```
import pandas as pd
import numpy as np
import seaborn as sns
from datetime import datetime
from matplotlib import pyplot as plt

import tensorflow as tf
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import RandomForestRegressor
# from prophet import Prophet

%matplotlib inline
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
# model_dir = "./Models"
model_dir = '/content/drive/MyDrive/UNIVERSITAT/Inteligencia artificial/Treball/Models'
# dataset_dir = "./Dataset"
dataset_dir = '/content/drive/MyDrive/UNIVERSITAT/Inteligencia artificial/Treball/Dataset'
```

```
columns = [
    'year', 'month', 'day', 'hour', 'minute', 'second', 'glucose_level', 'finger_stick', 'basal', 'bolus', 'sleep',
    'work', 'stressors', 'hypo_event', 'illness', 'exercise', 'basis_heart_rate',
    'basis_gsr', 'basis_skin_temperature', 'basis_air_temperature',
    'basis_step', 'basis_sleep', 'meal', 'type_of_meal'
]
```

› UTILS

[ ] ↳ *4 cells hidden*

› ANALYSIS

[ ] ↳ *23 cells hidden*

› Transoformer encoder (1 Param)

[ ] ↳ *21 cells hidden*

⌄ Transformer encoder (N PARAMS)

› DATASET

[ ] ↳ *2 cells hidden*

⌄ DEFINE MODEL

```
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Normalization and Attention
    x = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = tf.keras.layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = tf.keras.layers.Dropout(dropout)(x)
    res = x + inputs

    # Feed Forward Part
    x = tf.keras.layers.LayerNormalization(epsilon=1e-6)(res)
    x = tf.keras.layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = tf.keras.layers.Dropout(dropout)(x)
    x = tf.keras.layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res
```

```
def build_model(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
    mlp_dropout=0,
):
    inputs = tf.keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = tf.keras.layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = tf.keras.layers.Dense(dim, activation="relu")(x)
        x = tf.keras.layers.Dropout(mlp_dropout)(x)
    outputs = tf.keras.layers.Dense(1)(x)
    return tf.keras.Model(inputs, outputs)
```

› BUILD MODEL

[ ] ↳ *1 cell hidden*

⌄ TRAIN

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
    # tf.keras.callbacks.ModelCheckpoint("Models/model_checkpoint.h5", save_best_only=True),
    # tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=5),
    # tf.keras.callbacks.TensorBoard(log_dir="logs"),
    # tf.keras.callbacks.CSVLogger("training.log"),
    # tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-3 * (0.9 ** epoch)),
    # tf.keras.callbacks.TerminateOnNaN(),
    # tf.keras.callbacks.RemoteMonitor(root='http://localhost:9000'),
    # tf.keras.callbacks.LambdaCallback(on_epoch_end=lambda epoch, logs: print(f"Epoch {epoch+1}, Loss: {logs['loss']}, Val_loss: {logs
]

model.fit(
    x_train,
    y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=64,
    callbacks=callbacks,
)

model.save(f"{model_dir}/model")
```

```
Epoch 1/50
151/151 [==============================] - 19s 24ms/step - loss: 15062.3262 - val_loss: 6720.8574
Epoch 2/50
151/151 [==============================] - 3s 20ms/step - loss: 6342.7305 - val_loss: 5828.9614
Epoch 3/50
151/151 [==============================] - 4s 23ms/step - loss: 5435.2017 - val_loss: 5236.8242
Epoch 4/50
151/151 [==============================] - 3s 20ms/step - loss: 4939.0425 - val_loss: 4921.4160
Epoch 5/50
151/151 [==============================] - 3s 20ms/step - loss: 4623.9062 - val_loss: 4659.7183
Epoch 6/50
151/151 [==============================] - 3s 20ms/step - loss: 4422.3521 - val_loss: 4348.4736
Epoch 7/50
151/151 [==============================] - 3s 22ms/step - loss: 4304.4912 - val_loss: 4204.7915
Epoch 8/50
151/151 [==============================] - 3s 22ms/step - loss: 4244.2676 - val_loss: 4044.8628
Epoch 9/50
151/151 [==============================] - 3s 20ms/step - loss: 4064.3223 - val_loss: 3917.1453
Epoch 10/50
151/151 [==============================] - 3s 19ms/step - loss: 4034.6536 - val_loss: 3806.8635
Epoch 11/50
151/151 [==============================] - 3s 22ms/step - loss: 3924.2988 - val_loss: 3692.0701
Epoch 12/50
151/151 [==============================] - 3s 23ms/step - loss: 3718.5168 - val_loss: 3588.0547
Epoch 13/50
151/151 [==============================] - 3s 20ms/step - loss: 3663.0510 - val_loss: 3438.0911
Epoch 14/50
151/151 [==============================] - 3s 19ms/step - loss: 3579.0317 - val_loss: 3339.7676
Epoch 15/50
151/151 [==============================] - 3s 21ms/step - loss: 3464.1572 - val_loss: 3203.8481
```

