Imports

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import lag_plot
from scipy.stats import gamma
import math
```

#Deaths Data Cleaning and Visualization

#Deaths data taken from JHU dataset for NY state

https://github.com/CSSEGISandData/COVID-19

```
deaths = pd.read_csv("CSE544_Project/deaths.csv")
```

```
deaths.head()
```

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat
0	16.0	AS	ASM	16	60.0	NaN	American Samoa	US	-14.2710
1	316.0	GU	GUM	316	66.0	NaN	Guam	US	13.4443
2	580.0	MP	MNP	580	69.0	NaN	Northern Mariana Islands	US	15.0979
3	630.0	PR	PRI	630	72.0	NaN	Puerto Rico	US	18.2208
4	850.0	VI	VIR	850	78.0	NaN	Virgin Islands	US	18.3358

5 rows × 113 columns

```
if 'New York' in deaths['Province_State'].unique():
  print("New York available")
```

```
New York available
```

```
ny_deaths = deaths.loc[deaths['Province_State'] == 'New York']
```

```
ny_deaths.head()
```

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Reg
1833	84036001.0	US	USA	840	36001.0	Albany	New York	
1834	84036003.0	US	USA	840	36003.0	Allegany	New York	
1835	84036005.0	US	USA	840	36005.0	Bronx	New York	
1836	84036007.0	US	USA	840	36007.0	Broome	New York	
1837	84036009.0	US	USA	840	36009.0	Cattaraugus	New York	

5 rows × 113 columns

```
ny_deaths.shape
```

```
(64, 113)
```

```
# since there are 64 rows of NY state data for different counties, we sum to get the
ny_deaths = ny_deaths.groupby(['Province_State']).sum()
```

```
ny_deaths.head()
```

	UID	code3	FIPS	Lat	Long_	Population
Province_State						
New York	5.378406e+09	53760	2405916.0	2637.737383	-4679.399365	23628065

1 rows × 107 columns

```
ny_deaths.drop(['UID', 'code3', 'FIPS', 'Lat', 'Long_', 'Population'], inplace=True,
```

```
ny_deaths.head()
```

```
ny_deaths = ny_deaths.transpose()
```

```
ny_deaths.columns = ['#Deaths']
```

```
ny_deaths.head()
```

	#Deaths
1/22/20	0
1/23/20	0
1/24/20	0
1/25/20	0
1/26/20	0

```
# Wanted to work with non-zero data points
ny_deaths_nonZero = ny_deaths.loc[ny_deaths['#Deaths'] > 0]
ny_deaths_nonZero.shape
```

```
(51, 1)
```

```
# plotting to see the trend of non-zero #deaths
ny_deaths_nonZero.plot(kind='bar', figsize=(15,10), title="#Deaths against date")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa65564e0>
```

By the graph it was evident that the values were cumulative, so we needed to calculate the #deaths_p

Also, we could see there was inconsistency in the data for a day where the cumulative #deaths dropp

```
ny_deaths_indexed = ny_deaths_nonZero
ny_deaths_indexed['#Deaths_per_day'] = ny_deaths_indexed['#Deaths'].diff().fillna(ny_
ny_deaths_indexed['index'] = np.arange(len(ny_deaths_indexed))  # need the index to a
ny_deaths_indexed
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  This is separate from the ipykernel package so we can avoid doing imports unti
```

	#Deaths	#Deaths_per_day	index
3/12/20	1	1.0	0
3/13/20	2	1.0	1
3/14/20	6	4.0	2
3/15/20	12	6.0	3
3/16/20	24	12.0	4
3/17/20	38	14.0	5
3/18/20	63	25.0	6
3/19/20	96	33.0	7
3/20/20	151	55.0	8
3/21/20	195	44.0	9
3/22/20	286	91.0	10
3/23/20	387	101.0	11
3/24/20	512	125.0	12
3/25/20	659	147.0	13
3/26/20	902	243.0	14
3/27/20	1211	309.0	15
3/28/20	1588	377.0	16
3/29/20	2016	428.0	17
3/30/20	2556	540.0	18
3/31/20	3207	651.0	19
4/1/20	3917	710.0	20
4/2/20	4730	813.0	21
4/3/20	5418	688.0	22
4/4/20	5991	573.0	23

4/5/20	6785	794.0	24
4/6/20	7809	1024.0	25
4/7/20	8886	1077.0	26
4/8/20	9834	948.0	27
4/9/20	10778	944.0	28
4/10/20	11605	827.0	29
4/11/20	12498	893.0	30
4/12/20	13442	944.0	31
4/13/20	14390	948.0	32
4/14/20	15261	871.0	33

NOTE: Since we did not want to deal with negative value in #deaths_per_day and also wanted data wi
our data to a subset from March 12, 2020 to April 18, 2020 which is a total of 38 days, i.e., more than

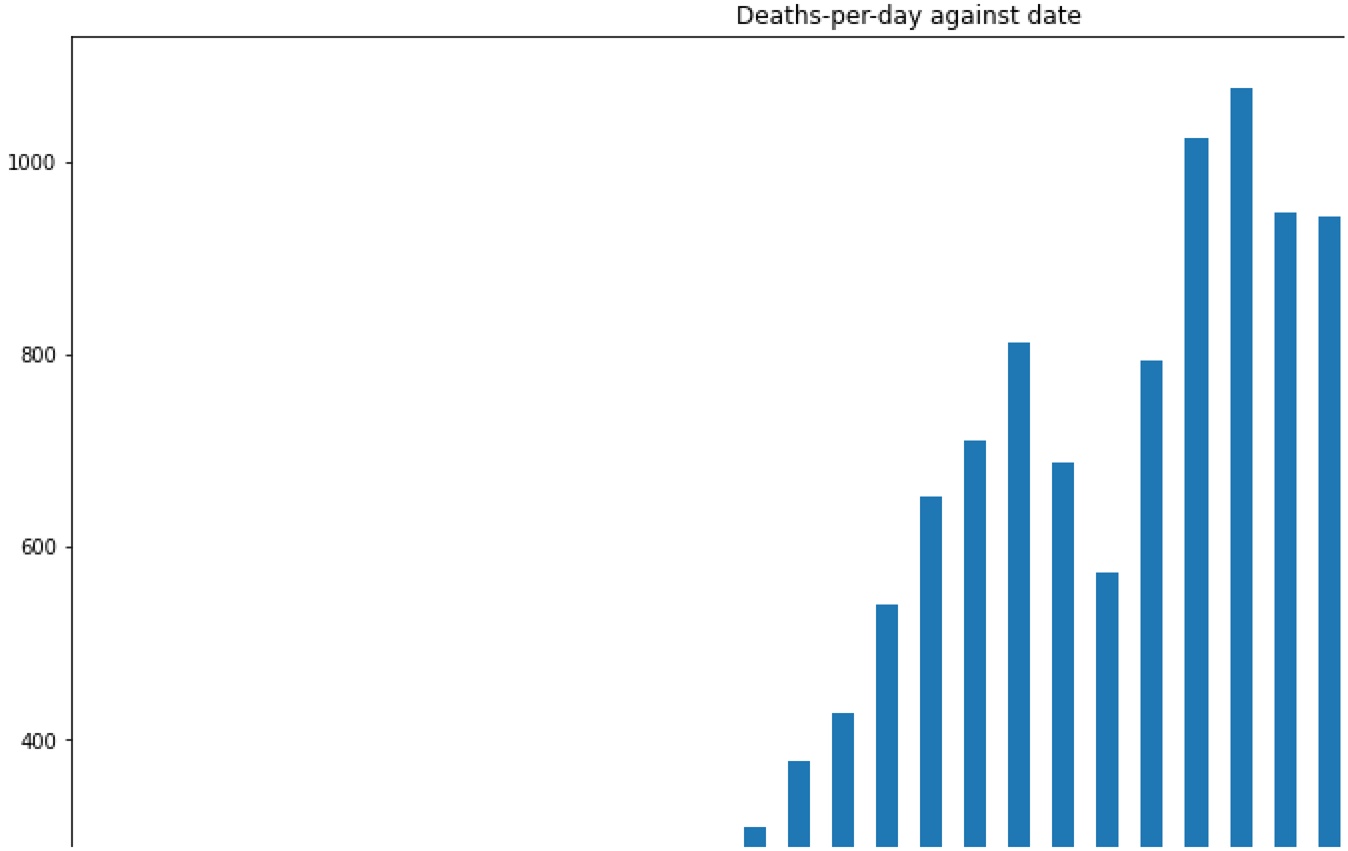
4/17/20	17755	1026.0	36

```
ny_deaths_cap = ny_deaths_indexed.loc[ny_deaths_indexed['index'] < 38]
ny_deaths_cap
```

	#Deaths	#Deaths_per_day	index
3/12/20	1	1.0	0
3/13/20	2	1.0	1
3/14/20	6	4.0	2
3/15/20	12	6.0	3
3/16/20	24	12.0	4
3/17/20	38	14.0	5
3/18/20	63	25.0	6
3/19/20	96	33.0	7
3/20/20	151	55.0	8
3/21/20	195	44.0	9
3/22/20	286	91.0	10
3/23/20	387	101.0	11
3/24/20	512	125.0	12
3/25/20	659	147.0	13
3/26/20	902	243.0	14
3/27/20	1211	309.0	15
3/28/20	1588	377.0	16
3/29/20	2016	428.0	17
3/30/20	2556	540.0	18
3/31/20	3207	651.0	19
4/1/20	3917	710.0	20
4/2/20	4730	813.0	21

```
# plot to see the #deaths_per_day
ny_deaths_cap.plot(kind='bar', y='#Deaths_per_day', figsize=(15,10), title="Deaths-pe
```

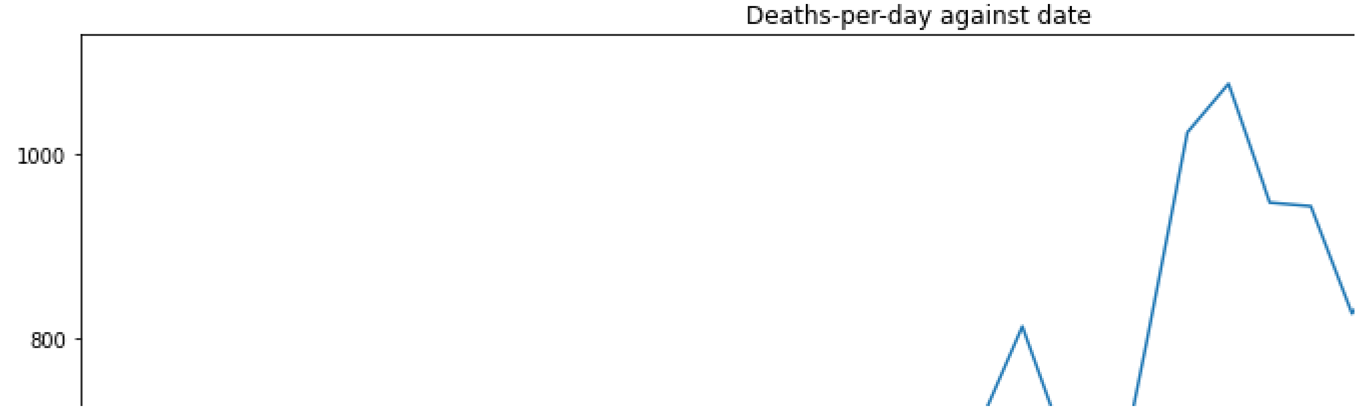
```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7faaa697fef0&gt;
```



We could see there was a continuous increase in #deaths_per_day intially which was followed by alte

```
# plotting a line graph
ny_deaths_cap.plot(y='#Deaths_per_day', figsize=(15,10), title="Deaths-per-day agains
```

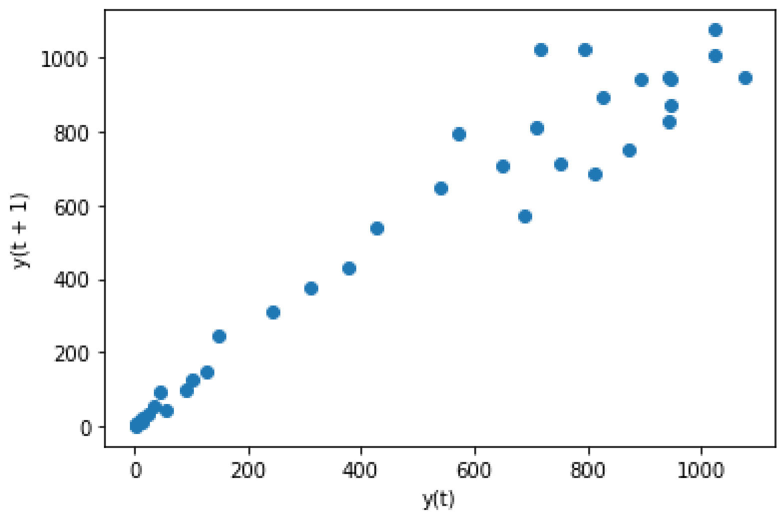
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa697f4e0>
```



The exponential increase follwed by dip, peak and alternating dips and increases is easier to visualize

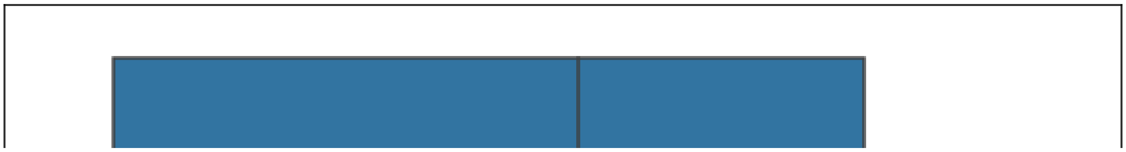
```
lag_plot(ny_deaths_cap['#Deaths_per_day'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa6378da0>
```




```
# boxplot
fig, ax = plt.subplots(figsize=(10,5))
sns.boxplot(x=ny_deaths_cap['#Deaths_per_day'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa62d0400>
```



The box plot shows there are no outliers and that the median #deaths falls short of 600 for our timefi

Outlier detection using TUKEY'S Rule

```
d_Q1 = ny_deaths_cap['#Deaths_per_day'].quantile(0.25)
d_Q3 = ny_deaths_cap['#Deaths_per_day'].quantile(0.75)
d_IQR = d_Q3 - d_Q1
print("IQR for #deaths_per_day = ", d_IQR)
```

```
IQR for #deaths_per_day =  796.0
```

```
print((ny_deaths_cap['#Deaths_per_day'] < (d_Q1 - 1.5 * d_IQR)) |(ny_deaths_cap['#Dea
```

```
3/12/20    False
3/13/20    False
3/14/20    False
3/15/20    False
3/16/20    False
3/17/20    False
3/18/20    False
3/19/20    False
```

No outliers detected using Tukey's rule. So, we do not remove any data point.

PS: Had already worked on a bunch of inferences with this Outlier result after having confirmed with A
multiplier value to detect outliers further.

#Cases Data Cleaning and Visualization

```
3/30/20    False
```

#Cases data taken from JHU dataset for NY state

https://github.com/CSSEGISandData/COVID-19

```
4/4/20     False
```

```
cases = pd.read_csv("CSE544_Project/cases.csv")
```

```
cases.head()
```

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat
0	16.0	AS	ASM	16	60.0	NaN	American Samoa	US	-14.2710
1	316.0	GU	GUM	316	66.0	NaN	Guam	US	13.4443
2	580.0	MP	MNP	580	69.0	NaN	Northern Mariana Islands	US	15.0979
3	630.0	PR	PRI	630	72.0	NaN	Puerto Rico	US	18.2208
4	850.0	VI	VIR	850	78.0	NaN	Virgin Islands	US	18.3358

5 rows × 112 columns

```
if 'New York' in cases['Province_State'].unique():
  print("New York available")
```

```
New York available
```

```
ny_cases = cases.loc[cases['Province_State'] == 'New York']
```

```
ny_cases.head()
```

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Reg
1833	84036001.0	US	USA	840	36001.0	Albany	New York	
1834	84036003.0	US	USA	840	36003.0	Allegany	New York	
1835	84036005.0	US	USA	840	36005.0	Bronx	New York	
1836	84036007.0	US	USA	840	36007.0	Broome	New York	
1837	84036009.0	US	USA	840	36009.0	Cattaraugus	New York	

5 rows × 112 columns

```
ny_cases.shape
```

```
(64, 112)
```

```
# since there are 64 rows of NY state data for different counties, we sum to get the

ny_cases = ny_cases.groupby(['Province_State']).sum()

ny_cases.head()
```

	UID	code3	FIPS	Lat	Long_	1/22/20	1/2
Province_State							
New York	5.378406e+09	53760	2405916.0	2637.737383	-4679.399365	0	

1 rows × 106 columns

```
ny_cases.drop(['UID', 'code3', 'FIPS', 'Lat', 'Long_'], inplace=True, axis=1)

ny_cases = ny_cases.transpose()

ny_cases.columns = ['#Cases']

ny_cases.head()
```

	#Cases
1/22/20	0
1/23/20	0
1/24/20	0
1/25/20	0
1/26/20	0

```
ny_cases.shape
```

```
(101, 1)
```

```
# Wanted to work with non-zero data points
ny_cases = ny_cases.loc[ny_cases['#Cases'] > 0]
ny_cases.shape
```

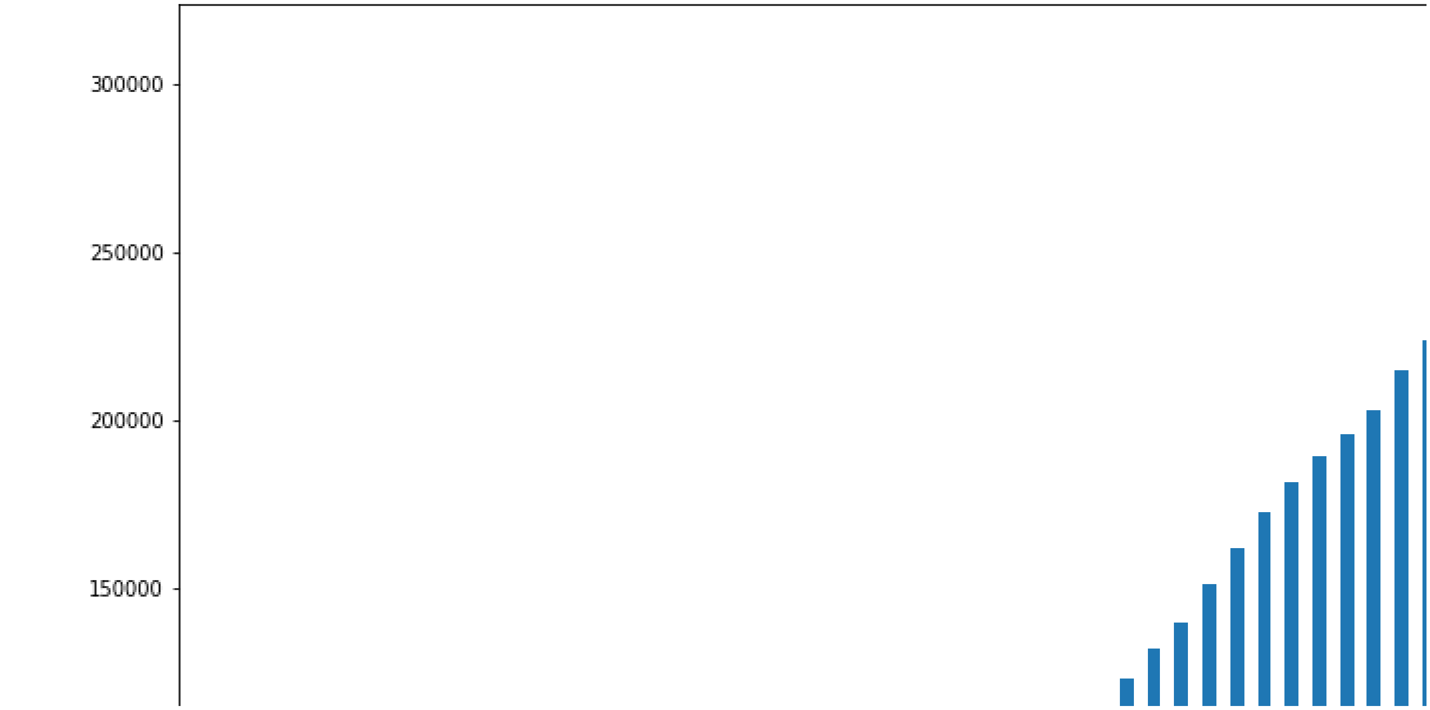
```
(61, 1)
```

```
ny_cases.head()
```

	#Cases
3/2/20	1
3/3/20	2
3/4/20	11
3/5/20	23
3/6/20	31

```
# plotting to see the trend of non-zero #deaths

ny_cases.plot(kind='bar', figsize = (15,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa3996cf8>
```



By the graph it was evident that the values were cumulative, so we needed to calculate the #cases_pe

