Wald's 2 sample test
- Unpaired 2-sample T-test

The Null hypothesis is rejected by all the above tests as the statistic calculated for each test is greater than the critical value for each of the above tests.

These tests are applicable as the number of samples can be assumed to be large. Thus, the mean is asymptotically normal using CLT. We estimate the true variance for the Z-test using the entire data.

```
########################### Tests for ID death cases handle ##########################


handle = 'ID deaths'
mean_feb = np.mean(feb_data[handle])

## Using mean_feb as guess for mean_march ##

# Wald's test #
# w = (theta_hat - theta_0)/ se_hat(theta_hat)
# theta_hat is estimator of theta

# Null Hypothesis : mean_march = mean_feb
# Alternate Hypothesis : mean_march != mean_feb

# Assuming the distribution of march data to be poisson. MLE_mean = Sample_mean

sample_mean_march = np.mean(march_data[handle])

standard_error_estimate = np.sqrt(sample_mean_march/len(march_data))

walds_statistic = np.abs((sample_mean_march - mean_feb)/standard_error_estimate)

print('Walds test')
print('MLE for March data ',sample_mean_march )
print('Guess mean ', mean_feb)
print('Standard Error ', standard_error_estimate)
print('Walds Statistic', walds_statistic )

# Reject Null Hypothesis(greater than 1.96)
print('########################################')


## Z-test ##

# z_statistic = (sample_mean - guess)/ root(true_variance/n)
# true variance = corrected sample standard deviation
```

```
sample_mean_full_data = np.mean(data[handle])
true_variance = np.sum(np.square(data[handle] - sample_mean_full_data))/(len(data)-1)

z_statistic = np.abs((sample_mean_march - mean_feb)/(np.sqrt(true_variance)/np.sqrt(le

print('Z test')
print(' True Variance ', true_variance)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' z_statistic ', z_statistic)


# Reject Null Hypothesis(greater than 1.96)

print('########################################')


## t-test ##

# t_statistic = (sample_mean - guess)/ corrected_sample_standard_deviation/root(n)
corrected_sample_SD = np.sqrt(np.sum(np.square(march_data[handle] - sample_mean_march)

t_statistic = np.abs((sample_mean_march - mean_feb)/(corrected_sample_SD/np.sqrt(len(m

print('T-test')
print(' Corrected Sample Standard Deviation', corrected_sample_SD)
print(' Sample Mean ',sample_mean_march )
print(' Guess ', mean_feb)
print(' t_statistic ', t_statistic)

# degrees of freedom : 28 + 31 -2 = 57
# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')

## Walds - 2 sample test ##

# delta = mean_march - mean_feb

# w_stat = delta_hat/ SE_hat(delta_hat)

# Assumption : Data is poisson distributed

# Null Hypothesis : delta = 0
# Alternate Hypothesis : delta != 0


sample_mean_march = np.mean(march_data[handle])
sample_mean_feb = np.mean(feb_data[handle])
```

```
delta_hat = np.abs(sample_mean_march - sample_mean_feb)

SE_hat = np.sqrt(sample_mean_march/len(march_data) + sample_mean_feb/len(feb_data))

w_stat = np.abs(delta_hat/SE_hat)

print('Walds - 2 sample test')
print(' Standard Error Estimate', SE_hat)
print(' Delta Estimate ',delta_hat )
print(' w_stat ', w_stat)

# Reject Null Hypothesis

print('########################################')

## 2 sample unpaired t-test ##

# D_bar = X_bar - Y_bar
# t_stat = D_bar/root(corrected_var_x/n + corrected_var_y/m)

corrected_variance_march = np.sum(np.square(march_data[handle] - sample_mean_march))/(
corrected_variance_feb =  np.sum(np.square(feb_data[handle] - sample_mean_feb))/(len(f

D_bar = np.abs(sample_mean_march - sample_mean_feb)

t_stat = D_bar/np.sqrt(corrected_variance_march/len(march_data) + corrected_variance_f

print('2 sample unpaired t-test ')
print(' Corrected Variance March', corrected_variance_march)
print(' Corrected Variance Feb', corrected_variance_feb)
print(' D bar',D_bar )
print(' t_statistic ', t_stat)

# threshold = 2.002465

# Reject Null Hypothesis

print('########################################')
```

```
Walds test
MLE for March data  3.2903225806451615
Guess mean  4.821428571428571
Standard Error  0.3257904818826477
Walds Statistic 4.699664587914285
########################################
Z test
 True Variance  57.36256961641746
 Sample Mean  3.2903225806451615
 Guess  4.821428571428571
 z_statistic  1.125568171640552
########################################
T-test
```

```
 Corrected Sample Standard Deviation 3.61656880101473
 Sample Mean  3.2903225806451615
 Guess  4.821428571428571
 t_statistic  2.357161674570531
########################################
Walds - 2 sample test
 Standard Error Estimate 0.5275730429394097
 Delta Estimate  1.5311059907834097
 w_stat  2.9021687352574856
########################################
2 sample unpaired t-test
 Corrected Variance March 13.079569892473122
 Corrected Variance Feb 26.003968253968257
 D bar 1.5311059907834097
 t_statistic  1.3174565296027811
########################################
```

✓ 0s completed at 2:45 PM

Question C):

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.

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.stats import geom
from scipy.stats import binom
```

```
%cd D:\StonyBrook\Study\Prob&Stats CSE544\Project

# from google.colab import drive
# drive.mount('/content/gdrive')

# %cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Mounted at /content/gdrive
/content/gdrive/My Drive/Prob_stats_proj
```

```
data = pd.read_csv('7.csv')
```

```
## converting date column to datetime data type ##
data['Date'] = pd.to_datetime(data['Date'])
```

Below printed is the data that is obtained from the CSV file for the states IA and ID.

