Nikhil Kurmadasu(114361880)

Krishna Teja Reddy Chinnakotla(114507668)

Pranith Kumar Raparthi(113217634)

Varun Doniparti(113270819)

Lakshmi Datta Meghana Siddavatam(114777708)

Data Exploration

In [1]:
```
# First we will import the required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import math
import scipy.stats as st
from scipy.stats import norm,poisson,binom,geom,gamma
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_dark"
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
# Read the covid cases dataset
df_covid=pd.read_csv("United_States_COVID-19_Cases_and_Deaths_by_State_over_Time.csv")
df_covid.head()
```

Out[2]:

	submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_death	conf_death	prob_dea
0	01/14/2022	KS	621273	470516.0	150757.0	19414	6964.0	7162	NaN	Na
1	01/02/2022	AS	11	NaN	NaN	0	0.0	0	NaN	Na
2	04/09/2022	FL	5866358	NaN	NaN	1565	253.0	73809	NaN	Na
3	07/23/2020	TX	361125	NaN	NaN	9507	0.0	7981	NaN	Na
4	08/12/2020	AS	0	NaN	NaN	0	0.0	0	NaN	Na

In [3]:
```
# columns present in the covid _cases data
df_covid.columns
```

Out[3]:
```
Index(['submission_date', 'state', 'tot_cases', 'conf_cases', 'prob_cases',
       'new_case', 'pnew_case', 'tot_death', 'conf_death', 'prob_death',
       'new_death', 'pnew_death', 'created_at', 'consent_cases',
       'consent_deaths'],
      dtype='object')
```

Let us first choose part of data that is in our interest, that is of states Kansas(KS) and Illinois(IL)

In [4]:
```
df_case = df_covid.loc[df_covid['state'].isin(['KS', 'IL'])]
```

In [5]:
```
# Now let us change the date in string format to datetime format
df_case['submission_date']= pd.to_datetime(df_case['submission_date'])
```

Number of Nan / missing values in each column of data

In [6]:
```
print("Column_name ---- Count of Null Values")
df_case.isnull().sum()
```

Out[6]:
```
Column_name ---- Count of Null Values
submission_date      0
state                0
tot_cases            0
conf_cases         177
prob_cases         177
new_case             0
pnew_case          177
tot_death            0
conf_death         930
prob_death         930
new_death            0
pnew_death         177
created_at           0
consent_cases        0
consent_deaths     842
dtype: int64
```

Our required columns for the task are "state" ,"submission_date" ,"tot_cases" ,"tot_death" ,"new_case" ,"new_death"

Based on our observation and our requirement for the present task none of the above mentiones required columns have null values. So we remove all the other columns which eleminates columns containing null values too.

Select required columns and Remove unwanted columns

In [7]:
```
df_case_final=(df_case[["state","submission_date","tot_cases","tot_death"]])
```

In [8]:
```
# Split data between states and Sort based on Date for both states individually
df_case_final=df_case_final.sort_values(['state', 'submission_date'])
```

In [9]:
```
df_case_final.head()
```

Out[9]:

	state	submission_date	tot_cases	tot_death
27554	IL	2020-01-22	0	0
26382	IL	2020-01-23	0	0
32726	IL	2020-01-24	1	0
28933	IL	2020-01-25	1	0
32825	IL	2020-01-26	1	0

let us compute cases occured per day and deaths occured per day by subtracting consecutive rows

In [10]:
```
df_case_final['per_day_cases'] = df_case_final['tot_cases'] - df_case_final['tot_cases'].s
df_case_final['per_day_deaths'] = df_case_final['tot_death'] - df_case_final['tot_death'].
```

In [11]:
```
df_case_final['per_day_cases'] = df_case_final['per_day_cases'].fillna(0)
df_case_final['per_day_deaths'] = df_case_final['per_day_deaths'].fillna(0)
```

In [12]:
```
df_case_final.reset_index(drop=True, inplace=True)
```

In [13]:
```
# Set the value of per day cases and per day deaths to 0 for the starting date of Kansas s
# is changing from IL and KS.So difference shouldn't be done here.

df_case_final.at[842, 'per_day_deaths'] = 0
df_case_final.at[842, 'per_day_cases'] = 0
```

In [14]:
```
df_case_final["per_day_deaths"] = df_case_final["per_day_deaths"].astype(int)
df_case_final["per_day_cases"] = df_case_final["per_day_cases"].astype(int)
```

In [15]:
```
df_case_final.head()
```

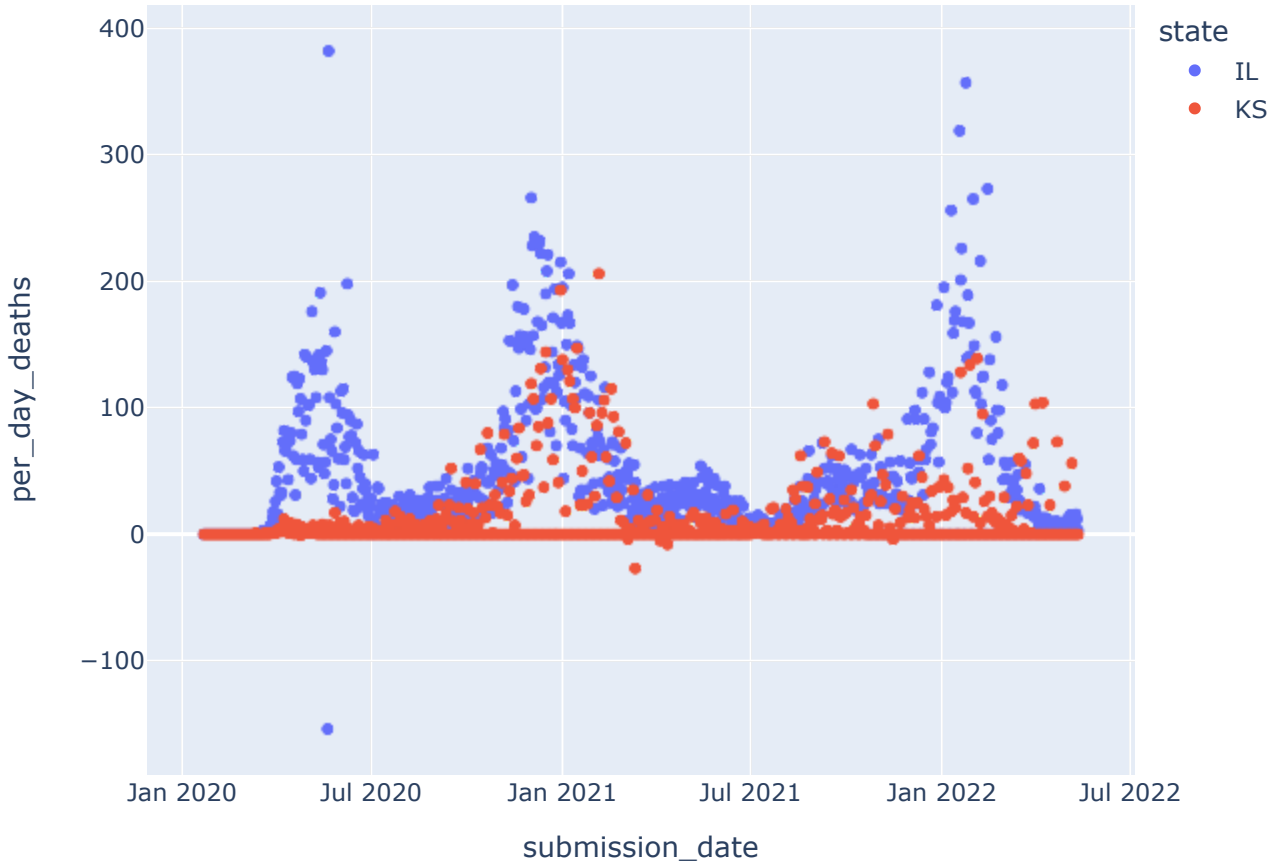
Out[15]:

	state	submission_date	tot_cases	tot_death	per_day_cases	per_day_deaths
0	IL	2020-01-22	0	0	0	0
1	IL	2020-01-23	0	0	0	0
2	IL	2020-01-24	1	0	1	0
3	IL	2020-01-25	1	0	0	0
4	IL	2020-01-26	1	0	0	0

In [16]:
```
df_case_IL = df_case_final[df_case_final['state'] == 'IL']
df_case_KS = df_case_final[df_case_final['state'] == 'KS']
```

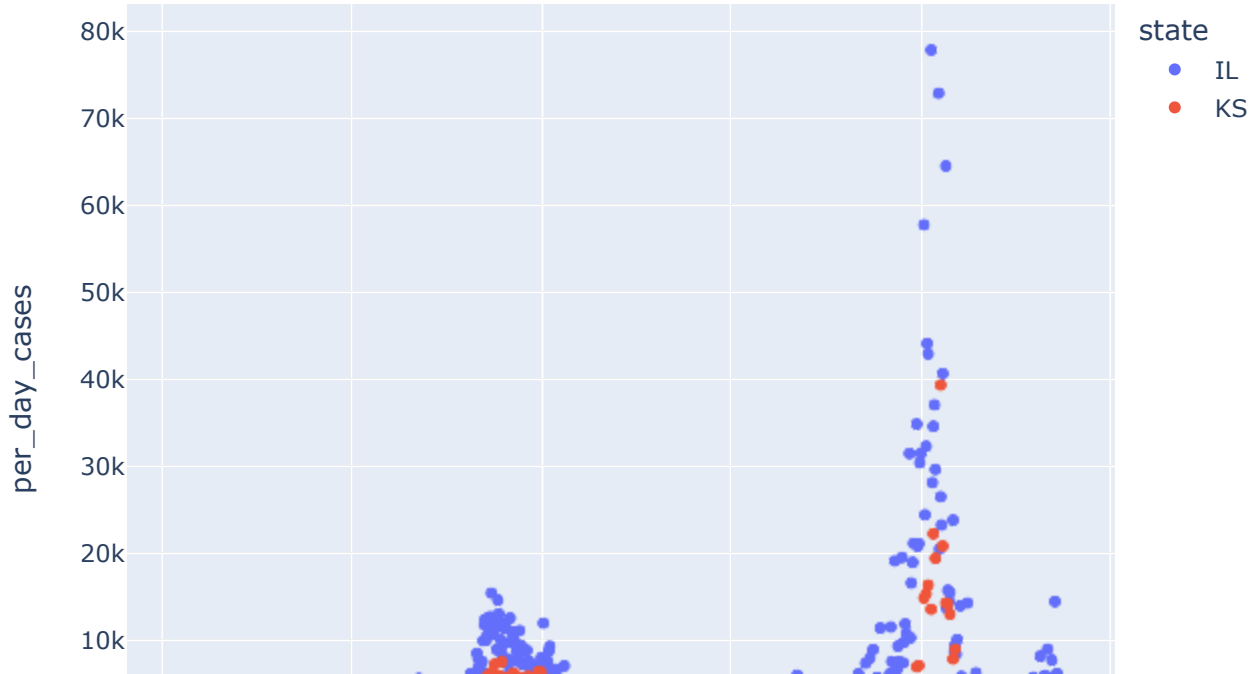
Scatter plot showing number of per day deaths for two states

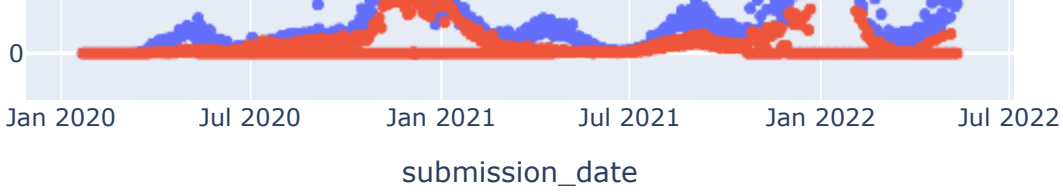
In [17]:
```
pio.templates.default = "plotly"

px.scatter(df_case_final, x="submission_date", y="per_day_deaths", color="state")
```



Scatter plot showing number of per day cases for two states

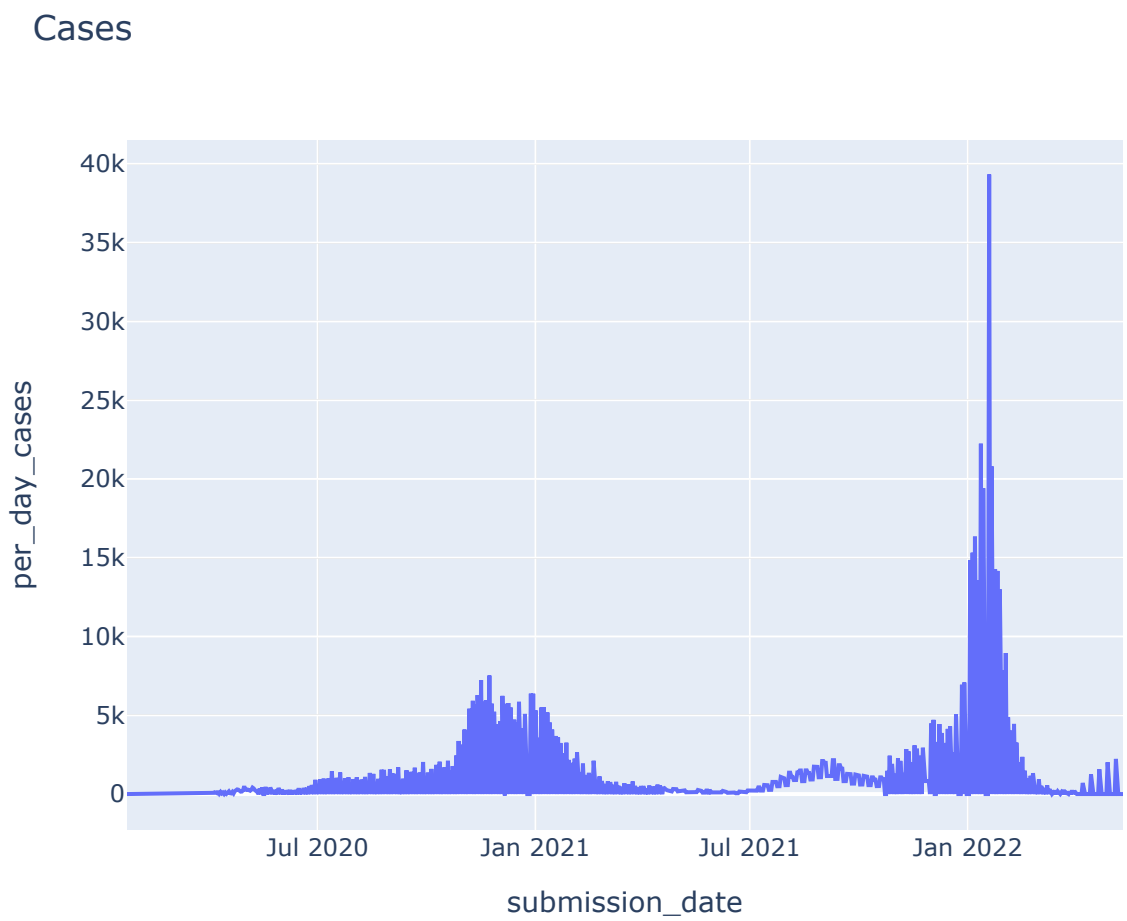
In [18]:
```
px.scatter(df_case_final, x="submission_date", y="per_day_cases", color="state")
```