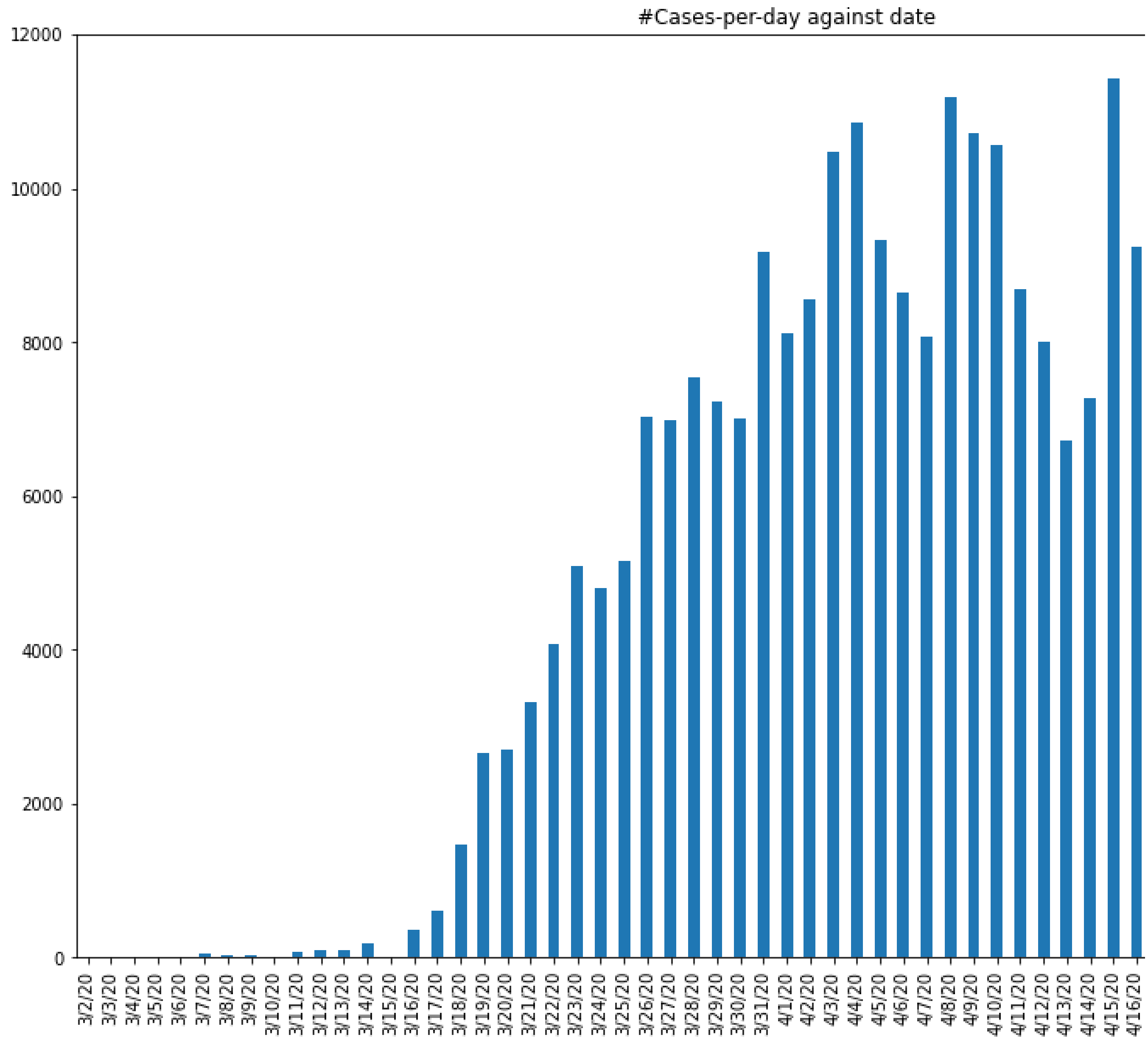
Also, we could see the cumulative data for #cases did not have a drop like we saw in the case of #de

```
ny_cases_indexed = ny_cases
ny_cases_indexed['#Cases_per_day'] = ny_cases_indexed['#Cases'].diff().fillna(ny_case
ny_cases_indexed['index'] = np.arange(len(ny_cases)) # need the index to access recor
ny_cases_indexed
```

	#Cases	#Cases_per_day	index

```
# bar plot to see the #Cases_per_day
ny_cases_indexed.plot(kind='bar', y='#Cases_per_day', figsize=(15,10), title="#Cases-
```

<matplotlib.axes._subplots.AxesSubplot at 0x7faaa395d630>



We could see there was a continuous increase in #deaths_per_day intially which was followed by sca being at a similar level for patches in between.

NOTE: Since we wanted to use corresponding data of #cases_per_day and #deaths_per_day, we extra used for #cases_per_day:

```
# to match the dates with #deaths data
ny_cases_cap = ny_cases_indexed.loc[ny_cases_indexed['index'] > 9]
```

```
ny_cases_cap = ny_cases_cap.loc[ny_cases_indexed['index'] < 48]
ny_cases_cap['index'] = np.arange(len(ny_cases_cap))
ny_cases_cap
```

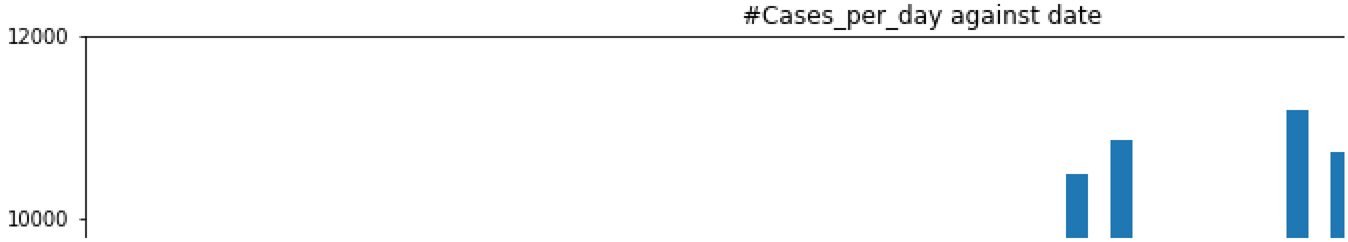
	#Cases	#Cases_per_day	index
3/12/20	327	107.0	0
3/13/20	421	94.0	1
3/14/20	613	192.0	2
3/15/20	615	2.0	3
3/16/20	967	352.0	4
3/17/20	1578	611.0	5
3/18/20	3038	1460.0	6
3/19/20	5704	2666.0	7
3/20/20	8403	2699.0	8

```
ny_cases_cap.shape
```

```
(38, 3)
```

```
# bar plot to visualize #cases_per_day for the concerned date range
ny_cases_cap.plot(kind='bar', y='#Cases_per_day', figsize=(15,10), title="#Cases_per_
```

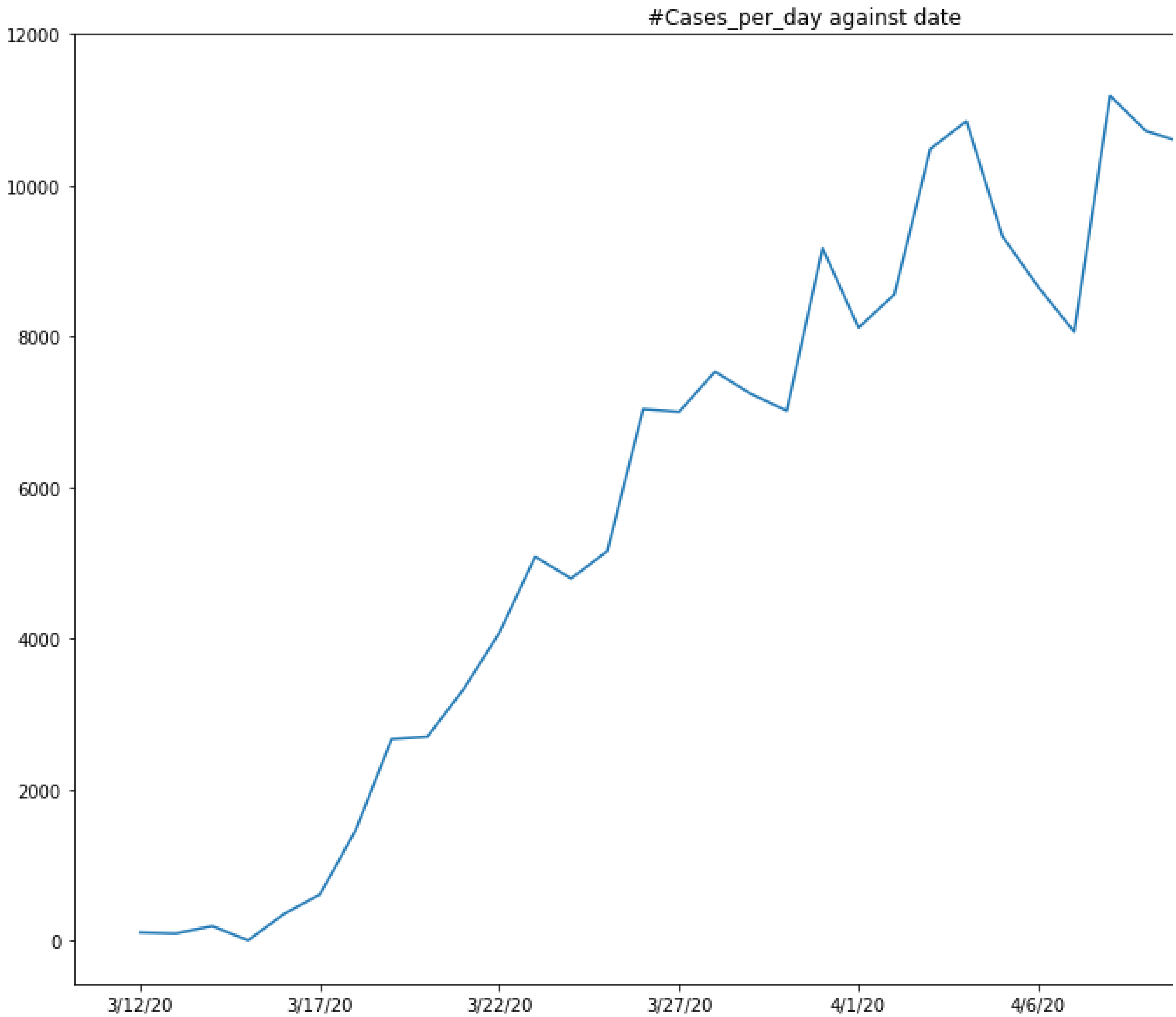
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa37e1748>



The trend is similar to what we discussed before.

```
# line plot
ny_cases_cap.plot(y='#Cases_per_day', figsize=(15,10), title="#Cases_per_day against
```

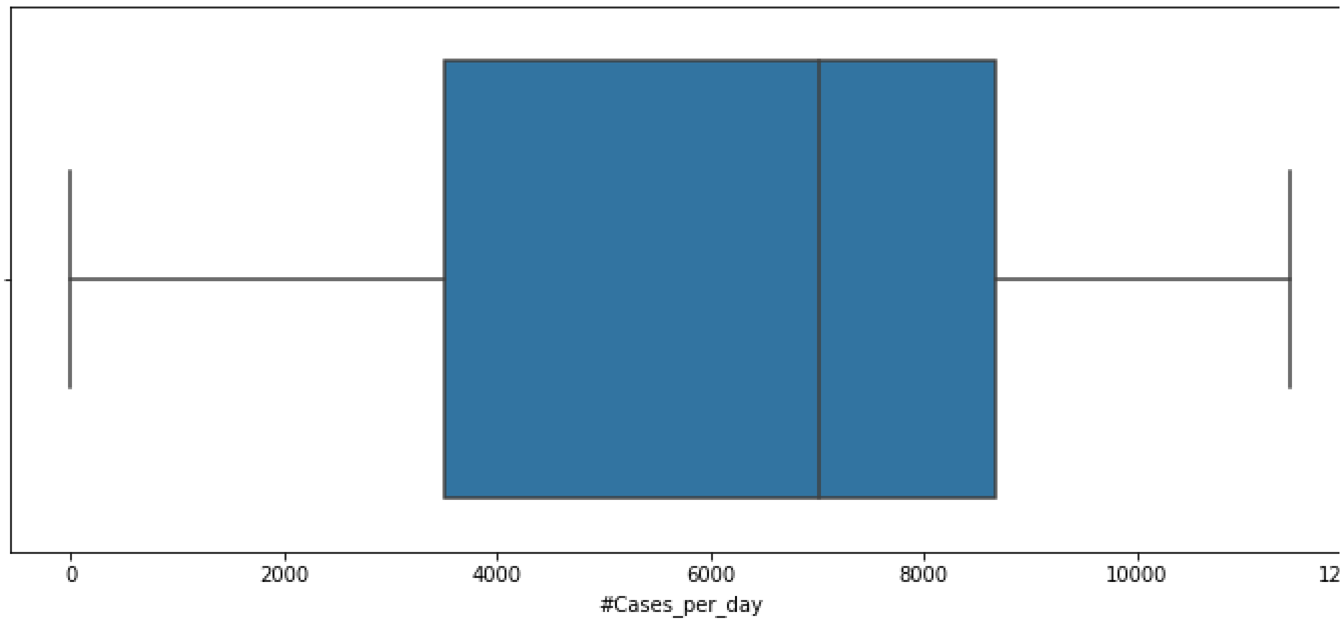
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa36b1908>



The exponential increase follwed by alternating dips and increases is easier to visualize with the line

```
fig, ax = plt.subplots(figsize=(12,5))
sns.boxplot(x=ny_cases_cap['#Cases_per_day'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa6355278>
```



The box plot shows there are no outliers and that the median #cases lies nearby 7000 for our timefra

Outlier detection using TUKEY'S Rule

```
Q1 = ny_cases_cap['#Cases_per_day'].quantile(0.25)
Q3 = ny_cases_cap['#Cases_per_day'].quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
5161.0
```

```
print((ny_cases_cap['#Cases_per_day'] < (Q1 - 1.5 * IQR)) |(ny_cases_cap['#Cases_per_
```

```
3/12/20    False
3/13/20    False
3/14/20    False
3/15/20    False
3/16/20    False
3/17/20    False
3/18/20    False
3/19/20    False
3/20/20    False
3/21/20    False
3/22/20    False
3/23/20    False
3/24/20    False
3/25/20    False
3/26/20    False
3/27/20    False
3/28/20    False
3/29/20    False
3/30/20    False
3/31/20    False
4/1/20     False
4/2/20     False
4/3/20     False
4/4/20     False
4/5/20     False
4/6/20     False
```

No outliers found using the Tukey's rule. So, we do not remove any data point.

X Data cleaning and visualization

```
4/14/20    False
```

Data is taken from: https://data.ny.gov/Transportation/Hourly-Traffic-on-Metropolitan-Transportation

Our X data will be limited to the #Vehicles commuting on Throgs Neck Bridge

```
Name: #Cases per day, dtype: bool
```

```
traffic_data = pd.read_csv("CSE544_Project/X_data.csv")
```

```
traffic_data.shape
```

```
(1591991, 6)
```

```
traffic_data.columns
```

```
Index(['Plaza ID', 'Date', 'Hour', 'Direction', '# Vehicles - ETC (E-ZPass)',
       '# Vehicles - Cash/VToll'],
      dtype='object')
```

```
traffic_data.isnull().sum()
```

```
Plaza ID                      0
Date                          0
Hour                          0
Direction                     0
# Vehicles - ETC (E-ZPass)    0
# Vehicles - Cash/VToll       0
```

No nulls found! Data is clean

```
traffic_data.head()
```

	Plaza ID	Date	Hour	Direction	# Vehicles - ETC (E-ZPass)	# Vehicles -
0	21	05/02/2020	0	I	707	
1	21	05/02/2020	1	I	409	
2	21	05/02/2020	2	I	325	
3	21	05/02/2020	3	I	334	
4	21	05/02/2020	4	I	452	

```
traffic_data.tail()
```

	Plaza ID	Date	Hour	Direction	# Vehicles - ETC (E-ZPass)	# Vehic
1591986	11	01/01/2010	19	I	2675	
1591987	11	01/01/2010	20	I	2580	
1591988	11	01/01/2010	21	I	2302	
1591989	11	01/01/2010	22	I	2170	
1591990	11	01/01/2010	23	I	1837	

https://data.ny.gov/Transportation/Daily-Traffic-on-Throgs-Neck-Bridge-Time-Line/emsg-shxw

```
traffic_data['Plaza ID'].unique()
```

```
array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30,  3,  9, 11,  1,  2,  5,  6,
        7,  8,  4])
```

Per the documentation for the throgs neck bridge we are concerned with Plaza ID 29

```
throg_neck = pd.DataFrame(traffic_data.loc[traffic_data['Plaza ID'] == 29])
```

```
throg_neck = pd.DataFrame(traffic_data.loc[traffic_data['Plaza ID'] == 29])
```

```
throg_neck.shape
```

```
(42420, 6)
```

```
throg_neck.head()
```

	Plaza ID	Date	Hour	Direction	# Vehicles - ETC (E-ZPass)	# Vehicles
336	29	05/02/2020	0	I	231	
337	29	05/02/2020	0	O	318	
338	29	05/02/2020	1	I	179	
339	29	05/02/2020	1	O	234	
340	29	05/02/2020	2	I	171	

```
throg_neck.dtypes
```

```
Plaza ID                       int64
Date                          object
Hour                           int64
Direction                     object
# Vehicles - ETC (E-ZPass)     int64
# Vehicles - Cash/VToll        int64
dtype: object
```

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

```
throg_neck = throg_neck[['Date','# Vehicles - ETC (E-ZPass)','# Vehicles - Cash/VToll
```

```
throg_neck['total vehicles'] = throg_neck['# Vehicles - Cash/VToll']+throg_neck['# Ve
```

```
throg_neck = throg_neck[['Date','total vehicles']]
```

```
throg_neck = throg_neck.groupby([throg_neck['Date']]).sum()
```

```
throg_neck.shape
```

```
(884, 1)
```

```
throg_neck.head()
```

Date	total vehicles
2017-10-22	134475
2017-10-23	118294
2017-10-24	112622
2017-10-26	121376

```
throg_neck_dated = throg_neck[(throg_neck.index >= '03/12/2020') & (throg_neck.index
throg_neck_dated.head()
```

Date	total vehicles
2020-03-12	110415
2020-03-13	109217
2020-03-14	89902
2020-03-15	78542
2020-03-16	92168

```
Q1 = np.quantile(throg_neck_dated['total vehicles'],0.25)
Q3 = np.quantile(throg_neck_dated['total vehicles'],0.75)
IQR = Q3 - Q1

print((throg_neck_dated['total vehicles'] < (Q1 - 1.5 * IQR)) |(throg_neck_dated['tot
```

```
Date
2020-03-12     True
2020-03-13     True
2020-03-14     True
2020-03-15     True
2020-03-16     True
2020-03-17     True
2020-03-18     True
2020-03-19     True
2020-03-20     True
2020-03-21    False
2020-03-22    False
2020-03-23    False
2020-03-24    False
2020-03-25    False
2020-03-26    False
2020-03-27    False
2020-03-28    False
2020-03-29     True
```

If we consider the total data together, that essentially leads to 11 of 38 entries to be classified as outl going ahead with weekly detection and removal of outliers[discussed this with Professor]

```
2020-04-03    False
```

```
throg_neck_2019 = throg_neck[(throg_neck.index >= '03/12/2019') & (throg_neck.index <
```

```
throg_neck_2019.shape
```

```
(38, 1)
```

```
throg_neck_2020 = throg_neck[(throg_neck.index >= '03/12/2020') & (throg_neck.index <
```

```
throg_neck_2020.shape
```

```
(38, 1)
```

```
# Convert data to weekly frequency
def convert_data_to_weekly(data):
  return_list = []
  i = 0
  while (i<len(data)):
    return_list.append(data[i:i+7])
    i = i+7
  return return_list

throg_neck_2019_weekly = convert_data_to_weekly(throg_neck_2019)
throg_neck_2020_weekly = convert_data_to_weekly(throg_neck_2020)


# detect outlier and remove them

# for 2019's data
```

```
for week in throg_neck_2019_weekly:
  # print(week)
  Q1_week = np.quantile(week,0.25)
  Q3_week = np.quantile(week,0.75)
  IQR_week = Q3_week - Q1_week
  print("IQR: ",IQR_week)
  print((week < (Q1_week - 1.5 * IQR_week)) |(week > (Q3_week + 1.5 * IQR_week)))
  # week.boxplot()
  # plt.show()
```

```
IQR:  5210.5
            total vehicles
Date
2019-03-12           False
2019-03-13           False
2019-03-14           False
2019-03-15           False
2019-03-16           False
2019-03-17           False
2019-03-18           False
```

```
# detect outlier and remove them

# for 2020's data
for week in throg_neck_2020_weekly:
  Q1_week = np.quantile(week,0.25)
  Q3_week = np.quantile(week,0.75)
  IQR_week = Q3_week - Q1_week
  print("IQR: ",IQR_week)
  print((week < (Q1_week - 1.5 * IQR_week)) |(week > (Q3_week + 1.5 * IQR_week)))
```

```
IQR:  13719.5
            total vehicles
Date
2020-03-12           False
2020-03-13           False
2020-03-14           False
2020-03-15           False
2020-03-16           False
2020-03-17           False
2020-03-18           False
IQR:  15473.0
            total vehicles
Date
2020-03-19           False
2020-03-20           False
2020-03-21           False
2020-03-22           False
2020-03-23           False
2020-03-24           False
2020-03-25           False
IQR:  10950.0
```