```
data
```

	Date	IA confirmed cumulative	ID confirmed cumulative	IA deaths cumulative	ID deaths cumulative	IA confirmed	confi
0	2020-01-22	0	0	0	0	0	
1	2020-01-23	0	0	0	0	0	
2	2020-01-24	0	0	0	0	0	
3	2020-01-25	0	0	0	0	0	
4	2020-	0	0	0	0	0	

```
def get_eCDF(X):
    n = len(X)
    Srt = sorted(X)
    delta = .1
    X = []
    Y = [0]
    for i in range(0,n):
        X = X + [Srt[i]]
        Y = Y + [Y[len(Y)-1]+(1/n)]
    Y = Y + [1]

    return X,Y


def KS_Test_1_sample(X1,Y1, CDF_function, parameter):
    tot_max = -1

    Table = np.zeros((len(X1),4))
    for i in range(len(Table)):
        Table[i,0] = Y1[i]
        Table[i,1] = Y1[i+1]
        F_x = CDF_function(parameter, X1[i])
        Table[i,2] = abs(Table[i,0] - F_x)
        Table[i,3] = abs(Table[i,1] - F_x)
        cmax = max(Table[i,2], Table[i,3])
        if cmax > tot_max:
            tot_max = cmax


    return tot_max


def KS_test_2_sample(X1,Y1, X2,Y2):
    Table = np.zeros((len(X1),6))
    tot_max = -1
    for i in range(len(Table)):
```

```
        Table[i,0] = Y1[i]
        Table[i,1] = Y1[i+1]
        index1 = [idx for idx, val in enumerate(X2) if val >= X1[i]]
        index2 = [idx for idx, val in enumerate(X2) if val < X1[i]]
        if index1 == []:
            Table[i,3] = 1
        else :
            Table[i,3] = Y2[index1[0]]
        if index2 == []:
            Table[i,2] = 0
        else:
            Table[i,2] = Y2[index2[-1]]
        #print(index1, index2)

        #Table[i,3] = Y2[index1[0]]
        Table[i,4] = abs( Table[i,0] - Table[i,2])
        Table[i,5] = abs(Table[i,1] - Table[i,3])
        cmax = max(Table[i,4], Table[i,5])
        if cmax > tot_max:
            tot_max = cmax
            x1_max = X1[i]
            y1_max = Table[i,0]
            y2_max = Table[i,2]

    return tot_max


def get_Ti(n_perm, data, n1):
    T = []
    for i in range(n_perm):
        permute = np.random.permutation(len(data))
        D1 = data[permute[:n1]]
        D2 = data[permute[n1:]]
        T.append(abs(np.mean(D1) - np.mean(D2)))

    return np.array(T)

def get_p_value(T,T_obs):
    count = np.sum(T > T_obs)
    p_val = count/len(T)
    return p_val
```

We filter the data to obtain the data only in the date range '2020-10-01', '2020-12-31'.

```
###### Getting Oct- Dec 2020 data ######
start_date, end_date = '2020-10-01', '2020-12-31'
condition = (data['Date'] >= start_date) & (data['Date'] <= end_date)
oct_dec_data = data.loc[condition]


################## 1- sample KS test ####################
```

```
################### 1- sample KS test ####################
```

1- Sample KS test

Tests for IA confirmed cases with ID confirmed cases

```
###### Tests for IA confirmed cases with ID confirmed cases ######

handle = 'IA confirmed'
test_handle = 'ID confirmed'

# Obtaining eCDF for test_handle
test_handle_data = oct_dec_data[test_handle]

test_X, test_Y = get_eCDF(test_handle_data)
```

Poisson distribution :

```
# Assuming Poisson distribution

def MME_Poisson(X):
    estimate = np.mean(X)
    return estimate

def CDF_Poisson(lambda_, x):
    prob = poisson.cdf(x, lambda_)
    return prob

CDF_dist = CDF_Poisson

# Obtaining MME for IA confirmed cases
print('####### Poisson Distribution ########')
lambda_ = MME_Poisson(oct_dec_data[handle])
print(' Poisson parameter lambda : ', lambda_)

# KS-test to be performed on test_handle
KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, lambda_ )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05

# Reject Null hypothesis
```

```
    ####### Poisson Distribution ########
```

```
Poisson parameter lambda :  2084.217391304348
KS statistic :  0.9238499249850628
```

Result of 1 sample ks test for poisson distribution:

Null hypothesis (H0) : Distribution of confirmed cases in the period equals poisson distribution.

Alternate hypothesis (H1) : Distribution of confirmed cases in the period is not poisson distribution.

Procedure: We have obtained the lambda parameter for poisson distribution by using MME on the IA state's data which is the mean of the distribution. Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.808 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Geometric distribution :

```
# Assuming Geometric distribution

def MME_Geometric(X):
    sample_mean = np.mean(X)
    estimate = 1/sample_mean

    return estimate

def CDF_Geometric(p,x):
    prob = geom.cdf(x, p)
    return prob

print('####### Geometric Distribution ########')
p = MME_Geometric(oct_dec_data[handle])
CDF_dist = CDF_Geometric

print(' Geometric parameter : ', p)

KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, p )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05

# Reject Null hypothesis
```

```
####### Geometric Distribution ########
 Geometric parameter :  0.00047979639944093286
 KS statistic :  0.3295560815317995
```

Result of 1 sample ks test for geometric distribution:

Null hypothesis (H0) : Distribution of confirmed cases in the period equals geometric distribution.

Alternate hypothesis (H1) : Distribution of confirmed cases in the period is not geometric distribution.

Procedure: We have obtained the geometric parameter for geometric distribution by using MME on the IA state's data which is the (1/mean of the distribution). Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.271 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Binomial distribution :

