WEEK 3: AI/ML CORE DEVELOPMENT

GOAL:

By the end of this week, your team will implement a core AI/ML module tailored to your specific energy workflow. Whether your use case involves anomaly detection, classification, document extraction, or summarization, you will build a functional script using machine learning libraries like scikit-learn, create visuals, and produce a clear natural-language summary. The goal is not to build a perfect model but to deliver a meaningful automation feature that makes your chosen workflow faster or easier.

This week is highly practical: you will write and run Python code, visualize results, and begin to think about how these tools will fit into an end-to-end automation pipeline.

DELIVERABLES (Due Friday):

- Working AI/ML scripts or notebooks tailored to your team's workflow
- At least one useful visualization (chart, table, label)
- A natural-language summary (from GPT or mock)
- A Python module (ai_module.py) wrapping your logic into reusable functions

DAY 1: DEFINE YOUR USE CASE + BASE MODEL

Task 1.1: Define Your ML Problem

Create a markdown cell or section in your notebook answering:

- What workflow pain point are you solving?
- What is your input data (e.g., CSV, text, timeseries)?
- What output do you expect (e.g., labels, scores, summaries)?

Task 1.2: Choose a Baseline Model

Use scikit-learn to pick a starting model:

- LinearRegression → forecasting
- IsolationForest → anomaly detection
- LogisticRegression → classification

Install packages (if needed):

```
pip install scikit-learn pandas matplotlib
```

Task 1.3: Build and Test Your First Model

Depending on your workflow, choose the most relevant approach below. You only need to implement the one that fits your problem (e.g., forecasting, anomaly detection, or classification).

Option 1: Anomaly Detection (e.g., unusual production events)

```
from sklearn.ensemble import IsolationForest

model = IsolationForest(contamination=0.05)
model.fit(df[["output_kwh"]])
df["anomaly"] = model.predict(df[["output_kwh"]]) == -1
print(df[["date", "output_kwh", "anomaly"]].head())
```

Option 2: Classification (e.g., categorize energy type by features)

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = df[["wind_speed", "solar_irradiance"]]
y = df["energy_type"]  # e.g., "wind", "solar"
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
model = LogisticRegression().fit(X_train, y_train)
preds = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, preds))
```

Option 3: Forecasting (e.g., energy output prediction)

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X = df[["day_index"]]  # Convert dates to numerical index if needed
y = df["output_kwh"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
model = LinearRegression().fit(X_train, y_train)
preds = model.predict(X_test)
```

```
print("MSE:", mean_squared_error(y_test, preds))
```

DAY 2: TRAINING, TUNING & INTERPRETATION

Task 2.1: Improve Data Handling

- Use `.info()` and `.describe()` to check your columns
- Create new features, e.g., rolling averages:

```
df["rolling_7d"] = df["output_kwh"].rolling(7).mean()
```

Task 2.2: Tune Your Model

- Adjust parameters or try a second model
- Use GridSearchCV if applicable:

```
from sklearn.model_selection import GridSearchCV
params = {"max_depth": [2, 5, 10]}
grid = GridSearchCV(DecisionTreeRegressor(), param_grid=params,
cv=3)
grid.fit(X_train, y_train)
print("Best params:", grid.best_params_)
```

Task 2.3: Interpret Key Outputs

- Show top features:

```
importance = model.coef_ if hasattr(model, 'coef_') else
model.feature_importances_
print("Feature importances:", importance)
```

- Use charts to interpret results

DAY 3: VISUALIZATION + MODULE WRAP-UP

Task 3.1: Create Charts

- Use either matplotlib or plotly:

```
import matplotlib.pyplot as plt
plt.plot(df["date"], df["output_kwh"])
plt.title("Energy Output Over Time")
plt.show()
```

Task 3.2: Highlight AI-Detected Patterns

- If using classification or anomaly detection:

```
anomalies = df[df["anomaly"] == True]
plt.scatter(anomalies["date"], anomalies["output_kwh"], color='red')
```

