COVID-19 Data Analysis - A Machine Learning Case Study

Team

1. Aradhana J. (17308)
2. Arnab Datta (17309)
3. Ashwin Bala (17310)
4. Niveth Saran V J (17337)

Objective

1. Predict the number of COVID cases using TimeSeries Dataset
2. Use 2 different algorithms for prediction to get more insights.

Contents of this notebook

Under every individual roll number, we have

1. Preprocessing and EDA
2. Model training and prediction
3. Testing and Comparison

After inidvidual analysis, we also provide comparison and conclusion of our analysis.

Importing Required Libraries

In [ ]:

```python
## General Libraries
import numpy as np # linear algebra
import warnings
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
import datetime
from tqdm import tqdm
import matplotlib
import seaborn as sns
import statsmodels.api as sm
from math import sqrt

## Preprocessing
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.stattools import adfuller
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

## Keras Models
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')



matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:1
9: FutureWarning: pandas.util.testing is deprecated. Use the functions
in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

CB.EN.U4CSE17308

PART-1: Exploratory Data Analysis

In [ ]:
```
files = [
        'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_confirmed.csv',
        'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_confirmed_US.csv',
        'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_deaths.csv',
        'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_deaths_US.csv',
        'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_recovered.csv',
]
```

In [ ]:
```
df = pd.read_csv(files[0])
confirmed = df.melt(['Province/State', 'Country/Region'], df.columns[4:], var_name='Dates', value_name='Count')
confirmed.Dates = pd.to_datetime(confirmed.Dates)
confirmed.head()
```

Out[ ]:

	Province/State	Country/Region	Dates	Count
0	NaN	Afghanistan	2020-01-22	0
1	NaN	Albania	2020-01-22	0
2	NaN	Algeria	2020-01-22	0
3	NaN	Andorra	2020-01-22	0
4	NaN	Angola	2020-01-22	0

In [ ]:
```
df = pd.read_csv(files[1])
confirmed_US = df.melt(['Province_State', 'Country_Region'], df.columns[11:], var_name='Dates', value_name='Count')
confirmed_US.rename(columns={"Province_State": "Province/State", "Country_Region": "Country/Region"}, inplace=True)
confirmed_US.Dates = pd.to_datetime(confirmed_US.Dates)
confirmed_US.head()
```

Out[ ]:

	Province/State	Country/Region	Dates	Count
0	Alabama	US	2020-01-22	0
1	Alabama	US	2020-01-22	0
2	Alabama	US	2020-01-22	0
3	Alabama	US	2020-01-22	0
4	Alabama	US	2020-01-22	0

In [ ]:
```
df = pd.read_csv(files[2])
deaths = df.melt(['Province/State', 'Country/Region'], df.columns[4:],
var_name='Dates', value_name='Count')
deaths.Dates = pd.to_datetime(deaths.Dates)
deaths.head()
```

Out[ ]:

	Province/State	Country/Region	Dates	Count
0	NaN	Afghanistan	2020-01-22	0
1	NaN	Albania	2020-01-22	0
2	NaN	Algeria	2020-01-22	0
3	NaN	Andorra	2020-01-22	0
4	NaN	Angola	2020-01-22	0

In [ ]:
```
df = pd.read_csv(files[3])
deaths_US = df.melt(['Province_State', 'Country_Region'], df.columns[1
2:], var_name='Dates', value_name='Count')
deaths_US.rename(columns={"Province_State": "Province/State", "Country
_Region": "Country/Region"}, inplace=True)
deaths_US.Dates = pd.to_datetime(deaths_US.Dates)
deaths_US.head()
```

Out[ ]:

	Province/State	Country/Region	Dates	Count
0	Alabama	US	2020-01-22	0
1	Alabama	US	2020-01-22	0
2	Alabama	US	2020-01-22	0
3	Alabama	US	2020-01-22	0
4	Alabama	US	2020-01-22	0

In [ ]:
```
df = pd.read_csv(files[4])
recovered = df.melt(['Province/State', 'Country/Region'], df.columns[4
:], var_name='Dates', value_name='Count')
recovered.Dates = pd.to_datetime(recovered.Dates)
recovered.head()
```

Out[ ]:

	Province/State	Country/Region	Dates	Count
0	NaN	Afghanistan	2020-01-22	0
1	NaN	Albania	2020-01-22	0
2	NaN	Algeria	2020-01-22	0
3	NaN	Andorra	2020-01-22	0
4	NaN	Angola	2020-01-22	0

In [ ]:

```
df1 = confirmed.groupby('Dates').sum().reset_index()
df2 = deaths.groupby('Dates').sum().reset_index()
df3 = recovered.groupby('Dates').sum().reset_index()

cdr = pd.DataFrame({
    'date': df1.Dates,
    'confirmed': df1.Count,
    'deaths': df2.Count,
    'recovered': df3.Count,
})

cdr.head()
```

Out[ ]:

	date	confirmed	deaths	recovered
0	2020-01-22	555	17	28
1	2020-01-23	654	18	30
2	2020-01-24	941	26	36
3	2020-01-25	1434	42	39
4	2020-01-26	2118	56	52

In [ ]:

```
cdr = cdr.melt('date', cdr.columns[1:], var_name='condition', value_na
me='count')
cdr.head()
```

Out[ ]:

	date	condition	count
0	2020-01-22	confirmed	555
1	2020-01-23	confirmed	654
2	2020-01-24	confirmed	941
3	2020-01-25	confirmed	1434
4	2020-01-26	confirmed	2118

In [ ]:
```
countrywise_confirmed = confirmed.groupby(['Country/Region']).sum().reset_index()
countrywise_deaths = deaths.groupby(['Country/Region']).sum().reset_index()
countrywise_recovered = recovered.groupby(['Country/Region']).sum().reset_index()

countrywise_cdr = pd.DataFrame({
    'country': countrywise_confirmed['Country/Region'],
    'confirmed': countrywise_confirmed.Count,
    'deaths': countrywise_deaths.Count,
    'recovered': countrywise_recovered.Count,
})

countrywise_cdr.head()
```

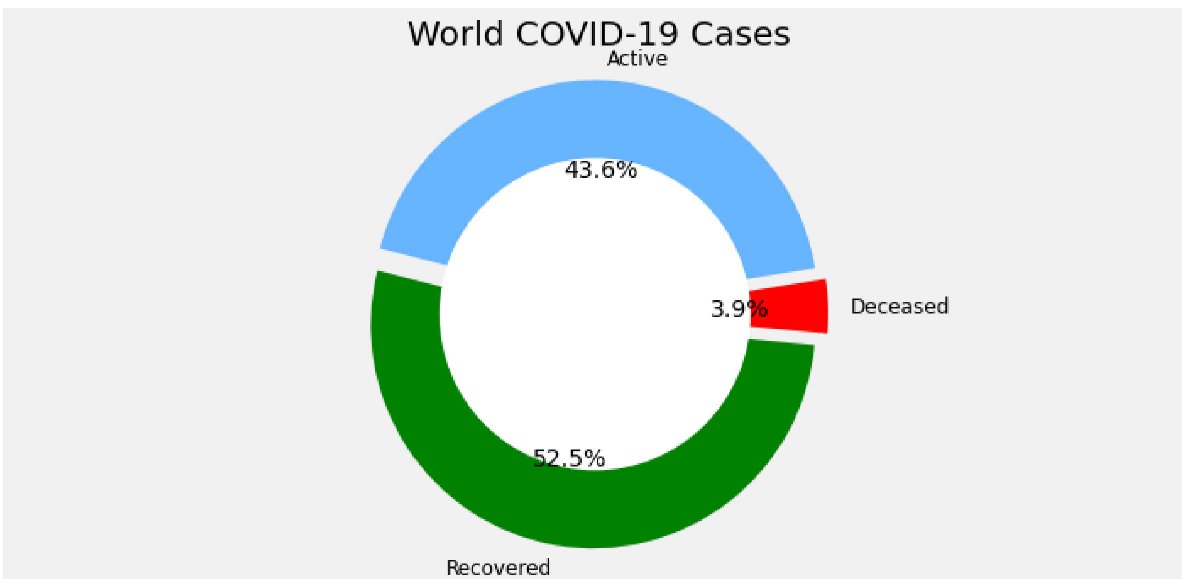
Out[ ]:

	country	confirmed	deaths	recovered
0	Afghanistan	3745342	114595	2139148
1	Albania	583139	17360	329905
2	Algeria	3088876	145505	2089806
3	Andorra	145841	7952	111054
4	Angola	125569	5483	45985

World covid19 cases

Number of active cases can be calculated by subtracting the sum of recovered and death cases from the confirmed cases

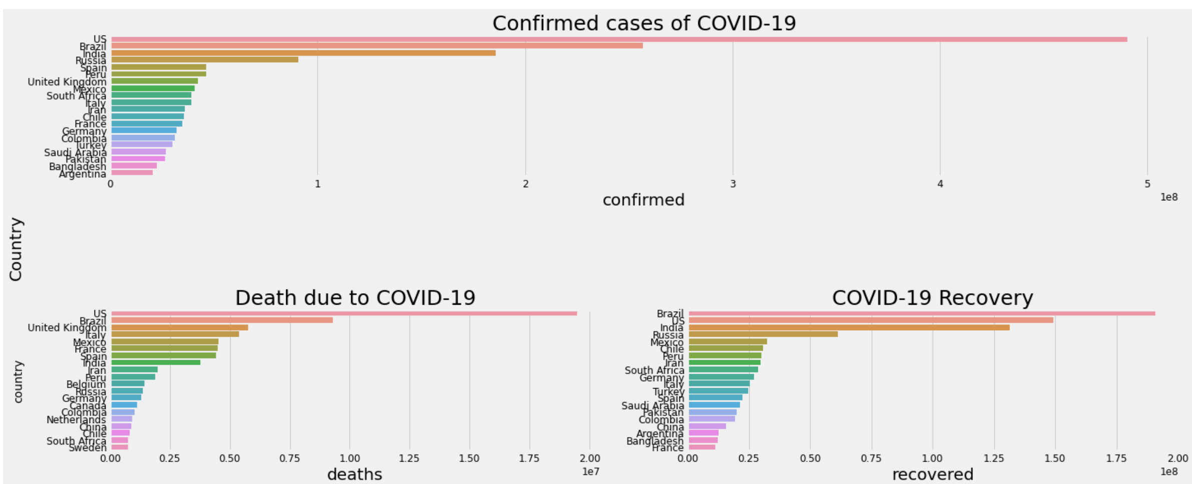
In [ ]:
```
conf=countrywise_cdr['confirmed'].sum()
deth=countrywise_cdr['deaths'].sum()
rec=countrywise_cdr['recovered'].sum()
active=conf-(rec-deth)
labels = ['Active','Recovered','Deceased']
sizes = [active,rec,deth]
color= ['#66b3ff','green','red']
explode = []

for i in labels:
    explode.append(0.05)

plt.figure(figsize= (10,5))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=9, explode
=explode,colors = color)
centre_circle = plt.Circle((0,0),0.70,fc='white')

fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('World COVID-19 Cases',fontsize = 20)
plt.axis('equal')
plt.tight_layout()
```



In [ ]:
```
temp1 = countrywise_cdr.sort_values('confirmed', ascending=False).head
(20)
temp2 = countrywise_cdr.sort_values('deaths', ascending=False).head(20
)
temp3 = countrywise_cdr.sort_values('recovered', ascending=False).head
(20)
```

In [ ]:

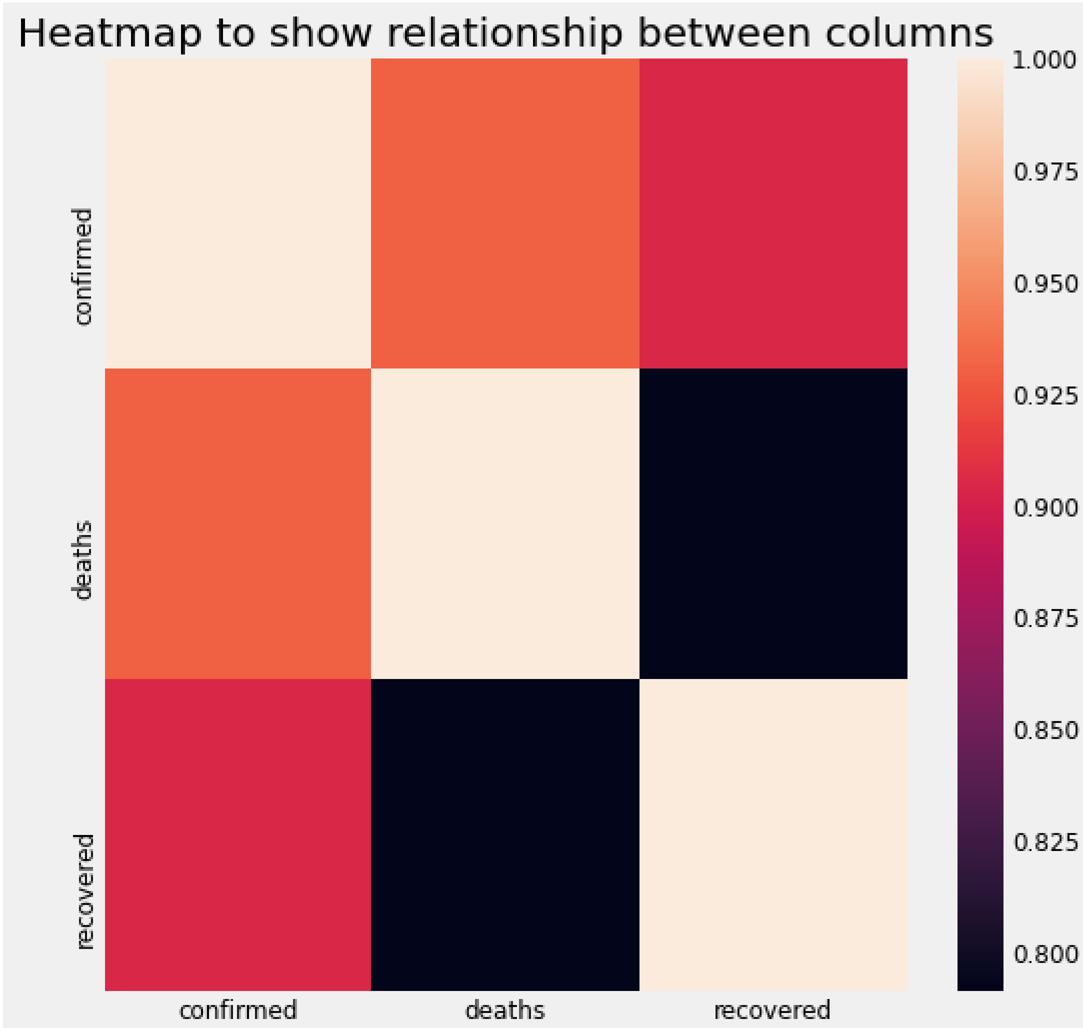
```
fig3 = plt.figure(constrained_layout=True, figsize=(20,8));
gs = fig3.add_gridspec(2, 2);

f3_ax1 = fig3.add_subplot(gs[0, :]);
f3_ax1.set_title('Confirmed cases of COVID-19',fontsize=25);
f3_ax1.set_xlabel("Number of people recovered",fontsize=20);
sns.barplot(temp1["confirmed"], temp1["country"],ax=f3_ax1);

f3_ax2 = fig3.add_subplot(gs[1, :1]);
f3_ax2.set_title('Death due to COVID-19',fontsize=25);
f3_ax2.set_xlabel("Number of people recovered",fontsize=20);
sns.barplot(temp2["deaths"], temp2["country"],ax=f3_ax2);

f3_ax2 = fig3.add_subplot(gs[1, 1:]);
f3_ax2.set_title('COVID-19 Recovery',fontsize=25);
f3_ax2.set_xlabel("Number of people recovered",fontsize=20);
sns.barplot(temp3["recovered"], temp3["country"],ax=f3_ax2);


f3_ax1.set_ylabel("Country",fontsize=20, position=(0,-0.5));
f3_ax2.set_ylabel(" ",fontsize=0);
f3_ax2.set_ylabel(" ",fontsize=0);
```



In [ ]:

Correlation between new cases,deaths and recoveries

In [ ]:
```
plt.figure(figsize=(8,8))
sns.heatmap(countrywise_cdr.corr(), cbar=True)
plt.title('Heatmap to show relationship between columns')
plt.show()
```



Correlation map shows that confirmed cases are positively correlated with the death cases,whereas the a similar correlation is present with the confirmed cases and the recovered cases as well

PART-2: Building the ARIMA model for forecasting the timeseries

Loading the data

In [ ]:
```
files = {
        'confirmed': 'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_confirmed.csv',
        'deaths': 'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_deaths.csv',
        'recovered':'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/time_series_covid_19_recovered.csv'
}
```

In [ ]:
```
confirmed = pd.read_csv(files['confirmed'])
deaths = pd.read_csv(files['deaths'])
recovered= pd.read_csv(files['recovered'])
```

In [ ]:
```
confirmed.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26,
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	

5 rows × 240 columns

In [ ]:
```
deaths.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26,
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	

5 rows × 240 columns

In [ ]:
```
recovered.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26,
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	

5 rows × 240 columns

In [ ]:
```
confirmed_m = confirmed.melt(['Country/Region'], confirmed.columns[4
:], var_name='Dates', value_name='Count')
confirmed_m.rename(columns={"Country/Region": "Country"}, inplace=True
)
confirmed_m.Dates = pd.to_datetime(confirmed_m.Dates)

deaths_m =deaths.melt(['Country/Region'], confirmed.columns[4:], var_n
ame='Dates', value_name='Count')
deaths_m.rename(columns={"Country/Region": "Country"}, inplace=True)
deaths_m.Dates = pd.to_datetime(deaths_m.Dates)

recovered_m = recovered.melt(['Country/Region'], confirmed.columns[4
:], var_name='Dates', value_name='Count')
recovered_m.rename(columns={"Country/Region": "Country"}, inplace=True
)
recovered_m.Dates = pd.to_datetime(recovered_m.Dates)
```

In [ ]:
```
confirmed_m.head()
```

Out[ ]:

	Country	Dates	Count
0	Afghanistan	2020-01-22	0
1	Albania	2020-01-22	0
2	Algeria	2020-01-22	0
3	Andorra	2020-01-22	0
4	Angola	2020-01-22	0

In [ ]:
```
deaths_m.head()
```

Out[ ]:

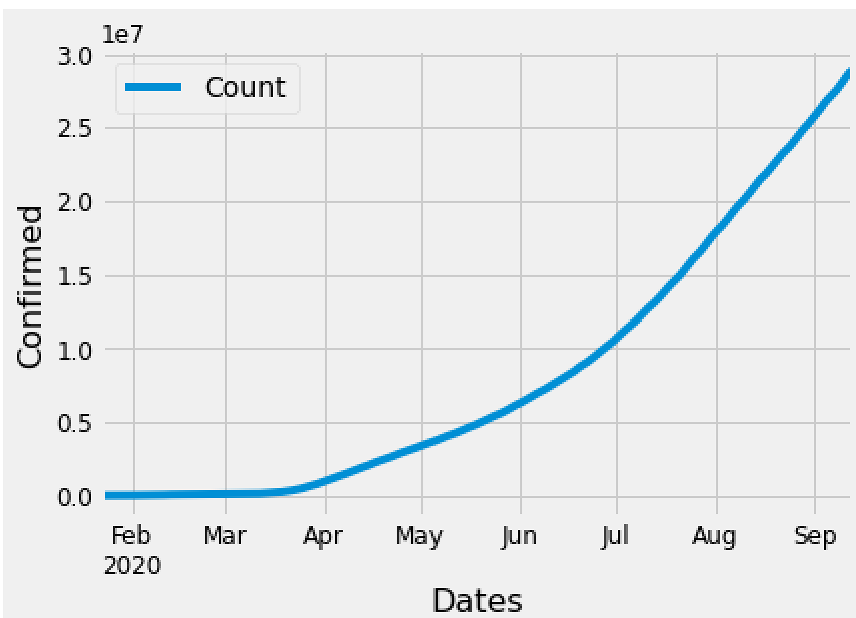
	Country	Dates	Count
0	Afghanistan	2020-01-22	0
1	Albania	2020-01-22	0
2	Algeria	2020-01-22	0
3	Andorra	2020-01-22	0
4	Angola	2020-01-22	0

In [ ]:
```
recovered_m.head()
```

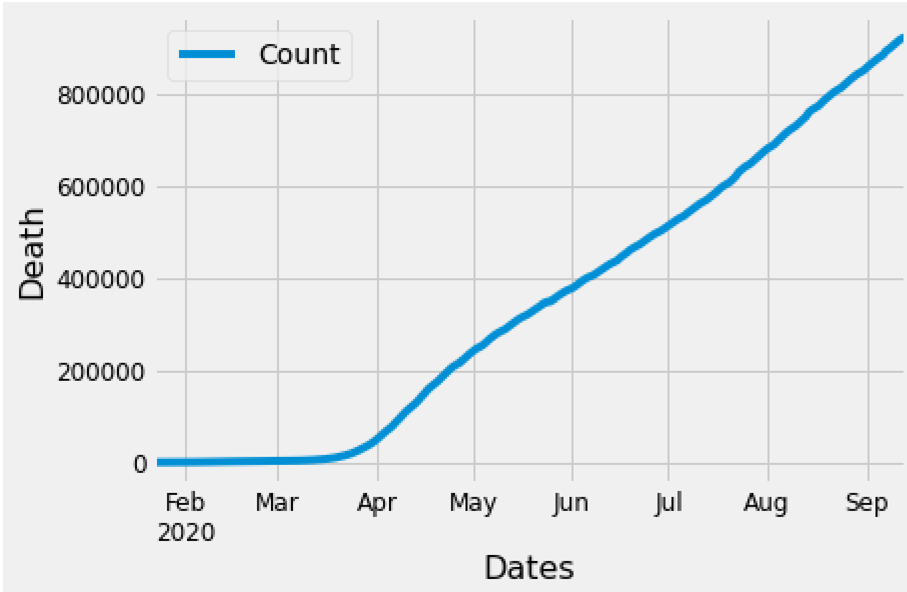
Out[ ]:

	Country	Dates	Count
0	Afghanistan	2020-01-22	0
1	Albania	2020-01-22	0
2	Algeria	2020-01-22	0
3	Andorra	2020-01-22	0
4	Angola	2020-01-22	0

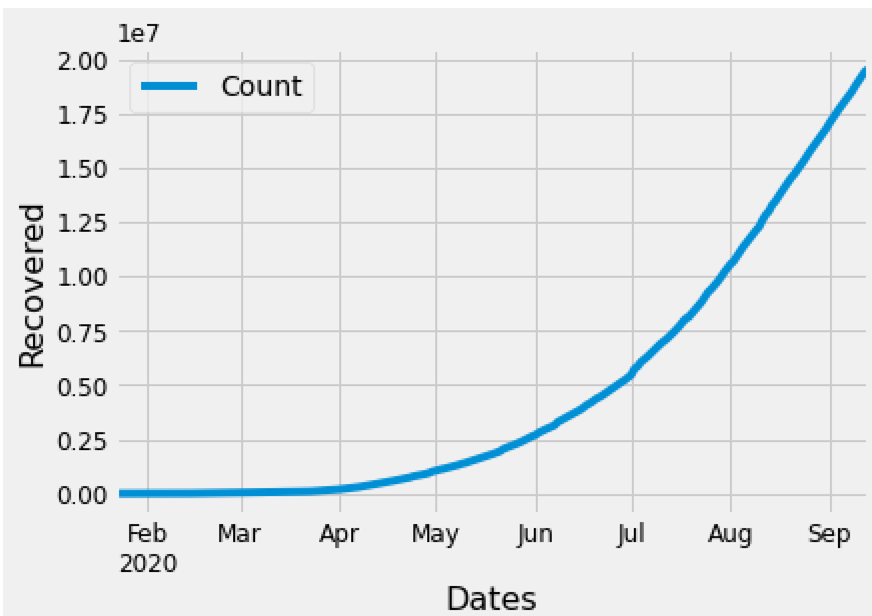
In [ ]:
```
confirmed_m.groupby('Dates').sum().plot();
plt.ylabel('Confirmed', fontsize=16);
plt.xlabel('Dates', fontsize=16);
```



In [ ]:
```
deaths_m.groupby('Dates').sum().plot();
plt.ylabel('Death', fontsize=16);
plt.xlabel('Dates', fontsize=16);
```



In [ ]:
```
recovered_m.groupby('Dates').sum().plot();
plt.ylabel('Recovered', fontsize=16);
plt.xlabel('Dates', fontsize=16);
```



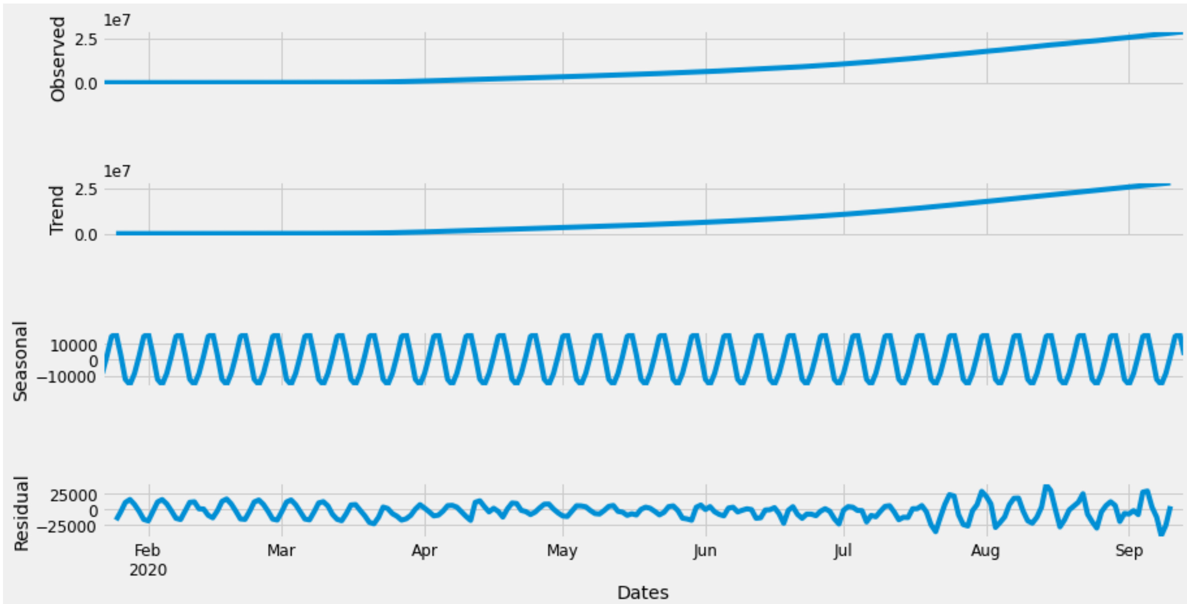
ETS Decomposition

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

Confirmed

In [ ]:
```
temp = confirmed_m.groupby('Dates').sum()[confirmed_m.groupby('Dates')
.sum()['Count']>=0]['Count']
result1 = seasonal_decompose(temp)
```

In [ ]:
```
fig = result1.plot()
fig.set_size_inches(15,8)
```

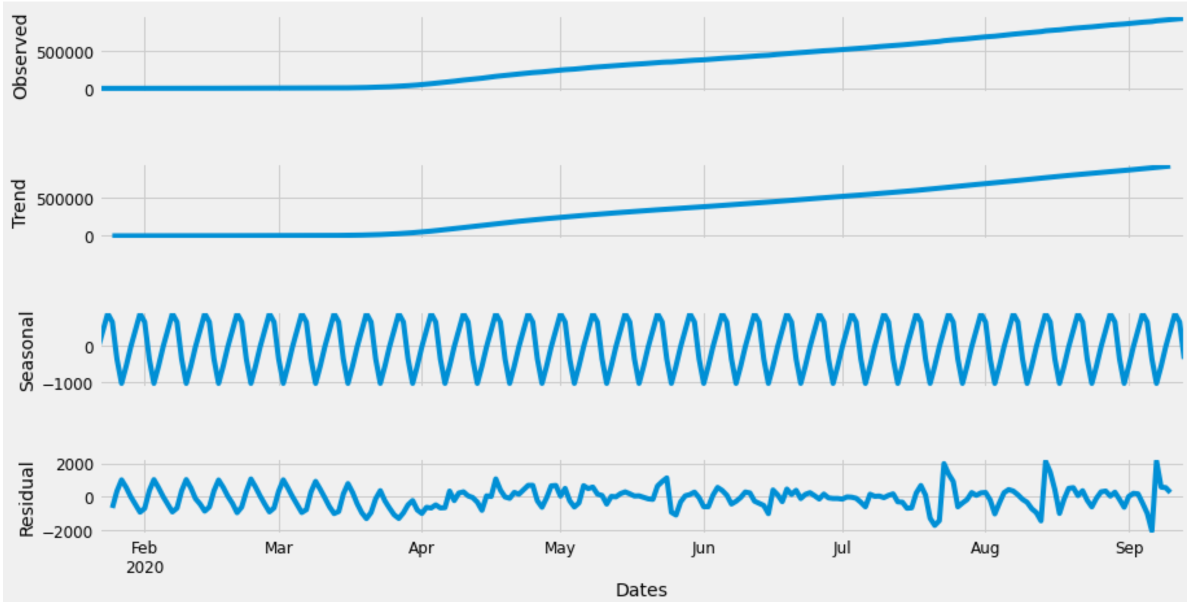


The above plot shows that an increasing trend is present,as well as that the data has seasonal components.It can also be inferred that noise exists in the data from the residual component

Deaths

In [ ]:
```
temp = deaths_m.groupby('Dates').sum()[deaths_m.groupby('Dates').sum()
['Count']>=0]['Count']
result2 = seasonal_decompose(temp)
```

In [ ]:
```
fig = result2.plot()
fig.set_size_inches(15,8)
```

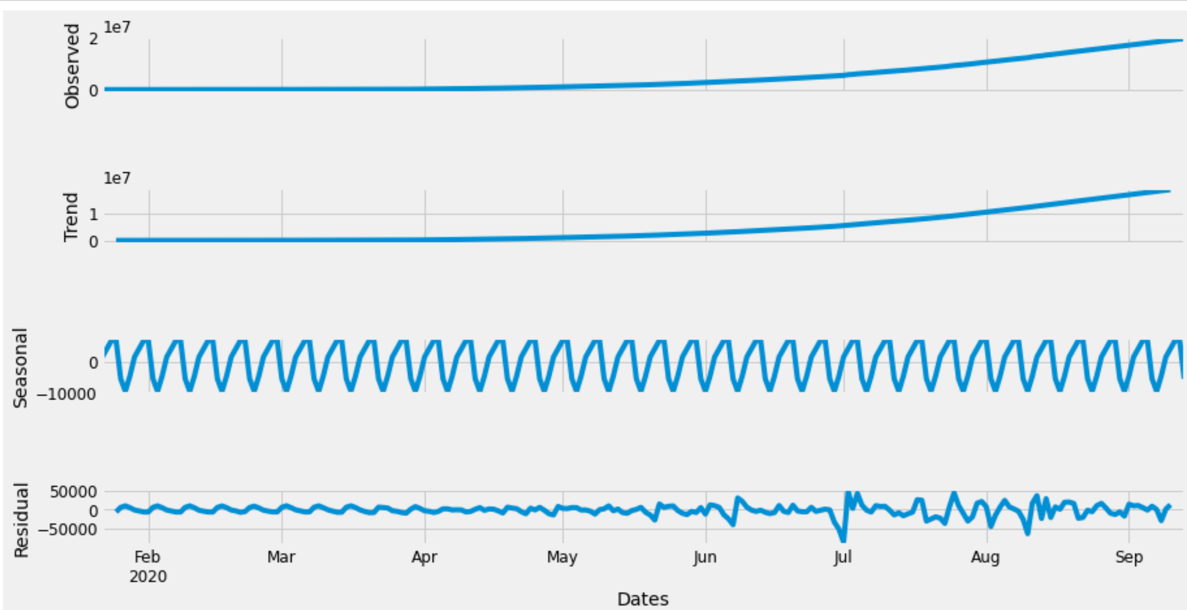


The above plot shows that an increasing trend is present,as well as that the data has seasonal components.It can also be inferred that noise exists in the data from the residual component

Recovered data

In [ ]:
```
temp = recovered_m.groupby('Dates').sum()[recovered_m.groupby('Dates')
.sum()['Count']>=0]['Count']
result3 = seasonal_decompose(temp)
```

In [ ]:
```
fig = result3.plot()
fig.set_size_inches(15,8)
```



The above plot shows that an increasing trend is present,as well as that the data has seasonal components.It can also be inferred that noise exists in the data from the residual component

Stationary test using ADF

The Augmented Dickey Fuller test is performed to check for the staionarity of the data.In time series predictions,having a stationary dataset is important to have correct forecasts.

