A Project Report

on

"Impact of Pandemic on Global Economy"

Submitted in partial fulfilment of the requirement of
University of Mumbai

For the Third Year Degree of **Bachelor of Engineering**

in

COMPUTER ENGINEERING

Submitted by

Alwin Jacob (101801)

Soumitra Bhagdikar (101807)

Bhagyashree L (101808)

Arnold Dsouza (101815)

Aryan Fernandes (101816)

Justin Mathew (101831)

K. Julie Jose (101837)

Merin Ann Philipose (101840)

Harshul Raina (101851)

Ashish Singh (101860)

Simran Kalaskar (101871)

Supervised by

Prof. Smita Dange

Department of Computer Engineering

Fr. Conceicao Rodrigues Institute of Technology

Sector 9A, Vashi, Navi Mumbai - 400703

UNIVERSITY OF MUMBAI 2020-2021

CERTIFICATE

This is to certify that the project entitled

Impact of Pandemic on Global Economy

Submitted by

Alwin Jacob (101801)

Soumitra Bhagdikar (101807)

Bhagyashree L (101808)

Arnold Dsouza (101815)

Aryan Fernandes (101816)

Justin Mathew (101831)

K. Julie Jose (101837)

Merin Ann Philipose (101840)

Harshul Raina (101851)

Ashish Singh (101860)

Simran Kalaskar (101871)

Date: **Supervisor:**

Time: **Examiner:1.**

2.

Head of Department:

INDEX

PROBLEM STATEMENT

The Covid 19 pandemic brought in a global health crisis unlike any in 75 years' history- killing people, spreading human suffering, and upending people's lives. But it is much more than a health crisis. It is a human, economic and social crisis. It forced people worldwide into some form of lockdown, caused widespread unemployment, underutilization of labor and capital, an increase in international trade costs, a drop in travel services, and a redirection of demand away from activities that require proximity between people. The gross domestic product also fell by 2 percent below the benchmark for the world, 2.5 percent for developing countries, and 1.8 percent for industrial countries. The governments, in response to the outbreak, have taken a host of complex measures including restrictions on public transport and international travel, school and workplace closures, bans on public gatherings and other steps to create social distancing, and are also investing in economic stimulus packages and emergency healthcare. Create a machine learning model(s) that: i)Predict GDP from the given parameters of covid impact ii)Estimate the required stringency index of a country, given its situation iii)Cluster countries according to the level to which they are affected by the pandemic. Visualize the findings appropriately.

DATASET

2.1.Source

Kaggle:https://www.kaggle.com/shashwatwork/impact-of-covid19-pandemic-on-the-global-economy

2.2.No of records

The total number of records in the dataset initially was 50419.

2.3.Description of fields

- **CODE-**country codes are short numeric geographical or alphabetic developed to represent countries and dependent areas, for use in data preprocessing, communication, etc.
- **COUNTRY-** It is the name of the country in our dataset which is affected by covid-19.
- **DATE-**It is the date till which the impact of covid-19 on a particular country is considered.
- **HDI (human_development_index)-** The Human development index is a summary measure of the average achievement in key dimensions of human development.
- **TC(Total cases)**- It is the total number of cases reported till the date in the corresponding Date column.
- **TD(Total Deaths)**-It is the total number of deaths reported till the date in the corresponding Date column.
- **STI(stringency_index)**-It is a composite measure based on different response indicators including school closures, workplace closures and travel bans, rescaled from value 1 to 100(100=strictest).
- **POP(population)**-It is the population of a country recorded till the date in the corresponding date column.

- **GDP CAP(GDP PER CAPITA)-**Gross Domestic Product per capita is a metric that breaks down a country's economic output per person and is calculated by dividing the GDP of a country by its population.

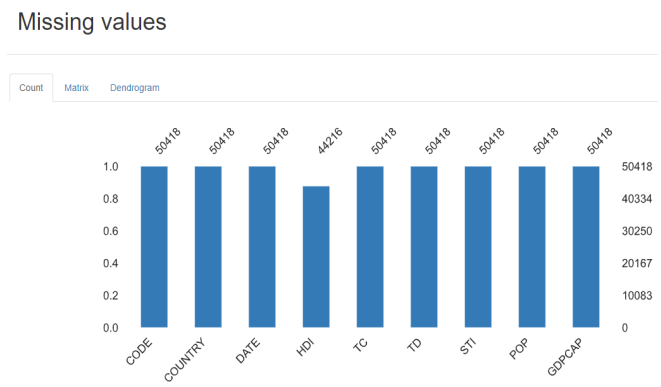
PREPROCESSING

The data was taken from Kaggle and analyzed.

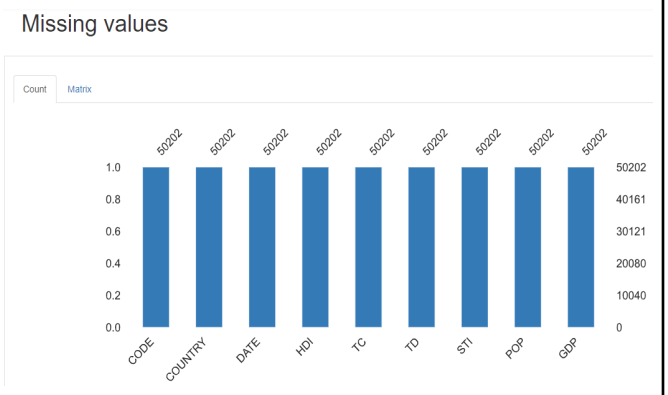
The first problem to be treated was the missing values. Columns like HDI, GDPCAP and TD had some null values in them. Usually, we solve the missing values problem by filling them with mean/median values . But the null values cannot be filled in with mean, or a SimpleImputer as these values differ with each country.

Hence we tried to replace the null values in these columns with the actual values which we got from the internet. But not all missing data was found on the internet. We replaced some values in TD by zeroes, in case of missing elements belonging to the initial dates. Some insufficient records had to be dropped entirely.

Initial analysis

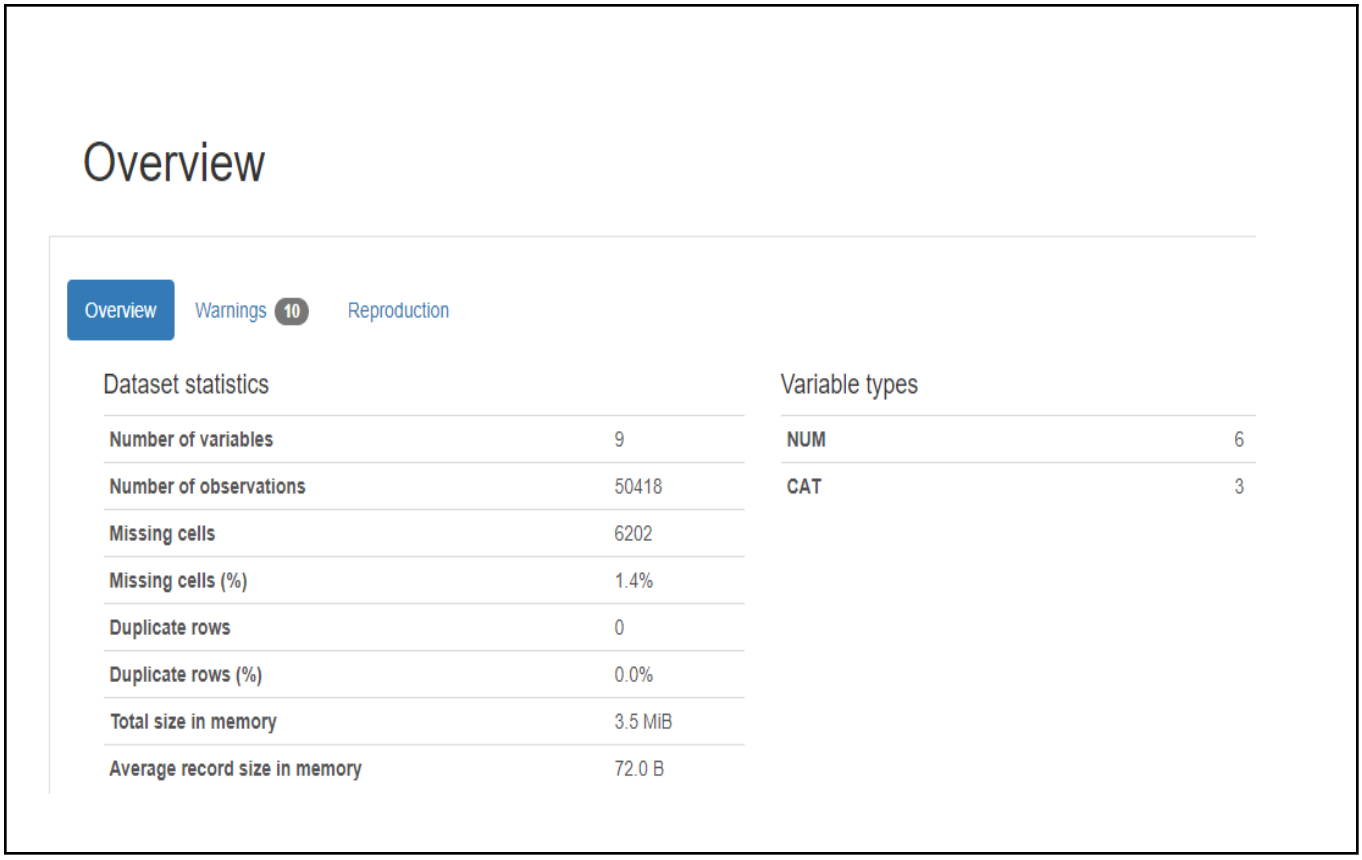


Final Analysis



The second problem in the dataset was that the Date column in the dataframe was a string object. Since we were treating the dataset in general irrespective of whether the column was to be used or not, we also changed the type of Date into "datetime". Also, some of the columns were mislabelled in the original dataset, which were then corrected.