Outlier 5th April observed. Removing the data from both 2019 and 2020 data. Can remove it since it's

```
2020-03-26           False
```

```
print(throg_neck_2019.index[24])
```

```
2019-04-05 00:00:00
```

```
throg_neck_2019 = throg_neck_2019.drop(throg_neck_2019.index[24])
throg_neck_2020 = throg_neck_2020.drop(throg_neck_2020.index[24])
```

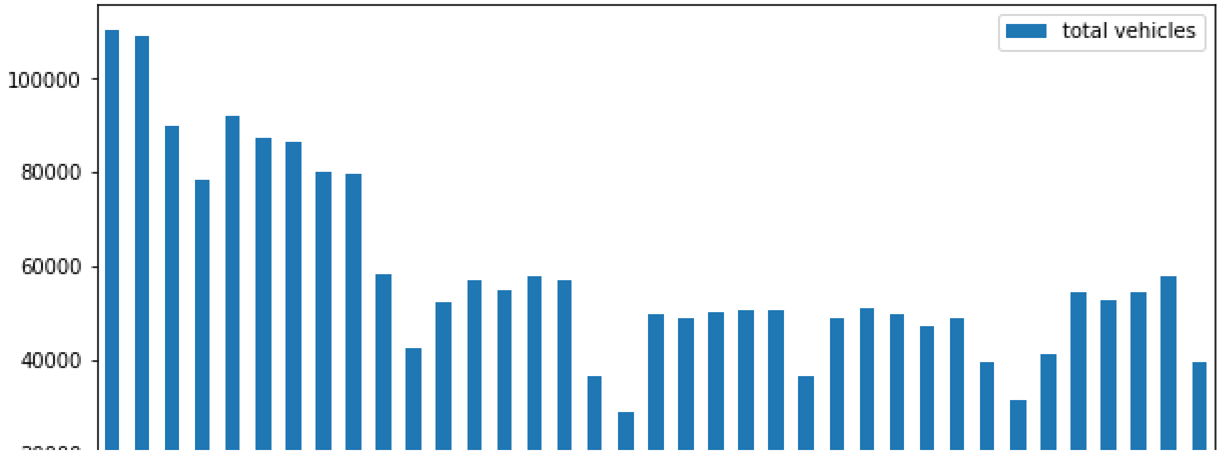
```
throg_neck_2019_weekly = convert_data_to_weekly(throg_neck_2019)
throg_neck_2020_weekly = convert_data_to_weekly(throg_neck_2020)
```

```
print(len(throg_neck_2019_weekly))
print(len(throg_neck_2020_weekly))
```

```
6
6
```

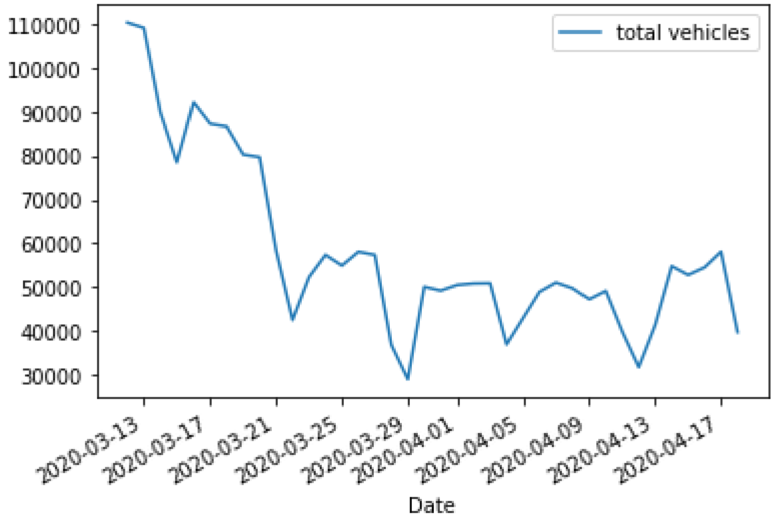
```
throg_neck_2020.plot(kind='bar',y='total vehicles',figsize=(10,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa399e7b8>
```



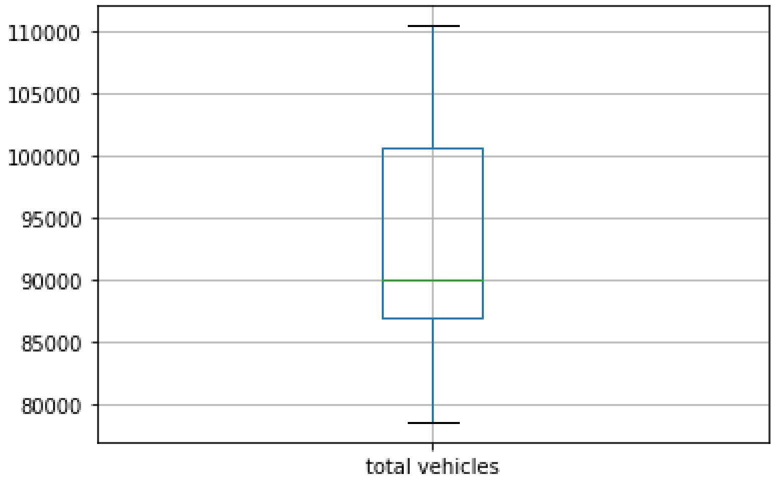

```
throg_neck_2020.plot()
```

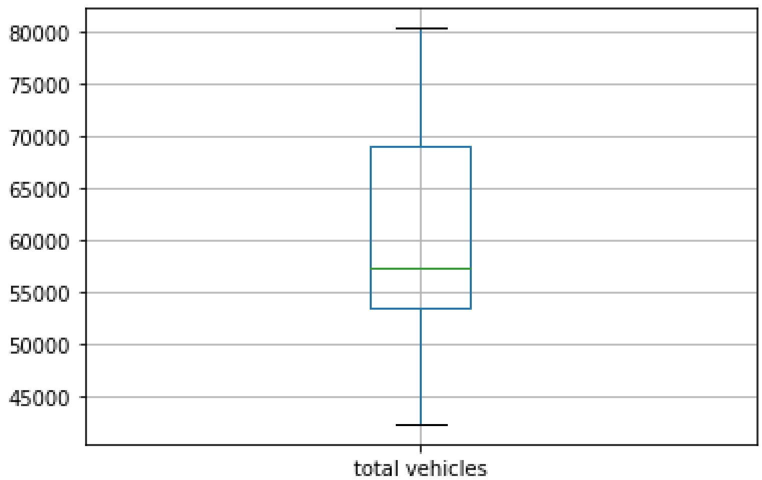
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faaa700dd30>
```

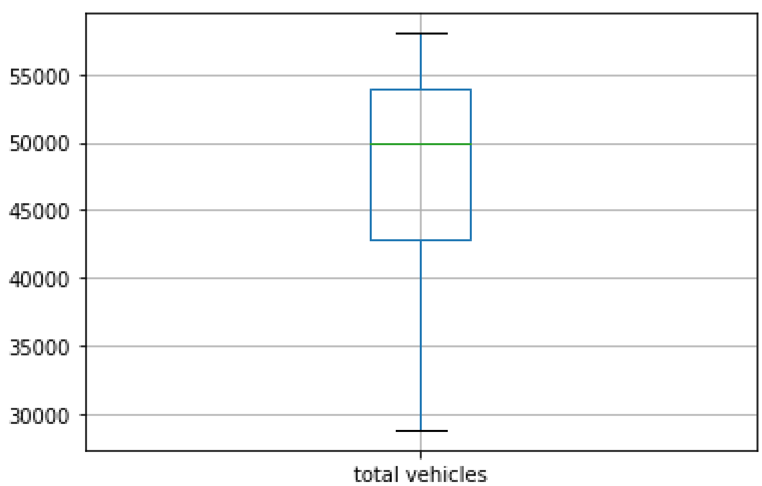


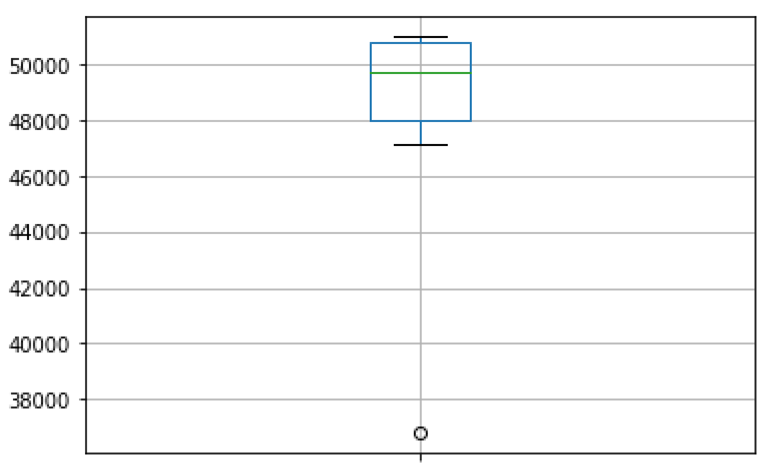
The histogram and the line chart both show significant drop in the period of covid-19

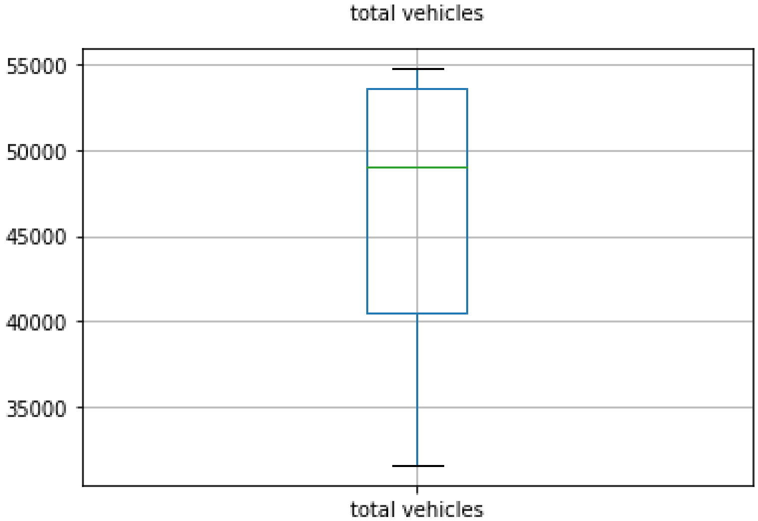
```
for week in throg_neck_2020_weekly:
  week.boxplot()
  plt.show()
```


110000
105000
100000
95000
90000
85000
80000
total vehicles


80000
75000
70000
65000
60000
55000
50000
45000
total vehicles


55000
50000
45000
40000
35000
30000
total vehicles


50000
48000
46000
44000
42000
40000
38000



Box plot shows that the outlier removal was successful and hence their are no points outside the ma:

Required Inference 1

Use your COVID19 dataset to predict the COVID19 fatality and #cases for the next one week. Use the
AR(3), (ii) AR(5), (iii) EWMA with alpha = 0.5, and (iv) EWMA with alpha = 0.8. Make sure that your dat
prediction. For example, use the first three weeks of data to predict the fourth week, and report the ac
actual fourth week data. Use metrics learned in class (MAPE as a % and MSE) to report accuracy nun

MAPE and MSE

```
def calculate_mse(Y, Y_hat):
  return np.sum(np.square(Y-Y_hat))/len(Y)

def calculate_MAPE(Y, Y_hat):
  return np.sum(((np.absolute(Y-Y_hat))/Y)*100)/len(Y)
```

EWMA

Observation: We could see from below execution that MSE and MAPE were high hence inferring that

Setting variables for #cases data:

```
ny_cases_cap
```

	#Cases	#Cases_per_day	index
3/12/20	327	107.0	0
3/13/20	421	94.0	1
3/14/20	613	192.0	2
3/15/20	615	2.0	3
3/16/20	967	352.0	4
3/17/20	1578	611.0	5
3/18/20	3038	1460.0	6
3/19/20	5704	2666.0	7
3/20/20	8403	2699.0	8
3/21/20	11727	3324.0	9
3/22/20	15800	4073.0	10
3/23/20	20884	5084.0	11
3/24/20	25681	4797.0	12
3/25/20	30841	5160.0	13
3/26/20	37877	7036.0	14
3/27/20	44876	6999.0	15
3/28/20	52410	7534.0	16
3/29/20	59648	7238.0	17
3/30/20	66663	7015.0	18
3/31/20	75833	9170.0	19
4/1/20	83948	8115.0	20
4/2/20	92506	8558.0	21
4/3/20	102987	10481.0	22
4/4/20	113833	10846.0	23
4/5/20	123160	9327.0	24
4/6/20	131815	8655.0	25
4/7/20	139875	8060.0	26
4/8/20	151061	11186.0	27
4/9/20	161779	10718.0	28
4/10/20	172348	10569.0	29

4/11/20	181026	8678.0	30
4/12/20	189033	8007.0	31
4/13/20	195749	6716.0	32
4/14/20	203020	7271.0	33
4/15/20	214454	11434.0	34
4/16/20	223691	9237.0	35
4/17/20	230597	6906.0	36
4/18/20	237474	6877.0	37

```
pdc = ny_cases_cap['#Cases_per_day'].reset_index(drop = True)  # getting the per_day_
X = pdc[:len(pdc)-7]      # using all data except last week for prediction
Y = pdc[len(pdc)-7:]      # This is the actual Y for last week
Y = np.array([Y]).T
Y = Y.squeeze()
Y_hat = np.array([])      # This is the predicted Y_hat for last week
print("Y = ", Y)
print("shape of Y = ", Y.shape)
print("initial Y_hat = ", Y_hat)
print("shape of Y_hat = ", Y_hat.shape)
X = np.array([X]).T
X = np.insert(X, 0, X[0], axis=0)   # Adding y_1_hat as y_1
X = np.flip(X)  # Flipping to get in the order of y_t, y_t-1,....,y_1
X = X.squeeze()
Y_31 = X[0]   # saving the true value of Y_31
X = X[1:]     # removing Y_31 to calculare y_hat(31|30)
              # Y31_hat is needed to start the predictions of last week (32-38)
print("X = ", X)
print(" shape of X = ", X.shape)
print("Y_31 = ", Y_31)
```

```
Y =  [ 8007.  6716.  7271. 11434.  9237.  6906.  6877.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X =  [1.0569e+04 1.0718e+04 1.1186e+04 8.0600e+03 8.6550e+03 9.3270e+03
 1.0846e+04 1.0481e+04 8.5580e+03 8.1150e+03 9.1700e+03 7.0150e+03
 7.2380e+03 7.5340e+03 6.9990e+03 7.0360e+03 5.1600e+03 4.7970e+03
 5.0840e+03 4.0730e+03 3.3240e+03 2.6990e+03 2.6660e+03 1.4600e+03
 6.1100e+02 3.5200e+02 2.0000e+00 1.9200e+02 9.4000e+01 1.0700e+02
 1.0700e+02]
 shape of X =  (31,)
Y_31 =  8678.0
```

Setting variables for #deaths data:

```
ny_deaths_cap
```

	#Deaths	#Deaths_per_day	index
3/12/20	1	1.0	0
3/13/20	2	1.0	1
3/14/20	6	4.0	2
3/15/20	12	6.0	3
3/16/20	24	12.0	4
3/17/20	38	14.0	5
3/18/20	63	25.0	6
3/19/20	96	33.0	7
3/20/20	151	55.0	8
3/21/20	195	44.0	9
3/22/20	286	91.0	10
3/23/20	387	101.0	11
3/24/20	512	125.0	12
3/25/20	659	147.0	13
3/26/20	902	243.0	14
3/27/20	1211	309.0	15
3/28/20	1588	377.0	16
3/29/20	2016	428.0	17
3/30/20	2556	540.0	18
3/31/20	3207	651.0	19
4/1/20	3917	710.0	20
4/2/20	4730	813.0	21
4/3/20	5418	688.0	22
4/4/20	5991	573.0	23
4/5/20	6785	794.0	24
4/6/20	7809	1024.0	25
4/7/20	8886	1077.0	26
4/8/20	9834	948.0	27
4/9/20	10778	944.0	28
4/10/20	11605	827.0	29

4/11/20	12498	893.0	30
4/12/20	13442	944.0	31
4/13/20	14390	948.0	32
4/14/20	15261	871.0	33
4/15/20	16013	752.0	34
4/16/20	16729	716.0	35
4/17/20	17755	1026.0	36

```
pdd = ny_deaths_cap['#Deaths_per_day'].reset_index(drop = True)  # getting the per_da
X_deaths = pdd[:len(pdd)-7]      # using all data except last week for prediction
Y_deaths = pdd[len(pdd)-7:]      # This is the actual Y for last week
Y_deaths = np.array([Y_deaths]).T
Y_deaths = Y_deaths.squeeze()
Y_hat_deaths = np.array([])      # This is the predicted Y_hat for last week
print("Y for #deaths = ", Y_deaths)
print("shape of Y = ", Y_deaths.shape)
print("initial Y_hat = ", Y_hat_deaths)
print("shape of Y_hat = ", Y_hat_deaths.shape)
X_deaths = np.array([X_deaths]).T
X_deaths = np.insert(X_deaths, 0, X_deaths[0], axis=0)   # Adding y_1_hat as y_1
X_deaths = np.flip(X_deaths)  # Flipping to get in the order of y_t, y_t-1,....,y_1
X_deaths = X_deaths.squeeze()
Y_deaths_31 = X_deaths[0]   # saving the true value of Y_31
X_deaths = X_deaths[1:]     # removing Y_31 to calculare y_hat(31|30)
              # Y31_hat is needed to start the predictions of last week (32-38)
print("X for #deaths = ", X_deaths)
print(" shape of X = ", X_deaths.shape)
print("Y_31 for #deaths = ", Y_deaths_31)
```

```
Y for #deaths =  [ 944.  948.  871.  752.  716. 1026. 1005.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X for #deaths =  [8.270e+02 9.440e+02 9.480e+02 1.077e+03 1.024e+03 7.940e+02 5.
 6.880e+02 8.130e+02 7.100e+02 6.510e+02 5.400e+02 4.280e+02 3.770e+02
 3.090e+02 2.430e+02 1.470e+02 1.250e+02 1.010e+02 9.100e+01 4.400e+01
 5.500e+01 3.300e+01 2.500e+01 1.400e+01 1.200e+01 6.000e+00 4.000e+00
 1.000e+00 1.000e+00 1.000e+00]
 shape of X =  (31,)
Y_31 for #deaths =  893.0
```

Y_hat (t+1 | t) = sum(W*X) ,

W = [alpha, aplha(1-alpha), alpha(1-alpha)^2, ... alpha(1-alpha)^t-1, (1-alpha)^t],

X = [y_t, y_t-1, ..., y_1, y_1_hat]

here,

Yhat(31|30) = sum(W*X) and

W = [alpha, aplha(1-alpha), alpha(1-alpha)^2, ... alpha(1-alpha)^29, (1-alpha)^30]

X = [y_30, y_29, ..., y_2, y_1, y_1]

Post this, we calculate Y_hat (32|31), ... Y_hat (38|37)

where Y_hat (32|31) = (32nd day prediction given #cases_per_day for 31 days)

and so on.

y_hat (t+1|t) = alpha * y_t + (1-alpha) * y_hat (t+1|t)

that is:

y_hat (32|31) = alpha * y_31 + (1-alpha) * y_hat (31|3)

...

y_hat (38|37) = alpha * y_37 + (1-alpha) * y_hat (37|36)

```
def build_W(alpha, X):
  W = np.arange(len(X)-1)
  W = alpha * ((1-alpha) ** W)
  print(W.shape)
  W = np.append(W, (1-alpha)**(len(X)-1))  #appending the last term of alpha
  print(W.shape)
  return W



def ewma(alpha, Y, Y_hat, W, X):
  Y31_hat = np.sum(W * X)   # needed to start the prediction of the last week
                            # but will not be used in MAPE and MSE calculations
  print("Y31_hat = ", Y31_hat)

