Optimization Techniques Final Project

COVID-19 Spreading Causes

In this project, we aim to use the optimization tools you learnt to understand the causes promoting Covid-19 spread.

Prof. Dr. Yasmine Abo El-Seoud

Omar Reda 4767 – Mohamed Hossam 4840 – Abo Bakr Hussien 4553 – Islam Mustafa 4595

Optimization Techniques Final Project

COVID-19 Spreading Causes

Problem Solving process we should follow:

1. Problem understanding
2. Solutions design
3. Solutions implementation.

Close Contact (6 feet, 1.8 meters) and Respiratory Droplets

"The virus is thought to spread mainly from person-to-person.

- Between people who are in **close contact** with one another (within about 6 feet)
- Through **respiratory droplets** produced when an infected person coughs, sneezes **or talks**"

This idea, that large droplets of virus-laden mucus are the primary mode of transmission, guides the US CDC's advice to maintain at least a **6-foot distance:** "Maintaining good social distance (about 6 feet) is very important in preventing the spread of COVID-19"

Is 6 feet enough?

Some experts contacted by *LiveScience* think that **6 feet (1.8 meters) is not enough**.

COVID-19

COVID-19 is a **new disease** and we are **still learning about how it spreads**" according to the US Centers for Disease Control and Prevention (CDC)

In general, respiratory virus infection can occur through:

- Contact
- Droplet spray in short range
- Aerosol in long-range

Daily Cases Model

New York Analysis

We used data for New York City, including **number of cases**, **temperature** and **humidity** from date range of 11th of March till 11th of May 2020.

Data Sample

	A	B	C	D	E	F	G	H	I
1	NumberOfDays	Daily_cases	Temp	Humidity	Total	Avg_Temp	Pop_Density	Growth_Rate	Date
2	1	93	21	0.47	93	10	19.45	2.46	11/3/2020
3	2	107	14	0.65	200	10.1	19.45	2.45	12/3/2020
4	3	112	12	0.68	312	10.2	19.45	2.44	13/3/2020
5	4	235	7	0.43	547	10.3	19.45	2.43	14/3/2020
6	5	742	12	0.45	1289	10.4	19.45	2.42	15/03/2020
7	6	1342	12	0.48	2631	10.5	19.45	2.41	16/03/2020
8	7	2341	11	0.79	4972	10.6	19.45	2.4	17/03/2020
9	8	3052	25	0.52	8024	10.7	19.45	2.39	18/03/2020
10	9	1993	18	0.86	10017	10.8	19.45	2.38	19/03/2020
11	10	5440	7	0.83	15457	10.9	19.45	2.37	20/03/2020
12	11	5123	5	0.37	20580	11	19.45	2.36	21/03/2020
13	12	5516	13	0.43	26296	11.1	19.45	2.35	22/03/2020
14	13	6674	7	0.8	32770	11.2	19.45	2.34	23/03/2020
15	14	6097	16	0.64	38867	11.3	19.45	2.33	24/03/2020
16	15	7380	21	0.69	46247	11.4	19.45	2.32	25/03/2020
17	16	7250	12	0.59	53497	11.5	19.45	2.31	26/03/2020
18	17	7413	8	0.54	60910	11.6	19.45	2.3	27/03/2020
19	18	6785	11	0.68	67695	11.7	19.45	2.29	28/03/2020
20	19	8823	9	0.88	76518	11.8	19.45	2.28	29/03/2020
21	20	8104	13	0.72	84622	11.9	19.45	2.27	30/03/2020
22	21	9353	14	0.68	93975	12	19.45	2.26	31/03/2020
23	22	10628	11	0.55	104603	12.1	19.45	2.25	1/4/2020
24	23	11506	17	0.53	116109	12.2	19.45	2.24	2/4/2020
25	24	8477	18	0.7	124586	12.3	19.45	2.23	3/4/2020

Procedures

- We used 2 different models (Exponential Model, Neural Networks Model).
- We trained both models on **60 days** data including the factors affecting the **daily growth.**
- In exponential model we used the following equation **(Y=A.B^x1.C^x2)** then we tried to solve it to get the best values for coefficients A, B, C using Newton Raphson method, x1 is the first factor(**temperature**) and x2 is the second factor(**Humidity**) and Y is the result (**daily number of cases**)

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{N\times 1} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}_{N\times 1} - \begin{bmatrix} & \mathbf{J}^{-}(\mathbf{x}_n) & \end{bmatrix}_{N\times M} \begin{bmatrix} f_1(\mathbf{x}_n) \\ \vdots \\ \vdots \\ \vdots \\ f_M(\mathbf{x}_N) \end{bmatrix}_{M\times 1}$$

- By substituting in the above equation with partial derivative calculations and substitutions we get the best 3 values for A, B, C by calculating the error by trying **random initial guess** to get the closest value to the real ones.
- Then we use the values of A, B, C to get the new values of Y (Predictable Cases).

Code Samples

Exponential Curve Model

```
while(err>(10**(-3))):

    print("*********************************************")
    print("Iteration",iterations)
    print("*********************************************")

    #get the old root
    old_root=np.array(root).astype(np.float64) # you may need to convert root matrix to list

    #Compute Hessian Matrix
    for i in range(0,length):
        for j in range(0,length):
            jacabMatrixVal[i][j] = jacabMatrix[i][j].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Compute Function Matrix
    for i in range(0,length):
        FVal[i]=F[i].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Convert lists to matrices
    jacabMatrixVal = np.asarray(jacabMatrixVal)
    FVal=np.asarray(FVal)

    #compute Hessian Inverse
    jacabMatrixInv = inv(np.matrix(jacabMatrixVal, dtype='float'))

    # Compute new roots
    root =root-jacabMatrixInv.dot(FVal)


    #break at 100 iteration
    iterations=iterations+1
    if(iterations == 100):
        break


    root_norm=np.array(root).astype(np.float64)

    #compute Error
    err=abs(np.linalg.norm(root_norm - old_root)/np.linalg.norm(old_root))
    root=root.tolist()[0]

    print("Result root : ",root)
    print("err : ",err)
```

Neural Networks Model (Instead of Cairo New York)

```python
def train_NN_daily(self):
    excel_sheet_format = pd.read_csv("./Cairo_daily.csv")   # read data from file
    rows = 60   # here it is number of days taken
    features_number = 2
    excel_sheet_format_list = np.array(excel_sheet_format.values.tolist())
    factors_data = excel_sheet_format_list[:, 0:features_number]
    labels = excel_sheet_format_list[:, features_number].reshape(rows, 1)  # cases
    x_train, x_test, y_train, y_test = train_test_split(    # capture 30 % of samples as test
        factors_data, labels, test_size=0.3, random_state=42)
    xx_train = x_train.reshape(x_train.shape[0], features_number, 1)
    xx_test = x_test.reshape(x_test.shape[0], features_number, 1)

    model = Sequential([
        # input layer
        Dense(128, kernel_initializer='normal', input_shape=(
            features_number, 1), activation='relu'),
        # hidden layer
        Dense(256, kernel_initializer='normal', activation='relu'),
        Dense(256, kernel_initializer='normal', activation='relu'),
        Flatten(),
        # output layer
        Dense(1, kernel_initializer='normal', activation='linear'),
    ])

    model.compile(
        optimizer='adam',
        loss='mean_absolute_error',
        metrics=['accuracy', 'mean_absolute_error'],
    )
    model.summary()
    model.fit(xx_train, y_train, epochs=500, batch_size=32,
              validation_split=0.2)
```

