Appliances Energy Prediction

Project Summary

This project develops and evaluates machine learning models to predict household appliance energy consumption (Wh) using the UCI "Appliances Energy Prediction" dataset. The analysis includes extensive feature engineering, model training and hyperparameter tuning, model comparison, and a lightweight model-serving prototype (Flask).

Key outcome: ensemble tree models performed best — the Extra Trees model achieved a test RMSE ≈ 63.07 and R^2 ≈ 0.60 (reported in the notebook analyses).

Problem Statement

Predict short-term appliance energy consumption in a single-family residential building using environmental (indoor/outdoor) measurements and engineered temporal features. Accurate predictions can support demand-side management, energy-efficiency interventions, and intelligent control systems.

Dataset

- Source: UCI Machine Learning Repository — "Appliances Energy Prediction" dataset.
- Instances: ~19,735 measurements taken every 10 minutes.
- Key variables: `Appliances` (target, Wh), room temperatures `T1..T9`, humidities `RH_*`, outdoor weather features (`T_out`, `RH_out`, `Windspeed`, `Visibility`, `Press_mm_hg`, `Tdewpoint`), `lights`, and two random variables `rv1`, `rv2`.

Data is stored in `data/energydata_complete.csv` and is preprocessed, feature-engineered and split in `data_analysis.ipynb`.

Variables and their Description

Variable	Description
date	Time (year-month-day hour:minute)
Appliances	Energy use in Wh
lights	Energy use of light fixtures in the house in Wh
T1	Temperature in kitchen area, in Celsius
RH_1	Humidity in kitchen area, in %
T2	Temperature in living room area, in Celsius
RH_2	Humidity in living room area, in %
T3	Temperature in laundry room area, in Celsius
RH_3	Humidity in laundry room area, in %

Variable	Description
T4	Temperature in office room, in Celsius
RH_4	Humidity in office room, in %
T5	Temperature in bathroom, in Celsius
RH_5	Humidity in bathroom, in %
T6	Temperature outside the building (north side), in Celsius
RH_6	Humidity outside the building (north side), in %
T7	Temperature in ironing room, in Celsius
RH_7	Humidity in ironing room, in %
T8	Temperature in teenager room 2, in Celsius
RH_8	Humidity in teenager room 2, in %
T9	Temperature in parents room, in Celsius
RH_9	Humidity in parents room, in %
To	Temperature outside (from Chièvres weather station), in Celsius
Pressure	Pressure (from Chièvres weather station), in mm Hg
RH_out	Humidity outside (from Chièvres weather station), in %
Windspeed	Windspeed (from Chièvres weather station), in m/s
Visibility	Visibility (from Chièvres weather station), in km
Tdewpoint	Tdewpoint (from Chièvres weather station), °C
rv1	Random variable 1, nondimensional
rv2	Random variable 2, nondimensional

Feature Engineering & EDA (brief)

- Temporal features extracted from the timestamp: seconds-from-midnight (NSM), hour, day, month, day-of-week, week status (Weekend/Weekday), and cyclic encodings for hour (`Hour_sin`, `Hour_cos`).
- Seasonal categorical variable derived from month (Winter, Spring, Summer, Autumn).
- Basic exploratory plots produced: pairplots, time series profiles, histograms, boxplots, and heatmaps to inspect hourly/weekly consumption patterns.
- Correlation heatmap computed for numeric columns to inspect multicollinearity.

Modelling Approach

Models trained and compared:

- Multiple Linear Regression (LM)
- Logistic Regression (Logit) — included as baseline (kept for experimentation)

- Decision Tree Regressor (DT)
- Random Forest Regressor (RF)
- Extra Trees Regressor (ET)
- Gradient Boosting Regressor (GB)
- XGBoost (XGB)
- Support Vector Regression (SVR)

Model selection & tuning

- Data splits: training / validation / test splits created in the notebook.
- Preprocessing: one-hot encoding of categorical variables (day-of-week, season etc.) and `StandardScaler` for numeric features.
- Hyperparameter tuning performed using `HalvingRandomSearchCV` / `HalvingGridSearchCV` (efficient halving search) for tree and boosting models.
- Evaluation via cross-validation and hold-out test set.

Evaluation metrics

- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)
- R-squared (R^2)

Key result (notebook)

- The Extra Trees model (ET) performed best overall on these experiments with test RMSE ≈ 63.07 and R^2 ≈ 0.60. Random Forest was close behind.

