SUYASH PRATAP SINGH(181B226)

TASKS:-

1. Download the csv file for time series data about Covid-19 cases in India.
2. Pre-process the data to check about any missing data.
3. Plot the time series to visualise it.
4. Identify the trend, seasonality and remainder parts of the time series.
5. Identify the SARIMAX parameters
6. Forecast the time series for next 100 days.
7. Plot your forecasted data along with the original data.

In [1]:

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
```

In [2]:

```
kp = pd.read_csv(r'C:\Users\Admin\Downloads\covid_19_india.csv')
```

In [3]:
```
#print first five row
kp.head()
```

Out[3]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational	Cu
0	1	30-01-2020	6:00 PM	Kerala	1	0	
1	2	31-01-2020	6:00 PM	Kerala	1	0	
2	3	01-02-2020	6:00 PM	Kerala	2	0	
3	4	02-02-2020	6:00 PM	Kerala	3	0	
4	5	03-02-2020	6:00 PM	Kerala	3	0	

In [4]:
```
#print random five row
kp.sample(5)
```

Out[4]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational
1716	1717	08-05-2020	8:00 AM	Himachal Pradesh	-	-
4886	4887	06-08-2020	8:00 AM	Chandigarh	-	-
3832	3833	07-07-2020	8:00 AM	Jammu and Kashmir	-	-
6992	6993	05-10-2020	8:00 AM	Haryana	-	-
2944	2945	12-06-2020	8:00 AM	Puducherry	-	-

In [5]:
```
#full information
kp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8486 entries, 0 to 8485
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Sno                       8486 non-null   int64 
 1   Date                      8486 non-null   object
 2   Time                      8486 non-null   object
 3   State/UnionTerritory      8486 non-null   object
 4   ConfirmedIndianNational   8486 non-null   object
 5   ConfirmedForeignNational  8486 non-null   object
 6   Cured                     8486 non-null   int64 
 7   Deaths                    8486 non-null   int64 
 8   Confirmed                 8486 non-null   int64 
dtypes: int64(4), object(5)
memory usage: 596.8+ KB
```

In [6]:
```
#statistical information
kp[['Date','Time']].describe()
```

Out[6]:

	Date	Time
count	8486	8486
unique	292	7
top	11-06-2020	8:00 AM
freq	37	6848

In [7]:
```
#covariance of dataset
kp[['Deaths','Confirmed']].cov()
```

Out[7]:

	Deaths	Confirmed
Deaths	1.751980e+07	7.229719e+08
Confirmed	7.229719e+08	3.557989e+10

In [8]:
```
#correlation of dataset
kp[['Deaths','Confirmed']].corr()
```

Out[8]:

	Deaths	Confirmed
Deaths	1.000000	0.915703
Confirmed	0.915703	1.000000

```
In [9]: # unstacking the data
        kp.unstack().head()
```

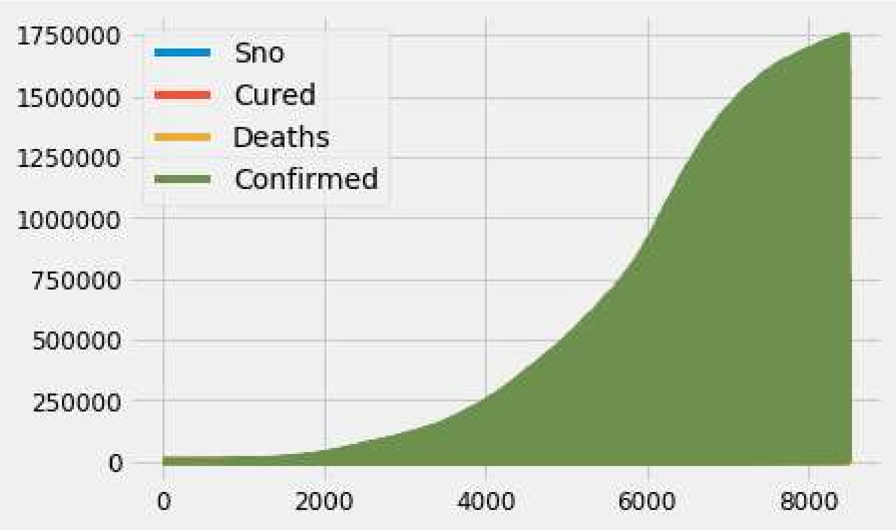
```
Out[9]: Sno  0    1
             1    2
             2    3
             3    4
             4    5
        dtype: object
```

