INF-552
MACHINE LEARNING FOR DATA SCIENCE

FINAL PROJECT
COVID -19 FORECASTING AND FACE MASK DETECTION

NAME	USC-ID	EMAIL
Amitabh Rajkumar Saini	7972003272	amitabhr@usc.edu
Shilpa Jain	4569044628	shilpaj@usc.edu
Sushumna Khandelwal	7458911214	sushumna@usc.edu

Table of Contents

COVID-19 FORECASTING

❖ What is COVID-19?

COVID-19 is a respiratory illness caused by a new virus. Symptoms include fever, coughing, sore throat and shortness of breath. The virus can spread from person to person, but good hygiene can prevent infection.

❖ Related Information about COVID-19

COVID-19 may not be fatal but it spreads faster than other diseases, like common cold. Every virus has Basic Reproduction number (R0) which implies how many people will get the disease from the infected person. As per initial research work R0 of COVID-19 is 2.7.

Currently the goal of all scientists around the world is to "Flatten the Curve". COVID-19 currently has exponential growth rate around the world which we will be seeing in the notebook ahead. Flattening the Curve typically implies even if the number of Confirmed Cases is increasing but the distribution of those cases should be over longer timestamp. To put it in simple words if say suppose COVID-19 is going infect 100K people then those many people should be infected in 1 year but not in a month.

The sole reason to Flatten the Curve is to reduce the load on the Medical Systems so as to increase the focus of Research to find the Medicine for the disease.

Every Pandemic has four stages:

- Stage 1: Confirmed Cases come from other countries
- Stage 2: Local Transmission Begins
- Stage 3: Communities impacted with local transmission
- Stage 4: Significant Transmission with no end in sight

Italy, USA, UK and France are the two countries which are currently in Stage 4 While India is in on the edge of Stage 3.

Other ways to tackle the disease like Corona other than Travel Ban, Cross-Border shutdown, ban on immigrants are Testing, Contact Tracing and Quarantine.

❖ Objective of the Notebook

Objective of this notebook is to study COVID-19 outbreak with the help of some basic visualization techniques. Perform predictions and Time Series forecasting in order to study the impact and spread of the COVID-19 in coming days.

The whole notebook is divided into 3 sections:

A. Exploratory Data Analysis
B. Simulation Modelling using SEIRD Model
C. Prediction and Time Series Forecasting

❖ Data Sources

- World Health Organization (WHO): https://www.who.int/
- US CDC: https://www.cdc.gov/coronavirus/2019-ncov/index.html
- 1Point3Arces: https://coronavirus.1point3acres.com/en
- WorldoMeters: https://www.worldometers.info/coronavirus/
- COVID Tracking Project: https://covidtracking.com/data. (US Testing and Hospitalization Data. We use the maximum reported value from "Currently" and "Cumulative" Hospitalized for our hospitalization number reported for each state.)
- Washington State Department of Health: https://www.doh.wa.gov/emergencies/coronavirus
- Maryland Department of Health: https://coronavirus.maryland.gov/
- New York State Department of Health: https://health.data.ny.gov/Health/New-York-State-Statewide-COVID-19-Testing/xdss-u53e/data
- NYC Department of Health and Mental Hygiene: https://www1.nyc.gov/site/doh/covid/covid-19-data.page and https://github.com/nychealth/coronavirus-data
- Florida Department of Health Dashboard: https://services1.arcgis.com/CY1LXxl9zlJeBuRZ/arcgis/rest/services/Florida_COVID19_Cases/FeatureServer/0 and https://fdoh.maps.arcgis.com/apps/opsdashboard/index.html#/8d0de33f260d444c852a615dc7837c86

❖ Libraries Used

- Pandas
- Numpy
- Matplotlib.pyplot
- Seaborn
- Datetime
- Itertools
- Sklearn.preprocessing
- Sklearn.cluster
- Sklearn.metrics
- Sklearn.linear_model
- Sklearn.model_selection
- Sklearn.svm
- Statsmodels.api
- Statsmodels.tsa.api
- Statsmodels.tsa.stattools
- Statsmodels.tsa.ar_model
- Statsmodels.tsa.arima_model
- Statsmodels.graphics.tsaplot
- Scipy.integrate
- Lmfit
- Warnings

❖ Dataframe Description

covid_data: dataframe containing worldwide with columns as ObservationDate, Province/State, Country/Region, Last Update, Confirmed, Deaths, Recovered.

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

Out[3]:

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	1	01/22/2020	Anhui	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
1	2	01/22/2020	Beijing	Mainland China	1/22/2020 17:00	14.0	0.0	0.0
2	3	01/22/2020	Chongqing	Mainland China	1/22/2020 17:00	6.0	0.0	0.0
3	4	01/22/2020	Fujian	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
4	5	01/22/2020	Gansu	Mainland China	1/22/2020 17:00	0.0	0.0	0.0

covid_data_us: dataframe containing only United States data with columns as ObservationDate, Province/State, Country/Region, Last Update, Confirmed, Deaths, Recovered.

```
covid_data_us.head()
```

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
31	32	2020-01-22	Washington	US	2020-01-22 17:00:00	1.0	0.0	0.0
69	70	2020-01-23	Washington	US	2020-01-23 17:00:00	1.0	0.0	0.0
117	118	2020-01-24	Washington	US	2020-01-24 17:00:00	1.0	0.0	0.0
118	119	2020-01-24	Chicago	US	2020-01-24 17:00:00	1.0	0.0	0.0
158	159	2020-01-25	Washington	US	2020-01-25 17:00:00	1.0	0.0	0.0

datewise: dataframe containing count of Confirmed Cases, Death Cases, Recovered Cases of United States with columns as Confirmed, Deaths, ObservationDate, Recovered.

```
datewise.head()
```

	Confirmed	Deaths	ObservationDate	Recovered
0	1.0	0.0	2020-01-22	0.0
1	1.0	0.0	2020-01-23	0.0
2	2.0	0.0	2020-01-24	0.0
3	2.0	0.0	2020-01-25	0.0
4	5.0	0.0	2020-01-26	0.0

statewise: dataframe containing count of Confirmed Cases, Death Cases of United States group by each states. Here Province/State is the index with columns as Confirmed, Deaths, Mortality

```
statewise.head()
```

Province/State	Confirmed	Deaths	Mortality
New York	295106.0	22912.0	7.763990
New Jersey	113856.0	6442.0	5.658024
Massachusetts	58302.0	3153.0	5.408048
Illinois	48102.0	2125.0	4.417696
California	46164.0	1864.0	4.037778

beds: ICU beds count of each country in world with columns as Country, continent, ICU_Beds

```
beds.head()
```

	Country	Continent	ICU_Beds
0	Japan	Asia	7.3
1	South Korea	Asia	10.6
2	Russia	Europe	8.3
3	Germany	Europe	29.2
4	Austria	Europe	21.8

probabilities: dataframe that stores probabilities agewise with columns as Age, prob_1_to_ICU_1, prob_1_to_ICU_2, prob_1_to_ Death _1, prob_1_to_Death_2

	Age	prob_1_to_ICU_1	prob_ICU_to_Death_1	prob_1_to_ICU_2	prob_ICU_to_Death_2
0	0_9	0.001	0.00	0.001	0.00
1	10_19	0.003	0.00	0.003	0.00
2	20_29	0.009	0.05	0.006	0.00
3	30_39	0.025	0.06	0.009	0.00
4	40_49	0.030	0.10	0.025	0.12

❖ [A] Exploratory Data Analysis

1. Data Understanding

This step involves understanding the type of data, columns of data and finding the missing entries in dataset. We are preferably dealing with covid_data Dataframe.

```
Size/Shape of the dataset: (3736, 8)
Checking for null values:
SNo                0
ObservationDate    0
Province/State     0
Country/Region     0
Last Update        0
Confirmed          0
Deaths             0
Recovered          0
dtype: int64
Checking Data-type of each column:
SNo                         int64
ObservationDate    datetime64[ns]
Province/State             object
Country/Region             object
Last Update        datetime64[ns]
Confirmed                 float64
Deaths                    float64
Recovered                 float64
dtype: object
```

2. Data Cleaning

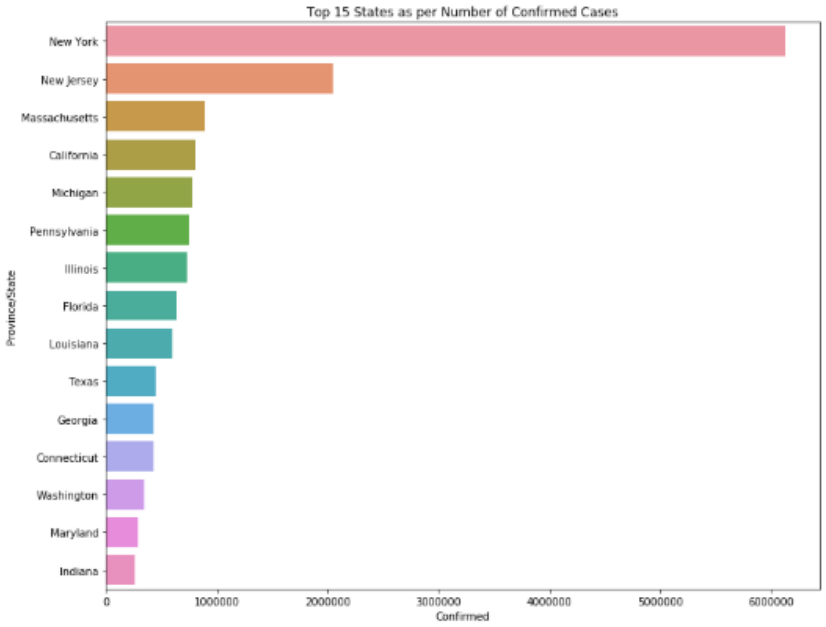
Data cleaning consists of four steps: missing data imputation, outlier detection, noise removal, and time alignment and delay estimation.

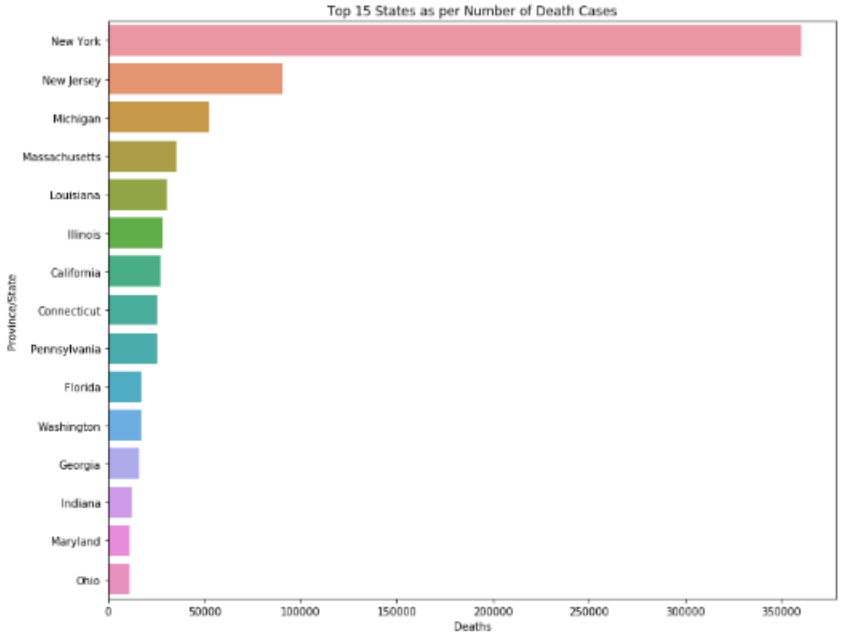
In doing so we observed that in covid_data_us dataframe the value of recovered is 0 and there is a separate entry datewise in Province/State column with value as Recovered

3. Data Analysis

We derived various key insights using statistics and visualization while performing exploratory data analysis to get detailed understanding of the data.

- **State-wise Confirmed and Death Count of top 15 States:**



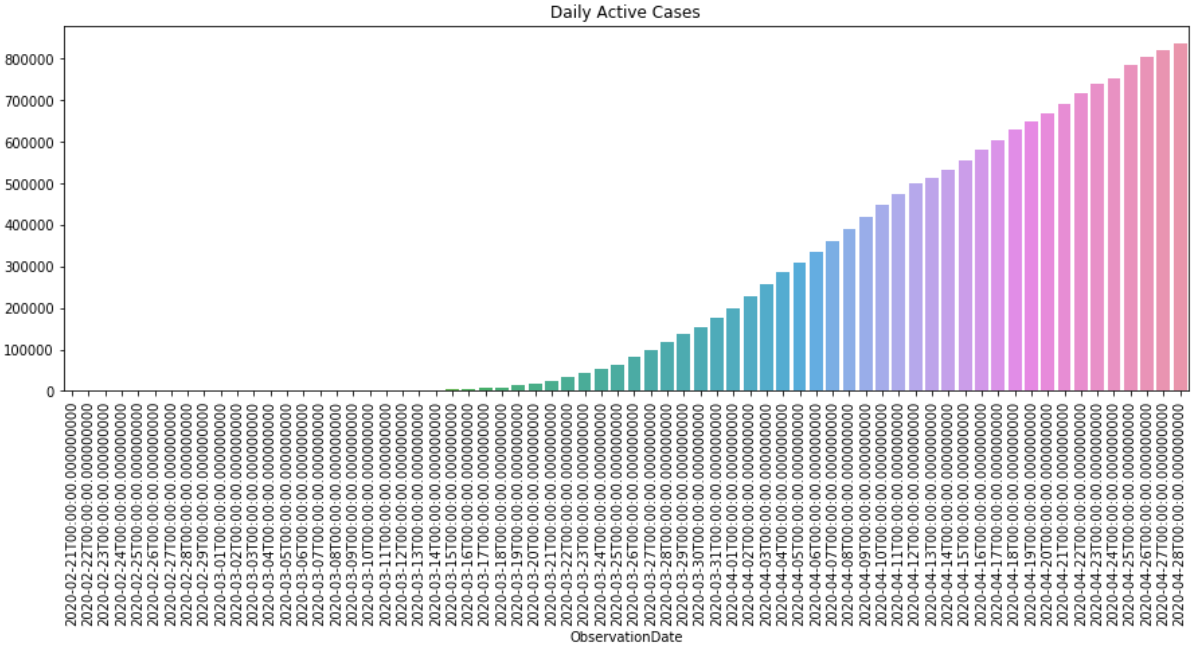


- **Total Number of Confirmed, Recovered, Death, Active cases per day per hour and overall:**

