COVID-19 Data Analysis, Visualization & Prediction

Neelam Dighe | Sarvesh Sawant | Kunbo Chen

CONTENTS

1. Introduction

The Project aims at exploring pandemic's behavior through Data analytics and modelling. This will help to estimate health care demand allows planning resources to fight against COVID-19

2. Project Overview

3. Data Preprocessing – Data Source

- The COVID-19 Data Source is broad, specific and updated timely, like *Johns Hopkins University, World Health Organization and Centers for Disease Control and Prevention*
- In this project the dataset is provided by *Johns Hopkins University*
- Data range: 01/22/2020 - 08/22/2022

```
1 latest_data.head()
```

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined_Key	Incident_Rat
0	NaN	NaN	NaN	Afghanistan	2022-08-15 04:20:59	33.93911	67.709953	189045	7758	NaN	NaN	Afghanistan	485.62305
1	NaN	NaN	NaN	Albania	2022-08-15 04:20:59	41.15330	20.168300	321804	3571	NaN	NaN	Albania	11182.29202
2	NaN	NaN	NaN	Algeria	2022-08-15 04:20:59	28.03390	1.659600	269141	6878	NaN	NaN	Algeria	613.76191
3	NaN	NaN	NaN	Andorra	2022-08-15 04:20:59	42.50630	1.521800	45899	154	NaN	NaN	Andorra	59404.64634
4	NaN	NaN	NaN	Angola	2022-08-15 04:20:59	-11.20270	17.873900	102636	1917	NaN	NaN	Angola	312.28370

```
1 confirmed_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	8/12/22	8/13/22	8/14/22	8/15/22	8/16/22	8/
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	188704	188820	189045	189343	189477	18
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	320781	321345	321804	322125	322837	32
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	268866	269008	269141	269269	269381	26
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	45899	45899	45899	45899	45899	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	102636	102636	102636	102636	102636	10

5 rows × 947 columns

Data Sources Website
JHU: https://github.com/CSSEGISandData/COVID-19
WHO: https://www.who.int/emergencies/diseases/novel-coronavirus-2019
CDC: https://www.cdc.gov/coronavirus/2019-ncov/

3. Data Preprocessing – Import Packages

There are three kinds of packages imported:

- *Data Manipulation Packages*
- *Data Visualization Packages*
- *Modeling Packages*

```
import pandas as pd       # to store and process data in dataframe
import numpy as np        # for numerical analysis
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt      # basic visualization package
import matplotlib.colors as mcolors
import seaborn as sns      # advanced ploting
#import random
import math
import time
import datetime
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error,mean_absolute_error
from statsmodels.tsa.stattools import adfuller, acf, pacf,arma_order_select_ic
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import pmdarima as pm

import plotly.io as pio
import plotly.express as px
import plotly.graph_objects as go

#pio.write_html(fig, file="index.html", auto_open=True)

# Load specific forecasting tools
from statsmodels.tsa.arima.model import ARIMA,ARIMAResults

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining
from pmdarima import auto_arima # for determining ARIMA orders
```

3. Data Preprocessing – Data Overview

```
confirmed_df.head()
```

ovince/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	8/13/22	8/14/22	8/15/22	8/16/22
NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	188820	189045	189343	189477
NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	321345	321804	322125	322837
NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	269008	269141	269269	269381
NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	45899	45899	45899	45899
NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	102636	102636	102636	102636

× 948 columns

```
deaths_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	8/13/22	8/14/22	8/15/22	8
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	7758	7758	7759	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	3570	3571	3571	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	6878	6878	6878	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	154	154	154	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	1917	1917	1917	

5 rows × 948 columns

3. Data Preprocessing – Calculation

Some first most calculations for further use:

- world_cases
- total_deaths
- fatality_rate
- world_daily_increase
- world_confirmed_avg
- world_daily_increase_avg
- world_daily_death
- world_death_avg
- world_daily_death_avg

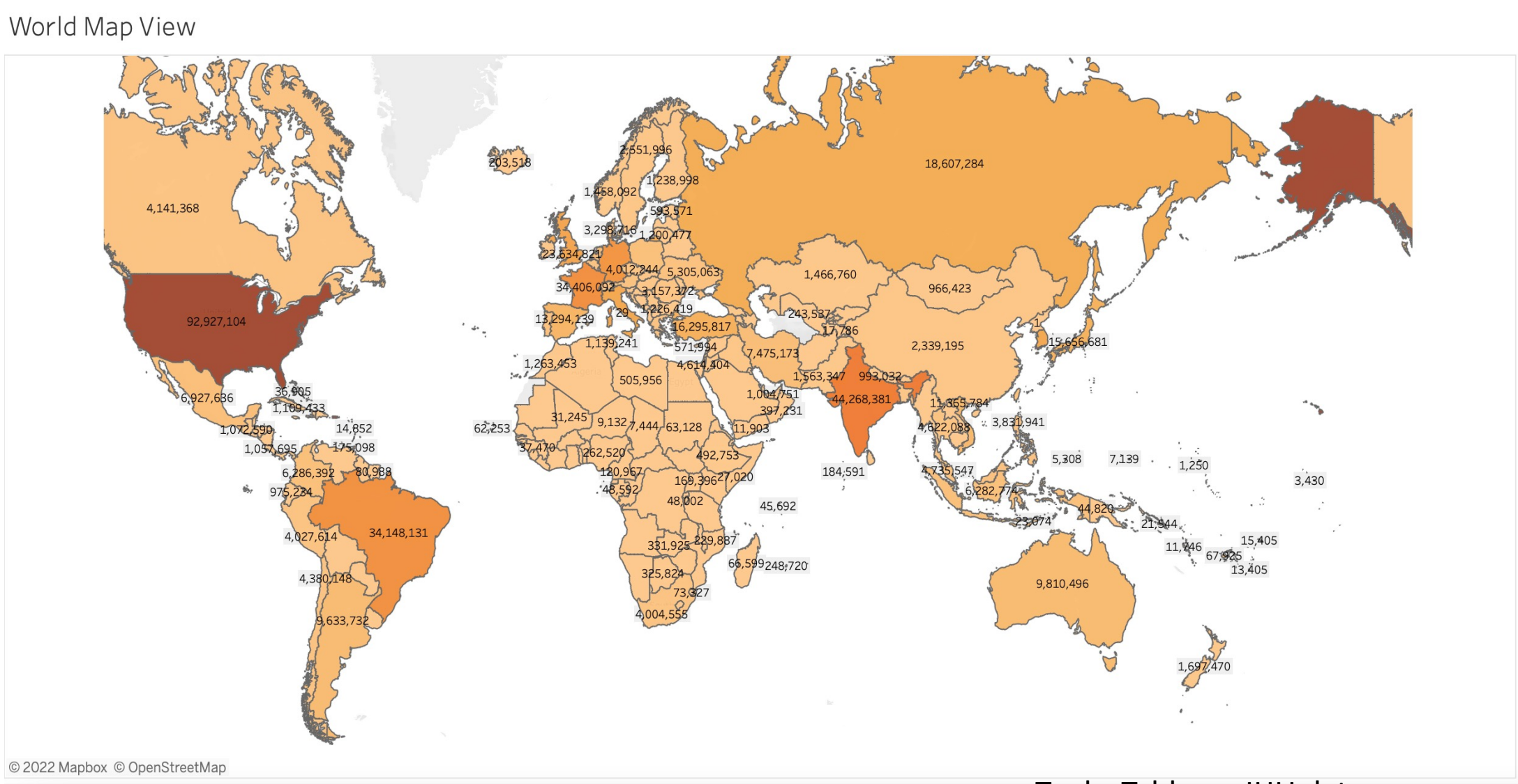
```
cols = confirmed_df.keys()
print(cols)
# get rid of the first 4 columns, get all the dates data onl
confirmed = confirmed_df.loc[:, cols[4]:cols[-1]]
deaths = deaths_df.loc[:, cols[4]:cols[-1]]

dates = confirmed.keys()
world_cases = []
total_deaths = []
fatality_rate = []

# for loop to sum and divide to get rates for use afterward
for i in dates:
    confirmed_sum = confirmed[i].sum()
    death_sum = deaths[i].sum()

    world_cases.append(confirmed_sum)
    total_deaths.append(death_sum)

    # calculate fatality rates
    fatality_rate.append(death_sum/confirmed_sum)
```

```
# for loop the data to get daily_increase and moving_average function fo
def daily_increase(data):
    d = []
    for i in range(len(data)):
        if i == 0:
            d.append(data[0])
        else:
            d.append(data[i]-data[i-1])
    return d

def moving_average(data, window_size):
    moving_average = []
    for i in range(len(data)):
        if i + window_size < len(data):
            moving_average.append(np.mean(data[i:i+window_size]))
        else:
            moving_average.append(np.mean(data[i:len(data)]))
    return moving_average

# set the window size
window = 7

# confirmed cases
world_daily_increase = daily_increase(world_cases)
world_confirmed_avg= moving_average(world_cases, window)
world_daily_increase_avg = moving_average(world_daily_increase, window)

# deaths
world_daily_death = daily_increase(total_deaths)
world_death_avg = moving_average(total_deaths, window)
world_daily_death_avg = moving_average(world_daily_death, window)
```

