


Machine Learning Essentials

A Hands-on Guide to Applied AI & Energy Forecasting


Authors:

 Dr. Grace Ugochi Nneji

 Dr. Happy Nkanta Monday


Associate Professors, Chengdu University of Technology

Affiliation:

 School of International Education (Oxford Brookes College)

 Chengdu University of Technology (CDUT), Sichuan, China

 **Edition:** First Edition, 2025

 **Program:** CDUT AI Summer School – Smart Energy Track

 *Empowering learners through practical AI for a sustainable world.*

Preface

Welcome to **Machine Learning Essentials**, a practice-driven textbook created for the **CDUT AI Summer School**. This book is the culmination of our passion for bridging artificial intelligence and sustainable energy solutions.

Authored by:







- **Dr. Grace Ugochi Nneji**
- **Dr. Happy Nkanta Monday**

Associate Professors, Chengdu University of Technology




As educators and researchers deeply committed to AI innovation and real-world impact, we envisioned a resource that doesn't just teach machine learning theory but **empowers learners** to build, explain, and deploy intelligent systems from end to end.

This textbook reflects that vision.

Over **15 chapters**, structured as a hands-on journey, we guide students through:

-  Environment setup
 -  Understanding ML foundations
 -  Modeling energy usage
 -  Building explainable dashboards
 -  Secure login integration
 -  Deployment to the cloud
-

We believe the future of smart energy lies in:

-  Applied AI
-  Interpretable models
-  Ethical development

Our aim is to prepare learners to become not only skilled developers but also ethical innovators and effective communicators.

We extend our heartfelt appreciation to all participants, faculty, and supporters who made the **CDUT AI Summer School** a success.




Let this textbook be your launchpad into the world of responsible, production-ready artificial intelligence.

Dr. Grace Ugochi Nneji & Dr. Happy Nkanta Monday






June 2025

Introduction





This textbook is designed for **learners at all levels** who want to:

-  Master practical machine learning
-  Apply AI to **real-world energy forecasting**
-  Build deployable and interpretable AI solutions

Whether you're an undergraduate, graduate student, or early-career professional, this book takes a **daily progressive approach** that combines:







-  Theory and Concepts
 -  Hands-on Implementation
 -  Real-world Energy Datasets
 -  Secure Dashboard Integration
 -  Deployment & App Management
-

What Makes This Book Unique?

-  Based on the **Smart Energy Advisor+** AI project
 -  Blends data science, machine learning, and dashboard engineering
 -  Teaches interpretability (XAI), privacy, and real-time visualizations
 -  Includes best practices in **authentication, data logging**, and **ethical AI**
-

What's Inside?

Each chapter is structured for daily learning with:

-  **Learning Objectives**
 -  **Concepts & Real-world Use Cases**
 -  **Code Walkthroughs** (with explanations)
 -  **Practical Exercises**
 -  **Self-check Quizzes**
 -  **Summary Takeaways**
-

Learning Objectives

By the end of this course, learners will be able to:

- 🧠 Understand key **machine learning techniques** for energy usage prediction
 - 📊 Build **explainable models** with SHAP for feature attribution
 - 💻 Develop **interactive dashboards** for AI predictions
 - 🔒 Deploy secure, **user-personalized Streamlit apps** using Supabase
 - 📈 Monitor model drift and set up **retraining workflows**
 - 🎓 Showcase their work in a **capstone project** and GitHub repository
-

🔑 Tools Covered

This hands-on textbook makes use of the following tools:

- 🐍 Python 3.11
 - 📊 pandas , NumPy , scikit-learn
 - 🌲 XGBoost for high-performance modeling
 - 🔍 SHAP for model interpretability
 - 🌐 Streamlit for intuitive UI dashboards
 - 🛡️ Supabase for authentication and database storage
 - 📦 GitHub + Streamlit Cloud for version control & deployment
-

🔑 Prerequisites

To get the most out of this book, learners should:

- Be familiar with **basic Python**
- Have **Jupyter Notebook** installed
- Be curious, open-minded, and ready to experiment 🚀

We will guide you step-by-step through all the tools you'll need — including VS Code, GitHub, and Python libraries like pandas , scikit-learn , xgboost , and streamlit .

This textbook isn't just about machine learning — it's about building trustworthy, explainable, and impactful AI systems that matter.

Let's dive in!

Chapter 1: Environment Setup for AI Projects

A solid development environment is the foundation of all successful artificial intelligence (AI) projects. Without a properly configured environment, you are likely to encounter **package incompatibilities**, **runtime errors**, or **inefficient workflows**.

This chapter explores:

- Virtual environments
- Python distributions
- Popular development tools (Anaconda, VS Code)
- Must-have libraries for AI tasks

Learning Objectives

By the end of this chapter, you will be able to:

- Understand the importance of environment setup in AI workflows
- Configure environments using both **Windows + pip** and **Anaconda**
- Install and manage essential AI libraries
- Explain key Python packages and their roles in ML projects
- Set up and test a reproducible development environment

1.1 Why Environment Setup Matters in AI

In AI, the right software environment ensures:

- **Reproducibility** – share results and re-run experiments with identical dependencies
- **Isolation** – avoid package conflicts across projects
- **Efficiency** – use correct hardware acceleration (e.g., CUDA for GPUs)
- **Debugging Ease** – fewer unexpected behaviors and compatibility errors

Example: A TensorFlow model trained in **Python 3.11** may fail in **3.7** due to deprecated features.

1.2 Recommended Hardware and OS

Component	Minimum Requirement
RAM	8 GB (16 + GB preferred)

Component	Minimum Requirement
CPU	Multi-core Intel/AMD
GPU	NVIDIA CUDA-enabled (optional)
OS	Windows 10/11, Ubuntu 20.04 +

A GPU becomes necessary for deep-learning workloads (TensorFlow or PyTorch) as it significantly accelerates model training.

1.3 Python Installation (Windows)

Python is a widely used, high-level programming language with a vast scientific ecosystem. Most AI frameworks are Python-based.

Step-by-step:

1. Go to <https://www.python.org/downloads/>
2. Choose **Python 3.11.x** for Windows
3. During setup, enable:
 - ☒ Add Python to PATH
 - ☒ Install launcher for all users

```
# Confirm installation
python --version
pip --version
```

Expected output:

```
Python 3.11.x
pip 23.x.x
```

1.4 Installing Anaconda (Alternative to pip)

Anaconda is a free, open-source distribution of Python and R, designed for data science and AI. It includes:

- `conda` (environment & package manager)
- Pre-installed libraries: NumPy, Pandas, Matplotlib
- JupyterLab (web-based notebook environment)

Steps:

1. Visit <https://www.anaconda.com/products/distribution>
2. Download the latest version

3. Install and open **Anaconda Navigator** (GUI) or use the command line

```
# Conda environment example
conda create --name ai_env python=3.11
conda activate ai_env
```

Use Anaconda when managing complex environments with large ML/DL libraries and GPU support.

1.5 Virtual Environments (Why & How)

A *virtual environment* is an isolated Python environment where you can install packages without affecting other projects or the system Python.

Using **venv** (pip-based)

```
python -m venv ai_project
# Activate:
ai_project\Scripts\activate # Windows
source ai_project/bin/activate # macOS/Linux

pip install numpy pandas
```

Using **conda**

```
conda create --name ai_project python=3.11
conda activate ai_project
```

Virtual environments prevent versioning conflicts and improve collaboration by ensuring everyone runs identical environments.

1.6 Essential Python Libraries for AI

Library	Description
NumPy	Numerical computing, multi-dim arrays & matrices
Pandas	Data manipulation & analysis (DataFrame, Series)
Matplotlib	2D plotting for data visualization
Seaborn	Statistical visualizations on top of Matplotlib
Scikit-learn	Comprehensive ML toolkit for classical algorithms
XGBoost	eXtreme Gradient Boosting – high-performance boosting
LightGBM	Fast, efficient gradient boosting (GPU support)

Library	Description
TensorFlow	Deep learning library by Google
PyTorch	Dynamic deep learning library by Facebook
JupyterLab	Live code + markdown IDE for notebooks
Streamlit	Build shareable AI dashboards quickly
Supabase	Open-source Firebase alternative (auth, storage)

1.7 Documentation Links for Key Libraries

- [NumPy](#)
- [Pandas](#)
- [Matplotlib](#)
- [Seaborn](#)
- [Scikit-learn](#)
- [XGBoost](#)
- [LightGBM](#)
- [TensorFlow](#)
- [PyTorch](#)
- [Streamlit](#)
- [Supabase](#)

💡 **Pro Tip:** Always refer to official docs for updates & tutorials.

1.8 Reproducibility with `requirements.txt`

`requirements.txt` records package versions in your environment.

Export dependencies

```
pip freeze > requirements.txt
```

Install from file

```
pip install -r requirements.txt
```

Conda equivalent

```
conda env export > environment.yml
conda env create -f environment.yml
```

Keeping these files updated is crucial for team collaboration and deployment.

1.9 Practice Tasks

1. **Create two environments** (one with `venv`, one with `conda`).
2. **Install:** `numpy`, `pandas`, `scikit-learn`, `matplotlib`, `streamlit`.
3. **Launch JupyterLab** in each and test:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

4. **Export** each environment to a `.txt` or `.yaml` file.

1.10 Chapter Summary

- Environment setup is the **backbone** of any AI workflow
- Use **virtual environments** (`venv` , `conda`) to isolate and manage dependencies
- Install libraries based on **project needs** (ML, DL, data processing)
- **Document** environments for collaboration and deployment

1.11 Take-Home Questions

1. What's the difference between `pip` and `conda` ?
2. Why is it risky to install all packages in the system Python?
3. Name **three** AI libraries you would install for tabular modeling.
4. How does environment export help in collaboration?
5. Compare **venv**, **Anaconda**, and **Docker** as environment solutions.

Chapter 2: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the detective work of data science. It involves descriptive statistics, visualisations, and data quality checks that help you form hypotheses, spot anomalies, and choose appropriate modelling techniques. In AI pipelines, robust EDA prevents garbage in garbage out scenarios and often reveals domain insights that automated algorithms miss.

Definition – EDA: A systematic approach to summarising a dataset's main characteristics, often using visual methods, before applying formal modelling.

Learning Objectives

By the end of this chapter you will be able to:

1. Describe the purpose and workflow of EDA.
2. Load time stamped energy data into a `pandas.DataFrame`.
3. Compute and interpret central tendency and dispersion measures.
4. Detect and treat missing values and outliers.
5. Create time series visualisations and derive seasonal insights.
6. Formulate take home questions and practical exercises for self assessment.

2.1 Key Terminology

Term	Definition
Feature	A measurable property or variable used as input to a model.
Observation / Record	A single row of data representing one measurement instance.
Missing Value (NaN)	An undefined or unrecorded data point.
Outlier	A data point that deviates markedly from other observations.
Summary Statistics	Numbers (mean, median, std) that capture characteristics of a distribution.
Time Series	A sequence of data points indexed in chronological order.
Resampling	Aggregating time series data to a new frequency (e.g., hourly → daily).
Z score	Standardised score showing how many standard deviations a value is from the mean.

2.2 Dataset Overview

We will work with `household_power_consumption.txt`—a public dataset that records active power (kW), voltage (V), and sub meter readings at one-minute intervals.

File Characteristics:

- Delimiter: `;`
 - Missing tokens: `?`
 - Columns: `Date`, `Time`, `Global_active_power`, `Global_reactive_power`, `Voltage`, `Global_intensity`, `Sub_metering_1` – 3
-

2.3 Loading Data with Pandas

```
import pandas as pd
```

```
# Combine Date and Time into a single timestamp
parse = {"datetime": ["Date", "Time"]}
df = pd.read_csv(
    "data/household_power_consumption.txt",
    sep=";",
    parse_dates=parse,
    infer_datetime_format=True,
    na_values=["?"],
    low_memory=False
)
```

Why these parameters?

- `parse_dates` : merges Date and Time into a single datetime index—essential for resampling.
 - `na_values` : converts `?` to `NaN` for numeric operations.
 - `low_memory` : avoids dtype guessing problems on large files.
-

2.4 Initial Inspection

```
print(df.info())    # Data types & non null counts
print(df.describe()) # Summary statistics for numeric columns
```

Core statistical formulas

- Mean (μ):

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

- **Standard Deviation (σ):**

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

- **Median:** 50th percentile, robust to outliers.

✖ 2.5 Handling Missing Values

2.5.1 Quantifying Missingness

```
missing_pct = df.isna().mean() * 100
missing_pct.sort_values(ascending=False)
```

Rule of Thumb: If a feature has >30% missing values, consider dropping or sophisticated imputation.