Task 3.3: Convert to Reusable Functions

```
# In ai_module.py:
def train_model(X, y):
    model = LinearRegression()
    model.fit(X, y)
    return model

def predict(model, X):
    return model.predict(X)
```

Task 3.4: Validate Your Output

- Create a test script to run your functions
- Save a test output file: predictions.csv, model.pkl

DAY 4: AI-GENERATED SUMMARY

Task 4.1: Write a Prompt Template

- Design a structured input for GPT or your logic:

  Summarize report from Wind Site A:

      - Avg output: 4000 kWh
      - Anomalies: June 5, June 9
      - Peak output: 6000 kWh

Task 4.2: Build a Summary Function

- With GPT API:

```
import openai

def gpt_summary(prompt_text):
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt_text}]
    )
    return response.choices[0].message.content
```

- Without GPT (mock):

```python
def generate_mock_summary(df):
    anomalies = df[df["anomaly"] == True]["date"].tolist()
    return f"Detected anomalies on: {', '.join(anomalies)}"
```

Task 4.3: Export the Summary

```python
with open("weekly_summary.txt", "w") as f:
    f.write(summary)
```

Attach to CSV:

```python
df["summary"] = summary
df.to_csv("final_output_with_summary.csv")
```

DAY 5: COMBINE + DOCUMENT

Task 5.1: Finalize Your Workflow

- Create main_pipeline.py or a new notebook that runs:
  - Load cleaned data
  - Call your model function
  - Predict and visualize
  - Generate and export a summary

Task 5.2: Write Documentation

- Create a README.md or markdown cell that explains:
  - What your workflow does
  - What model and metrics you used
  - What outputs the user sees (chart, table, text)
  - What isn't working yet

Task 5.3: Prepare Your Deliverables Folder

- Upload:
  - ai_module.py
  - main_pipeline.py or notebook
  - final_output.csv and weekly_summary.txt
  - README.md

DAILY BREAKDOWN - SOLO VERSION

Day 1 – Problem Setup + First Model

Deliverable 1: In ai_model_dev.ipynb, clearly define the problem you're solving. Include:

- What energy workflow are you addressing?
- What kind of input data are you using?
- What output does your model need to generate?

Deliverable 2: Implement one baseline ML model using scikit-learn.
Choose one:

- LinearRegression for forecasting
- IsolationForest for anomaly detection
- LogisticRegression for classification

Print basic output: predictions, accuracy, or MSE.

Day 2 – Feature Engineering + Tuning

Deliverable 3: Add at least two useful features to your dataset. Example ideas:

- 7-day rolling average
- Time-based indicators (e.g., weekday)
- Lag features (e.g., yesterday's output)

Deliverable 4: Tune model parameters or try an alternate model. Optionally use GridSearchCV.

Deliverable 5: Briefly document which features mattered most and what changes improved the results.

Day 3 – Visualization + Reusable Code

Deliverable 6: Create one visualization that clearly shows model output or energy behavior.

- Line plot of predictions
- Scatter plot of anomalies
- Bar chart or confusion matrix for classification

Deliverable 7: Create ai_module.py with at least two reusable functions:

- train_model(X, y)
- predict(model, X)

Deliverable 8: Save your model output (predictions or labels) to predictions.csv.

Day 4 – AI Summary

Deliverable 9: Write a prompt template for a summary, manually or for GPT. For example: "Summarize site performance: Avg = 4300 kWh, Anomalies = June 2, 4, Peak = 6000 kWh"

Deliverable 10: Write a function to generate that summary.

- If using GPT, create a wrapper for the API
- If not, write a mock summary function using your output data

Deliverable 11: Save your summary to weekly_summary.txt. You can also attach it to predictions.csv if useful.

Day 5 – Final Integration + Documentation

Deliverable 12: Create main_pipeline.py (or notebook) that performs:

- Load cleaned dataset
- Train model
- Predict or label new data
- Generate summary
- Save all outputs

Deliverable 13: Write a short README.md explaining:

- What your ML module does
- What model and metrics you used
- What outputs the user should expect
- What still needs improvement