Passion_v7

August 1, 2020

1 Welcome to Passion!

Passion is a model that can detection anomaly using different methods (Both supervised and unsupervised)

1. The goal for this project is to study the difference between different anomnaly detection model, and to find the state of art method for detecting anomaly in real world data
2. Evaluate the results based on this :real server data+ https://www.kaggle.com/sohier/30-years-of-european-wind-generation (Unsupervised)+ https://github.com/numenta/NAB (Unsupervised+Supervised) https://www.cs.ucr.edu/~eamonn/time_series_data/ (Supervised)
3. Also use real data generated from server.
4. The model has the following fuctions:
    a. Visualize the input data. Help the user to find critical features within the inputs.
    b. Give user options to choose different models that are suitable for different circumstances.
    c. Evaluate the performance based on the rules in this link https://github.com/numenta/NAB
    d. Save model. Easy to be appplied to other dataset.
5. Add un-labeled and labeled data

2 What's new in version 6

1. Add Transformer in multi-variable time series prediction

In [1]:
```
# import packages


from matplotlib.pylab import rc
import torch
from scipy.stats import chisquare
from scipy.stats import pearsonr
import pickle
import pandas as pd
import datetime
import matplotlib
```

```
import tensorflow as tf
import sklearn
import math
import matplotlib.pyplot as plt
import xgboost
from xgboost import XGBClassifier
from xgboost import plot_importance
import numpy as np
from sklearn.model_selection import train_test_split
import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import copy
import scipy
import datetime
import time
import os
from sklearn.model_selection import KFold
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.covariance import EllipticEnvelope
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM
import gc
import json
plot_path = "plots/"
```

In [2]:
```
# Real server data (Unsupervised)

root_path = "Data/Ant_202007/"


cif = pd.read_json(root_path+'cif.json', orient='index')
paycore = pd.read_json(root_path+'paycore.json', orient='index')
paydecision = pd.read_json(root_path+'paydecision.json', orient='index')
paydecision2 = pd.read_json(root_path+'paydecision2.json', orient='index')
paydecision3 = pd.read_json(root_path+'paydecision3.json', orient='index')

df = pd.DataFrame()
df["time_stamp"] = cif.index
df["cif"] = cif[0].values
df["paycore"] = paycore[0].values
df["paydecision"] = paydecision[0].values
df["paydecision2"] = paydecision2[0].values
df["paydecision3"] = paydecision3[0].values
```

```
        # Optional
        if False:
            df.to_csv(root_path+"fusion.csv")


        # convert time stamp
        df['time_stamp'] = pd.to_datetime(df['time_stamp'])
        names_array = np.array(df.keys()[1:],dtype="str")
        os.listdir(root_path)
```

```
Out[2]: ['.ipynb_checkpoints',
         'cif.json',
         'fusion.csv',
         'paycore.json',
         'paydecision.json',
         'paydecision2.json',
         'paydecision3.json']
```

```
In [3]: if False:

            # calculate previous hour high low:
            # convert to seconds
            temp = df['time_stamp'] - min(df['time_stamp'])
            temp = temp.dt.total_seconds().astype(int)
            df["hours"] = temp//3600

            h_max = max(df["hours"])+1

            for n in range(len(names_array)):
                df[names_array[n]+"_open"] = df[names_array[n]]
                df[names_array[n]+"_close"] = df[names_array[n]]
                df[names_array[n]+"_max"] = df[names_array[n]]
                df[names_array[n]+"_min"] = df[names_array[n]]

            for j in range(1,h_max):
                mask_j = df["hours"]==j-1
                max_val = df[mask_j][names_array].max(axis=0).values
                min_val = df[mask_j][names_array].max(axis=0).values
                open_val = df[mask_j][names_array].values[0,:]
                close_val = df[mask_j][names_array].values[-1,:]
                mask_i = df["hours"]==j
                r = df[mask_i][names_array].shape[0]
                df.loc[mask_i,[r+"_open" for r in names_array]] = np.tile(open_val,(r,1))
                df.loc[mask_i,[r+"_close" for r in names_array]] = np.tile(close_val,(r,1))

                df.loc[mask_i,[r+"_max" for r in names_array]] = np.tile(max_val,(r,1))
                df.loc[mask_i,[r+"_min" for r in names_array]] = np.tile(min_val,(r,1))
```

```
names_array = list(df.keys())[1:]
```

In [4]:
```
# scale dot attention:

import tensorflow as tf
import os

from sklearn import preprocessing

from sklearn.model_selection import train_test_split




def scaled_dot_product_attention(q, k, v, mask):
    matmul_qk = tf.matmul(q, k, transpose_b=True)
    # Dimension of k
    dk = tf.cast(tf.shape(k)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)
    if mask is not None:
        scaled_attention_logits += (mask * -1e9)
    # calculate attention weight:
    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)
    output = tf.matmul(attention_weights, v)
    return output, attention_weights

# Multi-head Attention:
# This is what we use
class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads):

        # Always use Super to inheriatte and avoid extra code.
        assert d_model%num_heads==0
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.d_model = d_model
        # sanity check:
        assert d_model % self.num_heads == 0
        self.depth = d_model // self.num_heads
        # Q K W:
        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)

        self.dense = tf.keras.layers.Dense(d_model)
```

```
        def split_heads(self, x, batch_size):
            # Transpose the result such that the shape is (batch_size, num_heads, seq_len,
            x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
            return tf.transpose(x, perm=[0, 2, 1, 3])
        def call(self, v, k, q, mask):
            batch_size = tf.shape(q)[0]
            q = self.wq(q)  # (batch_size, seq_len, d_model)
            k = self.wk(k)  # (batch_size, seq_len, d_model)
            v = self.wv(v)  # (batch_size, seq_len, d_model)

            q = self.split_heads(q, batch_size)  # (batch_size, num_heads, seq_len_q, depti
            k = self.split_heads(k, batch_size)  # (batch_size, num_heads, seq_len_k, depti
            v = self.split_heads(v, batch_size)  # (batch_size, num_heads, seq_len_v, depti

            # scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
            # attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)

            scaled_attention, attention_weights = scaled_dot_product_attention(q, k, v, mas
            # https://www.tensorflow.org/api_docs/python/tf/transpose : perm
            scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])  # (batch_
            concat_attention = tf.reshape(scaled_attention,
                                          (batch_size, -1, self.d_model))  # (batch_size, seq_
            output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)
            return output, attention_weights
```

