

# Anomaly Detection Pipeline Design

i am working on a hackathon, the challenge overview is as follows : existing reconciliation tools process huge number of transactions daily and monthly, effort is very significant identifying data anomalies post-reconciliation for "breaks" and fixing them.

We have to create an anomaly detection system. The system should be able to:

1. detect anomalous data by comparing historical data
2. provide insight into potential root causes

Operators will use our AI backed system to carry out rca and detecting anomalies.

- > For detecting anomalies and categorizing them we have to use your OpenAi apis (LLMs)
- > We have to classify the anomalies (predefined buckets)
- > We are not bound to use llms, for categorizing and detecting anomalies we can use clustering algorithms, probably some sort of bollinger bands, etc.
- > The tool should be interactive and should have ability to recieve feedback, the operators can tag the detected anomalies from our system as false positive, etc.
- > ok this im not quite sure about, we can use agentic ai to look at reconciliation breaks (including anomalies) and provide concise summaries of break resolutions.
- > These agents should be able to learn from and mimic operator with a human in the loop to take corrective action for a break.,
- > streamline the workflow with operator assist agents - create resolution tasks, call apis, send emails, create jira tickets etc.

So objective is to create end-to-end pipeline that accepts reconciliation details, current and historical data as input, output:

- > Anomaly detection results (classification)
- > "break" resolution summaries

> tries to act autonomously act on the break resolutions

Ok, this was the ps, i will provide detailed example, please let me know if you have any questions regarding ps

Got it! This is an interesting problem involving anomaly detection, RCA, and automation. Some questions to clarify the approach:

**1. Data Format & Sources:**

- What does the reconciliation data look like (structured tables, JSON, etc.)?
- What kind of transactions are being reconciled (financial, inventory, etc.)?
- How much historical data do you have access to?

**2. Anomaly Detection Approach:**

- Do you have predefined thresholds for anomalies, or should the system learn dynamically?
- Are anomalies only numerical (like balance mismatches) or also categorical (unexpected transaction types)?

**3. Classification Buckets:**

- What are the predefined anomaly categories? Are they human-defined or do we need to discover them?

**4. AI Usage & Agent Actions:**

- Do we need a detailed breakdown of resolution tasks (e.g., calling APIs, emails, Jira creation)?
- Will the AI suggest break resolutions, or should it learn from operator actions over time?

**5. Feedback Loop & UI:**

- Should operators provide direct feedback (labeling false positives, correcting categorization)?
- Will the system need an interactive UI or integrate with an existing dashboard?

Once I get these details, I can help refine the architecture and suggest models/methods.

Ok now to answer ur questions. before that i want to clarify what reconciliation is : it is the process of matching \*2\*

data sources like payment, settlement data etc. The outcome could be a match or a break, we are not concerned with matches. We need to analyze each break that we receive (simple to deduce) and take corrective actions to either fix the break, or document its validity. Also the system should be generic enough to process all sorts of data provided.

Data Availability: Can you provide some sample data or the structure of reconciliation data that we'll be processing?

> Data will be of csv format, the data will consist of multiple columns but all in all can be classified into the following:

- i. Key columns : These are columns based on which the Source 1 data is matched to source 2 data.
- ii. Criteria columns : the columns for matching or breaking purpose (actual data points to analyze and conclude whether it is a match or a break)
- iii. Derived columns could be as simple as absolute difference between the criteria columns (for eg., it could be difference between 2 data received from 2 source systems and if it is close to zero its a match otherwise its a break).
- iv. date columns : to track back history
- v. comment columns : documentation of whether its match or break, if its a break, is it an anomaly, are there any corrective action taken, could also be empty

Types of Anomalies: Are there predefined categories of anomalies that you're aware of, or will the system need to identify them on its own? Any examples of "breaks" or anomaly categories?

> Anomalies are out-of-pattern incidences, so for eg., if for a given set of key columns we are getting multiple breaks, if we get a break in realtime and the values (criteria column) is showing out-of-trend or anomalous data, we tag it as an anomaly, the category of the anomaly could be as simple as huge spikes in our one of the derived columns. remember, we are analyzing whether the breaks are anomalous or not, and breaks are derived from some sort of rule on the derived columns.

Data Size: How large is the volume of data we need to process? Is it in real-time or batch mode?

