In [1]:

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.stats import geom
from scipy.stats import binom
from itertools import permutations
import math
from statistics import mean
import random
from scipy.stats import gamma
import matplotlib.pyplot as  plt
import copy
import scipy.stats
```

In [2]:

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

```
Mounted at /content/gdrive
```

#Import Cases, Vaccination and AQI datasets (X dataset)

In [3]:

```
cases_df=pd.read_csv("/content/gdrive/MyDrive/United_States_COVID-19_Cases_and_Deaths_by_St
vaccine_df=pd.read_csv("/content/gdrive/MyDrive/COVID-19_Vaccinations_in_the_United_States_
aqi_2021=pd.read_csv("/content/gdrive/MyDrive/daily_aqi_by_county_2021.csv")
aqi_2020=pd.read_csv("/content/gdrive/MyDrive/daily_aqi_by_county_2020.csv")
```

In [4]:

```
cases_df
```

Out[4]:

	submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_
0	12/01/2021	ND	163565	135705.0	27860.0	589	220.0	
1	06/12/2021	AL	545637	421169.0	124468.0	130	51.0	
2	10/30/2021	WA	726837	NaN	NaN	1841	259.0	
3	03/28/2022	VT	107785	NaN	NaN	467	35.0	
4	03/11/2021	MD	390490	NaN	NaN	924	0.0	
...	...	...	...	...	...	...	...	
50575	12/11/2020	AZ	394804	380243.0	14561.0	6986	459.0	
50576	08/13/2021	FSM	7	7.0	0.0	0	0.0	
50577	12/15/2020	DC	25339	NaN	NaN	301	0.0	
50578	07/06/2021	OR	209494	209494.0	0.0	117	0.0	
50579	11/06/2021	RI	172709	NaN	NaN	305	0.0	

50580 rows × 15 columns

In [5]:

```
vaccine_df
```

Out[5]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Modern
0	05/15/2022	20	PR	7552350	215000	266212
1	05/15/2022	20	KS	6121515	256400	235494
2	05/15/2022	20	VA	19949085	785300	710870
3	05/15/2022	20	MT	2004895	105200	82520
4	05/15/2022	20	IH2	2965895	108400	131168
...	...	...	...	...	...	.
33427	12/13/2020	51	AS	3900	0	
33428	12/13/2020	51	MP	4875	0	
33429	12/13/2020	51	VI	975	0	
33430	12/13/2020	51	US	13650	0	
33431	12/13/2020	51	GU	3900	0	

33432 rows × 82 columns

Filtering rows to get data for the two assigned states - Delaware and Idaho

In [6]:

```
cases_df_id_de = cases_df[(cases_df.state=="DE") | (cases_df.state=="ID")]
cases_df_id_de
```

Out[6]:

	submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_
6	10/06/2021	DE	134690	124148.0	10542.0	444	45.0	
18	07/16/2021	DE	109472	102240.0	7232.0	40	14.0	
30	10/05/2021	ID	263294	209207.0	54087.0	1835	433.0	
38	08/17/2020	ID	27942	26091.0	1851.0	282	32.0	
47	08/05/2021	DE	111201	103716.0	7485.0	182	13.0	
...	...	...	...	...	...	...	...	
45456	07/02/2021	ID	195172	156211.0	38961.0	83	26.0	
45461	05/09/2020	DE	6476	6265.0	211.0	167	5.0	
45462	02/06/2022	ID	391067	306841.0	84226.0	0	0.0	
45463	12/04/2021	DE	154934	142206.0	12728.0	761	118.0	
45506	07/06/2020	DE	12338	11463.0	875.0	109	1.0	

1686 rows × 15 columns

#Removing null values

In [7]:

```
cases_df_id_de = cases_df_id_de.dropna()
cases_df_id_de.isnull().sum()
```

Out[7]:

```
submission_date    0
state              0
tot_cases          0
conf_cases         0
prob_cases         0
new_case           0
pnew_case          0
tot_death          0
conf_death         0
prob_death         0
new_death          0
pnew_death         0
created_at         0
consent_cases      0
consent_deaths     0
dtype: int64
```

#Since directly removing the entire row was resulting in a major loss of information, we are filtering just the rows which have NaN value for the column 'Administered'

In [8]:

```
vaccines_df_id_de=vaccine_df[(vaccine_df.Location=="DE") | (vaccine_df.Location=="ID")]
vaccines_df_id_de=vaccines_df_id_de[vaccines_df_id_de["Administered"].isna()==False]
vaccines_df_id_de
```

Out[8]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Modern
24	05/15/2022	20	ID	3445790	157600	135108
58	05/15/2022	20	DE	2401655	100100	92980
77	05/14/2022	19	ID	3445790	157600	135108
88	05/14/2022	19	DE	2401655	100100	92980
141	05/13/2022	19	DE	2398555	100100	92900
...	...	...	...	...	...	.
33294	12/16/2020	51	DE	8775	0	
33310	12/15/2020	51	ID	2925	0	
33364	12/15/2020	51	DE	975	0	
33389	12/14/2020	51	DE	975	0	
33414	12/14/2020	51	ID	1950	0	

1036 rows × 82 columns

#Sorting the datasets by the date to create daily stats from cumulative stats

In [9]:

```
cases_df_id_de['submission_date'] = pd.to_datetime(cases_df_id_de['submission_date'])
cases_df_id_de=cases_df_id_de.sort_values(by='submission_date')
cases_df_id_de
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithC
opyWarning:
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/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[9]:

	submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_
3054	2020-01-22	DE	0	0.0	0.0	0	0.0	
974	2020-01-23	DE	0	0.0	0.0	0	0.0	
7903	2020-01-24	DE	0	0.0	0.0	0	0.0	
533	2020-01-25	DE	0	0.0	0.0	0	0.0	
41165	2020-01-26	DE	0	0.0	0.0	0	0.0	
...	...	...	...	...	...	...	...	
3886	2022-05-11	ID	447540	350515.0	97025.0	381	112.0	
6263	2022-05-12	ID	447540	350515.0	97025.0	0	0.0	
43898	2022-05-12	DE	266673	242758.0	23915.0	966	106.0	
6672	2022-05-13	DE	267265	243270.0	23995.0	592	80.0	
1356	2022-05-13	ID	447913	350791.0	97122.0	373	97.0	

1602 rows × 15 columns

In [10]:

```
vaccines_df_id_de['Date'] = pd.to_datetime(vaccines_df_id_de['Date'])
vaccines_df_id_de=vaccines_df_id_de.sort_values(by='Date')
vaccines_df_id_de
```

Out[10]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna	Di
33414	2020-12-14	51	ID	1950	0	0	
33389	2020-12-14	51	DE	975	0	0	
33310	2020-12-15	51	ID	2925	0	0	
33364	2020-12-15	51	DE	975	0	0	
33289	2020-12-16	51	ID	2925	0	0	
...	...	...	...	...	...	...	
141	2022-05-13	19	DE	2398555	100100	929000	
88	2022-05-14	19	DE	2401655	100100	929800	
77	2022-05-14	19	ID	3445790	157600	1351080	
58	2022-05-15	20	DE	2401655	100100	929800	
24	2022-05-15	20	ID	3445790	157600	1351080	

1036 rows × 82 columns

We create the daily stats from the cumulative stats by subtracting the previous row from each row for the columns - total cases and total deaths. We then add back the other columns to the dataset.

In [11]:

```
def change_to_daily(col):
    global cumulative
    temp=col
    col-=cumulative
    cumulative=temp
    return col

cases_df_id_de_daily_de=pd.DataFrame()
cases_df_id_de_daily_id=pd.DataFrame()
num_columns=['tot_cases',
       'tot_death']

for col_name in num_columns:
    cumulative=0
    cases_df_id_de_daily_de[col_name]=cases_df_id_de[col_name][cases_df_id_de.state=="DE"].
    cumulative=0
    cases_df_id_de_daily_id[col_name]=cases_df_id_de[col_name][cases_df_id_de.state=="ID"].
cases_df_id_de_daily=pd.concat([cases_df_id_de_daily_de,cases_df_id_de_daily_id])
```

#We create a new column for both the cases and vaccination dataset and end up with two columns - one with cumulative total for number of cases and deaths and one for the daily number of cases and deaths

In [12]:

```
vaccine_daily_de=vaccines_df_id_de[vaccines_df_id_de.Location=="DE"]
vaccine_daily_id=vaccines_df_id_de[vaccines_df_id_de.Location=="ID"]


cumulative=0
vaccine_daily_de["Administered_new"]=vaccine_daily_de["Administered"].apply(change_to_daily
cumulative=0
vaccine_daily_id["Administered_new"]=vaccine_daily_id["Administered"].apply(change_to_daily
# cases_df_id_de_daily=pd.concat([cases_df_id_de_daily_de,cases_df_id_de_daily_id])
# tot_cases_de
# vaccine_daily_de["Administered_new"].tolist()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithC
opyWarning:
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/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: SettingWithC
opyWarning:
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/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [13]:

```
rem_cols=['submission_date', 'state',
       'new_case', 'pnew_case',
       'new_death', 'pnew_death', 'created_at', 'consent_cases',
       'consent_deaths','prob_cases','conf_cases', 'conf_death', 'prob_death']
cases_df_id_de_daily_de_final=pd.concat([cases_df_id_de_daily_de,cases_df_id_de[rem_cols][c
cases_df_id_de_daily_de_final
cases_df_id_de_daily_id_final=pd.concat([cases_df_id_de_daily_id,cases_df_id_de[rem_cols][c
cases_df_id_de_daily_id_final
```

Out[13]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
45445	1587	41	2020-04-15	ID	101	80.0	2	
37699	22	0	2020-04-16	ID	22	1.0	0	
5527	46	2	2020-04-17	ID	46	8.0	2	
5994	13	1	2020-04-18	ID	13	2.0	1	
41733	4	1	2020-04-19	ID	4	0.0	1	
...	...	...	...	...	...	...	...	
3201	303	2	2022-05-09	ID	303	70.0	2	
673	0	0	2022-05-10	ID	0	0.0	0	
3886	381	0	2022-05-11	ID	381	112.0	0	
6263	0	0	2022-05-12	ID	0	0.0	0	
1356	373	3	2022-05-13	ID	373	97.0	3	

759 rows × 15 columns

In [14]:

```
cases_df_id_de_daily_id_final["tot_cases_old"]=cases_df_id_de["tot_cases"][cases_df_id_de.s
cases_df_id_de_daily_id_final["tot_death_old"]=cases_df_id_de["tot_death"][cases_df_id_de.s
cases_df_id_de_daily_de_final["tot_cases_old"]=cases_df_id_de["tot_cases"][cases_df_id_de.s
cases_df_id_de_daily_de_final["tot_death_old"]=cases_df_id_de["tot_death"][cases_df_id_de.s
cases_df_id_de_daily_id_final
```

Out[14]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
45445	1587	41	2020-04-15	ID	101	80.0	2	
37699	22	0	2020-04-16	ID	22	1.0	0	
5527	46	2	2020-04-17	ID	46	8.0	2	
5994	13	1	2020-04-18	ID	13	2.0	1	
41733	4	1	2020-04-19	ID	4	0.0	1	
...	...	...	...	...	...	...	...	
3201	303	2	2022-05-09	ID	303	70.0	2	
673	0	0	2022-05-10	ID	0	0.0	0	
3886	381	0	2022-05-11	ID	381	112.0	0	
6263	0	0	2022-05-12	ID	0	0.0	0	
1356	373	3	2022-05-13	ID	373	97.0	3	

759 rows × 17 columns

In [15]:

```
vaccines_df_id=vaccine_daily_id
vaccines_df_de=vaccine_daily_de
```

##Data Cleaning Issues

The data contained missing values. We dealt with this by removing these values.

The format of date was not consistent in the dataset so we use pd.to_datetime() to convert them to the same format.

We sort the data by date in order to covert data from cumulative to daily stats

Outlier detection using Tukey's rule

In [16]:

```
def outlier_detection(X, column_name):
  values = sorted(X[column_name])
  Q1 = values[int(np.ceil(len(values) / 4))]
  Q3 = values[int(np.ceil(3 * len(values) / 4))]

  iqr = float(Q3 - Q1)
  outliers = X.loc[(X[column_name] > (Q3 + (1.5 * iqr))) | (X[column_name] < (Q1 - (1.5 * i
  return outliers.index
```

Performing outlier detection on the Cases dataset

###Since we only perform outlier detection on numerical columns, we create a new list of numerical columns and the perform outlier detection for each column.

In [17]:

```
column_names = list(cases_df_id_de_daily_de_final.columns)
```

In [18]:

```
column_names.remove('submission_date')
column_names.remove('state')
column_names.remove('created_at')
column_names.remove('consent_cases')
column_names.remove('consent_deaths')
```

In [19]:

```
cases_df_id_de_daily_de_final
```

Out[19]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
3054	0	0	2020-01-22	DE	0	0.0	0	
974	0	0	2020-01-23	DE	0	0.0	0	
7903	0	0	2020-01-24	DE	0	0.0	0	
533	0	0	2020-01-25	DE	0	0.0	0	
41165	0	0	2020-01-26	DE	0	0.0	0	
...	...	...	...	...	...	...	...	
38501	1096	0	2022-05-09	DE	1096	166.0	0	
6554	235	0	2022-05-10	DE	235	16.0	0	
6349	0	0	2022-05-11	DE	0	0.0	0	
43898	966	0	2022-05-12	DE	966	106.0	0	
6672	592	3	2022-05-13	DE	592	80.0	3	

843 rows × 17 columns

In [20]:

```
total_outliers = []
# Outliers for Delaware state features in the Cases dataset
for c in column_names:
  outliers = outlier_detection(cases_df_id_de_daily_de_final, c)
  total_outliers.extend(outliers)
total_outliers = set(total_outliers)
print(len(total_outliers))
```

```
188
```

In [21]:

```
desired_index = [i for i in cases_df_id_de_daily_de_final.index if i not in total_outliers]
len(desired_index)
cases_df_id_de_daily_de_without_outliers = cases_df_id_de_daily_de_final.loc[desired_index]
```

In [22]:

```
cases_df_id_de_daily_de_without_outliers
```

Out[22]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
3054	0	0	2020-01-22	DE	0	0.0	0	
974	0	0	2020-01-23	DE	0	0.0	0	
7903	0	0	2020-01-24	DE	0	0.0	0	
533	0	0	2020-01-25	DE	0	0.0	0	
41165	0	0	2020-01-26	DE	0	0.0	0	
...	...	...	...	...	...	...	...	
4730	0	0	2022-05-07	DE	0	0.0	0	
8848	0	0	2022-05-08	DE	0	0.0	0	
6554	235	0	2022-05-10	DE	235	16.0	0	
6349	0	0	2022-05-11	DE	0	0.0	0	
6672	592	3	2022-05-13	DE	592	80.0	3	

655 rows × 17 columns

In [23]:

```
# Outliers for Idaho state features in the Cases dataset
total_outliers = []
for c in column_names:
  outliers = outlier_detection(cases_df_id_de_daily_id_final, c)
  total_outliers.extend(outliers)
total_outliers = set(total_outliers)
print(len(total_outliers))
```

```
122
```

In [24]:

```
desired_index = [i for i in cases_df_id_de_daily_id_final.index if i not in total_outliers]
len(desired_index)
cases_df_id_de_daily_id_without_outliers = cases_df_id_de_daily_id_final.loc[desired_index]
```

In [25]:

```
cases_df_id_de_daily_id_final
```

Out[25]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
45445	1587	41	2020-04-15	ID	101	80.0	2	
37699	22	0	2020-04-16	ID	22	1.0	0	
5527	46	2	2020-04-17	ID	46	8.0	2	
5994	13	1	2020-04-18	ID	13	2.0	1	
41733	4	1	2020-04-19	ID	4	0.0	1	
...	...	...	...	...	...	...	...	
3201	303	2	2022-05-09	ID	303	70.0	2	
673	0	0	2022-05-10	ID	0	0.0	0	
3886	381	0	2022-05-11	ID	381	112.0	0	
6263	0	0	2022-05-12	ID	0	0.0	0	
1356	373	3	2022-05-13	ID	373	97.0	3	

759 rows × 17 columns

In [26]:

```
cases_df_id_de_daily_id_without_outliers
```

Out[26]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_
5527	46	2	2020-04-17	ID	46	8.0	2	
36837	64	3	2020-04-20	ID	64	7.0	3	
4432	30	3	2020-04-21	ID	30	9.0	3	
8847	36	3	2020-04-22	ID	36	7.0	3	
941	34	0	2020-04-23	ID	34	1.0	0	
...	...	...	...	...	...	...	...	
3201	303	2	2022-05-09	ID	303	70.0	2	
673	0	0	2022-05-10	ID	0	0.0	0	
3886	381	0	2022-05-11	ID	381	112.0	0	
6263	0	0	2022-05-12	ID	0	0.0	0	
1356	373	3	2022-05-13	ID	373	97.0	3	

637 rows × 17 columns

In [27]:

```
cases_df_id_de_daily_de_without_outliers.to_csv('delaware.csv',index=False)
cases_df_id_de_daily_id_without_outliers.to_csv('idaho.csv',index=False)
```

In [28]:

```
cases_df_id_de_daily_de_final = cases_df_id_de_daily_de_without_outliers
cases_df_id_de_daily_id_final = cases_df_id_de_daily_id_without_outliers
```

Outlier detection Inference for the Cases dataset

For the Cases dataset, we get 188 rows as outliers from the 843 rows corresponding to the Delaware state. We get 122 rows as outliers from the 759 rows corresponding to the Idaho state.

Performing outlier detection for Vaccine dataset

###Since directly applying outlier detection to all columns was resulting in a major loss of information, we are performing the detection on just the columns which are planning to use for our exploratory analysis.

In [29]:

```
column_names = ['Distributed', 'Administered', 'Series_Complete_Yes']
```

In [30]:

```
# Outliers for Idaho state features in the Vaccines dataset
total_outliers = []
for c in column_names:
  outliers = outlier_detection(vaccines_df_id, c)
  total_outliers.extend(outliers)
total_outliers = set(total_outliers)
print(len(total_outliers))
```

```
0
```

In [31]:

```
desired_index = [i for i in vaccines_df_id.index if i not in total_outliers]
len(desired_index)
vaccines_df_id_without_outliers = vaccines_df_id.loc[desired_index]
```

In [33]:

```
vaccines_df_id_without_outliers
```

Out[33]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna	Di
33414	2020-12-14	51	ID	1950	0	0	
33310	2020-12-15	51	ID	2925	0	0	
33289	2020-12-16	51	ID	2925	0	0	
33230	2020-12-17	51	ID	13650	0	0	
33175	2020-12-18	51	ID	13650	0	0	
...	...	...	...	...	...	...	
293	2022-05-11	19	ID	3430390	157400	1341580	
240	2022-05-12	19	ID	3437290	157500	1345480	
188	2022-05-13	19	ID	3440390	157600	1347580	
77	2022-05-14	19	ID	3445790	157600	1351080	
24	2022-05-15	20	ID	3445790	157600	1351080	

518 rows × 83 columns

In [34]:

```
# Outliers for Delaware state features in the Vaccines dataset
total_outliers = []
for c in column_names:
  outliers = outlier_detection(vaccines_df_de, c)
  total_outliers.extend(outliers)
total_outliers = set(total_outliers)
print(len(total_outliers))
```

```
0
```

In [35]:

```
desired_index = [i for i in vaccines_df_de.index if i not in total_outliers]
len(desired_index)
vaccines_df_de_without_outliers = vaccines_df_de.loc[desired_index]
```

In [36]:

```
vaccines_df_de_without_outliers
```

Out[36]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna	Di
33389	2020-12-14	51	DE	975	0	0	
33364	2020-12-15	51	DE	975	0	0	
33294	2020-12-16	51	DE	8775	0	0	
33221	2020-12-17	51	DE	8775	0	0	
33151	2020-12-18	51	DE	8775	0	0	
...	...	...	...	...	...	...	
281	2022-05-11	19	DE	2379655	99600	922500	
230	2022-05-12	19	DE	2388255	100100	928500	
141	2022-05-13	19	DE	2398555	100100	929000	
88	2022-05-14	19	DE	2401655	100100	929800	
58	2022-05-15	20	DE	2401655	100100	929800	

518 rows × 83 columns

Outlier detection Inference for the Vaccines dataset

For the rows corresponding to the Idaho and Delaware state, we get no outliers for the columns mentioned above.

In [37]:

```
cases_df_id_de_daily_de_without_outliers.to_csv('delaware_cases.csv',index=False)
cases_df_id_de_daily_id_without_outliers.to_csv('idaho_cases.csv',index=False)
vaccines_df_de_without_outliers.to_csv('delaware_vaccines.csv',index=False)
vaccines_df_id_without_outliers.to_csv('idaho_vaccines.csv',index=False)
```

In [ ]:

```

```

Mandatory Part - Q2 (a)

In [146]:
```
import pandas as pd
import numpy as np
import math
```

In [147]:
```
# loading preprocessed state-wise cases & deaths datasets
delaware = pd.read_csv("/Users/meet/Desktop/544_project/Mandatory_data
idaho = pd.read_csv("/Users/meet/Desktop/544_project/Mandatory_dataset
```

In [148]:
```
idaho.head()
```

Out[148]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_deat
0	46	2	2020-04-17	ID	46	8.0	2	0.
1	64	3	2020-04-20	ID	64	7.0	3	0.
2	30	3	2020-04-21	ID	30	9.0	3	1.
3	36	3	2020-04-22	ID	36	7.0	3	1.
4	34	0	2020-04-23	ID	34	1.0	0	0.

Filtering & Preprocessing

Here, we are filtering the Covid cases & deaths dataset for the states Idaho and Delaware as required in the question.

Filtering Idaho state's cases and deaths dataset to retrieve February 2021 data only.

In [149]:
```
# filter idaho dataset for February 2021
idaho["submission_date"] = pd.to_datetime(idaho["submission_date"])
idaho_feb = idaho[(idaho["submission_date"].dt.month == 2) & (idaho["s
idaho_feb.head()
```

Out[149]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_de
251	482	10	2021-02-01	ID	482	102.0	10	
252	491	6	2021-02-02	ID	491	111.0	6	
253	507	7	2021-02-03	ID	507	114.0	7	
254	402	-1	2021-02-04	ID	402	94.0	-1	
255	404	11	2021-02-05	ID	404	78.0	11	

In [150]:
```
len(idaho_feb)
```

Out[150]: 28

Filtering Idaho state's cases and deaths dataset to retrieve March 2021 data only.

In [151]:
```
# filter idaho dataset for March 2021
idaho["submission_date"] = pd.to_datetime(idaho["submission_date"])
idaho_march = idaho[(idaho["submission_date"].dt.month == 3) & (idaho[
idaho_march.head()
```

Out[151]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_de
279	322	7	2021-03-01	ID	322	89.0	7	
280	374	4	2021-03-02	ID	374	108.0	4	
281	452	5	2021-03-03	ID	452	107.0	5	
282	299	0	2021-03-04	ID	299	95.0	0	
283	240	0	2021-03-05	ID	240	98.0	0	

In [152]:
```
len(idaho_march)
```

Out[152]: 30

Filtering Delaware state's cases and deaths dataset to retrieve February 2021 data only.

In [153]:
```
# filter delaware dataset for February 2021
delaware["submission_date"] = pd.to_datetime(delaware["submission_date
delaware_feb = delaware[(delaware["submission_date"].dt.month == 2) &
delaware_feb.head()
```

Out[153]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_de
309	251	5	2021-02-04	DE	251	32.0	5	
310	362	0	2021-02-08	DE	362	35.0	0	
311	385	7	2021-02-12	DE	385	58.0	7	
312	344	1	2021-02-14	DE	344	38.0	1	
313	252	0	2021-02-15	DE	252	27.0	0	

In [154]:
```
len(delaware_feb)
```

Out[154]: 10

Filtering Idaho state's cases and deaths dataset to retrieve March 2021 data only.

In [155]:
```
# filter delaware dataset for February 2021
delaware["submission_date"] = pd.to_datetime(delaware["submission_date
delaware_march = delaware[(delaware["submission_date"].dt.month == 3)
delaware_march.head()
```

Out[155]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_de
319	290	0	2021-03-01	DE	290	22.0	0	
320	136	4	2021-03-02	DE	136	16.0	4	
321	229	4	2021-03-04	DE	229	29.0	4	
322	219	9	2021-03-05	DE	219	6.0	9	
323	232	9	2021-03-07	DE	232	16.0	9	

In [156]:
```
len(delaware_march)
```

Out[156]: 24

Wald's Test

In [161]:
```
import pandas as pd
import numpy as np
import collections
import csv
import math
import warnings
warnings.filterwarnings('ignore')
```

In [162]:
```
# Reading CSV
vaccines = pd.read_csv('/Users/meet/Desktop/544_project/Mandatory_data
```

In [163]:
```
vaccines.head()
```

Out[163]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna
0	05/15/2022	20	PR	7552350	215000	2662120
1	05/15/2022	20	KS	6121515	256400	2354940
2	05/15/2022	20	VA	19949085	785300	7108700
3	05/15/2022	20	MT	2004895	105200	825200
4	05/15/2022	20	IH2	2965895	108400	1311680

5 rows × 82 columns

Wald's one sample test Function

In [164]:
```
# Function for one sample Wald's test
def walds(df,col):
    df["submission_date"] = pd.to_datetime(df["submission_date"])
    df_feb = df[(df["submission_date"].dt.month == 2) & (df["submissio
    df_mar = df[(df["submission_date"].dt.month == 3) & (df["submissio
    guess = df_feb[col].mean()

    se = np.sqrt(df_mar[col].mean()/len(df_mar))


    theta_cap = df_mar[col].mean() #for poisson's distribution MLE is

    w = (theta_cap - guess)/se

    print('w statistic is ',w)

    # alpha = 0.05

    z_a2 = 1.96

    if abs(w)>z_a2:
        print('We reject the null hypothesis as the absolute value of
    else:
        print('We accept the null hypothesis as the absolute value of
```

In [165]:
```
#code
print("Results of Wald's one sample test for cases for DE state")
walds(delaware,'tot_cases')
print('\n')
print("Results of Wald's one sample test for cases for ID state")
walds(idaho,'tot_cases')
print('\n')
print("Results of Wald's one sample test for deaths for DE state")
walds(delaware,'tot_death')
print('\n')
print("Results of Wald's one sample test for deaths for ID state")
walds(idaho,'tot_death')
print('\n')
```

```
Results of Wald's one sample test for cases for DE state
w statistic is  -24.805591076189682
We reject the null hypothesis as the absolute value of Wald's stastic
is greater than the Z value(at 0.025) = 1.96


Results of Wald's one sample test for cases for ID state
w statistic is  -0.1584623630185989
We accept the null hypothesis as the absolute value of Wald's stastic
is lesser than the Z value(at 0.025) = 1.96


Results of Wald's one sample test for deaths for DE state
w statistic is  2.0948917462655263
We reject the null hypothesis as the absolute value of Wald's stastic
is greater than the Z value(at 0.025) = 1.96


Results of Wald's one sample test for deaths for ID state
w statistic is  -5.89813106089253
We reject the null hypothesis as the absolute value of Wald's stastic
is greater than the Z value(at 0.025) = 1.96
```

Results of Wald's one sample test for mean of cases and deaths

Null hypothesis (H0):

Mean of cases/deaths in the month of March'21 is equal to the mean of cases/deaths in the month of February'21

Alternate hypothesis (H1):

Mean of cases/deaths in the month of March'21 is NOT equal to the mean of cases/deaths in the month of February'21

Result:

Results of Wald's one sample test for cases for DE state:

We reject the null hypothesis as the absolute value of Wald's stastic is greater than the Z value(at 0.025) = 1.96

Results of Wald's one sample test for cases for ID state:

We accept the null hypothesis as the absolute value of Wald's stastic is lesser than the Z value(at 0.025) = 1.96

Results of Wald's one sample test for deaths for DE state:

We reject the null hypothesis as the absolute value of Wald's stastic is greater than the Z value(at 0.025) = 1.96

Results of Wald's one sample test for deaths for ID state:

We reject the null hypothesis as the absolute value of Wald's stastic is greater than the Z value(at 0.025) = 1.96

Is the test applicable:

We have used MLE as estimator which is Asymptotically normal (as n goes to infinity). Since this is the required condition, the test is applicable.

Wald's two sample test Function

In [166]:
```
# Function for two sample Wald's test
def walds_twot(df,col):
    df["submission_date"] = pd.to_datetime(df["submission_date"])
    df_feb = df[(df["submission_date"].dt.month == 2) & (df["submissio
    df_mar = df[(df["submission_date"].dt.month == 3) & (df["submissio
    theta1 = df_feb[col].mean()
    theta2 = df_mar[col].mean()

    theta_cap = theta1 - theta2

    guess = 0

    se_cap = np.sqrt(df_feb[col].mean() /len(df_feb) + df_mar[col].mea

    w = (theta_cap - guess)/se_cap

    print('w statistic is ',w)

    # alpha = 0.05

    z_a2 = 1.96

    if abs(w)>z_a2:
        print('We reject the null hypothesis as the absolute value of
    else:
        print('We accept the null hypothesis as the absolute value of
```

In [167]:
```
print("Results of Wald's two sample test for cases for DE state")
walds_twot(delaware,'tot_cases')
print('\n')
print("Results of Wald's two sample test for cases for ID state")
walds_twot(idaho,'tot_cases')
print('\n')
print("Results of Wald's two sample test for deaths for DE state")
walds_twot(delaware,'tot_death')
print('\n')
print("Results of Wald's two sample test for deaths for ID state")
walds_twot(idaho,'tot_death')
print('\n')
```

```
Results of Wald's two sample test for cases for DE state
w statistic is  12.089401682022904
We reject the null hypothesis as the absolute value of Wald's stastic
is greater than the Z value(at 0.025) = 1.96


Results of Wald's two sample test for cases for ID state
w statistic is  0.1100535602293159
We accept the null hypothesis as the absolute value of Wald's stastic
is lesser than the Z value(at 0.025) = 1.96


Results of Wald's two sample test for deaths for DE state
w statistic is  -1.2405958875929843
We accept the null hypothesis as the absolute value of Wald's stastic
is lesser than the Z value(at 0.025) = 1.96


Results of Wald's two sample test for deaths for ID state
w statistic is  3.5623551365738373
We reject the null hypothesis as the absolute value of Wald's stastic
is greater than the Z value(at 0.025) = 1.96
```

Results of Wald's two sample test for mean of cases and deaths

Null hypothesis (H0):

Mean of cases/deaths in the month of March'21 is equal to the mean of cases/deaths in the month of February'21

Alternate hypothesis (H1):

Mean of cases/deaths in the month of March'21 is NOT equal to the mean of cases/deaths in the month of February'21

Result:

Results of Wald's two sample test for cases for DE state:

We reject the null hypothesis as the absolute value of Wald's stastic is greater than the Z value(at 0.025) = 1.96

Results of Wald's two sample test for cases for ID state:

We accept the null hypothesis as the absolute value of Wald's stastic is lesser than the Z value(at 0.025) = 1.96

Results of Wald's two sample test for deaths for DE state:

We accept the null hypothesis as the absolute value of Wald's stastic is lesser than the Z value(at 0.025) = 1.96

Results of Wald's two sample test for deaths for ID state:

We reject the null hypothesis as the absolute value of Wald's stastic is greater than the Z value(at 0.025) = 1.96

Is the test applicable:

We have used MLE as estimator which is Asymptotically normal (as n goes to infinity). Since this is the required condition, the test is applicable.

Z - Test

Performing Z-Test on the filtered data for February and March, as asked in the question

Z - Test on Daily Total Cases Recorded in Idaho

In [157]:
```
# z-test on idaho daily total cases
print("Null Hypothesis H0: Mean of daily cases for March 2021 == Mean
print("Alternate Hypothesis H1: Mean of daily cases for March 2021 !=

sample_mean_feb_idaho = np.mean(idaho_feb["tot_cases"])
sample_mean_march_idaho = np.mean(idaho_march["tot_cases"])
sample_mean_full_idaho = np.mean(idaho["tot_cases"])

denom_cases = np.sqrt((np.sum(np.square(idaho["tot_cases"] - sample_me

numerator_cases = abs(sample_mean_march_idaho - sample_mean_feb_idaho)

z_stat_cases = numerator_cases / denom_cases

z_threshold = 1.96
print()
if z_stat_cases > z_threshold:
    print("Z statistic = ", z_stat_cases, "is greater than Z alpha/2 =
    print("Hence, Null Hypothesis H0 is rejected and Alternate Hypothe
    print("i.e. Mean of daily cases for March 2021 != Mean of daily ca
else:
    print("Z statistic = ", z_stat_cases, "is less than Z alpha/2 = 1.
    print("Hence, Null Hypothesis H0 is accepted,")
    print("i.e. Mean of daily cases for March 2021 == Mean of daily ca
```

```
Null Hypothesis H0: Mean of daily cases for March 2021 == Mean of dai
ly cases for February 2021
Alternate Hypothesis H1: Mean of daily cases for March 2021 != Mean o
f daily cases for February 2021

Z statistic =  0.006766790033978814 is less than Z alpha/2 = 1.96
Hence, Null Hypothesis H0 is accepted,
i.e. Mean of daily cases for March 2021 == Mean of daily cases for Fe
bruary 2021
```

Z - Test on Daily Total Deaths Recorded in Idaho

In [158]:
```
# z-test on idaho daily total deaths

print("Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean
print("Alternate Hypothesis H1: Mean of daily deaths for March 2021 !=

sample_mean_feb_idaho_death = np.mean(idaho_feb["tot_death"])
sample_mean_march_idaho_death = np.mean(idaho_march["tot_death"])
sample_mean_full_idaho_death = np.mean(idaho["tot_death"])

denom_deaths = np.sqrt((np.sum(np.square(idaho["tot_death"] - sample_m

numerator_deaths = abs(sample_mean_march_idaho_death - sample_mean_feb

z_stat_deaths = numerator_deaths / denom_deaths

print()
if z_stat_deaths > z_threshold:
    print("Z statistic = ", z_stat_deaths, "is greater than Z alpha/2
    print("Hence, Null Hypothesis H0 is rejected and Alternate Hypothe
    print("i.e. Mean of daily deaths for March 2021 != Mean of daily d
else:
    print("Z statistic = ", z_stat_deaths, "is less than Z alpha/2 = 1
    print("Hence, Null Hypothesis H0 is accepted,")
    print("i.e. Mean of daily deaths for March 2021 == Mean of daily d
```

```
Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean of da
ily deaths for February 2021
Alternate Hypothesis H1: Mean of daily deaths for March 2021 != Mean
of daily deaths for February 2021

Z statistic =  2.049002692157555 is greater than Z alpha/2 = 1.96
Hence, Null Hypothesis H0 is rejected and Alternate Hypothesis H1 is
accepted,
i.e. Mean of daily deaths for March 2021 != Mean of daily deaths for
February 2021
```

Z - Test on Daily Total Cases Recorded in Delaware

In [159]:
```
# z-test on delaware cases feb-march

print("Null Hypothesis H0: Mean of daily cases for March 2021 == Mean
print("Alternate Hypothesis H1: Mean of daily cases for March 2021 !=

sample_mean_feb_delaware = np.mean(delaware_feb["tot_cases"])
sample_mean_march_delaware = np.mean(delaware_march["tot_cases"])
sample_mean_full_delaware = np.mean(delaware["tot_cases"])

denom_delaware = np.sqrt((np.sum(np.square(delaware["tot_cases"] - sam

numerator_delaware = abs(sample_mean_march_delaware - sample_mean_feb_

z_stat_delaware = numerator_delaware / denom_delaware

print()
if z_stat_delaware > z_threshold:
    print("Z statistic = ", z_stat_delaware, "is greater than Z alpha/
    print("Hence, Null Hypothesis H0 is rejected and Alternate Hypothe
    print("i.e. Mean of daily cases for March 2021 != Mean of daily ca
else:
    print("Z statistic = ", z_stat_delaware, "is less than Z alpha/2 =
    print("Hence, Null Hypothesis H0 is accepted,")
    print("i.e. Mean of daily cases for March 2021 == Mean of daily ca
```

```
Null Hypothesis H0: Mean of daily cases for March 2021 == Mean of dai
ly cases for February 2021
Alternate Hypothesis H1: Mean of daily cases for March 2021 != Mean o
f daily cases for February 2021

Z statistic =  1.9501906849818513 is less than Z alpha/2 = 1.96
Hence, Null Hypothesis H0 is accepted,
i.e. Mean of daily cases for March 2021 == Mean of daily cases for Fe
bruary 2021
```

Z - Test on Daily Total Deaths Recorded in Delaware

In [160]:
```
# z-test on delaware deaths feb-march

print("Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean
print("Alternate Hypothesis H1: Mean of daily deaths for March 2021 !=

sample_mean_feb_delaware_death = np.mean(delaware_feb["tot_death"])
sample_mean_march_delaware_death = np.mean(delaware_march["tot_death"]
sample_mean_full_delaware_death = np.mean(delaware["tot_death"])

denom_delaware_deaths = np.sqrt((np.sum(np.square(delaware["tot_death"

numerator_delaware_deaths = abs(sample_mean_march_delaware_death - sam

z_stat_delaware_deaths = numerator_delaware_deaths / denom_delaware_de

print()
if z_stat_delaware_deaths > z_threshold:
    print("Z statistic = ", z_stat_delaware_deaths, "is greater than Z
    print("Hence, Null Hypothesis H0 is rejected and Alternate Hypothe
    print("i.e. Mean of daily deaths for March 2021 != Mean of daily d
else:
    print("Z statistic = ", z_stat_delaware_deaths, "is less than Z al
    print("Hence, Null Hypothesis H0 is accepted,")
    print("i.e. Mean of daily deaths for March 2021 == Mean of daily d
```

```
Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean of da
ily deaths for February 2021
Alternate Hypothesis H1: Mean of daily deaths for March 2021 != Mean
of daily deaths for February 2021

Z statistic =  1.6950732971676468 is less than Z alpha/2 = 1.96
Hence, Null Hypothesis H0 is accepted,
i.e. Mean of daily deaths for March 2021 == Mean of daily deaths for
February 2021
```

Results for one sample Z - Test on daily cases and deaths in the states Idaho and Delaware for the months February and March 2021

1. Results for one sample Z - Test on daily total cases recorded in Idaho in February-March 2021

Null Hypothesis H0: Mean of daily cases for March 2021 == Mean of daily cases for February 2021 **Alternate Hypothesis H1:** Mean of daily cases for March 2021 != Mean of daily cases for February 2021

It can be inferred from the acquired **z-statistic (= 0.068)** that the Null Hypothesis **H0 will be accepted** as the stat is less than the threshold (z-alpha/2 = 1.96).

Hence, The mean of daily cases recorded in the month of February 2021 in Idaho is **similar** to that recorded in the month of March 2021.

2. Results for one sample Z - Test on daily total deaths recorded in Idaho in February-March 2021

Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean of daily deaths for February 2021 **Alternate Hypothesis H1:** Mean of daily deaths for March 2021 != Mean of daily deaths for February 2021

It can be inferred from the acquired **z-statistic (= 2.05)** that the Null Hypothesis **H0 will NOT be accepted** as the stat is greater than the threshold (z-alpha/2 = 1.96).

Hence, The mean of daily deaths recorded in the month of February 2021 in Idaho is **NOT similar** to that recorded in the month of March 2021.

3. Results for one sample Z - Test on daily total cases recorded in Delaware in February-March 2021

Null Hypothesis H0: Mean of daily cases for March 2021 == Mean of daily cases for February 2021 **Alternate Hypothesis H1:** Mean of daily cases for March 2021 != Mean of daily cases for February 2021

It can be inferred from the acquired **z-statistic (= 1.95)** that the Null Hypothesis **H0 will be accepted** as the stat is less than (very close but still less) the threshold (z-alpha/2 = 1.96).

Hence, The mean of daily cases recorded in the month of February 2021 in Delaware is **similar** to that recorded in the month of March 2021.

4. Results for one sample Z - Test on daily total deaths recorded in Delaware in February-March 2021

Null Hypothesis H0: Mean of daily deaths for March 2021 == Mean of daily deaths for February 2021 **Alternate Hypothesis H1:** Mean of daily deaths for March 2021 != Mean of daily deaths for February 2021

It can be inferred from the acquired **z-statistic (= 1.69)** that the Null Hypothesis **H0 will be accepted** as the stat is less than (very close but still less) the threshold (z-alpha/2 = 1.96).

Hence, The mean of daily deaths recorded in the month of February 2021 in Delaware is **similar** to that recorded in the month of March 2021.

Z - Stat Formula

Z = (Xbar - Mu0) / (sigma / sqrt(n)) Here, we have used the mean cases/deaths of February 2021 as the guessed mean for verifying the Z-Test on mean cases/deaths for March 2021. Additionally, for calculating the standard deviation, we are considering the entire state-wise data instead of using the month-wise seperated data.

Is Z - Test applicable?

Here, we are calculating the true variance for the entire state's data whereas performing the Z-Test on February-March months only, where the number of rows (n) in all 4 of the cases is <=30. Since n is small (<=30) here, Z-Test will not be reliable / applicable for this scenario.

T test

One sample T test Function

In [168]:
```
# Function for one sample T test
def ttest(df,col):
    df["submission_date"] = pd.to_datetime(df["submission_date"])
    df_feb = df[(df["submission_date"].dt.month == 2) & (df["submission
    df_mar = df[(df["submission_date"].dt.month == 3) & (df["submission
    n = 31
    guess = df_feb[col].mean()

    mar_mean = df_mar[col].mean()
    root_se = np.sqrt(np.sum(np.square(df_mar[col]-mar_mean))/len(df_ma


    denominator = (root_se/math.sqrt(n))
    t_statistic = (mar_mean - guess)/denominator


    print('t statistic is ',t_statistic)

