Covid-19 Delhi Prediction

A Project Report

Submitted by:

Sonali Gupta (52255102716)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHONOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

at

MAHAVIR SWAMI INSTITUTE OF TECHNOLOGY

SONEPAT

(AFFILIATED TO GGSIPU DWARKA, NEW DELHI)

(2016-2020)

DECLARATION OF THE STUDENT

I, **Sonali Gupta** ROLLNO **52255102716** hereby declare that the project entitled **"Covid-19 Delhi Prediction"** submitted for the B. Tech. (CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student

Place:

Date:

Certificate of the Guide

This is to certify that the project titled "**Covid-19 Delhi Prediction**" is the bona fide work carried out by **Sonali Gupta**, a student of B Tech (CSE) of **Mahavir Swami Institute of Technology, Sonipat (Haryana)** affiliated to **Guru Gobind Singh Indraprastha University, Delhi(India)** during the academic year 2019-20, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Computer Science and Engineering) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Signature of the Guide

Place:

Date:

Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our final year project manager and the head of the project, **MS SHRUTY AHUJA,** whose contribution in stimulating suggestions and encouragement, helped me to coordinate and whose have invested her full effort in guiding the team in achieving the goal. my project especially in writing this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the staff of **MS NISHA TAYAL**, who gave the permission to use all required equipment and the necessary materials to complete the project "***COVID-19 DELHI PREDICTION***". I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

Abstract

Human history is observing a very strange time fighting an invisible enemy; the novel COVID-19, also known as the Corona Virus. Initially observed in the Wuhan province of China, now rapidly spreading around the world.

Unfortunately, India is recording the Fourth highest number of Covid-19 infected people in Asia. We've been facing this new enemy since January 2020, as the first case of the COVID-19 pandemic in India was reported on 30 January 2020 and we are all fighting every day against all the Economical and Social implications of this virus. Then, the COVID-19 pandemic was confirmed in the Indian union territory of Delhi, with the first case reported on 2 March 2020. Total infected people reported as on 18 May 2020 in Delhi is 10,054 consisting of 160 deaths and 4,485 recovery.

For the safety of the citizens of India, our Prime Minister Narendra Modi, has decided a nationwide lockdown. Experts say that a national lockdown imposed in India to slow the spread of the disease appeared to have been effective in slowing the infection. After lockdown 1.0, 2.0, 3.0, lockdown 4.0 has started from 18 May 2020 and "Delhi to put own guidelines", Arvind Kejriwal (Delhi CM).

Government of India is taking all necessary steps to ensure that we are prepared well to face the challenge and threat posed by the growing pandemic of COVID-19. With active support of the people of India, we have been able to contain the spread of the Virus in our country. The most important factor in preventing the spread of the Virus locally is to empower the citizens with the right information and taking precautions as per the advisories being issued by Ministry of Health & Family Welfare. The Indian government has also introduced an app known as Aarogya Setu App, App to connect health services with the people of India to fight COVID-19, which tracks cases of different states and forming a dataset to perform prediction. On such available data provided by the government, I'll show you a simple mathematical analysis of the infection growth in people using Python and two models to better understand the evolution of the infection, which are the logistic function and the exponential function. The prediction has been done on Data Science using Machine Learning which is based on Supervised learning Technique.

The aim of the project is to predict the expected infection end date (till 18th May 2020) using Time Series models by analysing the data before applying lockdown 4.0 by Indian government which has several relaxations. Analysis included: 1) summary of patient characteristics; 2) examination of infection speed, day with the maximum infections occurred and total number of recorded infected people at the infection's end; 3) calculation of expected infection end date; 4) Time series analysis of viral spread; 5) Logistic Model and Exponential Model curve construction; and 6) Residual analysis.

This situation has opened up doors that we would have never thought to knock on. There is absolutely going to be a treatment. It's not a question of if, but when. And that treatment will most likely come from collaborations between drug discovery researchers and experts in data analytics and AI.

List of Figures

Table of Contents (page no)

1. PROBLEM DEFINITION AND SCOPE OF PROJECT

1.1 Introduction:

The pandemic caused by COVID-19 is the first global public health crisis of the 21st century. And today, multiple AI-powered projects based on data science, ‘Machine Learning‘ or ‘Big Data’, are being used across a broad range of fields to predict, explain and manage the different scenarios caused by the health crisis.

Machine learning and data analytics are going to play a really important role in understanding the spread of disease, as well as understanding the effectiveness of our different responses to disease. AI is being used to support and help those making decisions. No decisions, at any step, are fully and exclusive delegated on the algorithm. AI and data analytics have featured largely in the healthcare industry’s line of defence against COVID-19. Researchers have leveraged these tools to do everything from tracking hospital capacity to identifying high-risk patients, and many believe that these technologies are critical to preparing for similar situations in the future.

1.2 Problem Definition:

Build a machine learning model To predict the total number of infected people till date (i.e. the actually infected people plus the people who have had been infected) and the expected infection end date (till 18th May 2020) using Time Series models by analysing the data before applying lockdown 4.0 by Indian government which has several relaxations.

1.3 Data Set Information:

Our dataset is Classified, technique used is Supervised Learning technique using Time Series Model and the algorithm used Logistic Function and Exponential Function**,** that will be estimated by a curve-fitting calculation on the historical data. This data set is populated by capturing covid19 infection cases from Aarogya Setu Application, Aarogya Setu is a mobile application developed by the Government of India to connect essential health services with the people of India in our combined fight against COVID-19. The App is aimed at augmenting the initiatives of the Government of India, particularly the Department of Health, in proactively reaching out to and informing the users of the app regarding risks, best practices and relevant advisories pertaining to the containment of COVID-19. This information, collected from various sources can help us identify patterns and make critical decisions. Cases from 13 categories across Delhi are considered. We are analysing the data for the predictions from 20 April 2020 to 18 May 2020. Therefore, dataset has 26 Instances (rows) and 13 Attributes (columns).

1.4 Purpose:

COVID-19 is a new disease, one that doctors haven’t seen before and signs of an impending severe case are hard to spot. AI, which can recognize many elusive patterns simultaneously, is the perfect tool to help doctors identify high–risk patients early. This gives them time to better prepare for these cases and could save lives.

While the AI we designed is only a first test, the results are extremely encouraging. We believe AI has a role to play in fighting this pandemic and hope to soon put our system to work helping doctors on the front lines. In a general sense, this type of AI looks at existing data to find patterns and then uses those patterns to make predictions about the future.

The algorithms we designed were trained on a small dataset and at this point are only a proof-of-concept tool, but with more data we believe later versions could be extremely helpful to medical professionals.

1.5 Objective:

Our goals for this study is to see how effective we can be at capturing this kind of data from the general population when everyone is dispersed, and people have to start working from home. In the long term, this survey can tell us whether these direct data collection methods can be useful in a situation like a pandemic or disease outbreak.

In addition to real-world data, researchers have increasingly turned to open, publicly available datasets to ensure they’re accessing quality COVID-19 information. Using these datasets, teams can develop artificial intelligence and Machine Learning algorithms to better understand the virus and its impact.

Machine Learning and Data Science helps to extrapolate from existing data to predict the progression of an infectious disease outbreak, have come to play an integral role in infectious disease epidemiology.
Such modelling helps one understand the trajectory of a disease over time, how fast it is increasing and what might determine that increase.

1.6 Project Scope:

"Machine Learning and Artificial Intelligence algorithms allows us to diagnose and customize medical care and follow-up plans to get better results."

In recent years data science has become one of the most promising technologies bringing changes to various industries. New ways of collecting, sharing, and evaluating data will hopefully extend into life after COVID-19, leading to new technological advancements that have previously eluded the industry.

With better data and faster analysis, researchers could transform patient care and prepare for any future emergencies. Researchers believe in the old adage that necessity becomes the mother of invention. This pandemic has been a catalyst for transformation.

We did not know how to face this crisis, but the tools that we're developing could be very helpful for future drug discoveries, future pandemics, or other diseases that don't generate much pharmaceutical interest because they don't impact people in developed countries.

Now, with this model and what we will develop in the future, anybody can use it for whatever target they want. The outbreak could also revolutionize the process of vaccine development. This will transform the way we're discovering drugs going forward.

1.7 Technologies To be used(Front end & Back end):

1.7.1 Front end Technology:

1. OS: Windows 10
2. Programming language: Python, Data Science, Machine Learning
3. Software: MS Excel, Anaconda

1.7.2 Backend Technology:

1. Dataset: CSV format

2. LITERATURE SURVEY

2.1 Existing System

In the past, some software systems and algorithms may have fallen short because they didn't have the refined content that they needed to find actionable insights or drive informed decision-making. The algorithms are the algorithms, but having that quality underlying data is what makes them work.

India currently has the fourth largest number of confirmed cases in Asia with number of cases breaching the 100,000 mark on 18 May 2020. The highest single day surge in new cases was recorded on 17 May 2020, when 5,049 cases were reported. India's case fatality rate is relatively lower at 3.09%, against the global 6.63% as of 18 May 2020. As of 18 May 2020, the Ministry of Health and Family Welfare have confirmed a total of 10,054 cases, 4,485 recoveries and 160 deaths in the Delhi.

On 22 March 2020, India observed a 14-hour voluntary public curfew at the instance of the prime minister Narendra Modi. The government followed it up with lockdowns in 75 districts where COVID-19 cases had occurred as well as all major cities. Further, on 24 March, the prime minister ordered a nationwide lockdown for 21 days, affecting the entire 1.3 billion population of India. On 14 April, the prime minister extended the ongoing nationwide lockdown till 3 May. On 1 May, lockdown across the country was further extended by two more weeks till 17 May.

India was quick to close its international borders and enforce an immediate lockdown, which WHO praised as “tough and timely”. The lockdown has also given the government time to prepare for a possible surge in cases when the pandemic is forecasted to peak in the coming weeks. Still, India's population of 1·3 billion across diverse states, health inequalities, widening economic and social disparities, and distinct cultural values present unique challenges.

Preparedness and response to COVID-19 have differed at the state level.

Government recently released an App known Aarogya Setu to support research, analytics, and machine learning applications. The dataset supports the government’s recent call to action for experts to develop AI solutions in response to COVID-19.

Using an Aarogya Setu app, the researchers are able to target billions of drug compounds against COVID-19 proteins in a matter of days, drastically reducing the time it takes to analyze possible treatments. This helps us get to the finish line in a much shorter amount of time, which we would have never imagined.

If three or more patients are diagnosed, all houses within 3 km are surveyed to detect further cases, trace contacts, and raise awareness. Whether this strategy will be successful is still unclear. The premise relies on there not being community transmission, and there is danger of stigmatisation and coercion. But states deserve much of the credit for India's COVID-19 response.

As government begin to open up the economy and relax social distancing measures, we need to have a better sense of the underlying prevalence of the virus, because we think that many people who have this disease are asymptomatic or have very mild symptoms.

Going forward, there will be a much greater emphasis on quality information and quality data. Because that will accelerate research not just in the drug discovery space, but in other spaces, too. And COVID-19 is going to prove that in spades.

2.2 Proposed System:

In the months since COVID-19 has evolved from a blip on the world’s radar to a full-blown global health crisis, the virus has managed to shine a glaring light on some of healthcare’s most foundational cracks.

As the India continues to monitor the spread of coronavirus and the country starts to think about relaxing social distancing measures, healthcare leaders are examining their abilities to mitigate the impact of this outbreak now and going forward.

For many organizations, this will mean implementing or enhancing artificial intelligence and data analytics in healthcare.

Although the need for quality, timely data is always a priority in healthcare, the ever-changing coronavirus outbreak has made data accuracy even more necessary.

Everything about COVID-19 is still in the early stages. There's so much new data coming in and so much information evolving. A great deal of research is required to understand how we can best address the disease and use technology to improve our response, as well as help develop future therapeutics.

With the healthcare system encouraging people to stay home if they can manage their illness on their own. The project allows people to report on their symptoms from their mobile device or web browser, providing researchers with a comprehensive picture of disease trends and hotspots.

What government hoping to do is capture enough data from different geographic locations so that we can start to see patterns in symptoms. If we can see the clusters of symptoms going down over time, then we know that people are adhering to social distancing measures, and we can start to move forward with reopening the economy.

This situation has opened up doors that we would have never thought to knock on.

There is absolutely going to be a treatment. It's not a question of if, but when. And that treatment will most likely come from collaborations between drug discovery researchers and experts in data analytics and AI.

This information is freely accessible online and generates so-called “open data or dataset”. This data set, csv format named covid19_Delhi_Data.csv, is populated by capturing covid19 infection cases from Aarogya Setu Application. Cases from 13 categories across Delhi are considered. We are analysing the data for the predictions from 20 April 2020 to 18 May 2020. Therefore, dataset has 26 Instances (rows) and 13 Attributes (columns).

Different techniques inherited from the wide field of Intelligent Data Analysis like Time Series Model which have been crawled by Jupyter Notebook (Anaconda). Overall, this dataset deals with 'Date', 'Total Confirmed Cases', 'Total Active Cases', 'Total Recovered Cases', 'Total Deceased Cases', 'Total Tests Cases', 'Daily Confirmed Cases', 'Daily Active Cases', 'Daily Recovered Cases', 'Daily Deceased Cases', 'Daily Tests Cases', 'Doubling Time', 'Growth Rate'.

2.2.1 Libraries Used:

a) **Pandas:** Data manipulation and analysis
b) **Matplotlib:** Comprehensive 2D/3D plotting
c) **Numpy:** Used for collection of high-level mathematical functions
d) **Seaborn:** Data visualization library based on matplotlib
e) **Sckikit-Learn:** Provides a range of supervised and unsupervised learning algorithms
f) **Scipy:** builds on the NumPy array object and is part of the NumPy stack
g) **Datetime**: Supplies classes for manipulating dates and times

To this end, our study contributes to the process of predicting covid-19 pandemic future preferences via software that analyzes a large set of the dataset (i.e. covid19_Delhi_Data) that is freely available. This is devised by generating the classification function and the best model for predicting the destination tourists would potentially select.

2.3 Feasibility Study:

- Technical Feasibility
- Social and Operational Feasibility
- Time Feasibility

3. Methodology/Planning of Project

To carry out the study, the following methodology is applied: a preliminary statistical analysis is performed to acquire general knowledge about the datasets, such as the geographical distribution of cases, their activities, and a comparison among the Logistic Model and Exponential Model, and Residual analysis.

We're going to see a lot of effect of data analytics in real-time. This kind of research is highlighting the fact that there are a lot of new things that we can do to make data analytics more easily repeatable and specific to healthcare use cases. While coronavirus has accentuated the promise of advanced analytics tools, the pandemic has also revealed the relative immaturity of the technology. Issues around data access, sharing, and quality still impact the accuracy of algorithms, as well as the ability to develop algorithms in the first place.

With COVID-19 disrupting the standard state of affairs, the stage is set for major industry players to come together and improve their data analytics capabilities. Experts and leaders from all sectors are working to overcome healthcare's long-standing data challenges, preparing the way for artificial intelligence to take on a larger role in patient care.

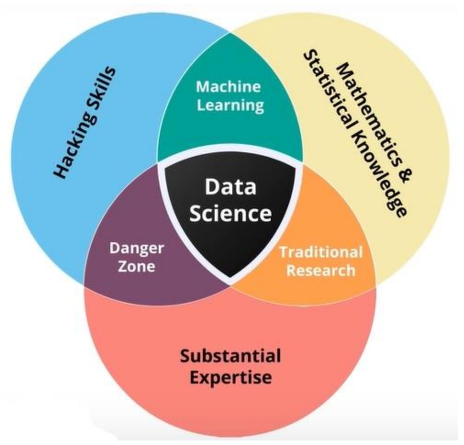


Fig: 1

3.1 Brief Description of Algorithms Used:

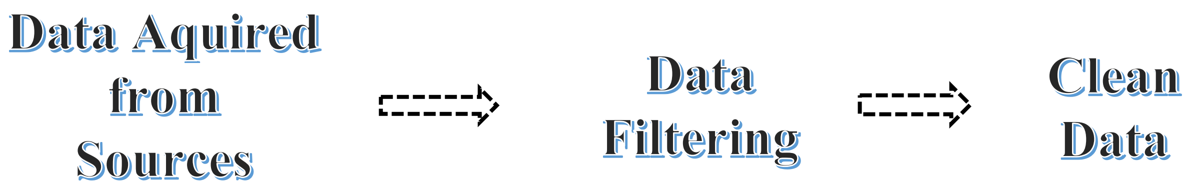
3.1.1 Data Science: is the area of study which involves extracting insights from various vast amounts of data by the use of various scientific methods, algorithms and processes helps to discover hidden pattern from the raw data. It is the process of using data to find solution to predict outcome for a problem statement. It is also known as Data-Driven Science.

3.1.2 Machine Learning: It's a class of algorithm which is data- driven, i.e. unlike "Normal" algorithm. It is the data that "Tells" what the "Good Answer" is. It's the application of Artificial Intelligence that provides systems, the ability to automatically learn and improve from experience without being implicitly programmed.

Getting computers to program themselves and also teaching them to make decision using data "where writing software is the bottleneck, let the data do the work instead."

3.1.3 Machine Learning Life Cycle:

Step-1: <u>Collecting Data:</u> Data is collected from various sources in a server.
Step-2: <u>Data Wrangling:</u> It is a process of cleaning and converting the raw data into a format that allows convenient consumption.



Step-3: <u>Analyse Data:</u> Data is analysed to select and filter data required to prepare the model. In this, we take data, use Machine Learning Algorithms to create a particular model.
Step-4: **<u>Train Algorithm</u>**: Here, we are training the model. Here, we use the dataset and Algorithm is trained on training dataset through which the Algorithm understand the pattern and rules which governs the particular data.
Step-5: <u>Test Algorithm:</u> Testing dataset determines the accuracy of the model and tells us the accuracy of the model.
Step-6: <u>Operation and Optimisation:</u> If the speed and accuracy of the model is acceptable, then that model should be deployed in the real system. The model i.e. used in the available data.
Models improve with the amount of available data is used to create the data. The result of the data needs to incorporated in the business strategy. After the model is deployed based upon the performance, the model is updated and improved. If there is a drip in the performance, the model is retrained.

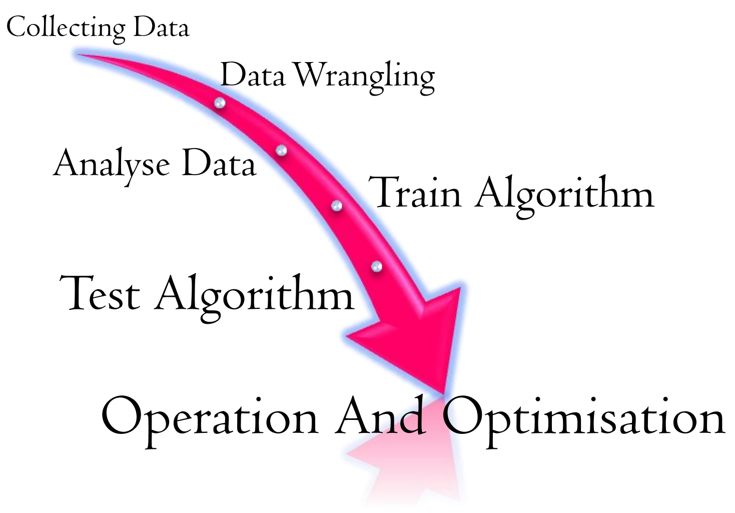


Fig: 2

3.1.4 Supervised learning: It is a type of machine learning algorithm used to draw influences from data sets consisting of input data with labelled response. The data is structured and labelled. It is a type of machine learning algorithm used to draw influences from data sets consisting of input data with labelled response. The data is structured and labelled. Its where you have input variable(x) and an output variable(y) and you use an algorithm to learn the mapping function from the input to the output. It's called so because the process of an algorithm learning from the training dataset can be thought as a teacher supervising the learning process.

3.1.5 Logistic Model:

The logistic model has been widely used to describe the growth of a population. An infection can be described as the growth of the population of a pathogen agent, so a logistic model seems reasonable.

This formula is very known among data scientists because it's used in the logistic regression classifier and as an activation function of neural networks.

The most generic expression of a logistic function is:

$$f(x,a,b,c) = \frac{c}{1+e^{-(x-b)/a}}$$

In this formula, we have the variable *x* that is the time and three parameters: *a, b, c*.

- *a* refers to the infection speed
- *b* is the day with the maximum infections occurred
- *c* is the total number of recorded infected people at the infection's end

At high time values, the number of infected people gets closer and closer to *c* and that's the point at which we can say that the infection has ended. This function has also an inflection point at *b*, that is the point at which the first derivative starts to decrease (i.e. the peak after which the infection starts to become less aggressive and decreases).

- ***curve_fit* function** of *scipy* library: To estimate the parameter values and errors starting from the original data.
- ***fsolve* function** of *scipy:* To numerically find the root of the equation that defines the infection end day.

3.1.6 Exponential Model:

While the logistic model describes an infection growth that is going to stop in the future, the exponential model describes an unstoppable infection growth. For example, if a patient infects 2 patients per day, after 1 day we'll have 2 infections, 4 after 2 days, 8 after 3 and so on.

The most generic exponential function is:

$$f(x, a, b, c) = a \cdot e^{b(x-c)}$$

The variable *x* is the time and we still have the parameters *a*, *b, c*. The meaning, however, is different from the logistic function parameters.

3.1.7 Analysis of Residuals:

Residuals are the differences between each experimental point and the corresponding theoretical point. We can analyse the residuals of both models in order to verify the best fitting curve. In a first approximation, the lower Mean Squared Error between theoretical and experimental data, the better the fit.

