Final_Code

September 16, 2020

1 COVID-19 visulization and forcasting

1.1 Goals

- Visualize the evolution of the confimed cases, deaths and recoverd cases
- Forcasting the confirmed cases, deaths and actives cases using ARIMA, prophet and LSTM

[1]:
```
import plotly.offline as pyo       # To work with plotly offline
pyo.init_notebook_mode()

import plotly.graph_objects as go  # Plotly graph objects see documents for more


#-- General packages
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns # To visualize data
# To avoid plt.show()
%matplotlib inline

# sklearn packages
from sklearn.impute import SimpleImputer # to replace missing values with␣
↪appropriate central tendencies (mean,mode,median)


#-- Statistics model required for forcasting
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, acf, pacf,arma_order_select_ic
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA

import warnings
warnings.simplefilter('ignore')
```

1.2 Exploratory Data Anaylsis

- Load the required data to variables using Pandas
- Recent population data is from https://uidai.gov.in/images/state-wise-aadhaar-saturation.pdf

- Latitude and Longitude is from https://www.kaggle.com/adityarc/india-state-wise-latitudes-and-longitudes-2020
- Census and Covid cases is from https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid_19_india.csv

[2]:
```
# Data on cases upto 30th August 2020
cases = pd.read_csv("covid_19_india.csv")
# It contains latitude and longitude coordinates of indian states.
lat_long = pd.read_csv("Latitude_and_Longitude_State.csv")

# It contains the data on Area (data on population is ingnored)
popul = pd.read_csv("population_india_census2011.csv")
# Current population data from Adhaar data base (exact year is of 2019␣
↪population)

population_adhaar = pd.read_csv("populationData.csv") #

# Beds available in each state
beds = pd.read_csv("HospitalBedsIndia.csv")

# Age group affected
age_group = pd.read_csv("AgeGroupDetails.csv")

# Sustainable Development Goals Index (Ranking data with score of maximum␣
↪possible 100)
# https://sdgindiaindex.niti.gov.in/#/ranking
# SDG_Index = pd.read_csv("India_SDG_Index_Rank_Data.csv") # Not used
# details of individual scores
SDG_Index_det = pd.read_csv("India_SDG_Index_Indicator_List.csv")
# Data on GDP or purchasing capacity of citizens statewise needed

# Copy the data
population = popul.copy()
```

[3]:
```
# Display census data
# popul
```

[4]:
```
# Replace old population data with new population data in census data and␣
↪calculate density accordingly
# Retian columns State / Union Territory        Area ( also change column name␣
↪without space)
```

```
population_data = pd.DataFrame()
population_data[['State/UnionTerritory','Area']] = popul[['State / Union␣
↪Territory','Area']]
# Convert Area into float (remove all strings from columns)
population_data['Area'] = population_data['Area'].str.replace(r"\(.*\)","")
population_data['Area'] = population_data['Area'].str.replace("km2","")
population_data['Area'] = population_data['Area'].str.replace(",","")
population_data['Area'] = population_data['Area'].astype('float')

# sort the data frame alphabetical order of states
population_data.sort_values(by= 'State/UnionTerritory',inplace = True)
population_data.reset_index(inplace=True)
population_data.drop('index',axis=1,inplace=True)
```

Adhaar population data

[5]:
```
# population_adhaar
```

[6]:
```
population_adhaar.rename(columns={"States":"State/UnionTerritory"},inplace=True)
population_adhaar.drop(['Unnamed: 0'],axis=1,inplace=True)
population_adhaar.sort_values(by= 'State/UnionTerritory',inplace=True)
```

Calculate new population density

[7]:
```
population_data['Density'] = population_adhaar['Population']/
↪population_data['Area']
population_data['Population'] = population_adhaar['Population']
```

[8]:
```
# population_data
```

1.2.1 Sustainable Development Goals

- data source : https://sdgindiaindex.niti.gov.in/#/ranking 2019 data
- This is used to get information about the doctor availability, below poverty line and Health insurance

[9]:
```
# Use SDG_Index_det.columns.tolist() to view all columns
# SDG_Index_det.columns.tolist()
```

[10]:
```
# SDG_Index_det
```

[11]:
```
# Drop SNo column, drop Target and India row, ANd simplify the required column␣
↪names
# Also note that the many null values have '-' symbol in this SDG_Index_det ,␣
↪these are to be replaced with some numerical values
SDG_Index_det['Area'] = SDG_Index_det['Area'].sort_values()
SDG_Index_det.drop([0,1],inplace = True)
```

```
SDG_reduced = pd.DataFrame()
SDG_reduced[['State/UnionTerritory','Below_poverty_line',␣
↪'Health_insurance_covered', 'Doctors_nurses_available' ]] =␣
↪SDG_Index_det[['Area','Population living below National Poverty line (%)␣
↪(Goal 1)','Households with any usual member covered by any health scheme or␣
↪health insurance (%) (Goal 1)','Total physicians nurses and midwives per␣
↪10000 population (Goal 3)']]

SDG_reduced.reset_index(inplace=True)
SDG_reduced.drop('index',axis=1,inplace=True)
```

[12]:
```
SDG_reduced.replace({'-':np.nan},inplace=True)
SDG_reduced['Below_poverty_line'] = SDG_reduced['Below_poverty_line'].
↪astype('float')
SDG_reduced['Doctors_nurses_available'] =␣
↪SDG_reduced['Doctors_nurses_available'].astype('float')
```

Imputing missing values using median of the feature values - Telangana below poverty line was missing so adjusted with median of the data - Doctors_nurses_available has missing values replaced with median of the data - However Doctors_nurses_available may have been replaced using correlation with GDP and area of the state (which I will try explore later)

[13]:
```
# imputer = SimpleImputer(missing_values=np.nan,strategy='median')

SDG_reduced['Below_poverty_line'] = SDG_reduced['Below_poverty_line'].
↪replace({np.nan:SDG_reduced['Below_poverty_line'].median()})
SDG_reduced['Doctors_nurses_available']=␣
↪SDG_reduced['Doctors_nurses_available'].replace({np.nan:
↪SDG_reduced['Below_poverty_line'].median()})
```

[14]:
```
# lat_long # Print the data to see the row of Daman Diu which is 8
```

[15]:
```
# Drop Daman Diu Lattitude and Longitude (consider same as Dadra and Nagar␣
↪Haveli)
lat_long.drop(8,inplace=True)  # Drop Daman Diu
lat_long.drop('ilist',axis=1,inplace=True) # Drop 'ilist' column not needed
lat_long.sort_values(by='State',inplace=True)
lat_long.set_index('State',inplace=True)
lat_long.reset_index(inplace=True)
```

[16]:
```
# beds
```

[17]:
```
# beds.rename(columns={"States":"State/UnionTerritory"},inplace=True)
# beds.drop(['Hospital beds in public sector', 'Hospital beds in private␣
↪sector'],axis=1,inplace=True)
# beds.sort_values(by='State/UnionTerritory',inplace=True)
# beds.reset_index(inplace=True)
```

```
# beds.drop('index',axis=1,inplace=True)
```

[18]:
```
# age_group.drop('Sno',axis=1,inplace=True)
```

[19]:
```
# population_data
```

Combine the SDG values with population values

[20]:
```
population_data[['Below_poverty_line',
       'Health_insurance_covered', 'Doctors_nurses_available']] =␣
 ↪SDG_reduced[['Below_poverty_line',
       'Health_insurance_covered', 'Doctors_nurses_available']]
population_data[['Longitude', 'Latitude']]  = lat_long[['Longitude',␣
 ↪'Latitude']]
```

