DengIA Project

Roumaissa OMARI / Fadi EL CHEIKH TAHA / 27.04.2021

Table of contents

Note to readers : we have made our article as simple as possible. If you are interested only by the results of our algorithms, please pay attention to the boxes (□) arranged throughout the article.

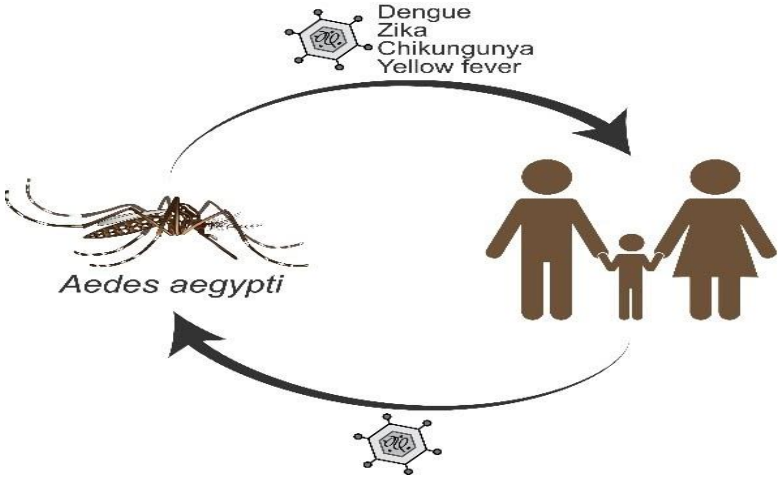
INTRODUCTION

Git repo : https://github.com/HubHetic/DengAI

Machine learning prediction models using time-series weather data.

What is Dengue?

Dengue is a mosquito-borne disease that occurs in tropical and subtropical regions of the world. In mild cases, symptoms are similar to the flu: fever, rash, and muscle and joint pain. In severe cases, dengue can cause severe bleeding, low blood pressure, and even death.
Dengue fever occurs mainly throughout the intertropical zone. According to current OMS estimates, there may be 50 to 100 million cases worldwide each year.



The Aedes Aegypti mosquito carries many viruses, including Dengue fever, a disease that threatens between 1/3 and 1/2 of the world's 7.5 billion people.
Because mosquitoes carry it, the transmission dynamics of dengue are related to climate variables such as temperature and precipitation. Although the relationship to climate is complex, a growing number of scientists argue that climate change is likely to produce distributional shifts that will have significant public health implications worldwide.

In recent years dengue fever has been spreading. Historically, the disease has been most prevalent in Southeast Asia and the Pacific islands. These days many of the nearly half-billion cases per year are occurring in Latin America.

GETTING THE DATA

https://www.drivendata.org/competitions/44/dengai-predicting-disease-spread/

Data was given by the Driven Data Competition as CSV including two sources :

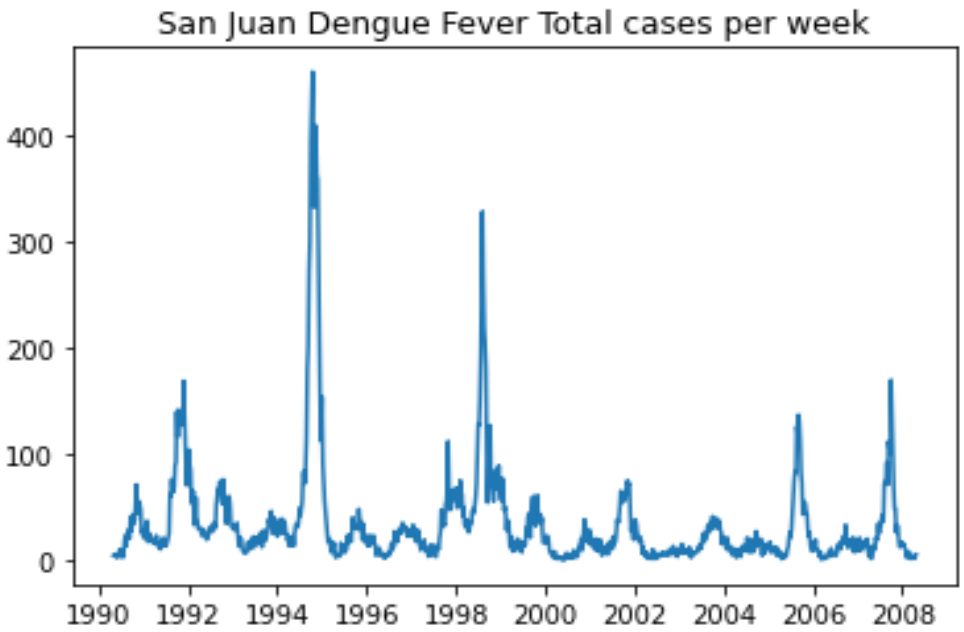
- Environmental data collected by various U.S. Federal Government agencies—from the Centers for Disease Control and Prevention to the National Oceanic

