USA COVID CASES PREDICTION

Problem Statement: *"Complete code to train a non-linear model for predicting corona cases in USA States using regularization."*

DATA SET DETAILS:

Gathering & Cleaning:

- Main Data file can be excessed as attached *"USA State 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.
- Landscape area, population density collected from statewide USA data available on www.google.com
- Above excel file is maintained and edited so that each state has been allotted a numeric value ranging from 1 to 56.
- All dates has been ranged from 06-Dec-2020 to 1-Jan-2020. By taking 06-Dec-2020 as 1.

DATA SET DETAILS

GATHERING & CLEANING

```
Data = pd.read_excel("USA Statewise Covid Data.xlsx")
Data.head()
```

	Date	Xo	Date_Code	States_Code	Temperatures	Humidity	LandArea	Population_Density	Cases_per_day
0	2020-12-06	1	0	1	11.6	77.1	570641	1.2863	757
1	2020-12-06	1	0	2	46.6	71.6	50645	96.9221	2288
2	2020-12-06	1	0	3	41.3	70.9	52035	58.403	1542
3	2020-12-06	1	0	4	38.0	80.0	77	716	0
4	2020-12-06	1	0	5	43.6	38.5	113594	64.9549	5376

FEATURES:

1. Xo (All entries as 1)
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. Temperature * Humidity
9. Humidity Square

10. Population Density Square

	Xo	Date_Code	States_Code	Temperatures	Humidity	Population_Density	Weather	Temperature_sq	Humidity_sq	Population_Density_sq
8985	1	160	26	64.6	70.4	71.5922	4547.84	4173.16	4956.16	5125.443101
5351	1	95	32	57.1	70.9	11.0393	4048.39	3260.41	5026.81	121.866144
13244	1	236	29	63.6	73.6	63.7056	4680.96	4044.96	5416.96	4058.403471
9412	1	168	5	76.0	38.5	64.9549	2926.00	5776.00	1482.25	4219.139034
9036	1	161	21	79.8	74.0	107.5174	5905.20	6368.04	5476.00	11559.991303
...	...	...	...	...	...	...	...	...	...	...
14643	1	261	28	0.0	70.0	324.0000	0.00	0.00	4900.00	104976.000000
2	1	0	3	41.3	70.9	58.4030	2928.17	1705.69	5026.81	3410.910409
3045	1	54	22	50.5	71.1	894.4359	3590.55	2550.25	5055.21	800015.579209
4763	1	85	4	0.0	80.0	716.0000	0.00	0.00	6400.00	512656.000000
5063	1	90	24	57.1	71.7	43.6336	4094.07	3260.41	5140.89	1903.891049

14840 rows × 10 columns

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

```
x = X/(X.max(axis=0) + np.spacing(0))
```

```
x.head()
```

	Xo	Date_Code	States_Code	Temperatures	Humidity	Population_Density	Weather	Temperature_sq	Humidity_sq	Population_Density_sq
8985	1.0	0.606061	0.464286	0.780193	0.88000	0.006207	0.745358	0.608701	0.774400	3.852088e-05
5351	1.0	0.359848	0.571429	0.689614	0.88625	0.000957	0.663502	0.475567	0.785439	9.158996e-07
13244	1.0	0.893939	0.517857	0.768116	0.92000	0.005523	0.767176	0.590002	0.846400	3.050142e-05
9412	1.0	0.636364	0.089286	0.917874	0.48125	0.005631	0.479550	0.842493	0.231602	3.170944e-05
9036	1.0	0.609848	0.375000	0.963768	0.92500	0.009321	0.967820	0.928849	0.855625	8.688050e-05

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

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, 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
```

```python
def Gradient_Descent(train_x, train_y, theeta, alpha, i,lamda):
    J = []  # Initial value of Cost (J) is empty.
    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.

```python
        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)
```

OUTPUT OF MODEL:

OUTPUT OF THE MODEL

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

```python
theeta
```

```
array([ 365, -115,   82,  156,  201,    0,  135,  100,  154,    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
```

```
418      872.379424
824      867.149698
11394    885.628456
3131     911.057620
9125     963.818353
            ...
14643    586.088068
2        835.964990
3045     885.757246
4763     688.830628
5063     910.507069
```

