COVID-19 PREDICTIONS FOR INDIA

Machine Learning (UEC713)

Maulik Gupta

Roll No. 101815105

Under the Guidance of

Dr. Neeru Jindal

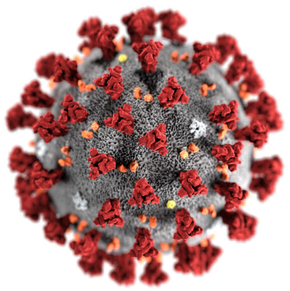
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, PATIALA

2018 – 2022

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. Following image shows an image of SARS-CoV2 as seen under a microscope.



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.

Common signs of infection include respiratory symptoms, fever, cough, shortness of breath and breathing difficulties. In more severe cases, infection can cause pneumonia, severe acute respiratory syndrome, kidney failure and even death.

Standard recommendations to prevent infection spread include regular hand washing, covering mouth and nose when coughing and sneezing, thoroughly cooking meat and eggs. Avoid close contact with anyone showing symptoms of respiratory illness such as coughing and sneezing.

Objective :-

1. Build an ML model for predicting the number of total infections in India till 31st of July 2021.
2. Build an ML model for predicting the number of deaths in India till 31st of July 2021.
3. Build an ML model and predict recovery rate and death rate in India for the period 15th June to 31st of July 2021.
4. Build an ML model and predict infection rate for which data are available in the age group 35-50 till 31st of July 2021.
5. Using information obtained from the above models and other data available on the link, predict the tentative date (or interval spanning a week) after which the infection spread (Active cases) is 10% of its peak value in India.

Methodology :-

To complete the above mentioned objectives the following approach needs to be followed

1. Fetching the dataset.
2. Preprocessing or cleaning the dataset.

3. Training multiple machine learning models.
4. Using the most accurate machine learning model to predict required future values

Dataset :-

The dataset has been taken from covid19india.org .

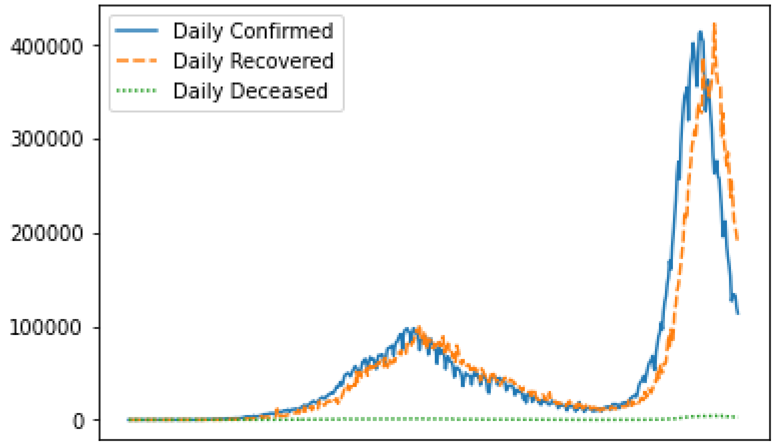
```python
# importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from sklearn.model_selection import train_test_split
from datetime import date, timedelta

from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor


# reading data
data = pd.read_csv('./case_time_series.csv')


data.tail()
```

	Date	Date_YMD	Daily Confirmed	Total Confirmed	Daily Recovered	Total Recovered	Daily Deceased	Total Deceased
488	1 June 2021	2021-06-01	133152	28306818	231397	26170916	3205	334525
489	2 June 2021	2021-06-02	134044	28440862	211890	26382806	2898	337423
490	3 June 2021	2021-06-03	132424	28573286	206722	26589528	2717	340140
491	4 June 2021	2021-06-04	120454	28693740	197763	26787291	3372	343512
492	5 June 2021	2021-06-05	113898	28807638	187991	26975282	2669	346181

```python
#taking only necessary data
sns.lineplot(x='Date_YMD', y='Daily Confirmed', data=data)
plt.xticks([])
plt.show()
```

Plotting graph for Total confirmed, recovered and deceased

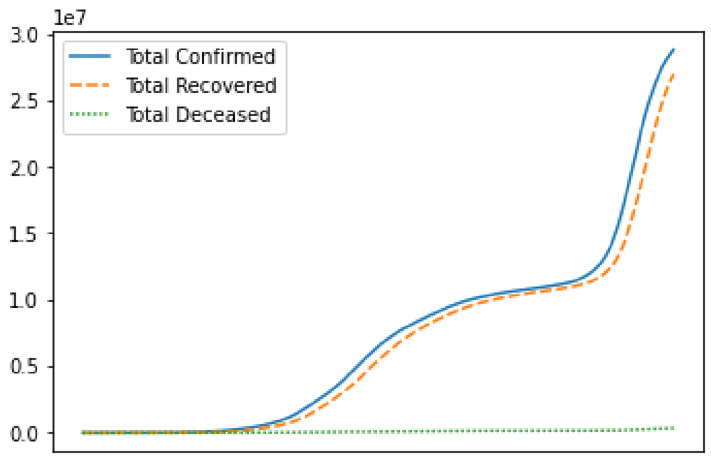
```
DailyData = data.drop(columns=['Total Confirmed', 'Total Recovered',
'Total Deceased'])
sns.lineplot(data=DailyData)
plt.xticks([])
plt.show()
```



From this graph it is clear that up until 5th June 2021, India has experienced two covid 19 waves and now the total cases are decreasing.