Training

Neural Networks Model

```
Train on 38 samples, validate on 10 samples
Epoch 1/500
38/38 [==============================] - 0s 6ms/step - loss: 5796.1408 - accuracy: 0.0000e+00 - mean_absolute_error: 5796.1406 - val_loss: 7130.8306 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7130.8306
Epoch 2/500
38/38 [==============================] - 0s 157us/step - loss: 5795.5465 - accuracy: 0.0000e+00 - mean_absolute_error: 5795.5464 - val_loss: 7130.0830 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7130.0830
Epoch 3/500
38/38 [==============================] - 0s 131us/step - loss: 5794.8795 - accuracy: 0.0000e+00 - mean_absolute_error: 5794.8794 - val_loss: 7129.1211 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7129.1211
Epoch 4/500
38/38 [==============================] - 0s 158us/step - loss: 5794.0136 - accuracy: 0.0000e+00 - mean_absolute_error: 5794.0137 - val_loss: 7127.8789 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7127.8789
Epoch 5/500
38/38 [==============================] - 0s 157us/step - loss: 5792.8547 - accuracy: 0.0000e+00 - mean_absolute_error: 5792.8545 - val_loss: 7126.2373 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7126.2373
Epoch 6/500
38/38 [==============================] - 0s 158us/step - loss: 5791.3904 - accuracy: 0.0000e+00 - mean_absolute_error: 5791.3906 - val_loss: 7124.0967 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7124.0967
Epoch 7/500
38/38 [==============================] - 0s 184us/step - loss: 5789.4015 - accuracy: 0.0000e+00 - mean_absolute_error: 5789.4014 - val_loss: 7121.3320 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7121.3320
Epoch 8/500
38/38 [==============================] - 0s 131us/step - loss: 5786.9987 - accuracy: 0.0000e+00 - mean_absolute_error: 5786.9985 - val_loss: 7117.7983 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7117.7983
Epoch 9/500
38/38 [==============================] - 0s 157us/step - loss: 5783.7223 - accuracy: 0.0000e+00 - mean_absolute_error: 5783.7227 - val_loss: 7113.3428 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7113.3428
Epoch 10/500
38/38 [==============================] - 0s 131us/step - loss: 5779.7278 - accuracy: 0.0000e+00 - mean_absolute_error: 5779.7280 - val_loss: 7107.6797 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7107.6797
Epoch 11/500
38/38 [==============================] - 0s 131us/step - loss: 5774.5165 - accuracy: 0.0000e+00 - mean_absolute_error: 5774.5166 - val_loss: 7100.6499 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7100.6499
Epoch 12/500
38/38 [==============================] - 0s 159us/step - loss: 5768.1930 - accuracy: 0.0000e+00 - mean_absolute_error: 5768.1929 - val_loss: 7091.8657 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7091.8657
Epoch 13/500
38/38 [==============================] - 0s 131us/step - loss: 5760.2393 - accuracy: 0.0000e+00 - mean_absolute_error: 5760.2393 - val_loss: 7081.0220 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7081.0220
Epoch 14/500
38/38 [==============================] - 0s 236us/step - loss: 5750.5753 - accuracy: 0.0000e+00 - mean_absolute_error: 5750.5752 - val_loss: 7067.7710 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7067.7710
Epoch 15/500
38/38 [==============================] - 0s 420us/step - loss: 5738.4026 - accuracy: 0.0000e+00 - mean_absolute_error: 5738.4023 - val_loss: 7051.7656 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7051.7656
Epoch 16/500
38/38 [==============================] - 0s 367us/step - loss: 5723.8433 - accuracy: 0.0000e+00 - mean_absolute_error: 5723.8433 - val_loss: 7032.2891 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7032.2891
Epoch 17/500
38/38 [==============================] - 0s 394us/step - loss: 5706.4498 - accuracy: 0.0000e+00 - mean_absolute_error: 5706.4497 - val_loss: 7008.8765 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 7008.8765
Epoch 18/500
38/38 [==============================] - 0s 289us/step - loss: 5685.9341 - accuracy: 0.0000e+00 - mean_absolute_error: 5685.9341 - val_loss: 6981.4673 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 6981.4673
Epoch 19/500
38/38 [==============================] - 0s 339us/step - loss: 5662.5434 - accuracy: 0.0000e+00 - mean_absolute_error: 5662.5435 - val_loss: 6949.0688 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 6949.0688
Epoch 20/500
38/38 [==============================] - 0s 289us/step - loss: 5636.5135 - accuracy: 0.0000e+00 - mean_absolute_error: 5636.5137 - val_loss: 6911.0518 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 6911.0518
Epoch 21/500
38/38 [==============================] - 0s 289us/step - loss: 5603.9931 - accuracy: 0.0000e+00 - mean_absolute_error: 5603.9932 - val_loss: 6866.3882 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 6866.3882
```

Exponential Curve Model

```
Iteration 0
**********************************************
Result root :  [1.46892682219041, 1.47418238339874, 1.46950309718011]
err :  0.01948276650734132
**********************************************
Iteration 1
**********************************************
Result root :  [1.43050513947553, 1.45001480092527, 1.43564283707724]
err :  0.022228044131642886
**********************************************
Iteration 2
**********************************************
Result root :  [1.37527671888834, 1.42860258887645, 1.39359469749829]
err :  0.02914990628966856
**********************************************
Iteration 3
**********************************************
Result root :  [1.27968086797164, 1.41227123635933, 1.33132402235511]
err :  0.047551591527950667
**********************************************
Iteration 4
**********************************************
Result root :  [1.07523299337684, 1.40663225018011, 1.21514043049572]
err :  0.10118109647550742
**********************************************
Iteration 5
**********************************************
Result root :  [0.474863637861927, 1.43247672767508, 0.943026689382976]
err :  0.30719283656326285
**********************************************
Iteration 6
**********************************************
Result root :  [0.858055919010046, 1.38316787264029, 1.50548540466186]
err :  0.3834511640996457
**********************************************
```

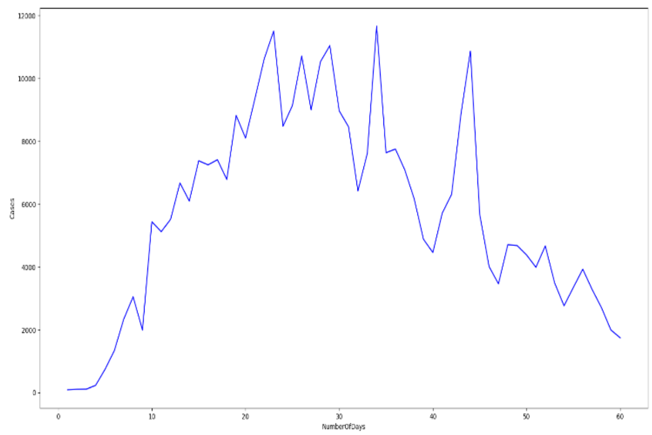
Results

Neural Networks Model

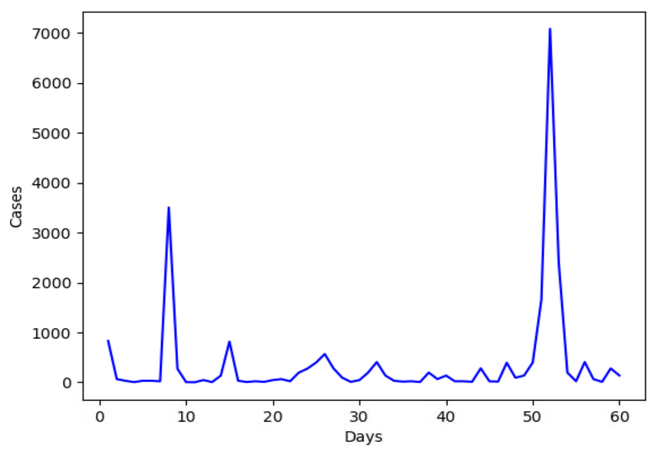
```
*************************Final Results*********************
Random Factors Samples (Temp & Humidity)
[['21' '0.47']
 ['12' '0.48']
 ['8' '0.36']
 ['10' '0.6']
 ['16' '0.64']
 ['11' '0.47']
 ['12' '0.76']
 ['16' '0.53']
 ['7' '0.8']
 ['9' '0.44']
 ['19' '0.78']
 ['23' '0.84']]
*************************
Expected Daily Cases
[['93']
 ['1342']
 ['7090']
 ['4013']
 ['6097']
 ['3352']
 ['11661']
 ['4681']
 ['6674']
 ['2704']
 ['3464']
 ['3991']]
*************************
Model's Prediction
[[7462.2686]
 [5595.299 ]
 [4691.2354]
 [5250.7334]
 [6523.242 ]
 [5381.2236]
 [5762.4697]
 [6457.5684]
 [4745.828 ]
 [4947.1035]
 [7231.141 ]
 [8099.382 ]]
**************************************************
```

Exponential Curve Model

Original



Predicted



Cairo Analysis

We used data for Cairo City, including **number of cases**, **temperature**, **humidity** and from date range of 11th of March till 11th of May 2020.