> Millions of data points

Automation Scope: How much autonomy do you want the system to have? For example, are we just generating summaries and suggestions, or do we need full automation for corrective actions?

> We ARE GENERATING CORRECTIVE MEASURES AT MAX for the anomalies

Feedback Process: How will operators tag anomalies as false positives or provide feedback? Will there be a user interface for this?

> future we will decide

ok for tech stack we will use

> Django for apis

> Python for dev

> OpenAi apis for llms or even huggingface models (free resources)

can u suggest a folder directory (also input file is a csv)

Here's a suggested folder structure for your project:

```
bash
```

```
reconciliation_ai/
├── data/                # Store sample/input CSV files
│   ├── sample_input.csv
│   └── historical_data.csv
├── src/                 # Main source code
│   ├── anomaly_detection/ # Core anomaly detection logic
│   │   ├── detector.py    # Anomaly detection algorithms (LLMs, clustering,
│   │   │                   Bollinger Bands, etc.)
│   │   ├── classifier.py  # Categorization logic for anomalies
│   │   └── rca.py         # Root cause analysis logic
│   └── automation/        # Automation-related logic (API calls, Jira, Emails)
│       └── task_manager.py # Workflow automation tasks
```

```

|   |   |─ email_handler.py    # Email automation (if required)
|   |   |─ api/                # Django REST APIs
|   |   |─ views.py            # API endpoints for anomaly detection and RCA
|   |   |─ serializers.py      # API data serialization
|   |   |─ urls.py             # Route definitions
|   |   |─ models/             # ML models (if training custom models)
|   |   |─ model.py            # Custom ML models (if needed)
|   |   |─ preprocess.py       # Data preprocessing scripts
|─ tests/                      # Unit tests
|─ config/                    # Configuration settings
|   |─ settings.py            # Django & environment settings
|   |─ logging.conf           # Logging configurations
|─ scripts/                   # Utility scripts (data ingestion, preprocessing)
|   |─ ingest_data.py         # Script to load and process input CSVs
|─ requirements.txt           # Python dependencies
|─ manage.py                  # Django management script
|─ README.md                  # Documentation

```

## Explanation:

- `data/` : Stores input CSVs for testing.
- `src/anomaly_detection/` : Houses core logic for detecting and classifying anomalies.
- `src/automation/` : Manages AI-powered workflow automation (Jira, API calls, emails).
- `src/api/` : Django API layer for interacting with the system.
- `src/models/` : Optional ML model training/preprocessing if needed.
- `tests/` : Unit tests for different components.
- `scripts/` : Helpful scripts for data ingestion and preprocessing.

Would you like me to generate some boilerplate code for anomaly detection or Django API setup?

yes please

I'll provide a basic boilerplate for the anomaly detection system and Django API setup.

## Step 1: Install Dependencies

Run this in your virtual environment:

```
sh

pip install django djangorestframework pandas numpy scikit-learn openai
```

(Modify based on any additional dependencies you need.)

---

## Step 2: Django API Setup

### 1. Create Django Project & App

```
sh

django-admin startproject reconciliation_ai
cd reconciliation_ai
django-admin startapp api
```

### 2. Add `api` to `INSTALLED_APPS` in `settings.py`

Edit `reconciliation_ai/settings.py`:

```
python

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'api',  # Add this
]
```

---

## Step 3: API Endpoints

## 1. Define Models (Optional)

If you need to store detected anomalies, create a model in `api/models.py`:

```
python

from django.db import models

class Anomaly(models.Model):
    id = models.AutoField(primary_key=True)
    key_columns = models.JSONField()
    criteria_columns = models.JSONField()
    derived_columns = models.JSONField()
    date_detected = models.DateTimeField(auto_now_add=True)
    anomaly_type = models.CharField(max_length=255, null=True, blank=True)
    resolution_status = models.CharField(max_length=50, choices=[('Pending',
'Pending'), ('Resolved', 'Resolved')], default='Pending')

    def __str__(self):
        return f"Anomaly {self.id} - {self.anomaly_type}"
```

Run migrations:

```
sh

python manage.py makemigrations
python manage.py migrate
```