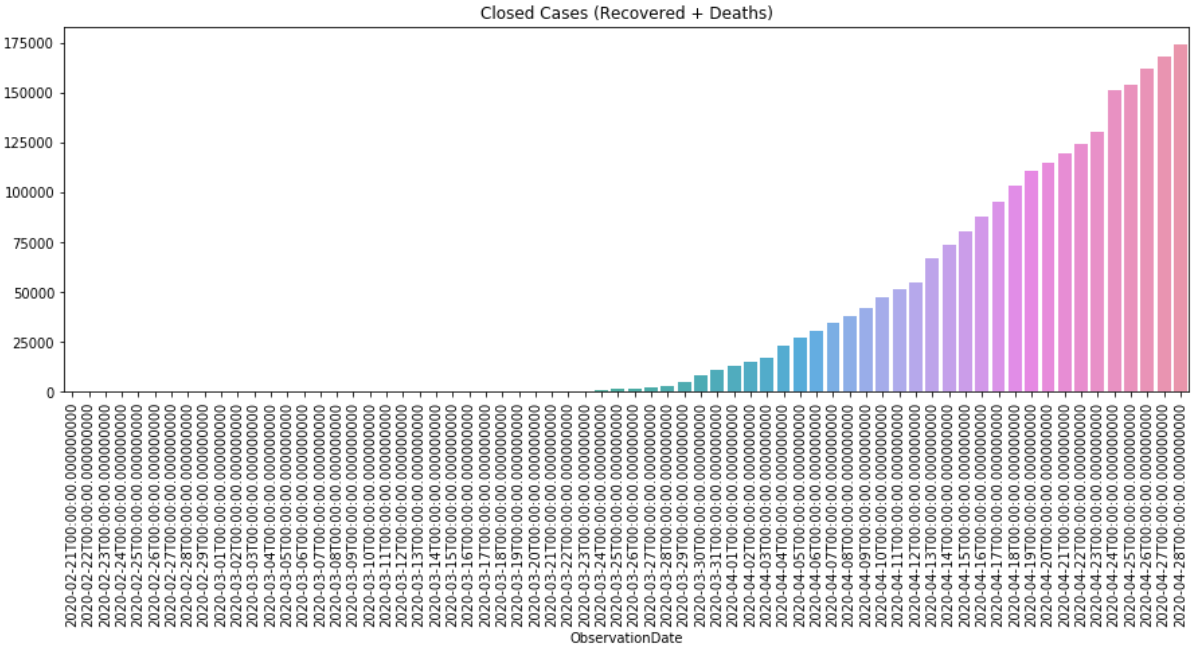
```
Total number of States with Disease Spread:  198
Total number of Confirmed Cases:  1012582.0
Total number of Recovered Cases:  115936.0
Total number of Deaths Cases:  58355.0
Total number of Active Cases:  838291.0
Total number of Closed Cases:  174291.0
Approximate number of Confirmed Cases per Day:  10332.0
```

```
Approximate number of Recovered Cases per Day:  1183.0
Approximate number of Death Cases per Day:  595.0
Approximate number of Confirmed Cases per hour around:  431.0
Approximate number of Recovered Cases per hour around:  49.0
Approximate number of Death Cases per hour around the World:  2
5.0
Number of Confirmed Cases in last 24 hours:  24385.0
Number of Confirmed Cases in last 24 hours:  4512.0
Number of Confirmed Cases in last 24 hours:  2096.0
```

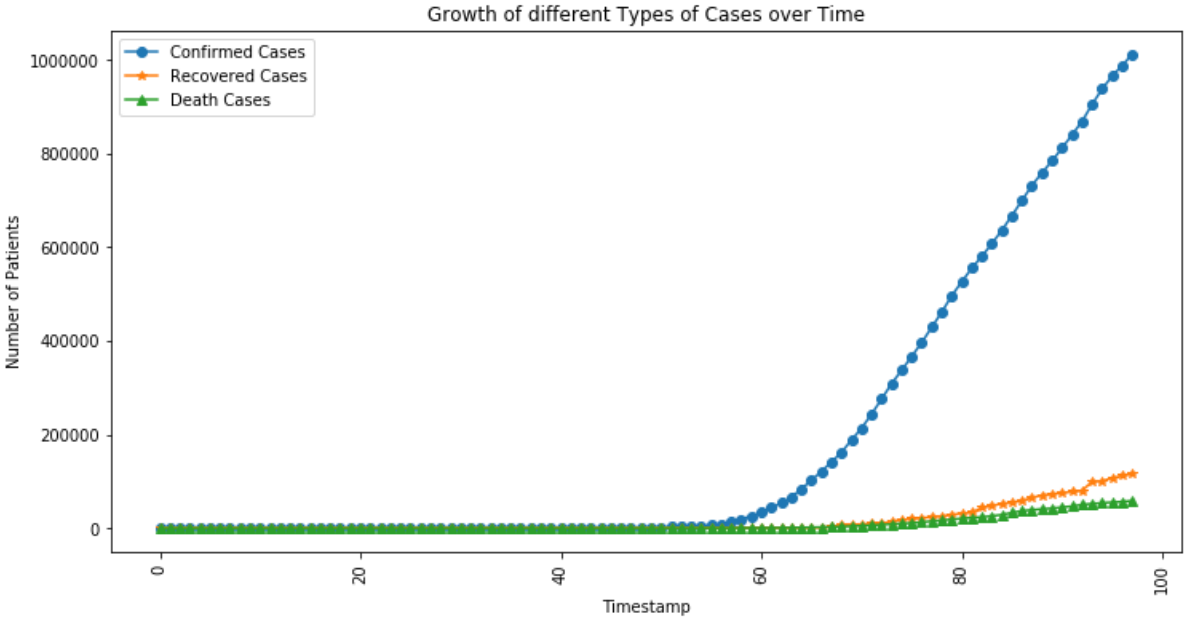
- **Daily Active Cases:**



- **Daily Closed (Deaths + Recovered) Cases:**

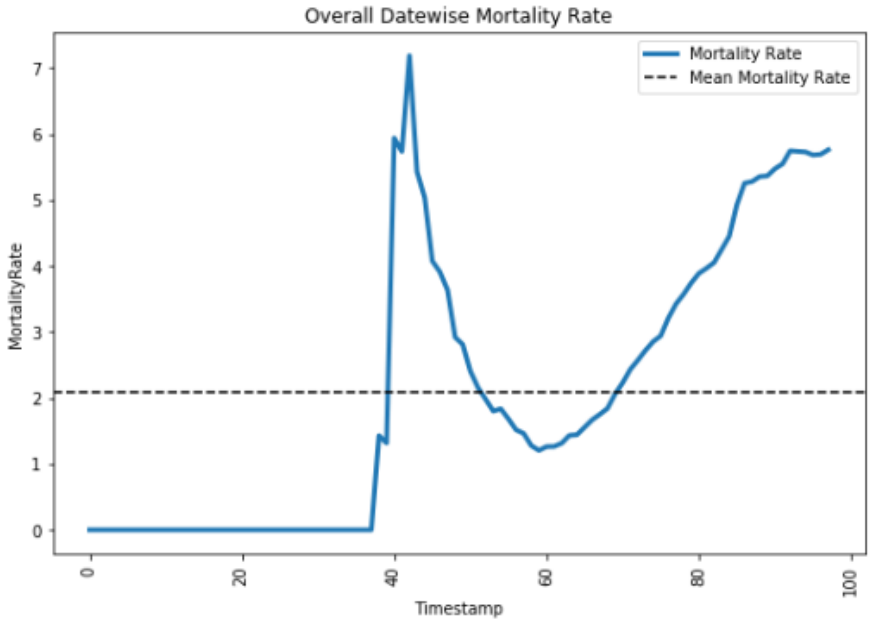


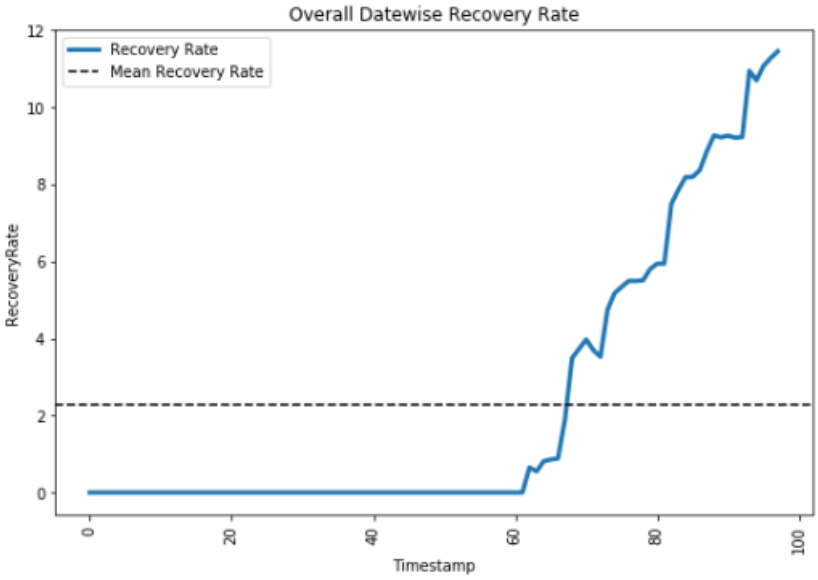
- **Growth Rate Analysis of Confirmed, Death, and Recovered Cases over time:**



- **Growth Rate Analysis of Confirmed, Death, and Recovered Cases over time:**

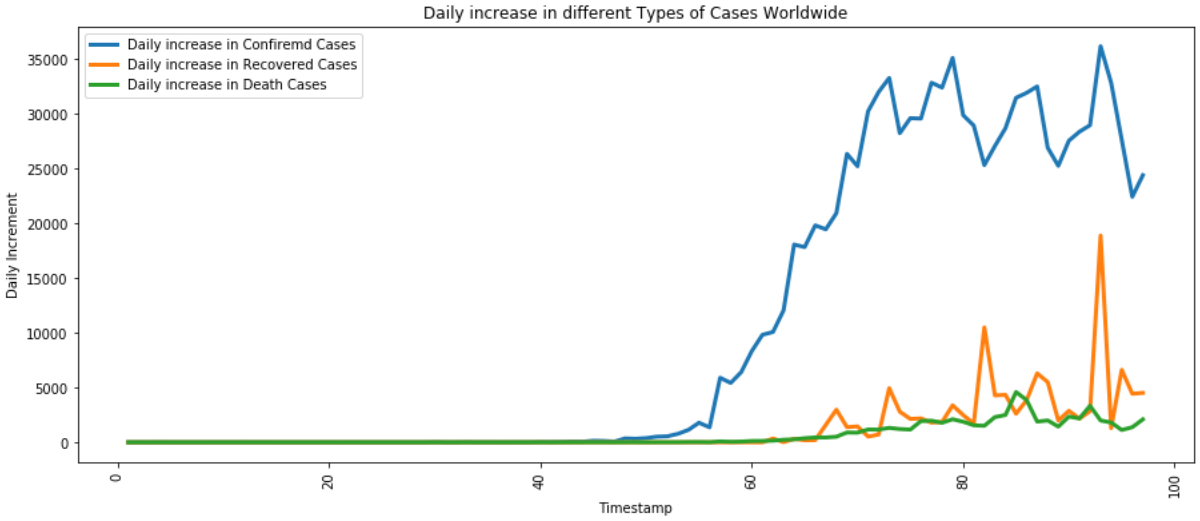
```
Mortality rate and Recovery Rate Analysis
Mortality rate = (Number of Death Cases / Number of Confirmed C
ases) x 100
Recovery Rate= (Number of Recoverd Cases / Number of Confirmed
Cases) x 100

Average Mortality Rate 2.0837672704739485
Median Mortality Rate 1.535385277814075
Average Recovery Rate 2.2863079142314673
Median Recovery Rate 0.0
```



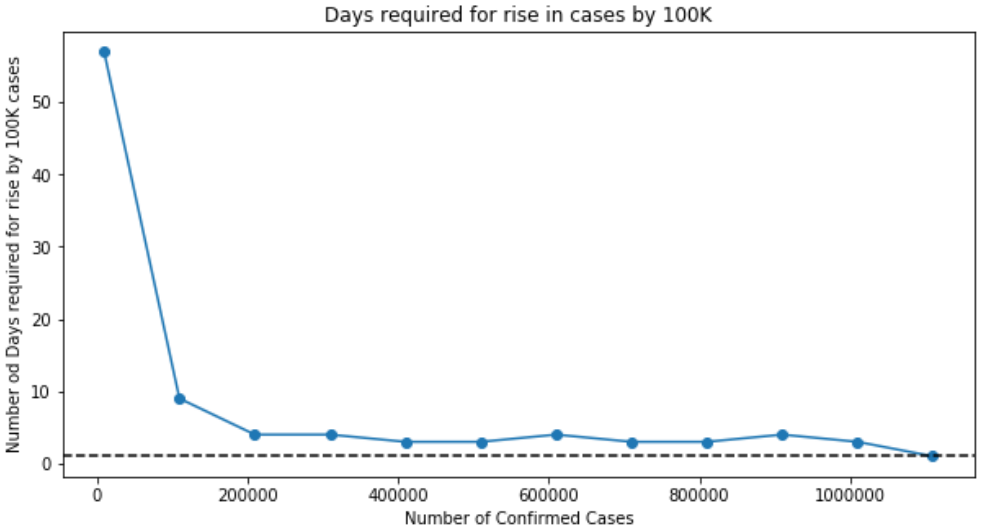


- **Average Daily increase in Confirmed, Death and Recovered Cases:**

```
Average increase in number of Confirmed Cases every day:  10439.0
Average increase in number of Recovered Cases every day:  1195.0
Average increase in number of Deaths Cases every day:  602.0
```



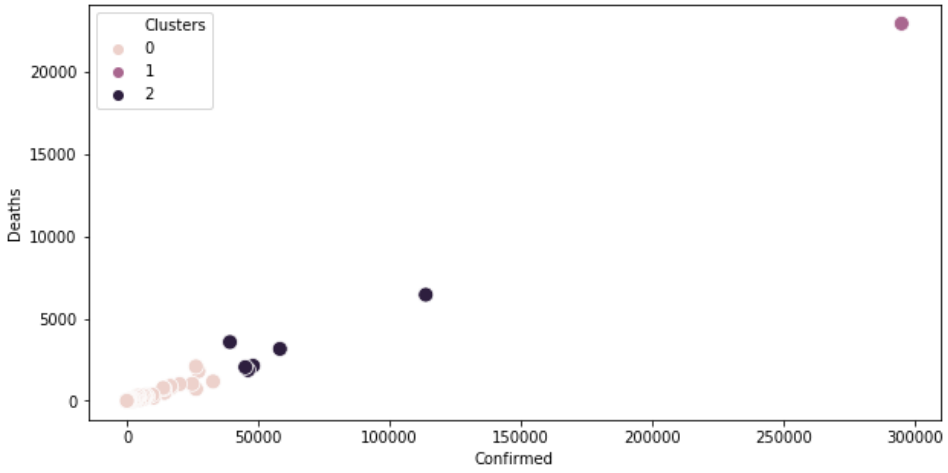
- **Days required for rise in cases by 100k:**



- **Scatter plot of states following similar trend using similarity between states:**

|:

Province/State	Confirmed	Deaths	Mortality	Clusters
New York	295106.0	22912.0	7.763990	1
New Jersey	113856.0	6442.0	5.658024	2
Massachusetts	58302.0	3153.0	5.408048	2
Illinois	48102.0	2125.0	4.417696	2
California	46164.0	1864.0	4.037778	2
Pennsylvania	45137.0	2046.0	4.532867	2
Michigan	39262.0	3568.0	9.087667	2
Florida	32848.0	1171.0	3.564905	0
Louisiana	27286.0	1801.0	6.600454	0
Texas	26357.0	719.0	2.727928	0



❖ [B] SEIR Modelling (Simulation Based Modelling Approach)

The SEIR model is a compartmental model for modeling how a disease spreads through a population. It's an acronym for **Susceptible, Exposed, Infected, Recovered**.