  Y32_hat = np.sum((alpha*Y_31) + (1-alpha)*Y31_hat)
  print("Y_32 predicted = ", Y32_hat)
  Y_hat = np.append(Y_hat, Y32_hat)
  Y33_hat = np.sum((alpha*Y[0]) + (1-alpha)*Y_hat[0])
  print("Y_33 predicted = ", Y33_hat)
  Y_hat = np.append(Y_hat, Y33_hat)
  Y34_hat = np.sum((alpha*Y[1]) + (1-alpha)*Y_hat[1])
  print("Y_33 predicted = ", Y34_hat)
  Y_hat = np.append(Y_hat, Y34_hat)
  Y35_hat = np.sum((alpha*Y[2]) + (1-alpha)*Y_hat[2])
  print("Y_34 predicted = ", Y35_hat)
  Y_hat = np.append(Y_hat, Y35_hat)
  Y36_hat = np.sum((alpha*Y[3]) + (1-alpha)*Y_hat[3])
  print("Y_35 predicted = ", Y36_hat)
  Y_hat = np.append(Y_hat, Y36_hat)
  Y37_hat = np.sum((alpha*Y[4]) + (1-alpha)*Y_hat[4])
  print("Y_36 predicted = ", Y37_hat)
  Y_hat = np.append(Y_hat, Y37_hat)
```

```
    Y38_hat = np.sum((alpha*Y[5]) + (1-alpha)*Y_hat[5])
    print("Y_37 predicted = ", Y38_hat)
    Y_hat = np.append(Y_hat, Y38_hat)
    print("Y_hat [32-38] = ", Y_hat)
    print("actual Y [32-38] = ", Y)
    return Y_hat
```

EWMA (alpha =0.5)

For #cases

```
Y_hat = np.array([])
W = build_W(0.5, X)
Y_hat = ewma(0.5, Y, Y_hat, W, X)
```

```
(30,)
(31,)
Y31_hat =  10440.446983339265
Y_32 predicted =  9559.223491669632
Y_33 predicted =  8783.111745834816
Y_33 predicted =  7749.555872917408
Y_34 predicted =  7510.277936458704
Y_35 predicted =  9472.138968229352
Y_36 predicted =  9354.569484114676
Y_37 predicted =  8130.284742057338
Y_hat [32-38] =  [9559.22349167 8783.11174583 7749.55587292 7510.27793646 9472.1
 9354.56948411 8130.28474206]
actual Y [32-38] =  [ 8007.  6716.  7271. 11434.  9237.  6906.  6877.]
```

Calculate MAPE and MSE for EWMA(alpha=0.5)

```
print("MAPE with EWMA (alpha = 0.5) = ", calculate_MAPE(Y,Y_hat))
print("MSE with EWMA (alpha = 0.5) = ", calculate_mse(Y,Y_hat))
```

```
MAPE with EWMA (alpha = 0.5) =  21.041187987816038
MSE with EWMA (alpha = 0.5) =  4275494.97012502
```

For #deaths

```
Y_hat_deaths = np.array([])
W_deaths = build_W(0.5, X_deaths)
Y_hat_deaths = ewma(0.5, Y_deaths, Y_hat_deaths, W_deaths, X_deaths)
```

```
(30,)
(31,)
Y31_hat =  889.7040816582739
Y_32 predicted =  4783.852040829137
Y_33 predicted =  2863.9260204145685
Y_33 predicted =  1905.9630102072842
Y_34 predicted =  1388.4815051036421
Y_35 predicted =  1070.240752551821
Y_36 predicted =  893.1203762759105
```

Calculate MAPE and MSE for EWMA(alpha=0.5)

```
893.12037628  959.560188141
```

```
print("MAPE with EWMA (alpha = 0.5) = ", calculate_MAPE(Y_deaths,Y_hat_deaths))
print("MSE with EWMA (alpha = 0.5) = ", calculate_mse(Y_deaths,Y_hat_deaths))
```

```
MAPE with EWMA (alpha = 0.5) =  125.61094883754765
MSE with EWMA (alpha = 0.5) =  2862385.9473813153
```

EWMA (alpha = 0.8)

For #cases

```
Y_hat = np.array([])
W = build_W(0.8, X)
Y_hat = ewma(0.8, Y, Y_hat, W, X)
```

```
(30,)
(31,)
Y31_hat =  10593.766499258294
Y_32 predicted =  9061.15329985166
Y_33 predicted =  8217.830659970332
Y_33 predicted =  7016.366131994066
Y_34 predicted =  7220.073226398813
Y_35 predicted =  10591.214645279762
Y_36 predicted =  9507.842929055952
Y_37 predicted =  7426.36858581119
Y_hat [32-38] =  [ 9061.15329985  8217.83065997  7016.36613199  7220.0732264
 10591.21464528  9507.84292906  7426.36858581]
actual Y [32-38] =  [ 8007.   6716.   7271. 11434.   9237.   6906.   6877.]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with EWMA (alpha = 0.8) = ", calculate_MAPE(Y,Y_hat))
print("MSE with EWMA (alpha = 0.8) = ", calculate_mse(Y,Y_hat))
```

```
MAPE with EWMA (alpha = 0.8) =  19.458305817442902
MSE with EWMA (alpha = 0.8) =  4299148.792409782
```

For #deaths

```
Y_hat_deaths = np.array([])
W_deaths = build_W(0.8, X_deaths)
Y_hat_deaths = ewma(0.8, Y_deaths, Y_hat_deaths, W_deaths, X_deaths)
```

```
(30,)
(31,)
Y31_hat =  851.4211864346197
Y_32 predicted =  7112.684237286924
Y_33 predicted =  2177.7368474573846
Y_33 predicted =  1193.9473694914768
Y_34 predicted =  935.5894738982954
Y_35 predicted =  788.7178947796591
Y_36 predicted =  730.5435789559318
Y_37 predicted =  966.9087157911864
Y_hat [32-38] =  [7112.68423729 2177.73684746 1193.94736949  935.5894739   788.7
  730.54357896  966.90871579]
actual Y [32-38] =  [ 944.  948.  871.  752.  716. 1026. 1005.]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with EWMA (alpha = 0.8) = ", calculate_MAPE(Y_deaths,Y_hat_deaths))
print("MSE with EWMA (alpha = 0.8) = ", calculate_mse(Y_deaths,Y_hat_deaths))
```

```
MAPE with EWMA (alpha = 0.8) =  126.77369760969366
MSE with EWMA (alpha = 0.8) =  5685278.76666062
```

AR(3)

Observation: We could see from below execution that MSE and MAPE were high hence inferring that

Setting variables for #cases data:

```
pdc = ny_cases_cap['#Cases_per_day'].reset_index(drop = True)  # getting the per_day_
X_ar = pdc[:len(pdc)-7]      # using all data except last week's for prediction
X_ar = np.array([X_ar]).T
X_ar = X_ar.squeeze()
Y_ar = pdc[len(pdc)-7:]      # This is the actual Y for last week
Y_ar = np.array([Y_ar]).T
Y_ar = Y_ar.squeeze()
Y_hat_ar = np.array([])      # This is the predicted Y_hat for last week
print("Y for AR = ", Y_ar)
print("shape of Y = ", Y_ar.shape)
print("initial Y_hat = ", Y_hat_ar)
print("shape of Y_hat = ", Y_hat_ar.shape)

# building the training data (y_t; y_t-1, y_t-2, y_t-3)
X_train = np.zeros([len(X_ar)-3, 4])
```

```
print("X train = ", X_train.shape)
for i in range(3, len(X_ar)):
  X_train[i-3] = [1, X_ar[i-1], X_ar[i-2], X_ar[i-3]]
print("X train = ", X_train)
print(" shape of X = ", X_train.shape)
Y_train = X_ar[3:]
print("Y train = ", Y_train)
print("shape of Y train = ", Y_train.shape)
```

```
Y for AR =  [ 8007.  6716.  7271. 11434.  9237.  6906.  6877.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X train =  (28, 4)
X train =  [[1.0000e+00 1.9200e+02 9.4000e+01 1.0700e+02]
 [1.0000e+00 2.0000e+00 1.9200e+02 9.4000e+01]
 [1.0000e+00 3.5200e+02 2.0000e+00 1.9200e+02]
 [1.0000e+00 6.1100e+02 3.5200e+02 2.0000e+00]
 [1.0000e+00 1.4600e+03 6.1100e+02 3.5200e+02]
 [1.0000e+00 2.6660e+03 1.4600e+03 6.1100e+02]
 [1.0000e+00 2.6990e+03 2.6660e+03 1.4600e+03]
 [1.0000e+00 3.3240e+03 2.6990e+03 2.6660e+03]
 [1.0000e+00 4.0730e+03 3.3240e+03 2.6990e+03]
 [1.0000e+00 5.0840e+03 4.0730e+03 3.3240e+03]
 [1.0000e+00 4.7970e+03 5.0840e+03 4.0730e+03]
 [1.0000e+00 5.1600e+03 4.7970e+03 5.0840e+03]
 [1.0000e+00 7.0360e+03 5.1600e+03 4.7970e+03]
 [1.0000e+00 6.9990e+03 7.0360e+03 5.1600e+03]
 [1.0000e+00 7.5340e+03 6.9990e+03 7.0360e+03]
 [1.0000e+00 7.2380e+03 7.5340e+03 6.9990e+03]
 [1.0000e+00 7.0150e+03 7.2380e+03 7.5340e+03]
 [1.0000e+00 9.1700e+03 7.0150e+03 7.2380e+03]
 [1.0000e+00 8.1150e+03 9.1700e+03 7.0150e+03]
 [1.0000e+00 8.5580e+03 8.1150e+03 9.1700e+03]
 [1.0000e+00 1.0481e+04 8.5580e+03 8.1150e+03]
 [1.0000e+00 1.0846e+04 1.0481e+04 8.5580e+03]
 [1.0000e+00 9.3270e+03 1.0846e+04 1.0481e+04]
 [1.0000e+00 8.6550e+03 9.3270e+03 1.0846e+04]
 [1.0000e+00 8.0600e+03 8.6550e+03 9.3270e+03]
 [1.0000e+00 1.1186e+04 8.0600e+03 8.6550e+03]
 [1.0000e+00 1.0718e+04 1.1186e+04 8.0600e+03]
 [1.0000e+00 1.0569e+04 1.0718e+04 1.1186e+04]]
 shape of X =  (28, 4)
Y train =  [2.0000e+00 3.5200e+02 6.1100e+02 1.4600e+03 2.6660e+03 2.6990e+03
 3.3240e+03 4.0730e+03 5.0840e+03 4.7970e+03 5.1600e+03 7.0360e+03
 6.9990e+03 7.5340e+03 7.2380e+03 7.0150e+03 9.1700e+03 8.1150e+03
 8.5580e+03 1.0481e+04 1.0846e+04 9.3270e+03 8.6550e+03 8.0600e+03
 1.1186e+04 1.0718e+04 1.0569e+04 8.6780e+03]
shape of Y train =  (28,)
```

Setting variables for #deaths data:

```
pdd = ny_deaths_cap['#Deaths_per_day'].reset_index(drop = True)  # getting the per_da
X_ar_deaths = pdd[:len(pdd)-7]      # using all data except last week's for predictio
X_ar_deaths = np.array([X_ar_deaths]).T
```

```
X_ar_deaths = X_ar_deaths.squeeze()
Y_ar_deaths = pdd[len(pdd)-7:]      # This is the actual Y for last week
Y_ar_deaths = np.array([Y_ar_deaths]).T
Y_ar_deaths = Y_ar_deaths.squeeze()
Y_hat_ar_deaths = np.array([])      # This is the predicted Y_hat for last week
print("Y for AR = ", Y_ar_deaths)
print("shape of Y = ", Y_ar_deaths.shape)
print("initial Y_hat = ", Y_hat_ar_deaths)
print("shape of Y_hat = ", Y_hat_ar_deaths.shape)

# building the training data (y_t; y_t-1, y_t-2, y_t-3)
X_train_deaths = np.zeros([len(X_ar_deaths)-3, 4])
print("X train for #deaths = ", X_train_deaths.shape)
for i in range(3, len(X_ar_deaths)):
  X_train_deaths[i-3] = [1, X_ar_deaths[i-1], X_ar_deaths[i-2], X_ar_deaths[i-3]]
print("X train for #deaths = ", X_train_deaths)
print(" shape of X = ", X_train_deaths.shape)
Y_train_deaths = X_ar_deaths[3:]
print("Y train for #deaths = ", Y_train_deaths)
print("shape of Y train = ", Y_train_deaths.shape)
```

```
Y for AR =  [ 944.  948.  871.  752.  716. 1026. 1005.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X_train for #deaths =  (28, 4)
```

```
def calculate_beta_hat(X_train, Y_train):
  return (np.linalg.inv((X_train.T).dot(X_train))).dot((X_train.T).dot(Y_train))
```

```
def calculate_yhat_ar3(beta_hat, y1, y2, y3):
  return beta_hat[0] + beta_hat[1]*y1 + beta_hat[2]*y2 + beta_hat[3]*y3
```

For #cases

Calculate Y_hat (32|31)

Y_hat (t+1|t) = beta0_hat + beta1_haty*_t + beta2_hat*y_t-1 + beta3_hat*y_t-2

```
beta_hat = calculate_beta_hat(X_train, Y_train)
print("beta_hat = ", beta_hat)
```

```
beta_hat =  [ 9.65553096e+02  7.38815411e-01 -4.38058427e-02  2.21609092e-01]
```

```
Y_hat_ar = np.array([])
Y32_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[27], Y_train[26], Y_train[25])
Y_hat_ar = np.append(Y_hat_ar, Y32_hat_ar)
print("Y32 predicted = ", Y32_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
Y32 predicted =  9289.215531201979
modified Y_hat =  [9289.2155312]
shape of Y_train =  (28,)
```

Calculate Y_hat(33|32) , ..., Y_hat(38|37)

use the next seen data to calculate beta_hat, in turn use the newly calculated beta_hat to predict Y_ha

```
def ar_next_step(i, Y_train, X_train, beta_hat):
  Y_train = np.append(Y_train, Y_ar[i])
  print("new Y_train's shape = ", Y_train.shape)
  X_train = np.vstack([X_train, [1, Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
  print("new X_train's shape = ", X_train.shape)
  beta_hat = calculate_beta_hat(X_train, Y_train)
  print("new beta_hat = ", beta_hat)
  return Y_train, X_train, beta_hat


Y_train, X_train, beta_hat = ar_next_step(0, Y_train, X_train, beta_hat)
Y33_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y33_hat_ar)
```

```
print("Y33 predicted = ", Y33_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (29,)
new X_train's shape =  (29, 4)
new beta_hat =  [ 9.41066794e+02  8.20962640e-01 -9.62351056e-02  1.82060823e-01
Y33 predicted =  8603.587243188023
modified Y_hat =  [9289.2155312  8603.58724319]
```

```
Y_train, X_train, beta_hat = ar_next_step(1, Y_train, X_train, beta_hat)
Y34_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y34_hat_ar)
print("Y34 predicted = ", Y34_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (30,)
new X_train's shape =  (30, 4)
new beta_hat =  [ 8.84710141e+02  8.72581487e-01 -2.68853587e-02  5.15518324e-02
Y34 predicted =  6977.063141063769
modified Y_hat =  [9289.2155312  8603.58724319 6977.06314106]
```

```
Y_train, X_train, beta_hat = ar_next_step(2, Y_train, X_train, beta_hat)
Y35_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y35_hat_ar)
print("Y35 predicted = ", Y35_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (31,)
new X_train's shape =  (31, 4)
new beta_hat =  [ 8.96857385e+02  8.60390483e-01 -2.27516543e-02  5.99506792e-02
Y35 predicted =  7479.981564062033
modified Y_hat =  [9289.2155312  8603.58724319 6977.06314106 7479.98156406]
```

```
Y_train, X_train, beta_hat = ar_next_step(3, Y_train, X_train, beta_hat)
Y36_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y36_hat_ar)
print("Y36 predicted = ", Y36_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (32,)
new X_train's shape =  (32, 4)
new beta_hat =  [ 9.83321395e+02  9.01579877e-01 -2.18138061e-01  2.24266920e-01
Y36 predicted =  11212.080504781905
modified Y_hat =  [ 9289.2155312   8603.58724319  6977.06314106  7479.98156406
 11212.08050478]
```

```
Y_train, X_train, beta_hat = ar_next_step(4, Y_train, X_train, beta_hat)
Y37_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y37_hat_ar)
print("Y37 predicted = ", Y37_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (33,)
new X_train's shape =  (33, 4)
new beta_hat =  [ 1.09397599e+03  7.42047306e-01 -9.42375872e-02  2.41680163e-01
Y37 predicted =  8628.010847488751
modified Y_hat =  [ 9289.2155312   8603.58724319  6977.06314106  7479.98156406
 11212.08050478  8628.01084749]
```

```
Y_train, X_train, beta_hat = ar_next_step(5, Y_train, X_train, beta_hat)
Y38_hat_ar = calculate_yhat_ar3(beta_hat, Y_train[len(Y_train)-1], Y_train[len(Y_trai
Y_hat_ar = np.append(Y_hat_ar, Y38_hat_ar)
print("Y38 predicted = ", Y38_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
print("actual Y = ", Y)
```

```
new Y_train's shape =  (34,)
new X_train's shape =  (34, 4)
new beta_hat =  [ 1.13233897e+03  7.89005081e-01 -2.67439096e-01  3.57935118e-01
Y38 predicted =  8203.503270719939
modified Y_hat =  [ 9289.2155312   8603.58724319  6977.06314106  7479.98156406
 11212.08050478  8628.01084749  8203.50327072]
actual Y =  [ 8007.  6716.  7271. 11434.  9237.  6906.  6877.]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with AR(3) = ", calculate_MAPE(Y_ar,Y_hat_ar))
print("MSE with AR(3) = ", calculate_mse(Y_ar,Y_hat_ar))
```

```
MAPE with AR(3) =  21.19279790390568
MSE with AR(3) =  4221942.603478504
```

For #deaths

Calculate Y_hat (32|31)

Y_hat (t+1|t) = beta0_hat + beta1_haty_t + beta2_hat*y_t-1 + beta3_hat*y_t-2

```
beta_hat_deaths = calculate_beta_hat(X_train_deaths, Y_train_deaths)
print("beta_hat for #deaths = ", beta_hat_deaths)
```

```
beta_hat for #deaths =  [38.2223336   1.3391797  -0.76243656  0.41692611]
```

```
Y_hat_ar_deaths = np.array([])
Y32_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths, Y_train_deaths[27], Y_train_d
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y32_hat_ar_deaths)
print("Y32 predicted = ", Y32_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

Y32 predicted = 997.1530224759157

Calculate Y_hat(33|32) , ..., Y_hat(38|37)

use the next seen data to calculate beta_hat, in turn use the newly calculated beta_hat to predict Y_ha

```
def ar_next_step(i, Y_train, X_train, beta_hat, Y_ar):
  Y_train = np.append(Y_train, Y_ar[i])
  print("new Y_train's shape = ", Y_train.shape)
  X_train = np.vstack([X_train, [1, Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
  print("new X_train's shape = ", X_train.shape)
  beta_hat = calculate_beta_hat(X_train, Y_train)
  print("new beta_hat = ", beta_hat)
  return Y_train, X_train, beta_hat


Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(0, Y_train_deaths, X_t
Y33_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y33_hat_ar_deaths)
print("Y33 predicted = ", Y33_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (29,)
new X_train's shape =  (29, 4)
new beta_hat =  [38.86385783  1.31586971 -0.69878519  0.36876448]
Y33 predicted =  961.9979148707727
modified Y_hat =  [997.15302248 961.99791487]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(1, Y_train_deaths, X_t
Y34_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y34_hat_ar_deaths)
print("Y34 predicted = ", Y34_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (30,)
new X_train's shape =  (30, 4)
new beta_hat =  [39.14602967  1.31425809 -0.69974808  0.3697322 ]
Y34 predicted =  954.6713626805597
modified Y_hat =  [997.15302248 961.99791487 954.67136268]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(2, Y_train_deaths, X_t
                                                              beta_hat_deaths, Y_ar_
Y35_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y35_hat_ar_deaths)
```

```
print("Y35 predicted = ", Y35_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (31,)
new X_train's shape =  (31, 4)
new beta_hat =  [40.40157077  1.32217859 -0.72026973  0.37287496]
Y35 predicted =  861.1973768138724
modified Y_hat =  [997.15302248 961.99791487 954.67136268 861.19737681]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(3, Y_train_deaths, X_t
                                                               beta_hat_deaths, Y_ar_
Y36_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y36_hat_ar_deaths)
print("Y36 predicted = ", Y36_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (32,)
new X_train's shape =  (32, 4)
new beta_hat =  [40.34610037  1.36568039 -0.76441253  0.36275403]
Y36 predicted =  745.4252568011416
modified Y_hat =  [997.15302248 961.99791487 954.67136268 861.19737681 745.42525
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(4, Y_train_deaths, X_t
                                                               beta_hat_deaths, Y_ar_
Y37_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y37_hat_ar_deaths)
print("Y37 predicted = ", Y37_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (33,)
new X_train's shape =  (33, 4)
new beta_hat =  [39.89544793  1.37835372 -0.76818556  0.35164284]
Y37 predicted =  755.4020832206029
modified Y_hat =  [997.15302248 961.99791487 954.67136268 861.19737681 745.42525
 755.40208322]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(5, Y_train_deaths, X_t
                                                               beta_hat_deaths, Y_ar_
Y38_hat_ar_deaths = calculate_yhat_ar3(beta_hat_deaths,
                                       Y_train_deaths[len(Y_train_deaths)-1],
                                       Y_train_deaths[len(Y_train_deaths)-2],
                                       Y_train_deaths[len(Y_train_deaths)-3])
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y38_hat_ar_deaths)
print("Y38 predicted = ", Y38_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
print("actual Y = ", Y_deaths)
```

```
new Y_train's shape =  (34,)
new X_train's shape =  (34, 4)
new beta_hat =  [43.22440802  1.38041002 -0.91783486  0.51671735]
Y38 predicted =  1190.926776307452
modified Y_hat =  [ 997.15302248  961.99791487  954.67136268  861.19737681  745.4
  755.40208322 1190.92677631]
actual Y =  [ 944.  948.  871.  752.  716. 1026. 1005.]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with AR(3) = ", calculate_MAPE(Y_ar_deaths,Y_hat_ar_deaths))
print("MSE with AR(3) = ", calculate_mse(Y_ar_deaths,Y_hat_ar_deaths))
```

```
MAPE with AR(3) =  11.459769931730618
MSE with AR(3) =  18657.713415164446
```