- Atmospheric Administration in the U.S. Department of Commerce.

Our goal

Our goal was to create a machine learning model able to accurately predict the number of weekly cases of Dengue that will occur at two locations : San Juan in Puerto Rico, and Iquitos in Peru.

Visualise the weekly reported cases in San Juan



- We can see that the time series has seasonality. Seasonality refers to a periodic pattern within years related to the calendar day, month, quarter, etc...

- We can see that the time series does not appear to have a trend. There is no long-run upward or downward direction in the series.

PREPROCESSING

The Data in this project is referred to as a time series. Time-series Data requires considerations and particular pre-processing for Machine Learning. Missing Values has to be handled carefully before fitting any predictive model. Here is a view of the variables of our data set, and their missing values.

Your selected dataframe has 20 columns.
There are 20 columns that have missing values.

	Missing Values	% of Total Values
ndvi_ne	191	20.41
ndvi_nw	49	5.24
ndvi_se	19	2.03
ndvi_sw	19	2.03
reanalysis_sat_precip_amt_mm	9	0.96
precipitation_amt_mm	9	0.96
station_min_temp_c	6	0.64
station_max_temp_c	6	0.64
station_diur_temp_rng_c	6	0.64
station_avg_temp_c	6	0.64
reanalysis_tdtr_k	6	0.64
reanalysis_specific_humidity_g_per_kg	6	0.64
reanalysis_precip_amt_kg_per_m2	6	0.64
reanalysis_relative_humidity_percent	6	0.64
reanalysis_min_air_temp_k	6	0.64
reanalysis_max_air_temp_k	6	0.64
reanalysis_dew_point_temp_k	6	0.64
reanalysis_avg_temp_k	6	0.64
reanalysis_air_temp_k	6	0.64
station_precip_mm	6	0.64

