# MANAI MOHAMED MORTADHA - 3GII/SSE

```
# Import necessary libraries
import pandas as pd  # For data manipulation and analysis
import matplotlib.pyplot as plt  # For plotting graphs
import numpy as np  # For numerical operations
import tensorflow as tf  # For machine learning

# Import specific modules from TensorFlow
from tensorflow.keras.models import Sequential  # Sequential model for stacking layers
from tensorflow.keras.layers import Dropout, Dense, LSTM  # Different types of neural network layers
```

```
# Read the Excel file into a Pandas DataFrame
df = pd.read_excel("/content/consumption.xlsx")
```

```
df.head() #"df.head()" is a method used in Pandas to display the first few rows of a DataFrame named "df".
```

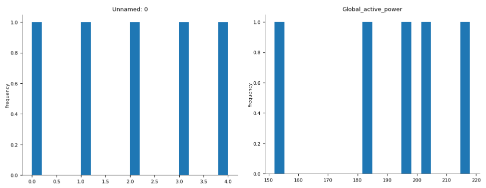
1 to 5 of 5 entries Filter

| index | Unnamed: 0 | datetime | Global_active_power |
|---|---|---|---|
| 0 | 0 | 2006-12-16 17:00:00 | 152.024 |
| 1 | 1 | 2006-12-16 18:00:00 | 217.932 |
| 2 | 2 | 2006-12-16 19:00:00 | 204.014 |
| 3 | 3 | 2006-12-16 20:00:00 | 196.114 |
| 4 | 4 | 2006-12-16 21:00:00 | 183.388 |

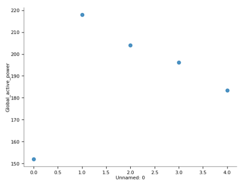
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

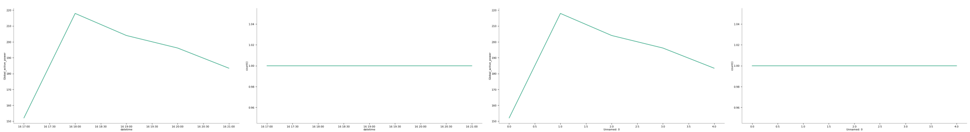
**Distributions**



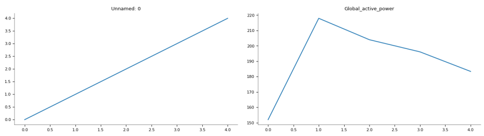

**2-d distributions**



**Time series**



**Values**




```
df.tail() #"df.tail()" is used in Pandas to display the last few rows of a DataFrame named "df".
```

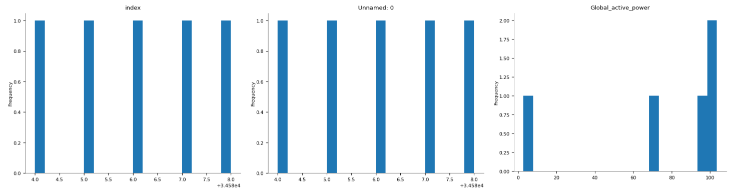
1 to 5 of 5 entries Filter

| index | Unnamed: 0 | datetime | Global_active_power |
|---|---|---|---|
| **34584** | 34584 | 2010-11-26 17:00:00 | 103.554 |
| **34585** | 34585 | 2010-11-26 18:00:00 | 94.408 |
| **34586** | 34586 | 2010-11-26 19:00:00 | 99.56 |
| **34587** | 34587 | 2010-11-26 20:00:00 | 69.822 |
| **34588** | 34588 | 2010-11-26 21:00:00 | 2.804 |

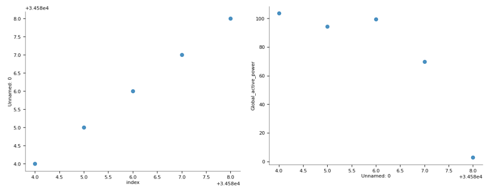
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

**Distributions**



**2-d distributions**



**Time series**

```
# Convert the 'datetime' column to a datetime data type and 'Global_active_power' column to numeric,
# handling any errors by converting them to 'NaN' (Not a Number)
df['datetime'] = pd.to_datetime(df['datetime'])
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')
```

```
# Display the data types of each column in the DataFrame
print(df.dtypes)
```

```
Unnamed: 0                      int64
datetime               datetime64[ns]
Global_active_power           float64
dtype: object
```

