Q1. SVD for Image Compression:

Images usually take a lot of memory space and the storage of high-quality images is a problem. SVD solves this problem by reducing the size of the image until acceptable quality is reached. Image compression takes advantage of the fact that only a few singular values are large and we can approximate the image using the larger singular values.

Our approach to the problem:

Given	Inference
64 x 64 shape images	(m,n) = (64,64) M = m × n = 64 × 64 M = 4096
15 Subjects	For K = 1,2,3…..15 We have: $N^K = 10$
10 Images/Subject	

Displaying the first 50 images in the dataset

Algorithm:

1. For a subject K convert,
    a. Convert each of the images into a 4096 X 1 column (a_i^K where i = 1,2,3….10)
    b. Stack these images horizontally to obtain a 4096 X 10 matrix (A^K)

$$A^K = [a_1^K \;\; a_2^K \;\; a_3^K \;\; \ldots\ldots \;\; a_{10}^K]$$

    c. Find the mean of all the columns, as described in [1] (here N = 10)

$$a_{mean}^K = (1/N) \times \sum_{i=1}^{N} a_i^K$$

    d. Subtract the mean from each of the columns of the image matrix

$$x_i^K = a_i^K - a_{mean}^K \quad \text{for i=1,2,3…..N}$$

    to obtain mean centred matrix (X^K)

$$X^K = [x_1^K \;\; x_2^K \;\; x_3^K \;\; \ldots\ldots \;\; x_{10}^K]$$

First 50 images in the database after their corresponding mean is subtracted from them

e. We perform SVD on the mean centred matrix to get:

$$X^K = U^K \Sigma V^T$$

$$U^K = [\, u_1^K \;\; u_2^K \;\; u_3^K \;\; \ldots\ldots \;\; u_{4096}^K \,]$$ (M x M matrix i.e 4096 x 4096)

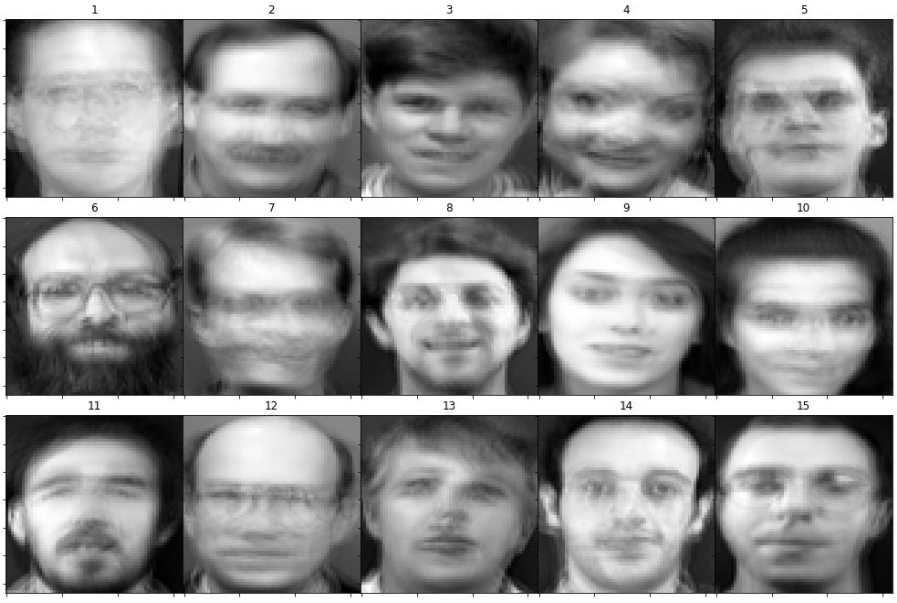
f. Since the question asks us to generate a single representative image for each of the subjects, we choose just one eigenface corresponding to the largest singular value. Due to the characteristic properties of SVD decomposition, this eigenface automatically ends up being the first eigenface in the U^K matrix,
I.e u_1^K eigenvector (4096 x 1 column vector)

g. This eigenface is then added with the corresponding mean (belonging to that subject) to obtain a representative image R^K :

$$R^K = u_1^K + a_{mean}^K$$

2. The same procedure is repeated for each of the subjects 15 representative images (R^K K = 0,1,2,….15) are obtained. The representative images are then reshaped to the original dimension i.e 64 x 64

Representative images corresponding to each of the subjects with their labels



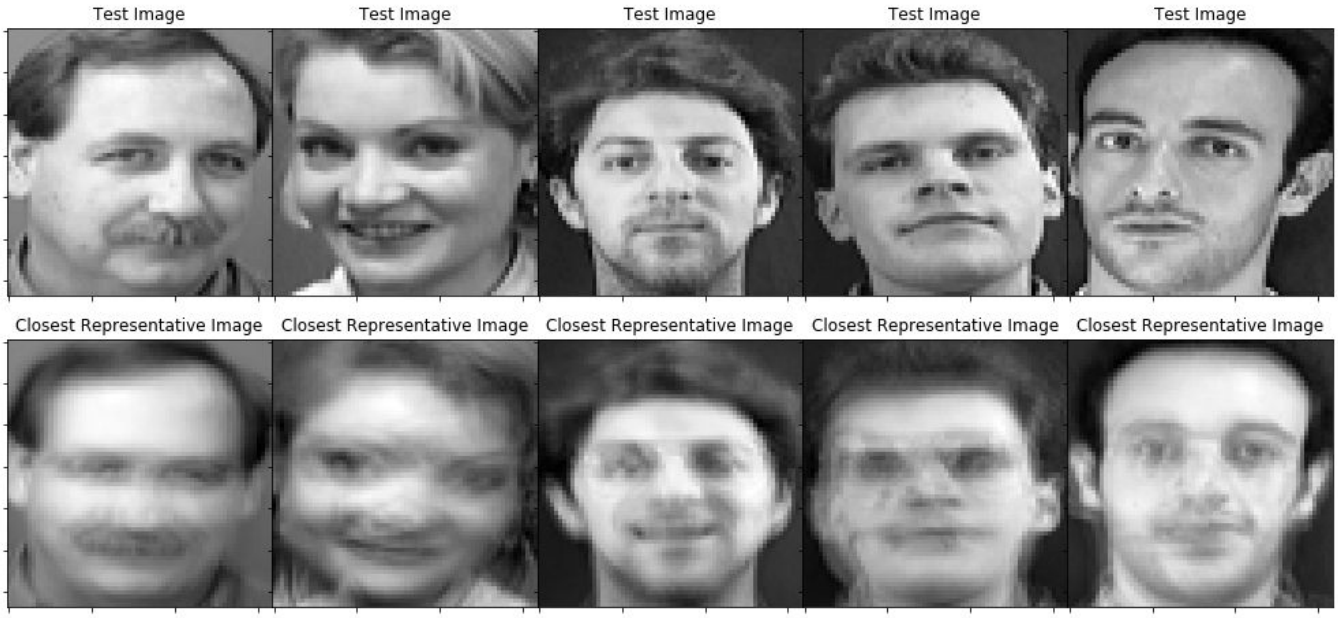
3. Given a test image T, we can check the subject/class that the test image belongs to by finding its Euclidean distance (L2 norm) for each of the representative images (R^K) using the following formula :

$$\delta_K(T, R^K) = \left\| T - R^K \right\|$$

The norm δ_K is the square root of the sum of the squared differences between the image pixels.