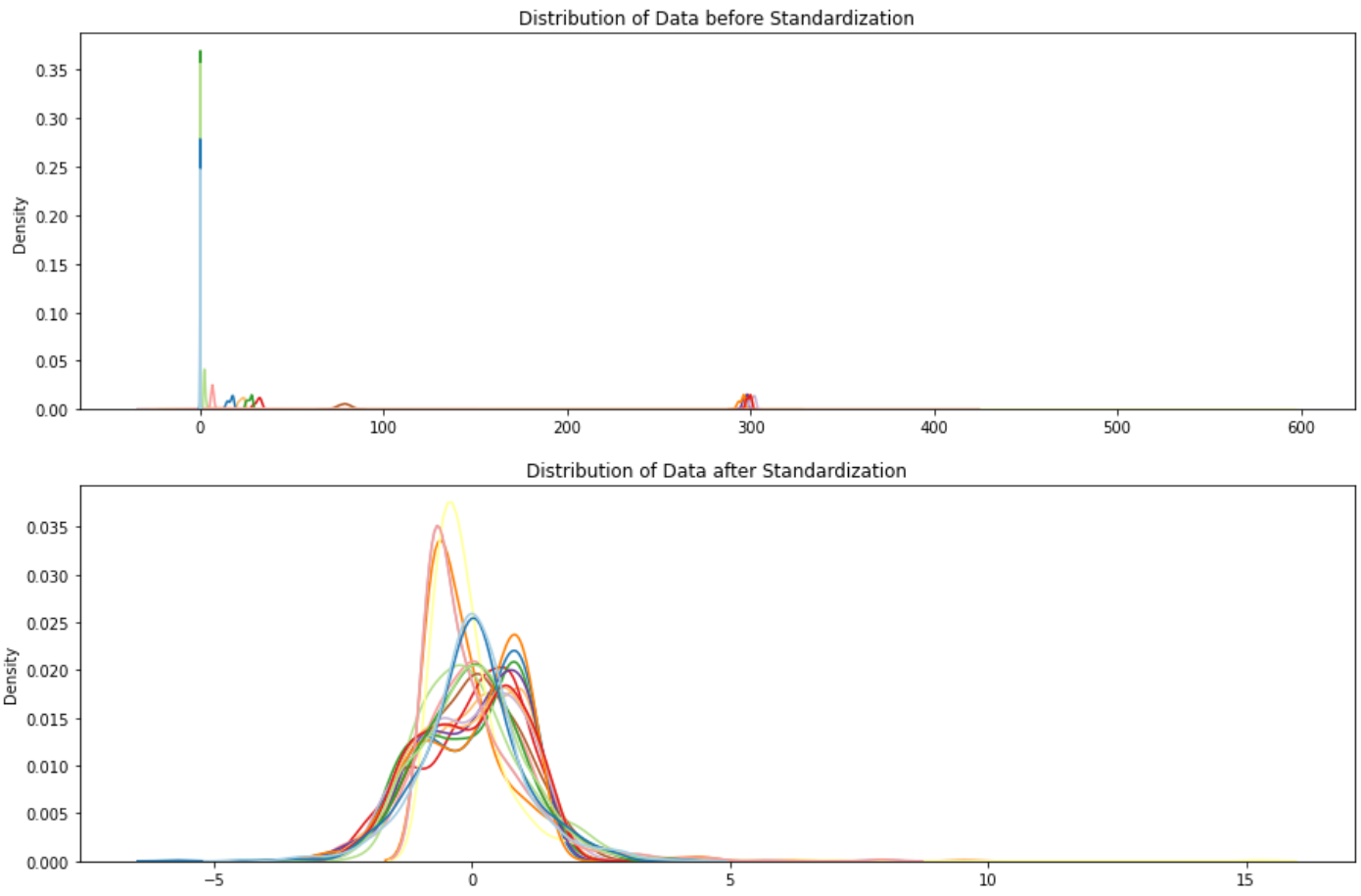
We chose a method to fill missing values by the value before. Since we are working on time series, this one seems to be the best way to fill missing values

```
#Method to fill the missing value by the value before, since we are working on time series , this method seems to be
#the best way to fill missing values .sj_train_features.fillna(method='ffill', inplace=True)
sj_train_features.fillna(method='ffill', inplace=True)
iq_train_features.fillna(method='ffill', inplace=True)
```

Since our Data set has no more missing values now, we can focus on preprocessing the Data using Standardization to use it with the models we have chosen to predict Dengue disease.

```
keepCols = ['ndvi_ne', 'ndvi_nw', 'ndvi_se', 'ndvi_sw', 'precipitation_amt_mm', 'reanalysis_air_temp_k', 'reanalysis_avg_temp_k',
'reanalysis_dew_point_temp_k', 'reanalysis_max_air_temp_k', 'reanalysis_min_air_temp_k', 'reanalysis_precip_amt_kg_per_m2',
'reanalysis_relative_humidity_percent', 'reanalysis_sat_precip_amt_mm', 'reanalysis_specific_humidity_g_per_kg', 'reanalysis_tdtr_k',
'station_avg_temp_c', 'station_diur_temp_rng_c', 'station_max_temp_c', 'station_min_temp_c', 'station_precip_mm']

scaler = StandardScaler()
scaled_df = scaler.fit_transform(sj_train_features[keepCols])
scaled_df = pd.DataFrame(scaled_df, columns=keepCols)

plt.rcParams["figure.figsize"]=[15,10]
plt.subplot(211)
plt.title('Distribution of Data before Standardization')
sns.kdeplot(data=sj_train_features[keepCols], legend=None, palette="Paired")
plt.subplot(212)
plt.title('Distribution of Data after Standardization')
sns.kdeplot(data=scaled_df, legend=None, palette="Paired")
plt.draw()
```