Next Steps

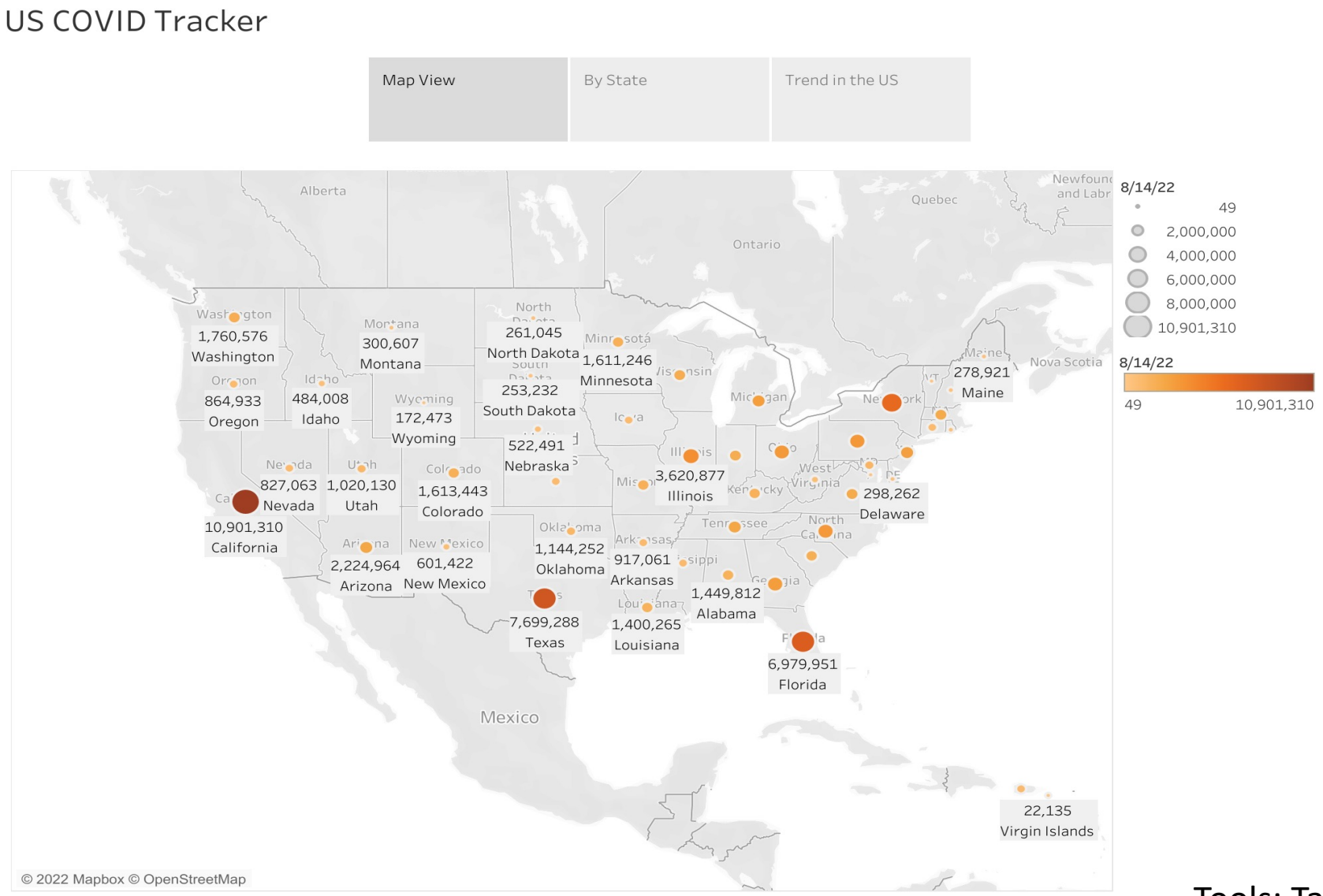
- Covid-19 Trend Visualization
- Exploratory data analysis before modeling