4. The representative image corresponding to the smallest norm is considered to be the subject that the test image T belongs to.

Testing performance of the classifier on 5 randomly chosen images from the dataset



Applying technique on all 150 images in the dataset we get the following result :

```
print(correct_preds,'out of the',total_samples,'samples have been correctly identified')
print('Accuracy of the face recognition algorithm is: ',round((correct_preds/total_samples)*100,2),'%')
```

```
148 out of the 150 samples have been correctly identified
Accuracy of the face recognition algorithm is:  98.67 %
```

Q2. Logistic Regression on reactor data

The provided dataset describes the operating conditions of a reactor and contains class labels about whether the reactor will operate or fail under the respective operating conditions. There are five operating conditions: temperature, pressure, feed flow rate, coolant flow rate and inlet reactant concentration. The test column with the fail/ pass annotation has been converted to a binary column of 0s and 1s.

Statistics of the data:

	Temperature	Pressure	Feed Flow rate	Coolant Flow rate	Inlet reactant concentration	Test
count	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	546.76643	25.493270	125.029060	2295.797770	0.302692	0.585000
std	86.85878	14.252407	43.508159	763.680625	0.116062	0.492969
min	400.31000	1.060000	50.030000	1002.530000	0.100300	0.000000
25%	469.73500	12.725000	88.587500	1635.682500	0.199075	0.000000
50%	545.80000	25.375000	124.590000	2268.710000	0.308850	1.000000
75%	618.87750	37.820000	162.562500	2983.692500	0.401625	1.000000
max	699.87000	49.890000	199.960000	3595.620000	0.499600	1.000000

The mean values and standard deviation for all the operating conditions can be seen in the above image

```
df.groupby('Test').mean()
```

Test	Temperature	Pressure	Feed Flow rate	Coolant Flow rate	Inlet reactant concentration
0	547.634964	26.320747	129.829783	1605.060819	0.301568
1	546.150291	24.906256	121.623419	2785.807744	0.303489

The above image represents the mean values of the operating conditions categorized by whether the reactor operated or failed to operate under the given operating conditions.

In order to split the data, the first 700 rows of the 1000x6 dataset have been taken as the training data (**train_data**) and the rest of the 300 rows (**test_data**) have been taken as the test data. Next, the training data is split into **X_train** (containing the operating conditions) and **y_train** (containing the annotation as to whether the reactor operated or failed to). Similarly, the test data is split into **X_test** and **y_test**.

The values of the boundary conditions have large differences in their ranges (for example, temperature range is 400-700 K, inlet reactant concentration range is 0.1-0.5 mole fraction, while the range for coolant flow rate is 1000-3600 L/hr). Thus, min-max normalization is done on the operating condition values such that all the values are in the range of 0 to 1. This is done

because normalizing the features used in a model lets us obtain the optimum coefficients that best fit the data more accurately.[1]

The equation used for this model is of the form **$z = a + bx_1 + cx_2 + dx_3 + ex_4 + fx_5$**. To get the intercept value of the equation, an **intercept column** is added to both X_train and X_test. This intercept column consists of only 1s. The values a, b, c, d, e, f are characterized by the operating condition values of the dataset whereas the parameters x_1, x_2, x_3, x_4, x_5 are stored in **theta**.

The activation function for logistic regression is the **sigmoid** function, $S(z) = \frac{1}{1+e^{-z}}$ and has been defined in the code:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

The **objective function/ cost function** is taken as

$$J(\theta) = -\frac{1}{m} \cdot (-y^T \log log\ (h) - (1-y)^T \log log\ (1-h)\ ), \text{ where } h = S(X * \theta)$$

For this dataset, X corresponds to the operating condition values, θ corresponds to the parameter values, y corresponds to the True/False annotation, and m is the length of y. The dimension of θ is 6 x 1.

Fitting the logistic regression model on the training set:

The initialisation for θ, **initial_theta** is taken as the mean of X_train:

```
initial_theta = X_train.mean()
initial_theta
```

```
Intercept                     1.000000
Temperature                   0.487468
Pressure                      0.495030
Feed Flow rate                0.497109
Coolant Flow rate             0.487800
Inlet reactant concentration  0.498246
dtype: float64
```

Now, a function **cost_function** is defined in order to calculate the value of objective function at every iteration of the parameters. The inputs to this function are: θ (parameters; dimension: 6x1), X (operating condition values; dimension: 700x6 for X_train), y (True/False annotations; dimension: 700x1). Definition of the function:

```
def cost_function(theta, X, y):

    m = len(y)

    predictions = sigmoid(X.dot(theta))
    cost = (1/m) * ((-(np.transpose(y)).dot(np.log(predictions)) - np.transpose(1-y).dot(np.log(1-predictions))))

    return cost
```

Defining the gradient descent function, **gradient_descent:**

Inputs to this function: X_train, y_train, θ, α (learning rate), number of iterations

```
def gradient_descent(X, y, theta, alpha, iterations):

    iter = 0
    m = len(y)

    Cost_history = np.zeros(shape = [iterations, 1])

    while(iter < iterations):

        Cost_history[iter] = cost_function(theta, X, y)
        theta = theta - alpha * (1/m) * np.transpose((np.transpose(X.dot(theta)-y).dot(X)))
        iter = iter + 1

    return Cost_history, theta
```

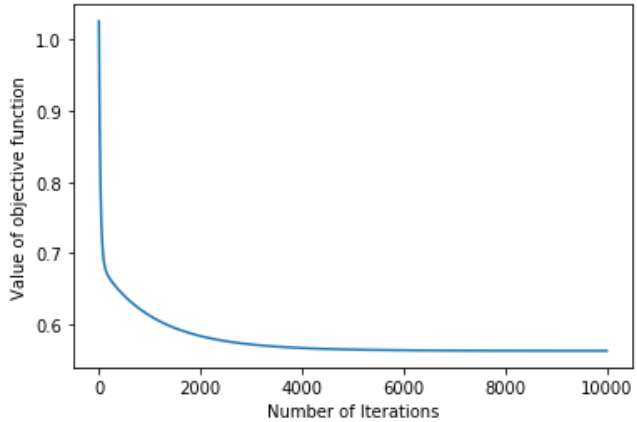
Theory: Gradient is the extension of derivative in multi-dimensional space; it tells the direction along which the loss or error is optimally minimized. Therefore, the vectorised formula for gradient descent is simply:

$\theta = \theta - \alpha \frac{\partial}{\partial \theta} J(\theta)$ repeated until convergence or in our case, for the number of iterations

For every iteration, the value of the objective function for the corresponding value of θ is stored in the array **Cost_history**.

Assumption: learning rate = 0.01, iterations = 10000.

Thus, passing these values into the function gradient_descent, we can plot the value of the objective function vs. the number of iterations. The graph obtained is:



As can be seen, the value of objective function becomes nearly constant after 6000 iterations.

The final value of the parameters corresponding to the operating conditions, theta is:

```
theta
```

```
Intercept                          0.173521
Temperature                       -0.059420
Pressure                          -0.188315
Feed Flow rate                    -0.185543
Coolant Flow rate                  1.280578
Inlet reactant concentration      -0.030402
dtype: float64
```

The **predictions on X_test**,

$$predictions = S(X\_test * \theta)$$

Assumption: The **threshold value** for the sigmoid function is taken as **0.615** for the best results.