```
In [10]: kp.unstack().head().values
```

```
Out[10]: array([1, 2, 3, 4, 5], dtype=object)
```

```
In [11]: kp.plot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x26795616640>
```

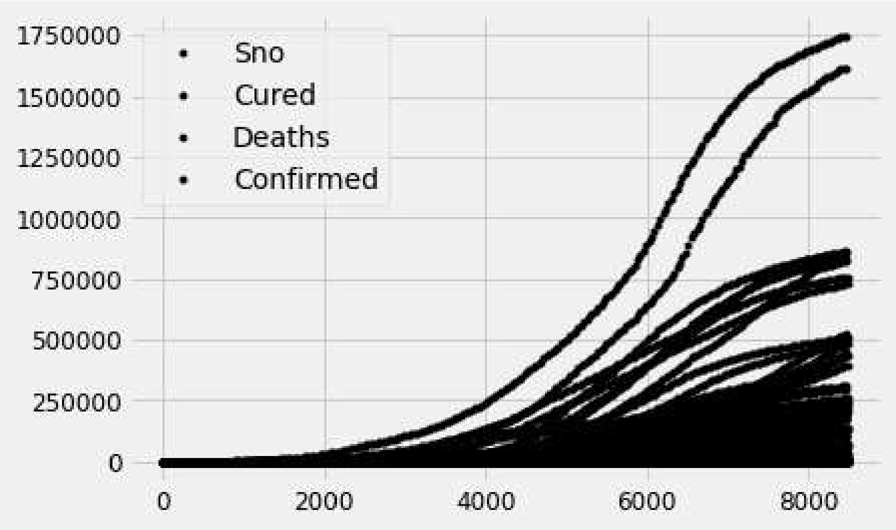


```
In [12]: kp['Date'].min(), kp['Date'].max()
```

```
Out[12]: ('01-02-2020', '31-10-2020')
```

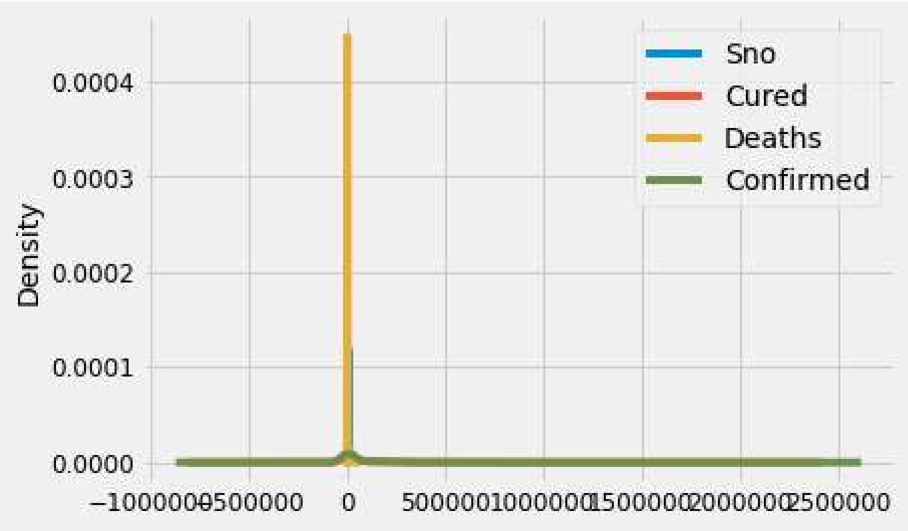
In [13]:
```
kp.plot(style='k.')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26797a67490>
```



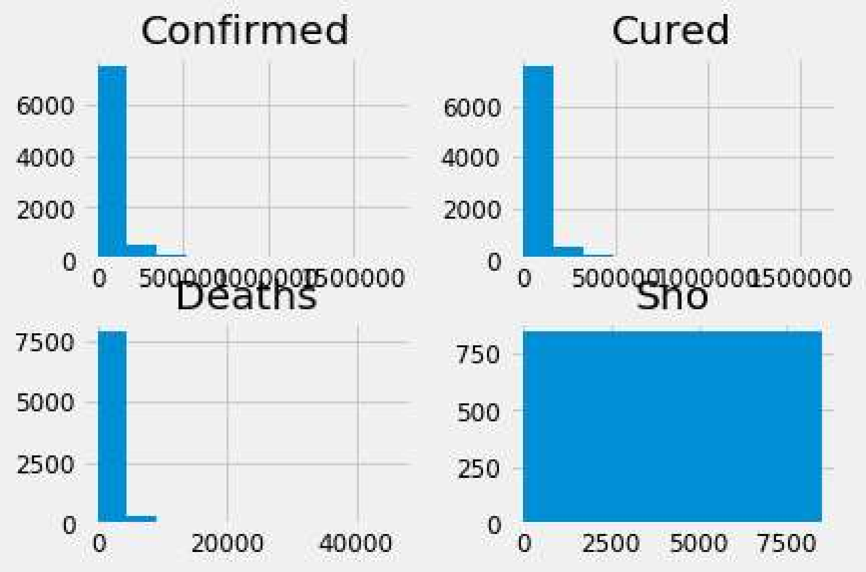
In [14]:
```
kp.plot(kind='kde')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26797ddf490>
```



In [15]:
```
kp.hist()
```

Out[15]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000026797DDB130>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000267977CC4F0
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000267978CD2E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000267978F3AC0
>]],
      dtype=object)
```