```
Epoch 16/50
151/151 [==============================] - 4s 24ms/step - loss: 3391.7612 - val_loss: 3075.8428
Epoch 17/50
151/151 [==============================] - 3s 19ms/step - loss: 3273.6882 - val_loss: 2987.7976
Epoch 18/50
151/151 [==============================] - 3s 19ms/step - loss: 3170.2944 - val_loss: 2912.6921
Epoch 19/50
151/151 [==============================] - 3s 19ms/step - loss: 3059.2397 - val_loss: 2787.0422
Epoch 20/50
151/151 [==============================] - 4s 25ms/step - loss: 2960.4429 - val_loss: 2714.5588
Epoch 21/50
151/151 [==============================] - 3s 20ms/step - loss: 2971.7781 - val_loss: 2640.8110
Epoch 22/50
151/151 [==============================] - 3s 20ms/step - loss: 2936.5708 - val_loss: 2603.4917
Epoch 23/50
151/151 [==============================] - 3s 20ms/step - loss: 2867.2085 - val_loss: 2579.0923
Epoch 24/50
151/151 [==============================] - 4s 26ms/step - loss: 2816.6814 - val_loss: 2486.2256
Epoch 25/50
151/151 [==============================] - 3s 19ms/step - loss: 2752.2720 - val_loss: 2425.2878
Epoch 26/50
151/151 [==============================] - 3s 19ms/step - loss: 2726.2458 - val_loss: 2385.5586
Epoch 27/50
151/151 [==============================] - 4s 24ms/step - loss: 2723.8228 - val_loss: 2344.2307
Epoch 28/50
151/151 [==============================] - 4s 25ms/step - loss: 2652.6396 - val_loss: 2291.9805
Epoch 29/50
151/151 [==============================] - 3s 19ms/step - loss: 2568.5210 - val_loss: 2281.3926
```

TEST

LOAD MODEL

```
loaded_model = tf.keras.saving.load_model(f"{model_dir}/model")
loaded_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to                  
==================================================================================================
 input_1 (InputLayer)        [(None, 24, 18)]             0         []                            
                                                                                                  
 layer_normalization (Layer  (None, 24, 18)               36        ['input_1[0][0]']             
 Normalization)                                                                                   
                                                                                                  
 multi_head_attention (Mult  (None, 24, 18)               76818     ['layer_normalization[0][0]', 
 iHeadAttention)                                                     'layer_normalization[0][0]'] 
                                                                                                  
 dropout (Dropout)           (None, 24, 18)               0         ['multi_head_attention[0][0]']
                                                                                                  
 tf.__operators__.add (TFOp  (None, 24, 18)               0         ['dropout[0][0]',             
 Lambda)                                                             'input_1[0][0]']             
                                                                                                  
 layer_normalization_1 (Lay  (None, 24, 18)               36        ['tf.__operators__.add[0][0]']
 erNormalization)                                                                                 
                                                                                                  
 conv1d (Conv1D)             (None, 24, 4)                76        ['layer_normalization_1[0][0]'
                                                                    ]                             
                                                                                                  
 dropout_1 (Dropout)         (None, 24, 4)                0         ['conv1d[0][0]']              
                                                                                                  
 conv1d_1 (Conv1D)           (None, 24, 18)               90        ['dropout_1[0][0]']           
                                                                                                  
 tf.__operators__.add_1 (TF  (None, 24, 18)               0         ['conv1d_1[0][0]',            
 OpLambda)                                                           'tf.__operators__.add[0][0]']
                                                                                                  
 layer_normalization_2 (Lay  (None, 24, 18)               36        ['tf.__operators__.add_1[0][0]
 erNormalization)                                                   ']                            
                                                                                                  
 multi_head_attention_1 (Mu  (None, 24, 18)               76818     ['layer_normalization_2[0][0]'
 ltiHeadAttention)                                                  , 'layer_normalization_2[0][0]
                                                                    ']                            
                                                                                                  