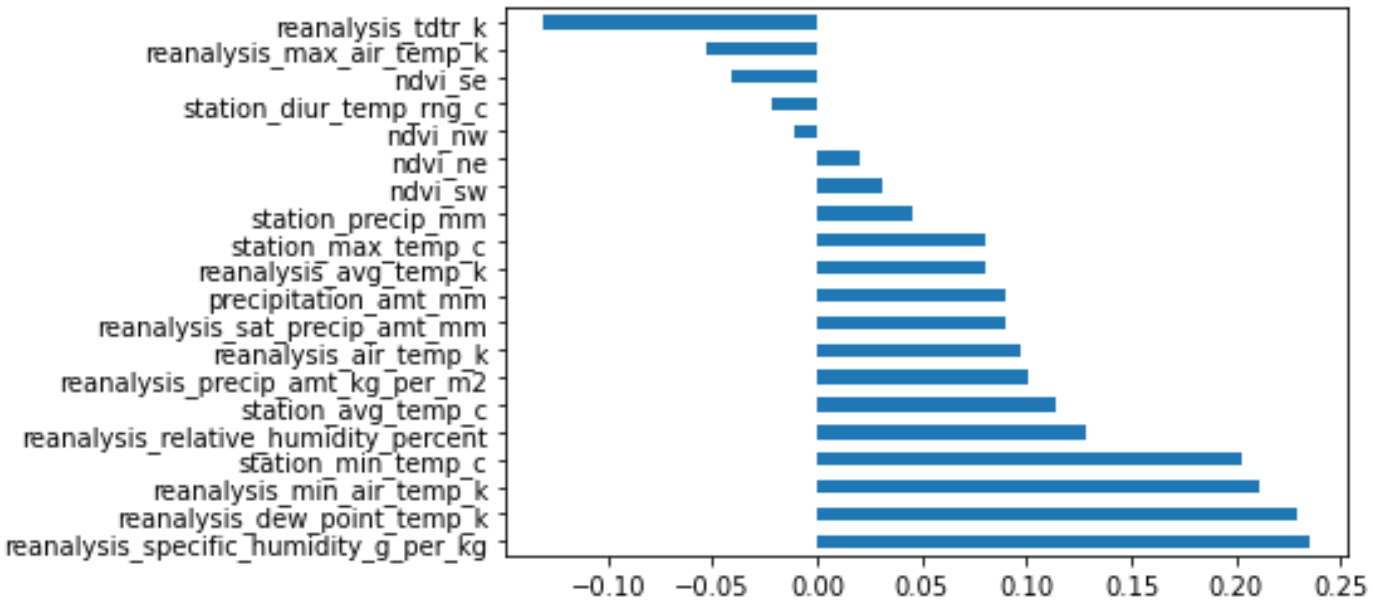
Consider columns as variables. If a column is standardized, a mean value of the column is subtracted from each value, and then values are divided by the standard deviation of the column. The resulting columns have a standard deviation of 1 and a mean that is very close to zero.



We end up having features (columns) that have almost a normal distribution.

We are choosing the variables with the best correlation rate with our target variable.

```
sj_correlations = sj_train_features.corr()
iq_correlations = iq_train_features.corr()

#Graph showing the correlation rates of our variables
(sj_correlations
     .total_cases
     .drop('total_cases')
     .sort_values(ascending=False)
     .plot
     .barh())
```



- We can see that reanalysis_specific_humidity_g_per_kg and reanalysis_dew_point_temp_k are firmly correlated to total cases. These variables are related to the humidity of the climate.

- Average and Min temperature is also strongly correlated to the target variable.