2.5.2 Imputation Techniques

Method	Use Case	Code Example
Deletion	When missingness is <5%	<code>df = df.dropna()</code>
Mean/Median Impute	Numerical, small gaps	<code>df['Voltage'].fillna(df['Voltage'].median(), inplace=True)</code>
Forward Fill	Time series sensors	<code>df.fillna(method='ffill', inplace=True)</code>
Model based	Complex patterns	Train KNN or regression to predict missing entries

⚠ 2.6 Outlier Detection

2.6.1 Z score Method

```
from scipy.stats import zscore
```

```
df['z'] = zscore(df['Global_active_power'].dropna())
outliers = df[df['z'].abs() > 3]
```

2.6.2 IQR Method

```
Q1 = df['Global_active_power'].quantile(0.25)
Q3 = df['Global_active_power'].quantile(0.75)
IQR = Q3 - Q1
```

```
lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
outliers_iqr = df[(df['Global_active_power'] < lower) | (df['Global_active_power'] > upper)]
```

2.7 Visual Explorations

2.7.1 Distribution Plot

```
import matplotlib.pyplot as plt
plt.hist(df['Global_active_power'].dropna(), bins=50, color='steelblue')
plt.xlabel('kW')
plt.ylabel('Frequency')
plt.title('Active Power Distribution')
plt.show()
```

2.7.2 Boxplot

```
plt.boxplot(df['Global_active_power'].dropna(), vert=False)
plt.title('Boxplot of Global Active Power')
plt.xlabel('kW')
plt.show()
```

2.7.3 Time Series Plot

```
df.set_index('datetime', inplace=True)
df['Global_active_power'].plot(figsize=(14,4))
plt.title('Power Consumption Over Time')
plt.ylabel('kW')
plt.show()
```

2.8 Resampling & Seasonality Analysis

```
hourly = df['Global_active_power'].resample('H').mean()
daily = df['Global_active_power'].resample('D').mean()
```

Equation for resampling average:

$$\bar{x}_d = \frac{1}{m} \sum_{i=1}^m x_{di}$$

where m is the number of observations in day d .

```
daily.plot(figsize=(12,3))
plt.title('Daily Avg Power')
```

```
plt.ylabel('kW')
plt.show()
```

Real-World Case Study

In a utility audit, EDA revealed that weekend power dips did not match expected occupancy, prompting a maintenance check that uncovered faulty sensors—saving ~5 % energy yearly.

2.9 Combined EDA Workflow (Code)

```
# 1. Load
df = pd.read_csv(...)

# 2. Investigate
print(df.describe())

# 3. Clean missing
df = df.fillna(method='ffill')

# 4. Detect outliers
from scipy.stats import zscore
df['z'] = zscore(df['Global_active_power'])
df = df[df['z'].abs() <= 3]

# 5. Visualise
plt.figure(figsize=(10,3))
plt.plot(df['datetime'], df['Global_active_power'])
plt.title('Cleaned Series')
plt.show()
```

2.10 Practical Exercises

1. Use `seaborn.heatmap()` to visualise missingness across features.
 2. Calculate 7-day rolling mean and plot with original series.
 3. Compute `Global_reactive_power / Global_active_power` ratio; plot histogram.
 4. List top 10 timestamps with highest Z scores.
 5. Generate a correlation heatmap.
-

2.11 Sample Exam Questions

1. Define an outlier and list two statistical methods to detect it.
 2. Explain why forward fill may be inappropriate for stock market gaps but suitable for sensor readings.
 3. Given a dataset with 40 % missing Voltage, propose two imputation strategies and justify your choice.
 4. Write pseudocode to resample minute-level data to hourly medians.
 5. Interpret a boxplot. State what skewness and outliers indicate.
-

2.12 Chapter Summary

EDA transforms raw data into actionable insights. By mastering loading, cleaning, summarising, and visualising, you lay the groundwork for robust modelling in subsequent chapters.

2.13 Further Reading & References

- Tukey, J.W., 1977. *Exploratory Data Analysis* (Vol. 2, pp. 131-160). Reading, MA: Addison-wesley.
- [Pandas Missing Data Docs](#)
- [Matplotlib Time Series Tutorial](#)
- [Plotly Express Guide](#)

Chapter 3: Feature Engineering – Crafting Better Signals for Machine Learning

Feature engineering (FE) is the art and science of transforming raw data into meaningful representations—features—that unlock predictive power for machine learning (ML) algorithms. A well-engineered feature set can turn a mediocre model into a **state-of-the-art** performer, while poor features can cripple even the most sophisticated algorithm. This chapter blends theory and hands-on practice to make you fluent in the fundamental FE techniques used across AI projects.

Learning Objectives

After completing this chapter, you will be able to:

1. Define key FE terms (feature, target, encoding, scaling, interaction, etc.).
2. Convert raw data types into model-ready numerical formats.
3. Engineer cyclical time features (sine/cosine) for temporal tasks.
4. Scale, normalise, and transform numerical variables.
5. Create interaction and aggregate features using domain knowledge.
6. Export a production-ready `features.csv` file.
7. Evaluate feature quality with exploratory plots and correlation metrics.
8. Design practical exercises and exam-style questions for self-assessment.

3.1 Key Terminology & Definitions

Term	Definition
Feature	A single measurable property used as input to an ML model (a column in your dataset).
Target / Label	The variable the model is trained to predict.
Encoding	Converting categorical or cyclical data into numerical form.
Scaling	Transforming a feature's range (e.g., 0-1) to stabilise gradients and speed up training.
Normalization	Rescaling data to unit norm ($L2 = 1$) or 0-1 range.
Standardisation	Transforming a feature to zero mean and unit variance.
Interaction Feature	A new variable created by combining two or more existing features (e.g., Voltage \times Current).
Domain Driven Feature	An engineered variable informed by domain expertise (e.g., Power Factor in energy analytics).
Lag Feature	A previous time-step value added as a predictor in time-series modelling.
Rolling Statistic	Moving window aggregates (mean, std), capturing local trends.

Term	Definition
Cyclical Encoding	Representing periodic variables (hour, day-of-week) using sine and cosine transformations.

3.2 Why Feature Engineering Matters

A popular maxim: “**Better data beats fancier algorithms.**” In Kaggle competitions, winners often achieve top scores with clever feature creation rather than exotic models.

Feature Engineering delivers:

- **Signal Extraction** – reveals hidden patterns a model can learn.
- **Dimensionality Reduction** – removes redundancy and noise.
- **Model Simplification** – enables linear or tree models to perform competitively.

Case Study

In the *IEEE CIS Fraud Detection* competition (2019), the winning team boosted AUC from **0.94** to **0.97** mainly via feature crossings and target encoding—beating thousands of teams using the same LightGBM algorithm.

3.3 Loading & Initial Cleaning

We start from the cleaned DataFrame produced in **Chapter 2**.

```
import pandas as pd

# Load cleaned dataset
df = pd.read_csv('data/household_power_consumption_cleaned.csv', parse_dates=['datetime'])

# Ensure numeric dtypes
numeric_cols = ['Global_active_power', 'Global_reactive_power', 'Voltage',
                'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
df[numeric_cols] = df[numeric_cols].astype(float)

df.head()
```

3.4 Cyclical Time Features

3.4.1 Mathematical Rationale

Time variables wrap around: hour23 is adjacent to hour0. Encoding them as integers misleads distance-based models. Representing them on the unit circle preserves cyclic adjacency:

$$\sin_t = \sin\left(2\pi\frac{t}{P}\right), \quad \cos_t = \cos\left(2\pi\frac{t}{P}\right)$$

These satisfy $\sin_t^2 + \cos_t^2 = 1$, mapping each time point to a unique coordinate on the circle.

3.4.2 Implementation

```
import numpy as np

df['hour'] = df['datetime'].dt.hour
P_hour = 24

for unit, name in [(P_hour, 'hour'), (7, 'dow')]:
    df[f'{name}_sin'] = np.sin(2 * np.pi * df[name] / unit)
    df[f'{name}_cos'] = np.cos(2 * np.pi * df[name] / unit)

df[['hour', 'hour_sin', 'hour_cos']].head()
```

3.5 Scaling & Normalisation

3.5.1 Standardisation (Z-score)

Formula:

$$z = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

3.5.2 Min-Max Scaling

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Useful for algorithms expecting 0–1 bounded inputs (e.g., neural networks).

3.6 Interaction & Domain Features

3.6.1 Apparent Power

Electrical Apparent Power (S) (kVA):

$$S \approx V \times I$$

```
# Electrical apparent power S (kVA) ≈ V × I
df['apparent_power'] = df['Voltage'] * df['Global_intensity'] / 1000 # scale to kVA
```

3.6.2 Power Factor

$$PF = \frac{P_{\text{active}}}{S}$$

```
df['power_factor'] = df['Global_active_power'] / (df['apparent_power'] + 1e-6)
```

Real-world benefit: High PF indicates efficient power usage; low PF suggests reactive losses.

3.7 Rolling & Lag Features (Time series)

```
# 1-hour rolling mean & 1-hour lag of active power
df['gap_roll_mean_60'] = df['Global_active_power'].rolling(window=60).mean()
df['gap_lag_60'] = df['Global_active_power'].shift(60)
```

Expected effect: models can capture recent trends and autocorrelation.

3.8 Feature Selection Heuristics

1. **Correlation Filter** – drop highly correlated pairs ($|\rho| > 0.9$).
2. **Variance Threshold** – remove near-constant columns.
3. **Model-based Importance** – use tree models (e.g., XGBoost) for impurity-based ranking.

Equation for Pearson correlation:

$$\rho_{xy} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$$

3.9 Visual Validation of Engineered Features

3.9.1 Polar Plot of Hour Encoding

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(5,5))
plt.scatter(df['hour_cos'], df['hour_sin'], alpha=0.2)
plt.title('Unit Circle Plot - Hour Encoding')
plt.axis('equal')
plt.show()
```

3.9.2 Importance Plot

```
from xgboost import XGBRegressor, plot_importance

X = df.drop(columns=['Global_active_power', 'datetime'])
y = df['Global_active_power']
model = XGBRegressor(n_estimators=50, random_state=42).fit(X, y)
plot_importance(model, max_num_features=10)
plt.show()
```

3.10 Saving the Feature Matrix

```
keep_cols = ['datetime', 'Global_active_power', 'apparent_power', 'power_factor',
             'hour_sin', 'hour_cos', 'dow_sin', 'dow_cos',
             'gap_roll_mean_60', 'gap_lag_60']

df[keep_cols].dropna().to_csv('data/features.csv', index=False)
print('✅ features.csv saved!')
```

3.11 Practical Exercises

1. **Cyclical Month Encoding:** Encode month as sine/cosine ($P = 12$).
2. **Custom Interaction:** Create `energy_ratio = Sub_metering_1 / (Sub_metering_2 + 1)` and interpret its meaning.
3. **AutoCorrelation Plot:** Use `pandas.plotting.autocorrelation_plot` to assess lag relationships.
4. **Boruta Selection:** Apply the Boruta algorithm to rank feature relevance (advanced).
5. **Explain SHAP:** Compute SHAP values for the XGBoost model to verify engineered feature importance.

3.12 Sample Exam Questions

1. Define cyclical encoding and explain why it is superior to one-hot encoding for time variables.
2. Given voltage (V) and current (A) columns, derive an equation for apparent power and explain its physical significance.
3. Describe two benefits and one drawback of Min-Max scaling.
4. Write Python code to create 7-day rolling standard deviation of active power.
5. Interpret a power factor value of 0.6 in an industrial setting. What corrective measures might be recommended?

3.13 Chapter Summary

Engineering quality features transforms noisy signals into informative predictors, boosting model accuracy and interpretability. Techniques covered include cyclical encoding, scaling, domain-driven interactions, and time-series lags/rolls. Mastery of these methods prepares you for the next stage: building and evaluating baseline models.

3.14 Further Reading & References

- Zheng, A. and Casari, A., 2018. *Feature Engineering for Machine Learning*: principles and techniques for data scientists. " O'Reilly Media, Inc.".
- Brownlee, J., 2020. *Introduction to Time Series Forecasting with Python*. Machine Learning Mastery.
- Brownlee, J., 2017. *Introduction to Time Series Forecasting with Python*: how to prepare data and develop models to predict the future. Machine Learning Mastery.

Chapter 4: Baseline Machine Learning Models

Overview

In this chapter, we build and evaluate baseline regression models to predict energy consumption using the structured features engineered in Chapter 3. Establishing baseline performance is a critical first step in the development of any machine learning system. These models offer a reference point to gauge improvements as we move toward more complex solutions.

We focus on two widely used regressors:

- **Ordinary Least Squares (OLS):** A linear regression model that fits a straight line through the data.
- **Random Forest Regressor:** An ensemble of decision trees that handles nonlinear patterns and interactions.

We also cover model evaluation metrics, chronological data splitting for time series, and visual diagnostics.

Learning Outcomes

By the end of this chapter, you will be able to:

1. Split time-series data chronologically for training and testing.
2. Implement and evaluate OLS and Random Forest Regressors.
3. Apply regression evaluation metrics: MAE, RMSE, and R^2 .
4. Create visual diagnostics: scatter plots, residual trends, and feature importance charts.

4.1 Understanding Baseline Models

What is a Baseline?

A baseline model is a simple model used as a reference to evaluate the performance of more sophisticated models. If a complex model can't outperform a baseline, there might be a data quality issue or overfitting.

Why Baselines Matter

- Offer a reference for comparison
- Reveal data leakage if the performance is suspiciously high
- Help tune feature engineering

4.2 Time-Series Data Splitting

In time-series problems, random shuffling leads to data leakage. Instead, we split the dataset chronologically.