4. Visualization – COVID-19 Global Map



Tools: Tableau, JHU data source

4. Visualization – COVID-19 Trend US Map

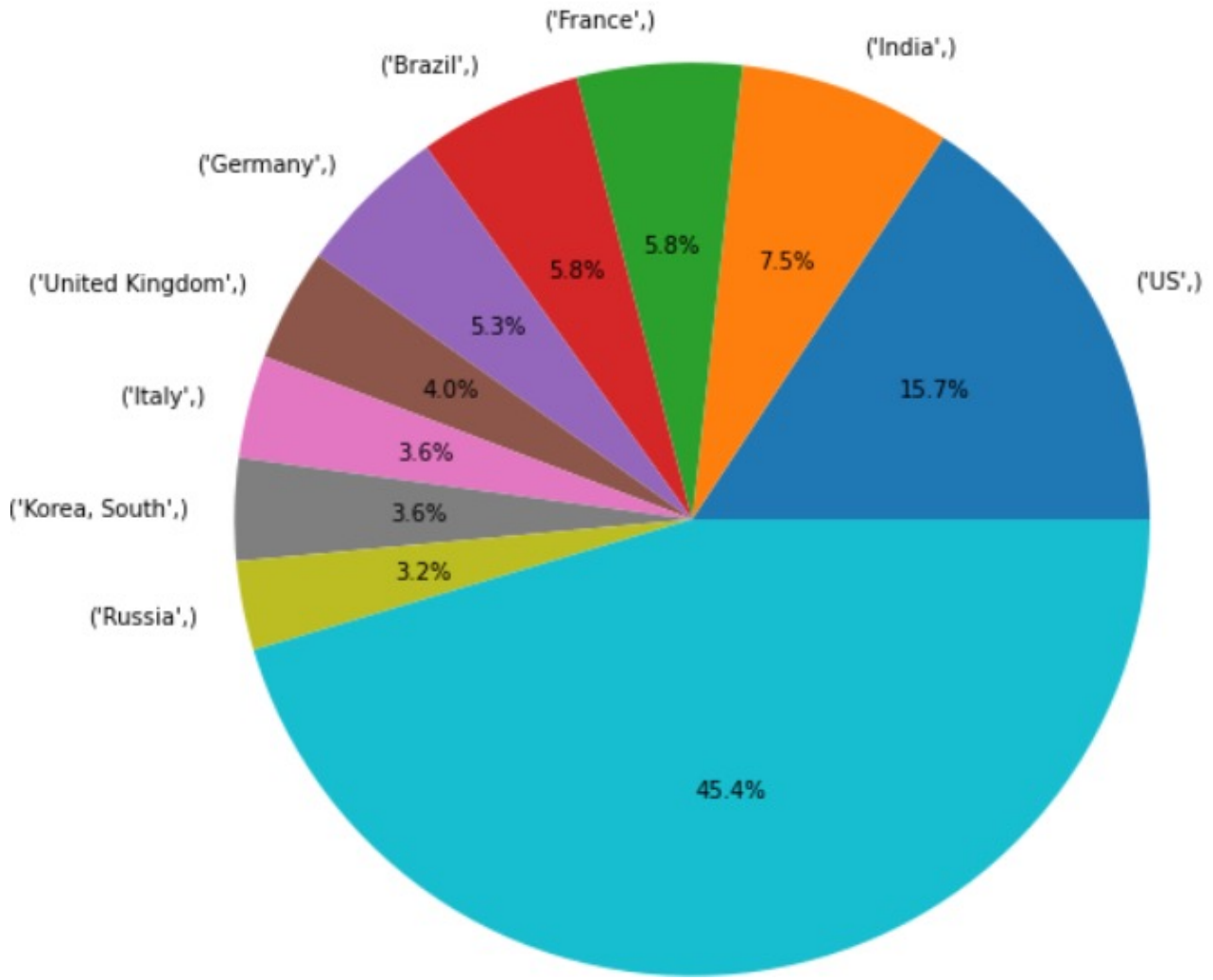


Tools: Tableau, JHU data source

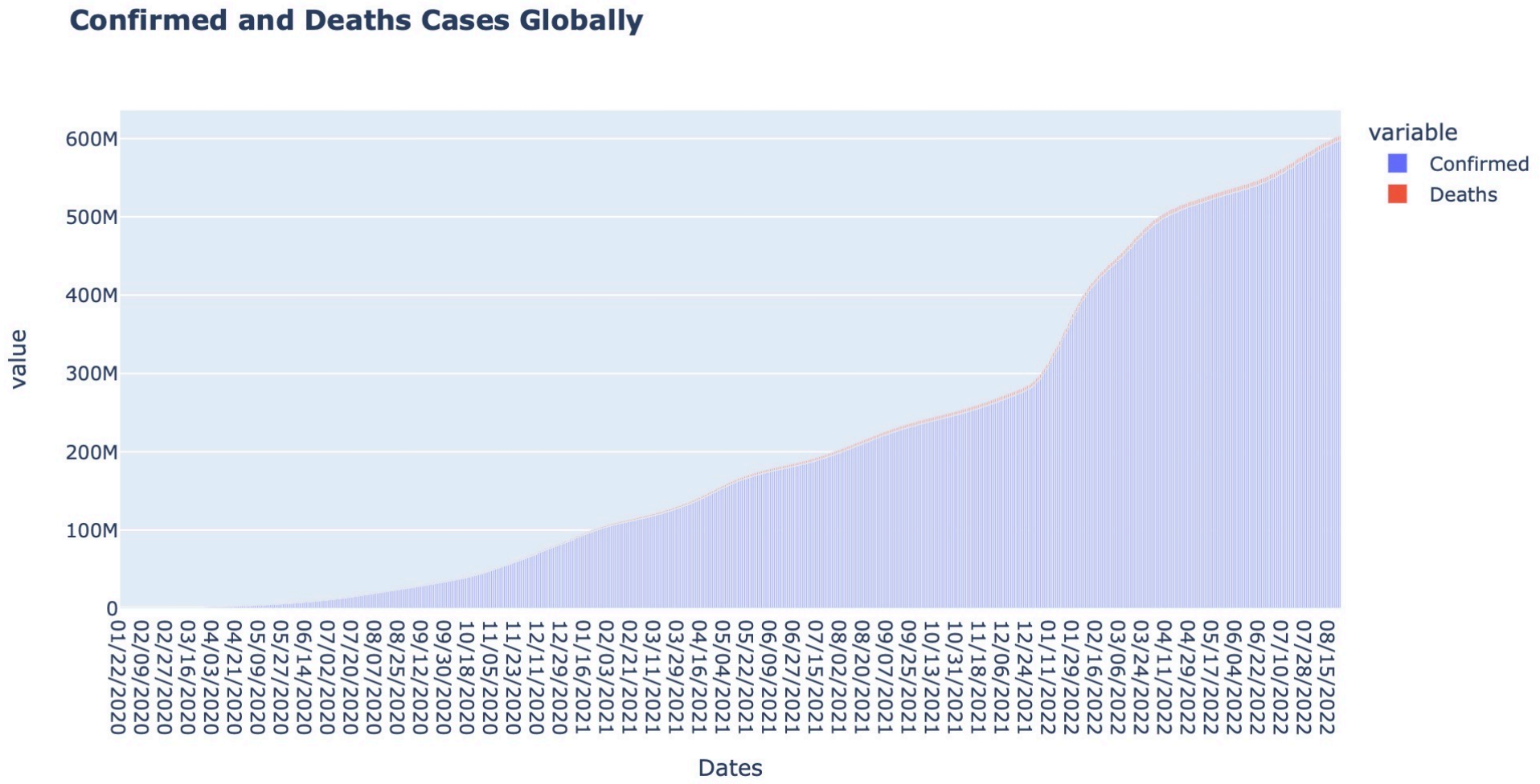
4. Visualization - Covid-19 Trend With Numbers

	Date	Country	Confirmed	Deaths	Recovered	Active	Recovered Cases Rate %	Deaths Cases Rate %	Total Cases Rate %
192	2022-08-15 04:20:59	US	92934013	1037118	0.000000	91896895.000000	0.000000	1.115972	15.742409
88	2022-08-15 04:20:59	India	44268381	527069	0.000000	43741312.000000	0.000000	1.190622	7.498772
71	2022-08-15 04:20:59	France	34406092	154104	0.000000	34251988.000000	0.000000	0.447897	5.828165
32	2022-08-15 04:20:59	Brazil	34148131	681253	0.000000	33466878.000000	0.000000	1.994994	5.784468
75	2022-08-15 04:20:59	Germany	31535340	145698	0.000000	31389642.000000	0.000000	0.462015	5.341879
196	2022-08-15 04:20:59	United Kingdom	23634821	204716	0.000000	23430105.000000	0.000000	0.866163	4.003583
94	2022-08-15 04:20:59	Italy	21499531	174060	0.000000	21325471.000000	0.000000	0.809599	3.641879
102	2022-08-15 04:20:59	Korea, South	21418036	25673	0.000000	21392363.000000	0.000000	0.119866	3.628074
154	2022-08-15 04:20:59	Russia	18607284	375360	0.000000	18231924.000000	0.000000	2.017275	3.151951
191	2022-08-15 04:20:59	Turkey	16295817	99678	0.000000	16196139.000000	0.000000	0.611678	2.760404
96	2022-08-15 04:20:59	Japan	15652210	35194	0.000000	15617016.000000	0.000000	0.224850	2.651381
174	2022-08-15 04:20:59	Spain	13294139	111667	0.000000	13182472.000000	0.000000	0.839972	2.251939
201	2022-08-15 04:20:59	Vietnam	11365784	43098	0.000000	11322686.000000	0.000000	0.379191	1.925289
17	2022-08-15 04:20:59	Australia	9810496	12886	0.000000	9797610.000000	0.000000	0.131349	1.661833