In [16]:
```
teju = kp.loc[kp['State/UnionTerritory'] == 'Uttar Pradesh']
teju.head()
```

Out[16]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational	Cu
39	40	04-03-2020	6:00 PM	Uttar Pradesh	6	0	
50	51	05-03-2020	6:00 PM	Uttar Pradesh	7	0	
55	56	06-03-2020	6:00 PM	Uttar Pradesh	7	0	
58	59	07-03-2020	6:00 PM	Uttar Pradesh	7	0	
72	73	08-03-2020	6:00 PM	Uttar Pradesh	7	0	

In [17]:
```
#shape of dataset
teju.shape
```

Out[17]:
```
(258, 9)
```

In [18]:

```
teju.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 258 entries, 39 to 8484
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Sno                       258 non-null    int64 
 1   Date                      258 non-null    object
 2   Time                      258 non-null    object
 3   State/UnionTerritory      258 non-null    object
 4   ConfirmedIndianNational   258 non-null    object
 5   ConfirmedForeignNational  258 non-null    object
 6   Cured                     258 non-null    int64 
 7   Deaths                    258 non-null    int64 
 8   Confirmed                 258 non-null    int64 
dtypes: int64(4), object(5)
memory usage: 20.2+ KB
```

In [19]:

```
teju.isnull()
```

Out[19]:

	Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational
39	False	False	False	False	False	False
50	False	False	False	False	False	False
55	False	False	False	False	False	False
58	False	False	False	False	False	False
72	False	False	False	False	False	False
...	...	...	...	...	...	...
8344	False	False	False	False	False	False
8379	False	False	False	False	False	False
8414	False	False	False	False	False	False
8449	False	False	False	False	False	False
8484	False	False	False	False	False	False

258 rows × 9 columns

In [20]:
```
teju.isnull().sum()
```

Out[20]:
```
Sno                         0
Date                        0
Time                        0
State/UnionTerritory        0
ConfirmedIndianNational     0
ConfirmedForeignNational    0
Cured                       0
Deaths                      0
Confirmed                   0
dtype: int64
```

In [21]:
```
teju.nunique()
```

Out[21]:
```
Sno                         258
Date                        258
Time                          7
State/UnionTerritory          1
ConfirmedIndianNational      18
ConfirmedForeignNational      3
Cured                       227
Deaths                      213
Confirmed                   247
dtype: int64
```