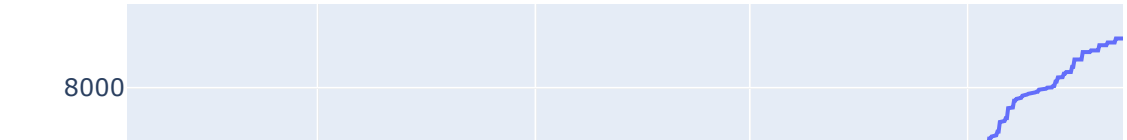
Cases vs date graph below shows the two peaks occuring

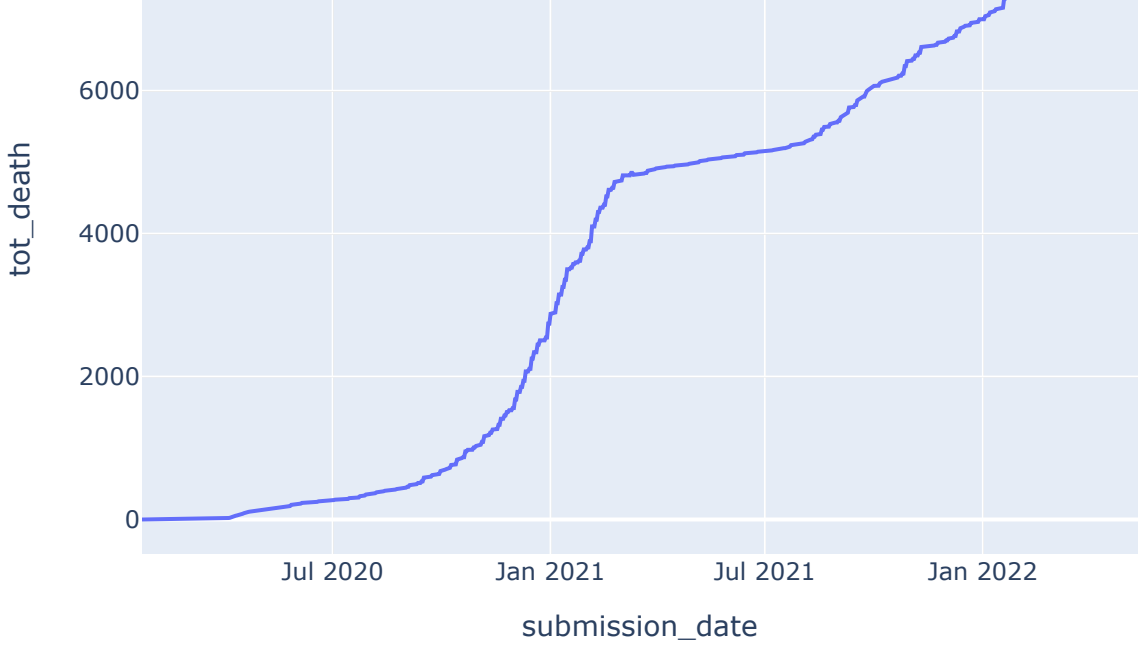
In [19]:
```
px.line(df_case_KS,x="submission_date",y="per_day_cases",title="Cases")
```



Total deaths vs date graph below shows the sudden rise in number of deaths during peak of covid waves

In [20]:
```
px.line(df_case_KS,x="submission_date",y="tot_death",title="World Wide Confirmed Cases ")
```

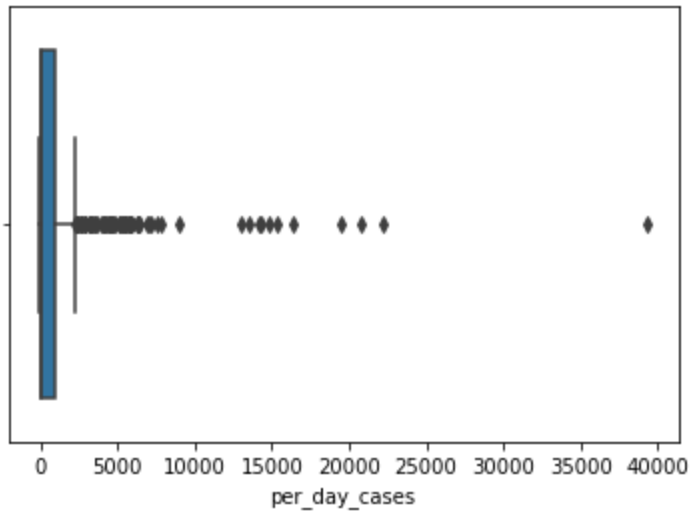




In [21]:
```
df_case_KS.reset_index(drop=True, inplace=True)
df_case_IL.reset_index(drop=True, inplace=True)
```

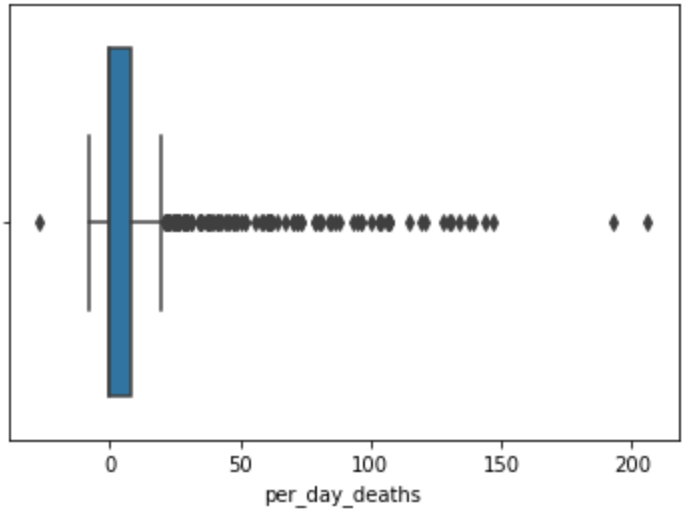
In [22]:
```
import seaborn as sns

ax = sns.boxplot(x=df_case_KS["per_day_cases"])
ax.set_xlabel('per_day_cases')
```

Out[22]:
```
Text(0.5, 0, 'per_day_cases')
```



In [23]:
```
import seaborn as sns

ax = sns.boxplot(x=df_case_KS["per_day_deaths"])
ax.set_xlabel('per_day_deaths')
```

Out[23]:
```
Text(0.5, 0, 'per_day_deaths')
```

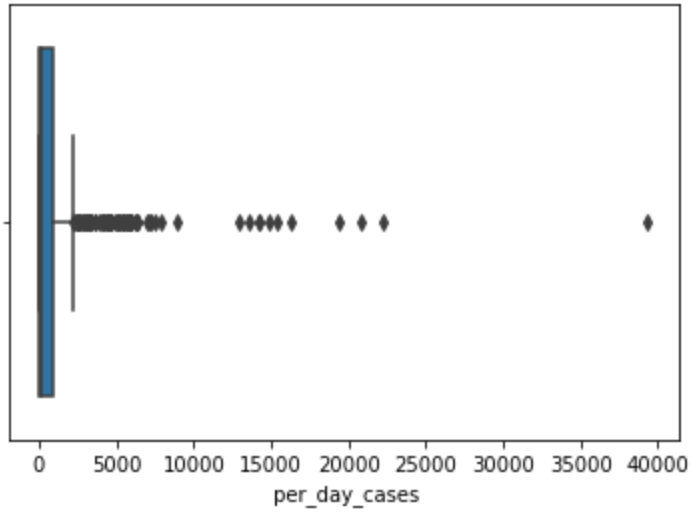


The above box plots shows that there are few data points where per_days_cases and per_day_deaths values are less than 0. Since, they can't be negative we can set these negative values to 0.

In [24]:
```
neg_deaths_KS = df_case_KS[df_case_KS['per_day_deaths']<0].index.values.tolist()
neg_cases_KS = df_case_KS[df_case_KS['per_day_cases']<0].index.values.tolist()
neg_deaths_IL = df_case_IL[df_case_IL['per_day_deaths']<0].index.values.tolist()
neg_cases_IL = df_case_IL[df_case_IL['per_day_cases']<0].index.values.tolist()

if len(neg_cases_KS)>0:
    for i in (neg_cases_KS):
        df_case_KS.at[i, 'per_day_cases'] = 0

if len(neg_deaths_KS)>0:
    for i in (neg_deaths_KS):
        df_case_KS.at[i, 'per_day_deaths'] = 0

if len(neg_cases_IL)>0:
    for i in (neg_cases_IL):
        df_case_IL.at[i, 'per_day_cases'] = 0

if len(neg_deaths_IL)>0:
    for i in (neg_deaths_IL):
        df_case_IL.at[i, 'per_day_deaths'] = 0
```

In [25]:
```
import seaborn as sns

ax = sns.boxplot(x=df_case_KS["per_day_cases"])
ax.set_xlabel(' per_day_cases')
```

Out[25]:
```
Text(0.5, 0, ' per_day_cases')
```

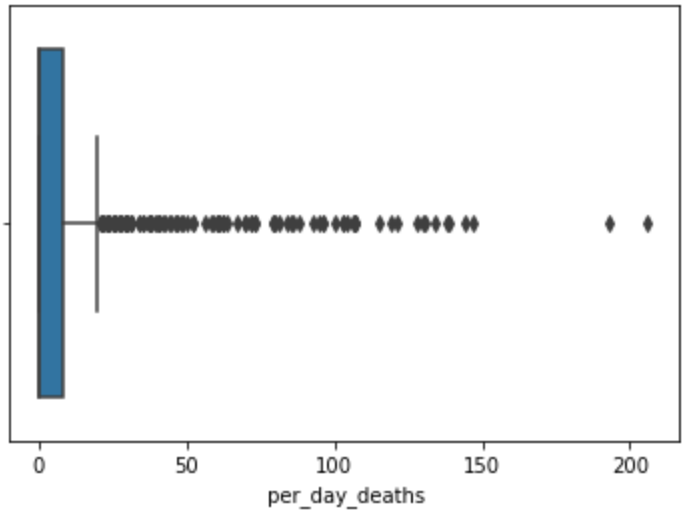


In [26]:

```
import seaborn as sns

ax = sns.boxplot(x=df_case_KS["per_day_deaths"])
ax.set_xlabel(' per_day_deaths')
```

Out[26]:

```
Text(0.5, 0, ' per_day_deaths')
```



In [27]:

```
#Tukey's method
def tukeys_method(df, variable):
    #Takes two parameters: dataframe & variable of interest as string
    q1 = df[variable].quantile(0.25)
    q3 = df[variable].quantile(0.75)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    #inner fence lower and upper end
    inner_fence_le = q1-inner_fence
    inner_fence_ue = q3+inner_fence

    outliers_poss = []

    for index, x in enumerate(df[variable]):
        if x <= inner_fence_le or x >= inner_fence_ue:
            outliers_poss.append(index)
    return  outliers_poss
```

In [28]:

```
possible_outliers_tm = tukeys_method(df_case_KS, "per_day_deaths")
```

In [29]:
```
len(possible_outliers_tm)
```

Out[29]:
```
121
```

As per tukeys rule the number of outliers are 121. But when we observed carefully these are not outliers in real life. These are days when the covid waves were at peak or at low.

Vaccinations

In [30]:
```
df=pd.read_csv("COVID-19_Vaccinations_in_the_United_States_Jurisdiction.csv")
df.head()
```

Out[30]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna	Distributed_Pfizer	Di
0	05/13/2022	19	MA	17188110	619300	6535820	10032990	
1	05/13/2022	19	DE	2398555	100100	929000	1369455	
2	05/13/2022	19	VA	19930285	784600	7104400	12041285	
3	05/13/2022	19	GU	320560	24100	85780	210680	
4	05/13/2022	19	NV	6020510	259200	2068300	3693010	

5 rows × 82 columns

In [33]:
```
df.columns
```

Out[33]:
```
Index(['Date', 'MMWR_week', 'Location', 'Distributed', 'Distributed_Janssen',
       'Distributed_Moderna', 'Distributed_Pfizer', 'Distributed_Unk_Manuf',
       'Dist_Per_100K', 'Distributed_Per_100k_12Plus',
       'Distributed_Per_100k_18Plus', 'Distributed_Per_100k_65Plus',
       'Administered', 'Administered_12Plus', 'Administered_18Plus',
       'Administered_65Plus', 'Administered_Janssen', 'Administered_Moderna',
       'Administered_Pfizer', 'Administered_Unk_Manuf', 'Admin_Per_100K',
       'Admin_Per_100k_12Plus', 'Admin_Per_100k_18Plus',
       'Admin_Per_100k_65Plus', 'Recip_Administered',
       'Administered_Dose1_Recip', 'Administered_Dose1_Pop_Pct',
       'Administered_Dose1_Recip_12Plus',
       'Administered_Dose1_Recip_12PlusPop_Pct',
       'Administered_Dose1_Recip_18Plus',
       'Administered_Dose1_Recip_18PlusPop_Pct',
       'Administered_Dose1_Recip_65Plus',
       'Administered_Dose1_Recip_65PlusPop_Pct', 'Series_Complete_Yes',
       'Series_Complete_Pop_Pct', 'Series_Complete_12Plus',
       'Series_Complete_12PlusPop_Pct', 'Series_Complete_18Plus',
       'Series_Complete_18PlusPop_Pct', 'Series_Complete_65Plus',
       'Series_Complete_65PlusPop_Pct', 'Series_Complete_Janssen',
       'Series_Complete_Moderna', 'Series_Complete_Pfizer',
       'Series_Complete_Unk_Manuf', 'Series_Complete_Janssen_12Plus',
       'Series_Complete_Moderna_12Plus', 'Series_Complete_Pfizer_12Plus',
       'Series_Complete_Unk_Manuf_12Plus', 'Series_Complete_Janssen_18Plus',
       'Series_Complete_Moderna_18Plus', 'Series_Complete_Pfizer_18Plus',
       'Series_Complete_Unk_Manuf_18Plus', 'Series_Complete_Janssen_65Plus',
       'Series_Complete_Moderna_65Plus', 'Series_Complete_Pfizer_65Plus',
       'Series_Complete_Unk_Manuf_65Plus', 'Additional_Doses',
       'Additional_Doses_Vax_Pct', 'Additional_Doses_12Plus',
       'Additional_Doses_12Plus_Vax_Pct', 'Additional_Doses_18Plus',
       'Additional_Doses_18Plus_Vax_Pct', 'Additional_Doses_50Plus',
```

```
       'Additional_Doses_50Plus_Vax_Pct', 'Additional_Doses_65Plus',
       'Additional_Doses_65Plus_Vax_Pct', 'Additional_Doses_Moderna',
       'Additional_Doses_Pfizer', 'Additional_Doses_Janssen',
       'Additional_Doses_Unk_Manuf', 'Administered_Dose1_Recip_5Plus',
       'Administered_Dose1_Recip_5PlusPop_Pct', 'Series_Complete_5Plus',
       'Series_Complete_5PlusPop_Pct', 'Administered_5Plus',
       'Admin_Per_100k_5Plus', 'Distributed_Per_100k_5Plus',
       'Series_Complete_Moderna_5Plus', 'Series_Complete_Pfizer_5Plus',
       'Series_Complete_Janssen_5Plus', 'Series_Complete_Unk_Manuf_5Plus'],
      dtype='object')
```

