**Milestone 7: ML-based Forecasting & Anomaly Detection**

**🎯 Objective:**

To enable **forecasting of KPIs** and **detect anomalies** using uploaded .csv files containing city data (e.g., energy, water, air quality). These analyses will be powered by scikit-learn and basic statistical techniques.

**✅ Step 1: Project Structure**

arduino

CopyEdit

services/

├── kpi\_file\_forecaster.py ← Forecast future trends

├── anomaly\_detector.py ← Detect outliers/anomalies

routers/

├── kpi\_upload\_router.py ← Upload CSVs & trigger ML

├── anomaly\_checker.py

**✅ Step 2: Forecasting Logic**

**📄 services/kpi\_file\_forecaster.py**

python

CopyEdit

import pandas as pd

from sklearn.linear\_model import LinearRegression

import numpy as np

def forecast\_kpi(csv\_bytes):

df = pd.read\_csv(csv\_bytes)

df.dropna(inplace=True)

# Assume format: Date, Value

df['Date'] = pd.to\_datetime(df['Date'])

df['Ordinal'] = df['Date'].map(pd.Timestamp.toordinal)

model = LinearRegression()

model.fit(df[['Ordinal']], df['Value'])

# Predict next 7 days

future\_days = [df['Ordinal'].max() + i for i in range(1, 8)]

predictions = model.predict(np.array(future\_days).reshape(-1, 1))

future\_dates = pd.to\_datetime(future\_days, origin='unix', unit='D')

forecast = {str(date.date()): round(pred, 2) for date, pred in zip(future\_dates, predictions)}

return forecast

**✅ Step 3: Anomaly Detection Logic**

**📄 services/anomaly\_detector.py**

python

CopyEdit

import pandas as pd

def detect\_anomalies(csv\_bytes):

df = pd.read\_csv(csv\_bytes)

df.dropna(inplace=True)

mean = df['Value'].mean()

std\_dev = df['Value'].std()

# Outliers > 2 std from mean

df['Anomaly'] = abs(df['Value'] - mean) > 2 \* std\_dev

anomalies = df[df['Anomaly'] == True]

return anomalies.to\_dict(orient="records")

**✅ Step 4: FastAPI Upload Endpoint**

**📄 routers/kpi\_upload\_router.py**

python

CopyEdit

from fastapi import APIRouter, UploadFile, File

from services.kpi\_file\_forecaster import forecast\_kpi

from services.anomaly\_detector import detect\_anomalies

import io

router = APIRouter()

@router.post("/forecast-kpi")

async def upload\_forecast(file: UploadFile = File(...)):

forecast = forecast\_kpi(io.BytesIO(await file.read()))

return {"forecast": forecast}

@router.post("/detect-anomalies")

async def upload\_anomaly(file: UploadFile = File(...)):

anomalies = detect\_anomalies(io.BytesIO(await file.read()))

return {"anomalies": anomalies}

**✅ Step 5: Frontend Integration (Optional)**

You can now build Streamlit upload UIs for:

* **KPI Forecasting**
* **Anomaly Checker**

Let me know if you'd like Streamlit UIs (forecast\_kpi.py, anomaly\_checker.py) built next.

**Activity 7.1: Forecasting with Linear Regression**

**🎯 Goal:**

Use **Linear Regression** to predict future **water/energy usage trends** from uploaded .csv files and display the forecast results on the dashboard.

**✅ Step 1: Forecasting Service Code**

**📄 services/kpi\_file\_forecaster.py**

python

CopyEdit

import pandas as pd

from sklearn.linear\_model import LinearRegression

import numpy as np

import io

def forecast\_kpi(csv\_bytes\_io: io.BytesIO):

df = pd.read\_csv(csv\_bytes\_io)

if 'Date' not in df.columns or 'Value' not in df.columns:

return {"error": "CSV must have 'Date' and 'Value' columns."}

df.dropna(inplace=True)

df['Date'] = pd.to\_datetime(df['Date'])

df['Ordinal'] = df['Date'].map(pd.Timestamp.toordinal)

model = LinearRegression()

model.fit(df[['Ordinal']], df['Value'])

# Forecast next 7 days

last\_date = df['Date'].max()

future\_dates = pd.date\_range(start=last\_date + pd.Timedelta(days=1), periods=7)

future\_ordinals = [date.toordinal() for date in future\_dates]

predictions = model.predict(np.array(future\_ordinals).reshape(-1, 1))

forecast = {str(date.date()): round(pred, 2) for date, pred in zip(future\_dates, predictions)}

return forecast

**✅ Step 2: API Endpoint to Handle Upload**

**📄 routers/kpi\_upload\_router.py**

python

CopyEdit

from fastapi import APIRouter, UploadFile, File

from services.kpi\_file\_forecaster import forecast\_kpi

import io

router = APIRouter()

