A PROJECT REPORT

ON

COVID 19 DATA ANALYSIS AND FUTURE PREDICTION USING

VARIOUS MACHINE LEARNING ALGORITHIM

Submitted in partial fulfillment for the requirement of the award of

TRAINING

IN

Data Analytics, Machine Learning and AI using Python

Submitted By

Manish Kumar (R.V.S College of Engineering and Technology, Jamshedpur**)**

Under the guidance of

Mr. Bipul Shahi

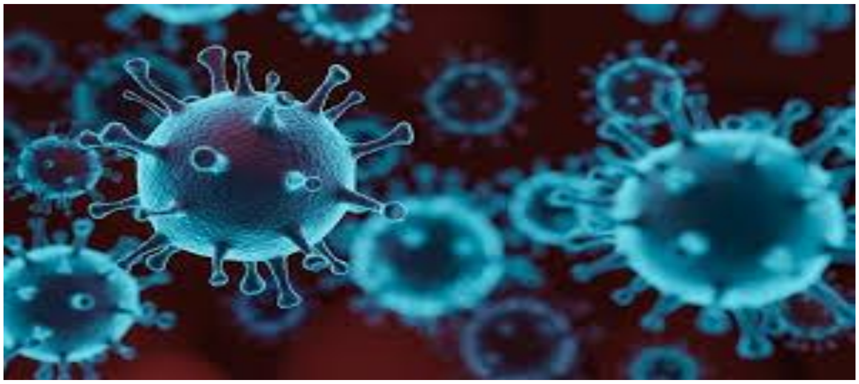
ACKNOWLEDGEMENT

My sincere gratitude and thanks towards my project paper guide Bipul Shahi, Corporate Trainer, Developer, Traveller!!! IOT, Artificial Intelligence, Robotics, Cloud Computing, Android Apps!!!

It was only with his backing and support that I could complete the report. He provided me all sorts of help and corrected me if ever seemed to make mistakes. I have no such words to express my gratitude. I acknowledge my sincere gratitude to the HOD of Computer Science Department and Training & Placement Department of RVSCET,Jamshedpur. He gave me the permission to do the project work. Without his support I couldn’t even start the work. So I am grateful to him. I acknowledge my sincere gratitude to the lecturers, research scholars and the lab technicians for their valuable guidance and helping attitude even in their very busy schedule. And at last but not the least, I acknowledge my dearest parents for being such a nice source of encouragement and moral support that helped me tremendously in this aspect. I also declare to the best of my knowledge and belief that the Project Work has not been submitted anywhere else.

INTRODUCTION

Coronaviruses (CoV) are a large family of viruses that cause illness ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS-CoV) and Severe Acute Respiratory Syndrome (SARS-CoV). A novel coronavirus (nCoV) is a new strain that has not been previously identified in humans.
Coronaviruses are zoonotic, meaning they are transmitted between animals and people. Detailed investigations found that SARS-CoV was transmitted from civet cats to humans and MERS-CoV from dromedary camels to humans. Several known coronaviruses are circulating in animals that have not yet infected humans.



Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered coronavirus. Most people infected with the COVID-19 virus will experience mild to moderate respiratory illness and recover without requiring special treatment. Older people, and those with underlying medical problems like cardiovascular disease, diabetes, chronic respiratory disease, and cancer are more likely to develop serious illness.

The best way to prevent and slow down transmission is be well informed about the COVID-19 virus, the disease it causes and how it spreads. Protect yourself and others from infection by washing your hands or using an alcohol based rub frequently and not touching your face.

The COVID-19 virus spreads primarily through droplets of saliva or discharge from the nose when an infected person coughs or sneezes, so it’s important that you also practice respiratory etiquette (for example, by coughing into a flexed elbow).

At this time, there are no specific vaccines or treatments for COVID-19. However, there are many ongoing clinical trials evaluating potential treatments.

Problem statement

The Cataclysmic outbreak of Severe Acute Respiratory Syndrom- Coronavirus (SARS-CoV-2) also known as COVID-2019 has brought the worldwide threat to the living society. The whole world is putting incredible efforts to fight against the spread of this deadly disease in terms of infrastruc- ture, finance, data sources, protective gears, life-risk treatments and several other resources. The artificial intelligence researchers are focusing their exper- tise knowledge to develop mathematical models for analyzing this epidemic situation using nationwide shared data. To contribute towards the well-being of living society, this article proposes to utilize the machine learning models with the aim for understanding its everyday exponential be- haviour along with the prediction of future reachability of the COVID-2019 across the nations by utilizing the real-time information from the Johns Hop- kins dashboard.

Technology and Concepts

Machine Learining

Machine learning (**ML**) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.

Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Types of learning algorithms

- *Supervised learning*
- *Unsupervised learning*
- *Semi-supervised learning*
- *Reinforcement learning*
- *Self learning*
- *Feature learning*
- *Sparse dictionary learning*
- *Anomaly detection*
- *Robot learning*
- *Association rules*

Models

- Linear Regression
- Polynomial Regression
- Mean Squared Error
- Support Vector Machine
- Fb Prophet

Linear Regression

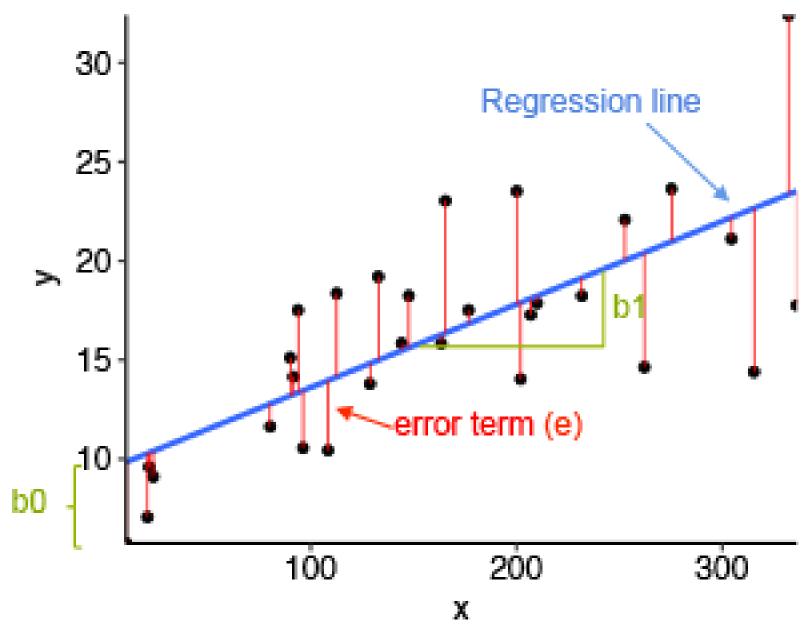
The following are a set of methods intended for regression in which the target value is expected to be a linear combination of the features. In mathematical notation, if $\hat{y}$ is the predicted value.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

Across the module, we designate the vector $w = (w_1, \ldots, w_p)$ as coef_ and w_0 as intercept_.

LinearRegression fits a linear model with coefficients $w = (w_1, \ldots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} ||Xw - y||_2^2$$



Polynomial Regression

One common pattern within machine learning is to use linear models trained on nonlinear functions of the data. This approach maintains the generally fast performance of linear methods, while allowing them to fit a much wider range of data.

For example, a simple linear regression can be extended by constructing **polynomial features** from the coefficients. In the standard linear regression case, you might have a model that looks like this for two-dimensional data:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2$$

If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

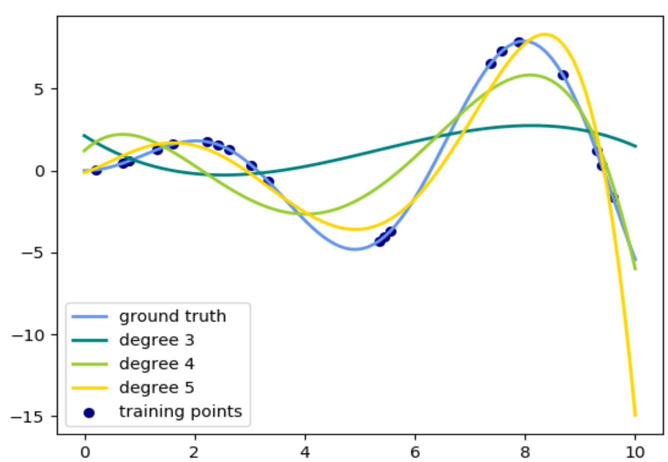
The (sometimes surprising) observation is that this is *still a linear model*: to see this, imagine creating a new set of features

$$z = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]$$

With this re-labeling of the data, our problem can be written

$$\hat{y}(w, z) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$

We see that the resulting *polynomial regression* is in the same class of linear models we considered above (i.e. the model is linear in w) and can be solved by the same techniques. By considering linear fits within a higher-dimensional space built with these basis functions, the model has the flexibility to fit a much broader range of data.



Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

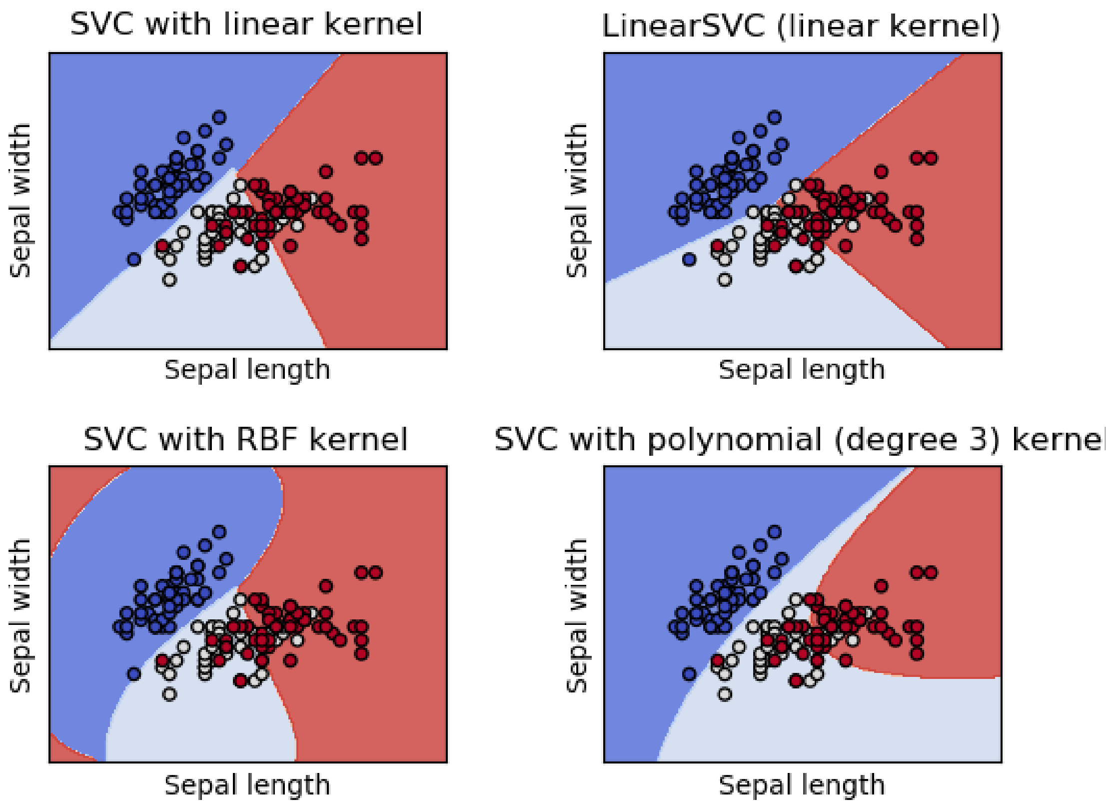
The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

SVC, **NuSVC** and **LinearSVC** are classes capable of performing binary and multi-class classification on a dataset.

SVC and **NuSVC** are similar methods, but accept slightly different sets of parameters and have different mathematical formulations (see section Mathematical formulation). On the other hand, **LinearSVC** is another (faster) implementation of Support Vector Classification for the case of a linear kernel. Note that **LinearSVC** does not accept parameter `kernel`, as this is assumed to be linear. It also lacks some of the attributes of **SVC** and **NuSVC**, like `support`



Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by sign $(w^T \phi(x) + b)$ is correct for most samples.

SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to} \quad y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, \ldots, n$$

Intuitively, we're trying to maximize the margin (by minimizing $||w||^2 = w^T w$, while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i(w^T \phi(x_i) + b)$ would be ≥1 for all samples, which indicates a perfect prediction. But problems are usually not always perfectly separable with a hyperplane, so we allow some samples to be at a distance ζ_i from their correct margin boundary. The penalty term `C` controls the strength of this penalty, and as a result, acts as an inverse regularization parameter (see note below).

The dual problem to the primal is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$
$$\text{subject to} \quad y^T \alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1, \ldots, n$$

where e is the vector of all ones, and Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. The terms αi are called the dual coefficients, and they are upper-bounded by C. This dual representation highlights the fact that training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ: see kernel trick.

Once the optimization problem is solved, the output of decision_function for a given sample x becomes:

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b,$$

and the predicted class correspond to its sign. We only need to sum over the support vectors (i.e. the samples that lie within the margin) because the dual coefficients αi are zero for the other samples.

These parameters can be accessed through the attributes `dual_coef_` which holds the product $y_i \alpha_i$, `support_vectors_` which holds the support vectors, and `intercept_` which holds the independent term b.

Note

While SVM models derived from libsvm and liblinear use C as regularization parameter, most other estimators use alpha. The exact equivalence between the amount of regularization of two models depends on the exact objective function optimized by the model. For example, when the estimator used is sklearn.linear_model.Ridge regression, the relation between them is given as

$$C = \frac{1}{alpha}$$

Fb Prophet

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.
Prophet is open source software released by Facebook's Core Data Science team. It is available for download on CRAN and PyPI.

- Prophet is used in many applications across Facebook for producing reliable forecasts for planning and goal setting. We've found it to perform better than any other approach in the majority of cases. We fit models in Stan so that you get forecasts in just a few seconds.
- Get a reasonable forecast on messy data with no manual effort. Prophet is robust to outliers, missing data, and dramatic changes in your time series.
- The Prophet procedure includes many possibilities for users to tweak and adjust forecasts. You can use human-interpretable parameters to improve your forecast by adding your domain knowledge.
- They've implemented the Prophet procedure in R and Python, but they share the same underlying Stan code for fitting. Use whatever language you're comfortable with to get forecasts.

Prophet follows the sklearn model API. We create an instance of the Prophet class and then call its fit and predict methods.

The input to Prophet is always a dataframe with two columns: ds and y. The ds (datestamp) column should be of a format expected by Pandas, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp. The y column must be numeric, and represents the measurement we wish to forecast.

Mean Squared Error

The **mean_squared_error** function computes mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error or loss.

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over n_{samples} is defined as