```
# Assuming Binomial distribution


def MME_Binomial(X):
    mean = np.mean(X)
    var = np.var(X)
    estimate_p = 1 - (var/mean)
    estimate_n = mean/estimate_p

    #print(mean,var)

    return estimate_p,estimate_n

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

print('####### Binomial Distribution ########')
n,p = MME_Binomial(oct_dec_data[handle])
CDF_dist = CDF_Binomial

print(' Binomial parameters(n,p) : ', n,p)

KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, [n,p] )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05
```

```
# Reject Null hypothesis

    ####### Binomial Distribution ########
     Binomial parameters(n,p) :  -986.4541984905915 -2.1128374682711906
     KS statistic :  0.9891304347826086
```

Result of 1 sample ks test for binomial distribution:

Null hypothesis (H0) : Distribution of confirmed cases in the period equals binomial distribution.

Alternate hypothesis (H1) : Distribution of confirmed cases in the period is not binomial distribution.

Procedure: We have obtained the binomial parameter using the formula in the def MME_Binomial and IA state data. Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.989 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Tests for IA death cases with ID death cases

```
###### Tests for IA death cases with ID death cases ######

handle = 'IA deaths'
test_handle = 'ID deaths'

test_handle_data = oct_dec_data[test_handle]

test_X, test_Y = get_eCDF(test_handle_data)
```

Poisson distribution :

```
# Assuming Poisson distribution
CDF_dist = CDF_Poisson

print('####### Poisson Distribution ########')
lambda_ = MME_Poisson(oct_dec_data[handle])
print(' Poisson parameter lambda : ', lambda_)

KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, lambda_ )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05
```

```
# Critical threshold is 0.05

# Reject Null hypothesis
```

```
####### Poisson Distribution ########
 Poisson parameter lambda :  27.706521739130434
 KS statistic :  0.7565852505895584
```

Result of 1 sample ks test for poisson distribution:

Null hypothesis (H0) : Distribution of deaths in the period equals poisson distribution.

Alternate hypothesis (H1) : Distribution of deaths in the period is not poisson distribution.

Procedure: We have obtained the lambda parameter for poisson distribution by using MME on the IA deaths data which is the mean of the distribution. Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.748 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Geometric distribution

```
# Assuming Geometric distribution

print('####### Geometric Distribution ########')
p = MME_Geometric(oct_dec_data[handle])
CDF_dist = CDF_Geometric

print(' Geometric parameter : ', p)

KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, p )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05

# Reject Null hypothesis
```

```
####### Geometric Distribution ########
 Geometric parameter :  0.036092585327579446
 KS statistic :  0.316365783780573
```

Result of 1 sample ks test for geometric distribution:

Null hypothesis (H0) : Distribution of deaths in the period equals geometric distribution.

Alternate hypothesis (H1) : Distribution of deaths in the period is not geometric distribution.

Procedure: We have obtained the geometric parameter for geometric distribution by using MME on the IA deaths data which is the (1/mean of the distribution). Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.373 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Binomial distribution:

```
# Assuming Binomial distribution
print('####### Binomial Distribution ########')
n,p = MME_Binomial(oct_dec_data[handle])
CDF_dist = CDF_Binomial

print(' Binomial parameters(n,p) : ', n,p)

KS_value = KS_Test_1_sample(test_X, test_Y, CDF_dist, [n,p] )

print(' KS statistic : ', KS_value)

# Critical threshold is 0.05

# Reject Null hypothesis
```

```
####### Binomial Distribution ########
 Binomial parameters(n,p) :  -115.31191686424339 -0.24027457432477942
 KS statistic :  1.0
```

Result of 1 sample ks test for binomial distribution:

Null hypothesis (H0) : Distribution of deaths in the period equals binomial distribution.

Alternate hypothesis (H1) : Distribution of deaths in the period is not binomial distribution.

Procedure: We have obtained the binomial parameter using the formula in the def MME_Binomial using IA deaths data. Then calculated the maximum differences between all the points in the cdf. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 1.0 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

KS 2-sample Test

Tests for IA confirmed cases with ID confirmed cases

```
###### Tests for IA confirmed cases with ID confirmed cases ######

handle = 'IA confirmed'
test_handle = 'ID confirmed'

# Obtaining eCDF for handles
handle_data = oct_dec_data[handle]

test_handle_data = oct_dec_data[test_handle]

X1, Y1 = get_eCDF(handle_data)
X2, Y2 = get_eCDF(test_handle_data)


KS_value = KS_test_2_sample(X1,Y1, X2,Y2)

print(' KS statistic : ', KS_value)

# Reject Null Hypothesis
```

```
 KS statistic :  0.369565217391306
```

Result of 2 sample ks test for IA confirmed and ID confirmed

Null hypothesis (H0) : Distribution of confirmed cases in the IA state equals Distribution of confirmed cases in the ID state

Alternate hypothesis (H1) : Distribution of confirmed cases in the IA state not equals Distribution of confirmed cases in the ID state

Procedure: Generate the cdf for both the distributions(both states). Then we apply the KS 2 sample test to get the maximum difference between the distributions. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.2808 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Tests for IA death cases with ID death cases

```
###### Tests for IA death cases with ID death cases ######

handle = 'IA deaths'
test_handle = 'ID deaths'

# Obtaining eCDF for handles
handle_data = oct_dec_data[handle]

test_handle_data = oct_dec_data[test_handle]

