# Documentation – AI

## Overview of the Application

This application is designed to predict energy demand using historical data and machine learning techniques. It includes several Python scripts for data preprocessing, model training, prediction, and visualization. The application aims to provide accurate forecasts of energy consumption patterns based on past trends, which can be useful for energy analysts, utility companies, and researchers.  
  
Each script in this application has a specific role, from combining raw data files to generating plots that compare actual vs. predicted energy demands.

## 1. xconcatenator.py

**Scope:**  
The `xconcatenator.py` script is responsible for combining multiple CSV files located in a specified folder into a single CSV file.   
This operation is critical for preparing a comprehensive dataset that can be used for training machine learning models or for further analysis.  
  
**How it Works:**  
1. \*\*Imports\*\*: The script utilizes `os` to handle file operations and `pandas` to manage data manipulation.

2. \*\*File Listing\*\*: It identifies all `.csv` files within a given folder (`folder\_path`). It uses list comprehension and the `os.listdir()` method to achieve this.

3. \*\*Sorting Files\*\*: The files are sorted chronologically based on the first four characters of their filenames, assuming they represent the year (e.g., '2020\_data.csv'). This is done using Python's built-in `sort()` method.

4. \*\*Reading and Concatenating\*\*: The script iterates through each CSV file, reads it into a `pandas DataFrame`, and appends it to a list. All DataFrames are then concatenated into a single DataFrame using `pd.concat()`.

5. \*\*Output\*\*: Finally, the concatenated DataFrame is saved as a new CSV file (`output\_file`).

**Key Function:**   
```python  
def concatenate\_csv\_files(folder\_path, output\_file):  
 ...  
```  
- Purpose: Combines all CSV files in a folder into one.  
- Parameters:  
 - `folder\_path`: The directory path containing CSV files.  
 - `output\_file`: The name of the output file where the combined data is saved.  
- Process:  
 - Retrieves CSV filenames, sorts them by year, reads each file, and concatenates the data.  
  
**Usage Notes:**  
- Default `folder\_path` is set to `'data\_full/raw/'`.  
- Default `output\_file` is `'concatenated\_years.csv'`.  
- Modify these paths as needed before running the script.

## 2. xplotter.py

**Scope:**  
The `xplotter.py` script is designed to generate visual comparisons between two datasets, specifically to compare actual vs. predicted energy loads.  
This script is particularly useful for evaluating the performance of the prediction model.  
  
**How it Works:**  
1. \*\*Imports\*\*: Utilizes `pandas` for data handling and `matplotlib` for plotting graphs.  
2. \*\*File Reading\*\*: Reads two CSV files (`xdata.csv` and `predicted\_energy\_demand\_2.csv`) using `pd.read\_csv()`.  
3. \*\*Column Selection\*\*: Selects specific columns from each dataset for comparison. The columns chosen are `'Biomass - Actual Aggregated [MW]'` from the first file and `'Predicted Load [MW]'` from the second file. Any column from the dataset can be chosen.  
4. \*\*Plotting\*\*: Uses `plt.plot()` to generate an overlay plot that displays both data columns on the same graph. This visual representation helps in comparing the actual and predicted values side-by-side.  
  
**Plotting process:**  
- Configures plot labels and titles using `plt` methods.  
- Displays the plot with `plt.show()` to provide a visual assessment.  
  
Ensure the input files (`xdata.csv` and `predicted\_energy\_demand\_2.csv`) are in the same directory or specify their paths accordingly.

## 3. xpredictor.py

**Scope:**  
The `xpredictor.py` script is the main module for generating energy demand predictions using a pre-trained machine learning model.   
It loads the model, takes input data (like dates), and outputs predictions based on that data.  
  
**How it Works:**  
1. \*\*Imports\*\*:   
 - `pandas` and `numpy` for data handling.  
 - `joblib` for loading the pre-trained model.  
 - `datetime` for handling date inputs and ranges.  
2. \*\*Model Loading\*\*: Loads a pre-trained model (`model\_biomass.pkl`) using `joblib.load()`.  
3. \*\*Date Input\*\*: Takes user input for the end date of the prediction range. Uses `datetime.strptime()` to convert the input string into a date object.  
4. \*\*Date Range Generation\*\*: Creates a range of dates from the current date to the user-specified end date, with a frequency of 15 minutes using `pd.date\_range()`.  
5. \*\*Data Preparation\*\*: Constructs a DataFrame (`data`) with columns for year, month, day, hour, and minute.  
6. \*\*Prediction\*\*: Uses the loaded model to make predictions on the generated data range.  
7. \*\*Output\*\*: Saves the predictions to a CSV file (`prediction.csv`).  
  
**Key Functionality:**  
- \*\*Loading the Model\*\*: Ensures the model file is correctly loaded and ready to make predictions.  
- \*\*Generating Predictions\*\*: Prepares the input data in the format expected by the model and generates predictions.  
  
Follow the prompts to enter the desired end date for prediction.

## 4. xtrainer.py

**Scope:**  
The `xtrainer.py` script is responsible for training the machine learning model on historical energy consumption data.  
It handles the data preprocessing, feature engineering, model training, and saving of the trained model.  
  
**How it Works:**  
1. \*\*Data Loading\*\*: Reads the training data from `concatenated\_years.csv`.  
2. \*\*Feature Engineering\*\*: Processes raw input data to create features relevant for the prediction model.  
3. \*\*Model Training\*\*: Utilizes a machine learning algorithm (e.g., Random Forest, Linear Regression) to train a model.  
4. \*\*Model Saving\*\*: Saves the trained model to a file (e.g `model\_biomass.pkl`) for use in predictions.  
  
**Key Functions:**  
- \*\*Feature Preparation\*\*: Transforms the raw data into a format suitable for model training.  
- \*\*Training Logic\*\*: Applies machine learning algorithms to learn patterns from the data.  
  
Ensure the required data file (`concatenated\_years.csv`) is available in the directory.

## 5. concatenated\_years.csv

**Scope:**  
`concatenated\_years.csv` is the combined dataset generated by the `xconcatenator.py` script.   
It consolidates multiple raw data files into a single file, making it easier to train models or conduct analyses.  
  
**Structure:**  
- The file contains multiple columns representing different features of energy consumption data, like date, time, energy type, and load.  
- Each row represents a time-stamped entry of energy consumption.

**Usage:**  
- This file is used as input for training models in `xtrainer.py`.

## 6. data\_full Directory

**Scope:**  
The `data\_full` directory contains all the raw data files that the application uses for model training and evaluation.  
  
**Contents:**  
- The directory typically contains CSV files, each representing data for different time periods (e.g., monthly or yearly data).  
- The files are read and concatenated by `xconcatenator.py` to form a comprehensive dataset.

## 7. Requirements

* torch
* pandas
* transformers
* scikit.learn
* joblib
* matplotlib
* numpy
* tqdm
* re
* datetime