 dropout_2 (Dropout)         (None, 24, 18)               0         ['multi_head_attention_1[0][0]
                                                                    ']                            
                                                                                                  
 tf.__operators__.add_2 (TF  (None, 24, 18)               0         ['dropout_2[0][0]',           
 OpLambda)                                                           'tf.__operators__.add_1[0][0]
                                                                    ']                            
                                                                                                  
 layer_normalization_3 (Lay  (None, 24, 18)               36        ['tf.__operators__.add_2[0][0]
 erNormalization)                                                   ']                            
                                                                                                  
 conv1d_2 (Conv1D)           (None, 24, 4)                76        ['layer_normalization_3[0][0]'
                                                                    ]                             
 dropout_3 (Dropout)         (None, 24, 4)                0         ['conv1d_2[0][0]']

 conv1d_3 (Conv1D)           (None, 24, 18)               90        ['dropout_3[0][0]']

 tf.__operators__.add_3 (TF  (None, 24, 18)               0         ['conv1d_3[0][0]',
 OpLambda)                                                           'tf.__operators__.add_2[0][0]
                                                                    ']
```

LOAD TEST DATA

```
number = 559

train = pd.read_csv(f'{dataset_dir}/{number}/{number}_train.csv', sep=';',encoding = 'unicode_escape', names=columns)
test = pd.read_csv(f'{dataset_dir}/{number}/{number}_test.csv', sep=';',encoding = 'unicode_escape', names=columns)

train = preprocess_data(train)
test = preprocess_data(test)
```

```
SEQUENCE_SIZE = 24
PREDICTION_TIME = 6

# Order by correlation and variance
sequence_columns = [
    'glucose_level',
    'basis_heart_rate',
    'basis_sleep',
    'basis_step',
    'basal',
    'finger_stick',
    'basis_skin_temperature',
    'basis_air_temperature',
    'work',
    'exercise',
    'bolus',
    'meal',
    'stressors',
    'illness',
    'type_of_meal',
    'hypo_event',
    'basis_gsr',
    'sleep'
]

target_columns = ['glucose_level']

x_test, y_test = to_sequences_multi(test, SEQUENCE_SIZE, PREDICTION_TIME, sequence_columns, target_columns)
print("Shape of train set: {}".format(x_train.shape))
print("Shape of test set: {}".format(x_test.shape))
print("Shape of test set: {}".format(y_test.shape))
```

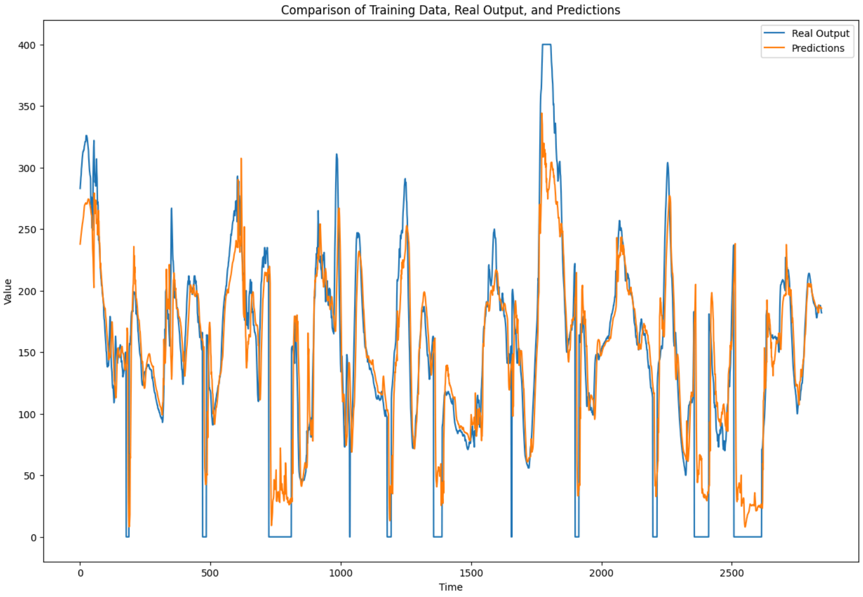
```
Shape of train set: (12052, 24, 18)
Shape of test set: (2845, 24, 18)
Shape of test set: (2845, 1)
```

TEST

```
pred = loaded_model.predict(x_test)
```

```
89/89 [==============================] - 1s 8ms/step
```

```
# Set the size of the figure
plt.figure(figsize=(15, 10))  # Adjust width and height as needed

# Plotting the real output (y_test)
plt.plot(np.arange(len(y_test)), y_test, label='Real Output')

# Plotting the predictions (pred)
plt.plot(np.arange(len(pred)), pred, label='Predictions')
# plt.plot(np.arange(len(pred)), y_test - pred, label='Difference')

# Adding labels and legend
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Comparison of Training Data, Real Output, and Predictions')
plt.legend()

# Show plot
plt.show()

# Calculating and printing the RMSE score
score = np.sqrt(metrics.mean_squared_error(pred, y_test))
print("Score (RMSE): {}".format(score))
```



```
Score (RMSE): 41.32217753057616
```

RNN

LSTM

DATASET

```
number = 559
train = pd.read_csv(f'{dataset_dir}/{number}/{number}_train.csv', sep=';',encoding = 'unicode_escape', names=columns)
train = preprocess_data(train, time_index=False)
```

```
SEQUENCE_SIZE = 24
PREDICTION_TIME = 6
# Order by correlation and variance
sequence_columns = [
    'glucose_level',
    'basis_heart_rate',
    'basis_sleep',
    'basis_step',
    'basal',
    'finger_stick',
    'basis_skin_temperature',
    'basis_air_temperature',
    'work',
    'exercise',
    'bolus',
    'meal',
    'stressors',
    'illness',
    'type_of_meal',
    'hypo_event',
    'basis_gsr',
    'sleep'
]

target_columns = ['glucose_level']

x_train, y_train = to_sequences_multi(train, SEQUENCE_SIZE, PREDICTION_TIME, sequence_columns, target_columns)
x_train_time, _ = to_sequences_multi(train, SEQUENCE_SIZE, PREDICTION_TIME, ['timestamp'], target_columns)
print("Shape of x training set: {}".format(x_train.shape))
print("Shape of y training set: {}".format(y_train.shape))
print("Shape of x time training set: {}".format(x_train_time.shape))
```

```
Shape of x training set: (12052, 24, 18)
Shape of y training set: (12052, 1)
Shape of x time training set: (12052, 24, 1)
```

DEFINE MODEL

```
def build_model(input_shape):
    input_features = tf.keras.layers.Input(shape=input_shape)

    # LSTM layers for input features
    lstm_output = tf.keras.layers.LSTM(512, return_sequences=True)(input_features)
    lstm_output = tf.keras.layers.Dropout(0.3)(lstm_output)

    # Additional LSTM layer
    lstm_output = tf.keras.layers.LSTM(512, return_sequences=True)(lstm_output)
    lstm_output = tf.keras.layers.Dropout(0.3)(lstm_output)

    # Additional LSTM layer
    lstm_output = tf.keras.layers.LSTM(512, return_sequences=True)(lstm_output)
    lstm_output = tf.keras.layers.Dropout(0.3)(lstm_output)

    # Additional LSTM layer
    lstm_output = tf.keras.layers.LSTM(512, return_sequences=False)(lstm_output)
    lstm_output = tf.keras.layers.Dropout(0.3)(lstm_output)

    # Dense layers
    dense_output = tf.keras.layers.Dense(128, activation='relu')(lstm_output)
    output = tf.keras.layers.Dense(1, activation='linear')(dense_output)

    model = tf.keras.models.Model(inputs=input_features, outputs=output)
    return model
```

BUILD MODEL

```
input_shape = x_train.shape[1:]

# Build the model
model = build_model(input_shape)

# Compile the model
model.compile(
    loss="mean_squared_error",
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
)

model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 24, 18)]          0

 lstm_4 (LSTM)               (None, 24, 512)           1087488

 dropout_17 (Dropout)        (None, 24, 512)           0

 lstm_5 (LSTM)               (None, 24, 512)           2099200

 dropout_18 (Dropout)        (None, 24, 512)           0

 lstm_6 (LSTM)               (None, 24, 512)           2099200

 dropout_19 (Dropout)        (None, 24, 512)           0

 lstm_7 (LSTM)               (None, 512)               2099200

 dropout_20 (Dropout)        (None, 512)               0

 dense_4 (Dense)             (None, 128)               65664

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 7450881 (28.42 MB)
Trainable params: 7450881 (28.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

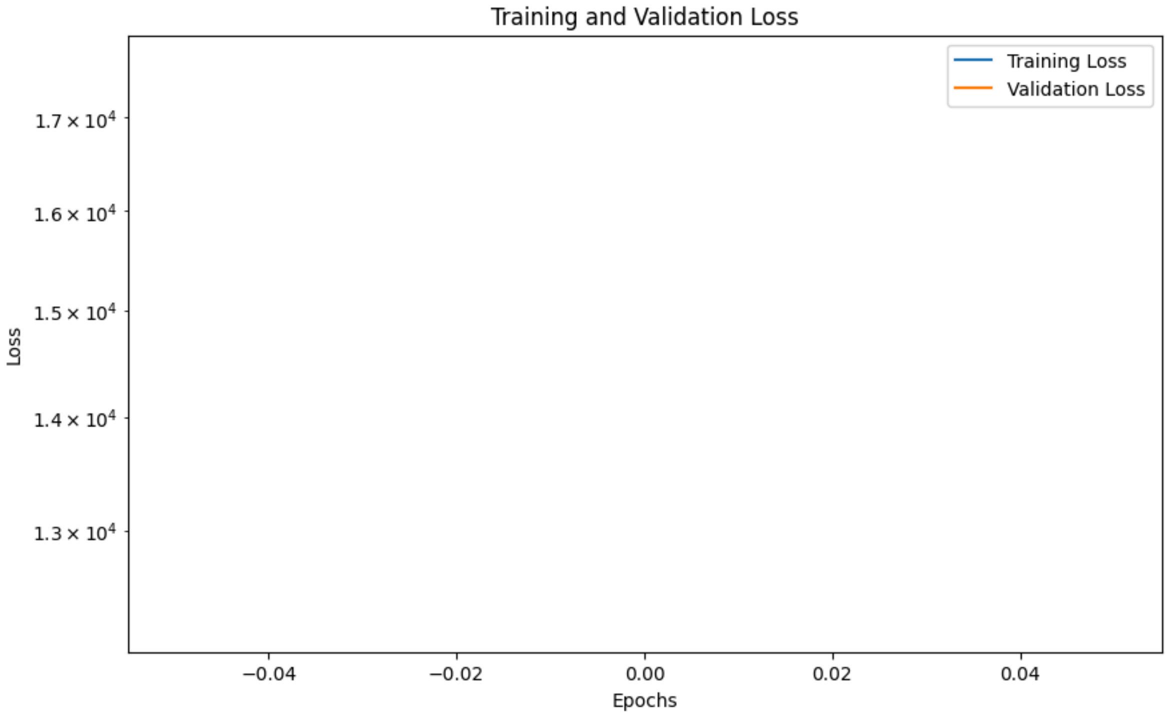
› CALLBACKS

[ ] ↳ *1 cell hidden*

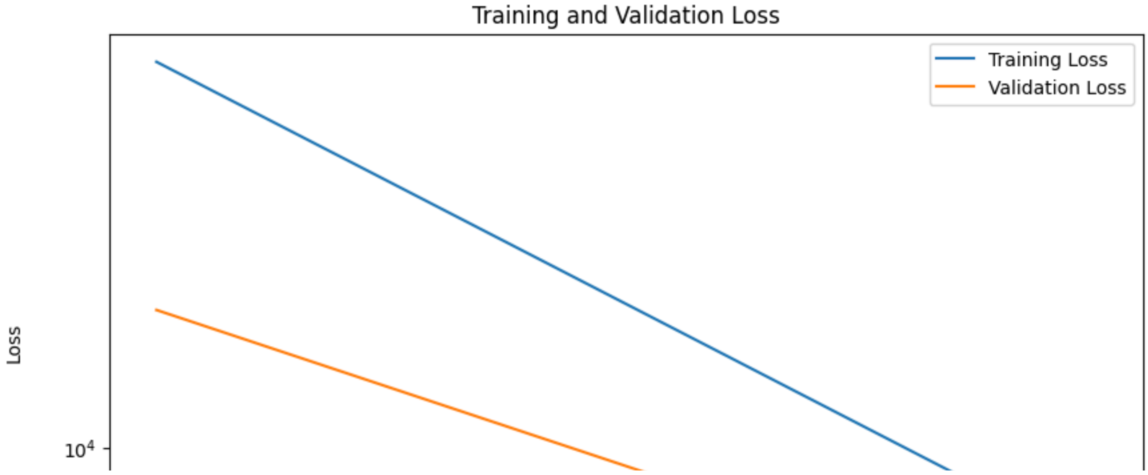
⌄ TRAIN

```
plot_loss = PlotLossCallback()

callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
    plot_loss
    # tf.keras.callbacks.ModelCheckpoint("Models/model_checkpoint.h5", save_best_only=True),
    # tf.keras.callbacks.ReduceLROnPlateau(factor=0.2, patience=5),
    # tf.keras.callbacks.TensorBoard(log_dir="logs"),
    # tf.keras.callbacks.CSVLogger("training.log"),
    # tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-3 * (0.9 ** epoch)),
    # tf.keras.callbacks.TerminateOnNaN(),
    # tf.keras.callbacks.RemoteMonitor(root='http://localhost:9000'),
    # tf.keras.callbacks.LambdaCallback(on_epoch_end=lambda epoch, logs: print(f"Epoch {epoch+1}, Loss: {logs['loss']}, Val_loss: {logs
]

model.fit(
    x_train,
    y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=64,
    callbacks=callbacks,
)

model.save(f"{model_dir}/model")
```

```
Epoch 1/50
151/151 [==============================] - ETA: 0s - loss: 17609.4648
```



```
151/151 [==============================] - 13s 44ms/step - loss: 17609.4648 - val_loss: 12238.7529
Epoch 2/50
151/151 [==============================] - ETA: 0s - loss: 8653.8105
```