Data Sample

	A	B	C	D	E	F	G	H	I
1	NumberOfDays	Daily_cases	Temp	Humidity	Total	Avg_Temp	Pop_Density	Growth_Rate	Date
2	1	8	26	0.62	8	26.9	103	1.95	11/3/2020
3	2	13	21	0.75	21	27	103	1.94	12/3/2020
4	3	13	19	0.7	34	27.1	103	1.93	13/3/2020
5	4	17	23	0.7	51	27.2	103	1.92	14/3/2020
6	5	16	27	0.55	67	27.3	103	1.91	15/03/2020
7	6	40	27	0.47	107	27.4	103	1.9	16/03/2020
8	7	30	18	0.59	137	27.5	103	1.89	17/03/2020
9	8	14	19	0.51	151	27.6	103	1.88	18/03/2020
10	9	46	20	0.49	197	27.7	103	1.87	19/03/2020
11	10	29	16	0.53	226	27.8	103	1.86	20/03/2020
12	11	9	19	0.49	235	27.9	103	1.85	21/03/2020
13	12	33	25	0.34	268	28	103	1.84	22/03/2020
14	13	39	28	0.24	307	28.1	103	1.83	23/03/2020
15	14	36	26	0.39	343	28.2	103	1.82	24/03/2020
16	15	54	24	0.5	397	28.3	103	1.81	25/03/2020
17	16	39	27	0.4	436	28.4	103	1.8	26/03/2020
18	17	41	29	0.4	477	28.5	103	1.79	27/03/2020
19	18	40	24	0.52	517	28.6	103	1.78	28/03/2020
20	19	33	27	0.31	550	28.7	103	1.77	29/03/2020
21	20	47	31	0.17	598	28.8	103	1.76	30/03/2020
22	21	54	30	0.23	651	28.9	103	1.75	31/03/2020
23	22	69	24	0.52	720	29	103	1.74	1/4/2020

Procedures

- We used 2 different models (Exponential Model, Neural Networks Model).
- We trained both models on **60 days** data including the factors affecting the **daily growth.**
- In exponential model we used the following equation **(Y=A.B^x1.C^x2)** then we tried to solve it to get the best values for coefficients A, B, C using Newton Raphson method, x1 is the first factor(**temperature**) and x2 is the second factor(**Humidity**) and Y is the result (**daily number of cases**)

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{N\times 1} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}_{N\times 1} - \begin{bmatrix} & \mathbf{J}^{-}(\mathbf{x}_n) & \end{bmatrix}_{N\times M} \begin{bmatrix} f_1(\mathbf{x}_n) \\ \vdots \\ \vdots \\ \vdots \\ f_M(\mathbf{x}_N) \end{bmatrix}_{M\times 1}$$

- By substituting in the above equation with partial derivative calculations and substitutions we get the best 3 values for A, B, C by calculating the error by trying **random initial guess** to get the closest value to the real ones.

- Then we use the values of A, B, C to get the new values of Y (Predictable Cases).

- For the neural networks model we used several built-in libraries as tensor flow, Keras (Sequential, Dense, Flatten) and Sklearn.model_selection in order to generate a NN model that train on the data comes from the csv file.
- Then we select a random sample from the total 60 samples and calculate the new predicted value for number of cases.

Code Samples

Exponential Curve Model

```
while(err>(10**(-3))):

    print("**********************************************")
    print("Iteration",iterations)
    print("**********************************************")

    #get the old root
    old_root=np.array(root).astype(np.float64) # you may need to convert root matrix to list

    #Compute Hessian Matrix
    for i in range(0,length):
        for j in range(0,length):
            jacabMatrixVal[i][j] = jacabMatrix[i][j].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Compute Function Matrix
    for i in range(0,length):
        FVal[i]=F[i].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Convert lists to matrices
    jacabMatrixVal = np.asarray(jacabMatrixVal)
    FVal=np.asarray(FVal)

    #compute Hessian Inverse
    jacabMatrixInv = inv(np.matrix(jacabMatrixVal, dtype='float'))

    # Compute new roots
    root =root-jacabMatrixInv.dot(FVal)


    #break at 100 iteration
    iterations=iterations+1
    if(iterations == 100):
        break


    root_norm=np.array(root).astype(np.float64)

    #compute Error
    err=abs(np.linalg.norm(root_norm - old_root)/np.linalg.norm(old_root))
    root=root.tolist()[0]

    print("Result root : ",root)
    print("err : ",err)
```

Neural Networks Model

```
def train_NN_daily(self):
    excel_sheet_format = pd.read_csv("./Cairo_daily.csv")   # read data from file
    rows = 60   # here it is number of days taken
    features_number = 2
    excel_sheet_format_list = np.array(excel_sheet_format.values.tolist())
    factors_data = excel_sheet_format_list[:, 0:features_number]
    labels = excel_sheet_format_list[:, features_number].reshape(rows, 1)  # cases
    x_train, x_test, y_train, y_test = train_test_split(    # capture 30 % of samples as test
        factors_data, labels, test_size=0.3, random_state=42)
    xx_train = x_train.reshape(x_train.shape[0], features_number, 1)
    xx_test = x_test.reshape(x_test.shape[0], features_number, 1)

    model = Sequential([
        # input layer
        Dense(128, kernel_initializer='normal', input_shape=(
            features_number, 1), activation='relu'),
        # hidden layer
        Dense(256, kernel_initializer='normal', activation='relu'),
        Dense(256, kernel_initializer='normal', activation='relu'),
        Flatten(),
        # output layer
        Dense(1, kernel_initializer='normal', activation='linear'),
    ])

    model.compile(
        optimizer='adam',
        loss='mean_absolute_error',
        metrics=['accuracy', 'mean_absolute_error'],
    )
    model.summary()
    model.fit(xx_train, y_train, epochs=500, batch_size=32,
              validation_split=0.2)
```