    # alpha = 0.05

    t_tab_val = 2.042

    if abs(t_statistic)>t_tab_val:
        print('We REJECT the null hypothesis as the absolute value of T
    else:
        print('We ACCEPT the null hypothesis as the absolute value of T
```

In [169]:
```
print("Results of one sample T test for cases for DE state")
ttest(delaware,'tot_cases')
print('\n')
print("Results of one sample T test for cases for ID state")
ttest(idaho,'tot_cases')
print('\n')
print("Results of one sample T test for deaths for DE state")
ttest(delaware,'tot_death')
print('\n')
print("Results of one sample T test for deaths for ID state")
ttest(idaho,'tot_death')
```

```
Results of one sample T test for cases for DE state
t statistic is  -6.26153336851101
We REJECT the null hypothesis as the absolute value of T stastic is g
reater than T(30,0.025) = 2.042


Results of one sample T test for cases for ID state
t statistic is  -0.019648179891557177
We ACCEPT the null hypothesis as the absolute value of T stastic is g
reater than T(30,0.025) = 2.042


Results of one sample T test for deaths for DE state
t statistic is  1.5914666058220657
We ACCEPT the null hypothesis as the absolute value of T stastic is g
reater than T(30,0.025) = 2.042


Results of one sample T test for deaths for ID state
t statistic is  -3.77112175944854
We REJECT the null hypothesis as the absolute value of T stastic is g
reater than T(30,0.025) = 2.042
```

Results of one sample T test for mean of cases and deaths

Null hypothesis (H0):

Mean of cases/deaths in the month of March'21 is equal to the mean of cases/deaths in the month of February'21

Alternate hypothesis (H1):

Mean of cases/deaths in the month of March'21 is NOT equal to the mean of cases/deaths in the month of February'21

Result:

Results of one sample T test for cases for DE state

We REJECT the null hypothesis as the absolute value of T stastic is greater than T(30,0.025) = 2.042

Results of one sample T test for cases for ID state

We ACCEPT the null hypothesis as the absolute value of T stastic is greater than T(30,0.025) = 2.042

Results of one sample T test for deaths for DE state

We ACCEPT the null hypothesis as the absolute value of T stastic is greater than T(30,0.025) = 2.042

Results of one sample T test for deaths for ID state

We REJECT the null hypothesis as the absolute value of T stastic is greater than T(30,0.025) = 2.042

Is the test applicable:

T test is applicable as the number of samples is high, using CLT, the mean becomes Asymptotically normal.

Two sample T test Function

In [170]:
```
# Function for two sample T test
def ttest_twot(df,col):
    df["submission_date"] = pd.to_datetime(df["submission_date"])
    df_feb = df[(df["submission_date"].dt.month == 2) & (df["submissio
    df_mar = df[(df["submission_date"].dt.month == 3) & (df["submissio
    guess = 0

    numerator = df_feb[col].mean()-df_mar[col].mean()

    corrected_se_feb = np.sum(np.square(df_feb[col]-df_feb[col].mean()
    corrected_se_feb = corrected_se_feb/(len(df_feb)-1)
    corrected_se_mar = np.sum(np.square(df_mar[col]-df_mar[col].mean()
    corrected_se_mar = corrected_se_mar/(len(df_mar)-1)


    denominator = np.sqrt((corrected_se_feb/len(df_feb))+(corrected_se
    t_statistic = numerator/denominator


    print('t statistic is ',t_statistic)

    # alpha = 0.05

    t_tab_val = 2.0025

    if abs(t_statistic)>t_tab_val:
        print('We REJECT the null hypothesis as the absolute value of
    else:
        print('We ACCEPT the null hypothesis as the absolute value of
```

In [171]:
```
print('Results of two sample T test for cases for DE state')
ttest_twot(delaware,'tot_cases')
print('\n')
print('Results of two sample T test for deaths for ID state')
ttest_twot(idaho,'tot_cases')
print('\n')
print('Results of two sample T test for cases for DE state')
ttest_twot(delaware,'tot_death')
print('\n')
print('Results of two sample T test for deaths for ID state')
ttest_twot(idaho,'tot_death')
```

```
Results of two sample T test for cases for DE state
t statistic is  2.4220780179024572
We REJECT the null hypothesis as the absolute value of T stastic is g
reater than T(57,0.025) = 2.0025


Results of two sample T test for deaths for ID state
t statistic is  0.012242143269691952
We ACCEPT the null hypothesis as the absolute value of T stastic is g
reater than T(57,0.025) = 2.0025


Results of two sample T test for cases for DE state
t statistic is  -0.6967059541905922
We ACCEPT the null hypothesis as the absolute value of T stastic is g
reater than T(57,0.025) = 2.0025


Results of two sample T test for deaths for ID state
t statistic is  1.7578410010137446
We ACCEPT the null hypothesis as the absolute value of T stastic is g
reater than T(57,0.025) = 2.0025
```

Results of two sample T test for mean of cases and deaths

Null hypothesis (H0):

Mean of cases/deaths in the month of March'21 is equal to the mean of cases/deaths in the month of February'21

Alternate hypothesis (H1):

Mean of cases/deaths in the month of March'21 is NOT equal to the mean of cases/deaths in the month of February'21

Result:

Results of two sample T test for cases for DE state:

We REJECT the null hypothesis as the absolute value of T stastic is greater than T(57,0.025) = 2.0025

Results of two sample T test for deaths for ID state:

We ACCEPT the null hypothesis as the absolute value of T stastic is greater than T(57,0.025) = 2.0025

Results of two sample T test for cases for DE state:

We ACCEPT the null hypothesis as the absolute value of T stastic is greater than T(57,0.025) = 2.0025

Results of two sample T test for deaths for ID state:

We ACCEPT the null hypothesis as the absolute value of T stastic is greater than T(57,0.025) = 2.0025

Is the test applicable:

T test is applicable as the number of samples is high, using CLT, the mean becomes Asymptotically normal.

In [ ]:

In [187]:

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.stats import geom
from scipy.stats import binom
from itertools import permutations
import math
from statistics import mean
import random
from scipy.stats import gamma
import matplotlib.pyplot as  plt
import copy
import scipy.stats
```

In [188]:

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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, ca
ll drive.mount("/content/gdrive", force_remount=True).
```

#Import Cases and Vaccination datasets (X dataset)

In [189]:

```
cases_df_id_de_daily_de_final=pd.read_csv("/content/gdrive/MyDrive/delaware_cases.csv")
cases_df_id_de_daily_id_final=pd.read_csv("/content/gdrive/MyDrive/idaho_cases.csv")
vaccine_daily_de=pd.read_csv("/content/gdrive/MyDrive/delaware_vaccines.csv")
vaccine_daily_id=pd.read_csv("/content/gdrive/MyDrive/idaho_vaccines.csv")
```

In [190]:

```
### Question 2b
```

#Question 2b

In [191]:

```
cases_df_id_de_daily_de_final['submission_date'] = pd.to_datetime(cases_df_id_de_daily_de_f
cases_df_id_de_daily_id_final['submission_date'] = pd.to_datetime(cases_df_id_de_daily_id_f
```

#Filtering rows to get data from 1st October to 31st December

In [192]:

```
three_month_condition_de = (cases_df_id_de_daily_de_final['submission_date'] >= '2021-10-01
three_month_condition_id = (cases_df_id_de_daily_id_final['submission_date'] >= '2021-10-01
```

In [193]:

```
three_month_de_data = cases_df_id_de_daily_de_final.loc[three_month_condition_de]
three_month_id_data = cases_df_id_de_daily_id_final.loc[three_month_condition_id]
```

In [194]:

```
three_month_de_data
```

Out[194]:

	tot_cases	tot_death	submission_date	state	new_case	pnew_case	new_death	pnew_de
510	630	0	2021-10-01	DE	630	79.0	0	
511	403	0	2021-10-03	DE	403	61.0	0	
512	381	0	2021-10-04	DE	381	53.0	0	
513	270	3	2021-10-05	DE	270	23.0	3	
514	444	2	2021-10-06	DE	444	45.0	2	
...	...	...	...	...	...	...	...	
571	683	0	2021-12-20	DE	683	61.0	0	
572	541	7	2021-12-21	DE	541	35.0	7	
573	665	0	2021-12-22	DE	665	38.0	0	
574	451	0	2021-12-27	DE	451	37.0	0	
575	621	5	2021-12-28	DE	621	48.0	5	

66 rows × 17 columns

In [195]:

```
def MME(type, X):
  if type == 'poisson':
    return np.mean(X)
  elif type == 'binomial':
    m = np.mean(X)
    v = np.var(X)
    return 1 - (v/m), m / (1 - (v/m))
  elif type == 'geometric':
    return 1 / np.mean(X)
```

In [196]:

```
def generate_CDF(type, X, params):
  if type == 'poisson':
    return poisson.cdf(X, params[0])
  elif type == 'binomial':
    return binom.cdf(X, params[0], params[1])
  elif type == 'geometric':
    return geom.cdf(X, params[0])
```

In [197]:

```
def generate_eCDF(X):
  n = len(X)
  Srt = sorted(X)
  delta = .1
  X = [min(Srt)-delta]
  Y = [0]
  for i in range(0, n):
      X = X + [Srt[i], Srt[i]]
      Y = Y + [Y[len(Y)-1], Y[len(Y)-1]+(1/n)]
  X = X + [max(Srt)+delta]
  # print(X)
  Y = Y + [1]
  return X, Y
```

In [198]:

```
def ks_test_1_sample(X1,Y1, cdf_function, parameter):
    tot_max = -1
    ks_table = np.zeros((len(X1),4))
    for i in range(len(ks_table)-1):
        ks_table[i,0] = Y1[i]
        ks_table[i,1] = Y1[i+1]
        F_x = generate_CDF(cdf_function, X1[i], parameter)
        ks_table[i,2] = abs(ks_table[i,0] - F_x)
        ks_table[i,3] = abs(ks_table[i,1] - F_x)
        cmax = max(ks_table[i,2], ks_table[i,3])
        if cmax > tot_max:
            tot_max = cmax
    return tot_max
```

#1-sample KS Test for number of cases

We utilize the data from Delaware state to calculate the parameters for the distributions and then use the data from the Idaho state to see if they have the same distribution.

Geometric

In [199]:

```
p_est = MME('geometric', three_month_de_data['tot_cases'])
```

In [200]:

```
print('Geometric estimate', p_est)
```

```
Geometric estimate 0.002589250686543743
```

In [201]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_cases'])
```

In [202]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'geometric', [p_est]))
```

```
0.21967884963185352
```

#Poisson

In [203]:

```
lambda_est = MME('poisson', three_month_de_data['tot_cases'])
```

In [204]:

```
print('Poisson estimate', lambda_est)
```

```
Poisson estimate 386.2121212121212
```

In [205]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_cases'])
```

In [206]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'poisson', [lambda_est]))
```

```
0.47461437726653877
```

In [207]:

```
print('Poisson estimate', lambda_est)
```

```
Poisson estimate 386.2121212121212
```

#Binomial

In [208]:

```
n_est, p_est = MME('binomial', three_month_de_data['tot_cases'])
```

In [209]:

```
print('Binomial estimates', n_est, p_est)
```

```
Binomial estimates -74.49239036104476 -5.1845848863253545
```

In [210]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_cases'])
```

In [211]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'binomial', [n_est, p_est]))
```

```
1.0
```

Inference of 1-sample KS Test for number of cases The KS statistic suggests the the hypothesis that the samples for Idaho state have the same distribution as the one calculated using MME estimates is rejected for all the three distributions - Poisson, Geometric and Binomial.

#1-sample KS Test for number of deaths

Geometric

In [212]:

```
p_est = MME('geometric', three_month_de_data['tot_death'])
```

In [213]:

```
print('Geometric estimate', p_est)
```

```
Geometric estimate 0.6226415094339623
```

In [214]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_death'])
```

In [215]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'geometric', [p_est]))
```

```
0.5018199214414883
```

Poisson

In [216]:

```
lambda_est = MME('poisson', three_month_de_data['tot_death'])
```

In [217]:

```
lambda_est = MME('poisson', three_month_de_data['tot_death'])
```

In [218]:

```
print('Poisson estimate', lambda_est)
```

```
Poisson estimate 1.606060606060606
```

In [219]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_death'])
```

In [220]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'poisson', [lambda_est]))
```

```
0.4859155896347853
```

Binomial

In [221]:

```
n_est, p_est = MME('binomial', three_month_de_data['tot_death'])
```

In [222]:

```
print('Binomial estimates', n_est, p_est)
```

```
Binomial estimates -2.073184676958265 -0.7746828461114164
```

In [223]:

```
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_death'])
```

In [224]:

```
print(ks_test_1_sample(idaho_X, idaho_Y, 'binomial', [n_est, p_est]))
```

```
1.0
```

Inference of 1-sample KS Test for number of deaths The KS statistic suggests the the hypothesis that the samples for Idaho state have the same distribution as the one calculated using MME estimates is rejected for all the three distributions - Poisson, Geometric and Binomial.

In [225]:

```
def ks_test_2_sample(X1, Y1, X2, Y2):
    tot_max = -1
    ks_table = np.zeros((len(X1),6))
    for i in range(len(ks_table)-1):
        ks_table[i,0] = Y1[i]
        ks_table[i,1] = Y1[i+1]
        ks_table[i,2]=0
        ks_table[i,3]=0
        for j in X2:
            if j<X1[i]:
                ks_table[i,2]+=1
            if j<=X1[i]:
                ks_table[i,3]+=1

        ks_table[i,3]/=len(X2)
        ks_table[i,2]/=len(X2)

        ks_table[i,4] = abs(ks_table[i,0] - ks_table[i,2])
        ks_table[i,5] = abs(ks_table[i,1] - ks_table[i,3])
        cmax = max(ks_table[i,4], ks_table[i,5])
        if cmax > tot_max:
            tot_max = cmax
            x1_max = X1[i]
            y1_max = ks_table[i,0]
            y2_max = ks_table[i,2]
    return tot_max
```

2-sample KS Test for number of cases

In [226]:

```
delaware_X, delaware_Y = generate_eCDF(three_month_de_data['tot_cases'])
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_cases'])
```

In [227]:

```
print(ks_test_2_sample(delaware_X, delaware_Y, idaho_X, idaho_Y))
```

```
0.21141098484848486
```

Inference of 2-sample KS Test for number of cases The KS statistic suggests the the hypothesis that the samples for Idaho state have the same distribution as the samples from the state of Delaware is rejected.

2-sample KS Test for number of deaths

In [228]:

```
delaware_X, delaware_Y = generate_eCDF(three_month_de_data['tot_death'])
idaho_X, idaho_Y = generate_eCDF(three_month_id_data['tot_death'])
```

In [229]:

```
print(ks_test_2_sample(delaware_X, delaware_Y, idaho_X, idaho_Y))
```

```
0.5523200757575754
```

Inference of 2-sample KS Test for number of deaths The KS statistic suggests the the hypothesis that the samples for Idaho state have the same distribution as the samples from the state of Delaware is rejected.

Permutation Test

Total Cases

In [230]:

```
d1=list(three_month_de_data['tot_cases'])
d2=list(three_month_id_data['tot_cases'])
d=d1+d2
n=len(d1)
m=len(d2)
to=abs(mean(d1)-mean(d2))
# l = list(permutations(d))
# l=random.sample(l, 1000)
c=0
for j in range(1000):
    l=np.random.permutation(d)
    # print(l)
    d1=l[:n]
    d2=l[n:]
    ti=abs(mean(d1)-mean(d2))
    if(ti>to):
        c+=1
# print(c)
print("p-value: {}".format(c/1000))
if c/1000>0.05:
    print("Accept Null Hypothesis")
else:
    print("Reject Null Hypothesis")
```

```
p-value: 0.299
Accept Null Hypothesis
```

Since p-value obtained from the permutation test is greater than the threshhold we accept the hypothesis that the samples of total cases of Idaho and Delaware have the same distribution

Total Deaths

In [231]:

```
d1=list(three_month_de_data['tot_death'])
d2=list(three_month_id_data['tot_death'])
d=d1+d2
n=len(d1)
m=len(d2)
to=abs(mean(d1)-mean(d2))
# print(to)
# l = list(permutations(d))
# l=random.sample(l, 1000)
c=0
for j in range(1000):
    l=np.random.permutation(d)
    # print(l)
    d1=l[:n]
    d2=l[n:]
    ti=abs(mean(d1)-mean(d2))
    # print(ti)
    if(ti>to):
        c+=1
# print(c)
print("p-value: {}".format(c/1000))
if c/1000>0.05:
    print("Accept Null Hypothesis")
else:
    print("Reject Null Hypothesis")
```

```
p-value: 0.0
Reject Null Hypothesis
```

Since p-value obtained from the permutation test is less than the threshhold we reject the hypothesis that the samples of total deaths of Idaho and Delaware have the same distribution

Q2C Bayesian Inference

In [232]:

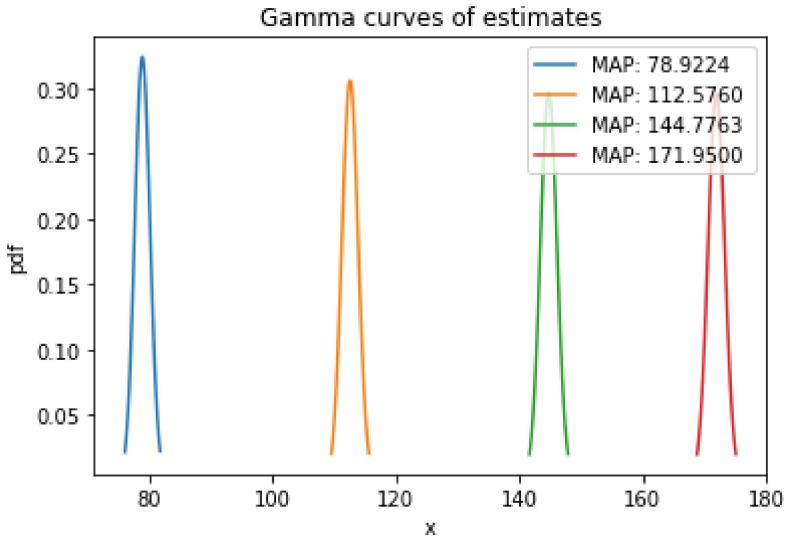
```
june_condition_de = (cases_df_id_de_daily_de_final['submission_date'] >= '2020-06-01') & (c
june_condition_id = (cases_df_id_de_daily_id_final['submission_date'] >= '2020-06-01') & (c
june_de_data = cases_df_id_de_daily_de_final.loc[june_condition_de]
june_id_data = cases_df_id_de_daily_id_final.loc[june_condition_id]
```

In [233]:

```
rem_8weeks_de_cond = (cases_df_id_de_daily_de_final['submission_date'] >= '2020-06-29') & (
rem_8weeks_id_cond = (cases_df_id_de_daily_id_final['submission_date'] >= '2020-06-29') & (
rem_8weeks_de = cases_df_id_de_daily_de_final.loc[rem_8weeks_de_cond]
rem_8weeks_id = cases_df_id_de_daily_id_final.loc[rem_8weeks_id_cond]
```

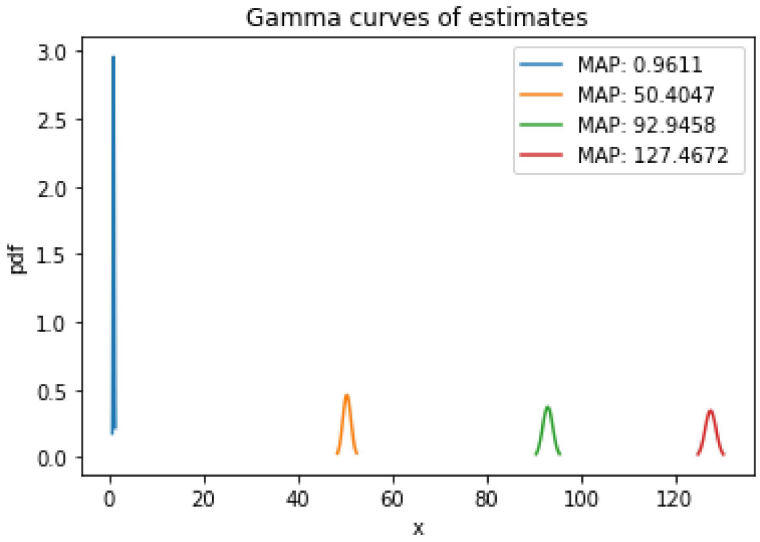
In [234]:

```
lambdaa=mean(list(june_de_data.tot_cases)+list(june_id_data.tot_cases))
beta=1/mean(list(june_de_data.tot_cases)+list(june_id_data.tot_cases))
likelihood_e=len(list(june_de_data.tot_cases)+list(june_id_data.tot_cases))
likelihood_l=sum(list(june_de_data.tot_cases)+list(june_id_data.tot_cases))
prior_e=beta
prior_l=0
post_vals=[]
for i in range(4):
    prior_l=likelihood_l+prior_l
    prior_e=likelihood_e+prior_e
    post_vals.append([prior_l+1,prior_e])
    next_week=list(rem_8weeks_de.tot_cases)[i*7:7+i*7]+list(rem_8weeks_id.tot_cases[i*7:7+i
    likelihood_e=len(next_week)
    likelihood_l=sum(next_week)
    x = np.linspace(gamma.ppf(0.01, prior_l, scale=1/prior_e),gamma.ppf(0.99, prior_l,scale
    MAP = x[np.argmax(gamma.pdf(x, prior_l+1,scale=1/prior_e))]
    plt.plot(x, gamma.pdf(x, prior_l+1,scale=1/prior_e), label = 'MAP: %.4f ' %(MAP))
plt.xlabel('x')
plt.ylabel('pdf')
plt.legend(loc="upper right")
plt.title('Gamma curves of estimates')
plt.show()
# print(post_vals)
```



In [235]:

```
lambdaa=mean(list(june_de_data.tot_death)+list(june_id_data.tot_death))
beta=1/mean(list(june_de_data.tot_death)+list(june_id_data.tot_death))
likelihood_e=len(list(june_de_data.tot_death)+list(june_id_data.tot_death))
likelihood_l=sum(list(june_de_data.tot_death)+list(june_id_data.tot_death))
prior_e=beta
prior_l=0
post_vals=[]
for i in range(4):
    prior_l=likelihood_l+prior_l
    prior_e=likelihood_e+prior_e
    post_vals.append([prior_l+1,prior_e])
    next_week=list(rem_8weeks_de.tot_cases)[i*7:7+i*7]+list(rem_8weeks_id.tot_cases[i*7:7+i
    likelihood_e=len(next_week)
    likelihood_l=sum(next_week)
    x = np.linspace(gamma.ppf(0.01, prior_l, scale=1/prior_e),gamma.ppf(0.99, prior_l,scale
    MAP = x[np.argmax(gamma.pdf(x, prior_l+1,scale=1/prior_e))]
    plt.plot(x, gamma.pdf(x, prior_l+1,scale=1/prior_e), label = 'MAP: %.4f ' %(MAP))
plt.xlabel('x')
plt.ylabel('pdf')
plt.legend(loc="upper right")
plt.title('Gamma curves of estimates')
plt.show()
# print(post_vals)
```



Q2D

Auto Regression

In [236]:

```
may_condition_de = (vaccine_daily_de['Date'] >= '2021-05-01') & (vaccine_daily_de['Date'] <
may_condition_id = (vaccine_daily_id['Date'] >= '2021-05-01') & (vaccine_daily_id['Date'] <
may_de_data_3week = list(vaccine_daily_de.loc[may_condition_de].Administered_new)
may_id_data_3week = list(vaccine_daily_id.loc[may_condition_id].Administered_new)


may_condition_de = (vaccine_daily_de['Date'] >= '2021-05-22') & (vaccine_daily_de['Date'] <
may_condition_id = (vaccine_daily_id['Date'] >= '2021-05-22') & (vaccine_daily_id['Date'] <
may_de_data_fweek = list(vaccine_daily_de.loc[may_condition_de].Administered_new)
may_id_data_fweek = list(vaccine_daily_id.loc[may_condition_id].Administered_new)
```

In [237]:

```
def Mape(pred_list,data_fweek):
    mape=0
    for i in range(len(data_fweek)):
        mape+=abs(data_fweek[i]-pred_list[i])*100/data_fweek[i]
        # print(mape)
    return mape/len(pred_list)
```

In [238]:

```
def SSE(pred_list,data_fweek):
    sse=0
    for i in range(len(data_fweek)):
        sse+=(data_fweek[i]-pred_list[i])**2
    return sse
```

In [239]:

```
def AR(X,data,n,data_fweek):
    Y = data[n:]
    beta = np.matmul(np.matmul(np.linalg.inv(np.matmul(np.transpose(X), X)), np.transpose(X
    X=np.array(data[21-n:])
    pred_list=[]
    for i in range(7):
        X_new=np.concatenate((np.array([1]),np.array(X)))
        # print(X_new)
        pred=np.matmul(X_new,np.transpose(beta))
        pred_list.append(pred)
        X=X[1:]
        X=np.append(X,np.array([pred]))
    mape=Mape(pred_list,data_fweek)
    sse=SSE(pred_list,data_fweek)
    print("Predicted values {}".format(pred_list))
    print("MAPE: {}".format(mape))
    print("SSE: {}".format(sse))
```

Delaware

In [240]:

```
X = np.zeros((len(may_de_data_3week) - 3, 4))
for i in range(3, len(may_de_data_3week)):
  X[i-3][0] = 1
  X[i-3][1] = may_de_data_3week[i-1]
  X[i-3][2] = may_de_data_3week[i-2]
  X[i-3][3] = may_de_data_3week[i-3]
# print(X)
AR(X,may_de_data_3week,3,may_de_data_fweek)
```

```
Predicted values [5245.135173350598, 6074.488106027009, 5315.767157849295, 5
668.735335358822, 5468.034418022151, 5602.7832622426195, 5531.740767125458]
MAPE: 39.301140146419925
SSE: 58853067.72178812
```

In [241]:

```
X = np.zeros((len(may_de_data_3week) - 5, 6))
for i in range(3, len(may_de_data_3week)):
  X[i-5][0] = 1
  X[i-5][1] = may_de_data_3week[i-1]
  X[i-5][2] = may_de_data_3week[i-2]
  X[i-5][3] = may_de_data_3week[i-3]
  X[i-5][4] = may_de_data_3week[i-4]
  X[i-5][5] = may_de_data_3week[i-5]
# print(X)
AR(X,may_de_data_3week,5,may_de_data_fweek)
```

```
Predicted values [6857.69661758383, 5785.737156960024, 4854.509485437036, 65
43.923710474235, 6078.440685476724, 5390.12640980446, 5426.535117905697]
MAPE: 41.282178654379955
SSE: 55904938.92663492
```

Idaho

In [242]:

```
X = np.zeros((len(may_id_data_3week) - 3, 4))
for i in range(3, len(may_id_data_3week)):
  X[i-3][0] = 1
  X[i-3][1] = may_id_data_3week[i-1]
  X[i-3][2] = may_id_data_3week[i-2]
  X[i-3][3] = may_id_data_3week[i-3]
# print(X)
AR(X,may_id_data_3week,3,may_id_data_fweek)
```

```
Predicted values [5731.793677557751, 5685.969911465983, 5469.399906431885, 5
341.457518041079, 5466.614318139787, 5546.920649589637, 5509.07078980633]
MAPE: 69.39807336281974
SSE: 31662624.053393118
```

In [243]:

```
X = np.zeros((len(may_id_data_3week) - 5, 6))
for i in range(3, len(may_id_data_3week)):
  X[i-5][0] = 1
  X[i-5][1] = may_id_data_3week[i-1]
  X[i-5][2] = may_id_data_3week[i-2]
  X[i-5][3] = may_id_data_3week[i-3]
  X[i-5][4] = may_id_data_3week[i-4]
  X[i-5][5] = may_id_data_3week[i-5]
# print(X)
AR(X,may_id_data_3week,5,may_id_data_fweek)
```

```
Predicted values [7932.73158180788, 7243.634648882059, 5681.293730050635, 53
65.94944179637, 4378.324687104982, 3954.838264066976, 4876.170986369067]
MAPE: 58.47159547995818
SSE: 25178959.06810081
```

EWMA

In [244]:

```
def EWMA(data,alpha,data_fweek):
    data_c=copy.deepcopy(data)
    data_c+=data_fweek
    pred_list=[]
    y_t_hat=data_c[0]
    for j in range(28):
        # print(data_c)
        # ynew=0
        # for i in range(0,len(data_c)):
        #     ynew+=data_c[len(data_c)-(i+1)](1-alpha)*(i)
        # ynew*=alpha
        y_t=data_c[j]
        y_t_hat = alpha*y_t + (1-alpha)*y_t_hat
        pred_list.append(y_t_hat)
        # print(len(pred_list))
        # data_c.append(y_t_hat)
    mape=Mape(pred_list[21:],data_fweek)
    sse=SSE(pred_list[21:],data_fweek)
    # print(data_fweek)
    print("For alpha values: {}".format(alpha))
    print("Predicted values: {}".format(pred_list[21:]))
    print("MAPE: {}".format(mape))
    print("SSE: {}".format(sse))
```

Delaware

In [245]:

```
EWMA(may_de_data_3week,0.8,may_de_data_fweek)
print()
EWMA(may_de_data_3week,0.5,may_de_data_fweek)
```

```
For alpha values: 0.8
Predicted values: [8849.40445625943, 8066.680891251885, 4790.9361782503765,
3829.387235650075, 3547.477447130015, 4787.895489426003, 9539.9790978852]
MAPE: 8.00423366131092
SSE: 2728957.4234906263

For alpha values: 0.5
Predicted values: [7284.087044715881, 7577.543522357941, 5774.77176117897, 4
681.885880589485, 4079.4429402947426, 4588.721470147371, 7658.360735073686]
MAPE: 22.711774652755118
SSE: 19574796.35283082
```

Idaho

In [246]:

```
EWMA(may_id_data_3week,0.8,may_id_data_fweek)
print()
EWMA(may_id_data_3week,0.5,may_id_data_fweek)
```

```
For alpha values: 0.8
Predicted values: [6931.14917297051, 6166.229834594102, 3577.24596691882, 19
45.0491933837638, 3225.809838676753, 3805.1619677353506, 4166.63239354707]
MAPE: 10.463815894161465
SSE: 992641.5659346564

For alpha values: 0.5
Predicted values: [6110.145548820496, 6042.572774410248, 4486.286387205124,
3011.643193602562, 3278.821596801281, 3614.4107984006405, 3935.705399200320
2]
MAPE: 27.345784680299452
SSE: 6604119.262423538
```

In [62]:

```
import pandas as pd
import numpy as np
import collections
import csv
import math
import warnings
import scipy.stats

warnings.filterwarnings('ignore')
```

In [ ]:

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

Q2E

In [64]:

```
vaccine_del=pd.read_csv("/content/gdrive/MyDrive/delaware_vaccines.csv")
vaccine_id=pd.read_csv("/content/gdrive/MyDrive/idaho_vaccines.csv")
```

In [65]:

```
# Function for paired two sample T test
def ttest_twot_e(a,b):
    print("Mean Administered daily for Delaware {}".format(np.mean(a)))
    print("Mean Administered daily for Idaho {}".format(np.mean(b)))
    D = a-b
    # print(len(D))

    Dbar = np.mean(D)

    # print('dbar',Dbar)

    sd = np.sqrt(np.sum(np.square(D-Dbar))/(len(D)))

    # print('sd',sd)

    denom = sd/np.sqrt(len(D))

    # print('denom',denom)

    paired_t_stat = Dbar/denom

    print("t statistic: {}".format(paired_t_stat))

    t_tab_val = 2.000995

    if abs(paired_t_stat)>t_tab_val:
        print('Null hypothesis is rejected')
    else:
        print('Null hypothesis is accepted')
```

T-test for September plus November

In [66]:

```
# Vaccine dataframes for Delaware, Idaho
vaccine_del["Date"] = pd.to_datetime(vaccine_del["Date"])
vaccine_id["Date"] = pd.to_datetime(vaccine_id["Date"])
vaccine_del_sn = list(vaccine_del["Administered_new"][(vaccine_del["Date"].dt.month == 9) &
vaccine_id_sn = list(vaccine_id["Administered_new"][(vaccine_id["Date"].dt.month == 9) & (v
ttest_twot_e(np.array(vaccine_del_sn),np.array(vaccine_id_sn))
```

```
Mean Administered daily for Delaware 3441.6833333333334
Mean Administered daily for Idaho 4386.183333333333
t statistic: -0.6768374111408556
Null hypothesis is accepted
```

T-test for September

In [67]:

```
# Vaccine dataframes for Delaware, Idaho
vaccine_del["Date"] = pd.to_datetime(vaccine_del["Date"])
vaccine_id["Date"] = pd.to_datetime(vaccine_id["Date"])
vaccine_del_s = list(vaccine_del["Administered_new"][(vaccine_del["Date"].dt.month == 9) &
vaccine_id_s = list(vaccine_id["Administered_new"][(vaccine_id["Date"].dt.month == 9) & (va
ttest_twot_e(np.array(vaccine_del_s),np.array(vaccine_id_s))
```

```
Mean Administered daily for Delaware 1680.1666666666667
Mean Administered daily for Idaho 3540.8
t statistic: -4.4015096671754055
Null hypothesis is rejected
```

Since the means of daily Administered for Delaware and Idaho are very different we can see why the null hypothesis was rejected. The rejection hypothesis seems accurate the we are trying to compare the values of means of both the states and it seems from the results above that the means are quite contrastingly different.

T-test for November

In [68]:

```
# Vaccine dataframes for Delaware, Idaho
vaccine_del["Date"] = pd.to_datetime(vaccine_del["Date"])
vaccine_id["Date"] = pd.to_datetime(vaccine_id["Date"])
vaccine_del_n = list(vaccine_del["Administered_new"][(vaccine_del["Date"].dt.month == 11) &
vaccine_id_n = list(vaccine_id["Administered_new"][(vaccine_id["Date"].dt.month == 11) & (v
ttest_twot_e(np.array(vaccine_del_n),np.array(vaccine_id_n))
```

```
Mean Administered daily for Delaware 5203.2
Mean Administered daily for Idaho 5231.566666666667
t statistic: -0.01032054938492115
Null hypothesis is accepted
```

Since the means of daily Administered for Delaware and Idaho are very close we can see why the null hypothesis was accepted. The acceptance hypothesis is accurate in this case as the means of the vaccines administered in both the states seem to be close or comparable so it obviously makes sense that the hypothesis of mean comparision is that the means are equvalent (approximately).

In [ ]:

In [68]:

Inference 1 on Exploratory Data

Dataset details:

Link: https://aqs.epa.gov/aqsweb/airdata/download_files.html (https://aqs.epa.gov/aqsweb/airdata/download_files.html)

We have used a combination 2 types of Air Quality datasets for exploratory inferences:

1. County and State-wise AQI dataset for the years 2020 and 2021

   https://aqs.epa.gov/aqsweb/airdata/daily_aqi_by_county_2021.zip (https://aqs.epa.gov/aqsweb/airdata/daily_aqi_by_county_2021.zip)

   https://aqs.epa.gov/aqsweb/airdata/daily_aqi_by_county_2020.zip (https://aqs.epa.gov/aqsweb/airdata/daily_aqi_by_county_2020.zip)
2. County and State-wise AQI specific for Carbon Monoxide (CO) indexes for the years 2020 and 2021

   https://aqs.epa.gov/aqsweb/airdata/daily_42101_2021.zip (https://aqs.epa.gov/aqsweb/airdata/daily_42101_2021.zip)

   https://aqs.epa.gov/aqsweb/airdata/daily_42101_2020.zip (https://aqs.epa.gov/aqsweb/airdata/daily_42101_2020.zip)

Motivation

Air Quality Index (AQI) is an accurate measure for identifying the pollutants present in the air. During the time of Covid, lockdowns were imposed all around the US, which is bound to impact the overall AQI of the country. We wish to explore the relation between this measure and the cases/vaccines data provided in the question document. In addition, some particular pollutants (like Carbon Monoxide, etc) can also be changed in some way during and after the months of Covid, we wish to understand the overall trend of that.

In [47]:
```
import pandas as pd
import numpy as np
import math
```

In [48]:
```
vaccines = pd.read_csv("/Users/meet/Desktop/544_project/Mandatory_data
vaccines.head()
```

Out[48]:

	Date	MMWR_week	Location	Distributed	Distributed_Janssen	Distributed_Moderna
0	05/15/2022	20	PR	7552350	215000	2662120
1	05/15/2022	20	KS	6121515	256400	2354940
2	05/15/2022	20	VA	19949085	785300	7108700
3	05/15/2022	20	MT	2004895	105200	825200
4	05/15/2022	20	IH2	2965895	108400	1311680

5 rows × 82 columns

In [49]:
```
# filtering the vaccines data to get the location, date and distribute
vaccines_filtered = vaccines[["Date", "Location", "Distributed", "Dist

# sorting the data in ascending order by date
vaccines_filtered["Date"] = pd.to_datetime(vaccines_filtered["Date"])
vaccines_filtered.sort_values(by="Date", inplace=True)
vaccines_filtered.head()
```

Out[49]:

	Date	Location	Distributed	Distributed_Janssen	Distributed_Moderna	Distributed_Pfize
33431	2020-12-13	GU	3900	0	0	
33426	2020-12-13	LTC	0	0	0	
33427	2020-12-13	AS	3900	0	0	
33430	2020-12-13	US	13650	0	0	
33429	2020-12-13	VI	975	0	0	

In [50]:
```
vaccines_filtered.isnull().sum()
```

Out[50]:
```
Date                    0
Location                0
Distributed             0
Distributed_Janssen     0
Distributed_Moderna     0
Distributed_Pfizer      0
Administered            0
Administered_Janssen    0
Administered_Moderna    0
Administered_Pfizer     0
dtype: int64
```

Check the Dependency of Location (State) on AQI

Identify if locations where vaccines are manufactured and distributed more have a higher AQI or not

Motivation

A lot of pharma companies initiated their research on finding a vaccine for Covid and started mass manufacturing of the same. Hence, ideally the AQI should have gone low (got better) during the months of Covid due to lockdowns, but we would like to make a hypothesis that the states where vaccine were being manufactured and getting distributed the most, should still have bad AQI (worse than expected) due to the pollution from drug manufacturing factories, transport vehicles, and other factors included in vaccine administration.

In [292]:
```
# helper function to gerenrate estimate CDF
def generate_eCDF(X):
    n = len(X)
    Srt = sorted(X)
    delta = .1
    X = [min(Srt)-delta]
    Y = [0]
    for i in range(0, n):
        X = X + [Srt[i], Srt[i]]
        Y = Y + [Y[len(Y)-1], Y[len(Y)-1]+(1/n)]
    X = X + [max(Srt)+delta]
    # print(X)
    Y = Y + [1]
    return X, Y
```

In [293]:
```
# helper function to perform a 2 sample KS-Test
def ks_test_2_sample(X1, Y1, X2, Y2):
    tot_max = -1
    ks_table = np.zeros((len(X1),6))
    for i in range(len(ks_table)-1):
        ks_table[i,0] = Y1[i]
        ks_table[i,1] = Y1[i+1]
        ks_table[i,2]=0
        ks_table[i,3]=0
        for j in X2:
            if j<X1[i]:
                ks_table[i,2]+=1
            if j<=X1[i]:
                ks_table[i,3]+=1

        ks_table[i,3]/=len(X2)
        ks_table[i,2]/=len(X2)

        ks_table[i,4] = abs(ks_table[i,0] - ks_table[i,2])
        ks_table[i,5] = abs(ks_table[i,1] - ks_table[i,3])
        cmax = max(ks_table[i,4], ks_table[i,5])
        if cmax > tot_max:
            tot_max = cmax
            x1_max = X1[i]
            y1_max = ks_table[i,0]
            y2_max = ks_table[i,2]
    return tot_max
```

In [302]:
```
# helper function to perform Pearson Correlation
def pearson_corr(df):
    num = np.sum((df['Distributed'] - df['Distributed'].mean()) * (df[
    den = np.sqrt(np.sum(pow(df['Distributed'] - df['Distributed'].mea
    coeff = num/den

    return coeff
```

In [303]:
```
)upby(["Location"]).agg({"Distributed_Pfizer": "max"}).sort_values(by="
```

Out[303]:

Location	Distributed_Pfizer
US	432073335
LTC	73720035
CA	54867275
TX	37383255
FL	28281365
NY	28267955
PA	17641365
IL	16964305
OH	13683215
NJ	13345775

In [304]:
```
# Function to get the daily distributed vaccine data for particular st
def get_state_wise_daily(df, state):
    df_state = df.loc[df["Location"].str.startswith(state, na=False)].
    df_state.reset_index(drop=True, inplace=True)
    df_state_updated = pd.DataFrame(columns=["Date", "Location", "Dist
    df_state_updated = df_state_updated.append({"Date": df_state.Date[
    for i in range(1, len(df_state)):
        state_date = df_state.Date[i]
        state = df_state.Location[i]
        state_dis = df_state.Distributed[i] - df_state.Distributed[i -
        state_dis_jj = df_state.Distributed_Janssen[i] - df_state.Dist
        state_dis_md = df_state.Distributed_Moderna[i] - df_state.Dist
        state_dis_pf = df_state.Distributed_Pfizer[i] - df_state.Distr
        df_state_updated = df_state_updated.append({"Date": state_date

    return df_state_updated
```

In [155]:
```
# load aqi 2020 and 2021 dataset
aqi_2020 = pd.read_csv("/Users/meet/Desktop/544_project/X_dataset/dail
aqi_2021 = pd.read_csv("/Users/meet/Desktop/544_project/X_dataset/dail
```

In [156]:
```
# combining the AQI 2020 and 2021 dataset
aqi = pd.concat([aqi_2020, aqi_2021]).reset_index(drop=True)
print(len(aqi_2020) + len(aqi_2021), len(aqi))
```