Plotting graph for daily confirmed, recovered and deceased

```
totalData = data.drop(columns=['Daily Confirmed', 'Daily Recovered',
'Daily Deceased'])
sns.lineplot(data=totalData)
plt.xticks([])
plt.show()
```



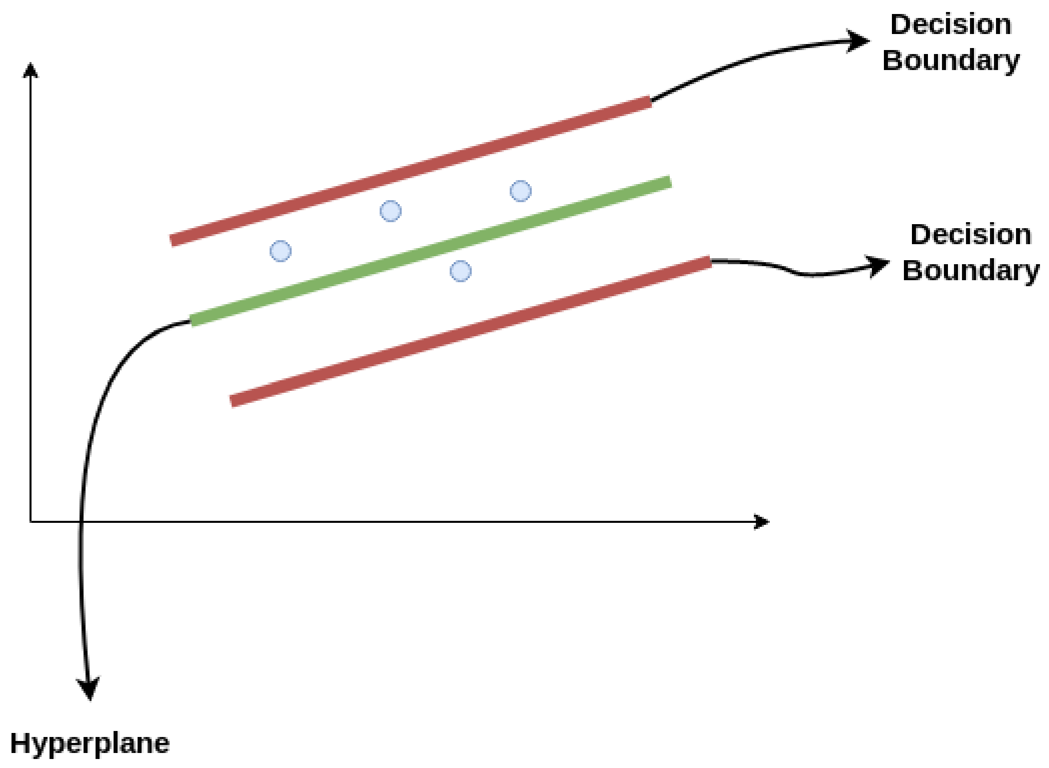
The two peaks can be observed from the daily cases as well.

Training Machine Learning Models

We will be using two models in this report:

1. Support Vector Regression
2. Random Forest Regression

1. **Support Vector Regression**

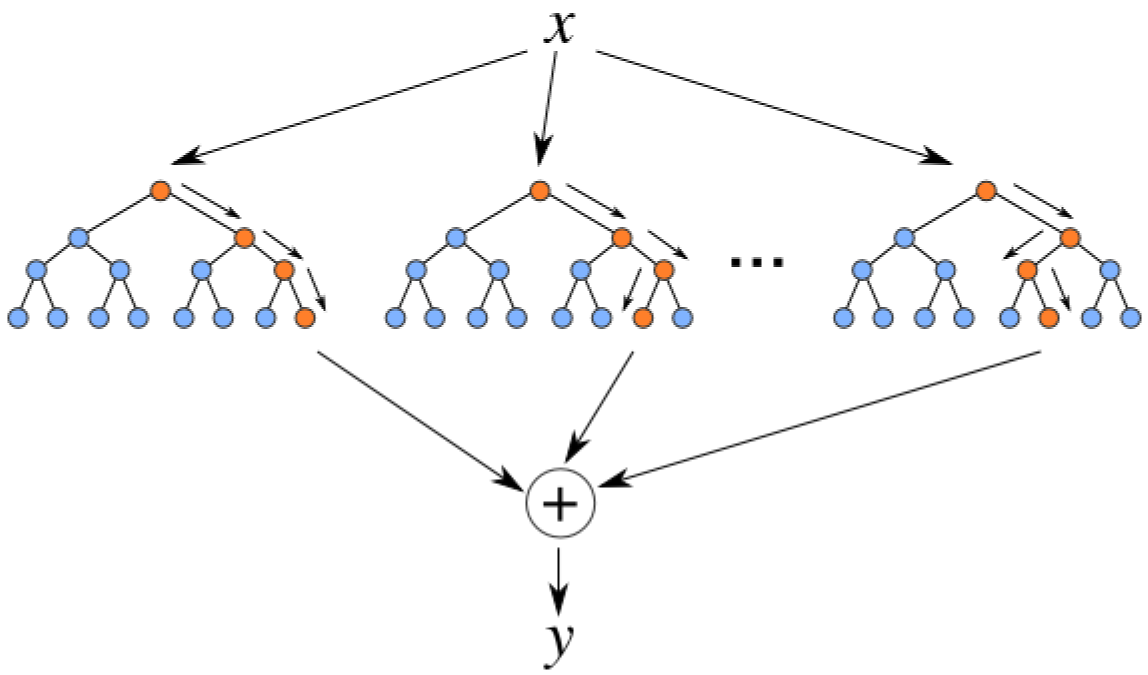


In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at AT&T Bell Laboratories by Vladimir Vapnik with colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik (1982, 1995) and Chervonenkis (1974). Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering[2] algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications

2. Random Forest Regression



Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned Random decision forests correct for decision trees' habit of overfitting to their training set.[3]:587–588 Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Housing the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

Random forests are frequently used as "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

```
# defining models
svr_rbf = SVR(kernel='rbf', epsilon=0.08)
RFRegressor =
RandomForestRegressor(max_depth=3,n_estimators=3,random_state=0)


# converting date-string to date timestamp
data['Date_YMD'] = pd.to_datetime(data["Date_YMD"])
data['Date_YMD'] = data['Date_YMD'].map(dt.datetime.toordinal)
```

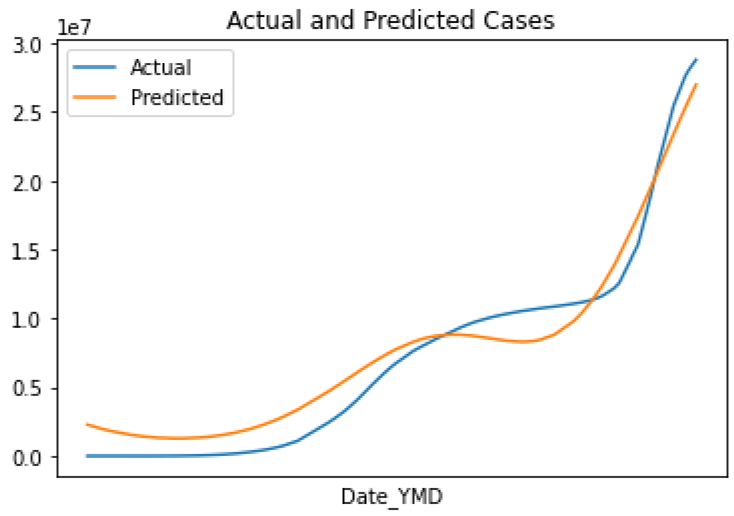
Now, we will be reading required data for number of total confirmed cases

```
# Reading required data
X = data.iloc[:, 1].values
y = data.iloc[:, 3].values
```

Applying train test split

```
X_train, X_test, y_train, y_test = train_test_split(data['Date_YMD'],
y, test_size=0.2, random_state=0)
X_train = X_train.to_numpy().reshape(-1, 1)
X_test_reshaped = X_test.to_numpy().reshape(-1, 1)


# Scaling current values down
maxConfirmedCases = max(data['Total Confirmed'])
maxDate = max(data['Date_YMD'])


# training model on normalized values
svr_rbf.fit(X_train/maxDate, y_train/maxConfirmedCases)
```

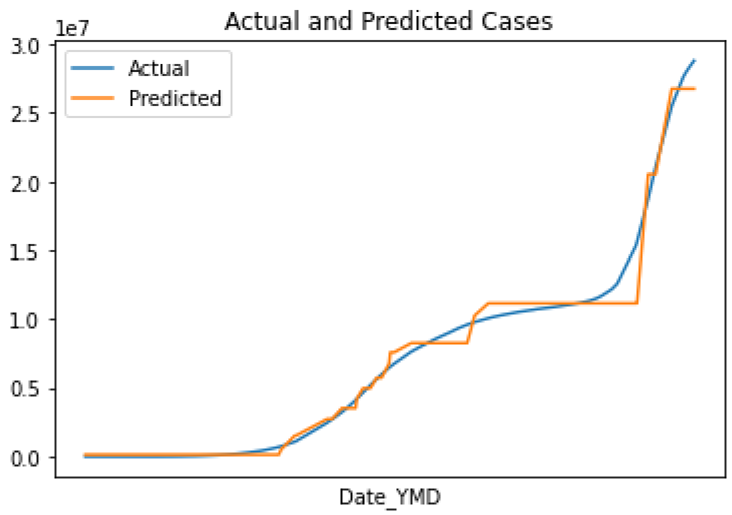
```
predicted_cases = svr_rbf.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted cases
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_cases*maxConfirmedCases)
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Cases')
plt.show()
# calculating accuracy of model
svr_rbf.score(X_test_reshaped/maxDate, y_test/maxConfirmedCases)
```



