*A project report on*

# NLP Dashboard & Rainfall Forecasting

*Submitted in partial fulfillment for the award of the degree of*

## Bachelor in Technology Information Technology

*by*

**Om Hemantkumar Purohit (17BIT0368)**



**SITE**

**April 2021**

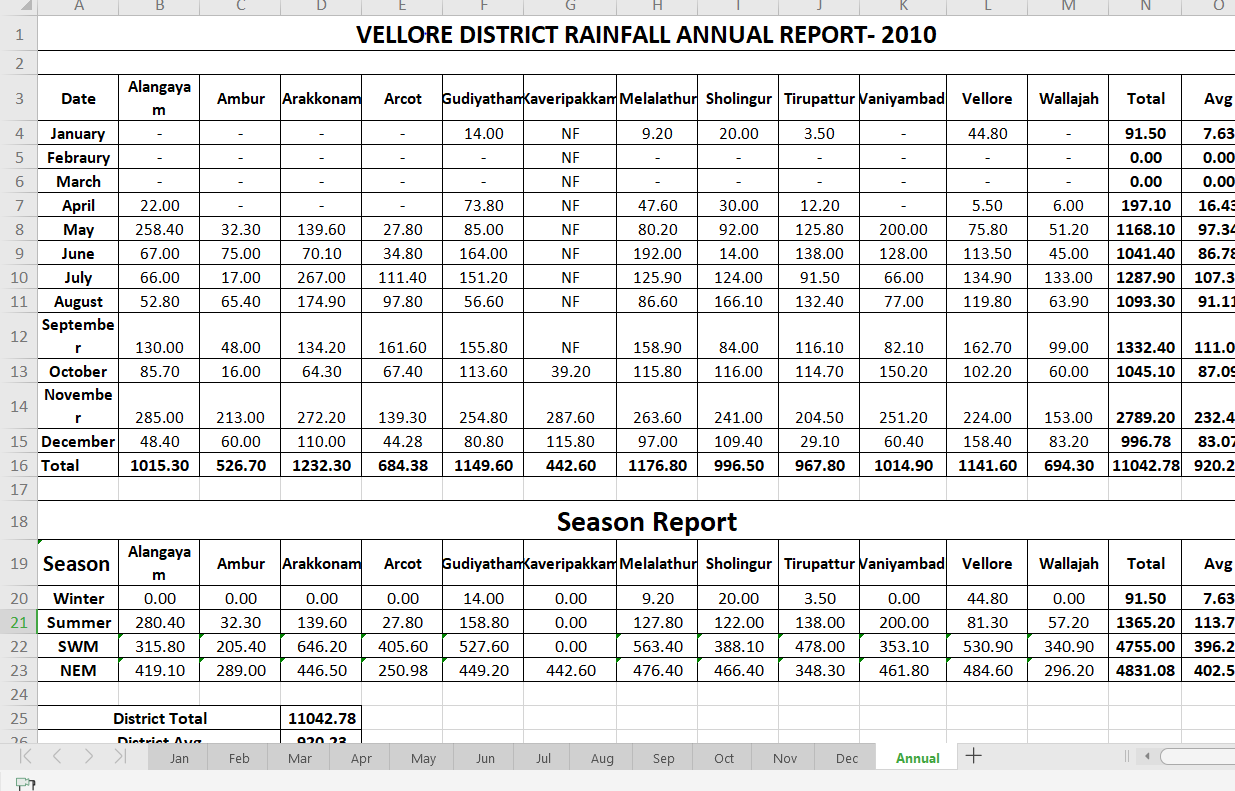
**RAINFALL FORECASTING**

**ABSTRACT**

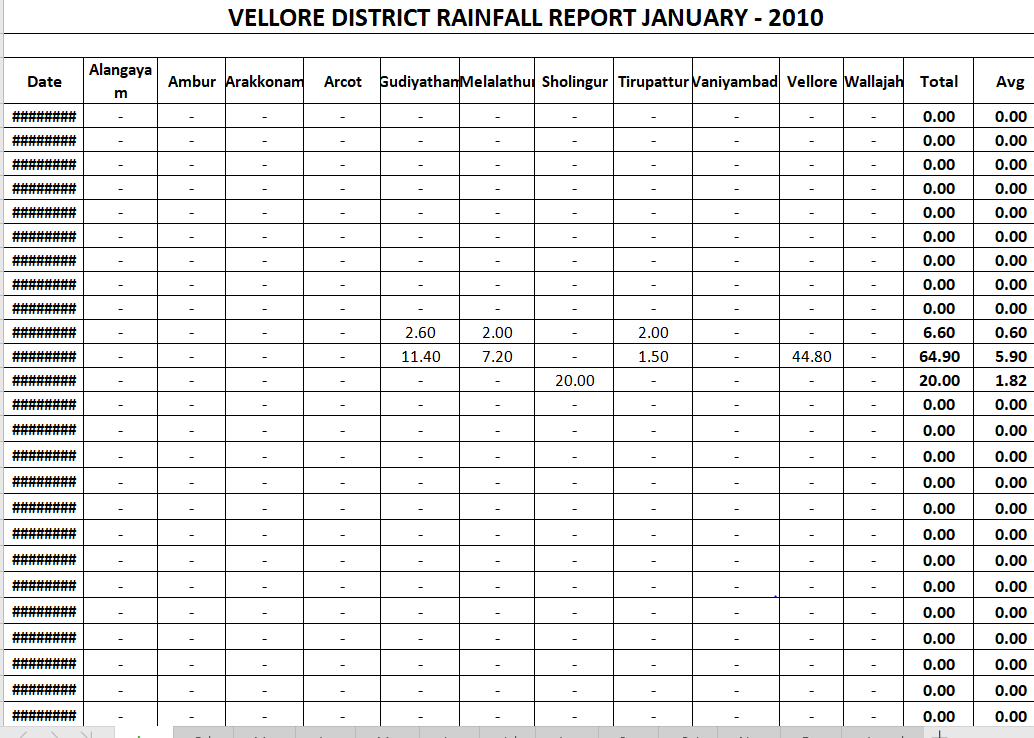
The biggest problem we face nowadays is not knowing whether it is going to rain or not. Imagine planning for something big, and all your plan gets canceled just because it starts raining. Sounds pretty irritating, doesn’t it? Well, now we have a solution to this problem and we call it the Rain Predictor. Rain Predictor is a software which will help us predict whether it is going to rain on a particular day or not. Rain Prediction will be done on the basis of a simple factor the date and district. We will even