Then, the best features to keep are : reanalysis_specific_humidity_g_per_kg, reanalysis_dew_point_te, station_avg_temp_c, station_min_temp_c.

BUILDING RANDOM FOREST MODEL

After all the data preparation work, we used Scikit-Learn libraries to construct our Random Forest Model model.

We import the random forest regression model from scikit-learn, X represents normalized features, and y our target we want to predict, the total cases.

We instantiate the model and fit (scikit-learn's name for training) the model on the training data.

```
randomforest=RandomForestRegressor(n_estimators=30)
```

```
X = scaled_df_sj
y = sj['total_cases']
```

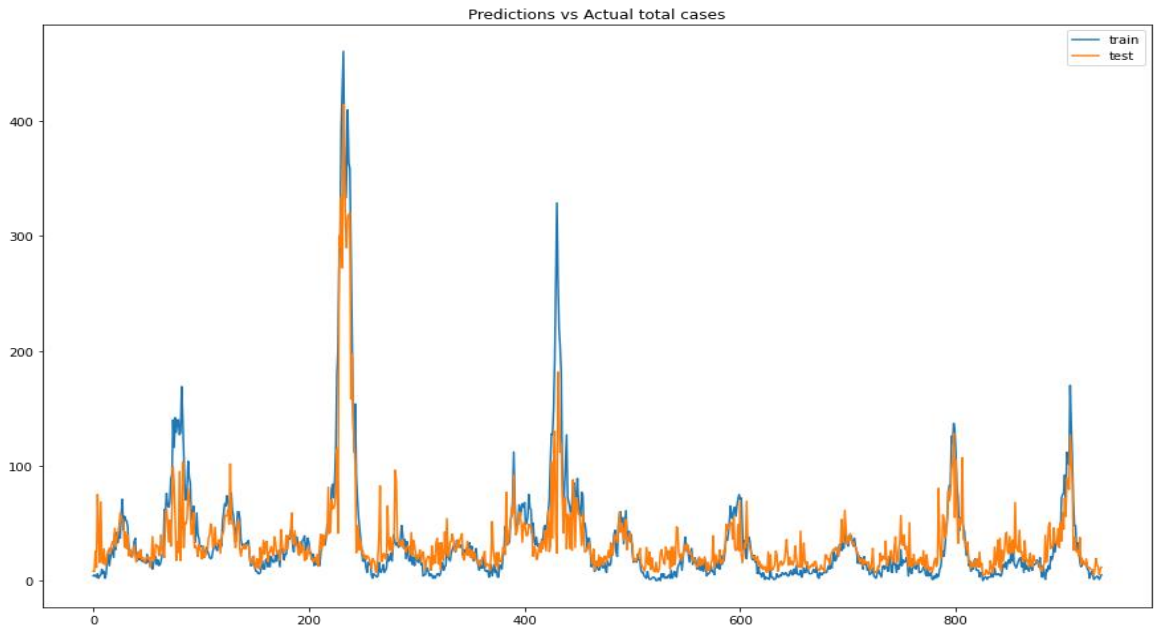
```
randomforest.fit(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)
```

```
regressor = RandomForestRegressor(n_estimators=100, random_state=42, max_depth=None)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_train)
```

```
y_pred2=regressor.predict(X)
```

This graph is a visualization of the predicted and actual cases.




```
#scores on train and test
print(regressor.score(X_train, y_train))
print(regressor.score(X_test, y_test))
```

```
0.9269127407313509
0.4521867201795493
```

That looks pretty good! Our model has learned how to predict the total cases for each week on the training set with 93% accuracy.

BUILDING RNN LSTM MODEL

We used the Keras and Tensorflow libraries to construct our model.

Long Short Term Memory Networks (LSTM) is an extension of recurrent neural networks extending their memory. Therefore, it is well suited for learning essential experiences that have a very long shift in between.

The units of an LSTM are used as building blocks for the layers of an RNN, which is then often called an LSTM network.

LSTMs allow RNNs to remember their inputs over a long period. This is because LSTMs hold their information in a memory, which is very similar to a computer's memory. After all, the LSTM can read, write and delete data from its memory.