In [22]:
```
teju.columns
```

Out[22]:
```
Index(['Sno', 'Date', 'Time', 'State/UnionTerritory',
       'ConfirmedIndianNational', 'ConfirmedForeignNational', 'Cured',
       'Deaths', 'Confirmed'],
      dtype='object')
```

In [23]:
```
teju.index
```

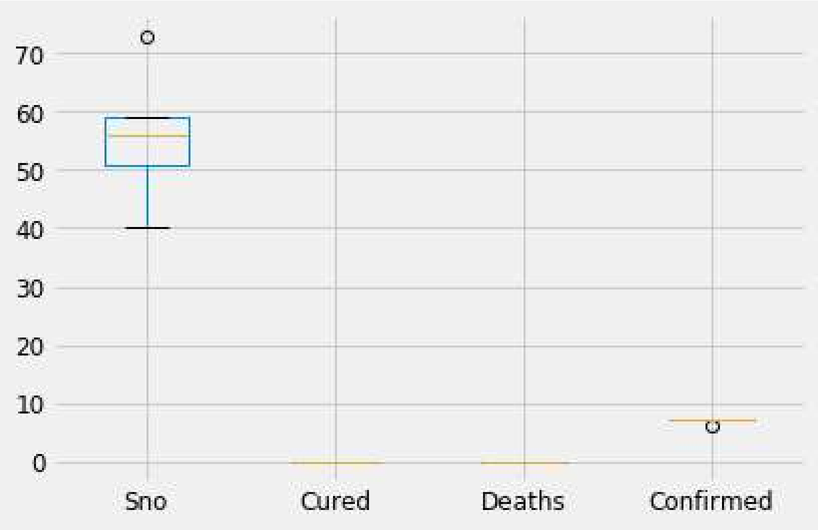
Out[23]:
```
Int64Index([  39,   50,   55,   58,   72,   82,   85,  103,  114,  127,
            ...
            8169, 8204, 8239, 8274, 8309, 8344, 8379, 8414, 8449, 8484],
           dtype='int64', length=258)
```

In [24]:

```
teju.head().boxplot()
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x267979f6c40>
```

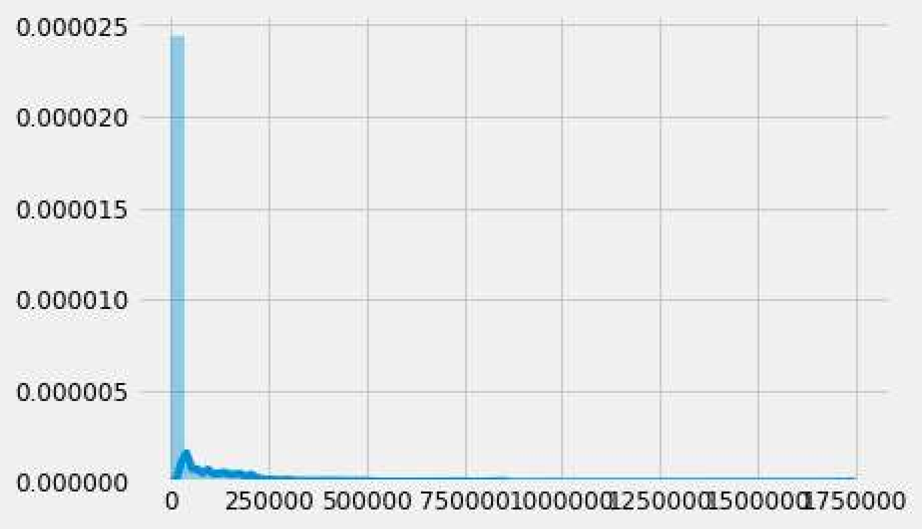


In [25]:

```
#Distplot
import seaborn as sns
sns.distplot(kp[['Deaths','Confirmed']])
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26797cd83a0>
```