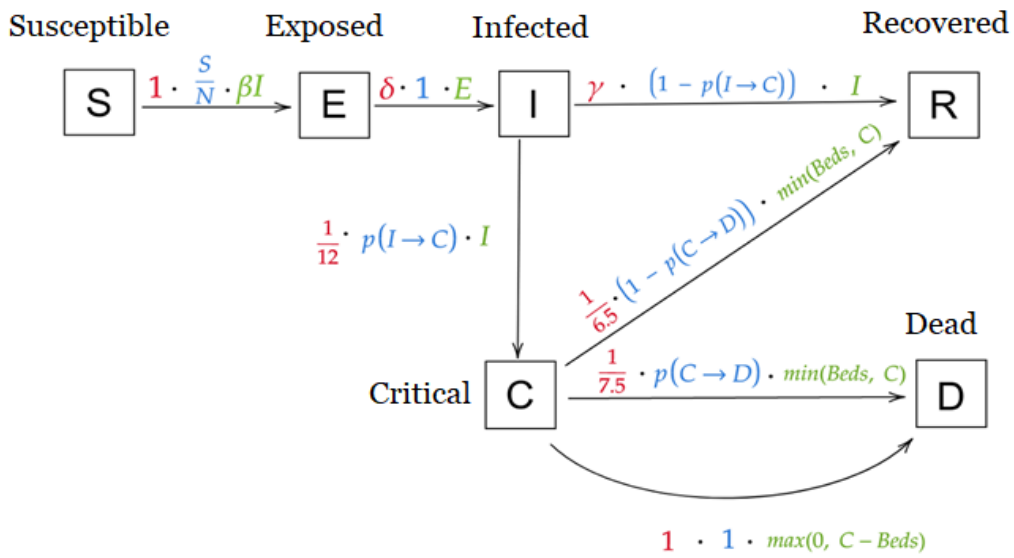
When a disease is introduced to a population, the people move from one of these classes (or compartments) to the next. When they reach the **R** state, they're no longer able to be infected, depending on your interpretation, they either survived the disease and are now immune or succumbed to the illness and are out of the population.

The SEIR model is a compartmental model for modeling how a disease spreads through a population. It's an acronym for Susceptible, Exposed, Infected, Recovered. When a disease is introduced to a population, the people move from one of these classes (or compartments) to the next. When they reach the R state, they're no longer able to be infected, depending on your interpretation, they either survived the disease and are now immune or succumbed to the illness and are out of the population

```
In [0]: def deriv(y, t, N, beta, gamma, delta):
            S, E, I, R = y
            dSdt = -beta * S * I / N
            dEdt = beta * S * I / N - delta * E
            dIdt = delta * E - gamma * I
            dRdt = gamma * I
            return dSdt, dEdt, dIdt, dRdt
```

- **N:** total population
- **S(t):** number of people susceptible on day t
- **I(t):** number of people infected on day t
- **R(t):** number of people recovered on day t
- **β:** expected amount of people an infected person infects per day
- **D:** number of days an infected person has and can spread the disease
- **γ:** the proportion of infected recovering per day (γ = 1/D)
- **R_o:** the total number of people an infected person infects (R_o = β / γ)

- **Extended SEIR Model**

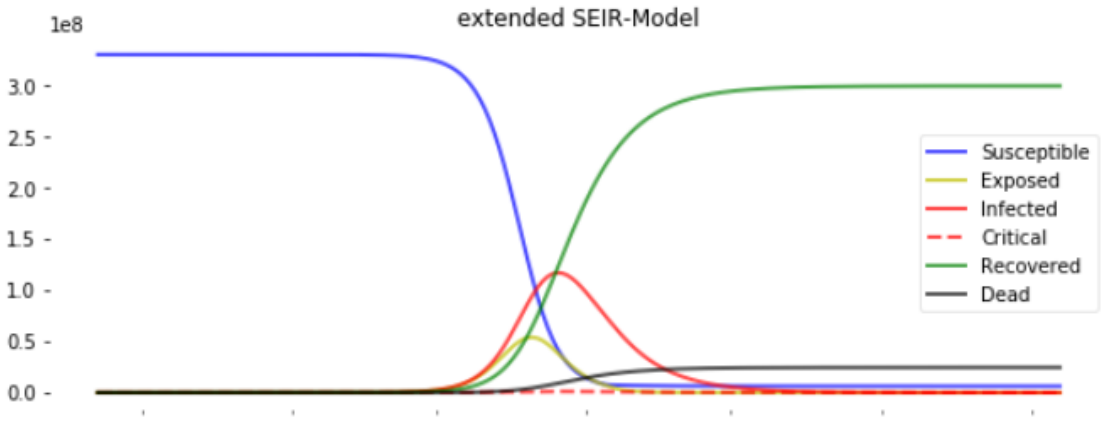


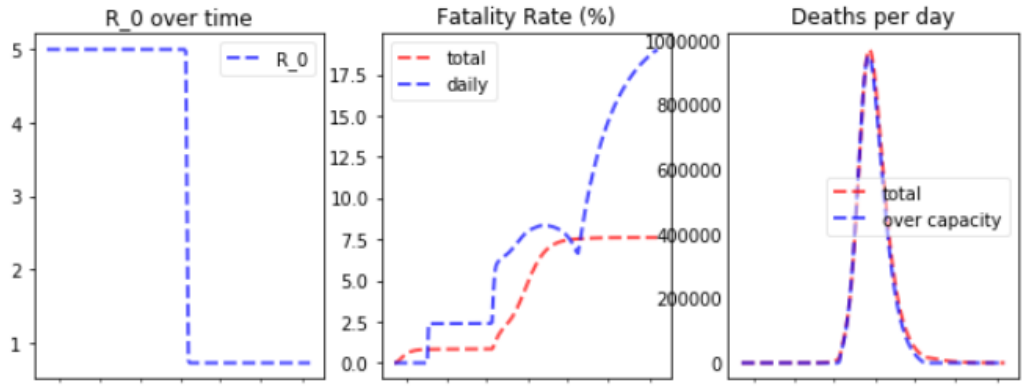
Updated Variables with Explanations:

- **N:** total population
- **S(t):** number of people susceptible on day t
- **E(t):** number of people exposed on day t
- **I(t):** number of people infected on day t
- **R(t):** number of people recovered on day t
- **D(t):** number of people dead on day t
- **β:** expected amount of people an infected person infects per day
- **D:** number of days an infected person has and can spread the disease
- **γ:** the proportion of infected recovering per day (γ = 1/D)
- **R_0:** the total number of people an infected person infects (R_0 = β / γ)
- **δ:** length of incubation period
- **α:** fatality rate
- **ρ:** rate at which people die (= 1/days from infected until death

Now Updated Equations will be

```
def deriv(y, t, beta, gamma, sigma, N, p_I_to_C, p_C_to_D, Beds):
    S, E, I, C, R, D = y

    dSdt = -beta(t) * I * S / N
    dEdt = beta(t) * I * S / N - sigma * E
    dIdt = sigma * E - 1/12.0 * p_I_to_C * I - gamma * (1 - p_I_to_C) * I
    dCdt = 1/12.0 * p_I_to_C * I - 1/7.5 * p_C_to_D * min(Beds(t), C) - max(0, C-Beds(t))
                - (1 - p_C_to_D) * 1/6.5 * min(Beds(t), C)
    dRdt = gamma * (1 - p_I_to_C) * I + (1 - p_C_to_D) * 1/6.5 * min(Beds(t), C)
    dDdt = 1/7.5 * p_C_to_D * min(Beds(t), C) + max(0, C-Beds(t))
    return dSdt, dEdt, dIdt, dCdt, dRdt, dDdt
```


extended SEIR-Model
1e8
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Susceptible
Exposed
Infected
Critical
Recovered
Dead

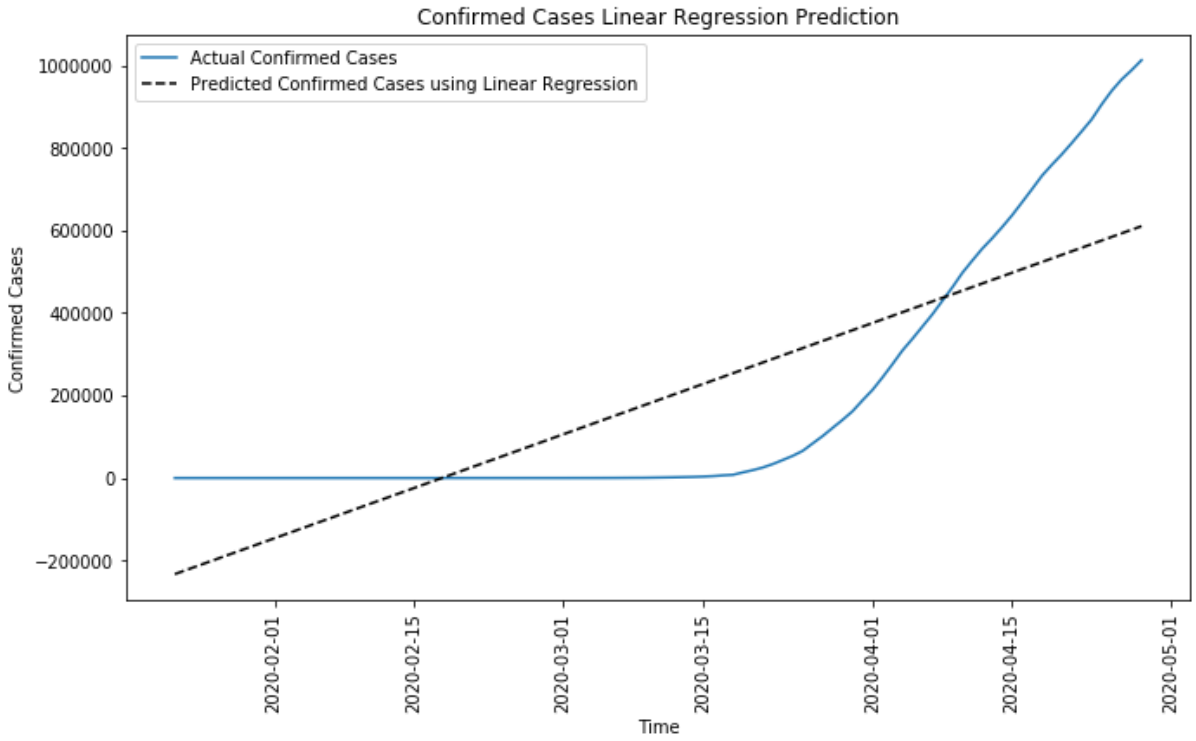

R_0 over time
R_0
Fatality Rate (%)
total
daily
Deaths per day
total
over capacity

❖ [C] Prediction Using Machine Learning Models

- **Linear Regression Model for Confirm Cases Prediction**

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
Root Mean Squared Error:  179241.2841306577
```

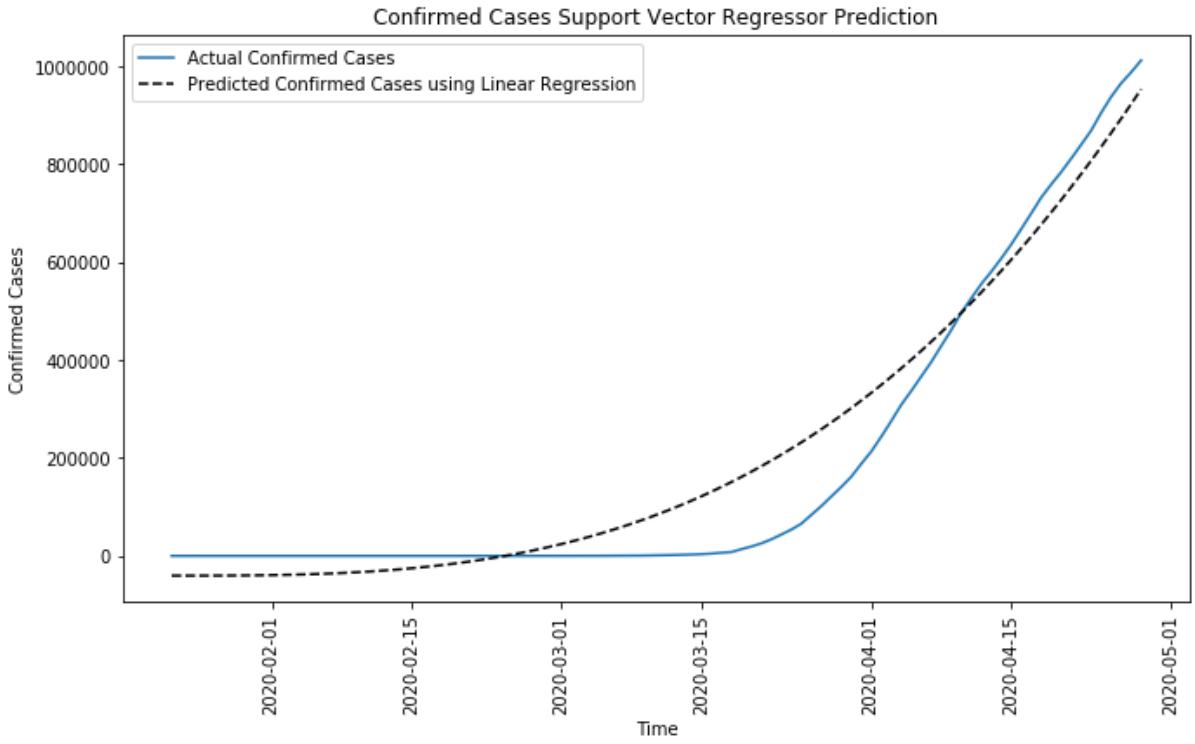


Observation: The Linear Regression Model is absolutely falling apart. As it is clearly visible that the trend of Confirmed Cases in not Linear.

- **Support Vector Machine Model Regressor for Prediction of Confirmed Cases**

```
SVR(C=1, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
    gamma='auto_deprecated', kernel='poly', max_iter=-1, shrinking=True,
    tol=0.001, verbose=False)
```