Project Files (important)

- `data_analysis.ipynb` — full analysis, EDA, feature engineering, training, hyperparameter tuning and notebook cells that save artifacts.
- `data/energydata_complete.csv` — original dataset.
- `new_data/` — CSVs with training/validation/test splits and saved arrays created by the notebook (`X_train.csv`, `y_train.csv`, etc.).
- `models/` — serialized model artifacts saved by the notebook (e.g. `models/best_et.pkl`).
- `et_model.bin` — a bundled pickle file (created by the notebook) containing `{'model': estimator, 'scaler': scaler, 'features': features}` used by the serving app.
- `app.py` — simple Flask app that loads `et_model.bin` and serves predictions via `POST /predict` and a minimal web UI.
- `predict.py` — small CLI helper to run a single-sample prediction using `et_model.bin`.
- `requirements.txt` — Python dependency list.

Running the Flask app and using the CLI predictor

This document explains how to run the Flask web app (`app.py`) and how to call the CLI `predict.py` provided in this folder.

Prerequisites

- Python 3.8+ installed
- Recommended: create a virtual environment and install required packages.

Quick setup (PowerShell)

```
# Go to the Flask app folder
cd "\midterm\Flask-App"

# Create and activate a virtual environment (PowerShell)
python -m venv .venv
.\.venv\Scripts\Activate.ps1

# Install dependencies
pip install --upgrade pip
pip install -r requirements.txt
```

Place the trained model

- The app expects the model file `extra_trees_energy_model.bin` to live in the same folder as `app.py`.
- After training (or when you obtain the trained bundle), copy it here:
- Alternatively, running the `train.py` will create the model for you.

```
copy "..\extra_trees_energy_model.bin" .
# or move the file into this folder in Explorer
```

Run the Flask app (local)

```
# Start the app (development server)
python app.py
# The app will start on http://0.0.0.0:5000 (open http://localhost:5000 in the
browser)
```

Use the web UI

- Open `http://localhost:5000` in your browser. The page shows a preview of model features and a JSON form to send a single prediction.

Call the API while the app is running at the localhost. Use any of these:

- Single sample (JSON object of feature:value pairs):

```
curl -X POST -H "Content-Type: application/json" -d "{\"T1\": 20, \"RH_1\": 40,
\"Hour_sin\": 0.0, ... }" http://localhost:5000/predict
```

- Use this if CURL fails:

```
$body = @{
    T1 = 20
    RH_1 = 40
} | ConvertTo-Json

Invoke-RestMethod -Uri "http://127.0.0.1:5000/predict" -Method POST -Body $body -
ContentType "application/json"
```

- The response is JSON: `{ "predictions": [ <value> ], "n_features": <int> }`.

Using `predict.py` from the command line

- `predict.py` supports single-dict prediction and batch CSV prediction. Example usages from this folder:

Single JSON (print result):

```
python predict.py -d "{\"T1\":20.5, \"RH_1\":30, \"Hour_sin\":0.0, ... }"
```

Single JSON (file) (output printed):

```
python predict.py -df path\to\my_inputs.json
```

Batch CSV (output printed):

```
python predict.py -b path\to\my_inputs.csv
```

Notes and troubleshooting

- The feature names used by the model are recorded inside the model bundle under the `features` key. Ensure any input JSON or CSV includes these columns (or at least the keys) — missing features will be filled with zeros by the app.
- If you changed the model filename or structure, update `MODEL_PATH` inside `app.py` or place the model file in this directory with the expected name.
- For production use, run the Flask app behind a WSGI server (Gunicorn, Waitress) and disable `debug=True`.

Enjoy!

Docker (example)

Create a `Dockerfile` like the following and build the image. This example is a recommended convenience; a `Dockerfile` is not included by default in this folder but may be added for submission.

```
FROM python:3.11-slim
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

Build & run (local):

```
docker build -t appliances-energy-app .
docker run -p 5000:5000 appliances-energy-app
```

Limitations & Next Steps

- Dataset scope: the UCI dataset is from a single household/location and may not generalise to other homes or climates.
- Model bundle: the current serving approach pickles a model+scaler+feature list. For production, prefer a reproducible pipeline (e.g. sklearn Pipeline or joblib) and avoid insecure pickle loading for untrusted inputs.
- Preprocessing gaps: the server assumes one-hot encoding and numeric inputs; it fills missing features with zeros. A more robust preprocessor that replicates notebook encoding (categories, missing-value handling) would improve reliability.
- Deployment: add integration tests, CI checks, a Dockerfile in the repo, and optionally deploy to a cloud platform.
- Monitoring & fairness: add drift detection and monitoring for model degradation over time.

Reproducibility

To reproduce the results, run `data_analysis.ipynb` end-to-end (or refactor to `train.py`) to re-create splits, train models and save `et_model.bin`. The notebook contains the exact preprocessing and hyperparameter search steps used in the experiments.

References

Include paper and dataset references in your final report. Use citation placeholders like (Author, Year) if needed.