# CSE303: Statistics for Data Science
# [Fall 2021]

# Project Report
# Group No. – 8 (Section 1)

**Submitted by:**

| Student ID | Student Name | Contribution Percentage | Signature |
|---|---|---|---|
| 2019-1-60-068 | Md. Shahadat Anik Sheikh | 33.3 | Anik |
| 2019-1-60-108 | A. M. Feroz Ehses Shishir | 33.3 | Shishir |
| 2019-3-60-052 | Abdullah Al Munem | 33.3 | Munem |

## 1. Introduction

This project focuses on developing supervised machine learning model for both regression and classification with the given three datasets. The datasets are "covid_dataset", "covid_first_dose" and "covid_second_dose". These datasets are based on during Bangladesh's Covid-19 situation of last 21 months. Here data is distributed from many aspects like lab test, confirmed case, death case, number of vaccinations etc. We have tried to implement some regression and classification models over these datasets.

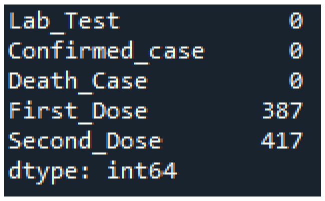
For implementing regression and classification models, first we have merged the datasets. After merging these datasets, it was needed to perform some pre-processing. In order to implement the algorithm and output the best result this data has been pre-processed by some criteria. We have used Label Encoder for the transformation and Moving Average for smoothing the data. This was very necessary steps.

After doing all of this, we have used some significant models of both regression and classification model using the Scikit-learn(sklearn) library from Python, in order to find the best fit model and examine the accuracy rate. For regression, we have used Linear regression, Polynomial regression, Lasso Regression, Ridge Regression, Elastic Net Regression etc. and for classification, we have used Logistic Regression and Linear Support Vector Classifier.

## 2. Data Pre-processing

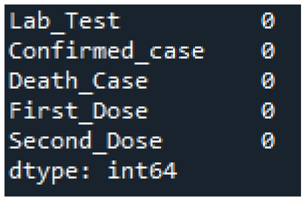
Prepossessing is an important part of the data analysis and researching as it is the key to have the most correct accuracy. Therefore, implementing the machine learning algorithms and output the best results, we have pre-processed the dataset. We are given three datasets those are "covid_dataset", "covid_first_dose" and "covid_second_dose".

1. **Filling Null Values**:
   At first, we have merged the given three datasets where "Day" is used as indexing. After merging, we have checked whether any column has missing values or not because, missing or null values would not be readable by our machine learning models. We have used 'isnull()' function for checking the missing values of columns and 'print(df.isnull().sum())' this code gives us a list where we can see number of missing values have in columns. If any column has missing values, then we have replaced them by 0 with using 'fillna()' function.



```
Lab_Test           0
Confirmed_case     0
Death_Case         0
First_Dose       387
Second_Dose      417
dtype: int64
```

Fig: Before using ‘Fillna’

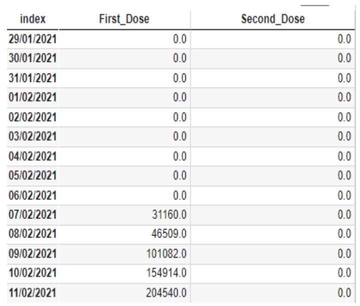


```
Lab_Test           0
Confirmed_case     0
Death_Case         0
First_Dose         0
Second_Dose        0
dtype: int64
```

Fig: After using ‘Fillna’

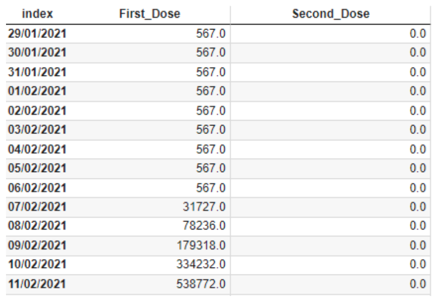
2. **Duplicate Checking**:
   After that we have checked whether there is any duplicate values or not because, in data analysis working with duplicate values will hamper our training and testing results which is very much inefficient. If there is any duplicate row then the row will be dropped.

3. **Cumulative Frequency Distribution (CFD)**:
   We have used cumulative frequency distribution for our two of the columns ‘First_Dose’ and ‘Second_Dose’. The reason behind this is when we are targeting the ‘Confirmed_case’ column, these independent columns ‘First_Dose’ and ‘Second_Dose’ should indicate that how many people have already taken the first dose and second dose because, the confirmed case won’t be recognized how many people have taken the first dose or second dose in that particular day.



| index | First_Dose | Second_Dose |
|---|---|---|
| 29/01/2021 | 0.0 | 0.0 |
| 30/01/2021 | 0.0 | 0.0 |
| 31/01/2021 | 0.0 | 0.0 |
| 01/02/2021 | 0.0 | 0.0 |
| 02/02/2021 | 0.0 | 0.0 |
| 03/02/2021 | 0.0 | 0.0 |
| 04/02/2021 | 0.0 | 0.0 |
| 05/02/2021 | 0.0 | 0.0 |
| 06/02/2021 | 0.0 | 0.0 |
| 07/02/2021 | 31160.0 | 0.0 |
| 08/02/2021 | 46509.0 | 0.0 |
| 09/02/2021 | 101082.0 | 0.0 |
| 10/02/2021 | 154914.0 | 0.0 |
| 11/02/2021 | 204540.0 | 0.0 |

Fig: Before using ‘CFD’



| index | First_Dose | Second_Dose |
|---|---|---|
| 29/01/2021 | 567.0 | 0.0 |
| 30/01/2021 | 567.0 | 0.0 |
| 31/01/2021 | 567.0 | 0.0 |
| 01/02/2021 | 567.0 | 0.0 |
| 02/02/2021 | 567.0 | 0.0 |
| 03/02/2021 | 567.0 | 0.0 |
| 04/02/2021 | 567.0 | 0.0 |
| 05/02/2021 | 567.0 | 0.0 |
| 06/02/2021 | 567.0 | 0.0 |
| 07/02/2021 | 31727.0 | 0.0 |
| 08/02/2021 | 78236.0 | 0.0 |
| 09/02/2021 | 179318.0 | 0.0 |
| 10/02/2021 | 334232.0 | 0.0 |
| 11/02/2021 | 538772.0 | 0.0 |

Fig: After using ‘CFD’

4. **Moving Average**:
   Then we have used moving average to our dataset. Moving averages are measures of momentum over a series of observed values. These measures are commonly made across a subset of values within a larger set. In the given dataset there were very much fluctuating data. So, smooth out the dataset, here we have used 14 days or two weeks’ time period for optimal training and testing.

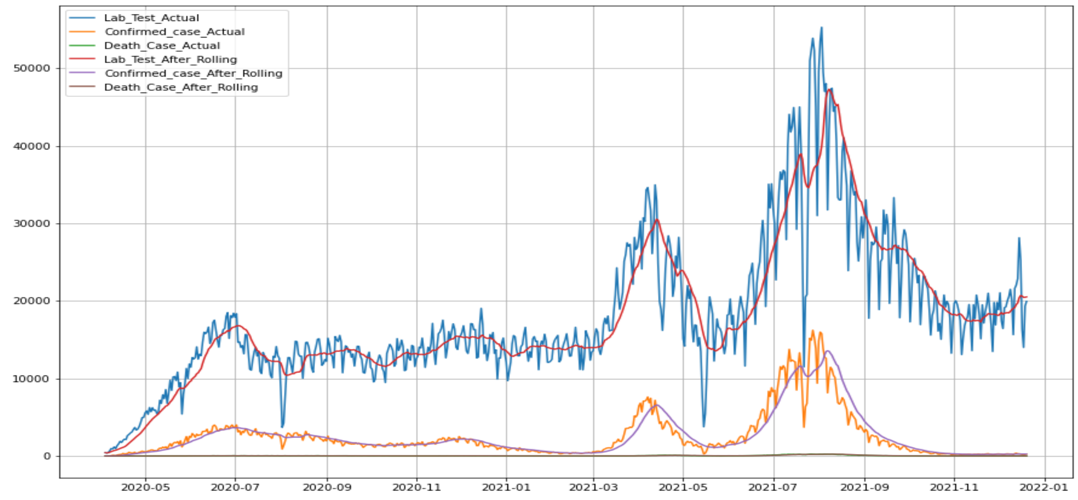
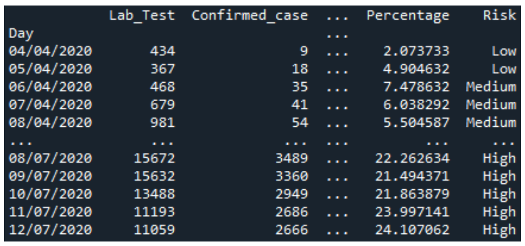


Fig: Raw data VS Moving Average of two weeks

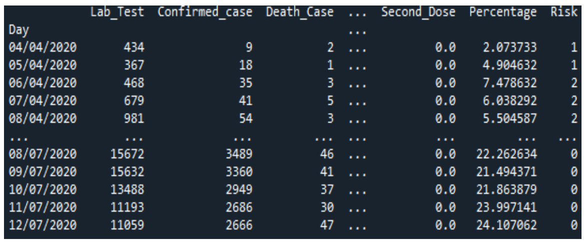
5. **Label Encoding**:

   For encoding out dataset, we have used 'LabelEncoder()'. We used it, when we were classifying the 'Risk'. We have used low, medium and high three risky time which are string and are not compatible for our machine learning models. Thus we needed to perform Label Encoding to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. [1] In 'LabelEncoder()' technique, every class converted to serial wise numbers and which is started from 0.



```
            Lab_Test  Confirmed_case  ...  Percentage    Risk
Day                                   ...
04/04/2020       434               9  ...    2.073733     Low
05/04/2020       367              18  ...    4.904632     Low
06/04/2020       468              35  ...    7.478632  Medium
07/04/2020       679              41  ...    6.038292  Medium
08/04/2020       981              54  ...    5.504587  Medium
...              ...             ...  ...         ...     ...
08/07/2020     15672            3489  ...   22.262634    High
09/07/2020     15632            3360  ...   21.494371    High
10/07/2020     13488            2949  ...   21.863879    High
11/07/2020     11193            2686  ...   23.997141    High
12/07/2020     11059            2666  ...   24.107062    High
```

Fig: Before Label Encoding



```
            Lab_Test  Confirmed_case  Death_Case  ...  Second_Dose  Percentage  Risk
Day                                               ...
04/04/2020       434               9           2  ...          0.0    2.073733     1
05/04/2020       367              18           1  ...          0.0    4.904632     1
06/04/2020       468              35           3  ...          0.0    7.478632     2
07/04/2020       679              41           5  ...          0.0    6.038292     2
08/04/2020       981              54           3  ...          0.0    5.504587     2
...              ...             ...         ...  ...          ...         ...   ...
08/07/2020     15672            3489          46  ...          0.0   22.262634     0
09/07/2020     15632            3360          41  ...          0.0   21.494371     0
10/07/2020     13488            2949          37  ...          0.0   21.863879     0
11/07/2020     11193            2686          30  ...          0.0   23.997141     0
12/07/2020     11059            2666          47  ...          0.0   24.107062     0
```

Fig: After Label Encoding

## 3. Dataset Characteristics and Exploratory Data Analysis

Our merged dataset has 626 rows and 5 columns and where "Day" is used as indexing. In these 5 columns we have "Lab_Test", "Confirmed_case", "Death_Case", "First_Dose", "Second_Dose" where, each row indicates on a particular day how many labs have been tested for Covid-19, how many have been confirmed positive for Covid-19, how many have died with Covid-19 symptoms and (including that day) total how many have been given the first dose of the vaccine and total how many have been given the second dose of the vaccine.

We have copped our main dataset and created another two new datasets because, we wanted to show that how our pre-processed data gives us better results alongside with raw data. The correlation of our dataset is given below.

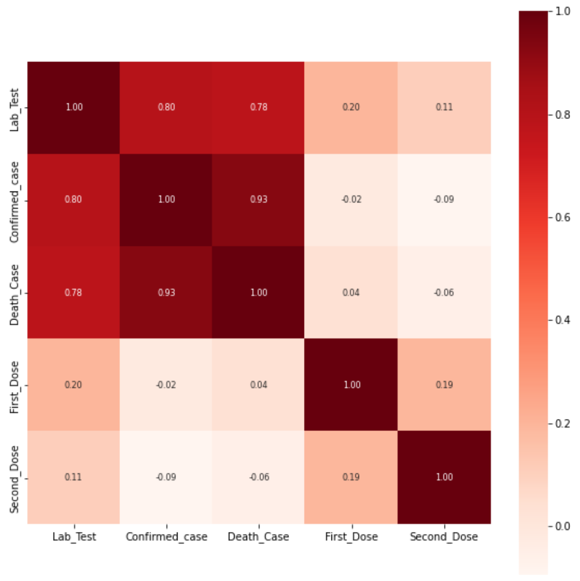


Fig: Correlation of raw dataset

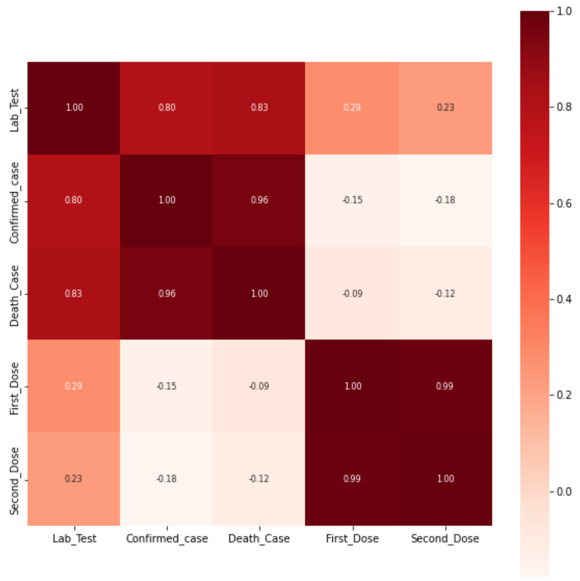


Fig: Correlation of pre-processed dataset

From the heatmap of raw and pre-processed datasets we can see that, the correlation haven't changed that much except the correlation in between "First_Dose" and "Second_Dose". The reason behind this is, in our pre-processed dataset we have cumulatively sum up the "First_Dose" and "Second_Dose" because, the "Death_Case" and "Confirmed_case" will be depend on total how many have got this vaccine. Therefore, we can work on with our pre-processed dataset.

From the correlation of pre-processed data we can see that, the "First_Dose" and "Second_Dose" have weak negative correlation with "Confirmed_case" and "Death_Case". Which means that when "First_Dose" and "Second_Dose" was low the "Confirmed_case" and "Death_Case" was high and increasing the number of "First_Dose" and "Second_Dose" makes the number of "Confirmed_case" and "Death_Case" low.

## 4. Machine Learning Models

Among the machine learning models we have used Linear regression, Polynomial regression, Lasso Regression, Ridge Regression, Elastic Net Regression etc. as regression model and Logistic Regression and Linear Support Vector Classifier as classification model.

**Regression Models:**

1. **Linear Regression**:

   Linear regression is a statistical model used to model the relationship between two variables. It is the simplest linear statistical model used for regression problems. It is a linear model where the dependent variable is calculated using the values of the independent variables and a linear equation. The results of the linear equation make predictions about the data.

1.1. **Scikit-learn**:

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. In Scikit-learn library Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.[2]

1.2. **Ordinary Least Squares**: Ordinary Least Squares (OLS) is a useful method for evaluating a linear regression model. It does this by using specific statistical performance metrics about the model as a whole and each specific parameter of the model. The OLS method comes from the StatsModel python package. This module is well known for offering multiple classes and functions that both estimate different types of statistical models and conducting multiple statistical tests. This model is created by fitting a linear equation to the observed data. [3]

2. **Polynomial Regression**:

The link among the independent x variables and the dependent y variable is treated as an nth degree polynomial in x in polynomial regression. Polynomial regression is a widely used method. Because we build certain polynomial characteristics before creating a linear regression, it is a particular instance of linear regression. It is applied to nonlinear phenomena. Polynomial L1 and Polynomial L2 are the regularization of polynomial model, where in L1 uses Lasso and L2 uses Ridge.

3. **Lasso Regression**:

The term "LASSO" also known for "Least Absolute Shrinkage" is a statistical technique for regularizing data models and selecting features. For a more precise prediction, it is preferred over regression approaches. Shrinkage is used in this model i.e. data values are compressed towards a central point known as the average in shrinkage. This type of regression is ideal for models with a lot of multicollinearities or when we wish to automate elements of the model selection process, such as variable selection and parameter removal.

4. **Ridge Regression**:

In Ridge regression we actually give penalty or model with L2 regularization. With our actual equation the square value should be added and it gives high penalty to our model. And this is how our model try to minimize the penalty and it gives us the most optimal R-squared value so far. Ridge is works better than the lasso and normal linear regression only because on the L2 norm penalty. So with ridge we can get the most optimal R squared value for the regression model.

5. **Elastic Net Regression**:
   ElasticNet regression is a combination of ridge and LASSO methods: add both penalty terms to the usual least squares loss function and you will get the ElasticNet regression. It also has two shrinkage parameters. It is especially helpful when you have multiple correlated features. When two features are correlated, LASSO tends to choose one of them randomly, while ElasticNet keeps both. Similar to the ridge regression, ElasticNet is also more stable in many cases.

**Classification Models:**

1. **Logistic regression:**

When the variable is categorical, logistic regression is the best regression strategy to use . The logistic regression, like all regression studies, is a trend analysis. To define data and understand the relationship among one dependent binary variable and one or more ratio-level independent variables, logistic regression is utilized. This statistical model that make predictions for binary classification problems. It uses a linear method used in various fields of machine learning by determining a relationship between features and the probabilities of particular outcomes.

2. **Linear Support Vector Classifier:**

The main goal of the Linear Support Vector Classifier is when we provide it a dataset it fits the provided dataset and return a best fit hyperplane that divides, or categorizes, our dataset. From there, after getting the hyperplane, we can then feed some features to our classifier to see what the predicted class is.

**Time Series Analysis:**

**Autoregressive Integrated Moving Average (ARIMA):**

The ARIMA model is used for forecasting something based on previous data. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration. This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

1) Autoregression (AR): A model that uses the dependent relationship between an observation and some number of lagged observations.
2) Integrated (I): The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.[5]
3) Moving Average (MA): A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

## 5. Description of Models and Associated Parameters

For this project we have used many machine learning models where in those models have many types of parameters. In below, we have given the detailed description of the parameters and where we have used it.

| Parameters | Functions | Used In Models | Description |
|---|---|---|---|
| test_size | train_test_split(X, Y, test_size = ratio, random_state = 0) | Used In All Models | If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split.<br><br>If int, represents the absolute number of test samples.<br><br>If None, the value is set to the complement of the train size. If train_size is also None, it will be set to 0.25. |
| random_state | train_test_split(X, Y, test_size = ratio, random_state = 0) | Used In All Models | Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. |
| degree | PolynomialFeatures (degree) | Polynomial_Regression(df, X, Y, degree=2, ratio = 0.2, flg=0)<br><br>Polynomial_Regression_L1(df, X, Y, degree=2, alpha = 0.5, ratio = 0.2, flg=0)<br><br>Polynomial_Regression_L2(df, X, Y, degree=2, alpha = 0.5, ratio = 0.2, flg=0) | Generate a new feature matrix consisting of all polynomial combinations of the features. If you have features [a, b, c] the default polynomialFeatures (in sk-learn the default degree is 2) should be [1, a, b, c, $a^2$, $b^2$, $c^2$, ab, bc, ca].<br>We can choose the degree of polynomial based on the relationship between target and predictor. The 1-degree polynomial is a simple linear regression; therefore, the value of degree must be greater than 1. With the increasing degree of the polynomial, the complexity of the model also increases. |
|  |  | Lasso_Regression(df, X, Y, alpha = |  |

| | | | |
|---|---|---|---|
| alpha | Lasso(alpha = alpha)<br><br>Ridge(alpha = alpha) | 0.5, ratio = 0.2, flg=0)<br><br>Ridge_Regression( df, X, Y, alpha = 0.5, ratio = 0.2, flg=0)<br><br>ElasticNet_Regression(df, X, Y, alpha = 0.5, l1_ratio = 0.5, ratio = 0.2, flg=0) | Lasso and Ridge regression comes with a parameter alpha, and the higher the alpha, the most feature coefficients are zero. When alpha is 0, regression produces the same coefficients as a linear regression. |
| C | LogisticRegression( C = c)<br><br>SVC(kernel='linear' ,C = c, gamma=gamma) | Logistic_Regression(X, Y, c=10, ratio = 0.2)<br><br>SVM(X, Y, gamma='auto', ratio = 0.2, c=10) | Parameter C will work the other way around. For small values of C, we increase the regularization strength which will create simple models, which underfit the data. For big values of C, we low the power of regularization which impels the model is allowed to increase its complexity, and therefore, overfit the data. |
| kernel | SVC(kernel='linear' ,C=c, gamma=gamma) | SVM(X, Y, gamma='auto', ratio = 0.2, c=10) | Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a Large number of Features in a particular Data Set.<br>Training a SVM with a Linear Kernel is Faster than with any other Kernel.<br>When training a SVM with a Linear Kernel, only the optimisation of the C Regularisation parameter is required. On the other hand, when training with other kernels, there is a need to optimise the γ parameter which means that performing a grid search will usually take more time. |
| gamma | SVC(kernel='linear' ,C=c, gamma=gamma) | SVM(X, Y, gamma='auto', ratio = 0.2, c=10) | It is the kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’. If gamma is |

| | | | |
|---|---|---|---|
| | | | ‘auto’, then 1/n_features will be used instead. |
| window | rolling(window=14, min_periods=1).mean() | | The method or model using repeatedly to the sub-data sets or sub-series in your full data set or series. |
| x | adfuller(timeseries, autolag='AIC') | test_stationarity(timeseries) | It’s like a 1d array to test the data series. |
| autolag | adfuller(timeseries, autolag='AIC') | test_stationarity(timeseries) | If “AIC” (default), then the number of lags is chosen to minimize the corresponding information criterion. |
| x | seasonal_decompose(df_freq_2, model = 'additive') | | It is refer to the dataset. |
| model | seasonal_decompose(df_freq_2, model = 'additive') | | An additive model is linear where changes over time are consistently made by the same amount. |
| lags | sgt.plot_acf(df_freq_2, lags=40, zero = False) | | Lags 40 means that we are calculating correlation between a present series and series 40 time periods before. |
| zero | sgt.plot_acf(df_freq_2, lags=40, zero = False) | | zero = false means that, we don’t calculate correlation between series now and now, because that will always be one. |
| method | sgt.plot_pacf(df_freq_2, lags=40, zero = False, method=('ols')) | | “ols” : regression of time series on lags of it and on constant. |
| order | ARIMA(df_freq_2, order=(5,0,5)) | | The (p,d,q) order of the model for the autoregressive, differences, and moving average components. d is always an integer, while p and q may either be integers or lists of integers. |

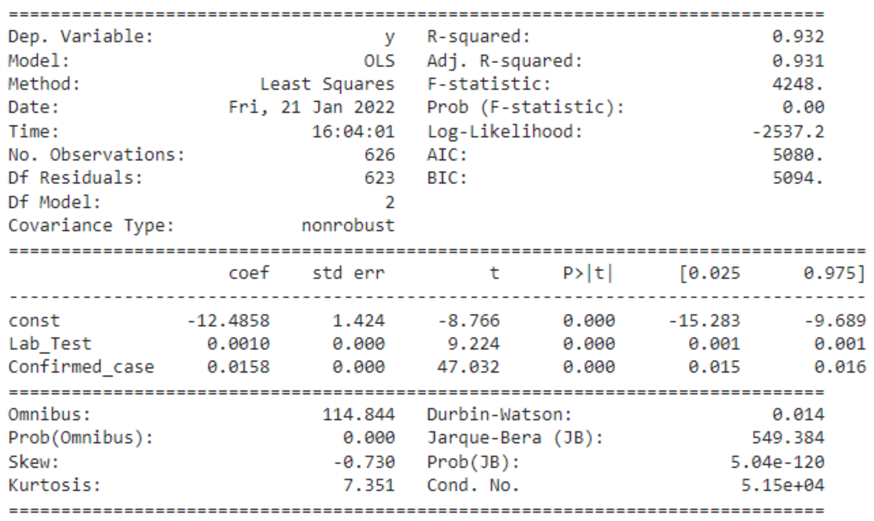
## 6. Performance Evaluation

In this section, we are comparing our models when it works better and getting that better performed model for the next section.

1. **Predict the Death Case**:

1) **Ordinary Least Squares (OLS)**:

We have used OLS method to find out the equation of the Linear Regression to predict the death case.



```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.932
Model:                            OLS   Adj. R-squared:                  0.931
Method:                 Least Squares   F-statistic:                     4248.
Date:                Fri, 21 Jan 2022   Prob (F-statistic):               0.00
Time:                        16:04:01   Log-Likelihood:                -2537.2
No. Observations:                 626   AIC:                             5080.
Df Residuals:                     623   BIC:                             5094.
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const            -12.4858      1.424     -8.766      0.000     -15.283      -9.689
Lab_Test           0.0010      0.000      9.224      0.000       0.001       0.001
Confirmed_case     0.0158      0.000     47.032      0.000       0.015       0.016
==============================================================================
Omnibus:                      114.844   Durbin-Watson:                   0.014
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              549.384
Skew:                          -0.730   Prob(JB):                    5.04e-120
Kurtosis:                       7.351   Cond. No.                     5.15e+04
==============================================================================
```

Fig: OLS Regression Results

From the summary we have found the equation which is,

$$\text{Death Case} = -12.4858 + 0.0010 \times \text{Lab Test} + 0.0158 \times \text{Confirmed Case}$$

2) **Linear Regression (LR):**

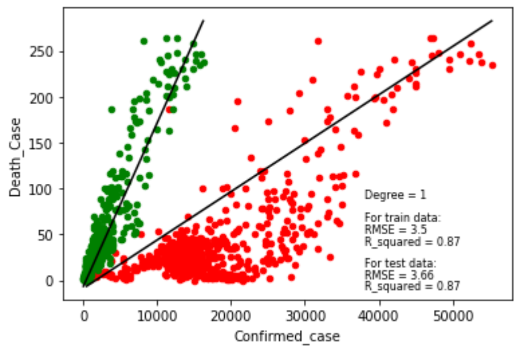


Fig: LR with raw dataset

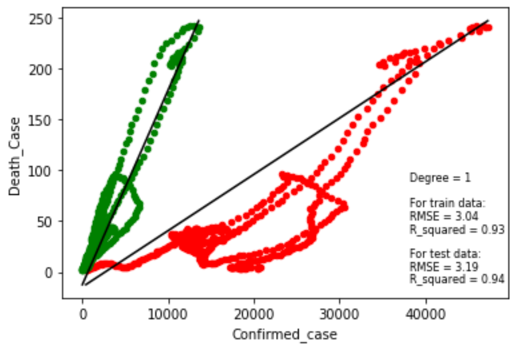


Fig: LR with pre-processed dataset

3) **Polynomial Regression (PR)**:

The most important hyperparameter in the PolynomialFeatures() class is degree. We have used various degree so that we can find out which degree performs better.

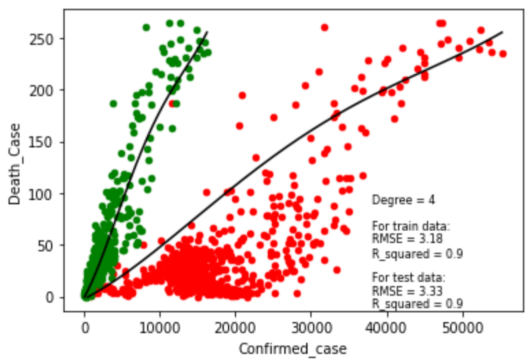


Fig: PR with raw data with degree 4

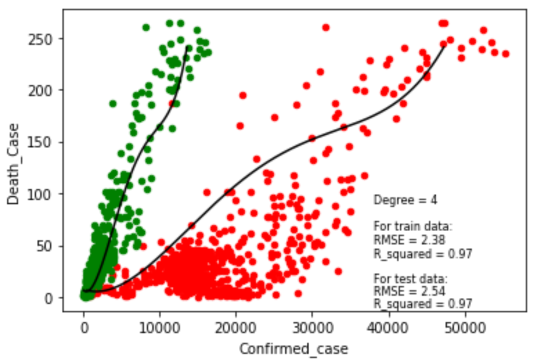


Fig: PR with clean data with degree 4

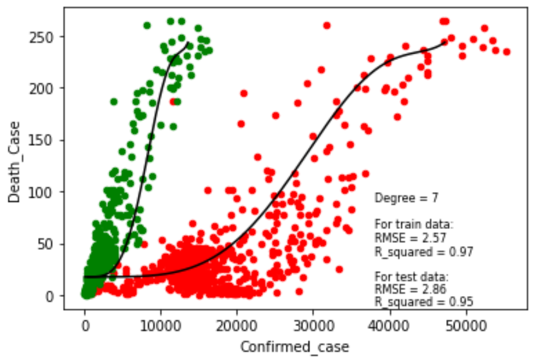


Fig: PR with clean data with degree 7

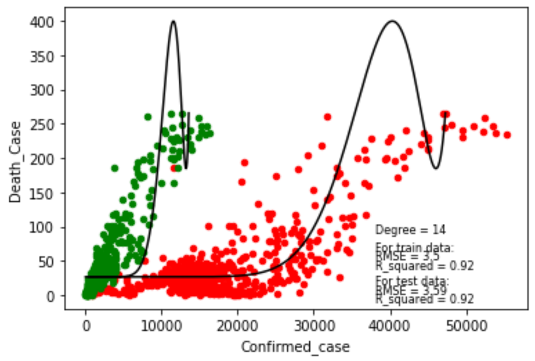


Fig: PR with clean data with degree 14

Here we have took the degree of 4 in our analysis because, increasing the degree farther more overfitting our data.

4) **Polynomial Regression L1 (PR_L1)**:
Here we have also used various degree to find out which degree performs better. But in here we have used PolynomialFeatures() over Lasso as regularization parameter.

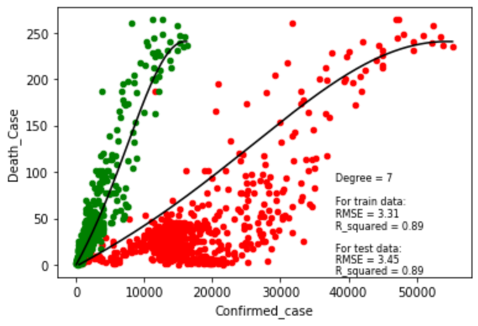


Fig: PR_L1 with raw data with degree 7

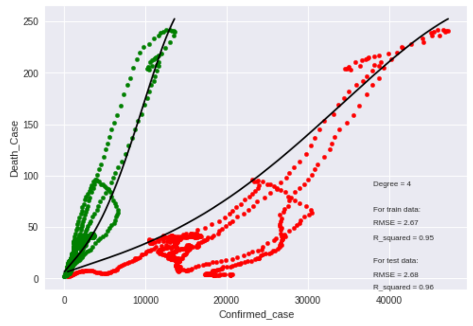


Fig: PR_L1 with clean data with degree 4

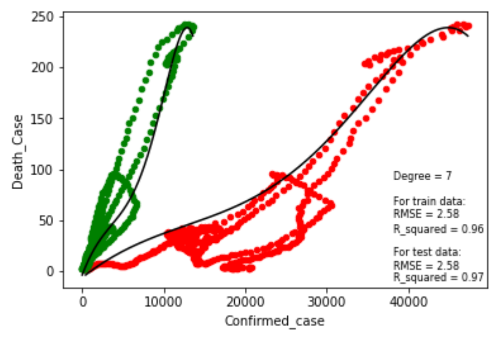


Fig: PR_L1 with clean data with degree 7

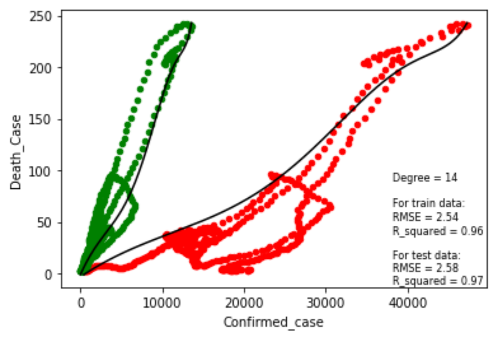


Fig: PR_L1 with clean data with degree 14

Here we have took the degree of 7 as our analysis because here the difference of train and testing RMSE is minimum also degree 7 has higher R-squired value and increasing the degree farther more overfitting our data.

5) **Polynomial Regression L2 (PR_L2)**:

Here we have also used various degree to find out which degree performs better. But in here we have used PolynomialFeatures() over Ridge as regularization parameter.

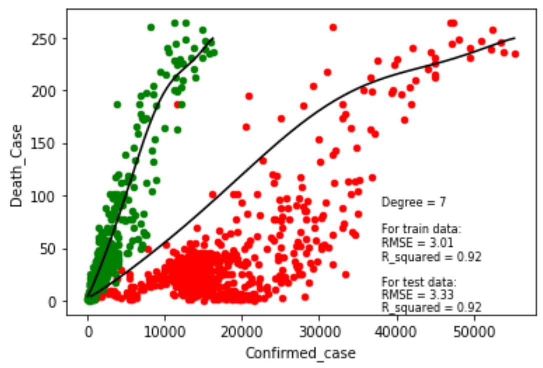


Fig: PR_L2 with raw data with degree 7

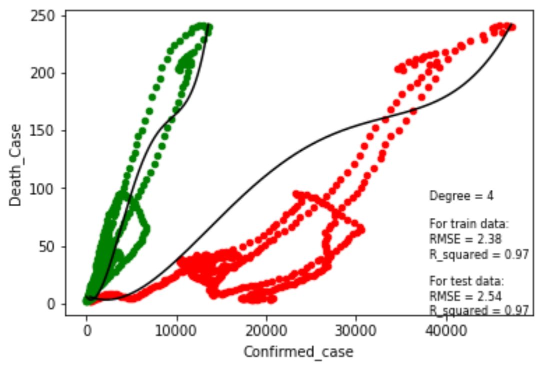


Fig: PR_L2 with clean data with degree 4

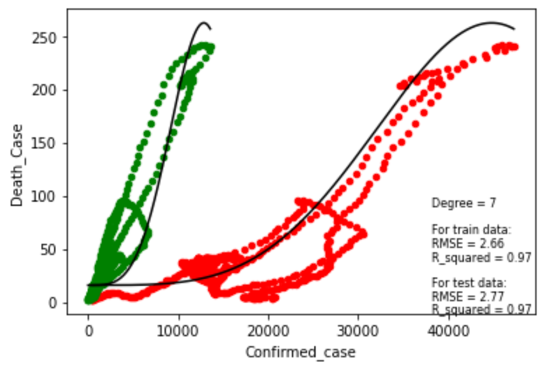


Fig: PR_L2 with clean data with degree 7

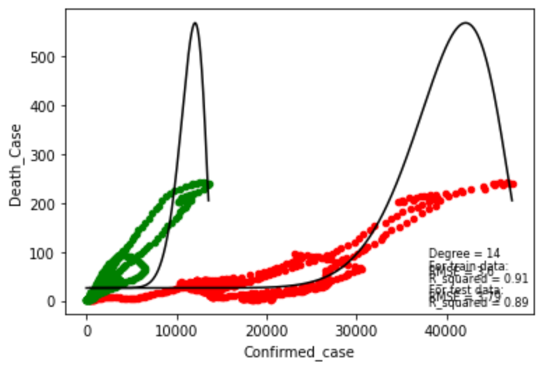


Fig: PR_L2 with clean data with degree 14

Here we have took the degree of 7 as our analysis because here the difference of train and testing RMSE is minimum although both degree 7 and degree 4 have similar R-squired value.

6) **Lasso Regression (LR)**:

Here we have used L1 penalty over Linear Regression and where alpha is the regularization parameter.

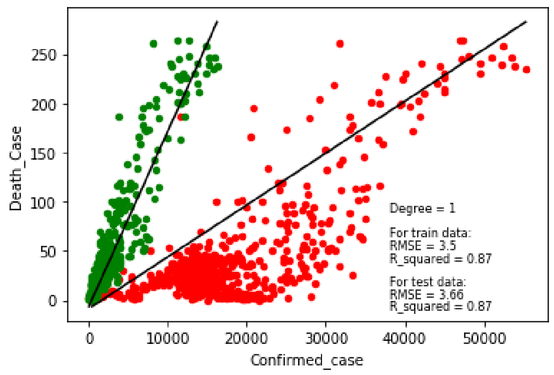


Fig: LR with raw data with alpha=0.1

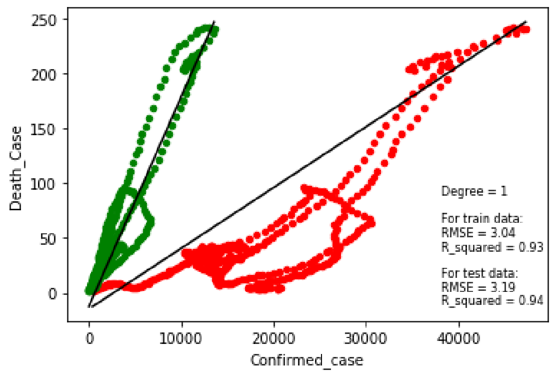


Fig: LR with clean data with alpha=0.3, 0.5 and 0.7

7) **Ridge Regression (RR)**:

Here we have used L2 penalty over Linear Regression and where alpha is the regularization parameter.

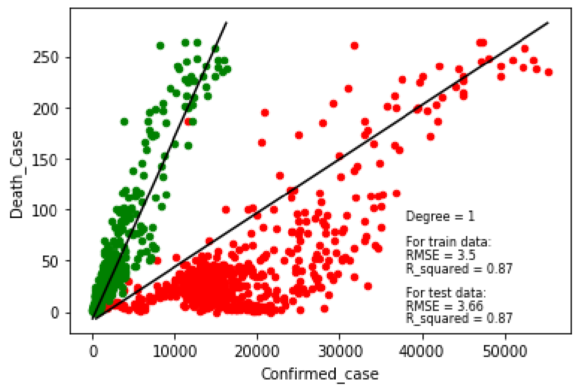


Fig: RR with raw data with alpha=0.7

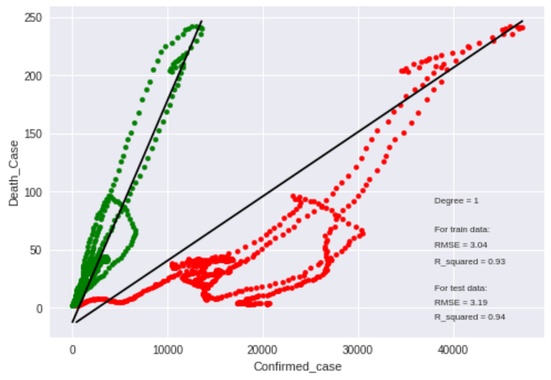


Fig: RR with clean data with alpha=0.3, 0.5 and 0.7

8) **ElasticNet Regression (ER)**:

Here we have used both L1 and L2 penalty over Linear Regression and where alpha is the regularization parameter.

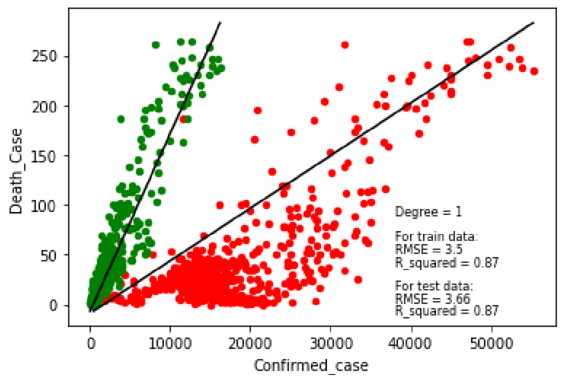


Fig: ER with raw data with alpha=0.7

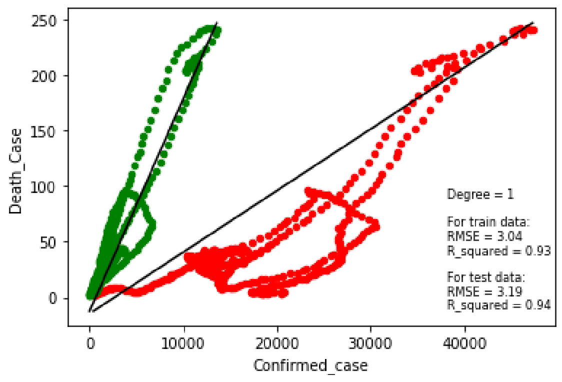
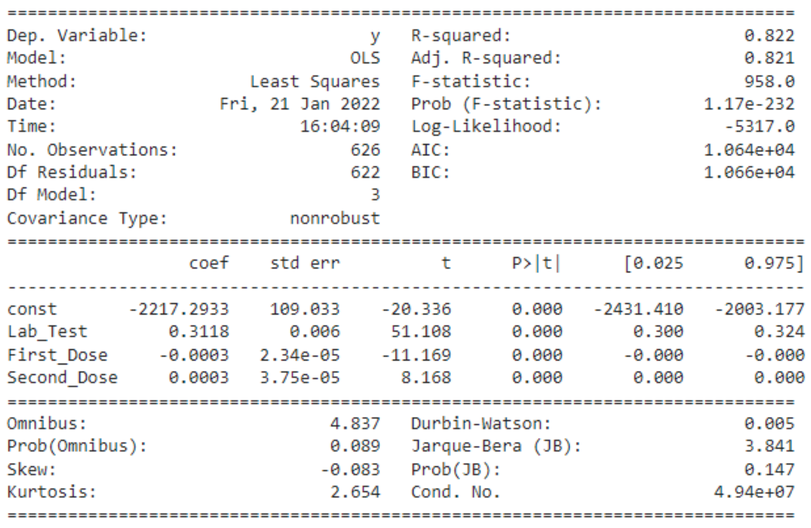


Fig: ER with clean data with l1_ratio = 0.3, 0.5 and 0.7

2. **Predict the Confirmed Case**:

1) **Ordinary Least Squares (OLS)**:

We have used OLS method to find out the equation of the Linear Regression to predict the confirmed case.



```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.822
Model:                            OLS   Adj. R-squared:                  0.821
Method:                 Least Squares   F-statistic:                     958.0
Date:                Fri, 21 Jan 2022   Prob (F-statistic):          1.17e-232
Time:                        16:04:09   Log-Likelihood:                -5317.0
No. Observations:                 626   AIC:                         1.064e+04
Df Residuals:                     622   BIC:                         1.066e+04
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -2217.2933    109.033    -20.336      0.000   -2431.410   -2003.177
Lab_Test        0.3118      0.006     51.108      0.000       0.300       0.324
First_Dose     -0.0003   2.34e-05    -11.169      0.000      -0.000      -0.000
Second_Dose     0.0003   3.75e-05      8.168      0.000       0.000       0.000
==============================================================================
Omnibus:                        4.837   Durbin-Watson:                   0.005
Prob(Omnibus):                  0.089   Jarque-Bera (JB):                3.841
Skew:                          -0.083   Prob(JB):                        0.147
Kurtosis:                       2.654   Cond. No.                     4.94e+07
==============================================================================
```

Fig: OLS Regression Results

From the summary we have found the equation which is,

$$\text{Confirmed Case} = -2217.2933 + 0.3118 \times \text{Lab Test} - 0.0003 \times \text{First Dose} + 0.0003 \times \text{Second Dose}$$

2) **Linear Regression (LR):**

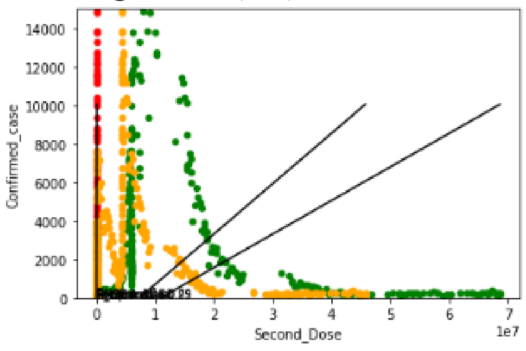


Fig: LR with raw dataset

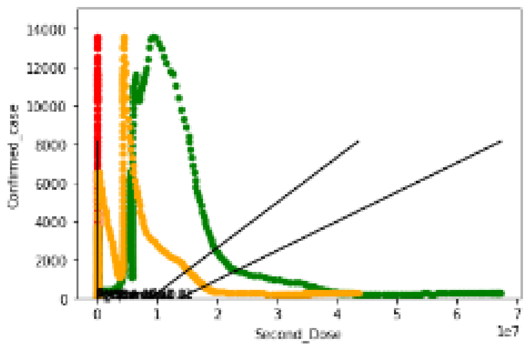


Fig: LR with clean dataset

3) **Polynomial Regression (PR):**

The most important hyperparameter in the PolynomialFeatures() class is degree. We have used various degree so that we can find out which degree performs better.

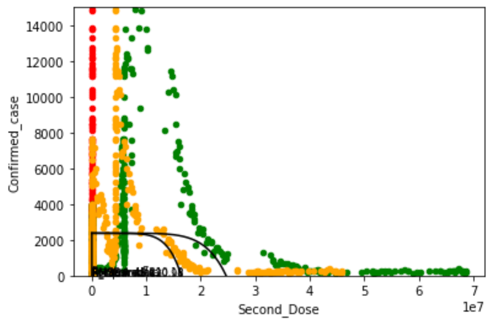


Fig: PR with raw data with degree 4

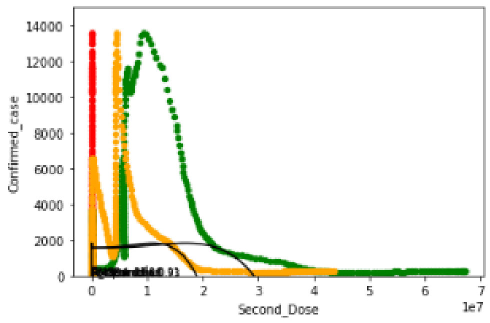


Fig: PR with clean data with degree 4

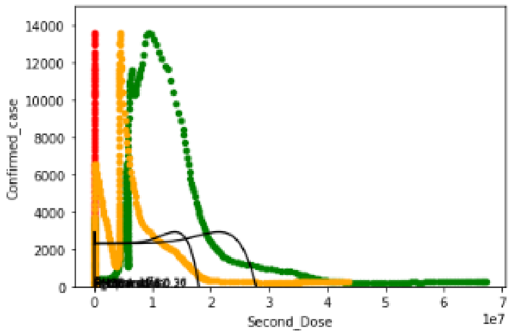


Fig: PR with clean data with degree 7

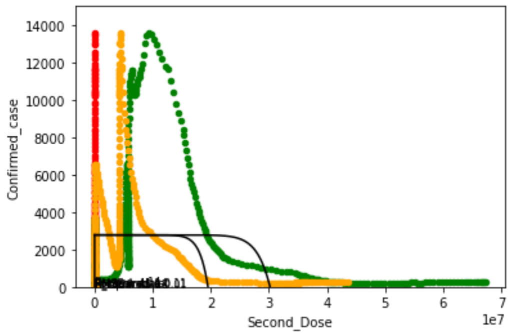


Fig: PR with clean data with degree 14

From these graphs, we can see that a very poor result for Predicting the Confirmed Case. Therefore, we have used Time Series Analysis for future prediction.

3. **Time Series Analysis**:

1. **Stationarity test using Augmented Dickey-Fuller**:

| | |
|---|---|
| Test statistic | -2.662739 |
| p-value | 0.080711 |
| Lags | 17 |
| Number of Observations | 608 |
| Critical Value (1%) | -3.441151 |
| Critical Value (5%) | -2.866305 |
| Critical Value (10%) | -2.569308 |

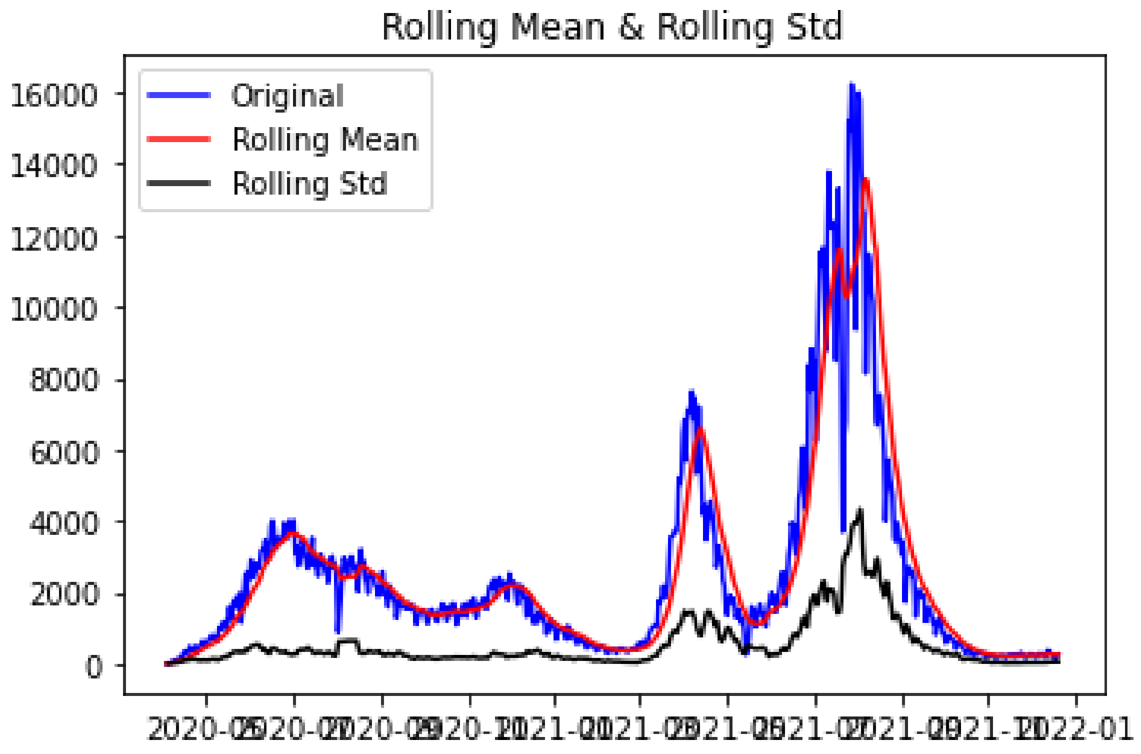
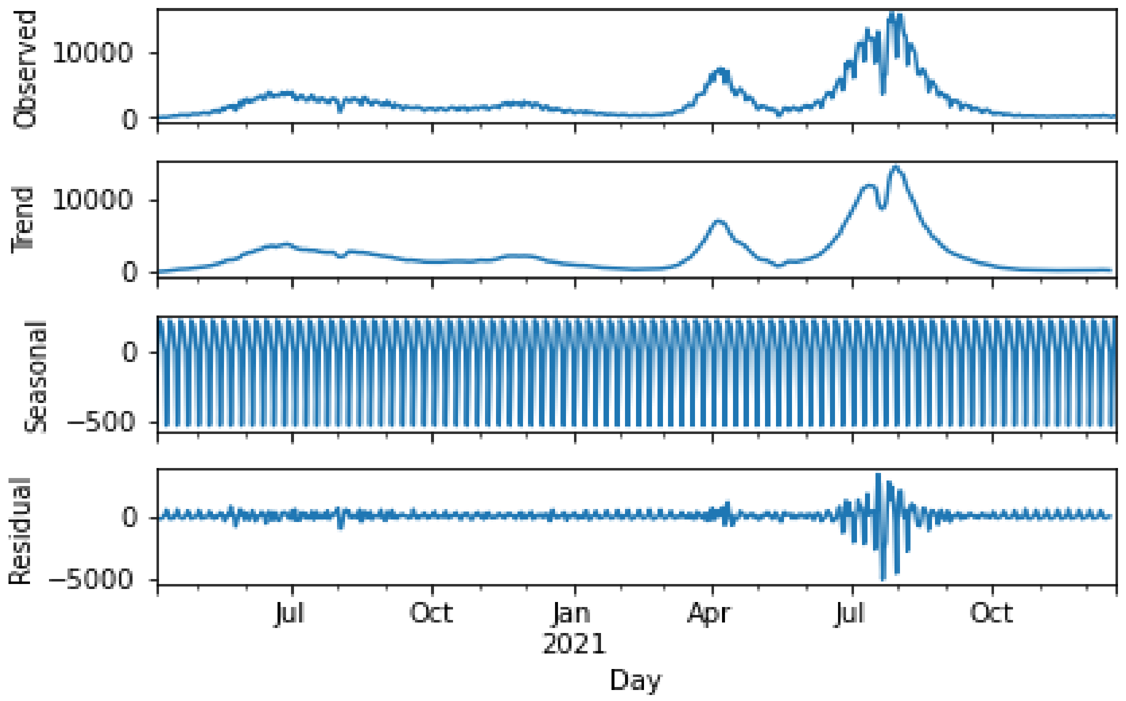


Fig: Stationarity test

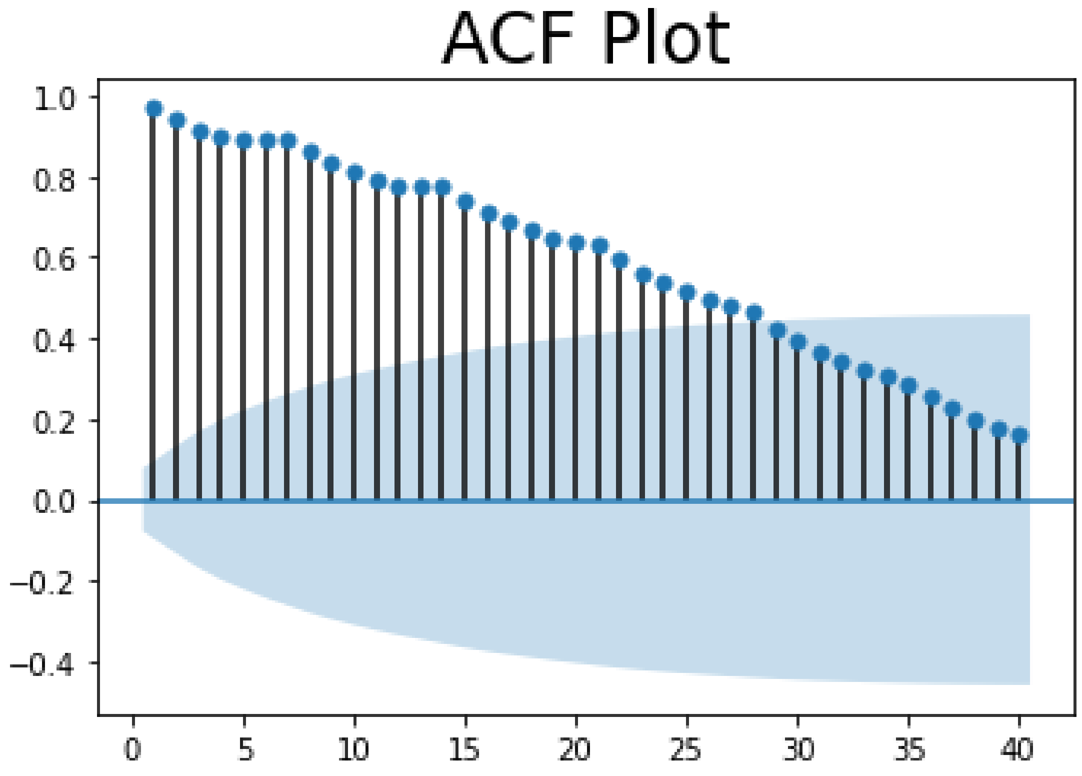
2. **Seasonality Check**:

| **Seasonality** | **Test Result** |
|---|---|
| Trend | Found |
| Seasonal | Not Found |
| Residual | Found in July to August |



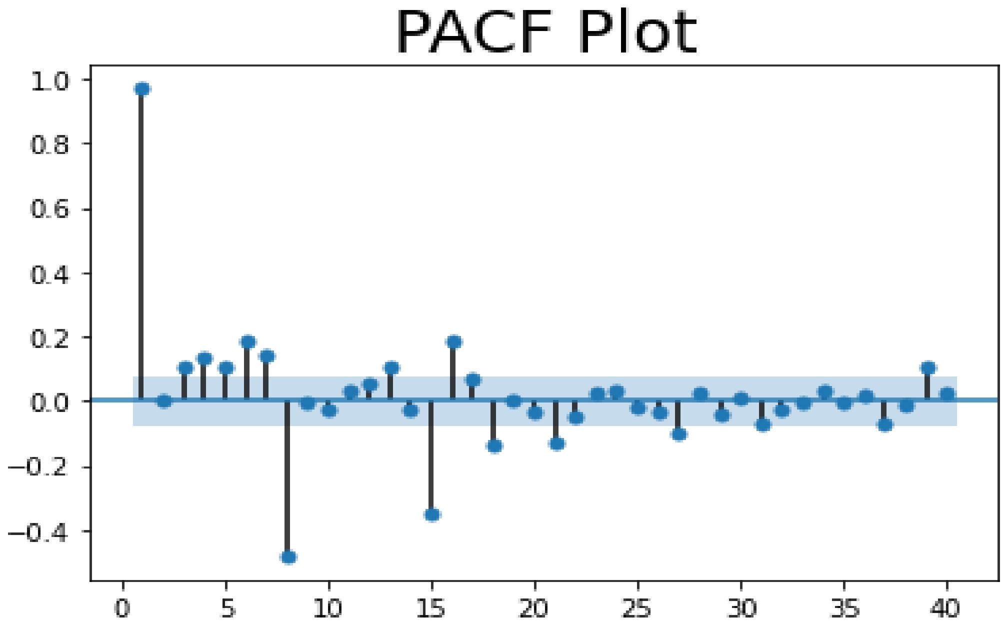
3. **Autocorrelation Plot**:

Autocorrelation found up to t-27 days. Here t means today. And t-i means ith day before from today. Autocorrelation means there is both direct and indirect effect on today for each (t-i)th day.



4. **Partial Autocorrelation Plot**:

   Strong direct autocorrelation found up to t-17 days.



5. **Using ARIMA model for the prediction**:

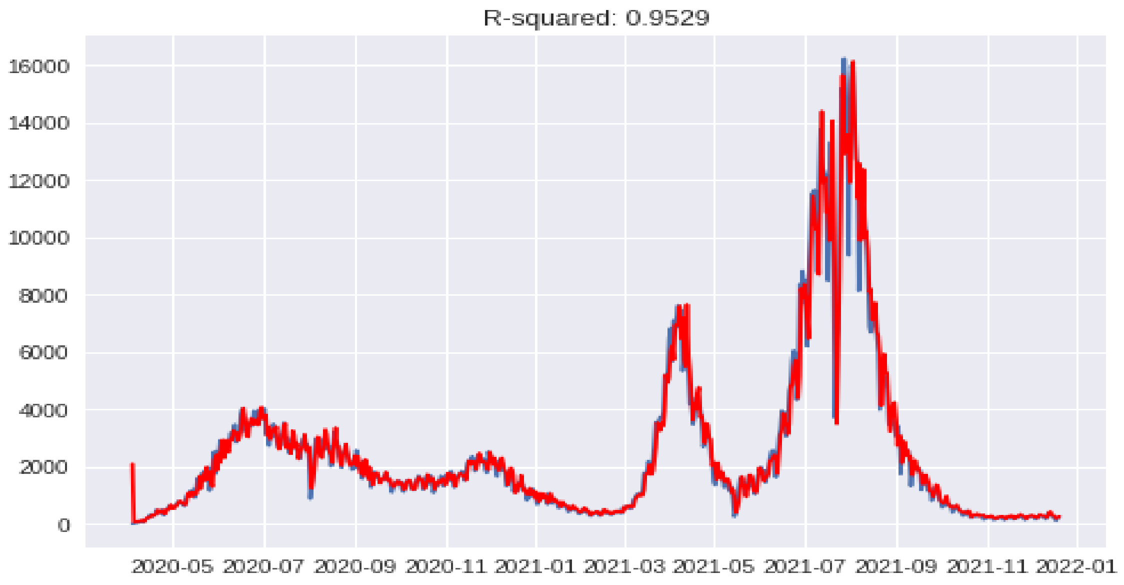
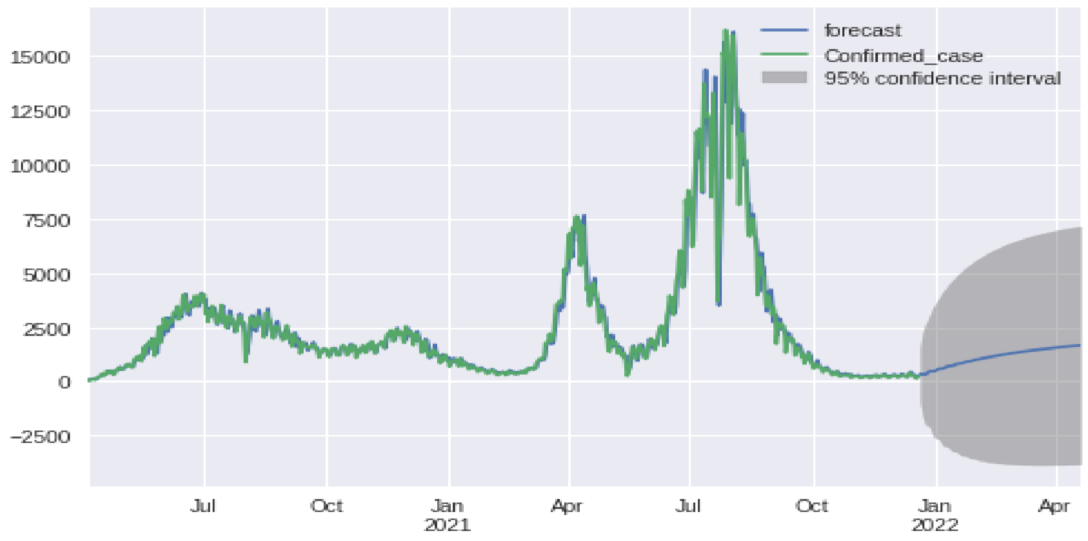


Fig: Actual Confirmed case vs Predicted Confirmed case

6. **Forecasting the Confirmed case rate**:

   It predicted with 95% confidence interval, that from January to April, the confirmed case will be increase slowly. Here we predict for next four month.



4. **Classifications**:

1. **Logistic Regression**:

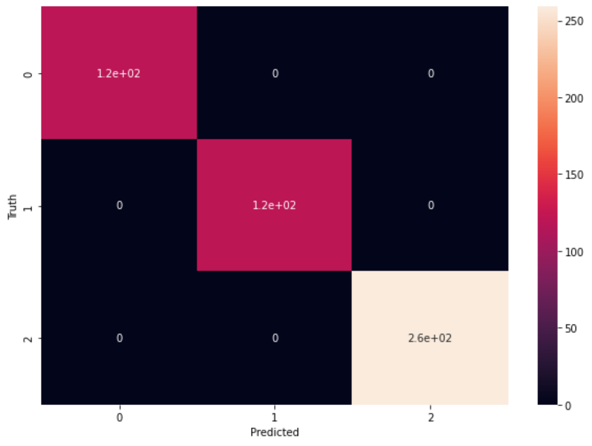
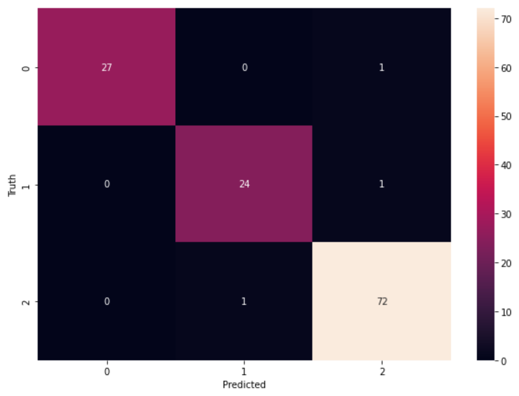


Fig: Confusion matrix of all training data for C = .1, 1, 10, 20



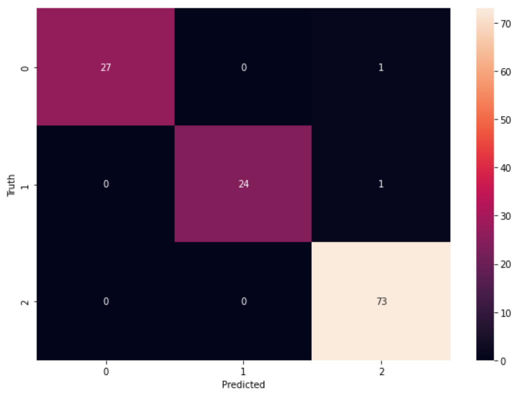


Fig: Confusion matrix of testing data for C = .1

Fig: Confusion matrix of testing data for C = 1

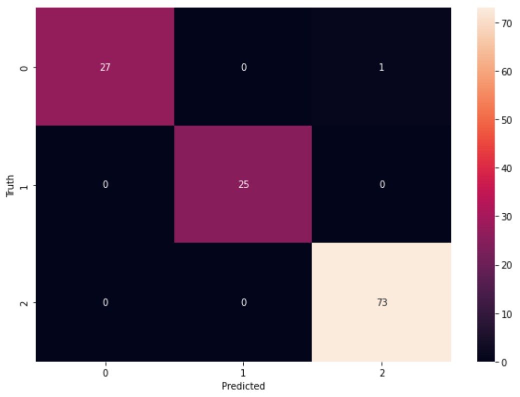


Fig: Confusion matrix of testing data for C = 10

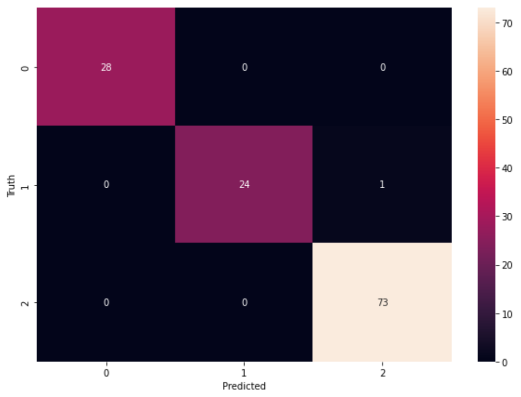


Fig: Confusion matrix of testing data for C = 20

From these confusion matrix we see that, by increasing the C parameter value we are getting more better result but it also over fits the data. Here both C=10 and C=20 has the same number of false value but true positive values are different for them. Here C=10 gives us the valuable result.

2. **Linear Support Vector Classifier:**

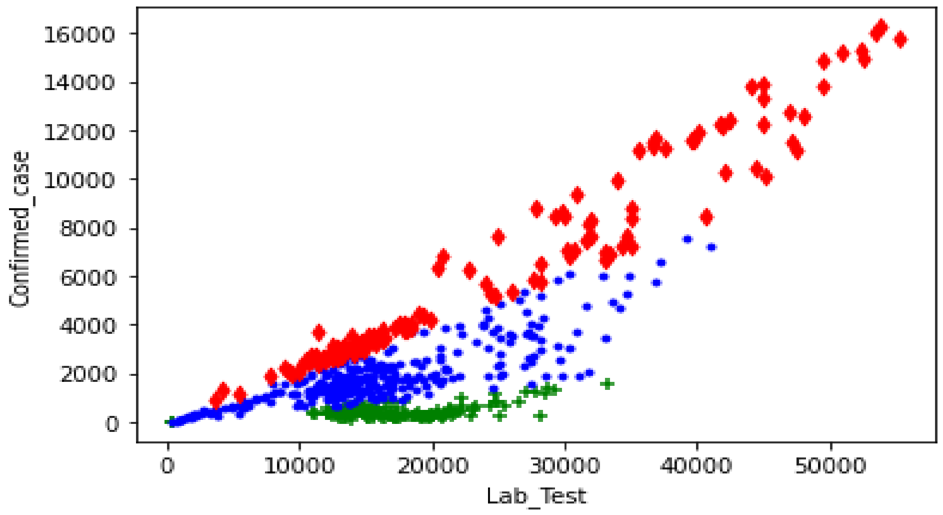


Fig: Risk areas

This plot represents the visualization of when the risk is low(color: green), medium (color: blue) and high(color: red).

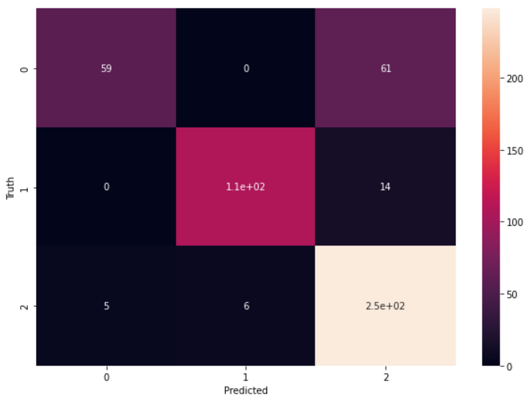


Fig: CF of training data for C = .1

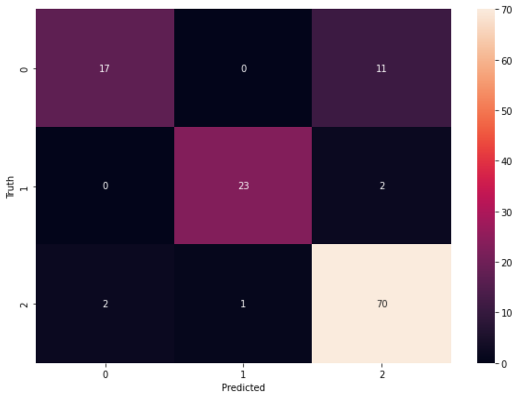


Fig: CF of testing data for C = .1

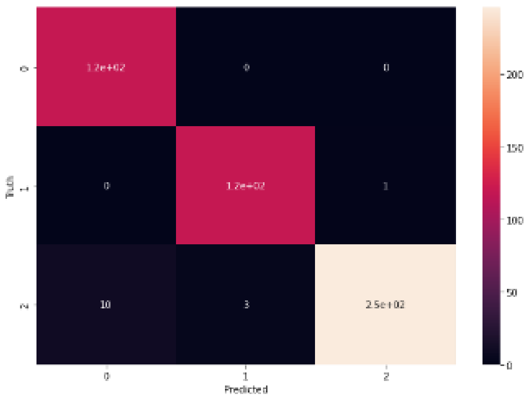


Fig: CF of training data for C = 1

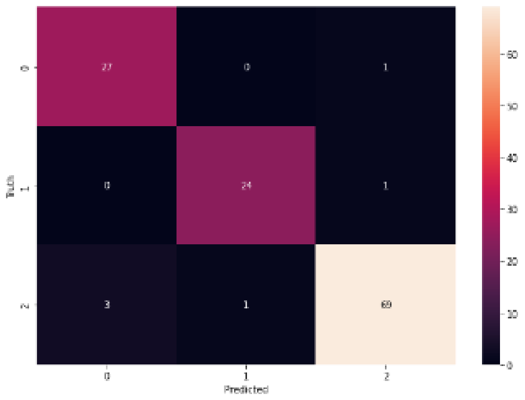


Fig: CF of testing data for C = 1

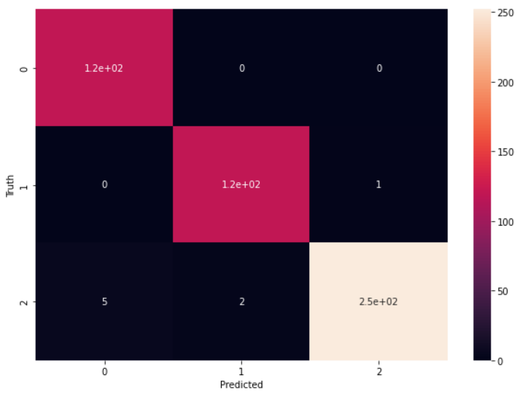


Fig: CF of training data for C = 10

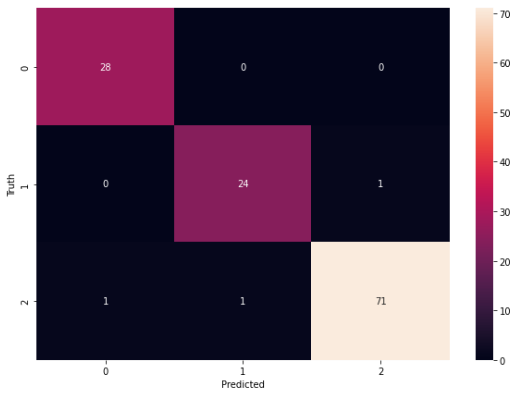


Fig: CF of testing data for C = 10

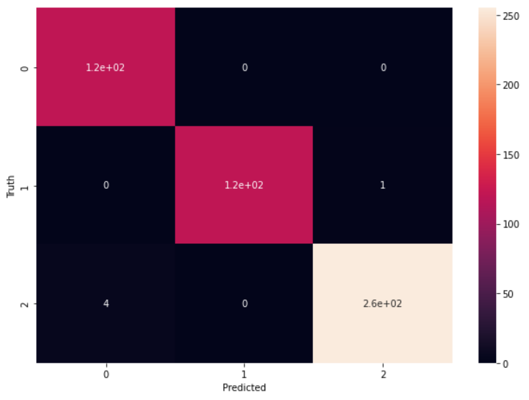


Fig: CF of training data for C = 20

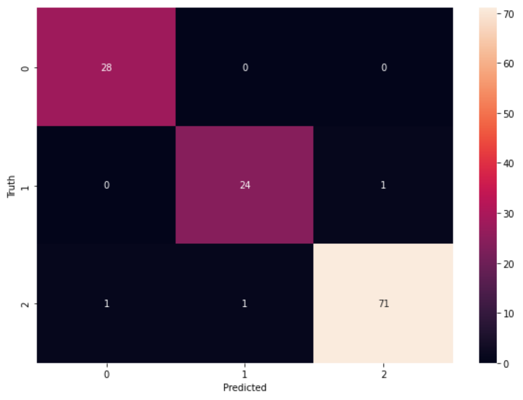


Fig: CF of testing data for C = 20

From these confusion matrix we see that, by increasing the C parameter value we are getting more better result but also it also over fits the data. Here C=10 gives us the valuable result.

## 7. Discussion

In this section we have compared our machine learning modes based on the evaluation scores which we have got from the previous section.

1. **Regression Models**:

The Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R-Squared for every regression model is given below for analysing the performance of the regression models.

| | | **OLS Method** | **Linear Regression** | **Polynomial Regression** | **Polynomial Regression L1** | **Polynomial Regression L2** | **Lasso Regression** | **Ridge Regression** | **Elastic Net Regression** |
|---|---|---|---|---|---|---|---|---|---|
| **Mean Absolute Error** | Train | 9.21 | 9.22 | 5.67 | 6.68 | 7.09 | 9.21 | 9.21 | 9.21 |
| | Test | 10.17 | 10.17 | 6.46 | 6.66 | 7.69 | 10.17 | 10.17 | 10.17 |
| **Mean Squared Error** | Train | 193.18 | 193.18 | 84.43 | 109.55 | 86.48 | 193.18 | 193.18 | 193.18 |
| | Test | 198.92 | 198.92 | 104.46 | 93.74 | 113.57 | 198.92 | 198.91 | 198.91 |
| **Root Mean Squared Error** | Train | 3.04 | 3.04 | 2.38 | 2.58 | 2.66 | 3.04 | 3.04 | 3.03 |
| | Test | 3.19 | 3.19 | 2.54 | 2.58 | 2.77 | 3.18 | 3.19 | 3.18 |
| | Train | 0.929 | 0.929 | 0.969 | 0.959 | 0.968 | 0.929 | 0.929 | 0.929 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| **R-Squared** | Test | 0.938 | 0.938 | 0.967 | 0.971 | 0.965 | 0.938 | 0.938 | 0.938 |

From the table we can see that, for training dataset, Polynomial Regression and Polynomial Regression with L2 regularization has performed better than others and for testing dataset, Polynomial Regression and Polynomial Regression with L1 regularization has performed better than others. Here OLS method, Linear, Lasso, Ridge and Elastic Net Regression didn't give us very poor performance compared to others. Therefore, we can say that, among our regression models, overall Polynomial regression model gives us better performance.

From the table we can also see that, a very similar performance for our OLS method, Linear, Lasso, Ridge and Elastic Net Regression although OLS method uses a simple mathematics and others uses machine learning to give us the result. The reason behind this is our dataset is not too complicated to work with it. Therefore after giving the regularization our models work like as OLS method.

Here Polynomial regression, Polynomial regression with L1 regularization and Polynomial regression with L2 regularization gives us a curvey line where other models gave us linear line for regression. From the graphs we can see that, our dataset is not trended as linear rather than it is more likely trended as curvey. Therefore, here polynomial regression models are performed better.

2. **Time series**:

Since we are getting poor results to predict the confirmed case using the regression method, we decide to apply the time series analysis method to predict or forecast the total number of confirm cases on a particular day.

We used the ARIMA model for forecasting. First of all, we check whether our dataset is stationary or not. By using the Augmented Dickey-Fuller test, we found that our dataset is not stationary. As our dataset is not stationary that's why we try to find the seasonality of our dataset. Using the seasonal_decompose method we found that our dataset has trends but not has any seasonality.

After that, we plot the ACF and PACF graph to find out those lags that are directly or indirectly autocorrelated with the t-th day. We found that up to 27 lags have both direct and indirect autocorrelation but only 5 to 7 lags have direct autocorrelation. Thus we train our model according to those lags. We use 5 lags to train our model. And it works very well to predict and forecast the total number of confirmed cases on a particular day.

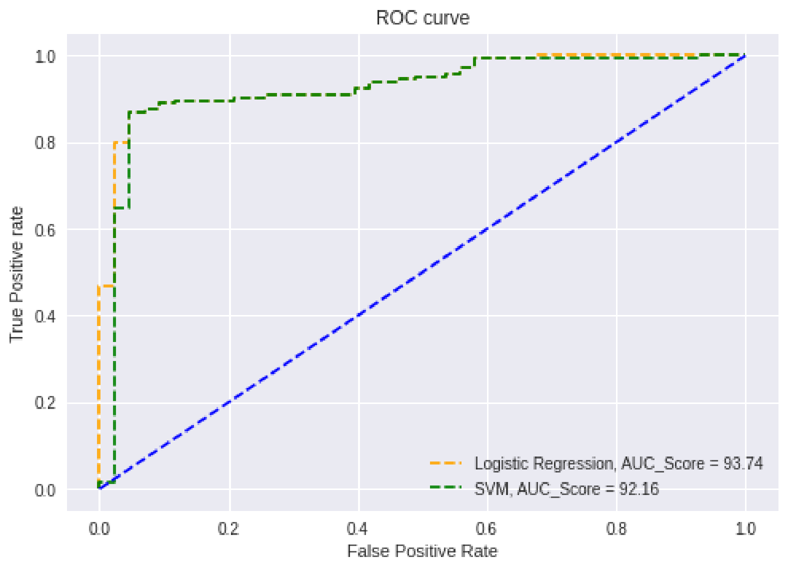
In our Performance Evaluation section, we show the result of the forecast for the upcoming four months. Our forecast is almost perfectly matched with the real-time situation. Our model predicts that the number of the confirmed cases will be increased from January and it is happening actually.

3. **Classification Models**:

The Accuracy Score, Precision Score, Recall Score, Area Under Curve (AUC) for every classification model is given below for analysing the performance of the classification models.

| | | **Accuracy Score** | **Precision Score** | **Recall Score** |
|---|---|---|---|---|
| **Logistic Regression** | Train | 1.0 | [1.  1.  1.] | [1.  1.  1.] |
| | Test | 0.984 | [1.  1.  0.97] | [0.96  0.96  1.] |
| **SVM** | Train | 0.984 | [0.96  0.98  0.99] | [1.  0.99  0.97] |
| | Test | 0.976 | [0.96  0.96  0.98] | [1.  0.96  0.97] |

From the table we can see that, for training dataset, Logistic Regression has performed better than SVM and also for testing dataset, Logistic Regression has performed better than SVM.



For ROC curve, higher the Area Under Curve (AUC), the better the model is at predicting 0 classes as 0 and 1 classes as 1. So, the Higher the AUC, the better the model is at distinguishing between Confirmed Case with High Risk and Low Risk. So we can more precisely say that, Logistic Regression has performed better than SVM.

SVM tries to finds the “best” margin (distance between the line and the support vectors) that separates the classes and this reduces the risk of error on the data, while logistic regression does not, instead it can have different decision boundaries with different weights that are near the optimal point. Also SVM works well with unstructured and semi-structured data like text and images while logistic regression works with already identified independent variables.[4]

## 8. Conclusion

This project has increased our knowledge on making different type of models & predict the accuracy level in respect with different type of information's. By applying several models, it gave us a new incite and taste of working in Machine Learning. We are more intrigued and motivated as we are confident enough in our ability to take a further difficult challenge to improve ourselves after completing this project.

In this project we had faced the challenge when we were pre-processing the fluctuating data. At first, we have merged the three datasets, then we tried to standardize and normalize the dataset, but it was not giving us any significant advantage over the dataset. Then we had calculate the moving average of 14 days and created a new dataset with the moving average. After doing all of that, we have worked with our data.

Working with this project, we have learned about a lot of new and interesting things. We have gained a great perception on Data Science and it has made us very interested to learn new things as well. Also we are now able to analyse the real-life data which will be beneficial for our future learning.

Thank you sir for giving us this great opportunity of doing this wonderful machine learning project. Through this project and entire course, we have really grasped the fundamentals of data science firmly.

## References

1. https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
3. https://mkatzenbach8.medium.com/introduction-to-ordinary-least-squares-ols-using-statsmodels-3329d120eadd#:~:text=OLS%20or%20Ordinary%20Least%20Squares,evaluating%20a

%20linear%20regression%20model.&text=This%20model%20is%20created%20by,going%20to%20use%20in%20StatsModels.

4. https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16

5. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/#:~:text=time%20series%20forecasts.-,ARIMA%20is%20an%20acronym%20that%20stands%20for%20AutoRegressive%20Integrated%20Moving,adds%20the%20notion%20of%20integration.&text=AR%3A%20Autoregression.,some%20number%20of%20lagged%20observations

## Appendix

```
"""303-project.ipynb

Original file is located at
    https://colab.research.google.com/drive/19tDw1fLEq8wGV-asQstCQVhWmjH8PbSE
"""

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet,
LogisticRegression
from sklearn.svm import SVC

from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt
import seaborn as sns

"""# Data Pre Processing """

data_covid_dataset = pd.read_csv('covid_dataset.csv')
data_covid_dataset['Day'] = pd.to_datetime(data_covid_dataset['Day'])
data_covid_dataset.set_index('Day', inplace=True, drop=True)

data_fd = pd.read_csv('covid_first_dose.csv')
data_fd['Day'] = pd.to_datetime(data_fd['Day'])
data_fd.set_index('Day', inplace=True, drop=True)

data_sd = pd.read_csv('covid_second_dose.csv')
```

```
data_sd['Day'] = pd.to_datetime(data_sd['Day'])
data_sd.set_index('Day', inplace=True, drop=True)

data =  pd.concat([data_covid_dataset, data_fd, data_sd], axis=1,
ignore_index=True)
data.rename(columns =
{0:"Lab_Test",1:"Confirmed_case",2:"Death_Case",3:"First_Dose",4:"Second_Dose
",}, inplace = True)

print(data.tail(20))
cnt = 0

data.isnull().sum()

data.fillna(0, inplace=True)

print(data.duplicated().sum())
data.drop_duplicates(inplace=True)
print(data.isnull().sum())

corr = data.corr()

#corelation analysis
plt.figure(figsize=(10,10))
sns.heatmap(corr,cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size'
:8}, cmap='Reds')

if cnt == 0:
    sum1 = 0
    sum2 = 0
    for i in range(data.shape[0]):
        sum1 += data.iloc[i,3]
        data.iloc[i,3] = sum1

        sum2 += data.iloc[i,4]
        data.iloc[i,4] = sum2

    cnt += 1

corr = data.corr()

#corelation analysis
plt.figure(figsize=(10,10))
sns.heatmap(corr,cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size'
:8}, cmap='Reds')

df2 =  data.copy()
print(df2.head())
X_Death = df2.drop(columns = ['Death_Case', 'First_Dose', 'Second_Dose'])
Y_Death =  np.array(df2[['Death_Case']])

X_Confirmed = df2.drop(columns = ['Confirmed_case','Death_Case'])
Y_Confirmed =  np.array(df2[['Confirmed_case']])

df2["Lab_Test"] = df2["Lab_Test"].rolling(window=14, min_periods=1).mean()
df2["Lab_Test"] = df2["Lab_Test"].round(decimals = 0)
```

```
df2["Confirmed_case"] = df2["Confirmed_case"].rolling(window=14,
min_periods=1).mean()
df2["Confirmed_case"] = df2["Confirmed_case"].round(decimals = 0)

df2["Death_Case"] = df2["Death_Case"].rolling(window=14,
min_periods=1).mean()
df2["Death_Case"] = df2["Death_Case"].round(decimals = 0)

df2['First_Dose'] = df2['First_Dose'].rolling(window=14,
min_periods=1).mean()
df2['First_Dose'] = df2['First_Dose'].round(decimals = 0)

df2['Second_Dose'] = df2['Second_Dose'].rolling(window=14,
min_periods=1).mean()
df2['Second_Dose'] = df2['Second_Dose'].round(decimals = 0)

X_Death_2 = df2.drop(columns = ['Death_Case', 'First_Dose', 'Second_Dose'])
Y_Death_2 =  np.array(df2[['Death_Case']])

X_Confirmed_2 = df2.drop(columns = ['Confirmed_case','Death_Case'])
Y_Confirmed_2 =  np.array(df2[['Confirmed_case']])

"""# Functions"""

def evaluate_model(y, y_predict, flg):
    if flg == 1:
        print('MAE: ', metrics.mean_absolute_error(y, y_predict))
        print('MSE: ', metrics.mean_squared_error(y, y_predict))
        print('RMSE: ', np.sqrt(metrics.mean_absolute_error(y, y_predict)))
        print('R-squared: ', metrics.r2_score(y, y_predict))
        print()
    elif flg == 2:
        print("Confusion Matrix: ")
        print(confusion_matrix(y, y_predict))
        plt.figure(figsize=(10,7))
        sns.heatmap(confusion_matrix(y, y_predict), annot=True)
        plt.xlabel('Predicted')
        plt.ylabel('Truth')

        print("Accuracy Score:  ",accuracy_score(y, y_predict))
        print('Precision Score: ', precision_score(y, y_predict,
average=None))
        print('Recall Score:    ', recall_score(y, y_predict, average=None))
        print()
    elif flg == 3:
        RMSE = np.sqrt(metrics.mean_absolute_error(y, y_predict))
        R_squared = metrics.r2_score(y, y_predict)
        return RMSE, R_squared
    else:
        print("Invalid Flg value!")

def plot_model(x, y, x_test, y_test, model_1, model_2 = None, degree=1):
    X_seq = np.linspace(x.min(), x.max(), 100).reshape(-1,1)
    plt.figure()
    plt.scatter(x, y)
    plt.plot(X_seq, model_1.predict(X_seq), color = "black")

    RMSE_train, R_squared_train = evaluate_model(y, model_1.predict(x), 3)
    RMSE_test, R_squared_test = evaluate_model(y_test, 
model_1.predict(x_test), 3)

    plt.text(38100, 90, f'Degree = {degree}', fontsize = 8)

    plt.text(38100, 65, 'For train data: ', fontsize = 8)
    plt.text(38100, 52, f'RMSE = {round(RMSE_train,2)}', fontsize = 8)
    plt.text(38100, 38, f'R_squared = {round(R_squared_train,2)}', fontsize = 
8)

    plt.text(38100, 15, 'For test data: ', fontsize = 8)
    plt.text(38100, 2, f'RMSE = {round(RMSE_test,2)}', fontsize = 8)
    plt.text(38100, -10, f'R_squared = {round(R_squared_test,2)}', fontsize = 
8)

    if model_2 != None:
        plt.plot(X_seq, model_2.predict(X_seq), color = "red")

def plot_model_2(df, x, y, x_test, y_test, model_1, model_2 = None, 
degree=1):
    x_sec = np.linspace(x.min(), x.max(), 1000)

    plt.figure()
    ax1 = df.plot(kind='scatter', x='Lab_Test', y='Confirmed_case', 
color='r')
    ax2 = df.plot(kind='scatter', x='First_Dose', y='Confirmed_case', 
color='g', ax=ax1)
    ax2 = df.plot(kind='scatter', x='Second_Dose', y='Confirmed_case', 
color='orange', ax=ax1)
    plt.plot(x_sec, model_1.predict(x_sec), color = "black")

    RMSE_train, R_squared_train = evaluate_model(y, model_1.predict(x), 3)
    RMSE_test, R_squared_test = evaluate_model(y_test, 
model_1.predict(x_test), 3)

    plt.text(38100, 90, f'Degree = {degree}', fontsize = 8)

    plt.text(38100, 65, 'For train data: ', fontsize = 8)
    plt.text(38100, 52, f'RMSE = {round(RMSE_train,2)}', fontsize = 8)
    plt.text(38100, 38, f'R_squared = {round(R_squared_train,2)}', fontsize = 
8)

    plt.text(38100, 15, 'For test data: ', fontsize = 8)
    plt.text(38100, 2, f'RMSE = {round(RMSE_test,2)}', fontsize = 8)
    plt.text(38100, -10, f'R_squared = {round(R_squared_test,2)}', fontsize = 
8)

    plt.ylim(0, 15000)
    if model_2 != None:
        plt.plot(x_sec, model_2.predict(x_sec), color = "blue")

def plot_model_3(df, x, y, x_test, y_test, model_1, model_2 = None, 
degree=1):
    x_sec = np.linspace(x.min(), x.max(), 1000)
    plt.figure()
    ax1 = df.plot(kind='scatter', x='Lab_Test', y='Death_Case', color='r')
    ax2 = df.plot(kind='scatter', x='Confirmed_case', y='Death_Case',
color='g', ax=ax1)
    plt.plot(x_sec, model_1.predict(x_sec), color = "black")

    RMSE_train, R_squared_train = evaluate_model(y, model_1.predict(x), 3)
    RMSE_test, R_squared_test = evaluate_model(y_test,
model_1.predict(x_test), 3)

    plt.text(38100, 90, f'Degree = {degree}', fontsize = 8)

    plt.text(38100, 65, 'For train data: ', fontsize = 8)
    plt.text(38100, 52, f'RMSE = {round(RMSE_train,2)}', fontsize = 8)
    plt.text(38100, 38, f'R_squared = {round(R_squared_train,2)}', fontsize =
8)

    plt.text(38100, 15, 'For test data: ', fontsize = 8)
    plt.text(38100, 2, f'RMSE = {round(RMSE_test,2)}', fontsize = 8)
    plt.text(38100, -10, f'R_squared = {round(R_squared_test,2)}', fontsize =
8)

    if model_2 != None:
        plt.plot(x_sec, model_2.predict(x_sec), color = "blue")


def OLS(X, Y, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    X_train = sm.add_constant(X_train)
    X_test = sm.add_constant(X_test)
    ols_model = sm.OLS(Y_train,X_train)
    results = ols_model.fit()

    '''print(X_train.shape, X_test.shape)

    print(X_train.head())

    print(X_test.head())'''

    Y_pred_train = results.predict(X_train)
    Y_pred_test = results.predict(X_test)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)

    print(results.summary())


def Linear_Regression(df, X, Y, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    linear_regression = LinearRegression()
    linear_regression.fit(X_train, Y_train)
    Y_pred_train = linear_regression.predict(X_train)
    Y_pred_test = linear_regression.predict(X_test)

    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, linear_regression)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test, 
linear_regression)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, linear_regression)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test, 
linear_regression)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return linear_regression

def Polynomial_Regression(df, X, Y, degree=2, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 
ratio, random_state = 0)
    polynomial_regression = make_pipeline(PolynomialFeatures(degree), 
LinearRegression())
    polynomial_regression.fit(X_train, Y_train)
    Y_pred_train = polynomial_regression.predict(X_train)
    Y_pred_test = polynomial_regression.predict(X_test)

    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, 
polynomial_regression, degree=degree)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test, 
polynomial_regression, degree=degree)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, 
polynomial_regression, degree=degree)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test, 
polynomial_regression, degree=degree)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return polynomial_regression

def Polynomial_Regression_L1(df, X, Y, degree=2, alpha = 0.5, ratio = 0.2, 
flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    polynomial_regression_L1 = make_pipeline(PolynomialFeatures(degree),
Lasso(alpha=alpha))
    polynomial_regression_L1.fit(X_train, Y_train)
    Y_pred_train = polynomial_regression_L1.predict(X_train)
    Y_pred_test = polynomial_regression_L1.predict(X_test)

    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test,
polynomial_regression_L1, degree=degree)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test,
polynomial_regression_L1, degree=degree)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test,
polynomial_regression_L1, degree=degree)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test,
polynomial_regression_L1, degree=degree)


    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return polynomial_regression_L1

def Polynomial_Regression_L2(df, X, Y, degree=2, alpha = 0.5, ratio = 0.2,
flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    polynomial_regression_L2 = make_pipeline(PolynomialFeatures(degree),
Ridge(alpha=alpha))
    polynomial_regression_L2.fit(X_train, Y_train)
    Y_pred_train = polynomial_regression_L2.predict(X_train)
    Y_pred_test = polynomial_regression_L2.predict(X_test)


    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test,
polynomial_regression_L2, degree=degree)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test,
polynomial_regression_L2, degree=degree)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test,
polynomial_regression_L2, degree=degree)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test,
polynomial_regression_L2, degree=degree)
```

```
    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return polynomial_regression_L2

def Lasso_Regression(df, X, Y, alpha = 0.5, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    lasso = Lasso(alpha = alpha)
    lasso.fit(X_train, Y_train)
    Y_pred_train = lasso.predict(X_train)
    Y_pred_test = lasso.predict(X_test)


    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, lasso)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test, lasso)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, lasso)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test, lasso)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return lasso

def Ridge_Regression(df, X, Y, alpha = 0.5, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    ridge = Ridge(alpha = alpha)
    ridge.fit(X_train, Y_train)
    Y_pred_train = ridge.predict(X_train)
    Y_pred_test = ridge.predict(X_test)


    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, ridge)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test, ridge)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, ridge)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test, ridge)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)
    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return ridge

def ElasticNet_Regression(df, X, Y, alpha = 0.5, l1_ratio = 0.5, ratio = 0.2, 
flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 
ratio, random_state = 0)
    elasticNet = ElasticNet(alpha = alpha, l1_ratio=l1_ratio)
    elasticNet.fit(X_train, Y_train)
    Y_pred_train = elasticNet.predict(X_train)
    Y_pred_test = elasticNet.predict(X_test)


    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, elasticNet)
        else:
            plot_model_3(df, X_train, Y_train, X_test, Y_test, elasticNet)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, X_test, Y_test, elasticNet)
        else:
            plot_model_2(df, X_train, Y_train, X_test, Y_test, elasticNet)

    print("For Train Dataset: ")
    evaluate_model(Y_train, Y_pred_train, 1)

    print("For Test Dataset: ")
    evaluate_model(Y_test, Y_pred_test, 1)
    return elasticNet

def Linear_VS_Polynomial_Regression(df, X, Y, degree=2, ratio = 0.2, flg=0):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 
ratio, random_state = 0)

    linear_regression = Linear_Regression(df, X,Y)
    polynomial_regression = Polynomial_Regression(df, X,Y, degree=degree)

    if flg == 0:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, linear_regression, 
polynomial_regression)
        else:
            plot_model_3(df, X_train, Y_train, linear_regression, 
polynomial_regression)
    else:
        if X.shape[1] == Y.shape[1]:
            plot_model(X_train, Y_train, linear_regression, 
polynomial_regression)
        else:
            plot_model_2(df, X_train, Y_train, linear_regression, 
polynomial_regression)


def Logistic_Regression(X, Y, c=10, ratio = 0.2):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    logistic_regression = LogisticRegression(C = c)
    logistic_regression.fit(X_train, Y_train)
    Y_pred_train = logistic_regression.predict(X_train)
    Y_pred_test = logistic_regression.predict(X_test)

    print("------------------------Logistic_Regression-----------------------
-------------------------")
    print("-------------------------------------------C=",c)


    prediction_train = list(map(round,Y_pred_train))
    prediction_test = list(map(round,Y_pred_test))

    print("For Train Dataset: ")
    evaluate_model(Y_train, prediction_train, 2)

    print("For Test Dataset: ")
    evaluate_model(Y_test, prediction_test, 2)

    return logistic_regression


def SVM(X, Y, gamma='auto', ratio = 0.2, c=10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
    svm_model = make_pipeline(preprocessing.StandardScaler(),
SVC(kernel='linear',C=c, gamma=gamma))
    svm_model.fit(X_train, Y_train)

    Y_pred_train = svm_model.predict(X_train)
    Y_pred_test = svm_model.predict(X_test)


    print("------------------------SVM---------------------------------------
---------")


    prediction_train = list(map(round,Y_pred_train))
    prediction_test = list(map(round,Y_pred_test))

    print("For Train Dataset: ")
    evaluate_model(Y_train, prediction_train, 2)

    print("For Test Dataset: ")
    evaluate_model(Y_test, prediction_test, 2)
    return svm_model


def auc_score_roc_curve(X,Y):
  c = 10
  ratio = 0.3
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
ratio, random_state = 0)
  logistic_regression = LogisticRegression(C = c)
  logistic_regression.fit(X_train, Y_train)
  svm_model = SVC(probability=True)
  svm_model.fit(X_train,Y_train)

  pred_prob1 = logistic_regression.predict_proba(X_test)
  pred_prob2 = svm_model.predict_proba(X_test)

  # roc curve for models
  fpr1, tpr1, thresh1 = roc_curve(Y_test, pred_prob1[:,1], pos_label=1)
  fpr2, tpr2, thresh2 = roc_curve(Y_test, pred_prob2[:,1], pos_label=1)

  # roc curve for tpr = fpr
  random_probs = [0 for i in range(len(Y_test))]
  p_fpr, p_tpr, _ = roc_curve(Y_test, random_probs, pos_label=1)

  # auc scores
  auc_score1 = roc_auc_score(Y_test, pred_prob1[:,1])
  auc_score2 = roc_auc_score(Y_test, pred_prob2[:,1])

  print(auc_score1*100, auc_score2*100)

  plt.style.use('seaborn')

  # plot roc curves
  plt.plot(fpr1, tpr1, linestyle='--',color='orange', label=f'Logistic
Regression, AUC_Score = {round(auc_score1*100,2)}')
  plt.plot(fpr2, tpr2, linestyle='--',color='green', label=f'SVM, AUC_Score =
{round(auc_score2*100,2)}')
  plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
  # title
  plt.title('ROC curve')
  # x label
  plt.xlabel('False Positive Rate')
  # y label
  plt.ylabel('True Positive rate')

  plt.legend(loc='best')
  #plt.savefig('ROC',dpi=300)
  plt.show();

"""# Classifications"""

df3 = data.copy()
df3['Percentage'] = (df3['Confirmed_case'] / df3['Lab_Test'])*100
condlist = [df3['Percentage']<5, df3['Percentage']>20]
choicelist = ['Low', 'High']
df3['Risk'] = np.select(condlist, choicelist, default='Medium')

df3['Risk'].value_counts()

le = preprocessing.LabelEncoder()
for feature in df3.columns:
    unique_values = df3[feature].unique()
    example_value = unique_values[0]
    if isinstance(example_value, str):
        df3[feature] = le.fit_transform(df3[feature])
```

```
X2 = df3[['Lab_Test','Confirmed_case']]
Y2 =  np.array(df3['Risk'])

X_train, X_test, Y_train, Y_test = train_test_split(X2, Y2, test_size = 0.2,
random_state = 0)

print(X2.shape, X_train.shape, X_test.shape)

pd.DataFrame(Y_test).value_counts()

Logistic_Regression(X2,Y2, c=.1)

Logistic_Regression(X2,Y2, c=1)

Logistic_Regression(X2,Y2, c=10)

Logistic_Regression(X2,Y2, c=20)

low = df3[df3.Risk == 1]
Medium = df3[df3.Risk == 2]
High = df3[df3.Risk == 0]


plt.xlabel('Lab_Test')
plt.ylabel('Confirmed_case')
plt.scatter(low['Lab_Test'], low['Confirmed_case'], color='green',
marker='+')
plt.scatter(Medium['Lab_Test'], Medium['Confirmed_case'], color='blue',
marker='.')
plt.scatter(High['Lab_Test'], High['Confirmed_case'], color='red',
marker='d')

SVM(X2,Y2,c=.1)

SVM(X2,Y2,c=1)

SVM(X2,Y2,c=10)

SVM(X2,Y2,c=20)

"""# ROC"""

df = data.copy()
df['Percentage'] = (df['Confirmed_case'] / df['Lab_Test'])*100
condlist = [df['Percentage']>20]
choicelist = ['High']
df['Risk'] = np.select(condlist, choicelist, default='Low')


le = preprocessing.LabelEncoder()
for feature in df.columns:
    unique_values = df[feature].unique()
    example_value = unique_values[0]
    if isinstance(example_value, str):
        df[feature] = le.fit_transform(df[feature])
```

```
X = df[['Confirmed_case']]
Y =  np.array(df['Risk'])

auc_score_roc_curve(X,Y)

X = df[['Lab_Test']]
Y =  np.array(df['Risk'])

auc_score_roc_curve(X,Y)

"""# Regressions

# Predict the Death Case
"""

OLS(X_Death_2,Y_Death_2)

"""Death Case = -12.4858 + 0.0010 * Lab Test + 0.0158 * Confirmed Case"""

print("--------------------Linear_Regression--------------------")
print()

print("The original dataset: ")
print()
Linear_Regression(data,X_Death,Y_Death)

print("------------------------------")

print("The Clean dataset: ")
print()
Linear_Regression(df2,X_Death_2,Y_Death_2)

print("--------------------Polynomial_Regression--------------------")
print()

print("The original dataset: ")
print()
Polynomial_Regression(data,X_Death,Y_Death, degree=4)

print("------------------------------")

print("The Clean dataset degree=4: ")
print()
Polynomial_Regression(data,X_Death_2,Y_Death_2, degree=4)

print("The Clean dataset degree=7: ")
print()
Polynomial_Regression(data,X_Death_2,Y_Death_2, degree=7)

print("The Clean dataset degree=14: ")
print()
Polynomial_Regression(data,X_Death_2,Y_Death_2, degree=14)

print("--------------------Polynomial_Regression_L1--------------------")
print()

print("The original dataset: ")
print()
Polynomial_Regression_L1(data,X_Death,Y_Death, degree=7, alpha=0.7)

print("------------------------------")

print("The Clean dataset degree=4: ")
print()
Polynomial_Regression_L1(df2,X_Death_2,Y_Death_2, degree=4, alpha=0.7)

print("------------------------------")

print("The Clean dataset degree=7: ")
print()
Polynomial_Regression_L1(df2,X_Death_2,Y_Death_2, degree=7, alpha=0.7)

print("------------------------------")

print("The Clean dataset degree=14: ")
print()
Polynomial_Regression_L1(df2,X_Death_2,Y_Death_2, degree=14, alpha=0.7)

print("--------------------Polynomial_Regression_L2--------------------")
print()

print("The original dataset: ")
print()
Polynomial_Regression_L2(data,X_Death,Y_Death, degree=7)

print("------------------------------")

print("The Clean dataset degree=4: ")
print()
Polynomial_Regression_L2(df2,X_Death_2,Y_Death_2, degree=4)

print("------------------------------")

print("The Clean dataset degree=7: ")
print()
Polynomial_Regression_L2(df2,X_Death_2,Y_Death_2, degree=7)

print("------------------------------")

print("The Clean dataset degree=14: ")
print()
Polynomial_Regression_L2(df2,X_Death_2,Y_Death_2, degree=14)

print("--------------------Lasso_Regression--------------------")
print()

print("The original dataset: ")
print()
Lasso_Regression(data,X_Death,Y_Death, alpha=0.1)

print("------------------------------")

print("The Clean dataset alpha=0.3: ")
print()
```

```
Lasso_Regression(df2,X_Death_2,Y_Death_2, alpha=0.3)

print("------------------------------")

print("The Clean dataset alpha=0.5: ")
print()
Lasso_Regression(df2,X_Death_2,Y_Death_2, alpha=0.5)

print("------------------------------")

print("The Clean dataset alpha=0.7: ")
print()
Lasso_Regression(df2,X_Death_2,Y_Death_2, alpha=0.7)

print("--------------------Ridge_Regression--------------------")
print()

print("The original dataset: ")
print()
Ridge_Regression(data,X_Death,Y_Death, alpha=0.7)

print("------------------------------")

print("The Clean dataset  alpha=0.3: ")
print()
Ridge_Regression(df2,X_Death_2,Y_Death_2, alpha=0.3)

print("------------------------------")

print("The Clean dataset  alpha=0.5: ")
print()
Ridge_Regression(df2,X_Death_2,Y_Death_2, alpha=0.5)

print("------------------------------")

print("The Clean dataset  alpha=0.7: ")
print()
Ridge_Regression(df2,X_Death_2,Y_Death_2, alpha=0.7)

corr = data.corr()
print(corr)
print()

#corelation analysis
plt.figure(figsize=(10,10))
sns.heatmap(corr,cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size'
:8}, cmap='Reds')

print("--------------------ElasticNet_Regression--------------------")
print()

print("The original dataset: ")
print()
ElasticNet_Regression(data,X_Death,Y_Death, l1_ratio=0.5)

print("------------------------------")
print("The Clean dataset l1_ratio=0.3: ")
print()
ElasticNet_Regression(df2,X_Death_2,Y_Death_2, l1_ratio=0.3)

print("------------------------------")

print("The Clean dataset l1_ratio=0.5: ")
print()
ElasticNet_Regression(df2,X_Death_2,Y_Death_2, l1_ratio=0.5)

print("------------------------------")

print("The Clean dataset l1_ratio=0.7: ")
print()
ElasticNet_Regression(df2,X_Death_2,Y_Death_2, l1_ratio=0.7)

"""# Predict the Confirmed Case"""

OLS(X_Confirmed_2,Y_Confirmed_2)

"""Confirme Case = -2217.2933 + 0.3118 * Lab Test - 0.0003 * First Dose +
0.0003 * Second Dose"""

print("--------------------Linear_Regression--------------------")
print()

print("The original dataset: ")
print()
Linear_Regression(data,X_Confirmed,Y_Confirmed, flg=1)

print("------------------------------")

print("The Clean dataset: ")
print()
Linear_Regression(df2,X_Confirmed_2,Y_Confirmed_2, flg=1)

print("--------------------Polynomial_Regression--------------------")
print()

print("The original dataset: ")
print()
Polynomial_Regression(data,X_Confirmed,Y_Confirmed, degree=7,flg=1)

print("------------------------------")

print("The Clean dataset degree=4: ")
print()
Polynomial_Regression(df2,X_Confirmed_2,Y_Confirmed_2, degree=4,flg=1)

print("------------------------------")

print("The Clean dataset degree=7: ")
print()
Polynomial_Regression(df2,X_Confirmed_2,Y_Confirmed_2, degree=7,flg=1)

print("------------------------------")
print("The Clean dataset degree=14: ")
print()
Polynomial_Regression(df2,X_Confirmed_2,Y_Confirmed_2, degree=14,flg=1)

"""# Time Series Analysis"""

import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima_model import ARIMA
from scipy.stats.distributions import chi2

df = data.copy()
print(df.isna().sum())

df.Confirmed_case.plot(figsize=(20,5), title='Death_Case', color = "red")

df_freq = df.asfreq('d')
df_freq_2 = df_freq.drop(columns = ['Lab_Test','Death_Case', 'First_Dose',
'Second_Dose'])
df_freq_2.head(10)

def test_stationarity(timeseries):

    # Determing rolling statistics
    rolmean = timeseries.rolling(window=14, min_periods=1).mean()
    rolstd = timeseries.rolling(window=14, min_periods=1).std()

    # Plot rolling statistics
    orig = plt.plot(timeseries, color='blue', label = 'Original')
    mean = plt.plot(rolmean, color='red', label = 'Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Rolling Std')
    plt.show()

    # perform Dickey-fuller test
    print("Results of Augmented Dickey-fuller test: ")

    dftest = adfuller(timeseries, autolag='AIC')

    dfoutput = pd.Series(dftest[0:4], index=['Test_statistic', 'p-value',
'#Lags Used', 'Number of Observations Used'])

    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value

    print(dfoutput)

print(df_freq_2.iloc[:,0])

test_stationarity(df_freq_2.iloc[:,0])

# Seasonality
```

```
s_dec_additive = seasonal_decompose(df_freq_2, model = 'additive')
s_dec_additive.plot()

# there is no seasonality

# ACF (Direct + Indirect)

sgt.plot_acf(df_freq_2, lags=40, zero = False)
plt.title('ACF Plot', size= 24)

# PACF (Direct)

sgt.plot_pacf(df_freq_2, lags=40, zero = False, method=('ols'))
plt.title('PACF Plot', size= 24)

# ARIMA model
model = ARIMA(df_freq_2, order=(5,0,5))
result_ARIMA = model.fit(disp=-1)
plt.plot(df_freq_2)
plt.plot(result_ARIMA.fittedvalues, color='red')
plt.title('R-squared: %.4f'% metrics.r2_score(df_freq_2.Confirmed_case,
result_ARIMA.fittedvalues))
print('Plotting ARIMA Model')

predict_ARIMA = pd.Series(result_ARIMA.fittedvalues, copy = True)
print(predict_ARIMA.tail())

df_freq_2.tail()

plt.plot(df_freq_2.Confirmed_case)

plt.plot(predict_ARIMA,color='red')

df_freq_2.shape

result_ARIMA.plot_predict(1,626+120)
```