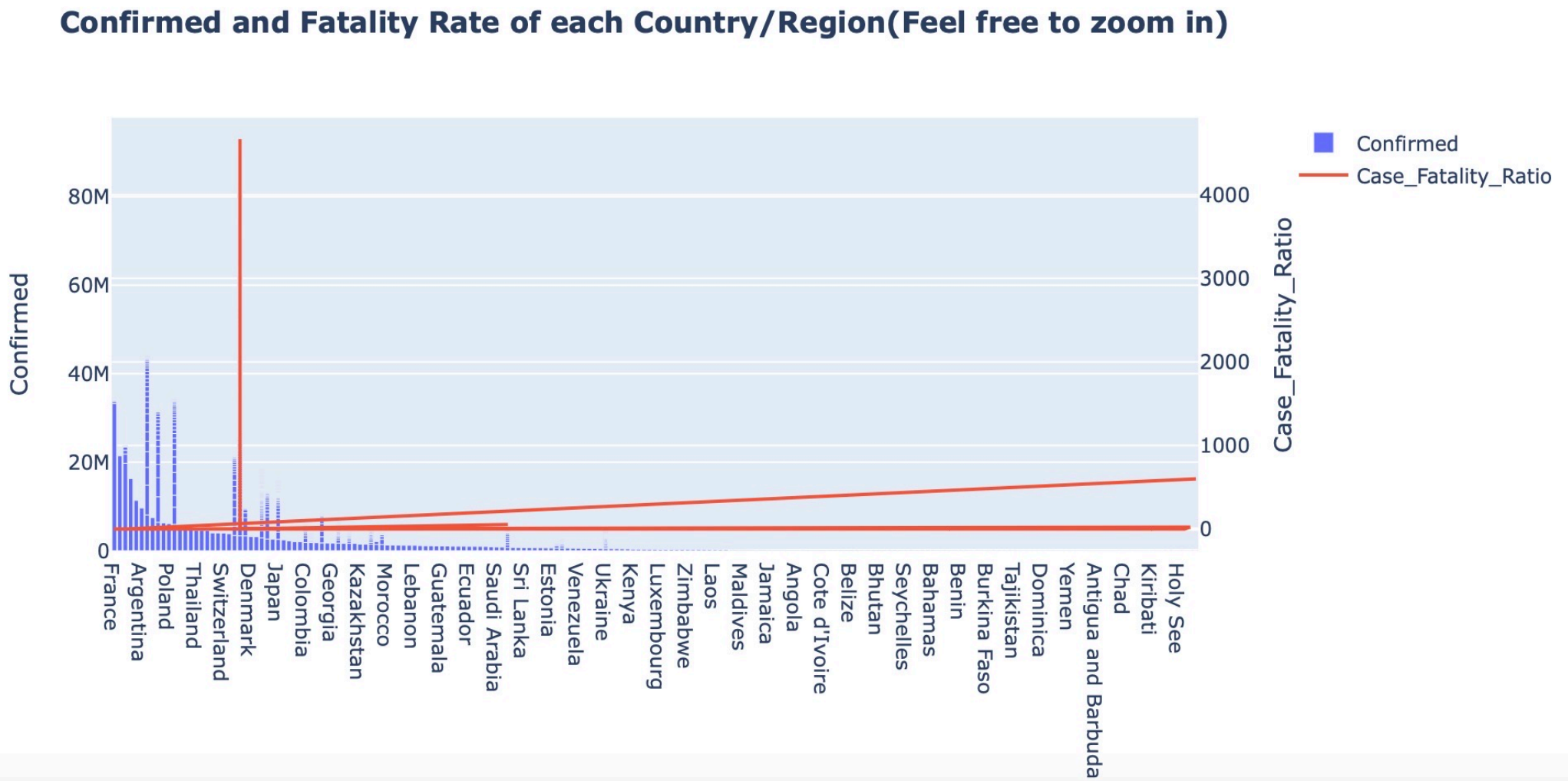
4. Visualization – Pie Chart



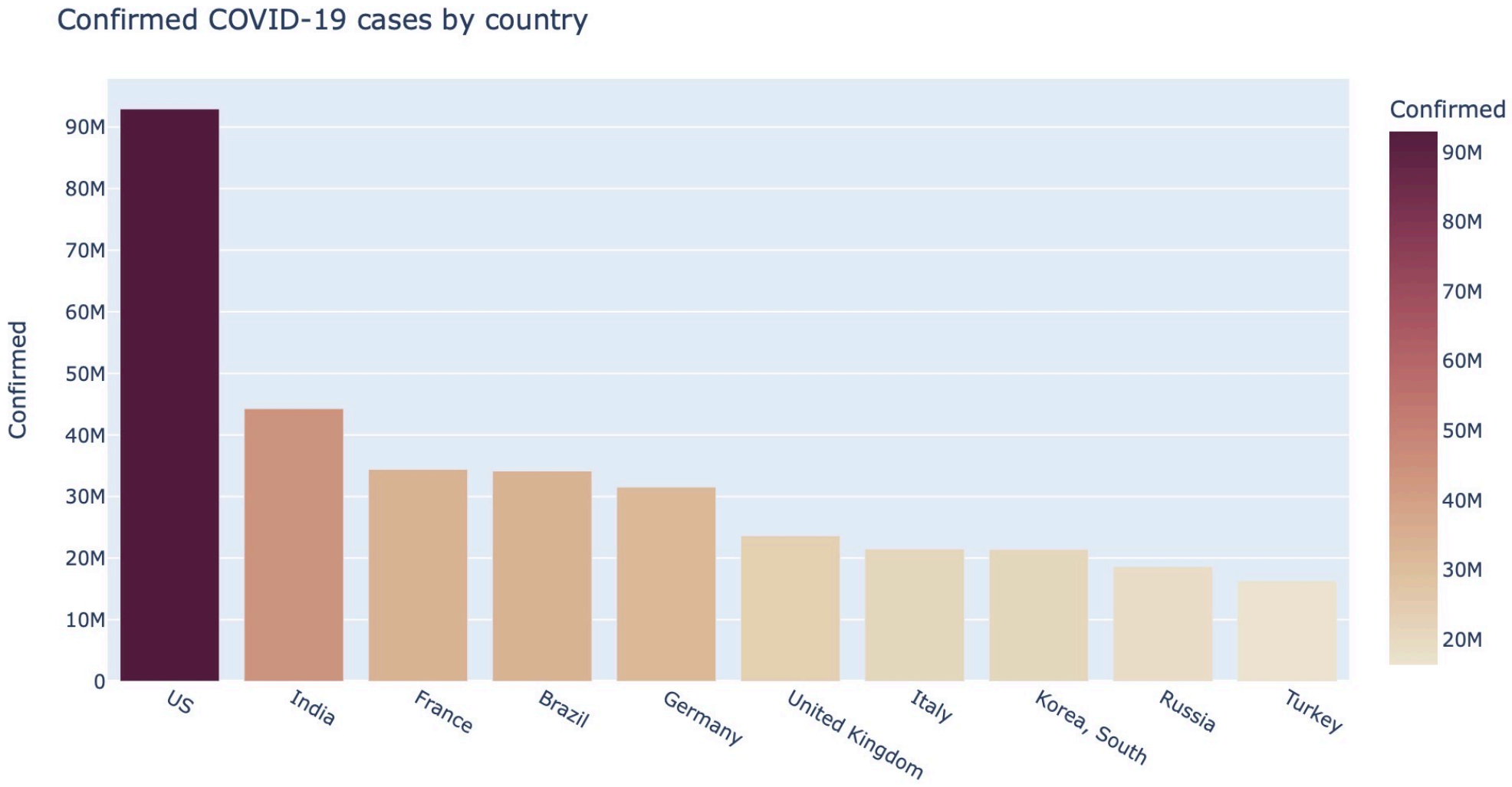
4. Exploratory data analysis before modeling



