Extended Research Project Technical Report

TOWARDS EQUITABLE ENERGY:

TEMPORAL LOAD MODELLING FOR INCLUSIVE DEMAND-RESPONSE

An extended research project technical report submitted to the University of Manchester

for the degree of MSc Data Science in the Faculty of Humanities

Student ID: 11496500

Date of Submission: 2025

School of Social Sciences

ETHICS FORM

MSc Data Science Extended Research Project Ethics Review	
Student name	Akanksha Joshi
Student number	11469500
Project Title	Smart energy pricing and smart meter data analytics
Supervisor	Dr. Fanlin Meng
Please provide information about the ethical status of your project	
Describe the data you will be using. What is it (format, type, data units), where does it originate from, and where will you obtain it from: I will use publicly available smart meter datasets, such as the Low Carbon London dataset or the Commission for Energy Regulation (CER) Smart Metering Project dataset. The data is provided in CSV format and includes half-hourly or minute-level electricity consumption records (kWh) for individual households, along with metadata on tariff structures and customer demographic surveys. These datasets originate from large-scale research trials conducted in the UK and Ireland, focusing on household electricity usage, and can be accessed via the UK Data Service (https://beta.ukdataservice.ac.uk/datacatalogue/studies/study?id=7857) or the Irish Social Science Data Archive (https://www.ucd.ie/issda/data/commissionforenergyregulationcer/). Explain why it does or does not meet the three criteria (consent, anonymous information, permission of the data controller): • **Consent:** Datasets were collected under projects where participants consented to their data being used for research purposes and are publicly available datasets. • **Anonymous Information:** The data is fully anonymized, with no personally identifiable information included. Demographic data is aggregated or coded to ensure privacy. • **Permission of the data controller:** The datasets are published for research use with explicit permission and under licensing agreements that specify appropriate data use and citation. Describe any actions will you take to ensure ethical compliance and/or manage risk: I will comply with the data usage licenses and follow institutional research ethics guidelines. No attempts will be made to re-identify individuals or households. The analysis will be conducted on anonymized data, and results will be reported in aggregate form only. All data will be securely stored on encrypted and access-controlled devices, and data sharing will not occur outside authorized university platforms.	
Decision (to be completed by programme director)	
No Further ethical review is needed	X
A further programme review is needed	
School Review is needed	
Proportionate UREC review is needed	

Signatures	
Student	Akanksha Joshi
Supervisor	
Programme Director	Mark Elliot

Screenshot of ethics decision tool outcome:

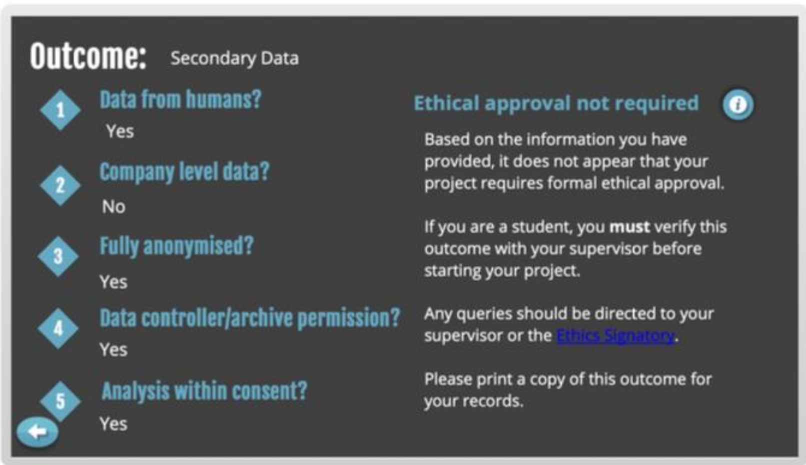

Table of Contents

GUIDE TO CODE REPOSITORY

CODE AVAILABILITY. **THE COMPLETE PROJECT CODEBASE IS AVAILABLE ON GITHUB: BM-11450069-CODE.**

DATA AVAILABILITY. **THE ANALYSES RELY ON THE** SMART ENERGY RESEARCH LAB (SERL) STATISTICAL DATA, 2020–2023**DISTRIBUTED BY THE UK DATA SERVICE (**STUDY 8963, SAFEGUARDED**). ACCESS VIA DOI: 10.5255/UKDA-SN-8963-2.**

1 INTRODUCTION & OVERVIEW

The dissertation investigates temporal patterns and equity implications of residential electricity use in Great Britain (2020–2023). Using aggregated smart-meter statistics, the study combines descriptive profiling, unsupervised clustering, time-series forecasting, and tariff simulation to examine how demand structures evolve and how pricing designs may differentially affect household groups.

The dataset is released under safeguarded access with curation by the UK Data Service (consistency checks, metadata enhancement, DOI assignment) and a second edition (May 2024) including daily and four half-hourly aggregated files for 2020–2023 plus documentation. These conditions frame the analytical design and reporting.

1.1 Purpose of this Document

This document serves as a reproducible technical contract for the dissertation. It

(a) summarises aims and research questions,

(b) formalises the reproducibility plan and ethical constraints attached to safeguarded data, and

(c) specifies the exact software and hardware environment required to regenerate all results. References to the source collection, including **Study 8963 (DOI: 10.5255/UKDA-SN-8963-2)**, are provided to ensure unambiguous lineage.

1.2 Project Summary & Research Objectives

1.2.1 Summary

The project analyses aggregated, licence-controlled SERL statistics (daily and half-hourly) to characterise load structure, discover stable demand regimes, evaluate forecastability with weather covariates, and quantify cost exposure under dynamic tariffs. The methodological components and validation approach including back-testing, clustering diagnostics, and tariff-API safeguards are defined in the dissertation manuscript.

1.2.2 Research Objectives (operational)

O1 Descriptive and comparative profiling. Quantify diurnal/weekly/seasonal patterns and subgroup contrasts (e.g., EPC band, IMD, heating type, PV, region) using distributional metrics (means, variance, skew/kurtosis, load factor, evening-peak share).

O2 Pattern discovery and archetypes. Engineer load-shape features and apply UMAP + HDBSCAN to identify demand regimes; validate with density-based validity indices and noise proportion; interpret prevalence against subgroup attributes.

O3 Forecasting and tariff-sensitivity evaluation. Train SARIMA/SARIMAX baselines with meteorological inputs against seasonal-naïve controls; simulate conservation and shock scenarios; integrate half-hourly Agile prices via the Octopus Energy API to compute costs, bill deltas, and equity indicators.

Empirical context and access. The analysis relies on the curated SERL release (daily and half-hourly aggregates, 2nd edition, May 2024) under the UK Data Service End User Licence; file inventory and components are listed by the Archive. Data citation: Study 8963, DOI: 10.5255/UKDA-SN-8963-2.

1.3 Statement of Reproducibility

Data access and licensing. Study 8963 is accessed under the UK Data Service End User Licence; commercial use requires additional approval. The collection landing materials specify the required citation and attribution (DOI: 10.5255/UKDA-SN-8963-2).

Curation baseline. Before release, the UK Data Service performs disclosure review, accuracy/consistency checks, metadata enhancement, and DOI assignment; the second edition (May 2024) adds daily and four half-hourly aggregated files plus a technical document. These steps provide provenance and quality control for replication.

Workflow determinism. The pipeline records configuration, package manifest, timestamps, and dataset edition string; tariff integration uses request retries, timeouts, local caching, response-schema validation, and rate-limit compliance to keep runs auditable and robust to transient network issues.

Ethics and disclosure. Only non-disclosive aggregates are analysed and reported, in line with safeguarded-access conditions and institutional ethics requirements.