perform some feature engineering and generate new features to fine tune our model. etc. On the basis of the available dataset on Vellore, which is a Time-series data ranging from year 2010 to 2019. We are going to perform forecasting techniques such as ARIMA, Exponentional Smoothening. We will also create neural networks and apply LSTM as it is a well known fact that LSTM can handle be used to model on Time-series data. Lastly we will see if the problem statement at hand is a regression problem or not? We will also look at regression techniques and how we can manipulate our dataset and remove the time series component in order to perform regression techniques.

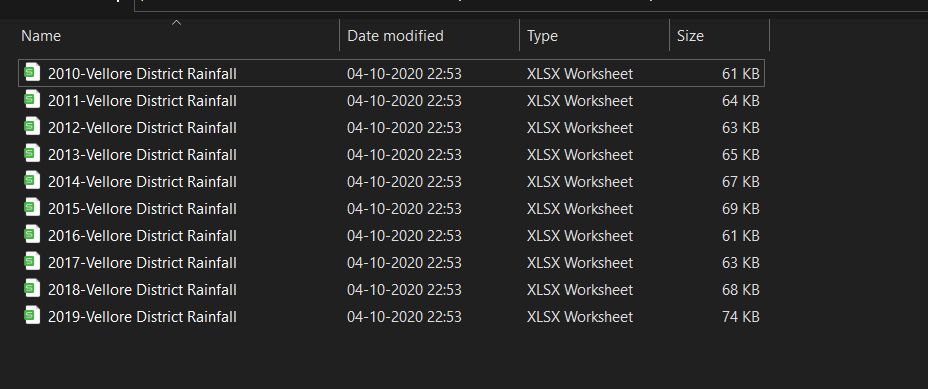
**The Dataset**

We have Excel files ranging from the year 2010- 2019, below picture is the Raw unprocessed data! The below Image is for the year 2010 Annual Rainfall in millimeter.



