# CC483: OPTIMIZATION TECHNIQUES

## FINAL PROJECT


| NAME | ID |
|---|---|
| AHMED EL-RAMADY | 4298 |
| ABDELAZIZ HUSSEIN | 4126 |
| ZEYAD EZZAT | 4492 |
| OMAR AHMED IBRAHIM | 3954 |
| MOSTAFA TOUHAMY | 3851 |

# Introduction

The coronavirus has certainly impacted the world since its outbreak. The virus originated from China and diffused to the majority of countries around the world with the total number of COIVD-19 cases reaching over 2,500,000 cases.

When it comes to the current circumstances in Egypt, the aforementioned figure is not something to be taken lightly because it presents a huge threat that should be taken into consideration by authorities to avoid any catastrophes in the future.

In this report, we illustrate an Artificial Neural Network (ANN) that uses several factors to aid the prediction of the number of daily and overall cases across every country. This method could help authorities determine the necessary precautionary measures to be taken in order to reduce the spread of the virus.

# Description

All data used for this project was obtained after thorough research using a repository on GitHub to get the number of cases worldwide. Additionally, we used a web application to download the factors leading to coronavirus symptoms in every country from World Weather Online. The image below illustrates how data is retrieved from every country.

```
In [10]: from wwo_hist import retrieve_hist_data
```

```
In [12]: import os
         os.chdir("C:\\Users\\Omar\\Desktop\\Optimization Project\\data")
```

```
In [13]: from datetime import date
         frequency = 24
         start_date = '22-JAN-2020'
         end_date = date.today()
         api_key = '978e02711f8a4085ab3131813200205'
         location_list = ['USA', 'Spain', 'Italy', 'UK', 'France', 'Germany', 'Ru
         ssia', 'Turkey', 'Iran', 'Brazil', 'Canada', 'Belgium', 'Peru', 'Netherl
         ands', 'India', 'Switzerland', 'Ecuador', 'Saudi Arabia', 'Portugal', 'S
         weden', 'Ireland', 'Mexico', 'Pakistan', 'Singapore', 'Chile', 'Israel',
         'Belarus', 'Austria', 'Qatar', 'Japan', 'Poland', 'UAE', 'Romania', 'Ukr
         aine', 'Indonesia', 'S. Korea', 'Denmark', 'Serbia', 'Philippines', 'Ban
         gladesh', 'Norway', 'Czechia', 'Dominican Republic', 'Colombia', 'Austra
         lia', 'Panama', 'Malaysia', 'South Africa', 'Egypt', 'Finland', 'Morocc
         o', 'Kuwait', 'Argentina', 'Algeria', 'Moldova', 'Luxembourg', 'Kazakhst
         an', 'Bahrain', 'Thailand', 'Hungary', 'Greece', 'Oman', 'Afghanistan',
         'Armenia', 'Nigeria', 'Iraq', 'Uzbekistan', 'Croatia', 'Ghana', 'Azerbai
         jan', 'Bosnia and Herzegovina', 'Cameroon', 'Iceland', 'Estonia', 'Bulga
         ria', 'Cuba', 'Guinea', 'North Macedonia', 'New Zealand', 'Slovenia', 'S
         lovakia', 'Lithuania', 'Ivory Coast', 'Bolivia', 'Djibouti', 'Hong Kon
         g', 'Senegal', 'Tunisia', 'Honduras', 'Latvia', 'Cyprus', 'Albania', 'Ky
         rgyzstan', 'Andorra', 'Lebanon', 'Niger', 'Costa Rica', 'Diamond Princes
         s', 'Sri Lanka', 'Burkina Faso', 'Uruguay', 'Guatemala', 'DRC', 'Somali
         a', 'Georgia', 'San Marino', 'Mayotte', 'Channel Islands', 'Sudan', 'Mal
         i', 'Maldives', 'Tanzania', 'Malta', 'Jordan', 'El Salvador', 'Jamaica',
         'Taiwan', 'Réunion', 'Kenya', 'Palestine', 'Venezuela', 'Paraguay', 'Mau
         ritius', 'Montenegro', 'Isle of Man', 'Equatorial Guinea', 'Gabon', 'Vie
         tnam', 'Guinea-Bissau', 'Rwanda', 'Congo', 'Faeroe Islands', 'Martiniqu
         e', 'Sierra Leone', 'Liberia', 'Guadeloupe', 'Myanmar', 'Gibraltar', 'Br
         unei', 'Madagascar', 'Ethiopia', 'French Guiana', 'Togo', 'Cabo Verde',
         'Cambodia', 'Zambia', 'Trinidad and Tobago', 'Bermuda', 'Eswatini', 'Aru
         ba', 'Monaco', 'Benin', 'Haiti', 'Uganda', 'Bahamas', 'Guyana', 'Liechte
         nstein', 'Barbados', 'Mozambique', 'Sint Maarten', 'Cayman Islands', 'Ch
         ad', 'CAR', 'Libya', 'Nepal', 'French Polynesia', 'Macao', 'South Suda
         n', 'Syria', 'Eritrea', 'Mongolia', 'Saint Martin', 'Malawi', 'Zimbabw
         e', 'Tajikistan', 'Angola', 'Antigua and Barbuda', 'Timor-Leste', 'Botsw
         ana', 'Grenada', 'Laos', 'Belize', 'Fiji', 'New Caledonia', 'Saint Luci
         a', 'Curaçao', 'Sao Tome and Principe', 'Dominica', 'Namibia', 'St. Vinc
         ent Grenadines', 'Saint Kitts and Nevis', 'Nicaragua', 'Falkland Island
         s', 'Gambia', 'Turks and Caicos', 'Burundi', 'Montserrat', 'Greenland',
         'Vatican City', 'Seychelles', 'Suriname', 'MS Zaandam', 'Mauritania', 'P
         apua New Guinea', 'Yemen', 'Bhutan', 'British Virgin Islands', 'Caribbea
         n Netherlands', 'St. Barth', 'Western Sahara', 'Anguilla', 'Comoros', 'S
         aint Pierre Miquelon', 'China']

         hist_weather_data = retrieve_hist_data(api_key,
                                         location_list,
                                         start_date,
                                         end_date,
                                         frequency,
                                         location_label = False,
                                         export_csv = True,
                                         store_df = True)

         Retrieving weather data for Afghanistan
```

# Discussion, Code, and Results

The daily number of COVID-19 data retrieved from GitHub is loaded into a Pandas data frame, the categorical columns are then converted into numeric ones to ease the preprocessing step.

```
[ ] import pandas as pd
    import numpy as np
    df = pd.read_csv("/content/time_series_19-covid-Confirmed.csv")
    df_arr = np.array(df)
```

```
[ ] row = df_arr[0,4:]
```

```
[ ] countries = []
    for index,row in df.iterrows():
        countries.append(row['Country/Region'])
```

```
[ ] # import the necessary module
    from sklearn import preprocessing
    # create the LabelEncoder Object
    le = preprocessing.LabelEncoder()
    # convert the categorical columns into numeric
    df['Country/Region'] = le.fit_transform(df['Country/Region'])
```

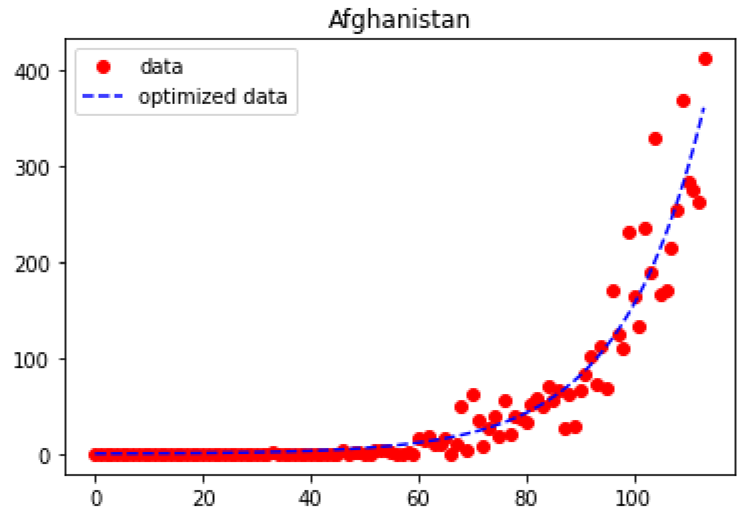
A new array of cases is added, where the difference in the number of daily cases in every country is computed. We also added the date of the discovery of coronavirus in every country.

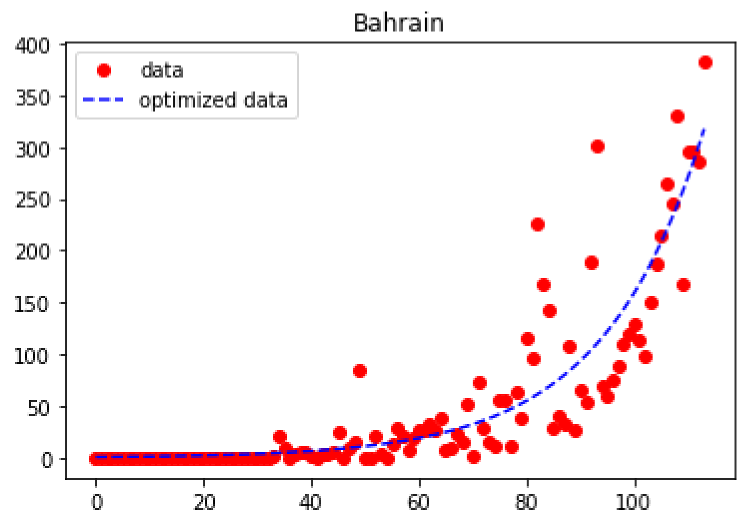
```
[ ] country_newCases = []

    for i in range(0, len(countries)):
        country_row = list(df.loc[i].values)
        country_cases = country_row[4:]
        newCases = [0]
        for i in range(len(country_cases) - 1):
            if country_cases[i + 1] > country_cases[i]:
                newCases.append(country_cases[i + 1] - country_cases[i])
            else:
                newCases.append(0)
        country_newCases.append(newCases)

    number_of_days = range(0,len(df_arr[0,4:]))
    temp = []
    for i in range(len(number_of_days)):
        temp.append(float(number_of_days[i]))
    temp = np.array(temp)
```

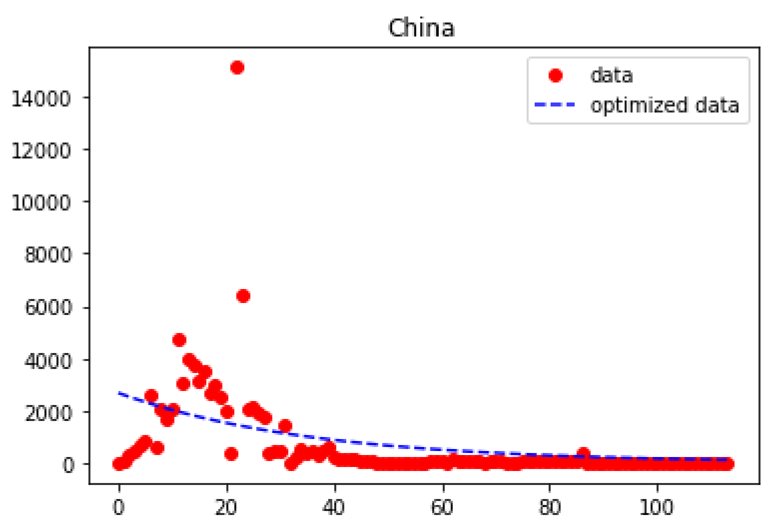
The following is the model which uses the exponential function to fit the data:

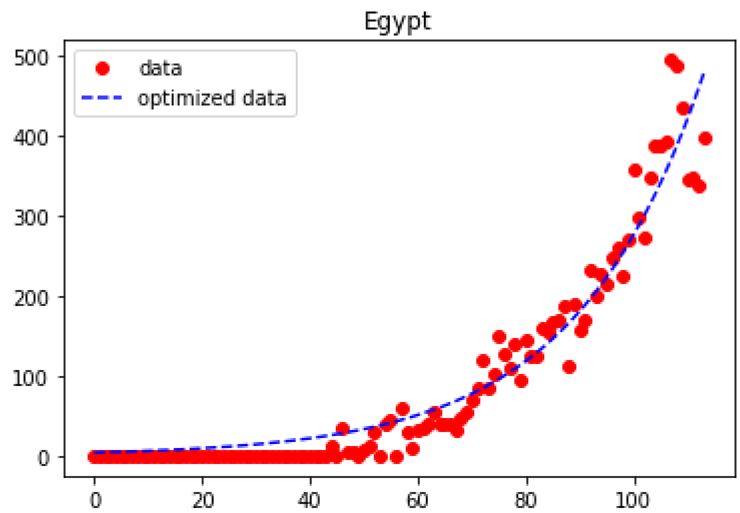
```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

no_optimal = []
number_of_days = range(0,len(df_arr[0,4:]))

for i in range(0, len(countries)):
    try:
        param, param_cov = curve_fit(lambda t,a,b: a*np.exp(b*t),  number_of_days,  country_newCases[i],  p0=(4, 0.1))
        ans = (param[0]*(np.exp(param[1] * temp)))

        plt.figure()
        plt.plot(number_of_days, country_newCases[i], 'o', color = 'red', label = "data")
        plt.plot(number_of_days, ans, '--', color = 'blue', label = 'optimized data')
        plt.legend()
        plt.title(countries[i])
        plt.show()
    except:
        no_optimal.append(i)
```

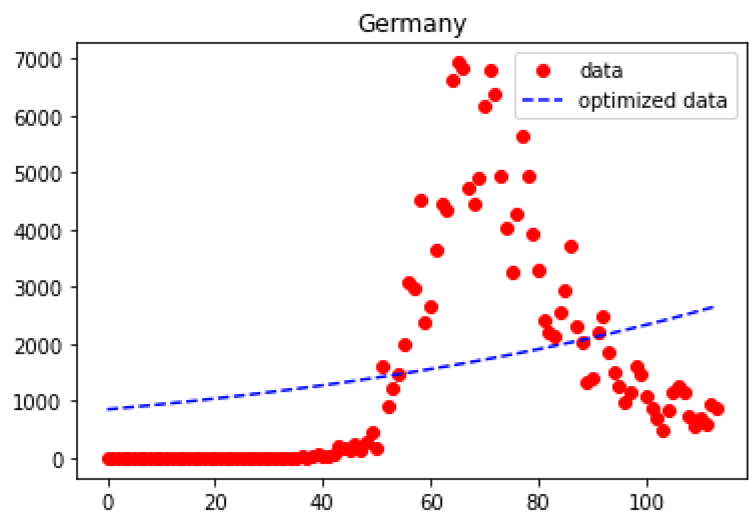
The images below are examples of countries using the exponential model.

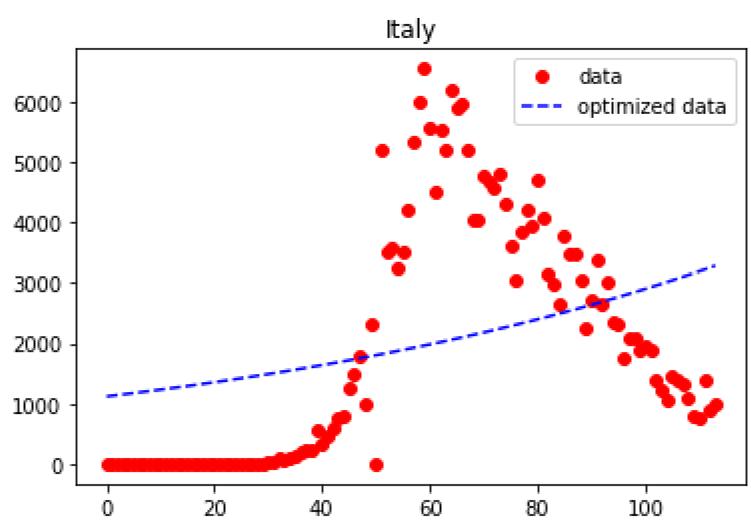


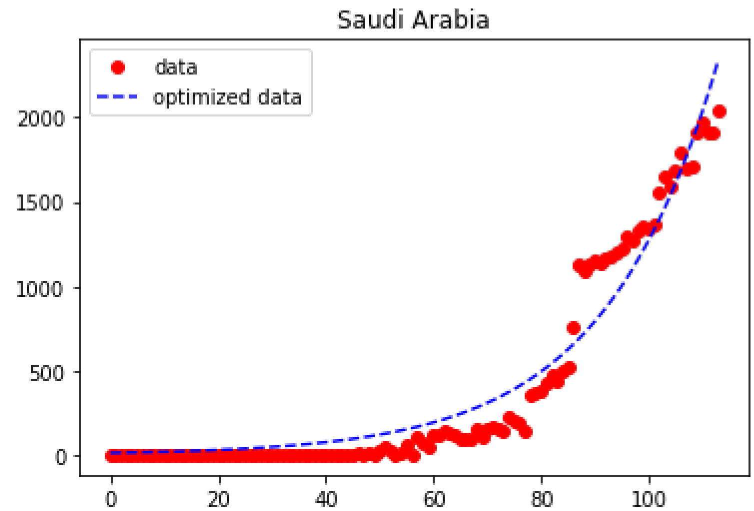



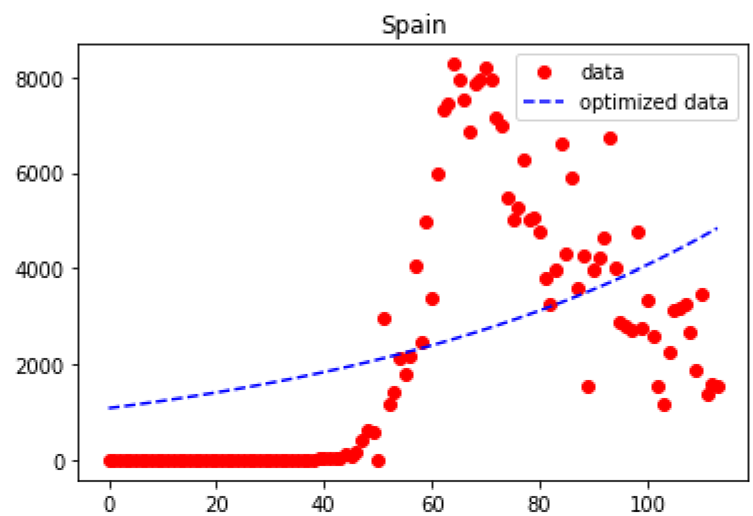



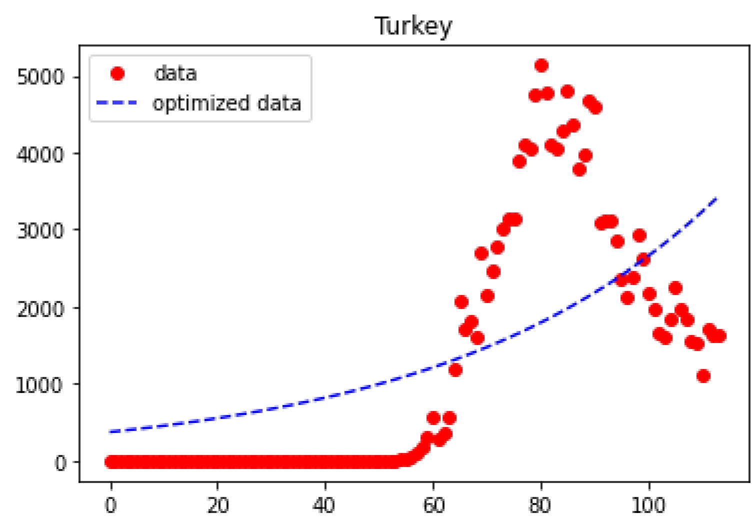

Egypt
data
optimized data
500
400
300
200
100
0
0
20
40
60
80
100

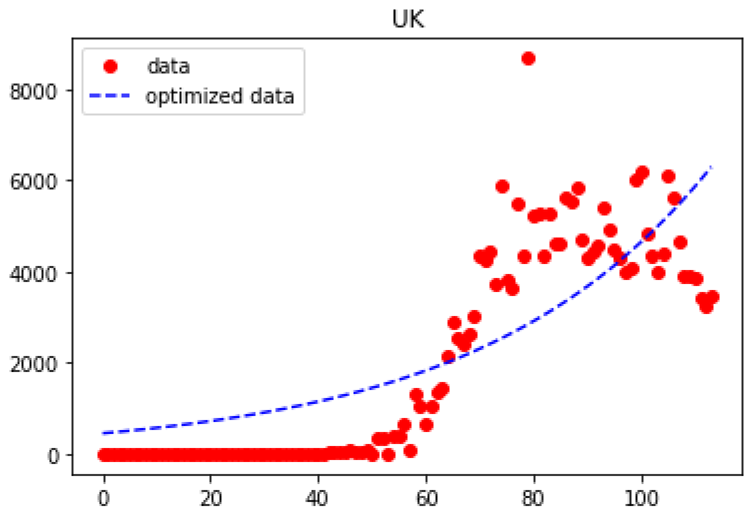

Germany
data
optimized data
7000
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100

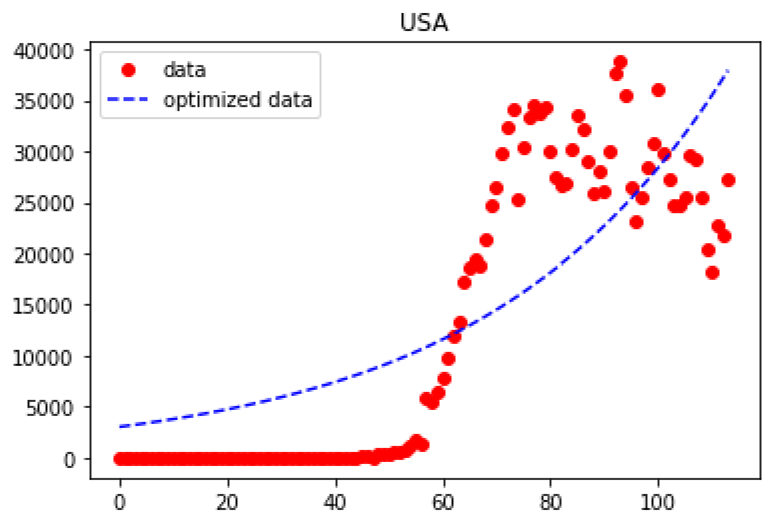

Italy
data
optimized data
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100