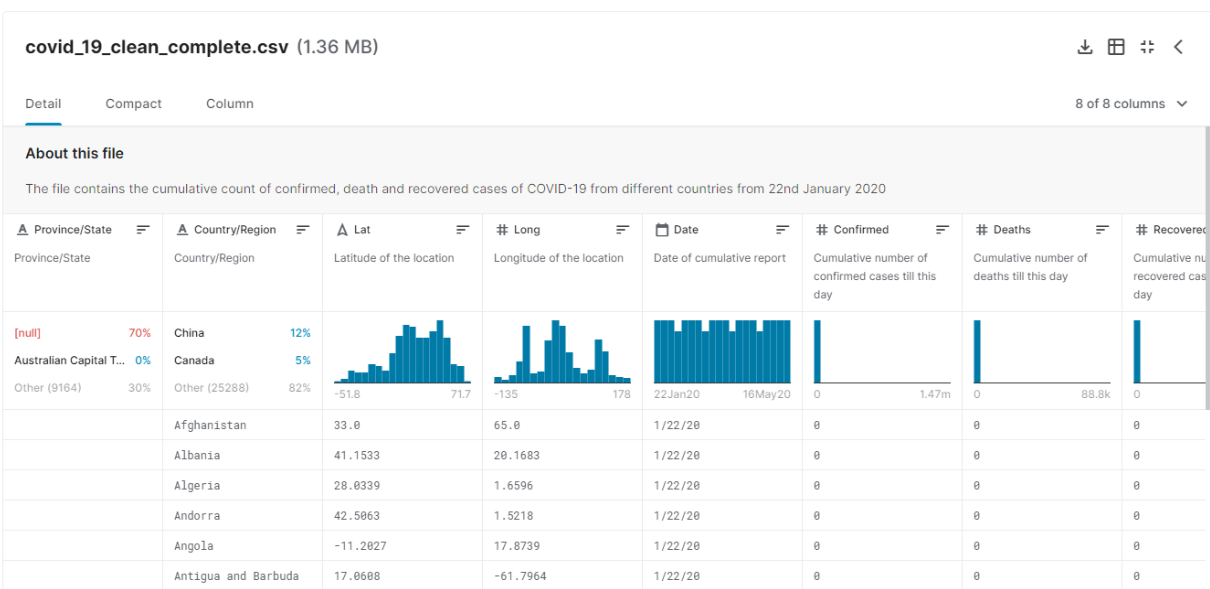
$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Dataset Description

The day to day prevalence data of COVID-2019 from January 22, 2020, to April 19, 2020, were retrieved from the official repository of Johns Hopkins University. The dataset consists of daily case reports and daily time series summary tables. In the present study, we have taken time-series summary tables in CSV format having three tables for confirmed, death and recovered cases of COVID-2019 with six attributes i.e. province/state, country/region, last update, confirmed, death and recovered cases, where the update frequency of the dataset is once in a day .

Figure presents the COVID-19 confirmed,

recovered, and death cases distribution across the world since the time data was recorded. It is easy to observe the exponential growth of the spread which needs to be controlled.



Importing required Python Packages and Libraries

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objects as go
import datetime as dt
from datetime import timedelta
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from fbprophet import Prophet
```

In [2]:
```
df=pd.read_csv("H:\IIT COGNIGENCE\data set\covid_19_clean_complete.csv")
```

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

Out[3]:

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	Recovered
0	NaN	Afghanistan	33.0000	65.0000	1/22/20	0	0	0
1	NaN	Albania	41.1533	20.1683	1/22/20	0	0	0
2	NaN	Algeria	28.0339	1.6596	1/22/20	0	0	0
3	NaN	Andorra	42.5063	1.5218	1/22/20	0	0	0
4	NaN	Angola	-11.2027	17.8739	1/22/20	0	0	0

Data Information,Changing Dtype object into date format ,Renaming columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30740 entries, 0 to 30739
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Province/State  9280 non-null   object
 1   Country/Region  30740 non-null  object
 2   Lat             30740 non-null  float64
 3   Long            30740 non-null  float64
 4   Date            30740 non-null  object
 5   Confirmed       30740 non-null  int64
 6   Deaths          30740 non-null  int64
 7   Recovered       30740 non-null  int64
dtypes: float64(2), int64(3), object(3)
memory usage: 1.9+ MB
```

In [5]:
```
df=pd.read_csv("H:\IIT COGNIGENCE\data set\covid_19_clean_complete.csv",parse_dates=['Date'])
```

In [6]:
```
df.rename(columns={"Country/Region":"country",'Province/State':'state'},inplace=True)
```

In [7]:
```
df.head()
```

Out[7]:

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered
0	NaN	Afghanistan	33.0000	65.0000	2020-01-22	0	0	0
1	NaN	Albania	41.1533	20.1683	2020-01-22	0	0	0
2	NaN	Algeria	28.0339	1.6596	2020-01-22	0	0	0
3	NaN	Andorra	42.5063	1.5218	2020-01-22	0	0	0
4	NaN	Angola	-11.2027	17.8739	2020-01-22	0	0	0

Calculation of active cases

```
In [8]: df['active'] = df['Confirmed'] - df['Deaths'] - df['Recovered']
        df.head(10)
```

Out[8]:

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	active
0	NaN	Afghanistan	33.0000	65.0000	2020-01-22	0	0	0	0
1	NaN	Albania	41.1533	20.1683	2020-01-22	0	0	0	0
2	NaN	Algeria	28.0339	1.6596	2020-01-22	0	0	0	0
3	NaN	Andorra	42.5063	1.5218	2020-01-22	0	0	0	0
4	NaN	Angola	-11.2027	17.8739	2020-01-22	0	0	0	0
5	NaN	Antigua and Barbuda	17.0608	-61.7964	2020-01-22	0	0	0	0
6	NaN	Argentina	-38.4161	-63.6167	2020-01-22	0	0	0	0
7	NaN	Armenia	40.0691	45.0382	2020-01-22	0	0	0	0
8	Australian Capital Territory	Australia	-35.4735	149.0124	2020-01-22	0	0	0	0
9	New South Wales	Australia	-33.8688	151.2093	2020-01-22	0	0	0	0

Global Status of Confirmed,Deaths,Active ,Recovered Cases.

```
In [9]: df[['Confirmed','Deaths','active','Recovered']].sum(axis=0)

Out[9]: Confirmed    137245635
        Deaths         9141572
        active        87799736
        Recovered     40304327
        dtype: int64
```

top countries

```
In [10]: top = df[df['Date'] == df['Date'].max()]

In [11]: top.head()
```

Out[11]:

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	active
30475	NaN	Afghanistan	33.0000	65.0000	2020-05-16	6402	168	745	5489
30476	NaN	Albania	41.1533	20.1683	2020-05-16	933	31	714	188
30477	NaN	Algeria	28.0339	1.6596	2020-05-16	6821	542	3409	2870
30478	NaN	Andorra	42.5063	1.5218	2020-05-16	761	51	615	95
30479	NaN	Angola	-11.2027	17.8739	2020-05-16	48	2	17	29

```
In [12]: world=top.groupby('country')['Confirmed','Deaths','active'].sum().reset_index()
         world.head()
```

Out[12]:

	country	Confirmed	Deaths	active
0	Afghanistan	6402	168	5489
1	Albania	933	31	188
2	Algeria	6821	542	2870
3	Andorra	761	51	95
4	Angola	48	2	29

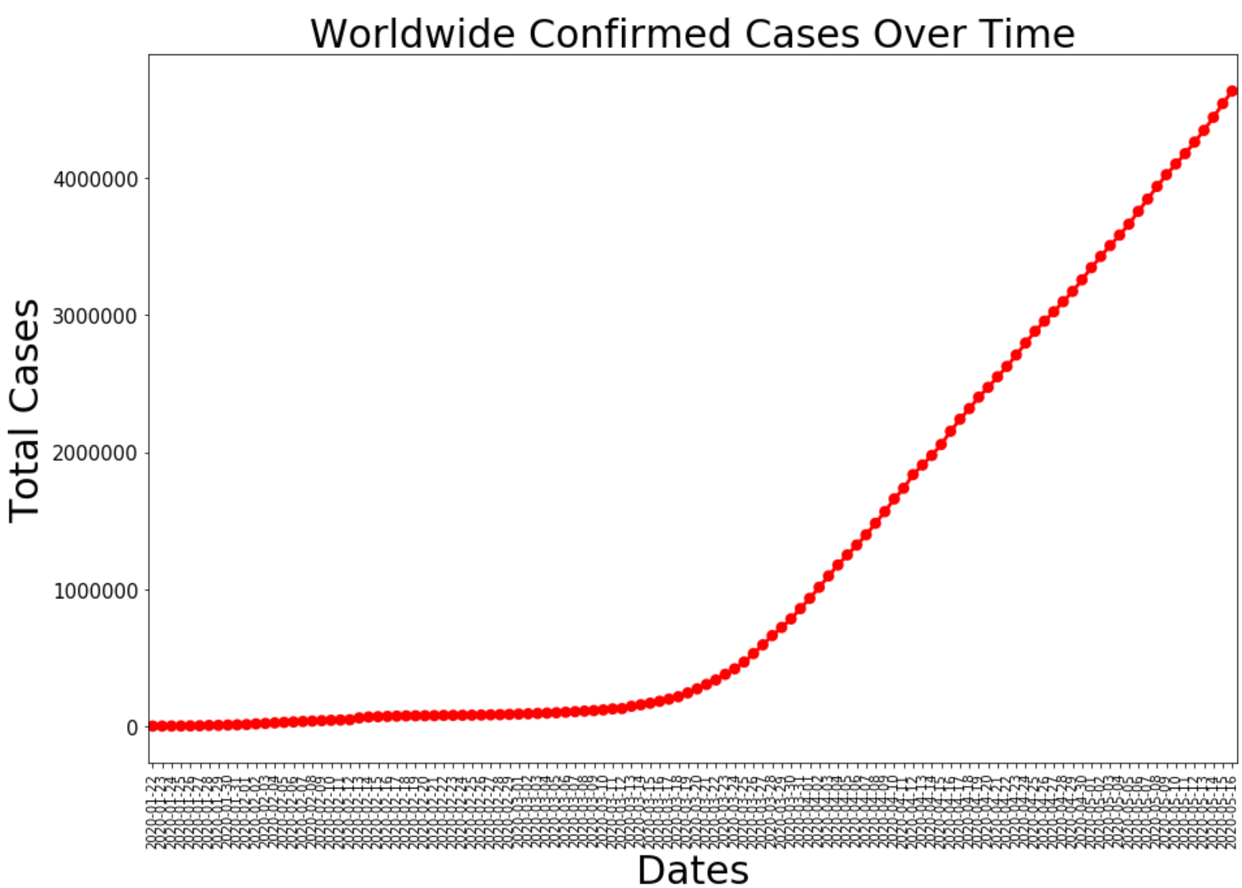
Plotting On world map

```
In [13]: figure = px.choropleth(world,locations="country",locationmode='country names'
                      ,color="active",hover_name="country",range_color=[1,1000],
                     color_continuous_scale="Peach",title="Country with Active Cases")
         figure.show()
```

