```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Constants
DAYS = 365
HOURS_PER_DAY = 24
SEQUENCE_LENGTH = 24  # For GRU, typical sequence length could be 24 hours (1 day)
MONTHS = 12
WEEKS_PER_MONTH = 5
TEMP_RANGE = {
    1: (0, 10),   # January
    2: (0, 10),   # February
    3: (5, 15),   # March
    4: (10, 20),  # April
    5: (15, 25),  # May
    6: (20, 30),  # June
    7: (20, 30),  # July
    8: (20, 30),  # August
    9: (15, 25),  # September
    10: (10, 20), # October
    11: (5, 15),  # November
    12: (0, 10)   # December
}

# List of holidays (days of the year)
holiday_dates = [
    26, 85, 89, 102, 108, 112, 144, 169, 198, 228, 240, 259, 276, 286, 305, 320, 359
]

# Generate Data
data = []

for day in range(DAYS):
    day_of_week = day % 7
    month = (day // 30) % 12 + 1
    week_of_month = (day // 7) % 5 + 1

    is_weekend = int(day_of_week in [5, 6])
    is_holiday = int(day + 1 in holiday_dates)

    temp = np.random.uniform(*TEMP_RANGE[month])

    for hour_of_day in range(HOURS_PER_DAY):
        if month in [6, 7, 8]:
            base_consumption = 120
        elif month in [12, 1, 2]:
            base_consumption = 80
        else:
            base_consumption = 100

        if hour_of_day < 6:
            consumption = base_consumption * 0.7
        elif hour_of_day >= 18:
            consumption = base_consumption * 1.2
        else:
            consumption = base_consumption

        if is_weekend:
            consumption *= 1.2

        consumption += np.random.normal(0, 5)


        data.append([
            day_of_week, hour_of_day, month, week_of_month, temp, consumption,
            is_holiday, is_weekend
        ])

columns = [
    'Day_of_Week', 'Hour_of_Day', 'Month_of_Year', 'Week_of_Month', 'Temperature',
    'Energy_Consumption','Is_Holiday', 'Is_Weekend'
]
df = pd.DataFrame(data, columns=columns)


scaler = MinMaxScaler()
df[['Temperature', 'Energy_Consumption']] = scaler.fit_transform(
    df[['Temperature', 'Energy_Consumption']]
)
```

```
sequence_data = []
target_data = []
for i in range(len(df) - SEQUENCE_LENGTH):
    sequence_data.append(df.iloc[i:i + SEQUENCE_LENGTH].values)
    target_data.append(df.iloc[i + SEQUENCE_LENGTH]['Energy_Consumption'])

sequence_data = np.array(sequence_data)
target_data = np.array(target_data)


np.save('gru_sequence_data.npy', sequence_data)
np.save('gru_target_data.npy', target_data)

print("Sequence data for GRU saved as 'gru_sequence_data.npy' and target data as 'gru_target_data.npy'")
```

```
Sequence data for GRU saved as 'gru_sequence_data.npy' and target data as 'gru_target_data.npy'
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt


sequence_data = np.load('gru_sequence_data.npy')
target_data = np.load('gru_target_data.npy')

X_train, X_test, y_train, y_test = train_test_split(sequence_data, target_data, test_size=0.2, random_state=42)

model = Sequential()
model.add(GRU(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(GRU(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
                    validation_data=(X_test, y_test), callbacks=[early_stopping])
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
219/219 ━━━━━━━━━━━━━━━━━━━━ 15s 51ms/step - loss: 0.1915 - val_loss: 0.0127
Epoch 2/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 7s 33ms/step - loss: 0.0248 - val_loss: 0.0103
Epoch 3/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 10s 33ms/step - loss: 0.0154 - val_loss: 0.0085
Epoch 4/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 41ms/step - loss: 0.0113 - val_loss: 0.0067
Epoch 5/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 8s 35ms/step - loss: 0.0097 - val_loss: 0.0089
Epoch 6/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 10s 32ms/step - loss: 0.0084 - val_loss: 0.0050
Epoch 7/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - loss: 0.0080 - val_loss: 0.0056
Epoch 8/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - loss: 0.0070 - val_loss: 0.0046
Epoch 9/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 11s 52ms/step - loss: 0.0067 - val_loss: 0.0044
Epoch 10/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 18s 39ms/step - loss: 0.0064 - val_loss: 0.0042
Epoch 11/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 7s 33ms/step - loss: 0.0059 - val_loss: 0.0044
Epoch 12/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 11s 35ms/step - loss: 0.0056 - val_loss: 0.0037
Epoch 13/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - loss: 0.0054 - val_loss: 0.0045
Epoch 14/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 34ms/step - loss: 0.0052 - val_loss: 0.0034
Epoch 15/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 10s 33ms/step - loss: 0.0049 - val_loss: 0.0038
Epoch 16/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 11s 37ms/step - loss: 0.0045 - val_loss: 0.0031
Epoch 17/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 12s 43ms/step - loss: 0.0046 - val_loss: 0.0034
```