AR(5)

Observation: We could see from below execution that MSE and MAPE were high hence inferring that

Setting variables for #cases data:

```
pdc = ny_cases_cap['#Cases_per_day'].reset_index(drop = True)  # getting the per_day_
X_ar = pdc[:len(pdc)-7]      # using all data except last week's for prediction
X_ar = np.array([X_ar]).T
X_ar = X_ar.squeeze()
Y_ar = pdc[len(pdc)-7:]       # This is the actual Y for last week
Y_ar = np.array([Y_ar]).T
Y_ar = Y_ar.squeeze()
Y_hat_ar = np.array([])       # This is the predicted Y_hat for last week
print("Y for AR = ", Y_ar)
print("shape of Y = ", Y_ar.shape)
print("initial Y_hat = ", Y_hat_ar)
print("shape of Y_hat = ", Y_hat_ar.shape)

# building the training data (y_t; y_t-1, y_t-2, y_t-3, y_t-4, y_t-5)
X_train = np.zeros([len(X_ar)-5, 6])
print("X train = ", X_train.shape)
for i in range(5, len(X_ar)):
  X_train[i-5] = [1, X_ar[i-1], X_ar[i-2], X_ar[i-3], X_ar[i-4], X_ar[i-5]]
print("X train = ", X_train)
print(" shape of X = ", X_train.shape)
Y_train = X_ar[5:]
print("Y train = ", Y_train)
print("shape of Y train = ", Y_train.shape)
```

```
Y for AR =  [ 8007.  6716.  7271. 11434.  9237.  6906.  6877.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X train =  (26, 6)
X train =  [[1.0000e+00 3.5200e+02 2.0000e+00 1.9200e+02 9.4000e+01 1.0700e+02]
 [1.0000e+00 6.1100e+02 3.5200e+02 2.0000e+00 1.9200e+02 9.4000e+01]
 [1.0000e+00 1.4600e+03 6.1100e+02 3.5200e+02 2.0000e+00 1.9200e+02]
 [1.0000e+00 2.6660e+03 1.4600e+03 6.1100e+02 3.5200e+02 2.0000e+00]
 [1.0000e+00 2.6990e+03 2.6660e+03 1.4600e+03 6.1100e+02 3.5200e+02]
 [1.0000e+00 3.3240e+03 2.6990e+03 2.6660e+03 1.4600e+03 6.1100e+02]
 [1.0000e+00 4.0730e+03 3.3240e+03 2.6990e+03 2.6660e+03 1.4600e+03]
 [1.0000e+00 5.0840e+03 4.0730e+03 3.3240e+03 2.6990e+03 2.6660e+03]
 [1.0000e+00 4.7970e+03 5.0840e+03 4.0730e+03 3.3240e+03 2.6990e+03]
 [1.0000e+00 5.1600e+03 4.7970e+03 5.0840e+03 4.0730e+03 3.3240e+03]
 [1.0000e+00 7.0360e+03 5.1600e+03 4.7970e+03 5.0840e+03 4.0730e+03]
 [1.0000e+00 6.9990e+03 7.0360e+03 5.1600e+03 4.7970e+03 5.0840e+03]
 [1.0000e+00 7.5340e+03 6.9990e+03 7.0360e+03 5.1600e+03 4.7970e+03]
 [1.0000e+00 7.2380e+03 7.5340e+03 6.9990e+03 7.0360e+03 5.1600e+03]
 [1.0000e+00 7.0150e+03 7.2380e+03 7.5340e+03 6.9990e+03 7.0360e+03]
 [1.0000e+00 9.1700e+03 7.0150e+03 7.2380e+03 7.5340e+03 6.9990e+03]
 [1.0000e+00 8.1150e+03 9.1700e+03 7.0150e+03 7.2380e+03 7.5340e+03]
 [1.0000e+00 8.5580e+03 8.1150e+03 9.1700e+03 7.0150e+03 7.2380e+03]
 [1.0000e+00 1.0481e+04 8.5580e+03 8.1150e+03 9.1700e+03 7.0150e+03]
 [1.0000e+00 1.0846e+04 1.0481e+04 8.5580e+03 8.1150e+03 9.1700e+03]
 [1.0000e+00 9.3270e+03 1.0846e+04 1.0481e+04 8.5580e+03 8.1150e+03]
 [1.0000e+00 8.6550e+03 9.3270e+03 1.0846e+04 1.0481e+04 8.5580e+03]
 [1.0000e+00 8.0600e+03 8.6550e+03 9.3270e+03 1.0846e+04 1.0481e+04]
 [1.0000e+00 1.1186e+04 8.0600e+03 8.6550e+03 9.3270e+03 1.0846e+04]
 [1.0000e+00 1.0718e+04 1.1186e+04 8.0600e+03 8.6550e+03 9.3270e+03]
 [1.0000e+00 1.0569e+04 1.0718e+04 1.1186e+04 8.0600e+03 8.6550e+03]]
 shape of X =  (26, 6)
Y train =  [  611.  1460.  2666.  2699.  3324.  4073.  5084.  4797.  5160.  7036.
  6999.  7534.  7238.  7015.  9170.  8115.  8558. 10481. 10846.  9327.
  8655.  8060. 11186. 10718. 10569.  8678.]
shape of Y train =  (26,)
```

Setting variables for #deaths data:

```
pdd = ny_deaths_cap['#Deaths_per_day'].reset_index(drop = True)  # getting the per_da
X_ar_deaths = pdd[:len(pdd)-7]      # using all data except last week's for predictio
X_ar_deaths = np.array([X_ar_deaths]).T
X_ar_deaths = X_ar_deaths.squeeze()
Y_ar_deaths = pdd[len(pdd)-7:]      # This is the actual Y for last week
Y_ar_deaths = np.array([Y_ar_deaths]).T
Y_ar_deaths = Y_ar_deaths.squeeze()
Y_hat_ar_deaths = np.array([])      # This is the predicted Y_hat for last week
print("Y for AR = ", Y_ar_deaths)
print("shape of Y = ", Y_ar_deaths.shape)
print("initial Y_hat = ", Y_hat_ar_deaths)
print("shape of Y_hat = ", Y_hat_ar_deaths.shape)

# building the training data (y_t; y_t-1, y_t-2, y_t-3, y_t-4, y_t-5)
X_train_deaths = np.zeros([len(X_ar_deaths)-5, 6])
print("X train = ", X_train_deaths.shape)
```

```
for i in range(5, len(X_ar_deaths)):
  X_train_deaths[i-5] = [1, X_ar_deaths[i-1], X_ar_deaths[i-2],
                X_ar_deaths[i-3], X_ar_deaths[i-4], X_ar_deaths[i-5]]
print("X train = ", X_train_deaths)
print(" shape of X = ", X_train_deaths.shape)
Y_train_deaths = X_ar_deaths[5:]
print("Y train = ", Y_train_deaths)
print("shape of Y train = ", Y_train_deaths.shape)
```

```
Y for AR =  [ 944.  948.  871.  752.  716. 1026. 1005.]
shape of Y =  (7,)
initial Y_hat =  []
shape of Y_hat =  (0,)
X train =  (26, 6)
X train =  [[1.000e+00 1.200e+01 6.000e+00 4.000e+00 1.000e+00 1.000e+00]
 [1.000e+00 1.400e+01 1.200e+01 6.000e+00 4.000e+00 1.000e+00]
 [1.000e+00 2.500e+01 1.400e+01 1.200e+01 6.000e+00 4.000e+00]
 [1.000e+00 3.300e+01 2.500e+01 1.400e+01 1.200e+01 6.000e+00]
 [1.000e+00 5.500e+01 3.300e+01 2.500e+01 1.400e+01 1.200e+01]
 [1.000e+00 4.400e+01 5.500e+01 3.300e+01 2.500e+01 1.400e+01]
 [1.000e+00 9.100e+01 4.400e+01 5.500e+01 3.300e+01 2.500e+01]
 [1.000e+00 1.010e+02 9.100e+01 4.400e+01 5.500e+01 3.300e+01]
 [1.000e+00 1.250e+02 1.010e+02 9.100e+01 4.400e+01 5.500e+01]
 [1.000e+00 1.470e+02 1.250e+02 1.010e+02 9.100e+01 4.400e+01]
 [1.000e+00 2.430e+02 1.470e+02 1.250e+02 1.010e+02 9.100e+01]
 [1.000e+00 3.090e+02 2.430e+02 1.470e+02 1.250e+02 1.010e+02]
 [1.000e+00 3.770e+02 3.090e+02 2.430e+02 1.470e+02 1.250e+02]
 [1.000e+00 4.280e+02 3.770e+02 3.090e+02 2.430e+02 1.470e+02]
 [1.000e+00 5.400e+02 4.280e+02 3.770e+02 3.090e+02 2.430e+02]
 [1.000e+00 6.510e+02 5.400e+02 4.280e+02 3.770e+02 3.090e+02]
 [1.000e+00 7.100e+02 6.510e+02 5.400e+02 4.280e+02 3.770e+02]
 [1.000e+00 8.130e+02 7.100e+02 6.510e+02 5.400e+02 4.280e+02]
 [1.000e+00 6.880e+02 8.130e+02 7.100e+02 6.510e+02 5.400e+02]
 [1.000e+00 5.730e+02 6.880e+02 8.130e+02 7.100e+02 6.510e+02]
 [1.000e+00 7.940e+02 5.730e+02 6.880e+02 8.130e+02 7.100e+02]
 [1.000e+00 1.024e+03 7.940e+02 5.730e+02 6.880e+02 8.130e+02]
 [1.000e+00 1.077e+03 1.024e+03 7.940e+02 5.730e+02 6.880e+02]
 [1.000e+00 9.480e+02 1.077e+03 1.024e+03 7.940e+02 5.730e+02]
 [1.000e+00 9.440e+02 9.480e+02 1.077e+03 1.024e+03 7.940e+02]
 [1.000e+00 8.270e+02 9.440e+02 9.480e+02 1.077e+03 1.024e+03]]
 shape of X =  (26, 6)
Y train =  [  14.   25.   33.   55.   44.   91.  101.  125.  147.  243.  309.  3
  428.  540.  651.  710.  813.  688.  573.  794. 1024. 1077.  948.  944.
  827.  893.]
shape of Y train =  (26,)
```

```
def calculate_beta_hat(X_train, Y_train):
  return (np.linalg.inv((X_train.T).dot(X_train))).dot((X_train.T).dot(Y_train))
```

```
def calculate_yhat_ar5(beta_hat, y1, y2, y3, y4, y5):
  return beta_hat[0] + beta_hat[1]*y1 + beta_hat[2]*y2 + beta_hat[3]*y3 + beta_hat[4]
```

For #cases

Calculate Y_hat (32|31)

Y_hat (t+1|t) = beta0_hat + beta1_hat*y_t + beta2_hat*y_t-1 + beta3_hat*y_t-2

```
beta_hat = calculate_beta_hat(X_train, Y_train)
print("beta_hat = ", beta_hat)
```

```
beta_hat =  [ 1.78752885e+03  4.64502450e-01 -3.64861849e-02 -8.01773065e-02
  3.53347090e-01  1.70691993e-01]
```

```
Y32_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y32_hat_ar)
print("Y32 predicted = ", Y32_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
Y32 predicted =  9901.83626790165
modified Y_hat =  [9901.8362679]
```

Calculate Y_hat(33|32) , ..., Y_hat(38|37)

use the next seen data to calculate beta_hat, in turn use the newly calculated beta_hat to predict Y_ha

```
def ar_next_step(i, Y_train, X_train, beta_hat):
  Y_train = np.append(Y_train, Y_ar[i])
  print("new Y_train's shape = ", Y_train.shape)
  X_train = np.vstack([X_train,
                       [1, Y_train[len(Y_train)-2],
                        Y_train[len(Y_train)-3], Y_train[len(Y_train)-4],
                        Y_train[len(Y_train)-5], Y_train[len(Y_train)-6]]])
  print("new X_train's shape = ", X_train.shape)
  beta_hat = calculate_beta_hat(X_train, Y_train)
  print("new beta_hat = ", beta_hat)
  return Y_train, X_train, beta_hat


Y_train, X_train, beta_hat = ar_next_step(0, Y_train, X_train, beta_hat)
Y33_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y33_hat_ar)
print("Y33 predicted = ", Y33_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (27,)
```

```
Y_train, X_train, beta_hat = ar_next_step(1, Y_train, X_train, beta_hat)
Y34_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y34_hat_ar)
print("Y33 predicted = ", Y34_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (28,)
new X_train's shape =  (28, 6)
new beta_hat =  [ 1.41091705e+03  7.45621136e-01 -1.50484812e-02 -1.54193832e-01
  2.02879362e-01  7.46323969e-02]
Y33 predicted =  7904.063336374601
modified Y_hat =  [9901.8362679  9877.71537361 7904.06333637]
```

```
Y_train, X_train, beta_hat = ar_next_step(2, Y_train, X_train, beta_hat)
Y35_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y35_hat_ar)
print("Y33 predicted = ", Y35_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (29,)
new X_train's shape =  (29, 6)
new beta_hat =  [ 1.33434300e+03  7.83415909e-01 -2.38692984e-02 -1.30441383e-01
  1.84436972e-01  4.21812457e-02]
Y33 predicted =  7872.167339866079
modified Y_hat =  [9901.8362679  9877.71537361 7904.06333637 7872.16733987]
```

```
Y_train, X_train, beta_hat = ar_next_step(3, Y_train, X_train, beta_hat)
Y36_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y36_hat_ar)
print("Y33 predicted = ", Y36_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (30,)
new X_train's shape =  (30, 6)
new beta_hat =  [ 1.62435070e+03  7.49812961e-01 -1.60486133e-01 -8.55012369e-02
  4.48868029e-02  3.03975924e-01]
Y33 predicted =  11453.902802605091
modified Y_hat =  [ 9901.8362679   9877.71537361  7904.06333637  7872.16733987
 11453.90280261]
```

```
Y_train, X_train, beta_hat = ar_next_step(4, Y_train, X_train, beta_hat)
Y37_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                                Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                                Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
```

```
Y_hat_ar = np.append(Y_hat_ar, Y37_hat_ar)
print("Y33 predicted = ", Y37_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (31,)
new X_train's shape =  (31, 6)
new beta_hat =  [ 1.76982522e+03  5.75245169e-01 -2.93519609e-02 -1.47650314e-02
  3.22395164e-04  2.96526637e-01]
Y33 predicted =  9016.851965628057
modified Y_hat =  [ 9901.8362679   9877.71537361  7904.06333637  7872.16733987
 11453.90280261  9016.85196563]
```

```
Y_train, X_train, beta_hat = ar_next_step(5, Y_train, X_train, beta_hat)
Y38_hat_ar = calculate_yhat_ar5(beta_hat, Y_train[len(Y_train)-1],
                               Y_train[len(Y_train)-2], Y_train[len(Y_train)-3],
                               Y_train[len(Y_train)-4], Y_train[len(Y_train)-5])
Y_hat_ar = np.append(Y_hat_ar, Y38_hat_ar)
print("Y33 predicted = ", Y38_hat_ar)
print("modified Y_hat = ", Y_hat_ar)
```

```
new Y_train's shape =  (32,)
new X_train's shape =  (32, 6)
new beta_hat =  [ 1.79031454e+03  6.43073374e-01 -2.38937255e-01  1.15832781e-01
  6.89405882e-02  2.27951291e-01]
Y33 predicted =  7380.935741484208
modified Y_hat =  [ 9901.8362679   9877.71537361  7904.06333637  7872.16733987
 11453.90280261  9016.85196563  7380.93574148]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with AR(3) = ", calculate_MAPE(Y_ar,Y_hat_ar))
print("MSE with AR(3) = ", calculate_mse(Y_ar,Y_hat_ar))
```

```
MAPE with AR(3) =  24.64194297763653
MSE with AR(3) =  5185510.708728259
```

For #deaths

Calculate Y_hat (32|31)

Y_hat (t+1|t) = beta0_hat + beta1_haty*_t + beta2_hat*y_t-1 + beta3_hat*y_t-2

```
beta_hat_deaths = calculate_beta_hat(X_train_deaths, Y_train_deaths)
print("beta_hat = ", beta_hat_deaths)
```

```
Y32_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                               Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                               Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y32_hat_ar_deaths)
print("Y32 predicted = ", Y32_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
Y32 predicted =  1037.4769106860795
modified Y_hat =  [1037.47691069]
```

Calculate Y_hat(33|32) , ..., Y_hat(38|37)

use the next seen data to calculate beta_hat, in turn use the newly calculated beta_hat to predict Y_ha

```
def ar_next_step(i, Y_train, X_train, beta_hat, Y_ar):
  Y_train = np.append(Y_train, Y_ar[i])
  print("new Y_train's shape = ", Y_train.shape)
  X_train = np.vstack([X_train,
                     [1, Y_train[len(Y_train)-2],
                      Y_train[len(Y_train)-3], Y_train[len(Y_train)-4],
                      Y_train[len(Y_train)-5], Y_train[len(Y_train)-6]]])
  print("new X_train's shape = ", X_train.shape)
  beta_hat = calculate_beta_hat(X_train, Y_train)
  print("new beta_hat = ", beta_hat)
  return Y_train, X_train, beta_hat
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(0, Y_train_deaths, X_t
                                                                 beta_hat_deaths, Y_ar_
Y33_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                               Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                               Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y33_hat_ar_deaths)
print("Y33 predicted = ", Y33_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (27,)
new X_train's shape =  (27, 6)
new beta_hat =  [ 4.89789199e+01  1.23328392e+00 -5.51120604e-01  9.84381412e-02
  2.15947492e-01 -1.06850121e-02]
Y33 predicted =  996.1816204139302
modified Y_hat =  [1037.47691069  996.18162041]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(1, Y_train_deaths, X_t
                                                                 beta_hat_deaths, Y_ar_
Y34_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                               Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                               Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y34_hat_ar_deaths)
print("Y34 predicted = ", Y34_hat_ar_deaths)
```

```
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (28,)
new X_train's shape =  (28, 6)
new beta_hat =  [ 4.90618482e+01  1.24537943e+00 -5.87438182e-01  1.63069811e-01
  1.70663089e-01 -1.28365769e-02]
Y34 predicted =  949.781894453004
modified Y_hat =  [1037.47691069  996.18162041  949.78189445]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(2, Y_train_deaths, X_t
                                                                beta_hat_deaths, Y_ar_
Y35_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                              Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                              Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y35_hat_ar_deaths)
print("Y35 predicted = ", Y35_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (29,)
new X_train's shape =  (29, 6)
new beta_hat =  [48.45574975  1.26319106 -0.5905233   0.11380732  0.27530227 -0.(
Y35 predicted =  864.4417080194598
modified Y_hat =  [1037.47691069  996.18162041  949.78189445  864.44170802]
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(3, Y_train_deaths, X_t
                                                                beta_hat_deaths, Y_ar_
Y36_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                              Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                              Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y36_hat_ar_deaths)
print("Y36 predicted = ", Y36_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (30,)
new X_train's shape =  (30, 6)
new beta_hat =  [48.02106947  1.3136075  -0.6453471   0.12035455  0.26479763 -0.(
Y36 predicted =  751.1070658253444
modified Y_hat =  [1037.47691069  996.18162041  949.78189445  864.44170802  751.:
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(4, Y_train_deaths, X_t
                                                                beta_hat_deaths, Y_ar_
Y37_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                              Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                              Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y37_hat_ar_deaths)
print("Y37 predicted = ", Y37_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
```

```
new Y_train's shape =  (31,)
```

```
Y_train_deaths, X_train_deaths, beta_hat_deaths = ar_next_step(5, Y_train_deaths, X_t
                                                              beta_hat_deaths, Y_ar_
Y38_hat_ar_deaths = calculate_yhat_ar5(beta_hat_deaths, Y_train_deaths[len(Y_train_de
                              Y_train_deaths[len(Y_train_deaths)-2], Y_train_deaths[
                              Y_train_deaths[len(Y_train_deaths)-4], Y_train_deaths[
Y_hat_ar_deaths = np.append(Y_hat_ar_deaths, Y38_hat_ar_deaths)
print("Y38 predicted = ", Y38_hat_ar_deaths)
print("modified Y_hat = ", Y_hat_ar_deaths)
print("actual Y = ", Y_deaths)
```

```
new Y_train's shape =  (32,)
new X_train's shape =  (32, 6)
new beta_hat =  [ 5.48926796e+01  1.28665939e+00 -7.32962537e-01  1.69532447e-01
  2.72705917e-01 -2.32837735e-02]
Y38 predicted =  1193.1462735512414
modified Y_hat =  [1037.47691069  996.18162041  949.78189445  864.44170802  751.
  761.51356915 1193.14627355]
actual Y =  [ 944.  948.  871.  752.  716. 1026. 1005.]
```