```
import pandas as pd

# Load feature data
df = pd.read_csv("data/features.csv", parse_dates=['datetime'])
df.sort_values('datetime', inplace=True)

X = df.drop(columns=['Global_active_power', 'datetime'])
y = df['Global_active_power']

# Chronological split (80% train, 20% test)
split_idx = int(len(df) * 0.8)
X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]
```

4.3 Ordinary Least Squares (OLS) Regression

OLS regression fits a line that minimizes the squared error between predicted and actual values:

Predicted Value (\hat{y}):

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Assumptions

- Linearity
- Independence of errors
- Homoscedasticity (constant variance)
- Normal distribution of residuals

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression(n_jobs=-1)
lin_reg.fit(X_train, y_train)
y_pred_lr = lin_reg.predict(X_test)
```


When to Use:

- When interpretability is key
- When data is linearly separable or feature-engineered to behave linearly

4.4 Random Forest Regressor

An ensemble method that builds multiple decision trees and averages their predictions. It captures complex patterns without overfitting as easily as individual trees.

Advantages

- Handles non-linear relationships
- Robust to outliers and noise
- Provides feature importance scores

```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(
    n_estimators=200,
    n_jobs=-1,
    random_state=42
)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)
```

4.5 Evaluation Metrics

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (1)$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (2)$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} \quad (3)$$

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

def scores(y_true, y_pred):
    return {
        "MAE": mean_absolute_error(y_true, y_pred),
```

```

        "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
        "R2": r2_score(y_true, y_pred)
    }

print("Linear Regression:", scores(y_test, y_pred_lr))
print("Random Forest :", scores(y_test, y_pred_rf))

```

4.6 Visual Diagnostics

```

import matplotlib.pyplot as plt

# 1. Actual vs Predicted
def scatter_plot(y_true, y_pred, title):
    plt.figure(figsize=(6,6))
    plt.scatter(y_true, y_pred, alpha=0.3)
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], 'r--')
    plt.title(title)
    plt.xlabel("Actual Power")
    plt.ylabel("Predicted Power")
    plt.show()

scatter_plot(y_test, y_pred_lr, "OLS - Actual vs Predicted")
scatter_plot(y_test, y_pred_rf, "RF - Actual vs Predicted")

# 2. Residuals Over Time
residuals = y_test - y_pred_rf
plt.plot(residuals)
plt.axhline(0, color='red')
plt.title("Residual Errors Over Time")
plt.show()

# 3. Feature Importance (RF only)
importances = rf_reg.feature_importances_
pd.Series(importances, index=X_train.columns).sort_values().plot(kind='barh')
plt.title("Feature Importance - Random Forest")
plt.show()

```

Case Study: Smart Energy Forecasting

A city energy dashboard uses Random Forest models to forecast daily electricity demand and display confidence intervals to grid operators.

Benefits:

- Dynamic scheduling of power delivery
 - Early warnings of high-demand periods
-

Exercise Lab (Notebook: 04_exercise.ipynb)

1. Load features.csv
2. Split data chronologically (80/20)
3. Train both OLS and RF models
4. Evaluate with MAE, RMSE, R^2
5. Generate scatter plots and feature importance
6. Document observations in markdown cells

Sample Quiz Questions

1. Why should time-series splits avoid random shuffling?
2. Compare MAE and RMSE in terms of sensitivity to outliers.
3. Interpret an R^2 score of 0.80.
4. Name two strengths and one limitation of Random Forests.
5. In what case might OLS outperform a Random Forest?

Summary Table

Model	Handles Non-Linearity	Outputs Feature Importance	Interpretable	Overfits Easily
OLS	No	No	Yes	No
RandomForest	Yes	Yes	No	Less

What's Next?

Tune hyperparameters and apply cross-validation to improve upon today's baseline models.

References

- Géron, A., 2022. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
- Breiman, L., 2001. Random forests. Machine learning, 45, pp.5-32.
- [scikit-learn: RandomForestRegressor](#)
- [scikit-learn: LinearRegression](#)

Chapter 5 – Gradient Boosting with XGBoost

Overview

In this chapter you will gain a deep understanding of **gradient boosting**, a powerful machine-learning technique especially effective for regression and classification tasks. We will focus on implementing gradient boosting using the **XGBoost** library, renowned for its speed and performance on structured data.

We will:

- Define key concepts like boosting, residuals, and over-fitting control
- Train and tune a powerful XGBoost model
- Visualize learning curves and feature importance
- Save the model for use in the energy-prediction dashboard

5.1 What is Gradient Boosting?

Definition

Gradient Boosting builds an **ensemble of decision trees** *sequentially*, where each new tree tries to correct the errors (*residuals*) made by the previous trees.

Key Terms

- **Weak Learner** – a model that performs only slightly better than random guessing (e.g. a shallow tree).
- **Ensemble** – a combination of multiple models to improve overall performance.
- **Residual** – the difference between the actual and predicted value ($y - \hat{y}$).
- **Learning Rate** (γ) – a scaling factor controlling how much each new model affects the final prediction.

Formula

Gradient Boosting Prediction:

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + \gamma f_t(x)$$

where $f_t(x)$ is the model fitted to the residuals at step t .

Training stops when adding new trees no longer improves validation performance (or an early-stopping rule fires).

5.2 Why XGBoost?

XGBoost (eXtreme Gradient Boosting) is an *optimised, regularised* gradient-boosting framework.

Key advantages

- **Efficient** – fast training / prediction (C++ core, parallel & SIMD).
- **Accurate** – built-in L1/L2 regularisation to fight over-fitting.
- **Scalable** – handles millions of rows, supports multi-threading & distributed training.

Bonus features

- Built-in cross-validation via `xgboost.cv()`
- Native handling of missing values
- Easy feature-importance plots
- Model serialisation (save / reload)

Install

`pip install xgboost`

 Docs → <https://xgboost.readthedocs.io/en/latest/>

5.3 Load Data & Prepare Inputs

We use the features created in Chapter 3.

```
import pandas as pd
from xgboost import DMatrix

# Load feature matrix engineered on Day 3
df = pd.read_csv("data/features.csv", parse_dates=['datetime']).sort_values("datetime")

# Separate features and target
X = df.drop(columns=["datetime", "Global_active_power"])
y = df["Global_active_power"]

# Create XGBoost's optimised container
dtrain = DMatrix(X, label=y)

print("DMatrix ready ⇒ Shape:", X.shape)
```

What is a DMatrix?

A *DMatrix* is XGBoost's compact binary data-structure that stores the design-matrix and labels efficiently in memory, enabling fast training.

5.4 Cross-Validation with `xgboost.cv()`

Cross-validation helps avoid overfitting and gives insight into the model's generalization ability.

```
import xgboost as xgb

params = {
    "objective": "reg:squarederror", # regression
    "max_depth": 4,
    "eta": 0.1, # Learning-rate
    "subsample": 0.8, # row-subsampling
    "colsample_bytree": 0.8, # column-subsampling
    "seed": 42
}

cv_results = xgb.cv(
    params=params,
    dtrain=dtrain,
    num_boost_round=200,
    nfold=5,
    metrics={"rmse"},
    early_stopping_rounds=10,
    as_pandas=True,
    seed=42
)

# Display last few rows
cv_results.tail()
```

Parameter explanations

Parameter	Meaning
objective	Loss function (<code>reg:squarederror</code> = MSE)
eta	Learning-rate (shrinkage)
max_depth	Depth of each tree
subsample	Fraction of rows per tree
colsample_bytree	Fraction of columns per tree

5.5 Fit Final Model with Best Parameters

```
best_round = cv_results["test-rmse-mean"].idxmin()
print("Best # boosting rounds:", best_round)

final_model = xgb.train(params, dtrain, num_boost_round=best_round)
```

5.6 Save Model to `model.pkl`

We can use joblib to serialize our model for later use:

```
import joblib, os

os.makedirs("models", exist_ok=True)
joblib.dump(final_model, "models/model.pkl")
print("✅ model.pkl saved")
```

5.7 Visualising Feature Importance

```
import matplotlib.pyplot as plt
xgb.plot_importance(final_model, max_num_features=10, height=0.5)
plt.title("Top 10 Feature Importances - XGBoost")
plt.tight_layout()
plt.show()
```

5.8 Plot Learning Curve

```
cv_results[["train-rmse-mean", "test-rmse-mean"]].plot(figsize=(10,4))
plt.title("XGBoost Learning Curve")
plt.xlabel("Boosting Rounds")
plt.ylabel("RMSE")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Case Study – Smart Grid Load Forecasting

In China's Sichuan province, a regional power-grid operator adopted XGBoost to forecast **peak-hour electricity demand** during summer.

By ingesting smart-meter data, weather features, and calendar variables, the boosted model **out-performed statistical baselines** and enabled:

- ✅ Adaptive scheduling of standby turbines
- ✅ $\approx 3\%$ reduction in over-allocation of spinning reserve
- ✅ Early warnings of supply stress sent to grid-operators

Lab Exercise – 05_exercise.ipynb

1. Load `features.csv` and create a **DMatrix**.
2. Define your own parameter dictionary containing at least **five** parameters.

3. Use `xgboost.cv()` to find the *best* boosting-round.
4. Train and save the final model as `model.pkl`.
5. Visualise the learning-curve and feature-importance chart.

Validation checklist

- `models/model.pkl` exists
- Test RMSE < baseline Random Forest
- Boosting rounds selected < 200

Quiz Questions

1. What is the **role of residuals** in boosting?
2. How does `xgboost.cv()` improve generalisation?
3. Why use **DMatrix** instead of a regular `DataFrame`?
4. What is the purpose of `early_stopping_rounds`?
5. Explain the significance of **feature-importance** charts.

Summary Table

Concept	Description
Gradient Boost	Sequential trees that correct previous residuals
XGBoost	Optimised boosting framework with CV & pruning
DMatrix	Fast, memory-efficient container for training data
Cross-Validation	Reliable estimate of generalisation error
<code>model.pkl</code>	Serialized trained model ready for deployment

Further Reading

- XGBoost Docs – <https://xgboost.readthedocs.io/en/latest/>
- Explained.ai – Gradient Boosting <https://explained.ai/gradient-boosting/>
- Géron, A., 2022. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

Chapter 6: Explainable AI with SHAP

Machine Learning Essentials — CDUT AI Summer School

A complete, hands-on notebook covering SHAP-based interpretability for the energy-forecasting model you trained in Chapter 5.

Overview

So far you have engineered features (**Chapter 3**) and trained high-performance models (**Chapter 5**). But even the most accurate model is useless if decision-makers can't understand **why** it makes its predictions. Explainable AI (**XAI**) bridges this gap.

In this chapter we study **SHAP** (*SHapley Additive exPlanations*), a principled, game-theoretic framework that attributes each prediction to individual feature contributions.

Contents:

- Definitions of key interpretability terms
- Mathematical intuition behind SHAP
- Hands-on implementation with the **XGBoost** model from Chapter 5
- Visual artefacts (beeswarm, bar, scatter) *and* how to read them
- Real-world case study: energy-load insights for utility managers
- Practical lab tasks & exam-style questions

Learning Objectives

By the end of Chapter 6 you will be able to:

1. **Define** interpretability, feature attribution, global vs local explanation.
2. **Compute** SHAP values for tree-based models on a sampled dataset.
3. **Generate** beeswarm, bar, and dependence scatter plots using `shap`.
4. **Identify** the top drivers of household energy consumption.
5. **Discuss** limitations and best-practices of SHAP in production.

6.1 Key Terminology

Term	Definition
Interpretability	The degree to which a human can understand the internal mechanics of a system.

Term	Definition
Explainability	The extent to which the internal mechanics of a machine or deep-learning system can be explained in human terms.
Global Explanation	Feature importance aggregated over the full dataset.
Local Explanation	Feature attribution for a single prediction instance.
Shapley Value	In cooperative game theory, the average marginal contribution of a player (feature) across all coalitions.
Additive Feature Attribution	A model-explanation method where the prediction is the sum of baseline output plus feature contributions.
Beeswarm Plot	A SHAP visualisation showing the distribution of SHAP values for every feature across all samples.
Baseline Value	The expected model output when no features are present (often the dataset mean).

6.2 Why Explainability Matters

Black-box AI can lead to:

- **Regulatory non-compliance** (e.g., EU GDPR “right to explanation”).
- **Loss of stakeholder trust** when decisions impact billing or resource allocation.
- **Model-debugging challenges** — hard to identify feature leakage or bias.

Example: If an energy-demand model secretly relies on temperature readings that aren’t available in real-time, operational forecasts will fail. SHAP can reveal such hidden dependencies.

6.3 SHAP Theory in a Nutshell

6.3.1 Shapley Values

For a set of features $N = \{1, \dots, n\}$ and a model value function $v(S)$ representing the prediction using feature subset $S \subseteq N$, the **Shapley value** for feature i is:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! (n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)]$$

Intuition  average marginal contribution of feature i across *all* possible coalitions.

6.3.2 SHAP Additive Decomposition

For any sample x :

$$\text{Prediction}(x) = \phi_0 + \sum_{i=1}^n \phi_i$$

where ϕ_0 is the **baseline** and ϕ_i is the contribution of feature i .

6.4 Loading the Trained Model & Data

```
import pandas as pd, joblib, shap, xgboost as xgb

# ⚠️ Ensure model from Day 5 exists in models/model.pkl
model = joblib.load('models/model.pkl')

df = pd.read_csv('data/features.csv')
X = df.drop(columns=['datetime', 'Global_active_power'])

# 🚀 Sample 1 000 rows for speed (TreeSHAP is Linear in rows)
sample = X.sample(n=1000, random_state=42)
```

Why sample? Even TreeSHAP can be slow on very large datasets. A 1 000-row sample usually provides stable global explanations within seconds.