In [38]:
```
df_vac_final = df[["Date","MMWR_week","Location","Administered","Administered_Janssen","Ad
       'Admin_Per_100k_12Plus', 'Admin_Per_100k_18Plus',
       'Admin_Per_100k_65Plus']]
```

There are no null values in the columns of our interest

In [39]:
```
df_vac_final.isnull().sum()
```

Out[39]:
```
Date                      0
MMWR_week                 0
Location                  0
Administered              0
Administered_Janssen      0
Administered_Moderna      0
Administered_Pfizer       0
Administered_Unk_Manuf    0
Administered_12Plus       0
Administered_18Plus       0
Administered_65Plus       0
Admin_Per_100K            0
Admin_Per_100k_12Plus     0
Admin_Per_100k_18Plus     0
Admin_Per_100k_65Plus     0
dtype: int64
```

In [40]:
```
df_vac_final = df_vac_final.loc[df_vac_final['Location'].isin(['KS', 'IL'])]
df_vac_final['Date']= pd.to_datetime(df_vac_final['Date'])
df_vac_final = df_vac_final.sort_values(['Location', 'Date'])
```

In [41]:
```
df_vac_final.reset_index(drop=True, inplace=True)
```

In [42]:
```
df_vac_final
```

Out[42]:

	Date	MMWR_week	Location	Administered	Administered_Janssen	Administered_Moderna	Administered_Pfiz
0	2020-12-14	51	IL	0	0	0	
1	2020-12-15	51	IL	0	0	0	
2	2020-12-16	51	IL	0	0	0	
3	2020-12-17	51	IL	0	0	0	
4	2020-12-18	51	IL	2076	0	0	20

	Date	MMWR_week	Location	Administered	Administered_Janssen	Administered_Moderna	Administered_Pfiz
...	...	...	...	...	...	...	
1027	2022-05-09	19	KS	4629885	138729	1758934	27278
1028	2022-05-10	19	KS	4632275	138737	1759634	27295
1029	2022-05-11	19	KS	4634977	138752	1760710	27311
1030	2022-05-12	19	KS	4638623	138797	1762047	27334
1031	2022-05-13	19	KS	4642352	138824	1763462	27357

1032 rows × 15 columns

In [43]:
```
df_vac_final['per_day_Administered'] = df_vac_final['Administered'].diff()
df_vac_IL = df_vac_final.loc[df_vac_final['Location'].isin(['IL'])]
df_vac_KS = df_vac_final.loc[df_vac_final['Location'].isin(['KS'])]
```

2a. Using one sample 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. Also, running the two sample version of Wald's and two-sample unpaired t-test.

Functions for Wald's test

In [73]:
```
def get_corrected_variance(data, mean):
    return float(sum([(x-mean)**2 for x in data])) / (len(data)-1)

def get_uncorrected_variance(data, mean):
    return float(sum([(x-mean)**2 for x in data])) / len(data)

def waldstest(X, true_mean, var_x):
    n_X = len(X)
#     uncorrected_variance_X = get_uncorrected_variance(X, np.mean(X))
#     return (np.mean(X) - true_mean) / math.sqrt(uncorrected_variance_X/ n_X)
    w = abs((np.mean(X) - true_mean) / math.sqrt(var_x/ n_X))
    if w>1.96:
        print("REJECT THE HYPOTHESIS")
    else:
        print("ACCEPT THE HYPOTHESIS")
    return w

def waldstest_2sample(X, Y):
    n_X = len(X)
    n_Y = len(Y)
    uncorrected_variance_X = get_uncorrected_variance(X, np.mean(X))
    uncorrected_variance_Y = get_uncorrected_variance(Y, np.mean(Y))
#     print(uncorrected_variance_X, uncorrected_variance_Y)
    w = abs((np.mean(X) - np.mean(Y)) / math.sqrt((uncorrected_variance_X / n_X) + (uncorr
    if w>1.96:
        print("REJECT THE HYPOTHESIS")
    else:
        print("ACCEPT THE HYPOTHESIS")
    return w
```

Function for Z-test

In [74]:
```
def custom_z_test(X, Y, var_x, var_y):
    X_mean = np.mean(X)
    Y_mean = np.mean(Y)
    n = len(X)
    m = len(Y)
    Z = abs((X_mean - Y_mean) / math.sqrt( (var_x / n) + (var_y / m)))
    p = 2 * (st.norm.sf(abs(Z)))
    if Z>1.96:
        print("REJECT THE HYPOTHESIS")
    else:
        print("ACCEPT THE HYPOTHESIS")
    return Z, p
```

Functions for t-test

In [75]:
```
def t_test_1sample(x, true_mean):
    n = len(x)
    x_mean = np.mean(x)
    std = 0
    for ele in x:
        std += (ele - x_mean) ** 2
    std = np.sqrt(std / n)
    t = abs((x_mean - true_mean) / (std / np.sqrt(n)))
    if t>3.46:
        print("REJECT THE HYPOTHESIS")
    else:
        print("ACCEPT THE HYPOTHESIS")
    return t

def t_test_2sample(x, y):
    n = len(x)
    x_mean = np.mean(x)
    m = len(y)
    y_mean = np.mean(y)
    var_x = 0
    var_y = 0
    for ele in x:
        var_x += (ele - x_mean) ** 2
    for ele in y:
        var_y += (ele - y_mean) ** 2
    var_x = var_x/(n-1)
    var_y = var_y/(m-1)
    t = abs((x_mean - y_mean) / (math.sqrt( (var_x/n) + (var_y/m))))
    if t>3.46:
        print("REJECT THE HYPOTHESIS")
    else:
        print("ACCEPT THE HYPOTHESIS")
    return t
```

In [76]:
```
# Initialize seperate dataframes for each state
startDateFeb = '2021-02-01'
endDateFeb = '2021-03-01'
startDateMarch = '2021-03-01'
endDateMarch = '2021-04-01'

df_case_IL_Feb = df_case_IL[(df_case_IL['submission_date'] >= pd.to_datetime(startDateFeb)
                                    & (df_case_IL['submission_date'] < pd.to_datetime(endDa
df_case_IL_March = df_case_IL[(df_case_IL['submission_date'] >= pd.to_datetime(startDateMa
```

```
                                    & (df_case_IL['submission_date'] < pd.to_datetime(endDa
df_case_IL_Feb.reset_index(drop=True, inplace=True)
df_case_IL_March.reset_index(drop=True, inplace=True)

df_case_KS_Feb = df_case_KS[(df_case_KS['submission_date'] >= pd.to_datetime(startDateFeb)
                                    & (df_case_KS['submission_date'] < pd.to_datetime(endDa
df_case_KS_March = df_case_KS[(df_case_KS['submission_date'] >= pd.to_datetime(startDateMa
                                    & (df_case_KS['submission_date'] < pd.to_datetime(endDa
df_case_KS_Feb.reset_index(drop=True, inplace=True)
df_case_KS_March.reset_index(drop=True, inplace=True)
```

One Sample tests

In [87]:

```
# given poisson, so the theta_cap will be lambda which is both mean and var.
print("One-sample Wald's test for IL state for daily cases:", waldstest(df_case_IL_March['
                df_case_IL_Feb['per_day_cases'].mean(), df_case_IL_Feb['per_day_cases'].me
print("One-sample Wald's test for IL state for daily deaths:", waldstest(df_case_IL_March
                df_case_IL_Feb['per_day_deaths'].mean(), df_case_IL_Feb['per_day_deaths'].
```

```
REJECT THE HYPOTHESIS
One-sample Wald's test for IL state for daily cases: 34.71552342083148
REJECT THE HYPOTHESIS


One-sample Wald's test for IL state for daily deaths: 19.670509356669946
```

In [78]:

```
print("One-sample Wald's test for KS state for daily cases:", waldstest(df_case_KS_March['
                df_case_KS_Feb['per_day_cases'].mean(), df_case_KS_Feb['per_day_cases'].me
print("One-sample Wald's test for KS state for daily deaths:", waldstest(df_case_KS_March
                df_case_KS_Feb['per_day_deaths'].mean(), df_case_KS_Feb['per_day_deaths'].
```

```
REJECT THE HYPOTHESIS
One-sample Wald's test for KS state for daily cases: 84.87138281749313
REJECT THE HYPOTHESIS
One-sample Wald's test for KS state for daily deaths: 26.10935164742869
```

In [79]:

```
df_case_IL_var = df_case_IL.var()

print("Z-test for IL state for daily cases:", custom_z_test(df_case_IL_Feb['per_day_cases'
                                                            df_case_IL_March['per_day_case
                                                            df_case_IL_var['per_day_cases'
                                                            df_case_IL_var['per_day_cases'


print("Z-test for IL state for daily deaths:", custom_z_test(df_case_IL_Feb['per_day_death
                                                            df_case_IL_March['per_day_deat
                                                            df_case_IL_var['per_day_deaths
                                                            df_case_IL_var['per_day_deaths
```

```
ACCEPT THE HYPOTHESIS
Z-test for IL state for daily cases: (0.1601505639427598, 0.8727624708992866)
ACCEPT THE HYPOTHESIS
Z-test for IL state for daily deaths: (1.8199473453975115, 0.06876702386362198)
```

In [80]:

```
df_case_KS_var = df_case_KS.var()

print("Z-test for KS state for daily cases:", custom_z_test(df_case_KS_Feb['per_day_cases'
                                                            df_case_KS_March['per_day_case
                                                            df_case_KS_var['per_day_cases'
                                                            df_case_KS_var['per_day_cases'


print("Z-test for KS state for daily deaths:", custom_z_test(df_case_KS_Feb['per_day_death
```

```
                                                    df_case_KS_March['per_day_deat
                                                    df_case_KS_var['per_day_deaths
                                                    df_case_KS_var['per_day_deaths
```

```
ACCEPT THE HYPOTHESIS
Z-test for KS state for daily cases: (0.5953769838509666, 0.5515915069584897)
REJECT THE HYPOTHESIS
Z-test for KS state for daily deaths: (4.193710927815825, 2.744276588145579e-05)
```

In [81]:
```
print("One-sample t-test for IL state for daily cases:", t_test_1sample(df_case_IL_March['
                                                                        df_case_IL_Feb['pe
print("One-sample t-test for IL state for daily deaths:", t_test_1sample(df_case_IL_March
                                                                         df_case_IL_Feb['p
```

```
ACCEPT THE HYPOTHESIS
One-sample t-test for IL state for daily cases: 2.9839239857535707
REJECT THE HYPOTHESIS
One-sample t-test for IL state for daily deaths: 10.393174032072313
```

In [82]:
```
print("One-sample t-test for KS state for daily cases:", t_test_1sample(df_case_KS_March['
                                                                        df_case_KS_Feb['pe
print("One-sample t-test for KS state for daily deaths:", t_test_1sample(df_case_KS_March
                                                                         df_case_KS_Feb['p
```

```
REJECT THE HYPOTHESIS
One-sample t-test for KS state for daily cases: 6.897203790195107
REJECT THE HYPOTHESIS
One-sample t-test for KS state for daily deaths: 10.444722062112108
```

Two Sample tests

In [83]:
```
print("Two-sample Wald's test for IL state for daily cases:", waldstest_2sample(df_case_IL
                                                                                 df_case_IL
print("Two-sample Wald's test for IL state for daily deaths:", waldstest_2sample(df_case_I
                                                                                 df_case_IL
```

```
ACCEPT THE HYPOTHESIS
Two-sample Wald's test for IL state for daily cases: 1.8725449108939811
REJECT THE HYPOTHESIS
Two-sample Wald's test for IL state for daily deaths: 4.890363996055578
```

In [84]:
```
print("Two-sample Wald's test for KS state for daily cases:", waldstest_2sample(df_case_KS
                                                                                 df_case_KS
print("Two-sample Wald's test for KS state for daily deaths:", waldstest_2sample(df_case_K
                                                                                 df_case_KS
```

```
REJECT THE HYPOTHESIS
Two-sample Wald's test for KS state for daily cases: 2.2926348911519003
REJECT THE HYPOTHESIS
Two-sample Wald's test for KS state for daily deaths: 2.732761386049286
```

In [85]:
```
print("Two-sample t-test for IL state for daily cases:", t_test_2sample(df_case_IL_Feb['pe
                                                                        df_case_IL_March['
print("Two-sample t-test for IL state for daily deaths:", t_test_2sample(df_case_IL_Feb['p
                                                                         df_case_IL_March
```

```
ACCEPT THE HYPOTHESIS
Two-sample t-test for IL state for daily cases: 1.8400970766123204
REJECT THE HYPOTHESIS
Two-sample t-test for IL state for daily deaths: 4.804141875645057
```

In [86]:
```
print("Two-sample t-test for KS state for daily cases:", t_test_2sample(df_case_KS_Feb['pe
                                                                        df_case_KS_March['
print("Two-sample t-test for KS state for daily deaths:", t_test_2sample(df_case_KS_Feb['p
                                                                        df_case_KS_March
```

```
ACCEPT THE HYPOTHESIS
Two-sample t-test for KS state for daily cases: 2.251767096878278
ACCEPT THE HYPOTHESIS
Two-sample t-test for KS state for daily deaths: 2.6838465085273784
```

In [ ]:
```
```

2b. Inference the equality of distributions between states KS, IL (distributions of daily cases and deaths) for the last three months of 2021 (Oct, Nov, Dec) using K-S test (1-sample and 2-sample tests) and Permutation test.