Training

Neural Networks Model training

```
Epoch 1/500
38/38 [==============================] - 0s 4ms/step - loss: 137.4385 - accuracy: 0.0000e+00 - mean_absolute_error: 137.4385 - val_loss: 154.4441 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 154.4441
Epoch 2/500
38/38 [==============================] - 0s 315us/step - loss: 136.4268 - accuracy: 0.0000e+00 - mean_absolute_error: 136.4268 - val_loss: 153.4972 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 153.4972
Epoch 3/500
38/38 [==============================] - 0s 230us/step - loss: 135.4768 - accuracy: 0.0000e+00 - mean_absolute_error: 135.4768 - val_loss: 152.3268 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 152.3268
Epoch 4/500
38/38 [==============================] - 0s 184us/step - loss: 134.2995 - accuracy: 0.0000e+00 - mean_absolute_error: 134.2995 - val_loss: 150.7660 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 150.7660
Epoch 5/500
38/38 [==============================] - 0s 157us/step - loss: 132.7102 - accuracy: 0.0000e+00 - mean_absolute_error: 132.7102 - val_loss: 148.7008 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 148.7008
Epoch 6/500
38/38 [==============================] - 0s 184us/step - loss: 130.6521 - accuracy: 0.0000e+00 - mean_absolute_error: 130.6521 - val_loss: 145.9908 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 145.9908
Epoch 7/500
38/38 [==============================] - 0s 130us/step - loss: 127.8970 - accuracy: 0.0000e+00 - mean_absolute_error: 127.8970 - val_loss: 142.4571 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 142.4571
Epoch 8/500
38/38 [==============================] - 0s 157us/step - loss: 124.3216 - accuracy: 0.0263 - mean_absolute_error: 124.3216 - val_loss: 137.9044 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 137.9044
Epoch 9/500
38/38 [==============================] - 0s 157us/step - loss: 120.0668 - accuracy: 0.0000e+00 - mean_absolute_error: 120.0668 - val_loss: 132.5171 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 132.5171
Epoch 10/500
38/38 [==============================] - 0s 158us/step - loss: 115.1599 - accuracy: 0.0000e+00 - mean_absolute_error: 115.1599 - val_loss: 126.2423 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 126.2423
Epoch 11/500
38/38 [==============================] - 0s 132us/step - loss: 109.5270 - accuracy: 0.0000e+00 - mean_absolute_error: 109.5270 - val_loss: 119.5115 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 119.5115
Epoch 12/500
38/38 [==============================] - 0s 210us/step - loss: 102.6481 - accuracy: 0.0263 - mean_absolute_error: 102.6481 - val_loss: 112.3389 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 112.3389
Epoch 13/500
38/38 [==============================] - 0s 210us/step - loss: 97.1340 - accuracy: 0.0000e+00 - mean_absolute_error: 97.1340 - val_loss: 106.9888 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 106.9888
Epoch 14/500
38/38 [==============================] - 0s 368us/step - loss: 91.3873 - accuracy: 0.0000e+00 - mean_absolute_error: 91.3873 - val_loss: 103.0051 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 103.0051
Epoch 15/500
38/38 [==============================] - 0s 839us/step - loss: 85.8467 - accuracy: 0.0263 - mean_absolute_error: 85.8467 - val_loss: 99.2332 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 99.2332
Epoch 16/500
38/38 [==============================] - 0s 158us/step - loss: 80.8169 - accuracy: 0.0263 - mean_absolute_error: 80.8169 - val_loss: 97.6492 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 97.6492
Epoch 17/500
38/38 [==============================] - 0s 876us/step - loss: 76.7095 - accuracy: 0.0000e+00 - mean_absolute_error: 76.7095 - val_loss: 97.5424 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 97.5424
Epoch 18/500
38/38 [==============================] - 0s 551us/step - loss: 76.4590 - accuracy: 0.0000e+00 - mean_absolute_error: 76.4590 - val_loss: 103.1368 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 103.1368
Epoch 19/500
38/38 [==============================] - 0s 525us/step - loss: 80.0960 - accuracy: 0.0000e+00 - mean_absolute_error: 80.0960 - val_loss: 105.6446 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 105.6446
Epoch 20/500
38/38 [==============================] - 0s 604us/step - loss: 80.9466 - accuracy: 0.0000e+00 - mean_absolute_error: 80.9466 - val_loss: 104.3506 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 104.3506
Epoch 21/500
```

Exponential Curve Model Training

```
Intial Condition :  [1.5, 1.5, 1.5]
*********************************************
Iteration 0
*********************************************
Result root :  [1.48249911143137, 1.48009837551559, 1.48567853443060]
err :  0.011626669033499683
*********************************************
Iteration 1
*********************************************
Result root :  [1.46543605497883, 1.46042061629513, 1.47202472761975]
err :  0.011454728551954878
*********************************************
Iteration 2
*********************************************
Result root :  [1.44883314581924, 1.44096008664271, 1.45910519020667]
err :  0.011255339121661642
*********************************************
Iteration 3
*********************************************
Result root :  [1.43271577281066, 1.42170991458139, 1.44699618829483]
err :  0.011024334801305352
*********************************************
Iteration 4
*********************************************
Result root :  [1.41711235331770, 1.40266309451574, 1.43578355202034]
err :  0.010757345967256584
*********************************************
Iteration 5
*********************************************
Result root :  [1.40205394619251, 1.38381270647901, 1.42556130067451]
err :  0.010450167440021632
*********************************************
Iteration 6
*********************************************
Result root :  [1.38757326018890, 1.36515233915644, 1.41642789002987]
err :  0.01009947639783745
*********************************************
Iteration 7
*********************************************
Result root :  [1.37370263049000, 1.34667686449999, 1.40847821783662]
err :  0.009704153279984694
a =  1.37370263049000
b =  1.34667686449999
c =  1.40847821783662
```

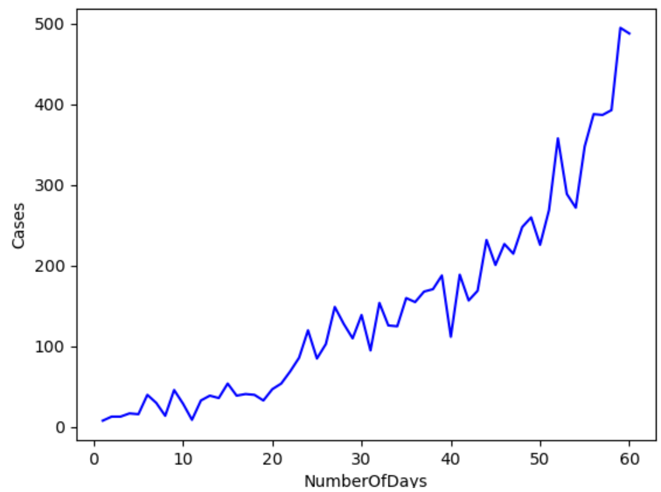
Results

Neural Networks Model

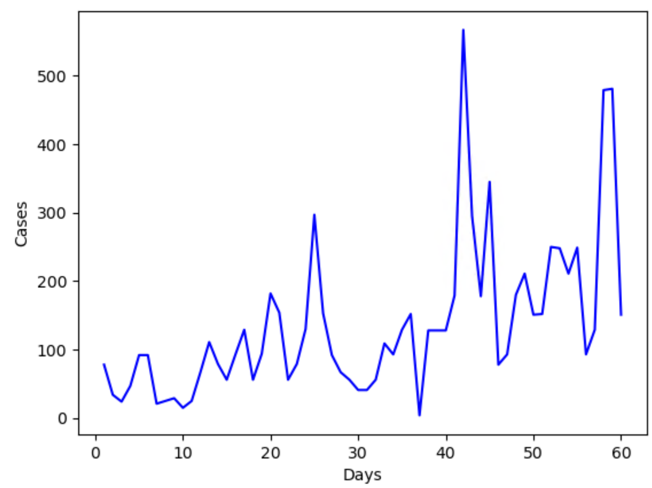
```
************************Final Results************************
Random Factors Samples (Temp & Humidity)
[['26' '0.62']
 ['27' '0.47']
 ['8' '0.34']
 ['26' '0.51']
 ['26' '0.39']
 ['33' '0.34']
 ['27' '0.44']
 ['32' '0.38']
 ['28' '0.24']
 ['37' '0.34']
 ['27' '0.46']
 ['30' '0.39']]
************************
Expected Daily Cases
[['8']
 ['40']
 ['168']
 ['227']
 ['36']
 ['348']
 ['125']
 ['260']
 ['39']
 ['393']
 ['215']
 ['269']]
************************
Model's Prediction
[[113.50916 ]
 [141.20886 ]
 [ 15.747493]
 [127.31291 ]
 [145.11647 ]
 [202.94705 ]
 [145.67574 ]
 [194.2007  ]
 [102.3328  ]
 [234.70764 ]
 [142.69783 ]
 [176.87706 ]]
*************************************************
```

Exponential Curve Model

Original



Predicted



Total Cases Model

New York Analysis

We used data for New York City, including **accumulative number of cases**, **average temperature** and **economic growth rate** from date range of 11th of March till 11th of May 2020.