INITIAL ANALYSIS



Overview

Overview | Warnings 10 | Reproduction

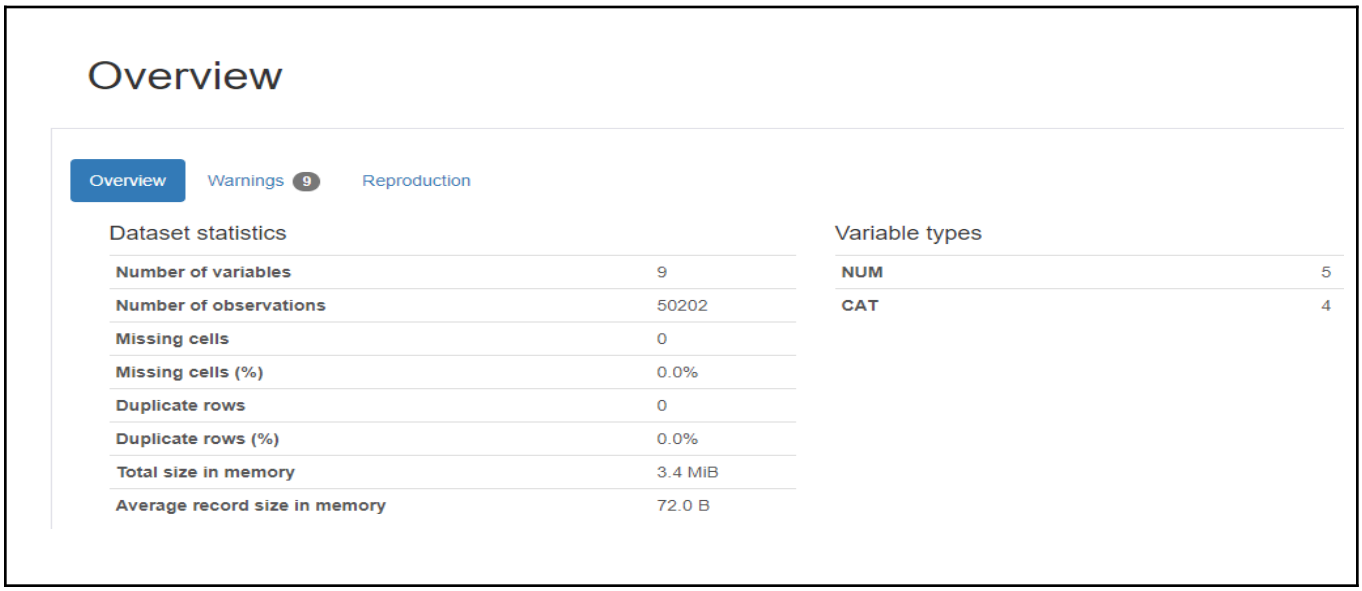
Dataset statistics

Number of variables	9
Number of observations	50418
Missing cells	6202
Missing cells (%)	1.4%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	3.5 MiB
Average record size in memory	72.0 B

Variable types

NUM	6
CAT	3

FINAL ANALYSIS



Overview

Overview | Warnings 9 | Reproduction

Dataset statistics

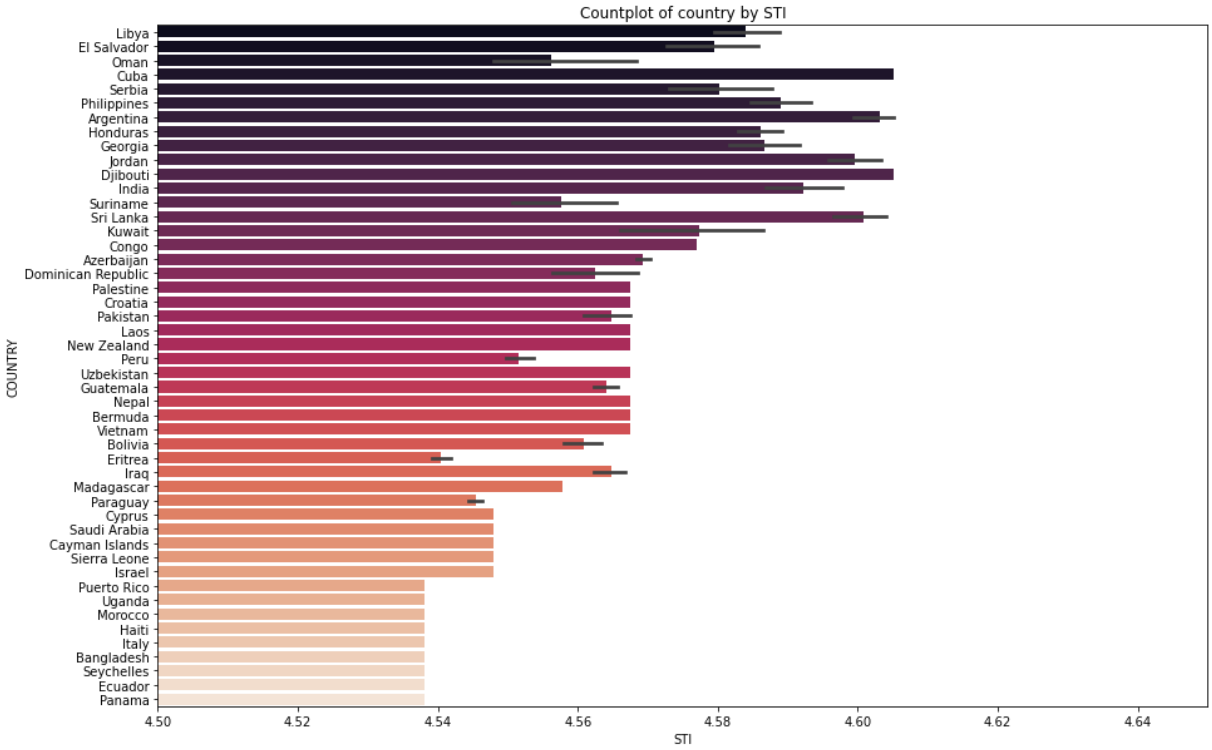
Number of variables	9
Number of observations	50202
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	3.4 MiB
Average record size in memory	72.0 B

Variable types

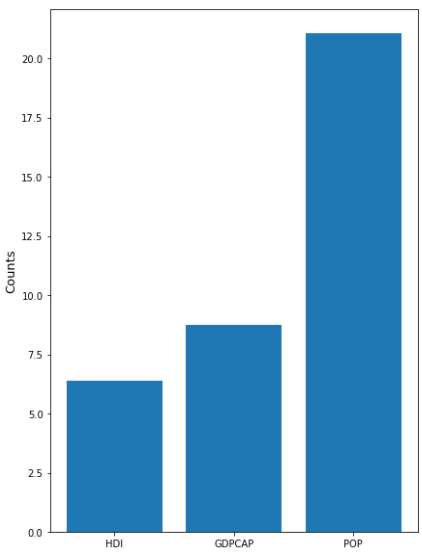
NUM	5
CAT	4

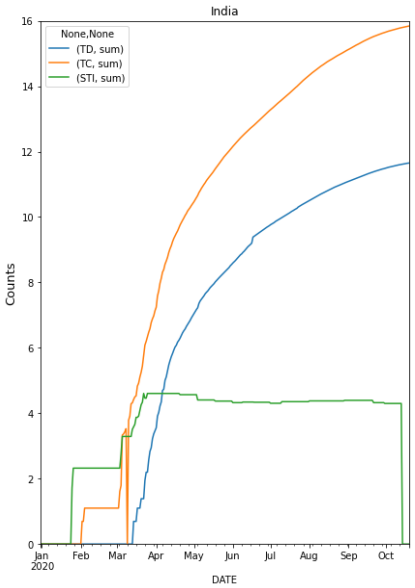
VISUALIZATION:

HUMAN DEVELOPMENT INDEX (HDI)

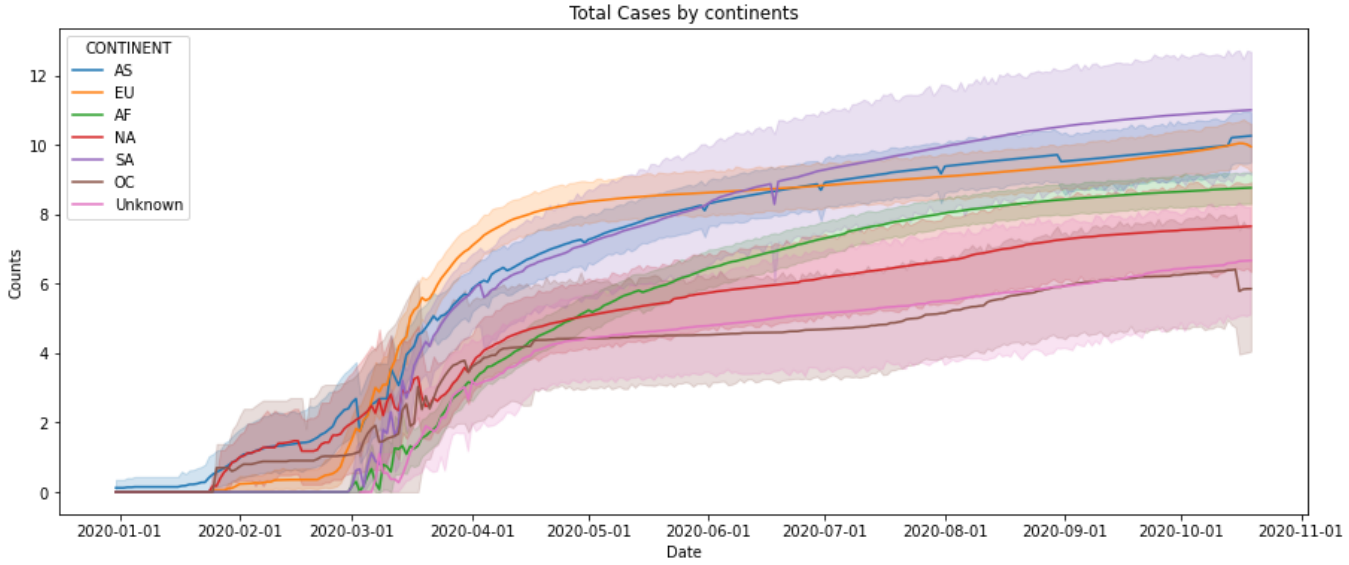


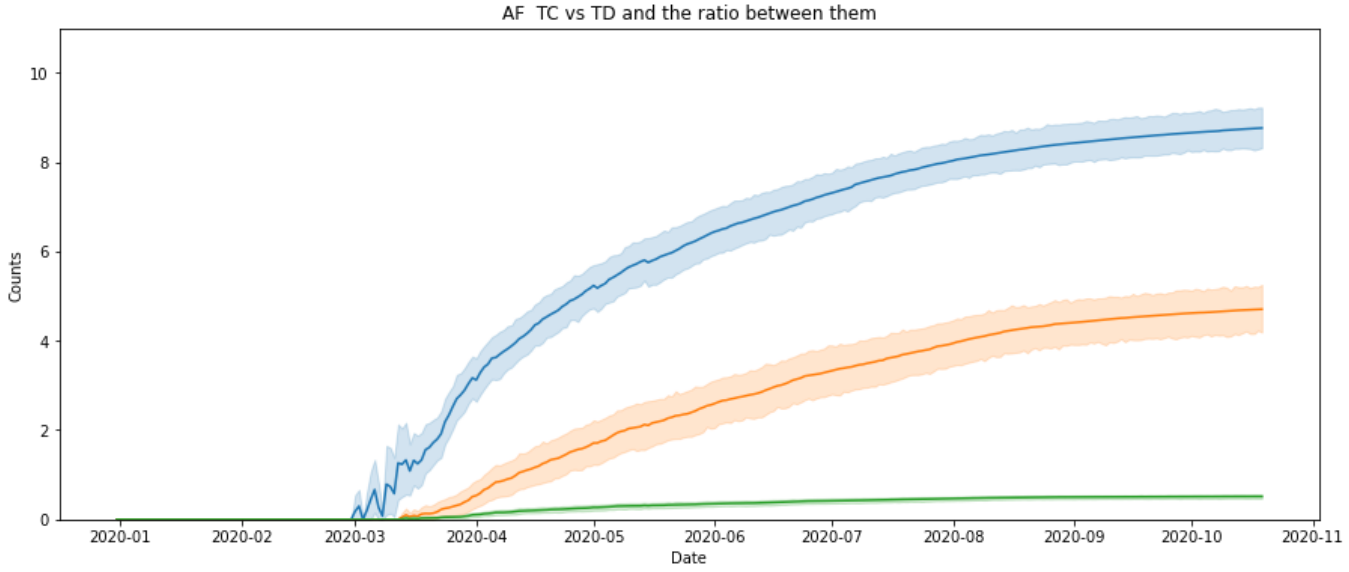
IMPACT OF COVID-19 COUNTRYWISE





IMPACT OF COVID-19 CONTINENT WISE





ALGORITHMS USED:

A) Multiple Linear Regression Model:

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. The population regression line for p explanatory variables $x_1, x_2, \ldots, x_p$ is defined to be $_y = _0 + _1x_1 + _2x_2 + \ldots + _px_p$. This line describes how the mean response $_y$ changes with the explanatory variables. The observed values vary about their means $_y$ and are assumed to have the same standard deviation . The fitted values b_0, b_1, ..., b_p estimate the parameters $_0, _1, \ldots, _p$ of the population regression line. Since the observed values vary about their means $_y$, the multiple regression model includes a term for this variation. In words, the model is expressed as DATA = FIT + RESIDUAL, where the "FIT" term represents the expression $_0 + _1x_1 + _2x_2 + \ldots _px_p$. The "RESIDUAL" term represents the deviations of the observed values y from their means $_y$, which are normally distributed with mean 0 and variance . The notation for the model deviations is .