```
556391 556391
```

In [157]:
```
aqi.head()
```

Out[157]:

	State Name	county Name	State Code	County Code	Date	AQI	Category	Defining Parameter	Defining Site	Number of Sites Reporting
0	Alabama	Baldwin	1	3	2020-01-01	48	Good	PM2.5	01-003-0010	1
1	Alabama	Baldwin	1	3	2020-01-04	13	Good	PM2.5	01-003-0010	1
2	Alabama	Baldwin	1	3	2020-01-07	14	Good	PM2.5	01-003-0010	1
3	Alabama	Baldwin	1	3	2020-01-10	39	Good	PM2.5	01-003-0010	1
4	Alabama	Baldwin	1	3	2020-01-13	29	Good	PM2.5	01-003-0010	1

In [158]:
```
aqi.rename(columns={"State Name": "state", "county Name": "county", "D
```

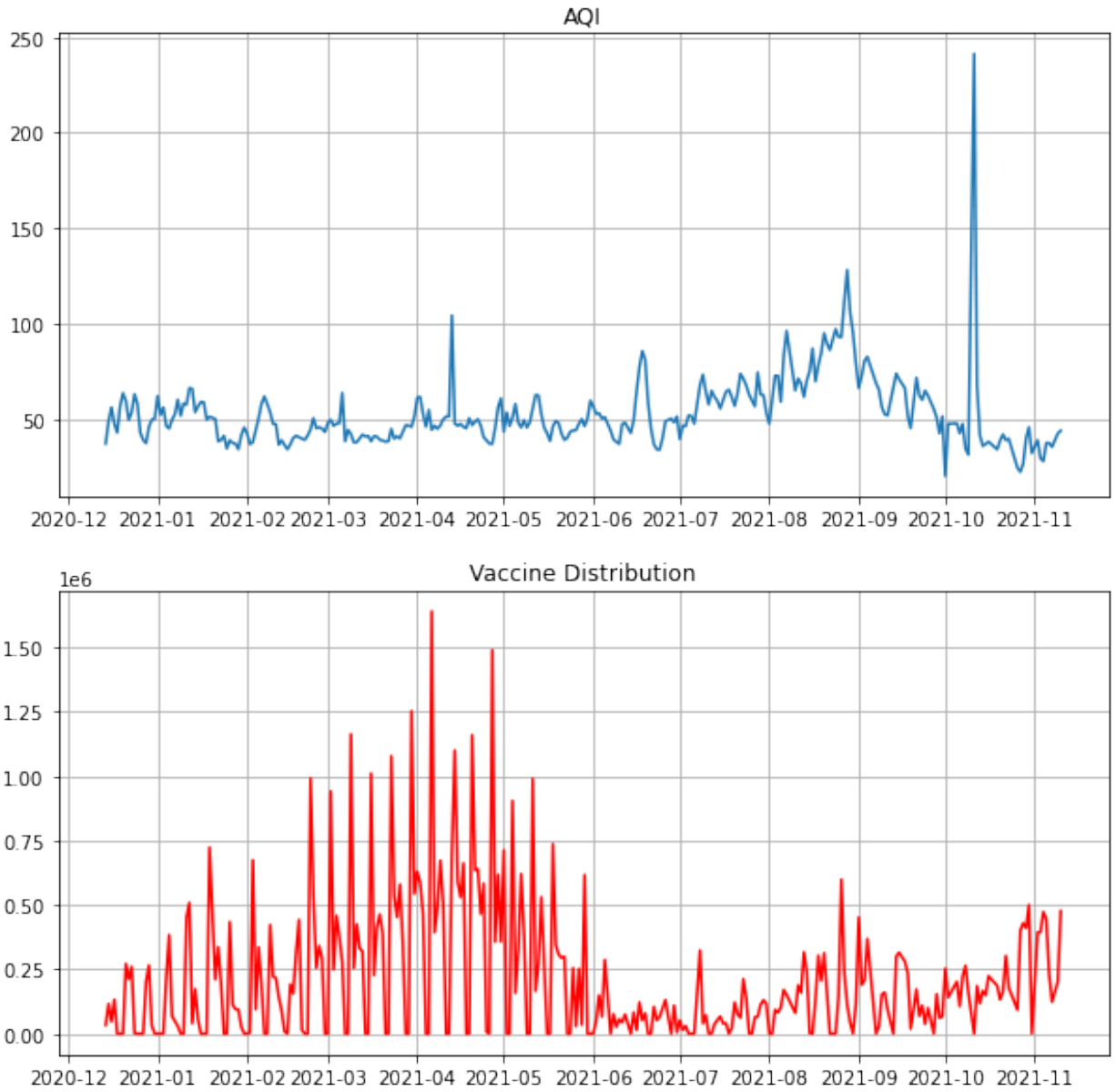
California

California has the highest number of vaccines distributed, so we tried to perform our inference on the state-specific data for this. **Null Hypothesis H0:** The AQI should be dependent on the vaccines distributed **Alternate Hypothesis H1:** The AQI should not be dependent on the vaccines distributed

In [296]:
```
vaccines_ca = get_state_wise_daily(vaccines_filtered, "CA")
vaccines_ca = vaccines_ca[vaccines_ca["Distributed"] >= 0]
vaccines_ca.head()

aqi_ca = aqi[["state", "Date", "AQI", "Category", "defining_param"]][a

aqi_ca["Date"] = pd.to_datetime(aqi_ca["Date"])
aqi_ca.sort_values(by="Date", inplace=True)
aqi_ca.reset_index(drop=True, inplace=True)
aqi_ca.head()

aqi_ca_mean = aqi_ca[["state", "Date", "AQI", "Category", "defining_pa
aqi_ca_mean.reset_index(inplace=True)
aqi_ca_mean.head()

vaccines_aqi_ca = pd.merge(vaccines_ca, aqi_ca_mean, on=("Date"))
vaccines_aqi_ca.head()

vaccines_ca_x, vaccines_ca_y = generate_eCDF(vaccines_aqi_ca.Distribut
aqi_ca_x, aqi_ca_y = generate_eCDF(vaccines_aqi_ca.AQI.to_list())
print("KS Statistic", ks_test_2_sample(vaccines_ca_x, vaccines_ca_y, a

print("Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_ca)
```

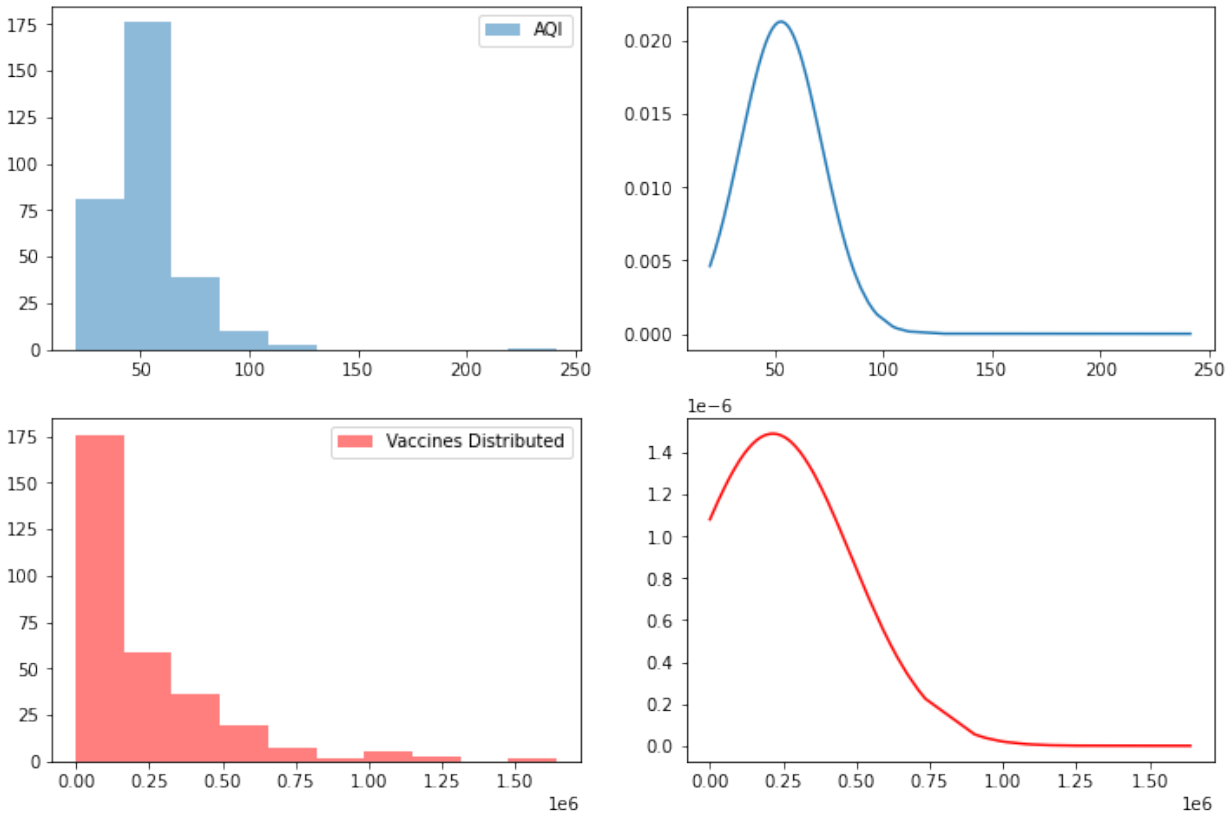
```
KS Statistic 0.7750965654034869
Pearson Correlation Coefficient -0.12324445671111955
```

In [263]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_ca["Date"], vaccines_aqi_ca["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_ca["Date"], vaccines_aqi_ca["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [290]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_ca.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_ca.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[290]: <matplotlib.legend.Legend at 0x7f7e09efbe20>



Results for California

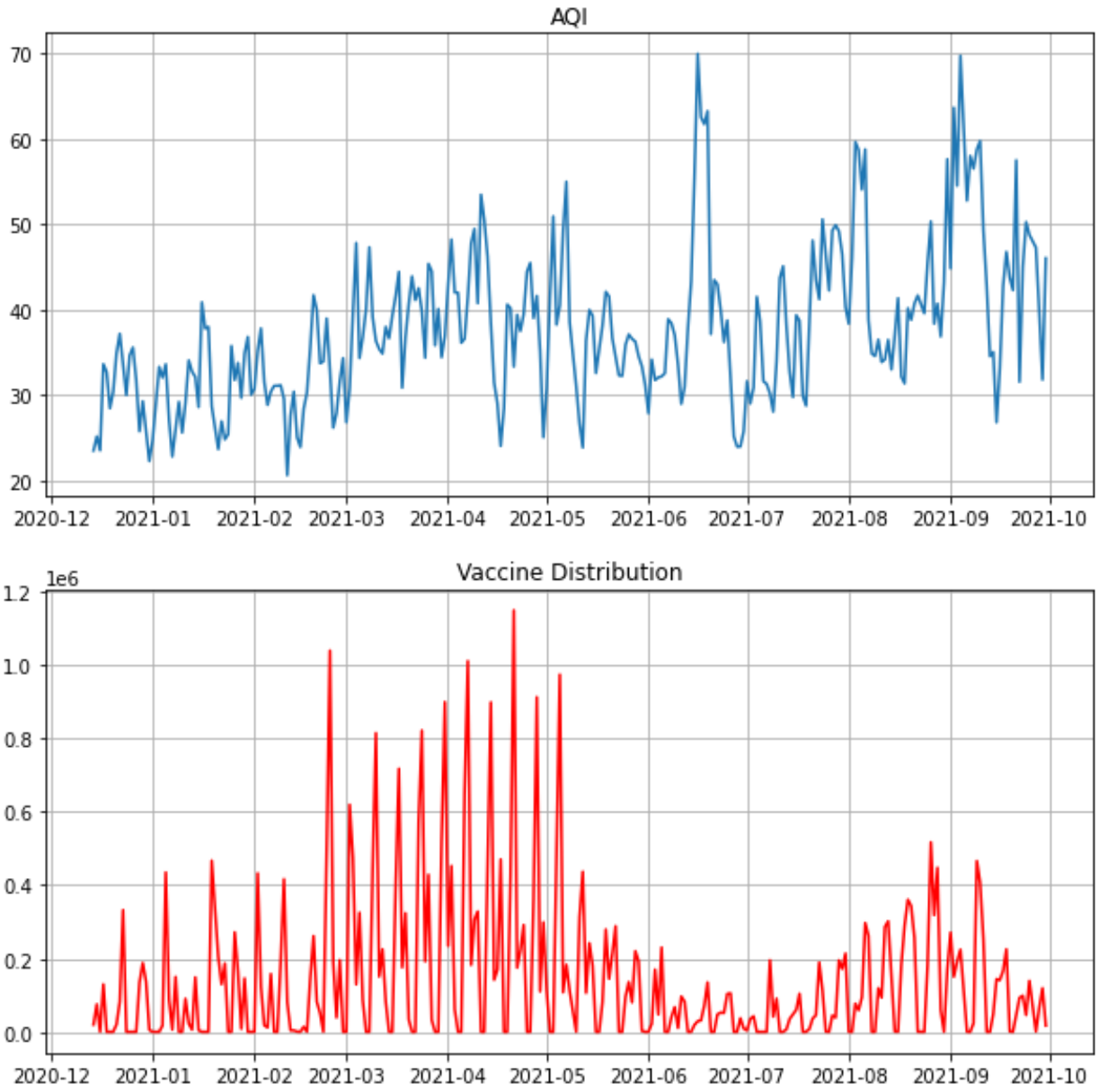
From the Pearson Correlation, KS-Statistic value, and the plots above, it can be inferred that the null hypothesis H0 will be rejected as the correlation coefficient is less than 0.5 and KS-stat is much larger than 0.05. It signifies that the number of vaccines distributed in a state does not provide dependency to the AQI of the state.

Texas

In [297]:
```
s_tx = get_state_wise_daily(vaccines_filtered, "TX")
s_tx = vaccines_tx[vaccines_tx["Distributed"] >= 0]
s_tx.head()

= aqi[["state", "Date", "AQI", "Category", "defining_param"]][aqi["sta

"Date"] = pd.to_datetime(aqi_tx["Date"])
sort_values(by="Date", inplace=True)
reset_index(drop=True, inplace=True)
head()

mean = aqi_tx[["state", "Date", "AQI", "Category", "defining_param"]].
mean.reset_index(inplace=True)
mean.head()

s_aqi_tx = pd.merge(vaccines_tx, aqi_tx_mean, on=("Date"))
s_aqi_tx.head()

s_tx_x, vaccines_tx_y = generate_eCDF(vaccines_aqi_tx.Distributed.to_l
x, aqi_tx_y = generate_eCDF(vaccines_aqi_tx.AQI.to_list())
KS Statistic", ks_test_2_sample(vaccines_tx_x, vaccines_tx_y, aqi_tx_x

Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_tx))
```

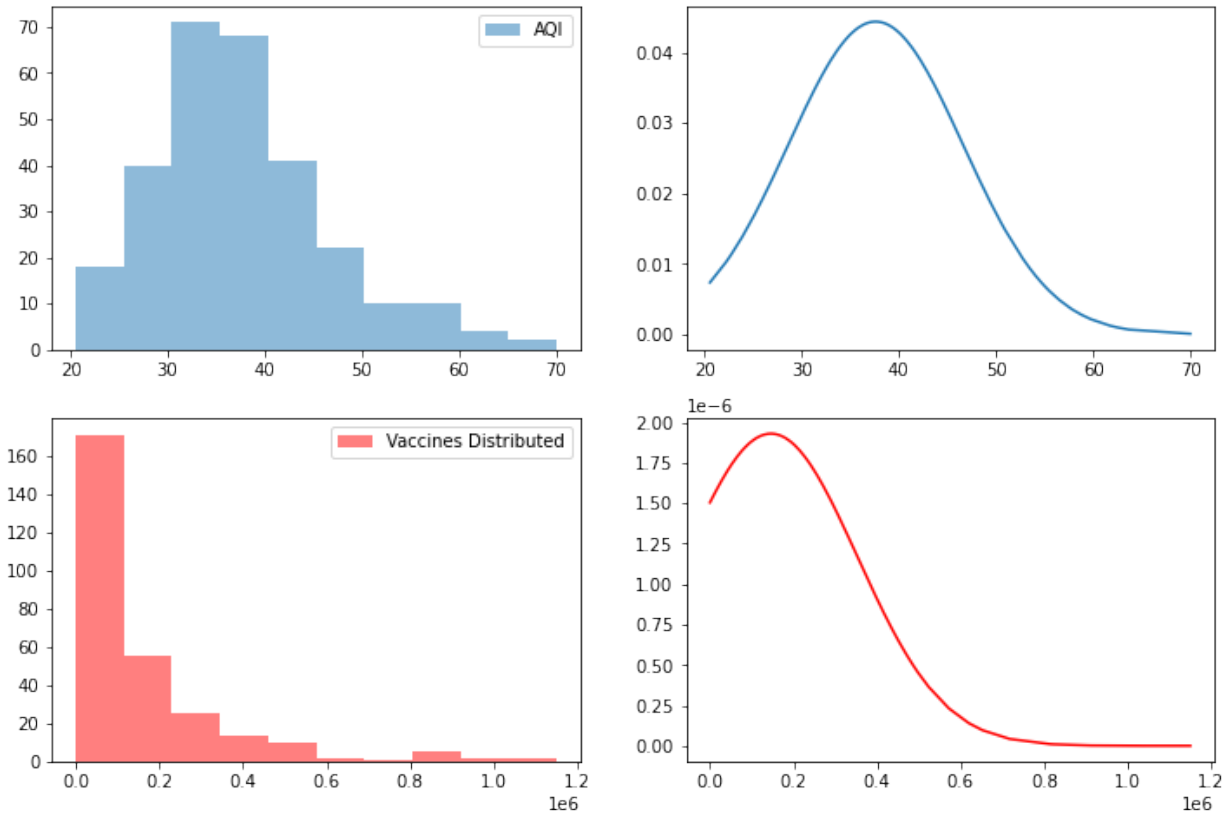
```
KS Statistic 0.7272727272727275
Pearson Correlation Coefficient 0.0733431010113787
```

In [265]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_tx["Date"], vaccines_aqi_tx["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_tx["Date"], vaccines_aqi_tx["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [289]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_tx.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_tx.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[289]: <matplotlib.legend.Legend at 0x7f7da8359ac0>



Results for Texas

The results are again similar to that of California. From the Pearson Correlation, KS-Statistic value, and the plots above, it can be inferred that the null hypothesis H0 will be rejected as the correlation coefficient is less than 0.5 and KS-stat is much larger than 0.05. It signifies that the number of vaccines distributed in a state does not provide dependency to the AQI of the state.

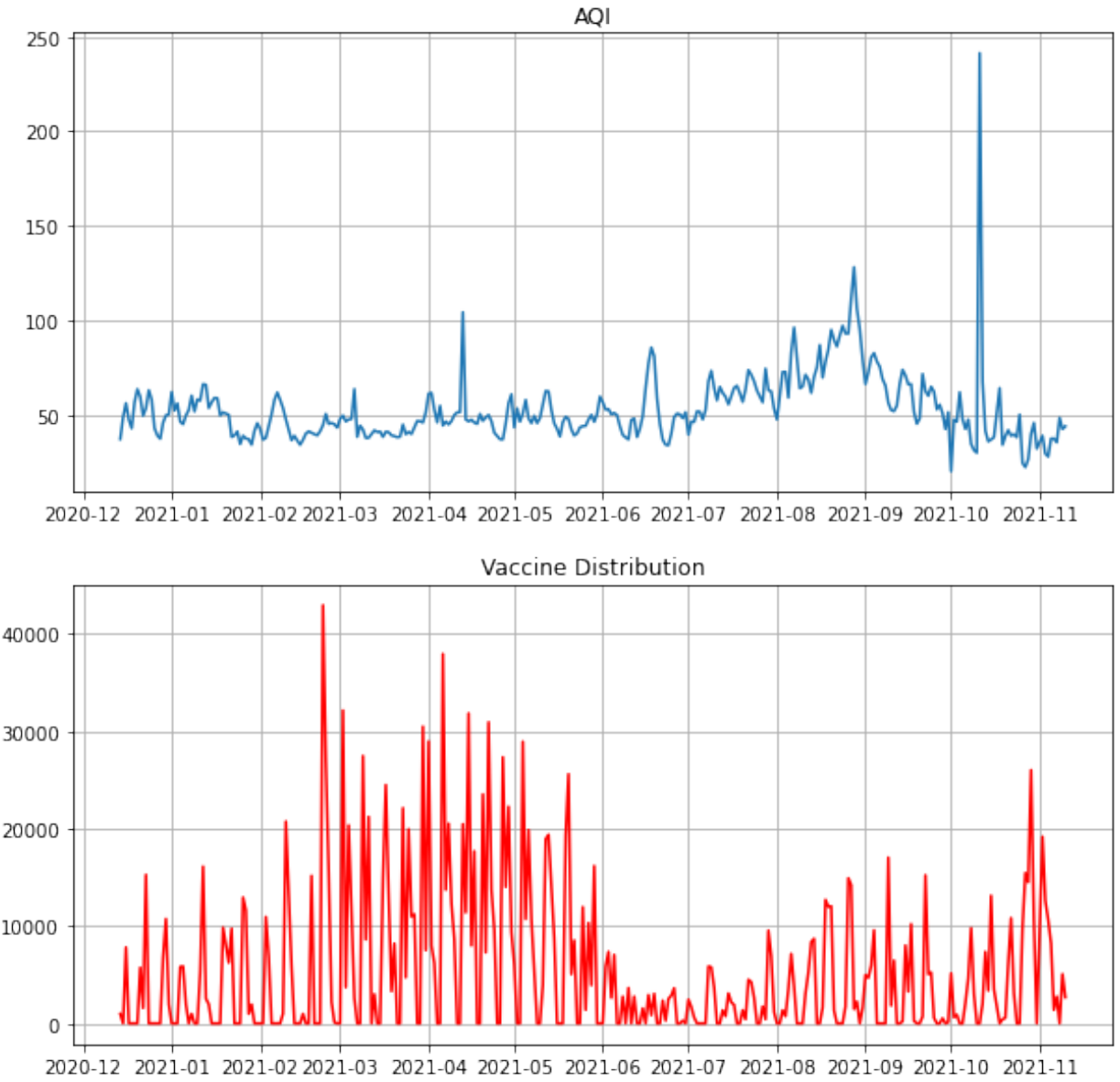
Delaware

Now, here we try to identify if the size of state has any impact on the results or not. For that, we try to perform the same hypothesis on Delaware and Rhode Island which are the 2 smallest states of US based on their size.

In [298]:
```
vaccines_de = get_state_wise_daily(vaccines_filtered, "DE")
vaccines_de = vaccines_de[vaccines_de["Distributed"] >= 0]
vaccines_de.head()