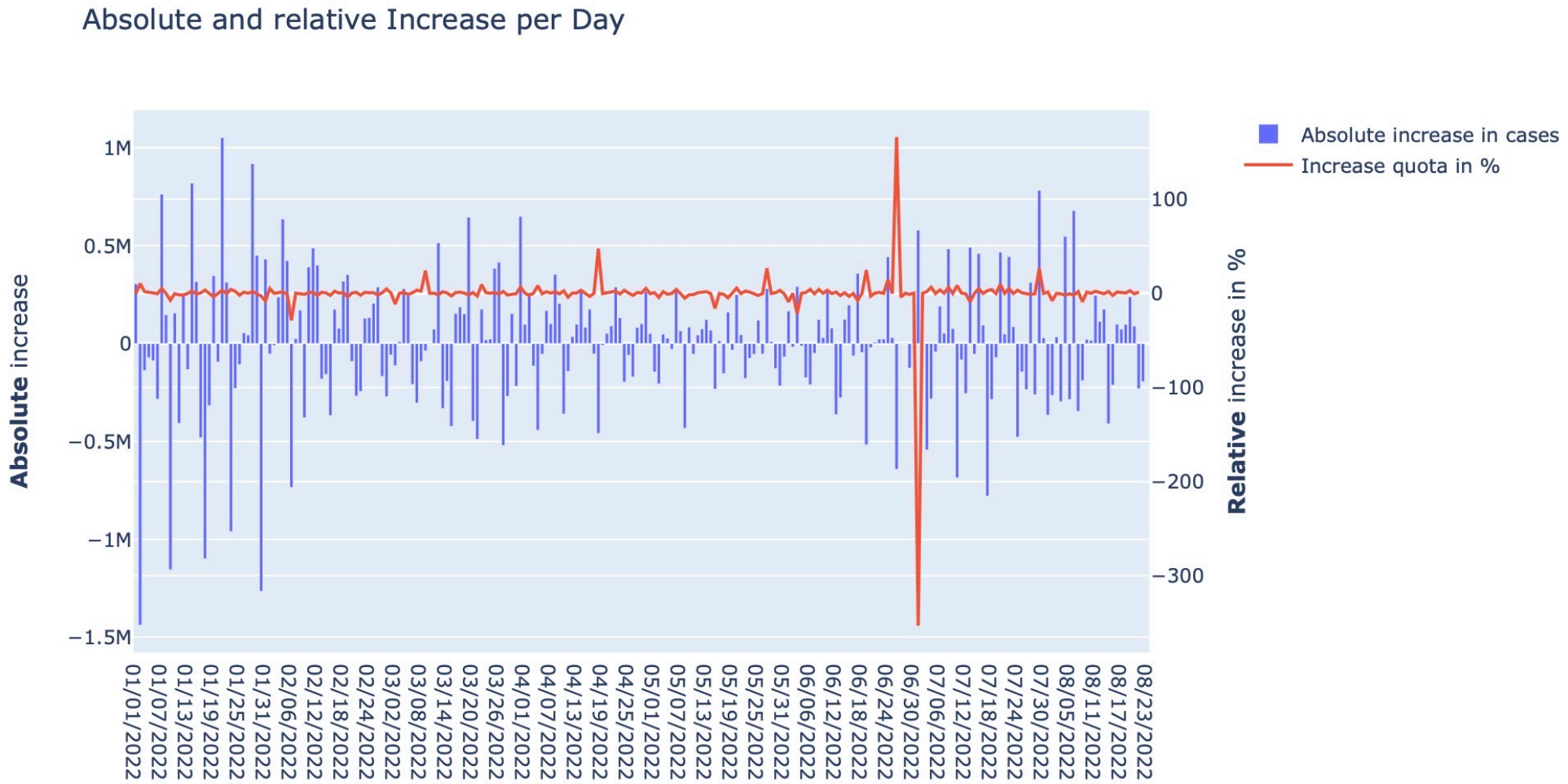
4. Exploratory data analysis before modeling



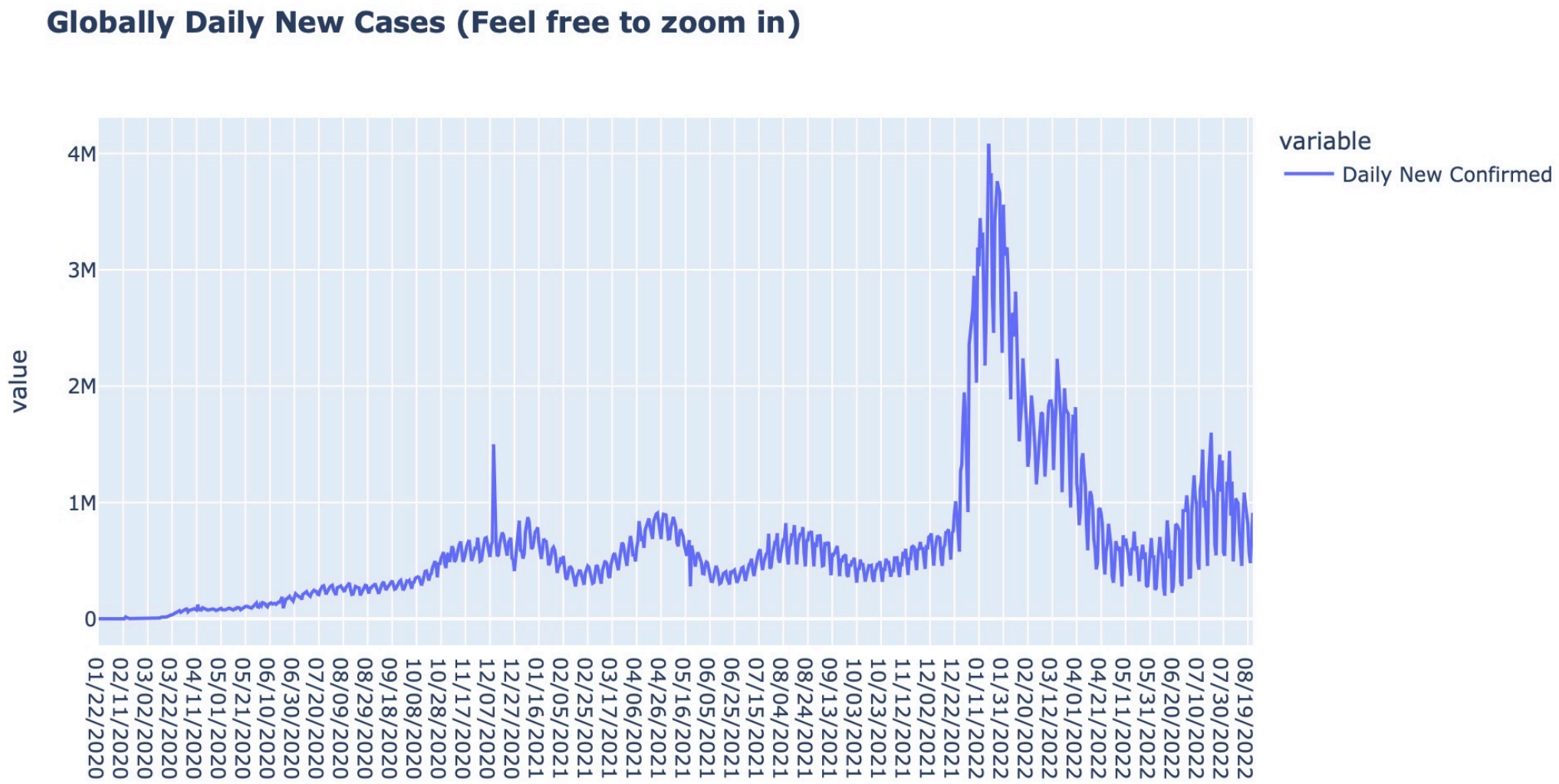
4. Exploratory data analysis before modeling



4. Exploratory data analysis before modeling



4. Exploratory data analysis before modeling



Next Steps

- ARIMA Model Building and prediction on a global level.
- Bayesian Ridge Regression Model Building and Prediction on a global level

5. **Modeling & Prediction**

Autoregressive Integrated Moving Average (ARIMA) : models the next step in the sequence as a linear function of the observations and residual errors at prior time steps.

$$X_t = c + \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \ldots + \alpha_p X_{t-p} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}$$

Why ARIMA:

- This model targets at using the existing historical time data to predict the future trend, and mainly aimed at stationary non-white noise sequence data.
- The data sequence is stationary, which means that the mean and variance should not change with time. The series can be made stationary by log transformation or differencing.
- The input data must be a univariate series because ARIMA uses past values to predict future values.
- Through the preliminary analysis of the data, this model excellently meets the application requirements

Basic Thought:

- The sequence obtained by arranging the data to be studied in chronological order is assumed to be a set of random sequences. The time sequence is a set of random variables that depend on time t. The development law of this sequence is simulated by a feasible model.
- When constructing the ARIMA forecasting model, the time series data is required to be stable, that is, the time series is required to be a stationary random sequence with zero mean.
- When applying the ARIMA model to forecast time series data, there are mainly three stages: identification of the stationarity of time series series, estimation and diagnosis of the fitted model, and prediction application of the model. Repeat these three steps, and finally get the optimized ARIMA model after several attempts to compare the parameters by manually modifying the parameters.