Performance of the model on test data:

The **RMSE** (Root Mean Square Error) is **5.66%**

```
error = sum((predictions - y_test)**2)/len(y_test)
error
```

```
0.056666666666666664
```

The **Confusion Matrix** is as follows:

```
confusion_matrix
```

Predicted	0	1
Actual		
0	104	9
1	8	179

From the obtained confusion matrix,

- No. of True Positives = 179
- No. of True Negatives = 109
- No. of False Positives = 9
- No. of False Negatives = 8

Now, the F1 score is calculated from the formula:

$$F1\ score = \frac{2*Precision*Recall}{Precision+Recall}$$

Where

$$Precision = \frac{True\ Positives}{True\ Positives+False\ Negatives}$$

And

$$Recall = \frac{True\ Positives}{True\ Positives+False\ Negatives}$$

For our dataset,

Precision = 0.9521276595744681,

Recall = 0.9572192513368984,

F1 score = 0.95466666666666

Q3. An Analysis of COVID-19 Trends in India

Programming Language used: Python

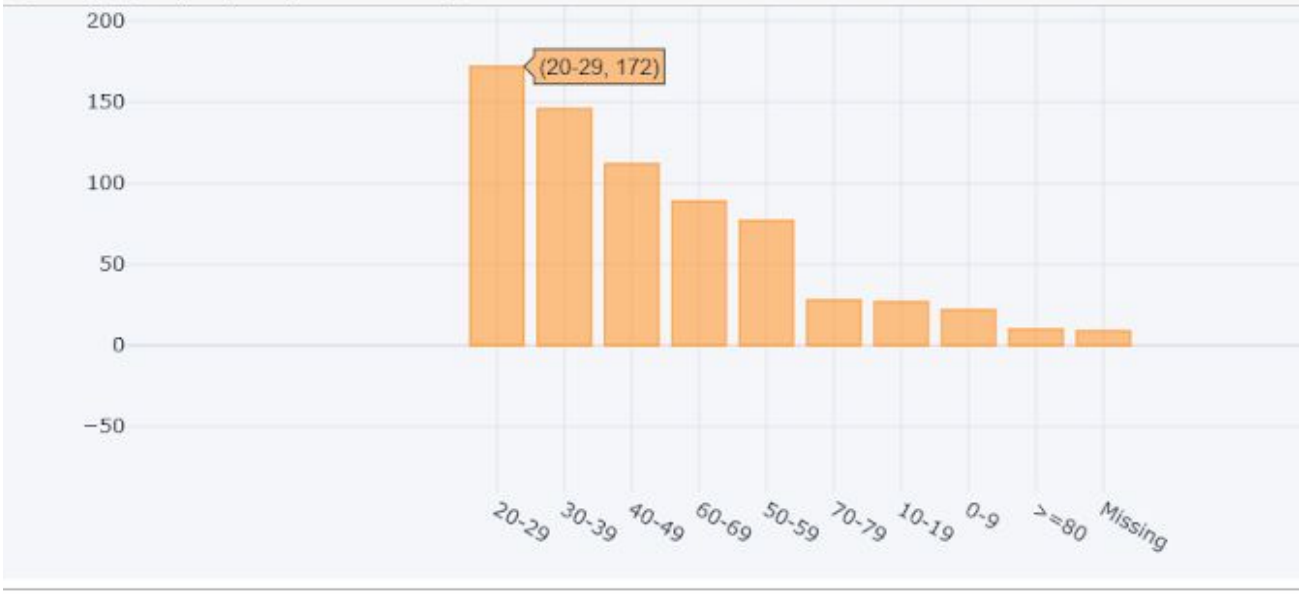
Libraries used:

- Pandas - for reading and manipulating data from the files
- Matplotlib - for plotting graphs
- Geopandas - for plotting geographic plots
- Numpy - for array and matrix manipulation
- Plotly - for interactive plots
- Cufflinks - wrapper library for Plotly

Dataset used:
https://www.kaggle.com/sudalairajkumar/covid19-in-india

1. Most infected Age Groups

Data file: AgeGroupDetails.csv



By sorting the data obtained from AgeGroupDetails.csv in decreasing order of cases recorded, we find the ages 20-29yrs(172 cases) most infected by the virus.

2. Daily cases observed, recovered, deaths country-wide and state-wise.

Data files used: covid_19_india.csv (updated till 10.05.2020)

State-wise graphs:

Taking input from the user for the state whose details are required.

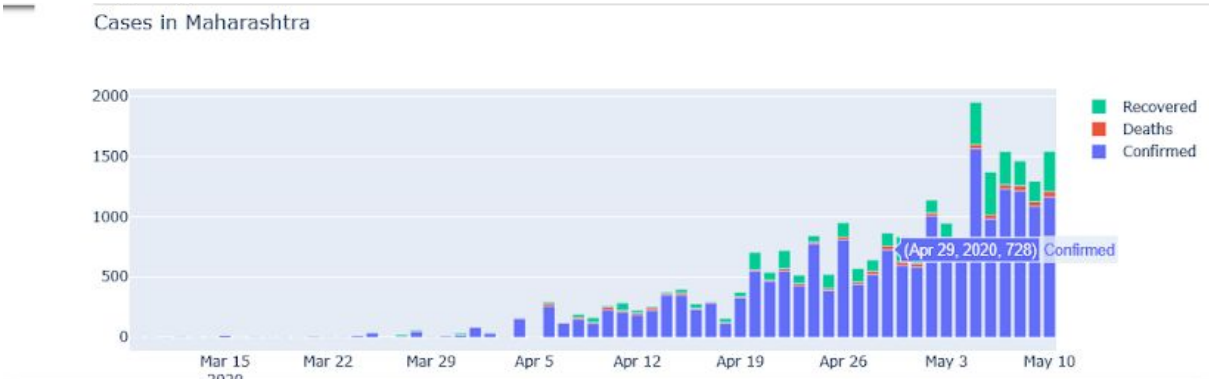
```
state = input("Enter State/UninonTerritory ")
print('State:',state)
```

```
Enter State/UninonTerritory Maharashtra
```

Day-wise statistics are obtained by performing differences between the recorded cumulative observed, recovered and deaths of consecutive days. These columns are added to the working data frame.

	Date	State/UnionTerritory	Cured	Deaths	Confirmed	Cured_perday	Deaths_perday	Confirmed_perday
	2020-05-06	Maharashtra	2819	617	15525	354.0	34.0	984.0
	2020-05-07	Maharashtra	3094	651	16758	275.0	34.0	1233.0
	2020-05-08	Maharashtra	3301	694	17974	207.0	43.0	1216.0
	2020-05-09	Maharashtra	3470	731	19063	169.0	37.0	1089.0
	2020-05-10	Maharashtra	3800	779	20228	330.0	48.0	1165.0

A stack type, interactive bar graph is plotted for per day Confirmed, Deaths, Recovered for the input state.



Nation-wide graph:

Nation-wide statistics are obtained by performing date-wise summation of cases across all states, from the datafile ‘covid_19_india.csv’

	Date	Confirmed_sum	Deaths_sum	Cured_sum
97	2020-05-06	49391	1694	14183
98	2020-05-07	52952	1783	15267
99	2020-05-08	56342	1886	16540
100	2020-05-09	59662	1981	17847
101	2020-05-10	62939	2109	19358

Per-day confirmed cases, deaths, recovered are computed using the same logic employed in the state-wise distribution.