The below image is the data for Daily rainfall in mm of month of January 2010





**Preprocessing and Cleaning Data**

import pandas as pd

# %%

df10=pd.read\_excel("rainfalldata/2010-Vellore District Rainfall.xlsx",sheet\_name=[0,1,2,3,4,5,6,7,8,9,10,11,12])

df11=pd.read\_excel("rainfalldata/2011-Vellore District Rainfall.xlsx")

df12=pd.read\_excel("rainfalldata/2012-Vellore District Rainfall.xlsx")

# %%

df10[1]

# %%

df10[0].columns=df10[0].iloc[1,0:]

df10[0]

# %%

newcolumns=df10[0].columns[1:-2]

# %%

newcolumns

# %%

tempdata={"Date":[],"District":[],"Rain":[]}

# %%

for year in range(10,20):

df10=pd.read\_excel("rainfalldata/20"+str(year)+"-Vellore District Rainfall.xlsx",sheet\_name=[0,1,2,3,4,5,6,7,8,9,10,11,12])

for i in range(0,12):

df10[i].columns=df10[i].iloc[1,0:]

for row,frame in df10[i].iterrows():

if row<2 or row>(df10[i].shape[0]-5):

continue

else:

newcolumns=df10[i].columns[1:-2]

for newcol in newcolumns:

tempdata["Date"].append(frame["Date"])

tempdata["District"].append(newcol)

tempdata["Rain"].append(frame[newcol])

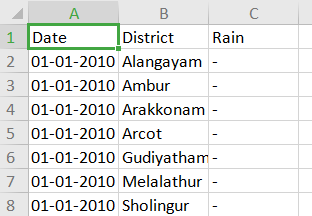
# %%

BigDF=pd.DataFrame(tempdata)

# %%

BigDF.to\_csv("combined.csv",index=False)

Combining the Data from 2010 to 2019, we create a final dataset called combined.csv



**Dataset Shape ===> 46293\*3**

**Replacing “-” with 0.0 float values, indicating that no rain had occurred.**

def convertToFloat(x):

x=x.strip()

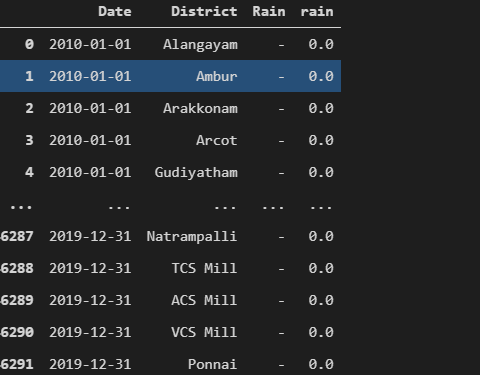
if x=="-" or x=="NR" or x=='' or x=="." or x=="" or x==' ':

return 0

else:

thevar[0]=x

return float(x)



**Exploratory Data Analysis**

Using Altair graphing library, we can create **interactive graphs.** For example one can easily use these graphs to fetch the rainfall in mm of a given input of (month, year). It features rich sliders and dropdown menu as widgets to make a tiny UI for better user experience.

# %%

get\_ipython().system('pip install altair vega\_datasets')

# %%

BigDF=pd.read\_csv("combined.csv")

# %%

from datetime import datetime

def getYear(x):

print(x)

return int((datetime.strptime(str(x),"%Y-%m-%d").year))

def getMonth(x):

return int(datetime.strptime(str(x),"%Y-%m-%d").month)

def getDay(x):

return int(datetime.strptime(str(x),"%Y-%m-%d").day)

# %%

# %%

BigDF["year"]=BigDF["Date"].apply(getYear)

BigDF["month"]=BigDF["Date"].apply(getMonth)

BigDF["day"]=BigDF["Date"].apply(getDay)

# %%

def ChangeRainToFloat(x):

try:

return float(x)

except:

return 0

BigDF["Rain"]=BigDF["Rain"].apply(ChangeRainToFloat)

# %%

"Total" in BigDF.Date.unique()

# %%

BigDF

# %%

df\_grouped=BigDF.groupby(["District","year","month"]).sum().reset\_index()

df\_grouped

# %%

df\_grouped["District"].unique()

# %%

import altair as alt

slider = alt.binding\_range(min=2010, max=2019, step=1)

select\_year= alt.selection\_single(name='year', fields=['year'],

bind=slider, init={'year': 2011})

slider2 = alt.binding\_select(options=df\_grouped["District"].unique())

select\_district= alt.selection\_single(name="District", fields=['District'],

bind=slider2, init={'District': "Vellore"})