[21]:
```
# Change name of Telangana from Telengana
population_data['State/UnionTerritory'].replace({'Telengana':
 ↪'Telangana'},inplace=True)
```

[22]:
```
# population_data
```

To analyze the data and visualize the time series of the confirmed cases

[23]:
```
# Replace the name of Telangana, Daman and Diu , and remove unassinged cases

cases['State/UnionTerritory'].replace({"Telengana" : "Telangana",␣
 ↪"Telengana***" : "Telangana",
                                        "Telangana***" : "Telangana"}, inplace␣
 ↪= True)

cases['State/UnionTerritory'].replace({"Daman & Diu" : "Dadra and Nagar Haveli␣
 ↪and Daman and Diu",
                                          "Dadar Nagar Haveli" : "Dadra and␣
 ↪Nagar Haveli and Daman and Diu"},
                                         inplace = True)
cases = cases[(cases['State/UnionTerritory'] != 'Unassigned') &
                    (cases['State/UnionTerritory'] != 'Cases being reassigned␣
 ↪to states')]
cases['State/UnionTerritory'].unique()
```

[23]:
```
array(['Kerala', 'Telangana', 'Delhi', 'Rajasthan', 'Uttar Pradesh',
       'Haryana', 'Ladakh', 'Tamil Nadu', 'Karnataka', 'Maharashtra',
       'Punjab', 'Jammu and Kashmir', 'Andhra Pradesh', 'Uttarakhand',
       'Odisha', 'Puducherry', 'West Bengal', 'Chhattisgarh',
       'Chandigarh', 'Gujarat', 'Himachal Pradesh', 'Madhya Pradesh',
       'Bihar', 'Manipur', 'Mizoram', 'Andaman and Nicobar Islands',
       'Goa', 'Assam', 'Jharkhand', 'Arunachal Pradesh', 'Tripura',
```

```
       'Nagaland', 'Meghalaya',
       'Dadra and Nagar Haveli and Daman and Diu', 'Sikkim'], dtype=object)
```

[24]:
```
# cases does not contain Lakshadeep (so drop Lakshadweep in other data )
# Convert the datatime using pandas datetime
cases['Date'] = pd.to_datetime(cases['Date'], dayfirst=True)
cases.drop(['Sno', 'Time', 'ConfirmedIndianNational',␣
 ↪'ConfirmedForeignNational'], axis = 1, inplace=True)
cases.head()
# cases[cases.Date == max(cases.Date)] to see the latest cases
# drop_row_list = ['Lakshadweep']
```

[24]:

	Date	State/UnionTerritory	Cured	Deaths	Confirmed
0	2020-01-30	Kerala	0	0	1
1	2020-01-31	Kerala	0	0	1
2	2020-02-01	Kerala	0	0	2
3	2020-02-02	Kerala	0	0	3
4	2020-02-03	Kerala	0	0	3

[25]:
```
print("Starting date : ", min(cases.Date.values))
print("Ending date : ", max(cases.Date.values))
```

```
Starting date :  2020-01-30T00:00:00.000000000
Ending date :  2020-08-30T00:00:00.000000000
```

[26]:
```
# Dialy increment in active cases in whole nation
daily_cases = cases.groupby('Date').sum().reset_index() # Total cases in the␣
 ↪nation
daily_cases['Active'] = 1

for val in daily_cases.index:
    if val != 0:
        daily_cases['Active'].loc[val] = daily_cases['Confirmed'].loc[val] -␣
 ↪daily_cases['Cured'].loc[val-1] - daily_cases['Deaths'].loc[val-1]

daily_cases
```

[26]:

	Date	Cured	Deaths	Confirmed	Active
0	2020-01-30	0	0	1	1
1	2020-01-31	0	0	1	1
2	2020-02-01	0	0	2	2
3	2020-02-02	0	0	3	3
4	2020-02-03	0	0	3	3
..	…	…	…	…	…
209	2020-08-26	2467758	59449	3234474	771499
210	2020-08-27	2523771	60472	3310234	783027
211	2020-08-28	2583948	61529	3387500	803257

```
212 2020-08-29  2648998   62550    3463972  818495
213 2020-08-30  2713933   63498    3542733  831185

[214 rows x 5 columns]
```

1.3 Visualise how cases are changing with Date

[27]:

```
# To visulize the increase in cases
fig = go.Figure()
fig.add_trace(go.Scatter(x = daily_cases['Date'], y = daily_cases['Confirmed'],␣
 ↪name = 'Confirmed'))
fig.add_trace(go.Scatter(x = daily_cases['Date'], y = daily_cases['Cured'],␣
 ↪name = 'Cured'))
fig.add_trace(go.Scatter(x = daily_cases['Date'], y = daily_cases['Deaths'],␣
 ↪name = 'Deaths'))
fig.add_trace(go.Scatter(x = daily_cases['Date'], y = daily_cases['Active'],␣
 ↪name = 'Active Cases'))

fig.update_layout(title = 'CORONA VIRUS CASES IN INDIA', yaxis_title = 'Cases␣
 ↪Count (in lakhs)')

fig.show()
```

[28]:

```
# Total population of India and Finding percentage of population infected
total_pop = population_data['Population'].sum()
print("The total population of India is : ", total_pop)
# To calulate the percentage population infectd at each date.
# Merge the population data, povery data, health insurance and Doctors avilable␣
 ↪per 1000 data
popul = cases.merge(population_data[['State/UnionTerritory',␣
 ↪'Population','Density']])
popul['ConfirmPerc'] = 0
# To find the percentage confirmed cases in each state
popul['ConfirmPerc'] = (popul['Confirmed']/popul['Population'])*100
```

```
The total population of India is :  1371360351
```

[29]:

```
# Percentage population infected in each state
fig = go.Figure()
for st in popul['State/UnionTerritory'].unique():
    df = popul[popul['State/UnionTerritory'] == st]
    fig.add_trace(go.Scatter(x = df['Date'], y = df['ConfirmPerc'], name = st))

fig.update_layout(title = 'Positive Cases Percentage Per Population',␣
 ↪yaxis_title = 'Percentage (%)')
```

```
fig.show()
```

[30]:
```
# Group the data on the basis of the Date and find sum to calculate cases for␣
 ↪whole nation
popul_nation = popul.drop('ConfirmPerc', axis=1).groupby('Date').sum()

# Total population
popul_nation['Population'] = total_pop

# Calculating total percentage of positive cases in whole nation
popul_nation['TotConfirmPerc'] = (popul_nation['Confirmed']/
 ↪popul_nation['Population'])*100
popul_nation
```

[30]:
```
              Cured  Deaths  Confirmed  Population        Density  \
Date
2020-01-30        0       0          1  1371360351     918.597200
2020-01-31        0       0          1  1371360351     918.597200
2020-02-01        0       0          2  1371360351     918.597200
2020-02-02        0       0          3  1371360351     918.597200
2020-02-03        0       0          3  1371360351     918.597200
…                 …       …          …           …              …
2020-08-26  2467758   59449    3234474  1371360351   38590.100603
2020-08-27  2523771   60472    3310234  1371360351   38590.100603
2020-08-28  2583948   61529    3387500  1371360351   38590.100603
2020-08-29  2648998   62550    3463972  1371360351   38590.100603
2020-08-30  2713933   63498    3542733  1371360351   38590.100603

            TotConfirmPerc
Date
2020-01-30    7.292029e-08
2020-01-31    7.292029e-08
2020-02-01    1.458406e-07
2020-02-02    2.187609e-07
2020-02-03    2.187609e-07
…                      …
2020-08-26    2.358588e-01
2020-08-27    2.413832e-01
2020-08-28    2.470175e-01
2020-08-29    2.525939e-01
2020-08-30    2.583371e-01

[214 rows x 6 columns]
```