Saudi Arabia
data
optimized data
0
500
1000
1500
2000
0
20
40
60
80
100


Spain
data
optimized data
0
2000
4000
6000
8000
0
20
40
60
80
100

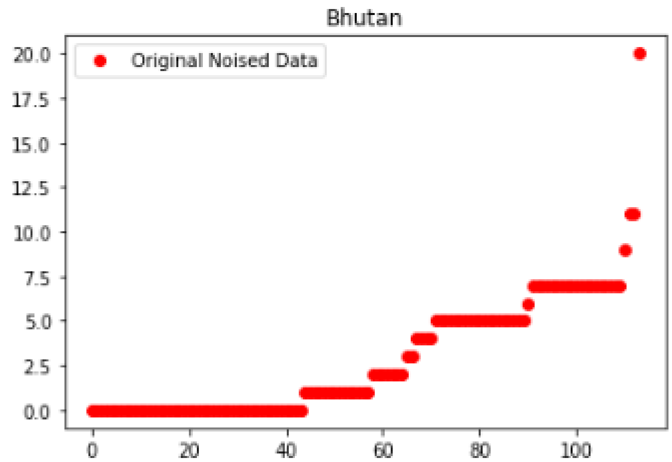

Turkey
data
optimized data
0
1000
2000
3000
4000
5000
0
20
40
60
80
100

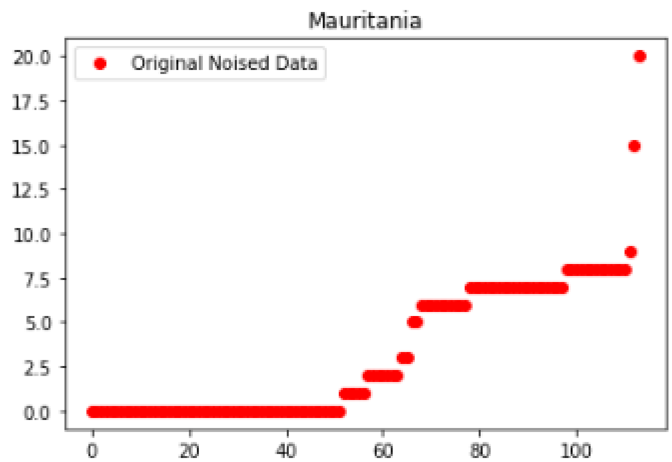



```
[ ] import matplotlib.pyplot as plt

    print("We cannot find Optimal Solutions for these Countries!")

    for i in range(len(no_optimal)):
        y = df_arr[no_optimal[i], 4:]
        plt.figure()
        plt.plot(range(0, len(y)), y, 'o', color ='red', label ="Original Noised Data")
        plt.legend()
        plt.title(df_arr[no_optimal[i],0])
        plt.show()
```

```
We cannot find Optimal Solutions for these Countries!
```

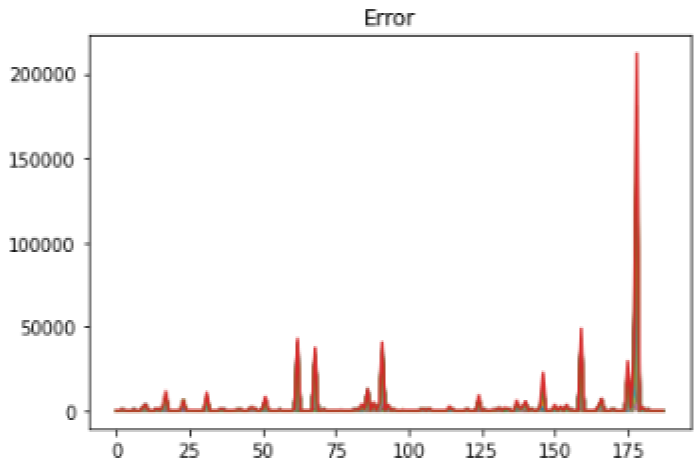



```
[ ]  def func(x, a, b, c):
         return a * np.exp(-b * x) + c

     error = []
     for i in range(0, df_arr.shape[0]):
         x = np.linspace(0, 1, len(df_arr[i, 4:]))
         y = df_arr[i, 4:]
         try:
             popt, pcov = curve_fit(func, x, y)
             yn = np.array(func(x, *popt))
             error.append(abs(yn - y))
         except:
             continue
```

```
[ ] print(len(error))
    plt.figure()
    plt.title('Error')
    plt.plot(error)
    plt.show()
```

```
188
```



We opted to get the factors from three countries, that is Egypt, Italy, and Spain. Hence, it’s easier to analyze the data and get clear results.

**Model Class One : Number of daily cases : Neural Network**

```
[ ] path ="/content/Egypt.csv"
    path1 ="/content/Italy.csv"
    path2 ="/content/Spain.csv"

    factors = pd.read_csv(path)
    factors1 = pd.read_csv(path1)
    factors2 = pd.read_csv(path2)
```

```
[ ] import datetime as DT
    def numOfDays(date2, date1):
        d = (date1 - date2).days
        if d > 0:
            return d
        else:
            return 0
```

The curfew start dates of every country is taken into account for the three aforementioned countries.

```
[ ] # curfew start dates of --> Egypt, Spain, Italy
    curfew_start_date = [DT.date(2020, 3, 23),DT.date(2020, 3, 14),DT.date(2020, 3, 9)]
    first_day = DT.date(2020, 1, 22)
    today = DT.date(2020,5,14)
    difference = numOfDays(first_day, today)
```

A new list is added to compute the number of days since the start of the curfew for every country. (Egypt, Italy, and Spain)

```
[ ] curfew_diff = []
    number_of_days_since_curfew = []
    for j in range(len(curfew_start_date)):
        curfew_diff = numOfDays(curfew_start_date[j], today)
        before_curfew = difference - curfew_diff
        number_of_days_since_curfew.append([0] * before_curfew)
        i = curfew_diff
        for i in range(curfew_diff+1):
            day_i = today - DT.timedelta(days=i)
            number_of_days_since_curfew[j].append(numOfDays(day_i, today))
```

```
[ ] factors['Number of days since the start of curfew'] = number_of_days_since_curfew[0]
    factors1['Number of days since the start of curfew'] = number_of_days_since_curfew[1]
    factors2['Number of days since the start of curfew'] = number_of_days_since_curfew[2]
```

The correlation between every factor and the number of new cases is calculated as the non-correlated factors are not taken into consideration. Only factors x3, x6, x7, and x8 were deemed as neutral correlations and therefore, not considered in our evaluations.

```
[ ] x1 = factors['mintempC'].values
    x2 = factors['maxtempC'].values
    x3 = factors['totalSnow_cm'].values
    x4 = factors['Number of days since the start of curfew'].values
    x5 = factors['humidity'].values
    x6 = factors['pressure'].values
    x7 = factors['winddirDegree'].values
    x8 = factors['moon_illumination'].values
    y = factors['New Cases'].values
    from scipy.stats import pearsonr
    corr1, _ = pearsonr(x1, y)
    corr2, _ = pearsonr(x2, y)
    corr3, _ = pearsonr(x3, y)
    corr4, _ = pearsonr(x4, y)
    corr5, _ = pearsonr(x5, y)
    corr6, _ = pearsonr(x6, y)
    corr7, _ = pearsonr(x7, y)
    corr8, _ = pearsonr(x8, y)
    print('Pearsons correlation: %.3f' % corr1)
    print('Pearsons correlation: %.3f' % corr2)
    print('Pearsons correlation: %.3f' % corr3)
    print('Pearsons correlation: %.3f' % corr4)
    print('Pearsons correlation: %.3f' % corr5)
    print('Pearsons correlation: %.3f' % corr6)
    print('Pearsons correlation: %.3f' % corr7)
    print('Pearsons correlation: %.3f' % corr8)
```

```
Pearsons correlation: 0.629
Pearsons correlation: 0.741
Pearsons correlation: nan
Pearsons correlation: 0.971
Pearsons correlation: -0.406
Pearsons correlation: -0.393
Pearsons correlation: 0.020
Pearsons correlation: 0.239
```

Maximum and minimum temperatures along with humidity, and the number of days since the start of curfew were the factors selected in our evaluations.

```
[ ] # Considering Egypt, Italy, Spain
    factors['New Cases'] = country_newCases[54]
    factors1['New Cases'] = country_newCases[93]
    factors2['New Cases'] = country_newCases[163]
```

```
[ ] cols = [col for col in factors.columns if col in ['maxtempC','mintempC','humidity','Number of days since the start of curfew']]
    cols1 = [col for col in factors1.columns if col in ['maxtempC','mintempC','humidity','Number of days since the start of curfew']]
    cols2 = [col for col in factors2.columns if col in ['maxtempC','mintempC','humidity','Number of days since the start of curfew']]

    # feature
    data = factors[cols]
    data1 = factors1[cols1]
    data2 = factors2[cols2]
```

```
[ ] target = factors['New Cases']
    target1 = factors1['New Cases']
    target2 = factors2['New Cases']
```

The following is the neural network model:

```
[ ] from keras.models import Sequential
    from tensorflow.keras import layers
    from tensorflow.keras import regularizers
    from tensorflow.keras import initializers
    from keras.layers import Dense, Dropout
    from keras import metrics
    from keras import backend as K
    from keras.wrappers.scikit_learn import KerasRegressor
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
    from sklearn.preprocessing import StandardScaler
    from sklearn.pipeline import Pipeline
    import matplotlib.pyplot as plt
```

The factors of every country are preprocessed, scaled, and then split into training and test data with an allocation of 15% to test data and the remaining to train data, where 18 test examples and 96 train examples are used.

```
X_egypt = data.values
Y_egypt = target

X_italy = data1.values
Y_italy = target1

X_spain = data2.values
Y_spain = target2

from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_scale1 = min_max_scaler.fit_transform(X_egypt)
X_scale2 = min_max_scaler.fit_transform(X_italy)
X_scale3 = min_max_scaler.fit_transform(X_spain)

from sklearn.model_selection import train_test_split

X_egypt_train, X_egypt_test, Y_egypt_train, Y_egypt_test = train_test_split(X_scale1, Y_egypt, test_size=0.15)
X_italy_train, X_italy_test, Y_italy_train, Y_italy_test = train_test_split(X_scale2, Y_italy, test_size=0.15)
X_spain_train, X_spain_test, Y_spain_train, Y_spain_test = train_test_split(X_scale3, Y_spain, test_size=0.15)

print(X_egypt_train.shape,X_egypt_test.shape,Y_egypt_train.shape,Y_egypt_test.shape)
print(X_italy_train.shape,X_italy_test.shape,Y_italy_train.shape,Y_italy_test.shape)
print(X_spain_train.shape,X_spain_test.shape,Y_spain_train.shape,Y_spain_test.shape)
```

```
(96, 4) (18, 4) (96,) (18,)
(96, 4) (18, 4) (96,) (18,)
(96, 4) (18, 4) (96,) (18,)
```

This is the first model we used, where the input takes all the factors mentioned above. The first hidden layer has 4,000 neurons, 400 neurons on the next, followed by 40 neurons on the layer after that, and finally, the last layer has just one neuron. At every layer, we used an L2 regularizer with a Rectified Linear Unit (ReLU) function and a dropout with value 0.3 to reduce overfitting and to optimize the cost function. The loss function used is a Mean Squared Error (MSE). However, in the final layer, we used a linear function instead of a ReLU one.