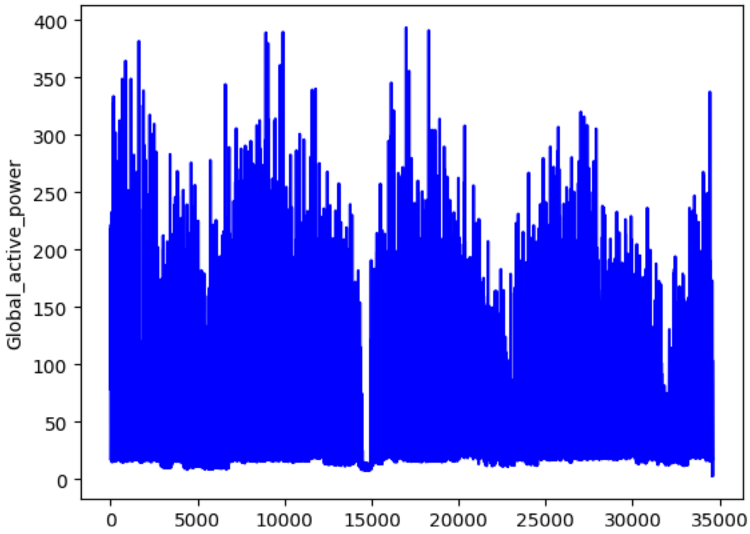
```
# Display the shape of the DataFrame, indicating the number of rows and columns
print(df.shape)
```

```
(34589, 3)
```

```
# Generate a concise summary of the DataFrame's information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34589 entries, 0 to 34588
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Unnamed: 0           34589 non-null  int64
 1   datetime             34589 non-null  datetime64[ns]
 2   Global_active_power  34589 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 810.8 KB
```

```
# Plotting the 'Global_active_power' against 'datetime'
plt.xlabel("datetime")  # Label for the x-axis indicating datetime
plt.ylabel("Global_active_power")  # Label for the y-axis indicating Global_active_power
plt.plot(df['Global_active_power'], color='blue')  # Plotting Global_active_power values in blue
plt.show()  # Displaying the plot
```




```
# Calculating the training size for the dataset, which is 80% of the 'Global_active_power' column's length
training_size = int(len(df['Global_active_power']) * 0.8)
training_size  # Displaying the calculated training size
```

```
27671
```

```
# Function to load data for a sequence model
def load_data(data, seq_len):
    x = []  # List to store input sequences
    y = []  # List to store output sequences
    for i in range(seq_len, len(data)):
        # Append sequences of length 'seq_len' to 'x' and the corresponding next value to 'y'
        x.append(data.iloc[i - seq_len: i, 1])  # Input sequence
        y.append(data.iloc[i, 1])  # Corresponding output value

    return x, y  # Return the input sequences and output values
```

```
# Generating input sequences ('x') and corresponding output values ('y') using the 'load_data' function
x, y = load_data(df, 20)
```

```
# Determining the number of input sequences generated ('x')
len(x)
```

```
34569
```

```
# Splitting the generated sequences and corresponding output values into training and test sets
x_train = x[:training_size]  # Training input sequences
y_train = y[:training_size]  # Corresponding output values for training
x_test = x[training_size:]   # Test input sequences
y_test = y[training_size:]   # Corresponding output values for testing
```

```
# Converting the training and test sets from lists to NumPy arrays
x_train = np.array(x_train)  # Training input sequences as a NumPy array
y_train = np.array(y_train)  # Corresponding output values for training as a NumPy array
x_test = np.array(x_test)    # Test input sequences as a NumPy array
y_test = np.array(y_test)    # Corresponding output values for testing as a NumPy array
```

```
# Displaying the shapes of the training and test sets
print('x_train.shape = ', x_train.shape)  # Shape of the training input sequences
print('y_train.shape = ', y_train.shape)  # Shape of the corresponding output values for training
print('x_test.shape = ', x_test.shape)    # Shape of the test input sequences
print('y_test.shape = ', y_test.shape)    # Shape of the corresponding output values for testing
```

```
x_train.shape =  (27671, 20)
y_train.shape =  (27671,)
x_test.shape =  (6898, 20)
y_test.shape =  (6898,)
```

```
# Reshaping the input sequences for compatibility with LSTM model
x_train = np.reshape(x_train, (training_size, 20, 1))  # Reshaping training input sequences to (training_size, 20, 1)
x_test = np.reshape(x_test, (x_test.shape[0], 20, 1))  # Reshaping test input sequences to match the LSTM input shape



# Displaying the shapes of the reshaped training and test sets
print('x_train.shape = ', x_train.shape)  # Shape of the reshaped training input sequences
print('y_train.shape = ', y_train.shape)  # Shape of the corresponding output values for training
print('x_test.shape = ', x_test.shape)    # Shape of the reshaped test input sequences
print('y_test.shape = ', y_test.shape)    # Shape of the corresponding output values for testing
```