World Confirmed Cases Over time

In [14]:
```
plt.figure(figsize=(15,10))
plt.xticks(rotation=90,fontsize=10)
plt.yticks(fontsize=15)
plt.xlabel("Dates",fontsize=30)
plt.ylabel("Total Cases",fontsize=30)
plt.title("Worldwide Confirmed Cases Over Time",fontsize=30)
total_cases = df.groupby('Date')['Date','Confirmed'].sum().reset_index()
total_cases['Date'] = pd.to_datetime(total_cases['Date'])
a = sns.pointplot(x = total_cases.Date.dt.date,y=total_cases.Confirmed,color='r')
a.set(xlabel="Dates",ylabel="Total Cases")
```

Out[14]:
```
[Text(0, 0.5, 'Total Cases'), Text(0.5, 0, 'Dates')]
```



Prediction and Forecasting

prediction for Confirmed Cases

In [15]:
```
datewise=df.groupby(["Date"]).agg({"Confirmed":'sum',"Recovered":'sum',"Deaths":'sum'})
datewise["Days Since"]=datewise.index-datewise.index.min()
```

Linear Regression

In [16]:
```
datewise["Days Since"]=datewise.index-datewise.index[0]
datewise["Days Since"]=datewise["Days Since"].dt.days
```

In [17]:
```
train=datewise.iloc[:int(datewise.shape[0]*0.95)]
test=datewise.iloc[int(datewise.shape[0]*0.95):]
model_scores=[]
```

In [18]:
```
lin_reg=LinearRegression(normalize=True)
```

In [19]:
```
lin_reg.fit(np.array(train["Days Since"]).reshape(-1,1),np.array(train["Confirmed"]).reshape(-1,1))
```

Out[19]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

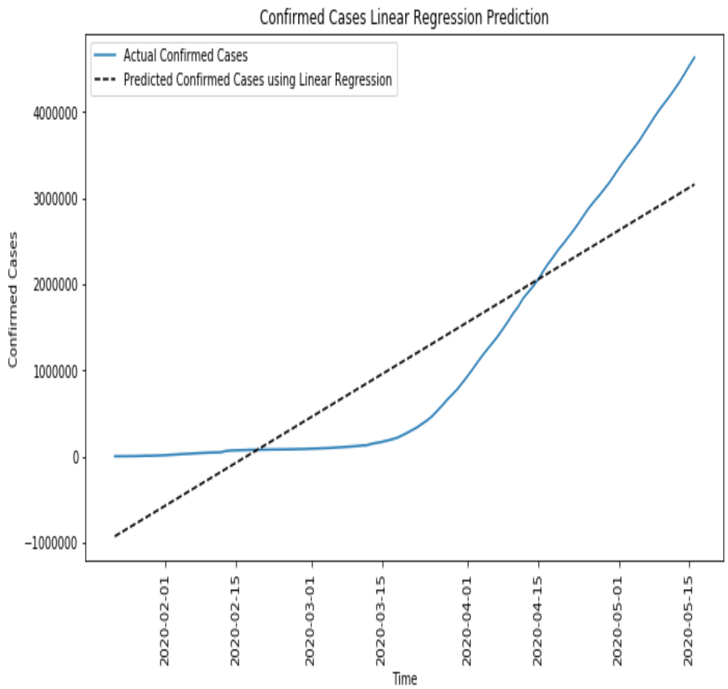
In [20]:
```
prediction_valid_linreg=lin_reg.predict(np.array(test["Days Since"]).reshape(-1,1))
```

In [21]:
```
model_scores.append(np.sqrt(mean_squared_error(test["Confirmed"],prediction_valid_linreg)))
print("Root Mean Square Error for Linear Regression: ",np.sqrt(mean_squared_error(test["Confirmed"],prediction_valid_linreg)))
```

```
Root Mean Square Error for Linear Regression:  1333986.3921440754
```

In [22]:
```
plt.figure(figsize=(11,6))
prediction_linreg=lin_reg.predict(np.array(datewise["Days Since"]).reshape(-1,1))
plt.plot(datewise["Confirmed"],label="Actual Confirmed Cases")
plt.plot(datewise.index,prediction_linreg, linestyle='--',
         label="Predicted Confirmed Cases using Linear Regression",color='black')
plt.xlabel('Time')
plt.ylabel('Confirmed Cases')
plt.title("Confirmed Cases Linear Regression Prediction")
plt.xticks(rotation=90)
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x20468058888>



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

Polynomial Regression

In [23]:
```
train=datewise.iloc[:int(datewise.shape[0]*0.95)]
test=datewise.iloc[int(datewise.shape[0]*0.95):]
```

In [24]:
```
poly = PolynomialFeatures(degree = 4)
```

In [25]:
```
train_poly=poly.fit_transform(np.array(train["Days Since"]).reshape(-1,1))
test_poly=poly.fit_transform(np.array(test["Days Since"]).reshape(-1,1))
y=train["Confirmed"]
```

In [26]:
```
linreg=LinearRegression(normalize=True)
linreg.fit(train_poly,y)
```

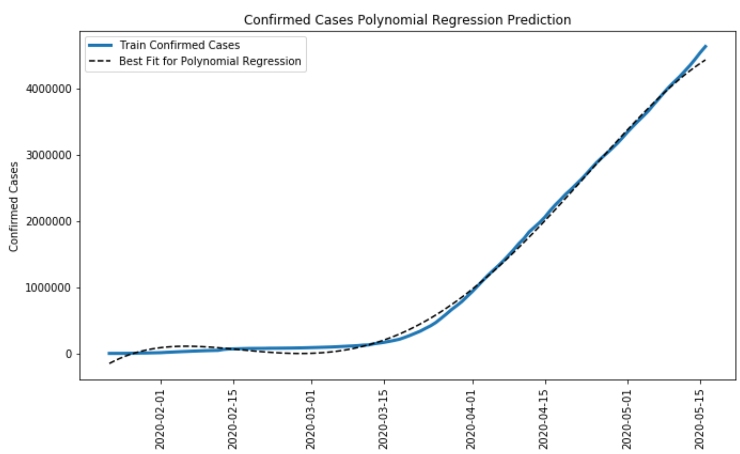
Out[26]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

In [27]:
```
prediction_poly=linreg.predict(test_poly)
rmse_poly=np.sqrt(mean_squared_error(test["Confirmed"],prediction_poly))
model_scores.append(rmse_poly)
print("Root Mean Squared Error for Polynomial Regression: ",rmse_poly)
```