Functions for KS test

In [88]:
```
def plot(a, label, min_x = 0, max_x = 10):
    n = len(a)
    Srt = sorted(a)
    X = [min_x]
    Y = [0]
    cdf = [0.0]
    for i in range(0, n):
        X = X + [Srt[i], Srt[i]]
        Y = Y + [Y[len(Y)-1], Y[len(Y)-1]+(1/n)]
        cdf = cdf + [Y[len(Y)-1]]
    X = X + [max_x]
    Y = Y + [1.0]

    plt.plot(X,Y, label=label)
    plt.xlabel('x')
    plt.ylabel('Pr[X<=x]')
    plt.legend(loc='best')
    return cdf

def get_cdf(X):
    Fx = [0]
    for i in range(0, len(X)):
        Fx = Fx + [Fx[len(Fx)-1] + 1/len(X)]
    return Fx

def find_cdf_at(X, CDF, change_point):
    # First find the first element larger than the change_point
    index = -1
    for i, x in enumerate(X):
        if x >= change_point:
            index = i
            break
    # Return the CDF value at that point
    return CDF[index]

def ks_test_2_sample(X, Y, week, ho, threshold = 0.05):
    X = sorted(X)
    Y = sorted(Y)

    x_min = min(X[0], Y[0])
    x_max = max(X[len(X) - 1], Y[len(Y) - 1])
    temp = (x_max - x_min)/10
    x_min = x_min - temp
```

```
    x_max = x_max + temp
    fig= plt.figure(figsize=(12,9))
    plt.grid(True)

    x_cdf = plot(X, 'week ' + str(week), x_min, x_max)
    y_cdf = plot(Y, 'week ' + str(week+1), x_min, x_max)

    Fx = [find_cdf_at(X, x_cdf, change_point) for change_point in Y]
    Fy_minus = y_cdf[0:-1]
    Fy_plus = y_cdf[1:]

    max_val = 0
    max_index = 0
    left = True
    for i in range(0, len(Fx)):
        if abs(Fx[i] - Fy_minus[i]) > max_val:
            max_val = abs(Fx[i] - Fy_minus[i])
            max_index = i
            left = True
        if abs(Fx[i] - Fy_plus[i]) > max_val:
            max_val = abs(Fx[i] - Fy_plus[i])
            max_index = i
            left = False

    delta = -0.01
    ymin = 0
    ymax = 0
    if left == False:
        delta = delta * -1
        # Also need to find the limits for the vertical line
        ymin = min(Fy_plus[max_index], Fx[max_index])
    else:
        ymin = min(Fy_minus[max_index], Fx[max_index])

    if max_val > threshold:
        print("D > C, We reject Ho:", ho)

    # plt.axvline(x=Y[max_index], ymax=ymin+max_val, ymin = ymin)
    plt.plot([Y[max_index],Y[max_index]],[ymin,ymin+max_val])
    annotation_str = "Max Diff=" , max_val
    plt.annotate(annotation_str, xy = [Y[max_index], ymin+max_val/2])

    return

def ks_test_1_sample(pdf, X, Y, ho, state, threshold = 0.05):
    fig= plt.figure(figsize=(12,9))
    plt.grid(True)

    plt.plot(pdf, X, label = state)
    plt.plot(pdf, Y, label = 'Poisson dist')
    plt.xlabel('x')
    plt.ylabel('Pr[X<=x]')
    plt.legend(loc='best')
    X_minus = X[0:-1]
    X_plus = X[1:]
    max_val = 0
    max_val_ind = 0
    left = True

    for i in range(1, len(Y) - 1):
        if abs(Y[i] - X_minus[i]) > max_val:
            max_val = abs(Y[i] - X_minus[i])
            max_val_ind = i
            left = True

        if abs(Y[i] - X_plus[i]) > max_val:
```

```
            max_val = abs(Y[i] - X_plus[i])
            max_val_ind = i
            left = False

    if max_val > threshold:
        print("Max value = {0} > C, We reject Ho: {1}".format(max_val, ho))

    ymin = 0
    ymax = 0
    if left == False:
        # Also need to find the limits for the vertical line
        ymin = min(X_plus[max_val_ind], Y[max_val_ind])
    else:
        ymin = min(X_minus[max_val_ind], Y[max_val_ind])
    plt.plot([pdf[max_val_ind], pdf[max_val_ind]], [ymin, ymin + max_val])
    annotation_str = "Max Diff=" , max_val
    plt.annotate(annotation_str, xy = [Y[max_val_ind], ymin + max_val/2])

    return
```

Function for permutation test

In [89]:

```
def permutation_test(X, Y, n, threshold):
    T_obs = abs(np.mean(X) - np.mean(Y))
#     print(T_obs, np.mean(X), np.mean(Y))
    xy = np.append(X,Y)
#     xy.info()
    p_value = 0.0
    for i in range(n):
        permutation = np.random.permutation(xy)
        X1 = permutation[:len(X)]
        Y1 = permutation[len(X):]
        Ti = abs(np.mean(X1) - np.mean(Y1))
        if(Ti > T_obs):
            p_value += 1.0
#         print(p_value, T_obs, Ti)
    p_value = p_value/n
    print("The p-value is: ", p_value)
    if(p_value <= threshold):
        print("==> Reject the Null Hypothesis")
    else:
        print("==> Accept the Null Hypothesis")
    return
```

In [90]:

```
# Initialize seperate dataframes for each state for the months of Oct, Nov, Dec
startDateOct = '2021-10-01'
endDateDec = '2021-12-31'

df_case_IL_OctDec = df_case_IL[(df_case_IL['submission_date'] >= pd.to_datetime(startDateO
                                & (df_case_IL['submission_date'] <= pd.to_datetime(endD
df_case_IL_OctDec = df_case_IL_OctDec.reset_index()

df_case_KS_OctDec = df_case_KS[(df_case_KS['submission_date'] >= pd.to_datetime(startDateO
                                & (df_case_KS['submission_date'] <= pd.to_datetime(endD
df_case_KS_OctDec = df_case_KS_OctDec.reset_index()
```

Permutation test : Hypotheses and Results

In [92]:

```
print("H0: For Oct'21 to Dec'21, the distribution of #deaths due to COVID is similar in IL
permutation_test(np.array(df_case_IL_OctDec['per_day_deaths']), np.array(df_case_KS_OctDec
```

```
print("-------------------------------------------------------------------------------
print("H0: For Oct'21 to Dec'21, the distribution of #cases due to COVID is similar in IL
permutation_test(np.array(df_case_IL_OctDec['per_day_cases']), np.array(df_case_KS_OctDec
```

```
H0: For Oct'21 to Dec'21, the distribution of #deaths due to COVID is similar in IL and KS
The p-value is:  0.0
==> Reject the Null Hypothesis
-------------------------------------------------------------------------------------------
-----------
H0: For Oct'21 to Dec'21, the distribution of #cases due to COVID is similar in IL and KS
The p-value is:  0.0
==> Reject the Null Hypothesis
```

1 sample KS test : Poisson

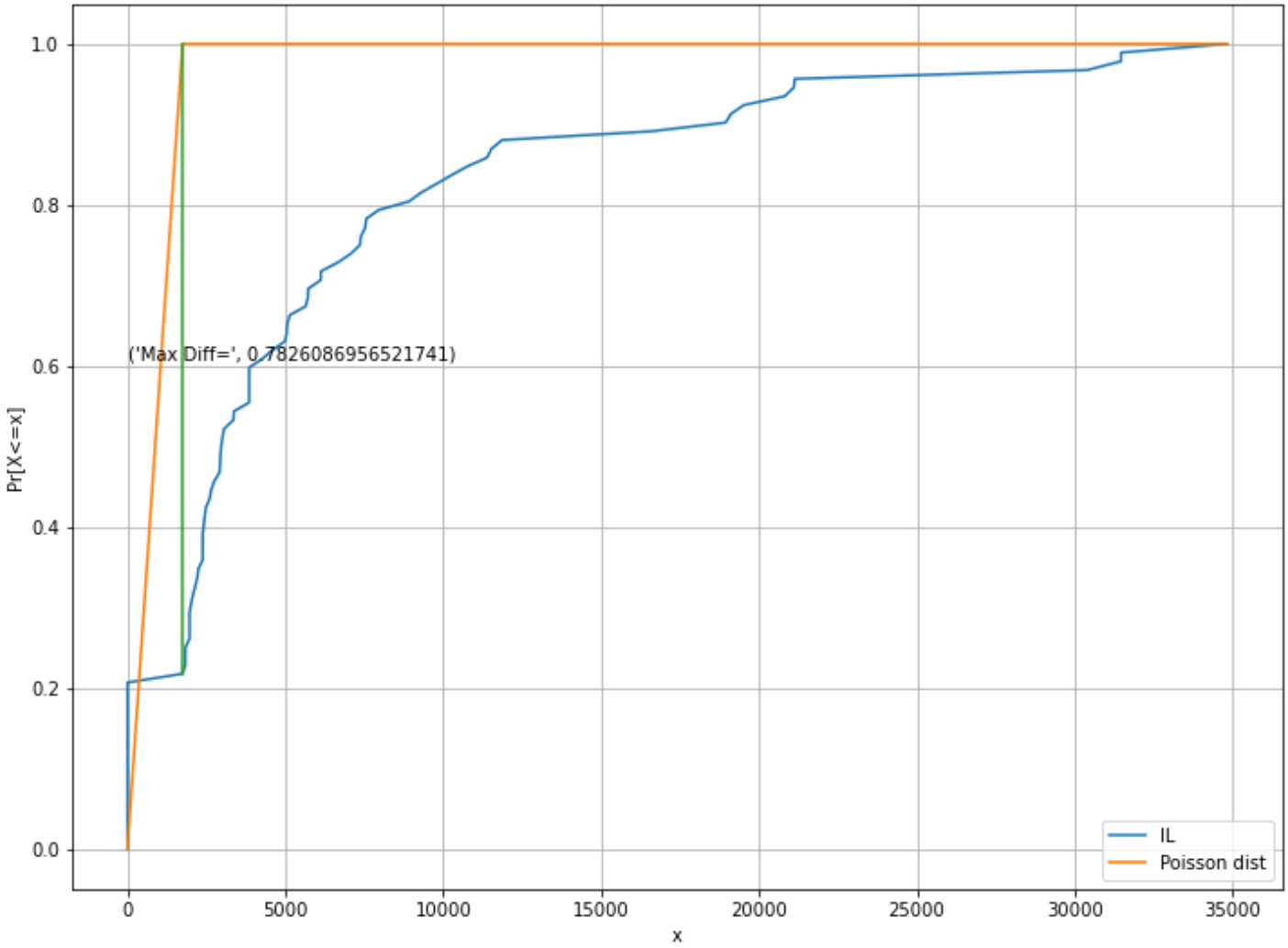
In [93]:

```
IL_OctDec_cases = sorted(df_case_IL_OctDec['per_day_cases'])
IL_OctDec_deaths = sorted(df_case_IL_OctDec['per_day_deaths'])

IL_OctDec_cases_cdf = get_cdf(IL_OctDec_cases)
IL_OctDec_deaths_cdf = get_cdf(IL_OctDec_deaths)
```

In [94]:

```
# First lets sample mean for the second last week
lambda_cases_mme = np.mean(df_case_KS_OctDec['per_day_cases'])

# First get all the cdf values for Poisson distribution
cases_poisson_cdf = [0] + [poisson.cdf(i, lambda_cases_mme) for i in IL_OctDec_cases]
ks_test_1_sample([0] + IL_OctDec_cases, IL_OctDec_cases_cdf, cases_poisson_cdf,
                 "The cases in IL follow Poission distribution of KS", 'IL')
```

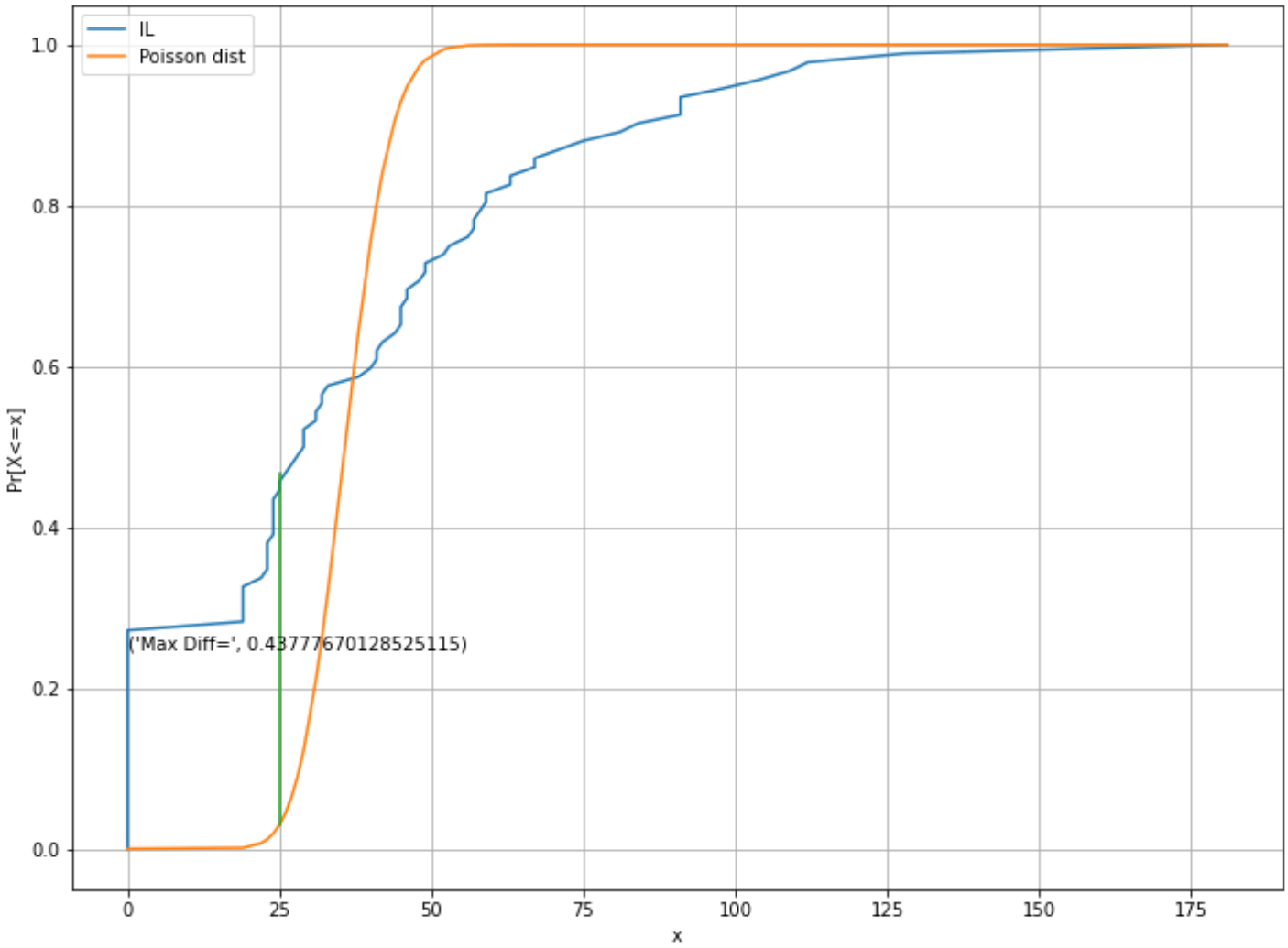
```
Max value = 0.7826086956521741 > C, We reject Ho: The cases in IL follow Poission distribu
tion of KS
```