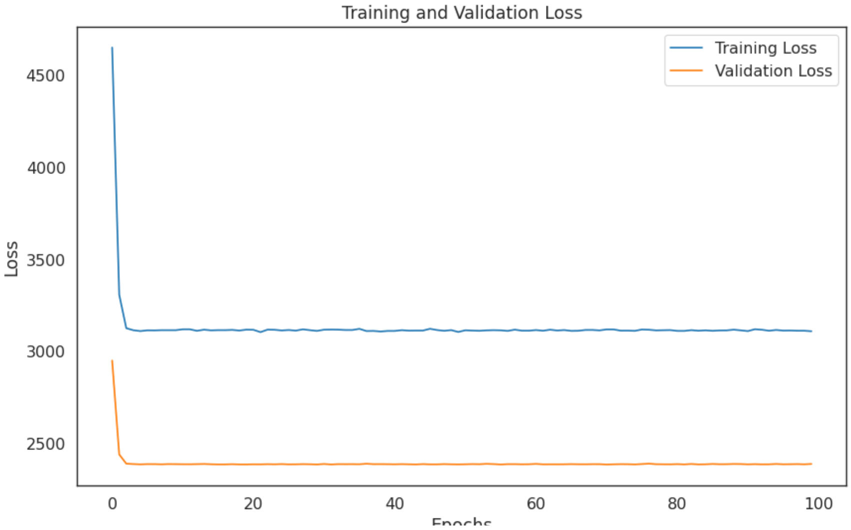
```
x_train.shape =  (27671, 20, 1)
y_train.shape =  (27671,)
x_test.shape =  (6898, 20, 1)
y_test.shape =  (6898,)
```

```
# Prepare input and output sequences for LSTM
sequence_length = 10  # Length of the sequence to consider
data = df['Global_active_power'].values
timestamps = df['datetime'].values.astype(np.int64) // 10**9  # Convert to UNIX timestamp
X, y = [], []
for i in range(len(data) - sequence_length):
    X.append(timestamps[i:i+sequence_length])  # Use datetime as the sequence
    y.append(data[i+sequence_length])  # Target value after the sequence

X = np.array(X)
y = np.array(y)

# Reshape input for LSTM (samples, time steps, features)
X = np.reshape(X, (X.shape[0], sequence_length, 1))

# Splitting the data into training and test sets
train_size = int(len(X) * 0.7)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size], X[train_size:len(X)]
y_train, y_test = y[0:train_size], y[train_size:len(y)]

# Building the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the LSTM model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Plotting the training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

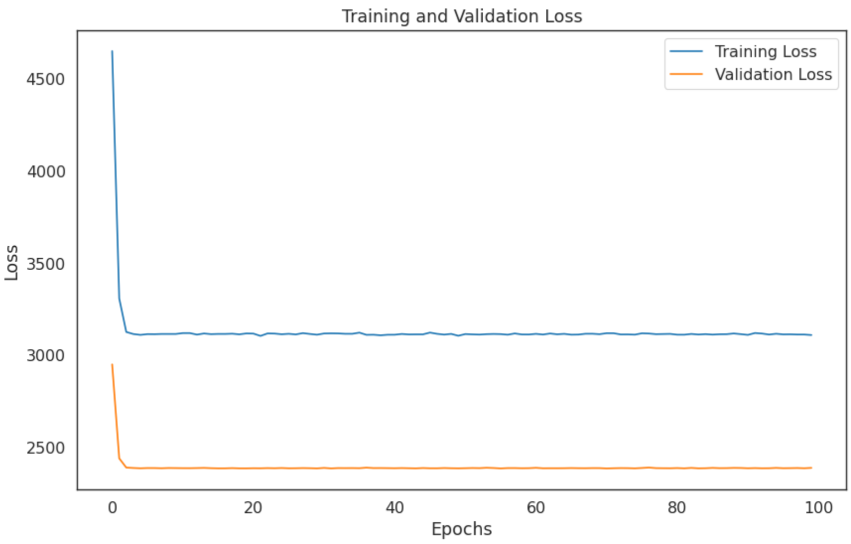
```
Epoch 3/100
757/757 [==============================] - 12s 15ms/step - loss: 3125.2217 -
Epoch 4/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.0281 -
Epoch 5/100
757/757 [==============================] - 12s 15ms/step - loss: 3109.4314 -
Epoch 6/100
757/757 [==============================] - 12s 16ms/step - loss: 3113.5483 -
Epoch 7/100
757/757 [==============================] - 11s 14ms/step - loss: 3113.1829 -
Epoch 8/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.3462 -
Epoch 9/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.3560 -
Epoch 10/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.3132 -
Epoch 11/100
757/757 [==============================] - 11s 15ms/step - loss: 3119.0837 -
Epoch 12/100
757/757 [==============================] - 11s 15ms/step - loss: 3119.1660 -
Epoch 13/100
757/757 [==============================] - 12s 15ms/step - loss: 3110.8628 -
Epoch 14/100
757/757 [==============================] - 11s 14ms/step - loss: 3117.0364 -
Epoch 15/100
757/757 [==============================] - 12s 15ms/step - loss: 3113.4033 -
Epoch 16/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.5281 -
Epoch 17/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.5181 -
Epoch 18/100
757/757 [==============================] - 11s 15ms/step - loss: 3115.8413 -
Epoch 19/100
757/757 [==============================] - 12s 16ms/step - loss: 3112.3562 -
Epoch 20/100
757/757 [==============================] - 11s 14ms/step - loss: 3117.2524 -
Epoch 21/100
757/757 [==============================] - 11s 14ms/step - loss: 3116.4170 -
Epoch 22/100
757/757 [==============================] - 12s 15ms/step - loss: 3103.7366 -
Epoch 23/100
757/757 [==============================] - 12s 15ms/step - loss: 3117.3132 -
Epoch 24/100
757/757 [==============================] - 11s 15ms/step - loss: 3116.4949 -
Epoch 25/100
757/757 [==============================] - 11s 15ms/step - loss: 3112.8364 -
Epoch 26/100
757/757 [==============================] - 11s 14ms/step - loss: 3115.2529 -
Epoch 27/100
757/757 [==============================] - 11s 14ms/step - loss: 3111.9551 -
Epoch 28/100
757/757 [==============================] - 11s 15ms/step - loss: 3118.9329 -
Epoch 29/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.3821 -
Epoch 30/100
757/757 [==============================] - 12s 15ms/step - loss: 3110.3892 -
Epoch 31/100
757/757 [==============================] - 12s 15ms/step - loss: 3116.9946 -
Epoch 32/100
757/757 [==============================] - 11s 14ms/step - loss: 3117.7412 -
Epoch 33/100
757/757 [==============================] - 11s 15ms/step - loss: 3117.2356 -
Epoch 34/100
757/757 [==============================] - 12s 15ms/step - loss: 3115.3191 -
Epoch 35/100
757/757 [==============================] - 12s 16ms/step - loss: 3115.3359 -
Epoch 36/100
757/757 [==============================] - 12s 16ms/step - loss: 3121.6221 -
Epoch 37/100
757/757 [==============================] - 12s 15ms/step - loss: 3109.6636 -
Epoch 38/100
757/757 [==============================] - 12s 15ms/step - loss: 3110.2100 -
Epoch 39/100
757/757 [==============================] - 11s 14ms/step - loss: 3107.2197 -
Epoch 40/100
757/757 [==============================] - 11s 15ms/step - loss: 3110.1248 -
Epoch 41/100
757/757 [==============================] - 13s 17ms/step - loss: 3110.2742 -
Epoch 42/100
757/757 [==============================] - 12s 16ms/step - loss: 3114.3384 -
Epoch 43/100
757/757 [==============================] - 14s 18ms/step - loss: 3111.8499 -
Epoch 44/100
757/757 [==============================] - 12s 15ms/step - loss: 3112.2090 -
Epoch 45/100
757/757 [==============================] - 13s 17ms/step - loss: 3112.2339 -
Epoch 46/100
757/757 [==============================] - 12s 16ms/step - loss: 3121.8862 -
Epoch 47/100
757/757 [==============================] - 12s 15ms/step - loss: 3115.0747 -
Epoch 48/100
```

