WEEK 4: AI INTEGRATION & APP PROTOTYPING

GOAL:
This week, you will integrate your AI/ML module into a working **app prototype** that automates part of your selected energy workflow. You'll connect AI results (e.g., charts, summaries, anomaly flags) into a user-facing interface and automate alerting using tools like Zapier. The goal is to create a functional tool that feels realistic for an energy professional to use — even if basic.

IMPORTANT: Your final product this week should be a **working app prototype**, not just a script. It should reflect the **UI structure you proposed in Week 1 (Figma sketch)**. You are encouraged to adjust layouts or features, but your final interface must be based on your original plan. You may use **Streamlit, React, or another simple frontend framework** for the interface.

Team Tip: Split responsibilities this week:

- Half your team can work on **backend improvements** — e.g., refining your AI logic, cleaning up ML scripts, improving summaries.

- The other half should focus on the **frontend interface** — building file upload, charts view, table of uploaded files, summaries, alerts panel, and other key screens from your Figma.

DELIVERABLES (Due Friday):

- An interactive **web app prototype** (Streamlit, React, etc.) that includes:
    - Chart(s) showing real energy data
    - AI-generated summary (mock or real)
    - Anomaly or classification markers clearly labeled
- Automated **alert trigger** using **Zapier** (e.g., sends email/Slack if anomaly is found)
- A working codebase (e.g., app.py, alerts.py, dashboard.jsx) in your repo
- Documentation covering:
    - What you completed this week
    - What features didn't work or weren't completed
    - What you plan to complete in Week 5 – You'll only have half the week next week to finish up this project, so choose the most pressing/important features that need to be worked on

DAY 1: ADD SUMMARIES TO UI

Task 1.1: Display Summary Output

- Import your generate_summary() or gpt_summary() function

- Display the summary clearly in your web interface:

```
# Streamlit
import streamlit as st
summary = generate_mock_summary(df)
st.markdown(f"**Weekly Summary:**\n{summary}")
```

```
// React SummaryCard.js
function SummaryCard({ text }) {
  return (
    <div className="summary-card">
      <h3>Weekly Summary</h3>
      <p>{text}</p>
    </div>
  );
}
```

Task 1.2: Style Your Layout

- Follow your Week 1 Figma UI plan
- Suggested UI sections:
    - Header/title bar
    - File upload module
    - Charts section (data trends)
    - Summary view (GPT response or mock)
    - Anomaly alert logs or recent alerts

DAY 2: ANOMALY DETECTION + ALERT LOGIC

Task 2.1: Run Anomaly Detection

Use your IsolationForest or Z-score model:

```
from sklearn.ensemble import IsolationForest
model = IsolationForest(contamination=0.05)
df['anomaly'] = model.fit_predict(df[['output_kwh']]) == -1
```

Task 2.2: Show Anomalies on Chart

```
import matplotlib.pyplot as plt
anomalies = df[df["anomaly"] == True]
plt.plot(df["date"], df["output_kwh"])
plt.scatter(anomalies["date"], anomalies["output_kwh"], color="red")
plt.title("Energy Output with Anomalies")
```

```
// React + Chart.js
import { Line } from 'react-chartjs-2';
const data = {
  labels: dates,
  datasets: [
    { label: 'Output (kWh)', data: outputs, borderColor: 'blue' },
    { label: 'Anomalies', data: anomalyData, borderColor: 'red', type:
'scatter' }
  ]
};
```

Task 2.3: Build Alerts Table or Section

```
# Streamlit
st.write("Alerts:")
for i, row in anomalies.iterrows():
    st.write(f"Anomaly on {row['date']}: {row['output_kwh']} kWh")
```

```
// React Alerts.jsx
function AlertTable({ alerts }) {
  return (
    <table>
      <thead><tr><th>Date</th><th>Output (kWh)</th></tr></thead>
      <tbody>
        {alerts.map((a, i) => <tr
key={i}><td>{a.date}</td><td>{a.output}</td></tr>)}
      </tbody>
    </table>
  );
}
```

DAY 3: DASHBOARD UI IMPLEMENTATION

Task 3.1: Build Full Layout

- **Streamlit:** Use st.sidebar, st.line_chart, st.file_uploader, st.markdown
- **React:** Organize with components like Header, SummaryCard, UploadForm, ChartPanel, AlertTable
- Optional: Flask backend can serve predictions/summaries via REST API

Task 3.2: Connect Data + Display

- Connect chart points to summary or anomaly details
- Example: Hovering over chart reveals exact value & date
- Connect React frontend with Flask backend using axios:

```
axios.post("/api/analyze", uploadedData).then(res =>
setSummary(res.data.summary));
```

DAY 4: AUTOMATE ALERTS WITH ZAPIER

Task 4.1: Prepare Alert Trigger Files

```
alerts = anomalies[["date", "output_kwh"]]
alerts.to_csv("alerts_today.csv", index=False)
```

Task 4.2: Create a Zapier Flow

- Trigger: New row in Google Sheets or file upload to Google Drive
- Action: Send email (Gmail) or Slack message
- Example email content:

Subject: Anomaly Detected
Body:
Energy dip detected on 2024-07-03
Output: 3200 kWh (below threshold)
Weekly summary: [insert GPT output]

Task 4.3: Test and Capture

- Trigger your pipeline → check if alert fires
- Take screenshots of:
    - Alerted email or Slack post

