covid_predictions

December 6, 2020

[1]:
```
from IPython.display import Image
Image("../Images/Logo.jpg")
```

[1]:

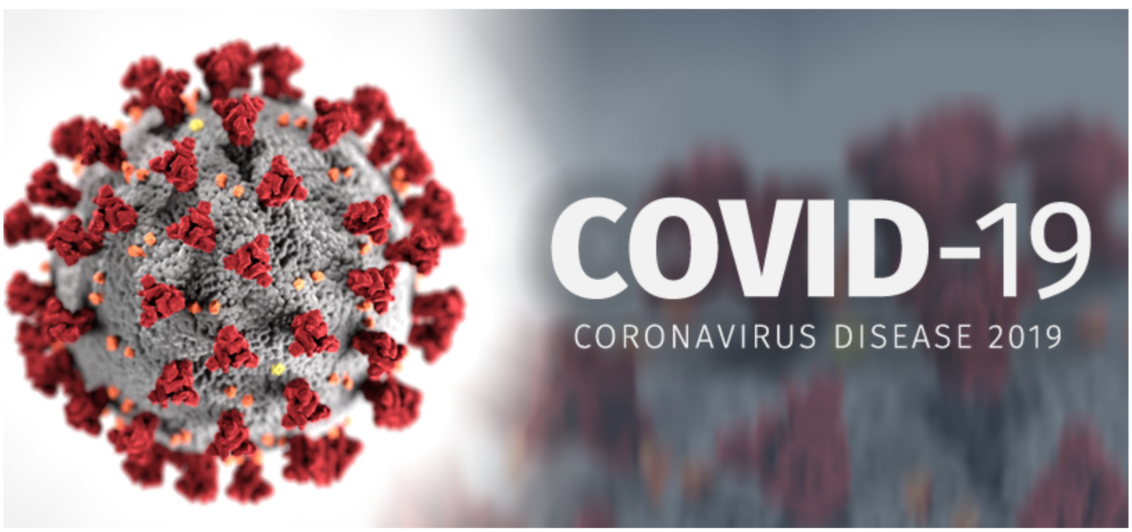
#

Graduate Project ENEL 698

Github Link

[2]:
```
Image("../Images/Covid-19.png")
```

[2]:



1 Context

1.0.1 Novel Coronavirus 2019 (nCoV-2019) is a virus which affects respiratory system and was first discovered in wuhan, China. Some early reports suggested that virus may have been transmitted from animal to person. As we know whole world has been shutdown because of the widespread cases. At this time it's unclear how easily or sustainably this virus is spreading between people.

2 Current Cases (WorldWide)

2.0.1 To know how bad the world has been affected lets get some information on current situation.

Lets import all the dependencies for scrapping the website

[3]:
```
import bs4
from urllib.request import Request, urlopen
from urllib.request import urlopen as uReq
from bs4 import BeautifulSoup as soup
import pandas as pd
```

Dataset used

[4]:
```
pd.read_excel('../covid_data/Data/meta/covid_data.xlsx')
```

[4]:
```
                    Data Used  \
0               nCoV2019.live
1   Confirmed cases dataframe

                                                Info  \
0  This data has been scraped from nCoV2019.live…
1  This data is a time series representation of c…

                                                Link
0                             https://ncov2019.live/
1  https://raw.githubusercontent.com/CSSEGISandDa…
```

[5]:
```
# grabbing the url

url = "https://ncov2019.live/"
req = Request(url, headers={"User-Agent" : "Mozilla/5.0"})

webpage = urlopen(req).read()

#parsing it as lxml
pagesoup = soup(webpage,"lxml")
```

Website Information

1. Website Name

2. Link to Website

[6]:
```python
from IPython.display import display, Markdown
```

[7]:
```python
#finding the relevant tags to scrap the data from website

website_name = pagesoup.find('a',class_ = "navbar-brand")
link = "https://ncov2019.live/"
Markdown('<strong>{}</strong>{}'.format(website_name.text,link))
```

[7]:

nCoV2019.live

https://ncov2019.live/

[8]:
```python
#some quick facts from the website

quickfacts = pagesoup.find('div', class_ = "container--wrap bg-navy-4")
Markdown('<strong align="center">{}</strong>'.format(quickfacts))
```

[8]:

Quick Facts

updated: A few minutes ago

62,882,389

Total Confirmed

105,398

Total Critical

1,463,107

Total Deceased

18,304,577

Total Active

42,945,040

Total Recovered

183

Total Vaccines In Development

2.1 World COVID-19 Stats

We will scrap worldwide covid cases.

1. We'll use pandas read.html which lets us read the webpage table without much of complexity.
2. Convert the table into dataframe for further processing.

3. In the header of the list generated you see a number ``1'', which was used in the original website as a filter for arranging data in ascending or descending order.

[9]:
```
import pandas as pd
import requests
```

[10]:
```
# grabbing latest worldwide data

url = "https://ncov2019.live/data/world"

r = requests.get(url)
df_list = pd.read_html(r.text)           #this parse all html tables from a␣
↪webpage to alist
world_df = df_list[2]
world_df
```

[10]:
```
                  Name  Confirmed Per Million  Changes Today  \
0                TOTAL   62885020        8072         329868
1          Afghanistan      46215        1176            249
2              Albania      37625       13080            835
3              Algeria      82221        1861           1009
4              Andorra       6610           0              0
..                 ...        ...         ...            ...
215         Montserrat         13           0              0
216           Anguilla          4           0              0
217  Wallis and Futuna          3           0              0
218              Samoa          2           0              0
219              China      86512          60             11

    Percentage Day Change Critical Deceased Per Million.1  Changes Today.1  \
0                   0.53%   105412  1463137           188             5590
1                   0.54%       93     1763            45               11
2                   2.27%       27      798           277               11
3                   1.24%       44     2410            55               17
4                      0%       20       76             0                0
..                    ...      ...      ...           ...              ...
215                    0%  Unknown        1             0                0
216                    0%  Unknown  Unknown       Unknown                0
217                    0%  Unknown  Unknown       Unknown                0
218                    0%  Unknown  Unknown       Unknown                0
219                 0.01%        8     4634             3                0

    Percentage Death Change      Tests    Active Recovered Per Million.2  \
0                     0.38%  997192334  18304500  42947706          5513
1                     0.63%     147800      7721     36731           935
2                      1.4%     184097     18346     18481          6425
```

```
3                           0.71%    Unknown   Unknown     53204          1204
4                              0%     168635       824      5710             0
..                             …         …         …        …           …
215                            0%        577         0        12             0
216                            0%       2651   Unknown         3             0
217                            0%       1149   Unknown         1             0
218                            0%    Unknown   Unknown   Unknown       Unknown
219                            0%  160000000       280     81598            57

      Population
0     7790414058
1       39283186
2        2876495
3       44173038
4          77316
..           …
215         4993
216        15058
217        11156
218       198956
219   1439323776

[220 rows x 15 columns]
```

Sorting the data on number of confirmed cases

[11]:
```python
# We will now sort the countries based on total confirmed cases column

world_df = world_df.sort_values("Confirmed" , ascending = False)



#Lets get top 10 affected countries

# world_df.head(10)
```

Lets see many coulmns are missing values.

[12]:
```python
world_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 220 entries, 0 to 163
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Name                     220 non-null    object
 1   Confirmed                220 non-null    int64
 2   Per Million              220 non-null    object
```

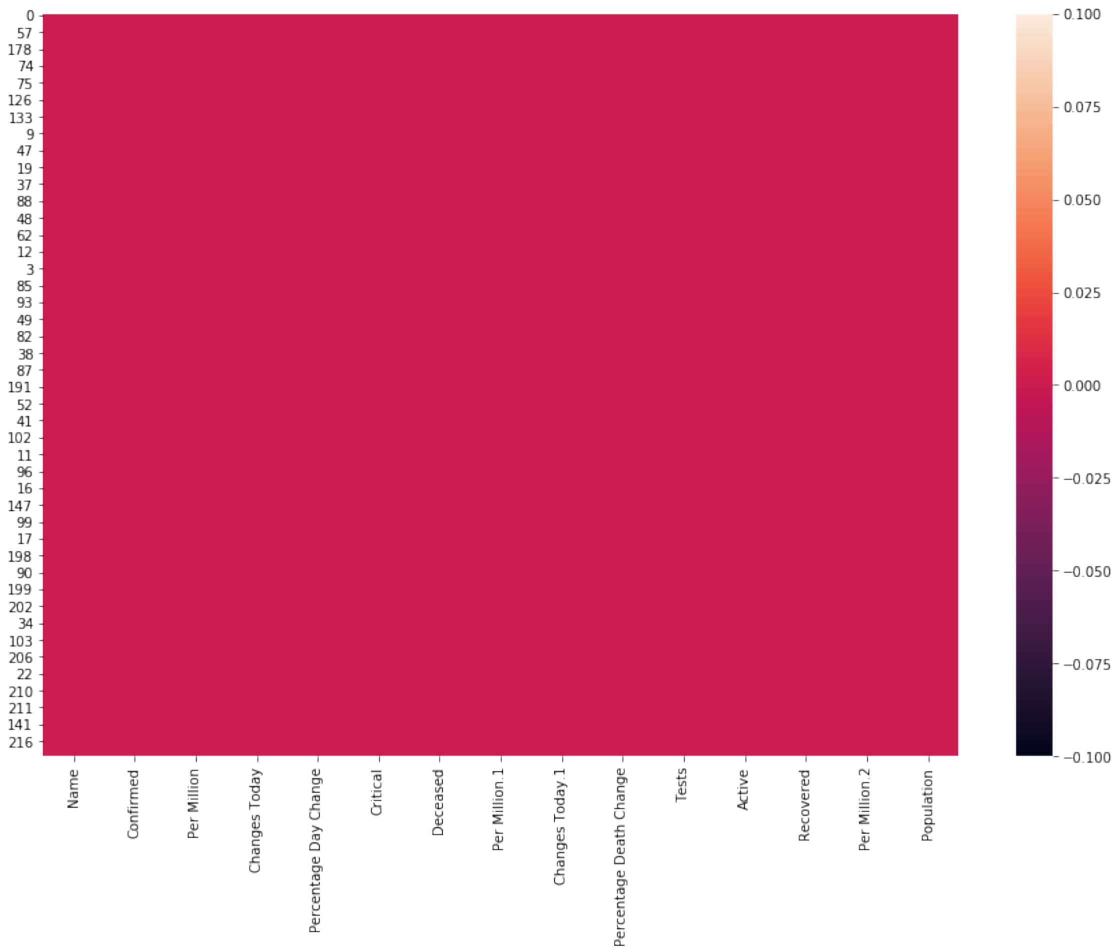
```
 3   Changes Today           220 non-null    int64
 4   Percentage Day Change   220 non-null    object
 5   Critical                220 non-null    object
 6   Deceased                220 non-null    object
 7   Per Million.1           220 non-null    object
 8   Changes Today.1         220 non-null    int64
 9   Percentage Death Change  220 non-null    object
 10  Tests                   220 non-null    object
 11  Active                  220 non-null    object
 12  Recovered               220 non-null    object
 13  Per Million.2           220 non-null    object
 14  Population              220 non-null    object
dtypes: int64(3), object(12)
memory usage: 27.5+ KB
```

Lets import seaborn as well as matplotlib

[13]:
```
#We can also visualize the same using seaborn

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[14]:
```
plt.figure(figsize=(15,10))
sns.heatmap(world_df.isnull())
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1632fcd2848>
```



We'll use plotly express for visualization.

1. It generates graphs which are interactive and user friendly.
2. We can use zoom in and zoom out feature for proper understanding to a specific part of graph.

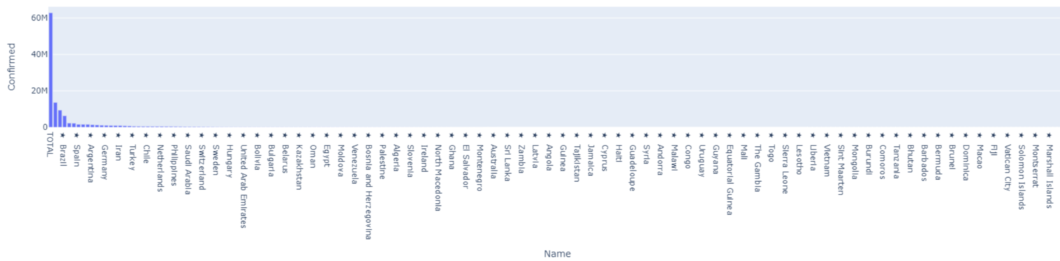
[15]:
```
import plotly.express as px
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

[16]:
```
import plotly.io as pio
pio.renderers.default = 'jupyterlab'
```

Plot number of confirmed cases.