aqi_de = aqi[["state", "Date", "AQI", "Category", "defining_param"]][a

aqi_de["Date"] = pd.to_datetime(aqi_de["Date"])
aqi_de.sort_values(by="Date", inplace=True)
aqi_de.reset_index(drop=True, inplace=True)
aqi_de.head()

aqi_de_mean = aqi_de[["state", "Date", "AQI", "Category", "defining_pa
aqi_de_mean.reset_index(inplace=True)
aqi_de_mean.head()

vaccines_aqi_de = pd.merge(vaccines_de, aqi_de_mean, on=("Date"))
vaccines_aqi_de.head()

vaccines_de_x, vaccines_de_y = generate_eCDF(vaccines_aqi_de.Distribut
aqi_de_x, aqi_de_y = generate_eCDF(vaccines_aqi_de.AQI.to_list())
print("KS Statistic", ks_test_2_sample(vaccines_de_x, vaccines_de_y, a

print("Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_de)
```

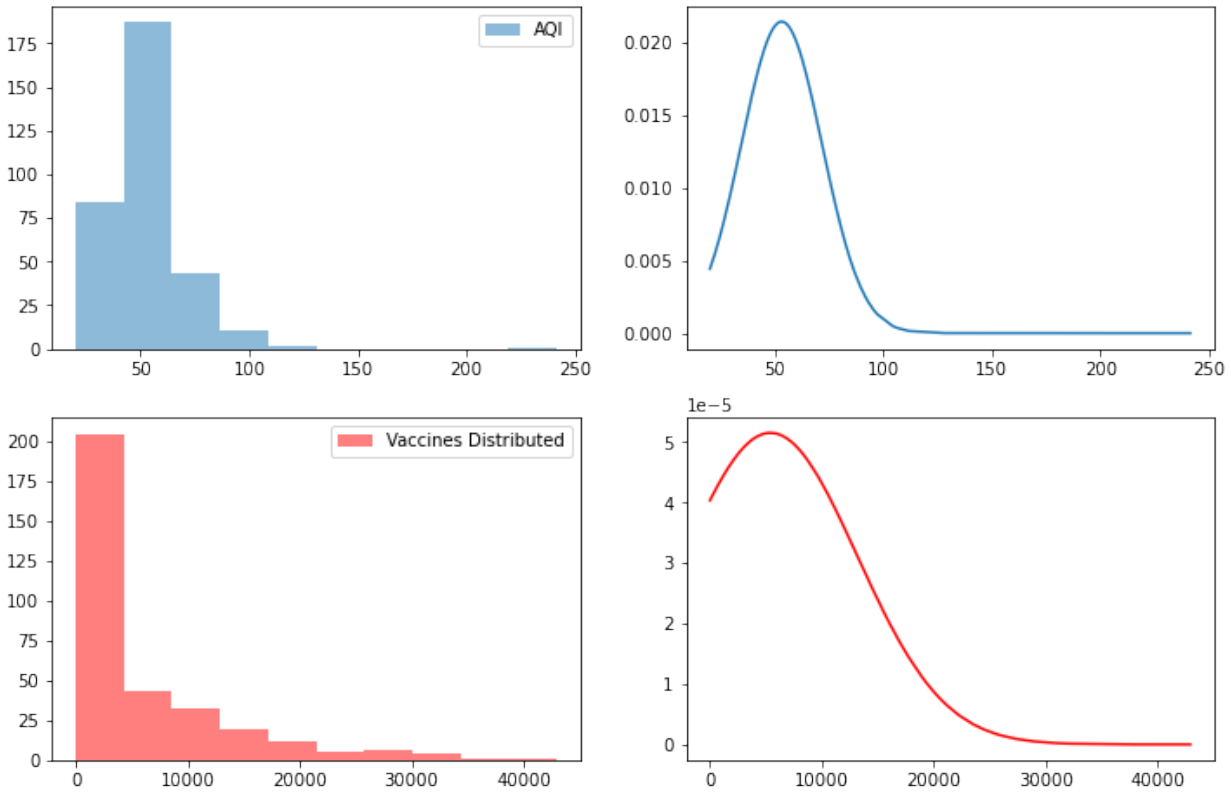
```
KS Statistic 0.6143524353176659
Pearson Correlation Coefficient -0.0903180969589416
```

In [266]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_de["Date"], vaccines_aqi_de["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_de["Date"], vaccines_aqi_de["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [288]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_de.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_de.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[288]: <matplotlib.legend.Legend at 0x7f7e09c64580>

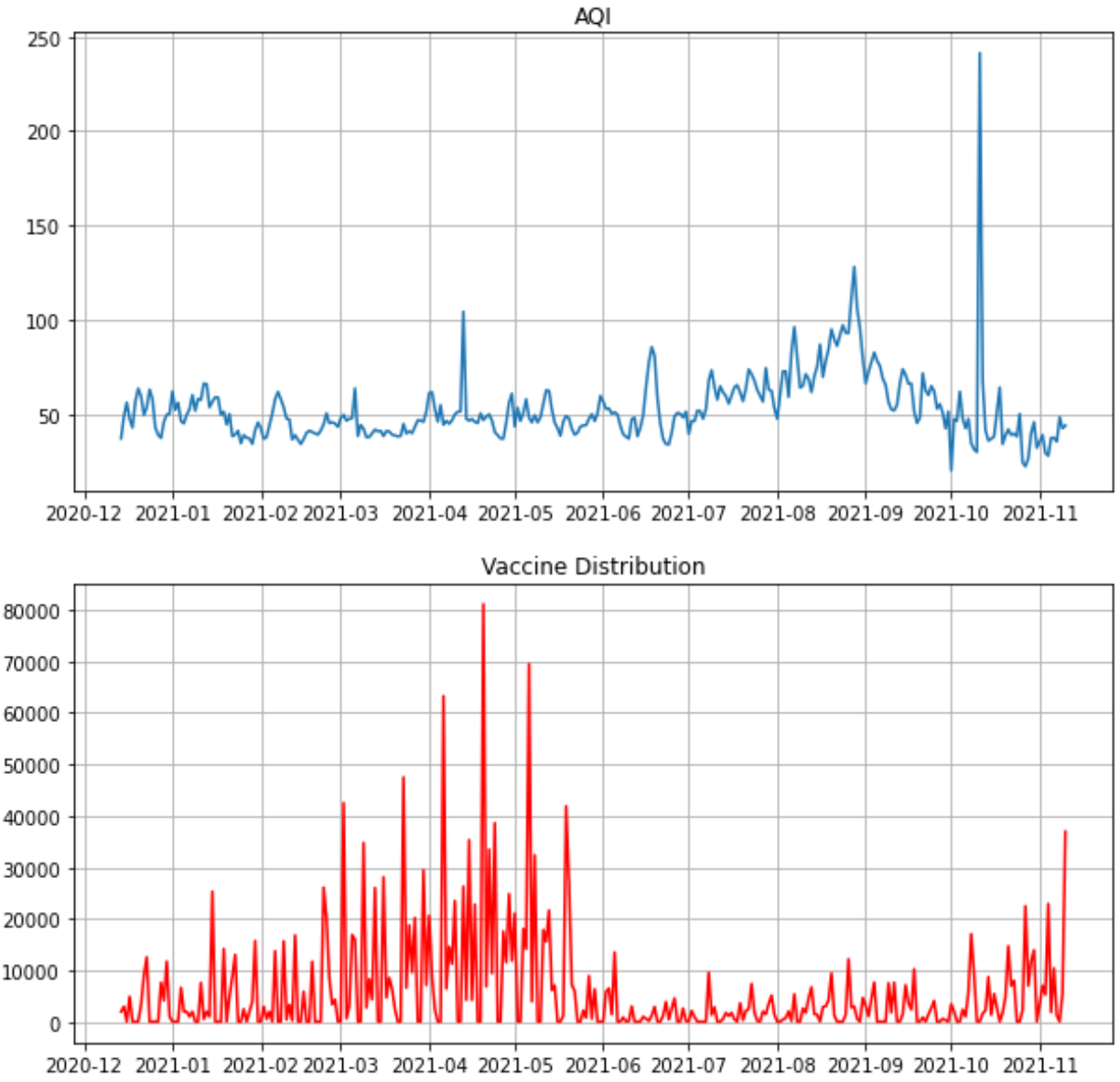


Rhode Island

In [299]:
```
vaccines_ri = get_state_wise_daily(vaccines_filtered, "RI")
vaccines_ri = vaccines_ri[vaccines_ri["Distributed"] >= 0]
vaccines_ri.head()

aqi_ri = aqi[["state", "Date", "AQI", "Category", "defining_param"]][a

aqi_ri["Date"] = pd.to_datetime(aqi_ri["Date"])
aqi_ri.sort_values(by="Date", inplace=True)
aqi_ri.reset_index(drop=True, inplace=True)
aqi_ri.head()

aqi_ri_mean = aqi_ca[["state", "Date", "AQI", "Category", "defining_pa
aqi_ri_mean.reset_index(inplace=True)
aqi_ri_mean.head()

vaccines_aqi_ri = pd.merge(vaccines_ri, aqi_ri_mean, on=("Date"))
vaccines_aqi_ri.head()

vaccines_ri_x, vaccines_ri_y = generate_eCDF(vaccines_aqi_ri.Distribut
aqi_ri_x, aqi_ri_y = generate_eCDF(vaccines_aqi_ri.AQI.to_list())
print("KS Statistic", ks_test_2_sample(vaccines_ri_x, vaccines_ri_y, a

print("Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_ri)
```

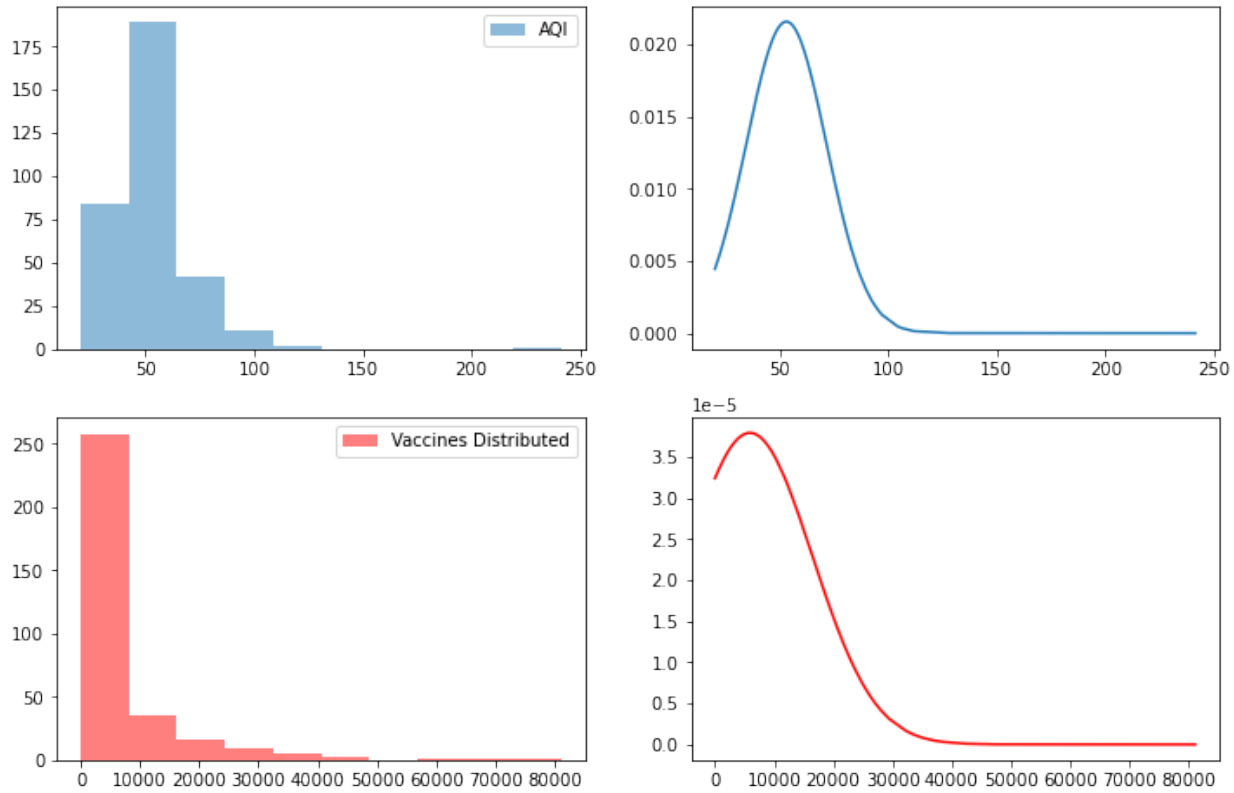
```
KS Statistic 0.6611080408952756
Pearson Correlation Coefficient -0.11396235481327478
```

In [269]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_ri["Date"], vaccines_aqi_ri["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_ri["Date"], vaccines_aqi_ri["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [287]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_ri.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_ri.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[287]: <matplotlib.legend.Legend at 0x7f7e0831b7c0>



Results for Small States

Results above again signify similarity to that of states with most distribution of vaccines.

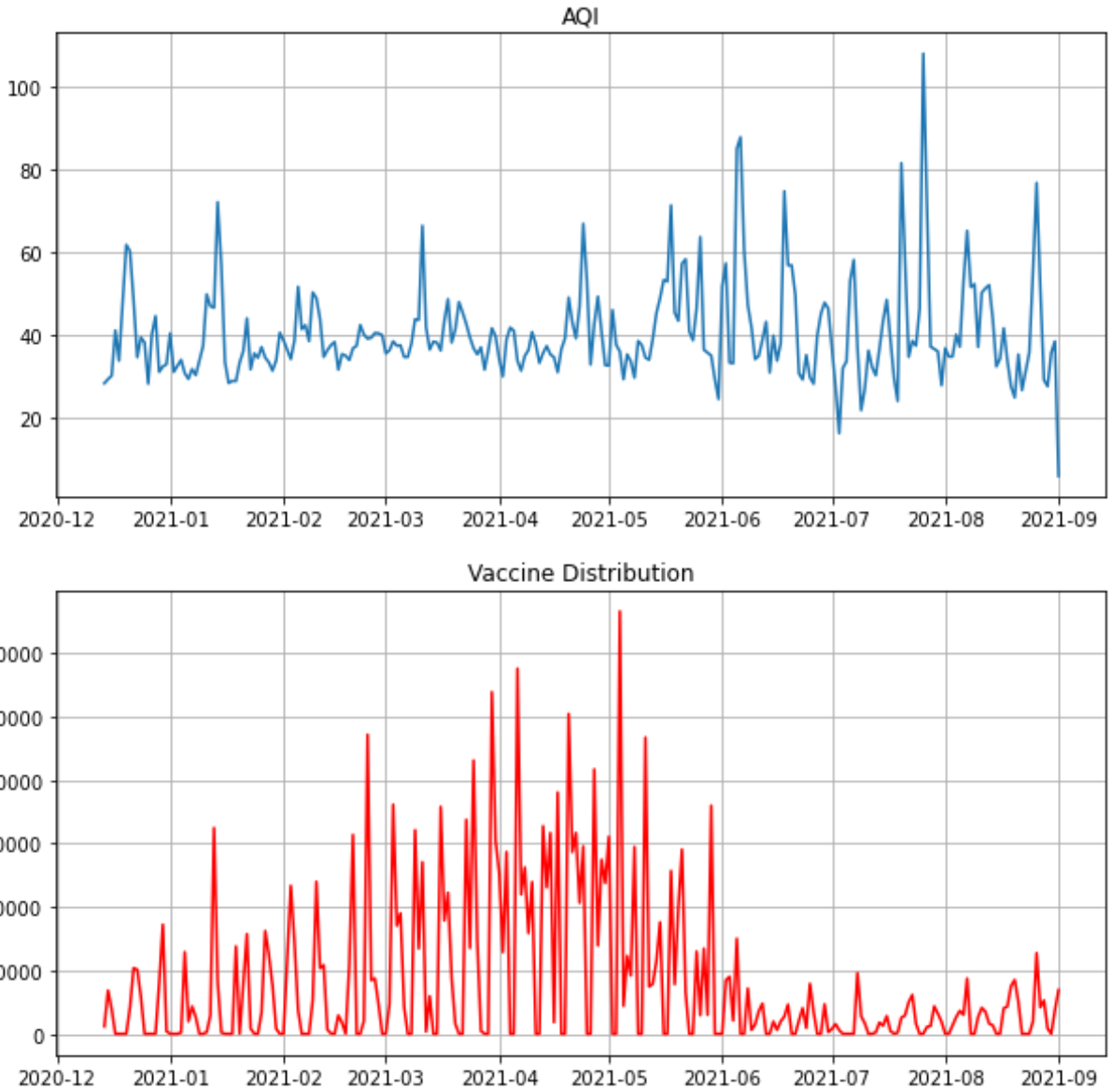
Massechusetts

Now, we try to perform the same hypothesis testing on the states where vaccines like Pfizer and Moderna are manufactured. i.e. Massechussetts and New York, as the factories emmit several pollutants during the process of manufacturing the vaccines.

In [300]:
```
vaccines_ma = get_state_wise_daily(vaccines_filtered, "MA")
vaccines_ma = vaccines_ma[vaccines_ma["Distributed"] >= 0]
vaccines_ma.head()

aqi_ma = aqi[["state", "Date", "AQI", "Category", "defining_param"]][a

aqi_ma["Date"] = pd.to_datetime(aqi_ma["Date"])
aqi_ma.sort_values(by="Date", inplace=True)
aqi_ma.reset_index(drop=True, inplace=True)
aqi_ma.head()

aqi_ma_mean = aqi_ma[["state", "Date", "AQI", "Category", "defining_pa
aqi_ma_mean.reset_index(inplace=True)
aqi_ma_mean.head()

vaccines_aqi_ma = pd.merge(vaccines_ma, aqi_ma_mean, on=("Date"))
vaccines_aqi_ma.head()

vaccines_ma_x, vaccines_ma_y = generate_eCDF(vaccines_aqi_ma.Distribut
aqi_ma_x, aqi_ma_y = generate_eCDF(vaccines_aqi_ma.AQI.to_list())
print("KS Statistic", ks_test_2_sample(vaccines_ma_x, vaccines_ma_y, a