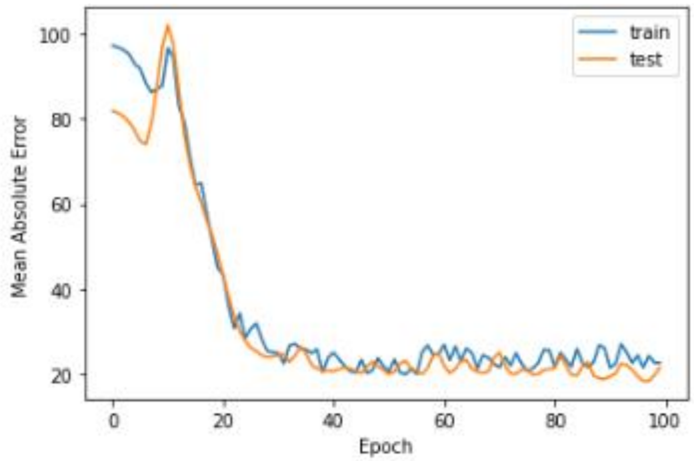
```
def create_model1(X_train):
    # create model
    model1 = Sequential()

    model1.add(Dense(4000, input_dim=X_train.shape[1], activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(400, activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(40, activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(1,activation='linear',kernel_regularizer=regularizers.l2(1e-4)))

    model1.compile(optimizer ='adam', loss = 'mean_squared_error',
              metrics =[metrics.mae])

    return model1
model1 = create_model1(X_egypt_train)
model1.summary()
```

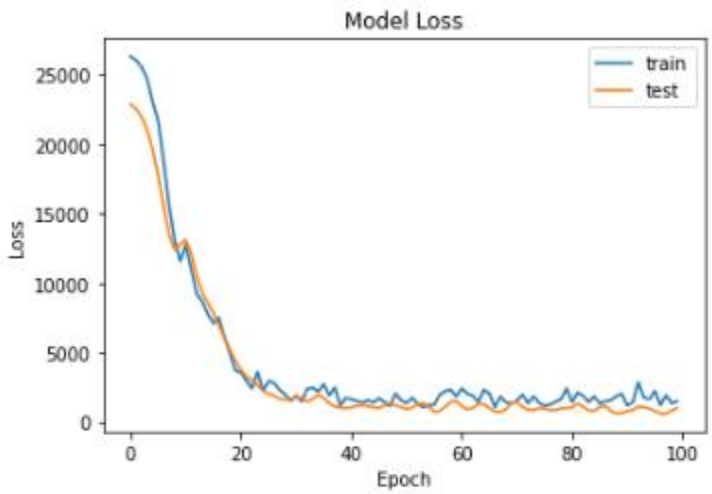
The training starts here and the mean absolute error is plotted along with the value mean absolute error.

```
history1 = model1.fit(X_egypt_train, Y_egypt_train, validation_data=(X_egypt_test,Y_egypt_test), epochs=100, batch_size=32)
plt.plot(history1.history['mean_absolute_error'])
plt.plot(history1.history['val_mean_absolute_error'])
plt.title('Model Accuracy')
plt.ylabel('Mean Absolute Error')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

train_mae_1 = history1.history['mean_absolute_error'][-1]
val_mae_1 = history1.history['val_mean_absolute_error'][-1]

print("Egypt :Mean Absolute Error: ", train_mae_1)
print("Egypt :Validation Mean Absolute Error: ", val_mae_1)
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_33 (Dense)             (None, 4000)              20000     
_________________________________________________________________
dropout_23 (Dropout)         (None, 4000)              0         
_________________________________________________________________
dense_34 (Dense)             (None, 400)               1600400   
_________________________________________________________________
dropout_24 (Dropout)         (None, 400)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 40)                16040     
_________________________________________________________________
dropout_25 (Dropout)         (None, 40)                0         
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 41        
=================================================================
Total params: 1,636,481
Trainable params: 1,636,481
Non-trainable params: 0
```

```
Epoch 89/100
96/96 [==============================] - 0s 1ms/step - loss: 1861.4242 - mean_absolute_error: 26.7726 - val_loss: 619.4811 - val_mean_absolu
Epoch 90/100
96/96 [==============================] - 0s 1ms/step - loss: 2039.7529 - mean_absolute_error: 26.1433 - val_loss: 642.5919 - val_mean_absolu
Epoch 91/100
96/96 [==============================] - 0s 1ms/step - loss: 1189.9366 - mean_absolute_error: 21.4275 - val_loss: 795.5047 - val_mean_absolu
Epoch 92/100
96/96 [==============================] - 0s 1ms/step - loss: 1438.8520 - mean_absolute_error: 22.4462 - val_loss: 883.9300 - val_mean_absolu
Epoch 93/100
96/96 [==============================] - 0s 1ms/step - loss: 2861.7546 - mean_absolute_error: 27.0600 - val_loss: 1111.9863 - val_mean_absol
Epoch 94/100
96/96 [==============================] - 0s 1ms/step - loss: 1764.8725 - mean_absolute_error: 25.0496 - val_loss: 1074.6169 - val_mean_absol
Epoch 95/100
96/96 [==============================] - 0s 1ms/step - loss: 1646.4216 - mean_absolute_error: 22.5753 - val_loss: 968.7142 - val_mean_absolu
Epoch 96/100
96/96 [==============================] - 0s 1ms/step - loss: 2244.7614 - mean_absolute_error: 24.3908 - val_loss: 793.2612 - val_mean_absolu
Epoch 97/100
96/96 [==============================] - 0s 1ms/step - loss: 1219.8595 - mean_absolute_error: 21.5102 - val_loss: 609.4415 - val_mean_absolu
Epoch 98/100
96/96 [==============================] - 0s 1ms/step - loss: 1931.0536 - mean_absolute_error: 24.2556 - val_loss: 607.3083 - val_mean_absolu
Epoch 99/100
96/96 [==============================] - 0s 1ms/step - loss: 1358.8926 - mean_absolute_error: 22.6146 - val_loss: 820.1610 - val_mean_absolu
Epoch 100/100
96/96 [==============================] - 0s 1ms/step - loss: 1492.0078 - mean_absolute_error: 22.6435 - val_loss: 994.8509 - val_mean_absolu
```




The optimal model is determined by the least difference between the mean absolute error and the validation mean absolute error, and in this case, this model this model turned out to be the best during our evaluation.

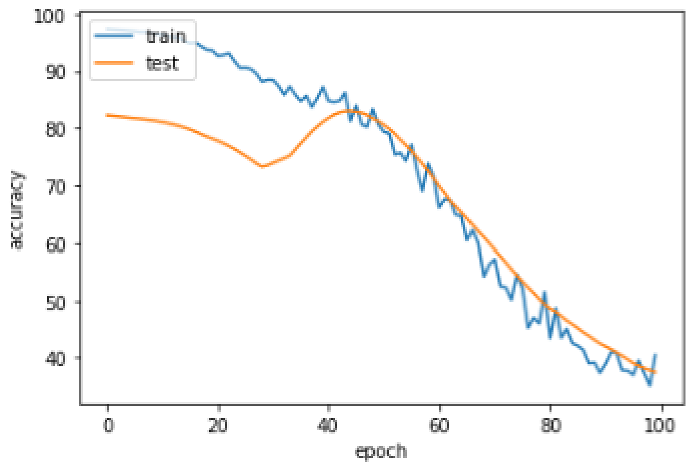
```
Egypt :Mean Absolute Error:  22.643454
Egypt :Validation Mean Absolute Error:  21.47761344909668
```

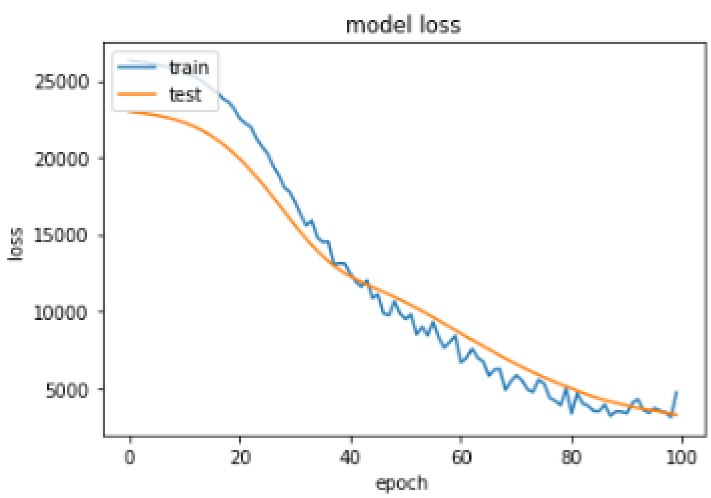
This is the second model we used. The first hidden layer has 400 neurons, 40 neurons on the next, and finally, the last layer has just one neuron. At every layer, we used an L2 regularizer with a Rectified Linear Unit (ReLU) function and a dropout with value 0.3 to reduce overfitting and to optimize the cost function. The loss function used is a Mean Squared Error (MSE). However, in the final layer, we used a linear function instead of a ReLU one.

```
def create_model2(X_train):
    # create model
    model1 = Sequential()

    model1.add(Dense(400, input_dim=X_train.shape[1], activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(40, activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(1,activation='linear',kernel_regularizer=regularizers.l2(1e-4)))

    model1.compile(optimizer ='adam', loss = 'mean_squared_error',
              metrics =[metrics.mae])

    return model1
model2 = create_model2(X_egypt_train)
model2.summary()
```

```
history2 = model2.fit(X_egypt_train, Y_egypt_train, validation_data=(X_egypt_test,Y_egypt_test), epochs=100, batch_size=32)
plt.plot(history2.history['mean_absolute_error'])
plt.plot(history2.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

train_mae_1 = history2.history['mean_absolute_error'][-1]
val_mae_1 = history2.history['val_mean_absolute_error'][-1]

print("Egypt :Mean Absolute Error: ", train_mae_1)
print("Egypt :Validation Mean Absolute Error: ", val_mae_1)
```

```
96/96 [==============================] - 0s 149us/step - loss: 3924.6633 - mean_absolute_error: 41.3842 - val_loss: 4214.0864 - val_mean_abs
Epoch 88/100
96/96 [==============================] - 0s 171us/step - loss: 3192.4777 - mean_absolute_error: 39.0194 - val_loss: 4115.6763 - val_mean_abs
Epoch 89/100
96/96 [==============================] - 0s 141us/step - loss: 3477.2100 - mean_absolute_error: 39.1407 - val_loss: 4028.8643 - val_mean_abs
Epoch 90/100
96/96 [==============================] - 0s 150us/step - loss: 3458.6314 - mean_absolute_error: 37.3775 - val_loss: 3947.1113 - val_mean_abs
Epoch 91/100
96/96 [==============================] - 0s 134us/step - loss: 3355.3369 - mean_absolute_error: 38.9178 - val_loss: 3854.0649 - val_mean_abs
Epoch 92/100
96/96 [==============================] - 0s 133us/step - loss: 4022.7564 - mean_absolute_error: 41.0426 - val_loss: 3763.3491 - val_mean_abs
Epoch 93/100
96/96 [==============================] - 0s 149us/step - loss: 4273.5872 - mean_absolute_error: 40.8850 - val_loss: 3687.3840 - val_mean_abs
Epoch 94/100
96/96 [==============================] - 0s 156us/step - loss: 3563.9448 - mean_absolute_error: 37.8635 - val_loss: 3631.7832 - val_mean_abs
Epoch 95/100
96/96 [==============================] - 0s 140us/step - loss: 3378.4758 - mean_absolute_error: 37.7896 - val_loss: 3570.1135 - val_mean_abs
Epoch 96/100
96/96 [==============================] - 0s 137us/step - loss: 3688.5472 - mean_absolute_error: 37.0047 - val_loss: 3528.7161 - val_mean_abs
Epoch 97/100
96/96 [==============================] - 0s 165us/step - loss: 3518.6310 - mean_absolute_error: 39.5166 - val_loss: 3458.7183 - val_mean_abs
Epoch 98/100
96/96 [==============================] - 0s 137us/step - loss: 3405.4661 - mean_absolute_error: 37.2293 - val_loss: 3391.5835 - val_mean_abs
Epoch 99/100
96/96 [==============================] - 0s 151us/step - loss: 3092.0938 - mean_absolute_error: 35.1418 - val_loss: 3323.9485 - val_mean_abs
Epoch 100/100
96/96 [==============================] - 0s 143us/step - loss: 4707.3357 - mean_absolute_error: 40.4603 - val_loss: 3256.5901 - val_mean_abs
```



As shown below, the difference between the MEA and the VMEA is higher than the previous model. Therefore, the previous model was better.