Formally, the model for multiple linear regression, given n observations, is

$y_i = _0 + _1x_{i1} + _2x_{i2} + \ldots _px_{ip} + _i$ for $i = 1,2, \ldots n$.

So for the implementation of Multiple Linear Regression We have used four variables HDI (Human Development Index), Total Cases, STI and GDP where HDI, Total Cases and STI is an independent variable while GDP is a dependent variable.

Preprocessing

We removed the td and pop column as we don't need those values.Here we collected data from countries like India, USA, Brazil, UK, China, Russia, Sudan and Australia and then concatenated it into dm_conc.

```
dm_new  = dm[dm['CODE'] == 'IND']
dm_new1 = dm[dm['CODE'] == 'USA']
dm_new2 = dm[dm['CODE'] == 'BRA']
dm_new3 = dm[dm['CODE'] == 'GBR']
dm_new4 = dm[dm['CODE'] == 'CHN']
dm_new5 = dm[dm['CODE'] == 'RUS']
dm_new6 = dm[dm['CODE'] == 'SSD']
dm_new7 = dm[dm['CODE'] == 'AUS']
```

```
dm_conc = pd.concat([dm_new, dm_new1, dm_new2, dm_new3, dm_new4, dm_new5, dm_new6, dm_new7])
```

Importing Libraries

Importing Lib

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import sklearn.metrics as met
import matplotlib.pyplot as plt
```

Reading Dataset

```
dm = pd.read_excel("modified.xlsx")
dm.head()
```

	CODE	COUNTRY	DATE	HDI	TC	TD	STI	POP	GDP
0	AFG	Afghanistan	31-12-2019	0.498	0	0	0.0	38928341	1803.987
1	AFG	Afghanistan	2020-01-01 00:00:00	0.498	0	0	0.0	38928341	1803.987
2	AFG	Afghanistan	2020-02-01 00:00:00	0.498	0	0	0.0	38928341	1803.987
3	AFG	Afghanistan	2020-03-01 00:00:00	0.498	0	0	0.0	38928341	1803.987
4	AFG	Afghanistan	2020-04-01 00:00:00	0.498	0	0	0.0	38928341	1803.987

```
dm.tail()
```

	CODE	COUNTRY	DATE	HDI	TC	TD	STI	POP	GDP
50197	ZWE	Zimbabwe	15-10-2020	0.535	8055	231	76.85	14862927	1899.775
50198	ZWE	Zimbabwe	16-10-2020	0.535	8075	231	76.85	14862927	1899.775
50199	ZWE	Zimbabwe	17-10-2020	0.535	8099	231	76.85	14862927	1899.775
50200	ZWE	Zimbabwe	18-10-2020	0.535	8110	231	76.85	14862927	1899.775
50201	ZWE	Zimbabwe	19-10-2020	0.535	8147	231	76.85	14862927	1899.775

```
dm = dm.drop(['TD','POP'], axis = 1)
dm
```

	CODE	COUNTRY	DATE	HDI	TC	STI	GDP
0	AFG	Afghanistan	31-12-2019	0.498	0	0.00	1803.987
1	AFG	Afghanistan	2020-01-01 00:00:00	0.498	0	0.00	1803.987
2	AFG	Afghanistan	2020-02-01 00:00:00	0.498	0	0.00	1803.987
3	AFG	Afghanistan	2020-03-01 00:00:00	0.498	0	0.00	1803.987
4	AFG	Afghanistan	2020-04-01 00:00:00	0.498	0	0.00	1803.987
...	...	...	...	...	...	...	...
50197	ZWE	Zimbabwe	15-10-2020	0.535	8055	76.85	1899.775
50198	ZWE	Zimbabwe	16-10-2020	0.535	8075	76.85	1899.775
50199	ZWE	Zimbabwe	17-10-2020	0.535	8099	76.85	1899.775
50200	ZWE	Zimbabwe	18-10-2020	0.535	8110	76.85	1899.775
50201	ZWE	Zimbabwe	19-10-2020	0.535	8147	76.85	1899.775

50202 rows × 7 columns

dm.describe()

mean	0.737259	6.242276e+04	55.570615	24007.053030
std	0.152764	3.930920e+05	26.894131	26569.467204
min	0.354000	0.000000e+00	0.000000	105.000000
25%	0.640000	6.700000e+01	38.890000	6171.884000
50%	0.768000	1.222000e+03	59.970000	15663.986000
75%	0.863000	1.355750e+04	77.780000	35044.670000
max	0.953000	8.154595e+06	100.000000	180829.020000

dm.isnull().sum()

Plotting correlation matrix

```
import seaborn as sns
corrMatrix=dm.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```

Picking the instances with features in "CODE" as IND(India), USA, BRA(Brazil), GBR(Great Britain), CHN(China), RUS(Russia), SSD(Sudan), AUS(Australia) and storing them in Respective datasets.

```
dm_new  = dm[dm['CODE'] == 'IND']
dm_new1 = dm[dm['CODE'] == 'USA']
dm_new2 = dm[dm['CODE'] == 'BRA']
dm_new3 = dm[dm['CODE'] == 'GBR']
dm_new4 = dm[dm['CODE'] == 'CHN']
dm_new5 = dm[dm['CODE'] == 'RUS']
dm_new6 = dm[dm['CODE'] == 'SSD']
dm_new7 = dm[dm['CODE'] == 'AUS']
```

Concatenating all 7 formed dataset above into one single dataset i.e dm_conc.

```
dm_conc = pd.concat([dm_new, dm_new1, dm_new2, dm_new3, dm_new4, dm_new5, dm_new6,
dm_new7])
dm_conc
```

	CODE	COUNTRY	DATE	HDI	TC	STI	GDP
21212	IND	India	31-12-2019	0.640	0	0.00	6426.674
21213	IND	India	2020-01-01 00:00:00	0.640	0	0.00	6426.674
21214	IND	India	2020-02-01 00:00:00	0.640	0	0.00	6426.674
21215	IND	India	2020-03-01 00:00:00	0.640	0	0.00	6426.674
21216	IND	India	2020-04-01 00:00:00	0.640	0	0.00	6426.674
...	...	...	...	...	...	...	...
2723	AUS	Australia	15-10-2020	0.939	27341	68.06	44648.710
2724	AUS	Australia	16-10-2020	0.939	27362	65.28	44648.710
2725	AUS	Australia	17-10-2020	0.939	27371	65.28	44648.710
2726	AUS	Australia	18-10-2020	0.939	27383	65.28	44648.710
2727	AUS	Australia	19-10-2020	0.939	27390	65.28	44648.710

2255 rows × 7 columns

Initializing x and y variables from the concatenated data .

```
x = dm_conc.iloc[:,[3,4,5]].values
y = dm_conc.iloc[:,-1].values
```

Scaling the data using the Standard Scaler.

```
from sklearn.preprocessing import StandardScaler #Scaling the values
sc = StandardScaler( )
x = sc.fit_transform(x)
print(x)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=9)
```

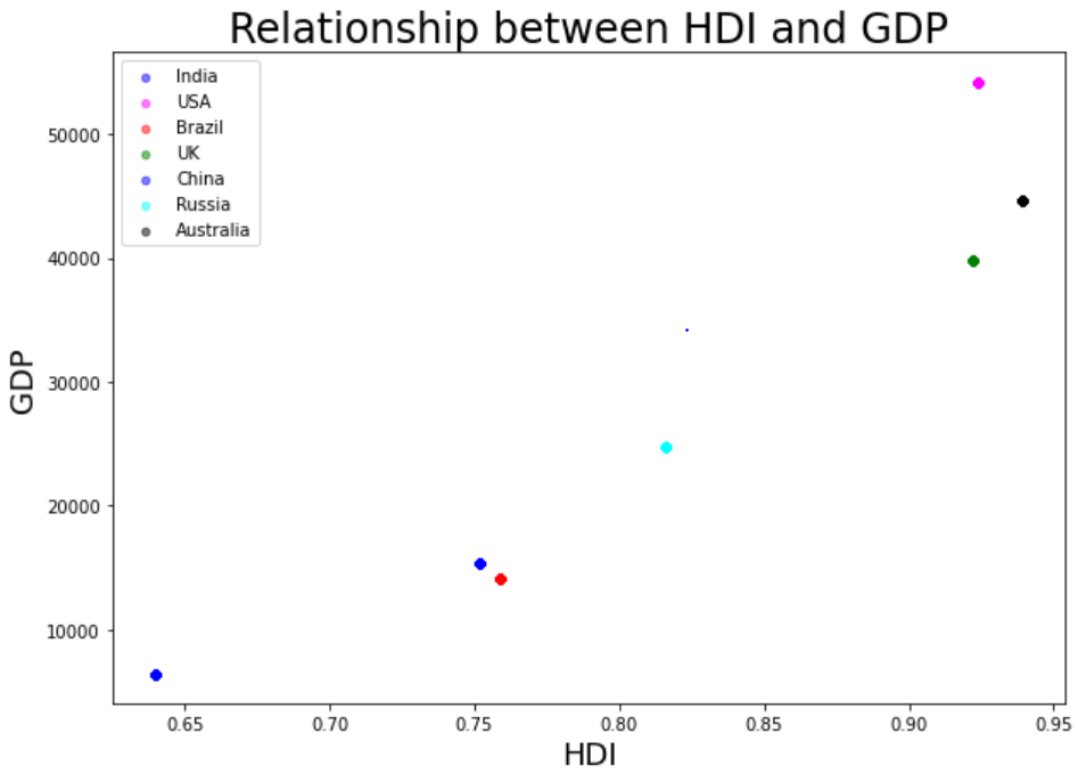
```
[[-0.91104357 -0.48336692 -2.06096684]
 [-0.91104357 -0.48336692 -2.06096684]
 [-0.91104357 -0.48336692 -2.06096684]
 ...
 [ 0.98294692 -0.46676316  0.23399537]
 [ 0.98294692 -0.46675588  0.23399537]
 [ 0.98294692 -0.46675163  0.23399537]]
```

```
lin_reg_mod = LinearRegression()
lin_reg_mod.fit(x_train, y_train)
print(round(lin_reg_mod.score(x_test, y_test)*100,2),"%")
```