In [5]:
```
## Encoder decoder for Time series:

    # pointwise feed forward network
    def point_wise_feed_forward_network(d_model, dff):
        # Two FC layers:
        return tf.keras.Sequential([
          tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size, seq_len, dff)
          tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
      ])

    # Change embedding since it's not int anymore:
    class EmbeddingLayer(tf.keras.layers.Layer):
        def __init__(self,embedding_size):
            super(EmbeddingLayer,self).__init__()
            self.embedding_size=embedding_size

        def build(self,input_shape):
            with tf.name_scope('embedding'):
                self.shared_weights=self.add_weight(name='weights',
```

```
                                                shape=[input_shape[-1],self.embedding_s
                                                initializer=tf.random_normal_initializ

        super(EmbeddingLayer,self).build(input_shape)


    def call(self,x):
        y=tf.einsum('bsf,fk->bsk',x,self.shared_weights)
        return y


class EncoderLayer(tf.keras.layers.Layer):
    # Here we use a 0.1 dropout rate as default
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(EncoderLayer, self).__init__()
        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):
        attn_output, _ = self.mha(x, x, x, mask)  # (batch_size, input_seq_len, d_mode
        attn_output = self.dropout1(attn_output, training=training)

        out1 = self.layernorm1(x + attn_output)  # (batch_size, input_seq_len, d_model
        ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
        ffn_output = self.dropout2(ffn_output, training=training)

        out2 = self.layernorm2(out1 + ffn_output)  # (batch_size, input_seq_len, d_mod

        return out2
sample_encoder_layer = EncoderLayer(512, 8, 2048)

sample_encoder_layer_output = sample_encoder_layer(tf.random.uniform((64, 43, 512)), Fa

print(sample_encoder_layer_output.shape)  # (batch_size, input_seq_len, d_model)

class DecoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(DecoderLayer, self).__init__()

        self.mha1 = MultiHeadAttention(d_model, num_heads)
        self.mha2 = MultiHeadAttention(d_model, num_heads)
```

```
        self.ffn = point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)
        self.dropout3 = tf.keras.layers.Dropout(rate)

    def call(self, x, enc_output, training, look_ahead_mask, padding_mask):
        # enc_output.shape == (batch_size, input_seq_len, d_model)

        attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)  # (batch_siz
        attn1 = self.dropout1(attn1, training=training)
        out1 = self.layernorm1(attn1 + x)

        attn2, attn_weights_block2 = self.mha2(
            enc_output, enc_output, out1, padding_mask)  # (batch_size, target_seq_len
        attn2 = self.dropout2(attn2, training=training)
        out2 = self.layernorm2(attn2 + out1)  # (batch_size, target_seq_len, d_model)

        ffn_output = self.ffn(out2)  # (batch_size, target_seq_len, d_model)
        ffn_output = self.dropout3(ffn_output, training=training)
        out3 = self.layernorm3(ffn_output + out2)  # (batch_size, target_seq_len, d_mo

        return out3, attn_weights_block1, attn_weights_block2
```

```
(64, 43, 512)
```

```
In [6]: def get_angles(pos, i, d_model):
            angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_model))
            return pos * angle_rates

        def positional_encoding(position, d_model):
            angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                                    np.arange(d_model)[np.newaxis, :],
                                    d_model)

            # apply sin to even indices in the array; 2i
            angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

            # apply cos to odd indices in the array; 2i+1
            angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
            pos_encoding = angle_rads[np.newaxis, ...]
            return tf.cast(pos_encoding, dtype=tf.float32)
```

```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
                 maximum_position_encoding, rate=0.1):
        super(Encoder, self).__init__()

        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = tf.keras.layers.Embedding(input_vocab_size, d_model)
        self.pos_encoding = positional_encoding(maximum_position_encoding,
                                                self.d_model)


        self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                           for _ in range(num_layers)]

        self.dropout = tf.keras.layers.Dropout(rate)

    def call(self, x, training, mask):
        seq_len = tf.shape(x)[1]

        # adding embedding and position encoding.
        #print("Check",x.shape)
        x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
        #x = tf.keras.layers.Dense(self.d_model)(x)
        #print("check 2",x.shape)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]
        #print("check 3",x.shape)

        x = self.dropout(x, training=training)
        #print("check 4",x.shape)

        for i in range(self.num_layers):
            x = self.enc_layers[i](x, training, mask)
        return x  # (batch_size, input_seq_len, d_model)

class Decoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, d_model, num_heads, dff, target_vocab_size,
                 maximum_position_encoding, rate=0.1):
        super(Decoder, self).__init__()
        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = tf.keras.layers.Embedding(target_vocab_size, d_model)
        self.pos_encoding = positional_encoding(maximum_position_encoding, d_model)

        self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
```

```
                                    for _ in range(num_layers)]
        self.dropout = tf.keras.layers.Dropout(rate)

    def call(self, x, enc_output, training, look_ahead_mask, padding_mask):
        seq_len = tf.shape(x)[1]
        attention_weights = {}

        x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
        #x = tf.keras.layers.Dense(self.d_model)(x)

        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]

        x = self.dropout(x, training=training)
        for i in range(self.num_layers):
            x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                                   look_ahead_mask, padding_mask)

            attention_weights['decoder_layer{}_block1'.format(i+1)] = block1
            attention_weights['decoder_layer{}_block2'.format(i+1)] = block2

        # x.shape == (batch_size, target_seq_len, d_model)
        return x, attention_weights


In [7]: class Transformer(tf.keras.Model):
    def __init__(self, num_layers, d_model, num_heads, dff, input_seq_size,
                 output_seq_size, input_delta_t, output_delta_t, rate=0.1):
        super(Transformer, self).__init__()

        self.encoder = Encoder(num_layers, d_model, num_heads, dff,
                               input_seq_size, input_delta_t, rate)

        self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                               output_seq_size, output_delta_t, rate)

        # Now it output one cell: we ignore sigma for now and only miu

        #self.final_layer = tf.keras.layers.Dense(output_seq_size)
        self.final_layer = tf.keras.layers.Dense(1)

        # Optional: Add sigma to model
        #self.final_layer_sigma = tf.keras.layers.Dense(1)


    def call(self, inp, tar, training, enc_padding_mask,
             look_ahead_mask, dec_padding_mask):
        enc_output = self.encoder(inp, training, enc_padding_mask)  # (batch_size, inp
```

```
        #print("check encoder size",enc_output.shape)


        # dec_output.shape == (batch_size, tar_seq_len, d_model)
        dec_output, attention_weights = self.decoder(
            tar, enc_output, training, look_ahead_mask, dec_padding_mask)

        #print("check decoder size",dec_output.shape)

        final_output = self.final_layer(dec_output)  # (batch_size, tar_seq_len, targe

        return final_output, attention_weights
```