TEST

LOAD MODEL

```
loaded_model = tf.keras.saving.load_model(f"{model_dir}/model")
loaded_model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 24, 18)]          0         
                                                                 
 lstm_4 (LSTM)               (None, 24, 512)           1087488   
                                                                 
 dropout_17 (Dropout)        (None, 24, 512)           0         
                                                                 
 lstm_5 (LSTM)               (None, 24, 512)           2099200   
                                                                 
 dropout_18 (Dropout)        (None, 24, 512)           0         
                                                                 
 lstm_6 (LSTM)               (None, 24, 512)           2099200   
                                                                 
 dropout_19 (Dropout)        (None, 24, 512)           0         
                                                                 
 lstm_7 (LSTM)               (None, 512)               2099200   
                                                                 
 dropout_20 (Dropout)        (None, 512)               0         
 dense_4 (Dense)             (None, 128)               65664

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 7450881 (28.42 MB)
Trainable params: 7450881 (28.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

LOAD TEST DATA

```
151/151 [==============================] - ETA: 0s - loss: 7098.4229
```

```
number = 559

train = pd.read_csv(f'{dataset_dir}/{number}/{number}_train.csv', sep=';',encoding = 'unicode_escape', names=columns)
test = pd.read_csv(f'{dataset_dir}/{number}/{number}_test.csv', sep=';',encoding = 'unicode_escape', names=columns)

train = preprocess_data(train, time_index=False)
test = preprocess_data(test, time_index=False)
```

```
SEQUENCE_SIZE = 24
PREDICTION_TIME = 6

# Order by correlation and variance
sequence_columns = [
    'glucose_level',
    'basis_heart_rate',
    'basis_sleep',
    'basis_step',
    'basal',
    'finger_stick',
    'basis_skin_temperature',
    'basis_air_temperature',
    'work',
    'exercise',
    'bolus',
    'meal',
    'stressors',
    'illness',
    'type_of_meal',
    'hypo_event',
    'basis_gsr',
    'sleep'
]

target_columns = ['glucose_level']

x_test, y_test = to_sequences_multi(test, SEQUENCE_SIZE, PREDICTION_TIME, sequence_columns, target_columns)
x_test_time, _ = to_sequences_multi(test, SEQUENCE_SIZE, PREDICTION_TIME, ['timestamp'], target_columns)
print("Shape of x test set: {}".format(x_test.shape))
print("Shape of y test set: {}".format(y_test.shape))
print("Shape of x time test set: {}".format(x_test_time.shape))
```

```
Shape of x test set: (2845, 24, 18)
Shape of y test set: (2845, 1)
Shape of x time test set: (2845, 24, 1)
```

TEST

```
pred = loaded_model.predict(x_test)
pred.shape
```

```
89/89 [==============================] - 2s 8ms/step
(2845, 1)
```

```
# Set the size of the figure
plt.figure(figsize=(15, 10))  # Adjust width and height as needed

# Plotting the real output (y_test)
plt.plot(np.arange(len(y_test)), y_test, label='Real Output')

# Plotting the predictions (pred)
plt.plot(np.arange(len(pred)), pred, label='Predictions')

# Adding labels and legend
plt.xlabel('Time')
```