```
757/757 [==============================] - 11s 14ms/step - loss: 3110.9304 -
Epoch 49/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.7515 -
Epoch 50/100
757/757 [==============================] - 12s 15ms/step - loss: 3104.6216 -
Epoch 51/100
757/757 [==============================] - 11s 15ms/step - loss: 3113.8538 -
Epoch 52/100
757/757 [==============================] - 13s 17ms/step - loss: 3112.1936 -
Epoch 53/100
757/757 [==============================] - 13s 18ms/step - loss: 3110.9788 -
Epoch 54/100
757/757 [==============================] - 12s 15ms/step - loss: 3113.3755 -
Epoch 55/100
757/757 [==============================] - 11s 14ms/step - loss: 3114.3760 -
Epoch 56/100
757/757 [==============================] - 11s 14ms/step - loss: 3113.5776 -
Epoch 57/100
757/757 [==============================] - 11s 15ms/step - loss: 3110.3975 -
Epoch 58/100
757/757 [==============================] - 11s 15ms/step - loss: 3117.0984 -
Epoch 59/100
757/757 [==============================] - 11s 15ms/step - loss: 3111.7710 -
Epoch 60/100
757/757 [==============================] - 11s 15ms/step - loss: 3111.5488 -
Epoch 61/100
757/757 [==============================] - 11s 14ms/step - loss: 3114.9709 -
Epoch 62/100
757/757 [==============================] - 11s 15ms/step - loss: 3111.3291 -
Epoch 63/100
757/757 [==============================] - 11s 15ms/step - loss: 3116.9019 -
Epoch 64/100
757/757 [==============================] - 12s 15ms/step - loss: 3112.6287 -
Epoch 65/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.9800 -
Epoch 66/100
757/757 [==============================] - 13s 17ms/step - loss: 3110.5308 -
Epoch 67/100
757/757 [==============================] - 12s 15ms/step - loss: 3111.1384 -
Epoch 68/100
757/757 [==============================] - 12s 15ms/step - loss: 3115.4910 -
Epoch 69/100
757/757 [==============================] - 11s 14ms/step - loss: 3115.6174 -
Epoch 70/100
757/757 [==============================] - 12s 15ms/step - loss: 3113.1858 -
Epoch 71/100
757/757 [==============================] - 12s 15ms/step - loss: 3118.5520 -
Epoch 72/100
757/757 [==============================] - 12s 15ms/step - loss: 3118.3354 -
Epoch 73/100
757/757 [==============================] - 12s 15ms/step - loss: 3111.7527 -
Epoch 74/100
757/757 [==============================] - 11s 15ms/step - loss: 3111.9653 -
Epoch 75/100
757/757 [==============================] - 10s 14ms/step - loss: 3110.7039 -
Epoch 76/100
757/757 [==============================] - 11s 15ms/step - loss: 3117.9739 -
Epoch 77/100
757/757 [==============================] - 12s 15ms/step - loss: 3116.7124 -
Epoch 78/100
757/757 [==============================] - 12s 15ms/step - loss: 3113.2629 -
Epoch 79/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.0879 -
Epoch 80/100
757/757 [==============================] - 11s 15ms/step - loss: 3114.8333 -
Epoch 81/100
757/757 [==============================] - 11s 14ms/step - loss: 3110.5154 -
Epoch 82/100
757/757 [==============================] - 11s 15ms/step - loss: 3110.4258 -
Epoch 83/100
757/757 [==============================] - 12s 15ms/step - loss: 3114.4690 -
Epoch 84/100
757/757 [==============================] - 12s 15ms/step - loss: 3111.3457 -
Epoch 85/100
757/757 [==============================] - 12s 15ms/step - loss: 3113.4619 -
Epoch 86/100
757/757 [==============================] - 11s 15ms/step - loss: 3110.9160 -
Epoch 87/100
757/757 [==============================] - 14s 18ms/step - loss: 3112.8567 -
Epoch 88/100
757/757 [==============================] - 16s 21ms/step - loss: 3113.2490 -
Epoch 89/100
757/757 [==============================] - 11s 15ms/step - loss: 3117.0815 -
Epoch 90/100
757/757 [==============================] - 13s 17ms/step - loss: 3113.3994 -
Epoch 91/100
757/757 [==============================] - 12s 16ms/step - loss: 3109.3975 -
Epoch 92/100
757/757 [==============================] - 11s 15ms/step - loss: 3119.5122 -
Epoch 93/100
757/757 [==============================] - 13s 17ms/step - loss: 3116.6841 -
```