In [10]:
```
# sanity check:
# We encoder the float32 input to input_seq_size/output_seq_size integers
# The output is a sliding time table for different time scale prediction:
# Eg: you need to make sure your prediction delta_t<output delta_t and input data del
# For GTX 1060 we can set batch=16 and use 4X batch size for Tesla P40

batch = 8

sample_transformer = Transformer(
    num_layers=2, d_model=512, num_heads=8, dff=2048,
    input_seq_size=1000, output_seq_size=1000,
    input_delta_t=1440, output_delta_t=240)

# input: batch+sequence length
# biggest length for in/out put is pe_input,  pe_target
temp_input = tf.random.uniform((batch, 720), dtype=tf.int64, minval=0, maxval=1000)
temp_target = tf.random.uniform((batch, 3), dtype=tf.int64, minval=0, maxval=1000)

#temp_input = tf.cast(temp_input,dtype=tf.float32)
#temp_target = tf.cast(temp_target,dtype=tf.float32)

fn_out, _ = sample_transformer(temp_input, temp_target, training=False,
                               enc_padding_mask=None,
                               look_ahead_mask=None,
                               dec_padding_mask=None)

print("final output size",fn_out.shape)  # (batch_size, tar_seq_len, target_vocab_siz
```

```
final output size (8, 3, 1)
```

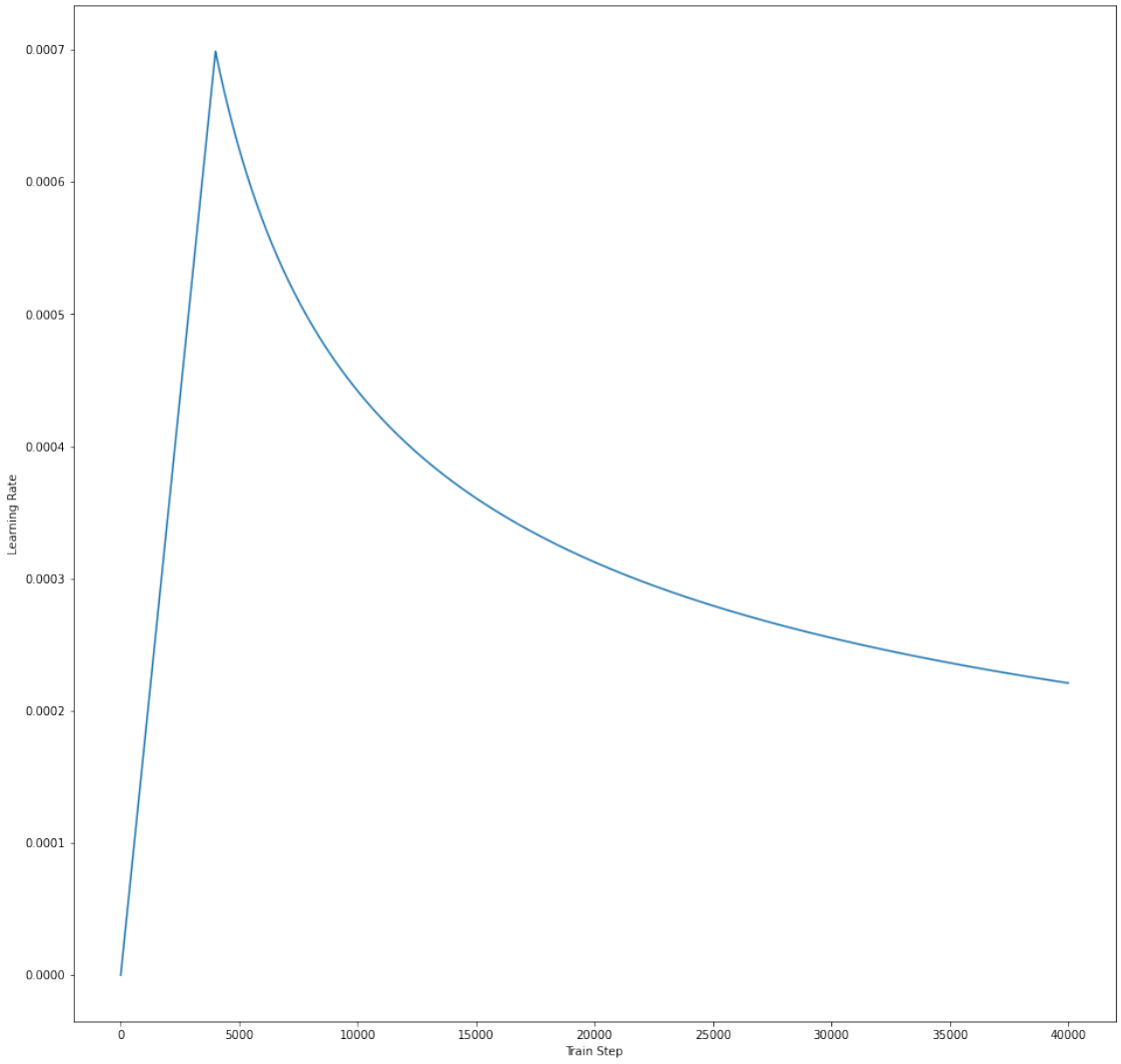
In [82]:
```
# Let's do a three variables version

temp = df[names_array].values
```

```
        # normalize first

        temp = (temp - temp.min(axis=0)) / (temp.max(axis=0) - temp.min(axis=0))

        lower, upper = 0, 999
        temp = lower + (upper - lower) * temp
        temp = np.array(temp,dtype=int)

        # Longer peroid

        delta_t = 420
        delta_t_out = 60



        # prepare data: fow now I only use 1D data, but it can be extended to multiple channe

        # Normalize to 0-1000

        #X = np.zeros((temp.shape[0]-delta_t-delta_t_out,delta_t*temp.shape[1]),dtype=int)


        for j in range(temp.shape[1]):
            for i in range(delta_t_out):
                if i==0 and j==0:
                    y = temp[delta_t:-delta_t_out,j]
                else:
                    y = np.c_[y,temp[delta_t+i:-(delta_t_out-i),j]]



        for j in range(temp.shape[1]):
            for i in range(delta_t):
                if i%300==0:
                    print("Doing %.2f percent"%((100*i+100*j*delta_t)/delta_t/temp.shape[1]))
                if i==0 and j==0:
                    X = temp[delta_t_out:-delta_t,j]
                else:
                    X = np.c_[X,temp[delta_t_out+i:-(delta_t-i),j]]



        X = np.atleast_3d(X)
        print("Data ready")
        #train_dataset_TS = tf.data.Dataset.from_tensor_slices((X,y))
```

```
Doing 0.00 percent
Doing 14.29 percent
```

```
Doing 0.05 percent
Doing 14.33 percent
Doing 0.10 percent
Doing 14.38 percent
Doing 0.14 percent
Doing 14.43 percent
Doing 0.19 percent
Doing 14.48 percent
Data ready
```