A stack type, interactive bar graph is plotted.

3. Identify the positive cases on a state level. Quantify the intensity of virus spread for each state.

Data files: population_india_census2011.csv, covid_19_india.csv
Shapefile: Indian_States.shp

The population density data from population_india_census2011.csv is modified so that the non numeric characters are removed and the data type becomes an int/float instead of a string. Intensity is calculated by dividing the number of confirmed cases in a state by the population density of the state.

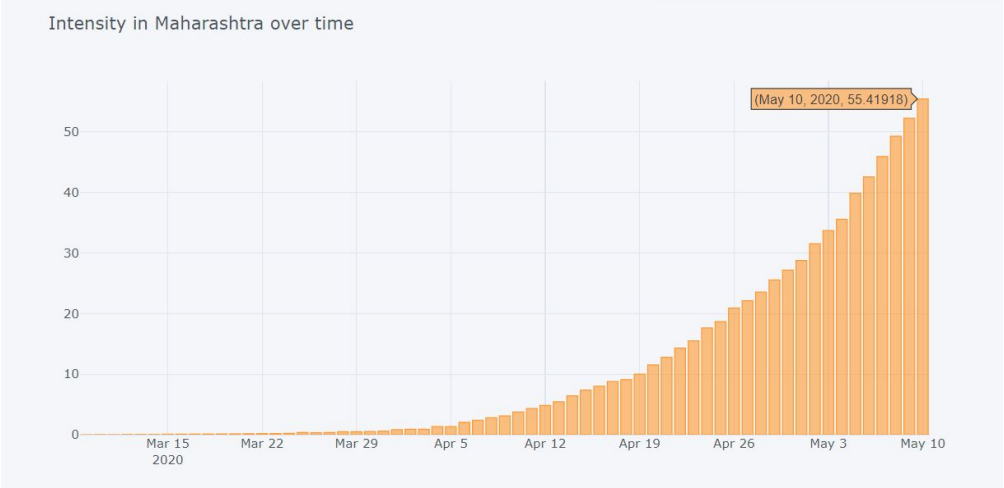
Graph for certain state:

Taking input from the user for the state whose details are required:

```
state = input("Enter State ")
filt = by_state['State/UnionTerritory']== state
```

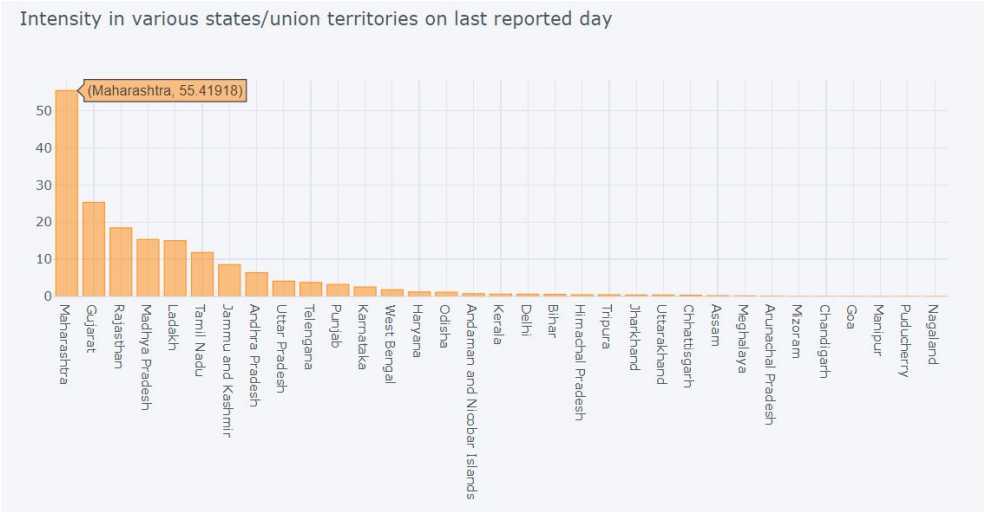
```
Enter State Maharashtra
```

An interactive bar graph is then plotted for the state which the user has specified



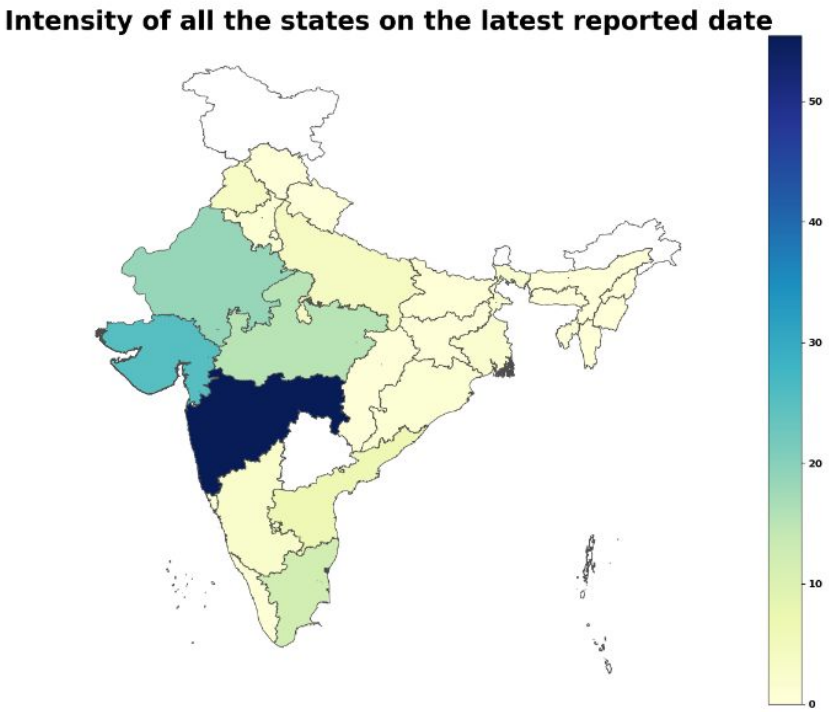
Graphing Intensity for all the States and UTs:

The intensity data for each state on the last day is computed by dividing the number of confirmed cases by the population density for each state/UT on the last reported date. An interactive bar graph is plotted.



Choropleth Map of Intensity spread:

The shapefile data is read from Indian_States.shp and the polygon data is used to plot the intensity of the virus spread in all the states on the state map of India with the colormap “YlGnBu”. States without the virus appear as white on the map.