X1, Y1 = get_eCDF(handle_data)
X2, Y2 = get_eCDF(test_handle_data)


KS_value = KS_test_2_sample(X1,Y1, X2,Y2)

print(' KS statistic : ', KS_value)

# Reject Null Hypothesis
```

```
KS statistic :  0.28260869565217406
```

Result of 2 sample ks test for IA deaths and ID deaths

Null hypothesis (H0) : Distribution of deaths in the IA state equals Distribution of deaths in the ID state

Alternate hypothesis (H1) : Distribution of deaths in the IA state not equals Distribution of deaths in the ID state

Procedure: Generate the cdf for both the distributions(both states). Then we apply the KS 2 sample test to get the maximum difference between the distributions. The critical value of 0.05 is used as mentioned in the question.

As the KS value obtained is 0.236 and c = 0.05. As KS value is greater than critical value, we reject the null hypothesis.

Is KS test applicable? There are no assumptions under KS test. Hence the test is applicable.

Permutation test

Test for IA confirmed cases with ID confirmed cases

```
###### Test for IA confirmed cases with ID confirmed cases ######
```

```
###### Test for IA confirmed cases with ID confirmed cases ######

handle = 'IA confirmed'
test_handle = 'ID confirmed'

# Obtaining eCDF for handles
handle_data = oct_dec_data[handle]

test_handle_data = oct_dec_data[test_handle]

T_obs = np.abs(np.mean(handle_data) - np.mean(test_handle_data))

print(" T_observed is : " ,T_obs)

total_data = np.concatenate((np.array(handle_data) , np.array(test_handle_data)))

T_i = get_Ti(1000, total_data, len(handle_data))

p = get_p_value(T_i, T_obs)

print(' p statistic : ', p)

# Reject Null Hypothesis
```

```
     T_observed is :  1007.8478260869567
     p statistic :  0.0
```

Result of Permutation test for IA confirmed and ID confirmed

Null hypothesis (H0) : Distribution of confirmed cases in the IA state equals Distribution of confirmed cases in the ID state

Alternate hypothesis (H1) : Distribution of confirmed cases in the IA state not equals Distribution of confirmed cases in the ID state

Procedure: Find T_obs value by using:
T_obs = | mean(IA confirmed) - mean(ID_confirmed) | Then we concatenate all the confirmed cases counts from both train and test sets. And then create 1000 permutations from the set and calculate the p-value for each of the permutation generated by using the same formula mentioned above for the permuted two partitions. Then we count the number of permutations that resulted in a p value greater than T_obs. P-value is calculated by divind count obtained by 1000. If the p-value is less than or equal to c. Then we reject the null hypothesis.

Result: As the obtained p-value is 0 which is lower than the given threshold of 0.05. We reject the null hypothesis.

Is Permutation test applicable? There are no assumptions under Permutation test. Hence the test is applicable.

Test for IA death cases with ID death cases

```
###### Test for IA death cases with ID death cases ######


handle = 'IA deaths'
test_handle = 'ID deaths'

# Obtaining eCDF for handles
handle_data = oct_dec_data[handle]

test_handle_data = oct_dec_data[test_handle]

T_obs = np.abs(np.mean(handle_data) - np.mean(test_handle_data))

print(" T_observed is : " ,T_obs)


total_data = np.concatenate((np.array(handle_data) , np.array(test_handle_data)))

T_i = get_Ti(1000, total_data, len(handle_data))

p = get_p_value(T_i, T_obs)

print(' p statistic : ', p)

# Reject Null Hypothesis
```

```
T_observed is :  17.195652173913043
p statistic :  0.0
```

Result of Permutation test for IA deaths and ID deaths

Null hypothesis (H0) : Distribution of deaths in the IA state equals Distribution of deaths in the ID state

Alternate hypothesis (H1) : Distribution of deaths in the IA state not equals Distribution of deaths in the ID state

Procedure: Find T_obs value by using: T_obs = | mean(IA confirmed) - mean(ID_confirmed) | Then we concatenate all the deaths counts from both train and test sets. And then create 1000 permutations from the set and calculate the p-value for each of the permutation generated by using the same formula mentioned above for the permuted two partitions. Then we count the number of permutations that resulted in a p value greater than T_obs. P-value is calculated by divind count obtained by 1000. If the p-value is less than or equal to c. Then we reject the null hypothesis.

Result: As the obtained p-value is 0 which is lower than the given threshold of 0.05. We reject the null hypothesis.

Is Permutation test applicable? There are no assumptions under Permutation test. Hence the test is applicable.

Question D) :

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 λ. Assume an Exponential prior (with mean β) on λ. Assume β = λMME 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 λ 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.

```
import pandas as pd
import numpy as np
from scipy.stats import gamma
import matplotlib.pyplot as  plt
```

```
%cd D:\StonyBrook\Study\Prob&Stats CSE544\Project

# from google.colab import drive
# drive.mount('/content/gdrive')

# %cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Mounted at /content/gdrive
/content/gdrive/My Drive/Prob_stats_proj
```

```
data = pd.read_csv('7.csv')
```

```
## converting date column to datetime data type ##
data['Date'] = pd.to_datetime(data['Date'])
```