5. **Modeling & Prediction**

Introduction:

- Combining the autoregressive model (AR), the moving average model (MA), and the difference method, we will get the difference autoregressive moving average model ARIMA (p, d, q), where d is the order of the need to differentiate the data;
- Through using the modeling tools in the Python (sklearn), we could get directly which model fits best (AR, MA, or ARIMA) and which parameters can fit better

Steps to generate an ARIMA model:

- Perform ADF test to observe whether the sequence is stationary; for non-stationary time series, first d-order difference is performed to convert it into a stationary time series;
- After the first step of processing, a stationary time series has been obtained. To obtain the autocorrelation coefficient (ACF) and partial autocorrelation coefficient (PACF) of the stationary time series, the optimal order p and q can be obtained by analyzing the autocorrelation diagram and the partial autocorrelation diagram;
- From the d, q, and p obtained above, we will get the ARIMA model;
- Through using Python tools, the steps are much easy for us, and next we will do it

5. **Modeling & Prediction**

- Check stationarity
- Write a function that performs the augmented Dickey-Fuller Test.

```
adf_test(world_daily['Daily New Confirmed'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic       -2.749178
p-value                   0.065908
# lags used               20.000000
# observations           923.000000
critical value (1%)      -3.437455
critical value (5%)      -2.864676
critical value (10%)     -2.568440
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is Non-Stationary
```

```
def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles d

    labels = ['ADF test statistic','p-value','# lags used','# observations
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and" + " Stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is" + " Non-Stationary")
```

The result means that time series is non-stationary or have time-dependent structure. The ARIMA model predicts the future by finding the autocorrelation between historical data (assuming that the future will repeat the historical trend), and requires that the series must be stationary

5. **Modeling & Prediction**

- Difference n times and then check stationary again, the result shows that it could be stationary after management

```
from statsmodels.tsa.statespace.tools import diff
world_daily['Daily New Confirmed_diff1'] = diff(world_daily['Daily New Confirmed'], k_diff = 1)

adf_test(world_daily['Daily New Confirmed_diff1'] ,'Real Manufacturing and Trade Inventories')
```

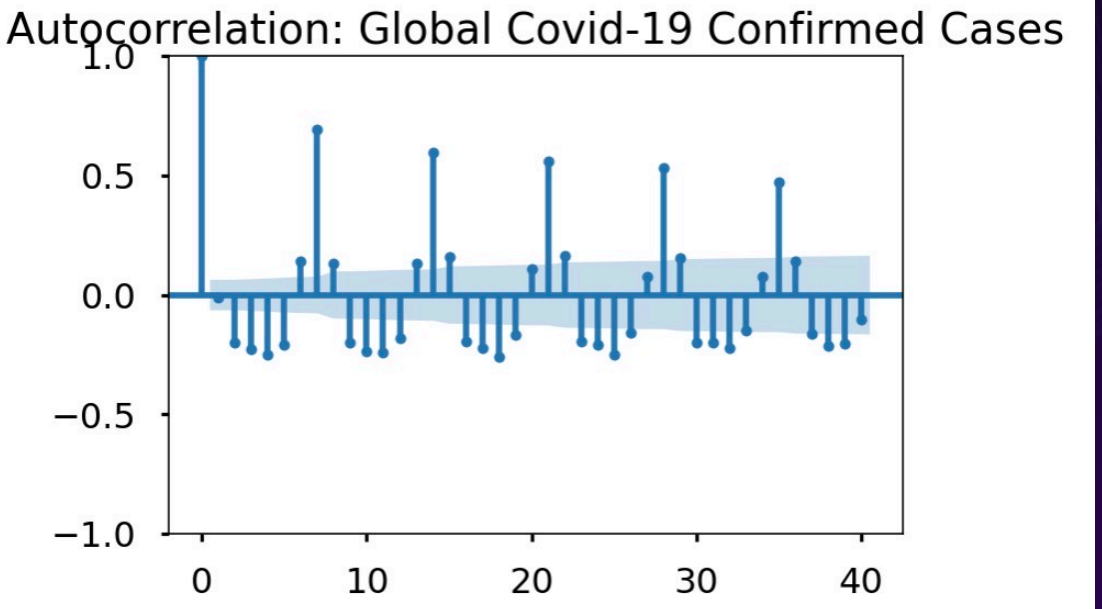
```
Augmented Dickey-Fuller Test: Real Manufacturing and Trade Inventories
ADF test statistic     -6.111000e+00
p-value                 9.333540e-08
# lags used             1.900000e+01
# observations          9.240000e+02
critical value (1%)    -3.437447e+00
critical value (5%)    -2.864673e+00
critical value (10%)   -2.568438e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and Stationary
```

5. **Modeling & Prediction**

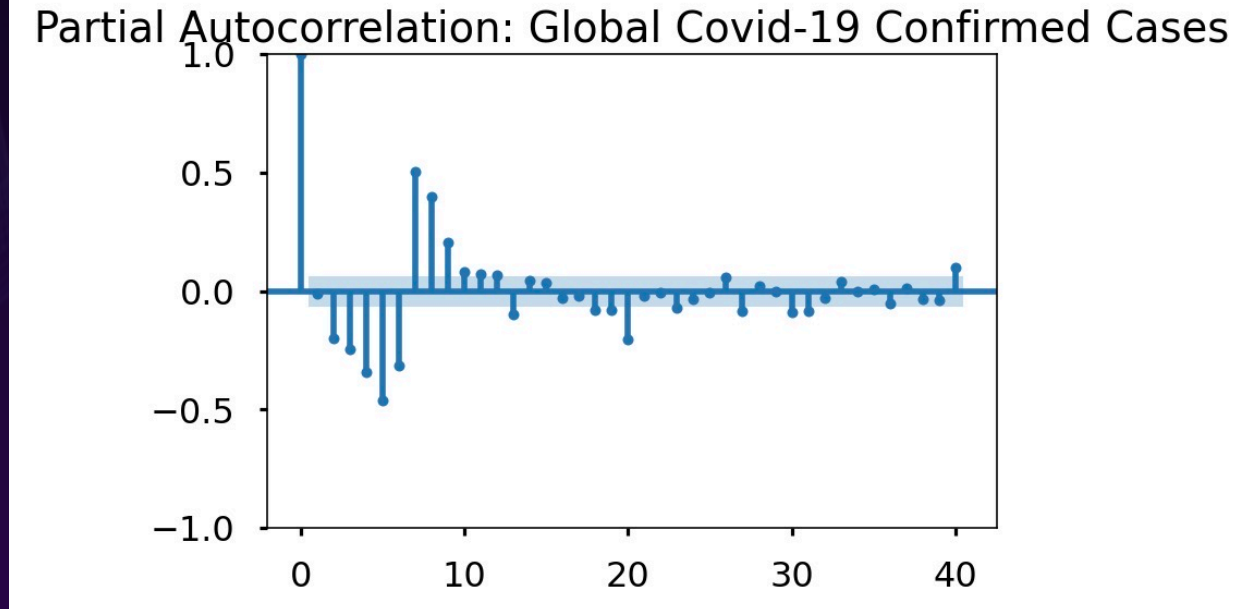
- ACF and PACF plots - View the ACF, PACF of the differenced sequence

From the plots of the sequences ACF and PACF, obvious tailing or truncation can be found, which again confirms that this dataset is suitable for fitting with the ARIMA model. Next, we use the library that comes with Python to select the ARIMA model for fitting.

```
title = 'Autocorrelation: Global Covid-19 Confirmed Cases'
lags = 40
plot_acf(world_daily_diff ,title=title,lags=lags);
```




```
title = 'Partial Autocorrelation: Global Covid-19 Confirmed Cases'
lags = 40
plot_pacf(world_daily_diff,title=title,lags=lags);
```



5. **Modeling & Prediction**

- Use pmdarima.auto_arima to determine ARIMA Orders
- Best model: ARIMA(3,1,2). This suggests that we should fit an ARIMA(3,1,2) model to best forecast future values of the series.