[31]:
```
# Date vs Percentage cases
fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x = popul_nation.index, y =␣
↪popul_nation['TotConfirmPerc']))
fig.update_layout(title = 'Percentage of positive cases across India',␣
↪yaxis_title = 'Percentage (%)')
fig.show()
```

[32]:
```
# # To visulize the daily active cases in the country
# daily_cases_nation = cases.groupby('Date').sum().reset_index()
# daily_cases_nation['Active'] = 1

# for val in daily_cases_nation.index:
#     if val != 0:
#         daily_cases_nation['Active'].loc[val] =␣
↪daily_cases_nation['Confirmed'].loc[val] - daily_cases_nation['Cured'].
↪loc[val-1] - daily_cases_nation['Deaths'].loc[val-1]

# fig = go.Figure()
# fig.add_trace(go.Scatter(x = daily_cases_nation['Date'], y =␣
↪daily_cases['Active'], name = 'Active Cases'))

# fig.update_layout(title = 'Daily Active Cases', xaxis_title = 'Time',␣
↪yaxis_title = 'Count (in lakhs)')
# fig.show()
```

[33]:
```
# State wise actives versus the Date
statewise_daily_cases = cases.sort_values(by=['State/UnionTerritory', 'Date']).
↪reset_index(drop=True)
statewise_daily_cases['ActiveCases'] = 0

for st in sorted(cases['State/UnionTerritory'].unique()):
    df = statewise_daily_cases[statewise_daily_cases['State/UnionTerritory'] ==␣
↪st]
    for i in df.index:
        conf = statewise_daily_cases['Confirmed'].iloc[i]
        rec = statewise_daily_cases['Cured'].iloc[i-1]
        death = statewise_daily_cases['Deaths'].iloc[i-1]

        statewise_daily_cases['ActiveCases'].iloc[i] = conf - rec - death
    statewise_daily_cases['ActiveCases'].iloc[df.index[0]] =␣
↪statewise_daily_cases['Confirmed'].iloc[df.index[0]]

fig = go.Figure()
for st in statewise_daily_cases['State/UnionTerritory'].unique():
    df = statewise_daily_cases[statewise_daily_cases['State/UnionTerritory'] ==␣
↪st]
    fig.add_trace(go.Scatter(x = df['Date'], y = df['ActiveCases'], name = st))
```

```
fig.update_layout(title = 'Daily Active Cases', xaxis_title = 'Time',␣
 ↪yaxis_title = 'Count (in lakhs)')
fig.show()
```

[34]:
```
# state wise daily cases with percentage active cases and confirmed cases at␣
 ↪each date along with poverty line etc.
popul = popul.sort_values(by=['State/UnionTerritory', 'Date']).
 ↪reset_index(drop=True)
statewise_daily_cases[['Population'        ,'ConfirmPerc','Density']] =␣
 ↪popul[['Population'        ,'ConfirmPerc','Density']]
statewise_daily_cases['ActivePerc'] = (statewise_daily_cases['ActiveCases']/
 ↪statewise_daily_cases['Population'])*100
```

[35]:
```
statewise_daily_cases.dtypes
```

[35]:
```
Date                       datetime64[ns]
State/UnionTerritory               object
Cured                               int64
Deaths                              int64
Confirmed                           int64
ActiveCases                         int64
Population                          int64
ConfirmPerc                       float64
Density                           float64
ActivePerc                        float64
dtype: object
```

[36]:
```
popul.dtypes
```

[36]:
```
Date                       datetime64[ns]
State/UnionTerritory               object
Cured                               int64
Deaths                              int64
Confirmed                           int64
Population                          int64
Density                           float64
ConfirmPerc                       float64
dtype: object
```

To find the correlation between confirmed percentage cases and density, poverty, doctors avilability

[37]:
```
dummy = statewise_daily_cases.groupby('Date')
```

[38]:
```
Density_corr = []

date = []
```

```
cases_name = 'ConfirmPerc'
for name,group in dummy:
    date.append(name)
    Density_corr.append(group[cases_name].corr(group['Density']))
```

[39]:
```
correlation_time = pd.DataFrame()
correlation_time['Date'] = date
correlation_time['Density_corr'] = Density_corr
```

[40]:
```
correlation_time.fillna(0,inplace=True)
```

[41]:
```
fig = px.line(correlation_time,x='Date',y='Density_corr',title="Correlation␣
↪with amount of population in each state")
fig.show()
```

Conclusion - From plot on density versus the confirmed cases percentage, it can be seen that the correlation between density and confrimed percentage cases increases, which means to say that, where there is a high popultion density there are more covid cases. - However this may not be very clear due to less testing of the individuals which creates a sampling bias towards the symptotoc patients over asymptotic patients

1.4 Forcasting using ARIMA and Prophet

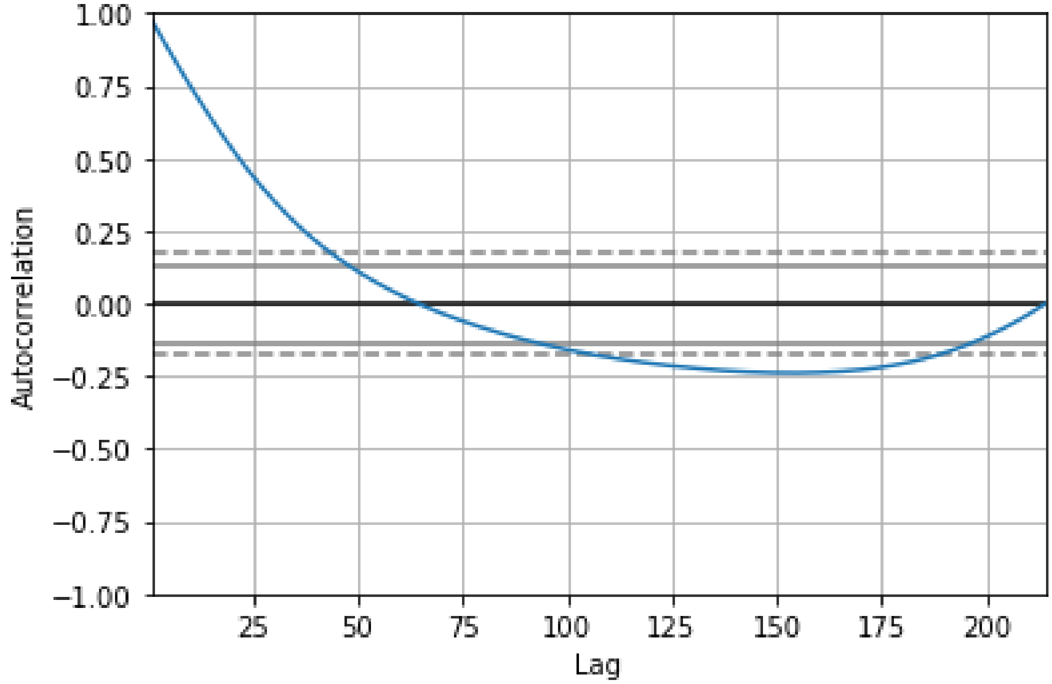
1.4.1 ARIMA model

[42]:
```
from matplotlib import pyplot
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.arima_model import ARIMA
import itertools
```