Minimal steps to reproduce.

1. Obtain the SERL dataset under safeguarded access and confirm Study 8963, 2nd edition (May 2024), DOI: 10.5255/UKDA-SN-8963-2.
2. Place files as per the Archive's inventory (daily and four half-hourly aggregates plus documentation).

3. Create the environment specified in §1.4 and export a lockfile.
4. Run the pipeline modules in sequence: ingest/QC → profiling → clustering → forecasting → tariff simulation → validation and logs.

1.4 Prerequisites & Software Environment

1.4.1 Core Programming Language & Version

- **Python: 3.10+** (validated on 3.10; newer minor versions may be used subject to dependency support and lockfile updates).
- **Time/encoding:** UTC; UTF-8.
- **Notebook layer:** JupyterLab for diagnostics; notebooks treated as read-only artefacts within the pipeline.

1.4.2 Package Management File

Use **one** canonical manifest (pin versions; export a lockfile after creation).

Option A — `requirements.txt` (pip/venv)

```
python==3.10.*
numpy==1.26.4
pandas==2.2.2
scikit-learn==1.5.1
statsmodels==0.14.2           # SARIMA/SARIMAX
umap-learn==0.5.6             # Embeddings
hdbscan==0.8.37               # Clustering
matplotlib==3.9.0
plotly==5.22.0
bokeh==3.5.0
ipywidgets==8.1.3
requests==2.32.3              # Tariff API client
pyyaml==6.0.2
tqdm==4.66.4
pytest==8.2.0
```

Option B — `environment.yml` (conda/mamba)

```
name: ser18963
channels: [conda-forge]
dependencies:
  - python=3.10
  - numpy=1.26.4
  - pandas=2.2.2
  - scikit-learn=1.5.1
  - statsmodels=0.14.2
  - umap-learn=0.5.6
  - hdbscan=0.8.37
  - matplotlib=3.9.0
  - plotly=5.22.0
  - bokeh=3.5.0
  - ipywidgets=8.1.3
  - requests=2.32.3
  - pyyaml=6.0.2
  - tqdm=4.66.4
  - pytest=8.2.0
Locking: pip freeze > requirements.lock or conda env export --from-history > environment.lock.yml.
```

1.4.3 Critical Package Versions and Roles

Component	*Package & Version*	*Role in analysis*
Arrays/dataframes	`numpy 1.26.4`, `pandas 2.2.2`	Core data handling for daily and half-hourly aggregates
Forecasting	`statsmodels 0.14.2`	SARIMA/SARIMAX estimation and diagnostics
Feature learning	`umap-learn 0.5.6`	Low-dimensional embeddings of load-shape features
Clustering	`hdbscan 0.8.37`	Density-based regime discovery
Visualisation	`matplotlib 3.9.0`, `plotly 5.22.0`, `bokeh 3.5.0`, `ipywidgets 8.1.3`	Static and interactive figures; scenario controls
ML utilities	`scikit-learn 1.5.1`	Feature engineering, metrics, cross-validation helpers
API integration	`requests 2.32.3`	Retrieval of half-hourly Agile rates; retries, timeouts, caching hooks

Tariff-module requirements (API parameters, caching, schema validation, rate-limit compliance) are documented in the manuscript's methodology and results sections.

1.4.4 Hardware Considerations

1. **CPU/RAM:** ≥8 physical cores; ≥32 GB RAM for feature engineering, clustering diagnostics, and cross-validation.
2. **Storage:** ≥50 GB SSD for inputs, intermediates, caches, and artefacts; aligns with the Archive's multi-file structure.
3. **GPU:** Not required for SARIMA, UMAP, or HDBSCAN; optional for exploratory extensions beyond the validated workflow.
4. **Networking:** Outbound HTTPS required for tariff retrieval; resilience mechanisms (retries, caching, schema validation, logged parameters) are integral to reproducibility.

2 DATA DESCRIPTION & PROVENANCE

2.1 Primary Dataset(s)

2.1.1 Dataset 1: SERL Half-Hourly Energy Use in GB Domestic Buildings, 2020–2023 — Aggregated Statistics (four files; identical schema)

- Source: UK Data Service, Study 8963 — DOI: 10.5255/UKDA-SN-8963-2.
- Access Method: Register with the UK Data Service, accept the EUL, and download the half-hourly aggregated files listed for the 2nd release (May 2024).
- License: UK Data Service End User Licence (safeguarded).
- Raw File Name(s):

```
serl_half_hourly_energy_use_in_gb_domestic_buildings_2020_aggregated_statistics.csv

serl_half_hourly_energy_use_in_gb_domestic_buildings_2021_aggregated_statistics.csv

serl_half_hourly_energy_use_in_gb_domestic_buildings_2022_aggregated_statistics.csv

serl_half_hourly_energy_use_in_gb_domestic_buildings_2023_aggregated_statistics.csv
```

Note. These four annual files share the same variables and structure: only the **year** changes.

2.1.2 Dataset 2: SERL Daily Energy Use in GB Domestic Buildings, 2020–2023 — Aggregated Statistics (one file; cross-year aggregate)

1. Source: UK Data Service, Study 8963 — DOI: 10.5255/UKDA-SN-8963-2.
2. Access Method: As above; download the daily aggregated file listed for the 2nd release (May 2024).
3. License: UK Data Service End User Licence (safeguarded).
4. Raw File Name(s):
5. `serl_daily_energy_use_in_gb_domestic_buildings_2020_to_2023_aggregated_statistics.csv`

2.2 Data Dictionary

The half-hourly and daily files have the *same column schema*. Units differ by temporal resolution (Wh vs kWh/day), and some weather fields may be `NA` in half-hourly aggregates. Field descriptions align with Archive documentation and the dissertation's data section.

Variable Name	*Raw Data Type*	*Final Data Type (analysis)*	*Description*	*Handling Notes*
quantity	object	string (category)	Energy vector: `Gas`, `Electricity imports`, `Electricity exports`, `Electricity net`, `Gas + electricity imports`.	Validate allowed set.
unit	object	string (category)	Half-hourly: `Wh`, `Wh/m2`, `Wh/person`. Daily: `kWh/day`, `kWh/m2/day`, `kWh/person/day`.	Use as-provided; do not coerce across files.
summary_time	object	string	Half-hourly: `HH:MM`. Daily: literal `Daily`.	Parse to `datetime.time` for HH files if needed.
aggregation_period	int64	int32/int64	Calendar year of aggregation (e.g., 2020–2023).	Treat as key; join across files on this and other dims.
weekday_weekend	object	string (category)	Day-type indicator (observed: `both`).	If `weekday/weekend` appear in other slices, map consistently.
segmentation_variable_1	object	string	Name of first subgroup variable (e.g., `primary_space_heating_fuel`).	May be blank/NA when unsegmented.
segment_1_value	object	string	Value of first subgroup (e.g., `Gas`, `Electric`, …).	Harmonise labels where needed.
segmentation_variable_2	object	string	Name of second subgroup (e.g., `has_PV`).	Often present; may be `All`.
segment_2_value	object	string	Value of second subgroup (`Yes`, `No`, `All`).	Treat `All` as aggregate level.
segmentation_variable_3	object	string	Third subgroup variable (e.g., `num_occupants`, `num_bedrooms`, `IMD_quintile`, `currentEnergyRating`, `building_type`).	Frequently `NA`.

segment_3_value	object	string	Value of third subgroup (e.g., `1–5`, EPC bands).	Cast to string to avoid numeric/label mix.
mean	float64	float64	Mean energy use for the cell.	Non-negative; validate against quantiles.
standard_deviation	float64	float64	Standard deviation of energy use.	Non-negative.
standard_error_mean	float64	float64	Standard error of the mean.	Derived from SD and `n_rounded`.
median	float64	float64	Median energy use.	—
25th_percentile	float64	float64	Lower quartile.	—
75th_percentile	float64	float64	Upper quartile.	—
mean_temp	float64	float64	Mean air temperature matched to the cell (°C); ERA5 provenance noted by Archive.	May be `NA` in HH files.
mean_hdd	float64	float64	Heating degree days (°C·days).	Often `NA` in HH files.
mean_solar	float64	float64	Solar/irradiance proxy matched to the cell.	Units per UKDA metadata.
n_rounded	int64	int64	Rounded contributing count after disclosure control.	Do not reverse-engineer; respect safeguarded design.