Source Code

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
from datetime import datetime, timedelta
from sklearn.metrics import mean_squared_error
from scipy.optimize import fsolve
from scipy.optimize import curve_fit
from scipy.stats import norm
from scipy import stats
import scipy.optimize as optimize
```

Import Dataset

In [2]:
```
# import csv files
read_df = pd.read_csv('covid19_Delhi_Data_final.csv')
```

In [3]:
```
# print the data set with number of rows and columns
read_df
```

Out[3]:

	Date	Total Confirmed Cases	Total Active Cases	Total Recovered Cases	Total Deceased Cases	Total Tests Cases	Daily Confirmed Cases	Daily Active Cases	Daily Recovered Cases	De
0	20-04-2020	2081	1603	431	47	25900.0	78	-65	141	
1	21-04-2020	2156	1498	611	47	26627.0	75	-105	180	
2	22-04-2020	2248	1476	724	48	28309.0	92	-22	113	
3	23-04-2020	2376	1518	808	50	30560.0	128	42	84	
4	24-04-2020	2514	1604	857	53	33672.0	138	86	49	
5	25-04-2020	2625	1702	869	54	35519.0	111	98	12	
6	26-04-2020	2918	1987	877	54	37613.0	293	285	8	
7	27-04-2020	3108	2177	877	54	39911.0	190	190	0	
8	28-04-2020	3314	2182	1078	54	43370.0	206	5	201	
9	29-04-2020	3439	2291	1092	56	47225.0	125	109	14	
10	30-04-2020	3515	2362	1094	59	NaN	76	71	2	
11	01-05-2020	3738	2510	1167	61	NaN	223	108	73	
12	02-05-2020	4122	2802	1256	64	58210.0	384	292	89	
13	03-05-2020	4549	3123	1362	64	60246.0	427	321	106	
14	04-05-2020	4898	3403	1431	64	64108.0	349	280	69	
15	05-05-2020	5104	3572	1468	64	67852.0	206	169	37	
16	06-05-2020	5532	3925	1542	65	71934.0	428	353	74	
17	08-05-2020	6542	4454	2020	68	81367.0	338	247	89	

	Date	Total Confirmed Cases	Total Active Cases	Total Recovered Cases	Total Deceased Cases	Total Tests Cases	Daily Confirmed Cases	Daily Active Cases	Daily Recovered Cases	De
18	09-05-2020	6542	4454	2020	68	84226.0	224	224	0	
19	12-05-2020	7639	5041	2512	86	106109.0	406	10	383	
20	13-05-2020	7998	5034	2858	106	113345.0	359	-7	346	
21	14-05-2020	8470	5310	3045	115	119736.0	472	276	187	
22	15-05-2020	8895	5254	3518	123	125189.0	425	-56	473	
23	16-05-2020	9333	5278	3926	129	130845.0	438	24	408	
24	17-05-2020	9755	5405	4202	148	135791.0	422	127	276	
25	18-05-2020	10054	5409	4485	160	139727.0	299	4	283	

Data Cleaning

In [4]:
```
# head() defines read first 5 data
read_df.head()
```

Out[4]:

	Date	Total Confirmed Cases	Total Active Cases	Total Recovered Cases	Total Deceased Cases	Total Tests Cases	Daily Confirmed Cases	Daily Active Cases	Daily Recovered Cases	Dece C
0	20-04-2020	2081	1603	431	47	25900.0	78	-65	141	
1	21-04-2020	2156	1498	611	47	26627.0	75	-105	180	
2	22-04-2020	2248	1476	724	48	28309.0	92	-22	113	
3	23-04-2020	2376	1518	808	50	30560.0	128	42	84	
4	24-04-2020	2514	1604	857	53	33672.0	138	86	49	

In [5]:
```
# tail() defines read last 5 data
read_df.tail()
```

Out[5]:

	Date	Total Confirmed Cases	Total Active Cases	Total Recovered Cases	Total Deceased Cases	Total Tests Cases	Daily Confirmed Cases	Daily Active Cases	Daily Recovered Cases	De
21	14-05-2020	8470	5310	3045	115	119736.0	472	276	187	
22	15-05-2020	8895	5254	3518	123	125189.0	425	-56	473	
23	16-05-2020	9333	5278	3926	129	130845.0	438	24	408	
24	17-05-2020	9755	5405	4202	148	135791.0	422	127	276	
25	18-05-2020	10054	5409	4485	160	139727.0	299	4	283	

In [6]:
```
read_df.columns
```

Out[6]:
```
Index(['Date', 'Total Confirmed Cases', 'Total Active Cases',
       'Total Recovered Cases', 'Total Deceased Cases', 'Total Tests Cases',
       'Daily Confirmed Cases', 'Daily Active Cases', 'Daily Recovered Cases',
       'Daily Deceased Cases', 'Daily Tests Cases', 'Doubling Time',
       'Growth Rate'],
      dtype='object')
```

In [7]:
```
# changing column name
column_names = ['Date', 'Total_Confirmed_Cases', 'Total_Active_Cases',
       'Total_Recovered_Cases', 'Total_Deceased_Cases', 'Total_Tests_Cases',
       'Daily_Confirmed_Cases', 'Daily_Active_Cases', 'Daily_Recovered_Cases',
       'Daily_Deceased_Cases', 'Daily_Tests_Cases', 'Doubling_Time',
       'Growth_Rate']

read_df.columns=column_names
```

In [8]:
```
# print columns
read_df
```

Out[8]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Ca
0	20-04-2020	2081	1603	431	
1	21-04-2020	2156	1498	611	
2	22-04-2020	2248	1476	724	
3	23-04-2020	2376	1518	808	
4	24-04-2020	2514	1604	857	
5	25-04-2020	2625	1702	869	
6	26-04-2020	2918	1987	877	
7	27-04-2020	3108	2177	877	
8	28-04-2020	3314	2182	1078	
9	29-04-2020	3439	2291	1092	
10	30-04-2020	3515	2362	1094	
11	01-05-2020	3738	2510	1167	
12	02-05-2020	4122	2802	1256	
13	03-05-2020	4549	3123	1362	
14	04-05-2020	4898	3403	1431	
15	05-05-2020	5104	3572	1468	
16	06-05-2020	5532	3925	1542	
17	08-05-2020	6542	4454	2020	

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Ca
18	09-05-2020	6542	4454	2020	
19	12-05-2020	7639	5041	2512	
20	13-05-2020	7998	5034	2858	
21	14-05-2020	8470	5310	3045	
22	15-05-2020	8895	5254	3518	
23	16-05-2020	9333	5278	3926	
24	17-05-2020	9755	5405	4202	
25	18-05-2020	10054	5409	4485	

In [9]:
```
# shape define number of rows and columns
read_df.shape
```
Out[9]:
```
(26, 13)
```

In [10]:
```
# describe shows mean, std, min,max
read_df.describe()
```
Out[10]:

	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Cases
count	26.000000	26.000000	26.000000	26.000000
mean	5133.269231	3283.615385	1774.230769	75.423077
std	2649.864382	1482.093467	1188.580475	32.934083
min	2081.000000	1476.000000	431.000000	47.000000
25%	2965.500000	2034.500000	877.000000	54.000000
50%	4335.500000	2962.500000	1309.000000	64.000000
75%	7364.750000	4889.000000	2389.000000	81.500000
max	10054.000000	5409.000000	4485.000000	160.000000

In [11]:
```
# info shows detail of all columns
read_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 13 columns):
Date                     26 non-null object
Total_Confirmed_Cases    26 non-null int64
Total_Active_Cases       26 non-null int64
Total_Recovered_Cases    26 non-null int64
Total_Deceased_Cases     26 non-null int64
Total_Tests_Cases        24 non-null float64
Daily_Confirmed_Cases    26 non-null int64
Daily_Active_Cases       26 non-null int64
Daily_Recovered_Cases    26 non-null int64
Daily_Deceased_Cases     26 non-null int64
Daily_Tests_Cases        23 non-null float64
Doubling_Time            26 non-null float64
Growth_Rate              26 non-null float64
dtypes: float64(4), int64(8), object(1)
memory usage: 2.7+ KB
```

In [12]:
```
# dtype show each data types
read_df.dtypes
```

Out[12]:
```
Date                      object
Total_Confirmed_Cases      int64
Total_Active_Cases         int64
Total_Recovered_Cases      int64
Total_Deceased_Cases       int64
Total_Tests_Cases        float64
Daily_Confirmed_Cases      int64
Daily_Active_Cases         int64
Daily_Recovered_Cases      int64
Daily_Deceased_Cases       int64
Daily_Tests_Cases        float64
Doubling_Time            float64
Growth_Rate              float64
dtype: object
```

In [13]:
```
# checking the columns if it contains null values(NA)[1] or not[0]
# isnull() defines checks null values
# sum() defines no. of null values
read_df[column_names].isnull().sum()
```

Out[13]:
```
Date                     0
Total_Confirmed_Cases    0
Total_Active_Cases       0
Total_Recovered_Cases    0
Total_Deceased_Cases     0
Total_Tests_Cases        2
Daily_Confirmed_Cases    0
Daily_Active_Cases       0
Daily_Recovered_Cases    0
Daily_Deceased_Cases     0
Daily_Tests_Cases        3
Doubling_Time            0
Growth_Rate              0
dtype: int64
```

In [14]:
```
# Here, we are trying to find all the categorical and numerical variables separ
cat_list = []
num_list = []

for variable in read_df.columns:
    if read_df[variable].dtype.name in ['object']:
        cat_list.append(variable)
    else:
        num_list.append(variable)

print("Categorical Variables : ", cat_list, '\n')
print("Numerical Variables : ", num_list)
```

```
Categorical Variables :  ['Date']

Numerical Variables :  ['Total_Confirmed_Cases', 'Total_Active_Cases', 'Total_
Recovered_Cases', 'Total_Deceased_Cases', 'Total_Tests_Cases', 'Daily_Confirme
d_Cases', 'Daily_Active_Cases', 'Daily_Recovered_Cases', 'Daily_Deceased_Case
s', 'Daily_Tests_Cases', 'Doubling_Time', 'Growth_Rate']
```

In [15]:
```
# describe first 7 columns
read_df[column_names[:7]].describe()
```

Out[15]:

	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Cases
count	26.000000	26.000000	26.000000	26.000000
mean	5133.269231	3283.615385	1774.230769	75.423077
std	2649.864382	1482.093467	1188.580475	32.934083
min	2081.000000	1476.000000	431.000000	47.000000
25%	2965.500000	2034.500000	877.000000	54.000000
50%	4335.500000	2962.500000	1309.000000	64.000000
75%	7364.750000	4889.000000	2389.000000	81.500000
max	10054.000000	5409.000000	4485.000000	160.000000

In [16]:
```
# describe after 7th column till 11th column
read_df[column_names[7:]].describe()
```

Out[16]:

	Daily_Active_Cases	Daily_Recovered_Cases	Daily_Deceased_Cases	Daily_Tests_Cases	Do
count	26.000000	26.000000	26.000000	23.000000	
mean	117.923077	142.192308	4.192308	3721.869565	
std	133.585006	138.776949	5.824220	1938.380142	
min	-105.000000	0.000000	0.000000	727.000000	
25%	6.250000	40.000000	0.000000	2172.500000	
50%	103.000000	89.000000	2.000000	3744.000000	
75%	241.250000	197.500000	5.250000	4539.500000	
max	353.000000	473.000000	20.000000	8431.000000	

In [17]:
```
read_df.Date.describe()
```
Out[17]:
```
count              26
unique             26
top        04-05-2020
freq                1
Name: Date, dtype: object
```

Data Processing

In [18]:
```
FMT = '%d-%m-%Y'
date = read_df['Date']
read_df['Date'] = date.map(lambda x : (datetime.strptime(x, FMT) - datetime.str
```

In [19]:
```
read_df
```
Out[19]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Ca
0	111	2081	1603	431	
1	112	2156	1498	611	
2	113	2248	1476	724	
3	114	2376	1518	808	
4	115	2514	1604	857	
5	116	2625	1702	869	
6	117	2918	1987	877	
7	118	3108	2177	877	
8	119	3314	2182	1078	
9	120	3439	2291	1092	
10	121	3515	2362	1094	
11	122	3738	2510	1167	
12	123	4122	2802	1256	
13	124	4549	3123	1362	
14	125	4898	3403	1431	
15	126	5104	3572	1468	
16	127	5532	3925	1542	
17	129	6542	4454	2020	
18	130	6542	4454	2020	
19	133	7639	5041	2512	
20	134	7998	5034	2858	
21	135	8470	5310	3045	
22	136	8895	5254	3518	
23	137	9333	5278	3926	
24	138	9755	5405	4202	
25	139	10054	5409	4485	

In [20]:
```
# head() defines read first 5 data
read_df.head()
```

Out[20]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Cas
0	111	2081	1603	431	
1	112	2156	1498	611	
2	113	2248	1476	724	
3	114	2376	1518	808	
4	115	2514	1604	857	

In [21]:
```
# tail() defines read last 5 data
read_df.tail()
```

Out[21]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Ca
21	135	8470	5310	3045	
22	136	8895	5254	3518	
23	137	9333	5278	3926	
24	138	9755	5405	4202	
25	139	10054	5409	4485	

In [22]:
```
# Here, we are trying to find all the categorical and numerical variables separ
cat_list = []
num_list = []

for variable in read_df.columns:
    if read_df[variable].dtype.name in ['object']:
        cat_list.append(variable)
    else:
        num_list.append(variable)

print("Categorical Variables : ", cat_list, '\n')
print("Numerical Variables : ", num_list)
```

```
Categorical Variables :  []

Numerical Variables :  ['Date', 'Total_Confirmed_Cases', 'Total_Active_Cases',
'Total_Recovered_Cases', 'Total_Deceased_Cases', 'Total_Tests_Cases', 'Daily_C
onfirmed_Cases', 'Daily_Active_Cases', 'Daily_Recovered_Cases', 'Daily_Decease
d_Cases', 'Daily_Tests_Cases', 'Doubling_Time', 'Growth_Rate']
```

In [23]:
```
# describe shows mean, std, min,max
read_df.describe()
```

Out[23]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Dec
count	26.000000	26.000000	26.000000	26.000000	
mean	124.384615	5133.269231	3283.615385	1774.230769	
std	8.777594	2649.864382	1482.093467	1188.580475	
min	111.000000	2081.000000	1476.000000	431.000000	
25%	117.250000	2965.500000	2034.500000	877.000000	
50%	123.500000	4335.500000	2962.500000	1309.000000	
75%	132.250000	7364.750000	4889.000000	2389.000000	
max	139.000000	10054.000000	5409.000000	4485.000000	

Data Analysis

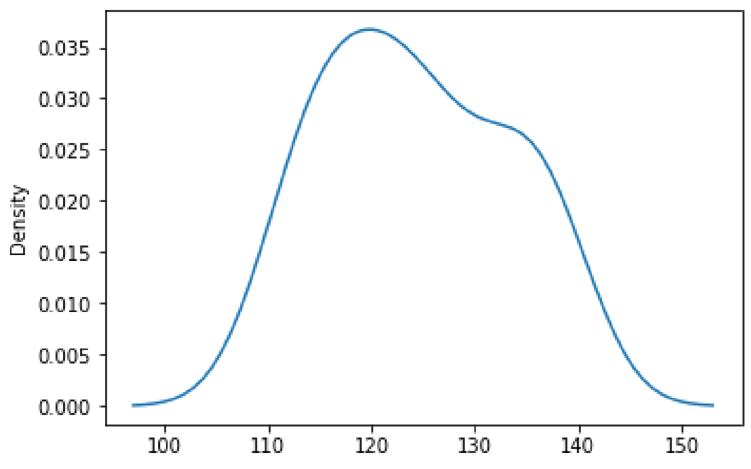
1. Date

In [24]:
```
read_df.Date.describe()
```

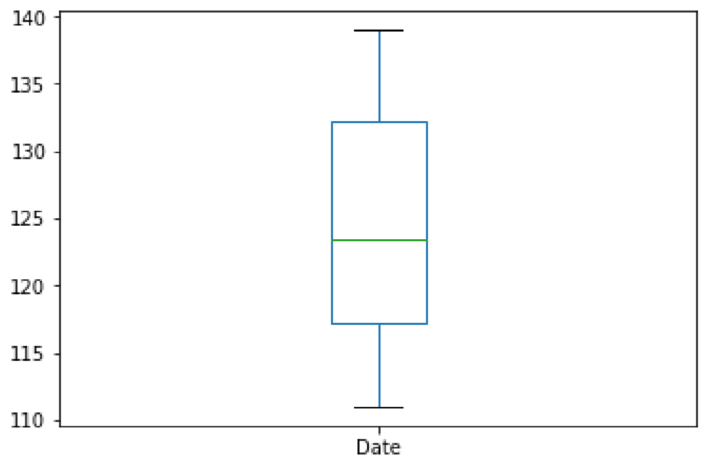
Out[24]:
```
count     26.000000
mean     124.384615
std        8.777594
min      111.000000
25%      117.250000
50%      123.500000
75%      132.250000
max      139.000000
Name: Date, dtype: float64
```

In [25]:
```
# plot density curve of Date after cleaning null values
read_df.Date.plot.density()
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1eacaf0bac8>
```



In [26]:
```
# Box and Whisker Plots
read_df.Date.plot(kind='box', subplots=True, sharex=False, sharey=False)
plt.show()
```



2. Total_Confirmed_Cases

In [27]:
```
read_df.Total_Confirmed_Cases.describe()
```

Out[27]:
```
count       26.000000
mean      5133.269231
std       2649.864382
min       2081.000000
25%       2965.500000
50%       4335.500000
75%       7364.750000
max      10054.000000
Name: Total_Confirmed_Cases, dtype: float64
```

In [28]:
```
# Displaying mean
read_df.Total_Confirmed_Cases.mean()
```

Out[28]:
```
5133.2692307692305
```

In [29]:
```
# Displaying mode
read_df.Total_Confirmed_Cases.mode()
```

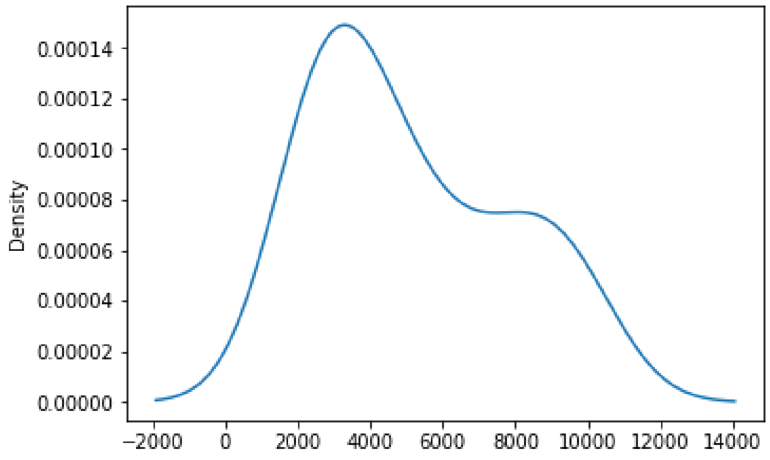
Out[29]:
```
0    6542
dtype: int64
```

In [30]:
```
# displaying median
read_df.Total_Confirmed_Cases.median()
```

Out[30]:
```
4335.5
```

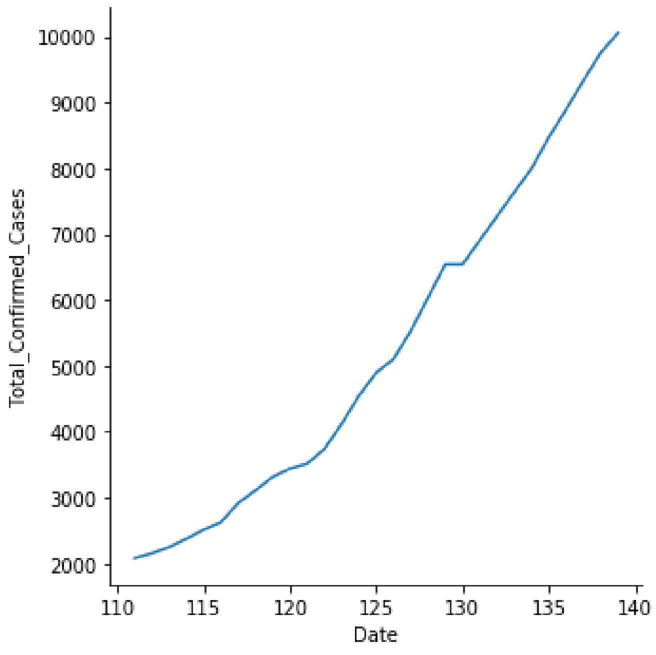
In [31]:
```python
# plot density curve of Total_Confirmed_Cases after cleaning null values
read_df.Total_Confirmed_Cases.plot.density()
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1eacaf9a240>
```



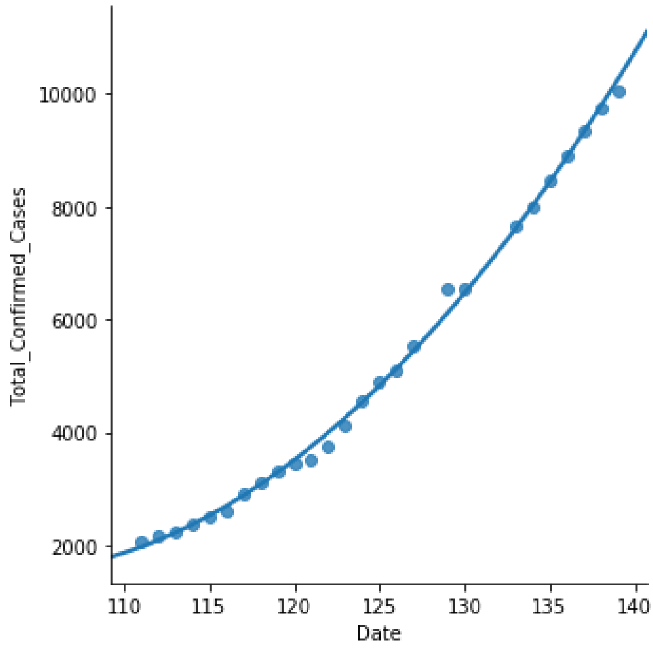
In [32]:
```python
# Relplot
sns.relplot(x="Date", y="Total_Confirmed_Cases", kind="line", data=read_df)
```

Out[32]:
```
<seaborn.axisgrid.FacetGrid at 0x1eacb26b550>
```

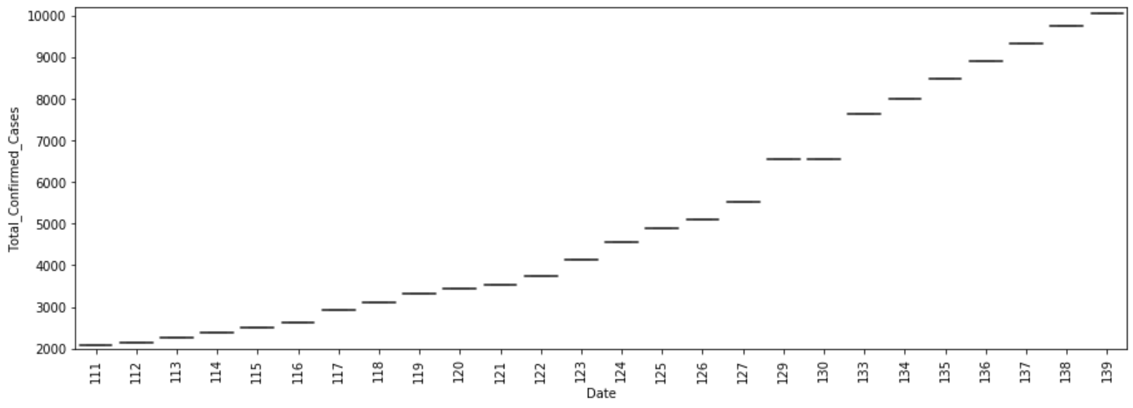


```
In [33]:  # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Total_Confirmed_Cases", data = read_df, order = 2, ci
```

```
Out[33]: <seaborn.axisgrid.FacetGrid at 0x1eacb2beb38>
```



```
In [34]:  # Box Plot
          var = 'Date'
          data2 = pd.concat([read_df['Total_Confirmed_Cases'], read_df[var]], axis=1)
          f, ax = plt.subplots(figsize=(15, 5))
          fig = sns.boxplot(x=var, y="Total_Confirmed_Cases", data=data2)
          fig.axis(ymin=2000, ymax=10200);
          plt.xticks(rotation=90);
```



3. Total_Active_Cases

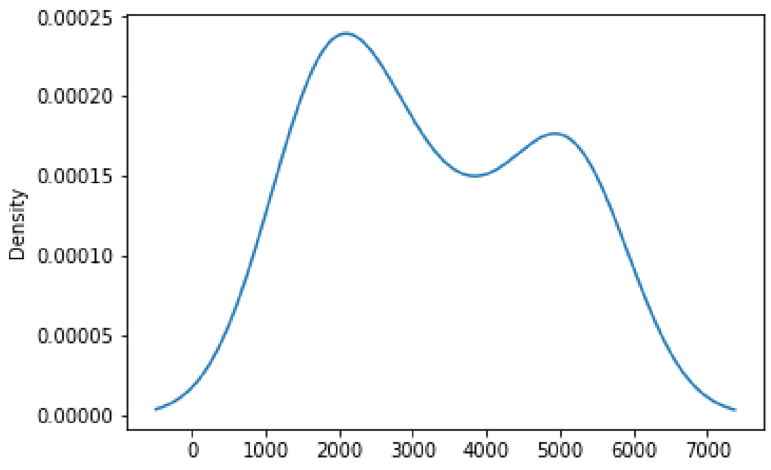
```
In [35]:  read_df.Total_Active_Cases.describe()

Out[35]:  count      26.000000
          mean     3283.615385
          std      1482.093467
          min      1476.000000
          25%      2034.500000
          50%      2962.500000
          75%      4889.000000
          max      5409.000000
          Name: Total_Active_Cases, dtype: float64
```

```
In [36]:  # Displaying mean
          read_df.Total_Active_Cases.mean()

Out[36]:  3283.6153846153848
```

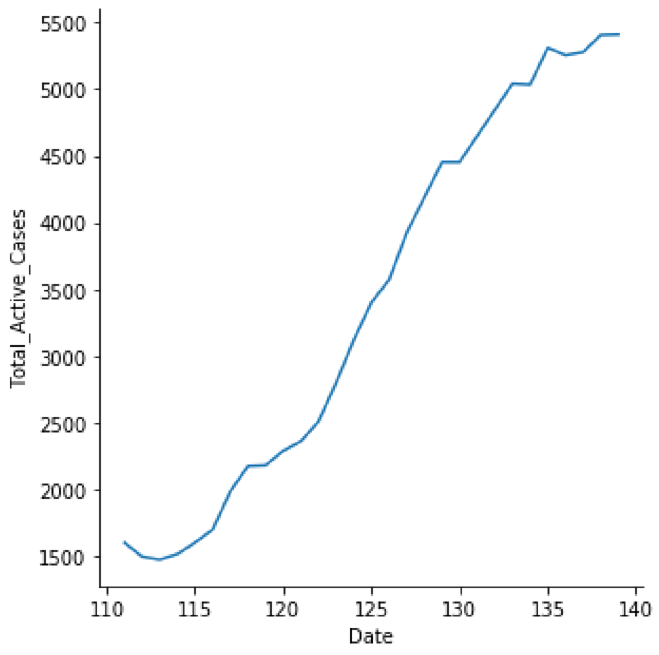
```
In [37]:  # displaying mode
          read_df.Total_Active_Cases.mode()

Out[37]:  0    4454
          dtype: int64
```

```
In [38]:  # displaying median
          read_df.Total_Active_Cases.median()