76.18 %

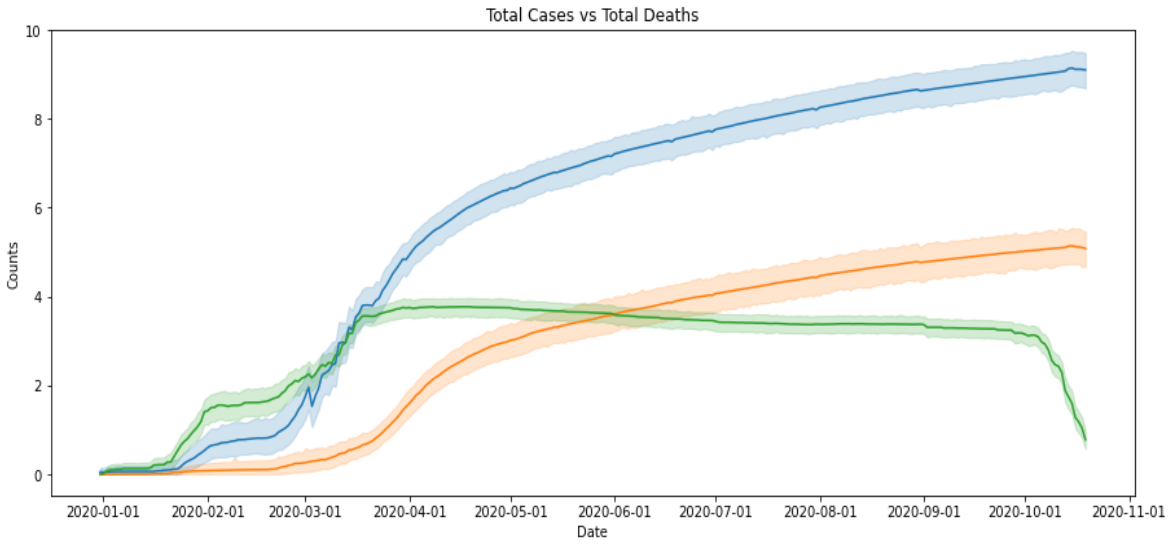
Scatter plot of Height and Weight

```
ax1 = dm[dm['CODE'] == 'IND'].plot(kind='scatter', x='HDI', y='GDP', color='blue', alpha=0.5, figsize=(10, 7))
dm[dm['CODE'] == 'USA'].plot(kind='scatter', x='HDI', y='GDP', color='magenta', alpha=0.5, figsize=(10 ,7), ax=ax1)
dm[dm['CODE'] == 'BRA'].plot(kind='scatter', x='HDI', y='GDP', color='red', alpha=0.5, figsize=(10 ,7), ax=ax1)
dm[dm['CODE'] == 'GBR'].plot(kind='scatter', x='HDI', y='GDP', color='green', alpha=0.5, figsize=(10 ,7), ax=ax1)
dm[dm['CODE'] == 'CHN'].plot(kind='scatter', x='HDI', y='GDP', color='blue', alpha=0.5, figsize=(10 ,7), ax=ax1)
dm[dm['CODE'] == 'RUS'].plot(kind='scatter', x='HDI', y='GDP', color='cyan', alpha=0.5, figsize=(10 ,7), ax=ax1)
dm[dm['CODE'] == 'AUS'].plot(kind='scatter', x='HDI', y='GDP', color='black', alpha=0.5, figsize=(10 ,7), ax=ax1)
plt.legend(labels=['India', 'USA', 'Brazil', 'UK', 'China', 'Russia', 'Australia'])
plt.title('Relationship between HDI and GDP', size=24)
plt.xlabel('HDI', size=18)
plt.ylabel('GDP', size=18);
```



Visualization

Plot between Total Cases and Total Deaths



Testing our Model

In the next step we have tested our model using the test data and the output we have is 76.18 % of the data is being correctly classified and the remaining data which 23.82% is not being classified properly.

Result

```
print(round(lin_reg_mod.score(x_test, y_test)*100,2),"%")
```

```
76.18 %
```

B. Logistic Regression Model

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of the target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

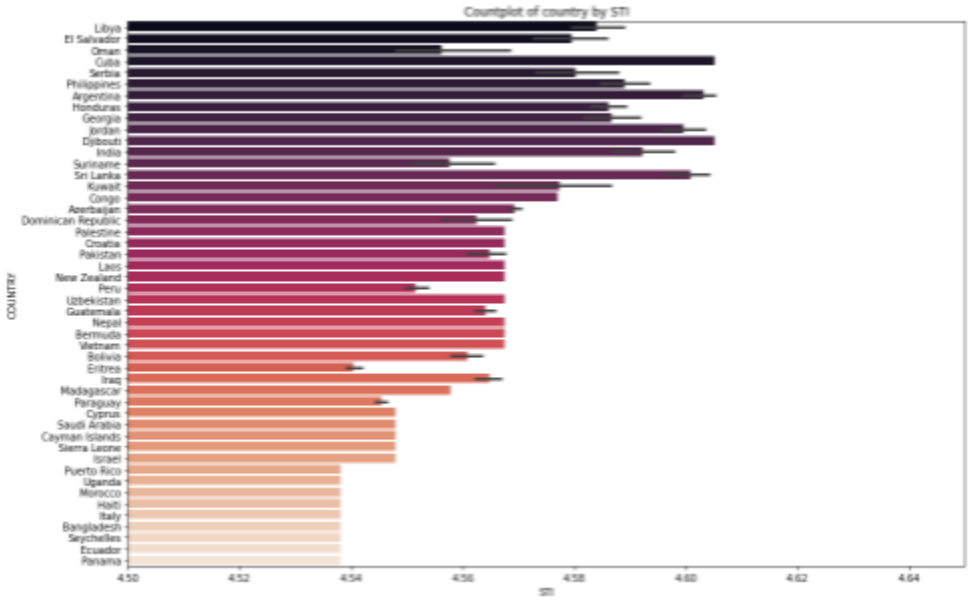
So for the implementation of Logistic Regression We have used two variables: Total Cases and STI where Total Cases is an independent variable while STI is a dependent variable.

What is the Stringency Index?

The Stringency Index is a number from 0 to 100 that reflects these indicators. A higher index score indicates a higher level of stringency. It provides a picture of the stage at which any country enforces its strongest measures.

Example - Some countries saw their deaths just begin to flatten as they reached their highest stringency, such as Italy, Spain, or France. In countries such as the UK, the US, and India, the Oxford graphs find that the death curve has not flattened after the strictest measures.

Comparison of Stringency Index Country Wise



Preprocessing

In order to build our model we have to preprocessed our dataset because the values in STI column ranges from 0 - 100 and for the implementation of logistic regression we need to have binary values in the STI column ie. 0 and 1

So we transformed the STI column into binary values by creating a function called sticonv which assigns 1 for all the values greater than 50 and 0 for below 50.

```
def sticonv(val):
    if val>=50:
        return 1
    else:
        return 0

df["STI"] = df["STI"].apply(sticonv, 1)
```

Implementation

Importing Libraries

```
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
import seaborn as sns
```

Reading Dataset

```
df = pd.read_csv (r'C:\Users\aryan\Desktop\Ar\archive\modified.csv')
df.head()
```

	CODE	COUNTRY	DATE	HDI	TC	TD	STI	POP	GDP
0	AFG	Afghanistan	31-12-2019	0.498	0	0	0.0	38928341	1803.987
1	AFG	Afghanistan	1/1/2020	0.498	0	0	0.0	38928341	1803.987
2	AFG	Afghanistan	2/1/2020	0.498	0	0	0.0	38928341	1803.987
3	AFG	Afghanistan	3/1/2020	0.498	0	0	0.0	38928341	1803.987
4	AFG	Afghanistan	4/1/2020	0.498	0	0	0.0	38928341	1803.987

```
df.tail()
```

	CODE	COUNTRY	DATE	HDI	TC	TD	STI	POP	GDP
50197	ZWE	Zimbabwe	15-10-2020	0.535	8055	231	76.85	14862927	1899.775
50198	ZWE	Zimbabwe	16-10-2020	0.535	8075	231	76.85	14862927	1899.775
50199	ZWE	Zimbabwe	17-10-2020	0.535	8099	231	76.85	14862927	1899.775
50200	ZWE	Zimbabwe	18-10-2020	0.535	8110	231	76.85	14862927	1899.775
50201	ZWE	Zimbabwe	19-10-2020	0.535	8147	231	76.85	14862927	1899.775

STI column values

df["STI"]

```
0          0.00
1          0.00
2          0.00
3          0.00
4          0.00
          ...
50197     76.85
50198     76.85
50199     76.85
50200     76.85
50201     76.85
Name: STI, Length: 50202, dtype: float64
```

df.median(axis = 0)

```
HDI          0.768
TC        1222.000
TD          24.000
STI         59.970
POP    8654618.000
dtype: float64
```

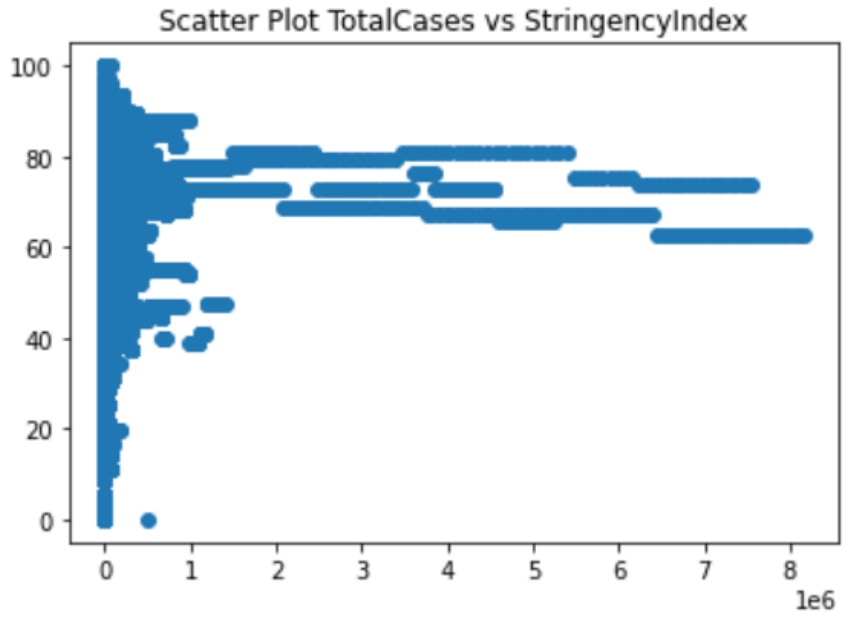
Checking for null values

df.isnull().sum()

```
CODE        0
COUNTRY     0
DATE        0
HDI         0
TC          0
TD          0
STI         0
POP         0
GDP         0
dtype: int64
```

Scatter plot between STI vs Total cases

```
a=df['TC']
b=df['STI']
plt.scatter( a, b, cmap='rainbow')
plt.title('Scatter Plot TotalCases vs StringencyIndex')
plt.show()
```



Function to transform the values to 0 and 1

```
def sticonv(val):
    if val>=50:
        return 1
    else:
        return 0
df["STI"] = df["STI"].apply(sticonv, 1)
```

Overall description of the dataset

```
df.describe()
```

	HDI	TC	TD	STI	POP
count	50202.000000	5.020200e+04	50202.000000	50202.000000	5.020200e+04
mean	0.737259	6.242276e+04	2327.752779	0.637524	4.269823e+07
std	0.152764	3.930920e+05	12292.572860	0.480720	1.567722e+08
min	0.354000	0.000000e+00	0.000000	0.000000	8.090000e+02
25%	0.640000	6.700000e+01	1.000000	0.000000	1.402985e+06
50%	0.768000	1.222000e+03	24.000000	1.000000	8.654618e+06
75%	0.863000	1.355750e+04	280.000000	1.000000	2.982597e+07
max	0.953000	8.154595e+06	219674.000000	1.000000	1.439324e+09