The base model we started with was as follows :

```
#Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 4)))
regressor.add(Dropout(0.2))

#Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

#Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
#Adding the output layer
regressor.add(Dense(units = 3))

#Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_absolute_error')
```

Then we experimented with adding a dropout layer of varying proportions, increasing the model's performance significantly.

This lead to the final function to fit the LSTM model on the dataset :

```
# create function to create models and display results of the loss ( here it's mean absolute error)

def run_lstm(dataset_in, dataset_name_string, chart_title, start_col, end_col,
             lookback_periods, train_proportion, dropout_proportion, epoch_count):
    '''
    dataset_in: pandas dataframe
    dataset_name_string: string for name of dataset_in, used in report
    chart_title: string for plot title
    start_col: index of column to start training data with
    end_col: index of column to end the training date from
    lookback_periods: prior time periods to use when predicting
    train_proportion: proportion of the dataset to use for the training data (0-1)
    dropout_proportion: proportion of nodes in the dropout layer to randomly disable during an epoch
    epoch_count: number of epochs to run the model
    '''

    #get values, starting with start_col, which in these datasets skips 'week_start_date'
    dataset = dataset_in.copy()
    values = dataset[dataset.columns[start_col:end_col]].values

    # ensure all data are of type float32
    values = values.astype('float32')

    # normalize all features
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(values)

    # convert data to proper format for supervised learning using lookback_periods
    reframed = series_to_supervised(scaled, lookback_periods, 1)

    # split the data into train and test sets based on given proportion
    values = reframed.values
    n_train_weeks = math.floor(len(values) * train_proportion)
    train = values[:n_train_weeks, :]
    test = values[n_train_weeks:, :]

    # split into explanatry (as opposed to independant) and response variables.
    # Last column, 'total_cases' is the response variable
    train_X, train_y = train[:, :-1], train[:, -1]
    test_X, test_y = test[:, :-1], test[:, -1]

    # reshape input to be 3D [samples, timesteps, features]
    train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
    test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
```