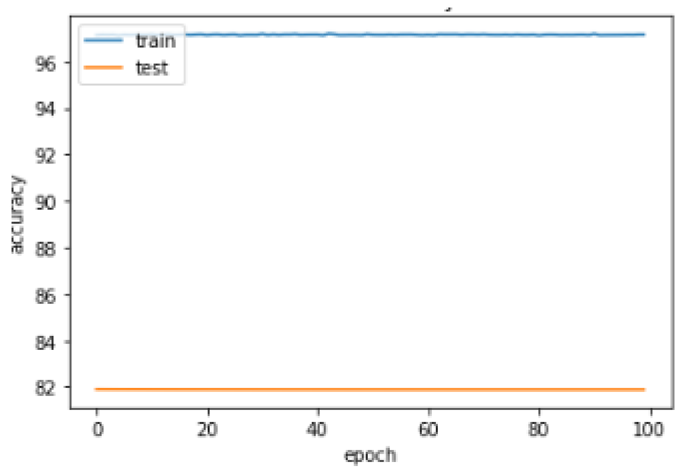
```
Egypt :Mean Absolute Error:  40.460308
Egypt :Validation Mean Absolute Error:  37.511749267578125
```

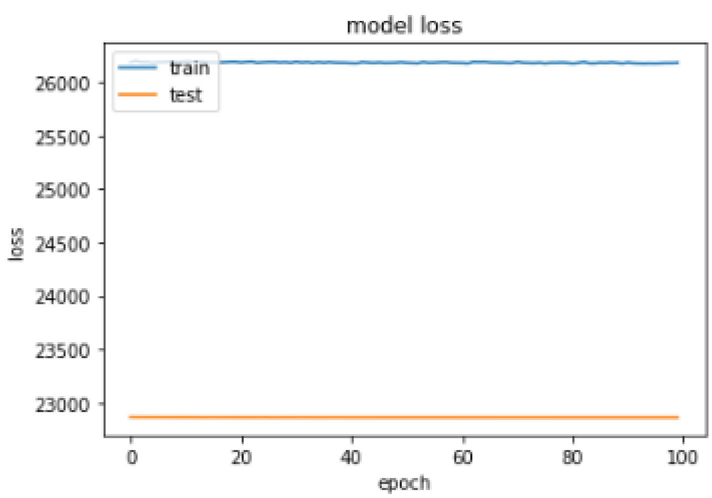
This is the third model we used, where the input takes all the factors mentioned above. The first hidden layer has 4 neurons, 4 neurons on the next, the last layer has just one neuron. This model takes a different approach because the weights are initialized with 1s and the bias with 0s. At every layer, we used an L2 regularizer with a Rectified Linear Unit (ReLU) function and a dropout with value 0.3 to reduce overfitting and to optimize the cost function. The loss function used is a Mean Squared Error (MSE). However, in the final layer, we used a sigmoid function instead of a ReLU one.

```python
def create_model3(X_train):
    # create model
    model1 = Sequential()

    model1.add(Dense(4, input_dim=X_train.shape[1], activation='relu',kernel_regularizer=regularizers.l2(1e-4),weights=[np.ones((4,4)),np.zeros(4)]))
    model1.add(Dropout(0.3))
    model1.add(Dense(4, activation='relu',kernel_regularizer=regularizers.l2(1e-4)))
    model1.add(Dropout(0.3))
    model1.add(Dense(1,activation='sigmoid',kernel_regularizer=regularizers.l2(1e-4)))

    model1.compile(optimizer ='adam', loss = 'mean_squared_error',
            metrics =[metrics.mae])

    return model1
model3 = create_model3(X_egypt_train)
model3.summary()
```

```python
history3 = model3.fit(X_egypt_train, Y_egypt_train, validation_data=(X_egypt_test,Y_egypt_test), epochs=100, batch_size=32)
plt.plot(history3.history['mean_absolute_error'])
plt.plot(history3.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

train_mae_1 = history3.history['mean_absolute_error'][-1]
val_mae_1 = history3.history['val_mean_absolute_error'][-1]

print("Egypt :Mean Absolute Error: ", train_mae_1)
print("Egypt :Validation Mean Absolute Error: ", val_mae_1)
```

```
96/96 [==============================] - 0s 97us/step - loss: 26180.8730 - mean_absolute_error: 97.1608 - val_loss: 22862.7617 - val_mean_ab
Epoch 90/100
96/96 [==============================] - 0s 107us/step - loss: 26175.8219 - mean_absolute_error: 97.1425 - val_loss: 22862.7598 - val_mean_a
Epoch 91/100
96/96 [==============================] - 0s 106us/step - loss: 26183.8340 - mean_absolute_error: 97.1886 - val_loss: 22862.7578 - val_mean_a
Epoch 92/100
96/96 [==============================] - 0s 105us/step - loss: 26177.8880 - mean_absolute_error: 97.1377 - val_loss: 22862.7539 - val_mean_a
Epoch 93/100
96/96 [==============================] - 0s 101us/step - loss: 26176.5840 - mean_absolute_error: 97.1431 - val_loss: 22862.7539 - val_mean_a
Epoch 94/100
96/96 [==============================] - 0s 108us/step - loss: 26175.4993 - mean_absolute_error: 97.1451 - val_loss: 22862.7480 - val_mean_a
Epoch 95/100
96/96 [==============================] - 0s 134us/step - loss: 26176.9961 - mean_absolute_error: 97.1465 - val_loss: 22862.7480 - val_mean_a
Epoch 96/100
96/96 [==============================] - 0s 107us/step - loss: 26175.5150 - mean_absolute_error: 97.1479 - val_loss: 22862.7441 - val_mean_a
Epoch 97/100
96/96 [==============================] - 0s 112us/step - loss: 26177.9355 - mean_absolute_error: 97.1572 - val_loss: 22862.7441 - val_mean_a
Epoch 98/100
96/96 [==============================] - 0s 163us/step - loss: 26179.5980 - mean_absolute_error: 97.1502 - val_loss: 22862.7422 - val_mean_a
Epoch 99/100
96/96 [==============================] - 0s 153us/step - loss: 26179.3646 - mean_absolute_error: 97.1640 - val_loss: 22862.7402 - val_mean_a
Epoch 100/100
96/96 [==============================] - 0s 120us/step - loss: 26182.2331 - mean_absolute_error: 97.1624 - val_loss: 22862.7363 - val_mean_a
```



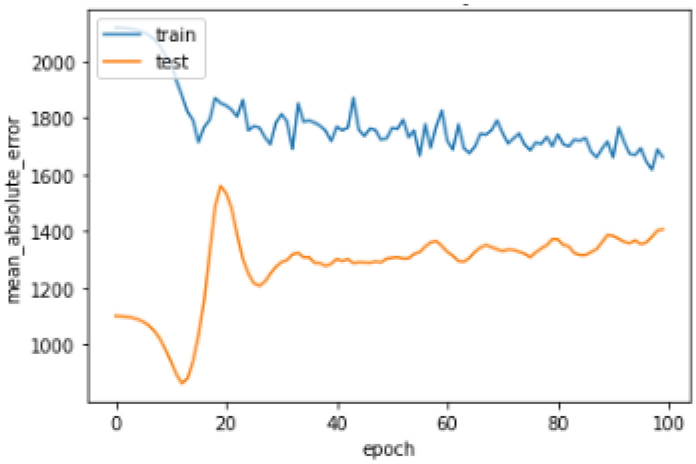
This model produced the least optimal results out of all the models mentioned above due to the high difference between the MEA and the VMEA.

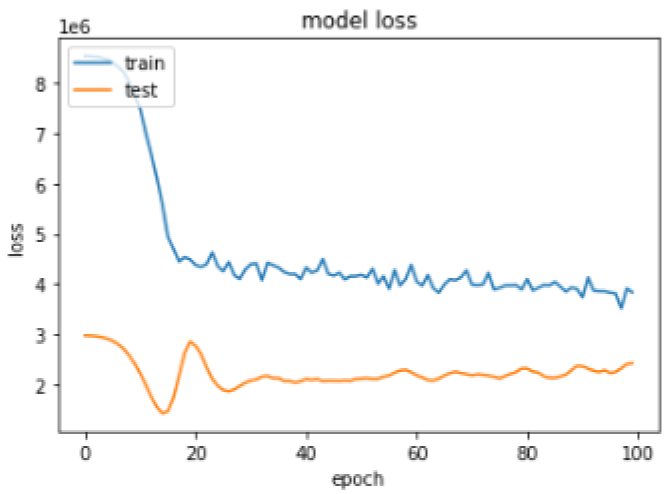
```
Egypt :Mean Absolute Error:  97.162415
Egypt :Validation Mean Absolute Error:  81.88937377929688
```

The first model was applied on data retrieved from Italy.

```
#italy
model4 = create_model1(X_italy_train)
model4.summary()

history4 = model4.fit(X_italy_train, Y_italy_train, validation_data=(X_italy_test,Y_italy_test), epochs=100, batch_size=32)
plt.plot(history4.history['mean_absolute_error'])
plt.plot(history4.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('mean_absolute_error')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
train_mae_1 = history4.history['mean_absolute_error'][-1]
val_mae_1 = history4.history['val_mean_absolute_error'][-1]

print("Italy :Mean Absolute Error: ", train_mae_1)
print("Italy :Validation Mean Absolute Error: ", val_mae_1)
```

```
Epoch 89/100
96/96 [==============================] - 0s 1ms/step - loss: 3927417.7500 - mean_absolute_error: 1690.7533 - val_loss: 2277259.2500 - val_
Epoch 90/100
96/96 [==============================] - 0s 1ms/step - loss: 3899769.0833 - mean_absolute_error: 1715.8385 - val_loss: 2362645.5000 - val_
Epoch 91/100
96/96 [==============================] - 0s 1ms/step - loss: 3734728.2500 - mean_absolute_error: 1660.0221 - val_loss: 2350962.5000 - val_
Epoch 92/100
96/96 [==============================] - 0s 1ms/step - loss: 4128252.8333 - mean_absolute_error: 1765.0073 - val_loss: 2307896.0000 - val_
Epoch 93/100
96/96 [==============================] - 0s 1ms/step - loss: 3864050.6667 - mean_absolute_error: 1713.9388 - val_loss: 2262891.0000 - val_
Epoch 94/100
96/96 [==============================] - 0s 1ms/step - loss: 3850488.9167 - mean_absolute_error: 1672.6683 - val_loss: 2241040.7500 - val_
Epoch 95/100
96/96 [==============================] - 0s 1ms/step - loss: 3850095.2500 - mean_absolute_error: 1669.0220 - val_loss: 2275169.0000 - val_
Epoch 96/100
96/96 [==============================] - 0s 1ms/step - loss: 3820547.5833 - mean_absolute_error: 1692.3424 - val_loss: 2220092.2500 - val_
Epoch 97/100
96/96 [==============================] - 0s 1ms/step - loss: 3806414.0833 - mean_absolute_error: 1645.5376 - val_loss: 2240912.2500 - val_
Epoch 98/100
96/96 [==============================] - 0s 1ms/step - loss: 3511325.8333 - mean_absolute_error: 1616.7386 - val_loss: 2311586.2500 - val_
Epoch 99/100
96/96 [==============================] - 0s 1ms/step - loss: 3910329.2500 - mean_absolute_error: 1688.1134 - val_loss: 2398092.5000 - val_
Epoch 100/100
96/96 [==============================] - 0s 1ms/step - loss: 3828652.0000 - mean_absolute_error: 1660.9160 - val_loss: 2412420.0000 - val_
```