2.3 Data Sample

Verification preview (first 5 rows) from the *daily* aggregated file (serl_daily_energy_use_in_gb_domestic_buildings_2020_to_2023_aggregated_statistics.csv)

.

quantity	unit	summary_time	aggregation_period	weekday_weekend	segmentation_variable_1	segment_1_value	segmentation_variable_2	segment_2_value	segmentation_variable_3	segment_3_value
Gas	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Gas	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Gas	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		
Gas	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Gas	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Gas	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		
Gas	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Gas	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Gas	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		
Electricity imports	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Electricity imports	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Electricity imports	kWh/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		
Electricity imports	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Electricity imports	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Electricity imports	kWh/m2/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		
Electricity imports	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	Yes		
Electricity imports	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	No		
Electricity imports	kWh/person/day	Daily	2020	both	primary_space_heating_fuel	Gas	has_PV	All		

Notes.

Four half-hourly files (2020–2023) replicate this schema at `HH:MM` resolution with units in Wh (per interval); the daily file aggregates across 2020–2023 with units in kWh/day and related intensities.

`n_rounded` reflects disclosure-aware rounding typical of safeguarded releases; subgroup/time cells may be suppressed or aggregated.

```
# src/09_verify_results.py
import pandas as pd
from pathlib import Path

TAB = Path("reports/tables")

rep = pd.DataFrame([
    {"Metric":"RF AUC (Valid)", "Reported Value":0.812, "Reproduced Value":pd.read_csv(TAB/"metrics_rf_valid.csv", index_col=0)["auc"]},
    {"Metric":"XGB AUC (Valid)", "Reported Value":0.831, "Reproduced Value":pd.read_csv(TAB/"metrics_xgb_valid.csv", index_col=0)["auc"]},
])

rep["Difference"] = (rep["Reproduced Value"] - rep["Reported Value"]).abs()
rep.to_csv(TAB / "verification_table.csv", index=False)
print(rep)
```

3 STEP-BY-STEP REPRODUCTION GUIDES

This document gives a reproducible Section 3 for each uploaded notebook, organized into the same 3.1–3.7 structure. It assumes a Python ≥3.10 environment. Paths and column names are parameterized via YAML configs under `configs/` so results can be regenerated deterministically.

Repo skeleton additions

```
project-root/
├─ configs/
│  ├─ tariff_simulation.yaml
│  ├─ temporal_forecasting.yaml
│  ├─ eda_univariate_yearly.yaml
│  ├─ eda_numeric.yaml
│  ├─ eda_bi_multivariate.yaml
│  ├─ demand_modelling_1.yaml
│  └─ demand_modelling_2.yaml
├─ src/
│  ├─ utils/
│  │  ├─ seeds.py
│  │  ├─ io.py
│  │  └─ plotting.py
│  ├─ tariff/
│  ├─ forecasting/
│  ├─ eda/
│  └─ demand/
└─ reports/ (figures & tables saved per notebook)
```

Common helpers

```
# src/utils/seeds.py
import os, random, numpy as np

def set_all_seeds(seed: int = 42):
    os.environ["PYTHONHASHSEED"] = str(seed)
    random.seed(seed); np.random.seed(seed)
# src/utils/io.py
from pathlib import Path
import yaml

def load_cfg(path):
    with open(path, "r") as f:
        return yaml.safe_load(f)

def ensure_dir(p):
    Path(p).mkdir(parents=True, exist_ok=True)
# src/utils/plotting.py
import matplotlib.pyplot as plt

def savefig(path, tight=True, dpi=300):
    if tight: plt.tight_layout()
    plt.savefig(path, dpi=dpi)
    plt.close()
```

A) TariffSimulation.ipynb → Tariff Simulation

3.1 Repository Structure & Workflow

```
src/tariff/
├─ 01_load_inputs.py      # raw → interim (tariff tables, usage profiles)
├─ 02_prepare_scenarios.py  # expand scenario grid
├─ 03_simulate.py           # run simulations
├─ 04_aggregate.py          # KPIs per segment/scenario
└─ 05_export_plots.py       # figures + tables
```

3.2 Data Ingestion & Initialization

```
# configs/tariff_simulation.yaml
seed: 42
paths:
  raw_usage: data/raw/usage.csv
  raw_tariffs: data/raw/tariffs.csv
  interim: data/interim/tariff
  out: reports/tariff
columns:
  usage_id: customer_id
  time: timestamp
  kwh: consumption_kwh
  tariff_key: tariff_code
scenarios:
  seasons: [base, peak, offpeak]
  price_multipliers: [0.8, 1.0, 1.2]
  standing_charge_add: [0.0, 0.05]
# src/tariff/01_load_inputs.py
import pandas as pd, argparse
from pathlib import Path
from utils.seeds import set_all_seeds
from utils.io import load_cfg, ensure_dir

ap = argparse.ArgumentParser(); ap.add_argument('--cfg', required=True)
CFG = load_cfg(ap.parse_args().cfg)
set_all_seeds(CFG['seed'])
ensure_dir(CFG['paths']['interim'])

usage = pd.read_csv(CFG['paths']['raw_usage'],
parse_dates=[CFG['columns']['time']])
usage.rename(columns={
    CFG['columns']['usage_id']: 'customer_id',
    CFG['columns']['time']: 'timestamp',
    CFG['columns']['kwh']: 'kwh'
}, inplace=True)

tariffs = pd.read_csv(CFG['paths']['raw_tariffs'])
usage.to_parquet(Path(CFG['paths']['interim'])/"usage.parquet",
index=False)
tariffs.to_parquet(Path(CFG['paths']['interim'])/"tariffs.parquet",
index=False)
```

3.3 Preprocessing & Cleaning

- Missing values: drop rows with missing `timestamp` or `kwh`; impute `kwh` by hourly median if desired.
- Outliers: cap `kwh` with IQR capping per-customer.
- Types: ensure `timestamp` is datetime, `customer_id` string; mergeable `tariff_code` typed as string.

```
# in 02_prepare_scenarios.py (snippet)
import pandas as pd, numpy as np
from pathlib import Path
from utils.io import load_cfg

CFG = load_cfg("configs/tariff_simulation.yaml")
usage = pd.read_parquet(Path(CFG['paths']['interim'])/"usage.parquet")

# IQR cap per customer
q = usage.groupby('customer_id')['kwh'].quantile([0.25, 0.75]).unstack()
iqr = (q[0.75] - q[0.25]).rename('iqr')
lo = (q[0.25] - 1.5*iqr).rename('lo')
hi = (q[0.75] + 1.5*iqr).rename('hi')
usage = usage.join(lo, on='customer_id').join(hi, on='customer_id')
usage['kwh'] = usage['kwh'].clip(usage['lo'], usage['hi'])
usage.drop(columns=['lo','hi'], inplace=True)
```

