Multivariate time series analysis DMA_225

1.Goal of the thesis: It is to do the DMA_225 prediction considering the external factors.

2.Target variable: DMA_225, which is considered in hourly format

3.Here the below features are considered for analysis along with the target variable.

```
: data_DMA225_indexed.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2208 entries, 2016-04-22 00:00:00 to 2016-07-22 23:00:00
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   DMA_225        2208 non-null   float64
 1   tempC          2208 non-null   int64  
 2   HeatIndexC     2208 non-null   int64  
 3   FeelsLikeC     2208 non-null   int64  
 4   WindChillC     2208 non-null   int64  
 5   windspeedKmph  2208 non-null   int64  
 6   sunHour        2208 non-null   float64
 7   precipMM       2208 non-null   float64
 8   humidity       2208 non-null   int64  
dtypes: float64(3), int64(6)
memory usage: 172.5 KB
```

4. Performed below two files: with 12 hours input, considering 24 hours output DMA_225 prediction.

- DMA_225_MV. ipynb
- DMA_225_timeseries_LSTM.ipynb

DMA_225_MV. Ipynb

1.Data Restructuring: with 12 hours input, considering 24 hours output DMA_225.

After it, the shape of the dataset is as shown below:

```
# Restructuring the data
def reshaped_hourlydata(data,step_input,step_output):
  final_data =  np.array([data[i:i + (step_input+step_output)].copy() for i in range(len(data) - (step_input+step_output))])
  return final_data

final_hourlydata = reshaped_hourlydata(scaled_hourlydata,12,24)
# #final_hourlydata = reshaped_hourlydata(data_hourly,12,24)
final_hourlydata.shape
```

```
(2172, 36, 9)
```

2. Train-test split:

The dataset is divided into 60% training set and rest into validation and test set.

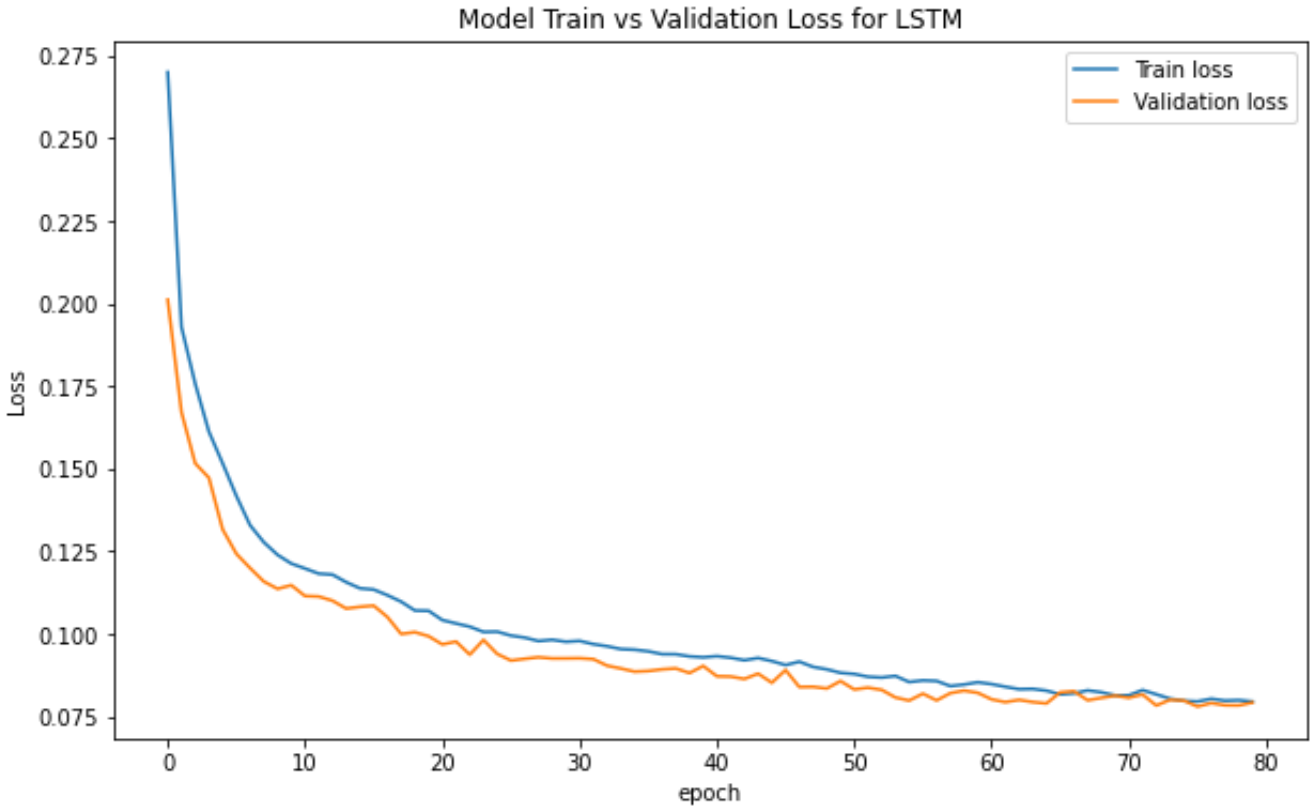
```
print("The shape of train_X is:",train_X.shape)
print("The shape of valid_X is:",valid_X.shape)
print("The shape of test_X is:",test_X.shape)

print("The shape of train_y is:",train_y.shape)
print("The shape of valid_y is:",valid_y.shape)
print("The shape of test_y is:",test_y.shape)
```

```
The shape of train_X is: (1303, 12, 9)
The shape of valid_X is: (608, 12, 9)
The shape of test_X is: (261, 12, 9)
The shape of train_y is: (1303, 12, 24)
The shape of valid_y is: (608, 12, 24)
The shape of test_y is: (261, 12, 24)
```