Accuracy of model : 0.9490311380188391

Using Random Forest

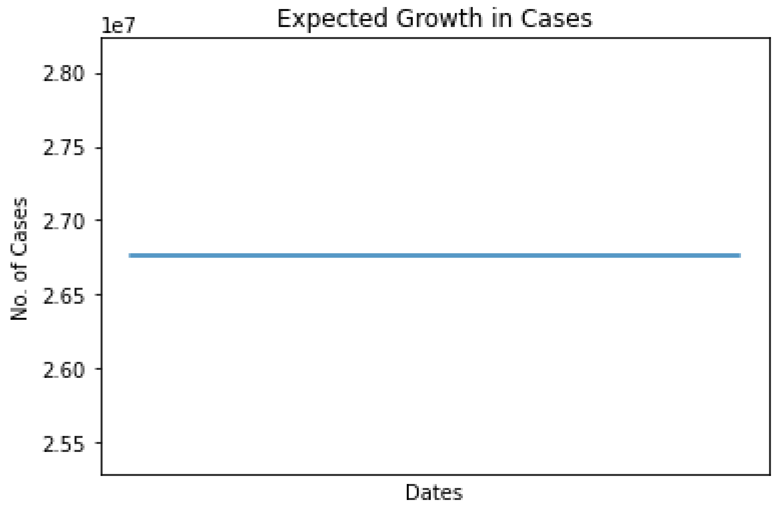
```
# training model on normalized values
RFRegressor.fit(X_train/maxDate, y_train/maxConfirmedCases)
predicted_cases = RFRegressor.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted cases
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_cases*maxConfirmedCases)
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Cases')
plt.show()
# calculating accuracy of model
RFRegressor.score(X_test_reshaped/maxDate, y_test/maxConfirmedCases)
```



Accuracy : 0.9861205507152978

Using Random Forest Regression as it is more accurate

```
# Predicting the confirmed cases in India till 31st July

# creating array of required dates
start = pd.to_datetime('2021-06-06')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_cases =
(RFRegressor.predict(datelist.to_numpy().reshape(-1, 1) /
maxDate))*maxConfirmedCases

# plotting predicted cases
sns.lineplot(x=datelist, y=predicted_cases)
plt.xlabel('Dates')
plt.ylabel('No. of Cases')
plt.title('Expected Growth in Cases')
plt.xticks([])
plt.show()
```

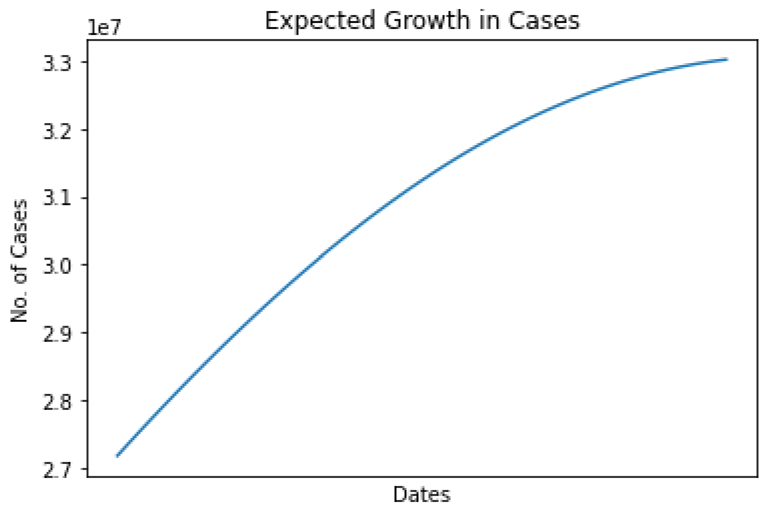


Here we can see our model is overtrained so we will use SVR.

Using SVR

```
# Predicting the confirmed cases in India till 31st July

# creating array of required dates
start = pd.to_datetime('2021-06-06')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_cases = (svr_rbf.predict(datelist.to_numpy().reshape(-1, 1)
/ maxDate))*maxConfirmedCases

# plotting predicted cases
```

```
sns.lineplot(x=datelist, y=predicted_cases)
plt.xlabel('Dates')
plt.ylabel('No. of Cases')
plt.title('Expected Growth in Cases')
plt.xticks([])
plt.show()
```