In [26]:

```
cols = ['Sno','Time','State/UnionTerritory','ConfirmedIndianNational','Confirm
edForeignNational','Cured','Deaths']
teju['Date'] = teju['Date']+' '+teju['Time']
teju.drop(cols, axis=1, inplace=True)
teju = teju.sort_values('Date')
teju.isnull().sum()
```

Out[26]:

```
Date         0
Confirmed    0
dtype: int64
```

In [27]:
```
teju.head()
```

Out[27]:

	Date	Confirmed
556	01-04-2020 7:30 PM	103
1508	01-05-2020 5:00 PM	2281
2555	01-06-2020 8:00 AM	7823
3636	01-07-2020 8:00 AM	23492
4739	01-08-2020 8:00 AM	85461

In [28]:
```
teju.sample(3)
```

Out[28]:

	Date	Confirmed
2735	06-06-2020 8:00 AM	9733
4844	04-08-2020 8:00 AM	97362
6034	07-09-2020 8:00 AM	266283

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

Out[29]:
```
258
```

In [30]:
```
teju = teju.groupby('Date')['Confirmed'].sum().reset_index()
```

In [31]:
```
teju = teju.set_index('Date')
teju.index = pd.to_datetime(teju.index)
teju.index
```

Out[31]:
```
DatetimeIndex(['2020-01-04 19:30:00', '2020-01-05 17:00:00',
               '2020-01-06 08:00:00', '2020-01-07 08:00:00',
               '2020-01-08 08:00:00', '2020-01-09 08:00:00',
               '2020-01-10 08:00:00', '2020-01-11 08:00:00',
               '2020-02-04 18:00:00', '2020-02-05 17:00:00',
               ...
               '2020-06-30 08:00:00', '2020-07-30 08:00:00',
               '2020-08-30 08:00:00', '2020-09-30 08:00:00',
               '2020-10-30 08:00:00', '2020-03-31 20:30:00',
               '2020-05-31 08:00:00', '2020-07-31 08:00:00',
               '2020-08-31 08:00:00', '2020-10-31 08:00:00'],
              dtype='datetime64[ns]', name='Date', length=258, freq=None)
```

In [32]:
```
y = teju['Confirmed'].resample('W').mean()
```

```
In [33]: y.index
```

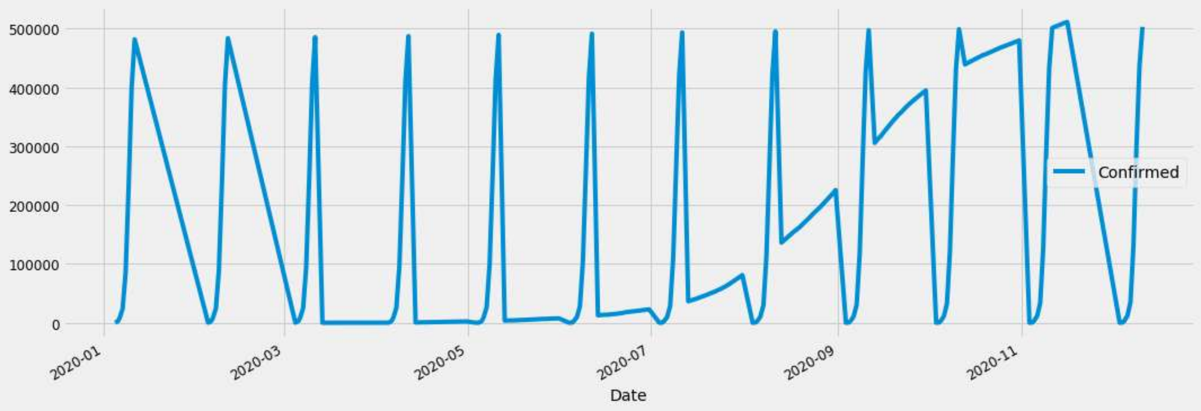
```
Out[33]: DatetimeIndex(['2020-01-05', '2020-01-12', '2020-01-19', '2020-01-26',
                        '2020-02-02', '2020-02-09', '2020-02-16', '2020-02-23',
                        '2020-03-01', '2020-03-08', '2020-03-15', '2020-03-22',
                        '2020-03-29', '2020-04-05', '2020-04-12', '2020-04-19',
                        '2020-04-26', '2020-05-03', '2020-05-10', '2020-05-17',
                        '2020-05-24', '2020-05-31', '2020-06-07', '2020-06-14',
                        '2020-06-21', '2020-06-28', '2020-07-05', '2020-07-12',
                        '2020-07-19', '2020-07-26', '2020-08-02', '2020-08-09',
                        '2020-08-16', '2020-08-23', '2020-08-30', '2020-09-06',
                        '2020-09-13', '2020-09-20', '2020-09-27', '2020-10-04',
                        '2020-10-11', '2020-10-18', '2020-10-25', '2020-11-01',
                        '2020-11-08', '2020-11-15', '2020-11-22', '2020-11-29',
                        '2020-12-06', '2020-12-13'],
                       dtype='datetime64[ns]', name='Date', freq='W-SUN')
```

```
In [34]: y1=kp
```