alt.Chart(df\_grouped).mark\_bar().encode(

x="month",

y="Rain",

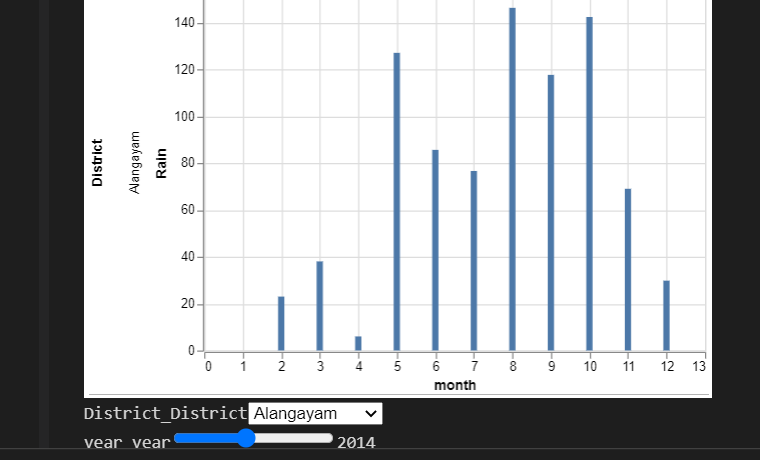
row="District",

tooltip=["Rain","month"]

).add\_selection(select\_year,select\_district).transform\_filter(select\_year).transform\_filter(select\_district)

# %%

BigDF[(BigDF["year"]==2010) & (BigDF["District"]=="Kaveripakkam")& (BigDF["month"]==11) ]

# %%

**Detecting Trend and Seasonality in the Time-Series Data**

**Trend**:The trend is the component of a time series that represents variations of low frequency in a time series, the high and medium frequency fluctuations having been filtered out. This component can be viewed as those variations with a period longer than a chosen threshold (usually 8 years is considered as the maximum length of the business cycle)

**Seasonality**:Seasonal variation, or seasonality, are cycles that repeat regularly over time. A repeating pattern within each year is known as seasonal variation, although the term is applied more generally to repeating patterns within any fixed period.

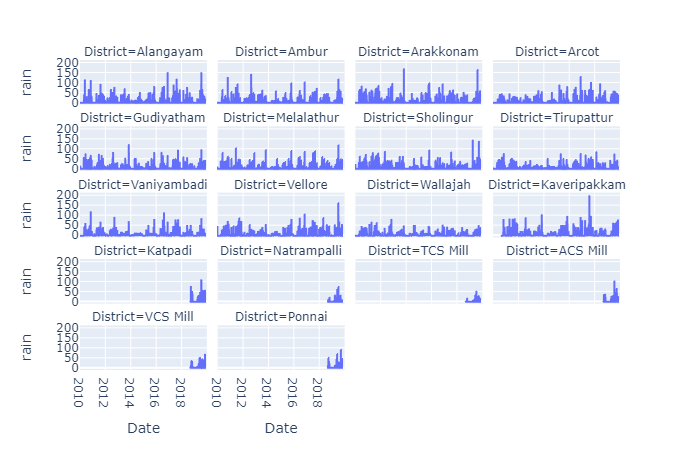
**Why we need to check for seasonality and trend? What does it mean a time-series to be stationary?**

A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time.In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behaviour is possible), with constant variance.

**Is our Time Series Data Stationary ?**

import plotly.express as px

fig = px.line(df, x="Date", y="rain", facet\_col="District",facet\_col\_wrap=4)



Although detecting seasonality from visualizations is not practical, it helps in detecting if there are any trends in the data. From the above visualization we can easily conclude that there is **no increasing or decreasing trend.**

**Augmented Dicky Fuller Test**

The Augmented Dickey Fuller Test (ADF) is unit root test for stationarity. Unit roots can cause unpredictable results in your time series analysis.

* The [null hypothesis](https://www.statisticshowto.com/probability-and-statistics/null-hypothesis/" \t "https://www.statisticshowto.com/adf-augmented-dickey-fuller-test/_blank) for this test is that there is a unit root.
* The [alternate hypothesis](https://www.statisticshowto.com/what-is-an-alternate-hypothesis/" \t "https://www.statisticshowto.com/adf-augmented-dickey-fuller-test/_blank) differs slightly according to which equation you’re using. The basic alternate is that the time series is stationary (or trend-stationary).

**Unit root tests** can be used to determine if trending data should be first differenced or regressed on deterministic functions of time to render the data stationary. Moreover, economic and finance theory often suggests the existence of long-run equilibrium relationships among nonsta- tionary time series variables.



Dt-> Deterministic component

Zt-> Random(Stochastic component)

Et-> Stationary Error

We are only concerned with the result of AD-Fuller test to detect whether our series is stationary or not.

We perform the test below.