[43]:
```
Confirmed_cases_country=popul_nation[['Confirmed','TotConfirmPerc']]
```

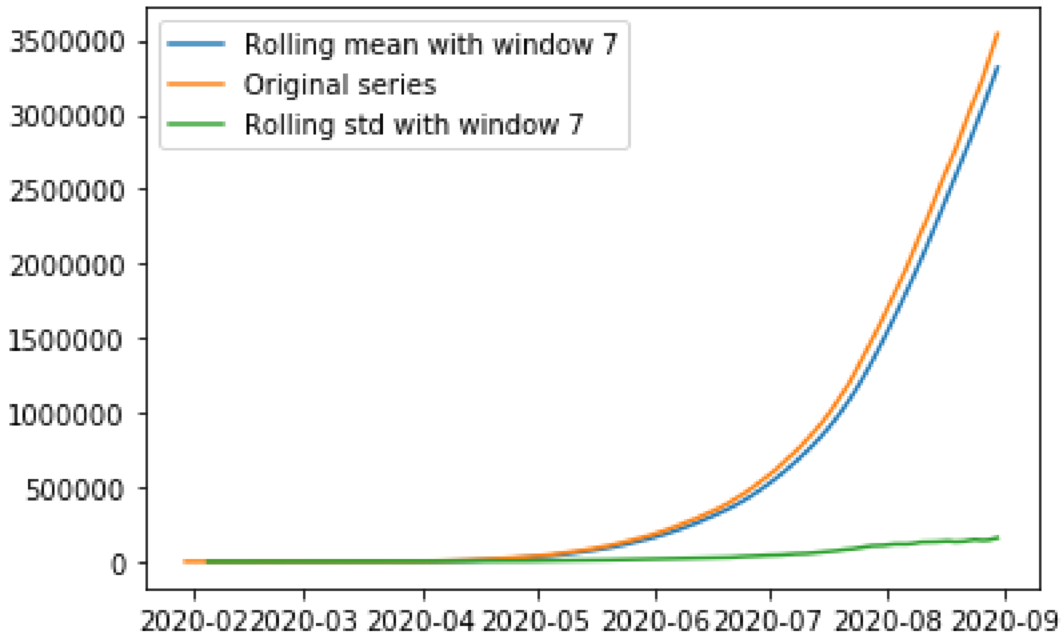
[44]:
```
autocorrelation_plot(Confirmed_cases_country['Confirmed'])
```