In [ ]:
```
from statsmodels.tsa.stattools import adfuller
```

In [ ]:
```
confirmed_to_fit = confirmed_m.groupby('Dates').sum()[confirmed_m.grou
pby('Dates').sum()['Count']>=0]
confirmed_to_fit.head()
```

Out[ ]:

	Count
Dates	
2020-01-22	555
2020-01-23	654
2020-01-24	941
2020-01-25	1434
2020-01-26	2118

In [ ]:
```
deaths_to_fit = deaths_m.groupby('Dates').sum()[deaths_m.groupby('Date
s').sum()['Count']>=0]
deaths_to_fit.head()
```

Out[ ]:

	Count
Dates	
2020-01-22	17
2020-01-23	18
2020-01-24	26
2020-01-25	42
2020-01-26	56

In [ ]:
```
recovered_to_fit = recovered_m.groupby('Dates').sum()[recovered_m.grou
pby('Dates').sum()['Count']>=0]
recovered_to_fit.head()
```

Out[ ]:

	Count
Dates	
2020-01-22	28
2020-01-23	30
2020-01-24	36
2020-01-25	39
2020-01-26	52

In [ ]:
```
def adf_check(time_series):

    #Pass in a time series, returns ADF report
    result = adfuller(time_series)

    print('Augmented Dickey-Fuller Test:')

    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of O
bservations Used']

    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis, reject the
null hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has
 a unit root, indicating it is non-stationary ")
```

In [ ]:
```
adf_check(confirmed_to_fit['Count'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -0.908311651116779
p-value : 0.7852229497147818
#Lags Used : 14
Number of Observations Used : 221
weak evidence against null hypothesis, time series has a unit root, in
dicating it is non-stationary
```

In [ ]:
```
adf_check(deaths_to_fit['Count'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 0.03563059096806636
p-value : 0.961381376575122
#Lags Used : 14
Number of Observations Used : 221
weak evidence against null hypothesis, time series has a unit root, in
dicating it is non-stationary
```

```
In [ ]: adf_check(recovered_to_fit['Count'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -2.060819805435561
p-value : 0.2605580586225077
#Lags Used : 8
Number of Observations Used : 227
weak evidence against null hypothesis, time series has a unit root, in
dicating it is non-stationary
```

According to the adf test,data is stationary,therefore differentation is to be done to make data stationary which will be done in the parameter 'd' of the SARIMAX model

Seasonal Arima

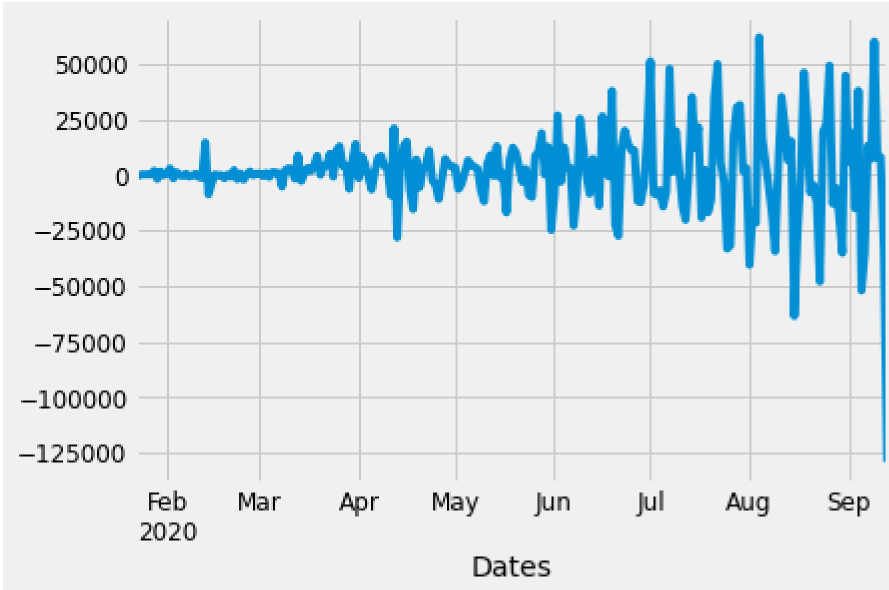
```
In [ ]: from statsmodels.tsa.arima_model import ARIMA
        from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [ ]:

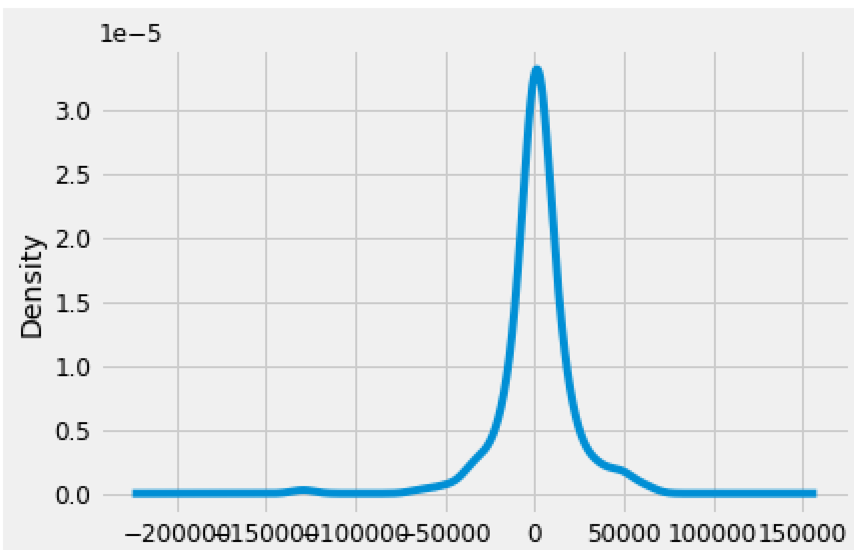
```
confirmed_to_fit = confirmed_m.groupby('Dates').sum()[confirmed_m.grou
pby('Dates').sum()['Count']>=0]

model = SARIMAX(confirmed_to_fit['Count'],order=(1,1,0), seasonal_orde
r=(1,1,1,12))
results = model.fit()
print(results.summary())
```

```
                                 Statespace Model Results
======================================================================
====================
Dep. Variable:                              Count   No. Observations:
236
Model:             SARIMAX(1, 1, 0)x(1, 1, 1, 12)   Log Likelihood
-2537.654
Date:                            Tue, 03 Nov 2020   AIC
5083.308
Time:                                    11:41:25   BIC
5096.937
Sample:                                01-22-2020   HQIC
5088.810
                                     - 09-13-2020
Covariance Type:                              opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.9795      0.024     41.473      0.000       0.933
1.026
ar.S.L12      -0.2433      0.120     -2.032      0.042      -0.478
-0.009
ma.S.L12      -0.8403      0.121     -6.958      0.000      -1.077
-0.604
sigma2      6.321e+08   1.93e-11   3.28e+19      0.000    6.32e+08
6.32e+08
======================================================================
=============
Ljung-Box (Q):                      734.23   Jarque-Bera (JB):
846.82
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):              21.87   Skew:
-1.12
Prob(H) (two-sided):                  0.00   Kurtosis:
12.28
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition num
ber 5.21e+35. Standard errors may be unstable.
```

In [ ]:
```
results.resid.plot();
```

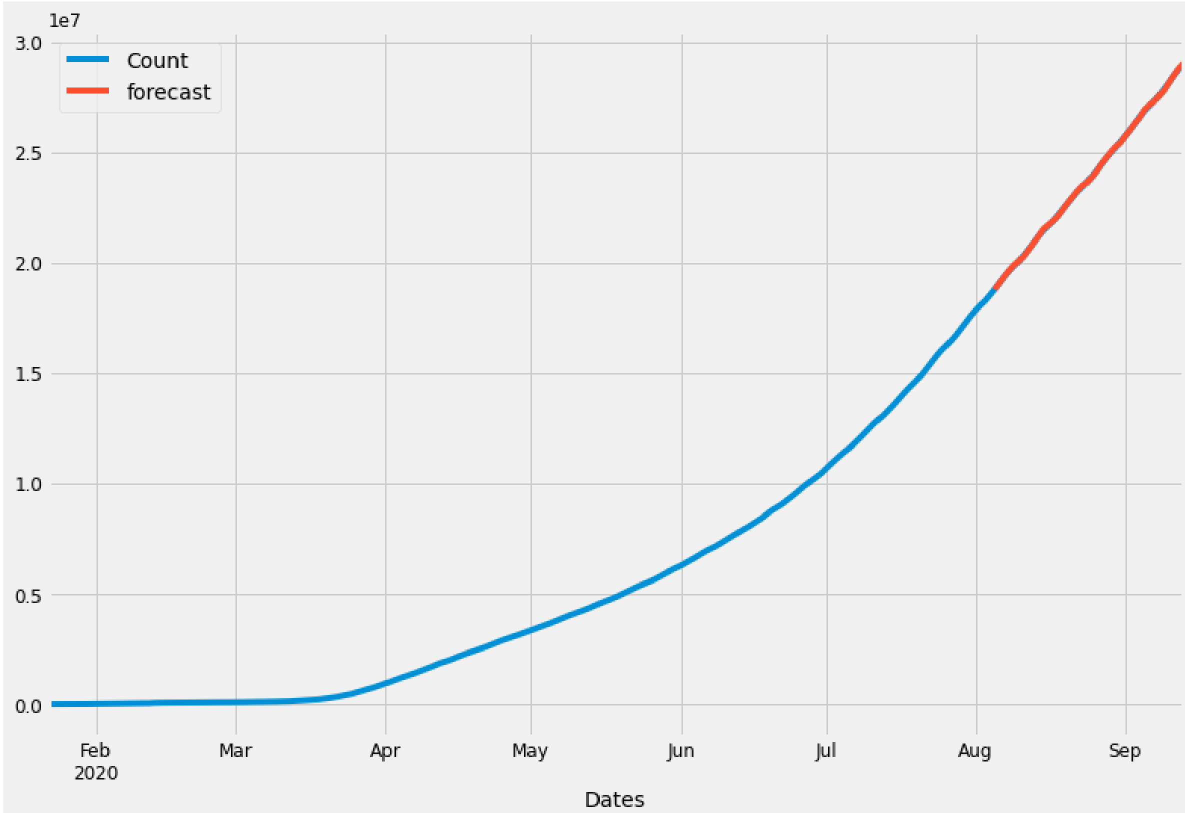


In [ ]:
```
results.resid.plot(kind='kde');
```



The residual plots seem stationary with 0 mean and uniform variance

In [ ]:
```
confirmed_to_fit['forecast'] = results.predict(start = 196, end= 236)
confirmed_to_fit[['Count','forecast']].plot(figsize=(12,8));
```



In [ ]:
```
from pandas.tseries.offsets import DateOffset
```

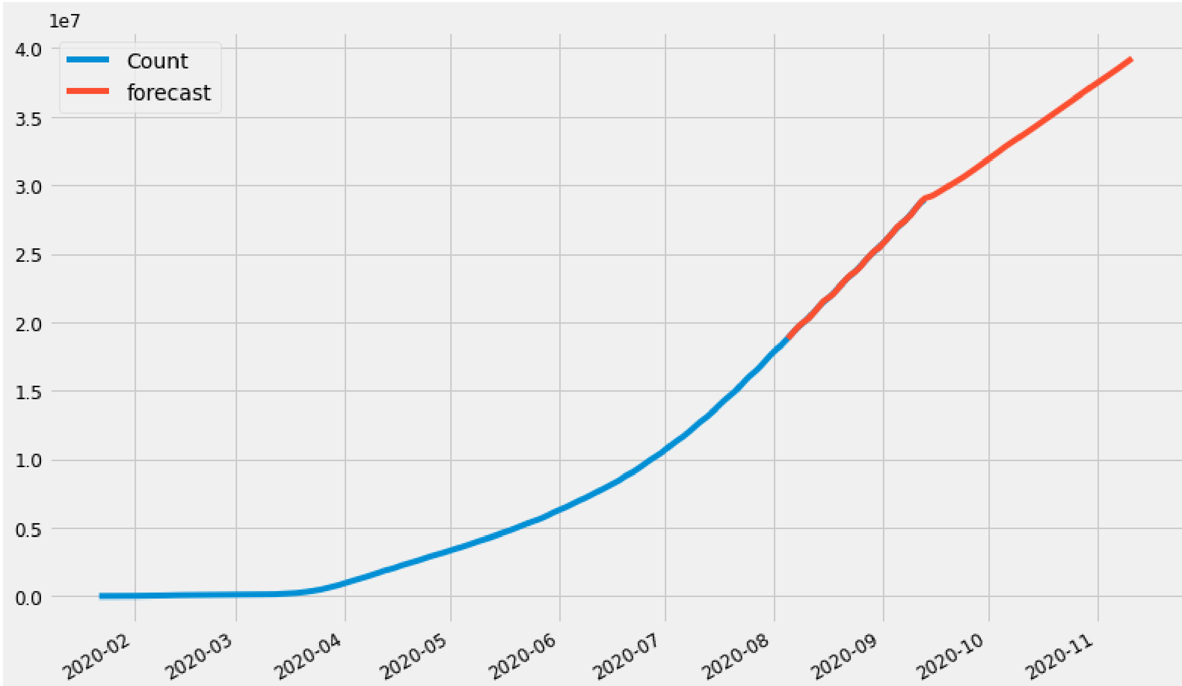
In [ ]:
```
n = 60
last_date = confirmed_to_fit.index[-1]

future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]
```

In [ ]:
```
future_dates_df = pd.DataFrame(index=future_dates[1:],columns=confirme
d_to_fit.columns)
future_df = pd.concat([confirmed_to_fit,future_dates_df])
future_df.tail()
```

Out[ ]:

	Count	forecast
2020-11-07	NaN	NaN
2020-11-08	NaN	NaN
2020-11-09	NaN	NaN
2020-11-10	NaN	NaN
2020-11-11	NaN	NaN

In [ ]:
```
future_df['forecast'] = results.predict(start = 196, end = future_df.index[-1]);
future_df[['Count', 'forecast']].plot(figsize=(12, 8));
```



Accuracy metrics for confirmed cases prediction

MEAN ABSOLUTE PERCENTAGE ERROR

In [ ]:
```
mape = np.mean(np.abs(confirmed_to_fit['forecast']-confirmed_to_fit['Count'])/np.abs(confirmed_to_fit['Count']))
mape*100
```

Out[ ]: 0.10637174121099298

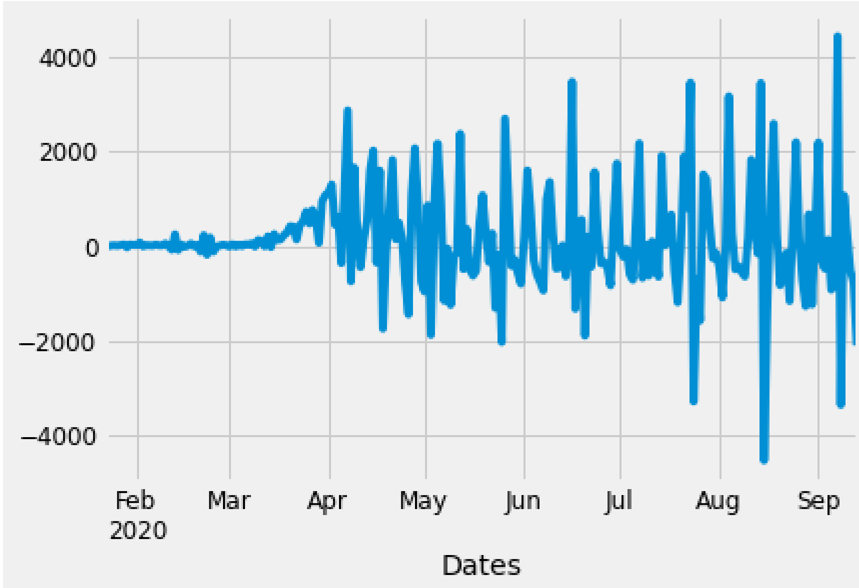
The prediction has an error of .106% indicating that the model is very much accurate with an accuracy of 99.9%

Deaths forecasting

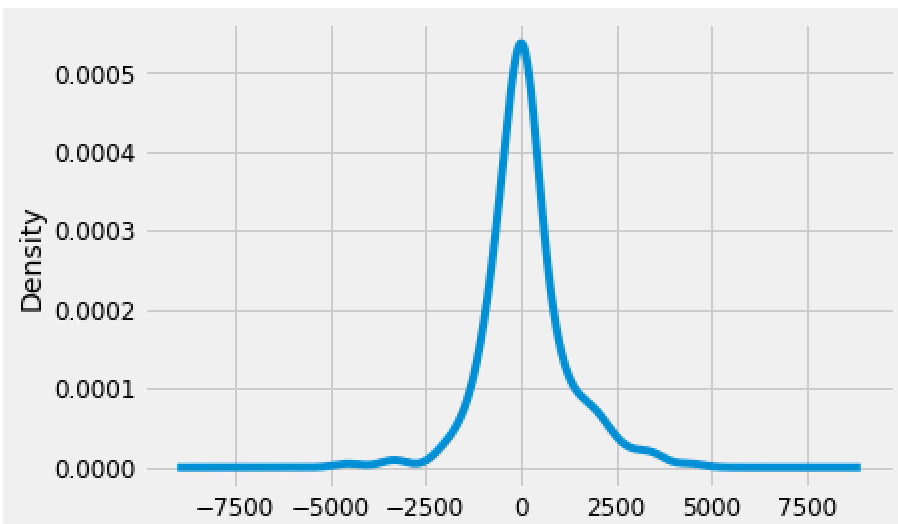
In [ ]:
```
deaths_to_fit = deaths_m.groupby('Dates').sum()[confirmed_m.groupby('D
ates').sum()['Count']>=0]

model = SARIMAX(deaths_to_fit['Count'],order=(1,1,0), seasonal_order=(
1,1,1,12))
results = model.fit()
print(results.summary())
```

```
                                     Statespace Model Results
==========================================================================
====================
Dep. Variable:                                   Count   No. Observations:
236
Model:             SARIMAX(1, 1, 0)x(1, 1, 1, 12)   Log Likelihood
-1888.637
Date:                             Tue, 03 Nov 2020   AIC
3785.273
Time:                                     11:41:28   BIC
3798.902
Sample:                                 01-22-2020   HQIC
3790.775
                                      - 09-13-2020
Covariance Type:                               opg
==========================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
--------
ar.L1          0.9092      0.025     36.889      0.000       0.861
0.958
ar.S.L12      -0.2785      0.071     -3.900      0.000      -0.419
-0.139
ma.S.L12      -0.7908      0.049    -15.979      0.000      -0.888
-0.694
sigma2      1.273e+06   8.46e+04     15.046      0.000    1.11e+06
1.44e+06
==========================================================================
=============
Ljung-Box (Q):                      332.54   Jarque-Bera (JB):
84.35
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               5.27   Skew:
0.32
Prob(H) (two-sided):                  0.00   Kurtosis:
5.95
==========================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

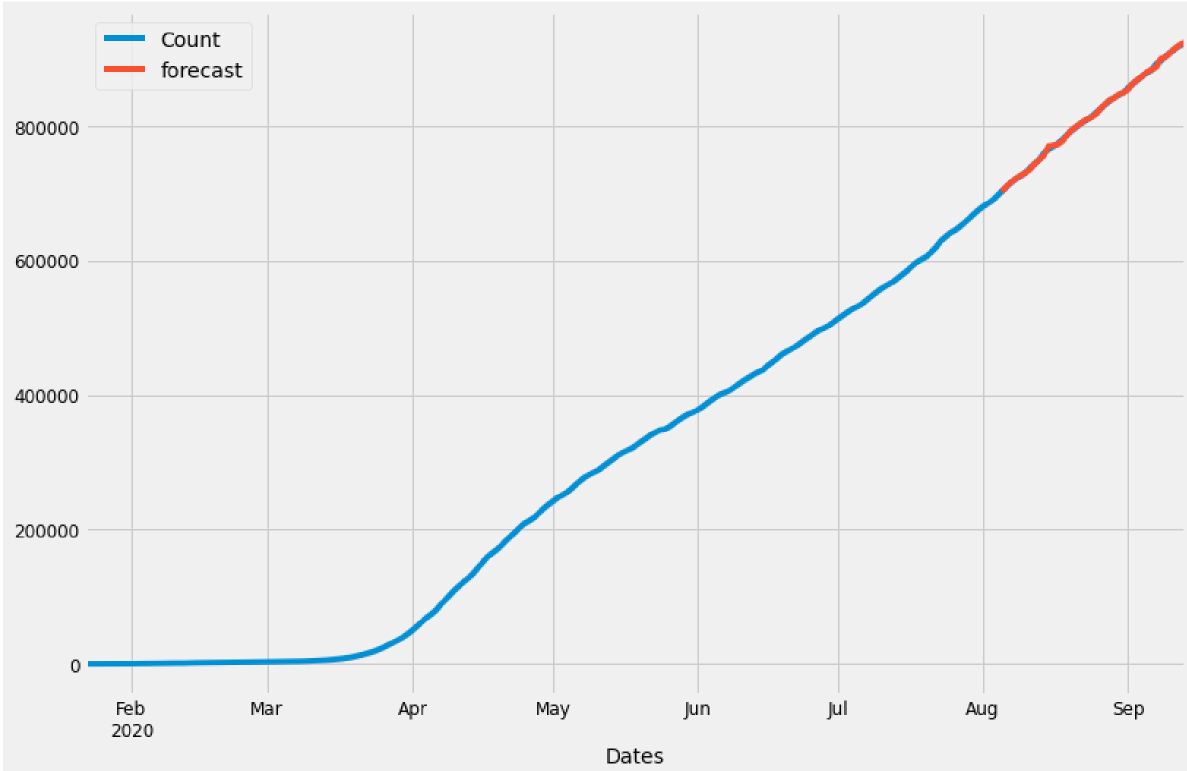
In [ ]:
```
results.resid.plot();
```



In [ ]:
```
results.resid.plot(kind='kde');
```



In [ ]:
```
deaths_to_fit['forecast'] = results.predict(start = 196, end= 236);
deaths_to_fit[['Count','forecast']].plot(figsize=(12,8));
```



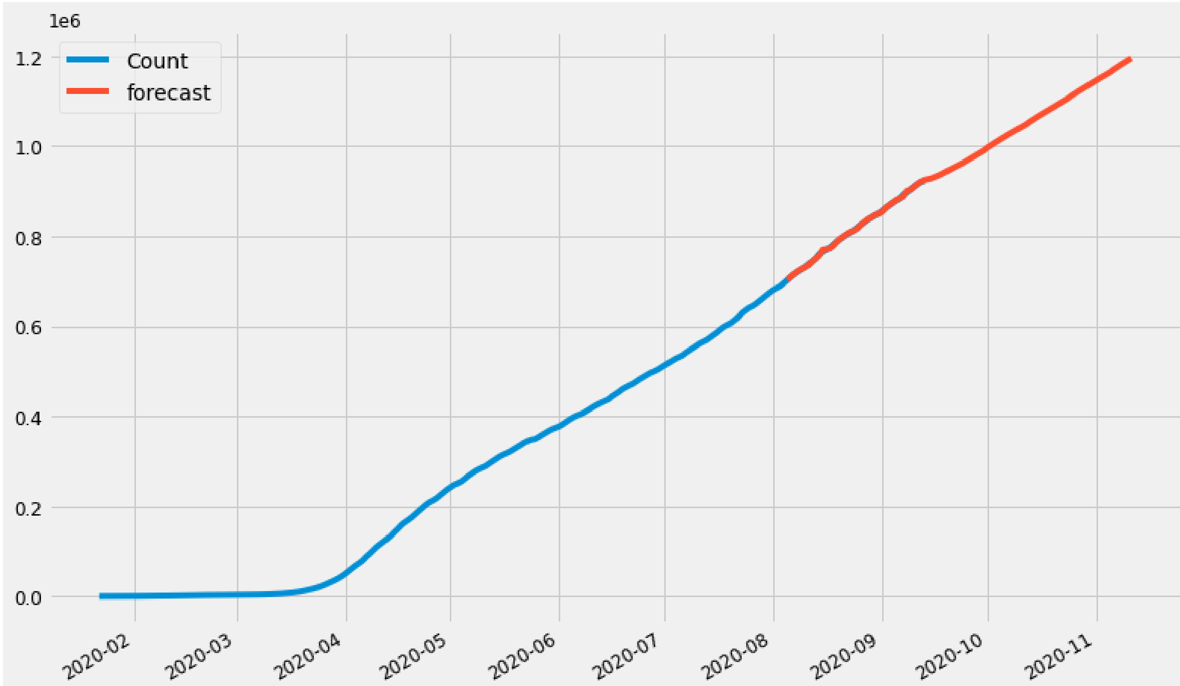
In [ ]:
```
n = 60
last_date = deaths_to_fit.index[-1]

future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]
```

In [ ]:
```
future_dates_df = pd.DataFrame(index=future_dates[1:],columns=deaths_to_fit.columns)
future_df = pd.concat([deaths_to_fit,future_dates_df])
future_df.tail()
```

Out[ ]:

	Count	forecast
2020-11-07	NaN	NaN
2020-11-08	NaN	NaN
2020-11-09	NaN	NaN
2020-11-10	NaN	NaN
2020-11-11	NaN	NaN

In [ ]:
```
future_df['forecast'] = results.predict(start = 196, end = future_df.i
ndex[-1]);
future_df[['Count', 'forecast']].plot(figsize=(12, 8));
```



In [ ]:
```
mape = np.mean(np.abs(deaths_to_fit['forecast']-deaths_to_fit['Count'
])/np.abs(deaths_to_fit['Count']))
mape*100
```

Out[ ]:
```
0.13808435966875504
```

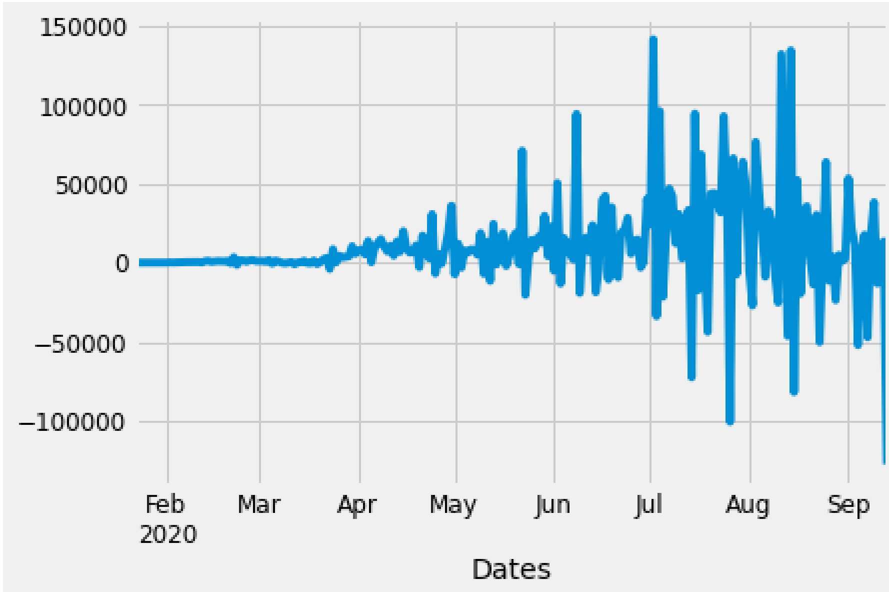
The prediction has an error of .13% indicating that the model is very much accurate with an accuracy of 99.86%

Recovered Cases forecasting

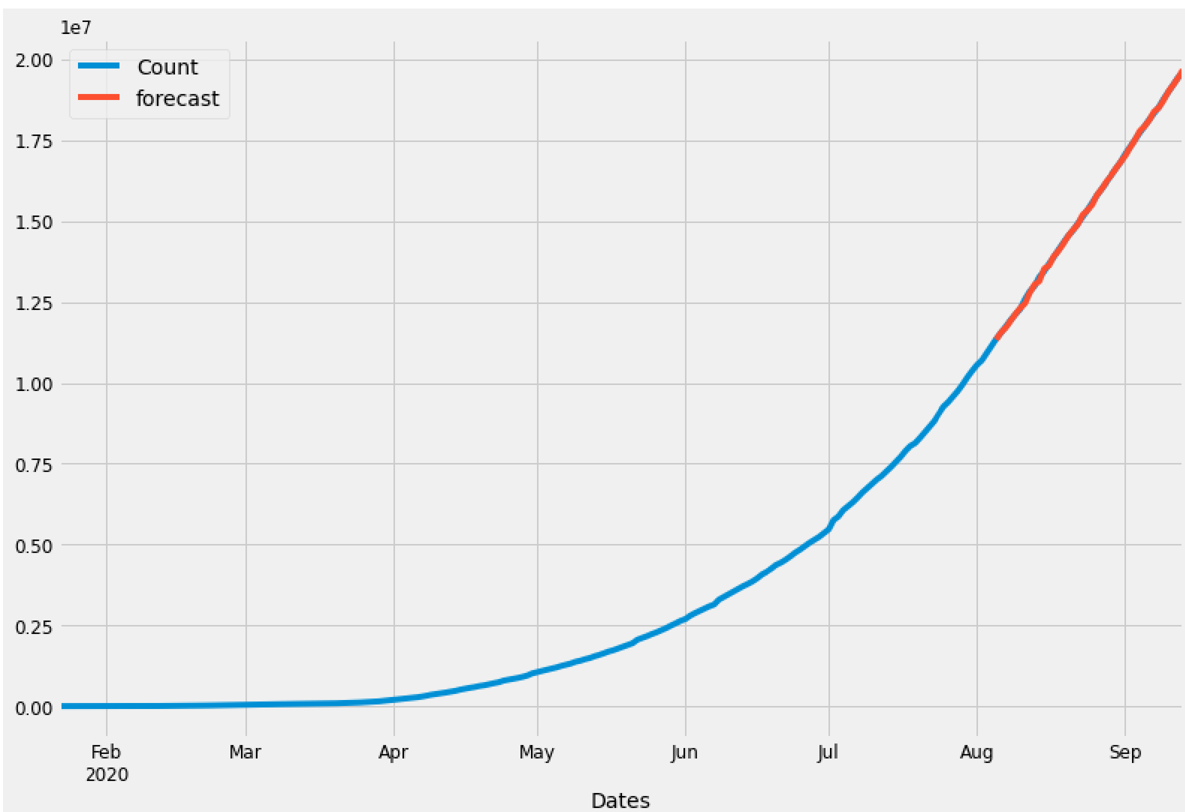
In [ ]:
```
recovered_to_fit = recovered_m.groupby('Dates').sum()[recovered_m.grou
pby('Dates').sum()['Count']>=0]

model = SARIMAX(recovered_to_fit['Count'],order=(1,1,0), seasonal_orde
r=(1,1,1,12))
results = model.fit()
print(results.summary())
```

```
                              Statespace Model Results
======================================================================
====================
Dep. Variable:                               Count   No. Observations:
236
Model:             SARIMAX(1, 1, 0)x(1, 1, 1, 12)   Log Likelihood
-2635.202
Date:                             Tue, 03 Nov 2020   AIC
5278.405
Time:                                     11:41:30   BIC
5292.033
Sample:                                 01-22-2020   HQIC
5283.907
                                      - 09-13-2020
Covariance Type:                               opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.4353      0.063      6.870      0.000       0.311
0.560
ar.S.L12      -0.3240      0.108     -2.987      0.003      -0.537
-0.111
ma.S.L12      -0.2050      0.128     -1.605      0.108      -0.455
0.045
sigma2      1.237e+09   1.31e-10   9.47e+18      0.000    1.24e+09
1.24e+09
======================================================================
=============
Ljung-Box (Q):                      113.27   Jarque-Bera (JB):
307.38
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):              89.51   Skew:
0.55
Prob(H) (two-sided):                  0.00   Kurtosis:
8.65
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition num
ber 1.15e+34. Standard errors may be unstable.
```

In [ ]:
```
results.resid.plot();
```



In [ ]:
```
recovered_to_fit['forecast'] = results.predict(start = 196, end= 236);
recovered_to_fit[['Count','forecast']].plot(figsize=(12,8));
```



In [ ]:
```
from pandas.tseries.offsets import DateOffset
```

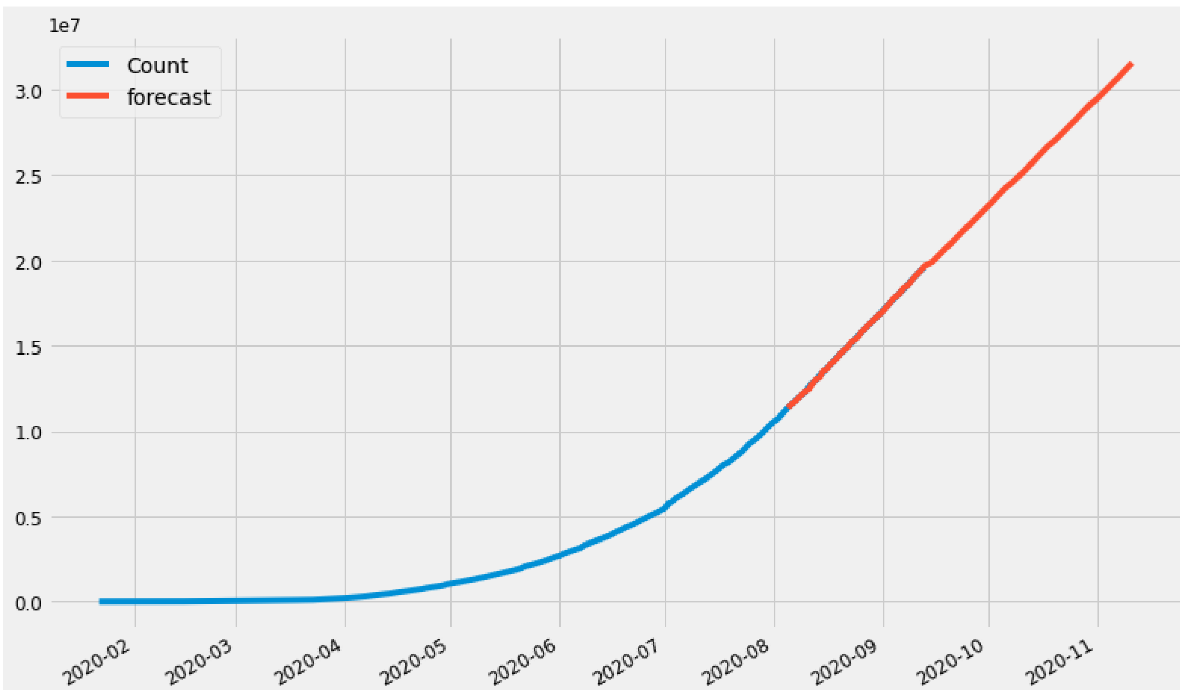
In [ ]:
```
n = 60
last_date = recovered_to_fit.index[-1]

future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]
```

In [ ]:
```
future_dates_df = pd.DataFrame(index=future_dates[1:],columns=recovered_to_fit.columns)
future_df = pd.concat([recovered_to_fit,future_dates_df])
future_df.tail()
```

Out[ ]:

	Count	forecast
2020-11-07	NaN	NaN
2020-11-08	NaN	NaN
2020-11-09	NaN	NaN
2020-11-10	NaN	NaN
2020-11-11	NaN	NaN

In [ ]:
```
future_df['forecast'] = results.predict(start = 196, end = future_df.index[-1]);
future_df[['Count', 'forecast']].plot(figsize=(12, 8));
```



In [ ]:
```
mape = np.mean(np.abs(recovered_to_fit['forecast']-recovered_to_fit['Count'])/np.abs(recovered_to_fit['Count']))
mape*100
```

Out[ ]: 0.22739627662184567

The prediction has an error of .13% indicating that the model is very much accurate with an accuracy of 99.78%

In [ ]:

CB.EN.U4CSE17309

In [ ]:
```
import numpy as np
from scipy import stats
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Loading the Data

In [ ]:
```
files = { 'National_data': 'https://raw.githubusercontent.com/Ashwin19
99/COVID-19-Data-Mining/master/COVID-DataTimeSeries(India)/nation_leve
l_daily.csv'}
```

In [ ]:
```
National_data = pd.read_csv(files['National_data'])
National_data.head()
```

Out[ ]:

	Date	Daily Confirmed	Total Confirmed	Daily Recovered	Total Recovered	Daily Deceased	Total Deceased
0	30 January	1	1	0	0	0	0
1	31 January	0	1	0	0	0	0
2	01 February	0	1	0	0	0	0
3	02 February	1	2	0	0	0	0
4	03 February	1	3	0	0	0	0

In [ ]:
```
National_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 190 entries, 0 to 189
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Date             190 non-null    object
 1   Daily Confirmed  190 non-null    int64 
 2   Total Confirmed  190 non-null    int64 
 3   Daily Recovered  190 non-null    int64 
 4   Total Recovered  190 non-null    int64 
 5   Daily Deceased   190 non-null    int64 
 6   Total Deceased   190 non-null    int64 
dtypes: int64(6), object(1)
memory usage: 10.5+ KB
```

1. Data organisation

In [ ]:
```
National_data['Date'] = National_data['Date'].apply(lambda x: x+' 202
0')
```

In [ ]:
```
National_data.head(100)
```

Out[ ]:

	Date	Daily Confirmed	Total Confirmed	Daily Recovered	Total Recovered	Daily Deceased	Total Deceased
0	30 January 2020	1	1	0	0	0	0
1	31 January 2020	0	1	0	0	0	0
2	01 February 2020	0	1	0	0	0	0
3	02 February 2020	1	2	0	0	0	0
4	03 February 2020	1	3	0	0	0	0
...	...	...	...	...	...	...	...
95	04 May 2020	3656	46434	1082	12845	103	1566
96	05 May 2020	2971	49405	1295	14140	128	1694
97	06 May 2020	3602	53007	1161	15301	91	1785
98	07 May 2020	3344	56351	1475	16776	104	1889
99	08 May 2020	3339	59690	1111	17887	97	1986

100 rows × 7 columns

In [ ]:
```
National_data['Date']=pd.to_datetime(National_data['Date'])
National_data.head()
```

Out[ ]:

	Date	Daily Confirmed	Total Confirmed	Daily Recovered	Total Recovered	Daily Deceased	Total Deceased
0	2020-01-30	1	1	0	0	0	0
1	2020-01-31	0	1	0	0	0	0
2	2020-02-01	0	1	0	0	0	0
3	2020-02-02	1	2	0	0	0	0
4	2020-02-03	1	3	0	0	0	0

In [ ]:
```
Confirmed_India= National_data[['Date', 'Daily Confirmed', 'Total Conf
irmed']]
Confirmed_India.head()
```

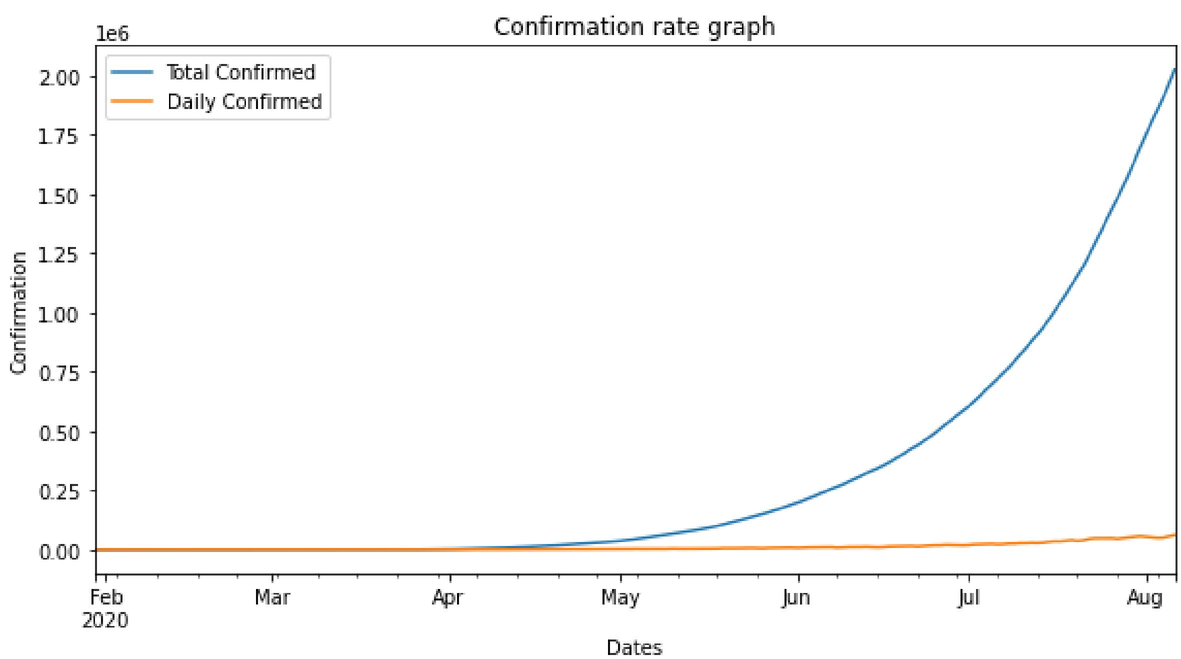
Out[ ]:

	Date	Daily Confirmed	Total Confirmed
0	2020-01-30	1	1
1	2020-01-31	0	1
2	2020-02-01	0	1
3	2020-02-02	1	2
4	2020-02-03	1	3

3. Data Visualisation

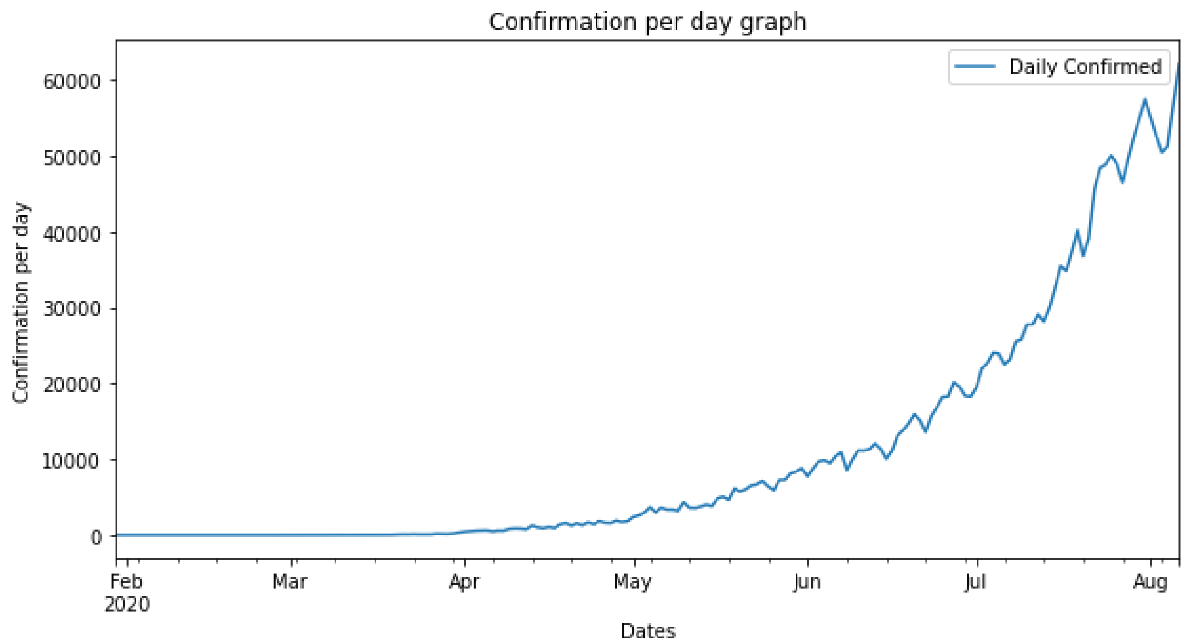
In [ ]:
```
Confirmed_India.plot(x='Date', y=['Total Confirmed','Daily Confirmed'
], figsize=[10,5])
plt.title('Confirmation rate graph')
plt.ylabel('Confirmation', fontsize=10);
plt.xlabel('Dates', fontsize=10)
```

Out[ ]: Text(0.5, 0, 'Dates')



In [ ]:
```
Confirmed_India.plot(x='Date', y=['Daily Confirmed'], figsize=[10,5])
plt.title('Confirmation per day graph')
plt.ylabel('Confirmation per day', fontsize=10);
plt.xlabel('Dates', fontsize=10)
```

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



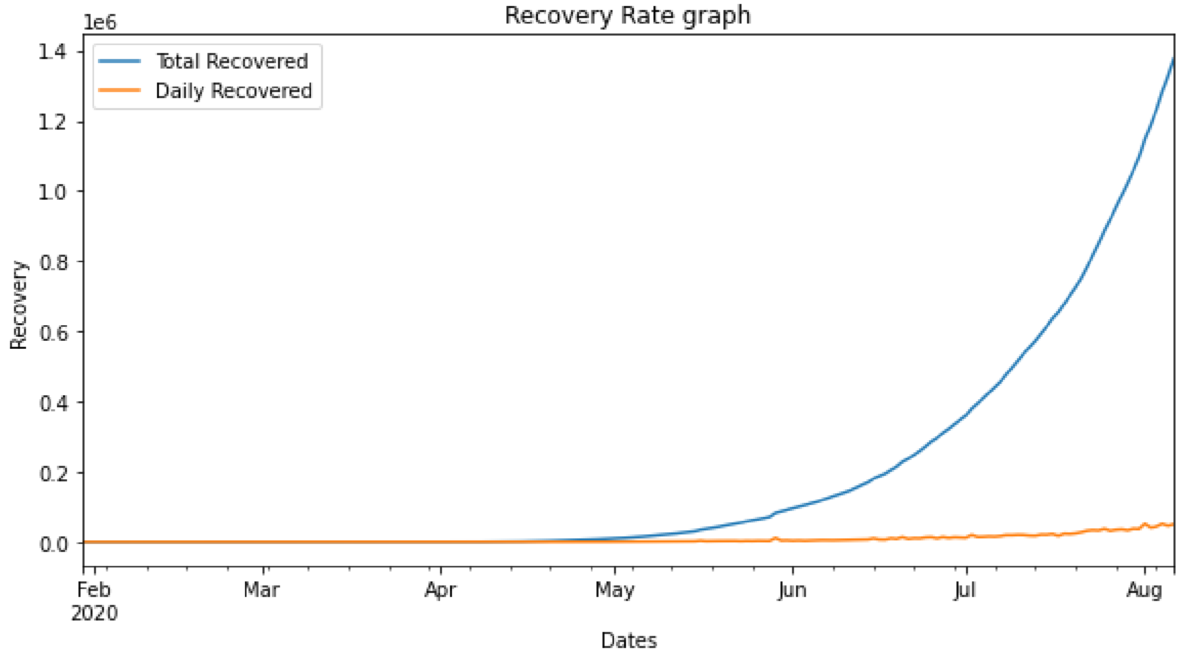
In [ ]:
```
Recovered_India= National_data[['Date', 'Daily Recovered', 'Total Reco
vered']]
Recovered_India.head()
```

Out[ ]:

	Date	Daily Recovered	Total Recovered
0	2020-01-30	0	0
1	2020-01-31	0	0
2	2020-02-01	0	0
3	2020-02-02	0	0
4	2020-02-03	0	0

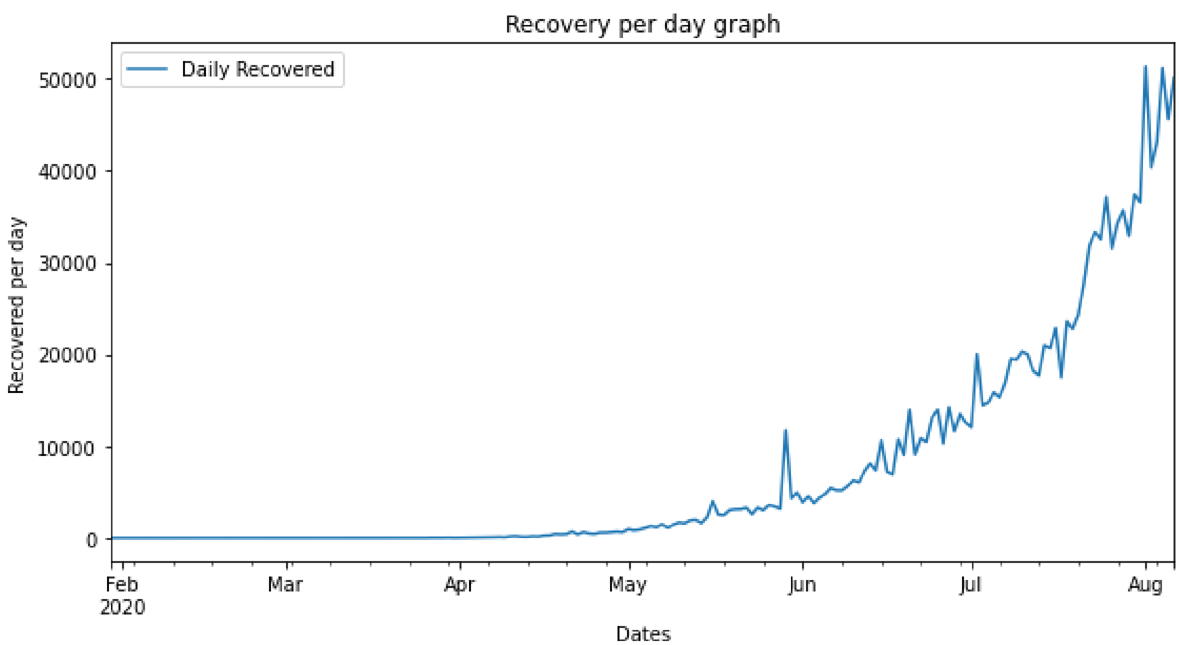
In [ ]:
```
Recovered_India.plot(x='Date', y=['Total Recovered','Daily Recovered'
], figsize=[10,5])
plt.title('Recovery Rate graph')
plt.ylabel('Recovery', fontsize=10);
plt.xlabel('Dates', fontsize=10)
```

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



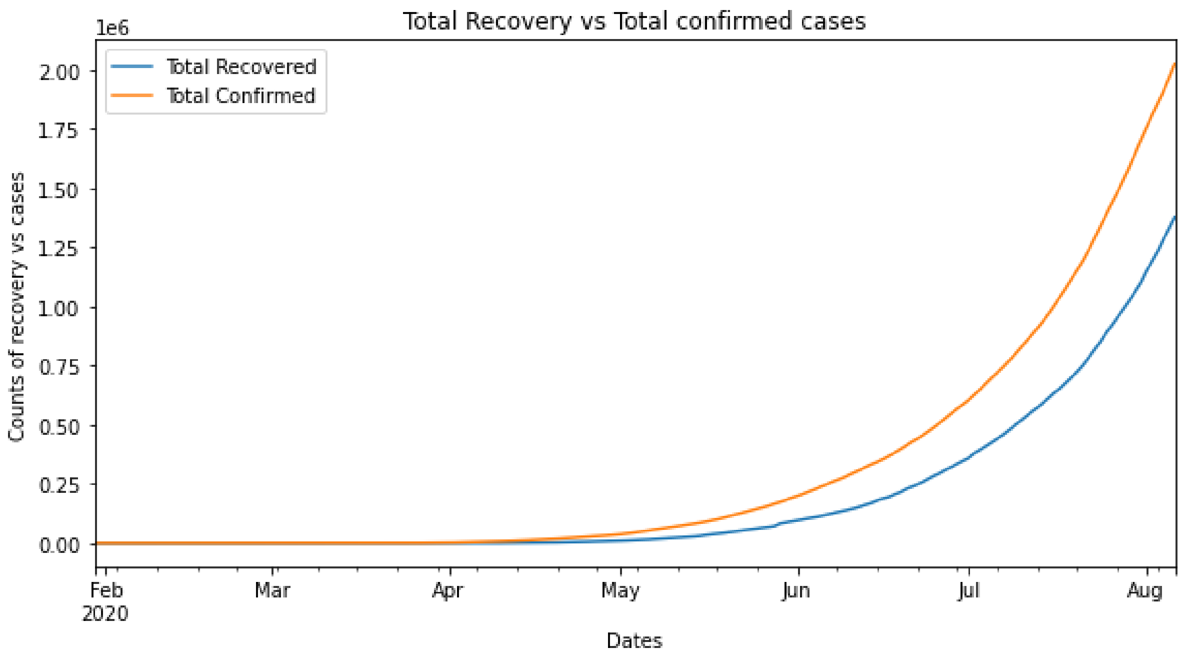
In [ ]:
```
Recovered_India.plot(x='Date', y=['Daily Recovered'], figsize=[10,5])
plt.title('Recovery per day graph')
plt.ylabel('Recovered per day', fontsize=10);
plt.xlabel('Dates', fontsize=10)
```

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

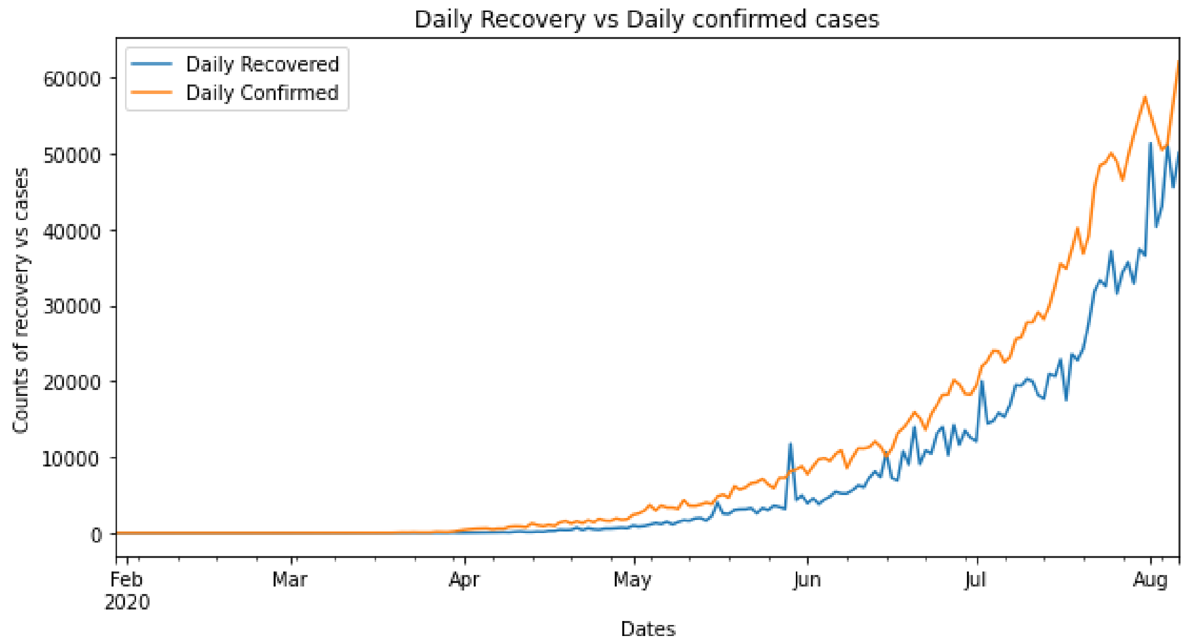


In [ ]:
```
National_data.plot(x='Date', y=['Total Recovered', 'Total Confirmed'],
figsize=[10,5])
plt.title('Total Recovery vs Total confirmed cases')
plt.ylabel('Counts of recovery vs cases', fontsize=10);
plt.xlabel('Dates', fontsize=10)
```

Out[ ]: Text(0.5, 0, 'Dates')



```
In [ ]: National_data.plot(x='Date', y=['Daily Recovered', 'Daily Confirmed'],
        figsize=[10,5])
        plt.title('Daily Recovery vs Daily confirmed cases')
        plt.ylabel('Counts of recovery vs cases', fontsize=10);
        plt.xlabel('Dates', fontsize=10)

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



4. ETS Decomposition

```
In [ ]: from statsmodels.tsa.seasonal import seasonal_decompose
```

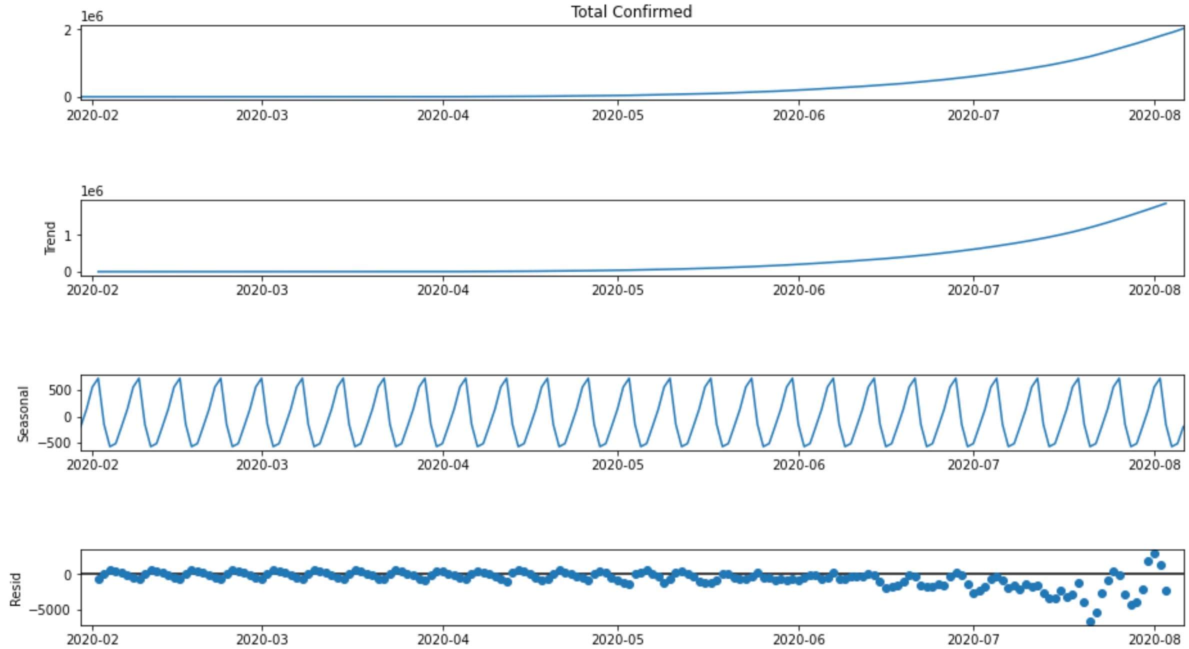
```
In [ ]: Total_confirmed_India = Confirmed_India[['Date', 'Total Confirmed']]
        Daily_confirmed_India  = Confirmed_India[['Date', 'Daily Confirmed']]
        Total_Recovered_India  = Recovered_India[['Date', 'Total Recovered']]
        Daily_Recovered_India = Recovered_India[['Date', 'Daily Recovered']]
```

```
In [ ]: C1= Total_confirmed_India.groupby('Date').sum()[Total_confirmed_India.
        groupby('Date').sum()['Total Confirmed']>=0]['Total Confirmed']
        C2= Daily_confirmed_India.groupby('Date').sum()[Daily_confirmed_India.
        groupby('Date').sum()['Daily Confirmed']>=0]['Daily Confirmed']
        R1= Total_Recovered_India.groupby('Date').sum()[Total_Recovered_India.
        groupby('Date').sum()['Total Recovered']>=0]['Total Recovered']
        R2= Daily_Recovered_India.groupby('Date').sum()[Daily_Recovered_India.
        groupby('Date').sum()['Daily Recovered']>=0]['Daily Recovered']
```

```
In [ ]: result1 = seasonal_decompose(C1)
        result2 = seasonal_decompose(C2)
        result3 = seasonal_decompose(R1)
        result4 = seasonal_decompose(R2)
```

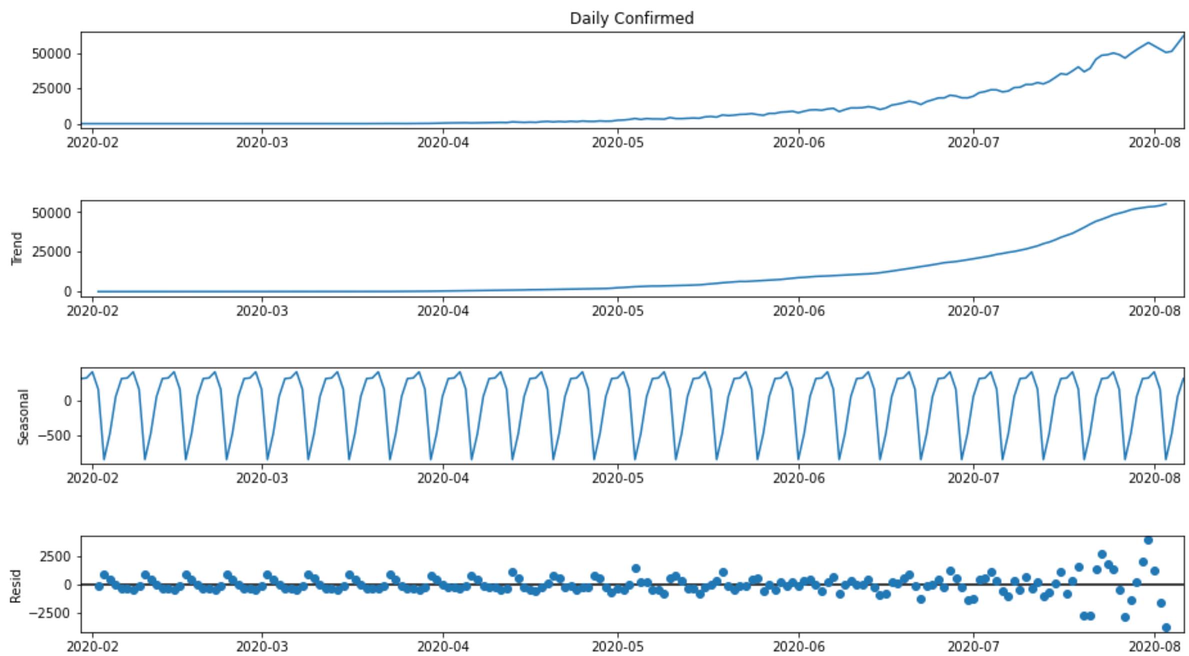
In [ ]:

```
fig = result1.plot()
fig.set_size_inches(15,8)
```

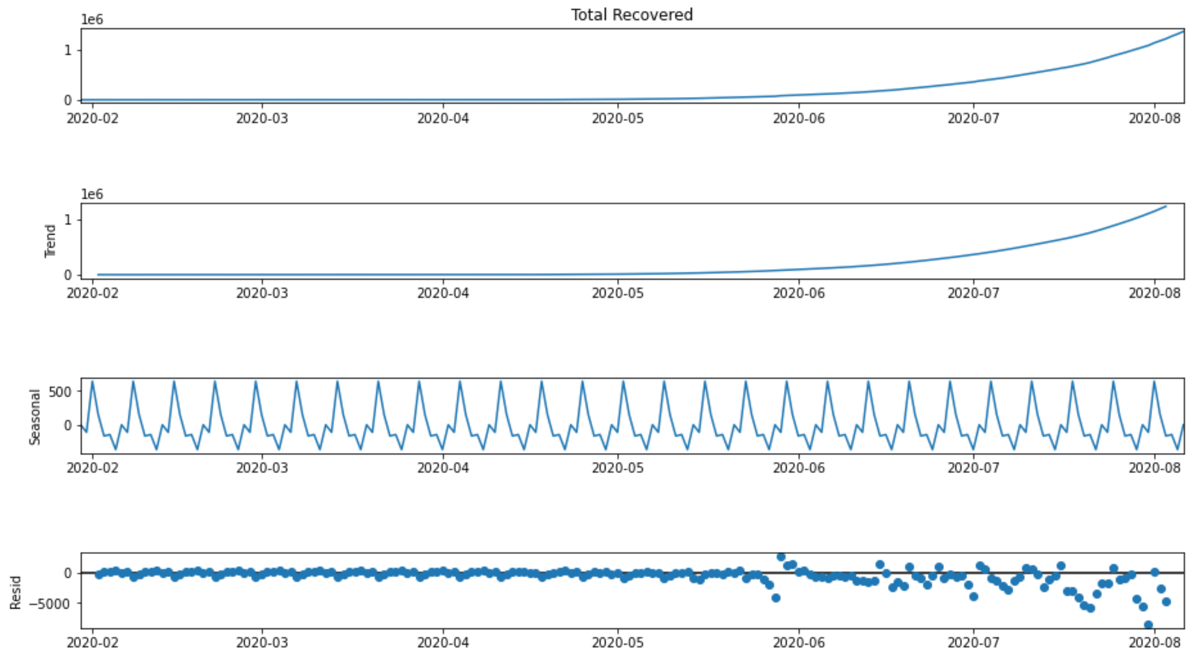


In [ ]:

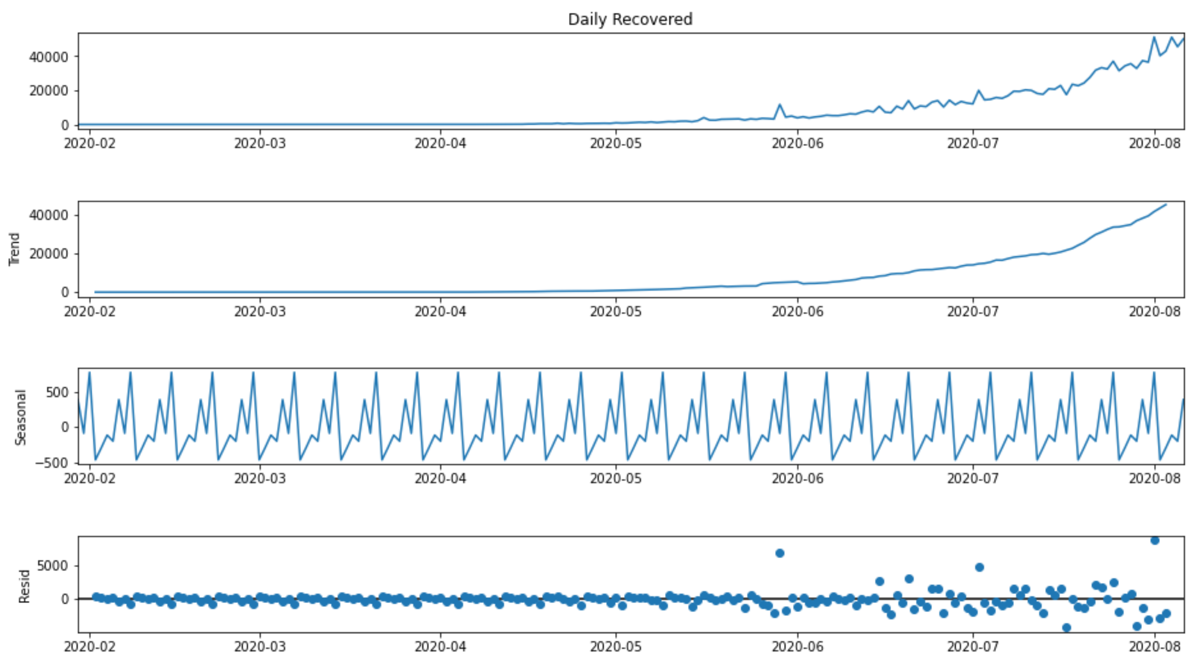
```
fig = result2.plot()
fig.set_size_inches(15,8)
```



```
In [ ]: fig = result3.plot()
        fig.set_size_inches(15,8)
```




```
In [ ]: fig = result4.plot()
        fig.set_size_inches(15,8)
```



6.Stationary Test

```
In [ ]: from statsmodels.tsa.stattools import adfuller
```

```
In [ ]: total_confirmed_to_fit = Total_confirmed_India.groupby('Date').sum()[Total_confirmed_India.groupby('Date').sum()['Total Confirmed']>=0]
        daily_confirmed_to_fit = Daily_confirmed_India.groupby('Date').sum()[Daily_confirmed_India.groupby('Date').sum()['Daily Confirmed']>=0]
        total_recovered_to_fit = Total_Recovered_India.groupby('Date').sum()[Total_Recovered_India.groupby('Date').sum()['Total Recovered']>=0]
        daily_recovered_to_fit = Daily_Recovered_India.groupby('Date').sum()[Daily_Recovered_India.groupby('Date').sum()['Daily Recovered']>=0]
```

```
In [ ]: def adf_check(time_series):

            #Pass in a time series, returns ADF report
            result = adfuller(time_series)

            print('Augmented Dickey-Fuller Test:')

            labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']

            for value,label in zip(result,labels):
                print(label+' : '+str(value) )

            if result[1] <= 0.05:
                print("strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root and is stationary")
            else:
                print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
```

```
In [ ]: adf_check(total_confirmed_to_fit['Total Confirmed'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 1.0951132592667092
p-value : 0.9951746996904941
#Lags Used : 14
Number of Observations Used : 175
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

```
In [ ]: adf_check(daily_confirmed_to_fit['Daily Confirmed'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 4.576459292538224
p-value : 1.0
#Lags Used : 13
Number of Observations Used : 176
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [ ]:
```
adf_check(total_recovered_to_fit['Total Recovered'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 2.269294561442034
p-value : 0.9989348430184178
#Lags Used : 15
Number of Observations Used : 174
weak evidence against null hypothesis, time series has a unit root, in
dicating it is non-stationary
```

In [ ]:
```
adf_check(daily_recovered_to_fit['Daily Recovered'])
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 4.306772156564263
p-value : 1.0
#Lags Used : 14
Number of Observations Used : 175
weak evidence against null hypothesis, time series has a unit root, in
dicating it is non-stationary
```

7. Seasonal ARIMA Model

In [ ]:
```
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

7.1 Forecasting total confirmed cases in upcoming days

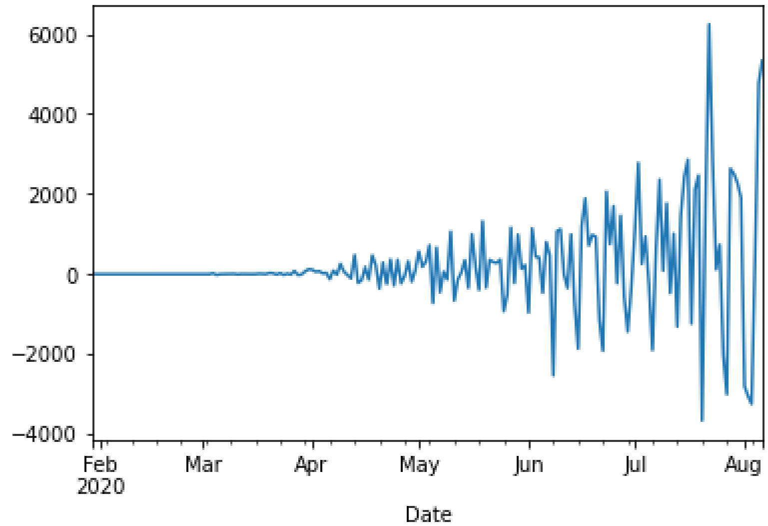
In [ ]:

```
model = SARIMAX(total_confirmed_to_fit['Total Confirmed'],order=(1,1,0
), seasonal_order=(1,1,1,12))
results = model.fit()
print(results.summary())
```

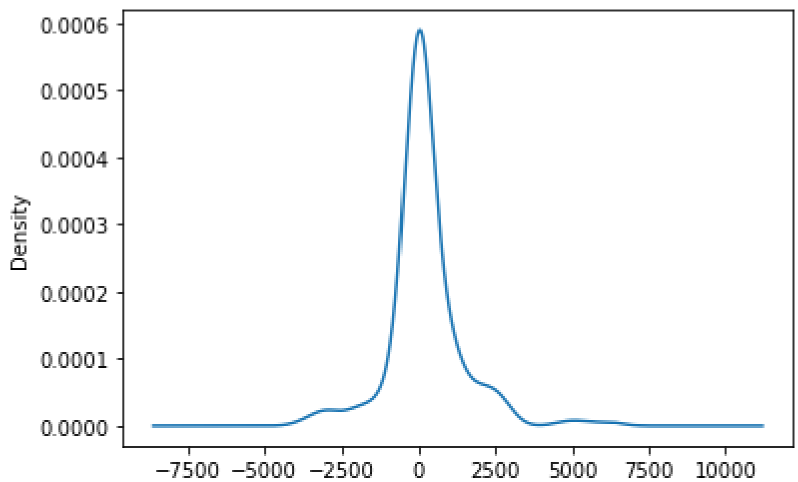
```
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:994: UserWarning: Non-stationary starting seasonal autoreg
ressive Using zeros as starting parameters.
  warn('Non-stationary starting seasonal autoregressive'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:1006: UserWarning: Non-invertible starting seasonal moving
average Using zeros as starting parameters.
  warn('Non-invertible starting seasonal moving average'
```