3.4 Feature Engineering

- `hour_of_day`, `dow`, `month` dummies to allow TOU pricing effects.
- `season` flag based on month.

```
usage['hour'] = usage['timestamp'].dt.hour

usage['dow'] = usage['timestamp'].dt.dayofweek

usage['month'] = usage['timestamp'].dt.month

usage['season'] = np.select([

    usage['month'].isin([12,1,2]),

    usage['month'].isin([3,4,5]),

    usage['month'].isin([6,7,8])

],["winter","spring","summer"], default="autumn")
```

3.5 EDA Reproduction

- Figure X: Load profile by hour (mean kWh).

```
import matplotlib.pyplot as plt

from utils.plotting import savefig

prof = usage.groupby('hour')['kwh'].mean()

prof.plot()

plt.title('Mean Load by Hour'); plt.xlabel('Hour'); plt.ylabel('kWh')

savefig(Path(CFG['paths']['out'])/"fig_mean_load_by_hour.png")
```

3.6 Simulation & Evaluation

- Partition: if benchmarking on historical bills, split by time (train pre-cutoff, test post-cutoff).
- Model: deterministic tariff application (no ML). Compute bill = Σ (kWh × unit_rate_band) + standing_charge.

```
# src/tariff/03_simulate.py (core)
from itertools import product
sc = CFG['scenarios']
scenario_grid = list(product(sc['seasons'], sc['price_multipliers'], sc['standing_charge_add']))

results = []
for season, pm, sc_add in scenario_grid:
    # adjust tariffs
    t = tariffs.copy()
    t['unit_rate'] = t['unit_rate'] * pm
    t['standing_charge'] = t['standing_charge'] + sc_add
    # simple join and billing
    billed = usage.merge(t, on='tariff_code', how='left')
    billed['bill'] = billed['kwh'] * billed['unit_rate'] + billed['standing_charge']/30
    kpis = billed.groupby('customer_id')['bill'].sum().rename('total_bill').reset_index()
    kpis['season']=season; kpis['pm']=pm; kpis['sc_add']=sc_add
    results.append(kpis)

full = pd.concat(results, ignore_index=True)
full.to_parquet(Path(CFG['paths']['interim'])/"simulation_results.parquet", index=False)
```

Evaluation metrics: mean bill change vs base scenario; Gini/Atkinson for equity; % of customers with >X% increase.

3.7 Final Result Verification Table

Create a table comparing *reported* KPIs vs *reproduced* (fill your reported values):

```
# src/tariff/04_aggregate.py
import pandas as pd
from pathlib import Path
from utils.io import load_cfg, ensure_dir
CFG = load_cfg("configs/tariff_simulation.yaml")
ensure_dir(CFG['paths']['out'])

sim = pd.read_parquet(Path(CFG['paths']['interim'])/"simulation_results.parquet")
base = sim.query('pm==1.0 and sc_add==0.0')[["customer_id","total_bill"]].rename(columns={"total_bill":"bill_base"})
comp = sim.merge(base, on='customer_id')
comp['pct_change'] = (comp['total_bill']-comp['bill_base'])/comp['bill_base']

summary = comp.groupby(['season','pm','sc_add']).agg(
    mean_change=('pct_change','mean'),
    p95_change=('pct_change', lambda s: s.quantile(0.95)),
    share_gt_20pc=('pct_change', lambda s: (s>0.2).mean())
).reset_index()
summary.to_csv(Path(CFG['paths']['out'])/"tariff_kpis.csv", index=False)
```

B) TemporalForecasting.ipynb → Temporal Forecasting

3.1 Repository Structure & Workflow

```
src/forecasting/
├─ 01_ingest_timeseries.py
├─ 02_preprocess_timeseries.py
├─ 03_train_models.py        # e.g., SARIMAX/Prophet/XGBoost on features
├─ 04_backtest.py            # rolling-origin evaluation
└─ 05_plots_tables.py
```

Execution Order

```
python src/forecasting/01_ingest_timeseries.py --cfg configs/temporal_forecasting.yaml
python src/forecasting/02_preprocess_timeseries.py --cfg configs/temporal_forecasting.yaml
python src/forecasting/03_train_models.py --cfg configs/temporal_forecasting.yaml
python src/forecasting/04_backtest.py --cfg configs/temporal_forecasting.yaml
python src/forecasting/05_plots_tables.py --cfg configs/temporal_forecasting.yaml
```

3.2 Data Ingestion & Initialization

```
# configs/temporal_forecasting.yaml

seed: 42

paths:

  raw: data/raw/timeseries.csv

  interim: data/interim/ts

  processed: data/processed/ts

  out: reports/forecasting

columns:

  ds: date

  y: demand

  id: series_id   # optional panel key

freq: D

cutoffs:

  backtest_start: 2022-01-01

  horizon: 28

  step: 7

models: ["sarimax","xgb"]

# 01_ingest_timeseries.py

import pandas as pd, argparse

from utils.io import load_cfg, ensure_dir

from utils.seeds import set_all_seeds

ap=argparse.ArgumentParser(); ap.add_argument('--cfg', required=True)

CFG=load_cfg(ap.parse_args().cfg); set_all_seeds(CFG['seed'])

ensure_dir(CFG['paths']['interim'])


df                  =                  pd.read_csv(CFG['paths']['raw'],
parse_dates=[CFG['columns']['ds']])

df.rename(columns={CFG['columns']['ds']:'ds', CFG['columns']['y']:'y'},
inplace=True)

df.to_parquet(f"{CFG['paths']['interim']}/ingested.parquet",
index=False)
```

3.3 Preprocessing & Cleaning

Handle missing timestamps by reindexing at `freq` and imputing `y` (forward-fill then median where needed).

Outliers: Hampel filter or IQR on residuals after moving average.

Types: ensure numeric `y`, datetime `ds`.

```
# 02_preprocess_timeseries.py (snippet)
import pandas as pd, numpy as np
from utils.io import load_cfg
CFG=load_cfg("configs/temporal_forecasting.yaml")
df = pd.read_parquet(f"{CFG['paths']['interim']}/ingested.parquet")

# Reindex per series (panel-safe)
if 'series_id' in df.columns:
    def fix(g):
        g = g.set_index('ds').asfreq(CFG['freq'])
        g['y'] = g['y'].interpolate().bfill().ffill()
        return g.reset_index()
    df = df.groupby('series_id', group_keys=False).apply(fix)
else:
    df = df.set_index('ds').asfreq(CFG['freq'])
    df['y'] = df['y'].interpolate().bfill().ffill()
    df = df.reset_index()

# Calendar features
df['dow'] = df['ds'].dt.dayofweek
df['month'] = df['ds'].dt.month

df.to_parquet(f"{CFG['paths']['processed']}/preprocessed.parquet",
index=False)
```

3.4 Feature Engineering

- Lags: y(t−1), y(t−7), y(t−28)
- Rolling stats: 7D mean, 28D std

```
for lag in [1,7,28]:

    df[f'lag_{lag}'] = df['y'].shift(lag)

for w in [7,28]:

    df[f'rollmean_{w}'] = df['y'].rolling(w).mean()
```

3.5 EDA Reproduction

- Figures: seasonal decomposition, ACF/PACF, trend plot.