In [89]:
```
## Optimizor:
import matplotlib.pyplot as plt

d_model=512

class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
    def __init__(self, d_model, warmup_steps=4000):
        super(CustomSchedule, self).__init__()

        self.d_model = d_model
        self.d_model = tf.cast(self.d_model, tf.float32)

        self.warmup_steps = warmup_steps

    def __call__(self, step):
        arg1 = tf.math.rsqrt(step)
        arg2 = step * (self.warmup_steps ** -1.5)

        return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)

learning_rate = CustomSchedule(d_model)

optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,
                                     epsilon=1e-9)
# Learning rate curve:
temp_learning_rate_schedule = CustomSchedule(d_model)

plt.plot(temp_learning_rate_schedule(tf.range(40000, dtype=tf.float32)))
plt.ylabel("Learning Rate")
plt.xlabel("Train Step")
fig = matplotlib.pyplot.gcf()

fig.set_size_inches(16,16)
plt.show()
```



In [90]:
```
# Loss function:
# loss and metric

# For now I use sparse-cross entropy. But MAE may make more sense here:

loss_object = tf.keras.losses.MeanSquaredError(reduction='none')


def loss_function(real, pred):
    #mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    #mask = tf.cast(mask, dtype=loss_.dtype)
    #loss_ *= mask
```

```
        return tf.reduce_sum(loss_)/tf.cast(len(loss_),dtype=tf.float32)



    train_loss = tf.keras.metrics.Mean(name='train_loss')


    train_accuracy = tf.keras.metrics.MeanSquaredError(name='mean_squared_error',dtype=tf



    # Optional
    #train_accuracy = tf.keras.metrics.MeanSquaredError(name='train_MSE')
```

In [91]:
```
    def create_padding_mask(seq):
        seq = tf.cast(tf.math.equal(seq, 0), tf.float32)

        # add extra dimensions to add the padding
        # to the attention logits.
        return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)

    def create_look_ahead_mask(size):
        mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
        return mask  # (seq_len, seq_len)

    def create_masks(inp, tar):
        # Encoder padding mask
        enc_padding_mask = create_padding_mask(inp)

        # Used in the 2nd attention block in the decoder.
        # This padding mask is used to mask the encoder outputs.
        dec_padding_mask = create_padding_mask(inp)

        # Used in the 1st attention block in the decoder.
        # It is used to pad and mask future tokens in the input received by
        # the decoder.
        look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
        dec_target_padding_mask = create_padding_mask(tar)
        combined_mask = tf.maximum(dec_target_padding_mask, look_ahead_mask)


        return enc_padding_mask, combined_mask, dec_padding_mask
```

In [92]:
```
    batch = 8

    transformer = Transformer(
        num_layers=2, d_model=512, num_heads=8, dff=2048,
```

```
    input_seq_size=1000, output_seq_size=1000,
    input_delta_t=2400, output_delta_t=300)



# save file: optional
import os

checkpoint_path = "checkpoints/train_TS_CIF"
os.system("mkdir %s"%checkpoint_path)

ckpt = tf.train.Checkpoint(transformer=transformer,
                           optimizer=optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
    ckpt.restore(ckpt_manager.latest_checkpoint)
    print ('Latest checkpoint restored!!')



train_step_signature = [
        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
    ]

@tf.function(input_signature=train_step_signature)

def train_step(inp, tar):

    tar_inp = tar
    tar_real = tar

    enc_padding_mask, combined_mask, dec_padding_mask = create_masks(inp, tar_inp)
    with tf.GradientTape() as tape:
        # No mask for now : Optional
        enc_padding_mask, combined_mask, dec_padding_mask = None,None,None
        predictions, _ = transformer(inp, tar_inp, True, enc_padding_mask, combined_ma
        predictions = predictions[:,:,0]
        loss = loss_function(tar_real, predictions)
        ## Optional: Add MSE error term. Since the number in SCCE doesn't make sense.
        #predictions_id = tf.argmax(predictions, axis=-1)
        #loss+=float(tf.reduce_sum(tf.keras.losses.MSE(tar,predictions_id))/(10000*ba
        #value = float(tf.reduce_sum(tf.keras.losses.MSE(tar,predictions_id))/(1*batc
        # Avoid gradient exploding
        """
```

```
                    if not loss>0:
                        value=float(100000)
                    loss+=value

                    """


                    # Or we can only use MSE loss.

                gradients = tape.gradient(loss, transformer.trainable_variables)
                optimizer.apply_gradients(zip(gradients, transformer.trainable_variables))

                train_loss(loss)
                train_accuracy(tar_real, predictions)
```