---

## 2. Create API Views for Anomaly Detection

Edit `api/views.py`:

```
python

import pandas as pd
import numpy as np
import openai
from rest_framework.response import Response
from rest_framework.decorators import api_view
from .anomaly_detection.detector import detect_anomalies # Import detection logic
```

```
openai.api_key = "YOUR_OPENAI_API_KEY" # Replace with actual API key
```

```
@api_view(['POST'])
```

```
def upload_csv(request):
```

```
    """
```

```
    API endpoint to process uploaded CSV and detect anomalies.
```

```
    """
```

```
    file = request.FILES.get('file')
```

```
    if not file:
```

```
        return Response({"error": "No file uploaded"}, status=400)
```

```
    df = pd.read_csv(file)
```

```
    anomalies = detect_anomalies(df)
```

```
    return Response({"anomalies": anomalies})
```

### 3. Define URLs

Edit `api/urls.py`:

```
python
```

```
from django.urls import path
```

```
from .views import upload_csv
```

```
urlpatterns = [
```

```
    path('upload_csv/', upload_csv, name='upload_csv'),
```

```
]
```

And link it in `reconciliation_ai/urls.py`:

```
python
```

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```



```
path('api/', include('api.urls')),  
]
```

## Step 4: Anomaly Detection Logic

Create `src/anomaly_detection/detector.py`:

python

```
import pandas as pd  
import numpy as np  
from sklearn.ensemble import IsolationForest  
import openai  
  
def detect_anomalies(df):  
    """  
    Detect anomalies in reconciliation data.  
    """  
  
    # Example: Using absolute difference between criteria columns  
    df["abs_diff"] = abs(df["criteria_col_1"] - df["criteria_col_2"])  
  
    # Using Bollinger Bands for anomaly detection  
    df["rolling_mean"] = df["abs_diff"].rolling(window=10).mean()  
    df["rolling_std"] = df["abs_diff"].rolling(window=10).std()  
    df["upper_band"] = df["rolling_mean"] + (2 * df["rolling_std"])  
    df["lower_band"] = df["rolling_mean"] - (2 * df["rolling_std"])  
    df["anomaly"] = (df["abs_diff"] > df["upper_band"]) | (df["abs_diff"] <  
df["lower_band"])  
  
    # Using Isolation Forest (ML-based approach)  
    iso_forest = IsolationForest(contamination=0.05)  
    df["anomaly_ml"] = iso_forest.fit_predict(df[["abs_diff"]])  
  
    # OpenAI LLM for classification (Optional)  
    anomalies = df[df["anomaly"] == True]  
    classified_anomalies = []  
    for _, row in anomalies.iterrows():  
        prompt = f"Given this reconciliation break: {row.to_dict()}, classify the  
anomaly."
```

```
response = openai.ChatCompletion.create(model="gpt-4", messages=[{"role":  
"user", "content": prompt}])  
category = response["choices"][0]["message"]["content"]  
classified_anomalies.append({"data": row.to_dict(), "category": category})  
  
return classified_anomalies
```

## Step 5: Running the Server

Run Django server:

```
sh  
  
python manage.py runserver
```

Now, you can test the API by sending a CSV file using Postman or cURL:

```
sh  
  
curl -X POST -F "file=@data/sample_input.csv" http://127.0.0.1:8000/api/upload_csv/
```

This is a basic setup. Let me know if you need modifications, like integrating agentic AI for workflow automation! 🚀

2/2

can u provide a .zip for starters

I'll generate a starter project with the folder structure, boilerplate code, and a sample CSV file. Let me package it into a `.zip` file for you.

I've generated a starter project for you. You can download it here:

 [Download reconciliation\\_ai.zip](#)

It includes the folder structure, boilerplate code for anomaly detection, a Django API, and a sample CSV file. Let me know if you need modifications! 🚀

U havent included all the files in the link yiou provided, I want the entire folder structure with boilerplate code