Calculate MAPE and MSE for EWMA(alpha=0.8)

```
print("MAPE with AR(3) = ", calculate_MAPE(Y_ar_deaths,Y_hat_ar_deaths))
print("MSE with AR(3) = ", calculate_mse(Y_ar_deaths,Y_hat_ar_deaths))
```

```
MAPE with AR(3) =  12.626380176112054
MSE with AR(3) =  19499.103485849348
```

Required Inference 2

Apply the Wald's test, Z-test, and t-test (assume all are applicable) to check whether the mean of COV
from the second-last week to the last week in your dataset. Use MLE for Wald's test as the estimator;
that daily data is Poisson distributed. Note, you have to report results for deaths and #cases separate
After running the test and reporting the numbers, check and comment on whether the tests are applic
computing the mean of the second-last week data and using that as guess for last week data. Then, r
and t-tests. For t-test, use both paired and unpaired tests. Use alpha value of 0.05 for all. For t-test, th
alpha/2 for two-tailed, where n is the number of data points. You can find these values in online t tabl
sample standard deviation of the entire covid19 dataset you have and use that as the true sigma valu

One sample test

Apply the Wald's test, Z-test, and t-test (assume all are applicable) to check whether the mean of COV
from the second-last week to the last week in your dataset. Use MLE for Wald's test as the estimator;
that daily data is Poisson distributed. Note, you have to report results for deaths and #cases separat
After running the test and reporting the numbers, check and comment on whether the tests are applic
computing the mean of the second-last week data and using that as guess for last week data. Then, r
and t-tests. For t-test, use both paired and unpaired tests. Use alpha value of 0.05 for all. For t-test, th
alpha/2 for two-tailed, where n is the number of data points. You can find these values in online t tabl
sample standard deviation of the entire covid19 dataset you have and use that as the true sigma valu

One sample wald's test

```
Z_alphaBy2 = 1.96

def walds_test(second_last_week, last_week):
  # Given that second last week's data is poisson distributed
  # and we have to use MLE as the estimator, theta_knot hence becomes sample mean
  theta_knot = np.mean(second_last_week)
  print("theta_knot: ", theta_knot)

  # finding the sample mean theta hat of current distribution
  theta_hat = np.mean(last_week)
  print("theta_hat: ", theta_hat)
  X = last_week
  # sample se theta hat for poisson distribution
  se_hat_theta_hat = np.sqrt(theta_hat/len(last_week))
  print("se_hat_theta_hat: ", se_hat_theta_hat)

  w = (theta_hat - theta_knot)/se_hat_theta_hat
  print("w: ", w)

  if np.absolute(w) > Z_alphaBy2:
    print("reject H0")
  else:
    print("accept H0")
```

Death

```
second_last_week_death = ny_deaths_cap[-14:-7]
# print(second_last_week_death)
last_week_death = ny_deaths_cap[-7:]
# print(last_week_death)

walds_test(second_last_week_death['#Deaths per day'], last_week_death ['#Deaths per d
```

```
theta_knot:  929.5714285714286
theta_hat:  894.5714285714286
se_hat_theta_hat:  11.304685681935034
w:  -3.0960613134012425
reject H0
```

Cases

```
second_last_week_cases = ny_cases_cap[-14:-7]
# print(second_last_week_cases)
last_week_cases = ny_cases_cap[-7:]
# print(last_week_cases)

walds_test(second_last_week_cases['#Cases_per_day'], last_week_cases['#Cases_per_day'
```

```
theta_knot:  9599.0
theta_hat:  8064.0
se_hat_theta_hat:  33.94112549695428
w:  -45.22537121338961
reject H0
```

Applicable/Inference

H0: check whether the mean of COVID19 #deaths and #cases are different from the second-last wee

Cases: reject H0. It's goes with the data since the number are changing week by week very drastically isn't same as that of last week.

Death: reject H0. It's goes with the data since the number are changing week by week very drastically isn't same as that of last week.

Assumption for applying wald's test is theta hat is that the estimator is asymptotic normal, which imp that's not the case, it's just 7 days and hence we can't apply the test here reliably.

One sample Z test

```
import math
def one_sample_z_test(second_last_week_data, last_week_data, known_std_dev):
  n = len(last_week_data) # is this correct?
  mu_knot = np.mean(second_last_week_data)
  x_bar = np.mean(last_week_data)
  z = (x_bar - mu_knot)/(known_std_dev/math.sqrt(n))
  print(z)
  if abs(z) > Z_alphaBy2:
    print("reject H0")
  else:
```

```
    print("accept H0")
```

Death

```
known_std_dev = np.std(ny_deaths_cap['#Deaths_per_day'])
one_sample_z_test(second_last_week_death['#Deaths_per_day'],last_week_death['#Deaths_
```

```
-0.23779008325909173
accept H0
```

Cases

```
known_std_dev = np.std(ny_cases_cap['#Cases_per_day'])
one_sample_z_test(second_last_week_cases['#Cases_per_day'],last_week_cases['#Cases_pe
```

```
-1.150596324816142
accept H0
```

Applicable/Observation

Assumptions:

1. sigma is known
2. data is normally distributed or n tends to infinity

Death: accept H0 Cases: accept H0

The test isn't applicable since the assumptions are not getting satisfied. for known sigma we have as
While for the distribution of data is not known to be normal and our n isn't tending towards infinity eitl

T test

One sample

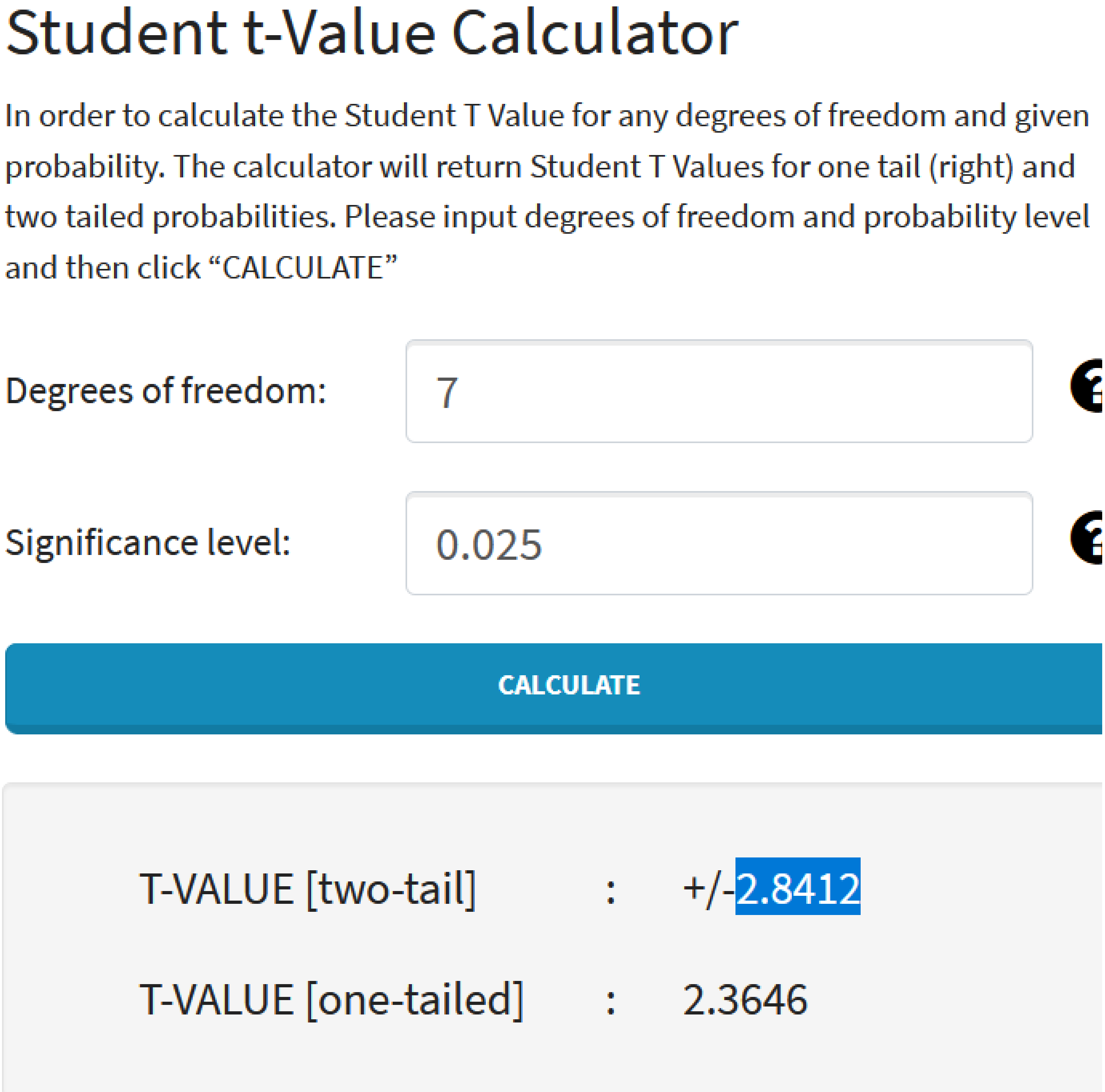
```
## Assumptions
## 1. {X1,...,Xn} ~ Nor(mu, sigma^2)
## useful when: sigma need not be known ; when n is small

# for deaths
X = second_last_week_death['#Deaths_per_day']
mu0 = np.mean(last_week_death['#Deaths_per_day'])
t_thresh = 1.96

s = np.sqrt(calculate_sampleVar(X))
t = (np.mean(X) - mu0)/(s*np.sqrt(len(X)))
```

```
print("t = ", t)
print("mod t = ", np.absolute(t))

if np.absolute(t) > t_thresh:
  print("reject H0")
else:
  print("accept H0")
```

```
t =  0.14120514443869592
mod t =  0.14120514443869592
accept H0
```

```
## Assumptions
## 1. {X1,...,Xn} ~ Nor(mu, sigma^2)
## useful when: sigma need not be known ; when n is small

# for cases
X = last_week_cases['#Cases_per_day']
mu0 = np.mean(second_last_week_cases['#Cases_per_day'])
t_thresh = 1.96

s = np.sqrt(calculate_sampleVar(X))
t = (np.mean(X) - mu0)/(s*np.sqrt(len(X)))

print("t = ", t)
print("mod t = ", np.absolute(t))

if np.absolute(t) > t_thresh:
  print("reject H0")
else:
  print("accept H0")
```

```
t =  -0.3624886816119439
mod t =  0.3624886816119439
accept H0
```



Paired

```
def paired_t_test(last_week_data, second_last_week_data,t_threshold):
  d = last_week_data.values - second_last_week_data.values
  d_bar = np.mean(d)
  T = (d_bar-0)/(np.std(d)/math.sqrt(len(d)))
  print("mod T: ", abs(T))
  if abs(T) > t_threshold:
      print("reject H0")
  else:
```

```
        print("accept H0")
```

Cases

```
t_alpha_by_2 = 2.8412
paired_t_test(last_week_cases['#Cases_per_day'], second_last_week_cases['#Cases_per_d
```

```
mod T:  3.6753353898034193
reject H0
```

Applicable/Observation:

Cases:

Death:

Death

```
paired_t_test(last_week_death['#Deaths_per_day'], second_last_week_death['#Deaths_per
```

```
mod T:  0.5415346603446712
accept H0
```

Unpaired

```
def unpaired_t_test(last_week_data, second_last_week_data):
  d = last_week_data.values - second_last_week_data.values
  d_bar = np.mean(d)
  sample_std_dev_last_week = np.std(last_week_data)
  sample_std_dev_second_last_week = np.std(second_last_week_data)
  pooled_std_dev = math.sqrt((sample_std_dev_last_week*sample_std_dev_last_week)/len(
                           sample_std_dev_second_last_week*sample_std_dev_second_la
  T = (d_bar-0)/pooled_std_dev
  print("mod T: ", abs(T))
  if abs(T) > t_alpha_by_2:
      print("reject H0")
  else:
      print("accept H0")
```

Cases

```
unpaired_t_test(last_week_cases['#Cases_per_day'], second_last_week_cases['#Cases_per
```

```
mod T:  2.0745920119311894
accept H0
```

Applicable or not?

Death

```
unpaired_t_test(last_week_death['#Deaths_per_day'], second_last_week_death['#Deaths_p
```

```
mod T:  0.6347631079116565
accept H0
```

Applicable?

Even though it satisfies the primary criteria of applicability of t-test i.e. the number of data samples ar
normally distributed or not. Hence can't apply the test reliably.

Two sample wald's test

Cases

```
Z_alphaBy2 = 1.96

X1 = last_week_cases['#Cases_per_day']
X2 = second_last_week_cases['#Cases_per_day']

p1_hat = np.mean(X1)
p2_hat = np.mean(X2)

delta_hat = p1_hat - p2_hat
se_hat_delta = np.sqrt((p1_hat * (1-p1_hat))/len(X1) + (p2_hat *(1-p2_hat))/len(X2))
w_delta = delta_hat/se_hat_delta

print("p1_mle = ", p1_hat)
print("p2_mle = ", p2_hat)
print("delta_hat = ", delta_hat)
print("se_hat_delta = ", se_hat_delta)
print("w = ", w_delta)
print("mod w = ", np.absolute(w_delta))

if np.absolute(w_delta) > Z_alphaBy2:
  print("reject H0")
else:
  print("accept H0")
```

```
p1_mle =  8064.0
p2_mle =  9599.0
delta_hat =  -1535.0
se_hat_delta =  nan
w =  nan
```

Death

```
Z_alphaBy2 = 1.96

X1 = last_week_death['#Deaths_per_day']
X2 = second_last_week_death['#Deaths_per_day']

p1_hat = np.mean(X1)
p2_hat = np.mean(X2)

delta_hat = p1_hat - p2_hat
se_hat_delta = np.sqrt((p1_hat * (1-p1_hat))/len(X1) + (p2_hat *(1-p2_hat))/len(X2))
w_delta = delta_hat/se_hat_delta

print("p1_mle = ", p1_hat)
print("p2_mle = ", p2_hat)
print("delta_hat = ", delta_hat)
print("se_hat_delta = ", se_hat_delta)
print("w = ", w_delta)
print("mod w = ", np.absolute(w_delta))

if np.absolute(w_delta) > Z_alphaBy2:
  print("reject H0")
else:
  print("accept H0")
```

```
p1_mle =  894.5714285714286
p2_mle =  929.5714285714286
delta_hat =  -35.0
se_hat_delta =  nan
w =  nan
mod w =  nan
accept H0
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: RuntimeWarning:
  # Remove the CWD from sys.path while we load stuff.
```

Applicable

Assumption is that both the estimators for X and Y, the mu_x and mu_y should be asymptotic normal commit this assumption. Hence not applicable.

Required Inference 3

Repeat inference 2 above but for equality of distributions (distribution of second-last week and last w
For the K-S test, use both 1-sample and 2-sample tests. For the 1-sample test, try Poisson, Geometric
these distributions to check against in 1-sample KS, use MME on second last week's data to obtain p
check whether the last week's data has the distribution with the obtained MME parameters. Use a thr
Permutation test.

One sample KS test with Poisson distribution

```
import math
from scipy.stats import poisson

def frequency_count(data):
  # Creating an empty dictionary
  freq = {}
  for item in data:
    if (item in freq):
      freq[item] += 1
    else:
      freq[item] = 1
  return freq

# week 1 data is the second last week's data used to obtain parameters of the distrib
# week 2 data is against what we will check against
def one_sample_ks_test_with_poisson_distribution(second_last_week, last_week, thresho
  # for poisson distribution, mme estimate of lambda is sample mean
  lambda_mme = np.mean(second_last_week)
  last_week = np.sort(last_week)
  max_diff = float("-inf")
  step_size = 1/len(last_week)
  frequency_map = frequency_count(last_week)
  f_x_x_neg = 0
  for x in np.unique(last_week):
    fy_x = poisson.cdf(x, lambda_mme)
    current_element_count = frequency_map[x]
    max_diff = max(abs(f_x_x_neg - fy_x), abs(current_element_count*step_size - fy_x)
    f_x_x_neg = current_element_count*step_size

  if max_diff > threshold:
    print("Rejecting null hypothesis H_0")
  else:
    print("Accepting null hypothesis H_0")
  return max_diff
```

Cases

```
one_sample_ks_test_with_poisson_distribution(second_last_week_cases["#Cases_per_day"]
                                             last_week_cases["#Cases_per_day"],
                                             0.05)
```

```
Rejecting null hypothesis H_0
0.8571428571428572
```

Death

```
one_sample_ks_test_with_poisson_distribution(second_last_week_death["#Deaths_per_day"
                                             last_week_death["#Deaths_per_day"],
                                             0.05)
```

```
Rejecting null hypothesis H_0
0.8562734052249001
```

The KS test rejects the null hypotheses for both the deaths and the cases. That means the data doesı
Concluding that the Deaths and cases don't follow poisson distribution.

One sample KS test with geometric distribution

```
def one_sample_ks_test_with_geometric_distribution(second_last_week, last_week,thresh
  p_mme = 1/np.mean(second_last_week)
  last_week = np.sort(last_week)
  max_diff = float("-inf")
  step_size = 1/len(last_week)
  frequency_map = frequency_count(last_week)
  f_x_x_neg = 0
  for x in np.unique(last_week):
    fy_x = 1- pow(1-p_mme, x) #fy_x + (math.exp(-1 * lambda_mme)*pow(lambda_mme,i))/m
    current_element_count = frequency_map[x]
    max_diff = max(abs(f_x_x_neg - fy_x), abs(current_element_count*step_size - fy_x)
    f_x_x_neg = current_element_count*step_size
  print("max_diff is: ", max_diff)
  if max_diff > threshold:
    print("Rejecting null hypothesis H_0")
  else:
    print("Accepting null hypothesis H_0")
  return max_diff
```

Cases

```
one_sample_ks_test_with_geometric_distribution(second_last_week_cases["#Cases_per_day
                                             last_week_cases["#Cases_per_day"],
                                             0.05)
```

```
max_diff is:  0.5532948858470867
Rejecting null hypothesis H_0
0.5532948858470867
```

Applicable or not?

Death

```
one_sample_ks_test_with_geometric_distribution(second_last_week_death["#Deaths_per_da
                                                last_week_death["#Deaths_per_day"],
                                                0.05)
```

```
max_diff is:  0.525709507066829
Rejecting null hypothesis H_0
0.525709507066829
```

Observation: the test rejects the assumption that the data comes from geometric distribution.

One sample KS test with Binomial distribution

```
from scipy.stats import binom

def one_sample_ks_test_with_binomial_distribution(second_last_week, last_week, thresh
  p_mme = 1 - np.var(second_last_week)/np.mean(second_last_week)
  print(p_mme)
  n_mme = pow(np.mean(second_last_week),2)/(np.mean(second_last_week)-np.var(second_l
  print(n_mme)

  n = len(last_week)
  last_week = np.sort(last_week)
  max_diff = float("-inf")
  step_size = 1/n
  frequency_map = frequency_count(last_week)
  f_x_x_neg = 0
  for x in np.unique(last_week):
    fy_x =  binom.cdf(x, n, p_mme)
    current_element_count = frequency_map[x]
    max_diff = max(abs(f_x_x_neg - fy_x), abs(current_element_count*step_size - fy_x)
    f_x_x_neg = current_element_count*step_size
  print("max_diff is: ", max_diff)
  if max_diff > threshold:
    print("Rejecting null hypothesis H_0")
  else:
    print("Accepting null hypothesis H_0")
  return max_diff
```

Cases

```
one_sample_ks_test_with_binomial_distribution(second_last_week_cases["#Cases_per_day"
                                              last_week_cases["#Cases_per_day"],
                                              0.05)
```

```
-131.35771583349455
-73.07526580446503
max_diff is:  0.8571428571428572
Rejecting null hypothesis H_0
0.8571428571428572
```

Death

```
one_sample_ks_test_with_binomial_distribution(second_last_week_death["#Deaths_per_day
                                              last_week_death["#Deaths_per_day"],
                                              0.05)
```

```
-8.441787964609542
-110.11546753702716
max_diff is:  0.8571428571428572
Rejecting null hypothesis H_0
0.8571428571428572
```

Observation

the test rejects the assumption that the data comes from binomial distribution.