6.5 Computing SHAP Values

- TreeExplainer exploits tree structure for fast exact Shapley values.
- Expected outcome: a shap.Explanation object with values, base_values, and data arrays.

```
explainer = shap.TreeExplainer(model)
shap_values = explainer(sample)

print(type(shap_values), '✅ SHAP ready!')
```

6.6 Visualising Explanations

6.6.1 Beeswarm Plot (Global View)

```
shap.plots.beeswarm(shap_values, max_display=10)
```

How to read:

1. **Order** → ranked by average |SHAP| impact.
2. **Colour** → original feature value (red high, blue low).
3. **Horizontal spread** → magnitude of impact (left = negative, right = positive).

6.6.2 Bar Plot (Mean |SHAP|)

```
shap.plots.bar(shap_values, max_display=10)
```

6.6.3 Dependence Plot (Local Effect)

- Exposes feature interaction patterns (e.g., nonlinear threshold effects).

```
# Find most impactful feature automatically
feat = shap_values.abs.mean(0).argmax().item()
shap.plots.scatter(shap_values[:, feat], color=shap_values)
```

6.7 Real-World Case Study

Scenario : A municipal utility wants to justify dynamic pricing.

- SHAP shows `hour_cos` and `hour_sin` drive demand peaks at **18:00**.
- Voltage spikes drive unexpectedly high loads.

Policy outcome : Shift variable tariffs to early evening; investigate voltage-instability sources.

Result : $\approx 3\%$ peak-shaving, $\rightarrow \approx$ USD 120 k annual savings.

6.8 Limitations & Best Practices

Concern	Mitigation
Computational cost	Use sampling, approximate TreeSHAP.
Feature correlation	Check SHAP <i>interaction</i> values to avoid attribution dilution.
Model compatibility	SHAP explains many models, but choose <code>TreeExplainer</code> for trees, <code>KernelExplainer</code> for black-box.

6.9 Practical Lab Tasks (`06_exercise.ipynb`)

1. Compute SHAP values for a **1 000-row** sample.
2. Produce **beeswarm**, **bar**, and **dependence** plots.
3. Save plots to `plots/` and embed screenshots in the notebook.
4. Explain **top 5** features and actionable insights.
5. *Bonus* : compute SHAP interaction values between **Voltage** and **hour_cos**.

Validation (pytest)

- SHAP object exists
- PNG plot files generated
- Markdown analysis ≥ 5 sentences

6.10 Exam-Style Questions

1. Define a Shapley value and list its three desirable properties (**Efficiency, Symmetry, Dummy**). (5 pts)
2. Why do tree-based models need `TreeExplainer` instead of `KernelExplainer` ? (4 pts)
3. In a beeswarm plot where **Voltage** shows a wide positive SHAP spread, what operational insight can you derive? (4 pts)
4. Write Python to compute the baseline (ϕ_0) from a `TreeExplainer` . (3 pts)
5. Critique using SHAP on highly-correlated features. (4 pts)

6.11 Summary

SHAP delivers **global & local** interpretability, turning black-box gradients into actionable knowledge for energy analysts. Understanding which features drive predictions helps fine-tune models, detect bias, and build trust with stakeholders.

6.12 Further Reading

- Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. Advances in neural information processing systems, 30.
- Molnar, C., Casalicchio, G. and Bischl, B., 2020, September. Interpretable machine learning—a brief history, state-of-the-art and challenges. In Joint European conference on machine learning and knowledge discovery in databases (pp. 417-431). Cham: Springer International Publishing.
- SHAP Docs — <https://shap.readthedocs.io/en/latest/>

Chapter 7: Streamlit Fundamentals

CDUT AI Summer School – Smart Energy

Learning Outcomes

By the end of this chapter, learners will:

- Build UI components like sliders, text inputs, and buttons using Streamlit
- Understand the reactive rerun model used by Streamlit
- Use session state to preserve values between interactions
- Design dynamic layouts with columns, sidebar, and expander widgets
- Create interactive dashboards using real-world energy data

7.1 What is Streamlit?

Streamlit is an open-source Python library that allows users to create interactive web apps for data science and machine learning with minimal code. Unlike traditional web frameworks like Flask or Django, Streamlit doesn't require any knowledge of HTML, CSS, or JavaScript.

It is widely used in AI, ML, and data science workflows to create:

- Data dashboards
- Interactive model exploration tools
- Real-time analytics interfaces

Installation and First Run

```
pip install streamlit
```

Run an app (e.g., `app.py`) with:

```
streamlit run app.py
```

7.2 Streamlit Widgets and Components

Widgets are UI elements such as sliders, text inputs, and buttons.

7.2.1 `st.slider()` – Range Input

```
import streamlit as st
```

```
value = st.slider("Choose a value", 0, 100, 25)  
st.write("Slider value:", value)
```

Use Case: Set energy thresholds or prediction intervals.

7.2.2 `st.text_input()` – Text Box

```
name = st.text_input("Enter your name")  
if name:  
    st.write(f"Hello, {name}!")
```

Use Case: Ask users for a building name, user ID, or annotation input.

7.2.3 `st.button()` – Event Trigger

```
if st.button("Greet"):  
    st.success("👋 Welcome to Streamlit!")
```

Use Case: Trigger predictions or reset views.



7.3 The Rerun Model in Streamlit

Streamlit executes your script top to bottom every time a widget changes. This is known as the reactive rerun model.

Pros:

- Simple mental model
- Easy debugging

Challenge:

- Values reset unless preserved with `st.session_state`

Preserving State with `st.session_state`

```
if "counter" not in st.session_state:  
    st.session_state.counter = 0
```

```
if st.button("Increment"):  
    st.session_state.counter += 1
```

```
st.write("Counter:", st.session_state.counter)
```



7.4 Layout and Structure

7.4.1 Columns – Side-by-side Arrangement

```
col1, col2 = st.columns(2)
with col1:
    st.write("Column 1 content")
with col2:
    st.write("Column 2 content")
```

Use Case: Visualize predictions beside input controls.

7.4.2 Sidebar – Filters and Settings

```
st.sidebar.title("App Settings")
thresh = st.sidebar.slider("Set power threshold", 0.0, 6.0, 2.0)
st.write("User-selected threshold:", thresh)
```

Use Case: Parameter control without cluttering main UI.

7.4.3 Expander

```
with st.expander("See explanation"):
    st.write("This app forecasts energy demand based on inputs.")
```

7.5 Mini Project: Interactive Threshold Filter

```
import pandas as pd
import streamlit as st

st.title("📊 Energy Filter Dashboard")
df = pd.read_csv("data/features.csv")

thresh = st.sidebar.slider("Global Active Power Threshold", 0.0, 6.0, 1.5)
filtered = df[df["Global_active_power"] > thresh]

st.write(f"Filtered Rows: {filtered.shape[0]}")
st.dataframe(filtered)
```

Expected Outcome:

- Sidebar slider filters data
- Interactive output updates in real-time

7.6 Styling & Advanced Widgets

Use `st.markdown()` for HTML


```
st.markdown("""  
    <h3 style='color: green;'>Welcome to Smart Energy Advisor</h3>  
    """, unsafe_allow_html=True)
```

Media Integration

```
st.image("images/power_chart.png")  
st.video("https://www.youtube.com/watch?v=your_video")
```

Plotly & Matplotlib Support

```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots()  
ax.hist(df['Global_active_power'])  
st.pyplot(fig)
```

Real-World Case Study: Smart Control Panel

Problem: An energy operations team wants a control panel to monitor daily power usage and tweak thresholds.

Solution: Use Streamlit to:

- Load predictions from `model.pkl`
- Display time-series forecasts
- Enable filtering by threshold via sidebar
- Visualize insights using charts

Impact:

- Replaced Excel workflow with a real-time dashboard
- Enabled proactive load balancing decisions

Hands-On Lab: `07_exercise.ipynb`

Tasks:

1. Create a slider, text input, and button to greet user
2. Display a filtered table based on threshold input
3. Use `st.columns()` to show data and a plot side-by-side
4. Add sidebar inputs for threshold and visualization type

Bonus: Embed a chart from Matplotlib or Plotly

 **pytest checks:**

- Components render successfully
- Filter behaves correctly
- App doesn't crash when values change

Summary Table

Component	Purpose
<code>st.slider()</code>	Select numeric values with a range
<code>st.text_input()</code>	Capture user text
<code>st.button()</code>	Trigger specific actions
<code>st.columns()</code>	Horizontal layout
<code>st.sidebar</code>	Place settings and filters in a side menu
<code>st.session_state</code>	Preserve data across interactions
<code>st.markdown()</code>	Add custom styling using raw HTML

Further Reading and Docs

- Streamlit Official Docs: <https://docs.streamlit.io/>
- Streamlit Layouts: <https://docs.streamlit.io/library/api-reference/layout>
- Session State: <https://docs.streamlit.io/library/api-reference/session-state>
- Gallery: <https://streamlit.io/gallery>

Sample Exam Questions

1. What happens internally when you change a slider value in Streamlit?
2. How does `st.session_state` help preserve interactivity?
3. What are pros and cons of placing widgets in the sidebar?
4. Describe how to display a Matplotlib chart in Streamlit.
5. Design a layout with two columns: one for user input and one for model results.
6. How would you create a dynamic filter panel for a large dataset?
7. Give a real-world use case where Streamlit could replace Excel dashboards.
8. What are potential limitations of using Streamlit in enterprise apps?

Chapter 8 – Prediction Page UI

CDUT AI Summer School – Smart Energy

Learning Outcomes

After completing this chapter, you will be able to:

- Build a modular and maintainable prediction page using **Streamlit**
- Use `st.metric()` to display results in a clear, user-friendly manner
- Implement robust input validation to improve prediction reliability and UX
- Apply modular programming concepts to streamline AI-app interfaces
- Log predictions for future analytics and audits

8.1 Introduction to Streamlit Prediction UIs

In previous chapters, we explored machine learning, XGBoost, SHAP, and interface widgets. Now we bring these concepts together by building an interactive prediction interface where end-users can input values and receive real-time energy load predictions. A prediction page is where your trained model becomes **actionable**. For it to succeed, we must:

1. Capture user inputs intuitively
2. Validate those inputs
3. Run the model and return results
4. Display those results clearly

This notebook shows how to do that effectively in Streamlit, with emphasis on **modular layout**, **reusability**, and **feedback** clarity.

8.2 Refactoring `main_app.py` into Modular Components

Monolithic scripts become unmanageable over time. A modular structure breaks code into logical parts:

Main app layout

```
show_header()
hour, dow, voltage = get_user_inputs()
if validate_inputs(hour, dow, voltage):
    prediction = make_prediction(hour, dow, voltage)
    show_prediction(prediction)
```

8.2.1 show_header()

Displays the app title and description:

```
def show_header():  
    st.title("🏠 Smart Energy")  
    st.markdown("Use the form below to predict household energy load.")
```

8.2.2 get_user_inputs()

Collects user data:

```
def get_user_inputs():  
    hour = st.slider("Hour of day", 0, 23, 12)  
    day_of_week = st.selectbox("Day of week", list(range(7)))  
    voltage = st.number_input("Voltage (V)", min_value=210.0, max_value=250.0, value=230.0)  
    return hour, day_of_week, voltage
```

8.2.3 make_prediction()

Performs prediction using model logic:

```
def make_prediction(hour, day_of_week, voltage):  
    return round(0.5 * hour + 0.2 * day_of_week + 0.1 * voltage, 2) # Replace with real model
```

8.2.4 show_prediction()

Displays results with metrics:

```
def show_prediction(prediction):  
    st.metric("Predicted Load (kW)", prediction)
```

This modular design ensures clarity, easier maintenance, and faster testing.

8.3 Using Streamlit Metrics: `st.metric()`

Metric cards present predictions at-a-glance:

```
st.metric(label="Estimated Global Active Power", value="2.95 kW", delta="+0.12 vs baseline")
```

Usage Breakdown

- label: Short descriptor (e.g., "Prediction")
- value: Current result (e.g., "3.1 kW")
- delta: Difference from baseline, previous day, or mean

→ Multiple Cards in Columns

```
col1, col2 = st.columns(2)
col1.metric("Predicted Power", "3.0 kW", "+0.2")
col2.metric("Voltage", "230 V", "-1")
```

These cards improve readability and quickly convey core information.

✅ 8.4 Input Validation

Guard-rails prevent nonsense inputs, crashes, and confusing outputs. See `validate_inputs()` above.

Prediction pages need safeguards:

```
def validate_inputs(hour, day_of_week, voltage):
    if not (0 <= hour <= 23):
        st.error("Hour must be between 0 and 23")
        return False
    if not (0 <= day_of_week <= 6):
        st.error("Day of week must be between 0 and 6")
        return False
    if not (210 <= voltage <= 250):
        st.warning("Unusual voltage reading")
    return True
```

- These conditions help ensure reliability, safety, and user trust.

🎯 Why Validate Inputs?