```
data
```

	Date	IA confirmed cumulative	ID confirmed cumulative	IA deaths cumulative	ID deaths cumulative	IA confirmed	ID confirmed
0	2020-01-22	0	0	0	0	0	0
1	2020-01-23	0	0	0	0	0	0
2	2020-01-24	0	0	0	0	0	0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438 entries, 0 to 437
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date                     438 non-null    datetime64[ns]
 1   IA confirmed cumulative  438 non-null    int64
 2   ID confirmed cumulative  438 non-null    int64
 3   IA deaths cumulative     438 non-null    int64
 4   ID deaths cumulative     438 non-null    int64
 5   IA confirmed             438 non-null    int64
 6   ID confirmed             438 non-null    int64
 7   IA deaths                438 non-null    int64
 8   ID deaths                438 non-null    int64
dtypes: datetime64[ns](1), int64(8)
memory usage: 30.9 KB
```

```
#### June - July 2020 data ######

june_start_date, june_end_date = '2020-06-01' , '2020-06-28' # 28 days only
posterior_week_start_date, posterior_week_end_date = '2020-06-29' , '2020-07-05'

condition = (data['Date'] >= june_start_date) & (data['Date'] <= june_end_date)
june_28_data = data.loc[condition]

condition = (data['Date'] >= posterior_week_start_date) & (data['Date'] <= posterior_w
posterior_week_data = data.loc[condition]


##### summing up deaths data #####
handle_state_1 = 'IA deaths'
handle_state_2 = 'ID deaths'

june_28_data = june_28_data[handle_state_1] + june_28_data[handle_state_2]

posterior_week_data_ = posterior_week_data[handle_state_1] + posterior_week_data[hand]
```

Bayesian Inference

Procedure: We calculated lambda value using MME for Poisson distribution for first 28 days of data. Using this lambda we calculated beta as 1/lambda and subsituted this in the prior. Next when finding the posterior, the distribution comes out as gamma where in every loop we ffind the powers of exponential and lambda and store it in table. Next we plotted the gamma distribution with MAP =

```
def MME_Poisson(X):
    estimate = np.mean(X)
    return estimate

def get_estimates(exp_a, lambda_a, exp_b, lambda_b):
    return exp_a + exp_b, lambda_a + lambda_b

def plot_gamma(table):
    for estimate in table:
        alpha, beta = estimate[0], estimate[1]
        x = np.linspace(gamma.ppf(0.01, alpha, scale=1/beta),gamma.ppf(0.99, alpha,sca
        MAP = (alpha)/beta
        plt.plot(x, gamma.pdf(x, alpha,scale=1/beta), label = 'MAP: %.4f ' %(MAP))
        plt.xlabel('x')
        plt.ylabel('pdf')
    plt.legend(loc="upper right")
    plt.title('Gamma curves of estimates')
    plt.show()


# Assuming it to be poisson distributed

# Obtaining MME

lambda_ = MME_Poisson(june_28_data)

print(' MME of poisson distributed data ', lambda_)

# prior beta
exp_lambda = 1/lambda_

print(' Prior beta value ', exp_lambda)

## Since the prior is exponential and likelihood is poisson, the posterior is gamma di

likelihood_exp_power = len(posterior_week_data_)
likelihood_lambda_power = np.sum(posterior_week_data_)
prior_exp_power = exp_lambda
prior_lambda_power = 0

table = []
for i in range(4): # till 8th week
    prior_exp_power, prior_lambda_power = get_estimates(likelihood_exp_power, likeliho
```

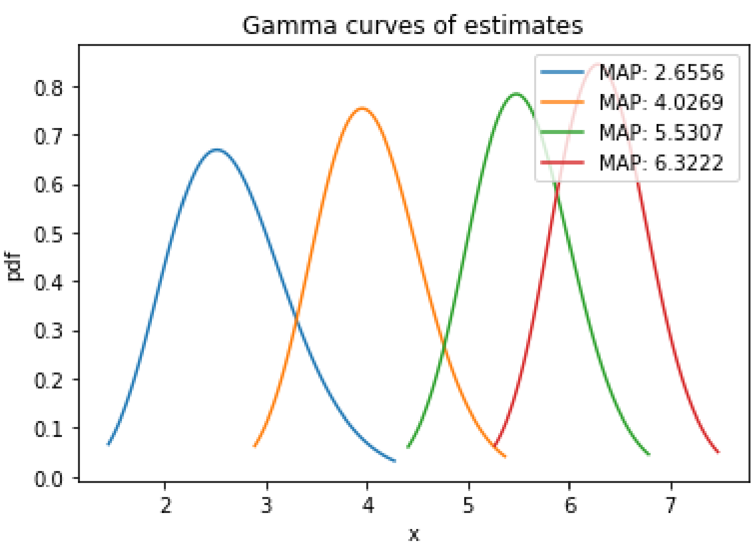
```
    table.append([prior_lambda_power + 1,prior_exp_power])

    condition+=7
    posterior_week_data = data[166 + 7*i : 173 + 7*i]

    posterior_week_data_ = posterior_week_data[handle_state_1] + posterior_week_data[h

    likelihood_exp_power = len(posterior_week_data_)
    likelihood_lambda_power = np.sum(posterior_week_data_)
```

```
MME of poisson distributed data  6.464285714285714
Prior beta value  0.15469613259668508
```

```
print(table)
```

```
[[19, 7.154696132596685], [57, 14.154696132596685], [117, 21.154696132596683], [
```

```
plot_gamma(table)
```



Exploratory task:

We have taken bitcoin pricing as our dataset as there has been a surge in the investors since the pandemic started due to various reasons like:

1. Increase in the number of investors in the market.
2. People investing their unspent money during pandemic beacuse their expenditures got lower.
3. Shortage of bitcoins available in the market

So we decided on inferring hypothesis between the covid cases and the bitcoin price. If they had a effect on each other during the 2020 pandemic period.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
#%cd D:\StonyBrook\Study\Prob&Stats CSE544\Project

from google.colab import drive
drive.mount('/content/gdrive')