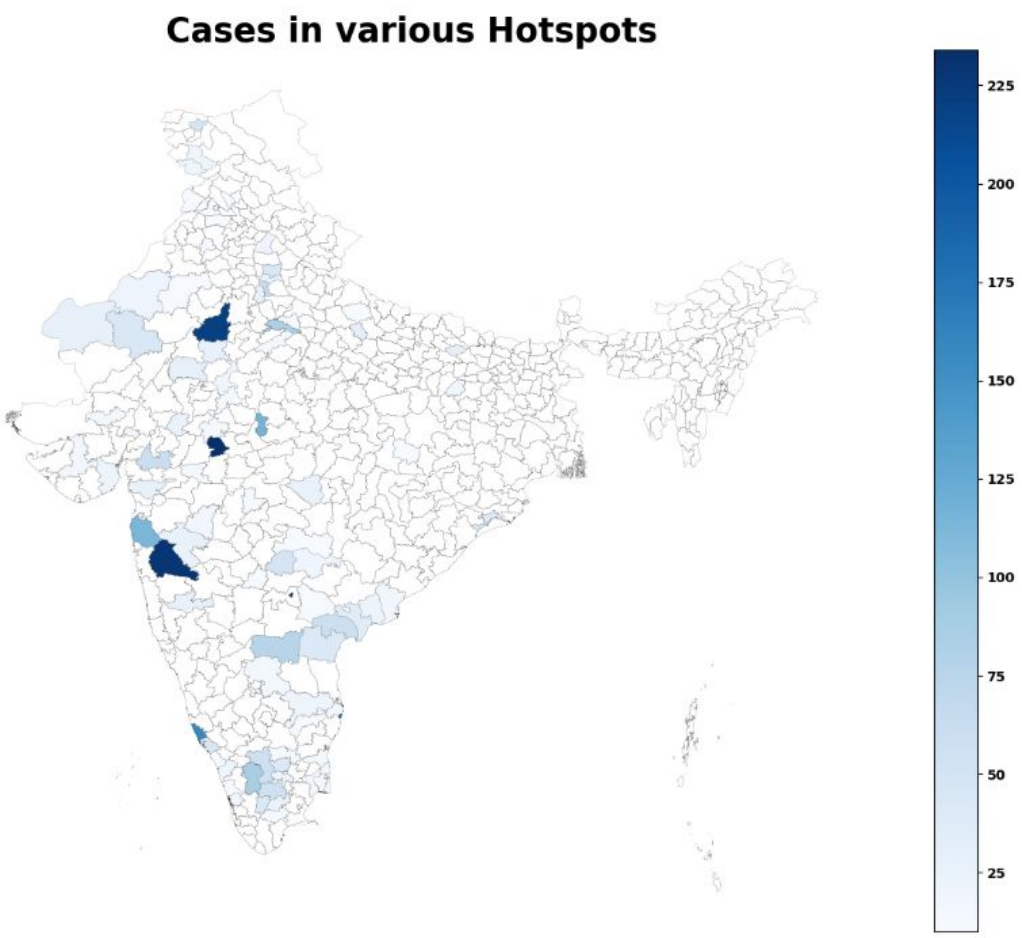
4. List places in the country which are active hotspots/clusters as on 10.04.2020.

Data files: IndividualDetails.csv
Shape file: IND_adm2.shp (District-Wise map of India)

Listing out the various hotspots:

	detected_district	Active_Cases
0	Mumbai	935
1	Indore	234
2	Pune	229
3	Hyderabad	223
4	Jaipur	220
...	...	...
123	Korba	10
124	Ludhiana	10
125	Bidar	10
126	Belagavi	10
127	Kamareddy	10

Choropleth Map of Hotspots:
Certain locations like Mumbai aren't actually districts and their geometries are not available in the district map shapefile of India. The districts whose geometries are available are plotted on the map of India with the colormap "Blues". Districts with no hotspots appear white in colour.



5. Which states have the maximum change (consider increase and decrease separately) in the number of hotspots on a weekly basis from 20.03.2020 to 10.04.2020 (3 weeks).

Data Files: IndividualDetails.csv

Week-wise Hotspot Computation

Data is read from IndividualDetails.csv and divided into 3 different categories on the basis of what week the data is from. Then the week-wise hotspot data is computed

	detected_state	Hotspots1	Hotspots2	Hotspots3
0	Maharashtra	15	21	229
1	Uttar Pradesh	11	23	226
2	Kerala	11	13	122
3	Rajasthan	9	18	181
4	Tamil Nadu	9	27	264
5	Haryana	7	13	73
6	Delhi	7	7	15

Maximum increase in the number of hotspots from Week 1 to Week 2

The week-wise change is calculated and the states which have the five greatest positive integer changes in hospots from week 1 to 2 are tabulated.

	States with highest increase from W1 to W2	Increase
0	Tamil Nadu	18
1	Uttar Pradesh	12
2	Rajasthan	9
3	Karnataka	8
4	Telangana	8

Maximum increase in the number of hotspots from Week 2 to Week 3

The week-wise change is calculated and the states which have the five greatest positive integer changes in hospots from week 2 to 3 are tabulated.

	States with highest increase from W2 to W3	Increase
0	Tamil Nadu	237
1	Maharashtra	208
2	Uttar Pradesh	203
3	Rajasthan	163
4	Gujarat	138

Maximum decrease in the number of hotspots from Week 1 to Week 2

The week-wise change is calculated and the states which have the five greatest negative integer changes in hospots from week 1 to 2 are tabulated. If there is no state with a decrease in hotspots, a message is displayed stating the same.

```
There is no state that shows a decrease in the number of hotspots between Week 1 and Week 2
```

Maximum decrease in the number of hotspots from Week 2 to Week 3

The week-wise change is calculated and the states which have the five greatest negative integer changes in hospots from week 2 to 3 are tabulated. If there is no state with a decrease in hotspots, a message is displayed stating the same.

```
There is no state that shows a decrease in the number of hotspots between Week 2 and Week 3
```

6. Quantify primary, secondary and tertiary cases based on the percentage for the top 5 states with maximum cases till 10.04.2020.

Data File: IndividualDetails.csv
Shape File: Indian_States.shp

To divide the cases into primary secondary and tertiary, we have to do so by looking at the 'notes' field in Indian_States.shp, we use the following logic:

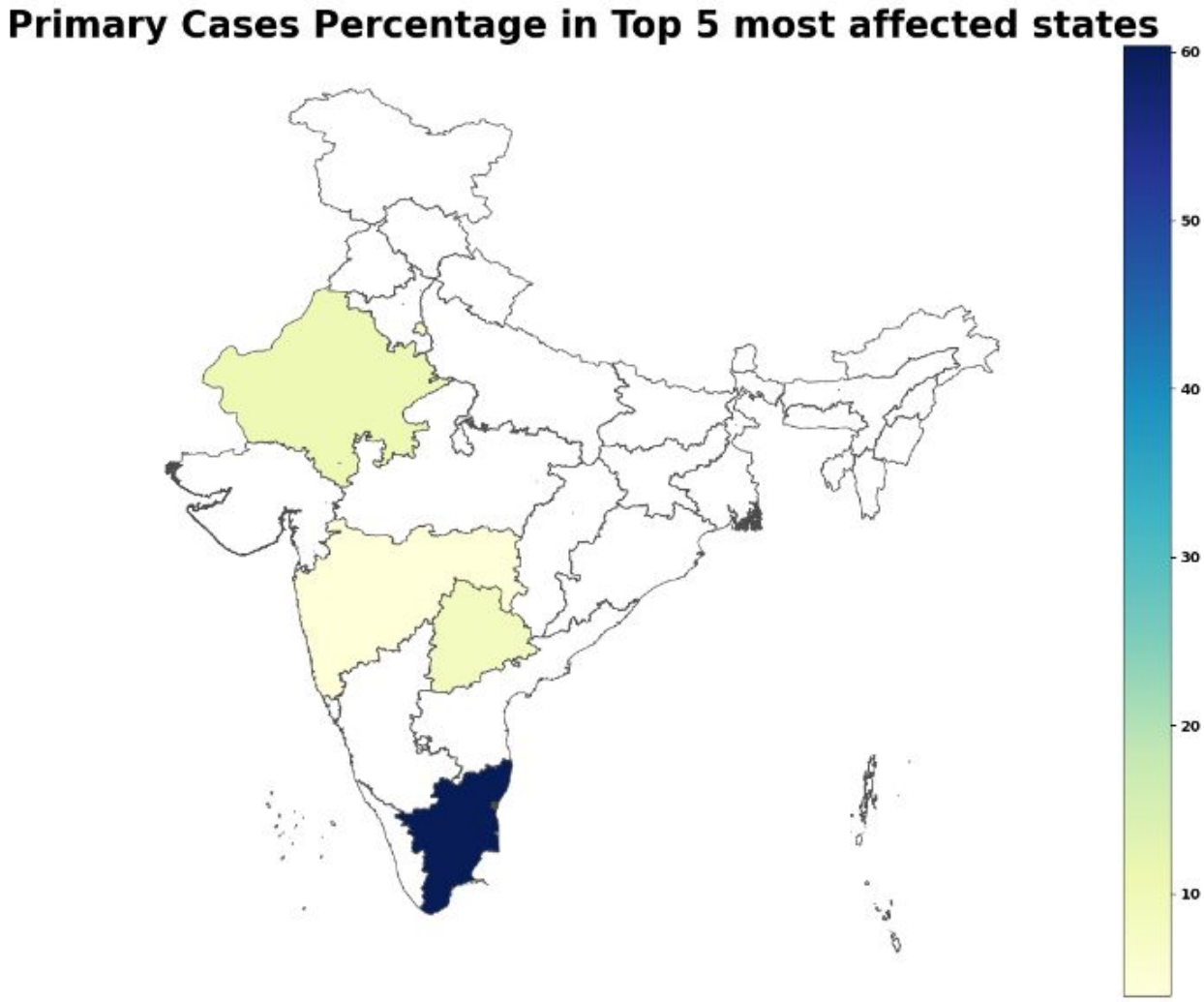
- If the note contains the substring "Travelled" then it is a primary case.
- If the note contains the substring "contact" or "contact" or the patient ID format {(letter,number),(letter,number,number),(letter,number,number,number),(letter,number,number,number,number),(letter,number,number,number,number,number)} it is a secondary case.
- If the case is neither a primary case, nor a secondary case, then it is a tertiary case

The states are sorted on the basis of the total number of cases before '10-04-2020' and the top 5 states with the highest cases are then plotted (on the map of india) on the basis of the state-wise percentage of the different cases.
Percentage of each case type is calculated by dividing the number of cases of one type by the total number of cases in the state.
States not in the top 5 most infected states (by '10-04-2020') appear as white on the choropleth map.

Choropleth map of Primary Cases Percentage:

The shapefile data is read from Indian_States.shp and the polygon data is used to plot the percentage of primary cases of the Top 5 most infected states on the state map of India with the colormap "YlGnBu".

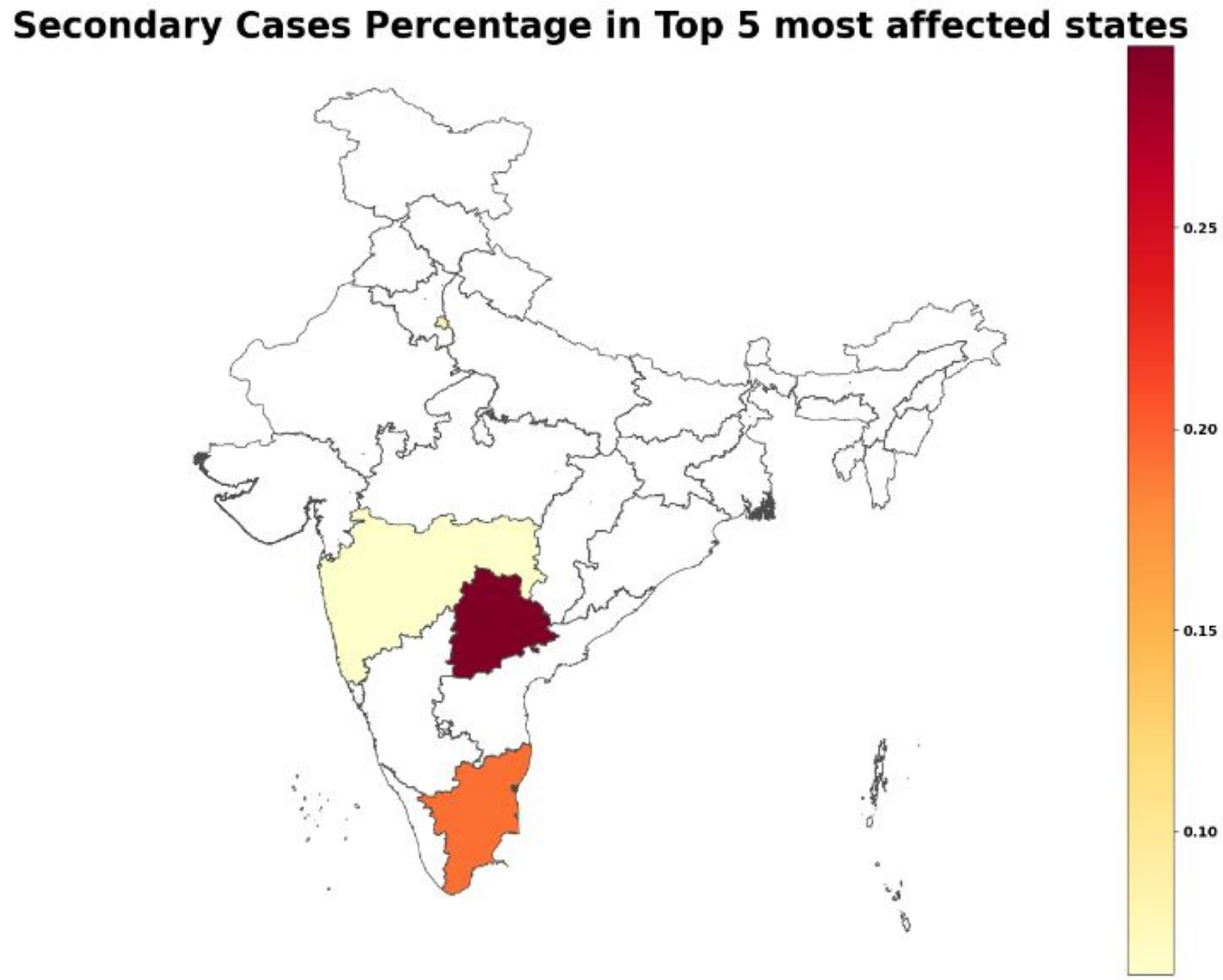
	detected_state	Primary Cases Percent
0	Maharashtra	3.939009
1	Tamil Nadu	60.373216
2	Delhi	5.315615
3	Rajasthan	10.160428
4	Telangana	8.418891



Choropleth map of Secondary Cases Percentage:

The shapefile data is read from Indian_States.shp and the polygon data is used to plot the percentage of secondary cases of the Top 5 most infected states on the state map of India with the colormap “YlOrRd”.
It is interesting to note that Rajasthan has no secondary cases and hence appears white on the map like the rest of the states not in the top 5 most infected states