```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                      Total Confirmed   No. Observation
s:                  190
Model:             SARIMAX(1, 1, 0)x(1, 1, [1], 12)   Log Likelihood
-1520.482
Date:                              Tue, 03 Nov 2020   AIC
3048.963
Time:                                      17:13:17   BIC
3061.668
Sample:                                  01-30-2020   HQIC
3054.116
                                       - 08-06-2020
Covariance Type:                                opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------------------
ar.L1          1.0000      0.004    224.304      0.000       0.991
1.009
ar.S.L12      -0.2234      0.118     -1.899      0.058      -0.454
0.007
ma.S.L12      -0.6986      0.118     -5.936      0.000      -0.929
-0.468
sigma2      1.637e+06   8.77e-08   1.87e+13      0.000    1.64e+06
1.64e+06
==========================================================================================
Ljung-Box (Q):                      290.18   Jarque-Bera (JB):
237.00
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):            1913.50   Skew:
0.90
Prob(H) (two-sided):                  0.00   Kurtosis:
8.37
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

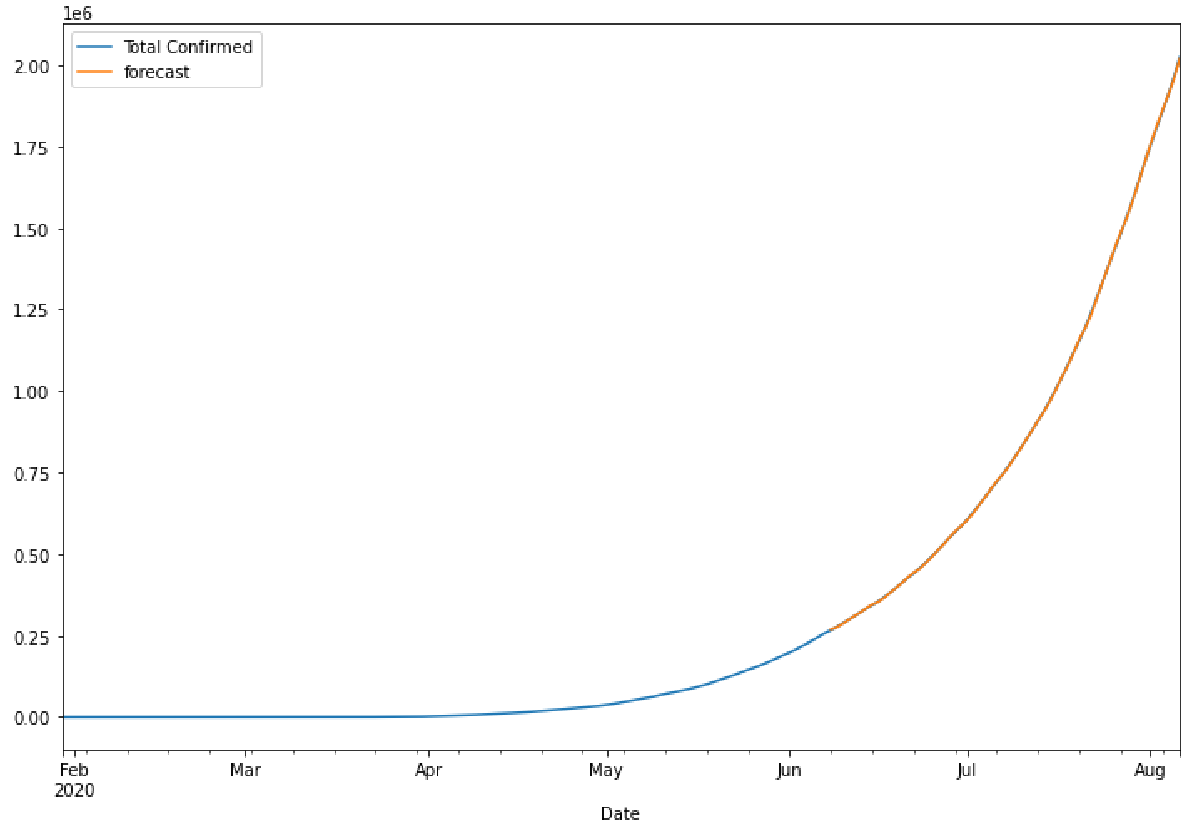
In [ ]:
```
results.resid.plot()
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba430e6a0>



In [ ]:
```
results.resid.plot(kind='kde')
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba439dd60>



In [ ]:
```
total_confirmed_to_fit['forecast'] = results.predict(start = 130, end=
190)
total_confirmed_to_fit[['Total Confirmed','forecast']].plot(figsize=(1
2,8));
```



In [ ]:
```
from pandas.tseries.offsets import DateOffset
```

In [ ]:
```
n = 80
last_date = total_confirmed_to_fit.index[-1]

future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]
```

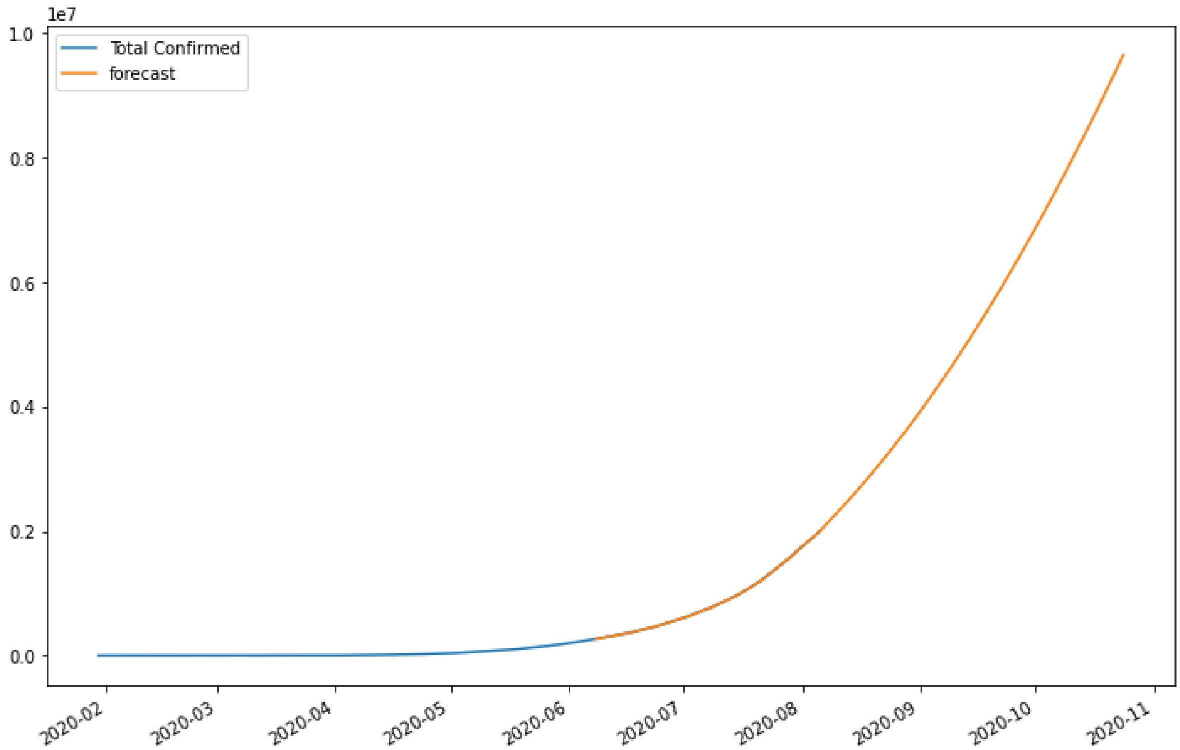
In [ ]:
```
future_dates_df = pd.DataFrame(index=future_dates[1:],columns=total_co
nfirmed_to_fit.columns)
future_df = pd.concat([total_confirmed_to_fit,future_dates_df])
total_confirmed_to_fit.tail()
```

Out[ ]:

	Total Confirmed	forecast
Date		
2020-08-02	1804857	1.807915e+06
2020-08-03	1855345	1.858599e+06
2020-08-04	1906627	1.906085e+06
2020-08-05	1963253	1.958476e+06
2020-08-06	2025423	2.020087e+06

In [ ]:
```
future_df['forecast'] = results.predict(start = 130, end = future_df.index[-1])
future_df[['Total Confirmed', 'forecast']].plot(figsize=(12, 8))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d1d24f0>



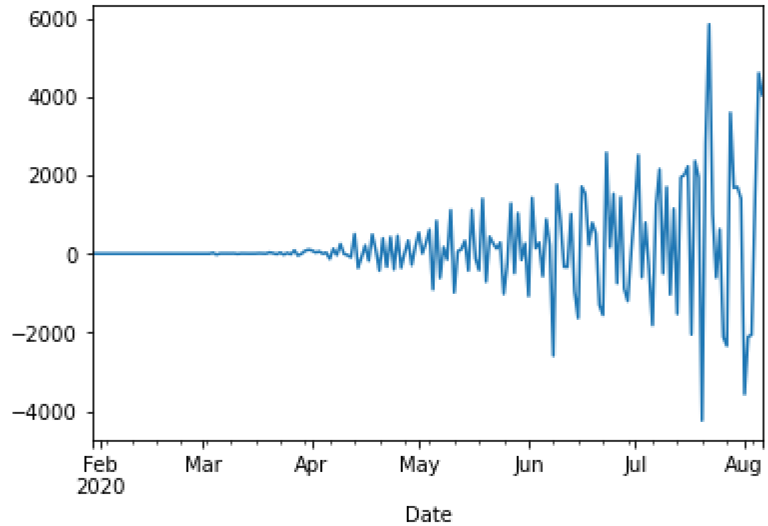
7.2 Forecasting daily confirmed cases in upcoming days

In [ ]:

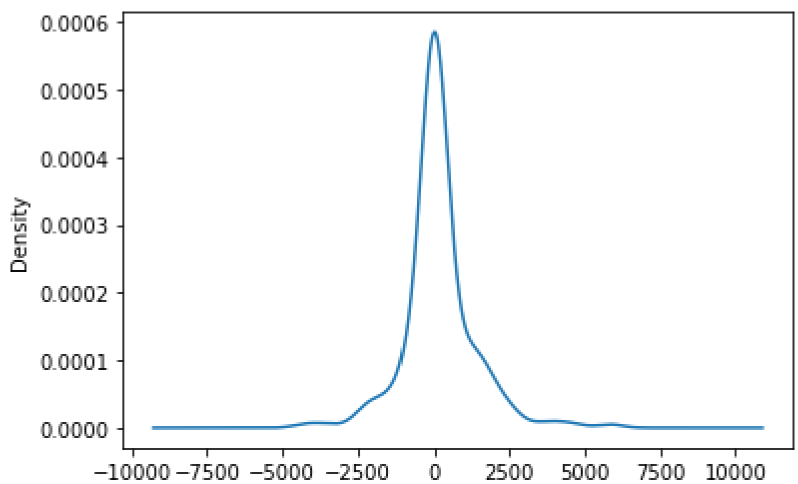
```
model = SARIMAX(daily_confirmed_to_fit['Daily Confirmed'],order=(1,1,0
), seasonal_order=(1,1,1,12))
results = model.fit()
print(results.summary())
```

```
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'

                                      SARIMAX Results
======================================================================
======================
Dep. Variable:                      Daily Confirmed   No. Observation
s:                  190
Model:             SARIMAX(1, 1, 0)x(1, 1, [1], 12)   Log Likelihood
-1513.631
Date:                              Tue, 03 Nov 2020   AIC
3035.262
Time:                                      17:13:19   BIC
3047.966
Sample:                                  01-30-2020   HQIC
3040.414
                                       - 08-06-2020
Covariance Type:                                opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.2797      0.043      6.493      0.000       0.195
0.364
ar.S.L12      -0.2566      0.096     -2.659      0.008      -0.446
-0.067
ma.S.L12      -0.7215      0.104     -6.925      0.000      -0.926
-0.517
sigma2      1.516e+06   9.29e+04     16.323      0.000    1.33e+06
1.7e+06
======================================================================
=============
Ljung-Box (Q):                      206.60   Jarque-Bera (JB):
168.18
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):            1800.56   Skew:
0.69
Prob(H) (two-sided):                  0.00   Kurtosis:
7.57
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

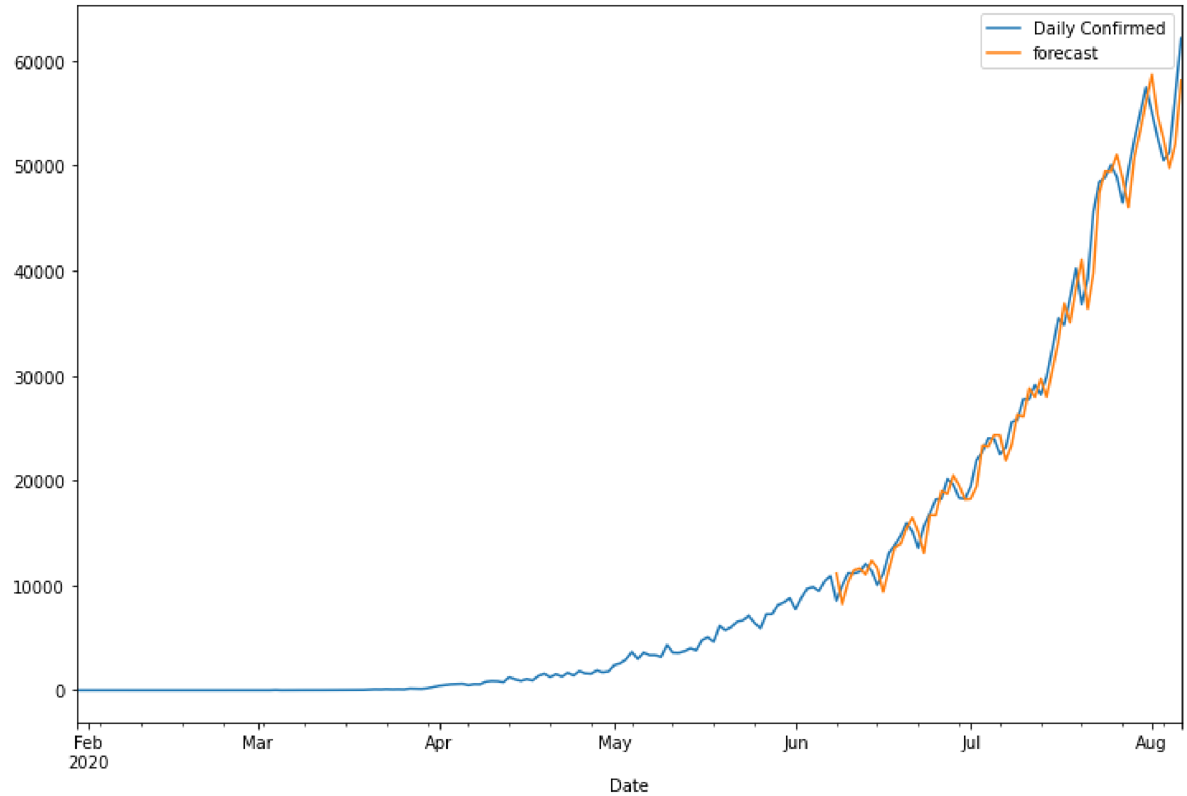
In [ ]:
```
results.resid.plot()
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d460610>



In [ ]:
```
results.resid.plot(kind='kde')
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba5831e20>



In [ ]:
```
daily_confirmed_to_fit['forecast'] = results.predict(start = 130, end=190)
daily_confirmed_to_fit[['Daily Confirmed','forecast']].plot(figsize=(12,8));
```



In [ ]:
```
from pandas.tseries.offsets import DateOffset
```

In [ ]:
```
n = 80
last_date = daily_confirmed_to_fit.index[-1]
future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]

future_dates_df = pd.DataFrame(index=future_dates[1:],columns=daily_confirmed_to_fit.columns)
future_df = pd.concat([daily_confirmed_to_fit,future_dates_df])
future_df.tail()
```

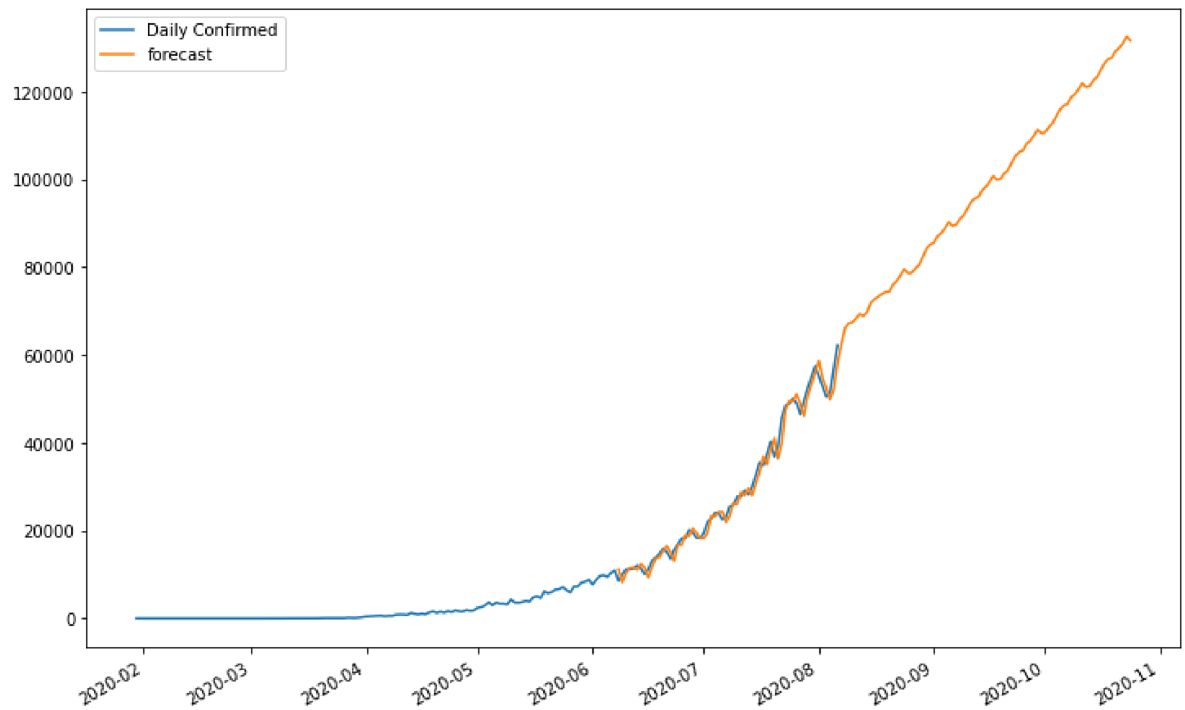
Out[ ]:

	Daily Confirmed	forecast
2020-10-20	NaN	NaN
2020-10-21	NaN	NaN
2020-10-22	NaN	NaN
2020-10-23	NaN	NaN
2020-10-24	NaN	NaN

In [ ]:

```
future_df['forecast'] = results.predict(start = 130, end = future_df.index[-1])
future_df[['Daily Confirmed', 'forecast']].plot(figsize=(12, 8))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d0a2370>



7.3 Forecasting total recovery in upcoming days

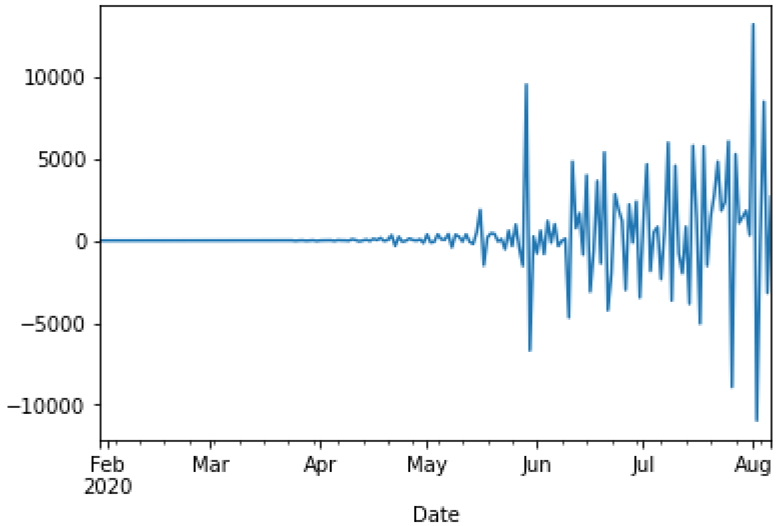
In [ ]:

```
model = SARIMAX(total_recovered_to_fit['Total Recovered'],order=(1,1,0
), seasonal_order=(1,1,1,12))
results = model.fit()
print(results.summary())
```

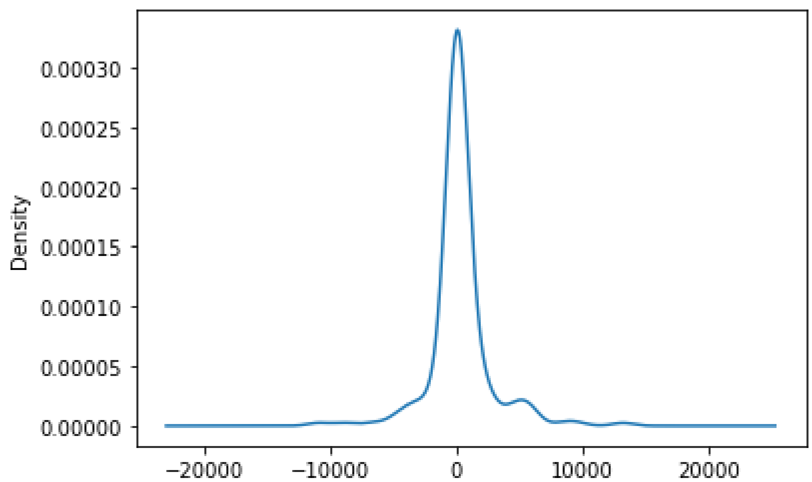
```
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:994: UserWarning: Non-stationary starting seasonal autoreg
ressive Using zeros as starting parameters.
  warn('Non-stationary starting seasonal autoregressive'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:1006: UserWarning: Non-invertible starting seasonal moving
average Using zeros as starting parameters.
  warn('Non-invertible starting seasonal moving average'
```

```
                                      SARIMAX Results                                       
======================================================================
======================
Dep. Variable:                        Total Recovered   No. Observation
s:                   190
Model:             SARIMAX(1, 1, 0)x(1, 1, [1], 12)   Log Likelihood
-1638.831
Date:                                Tue, 03 Nov 2020   AIC
3285.662
Time:                                        17:13:21   BIC
3298.367
Sample:                                    01-30-2020   HQIC
3290.815
                                         - 08-06-2020
Covariance Type:                                  opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.9299      0.014     64.474      0.000       0.902
0.958
ar.S.L12      -0.6523      0.189     -3.447      0.001      -1.023
-0.281
ma.S.L12       0.3600      0.208      1.727      0.084      -0.049
0.769
sigma2      6.441e+06   3.64e+05     17.698      0.000    5.73e+06
7.15e+06
======================================================================
=============
Ljung-Box (Q):                      143.35   Jarque-Bera (JB):
450.48
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):           90192.62   Skew:
0.58
Prob(H) (two-sided):                  0.00   Kurtosis:
10.73
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

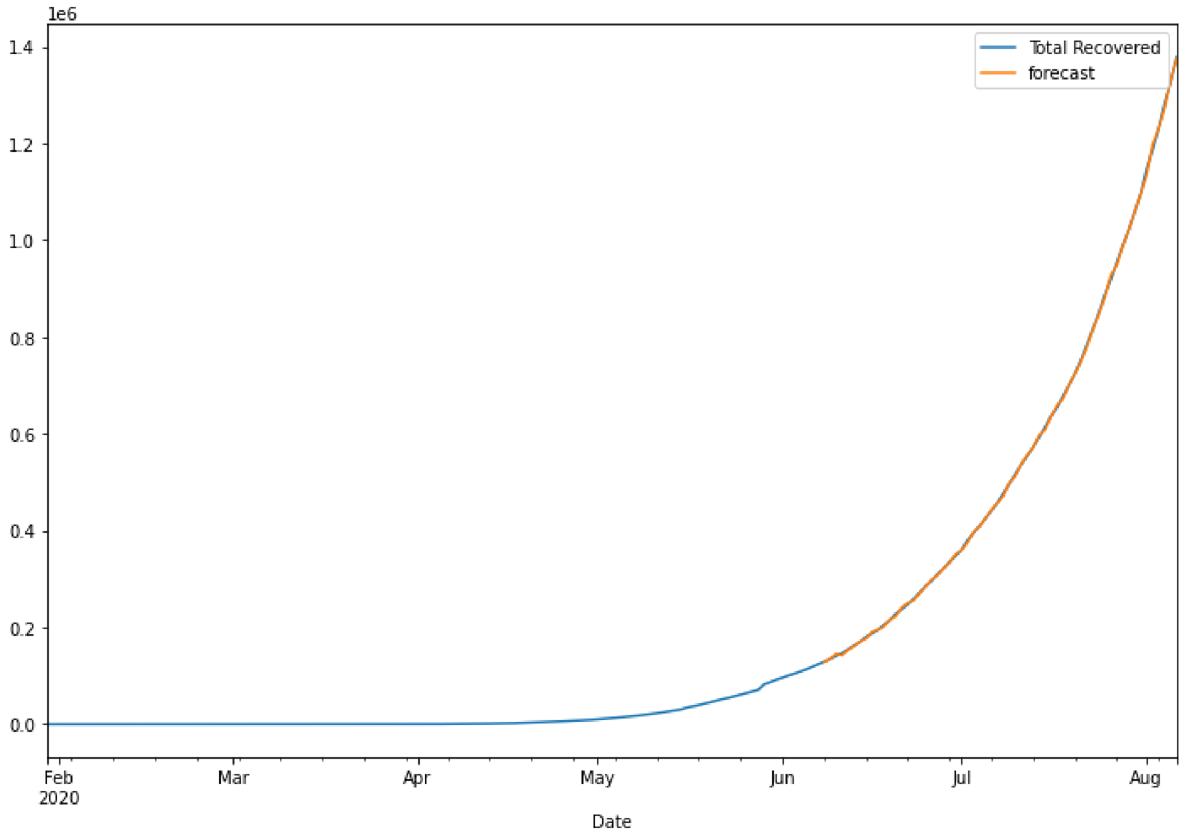
In [ ]:
```
results.resid.plot()
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba46e7d90>



In [ ]:
```
results.resid.plot(kind='kde')
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba583cdf0>



In [ ]:
```
total_recovered_to_fit['forecast'] = results.predict(start = 130, end=190)
total_recovered_to_fit[['Total Recovered','forecast']].plot(figsize=(12,8));
```



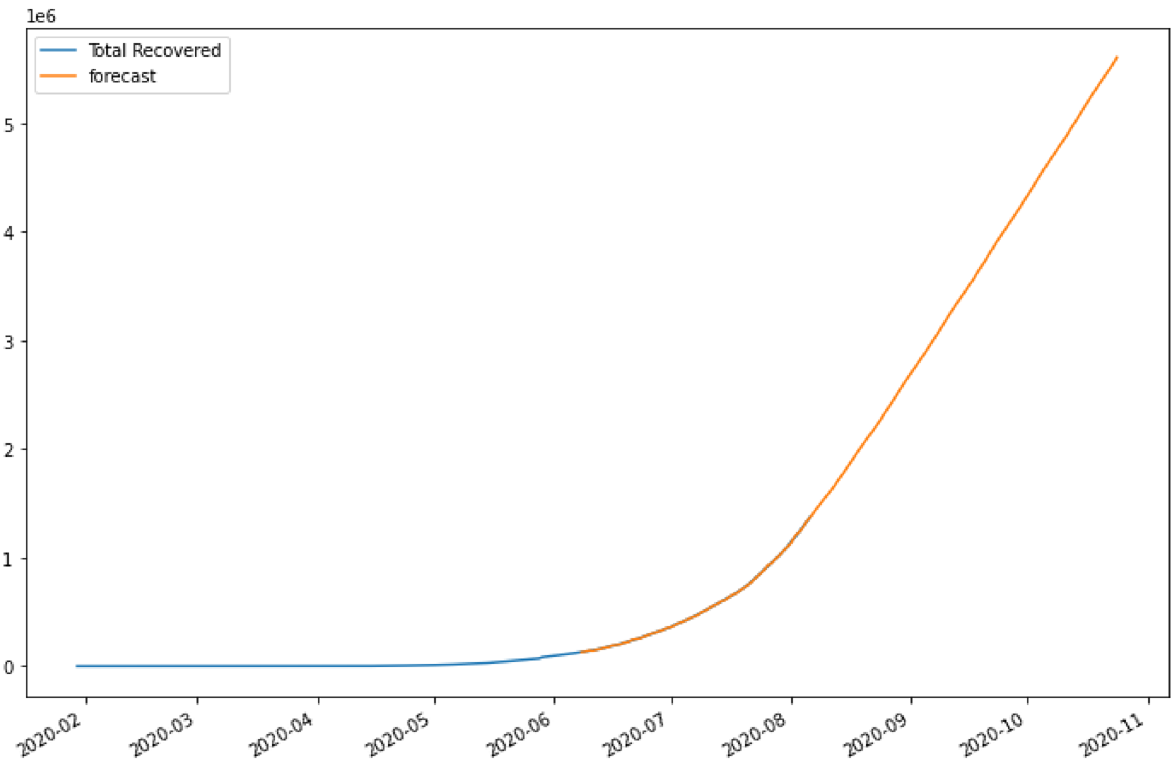
In [ ]:
```
n = 80
last_date = total_recovered_to_fit.index[-1]
future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]

future_dates_df = pd.DataFrame(index=future_dates[1:],columns=total_recovered_to_fit.columns)
future_df = pd.concat([total_recovered_to_fit,future_dates_df])
future_df.tail()
```

Out[ ]:

	Total Recovered	forecast
2020-10-20	NaN	NaN
2020-10-21	NaN	NaN
2020-10-22	NaN	NaN
2020-10-23	NaN	NaN
2020-10-24	NaN	NaN

```
In [ ]: future_df['forecast'] = results.predict(start = 130, end = future_df.index[-1])
        future_df[['Total Recovered', 'forecast']].plot(figsize=(12, 8))
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d05f040>
```



7.4 Forecasting daily recovery in upcoming days

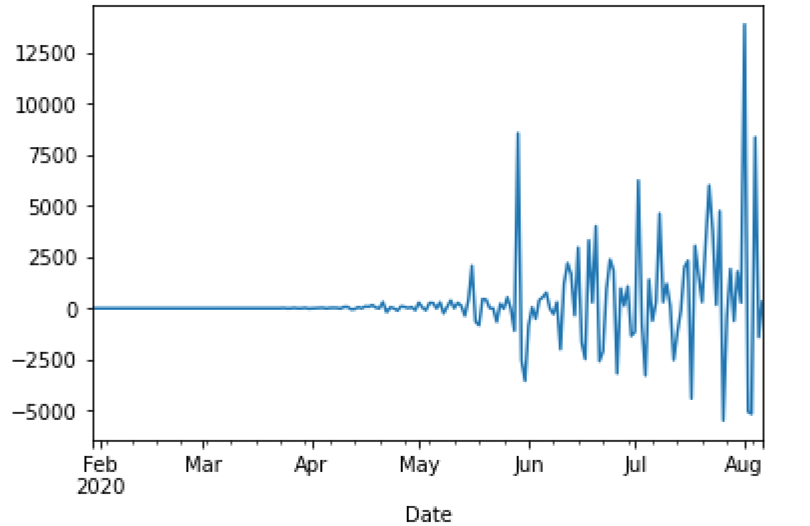
In [ ]:

```
model = SARIMAX(daily_recovered_to_fit['Daily Recovered'],order=(1,1,0
), seasonal_order=(1,1,1,12))
results = model.fit()
print(results.summary())
```

```
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'
C:\Users\arnab\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:159: ValueWarning: No frequency information was provided, so in
ferred frequency D will be used.
  warnings.warn('No frequency information was'

                                      SARIMAX Results
======================================================================
======================
Dep. Variable:                      Daily Recovered   No. Observation
s:                    190
Model:             SARIMAX(1, 1, 0)x(1, 1, [1], 12)   Log Likelihood
-1605.805
Date:                              Tue, 03 Nov 2020   AIC
3219.610
Time:                                      17:13:23   BIC
3232.314
Sample:                                  01-30-2020   HQIC
3224.762
                                       - 08-06-2020
Covariance Type:                                opg
======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.5362      0.038    -14.239      0.000      -0.610
-0.462
ar.S.L12       0.1656      0.124      1.337      0.181      -0.077
0.408
ma.S.L12      -0.8238      0.099     -8.338      0.000      -1.017
-0.630
sigma2      4.294e+06   1.67e+05     25.672      0.000    3.97e+06
4.62e+06
======================================================================
=============
Ljung-Box (Q):                     103.13   Jarque-Bera (JB):
1463.13
Prob(Q):                             0.00   Prob(JB):
0.00
Heteroskedasticity (H):          79315.77   Skew:
2.30
Prob(H) (two-sided):                 0.00   Kurtosis:
16.31
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

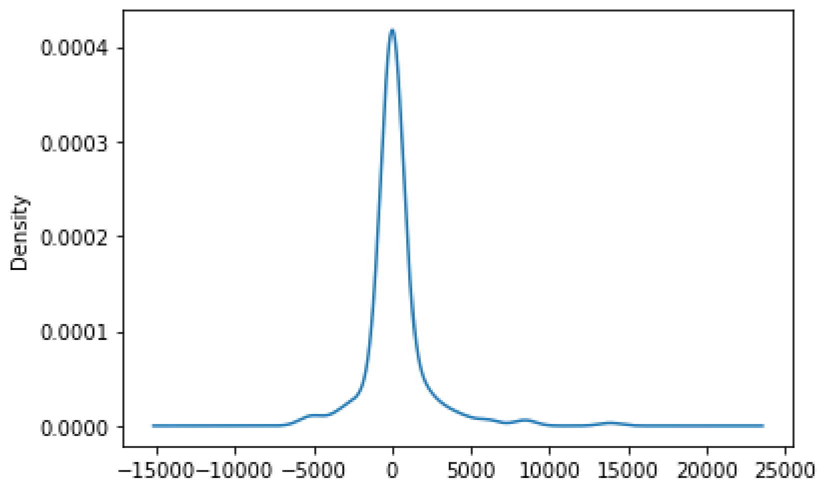
```
In [ ]: results.resid.plot()
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d528430>
```

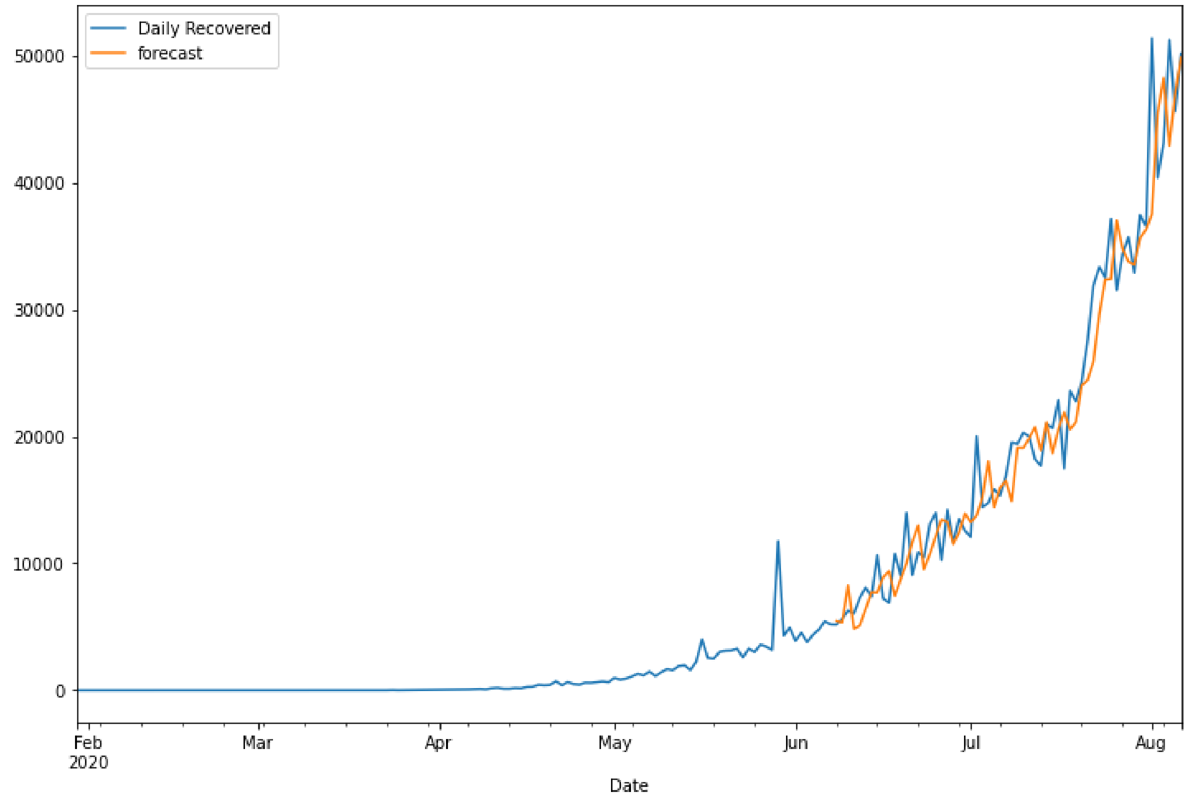