Converting the column values to array

```
arr1 = df["TC"].to_numpy()
arr2 = df["STI"].to_numpy()
```

Assigning the values to values to x and y

```
x,y = arr1,arr2
```

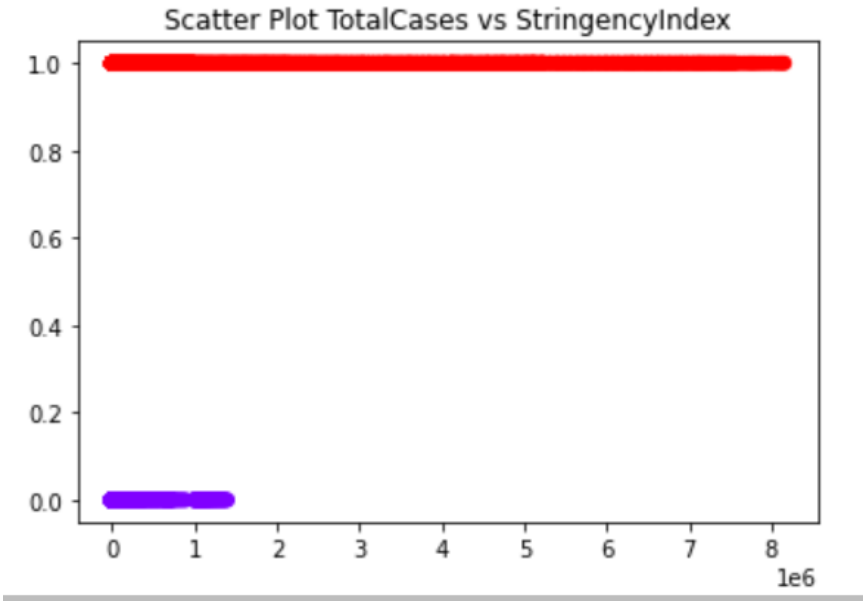
```
print(y)
```

```
[0 0 0 ... 1 1 1]
```

Plotting the scatter plot for the transform values

```
plt.scatter(x, y, c=y, cmap='rainbow')
```

```
plt.title('Scatter Plot TotalCases vs StringencyIndex')
plt.show()
```



Splitting the dataset into test and train and checking its shape

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
x_train = np.reshape(x_train, (37651, 1))
x_test= np.reshape(x_test, (12551, 1))
print(x_train.shape)
print(y_train.shape)
(37651, 1)
(37651,)
```

Creating our model and fitting our data

```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
LogisticRegression()
```

Printing the values of x coefficient and intercept from the logistic function formula

```
print(logreg.coef_)

print(logreg.intercept_)

[[5.828816e-06]]
[1.08758367e-09]
```

Predicting the values for the test data set

```
y_pred = logreg.predict(x_test)
```

Plotting the confusion matrix for our test data

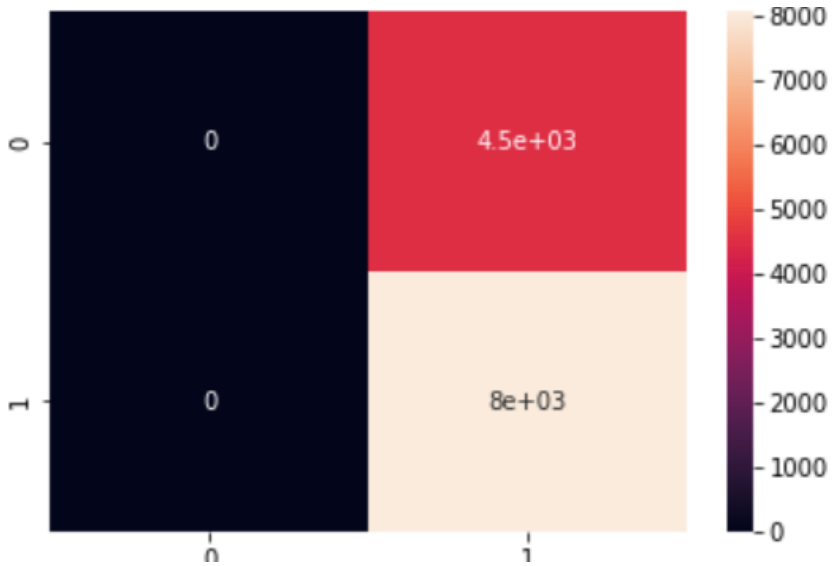
```
cf_matrix=confusion_matrix(y_test, y_pred)

confusion_matrix(y_test, y_pred)

array([[   0, 4507],
       [   0, 8044]], dtype=int64)
```

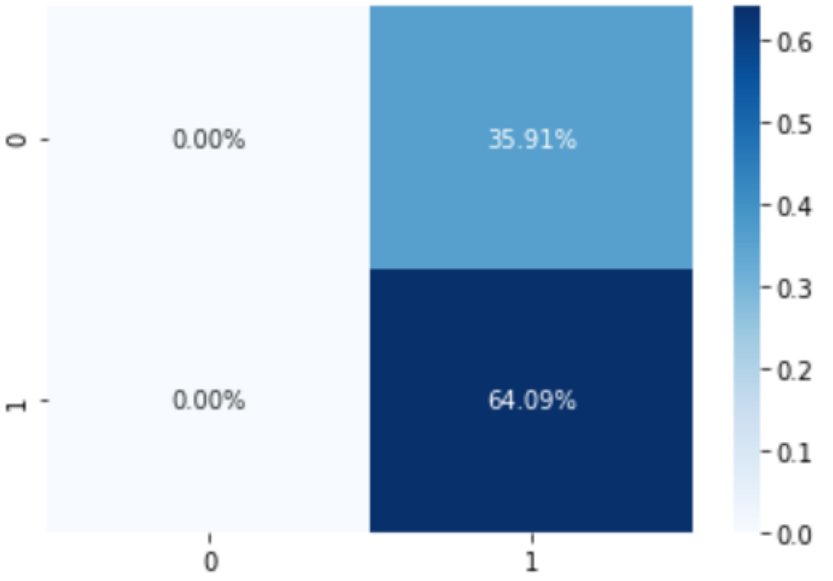
Heat map for confusion matrix

```
sns.heatmap(cf_matrix, annot=True)
```



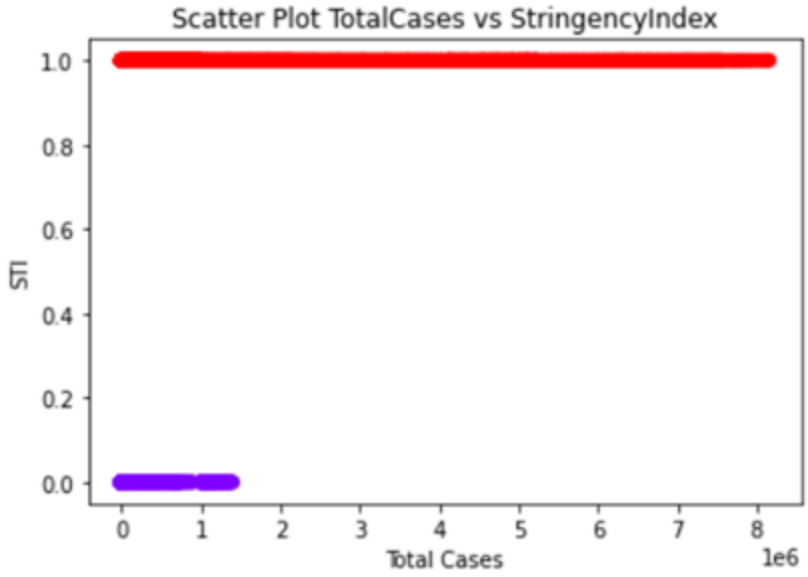

```
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

```
<AxesSubplot:>
```



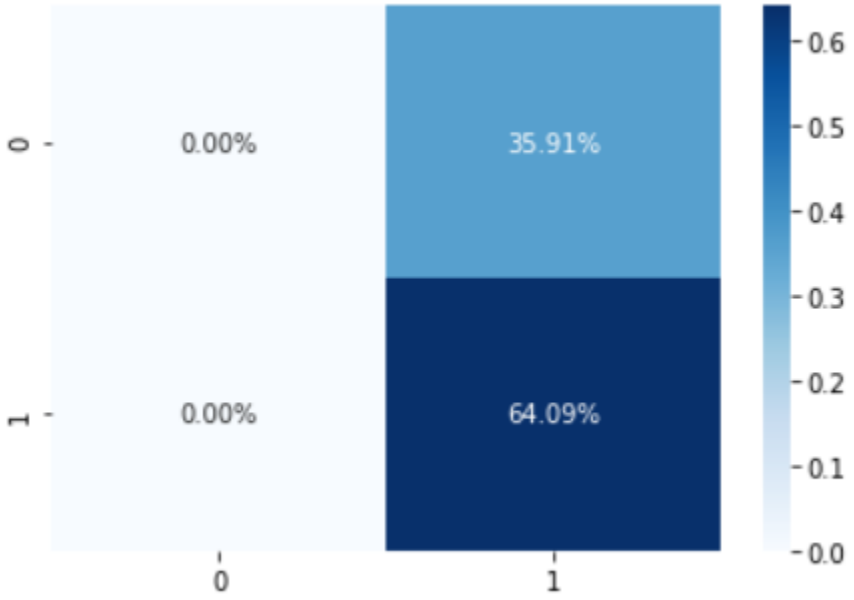
Visualization

To have a look at our data we have further plotted a scatter plot of the total cases on x-axis and STI on y-axis. Where we can clearly see that in our data there are maximum values which correspond to STI value 1. .



Testing our Model

In the next step we have tested our model using the test data and the output we have got is 64.09% of the data is being correctly classified and the remaining data which 35.91% is not being classified properly.



Result

In the final result our model was able to predict the STI for any number of cases given as input.

```
#accuracy obtained
score = logreg.score(x_test, y_test)
print(score)
```

```
0.6409051071627758
```

```
#predicting the sti for any given number of cases
mtest = [[4384]]
prediction = logreg.predict(mtest)
print(prediction)
```

```
[1]
```

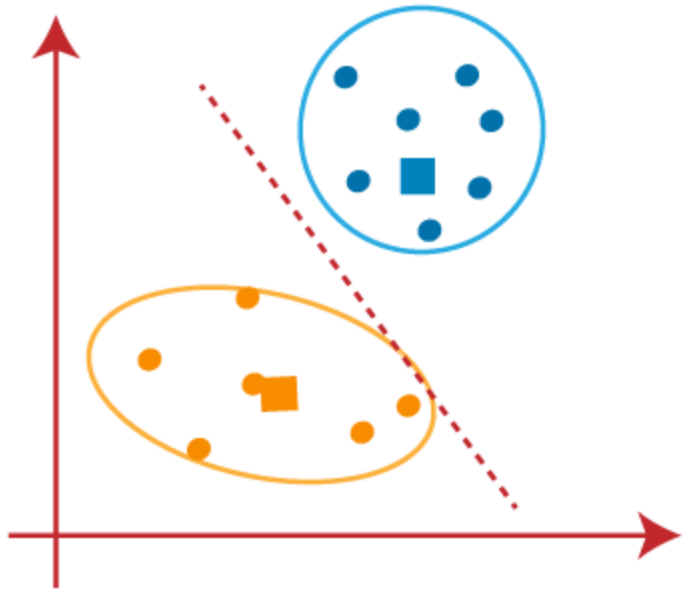
C) K Means

K-Means Clustering is an Unsupervised Learning algorithm that allows us to cluster the data into different groups in the unlabeled dataset on its own without the need for any training, the 'K' referring to the number of clusters. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into a k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



Rationale

- Since supervised learning algorithms didn't yield a very good result on the dataset, we decided to perform some unsupervised computation, here clustering.
- K-means algorithm is a highly efficient clustering algorithm, which was used by us here to cluster the countries in our dataset into 3 non-overlapping categories, least affected, moderately affected, and most affected by the pandemic.
- Since we don't have any past record of how the data was classified, ie, we have unlabelled data, previous training is not possible which is also one of the reasons we decided to go with this algorithm.
- Kmeans, as compared to other clustering algorithms, are flexible, as the number of clusters can be specified explicitly by us, which was what we needed here.
- But in a case where the number of clusters is unknown, there are ways to find out the optimum value of K that will yield the best results. Such a method is the elbow method, where the sharp edge of the graph indicates the K to be taken.
- Since K means algorithm is sensitive to outliers, values having different ranges, and gets biased when highly correlated columns are a part of the input, we tried our best to treat these problems in our

dataset before applying the model, i.e., removed all the outliers we could and scaled all the necessary columns, and analyzed the correlation before choosing the columns.