**Using AutoLag**

**Lag** in a time series data is displacement of the data by an order of **t**, where t is the assumed time period where seasonality occurs. We will be using AutoLag in our Augmented Dicky Fuller test, where the test will automatically detect a lag for testing.

from statsmodels.tsa.stattools import adfuller

def test\_stationarity(timeseries):

print('Results of Dickey-Fuller Test:')

dftest = adfuller(timeseries)

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value',

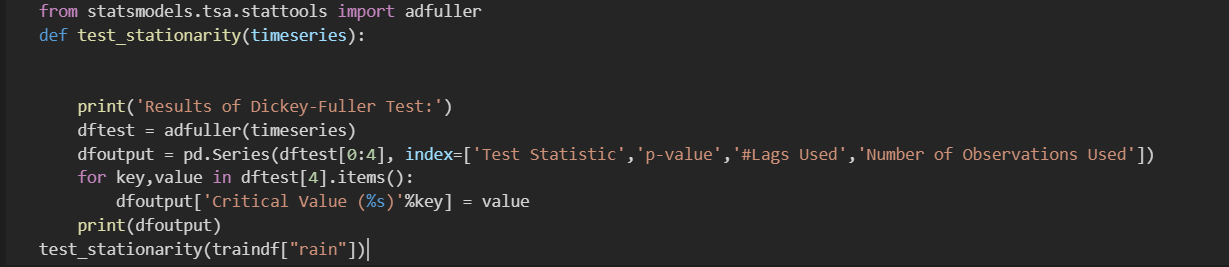
'#Lags Used','Number of Observations Used'])

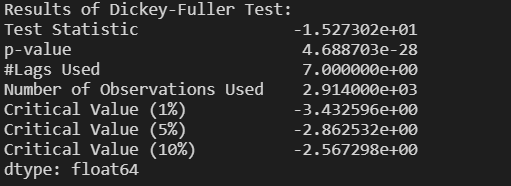
for key,value in dftest[4].items():

dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)

test\_stationarity(traindf["rain"])





From the above test the important results for us are the **Test Statistic** and **Critical Value(10%)**(this is the benchmark critical value) We observe that the critical value **-3.432** is greater than Test statistic **-15.273. The lag automatically detected is 7**

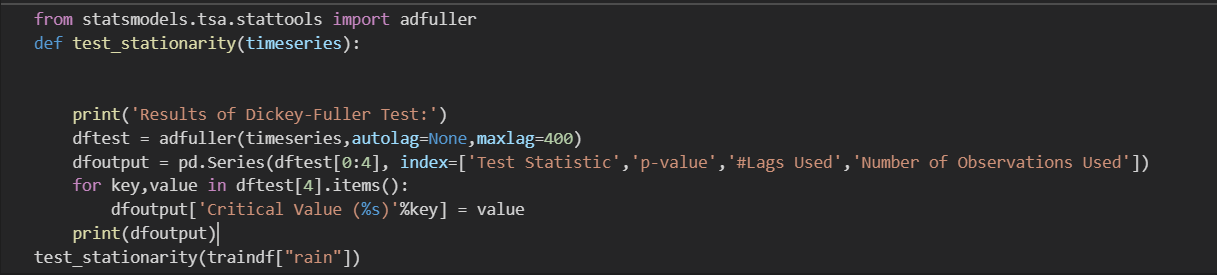
**-15.273 < -3.432**

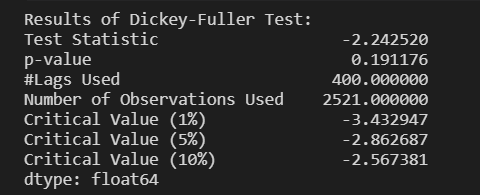
**Test Statistic < Critical value**

**We reject null hypothesis and accept the alternate hypothesis.**

The result is astonishing! The test is concluding that our data is already **stationary**! It is surprising as rainfall data should always contain **seasonality** and be **non-stationary.**

**Using MaxLag=400 and AutoLag=None**





From the above test the important results for us are the **Test Statistic** and **Critical Value(10%)**(this is the benchmark critical value) We observe that the critical value **-2.56** is greater than Test statistic **-3.**

**-2.24 < -2.56**

**Test Statistic < Critical value**

**We accept the null hypothesis.**

Using lag of 400 which is approximately a time period of 1 year, the test is able to detect the seasonality. In order to remove seasonality we use **Differencing** with interval of 400

def difference(dataset, interval=1):

diff = list()

for i in range(interval, len(dataset)):

print(i)

value = dataset[i] - dataset[i - interval]