```
In [ ]: results.resid.plot(kind='kde')
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26b9d8b8b50>
```



In [ ]:
```
daily_recovered_to_fit['forecast'] = results.predict(start = 130, end=
190)
daily_recovered_to_fit[['Daily Recovered','forecast']].plot(figsize=(1
2,8));
```



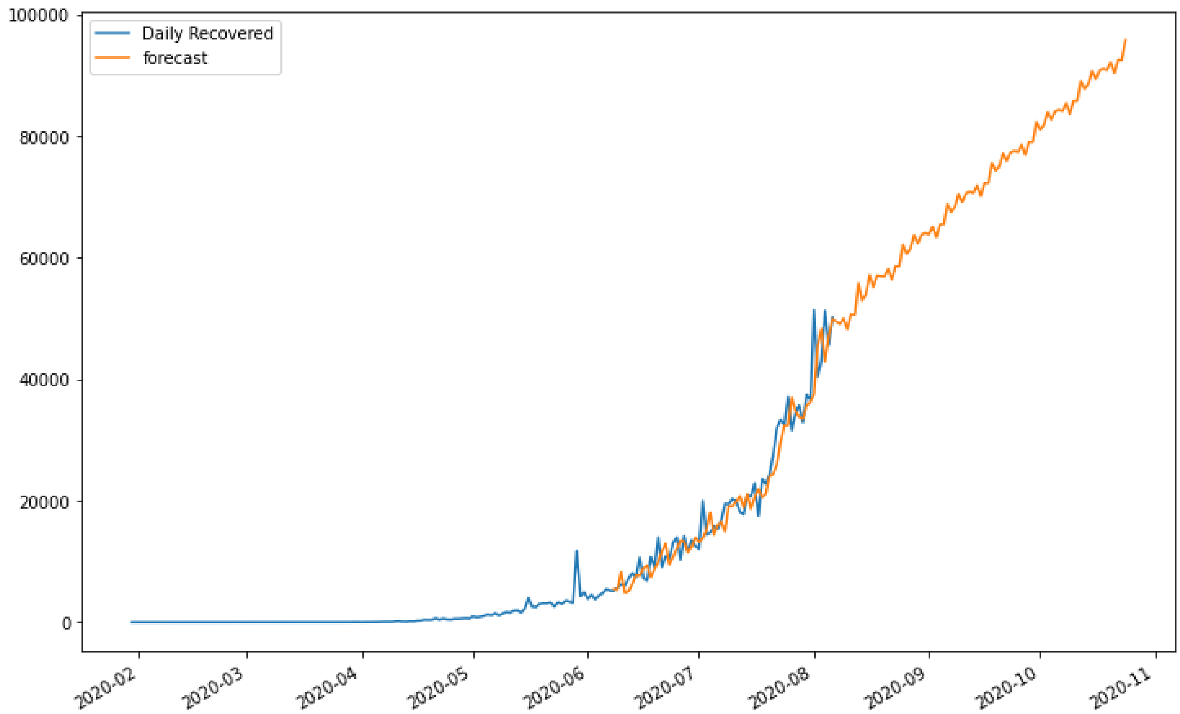
In [ ]:
```
n = 80
last_date = daily_recovered_to_fit.index[-1]
future_dates = [last_date+DateOffset(days=x) for x in range(1, n)]

future_dates_df = pd.DataFrame(index=future_dates[1:],columns=daily_re
covered_to_fit.columns)
future_df = pd.concat([daily_recovered_to_fit,future_dates_df])
future_df.tail()
```

Out[ ]:

	Daily Recovered	forecast
2020-10-20	NaN	NaN
2020-10-21	NaN	NaN
2020-10-22	NaN	NaN
2020-10-23	NaN	NaN
2020-10-24	NaN	NaN

```
In [ ]: future_df['forecast'] = results.predict(start = 130, end = future_df.index[-1])
        future_df[['Daily Recovered', 'forecast']].plot(figsize=(12, 8))
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x26ba59e3d00>
```



8. Performance Evaluation

```
In [ ]: from sklearn.metrics import mean_squared_error
        from math import sqrt
        from sklearn.metrics import r2_score
```

8.1 Mean Absolute Percentage Error

In [ ]:
```
test1 = total_confirmed_to_fit.iloc[130:, :]
test2 = daily_confirmed_to_fit.iloc[130:, :]
test3 = total_recovered_to_fit.iloc[130:, :]
test4 = daily_recovered_to_fit.iloc[130:, :]
test1.head()
```

Out[ ]:

	Total Confirmed	forecast
Date		
2020-06-08	266021	268561.098616
2020-06-09	276002	274939.976413
2020-06-10	287158	286044.093148
2020-06-11	298293	298300.933165
2020-06-12	309599	309949.449513

In [ ]:
```
mape1 = np.mean(np.abs(test1['forecast']-test1['Total Confirmed'])/np.
abs(test1['Total Confirmed']))
print('Accuracy of total confirmations forecast: ',100-(mape1*100))
```

```
Accuracy of total confirmations forecast:  99.7786465519674
```

In [ ]:
```
mape2 = np.mean(np.abs(test2['forecast']-test2['Daily Confirmed'])/np.
abs(test2['Daily Confirmed']))
print('Accuracy of daily confirmations forecast: ',100-(mape2*100))
```

```
Accuracy of daily confirmations forecast:  93.37483878422542
```

In [ ]:
```
mape3 = np.mean(np.abs(test3['forecast']-test3['Total Recovered'])/np.
abs(test3['Total Recovered']))
print('Accuracy of total recovery forecast: ',100-(mape3*100))
```

```
Accuracy of total recovery forecast:  99.27099011849319
```

In [ ]:
```
mape4 = np.mean(np.abs(test4['forecast']-test4['Daily Recovered'])/np.
abs(test4['Daily Recovered']))
print('Accuracy of daily recovery forecast: ',100-(mape4*100))
```

```
Accuracy of daily recovery forecast:  87.10281048121178
```

8.2 Root Mean squared error

In [ ]:
```
realVals1 = test1['Total Confirmed']
predictedVals1 = test1['forecast']
mse1 = mean_squared_error(realVals1, predictedVals1)
rmse1 = sqrt(mse1)
print('Root mean square error of total confirmed cases forecast: ',rms
e1)
```

```
Root mean square error of total confirmed cases forecast:  2125.131735
9060663
```

In [ ]:
```
realVals2 = test2['Daily Confirmed']
predictedVals2 = test2['forecast']
mse2 = mean_squared_error(realVals2, predictedVals2)
rmse2 = sqrt(mse2)
print('Root mean square error of daily confirmed cases forecast: ' ,rm
se2)
```

```
Root mean square error of daily confirmed cases forecast:  2022.747820
7589352
```

In [ ]:
```
realVals3 = test3['Total Recovered']
predictedVals3 = test3['forecast']
mse3 = mean_squared_error(realVals3, predictedVals3)
rmse3 = sqrt(mse3)
print('Root mean square error of total recovery forecast: ',rmse3)
```

```
Root mean square error of total recovery forecast:  4053.037978541228
```

In [ ]:
```
realVals4 = test4['Daily Recovered']
predictedVals4 = test4['forecast']
mse4 = mean_squared_error(realVals4, predictedVals4)
rmse4 = sqrt(mse4)
print('Root mean square error of daily recovery forecast: ', rmse4)
```

```
Root mean square error of daily recovery forecast:  3305.3554918793507
```

8.3 R2 score

In [ ]:
```
r2_score_1= r2_score(realVals1, predictedVals1)
print('R2 score of total confirmed cases forecast: ', r2_score_1)
```

```
R2 score of total confirmed cases forecast:  0.9999827456330113
```

In [ ]:
```
r2_score_2= r2_score(realVals2, predictedVals2)
print('R2 score of daily confirmed cases forecast: ', r2_score_2)
```

```
R2 score of daily confirmed cases forecast:  0.9833497185909372
```

In [ ]:
```
r2_score_3= r2_score(realVals3, predictedVals3)
print('R2 score of total recovery forecast: ', r2_score_3)
```

```
R2 score of total recovery forecast:  0.9998682037756753
```

In [ ]:
```
r2_score_4= r2_score(realVals4, predictedVals4)
print('R2 score of daily recovery forecast: ', r2_score_4)
```

```
R2 score of daily recovery forecast:  0.9304260281962908
```

CB.EN.U4CSE17310

PART-1: Exploratory Data Analysis

1. Getting the data

In [ ]:
```
# storing the file paths of our datasets

dfs = [
    'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/COVID-Time%20Series%20Data%20-%20Refined/confirmed.csv',
    'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/COVID-Time%20Series%20Data%20-%20Refined/deaths.csv',
    'https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/master/COVID-Time%20Series%20Data/COVID-Time%20Series%20Data%20-%20Refined/recovered.csv',
]
```

In [ ]:
```
confirmed = pd.read_csv(dfs[0])
deaths = pd.read_csv(dfs[1])
recovered = pd.read_csv(dfs[2])
```

In [ ]:
```
confirmed.head(3)
```

Out[ ]:

	State	Country	Latitude	Longitude	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0

3 rows × 240 columns

In [ ]:
```
deaths.head(3)
```

Out[ ]:

	State	Country	Latitude	Longitude	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/2
0	Alabama	US	32.539527	-86.644082	0	0	0	0	0	[illegible]
1	Alabama	US	30.727750	-87.722071	0	0	0	0	0	[illegible]
2	Alabama	US	31.868263	-85.387129	0	0	0	0	0	[illegible]

3 rows × 240 columns

In [ ]:
```
recovered.head(3)
```

Out[ ]:

	State	Country	Latitude	Longitude	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0

3 rows × 240 columns

2. Reshaping the data into a manageable format

In [ ]:
```
confirmed = confirmed.melt(['State', 'Country'], confirmed.columns[4
:], var_name='Dates', value_name='Confirmed')
confirmed.Dates = pd.to_datetime(confirmed.Dates)

deaths = deaths.melt(['State', 'Country'], deaths.columns[4:], var_nam
e='Dates', value_name='Deaths')
deaths.Dates = pd.to_datetime(deaths.Dates)

recovered = recovered.melt(['State', 'Country'], recovered.columns[4
:], var_name='Dates', value_name='Recovered')
recovered.Dates = pd.to_datetime(recovered.Dates)
```

In [ ]:
```
confirmed.head()
```

Out[ ]:

	State	Country	Dates	Confirmed
0	NaN	Afghanistan	2020-01-22	0
1	NaN	Albania	2020-01-22	0
2	NaN	Algeria	2020-01-22	0
3	NaN	Andorra	2020-01-22	0
4	NaN	Angola	2020-01-22	0

In [ ]:
```
confirmed_dates = confirmed.groupby('Dates').sum().reset_index()
deaths_dates = deaths.groupby('Dates').sum().reset_index()
recovered_dates = recovered.groupby('Dates').sum().reset_index()
```

In [ ]:
```
confirmed_dates.head()
```

Out[ ]:

	Dates	Confirmed
0	2020-01-22	556
1	2020-01-23	655
2	2020-01-24	943
3	2020-01-25	1436
4	2020-01-26	2123

In [ ]:
```
combined=pd.DataFrame({
    'Date': confirmed_dates.Dates,
    'Confirmed': confirmed_dates.Confirmed,
    'Deaths': deaths_dates.Deaths,
    'Recovered': recovered_dates.Recovered
})
combined = combined.melt('Date', combined.columns[1:], var_name='Condi
tion', value_name='Count')
combined.head()
```

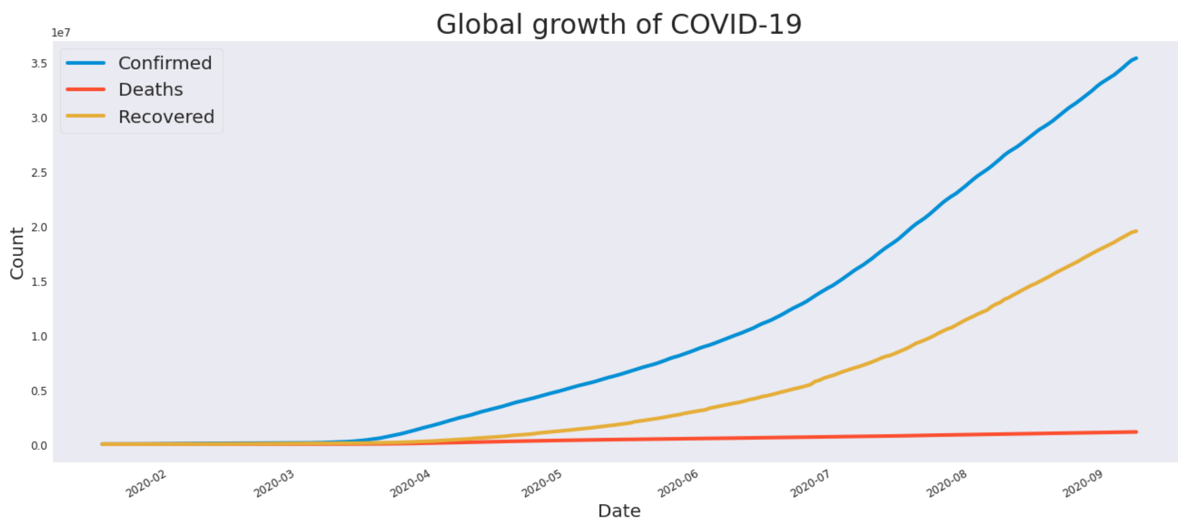
Out[ ]:

	Date	Condition	Count
0	2020-01-22	Confirmed	556
1	2020-01-23	Confirmed	655
2	2020-01-24	Confirmed	943
3	2020-01-25	Confirmed	1436
4	2020-01-26	Confirmed	2123

3. Graphing the number of confirmed cases, deaths and recoveries

In [ ]:

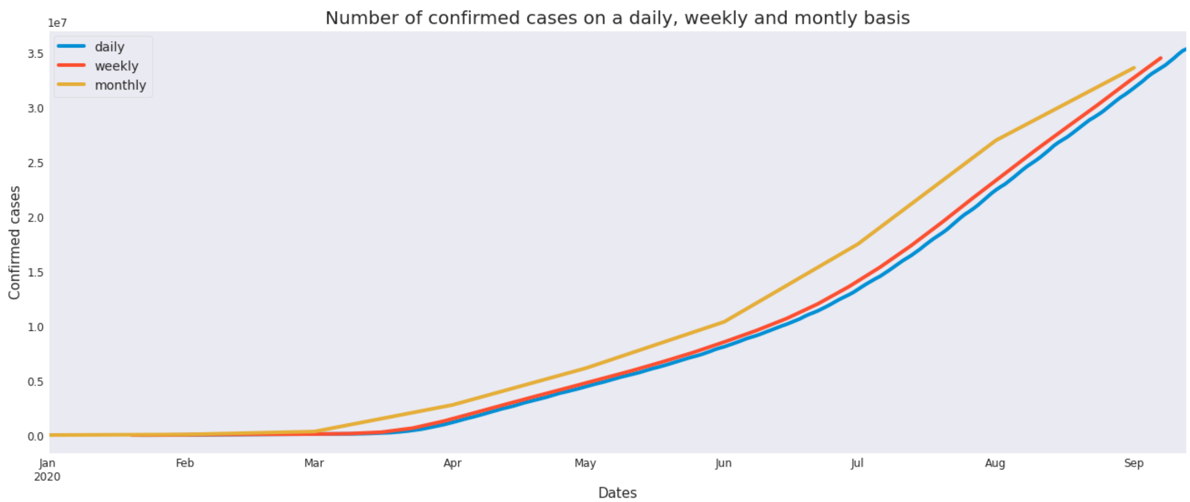
```
sns.set_style("dark")

plt.figure(figsize=(20,8));

growth = sns.lineplot(combined["Date"], combined["Count"], hue=combine
d["Condition"]);

growth.set_title("Global growth of COVID-19",fontsize=30);
growth.set_xlabel("Date",fontsize=20);
growth.set_ylabel("Count",fontsize=20);

plt.xticks(rotation=30);
plt.legend(fontsize='x-large');
```



a. Number of confirmed cases of COVID-19 on a daily, weekly and monthly basis

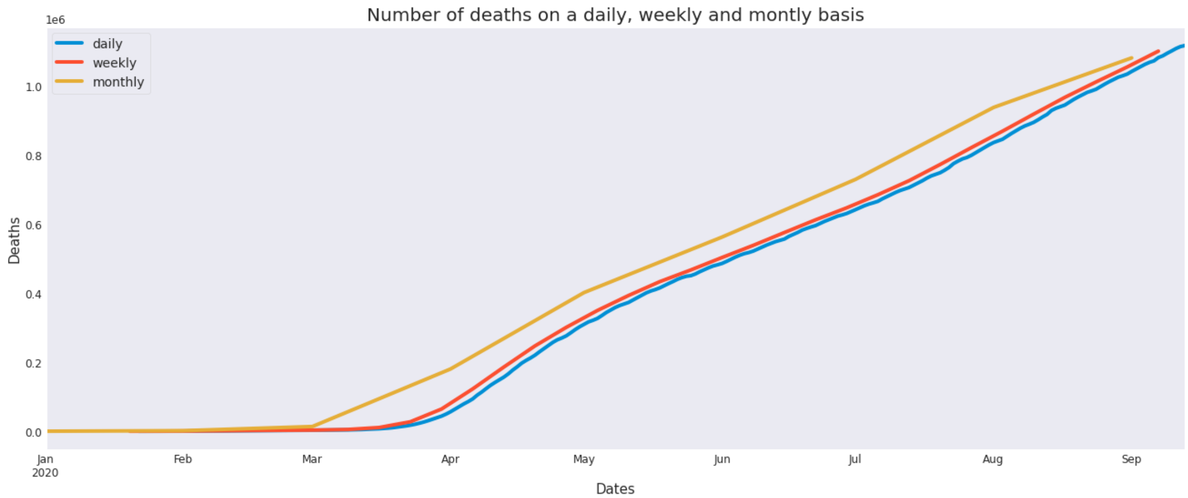
In [ ]:

```
plt.figure(figsize=(20,8));

conf = confirmed.groupby('Dates').sum()

conf.Confirmed.resample('d').mean().plot(label='daily');
conf.Confirmed.resample('w').mean().plot(label='weekly');
conf.Confirmed.resample('m').mean().plot(label='monthly');

plt.title('Number of confirmed cases on a daily, weekly and montly bas
is',fontsize=20);
plt.ylabel('Confirmed cases',fontsize=15);
plt.xlabel('Dates',fontsize=15)

plt.legend();
```



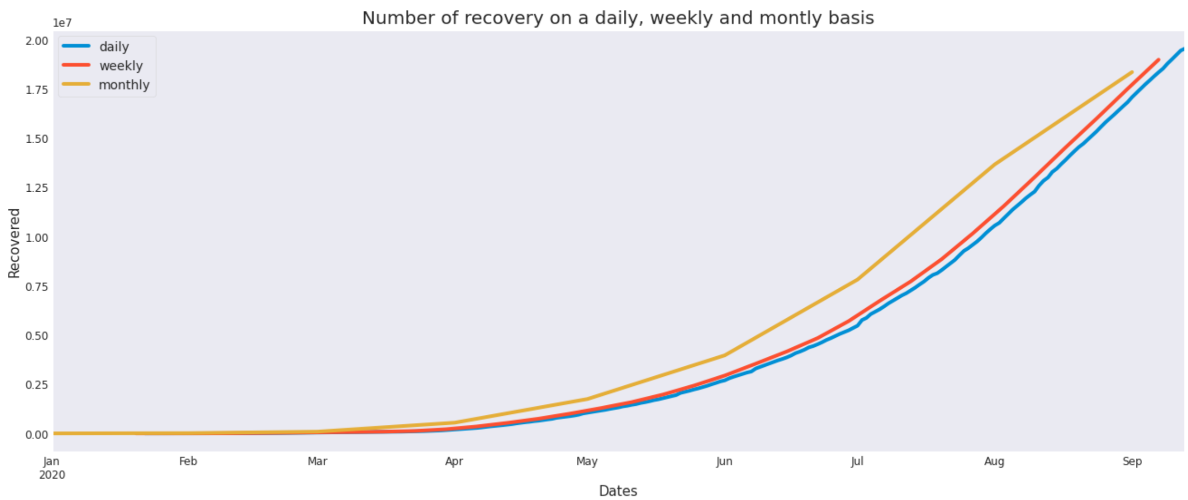
b. Number of COVID-19 related deaths on a daily, weekly and monthly basis

In [ ]:
```
plt.figure(figsize=(20,8));

mort = deaths.groupby('Dates').sum()

mort.Deaths.resample('d').mean().plot(label='daily');
mort.Deaths.resample('w').mean().plot(label='weekly');
mort.Deaths.resample('m').mean().plot(label='monthly');

plt.title('Number of deaths on a daily, weekly and montly basis', font
size=20);
plt.ylabel('Deaths',fontsize=15);
plt.xlabel('Dates',fontsize=15)

plt.legend();
```



c. Number of recovery from COVID-19 on a daily, weekly and monthly basis

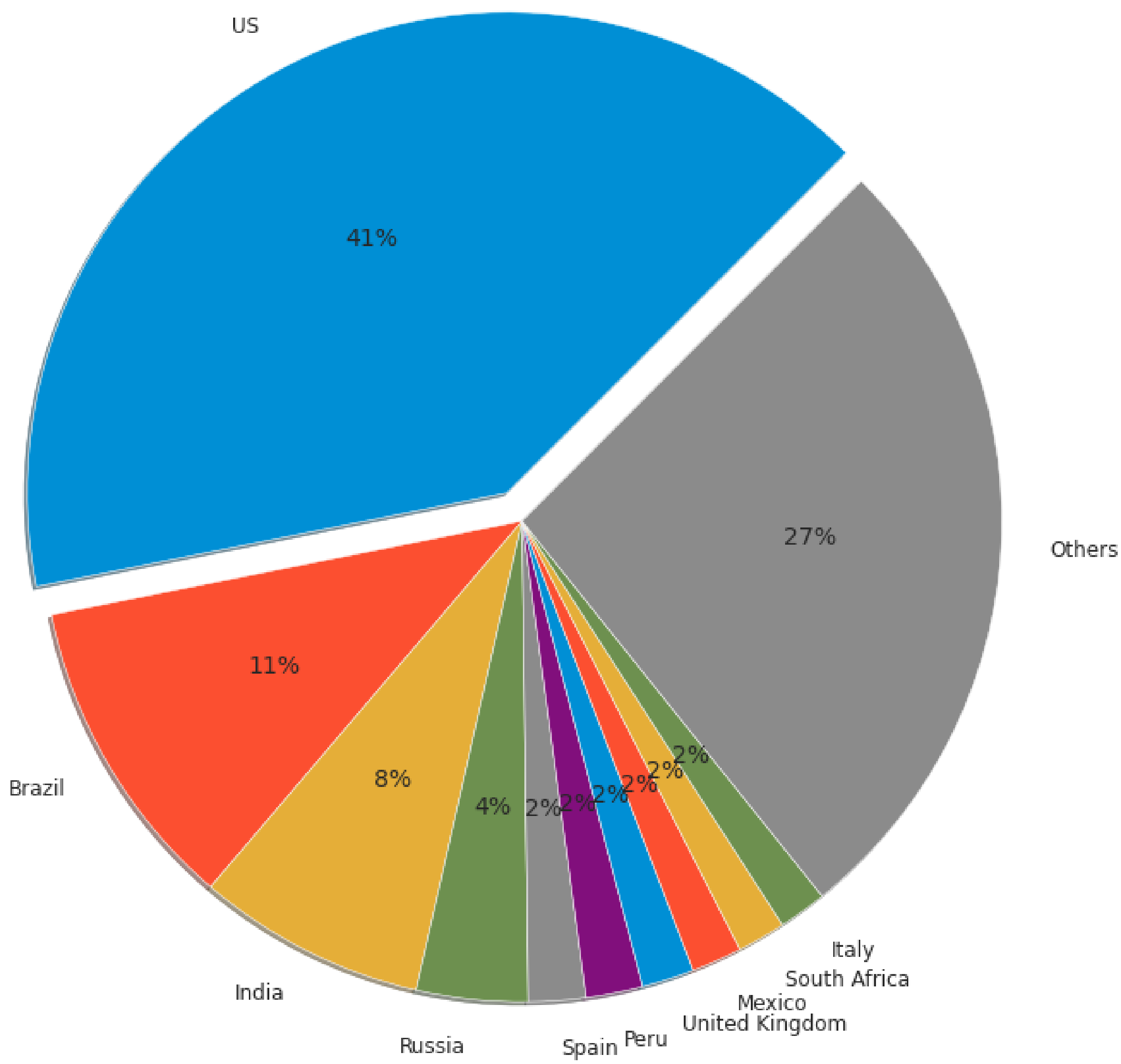
In [ ]:
```
plt.figure(figsize=(20,8));

rec = recovered.groupby('Dates').sum()

rec.Recovered.resample('d').mean().plot(label='daily');
rec.Recovered.resample('w').mean().plot(label='weekly');
rec.Recovered.resample('m').mean().plot(label='monthly');

plt.title('Number of recovery on a daily, weekly and montly basis',fontsize=20);
plt.ylabel('Recovered', fontsize=15);
plt.xlabel('Dates',fontsize=15)

plt.legend();
```



4. Pie Chart Visualizations for COVID-19

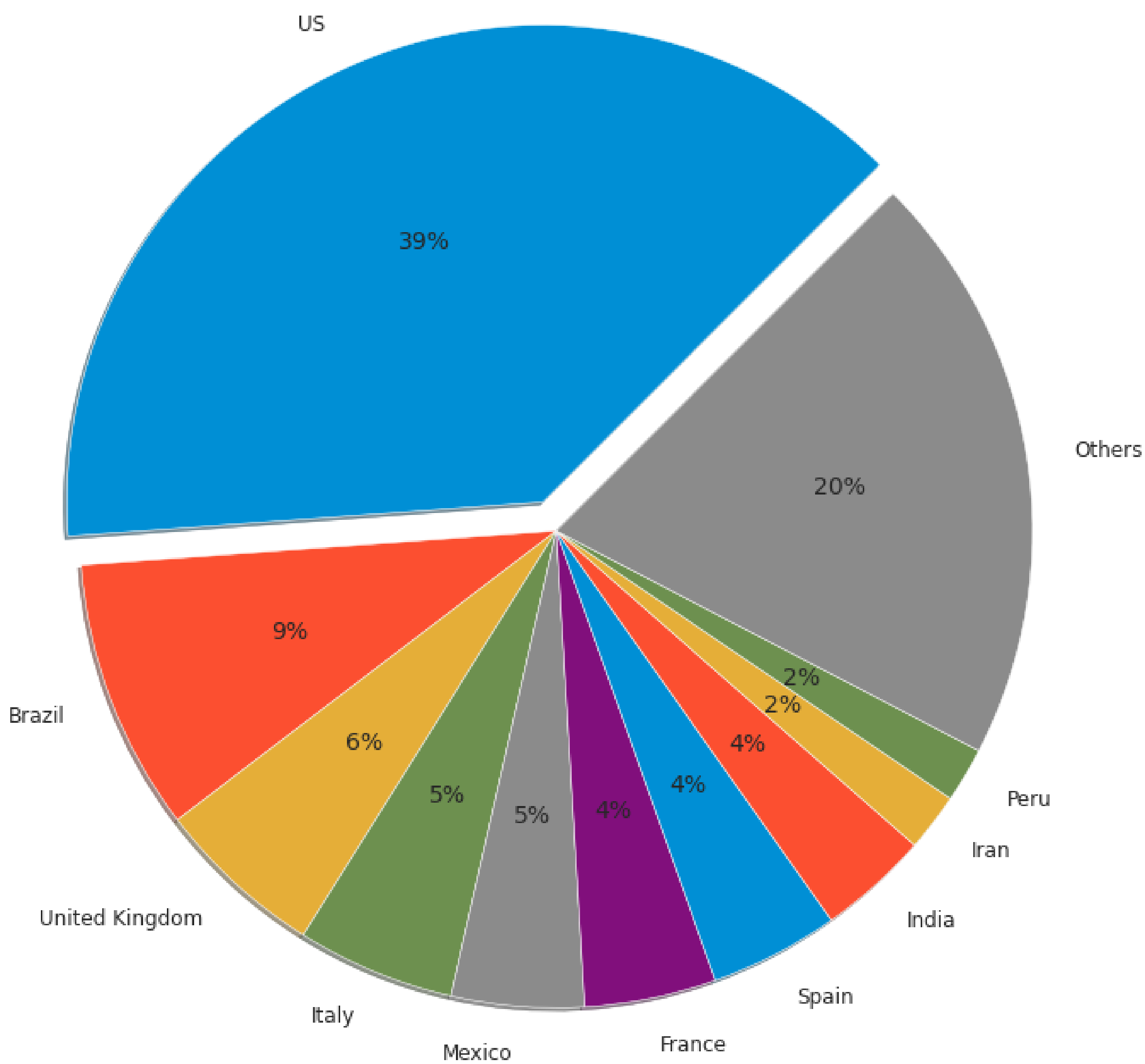
a. Confirmed COVID-19 cases

```
In [ ]: confirmed_country = confirmed.groupby('Country').sum().reset_index()
        confirmed_country.sort_values(by='Confirmed', ascending=False, inplace
        =True)
        confirmed_country_top10 = confirmed_country.iloc[:10,:]
        others = pd.DataFrame({'Country': ['Others'], 'Confirmed':[sum(confirm
        ed_country.iloc[10:,:].Confirmed)]})
        confirmed_country_top10 = pd.concat([confirmed_country_top10,others]).
        reset_index().drop('index', axis=1)

        plt.axis("equal")
        plt.pie(confirmed_country_top10.Confirmed, labels=confirmed_country_to
        p10.Country, radius=3, autopct='%0.0f%%', shadow=True, startangle=45,e
        xplode=[0.2,0,0,0,0,0,0,0,0,0,0])
        plt.show()
```



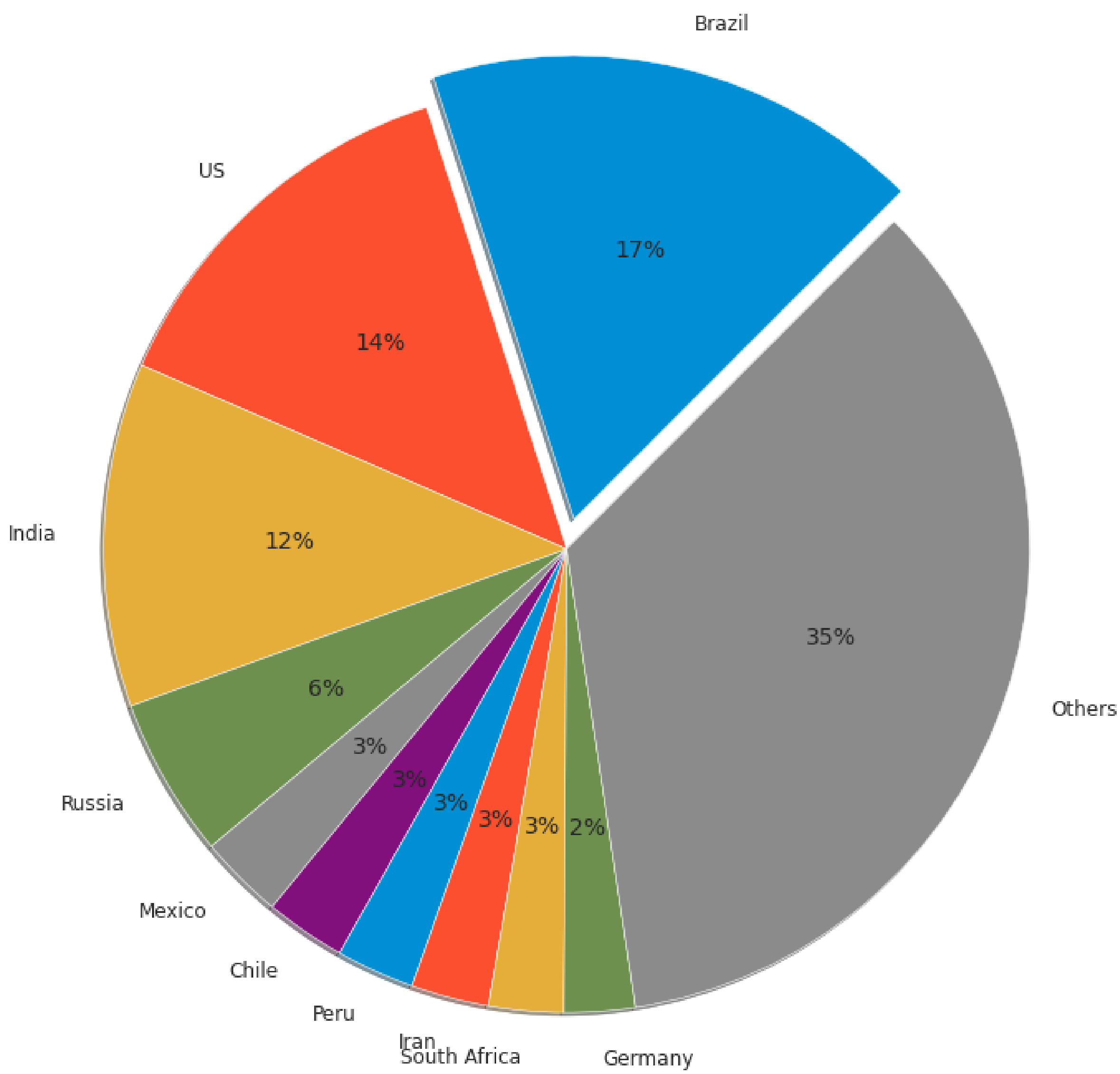
b. COVID-19 related deaths

In [ ]:
```
deaths_country = deaths.groupby('Country').sum().reset_index()
deaths_country.sort_values(by='Deaths', ascending=False, inplace=True)
deaths_country_top10 = deaths_country.iloc[:10,:]
others = pd.DataFrame({'Country': ['Others'], 'Deaths':[sum(deaths_cou
ntry.iloc[10:,:].Deaths)]})
deaths_country_top10 = pd.concat([deaths_country_top10,others]).reset_
index().drop('index', axis=1)

plt.axis("equal")
plt.pie(deaths_country_top10.Deaths, labels=deaths_country_top10.Count
ry, radius=3, autopct='%0.0f%%', shadow=True, startangle=45,explode=[
0.2,0,0,0,0,0,0,0,0,0,0])
plt.show()
```



c. Recovery from COVID-19

In [ ]:
```
recovered_country = recovered.groupby('Country').sum().reset_index()
recovered_country.sort_values(by='Recovered', ascending=False, inplace
=True)
recovered_country_top10 = recovered_country.iloc[:10,:]
others = pd.DataFrame({'Country': ['Others'], 'Recovered':[sum(recover
ed_country.iloc[10:,:].Recovered)]})
recovered_country_top10 = pd.concat([recovered_country_top10,others]).
reset_index().drop('index', axis=1)

plt.axis("equal")
plt.pie(recovered_country_top10.Recovered, labels=recovered_country_to
p10.Country, radius=3, autopct='%0.0f%%', shadow=True, startangle=45,e
xplode=[0.2,0,0,0,0,0,0,0,0,0,0])
plt.show()
```



5. Mortality rate and recovery rate visualized

In [ ]:
```
confirmed_country = confirmed.groupby('Country').sum().reset_index()
deaths_country = deaths.groupby('Country').sum().reset_index()
recovered_country = recovered.groupby('Country').sum().reset_index()

combined_countries = confirmed_country.merge(deaths_country, on=['Coun
try']).merge(recovered_country,on=['Country'])
combined_countries.head()
```

Out[ ]:

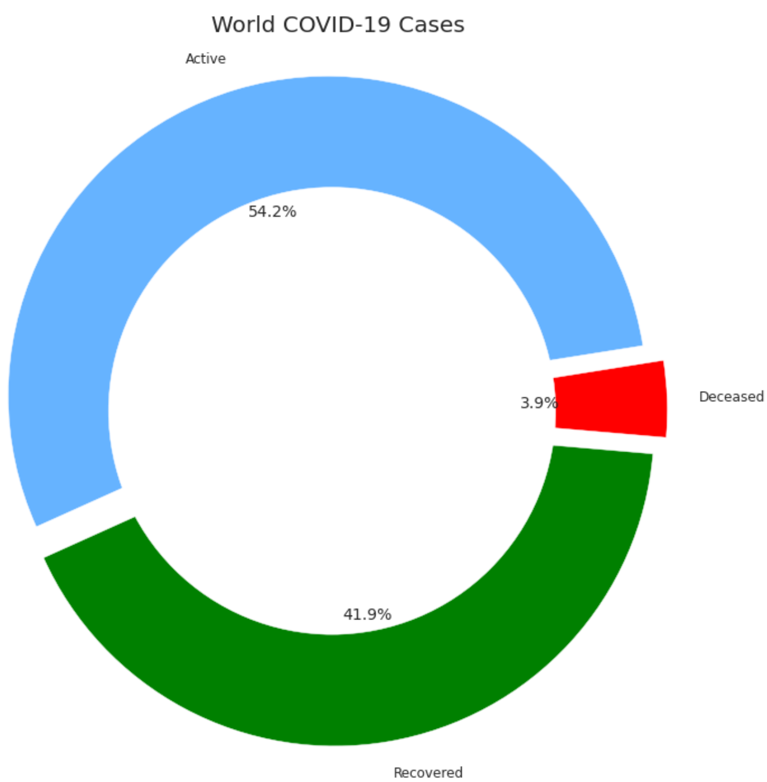
	Country	Confirmed	Deaths	Recovered
0	Afghanistan	3745342	114595	2139148
1	Albania	583139	17360	329905
2	Algeria	3088876	145505	2089806
3	Andorra	145841	7952	111054
4	Angola	125569	5483	45985

World Covid19 cases

Number of active cases can be calculated by subtracting the sum of recovered and death cases from the confirmed cases

In [ ]:

```
conf=combined_countries['Confirmed'].sum()
deth=combined_countries['Deaths'].sum()
rec=combined_countries['Recovered'].sum()
active=conf-(rec-deth)
labels = ['Active','Recovered','Deceased']
sizes = [active,rec,deth]
color= ['#66b3ff','green','red']
explode = []

for i in labels:
    explode.append(0.05)

plt.figure(figsize= (15,10))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=9, explode
=explode,colors = color)
centre_circle = plt.Circle((0,0),0.70,fc='white')

fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('World COVID-19 Cases',fontsize = 20)
plt.axis('equal')
plt.tight_layout()
```



The dataset we have contains the columns confirmed cases, deaths and recovery. We can make use of these fields and get two other fields, **Mortality rate** and **Recovery rate**, which are equally important in our analysis

In [ ]:
```
combined_countries['Mortality_rate'] = combined_countries['Deaths']/co
mbined_countries['Confirmed']
combined_countries['Recovery_rate'] = combined_countries['Recovered']/
combined_countries['Confirmed']

combined_countries.head()
```

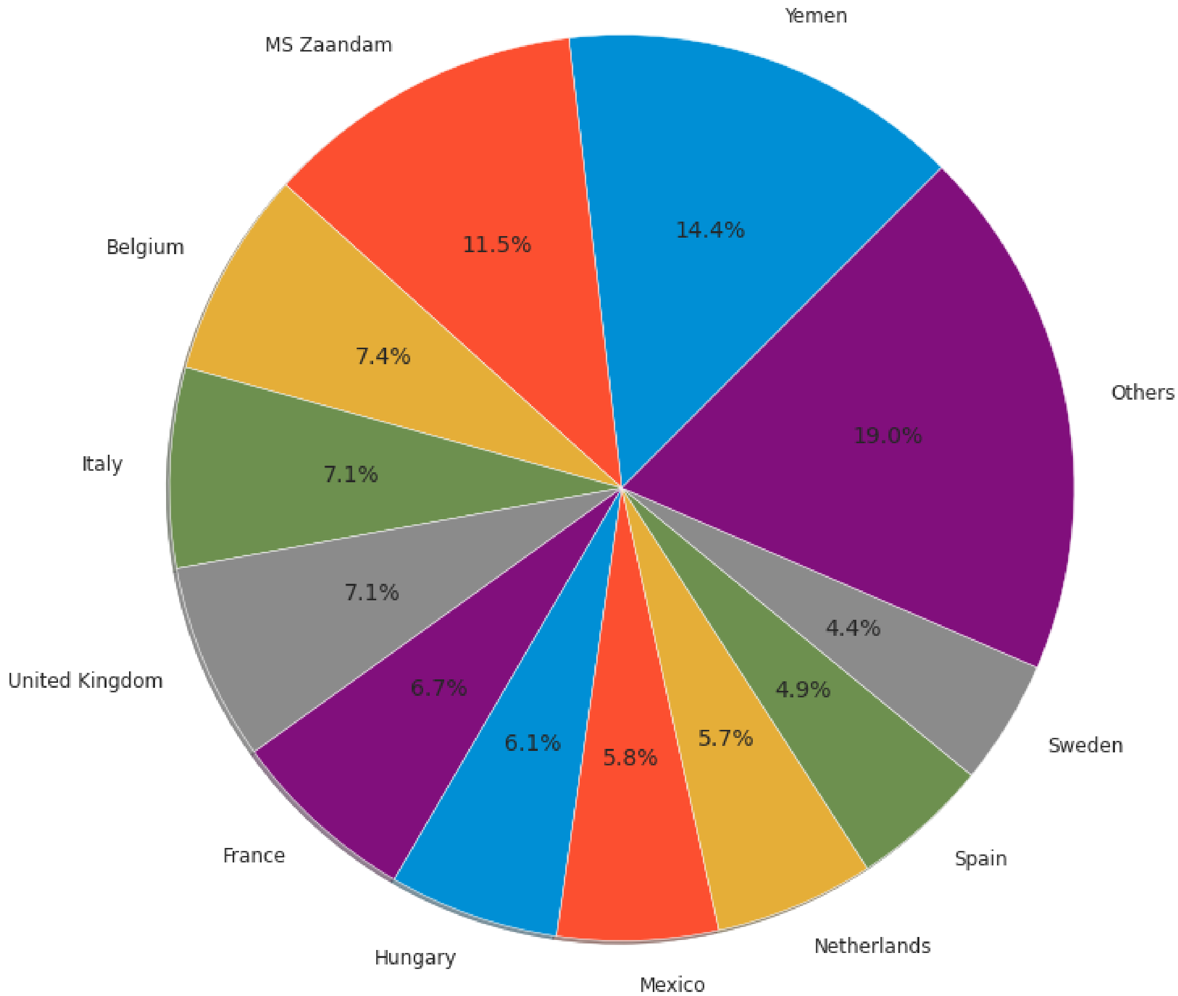
Out[ ]:

	Country	Confirmed	Deaths	Recovered	Mortality_rate	Recovery_rate
0	Afghanistan	3745342	114595	2139148	0.030597	0.571149
1	Albania	583139	17360	329905	0.029770	0.565740
2	Algeria	3088876	145505	2089806	0.047106	0.676559
3	Andorra	145841	7952	111054	0.054525	0.761473
4	Angola	125569	5483	45985	0.043665	0.366213

In [ ]:
```
x, y = 10, 20

top20_mortality = combined_countries.sort_values(by='Mortality_rate',
ascending=False).reset_index().loc[:x,['Country','Mortality_rate']]
top20_recovery = combined_countries.sort_values(by='Recovery_rate', as
cending=False).reset_index().loc[:y,['Country','Recovery_rate']]

otherMortality = sum(combined_countries.sort_values(by='Mortality_rat
e', ascending=False).loc[x:,['Country','Mortality_rate']]['Mortality_r
ate'])
otherRecovery = sum(combined_countries.sort_values(by='Recovery_rate',
ascending=False).loc[y:,['Country','Recovery_rate']]['Recovery_rate'])

otherMortality = pd.DataFrame({
    'Country':['Others'],
    'Mortality_rate':[otherMortality]
})

otherRecovery = pd.DataFrame({
    'Country':['Others'],
    'Recovery_rate':[otherRecovery]
})

top20_mortality = pd.concat([top20_mortality,otherMortality])
top20_recovery = pd.concat([top20_recovery,otherRecovery])
```

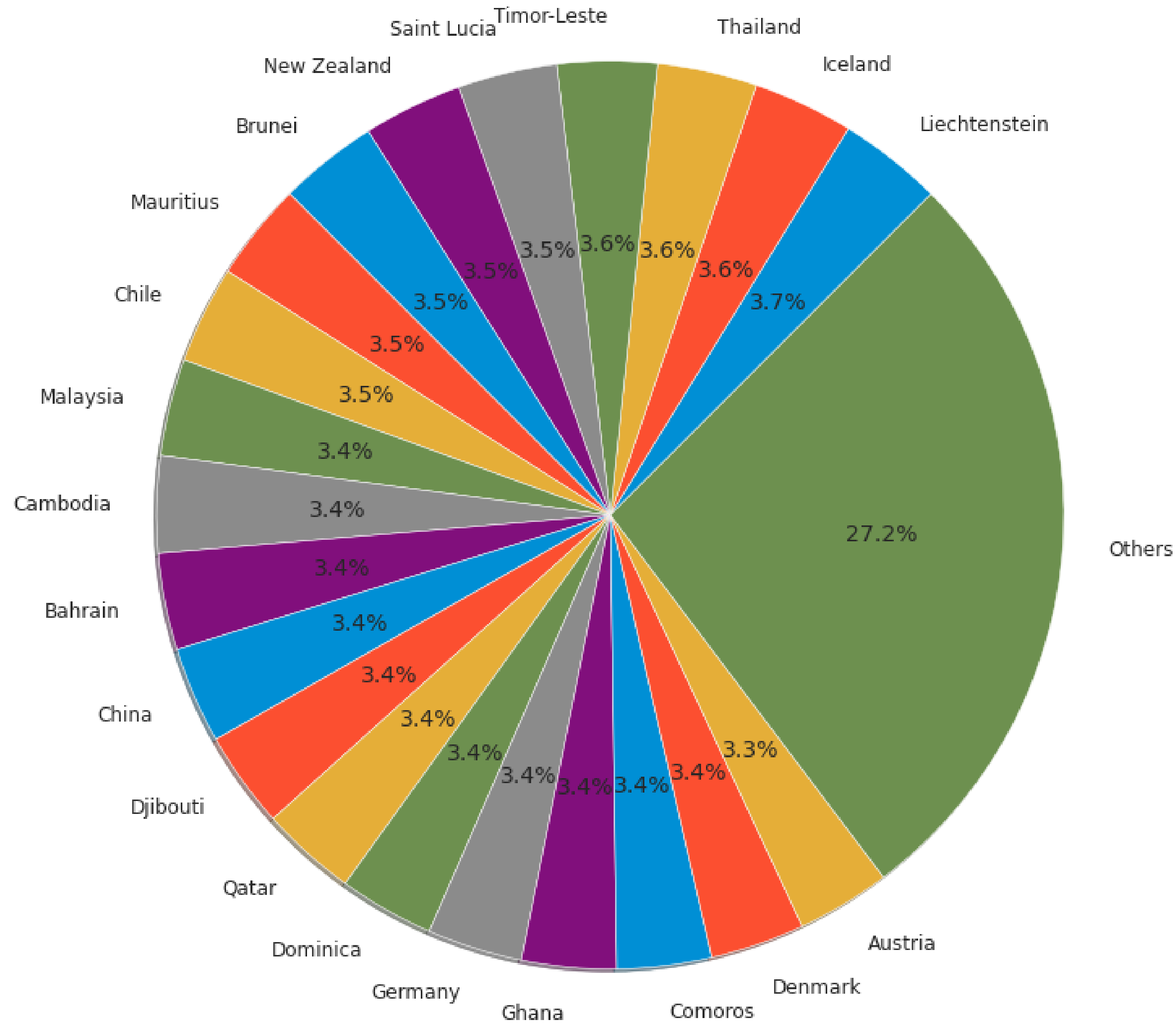
a. Top 10 countries with high COVID-19 mortality rate

```
In [ ]: plt.axis("equal")
        plt.pie(top20_mortality.Mortality_rate, labels=top20_mortality.Country
        , radius=3, autopct='%0.1f%%', shadow=True, startangle=45)
        plt.show()
```



b. Top 20 countries with high COVID-19 recovery rate

```
In [ ]: plt.axis("equal")
        plt.pie(top20_recovery.Recovery_rate, labels=top20_recovery.Country, r
        adius=3, autopct='%0.1f%%', shadow=True, startangle=45)
        plt.show()
```



PART-2: Building the Deep learning model to forecast the number of confirmed COVID-19 cases

With the situation at hand, we are in need of a ML/DL model which is capable of forecasting the expected number of confirmed cases in the near future.

Here, I've trained a deep learning model which uses LSTM for forcasting into the future.

The model will learn different trends in the provided timeseries data and will help us forecast the supposed number of cases.

1. Loading the data

The goal here is to build a deep learning model which uses LSTM to forecast the number of active cases in the world.

In [ ]:

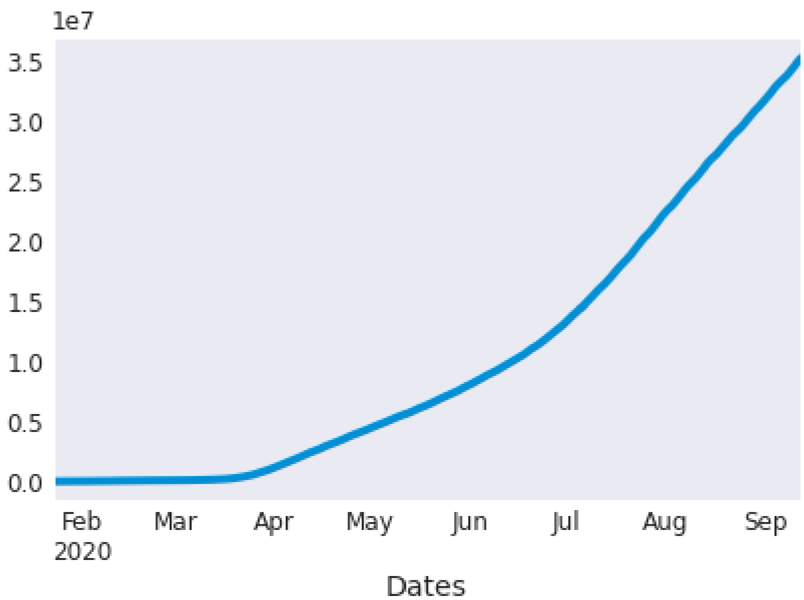
```
# dfs[0] has the link to the confirmed.csv dataset, which is the datas
et I'm going to train this model on
confirmed = pd.read_csv(dfs[0])

# I've use the pandas melt to reshape the dataframe into a format that
is easy to work with.
confirmed = confirmed.melt(['State', 'Country'], confirmed.columns[4
:], var_name='Dates', value_name='Confirmed')

"""

Also, I convert the date column from string object to datetime object
 since we have the need to perform timeseries
forecasting.


"""

confirmed.Dates = pd.to_datetime(confirmed.Dates)

confirmed.head()
```

Out[ ]:

	State	Country	Dates	Confirmed
0	NaN	Afghanistan	2020-01-22	0
1	NaN	Albania	2020-01-22	0
2	NaN	Algeria	2020-01-22	0
3	NaN	Andorra	2020-01-22	0
4	NaN	Angola	2020-01-22	0

In [ ]:

```
"""

I've then use the pandas groupby to group the confirmed cases by the d
ate

Note: Since we are using the global data, when we use groupby we essen
tially sum all the confirmed cases for a given day
ie if we have 100 cases in place-1 and 200 cases in place-2 on a given
day, when we use groupby date the total number of
cases for that date is taken ie 200+100=300.


"""

days = confirmed.groupby('Dates').sum()
days.Confirmed.resample('d').sum().plot();
```



In [ ]:

```
days.head()
```

Out[ ]:

Dates	Confirmed
2020-01-22	556
2020-01-23	655
2020-01-24	943
2020-01-25	1436
2020-01-26	2123

In [ ]:
```
days.tail()
```

Out[ ]:

Dates	Confirmed
2020-09-09	34222892
2020-09-10	34557534
2020-09-11	34924974
2020-09-12	35244068
2020-09-13	35422326

In [ ]:
```
# total number of rows in our dataset

len(days)
```

Out[ ]: 236

2. Preparing the data for the LSTM model

In [ ]:
```
# I've taken the number of data in the test set is taken as 25, which
 basically is the last 25 days of our timeseries.
# The remaining is used traing our LSTM model

test_size = 25
test_ind = len(days) - test_size

train = days.iloc[:test_ind,:]
test = days.iloc[test_ind:,:]
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

# The data is normalized
scaler = MinMaxScaler(feature_range=(0,1))

scaler.fit(train)

scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

In [ ]:

```
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerato
r

"""

I've set the batch size for the model as 20.

I'll be passing the data of the previous 20 days to predict the estima
ted number of cases for the  next day

To make the work easy, I've used the Tensorflow's TimeseriesGenerator
 to make the training and validation timeseries data
That we have to train the model with

Note: batch size should be less than len(test)


"""

BS = 20

generator = TimeseriesGenerator(scaled_train, scaled_train, length=BS,
batch_size=1)
```

In [ ]:

```
x1, y1 = generator[0]
x2, y2 = generator[1]
```

In [ ]:
```
# given data of last BS days it will predict the expected number of ca
ses for the (BS+1)th day

print(f'example 1: {x1}\n{y1}\n\n\nexample 2: {x2}\n{y2}')
```

```
example 1: [[[0.00000000e+00]
  [3.54362670e-06]
  [1.38523589e-05]
  [3.14989040e-05]
  [5.60895256e-05]
  [8.50470407e-05]
  [1.79937489e-04]
  [2.00984484e-04]
  [2.75006908e-04]
  [3.35642298e-04]
  [4.11275462e-04]
  [5.81262161e-04]
  [6.92331592e-04]
  [8.35902164e-04]
  [9.69951477e-04]
  [1.08302538e-03]
  [1.21163398e-03]
  [1.30949534e-03]
  [1.41795180e-03]
  [1.51133889e-03]]]
[[0.00158447]]


example 2: [[[3.54362670e-06]
  [1.38523589e-05]
  [3.14989040e-05]
  [5.60895256e-05]
  [8.50470407e-05]
  [1.79937489e-04]
  [2.00984484e-04]
  [2.75006908e-04]
  [3.35642298e-04]
  [4.11275462e-04]
  [5.81262161e-04]
  [6.92331592e-04]
  [8.35902164e-04]
  [9.69951477e-04]
  [1.08302538e-03]
  [1.21163398e-03]
  [1.30949534e-03]
  [1.41795180e-03]
  [1.51133889e-03]
  [1.58446646e-03]]]
[[0.00159943]]
```

In [ ]:
```
valGenerator = TimeseriesGenerator(scaled_test, scaled_test, length=BS
, batch_size=1)
```

3. Building and compiling the model

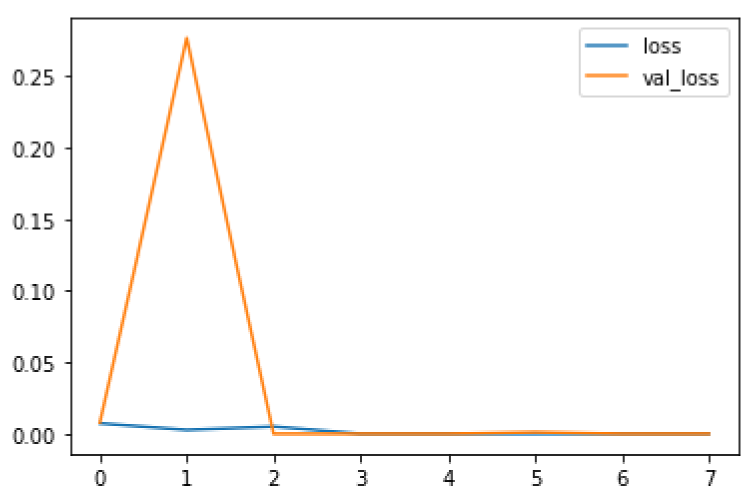
In [ ]:

```
# part 1: Building the model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM

n_features = 1
model = None
model = Sequential()

# a. Input layer: An LSTM layer with 128 cells and rectified linear un
it as activation.
model.add(LSTM(256, activation='relu', input_shape=(BS, n_features)))
# the input shape is 20x1
# b. Final layer: A dense layer with just one neuron.
model.add(Dense(1))


# ----------------------------------------------------------------------
-------------------------------------------------------
# part 2: compiling the model

from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(), loss='mse')
print(model.summary())


# ----------------------------------------------------------------------
-------------------------------------------------------
# part 3: training the model

EPOCHS = 25
# With early stopping

from tensorflow.keras.callbacks import EarlyStopping

earlyStop = EarlyStopping(monitor='val_loss', patience=4)
H = model.fit_generator(generator, epochs=EPOCHS, validation_data=valG
enerator, callbacks=[earlyStop])
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_6 (LSTM)                (None, 256)               264192    
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 257       
=================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/25
191/191 [==============================] - 10s 51ms/step - loss: 0.007
3 - val_loss: 0.0080
Epoch 2/25
191/191 [==============================] - 10s 50ms/step - loss: 0.002
7 - val_loss: 0.2766
Epoch 3/25
191/191 [==============================] - 9s 48ms/step - loss: 0.0051
- val_loss: 7.2416e-06
Epoch 4/25
191/191 [==============================] - 9s 48ms/step - loss: 3.4809
e-05 - val_loss: 5.8956e-06
Epoch 5/25
191/191 [==============================] - 9s 48ms/step - loss: 3.1669
e-05 - val_loss: 6.4470e-06
Epoch 6/25
191/191 [==============================] - 9s 48ms/step - loss: 2.9924
e-05 - val_loss: 0.0012
Epoch 7/25
191/191 [==============================] - 9s 49ms/step - loss: 4.6277
e-05 - val_loss: 9.6539e-06
Epoch 8/25
191/191 [==============================] - 10s 51ms/step - loss: 2.460
3e-05 - val_loss: 3.1035e-05
```

In [ ]:

```
losses = pd.DataFrame(H.history)
losses.plot();
```



From the above diagram we can clearly see that the training and validation **loss** is **significantly low** (lower the loss, the better).

Now let us try evaluating our trained model for **forecasting** the number of active cases for the **next 25 days**.

4. Evaluating the model

To predict the values for the last 25 days we have to get the confirmed cases for the previous 20 days, which is actually present in the train set.

We bassically grab the data of the last 20 days of the train set inorder to predict/forecast the result for the first day in the test set.

Then we append this result to a list and use our predicted value along with the last 19 values in the train set to forecast the result for the second day.

By this way we can forecast for all the other days in the test set.

Since we're using the previous predicted value inorder to predict the next value, there will be some noise in the forecasted value. For forecasting the values for few days will not give us a bad result but if we had to forecast for the next 3 or 4 months, the result will be too noisy.

In [ ]:

```
# last 20 rows from scaled_train

first_eval_batch = scaled_train[-BS:]
print(first_eval_batch)
```

```
[[0.79313202]
 [0.80421033]
 [0.81411545]
 [0.82297194]
 [0.83429314]
 [0.8458919 ]
 [0.85823514]
 [0.87037238]
 [0.88167992]
 [0.89138677]
 [0.90130262]
 [0.91209941]
 [0.92402855]
 [0.93610895]
 [0.9493043 ]
 [0.95990888]
 [0.96902201]
 [0.97776321]
 [0.98854696]
 [1.        ]]
```

In [ ]:
```
# array to store the forcasted results
test_predictions = []

current_batch = first_eval_batch.reshape((1, BS, n_features))

count=1

for i in range(len(test)):

    current_pred = model.predict(current_batch)[0]
    test_predictions.append(current_pred)

#     print(f'Result {count}:')
#     print(f'current batch:\n{current_batch},\n prediction: {current_
pred}')

    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],a
xis=1)
    count += 1

#     print('\n\n')
```

In [ ]:
```
true_predictions = scaler.inverse_transform(test_predictions)
print(true_predictions)
```

```
[[28305699.62691879]
 [28607258.00246334]
 [28908496.6589222 ]
 [29209045.92110252]
 [29508779.23353291]
 [29807303.60817051]
 [30104465.84628677]
 [30400059.46263885]
 [30694037.83152676]
 [30986537.49964333]
 [31277708.33530998]
 [31567427.11346245]
 [31855617.23473644]
 [32141932.33678913]
 [32426475.66224194]
 [32709170.61173058]
 [32990253.64416218]
 [33269847.98460102]
 [33547857.05123997]
 [33824227.55756474]
 [34098902.8866539 ]
 [34371703.19652176]
 [34642705.08653259]
 [34911951.85197926]
 [35179486.7881546 ]]
```

In [ ]:
```
test['Predictions'] = true_predictions.astype(int)
test
```

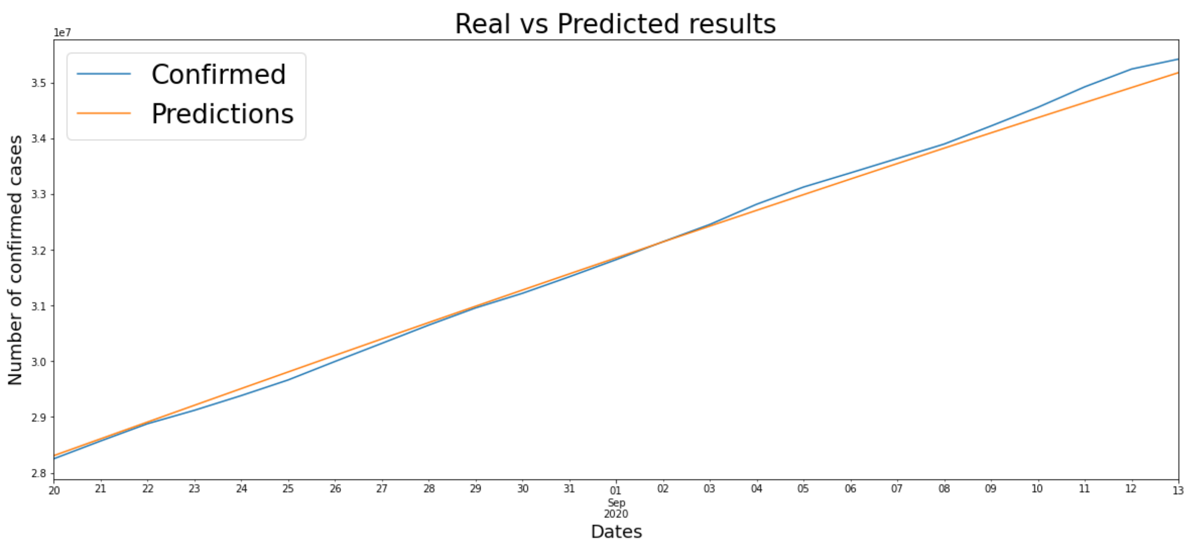
```
C:\Users\Python\anaconda3\lib\site-packages\ipykernel_launcher.py:1: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

Out[ ]:

	Confirmed	Predictions
Dates		
2020-08-20	28249176	28305699
2020-08-21	28568520	28607258
2020-08-22	28878633	28908496
2020-08-23	29119151	29209045
2020-08-24	29383390	29508779
2020-08-25	29663595	29807303
2020-08-26	29995565	30104465
2020-08-27	30320729	30400059
2020-08-28	30648389	30694037
2020-08-29	30957550	30986537
2020-08-30	31219140	31277708
2020-08-31	31515354	31567427
2020-09-01	31823482	31855617
2020-09-02	32144920	32141932
2020-09-03	32454872	32426475
2020-09-04	32818489	32709170
2020-09-05	33126517	32990253
2020-09-06	33380210	33269847
2020-09-07	33638382	33547857
2020-09-08	33897751	33824227
2020-09-09	34222892	34098902
2020-09-10	34557534	34371703
2020-09-11	34924974	34642705
2020-09-12	35244068	34911951
2020-09-13	35422326	35179486

In [ ]:
```
test.plot(figsize=(20,8));
plt.title('Real vs Predicted results', fontsize=26)
plt.xlabel('Dates', fontsize=18)
plt.ylabel('Number of confirmed cases', fontsize=18)
plt.legend(loc=2, prop={'size': 26})
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x237500aac88>
```



In [ ]:
```
from sklearn.metrics import r2_score, mean_squared_error

mse = mean_squared_error(test.Confirmed, test.Predictions)
# length = len(test)
rmse = np.sqrt(mse)

r2s = r2_score(test.Confirmed, test.Predictions)

print(f'RMSE = {rmse}; r2 score = {r2s}')
```

```
RMSE = 131797.31945756712; r2 score = 0.9963465838706869
```

At the first glance we might think the model's rmse is bad, but when we look the **coefficient of determination** (ie r2) we see that it has an **99.6% good fit**. The results are pretty good.

Note: We use the r^2 value along with the RMSE because it has a fixed range from 0 to 1.

A value of 0 indicates that the response variable cannot be explained by the predictor variable at all. A value of 1 indicates that the response variable can be perfectly explained without error by the predictor variable.

Basically if it is closer to 1, our model is good.

5. Forecasting the number of confirmed cases for the next N days

In [ ]:
```
from pandas.tseries.offsets import DateOffset

ndays=60
last_date = days.index[-1]
future_dates = [last_date+DateOffset(days=x) for x in range(1, ndays+1
)]
```

In [ ]:
```
days_copy = days.copy()
days_copy.tail()
```

Out[ ]:

	Confirmed
Dates	
2020-09-09	34222892
2020-09-10	34557534
2020-09-11	34924974
2020-09-12	35244068
2020-09-13	35422326

In [ ]:
```
# get the scaled data for the last BS days in our dataset

last_index = len(days)
last_n_days = scaler.transform(days_copy.iloc[-BS:last_index,:])
last_n_days
```

Out[ ]:
```
array([[1.06176502],
       [1.07364762],
       [1.08528661],
       [1.09701494],
       [1.10808112],
       [1.11744452],
       [1.12804727],
       [1.13907647],
       [1.15058209],
       [1.16167657],
       [1.17469196],
       [1.18571757],
       [1.19479831],
       [1.20403938],
       [1.21332328],
       [1.22496145],
       [1.2369397 ],
       [1.25009192],
       [1.26151364],
       [1.26789424]])
```

In [ ]:
```
# array to store the forcasted results for the next ndays
test_predictions_ndays = []

current_batch = last_n_days.reshape((1, BS, n_features))

count=1

for i in range(ndays):

    current_pred = model.predict(current_batch)[0]
    test_predictions_ndays.append(current_pred)

    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],a
xis=1)
    count += 1
```

In [ ]:
```
true_predictions_ndays = scaler.inverse_transform(test_predictions_nda
ys).astype(int)
future_dates_df = pd.DataFrame(index=future_dates[:],columns=days.colu
mns, data=true_predictions_ndays)
future_df = pd.concat([days_copy,future_dates_df])
```

In [ ]:
```
future_df.head()
```

Out[ ]:

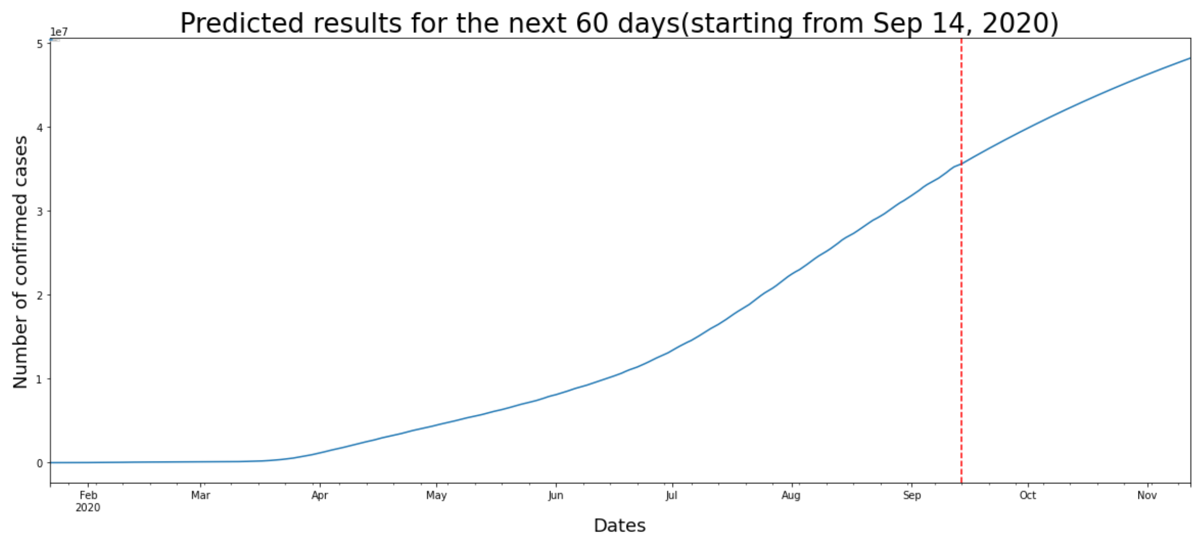
	Confirmed
2020-01-22	556
2020-01-23	655
2020-01-24	943
2020-01-25	1436
2020-01-26	2123

In [ ]:
```
future_df.tail()
```

Out[ ]:

	Confirmed
2020-11-08	47493597
2020-11-09	47665822
2020-11-10	47836692
2020-11-11	48006216
2020-11-12	48174409

In [ ]:
```
future_df.plot(figsize=(20,8));

plt.axvline("2020-09-14", color="red", linestyle="--");

plt.title(f'Predicted results for the next {ndays} days(starting from
 Sep 14, 2020)', fontsize=26);
plt.xlabel('Dates', fontsize=18);
plt.ylabel('Number of confirmed cases', fontsize=18);
plt.legend(loc=2, prop={'size': 0});
```



From the above graph, we can tell that the values are most probably fairly accurate. Since we dont have the actual values we can never be too sure but it more or less looks like a good forecast.

We then save our models architecture and weights.

In [ ]:
```
name = 'model1_weights.h5'

model.save_weights(name)
model.save('model1_architecture.h5')
```

CB.EN.U4CSE17337

India Covid Data

In [ ]:

```
indiadaily=pd.read_csv('https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/Arnab/COVID-DataTimeSeries(India)/nation_level_daily.csv')
indiadaily['Date'] = indiadaily['Date'].str.slice(0,6) +' 2020'
indiadaily['Date']= indiadaily['Date'].astype(str).apply(lambda x: datetime.datetime.strptime(x,'%d %b %Y'))
indiadaily['Date']= pd.to_datetime(indiadaily['Date'])
indiadaily.rename(columns={'Daily Confirmed':'casesdaily',
                   'Total Confirmed':'cases',
                   'Daily Recovered':'recoverydaily',
                   'Total Recovered':'recovery',
                   'Daily Deceased' :'deathdaily',
                   'Total Deceased':'death'},inplace=True)
print(indiadaily.dtypes)
indiadaily.tail()
```

```
Date             datetime64[ns]
casesdaily                int64
cases                     int64
recoverydaily             int64
recovery                  int64
deathdaily                int64
death                     int64
dtype: object
```

Out[ ]:

	Date	casesdaily	cases	recoverydaily	recovery	deathdaily	death
185	2020-08-02	52672	1804857	40355	1187261	760	38180
186	2020-08-03	50488	1855345	43070	1230331	806	38986
187	2020-08-04	51282	1906627	51220	1281551	849	39835
188	2020-08-05	56626	1963253	45583	1327134	919	40754
189	2020-08-06	62170	2025423	50141	1377275	899	41653

In [ ]:
```
complete = pd.read_csv('https://raw.githubusercontent.com/Ashwin1999/COVID-19-Data-Mining/Arnab/COVID-DataTimeSeries(India)/complete.csv')
complete['Date']= pd.to_datetime(complete['Date'])
complete.rename(columns={'Name of State / UT':'State'},inplace=True)
complete['Death'] = complete['Death'].str.extract('(\d+)', expand=False)
complete['Death'] = complete['Death'].astype(int)
print(complete.dtypes)
complete.head()
```

```
Date                          datetime64[ns]
State                                 object
Latitude                             float64
Longitude                            float64
Total Confirmed cases                float64
Death                                  int64
Cured/Discharged/Migrated            float64
New cases                              int64
New deaths                             int64
New recovered                          int64
dtype: object
```

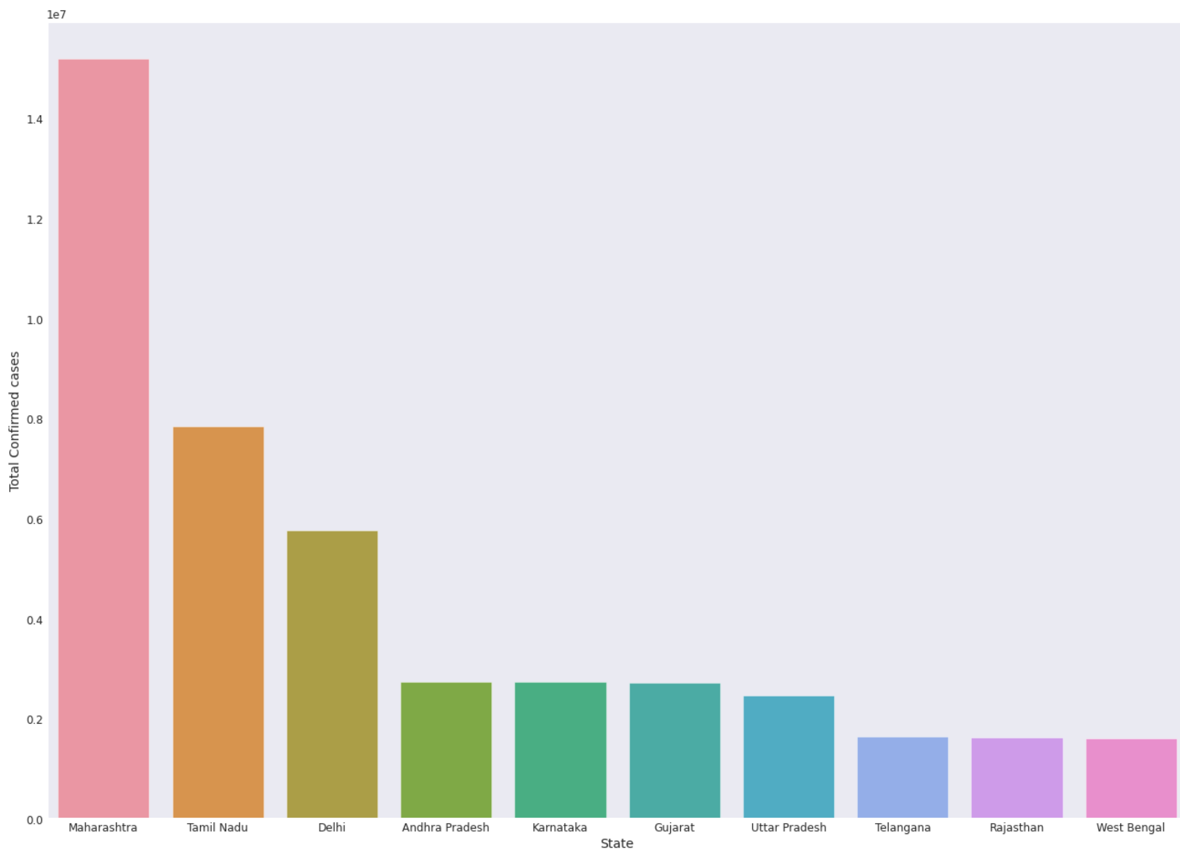
Out[ ]:

	Date	State	Latitude	Longitude	Total Confirmed cases	Death	Cured/Discharged/Migrated	New cases	New deaths
0	2020-01-30	Kerala	10.8505	76.2711	1.0	0	0.0	0	0
1	2020-01-31	Kerala	10.8505	76.2711	1.0	0	0.0	0	0
2	2020-02-01	Kerala	10.8505	76.2711	2.0	0	0.0	1	0
3	2020-02-02	Kerala	10.8505	76.2711	3.0	0	0.0	1	0
4	2020-02-03	Kerala	10.8505	76.2711	3.0	0	0.0	0	0

In [ ]:

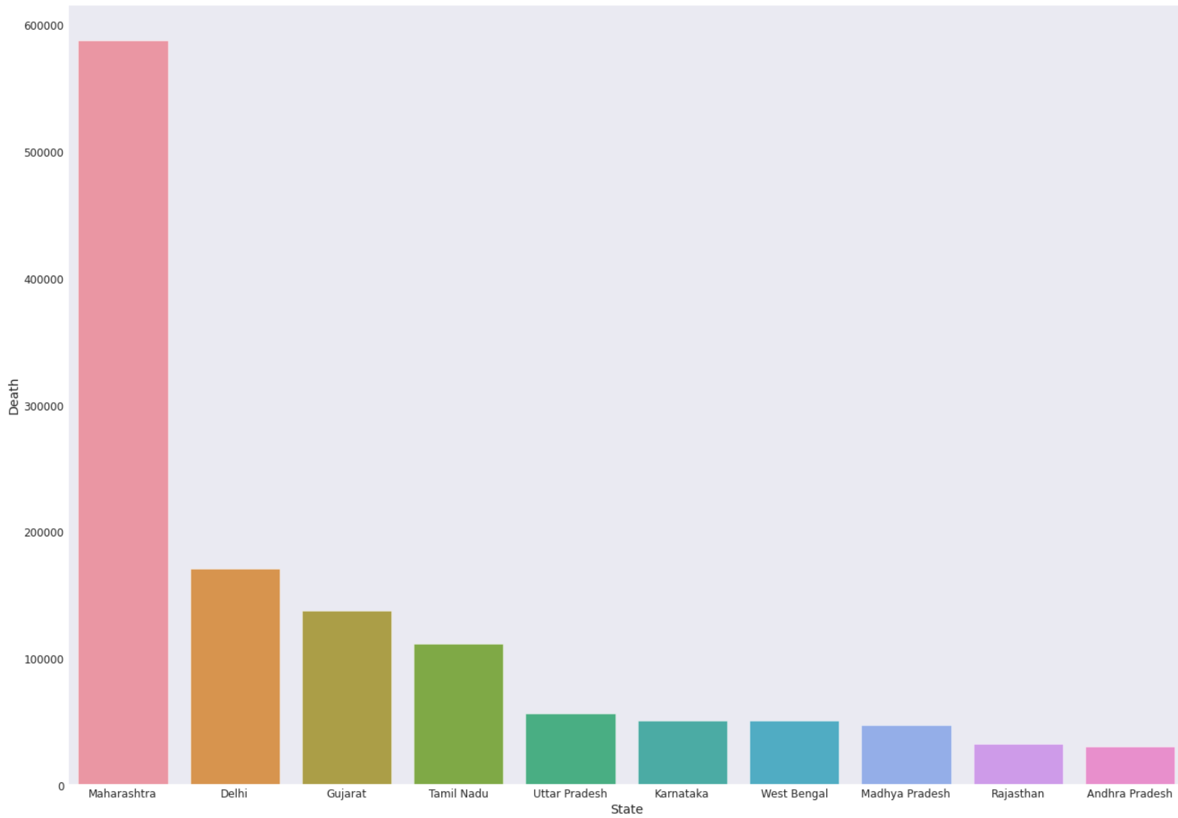
```
plt.figure(figsize=(20,15))
statewisedata=complete.groupby('State').sum()
statewisedata.reset_index(inplace=True)
statewisedata.sort_values(by='Total Confirmed cases',inplace=True,ascending=False)
sns.barplot(y='Total Confirmed cases',x='State',data=statewisedata.iloc[:10])
# # statewisedata.plot.bar(rot=0,figsize=(20,10))
# # statewisedata.plot.bar(rot=0,figsize=(20,10))

# plt.xticks(rotation=70)
# statewisedata.head()
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f95abdcef28>



In [ ]:
```
plt.figure(figsize=(20,15))
statewisedata=complete.groupby('State').sum()
statewisedata.reset_index(inplace=True)
statewisedata.sort_values(by='Death',inplace=True,ascending=False)
sns.barplot(y='Death',x='State',data=statewisedata.iloc[:10])
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f95abd18c88>

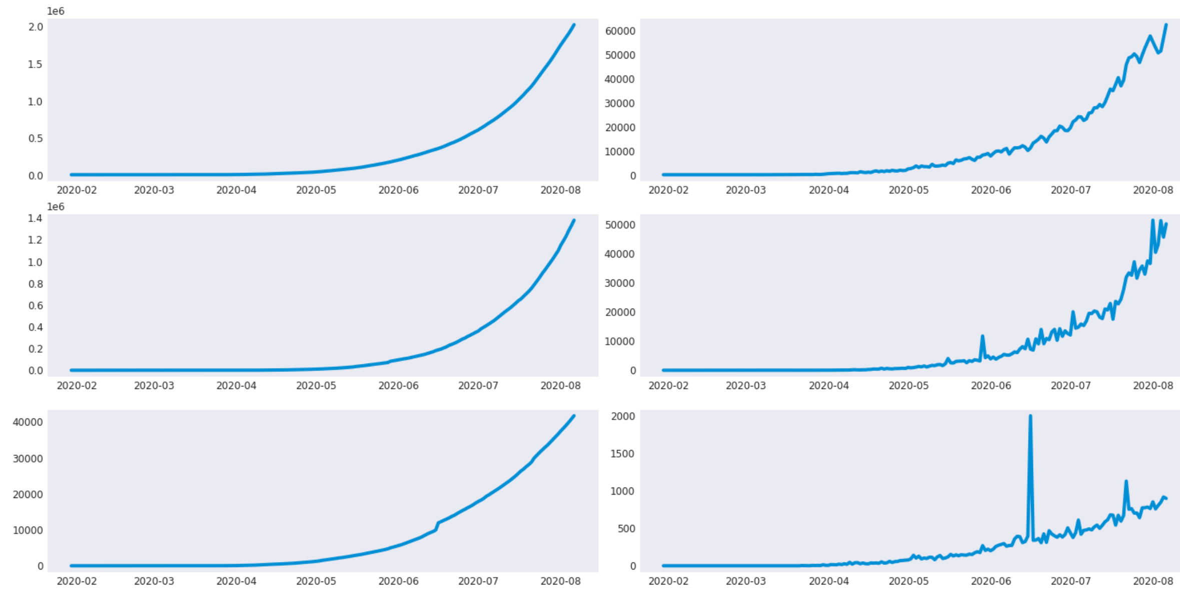


In [ ]:

```
fig,ax=plt.subplots(3,2,figsize=(20,10))
fig.tight_layout()
ax[0,0].plot(indiadaily['Date'],indiadaily['cases'])
ax[0,1].plot(indiadaily['Date'],indiadaily['casesdaily'])
ax[1,0].plot(indiadaily['Date'],indiadaily['recovery'])
ax[1,1].plot(indiadaily['Date'],indiadaily['recoverydaily'])
ax[2,0].plot(indiadaily['Date'],indiadaily['death'])
ax[2,1].plot(indiadaily['Date'],indiadaily['deathdaily'])

# plt.plot(indiadaily['Date'],indiadaily['Total Recovered'])
# plt.plot(indiadaily['Date'],indiadaily['Total Deceased'])
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f95abeb0668>]
```



In [ ]:

```
indiatesting=pd.read_csv('https://raw.githubusercontent.com/Ashwin199
9/COVID-19-Data-Mining/Arnab/COVID-DataTimeSeries(India)/tests_day_wis
e.csv')
indiatesting=indiatesting[['Tested As Of','Total Samples Tested']]
indiatesting['Tested As Of']=indiatesting['Tested As Of'].str.replace(
'/','-')
indiatesting.rename(columns={'Tested As Of':'Date','Total Samples Test
ed':'tests'},inplace=True)
indiatesting.dropna(how='any',axis=0,inplace=True)
indiatesting['Date']= indiatesting['Date'].astype(str).apply(lambda x:
datetime.datetime.strptime(x,'%d-%m-%Y'))
indiatesting['Date']= pd.to_datetime(indiatesting['Date'])
indiatesting.head()
```

Out[ ]:

	Date	tests
0	2020-03-13	6500.0
1	2020-03-18	13125.0
2	2020-03-19	13316.0
3	2020-03-19	14175.0
4	2020-03-20	14376.0

In [ ]:
```
indiacovidfinal = indiadaily.merge(indiatesting,how='inner',on='Date')

indiacovidfinal = indiacovidfinal[['Date','casesdaily', 'recoverydail
y', 'recovery', 'deathdaily','death', 'tests', 'cases']]
indiacovidfinal.tail()
```

Out[ ]:

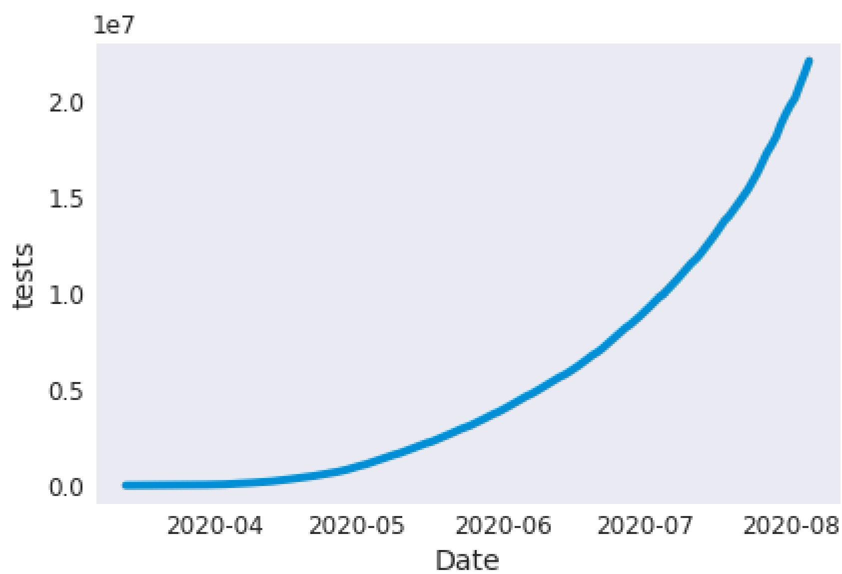
	Date	casesdaily	recoverydaily	recovery	deathdaily	death	tests	cases
141	2020-08-01	55117	51368	1146906	854	37420	19821831.0	1752185
142	2020-08-02	52672	40355	1187261	760	38180	20202858.0	1804857
143	2020-08-03	50488	43070	1230331	806	38986	20864750.0	1855345
144	2020-08-04	51282	51220	1281551	849	39835	21484402.0	1906627
145	2020-08-05	56626	45583	1327134	919	40754	22149351.0	1963253

In [ ]:
```
indiacovidfinal.columns
```

Out[ ]:
```
Index(['Date', 'casesdaily', 'recoverydaily', 'recovery', 'deathdail
y',
       'death', 'tests', 'cases'],
      dtype='object')
```

In [ ]:
```
sns.lineplot(x='Date',y='tests',data=indiacovidfinal)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95b61f1e80>
```



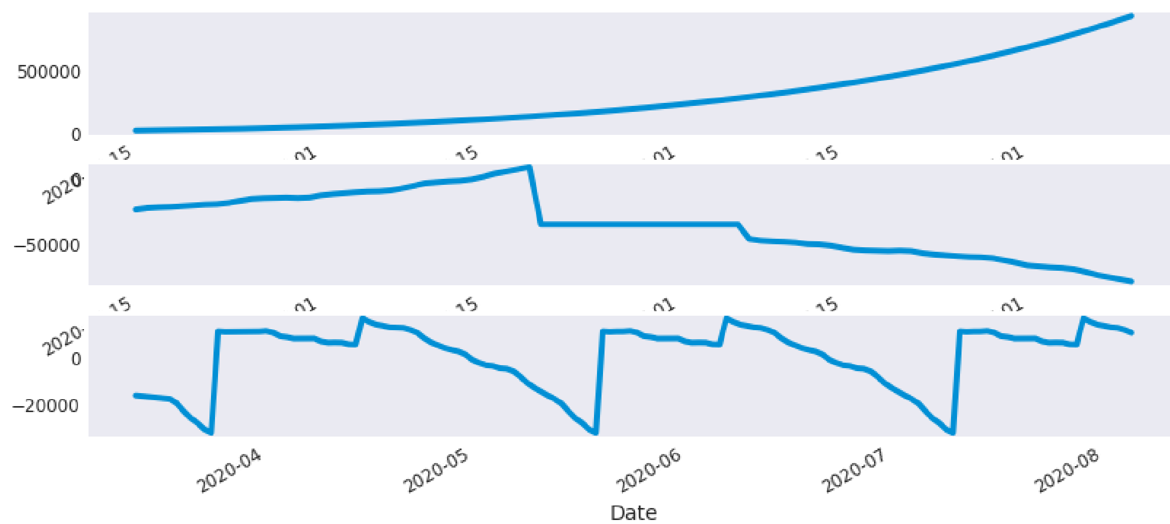
EDA

In [ ]:
```
indiacovidfinal=indiacovidfinal.groupby('Date').sum()
indiacovidfinal.sort_index(inplace= True)
indiacovidfinal.head()
```

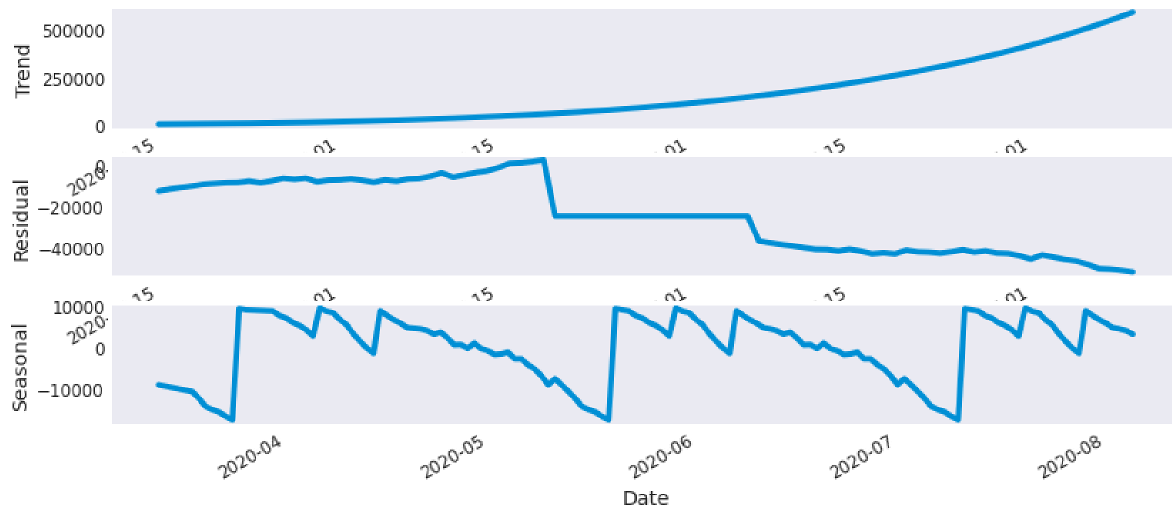
Out[ ]:

	casesdaily	recoverydaily	recovery	deathdaily	death	tests	cases
Date							
2020-03-13	10	6	10	0	1	6500.0	91
2020-03-18	25	0	15	0	3	13125.0	171
2020-03-19	54	10	40	2	8	27491.0	396
2020-03-20	116	6	46	0	8	29780.0	512
2020-03-21	156	0	46	0	8	32612.0	668

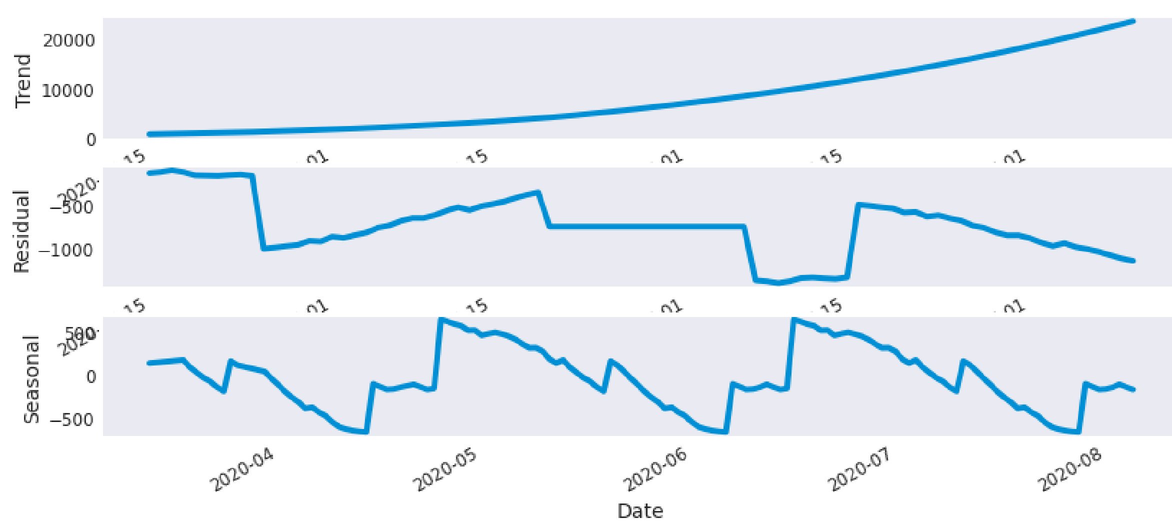
In [ ]:
```
toplot=indiacovidfinal['cases']
result = seasonal_decompose(toplot, model='additive',freq=52)
fig, (ax1,ax2,ax3) = plt.subplots(3,1, figsize=(12,6))
result.trend.plot(ax=ax1)
result.resid.plot(ax=ax2)
result.seasonal.plot(ax=ax3)
plt.show()
```



```
In [ ]: toplot=indiacovidfinal['recovery']
        result = seasonal_decompose(toplot, model='additive',freq=52)
        fig, (ax1,ax2,ax3) = plt.subplots(3,1, figsize=(12,6))
        result.trend.plot(ax=ax1)
        result.resid.plot(ax=ax2)
        result.seasonal.plot(ax=ax3)
        ax1.set_ylabel("Trend")
        ax2.set_ylabel("Residual")
        ax3.set_ylabel("Seasonal")
        plt.show()
```




```
In [ ]: toplot=indiacovidfinal['death']
        result = seasonal_decompose(toplot, model='additive',freq=52)
        fig, (ax1,ax2,ax3) = plt.subplots(3,1, figsize=(12,6))
        result.trend.plot(ax=ax1)
        result.resid.plot(ax=ax2)
        result.seasonal.plot(ax=ax3)
        ax1.set_ylabel("Trend")
        ax2.set_ylabel("Residual")
        ax3.set_ylabel("Seasonal")
        plt.show()
```



Stationarity Testing

In [ ]:
```
result = adfuller(indiacovidfinal['cases'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: 3.495026
p-value: 1.000000
        1%: -3.483
        5%: -2.885
        10%: -2.579
```

In [ ]:
```
result = adfuller(indiacovidfinal['death'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: 18.247325
p-value: 1.000000
        1%: -3.479
        5%: -2.883
        10%: -2.578
```

LSTM

In [ ]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(indiacovidfinal)
```

In [ ]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.80)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset
),:]
print(len(train), len(test))
print(train.shape)
print(test.shape)
```

```
110 28
(110, 7)
(28, 7)
```

In [ ]:
```
trainX, trainY = train[:,0:6],train[:,6]
testX, testY = test[:,0:6],test[:,6]
```

In [ ]:
```
# trainY = np.insert(trainY, 0, 0)
# define generator
n_input = 1
generator = TimeseriesGenerator(trainX, trainY, length=n_input, batch_
size=1)
generator_test = TimeseriesGenerator(testX, testY, length=n_input, bat
ch_size=1)
```

In [ ]:
```
n_features = 6
# trainX = trainX.reshape((trainX.shape[0], trainX.shape[1], 1))
```

In [ ]:
```
num_epochs=200
lstm_model = Sequential()
lstm_model.add(LSTM(58, activation='relu', input_shape=(n_input, n_fea
tures)))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse',metrics='mse')
lstm_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 58)                15080
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 59
=================================================================
Total params: 15,139
Trainable params: 15,139
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
lstm_model.fit_generator(generator=generator,epochs=num_epochs,validat
ion_data=generator_test)
```

```
Epoch 1/200
109/109 [==============================] - 0s 4ms/step - loss: 0.0089
- mse: 0.0089 - val_loss: 0.1347 - val_mse: 0.1347
Epoch 2/200
109/109 [==============================] - 0s 2ms/step - loss: 0.0012
- mse: 0.0012 - val_loss: 0.0262 - val_mse: 0.0262
Epoch 3/200
109/109 [==============================] - 0s 3ms/step - loss: 1.2763e
-04 - mse: 1.2763e-04 - val_loss: 0.0073 - val_mse: 0.0073
Epoch 4/200
109/109 [==============================] - 0s 3ms/step - loss: 7.9202e
-05 - mse: 7.9202e-05 - val_loss: 0.0078 - val_mse: 0.0078
Epoch 5/200
109/109 [==============================] - 0s 3ms/step - loss: 6.5295e
-05 - mse: 6.5295e-05 - val_loss: 0.0062 - val_mse: 0.0062
Epoch 6/200
109/109 [==============================] - 0s 2ms/step - loss: 3.7721e
-05 - mse: 3.7721e-05 - val_loss: 0.0041 - val_mse: 0.0041
Epoch 7/200
109/109 [==============================] - 0s 2ms/step - loss: 2.5681e
-05 - mse: 2.5681e-05 - val_loss: 0.0036 - val_mse: 0.0036
Epoch 8/200
109/109 [==============================] - 0s 2ms/step - loss: 2.2859e
-05 - mse: 2.2859e-05 - val_loss: 0.0033 - val_mse: 0.0033
Epoch 9/200
109/109 [==============================] - 0s 3ms/step - loss: 1.7094e
-05 - mse: 1.7094e-05 - val_loss: 0.0030 - val_mse: 0.0030
Epoch 10/200
109/109 [==============================] - 0s 3ms/step - loss: 1.5003e
-05 - mse: 1.5003e-05 - val_loss: 0.0031 - val_mse: 0.0031
Epoch 11/200
109/109 [==============================] - 0s 3ms/step - loss: 1.7959e
-05 - mse: 1.7959e-05 - val_loss: 0.0042 - val_mse: 0.0042
Epoch 12/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4118e
-05 - mse: 1.4118e-05 - val_loss: 0.0019 - val_mse: 0.0019
Epoch 13/200
109/109 [==============================] - 0s 3ms/step - loss: 1.5426e
-05 - mse: 1.5426e-05 - val_loss: 0.0025 - val_mse: 0.0025
Epoch 14/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3942e
-05 - mse: 1.3942e-05 - val_loss: 0.0030 - val_mse: 0.0030
Epoch 15/200
109/109 [==============================] - 0s 3ms/step - loss: 1.5358e
-05 - mse: 1.5358e-05 - val_loss: 0.0025 - val_mse: 0.0025
Epoch 16/200
109/109 [==============================] - 0s 3ms/step - loss: 1.4720e
-05 - mse: 1.4720e-05 - val_loss: 0.0027 - val_mse: 0.0027
Epoch 17/200
109/109 [==============================] - 0s 3ms/step - loss: 1.5688e
-05 - mse: 1.5688e-05 - val_loss: 0.0030 - val_mse: 0.0030
Epoch 18/200
109/109 [==============================] - 0s 3ms/step - loss: 1.5999e
-05 - mse: 1.5999e-05 - val_loss: 0.0034 - val_mse: 0.0034
Epoch 19/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1944e
-05 - mse: 1.1944e-05 - val_loss: 0.0030 - val_mse: 0.0030
```

```
Epoch 20/200
109/109 [==============================] - 0s 3ms/step - loss: 1.2649e
-05 - mse: 1.2649e-05 - val_loss: 0.0018 - val_mse: 0.0018
Epoch 21/200
109/109 [==============================] - 0s 3ms/step - loss: 1.9627e
-05 - mse: 1.9627e-05 - val_loss: 0.0028 - val_mse: 0.0028
Epoch 22/200
109/109 [==============================] - 0s 3ms/step - loss: 1.0693e
-05 - mse: 1.0693e-05 - val_loss: 0.0022 - val_mse: 0.0022
Epoch 23/200
109/109 [==============================] - 0s 3ms/step - loss: 1.4887e
-05 - mse: 1.4887e-05 - val_loss: 0.0033 - val_mse: 0.0033
Epoch 24/200
109/109 [==============================] - 0s 3ms/step - loss: 1.3550e
-05 - mse: 1.3550e-05 - val_loss: 0.0029 - val_mse: 0.0029
Epoch 25/200
109/109 [==============================] - 0s 3ms/step - loss: 1.3508e
-05 - mse: 1.3508e-05 - val_loss: 0.0033 - val_mse: 0.0033
Epoch 26/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2451e
-05 - mse: 1.2451e-05 - val_loss: 0.0025 - val_mse: 0.0025
Epoch 27/200
109/109 [==============================] - 0s 3ms/step - loss: 1.1521e
-05 - mse: 1.1521e-05 - val_loss: 0.0020 - val_mse: 0.0020
Epoch 28/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4624e
-05 - mse: 1.4624e-05 - val_loss: 0.0014 - val_mse: 0.0014
Epoch 29/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3006e
-05 - mse: 1.3006e-05 - val_loss: 0.0027 - val_mse: 0.0027
Epoch 30/200
109/109 [==============================] - 0s 2ms/step - loss: 2.8967e
-05 - mse: 2.8967e-05 - val_loss: 0.0022 - val_mse: 0.0022
Epoch 31/200
109/109 [==============================] - 0s 3ms/step - loss: 2.2176e
-05 - mse: 2.2176e-05 - val_loss: 0.0026 - val_mse: 0.0026
Epoch 32/200
109/109 [==============================] - 0s 3ms/step - loss: 2.7752e
-05 - mse: 2.7752e-05 - val_loss: 0.0059 - val_mse: 0.0059
Epoch 33/200
109/109 [==============================] - 0s 2ms/step - loss: 2.3192e
-05 - mse: 2.3192e-05 - val_loss: 0.0025 - val_mse: 0.0025
Epoch 34/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1733e
-05 - mse: 1.1733e-05 - val_loss: 0.0023 - val_mse: 0.0023
Epoch 35/200
109/109 [==============================] - 0s 2ms/step - loss: 9.6113e
-06 - mse: 9.6113e-06 - val_loss: 0.0023 - val_mse: 0.0023
Epoch 36/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2512e
-05 - mse: 1.2512e-05 - val_loss: 0.0024 - val_mse: 0.0024
Epoch 37/200
109/109 [==============================] - 0s 2ms/step - loss: 1.6163e
-05 - mse: 1.6163e-05 - val_loss: 0.0026 - val_mse: 0.0026
Epoch 38/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2634e
-05 - mse: 1.2634e-05 - val_loss: 0.0021 - val_mse: 0.0021
```