```
Root Mean Squared Error for Polynomial Regression:  123695.82245564848
```

In [28]:
```
comp_data=poly.fit_transform(np.array(datewise["Days Since"]).reshape(-1,1))
plt.figure(figsize=(11,6))
predictions_poly=linreg.predict(comp_data)
plt.plot(datewise["Confirmed"],label="Train Confirmed Cases",linewidth=3)
plt.plot(datewise.index,predictions_poly, linestyle='--',label="Best Fit for Polynomial Regression",color='black')
plt.xlabel('Time')
plt.ylabel('Confirmed Cases')
plt.title("Confirmed Cases Polynomial Regression Prediction")
plt.xticks(rotation=90)
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x20468009c88>
```



In [29]:
```
new_prediction_poly=[]
for i in range(1,18):
    new_date_poly=poly.fit_transform(np.array(datewise["Days Since"].max()+i).reshape(-1,1))
    new_prediction_poly.append(linreg.predict(new_date_poly)[0])
```

Support Vector Machine

In [30]:
```
train=datewise.iloc[:int(datewise.shape[0]*0.95)]
test=datewise.iloc[int(datewise.shape[0]*0.95):]
```

In [31]:
```
#Intializing SVR Model
svm=SVR(C=1,degree=5,kernel='poly',epsilon=0.01)
```

In [32]:
```
#Fitting model on the training data
svm.fit(np.array(train["Days Since"]).reshape(-1,1),np.array(train["Confirmed"]).reshape(-1,1))
```

Out[32]:
```
SVR(C=1, cache_size=200, coef0=0.0, degree=5, epsilon=0.01, gamma='scale',
    kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

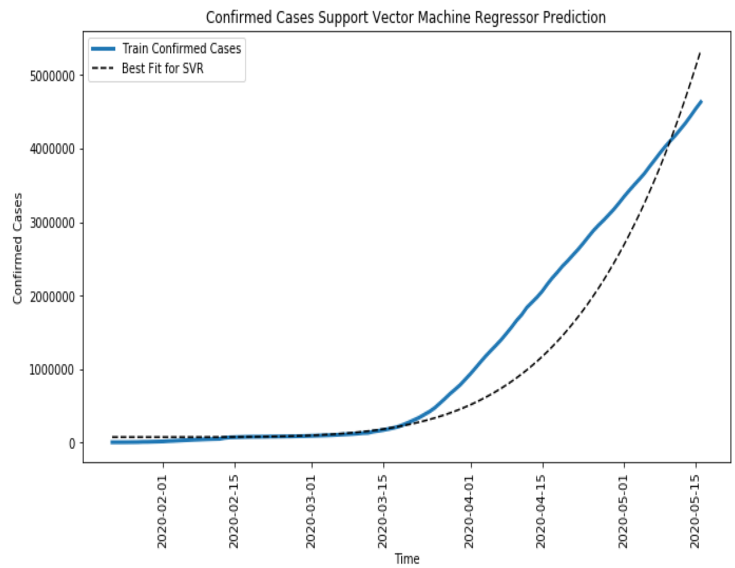
In [33]:
```
prediction_valid_svm=svm.predict(np.array(test["Days Since"]).reshape(-1,1))
```

In [34]:
```
model_scores.append(np.sqrt(mean_squared_error(test["Confirmed"],prediction_valid_svm)))
print("Root Mean Square Error for S V M: ",np.sqrt(mean_squared_error(test["Confirmed"],prediction_valid_svm)))
```

```
Root Mean Square Error for S V M:  449529.86262025463
```

In [35]:
```
plt.figure(figsize=(11,6))
prediction_svm=svm.predict(np.array(datewise["Days Since"]).reshape(-1,1))
plt.plot(datewise["Confirmed"],label="Train Confirmed Cases",linewidth=3)
plt.plot(datewise.index,prediction_svm, linestyle='--',label="Best Fit for SVR",color='black')
plt.xlabel('Time')
plt.ylabel('Confirmed Cases')
plt.title("Confirmed Cases Support Vector Machine Regressor Prediction")
plt.xticks(rotation=90)
plt.legend()
```

Out[35]: <matplotlib.legend.Legend at 0x2046884f608>



In [36]:
```
new_date=[]
new_prediction_lr=[]
new_prediction_svm=[]
for i in range(1,18):
    new_date.append(datewise.index[-1]+timedelta(days=i))
    new_prediction_lr.append(lin_reg.predict(np.array(datewise["Days Since"].max()+i).reshape(-1,1))[0][0])
    new_prediction_svm.append(svm.predict(np.array(datewise["Days Since"].max()+i).reshape(-1,1))[0])
```

In [37]:
```
pd.set_option('display.float_format', lambda x: '%.6f' % x)
model_predictions=pd.DataFrame(zip(new_date,new_prediction_lr,new_prediction_poly,new_prediction_svm),
                               columns=["Dates","Linear Regression Prediction","Polynonmial Regression Prediction","SVM Prediction"])
model_predictions.head()
```

Out[37]:

	Dates	Linear Regression Prediction	Polynonmial Regression Prediction	SVM Prediction
0	2020-05-17	3194781.610969	4479756.919335	5573183.835152
1	2020-05-18	3230344.826225	4523773.522137	5814468.184793
2	2020-05-19	3265908.041481	4563680.939991	6064144.098063
3	2020-05-20	3301471.256736	4599262.970784	6322428.582237
4	2020-05-21	3337034.471992	4630299.716464	6589542.353985

Prophet

In [38]:
```
confirmed = df.groupby('Date').sum()['Confirmed'].reset_index()
confirmed.head()
```

Out[38]:

	Date	Confirmed
0	2020-01-22	555
1	2020-01-23	654
2	2020-01-24	941
3	2020-01-25	1434
4	2020-01-26	2118

In [39]:
```
deaths = df.groupby('Date').sum()['Deaths'].reset_index()
deaths.head()
```

Out[39]:

	Date	Deaths
0	2020-01-22	17
1	2020-01-23	18
2	2020-01-24	26
3	2020-01-25	42
4	2020-01-26	56

In [40]:
```
recovered = df.groupby('Date').sum()['Recovered'].reset_index()
recovered.head()
```

Out[40]:

	Date	Recovered
0	2020-01-22	28
1	2020-01-23	30
2	2020-01-24	36
3	2020-01-25	39
4	2020-01-26	52

In [41]:
```
confirmed.columns = ['ds','y']
confirmed.head()
```

Out[41]:

	ds	y
0	2020-01-22	555
1	2020-01-23	654
2	2020-01-24	941
3	2020-01-25	1434
4	2020-01-26	2118

In [42]:
```
confirmed['ds'] = pd.to_datetime(confirmed['ds'])
```

In [43]:
```
m = Prophet(interval_width=0.95)
m.fit(confirmed)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[43]:

	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22
122	2020-05-23

In [45]:
```
forecast.head()
```

Out[45]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	wee
0	2020-01-22	-2910.691316	-22868.349046	7686.952084	-2910.691316	-2910.691316	-3869.984421	-3869.984421	-3869.984421	-3869.984421	-38
1	2020-01-23	-1422.502830	-17337.232889	12483.991097	-1422.502830	-1422.502830	-603.894825	-603.894825	-603.894825	-603.894825	-6
2	2020-01-24	65.685656	-10579.654702	19135.881392	65.685656	65.685656	4288.429385	4288.429385	4288.429385	4288.429385	42
3	2020-01-25	1553.874141	-9175.944994	20588.197956	1553.874141	1553.874141	4550.750614	4550.750614	4550.750614	4550.750614	45
4	2020-01-26	3042.062627	-10994.455477	19371.617403	3042.062627	3042.062627	1375.332587	1375.332587	1375.332587	1375.332587	13

In [46]:
```
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