[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa31f899590>
```



[45]:
```
rolling_mean = Confirmed_cases_country.rolling(window=7,center=False).mean().
 ↪dropna()
plt.plot(rolling_mean['Confirmed'],label='Rolling mean with window 7')
plt.plot(Confirmed_cases_country['Confirmed'],label='Original series')
rolling_std = Confirmed_cases_country.rolling(window=7,center=False).std().
 ↪dropna()
plt.plot(rolling_std['Confirmed'],label='Rolling std with window 7')
plt.legend()
```

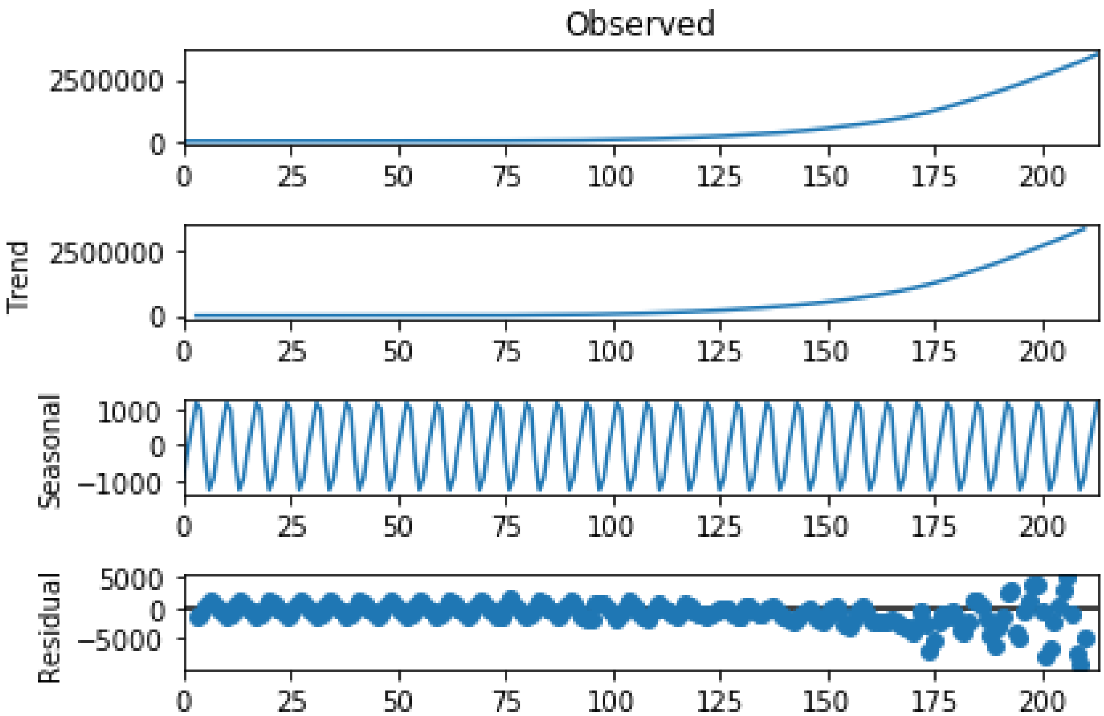
[45]:
```
<matplotlib.legend.Legend at 0x7fa31f7a5250>
```

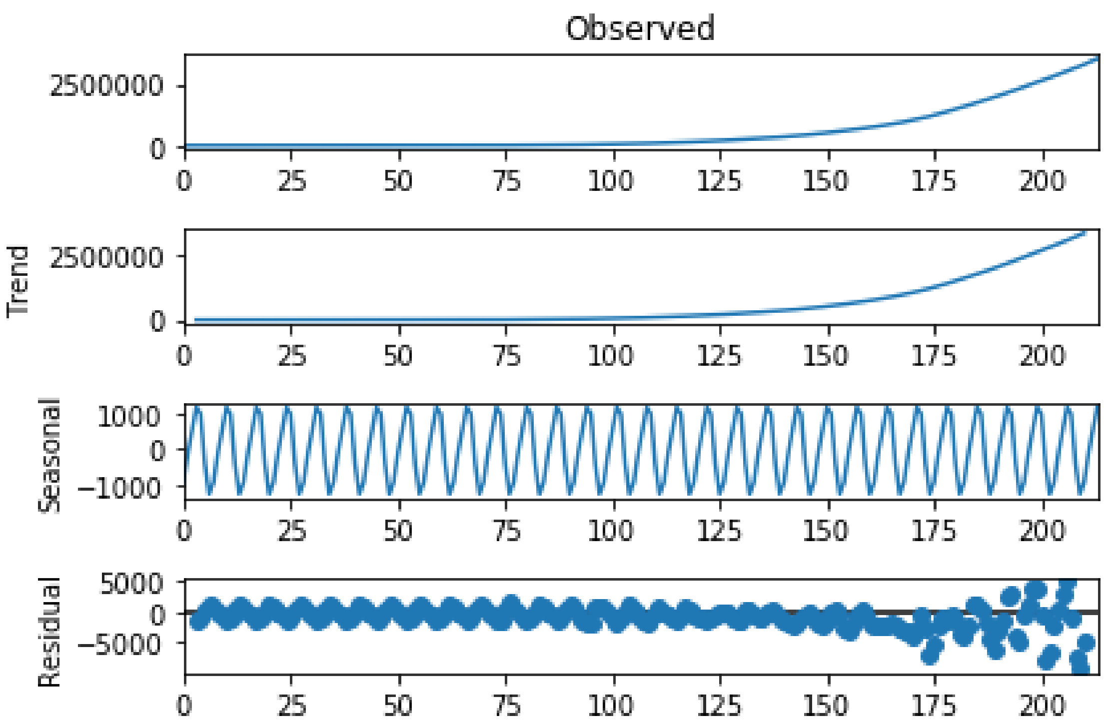


Decomposing the seasonalities - If freq is taken to be 7, we expect the time series to follow some seasonalities - There is also trend observed which is evodent since cases are increasing - Seasonality says that after every week the cases follow same pattern, may be in week days the cases are more as people go outside for jobs

[46]:
```
sm.tsa.seasonal_decompose(Confirmed_cases_country['Confirmed'].values,freq=7).
↪plot()
```

[46]:


Observed
2500000
0
Trend
Seasonal
1000
−1000
Residual
5000
−5000
0 25 50 75 100 125 150 175 200


Observed
2500000
0
Trend
Seasonal
1000
−1000
Residual
5000
−5000
0 25 50 75 100 125 150 175 200

To forcast the time series should be stationary so following code checks wether the series is stationary or not

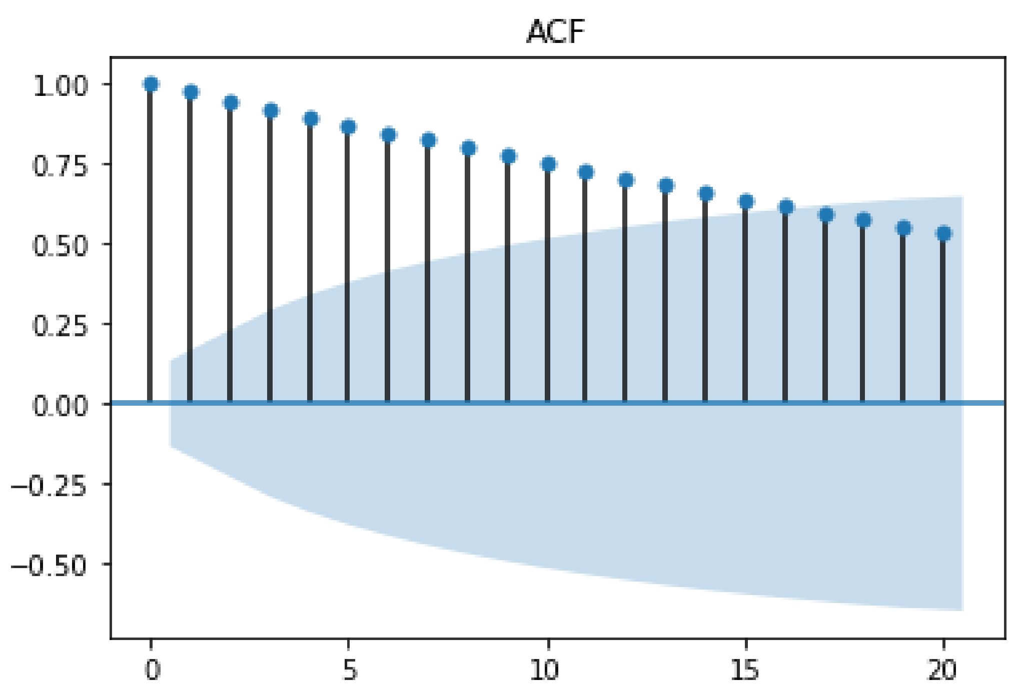
[47]:
```
print('Results of Dickey-Fuller Test:')
test = adfuller(Confirmed_cases_country['Confirmed'].values, autolag='AIC')
results = pd.Series(test[0:4], index=['Test Statistic','p-value','#Lags␣
↪Used','Number of Observations Used'])
for i,val in test[4].items():
    results['Critical Value (%s)'%i] = val
print (results)
```

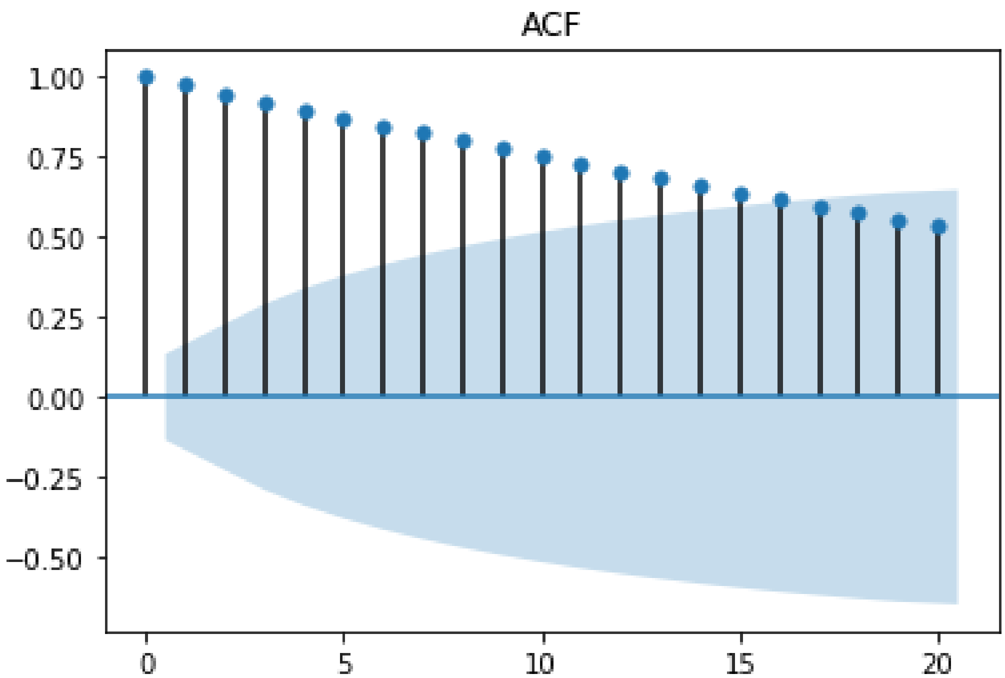
```
Results of Dickey-Fuller Test:
Test Statistic                   1.187770
p-value                          0.995900
#Lags Used                       9.000000
Number of Observations Used    204.000000
Critical Value (1%)             -3.462818
Critical Value (5%)             -2.875815
Critical Value (10%)            -2.574379
dtype: float64
```

Since the p-value is more than 0.05, the time series is not stationary

[48]:
```
plot_acf(Confirmed_cases_country['Confirmed'].values,lags=20,title="ACF")
```

[48]:

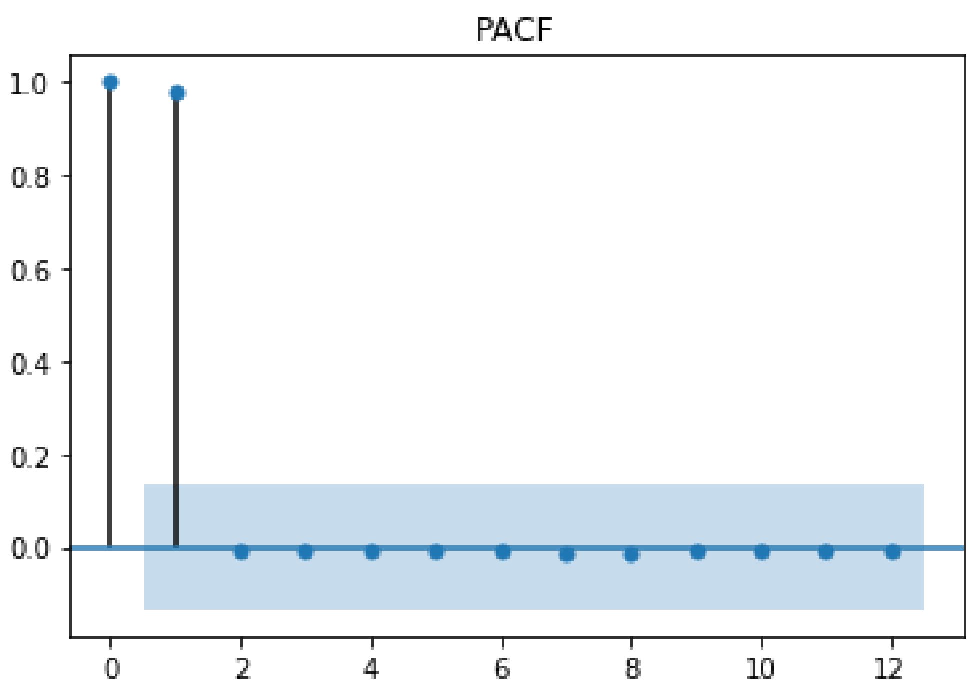


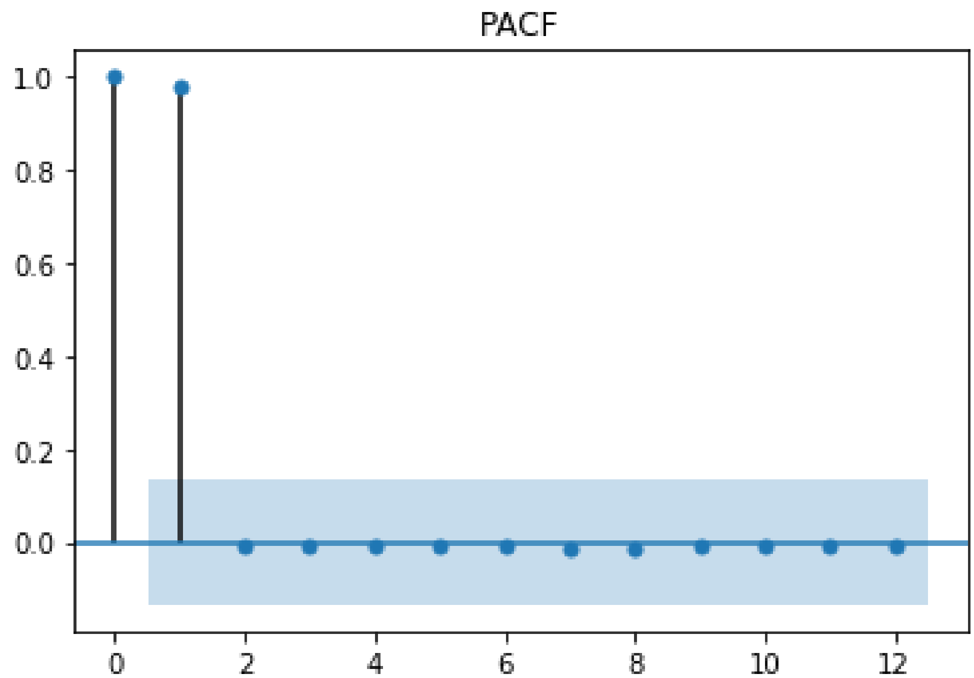


Since the ACF is decaying with lags a moving averge model upto 15 lags can be considered

[49]:
```
plot_pacf(Confirmed_cases_country['Confirmed'].values,lags=12,title="PACF")
```

[49]:





From PACF we may have to consider lag of 1 of the auto regressive part

Create features for data frame: - As it was clear from seasonality that there exist some repition of pattern of time series every week - Along with that feature other features such as month, week and day of week etc can be considered

[50]:
```
def create_features(df,label=None):
    """
    time series features from datetime index are created
    """
    df = df.copy() # Copy the dataframe
    df['Date'] = df.index
    df['hour'] = df['Date'].dt.hour
    df['dayofweek'] = df['Date'].dt.dayofweek
    df['quarter'] = df['Date'].dt.quarter
    df['month'] = df['Date'].dt.month
    df['year'] = df['Date'].dt.year
    df['dayofyear'] = df['Date'].dt.dayofyear
    df['dayofmonth'] = df['Date'].dt.day
    df['weekofyear'] = df['Date'].dt.weekofyear