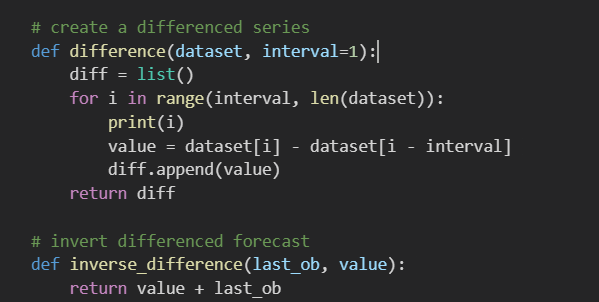
diff.append(value)

return diff

# invert differenced forecast

def inverse\_difference(last\_ob, value):

return value + last\_ob



**The ARIMA model**

ARIMA stands for Auto Regressive Integrated Moving Average model. An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends.

**Autoregressive**

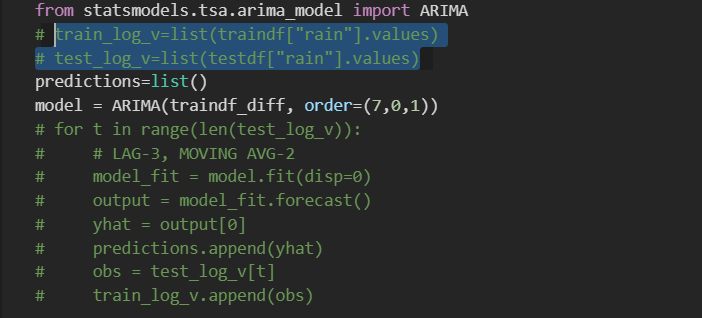
A statistical model is Autoregressive if it it predicts the time series data from previous data. It seeks to use a lag to shift the time series data. A lag of k means it will predict the values using previous k terms of the dataset.

**Moving Average**

Moving average is a simple statistical measure to calculate averages in a moving window. It is also used in technical analysis of stock data, which is a well known time series data.

The formula is ARIMA(x,y,z). x determines the Autoregression lag to be used, moving average is determined by a lag z and y is used for integration, i.e the differencing order. We already have performed differencing hence we initalise this to 0.

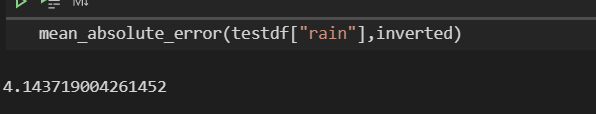
ARIMA(7,0,1)



**TEST ACCURACY**

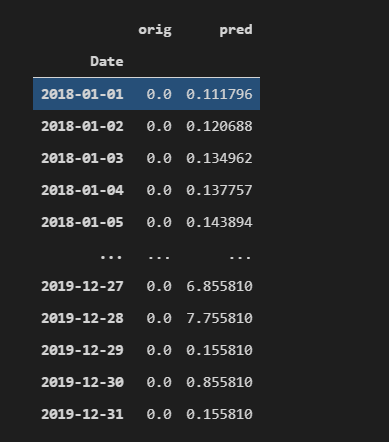
We will be using a common metric for measuring error of various models in the future**. MAE, mean absolute error** is a great choice compared to other metrics. As some metrics tend to divide the error by the the true value, for the case of data containing 0’s it can lead to error reaching **infinity.**

**For MAE the error more close to 0.0 is better!**



**Mean Absolute Error-> 4.143**

The error is large and **inefficient model.**



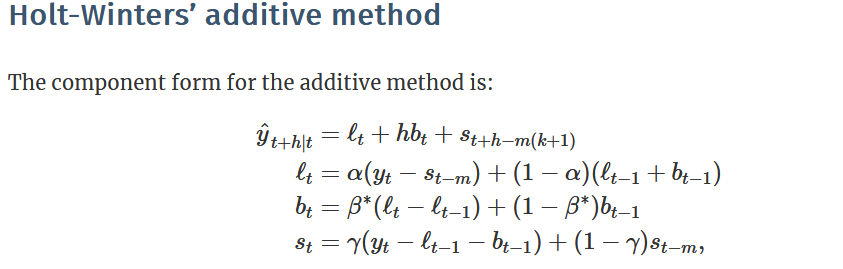
It is obvious from the above image, the model is not able to predict rainfall of more than **1mm** accuracy.