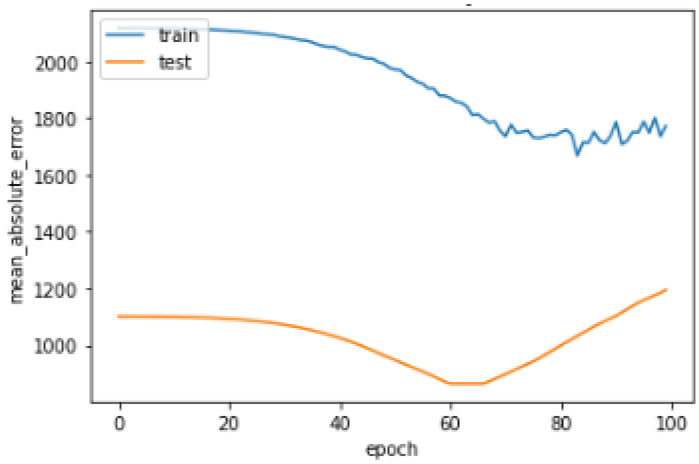



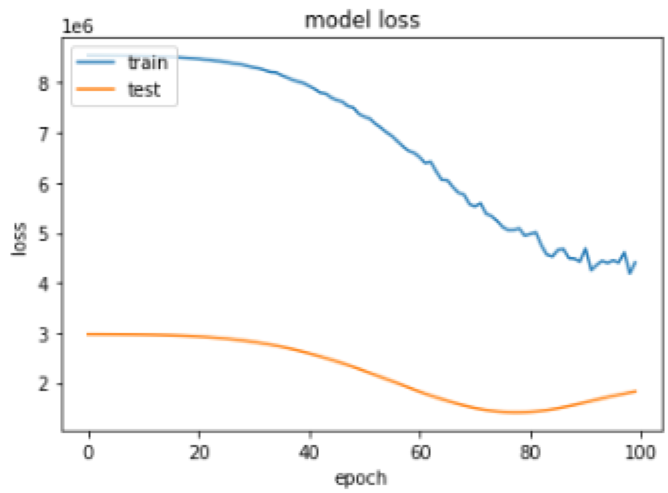

```
Italy :Mean Absolute Error:  1660.916
Italy :Validation Mean Absolute Error:  1406.138671875
```

The second model used on Italy's data. Albeit, the first model was better.

```
#italy
model4 = create_model2(X_italy_train)
model4.summary()

history4 = model4.fit(X_italy_train, Y_italy_train, validation_data=(X_italy_test,Y_italy_test), epochs=100, batch_size=32)
plt.plot(history4.history['mean_absolute_error'])
plt.plot(history4.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('mean_absolute_error')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
train_mae_1 = history4.history['mean_absolute_error'][-1]
val_mae_1 = history4.history['val_mean_absolute_error'][-1]

print("Italy :Mean Absolute Error: ", train_mae_1)
print("Italy :Validation Mean Absolute Error: ", val_mae_1)
```

```
96/96 [==============================] - 0s 155us/step - loss: 4481754.3333 - mean_absolute_error: 1711.4432 - val_loss: 1558346.1250 - val_
Epoch 90/100
96/96 [==============================] - 0s 149us/step - loss: 4420541.1667 - mean_absolute_error: 1737.2152 - val_loss: 1583700.8750 - val_
Epoch 91/100
96/96 [==============================] - 0s 148us/step - loss: 4681002.5833 - mean_absolute_error: 1785.7289 - val_loss: 1608449.6250 - val_
Epoch 92/100
96/96 [==============================] - 0s 121us/step - loss: 4249491.8333 - mean_absolute_error: 1708.0820 - val_loss: 1635978.5000 - val_
Epoch 93/100
96/96 [==============================] - 0s 138us/step - loss: 4354570.6667 - mean_absolute_error: 1719.6051 - val_loss: 1662110.3750 - val_
Epoch 94/100
96/96 [==============================] - 0s 127us/step - loss: 4438614.1667 - mean_absolute_error: 1750.8751 - val_loss: 1688264.6250 - val_
Epoch 95/100
96/96 [==============================] - 0s 161us/step - loss: 4396349.1667 - mean_absolute_error: 1749.7775 - val_loss: 1714496.1250 - val_
Epoch 96/100
96/96 [==============================] - 0s 126us/step - loss: 4450088.5833 - mean_absolute_error: 1787.6769 - val_loss: 1738595.7500 - val_
Epoch 97/100
96/96 [==============================] - 0s 121us/step - loss: 4402674.7500 - mean_absolute_error: 1748.3428 - val_loss: 1760357.5000 - val_
Epoch 98/100
96/96 [==============================] - 0s 119us/step - loss: 4605226.8333 - mean_absolute_error: 1801.0975 - val_loss: 1783160.0000 - val_
Epoch 99/100
96/96 [==============================] - 0s 122us/step - loss: 4179577.9167 - mean_absolute_error: 1736.3628 - val_loss: 1806716.0000 - val_
Epoch 100/100
96/96 [==============================] - 0s 143us/step - loss: 4405646.1667 - mean_absolute_error: 1771.9838 - val_loss: 1828518.2500 - val_
```



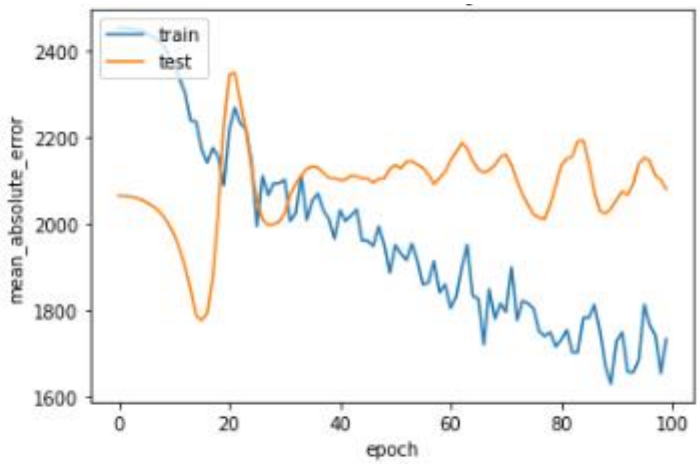



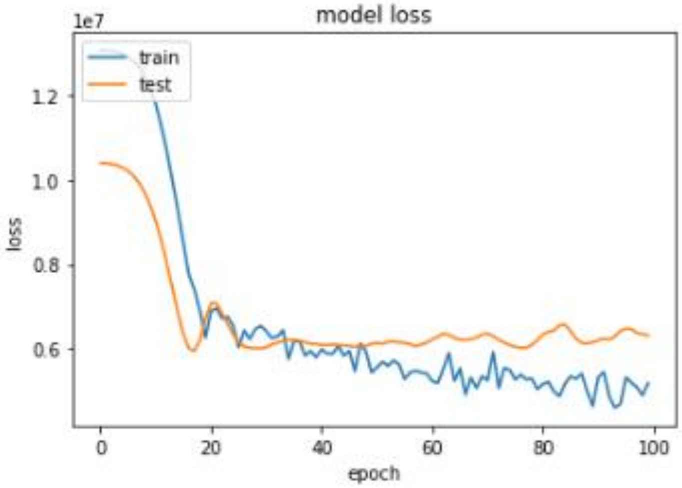

```
Italy :Mean Absolute Error:  1771.9838
Italy :Validation Mean Absolute Error:  1194.79931640625
```

The first model used on Spain's data.

```
#spain
model5 = create_model1(X_spain_train)
model5.summary()

history5 = model5.fit(X_spain_train, Y_spain_train, validation_data=(X_spain_test,Y_spain_test), epochs=100, batch_size=32)
plt.plot(history5.history['mean_absolute_error'])
plt.plot(history5.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('mean_absolute_error')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history5.history['loss'])
plt.plot(history5.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
train_mae_1 = history5.history['mean_absolute_error'][-1]
val_mae_1 = history5.history['val_mean_absolute_error'][-1]

print("Spain :Mean Absolute Error: ", train_mae_1)
print("Spain :Validation Mean Absolute Error: ", val_mae_1)
```

```
96/96 [==============================] - 0s 1ms/step - loss: 4982606.6667 - mean_absolute_error: 1669.4586 - val_loss: 6119982.0000 - val_
Epoch 90/100
96/96 [==============================] - 0s 1ms/step - loss: 4634621.8333 - mean_absolute_error: 1628.9269 - val_loss: 6148547.0000 - val_
Epoch 91/100
96/96 [==============================] - 0s 1ms/step - loss: 5318962.5000 - mean_absolute_error: 1727.3905 - val_loss: 6193061.0000 - val_
Epoch 92/100
96/96 [==============================] - 0s 1ms/step - loss: 5443781.1667 - mean_absolute_error: 1748.2072 - val_loss: 6238409.5000 - val_
Epoch 93/100
96/96 [==============================] - 0s 1ms/step - loss: 4854700.5000 - mean_absolute_error: 1656.6305 - val_loss: 6212611.5000 - val_
Epoch 94/100
96/96 [==============================] - 0s 1ms/step - loss: 4599371.9167 - mean_absolute_error: 1655.2726 - val_loss: 6280548.0000 - val_
Epoch 95/100
96/96 [==============================] - 0s 1ms/step - loss: 4690350.1667 - mean_absolute_error: 1684.4059 - val_loss: 6421337.0000 - val_
Epoch 96/100
96/96 [==============================] - 0s 1ms/step - loss: 5315533.5000 - mean_absolute_error: 1812.2451 - val_loss: 6471478.0000 - val_
Epoch 97/100
96/96 [==============================] - 0s 1ms/step - loss: 5189225.6667 - mean_absolute_error: 1765.4447 - val_loss: 6459458.0000 - val_
Epoch 98/100
96/96 [==============================] - 0s 1ms/step - loss: 5076890.6667 - mean_absolute_error: 1741.4613 - val_loss: 6367289.0000 - val_
Epoch 99/100
96/96 [==============================] - 0s 1ms/step - loss: 4887854.8333 - mean_absolute_error: 1653.5142 - val_loss: 6347494.5000 - val_
Epoch 100/100
96/96 [==============================] - 0s 1ms/step - loss: 5179847.5833 - mean_absolute_error: 1731.9385 - val_loss: 6304665.0000 - val_
```