Out[38]:  2962.5
```

```
In [39]:  # plot density curve of Total_Active_Cases after cleaning null values
          read_df.Total_Active_Cases.plot.density()

Out[39]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacb73cc88>
```

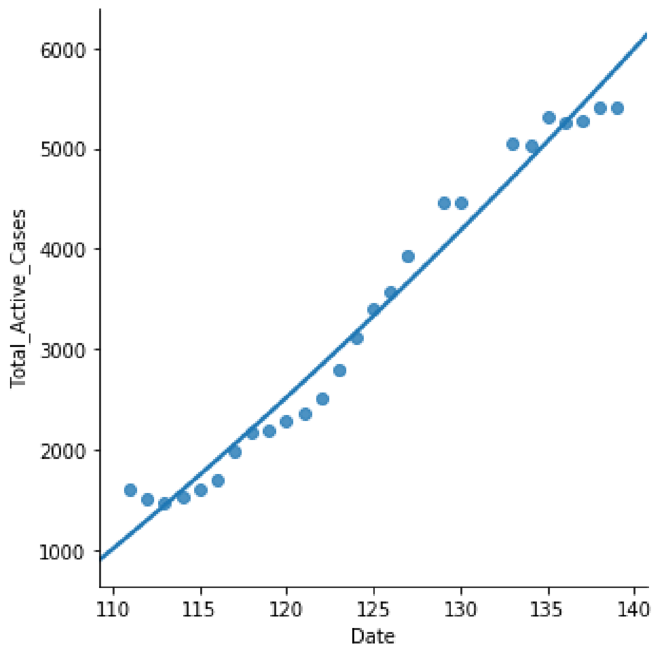


```
In [40]:  # Relplot
          sns.relplot(x="Date", y="Total_Active_Cases", kind="line", data=read_df)

Out[40]:  <seaborn.axisgrid.FacetGrid at 0x1eacae940f0>
```

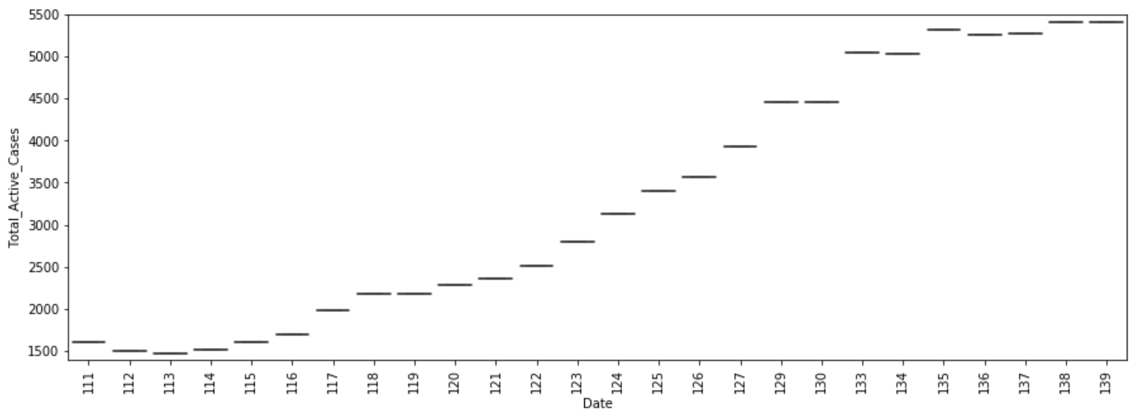


In [41]:
```
# Plotting the data scatter
sns.lmplot(x ="Date", y ="Total_Active_Cases", data = read_df, order = 2, ci =
```

Out[41]:
```
<seaborn.axisgrid.FacetGrid at 0x1eacb5f2978>
```



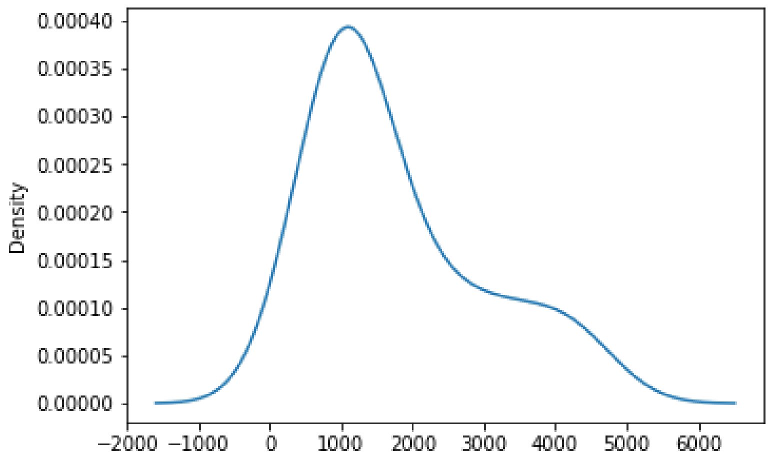
In [42]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Total_Active_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Total_Active_Cases", data=data2)
fig.axis(ymin=1400, ymax=5500);
plt.xticks(rotation=90);
```



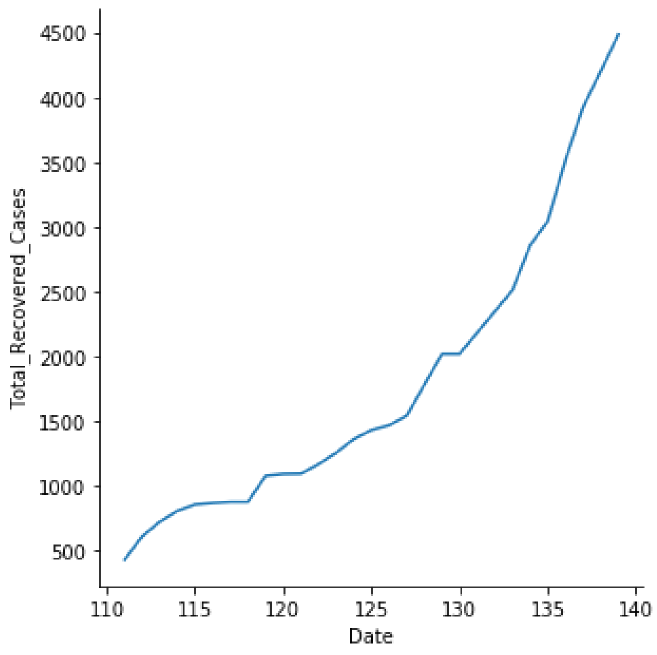
4. Total_Recovered_Cases

In [43]:
```
read_df.Total_Recovered_Cases.describe()
```

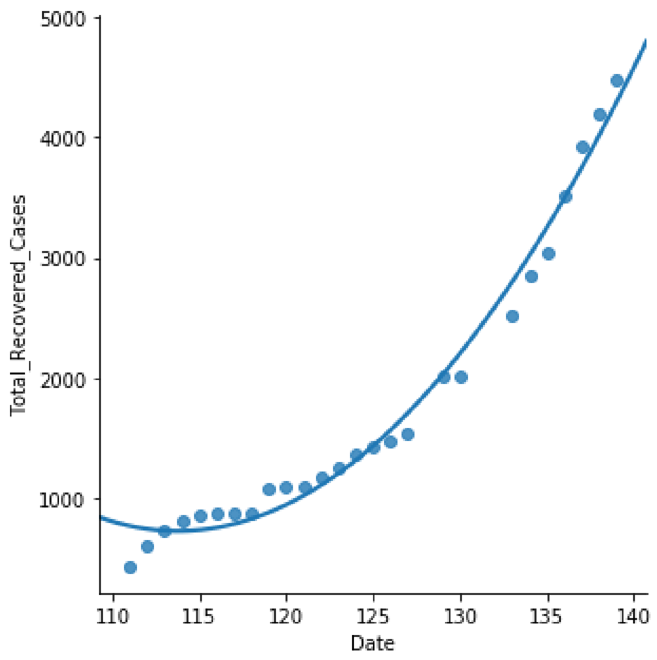
Out[43]:
```
count      26.000000
mean     1774.230769
std      1188.580475
min       431.000000
25%       877.000000
50%      1309.000000
75%      2389.000000
max      4485.000000
Name: Total_Recovered_Cases, dtype: float64
```

In [44]:
```
# Displaying mean
read_df.Total_Recovered_Cases.mean()
```
Out[44]:
```
1774.2307692307693
```
In [45]:
```
# displaying mode
read_df.Total_Recovered_Cases.mode()
```
Out[45]:
```
0     877
1    2020
dtype: int64
```
In [46]:
```
# displaying median
read_df.Total_Recovered_Cases.median()
```
Out[46]:
```
1309.0
```
In [47]:
```
# plot density curve of Total_Recovered_Cases after cleaning null values
read_df.Total_Recovered_Cases.plot.density()
```
Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1eacbc08588>
```


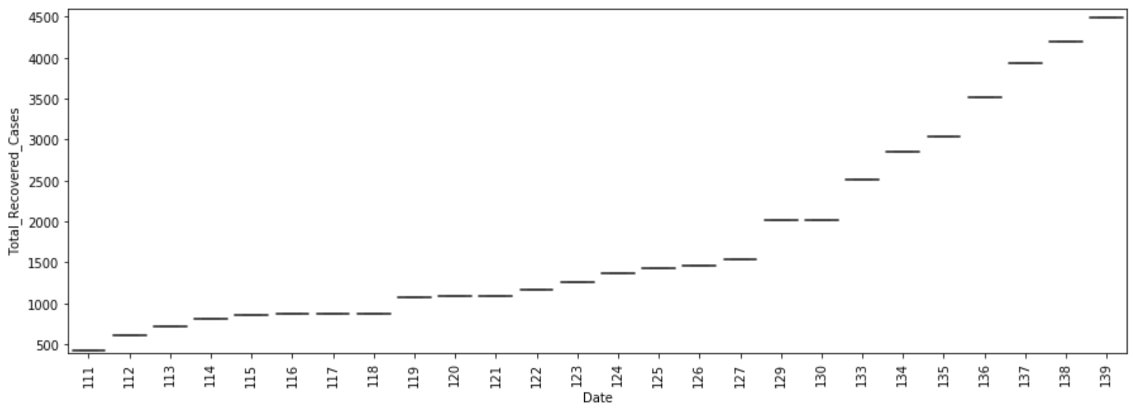
```
In [48]:  # Relplot
          sns.relplot(x="Date", y="Total_Recovered_Cases", kind="line", data=read_df)

Out[48]:  <seaborn.axisgrid.FacetGrid at 0x1eacb388470>
```



```
In [49]:  # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Total_Recovered_Cases", data = read_df, order = 2, ci

Out[49]:  <seaborn.axisgrid.FacetGrid at 0x1eacade3400>
```



In [50]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Total_Recovered_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Total_Recovered_Cases", data=data2)
fig.axis(ymin=400, ymax=4600);
plt.xticks(rotation=90);
```



5. Total_Deceased_Cases

In [51]:
```
read_df.Total_Deceased_Cases.describe()
```

Out[51]:
```
count     26.000000
mean      75.423077
std       32.934083
min       47.000000
25%       54.000000
50%       64.000000
75%       81.500000
max      160.000000
Name: Total_Deceased_Cases, dtype: float64
```

In [52]:
```
# Displaying mean
read_df.Total_Deceased_Cases.mean()
```

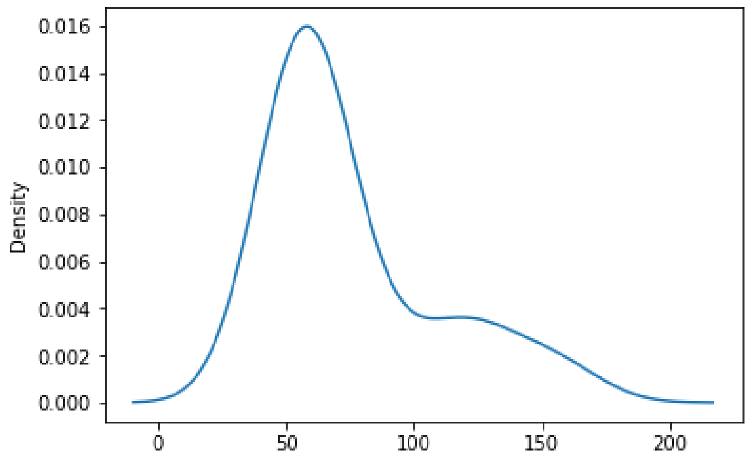
Out[52]:
```
75.42307692307692
```

In [53]:
```
# displaying mode
read_df.Total_Deceased_Cases.mode()
```

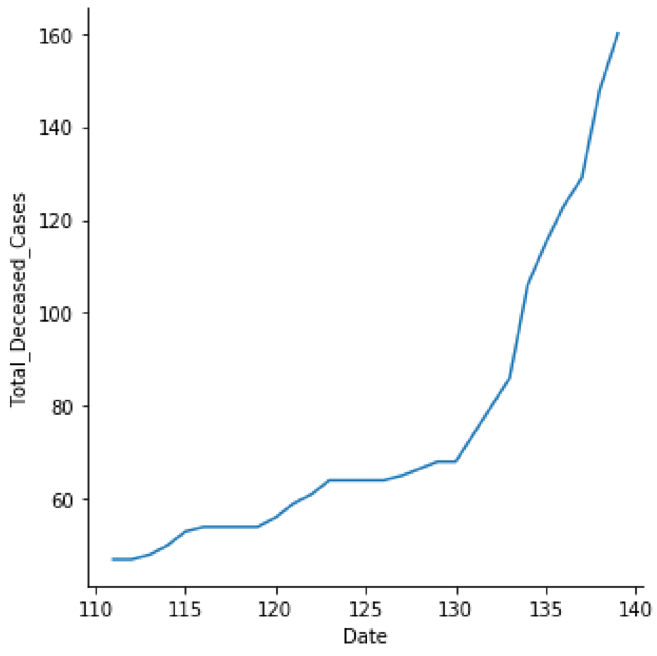
Out[53]:
```
0    54
1    64
dtype: int64
```

```
In [54]:  # displaying median
           read_df.Total_Deceased_Cases.median()

Out[54]:  64.0
```

```
In [55]:  # plot density curve of Total_Deceased_Cases after cleaning null values
           read_df.Total_Deceased_Cases.plot.density()

Out[55]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacb7fcf28>
```

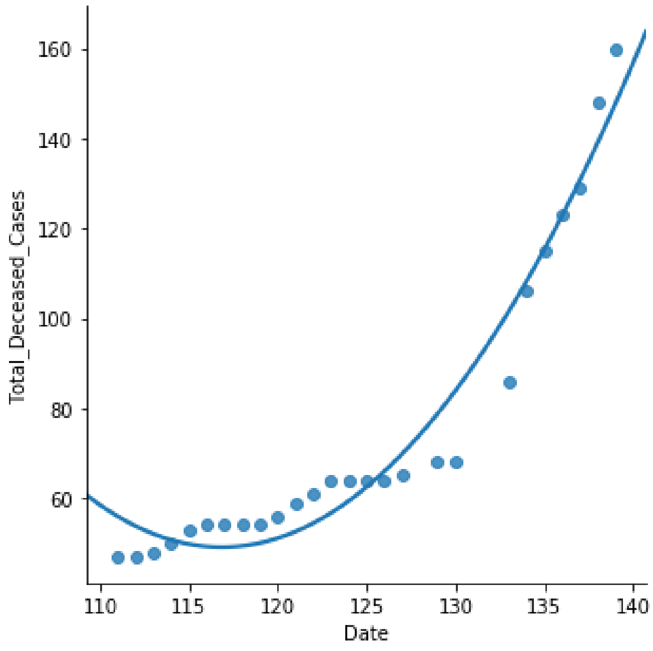


```
In [56]:  # Relplot
           sns.relplot(x="Date", y="Total_Deceased_Cases", kind="line", data=read_df)

Out[56]:  <seaborn.axisgrid.FacetGrid at 0x1eacb86dcc0>
```

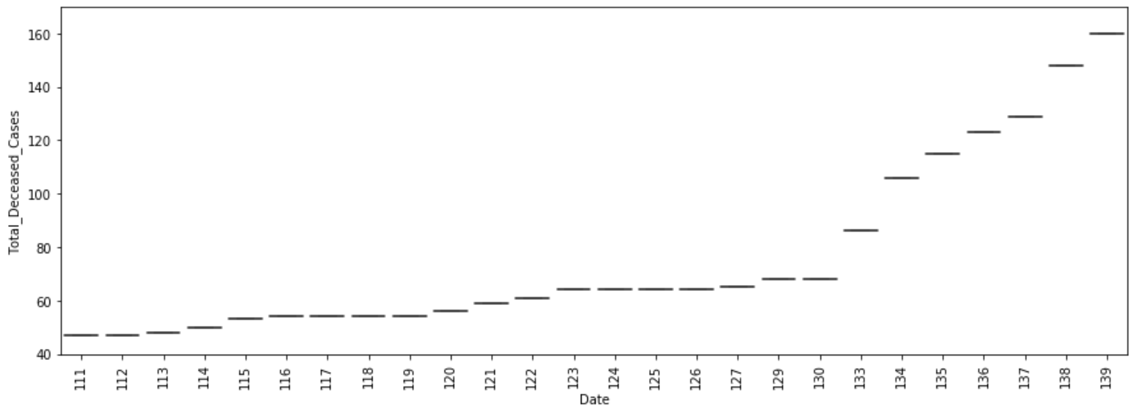


In [57]:
```
# Plotting the data scatter
sns.lmplot(x ="Date", y ="Total_Deceased_Cases", data = read_df, order = 2, ci
```

Out[57]:
```
<seaborn.axisgrid.FacetGrid at 0x1eacba2c9b0>
```



In [58]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Total_Deceased_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Total_Deceased_Cases", data=data2)
fig.axis(ymin=40, ymax=170);
plt.xticks(rotation=90);
```



6. Total_Tests_Cases

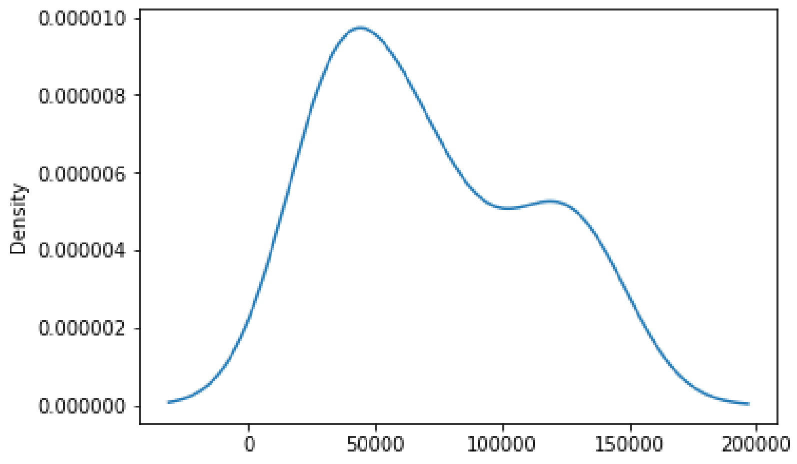
```
In [59]: read_df.Total_Tests_Cases.describe()

Out[59]: count        24.000000
         mean      71141.291667
         std       38999.763719
         min       25900.000000
         25%       37089.500000
         50%       62177.000000
         75%      107918.000000
         max      139727.000000
         Name: Total_Tests_Cases, dtype: float64
```

```
In [60]: # Displaying mean
         read_df.Total_Tests_Cases.mean()

Out[60]: 71141.29166666667
```

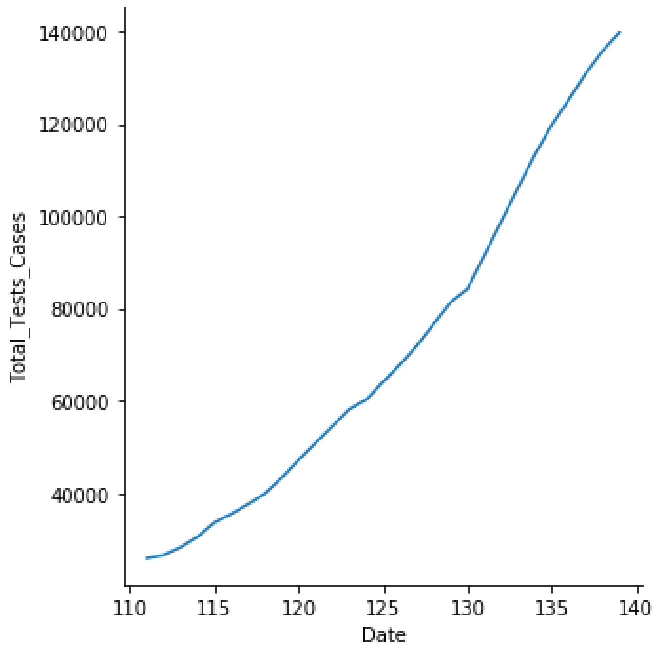
```
In [61]: # displaying mode
         read_df.Total_Tests_Cases.mode()

Out[61]: 0      25900.0
         1      26627.0
         2      28309.0
         3      30560.0
         4      33672.0
         5      35519.0
         6      37613.0
         7      39911.0
         8      43370.0
         9      47225.0
         10     58210.0
         11     60246.0
         12     64108.0
         13     67852.0
         14     71934.0
         15     81367.0
         16     84226.0
         17    106109.0
         18    113345.0
         19    119736.0
         20    125189.0
         21    130845.0
         22    135791.0
         23    139727.0
         dtype: float64
```

```
In [62]: # displaying median
         read_df.Total_Tests_Cases.median()

Out[62]: 62177.0
```

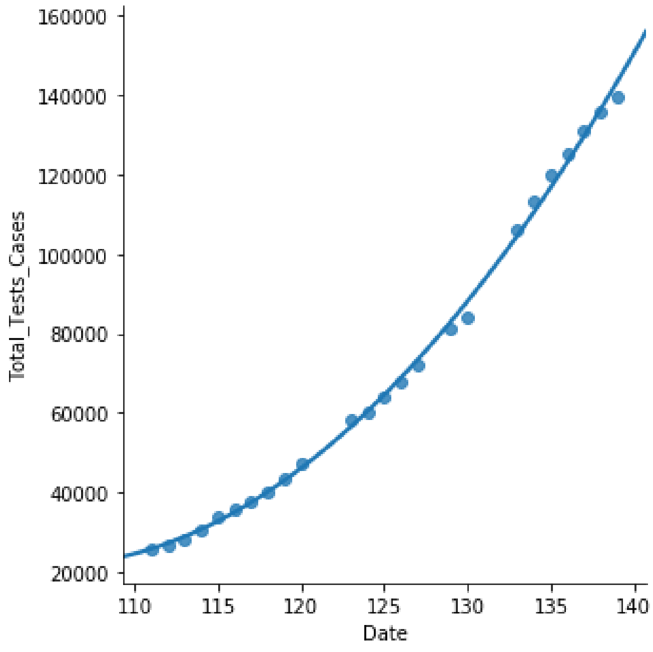
```
In [63]:  # plot density curve of Total_Tests_Cases after cleaning null values
          read_df.Total_Tests_Cases.plot.density()

Out[63]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacd13b9b0>
```



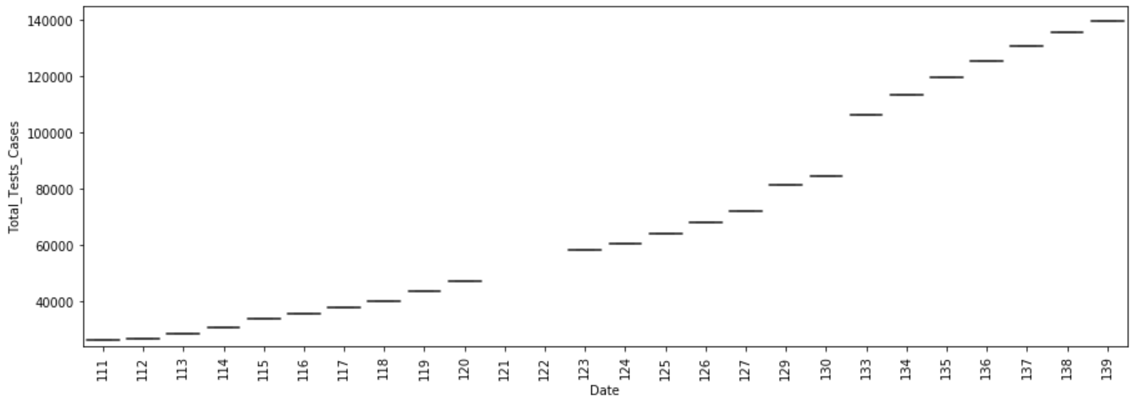
```
In [64]:  # Relplot
          sns.relplot(x="Date", y="Total_Tests_Cases", kind="line", data=read_df)