```
Root Mean Squared Error:  76909.09277820226
```

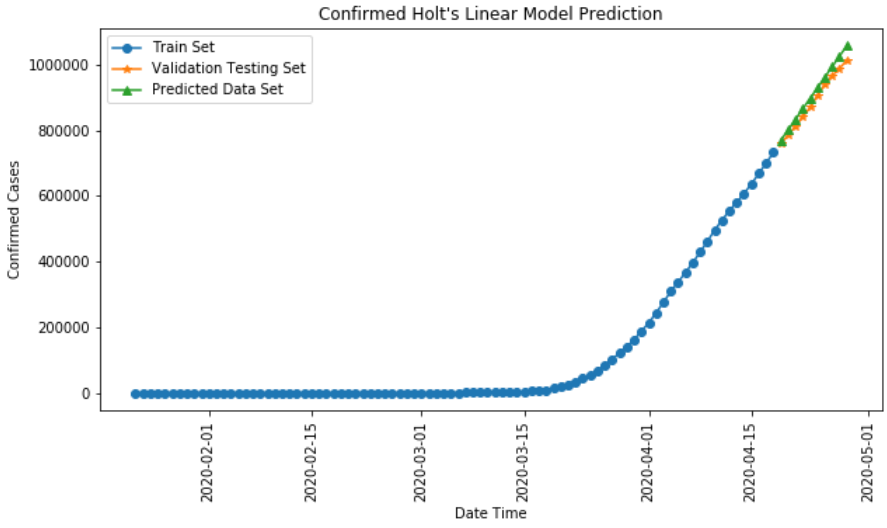


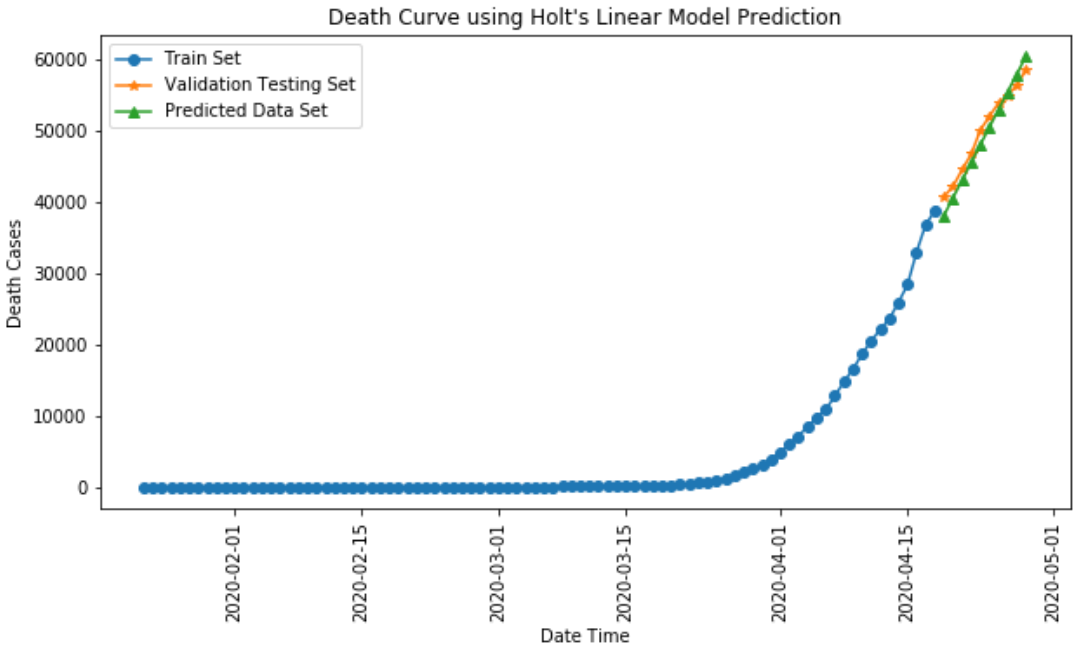
Observation: SVM Regression tries to fit the data, but other techniques might work better in this case. Regression is not a good option to proceed with.

- **Time Series Forecasting**
  1. **Holt's Linear Trend Model**

```
holt=Holt(X_train["Confirmed"])
holt=holt.fit(smoothing_level=0.2, smoothing_slope=0.2)
```

```
Root Mean Squared Error:  27297.30950495782
```





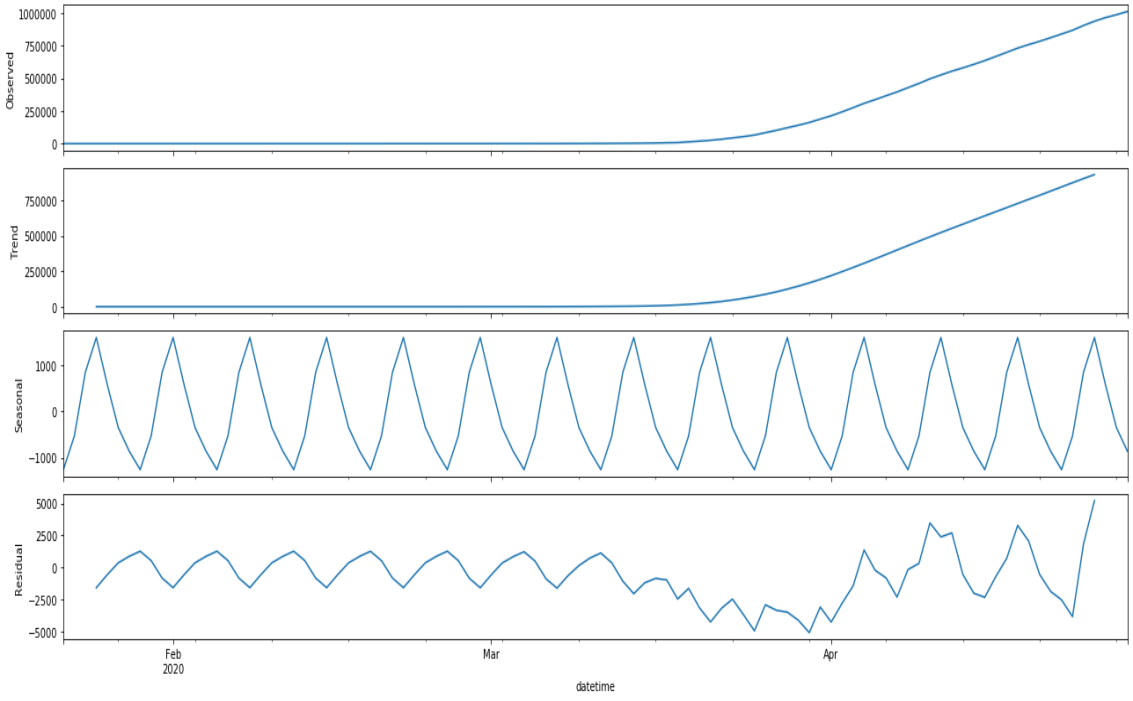
2. ARIMA Model

ARIMA Itself comprises of AR (Auto Regression) and MA (Moving Average)
I stand for integration.
We will first compute results of AR models and MA models and then for better results we will implement complete ARIMA.

3. AR Model (using AUTO ARIMA)

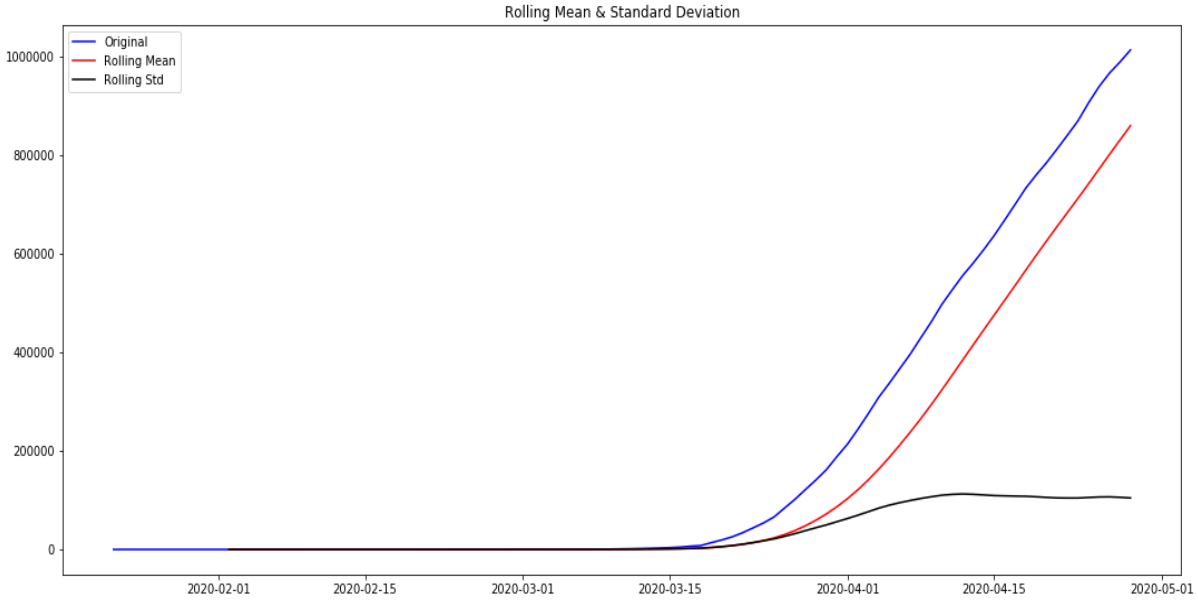
Checking the Data on the basis of Trend, Seasonality, Observed, Residual (Noise/Irregularity)

A. Using Visualization



B. Using Dickey Fuller Test

Determining Rolling Statistics

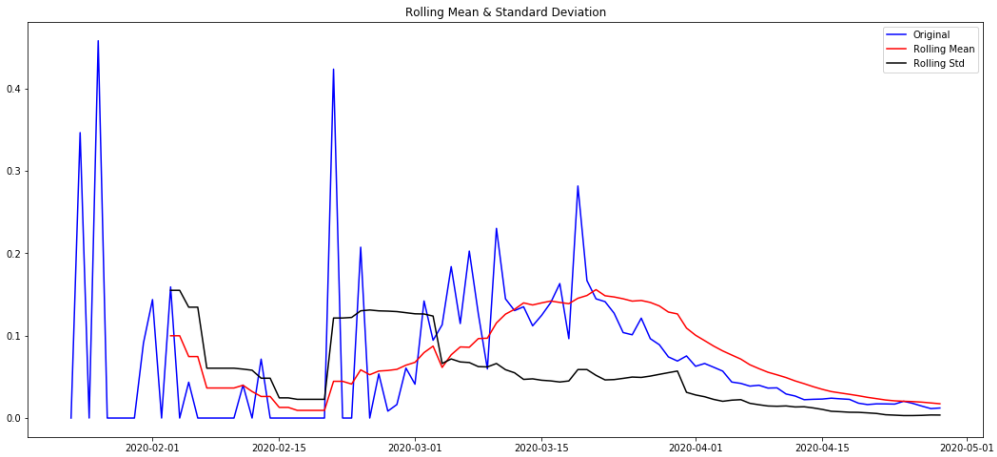



```
Results of Dickey-Fuller Test:
(-1.1111540172278438, 0.7106033762256079, 12, 85, {'1%': -3.5097356063504983, '5%': -2.8961947486260944, '10%': -2.585257612
4567475}, 1488.2479018752283)
Test Statistic                 -1.111154
p-value                         0.710603
#Lags Used                     12.000000
Number of Observations Used    85.000000
Critical Value (1%)            -3.509736
Critical Value (5%)            -2.896195
Critical Value (10%)           -2.585258
dtype: float64
```

We can see that p-value is greater than 0.5. We fail to reject the null hypothesis. So, the data is not stationary.

From the above graph variance is small but mean is not constant throughout the data. To make the Data stationary we will transform the data set by:

1. Taking Logarithm
2. Subtracting Moving average from the data set

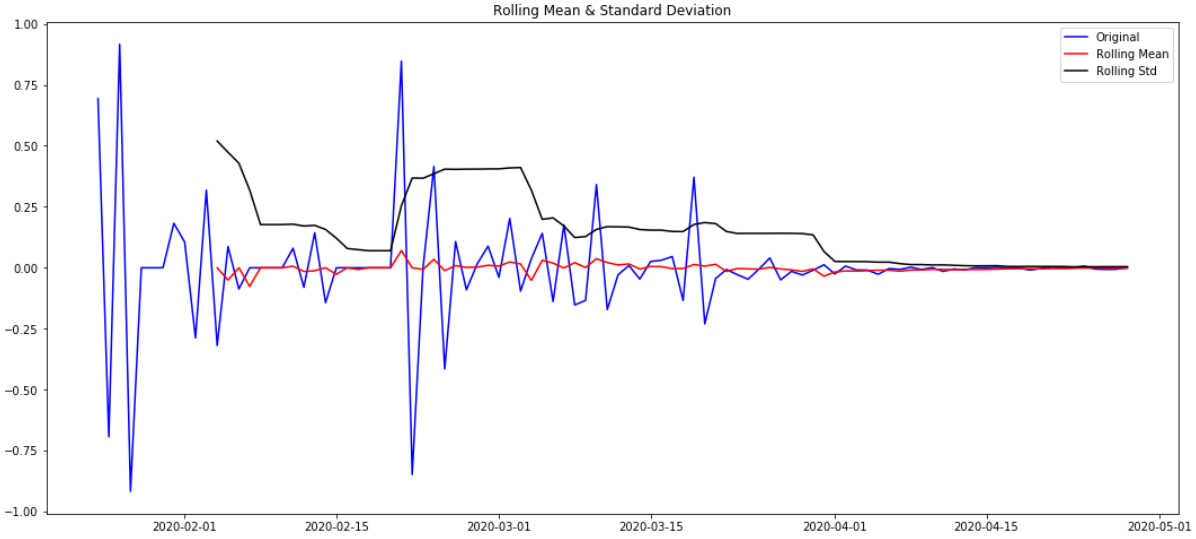


```
Results of Dickey-Fuller Test:
(-3.036071916228506, 0.03166031258698449, 2, 94, {'1%': -3.5019123847798657, '5%': -2.892815255482889, '10%': -2.583453861475781}, -228.34707059577295)
Test Statistic                -3.036072
p-value                        0.031660
#Lags Used                     2.000000
Number of Observations Used   94.000000
Critical Value (1%)           -3.501912
Critical Value (5%)           -2.892815
Critical Value (10%)          -2.583454
dtype: float64
```

Observations

p-value is still high and also the Test Statistics and Critical Values are not equal. Therefore, this transformation will not work.

Applying diff () - Find discrete difference



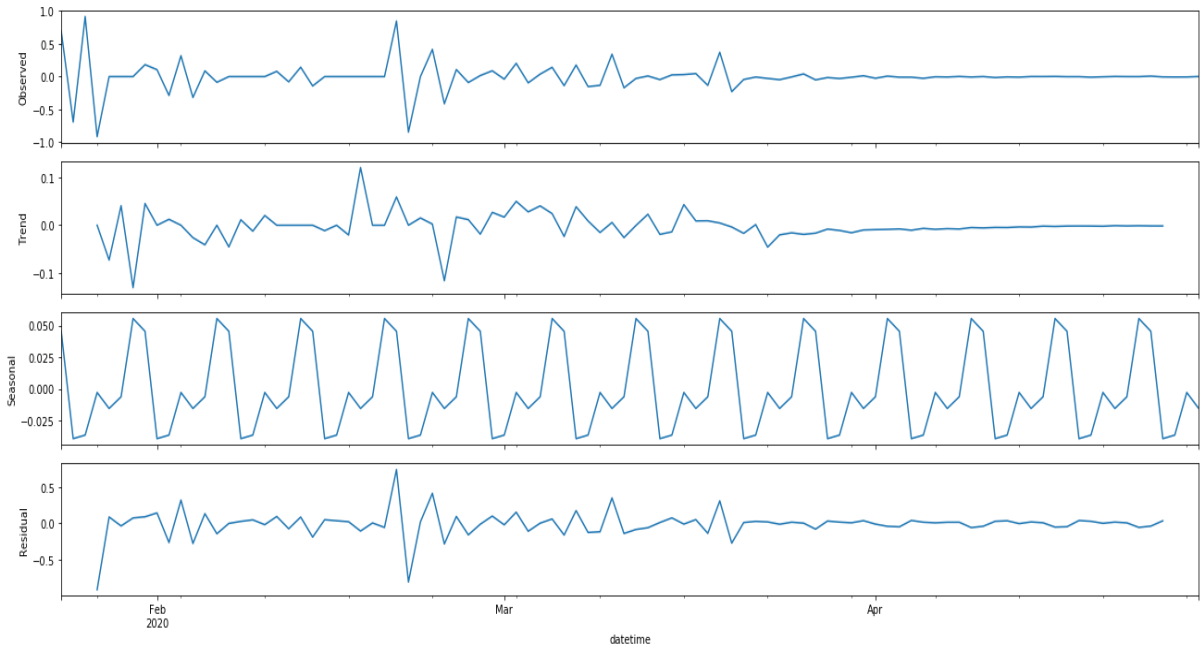

