Electricity Demand and Price Forecasting

November 26, 2024

[1]:

```
#import modules and packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from xgboost import XGBRegressor, plot_importance
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from keras import optimizers
from keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, MaxPooling1D
from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed, Flatten
from sklearn.metrics import r2_score
```

[2]:

```
# Load Dataseti
from google.colab import drive
drive.mount('/content/drive')
!ls /content/drive/MyDrive/
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/powerconsumption.csv')
```

```
Mounted at /content/drive
'Colab Notebooks'
'Csc '
'ELECTRICITY DEMAND AND PRICE FORECASTING (1).gsheet'
'ELECTRICITY DEMAND AND PRICE FORECASTING.gsheet'
 IMG-20211220-WA0006.jpg
 IMG-20211220-WA0016.jpg
 IMG-20211220-WA0017.jpg
 IMG-20211220-WA0021.jpg
'ManthenaAbhinav-FullStackWebDeveloper-ygBg (1).pdf'
'new_resume_Rushikesh-2 (1).pdf'
 new_resume_Rushikesh-2.pdf
 powerconsumption.csv
'Rushikesh Doosa_21831A6215 (1).pdf'
'Rushikesh Doosa_21831A6215 (2).pdf'
```

```
'Rushikesh Doosa_21831A6215 (3).pdf'
'Rushikesh Doosa_21831A6215.pdf'
 Rushikesh_Doosa_ElectricityDemandandPriceForecasting
'Rushikesh Doosa Major Project.pdf'
'Rushikesh Doosa Minor Project.pdf'
 Screenshot_20240130_120901_Chrome.jpg
'Screenshot_20240214_150827_Samsung Internet.jpg'
 Screenshot_20240314_101422_PhonePe.jpg
 Screenshot_2024-08-01-14-59-34-58_40deb401b9ffe8e1df2f1cc5ba480b12.jpg
```

[3]:

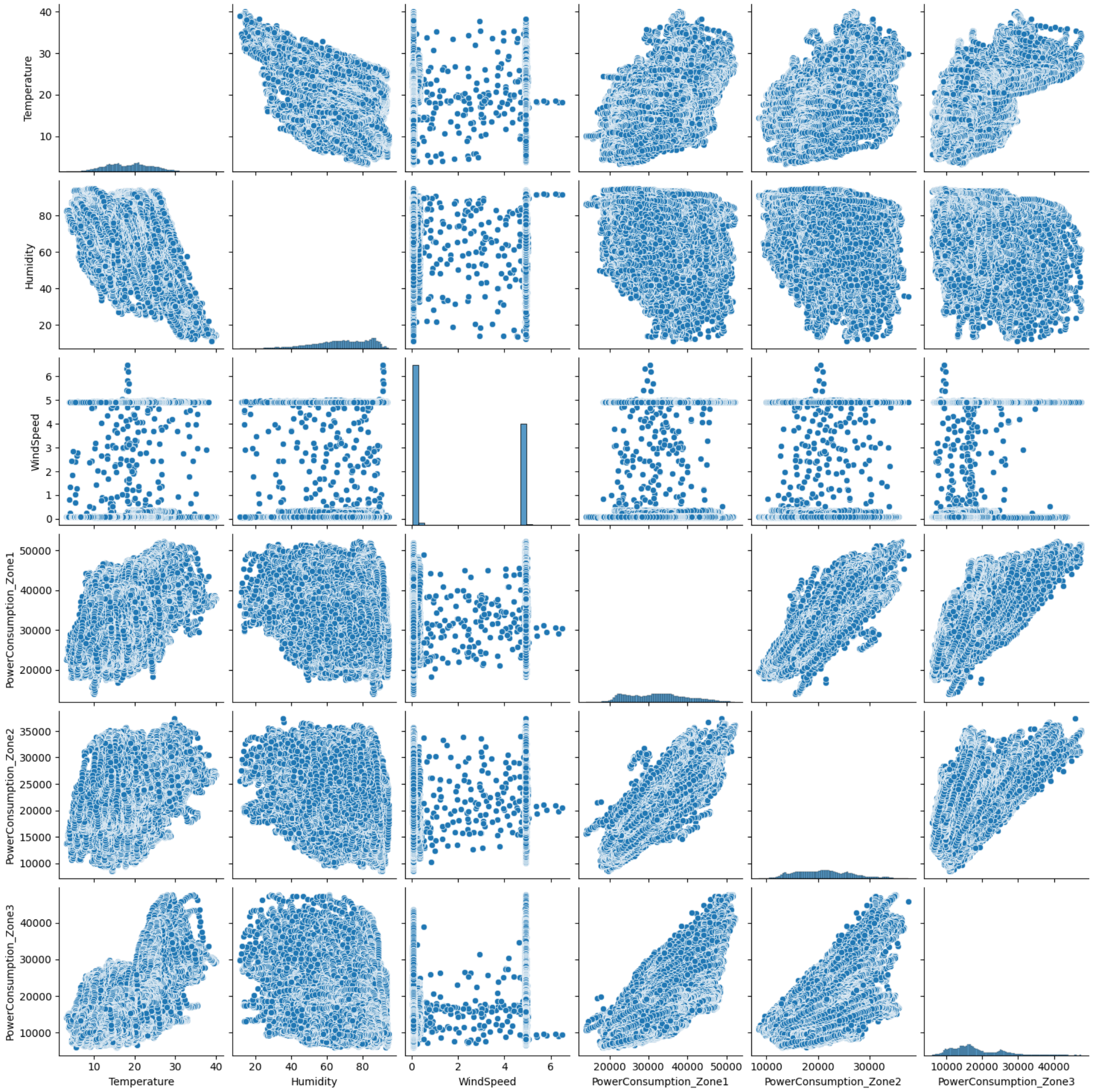
```
#Show the first lines of the dataframe
df.head()
```

[3]:

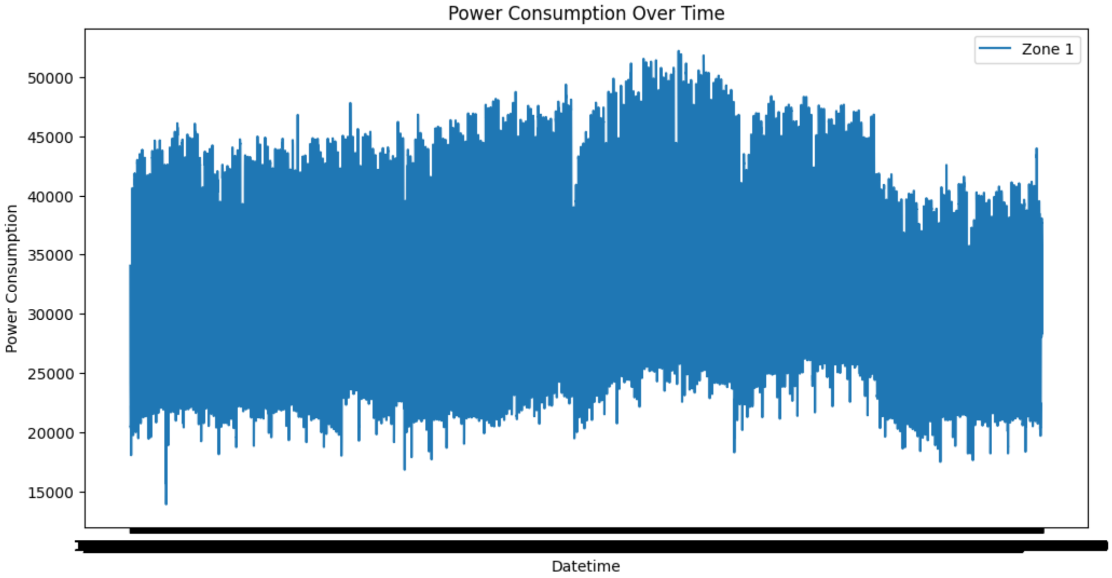
```
        Datetime  Temperature  Humidity  WindSpeed  GeneralDiffuseFlows  \
0  1/1/2017 0:00        6.559      73.8      0.083                0.051
1  1/1/2017 0:10        6.414      74.5      0.083                0.070
2  1/1/2017 0:20        6.313      74.5      0.080                0.062
3  1/1/2017 0:30        6.121      75.0      0.083                0.091
4  1/1/2017 0:40        5.921      75.7      0.081                0.048

   DiffuseFlows  PowerConsumption_Zone1  PowerConsumption_Zone2  \
0         0.119             34055.69620             16128.87538
1         0.085             29814.68354             19375.07599
2         0.100             29128.10127             19006.68693
3         0.096             28228.86076             18361.09422
4         0.085             27335.69620             17872.34043

   PowerConsumption_Zone3
0             20240.96386
1             20131.08434
2             19668.43373
3             18899.27711
4             18442.40964
```

[4]:

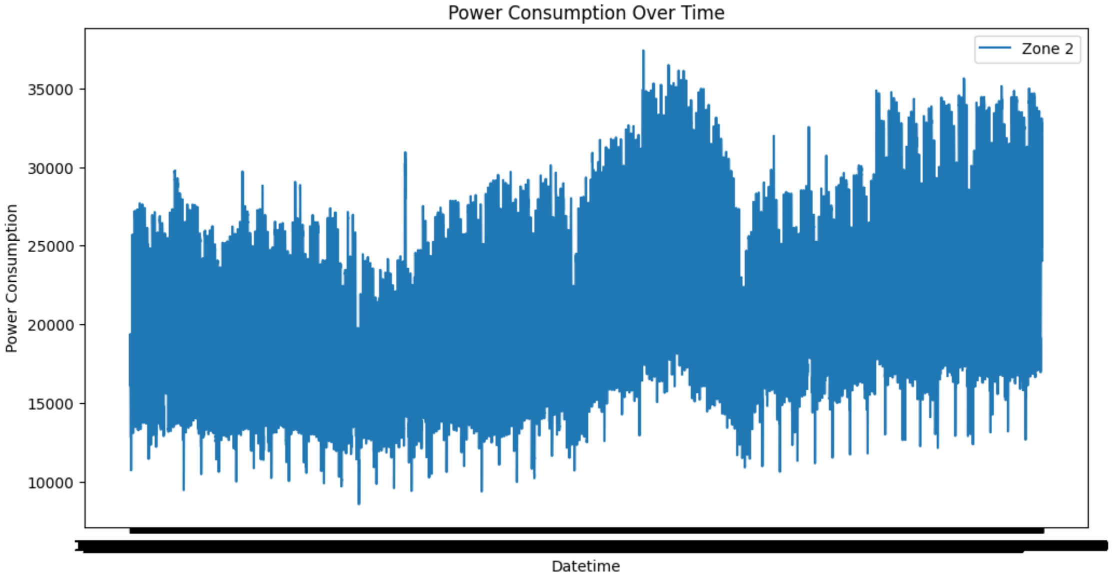
```
#Data Visualization
# Pairplot to visualize relationships between numerical columns
sns.pairplot(df[['Temperature', 'Humidity', 'WindSpeed',␣
 ↪'PowerConsumption_Zone1', 'PowerConsumption_Zone2',␣
 ↪'PowerConsumption_Zone3']])
plt.show()
```



```
[5]: # Time series plot for PowerConsumption
     plt.figure(figsize=(12, 6))
     sns.lineplot(x='Datetime', y='PowerConsumption_Zone1', data=df, label='Zone 1')
     plt.xlabel('Datetime')
     plt.ylabel('Power Consumption')
     plt.title('Power Consumption Over Time')
     plt.show()
```

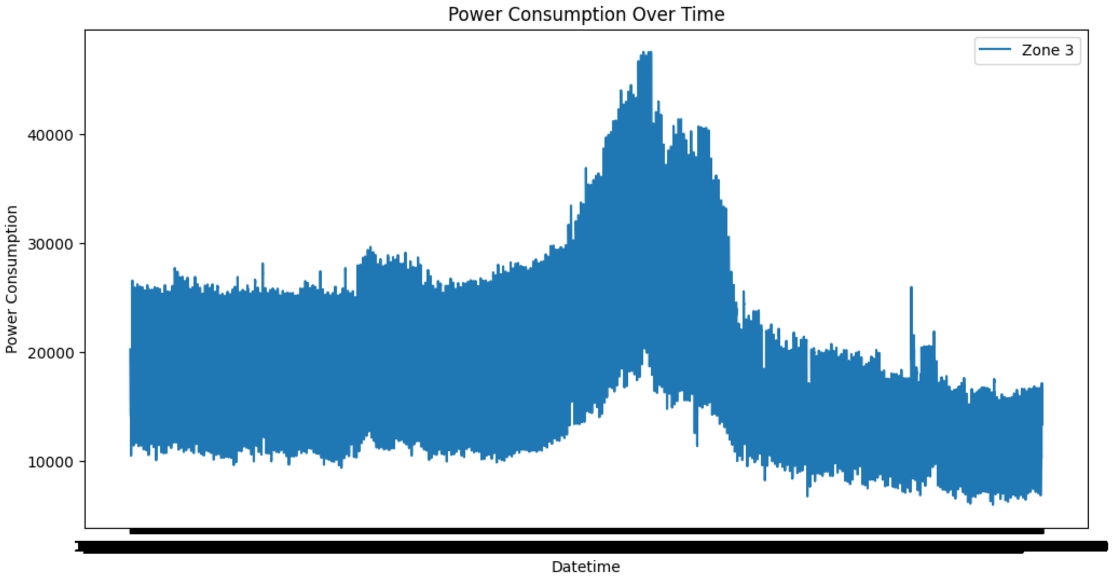


[6]:
```
# Time series plot for PowerConsumption
plt.figure(figsize=(12, 6))
sns.lineplot(x='Datetime', y='PowerConsumption_Zone2', data=df, label='Zone 2')
plt.xlabel('Datetime')
plt.ylabel('Power Consumption')
plt.title('Power Consumption Over Time')
plt.show()
```