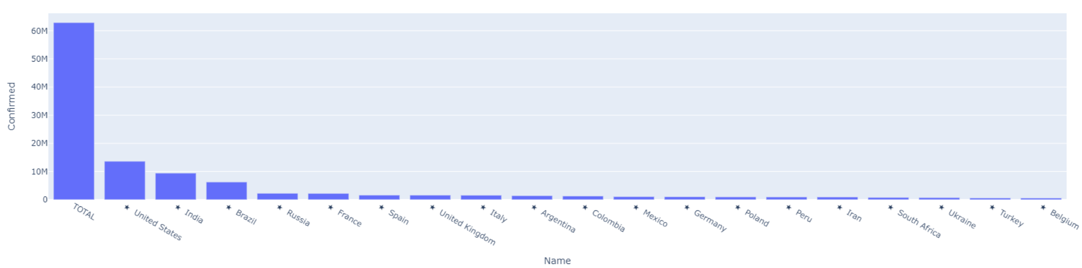
[17]:
```
# plotting world_df based on confirmed cases by country names.
```

```
world_fig = px.bar(world_df, x = 'Name' , y = 'Confirmed')
world_fig.show()
```



- We can zoom in the graph, thats the beauty of plotly.

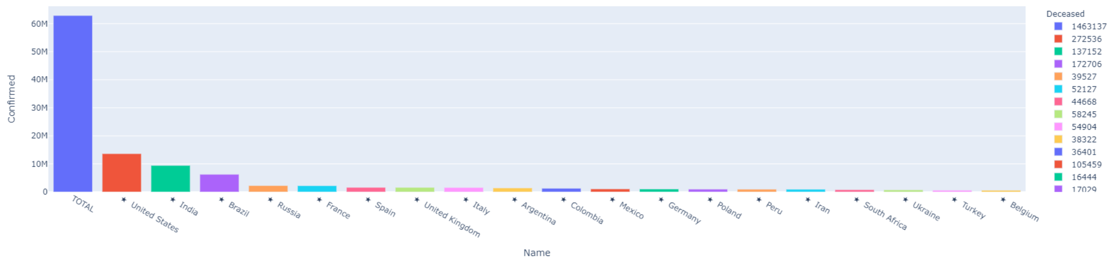
[18]:
```
# Lets plot top 20 countries based on confirmed cases.

world_fig = px.bar(world_df.head(20), x = "Name" , y = 'Confirmed')
world_fig.show()
```



- Now we can see United states holds number 1 position. (cough cough ``we don't wear masks'' - americans)
- Brazil and India comes at the second and third position surpassing Russia respectively.

Now we'll try to explore the world_df in more details.(based on number of Deceased People)

[19]:
```
# Lets see how many people have died with respect to countries. (For top 20␣
↪countries)

world_fig = px.bar(world_df.head(20), x = 'Name', y = 'Confirmed', color =␣
↪"Deceased")
world_fig.show()
```



- Here the color of each bar corrosponds to how many people have died.
- We cannot make out which country has most number of deceased people in a descending order.

[20]:
```python
# lets grab the world_df based on deceased column.
world_df.sort_values('Deceased',ascending = False)
```

[20]:
```
                        Name  Confirmed Per Million  Changes Today  \
163                  Vanuatu          1           0              0
211            New Caledonia         32           0              0
204            Faroe Islands        502           0              0
18                    Bhutan        396           0              1
28                  Cambodia        315          19              7
..                       ...        ...         ...            ...
103                Mauritius        501         394              0
215               Montserrat         13           0              0
167           Western Sahara         10           0              0
210   British Virgin Islands         71           0              0
26                   Burundi        681          57              0

    Percentage Day Change Critical Deceased Per Million.1  Changes Today.1  \
163                    0%  Unknown  Unknown       Unknown                0
211                    0%  Unknown  Unknown       Unknown                0
204                    0%  Unknown  Unknown       Unknown                0
18                  0.25%  Unknown  Unknown       Unknown                0
28                  2.27%  Unknown  Unknown       Unknown                0
..                    ...      ...      ...           ...              ...
103                    0%  Unknown       10             8                0
215                    0%  Unknown        1             0                0
167                    0%  Unknown        1             0                0
210                    0%  Unknown        1             0                0
26                     0%  Unknown        1             0                0

    Percentage Death Change    Tests   Active Recovered Per Million.2  \
163                      0%  Unknown  Unknown   Unknown        Unknown
211                      0%    17179  Unknown        32              0
```

```
204                        0%    166881  Unknown       498              0
18                         0%    200176  Unknown       373              0
28                         0%    228972  Unknown       301             18
..                        …       …       …       …           …
103                        0%    289552       48       443            348
215                        0%       577        0        12              0
167                        0%   Unknown  Unknown         8              0
210                        0%      5193        0        70              0
26                         0%     62215      105       575             48

    Population
163     310047
211     286624
204      48940
18      775088
28    16813462
..         …
103    1272640
215       4993
167     603270
210      30314
26    12033001

[220 rows x 15 columns]
```

- The column contains many unknown values.
- We'll replace all the unknown values with zero.
- Then we will arrange the column in descending order for visualization purpose.

[21]:
```python
# lets replace unknown values to 0.

world_df['Deceased'].replace("Unknown", 0,inplace=True)
world_df['Deceased'] = pd.to_numeric(world_df['Deceased'])          #convert␣
 ↪column from type object to int64
world_df['Deceased']
```

[21]:
```
0      1463137
171     272536
172     137152
173     172706
174      39527
        …
216          0
101          0
217          0
218          0
```

```
163          0
Name: Deceased, Length: 220, dtype: int64
```

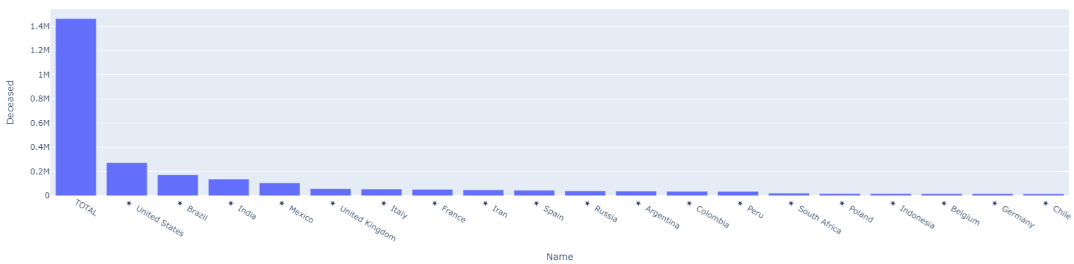
[22]:
```python
# now again lets grab world_df based on deceased column.

world_df.sort_values('Deceased',ascending = False)
```

[22]:
```
                 Name  Confirmed Per Million  Changes Today  \
0               TOTAL   62885020        8072         329868
171     United States   13643294       41119          32937
173            Brazil    6295695       29532           5423
172             India    9430724        6806          37685
179            Mexico    1100683        8500          10008
..                ...        ...         ...            ...
106          Mongolia        784         238             24
51            Eritrea        577         162              0
136        Seychelles        173           0              0
204     Faroe Islands        502           0              0
163           Vanuatu          1           0              0

    Percentage Day Change Critical  Deceased Per Million.1  Changes Today.1  \
0                   0.53%   105412   1463137           188             5590
171                 0.24%    24671    272536           821              282
173                 0.09%     8318    172706           810               69
172                  0.4%     8944    137152            99              419
179                 0.92%     3335    105459           814              586
..                    ...      ...       ...           ...              ...
106                 3.16%        6         0       Unknown                0
51                     0%  Unknown         0       Unknown                0
136                    0%  Unknown         0       Unknown                0
204                    0%  Unknown         0       Unknown                0
163                    0%  Unknown         0       Unknown                0

    Percentage Death Change      Tests    Active Recovered Per Million.2  \
0                     0.38%  997192334  18304500  42947706           5513
171                    0.1%  191296242   5317138   8053620          24272
173                   0.04%   21900000    560450   5562539          26093
172                   0.31%  139503803    448821   8844751           6383
179                   0.56%    2849307    181970    813254           6281
..                      ...        ...       ...       ...            ...
106                      0%     165620   Unknown       354            107
51                       0%      21655   Unknown       498            140
136                      0%       5200   Unknown       162              0
204                      0%     166881   Unknown       498              0
163                      0%    Unknown   Unknown   Unknown        Unknown

     Population
```

```
0     7790414058
171    331801570
173    213180600
172   1385567591
179    129487827
..           …
106      3299801
51       3566467
136        98598
204        48940
163       310047

[220 rows x 15 columns]
```

- Perfecto!. Now we can see that column has been cleared off all the ``Unknown''.

[23]:
```python
# lets again try ro visualize world_df based on death poll for top 20 countries.

world_fig = px.bar(world_df.sort_values('Deceased', ascending=False).head(20),␣
↪x = 'Name' , y = 'Deceased')
world_fig.show()
```

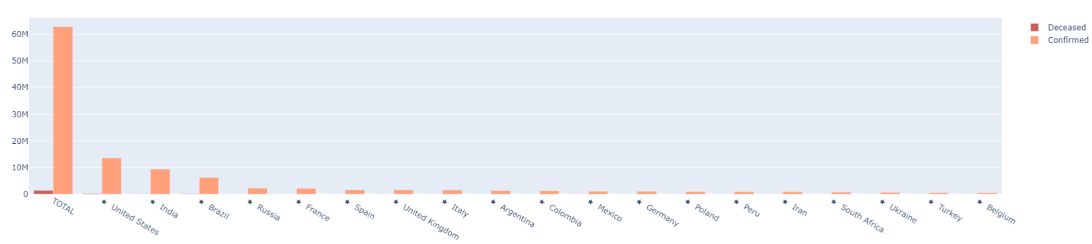


Click on the link for more information.

- United States tops the chart. If you want to know why United States leads in coronvirus cases, but not pandemic response
- Brazil also surpasses 100,000 deaths and becomes the one of the worst affected countries. `Death became normal': Brazil surpasses 100,000 deaths from COVID-19
- Mexico's death toll also reached 59.106k and many young people are dying of COVID-19 Why Are So Many Young People Dying Of Covid-19 In Mexico City?
- India has also reached 56k and there are many questions about India's rising COVID-19 infection Five key questions about India's rising Covid-19 infections

Lets visualize the death toll in relation to total confirmed case

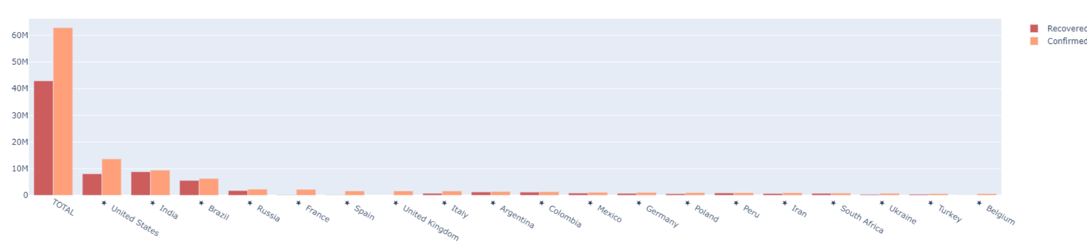
[24]:
```
# lets visualize the death toll based on total confirmed case

import plotly.graph_objects as go


# for grouped barplot using Deceased numbers per country and total number of␣
↪cases per country.

fig = go.Figure(data = [
go.Bar(
    x = world_df['Name'],
    y = world_df["Deceased"].head(20),
    name = "Deceased",
    marker_color = "indianred"
),
go.Bar(
    x = world_df['Name'],
    y = world_df['Confirmed'].head(20),
    name = 'Confirmed',
    marker_color = "lightsalmon"
)
])

# Here we modify the tickangle of the xaxis, resulting in rotated labels.
fig.update_layout(barmode='group')
fig.show()
```



- Here we can see the Death toll is very low as compared to confirmed cases, which is because most of the people recover from COVID-19. Early estimates predicted that the overall COVID-19 recovery rate is between 97% and 99.75%.
- Mortality rate calculated = 3.4% (802.318k/23.09665M)

lets visualize the recovered cases based on total confirmed case

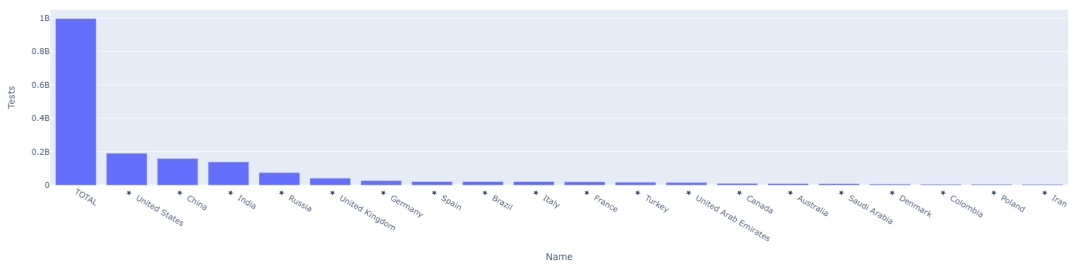
[25]:
```
# lets visualize the recovered case based in relation to total confirmed case

import plotly.graph_objects as go


# for grouped barplot using recovered cases per country and total number of cases per country.

fig = go.Figure(data = [
go.Bar(
    x = world_df['Name'],
    y = world_df["Recovered"].head(20),
    name = "Recovered",
    marker_color = "indianred"
),
go.Bar(
    x = world_df['Name'],
    y = world_df['Confirmed'].head(20),
    name = 'Confirmed',
    marker_color = "lightsalmon"
)
])

# Here we modify the tickangle of the xaxis, resulting in rotated labels.
fig.update_layout(barmode='group')
fig.show()
```



- Here we can see how many person recovered in relation to total cases registered.
- Recovery rate = 67% (15.4827M/23.09665M), this contradicts early predicted value of recovery rate which was 97%.
- Recovery rate and mortality rate are based on how well a country is implementing the testing of its people. Estimating mortality from COVID-19

Lets see who has implemented testing vastly.