In [94]:
```
#Train and save:

import time
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

EPOCHS = 20
train_dataset = tf.data.Dataset.from_tensor_slices((X_train,y_train))
batch=4

N = len(y_train)

for epoch in range(EPOCHS):
    start = time.time()

    train_loss.reset_states()
    train_accuracy.reset_states()
    for i in range(N//batch):
        inp, tar=X_train[batch*i:min(batch*i+batch,N),:,0],y_train[batch*i:min(batch*i
        tar = np.atleast_2d(tar)
        lo = train_step(inp, tar)
        if i%500==0 and epoch%2==0:
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, s

            print("Doing %d (%d) batch in epoch %d "%(i,N//batch,epoch))

            #print("Loss",train_loss.result(), "MSE",train_accuracy.result())
            print("MSE",train_accuracy.result())
```

```
Doing 0 (7224) batch in epoch 0
MSE tf.Tensor(1557.6375, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 0
```

```
MSE tf.Tensor(456.18573, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 0
MSE tf.Tensor(437.0367, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 0
MSE tf.Tensor(425.07895, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 0
MSE tf.Tensor(434.84268, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 0
MSE tf.Tensor(412.7663, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 0
MSE tf.Tensor(418.66693, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 0
MSE tf.Tensor(384.3739, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 0
MSE tf.Tensor(392.38623, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 0
MSE tf.Tensor(390.80124, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 0
MSE tf.Tensor(373.64188, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 0
MSE tf.Tensor(369.07153, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 0
MSE tf.Tensor(356.31006, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 0
MSE tf.Tensor(350.43042, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 0
MSE tf.Tensor(351.46066, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 1
MSE tf.Tensor(1734.0157, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 1
MSE tf.Tensor(317.74295, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 1
MSE tf.Tensor(299.07175, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 1
MSE tf.Tensor(303.45477, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 1
MSE tf.Tensor(299.81357, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 1
MSE tf.Tensor(299.84183, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 1
MSE tf.Tensor(331.77292, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 1
MSE tf.Tensor(323.7115, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 1
MSE tf.Tensor(378.50113, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 1
MSE tf.Tensor(379.33463, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 1
```

```
MSE tf.Tensor(362.96402, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 1
MSE tf.Tensor(367.7256, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 1
MSE tf.Tensor(361.24255, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 1
MSE tf.Tensor(348.96536, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 1
MSE tf.Tensor(339.54065, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 2
MSE tf.Tensor(1990.4069, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 2
MSE tf.Tensor(205.6409, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 2
MSE tf.Tensor(198.5994, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 2
MSE tf.Tensor(193.40195, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 2
MSE tf.Tensor(214.59586, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 2
MSE tf.Tensor(248.38918, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 2
MSE tf.Tensor(281.56534, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 2
MSE tf.Tensor(274.00064, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 2
MSE tf.Tensor(321.4357, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 2
MSE tf.Tensor(324.1449, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 2
MSE tf.Tensor(316.32407, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 2
MSE tf.Tensor(320.79355, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 2
MSE tf.Tensor(310.98016, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 2
MSE tf.Tensor(296.21188, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 2
MSE tf.Tensor(287.66946, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 3
MSE tf.Tensor(396.77655, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 3
MSE tf.Tensor(128.32188, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 3
MSE tf.Tensor(189.5269, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 3
MSE tf.Tensor(194.67087, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 3
```

```
MSE tf.Tensor(221.40767, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 3
MSE tf.Tensor(293.4881, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 3
MSE tf.Tensor(303.8386, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 3
MSE tf.Tensor(273.3004, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 3
MSE tf.Tensor(333.3386, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 3
MSE tf.Tensor(314.10727, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 3
MSE tf.Tensor(296.3349, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 3
MSE tf.Tensor(288.6793, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 3
MSE tf.Tensor(274.14038, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 3
MSE tf.Tensor(258.13098, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 3
MSE tf.Tensor(248.66708, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 4
MSE tf.Tensor(286.36206, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 4
MSE tf.Tensor(82.30283, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 4
MSE tf.Tensor(106.928825, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 4
MSE tf.Tensor(107.15357, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 4
MSE tf.Tensor(130.14264, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 4
MSE tf.Tensor(148.1889, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 4
MSE tf.Tensor(156.62164, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 4
MSE tf.Tensor(141.46382, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 4
MSE tf.Tensor(175.5232, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 4
MSE tf.Tensor(172.63779, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 4
MSE tf.Tensor(161.9429, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 4
MSE tf.Tensor(161.18356, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 4
MSE tf.Tensor(153.29314, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 4
```

```
MSE tf.Tensor(144.99791, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 4
MSE tf.Tensor(139.73366, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 5
MSE tf.Tensor(328.4469, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 5
MSE tf.Tensor(54.90034, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 5
MSE tf.Tensor(75.132576, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 5
MSE tf.Tensor(76.24584, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 5
MSE tf.Tensor(94.2626, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 5
MSE tf.Tensor(101.8204, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 5
MSE tf.Tensor(102.489525, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 5
MSE tf.Tensor(93.85098, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 5
MSE tf.Tensor(101.85342, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 5
MSE tf.Tensor(99.48083, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 5
MSE tf.Tensor(94.283806, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 5
MSE tf.Tensor(94.909546, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 5
MSE tf.Tensor(90.72022, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 5
MSE tf.Tensor(86.34381, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 5
MSE tf.Tensor(82.77667, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 6
MSE tf.Tensor(227.72322, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 6
MSE tf.Tensor(41.6291, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 6
MSE tf.Tensor(68.17322, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 6
MSE tf.Tensor(67.73288, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 6
MSE tf.Tensor(67.38165, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 6
MSE tf.Tensor(79.08902, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 6
MSE tf.Tensor(81.8315, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 6
```

```
MSE tf.Tensor(74.93177, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 6
MSE tf.Tensor(78.54248, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 6
MSE tf.Tensor(76.55807, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 6
MSE tf.Tensor(72.51572, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 6
MSE tf.Tensor(74.85602, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 6
MSE tf.Tensor(71.499344, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 6
MSE tf.Tensor(68.04528, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 6
MSE tf.Tensor(65.58096, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 7
MSE tf.Tensor(105.394516, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 7
MSE tf.Tensor(29.15246, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 7
MSE tf.Tensor(38.980717, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 7
MSE tf.Tensor(40.054035, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 7
MSE tf.Tensor(43.65205, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 7
MSE tf.Tensor(98.552315, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 7
MSE tf.Tensor(108.95531, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 7
MSE tf.Tensor(96.989685, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 7
MSE tf.Tensor(100.701035, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 7
MSE tf.Tensor(95.76961, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 7
MSE tf.Tensor(89.91083, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 7
MSE tf.Tensor(89.69798, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 7
MSE tf.Tensor(84.774185, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 7
MSE tf.Tensor(80.299965, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 7
MSE tf.Tensor(76.55547, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 8
MSE tf.Tensor(44.992516, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 8
```

```
MSE tf.Tensor(33.2646, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 8
MSE tf.Tensor(40.731583, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 8
MSE tf.Tensor(42.72421, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 8
MSE tf.Tensor(43.85545, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 8
MSE tf.Tensor(59.470757, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 8
MSE tf.Tensor(58.71805, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 8
MSE tf.Tensor(54.873432, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 8
MSE tf.Tensor(61.59323, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 8
MSE tf.Tensor(65.636536, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 8
MSE tf.Tensor(61.959927, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 8
MSE tf.Tensor(63.405254, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 8
MSE tf.Tensor(60.471313, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 8
MSE tf.Tensor(57.850803, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 8
MSE tf.Tensor(55.629044, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 9
MSE tf.Tensor(75.30797, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 9
MSE tf.Tensor(21.957064, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 9
MSE tf.Tensor(29.526123, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 9
MSE tf.Tensor(31.462536, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 9
MSE tf.Tensor(34.62655, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 9
MSE tf.Tensor(42.93526, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 9
MSE tf.Tensor(45.507053, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 9
MSE tf.Tensor(42.32367, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 9
MSE tf.Tensor(47.557106, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 9
MSE tf.Tensor(49.76052, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 9
```

```
MSE tf.Tensor(47.103043, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 9
MSE tf.Tensor(47.981003, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 9
MSE tf.Tensor(45.757515, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 9
MSE tf.Tensor(43.57795, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 9
MSE tf.Tensor(41.854565, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 10
MSE tf.Tensor(79.50717, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 10
MSE tf.Tensor(17.591572, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 10
MSE tf.Tensor(22.445482, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 10
MSE tf.Tensor(23.030912, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 10
MSE tf.Tensor(32.777657, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 10
MSE tf.Tensor(36.97348, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 10
MSE tf.Tensor(40.340733, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 10
MSE tf.Tensor(37.982983, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 10
MSE tf.Tensor(41.45149, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 10
MSE tf.Tensor(43.006992, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 10
MSE tf.Tensor(41.058918, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 10
MSE tf.Tensor(41.461937, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 10
MSE tf.Tensor(39.49996, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 10
MSE tf.Tensor(38.187706, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 10
MSE tf.Tensor(36.791958, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 11
MSE tf.Tensor(80.83739, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 11
MSE tf.Tensor(21.096878, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 11
MSE tf.Tensor(21.021, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 11
MSE tf.Tensor(21.529142, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 11
```

```
MSE tf.Tensor(22.357042, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 11
MSE tf.Tensor(30.332518, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 11
MSE tf.Tensor(31.780802, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 11
MSE tf.Tensor(29.512009, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 11
MSE tf.Tensor(32.119057, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 11
MSE tf.Tensor(32.382935, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 11
MSE tf.Tensor(30.706596, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 11
MSE tf.Tensor(31.331934, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 11
MSE tf.Tensor(30.156914, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 11
MSE tf.Tensor(28.852682, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 11
MSE tf.Tensor(27.824682, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 12
MSE tf.Tensor(47.90979, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 12
MSE tf.Tensor(13.727204, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 12
MSE tf.Tensor(16.08337, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 12
MSE tf.Tensor(17.806574, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 12
MSE tf.Tensor(18.601719, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 12
MSE tf.Tensor(33.41841, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 12
MSE tf.Tensor(31.587156, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 12
MSE tf.Tensor(29.15336, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 12
MSE tf.Tensor(32.663185, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 12
MSE tf.Tensor(44.654224, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 12
MSE tf.Tensor(41.67623, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 12
MSE tf.Tensor(41.9874, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 12
MSE tf.Tensor(39.91877, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 12
```

```
MSE tf.Tensor(37.82125, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 12
MSE tf.Tensor(35.971043, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 13
MSE tf.Tensor(49.932472, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 13
MSE tf.Tensor(11.632861, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 13
MSE tf.Tensor(15.2532215, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 13
MSE tf.Tensor(17.159185, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 13
MSE tf.Tensor(19.53133, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 13
MSE tf.Tensor(40.730755, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 13
MSE tf.Tensor(38.08206, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 13
MSE tf.Tensor(34.28149, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 13
MSE tf.Tensor(33.825634, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 13
MSE tf.Tensor(35.698475, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 13
MSE tf.Tensor(33.3946, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 13
MSE tf.Tensor(32.75891, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 13
MSE tf.Tensor(31.102537, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 13
MSE tf.Tensor(29.618559, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 13
MSE tf.Tensor(28.429422, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 14
MSE tf.Tensor(27.766077, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 14
MSE tf.Tensor(14.094536, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 14
MSE tf.Tensor(13.391789, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 14
MSE tf.Tensor(14.969115, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 14
MSE tf.Tensor(15.278382, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 14
MSE tf.Tensor(19.714092, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 14
MSE tf.Tensor(20.727362, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 14
```

```
MSE tf.Tensor(19.62329, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 14
MSE tf.Tensor(20.542944, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 14
MSE tf.Tensor(24.43969, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 14
MSE tf.Tensor(23.03545, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 14
MSE tf.Tensor(23.104023, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 14
MSE tf.Tensor(22.163523, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 14
MSE tf.Tensor(21.250925, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 14
MSE tf.Tensor(20.4046, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 15
MSE tf.Tensor(36.133995, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 15
MSE tf.Tensor(11.3775015, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 15
MSE tf.Tensor(14.816397, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 15
MSE tf.Tensor(14.95517, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 15
MSE tf.Tensor(14.590797, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 15
MSE tf.Tensor(20.974201, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 15
MSE tf.Tensor(21.21553, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 15
MSE tf.Tensor(19.823044, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 15
MSE tf.Tensor(22.139305, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 15
MSE tf.Tensor(23.535707, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 15
MSE tf.Tensor(22.137453, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 15
MSE tf.Tensor(22.110207, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 15
MSE tf.Tensor(21.128195, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 15
MSE tf.Tensor(20.643085, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 15
MSE tf.Tensor(19.7948, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 16
MSE tf.Tensor(19.634312, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 16
```

```
MSE tf.Tensor(9.560932, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 16
MSE tf.Tensor(10.189792, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 16
MSE tf.Tensor(11.056605, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 16
MSE tf.Tensor(12.7770815, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 16
MSE tf.Tensor(28.226444, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 16
MSE tf.Tensor(27.020905, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 16
MSE tf.Tensor(24.346853, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 16
MSE tf.Tensor(26.46442, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 16
MSE tf.Tensor(27.468512, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 16
MSE tf.Tensor(25.795319, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 16
MSE tf.Tensor(25.27831, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 16
MSE tf.Tensor(23.93529, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 16
MSE tf.Tensor(22.66719, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 16
MSE tf.Tensor(21.592422, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 17
MSE tf.Tensor(25.81221, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 17
MSE tf.Tensor(8.841332, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 17
MSE tf.Tensor(9.16017, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 17
MSE tf.Tensor(9.969313, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 17
MSE tf.Tensor(12.722966, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 17
MSE tf.Tensor(14.54161, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 17
MSE tf.Tensor(14.671089, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 17
MSE tf.Tensor(13.7319355, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 17
MSE tf.Tensor(17.574074, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 17
MSE tf.Tensor(19.530985, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 17
```

```
MSE tf.Tensor(18.375368, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 17
MSE tf.Tensor(18.378239, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 17
MSE tf.Tensor(17.58805, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 17
MSE tf.Tensor(16.76518, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 17
MSE tf.Tensor(16.128517, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 18
MSE tf.Tensor(23.068724, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 18
MSE tf.Tensor(8.172438, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 18
MSE tf.Tensor(8.954523, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 18
MSE tf.Tensor(9.461392, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 18
MSE tf.Tensor(9.953835, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 18
MSE tf.Tensor(15.334615, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 18
MSE tf.Tensor(15.739411, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 18
MSE tf.Tensor(14.553854, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 18
MSE tf.Tensor(16.406599, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 18
MSE tf.Tensor(17.838398, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 18
MSE tf.Tensor(16.772686, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 18
MSE tf.Tensor(16.682568, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 18
MSE tf.Tensor(16.00891, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 18
MSE tf.Tensor(15.275033, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 18
MSE tf.Tensor(14.700355, shape=(), dtype=float32)
Doing 0 (7224) batch in epoch 19
MSE tf.Tensor(22.166698, shape=(), dtype=float32)
Doing 500 (7224) batch in epoch 19
MSE tf.Tensor(7.7081866, shape=(), dtype=float32)
Doing 1000 (7224) batch in epoch 19
MSE tf.Tensor(8.7652235, shape=(), dtype=float32)
Doing 1500 (7224) batch in epoch 19
MSE tf.Tensor(9.116581, shape=(), dtype=float32)
Doing 2000 (7224) batch in epoch 19
```

```
MSE tf.Tensor(9.191524, shape=(), dtype=float32)
Doing 2500 (7224) batch in epoch 19
MSE tf.Tensor(9.958986, shape=(), dtype=float32)
Doing 3000 (7224) batch in epoch 19
MSE tf.Tensor(10.9242, shape=(), dtype=float32)
Doing 3500 (7224) batch in epoch 19
MSE tf.Tensor(10.273735, shape=(), dtype=float32)
Doing 4000 (7224) batch in epoch 19
MSE tf.Tensor(21.596085, shape=(), dtype=float32)
Doing 4500 (7224) batch in epoch 19
MSE tf.Tensor(22.659357, shape=(), dtype=float32)
Doing 5000 (7224) batch in epoch 19
MSE tf.Tensor(21.056007, shape=(), dtype=float32)
Doing 5500 (7224) batch in epoch 19
MSE tf.Tensor(21.039354, shape=(), dtype=float32)
Doing 6000 (7224) batch in epoch 19
MSE tf.Tensor(20.111843, shape=(), dtype=float32)
Doing 6500 (7224) batch in epoch 19
MSE tf.Tensor(19.08461, shape=(), dtype=float32)
Doing 7000 (7224) batch in epoch 19
MSE tf.Tensor(18.205341, shape=(), dtype=float32)
```

```
In [95]: # testing:
         N_test = len(y_test)


         for i in range(N_test//batch):
             if i%200==0:
                     print("Doing %d (%d)"%(i,N_test//batch))

             inp, tar=X_test[batch*i:min(batch*i+batch,N),:,0],y_test[batch*i:min(batch*i+batch
             tar = tar
             tar_inp = tar
             tar_real = tar

             # enc_padding_mask, combined_mask, dec_padding_mask = None,None,None

             predictions, attention_weights = transformer(inp,
                                                          tar,
                                                          False,
                                                          None,None,None)
             if i==0:
                 y_pred_all = predictions
             else:
                 y_pred_all = np.r_[y_pred_all,predictions]
```

```
        y_pred_all = np.array(y_pred_all)

        print("Train+Test all set!")
```