    X = df[['hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear']]
```

```
    return X
```

[51]:
```
def mape(y1, y_pred):
    y1, y_pred = np.array(y1), np.array(y_pred)
    return np.mean(np.abs((y1 - y_pred) / y1)) * 100

def split(ts):
    #splitting 85%/15% because of little amount of data
    size = int(len(ts) * 0.85)
    train= ts[:size]
    test = ts[size:]
    return(train,test)
```

[52]:
```
def split(df):
    #splitting 85%/15% because of little amount of data
    size = int(len(df) * 0.85)
    train= df[:size]
    test = df[size:]
    return(train,test)

def arima(ts,test):
    p=range(0,15)
    d = range(0,15)
    q=range(0,15)
    a=99999
    pdq=list(itertools.product(p,d,q))

    #Determining the best parameters
    for var in pdq:
        try:
            model = ARIMA(ts, order=var)
            result = model.fit()

            if (result.aic<=a) :
                a=result.aic
                param=var
        except:
            continue

    #Modeling
    model = ARIMA(ts, order=param)
    result = model.fit()
    result.plot_predict(start=int(len(ts) * 0.7), end=int(len(ts) * 1.2))
    pred=result.forecast(steps=len(test))[0]
    #Plotting results
    f,ax=plt.subplots()
```