[26]:
```
# replace unknown values from the column

world_df['Tests'].replace("Unknown", 0, inplace=True)
world_df['Tests'] = pd.to_numeric(world_df['Tests'])          #convert column␣
↪from type object to int64



#Now lets plot the data

world_fig = px.bar(world_df.sort_values('Tests', ascending=False).head(20), x =␣
↪'Name' , y = 'Tests')
world_fig.show()
```

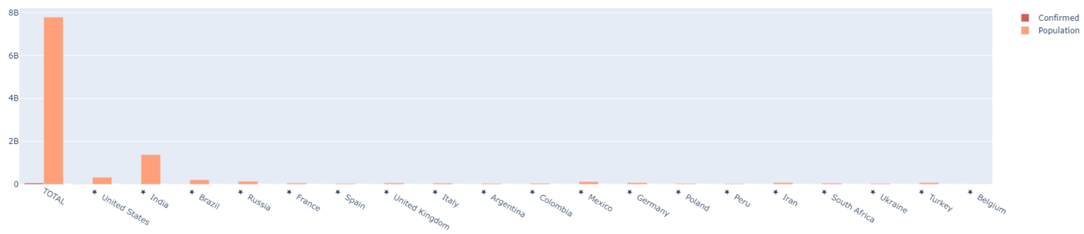


- China on first position that was unexpected. I was expecting United States.
- As you can see the countries who are vastly testing their people have a upper hand on curbing the spread of virus by implementing policies.

lets explore the Confirmed cases in relation to total population

[27]:
```
# lets visualize the confirmed case based in relation to total population

import plotly.graph_objects as go


# for grouped barplot using confirmed cases per country and population per␣
↪country.

fig = go.Figure(data = [
go.Bar(
    x = world_df['Name'],
    y = world_df["Confirmed"].head(20),
    name = "Confirmed",
    marker_color = "indianred"
),
go.Bar(
```

```
    x = world_df['Name'],
    y = world_df['Population'].head(20),
    name = 'Population',
    marker_color = "lightsalmon"
)
])

# Here we modify the tickangle of the xaxis, resulting in rotated labels.
fig.update_layout(barmode='group')
fig.show()
```

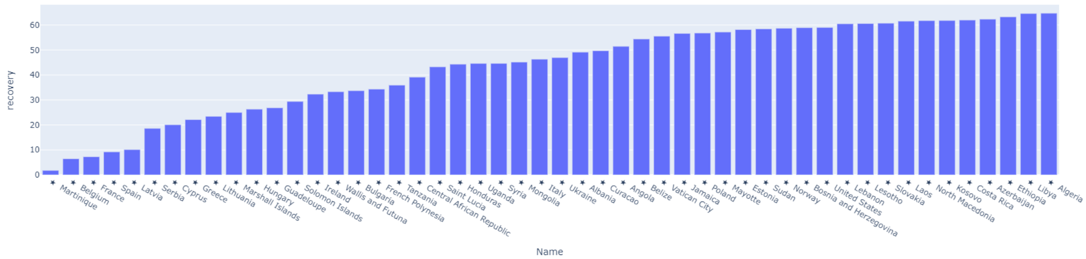


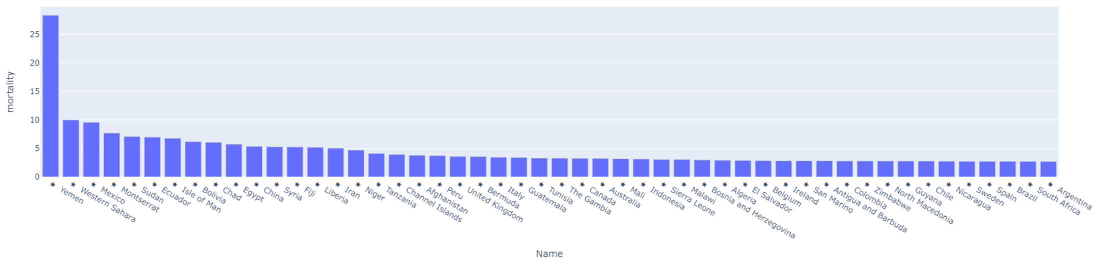
- This graph shows a small percentage of people are affected by the novel coronavirus. People who are at high risk for severe illness from COVID-19

[28]:
```
#Mortality calculation

world_df['mortality'] = world_df[['Confirmed','Deceased']].apply(lambda x: (x['Deceased']*100/x['Confirmed']),axis=1 )

#Recovery calculation

world_df['Recovered'] = pd.to_numeric(world_df['Recovered'],errors='coerce')
world_df['recovery'] = world_df[['Confirmed','Recovered']].apply(lambda x: (x['Recovered']*100/x['Confirmed']),axis=1 )
```

[29]:
```
def recovery_mortality_plot():

    name = ['recovery','mortality']
    Value=[True,False]

    for i,j in zip(name,Value):

        world_fig = px.bar(world_df.sort_values(i, ascending=j).head(50), x = 'Name' , y = i)
        world_fig.show()
```

```
recovery_mortality_plot()
```





- **Martinique, Belgium, France** has lowest recovery rate among countries.
- Yemen has highest mortality rate **~30%**, which is one of the highest in the world and five times the global average. Covid-19: Deaths in Yemen are five times global average as healthcare collapses

Lets plot world data using Choropleth Map

[30]:
```
#something worng with the country names. plotly uses standard ISO-3_codes. Lets
↪try to create a column for country codes

print("{} countries in the list.". format(world_df['Name'].nunique()))
```

220 countries in the list.

The country converter (coco) - a Python package for converting country names between different classifications schemes. For more info please click here.

[31]:
```
import country_converter as coco
```

[32]:
```
# Creating a list and appending all the names from world_df column.

Names = []
for i in range(1,215):
```

```
    Names.append(world_df.iloc[i]['Name'][3:])

# Insert Total at index 0. we left that because it doesn't contain any start in␣
 ↪it.

Names.insert(0,'TOTAL')
```

[33]:
```
standard_names = coco.convert(names= Names, to='ISO3')
```

```
WARNING:root:TOTAL not found in regex
WARNING:root:Channel Islands not found in regex
WARNING:root:SÃ£o TomÃ© and PrÃncipe not found in regex
```

[34]:
```
map_data = world_df[world_df['Name']!='TOTAL']
print(map_data.nunique())
print(len(map_data))
```

```
Name                        219
Confirmed                   216
Per Million                 154
Changes Today               100
Percentage Day Change        93
Critical                     96
Deceased                    171
Per Million.1               115
Changes Today.1              53
Percentage Death Change      75
Tests                       201
Active                      172
Recovered                   209
Per Million.2               151
Population                  219
mortality                   194
recovery                    210
dtype: int64
219
```

[35]:
```
# Adding the ISO3 code in a new world_df['Code'] column.

map_data = map_data[:215]
map_data['code'] = standard_names

map_data['code'] = map_data['code'].shift(-1)

# Removing countries of which ISO3 code is not available

choropleth_data = map_data[map_data['code'] != "NaN"]
```

```
# choropleth_data
```

[36]:

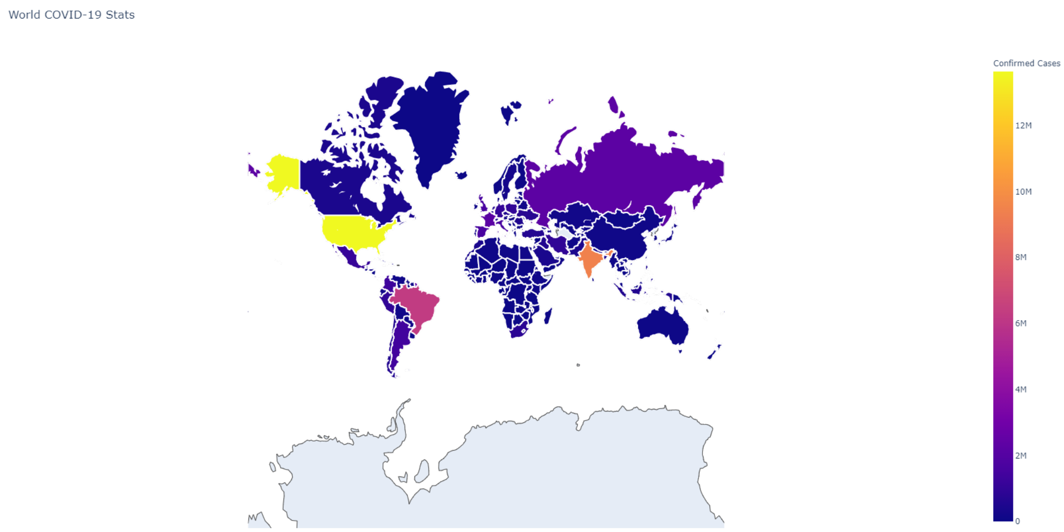
```
#lets again try to plot the data using choropleth dataframe.

# For using choropleth first we have to make a dictionary

data = dict(
        type = 'choropleth',
        locations = choropleth_data['code'],
        z = choropleth_data['Confirmed'],
        text = choropleth_data['Deceased'],
        marker = dict(line = dict(color = 'rgb(255,255,255)',width = 2)),
        colorbar = {'title' : "Confirmed Cases"}
        )
```

[37]:

```
# Now create a layout for the graph

layout = dict(

    title = 'World COVID-19 Stats',
    width=1080,
    height=900,
    geo = dict(
        showframe = False,
        projection = {'type':'mercator'}
    )
    )
```

[38]:

```
# Finally we will pass both layout and data dictionary to generate the map.
choromap = go.Figure(data = [data],layout = layout)
choromap.show()
```



2.2 Canada COVID-19 Stats

Lets get Latest Canada's information.

1. We'll use the pandas read.html which lets us read the webpage table without much of complexity.
2. We can also use the lsit to convert it to a dataframe.
3. In the header of the list generated you see a number ``1'', which was used in the original website as a filter for arranging data in ascending or descending order.

[39]:
```
#grabbing latest canada specific data

url = "https://ncov2019.live/data/canada"

r = requests.get(url)
df_list = pd.read_html(r.text) # this parses all the tables in webpages to a␣
↪list
canada_df = df_list[2]
canada_df
```

[39]:
```
                         Name  Confirmed Per Million  Changes Today  \
0                       TOTAL     368266     Unknown              0
1                     Alberta      54836     Unknown              0
2            British Columbia      30884     Unknown              0
3                    Manitoba      16118     Unknown              0
4               New Brunswick        481     Unknown              0
5   Newfoundland and Labrador        333     Unknown              0
```

6	Northwest Territories	15	Unknown	0
7	Nova Scotia	1257	Unknown	0
8	Nunavut	164	Unknown	0
9	Ontario	116517	Unknown	0
10	Prince Edward Island	72	Unknown	0
11	Quebec	139643	Unknown	0
12	Repatriated Travellers	13	Unknown	0
13	Saskatchewan	7888	Unknown	0
14	Yukon	45	Unknown	0

	Percentage Day Change	Critical	Deceased	Per Million.1	Changes Today.1	\
0	0%	450	12012	Unknown	0	
1	0%	Unknown	524	Unknown	0	
2	0%	Unknown	395	Unknown	0	
3	0%	Unknown	290	Unknown	0	
4	0%	Unknown	7	Unknown	0	
5	0%	Unknown	4	Unknown	0	
6	0%	Unknown	Unknown	Unknown	0	
7	0%	Unknown	65	Unknown	0	
8	0%	Unknown	Unknown	Unknown	0	
9	0%	Unknown	3640	Unknown	0	
10	0%	Unknown	Unknown	Unknown	0	
11	0%	Unknown	7021	Unknown	0	
12	0%	Unknown	Unknown	Unknown	0	
13	0%	Unknown	45	Unknown	0	
14	0%	Unknown	1	Unknown	0	

	Percentage Death Change	Tests	Active	Recovered	Per Million.2	\
0	0%	11344925	60811	295462	Unknown	
1	0%	Unknown	Unknown	39381	Unknown	
2	0%	Unknown	Unknown	21304	Unknown	
3	0%	Unknown	Unknown	6804	Unknown	
4	0%	Unknown	Unknown	363	Unknown	
5	0%	Unknown	Unknown	297	Unknown	
6	0%	Unknown	Unknown	15	Unknown	
7	0%	Unknown	Unknown	1078	Unknown	
8	0%	Unknown	Unknown	33	Unknown	
9	0%	Unknown	Unknown	100650	Unknown	
10	0%	Unknown	Unknown	68	Unknown	
11	0%	Unknown	Unknown	120906	Unknown	
12	0%	Unknown	Unknown	13	Unknown	
13	0%	Unknown	Unknown	4521	Unknown	
14	0%	Unknown	Unknown	29	Unknown	

	Population
0	Unknown
1	Unknown

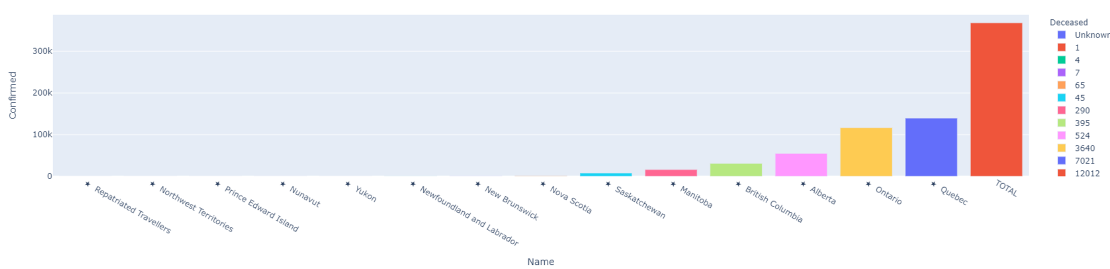
```
2       Unknown
3       Unknown
4       Unknown
5       Unknown
6       Unknown
7       Unknown
8       Unknown
9       Unknown
10      Unknown
11      Unknown
12      Unknown
13      Unknown
14      Unknown
```

2.3 Canada COVID-19 Stats

Lets visualize Canada's Data and see which province has been worst effected.