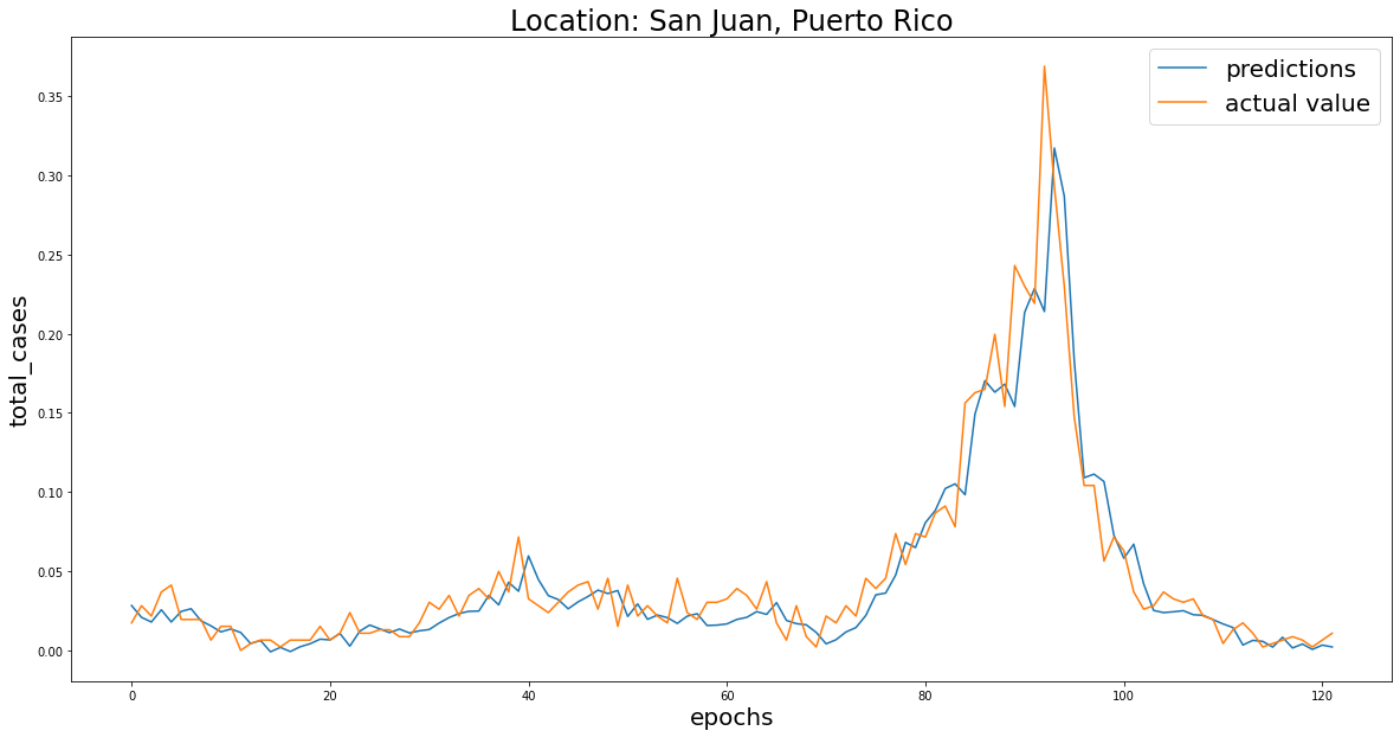
This function takes a data set, and first, it applies a function to converts a time series for supervised learning and then normalizes and fits the values on our three layers RNN-LSTM model.

```
    # Create an LSTM model
    model = Sequential()
    #First Layer
    model.add(LSTM(units = 50, return_sequences = True, input_shape=(train_X.shape[1], train_X.shape[2])))
      # dropout layer
    model.add(Dropout(dropout_proportion))
    #Second Layer
    model.add(LSTM(units = 50, return_sequences = True))
       # dropout layer
        #THIRD Layer
    model.add(LSTM(units = 50, return_sequences = True))
       # dropout layer
    model.add(Dropout(dropout_proportion))

    model.add(Dense(50, activation='relu'))

    # output layer
    model.add(Dense(1))

    model.compile(optimizer = 'adam', loss = 'mean_absolute_error')

    # fit model
    history = model.fit(train_X, train_y, epochs=epoch_count, batch_size=72,
                        validation_data=(test_X, test_y), verbose=0, shuffle=False)

    # output
    print('Dataset name: {}'.format(dataset_name_string))
    print('Lookback window periods: {}'.format(lookback_periods))
    print('Training proportion: {}'.format(train_proportion))
    print('Dropout proportion: {}'.format(dropout_proportion))
    print('last epoch loss: {}'.format(history.history['loss'][-1]))
    print('last epoch val_loss: {}'.format(history.history['val_loss'][-1]))
# plot history
    plt.figure(figsize=(20,10))
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.xlabel('epochs', fontsize=20)
    plt.ylabel('loss proportion', fontsize=20)
    plt.legend(fontsize=20)
    plt.title('Location: {}'.format(chart_title), fontsize=24)
    plt.show()
```

This model is using the mean absolute error to evaluate its performance.

MAKING PREDICTIONS

Here's a chart showing the number of cases, predicted vs. actual cases :




```
#Mae score
print(temp)
```

```
0.01390035729855299
```

The mean absolute error of 0.014 (the average difference between actual cases and predicted Dengue cases) shows there is room for improvement. Still, the model accurately predicts the massive spike of over 150 cases around week 90.

Information like this can be extremely valuable to decision-makers in a region.

CONCLUSION

With these graphs, we have completed an entire end-to-end machine learning prediction ! If we want to improve our model, we could try different hyperparameters (settings), test more different algorithms, or, the best approach of all, gather more data !

Moreover, we hope everyone who made it through enjoyed reading us. We want to thank everyone who made this project possible, our school HETIC with the General Manager Frédéric Sitterlé who allowed Deepnet to offer us their project. A vast thanks to Max Cohen and Serhat Yildirim. They have been excellent mentors throughout this project and showed us how accessible machine learning was and made us more passionate about the potential of data science to make the world a better place !