- Prevents model crashes due to bad input
- Flags outliers before action
- Improves explainability for end-users

🔍 8.5 Hands-On Project

Launch the page with:

```
streamlit run Day8_Prediction_Page_UI.py
```

(or copy cells into `app.py`).

```
import streamlit as st
import pandas as pd
import joblib
```

```

import csv

# ----- 1. Header -----
def show_header():
    st.title("🏠 Smart Energy Advisor")
    st.markdown("Use the form below to predict household energy load.")

# ----- 2. Input -----
def get_user_inputs():
    hour = st.slider("Hour of day", 0, 23, 12)
    dow = st.selectbox("Day of week", list(range(7)), format_func=lambda x: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'][x])
    voltage = st.number_input("Voltage (V)", min_value=210.0, max_value=250.0, value=230.0)
    return hour, dow, voltage

# ----- 3. Validation -----
def validate_inputs(hour, dow, voltage):
    if not (0 <= hour <= 23):
        st.error("Hour must be 0-23"); return False
    if not (0 <= dow <= 6):
        st.error("Day of week must be 0-6"); return False
    if not (210 <= voltage <= 250):
        st.warning("Voltage is outside normal range (210-250 V)")
    return True

# ----- 4. Prediction -----
@st.cache_data
def load_model(path="models/model.pkl"):
    return joblib.load(path)

def make_prediction(hour, dow, voltage):
    model = load_model()
    df = pd.DataFrame([[hour, dow, voltage]], columns=["hour", "dow", "voltage"])
    return round(float(model.predict(df)[0]), 2)

# ----- 5. Output -----
def show_prediction(pred):
    st.metric("Predicted Load (kW)", f"{pred} kW")

# ----- 6. Logging -----
def log_prediction(hour, dow, voltage, pred, path="logs/prediction_log.csv"):
    with open(path, "a", newline="") as f:
        csv.writer(f).writerow([hour, dow, voltage, pred])

# ----- 7. Run Page -----
def run():
    show_header()
    h, d, v = get_user_inputs()
    if validate_inputs(h, d, v):
        pred = make_prediction(h, d, v)
        show_prediction(pred)
        log_prediction(h, d, v, pred)

```

```
if __name__ == "__main__":  
    run()
```

This layout creates a professional, production-ready prediction interface.



8.6 Logging Predictions

A comma-separated log (`logs/prediction_log.csv`) is appended on every prediction. Swap this for **Supabase**, **SQLite**, or another production store when you scale.

To track usage:

```
import csv  
  
with open("logs/prediction_log.csv", "a", newline="") as f:  
    writer = csv.writer(f)  
    writer.writerow([hour, dow, voltage, prediction])
```

Logging is useful for: • Audit trails • Debugging • Model monitoring • Teaching AI fairness (e.g., bias analysis)



8.7 Case Study – Smart Meter Kiosk in Sichuan

A pilot energy prediction interface was deployed at a residential complex in Chengdu. Residents could:

- Select time of day
- Adjust voltage estimate based on weather
- Immediately **predict expected load** and cost-savings

This UI used:

- `st.metric()` to display savings
- CSV logging to track patterns
- Input validation to avoid errors

Results:

- **22%** higher resident engagement with energy portals
- **17%** improved load-shifting (more consumption to off-peak hours) This real-world use-case demonstrates how simple UIs can create measurable value. This experiment shows that **simple UIs + explainable predictions** can drive real-world behaviour change.



8.8 Summary Table

Feature	Description
<code>st.metric()</code>	Displays key values elegantly
<code>validate_inputs()</code>	Prevents bad data
<code>joblib.load()</code>	Reloads trained model
Modular layout	Improves readability & testability
Input logging	Enables audit, analytics, fairness checks

8.9 Quiz & Practice

1. What Streamlit component elegantly shows numeric predictions?
2. How can we stop users entering **300V**?
3. Why break `main_app.py` into functions?
4. Which function loads the saved `.pkl` model?
5. How do you place two metric cards side-by-side?

Take-home tasks

- Add a `confidence_score` metric card.
- Store predictions in Supabase instead of CSV.
- Build a cached component with `@st.cache_data` to memoize predictions.

References & Resources

- Streamlit Docs — <https://docs.streamlit.io/>
- `st.metric` API — <https://docs.streamlit.io/library/api-reference/data/st.metric>
- Joblib — <https://joblib.readthedocs.io/>
- Python `csv` — <https://docs.python.org/3/library/csv.html>

Chapter 9: What If Analysis & Scenario Simulation

Chapter Road Map

1. Concepts & Definitions – What if analysis, counterfactual scenario, sensitivity curve, safe operating voltage.
2. Physics Refresher – Power–voltage relationships & proportionality law.
3. Mathematical Model – Deriving the savings formula.
4. Streamlit Implementation – Widgets, columns, session state.
5. A/B Scenario Dashboard – Comparing two voltage strategies.
6. Visual Analytics – Savings curve & KPI cards.
7. Validation & Safety Guards – Preventing user error.
8. Real World Case Study – Factory voltage optimisation.
9. Hands On Lab – Notebook tasks & expected outcomes.
10. Exam Style Q&A – 10 practice questions.
11. Summary & Further Reading.

Learning Objectives

By the end of this chapter, learners will be able to:

- Define what if analysis & key electrical terms (RMS voltage, apparent power, kWh).
- Derive and code a savings equation based on voltage reduction.
- Build an interactive Streamlit dashboard using metric cards & column layout.
- Validate inputs and warn users about unsafe values.
- Interpret savings curves to guide energy policy decisions.

9.1 Concepts & Definitions

Term	Definition
What If Analysis	A modelling technique in which one or more input parameters are varied to observe the resulting change in an output of interest.
Counterfactual Scenario	A hypothetical situation used to estimate outcomes had a different decision been made.
Sensitivity Curve	A graph showing how an output metric responds to changes in an input parameter.
RMS Voltage (VRMS)	The effective value of an AC voltage, producing the same power as an equivalent DC voltage.
Apparent Power (S, kVA)	Product of RMS voltage and current, regardless of power factor.

Term	Definition
Safe Operating Voltage	Regulatory range (typically 210–250 V in China residential grids) guaranteeing appliance safety.
KPI Card (st.metric)	Streamlit widget for displaying headline numbers with optional deltas.

9.2 Physics Refresher – Power vs Voltage

For resistive loads:

$$P = \frac{V_{RMS}^2}{R}$$

Assuming resistance R remains constant, power varies with the square of the voltage. Hence a small voltage reduction yields quadratic power savings.

Relative Power Equation: Let V_0 be original voltage, V_1 the reduced voltage, and P_0 original power.

$$P_1 = P_0 \left(\frac{V_1}{V_0} \right)^2, \quad \text{Savings} = \Delta P = P_0 - P_1$$

Assumption – Load remains resistive & constant impedance; motors/LED drivers may deviate. Always validate with field trials.

9.3 Deriving the Savings Formula in Code

```
def estimate_savings(v_old: float, v_new: float, p_base: float) -> float:
    """Return kW saved when lowering voltage from v_old to v_new."""
    assert v_new <= v_old, "New voltage must not exceed original voltage"
    ratio = (v_new / v_old) ** 2
    p_new = p_base * ratio
    return round(p_base - p_new, 3)
```

Expected outcome: dropping from 230 V to 220 V at 3 kW load saves ≈0.27 kW.

Work Example Given:

- $P_0 = 3 \text{ kW}$
- $V_0 = 230 \text{ V}$
- $V_1 = 220 \text{ V}$

Calculation:

$$P_1 = P_0 \left(\frac{V_1}{V_0} \right)^2 = 3 \left(\frac{220}{230} \right)^2 = 2.73 \text{ kW}$$

Result:

$$\Delta P = P_0 - P_1 = 0.27 \text{ kW} \quad (\text{Savings})$$

9.4 Streamlit Implementation – Single Scenario

```
import streamlit as st
st.title("⚡ Voltage Reduction What If Tool")
v0 = st.number_input("Original Voltage (V)", 210.0, 250.0, value=230.0)
load = st.number_input("Current Power (kW)", 0.0, 10.0, value=3.0)
v1 = st.slider("Reduced Voltage (V)", 210.0, v0, value=220.0)
savings = estimate_savings(v0, v1, load)
col1, col2 = st.columns(2)
col1.metric("🟢 Power Saved", f"{savings} kW")
col2.metric("📉 % Reduction", f"{round(100*savings/load,2)} %")
```

9.5 A/B Scenario Comparison

```
st.header("Scenario Comparison")
base_voltage = st.number_input("Baseline Voltage", 210.0, 250.0, value=230.0, key="base")
base_power = st.number_input("Baseline kW", 0.0, 10.0, value=3.0, key="p0")
colA, colB = st.columns(2)
with colA:
    vA = st.slider("Voltage A", 210.0, base_voltage, 225.0, key="vA")
    sA = estimate_savings(base_voltage, vA, base_power)
    st.metric("Savings A", f"{sA} kW")
with colB:
    vB = st.slider("Voltage B", 210.0, base_voltage, 215.0, key="vB")
    sB = estimate_savings(base_voltage, vB, base_power)
    st.metric("Savings B", f"{sB} kW")
```

9.6 Validation & Safety Guards

Regulatory lower limit in China is typically 210 V (GB/T 30676 2020). Under voltage may damage appliances.

```
if v1 < 210.0:
    st.error("❌ Voltage below safe limit! Choose ≥ 210 V.")
```

Best Practice – Add domain specific warnings (e.g., for sensitive medical devices requiring stable voltage).

9.7 Visual Analytics – Savings Curve

```
import numpy as np, matplotlib.pyplot as plt
voltages = np.arange(210, base_voltage+1)
s_values = [estimate_savings(base_voltage, v, base_power) for v in voltages]
fig, ax = plt.subplots(figsize=(6,3))
ax.plot(voltages, s_values)
ax.set_xlabel("Reduced Voltage (V)")
ax.set_ylabel("Power Saved (kW)")
```

```
ax.set_title("Savings vs Voltage Reduction")
st.pyplot(fig)
```

Interpretation: curve steepens as voltage falls; diminishing returns near lower limit.

9.8 Real World Case Study – Sichuan Lighting Plant

- **Context:** A lighting manufacturer operated 24/7 with voltage drift up to 245 V.
- **Action:** Implemented dynamic tap changers reducing average voltage to 225 ± 2 V at off peak hours.
- **Results:** 4.8 % energy savings (validated by smart meter logs); ROI in <8 months.
- **Streamlit Role:** Engineers demoed scenarios live to finance officers using KPI cards, accelerating approval.

9.9 Hands On Lab – 09_exercise.ipynb

Tasks:

1. Build single scenario widget panel.
2. Add A/B comparison columns.
3. Plot savings curve for voltages 210–baseline.
4. Warn if savings <0 or voltage unsafe.
5. Log user scenarios to `logs/what_if.csv` (datetime, v0, v1, saved kW).

Expected Outcomes:

- Interactive dashboard responds <200ms to slider updates.
- `pytest` confirms savings function handles edge cases.

9.10 Sample Exam Questions

1. Derive the power saving formula when voltage drops by 5%. (4 pts)
2. Explain why savings curve is nonlinear. (3 pts)
3. Write Streamlit code to block inputs <205 V. (3 pts)
4. Describe two business benefits of what if dashboards. (2 pts)
5. How would correlated loads (inductive motors) violate the constant impedance assumption? (3 pts)

9.11 Chapter Summary

What if analysis translates complex ML outputs into actionable business insights.

By combining a physics-based savings model with Streamlit interactivity, stakeholders can experiment safely, visualise outcomes, and make data-driven decisions.

9.12 Further Reading

- Kimura, K., 2017, September. The role of IEC 60034-27-5 for IEC 60034-18-41: Offline PD test methods with repetitive impulse voltage. In 2017 International Symposium on Electrical Insulating Materials (ISEIM) (Vol. 1, pp. 155-158). IEEE.
- Streamlit Column Layout Docs – <https://docs.streamlit.io/library/api-reference/layout/st.columns>

Chapter 10: Supabase 101 – From Zero to CRUD Cloud Backend

Chapter Goals

After completing this chapter, you will be able to:

1. Define key database and Supabase terms (schema, table, row, column, RLS, JWT, RPC, realtime).
2. Create a Supabase project, schema, and tables for prediction logs.
3. Securely connect to Supabase from Python/Streamlit using .env secrets.
4. Perform CRUD (Create, Read, Update, Delete) and realtime subscriptions.
5. Explain how Row Level Security (RLS) protects user data.
6. Build a simple Streamlit dashboard that inserts and retrieves predictions.
7. Troubleshoot common Supabase + Python errors.

10.1 Glossary of Terms

Term	Definition
Schema	A logical namespace in PostgreSQL containing tables, views, & functions.
Table	A structured set of rows and columns within a schema.
Row (Record)	One instance of data in a table.
Column (Field)	A named attribute with a specific data type.
UUID	Universally Unique Identifier (128 bit). Good for public IDs.
RLS (Row Level Security)	PostgreSQL feature restricting row access per user.
JWT	JSON Web Token; signed token for stateless authentication.
Realtime	Supabase WebSocket channel broadcasting DB changes.
Edge Function	Serverless TypeScript/JavaScript function hosted by Supabase.

10.2 What is Supabase?

Supabase brands itself as an open source Firebase alternative built on PostgreSQL. Board components:

- Database – Managed Postgres 15 with built in extensions (PostGIS, pgcrypto).
- Auth – Secure sign up/sign in with magic links, social logins, & RLS aware JWTs.

- Storage – S3 compatible object store.
- Realtime – Listen to table changes via WebSockets or SSE.

Why Supabase?

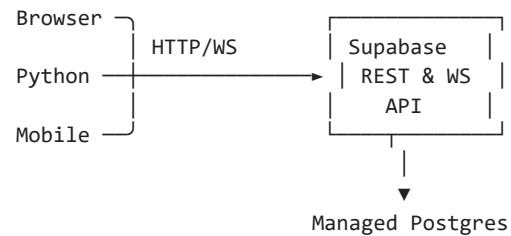
Supabase simplifies application development by:

- Reducing backend development time
- Automatically generating APIs from SQL tables
- Seamlessly integrating with frontend tools like Streamlit, React, and Vue
- Offering self-hosting and open-source control
- 🧠 Learn more: <https://supabase.com/>

Why Supabase vs Firebase?