@router.post("/forecast-kpi")

async def forecast\_kpi\_route(file: UploadFile = File(...)):

csv\_bytes\_io = io.BytesIO(await file.read())

result = forecast\_kpi(csv\_bytes\_io)

return {"forecast": result}

**✅ Step 3: Streamlit Frontend UI**

**📄 frontend/components/kpi\_forecasting.py**

python

CopyEdit

import streamlit as st

import pandas as pd

from utils.api\_utils import post\_csv\_request

def forecast\_kpi():

st.subheader("📈 KPI Forecasting (Next 7 Days)")

uploaded\_file = st.file\_uploader("Upload CSV with 'Date' and 'Value' columns", type="csv")

if uploaded\_file:

result = post\_csv\_request("/forecast-kpi", uploaded\_file)

if "forecast" in result:

forecast = result["forecast"]

st.success("Forecast generated!")

st.dataframe(pd.DataFrame(forecast.items(), columns=["Date", "Predicted Value"]))

else:

st.error(result.get("error", "Unknown error"))

**📄 utils/api\_utils.py (extend with CSV handler)**

python

CopyEdit

import requests

BASE\_URL = "http://localhost:8000"

def post\_request(endpoint, data):

res = requests.post(f"{BASE\_URL}{endpoint}", json=data)

return res.json()

def get\_request(endpoint, params=None):

res = requests.get(f"{BASE\_URL}{endpoint}", params=params)

return res.json()

def post\_csv\_request(endpoint, file):

files = {'file': (file.name, file, 'text/csv')}

res = requests.post(f"{BASE\_URL}{endpoint}", files=files)

return res.json()

**Activity 7.2: Anomaly Detection**

**🎯 Goal:**

Detect **abnormal spikes or drops** in KPI values using statistical thresholds, and **display the results** in Streamlit using a **table or colored badges**.

**✅ Step 1: Create the Anomaly Detection Logic**

**📄 services/anomaly\_file\_checker.py**

python

CopyEdit

import pandas as pd

import io

def detect\_anomalies(csv\_bytes\_io: io.BytesIO):

df = pd.read\_csv(csv\_bytes\_io)

if 'Date' not in df.columns or 'Value' not in df.columns:

return {"error": "CSV must have 'Date' and 'Value' columns."}

df.dropna(inplace=True)

df['Date'] = pd.to\_datetime(df['Date'])

mean = df['Value'].mean()

std = df['Value'].std()

# Flag anomalies where value is more than ±2 standard deviations

df['Anomaly'] = df['Value'].apply(lambda x: abs(x - mean) > 2 \* std)

anomalies\_df = df[df['Anomaly']]

anomalies = anomalies\_df[['Date', 'Value']].to\_dict(orient='records')

return {

"total\_points": len(df),

"anomalies\_found": len(anomalies),

"anomalies": anomalies

}

**✅ Step 2: FastAPI Endpoint**

**📄 routers/anomaly\_checker.py**

python

CopyEdit

from fastapi import APIRouter, UploadFile, File

from services.anomaly\_file\_checker import detect\_anomalies

import io

router = APIRouter()

@router.post("/detect-anomalies")

async def detect\_anomalies\_route(file: UploadFile = File(...)):

csv\_bytes\_io = io.BytesIO(await file.read())

result = detect\_anomalies(csv\_bytes\_io)

return result

**✅ Step 3: Streamlit UI Component**

**📄 frontend/components/anomaly\_checker.py**

python

CopyEdit

import streamlit as st

import pandas as pd

from utils.api\_utils import post\_csv\_request

def check\_anomalies():

st.subheader("🚨 Anomaly Detection")

uploaded\_file = st.file\_uploader("Upload KPI CSV with 'Date' and 'Value' columns", type="csv")

if uploaded\_file:

result = post\_csv\_request("/detect-anomalies", uploaded\_file)

if "error" in result:

st.error(result["error"])

else:

st.info(f"✅ Total Data Points: {result['total\_points']}")

st.warning(f"⚠️ Anomalies Found: {result['anomalies\_found']}")

if result['anomalies\_found'] > 0:

df = pd.DataFrame(result['anomalies'])

st.dataframe(df.style.highlight\_max(axis=0, color='orange'))

else:

st.success("🎉 No anomalies detected.")