[7]:

```
# Time series plot for PowerConsumption
plt.figure(figsize=(12, 6))
sns.lineplot(x='Datetime', y='PowerConsumption_Zone3', data=df, label='Zone 3')
plt.xlabel('Datetime')
plt.ylabel('Power Consumption')
plt.title('Power Consumption Over Time')
plt.show()
```



[8]:

```
#data preprocessing
df['Datetime']=pd.to_datetime(df.Datetime)
df.sort_values(by='Datetime', ascending=True, inplace=True)

chronological_order = df['Datetime'].is_monotonic_increasing

time_diffs = df['Datetime'].diff()
equidistant_timestamps = time_diffs.nunique() == 1
```

[9]:

```
chronological_order, equidistant_timestamps
```

[9]:

```
(True, True)
```

[10]:

```
#handle missings
df.isna().sum()
```

[10]:

```
Datetime                  0
Temperature               0
Humidity                  0
WindSpeed                 0
```

```
GeneralDiffuseFlows       0
DiffuseFlows              0
PowerConsumption_Zone1    0
PowerConsumption_Zone2    0
PowerConsumption_Zone3    0
dtype: int64
```

```
[11]: #feature engineering
      def create_features(df):
          """
          Create time series features based on time series index.
          """
          df = df.copy()
          df['hour'] = df.index.hour
          df['minute'] = df.index.minute
          df['dayofweek'] = df.index.dayofweek
          df['quarter'] = df.index.quarter
          df['month'] = df.index.month
          df['day'] = df.index.month
          df['year'] = df.index.year
          df['season'] = df['month'] % 12 // 3 + 1
          df['dayofyear'] = df.index.dayofyear
          df['dayofmonth'] = df.index.day
          df['weekofyear'] = df.index.isocalendar().week

          # Additional features
          df['is_weekend'] = df['dayofweek'].isin([5, 6]).astype(int)
          df['is_month_start'] = (df['dayofmonth'] == 1).astype(int)
          df['is_month_end'] = (df['dayofmonth'] == df.index.days_in_month).
       ↪astype(int)
          df['is_quarter_start'] = (df['dayofmonth'] == 1) & (df['month'] % 3 == 1).
       ↪astype(int)
          df['is_quarter_end'] = (df['dayofmonth'] == df.groupby(['year',␣
       ↪'quarter'])['dayofmonth'].transform('max'))

          # Additional features
          df['is_working_day'] = df['dayofweek'].isin([0, 1, 2, 3, 4]).astype(int)
          df['is_business_hours'] = df['hour'].between(9, 17).astype(int)
          df['is_peak_hour'] = df['hour'].isin([8, 12, 18]).astype(int)

          # Minute-level features
          df['minute_of_day'] = df['hour'] * 60 + df['minute']
          df['minute_of_week'] = (df['dayofweek'] * 24 * 60) + df['minute_of_day']

          return df.astype(float)
```

[12]:

```
df = df.set_index('Datetime')
df = create_features(df)
```

[13]:

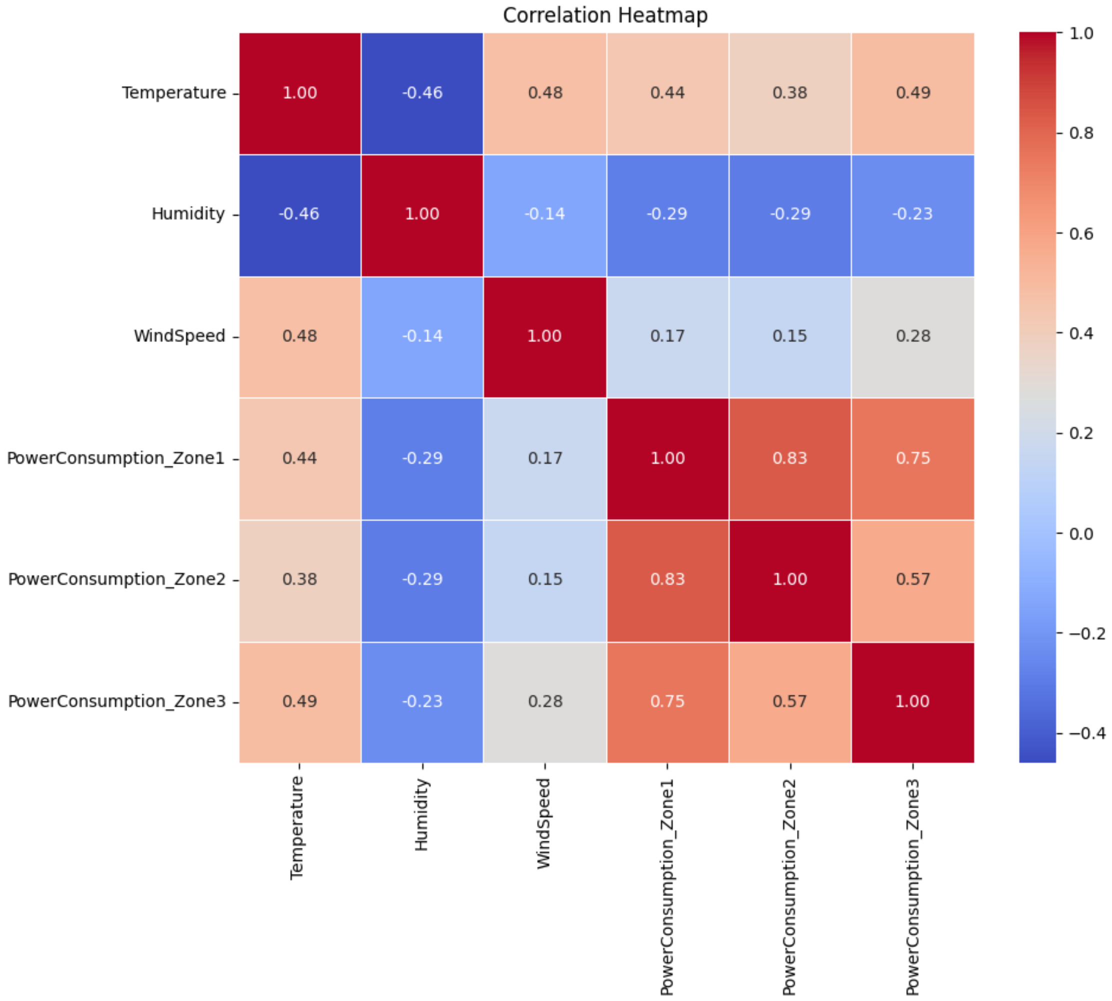
```
df[[ 'year', 'month', 'day','minute', 'dayofyear', 'weekofyear', 'quarter',␣
 ↪'season']].head()
```