Two sample KS test

```
def ks_test(X, Y, threshold):
    X = np.sort(X)
    Y = np.sort(Y)
    max_diff = float("-inf")
    max_x = 0
    y_min = 0
    y_max = 0
    for y in Y:
        count = 0;
        for x in X:
            if x < y:
                count = count + 1
        f_hat_x = count / len(X)
        f_hat_y_neg = np.searchsorted(Y, y, side='left') / len(Y)
        f_hat_y_pos = np.searchsorted(Y, y, side='right') / len(Y)
        if abs(f_hat_x - f_hat_y_neg) > max_diff or abs(f_hat_x - f_hat_y_pos) > max_
```

```
            max_diff = max(max_diff, abs(f_hat_x - f_hat_y_neg), abs(f_hat_x - f_hat_
            max_x = y
            y_min = max(y_min, f_hat_y_neg)
            y_max = max(y_max, f_hat_x)
    print("max_diff is: ", max_diff)
    if max_diff > threshold:
      print("Rejecting null hypothesis H_0")
    else:
      print("Accepting null hypothesis H_0")
```

Cases

```
ks_test(second_last_week_cases["#Cases_per_day"],last_week_cases["#Cases_per_day"],0.
```

```
max_diff is:  0.7142857142857143
Rejecting null hypothesis H_0
```

Deaths

```
ks_test(second_last_week_death["#Deaths_per_day"],last_week_death["#Deaths_per_day"],
```

```
max_diff is:  0.2857142857142857
Rejecting null hypothesis H_0
```

Observation

The test rightly rejects the assumption for both the cases and deaths. Assumption being data for sec distribution as last week. That's stemming from the fact that the values of data change significantly v

Permutation Test

```
def permutation_test(X, Y, p_count, p_threshold):
    x_bar = np.mean(X)
    y_bar = np.mean(Y)
    t_obs = abs(x_bar - y_bar)
    C = np.concatenate((X, Y))
    count = 0
    for i in range(p_count):
        p = np.random.permutation(C)
        mid = int(len(C) / 2)
        t_i = abs(np.mean(p[:mid]) - np.mean(p[mid:]))
        if t_i > t_obs:
            count += 1
    p_value = count / p_count
    print("p_value is: ", p_value)
```

```
    if p_value <= p_threshold:
        print("Rejecting Null Hypothesis, X and Y don't come from same distribution")
    else:
        print("Accepting Null Hypothesis, X and Y come from same distribution")
```

Cases

```
permutation_test(second_last_week_cases["#Cases_per_day"],
                 last_week_cases["#Cases_per_day"],
                 len(second_last_week_cases["#Cases_per_day"])+len(last_week_cases["#
                 0.05)
```

```
p_value is:  0.14285714285714285
Accepting Null Hypothesis, X and Y come from same distribution
```

Death

```
permutation_test(second_last_week_death["#Deaths_per_day"],
                 last_week_death["#Deaths_per_day"],
                 len(second_last_week_death["#Deaths_per_day"])+len(last_week_death["
                 0.05)
```

```
p_value is:  0.35714285714285715
Accepting Null Hypothesis, X and Y come from same distribution
```

Observation

The permutation test sometimes accept the null hypothesus of same distribution

Required Inference 4

Report the Pearson correlation value for #deaths and your X dataset, and also for #cases and your X most relevant column in X to compare against the covid numbers.

```
Xdata = throg_neck_2020
Xdata.shape
```

```
(37, 1)
```

Since the Xdata had 1 outlier that was removed while cleaning the data, we have 37 data points.

```
X_pear = np.array(Xdata['total vehicles'])
X_pear.shape
```

```
(37,)
```

```
def calculate_sampleVar(X):
  return np.sum(np.square((X - np.mean(X))))/len(X)
```

For #cases

```
ny_cases_cap
```

	#Cases	#Cases_per_day	index
3/12/20	327	107.0	0
3/13/20	421	94.0	1
3/14/20	613	192.0	2
3/15/20	615	2.0	3
3/16/20	967	352.0	4
3/17/20	1578	611.0	5
3/18/20	3038	1460.0	6
3/19/20	5704	2666.0	7
3/20/20	8403	2699.0	8
3/21/20	11727	3324.0	9
3/22/20	15800	4073.0	10
3/23/20	20884	5084.0	11
3/24/20	25681	4797.0	12
3/25/20	30841	5160.0	13
3/26/20	37877	7036.0	14

Since 1 data point from Xdata was removed -- removing the corresponding date's datapoint from #ca

i.e., April 5th, 2020 which is at index 24

3/29/20	59648	7238.0	17

```
Y_pear = np.array(ny_cases_cap['#Cases_per_day'])
Y_pear = np.delete(Y_pear, [24], axis=0)
Y_pear
```

```
array([1.0700e+02, 9.4000e+01, 1.9200e+02, 2.0000e+00, 3.5200e+02,
       6.1100e+02, 1.4600e+03, 2.6660e+03, 2.6990e+03, 3.3240e+03,
       4.0730e+03, 5.0840e+03, 4.7970e+03, 5.1600e+03, 7.0360e+03,
       6.9990e+03, 7.5340e+03, 7.2380e+03, 7.0150e+03, 9.1700e+03,
       8.1150e+03, 8.5580e+03, 1.0481e+04, 1.0846e+04, 8.6550e+03,
       8.0600e+03, 1.1186e+04, 1.0718e+04, 1.0569e+04, 8.6780e+03,
       8.0070e+03, 6.7160e+03, 7.2710e+03, 1.1434e+04, 9.2370e+03,
       6.9060e+03, 6.8770e+03])
```

```
print("shape of Y = ", Y_pear.shape)
print("shape of X = ", X_pear.shape)
```

```
shape of Y =  (37,)
shape of X =  (37,)
```

4/10/20	172348	10569.0	29

```
X = X_pear
```

```
X = X_pear
Y = Y_pear

# calculating the pearson correlation coefficient
numer = np.sum((X - np.mean(X))*(Y - np.mean(Y)))
denom = np.sqrt(calculate_sampleVar(X) * calculate_sampleVar(Y) * len(X) * len(Y))
denom_2 = np.sqrt(np.sum(np.square(X - np.mean(X))) * np.sum(np.square(Y - np.mean(Y)
rho_hat = numer/denom

print("rho hat value = ", rho_hat)

if rho_hat > 0.5:
  print("positive linear dependence/correlation")
elif rho_hat < -0.5:
  print("negative linear dependence/correlation")
else:
  print("No correlation")
```

```
rho hat value =  -0.8082305529213043
negative linear dependence/correlation
```

We can see from this result that our X data and #cases have a **very high** negative linear correlation.

For #deaths

```
ny_deaths_cap
```

	#Deaths	#Deaths_per_day	index
3/12/20	1	1.0	0
3/13/20	2	1.0	1
3/14/20	6	4.0	2
3/15/20	12	6.0	3
3/16/20	24	12.0	4
3/17/20	38	14.0	5
3/18/20	63	25.0	6
3/19/20	96	33.0	7
3/20/20	151	55.0	8
3/21/20	195	44.0	9
3/22/20	286	91.0	10
3/23/20	387	101.0	11
3/24/20	512	125.0	12
3/25/20	659	147.0	13
3/26/20	902	243.0	14
3/27/20	1211	309.0	15
3/28/20	1588	377.0	16
3/29/20	2016	428.0	17
3/30/20	2556	540.0	18
3/31/20	3207	651.0	19
4/1/20	3917	710.0	20
4/2/20	4730	813.0	21
4/3/20	5418	688.0	22
4/4/20	5991	573.0	23
4/5/20	6785	794.0	24

Since 1 data point from Xdata was removed -- removing the corresponding date's datapoint from #ca

i.e., April 5th, 2020 which is at index 24

```
Y_pear = np.array(ny_deaths_cap['#Deaths_per_day'])
Y_pear = np.delete(Y_pear, [24], axis=0)
Y_pear
```

```
array([1.000e+00, 1.000e+00, 4.000e+00, 6.000e+00, 1.200e+01, 1.400e+01,
       2.500e+01, 3.300e+01, 5.500e+01, 4.400e+01, 9.100e+01, 1.010e+02,
       1.250e+02, 1.470e+02, 2.430e+02, 3.090e+02, 3.770e+02, 4.280e+02,
       5.400e+02, 6.510e+02, 7.100e+02, 8.130e+02, 6.880e+02, 5.730e+02,
       1.024e+03, 1.077e+03, 9.480e+02, 9.440e+02, 8.270e+02, 8.930e+02,
       9.440e+02, 9.480e+02, 8.710e+02, 7.520e+02, 7.160e+02, 1.026e+03,
       1.005e+03])
```

```
print("shape of Y = ", Y_pear.shape)
print("shape of X = ", X_pear.shape)
```

```
shape of Y =  (37,)
shape of X =  (37,)
```

```
X = X_pear
Y = Y_pear

# calculating the pearson correlation coefficient
numer = np.sum((X - np.mean(X))*(Y - np.mean(Y)))
denom = np.sqrt(calculate_sampleVar(X) * calculate_sampleVar(Y) * len(X) * len(Y))
denom_2 = np.sqrt(np.sum(np.square(X - np.mean(X))) * np.sum(np.square(Y - np.mean(Y)
rho_hat = numer/denom
print("rho value = ", rho_hat)

if rho_hat > 0.5:
  print("positive linear dependence/correlation")
elif rho_hat < -0.5:
  print("negative linear dependence/correlation")
else:
  print("No correlation")
```

```
rho value =  -0.66720644607601
negative linear dependence/correlation
```

We can see from this result that our X data and #cases have a **high** negative linear correlation.

Required Inference 5

Observation: We could see from execution that as we saw more data the probability density decrease

Assume the daily deaths are Poisson distributed with parameter lambda. Assume an Exponential pric beta for the prior, equate the mean of the Exponential prior to that of the Poisson lambda_MME. That week's data, and equate this lambda to the mean of Exp(1/beta) to find beta for the prior. Use first we lambda via Bayesian inference. Now, use second week's data to obtain the new posterior, using prior end of week 4. Plot all posterior distributions on one graph. Report the MAP for all posteriors.

```
D = np.array(ny_deaths_cap['#Deaths_per_day'])
D.shape
```

```
(38,)
```

```
D1 = D[:7]
D2 = D[7:14]
D3 = D[14:21]
D4 = D[21:28]
print("First week's data = ", D1)
print("2nd week's data = ", D2)
print("3rd week's data = ", D3)
print("4th week's data = ", D4)
```

```
First week's data =  [ 1.   1.   4.   6. 12. 14. 25.]
2nd week's data =  [ 33.   55.   44.   91. 101. 125. 147.]
3rd week's data =  [243. 309. 377. 428. 540. 651. 710.]
4th week's data =  [ 813.   688.   573.   794. 1024. 1077.   948.]
```

```
# mean of exp(1/beta) = beta = lambda_mme = D1_bar

beta = np.mean(D1)
print("beta = ", beta)
```

```
beta =  9.0
```

Posterior of lambda via bayesian inference (after seeing D1) takes the form of a gamma distribution g

a = 1 + sum(D1)

b = len(D1) + 1/beta

On taking the derivative of the posterior distribution and equating to 0, we find:

lambda_MAP = sum(D1)/(len(D1) + 1/beta)

Now, when we use second week's data to obtain the new posterior, using prior as posterior after week gamma(a,b), such that,

a = 1 + sum(D1 and D2)

b = len(D1 and D2) + 1/beta

Also,

lambda_MAP = sum(D1 and D2)/(len(D1 and D2) + 1/beta)

And, so on, we see gamma(a,b) is the conjugate prior of Poisson distribution, thus,

f(lambda | D1 and D2 and D3) ~ gamma(a,b) , where

a = 1 + sum(D1 and D2 and D3)

b = len(D1 and D2 and D3) + 1/beta

Also,

lambda_MAP = sum(D1 and D2 and D3)/(len(D1 and D2 and D3) + 1/beta)

---

f(lambda | D1 and D2 and D3 and D4) ~ gamma(a,b) , where

a = 1 + sum(D1 and D2 and D3 and D4)

b = len(D1 and D2 and D3 and D4) + 1/beta

Also,

lambda_MAP = sum(D1 and D2 and D3 and D4)/(len(D1 and D2 and D3 and D4) + 1/beta)

```
def calculate_a(data):
  return 1 + np.sum(data)

def calculate_b(data):
  return len(data) + (1/beta)

def calculate_lambdaMAP(data):
  return np.sum(data)/(len(data) + (1/beta))

lambda_MAP = []
a = []
b = []
```

Calculate a, b, and lambda_MAP after week1, week2, week3 and week4

```
# Week1
a.append(calculate_a(D1))
b.append(calculate_b(D1))
lambda_MAP.append(calculate_lambdaMAP(D1))

# Week2
w2 = np.concatenate((D1,D2))
a.append(calculate_a(w2))
b.append(calculate_b(w2))
lambda_MAP.append(calculate_lambdaMAP(w2))

# Week3
w3 = np.concatenate((w2,D3))
a.append(calculate_a(w3))
b.append(calculate_b(w3))
lambda_MAP.append(calculate_lambdaMAP(w3))

# Week4
```

```
# Week4
w4 = np.concatenate((w3,D4))
a.append(calculate_a(w4))
b.append(calculate_b(w4))
lambda_MAP.append(calculate_lambdaMAP(w4))
```

Plot Posterior Distributions and Report lambda_MAP

```
def plot_posterior(a, b):
    x = np.linspace(0, 500, 1000)
    y1 = gamma.pdf(x, a[0], scale=1/b[0])
    y2 = gamma.pdf(x, a[1], scale=1/b[1])
    y3 = gamma.pdf(x, a[2], scale=1/b[2])
    y4 = gamma.pdf(x, a[3], scale=1/b[3])
    plt.subplots(figsize=(15,10))
    plt.title("Posterior Distributions")
    plt.xlabel("lambda")
    plt.ylabel("Probability Density")
    plt.plot(x, y1, label="Week 1", color='brown')
    plt.plot(x, y2, label="Week 2", color='purple')
    plt.plot(x, y3, label="Week 3", color='orange')
    plt.plot(x, y4, label="Week 4", color='green')
    plt.legend(bbox_to_anchor=(1, 1), loc='upper right',
               borderaxespad=1, fontsize=12)

plot_posterior(a, b)
```


```
map_df = pd.DataFrame(lambda_MAP, columns=["lambda_MAP"])
map_df.index = ["Week 1", "Week 2", "Week 3", "Week 4"]
map_df
```

	lambda_MAP
Week 1	8.859375
Week 2	46.700787
Week 3	185.542105
Week 4	349.826087

Additional inference with Week 5

Since we had 38 data points we tried plotting for week 5 too.

(post which 3 data points were remaining - ignored the same for this inference)

```
D5 = D[29:35]


# Week5
w5 = np.concatenate((w4,D5))
a.append(calculate_a(w5))
b.append(calculate_b(w5))
lambda_MAP.append(calculate_lambdaMAP(w5))

def plot_posterior(a, b):
    x = np.linspace(0, 500, 1000)
    y1 = gamma.pdf(x, a[0], scale=1/b[0])
    y2 = gamma.pdf(x, a[1], scale=1/b[1])
    y3 = gamma.pdf(x, a[2], scale=1/b[2])
    y4 = gamma.pdf(x, a[3], scale=1/b[3])
    y5 = gamma.pdf(x, a[4], scale=1/b[4])
    plt.subplots(figsize=(15,10))
    plt.title("Posterior Distributions")
    plt.xlabel("lambda")
    plt.ylabel("Probability Density")
    plt.plot(x, y1, label="Week 1", color='brown')
```

```
    plt.plot(x, y1, label="Week 1", color='brown')
    plt.plot(x, y2, label="Week 2", color='purple')
    plt.plot(x, y3, label="Week 3", color='orange')
    plt.plot(x, y4, label="Week 4", color='green')
    plt.plot(x, y5, label="Week 5", color='gray')
    plt.legend(bbox_to_anchor=(1, 1), loc='upper right',
               borderaxespad=1, fontsize=12)

plot_posterior(a, b)

map_df = pd.DataFrame(lambda_MAP, columns=["lambda_MAP"])
map_df.index = ["Week 1", "Week 2", "Week 3", "Week 4", "Week 5"]
map_df
```

	lambda_MAP
Week 1	8.859375

Additional Inferences

Inference 1

Use your X dataset to check if COVID19 had an impact on the X data. State your hypothesis clearly ar
those learned in class) to apply to your hypotheses. Also check whether the tool/test is applicable or

EXPLANATION:

Type 1. We want to test if #Vehicles_per_day commuting on Throgs Neck Bridge decreased due to C(
the same timeframe.

X = #vehicles_per_day during [03-12-2020, 04-18-2020] Y = #vehicles_per_day during [03-12-2019, 04-

Hypothesis:

H0: mean_X >= mean_Y

v/s

H1: mean_X < mean_Y

We use **one-sided unpaired and paired T-tests** to test the hypothesis.

Also, we wanted to check the extent of decrease observed so we calculate the p-value.

Type 2. **Further**, we wanted to check when did the traffic **started** to decrease, so we do a week by wee

We go ahead and define this test as:

X1 = second week of Y , i.e., dayds 8-14 Y1 = first week of Y, i.e., 1st 7 days

H0: mean_X >= mean_Y

v/s

H1: mean_X < mean_Y

Again, using the same tests and extent calculation by computing p-value.

We mean to do this for every consecutinve pairs of weeks till we find the week with the drop, or in oth

Applicability : Don't know data normal so don;t know applicable or not

But, we used T-test because n is small

```
# H0: mu_x >= mu_y and H1: mu_x < mu_y

def calculate_sampleVar(X):
  return np.sum(np.square((X - np.mean(X))))/len(X)

def one_sided_unpaired_t_test(X, Y, t_thresh):
  D_bar = np.mean(X) - np.mean(Y) # calculate d bar
  s_x2 = calculate_sampleVar(X) # sample variance of x
  s_y2 = calculate_sampleVar(Y) # sample variance of y
  pooled_sd = np.sqrt((s_x2/len(X)) + (s_y2/len(Y))) # pooled standard deviation
  u_t = D_bar/pooled_sd

  print("D_bar = ", D_bar)
  print("D_bar from Xbar - Ybar = ", np.mean(X) - np.mean(Y))
  print("pooled std dev= ", pooled_sd)
  print("t = ", u_t)

  if u_t < t_thresh:
    print("reject H0")
  else:
    print("accept H0")
```

Type 1

```
# X->  Xdat for [03-12-2020, 04-18-2020]
# Y->  Xdata [03-12-2019, 04-18-2019]


n = len(throg_neck_2019)
m = len(throg_neck_2020)
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.7823 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(throg_neck_2020.values,
                            throg_neck_2019.values,t_threshold)
```

```
Degree of freedoms:  72
T-threshold:  -1.7823
D_bar =  -63158.75675675675
D_bar from Xbar - Ybar =  -63158.75675675675
pooled std dev=  3415.9667858844314
t =  -18.489277184351856
reject H0
```

Type 2

```
# X -> second week of 2020
# Y -> first week of 2020
# apply one tailed, and if rejected x -> third week y-> first
# stop when accept
# find p-value
# apply paired t-test for same dataset and find p-value

n = len(throg_neck_2020_weekly[0])
m = len(throg_neck_2020_weekly[1])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6663 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(throg_neck_2020_weekly[1].values,
                              throg_neck_2020_weekly[0].values,t_threshold)

print()

n = len(throg_neck_2020_weekly[1])
m = len(throg_neck_2020_weekly[2])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6663 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(throg_neck_2020_weekly[2].values,
                              throg_neck_2020_weekly[1].values,t_threshold)
print()

n = len(throg_neck_2020_weekly[2])
m = len(throg_neck_2020_weekly[3])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6663 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(throg_neck_2020_weekly[3].values,
                              throg_neck_2020_weekly[2].values,t_threshold)
```

```
Degree of freedoms:  12
T-threshold:  -1.6663
D_bar =  -32713.71428571429
D_bar from Xbar - Ybar =  -32713.71428571429
pooled std dev=  6483.27370919493
t =  -5.045863517888798
reject H0

Degree of freedoms:  12
T-threshold:  -1.6663
```

So for 3rd-4th week did our data get different

```
pooled std dev=  6212.400952275757
```

Inference 2

```
Degree of freedoms:  12
```

Check if COVID19 data changed after some local event or rule was enforced, like lockdown or stay-at
data before and after the event. Maybe take into account that COVID19 takes some time to show sym
the lockdown to show its effects.

```
accept H0
```

EXPLANATION:

We want to test if #cases_per_day and #deaths_per_day decreased after the lockdown rule was pass

Here, we take 2 cases and hypothesize:

Type 1. Treating March 22, 2020 as the change point Type 2. Treating April 5, 2020 as the change poi
reason being 14 days of incubation period. But, in this case we consider the data for 10 days before M
April 5 because we wanted to capture the effects before and after incubation period. Type 3: Treating
14 days after the actual order: reason being 14 days of incubation period. But, in this case we conside
after April 5 because we wanted to capture the effects before and after incubation period.