Data Sample

	A	B	C	D	E	F	G	H	I
1	NumberOfDays	Daily_cases	Temp	Humidity	Total	Avg_Temp	Pop_Density	Growth_Rate	Date
2	1	93	21	0.47	93	10	19.45	2.46	11/3/2020
3	2	107	14	0.65	200	10.1	19.45	2.45	12/3/2020
4	3	112	12	0.68	312	10.2	19.45	2.44	13/3/2020
5	4	235	7	0.43	547	10.3	19.45	2.43	14/3/2020
6	5	742	12	0.45	1289	10.4	19.45	2.42	15/03/2020
7	6	1342	12	0.48	2631	10.5	19.45	2.41	16/03/2020
8	7	2341	11	0.79	4972	10.6	19.45	2.4	17/03/2020
9	8	3052	25	0.52	8024	10.7	19.45	2.39	18/03/2020
10	9	1993	18	0.86	10017	10.8	19.45	2.38	19/03/2020
11	10	5440	7	0.83	15457	10.9	19.45	2.37	20/03/2020
12	11	5123	5	0.37	20580	11	19.45	2.36	21/03/2020
13	12	5516	13	0.43	26296	11.1	19.45	2.35	22/03/2020
14	13	6674	7	0.8	32770	11.2	19.45	2.34	23/03/2020
15	14	6097	16	0.64	38867	11.3	19.45	2.33	24/03/2020
16	15	7380	21	0.69	46247	11.4	19.45	2.32	25/03/2020
17	16	7250	12	0.59	53497	11.5	19.45	2.31	26/03/2020
18	17	7413	8	0.54	60910	11.6	19.45	2.3	27/03/2020
19	18	6785	11	0.68	67695	11.7	19.45	2.29	28/03/2020
20	19	8823	9	0.88	76518	11.8	19.45	2.28	29/03/2020
21	20	8104	13	0.72	84622	11.9	19.45	2.27	30/03/2020
22	21	9353	14	0.68	93975	12	19.45	2.26	31/03/2020
23	22	10628	11	0.55	104603	12.1	19.45	2.25	1/4/2020
24	23	11506	17	0.53	116109	12.2	19.45	2.24	2/4/2020
25	24	8477	18	0.7	124586	12.3	19.45	2.23	3/4/2020

Procedures

- We used 2 different models (Exponential Model, Neural Networks Model).
- We trained both models on **60 days** data including the factors affecting the **total daily growth**.
- In exponential model we used the following equation **(Y=A.B^x1.C^x2)** then we tried to solve it to get the best values for coefficients A, B, C using Newton Raphson method, x1 is the first factor(**average temperature**) and x2 is the second factor(**economic growth rate**) and Y is the result (**accumulative number of cases**)

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{N\times 1} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}_{N\times 1} - \begin{bmatrix} \quad & \mathbf{J}^{-}(\mathbf{x}_n) & \quad \end{bmatrix}_{N\times M} \begin{bmatrix} f_1(\mathbf{x}_n) \\ \vdots \\ \vdots \\ \vdots \\ f_M(\mathbf{x}_N) \end{bmatrix}_{M\times 1}$$

- By substituting in the above equation with partial derivative calculations and substitutions we get the best 3 values for A, B, C by calculating the error by trying **random initial guess** to get the closest value to the real ones.
- Then we use the values of A, B, C to get the new values of Y (Predictable Cases).
- For the neural networks model we used several built-in libraries as tensor flow, Keras (Sequential, Dense, Flatten) and Sklearn.model_selection in order to generate a NN model that train on the data comes from the csv file.
- Then we select a random sample from the total 60 samples and calculate the new predicted value for number of cases.

Code Samples

Exponential Curve Model

```
while(err>(10**(-3))):

    print("*********************************************")
    print("Iteration",iterations)
    print("*********************************************")

    #get the old root
    old_root=np.array(root).astype(np.float64) # you may need to convert root matrix to list

    #Compute Hessian Matrix
    for i in range(0,length):
        for j in range(0,length):
            jacabMatrixVal[i][j] = jacabMatrix[i][j].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Compute Function Matrix
    for i in range(0,length):
        FVal[i]=F[i].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Convert lists to matrices
    jacabMatrixVal = np.asarray(jacabMatrixVal)
    FVal=np.asarray(FVal)

    #compute Hessian Inverse
    jacabMatrixInv = inv(np.matrix(jacabMatrixVal, dtype='float'))

    # Compute new roots
    root =root-jacabMatrixInv.dot(FVal)


    #break at 100 iteration
    iterations=iterations+1
    if(iterations == 100):
        break


    root_norm=np.array(root).astype(np.float64)

    #compute Error
    err=abs(np.linalg.norm(root_norm - old_root)/np.linalg.norm(old_root))
    root=root.tolist()[0]

    print("Result root : ",root)
    print("err : ",err)
```

Neural Networks Model (Instead of Cairo New York)

```python
def train_NN_total(self):
    excel_sheet_format = pd.read_csv("./Cairo_total.csv")   # read data from file
    rows = 60   # here it is number of days taken
    features_number = 2
    excel_sheet_format_list = np.array(excel_sheet_format.values.tolist())
    factors_data = excel_sheet_format_list[:, 0:features_number]     # temp , humidity
    labels = excel_sheet_format_list[:, features_number].reshape(rows, 1)  # cases
    x_train, x_test, y_train, y_test = train_test_split(    # capture 30 % of samples as test
        factors_data, labels, test_size=0.3, random_state=42)
    xx_train = x_train.reshape(x_train.shape[0], features_number, 1)
    xx_test = x_test.reshape(x_test.shape[0], features_number, 1)

    model = Sequential([
        # input layer
        Dense(128, kernel_initializer='normal', input_shape=(
            features_number, 1), activation='relu'),
        # hidden layer
        Dense(256, kernel_initializer='normal', activation='relu'),
        Dense(256, kernel_initializer='normal', activation='relu'),
        Flatten(),
        # output layer
        Dense(1, kernel_initializer='normal', activation='linear'),
    ])

    model.compile(
        optimizer='adam',
        loss='mean_absolute_error',
        metrics=['accuracy', 'mean_absolute_error'],
    )
    model.summary()
    model.fit(xx_train, y_train, epochs=500, batch_size=32,
              validation_split=0.2)
```

Training

Exponential Curve Model

```
Iteration 0
*******************************************
Result root :  [1.41424862676371, 1.40800721322390, 1.41881798105386]
err :  0.05761492994639464
*******************************************
Iteration 1
*******************************************
Result root :  [1.33371962929275, 1.32154554588376, 1.34263207419920]
err :  0.05741550949494350 6
*******************************************
Iteration 2
*******************************************
Result root :  [1.25786501673248, 1.24009436195010, 1.27087446824608]
err :  0.05737228119897572
*******************************************
Iteration 3
*******************************************
Result root :  [1.18633816253469, 1.16330763041860, 1.20319818245082]
err :  0.05738360668871784
*******************************************
Iteration 4
*******************************************
Result root :  [1.11886059079338, 1.09090425916982, 1.13932671360935]
err :  0.05741959916927848
*******************************************
Iteration 5
*******************************************
Result root :  [1.05518293614585, 1.02263433683260, 1.07901115619812]
err :  0.05747270243895089
*******************************************
Iteration 6
*******************************************
Result root :  [0.995073088115497, 0.958268005442464, 1.02201781936796]
err :  0.05754257708606018 4
*******************************************
```

