**Documentation for assignment 3**

**check\_data.py – Data Sanity Checker**

**Purpose**:  
This script performs a basic integrity check on the preprocessed NumPy arrays used for time series modeling. Specifically, it checks the shape and presence of missing (NaN) values in both the input (X) and output (y) arrays.

**Functionality**:

* Loads input sequences (X.npy) and target labels (y\_both.npy) from the data/sequences/ directory.
* Prints:
  + Shape of the X and y arrays.
  + Total number of values in each array.
  + Number of NaN values in each array.

**Key Output**:

X shape: (samples, time\_steps, features)

y shape: (samples, targets or time\_steps x targets)

Total values in X: N

Total NaNs in X: M

...

**Use Case**:  
Run this script after preprocessing and before model training to ensure:

* Your sequences are shaped as expected (e.g., (N, 168, 10) for 7 days of hourly data).
* There are no missing values that would crash training or skew predictions.

**Dependencies**:

* numpy

**create\_sequences.py – Sequence Generator for Time Series Modeling**

**Purpose**:  
This script converts a preprocessed flat time series dataset (CSV format) into sliding-window sequences suitable for training encoder-decoder RNN models (e.g., LSTM or GRU). It generates supervised input–output pairs using a fixed lookback window and prediction horizon.

**Functionality Summary**:

* Reads a processed dataset from data/processed\_dataset.csv
* Verifies that both target columns (Total Building (kW) and PV (kW)) are present
* Creates sequences of:
  + **Inputs (X)**: sliding window of past values (e.g., past 168 hours = 1 week)
  + **Outputs (y)**: future values over the prediction horizon (e.g., next 24 hours)
* Saves all sequences to .npy format for efficient training

**Key Configuration Parameters**:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| LOOKBACK | Number of time steps used as input (e.g., 168 for 7 days hourly) |
| HORIZON | Number of time steps to predict ahead (e.g., 24 hours) |
| PROCESSED\_FILE | Path to the cleaned CSV file |
| OUTPUT\_DIR | Folder to save .npy arrays |

**Main Function: create\_sequences(X\_df, y\_df)**  
Creates input-output sequence pairs from a DataFrame.

* **Inputs**:
  + X\_df: DataFrame of numeric input features
  + y\_df: DataFrame of target variable(s)
* **Outputs**:
  + X: NumPy array of shape (num\_samples, LOOKBACK, num\_features)
  + y: NumPy array of shape (num\_samples, HORIZON, num\_targets)

**Script Workflow**:

1. Load processed data from CSV
2. Confirm target columns (Total Building (kW) and PV (kW)) exist
3. Separate X and all 3 types of y (total only, PV only, both)
4. Generate input/output sequences using sliding windows
5. Save:
   * X.npy: shared input for all targets
   * y\_total.npy: output for total building prediction
   * y\_pv.npy: output for PV prediction
   * y\_both.npy: output for both targets

**Code Summary**:

X, y\_total = create\_sequences(X\_df, y\_total\_df)

\_, y\_pv = create\_sequences(X\_df, y\_pv\_df)

\_, y\_both = create\_sequences(X\_df, y\_both\_df)

np.save('X.npy', X)

np.save('y\_total.npy', y\_total)

...

**Typical Output (Shapes)**:

X shape: (n\_samples, 168, num\_features)

y\_total shape: (n\_samples, 24, 1)

y\_pv shape: (n\_samples, 24, 1)

y\_both shape: (n\_samples, 24, 2)

**Dependencies**:

* numpy
* pandas
* os

**eval\_gru.py – Evaluation Script for GRU-Based Energy Forecasting Model**

**Purpose**:  
This script evaluates a trained GRU-based sequence-to-sequence model for multi-output prediction of building energy consumption and PV generation. It loads the model, runs predictions on a test dataset, computes evaluation metrics, and visualizes the results.