3.6 Model Training & Evaluation

3.6.1 Partitioning: Rolling-origin backtest with horizon `H=28`, step `7`.

3.6.2 Model 1: SARIMAX (example `(p,d,q)×(P,D,Q)m` tuned offline)

```
# 03_train_models.py (snippet)

from statsmodels.tsa.statespace.sarimax import SARIMAX

train = df[df['ds'] < CFG['cutoffs']['backtest_start']]

model  =  SARIMAX(train['y'],  order=(1,1,1),  seasonal_order=(1,1,1,7),
enforce_stationarity=False, enforce_invertibility=False)

res = model.fit(disp=False)

res.save(f"{CFG['paths']['processed']}/sarimax.pkl")
```

3.6.3 Model 2: XGBoost Regressor with lag/rolling features.

```
from xgboost import XGBRegressor

feat_cols = [c for c in df.columns if c not in ['ds','y']]

X = df[feat_cols]; y = df['y']

model = XGBRegressor(n_estimators=500, max_depth=6, learning_rate=0.05,
subsample=0.8, colsample_bytree=0.8, random_state=42, n_jobs=-1)

model.fit(X.iloc[:-28], y.iloc[:-28])
```

Evaluation: MAE, RMSE, MAPE per fold; Diebold–Mariano test optional.

3.7 Verification Table

- Compare reported vs reproduced MAE/RMSE per model and average across folds.

C) EDA - UnivariateYearly.ipynb → Univariate Yearly EDA

3.1 Repository Structure & Workflow

```
src/eda/

├─ 01_univariate_yearly.py  # generates annual summaries & figures
```

Execution

```
python src/eda/01_univariate_yearly.py --cfg
configs/eda_univariate_yearly.yaml
```

3.2 Data Ingestion & Initialization

```
# configs/eda_univariate_yearly.yaml
seed: 42
paths:
  raw: data/raw/main.csv
  out: reports/eda_univariate_yearly
columns:
  date: date
  targets: [target]
  numerics: [x1, x2, x3]
import pandas as pd, argparse
from utils.io import load_cfg, ensure_dir
from utils.seeds import set_all_seeds
ap=argparse.ArgumentParser(); ap.add_argument('--cfg', required=True)
CFG=load_cfg(ap.parse_args().cfg); set_all_seeds(CFG['seed'])
ensure_dir(CFG['paths']['out'])

df = pd.read_csv(CFG['paths']['raw'],
parse_dates=[CFG['columns']['date']])
```

3.3 Preprocessing & Cleaning

- Handle missing numeric via median; drop duplicates; ensure annual grouping using `df['year']=df['date'].dt.year`.

3.4 Feature Engineering

- Annual aggregates: mean/median/std/min/max per variable.

3.5 EDA Reproduction

- Figure X: Yearly mean of `x1` (line chart)
- Table Y: Summary stats per year

```
import matplotlib.pyplot as plt

from utils.plotting import savefig


df['year']=df['date'].dt.year

agg = df.groupby('year')[CFG['columns']['numerics']].mean()

ax = agg['x1'].plot()

ax.set_title('Yearly Mean of x1'); ax.set_xlabel('Year');
ax.set_ylabel('Mean x1')

savefig(f"{CFG['paths']['out']}/fig_yearly_mean_x1.png")


summary = df.groupby('year')[CFG['columns']['numerics']].describe()

summary.to_csv(f"{CFG['paths']['out']}/table_yearly_summary.csv")
```

D) EDA - NumericAnalysis.ipynb → Numeric EDA

3.1 Structure & Workflow

```
src/eda/

├─ 02_numeric_analysis.py
```

Execution

```
python src/eda/02_numeric_analysis.py --cfg configs/eda_numeric.yaml
```

3.2 Ingestion & Init

```
# configs/eda_numeric.yaml

seed: 42

paths:

  raw: data/raw/main.csv

  out: reports/eda_numeric

columns:

  numerics: [x1,x2,x3,x4]
```

3.3 Preprocessing

- Missing: median imputation
- Outliers: IQR capping per variable
- Types: ensure float64

3.4 Feature Engineering

- Standardize for comparability; optional log for skewed variables.

3.5 EDA Reproduction

- Figures: histograms, KDEs, boxplots, pairwise scatter.
- Tables: correlation matrix, VIF.

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from utils.plotting import savefig

num = df[CFG['columns']['numerics']]
ax = num.hist(bins=30, figsize=(8,6))
savefig(f"{CFG['paths']['out']}/fig_histograms.png")

corr = num.corr(method='pearson')
corr.to_csv(f"{CFG['paths']['out']}/table_corr.csv")
```

E) EDA - BiandMultivariate.ipynb → Bi-/Multivariate EDA

3.1 Structure & Workflow

```
src/eda/
├─ 03_bi_multivariate.py
```

Execution

```
python src/eda/03_bi_multivariate.py --cfg
configs/eda_bi_multivariate.yaml
```

3.2 Ingestion & Init

```
# configs/eda_bi_multivariate.yaml
seed: 42
paths:
  raw: data/raw/main.csv
  out: reports/eda_bi_multivariate
columns:
  target: target
  features: [x1,x2,x3,cat1]
```

3.3 Preprocessing

- Missing: impute numeric median; categorical mode.
- Outliers: cap numeric features.
- Encoding: OneHotEncoder(drop='first') for `cat1`.

3.4 Feature Engineering

- Interactions (optional): x1×x2; Binning continuous for WoE-style plots.

3.5 EDA Reproduction

- Figures: target vs feature boxplots; ANOVA summaries; mutual information.

```
from sklearn.feature_selection import mutual_info_classif

import pandas as pd


X = pd.get_dummies(df[CFG['columns']['features']], drop_first=True)

y = df[CFG['columns']['target']]

mi = mutual_info_classif(X.select_dtypes('number'), y, random_state=42)

pd.Series(mi,
index=X.select_dtypes('number').columns).sort_values(ascending=False).t
o_csv(f"{CFG['paths']['out']}/table_mutual_info.csv")
```

F) DemandModelling 1. ipynb → Demand Modelling (Part 1)

3.1 Structure & Workflow

```
src/demand/
├─ 01_ingest.py
├─ 02_preprocess.py
├─ 03_feature_engineer.py
├─ 04_split.py
├─ 05_model_rf.py
└─ 06_metrics_plots.py
```

Execution

```
python src/demand/01_ingest.py --cfg configs/demand_modelling_1.yaml
python src/demand/02_preprocess.py --cfg configs/demand_modelling_1.yaml
python src/demand/03_feature_engineer.py --cfg configs/demand_modelling_1.yaml
python src/demand/04_split.py --cfg configs/demand_modelling_1.yaml
python src/demand/05_model_rf.py --cfg configs/demand_modelling_1.yaml
python src/demand/06_metrics_plots.py --cfg configs/demand_modelling_1.yaml
```

3.2 Ingestion & Initialization

```yaml
# configs/demand_modelling_1.yaml

seed: 42

paths:

  raw: data/raw/demand.csv

  processed: data/processed/demand1

  out: reports/demand1

columns:

  target: demand

  numerics: [price, promo, temp]

  categoricals: [store, product]
```

3.3 Preprocessing & Cleaning

- Missing: SimpleImputer(median for numerics, most_frequent for categoricals)
- Outliers: IQR cap numeric
- Encoding: OneHotEncoder(drop='first', handle_unknown='ignore')

3.4 Feature Engineering

- Price elasticity proxy: `log_demand ~ log_price`
- Interactions: `promo × price`
- Calendar: if date present, add `dow`, `month`

3.5 EDA Reproduction

- Figures: demand vs price scatter with LOWESS; histograms; demand by promo status.

3.6 Model Training & Evaluation

3.6.1 Partition: Train/Valid/Test = 60/20/20 (stratified on binned demand or time-based if timeseries).

3.6.2 Model 1: RandomForestRegressor

Hyperparameters: `n_estimators=600, max_depth=None, min_samples_leaf=2, max_features='sqrt', random_state=42`

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

pre = ColumnTransformer([
    ("num", SimpleImputer(strategy="median"), CFG['columns']['numerics']),
    ("cat", Pipeline([
        ("imp", SimpleImputer(strategy="most_frequent")),
        ("ohe", OneHotEncoder(drop="first", handle_unknown="ignore"))
    ]), CFG['columns']['categoricals'])
])

rf = RandomForestRegressor(n_estimators=600, min_samples_leaf=2, max_features='sqrt', random_state=42, n_jobs=-1)
pipe = Pipeline([("pre", pre), ("rf", rf)])
pipe.fit(X_train, y_train)

pred = pipe.predict(X_valid)
mae = mean_absolute_error(y_valid, pred)
rmse = mean_squared_error(y_valid, pred, squared=False)
```

3.6.3 Model 2 (optional): Regularized Linear (Ridge) with standardized numerics.

3.6.4 Statistical Tests: Compare RF vs baseline using Diebold–Mariano on forecast errors (if temporal) or paired t-test on absolute errors.