Neural Networks Model

```
38/38 [==============================] - 0s 262us/step - loss: 91138.3713 - accuracy: 0.0000e+00 - mean_absolute_error: 91138.3672 - val_loss: 91883.7969 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 91883.7969
Epoch 481/500
38/38 [==============================] - 0s 368us/step - loss: 91104.7558 - accuracy: 0.0000e+00 - mean_absolute_error: 91104.7578 - val_loss: 91952.6797 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 91952.6797
Epoch 482/500
38/38 [==============================] - 0s 289us/step - loss: 91069.3503 - accuracy: 0.0000e+00 - mean_absolute_error: 91069.3516 - val_loss: 92096.7344 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92096.7344
Epoch 483/500
38/38 [==============================] - 0s 211us/step - loss: 91038.3507 - accuracy: 0.0000e+00 - mean_absolute_error: 91038.3516 - val_loss: 92226.6094 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92226.6094
Epoch 484/500
38/38 [==============================] - 0s 236us/step - loss: 90966.3664 - accuracy: 0.0000e+00 - mean_absolute_error: 90966.3672 - val_loss: 92399.6094 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92399.6094
Epoch 485/500
38/38 [==============================] - 0s 237us/step - loss: 90915.3812 - accuracy: 0.0000e+00 - mean_absolute_error: 90915.3828 - val_loss: 92661.7656 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92661.7656
Epoch 486/500
38/38 [==============================] - 0s 236us/step - loss: 90959.0905 - accuracy: 0.0000e+00 - mean_absolute_error: 90959.0938 - val_loss: 92914.5625 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92914.5625
Epoch 487/500
38/38 [==============================] - 0s 367us/step - loss: 91024.0399 - accuracy: 0.0000e+00 - mean_absolute_error: 91024.0391 - val_loss: 93116.6797 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93116.6797
Epoch 488/500
38/38 [==============================] - 0s 262us/step - loss: 91109.6147 - accuracy: 0.0000e+00 - mean_absolute_error: 91109.6172 - val_loss: 93245.8984 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93245.8984
Epoch 489/500
38/38 [==============================] - 0s 131us/step - loss: 91105.5119 - accuracy: 0.0000e+00 - mean_absolute_error: 91105.5156 - val_loss: 93287.8594 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93287.8594
Epoch 490/500
38/38 [==============================] - 0s 211us/step - loss: 91137.2796 - accuracy: 0.0000e+00 - mean_absolute_error: 91137.2734 - val_loss: 93271.3516 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93271.3516
Epoch 491/500
38/38 [==============================] - 0s 131us/step - loss: 91131.4433 - accuracy: 0.0000e+00 - mean_absolute_error: 91131.4375 - val_loss: 93225.4922 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93225.4922
Epoch 492/500
38/38 [==============================] - 0s 342us/step - loss: 91074.0127 - accuracy: 0.0000e+00 - mean_absolute_error: 91074.0156 - val_loss: 93209.9766 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93209.9766
Epoch 493/500
38/38 [==============================] - 0s 288us/step - loss: 91057.2290 - accuracy: 0.0000e+00 - mean_absolute_error: 91057.2266 - val_loss: 93177.7812 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93177.7812
Epoch 494/500
38/38 [==============================] - 0s 157us/step - loss: 91039.3318 - accuracy: 0.0000e+00 - mean_absolute_error: 91039.3281 - val_loss: 93161.5391 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93161.5391
Epoch 495/500
38/38 [==============================] - 0s 315us/step - loss: 91009.2167 - accuracy: 0.0000e+00 - mean_absolute_error: 91009.2188 - val_loss: 93168.4531 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93168.4531
Epoch 496/500
38/38 [==============================] - 0s 315us/step - loss: 91012.3721 - accuracy: 0.0000e+00 - mean_absolute_error: 91012.3672 - val_loss: 93184.5391 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93184.5391
Epoch 497/500
38/38 [==============================] - 0s 236us/step - loss: 91040.6558 - accuracy: 0.0000e+00 - mean_absolute_error: 91040.6562 - val_loss: 93163.1641 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93163.1641
Epoch 498/500
38/38 [==============================] - 0s 210us/step - loss: 90994.5654 - accuracy: 0.0000e+00 - mean_absolute_error: 90994.5625 - val_loss: 93045.5625 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 93045.5625
Epoch 499/500
38/38 [==============================] - 0s 315us/step - loss: 90878.7860 - accuracy: 0.0000e+00 - mean_absolute_error: 90878.7891 - val_loss: 92916.0625 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92916.0625
Epoch 500/500
38/38 [==============================] - 0s 316us/step - loss: 90784.3532 - accuracy: 0.0000e+00 - mean_absolute_error: 90784.3516 - val_loss: 92678.4922 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 92678.4922
```

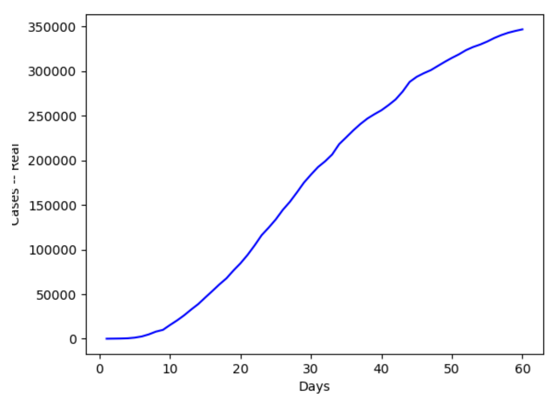
Results

Neural Networks Model

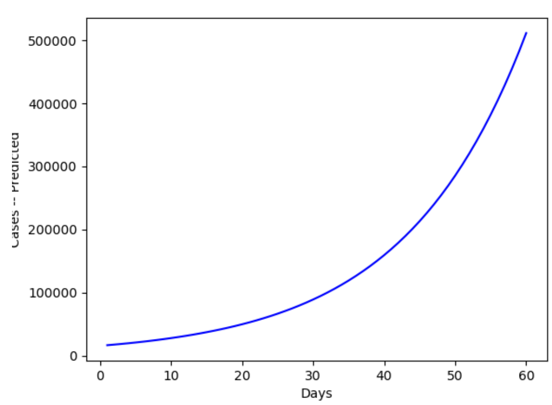
```
************************Final Results************************
Random Factors Samples (Avg. Temp & Growth Rate)
[['10.0' '19.45']
 ['10.5' '19.45']
 ['13.6' '19.45']
 ['14.5' '19.45']
 ['11.3' '19.45']
 ['15.4' '19.45']
 ['13.3' '19.45']
 ['14.8' '19.45']
 ['11.2' '19.45']
 ['15.7' '19.45']
 ['14.6' '19.45']
 ['15.0' '19.45']]
************************
Expected Daily Cases
[['93']
 ['2631']
 ['240613']
 ['297576']
 ['38867']
 ['333082']
 ['218134']
 ['310429']
 ['32770']
 ['343000']
 ['301040']
 ['318804']]
************************
Model's Prediction
[[164256.98]
 [169487.98]
 [201920.14]
 [211335.92]
 [177857.58]
 [220751.7 ]
 [198781.56]
 [214474.53]
 [176811.39]
 [223890.3 ]
 [212382.12]
 [216566.92]]
```

Exponential Curve

Original



Predicted



Cairo Analysis

We used data for New York City, including **accumulative number of cases**, **average temperature** and **economic growth rate** from date range of 11th of March till 11th of May 2020.