**Functionality Summary**:

* Loads saved test sequences from .npy files
* Loads a trained GRU model from disk
* Performs inference on the full dataset
* Computes evaluation metrics: RMSE, MAE, R², CVRMSE
* Generates prediction vs ground truth plots for both target variables

**Expected Inputs**:

|  |  |
| --- | --- |
| **File** | **Description** |
| data/sequences/X.npy | 3D array of input sequences (test data) |
| data/sequences/y\_both.npy | 3D array of ground truth target sequences |
| outputs/models/gru\_both.pt | Trained PyTorch GRU model checkpoint |

**Outputs**:

|  |  |
| --- | --- |
| **Output** | **Description** |
| Printed metrics | RMSE, MAE, R², CVRMSE |
| Line plots | For each target (Total Building kW, PV kW) |
| Saved images in outputs/plots/eval/ | PNG files showing predictions vs ground truth |

**Key Components**:

**compute\_metrics(y\_true, y\_pred)**

* Computes:
  + Root Mean Squared Error (RMSE)
  + Mean Absolute Error (MAE)
  + R-squared (R²)
  + Coefficient of Variation RMSE (CVRMSE)

**evaluate\_model()**

* Full pipeline:
  + Load data
  + Load model and transfer to device
  + Predict outputs using model(X)
  + Remove sequences containing NaNs
  + Compute and print metrics
  + Save visual plots for each output variable

**Device Usage**:

* Automatically uses GPU if available, otherwise falls back to CPU.

**Plot Example**:

* One plot for Total Building (kW)
* One plot for PV (kW)
* x-axis: Hour (prediction horizon)
* y-axis: Energy consumption / generation in kW

**Dependencies**:

* numpy
* matplotlib
* torch
* scikit-learn
* os, sys

**Code Summary**:

with torch.no\_grad():

predictions = model(X)

...

rmse, mae, r2, cvrmse = compute\_metrics(truths, predictions)

...

plt.plot(truths[0, :, i], label="Ground Truth")

plt.plot(predictions[0, :, i], label="Prediction")

**Example Evaluation Output**:

Evaluation Results:

RMSE: 13.5211

MAE: 9.1123

R²: 0.8724

CVRMSE: 0.1458

**eval\_lstm.py – Evaluation Script for LSTM-Based Building Energy Forecasting**

**Purpose**:  
This script evaluates a trained LSTM-based sequence-to-sequence model on preprocessed test data. It calculates standard regression metrics and visualizes the prediction vs. ground truth for total building energy consumption and PV power generation.

**Functionality Summary**:

* Loads saved sequences from .npy files (X.npy and y\_both.npy)
* Loads a trained LSTM model from outputs/models/lstm\_both.pt
* Predicts on the entire dataset using the trained model
* Computes error metrics: RMSE, MAE, R², CVRMSE
* Saves side-by-side prediction plots for a sample sequence

**Expected Inputs**:

|  |  |
| --- | --- |
| **Input File** | **Description** |
| X.npy | Input sequences (shape: samples × lookback × features) |
| y\_both.npy | Target sequences for Total Building and PV output |
| lstm\_both.pt | Trained LSTM model (PyTorch checkpoint) |

**Outputs**:

|  |  |
| --- | --- |
| **Output** | **Description** |
| Printed metrics | RMSE, MAE, R², CVRMSE |
| Saved plots | For both target variables (Total Building, PV) |
| Output directory | outputs/plots/eval/ |

**Key Functions**:

**compute\_metrics(y\_true, y\_pred)**

* Filters out NaNs from both prediction and truth values
* Calculates:
  + Root Mean Squared Error (RMSE)
  + Mean Absolute Error (MAE)
  + R-squared (R²)
  + Coefficient of Variation of RMSE (CVRMSE)

**evaluate\_model()**

* Loads the dataset and model
* Performs inference for each sample using teacher forcing = 0
* Aggregates predictions across all sequences
* Computes and prints evaluation metrics
* Visualizes a single sample prediction using line plots (defined by SAMPLE\_INDEX)

**Plotting Details**:

* One plot each for:
  + Total Building (kW)
  + PV (kW)
* X-axis: hour in the prediction horizon (e.g., 24 hours)
* Y-axis: scaled energy/power value
* Format: Ground truth (circles), prediction (crosses)

**Hyperparameters Used**:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| hidden\_dim | 64 |
| teacher\_forcing | 0 (in evaluation mode) |
| SAMPLE\_INDEX | 10 |

**Dependencies**:

* numpy, torch, matplotlib, scikit-learn, os, sys

**Example Output (printed)**:

Total predictions: 6720

NaNs in predictions: 0

NaNs in ground truths: 0

Evaluation Results:

RMSE: 10.2431

MAE: 7.5129

R²: 0.9041

CVRMSE: 0.1352

**Example Saved Plot Filenames**:

* prediction\_Total\_Building\_kW.png
* prediction\_PV\_kW.png

**preprocess.py – Data Preprocessing for Building Energy and Weather Time Series**

**Purpose**:  
This script prepares raw energy consumption and weather data for time series modeling by merging, cleaning, engineering features, and normalizing the dataset. It outputs a unified, scaled CSV file suitable for sequence generation.