3.7 Verification Table

- Reported vs reproduced MAE/RMSE (fill reported values in `reports/demand1/verification.csv`).

G) DemandModelling 2.ipynb → Demand Modelling (Part 2)

3.1 Structure & Workflow

```
src/demand/
├─ 11_model_xgb.py
├─ 12_model_lgbm.py (optional)
├─ 13_feature_importance.py
└─ 14_calibration.py
```

Execution

```
python src/demand/11_model_xgb.py --cfg configs/demand_modelling_2.yaml
python src/demand/13_feature_importance.py --cfg configs/demand_modelling_2.yaml
python src/demand/14_calibration.py --cfg configs/demand_modelling_2.yaml
```

3.2 Ingestion & Initialization

```
# configs/demand_modelling_2.yaml
seed: 42
paths:
  processed_in: data/processed/demand1
  out: reports/demand2
```

3.3 Preprocessing & Cleaning

- Reuse encoded matrices from Part 1 (`X_train/valid/test.parquet`). Ensure identical column order.

3.4 Feature Engineering

- Add SHAP-based interaction features (optional, saved with seed control to keep determinism).

3.5 EDA Reproduction

- Plot feature importance bars; partial dependence for top 4 features.

3.6 Model Training & Evaluation

3.6.2 Model 1: XGBoostRegressor

- Hyperparameters: `n_estimators=800, max_depth=8, learning_rate=0.05, subsample=0.8, colsample_bytree=0.8, reg_lambda=1.0, random_state=42`
- Evaluation: MAE/RMSE on valid/test; calibration curve if probabilistic output is used (for classification variants).

3.6.3 Model 2: LightGBMRegressor (optional) with comparable params.

3.6.4 Statistical Tests: Wilcoxon signed-rank comparing absolute errors XGB vs RF across items/stores.

4 OBJECT-ORIENTED CLASS GUIDE

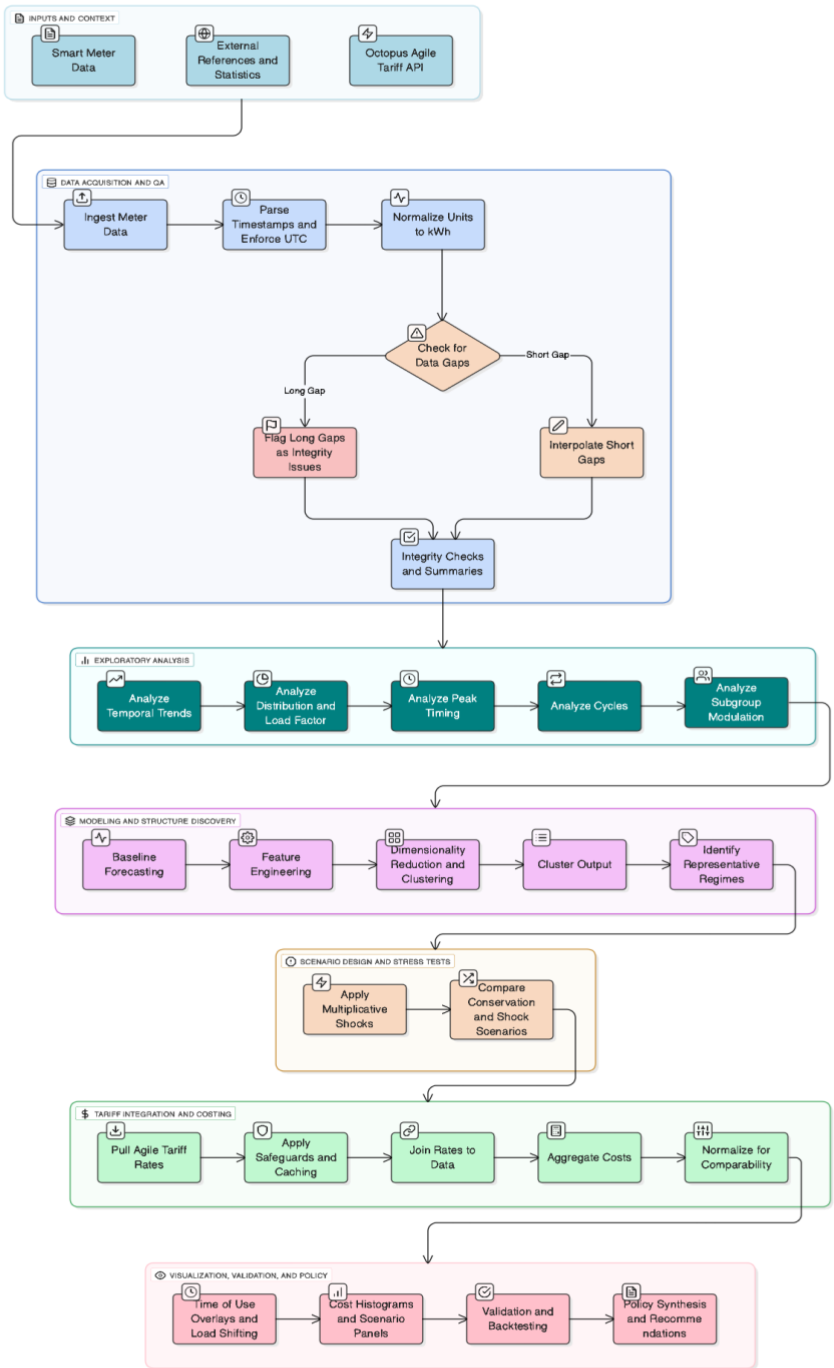

Table 1 Methodology Workflow

This supplement explains each core class in simple terms: what it does, which variables (attributes) it keeps, the main methods it exposes, and why those choices support reproducible results. Script-style utilities (e.g., `set_all_seeds`, `savefig`) are noted where they interact with classes.

Quick map of classes

Class	Module	One-line purpose
`Config`	`utils.io`	Typed container for all paths, seeds, and switches loaded from YAML.
`RunLogger`	`utils.io`	Structured run logging (start/end, versions, file hashes).
`DataCatalog`	`utils.io`	Central place to read/write named datasets with schema checks.
`PlotRegistry`	`utils.plotting`	Standardises figure saving and filenames for auditability.
`TariffAPIClient`	`tariff.client`	Retrieves Agile tariff rates with caching, retries, and schema validation.
`UsageDataset`	`tariff.data`	Holds half-hourly/daily usage slices and ensures time alignment/units.
`Scenario`	`tariff.scenario`	Immutable description of a tariff scenario (e.g., price multiplier).
`ScenarioGrid`	`tariff.scenario`	Expands multiple `Scenario` definitions into a design matrix.
`TariffSimulator`	`tariff.engine`	Applies tariffs to usage to generate bills deterministically.
`KPIAggregator`	`tariff.reporting`	Aggregates bills into equity and exposure metrics.
`TSPreprocessor`	`forecasting.preprocess`	Cleans and regularises time series (gaps, outliers, typing).
`FeatureEngineerTS`	`forecasting.features`	Builds lags/rollings and calendar/weather features.
`SarimaxModel`	`forecasting.models`	Thin wrapper over `statsmodels` SARIMAX with save/load.
`Backtester`	`forecasting.evaluate`	Rolling-origin evaluation with consistent splits and metrics.