1. We'll use the same above canada_df for visualization purpose.
2. We are going to use this dataframe because it's the latest data and our script we'll update the data every time we run the cell based on the website mentioned above.
3. I'm going to use plotly for visualization purpose as it generates graphs which are interactive and user friendly.

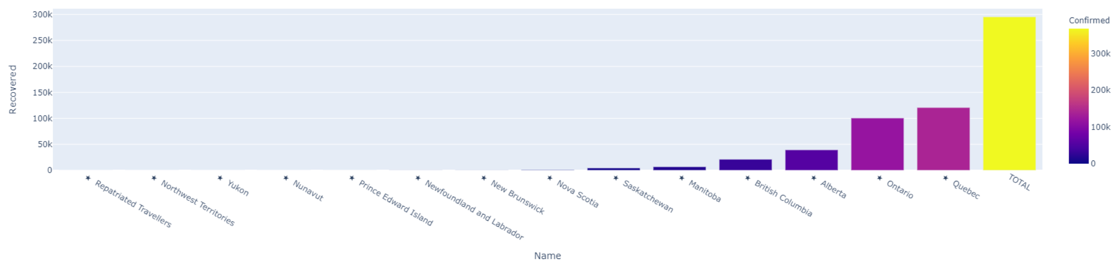
[40]:
```
canada_fig = px.bar(canada_df.
↪sort_values('Confirmed'),x='Name',y='Confirmed',color="Deceased")
canada_fig.show()
```



- Quebec has maximum number of confirmed cases and twice as many deceased people than ontario. Quebec leads Canada in Coronavirus deaths
- In this article I also found one more interesting thing that Alberta has done more testing per capita, and along with good policies the death polls remains below 500.
- There a some provinces where there were less to no cases, and no death has been reported, because quite a few people live there.

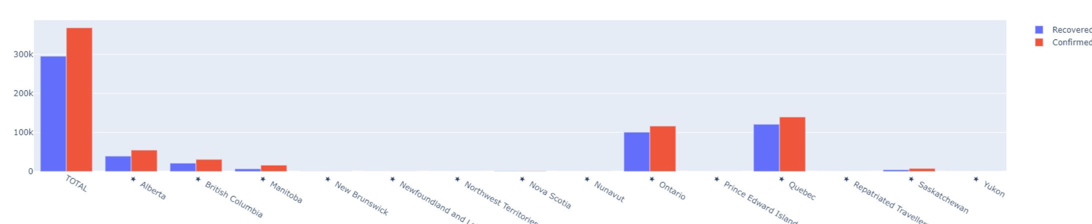
Lets see relation between total confirmed cases to recovered cases.

[41]:
```
canada_fig = px.bar(canada_df.sort_values('Recovered'), x = 'Name', y =␣
↪'Recovered',color='Confirmed')
canada_fig.show()
```



Lets calculate recovery rate in Canada and Alberta specifically

[42]:
```
fig = go.Figure(data = [
    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Recovered'],
    name = "Recovered"
    ),

    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Confirmed'],
    name = "Confirmed"
    )
])

fig.update_layout(barmode = "group")
fig.show()
```

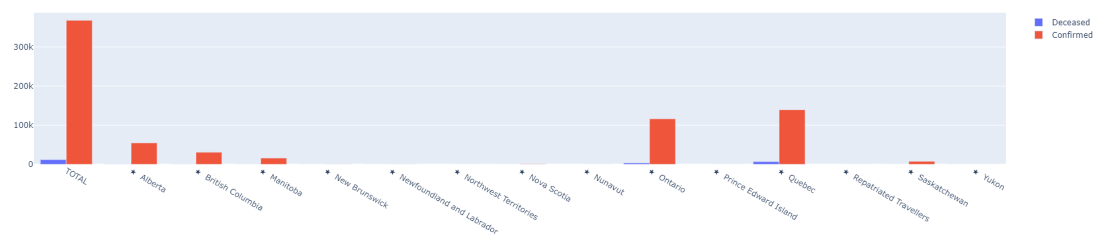


- Recovery rate canada wide is 88% which is 21% higher than the worldwide

recovery rate. This also brings in another factor the geographical location a patient is in and how is the healthcare system there.
- Alberta's recovery rate is also 89% which is close to overall recovery rate.

lets calculate mortality rate.

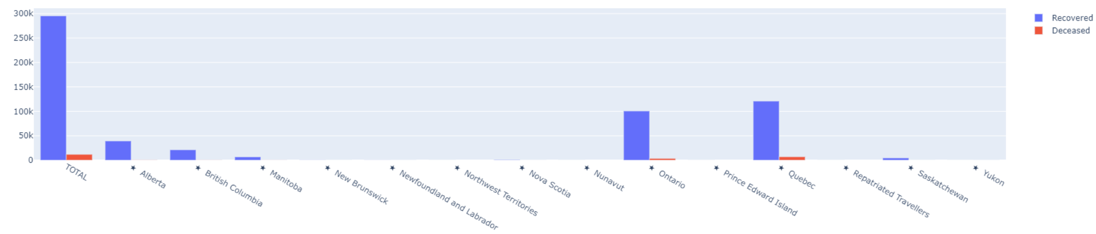
[43]:
```
fig = go.Figure(data = [
    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Deceased'],
    name = "Deceased"
    ),

    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Confirmed'],
    name = "Confirmed"
    )
])

fig.update_layout(barmode = "group")
fig.show()
```



- Mortality rate of overall canada is 7% (9118/126.804k)
- Mortality rate of Alberta is 1.8% which is quite astounding. Alberta is implementing policies very efficiently and because of that it has such a low mortality rate.
- Highest mortality rate is of Quebec 8.9%.
- Second highest mortality rate is of ontario 6.5%

[44]:
```
fig = go.Figure(data = [
    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Recovered'],
    name = "Recovered"
    ),
```

```
    go.Bar(
    x = canada_df['Name'],
    y = canada_df['Deceased'],
    name = "Deceased"
    )
])

fig.update_layout(barmode = "group")
fig.show()
```

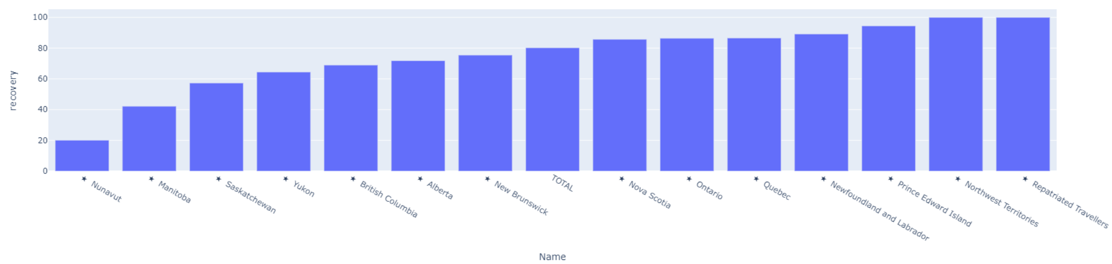


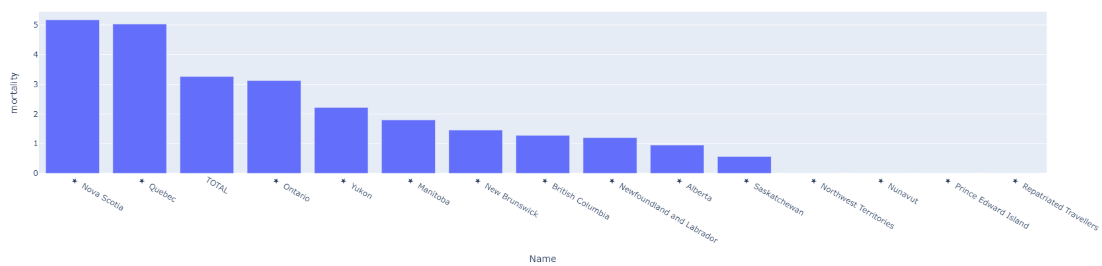
[45]:
```
# converting columns to int64 format from object dtype
canada_df['Deceased'].replace({'Unknown':0},inplace=True)
canada_df[['Deceased','Recovered']] = canada_df[['Deceased','Recovered']].
 ↪apply(pd.to_numeric,errors='ignore')
```

[46]:
```
#Mortality calculation

canada_df['mortality'] = canada_df[['Confirmed','Deceased']].apply(lambda x:␣
 ↪(x['Deceased']*100/x['Confirmed']),axis=1 )

#Recovery calculation

canada_df['Recovered'] = pd.to_numeric(canada_df['Recovered'],errors='coerce')
canada_df['recovery'] = canada_df[['Confirmed','Recovered']].apply(lambda x:␣
 ↪(x['Recovered']*100/x['Confirmed']),axis=1 )
```

[47]:
```
def recovery_mortality_plot():

    name = ['recovery','mortality']
    Value=[True,False]

    for i,j in zip(name,Value):

        canada_fig = px.bar(canada_df.sort_values(i, ascending=j).head(50), x =␣
 ↪'Name' , y = i)
```

```
        canada_fig.show()

recovery_mortality_plot()
```





- From these graphs we can see that overall recovery rate for canada is more than **~84%**. **Alberta** is very close with recovery rate of **~83%**.
- **Manitoba** has very lowest recovery rate **~50%**. Highest recovery rate is in **PEI**, which can be attributed to low population.
- Average mortality rate is close to **~4.5%**.
- Highest mortality rate is observerd in **Quebec**. **Alberta** is in bottom **5** in terms of mortality rate.