```
    plt.plot(pred,c='green', label= 'predictions')
    plt.plot(test, c='red',label='real values')
    plt.legend()
    plt.title('True vs predicted values')
    #Printing the error metrics
    print(result.aic)
    print(result.summary())

    print('\nMean absolute percentage error: %f'%mape(test,pred))
    return (pred)
```

[53]:
```
train_with_features , test_with_features =␣
↪split(Confirmed_cases_country['Confirmed'].values)
```

[54]:
```

```

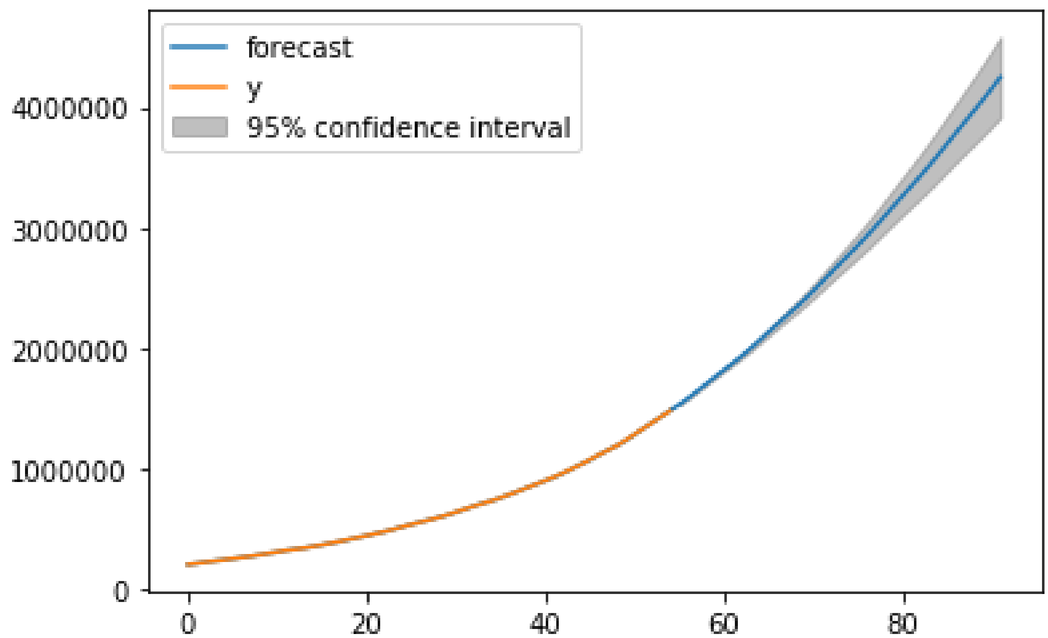
[55]:
```
pred=arima(train_with_features,test_with_features)
```

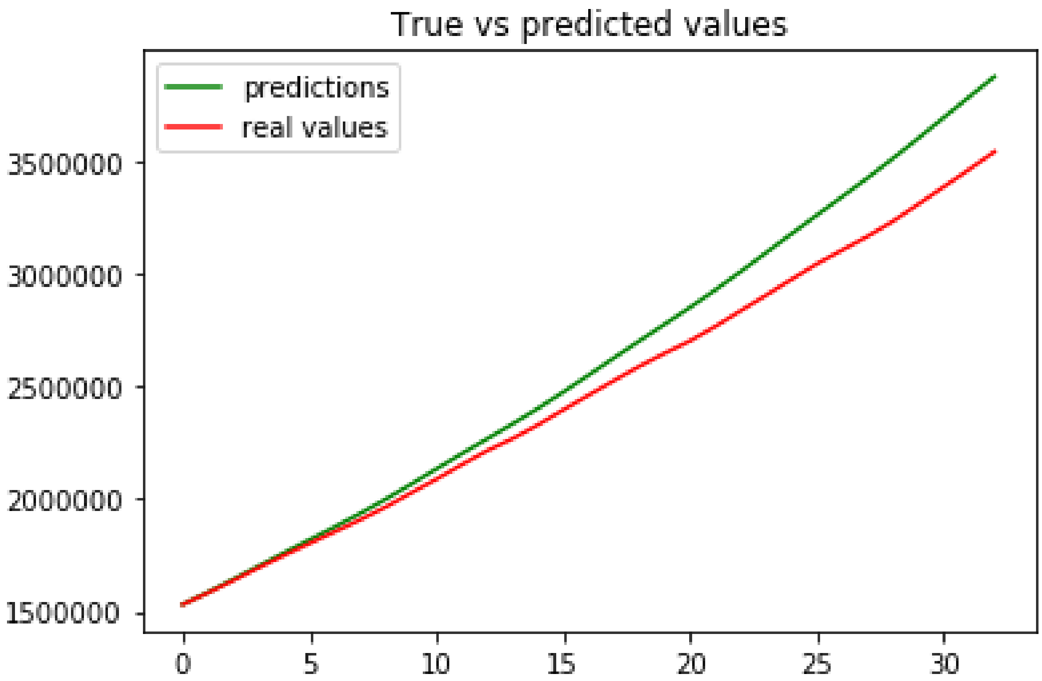
```
2884.071129740485
                             ARIMA Model Results
==============================================================================
Dep. Variable:                   D2.y   No. Observations:                  179
Model:                 ARIMA(3, 2, 8)   Log Likelihood               -1429.036
Method:                       css-mle   S.D. of innovations            691.307
Date:                Wed, 16 Sep 2020   AIC                           2884.071
Time:                        23:37:08   BIC                           2925.507
Sample:                             2   HQIC                          2900.873

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        266.5401    606.971      0.439      0.661    -923.101    1456.182
ar.L1.D2.y     2.1883      0.027     81.908      0.000       2.136       2.241
ar.L2.D2.y    -2.1750      0.034    -64.043      0.000      -2.242      -2.108
ar.L3.D2.y     0.9806      0.018     53.067      0.000       0.944       1.017
ma.L1.D2.y    -2.2393      0.086    -25.905      0.000      -2.409      -2.070
ma.L2.D2.y     1.8605      0.198      9.397      0.000       1.472       2.249
ma.L3.D2.y    -0.0817      0.224     -0.365      0.715      -0.520       0.357
ma.L4.D2.y    -0.7279      0.215     -3.387      0.001      -1.149      -0.307
ma.L5.D2.y    -0.3000      0.192     -1.566      0.117      -0.676       0.075
ma.L6.D2.y     1.2788      0.219      5.840      0.000       0.850       1.708
ma.L7.D2.y    -0.9370      0.221     -4.249      0.000      -1.369      -0.505
ma.L8.D2.y     0.2340      0.099      2.356      0.018       0.039       0.429
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
```

```
AR.1            1.0079          -0.0000j          1.0079          -0.0000
AR.2            0.6051          -0.8035j          1.0059          -0.1473
AR.3            0.6051          +0.8035j          1.0059           0.1473
MA.1           -0.9604          -0.6050j          1.1351          -0.4105
MA.2           -0.9604          +0.6050j          1.1351           0.4105
MA.3            0.5145          -0.9030j          1.0393          -0.1676
MA.4            0.5145          +0.9030j          1.0393           0.1676
MA.5            1.0890          -0.6447j          1.2655          -0.0851
MA.6            1.0890          +0.6447j          1.2655           0.0851
MA.7            1.3593          -0.2647j          1.3848          -0.0306
MA.8            1.3593          +0.2647j          1.3848           0.0306
-----------------------------------------------------------------------------

Mean absolute percentage error: 4.270643
```





So the model parameters which gave best prediction on the given data are (3, 2, 8), where, - 3 is the parameter for auto regressive part - 2 is the parameter for Integegrated part - 8 is for the moving average part

It has predicted that in 20 days the number of confirmed cases would double. However, note that if suddenly if lock down is announced then this model will fail to work, since it has learnt through previous history, which may not work well for sudden cahnge

1.4.2 Facebooks' Prophet model

[56]:
```
from fbprophet import Prophet
from fbprophet.plot import plot_plotly, add_changepoints_to_plot
```

[57]:
```
model = Prophet()
```

[58]:
```
Confirmed_cases_country.reset_index(inplace=True)
Confirmed_cases_prophet = Confirmed_cases_country[['Date','Confirmed']]
```

[59]:
```
Confirmed_cases_prophet.columns = ['ds', 'y']
Confirmed_cases_prophet_train , Confirmed_cases_prophet_test =␣
 ↪split(Confirmed_cases_prophet)
```

[60]:
```
model.fit(Confirmed_cases_prophet_train)
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
```

[60]:
```
<fbprophet.forecaster.Prophet at 0x7fa31cba6650>
```

[67]:
```python
future1 = model.make_future_dataframe(periods=33)

forecast_india_conf = model.predict(future1)
# forecast_india_conf
print('\nMean absolute percentage error: 
 ↪%f'%mape(Confirmed_cases_prophet_test['y'],forecast_india_conf['yhat'][181:
 ↪]))
```