```
Epoch 94/100
757/757 [==============================] - 12s 16ms/step - loss: 3111.2842 -
Epoch 95/100
757/757 [==============================] - 12s 16ms/step - loss: 3115.4736 -
Epoch 96/100
757/757 [==============================] - 12s 16ms/step - loss: 3112.0842 -
Epoch 97/100
757/757 [==============================] - 14s 18ms/step - loss: 3112.5420 -
Epoch 98/100
757/757 [==============================] - 14s 18ms/step - loss: 3111.3218 -
Epoch 99/100
757/757 [==============================] - 13s 17ms/step - loss: 3111.3115 -
Epoch 100/100
757/757 [==============================] - 13s 17ms/step - loss: 3108.5542 -
```



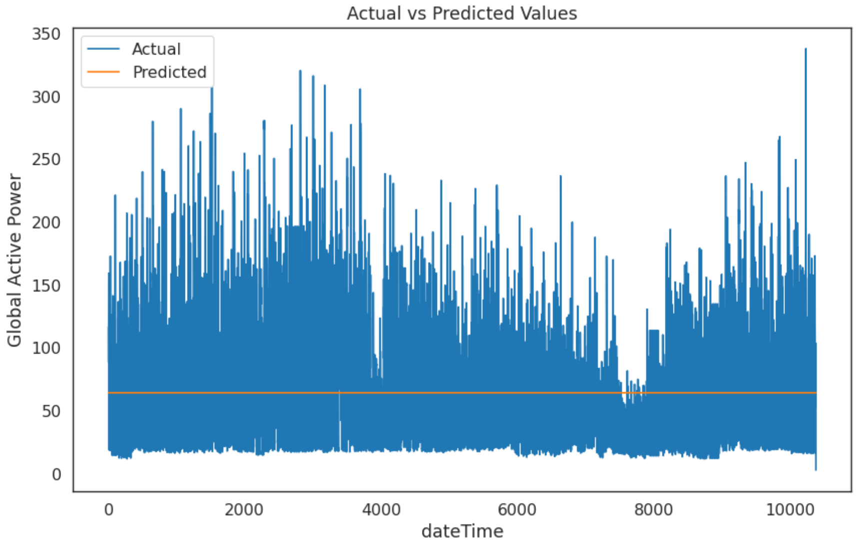