**Functionality Summary**:

* Loads raw energy and weather data CSVs
* Merges datasets based on timestamp
* Removes irrelevant or missing data
* Extracts time-based features (hour, day of week)
* Normalizes all numeric features using Min-Max scaling
* Saves a cleaned and processed CSV file
* Plots a trend visualization of energy usage for sanity check

**Input Files**:

|  |  |
| --- | --- |
| **File Path** | **Description** |
| data/rsfmeasureddata2011.csv | Raw building energy consumption data |
| data/rsfweatherdata2011.csv | Raw environmental weather data |

**Output Files**:

|  |  |
| --- | --- |
| **Output File** | **Description** |
| data/processed\_dataset.csv | Cleaned and scaled dataset ready for use |
| outputs/plots/energy\_trend.png | Visualization of initial energy usage trend |

**Key Functions**:

**load\_and\_merge\_data()**

* Loads both CSV files into DataFrames
* Standardizes the DateTime column
* Drops irrelevant columns
* Merges datasets on the DateTime field

**clean\_and\_engineer(df)**

* Removes missing values
* Extracts new features:
  + hour: Hour of the day
  + dayofweek: Day of the week (0=Monday, 6=Sunday)
* Removes original DateTime column

**normalize\_data(df)**

* Applies MinMaxScaler to scale all numeric features between 0 and 1

**plot\_trend(df)**

* Saves a preview plot of energy usage (first 500 hours) for inspection
* Automatically detects whether Total Building (kW) or Building Net (kW) exists

**Workflow in main()**:

1. Load & merge energy + weather data
2. Clean and engineer new time features
3. Normalize numeric features
4. Save as processed\_dataset.csv
5. Generate a line plot of building energy usage

**Example Plot**:  
A line graph showing energy trends over the first 500 hourly records — helps visually validate preprocessing.

**Dependencies**:

* pandas
* numpy
* matplotlib
* sklearn.preprocessing.MinMaxScaler
* os

**Console Output Example**:

Energy shape: (8760, 12)

Weather shape: (8760, 7)

Merged data shape: (8750, 17)

After cleaning: (8600, 16)

After normalization: (8600, 16)

Saved processed data to: data/processed\_dataset.csv

Saved plot to: outputs/plots/energy\_trend.png

Preprocessing complete.

**train\_gru.py – Training Script for GRU-Based Encoder-Decoder Model**

**Purpose**:  
This script trains a sequence-to-sequence GRU model to predict both total building energy consumption and PV power output. It reads prepared .npy data, performs training over multiple epochs, and saves both the trained model and a loss curve plot.

**Functionality Summary**:

* Loads preprocessed input/output sequences
* Initializes and trains a Seq2SeqGRU model
* Uses Mean Squared Error (MSE) as the loss function
* Implements teacher forcing during training
* Monitors and handles NaNs in model outputs
* Saves the model and a loss curve image to outputs/

**Expected Inputs**:

|  |  |
| --- | --- |
| **File** | **Description** |
| data/sequences/X.npy | 3D array of input sequences |
| data/sequences/y\_both.npy | 3D array of target sequences (2 variables) |
| models/seq2seq\_gru.py | Contains definition for the GRU model |

**Outputs**:

|  |  |
| --- | --- |
| **Output File** | **Description** |
| outputs/models/gru\_both.pt | Trained PyTorch GRU model weights |
| outputs/plots/loss\_gru\_both.png | Plot of training loss across epochs |

**Key Hyperparameters**:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| LOOKBACK | 168 (1 week) |
| HORIZON | 24 (1 day) |
| BATCH\_SIZE | 64 |
| EPOCHS | 20 |
| LEARNING\_RATE | 0.001 |
| TEACHER\_FORCING\_RATIO | 0.5 |

**Main Components**:

**load\_data()**

* Loads X.npy and y\_both.npy
* Converts them to PyTorch tensors
* Wraps them in a TensorDataset and returns a DataLoader
* Also returns input/output dimensions for model instantiation