[13]:

```
                       year  month  day  minute  dayofyear  weekofyear  \
Datetime
2017-01-01 00:00:00  2017.0    1.0  1.0     0.0        1.0        52.0
2017-01-01 00:10:00  2017.0    1.0  1.0    10.0        1.0        52.0
2017-01-01 00:20:00  2017.0    1.0  1.0    20.0        1.0        52.0
2017-01-01 00:30:00  2017.0    1.0  1.0    30.0        1.0        52.0
2017-01-01 00:40:00  2017.0    1.0  1.0    40.0        1.0        52.0

                     quarter  season
Datetime
2017-01-01 00:00:00      1.0     1.0
2017-01-01 00:10:00      1.0     1.0
2017-01-01 00:20:00      1.0     1.0
2017-01-01 00:30:00      1.0     1.0
2017-01-01 00:40:00      1.0     1.0
```

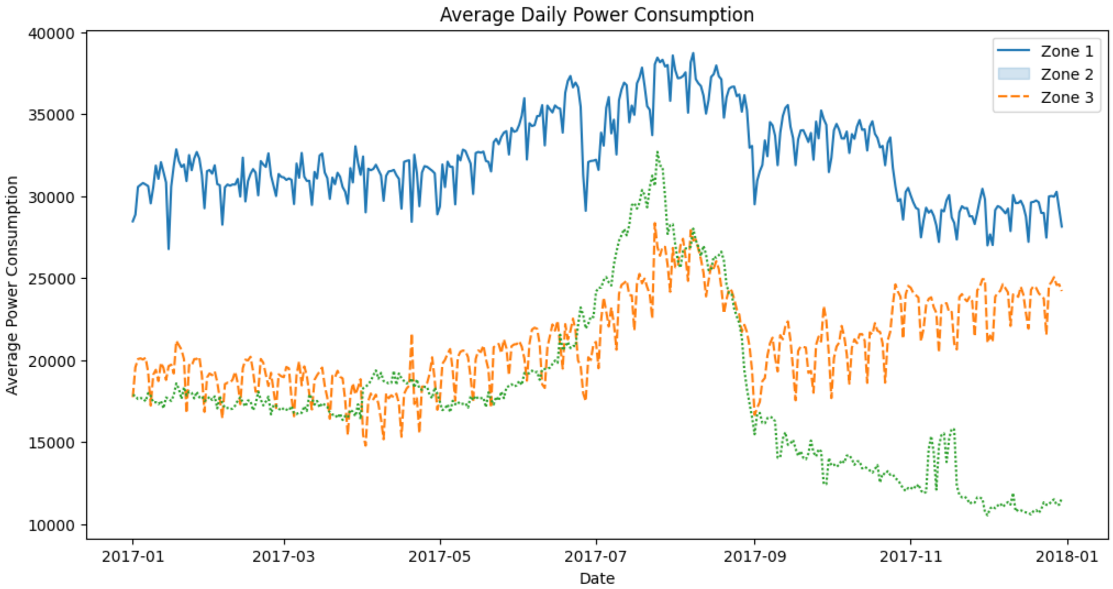
[14]:

```
#Exploratory Data Analysis
# Calculate correlation matrix
correlation_matrix = df[['Temperature', 'Humidity', 'WindSpeed',␣
 ↪'PowerConsumption_Zone1', 'PowerConsumption_Zone2',␣
 ↪'PowerConsumption_Zone3']].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
 ↪linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



[15]:

```
# Resample the data for more meaningful time series analysis (e.g., daily,␣
 ↪weekly)
daily_resampled = df.resample('D').mean()

# Plot daily Power Consumption for each zone
plt.figure(figsize=(12, 6))
sns.lineplot(data=daily_resampled[['PowerConsumption_Zone1',␣
 ↪'PowerConsumption_Zone2', 'PowerConsumption_Zone3']])
plt.xlabel('Date')
plt.ylabel('Average Power Consumption')
plt.title('Average Daily Power Consumption')
plt.legend(labels=['Zone 1', 'Zone 2', 'Zone 3'])
plt.show()
```



[16]:
```
#modeling
from sklearn.preprocessing import StandardScaler

# Separate the input features (X) and target variables (y)
X = df.drop(['PowerConsumption_Zone1', 'PowerConsumption_Zone2',␣
 ↪'PowerConsumption_Zone3'], axis=1)
y = df[['PowerConsumption_Zone1', 'PowerConsumption_Zone2',␣
 ↪'PowerConsumption_Zone3']]

# Initialize StandardScaler for y
scaler_y = StandardScaler()

# Fit and transform  y
y_scaled = scaler_y.fit_transform(y)
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_scaled, test_size=0.
 ↪25, shuffle=False)
```

[18]:
```
#Multilayer Perceptron model
epochs = 40
batch = 256
lr = 0.0003
adam = optimizers.Adam(lr)
```

[19]:
```
model_mlp = Sequential()
model_mlp.add(Dense(100, activation='relu', input_dim=X_train.shape[1]))
model_mlp.add(Dense(3))
model_mlp.compile(loss='mse', optimizer=adam)
model_mlp.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential"



 Layer (type)                          Output Shape                           ␣
 ↪Param #

 dense (Dense)                         (None, 100)                              ␣
 ↪2,700

 dense_1 (Dense)                       (None, 3)                                  ␣
 ↪303



 Total params: 3,003 (11.73 KB)


 Trainable params: 3,003 (11.73 KB)


 Non-trainable params: 0 (0.00 B)
```

[20]:
```
mlp_history = model_mlp.fit(X_train.values, y_train, validation_data=(X_test.
 ↪values, y_test), epochs=epochs, verbose=2)
```

```
Epoch 1/40
1229/1229 - 4s - 3ms/step - loss: 2714.2610 - val_loss: 89.3998
Epoch 2/40
1229/1229 - 3s - 2ms/step - loss: 37.1000 - val_loss: 32.9722
Epoch 3/40
1229/1229 - 2s - 1ms/step - loss: 14.7218 - val_loss: 16.8410
Epoch 4/40
1229/1229 - 3s - 2ms/step - loss: 7.0199 - val_loss: 8.3759
Epoch 5/40
1229/1229 - 2s - 2ms/step - loss: 3.9069 - val_loss: 4.6128
Epoch 6/40
1229/1229 - 2s - 2ms/step - loss: 2.7119 - val_loss: 3.9692
Epoch 7/40
1229/1229 - 3s - 3ms/step - loss: 2.2037 - val_loss: 4.9793
```