- We used K Means for implementation instead of K Means as the choosing of centroid points is random in case of Kmeans which could sometimes lead to wrong clustering while KMeans++ does calculated choosing of initial centroid points.

Implementation

Importing Libraries

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
df = pd.read_csv('Modified2.csv')
print(df)
```

```
    CODE      COUNTRY        DATE    HDI  ...    TD    STI       POP        GDP
0    AFG  Afghanistan  19/10/2020  0.498  ...  1492  25.00  38928341   1803.987
1    ALB      Albania  19/10/2020  0.785  ...   451  43.52   2877800  11803.431
2    DZA      Algeria  19/10/2020  0.754  ...  1856  75.93  43851043  13913.839
3    AND      Andorra  19/10/2020  0.858  ...    59  50.00     77265  40886.390
4    AGO       Angola  19/10/2020  0.581  ...   241  71.30  32866268   5819.495
..   ...          ...         ...    ...  ...   ...    ...       ...        ...
204  VEN    Venezuela  19/10/2020  0.761  ...   736  87.96  28435943  16745.022
205  VNM      Vietnam  19/10/2020  0.694  ...    35  51.85  97338583   6171.884
206  YEM        Yemen  19/10/2020  0.452  ...   597  45.37  29825968   1479.147
207  ZMB       Zambia  19/10/2020  0.588  ...   346  44.44  18383956   3689.251
208  ZWE     Zimbabwe  19/10/2020  0.535  ...   231  76.85  14862927   1899.775
```

```
print(df.shape)
```

```
(209, 9)
```

Tukey's method

```
def tukeys_method(df, variable):
    q1 = df[variable].quantile(0.25)
    q3 = df[variable].quantile(0.75)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    outer_fence = 3*iqr
 inner fence lower and upper end
 inner_fence_le = q1-inner_fence
 inner_fence_ue = q3+inner_fence
```

```
#outer fence lower and upper end
   outer_fence_le = q1-outer_fence
   outer_fence_ue = q3+outer_fence
    outliers_prob = []
   outliers_poss = []
   for index, x in enumerate(df[variable]):
       if x <= outer_fence_le or x >= outer_fence_ue:
           outliers_prob.append(index)
   for index, x in enumerate(df[variable]):
       if x <= inner_fence_le or x >= inner_fence_ue:
           outliers_poss.append(index)
   return outliers_prob, outliers_poss
probable_outliers_tm, possible_outliers_tm = tukeys_method(df, "TC")
print(probable_outliers_tm)
print(possible_outliers_tm)
```

```
[7, 15, 27, 40, 42, 67, 72, 89, 90, 91, 92, 96, 124, 149, 150, 156, 163, 174, 177, 193, 198, 199]
[7, 15, 18, 27, 40, 42, 67, 72, 89, 90, 91, 92, 95, 96, 124, 135, 144, 149, 150, 156, 163, 174, 177, 193, 196, 198, 199]
```

```
probable_outliers_tm, possible_outliers_tm = tukeys_method(df, "TD")
print(probable_outliers_tm)
print(possible_outliers_tm)
```

```
[7, 15, 18, 23, 27, 35, 40, 41, 42, 56, 57, 67, 72, 89, 90, 91, 92, 96, 124, 135, 144, 149, 150, 155, 156, 163, 174, 177, 182, 193, 196, 198, 199]
[7, 15, 18, 23, 27, 35, 40, 41, 42, 56, 57, 67, 72, 79, 89, 90, 91, 92, 96, 124, 130, 135, 144, 149, 150, 151, 155, 156, 163, 174, 177, 182, 193, 196, 198, 199]
```

```
probable_outliers_tm, possible_outliers_tm = tukeys_method(df, "HDI")
print(probable_outliers_tm)
print(possible_outliers_tm)

probable_outliers_tm, possible_outliers_tm = tukeys_method(df, "GDP")
print(probable_outliers_tm)
print(possible_outliers_tm)
```

```
[112, 126]
[74, 94, 112, 114, 126, 154, 168]
```

```
df["GDP"].replace({112: df["GDP"].mean, 126: df["GDP"].mean}, inplace=True)
df.isna().sum()
```

```
CODE       0
COUNTRY    0
DATE       0
HDI        0
TC         0
TD         0
STI        0
POP        0
GDP        0
dtype: int64
```

DataFrame.drop(self, labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

```
# df.drop(df.columns[0], axis=1)
# df.drop(columns=['Unnamed: 0'])
df.drop([27,89,199],axis=0,inplace=True)
print(df.shape)
```

(206, 9)

Checking Missing Values

df.isna().sum()

```
CODE       0
COUNTRY    0
DATE       0
HDI        0
TC         0
TD         0
STI        0
POP        0
GDP        0
dtype: int64
```

Extracting required row

Just 2 rows - TC, GDP

```
x = df.iloc[:, [4,8]].values
print(df.iloc[:, [4,8]])
print(x)
type(x)
```

```
        TC        GDP
0    40200   1803.987
1    17055  11803.431
2    54402  13913.839
3     3377  40886.390
4     7462   5819.495
..     ...        ...
204  86636  16745.022
205   1134   6171.884
206   2059   1479.147
207  15853   3689.251
208   8147   1899.775

[206 rows x 2 columns]
```

```
[[4.0200000e+04 1.8039870e+03]
 [1.7055000e+04 1.1803431e+04]
 [5.4402000e+04 1.3913839e+04]
 [3.3770000e+03 4.0886390e+04]
 [7.4620000e+03 5.8194950e+03]
 [3.0000000e+00 2.1068000e+04]
 [1.1900000e+02 2.1490943e+04]
 [9.8966700e+05 1.8933907e+04]
 [6.4694000e+04 8.7875800e+03]
 [4.3220000e+03 3.5973781e+04]
 [2.7390000e+04 4.4648710e+04]
 [6.5557000e+04 4.5436686e+04]
 [4.4964000e+04 1.5847419e+04]
 [5.7030000e+03 2.7717847e+04]
 [7.7902000e+04 4.3290705e+04]
 [3.8856900e+05 3.5239840e+03]
 [2.2200000e+02 1.6978068e+04]
 [8.7698000e+04 1.7167967e+04]
 [2.2212600e+05 4.2658576e+04]
 [2.8130000e+03 7.8243620e+03]
 [2.4960000e+03 2.0642360e+03]
 [1.8500000e+02 5.0669315e+04]
 [3.2700000e+02 8.7085970e+03]
 [1.3977100e+05 6.8858290e+03]
 [1.5000000e+02 2.3500000e+04]
 [3.3561000e+04 1.1713895e+04]
```

```
numpy.ndarray
from sklearn.preprocessing import StandardScaler #Scaling the values
sc = StandardScaler( )
x = sc.fit_transform(x)
print(x)
```

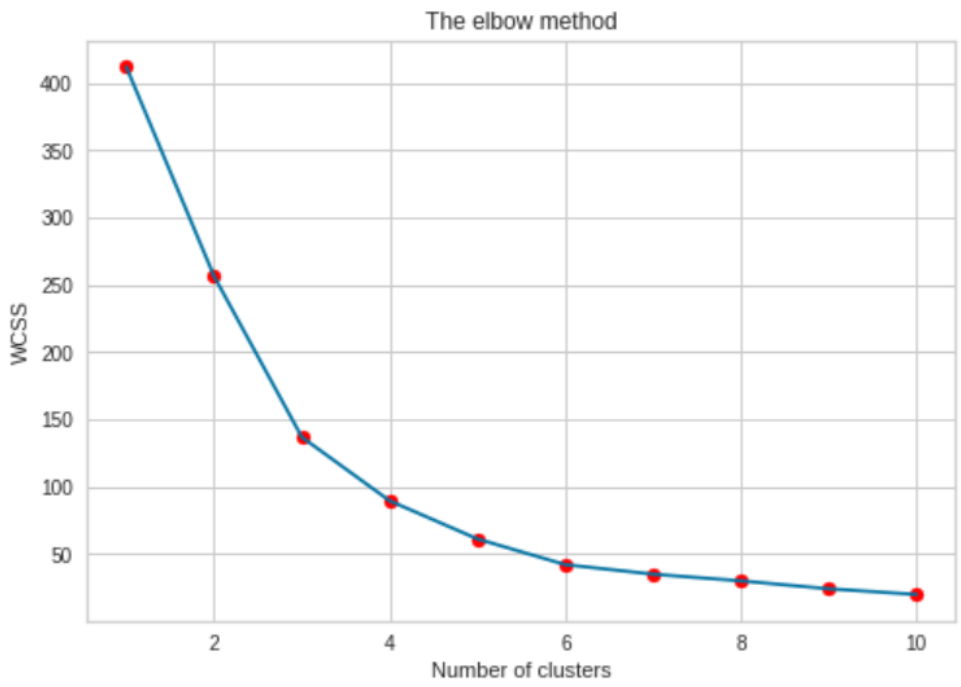
```
[[-2.55292288e-01 -8.04929354e-01]
 [-3.67133252e-01 -4.21344048e-01]
 [-1.86665571e-01 -3.40387397e-01]
 [-4.33227904e-01  6.94297555e-01]
 [-4.13488421e-01 -6.50891803e-01]
 [-4.49531702e-01 -6.59490344e-02]
 [-4.48971169e-01 -4.97246603e-02]
 [ 4.33270964e+00 -1.47814258e-01]
 [-1.36932704e-01 -5.37034093e-01]
 [-4.28661487e-01  5.05846614e-01]
 [-3.17192602e-01  8.38622646e-01]
 [-1.32762527e-01  8.68849928e-01]
 [-2.32271750e-01 -2.66213985e-01]
 [-4.21988237e-01  1.89143508e-01]
 [-7.31091805e-02  7.86528673e-01]
 [ 1.42809180e+00 -7.38949128e-01]
 [-4.48473453e-01 -2.22841539e-01]
 [-2.57730783e-02 -2.15556888e-01]
 [ 6.23808133e-01  7.62279785e-01]
 [-4.35953257e-01 -5.73983775e-01]
 [-4.37485060e-01 -7.94946029e-01]
 [-4.48652244e-01  1.06957705e+00]
 [-4.47966074e-01 -5.40063933e-01]
 [ 2.25853386e-01 -6.09986523e-01]
 [-4.48821371e-01  2.73440991e-02]
 [-2.87373177e-01 -4.24778708e-01]
```