    - Zapier flow steps

DAY 5: FINAL TESTING + CATCH-UP

Task 5.1: End-to-End Testing

- Upload data → generate summary → show chart → trigger alert
- Fix broken logic, edge cases, file format issues
- Refactor messy code or consolidate multiple scripts

Task 5.2: Document What's Left

In a README.md or week4_notes.md, write:

- What you completed this week
- What parts didn't get finished (e.g., GPT API not connected)
- What you'll work on in Week 5 (e.g., UI polishing, better summaries, login system)

STARTER TEMPLATES

Streamlit template:

```
import streamlit as st
import pandas as pd

df = pd.read_csv("cleaned_data.csv")
st.title("Energy Dashboard")
st.line_chart(df["output_kwh"])
```

React template: (uses Vite or Create React App)

```
import React from 'react';
import SummaryCard from './components/SummaryCard';
import ChartPanel from './components/ChartPanel';

function App() {
  return (
    <div>
      <h1>Energy AI Dashboard</h1>
      <SummaryCard text="This week saw 2 anomalies." />
      <ChartPanel />
    </div>
  );
}
export default App;
```

Flask API Scaffold:

```
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route("/api/analyze", methods=["POST"])
def analyze():
    data = request.get_json()
    # Run your model here
    return jsonify({"summary": "Anomalies detected on June 9 and 11."})
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

Sample Zapier Workflows

Workflow 1: Slack Alert for Energy Dip

- Trigger: Google Sheets → new row with "anomaly=True"
- Action: Slack → send message to channel
    - Message: "Anomaly Detected on {{date}} — Output: {{output_kwh}} kWh"

Workflow 2: Email Summary Weekly

- Trigger: Google Drive → file uploaded to "/summaries"
- Action: Gmail → send to ops@example.com
    - Subject: "Weekly Energy Summary"
    - Body: Use summary from text file as email body

Workflow 3: Send Dashboard Link When New File is Uploaded

- Trigger: Google Drive → new file in uploads folder
- Action: Gmail → send dashboard link to uploader

WEEK 4: AI INTEGRATION & APP PROTOTYPING (INDIVIDUAL VERSION)

GOAL:

This week, you'll take your AI/ML script and turn it into a simple app prototype that automates part of an energy workflow. You'll build a minimal user interface, show charts, generate a summary, detect anomalies, and set up a basic alert with Zapier. This version is simplified for one person to complete solo. Focus on: Making something that works, not something perfect. Build only what you need to demonstrate a basic flow from data → insight → alert.

DELIVERABLES (Due Friday):

- A working web app prototype (Streamlit recommended)
- A chart that shows your energy data
- An AI-generated summary (mock or real)
- Anomaly detection output clearly shown
- One Zapier alert (e.g., send email when anomaly is found)
- Simple codebase (app.py, model.py, etc.)
- A short note (in week4_notes.md) covering:
    - What you built this week
    - What didn't get finished
    - What you'll work on next week (choose 1–2 improvements)

DAY 1: SUMMARY AND UI STARTER

Task 1.1: Show a Summary in Your App

Use your generate_summary() or a mock result.

```
import streamlit as st
summary = "Anomalies detected on June 5 and 9."
st.markdown(f"**Weekly Summary:**\n{summary}")
```

Task 1.2: Add a Basic UI Layout

Start with 3 sections:

- File uploader
- Chart of energy output
- Summary and alerts

DAY 2: ANOMALIES AND CHARTS

Task 2.1: Detect Anomalies

```
from sklearn.ensemble import IsolationForest
model = IsolationForest(contamination=0.05)
df['anomaly'] = model.fit_predict(df[['output_kwh']]) == -1
```

Task 2.2: Visualize Output

```
import matplotlib.pyplot as plt
anomalies = df[df["anomaly"] == True]
plt.plot(df["date"], df["output_kwh"])
plt.scatter(anomalies["date"], anomalies["output_kwh"], color="red")
plt.title("Energy Output with Anomalies")
```

DAY 3: UI POLISH AND FILE HANDLING
Task 3.1: Clean Up Your Layout

```
import streamlit as st
import pandas as pd

uploaded = st.file_uploader("Upload CSV")
if uploaded:
    df = pd.read_csv(uploaded)
    st.line_chart(df['output_kwh'])
```

Task 3.2: Link Sections
Ensure the summary, chart, and alerts update when a new file is uploaded.

DAY 4: ZAPIER AUTOMATION
Task 4.1: Export Alerts

```
alerts = df[df['anomaly'] == True][["date", "output_kwh"]]
alerts.to_csv("alerts_today.csv", index=False)
```

Task 4.2: Set Up Zapier
Example Zapier setup:

- Trigger: New file in Google Drive
- Action: Gmail or Slack
- Message:
    - Subject: Anomaly Detected
    - Body: Output drop on {{date}}: {{output_kwh}} kWh

Take screenshots of:

- Alert email or Slack message
- Your Zapier setup steps

DAY 5: FINAL TEST AND NOTES

Task 5.1: Test the Flow

- Upload a CSV
- See chart and summary
- Confirm that the Zapier alert triggers

Task 5.2: Capture Notes

In week4_notes.md, include:

- What worked well
- What didn't (e.g., no GPT access, chart issues)
- What you'll fix or add next week (e.g., better UI, summary formatting)