%cd /content/gdrive/My Drive/Prob_stats_proj
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call d
/content/gdrive/My Drive/Prob_stats_proj
```

```
bitcoin_price_data = pd.read_csv('bitcoin_pricing.csv')
us_all_covid_data = pd.read_csv('US_confirmed_totals.csv')
```

```
## converting date column to datetime data type ##
bitcoin_price_data['Date'] = pd.to_datetime(bitcoin_price_data['Date'])
us_all_covid_data['Date'] = pd.to_datetime(us_all_covid_data['Date'])
```

```
us_all_covid_data
```

	Date	Confirmed cases cumulative	Confirmed cases
0	2020-01-22	1	0
1	2020-01-23	1	0
2	2020-01-24	2	1
3	2020-01-25	2	0
4	2020-01-26	5	3
...	...	...	...
433	2021-03-30	29796319	52741

```
bitcoin_price_data
```

	Date	Closing Price	Change
0	2013-10-01	123.655	0.000
1	2013-10-02	125.455	1.800
2	2013-10-03	108.585	-16.870
3	2013-10-04	118.675	10.090
4	2013-10-05	121.339	2.664
...	...	...	...
2776	2021-05-08	58788.210	1681.089
2777	2021-05-09	58102.191	-686.019
2778	2021-05-10	55715.547	-2386.644
2779	2021-05-11	56573.555	858.008
2780	2021-05-12	52147.821	-4425.734

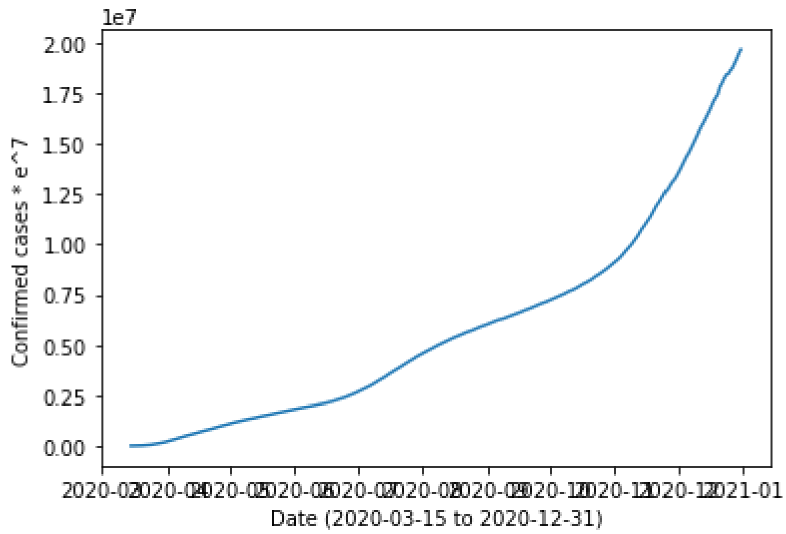
2781 rows × 3 columns

```
# data.info()
```

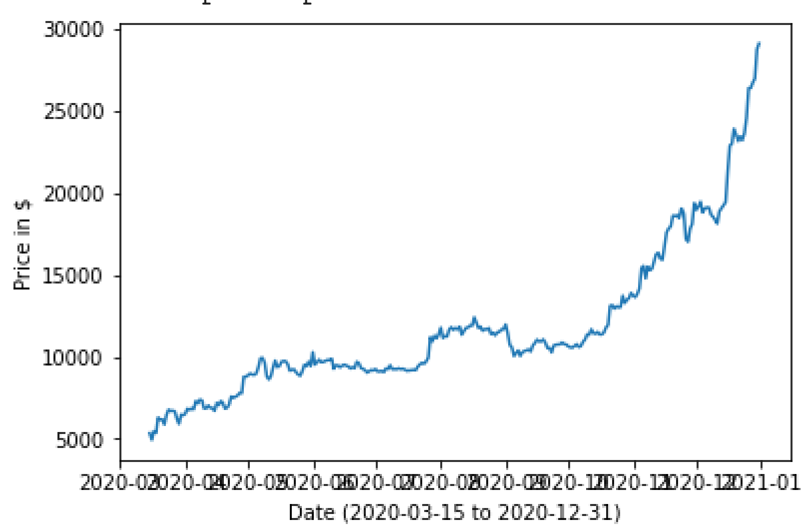
```
# Create three months data objs in here

#### June - July 2020 data ######
march_august_start_date, march_august_end_date = '2020-03-15' , '2020-12-31' # March (

condition = (bitcoin_price_data['Date'] >= march_august_start_date) & (bitcoin_price_c
march_august_bitcoin_price_data = bitcoin_price_data.loc[condition]

condition = (us_all_covid_data['Date'] >= march_august_start_date) & (us_all_covid_dat
march_august_covid_data = us_all_covid_data.loc[condition]
```

```
print("** Covid confirmed cases plot from 2020-03-15 to 2020-12-31 **" )
plt.plot(march_august_covid_data['Date'], march_august_covid_data['Confirmed cases cum
plt.xlabel('Date (2020-03-15 to 2020-12-31)')
plt.ylabel('Confirmed cases * e^7')
plt.show()

print("** Bit coin price plot from 2020-03-15 to 2020-12-31 **" )
plt.plot(march_august_bitcoin_price_data['Date'], march_august_bitcoin_price_data['Clo
plt.xlabel('Date (2020-03-15 to 2020-12-31)')
plt.ylabel('Price in $')
plt.show()
```