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}")
```

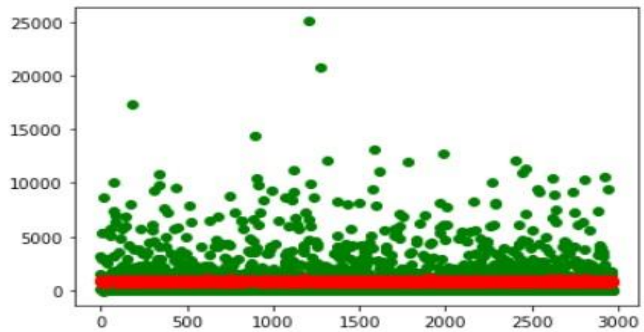
```
 Root Mean Square Error is:  972.2698140864471
```

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

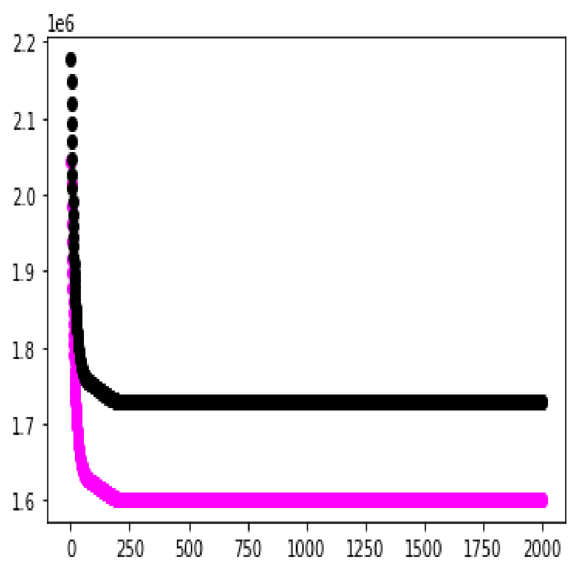
```
 Mean Square Error is:  3197262.1315421914
```

PLOTTING ACTUAL vs PREDICTED CASES

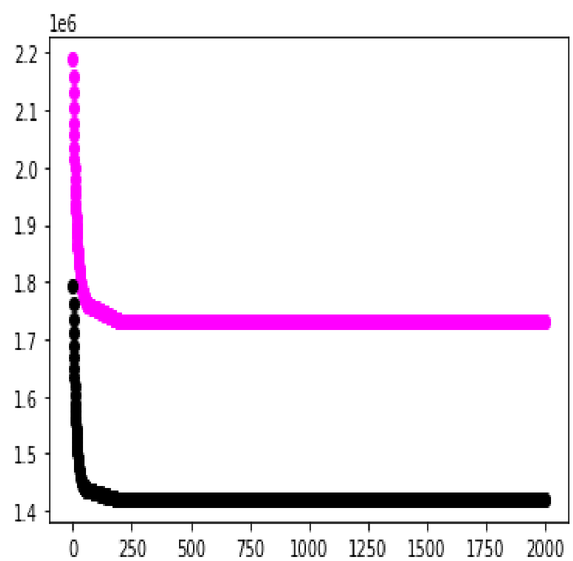
```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(x=list(range(0, len(test_x))),y= test_y, color='green')
plt.scatter(x=list(range(0, len(test_x))), y=y_1, color='red')
plt.show()
```



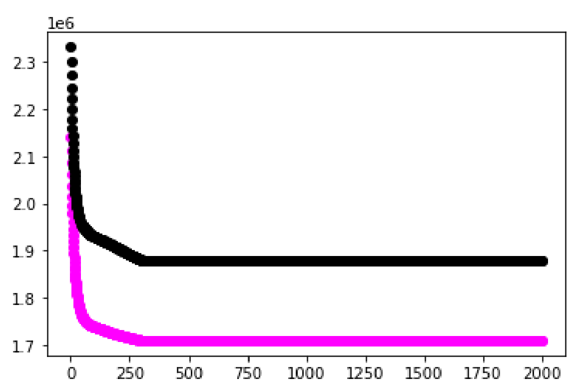
Plotting J and J_cv on different values of Lambda:



Lambda = 1000



Lambda=2000



Lambda = 5000

CONCLUSION:

Error function plot shows that by giving different values of Lambda the data becomes underfit 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)

'US Covid Cases with Regularization Machine Learning Project-(FAISAL JAVED)'

ANNEX C: Prediction Code (Python Notebook)

'Prediction Function for US Covid Cases with Regularization Machine Learning Project-(FAISAL JAVED)'