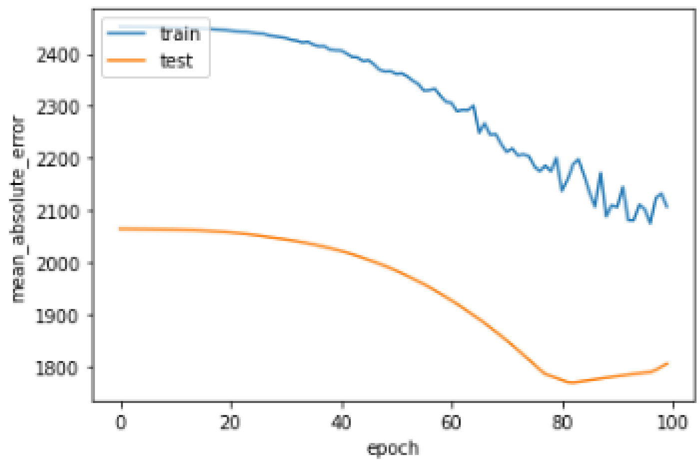



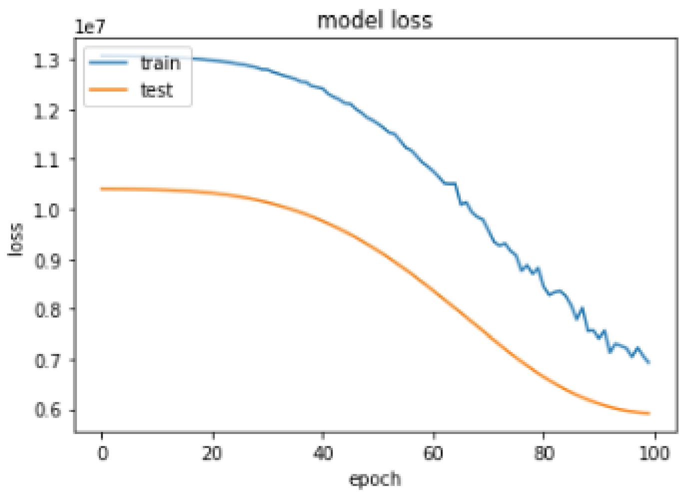

```
Spain :Mean Absolute Error:  1731.9385
Spain :Validation Mean Absolute Error:  2079.45458984375
```

The second model used on Spain's data. In this case, the second model produced better results than the first model, however, the difference between the two models was minimal.

```
#spain
model5 = create_model2(X_spain_train)
model5.summary()

history5 = model5.fit(X_spain_train, Y_spain_train, validation_data=(X_spain_test,Y_spain_test), epochs=100, batch_size=32)#
plt.plot(history5.history['mean_absolute_error'])
plt.plot(history5.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('mean_absolute_error')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history5.history['loss'])
plt.plot(history5.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
train_mae_1 = history5.history['mean_absolute_error'][-1]
val_mae_1 = history5.history['val_mean_absolute_error'][-1]

print("Spain :Mean Absolute Error: ", train_mae_1)
print("Spain :Validation Mean Absolute Error: ", val_mae_1)
```

```
96/96 [==============================] - 0s 152us/step - loss: 7409904.8333 - mean_absolute_error: 2106.3464 - val_loss: 6105362.0000 - val_
Epoch 92/100
96/96 [==============================] - 0s 128us/step - loss: 7562783.8333 - mean_absolute_error: 2145.4824 - val_loss: 6069977.0000 - val_
Epoch 93/100
96/96 [==============================] - 0s 123us/step - loss: 7128449.3333 - mean_absolute_error: 2081.3083 - val_loss: 6038578.5000 - val_
Epoch 94/100
96/96 [==============================] - 0s 166us/step - loss: 7306234.1667 - mean_absolute_error: 2081.0413 - val_loss: 6010859.5000 - val_
Epoch 95/100
96/96 [==============================] - 0s 132us/step - loss: 7261628.6667 - mean_absolute_error: 2111.4841 - val_loss: 5986213.0000 - val_
Epoch 96/100
96/96 [==============================] - 0s 152us/step - loss: 7215751.0000 - mean_absolute_error: 2102.2136 - val_loss: 5965582.0000 - val_
Epoch 97/100
96/96 [==============================] - 0s 127us/step - loss: 7044130.3333 - mean_absolute_error: 2075.0784 - val_loss: 5948670.0000 - val_
Epoch 98/100
96/96 [==============================] - 0s 151us/step - loss: 7226781.3333 - mean_absolute_error: 2123.8245 - val_loss: 5934349.5000 - val_
Epoch 99/100
96/96 [==============================] - 0s 169us/step - loss: 7063385.6667 - mean_absolute_error: 2132.5342 - val_loss: 5922743.5000 - val_
Epoch 100/100
96/96 [==============================] - 0s 120us/step - loss: 6930390.8333 - mean_absolute_error: 2107.3640 - val_loss: 5914531.5000 - val_
```




```
Spain :Mean Absolute Error:  2107.364
Spain :Validation Mean Absolute Error:  1805.109375
```

This model uses the total number of cases, where data is retrieved from Egypt, Germany, Italy, UK, Spain, Turkey, and USA. The average maximum and the minimum temperature in every country is computed over 114 days, which is the period since the start of the pandemic. Population is also another factor considered along with the latitude and longitude of every country. As for the economical factors, the inflation rate and the Gross Domestic Product (GDP) in USD billions were also added. Educational factors such as the number of schools, universities, and a percentage of the number of tertiary students along with other students preceding the tertiary stage.

**Model Class Two : Total Number of Cases**

```
[ ] countries_list = [54, 70, 93, 182, 163, 180, 183]
    countries_list.sort()
    df_2 = df.copy()
    df_2 = df.loc[countries_list]
```

```
[ ] countries_list_names = ['Egypt','Germany','Italy','UK','Spain','Turkey','USA']
    maxtempC_list = []
    mintempC_list = []

    for i in range(len(countries_list)):
        path ="/content/temp/" + countries_list_names[i] + '.csv'
        new_factors = pd.read_csv(path)

        # country_row = list(new_factors.loc[i].values)
        maxtempC = list(new_factors['maxtempC'])
        mintempC = list(new_factors['mintempC'])

        avgmaxtempC = sum(maxtempC)/len(maxtempC)
        avgmintempC = sum(mintempC)/len(mintempC)

        maxtempC_list.append(avgmaxtempC)
        mintempC_list.append(avgmintempC)

    df_2.loc[:,'avgmaxtempC'] = maxtempC_list
    df_2.loc[:,'avgmintempC'] = mintempC_list
```

```
[ ] countries_population = [102103353, 83752482, 60472207, 46752659, 84231508, 67844072, 330773982]
    df_2.loc[:,'Population'] = countries_population
```

```
[ ] cols11 = [col for col in df_2.columns if col in ['Province/State','Country/Region','Long','Lat','avgmaxtempC','avgmintempC','Population','5/1
    df_2 = df_2[cols11]
```

```
[ ] df_2 = df_2[['Province/State', 'Country/Region','Long','Lat', 'avgmaxtempC', 'avgmintempC', 'Population','5/14/20']]
    df_2
```

| | Province/State | Country/Region | Long | Lat | avgmaxtempC | avgmintempC | Population | 5/14/20 |
|---|---|---|---|---|---|---|---|---|
| 54 | Egypt | 54 | 30.0000 | 26.0000 | 25.263158 | 15.324561 | 102103353 | 10829 |
| 70 | Germany | 70 | 9.0000 | 51.0000 | 11.473684 | 4.921053 | 83752482 | 174975 |
| 93 | Italy | 93 | 12.0000 | 43.0000 | 17.745614 | 10.710526 | 60472207 | 223096 |
| 163 | Spain | 163 | -4.0000 | 40.0000 | 12.885965 | 7.000000 | 46752659 | 272646 |
| 180 | Turkey | 180 | 35.2433 | 38.9637 | 17.421053 | 9.342105 | 84231508 | 144749 |
| 182 | UK | 182 | -3.4360 | 55.3781 | 12.570175 | 2.684211 | 67844072 | 233151 |
| 183 | USA | 183 | -101.2500 | 39.9090 | 16.728070 | 7.903509 | 330773982 | 1457593 |

```
[ ] # Economy Factors
    countries_GDP = [315.00,4110.00,2014.00,1500.00,813.81,2744,20140.00]
    countries_inflation_rate = [5.9,0.32,0.24,1.05,0.85,1.5,0.3]
```

```
[ ] df_2.loc[:,'GDP(Billion)'] = countries_GDP
    df_2.loc[:,'Inflation Rate (%)'] = countries_inflation_rate
```

```
[ ] # Educational factors
    countries_tertiary = [11.6,28.58,18.67,36.35,20.01,45.74,46.36]
    countries_before_tertirary = [88.36,71.42,81.33,63.65,79.99,54.25,53.64]
    number_of_universities = [20,380,90,76,180,106,1626]
    df_2.loc[:,'Tertiary (%)'] = countries_tertiary
    df_2.loc[:,'Before Tertiary (%)'] = countries_before_tertirary
    df_2.loc[:,'# of universities'] = number_of_universities
```

```
[ ] df_2
```

| | Province/State | Country/Region | Long | Lat | avgmaxtempC | avgmintempC | Population | 5/14/20 | GDP(Billion) | Inflation Rate (%) | Tertiary (%) | Before Tertiary (%) | # of universities |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 54 | Egypt | 54 | 30.0000 | 26.0000 | 25.263158 | 15.324561 | 102103353 | 10829 | 315.00 | 5.90 | 11.60 | 88.36 | 20 |
| 70 | Germany | 70 | 9.0000 | 51.0000 | 11.473684 | 4.921053 | 83752482 | 174975 | 4110.00 | 0.32 | 28.58 | 71.42 | 380 |
| 93 | Italy | 93 | 12.0000 | 43.0000 | 17.745614 | 10.710526 | 60472207 | 223096 | 2014.00 | 0.24 | 18.67 | 81.33 | 90 |
| 163 | Spain | 163 | -4.0000 | 40.0000 | 12.885965 | 7.000000 | 46752659 | 272646 | 1500.00 | 1.05 | 36.35 | 63.65 | 76 |
| 180 | Turkey | 180 | 35.2433 | 38.9637 | 17.421053 | 9.342105 | 84231508 | 144749 | 813.81 | 0.85 | 20.01 | 79.99 | 180 |
| 182 | UK | 182 | -3.4360 | 55.3781 | 12.570175 | 2.684211 | 67844072 | 233151 | 2744.00 | 1.50 | 45.74 | 54.25 | 106 |
| 183 | USA | 183 | -101.2500 | 39.9090 | 16.728070 | 7.903509 | 330773982 | 1457593 | 20140.00 | 0.30 | 46.36 | 53.64 | 1626 |

```
[ ] df_2 = df_2[['Province/State', 'Country/Region','Long','Lat', 'avgmaxtempC', 'avgmintempC', 'Population','GDP(Billion)','Inflation Rate (%)','Tertiary (%)','Before Te
    df_2
```

| | Province/State | Country/Region | Long | Lat | avgmaxtempC | avgmintempC | Population | GDP(Billion) | Inflation Rate (%) | Tertiary (%) | Before Tertiary (%) | # of universities | 5/14/20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 54 | Egypt | 54 | 30.0000 | 26.0000 | 25.263158 | 15.324561 | 102103353 | 315.00 | 5.90 | 11.60 | 88.36 | 20 | 10829 |
| 70 | Germany | 70 | 9.0000 | 51.0000 | 11.473684 | 4.921053 | 83752482 | 4110.00 | 0.32 | 28.58 | 71.42 | 380 | 174975 |
| 93 | Italy | 93 | 12.0000 | 43.0000 | 17.745614 | 10.710526 | 60472207 | 2014.00 | 0.24 | 18.67 | 81.33 | 90 | 223096 |
| 163 | Spain | 163 | -4.0000 | 40.0000 | 12.885965 | 7.000000 | 46752659 | 1500.00 | 1.05 | 36.35 | 63.65 | 76 | 272646 |
| 180 | Turkey | 180 | 35.2433 | 38.9637 | 17.421053 | 9.342105 | 84231508 | 813.81 | 0.85 | 20.01 | 79.99 | 180 | 144749 |
| 182 | UK | 182 | -3.4360 | 55.3781 | 12.570175 | 2.684211 | 67844072 | 2744.00 | 1.50 | 45.74 | 54.25 | 106 | 233151 |
| 183 | USA | 183 | -101.2500 | 39.9090 | 16.728070 | 7.903509 | 330773982 | 20140.00 | 0.30 | 46.36 | 53.64 | 1626 | 1457593 |

```
[ ] data = [col for col in df_2.columns if col in ['Country/Region','Long','Lat','avgmaxtempC','avgmintempC','Population','GDP(Billion)','Inflation Rate (%)','Tertiary (%
    X_total_cases = df_2[data]
    X_total_cases = X_total_cases.values
    Y_total_cases = df_2['5/14/20']
```