2.4 Model for predicting the number of confirmed cases.

[48]:
```
# import confirmed cases data

confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/
 ↪COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/
 ↪time_series_covid19_confirmed_global.csv')

#Getting all the dates
cols = confirmed_df.keys()
confirmed = confirmed_df.loc[:4,cols[4]:cols[-1]]
```

```
dates = confirmed.keys()
```

[49]:
```
worldcases = []

for i in ((dates)):

    confirmed_sum = confirmed[i].sum()

    worldcases.append(confirmed_sum)
```

[50]:
```
import numpy as np
import random
import math
import time
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error
import datetime
```

Future Forecasting

[51]:
```
days_in_future = 10
future_forcast = np.array([i for i in range(len(dates)+days_in_future)]).
↪reshape(-1, 1)
adjusted_dates = future_forcast[:-10]
```

Convert integer into datetime for better visualization

[52]:
```
start = '1/20/2020'
start_date = datetime.datetime.strptime(start, '%m/%d/%Y')
future_forcast_dates = []
for i in range(len(future_forcast)):
    future_forcast_dates.append((start_date + datetime.timedelta(days=i)).
↪strftime('%m/%d/%Y'))
```

[53]:
```
days_from_1_20 = np.array([i for i in range(len(dates))]).reshape(-1,1)
```

Train Test Split

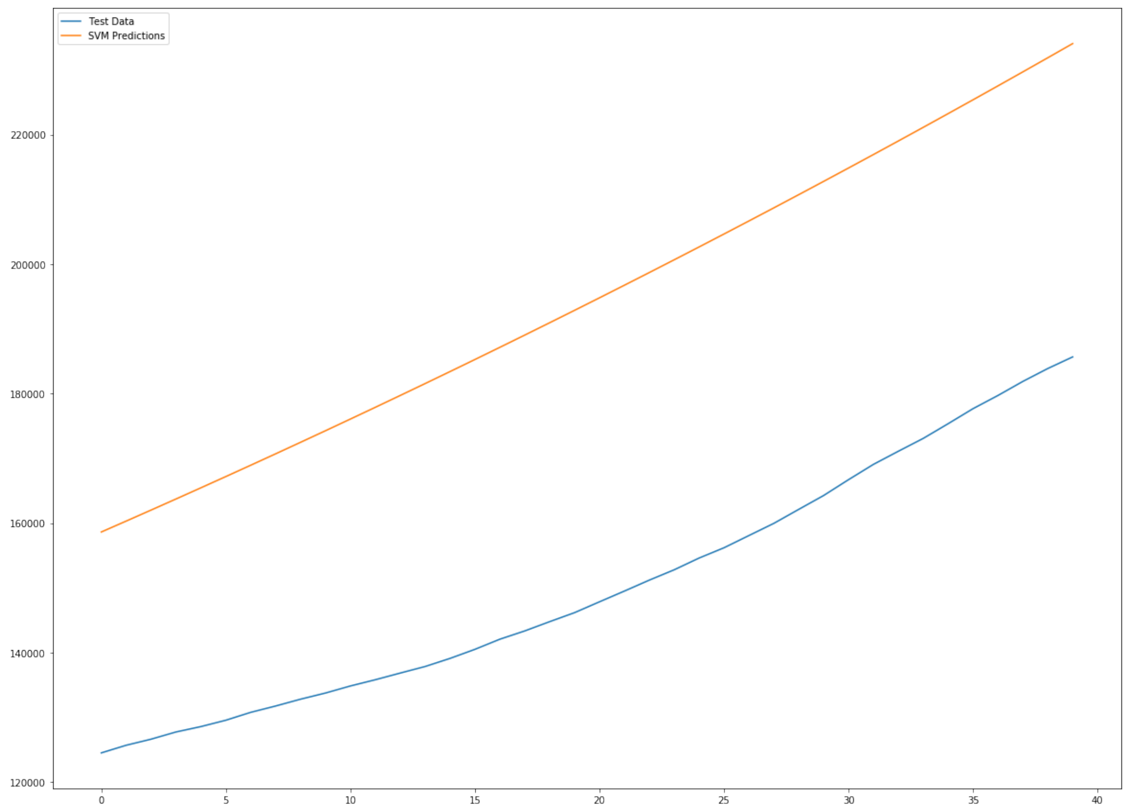
[54]:
```
X_train_confirmed, X_test_confirmed, y_train_confirmed, y_test_confirmed =␣
↪train_test_split(days_from_1_20[50:], worldcases[50:], test_size=0.15,␣
↪shuffle=False)
```

2.5 Support Vector Machine Model

[55]:
```
svm_confirmed = SVR(shrinking=True, kernel='poly',gamma=0.
↪01,epsilon=1,degree=3,C=0.1)
svm_confirmed.fit(X_train_confirmed,y_train_confirmed)
svm_pred = svm_confirmed.predict(future_forcast)
```

[56]:
```
svm_test_pred = svm_confirmed.predict(X_test_confirmed)
plt.figure(figsize=(20,15))
plt.plot(y_test_confirmed)
plt.plot(svm_test_pred)
plt.legend(['Test Data', 'SVM Predictions'])
print('MAE:', mean_absolute_error(svm_test_pred, y_test_confirmed))
print('MSE:',mean_squared_error(svm_test_pred, y_test_confirmed))
```

```
MAE: 44363.19515992949
MSE: 1989450718.2306314
```



Mean Absolute percentage error I prefer to use mean absolute percent error because it gives an simple percentage to communicate that shows how off the predictions are. MAPE is not included in Sklearn, so a custom feature must be used.

[57]:
```
def mean_absolute_percentage_error(y_true, y_pred):
    """Calculates MAPE given y_true and y_pred"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

[58]:
```
print('Mean absolute percentage error of SVM is␣
↪',mean_absolute_percentage_error(y_test_confirmed,svm_test_pred))
```

```
Mean absolute percentage error of SVM is  29.61332357413434
```

2.6 Linear Regression model

[59]:
```
# transform our data for polynomial regression
poly = PolynomialFeatures(degree=5)
poly_X_train_confirmed = poly.fit_transform(X_train_confirmed)
poly_X_test_confirmed = poly.fit_transform(X_test_confirmed)
poly_future_forcast = poly.fit_transform(future_forcast)
```

[60]:
```
# polynomial regression
linear_model = LinearRegression(normalize=True, fit_intercept=False)
linear_model.fit(poly_X_train_confirmed, y_train_confirmed)
test_linear_pred = linear_model.predict(poly_X_test_confirmed)
linear_pred = linear_model.predict(poly_future_forcast)
print('MAE:', mean_absolute_error(test_linear_pred, y_test_confirmed))
print('MSE:',mean_squared_error(test_linear_pred, y_test_confirmed))
```

```
MAE: 7745.815418206202
MSE: 83331043.02730493
```

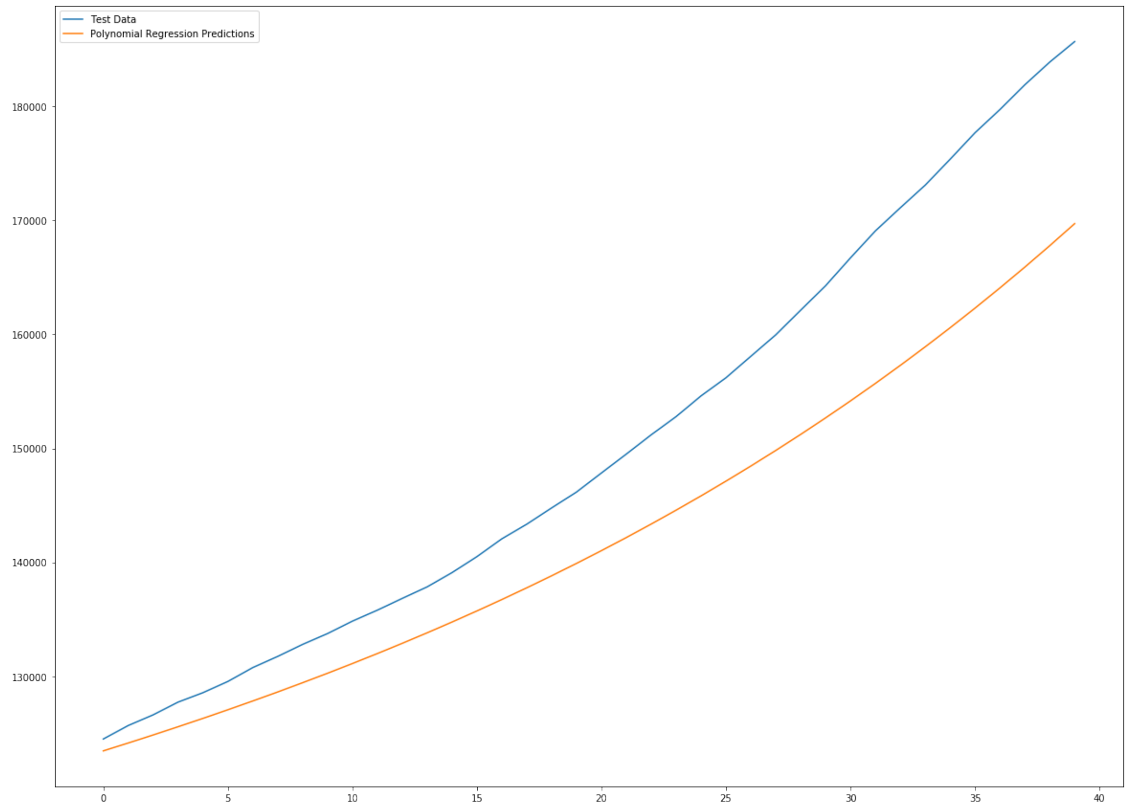
Mean Absolute percentage error

[61]:
```
print('Mean absolute percentage error of LR is ',␣
↪mean_absolute_percentage_error(y_test_confirmed,test_linear_pred))
```

```
Mean absolute percentage error of LR is  4.8453251749105295
```

[62]:
```
plt.figure(figsize=(20,15))
plt.plot(y_test_confirmed)
plt.plot(test_linear_pred)
plt.legend(['Test Data', 'Polynomial Regression Predictions'])
```

[62]:
```
<matplotlib.legend.Legend at 0x16333185fc8>
```



[63]:
```
# confirmed_df.head()
```

[64]:
```
# Transposing the row for time series analysis

confirmed_df = confirmed_df.T
confirmed_df = confirmed_df.rename(columns=confirmed_df.iloc[1])
# confirmed_df
```

[65]:
```
confirmed_df = confirmed_df[4:]
# confirmed_df
```

[66]:
```
confirmed_df['Total_cases'] = confirmed_df.sum(axis=1)
```

[67]:
```
# converting the index column to date

confirmed_df.reset_index(level=0,inplace=True)
# confirmed_df
```

[68]:
```
confirmed_df['dates'] = pd.to_datetime(confirmed_df['index'])
# confirmed_df.info()
```

[69]:
```
time_series_analysis_df = confirmed_df[['Total_cases','dates']]
# time_series_analysis_df
```

[70]:
```
# Now we will set the dates column as the index of the dataframe to allow us␣
↪really explore the our data.

time_series_analysis_df = time_series_analysis_df.set_index('dates')
time_series_analysis_df
```

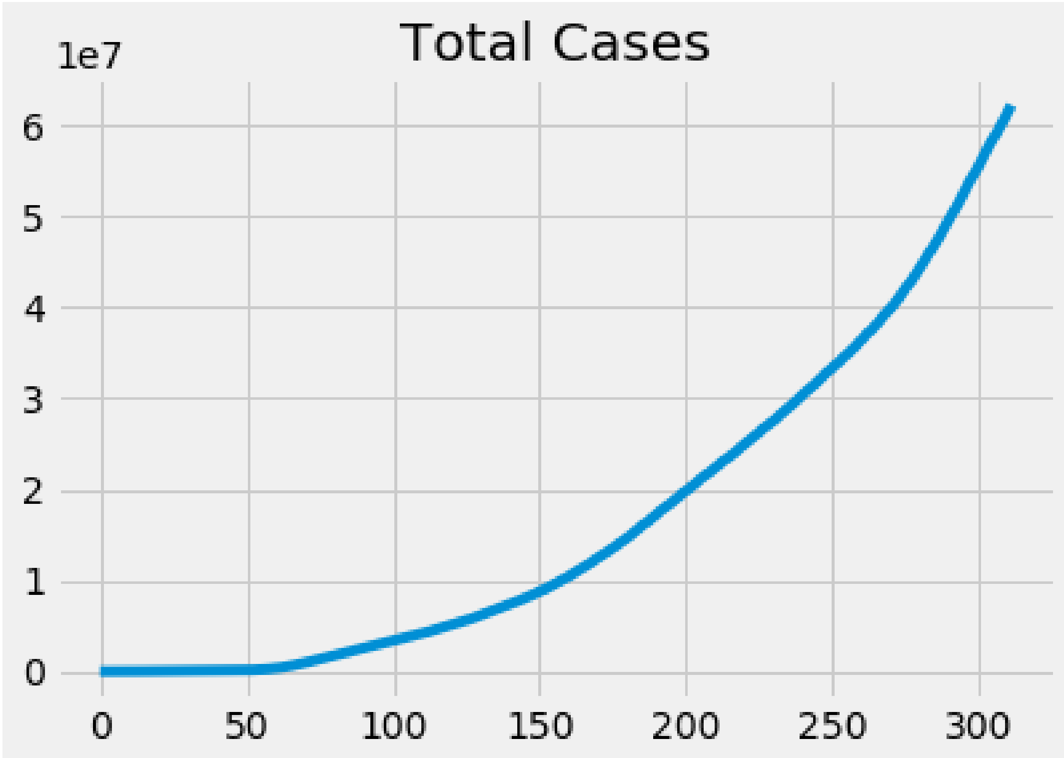
[70]:
```
             Total_cases
dates
2020-01-22         555.0
2020-01-23         654.0
2020-01-24         941.0
2020-01-25        1434.0
2020-01-26        2118.0
…                    …
2020-11-24    59759508.0
2020-11-25    60392453.0
2020-11-26    60973650.0
2020-11-27    61645535.0
2020-11-28    62244181.0