3. RNN – LSTM model architecture:

```
model = keras.models.Sequential([
keras.layers.LSTM(100,return_sequences=True, input_shape=(12,9)),
keras.layers.Dropout(0.1),
keras.layers.Dense(70,kernel_initializer='normal'),
#keras.layers.Dropout(0.35),
keras.layers.Dense(70,kernel_initializer='normal'),
#keras.layers.Dropout(0.45),
keras.layers.Dense(24)
])
#opt = keras.optimizers.Adam(learning_rate=0.009)
#model.compile(loss='mse', optimizer=opt),
model.compile(loss='mae', optimizer='adam'),
history = model.fit(train_X,train_y, batch_size = 100,epochs = 80,validation_data=(valid_X, valid_y))
```

4. Loss graph:



5. Predicting on test set:

```
#Generate predictions
train_pred = model.predict(train_X)
#evaluation = model.evaluate(x=X_test, y=y_test, verbose=1)
test_pred = model.predict(test_X)
predictions = test_pred
```

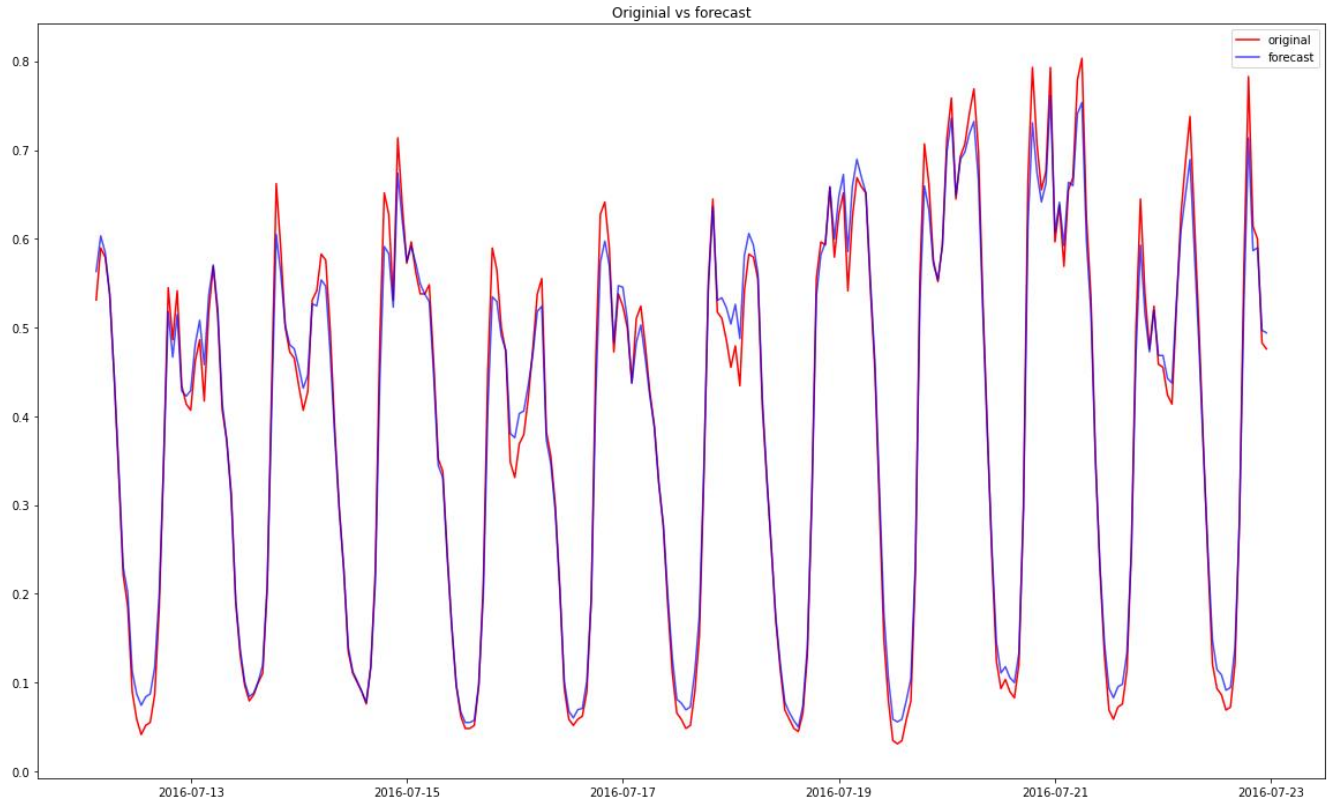
```
# To make a dataframe of original and predicted value:
lstm_df=[]
for i in range (0, len(predictions)):
  lstm_df.append((predictions[i][0][step_output-1]))
final_df = pd.DataFrame((test_X[:,0]))
final_df.rename(columns = {0:'original_value'}, inplace = True)
final_df['predicted_value'] = lstm_df
```

```
# To calculate the percentage difference between actual and predicted value:
final_df['total_difference'] = (final_df['predicted_value'] - final_df['original_value']).abs()
final_df['percentage_difference'] = ((final_df['total_difference'])/(final_df['original_value']))*100
```

```
import math
from sklearn.metrics import mean_squared_error
#testset = math.sqrt(mean_squared_error(final_df['original_value'], final_df['predicted_value']))
testset = math.sqrt(mean_squared_error(final_df['original_value'], final_df['predicted_value']))
print("The RMSE prediction value on testset: ",testset)
```

```
The RMSE prediction value on testset:  0.022413925553471954
```

6. Plotting original and forecast value:



DMA_225_timeseries_LSTM.ipynb

1.Feature-target variable: After data is considered into hourly basis, I considered feature and target variable to do DMA_225 prediction, when considering external climatic factors.

```
#Feature, Target dataset // 24 hour prediction
# feature_variable- last 24 hours columns and readings are removed
# target_variable- first 24 hours dma_225 columns and readings are removed

feature_variable = scaled_data[:][:-24]
target_variable = scaled_data[:, 0][24:]
```

```
feature_variable.shape
```

```
(2184, 9)
```

```
target_variable.shape
```

```
(2184,)
```