Data is scaled, and then split into five training examples and just two test examples.

```
from sklearn import preprocessing
print(X_total_cases.shape)
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X_total_cases)
```

```
(7, 11)
```

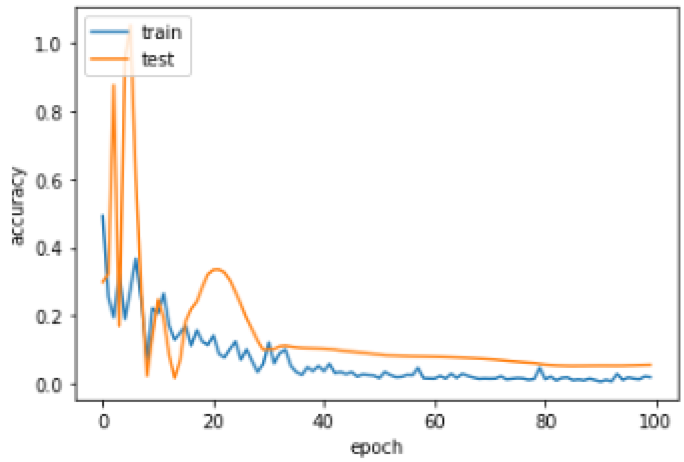
```
[ ] from sklearn.model_selection import train_test_split

    X_total_train, X_total_test, Y_total_train, Y_total_test = train_test_split(X_total_cases, Y_total_cases, test_size=0.15,shuffle=False)
    print(X_total_train.shape,X_total_test.shape,Y_total_train.shape,Y_total_test.shape)
```

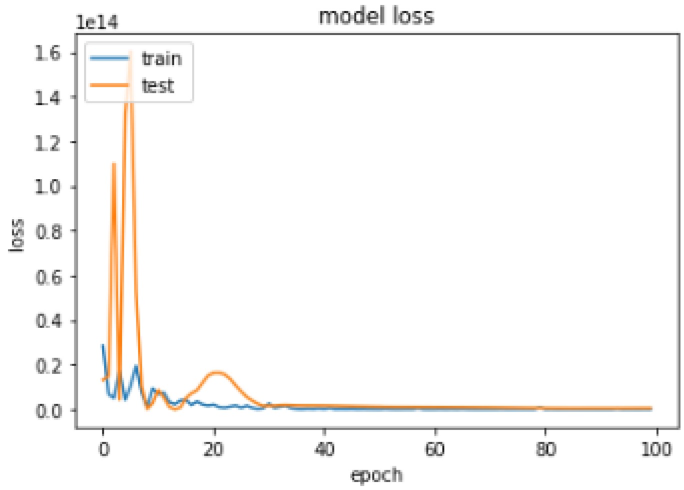
```
(5, 11) (2, 11) (5,) (2,)
```

The first model in the previous section is used here but with the new data, and as shown below, it yields the best results.

```
model1 = create_model1(X_total_train)
model1.summary()
history1 = model1.fit(X_total_train, Y_total_train, validation_data=(X_total_test,Y_total_test), epochs=100, batch_size=32)
plt.plot(history1.history['mean_absolute_error'])
plt.plot(history1.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
print("Mean Absolute Error: ", train_mae_1)
print("Validation Mean Absolute Error: ", val_mae_1)
```

```
Epoch 91/100
5/5 [==============================] - 0s 8ms/step - loss: 5475457024.0000 - mean_absolute_error: 62239.8516 - val_loss: 444928425984.0000 -
Epoch 92/100
5/5 [==============================] - 0s 7ms/step - loss: 14687540224.0000 - mean_absolute_error: 109565.0469 - val_loss: 445072867328.0000
Epoch 93/100
5/5 [==============================] - 0s 8ms/step - loss: 7509792256.0000 - mean_absolute_error: 68407.2266 - val_loss: 443652636672.0000 -
Epoch 94/100
5/5 [==============================] - 0s 7ms/step - loss: 189126066176.0000 - mean_absolute_error: 282711.0000 - val_loss: 442720157696.000
Epoch 95/100
5/5 [==============================] - 0s 6ms/step - loss: 15077697536.0000 - mean_absolute_error: 107703.6484 - val_loss: 444322250752.0000
Epoch 96/100
5/5 [==============================] - 0s 8ms/step - loss: 45142089728.0000 - mean_absolute_error: 179914.6875 - val_loss: 453325225984.0000
Epoch 97/100
5/5 [==============================] - 0s 8ms/step - loss: 24680148992.0000 - mean_absolute_error: 146820.9062 - val_loss: 460525666304.0000
Epoch 98/100
5/5 [==============================] - 0s 8ms/step - loss: 22221668352.0000 - mean_absolute_error: 128268.7031 - val_loss: 471466541056.0000
Epoch 99/100
5/5 [==============================] - 0s 7ms/step - loss: 55234371584.0000 - mean_absolute_error: 209502.2188 - val_loss: 479362154496.0000
Epoch 100/100
5/5 [==============================] - 0s 7ms/step - loss: 35284361216.0000 - mean_absolute_error: 185756.2812 - val_loss: 486861799424.0000
```



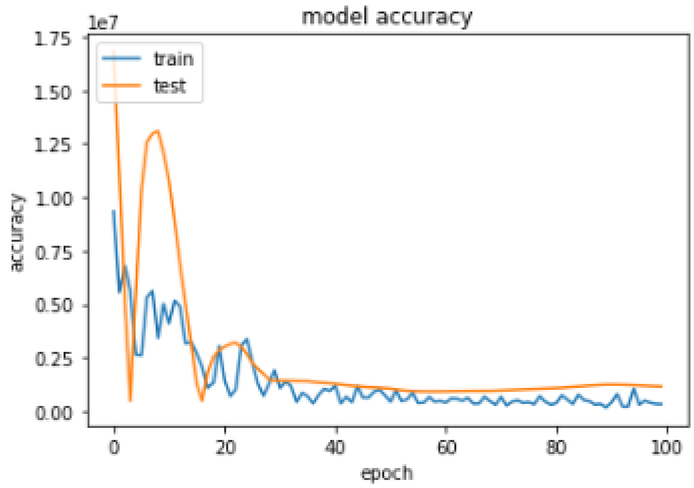



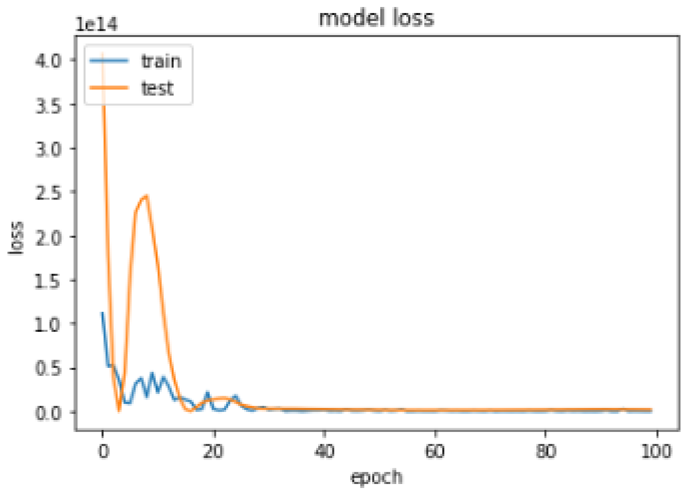

```
Mean Absolute Error:  2107.364
Validation Mean Absolute Error:  1805.109375
```

This is the second model from the previous section used with the new data, however, its results are far from optimal.

```
model2 = create_model2(X_total_train)
model2.summary()

history2 = model2.fit(X_total_train, Y_total_train, validation_data=(X_total_test,Y_total_test), epochs=100, batch_size=32)
plt.plot(history2.history['mean_absolute_error'])
plt.plot(history2.history['val_mean_absolute_error'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

train_mae_1 = history2.history['mean_absolute_error'][-1]
val_mae_1 = history2.history['val_mean_absolute_error'][-1]

print("Mean Absolute Error: ", train_mae_1)
print("Validation Mean Absolute Error: ", val_mae_1)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 1209622200320.0000 - mean_absolute_error: 796270.9375 - val_loss: 2271021891584.0
Epoch 93/100
5/5 [==============================] - 0s 831us/step - loss: 46985060352.0000 - mean_absolute_error: 211494.4688 - val_loss: 2247906557952.0
Epoch 94/100
5/5 [==============================] - 0s 1ms/step - loss: 48329760768.0000 - mean_absolute_error: 212161.8438 - val_loss: 2225832198144.000
Epoch 95/100
5/5 [==============================] - 0s 1ms/step - loss: 2831841755136.0000 - mean_absolute_error: 1049853.2500 - val_loss: 2181789384704.
Epoch 96/100
5/5 [==============================] - 0s 1ms/step - loss: 129080868864.0000 - mean_absolute_error: 293665.7812 - val_loss: 2141068460032.00
Epoch 97/100
5/5 [==============================] - 0s 1ms/step - loss: 378078560256.0000 - mean_absolute_error: 482743.4062 - val_loss: 2097664229376.00
Epoch 98/100
5/5 [==============================] - 0s 1ms/step - loss: 301411926016.0000 - mean_absolute_error: 386802.5625 - val_loss: 2057623306240.00
Epoch 99/100
5/5 [==============================] - 0s 1ms/step - loss: 164738695168.0000 - mean_absolute_error: 335157.3125 - val_loss: 2018521907200.00
Epoch 100/100
5/5 [==============================] - 0s 1ms/step - loss: 163860234240.0000 - mean_absolute_error: 322740.8125 - val_loss: 1976327471104.00
```




```
Mean Absolute Error:  322740.8
Validation Mean Absolute Error:  1147803.5
```

# Remarks

- Keras was used for the neural network because its an open source library in Python.
- L2 regularizer and dropout were used to reduce overfitting in our model.
- The Rectified Linear Unit (reLU) function produces much better results than the sigmoid function because reLu does not have a vanishing gradient.
- Data retrieval was one of the challenging stages in this project, especially in determining which factors were important and which were not.
- Another issue faced in this project is that the data-retrieval process was initially halted on 1 May. This presented a difficulty for us in finding some progress with our results.
- The addition of data until 14 May improved the performance of the model.