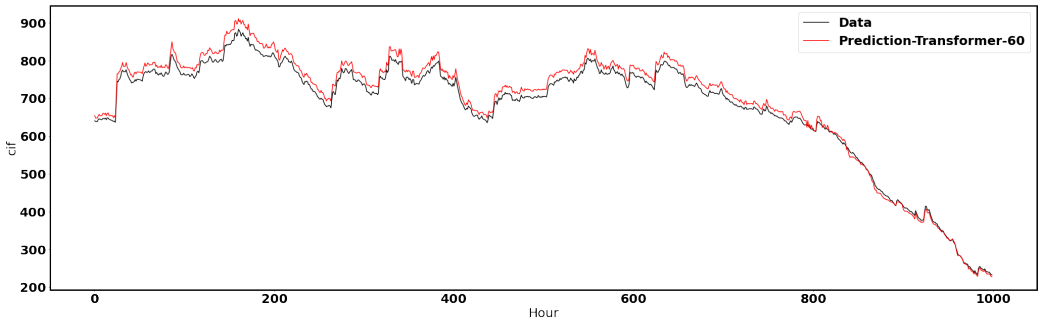
```
Doing 0 (3096)
Doing 200 (3096)
Doing 400 (3096)
Doing 600 (3096)
Doing 800 (3096)
Doing 1000 (3096)
Doing 1200 (3096)
Doing 1400 (3096)
Doing 1600 (3096)
Doing 1800 (3096)
Doing 2000 (3096)
Doing 2200 (3096)
Doing 2400 (3096)
Doing 2600 (3096)
Doing 2800 (3096)
Doing 3000 (3096)
Train+Test all set!
```

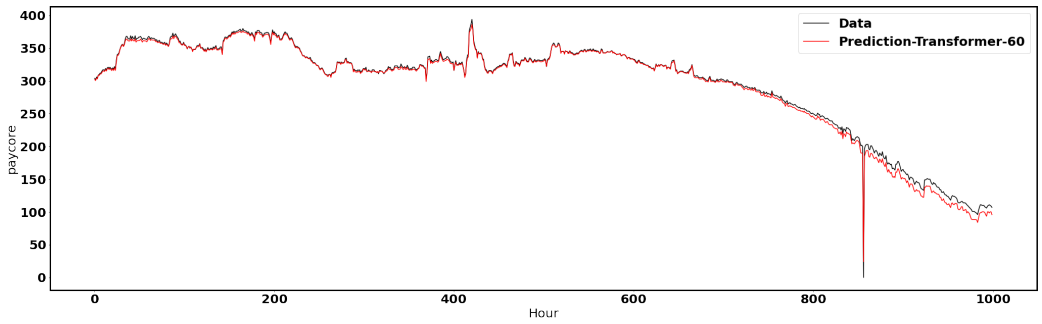
```
In [96]: # Convert testing to -1*60*3
         np.save("y_pred_all_60_a5.npy",y_pred_all)
         np.save("y_test_all_60_a5.npy",y_test)
```