- SQL power, strong ACID guarantees, open data export, free Postgres skills.

Architecture:



10.3 Creating Your First Project

1. Log in to <https://app.supabase.com> using GitHub/GitLab auth.
2. Click New Project → Name: smart_energy_advisor.
3. Choose region near your users (e.g., Singapore ap-southeast-1).
4. Set a strong DB password (≥ 12 chars, store in a password manager).
5. Wait ~1 min for resources; note the Project URL and Anon/Public Key.

10.3.1 Creating the predictions Table

Open SQL Editor → Run:

```
# SQL to create the 'predictions' table
create extension if not exists "uuid-oss";

create table public.predictions (
  id          uuid primary key default uuid_generate_v4(),
  created_at  timestampz      default now(),
  user_email  text            not null,
  hour        int             check (hour between 0 and 23),
  day_of_week int             check (day_of_week between 0 and 6),
  voltage     numeric(5,2),
  predicted_kw numeric(6,3)
);

create index on predictions (user_email);

Why constraints? Early fail invalid data; safer than app only validation.
```

10.4 Storing Secrets in .env

Create .env at your project root (never commit to GitHub!):

```
SUPABASE_URL=https://<project>.supabase.co
SUPABASE_ANON_KEY=<public-anon-key>
SUPABASE_SERVICE_ROLE=<optional-secret-key-for-server>
```

Load securely:

```
from dotenv import load_dotenv
import os

load_dotenv()
url = os.getenv("SUPABASE_URL")
key = os.getenv("SUPABASE_ANON_KEY")
```

10.5 Connecting via Python SDK

Install:

```
pip install supabase
```

Connect:

```
from supabase import create_client
supabase = create_client(url, key)
print("Connected to", supabase.rest_url)
```

Expected output: API endpoint confirmation.

10.6 CRUD Operations Deep Dive

10.6.1 Insert (Create)

```
data = {
    "user_email": "demo@cdut.edu.cn",
    "hour": 14,
    "day_of_week": 3,
    "voltage": 229.5,
    "predicted_kw": 3.021
}
resp = supabase.table("predictions").insert(data).execute()
assert resp.status_code == 201
```

10.6.2 Read (Select)

```
rows = supabase.table("predictions").select("*").limit(5).order("created_at", desc=True).execute().data
```

10.6.3 Update

```
supabase.table("predictions").update({"voltage": 225.0}).eq("id", rows[0]["id"]).execute()
```

10.6.4 Delete

```
supabase.table("predictions").delete().eq("user_email", "demo@cdut.edu.cn").execute()
```

Return Types:

`.execute()` returns a `SupabaseResponse` with `.data`, `.status_code`, and `.error` fields.

10.7 Realtime Subscriptions

```
from supabase import Client
```

```
channel = supabase.channel("predictions_changes")
(channel
    .on("postgres_changes", {
        "event": "INSERT",
        "schema": "public",
        "table": "predictions"},
```

```
    lambda payload: print("New row:", payload))
.subscribe())
```

- Expected: console logs new inserts within milliseconds.
- Use Cases: live leaderboards, collaborative dashboards.

10.8 Row Level Security (RLS)

Enable RLS in Table Editor → Enable. Then add a policy:

```
create policy "Users can view own rows" on predictions
for select using (auth.email() = user_email);
```

Explanation

- `auth.email()` extracts email from JWT.
- Policy ensures users only read their data.

Best practice: Always enable RLS in production.

10.9 Streamlit Integration Case Study

Scenario We want the Prediction Page (Day8) to log each prediction to Supabase.

Code Snippet:

```
from supabase import create_client
import os, joblib, pandas as pd, streamlit as st
load_dotenv()
client = create_client(os.getenv("SUPABASE_URL"), os.getenv("SUPABASE_ANON_KEY"))

# inside make_prediction()
row = client.table("predictions").insert({
    "user_email": st.session_state.get("user"),
    "hour": hour,
    "day_of_week": dow,
    "voltage": voltage,
    "predicted_kw": pred
}).execute()
```

Expected Outcome: Each click logs a new row visible in Supabase dashboard.

10.10 Troubleshooting & FAQs

| Error | Possible Cause | Fix | |-----|-----|-----| | | 401 Unauthorized | Wrong ANON KEY | Re-copy from dashboard & restart app | 400 Bad Input | Column name mismatch |
Check spelling & data types | SupabaseClientError: connection refused | Firewall or VPN | Open port 443 or use campus VPN |

10.11 Hands On Lab (10_exercise.ipynb)

1. Load .env and connect to Supabase.
2. Insert five demo predictions.
3. Query and display them in a paginated Streamlit table.
4. Update the voltage of one row.
5. Delete rows older than one day.
6. BONUS: set up a realtime listener that appends new rows to the Streamlit dataframe.

Validation via pytest

- .env present & keys not empty.
- .insert returns 201.
- Updated row shows new value.
- Deletion reduces row count.

10.12 Exam Style Questions

- Define Row Level Security and describe one advantage. (4 pts)
- Write SQL to create a users table with email (unique) and created_at. (4 pts)
- In the Python SDK, what does .execute() return? (3 pts)
- Why store Supabase keys in a .env file? (3 pts)
- Describe a use case for realtime subscriptions in an energy dashboard. (3 pts)
- Explain ACID properties and how Postgres ensures them. (Bonus 5 pts)
- What is Supabase and how does it differ from Firebase?
- What are the benefits of using a .env file?
- How do you insert and read from a Supabase table in Python? 5 What does .eq() do when querying Supabase?
- How does Row-Level Security enhance application security?

Security & Permissions (Optional)

Supabase uses Row-Level Security (RLS) to control who can access which data. 🗝️ What is RLS? RLS enables per-user access control at the row level, written in SQL policies. Example:
Only show user their own data:

```
CREATE POLICY "user only" ON predictions
FOR SELECT USING (auth.uid() = user_email);
```

Other Security Tips

- Avoid exposing service keys
- Use .env to manage credentials
- Always validate inputs before inserts

10.13 Chapter Summary

Supabase brings enterprise grade Postgres to data apps with a Firebase like developer experience. By mastering CRUD, secure connections, and realtime features, you now have the tools to turn your AI predictions into a full stack, production ready service.

Operation	Method Example	Description
Connect	<code>create_client(url, key)</code>	Start the session
Insert	<code>.insert({...}).execute()</code>	Add a new row
Read	<code>.select("*").limit(5).execute()</code>	Fetch rows
Update	<code>.update({...}).eq("id", val).execute()</code>	Modify values
Delete	<code>.delete().eq("email", val).execute()</code>	Remove rows

10.14 Lab Exercise – 10_exercise.ipynb



Tasks:

- 1. Create .env with your Supabase credentials
- 2. Insert 3 fake prediction entries
- 3. Fetch top 5 entries and display with st.dataframe
- 4. Add button in Streamlit to log predictions
- 5. Test update() and delete() using specific id



pytest Validations:

- .env exists and loads correctly
- Supabase client connects without error
- Insert and fetch operations run correctly
- Record deletion reflects in st.dataframe()




Reflective Case Study (Bonus)

Imagine a university dashboard that tracks student energy usage per lab. Each lab logs voltage, time, and predicted energy demand. Supabase stores this data securely, and researchers view their lab data via a Streamlit dashboard.

Questions:

1. What benefits does Supabase offer this university?
2. What risks could poor .env management cause?
3. How could role-based access control improve safety?

10.15 Further Reading & Links

- Supabase Python Client – <https://supabase.com/docs/reference/python>
- Postgres RLS Guide – <https://supabase.com/docs/guides/auth/row-level-security>
- JWT Structure – <https://jwt.io/introduction>
- Kleppmann, M., 2019. Designing data-intensive applications.
-  Supabase Docs: <https://supabase.com/docs>
-  Python-dotenv: <https://pypi.org/project/python-dotenv/>

Chapter 11: Auth Integration – Secure Access for AI Dashboards

Chapter Goals

After working through this chapter, you will be able to:

1. Define essential authentication and authorization terms (identity, session, RLS, JWT, CSRF, refresh token).
2. Configure Supabase email/password authentication for a new project.
3. Implement Register, Login, and Logout flows in Streamlit.
4. Persist user information securely with `st.session_state`.
5. Protect pages and API calls via route guard patterns.
6. Log predictions by user to the predictions table (tying back to Day 10).
7. Troubleshoot common auth errors and security pitfalls.

11.1 Vocabulary & Definitions

Term	Definition
Authentication	Process of verifying who a user is (identity).
Authorization	Process of determining what an authenticated user is allowed to do.
Session	A continuous period during which a user interacts with an app after authentication.
JWT (JSON Web Token)	Compact, signed token containing user claims (e.g., email, role, expiry).
Refresh Token	Long lived token used to obtain a new access JWT without re login.
CSRF	Cross Site Request Forgery – an attack tricking a browser into sending unwanted requests.
RLS (Row Level Security)	PostgreSQL rule allowing row based authorization policies.
Magic Link	Passwordless email link that logs the user in when clicked.
OAuth	Delegated auth framework enabling social logins (Google, GitHub, etc.).

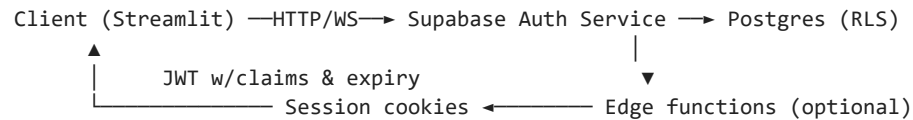
11.2 Why Authentication Matters in Energy Dashboards

- Personalised Analytics – Users see only their building’s data.
- Regulatory Compliance – Energy data often contains PII.
- Audit Trails – Identify who triggered predictions.

- Usage Based Billing – Secure access for premium services.

Case Reminder – Sichuan Lighting Plant (Chapter 9) needed per operator dashboards; authentication prevented accidental parameter changes by visitors.

11.3 Supabase Auth Architecture at a Glance



- Upon sign in, Supabase issues an access JWT (≈ 1 h) + refresh token (≈ 1 week).
- Calls to `supabase.auth` automatically include JWT in headers.
- PostgreSQL RLS can extract user info from JWT via `auth.jwt()` or `auth.uid()`.

11.4 Enabling Email/Password Auth

1. Supabase Dashboard → Auth → Providers.
2. Enable Email provider.
3. Set password rules and confirmation options.

Tip – For dev/testing, disable email confirmation; for production, keep it ON to reduce spam sign ups.

11.5 Storing Secrets in `.env`

```

SUPABASE_URL=https://<project>.supabase.co
SUPABASE_ANON_KEY=<anon-key>
SUPABASE_SERVICE_ROLE=<service-key>
  
```

- Anon key: has limited DB rights defined by RLS.
- Service role key: bypasses RLS – never ship to browsers.

11.6 Implementing Register / Login in Streamlit

Install SDK & dotenv:

```
pip install supabase python-dotenv
```

11.6.1 Initialise Client


```

from supabase import create_client
from dotenv import load_dotenv
import os, streamlit as st

load_dotenv()
url = os.getenv("SUPABASE_URL")
key = os.getenv("SUPABASE_ANON_KEY")
supabase = create_client(url, key)

```

11.6.2 Register Flow

```

def register(email: str, password: str):
    resp = supabase.auth.sign_up({"email": email, "password": password})
    if resp.user:
        st.success("✅ Please check your email to confirm your account.")
    else:
        st.error(resp.error.message)

```

11.6.3 Login Flow

```

def login(email: str, password: str):
    resp = supabase.auth.sign_in_with_password({"email": email, "password": password})
    if resp.session:
        st.session_state["user"] = resp.user.email
        st.session_state["jwt"] = resp.session.access_token
        st.experimental_rerun()
    else:
        st.error(resp.error.message)

```

11.6.4 UI Components

```

mode = st.sidebar.radio("Auth", ["Login", "Register"])
email = st.text_input("Email")
pwd = st.text_input("Password", type="password")

if mode == "Login" and st.button("Login"):
    login(email, pwd)
elif mode == "Register" and st.button("Register"):
    register(email, pwd)

```

11.7 Session State & Route Protection

```

if "user" not in st.session_state:
    st.warning("🔒 Please log in to continue.")
    st.stop()

```

- `st.stop()` halts further script execution.
- After login, rerun populates `st.session_state["user"]`, unlocking rest of page.

11.7.1 Storing Extra Metadata

```
st.session_state["login_time"] = resp.session.expires_at # epoch seconds
```

Useful for auto logout timers.

11.8 Logout Logic

```
if st.sidebar.button("🚪 Logout"):
    supabase.auth.sign_out()
    st.session_state.clear()
    st.experimental_rerun()
```

11.9 Logging Predictions by User

```
from datetime import datetime

def log_prediction(user_email, hour, dow, volt, pred):
    data = {
        "user_email": user_email,
        "input_hour": hour,
        "input_day": dow,
        "input_voltage": volt,
        "predicted_kw": pred,
        "created_at": datetime.utcnow()
    }
    supabase.table("predictions").insert(data).execute()

# Booking audit ready records for analytics.
```

11.10 Case Study – Apartment Energy Portal

- Problem – Residents want personal dashboards of monthly energy usage.
- Implementation – Streamlit + Supabase auth.
- Flow – Residents self register; RLS ensures each email sees only their rows.
- Outcome – 78% reduction in support tickets vs. Excel mailers; data refresh every 10 minutes.

