GDPR-CCPA Risk Pipeline

A reproducible project that fetches, processes, and forecasts GDPR and CCPA policy updates on an hourly basis using Apache Airflow, Python, and Prophet.

📋Prerequisites

Before you begin, ensure you have the following installed on your machine:

- **Operating System**: macOS or Linux (instructions here target macOS).
- **Git**: to clone the repository.
- **Python 3.8+**: with `venv` module available.
- **pip**: Python package manager.
- **Airflow 2.7.x**: installed within a virtual environment.
- **Prophet**: for time-series forecasting.
- **Jupyter Notebook or VS Code**: for editing and running scripts.

Optional (for troubleshooting):

- **lsof**, **pkill**, **pgrep**: CLI tools to inspect and kill processes.

🕔Installation & Troubleshooting Guide

Follow these steps carefully. Common pitfalls and their solutions are documented alongside each command.

1. Clone the repository

```
# Clone into your GitHub folder (avoid spaces in path if possible)
git clone https://github.com/yourusername/gdpr-ccpa-risk-pipeline.git
cd gdpr-ccpa-risk-pipeline
```

Pitfall: If `cd` fails, check your clone path or correct directory name.

2. Create and activate a Python virtual environment

```
python3 -m venv venv
source venv/bin/activate
```

- **Why**: Isolates dependencies.

- **Pitfall**: If you see `command not found: venv/bin/activate`, ensure you ran `python3 -m venv venv` successfully and you're in the project root.

3. Install Python dependencies

```
pip install --upgrade pip
pip install apache-airflow==2.7.1 prophet requests beautifulsoup4
```

- **Why**: Airflow core, forecasting, and data-fetch libraries.
- **Pitfall**: Installation errors often arise from missing compiler tools. On macOS, run `xcode-select --install` if Prophet fails to build.

4. Configure Airflow environment

a. Set `AIRFLOW_HOME`

Add to `~/.zshrc` (or `~/.bash_profile`):

```
export AIRFLOW_HOME="$HOME/path/to/gdpr-ccpa-risk-pipeline/airflow_home"
```

Reload:

```
source ~/.zshrc
```

- **Why**: Tells Airflow where to store its metadata and DAGs.
- **Pitfall**: Mismatched `AIRFLOW_HOME` leads to DAGs not appearing.

b. (Optional) Override `DAGS_FOLDER`

To keep DAGs in `project_root/dags`, also add:

```
export AIRFLOW__CORE__DAGS_FOLDER="$HOME/.../gdpr-ccpa-risk-pipeline/dags"
```

5. Initialize the Airflow database

```
airflow db init
```

- **Why**: Creates metadata tables.
- **Runs once**: Do **not** rerun every session.
- **Pitfall**: If prompted "Are you sure?", type `y`.

6. Create an Admin user

```
airflow users create \
  --username admin \
  --firstname Admin \
  --lastname User \
  --role Admin \
  --email you@example.com \
  --use-random-password
```

- **Why**: Enables UI login.
- **Pitfall**: Missing `--email` or `--role` flags cause errors.

🛠️Step-by-Step Setup & Execution

This section walks through from code setup in VS Code to seeing your DAG in the UI.

Step 1: Create & Configure Scripts in VS Code

1. **Open** the project in **Visual Studio Code**:

```
code .
```

1. **Under** the `scripts/` folder, create three files:

2. `fetch_policy_data.py`

3. `process_policy_data.py`

4. `forecast_policy_trends.py`

5. **In** `fetch_policy_data.py`, include:

```
# scripts/fetch_policy_data.py
import os, requests
from datetime import datetime

def fetch_policy_data():
    url = "https://www.dataprotectionreport.com/feed/"  # RSS feed for policy
updates
    resp = requests.get(url)
    resp.raise_for_status()
    fn = f"../data/raw/policy_updates_{datetime.utcnow():%Y%m%dT%H%M%SZ}.xml"
    with open(os.path.join(os.path.dirname(__file__), fn), 'wb') as f:
        f.write(resp.content)
    print(f"Fetched and saved raw data to {fn}")
```

1. **In** `process_policy_data.py`, parse XML and write CSV:

```
# scripts/process_policy_data.py
import os, pandas as pd
from bs4 import BeautifulSoup

def process_policy_data():
    raw_dir = os.path.join(os.path.dirname(__file__), '../data/raw')
    # ... load latest XML, parse entries, extract title, pubDate, source,
category ...
    # write to data/processed/cleaned_policies.csv
    print("Processed XML to CSV.")
```

1. **In** `forecast_policy_trends.py`, add forecasting logic (as detailed earlier).

Why: Separates each pipeline stage into its own reusable script.

Step 2: Airflow DAG Definition

1. Under `dags/`, create `gdpr_ccpa_risk_pipeline.py`:

```
# dags/gdpr_ccpa_risk_pipeline.py
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta

default_args = { ... }

def fetch_policy():
    from scripts.fetch_policy_data import fetch_policy_data
    fetch_policy_data()

def process_policy():
    from scripts.process_policy_data import process_policy_data
    process_policy_data()

def forecast_policy():
    from scripts.forecast_policy_trends import forecast_policy_trends
    forecast_policy_trends(periods=7)

with DAG(..., load_examples=False) as dag:
    t1 = PythonOperator(...)
    t2 = PythonOperator(...)
    t3 = PythonOperator(...)
    t1 >> t2 >> t3
```

2. In your `airflow.cfg` (inside `$AIRFLOW_HOME`), set:

```
load_examples = False
```

Why: Prevents example DAGs cluttering your UI.

Step 3: Start Airflow Services

a. Webserver (UI)

```
# Terminal A
cd gdpr-ccpa-risk-pipeline
env/bin/activate
airflow webserver --port 8081
```

b. Scheduler

```
# Terminal B
cd gdpr-ccpa-risk-pipeline
env/bin/activate
airflow scheduler
```

Step 4: Verify in the UI

1. **Open** http://localhost:8081 and **login**.

2. Ensure **load_examples=False** (no example DAGs).

3. **Refresh** to see only `gdpr_ccpa_risk_pipeline`.

4. Click its name → **Graph** tab → view three nodes:

```
fetch_policy_data → process_policy_data → forecast_policy_trends
```

1. **Trigger** (▶️) and watch each node run.

2. Confirm outputs:

```
ls data/processed/cleaned_policies.csv\ nls data/forecasts/forecast_*.csv
```

🔮 Next Steps

- **LLM Risk Scoring**: integrate T5 model to classify risk severity.
- **Dashboarding**: connect forecasts and risk scores to Looker/Tableau/Power BI.
- **Production Hardening**: migrate metadata DB to Postgres, switch to KubernetesExecutor or CeleryExecutor.