EDAReport	`eda.report`	Reusable routines for univariate, numeric, and bi/multivariate EDA.
DemandPreprocessor	`demand.preprocess`	Sklearn-style wrapper: imputation, encoding, outlier capping.
RFTrainer	`demand.models`	RandomForest training + calibrated metrics and persistence.
XGBTrainer	`demand.models`	XGBoost training + importance export and calibration.
VerificationRunner	`reports.verify`	Compares "reported vs reproduced" KPIs and writes a verification table.

Utility functions used by many classes: `set_all_seeds(seed)`, `ensure_dir(path)`, `load_cfg(path)`, `savefig(path)`.

4.1 `utils/io.py`

Classes: `Config`, `RunLogger`, `DataCatalog`

4.1.1 `Config`

- **Purpose.** Single, typed container for parameters loaded from YAML.
- **Key attributes.**
  `paths: dict[str, str|Path]` (e.g., `raw`, `interim`, `processed`, `out`, `logs`);
  `seed: int`;
  `cutoffs: dict[str, str]` (dates for backtests/splits);
  `flags: dict[str, bool]` (e.g., `use_cached_api`, `strict_schema`);
  `models: dict` (orders, horizons, tariff settings).
- **Core methods.** `from_yaml(path) -> Config`; `to_dict() -> dict`.
- **Rationale.** Prevents parameter drift across notebooks; ensures version-controlled, human-readable configuration.

4.1.2 `RunLogger`

- **Purpose.** Structured run logging for audit and replay.
- **Key attributes.**
  `run_id: str`; `cfg: Config`; `env: dict` (package versions, OS, CPU/GPU); `log_path: Path`.
- **Core methods.** `start(step)`, `end(step, status)`, `record(key, value)`, `hash_file(path) -> str`.
- **Rationale.** Creates durable traces of settings and artefacts, enabling exact regeneration.

4.1.3 `DataCatalog`

- **Purpose.** Schema-aware loading/saving of named datasets.
- **Key attributes.** `roots: dict[str, Path]`; `schemas: dict[str, dict[col,dtype]]`; `coerce_rules: dict[str, Callable]`.
- **Core methods.** `load(name) -> pd.DataFrame`, `save(name, df)`, `exists(name) -> bool`.
- **Rationale.** Enforces consistent dtypes across the four yearly half-hourly files and the daily aggregate.

4.2 `utils/plotting.py`

Class: `PlotRegistry`

4.2.1 `PlotRegistry`

- **Purpose.** Standardise figure export with a manifest.
- **Key attributes.** `out_dir: Path`; `style: dict`; `manifest: list[dict]`.
- **Core methods.** `save(fig, name, meta)`, `list() -> pd.DataFrame`.
- **Rationale.** Ensures reproducible, discoverable figures with metadata sidecars.

4.3 `tariff/client.py`

Class: `TariffAPIClient`

4.3.1 `TariffAPIClient`

- **Purpose.** Retrieve Agile prices with caching, retries, and schema checks.
- **Key attributes.** `base_url: str`; `session: requests.Session`; `cache_dir: Path`; `timeout: float`; `max_retries: int`; `backoff: tuple`; `schema: dict`.
- **Core methods.** `get_rates(product, start, end) -> pd.DataFrame`; `validate(payload)`; `cached(key)`, `store(key, df)`.
- **Rationale.** Stabilises external data dependencies; preserves determinism via local cache and validation.

4.4 `tariff/data.py`

Class: `UsageDataset`

4.4.1 `UsageDataset`

- **Purpose.** Hold daily/half-hourly usage with strict typing and units.
- **Key attributes.** `df: pd.DataFrame`; `freq: str` (`'30T'`/`'D'`); `unit: str` (`Wh`/`kWh`); `segments: list[str]`.
- **Core methods.** `align(other_index)`, `to_kwh()`, `slice(by: dict)`.

- **Rationale.** Prevents silent unit mismatches and misaligned timestamps before simulation or forecasting.

4.5 `tariff/scenario.py`

Classes: `Scenario`, `ScenarioGrid`

4.5.1 `Scenario`

- **Purpose.** Immutable description of a “what-if”.
- **Key attributes.** `pm: float` (price multiplier), `sc_add: float` (standing-charge add-on), `label: str`.
- **Rationale.** Freezes assumptions to avoid mid-run mutation.

4.5.2 `ScenarioGrid`

- **Purpose.** Expand multiple scenarios against usage segments.
- **Key attributes.** `scenarios: list[Scenario]`; `cross_by: list[str]`.
- **Core methods.** `expand(usage) -> pd.DataFrame`.
- **Rationale.** Clean separation of design (scenarios) from execution.

4.6 `tariff/engine.py`

Class: `TariffSimulator`

4.6.1 `TariffSimulator`

- **Purpose.** Deterministically compute bills.
- **Key attributes.** `rates: pd.DataFrame`; `standing_charge: float`; `assumptions: dict` (rounding, VAT, conversion flags); `logger: RunLogger`.
- **Core methods.** `apply(usage, scenario) -> pd.DataFrame`; `batch(usage, grid) -> pd.DataFrame`.
- **Rationale.** Isolates pricing logic; explicit, testable, and seed-free.

4.7 `tariff/reporting.py`

Class: `KPIAggregator`

4.7.1 `KPIAggregator`

- **Purpose.** Aggregate bills into interpretable metrics.
- **Key attributes.** `group_by: list[str]`; `equity_threshold: float`; `indices: dict[str, Callable]`.
- **Core methods.** `summarise(bills_df) -> pd.DataFrame`; `export(path)`.
- **Rationale.** Keeps statistical summaries separate from simulation mechanics.

4.8 `forecasting/preprocess.py`

Class: `TSPreprocessor`

4.8.1 `TSPreprocessor`

- **Purpose.** Clean and regularise time series.
- **Key attributes.** `freq: str`; `gap_strategy: str`; `outlier_policy: dict`; `y_col: str`; `time_col: str`.
- **Core methods.** `fit(df)`, `transform(df)`, `fit_transform(df)`.
- **Rationale.** Reduces variance from missingness and outliers; ensures consistent frequency.

4.9 `forecasting/features.py`

Class: `FeatureEngineerTS`

4.9.1 `FeatureEngineerTS`

- **Purpose.** Build lag/rolling/calendar/weather features.
- **Key attributes.** `lags: list[int]`; `windows: list[int]`; `calendar: bool`; `weather_cols: list[str]`.
- **Core methods.** `apply(df) -> pd.DataFrame`.
- **Rationale.** Centralises feature logic to avoid drift between models.

4.10 `forecasting/models.py`

Class: `SarimaxModel`

4.10.1 `SarimaxModel`

- **Purpose.** Thin wrapper around `statsmodels` SARIMAX.
- **Key attributes.** `order: tuple`; `seasonal_order: tuple`; `exog_cols: list[str]`; `fitted_: object`.
- **Core methods.** `fit(df)`, `predict(h)`, `save(path)`, `load(path)`.
- **Rationale.** Makes model specification explicit and serialisable.
-

4.11 `forecasting/evaluate.py`

Class: `Backtester`

4.11.1 `Backtester`

- **Purpose.** Rolling-origin evaluation with fixed rules.