11.11 Common Errors & Fixes

Error Explanation Fix SupabaseAuthError: Invalid login credentials Wrong email/pass verify input, check email confirmation JWT expired Access token aged out refresh (handled by SDK) or re login 403 RLS violation Table has RLS enabled but no policy create SELECT/INSERT policy

11.12 Hands On Lab (11_exercise.ipynb)

- Build an Auth page with Login & Register tabs.
- Protect the Prediction page from anonymous users.
- Store user_email and login_time in session.
- Add Logout button to sidebar.
- Log predictions to Supabase tied to user_email.

Expected Outcomes

- Visiting /predict while logged out sends warning + stop.
- After login, prediction table records new rows with correct email.
- Logout wipes session & returns to auth page.

11.13 Exam Style Questions

- Explain the difference between authentication and authorization. (4 pts)
- What is a JWT and what claims does Supabase include by default? (4 pts)
- Write Python code to protect a Streamlit page if user not in session. (3 pts)
- Describe one security benefit of Row Level Security. (3 pts)
- How would you implement social login (e.g., Google) in Supabase? (Bonus 5 pts)

11.14 Summary

Robust authentication underpins trustworthy AI dashboards. Supabase delivers password, magic link, and social auth with minimal configuration, while Streamlit's session_state makes it easy to manage user sessions client side. Combined with RLS, you can enforce fine grained data access and build production ready apps in hours.

11.15 Further Reading

- Supabase Auth Guide – <https://supabase.com/docs/guides/auth>
- OWASP Cheat Sheet: Auth – https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- Domoney, C., 2024. Defending APIs: Uncover advanced defense techniques to craft secure application programming interfaces. Packt Publishing Ltd.

Chapter 12: Data Upload & Persistence – Building User Centric History Dashboards

Chapter Overview

This chapter teaches you how to extend your AI dashboard with persistent storage so users can upload files, keep personal prediction logs, and revisit their past analyses. We combine Streamlit widgets with Supabase Storage and Postgres tables to build a full, self service data pipeline.

Key sections

1. Terminology & concept map
2. Why persistence matters
3. Storage buckets & ACLs
4. Upload workflow in Streamlit
5. Prediction history schema & logging
6. Personal dashboard construction
7. File listing & secure download
8. Troubleshooting & best practices
9. Hands on lab
10. Exam questions

12.1 Glossary & Definitions

Term	Definition
Persistence	Storing data so it remains available across sessions and device restarts.
Supabase Storage	S3 compatible object store managed by Supabase.
Bucket	Top level namespace in Storage, similar to an S3 bucket.
ACL (Access Control List)	Rules controlling who can read/write objects.
MIME Type	Standard identifier for file content (e.g., text/csv).
Multipart Upload	Protocol for uploading large files in parts.
ETag	Hash used to verify object integrity.
Prediction Log	Table row capturing inputs, outputs, timestamp, and user ID.
Dashboard	UI page summarising personal files & logs.

12.2 Why Persistence?

- **Traceability** – Research reproducibility and audit trails.
- **User Experience** – Users revisit previous uploads without re-uploading.
- **Data Governance** – Central storage vs. scattered local copies.
- **Monetisation** – Enable subscription tiers based on historical data depth.

📌 Case Study Snapshot

A regional energy consulting firm let clients upload monthly CSV meter dumps into a Streamlit portal; automated predictions plus historical comparison reduced analyst workload by 60%.

12.3 Supabase Storage Deep Dive

12.3.1 Bucket Anatomy

Each bucket has:

- **Name** – globally unique within project.
- **Public flag** – if false, signed URLs are required.
- **Policies** – RLS style JSON allowing fine-grained access.

12.3.2 Advantages over raw S3

Feature	S3	Supabase
Built in Auth	✗	✓ JWT integration
Dashboard upload UI	✗	✓
Postgres metadata	manual	auto via storage.objects view

12.4 Setting Up the uploads Bucket

1. Navigate to **Storage** → **New Bucket**, name it `uploads`.
2. Choose **Public** for tutorial simplicity (switch to private+policies in production).
3. Bucket path format:

`https://<project>.supabase.co/storage/v1/object/public/uploads/...`

12.5 Streamlit File Uploader Workflow

12.5.1 Widget → Bytes

```
uploaded = st.file_uploader("Upload CSV", type=["csv"]) # returns BytesIO
if uploaded:
    content = uploaded.getvalue() # bytes
```

12.5.2 Generate Unique Object Name

```
from datetime import datetime
fname = f"{st.session_state.user}/{datetime.utcnow().isoformat()}_{uploaded.name}"
```

Reason – Prevent overwriting & enable folder like structure per user.

12.5.3 Upload & Verify

```
resp = supabase.storage.from_("uploads").upload(
    path=fname,
    file=content,
    file_options={"content-type": "text/csv"}
)
if resp.status_code == 200:
    st.success("✅ Upload successful")
else:
    st.error(resp.error)
```

Expected outcome: New row appears in `storage.objects` Postgres view.

12.6 Schema for Prediction History

```
create table user_predictions (
  id uuid primary key default uuid_generate_v4(),
  user_email text not null,
  file_name text,
  hour int,
  day_of_week int,
  voltage numeric(5,2),
  predicted_kw numeric(6,3),
  created_at timestamptz default now()
);
```

Column rationale

- `file_name` links prediction row to uploaded dataset.
- `created_at` enables time series dashboards.

Add index:

```
create index on user_predictions (user_email, created_at desc);
```

12.7 Inserting Logs from Streamlit

```
def log_pred(file_name, inputs: dict, pred_kw: float):
    row = {
        "user_email": st.session_state.user,
        "file_name": file_name,
        **inputs,
        "predicted_kw": pred_kw
    }
    supabase.table("user_predictions").insert(row).execute()
```

Expected: Row visible in Supabase Table Editor.

12.8 Building the Personal Dashboard

```
import pandas as pd, plotly.express as px
```

```
st.header("📊 My Dashboard")
```

1. History table

```
hist = supabase.table("user_predictions")
    .select("hour, voltage, predicted_kw, created_at")
    .eq("user_email", st.session_state.user)
    .order("created_at", desc=True)
    .execute().data
```

```
df = pd.DataFrame(hist)
st.dataframe(df)
```

2. Trend chart

```
if not df.empty:
    fig = px.line(df, x="created_at", y="predicted_kw", title="Prediction Trend")
    st.plotly_chart(fig, use_container_width=True)
```

Expected UI: Sortable table + interactive line chart per user.

12.9 Listing & Downloading Uploads

```
objs = supabase.storage.from_("uploads").list(st.session_state.user)
for o in objs:
    url = supabase.storage.from_("uploads").get_public_url(f"{st.session_state.user}/{o['name']}")
    st.write(o["name"], " - ", f"[Download]({url})")
```

Why Signed URLs?

For private buckets:

```
download_link = supabase.storage.from_("uploads").create_signed_url(path, 3600)
3600s expiry prevents link leakage.
```

12.10 Validation & Security Tips

Risk	Mitigation
Large uploads DDOS	Set file size limit in file_uploader (e.g., max_upload)
Malicious file types	Restrict type="csv"; validate MIME server side
Bucket enumeration	Switch to private + RLS policies
PII leakage	Log minimal fields; hash IDs if public bucket

Equation – Max Daily Storage Cost (simplified):

$Cost_{day} = N \cdot S \cdot C$

where:

- N = number of files
- S = average file size (GB)
- C = storage cost per GB (\approx \$0.025 on Supabase)

Helps estimate daily storage budgeting.

12.11 Troubleshooting FAQ

Symptom	Cause	Fix
413 Payload Too Large	File > bucket limit	Compress or chunk upload
AuthHeaderMissing	User not logged in	Guard upload code with session check
425 Too Early	Clock skew	Sync server time or use NTP

12.12 Hands On Lab (`12_exercise.ipynb`)

1. Upload a CSV ≤ 1 MB & verify in dashboard.
2. Insert 3 prediction logs referencing that file.
3. Filter dashboard by date range (`st.date_input`).
4. Download an uploaded file using signed URL.
5. **BONUS:** Compute total kW predicted per file & show bar chart.

Expected Outcomes

- Table shows only current user rows.
- Chart updates with new logs within 2 s.
- Signed URL works for private bucket.

12.13 Exam Questions

1. Explain the difference between public and private buckets in Supabase. (4 pts)
2. Derive a formula to estimate monthly storage cost for 500 MB daily uploads. (4 pts)
3. Write Python code to create a signed URL valid for two hours. (3 pts)
4. What RLS policy would you write so users can only SELECT their own files? (3 pts)
5. List two UX benefits of a personal dashboard vs. raw database dumps. (3 pts)

12.14 Chapter Summary

Persistent data transforms a one-off prediction tool into a living analytics platform. Supabase Storage simplifies file handling, while history tables turn ephemeral predictions into trackable insights. With validation, RLS, and signed URLs, you can balance openness and security for real-world deployments.

12.15 Further Reading

- [Supabase Storage Guide](#)
- [Postgres Constraints & Indexes](#)
- [Streamlit File Handling](#)

Chapter 13: Cloud Deployment

Learning Outcomes

By the end of this chapter, learners will be able to:

- Deploy a Streamlit app via GitHub → Streamlit Cloud
 - Configure secrets securely in Streamlit Cloud
 - Validate the public URL and share with peers or instructors
-

13.1 What is Cloud Deployment?

Cloud deployment refers to the process of making an application accessible on the internet through cloud platforms. In the case of our AI-powered app (Smart Energy), cloud deployment allows users, instructors, or stakeholders to:

- Access the Streamlit web app from anywhere via a URL
- Interact with the energy prediction model without installing Python
- Enable real-time demos, class grading, and live peer reviews

Definition: A cloud platform is a service that provides the infrastructure, tools, and environments needed to deploy and manage web applications. Examples include Streamlit Cloud, Heroku, and AWS.

13.2 Git & GitHub: The Foundation of Deployment

13.2.1 Git Repository Initialization

```
git init
git add .
git commit -m "Initial commit"
```

This initializes a local Git repository and saves all changes with a commit.

13.2.2 Connect to GitHub & Push

Option 1: CLI via GitHub CLI

```
gh repo create smart-energy-advisor --public --source=. --remote=origin
git push -u origin main
```

Option 2: Manual setup

```
git remote add origin https://github.com/yourusername/smart-energy-advisor.git
git push -u origin main
```

Definition: Remote repository refers to the version of your project hosted on GitHub. It enables collaboration and cloud deployment.

13.3 Deployment-Ready Folder Checklist

Ensure the following files exist:

```
plaintext
├── .streamlit/
│   └── secrets.toml      ✓ (store credentials)
├── app.py or main_app.py ✓ (entry point)
├── requirements.txt      ✓ (Python dependencies)
└── README.md             ✓ (optional but good practice)
```

13.4 Secure Secrets with secrets.toml

You don't want to hardcode API keys (like Supabase keys) in your scripts. Instead, use Streamlit secrets.

13.4.1 Local secrets.toml

Create a file in `.streamlit/secrets.toml` with:

```
SUPABASE_URL = "https://xyzcompany.supabase.co"
SUPABASE_KEY = "your-anon-key"
```

Definition: secrets.toml is a secure configuration file used by Streamlit to store environment variables (like API keys).

13.4.2 Streamlit Online Secrets Editor

1. Visit <https://share.streamlit.io>
2. Open app > Settings > Secrets
3. Paste your secrets

```
SUPABASE_URL="https://xyzcompany.supabase.co" SUPABASE_KEY="your-anon-key"
```

Best Practice: Never expose secrets in `main_app.py` Always read from `st.secrets`.

13.5 Streamlit Cloud Deployment

Streamlit Cloud provides free hosting for Streamlit apps directly from GitHub. Steps to deploy:

1. Log into [Streamlit Cloud](https://share.streamlit.io) `https://share.streamlit.io`
 2. Click "New App"
 3. Choose:
 - GitHub repository: smart-energy-advisor
 - Branch: main
 - File: main_app.py
 4. Click "Deploy"
-

13.6 Validate the Deployment

Once the deployment completes:

- Visit your live URL: `https://yourusername.streamlit.app`
- Test functionality:
 - Login/auth works
 - Upload + predictions work
 - Logs display

Share the link via:

- Class portal
 - Job applications
 - LinkedIn / GitHub README
-

13.7 Real-World Case: CDUT Student Project

Jin Tao, a student at CDUT, deployed their energy prediction app using GitHub and Streamlit Cloud. The app included login, energy forecast, upload features, and a feedback section. They submitted the link in their assignment and received a perfect score for usability and presentation.

13.8 Common Deployment Errors & Fixes

Problem	Solution
ModuleNotFoundError	Add missing module to requirements.txt
App stuck loading	Check for syntax errors
secrets undefined	Upload secrets.toml properly
No repo found	Confirm GitHub repo is public

Tip: Use Streamlit logs tab to debug issues post-deployment.