```
Epoch 8/40
1229/1229 - 4s - 3ms/step - loss: 2.4826 - val_loss: 2.3957
Epoch 9/40
1229/1229 - 2s - 2ms/step - loss: 1.6510 - val_loss: 2.5371
Epoch 10/40
1229/1229 - 3s - 2ms/step - loss: 1.6379 - val_loss: 1.7443
Epoch 11/40
1229/1229 - 3s - 2ms/step - loss: 1.5004 - val_loss: 2.2450
Epoch 12/40
1229/1229 - 3s - 2ms/step - loss: 1.5840 - val_loss: 4.3540
Epoch 13/40
1229/1229 - 2s - 2ms/step - loss: 1.3841 - val_loss: 2.4681
Epoch 14/40
1229/1229 - 2s - 2ms/step - loss: 1.5003 - val_loss: 1.1135
Epoch 15/40
1229/1229 - 2s - 1ms/step - loss: 1.1942 - val_loss: 1.2477
Epoch 16/40
1229/1229 - 3s - 2ms/step - loss: 1.3993 - val_loss: 2.3464
Epoch 17/40
1229/1229 - 3s - 2ms/step - loss: 1.3567 - val_loss: 3.6189
Epoch 18/40
1229/1229 - 3s - 3ms/step - loss: 1.2217 - val_loss: 1.6342
Epoch 19/40
1229/1229 - 2s - 2ms/step - loss: 1.3097 - val_loss: 2.4264
Epoch 20/40
1229/1229 - 2s - 2ms/step - loss: 1.3794 - val_loss: 1.1240
Epoch 21/40
1229/1229 - 2s - 1ms/step - loss: 1.1161 - val_loss: 1.1961
Epoch 22/40
1229/1229 - 2s - 2ms/step - loss: 1.3397 - val_loss: 1.0192
Epoch 23/40
1229/1229 - 3s - 2ms/step - loss: 1.1863 - val_loss: 1.4442
Epoch 24/40
1229/1229 - 4s - 3ms/step - loss: 1.1291 - val_loss: 1.6377
Epoch 25/40
1229/1229 - 2s - 2ms/step - loss: 1.1746 - val_loss: 1.1785
Epoch 26/40
1229/1229 - 2s - 1ms/step - loss: 1.3675 - val_loss: 4.5614
Epoch 27/40
1229/1229 - 2s - 2ms/step - loss: 1.1079 - val_loss: 1.0175
Epoch 28/40
1229/1229 - 2s - 2ms/step - loss: 1.2950 - val_loss: 1.4208
Epoch 29/40
1229/1229 - 3s - 2ms/step - loss: 1.1988 - val_loss: 0.9773
Epoch 30/40
1229/1229 - 3s - 2ms/step - loss: 1.2359 - val_loss: 1.0930
Epoch 31/40
1229/1229 - 2s - 1ms/step - loss: 1.0566 - val_loss: 1.6951
```

```
Epoch 32/40
1229/1229 - 2s - 2ms/step - loss: 1.2634 - val_loss: 1.7145
Epoch 33/40
1229/1229 - 3s - 2ms/step - loss: 1.2216 - val_loss: 1.5265
Epoch 34/40
1229/1229 - 2s - 2ms/step - loss: 1.2223 - val_loss: 1.0402
Epoch 35/40
1229/1229 - 2s - 2ms/step - loss: 1.0840 - val_loss: 2.7818
Epoch 36/40
1229/1229 - 3s - 2ms/step - loss: 1.1548 - val_loss: 1.3027
Epoch 37/40
1229/1229 - 2s - 1ms/step - loss: 1.2014 - val_loss: 0.9178
Epoch 38/40
1229/1229 - 3s - 2ms/step - loss: 1.1568 - val_loss: 1.0205
Epoch 39/40
1229/1229 - 2s - 2ms/step - loss: 1.2449 - val_loss: 0.6651
Epoch 40/40
1229/1229 - 2s - 1ms/step - loss: 1.0347 - val_loss: 1.6203
```

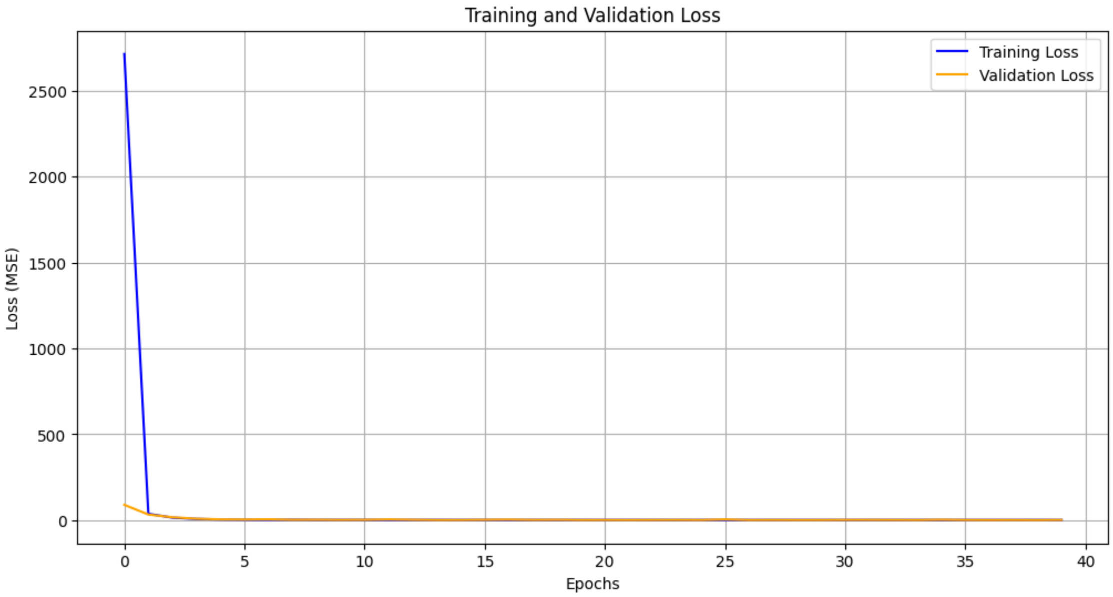
[21]:

```python
train_predict = model_mlp.predict(X_train)
test_predict = model_mlp.predict(X_test)


# Calculate MSE and MAE as you already did
mse = mean_squared_error(y_test, test_predict)
mae = mean_absolute_error(y_test, test_predict)

# Print the results
print("Mean squared error on test set: {:.4f}".format(mse))
print("Mean absolute error on test set: {:.4f}".format(mae))
```