```
Results of Dickey-Fuller Test:
(-12.585439975773342, 1.870401882961233e-23, 1, 94, {'1%': -3.5019123847798657, '5%': -2.892815255482889, '10%': -2.583453861475781}, -107.24939062802267)
Test Statistic               -1.258544e+01
p-value                       1.870402e-23
#Lags Used                    1.000000e+00
Number of Observations Used   9.400000e+01
Critical Value (1%)          -3.501912e+00
Critical Value (5%)          -2.892815e+00
Critical Value (10%)         -2.583454e+00
dtype: float64
```

Observations

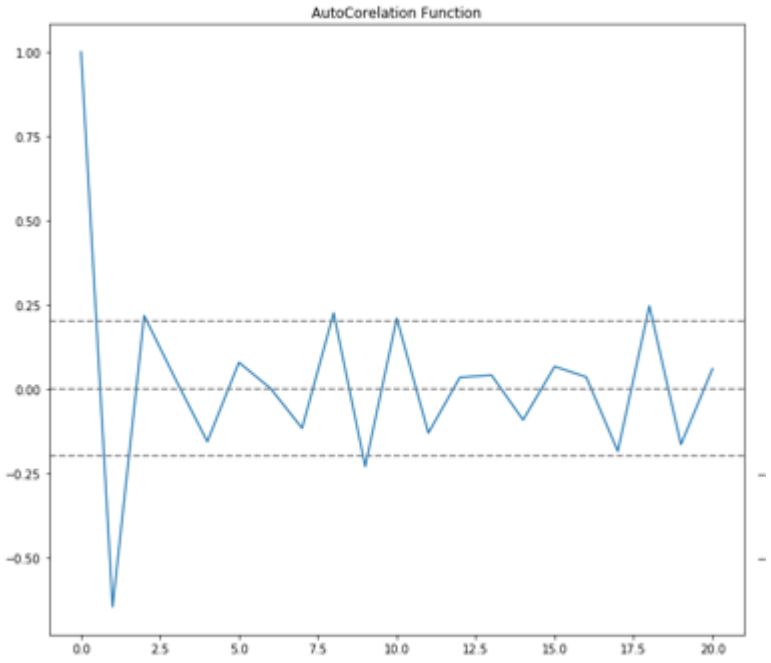
p-value has almost become 0 and therefore, we can reject the null hypothesis and by performing the transformations we have made the data stationary and ready to use in ARIMA Model.

Re-visualizing the Trend, Observed, Seasonality, Residual of discrete-discrete log data sets

```
Root Mean Square Error for AR Model:  48771.355887998
```

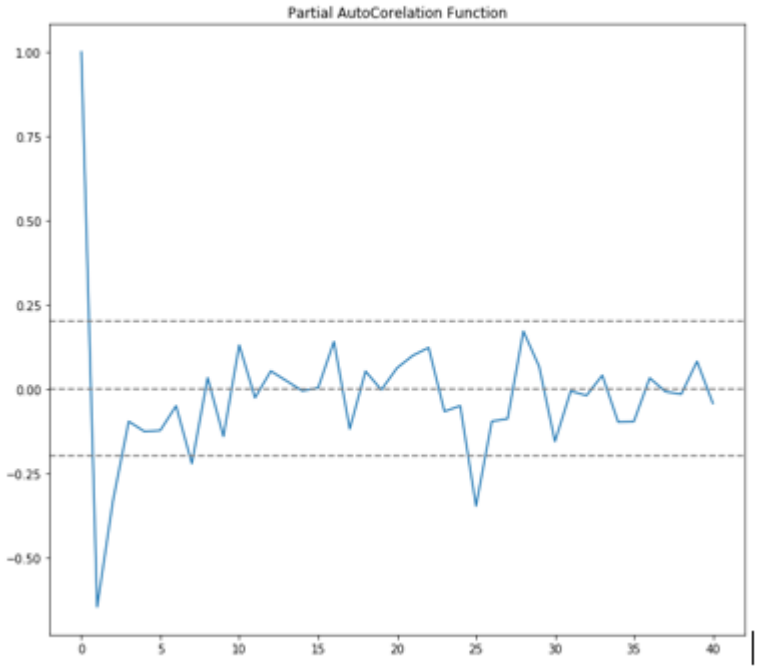


Calculating the value of ACF and PACF:



From here we can observe that q = 1

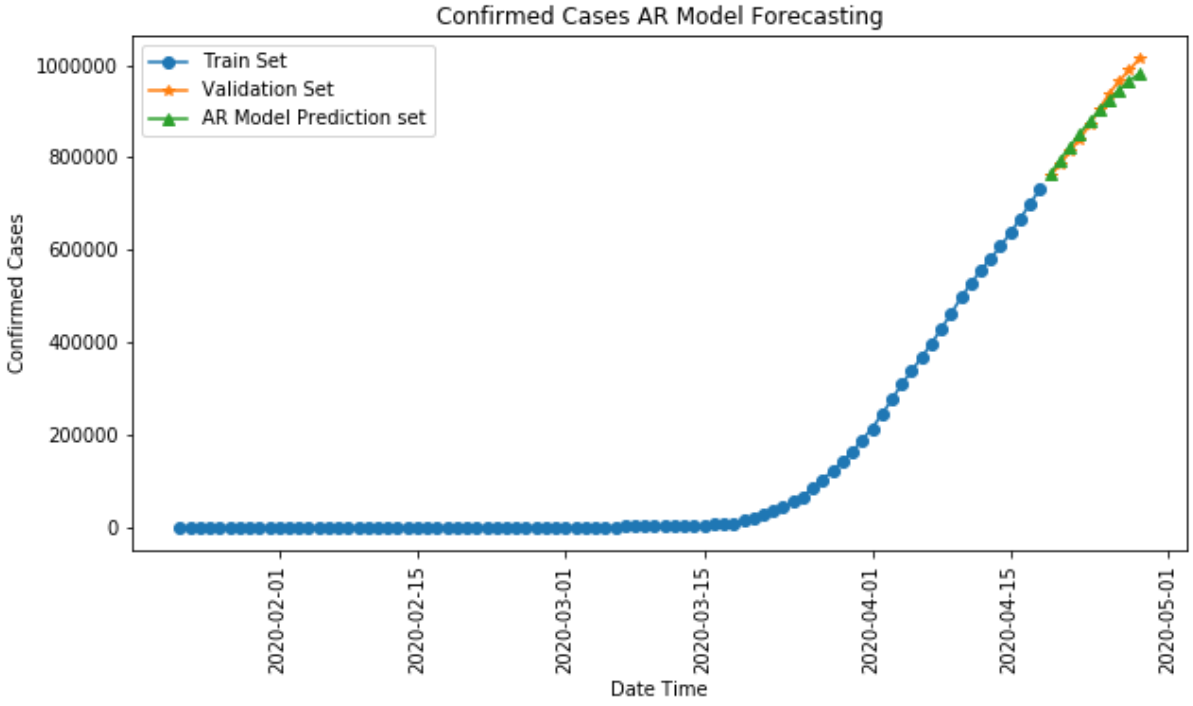
Point where it crosses first threshold



From here we can observe p =1

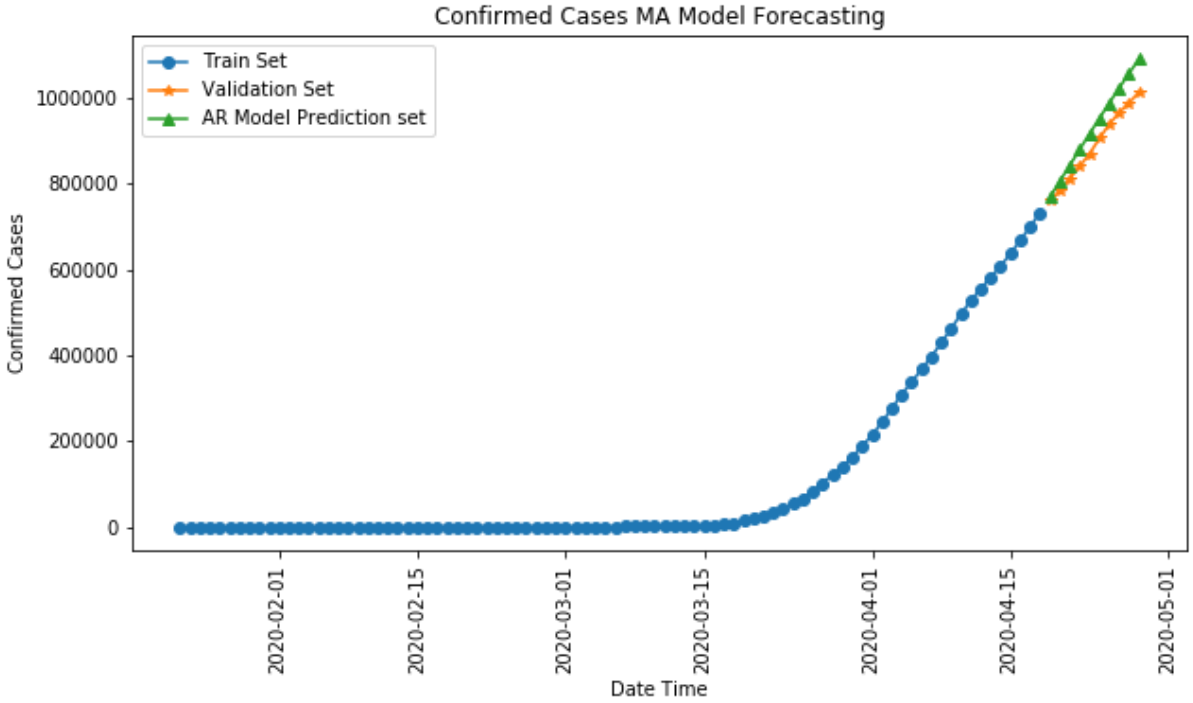
Point where it crosses first threshold

```
Root Mean Square Error for AR Model:  15944.200267095406
```



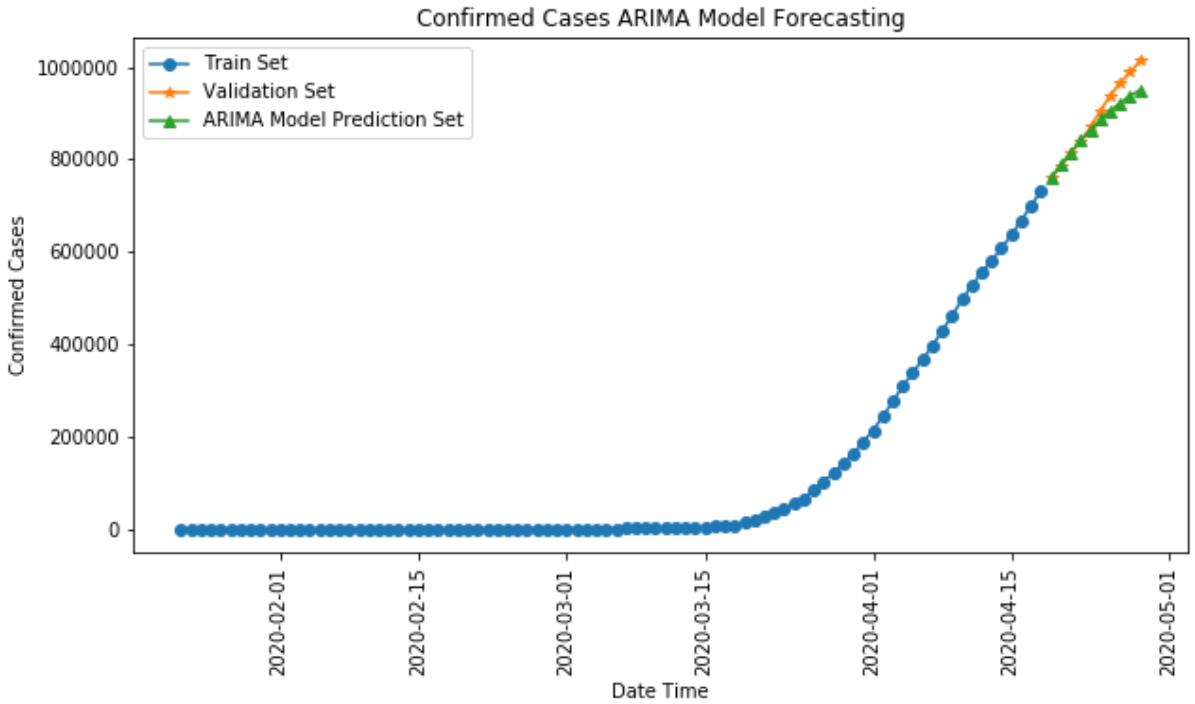
4. MA Model (using AUTO ARIMA)

```
Root Mean Square Error for AR Model:  48771.355887998
```



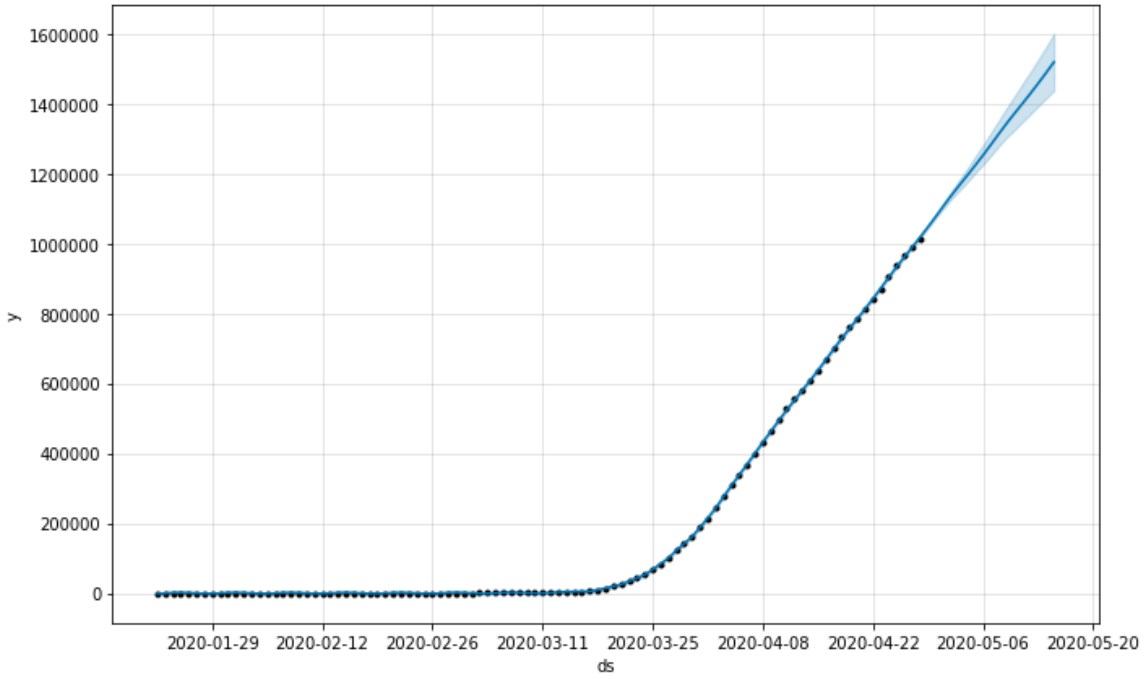
5. ARIMA Modelling