13.9 Lab Activity: 13_exercise.ipynb

1. Push code to GitHub with secrets
2. Deploy on Streamlit Cloud
3. Paste public URL in notebook
4. Add screenshots

 Instructor Checks:

- GitHub repo is linked
- Secrets configured
- App is live and functional

Quiz Time

1. What is the purpose of requirements.txt?
2. How does secrets.toml improve security?
3. What error occurs if SUPABASE_KEY is hardcoded?
4. How do you restart a Streamlit app manually?
5. Which file tells Streamlit which script to run first?

Summary Table

Concept	Description
Git & GitHub	Version control and cloud code hosting
requirements.txt	Lists all Python packages

Concept	Description
secrets.toml	Secure file for credentials
Streamlit Cloud	Platform to deploy Streamlit apps from GitHub
Public URL	Auto-generated link to hosted app

Take-Home Task

- Deploy your variant of Smart Energy Advisor+
- Add 2 custom widgets (e.g., feedback)
- Include "About" in README
- Submit:
 - GitHub link
 - Deployed app link
 - Screenshot of live UI

Further Reading

- [Streamlit Cloud Docs](https://docs.streamlit.io/) `https://docs.streamlit.io/`
 - [Securing Secrets in Apps](https://docs.streamlit.io/library/advanced-features/secrets-management) `https://docs.streamlit.io/library/advanced-features/secrets-management`
 - [GitHub Starter Guide](https://docs.github.com/en/get-started/quickstart/hello-world) `https://docs.github.com/en/get-started/quickstart/hello-world`
-

Chapter 14: Monitoring & Continuous Retraining – Keeping AI Fresh in Production

CDUT AI Summer School

Chapter Objectives

By the end of this chapter, you will be able to:

1. **Define** key ML-ops terms (model drift, data quality, feature skew, canary, CI/CD, cron, A/B shadow).
 2. **Instrument** a Streamlit + Supabase app to collect prediction vs actual metrics.
 3. **Visualise** rolling MAE/RMSE to detect drift.
 4. **Automate** nightly retraining with GitHub Actions.
 5. **Evaluate & promote** the best model using statistical guard-rails.
 6. **Document** the pipeline with status badges and README links.
-

14.1 Vocabulary & Concepts

Term	Definition
Model Drift	Gradual degradation of model performance due to changing data distributions.
Concept Drift	The underlying target concept ($P(y)$) changes.
Data Drift	The input distribution ($P(x)$) shifts even if ($P(y)$) stays stable.
Feature Skew	Mismatch between train-time and serving-time feature values.
Canary Deployment	Release strategy where a new model serves a small traffic slice before full rollout.
CI/CD	Continuous Integration / Continuous Deployment pipeline automating tests and releases.
Cron	Unix scheduler syntax (e.g. " <code>0 3 * * *</code> " → 03:00 UTC daily).
MAE	Mean Absolute Error, $(\frac{1}{n}\sum \hat{y}_i - y_i)$.
RMSE	Root Mean Squared Error, penalises large errors quadratically.
GitHub Action	YAML-described workflow executed in GitHub's runners.

14.2 Why Monitoring & Retraining?

Even the best initial model loses accuracy: households add solar panels, sensors get recalibrated, extreme weather events alter energy patterns. Neglecting drift risks:

- **Poor recommendations** → user distrust.
- **Regulatory penalties** for inaccurate billing.
- **Wasted compute cost** when predictions misguide optimisation.

Real-world Example – A UK utility saw MAE double during COVID-19 lockdowns; automated retraining restored accuracy within 48h.

14.3 Collecting Ground-Truth & Error Metrics

14.3.1 Extending the Prediction Log

Add actual load once measured:

```
ALTER TABLE user_predictions
ADD COLUMN actual_kw NUMERIC(6,3);
```

14.3.2 Updating Rows *post-hoc*

```
supabase.table("user_predictions")\
    .update({"actual_kw": observed_kw})\
    .eq("id", row_id).execute()
```

14.3.3 Error-Calculation Script

```
import pandas as pd, sqlalchemy as sa

hist = pd.read_sql(
    """SELECT created_at, predicted_kw, actual_kw
       FROM user_predictions
       WHERE actual_kw IS NOT NULL""", conn)

hist["abs_err"] = (hist.predicted_kw - hist.actual_kw).abs()
```

14.4 Visualising Drift – Rolling Metrics

```
import matplotlib.pyplot as plt
hist = hist.set_index("created_at")
mae7 = hist["abs_err"].rolling("7d").mean()

plt.figure(figsize=(10,4))
plt.plot(mae7, label="7-day MAE")
plt.axhline(mae7.iloc[0]*1.25, color="r", ls="--",
            label="Drift Threshold (+25%)")
```



```
plt.title("Rolling MAE - Drift Detection")
plt.legend(); plt.tight_layout();
```

Threshold Rule – If MAE exceeds **25 %** of baseline for **3 consecutive days** → trigger retrain job.

14.5 Automating Retraining with GitHub Actions

14.5.1 Directory Layout

```
.github/workflows/
  retrain.yml
scripts/
  train_model.py
models/
  model.pkl      # prod
  candidate.pkl
```

14.5.2 workflow YAML Explained

```
name: Nightly Retrain
on:
  schedule:
    - cron: "0 3 * * *" # 03:00 UTC daily
  workflow_dispatch:    # manual trigger

jobs:
  retrain:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: "3.11"
      - run: pip install -r requirements.txt
      - run: python scripts/train_model.py
```

14.5.3 train_model.py Skeleton

```
import joblib, pandas as pd, json, numpy as np
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv("data/features.csv")

X = df.drop(columns=["datetime", "Global_active_power"])
```

```

y = df["Global_active_power"]

Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.2, shuffle=False)

model = XGBRegressor(n_estimators=300,
                     learning_rate=0.05,
                     max_depth=5)
model.fit(Xtr, ytr)

rmse = np.sqrt(
    mean_squared_error(yte, model.predict(Xte)))

# Save candidate model + metrics
joblib.dump(model, "models/candidate.pkl")
with open("models/metrics.json", "w") as f:
    json.dump({"rmse": rmse}, f)

```

14.6 Conditional Promotion Logic

Add a promotion step:

```

- name: Promote if Better
  run: python scripts/promote.py

```

promote.py

```

import json, os, joblib

with open("models/metrics.json") as f:
    new_rmse = json.load(f)["rmse"]

with open("models/prod_metrics.json") as f:
    prod_rmse = json.load(f)["rmse"]

if new_rmse < prod_rmse * 0.97:          # ≥ 3 % better
    os.replace("models/candidate.pkl", "models/model.pkl")
    os.replace("models/metrics.json", "models/prod_metrics.json")
    print("Promoted new model 🍷")
else:
    print("Candidate worse - keeping current prod model")

```

Improvement ratio


$$I = \frac{\text{RMSE}_{\text{prod}} - \text{RMSE}_{\text{new}}}{\text{RMSE}_{\text{prod}}}$$

Equation: Improvement threshold

$$I = \frac{\text{RMSE}_{\text{prod}} - \text{RMSE}_{\text{new}}}{\text{RMSE}_{\text{prod}}}$$

14.7 Status Badges in `README.md`

![Retrain](https://github.com/<user>/smart-energy-advisor/actions/workflows/retrain.yml/badge.svg)

Shows  green on success or  red on failure.

14.8 Canary or Shadow Deployment (*Advanced*)

Instead of instant promotion, route **10%** of traffic to the candidate model:

- Log comparative errors.
 - If stable after 24 h → full promotion. Can be done by random hash split inside Streamlit code.
-

14.9 Hands-On Lab `14_exercise.ipynb`

1. Compute **30-day** rolling MAE plot.
2. Write `train_model.py` to load latest `features.csv`.
3. Configure `retrain.yml` with daily **cron**.
4. Threshold for promotion = **5%** improvement.
5. Trigger workflow via *Run workflow* button and inspect logs.

Expected Results

- Badge turns **green**.
 - `models/model.pkl` updated only if better.
 - Rolling MAE drops after promotion.
-

14.10 Exam Questions

1. Differentiate **data drift** and **concept drift** with examples.
 2. Derive the formula for **rolling RMSE** over a 1-week window.
 3. Explain how `workflow_dispatch` enables manual hot-fix training.
 4. List two risks of automated model promotion and mitigations.
 5. Write YAML to schedule a job at **02:00 CST (UTC+8)** daily.
-

14.11 Chapter Summary

Monitoring and retraining close the ML-ops loop: **measure** → **detect** → **adapt**.

Automated drift checks and nightly CI training keep **Smart Energy** accurate as real-world conditions evolve — all with free GitHub runners and a few dozen lines of code.

14.12 Further Reading

- Cruz López, A.D.L., 2023. State of the art practices for machine learning operations in Google Cloud (Doctoral dissertation, ETSI_Informatica).
- GitHub Actions Docs – <https://docs.github.com/actions>
- Continual Learning for Unsupervised Anomaly Detection in Continuous Auditing of Financial Accounting Data by Hamed Hemati, Marco Schreyer and Damian Borth. AAAI 2022 Workshop on AI in Financial Services: Adaptiveness, Resilience & Governance, 2022– <https://arxiv.org/abs/2112.13215>

Chapter 15: Capstone Demo Day

Learning Outcomes

This marks the culmination of our 15-chapter **Machine Learning** learning journey. By the end of this chapter, learners will be able to:

- Present a **5-minute live demonstration** of their deployed Streamlit app.
 - Summarize key concepts and techniques from the entire project lifecycle.
 - Reflect on technical challenges and lessons learned.
 - Submit a complete project with GitHub repo, Streamlit deployment, and a final technical report.
-

15.1 Live Demo Requirements

Capstone presentations offer a chance to showcase:

Core Demo Components

1. **End-to-End App Functionality** – Demonstrate how your app supports user authentication, predictions, data uploads, and historical tracking.
2. **Live Interactions** – Include login, file upload, model predictions, and explainability outputs.
3. **Deployed URL** – Ensure the public Streamlit Cloud URL is live and accessible.

Suggested 5-Minute Presentation Flow

Segment	Suggested Time
App Overview	1 min
Live Walkthrough	2.5 min
Challenges & Lessons	1 min
Future Work	0.5 min

Presentation Tips

- Practice twice to ensure flow.
- Use your notebook to rehearse.
- Open all required tabs/files before starting.

15.2 Reflection & Lessons Learned

Include a reflection in your notebook or in your final report. Address:

- **What was the most difficult component?** (e.g., connecting Supabase, handling `.env`, prediction accuracy)
- **How did you resolve it?** (e.g., using logs, reading docs, asking questions)
- **What would you add with more time?** (e.g., visual improvements, additional AI models, power-consumption simulation)

Example Reflections

“We struggled with SHAP library compatibility in Streamlit Cloud. We resolved this by downgrading matplotlib and explicitly setting the backend.”

“If given more time, I would enhance the dashboard with Baidu Maps API to visualize CO₂ emissions by province.”

15.3 Final Deliverables

A. GitHub Repository Checklist

```
|— main_app.py
|— .streamlit/           # secrets.toml excluded via .gitignore
|— data/
|— models/
|— scripts/train_model.py
|— .github/workflows/retrain.yml
|— requirements.txt
|— README.md            # include deployed app URL
```

B. Final Report (*2 pages max*)

- **Project Summary**
- **Architecture Diagram**
- **Results & Screenshots**
- **Lessons Learned & Future Directions**

C. Submission Checklist

- GitHub repo **with commit history**
- **Live app link** deployed via Streamlit Cloud
- **Final poster report**

- Completed **15_exercise.ipynb**
-

15.4 Lab Tasks – 15_exercise.ipynb

1. Paste your deployed app link.
2. Add two screenshots from your app (login screen, prediction screen).
3. Include SHAP or MAE plot from your history.
4. Answer reflection questions.
5. Confirm GitHub and report submission.

Sample Questions

1. What was the most technically rewarding part of the project?
 2. What is one thing you now understand better?
 3. How will you maintain your deployed app?
 4. What advice would you give to future students?
-

Summary Table

Deliverable	Description
Deployed App URL	Your cloud-based working AI interface
GitHub Repository	Version-controlled project code
Capstone Report	Summary of design, findings, next steps
Notebook Submission	Final reflections and visuals

Key Terms & Concepts

- **Capstone Project** – The final comprehensive task that combines knowledge across all lessons.
- **Deployment** – Putting a software product into production or public access.
- **Version Control** – Tracking changes in code over time using systems like Git.
- **User Flow** – The sequence of steps a user takes to interact with an application.
- **Reflections** – A metacognitive practice that helps learners think critically about their learning process.
- **SHAP** – SHapley Additive exPlanations, used to interpret model predictions.
- **Streamlit Cloud** – A platform for deploying and sharing Streamlit apps instantly.

Optional Resources

- Tips for Presenting Data Science Projects
 - Effective GitHub Project Layout
 - Example Streamlit Capstone
-

Wrap-Up Quiz

1. What are three key features expected in your final app demo?
 2. Why is it important to version-control your work?
 3. How does Streamlit Cloud simplify AI app delivery?
 4. What are signs of a well-prepared capstone report?
 5. What is one way you can extend your current project post-course?
-

Congratulations!

You have successfully completed the machine-learning journey. From **Chapter 1** to **Chapter 15**, you've learned how to:

- Preprocess real-world energy data
- Build baseline and advanced AI models
- Integrate interpretability tools (SHAP)
- Secure your app with Supabase
- Deploy, monitor, and retrain AI systems

 **You're now capable of developing, deploying, and managing production-level AI apps.**

Let your deployed app speak for your skills. Share your URL on LinkedIn, your CV, or your portfolio. Stay curious. Keep building!