Data Sample

	A	B	C	D	E	F	G	H	I
1	NumberOfDays	Daily_cases	Temp	Humidity	Total	Avg_Temp	Pop_Density	Growth_Rate	Date
2	1	8	26	0.62	8	26.9	103	1.95	11/3/2020
3	2	13	21	0.75	21	27	103	1.94	12/3/2020
4	3	13	19	0.7	34	27.1	103	1.93	13/3/2020
5	4	17	23	0.7	51	27.2	103	1.92	14/3/2020
6	5	16	27	0.55	67	27.3	103	1.91	15/03/2020
7	6	40	27	0.47	107	27.4	103	1.9	16/03/2020
8	7	30	18	0.59	137	27.5	103	1.89	17/03/2020
9	8	14	19	0.51	151	27.6	103	1.88	18/03/2020
10	9	46	20	0.49	197	27.7	103	1.87	19/03/2020
11	10	29	16	0.53	226	27.8	103	1.86	20/03/2020
12	11	9	19	0.49	235	27.9	103	1.85	21/03/2020
13	12	33	25	0.34	268	28	103	1.84	22/03/2020
14	13	39	28	0.24	307	28.1	103	1.83	23/03/2020
15	14	36	26	0.39	343	28.2	103	1.82	24/03/2020
16	15	54	24	0.5	397	28.3	103	1.81	25/03/2020
17	16	39	27	0.4	436	28.4	103	1.8	26/03/2020
18	17	41	29	0.4	477	28.5	103	1.79	27/03/2020
19	18	40	24	0.52	517	28.6	103	1.78	28/03/2020
20	19	33	27	0.31	550	28.7	103	1.77	29/03/2020
21	20	47	31	0.17	598	28.8	103	1.76	30/03/2020
22	21	54	30	0.23	651	28.9	103	1.75	31/03/2020
23	22	69	24	0.52	720	29	103	1.74	1/4/2020

Procedures

- We used 2 different models (Exponential Model, Neural Networks Model).
- We trained both models on **60 days** data including the factors affecting the **total daily growth**.
- In exponential model we used the following equation **(Y=A.B^x1.C^x2)** then we tried to solve it to get the best values for coefficients A, B, C using Newton Raphson method, x1 is the first factor(**average temperature**) and x2 is the second factor(**economic growth rate**) and Y is the result (**accumulative number of cases**)

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{N\times 1} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}_{N\times 1} - \begin{bmatrix} & \mathbf{J}^{-}(\mathbf{x}_n) & \end{bmatrix}_{N\times M} \begin{bmatrix} f_1(\mathbf{x}_n) \\ \vdots \\ \vdots \\ \vdots \\ f_M(\mathbf{x}_N) \end{bmatrix}_{M\times 1}$$

- By substituting in the above equation with partial derivative calculations and substitutions we get the best 3 values for A, B, C by calculating the error by trying **random initial guess** to get the closest value to the real ones.
- Then we use the values of A, B, C to get the new values of Y (Predictable Cases).
- For the neural networks model we used several built-in libraries as tensor flow, Keras (Sequential, Dense, Flatten) and Sklearn.model_selection in order to generate a NN model that train on the data comes from the csv file.
- Then we select a random sample from the total 60 samples and calculate the new predicted value for number of cases.

Code Samples

Exponential Curve Model

```
while(err>(10**(-3))):

    print("*********************************************")
    print("Iteration",iterations)
    print("*********************************************")

    #get the old root
    old_root=np.array(root).astype(np.float64) # you may need to convert root matrix to list

    #Compute Hessian Matrix
    for i in range(0,length):
        for j in range(0,length):
            jacabMatrixVal[i][j] = jacabMatrix[i][j].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Compute Function Matrix
    for i in range(0,length):
        FVal[i]=F[i].subs([(a, root[0]), (b, root[1]), (c, root[2])])

    #Convert lists to matrices
    jacabMatrixVal = np.asarray(jacabMatrixVal)
    FVal=np.asarray(FVal)

    #compute Hessian Inverse
    jacabMatrixInv = inv(np.matrix(jacabMatrixVal, dtype='float'))

    # Compute new roots
    root =root-jacabMatrixInv.dot(FVal)


    #break at 100 iteration
    iterations=iterations+1
    if(iterations == 100):
        break


    root_norm=np.array(root).astype(np.float64)

    #compute Error
    err=abs(np.linalg.norm(root_norm - old_root)/np.linalg.norm(old_root))
    root=root.tolist()[0]

    print("Result root : ",root)
    print("err : ",err)
```

Neural Networks Model

```python
def train_NN_total(self):
    excel_sheet_format = pd.read_csv("./Cairo_total.csv")   # read data from file
    rows = 60   # here it is number of days taken
    features_number = 2
    excel_sheet_format_list = np.array(excel_sheet_format.values.tolist())
    factors_data = excel_sheet_format_list[:, 0:features_number]     # temp , humidity
    labels = excel_sheet_format_list[:, features_number].reshape(rows, 1)  # cases
    x_train, x_test, y_train, y_test = train_test_split(    # capture 30 % of samples as test
        factors_data, labels, test_size=0.3, random_state=42)
    xx_train = x_train.reshape(x_train.shape[0], features_number, 1)
    xx_test = x_test.reshape(x_test.shape[0], features_number, 1)

    model = Sequential([
        # input layer
        Dense(128, kernel_initializer='normal', input_shape=(
            features_number, 1), activation='relu'),
        # hidden layer
        Dense(256, kernel_initializer='normal', activation='relu'),
        Dense(256, kernel_initializer='normal', activation='relu'),
        Flatten(),
        # output layer
        Dense(1, kernel_initializer='normal', activation='linear'),
    ])

    model.compile(
        optimizer='adam',
        loss='mean_absolute_error',
        metrics=['accuracy', 'mean_absolute_error'],
    )
    model.summary()
    model.fit(xx_train, y_train, epochs=500, batch_size=32,
              validation_split=0.2)
```

Training

Exponential Curve Model

```
Iteration 0
******************************************
Result root :  [1.47931114228465, 1.47739732974920, 1.48337883781749]
err :  0.013417360607751304
******************************************
Iteration 1
******************************************
Result root :  [1.45902172099852, 1.45512577453505, 1.46730191767514]
err :  0.01332127324406407
******************************************
Iteration 2
******************************************
Result root :  [1.43916773445412, 1.43319701209575, 1.45185676807725]
err :  0.013191846892515752
******************************************
Iteration 3
******************************************
Result root :  [1.41981376425900, 1.41163299492821, 1.43719826945539]
err :  0.013006495824745837
******************************************
Iteration 4
******************************************
Result root :  [1.40107471502276, 1.39047182697681, 1.42360392840053]
err :  0.012726279471059553
******************************************
Iteration 5
******************************************
Result root :  [1.38315613867240, 1.36977663970989, 1.41158056624782]
err :  0.012285071124809786
******************************************
Iteration 6
******************************************
Result root :  [1.36643310016992, 1.34964811674245, 1.40208277264335]
err :  0.01157764264056329
******************************************
Iteration 7
******************************************
Result root :  [1.35161740960595, 1.33023874558425, 1.39700042000715]
err :  0.010488601684363484
******************************************
```