Ordering the model through tailing and truncation is highly subjective, the pmdarima library in the Python helps us with this

```
l = len(world_daily)
stepwise_fit = auto_arima(world_daily['Daily New Confirmed'].iloc[:l-50], start_p=0, start_q=0,
                          max_p=3, max_q=3, m=3,
                          seasonal=False,
                          d=None, trace=True,
                          error_action='ignore',   # we don't want to know if an order does not
                          suppress_warnings=True,  # we don't want convergence warnings
                          stepwise=True)           # set to stepwise

stepwise_fit.summary()
```

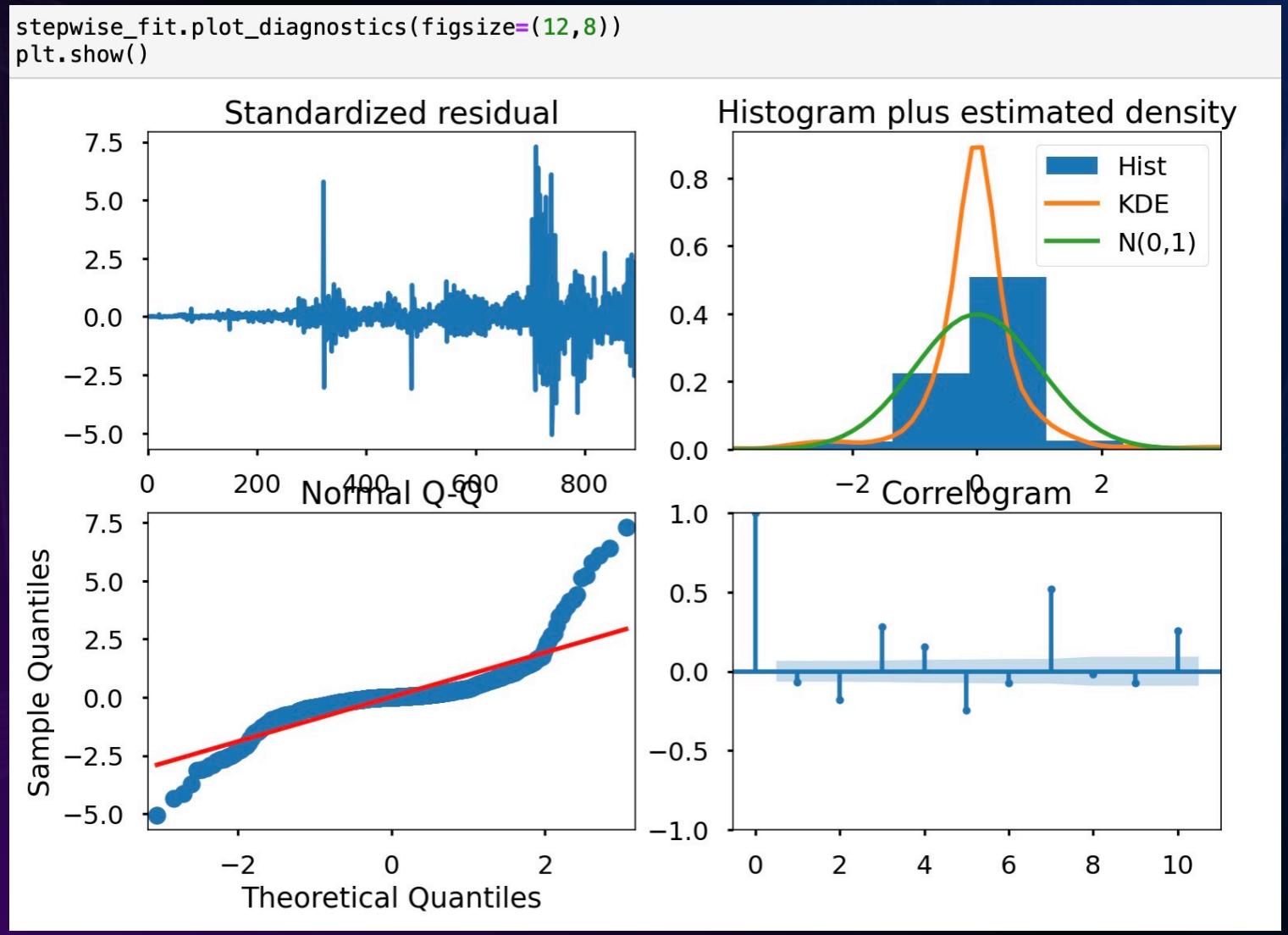
```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=24133.598, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=24135.598, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=24135.600, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=24131.628, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=24055.590, Time=0.10 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=23969.144, Time=0.11 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=24084.087, Time=0.04 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=23944.196, Time=0.16 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=24050.392, Time=0.05 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=23617.335, Time=0.52 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=23670.037, Time=0.46 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=23670.796, Time=0.57 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=24142.117, Time=0.36 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=23614.366, Time=0.51 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=23666.563, Time=0.37 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=23942.337, Time=0.14 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=23668.900, Time=0.50 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=23967.150, Time=0.10 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=24140.285, Time=0.34 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0]
Total fit time: 4.453 seconds
```

5. **Modeling & Prediction**

The distribution of residual errors is a Gaussian with a zero mean. It's the ideal. It is also a good idea to check the time series of the residual errors for any type of autocorrelation.
If present, it would suggest that the model has more opportunity to model the temporal structure in the data.

The diagnostics result is very good, it suggests that the data fits the model very well.

```
stepwise_fit.plot_diagnostics(figsize=(12,8))
plt.show()
```



5. **Modeling & Prediction**

```
n_forecast = len(test) + 10
pred = stepwise_fit.predict(n_forecast)

# Index nya Dates
dates = pd.date_range(train.index[-1], periods=n_forecast)
dates = dates.strftime('%d/%m/%Y')
pred = pd.DataFrame(pred, index=dates)

df = pd.concat([train, test, pred], axis=1)

df.columns = ['Data Train', 'Data Test', 'Prediction']
```

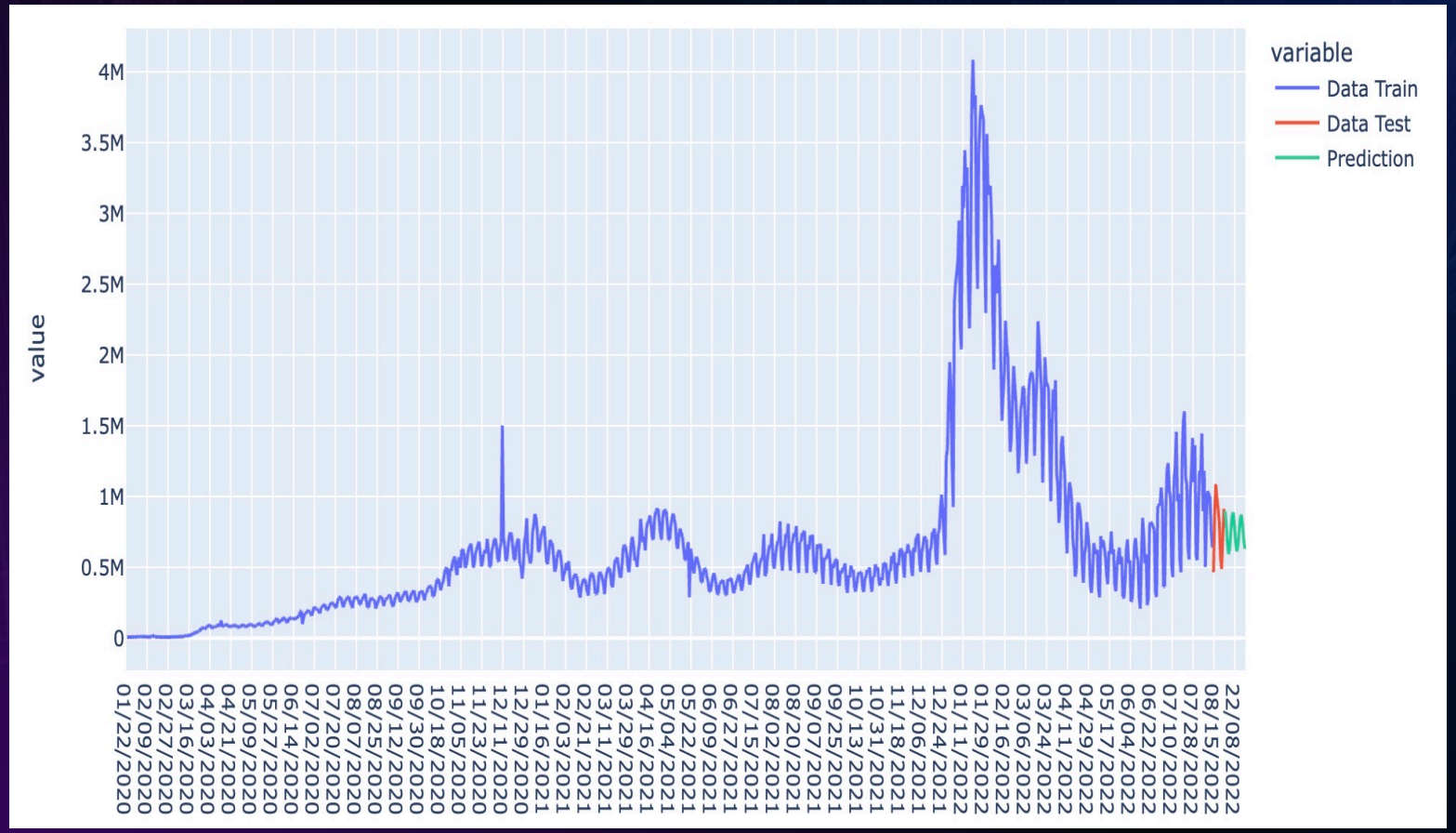
Predicted 10 days in the future on a global level. For it is quite difficult to predict further into the future since the status quo may change, i.e., a huge increase in vaccination numbers or the emergence of new variants.

```
n_forecast=len(arima_test)
arima_pred = stepwise_fit.predict(n_forecast)
from sklearn.metrics import r2_score
r2_score(arima_test,arima_pred)
```

```
0.45815091776820815
```