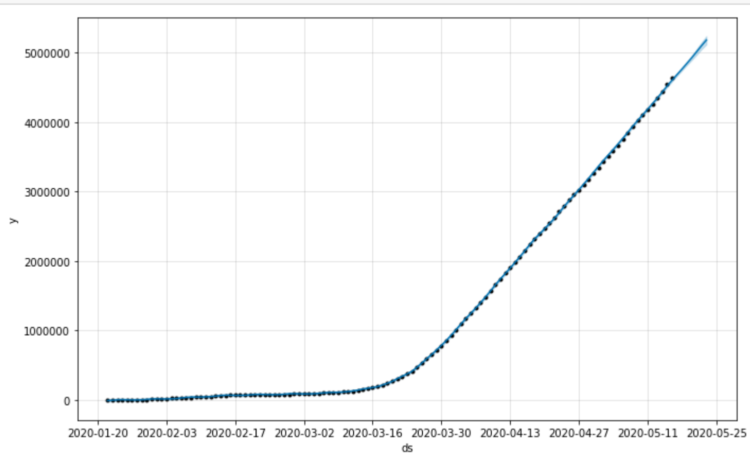
Out[46]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	4840514.888703	4817266.310877	4862025.878048
119	2020-05-20	4922621.175648	4892967.967097	4949993.443324
120	2020-05-21	5008444.936470	4967656.636708	5049301.554402
121	2020-05-22	5095894.931905	5048388.274155	5144009.144801
122	2020-05-23	5178714.924359	5117343.464303	5235847.870527

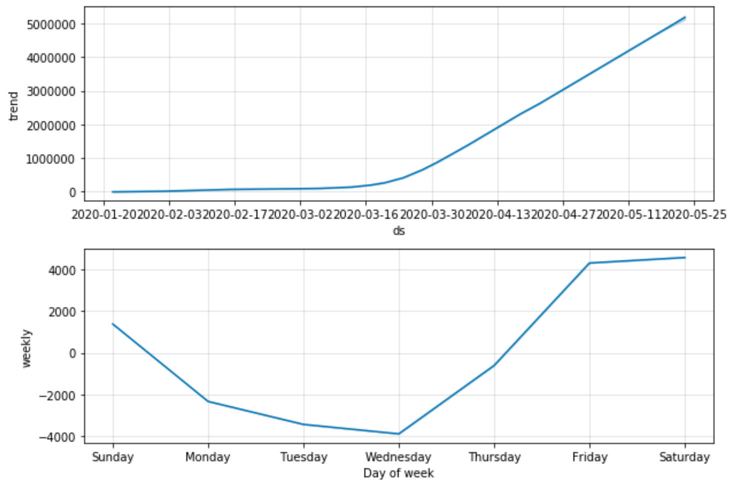
In [47]:
```
model_scores.append(np.sqrt(mean_squared_error(datewise["Confirmed"],forecast['yhat'].head(datewise.shape[0]))))
print("Root Mean Squared Error for Prophet Model: ",
      np.sqrt(mean_squared_error(datewise["Confirmed"],forecast['yhat'].head(datewise.shape[0]))))
```

```
Root Mean Squared Error for Prophet Model:  7693.31267800317
```

In [48]:
```
confirmed_forecast_plot = m.plot(forecast)
```



```
In [49]: confirmed_forecast_plot = m.plot_components(forecast)
```




```
In [50]: model_names=["Linear Regression","Polynomial Regression","Support Vector Machine Regressor",
                      "Facebook's Prophet Model"]
         model_summary=pd.DataFrame(zip(model_names,model_scores),
                                    columns=["Model Name","Root Mean Squared Error"]).sort_values(["Root Mean Squared Error"])
         model_summary
```

Out[50]:

	Model Name	Root Mean Squared Error
3	Facebook's Prophet Model	7693.312678
1	Polynomial Regression	123695.822456
2	Support Vector Machine Regressor	449529.862620
0	Linear Regression	1333986.392144

Here Facebook’s Prophet Model have less Root Mean Squared Error in comparison to others model.

```
In [51]: model_predictions["Prophet's Prediction"]=list(forecast["yhat"].tail(17))
         model_predictions["Prophet's Upper Bound"]=list(forecast["yhat_upper"].tail(17))
         model_predictions["Prophet's Lower Bound"]=list(forecast["yhat_lower"].tail(17))
         model_predictions.head()
```

Out[51]:

	Dates	Linear Regression Prediction	Polynonmial Regression Prediction	SVM Prediction	Prophet's Prediction	Prophet's Upper Bound	Prophet's Lower Bound
0	2020-05-17	3194781.610969	4479756.919335	5573183.835152	3852637.539318	3868380.797381	3837053.489917
1	2020-05-18	3230344.826225	4523773.522137	5814468.184793	3940087.534753	3956521.205543	3924020.643553
2	2020-05-19	3265908.041481	4563680.939991	6064144.098063	4022907.527207	4037126.434625	4006492.981704
3	2020-05-20	3301471.256736	4599262.970784	6322428.582237	4102289.780405	4117131.470179	4087945.428182
4	2020-05-21	3337034.471992	4630299.716464	6589542.353985	4181150.085844	4196149.400076	4166491.241160

Prediction for Deaths Cases

In [52]:
```
deaths.columns = ['ds','y']
deaths['ds'] = pd.to_datetime(deaths['ds'])
deaths.head()
```

Out[52]:

	ds	y
0	2020-01-22	17
1	2020-01-23	18
2	2020-01-24	26
3	2020-01-25	42
4	2020-01-26	56

In [53]:
```
m = Prophet(interval_width=0.95)
m.fit(deaths)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[53]:

	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22
122	2020-05-23

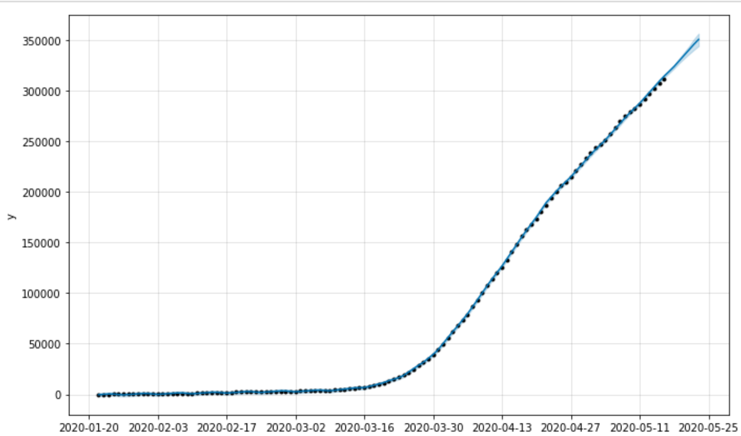
In [54]:
```
forecast = m.predict(future)
```

In [55]:
```
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

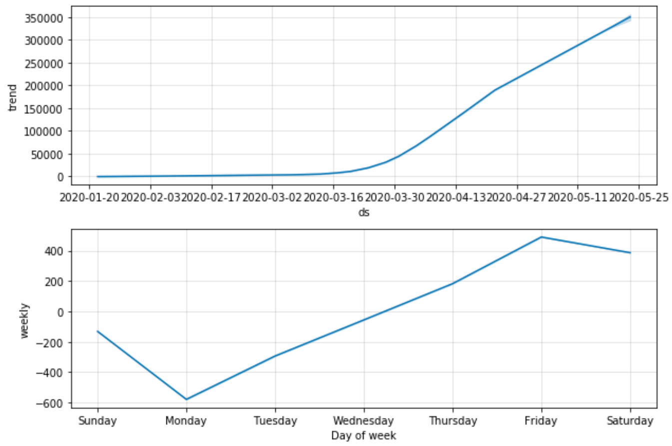
Out[55]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	329314.598658	327065.779164	331501.791098
119	2020-05-20	334741.622451	331616.559131	337759.727315
120	2020-05-21	340169.658850	336046.979252	343650.474950
121	2020-05-22	345666.206989	340281.435639	350745.383407
122	2020-05-23	350752.194802	344195.698756	356640.361195

