Project 1

Linear Regression Model for Prediction of Corona Cases

Submitted To: Dr. Hasan Sajid
Submitted By: Usman Zaheer
Registration Number: 00000327700
Subject: Machine Learning

Contents

1. Problem Statement:

Write complete code to train a linear/non-linear model for predicting corona cases in USA and world with regularization.

In this model, linear regression model with regularization is implemented for predicting corona cases in USA different states and in the different countries of world.

2. Dataset Details:

i. Gathering and Cleaning:

The following websites/links are used for getting data for USA data set and Worldwide. Data is extracted through manual methods and was cleaned in excel. After that further cleaning was done in Jupyter Notebook.

➢ USA Dataset Websites/Links

- **For corona cases data**
  https://covidtracking.com/data/api
- **For weather states temperature avg data**
  https://www.weatherbase.com/weather/state.php3?c=US&name=United+States+of+America
- **For humidity data of states**
  https://www.currentresults.com/Weather/US/annual-average-humidity-by-state.php-main
  https://www.forbes.com/sites/brianbrettschneider/2018/08/23/oh-the-humidity-why-is-alaska-the-most-humid-state/?sh=72a82a2e330c
- **Population density**
  https://worldpopulationreview.com/state-rankings/state-densities
- **Simple population**
  https://www.worldometers.info/coronavirus/country/us/

➢ Worldwide Cases Dataset Websites/Links

- **For corona cases data:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv
- **For weather temperature data:**
  https://www.timeanddate.com/weather/
- **Population density:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv
- **Human Development Index:**
  https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv

ii. Size of Data:

➢ USA Dataset Size:

The USA data set csv is composed of **14840** Rows and **10** Columns. It is further cleaned and spilited into train, test and validation sets.

➢ Worldwide Dataset Size

The worldwide data set csv is composed of **54720** Rows and **10** Columns. It is further cleaned for useful features and then spilited into train, test and validation sets.

iii. Features Details and Scaling:

➢ USA Dataset Features:

In USA data set there are **8** valid features:

- XO (Bias Term) = 1
- DATE_CODE = In this model "date" is numerically coded. For example:
  Model date starts from 6th December 2020 and ends at 17th March 2020.
  *Here: 17th March 2020 is 264 and 6th December is 0.
- States_Code: States codes are also numerically coded. For example:
  *Here: 1 = Alaska, 2= Alabama and 56=Wyoming.
- 4) Temperatures_(F): Temperatures in F of state at date.
- 5) Humidity (%): Average humidity in % of state at date.
- 6) Population_per_state: Average population in of state at date.
- 7) LandArea (sq miles): Land Area in square miles of particular state .
- 8) POPULATION_DENSITY: Population Density of state.
  Here, We have 1 label : Cases_per_day in particular state at particular date.

	XO	Date_Code	States_Code	Temperatures_(F)	Humidity_(%)	Population_per_state	LandArea_(sq miles)	Population_Density
0	1	0	1	11.6	77.1	731545	570641	1.29
1	1	0	2	46.6	71.6	4903185	50645	96.92
2	1	0	3	41.3	70.9	3017804	52035	58.40
3	1	0	4	38.0	80.0	55138	77	716.00
4	1	0	5	43.6	38.5	7278717	113594	64.95

Figure 1: Features in USA Data Set

In this data, there is 1 label : Cases_per_day in particular state at particular date.

➢ Worldwide Dataset Features:

In worldwide data set there are **6** valid features:

- Xo (Bias Term) = 1
- DATE_CODE = In this model "date" is numerically coded. For example:
  Model date starts from 23 January 2020 and ends at 7th December 2020.
  *Here: 23/01/2020 is 0 and 7th December is 319.
- COUNTRY_CODE: Country codes are also numerically coded. For example:
  *Here: 1 = Afghanistan, 2= Albania and 170=Zimbabwe.
- HUMAN_DEVELOPMENT_INDEX: Average Human development index of state
- POPULATION_DENSITY: Population Density of country.
- TEMPERATURES: Temperatures in C of country at date.

	Xo	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES
0	1	0	0	54.42	0.498	7.0
1	1	1	0	54.42	0.498	7.0
2	1	2	0	54.42	0.498	13.0
3	1	3	0	54.42	0.498	2.0
4	1	4	0	54.42	0.498	6.0

Figure 2: Worldwide data set features

Here, We have 1 label: CASES in particular country at particular date

➢ Scaling of features:

These features are scaled by using the following formula in both data sets

Here, x are the features.

for j in range(0,len(x.columns)):

x= (x-x.min())/(x.max()-x.min())

	XO	Date_Code	States_Code	Temperatures_(F)	Humidity_(%)	Population_per_state	LandArea_(sq miles)	Population_Density
0	1.0	0.0	0.000000	0.140097	0.930456	0.017143	1.000000	0.000000
1	1.0	0.0	0.018182	0.562802	0.798561	0.122869	0.088642	0.008291
2	1.0	0.0	0.036364	0.498792	0.781775	0.075086	0.091079	0.004952
3	1.0	0.0	0.054545	0.458937	1.000000	0.000000	0.000016	0.061967
4	1.0	0.0	0.072727	0.526570	0.004796	0.183074	0.198968	0.005519

Figure 3: USA Data set features after scaling

	Xo	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES
0	1.0	0.000000	0.0	0.002711	0.240401	0.338235
1	1.0	0.003135	0.0	0.002711	0.240401	0.338235
2	1.0	0.006270	0.0	0.002711	0.240401	0.426471
3	1.0	0.009404	0.0	0.002711	0.240401	0.264706
4	1.0	0.012539	0.0	0.002711	0.240401	0.323529

Figure 4: Worldwide data set features after scaling

iv. Code and Methodology:

The methodology opted for this model is that in the following steps:

1) Data import using pandas.
2) Data Cleaning.
3) Assigning features to x and y.
4) Features scaling.

5) Data splits.
6) Initiating Thetas values.
7) Writing functions for Hypothesis, Cost Function and Gradient Descent.
8) Training on training data split.
9) Error metrics and comparison.

3. Mathematical Model Details:

➢ Hypothesis Function:

Linear regression is implemented in this model, so hypothesis taken in this case, simple hypothesis is taken:

Thetas initial values are initiated using this function:

Thetas=np.array([0]*len(x_train.columns))

Here,

Thetas are Model Parameters or weights for each feature.

x_train is the training set features.

Hypothesis is given by:

```
def Hypothesis(Thetas,x_train):
    return Thetas*x_train
```

➢ Cost Function:

Cost function is basically the difference between prediction by the model and the prediction label.

Cost function for linear regression is given by:

```
def Cost_Function(x_train,y_train,Thetas,lambda_):
        H=Hypothesis(Thetas,x_train)
        H=np.sum(H,axis=1)
        Cost=(np.sum(np.power((H-y_train),2))+lambda_*np.sum(np.power(Thetas[1:],2)))/(2*m)
        return Cost
```

here,

x_train = features in training set. **y-train =** prediction label in training set.

Lambda_ = regularization parameter. **m** = the length of training set.

➢ Gradient Descent:

Gradient Descent is used to find the minimum values of thetas, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Regularization is also applied with Gradient Descent to prevent overfitting.

Gradient Descent is given by:

```
def Gradient_Descent(x_train, y_train, Thetas,lambda_, alpha, iterations):
    J_train = []  #cost of training set in each iterations
    J_valid = []  #cost of valid set in each iterations
    J_test =  []  #cost of test set in each iterations
    temp_var = 0
    while temp_var < iterations:
        H = Hypothesis(Thetas, x_train)
        H = np.sum(H, axis=1)
        for  i in range(0, len(x_train.columns)):
            if i==0:
                Thetas[0]=Thetas[0]-alpha*(sum((H-y_train)*x_train.iloc[:,0])/(m))
            else:
                Thetas[i] = Thetas[i]*(1-alpha*(lambda_/m)) - alpha*(sum((H-y_train)*x_train.iloc[:,i])/(m))
        j_t = Cost_Function(x_train,y_train, Thetas,lambda_)
        J_train.append(j_t)
        j_v = Cost_Function_valid(x_valid,y_valid, Thetas,lambda_)
        J_valid.append(j_v)
        j_te = Cost_Function_valid(x_test,y_test, Thetas,lambda_)
        J_test.append(j_te)
        temp_var += 1
    return J_train,J_valid,J_test, j_t,j_v,j_te, Thetas
```

Here,

alpha = learning rate, **iterations=** iterations for the loop to run, **Thetas** = Model Parameter after gradient descent.

j_t, j_v,j_te are the cost(error) after each iteration in the training, valid and test data.

4. Output of the Model:

The model parameters after training for both datasets USA and Worldwide are given below:

➢ USA Dataset Model parameters after training:

The model parameters of USA data set are given below:

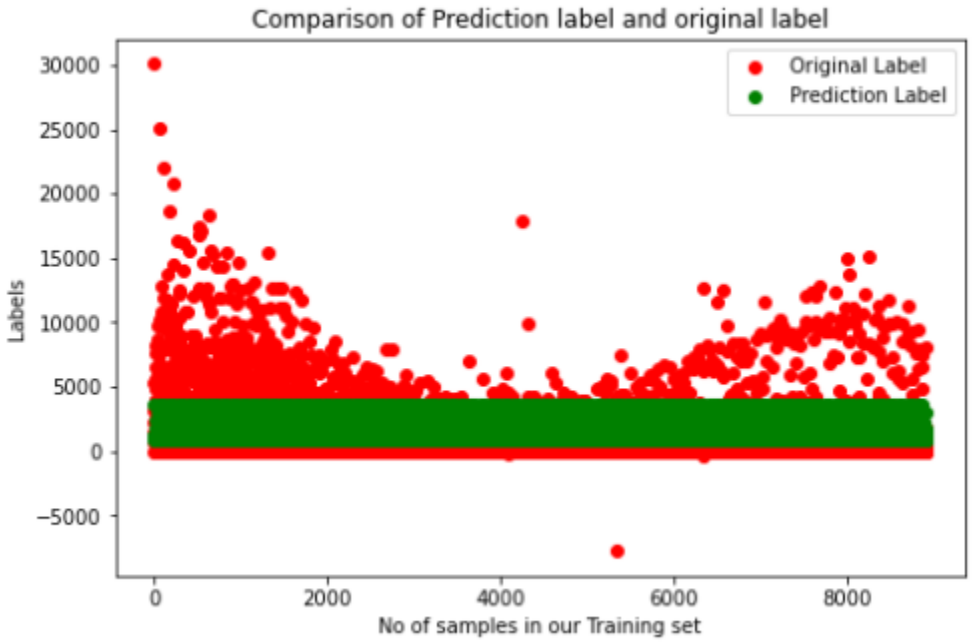
Thetas = array([598, 0, 2, 24, 350, 2864, 56, 0])

As, in USA data set we have 8 features including the bias term, therefore we got 8 thetas after optimization.

- **`py_train`** is our prediction and we are applying our model parameters on training set to get the predictions.

py_train= Hypothesis(Thetas,x_train)

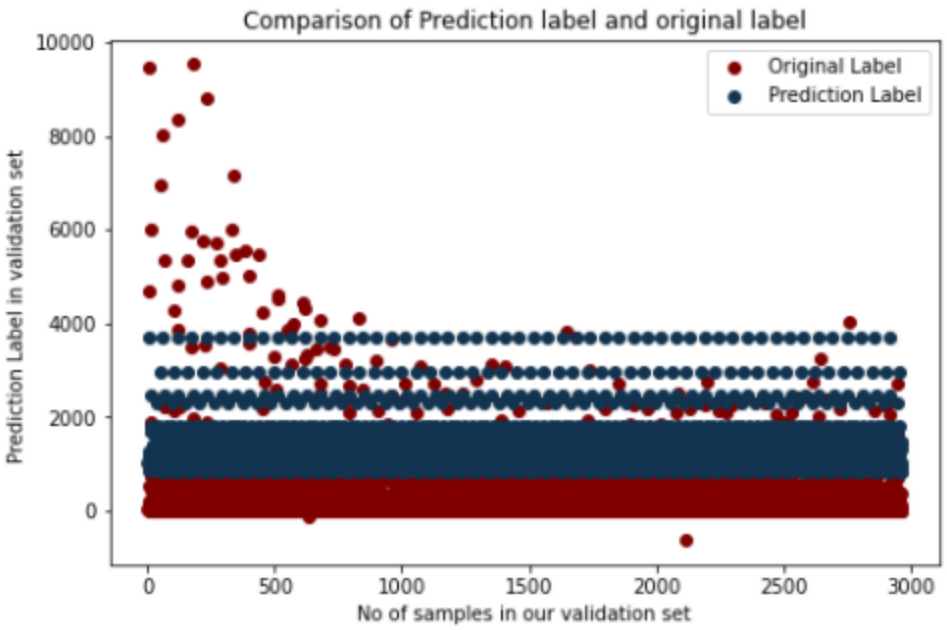
py_train= np.sum(py_train,axis=1)



- **`py_validation`** is our prediction on validation set and we are applying our model parameters on validation set to check the predictions and accuracy of our model.

py_validation= Hypothesis (Thetas,x_valid)

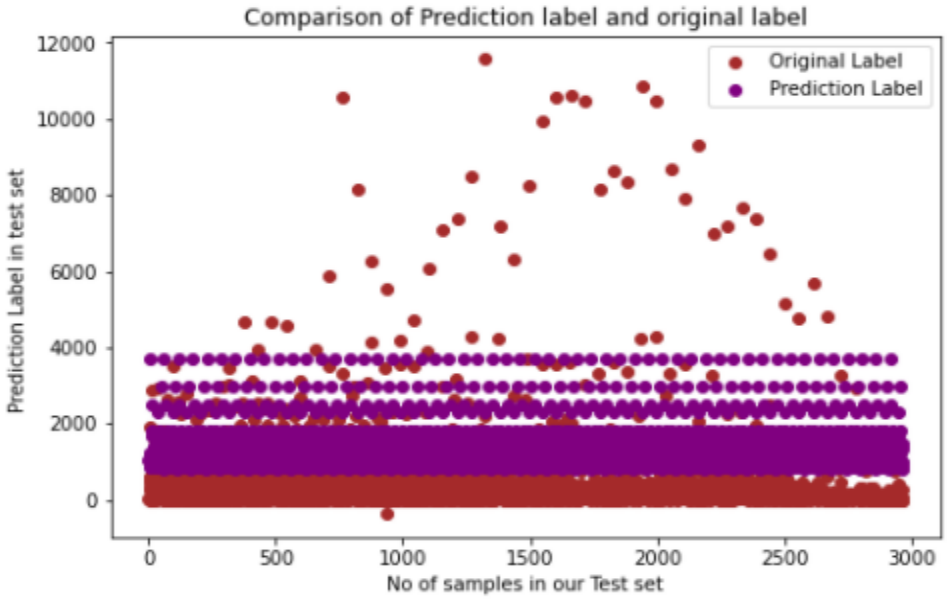
py_validation= np.sum(py_validation,axis=1)



- **`py_test`** is our prediction on test set and we are applying our model parameters on test set to check the predictions and accuracy of our model.

py_test= Hypothesis(Thetas,x_test)

py_test= np.sum(py_test,axis=1)



➢ Worldwide Dataset Model parameters after training:

The model parameters of Worldwide data set are given below:

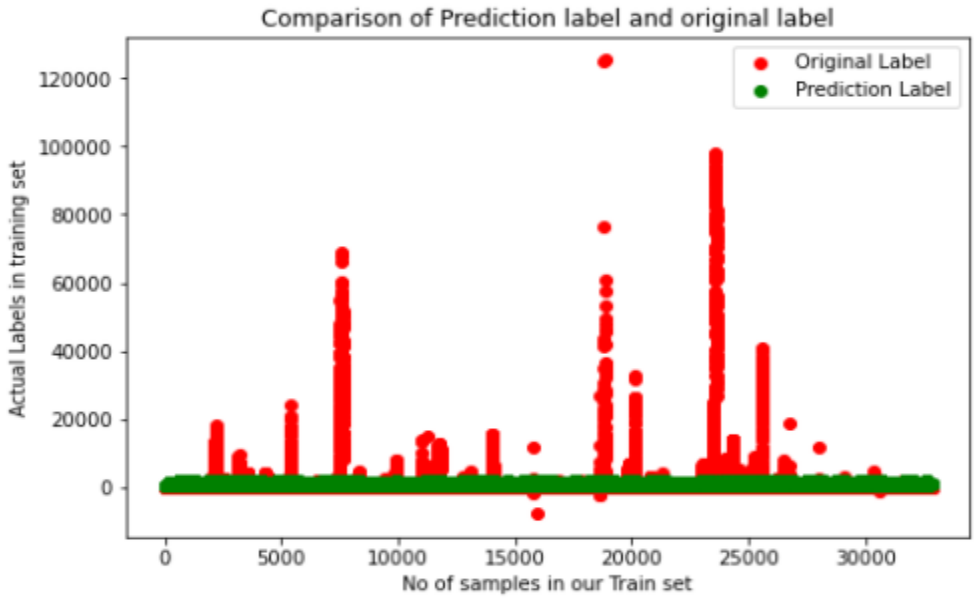
Thetas= array([137, 1410, 84, 0, 322, 0])

As, in Worldwide data set we have 6 features including the bias term, therefore we got 6 thetas after optimization.

- **`py_train`** is our prediction and we are applying our model parameters on training set to get the predictions.

py_train= Hypothesis(Thetas,x_train)

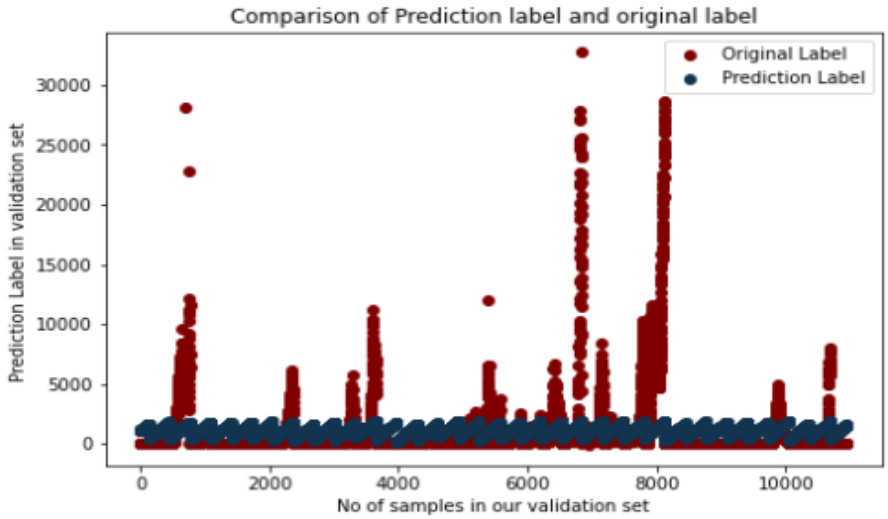
py_train= np.sum(py_train,axis=1)



- **`py_validation`** is our prediction on validation set and we are applying our model parameters on validation set to check the predictions and accuracy of our model.

py_validation= Hypothesis (Thetas,x_valid)

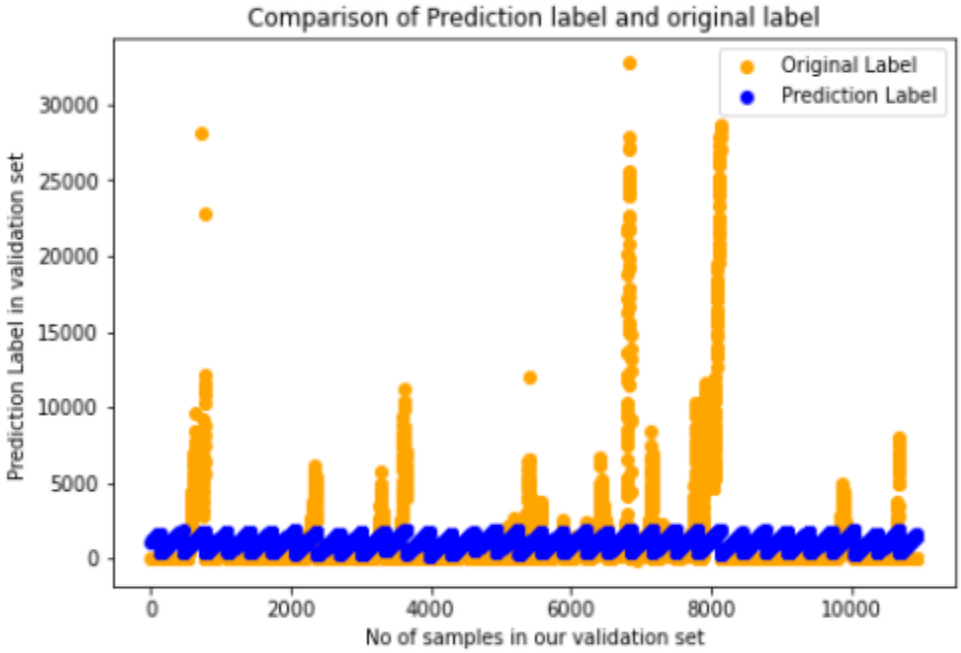
py_validation= np.sum(py_validation,axis=1)



- **`py_test`** is our prediction on test set and we are applying our model parameters on test set to check the predictions and accuracy of our model.

py_test= Hypothesis(Thetas,x_test)

py_test= np.sum(py_test,axis=1)



5. Model Training Details:

➢ USA Dataset Model training details:

For USA Data set training data is composed of **8904** rows and **8** columns.

- alpha is learning rate = **0.01**
- Iterations is our loop running factor = **2000**
- lambda_ is regularization parameter **= 10**

➢ Worldwide Dataset Model training details:

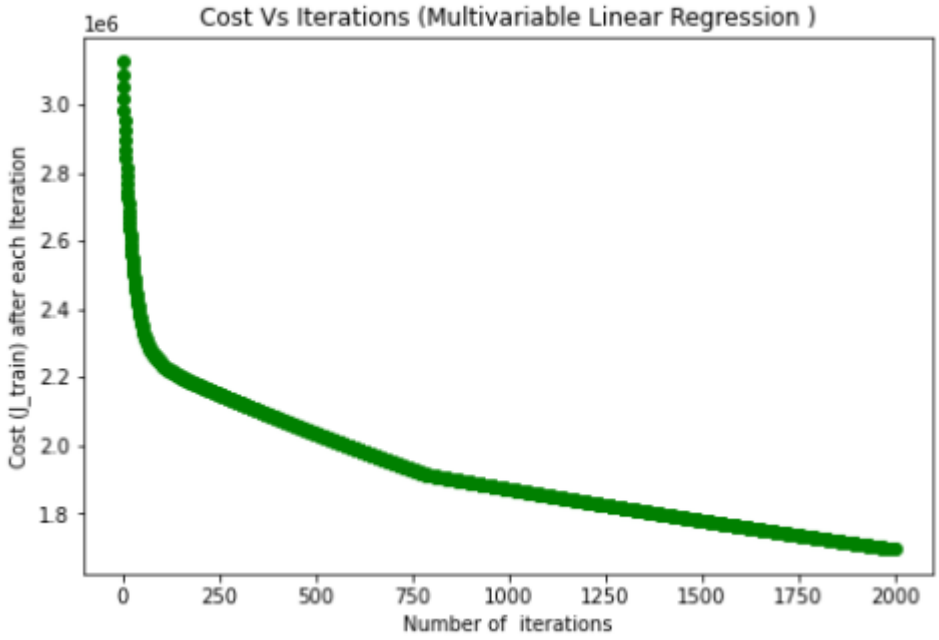
For Worldwide Data set training data is composed of **32832** rows and **6** columns.

- alpha is learning rate = **0.01**
- Iterations is our loop running factor = **1000**
- lambda_ is regularization parameter **= 10**

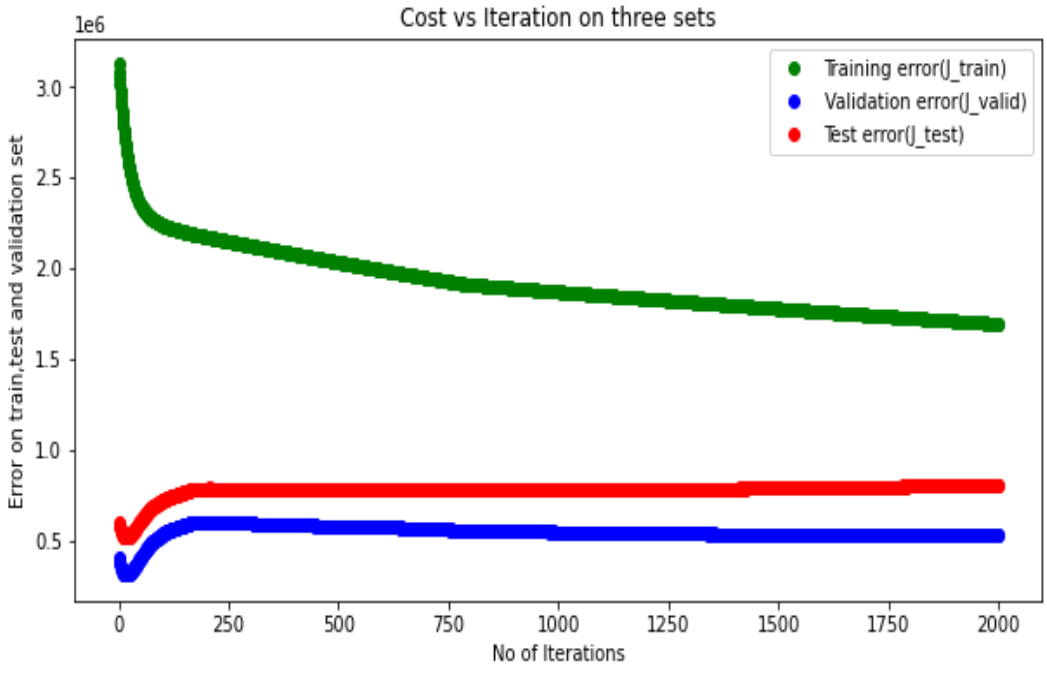
6. Plots:

➢ USA Dataset Plots:

The **training loss** plot with respect to iterations for USA data training set is given below:



Error Plot for all three data splits of USA Data set is given below:



- **Mean Absolute Error** for training set is given by:

MAE_train=np.sum(np.absolute(y_train-py_train))/len(x_train)

MAE_train= 1087.1478347699021

Mean absolute Error is `1087` for training set. It means that model is predicting `1087` values different from actual cases

- **Mean Absolute Error** for validation set is given by:

MAE_validation=np.sum(np.absolute(y_valid-py_validation))/len(x_valid)

MAE_validation= 932.9972011398406

Mean absolute Error is `932` for validation set. It means that model is predicting `932` values wrong from actual cases

- **Mean Absolute Error** for test set is given by:

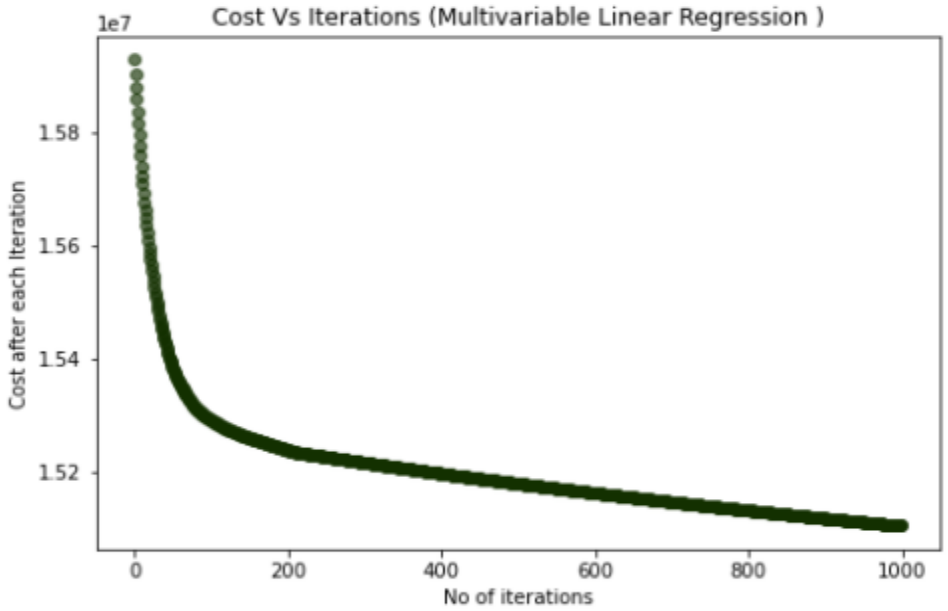
MAE_test=np.sum(np.absolute(y_test-py_test))/len(x_test)

MAE_test = 1068.2771819330865

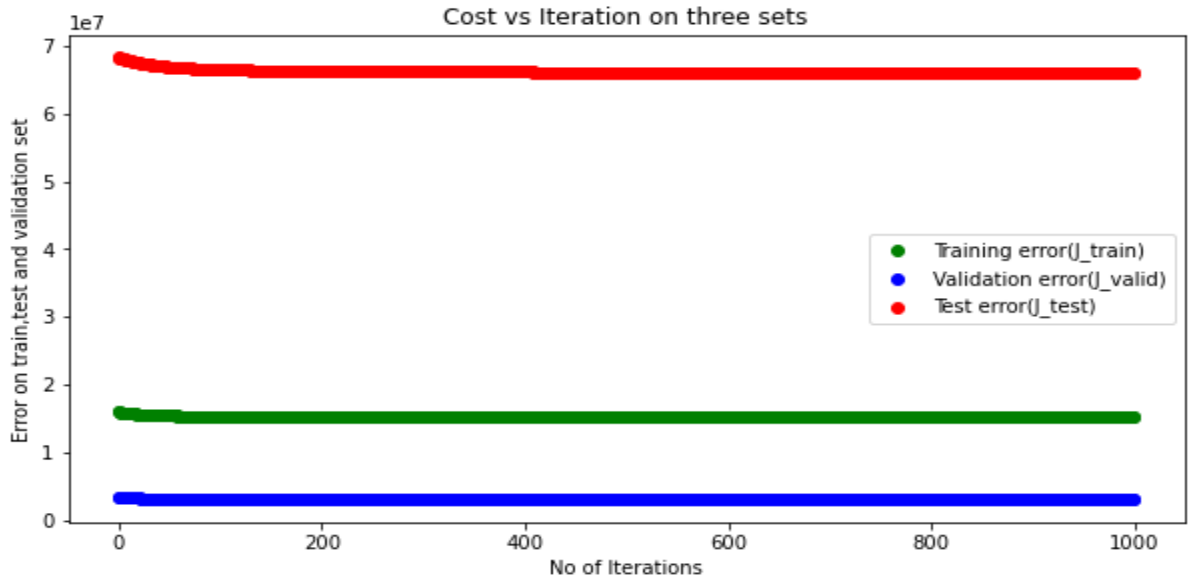
Mean absolute Error is `1068.34` for test set. It means that model is predicting **`1068`** values wrong from actual cases in test set.

➢ Worldwide Dataset Plots:

The **training loss** plot with respect to iterations for worldwide data training set is given below:



Error Plot for all three data splits of USA Data set is given below:



- **Mean Absolute Error** for training set is given by:

MAE_train=np.sum(np.absolute(y_train-py_train))/len(x_train)

MAE_train= 1599.0524808782307

Mean absolute Error is `1599` for training set. It means that model is predicting `1599` values different from actual cases

- **Mean Absolute Error** for validation set is given by:

MAE_validation=np.sum(np.absolute(y_valid-py_validation))/len(x_valid)

MAE_validation= 1285

Mean absolute Error is `1285` for validation set. It means that model is predicting **`1285`** values wrong from actual cases

- **Mean Absolute Error** for test set is given by:

MAE_test=np.sum(np.absolute(y_test-py_test))/len(x_test)

MAE_test = 2638.1314938164996

Mean absolute Error is `2638` for test set. It means that model is predicting **`2638`** values wrong from actual cases in test set.

7. Complete Codes:

Annex A

➢ Instructions on running the code:

Four Complete notebooks with all the optimal parameters are provided with the data for both USA Prediction Model and Worldwide Model.

1) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in USA."**
2) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in World.**
3) **Prediction Code for USA.**
4) **Prediction Code for World.**

For the first two (1 and 2) above mentioned books:

- If user only wants to see the already trained books for both Datasets. Just open both notebooks named:

1) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in USA."**
   This book is composed of USA prediction model, training code and all necessary steps.
   For good user experience, it is available in jupyter notebook format, pdf format as well as in HTML format.
2) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in World.**
   This jupyter notebook is composed of worldwide/country wise prediction model, training code and all necessary steps taken.
   For good user experience, it is available in jupyter notebook format, pdf format as well as in HTML format.

For the last two (3 and 4) above mentioned books:

3) Jupyter notebook named **"Prediction Code for USA**" is composed of Prediction code for USA data set with all the commands and instructions for the user. User have to just run the book and do as it says.

4) Jupyter Notebook named **"Prediction Code for Worldwide**" is composed of Prediction code for Worldwide/Country wise data set with all the commands and instructions for the user. User have to just run the book and do as it says.

Annex B

➢ Training Code with Optimal Parameters:

Two complete Trained notebooks named are in the folder of data:

1) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in USA."** is the name of notebook for USA model.

Training Code For USA Data set:

Initiating Thetas (Model Parameters) and defining Hypothesis for Linear Regression. As in this model Linear regression is applied, so Hypothesis is Thetas*x_train. Here:

* Thetas= Model Parameters or weights for each feature.
* x_train= Features of model

In this model, there are total 8 features including the bias term Xo. So initiating 8 thetas of value = 0.

```
Thetas=np.array([0]*len(x_train.columns))
Thetas
array([0, 0, 0, 0, 0, 0, 0, 0])
```

Hypothesis is given by following function:

```
def Hypothesis(Thetas,x_train):
    return Thetas*x_train
```

m is the length is our training set.

```
m = len(x_train)
```

Cost function

Cost function is basically the difference between prediction by the model and the prediction label. Here, Regularization is also applied with Cost Function

*lambda_ is the regulzarization parameter.

Cost function is given by:

```
def Cost_Function(x_train,y_train,Thetas,lambda_):
    H=Hypothesis(Thetas,x_train)
    H=np.sum(H,axis=1)
    Cost=(np.sum(np.power((H-y_train),2))+lambda_*np.sum(np.power(Thetas[1:],2)))/(2*m)
    return Cost
n=len(x_valid)
def Cost_Function_valid(x_valid,y_valid,Thetas,lambda_):
    H=Hypothesis_v(Thetas,x_valid)
    H=np.sum(H,axis=1)
```

```
    Cost=(np.sum(np.power((H-y_valid),2))+lambda_*np.sum(np.power(Thetas[1:],2)))/(2*n)
    return Cost
```

Gradient Descent

Gradient Descent is used to find the minimum values of thetas, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Here, Regularization is also applied with Gradient Descent to prevent overfitting

Gradient Descent is given by:

```
def Gradient_Descent(x_train, y_train, Thetas,lambda_, alpha, iterations):
  J_train = []  #cost of training set in each iterations is saved in this list
  J_valid = []  #cost of valid set in each iterations is saved in this list
  J_test =  []  #cost of test set in each iterations is saved in this list
  temp_var = 0
  while temp_var < iterations:
    H = Hypothesis(Thetas, x_train)
    H = np.sum(H, axis=1)
    for  i in range(0, len(x_train.columns)):
      if i==0:
        Thetas[0]=Thetas[0]-alpha*(sum((H-y_train)*x_train.iloc[:,0])/(m))
      else:
        Thetas[i] = Thetas[i]*(1-alpha*(lambda_/m)) - alpha*(sum((H-
y_train)*x_train.iloc[:,i])/(m))

        #thethas[i] = thethas[i] - alpha*(sum((H-y_train)*x_train.iloc[:,i])/(m))
    j_t = Cost_Function(x_train,y_train, Thetas,lambda_)
    J_train.append(j_t)
    j_v = Cost_Function_valid(x_valid,y_valid, Thetas,lambda_)
    J_valid.append(j_v)
    j_te = Cost_Function_valid(x_test,y_test, Thetas,lambda_)
    J_test.append(j_te)
    temp_var += 1
  return J_train,J_valid,J_test, j_t,j_v,j_te, Thetas
```

Calculating cost using gradient descent with the following parameters:

alpha is learning rate **=0.01**
Iterations is our loop running factor =**2000**
lambda_ is regularization = **10**

Here, when we put

*lambda_ = 0, there will no regularizarion

```
J_train,J_valid,J_test, j_t,j_v,j_te,Thetas=Gradient_Descent(x_train,y_train,Thetas,10,0.01,2000)
```

Thetas= array ([598, 0, 2, 24, 350, 2864, 56, 0])

2) **"Usman Zaheer_Notebook_Multivariate Linear Regression Model for Predicting Corona Cases in World.** Is the name of Worldwide prediction model.
    - In this notebook, complete training code for prediction of cases in USA is included along with the explanations and instructions.
    - PDF and HTML version of this notebook is also available in the folder.

Training Code For Worldwide Data set:

Initiating Thetas (Model Parameters) and defining Hypothesis for Linear Regression. As in this model Linear regression is applied, so Hypothesis is Thetas*x. Here:

Thetas= Model Parameters or weights for each feature.

x_train= Features of model.In this model, there are total 6 feaures including the bias term Xo. So initiating 6 thetas of initiation value = 0.

Thetas= np.array([0]*len(x_train.columns))

Thetas

array([0, 0, 0, 0, 0, 0])

Defining **Hypothesis:**

```
def Hypothesis(Thetas,x_train):
    return Thetas*x_train
```

m = length of training set

m =len(x_train)

Defining Cost function

Cost function is basically the difference between prediction by the model and the prediction label. Here, Regularization is also applied with Cost Function.

* lambda_ is the regulzarization parameter.

```
def Cost_Function(x_train,y_train,Thetas,lambda_):
    H=Hypothesis(Thetas,x_train)
    H=np.sum(H,axis=1)
    Cost=(np.sum(np.power((H-y_train),2))+lambda_*np.sum(np.power(Thetas[1:],2)))/(2*m)
    return Cost
```

n=len(x_valid)

```
def Cost_Function_valid(x_valid,y_valid,Thetas,lambda_):
    H=Hypothesis(Thetas,x_valid)
```

```
    H=np.sum(H,axis=1)
    Cost=(np.sum(np.power((H-y_valid),2))+lambda_*np.sum(np.power(Thetas[1:],2)))/(2*n)
    return Cost
```

Gradient Descent

Gradient Descent is used to find the minimum values of thetas, so that our cost will be minimum. Minimum cost indicates that the difference between our prediction and actual label is very low. Here, Regularization is also applied with Gradient Descent to prevent overfitting.

Defining **Gradient Descent**

```
def Gradient_Descent(x_train, y_train, Thetas,lambda_, alpha, iterations):
    J_train = []  #cost of training set in each iterations is saved in this list
    J_valid = []  #cost of valid set in each iterations is saved in this list
    J_test =  []  #cost of test set in each iterations is saved in this list
    temp_var = 0
    while temp_var < iterations:
        H = Hypothesis(Thetas, x_train)
        H = np.sum(H, axis=1)
        for  i in range(0, len(x_train.columns)):
            if i==0:
                Thetas[0]=Thetas[0]-alpha*(sum((H-y_train)*x_train.iloc[:,0])/(m))
            else:
                Thetas[i] = Thetas[i]*(1-alpha*(lambda_/m)) - alpha*(sum((H-y_train)*x_train.iloc[:,i])/(m))


                #thethas[i] = thethas[i] - alpha*(sum((H-y_train)*x_train.iloc[:,i])/(m))
        j_t = Cost_Function(x_train,y_train, Thetas,lambda_)
        J_train.append(j_t)
        j_v = Cost_Function_valid(x_valid,y_valid, Thetas,lambda_)
        J_valid.append(j_v)
        j_te = Cost_Function_valid(x_test,y_test, Thetas,lambda_)
        J_test.append(j_te)
```

temp_var += 1

return J_train,J_valid,J_test, j_t,j_v,j_te, Thetas

Calculating cost using gradient descent with the following parameters:

alpha is learning rate **=0.01**

Iterations=**1000**

lambda_ is regularization parameter= **10**

Here, when we put

* lambda_ = 0, there will no regularizarion

* Iterations = Number of iterations for the loop

J_train,J_valid,J_test, j_t,j_v,j_te, Thetas = Gradient_Descent(x_train,y_train,Thetas,10,0.01,1000)

Thetas (paramters) of our model on training data are as follows:

Thetas= array([137, 1410, 84, 0, 322, 0])

Annex C

➢ Prediction Code:

In the data folder two more notebooks are included. Users have to just run the books and enter the features as instructed by the program.

1) Prediction Code for USA:

```
import pandas as pd
import numpy as np
```

Instructions/Features Information:

Here, We have 8 valid features: ('USA Final.csv' file can be used for reference.)

- *1) XO (Bias Term) = 1
- *2) DATE_CODE = In this model "date" is numerically coded. For example:
  Model date starts from 6th December 2020 and ends at 17th March 2020.
  - *Here: 17th March 2020 is 264 and 7th December is 0.
- *3) States_Code: States codes are also numerically coded. For example:
  *Here: 1 = Alaska, 2= Alabama and 56=Wyoming.
- *4) Temperatures_(F): Tempearures in F of particular state at particular date.
- *5) Humidity_(%): Average humidity in % of particular state at particular date.
- *6) Population_per_state: Average populationin of particular state at particular date.
- *7) LandArea_(sq miles): Land Area in square miles of particular state .
- *8) POPULATION_DENSITY: Population Density of particular state.

The below code will ask user to enter features for this model as per his desire one by one when code is run.

Storing max and min values of each feature's column of worldwide data for normalization/scaling of user input features.

```
# Date_Code  max and min
D_max = 264
D_min= 0
# States_Code max and min
S_max= 56
S_min=1
# Temperatures_(F)  max and min
T_max= 82.80
T_min= 0
# Humidity_(%)   max and min
H_max= 80
H_min= 38.30
# Population_per_state max and min
P_max= 39512223.0
```

```
P_min= 55138.00
# LandArea_(sq miles) max and min
LA_max=  570641.0
LA_min = 68.00
# Population_Density max and min
PD_max= 11535.0
PD_min= 1.29

def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 12, 6)
   date1 = datetime.date(year,month,day)
   date_code = (date-date1)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return dayss
```

Getting 7 features as input from user as first feature is the bias term Xo=1

Getting date from User

```
year = int(input('Enter a year: '))
month = int(input('Enter a month: '))
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
Enter a year: 2020
Enter a month: 3
Enter a day: 17
Date_code is: 264
```

Normalized date code is: 1.0

Getting state code from User.

Note: For State codes, please look into "Master sheet for Reference" data file provided.

```
S = float(input('Enter state Code: '))
S = (S-S_min)/(S_max-S_min)
print("Normalized state code  is:", S)
#here for example, 1= Alaska
```

Enter state Code: 1

Normalized state code is: 0.0

Getting temperature in Farhenheit from User

```
T = float(input('Enter value of Temperature in F for particular state: '))
T = (T-T_min)/(T_max-T_min)
```

print("Normalized temperature is:", T)

Enter value of Temperature in F for particular state: 32

Normalized temperature is: 0.3864734299516908

Getting Humidity in % from User

H = float(input('Enter value of Humidity in % for particular state: '))

H = (H-H_min)/(H_max-H_min)

print("Normalized humidity is:", H)

Enter value of Humidity in % for particular state: 32

Normalized humidity is: -0.1510791366906474

Getting Population of state from User

P = float(input('Enter population of state: '))

P = (P-P_min)/(P_max-P_min)

print("Normalized population is:", P)

Enter population of state: 10000

Normalized population is: -0.0011439770576057507

Getting Landarea in square miles of state from User

LA = float(input('Enter land area in sq miles of state: '))

LA= (LA-LA_min)/(LA_max-LA_min)

print("Normalized land area is:", LA)

Enter land area in sq miles of state: 27373

Normalized land area is: 0.04785540149989572

Getting Population Density per square miles of state from User

P_max-P_min

PD = float(input('Enter population density of state: '))

PD= (PD-PD_min)/(P_max-P_min)

print("Normalized population density is:", PD)

Enter population density of state: 1000

Normalized population density is: 2.531129707123575e-05

Making array of user input features as per our model

User_input = np.array([1,D,S,T,H,P,LA,PD])

User_input

array([1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.86473430e-01,

-1.51079137e-01, -1.14397706e-03, 4.78554015e-02, 2.53112971e-05])

Model paramters of USA dataset from our trained model

Thetas =np.array([598, 0, 2, 24, 350, 2864, 56, 0])

def Prediction(Thetas,User_input):

```
    Cases =Thetas*User_input
    Cases= np.sum(Cases)
    return Cases
```

Output =Prediction(Thetas,User_input)

```
print("Number of Corona Cases predicted by model are:", Output)
```

Number of Corona Cases predicted by model are: 553.8012166681253

Enter actual Cases on the selected day for selected state

```
actual_cases = int(input('Enter Number of Actual Cases: '))

actual_cases
```

Enter Number of Actual Cases: 200

Calculating Mean Absolute Error

```
MAE_user =np.sum(np.absolute(actual_cases-Output))
MAE_user= 353.80121666812533
```

MAE_user will give the difference between actual cases and model predicted cases on the selected day for the selectd state of USA.

2) Prediction Code for Worldwide/Country Wise:

Instructions/Features Information

Here, We have 6 features: ('Worldwide Final.csv' file is used for reference.

*1) Xo (Bias Term) = 1
*2) DATE_CODE = In this model "date" is numerically coded. For example:
Model date starts from 23 January 2020 and ends at 7th December 2020.
*Here: 23/01/2020 is 0 and 7th December is 319.
*3) COUNTRY_CODE: Country codes are also numerically coded. For example:
*Here: 0 = Afghanistan, 2= Albania and 170=Zimbabwe.
*4) HUMAN_DEVELOPMENT_INDEX: Average Human development index of particular state
*5) POPULATION_DENSITY: Population Density of particular country.
*6) TEMPERATURES: Tempearures in C of particular country at particular date.

The below code will ask user to enter features for this model as per his desire one by one when code is run.
Storing max and min values of each feature's column of worldwide data for normalization/scaling of user input features.

```
# DATE_CODE max and min
D_max = 319
D_min= 0
# COUNTRY_CODE max and min
```

```
C_max= 170
C_min=0
# POPULATION_DENSITY max and min
P_max= 19347.5
P_min = 1.98
# HUMAN_DEVELOPMENT_INDEX max and min
HDI_max= 0.953
HDI_min= 0.354
 # TEMPERATURES  max and min
T_max= 52.0
T_min= -16
```

Creating function for numerical coding of date:

```
def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 1, 23) #fixing the date of my model as reference
   date1 = datetime.date(year,month,day)
   date_code = (date1 - date)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return days

def Date_numerical_coding(year,month,day):
   import datetime
   date = datetime.date(2020, 1, 23) #fixing the date of my model as reference
   date1 = datetime.date(year,month,day)
   date_code = (date1 - date)
   dayss = date_code.days
   print ('Date_code is:'  ,dayss)
   return dayss
Getting date from User
year = int(input('Enter a year: '))
month = int(input('Enter a month: '))
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
year = int(input('Enter a year: '))
month = int(input('Enter a month: '))
day = int(input('Enter a day: '))
D= Date_numerical_coding(year,month,day)
D = (D-D_min)/(D_max-D_min)
print("Normalized date code is:", D)
```

Enter a year: 2020
Enter a month: 12
Enter a day: 7
Date_code is: 319
Normalized date code is: 1.0

Getting country code from User.

Note: For Country codes, please look into "Master sheet for Reference" data file provided.

```
C = float(input('Enter Country Code: '))
C = (C-C_min)/(C_max-C_min)
print("Normalized country  is:", C)
#here for example, 35= China
Enter Country Code: 34
Normalized country  is: 0.2
```

Getting Population density from User

```
P = float(input('Enter population Density of selected country: '))
P = (P-P_min)/(P_max-P_min)
print("Normalized population density  is:", P)
```

Enter population Density of selected country: 45
Normalized population density is: 0.0022237706714526156

Getting Human Development index from User

```
HDI = float(input('Enter value of Human Devlopment Index of selected country: '))
HDI = (HDI-HDI_min)/(HDI_max-HDI_min)
print("Normalized human development index is:", HDI)
```

Enter value of Human Devlopment Index of selected country: 0.4
Normalized human development index is: 0.0767946577629383

Getting temperature in Centigrade from User

```
T = float(input('Enter value of Temperature in C for particular country: '))
T = (T-T_min)/(T_max-T_min)
print("Normalized temperature  is:", T)
```

Enter value of Temperature in C for particular country: -16
Normalized temperature is: 0.0

 # Making array of user input features as per our model

```
User_input = np.array([1,D,C,P,HDI,T])
User_input
array([1.       , 0.9968652 , 0.20588235, 0.00222377, 0.07679466, 0.       ])
```

Model paramters of Worldwide dataset from our trained model

```
Thetas = np.array([137, 1410,   84,    0,  322,    0])
Thetas = array([ 137, 1410,   84,    0,  322,    0])

def Prediction(Thetas,User_input):
 Cases =Thetas*User_input
Cases= np.sum(Cases)
 return Cases
```

Output =Prediction(Thetas,User_input)
print("Number of Corona Cases predicted by model are:", Output)

Number of Corona Cases predicted by model are: 1584.6019347508002

Enter actual Cases on the selected day for selected country

actual_cases = int(input('Enter Number of Actual Cases: '))
actual_cases

Enter Number of Actual Cases: 2000

Calculating Mean Absolute Error

MAE_user =np.sum(np.absolute(actual_cases-Output))
MAE_user = 415.3980652491998

`MAE_user` will give the difference between actual cases and model predicted cases on the selected day for the selected country.