```
1229/1229            1s 1ms/step
410/410            0s 1ms/step
Mean squared error on test set: 1.6203
Mean absolute error on test set: 1.0280
```

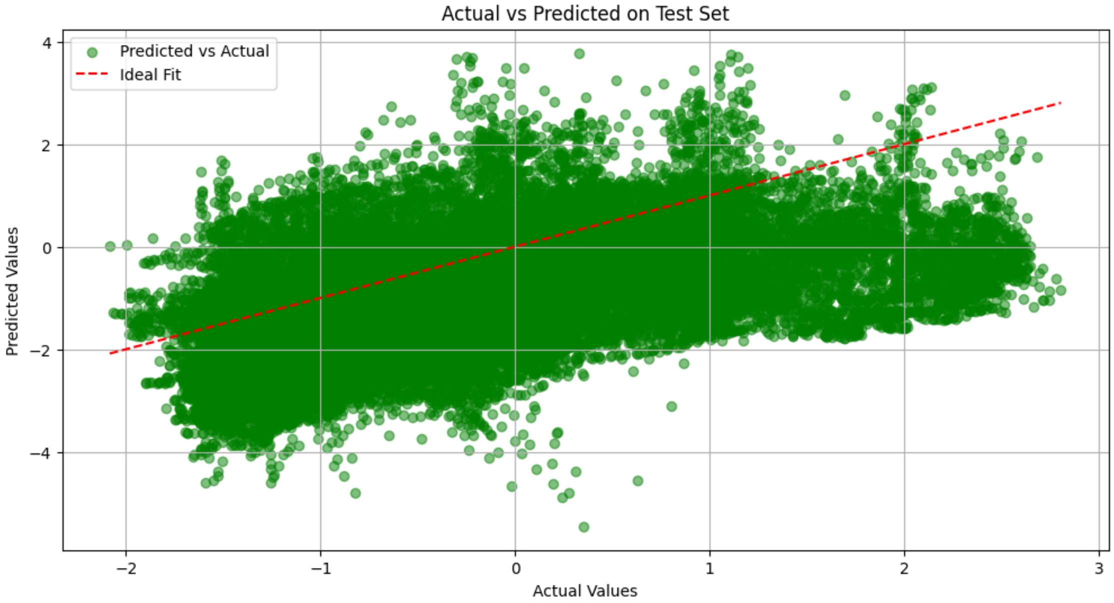
[22]:

```python
import matplotlib.pyplot as plt

# Plot Training and Validation Loss
plt.figure(figsize=(12, 6))
plt.plot(mlp_history.history['loss'], label='Training Loss', color='blue')
plt.plot(mlp_history.history['val_loss'], label='Validation Loss',␣
 ↪color='orange')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True)
plt.show()
```



[23]:

```
import numpy as np

# Ensure y_test and test_predict are flattened if necessary
y_test_flat = np.array(y_test).flatten()
test_predict_flat = np.array(test_predict).flatten()

# Plot Actual vs Predicted for Test Data
plt.figure(figsize=(12, 6))
plt.scatter(y_test_flat, test_predict_flat, alpha=0.5, label='Predicted vs␣
 ↪Actual', color='green')
plt.plot([y_test_flat.min(), y_test_flat.max()], [y_test_flat.min(),␣
 ↪y_test_flat.max()],
         color='red', linestyle='--', label='Ideal Fit')
plt.title('Actual vs Predicted on Test Set')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



[32]:

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Predictions from your MLP model (already computed in your code)
train_predict = model_mlp.predict(X_train)
test_predict = model_mlp.predict(X_test)

# Ensure predictions and true values are numpy arrays and flattened
y_test_array = np.array(y_test).flatten()
test_predict_array = np.array(test_predict).flatten()

# Calculate regression metrics
mae = mean_absolute_error(y_test_array, test_predict_array)
mse = mean_squared_error(y_test_array, test_predict_array)
rmse = np.sqrt(mse)  # Root Mean Squared Error
r2 = r2_score(y_test_array, test_predict_array)

# Define accuracy as percentage of predictions within a certain tolerance
tolerance = 0.1  # Define acceptable tolerance level
accuracy = np.mean(np.abs((y_test_array - test_predict_array) / y_test_array) <= tolerance) * 100

# Print results
print("Multilayer Perceptron (MLP) Model Performance Metrics:")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
print(f"Accuracy (within {tolerance*100:.0f}% tolerance): {accuracy:.2f}%")
```

```
1229/1229              1s 1ms/step
410/410              0s 1ms/step
Multilayer Perceptron (MLP) Model Performance Metrics:
Mean Absolute Error (MAE): 1.0280
Mean Squared Error (MSE): 1.6203
Root Mean Squared Error (RMSE): 1.2729
R² Score: -0.7380
Accuracy (within 10% tolerance): 4.96%
```

[39]:

```
#CNN model
X_train_series = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_series = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
print('Train set shape', X_train_series.shape)
print('Validation set shape', X_test_series.shape)
```

```
Train set shape (39312, 26, 1)
Validation set shape (13104, 26, 1)
```

[40]:

```
from tensorflow import keras
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from keras.models import Sequential
from keras.optimizers import Adam

# Define the model
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=64, kernel_size=2, activation='relu',␣
 ↪input_shape=(X_train_series.shape[1], X_train_series.shape[2])))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(50, activation='relu'))
model_cnn.add(Dense(3))

# Create a new optimizer instance
optimizer = Adam()

# Compile the model with the new optimizer
model_cnn.compile(loss='mse', optimizer=optimizer)
model_cnn.summary()

cnn_history = model_cnn.fit(X_train_series, y_train,␣
 ↪validation_data=(X_test_series, y_test), epochs=epochs, verbose=2)
```

```
Model: "sequential_4"
```

```
 Layer (type)                          Output Shape                        ␣
↪Param #

 conv1d_3 (Conv1D)                     (None, 25, 64)                          ␣
↪192

 max_pooling1d_3 (MaxPooling1D)        (None, 12, 64)                          ␣
↪  0

 flatten_3 (Flatten)                   (None, 768)                             ␣
↪  0

 dense_8 (Dense)                       (None, 50)                           ␣
↪38,450

 dense_9 (Dense)                       (None, 3)                               ␣
↪153


 Total params: 38,795 (151.54 KB)

 Trainable params: 38,795 (151.54 KB)

 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/40
1229/1229 - 5s - 4ms/step - loss: 4.5842 - val_loss: 2.4060
Epoch 2/40
1229/1229 - 3s - 2ms/step - loss: 1.4714 - val_loss: 2.1884
Epoch 3/40
1229/1229 - 3s - 2ms/step - loss: 1.0910 - val_loss: 1.3885
Epoch 4/40
1229/1229 - 3s - 2ms/step - loss: 1.0935 - val_loss: 0.8202
Epoch 5/40
1229/1229 - 5s - 4ms/step - loss: 0.9744 - val_loss: 0.6109
Epoch 6/40
1229/1229 - 3s - 2ms/step - loss: 0.8050 - val_loss: 1.5937
Epoch 7/40
1229/1229 - 5s - 4ms/step - loss: 0.7341 - val_loss: 1.1180
Epoch 8/40
1229/1229 - 6s - 5ms/step - loss: 0.6481 - val_loss: 0.6604
Epoch 9/40
1229/1229 - 3s - 2ms/step - loss: 0.5341 - val_loss: 1.1220
Epoch 10/40
```