In [56]:
```
deaths_forecast_plot = m.plot(forecast)
```



In [57]:
```
deaths_forecast_plot = m.plot_components(forecast)
```



Prediction for Recovered Cases

In [58]:
```
recovered.columns = ['ds','y']
recovered['ds'] = pd.to_datetime(recovered['ds'])
recovered.head()
```

Out[58]:

	ds	y
0	2020-01-22	28
1	2020-01-23	30
2	2020-01-24	36
3	2020-01-25	39
4	2020-01-26	52

In [59]:
```
m = Prophet(interval_width=0.95)
m.fit(recovered)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[59]:

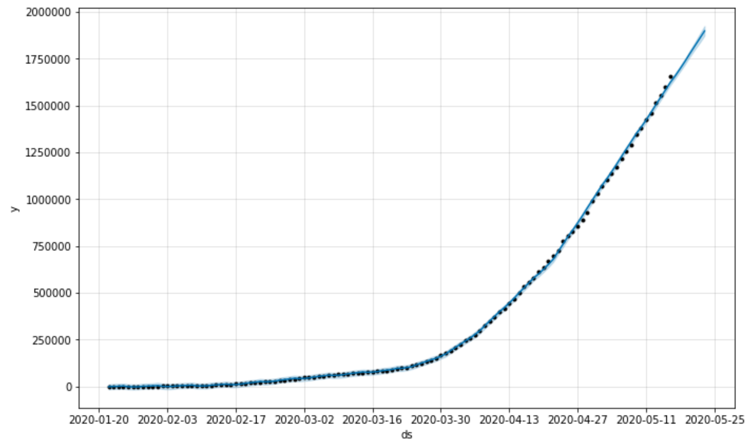
	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22

In [60]:
```
forecast = m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

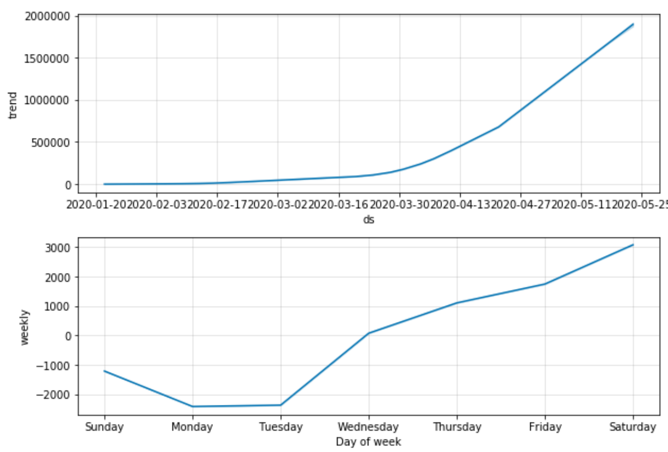
Out[60]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	1735206.918001	1719930.572457	1749760.640635
119	2020-05-20	1776907.079258	1761062.118429	1793772.968297
120	2020-05-21	1817193.527482	1798983.657315	1836510.104845
121	2020-05-22	1857090.646124	1834895.767630	1881846.862683
122	2020-05-23	1897681.083839	1874019.630879	1924625.683166

In [61]:
```
recovered_forecast_plot = m.plot(forecast)
```



In [62]:
```
recovered_forecast_plot = m.plot_components(forecast)
```



Prediction for India

In [63]:
```
df.head()
```

Out[63]:

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	active
0	NaN	Afghanistan	33.000000	65.000000	2020-01-22	0	0	0	0
1	NaN	Albania	41.153300	20.168300	2020-01-22	0	0	0	0
2	NaN	Algeria	28.033900	1.659600	2020-01-22	0	0	0	0
3	NaN	Andorra	42.506300	1.521800	2020-01-22	0	0	0	0
4	NaN	Angola	-11.202700	17.873900	2020-01-22	0	0	0	0

In [64]:
```
df_india=df.query('country=="India"')
```

In [65]:
```
df_india.head()
```

Out[65]:

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	active
131	NaN	India	21.000000	78.000000	2020-01-22	0	0	0	0
396	NaN	India	21.000000	78.000000	2020-01-23	0	0	0	0
661	NaN	India	21.000000	78.000000	2020-01-24	0	0	0	0
926	NaN	India	21.000000	78.000000	2020-01-25	0	0	0	0
1191	NaN	India	21.000000	78.000000	2020-01-26	0	0	0	0

In [66]:
```
df_india = df.query('country=="India"').groupby('Date')[['Confirmed','Deaths','Recovered']].sum().reset_index()
```

In [67]:
```
df_india.head()
```

Out[67]:

	Date	Confirmed	Deaths	Recovered
0	2020-01-22	0	0	0
1	2020-01-23	0	0	0
2	2020-01-24	0	0	0
3	2020-01-25	0	0	0
4	2020-01-26	0	0	0

In [68]:
```
india_confirmed,india_deaths,india_recovered = df_india[['Date','Confirmed']],df_india[['Date','Deaths']],df_india[['Date','Reco
vered']]
```

In [69]:
```
india_confirmed
```

Out[69]:

	Date	Confirmed
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0
...	...	...
111	2020-05-12	74292
112	2020-05-13	78055
114	2020-05-15	85784
115	2020-05-16	90648

116 rows × 2 columns

In [70]:
```
india_deaths
```

Out[70]:

	Date	Deaths
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0
...	...	...
111	2020-05-12	2415
112	2020-05-13	2551
113	2020-05-14	2649
114	2020-05-15	2753
115	2020-05-16	2871

116 rows × 2 columns

In [71]:
```
india_recovered
```

Out[71]:

	Date	Recovered
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0
...	...	...
111	2020-05-12	24420
112	2020-05-13	26400
113	2020-05-14	27969
114	2020-05-15	30258
115	2020-05-16	34224

116 rows × 2 columns

Prediction for confirmed cases in India

In [72]:
```
india_confirmed.columns = ['ds','y']
india_confirmed['ds'] = pd.to_datetime(india_confirmed['ds'])
```

In [73]:
```
india_confirmed.head()
```

Out[73]:

	ds	y
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0

In [74]:
```
m = Prophet(interval_width=0.95)
m.fit(india_confirmed)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[74]:

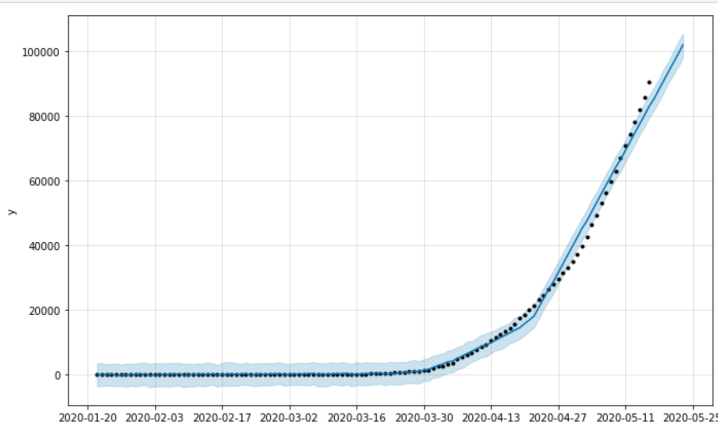
	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22

In [75]:
```
forecast = m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

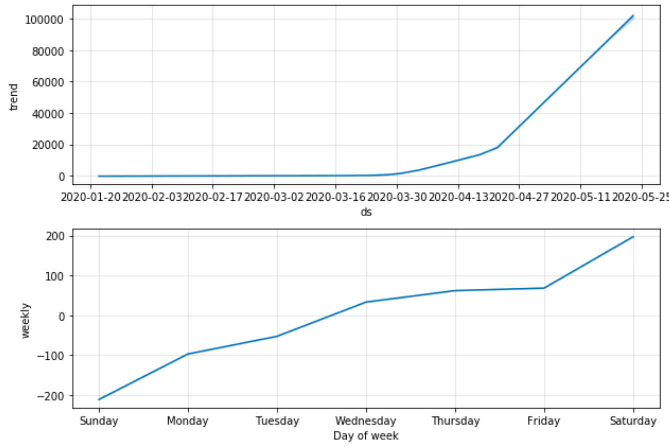
Out[75]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	90898.375311	87292.628372	94524.961765
119	2020-05-20	93685.799686	90342.033985	96920.957882
120	2020-05-21	96416.109379	92772.069928	99936.930342
121	2020-05-22	99123.977562	95207.541907	102934.858773
122	2020-05-23	101954.358178	98125.519038	105566.658492