In [95]:

```
lambda_deaths_mme = np.mean(df_case_IL_OctDec['per_day_deaths'])

deaths_poisson_cdf = [0] + [poisson.cdf(i, lambda_deaths_mme) for i in IL_OctDec_deaths]
ks_test_1_sample([0] + IL_OctDec_deaths, IL_OctDec_deaths_cdf, deaths_poisson_cdf, "The de
```

```
Max value = 0.43777670128525115 > C, We reject Ho: The deaths in IL follow Poission distri
bution of KS
```

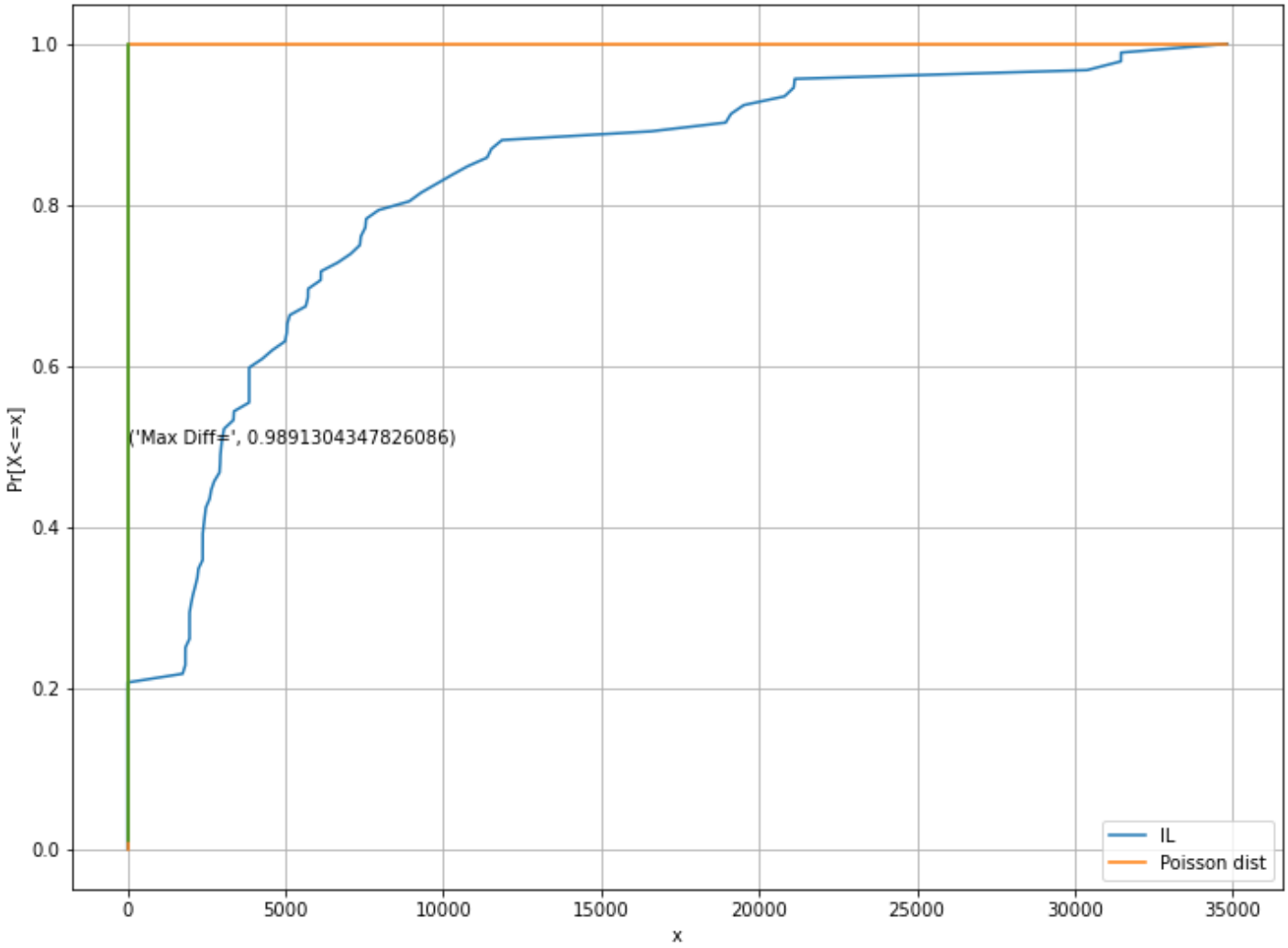


1 sample KS test : Binomial

In [96]:

```
# First perform the experiment for number of new cases
KS_cases_var = np.var(df_case_KS_OctDec['per_day_cases'])

n_binom_mme = (lambda_cases_mme**2) /(lambda_cases_mme - KS_cases_var)
p_binom_mme = 1 - KS_cases_var/lambda_cases_mme
print(n_binom_mme, p_binom_mme)
cases_binom_cdf = [0] + [binom.cdf(i, n_binom_mme, p_binom_mme) for i in IL_OctDec_cases]
ks_test_1_sample([0] + IL_OctDec_cases, IL_OctDec_cases_cdf, cases_binom_cdf, "The cases i
print(binom.cdf(0, 1, 1))
print(-2127.6316259787245 ** (0.5590314737548249))
```

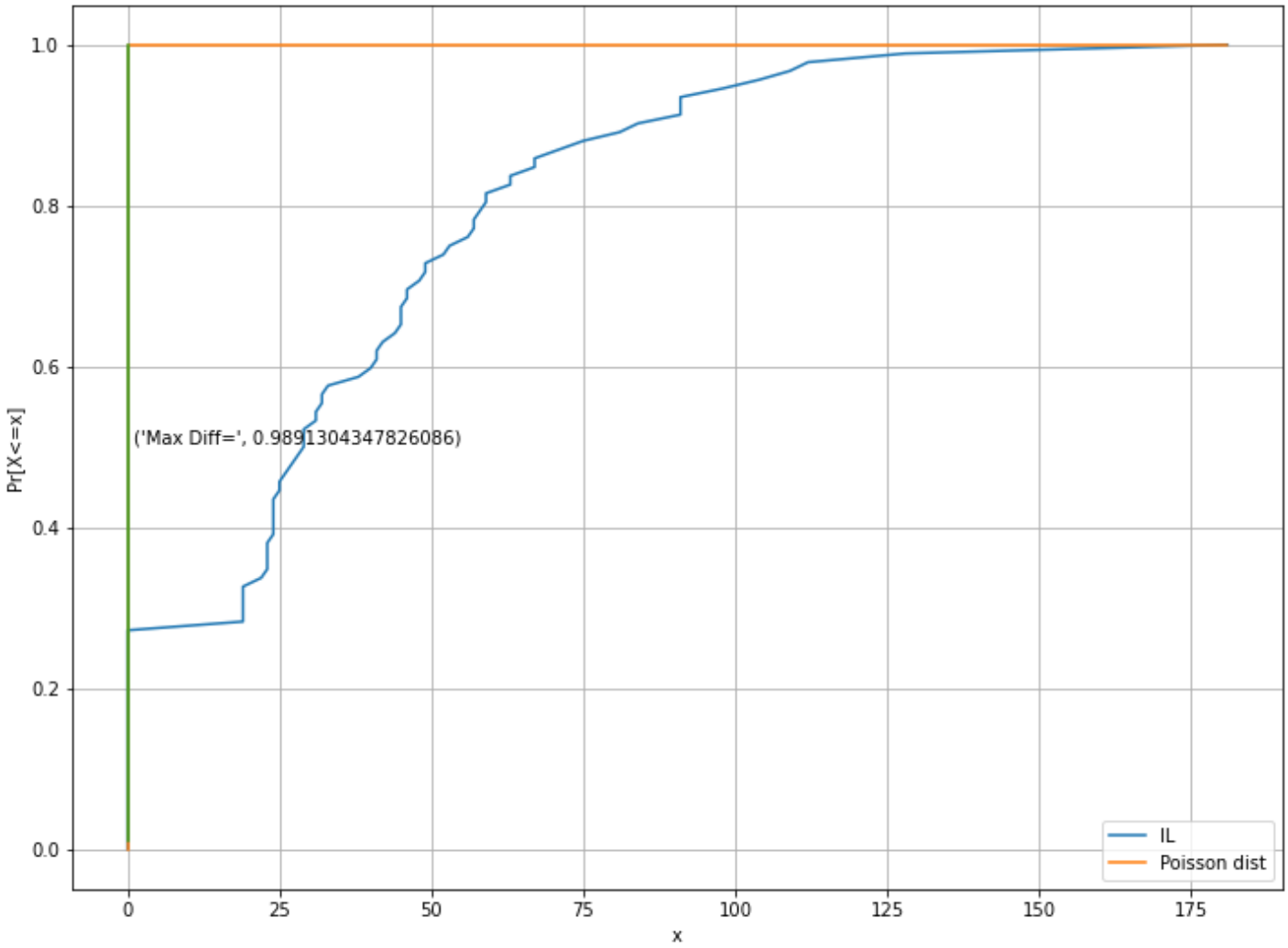
```
-0.5590314737548251 -2127.631625978724
Max value = 0.9891304347826086 > C, We reject Ho: The cases in last week follow Binomial d
istribution
0.0
-72.51015772409802
```



In [97]:

```
# Now, perform the same experiment for the number of deaths
KS_deaths_var = np.var(df_case_KS_OctDec['per_day_deaths'])

n_binom_mme = lambda_deaths_mme*lambda_deaths_mme/(lambda_deaths_mme - KS_deaths_var)
p_binom_mme = 1 - KS_deaths_var/lambda_deaths_mme

deaths_binom_cdf = [0] + [binom.cdf(i, n_binom_mme, p_binom_mme) for i in IL_OctDec_deaths
ks_test_1_sample([0] + IL_OctDec_deaths, IL_OctDec_deaths_cdf, deaths_binom_cdf, "The deat
```

```
Max value = 0.9891304347826086 > C, We reject Ho: The deaths in last week follow Binomial
distrubtion
```

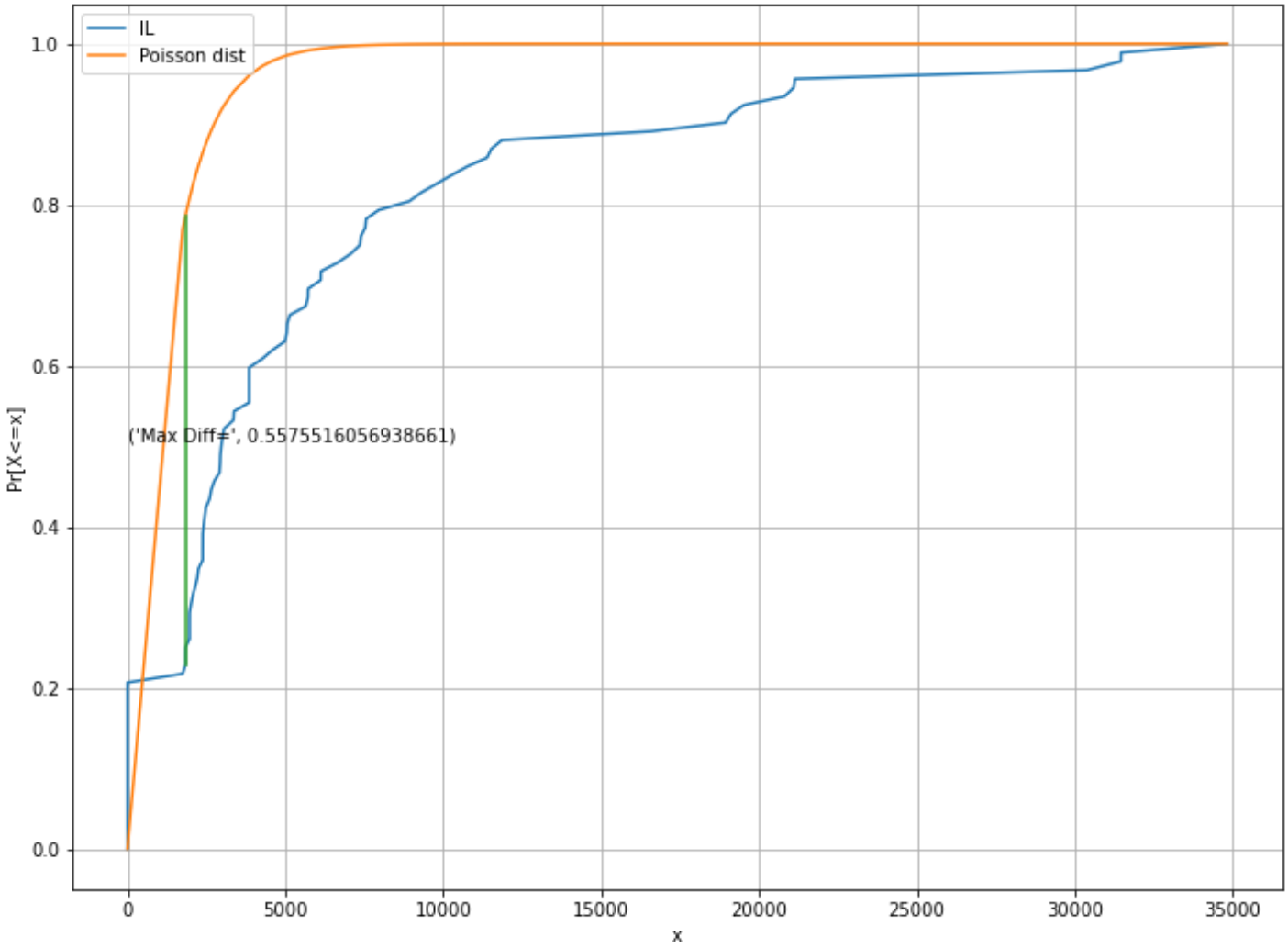


1 sample KS test : Geometric

In [98]:

```
# First perform the experiment for number of new cases
p_geom_mme = 1/lambda_cases_mme
cases_geom_cdf = [0] + [geom.cdf(i, p_geom_mme) for i in IL_OctDec_cases]
ks_test_1_sample([0] + IL_OctDec_cases, IL_OctDec_cases_cdf, cases_geom_cdf, "The cases ir
```