- Prediction

Next, we train the trend series, fit the training set data, and make predictions



5. **Modeling & Prediction**

- Prediction – ARIMA – Output New Confirmed cases

	Prediction of Daily New Confirmed Cases
13/08/2022	1117916.047635
14/08/2022	1189344.215510
15/08/2022	976269.233982
16/08/2022	670125.156869
17/08/2022	487839.505158
18/08/2022	558851.966826
19/08/2022	821918.968582
20/08/2022	1080051.530820
21/08/2022	1146807.029649
22/08/2022	980962.123489
23/08/2022	710900.891906
24/08/2022	535634.602838
25/08/2022	578367.772223
26/08/2022	800228.520138
27/08/2022	1034300.575982
28/08/2022	1110956.156199
29/08/2022	980557.268943
30/08/2022	744937.193937
31/08/2022	578249.696439
01/09/2022	598446.531833

5. **Modeling & Prediction**

Bayesian Ridge Regression : Bayesian Interpretation of Ridge Regression, Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity.

Basic Thought :

- Find maximum likelihood
- Find asymptotic solutions by EM, or "training" by "incremental learning"

Advantage:

- Bayesian regression has the strong ability to adapt to data, can reuse experimental data, and prevent overfitting.
- Bayesian regression can introduce a regular term into the estimation process.

5. **Modeling & Prediction**

Bayesian Ridge Regression

Assume that we are in the standard supervised learning setting, where we have a response vector $y \in \mathbb{R}^n$ and a design matrix $X \in \mathbb{R}^{n \times p}$. Ordinary least squares seeks the coefficient vector $\beta \in \mathbb{R}^p$ which minimizes the ***residual sum of squares* (RSS)**, i.e.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \ (y - X\beta)^T (y - X\beta).$$

Ridge regression is a commonly used regularization method which looks for β that minimizes the sum of the RSS and a penalty term:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \ (y - X\beta)^T (y - X\beta) + \lambda \|\beta\|_2^2,$$

where $\|\beta\|_2^2 = \beta_1^2 + \cdots + \beta_p^2$, and $\lambda \geq 0$ is a hyperparameter.

The ridge regression estimate has a Bayesian interpretation. Assume that the design matrix X is fixed. The ordinary least squares model posits that the conditional distribution of the response y is

$$y \mid X, \beta \sim \mathcal{N}(X\beta, \sigma^2 I),$$

where $\sigma > 0$ is some constant. In frequentism we think of β as being some fixed unknown vector that we want to estimate. In Bayesian statistics, we can impose a prior distribution on β and perform any estimation we want using the posterior distribution of β.

Let's say our prior distribution of β is that the β_j's are independent normals with the same variance, i.e. $\beta \sim \mathcal{N}(0, \tau^2 I)$ for some constant τ. This allows us to compute the posterior distribution of β:

$$\begin{aligned} p(\beta \mid y, X) &\propto p(\beta) \cdot p(y \mid X, \beta) \\ &\propto \exp\left[-\frac{1}{2}(\beta - 0)^T \frac{1}{\tau^2} I (\beta - 0)\right] \cdot \exp\left[-\frac{1}{2}(y - X\beta)^T \frac{1}{\sigma^2}(y - X\beta)\right] \\ &= \exp\left[-\frac{1}{2\sigma^2}(y - X\beta)^T (y - X\beta) - \frac{1}{2\tau^2}\|\beta\|_2^2\right]. \end{aligned}$$

From this expression, we can compute the mode of the posterior distribution, which is also known as the ***maximum a posteriori (MAP) estimate***. It is

$$\begin{aligned} \hat{\beta} &= \underset{\beta}{\operatorname{argmax}} \quad \exp\left[-\frac{1}{2\sigma^2}(y - X\beta)^T (y - X\beta) - \frac{1}{2\tau^2}\|\beta\|_2^2\right] \\ &= \underset{\beta}{\operatorname{argmin}} \quad \frac{1}{\sigma^2}(y - X\beta)^T (y - X\beta) + \frac{1}{\tau^2}\|\beta\|_2^2 \\ &= \underset{\beta}{\operatorname{argmin}} \quad (y - X\beta)^T (y - X\beta) + \frac{\sigma^2}{\tau^2}\|\beta\|_2^2, \end{aligned}$$

5. **Modeling & Prediction**

- Prediction – Bayesian Ridge Regression
  – Output Summed Confirmed Cases

```
# bayesian ridge regression
tol = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
alpha_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
alpha_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
normalize = [True, False]

bayesian_grid = {'tol': tol, 'alpha_1': alpha_1, 'alpha_2' : alpha_2,
                 'lambda_1': lambda_1, 'lambda_2' : lambda_2,
                 'normalize' : normalize}

bayesian = BayesianRidge(fit_intercept=False)
bayesian_search = RandomizedSearchCV(bayesian, bayesian_grid, scoring='neg_mean_squared_error',
                     cv=3, return_train_score=True, n_jobs=-1, n_iter=40, verbose=1)
bayesian_search.fit(bayesian_poly_X_train_confirmed, y_train_confirmed)
```

```
from sklearn.metrics import r2_score
r2_score(world_cases, bayesian_pred[:-10])
```

```
0.945031264625183
```

	Date	Bayesian Ridge Predicted # Confirmed Globally
0	08/24/2022	690942269.000000
1	08/25/2022	693137991.000000
2	08/26/2022	695338360.000000
3	08/27/2022	697543381.000000
4	08/28/2022	699753059.000000
5	08/29/2022	701967398.000000
6	08/30/2022	704186404.000000
7	08/31/2022	706410081.000000
8	09/01/2022	708638435.000000
9	09/02/2022	710871470.000000

Conclusion

- We used the COVID-19 dataset from Johns Hopkins University to do further analysis
- Data visualization of the overall dataset and specific US states
- Used confirmed cases data, fitted ARIMA & Bayesian Ridge Regression models to predict future covid cases on a global level, and got exact prediction value in the coming 10 days
- Finally, we got the prediction which r^2 score is 0.945, achieved a very accurate prediction result.

Thank you

Thank so much for Professor Gupta’s guidance and for your listening!