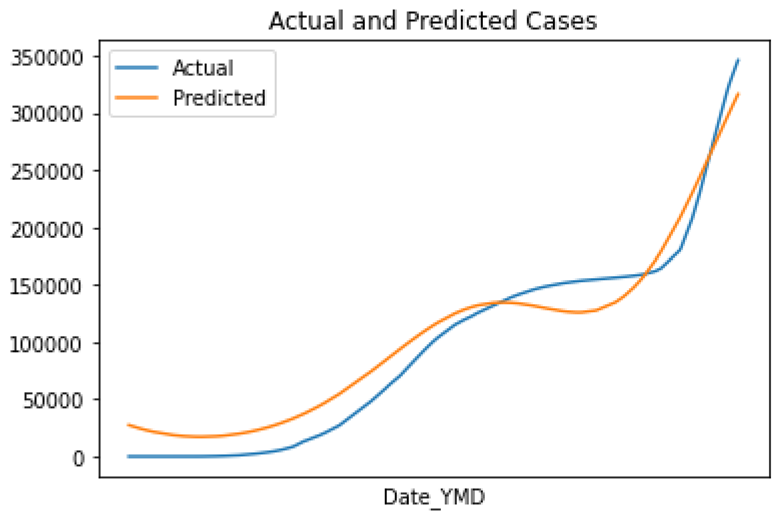
```
#number of cases on 31st July
start = pd.to_datetime('2021-07-30')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end - 
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_cases = (svr_rbf.predict(datelist.to_numpy().reshape(-1, 1)
/ maxDate))*maxConfirmedCases

print(int(predicted_cases))
```

Number of cases calculated using our model = 33156529

Objective 2 : Predicting total deaths

Fetching required data

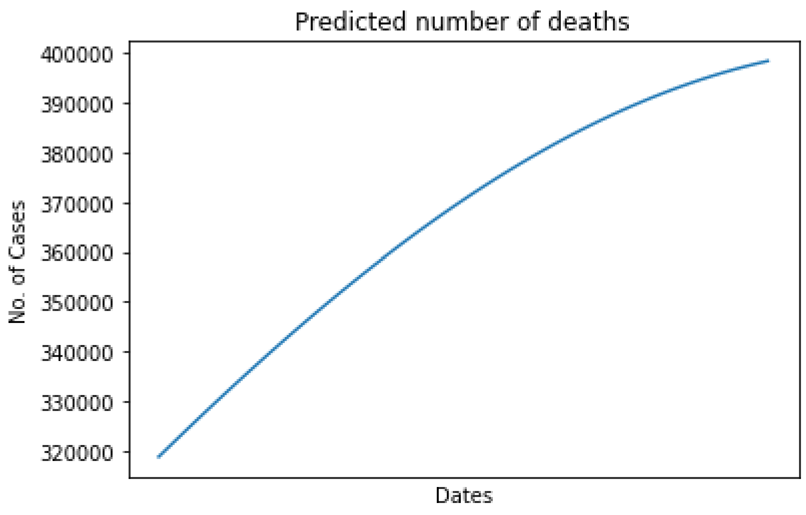
```
y = data.iloc[:, 7].values

X_train, X_test, y_train, y_test = train_test_split(data['Date_YMD'],
y, test_size=0.2, random_state=0)
X_train = X_train.to_numpy().reshape(-1, 1)
X_test_reshaped = X_test.to_numpy().reshape(-1, 1)


# Scaling current values down
maxDeaths = max(data['Total Deceased'])

# training model on normalized values
svr_rbf.fit(X_train/maxDate, y_train/maxDeaths)
predicted_deaths = svr_rbf.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted deaths
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_deaths*maxDeaths)
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Cases')
plt.show()
# calculating accuracy of model
svr_rbf.score(X_test_reshaped/maxDate, y_test/maxDeaths)
```



Model Accuracy = 94 %

# Predicting the total deaths in India till 31st July

```
# creating array of required dates
start = pd.to_datetime('2021-06-06')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_deaths = (svr_rbf.predict(datelist.to_numpy().reshape(-1,
1) / maxDate))*maxDeaths

# plotting predicted cases
sns.lineplot(x=datelist, y=predicted_deaths)
plt.xlabel('Dates')
plt.ylabel('No. of Cases')
plt.title('Predicted number of deaths')
plt.xticks([])
plt.show()
```



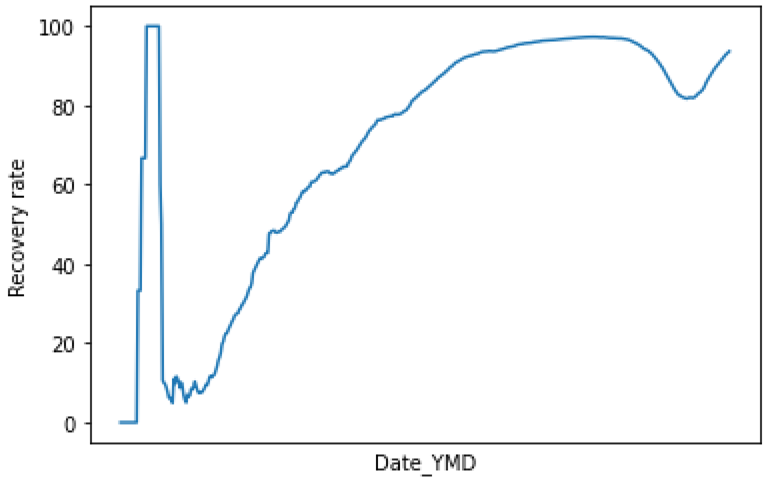
#number of deaths on 31st July

```
start = pd.to_datetime('2021-07-30')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_deaths = (svr_rbf.predict(datelist.to_numpy().reshape(-1,
1) / maxDate))*maxDeaths

print(int(predicted_deaths)
```