```
Root Mean Square Error for ARIMA Model:  31572.870901059312
```

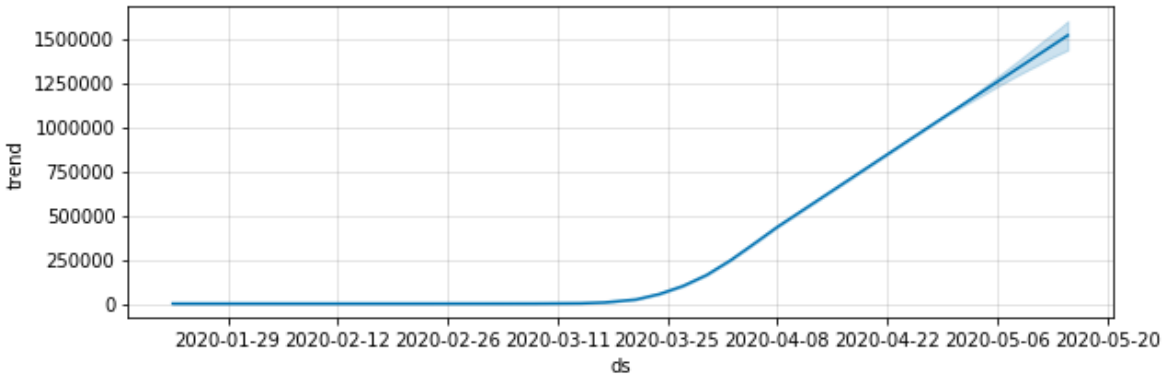


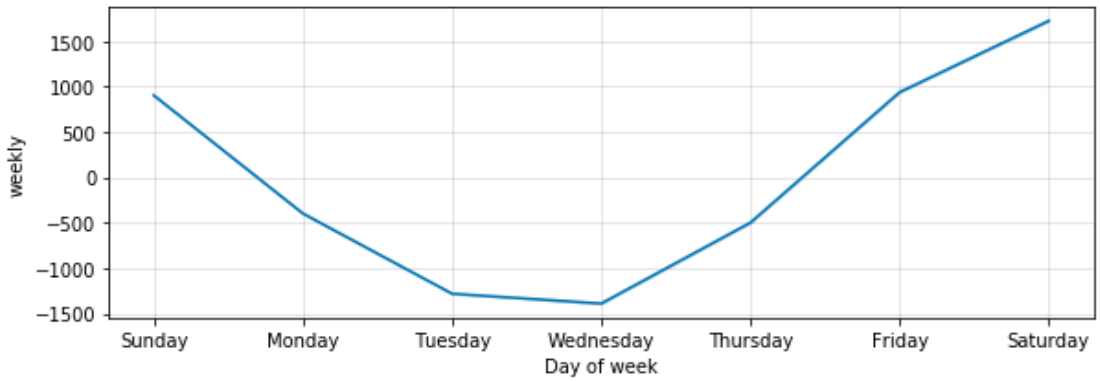
6. Facebook Prophet Model:

```
Root Mean Squared Error for Prophet Model:  12216.923277446744
```



Seasonal Components:





SUMMARY OF MODEL ACCURACY

	Root Mean Squared Error
Linear Regression	179241.284131
SVM Regressor	76909.092778
Holt's Linear	27297.309505
AR Model	15944.200267
MA Model	48771.355888
ARIMA Model	31572.870901
Facebook Prophet	12216.923277

Observation: From the above Accuracy Matrix we can clearly see that Facebook Prophet best fits the data set and predicts more accurate results.

- **Challenges**
  1. Faced challenges while collecting and authenticating data source.
  2. Very specific and limited data was available for United States as Recovery Data is not available state-wise.
  3. Difficulty in Analyzing which model to prefer, that's why compared all the time series models.
  4. ML Algorithms are data hungry and we do not have large amount of old/historical data or COVID-19 data, so to think about the transformations to how to make data stationary for time series prediction.
- **Applications**
  1. The models apply advanced analytics to hospital data to help optimize resources, improve situational awareness, and ensure demand-planning stability.
  2. Using this information, Government can predict and plan for future demands on the health system, including ICU beds, personal protective equipment, and ventilators. After reviewing possible COVID-19 surge scenarios generated by the models, Government can activate a plan to prepare for its worst-case scenario.
  3. "As the COVID-19 virus spreads in the US and around the world, public health providers and other types of responders have been trying to understand which areas may require the greatest amount of aid. We built this geographic map of US hospital bed utilization and capacity to solve some of these challenges," said Este Geraghty, ESRI's Chief Medical Officer.
  4. The resulting models include flexible control of model parameters and different model approaches that consider regional health and demographic variations and state-level assumptions.
  5. The analytics models were designed to create worst-case, best-case, and most-likely scenarios, unlike some forecasts that focus on a prediction based on a single set of assumptions. The models can adjust in real time as situations and data change, such as social distancing measures and disease spread.

FACE MASK DETECTION

❖ What is Face Mask Detection?

Looking at how serious COVID-19 is, it is crucial to take proper precautions in order to ensure the safety of self and others. Masks, being the primary component of precaution is the need of the hour.

In order to prevent the spread of this wide spreading pandemic, proper system needs to be in place to monitor people and their actions. This system detects whether a person is wearing a mask or not, so that further actions can be taken.

❖ Current precautionary measures

COVID-19 may not be fatal but it spreads faster than other diseases, like common cold. Current precautions being taken are:

- Clean your hands often, either with soap and water for 20 seconds or a hand sanitizer that contains at least 60% alcohol.
- Avoid close contact with people who are sick.
- Put distance between yourself and other people (at least 6 feet).
- Cover your mouth and nose with a cloth face cover when around others.
- Cover your cough or sneeze with a tissue, then throw the tissue in the trash.
- Clean and disinfect frequently touched objects and surfaces daily.

❖ Objective of the System

Objective of this system is to help prevent the spread of COVID-19 outbreak by detecting proper use of face mask by general population in real time video streams with the help of OpenCV, Keras/Tensorflow, Deep Learning. We have built custom CNN models over several pre-trained ImageNet Neural Networks – MobileNetV2, ResNet50V2, VGG19, InceptionV3.

Steps:

- Load face mask dataset.
- Train face mask classifier with Keras/Tensorflow.
- Serialize face mask classifier to disk.
- Load face mask classifier from disk.
- Detect face in images/video streams.
- Extract each face Region of Interest (ROI).
- Apply face mask classifier to each face to determine “mask” or “no mask”.
- Show results.

❖ Data Sources

- Face Mask Data [Link]

❖ Libraries Used

- Pandas
- Numpy
- Keras
- Tensorflow

- Matplotlib
- OpenCV
- Imutils
- Scikit-learn

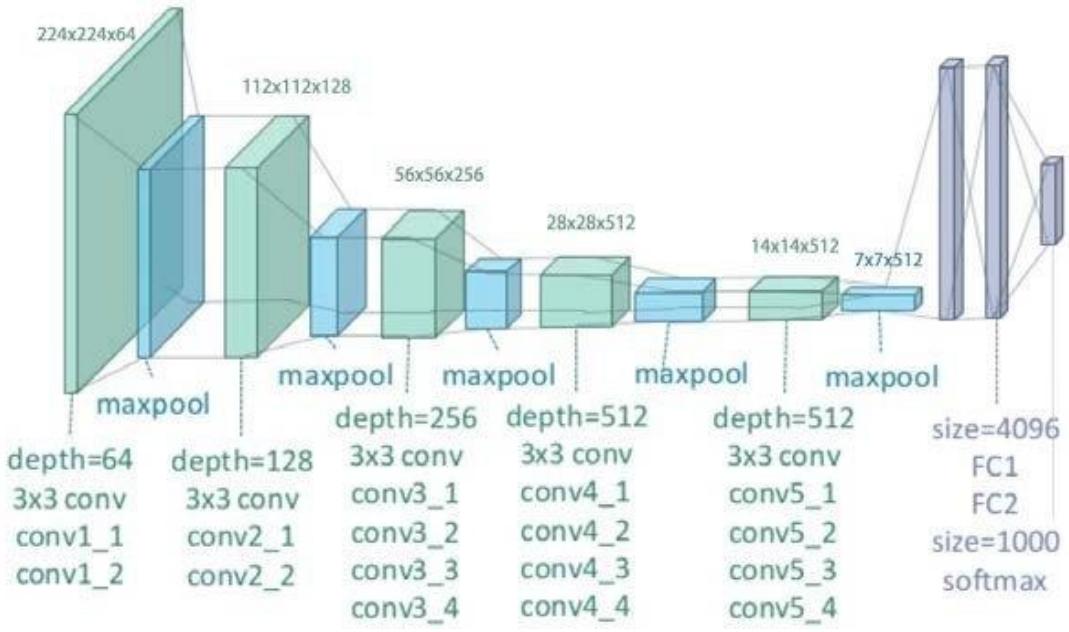
❖ Dataset Description

It is an artificially created dataset, by placing a mask image over actual image. It comprises of:

- with_mask [690 entries]
- without_mask [686 entries]

❖ ImageNet Models

- **VGG19**
  There are multiple variants of VGGNet (VGG16, VGG19 etc.) which differ only in the total number of layers in the network. The structural details of a VGG19 network has been shown below.



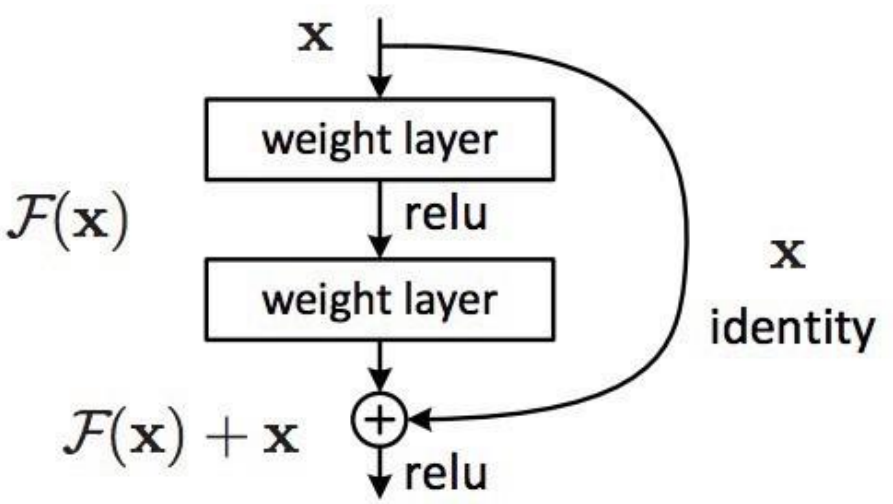
VGG19 Block Diagram (source: neurohive.io)

This network is characterized by its simplicity, using only *3×3* convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).
The "19" stand for the number of weight layers in the network

Two major drawbacks with VGG19:

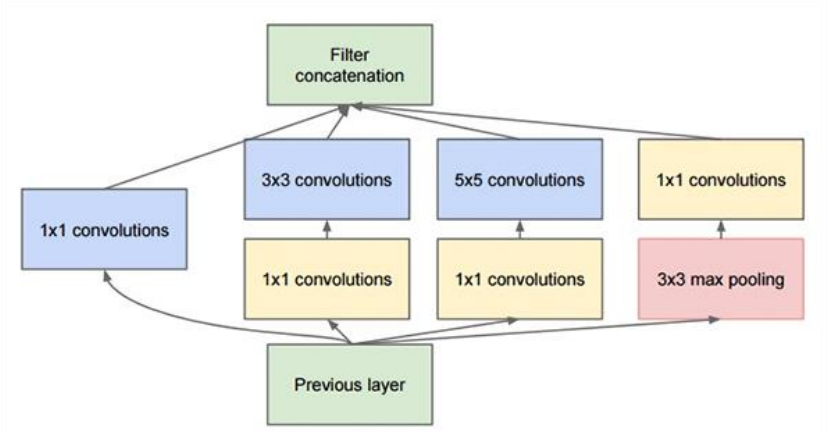
1. It is painfully slow to train.
2. The network architecture weights themselves are quite large (in terms of disk/bandwidth).

- **ResNet50V2**
  ResNet50V2 originated with the question, why do very deep nets perform worse as you keep adding layers? The authors of ResNet50V2 boiled these problems down to a single hypothesis: *direct mappings are hard to learn*. And they proposed a fix: instead of trying to learn an underlying mapping from x to H(x), learn the *difference* between the two, or the "residual." Then, to calculate H(x), we can just add the residual to the input.
  Say the residual is F(x)=H(x)-x. Now, instead of trying to learn H(x) directly, our nets are trying to learn F(x)+x.
  This gives rise to the famous ResNet50V2 (or "residual network") block you've probably seen:



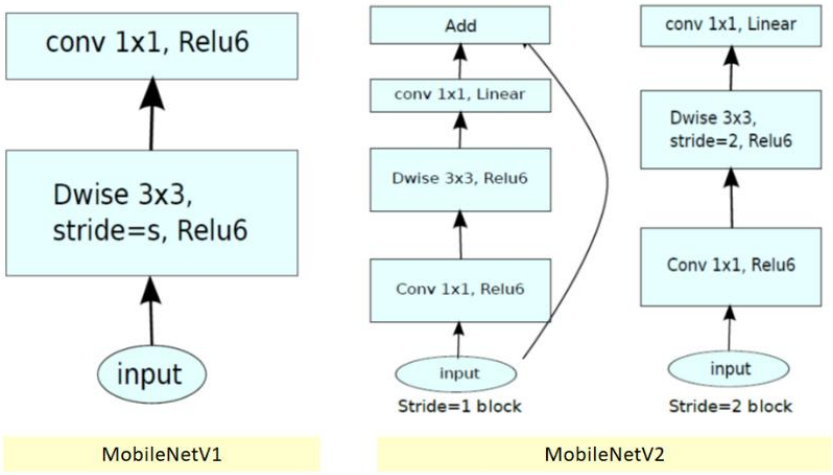
  Each "block" in ResNet50V2 consists of a series of layers and a "shortcut" connection adding the input of the block to its output. The "add" operation is performed element-wise, and if the input and output are of different sizes, zero-padding or projections (via 1x1 convolutions) can be used to create matching dimensions.

- **Inception V3**
  The goal of the inception module is to act as a "multi-level feature extractor" by computing *1×1*, *3×3*, and *5×5* convolutions within the *same* module of the network — the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network. The weights for Inception V3 are smaller than both VGG and ResNet50V2, coming in at 96MB.



- **MobileNetV2**

  In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing. There are 3 layers for both types of blocks. This time, the first layer is 1×1 convolution with ReLU6. The second layer is the depthwise convolution. The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain. And there is an expansion factor t. And t=6 for all main experiments. If the input got 64 channels, the internal output would get 64×t=64×6=384 channels.

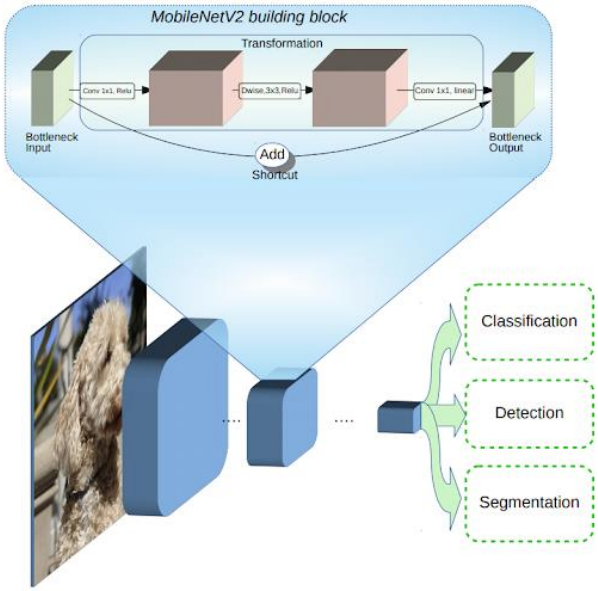


MobileNetV2 builds upon the ideas from MobileNetV1, using depth wise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture:

1) linear bottlenecks between the layers, and

2) shortcut connections between the bottlenecks1.

The basic structure is shown below.



❖ Mask Detection Model

Our set of tensorflow.keras imports allow for:

- Data augmentation where the random rotation, zoom, shear, shift, and flip parameters are established.
- Loading the Pre-trained classifiers – VGG, Inception, ResNet, MobileNet.
- Fine-tuning the models with pre-trained ImageNet weights
- Building a new fully-connected (FC) head
- Pre-processing
- Loading image data

We have used Scikit-learn (sklearn) for binarizing class labels, segmenting our dataset, and printing a classification report and we have used matplotlib to plot our training curves.

- First, we have loaded pre-trained models with pre-trained ImageNet weights, leaving off head of network, then we constructed a new FC head, and appended it to the base in place of the old head.
- Then, we have freezed the base layers of the network. The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights will be tuned.
- This strategy establishes a baseline model while saving considerable time.
- Next, we have compiled our model with the Adam optimizer, a learning rate decay schedule, and binary cross-entropy.
- Then we launch Face mask training and provide our data augmentation object with batches of mutated image data.
- Once training is complete, we'll evaluate the resulting model on the test set. Then we proceed on to make predictions on the test set, grabbing the highest probability class label indices. Then, we print a classification report and serialize our face mask classification model to disk.
- Our last step was to plot our accuracy and loss curves.

Implementing our model in Real time Video Streams:

We have run our models for each frame of the video using OpenCV. It detects faces and then applies our face mask classifier to each face ROI.

We construct a blob, detect faces, find Region Of Interests), face locations, loop over our detections and extract the confidence to measure against the --confidence threshold, filter out weak detections, extract bounding boxes while ensuring bounding box coordinates do not fall outside the bounds of the image and predict (the list of mask/no mask predictions)

Then we run our faces through our mask predictor.

❖ **Training Results:**

- **MobileNetV2**

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
30/30 [==============================] - 23s 778ms/step - loss: 0.0451 -
accuracy: 0.9854 - val_loss: 0.0143 - val_accuracy: 0.9976
Epoch 2/20
30/30 [==============================] - 24s 784ms/step - loss: 0.0261 -
accuracy: 0.9936 - val_loss: 0.0134 - val_accuracy: 0.9952
Epoch 3/20
30/30 [==============================] - 22s 741ms/step - loss: 0.0235 -
accuracy: 0.9925 - val_loss: 0.0130 - val_accuracy: 0.9927
Epoch 4/20
30/30 [==============================] - 22s 739ms/step - loss: 0.0193 -
accuracy: 0.9936 - val_loss: 0.0132 - val_accuracy: 0.9927
Epoch 5/20
30/30 [==============================] - 23s 766ms/step - loss: 0.0251 -
accuracy: 0.9903 - val_loss: 0.0104 - val_accuracy: 0.9976
Epoch 6/20
30/30 [==============================] - 23s 753ms/step - loss: 0.0187 -
accuracy: 0.9936 - val_loss: 0.0105 - val_accuracy: 0.9927
Epoch 7/20
30/30 [==============================] - 22s 748ms/step - loss: 0.0184 -
accuracy: 0.9968 - val_loss: 0.0081 - val_accuracy: 0.9976
Epoch 8/20
30/30 [==============================] - 23s 764ms/step - loss: 0.0135 -
accuracy: 0.9946 - val_loss: 0.0090 - val_accuracy: 0.9952
Epoch 9/20
30/30 [==============================] - 22s 729ms/step - loss: 0.0090 -
accuracy: 0.9989 - val_loss: 0.0072 - val_accuracy: 0.9976
Epoch 10/20
30/30 [==============================] - 22s 723ms/step - loss: 0.0183 -
accuracy: 0.9957 - val_loss: 0.0088 - val_accuracy: 0.9927
Epoch 11/20
30/30 [==============================] - 23s 775ms/step - loss: 0.0107 -
accuracy: 0.9957 - val_loss: 0.0074 - val_accuracy: 0.9976
Epoch 12/20
30/30 [==============================] - 22s 725ms/step - loss: 0.0073 -
accuracy: 0.9957 - val_loss: 0.0070 - val_accuracy: 0.9976
Epoch 13/20
30/30 [==============================] - 22s 743ms/step - loss: 0.0160 -
accuracy: 0.9946 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 14/20
30/30 [==============================] - 24s 795ms/step - loss: 0.0055 -
accuracy: 0.9989 - val_loss: 0.0064 - val_accuracy: 0.9976
Epoch 15/20
30/30 [==============================] - 22s 736ms/step - loss: 0.0085 -
accuracy: 0.9979 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 16/20
30/30 [==============================] - 22s 729ms/step - loss: 0.0107 -
accuracy: 0.9979 - val_loss: 0.0061 - val_accuracy: 0.9976
Epoch 17/20
30/30 [==============================] - 22s 735ms/step - loss: 0.0145 -
accuracy: 0.9968 - val_loss: 0.0058 - val_accuracy: 0.9976
Epoch 18/20
30/30 [==============================] - 22s 727ms/step - loss: 0.0180 -
accuracy: 0.9946 - val_loss: 0.0082 - val_accuracy: 1.0000
Epoch 19/20
```

```
30/30 [==============================] - 22s 750ms/step - loss: 0.0074 -
accuracy: 0.9979 - val_loss: 0.0053 - val_accuracy: 0.9976
Epoch 20/20
30/30 [==============================] - 22s 735ms/step - loss: 0.0036 -
accuracy: 1.0000 - val_loss: 0.0052 - val_accuracy: 0.9976
```