Out[64]:  <seaborn.axisgrid.FacetGrid at 0x1eacb79a320>
```



```
In [65]:  # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Total_Tests_Cases", data = read_df, order = 2, ci = N

Out[65]: <seaborn.axisgrid.FacetGrid at 0x1eacb97f320>
```



```
In [66]:  # Box Plot
          var = 'Date'
          data2 = pd.concat([read_df['Total_Tests_Cases'], read_df[var]], axis=1)
          f, ax = plt.subplots(figsize=(15, 5))
          fig = sns.boxplot(x=var, y="Total_Tests_Cases", data=data2)
          fig.axis(ymin=24000, ymax=145000);
          plt.xticks(rotation=90);
```



7. Daily_Confirmed_Cases

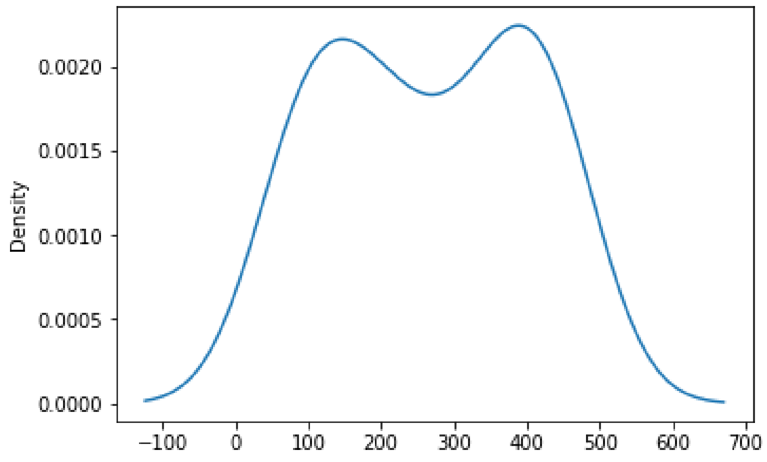
```
In [67]:  read_df.Daily_Confirmed_Cases.describe()

Out[67]:  count     26.000000
          mean     265.846154
          std      136.233533
          min       75.000000
          25%      130.500000
          50%      258.500000
          75%      400.500000
          max      472.000000
          Name: Daily_Confirmed_Cases, dtype: float64
```

```
In [68]:  # Displaying mean
          read_df.Daily_Confirmed_Cases.mean()

Out[68]:  265.84615384615387
```

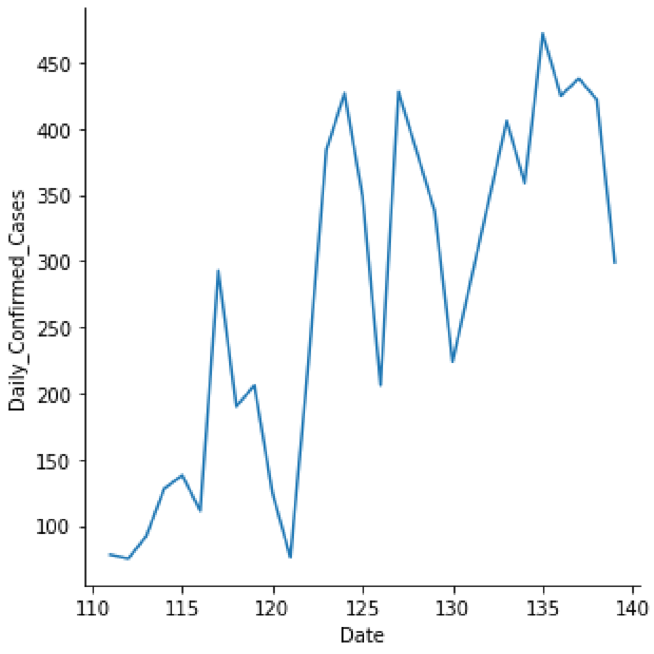
```
In [69]:  # displaying mode
          read_df.Daily_Confirmed_Cases.mode()

Out[69]:  0    206
          dtype: int64
```

```
In [70]:  # displaying median
          read_df.Daily_Confirmed_Cases.median()

Out[70]:  258.5
```

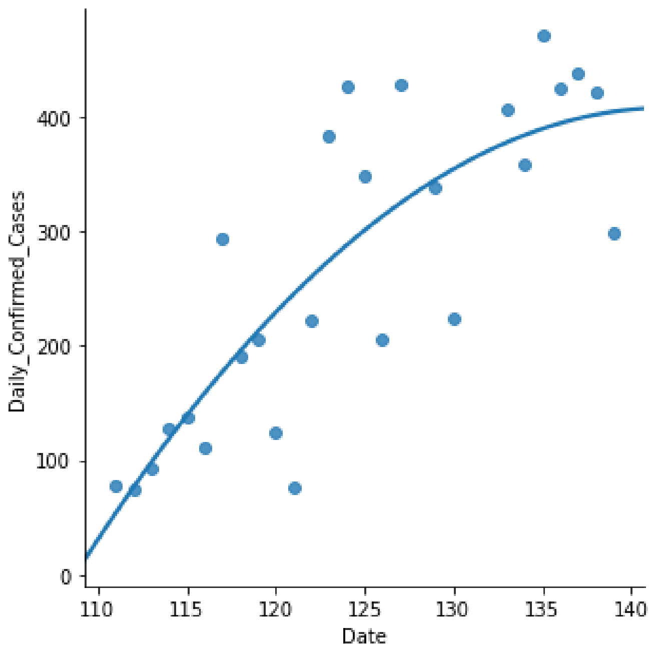
```
In [71]:  # plot density curve of Daily_Confirmed_Cases after cleaning null values
          read_df.Daily_Confirmed_Cases.plot.density()

Out[71]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacb496828>
```



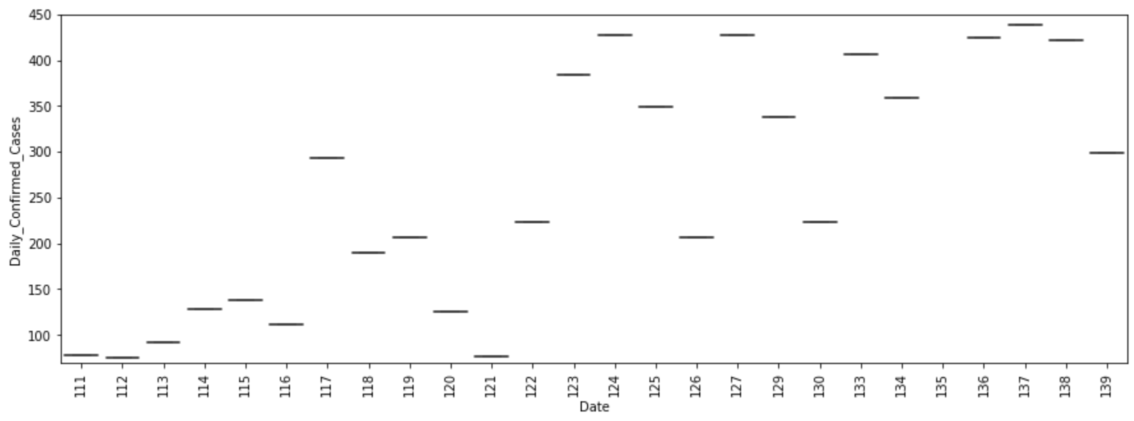
```
In [72]:  # Relplot
          sns.relplot(x="Date", y="Daily_Confirmed_Cases", kind="line", data=read_df)

Out[72]: <seaborn.axisgrid.FacetGrid at 0x1eacb44c438>
```



```
In [73]:  # Plotting the data scatter
           sns.lmplot(x ="Date", y ="Daily_Confirmed_Cases", data = read_df, order = 2, ci

Out[73]:  <seaborn.axisgrid.FacetGrid at 0x1eacb3e4cc0>
```



```
In [74]:  # Box Plot
           var = 'Date'
           data2 = pd.concat([read_df['Daily_Confirmed_Cases'], read_df[var]], axis=1)
           f, ax = plt.subplots(figsize=(15, 5))
           fig = sns.boxplot(x=var, y="Daily_Confirmed_Cases", data=data2)
           fig.axis(ymin=70, ymax=450);
           plt.xticks(rotation=90);
```



8. Daily_Active_Cases

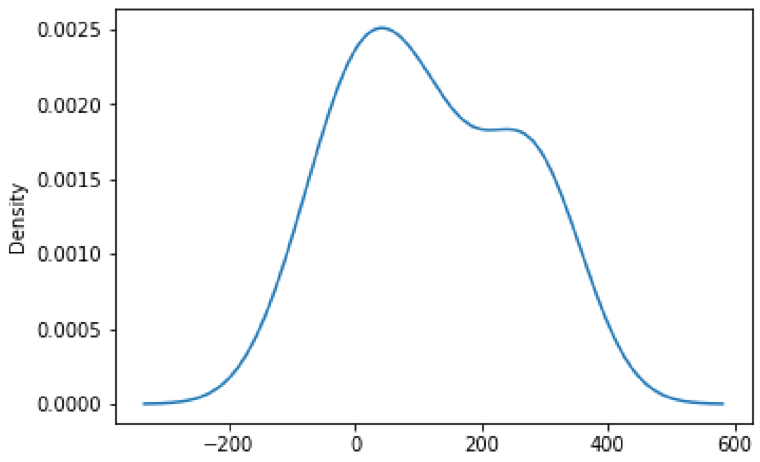
```
In [75]:  read_df.Daily_Active_Cases.describe()

Out[75]:  count     26.000000
          mean     117.923077
          std      133.585006
          min     -105.000000
          25%        6.250000
          50%      103.000000
          75%      241.250000
          max      353.000000
          Name: Daily_Active_Cases, dtype: float64
```

```
In [76]:  # Displaying mean
          read_df.Daily_Active_Cases.mean()

Out[76]:  117.92307692307692
```

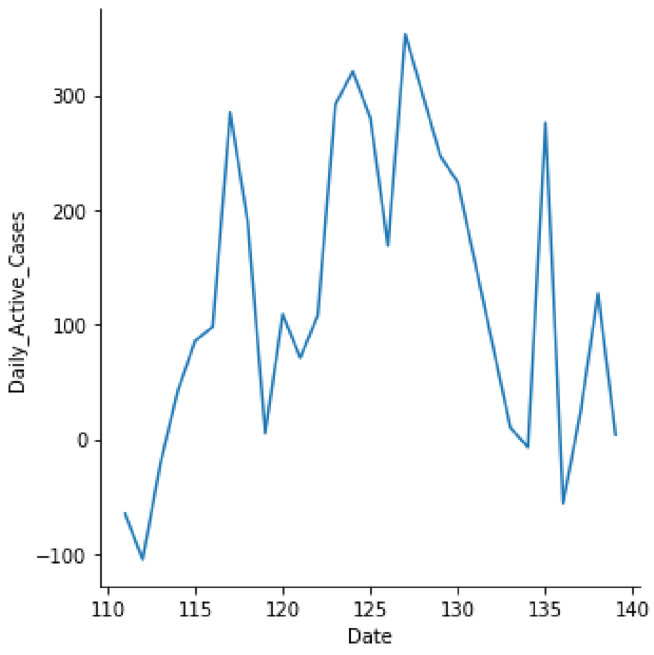
```
In [77]:  # displaying mode
          read_df.Daily_Active_Cases.mode()

Out[77]:  0     -105
          1      -65
          2      -56
          3      -22
          4       -7
          5        4
          6        5
          7       10
          8       24
          9       42
          10      71
          11      86
          12      98
          13     108
          14     109
          15     127
          16     169
          17     190
          18     224
          19     247
          20     276
          21     280
          22     285
          23     292
          24     321
          25     353
          dtype: int64
```

```
In [78]:  # displaying median
          read_df.Daily_Active_Cases.median()

Out[78]:  103.0
```

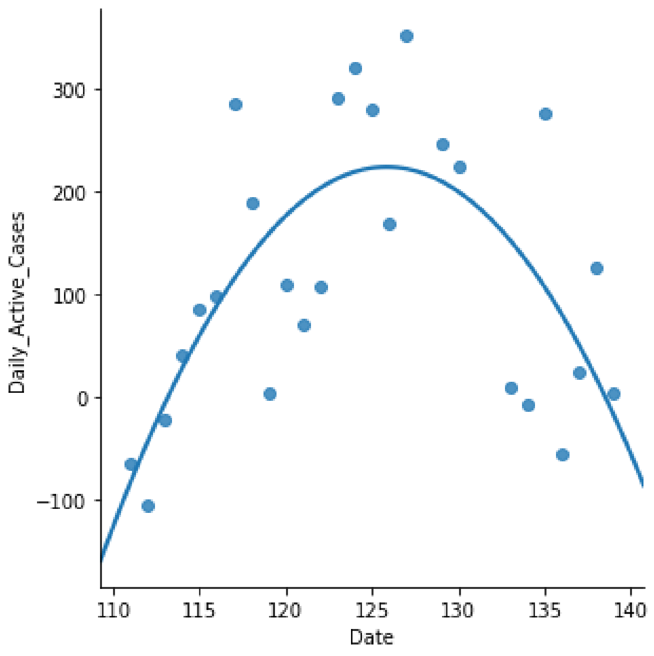
```
In [79]:  # plot density curve of Daily_Active_Cases after cleaning null values
          read_df.Daily_Active_Cases.plot.density()

Out[79]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacce0b128>
```



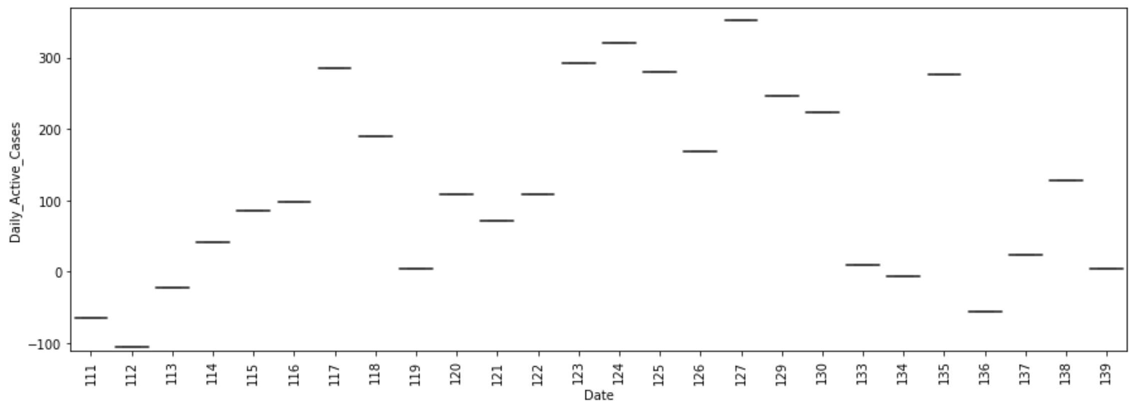
```
In [80]:  # Relplot
          sns.relplot(x="Date", y="Daily_Active_Cases", kind="line", data=read_df)

Out[80]:  <seaborn.axisgrid.FacetGrid at 0x1eacd0a7978>
```



```
In [81]:  # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Daily_Active_Cases", data = read_df, order = 2, ci =

Out[81]:  <seaborn.axisgrid.FacetGrid at 0x1eacb98a5f8>
```



In [82]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Daily_Active_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Daily_Active_Cases", data=data2)
fig.axis(ymin=-110, ymax=370);
plt.xticks(rotation=90);
```



9. Daily_Recovered_Cases

In [83]:
```
read_df.Daily_Recovered_Cases.describe()
```

Out[83]:
```
count     26.000000
mean     142.192308
std      138.776949
min        0.000000
25%       40.000000
50%       89.000000
75%      197.500000
max      473.000000
Name: Daily_Recovered_Cases, dtype: float64
```

In [84]:
```
# Displaying mean
read_df.Daily_Recovered_Cases.mean()
```

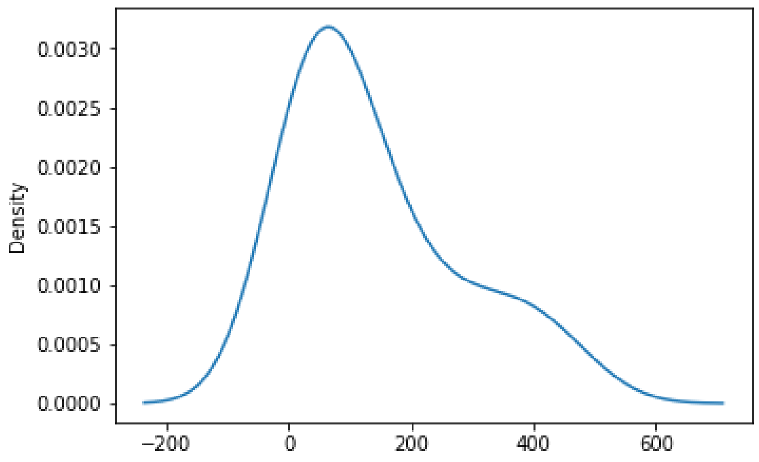
Out[84]:
```
142.19230769230768
```

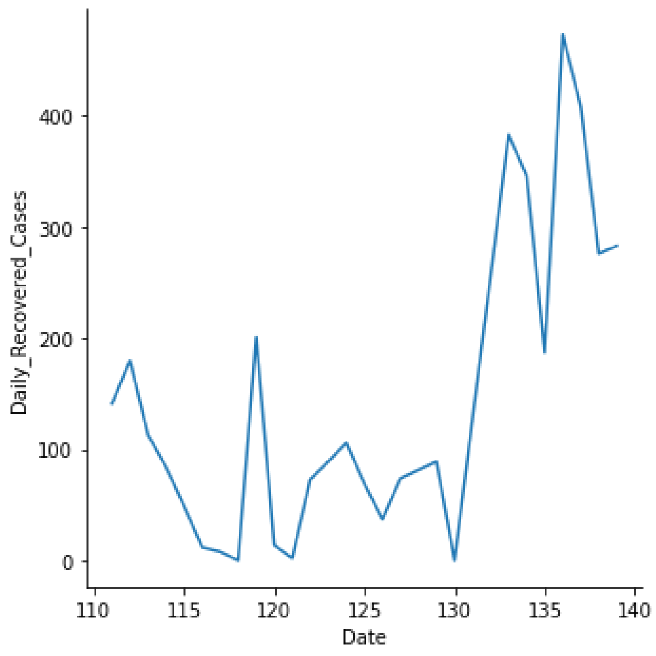
In [85]:
```
# displaying mode
read_df.Daily_Recovered_Cases.mode()
```

Out[85]:
```
0     0
1    89
dtype: int64
```

In [86]:
```
# displaying median
read_df.Daily_Recovered_Cases.median()
```

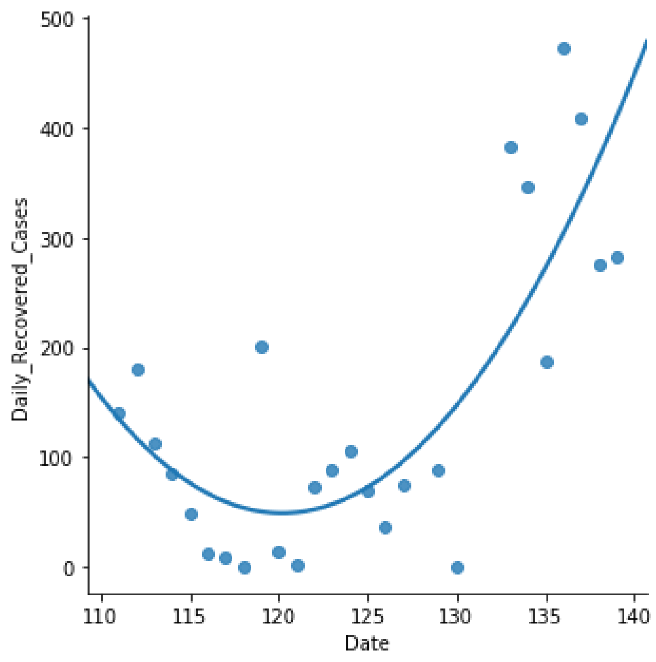
Out[86]:
```
89.0
```

In [87]:
```
# plot density curve of Daily_Recovered_Cases after cleaning null values
read_df.Daily_Recovered_Cases.plot.density()
```
Out[87]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1eacb3a4fd0>
```


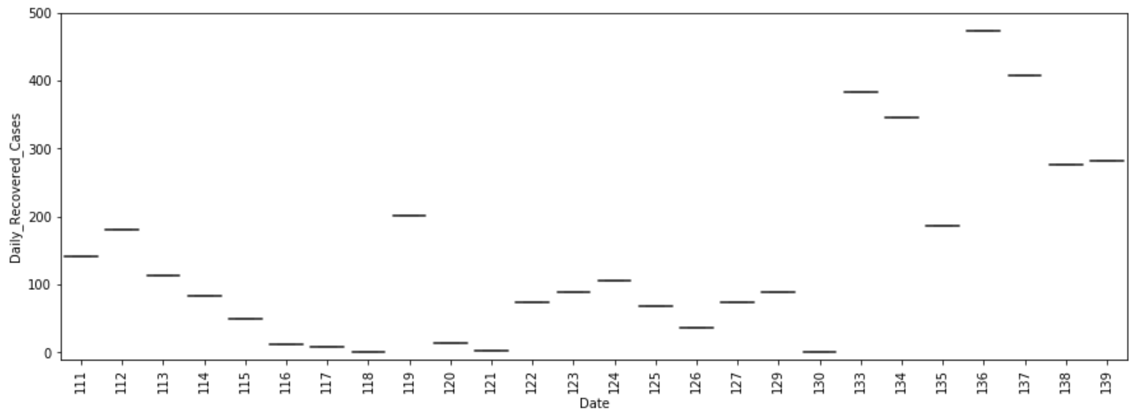
In [88]:
```
# Relplot
sns.relplot(x="Date", y="Daily_Recovered_Cases", kind="line", data=read_df)
```
Out[88]:
```
<seaborn.axisgrid.FacetGrid at 0x1eacbaf2f98>
```


```
In [89]:  # Plotting the data scatter
           sns.lmplot(x ="Date", y ="Daily_Recovered_Cases", data = read_df, order = 2, ci
```

```
Out[89]: <seaborn.axisgrid.FacetGrid at 0x1eacbb9fcf8>
```



In [90]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Daily_Recovered_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Daily_Recovered_Cases", data=data2)
fig.axis(ymin=-10, ymax=500);
plt.xticks(rotation=90);
```



10. Daily_Deceased_Cases

In [91]:
```
read_df.Daily_Deceased_Cases.describe()
```

Out[91]:
```
count    26.000000
mean      4.192308
std       5.824220
min       0.000000
25%       0.000000
50%       2.000000
75%       5.250000
max      20.000000
Name: Daily_Deceased_Cases, dtype: float64
```

In [92]:
```
# Displaying mean
read_df.Daily_Deceased_Cases.mean()
```

Out[92]:
```
4.1923076923076925
```

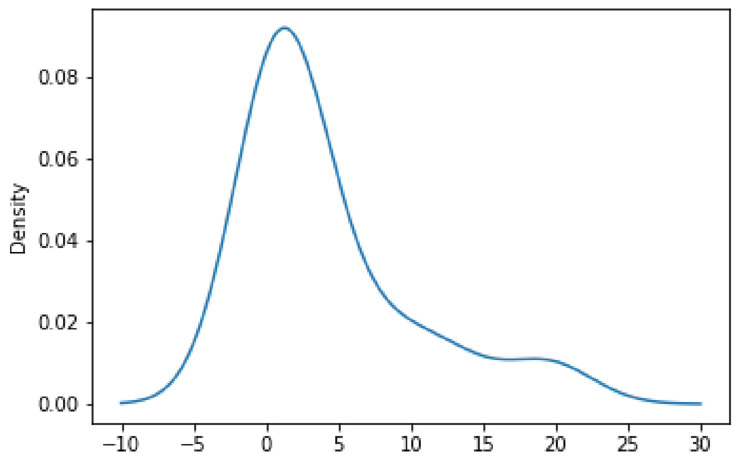
In [93]:
```
# displaying mode
read_df.Daily_Deceased_Cases.mode()
```