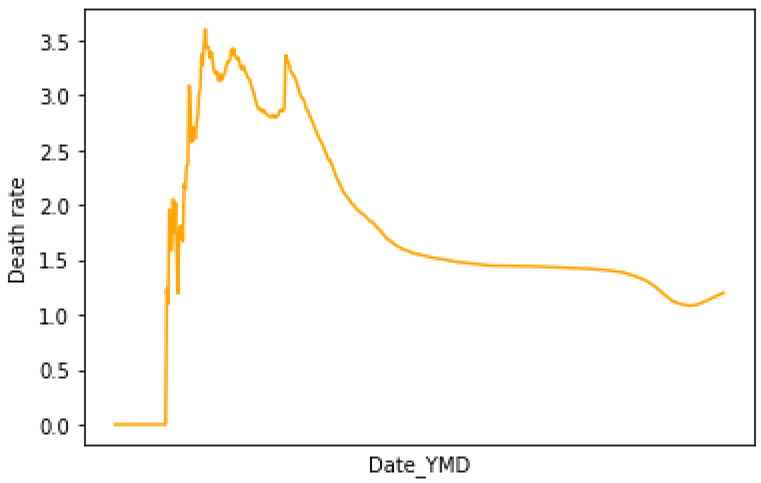
Number of deaths predicted = 39800

Objective 3 : Predict recovery rate and death rate in India for the period 15th June to 31st of July 2021

```
recovery_rate = (data['Total Recovered'] / data['Total
Confirmed'])*100
death_rate = (data['Total Deceased'] / data['Total Confirmed'])*100


sns.lineplot(x=data['Date_YMD'], y=recovery_rate)
plt.xticks([])
plt.ylabel('Recovery rate')
plt.show()
```



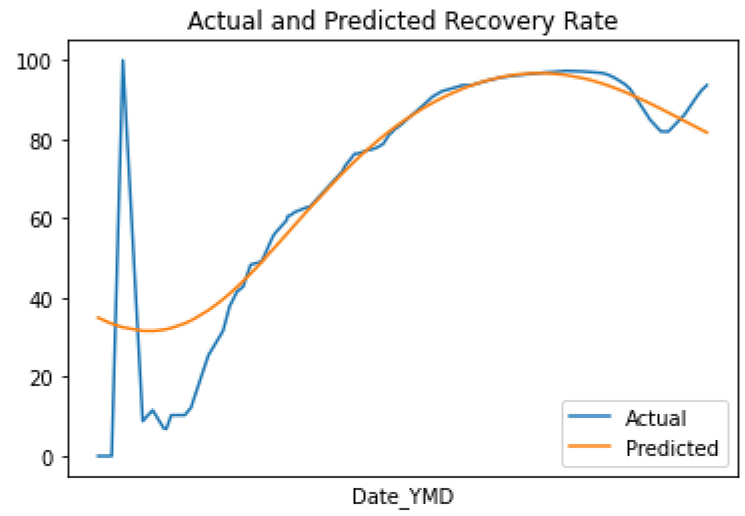
```
sns.lineplot(x=data['Date_YMD'], y=death_rate, color = 'orange')
plt.xticks([])
plt.ylabel('Death rate')
plt.show()
```



3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Death rate
Date_YMD

Predicting recovery rate

```
X_train, X_test, y_train, y_test = train_test_split(data['Date_YMD'],
recovery_rate, test_size=0.2, random_state=0)
X_train = X_train.to_numpy().reshape(-1, 1)
X_test_reshaped = X_test.to_numpy().reshape(-1, 1)


# training model on normalized values
svr_rbf.fit(X_train/maxDate, y_train)
predicted_recovery_rate = svr_rbf.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted cases
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_recovery_rate)
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Recovery Rate')
plt.show()
# calculating accuracy of model
svr_rbf.score(X_test_reshaped/maxDate, y_test)
```

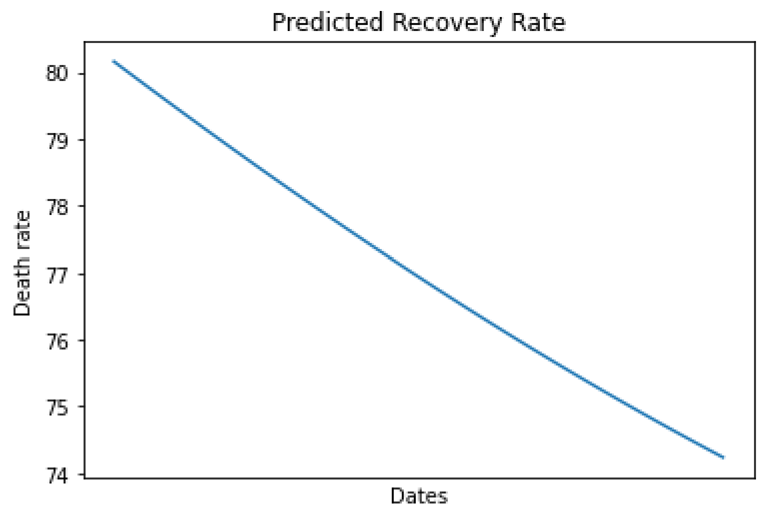


Model accuracy = 84%

# Predicting the recovery rate in India from 15th June to 31st July

```
# creating array of required dates
start = pd.to_datetime('2021-06-15')
end = pd.to_datetime('2021-07-31')
```