- **ResNet50V2**

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
30/30 [==============================] - 58s 2s/step - loss: 0.5224 - ac
curacy: 0.7510 - val_loss: 0.0691 - val_accuracy: 0.9903
Epoch 2/20
30/30 [==============================] - 58s 2s/step - loss: 0.0984 - ac
curacy: 0.9721 - val_loss: 0.0358 - val_accuracy: 0.9976
Epoch 3/20
30/30 [==============================] - 56s 2s/step - loss: 0.0581 - ac
curacy: 0.9860 - val_loss: 0.0260 - val_accuracy: 0.9976
Epoch 4/20
30/30 [==============================] - 56s 2s/step - loss: 0.0519 - ac
curacy: 0.9839 - val_loss: 0.0212 - val_accuracy: 0.9976
Epoch 5/20
30/30 [==============================] - 56s 2s/step - loss: 0.0412 - ac
curacy: 0.9882 - val_loss: 0.0188 - val_accuracy: 0.9976
Epoch 6/20
30/30 [==============================] - 57s 2s/step - loss: 0.0249 - ac
curacy: 0.9903 - val_loss: 0.0164 - val_accuracy: 0.9976
Epoch 7/20
30/30 [==============================] - 57s 2s/step - loss: 0.0218 - ac
curacy: 0.9957 - val_loss: 0.0148 - val_accuracy: 0.9976
Epoch 8/20
30/30 [==============================] - 56s 2s/step - loss: 0.0129 - ac
curacy: 0.9979 - val_loss: 0.0142 - val_accuracy: 0.9976
Epoch 9/20
30/30 [==============================] - 57s 2s/step - loss: 0.0214 - ac
curacy: 0.9936 - val_loss: 0.0139 - val_accuracy: 0.9976
Epoch 10/20
30/30 [==============================] - 56s 2s/step - loss: 0.0179 - ac
curacy: 0.9957 - val_loss: 0.0126 - val_accuracy: 0.9976
Epoch 11/20
30/30 [==============================] - 55s 2s/step - loss: 0.0142 - ac
curacy: 0.9957 - val_loss: 0.0127 - val_accuracy: 0.9976
Epoch 12/20
30/30 [==============================] - 55s 2s/step - loss: 0.0122 - ac
curacy: 0.9957 - val_loss: 0.0126 - val_accuracy: 0.9976
Epoch 13/20
30/30 [==============================] - 56s 2s/step - loss: 0.0085 - ac
curacy: 0.9989 - val_loss: 0.0121 - val_accuracy: 0.9976
Epoch 14/20
30/30 [==============================] - 56s 2s/step - loss: 0.0155 - ac
curacy: 0.9936 - val_loss: 0.0115 - val_accuracy: 0.9976
Epoch 15/20
30/30 [==============================] - 56s 2s/step - loss: 0.0096 - ac
curacy: 0.9968 - val_loss: 0.0108 - val_accuracy: 0.9976
Epoch 16/20
30/30 [==============================] - 56s 2s/step - loss: 0.0115 - ac
curacy: 0.9968 - val_loss: 0.0104 - val_accuracy: 0.9976
Epoch 17/20
```

```
30/30 [==============================] - 57s 2s/step - loss: 0.0064 - ac
curacy: 0.9989 - val_loss: 0.0101 - val_accuracy: 0.9976
Epoch 18/20
30/30 [==============================] - 61s 2s/step - loss: 0.0070 - ac
curacy: 0.9979 - val_loss: 0.0102 - val_accuracy: 0.9976
Epoch 19/20
30/30 [==============================] - 58s 2s/step - loss: 0.0056 - ac
curacy: 0.9979 - val_loss: 0.0101 - val_accuracy: 0.9976
Epoch 20/20
30/30 [==============================] - 57s 2s/step - loss: 0.0145 - ac
curacy: 0.9968 - val_loss: 0.0091 - val_accuracy: 0.9976
```

- **InceptionV3**

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
30/30 [==============================] - 40s 1s/step - loss: 0.0986 - ac
curacy: 0.9656 - val_loss: 0.0414 - val_accuracy: 0.9927
Epoch 2/20
30/30 [==============================] - 38s 1s/step - loss: 0.1158 - ac
curacy: 0.9592 - val_loss: 0.0364 - val_accuracy: 0.9927
Epoch 3/20
30/30 [==============================] - 42s 1s/step - loss: 0.1021 - ac
curacy: 0.9613 - val_loss: 0.0353 - val_accuracy: 0.9927
Epoch 4/20
30/30 [==============================] - 40s 1s/step - loss: 0.1065 - ac
curacy: 0.9613 - val_loss: 0.0314 - val_accuracy: 0.9952
Epoch 5/20
30/30 [==============================] - 39s 1s/step - loss: 0.0756 - ac
curacy: 0.9785 - val_loss: 0.0299 - val_accuracy: 0.9952
Epoch 6/20
30/30 [==============================] - 39s 1s/step - loss: 0.0850 - ac
curacy: 0.9678 - val_loss: 0.0271 - val_accuracy: 0.9976
Epoch 7/20
30/30 [==============================] - 41s 1s/step - loss: 0.0773 - ac
curacy: 0.9689 - val_loss: 0.0274 - val_accuracy: 0.9952
Epoch 8/20
30/30 [==============================] - 37s 1s/step - loss: 0.0616 - ac
curacy: 0.9785 - val_loss: 0.0253 - val_accuracy: 0.9952
Epoch 9/20
30/30 [==============================] - 40s 1s/step - loss: 0.0574 - ac
curacy: 0.9817 - val_loss: 0.0232 - val_accuracy: 0.9952
Epoch 10/20
30/30 [==============================] - 38s 1s/step - loss: 0.0871 - ac
curacy: 0.9699 - val_loss: 0.0209 - val_accuracy: 0.9976
Epoch 11/20
30/30 [==============================] - 37s 1s/step - loss: 0.0714 - ac
curacy: 0.9753 - val_loss: 0.0203 - val_accuracy: 0.9976
Epoch 12/20
30/30 [==============================] - 37s 1s/step - loss: 0.0602 - ac
curacy: 0.9817 - val_loss: 0.0200 - val_accuracy: 0.9952
Epoch 13/20
30/30 [==============================] - 37s 1s/step - loss: 0.0745 - ac
curacy: 0.9699 - val_loss: 0.0207 - val_accuracy: 0.9952
Epoch 14/20
30/30 [==============================] - 38s 1s/step - loss: 0.0694 - ac
curacy: 0.9753 - val_loss: 0.0175 - val_accuracy: 0.9952
Epoch 15/20
30/30 [==============================] - 38s 1s/step - loss: 0.0592 - ac
curacy: 0.9807 - val_loss: 0.0200 - val_accuracy: 0.9952
```

```
Epoch 16/20
30/30 [==============================] - 36s 1s/step - loss: 0.0640 - ac
curacy: 0.9807 - val_loss: 0.0196 - val_accuracy: 0.9952
Epoch 17/20
30/30 [==============================] - 36s 1s/step - loss: 0.0587 - ac
curacy: 0.9817 - val_loss: 0.0190 - val_accuracy: 0.9952
Epoch 18/20
30/30 [==============================] - 38s 1s/step - loss: 0.0557 - ac
curacy: 0.9850 - val_loss: 0.0148 - val_accuracy: 0.9976
Epoch 19/20
30/30 [==============================] - 37s 1s/step - loss: 0.0630 - ac
curacy: 0.9710 - val_loss: 0.0167 - val_accuracy: 0.9952
Epoch 20/20
30/30 [==============================] - 36s 1s/step - loss: 0.0691 - ac
curacy: 0.9785 - val_loss: 0.0164 - val_accuracy: 0.9976
```

- **VGG19**

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
30/30 [==============================] - 209s 7s/step - loss: 0.7287 - a
ccuracy: 0.5427 - val_loss: 0.6300 - val_accuracy: 0.7143
Epoch 2/20
30/30 [==============================] - 203s 7s/step - loss: 0.6674 - a
ccuracy: 0.6037 - val_loss: 0.5637 - val_accuracy: 0.8886
Epoch 3/20
30/30 [==============================] - 201s 7s/step - loss: 0.6101 - a
ccuracy: 0.6595 - val_loss: 0.5112 - val_accuracy: 0.9370
Epoch 4/20
30/30 [==============================] - 203s 7s/step - loss: 0.5264 - a
ccuracy: 0.7691 - val_loss: 0.4644 - val_accuracy: 0.9613
Epoch 5/20
30/30 [==============================] - 201s 7s/step - loss: 0.5135 - a
ccuracy: 0.7766 - val_loss: 0.4242 - val_accuracy: 0.9685
Epoch 6/20
30/30 [==============================] - 206s 7s/step - loss: 0.4519 - a
ccuracy: 0.8432 - val_loss: 0.3878 - val_accuracy: 0.9685
Epoch 7/20
30/30 [==============================] - 198s 7s/step - loss: 0.4157 - a
ccuracy: 0.8754 - val_loss: 0.3546 - val_accuracy: 0.9734
Epoch 8/20
30/30 [==============================] - 203s 7s/step - loss: 0.3920 - a
ccuracy: 0.8969 - val_loss: 0.3250 - val_accuracy: 0.9782
Epoch 9/20
30/30 [==============================] - 206s 7s/step - loss: 0.3628 - a
ccuracy: 0.9055 - val_loss: 0.3004 - val_accuracy: 0.9782
Epoch 10/20
30/30 [==============================] - 204s 7s/step - loss: 0.3401 - a
ccuracy: 0.9076 - val_loss: 0.2787 - val_accuracy: 0.9806
Epoch 11/20
30/30 [==============================] - 209s 7s/step - loss: 0.3210 - a
ccuracy: 0.9173 - val_loss: 0.2585 - val_accuracy: 0.9806
Epoch 12/20
30/30 [==============================] - 205s 7s/step - loss: 0.2913 - a
ccuracy: 0.9345 - val_loss: 0.2351 - val_accuracy: 0.9806
Epoch 13/20
30/30 [==============================] - 209s 7s/step - loss: 0.2616 - a
ccuracy: 0.9549 - val_loss: 0.2206 - val_accuracy: 0.9806
Epoch 14/20
```