In [35]:
```
y['2020':]
```

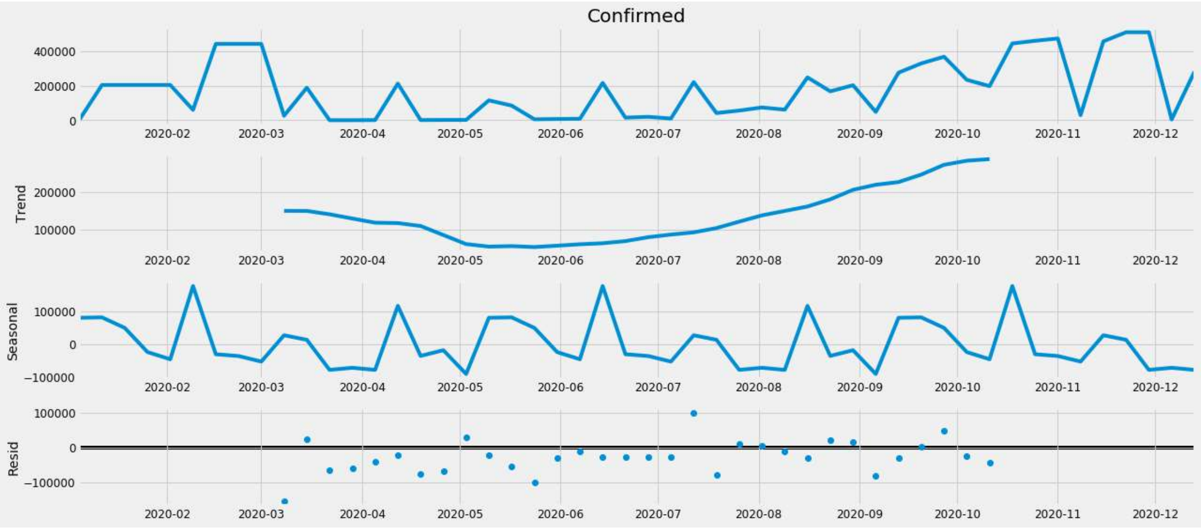
Out[35]:
```
Date
2020-01-05      1192.000000
2020-01-12    204689.166667
2020-01-19              NaN
2020-01-26              NaN
2020-02-02              NaN
2020-02-09     59896.166667
2020-02-16    443466.500000
2020-02-23              NaN
2020-03-01              NaN
2020-03-08     25781.000000
2020-03-15    189013.166667
2020-03-22        19.571429
2020-03-29        43.285714
2020-04-05       621.000000
2020-04-12    212825.000000
2020-04-19       792.428571
2020-04-26      1516.571429
2020-05-03      1664.600000
2020-05-10    115261.142857
2020-05-17     84852.000000
2020-05-24      5176.000000
2020-05-31      6891.142857
2020-06-07      8126.400000
2020-06-14    216446.333333
2020-06-21     14782.714286
2020-06-28     19598.285714
2020-07-05      9657.000000
2020-07-12    221348.333333
2020-07-19     41621.857143
2020-07-26     55985.714286
2020-08-02     73961.000000
2020-08-09     61307.714286
2020-08-16    248684.666667
2020-08-23    167800.571429
2020-08-30    203182.714286
2020-09-06     48042.000000
2020-09-13    276420.000000
2020-09-20    330161.571429
2020-09-27    369006.000000
2020-10-04    234650.800000
2020-10-11    197797.857143
2020-10-18    445994.666667
2020-10-25    461542.285714
2020-11-01    475068.666667
2020-11-08     29323.500000
2020-11-15    458330.500000
2020-11-22    511304.000000
2020-11-29              NaN
2020-12-06      4031.000000
2020-12-13    281207.600000
Freq: W-SUN, Name: Confirmed, dtype: float64
```