Now, we hypothesize Type 2 in 2 ways:

Type 2.1:

X = #cases_per_day before and including 03-22-2020 Y = #cases_per_day after 03-22-2020

H0: mean_X >= mean_Y

v/s

H1: mean_X < mean_Y

We apply **one sided unpaired T-test** here.

Type 2.2:

X = #cases_per_day 14 days before 04-05-2020 Y = #cases_per_day on 04-05-2020 and 13 days after

We apply **one sided paired AND unpaired T-tests** here.

Hypothesis:

H0: mean_X >= mean_Y

v/s

H1: mean_X < mean_Y

We use **one-sided unpaired and paired T-tests** to test the hypothesis.

Further, we go ahead and check the extent of effect so we compute the p-value.

Applicability : Don't know data normal so don;t know applicable or not

But, we used T-test because n is small

Cases

Type 1

```
# unpaired = March 12-21, Apr 5-14
n = len(ny_cases_cap['#Cases'][0:10])
m = len(ny_cases_cap['#Cases'][10:])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(ny_cases_cap['#Cases'][10:].values,
                          ny_cases_cap['#Cases'][0:10].values,t_threshold)
```

```
Degree of freedoms:  36
T-threshold:  -1.6883
D_bar =  117334.59285714285
D_bar from Xbar - Ybar =  117334.59285714285
pooled std dev=  13326.353883102527
t =  8.804703363454882
accept H0
```

https://www.socscistatistics.com/pvalues/tdistribution.aspx for given t, significance level and degree

Type 2.1

```
ny_cases_cap['#Cases'][24:34]
```

```
4/5/20     123160
4/6/20     131815
4/7/20     139875
4/8/20     151061
```

```
# paired
# X - > March 12-21(10 days),
# Y - > Apr 5-14(10 days)
n = len(ny_cases_cap['#Cases'][0:10])
m = len(ny_cases_cap['#Cases'][24:34])
print("Degree of freedoms: ", n+m-2)
t_threshold =  2.1009
print("T-threshold: ", t_threshold)
paired_t_test(ny_cases_cap['#Cases'][0:10],
                              ny_cases_cap['#Cases'][24:34],t_threshold)
```

```
Degree of freedoms:  18
T-threshold:  2.1009
mod T:  21.96629897501323
reject H0
```

Hence lockdown help change

The p-value is < .00001.

Type 2.2

```
# X -> March 12-21,
# Y -> March 22-Apr 18 [unpaired] -> not considering impact of incubation time
# whole data immediately after lockdown
n = len(ny_cases_cap['#Cases'][0:10])
m = len(ny_cases_cap['#Cases'][24:34])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(ny_cases_cap['#Cases'][24:34],
                              ny_cases_cap['#Cases'][:10],t_threshold)
```

```
Degree of freedoms:  18
T-threshold:  -1.6883
D_bar =  161547.30000000002
D_bar from Xbar - Ybar =  161547.30000000002
pooled std dev=  8432.035208951631
t =  19.158755389030862
accept H0
```

The p-value is < .00001.

Type 3

```
# X -> Apr 5 -14 Y -> Apr5 +14 [paired, unpaired] ->
# considering the change point to be start of lockdown order + incubation period


# paired
# why april 5? since lockdown was imposed on Mar 22 in new york
# so considering 14 days incubation period, we arrive at April 5

n = len(ny_cases_cap['#Cases'][10:24])
m = len(ny_cases_cap['#Cases'][24:])
print("Degree of freedoms: ", n+m-2)
t_threshold =  2.0555
print("T-threshold: ", t_threshold)
paired_t_test(ny_cases_cap['#Cases'][10:24],
                              ny_cases_cap['#Cases'][24:],t_threshold)
```

```
Degree of freedoms:  26
T-threshold:  2.0555
mod T:  62.71569615819643
reject H0
```

The p-value is < .00001.

```
# unpaired
n = len(ny_cases_cap['#Cases'][0:24])
m = len(ny_cases_cap['#Cases'][24:])
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(ny_cases_cap['#Cases'][24:],
                              ny_cases_cap['#Cases'][0:24],t_threshold)
```

```
Degree of freedoms:  36
T-threshold:  -1.6883
D_bar =  146790.0238095238
D_bar from Xbar - Ybar =  146790.0238095238
pooled std dev=  12098.47561971039
t =  12.132935455965947
accept H0
```

The p-value is < .00001.

Death

Death(to measure 14 days impact)

```
#Unpaired
```

```
# Y -> pre lock down data
# X -> Post lock down data
Y = ny_deaths_cap['#Deaths'][0:10]
X = ny_deaths_cap['#Deaths'][10:]
n = len(X)
m = len(Y)
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(X.values,
                              Y.values,t_threshold)
```

```
Degree of freedoms:  36
T-threshold:  -1.6883
D_bar =  7581.378571428571
D_bar from Xbar - Ybar =  7581.378571428571
pooled std dev=  1134.6318709005104
t =  6.681795889808335
accept H0
```

The p-value is < .00001.

```
# paired = March 12-21(10 days), Apr 5-14(10 days)
# Y -> pre lock down
# X -> Post lock down
Y = ny_deaths_cap['#Deaths'][0:10]
X = ny_deaths_cap['#Deaths'][24:34]

n = len(X)
m = len(Y)
print("Degree of freedoms: ", n+m-2)
t_threshold =  2.1009
print("T-threshold: ", t_threshold)
paired_t_test(X,Y,t_threshold)
```

```
Degree of freedoms:  18
T-threshold:  2.1009
mod T:  13.348843769238915
reject H0
```

The p-value is < .00001.

Death(not considering impact of incubation time)

```
# X -> post lockdown
# Y -> pre lockdown
X = ny_deaths_cap['#Deaths'][10:]
Y = ny_deaths_cap['#Deaths'][:10]
n = len(Y)
```

```
m = len(X)
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(X,Y,t_threshold)
```

```
Degree of freedoms:  36
T-threshold:  -1.6883
D_bar =  7581.378571428571
D_bar from Xbar - Ybar =  7581.378571428571
pooled std dev=  1134.6318709005104
t =  6.681795889808335
accept H0
```

The p-value is < .00001.

Deaths(considering the change point to be start of lockdown order + incubation period)

```
# paired
# X -> time between lockdown and incubation period
# Y -> post lockdown
X = ny_deaths_cap['#Deaths'][10:24]
Y = ny_deaths_cap['#Deaths'][24:]
n = len(X)
m = len(Y)
print("Degree of freedoms: ", n+m-2)
t_threshold =  2.0555
print("T-threshold: ", t_threshold)
paired_t_test(ny_deaths_cap['#Deaths'][10:24],
                              ny_deaths_cap['#Deaths'][24:],t_threshold)
```

```
Degree of freedoms:  26
T-threshold:  2.0555
mod T:  20.669435578811886
reject H0
```

The p-value is < .00001.

```
# unpaired
# X is post lockdown
# Y is pre lockdown
X = ny_deaths_cap['#Deaths'][24:]
Y = ny_deaths_cap['#Deaths'][0:24]
n = len(X)
m = len(Y)
print("Degree of freedoms: ", n+m-2)
t_threshold = -1.6883 # negative since it's one sided and decided per our hypothesis
print("T-threshold: ", t_threshold)
one_sided_unpaired_t_test(X,Y,t_threshold)
```

```
Degree of freedoms:  36
T-threshold:  -1.6883
D_bar =  11480.738095238095
D_bar from Xbar - Ybar =  11480.738095238095
pooled std dev=  1043.1366785513578
t =  11.005976811381819
accept H0
```

The p-value is < .00001.

Inference 3

Use linear regression to find the impact of age, gender, underlying conditions, etc., on the severity of c

EXPLANATION:

We want to check the impact of gender and age on #cases and #deaths.

To do that we needed data which was not readily available online. We had to manually collate the dat cases in a csv.

We used the following sources to do that:

https://www1.nyc.gov/site/doh/covid/covid-19-data-archive.page

https://www1.nyc.gov/assets/doh/downloads/pdf/imm/covid-19-daily-data-summary-hospitalizatior

Please note that this is not the same #cases and #deaths data as we used before for all other inferer

This data is from March 24, 2020 to April 24, 2020 which still satisfies the requirement of minimum 3

Now, we define our tests:

1. We apply simple linear regression for:
   1. Y = #hospitalized_cases , X = #males
   2. Y = #hospitalized_cases , X = #females
   3. Y = #hospitalized_cases , X = #cases_falling_in_age_group_0-17
   4. Y = #hospitalized_cases , X = #cases_falling_in_age_group_75_above

   We did not want to do multiple linear regression with gender because it would have turned out t X2.

   With the beta coefficients that we find using linear regression, we can estimate the impact of th
2. Further, we also apply Chi square test:
   1. H0: Y is independent of X

   v/s

H1: Y is not independent of X

We do this for:

X = gender , Y = hospitalization

and

X = age_group (less than 65 or above 65yrs) Y = hospitalization

We use the cumulative data of #females_hospitalized, #females_not_hospitalized, #males_hospitaliz
#less_than_65_hospitalized, #more_than_65_hospitalized,
from April 24, 2020 to apply chi square test.

Apllicabloty : don't know if ideal linear relationship exists so LR not applicable

Chi square : no assumptions so applicable

Observation from below: We can observe the value of b1_hat for males lr model is 1.72 while for fema
coefficients we can conclude that covid impacted males more. [since y is total hospitalization a lowe

gender impact

```
demographics_data = pd.read_csv("CSE544_Project/demographics.csv")
```

```
demographics_data.shape
```

```
(32, 19)
```

```
demographics_data.head()
```

	Date	Male	Female	Unknown	Male all cases	Female all cases	Unknown all cases	H-0 to 17	T-0 to 17	H-18 to 44	T-18 to 44	H-45 to 64
0	March 24,2020	1691	1191	1	8838	6736	23	28	384	629	7094	1061
1	March 25,2020	2293	1628	1	11325	8655	31	35	446	837	8880	1470
2	March 26,2020	2801	1918	1	12948	10124	40	46	495	950	10145	1749
3	March 27,2020	3003	2035	1	14863	11792	42	47	543	971	11617	1886
4	March 28,2020	3741	2545	1	17133	13592	40	58	591	1224	13213	2350

Male, Female and Unknown column denote the hospitalization

```
gender_hospitalization = demographics_data[['Male','Female']]
```

```
gender_hospitalization['total'] = gender_hospitalization['Male'] + gender_hospitaliza
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab]
  """Entry point for launching an IPython kernel.
```

```
gender_hospitalization
```

	Male	Female	total
0	1691	1191	2882
1	2293	1628	3921
2	2801	1918	4719
3	3003	2035	5038
4	3741	2545	6286
5	4408	3001	7409
6	4610	3130	7740
7	5093	3455	8548
8	5792	3980	9772
9	6295	4292	10587
10	6970	4765	11735

```
gender_hospitalization['male_per_day'] = demographics_data['Male'].diff().fillna(demo
gender_hospitalization['female_per_day'] = demographics_data['Female'].diff().fillna(
gender_hospitalization['total_per_day'] = gender_hospitalization['total'].diff().fill
gender_hospitalization
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  This is separate from the ipykernel package so we can avoid doing imports unti
```

	Male	Female	total	male_per_day	female_per_day	total_per_day
0	1691	1191	2882	1691.0	1191.0	2882.0
1	2293	1628	3921	602.0	437.0	1039.0
2	2801	1918	4719	508.0	290.0	798.0
3	3003	2035	5038	202.0	117.0	319.0
4	3741	2545	6286	738.0	510.0	1248.0
5	4408	3001	7409	667.0	456.0	1123.0
6	4610	3130	7740	202.0	129.0	331.0
7	5093	3455	8548	483.0	325.0	808.0
8	5792	3980	9772	699.0	525.0	1224.0
9	6295	4292	10587	503.0	312.0	815.0
10	6970	4765	11735	675.0	473.0	1148.0
11	7596	5116	12712	626.0	351.0	977.0
12	8487	5714	14201	891.0	598.0	1489.0
13	9163	6166	15329	676.0	452.0	1128.0
14	11462	7709	19171	2299.0	1543.0	3842.0
15	12128	8159	20287	666.0	450.0	1116.0
16	12769	8618	21387	641.0	459.0	1100.0

```
def calculate_b1hat(X,Y):
    b1_hat = (np.sum(X*Y) - (np.mean(X)*np.mean(Y)*len(X)))/(np.sum(X*X) - (len(X)*np
    return b1_hat

def calculate_b0hat(X,Y,b1_hat):
    b0_hat = np.mean(Y) - (b1_hat*np.mean(X))
```

```
    b0_hat = np.mean(Y) - (b1_hat*np.mean(X))
    return b0_hat

def calculate_Yhat(b0_hat, b1_hat, X):
  Y_hat = b0_hat + (b1_hat*X)
  return Y_hat

def calculate_sse(Y, Y_hat):
  return np.sum(np.square(Y-Y_hat))

def calculate_mse(sse, Y):
  return sse/len(Y)

def calculate_e_hat(Y, Y_hat):
  return Y-Y_hat

def calculate_MAPE(e_hat, Y):
  return np.sum(((np.absolute(e_hat))/Y)*100)/len(Y)



X = gender_hospitalization['male_per_day']
Y = gender_hospitalization['total_per_day']

b1_hat = calculate_b1hat(X, Y)
print("b1_hat = ", b1_hat)

b0_hat = calculate_b0hat(X, Y, b1_hat)
print("b0_hat = ", b0_hat)

Y_hat = calculate_Yhat(b0_hat, b1_hat, X)
# print("Y_hat = ", Y_hat)

e_hat = calculate_e_hat(Y, Y_hat)
# print("e_hat = ", e_hat)

sse = calculate_sse(Y, Y_hat)
print("SSE =", sse)

mse = calculate_mse(sse, Y)
print("Mean Squared Error =", mse)

mape = calculate_MAPE(e_hat, Y)
print("MAPE = ", mape)
```

```
b1_hat =  1.726443366380674
b0_hat =  -12.636012024423735
SSE = 77208.46146148548
Mean Squared Error = 2412.7644206714212
MAPE =  4.396630690055824
```

```
X = gender_hospitalization['female_per_day']
Y = gender_hospitalization['total_per_day']
```

```
b1_hat = calculate_b1hat(X, Y)
print("b1_hat = ", b1_hat)

b0_hat = calculate_b0hat(X, Y, b1_hat)
print("b0_hat = ", b0_hat)

Y_hat = calculate_Yhat(b0_hat, b1_hat, X)
# print("Y_hat = ", Y_hat)

e_hat = calculate_e_hat(Y, Y_hat)
# print("e_hat = ", e_hat)

sse = calculate_sse(Y, Y_hat)
print("SSE =", sse)

mse = calculate_mse(sse, Y)
print("Mean Squared Error =", mse)

mape = calculate_MAPE(e_hat, Y)
print("MAPE = ", mape)
```

```
b1_hat =  2.3556185370883114
b0_hat =  28.1364581872524
SSE = 144078.9831955201
Mean Squared Error = 4502.468224860003
MAPE =  6.590347873073126
```

Compare the value of B hat for finding the gender impact

age group impact

finding the impact of age group on hospitalization

```
age_group_hospitalization = demographics_data[['H-0 to 17','H-75 and over']]
age_group_hospitalization['H-0 to 17 per day'] = age_group_hospitalization['H-0 to 17
age_group_hospitalization['H-75 and over per day'] = age_group_hospitalization['H-75
age_group_hospitalization['total'] = gender_hospitalization['total_per_day']
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
age_group_hospitalization
```

	H-0 to 17	H-75 and over	H-0 to 17 per day	H-75 and over per day	total
0	28	615	28.0	615.0	2882.0
1	35	844	7.0	229.0	1039.0
2	46	1029	11.0	185.0	798.0
3	47	1103	1.0	74.0	319.0
4	58	1388	11.0	285.0	1248.0
5	67	1620	9.0	232.0	1123.0
6	72	1722	5.0	102.0	331.0

```
X = age_group_hospitalization['H-0 to 17 per day']
Y = age_group_hospitalization['total']

b1_hat = calculate_b1hat(X, Y)
print("b1_hat = ", b1_hat)

b0_hat = calculate_b0hat(X, Y, b1_hat)
print("b0_hat = ", b0_hat)

Y_hat = calculate_Yhat(b0_hat, b1_hat, X)
# print("Y_hat = ", Y_hat)

e_hat = calculate_e_hat(Y, Y_hat)
# print("e_hat = ", e_hat)

sse = calculate_sse(Y, Y_hat)
print("SSE =", sse)

mse = calculate_mse(sse, Y)
print("Mean Squared Error =", mse)

mape = calculate_MAPE(e_hat, Y)
print("MAPE = ", mape)
```

```
b1_hat =  88.50209965688533
b0_hat =  511.2953090592513
SSE = 9174620.364239259
Mean Squared Error = 286706.88638247683
MAPE =  69.42334468298904
```

```
X = age_group_hospitalization['H-75 and over per day']
Y = age_group_hospitalization['total']

b1_hat = calculate_b1hat(X, Y)
print("b1_hat = ", b1_hat)

b0_hat = calculate_b0hat(X, Y, b1_hat)
print("b0_hat = ", b0_hat)
```

```
Y_hat = calculate_Yhat(b0_hat, b1_hat, X)
# print("Y_hat = ", Y_hat)

e_hat = calculate_e_hat(Y, Y_hat)
# print("e_hat = ", e_hat)

sse = calculate_sse(Y, Y_hat)
print("SSE =", sse)

mse = calculate_mse(sse, Y)
print("Mean Squared Error =", mse)

mape = calculate_MAPE(e_hat, Y)
print("MAPE = ", mape)
```

```
b1_hat =  3.4016034963292934
b0_hat =  85.20756725735623
SSE = 5308154.092463317
Mean Squared Error = 165879.81538947867
MAPE =  29.913287726910763
```

Chi-square test

Gender vs hospitalization

```
# These values are picked from last value of hospitalization excel file.
# Need to attach that file

f_hospitalized = 16407
f_not_hospitalized = 72050 - f_hospitalized
m_hospitalized = 23142
m_not_hospitalized = 78193 - m_hospitalized

total_hospitalized = f_hospitalized + m_hospitalized
total_not_hospitalized = f_not_hospitalized + m_not_hospitalized
total_female = f_hospitalized + f_not_hospitalized
total_male = m_hospitalized + m_not_hospitalized
total = total_female + total_male

T = [[f_hospitalized, f_not_hospitalized, total_female],
     [m_hospitalized, m_not_hospitalized, total_male],
     [total_hospitalized, total_not_hospitalized, total]]

E = [[0,0,0],
     [0,0,0],
     [0,0,0]]

Q_obs = 0
```

```
for r in range(2):
  for c in range(2):
    E[r][c] = (T[2][c] * T[r][2])/T[2][2]
    print(E[r][c])
    Q_obs = Q_obs + np.square(E[r][c] - T[r][c])/E[r][c]

print("Q_obs = ", Q_obs)
print("df (degrees of freedom) = ", (2-1)*(2-1))  ####  (#rows -1) * (#col -1). do no
print("Table look up p-value = Pr(chi_square with df > Q_obs)")
print("if p-value < 0.05 then reject H0")
```

```
18965.978115452966
53084.02188454704
20583.021884547034
57609.97811545296
Q_obs =  900.4392863102421
df (degrees of freedom) =  1
Table look up p-value = Pr(chi_square with df > Q_obs)
if p-value < 0.05 then reject H0
```

Age group vs hospitalization

```
older_hospitalized = 19828 # considering 65 and over
younger_hospitalized = 19740 # considering less than 65
older_not_hospitalized = 36073 - older_hospitalized # considering 65 and over(total -
younger_not_hospitalized = 114207 - younger_hospitalized # considering less than 65(t

total_hospitalized = older_hospitalized + younger_hospitalized
total_not_hospitalized = older_not_hospitalized + younger_not_hospitalized
total_older = older_hospitalized + older_not_hospitalized
total_younger = younger_hospitalized + younger_not_hospitalized
total = total_older + total_younger

T = [[older_hospitalized, older_not_hospitalized, total_older],
     [younger_hospitalized, younger_not_hospitalized, total_younger],
     [total_hospitalized, total_not_hospitalized, total]]

E = [[0,0,0],
     [0,0,0],
     [0,0,0]]

Q_obs = 0

for r in range(2):
  for c in range(2):
    E[r][c] = (T[2][c] * T[r][2])/T[2][2]
    print(E[r][c])
    Q_obs = Q_obs + np.square(E[r][c] - T[r][c])/E[r][c]

print("Q_obs = ", Q_obs)
```

```
print("df (degrees of freedom) = ", (2-1)*(2-1))  ####  (#rows -1) * (#col -1). do no
print("Table look up p-value = Pr(chi_square with df > Q_obs)")
print("if p-value < 0.05 then reject H0")
```

```
9497.847112057492
26575.152887942506
30070.152887942506
84136.8471120575
Q_obs =  20067.96231668534
df (degrees of freedom) =  1
Table look up p-value = Pr(chi_square with df > Q_obs)
if p-value < 0.05 then reject H0
```