```
# Plotting the training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Generating predictions using the trained model on the test data (X_test)
predictions = model.predict(X_test)
```

```
325/325 [==============================] - 2s 5ms/step
```

```
# Visualizing the comparison between actual and predicted values
plt.figure(figsize=(10, 6))  # Set the figure size for the plot
plt.plot(y_test, label='Actual')  # Plotting actual values from the test set
plt.plot(predictions, label='Predicted')  # Plotting predicted values generated by the model
plt.title('Actual vs Predicted Values')  # Setting the title of the plot
plt.xlabel('dateTime')  # Label for the x-axis indicating time
plt.ylabel('Global Active Power')  # Label for the y-axis indicating Global Active Power
plt.legend()  # Displaying legend to differentiate between actual and predicted values
plt.show()  # Displaying the plot
```




```
predictions
```

```
array([[64.2538],
       [64.2538],
       [64.2538],
       ...,
       [64.2538],
       [64.2538],
       [64.2538]], dtype=float32)
```

```
y_test
```

```
array([116.328, 107.066,  88.468, ...,  99.56 ,  69.822,   2.804])
```

```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, predictions)).round(2)
mape = np.round(np.mean(np.abs(y_test-predictions)/y_test)*100,2)
```

```
rmse
```

```
48.85
```

```
mape
```

```
101.69
```