```
from sklearn.cluster import KMeans
wcss = [] # a list to store the wcss values for the different k values
for i in range(1,11): # we will use i to input different k values in the KMeans model
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(x)
```

The 'inertia_' function of the KMeans class is used to get the wcss values

```
wcss.append(kmeans.inertia_)
print(wcss)
x_axis = range(1,11)
plt.scatter(x_axis,wcss,color = 'red')
plt.plot(x_axis,wcss)
plt.title("The elbow method")
plt.xlabel("Number of clusters")
plt.ylabel('WCSS')
plt.show()
```

[411.9999999999998, 256.50592987627283, 136.96106013114235, 89.67158180699059, 61.337918528433434, 42.23541398015536, 35.14976303688289, 30.18588020101427, 24.29016891233743, 20.094374798459782]

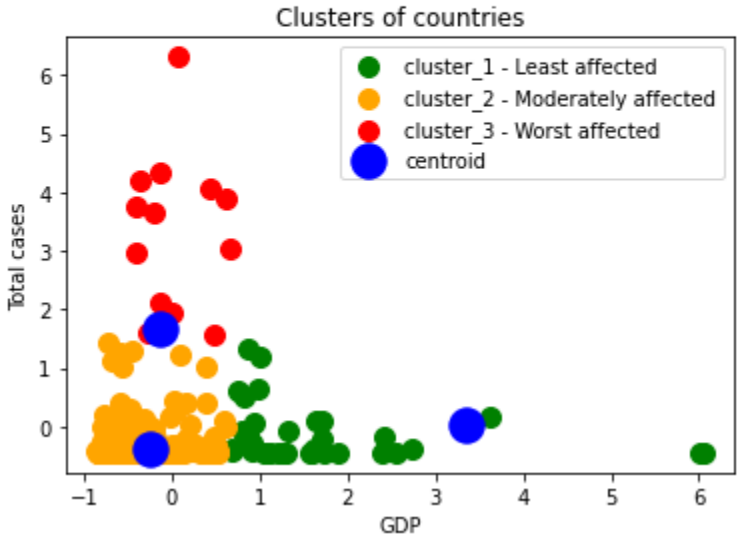


Visualizing the output

```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
y_clusters=  kmeans.fit_predict(x)
print(y_clusters)
#Now for those countries that have a low gdp but hardly any cases, we can consider them to be not much affected, the fh being least affected
#sh, moderately affected
#We have to do this because of the lack of variables that determine the gdp of the country and historical data
plt.scatter(x[y_clusters==0,1], x[y_clusters==0,0], s=100, c='green', label='cluster_1 - Least affected')
plt.scatter(x[y_clusters==1,1], x[y_clusters==1,0], s=100, c='orange', label='cluster_2 - Moderately affected')
plt.scatter(x[y_clusters==2,1], x[y_clusters==2,0], s=100, c='red', label='cluster_3 - Worst affected')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:, 1] , s=300, c='blue', label='centroid')
plt.title('Clusters of countries')
# ax.set_title("Clusters of countries")
plt.xlabel('GDP')
plt.ylabel('Total cases')
plt.legend( )
plt.show() #28, 90, 200
# plt.scatter(X[:,0], X[:,1])
# plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
# plt.show()
```

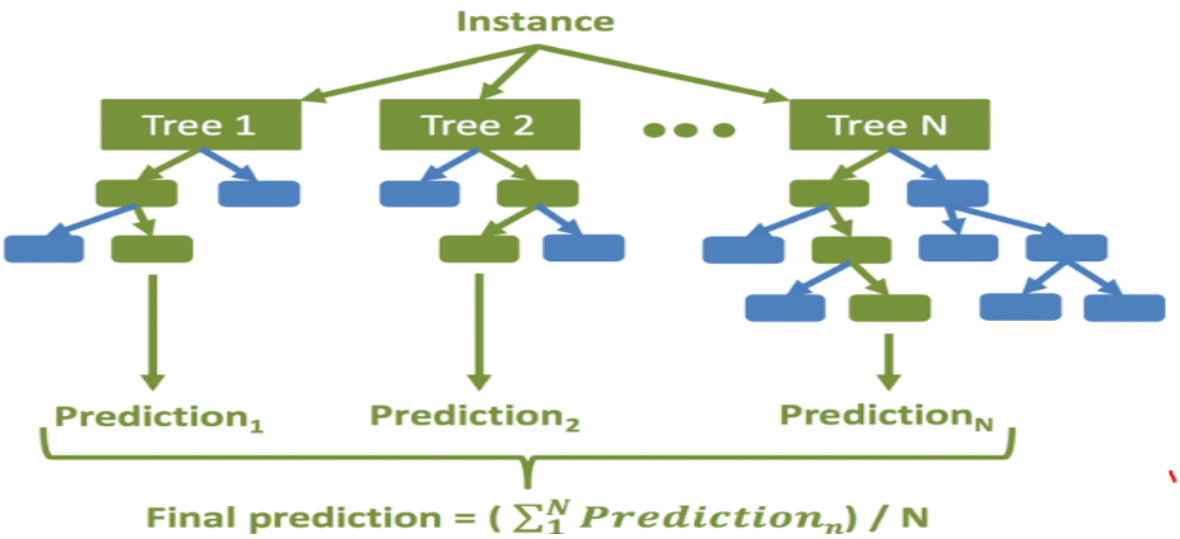
Use plotly module to make plots interactive in google colab - can't be achieved by matplotlib... ax, fig = plt.subplot( )... also... fig = plt.figure( ), ax alone can be a parameter to subplot. This is not package import, something separate

```
[1 1 1 0 1 1 1 2 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0
 1 1 2 1 2 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 2 1 1 1 1 0 1 0
 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 2 2 0 0 1 2 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0
 1 0 1 1 1 1 1 1 1 1 1 2 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 2
 1 1 1 1 0 1 2 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 2 1 1 2 1 1 1 1 0 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 1 1 1]
```



D) Random Forest Regression

Random Forest Regression is an ensemble learning model in which it takes multiple Decision Trees to train and test the model. In RFR it basically splits the dataset in samples and feeds it to the Decision Tree for training the model , suppose if there are 100 Decision Trees then there would be 100 samples of the dataset required for training.While testing when a new output is provided it feeds that output into all the decision trees which were used for training and then takes the mean of outputs given by all the Decision Trees.



Rationale

We used this model because there were many outliers and missing values in our dataset for which we had to alter the dataset continuously.Linear and Logistic Regression models were not able to fit the model properly and give a good accuracy so overfitting was also a main concern while training the dataset.

Random Forest Regression is best known for its ability to handle outliers and missing values, it also removed the issue of overfitting to a great extent and gave a good accuracy, as we were continuously altering the dataset it did not affect the accuracy as in random forest the dataset is splitted and trained on number trees, thus whenever sampling is done in the dataset its very well handled and compensated by the other Decision Trees. Apart from this not much preprocessing was required while training, we had to just handle the missing values and the model was ready to be trained.
Hence due to all these abilities of the model,we decided to implement this model
for our project

Implementation

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import tree
```

Reading Dataset

```
df=pd.read_excel("modified.xlsx")
```

Plotting correlation matrix

```
corrMatrix=df.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```

Plotting scatter plot

```
a=df.iloc[:,3]
plt.scatter(a,y)
plt.ylabel('GDP')
plt.xlabel('HDI')
plt.show()
```

Data Types

```
Out[36]: CODE        object
         COUNTRY     object
         DATE        object
         HDI        float64
         TC           int64
         TD           int64
         STI        float64
         POP          int64
         GDP        float64
         dtype: object
```

Missing values

```
df.isna().sum()
```

```
Out[37]: CODE       0
         COUNTRY    0
         DATE       0
         HDI        0
         TC         0
         TD         0
         STI        0
         POP        0
         GDP        0
         dtype: int64
```

Initializing x and y

```
x=df.iloc[:,3:7]
y=df.iloc[:,8]
```

Splitting dataset for training and testing

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x, y, test_size=0.2,random_state=0)
```

Importing RandomForestRegressor Model

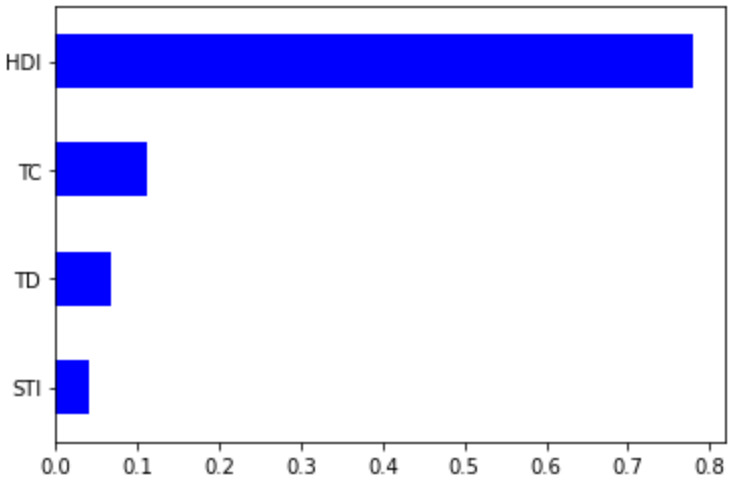
```
from sklearn.ensemble import RandomForestRegressor
RFReg=RandomForestRegressor(n_estimators=100,random_state= 0)
RFReg.fit(x_train,y_train)
```