```
30/30 [==============================] - 206s 7s/step - loss: 0.2497 - a
ccuracy: 0.9474 - val_loss: 0.2044 - val_accuracy: 0.9806
Epoch 15/20
30/30 [==============================] - 215s 7s/step - loss: 0.2283 - a
ccuracy: 0.9689 - val_loss: 0.1883 - val_accuracy: 0.9855
Epoch 16/20
30/30 [==============================] - 210s 7s/step - loss: 0.2244 - a
ccuracy: 0.9452 - val_loss: 0.1757 - val_accuracy: 0.9879
Epoch 17/20
30/30 [==============================] - 211s 7s/step - loss: 0.2078 - a
ccuracy: 0.9613 - val_loss: 0.1671 - val_accuracy: 0.9806
Epoch 18/20
30/30 [==============================] - 202s 7s/step - loss: 0.1986 - a
ccuracy: 0.9581 - val_loss: 0.1544 - val_accuracy: 0.9855
Epoch 19/20
30/30 [==============================] - 207s 7s/step - loss: 0.1857 - a
ccuracy: 0.9678 - val_loss: 0.1443 - val_accuracy: 0.9879
Epoch 20/20
30/30 [==============================] - 206s 7s/step - loss: 0.1715 - a
ccuracy: 0.9742 - val_loss: 0.1355 - val_accuracy: 0.9879
```

❖ **Classification Report:**

- **MobileNetV2**

```
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       1.00      1.00      1.00       207
without_mask       1.00      1.00      1.00       206

    accuracy                           1.00       413
   macro avg       1.00      1.00      1.00       413
weighted avg       1.00      1.00      1.00       413
```

- **ResNet50V2**

```
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       1.00      1.00      1.00       207
without_mask       1.00      1.00      1.00       206

    accuracy                           1.00       413
   macro avg       1.00      1.00      1.00       413
weighted avg       1.00      1.00      1.00       413
```

- **InceptionV3**

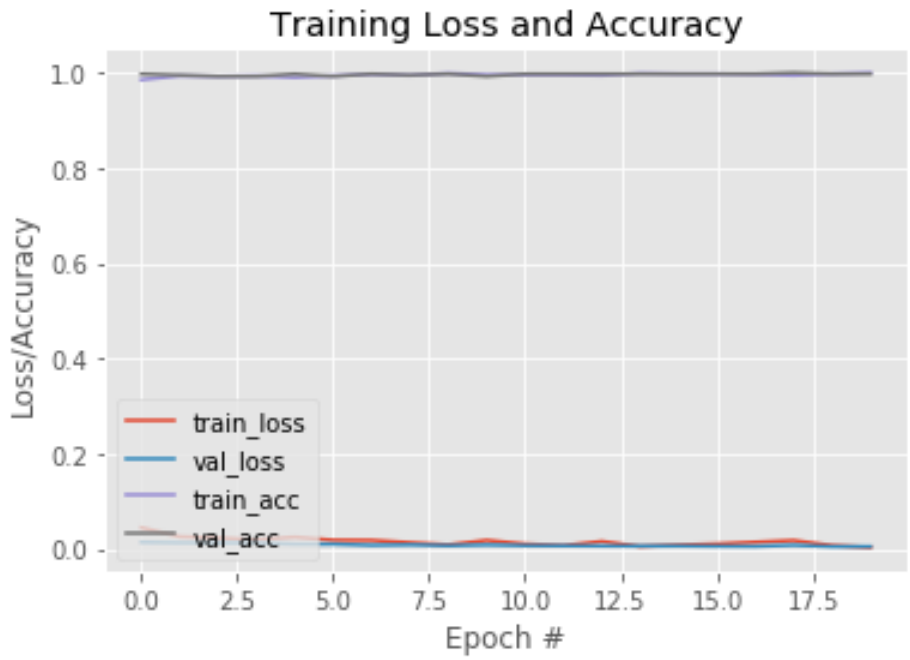
```
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       1.00      1.00      1.00       207
without_mask       1.00      1.00      1.00       206

    accuracy                           1.00       413
   macro avg       1.00      1.00      1.00       413
weighted avg       1.00      1.00      1.00       413
```

- **VGG19**

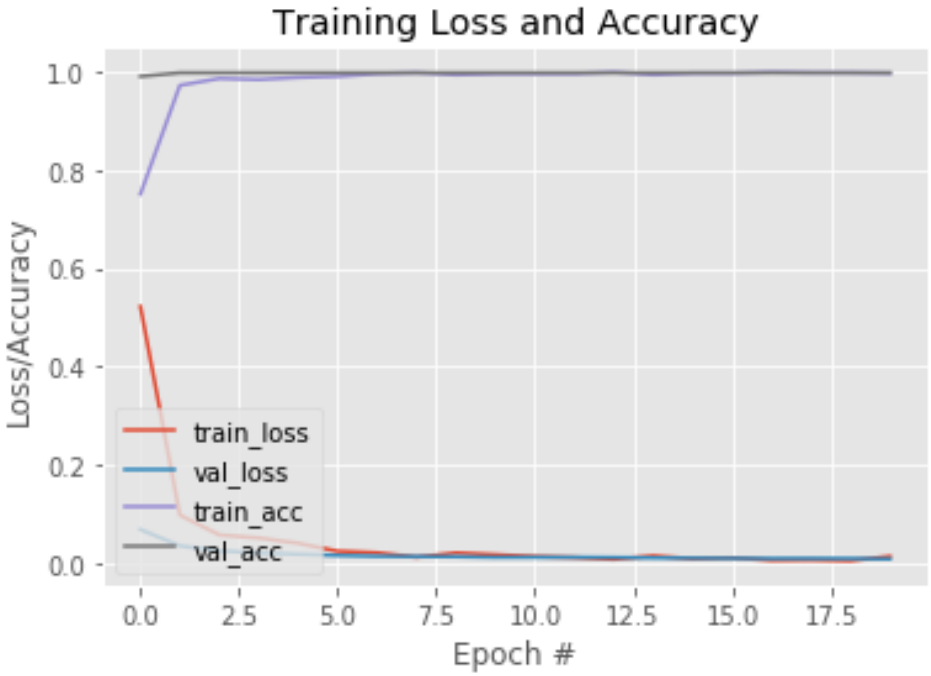
```
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       0.99      0.99      0.99       207
without_mask       0.99      0.99      0.99       206

    accuracy                           0.99       413
   macro avg       0.99      0.99      0.99       413
weighted avg       0.99      0.99      0.99       413
```

❖ **Graph/Plots:**

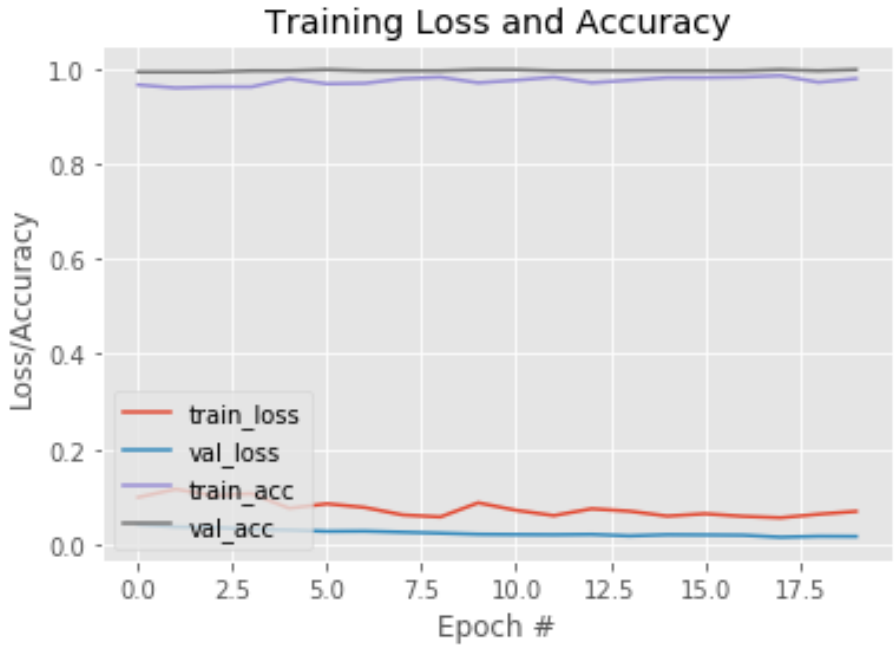
- **MobileNetV2**



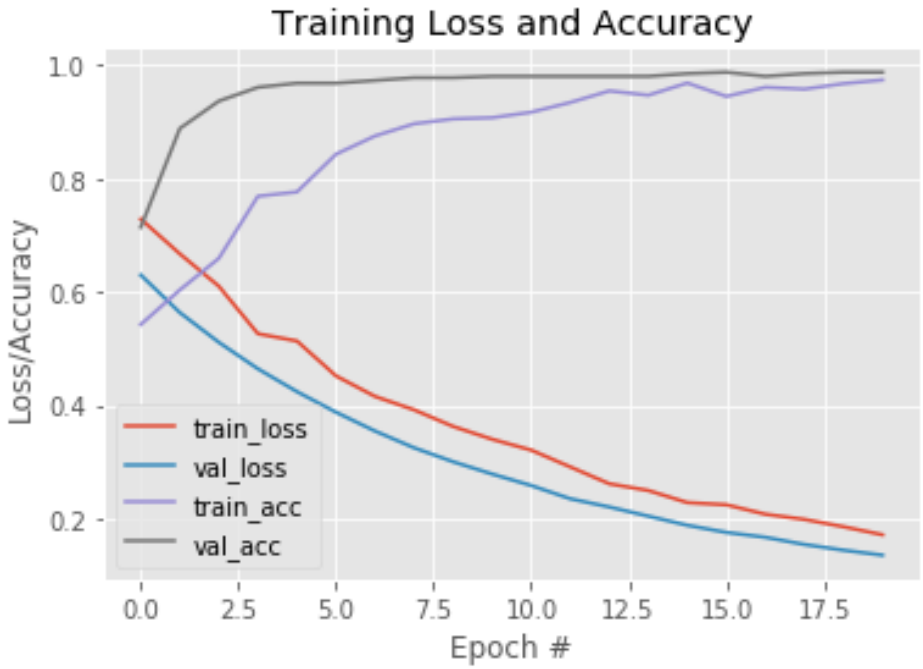
- **ResNet50V2**



- **InceptionV3**



- **VGG19**



❖ Real time Video Stream Results:

Results are uploaded in the Results_Mask_Detection folder.

Here are Google Drive Links for the same.

- MobileNetV2
- ResNet50V2
- InceptionV3
- VGG19

❖ Conclusion:

- All these models are having a very good accuracy.
- In terms of accuracy, accuracy of VGG19 increases vastly over time/epochs. Whereas, the accuracy for ResNet50V2 is less at the start but quickly gains the accuracy with increase in the number of epochs. InceptionV3 has good accuracy from the start which flickers little sometimes. MobileNetV2 was observed to have the best accuracy and it was steadily maintained. All these accuracies are in terms of training and validation test.
- In terms of speed, VGG19 took the maximum time to train and was the slowest with approx. 7s/step and approx. 209s for an epoch. Whereas, InceptionV3 took 1s/step and 40s for an epoch. ResNet50V2 took 2s/step and 56s for an epoch. MobileNetV2 took about 750ms/step and 22s to complete an epoch. MobileNetV2 being the fastest to train.
- While testing on real streaming video, we could observe some lag and discrepancies in VGG19 and extreme smoothness in MobileNetV2, and little less in the other two models.
- We can say, InceptionV3 focuses on computational cost, ResNet50V2 focuses on computational accuracy. VGG19 gets better with the training but takes away a lot of computation time. MobileNetsV2 are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases.

❖ Challenges:

- Deciding the layer configurations for our model was a challenging task.
- Visualization using OpenCV was a tough part to do.
- Learning advance use of Neural Networks and understanding them was difficult.
- Deciding on factors like confidence, learning rate, number of epochs and which pre-trained models to study on was quite confusing.
- Training several models takes a large amount of time and computation resources.
- Testing on Real Time Video Streams takes a lot of processing.

❖ Improvements:

- Multiple for-loops can be improved by using different data structures or vectorization and Numpy/Pandas. Doing this could save time and computational resources by making the models more efficient.
- Models misclassifies the videos when a person is using a fake mask or a wrapper or any other substitute, more varied training data could help improve the accuracy. IN short, model gets confused in such cases.
- Gathering actual images (rather than artificially generated images) of people wearing masks would definitely help model predict well on varied and generalized test data.
- We should consider training a dedicated two-class object detector rather than a simple image classifier as this would make the models more computationally efficient

❖ **Applications:**

- Mask detection models can be very helpful in preventing the spread of air-borne disease like COVID19, flu, etc.
- This model can aid in tracking the source of spread of virus so that its transmission can be controlled and proper distancing from the infected person can be done timely.
- Extending this model by incorporating it in any public space can help account for the number of violators of any such rules like compulsory wearing of masks. Further strict actions can be taken in order to curb such behaviors.
- Such models can be generalized to several different use cases and can aid in monitoring activities of people which might be harmful to their and others health.

Team Contribution:

The project was planned and implemented by all group members.

1. We all discussed the design of the project.
2. The code was built in group together with discussion and peer reviews within the group.
3. Library functions were studied by each member individually.
4. The project was divided into 3 sections equally and each section was done individually by each member which is again discussed and reviewed and necessary suggestions were incorporated.

REFERENCES

- "Very Deep Convolutional Networks for Large-Scale Image Recognition", Karen Simonyan, Andrew Zisserman
- "Deep Residual Learning for Image Recognition", Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
- "Rethinking the Inception Architecture for Computer Vision", Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna
- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam
- https://www.idmod.org/docs/hiv/model-seir.html
- https://machinelearningmastery.com/time-series-forecasting/
- https://medium.com/future-vision/intro-to-prophet-9d5b1cbd674e
- https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
- https://www.bing.com/covid
- https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series
- https://otexts.com/fpp2/holt.html
- https://www.cdc.gov/coronavirus/2019-ncov/covid-data/forecasting-us.html
- https://medium.com/@fransiska26/the-differences-between-inception-resnet-and-mobilenet-e97736a709b0
- https://www.pyimagesearch.com/opencv-tutorials-resources-guides/
- https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c
- https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html
- https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96
- https://github.com/prajnasb/observations/tree/master/mask_classifier/Data_Generator