[312 rows x 1 columns]
```

XGBoost

[71]:
```
from pandas import read_csv
from matplotlib import pyplot
import xgboost as xgb
from xgboost import plot_importance, plot_tree
plt.style.use('fivethirtyeight')
```

[74]:
```
path = r'C:
↪\Users\yrsin\Desktop\Project\CanadaCovid\Graduate-Project\covid_data\Data\Covid-19'
time_series_analysis_df.to_csv(path+'\series.csv')
```

[75]:
```
# load dataset
series = pd.read_csv('../covid_data/Data/Covid-19/series.csv', header=0,␣
↪index_col=0)
values = series.values
# plot dataset
pyplot.plot(values)
pyplot.title('Total Cases')
pyplot.show()
```



- We are using the XGBoost model on the dataset when making one-step forecasts for the data from September month.
- We will use previous 10 time steps as input to the model and default model hyperparameters, except we will change the loss to `reg:sqarederror' and use 1,000 trees in the ensemble.

[76]:
```
# forecast monthly births with xgboost
from numpy import asarray
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.metrics import mean_absolute_error
from xgboost import XGBRegressor
from matplotlib import pyplot

# transform a time series dataset into a supervised learning dataset
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[1]
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
```

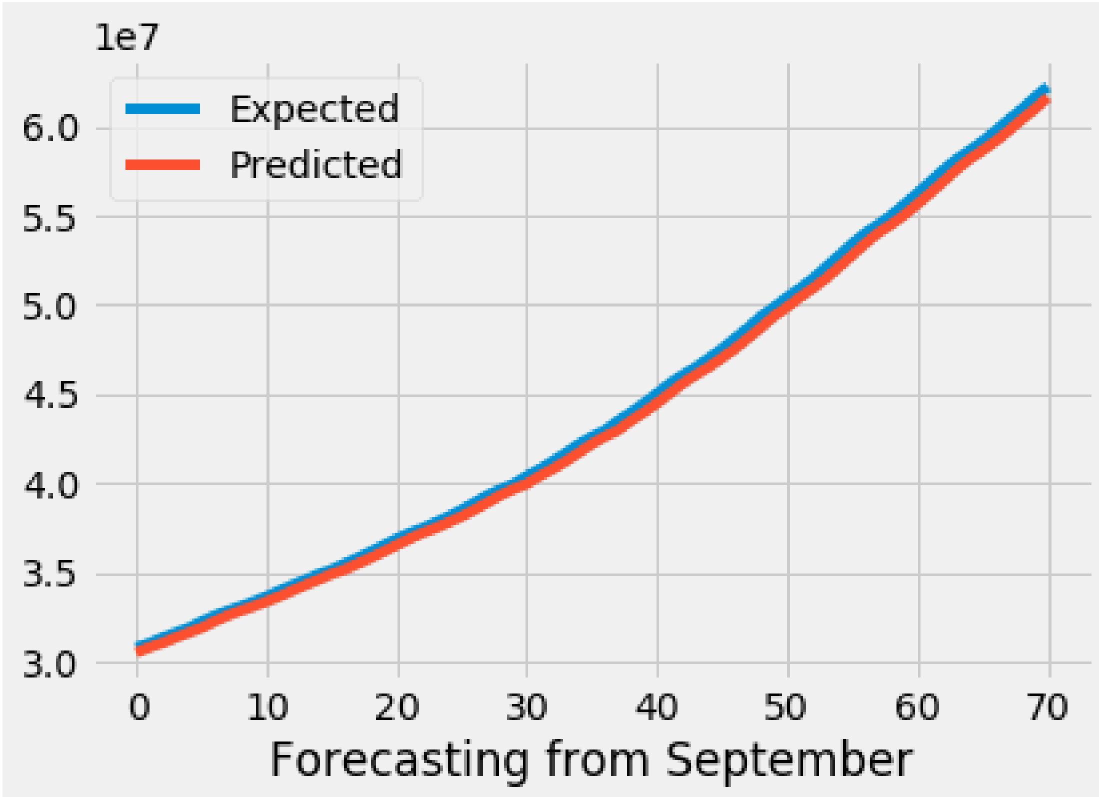
```
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
	# put it all together
	agg = concat(cols, axis=1)
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg.values

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:-n_test, :], data[-n_test:, :]

# fit an xgboost model and make a one step prediction
def xgboost_forecast(train, testX):
	# transform list into array
	train = asarray(train)
	# split into input and output columns
	trainX, trainy = train[:, :-1], train[:, -1]
	# fit model
	model = XGBRegressor(objective='reg:squarederror', n_estimators=1000)
	model.fit(trainX, trainy)
	# make a one-step prediction
	yhat = model.predict(asarray([testX]))
	return yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test):
	predictions = list()
	# split dataset
	train, test = train_test_split(data, n_test)
	# seed history with training dataset
	history = [x for x in train]
	# step over each time-step in the test set
	for i in range(len(test)):
		# split test row into input and output columns
		testX, testy = test[i, :-1], test[i, -1]
		# fit model on history and make a prediction
		yhat = xgboost_forecast(history, testX)
		# store forecast in list of predictions
		predictions.append(yhat)
		# add actual observation to history for the next loop
		history.append(test[i])

		print('>expected=%.1f, predicted=%.1f' % (testy, yhat))
	# estimate prediction error
```

```
        error = mean_absolute_error(test[:, -1], predictions)
        return error, test[:, -1], predictions

# load the dataset
series = read_csv('../covid_data/Data/Covid-19/series.csv', header=0,␣
 ↪index_col=0)
values = series.values
# transform the time series data into supervised learning
data = series_to_supervised(values, n_in=10)
# evaluate
mae, y, yhat = walk_forward_validation(data, 71)
print('MAE: %.3f' % mae)
# plot expected vs preducted
pyplot.plot(y, label='Expected')
pyplot.plot(yhat, label='Predicted')
pyplot.xlabel("Forecasting from September")
pyplot.legend()
pyplot.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\data.py:96: UserWarning:

Use subset (sliced data) of np.ndarray is not recommended because it will
generate extra copies and increase memory consumption


>expected=30773352.0, predicted=30493778.0
>expected=31016379.0, predicted=30773346.0
>expected=31314834.0, predicted=31016374.0
>expected=31594602.0, predicted=31314828.0
>expected=31861824.0, predicted=31594596.0
>expected=32224421.0, predicted=31861818.0
>expected=32552288.0, predicted=32224410.0
>expected=32829197.0, predicted=32552278.0
>expected=33070312.0, predicted=32829186.0
>expected=33346336.0, predicted=33070306.0
>expected=33631070.0, predicted=33346330.0
>expected=33957884.0, predicted=33631060.0
>expected=34275807.0, predicted=33957872.0
>expected=34571172.0, predicted=34275796.0
>expected=34889804.0, predicted=34571160.0
>expected=35138269.0, predicted=34889792.0
>expected=35467660.0, predicted=35138252.0
>expected=35793012.0, predicted=35467648.0
>expected=36142566.0, predicted=35793000.0
>expected=36504174.0, predicted=36142548.0
>expected=36863620.0, predicted=36504164.0
>expected=37193872.0, predicted=36863608.0
>expected=37463568.0, predicted=37193860.0
```

```
>expected=37788829.0, predicted=37463556.0
>expected=38117691.0, predicted=37788816.0
>expected=38498286.0, predicted=38117680.0
>expected=38905694.0, predicted=38498276.0
>expected=39316324.0, predicted=38905684.0
>expected=39657368.0, predicted=39316312.0
>expected=39943263.0, predicted=39657348.0
>expected=40391772.0, predicted=39943244.0
>expected=40780443.0, predicted=40391760.0
>expected=41224485.0, predicted=40780432.0
>expected=41696807.0, predicted=41224472.0
>expected=42192016.0, predicted=41696788.0
>expected=42603760.0, predicted=42192004.0
>expected=42957052.0, predicted=42603748.0
>expected=43494699.0, predicted=42957032.0
>expected=43964015.0, predicted=43494688.0
>expected=44473376.0, predicted=43964004.0
>expected=45024129.0, predicted=44473364.0
>expected=45594086.0, predicted=45024116.0
>expected=46070463.0, predicted=45594076.0
>expected=46503704.0, predicted=46070452.0
>expected=47012165.0, predicted=46503692.0
>expected=47527270.0, predicted=47012152.0
>expected=48124824.0, predicted=47527260.0
>expected=48718539.0, predicted=48124812.0
>expected=49360492.0, predicted=48718528.0
>expected=49871598.0, predicted=49360480.0
>expected=50436547.0, predicted=49871588.0
>expected=50938348.0, predicted=50436536.0
>expected=51494283.0, predicted=50938336.0
>expected=52139269.0, predicted=51494272.0
>expected=52786566.0, predicted=52139256.0
>expected=53435351.0, predicted=52786556.0
>expected=54029326.0, predicted=53435340.0
>expected=54502332.0, predicted=54029316.0
>expected=55030781.0, predicted=54502320.0
>expected=55638883.0, predicted=55030768.0
>expected=56262553.0, predicted=55638872.0
>expected=56913120.0, predicted=56262540.0
>expected=57579266.0, predicted=56913108.0
>expected=58165570.0, predicted=57579252.0
>expected=58649369.0, predicted=58165556.0
>expected=59171092.0, predicted=58649356.0
>expected=59759508.0, predicted=59171080.0
>expected=60392453.0, predicted=59759496.0
>expected=60973650.0, predicted=60392440.0
>expected=61645535.0, predicted=60973636.0
>expected=62244181.0, predicted=61645524.0
```

MAE: 447200.310



[77]:
```
print('Mean absolute percentage error of XGBosst␣
↪is',mean_absolute_percentage_error(y,yhat))
```

Mean absolute percentage error of XGBosst is 0.9998780813740501

2.7 Persistance model for timeseries forecasting

The persistence forecast is where the observation from the prior time step (t-1) is used to predict the observation at the current time step (t).

We can implement this by taking the last observation from the training data and history accumulated by walk-forward validation and using that to predict the current time step.

[78]:
```
from pandas import concat
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
```

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy
from pandas import datetime
```

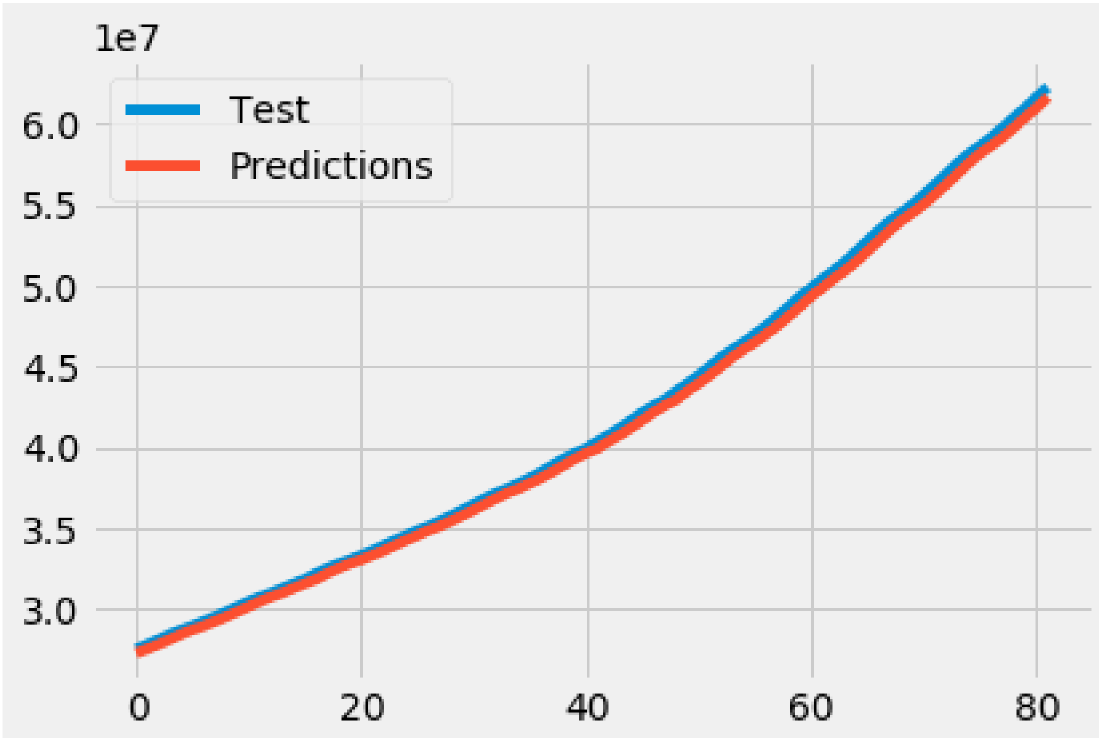
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:6:
FutureWarning:

The pandas.datetime class is deprecated and will be removed from pandas in a
future version. Import from datetime module instead.