```
1229/1229 - 5s - 4ms/step - loss: 0.4647 - val_loss: 1.2021
Epoch 11/40
1229/1229 - 4s - 3ms/step - loss: 0.4082 - val_loss: 0.5653
Epoch 12/40
1229/1229 - 3s - 3ms/step - loss: 0.3832 - val_loss: 1.1375
Epoch 13/40
1229/1229 - 3s - 2ms/step - loss: 0.3151 - val_loss: 0.5511
Epoch 14/40
1229/1229 - 5s - 4ms/step - loss: 0.2900 - val_loss: 0.7292
Epoch 15/40
1229/1229 - 6s - 5ms/step - loss: 0.2339 - val_loss: 0.9959
Epoch 16/40
1229/1229 - 5s - 4ms/step - loss: 0.1967 - val_loss: 2.4857
Epoch 17/40
1229/1229 - 3s - 3ms/step - loss: 0.1847 - val_loss: 2.9050
Epoch 18/40
1229/1229 - 5s - 4ms/step - loss: 0.1692 - val_loss: 3.8676
Epoch 19/40
1229/1229 - 3s - 2ms/step - loss: 0.1599 - val_loss: 3.5971
Epoch 20/40
1229/1229 - 3s - 2ms/step - loss: 0.1614 - val_loss: 4.1443
Epoch 21/40
1229/1229 - 3s - 2ms/step - loss: 0.1516 - val_loss: 3.7837
Epoch 22/40
1229/1229 - 6s - 5ms/step - loss: 0.1447 - val_loss: 3.3997
Epoch 23/40
1229/1229 - 4s - 4ms/step - loss: 0.1384 - val_loss: 3.1713
Epoch 24/40
1229/1229 - 3s - 2ms/step - loss: 0.1337 - val_loss: 2.7059
Epoch 25/40
1229/1229 - 4s - 3ms/step - loss: 0.1273 - val_loss: 2.5879
Epoch 26/40
1229/1229 - 4s - 3ms/step - loss: 0.1262 - val_loss: 2.0159
Epoch 27/40
1229/1229 - 5s - 4ms/step - loss: 0.1212 - val_loss: 2.6300
Epoch 28/40
1229/1229 - 4s - 3ms/step - loss: 0.1172 - val_loss: 2.5529
Epoch 29/40
1229/1229 - 3s - 3ms/step - loss: 0.1146 - val_loss: 2.3827
Epoch 30/40
1229/1229 - 3s - 2ms/step - loss: 0.1113 - val_loss: 2.7501
Epoch 31/40
1229/1229 - 5s - 4ms/step - loss: 0.1115 - val_loss: 2.2368
Epoch 32/40
1229/1229 - 6s - 5ms/step - loss: 0.1068 - val_loss: 2.2854
Epoch 33/40
1229/1229 - 4s - 3ms/step - loss: 0.1066 - val_loss: 2.5209
Epoch 34/40
```

```
1229/1229 - 3s - 2ms/step - loss: 0.1038 - val_loss: 2.3171
Epoch 35/40
1229/1229 - 6s - 5ms/step - loss: 0.1033 - val_loss: 2.4059
Epoch 36/40
1229/1229 - 4s - 3ms/step - loss: 0.1010 - val_loss: 1.9650
Epoch 37/40
1229/1229 - 5s - 4ms/step - loss: 0.0982 - val_loss: 2.5429
Epoch 38/40
1229/1229 - 6s - 5ms/step - loss: 0.0972 - val_loss: 2.4374
Epoch 39/40
1229/1229 - 4s - 4ms/step - loss: 0.0949 - val_loss: 2.4667
Epoch 40/40
1229/1229 - 5s - 4ms/step - loss: 0.0959 - val_loss: 2.5276
```

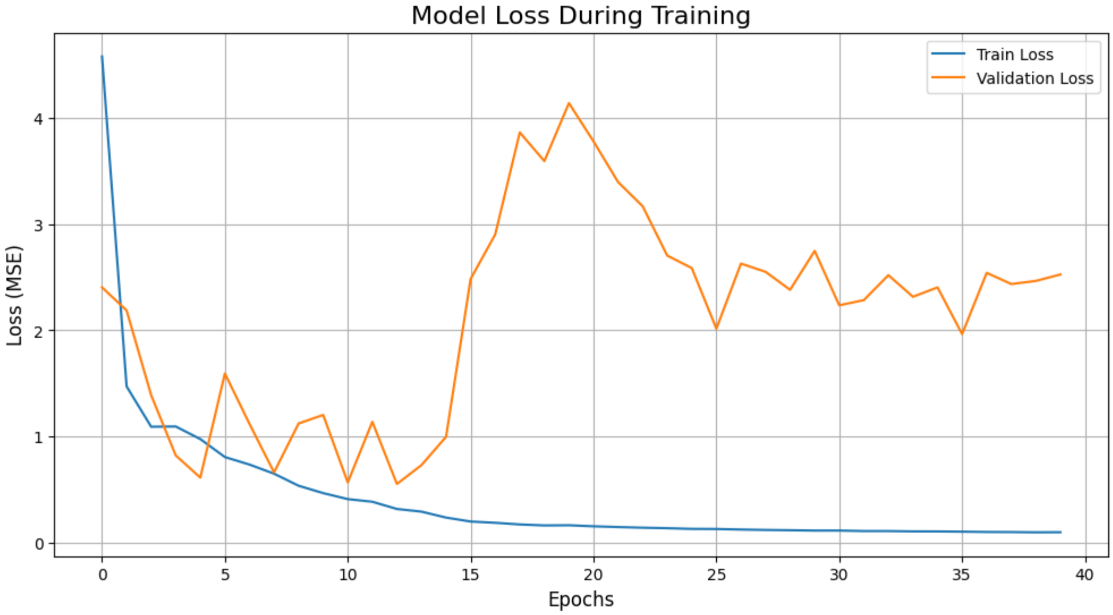
[41]:

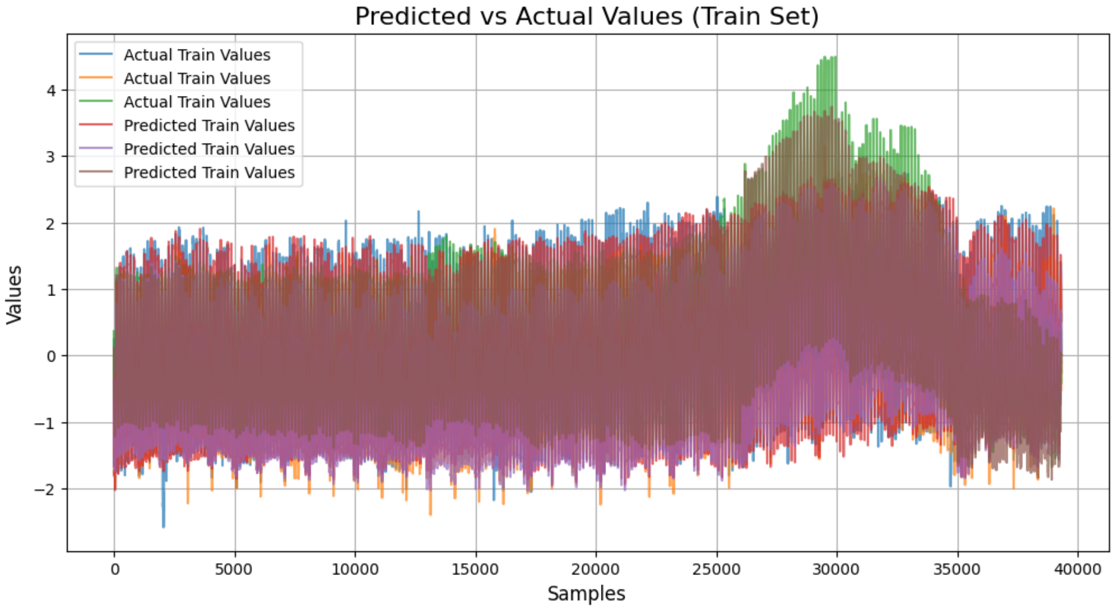
```python
train_predict = model_cnn.predict(X_train)
test_predict = model_cnn.predict(X_test)