Out[93]:
```
0    0
dtype: int64
```

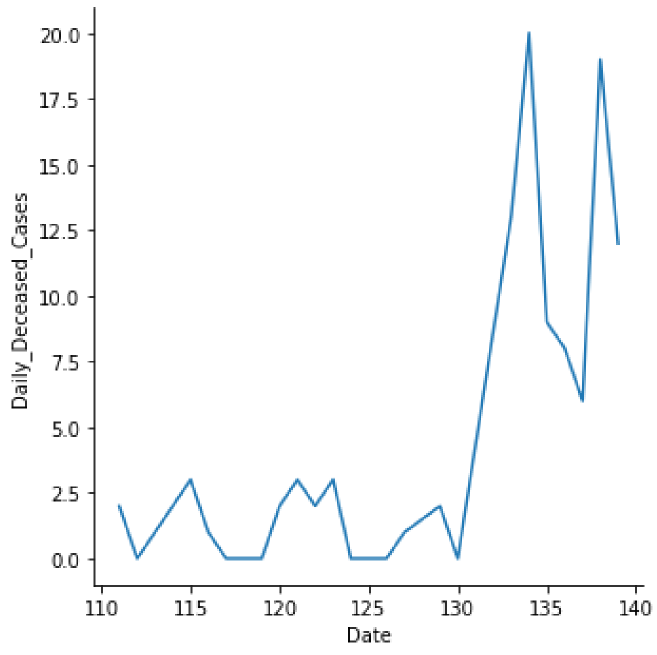
In [94]:
```
# displaying median
read_df.Daily_Deceased_Cases.median()
```

Out[94]:
```
2.0
```

```
In [95]:  # plot density curve of Daily_Deceased_Cases after cleaning null values
          read_df.Daily_Deceased_Cases.plot.density()

Out[95]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacd34eb00>
```

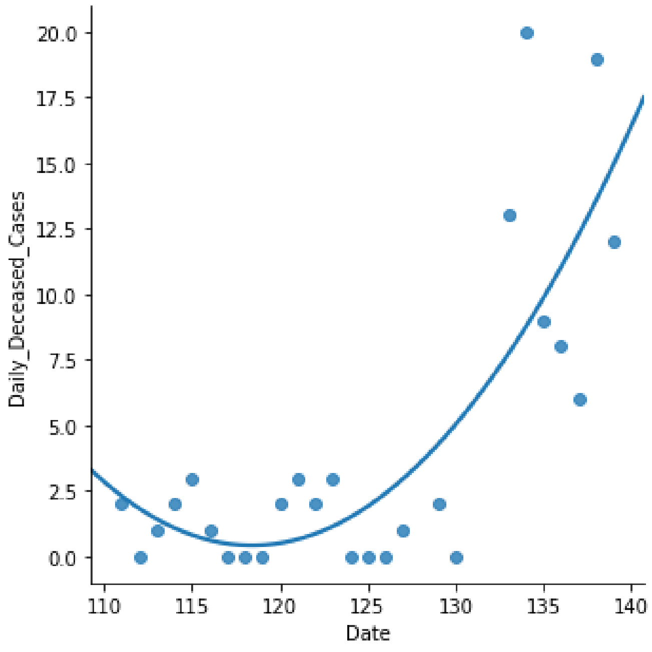


```
In [96]:  # Relplot
          sns.relplot(x="Date", y="Daily_Deceased_Cases", kind="line", data=read_df)

Out[96]:  <seaborn.axisgrid.FacetGrid at 0x1eacd39f748>
```

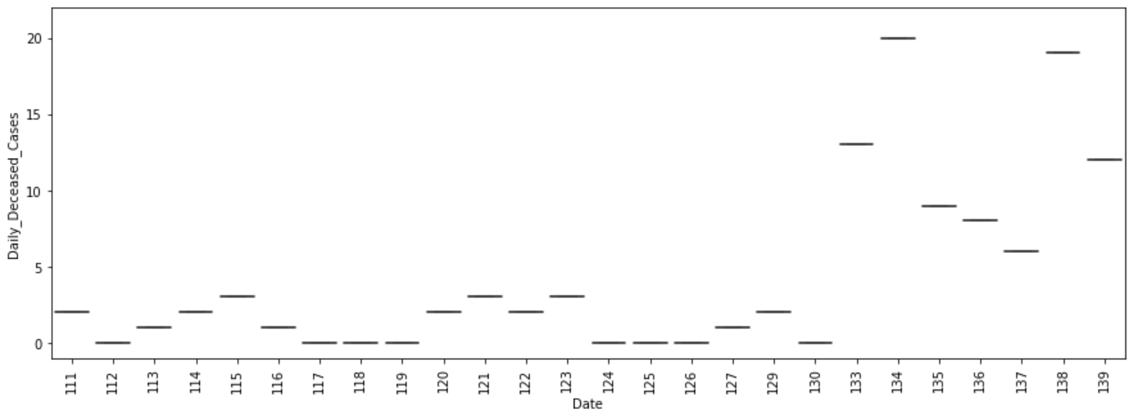


```
In [97]:  # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Daily_Deceased_Cases", data = read_df, order = 2, ci
```

```
Out[97]: <seaborn.axisgrid.FacetGrid at 0x1eacd3cccf8>
```



```
In [98]:  # Box Plot
          var = 'Date'
          data2 = pd.concat([read_df['Daily_Deceased_Cases'], read_df[var]], axis=1)
          f, ax = plt.subplots(figsize=(15, 5))
          fig = sns.boxplot(x=var, y="Daily_Deceased_Cases", data=data2)
          fig.axis(ymin=-1, ymax=22);
          plt.xticks(rotation=90);
```



11. Daily_Tests_Cases

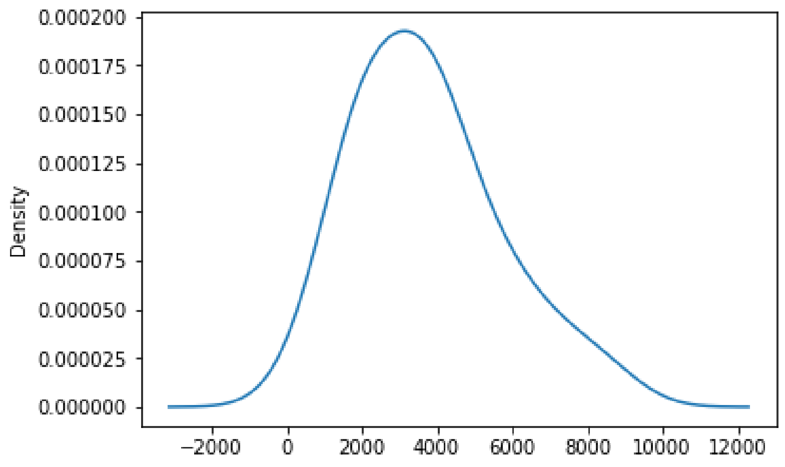
```
In [99]:  read_df.Daily_Tests_Cases.describe()

Out[99]:  count      23.000000
          mean     3721.869565
          std      1938.380142
          min       727.000000
          25%      2172.500000
          50%      3744.000000
          75%      4539.500000
          max      8431.000000
          Name: Daily_Tests_Cases, dtype: float64
```

```
In [100]:  # Displaying mean
           read_df.Daily_Tests_Cases.mean()

Out[100]:  3721.8695652173915
```

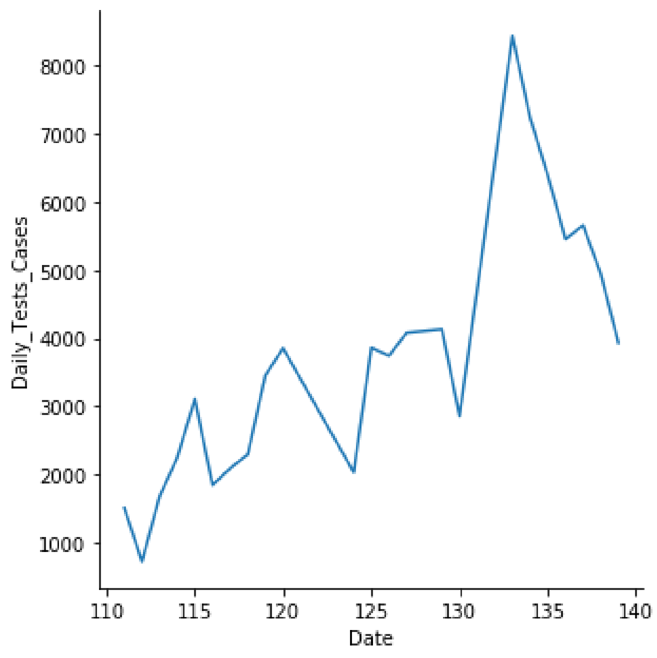
```
In [101]:  # displaying mode
           read_df.Daily_Tests_Cases.mode()

Out[101]:  0       727.0
           1      1513.0
           2      1682.0
           3      1847.0
           4      2036.0
           5      2094.0
           6      2251.0
           7      2298.0
           8      2859.0
           9      3112.0
           10     3459.0
           11     3744.0
           12     3855.0
           13     3862.0
           14     3936.0
           15     4082.0
           16     4133.0
           17     4946.0
           18     5453.0
           19     5656.0
           20     6391.0
           21     7236.0
           22     8431.0
           dtype: float64
```

```
In [102]:  # displaying median
           read_df.Daily_Tests_Cases.median()

Out[102]:  3744.0
```

```
In [103]:  # plot density curve of Daily_Tests_Cases after cleaning null values
           read_df.Daily_Tests_Cases.plot.density()

Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x1eaccf24668>
```

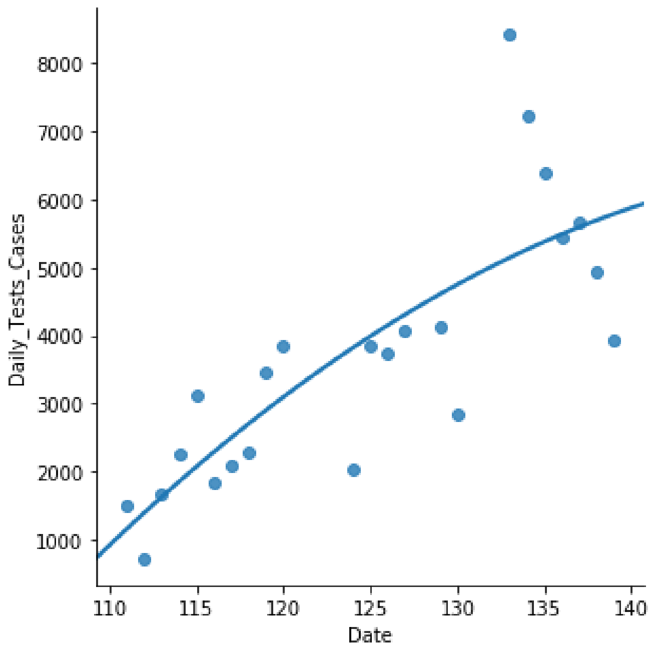


```
In [104]:  # Relplot
           sns.relplot(x="Date", y="Daily_Tests_Cases", kind="line", data=read_df)

Out[104]: <seaborn.axisgrid.FacetGrid at 0x1eaccd747b8>
```

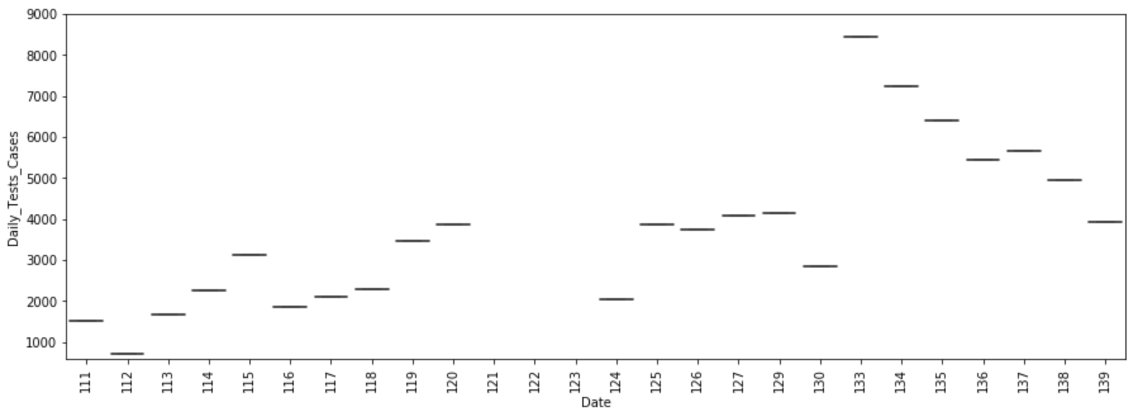


In [105]:
```
# Plotting the data scatter
sns.lmplot(x ="Date", y ="Daily_Tests_Cases", data = read_df, order = 2, ci = N
```

Out[105]:
```
<seaborn.axisgrid.FacetGrid at 0x1eaccd74cf8>
```



In [106]:
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Daily_Tests_Cases'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Daily_Tests_Cases", data=data2)
fig.axis(ymin=600, ymax=9000);
plt.xticks(rotation=90);
```



12. Doubling_Time

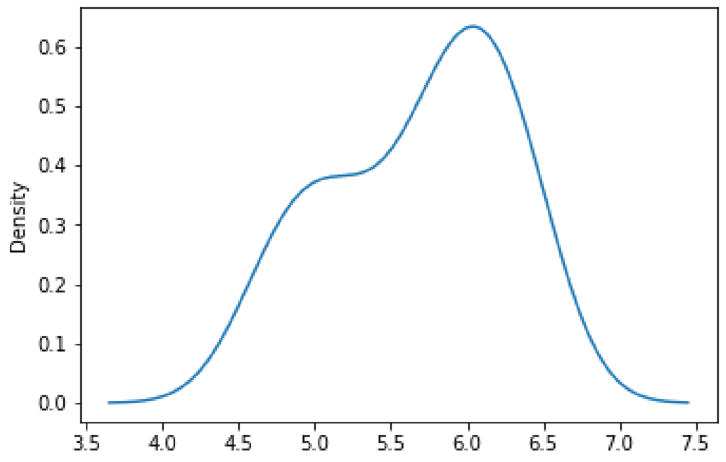
```
In [107]:  read_df.Doubling_Time.describe()

Out[107]:  count    26.000000
           mean      5.680769
           std       0.574121
           min       4.600000
           25%       5.200000
           50%       5.800000
           75%       6.100000
           max       6.500000
           Name: Doubling_Time, dtype: float64
```

```
In [108]:  # Displaying mean
           read_df.Doubling_Time.mean()

Out[108]:  5.680769230769231
```

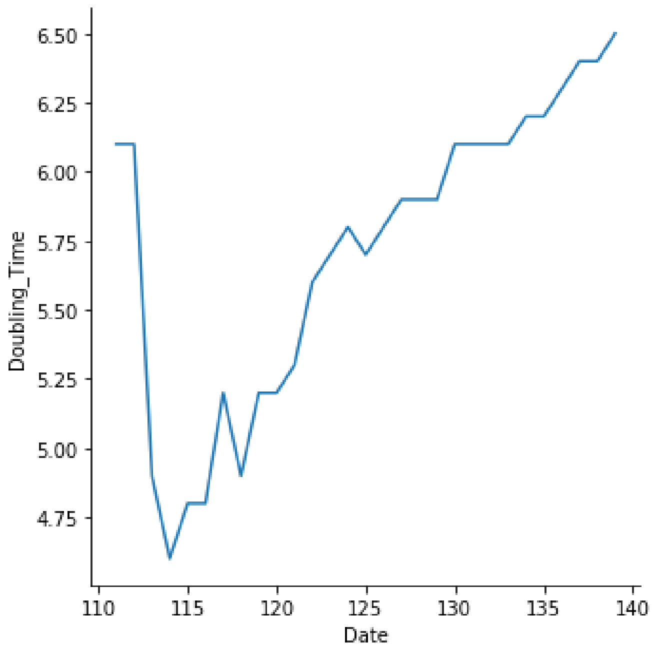
```
In [109]:  # displaying mode
           read_df.Doubling_Time.mode()

Out[109]:  0    6.1
           dtype: float64
```

```
In [110]:  # displaying median
           read_df.Doubling_Time.median()

Out[110]:  5.8
```

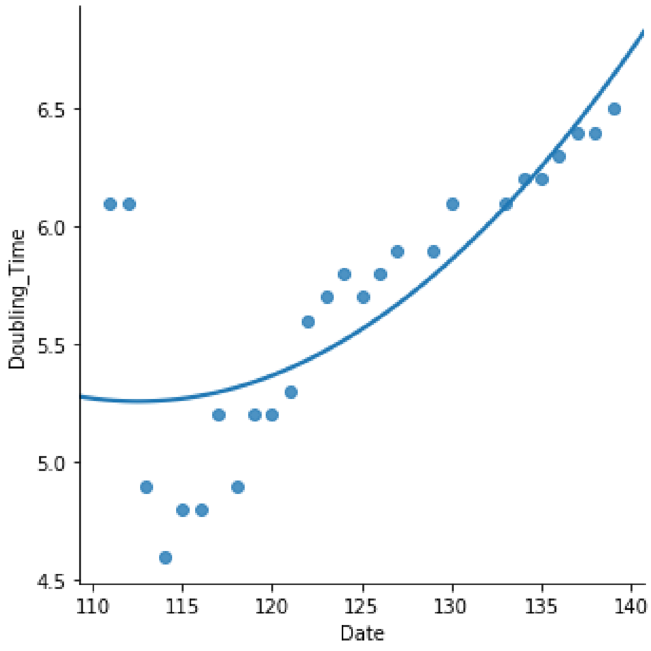
```
In [111]:  # plot density curve of Doubling_Time after cleaning null values
           read_df.Doubling_Time.plot.density()

Out[111]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eaccca7e48>
```



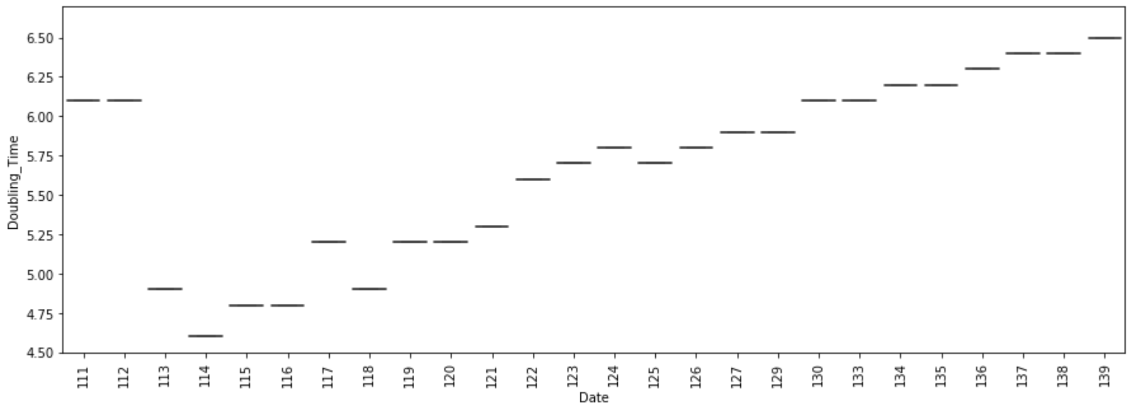
```
In [112]: # Relplot
          sns.relplot(x="Date", y="Doubling_Time", kind="line", data=read_df)

Out[112]: <seaborn.axisgrid.FacetGrid at 0x1eace86a0f0>
```



```
In [113]:  # Plotting the data scatter
            sns.lmplot(x ="Date", y ="Doubling_Time", data = read_df, order = 2, ci = None)

Out[113]:  <seaborn.axisgrid.FacetGrid at 0x1eace8d2f60>
```



```
In [114]:  # Box Plot
            var = 'Date'
            data2 = pd.concat([read_df['Doubling_Time'], read_df[var]], axis=1)
            f, ax = plt.subplots(figsize=(15, 5))
            fig = sns.boxplot(x=var, y="Doubling_Time", data=data2)
            fig.axis(ymin=4.5, ymax=6.7);
            plt.xticks(rotation=90);
```



13. Growth_Rate

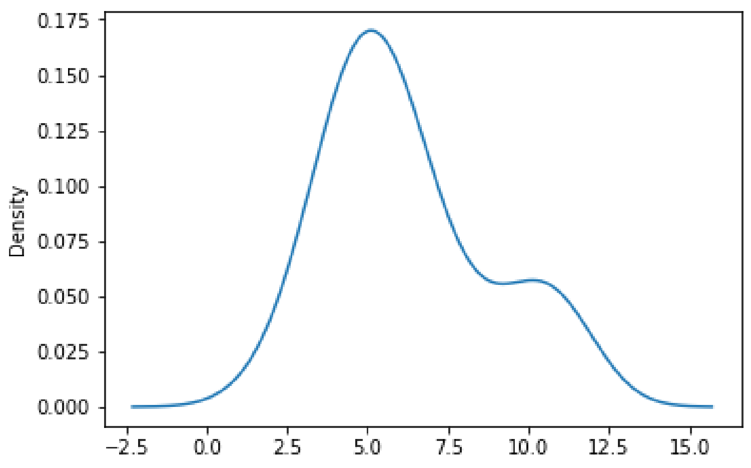
```
In [115]:  read_df.Growth_Rate.describe()

Out[115]:  count    26.000000
           mean      6.219231
           std       2.489019
           min       2.200000
           25%       4.500000
           50%       5.750000
           75%       7.425000
           max      11.200000
           Name: Growth_Rate, dtype: float64
```

```
In [116]:  # Displaying mean
           read_df.Growth_Rate.mean()

Out[116]:  6.2192307692307685
```

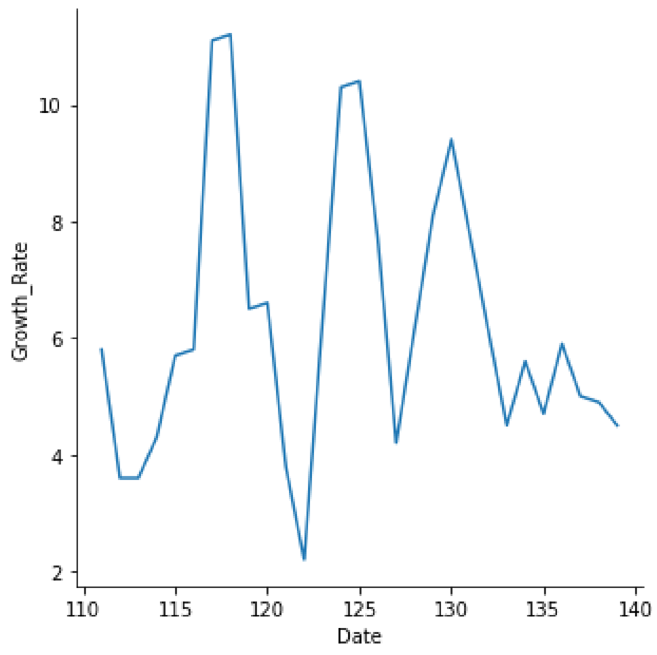
```
In [117]:  # displaying mode
           read_df.Growth_Rate.mode()

Out[117]:  0    3.6
           1    4.5
           2    5.8
           dtype: float64
```

```
In [118]:  # displaying median
           read_df.Growth_Rate.median()

Out[118]:  5.75
```

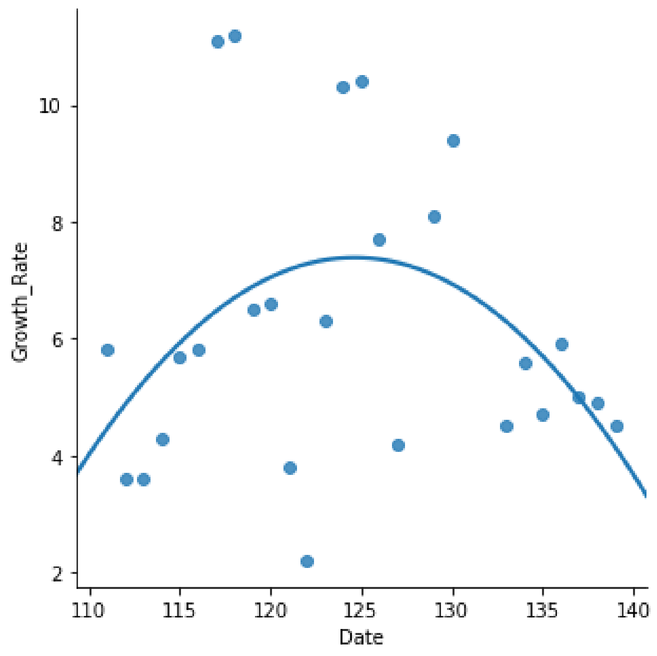
```
In [119]:  # plot density curve of Growth_Rate after cleaning null values
           read_df.Growth_Rate.plot.density()

Out[119]:  <matplotlib.axes._subplots.AxesSubplot at 0x1eacfcef780>
```



```
In [120]:  # Relplot
           sns.relplot(x="Date", y="Growth_Rate", kind="line", data=read_df)

