## 15 min ahead Code

User requests prediction data for

start date: 01-03-2020 12:00 AM

end Date: 01-04-2020 12:00 AM

### **validate\_start\_date(start\_date):**

return start\_date -12 hrs (i.e., 01-02-2020 12:00 PM)

### **prepare\_input(start\_date, end\_date, solar\_penetration, updated\_metric):**

Grabs all data from csv starting from time>= start\_date upto time< end\_date

Calculates the index positions for NaN values

If the metric is updated, replace the values where it is updated

Interpolate (linear) for the NaN values

Create sequence\_input (sequence length = 48)

Get y\_ground

Get temp, humidity, apparent\_power array

Create sequence y\_prev

Return sequence\_input, y\_ground, y\_prev [… others]

### **autoencoder\_func(sequence\_input, solar\_penetration):**

Use sequence\_input to fit on the Scaler1D() (= scaler\_target)

Use scaler\_target to transform sequence\_input to seq\_inp\_norm

Use seq\_inp\_norm on the encoder model to get pred\_train

Return pred\_train

### **kPF\_func(pred\_train, solar\_penetration):**

Return latent\_gen

### **lstm\_func(latent\_gen, sequence\_input, pred\_train, y\_ground, y\_prev, solar\_penetration):**

Getting the length of the sequence\_input( = total\_train)

Create a 2D matrix of zeroes with shape= (total\_train, 40) (=yyy)

First 20 columns of yyy is set to …

Next 20 (i.e., 20 to 40 columns) of yyy is set to …

Concatenate this with y\_prev after reshaping it into … (=yyy1)

Making a copy of y\_ground (=y\_train\_sol)

Concatenating yyy1 with y\_train\_sol by reshaping it into … (=total\_train\_data)

Use total\_train\_data to fit it on Sacler1D (=scaler\_target)

Use scaler\_target to transform total\_train\_data to total\_norm\_train

X = 0 to 40th column of total\_norm\_train, after reshaping it into (total\_norm\_train.shape[0],41,1)

Y = 41st column of total\_norm\_train

Use lstm\_model to predict using X (=y\_pred)

Multiply y\_pred by (difference between max and min of 41st column of total\_train\_data) and then the min

Multiply Y with the same factor and add the min to it (=Y\_test)

Calculate MAE and MAPE based on y\_pred and Y\_test

### **prepare\_output\_df(y\_pred, Y\_test, …):**

Create dataframes for net load and input variables

Return dictionary versions of these dataframes

### **processor():**

Call validate\_start\_date()

Call prepare\_input()

Call autoencoder\_func()

Call kPF\_func()

Call lstm\_func()

Call prepare\_output\_df()

Return the output in a dictionary format according to API specifications

## 24 hour ahead Code

User requests prediction data for

start date: 01-03-2020 12:00 AM

end Date: 01-04-2020 12:00 AM

### **validate\_start\_date(start\_date):**

return start\_date -12 hrs

### **get\_time\_intervals(start\_date, end\_date):**

edited\_start\_date = start\_date

edited\_end\_date = start\_date + 12 hrs

#### while(edited\_end\_date < end\_date):

Append [(edited\_start\_date -12), edited\_end\_date] to time\_intervals array

Increase edited\_start\_date and edited\_end\_date by 15 mins

#### Return time\_intervals\_array

### **prepare\_input(start\_date, end\_date, solar\_penetration, updated\_metric):**

Grabs all data from csv starting from time>= start\_date upto time< end\_date

If the metric is updated, replace the values where it is updated

Interpolate (linear) for the NaN values

Create sequence\_input (sequence length = 48)

Get y\_ground

Get temp, humidity, apparent\_power array

Create sequence y\_prev

Return sequence\_input, y\_ground, y\_prev [… others]

### **autoencoder\_func(sequence\_input, solar\_penetration):**

Use sequence\_input to fit on the Scaler1D() (= scaler\_target)

Use scaler\_target to transform sequence\_input to seq\_inp\_norm

Use seq\_inp\_norm on the encoder model to get pred\_train

Return pred\_train

### **kPF\_func(pred\_train, solar\_penetration):**

Return latent\_gen

### **lstm\_func(latent\_gen, sequence\_input, pred\_train, y\_ground, y\_prev, solar\_penetration):**

Getting the length of the sequence\_input( = total\_train)

Create a 2D matrix of zeroes with shape= (total\_train, 40) (=yyy)

First 20 columns of yyy is set to …

Next 20 (i.e., 20 to 40 columns) of yyy is set to …

Concatenate this with y\_prev after reshaping it into … (=yyy1)

Making a copy of y\_ground (=y\_train\_sol)

Concatenating yyy1 with y\_train\_sol by reshaping it into … (=total\_train\_data)

Use total\_train\_data to fit it on Sacler1D (=scaler\_target)

Use scaler\_target to transform total\_train\_data to total\_norm\_train

X = 0 to 40th column of total\_norm\_train, after reshaping it into (total\_norm\_train.shape[0],41,1)

Y = 41st column of total\_norm\_train

Use lstm\_model to predict using X (=y\_pred)

Multiply y\_pred by (difference between max and min of 41st column of total\_train\_data) and then the min

Multiply Y with the same factor and add the min to it (=Y\_test)

Return y\_pred and Y\_test

### **prepare\_output\_df(y\_pred\_mega, Y\_test\_mega, …):**

Calculate MAE and MAPE

Create dataframes for net load and input variables

Return dictionary versions of these dataframes along with MAE and MAPE

### **processor():**

Call get\_time\_intervals(validate\_start\_date(start\_date), end\_date)

#### For each time\_interval:

Call prepare\_input()

Call autoencoder\_func()

Call kPF\_func()

Call lstm\_func()

Save first value of y\_pred to another array (=y\_pred\_mega)

Do the same for Y\_test

#### Call prepare\_output\_df()

#### Return the output in a dictionary format according to API specifications