```
** Covid confirmed cases plot from 2020-03-15 to 2020-12-31 **
```




```
** Bit coin price plot from 2020-03-15 to 2020-12-31 **
```



Walds - 2 Sample test

delta = normalized_growth_of_cases - normalized_growth_of_bitcoin_value

Null Hypothesis : delta = 0 i.e. if the growth rate of bitcoin was proportional to the confirmed covid cases growth

Alternate Hypothesis : delta != 0 if the growth rate of bitcoin was not proportional to the confirmed covid cases growth

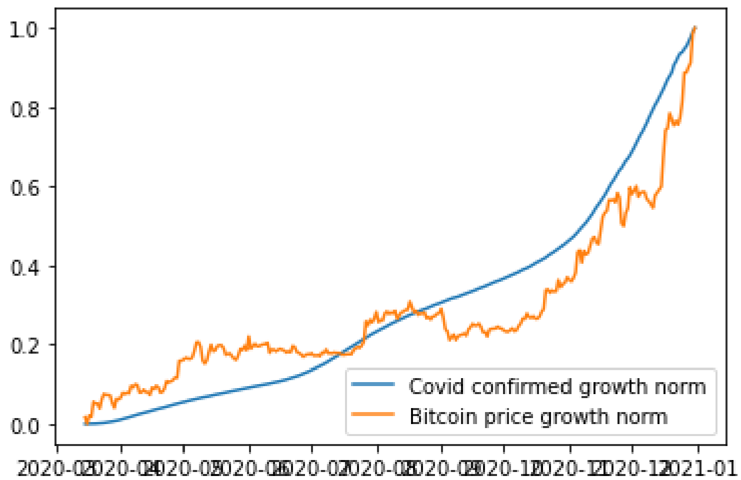
w_stat = delta_hat/ SE_hat(delta_hat)

alpha = 0.05 z_alpha/2 = 1.96

Procedure: The growth rate of covid cases and bitcoin during the period is normalised to show how the values changed from the start of period to the end of period. This graphs are plotted below to see the comparision. Then the two sided walds test is performed to check if their rate of growth was similar during the period.

```
## Walds - 2 sample test ##

handle = 'Confirmed cases cumulative'
test_handle = 'Closing Price'

march_august_covid_data_norm = (march_august_covid_data[handle] - np.min(march_august_

march_august_bitcoin_price_data_norm = (march_august_bitcoin_price_data[test_handle] -

print("** Normalised growth graphs for Covid cases and bitcoin price between dates 202
plt.plot(march_august_covid_data['Date'], march_august_covid_data_norm, label = 'Covic
plt.plot(march_august_bitcoin_price_data['Date'], march_august_bitcoin_price_data_norm
plt.legend(loc="lower right")
plt.show()

mean_x = np.mean(march_august_covid_data_norm)
mean_y = np.mean(march_august_bitcoin_price_data_norm)

delta_hat = np.abs(mean_x - mean_y )

SE_hat = np.sqrt( (np.var(march_august_covid_data_norm) / np.shape(march_august_covid_

w_stat = np.abs(delta_hat/SE_hat)

print(' Standard Error Estimate', SE_hat)
print(' Delta Estimate ',delta_hat )
print(' w_stat ', w_stat)

# Accept null hypothesis as w is less than 1.96

#-
#-
#-
#-
#-
#-
#-
```

```
** Normalised growth graphs for Covid cases and bitcoin price between dates 2020·
```




```
Standard Error Estimate 0.01860144411111878
Delta Estimate  0.015573072250839148
w_stat  0.8371969486783308
```

Result: On performing 2 sided walds test. We obtained a result w_statistic of 0.837. As we have considered the value of alpha as 0.05, so Z_alpha would be 1.96. As w_stat is less that 1.96. We accept the null hypothesis H0.

K-S test

Then we decided to check of the per day new covid cases and the each day value change of bitcoin had any particular distribution match on a daily basis i.e. if more covid cases in a day would result in rise of bitcoin pricing.

So to check if the both followed the same distribution. We ploted the cdf of both the distributions and done the 2 sample KS test for alpha 0.05.

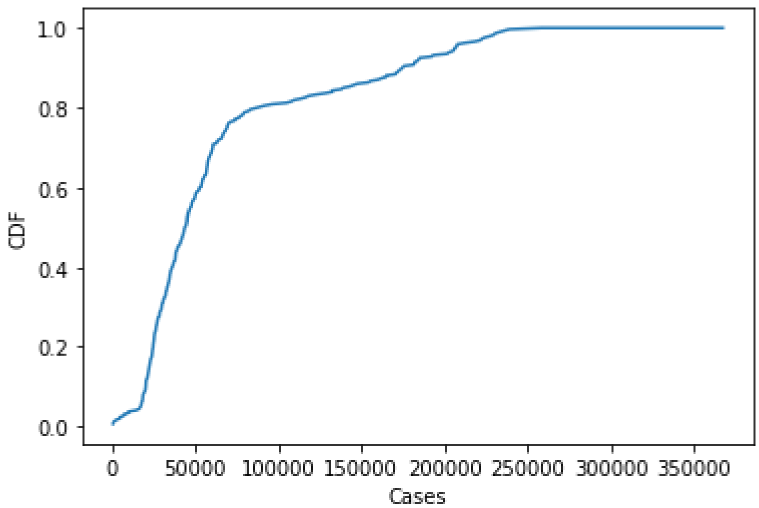
Null hypothesis H0 : Confirmed covid cases distribution equals bitcoin price change distributionn

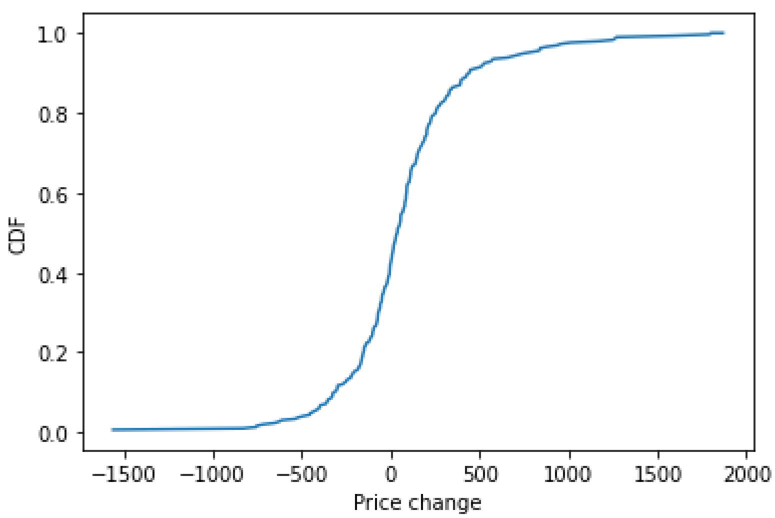
Alternate hypothesis H1: Confirmed covid cases distribution not equals bitcoin price change distributionn