In [36]:
```
teju.plot(figsize=(16, 6))
plt.show()
```



In [37]:
```
y.fillna(method='ffill',inplace=True) #Handling the missing value
```

In [38]:
```
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y,freq=18,model='additive')
fig = decomposition.plot()
plt.show()  #x must have 2 complete cycles requires 104 observations. x only has 50 observation(s):-freq=18
```



In [39]:

```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q
))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [40]:

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

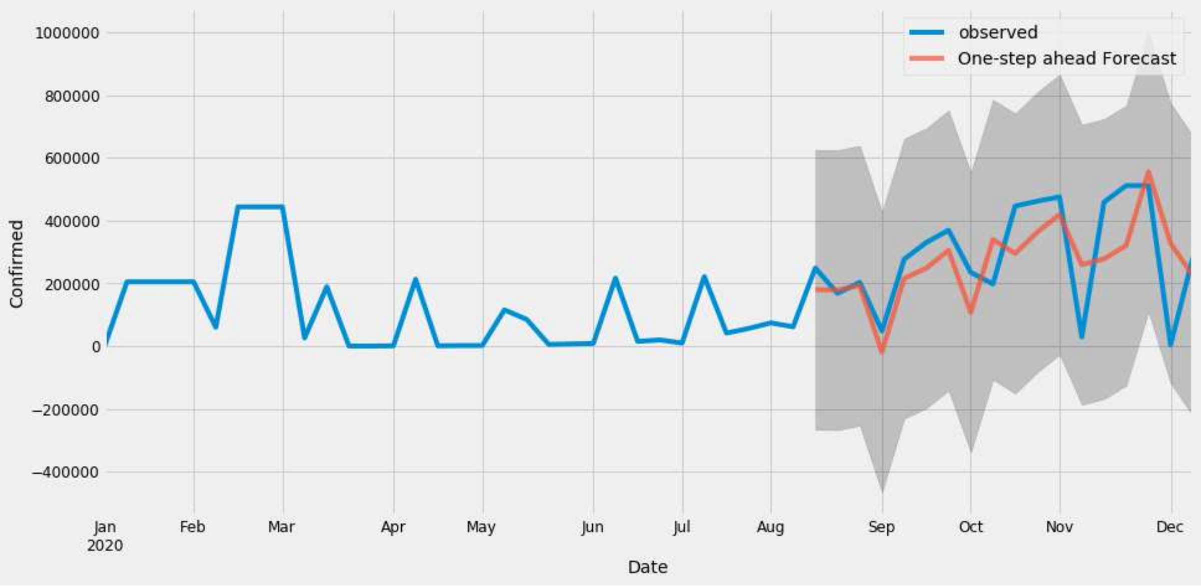
```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1357.6920771396594
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1034.4320605769403
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1026.7100341663531