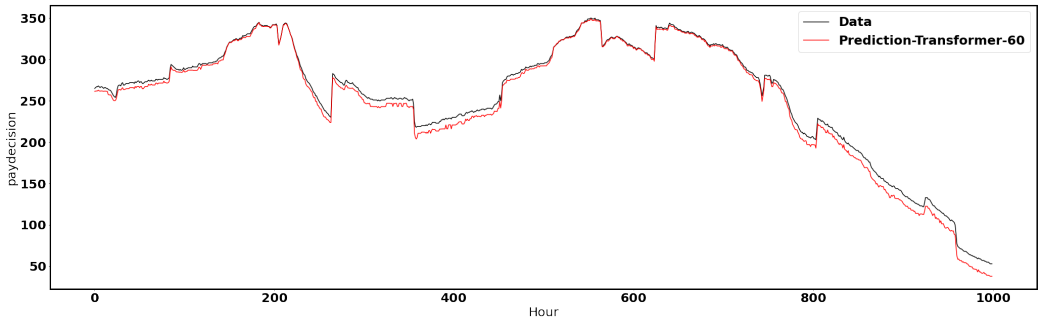
```
In [101]: y_pred_all = y_pred_all[:,:,0]
          plot_path = "plots/"

          y_test = y_test[:y_pred_all.shape[0]]


          import matplotlib
          from matplotlib.pylab import rc


          font = {'family': 'normal','weight': 'bold',
                  'size': 25}

          matplotlib.rc('font', **font)
          rc('axes', linewidth=3)

          bin_size = y_pred_all.shape[1]//temp.shape[1]

          for i in range(temp.shape[1]):
              plt.subplot(temp.shape[1],1,1+i)
```

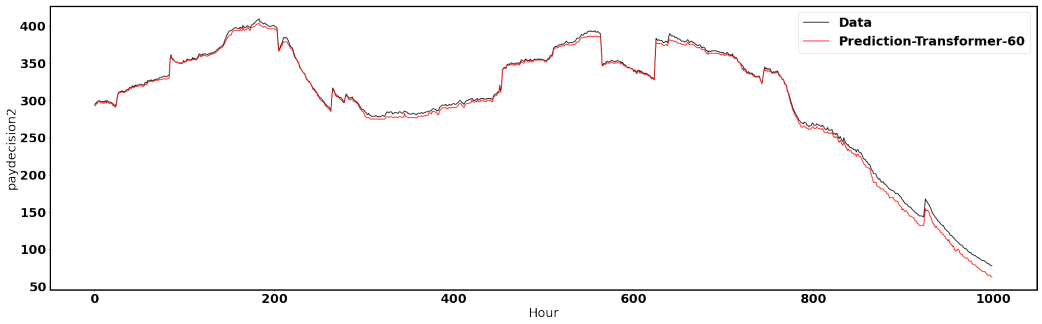
```
        y_pred_all_part = y_pred_all[:,i*bin_size:(i+1)*bin_size]
        y_pred_1d = y_pred_all_part[np.arange(0,y_pred_all.shape[0],bin_size),:]

        y_test_part = y_test[:,i*bin_size:(i+1)*bin_size]

        plt.plot(y_test_part[:1000,0],"k",label="Data")
        plt.plot(y_pred_1d.ravel()[:1000],"r",label="Prediction-Transformer-60")
        #diff = y_test[:1000,0]-y_pred_1d.ravel()[:1000]
        #plt.plot(diff,"b",label="Difference")


        plt.xlabel("Hour")
        plt.ylabel(r"%s"%names_array[i])
        plt.suptitle("Value vs Hour")
        plt.legend()

    fig = matplotlib.pyplot.gcf()


    fig.set_size_inches(35,12*temp.shape[1])
    plt.savefig(plot_path+"EU_Transformer_60_CIF_5.png")
```

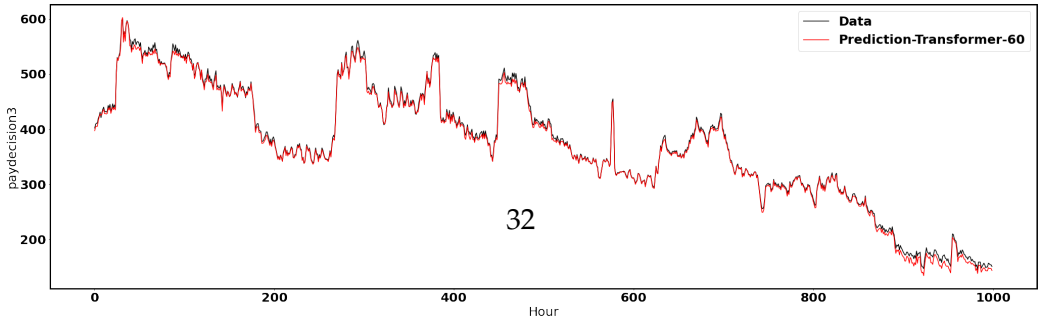
Value vs Hour











```
In [98]: print("Done")
```

```
Done
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [103]:
```

```
In [ ]:
```

```
In [136]:
```

```
In [145]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```