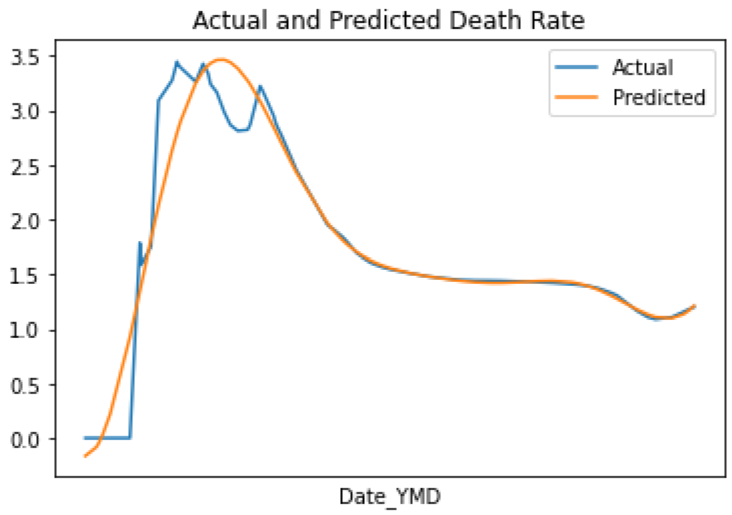
```
datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_recovery_rate =
(svr_rbf.predict(datelist.to_numpy().reshape(-1, 1) / maxDate))

# plotting predicted cases
sns.lineplot(x=datelist, y=predicted_recovery_rate)
plt.xlabel('Dates')
plt.ylabel('Death rate')
plt.title('Predicted Recovery Rate')
plt.xticks([])
plt.show()
```



Predicting Death rate

```
X_train, X_test, y_train, y_test = train_test_split(data['Date_YMD'],
death_rate, test_size=0.2, random_state=0)
X_train = X_train.to_numpy().reshape(-1, 1)
X_test_reshaped = X_test.to_numpy().reshape(-1, 1)

# training model on normalized values
svr_rbf.fit(X_train/maxDate, y_train)
predicted_death_rate = svr_rbf.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted cases
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_death_rate)
```

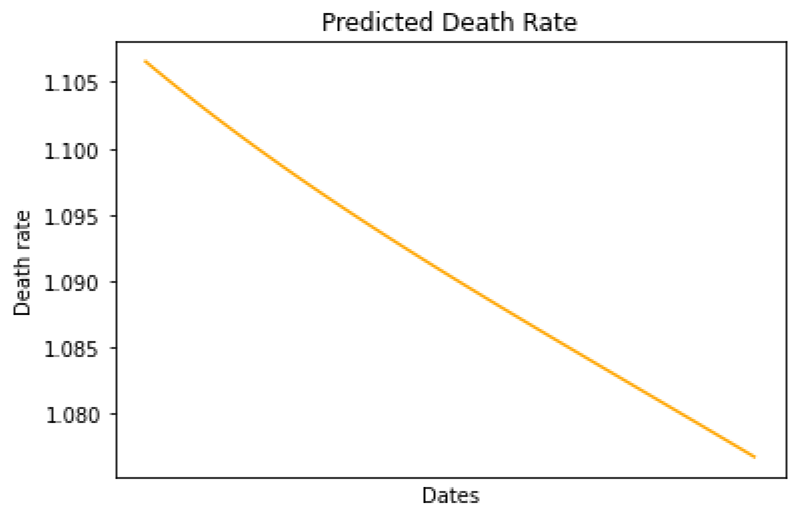
```
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Recovery Rate')
plt.show()
# calculating accuracy of model
svr_rbf.score(X_test_reshaped/maxDate, y_test)
```



Model accuracy= 92%

# Predicting the death rate in India from 15th June to 31st July

```
# creating array of required dates
start = pd.to_datetime('2021-06-15')
end = pd.to_datetime('2021-07-31')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_death_rate =
(svr_rbf.predict(datelist.to_numpy().reshape(-1, 1) / maxDate))

# plotting predicted cases
sns.lineplot(x=datelist, y=predicted_death_rate, color='orange')
plt.xlabel('Dates')
plt.ylabel('Death rate')
plt.title('Predicted Death Rate')
plt.xticks([])
plt.show()
```



Objective 4: Predicting infection rate of Indians

```
infection_rate = (int(predicted_cases)/1366400000)*100
```