Neural Networks Model

```
33/33 [==============================] - 0s 847us/step - loss: 1704.9283 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.9283 - val_loss: 2107.2205 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2107.2205
Epoch 479/500
33/33 [==============================] - 0s 332us/step - loss: 1704.3184 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.3184 - val_loss: 2107.1838 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2107.1838
Epoch 480/500
33/33 [==============================] - 0s 181us/step - loss: 1704.0505 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.0505 - val_loss: 2106.2173 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2106.2173
Epoch 481/500
33/33 [==============================] - 0s 424us/step - loss: 1704.3778 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.3778 - val_loss: 2104.6152 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2104.6152
Epoch 482/500
33/33 [==============================] - 0s 514us/step - loss: 1706.1841 - accuracy: 0.0000e+00 - mean_absolute_error: 1706.1841 - val_loss: 2102.5642 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2102.5642
Epoch 483/500
33/33 [==============================] - 0s 424us/step - loss: 1709.5722 - accuracy: 0.0000e+00 - mean_absolute_error: 1709.5723 - val_loss: 2101.0701 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2101.0701
Epoch 484/500
33/33 [==============================] - 0s 393us/step - loss: 1710.9883 - accuracy: 0.0000e+00 - mean_absolute_error: 1710.9883 - val_loss: 2100.8521 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2100.8521
Epoch 485/500
33/33 [==============================] - 0s 454us/step - loss: 1711.0956 - accuracy: 0.0000e+00 - mean_absolute_error: 1711.0957 - val_loss: 2100.7786 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2100.7786
Epoch 486/500
33/33 [==============================] - 0s 635us/step - loss: 1711.1263 - accuracy: 0.0000e+00 - mean_absolute_error: 1711.1263 - val_loss: 2100.8589 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2100.8589
Epoch 487/500
33/33 [==============================] - 0s 319us/step - loss: 1710.6988 - accuracy: 0.0000e+00 - mean_absolute_error: 1710.6989 - val_loss: 2101.8879 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2101.8879
Epoch 488/500
33/33 [==============================] - 0s 242us/step - loss: 1709.1485 - accuracy: 0.0000e+00 - mean_absolute_error: 1709.1486 - val_loss: 2103.7048 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2103.7048
Epoch 489/500
33/33 [==============================] - 0s 726us/step - loss: 1707.6791 - accuracy: 0.0000e+00 - mean_absolute_error: 1707.6791 - val_loss: 2105.2881 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2105.2881
Epoch 490/500
33/33 [==============================] - 0s 423us/step - loss: 1705.4109 - accuracy: 0.0000e+00 - mean_absolute_error: 1705.4110 - val_loss: 2105.7734 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2105.7734
Epoch 491/500
33/33 [==============================] - 0s 395us/step - loss: 1704.5093 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.5094 - val_loss: 2105.3308 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2105.3308
Epoch 492/500
33/33 [==============================] - 0s 393us/step - loss: 1704.7925 - accuracy: 0.0000e+00 - mean_absolute_error: 1704.7925 - val_loss: 2104.1277 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2104.1277
Epoch 493/500
33/33 [==============================] - 0s 453us/step - loss: 1706.0699 - accuracy: 0.0000e+00 - mean_absolute_error: 1706.0698 - val_loss: 2102.3403 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2102.3403
Epoch 494/500
33/33 [==============================] - 0s 514us/step - loss: 1708.1045 - accuracy: 0.0000e+00 - mean_absolute_error: 1708.1044 - val_loss: 2100.1199 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2100.1199
Epoch 495/500
33/33 [==============================] - 0s 514us/step - loss: 1711.8370 - accuracy: 0.0000e+00 - mean_absolute_error: 1711.8370 - val_loss: 2098.4524 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2098.4524
Epoch 496/500
33/33 [==============================] - 0s 393us/step - loss: 1713.0651 - accuracy: 0.0000e+00 - mean_absolute_error: 1713.0651 - val_loss: 2099.6318 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2099.6318
Epoch 497/500
33/33 [==============================] - 0s 362us/step - loss: 1714.4532 - accuracy: 0.0000e+00 - mean_absolute_error: 1714.4532 - val_loss: 2104.1941 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2104.1941
Epoch 498/500
33/33 [==============================] - 0s 151us/step - loss: 1716.5302 - accuracy: 0.0000e+00 - mean_absolute_error: 1716.5303 - val_loss: 2109.6484 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2109.6484
Epoch 499/500
33/33 [==============================] - 0s 181us/step - loss: 1720.0659 - accuracy: 0.0000e+00 - mean_absolute_error: 1720.0658 - val_loss: 2113.6084 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2113.6084
Epoch 500/500
33/33 [==============================] - 0s 514us/step - loss: 1721.1739 - accuracy: 0.0000e+00 - mean_absolute_error: 1721.1738 - val_loss: 2116.5244 - val_accuracy: 0.0000e+00 - val_mean_absolute_error: 2116.5244
```

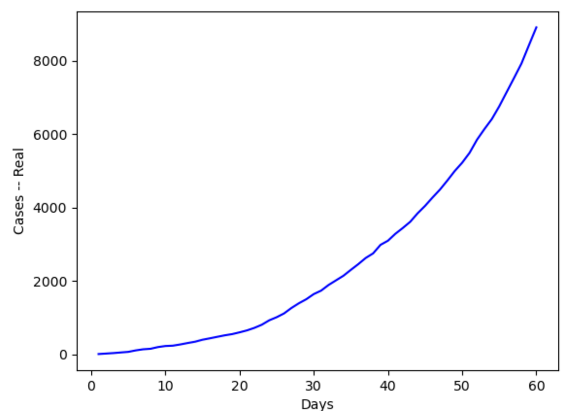
Results

Neural Networks Model

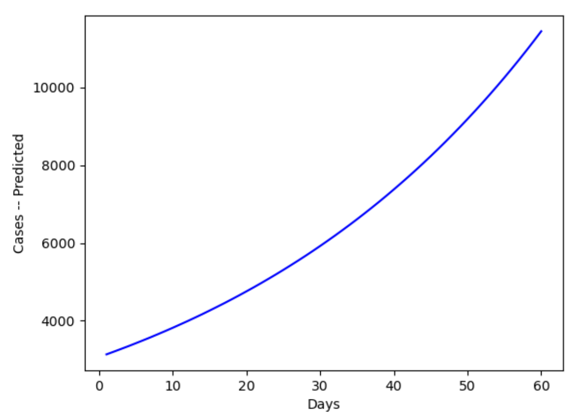
```
************************Final Results************************
Random Factors Samples (Avg. Temp & Growth Rate)
[['26.9' '1.95']
 ['27.4' '1.9']
 ['30.5' '1.59']
 ['31.4' '1.5']
 ['28.2' '1.82']
 ['32.3' '1.41']
 ['30.2' '1.62']
 ['31.7' '1.47']
 ['28.1' '1.83']
 ['32.6' '1.38']
 ['31.5' '1.49']
 ['31.9' '1.45']]
*************************
Expected Daily Cases
[['8']
 ['107']
 ['2623']
 ['4269']
 ['343']
 ['6754']
 ['2140']
 ['4992']
 ['307']
 ['7922']
 ['4484']
 ['5487']]
*************************
Model's Prediction
[[1244.3805]
 [1301.8226]
 [1657.9609]
 [1761.3568]
 [1393.7292]
 [1864.7521]
 [1623.4966]
 [1795.8217]
 [1382.2408]
 [1899.2167]
 [1772.8457]
 [1818.7988]]
```

Exponential Curve

Original



Predicted



Conclusion

- First of all, we found this project extremely helpful to enrich our knowledge with topics we didn't deal with it before, it was hard and we faced many problems and learned so much from this experience.
- We tried to search for several ways in order to make the project with the most proper way, in order to achieve real values as much as we can.
- After a long search we found a documentation talking about Newton Raphson optimization using multivariable function, and we decided to move forward with this solution.
- We built the model and trained it with a data of 2 locations (New York, Cairo) to test whether the model will predict the value of cases whether total or daily cases after training correctly or not.
- We didn't use countries as we found that it's not convenient to calculate a daily temperature for a country of daily humidity for a country as those were our factors, so we decided to move forward with states as we thought it will be more logical with our selected values.
- We noticed after running the code and training the model that the efficiency when dealing with total cases is much more accurate than dealing with the daily ones, also the factors played an important role in getting the accurate values for the cases avg temperature and growth rate was more efficient in total cases than temperature and humidity in daily cases, and that is very obvious from the daily plots compared to total plots.
- Also, the values of total cases before training are already in an exponential form because it's always increases, however the values of daily cases are increasing and decreasing as number of days increases, which also affect the accuracy of both exponential and Neural Networks models.

Optimization Techniques Final Project

All Files are Uploaded on Google Drive Folder

https://drive.google.com/drive/folders/1wMH4Qq7_KSvv9L8G3iudK44sb_2Hnhwn?usp=sharing

Thank You