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:15:
FutureWarning:

The pandas.datetime class is deprecated and will be removed from pandas in a
future version. Import from datetime module instead.
```

[79]:

```
X = time_series_analysis_df.values
train, test = X[0:230], X[230:]
print(train.shape, test.shape)

# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
        # make prediction
        predictions.append(history[-1])
        # observation
        history.append(test[i])
# report performance
rmse = sqrt(mean_squared_error(test, predictions))

print('RMSE: %.3f' % rmse)
print(mean_absolute_percentage_error(test,predictions))
from matplotlib import pyplot

# line plot of observed vs predicted
pyplot.plot(test,label='Test')
pyplot.plot(predictions, label = 'Predictions')
plt.legend(labels=('Test','Predictions'))
pyplot.show()
```

```
(230, 1) (82, 1)
```

```
RMSE: 446478.714
0.9986563659207627
```



LSTM in Keras The Long Short-Term Memory network (LSTM) is a type of Recurrent Neural Network (RNN).

A benefit of this type of network is that it can learn and remember over long sequences and does not rely on a pre-specified window lagged observation as input.

In Keras, this is referred to as stateful, and involves setting the ``stateful'' argument to ``True'' when defining an LSTM layer.

By default, an LSTM layer in Keras maintains state between data within one batch. A batch of data is a fixed-sized number of rows from the training dataset that defines how many patterns to process before updating the weights of the network. State in the LSTM layer between batches is cleared by default, therefore we must make the LSTM stateful. This gives us fine-grained control over when state of the LSTM layer is cleared, by calling the reset_states() function.

The LSTM layer expects input to be in a matrix with the dimensions: [samples, time steps, features].

Samples: These are independent observations from the domain, typically rows of data. Time steps: These are separate time steps of a given variable for a given observation. Features: These are separate measures observed at the time of observation. We have some flexibility in how the Total cases dataset is framed for the network. We will keep it simple and frame the problem as each time step in the original sequence is one separate sample, with one timestep and one feature.

Given that the training dataset is defined as X inputs and y outputs, it must be reshaped into the Samples/TimeSteps/Features format, for example:

Batch Size: 1 Epochs: 1500 Neurons: 1

[80]:
```
def parser(x):
    return datetime.strptime(x,'%Y-%m-%d')

def timeseries_to_supervised(data, lag=1):
    df = pd.DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df

def difference(dataset, interval=1):
        diff = list()
        for i in range(interval, len(dataset)):
                value = dataset[i] - dataset[i - interval]
                diff.append(value)
        return Series(diff)

def inverse_difference(history, yhat, interval=1):
        return yhat + history[-interval]

def scale(train, test):
        # fit scaler
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(train)
        # transform train
        train = train.reshape(train.shape[0], train.shape[1])
        train_scaled = scaler.transform(train)
        # transform test
        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)
        return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
        new_row = [x for x in X] + [value]
        array = numpy.array(new_row)
        array = array.reshape(1, len(array))
```

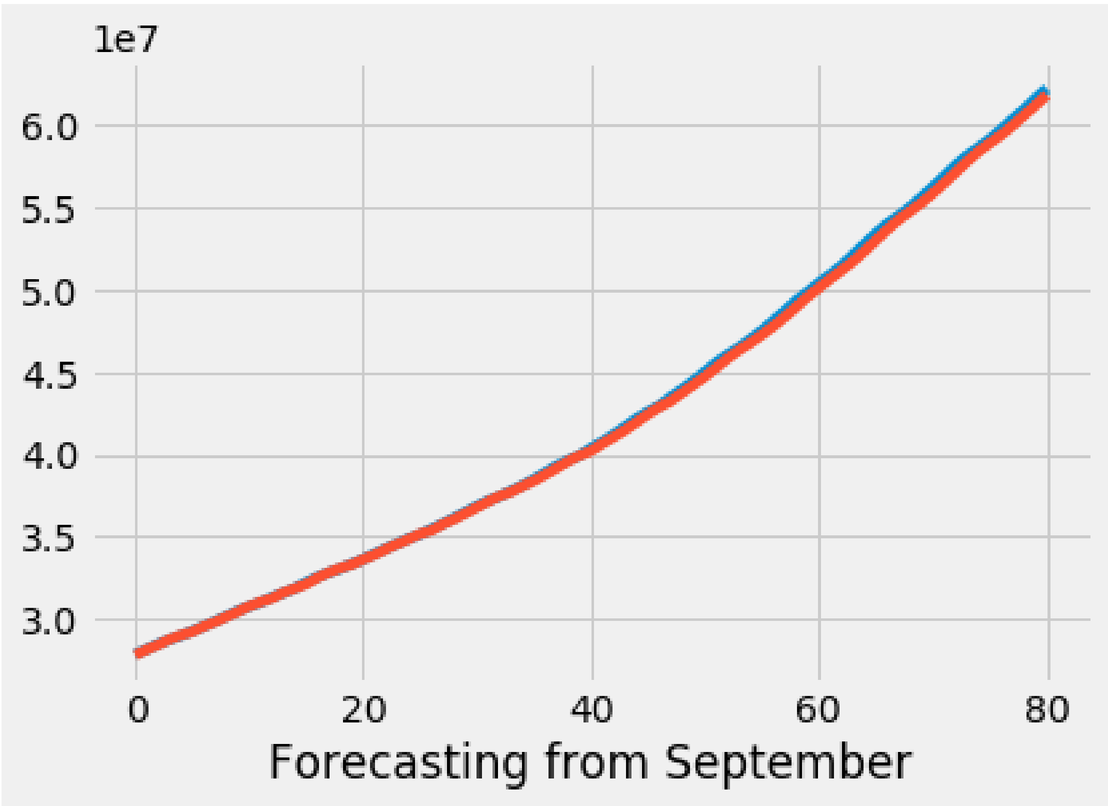
```
        inverted = scaler.inverse_transform(array)
        return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
        X, y = train[:, 0:-1], train[:, -1]
        X = X.reshape(X.shape[0], 1, X.shape[1])
        model = Sequential()
        model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.
 ↪shape[2]), stateful=True))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        for i in range(nb_epoch):
                model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0,␣
 ↪shuffle=False)
                model.reset_states()
        return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]


series = pd.read_csv('../covid_data/Data/Covid-19/series.csv', header = 0,␣
 ↪parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
print(series.head())


raw_values = series.values
diff_values = difference(raw_values, 1)

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, 1)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:230], supervised_values[230:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, 1, 1500, 1)
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
```

```
lstm_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
        # make one-step forecast
        X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
        yhat = forecast_lstm(lstm_model, 1, X)
        # invert scaling
        yhat = invert_scale(scaler, X, yhat)
        # invert differencing
        yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
        # store forecast
        predictions.append(yhat)
        expected = raw_values[len(train) + i + 1]
        print('day=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# report performance
rmse = sqrt(mean_squared_error(raw_values[231:], predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[231:])
pyplot.plot(predictions)
pyplot.xlabel('Forecasting from September')
pyplot.show()
```

```
dates
2020-01-22     555.0
2020-01-23     654.0
2020-01-24     941.0
2020-01-25    1434.0
2020-01-26    2118.0
Name: Total_cases, dtype: float64
day=1, Predicted=27820999.318522, Expected=27855918.000000
day=2, Predicted=28134953.489757, Expected=28154919.000000
day=3, Predicted=28431875.023008, Expected=28474744.000000
day=4, Predicted=28758014.147335, Expected=28751941.000000
day=5, Predicted=29017857.059897, Expected=28987746.000000
day=6, Predicted=29226084.457356, Expected=29267418.000000
day=7, Predicted=29546332.782367, Expected=29551476.000000
day=8, Predicted=29821885.021968, Expected=29856310.000000
day=9, Predicted=30138004.836670, Expected=30170279.000000
day=10, Predicted=30450958.496441, Expected=30493785.000000
day=11, Predicted=30776662.153901, Expected=30773352.000000
day=12, Predicted=31040908.114464, Expected=31016379.000000
day=13, Predicted=31261524.533884, Expected=31314834.000000
day=14, Predicted=31599816.533124, Expected=31594602.000000
```

```
day=15, Predicted=31860561.328137, Expected=31861824.000000
day=16, Predicted=32127641.890768, Expected=32224421.000000
day=17, Predicted=32511717.405235, Expected=32552288.000000
day=18, Predicted=32833907.578642, Expected=32829197.000000
day=19, Predicted=33095928.557158, Expected=33070312.000000
day=20, Predicted=33313823.482064, Expected=33346336.000000
day=21, Predicted=33622703.381071, Expected=33631070.000000
day=22, Predicted=33902769.836034, Expected=33957884.000000
day=23, Predicted=34243454.690367, Expected=34275807.000000
day=24, Predicted=34556229.944303, Expected=34571172.000000
day=25, Predicted=34847017.496380, Expected=34889804.000000
day=26, Predicted=35173173.733261, Expected=35138269.000000
day=27, Predicted=35381754.457737, Expected=35467660.000000
day=28, Predicted=35758464.376364, Expected=35793012.000000
day=29, Predicted=36072754.247835, Expected=36142566.000000
day=30, Predicted=36427434.052185, Expected=36504174.000000
day=31, Predicted=36787232.802211, Expected=36863620.000000
day=32, Predicted=37147391.954519, Expected=37193872.000000
day=33, Predicted=37476975.636384, Expected=37463568.000000
day=34, Predicted=37725109.735946, Expected=37788829.000000
day=35, Predicted=38076532.249725, Expected=38117691.000000
day=36, Predicted=38399020.097940, Expected=38498286.000000
day=37, Predicted=38780391.968836, Expected=38905694.000000
day=38, Predicted=39183271.847825, Expected=39316324.000000
day=39, Predicted=39594458.555899, Expected=39657368.000000
day=40, Predicted=39943040.401489, Expected=39943263.000000
day=41, Predicted=40213242.862923, Expected=40391772.000000
day=42, Predicted=40662800.777516, Expected=40780443.000000
day=43, Predicted=41064184.866221, Expected=41224485.000000
day=44, Predicted=41493393.093155, Expected=41696807.000000
day=45, Predicted=41962057.314368, Expected=42192016.000000
day=46, Predicted=42451835.842702, Expected=42603760.000000
day=47, Predicted=42885590.998334, Expected=42957052.000000
day=48, Predicted=43241526.238501, Expected=43494699.000000
day=49, Predicted=43737988.175404, Expected=43964015.000000
day=50, Predicted=44234698.768765, Expected=44473376.000000
day=51, Predicted=44728030.740873, Expected=45024129.000000
day=52, Predicted=45270635.404842, Expected=45594086.000000
day=53, Predicted=45836763.939281, Expected=46070463.000000
day=54, Predicted=46339342.477580, Expected=46503704.000000
day=55, Predicted=46778857.950229, Expected=47012165.000000
day=56, Predicted=47266013.937487, Expected=47527270.000000
day=57, Predicted=47783867.374843, Expected=48124824.000000
day=58, Predicted=48358034.877348, Expected=48718539.000000
day=59, Predicted=48956813.365926, Expected=49360492.000000
day=60, Predicted=49585258.001943, Expected=49871598.000000
day=61, Predicted=50133022.570550, Expected=50436547.000000
day=62, Predicted=50677709.115456, Expected=50938348.000000
```

```
day=63, Predicted=51200631.130987, Expected=51494283.000000
day=64, Predicted=51737775.448513, Expected=52139269.000000
day=65, Predicted=52362374.599560, Expected=52786566.000000
day=66, Predicted=53012086.865878, Expected=53435351.000000
day=67, Predicted=53660196.650179, Expected=54029326.000000
day=68, Predicted=54268541.212451, Expected=54502332.000000
day=69, Predicted=54772489.987882, Expected=55030781.000000
day=70, Predicted=55280209.567629, Expected=55638883.000000
day=71, Predicted=55870645.714908, Expected=56262553.000000
day=72, Predicted=56493069.921436, Expected=56913120.000000
day=73, Predicted=57136845.075477, Expected=57579266.000000
day=74, Predicted=57799956.230710, Expected=58165570.000000
day=75, Predicted=58407222.159495, Expected=58649369.000000
day=76, Predicted=58916388.502539, Expected=59171092.000000
day=77, Predicted=59423124.920442, Expected=59759508.000000
day=78, Predicted=59996113.322944, Expected=60392453.000000
day=79, Predicted=60619827.642432, Expected=60973650.000000
day=80, Predicted=61216004.834960, Expected=61645535.000000
day=81, Predicted=61861991.393383, Expected=62244181.000000
Test RMSE: 223051.564
```