Out[120]: <seaborn.axisgrid.FacetGrid at 0x1eacfd0a9e8>
```

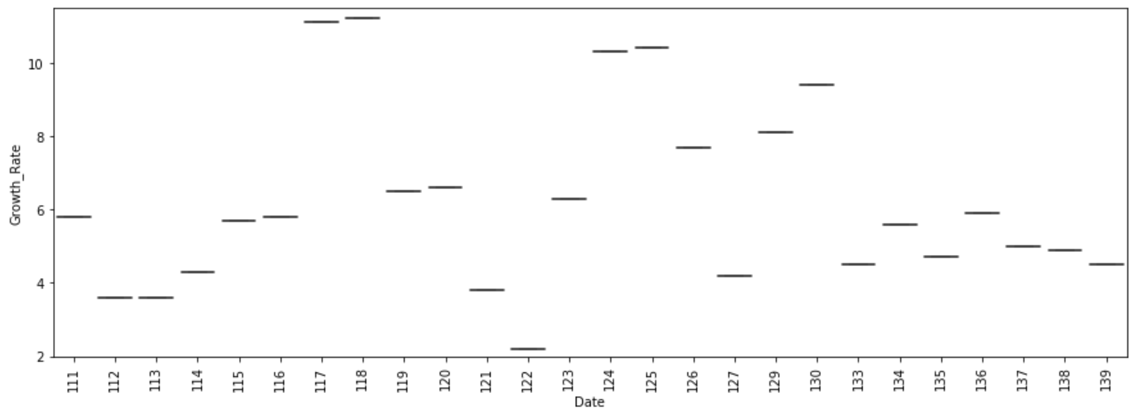


```
In [121]:  # Plotting the data scatter
           sns.lmplot(x ="Date", y ="Growth_Rate", data = read_df, order = 2, ci = None)

Out[121]: <seaborn.axisgrid.FacetGrid at 0x1eacb8c0c50>
```



In [122]:

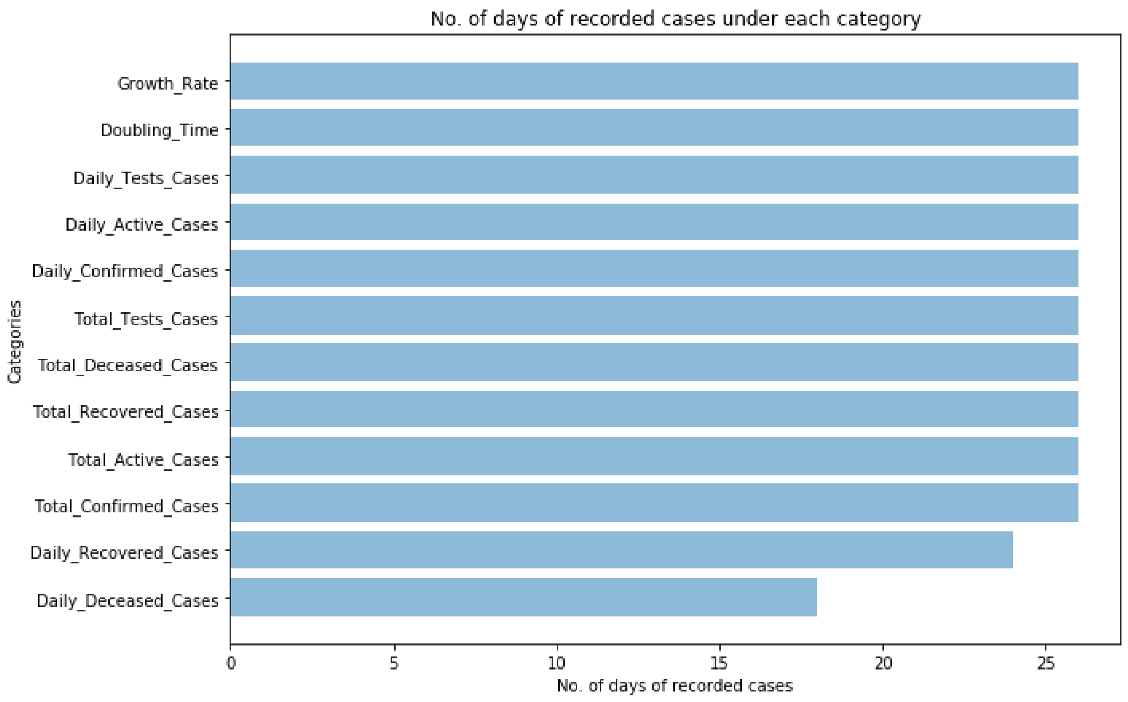
```
# Box Plot
var = 'Date'
data2 = pd.concat([read_df['Growth_Rate'], read_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 5))
fig = sns.boxplot(x=var, y="Growth_Rate", data=data2)
fig.axis(ymin=2, ymax=11.5);
plt.xticks(rotation=90);
```



Plotting All Columns

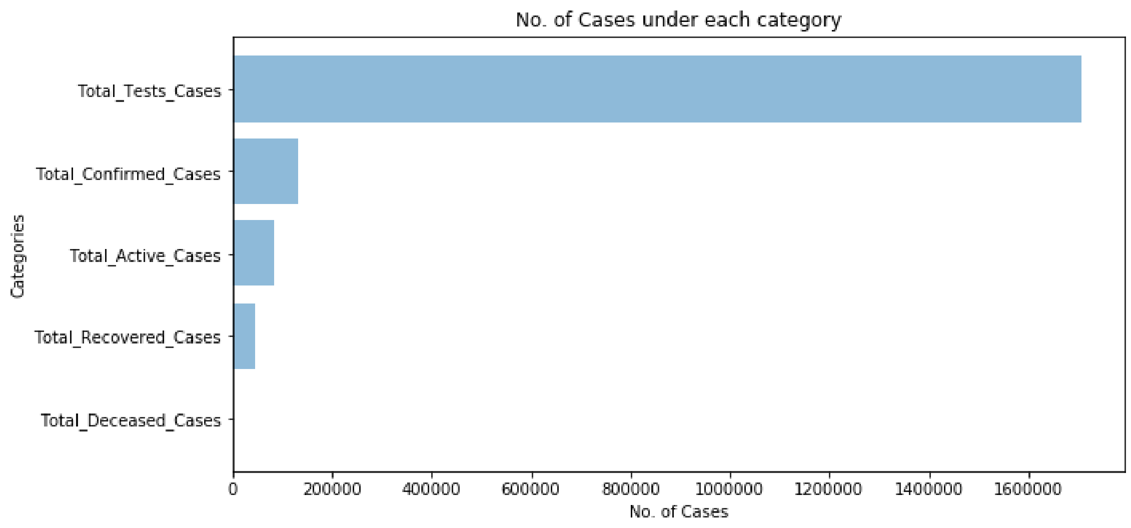
In [123]:
```
# number of user have given ratings to each category; Gyms and Bakeries having
no_of_zeros = read_df[column_names[1:]].astype(bool).sum(axis=0).sort_values()


plt.figure(figsize=(10,7))

plt.barh(np.arange(len(column_names[1:])), no_of_zeros.values, align='center',

plt.yticks(np.arange(len(column_names[1:])), no_of_zeros.index)

plt.xlabel('No. of days of recorded cases')

plt.ylabel('Categories')

plt.title('No. of days of recorded cases under each category')
```

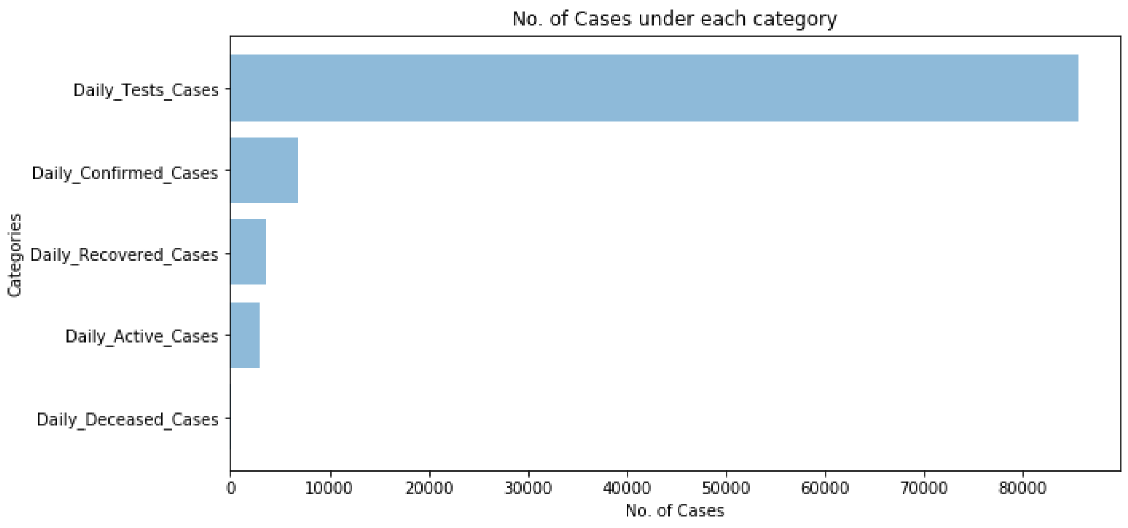
Out[123]:
```
Text(0.5, 1.0, 'No. of days of recorded cases under each category')
```

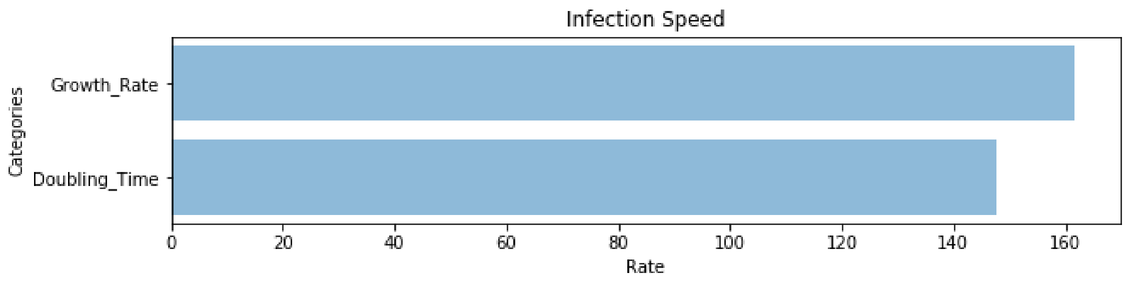


In [124]:
```
# number of user have given ratings to each category; Gyms and Bakeries having
no_of_zeros = read_df[column_names[1:6]].sum(axis=0).sort_values()


plt.figure(figsize=(10,5))

plt.barh(np.arange(len(column_names[1:6])), no_of_zeros.values, align='center',

plt.yticks(np.arange(len(column_names[1:6])), no_of_zeros.index)

plt.xlabel('No. of Cases')

plt.ylabel('Categories')

plt.title('No. of Cases under each category')
```

Out[124]:
```
Text(0.5, 1.0, 'No. of Cases under each category')
```



In [125]:
```
# number of user have given ratings to each category; Gyms and Bakeries having
no_of_zeros = read_df[column_names[6:11]].sum(axis=0).sort_values()


plt.figure(figsize=(10,5))

plt.barh(np.arange(len(column_names[6:11])), no_of_zeros.values, align='center'

plt.yticks(np.arange(len(column_names[6:11])), no_of_zeros.index)

plt.xlabel('No. of Cases')

plt.ylabel('Categories')

plt.title('No. of Cases under each category')
```
Out[125]:
```
Text(0.5, 1.0, 'No. of Cases under each category')
```


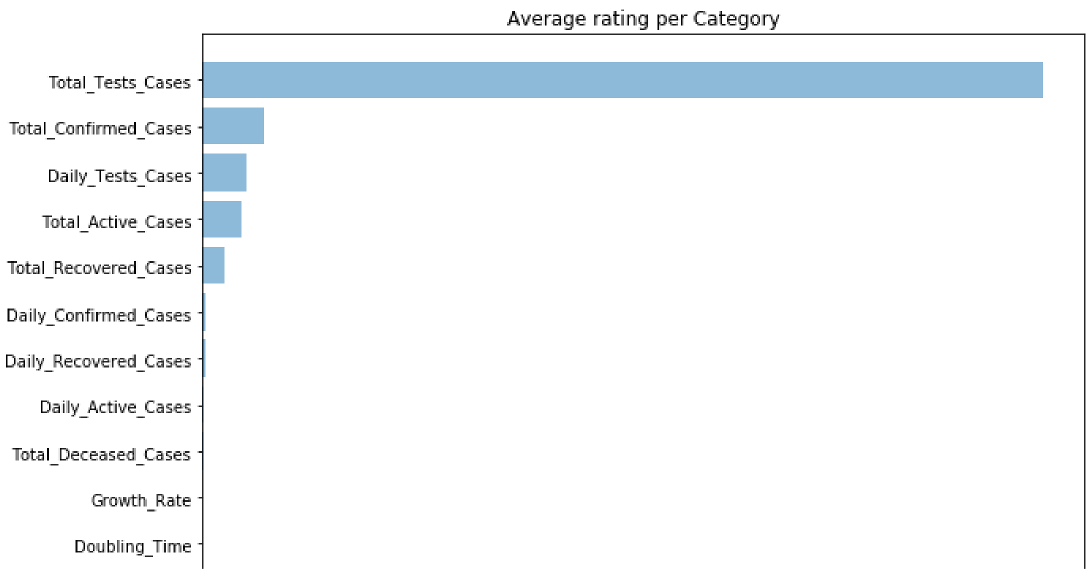
In [126]:
```
# number of user have given ratings to each category; Gyms and Bakeries having
no_of_zeros = read_df[column_names[11:13]].sum(axis=0).sort_values()


plt.figure(figsize=(10,2))

plt.barh(np.arange(len(column_names[11:13])), no_of_zeros.values, align='center

plt.yticks(np.arange(len(column_names[11:13])), no_of_zeros.index)

plt.xlabel('Rate')

plt.ylabel('Categories')

plt.title('Infection Speed')
```
Out[126]:
```
Text(0.5, 1.0, 'Infection Speed')
```


```
In [127]:  avg_rating = read_df[column_names[1:]].mean()
           avg_rating = avg_rating.sort_values()
           avg_rating
```

```
Out[127]:  Daily_Deceased_Cases         4.192308
           Doubling_Time                5.680769
           Growth_Rate                  6.219231
           Total_Deceased_Cases        75.423077
           Daily_Active_Cases         117.923077
           Daily_Recovered_Cases      142.192308
           Daily_Confirmed_Cases      265.846154
           Total_Recovered_Cases     1774.230769
           Total_Active_Cases        3283.615385
           Daily_Tests_Cases         3721.869565
           Total_Confirmed_Cases     5133.269231
           Total_Tests_Cases        71141.291667
           dtype: float64
```

```
In [128]:  plt.figure(figsize=(10,7))
           plt.barh(np.arange(len(column_names[1:])), avg_rating.values, align='center', a
           plt.yticks(np.arange(len(column_names[1:])), avg_rating.index)
           plt.xlabel('Average Rating')
           plt.title('Average rating per Category')
```

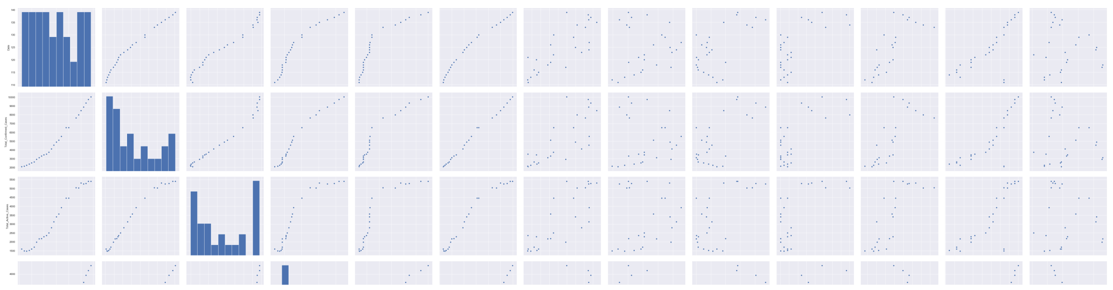
```
Out[128]:  Text(0.5, 1.0, 'Average rating per Category')
```



In [129]:

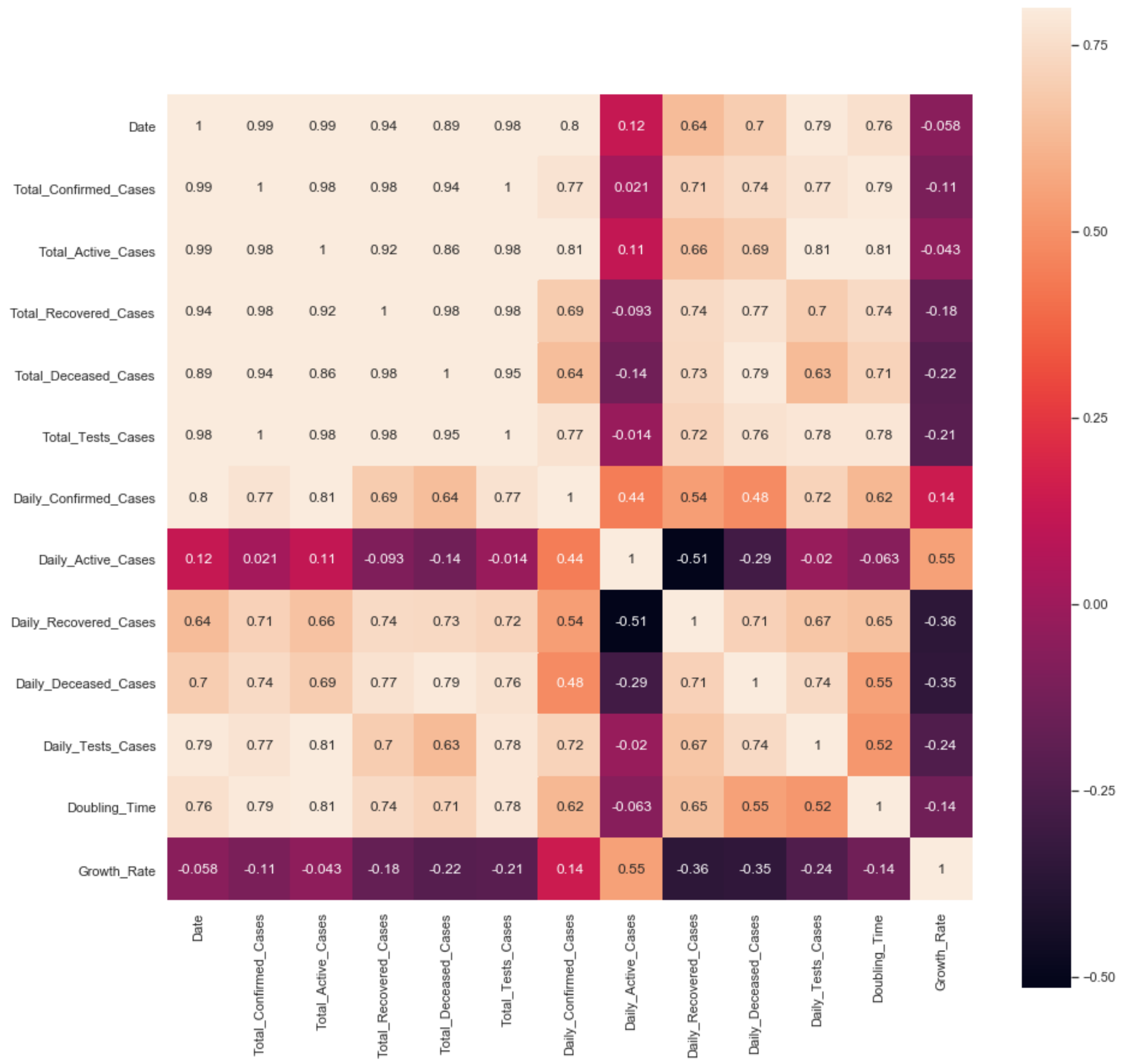
```
#scatterplot
sns.set()
sns.pairplot(read_df[column_names], size = 5.0)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: UserWa
rning: The `size` parameter has been renamed to `height`; pleaes update you
r code.
  warnings.warn(msg, UserWarning)
C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: Run
timeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: Run
timeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```



In [130]:

```python
#correlation matrix
corrmat_read_df = read_df.corr()
f, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(corrmat_read_df, vmax=.8, square=True, annot=True);
```

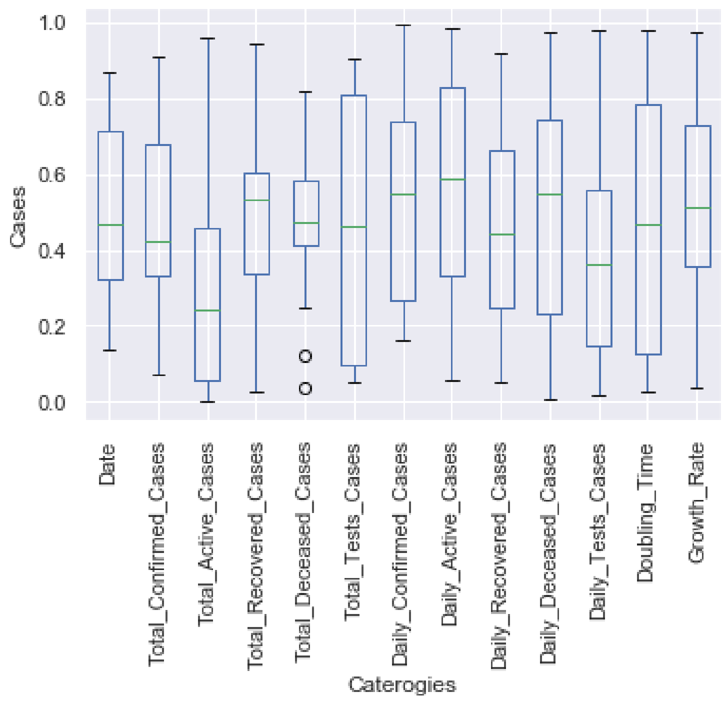


	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Cases	Total_Tests_Cases	Daily_Confirmed_Cases	Daily_Active_Cases	Daily_Recovered_Cases	Daily_Deceased_Cases	Daily_Tests_Cases	Doubling_Time	Growth_Rate
Date	1	0.99	0.99	0.94	0.89	0.98	0.8	0.12	0.64	0.7	0.79	0.76	-0.058
Total_Confirmed_Cases	0.99	1	0.98	0.98	0.94	1	0.77	0.021	0.71	0.74	0.77	0.79	-0.11
Total_Active_Cases	0.99	0.98	1	0.92	0.86	0.98	0.81	0.11	0.66	0.69	0.81	0.81	-0.043
Total_Recovered_Cases	0.94	0.98	0.92	1	0.98	0.98	0.69	-0.093	0.74	0.77	0.7	0.74	-0.18
Total_Deceased_Cases	0.89	0.94	0.86	0.98	1	0.95	0.64	-0.14	0.73	0.79	0.63	0.71	-0.22
Total_Tests_Cases	0.98	1	0.98	0.98	0.95	1	0.77	-0.014	0.72	0.76	0.78	0.78	-0.21
Daily_Confirmed_Cases	0.8	0.77	0.81	0.69	0.64	0.77	1	0.44	0.54	0.48	0.72	0.62	0.14
Daily_Active_Cases	0.12	0.021	0.11	-0.093	-0.14	-0.014	0.44	1	-0.51	-0.29	-0.02	-0.063	0.55
Daily_Recovered_Cases	0.64	0.71	0.66	0.74	0.73	0.72	0.54	-0.51	1	0.71	0.67	0.65	-0.36
Daily_Deceased_Cases	0.7	0.74	0.69	0.77	0.79	0.76	0.48	-0.29	0.71	1	0.74	0.55	-0.35
Daily_Tests_Cases	0.79	0.77	0.81	0.7	0.63	0.78	0.72	-0.02	0.67	0.74	1	0.52	-0.24
Doubling_Time	0.76	0.79	0.81	0.74	0.71	0.78	0.62	-0.063	0.65	0.55	0.52	1	-0.14
Growth_Rate	-0.058	-0.11	-0.043	-0.18	-0.22	-0.21	0.14	0.55	-0.36	-0.35	-0.24	-0.14	1