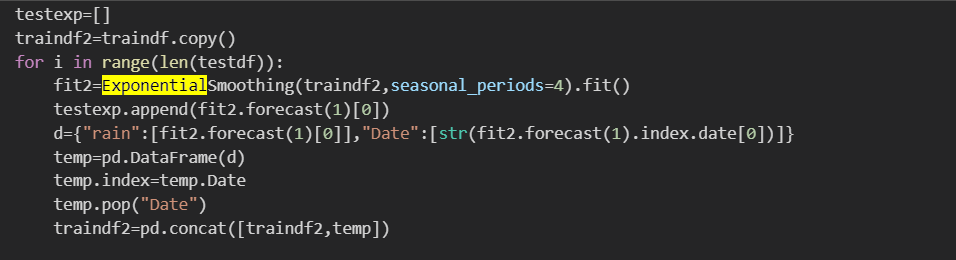
**The Holt-Winter Exponential Smoothing Model**

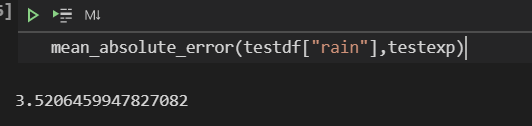
Exponential smoothing was proposed in the late 1950s (Brown, 1959; Holt, 1957; Winters, 1960), and has motivated some of the most successful forecasting methods. Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight. This framework generates reliable forecasts quickly and for a wide range of time series, which is a great advantage and of major importance to applications in industry.



Here the alpha value is the smoothing parameter. The algorithm gives more preference to the nearest predecessor, and the further the predecessor, the less impact it has on the forecast.

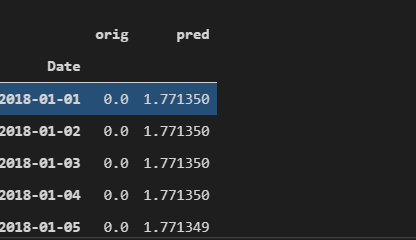






**Mean Absolute Error-> 3.5206**

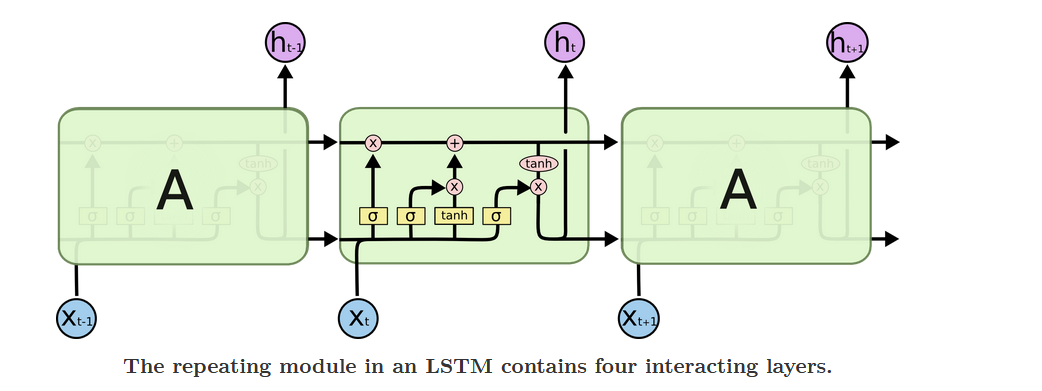
**Although it is less than ARIMA model, the model is actually worse.**



**Predicted values do not even exceed 2 mm!**

**LSTM-Long-Short Term Memory**

We enter the world of Deep learning to get better results using a well known implementation of a neural network for time series data, LSTM. It is an advancement of Recurrent neural networks which use forget gates to keep on the required information stored. It has been widely applied in Stock market technical analysis. The fully connected layers in the network are in fact cells in LSTM, which provide error flow to be smooth between gates. This allows us to bridge previous cells.