In [76]:
```
india_confirmed_forecast_plot = m.plot(forecast)
```



In [77]:
```
india_confirmed_forecast_plot = m.plot_components(forecast)
```



Prediction for Recovered Cases

In [78]:
```
india_recovered.columns=['ds','y']
india_recovered['ds']=pd.to_datetime(india_recovered['ds'])
```

In [79]:
```
india_recovered.head()
```

Out[79]:

	ds	y
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0

In [80]:
```
m = Prophet(interval_width=0.95)
m.fit(india_recovered)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[80]:

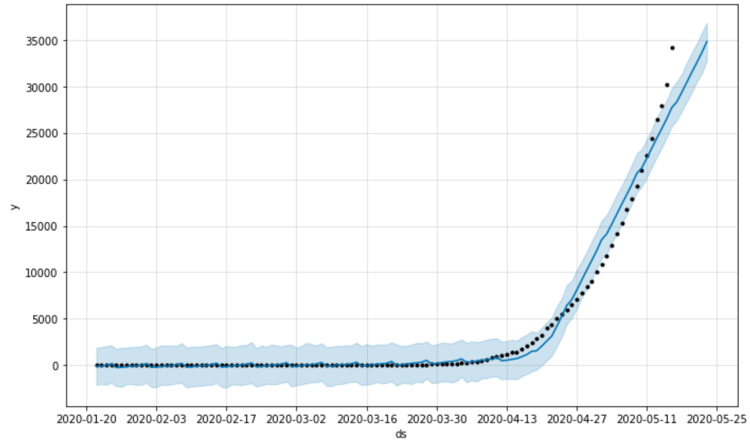
	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22
122	2020-05-23

In [81]:
```
forecast = m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

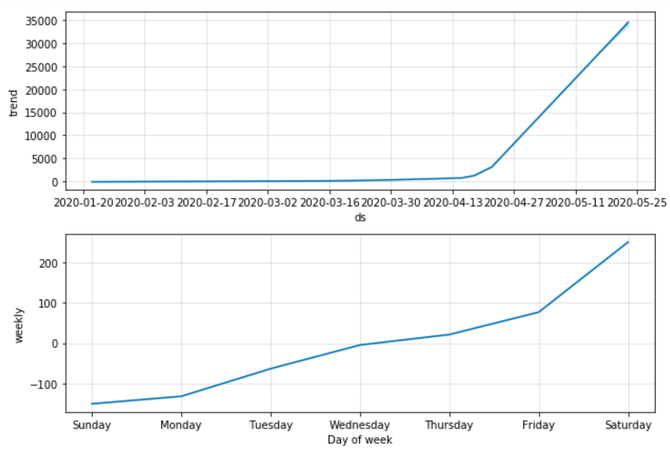
Out[81]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	30442.566417	28356.161565	32565.638288
119	2020-05-20	31515.056049	29354.464339	33638.894801
120	2020-05-21	32554.831990	30521.483073	34648.156392
121	2020-05-22	33624.187940	31395.621386	35779.798383
122	2020-05-23	34811.685019	32785.005667	36844.617987

In [82]:
```
india_recovered_forecast_plot = m.plot(forecast)
```



In [83]:
```
india_recovered_forecast_plot = m.plot_components(forecast)
```



Prediction for Deaths Cases

In [84]:
```
india_deaths.columns=['ds','y']
india_deaths['ds']=pd.to_datetime(india_deaths['ds'])
```

In [85]:
```
india_deaths.head()
```

Out[85]:

	ds	y
0	2020-01-22	0
1	2020-01-23	0
2	2020-01-24	0
3	2020-01-25	0
4	2020-01-26	0

In [86]:
```
m = Prophet(interval_width=0.95)
m.fit(india_deaths)
future = m.make_future_dataframe(periods=7)
future.tail()
```

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

Out[86]:

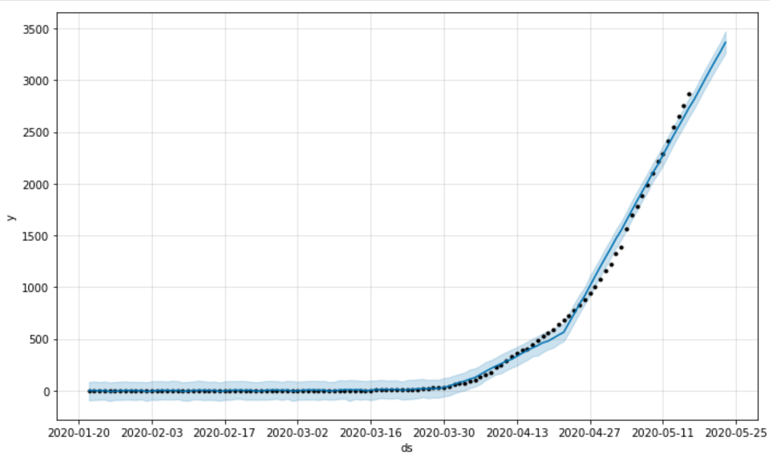
	ds
118	2020-05-19
119	2020-05-20
120	2020-05-21
121	2020-05-22
122	2020-05-23

In [87]:
```
forecast = m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

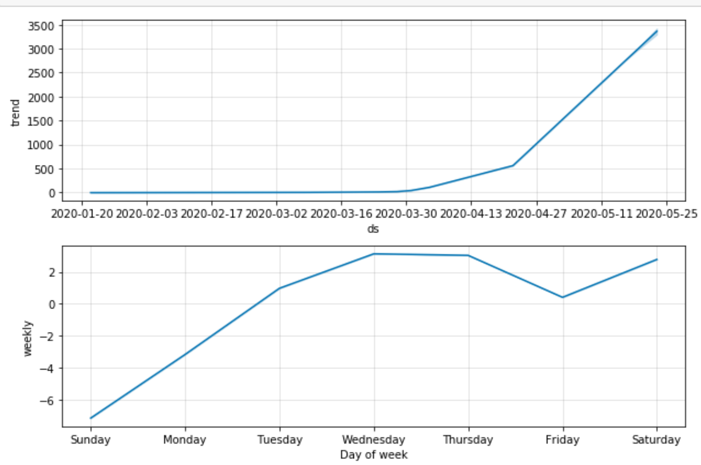
Out[87]:

	ds	yhat	yhat_lower	yhat_upper
118	2020-05-19	3002.949052	2917.305917	3097.168545
119	2020-05-20	3095.556322	3004.837448	3186.322441
120	2020-05-21	3185.908202	3095.079190	3279.418250
121	2020-05-22	3273.730762	3178.610745	3365.492340
122	2020-05-23	3366.549716	3262.974382	3474.648887

In [88]:
```
india_deaths_forecast_plot = m.plot(forecast)
# ploting predicted value
```



In [89]:
```
india_deaths_forecast_plot = m.plot_components(forecast)
```



Thankyou

Conclusions

The world is under the grasp of SARS-CoV2 (COVID-19) virus. Early prediction of the transmission can help to take necessary actions. This article proposed to utilize the machine learning and deep learning models for epidemic analysis using data from Johns Hopkins dashboard.

The results show that Facebook's Prophet Model yielded a minimum root mean square error (RMSE) score over other approaches in forecasting the COVID-19 transmission.

In a pandemic like this, providing timely information to the public is paramount.

However, if the spread follows the predicted trend of the Prophet model then it would lead to huge loss of lives as it presents the exponential growth of the transmission worldwide.

As observed the growth of the COVID-19 can be reduced and quenched by reducing the number of susceptible individuals from the infected individuals. This is achievable by becoming unsocial and following the lockdown initiative with discipline. The study can further be extended to utilize other machine learning and deep learning models.

Bibliography

- https://www.kaggle.com
- https://www.who.int/emergencies/diseases/novel-coronavirus-2019
- https://en.wikipedia.org/wiki/Coronavirus_disease_2019