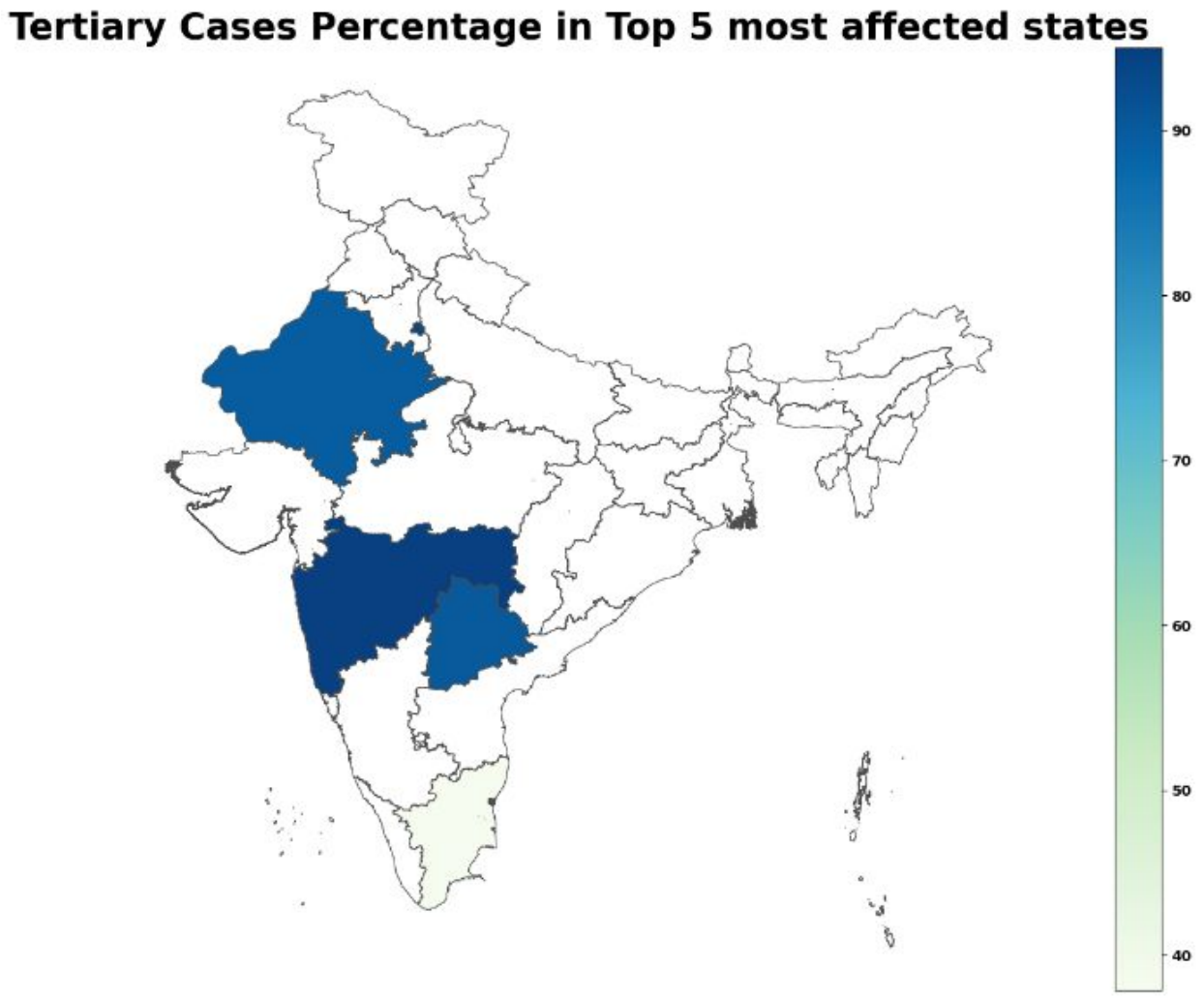
	detected_state	Secondary Cases Percent
0	Maharashtra	0.064582
1	Tamil Nadu	0.192789
2	Delhi	0.085846
3	Rajasthan	NaN
4	Telangana	0.295148



Choropleth map of Tertiary Cases Percentage:

The shapefile data is read from Indian_States.shp and the polygon data is used to plot the percentage of primary cases of the Top 5 most infected states on the state map of India with the colormap “GnBu”

	detected_state	Tertiary Cases Percent
0	Maharashtra	95.044473
1	Tamil Nadu	37.870472
2	Delhi	93.909192
3	Rajasthan	89.839572
4	Telangana	90.143737



7. Find out the number of additional labs needed from the current existing labs (assume 100 tests per day per lab) with an increase rate of 10% cases per day from 11.04.2020 - 20.04.2020. List out any further assumptions considered.

Data Files: ICMRTestingLabs.csv, StatewiseTestingDetails.csv

To find out the number of additional labs required, we need to find the *maximum per day increase* in the number of *total samples tested* for the virus.

Correlation between Positives cases and total samples tested
We are required to find the projected number of total samples tested.
We use the concept of test positivity ratio, for computing this.

Test Positivity Ratio(TPR) = Positive cases / Total samples tested

Since different States have different healthcare administrative systems and testing conditions, we compute the state-wise average TPR till the latest date of report(on/before 10.04.2020), for each state.

We shall use this average TPR for the subsequent days.

	State	TotalSamples_10th	Positive_10th	Mean_TPR
0	Andhra Pradesh	6374.0	365.0	0.065299
1	Arunachal Pradesh	206.0	1.0	0.004854
2	Assam	2863.0	29.0	0.013381
3	Bihar	5457.0	60.0	0.009790
4	Chandigarh	223.0	19.0	0.109396
5	Chhattisgarh	3473.0	18.0	0.006244
6	Delhi	11061.0	903.0	0.072526
7	Goa	354.0	7.0	0.020950
8	Gujarat	7718.0	378.0	0.046505

Maximum jump:

Since the positive cases are increasing at a rate of 10% per day, the maximum jump will occur on the last day, ie, on 20.04.2020.
The projected positive cases on 20.04.2020 is computed by subtracting the total projected positive cases till 20.04.2020 and 19.04.2020.
From this, the projected total samples tested on 20.04.2020 can be computed using the state-wise average TPR.

Computing Number of labs:

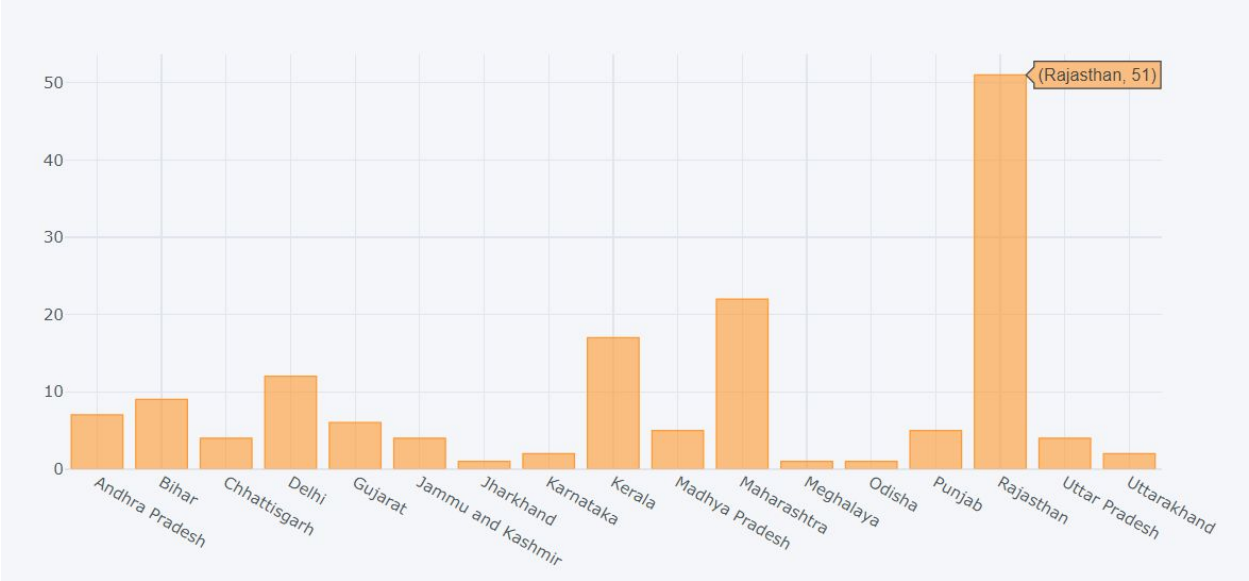
Assuming 100 tests per day per lab, the required number of labs is hence calculated as total samples tested / 100. The obtained float value is rounded-up using ceil() operation.

From the dataset ‘ICMRTestingLabs.csv’, the current number of labs is calculated by counting the labs per state.

Additional labs = Required labs - Current Labs

Plotting additional labs required:

Plotting the additional labs required using an interactive bar graph.



8. Plot the number of cases starting from 1st March - 10th April. Based on this plot can you comment on the popular notion of 'flattening the curve'

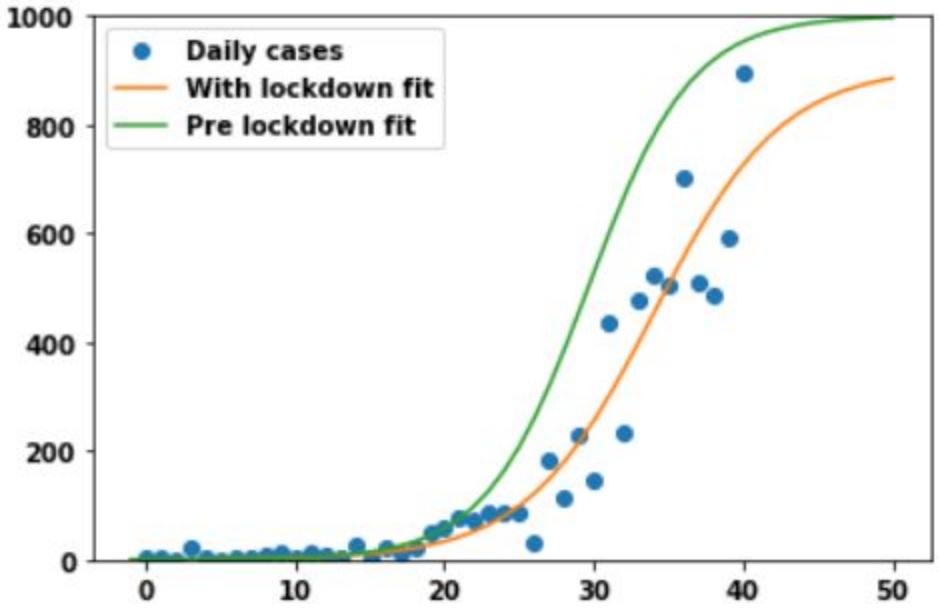
Data files:covid_19_india.csv

Total nation-wide cases each day is calculated by summing the number of cases in each state, each day.

An interactive graph is then plotted using the values computed

Curve fitting a sigmoid to show curve flattening

The graph below shows the effect of curve flattening that came about due to the lockdown.
The upper sigmoid is how the daily cases would have been without a lockdown.
The lower sigmoid depicts the trend of the curve up until April 10.
A sigmoid is used because one can see that it represents flattening of the curve, as seen in countries like China, whose daily cases graph was sigmoidal.



9. Comment on the 21-day Lockdown

Growth Rate & Doubling Time
Here, we use the metric of rate of growth and doubling time to analyse how efficient the 21-day lockdown was.
Doubling time is defined as the time required for the number of cases to double in size.
Since, a large surge or dip in the number of cases can give an erroneous impression of the spread, we calculate the doubling rate using a seven-day moving average.

- From the time series data (covid_19_india.csv), we first compute seven-day moving averages of cases.
- From this we compute the growth rate of the virus.
- We use the rule of 70 to calculate the doubling time from the growth rate.

Observations:

Lockdown period: 24 March 2020 – 14 April 2020

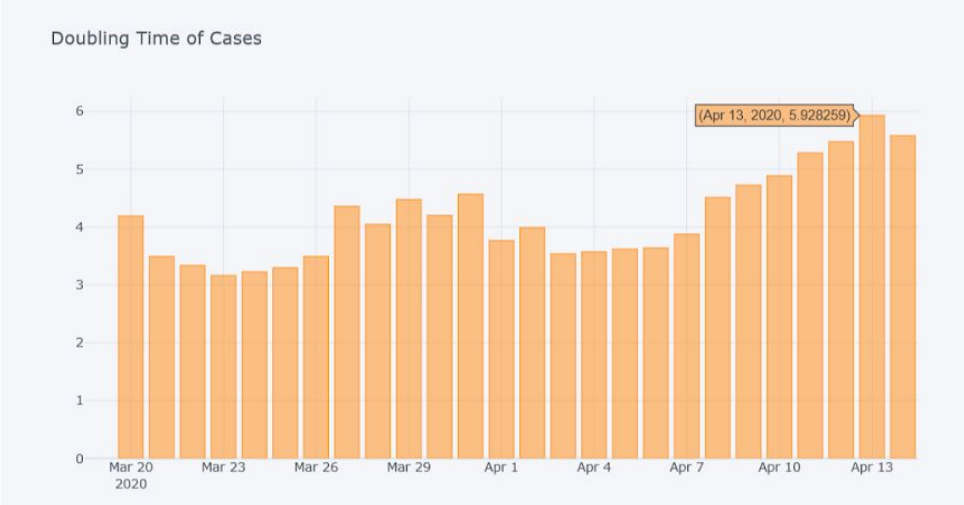
Growth Rate:

The growth rate in the number of cases overall dips with the effect of lockdown.

Doubling Time

- Overall, it can be seen that the doubling time went from 3.2 days before the lockdown to 6 days on 13th April. Hence, the lockdown was successful in flattening the curve.
- There was however an increase in growth factor in week two(31.03.2020-06.04.2020). This hike can be attributed to the tested positives from the Tablighi Jamaat congregation, two weeks prior.
- Since symptoms take about 14 days to manifest, it is advisable to attain a doubling time of more than 14 days before stringent lockdown measures are relaxed.

Date	23.03.2020 (before lockdown)	30.04.2020 (end of week 1)	06.04.2020 (end of week 2)	13.04.2020 (end of week 3)
Doubling Time	3.2 days	4.2 days	3.6 days	6 days



References:

1. https://www.robertoreif.com/blog/2017/12/16/importance-of-feature-scaling-in-data-modeling-part-1-h8nla