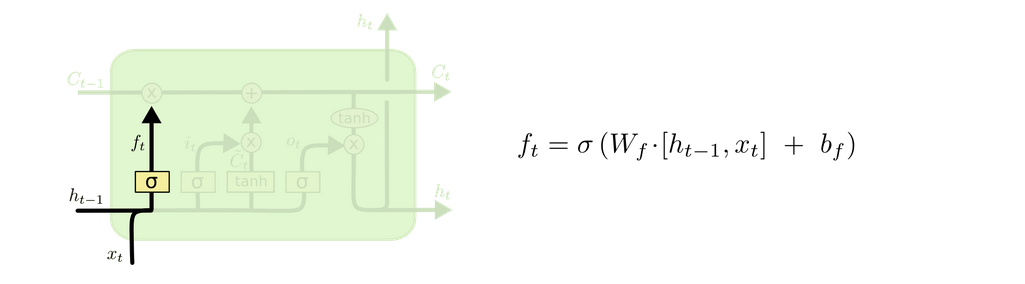


**Pink circles -**> **denote pointwise opeartion**

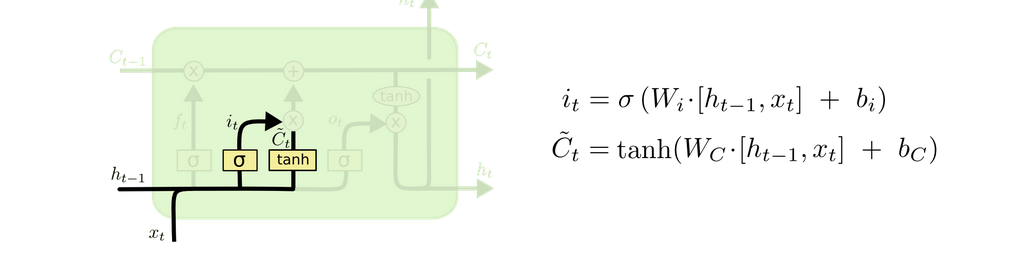
**Yellow boxes -> denotes neural network layers**

**Black arrows-> carries vectors(Cell state->analogous to conveyor belt)**

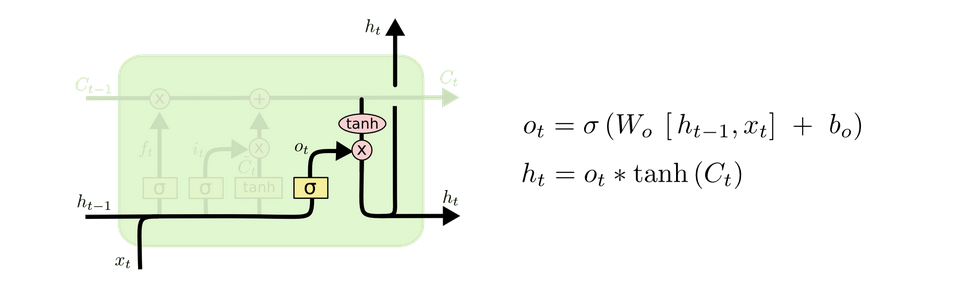
The cell state carries information seamlessly throughout cells. Gates such as the Sigmoid activation layer which usually outputs either a 0 or 1. a 0 will mean that no data can be passed to the pink circle, which is where the pointwise operation happen such as a dot product.



Here the previous cell output(h[t-1]) and input(x[t]) are concatenated and flow through the sigmoid gate which either outputs 0 or 1. If 0 is outputted then the information is rejected and not used, hence this gate is called **forget gate.**



We go over our next gate, which involves using the previous value we have forgotten to create a new candidate for cell state Ct. Using the tanh activation aides in creating this new vector and updating our cell state.



The last stage of the LSTM, is to decide what to output. Again the input goes through a sigmoid gate to decide what information to keep and a final tanh activation layer to convert the values in between -1 to 1.

Below is the implementation of our LSTM. I have decided to use look\_back=30, as we will use past 30 values to predict the future.

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow import keras

def CreateDataset(dataset,lookback):

DataX=[]

DataY=[]

for i in range(len(dataset)-lookback-1):

DataX.append(dataset[i:(lookback+i),0])

DataY.append(dataset[(lookback+i),0])

return np.array(DataX),np.array(DataY)

trainsize=3287

testsize=365

train, test = dataset[0:trainsize,:], dataset[trainsize:len(dataset),:]

look\_back=30

trainX,trainY=CreateDataset(train,look\_back)

testX,testY=CreateDataset(test,look\_back)

# reshape input to be [samples, time steps, features]

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

model = Sequential()

opt = keras.optimizers.Adam()

model.add(keras.layers.Bidirectional(LSTM(4, input\_shape=(1, look\_back))))

model.add(Dense(1,activation="relu"))

model.compile(loss='mean\_absolute\_error', optimizer=opt)

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

from sklearn.metrics import mean\_squared\_error

ypred=testPredict.reshape(334)

ty=testY.reshape(334)

The training error produced is lower than previous models(mean absolute error) 1.9907

**Mean Absolute Error(Test Dataset)->2.955**

**The problem in our forecast? Only 30 values out of 334 are Non-zero, hence our prediction is Sparse!**

**Feature Engineering and Xgboost gradient boosting**