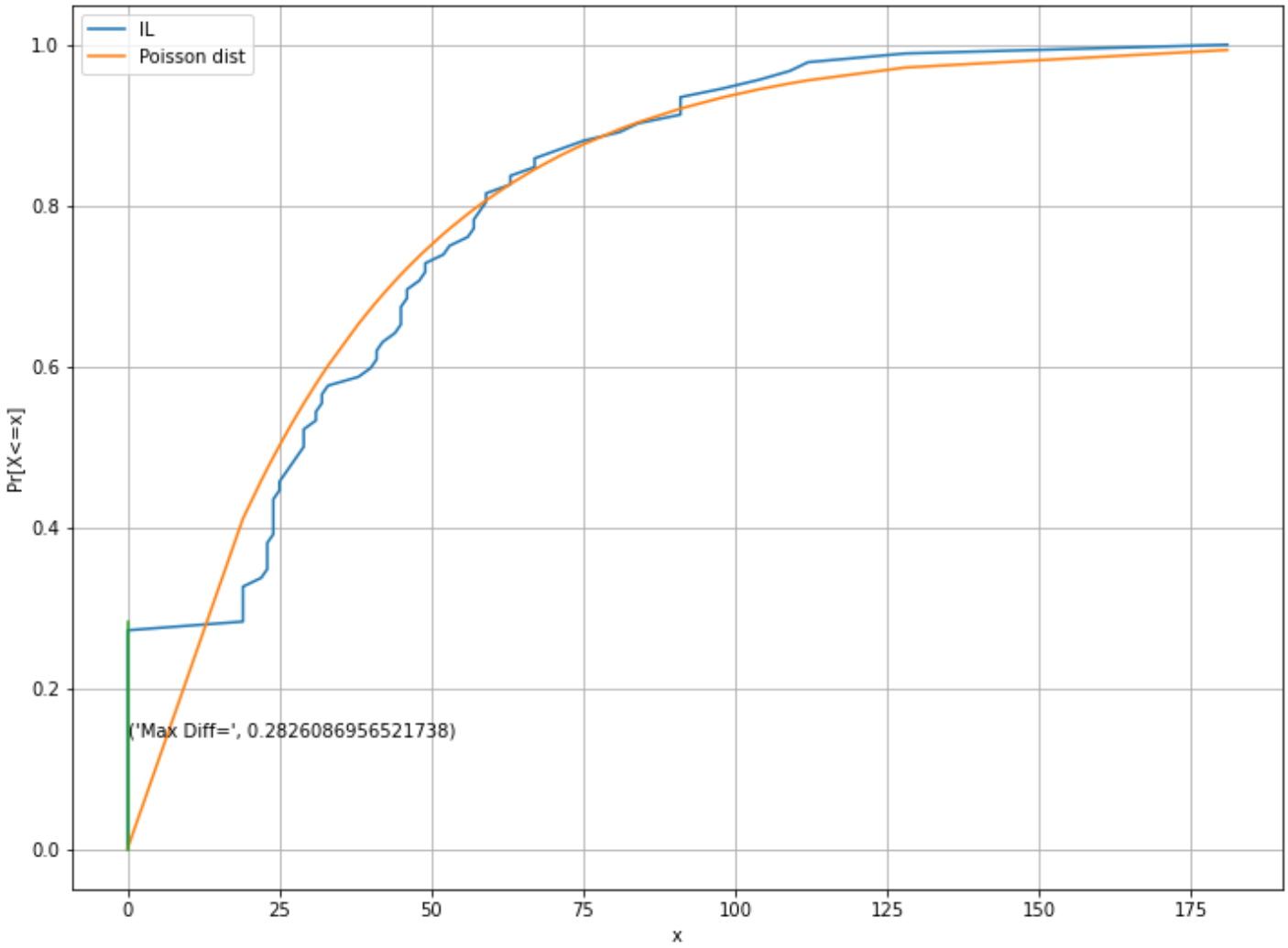
```
Max value = 0.5575516056938661 > C, We reject Ho: The cases in last week follow Geometric
distribution
```



In [99]:

```
# Now, perform the same experiment for the number of deathsp_geom_mme = 1/X_bar_cases
p_geom_mme = 1/lambda_deaths_mme
deaths_genom_cdf = [0] + [geom.cdf(i, p_geom_mme) for i in IL_OctDec_deaths]
ks_test_1_sample([0] + IL_OctDec_deaths, IL_OctDec_deaths_cdf, deaths_genom_cdf, "The deat
```

```
Max value = 0.2826086956521738 > C, We reject Ho: The deaths in last week follow Geometric
distribution
```

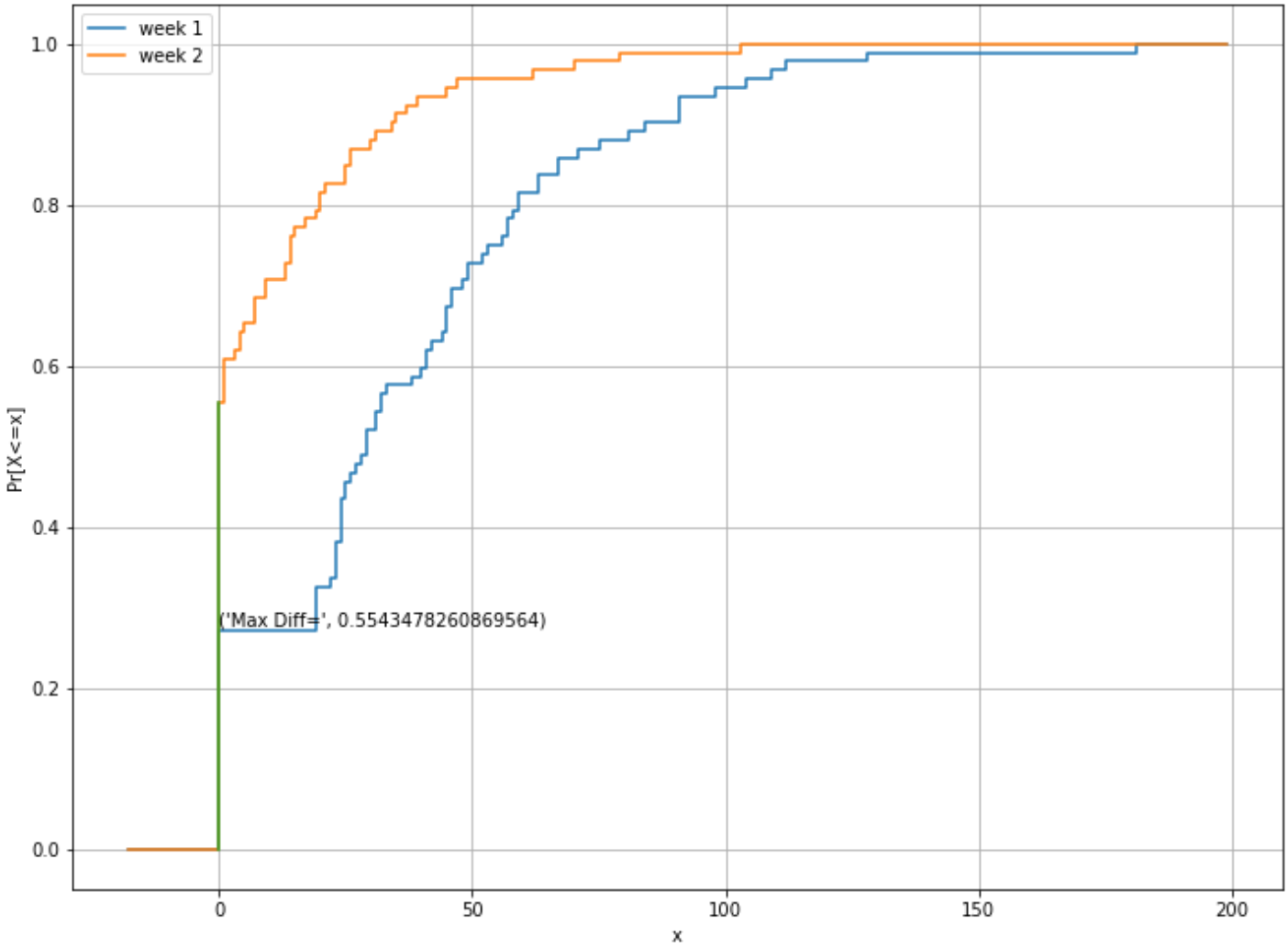


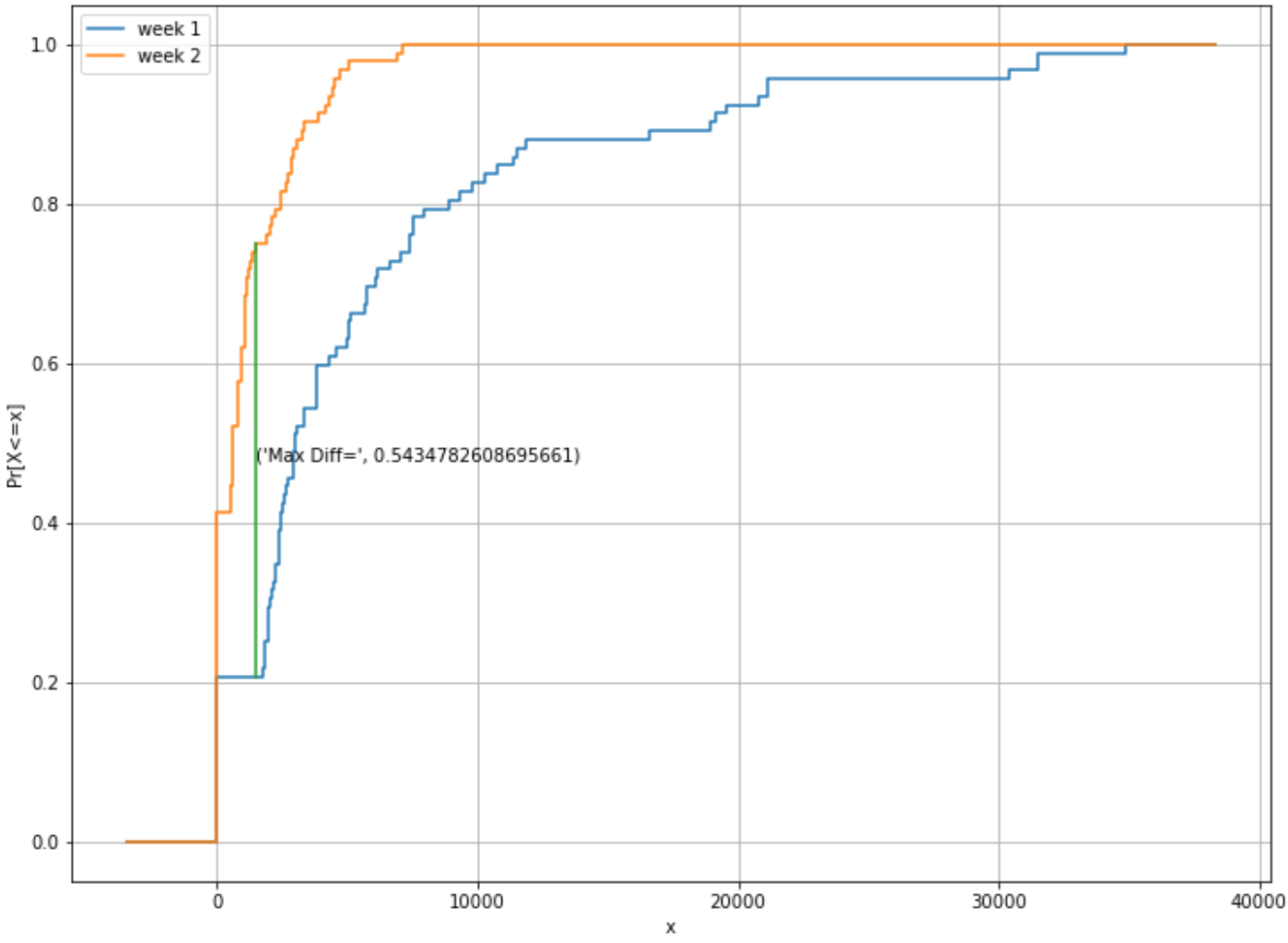
2 sample KS test

In [100…

```
ks_test_2_sample(np.array(df_case_IL_OctDec['per_day_deaths']),
                 np.array(df_case_KS_OctDec['per_day_deaths']), 1, "A")
ks_test_2_sample(np.array(df_case_IL_OctDec['per_day_cases']),
                 np.array(df_case_KS_OctDec['per_day_cases']), 1, "B")
```

```
D > C, We reject Ho: A
D > C, We reject Ho: B
```

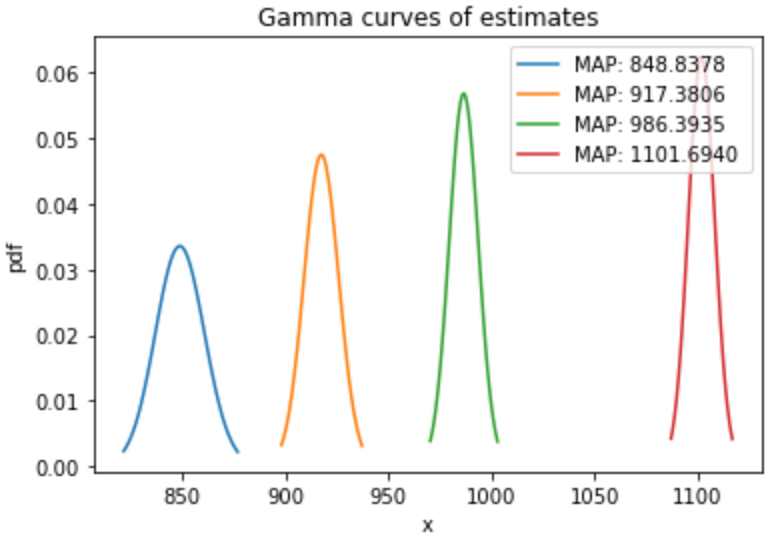

week 1
week 2
('Max Diff=', 0.5543478260869564)
Pr[X<=x]
x
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200


week 1
week 2
('Max Diff=', 0.5434782608695661)
Pr[X<=x]
x
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000