RFReg.estimators_

```
Out[44]: [DecisionTreeRegressor(max_features='auto', random_state=209652396),
          DecisionTreeRegressor(max_features='auto', random_state=398764591),
          DecisionTreeRegressor(max_features='auto', random_state=924231285),
          DecisionTreeRegressor(max_features='auto', random_state=1478610112),
          DecisionTreeRegressor(max_features='auto', random_state=441365315),
          DecisionTreeRegressor(max_features='auto', random_state=1537364731),
          DecisionTreeRegressor(max_features='auto', random_state=192771779),
          DecisionTreeRegressor(max_features='auto', random_state=1491434855),
          DecisionTreeRegressor(max_features='auto', random_state=1819583497),
          DecisionTreeRegressor(max_features='auto', random_state=530702035),
          DecisionTreeRegressor(max_features='auto', random_state=526610453),
          DecisionTreeRegressor(max_features='auto', random_state=1650985866),
          DecisionTreeRegressor(max_features='auto', random_state=1879422756),
          DecisionTreeRegressor(max_features='auto', random_state=1277981399),
          DecisionTreeRegressor(max_features='auto', random_state=1682652230),
          DecisionTreeRegressor(max_features='auto', random_state=243580376),
          DecisionTreeRegressor(max_features='auto', random_state=1991416408),
          DecisionTreeRegressor(max_features='auto', random_state=1171049868),
          DecisionTreeRegressor(max_features='auto', random_state=1646868794),
          DecisionTreeRegressor(max_features='auto', random_state=2051556033),
          DecisionTreeRegressor(max_features='auto', random_state=1252949478),
          DecisionTreeRegressor(max_features='auto', random_state=1340754471),
          DecisionTreeRegressor(max_features='auto', random_state=124102743),
          DecisionTreeRegressor(max_features='auto', random_state=2061486254),
          DecisionTreeRegressor(max_features='auto', random_state=292249176),
          DecisionTreeRegressor(max_features='auto', random_state=1686997841),
          DecisionTreeRegressor(max_features='auto', random_state=1827923621),
          DecisionTreeRegressor(max_features='auto', random_state=1443447321),
          DecisionTreeRegressor(max_features='auto', random_state=305097549),
          DecisionTreeRegressor(max_features='auto', random_state=1449105480),
          DecisionTreeRegressor(max_features='auto', random_state=374217481),
          DecisionTreeRegressor(max_features='auto', random_state=636393364),
          DecisionTreeRegressor(max_features='auto', random_state=86837363),
          DecisionTreeRegressor(max_features='auto', random_state=1581585360),
          DecisionTreeRegressor(max_features='auto', random_state=1428591347),
          DecisionTreeRegressor(max_features='auto', random_state=1963466437),
          DecisionTreeRegressor(max_features='auto', random_state=1194674174),
          DecisionTreeRegressor(max_features='auto', random_state=502801999),
          DecisionTreeRegressor(max_features='auto', random_state=1589190863),
          DecisionTreeRegressor(max_features='auto', random_state=1589512640),
          DecisionTreeRegressor(max_features='auto', random_state=2055650130),
          DecisionTreeRegressor(max_features='auto', random_state=2034131043),
          DecisionTreeRegressor(max_features='auto', random_state=1284876248),
          DecisionTreeRegressor(max_features='auto', random_state=1292401841),
          DecisionTreeRegressor(max_features='auto', random_state=1982038771),
          DecisionTreeRegressor(max_features='auto', random_state=87950109),
          DecisionTreeRegressor(max_features='auto', random_state=1204863635),
          DecisionTreeRegressor(max_features='auto', random_state=768281747),
          DecisionTreeRegressor(max_features='auto', random_state=507984782),
          DecisionTreeRegressor(max_features='auto', random_state=947610823),
          DecisionTreeRegressor(max_features='auto', random_state=600956192),
          DecisionTreeRegressor(max_features='auto', random_state=352272321),
          DecisionTreeRegressor(max_features='auto', random_state=615697673),
          DecisionTreeRegressor(max_features='auto', random_state=160516793),
          DecisionTreeRegressor(max_features='auto', random_state=1909838463),
          DecisionTreeRegressor(max_features='auto', random_state=1110745632),
          DecisionTreeRegressor(max_features='auto', random_state=93837855),
          DecisionTreeRegressor(max_features='auto', random_state=454869705),
          DecisionTreeRegressor(max_features='auto', random_state=1780959476),
          DecisionTreeRegressor(max_features='auto', random_state=2034098327),
          DecisionTreeRegressor(max_features='auto', random_state=1136257699),
          DecisionTreeRegressor(max_features='auto', random_state=800291326),
          DecisionTreeRegressor(max_features='auto', random_state=1177824715),
          DecisionTreeRegressor(max_features='auto', random_state=1017555826),
          DecisionTreeRegressor(max_features='auto', random_state=1959150775),
          DecisionTreeRegressor(max_features='auto', random_state=930076700),
          DecisionTreeRegressor(max_features='auto', random_state=293921570),
          DecisionTreeRegressor(max_features='auto', random_state=580757632),
```

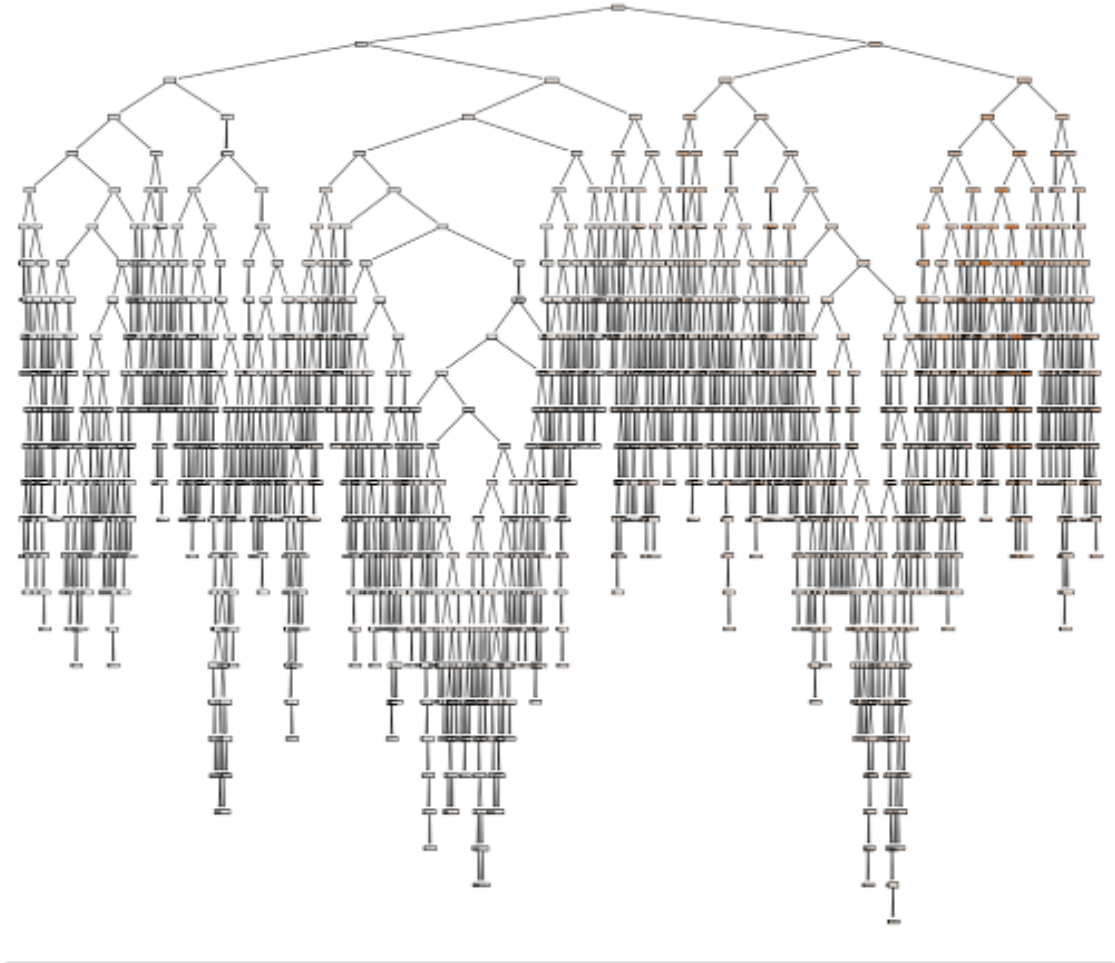
Plotting Feature Importance Graph

```
imp=RFReg.feature_importances_
col=x_train.columns
rfgraph=pd.Series(imp,col)
Rfgraph
 from matplotlib.pyplot import figure
figure(figsize=(10,10))
rfgraph.sort_values().plot.barh(color="blue")
```

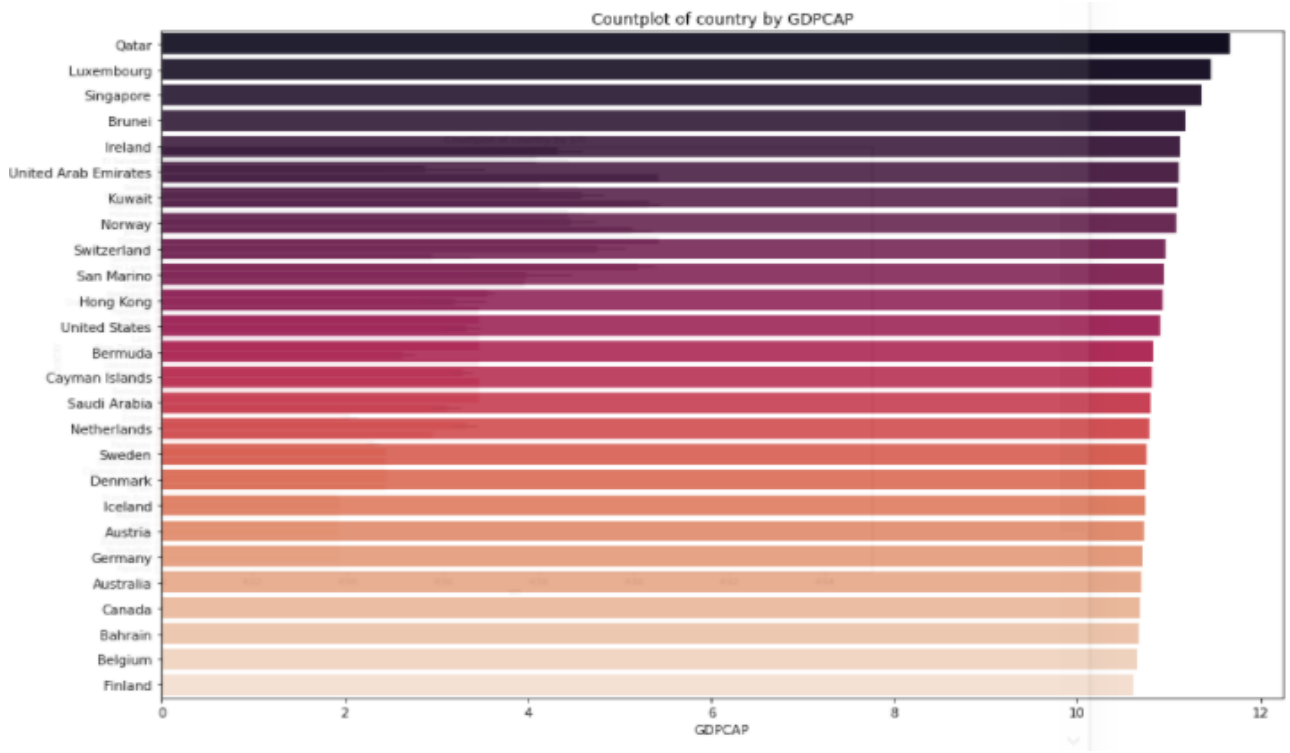


Plotting tree samples for our model

```
plt.figure(figsize=(20,20))
_ = tree.plot_tree(RFReg.estimators_[22],feature_names=col,filled=True)
```



Country wise comparison of GDP CAP



Result

```
y_pred=RFReg.predict((x_test))
from sklearn import metrics
r=metrics.r2_score(y_test,y_pred)
print(r)
```

0.985389549861

CONCLUSION

After analysing and visualising all the models, we can infer that HDI(Human development Index) and STI(Stringency Index) were having significant impact on the economy of a country , whereas Total Cases and Total Deaths were having minimal impact on the economy. It might be because the dataset which we got was from March 2020 to October 2020. As we all know during the first wave of covid number of severities and fatalities in most of the countries were very less, so if we compare Total Deaths and Total Cases(severe) with the population of a particular country then it almost makes a negligible difference, thus Total Deaths and Total Cases had a very minimal impact on the economy of a country.

HDI contains various factors such as life expectancy, literacy, unemployment ratio etc, it is an overall measure of development status of a country, thus it is a very important factor in determining how the economy of a particular country would get affected due to the pandemic situation

STI are the strict measures imposed by government to prevent the spread of Covid-19,when lockdown was imposed a wave of recession was generated leaving many people homeless and jobless thus hampering the economy and was one of the vital reasons for the fall of economy in majority countries.

.