```
Mean absolute percentage error: 25.785738
```

[74]:
```python
future2 = model.make_future_dataframe(periods=85)

forecast_india_conf = model.predict(future2)
```

[75]:
```python
fig = plot_plotly(model, forecast_india_conf)

fig.update_layout(template='plotly_white')

fig.show()
```

1.4.3 Conclusion

Prophet model is predicting that the cases by october is around 4 million which around same as predicted by ARIMA.

[76]:
```python
forecast_india_conf
```

[76]:
```
             ds         trend    yhat_lower    yhat_upper   trend_lower  \
0   2020-01-30 -2.637415e+03 -3.791495e+04  2.850014e+04 -2.637415e+03
1   2020-01-31 -2.554483e+03 -3.852557e+04  3.043029e+04 -2.554483e+03
2   2020-02-01 -2.471552e+03 -3.803263e+04  3.012891e+04 -2.471552e+03
3   2020-02-02 -2.388620e+03 -3.668145e+04  2.977933e+04 -2.388620e+03
4   2020-02-03 -2.305688e+03 -3.427198e+04  3.267913e+04 -2.305688e+03
..          …           …            …            …            …
261 2020-10-17  3.522173e+06  3.231173e+06  3.817047e+06  3.234154e+06
262 2020-10-18  3.549132e+06  3.256137e+06  3.855476e+06  3.256360e+06
263 2020-10-19  3.576090e+06  3.274735e+06  3.882147e+06  3.278256e+06
264 2020-10-20  3.603049e+06  3.296054e+06  3.929083e+06  3.299862e+06
265 2020-10-21  3.630007e+06  3.320255e+06  3.951313e+06  3.324012e+06
```

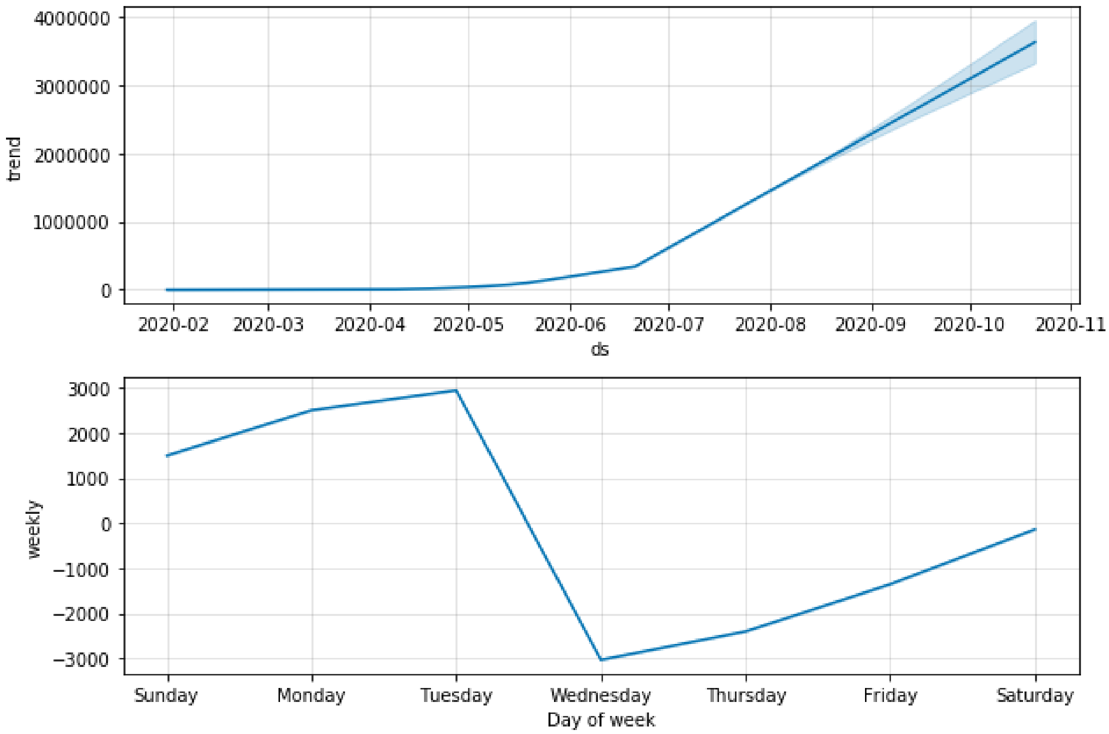
```
      trend_upper  additive_terms  additive_terms_lower  additive_terms_upper  \
0   -2.637415e+03    -2402.410044          -2402.410044          -2402.410044
1   -2.554483e+03    -1356.822664          -1356.822664          -1356.822664
2   -2.471552e+03     -140.362462           -140.362462           -140.362462
3   -2.388620e+03     1494.787003           1494.787003           1494.787003
4   -2.305688e+03     2498.038062           2498.038062           2498.038062
..            …               …                     …                     …
261  3.823359e+06     -140.362462           -140.362462           -140.362462
262  3.855927e+06     1494.787003           1494.787003           1494.787003
263  3.888865e+06     2498.038062           2498.038062           2498.038062
264  3.922810e+06     2940.543103           2940.543103           2940.543103
265  3.953066e+06    -3033.772997          -3033.772997          -3033.772997

          weekly  weekly_lower  weekly_upper  multiplicative_terms  \
0   -2402.410044  -2402.410044  -2402.410044                   0.0
1   -1356.822664  -1356.822664  -1356.822664                   0.0
2    -140.362462   -140.362462   -140.362462                   0.0
3    1494.787003   1494.787003   1494.787003                   0.0
4    2498.038062   2498.038062   2498.038062                   0.0
..           …             …             …                     …
261  -140.362462   -140.362462   -140.362462                   0.0
262  1494.787003   1494.787003   1494.787003                   0.0
263  2498.038062   2498.038062   2498.038062                   0.0
264  2940.543103   2940.543103   2940.543103                   0.0
265 -3033.772997  -3033.772997  -3033.772997                   0.0

     multiplicative_terms_lower  multiplicative_terms_upper          yhat
0                           0.0                         0.0 -5.039825e+03
1                           0.0                         0.0 -3.911306e+03
2                           0.0                         0.0 -2.611914e+03
3                           0.0                         0.0 -8.938330e+02
4                           0.0                         0.0  1.923497e+02
..                          …                           …             …
261                         0.0                         0.0  3.522033e+06
262                         0.0                         0.0  3.550627e+06
263                         0.0                         0.0  3.578588e+06
264                         0.0                         0.0  3.605989e+06
265                         0.0                         0.0  3.626973e+06

[266 rows x 16 columns]
```

[77]:
```
fig = model.plot_components(forecast_india_conf)
```



- As expected from earlier seasonality decompositions the cases are high in week day and tend to decrease in mid week, eventually again catching at week ends.
- This is mau be due to the working days in the week days

```
[ ]:
```