print("Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_ma)
```

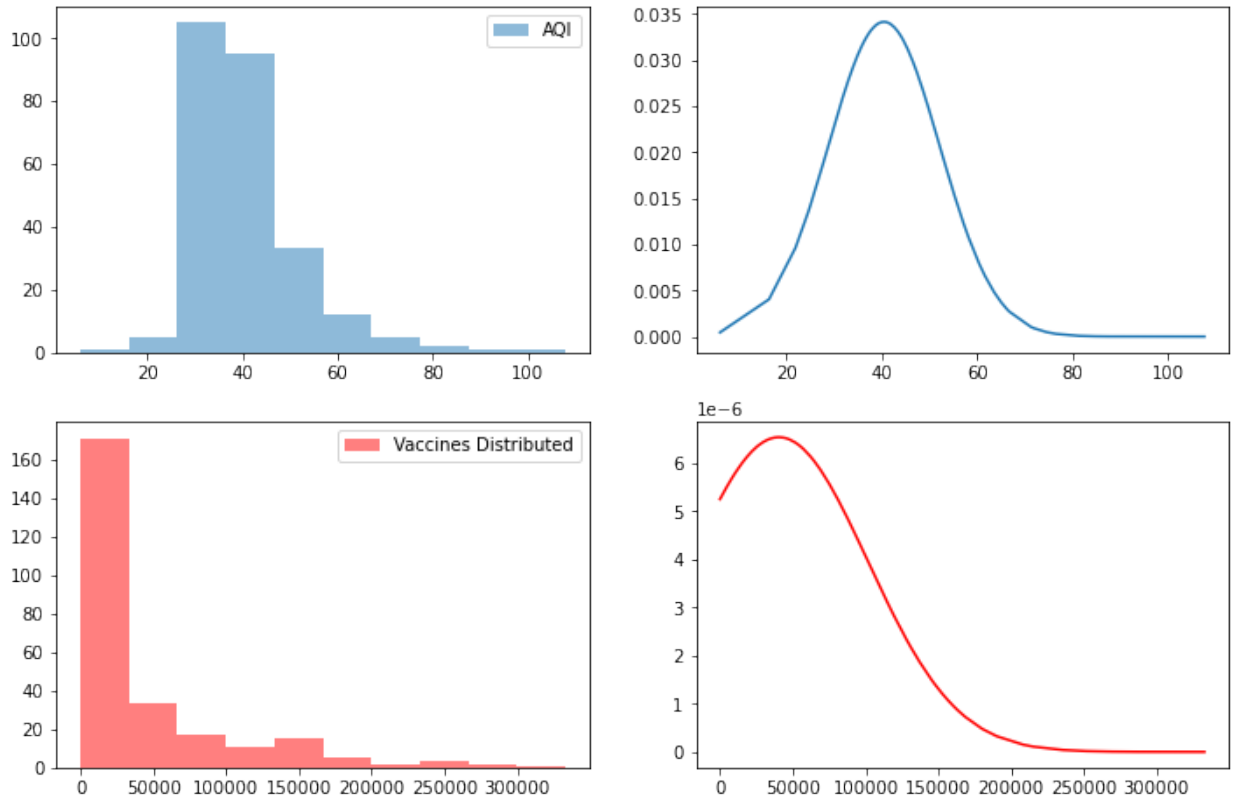
```
KS Statistic 0.6807692307692315
Pearson Correlation Coefficient 0.01600651989255773
```

In [271]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_ma["Date"], vaccines_aqi_ma["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_ma["Date"], vaccines_aqi_ma["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [286]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_ma.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_ma.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[286]: <matplotlib.legend.Legend at 0x7f7db8878ca0>

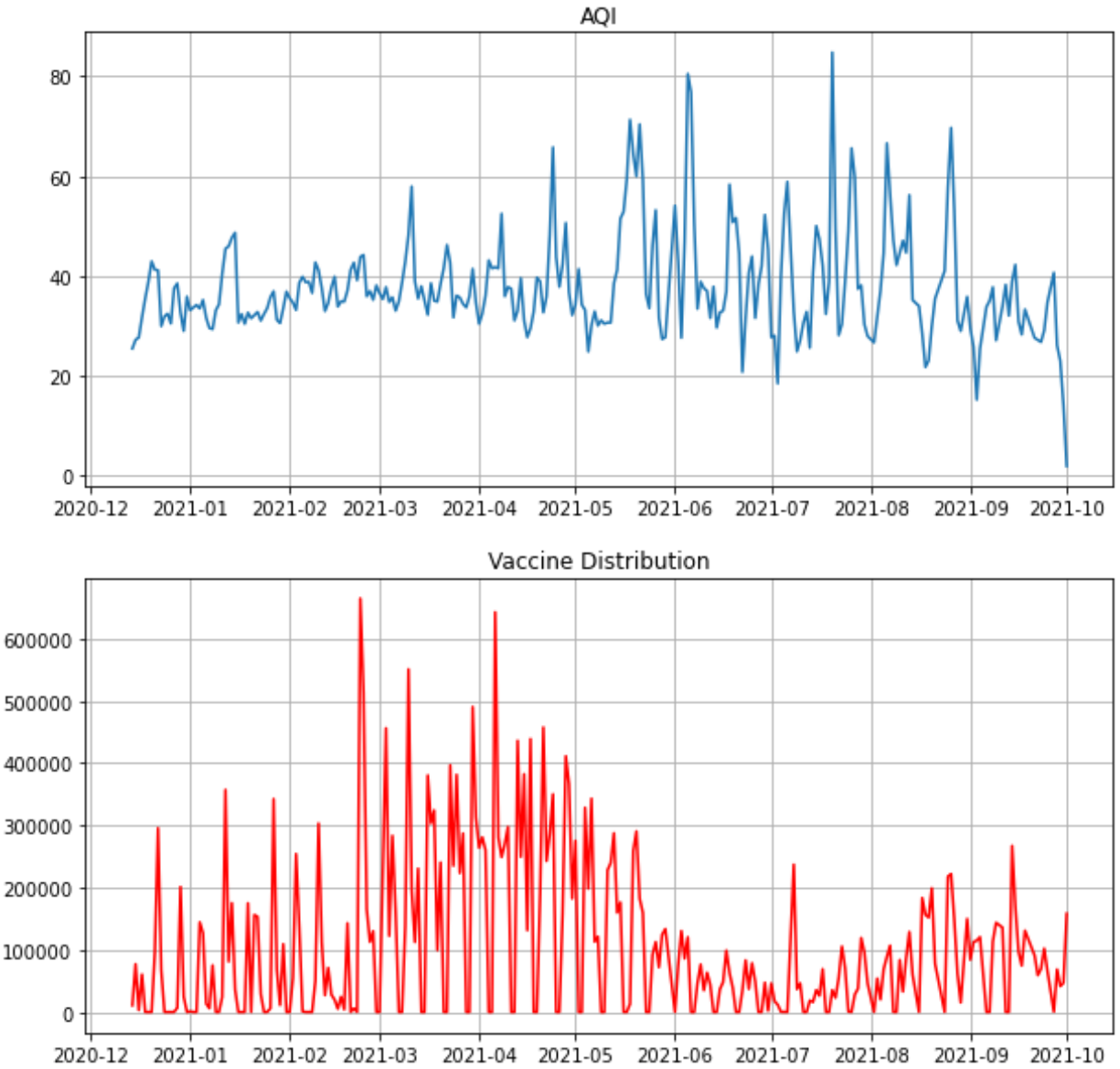


New York

In [301]:
```
vaccines_ny = get_state_wise_daily(vaccines_filtered, "NY")
vaccines_ny = vaccines_ny[vaccines_ny["Distributed"] >= 0]
vaccines_ny.head()

aqi_ny = aqi[["state", "Date", "AQI", "Category", "defining_param"]][a

aqi_ny["Date"] = pd.to_datetime(aqi_ny["Date"])
aqi_ny.sort_values(by="Date", inplace=True)
aqi_ny.reset_index(drop=True, inplace=True)
aqi_ny.head()

aqi_ny_mean = aqi_ny[["state", "Date", "AQI", "Category", "defining_pa
aqi_ny_mean.reset_index(inplace=True)
aqi_ny_mean.head()

vaccines_aqi_ny = pd.merge(vaccines_ny, aqi_ny_mean, on=("Date"))
vaccines_aqi_ny.head()

vaccines_ny_x, vaccines_ny_y = generate_eCDF(vaccines_aqi_ny.Distribut
aqi_ny_x, aqi_ny_y = generate_eCDF(vaccines_aqi_ny.AQI.to_list())
print("KS Statistic", ks_test_2_sample(vaccines_ny_x, vaccines_ny_y, a

print("Pearson Correlation Coefficient", pearson_corr(vaccines_aqi_ny)
```

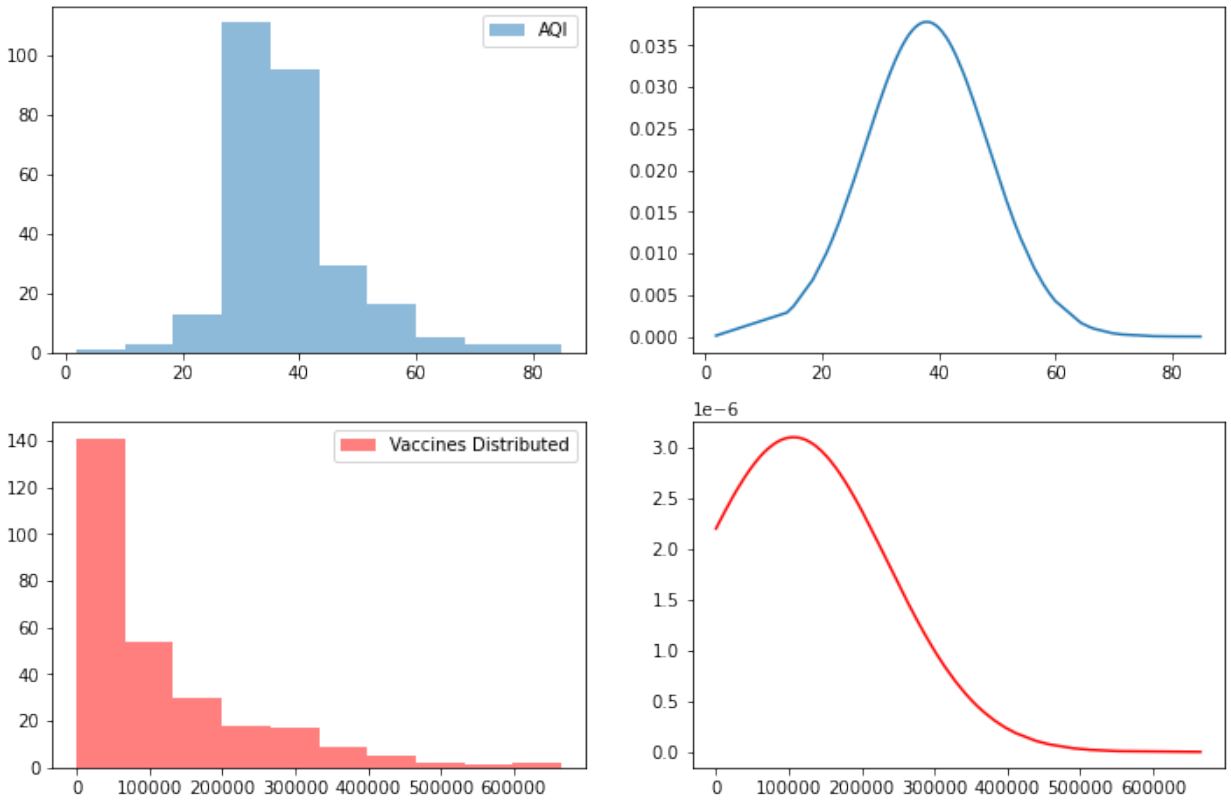
```
KS Statistic 0.7383512544802865
Pearson Correlation Coefficient 0.05423943373227678
```

In [273]:
```
f, axs = plt.subplots(2, 1, figsize=(10, 10))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(vaccines_aqi_ny["Date"], vaccines_aqi_ny["AQI"])
axs[0].title.set_text("AQI")
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(vaccines_aqi_ny["Date"], vaccines_aqi_ny["Distributed"], c
axs[1].title.set_text("Vaccine Distribution")
# plt.show()
```



In [285]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics

x_axis = sorted(vaccines_aqi_ny.AQI.to_list())
x_axis_2 = sorted(vaccines_aqi_ny.Distributed.to_list())

mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

mean2 = statistics.mean(x_axis_2)
sd2 = statistics.stdev(x_axis_2)

f, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0][0].hist(x_axis, alpha=0.5, label="AQI")
axs[0][1].plot(x_axis, norm.pdf(x_axis, mean, sd))
axs[0][0].legend(loc="upper right")
# axs[0][1].legend(loc="upper right")

axs[1][0].hist(x_axis_2, alpha=0.5, label="Vaccines Distributed", colo
axs[1][1].plot(x_axis_2, norm.pdf(x_axis_2, mean2, sd2), color="red")
axs[1][0].legend(loc="upper right")
# axs[1][1].legend(loc="upper right")
```

Out[285]: <matplotlib.legend.Legend at 0x7f7e1ddd9280>



Results

The results of theses states are again the same as before, which signifies that the vaccine manufacturing and distribution does not provide dependency to the AQI of those states.

Inference 2 - Correlation analysis of AQI values and number of deaths

In [443]:
```
import pandas as pd
import numpy as np
import collections
import csv
import math
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [444]:
```
# Reading CSV files
df = pd.read_csv('/Users/harikabandarupally/Desktop/SBU/Spring22/CSE 54
aqi1 = pd.read_csv('/Users/harikabandarupally/Desktop/SBU/Spring22/CSE 
aqi2 = pd.read_csv('/Users/harikabandarupally/Desktop/SBU/Spring22/CSE 
```

In [445]:
```
print(len(aqi1),len(aqi2))
```

```
338195 218196
```

In [590]:
```
# Dataframe containing data pertaining to cases and deaths during Covid
df.head()
```

Out[590]:

	Date	State Name	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_death	conf
31740	2021-01-01	AK	46241	NaN	NaN	470	0.0	285	
29287	2022-01-01	AK	151583	NaN	NaN	0	0.0	947	
19986	2021-01-02	AK	46530	NaN	NaN	289	0.0	287	
20302	2022-01-02	AK	151583	NaN	NaN	0	0.0	947	
31873	2021-01-03	AK	46698	NaN	NaN	168	0.0	289	

Data Pre-processing

In [448]:
```
# Finding if nulls are present
df['new_death'].isnull().sum()
```

Out[448]: 0

In [449]:
```
# Creating a merged AQI data frame for years 2020 and 2021
aqi = pd.concat([aqi1,aqi2],ignore_index=True)
```

In [450]:
```
aqi.head()
```

Out[450]:

	State Name	county Name	State Code	County Code	Date	AQI	Category	Defining Parameter	Defining Site	Number of Sites Reporting
0	Alabama	Baldwin	1	3	2020-01-01	48	Good	PM2.5	01-003-0010	1
1	Alabama	Baldwin	1	3	2020-01-04	13	Good	PM2.5	01-003-0010	1
2	Alabama	Baldwin	1	3	2020-01-07	14	Good	PM2.5	01-003-0010	1
3	Alabama	Baldwin	1	3	2020-01-10	39	Good	PM2.5	01-003-0010	1
4	Alabama	Baldwin	1	3	2020-01-13	29	Good	PM2.5	01-003-0010	1

In [451]:
```
# Finding if nulls are present
aqi['AQI'].isnull().sum()
```

Out[451]: 0

In [452]:
```
# New dataframe with columns State Name, Date and AQI after groupby
aqi_fin = aqi[['State Name', 'Date', 'AQI']].groupby(['State Name', 'Da
```

In [453]:
```
# Resetting Indices
aqi_fin = aqi_fin.reset_index()
```

In [454]:
```
aqi_fin.head()
```

Out[454]:

	State Name	Date	AQI
0	Alabama	2020-01-01	34.076923
1	Alabama	2020-01-02	25.777778
2	Alabama	2020-01-03	18.666667
3	Alabama	2020-01-04	20.642857
4	Alabama	2020-01-05	25.555556

In [455]:
```
# AQI data has states in the form of complete names whereas Covid data
state_to_id = {
'Alabama' : 'AL',
'Alaska' : 'AK',
'Arizona' : 'AZ',
'Arkansas' : 'AR',
'California' : 'CA',
'Colorado' : 'CO',
'Connecticut' : 'CT',
'Delaware' : 'DE',
'District of Columbia' : 'DC',
'Florida' : 'FL',
'Georgia' : 'GA',
'Hawaii' : 'HI',
'Idaho' : 'ID',
'Illinois' : 'IL',
'Indiana' : 'IN',
'Iowa' : 'IA',
'Kansas' : 'KS',
'Kentucky' : 'KY',
'Louisiana' : 'LA',
'Maine' : 'ME',
'Maryland' : 'MD',
'Massachusetts' : 'MA',
'Michigan' : 'MI',
'Minnesota' : 'MN',
'Mississippi' : 'MS',
'Missouri' : 'MO',
'Montana' : 'MT',
'Nebraska' : 'NE',
'Nevada' : 'NV',
'New Hampshire' : 'NH',
'New Jersey' : 'NJ',
'New Mexico' : 'NM',
'New York' : 'NY',
'North Carolina' : 'NC',
'North Dakota' : 'ND',
```