```
[81]: print('Mean absolute percentage error of LSTM is␣
       ↪',mean_absolute_percentage_error(raw_values[231:], predictions))
```

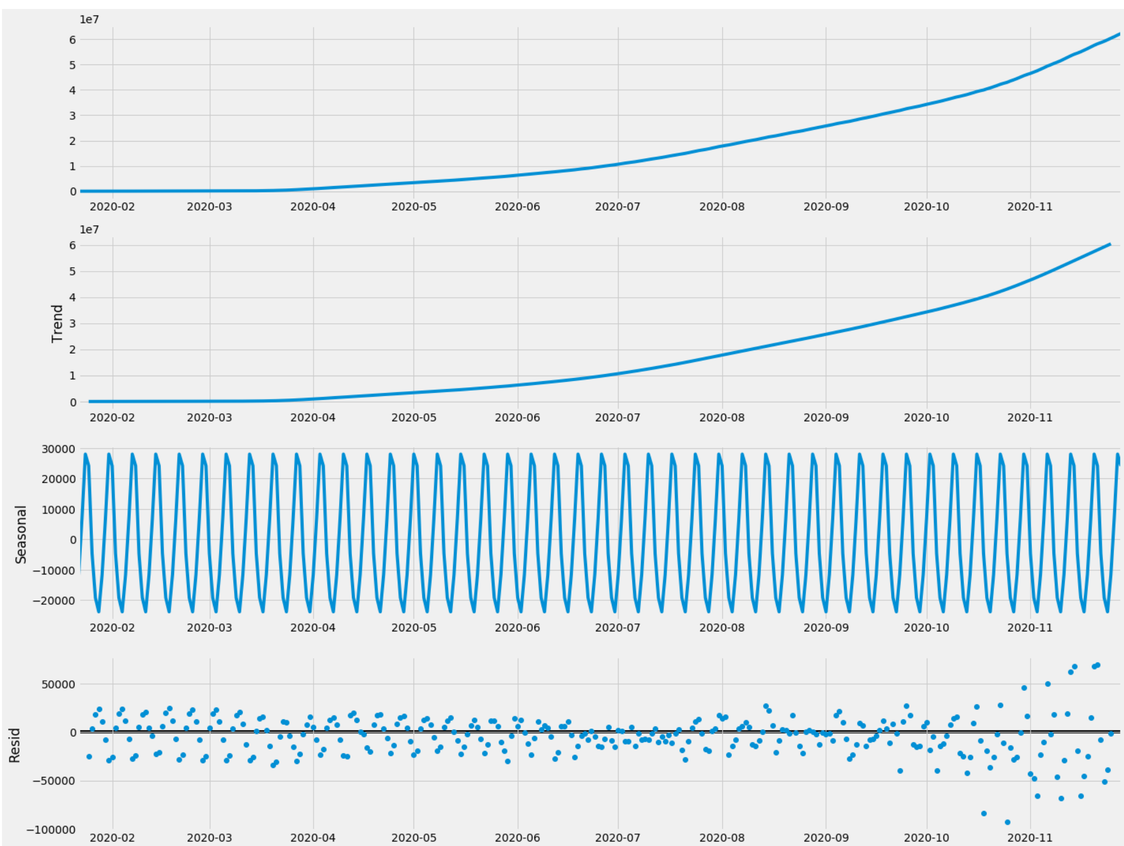
```
Mean absolute percentage error of LSTM is  0.35039339508351997
```

2.7.1 Comparing all the above model and their respective mean absolute percentage error, we can say that LSTM achieved greater accuracy.

Additive model

1. This model is used when the time series level does not vary with the variations around the trend. Here, the time series components are simply added together using the formula:
   - y(t) = Level(t) + Trend(t) + Seasonality(t) + Noise(t)

```
[93]: import matplotlib
      import statsmodels.api as sm
      decomposition = sm.tsa.
       ↪seasonal_decompose(time_series_analysis_df,model='additive')
      fir = decomposition.plot()
      matplotlib.rcParams['figure.figsize']=[20.0,15.0]
```



- Here we can see that trend is continously going up, **Total number of cases grew from 10 million in month of july to 40 million in the month of october.**
- The increase in the number of the cases can be attributed to some of the severly affected country mentioned above in the discussion.
- The sesonality shows us a sinusoidal trend which can be attributed to continous increasing trend in the number of confirmed cases.
- we can see some noise components in later months of **august, september, and october** which can be attributed to poorly affected countries above mentioned.

Time Series Forecasting with Arima (Autoregressive Integrated Moving Average) With the notation ARIMA(p, d, q), ARIMA models are denoted. The seasonality, pattern, and noise in the data account for these three parameters

[83]:
```
import itertools
```

[84]:
```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,␣
↪q))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

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

Parameter Selection

[85]:
```
# Use this for parameter selection

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

```
#               continue
```

The above output suggests that SARIMAX(1, 1, 1)x(1, 1, 1, 12) yields the lowest AIC value.

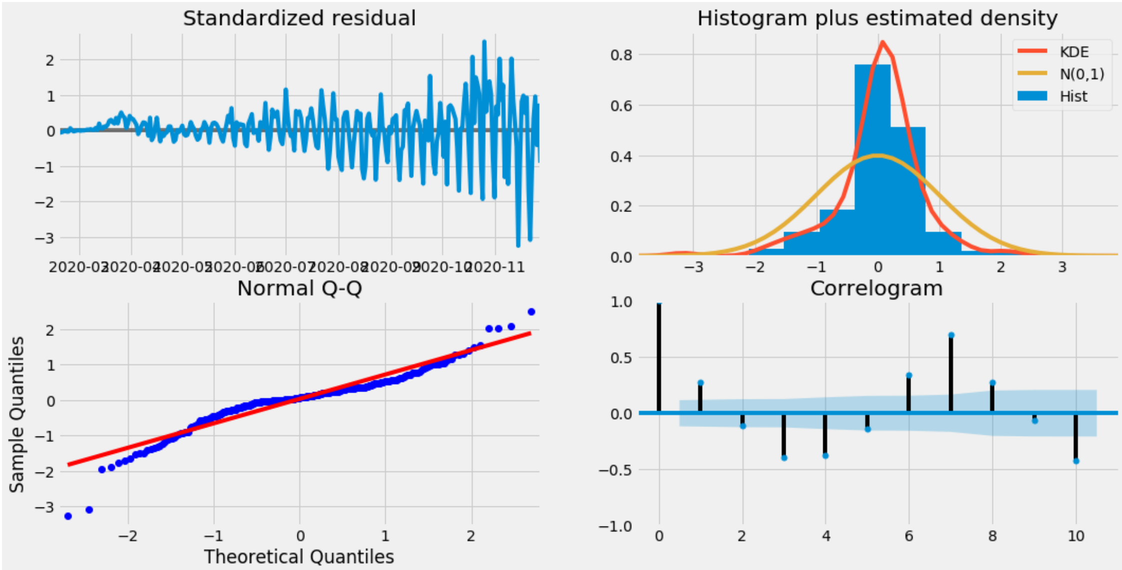
Fitting the model

[86]:
```
mod = sm.tsa.statespace.SARIMAX(time_series_analysis_df,
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning:

No frequency information was provided, so inferred frequency D will be used.

C:\ProgramData\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning:

No frequency information was provided, so inferred frequency D will be used.
```

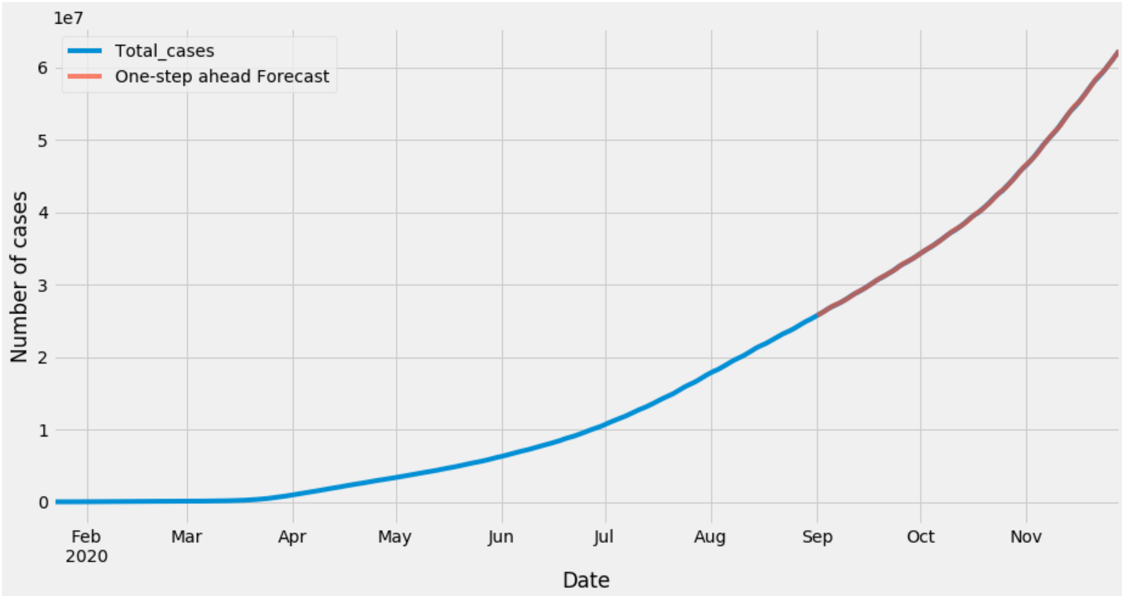
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0100	0.002	461.998	0.000	1.006	1.014
ma.L1	-0.7347	0.075	-9.838	0.000	-0.881	-0.588
ar.S.L12	-0.1863	0.113	-1.649	0.099	-0.408	0.035
ma.S.L12	-1.0704	0.047	-22.840	0.000	-1.162	-0.979
sigma2	2.077e+09	6.54e-12	3.18e+20	0.000	2.08e+09	2.08e+09

[87]:
```
results.plot_diagnostics(figsize=(16, 8))
plt.show()
```



Validation of forecasts

[88]:
```
pred = results.get_prediction(start=pd.to_datetime('2020-09-01'), dynamic=False)
pred_ci = pred.conf_int()
ax = time_series_analysis_df.plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7,␣
↪figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Number of cases')
plt.legend()
plt.show()
```




```
[89]: y_forecasted = pred.predicted_mean
      # print(y_forecasted)
      # time_series_analysis_df['Total_cases']['2020-09-01':]
      y_truth = time_series_analysis_df['Total_cases']['2020-09-01':]
      mse = ((y_forecasted - y_truth) ** 2).mean()
      print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 2518601386.41
```

```
[90]: print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.
       ↪sqrt(mse), 2)))
```

```
The Root Mean Squared Error of our forecasts is 50185.67
```

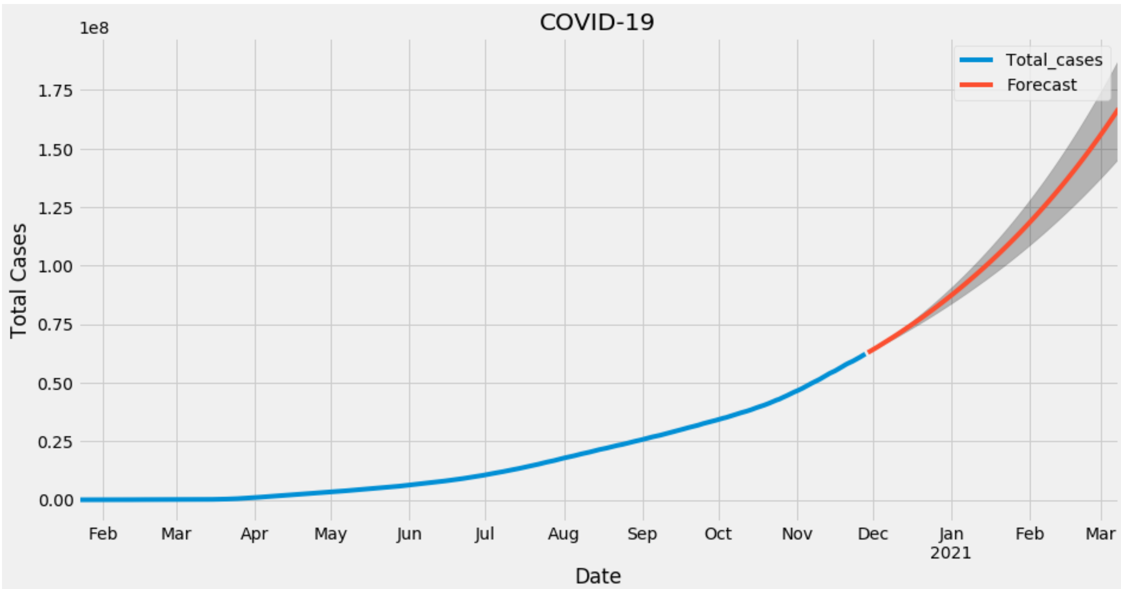
Mean Absolute percentage error

```
[91]: mean_absolute_percentage_error(y_truth,y_forecasted)
```

```
[91]: 0.09709434920506751
```

```
[92]: pred_uc = results.get_forecast(steps=100)
      pred_ci = pred_uc.conf_int()
      ax = time_series_analysis_df['Total_cases'].plot(label='Total_cases',␣
       ↪figsize=(14, 7))
      pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
      ax.fill_between(pred_ci.index,
                      pred_ci.iloc[:, 0],
                      pred_ci.iloc[:, 1], color='k', alpha=.25)
```

```python
ax.set_title('COVID-19')
ax.set_xlabel('Date')
ax.set_ylabel('Total Cases')
plt.legend()
plt.show()
```



- **Future Forecasting**
- We used Arima model from stats to predict future values as mean absolute percentage error of ARIMA model is very low ~0.098%
- Our model predicts that in the month of **Jan2021** we will have around [60million, 90million] cases.
- As we move further in the future the confidence interval of prediction drops because this model doesn't take into account various policies that have been implemented by countries to curb the spread the of the virus.