In [131]:
```python
df = pd.DataFrame(np.random.rand(13,13), columns = read_df.columns)
df.plot.box(grid = 'True', rot = 90)
plt.xlabel('Caterogies')
plt.ylabel('Cases')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:424: M
atplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

Out[131]:
```
Text(0, 0.5, 'Cases')
```



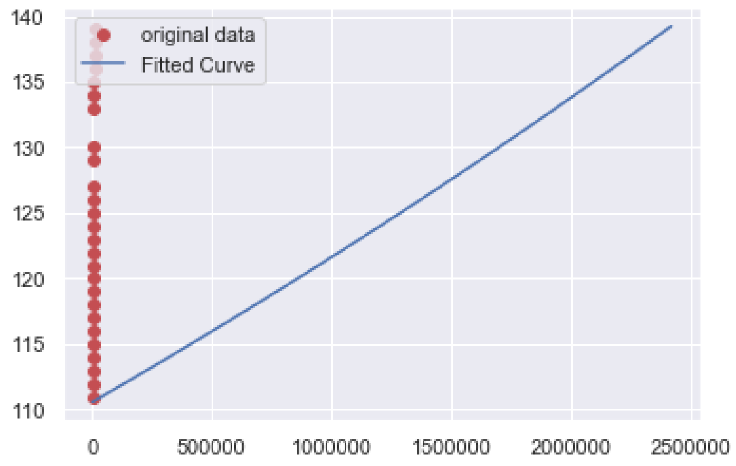
Logistic Regression

In [132]:
```python
def logistic_model(x,a,b,c):
    return c/(1+np.exp(-(x-b)/a))
```

In [133]:
```python
x = list(read_df.iloc[:,0])
y = list(read_df.iloc[:,1])
```

In [134]:
```
print("x: ", x)
print("")
print("y: ", y)
```
```
x:  [111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 12
5, 126, 127, 129, 130, 133, 134, 135, 136, 137, 138, 139]

y:  [2081, 2156, 2248, 2376, 2514, 2625, 2918, 3108, 3314, 3439, 3515, 3738, 4
122, 4549, 4898, 5104, 5532, 6542, 6542, 7639, 7998, 8470, 8895, 9333, 9755, 1
0054]
```

In [135]:
```
fit = curve_fit(logistic_model, x, y, p0=[6,150,20000], maxfev=1000)
fit
```
Out[135]:
```
(array([1.27105840e+01, 1.41284526e+02, 2.23263571e+04]),
 array([[3.47826201e-01, 1.59334970e+00, 1.52642508e+03],
        [1.59334970e+00, 7.61500756e+00, 7.36052815e+03],
        [1.52642508e+03, 7.36052815e+03, 7.13800564e+06]]))
```

In [136]:
```
plt.plot(y,x,'ro',label="original data")

t=np.linspace(0,3600*24*28,26)

popt, pcov = optimize.curve_fit(logistic_model, t, x, maxfev=20000)

#plt.plot(t, logistic_model(t, *fit), label="Fitted Curve")
plt.plot(t, logistic_model(t, *popt), label="Fitted Curve")

plt.legend(loc='upper left')
plt.show()
```


In [137]:
```
errors = [np.sqrt(fit[1][i][i]) for i in [0,1,2]]
errors
```
Out[137]:
```
[0.5897679209216782, 2.7595303155304656, 2671.704630976988]
```

In [138]:
```
a = read_df.iloc[-1, 11]
```

In [139]:
```
# Get the maximum element from a Numpy array
maxElement = np.amax(read_df.Daily_Confirmed_Cases)
print('Max element from Numpy Array : ', maxElement)

# Get the indices of maximum element in numpy array
result = np.where(read_df.Daily_Confirmed_Cases == maxElement)
print('List of Indices of maximum element :', result[0])
```

```
Max element from Numpy Array :  472
List of Indices of maximum element : [21]
```

In [140]:
```
# Get the day whose index of new case and date is same
b = read_df.iloc[21, 0]
```

In [141]:
```
c = read_df.iloc[-1, 1]
```

In [142]:
```
print("Here,")
print("  a: Infection Speed (Doubling Time): ", a)
print("  b: Day with maximum infection occured: ", b )
print("  c: Total number of recorded infected people at the infection's end: ",
```

```
Here,
  a: Infection Speed (Doubling Time):  6.5
  b: Day with maximum infection occured:  135
  c: Total number of recorded infected people at the infection's end:  10054
```

In [143]:
```
sol = int(fsolve(lambda x : logistic_model(x,a,b,c) - int(c),b))
sol
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\minpack.py:162: Runt
imeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

Out[143]: 368

Exponential model

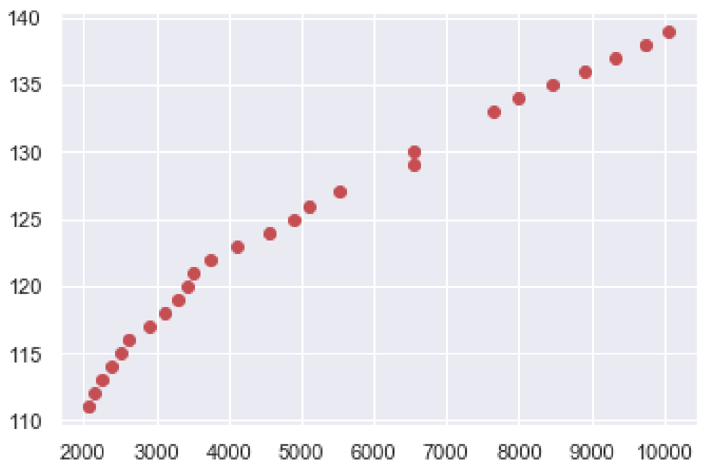
In [144]:
```
def exponential_model(x,a,b,c):
    return a*np.exp(b*(x-c))
```

In [145]:
```
exp_fit = curve_fit(exponential_model, x, y, p0=[1,0,1], maxfev=20000)
errors = [np.sqrt(exp_fit[1][i][i]) for i in [0,1,2]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWar
ning: invalid value encountered in sqrt
```

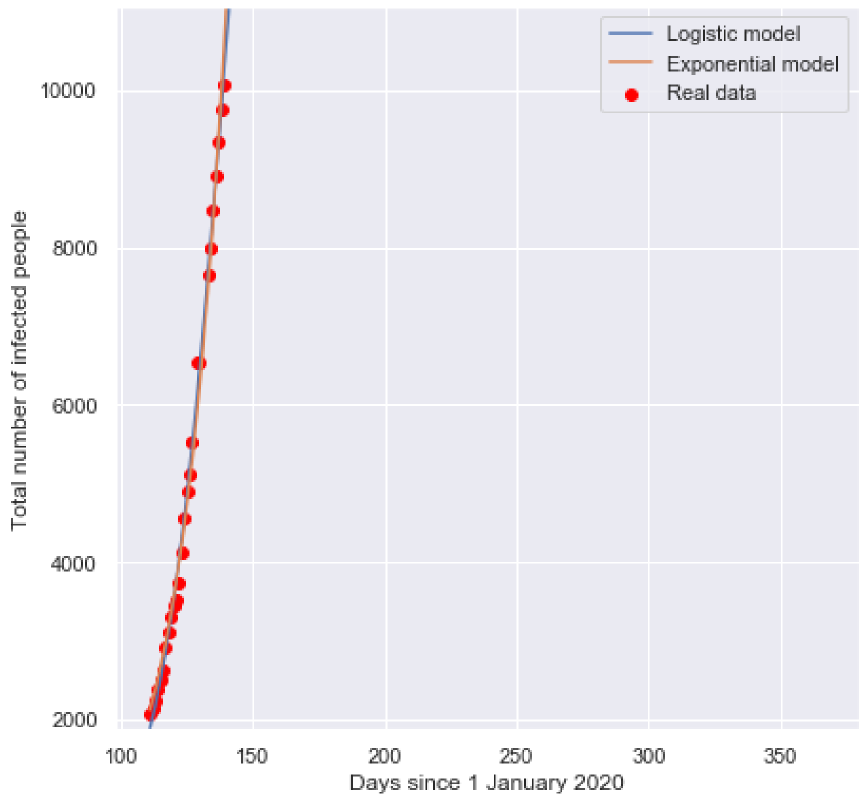
In [146]:
```
plt.plot(y,x,'ro',label="original data")

popt, pcov = optimize.curve_fit(exponential_model, y, x, maxfev=20000)
t=np.linspace(0,3600*24*28,11)

plt.plot(t, exponential_model(t, *popt), label="Fitted Curve")

plt.legend(loc='upper left')
plt.show()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-146-f82b9c20f4c2> in <module>
      1 plt.plot(y,x,'ro',label="original data")
      2 
----> 3 popt, pcov = optimize.curve_fit(exponential_model, y, x, maxfev=20000)
      4 t=np.linspace(0,3600*24*28,11)
      5 

C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\minpack.py in curve_
fit(f, xdata, ydata, p0, sigma, absolute_sigma, check_finite, bounds, method,
 jac, **kwargs)
    746         cost = np.sum(infodict['fvec'] ** 2)
    747         if ier not in [1, 2, 3, 4]:
--> 748             raise RuntimeError("Optimal parameters not found: " + errm
sg)
    749     else:
    750         # Rename maxfev (leastsq) to max_nfev (least_squares), if spec
ified.

RuntimeError: Optimal parameters not found: Number of calls to function has re
ached maxfev = 20000.
```



Plots

```
In [147]:  pred_x = list(range(max(x),sol))
           plt.rcParams['figure.figsize'] = [7, 7]
           plt.rc('font', size=14)

           # Real data
           plt.scatter(x,y,label="Real data",color="red")

           # Predicted logistic curve
           plt.plot(x+pred_x, [logistic_model(i,fit[0][0],fit[0][1],fit[0][2]) for i in x+

           # Predicted exponential curve
           plt.plot(x+pred_x, [exponential_model(i,exp_fit[0][0],exp_fit[0][1],exp_fit[0][
                    label="Exponential model" )
           plt.legend()
           plt.xlabel("Days since 1 January 2020")
           plt.ylabel("Total number of infected people")
           plt.ylim((min(y)*0.9,c*1.1))
           plt.show()
```



Analysis of residuals

```
In [148]:  y_pred_logistic = [logistic_model(i,fit[0][0],fit[0][1],fit[0][2]) for i in x]
           y_pred_logistic
```

```
Out[148]: [1886.7432281928363,
           2027.155271810352,
           2176.903909030907,
           2336.4414229483396,
           2506.2166126985812,
           2686.670300096075,
           2878.230266130926,
           3081.3056167498535,
           3296.2805925331104,
           3523.5078549986606,
           3763.3013032395666,
           4015.9284981759665,
           4281.602797406997,
           4560.475330759499,
           4852.626974159472,
           5158.060506136048,
           5476.693155589732,
           6152.756845087749,
           6509.53767203616,
           7648.745397143405,
           8049.1024733169,
           8458.339059808613,
           8875.447840397072,
           9299.332886410963,
           9728.819906182485,
           10162.668367394974]
```

```
In [149]:  y_pred_exp =  [exponential_model(i,exp_fit[0][0], exp_fit[0][1], exp_fit[0][2])
           y_pred_exp
```

```
Out[149]: [2122.860665957763,
           2247.6077610085013,
           2379.6854538573657,
           2519.5245173737044,
           2667.581038223918,
           2824.33790440323,
           2990.3063801802473,
           3166.0277735910936,
           3352.075201921609,
           3549.0554609357714,
           3757.611003946858,
           3978.422037186091,
           4212.208738302857,
           4459.733605232146,
           4721.803943090079,
           4999.274497208543,
           5293.050240896666,
           5933.406213033405,
           6282.074969632171,
           7455.9023867103115,
           7894.038614225908,
           8357.92133678733,
           8849.0635130716,
           9369.06700876835,
           9919.627821082548,
           10502.541610248376]
```

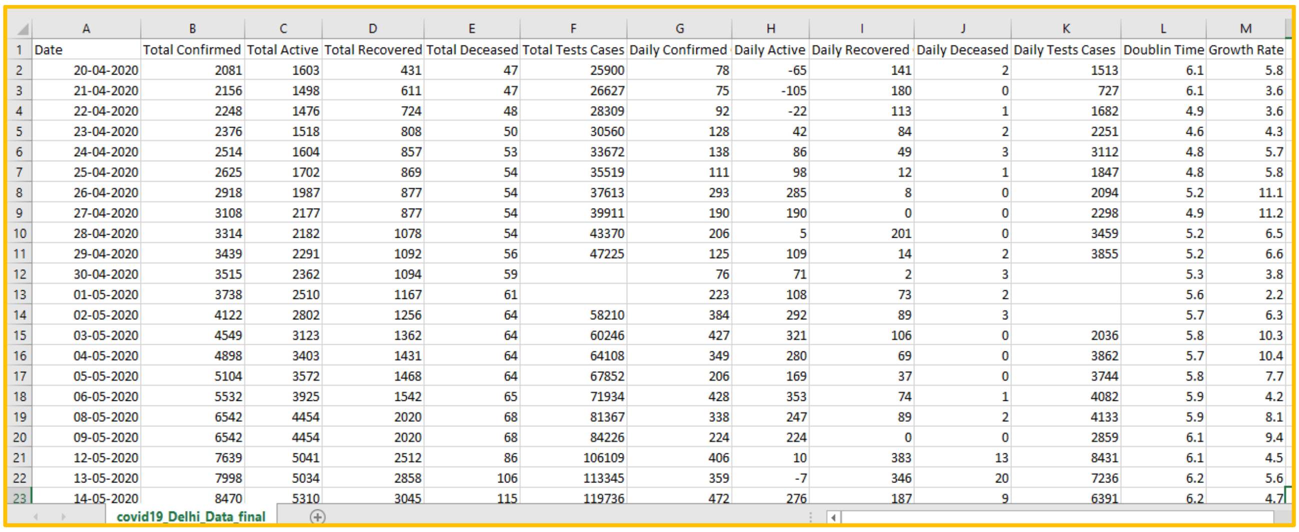
```
In [150]: mean_squared_error(y,y_pred_logistic)
Out[150]: 16020.339190962124
```

```
In [151]: mean_squared_error(y,y_pred_exp)
Out[151]: 41999.5309910943
```

Outputs

Dataset

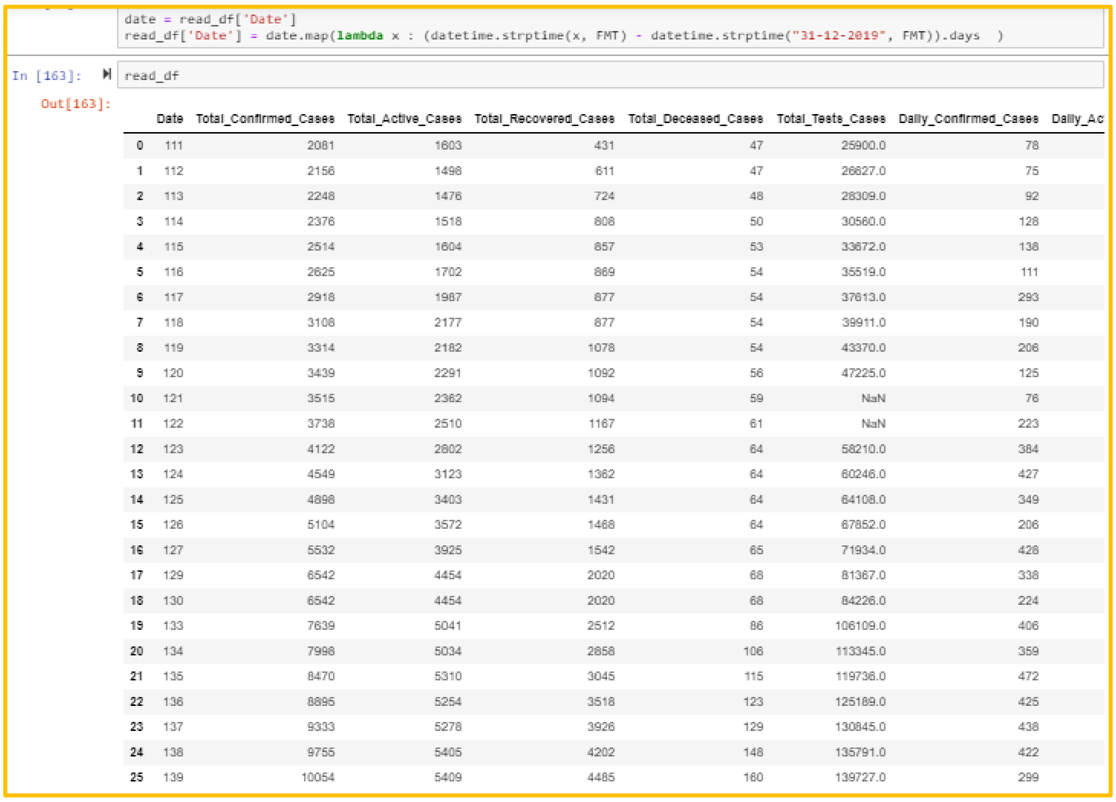
a) Raw Dataset:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Date	Total Confirmed	Total Active	Total Recovered	Total Deceased	Total Tests Cases	Daily Confirmed	Daily Active	Daily Recovered	Daily Deceased	Daily Tests Cases	Doublin Time	Growth Rate
2	20-04-2020	2081	1603	431	47	25900	78	-65	141	2	1513	6.1	5.8
3	21-04-2020	2156	1498	611	47	26627	75	-105	180	0	727	6.1	3.6
4	22-04-2020	2248	1476	724	48	28309	92	-22	113	1	1682	4.9	3.6
5	23-04-2020	2376	1518	808	50	30560	128	42	84	2	2251	4.6	4.3
6	24-04-2020	2514	1604	857	53	33672	138	86	49	3	3112	4.8	5.7
7	25-04-2020	2625	1702	869	54	35519	111	98	12	1	1847	4.8	5.8
8	26-04-2020	2918	1987	877	54	37613	293	285	8	0	2094	5.2	11.1
9	27-04-2020	3108	2177	877	54	39911	190	190	0	0	2298	4.9	11.2
10	28-04-2020	3314	2182	1078	54	43370	206	5	201	0	3459	5.2	6.5
11	29-04-2020	3439	2291	1092	56	47225	125	109	14	2	3855	5.2	6.6
12	30-04-2020	3515	2362	1094	59		76	71	2	3		5.3	3.8
13	01-05-2020	3738	2510	1167	61		223	108	73	2		5.6	2.2
14	02-05-2020	4122	2802	1256	64	58210	384	292	89	3		5.7	6.3
15	03-05-2020	4549	3123	1362	64	60246	427	321	106	0	2036	5.8	10.3
16	04-05-2020	4898	3403	1431	64	64108	349	280	69	0	3862	5.7	10.4
17	05-05-2020	5104	3572	1468	64	67852	206	169	37	0	3744	5.8	7.7
18	06-05-2020	5532	3925	1542	65	71934	428	353	74	1	4082	5.9	4.2
19	08-05-2020	6542	4454	2020	68	81367	338	247	89	2	4133	5.9	8.1
20	09-05-2020	6542	4454	2020	68	84226	224	224	0	0	2859	6.1	9.4
21	12-05-2020	7639	5041	2512	86	106109	406	10	383	13	8431	6.1	4.5
22	13-05-2020	7998	5034	2858	106	113345	359	-7	346	20	7236	6.2	5.6
23	14-05-2020	8470	5310	3045	115	119736	472	276	187	9	6391	6.2	4.7

covid19_Delhi_Data_final

b) Dataset after applying DateTime function (We'll take the days since January 1st):



```
date = read_df['Date']
read_df['Date'] = date.map(lambda x : (datetime.strptime(x, FMT) - datetime.strptime("31-12-2019", FMT)).days  )
```

In [163]:
```
read_df
```

Out[163]:

	Date	Total_Confirmed_Cases	Total_Active_Cases	Total_Recovered_Cases	Total_Deceased_Cases	Total_Tests_Cases	Daily_Confirmed_Cases	Daily_Ac
0	111	2081	1603	431	47	25900.0	78	
1	112	2156	1498	611	47	26627.0	75	
2	113	2248	1476	724	48	28309.0	92	
3	114	2376	1518	808	50	30560.0	128	
4	115	2514	1604	857	53	33672.0	138	
5	116	2625	1702	869	54	35519.0	111	
6	117	2918	1987	877	54	37613.0	293	
7	118	3108	2177	877	54	39911.0	190	
8	119	3314	2182	1078	54	43370.0	206	
9	120	3439	2291	1092	56	47225.0	125	
10	121	3515	2362	1094	59	NaN	76	
11	122	3738	2510	1167	61	NaN	223	
12	123	4122	2802	1256	64	58210.0	384	
13	124	4549	3123	1362	64	60246.0	427	
14	125	4898	3403	1431	64	64108.0	349	
15	126	5104	3572	1468	64	67852.0	206	
16	127	5532	3925	1542	65	71934.0	428	
17	129	6542	4454	2020	68	81367.0	338	
18	130	6542	4454	2020	68	84226.0	224	
19	133	7639	5041	2512	86	106109.0	406	
20	134	7998	5034	2858	106	113345.0	359	
21	135	8470	5310	3045	115	119736.0	472	
22	136	8895	5254	3518	123	125189.0	425	
23	137	9333	5278	3926	129	130845.0	438	
24	138	9755	5405	4202	148	135791.0	422	
25	139	10054	5409	4485	160	139727.0	299	

EDA (Exploratory Data Analysis)

1. load data in Jupyter Notebook using pandas: Df=pd.read_csv('covid19_Delhi_Data.csv')

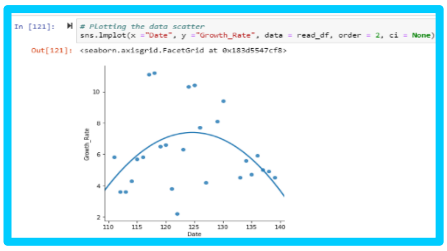
2. to check the shape of data: read_df.shape

3. print first 5 rows of data read_df.head()

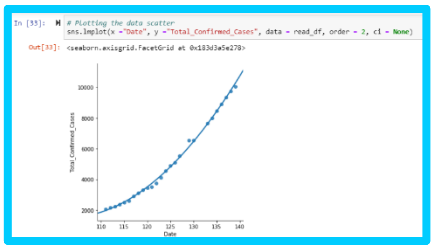
4. change the column name: read_df.columns = column_names

column_names = ['Date', 'Total_Confirmed_Cases', 'Total_Active_Cases', 'Total_Recovered_Cases', 'Total_Deceased_Cases', 'Total_Tests_Cases', 'Daily_Confirmed_Cases', 'Daily_Active_Cases', 'Daily_Recovered_Cases', 'Daily_Deceased_Cases', 'Daily_Tests_Cases', 'Doubling_Time', 'Growth_Rate']

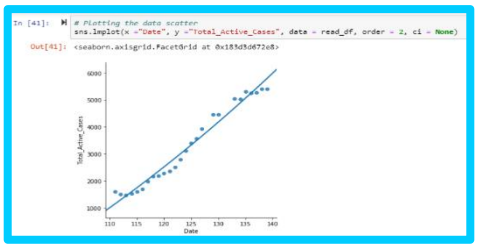
5. Describing Date column and Plotting the data scatter on each column:



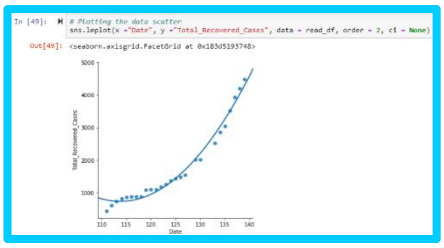
```
In [121]: # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Growth_Rate", data = read_df, order = 2, ci = None)
Out[121]: <seaborn.axisgrid.FacetGrid at 0x183d5547cf8>
```



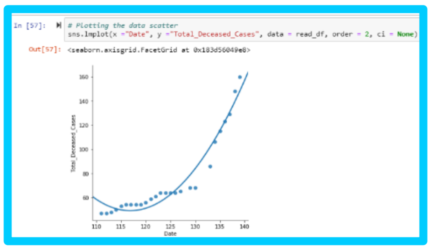
```
In [33]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Total_Confirmed_Cases", data = read_df, order = 2, ci = None)
Out[33]: <seaborn.axisgrid.FacetGrid at 0x183d3a5e278>
```



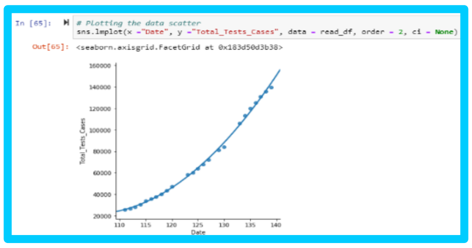
```
In [41]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Total_Active_Cases", data = read_df, order = 2, ci = None)
Out[41]: <seaborn.axisgrid.FacetGrid at 0x183d3d672e8>
```



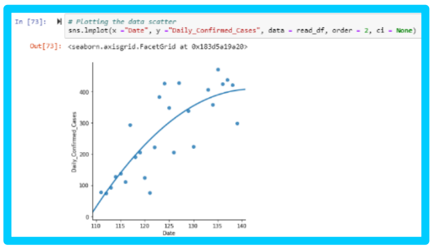
```
In [49]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Total_Recovered_Cases", data = read_df, order = 2, ci = None)
Out[49]: <seaborn.axisgrid.FacetGrid at 0x183d5193748>
```



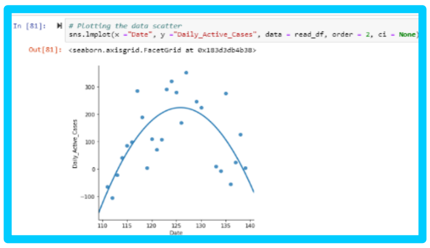
```
In [57]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Total_Deceased_Cases", data = read_df, order = 2, ci = None)
Out[57]: <seaborn.axisgrid.FacetGrid at 0x183d56049e8>
```



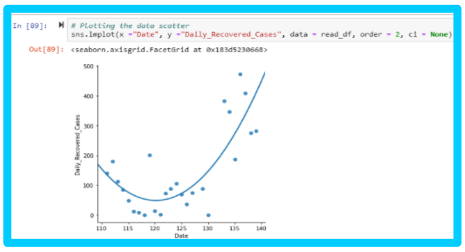
```
In [65]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Total_Tests_Cases", data = read_df, order = 2, ci = None)
Out[65]: <seaborn.axisgrid.FacetGrid at 0x183d50d3b38>
```



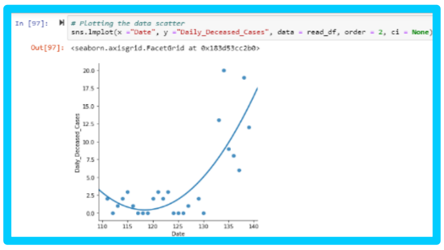
```
In [73]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Daily_Confirmed_Cases", data = read_df, order = 2, ci = None)
Out[73]: <seaborn.axisgrid.FacetGrid at 0x183d5a19a20>
```



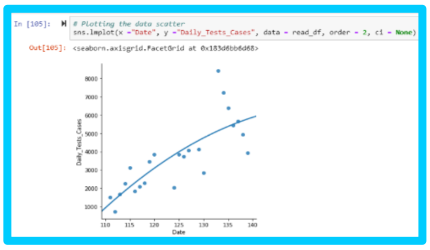
```
In [81]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Daily_Active_Cases", data = read_df, order = 2, ci = None)
Out[81]: <seaborn.axisgrid.FacetGrid at 0x183d3db4b38>
```



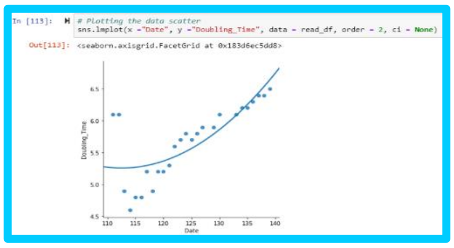
```
In [89]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Daily_Recovered_Cases", data = read_df, order = 2, ci = None)
Out[89]: <seaborn.axisgrid.FacetGrid at 0x183d5230668>
```



```
In [97]: # Plotting the data scatter
         sns.lmplot(x ="Date", y ="Daily_Deceased_Cases", data = read_df, order = 2, ci = None)
Out[97]: <seaborn.axisgrid.FacetGrid at 0x183d53cc2b0>
```

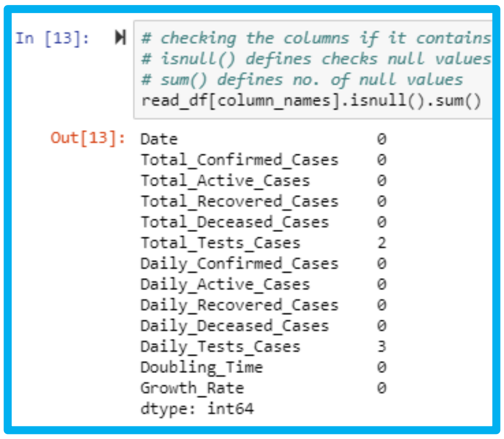


```
In [105]: # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Daily_Tests_Cases", data = read_df, order = 2, ci = None)
Out[105]: <seaborn.axisgrid.FacetGrid at 0x183d6bb6d68>
```



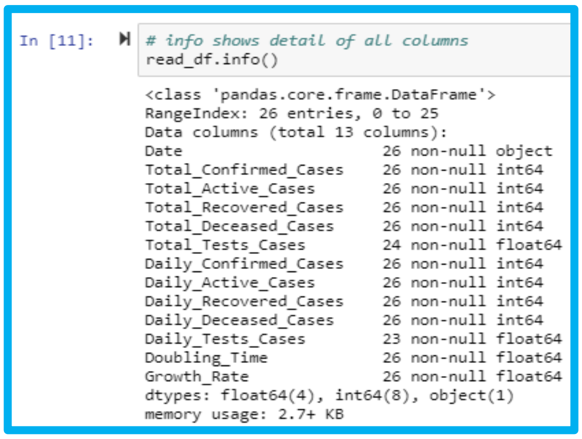
```
In [113]: # Plotting the data scatter
          sns.lmplot(x ="Date", y ="Doubling_Time", data = read_df, order = 2, ci = None)
Out[113]: <seaborn.axisgrid.FacetGrid at 0x183d6ec5dd8>
```

6. Check if any of the column having “na” values or not. We can see, Total_Tests_Cases and Daily_Tests_Cases columns having na values.



```
In [13]:  # checking the columns if it contains
          # isnull() defines checks null values
          # sum() defines no. of null values
          read_df[column_names].isnull().sum()

Out[13]:  Date                    0
          Total_Confirmed_Cases   0
          Total_Active_Cases      0
          Total_Recovered_Cases   0
          Total_Deceased_Cases    0
          Total_Tests_Cases       2
          Daily_Confirmed_Cases   0
          Daily_Active_Cases      0
          Daily_Recovered_Cases   0
          Daily_Deceased_Cases    0
          Daily_Tests_Cases       3
          Doubling_Time           0
          Growth_Rate             0
          dtype: int64
```

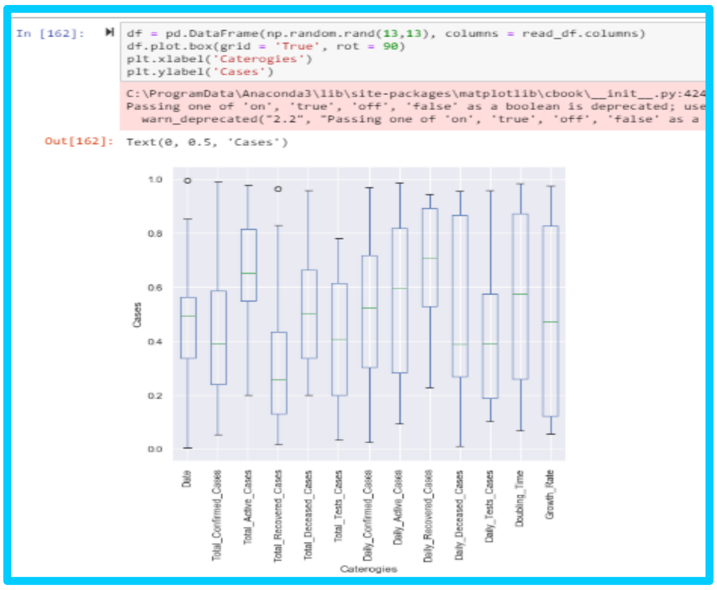
7. Showing details of all the columns from the dataset i.e. no. of values available, types of the column.



```
In [11]:  # info shows detail of all columns
          read_df.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 26 entries, 0 to 25
          Data columns (total 13 columns):
          Date                     26 non-null object
          Total_Confirmed_Cases    26 non-null int64
          Total_Active_Cases       26 non-null int64
          Total_Recovered_Cases    26 non-null int64
          Total_Deceased_Cases     26 non-null int64
          Total_Tests_Cases        24 non-null float64
          Daily_Confirmed_Cases    26 non-null int64
          Daily_Active_Cases       26 non-null int64
          Daily_Recovered_Cases    26 non-null int64
          Daily_Deceased_Cases     26 non-null int64
          Daily_Tests_Cases        23 non-null float64
          Doubling_Time            26 non-null float64
          Growth_Rate              26 non-null float64
          dtypes: float64(4), int64(8), object(1)
          memory usage: 2.7+ KB
```

8. Plotting Box Plot of all columns

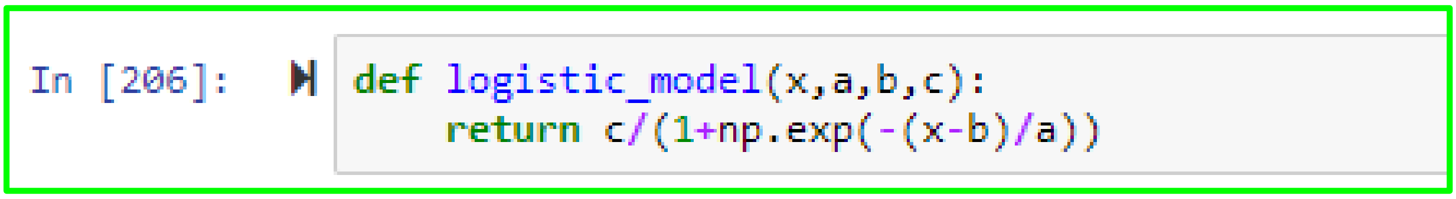
Observation: It is observed that on X- axis, there are Catagories and on Y- axis, there are Cases. We can see that there are outliers in Date and Total_Active_Cases columns but not affecting the rest of the data hence we can keep these outliers.



```
In [162]:  df = pd.DataFrame(np.random.rand(13,13), columns = read_df.columns)
           df.plot.box(grid = 'True', rot = 90)
           plt.xlabel('Caterogies')
           plt.ylabel('Cases')

           C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:424
           Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use
             warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a

Out[162]:  Text(0, 0.5, 'Cases')
```

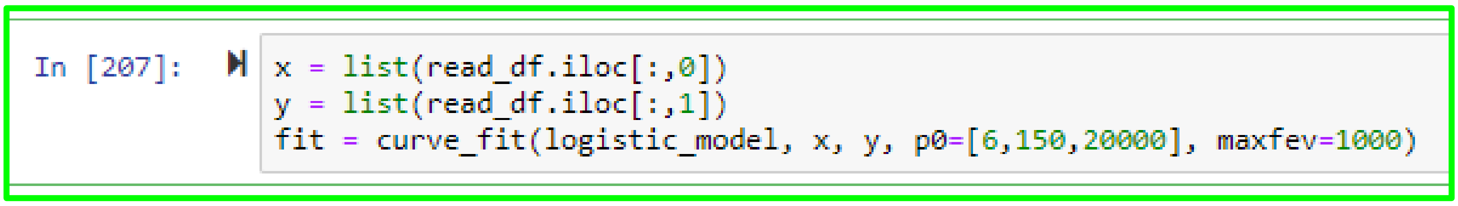
The Logistic Model

1. Defining function



```
In [206]:  def logistic_model(x,a,b,c):
               return c/(1+np.exp(-(x-b)/a))
```

2. *curve_fit* function

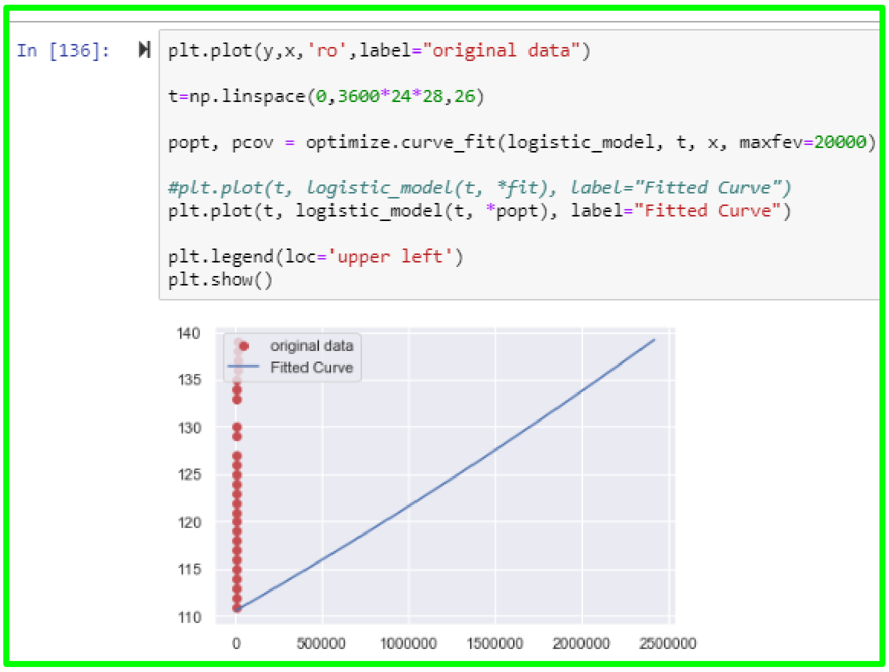


```
In [207]:  x = list(read_df.iloc[:,0])
           y = list(read_df.iloc[:,1])
           fit = curve_fit(logistic_model, x, y, p0=[6,150,20000], maxfev=1000)
```

Here are the values:

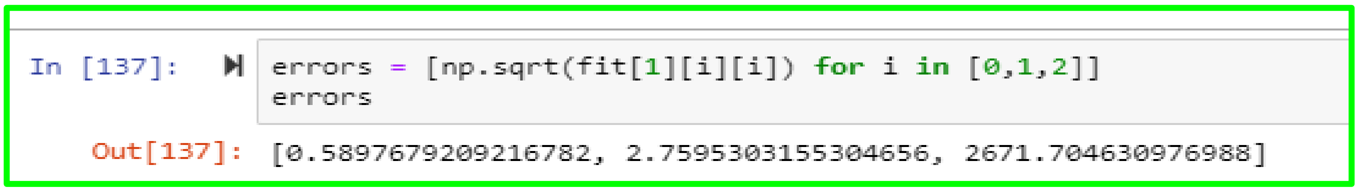
- a: 6.5
- b: 135
- c: 10054

3. Plotting Graph of Original data vs Fitted Curve



```
In [136]:  plt.plot(y,x,'ro',label="original data")

           t=np.linspace(0,3600*24*28,26)

           popt, pcov = optimize.curve_fit(logistic_model, t, x, maxfev=20000)

           #plt.plot(t, logistic_model(t, *fit), label="Fitted Curve")
           plt.plot(t, logistic_model(t, *popt), label="Fitted Curve")

           plt.legend(loc='upper left')
           plt.show()
```

The function returns the **covariance matrix** too, whose diagonal values are the variances of the parameters. Taking their square root we can calculate the standard errors.



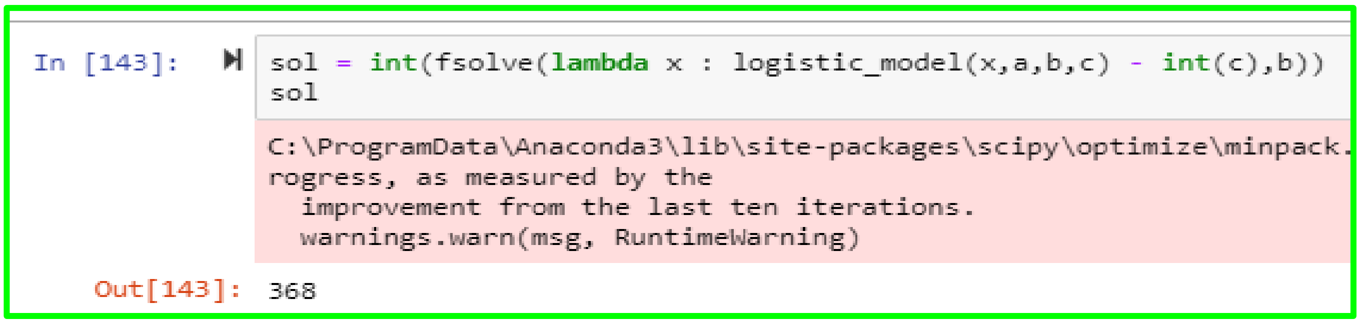
```
In [137]:  errors = [np.sqrt(fit[1][i][i]) for i in [0,1,2]]
           errors

Out[137]: [0.5897679209216782, 2.7595303155304656, 2671.704630976988]
```

- Standard error of *a:* 0.59
- Standard error of *b*: 2.76
- Standard error of *c*: 2671.70

These numbers give us many useful insights i.e.,

The **expected number of infected people** at infection end is **10054+/- 2671**.

4. fsolve function



```
In [143]:  sol = int(fsolve(lambda x : logistic_model(x,a,b,c) - int(c),b))
           sol

           C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\minpack.
           rogress, as measured by the
             improvement from the last ten iterations.
             warnings.warn(msg, RuntimeWarning)

Out[143]: 368
```

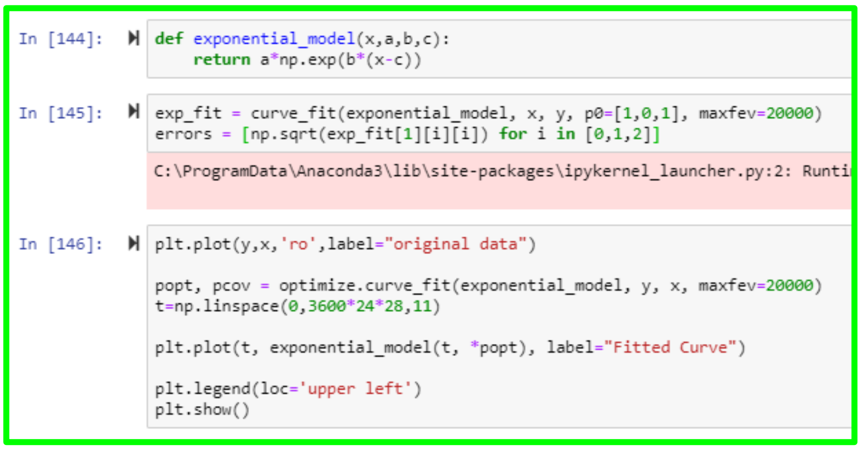
The **expected infection end is 368 i.e. on 2nd January 2021.**

The Exponential Model

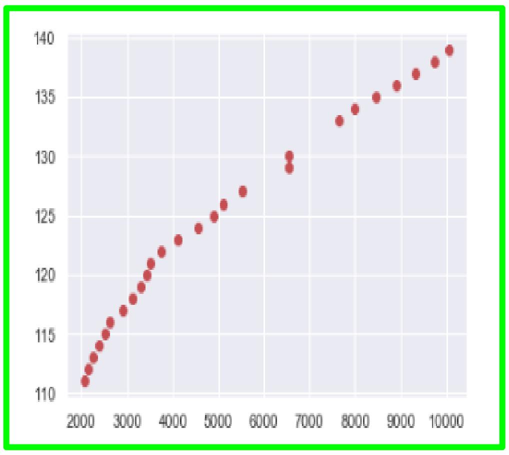
1. Defining function

2. curve_fit function

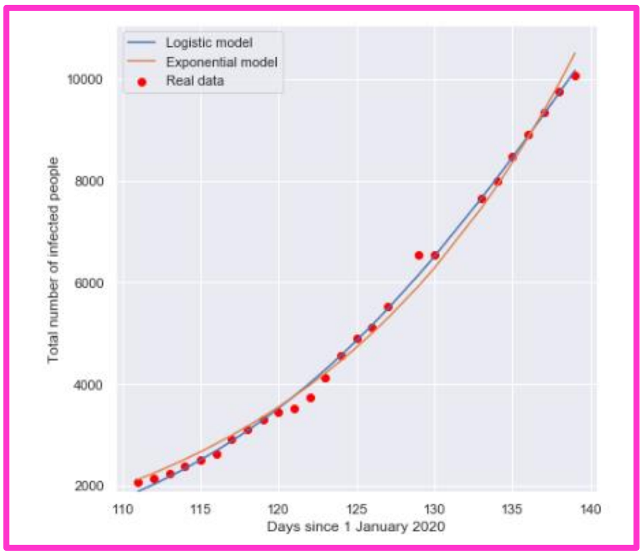
3. Plotting Graph of Original data vs Fitted Curve



```
In [144]:  def exponential_model(x,a,b,c):
               return a*np.exp(b*(x-c))

In [145]:  exp_fit = curve_fit(exponential_model, x, y, p0=[1,0,1], maxfev=20000)
           errors = [np.sqrt(exp_fit[1][i][i]) for i in [0,1,2]]

           C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Runti

In [146]:  plt.plot(y,x,'ro',label="original data")

           popt, pcov = optimize.curve_fit(exponential_model, y, x, maxfev=20000)
           t=np.linspace(0,3600*24*28,11)

           plt.plot(t, exponential_model(t, *popt), label="Fitted Curve")

           plt.legend(loc='upper left')
           plt.show()
```



<u>Plotting Final Model:</u> Now, we have now all the necessary data to visualize our results.



Conclusion

ML is being applied and delivering results in three fields: in virus research and the development of drugs and vaccines; in the management of services and resources at healthcare centers; and in the analysis of data to support public policy decisions aimed at managing the crisis, such as the confinement measures.

While analyzing the two models, the logistic function and the exponential function, each model has **three parameters**, whose values are:

- a: 6.5
- b: 135
- c: 10054

The function returns the **covariance matrix** too, whose diagonal values are the variances of the parameters. Taking their square root we can calculate the standard errors.

- Standard error of *a:* 0.59
- Standard error of *b*: 2.76
- Standard error of *c*: 2671.70

These numbers give us many useful insights i.e.,

The **expected number of infected people** at infection end is **10054+/- 2671**.

The **expected infection end is on 2nd January 2021.**

Residual Analysis:

Logistic model MSE: 16020.34
Logistic model MSE: 41999.53

Which is the right model?

Residuals analysis seems to point toward the **logistic model**. It's very likely because the **infection should end** someday in the future; even if everybody will be infected, they'll develop the proper **immunity defense** to avoid a second infection. That's right as long as the virus **doesn't mutate** too much (as, for example, influenza virus).

But there's something that **still worries me**. I've been fitting the logistic curve every day since the beginning of the infection and every day **I got different parameter values**. The number of infected people at the end **increases**, the maximum infection day is often the current day or the next day (which is compatible with the standard error of 1 day on this parameter). That's why I think that, although the logistic model seems to be the most reasonable one, the shape of the curve **will probably change** due to exogenous effects like new infection **hotspots**, government **actions to bind** the infection and so on.

That’s why I think that the predictions of this model will start to become useful only within a few weeks, reasonably after the infection peak.
I do hope we are building the infrastructures and processes to ensure that things flow more quickly and efficiently when the next pandemic occurs.

Future Enhancement

“The past cannot be changed.
The future is yet in your power.”

Machine learning is an important tool in fighting the current pandemic. If we take this opportunity to collect data, pool our knowledge, and combine our skills, we can save many lives – both now and in the future.

As regards application of AI to research, work seems to be progressing at a modest pace. There are still many unanswered questions surrounding the virus. But scientific teams are not giving up and researchers break new ground every day. Scientists are working to predict the protein structures of the novel coronavirus, which is essential to better understanding how it evolves and how to control it. Even to develop a vaccine in combination with the findings of other research projects.

Data science and big data are proving highly valuable for improving hospital management. The company is focusing on understanding better how hospitals work and helping them deliver their services better. They have achieved this by adapting their software, data questionnaires, diagnoses and different algorithms.
The project has a very vast scope in future. The project can be implemented on intranet in future. Project can be updated in near future as and when requirement for the same arises, as it is very flexible in terms of expansion. Although our system had been completed but it is not perfect, we had planned to make some enhancement in the future. We think that our system still has potential to grow.

Reference

- https://towardsdatascience.com/covid-19-infection-in-italy-mathematical-models-and-predictions-7784b4d7dd8d
- https://www.mygov.in/covid-19
- https://www.covid19india.org/
- https://en.wikipedia.org/wiki/COVID-19_pandemic_in_India
- https://en.wikipedia.org/wiki/COVID-19_pandemic_in_Delhi
- https://healthitanalytics.com/features/could-covid-19-help-refine-ai-data-analytics-in-healthcare
- https://www.bbva.com/en/how-artificial-intelligence-can-help-fight-covid-19/
- https://www.datarevenue.com/en-blog/machine-learning-covid-19
- https://economictimes.indiatimes.com/news/economy/policy/mathematical-models-are-used-to-predict-covid-19s-path-but-is-the-incoming-data-sound/articleshow/75101240.cms?utm_source=contentofinterest&utm_medium=text&utm_campaign=cppst
- https://www.researchgate.net/publication/340078234_CORONAVIRUS_COVID-19_AVAILABLE_FREE_LITERATURE_PROVIDED_BY_VARIOUS_COMPANIES_JOURNALS_AND_ORGANIZATIONS_AROUND_THE_WORLD_JOURNAL_OF_ONGOING_CHEMICAL_RESEARCH
- Aarogya Setu App

Thank

You!