c:\users\admin\appdata\local\programs\python\python38\lib\site-packages\stats
models\base\model.py:566: ConvergenceWarning: Maximum Likelihood optimization
failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1224.4538038092944
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1047.2301499262799
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1018.3698871413874
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:720.490146487559
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:736.48592943407
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1316.953412854893
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1027.2702326375672
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:996.2642954729932
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:2315.115101024365
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1043.4289313355262
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1027.4896958047302
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:717.9515880929886
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:665.2223608801502
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1301.307314590491
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:983.2932702904785
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1008.076875989129
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:2438.0731814559103
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:998.9837650938674
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:973.9590751092364
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:688.1014979371373
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:659.6844976611611
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1260.3134290651926
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:968.0100605715128
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:966.2801185897981
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:2332.7379662560293
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:986.7262548309574
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:935.6948384633358
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:686.8566262260093
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:627.5425054753762
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:1323.8674496171345
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:999.8426698535941
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:1021.4873980485692
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:2408.532982139236
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:998.698070097653
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:995.6320466853106
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:686.8102486079507
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:682.8752933354283
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:1288.9821896156836
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:962.7328311019434
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:992.839834277659
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:2314.626410825228
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:988.50202846937
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1006.4253000617433
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:687.5246295632178
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:653.5154104158365
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:1294.3404300518823
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:971.0981696122499
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:998.0997052745639
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:2433.701510499066
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:970.2202743014093
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:970.1141758626903
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:661.8233601525218
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:656.3288095177495
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:1262.0882382467023
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:968.6679003872614
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:968.2552641353567
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:2334.7292507765924
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:962.4004502493597
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:937.3017134812808
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:661.5453578881024
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:629.518223887256
```

In [41]:

```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(1, 0, 1),
                                seasonal_order=(1, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
```

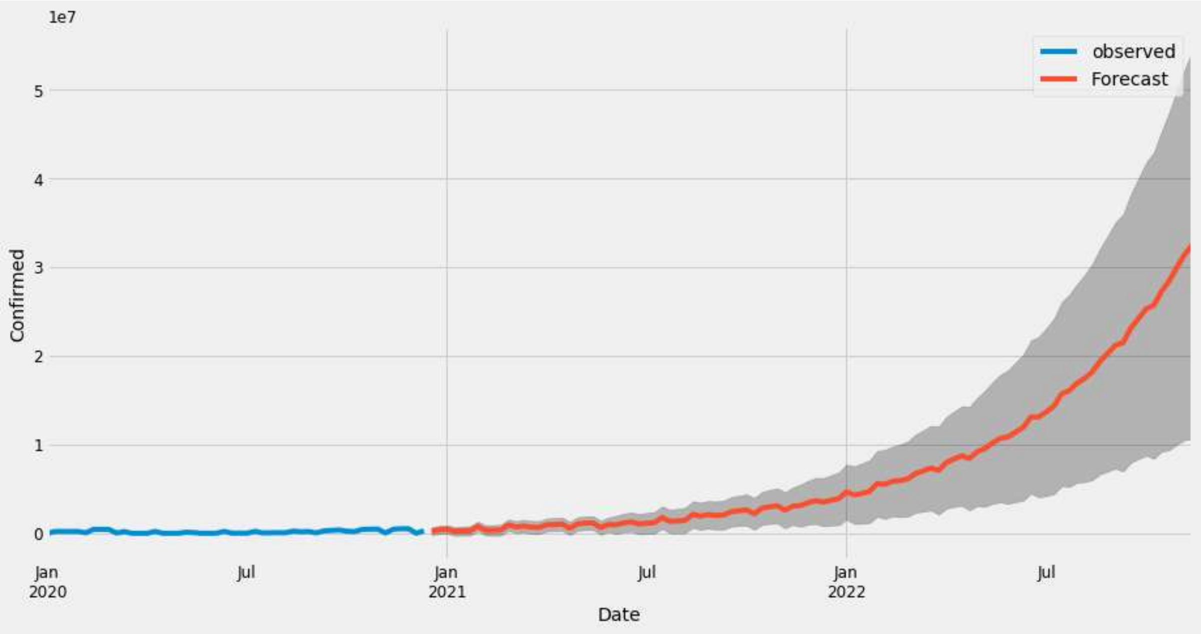
In [42]:

```
pred = results.get_prediction(start=pd.to_datetime('2020-08-16'), dynamic=False)
pred_ci = pred.conf_int()
ax = y['2020':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Confirmed')
plt.legend()
plt.show()
```



In [43]:

```
import matplotlib.pyplot as plt
pred_uc = results.get_forecast(steps=100)
pred_ci = pred_uc.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Confirmed')
plt.legend()
plt.show()
```



THANK YOU