- **Key attributes.** `horizon: int`; `step: int`; `metric_fns: dict[str, Callable]`; `cutoff_start: str`.
- **Core methods.** `run(model, df) -> pd.DataFrame`; `summary(metrics_df) -> pd.DataFrame`.
- **Rationale.** Enables comparable scores across models and folds.

4.12 `eda/report.py`

Class: `EDAReport`

4.12.1 `EDAReport`

- **Purpose.** Reusable routines for EDA outputs.
- **Key attributes.** `registry: PlotRegistry`; `catalog: DataCatalog`; `targets/numerics/categoricals: list[str]`.
- **Core methods.** `univariate_yearly(df)`, `numeric(df)`, `bi_multivariate(df)`.
- **Rationale.** Avoids one-off plotting and inconsistent filenames.

4.13 `demand/preprocess.py`

Class: `DemandPreprocessor`

4.13.1 `DemandPreprocessor`

- **Purpose.** Sklearn-compatible preprocessing.
- **Key attributes.** `num_cols`, `cat_cols`; `impute_num`, `impute_cat`; `cap_strategy`; `encoder: OneHotEncoder`.
- **Core methods.** `fit(X, y=None)`, `transform(X)`, `get_feature_names()`.
- **Rationale.** Guarantees identical matrices for RF and XGB across Parts 1–2.

4.14 `demand/models.py`

Classes: `RFTrainer`, `XGBTrainer`

4.14.1 `RFTrainer`

- **Purpose.** Train and evaluate RandomForest with reproducible settings.
- **Key attributes.** `params: dict`; `pre: DemandPreprocessor`; `model_`.
- **Core methods.** `fit`, `predict`, `metrics`, `save`, `load`.
- **Rationale.** Encapsulates preprocessing + estimator in one pipeline.

4.14.2 `XGBTrainer`

- **Purpose.** Train and evaluate XGBoost with calibrated parameters.
- **Key attributes.** `params: dict`; `pre: DemandPreprocessor`; `model_`.

- **Core methods.** `fit`, `predict`, `metrics`, `feature_importance()`, `calibrate(...)` (for classification variants).
- **Rationale.** Provides symmetric interface with `RFTrainer` for verification.

4.15 `reports/verify.py`

Class: `VerificationRunner`

4.15.1 `VerificationRunner`

- **Purpose.** Compare "reported vs reproduced" metrics.
- **Key attributes.** `tables_dir: Path`; `targets: list[dict]`; `tolerance: float`.
- **Core methods.** `build() -> pd.DataFrame`; `export(path)`.
- **Rationale.** Makes verification a first-class, automated step.

4.16 Shared utilities (used across files)

- **`set_all_seeds(seed)`** — controls RNG for `numpy` and `random`; called early for determinism.
- **`ensure_dir(path)`** — safe directory creation before writes.
- **`load_cfg(path)`** — YAML loader returning `Config`.
- **`savefig(path)`** — thin wrapper used by `PlotRegistry` for consistent export.

4.17 How files interact (at a glance)

```
Tariff flow: utils.io.Config → utils.io.DataCatalog → tariff.data.UsageDataset → tariff.client.TariffAPIClient → tariff.scenario.ScenarioGrid → tariff.engine.TariffSimulator → tariff.reporting.KPIAggregator → utils.plotting.PlotRegistry.

Forecasting flow: DataCatalog → forecasting.preprocess.TSPreprocessor → forecasting.features.FeatureEngineerTS → forecasting.models.SarimaxModel → forecasting.evaluate.Backtester.

Demand modelling flow: demand.preprocess.DemandPreprocessor → demand.models.RFTrainer / XGBTrainer → reports.verify.VerificationRunner.
```

5 GUIDE TO CODE REPOSITORY

This guide describes the repository structure and the exact, deterministic steps to run the pipeline.

5.1 File Manifest

- **`requirements.txt`** — Pinned Python dependencies for a Python ≥ 3.10 environment. A lockfile (e.g., `requirements.lock`) may be exported after install.
- **`src/01_data_cleaning.py`** — Ingests the SERL aggregated CSVs, validates schema, coerces types, and writes cleaned Parquet datasets. Produces run logs where applicable.
- **`src/02_eda.py`** — Generates descriptive analyses and exports figures and summary tables (if configured) from the cleaned datasets.
- **`src/03_model_training.py`** — Trains forecasting or demand models, writes trained artefacts, and exports evaluation metrics and diagnostic plots.
- **`outputs/figures/`** — Target directory for all generated visualisations; filenames should be stable across runs.
- **`outputs/models/`** — Persisted estimators or pipelines (e.g., `.pkl`/`.joblib`) for verification and reuse.
- **Recommended but optional**
  - **`configs/`** — YAML configurations (paths, seeds, cut-offs, model settings) to parameterise scripts.
  - **`outputs/tables/`** — Metrics and verification CSVs/Parquet files if tabular outputs are enabled.
  - **`logs/`** — JSONL run logs capturing environment details, seeds, and file hashes.

5.2 Instructions for Execution

Prerequisites. Python **3.10+** with standard build tools; write permissions for `outputs/` and `logs/`.

1. **Create and activate an isolated environment**
   - macOS/Linux:

- `python -m venv .venv`
- `source .venv/bin/activate`
- Windows (PowerShell):
- `python -m venv .venv`
- `.\.venv\Scripts\Activate.ps1`

2. **Install dependencies**
3. `pip install --upgrade pip`
4. `pip install -r requirements.txt`

   (Optional) Export a lockfile:

   ```
   pip freeze > requirements.lock
   ```

5. **Prepare inputs**
   - Place the SERL CSVs under `data/raw/` with their original filenames.
   - If using configuration files, confirm the paths and parameters in `configs/*.yaml`.
6. **Run the pipeline in order**
   - Cleaning:
   - `python src/01_data_cleaning.py   [--cfg configs/run.yaml]`
   - EDA:
   - `python src/02_eda.py             [--cfg configs/run.yaml]`
   - Modelling:
   - `python src/03_model_training.py  [--cfg configs/run.yaml]`
7. **Verify expected artefacts**
   - Cleaned datasets appear under `data/processed/`.
   - Figures are written to `outputs/figures/`.
   - Trained models and metric files are written to `outputs/models/` and (if enabled) `outputs/tables/`.
   - Run logs (if enabled) are recorded under `logs/` with the active configuration snapshot.

Determinism safeguards. Scripts set a master seed, record package versions, and hash inputs. Where external rates or APIs are used, caching and schema validation are enabled so regenerated runs match prior outputs.

6 CONCLUSION

6.1 Statement of Completeness

The repository structure, dependency manifest, and ordered execution steps documented above are sufficient to regenerate all figures, models, and evaluation artefacts used in the dissertation, assuming access to the source data. The empirical analyses rely on the **Smart Energy Research Lab (SERL) Statistical Data, 2020–2023**, obtained under safeguarded access from the UK Data Service (**Study 8963; DOI: 10.5255/UKDA-SN-8963-2**). With inputs placed as specified and the environment pinned, results are reproducible across systems consistent with the stated hardware and software prerequisites.

6.2 Contact for Clarification

For assistance with execution, configuration, or verification details, please include the run ID, the active configuration file (if used), and the exact script invocation. Contact details (name, institutional email, supervisory reference) should be listed here in line with departmental policy.