```
'Ohio' : 'OH',
'Oklahoma' : 'OK',
'Oregon' : 'OR',
'Pennsylvania' : 'PA',
'Rhode Island' : 'RI',
'South Carolina' : 'SC',
'South Dakota' : 'SD',
'Tennessee' : 'TN',
'Texas' : 'TX',
'Utah' : 'UT',
'Vermont' : 'VT',
'Virginia' : 'VA',
'Washington' : 'WA',
'West Virginia' : 'WV',
'Wisconsin' : 'WI',
'Wyoming' : 'WY'}
```

In [456]:
```
for d in state_to_id:
    aqi_fin = aqi_fin.replace(d,state_to_id[d])
```

In [457]:
```
# Sorting by state name and date
aqi_fin = aqi_fin.sort_values(by=['State Name','Date'])
```

In [458]:
```
aqi_fin.head()
```

Out[458]:

	State Name	Date	AQI
609	AK	2020-01-01	27.250000
610	AK	2020-01-02	30.833333
611	AK	2020-01-03	59.333333
612	AK	2020-01-04	50.625000
613	AK	2020-01-05	64.166667

In [459]:
```
aqi_fin.dtypes
```

Out[459]:
```
State Name     object
Date           object
AQI           float64
dtype: object
```

In [460]:
```
# Convert Date column in AQI data to 'Datetime' type
aqi_fin["Date"] = pd.to_datetime(aqi_fin["Date"])
```

In [461]:
```
aqi_fin.dtypes
```

Out[461]:
```
State Name            object
Date          datetime64[ns]
AQI                  float64
dtype: object
```

In [462]:
```
# Sorting the dataframe by date and state
df = df.sort_values(by=['state','submission_date'])
```

In [463]:
```
df.head()
```

Out[463]:

	submission_date	state	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_de
31740	01/01/2021	AK	46241	NaN	NaN	470	0.0	
29287	01/01/2022	AK	151583	NaN	NaN	0	0.0	
19986	01/02/2021	AK	46530	NaN	NaN	289	0.0	
20302	01/02/2022	AK	151583	NaN	NaN	0	0.0	
31873	01/03/2021	AK	46698	NaN	NaN	168	0.0	

In [464]:
```
df.dtypes
```

Out[464]:
```
submission_date     object
state               object
tot_cases            int64
conf_cases         float64
prob_cases         float64
new_case             int64
pnew_case          float64
tot_death            int64
conf_death         float64
prob_death         float64
new_death            int64
pnew_death         float64
created_at          object
consent_cases       object
consent_deaths      object
dtype: object
```

In [465]:
```
# Renaming columns in Covid dataframe to match with AQI dataframe
df = df.rename(columns={'submission_date': 'Date', 'state': 'State Name
```

Illustration of number of deaths in a few states in the US

Based on a research from Harvard, the number of deaths due to covid in regions where the AQI is higher (i.e. the places where pollution is higher, the hypothesis is that the number of deaths is also higher). This research was carried out for the countries round the globe. We wanted to understant the influence of AQI on deaths in the United States.

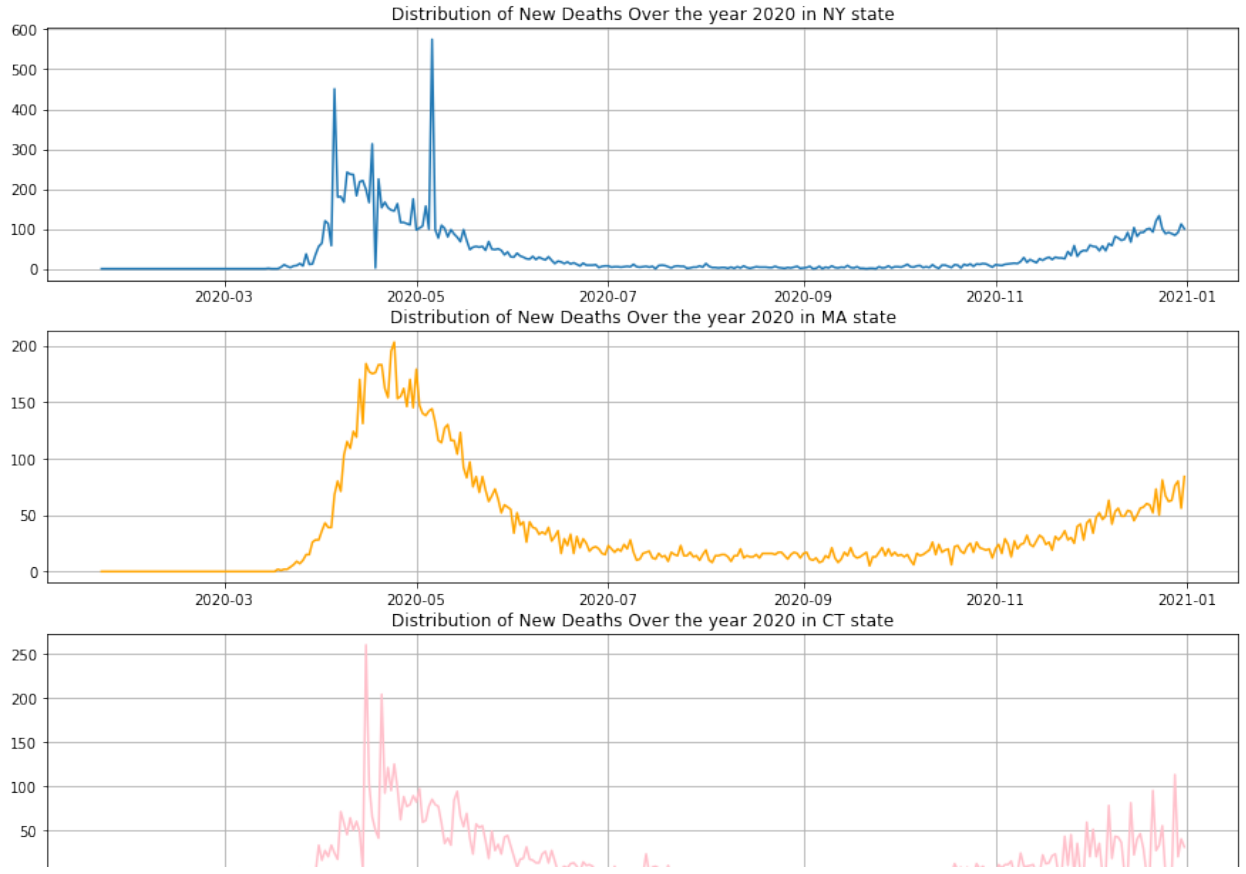
Hence,we are performing this **Pearson Correlation** experimental study to understand if AQI and number of deaths due to covid are positively correlated.

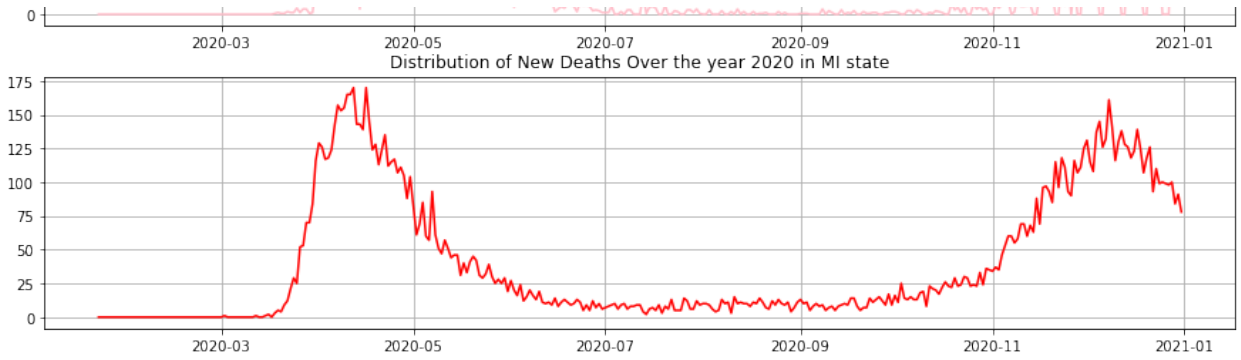
It can be seen from the following, that when we carried out the experiment over whole 2020 data, we get very low (0.074). Our hypothesis was that AQI and deaths are correlated, but the reason for this could be that the number of deaths suddenly rose between March to June 2020. We can also observe this from the graphs plotted below for the states New York, Massachutes, Michigan and Connecticut (considering these for simplicity, as these states have manufacturing units for the vaccines - J&J, Moderna and Pfizer).

Based on the illustration plotted below we can see that deaths rose from mid March to June. Hence, we are considering these months for realising the correlation between AQI and Number of deaths

In [598]:
```
# States considered - New York, Connnecticut, Maschacusets
df_total_NY = df[(df["Date"].dt.year == 2020) & (df["State Name"] == 'N
df_total_MA = df[(df["Date"].dt.year == 2020) & (df["State Name"] == 'M
df_total_MI = df[(df["Date"].dt.year == 2020) & (df["State Name"] == 'M
df_total_CT = df[(df["Date"].dt.year == 2020) & (df["State Name"] == 'C
```

In [604]:
```
# Graph plot for the states and new deaths over the year 2020
f, axs = plt.subplots(4, 1, figsize=(15, 15))
# axs[0].subplot(2, 1, 1)
axs[0].grid()
axs[0].plot(df_total_NY["Date"], df_total_NY["new_death"].astype(int))
axs[0].title.set_text("Distribution of New Deaths Over the year 2020 in
# plt.subplot(2, 1, 2)
axs[1].grid()
axs[1].plot(df_total_MA["Date"], df_total_MA["new_death"].astype(int), 
axs[1].title.set_text("Distribution of New Deaths Over the year 2020 in
# plt.show()
axs[2].grid()
axs[2].plot(df_total_CT["Date"], df_total_CT["new_death"].astype(int), 
axs[2].title.set_text("Distribution of New Deaths Over the year 2020 in
# plt.show()
axs[3].grid()
axs[3].plot(df_total_MI["Date"], df_total_MI["new_death"].astype(int), 
axs[3].title.set_text("Distribution of New Deaths Over the year 2020 in
# plt.show()
```





In [605]:
```
# Creating a subset dataframe based on above hypothesis
df["Date"] = pd.to_datetime(df["Date"])
df_fin = df
df_fin = df[(df["Date"].dt.month >=3) & (df["Date"].dt.month <=6) & (df
df_total = df[(df["Date"].dt.month >=1) & (df["Date"].dt.month <=12) &
```

In [567]:
```
# Merging the covid dataframe and AQI dataframe on Date and State Name
fin = pd.merge(aqi_fin, df_fin, on=("Date","State Name"), how='inner')
```

In [606]:
```
# Merging the covid dataframe and AQI dataframe on Date and State Name
fin_total = pd.merge(aqi_fin, df_total, on=("Date","State Name"), how='
```

In [568]:
```
fin.head()
```

Out[568]:

	State Name	Date	AQI	tot_cases	conf_cases	prob_cases	new_case	pnew_case	tot_death
0	AK	2020-02-01	38.000000	0	NaN	NaN	0	NaN	0
1	AK	2020-02-02	31.000000	0	NaN	NaN	0	NaN	0
2	AK	2020-02-03	21.777778	0	NaN	NaN	0	NaN	0
3	AK	2020-02-04	29.166667	0	NaN	NaN	0	NaN	0
4	AK	2020-02-05	32.333333	0	NaN	NaN	0	NaN	0

In [569]:
```
len(fin)
```

Out[569]: 7550

In [ ]:
```
# Outlier detection on merged dataframe
def outlier_detection(X, column_name):
    values = sorted(X[column_name])
    Q1 = values[int(np.ceil(len(values) / 4))]
    Q3 = values[int(np.ceil(3 * len(values) / 4))]
    iqr = float(Q3 - Q1)
    outliers = X.loc[(X[column_name] > (Q3 + (1.5 * iqr))) | (X[column_
    return outliers.index
```

In [571]:
```
# Finding the outliers
total_outliers = []
column_names = ['new_death']
for c in column_names:
    outliers = outlier_detection(fin, c)
    total_outliers.extend(outliers)
total_outliers = set(total_outliers)
```

In [572]:
```
# Keeping desired indices
desired_index = [i for i in fin.index if i not in total_outliers]
len(desired_index)
fin_without_outliers = fin.loc[desired_index]
```

In [573]:
```
len(fin_without_outliers)
```

Out[573]: 6575

In [574]:
```
# Pearson Correlation Test

def correlationCoeff(fin):
    num = np.sum((fin['new_death']-fin['new_death'].mean())*(fin['AQI']-
    den = np.sqrt(np.sum(pow(fin['new_death']-fin['new_death'].mean(),2)
    coeff = num/den

    return coeff
```

In [607]:
```
print("For the year 2020, correlation coeff between AQI and deaths due
correlationCoeff(fin_total)
```

```
For the year 2020, correlation coeff between AQI and deaths due to cov
id is
```

Out[607]: 0.07499495427412275

In [608]:
```
print("For the months of March to June in 2020, correlation coeff betwe
correlationCoeff(fin_without_outliers)
```

```
For the months of March to June in 2020, correlation coeff between AQI
and deaths due to covid is
```

Out[608]:
```
0.24128956898554116
```

Analysis

As mentioned above, our hypothesis is that AQI and number of deaths are positively correlated. And from the result we can observe that there exists a weak albeit positive correlation between between AQI and the number of deaths. This could be because air pollution and the quality of the air could add on to the covid disease which maily impact the lungs of human system. Again as mentioned above, for the entire 2020 data we see that there is an extremely weak correlation, and for the months where covid deaths were actually high, the correlation between AQI and number of deaths seemed decently correlated in a positive manner.

In [43]:

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.stats import geom
from scipy.stats import binom
from itertools import permutations
import math
from statistics import mean
import random
from scipy.stats import gamma
import matplotlib.pyplot as  plt
import copy
import scipy.stats
```

In [44]:

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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, ca
ll drive.mount("/content/gdrive", force_remount=True).
```

Import Cases, Vaccination and AQI datasets (X dataset)

In [45]:

```
cases_df=pd.read_csv("/content/gdrive/MyDrive/United_States_COVID-19_Cases_and_Deaths_by_St
vaccine_df=pd.read_csv("/content/gdrive/MyDrive/COVID-19_Vaccinations_in_the_United_States_
aqi_2021=pd.read_csv("/content/gdrive/MyDrive/daily_aqi_by_county_2021.csv")
aqi_2020=pd.read_csv("/content/gdrive/MyDrive/daily_aqi_by_county_2020.csv")
vaccine_daily_de=pd.read_csv("/content/gdrive/MyDrive/delaware_vaccines.csv")
vaccine_daily_id=pd.read_csv("/content/gdrive/MyDrive/idaho_vaccines.csv")
co_2020 = pd.read_csv('/content/gdrive/MyDrive/daily_42101_2020.csv')
co_2021 = pd.read_csv('/content/gdrive/MyDrive/daily_42101_2021.csv')
```

Inference 3

In [46]:

```
aqi=pd.concat([aqi_2020,aqi_2021])
aqi['Date'] = pd.to_datetime(aqi['Date'])
# aqi=aqi[aqi["State Name"]=="Idaho"]
aqi
```

Out[46]:

	State Name	county Name	State Code	County Code	Date	AQI	Category	Defining Parameter	Defining Site	Numbe of Sites Reportin
0	Alabama	Baldwin	1	3	2020-01-01	48	Good	PM2.5	01-003-0010	
1	Alabama	Baldwin	1	3	2020-01-04	13	Good	PM2.5	01-003-0010	
2	Alabama	Baldwin	1	3	2020-01-07	14	Good	PM2.5	01-003-0010	
3	Alabama	Baldwin	1	3	2020-01-10	39	Good	PM2.5	01-003-0010	
4	Alabama	Baldwin	1	3	2020-01-13	29	Good	PM2.5	01-003-0010	
...	...	...	...	...	...	...	...	...	...	..
218191	Wyoming	Weston	56	45	2021-06-26	41	Good	Ozone	56-045-0003	
218192	Wyoming	Weston	56	45	2021-06-27	40	Good	Ozone	56-045-0003	
218193	Wyoming	Weston	56	45	2021-06-28	41	Good	Ozone	56-045-0003	
218194	Wyoming	Weston	56	45	2021-06-29	48	Good	Ozone	56-045-0003	
218195	Wyoming	Weston	56	45	2021-06-30	54	Moderate	Ozone	56-045-0003	

556391 rows × 10 columns

In [47]:

```
list(vaccine_daily_id["Date"][vaccine_daily_id["Series_Complete_Yes"]>480000])[0]
```

Out[47]:

```
'2021-04-30'
```

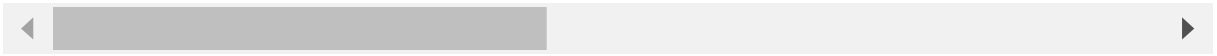
In [48]:

```
co_2020['Date Local']=pd.to_datetime(co_2020['Date Local'])
# co_2020=co_2020.sort_values(by='Date Local')
co_2021['Date Local']=pd.to_datetime(co_2021['Date Local'])
# co_2021=co_2021.sort_values(by='Date Local')
co_2020
```

Out[48]:

	State Code	County Code	Site Num	Parameter Code	POC	Latitude	Longitude	Datum	Parameter Name	Sam Dura
0	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
1	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
2	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
3	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
4	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
...	...	...	...	...	...	...	...	...	...	
179333	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
179334	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
179335	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
179336	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
179337	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC

179338 rows × 29 columns



In [49]:

```
co=pd.concat([co_2020,co_2021])
co['Date Local'] = pd.to_datetime(co['Date Local'])
# aqi=aqi[aqi["State Name"]=="Idaho"]
co
```

Out[49]:

	State Code	County Code	Site Num	Parameter Code	POC	Latitude	Longitude	Datum	Parameter Name	Sam Dura
0	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
1	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
2	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
3	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
4	1	73	23	42101	2	33.553056	-86.81500	WGS84	Carbon monoxide	1 HC
...	...	...	...	...	...	...	...	...	...	
106547	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
106548	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
106549	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
106550	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC
106551	72	127	3	42101	1	18.449814	-66.05251	WGS84	Carbon monoxide	8 F A E HC

285890 rows × 29 columns

In [50]:

```
co['AQI'] = co['AQI'].fillna(0)
```

In [51]:

```
co['AQI']
```

Out[51]:

```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
         ...
106547    8.0
106548    7.0
106549    8.0
106550    7.0
106551    5.0
Name: AQI, Length: 285890, dtype: float64
```

In [52]:

```
state_to_id = {
'Alabama' : 'AL',
'Alaska' : 'AK',
'Arizona' : 'AZ',
'Arkansas' : 'AR',
'California' : 'CA',
'Colorado' : 'CO',
'Connecticut' : 'CT',
'Delaware' : 'DE',
'District of Columbia' : 'DC',
'Florida' : 'FL',
'Georgia' : 'GA',
'Hawaii' : 'HI',
'Idaho' : 'ID',
'Illinois' : 'IL',
'Indiana' : 'IN',
'Iowa' : 'IA',
'Kansas' : 'KS',
'Kentucky' : 'KY',
'Louisiana' : 'LA',
'Maine' : 'ME',
'Maryland' : 'MD',
'Massachusetts' : 'MA',
'Michigan' : 'MI',
'Minnesota' : 'MN',
'Mississippi' : 'MS',
'Missouri' : 'MO',
'Montana' : 'MT',
'Nebraska' : 'NE',
'Nevada' : 'NV',
'New Hampshire' : 'NH',
'New Jersey' : 'NJ',
'New Mexico' : 'NM',
'New York' : 'NY',
'North Carolina' : 'NC',
'North Dakota' : 'ND',
'Ohio' : 'OH',
'Oklahoma' : 'OK',
'Oregon' : 'OR',
'Pennsylvania' : 'PA',
'Rhode Island' : 'RI',
'South Carolina' : 'SC',
'South Dakota' : 'SD',
'Tennessee' : 'TN',
'Texas' : 'TX',
'Utah' : 'UT',
'Vermont' : 'VT',
'Virginia' : 'VA',
'Washington' : 'WA',
'West Virginia' : 'WV',
'Wisconsin' : 'WI',
'Wyoming' : 'WY'}
```

In [53]:

```
co_fin = co[['State Name', 'Date Local', 'AQI']].groupby(['State Name', 'Date Local']).agg(
co_fin = co_fin.reset_index()
```

In [54]:

```
co_fin
```

Out[54]:

	State Name	Date Local	AQI
0	Alabama	2020-01-01	2.666667
1	Alabama	2020-01-02	1.500000
2	Alabama	2020-01-03	2.166667
3	Alabama	2020-01-04	1.000000
4	Alabama	2020-01-05	2.333333
...	...	...	...
31017	Wyoming	2021-09-27	1.000000
31018	Wyoming	2021-09-28	1.000000
31019	Wyoming	2021-09-29	1.000000
31020	Wyoming	2021-09-30	0.500000
31021	Wyoming	2021-10-01	1.000000

31022 rows × 3 columns

In [55]:

```
for d in state_to_id:
    co_fin = co_fin.replace(d,state_to_id[d])
```

In [56]:

```
co_fin
```

Out[56]:

	State Name	Date Local	AQI
0	AL	2020-01-01	2.666667
1	AL	2020-01-02	1.500000
2	AL	2020-01-03	2.166667
3	AL	2020-01-04	1.000000
4	AL	2020-01-05	2.333333
...	...	...	...
31017	WY	2021-09-27	1.000000
31018	WY	2021-09-28	1.000000
31019	WY	2021-09-29	1.000000
31020	WY	2021-09-30	0.500000
31021	WY	2021-10-01	1.000000

31022 rows × 3 columns

In [57]:

```
co=co_fin[co_fin["State Name"]=="ID"]
```

We compare the 3 distributions before and after 4800000 double doses were completed. The AQI distribution of both periods, AQI cause by carbon monoxide distribution of both periods, AQI vs CO distribution of the before period and after period. We assume that after this amount of double doses were completed the lockdown was lifted. We check if this causes a change in AQI. We use KS test since we wanted to compare 2 disributions and it is applicable since it has no assumption

In [58]:

```
before_data_co=list(co["AQI"][(co["Date Local"]>='2020-05-01') & (co["Date Local"]<'2020-12
after_data_co=list(co["AQI"][(co["Date Local"]>=list(vaccine_daily_id["Date"][vaccine_daily
```

In [59]:

```
def ks_test_2_sample(X1, Y1, X2, Y2):
    tot_max = -1
    ks_table = np.zeros((len(X1),6))
    for i in range(len(ks_table)-1):
        ks_table[i,0] = Y1[i]
        ks_table[i,1] = Y1[i+1]
        ks_table[i,2]=0
        ks_table[i,3]=0
        for j in X2:
            if j<X1[i]:
                ks_table[i,2]+=1
            if j<=X1[i]:
                ks_table[i,3]+=1

        ks_table[i,3]/=len(X2)
        ks_table[i,2]/=len(X2)

        ks_table[i,4] = abs(ks_table[i,0] - ks_table[i,2])
        ks_table[i,5] = abs(ks_table[i,1] - ks_table[i,3])
        cmax = max(ks_table[i,4], ks_table[i,5])
        if cmax > tot_max:
            tot_max = cmax
            x1_max = X1[i]
            y1_max = ks_table[i,0]
            y2_max = ks_table[i,2]
    if tot_max>0.05:
        print("We reject the null hypotheis that the 2 distributions follow the same distri
    else:
        print("We accept the null hypotheis that the 2 distributions follow the same distri
    return tot_max
```

In [60]:

```
def generate_eCDF(X):
  n = len(X)
  Srt = sorted(X)
  delta = .1
  X = [min(Srt)-delta]
  Y = [0]
  for i in range(0, n):
      X = X + [Srt[i], Srt[i]]
      Y = Y + [Y[len(Y)-1], Y[len(Y)-1]+(1/n)]
  X = X + [max(Srt)+delta]
  # print(X)
  Y = Y + [1]
  return X, Y
```

In [61]:

```
before_X, before_Y = generate_eCDF(before_data_co)
after_X, after_Y = generate_eCDF(after_data_co)
print(ks_test_2_sample(before_X, before_Y, after_X, after_Y))
```

```
We reject the null hypotheis that the 2 distributions follow the same distri
bution
0.24598422897196515
```

AQI

In [62]:

```
aqi_fin = aqi[['State Name', 'Date', 'AQI']].groupby(['State Name', 'Date']).agg({'AQI': 'm
aqi_fin = aqi_fin.reset_index()
```

In [63]:

```
aqi_fin = aqi[['State Name', 'Date', 'AQI']].groupby(['State Name', 'Date']).agg({'AQI': 'm
aqi_fin = aqi_fin.reset_index()
```

In [64]:

```
for d in state_to_id:
    aqi_fin = aqi_fin.replace(d,state_to_id[d])
```

In [65]:

```
aqi=aqi_fin[aqi_fin["State Name"]=="ID"]
```

In [66]:

```
before_data=list(aqi["AQI"][(aqi["Date"]>='2020-05-01') & (aqi["Date"]<'2020-12-01')])
after_data=list(aqi["AQI"][(aqi["Date"]>=list(vaccine_daily_id["Date"][vaccine_daily_id["Se
```

In [67]:

```
before_X, before_Y = generate_eCDF(before_data)
after_X, after_Y = generate_eCDF(after_data)
print(ks_test_2_sample(before_X, before_Y, after_X, after_Y))
```

```
We reject the null hypotheis that the 2 distributions follow the same distri
bution
0.2334338257461528
```

In [68]:

```
before_X, before_Y = generate_eCDF(before_data)
before_X_co, before_Y_co = generate_eCDF(before_data_co)
# after_X, after_Y = generate_eCDF(after_data)
print(ks_test_2_sample(before_X, before_Y, before_X_co, before_Y_co))
```

```
We reject the null hypotheis that the 2 distributions follow the same distri
bution
1.0
```

In [69]:

```
after_X, after_Y = generate_eCDF(after_data)
after_X_co, after_Y_co = generate_eCDF(after_data_co)
print(ks_test_2_sample(after_X, after_Y, after_X_co, after_Y_co))
```

```
We reject the null hypotheis that the 2 distributions follow the same distri
bution
1.0
```

In [69]:

```

```