2c

In [101…

```
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,scale =
        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()

prior_start_date = '2020-06-01'
prior_end_date = '2020-06-28'
df_case_IL_prior = df_case_IL[(df_case_IL['submission_date'] >= pd.to_datetime(prior_start
                                        & (df_case_IL['submission_date'] < pd.to_datetime(p
df_case_IL_prior['cases_plus_deaths'] = df_case_IL_prior['per_day_cases'] + \
                                                df_case_IL_prior['per_day_deaths']
posterior_start_date = '2020-06-29'
posterior_end_date = '2020-07-05'
df_case_IL_posterior = df_case_IL[(df_case_IL['submission_date'] >= pd.to_datetime(posteri
                                        & (df_case_IL['submission_date'] < pd.to_datetime(p
posterior_cases_plus_death = df_case_IL_posterior['per_day_cases'] + \
                                                df_case_IL_posterior['per_day_deaths']

lambda_sample = df_case_IL_prior['cases_plus_deaths'].mean()

# prior beta
prior_beta = 1/lambda_sample


## Since the prior is exponential and likelihood is poisson, the posterior is gamma distri
likelihood_exp_power = len(posterior_cases_plus_death)
likelihood_lambda_power = np.sum(posterior_cases_plus_death)
prior_exp_power = prior_beta
prior_lambda_power = 0

table = []
for i in range(4):
    prior_exp_power, prior_lambda_power = prior_exp_power + likelihood_exp_power, prior_la
    table.append([prior_lambda_power + 1,prior_exp_power])

    df_case_IL_posterior = df_case_IL[166 + 7*i : 173 + 7*i]

    posterior_cases_plus_death = df_case_IL_posterior['per_day_cases'] + df_case_IL_poster

    likelihood_exp_power = len(posterior_cases_plus_death)
    likelihood_lambda_power = np.sum(posterior_cases_plus_death)

plot_gamma(table)
```



2d

In [102…

```
startDateTrain = '2021-05-01'
endDateTrain = '2021-05-21'

startDateTest = '2021-05-21'
endDateTest = '2021-05-28'

df_vac_IL_May_train = df_vac_IL[(df_vac_IL['Date'] >= pd.to_datetime(startDateTrain))
                                & (df_vac_IL['Date'] <= pd.to_datetime(endDateTrain))]
df_vac_IL_May_test = df_vac_IL[(df_vac_IL['Date'] > pd.to_datetime(startDateTest))
                                & (df_vac_IL['Date'] <= pd.to_datetime(endDateTest))]

df_vac_KS_May_train = df_vac_KS[(df_vac_KS['Date'] >= pd.to_datetime(startDateTrain))
                                & (df_vac_KS['Date'] <= pd.to_datetime(endDateTrain))]
df_vac_KS_May_test = df_vac_KS[(df_vac_KS['Date'] > pd.to_datetime(startDateTest))
                                & (df_vac_KS['Date'] <= pd.to_datetime(endDateTest))]
```

In [103…

```
def ewma_plot(X, actual_values, pred_values):
    plt.plot(X, actual_values, label="original values")
    plt.plot(X, pred_values, label="Predicted values")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend(loc='upper left')
    plt.xticks(rotation=30)
    plt.show()

def ewma_train(alpha, data):
    y_t_hat = data['Administered'].iloc[0]
    for i in range(data.shape[0]):
        y_t = data['Administered'].iloc[i]
        y_t_hat = alpha*y_t + (1-alpha) * y_t_hat
    return y_t_hat

def ewma_predict(alpha, test, y_t_hat):
    n = len(test)
    mse = 0
    mape = 0
    pred_values = []
    for i in range(len(test)):
        y_t = test['Administered'].iloc[i]
        residual = y_t_hat - y_t
        print("Date:" , test['Date'].iloc[i], "Predicted Value:", round(y_t_hat, 2), "Actu
```

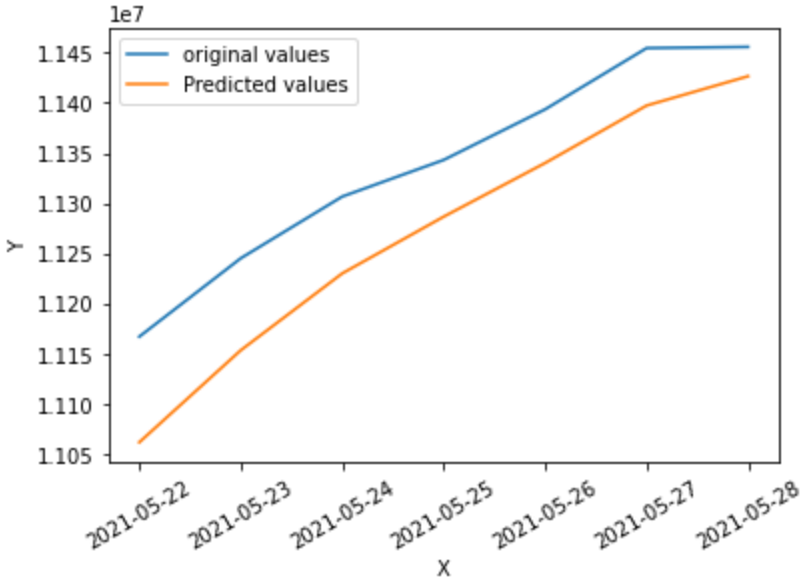
```
        mape += (abs(residual) / y_t) * 100
        mse += residual**2
        y_t_hat = alpha*y_t + (1-alpha) * y_t_hat
        pred_values.append(y_t_hat)

    print("MAPE:", round(mape/n, 2))
    print("MSE:", round(mse/n, 2))

    ewma_plot(np.array(test['Date']), test['Administered'], np.array(pred_values))
```

EWMA (0.5) for state IL

In [104…

```
alpha = 0.5
y_t_hat = ewma_train(alpha, df_vac_IL_May_train)
ewma_predict(alpha, df_vac_IL_May_test, y_t_hat)
```

```
Date: 2021-05-22 00:00:00 Predicted Value: 10956847.62 Actual Value: 11167219
Date: 2021-05-23 00:00:00 Predicted Value: 11062033.31 Actual Value: 11245253
Date: 2021-05-24 00:00:00 Predicted Value: 11153643.16 Actual Value: 11306682
Date: 2021-05-25 00:00:00 Predicted Value: 11230162.58 Actual Value: 11343172
Date: 2021-05-26 00:00:00 Predicted Value: 11286667.29 Actual Value: 11393564
Date: 2021-05-27 00:00:00 Predicted Value: 11340115.64 Actual Value: 11454543
Date: 2021-05-28 00:00:00 Predicted Value: 11397329.32 Actual Value: 11455715
MAPE: 1.19
MSE: 20278143102.54
```

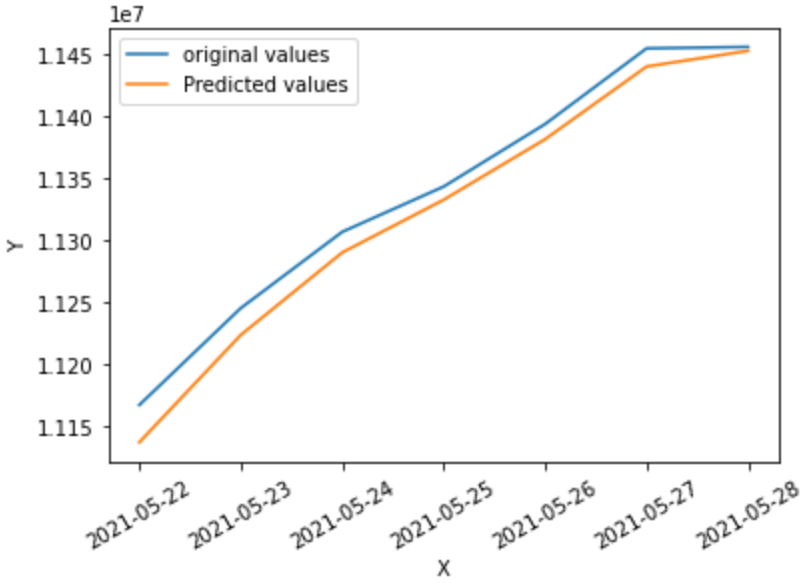


EWMA (0.8) for state IL

In [105…

```
alpha = 0.8
y_t_hat = ewma_train(alpha, df_vac_IL_May_train)
ewma_predict(alpha, df_vac_IL_May_test, y_t_hat)
```

```
Date: 2021-05-22 00:00:00 Predicted Value: 11017040.05 Actual Value: 11167219
Date: 2021-05-23 00:00:00 Predicted Value: 11137183.21 Actual Value: 11245253
Date: 2021-05-24 00:00:00 Predicted Value: 11223639.04 Actual Value: 11306682
Date: 2021-05-25 00:00:00 Predicted Value: 11290073.41 Actual Value: 11343172
Date: 2021-05-26 00:00:00 Predicted Value: 11332552.28 Actual Value: 11393564
Date: 2021-05-27 00:00:00 Predicted Value: 11381361.66 Actual Value: 11454543
Date: 2021-05-28 00:00:00 Predicted Value: 11439906.73 Actual Value: 11455715
MAPE: 0.69
MSE: 7610889878.77
```

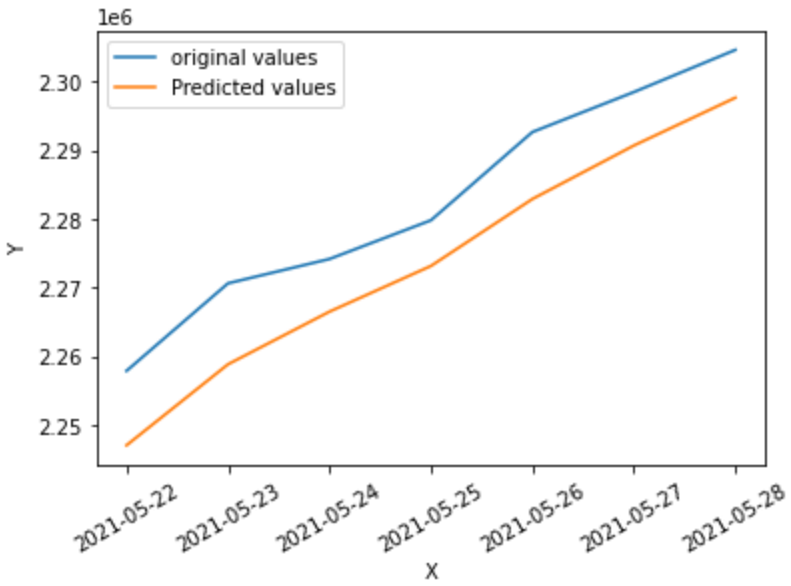


EWMA (0.5) for state KS

In [106…

```
alpha = 0.5
y_t_hat = ewma_train(alpha, df_vac_KS_May_train)
ewma_predict(alpha, df_vac_KS_May_test, y_t_hat)
```

```
Date: 2021-05-22 00:00:00 Predicted Value: 2236295.02 Actual Value: 2257939
Date: 2021-05-23 00:00:00 Predicted Value: 2247117.01 Actual Value: 2270670
Date: 2021-05-24 00:00:00 Predicted Value: 2258893.51 Actual Value: 2274207
Date: 2021-05-25 00:00:00 Predicted Value: 2266550.25 Actual Value: 2279836
Date: 2021-05-26 00:00:00 Predicted Value: 2273193.13 Actual Value: 2292673
Date: 2021-05-27 00:00:00 Predicted Value: 2282933.06 Actual Value: 2298485
Date: 2021-05-28 00:00:00 Predicted Value: 2290709.03 Actual Value: 2304616
MAPE: 0.77
MSE: 321278767.12
```



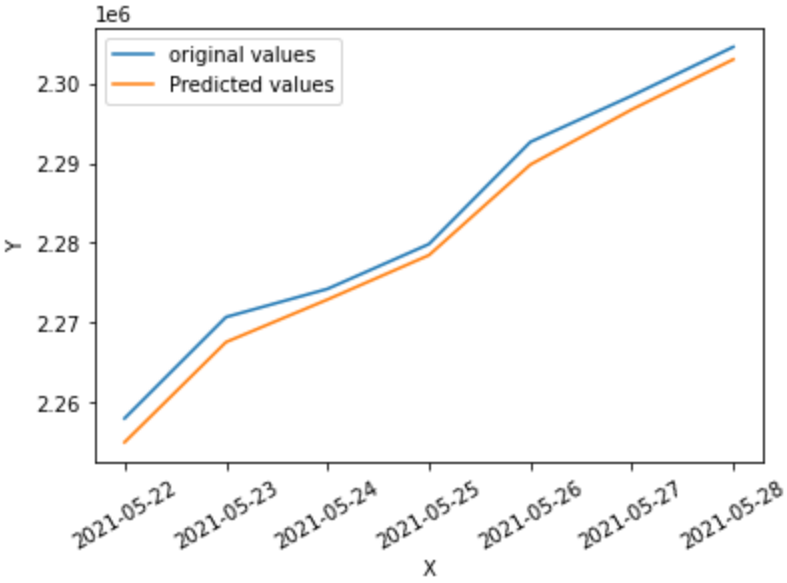
EWMA (0.8) for state KS

In [107…

```
alpha = 0.8
y_t_hat = ewma_train(alpha, df_vac_KS_May_train)
ewma_predict(alpha, df_vac_KS_May_test, y_t_hat)
```

```
Date: 2021-05-22 00:00:00 Predicted Value: 2242927.16 Actual Value: 2257939
Date: 2021-05-23 00:00:00 Predicted Value: 2254936.63 Actual Value: 2270670
```

```
Date: 2021-05-24 00:00:00 Predicted Value: 2267523.33 Actual Value: 2274207
Date: 2021-05-25 00:00:00 Predicted Value: 2272870.27 Actual Value: 2279836
Date: 2021-05-26 00:00:00 Predicted Value: 2278442.85 Actual Value: 2292673
Date: 2021-05-27 00:00:00 Predicted Value: 2289826.97 Actual Value: 2298485
Date: 2021-05-28 00:00:00 Predicted Value: 2296753.39 Actual Value: 2304616
MAPE: 0.47
MSE: 129338067.39
```