2. Train test split: using time series generator, here 12 hours step input is taken to predict 24 hours output as below

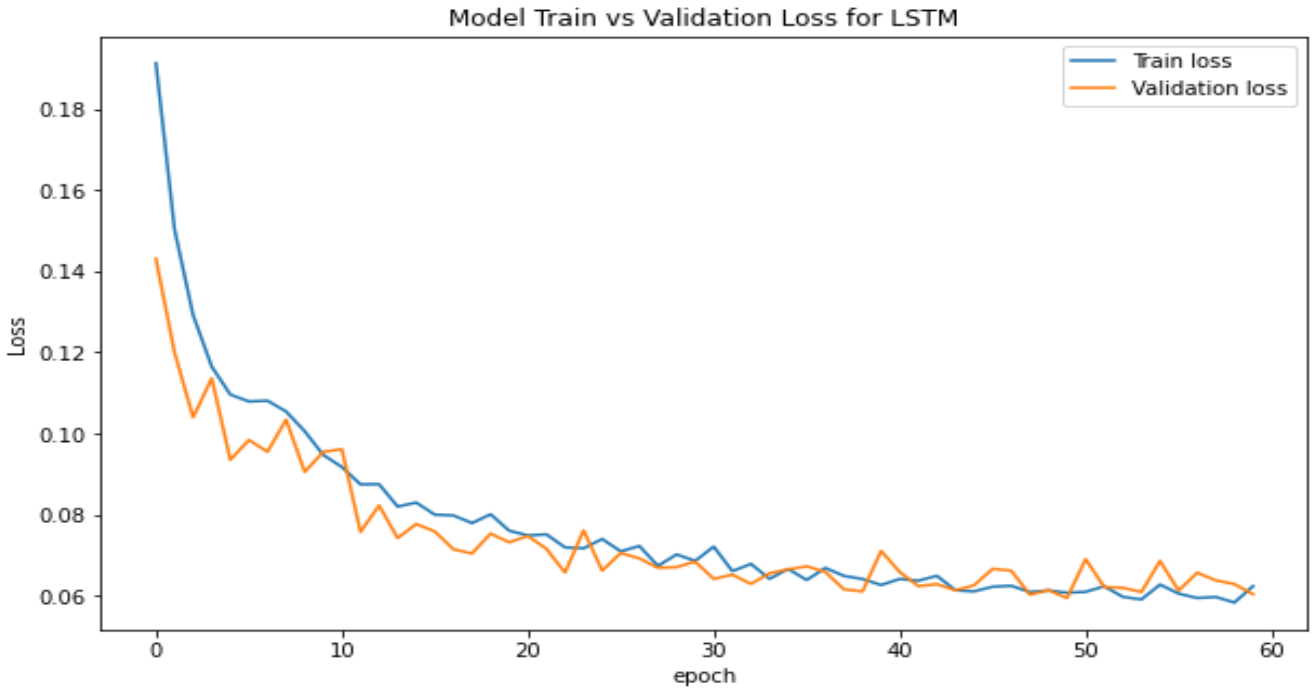
The dataset is divided into 60% training set and rest into test set.

```
#Train-Test Split:
train_X, test_X, train_y, test_y = train_test_split(feature_variable, target_variable, test_size=0.40, random_state=123, shuffle = False)

# Create window of lagged variables
timesteps = 12
batch_size=32
train_generator = TimeseriesGenerator(train_X, train_y, length=timesteps, sampling_rate=1, batch_size=batch_size)
test_generator = TimeseriesGenerator(test_X, test_y, length=timesteps, sampling_rate=1, batch_size=batch_size)
```

3.Model architecture:

```
model = keras.models.Sequential([
keras.layers.LSTM(100,return_sequences=True, input_shape=(12,9)),
keras.layers.ReLU(0.4),
#keras.layers.Dropout(0.1),
#keras.layers.LSTM(100,return_sequences=True,),
#keras.layers.LeakyReLU(0.5),
#keras.layers.Dropout(0.30),
keras.layers.LSTM(75,return_sequences=False),
#keras.layers.LeakyReLU(0.5),
keras.layers.Dropout(0.35),
keras.layers.Dense(70),
keras.layers.Dropout(0.45),
keras.layers.Dense(1)
])
#opt=keras.optimizers.Adam(learning_rate=0.0001,decay=0.1)
model.compile(loss='mae', optimizer='adam'),
history = model.fit(train_generator, epochs=65, validation_data=test_generator,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=10)],shuff
```

4.loss graph:



5.Prediction on test set:

```
#evaluation = model.evaluate(x=X_test, y=y_test, verbose=1)
test_pred = model.evaluate(test_generator)
forecast = model.predict(test_generator)
forecast.shape
```

```
# To make a dataframe of original and predicted value:
lstm_df=[]
for i in range (0, len(forecast)):
  lstm_df.append((forecast[i][0]))
final_df = pd.DataFrame((test_X[:,1][12:]))
final_df.rename(columns = {0:'original_value'}, inplace = True)
final_df['predicted_value'] = lstm_df
```

```
import math
from sklearn.metrics import mean_squared_error
#testset = math.sqrt(mean_squared_error(final_df['original_value'], final_df['predicted_value']))
testset = math.sqrt(mean_squared_error(final_df['original_value'], final_df['predicted_value']))
print("The RMSE prediction value on testset: ",testset)
```

```
The RMSE prediction value on testset:  0.36736699747058066
```

6. Forecast of DMA_225:

Considered the forecast column, did an inverse transform on it and got the final df “DMA_225_final.”

```
forecast_df=pd.concat([pd.DataFrame(forecast), pd.DataFrame(test_X[:,1:][12:])],axis=1)
inverse_transform=scaler.inverse_transform(forecast_df)
abc=inverse_transform[:,0]
DMA_225_final=hourly_data[forecast.shape[0]*-1:]
```

7. Forecasting results

The final df: ‘result_df’

```
result_df = DMA_225_final.reset_index()
# Plotting original and predicted graph:
plt.figure(figsize=(20, 12))
plt.plot(result_df.Date_time, result_df.DMA_225, color='red', label='original')
plt.plot(result_df.Date_time, result_df.DMA_225_forecast, color='blue', label='forecast', alpha=0.7)
plt.title('Originial vs forecast')
plt.legend()
plt.show()
```

```
result_df.head(10)
```

	Date_time	DMA_225	tempC	HeatIndexC	FeelsLikeC	WindChillC	windspeedKmph	sunHour	precipMM	humidity	DMA_225_forecast
0	2016-06-17 02:00:00	1.11100	10	10	8	8	9	12.8	0.0	98	1.217522
1	2016-06-17 03:00:00	1.24975	10	10	8	8	10	12.8	0.0	98	1.360786
2	2016-06-17 04:00:00	1.33300	10	10	9	9	10	12.8	0.0	97	1.630748
3	2016-06-17 05:00:00	2.25000	11	11	10	10	10	12.8	0.0	95	2.218893
4	2016-06-17 06:00:00	4.19475	12	12	11	11	10	12.8	0.0	94	3.499895
5	2016-06-17 07:00:00	5.58325	12	12	11	11	11	12.8	0.0	93	4.992936
6	2016-06-17 08:00:00	5.16675	12	12	11	11	12	12.8	0.0	92	5.137409
7	2016-06-17 09:00:00	4.91675	12	12	11	11	14	12.8	0.0	91	4.895118
8	2016-06-17 10:00:00	4.19425	13	13	12	12	13	12.8	0.0	87	4.745715
9	2016-06-17 11:00:00	4.02775	13	13	12	12	13	12.8	0.0	84	4.600745