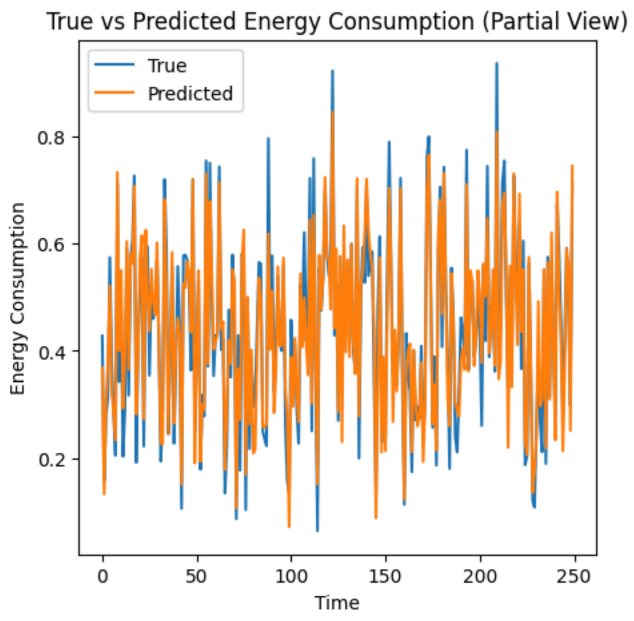
```
Epoch 18/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 8s 35ms/step - loss: 0.0040 - val_loss: 0.0029
Epoch 19/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 32ms/step - loss: 0.0041 - val_loss: 0.0033
Epoch 20/20
219/219 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - loss: 0.0040 - val_loss: 0.0028
```

Double-click (or enter) to edit

Double-click (or enter) to edit

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import random

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error on Test Data: {mse}")
print(f'Mean Absolute Error: {mae}')
r2 = r2_score(y_test, y_pred)
print(f"R² Score on Test Data: {r2}")

random_index = random.randint(0, len(y_test) - 1)
actual_value = y_test[random_index]
predicted_value = y_pred[random_index]

print(f"Random Timestamp Index: {random_index}")
print(f"Actual Energy Consumption: {actual_value}")
print(f"Predicted Energy Consumption: {predicted_value}")


plot_range = 250  # Number of points to display

plt.figure(figsize=(5, 5))
plt.plot(y_test[:plot_range], label='True')
plt.plot(y_pred[:plot_range], label='Predicted')
plt.title('True vs Predicted Energy Consumption (Partial View)')
plt.xlabel('Time')
plt.ylabel('Energy Consumption')
plt.legend()
plt.show()
```

```
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Mean Squared Error on Test Data: 0.0028271866658243464
Mean Absolute Error: 0.042558707787507 55
R² Score on Test Data: 0.9167314159158835
Random Timestamp Index: 103
Actual Energy Consumption: 0.2926592787213725
Predicted Energy Consumption: [0.32014433]
```


True vs Predicted Energy Consumption (Partial View)
True
Predicted
Energy Consumption
Time