In [108…

```python
def ar_plot(X, actual_values, pred_values):
    plt.plot(X, actual_values, label="original")
    plt.plot(X, pred_values, label="Predictions")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend(loc='upper left')
    plt.xticks(rotation=30)
    plt.show()

def ar_train(p, curr_len, train_data):
    X = []
    Y = []
    for i in range(curr_len):
        if(i+p < curr_len):
            X.append([1])
            X[i] = X[i]+list(train_data[i:i+p])
            Y.append(train_data[i+p])
        else:
            break
    beta=np.matmul(np.linalg.inv(np.matmul(np.transpose(X),X)),np.matmul(np.transpose(X),Y
    return beta

def ar_predict(p, train, test):
    train_dates = train['Date']
    train_data = np.array(train['Administered'])

    test_dates = np.array(test['Date'])
    test_data = np.array(test['Administered'])

    train_data = np.hstack([train_data, test_data])
    data_len = train_data.shape[0] - test.shape[0] #test data length

    error = np.zeros(test.shape[0])
    mse = np.zeros(test.shape[0])

    predictions = np.zeros(test.shape[0])
    for i in range(data_len,data_len + test.shape[0]):
        testx = [1]
```

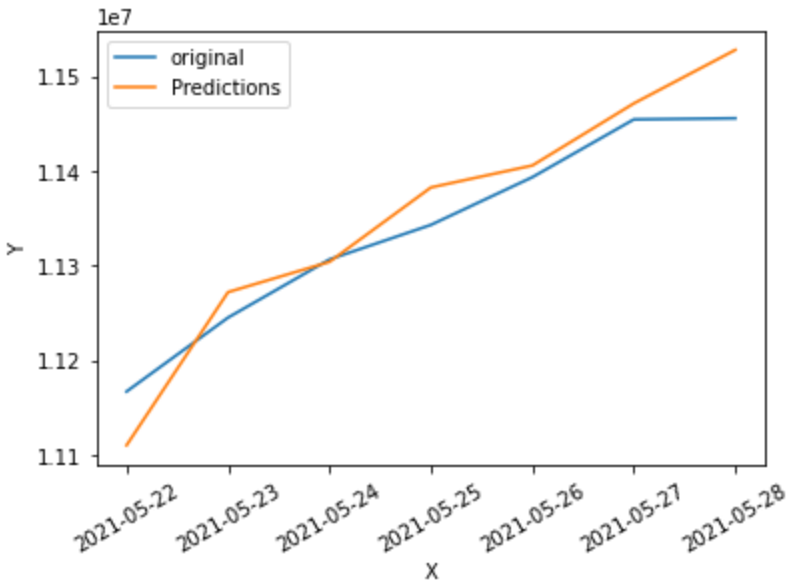
```
        testx = np.hstack([[1], train_data[i-p:i]])
        beta = ar_train(p, i, train_data)

        y_t_hat = predictions[i-data_len] = np.matmul(testx, beta)
        y_t = train_data.data[i]
        error[i-data_len] = (abs(predictions[i-data_len] - train_data[i]) / train_data[i])
        print("Date: ", test_dates[i-data_len], "Predicted Value:", round(predictions[i-da
        residual = y_t_hat - y_t
        mse[i-data_len] = residual**2
    ar_plot(test_dates, test_data, predictions)

    print("MAPE: " + "{:5.2f}".format(np.mean(error)))
    print("MSE : " + "{:5.2f}".format(np.mean(mse)))
    return np.mean(error)
```

AR(3) for state IL

In [109…

```
p = 3
ar_predict(p, df_vac_IL_May_train, df_vac_IL_May_test)
```

```
Date:  2021-05-22T00:00:00.000000000 Predicted Value: 11110454.85 Actual value: 11167219 E
rror: 0.51
Date:  2021-05-23T00:00:00.000000000 Predicted Value: 11272129.62 Actual value: 11245253 E
rror: 0.24
Date:  2021-05-24T00:00:00.000000000 Predicted Value: 11304100.81 Actual value: 11306682 E
rror: 0.02
Date:  2021-05-25T00:00:00.000000000 Predicted Value: 11382688.73 Actual value: 11343172 E
rror: 0.35
Date:  2021-05-26T00:00:00.000000000 Predicted Value: 11405889.94 Actual value: 11393564 E
rror: 0.11
Date:  2021-05-27T00:00:00.000000000 Predicted Value: 11471439.28 Actual value: 11454543 E
rror: 0.15
Date:  2021-05-28T00:00:00.000000000 Predicted Value: 11527926.52 Actual value: 11455715 E
rror: 0.63
```




```
MAPE:  0.29
MSE : 1594953077.54
```

Out[109…

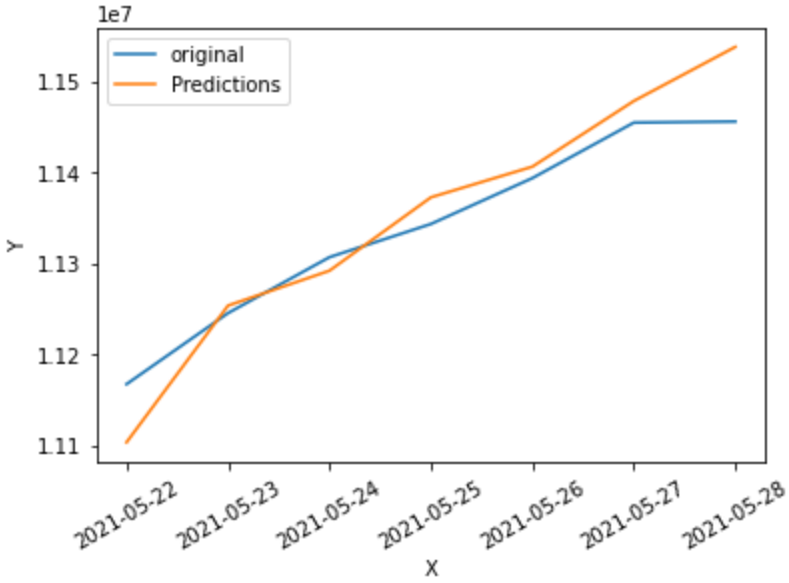
```
0.28636604675973937
```

AR(5) for state IL

In [110…

```
p = 5
ar_predict(p, df_vac_IL_May_train, df_vac_IL_May_test)
```

```
Date:  2021-05-22T00:00:00.000000000 Predicted Value: 11103365.41 Actual value: 11167219 E
rror: 0.57
Date:  2021-05-23T00:00:00.000000000 Predicted Value: 11253508.11 Actual value: 11245253 E
rror: 0.07
Date:  2021-05-24T00:00:00.000000000 Predicted Value: 11291843.08 Actual value: 11306682 E
rror: 0.13
Date:  2021-05-25T00:00:00.000000000 Predicted Value: 11372439.76 Actual value: 11343172 E
rror: 0.26
Date:  2021-05-26T00:00:00.000000000 Predicted Value: 11406116.72 Actual value: 11393564 E
rror: 0.11
Date:  2021-05-27T00:00:00.000000000 Predicted Value: 11478298.61 Actual value: 11454543 E
rror: 0.21
Date:  2021-05-28T00:00:00.000000000 Predicted Value: 11537690.44 Actual value: 11455715 E
rror: 0.72
```




```
MAPE:  0.30
MSE : 1809156645.82
```

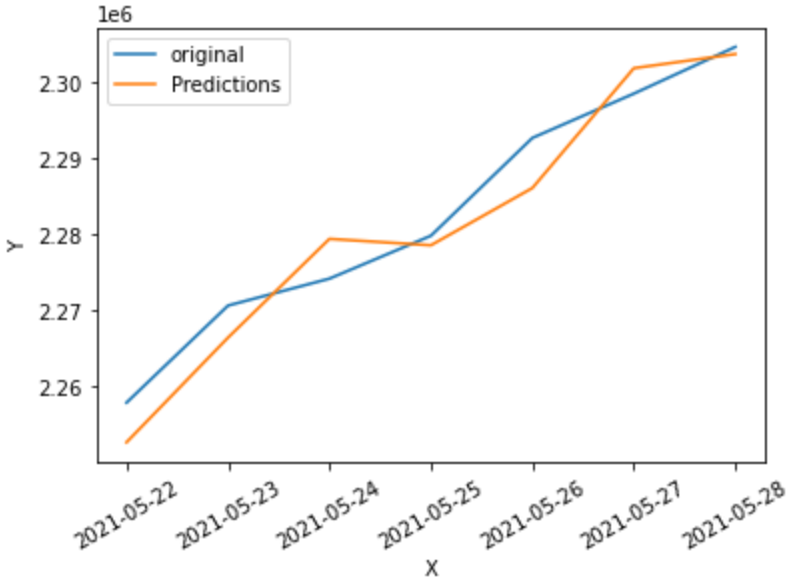
Out[110… 0.29537364214680933

AR(3) for state KS

In [111…

```
p = 3
ar_predict(p, df_vac_KS_May_train, df_vac_KS_May_test)
```

```
Date:  2021-05-22T00:00:00.000000000 Predicted Value: 2252731.98 Actual value: 2257939 Err
or: 0.23
Date:  2021-05-23T00:00:00.000000000 Predicted Value: 2266466.46 Actual value: 2270670 Err
or: 0.19
Date:  2021-05-24T00:00:00.000000000 Predicted Value: 2279423.77 Actual value: 2274207 Err
or: 0.23
Date:  2021-05-25T00:00:00.000000000 Predicted Value: 2278586.84 Actual value: 2279836 Err
or: 0.05
Date:  2021-05-26T00:00:00.000000000 Predicted Value: 2286099.3 Actual value: 2292673 Erro
r: 0.29
Date:  2021-05-27T00:00:00.000000000 Predicted Value: 2301824.01 Actual value: 2298485 Err
or: 0.15
Date:  2021-05-28T00:00:00.000000000 Predicted Value: 2303677.24 Actual value: 2304616 Err
or: 0.04
```




```
MAPE:  0.17
MSE : 18400226.53
0.1675204401674782
```

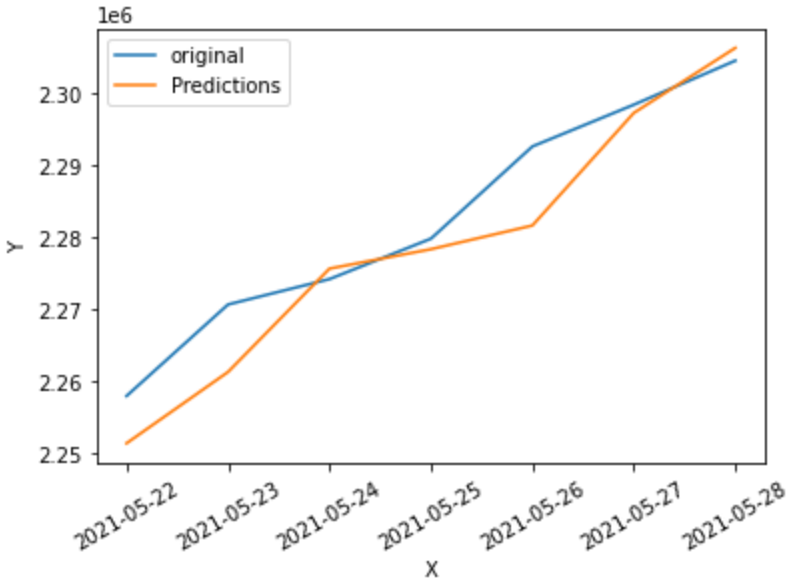
Out[111…

AR(5) for state KS

In [112…

```
p = 5
ar_predict(p, df_vac_KS_May_train, df_vac_KS_May_test)
```

```
Date:  2021-05-22T00:00:00.000000000 Predicted Value: 2251355.3 Actual value: 2257939 Erro
r: 0.29
Date:  2021-05-23T00:00:00.000000000 Predicted Value: 2261279.77 Actual value: 2270670 Err
or: 0.41
Date:  2021-05-24T00:00:00.000000000 Predicted Value: 2275665.2 Actual value: 2274207 Erro
r: 0.06
Date:  2021-05-25T00:00:00.000000000 Predicted Value: 2278368.67 Actual value: 2279836 Err
or: 0.06
Date:  2021-05-26T00:00:00.000000000 Predicted Value: 2281667.85 Actual value: 2292673 Err
or: 0.48
Date:  2021-05-27T00:00:00.000000000 Predicted Value: 2297340.04 Actual value: 2298485 Err
or: 0.05
Date:  2021-05-28T00:00:00.000000000 Predicted Value: 2306390.27 Actual value: 2304616 Err
or: 0.08
```




```
MAPE:  0.21
MSE : 37338999.53
```

Out[112… 0.20577424475826558

2e

In [113…
```
startDateSept = '2021-09-01'
endDateSept = '2021-09-30'

startDateNov = '2021-11-01'
endDateNov = '2021-11-30'

df_vac_IL_Sept = df_vac_IL[(df_vac_IL['Date'] >= pd.to_datetime(startDateSept))
                           & (df_vac_IL['Date'] <= pd.to_datetime(endDateSept))]
df_vac_IL_Nov = df_vac_IL[(df_vac_IL['Date'] >= pd.to_datetime(startDateNov))
                           & (df_vac_IL['Date'] <= pd.to_datetime(endDateNov))]

df_vac_KS_Sept = df_vac_KS[(df_vac_KS['Date'] >= pd.to_datetime(startDateSept))
                           & (df_vac_KS['Date'] <= pd.to_datetime(endDateSept))]
df_vac_KS_Nov = df_vac_KS[(df_vac_KS['Date'] >= pd.to_datetime(startDateNov))
                           & (df_vac_KS['Date'] <= pd.to_datetime(endDateNov))]

IL_Sept_adm = np.array(df_vac_IL_Sept['Administered'])
IL_Nov_adm = np.array(df_vac_IL_Nov['Administered'])
KS_Sept_adm = np.array(df_vac_KS_Sept['Administered'])
KS_Nov_adm = np.array(df_vac_KS_Nov['Administered'])

delta_Sept = np.subtract(IL_Sept_adm, KS_Sept_adm)
delta_Nov = np.subtract(IL_Nov_adm, KS_Nov_adm)
```

In [114…
```
def get_uncorrected_variance(data, mean):
    return float(sum([(x-mean)**2 for x in data])) / len(data)

def t_one_sample(data):
    x_bar = np.mean(data)
    sample_var = get_uncorrected_variance(data, x_bar)
    num = x_bar
    den = np.sqrt( sample_var / len(data))
    t_stats = num / den
    print("t statistic = " + str(abs(t_stats)))

    # Comparing our statistic with critical value
    # Critical value for n=30, alpha=0.05 is 2.045 -> 2 tailed so (29, 0.025)
    if abs(t_stats) > 2.045:
        print("Reject the Null Hypothesis")
    else:
        print("Accept the Null Hypothesis")
```

In [115…
```
t_one_sample(delta_Sept)
```

```
t statistic = 476.8743437550533
Reject the Null Hypothesis
```

In [116…
```
t_one_sample(delta_Nov)
```

```
t statistic = 205.69676113334904
Reject the Null Hypothesis
```

In [ ]: