WORLD-WIDE COVID CASES PREDICTION

Problem Statement: *"Complete code to train a non-linear model for predicting corona cases in different countries in the world using regularization."*

DATA SET DETAILS:

Gathering & Cleaning:

- Main Data file can be excessed as attached *"Country wise Covid Data.xlsx".*
- CSV files downloaded from https://covidtracking.com.
- CSV file contained large amount of data for each state. Till 06-Dec-2020. Covid cases and dates.
- Temperature & Humidity data collected from www.accuweather.com. Also for temperature data a separate folder "Temperature" can be used with detailed country wise temperature data.
- Human Development Index and population density data for each country is collected from available data on www.google.com
- Above excel file is maintained and edited so that each country has been allotted a numeric value ranging from 1 to 170.
- All dates has been ranged from 23-Jan-2020 to 07-Dec-2020. By taking 23-Jan-2020 as 1.

```
#Importing covid data excell file using pandas
Data = pd.read_excel("Country wise Covid Data.xlsx")
```

```
#printing data
Data
```

	CONTINENT	LOCATION	DATE	COUNTRY_SERIES	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES	CA
0	Asia	Afghanistan	2020-01-23	0	0	0	54.422	0.498	7.0	
1	Asia	Afghanistan	2020-01-24	1	1	0	54.422	0.498	7.0	
2	Asia	Afghanistan	2020-01-25	2	2	0	54.422	0.498	13.0	
3	Asia	Afghanistan	2020-01-26	3	3	0	54.422	0.498	2.0	
4	Asia	Afghanistan	2020-01-27	4	4	0	54.422	0.498	6.0	
...	...	...	...	...	...	...	...	...	...	
54715	Africa	Zimbabwe	2020-12-03	54715	315	170	42.729	0.535	31.0	
54716	Africa	Zimbabwe	2020-12-04	54716	316	170	42.729	0.535	26.0	
54717	Africa	Zimbabwe	2020-12-05	54717	317	170	42.729	0.535	21.0	
54718	Africa	Zimbabwe	2020-12-06	54718	318	170	42.729	0.535	22.0	
54719	Africa	Zimbabwe	2020-12-07	54719	319	170	42.729	0.535	22.0	

54720 rows × 10 columns

FEATURES:

1. Xo (All entries as 1)

```
#Add a column of ones for the bias term. I chose 1 because if you multiply one with any value, that value does not change.
Data = pd.concat([pd.Series(1,index=Data.index,name = '00'),Data],axis = 1)
Data.head()
```

	00	CONTINENT	LOCATION	DATE	COUNTRY_SERIES	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES
0	1	Asia	Afghanistan	2020-01-23	0	0	0	54.422	0.498	7.0
1	1	Asia	Afghanistan	2020-01-24	1	1	0	54.422	0.498	7.0
2	1	Asia	Afghanistan	2020-01-25	2	2	0	54.422	0.498	13.0
3	1	Asia	Afghanistan	2020-01-26	3	3	0	54.422	0.498	2.0
4	1	Asia	Afghanistan	2020-01-27	4	4	0	54.422	0.498	6.0

2. Date_Code
3. States_Code
4. Temperatures
5. Humidity
6. Population_Density

Additional features to make our model nonlinear are as under:

7. Temperature Square
8. HDI * Population Density
9. Population Density Square

```
#Head function gives top 5 entries of complete data set
X.head()
```

	00	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES	HDI	Temperature_sq	Population_Density_sq
26058	1	138	81	347.778	0.909	28.0	316.130202	784.0	1.209495e+05
51141	1	261	159	104.914	0.791	22.0	82.986974	484.0	1.100695e+04
33948	1	28	106	19347.500	0.500	15.0	9673.750000	225.0	3.743258e+08
43031	1	151	134	82.328	0.505	30.0	41.575640	900.0	6.777900e+03
42371	1	131	132	556.667	0.500	25.0	278.333500	625.0	3.098781e+05

SCALING OF DATA:

Data has been scaled by dividing each element in a column by maximum value in that column. This ranges data between 0 and 1.

SCALING OF DATA

```
#np.spacing(0) to avoid division 0/0.
x = X / (X.max(axis=0) + np.spacing(0))
x.head()
```

	00	DATE_CODE	COUNTRY_CODE	POPULATION DENSITY	HUMAN_DEVELOPMENT_INDEX	TEMPERATURES	HDI	Temperature_sq	Population_Density_sq
26058	1.0	0.432602	0.476471	0.017975	0.953830	0.538462	0.032679	0.289941	0.000323
51141	1.0	0.818182	0.935294	0.005423	0.830010	0.423077	0.008579	0.178994	0.000029
33948	1.0	0.087774	0.623529	1.000000	0.524659	0.288462	1.000000	0.083210	1.000000
43031	1.0	0.473354	0.788235	0.004255	0.529906	0.576923	0.004298	0.332840	0.000018
42371	1.0	0.410658	0.776471	0.028772	0.524659	0.480769	0.028772	0.231139	0.000828

SPLITTING OF DATA SET:

Data set has been splitted into 03 Data sets.

1. Training Data (60%)
2. Cross Validation Data (20%)
3. Testing Data (20%)

SPLITTING OF DATA SET INTO TRAIN,TEST & VALID DATA

```
data_train = round(0.6*len(Data))
data_valid = round(data_train+0.2*len(Data))
```

```
train_x = x[:data_train]
valid_x = x[data_train:data_valid]
test_x = x[data_valid:]
```

```
train_y = Y[:data_train]
valid_y = Y[data_train:data_valid]
test_y = Y[data_valid:]
```

MATHEMATICAL MODELING:

Gradient Descent Algorithm with Non-linear parameters.

First we have to define an array of Thetas equal to number of features.

Then hypothesis function is defined which takes input (Thetas and Training x parameters) and return predicted y.

Cost function gives the total cost of the training model.

Finally using Gradient descent algorithm thetas can be found and then these thetas are used in Cost Function and prediction function.

MATHEMATICAL MODEL:

```
theeta = np.array([0]*len(train_x.columns))
```

```
theeta
```

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

```
def hypothesis (theeta,train_x):    #Defining Hypothesis function
    h = theeta*train_x
    return h
```

```
def Cost_function (train_x,train_y,theeta,lamda):    # Cost function with regularisation.
    y_1 = hypothesis(theeta,train_x)
    y_1 = np.sum(y_1,axis=1)
    var2 = (lamda/(2*len(train_x)))*np.sum((theeta[1:]**2))    # var2 uses new variable lamda to regularise theeta.
    var1 = np.sum((y_1-train_y)**2)/(2*len(train_x))
    cost = var1 + var2
    return cost

# Reference Slide 09, Lecture 05
```

```
def Gradient_Descent(train_x, train_y, theeta, alpha, i,lamda):
    J = []
    J_cv = []
    for iterator in range (0,i):
        y_1 = hypothesis(theeta, train_x)
        y_1 = np.sum(y_1, axis=1)

#Gradient descent algorithm to find values of theeta (theeta(0)...theeta(6)). Regularisation term is added for all value
# of theetas except theeta(0)..... reference Slide 12, lecture 05

        for  c in range(0, len(train_x.columns)):
            if c == 0:
                theeta[c] = theeta[c] - alpha*(sum((y_1-train_y)*train_x.iloc[:,c])/len(train_x))
            else:
                theeta[c] = (theeta[c]*(1-(alpha*(lamda/len(train_x))))) - alpha*(sum((y_1-train_y)*train_x.iloc[:,c])/len(train_
        j = Cost_function(train_x, train_y, theeta,lamda)
        J.append(j)   #Storing value of J for each theeta
        j_cv = Cost_function(valid_x, valid_y, theeta,lamda)
        J_cv.append(j_cv)
    return J, j, theeta, J_cv,j_cv
```

REGULARIZATION:

Regularization is introduced in Cost function and Gradient descent Algorithm to reduce features and to avoid over fitting and under fitting of our data.

```
        theeta[c] = (theeta[c]*(1-(alpha*(lamda/len(train_x))))) - alpha*(sum((y_1-train_y)*train_x.iloc[:,c])/len(train_x))
j = Cost_function(train_x, train_y, theeta,lamda)
J.append(j)   #Storing value of J for each theeta
j_cv = Cost_function(valid_x, valid_y, theeta,lamda)
J_cv.append(j_cv)
```

OUTPUT OF MODEL:

OUTPUT OF THE MODEL

```
J, j, theeta,J_cv,j_cv= Gradient_Descent(train_x,train_y,theeta,0.01,2000,1000)
```

```
theeta
```

```
array([   0, 1509,  263,    0,  538, -102,    0,    0,    0])
```

Using Hypothesis function on Testing Data and Thetas we get the predicted values.

PREDICTION OF CASES ON TEST DATA

```
y_1 = hypothesis(theeta, test_x)
y_1 = np.sum(y_1, axis=1)
y_1
```

```
53182     806.863558
18738    1379.578540
31703     603.959165
37191     731.303188
31906    1568.154969
            ...
24572    1629.652000
45925    1460.520000
14624    1614.449370
30610    1587.924818
33715    1064.341948
```

Using Mean Square error and Root Mean Square Error the error in the model is given below:

ERROR ON TEST DATA SET

```
def RMSE(y_1, test_y):
    return np.sqrt((y_1 - test_y) ** 2).mean()
rmse_val = RMSE(np.array(y_1), np.array(test_y))
print(f" Root Mean Square Error is:  {rmse_val}")
```

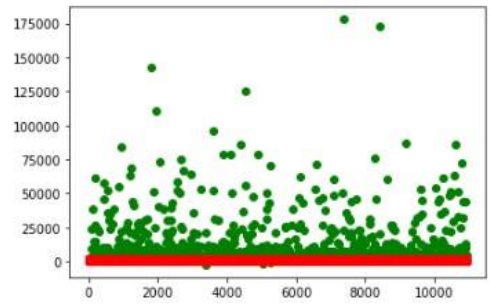
```
 Root Mean Square Error is:  1832.7631256766722
```

```
def MSE(y_1, test_y):
    return (1/2*len(test_x))*np.sum((y_1 - test_y) ** 2)
    #return np.sum((y_1 - test_y) ** 2).mean()
mse_val = MSE(np.array(y_1), np.array(test_y))
print(f" Mean Square Error is:  {mse_val}")
```

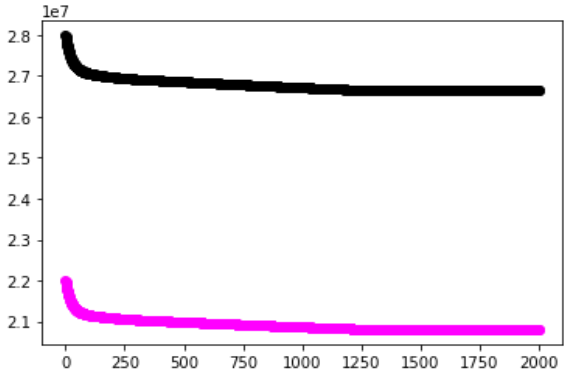
```
 Mean Square Error is:  2289129272281231.5
```

PLOTTING ACTUAL vs PREDICTED CASES

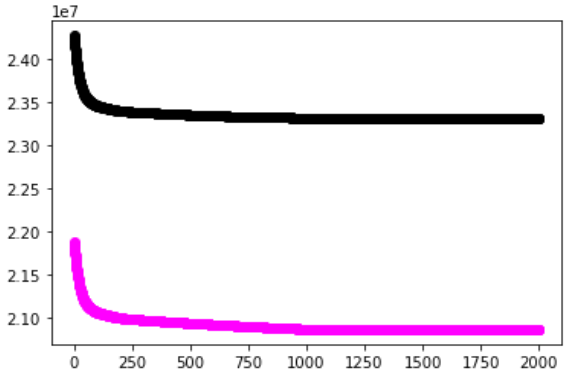
```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(x=list(range(0, len(test_x))),y= test_y, color='green')
plt.scatter(x=list(range(0, len(test_x))), y=y_1, color='red')
plt.show()
```



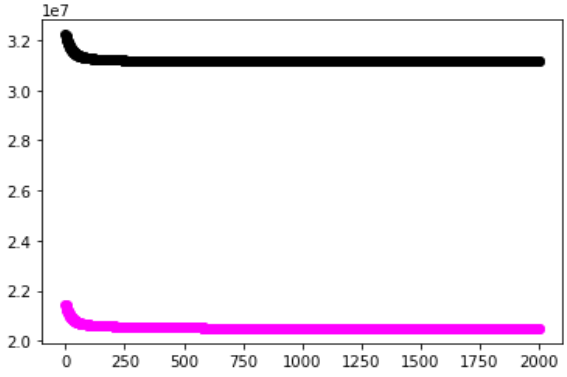
PLOTTING J AND J_CV ON DIFFERENT VALUES OF LAMBDA:



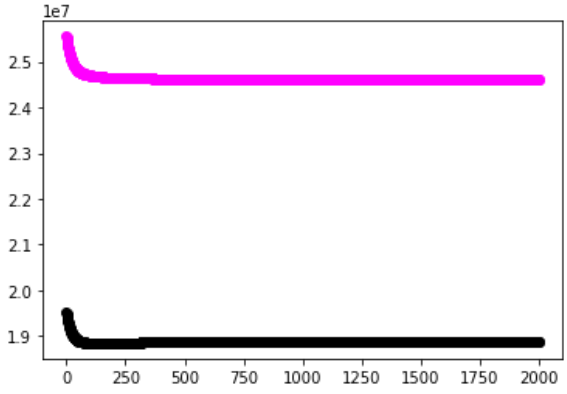
Lambda = 10



Lambda=1000



Lambda = 2000



Lambda = 5000

CONCLUSION:

Error function plot shows that by giving different values of Lambda the data becomes under fit or over fit. Because the features are selected based on the available data. But to predict actual Covid -19 cases we will have to get more features in detail. Like health conditions, age of infected patients, sex of infected patients, lockdown conditions, hospital data etc.

ANNEX A: Instructions on running the code (Readme File)

'Readme File (Instructions for Prediction Code)'

ANNEX B: Training Code (Python Notebook)

'World wise Cases with Regularization Machine Learning Project (Faisal Javed)'

ANNEX C: Prediction Code (Python Notebook)

'Prediction Function for World wise Cases with Regularization Machine Learning Project (Faisal Javed)'