**train()**

* Initializes the GRU model, optimizer, and MSE loss function
* Loops over data batches for a specified number of epochs
* Applies teacher forcing (50% chance to use ground truth)
* Monitors for NaN values and skips unstable batches
* Saves model if no NaNs are present in sample output
* Plots training loss over time

**Handling Instability**:

* Skips training batches that produce NaN outputs or loss
* Performs a sanity check on final model output before saving

**Visualization**:

* Loss plot is saved as loss\_gru\_both.png
* Helps visually confirm convergence and training dynamics

**Dependencies**:

* torch, numpy, matplotlib
* models/seq2seq\_gru.py
* sklearn (indirectly, if metrics or scaling is used elsewhere)

**Console Output Example**:

Training started...

Epoch 1/20 - Loss: 0.015612

Epoch 2/20 - Loss: 0.011348

...

Model saved to: outputs/models/gru\_both.pt

Loss plot saved to: outputs/plots/loss\_gru\_both.png

**train\_lstm.py – Training Script for LSTM-Based Encoder-Decoder Model**

**Purpose**:  
This script trains a sequence-to-sequence LSTM model to forecast both total building energy consumption and PV generation. It loads preprocessed training sequences, trains the model using teacher forcing, and saves both the trained model weights and a loss plot.

**Functionality Summary**:

* Loads input-output time series data from .npy files
* Instantiates the LSTM encoder-decoder model
* Trains the model over multiple epochs using MSE loss
* Uses teacher forcing with a 50% ratio
* Skips unstable batches that produce NaNs
* Saves the model checkpoint and a plot of training loss

**Expected Inputs**:

|  |  |
| --- | --- |
| **File** | **Description** |
| data/sequences/X.npy | 3D array of input sequences |
| data/sequences/y\_both.npy | 3D array of target sequences (2 outputs) |
| models/seq2seq\_lstm.py | Python file containing model class |

**Outputs**:

|  |  |
| --- | --- |
| **File Path** | **Description** |
| outputs/models/lstm\_both.pt | Trained PyTorch LSTM model weights |
| outputs/plots/loss\_lstm\_both.png | Plot of MSE training loss over epochs |

**Key Training Parameters**:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| LOOKBACK | 168 (hours) |
| HORIZON | 24 (hours) |
| BATCH\_SIZE | 64 |
| EPOCHS | 20 |
| LEARNING\_RATE | 1e-3 |
| TEACHER\_FORCING\_RATIO | 0.5 |

**Core Components**:

**load\_data()**

* Loads NumPy arrays from disk
* Converts them into PyTorch tensors
* Wraps the tensors in a TensorDataset and returns a DataLoader
* Extracts input\_dim and output\_dim based on tensor shape

**train()**

* Initializes the model, loss function (nn.MSELoss), and optimizer (Adam)
* Iterates over all batches for each epoch
* Applies teacher forcing while training
* Skips training steps if NaNs are detected in outputs or loss
* After training, runs a sample forward pass to check for NaNs
* Saves model only if the check passes
* Saves training loss history as a line plot

**Error Handling**:

* NaN in model output or loss → batch is skipped
* Final NaN check before saving model to disk

**Loss Plot**:

* Y-axis: MSE loss
* X-axis: Epoch
* Title: "LSTM Training Loss"

**Dependencies**:

* numpy
* torch
* matplotlib
* torch.nn
* torch.utils.data
* models/seq2seq\_lstm.py

**Example Output Log**:

Training started...

Epoch 1/20 - Loss: 0.017456

Epoch 2/20 - Loss: 0.013801

...

Model saved to: outputs/models/lstm\_both.pt

Loss plot saved to: outputs/plots/loss\_lstm\_both.png

**seq2seq\_gru.py – GRU-Based Sequence-to-Sequence Model for Time Series Forecasting**

**Purpose:**  
This file defines a GRU-based encoder-decoder architecture implemented in PyTorch. The model is designed for multistep time series forecasting, particularly for tasks such as predicting building energy consumption and PV power generation based on historical time series data.

**Class: Seq2SeqGRU**

class Seq2SeqGRU(nn.Module):

This class implements a sequence-to-sequence model using Gated Recurrent Units (GRUs) for both the encoder and decoder. It allows optional use of teacher forcing during training and supports multi-feature input and output.