```
Epoch 39/200
109/109 [==============================] - 0s 2ms/step - loss: 9.9263e
-06 - mse: 9.9263e-06 - val_loss: 0.0017 - val_mse: 0.0017
Epoch 40/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1249e
-05 - mse: 1.1249e-05 - val_loss: 0.0021 - val_mse: 0.0021
Epoch 41/200
109/109 [==============================] - 0s 2ms/step - loss: 2.2480e
-05 - mse: 2.2480e-05 - val_loss: 0.0023 - val_mse: 0.0023
Epoch 42/200
109/109 [==============================] - 0s 2ms/step - loss: 1.8776e
-05 - mse: 1.8776e-05 - val_loss: 0.0017 - val_mse: 0.0017
Epoch 43/200
109/109 [==============================] - 0s 2ms/step - loss: 9.8605e
-06 - mse: 9.8605e-06 - val_loss: 0.0018 - val_mse: 0.0018
Epoch 44/200
109/109 [==============================] - 0s 2ms/step - loss: 8.2857e
-06 - mse: 8.2857e-06 - val_loss: 0.0027 - val_mse: 0.0027
Epoch 45/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2072e
-05 - mse: 1.2072e-05 - val_loss: 0.0021 - val_mse: 0.0021
Epoch 46/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4138e
-05 - mse: 1.4138e-05 - val_loss: 9.4150e-04 - val_mse: 9.4150e-04
Epoch 47/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3232e
-05 - mse: 1.3232e-05 - val_loss: 0.0019 - val_mse: 0.0019
Epoch 48/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2061e
-05 - mse: 1.2061e-05 - val_loss: 0.0022 - val_mse: 0.0022
Epoch 49/200
109/109 [==============================] - 0s 2ms/step - loss: 1.7485e
-05 - mse: 1.7485e-05 - val_loss: 0.0011 - val_mse: 0.0011
Epoch 50/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4452e
-05 - mse: 1.4452e-05 - val_loss: 0.0016 - val_mse: 0.0016
Epoch 51/200
109/109 [==============================] - 0s 2ms/step - loss: 8.2511e
-06 - mse: 8.2511e-06 - val_loss: 0.0016 - val_mse: 0.0016
Epoch 52/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2473e
-05 - mse: 1.2473e-05 - val_loss: 0.0011 - val_mse: 0.0011
Epoch 53/200
109/109 [==============================] - 0s 2ms/step - loss: 6.0013e
-06 - mse: 6.0013e-06 - val_loss: 0.0019 - val_mse: 0.0019
Epoch 54/200
109/109 [==============================] - 0s 2ms/step - loss: 1.0097e
-05 - mse: 1.0097e-05 - val_loss: 0.0028 - val_mse: 0.0028
Epoch 55/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2259e
-05 - mse: 1.2259e-05 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 56/200
109/109 [==============================] - 0s 2ms/step - loss: 6.5523e
-06 - mse: 6.5523e-06 - val_loss: 0.0011 - val_mse: 0.0011
Epoch 57/200
109/109 [==============================] - 0s 2ms/step - loss: 6.2658e
-06 - mse: 6.2658e-06 - val_loss: 8.1490e-04 - val_mse: 8.1490e-04
```

```
Epoch 58/200
109/109 [==============================] - 0s 2ms/step - loss: 1.8836e
-05 - mse: 1.8836e-05 - val_loss: 7.0468e-04 - val_mse: 7.0468e-04
Epoch 59/200
109/109 [==============================] - 0s 2ms/step - loss: 3.1245e
-05 - mse: 3.1245e-05 - val_loss: 0.0026 - val_mse: 0.0026
Epoch 60/200
109/109 [==============================] - 0s 2ms/step - loss: 1.9147e
-05 - mse: 1.9147e-05 - val_loss: 1.7422e-04 - val_mse: 1.7422e-04
Epoch 61/200
109/109 [==============================] - 0s 2ms/step - loss: 1.5888e
-05 - mse: 1.5888e-05 - val_loss: 0.0022 - val_mse: 0.0022
Epoch 62/200
109/109 [==============================] - 0s 2ms/step - loss: 1.5801e
-05 - mse: 1.5801e-05 - val_loss: 0.0013 - val_mse: 0.0013
Epoch 63/200
109/109 [==============================] - 0s 2ms/step - loss: 6.4083e
-06 - mse: 6.4083e-06 - val_loss: 5.2192e-04 - val_mse: 5.2192e-04
Epoch 64/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3878e
-05 - mse: 1.3878e-05 - val_loss: 0.0012 - val_mse: 0.0012
Epoch 65/200
109/109 [==============================] - 0s 2ms/step - loss: 5.6791e
-06 - mse: 5.6791e-06 - val_loss: 0.0012 - val_mse: 0.0012
Epoch 66/200
109/109 [==============================] - 0s 2ms/step - loss: 7.8095e
-06 - mse: 7.8095e-06 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 67/200
109/109 [==============================] - 0s 2ms/step - loss: 5.5915e
-06 - mse: 5.5915e-06 - val_loss: 0.0021 - val_mse: 0.0021
Epoch 68/200
109/109 [==============================] - 0s 2ms/step - loss: 5.1312e
-06 - mse: 5.1312e-06 - val_loss: 0.0014 - val_mse: 0.0014
Epoch 69/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3389e
-05 - mse: 1.3389e-05 - val_loss: 1.5407e-04 - val_mse: 1.5407e-04
Epoch 70/200
109/109 [==============================] - 0s 2ms/step - loss: 2.8109e
-05 - mse: 2.8109e-05 - val_loss: 0.0016 - val_mse: 0.0016
Epoch 71/200
109/109 [==============================] - 0s 2ms/step - loss: 7.6130e
-06 - mse: 7.6130e-06 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 72/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2991e
-05 - mse: 1.2991e-05 - val_loss: 0.0014 - val_mse: 0.0014
Epoch 73/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6311e
-05 - mse: 2.6311e-05 - val_loss: 9.0710e-04 - val_mse: 9.0710e-04
Epoch 74/200
109/109 [==============================] - 0s 2ms/step - loss: 7.4828e
-06 - mse: 7.4828e-06 - val_loss: 8.3245e-04 - val_mse: 8.3245e-04
Epoch 75/200
109/109 [==============================] - 0s 2ms/step - loss: 1.0093e
-05 - mse: 1.0093e-05 - val_loss: 7.6227e-04 - val_mse: 7.6227e-04
Epoch 76/200
109/109 [==============================] - 0s 2ms/step - loss: 5.0660e
-06 - mse: 5.0660e-06 - val_loss: 1.9095e-04 - val_mse: 1.9095e-04
```

```
Epoch 77/200
109/109 [==============================] - 0s 2ms/step - loss: 2.8608e
-05 - mse: 2.8608e-05 - val_loss: 0.0012 - val_mse: 0.0012
Epoch 78/200
109/109 [==============================] - 0s 2ms/step - loss: 9.6321e
-06 - mse: 9.6321e-06 - val_loss: 8.8806e-04 - val_mse: 8.8806e-04
Epoch 79/200
109/109 [==============================] - 0s 2ms/step - loss: 6.1196e
-06 - mse: 6.1196e-06 - val_loss: 5.1513e-04 - val_mse: 5.1513e-04
Epoch 80/200
109/109 [==============================] - 0s 2ms/step - loss: 3.8470e
-06 - mse: 3.8470e-06 - val_loss: 7.0257e-04 - val_mse: 7.0257e-04
Epoch 81/200
109/109 [==============================] - 0s 2ms/step - loss: 4.8076e
-06 - mse: 4.8076e-06 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 82/200
109/109 [==============================] - 0s 2ms/step - loss: 5.1488e
-06 - mse: 5.1488e-06 - val_loss: 9.7043e-04 - val_mse: 9.7043e-04
Epoch 83/200
109/109 [==============================] - 0s 2ms/step - loss: 3.5571e
-06 - mse: 3.5571e-06 - val_loss: 0.0012 - val_mse: 0.0012
Epoch 84/200
109/109 [==============================] - 0s 2ms/step - loss: 3.5369e
-06 - mse: 3.5369e-06 - val_loss: 0.0013 - val_mse: 0.0013
Epoch 85/200
109/109 [==============================] - 0s 2ms/step - loss: 4.8388e
-06 - mse: 4.8388e-06 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 86/200
109/109 [==============================] - 0s 2ms/step - loss: 4.8716e
-06 - mse: 4.8716e-06 - val_loss: 0.0011 - val_mse: 0.0011
Epoch 87/200
109/109 [==============================] - 0s 2ms/step - loss: 9.1991e
-06 - mse: 9.1991e-06 - val_loss: 9.6578e-04 - val_mse: 9.6578e-04
Epoch 88/200
109/109 [==============================] - 0s 2ms/step - loss: 6.7693e
-06 - mse: 6.7693e-06 - val_loss: 7.6881e-04 - val_mse: 7.6881e-04
Epoch 89/200
109/109 [==============================] - 0s 2ms/step - loss: 7.0961e
-06 - mse: 7.0961e-06 - val_loss: 3.0893e-04 - val_mse: 3.0893e-04
Epoch 90/200
109/109 [==============================] - 0s 2ms/step - loss: 1.0928e
-05 - mse: 1.0928e-05 - val_loss: 2.5107e-04 - val_mse: 2.5107e-04
Epoch 91/200
109/109 [==============================] - 0s 2ms/step - loss: 1.0418e
-05 - mse: 1.0418e-05 - val_loss: 6.0148e-04 - val_mse: 6.0148e-04
Epoch 92/200
109/109 [==============================] - 0s 2ms/step - loss: 6.1147e
-06 - mse: 6.1147e-06 - val_loss: 7.4256e-04 - val_mse: 7.4256e-04
Epoch 93/200
109/109 [==============================] - 0s 2ms/step - loss: 3.5072e
-06 - mse: 3.5072e-06 - val_loss: 9.7091e-04 - val_mse: 9.7091e-04
Epoch 94/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4731e
-05 - mse: 1.4731e-05 - val_loss: 0.0014 - val_mse: 0.0014
Epoch 95/200
109/109 [==============================] - 0s 2ms/step - loss: 3.1506e
-05 - mse: 3.1506e-05 - val_loss: 8.2097e-04 - val_mse: 8.2097e-04
```

```
Epoch 96/200
109/109 [==============================] - 0s 2ms/step - loss: 5.3294e
-05 - mse: 5.3294e-05 - val_loss: 1.8721e-04 - val_mse: 1.8721e-04
Epoch 97/200
109/109 [==============================] - 0s 2ms/step - loss: 1.6470e
-05 - mse: 1.6470e-05 - val_loss: 5.1566e-04 - val_mse: 5.1566e-04
Epoch 98/200
109/109 [==============================] - 0s 2ms/step - loss: 3.4449e
-06 - mse: 3.4449e-06 - val_loss: 2.7398e-04 - val_mse: 2.7398e-04
Epoch 99/200
109/109 [==============================] - 0s 2ms/step - loss: 2.9835e
-06 - mse: 2.9835e-06 - val_loss: 3.0047e-04 - val_mse: 3.0047e-04
Epoch 100/200
109/109 [==============================] - 0s 2ms/step - loss: 3.8958e
-06 - mse: 3.8958e-06 - val_loss: 2.4105e-04 - val_mse: 2.4105e-04
Epoch 101/200
109/109 [==============================] - 0s 2ms/step - loss: 2.5389e
-06 - mse: 2.5389e-06 - val_loss: 5.2184e-04 - val_mse: 5.2184e-04
Epoch 102/200
109/109 [==============================] - 0s 2ms/step - loss: 2.3335e
-06 - mse: 2.3335e-06 - val_loss: 5.0156e-04 - val_mse: 5.0156e-04
Epoch 103/200
109/109 [==============================] - 0s 2ms/step - loss: 4.0986e
-06 - mse: 4.0986e-06 - val_loss: 6.1869e-04 - val_mse: 6.1869e-04
Epoch 104/200
109/109 [==============================] - 0s 2ms/step - loss: 6.2334e
-06 - mse: 6.2334e-06 - val_loss: 3.8304e-04 - val_mse: 3.8304e-04
Epoch 105/200
109/109 [==============================] - 0s 2ms/step - loss: 3.5254e
-06 - mse: 3.5254e-06 - val_loss: 5.4136e-04 - val_mse: 5.4136e-04
Epoch 106/200
109/109 [==============================] - 0s 2ms/step - loss: 4.7994e
-06 - mse: 4.7994e-06 - val_loss: 1.9356e-04 - val_mse: 1.9356e-04
Epoch 107/200
109/109 [==============================] - 0s 2ms/step - loss: 3.7571e
-06 - mse: 3.7571e-06 - val_loss: 3.2822e-04 - val_mse: 3.2822e-04
Epoch 108/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1835e
-05 - mse: 1.1835e-05 - val_loss: 4.0793e-04 - val_mse: 4.0793e-04
Epoch 109/200
109/109 [==============================] - 0s 2ms/step - loss: 3.9553e
-06 - mse: 3.9553e-06 - val_loss: 3.2156e-04 - val_mse: 3.2156e-04
Epoch 110/200
109/109 [==============================] - 0s 3ms/step - loss: 5.6027e
-06 - mse: 5.6027e-06 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 111/200
109/109 [==============================] - 0s 2ms/step - loss: 2.8004e
-05 - mse: 2.8004e-05 - val_loss: 2.7399e-04 - val_mse: 2.7399e-04
Epoch 112/200
109/109 [==============================] - 0s 2ms/step - loss: 4.0113e
-06 - mse: 4.0113e-06 - val_loss: 2.6728e-04 - val_mse: 2.6728e-04
Epoch 113/200
109/109 [==============================] - 0s 2ms/step - loss: 7.6745e
-06 - mse: 7.6745e-06 - val_loss: 9.0413e-04 - val_mse: 9.0413e-04
Epoch 114/200
109/109 [==============================] - 0s 2ms/step - loss: 5.5002e
-06 - mse: 5.5002e-06 - val_loss: 1.7059e-04 - val_mse: 1.7059e-04
```

```
Epoch 115/200
109/109 [==============================] - 0s 2ms/step - loss: 1.0087e
-05 - mse: 1.0087e-05 - val_loss: 0.0010 - val_mse: 0.0010
Epoch 116/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2698e
-05 - mse: 1.2698e-05 - val_loss: 6.8716e-04 - val_mse: 6.8716e-04
Epoch 117/200
109/109 [==============================] - 0s 2ms/step - loss: 2.0024e
-05 - mse: 2.0024e-05 - val_loss: 6.2126e-04 - val_mse: 6.2126e-04
Epoch 118/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1646e
-05 - mse: 1.1646e-05 - val_loss: 4.2350e-04 - val_mse: 4.2350e-04
Epoch 119/200
109/109 [==============================] - 0s 3ms/step - loss: 7.8003e
-06 - mse: 7.8003e-06 - val_loss: 1.9411e-04 - val_mse: 1.9411e-04
Epoch 120/200
109/109 [==============================] - 0s 2ms/step - loss: 3.1733e
-06 - mse: 3.1733e-06 - val_loss: 6.3571e-04 - val_mse: 6.3571e-04
Epoch 121/200
109/109 [==============================] - 0s 2ms/step - loss: 9.0198e
-06 - mse: 9.0198e-06 - val_loss: 1.5693e-04 - val_mse: 1.5693e-04
Epoch 122/200
109/109 [==============================] - 0s 2ms/step - loss: 3.1233e
-06 - mse: 3.1233e-06 - val_loss: 2.9518e-04 - val_mse: 2.9518e-04
Epoch 123/200
109/109 [==============================] - 0s 3ms/step - loss: 4.6903e
-06 - mse: 4.6903e-06 - val_loss: 3.5395e-04 - val_mse: 3.5395e-04
Epoch 124/200
109/109 [==============================] - 0s 2ms/step - loss: 4.4899e
-06 - mse: 4.4899e-06 - val_loss: 2.2433e-04 - val_mse: 2.2433e-04
Epoch 125/200
109/109 [==============================] - 0s 2ms/step - loss: 4.6194e
-06 - mse: 4.6194e-06 - val_loss: 1.8864e-04 - val_mse: 1.8864e-04
Epoch 126/200
109/109 [==============================] - 0s 2ms/step - loss: 3.5004e
-06 - mse: 3.5004e-06 - val_loss: 2.4682e-04 - val_mse: 2.4682e-04
Epoch 127/200
109/109 [==============================] - 0s 2ms/step - loss: 6.2115e
-06 - mse: 6.2115e-06 - val_loss: 3.6860e-04 - val_mse: 3.6860e-04
Epoch 128/200
109/109 [==============================] - 0s 2ms/step - loss: 5.8319e
-06 - mse: 5.8319e-06 - val_loss: 6.9464e-04 - val_mse: 6.9464e-04
Epoch 129/200
109/109 [==============================] - 0s 2ms/step - loss: 3.2368e
-06 - mse: 3.2368e-06 - val_loss: 2.4182e-04 - val_mse: 2.4182e-04
Epoch 130/200
109/109 [==============================] - 0s 2ms/step - loss: 5.2216e
-06 - mse: 5.2216e-06 - val_loss: 2.9794e-04 - val_mse: 2.9794e-04
Epoch 131/200
109/109 [==============================] - 0s 2ms/step - loss: 5.6852e
-06 - mse: 5.6852e-06 - val_loss: 3.5451e-04 - val_mse: 3.5451e-04
Epoch 132/200
109/109 [==============================] - 0s 2ms/step - loss: 7.4864e
-06 - mse: 7.4864e-06 - val_loss: 4.6384e-04 - val_mse: 4.6384e-04
Epoch 133/200
109/109 [==============================] - 0s 2ms/step - loss: 2.8816e
-05 - mse: 2.8816e-05 - val_loss: 0.0022 - val_mse: 0.0022
```

```
Epoch 134/200
109/109 [==============================] - 0s 2ms/step - loss: 1.6827e
-05 - mse: 1.6827e-05 - val_loss: 3.2375e-04 - val_mse: 3.2375e-04
Epoch 135/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4538e
-05 - mse: 1.4538e-05 - val_loss: 8.8248e-04 - val_mse: 8.8248e-04
Epoch 136/200
109/109 [==============================] - 0s 2ms/step - loss: 2.5458e
-05 - mse: 2.5458e-05 - val_loss: 4.2723e-04 - val_mse: 4.2723e-04
Epoch 137/200
109/109 [==============================] - 0s 2ms/step - loss: 9.7684e
-06 - mse: 9.7684e-06 - val_loss: 2.6187e-04 - val_mse: 2.6187e-04
Epoch 138/200
109/109 [==============================] - 0s 2ms/step - loss: 3.6488e
-06 - mse: 3.6488e-06 - val_loss: 3.3050e-04 - val_mse: 3.3050e-04
Epoch 139/200
109/109 [==============================] - 0s 2ms/step - loss: 6.3473e
-06 - mse: 6.3473e-06 - val_loss: 1.5126e-04 - val_mse: 1.5126e-04
Epoch 140/200
109/109 [==============================] - 0s 2ms/step - loss: 5.2499e
-06 - mse: 5.2499e-06 - val_loss: 2.6273e-04 - val_mse: 2.6273e-04
Epoch 141/200
109/109 [==============================] - 0s 2ms/step - loss: 2.0086e
-06 - mse: 2.0086e-06 - val_loss: 2.6759e-04 - val_mse: 2.6759e-04
Epoch 142/200
109/109 [==============================] - 0s 2ms/step - loss: 3.6819e
-06 - mse: 3.6819e-06 - val_loss: 3.4964e-04 - val_mse: 3.4964e-04
Epoch 143/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6921e
-06 - mse: 2.6921e-06 - val_loss: 1.8904e-04 - val_mse: 1.8904e-04
Epoch 144/200
109/109 [==============================] - 0s 3ms/step - loss: 2.1287e
-06 - mse: 2.1287e-06 - val_loss: 3.3149e-04 - val_mse: 3.3149e-04
Epoch 145/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6727e
-06 - mse: 2.6727e-06 - val_loss: 2.7568e-04 - val_mse: 2.7568e-04
Epoch 146/200
109/109 [==============================] - 0s 2ms/step - loss: 6.6378e
-06 - mse: 6.6378e-06 - val_loss: 3.0880e-04 - val_mse: 3.0880e-04
Epoch 147/200
109/109 [==============================] - 0s 2ms/step - loss: 3.3581e
-06 - mse: 3.3581e-06 - val_loss: 3.5063e-04 - val_mse: 3.5063e-04
Epoch 148/200
109/109 [==============================] - 0s 2ms/step - loss: 5.5349e
-06 - mse: 5.5349e-06 - val_loss: 1.6389e-04 - val_mse: 1.6389e-04
Epoch 149/200
109/109 [==============================] - 0s 2ms/step - loss: 8.6606e
-06 - mse: 8.6606e-06 - val_loss: 3.3932e-04 - val_mse: 3.3932e-04
Epoch 150/200
109/109 [==============================] - 0s 2ms/step - loss: 5.1212e
-06 - mse: 5.1212e-06 - val_loss: 4.8957e-04 - val_mse: 4.8957e-04
Epoch 151/200
109/109 [==============================] - 0s 2ms/step - loss: 1.4053e
-05 - mse: 1.4053e-05 - val_loss: 3.1438e-04 - val_mse: 3.1438e-04
Epoch 152/200
109/109 [==============================] - 0s 2ms/step - loss: 1.3455e
-05 - mse: 1.3455e-05 - val_loss: 3.8371e-04 - val_mse: 3.8371e-04
```

```
Epoch 153/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6274e
-05 - mse: 2.6274e-05 - val_loss: 2.3795e-04 - val_mse: 2.3795e-04
Epoch 154/200
109/109 [==============================] - 0s 2ms/step - loss: 1.7023e
-05 - mse: 1.7023e-05 - val_loss: 4.1250e-04 - val_mse: 4.1250e-04
Epoch 155/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2621e
-05 - mse: 1.2621e-05 - val_loss: 3.2641e-04 - val_mse: 3.2641e-04
Epoch 156/200
109/109 [==============================] - 0s 2ms/step - loss: 4.1493e
-06 - mse: 4.1493e-06 - val_loss: 5.6692e-04 - val_mse: 5.6692e-04
Epoch 157/200
109/109 [==============================] - 0s 2ms/step - loss: 9.3892e
-06 - mse: 9.3892e-06 - val_loss: 3.2599e-04 - val_mse: 3.2599e-04
Epoch 158/200
109/109 [==============================] - 0s 2ms/step - loss: 5.3626e
-06 - mse: 5.3626e-06 - val_loss: 2.3381e-04 - val_mse: 2.3381e-04
Epoch 159/200
109/109 [==============================] - 0s 2ms/step - loss: 3.3623e
-06 - mse: 3.3623e-06 - val_loss: 9.3593e-04 - val_mse: 9.3593e-04
Epoch 160/200
109/109 [==============================] - 0s 2ms/step - loss: 3.2040e
-06 - mse: 3.2040e-06 - val_loss: 1.8854e-04 - val_mse: 1.8854e-04
Epoch 161/200
109/109 [==============================] - 0s 2ms/step - loss: 2.1529e
-06 - mse: 2.1529e-06 - val_loss: 1.9125e-04 - val_mse: 1.9125e-04
Epoch 162/200
109/109 [==============================] - 0s 3ms/step - loss: 4.3155e
-06 - mse: 4.3155e-06 - val_loss: 2.7621e-04 - val_mse: 2.7621e-04
Epoch 163/200
109/109 [==============================] - 0s 2ms/step - loss: 5.5784e
-06 - mse: 5.5784e-06 - val_loss: 7.0597e-04 - val_mse: 7.0597e-04
Epoch 164/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1744e
-05 - mse: 1.1744e-05 - val_loss: 1.9495e-04 - val_mse: 1.9495e-04
Epoch 165/200
109/109 [==============================] - 0s 2ms/step - loss: 2.5018e
-06 - mse: 2.5018e-06 - val_loss: 2.0243e-04 - val_mse: 2.0243e-04
Epoch 166/200
109/109 [==============================] - 0s 3ms/step - loss: 4.1678e
-06 - mse: 4.1678e-06 - val_loss: 1.8627e-04 - val_mse: 1.8627e-04
Epoch 167/200
109/109 [==============================] - 0s 2ms/step - loss: 5.5117e
-06 - mse: 5.5117e-06 - val_loss: 1.8953e-04 - val_mse: 1.8953e-04
Epoch 168/200
109/109 [==============================] - 0s 2ms/step - loss: 6.3888e
-06 - mse: 6.3888e-06 - val_loss: 3.2767e-04 - val_mse: 3.2767e-04
Epoch 169/200
109/109 [==============================] - 0s 2ms/step - loss: 4.1994e
-05 - mse: 4.1994e-05 - val_loss: 4.5342e-04 - val_mse: 4.5342e-04
Epoch 170/200
109/109 [==============================] - 0s 2ms/step - loss: 3.7309e
-06 - mse: 3.7309e-06 - val_loss: 2.1066e-04 - val_mse: 2.1066e-04
Epoch 171/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6218e
-06 - mse: 2.6218e-06 - val_loss: 2.0160e-04 - val_mse: 2.0160e-04
```

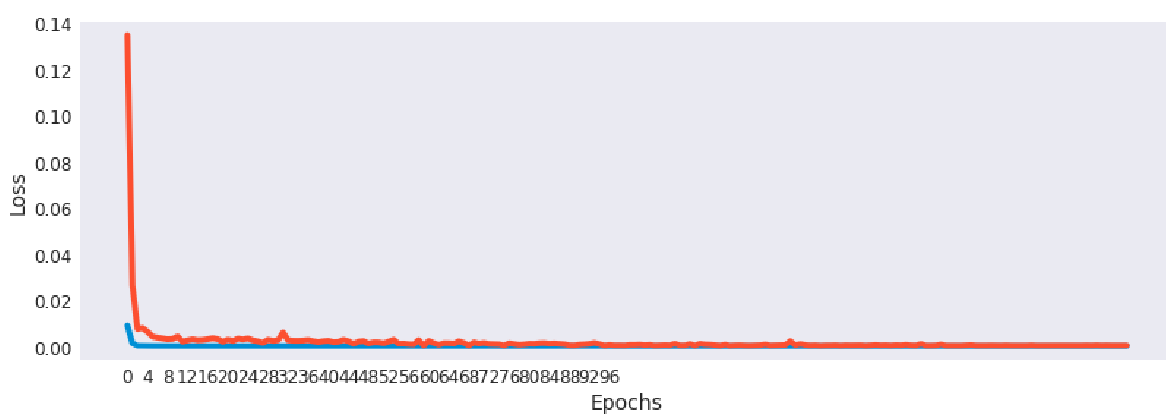
```
Epoch 172/200
109/109 [==============================] - 0s 2ms/step - loss: 3.8854e
-06 - mse: 3.8854e-06 - val_loss: 2.0873e-04 - val_mse: 2.0873e-04
Epoch 173/200
109/109 [==============================] - 0s 2ms/step - loss: 2.4817e
-06 - mse: 2.4817e-06 - val_loss: 2.6891e-04 - val_mse: 2.6891e-04
Epoch 174/200
109/109 [==============================] - 0s 2ms/step - loss: 2.3302e
-06 - mse: 2.3302e-06 - val_loss: 2.4592e-04 - val_mse: 2.4592e-04
Epoch 175/200
109/109 [==============================] - 0s 2ms/step - loss: 3.7059e
-06 - mse: 3.7059e-06 - val_loss: 2.1528e-04 - val_mse: 2.1528e-04
Epoch 176/200
109/109 [==============================] - 0s 2ms/step - loss: 2.5111e
-06 - mse: 2.5111e-06 - val_loss: 2.9326e-04 - val_mse: 2.9326e-04
Epoch 177/200
109/109 [==============================] - 0s 2ms/step - loss: 2.3314e
-06 - mse: 2.3314e-06 - val_loss: 2.0545e-04 - val_mse: 2.0545e-04
Epoch 178/200
109/109 [==============================] - 0s 2ms/step - loss: 6.0043e
-06 - mse: 6.0043e-06 - val_loss: 2.1533e-04 - val_mse: 2.1533e-04
Epoch 179/200
109/109 [==============================] - 0s 2ms/step - loss: 1.1343e
-05 - mse: 1.1343e-05 - val_loss: 2.0959e-04 - val_mse: 2.0959e-04
Epoch 180/200
109/109 [==============================] - 0s 2ms/step - loss: 2.2695e
-06 - mse: 2.2695e-06 - val_loss: 2.0161e-04 - val_mse: 2.0161e-04
Epoch 181/200
109/109 [==============================] - 0s 2ms/step - loss: 3.9263e
-06 - mse: 3.9263e-06 - val_loss: 3.2057e-04 - val_mse: 3.2057e-04
Epoch 182/200
109/109 [==============================] - 0s 2ms/step - loss: 8.0237e
-06 - mse: 8.0237e-06 - val_loss: 1.9099e-04 - val_mse: 1.9099e-04
Epoch 183/200
109/109 [==============================] - 0s 2ms/step - loss: 5.9347e
-06 - mse: 5.9347e-06 - val_loss: 2.0587e-04 - val_mse: 2.0587e-04
Epoch 184/200
109/109 [==============================] - 0s 2ms/step - loss: 4.8091e
-06 - mse: 4.8091e-06 - val_loss: 2.3308e-04 - val_mse: 2.3308e-04
Epoch 185/200
109/109 [==============================] - 0s 2ms/step - loss: 7.4019e
-06 - mse: 7.4019e-06 - val_loss: 2.0219e-04 - val_mse: 2.0219e-04
Epoch 186/200
109/109 [==============================] - 0s 2ms/step - loss: 4.8786e
-06 - mse: 4.8786e-06 - val_loss: 2.2445e-04 - val_mse: 2.2445e-04
Epoch 187/200
109/109 [==============================] - 0s 2ms/step - loss: 4.4574e
-06 - mse: 4.4574e-06 - val_loss: 2.3659e-04 - val_mse: 2.3659e-04
Epoch 188/200
109/109 [==============================] - 0s 2ms/step - loss: 2.0921e
-06 - mse: 2.0921e-06 - val_loss: 2.0501e-04 - val_mse: 2.0501e-04
Epoch 189/200
109/109 [==============================] - 0s 2ms/step - loss: 1.8341e
-06 - mse: 1.8341e-06 - val_loss: 2.2829e-04 - val_mse: 2.2829e-04
Epoch 190/200
109/109 [==============================] - 0s 3ms/step - loss: 3.7932e
-06 - mse: 3.7932e-06 - val_loss: 2.2098e-04 - val_mse: 2.2098e-04
```

```
Epoch 191/200
109/109 [==============================] - 0s 2ms/step - loss: 6.2323e-06 - mse: 6.2323e-06 - val_loss: 2.2904e-04 - val_mse: 2.2904e-04
Epoch 192/200
109/109 [==============================] - 0s 2ms/step - loss: 1.2508e-05 - mse: 1.2508e-05 - val_loss: 2.8811e-04 - val_mse: 2.8811e-04
Epoch 193/200
109/109 [==============================] - 0s 2ms/step - loss: 9.1237e-06 - mse: 9.1237e-06 - val_loss: 2.3423e-04 - val_mse: 2.3423e-04
Epoch 194/200
109/109 [==============================] - 0s 2ms/step - loss: 7.6162e-06 - mse: 7.6162e-06 - val_loss: 3.4074e-04 - val_mse: 3.4074e-04
Epoch 195/200
109/109 [==============================] - 0s 2ms/step - loss: 6.1994e-06 - mse: 6.1994e-06 - val_loss: 2.3569e-04 - val_mse: 2.3569e-04
Epoch 196/200
109/109 [==============================] - 0s 2ms/step - loss: 6.8630e-06 - mse: 6.8630e-06 - val_loss: 2.8789e-04 - val_mse: 2.8789e-04
Epoch 197/200
109/109 [==============================] - 0s 2ms/step - loss: 2.1549e-05 - mse: 2.1549e-05 - val_loss: 2.3220e-04 - val_mse: 2.3220e-04
Epoch 198/200
109/109 [==============================] - 0s 2ms/step - loss: 7.6966e-06 - mse: 7.6966e-06 - val_loss: 2.5211e-04 - val_mse: 2.5211e-04
Epoch 199/200
109/109 [==============================] - 0s 2ms/step - loss: 2.6226e-06 - mse: 2.6226e-06 - val_loss: 2.7398e-04 - val_mse: 2.7398e-04
Epoch 200/200
109/109 [==============================] - 0s 2ms/step - loss: 5.6496e-06 - mse: 5.6496e-06 - val_loss: 2.4774e-04 - val_mse: 2.4774e-04
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f95abe0b780>

In [ ]:
```
plt.figure(figsize=(12,4))
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.xticks(np.arange(0,100,4))
plt.plot(lstm_model.history.history['loss'], label='train')
plt.plot(lstm_model.history.history['val_loss'], label='test')
```

Out[ ]: [<matplotlib.lines.Line2D at 0x7f95aa2e9c50>]



In [ ]:
```
output=lstm_model.predict(generator_test)
```

In [ ]:
```
toInverseScaleOuput = np.hstack((testX[0:len(testX)-1],output))
inverseScaledOutput=scaler.inverse_transform(toInverseScaleOuput)

inverseScaledActual=scaler.inverse_transform(test[0:len(test)-1])
```

In [ ]:
```
inverseScaledActual[:,-1:],inverseScaledOutput[:,-1:]
```

Out[ ]:
```
(array([[ 794847.],
        [ 822609.],
        [ 850366.],
        [ 879472.],
        [ 907650.],
        [ 937567.],
        [ 970174.],
        [1005642.],
        [1040462.],
        [1077873.],
        [1118108.],
        [1154914.],
        [1194084.],
        [1239685.],
        [1288128.],
        [1337016.],
        [1387088.],
        [1436020.],
        [1482504.],
        [1532135.],
        [1584614.],
        [1639582.],
        [1697068.],
        [1752185.],
        [1804857.],
        [1855345.],
        [1906627.]]), array([[ 822537.22576827],
        [ 852813.55184072],
        [ 877512.83547282],
        [ 895246.53509289],
        [ 917995.26095909],
        [ 956807.96125621],
        [ 988848.44247174],
        [1031220.81269276],
        [1045470.14062357],
        [1089684.42150736],
        [1126857.17656982],
        [1153691.89941418],
        [1204592.80395043],
        [1289319.99235642],
        [1325150.48471558],
        [1369432.16541779],
        [1429528.11358476],
        [1462068.5352937 ],
        [1510766.19125175],
        [1577625.75800264],
        [1627167.78390813],
        [1706907.50276458],
        [1767512.69400406],
        [1868519.62987077],
        [1886455.58745289],
        [1958853.17261732],
        [2060850.97942567]]))
```

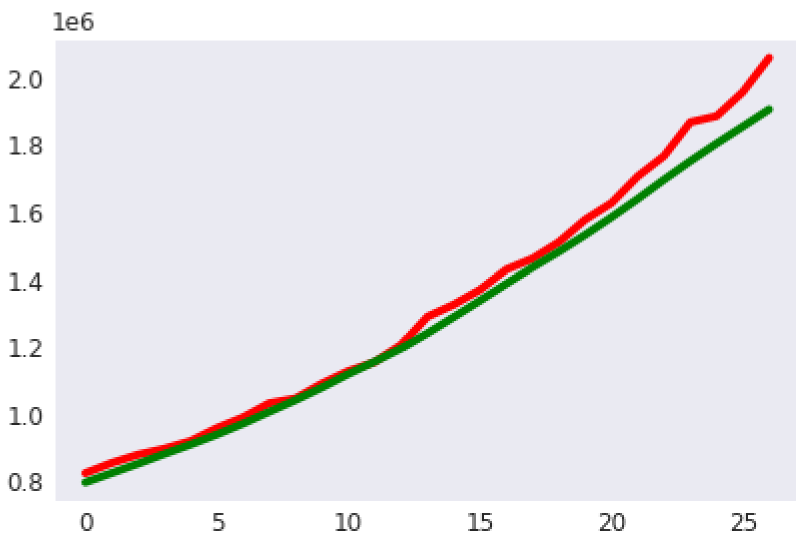
In [ ]:
```
rms = sqrt(mean_squared_error(inverseScaledActual[:,-1:],inverseScaled
Output[:,-1:]))
print('RMSE : ',rms)
```

```
RMSE :  54680.27538494734
```

In [ ]:
```
mae = mean_absolute_error(inverseScaledActual[:,-1:],inverseScaledOutp
ut[:,-1:])
print('MAE :' ,mae)
```

```
MAE : 41083.69849842788
```

In [ ]:
```
plt.plot(inverseScaledOutput[:,-1:],'r')
plt.plot(inverseScaledActual[:,-1:],'g')
plt.show()
```



In [ ]:
```
r2score=r2_score(inverseScaledActual[:,-1:],inverseScaledOutput[:,-1
:])
print('R2 Score: ',r2score )
```

```
R2 Score:  0.9741836751204442
```