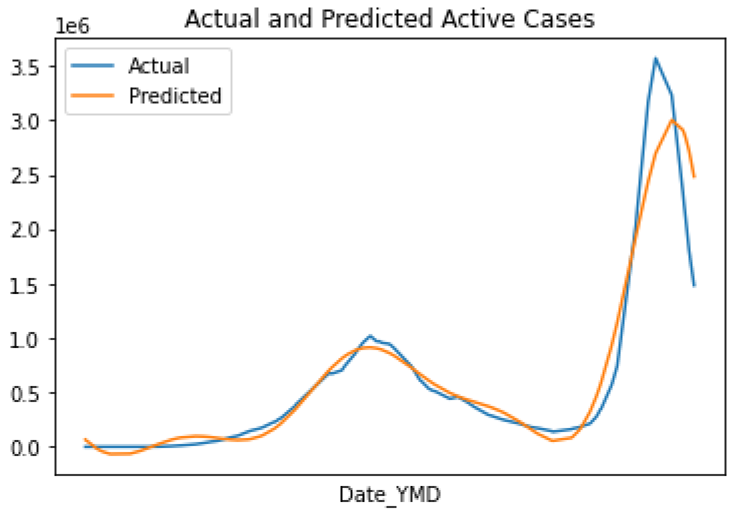
Infection rate = 2.416731850117096

Infection rate ios calculated using total confirmed cases as data was not available for age groups 35-50

Objective 5: Finding date with 10% of peak cases

```
active_cases = data['Total Confirmed'] - data['Total Recovered'] -
data['Total Deceased']
maxActiveCases = max(active_cases)


svr_rbf = SVR(kernel='rbf', epsilon=0.02, C=100)

X_train, X_test, y_train, y_test = train_test_split(data['Date_YMD'],
active_cases, test_size=0.2, random_state=0)
X_train = X_train.to_numpy().reshape(-1, 1)
X_test_reshaped = X_test.to_numpy().reshape(-1, 1)

# training model on normalized values
svr_rbf.fit(X_train/maxDate, y_train/maxActiveCases)
predicted_active_cases = svr_rbf.predict(X_test_reshaped/maxDate)
# plotting test cases vs predicted cases
sns.lineplot(x=X_test, y=y_test)
sns.lineplot(x=X_test, y=predicted_active_cases*maxActiveCases)
```

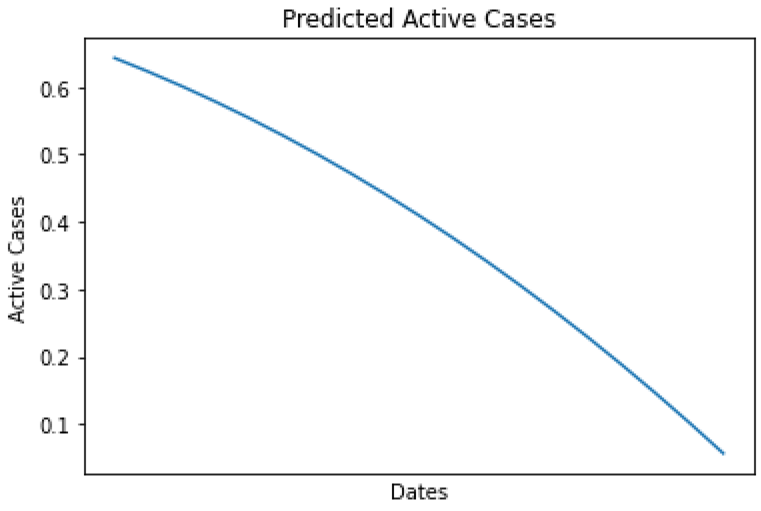
```
plt.legend(['Actual', 'Predicted'])
plt.xticks([])
plt.title('Actual and Predicted Active Cases')
plt.show()
# calculating accuracy of model
svr_rbf.score(X_test_reshaped/maxDate, y_test/maxActiveCases)
```



Model Accuracy = 89%

# Predicting the death rate in India from 15th June to 31st July

```
# creating array of required dates
start = pd.to_datetime('2021-06-6')
end = pd.to_datetime('2021-06-25')

datelist = pd.date_range(start, end -
timedelta(days=1)).map(dt.datetime.toordinal)
predicted_active_cases =
(svr_rbf.predict(datelist.to_numpy().reshape(-1, 1) / maxDate))

# plotting predicted cases
sns.lineplot(x=datelist, y=predicted_active_cases)
plt.xlabel('Dates')
plt.ylabel('Active Cases')
plt.title('Predicted Active Cases')
plt.xticks([])
plt.show()
```



```
start = 0
end = 0
flag = False
for i in range(0, len(predicted_active_cases)):
    if predicted_active_cases[i] < 0.2 and predicted_active_cases[i]
> 0.1 and not flag:
        start = i
        flag = True
    if predicted_active_cases[i] < 0.1:
        end = i
        break

start = dt.datetime.fromordinal(datelist[start])
end = dt.datetime.fromordinal(datelist[end])
print(start)
print(end)
```

```
2021-06-21 00:00:00
2021-06-24 00:00:00
```

In India number of covid cases less than 10% of peak value will be on 24 June 2021