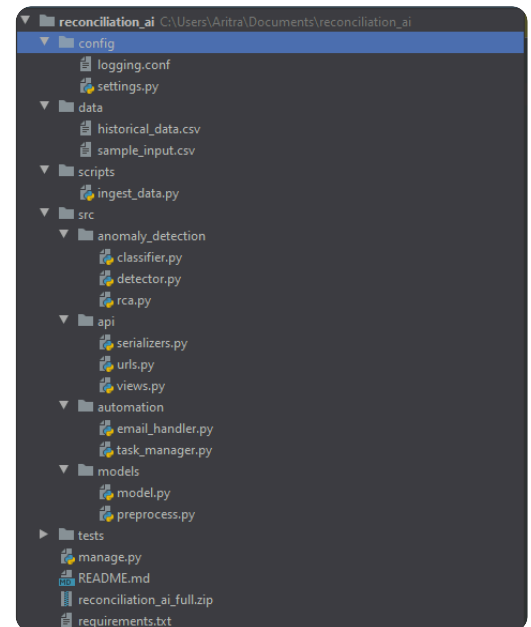
```
reconciliation_ai/
| — data/           # Store sample/input CSV files
|   | — sample_input.csv
|   | — historical_data.csv
| — src/           # Main source code
|   | — anomaly_detection/  # Core anomaly detection
logic
|   |   | — detector.py    # Anomaly detection
algorithms (LLMs, clustering, Bollinger Bands, etc.)
|   |   | — classifier.py  # Categorization logic for
anomalies
|   |   | — rca.py        # Root cause analysis logic
|   | — automation/      # Automation-related logic
(API calls, Jira, Emails)
|   |   | — task_manager.py # Workflow automation
tasks
|   |   | — email_handler.py # Email automation (if
required)
|   | — api/           # Django REST APIs
|   |   | — views.py      # API endpoints for anomaly
detection and RCA
|   |   | — serializers.py # API data serialization
|   |   | — urls.py       # Route definitions
|   | — models/        # ML models (if training custom
models)
|   |   | — model.py      # Custom ML models (if
needed)
|   |   | — preprocess.py  # Data preprocessing
scripts
| — tests/           # Unit tests
| — config/          # Configuration settings
|   | — settings.py      # Django & environment
settings
|   | — logging.conf     # Logging configurations
| — scripts/         # Utility scripts (data ingestion,
preprocessing)
```

—— ingest_data.py	# Script to load and process input CSVs
—— requirements.txt	# Python dependencies
—— manage.py	# Django management script
—— README.md	# Documentation

I've created the full project structure with boilerplate code.

 [Download reconciliation\\_ai\\_full.zip](#)

Let me know if you need any modifications! 🚀



Can you give a brief about each file

Here's a brief overview of each file in your project:

## 1. config/ (Configuration & Settings)

- `settings.py` – Placeholder for Django settings, including database configuration, installed apps, and middleware.
- `logging.conf` – Configuration file for logging, which helps track application behavior and errors.

## 2. data/ (Sample & Input Data)

- `sample_input.csv` – Example reconciliation data containing key, criteria, derived, date, and comment columns.
- `historical_data.csv` – Past transaction data for anomaly detection and trend analysis.

### 3. `scripts/` (Utility & Data Processing Scripts)

- `ingest_data.py` – Script to load and preprocess input CSV files before passing them into the anomaly detection pipeline.

### 4. `src/` (Main Source Code)

#### a. `anomaly_detection/` (Core Logic for Detecting Anomalies)

- `detector.py` – Implements anomaly detection using techniques like Bollinger Bands, Isolation Forest, or other ML models.
- `classifier.py` – Categorizes detected anomalies into predefined buckets.
- `rca.py` – Root Cause Analysis (RCA) module to determine why a break occurred.

#### b. `api/` (Django REST API for Anomaly Detection)

- `views.py` – Defines API endpoints for file uploads and anomaly detection.
- `serializers.py` – Converts Django models or request data into JSON format for API responses.
- `urls.py` – Routes API endpoints, such as `/upload_csv/`.

#### c. `automation/` (Agentic AI & Workflow Automation)

- `task_manager.py` – Handles automated workflows like task assignments and reconciliation corrections.
- `email_handler.py` – Automates email notifications for detected anomalies (if needed).

#### d. `models/` (Machine Learning & Data Processing)

- `model.py` – Placeholder for training or loading ML models for anomaly detection.
- `preprocess.py` – Handles data cleaning, transformation, and feature engineering.

### 5. `tests/` (Unit Testing)

- Placeholder directory for testing various components (e.g., anomaly detection, API, automation).

### 6. Root Files

- `manage.py` – Django management script for running the application.
- `README.md` – Project documentation, describing functionality and setup.

- `requirements.txt` – List of required Python libraries ( `Django` , `pandas` , `scikit-learn` , `OpenAI API` , etc.).

Would you like any refinements or additional explanations? 🚀