**Constructor**

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, num\_layers=2):

**Arguments:**

* input\_dim (int): The number of features in each input time step.
* hidden\_dim (int): The dimensionality of the GRU's hidden state.
* output\_dim (int): The number of output variables predicted at each time step.
* num\_layers (int): The number of GRU layers in both the encoder and decoder. Default is 2.

**Attributes:**

* self.encoder: A multi-layer GRU used to encode the input sequence.
* self.decoder: A multi-layer GRU used to decode and predict future steps.
* self.fc: A fully connected linear layer to map GRU outputs to the desired output dimension.

**Method: forward**

def forward(self, x, target=None, teacher\_forcing\_ratio=0.5):

**Arguments:**

* x (Tensor): The input sequence of shape (batch\_size, lookback, input\_dim).
* target (Tensor, optional): The ground truth output sequence of shape (batch\_size, horizon, output\_dim). Used for teacher forcing.
* teacher\_forcing\_ratio (float): The probability of using the true value from target as the next decoder input during training.

**Returns:**

* A tensor of shape (batch\_size, horizon, output\_dim) representing the predicted output sequence.

**Model Behavior**

* The input sequence is first encoded using the encoder GRU, which returns the final hidden state.
* The decoder GRU then generates predictions step-by-step:
  + At the first step, the decoder input is initialized to zeros.
  + For subsequent steps, the decoder input is either the previous ground truth (if using teacher forcing) or the previous predicted value.
* All predictions are collected and concatenated along the time axis to produce the final output sequence.

**Example Tensor Shapes**

|  |  |
| --- | --- |
| **Component** | **Shape** |
| Input x | (batch\_size, lookback, input\_dim) |
| Target target | (batch\_size, horizon, output\_dim) |
| Output | (batch\_size, horizon, output\_dim) |

**seq2seq\_lstm.py – LSTM-Based Sequence-to-Sequence Model for Multistep Forecasting**

**Purpose:**  
This module implements a sequence-to-sequence architecture using Long Short-Term Memory (LSTM) networks in PyTorch. It is designed for multistep time series forecasting tasks, such as predicting future energy consumption or PV generation based on past measurements.

**Class: Seq2SeqLSTM**

class Seq2SeqLSTM(nn.Module):

This class defines an encoder-decoder model based on LSTM units. It supports multi-feature input/output, sequence forecasting over a fixed horizon, and optional teacher forcing during training.

**Constructor**

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, num\_layers=2, dropout=0.2):

**Arguments:**

* input\_dim (int): Number of input features per time step (e.g., weather variables, historical energy usage).
* hidden\_dim (int): Number of hidden units in each LSTM layer.
* output\_dim (int): Number of output variables to predict per time step.
* num\_layers (int): Number of LSTM layers in both encoder and decoder. Default is 2.
* dropout (float): Dropout rate applied between LSTM layers. Default is 0.2.

**Attributes:**

* self.encoder: A stacked LSTM that encodes the input sequence into a context vector (hidden and cell states).
* self.decoder: A stacked LSTM that generates the output sequence from the context vector.
* self.fc: A linear layer that maps the decoder’s output at each time step to the final prediction.

**Method: forward**

def forward(self, x, y\_init=None, teacher\_forcing\_ratio=0.5):

**Arguments:**

* x (Tensor): Input tensor of shape (batch\_size, lookback, input\_dim).
* y\_init (Tensor, optional): Ground truth sequence for use during teacher forcing. Shape: (batch\_size, horizon, output\_dim).
* teacher\_forcing\_ratio (float): Probability of using ground truth instead of model prediction for the next decoder input.

**Returns:**

* A tensor of shape (batch\_size, horizon, output\_dim) representing the model's predicted future sequence.

**Model Behavior**

1. The encoder LSTM processes the input sequence x, returning the final hidden and cell states.
2. These states initialize the decoder LSTM, which predicts the output sequence one time step at a time.
3. The decoder starts with either a zero tensor or the first time step from y\_init.
4. For each future time step:
   * With probability equal to teacher\_forcing\_ratio, the ground truth from y\_init is used as input.
   * Otherwise, the model’s own previous prediction is used as input.
5. All outputs are collected and concatenated to form the full predicted sequence.

**Example Tensor Shapes**

|  |  |
| --- | --- |
| **Component** | **Shape** |
| Input x | (batch\_size, lookback, input\_dim) |
| Target y\_init | (batch\_size, horizon, output\_dim) |
| Output | (batch\_size, horizon, output\_dim) |