```
#Functions required for K-s test
def get_eCDF(X):
    n = len(X)
    Srt = sorted(X)
    delta = .1
    X = []
    Y = [0]
    for i in range(0,n):
        X = X + [Srt[i]]
        Y = Y + [Y[len(Y)-1]+(1/n)]
```

```
    Y = Y + [1]

    return X,Y


def KS_test_2_sample(X1,Y1, X2,Y2):
    Table = np.zeros((len(X1),6))
    tot_max = -1
    for i in range(len(Table)):
        Table[i,0] = Y1[i]
        Table[i,1] = Y1[i+1]
        index1 = [idx for idx, val in enumerate(X2) if val >= X1[i]]
        index2 = [idx for idx, val in enumerate(X2) if val < X1[i]]
        if index1 == []:
            Table[i,3] = 1
        else :
            Table[i,3] = Y2[index1[0]]
        if index2 == []:
            Table[i,2] = 0
        else:
            Table[i,2] = Y2[index2[-1]]
        #print(index1, index2)

        #Table[i,3] = Y2[index1[0]]
        Table[i,4] = abs( Table[i,0] - Table[i,2])
        Table[i,5] = abs(Table[i,1] - Table[i,3])
        cmax = max(Table[i,4], Table[i,5])
        if cmax > tot_max:
            tot_max = cmax
            x1_max = X1[i]
            y1_max = Table[i,0]
            y2_max = Table[i,2]

    return tot_max


######## KS 2-sample Test ##########

###### Tests for IA confirmed cases with ID confirmed cases ######

handle = 'Confirmed cases'
test_handle = 'Change'
#test_handle = 'Traveler Throughput'

# Obtaining eCDF for handles
handle_data = march_august_covid_data[handle]

test_handle_data = march_august_bitcoin_price_data[test_handle]

X1, Y1 = get_eCDF(handle_data)
X2, Y2 = get_eCDF(test_handle_data)
```

```
plt.plot(X1,Y1[2:], label = 'CDF - Daily new covid confirmed cases')
plt.xlabel("Cases")
plt.ylabel("CDF")
plt.show()
plt.plot(X2 ,Y2[2:], label = 'CDF - Bitcoin value change per day in dollors')
plt.xlabel("Price change")
plt.ylabel("CDF")
plt.show()

KS_value = KS_test_2_sample(X1,Y1, X2,Y2)

print(' KS statistic : ', KS_value)


# Reject Null Hypothesis as statistic is 0.98
```






```
KS statistic :  0.9863013698630171
```

Result: On calculating the 2 sample KS test. we obtained a KS- statistic of 0.9863. Which is far higher than 0.05. This is understandable as the bitcoin price change was negative as well though the confirmed covid cases was raising.

So we reject the null hypothesis.

Pearson correlation:

Though it was clearly visible that the bitcoin pricing also increased during the pandemic. But we also wanted to get that done using inference methods. So we applied the pearson test to know the correlation between the covid confirmed cases and bitcoin price.

Null hypothesis H0: Covid confirmed cases and bitcoin pricing are correlated.

Alternate hypothesis H1: Covid confirmed cases and bitcoin pricing are not correlated.

```
handle = 'Confirmed cases cumulative'
test_handle = 'Closing Price'

bitcoin_price_mean = np.mean(march_august_bitcoin_price_data[test_handle])

covid_confirmed_mean = np.mean(march_august_covid_data[handle])

numer = 0
denom_x = 0
denom_y = 0

#print(march_august_bitcoin_price_data)
for i in range(march_august_bitcoin_price_data.shape[0]):

  x_minus_Xavg = np.array(march_august_bitcoin_price_data[test_handle])[i] - bitcoin_p
  y_minus_Yavg = np.array(march_august_covid_data[handle])[i] - covid_confirmed_mean
  numer = numer + (x_minus_Xavg * y_minus_Yavg)

  denom_x =  denom_x + np.square(x_minus_Xavg)
  denom_y =  denom_y + np.square(y_minus_Yavg)


coeff = numer / np.sqrt(denom_x * denom_y)

print(coeff)

#as coeff is > 0.5 . They are positively correlated
```

```
0.9585374248082369
```

Result: On performing the Pearson correlation test we obtained a coefficient value of 0.959. As the value is far greater than 0.5. We **accept the null hypothesis** and are very confident that the rise in covid cases and bitcoin price were directly proportial and **postively correlated**.