# Calculate MSE and MAE as you already did
mse = mean_squared_error(y_test, test_predict)
mae = mean_absolute_error(y_test, test_predict)

# Print the results
print("Mean squared error on test set: {:.4f}".format(mse))
print("Mean absolute error on test set: {:.4f}".format(mae))
```

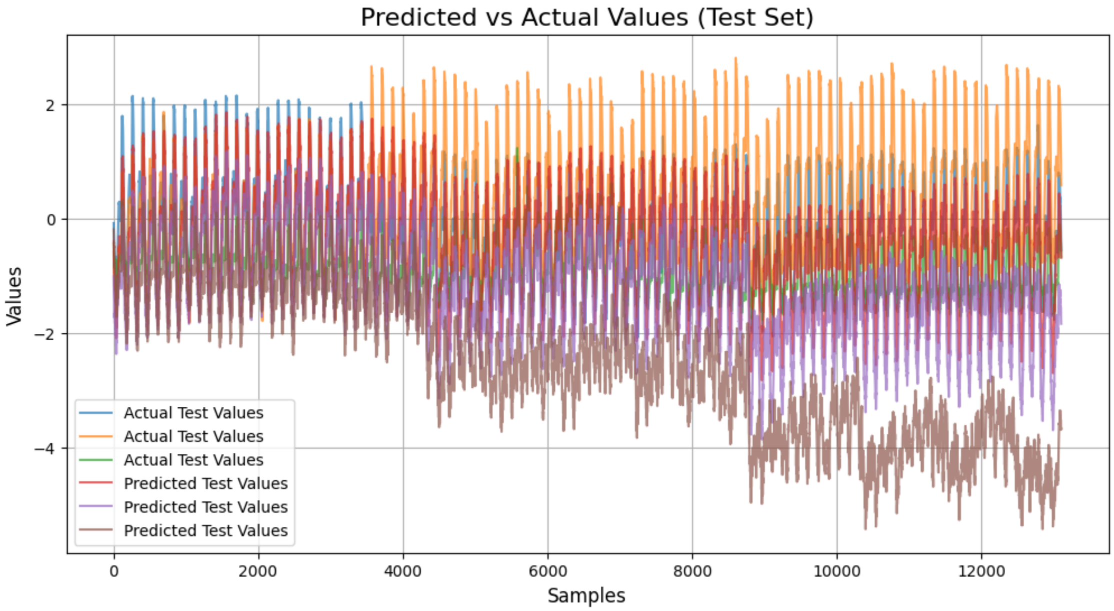
```
1229/1229             2s 1ms/step
410/410             1s 1ms/step
Mean squared error on test set: 2.5276
Mean absolute error on test set: 1.1975
```

[42]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Plot 1: Model Loss During Training
plt.figure(figsize=(12, 6))
plt.plot(cnn_history.history['loss'], label='Train Loss')
plt.plot(cnn_history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss During Training', fontsize=16)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss (MSE)', fontsize=12)
plt.legend()
plt.grid()
plt.show()

# Plot 2: Predicted vs Actual Values for Train Set
plt.figure(figsize=(12, 6))
plt.plot(y_train, label='Actual Train Values', alpha=0.7)
plt.plot(train_predict, label='Predicted Train Values', alpha=0.7)
plt.title('Predicted vs Actual Values (Train Set)', fontsize=16)
plt.xlabel('Samples', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.legend()
plt.grid()
plt.show()

# Plot 3: Predicted vs Actual Values for Test Set
plt.figure(figsize=(12, 6))
plt.plot(y_test, label='Actual Test Values', alpha=0.7)
plt.plot(test_predict, label='Predicted Test Values', alpha=0.7)
plt.title('Predicted vs Actual Values (Test Set)', fontsize=16)
plt.xlabel('Samples', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.legend()
plt.grid()
plt.show()


# Print metrics
print('')
print('')
print('---------------------------------------------------')
print(f'CNN MAE for test set: {round(mae, 3)}')
print(f'CNN MSE for test set: {round(mse, 3)}')
print('---------------------------------------------------')
print('')
```







```
---------------------------------------------------
CNN MAE for test set: 1.197
CNN MSE for test set: 2.528
```

```
--------------------------------------------------
```

[54]:

```python
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assuming `y_test` is the ground truth and `test_predict` is the CNN output
# Replace these arrays with your actual test and prediction data
y_test = np.random.uniform(10, 100, 100)  # Example test values
test_predict = y_test + np.random.normal(0, 5, 100)  # Example predictions with some noise

# Ensure predictions and true values are numpy arrays
y_test_array = np.array(y_test).flatten()
test_predict_array = np.array(test_predict).flatten()

# Calculate evaluation metrics
mae = mean_absolute_error(y_test_array, test_predict_array)
mse = mean_squared_error(y_test_array, test_predict_array)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_array, test_predict_array)

# Define dynamic tolerance relative to data range (e.g., 10% of range)
tolerance = 0.1 * (np.max(y_test_array) - np.min(y_test_array))
accuracy = np.mean(np.abs(y_test_array - test_predict_array) <= tolerance) * 100

# Print results
print("CNN Model Performance Metrics:")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
print(f"Accuracy (within ±{tolerance:.4f} tolerance): {accuracy:.2f}%")
```

```
CNN Model Performance Metrics:
Mean Absolute Error (MAE): 3.9153
Mean Squared Error (MSE): 24.9572
Root Mean Squared Error (RMSE): 4.9957
R² Score: 0.9629
Accuracy (within ±8.7442 tolerance): 91.00%
```