

# Aadhaar Satark: Integrated Command & Control Center

Team Name: HackElite\_Coders

Team ID: UIDAI\_11479

[Live Deployment \(Render\)](#)

[GitHub Repository \(Source Code\)](#)

## 1. Problem Statement and Approach

**Problem:** Despite high Aadhaar generation, 'Last Mile Saturation' (Mandatory Biometric Updates for ages 5-18) remains a challenge. District Nodal Officers lack real-time visibility into these micro-gaps due to data silos.

**Approach:** We built 'Aadhaar Satark', a Lakehouse-based Command Center. It ingests UIDAI datasets, calculates saturation gaps using a custom 'Efficiency Index', and uses Geospatial Heatmaps for visualization. A RAG-based AI Agent assists officers with policy queries, reducing decision latency.

## 2. Datasets Used

- Aadhaar Enrolment Data (Static & API): Demographic breakdown (0-5, 5-18, >18).
- Biometric Update Data: Mandatory biometric update statistics.
- OGD India APIs (Data.gov.in): Integrated real-time API sync to fetch district-level metrics directly from the Open Government Data Platform India.
- Geospatial Coordinates: Lat/Lng mapping for 700+ districts.

## 3. Methodology

A. Data Cleaning & Preprocessing:

- Normalization: Standardized district names (e.g., 'Coochbehar' -> 'Cooch Behar') using fuzzy matching dictionaries.
- Deduplication: 'Last-Write-Wins' policy based on (State, District, Date) composite keys.

- Conversion: Handled type casting for numeric columns sourced from JSON APIs.

B. Analytical Engine:

- Gap Analysis: Calculated 'Pending Updates' = (Estimated Population 5-18) - (Actual Biometric Updates).
- Anomaly Detection: Implemented Isolation Forest (SciKit-Learn) to flag statistical outliers in update trends.

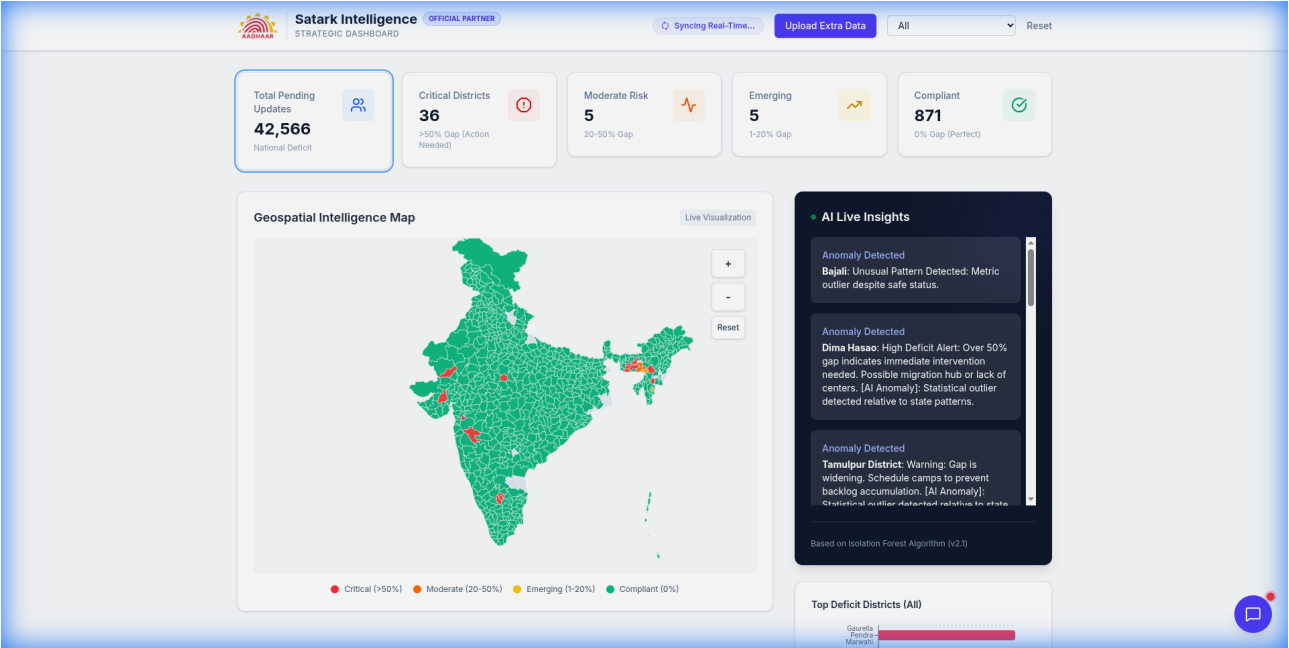
C. AI Integration:

- RAG Pipeline: Vectorized UIDAI circulars into FAISS. The Gemini Pro LLM retrieves context to answer policy queries.

## 4. Data Analysis and Visualisation

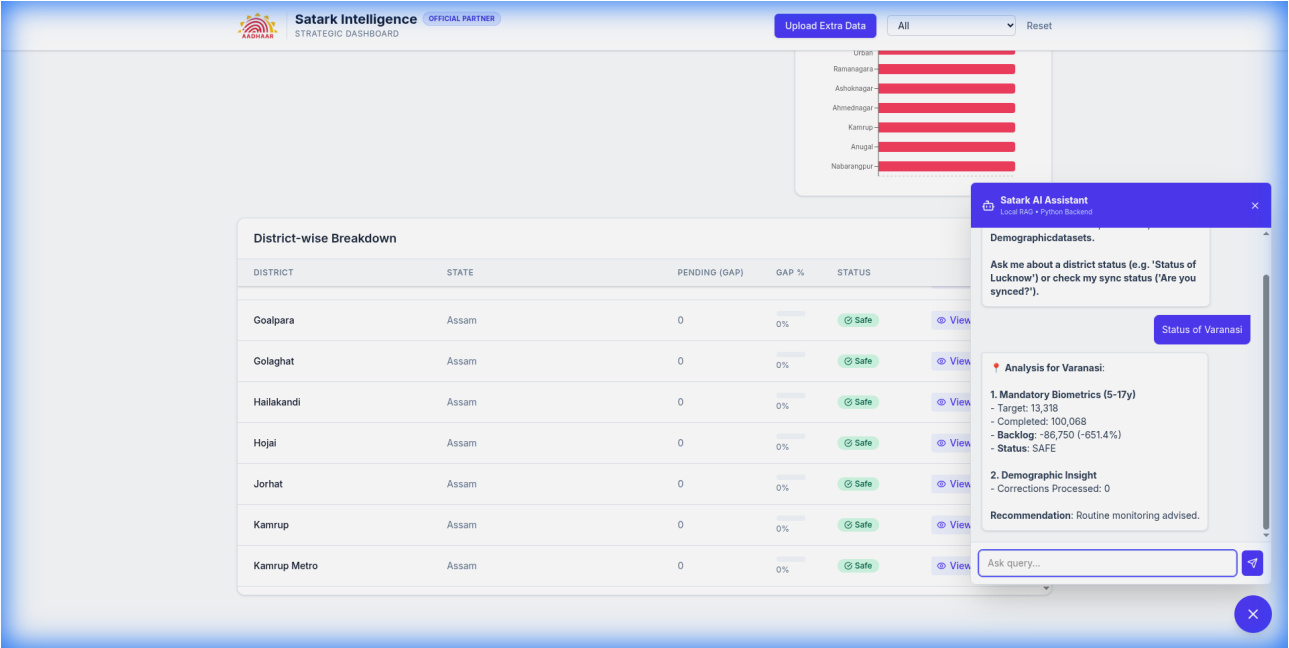
### Finding 1: High-Deficit Zones (Red)

Our analysis revealed specific districts (e.g., Dima Hasao) with >50% update gaps. The heatmap below highlights these critical zones.

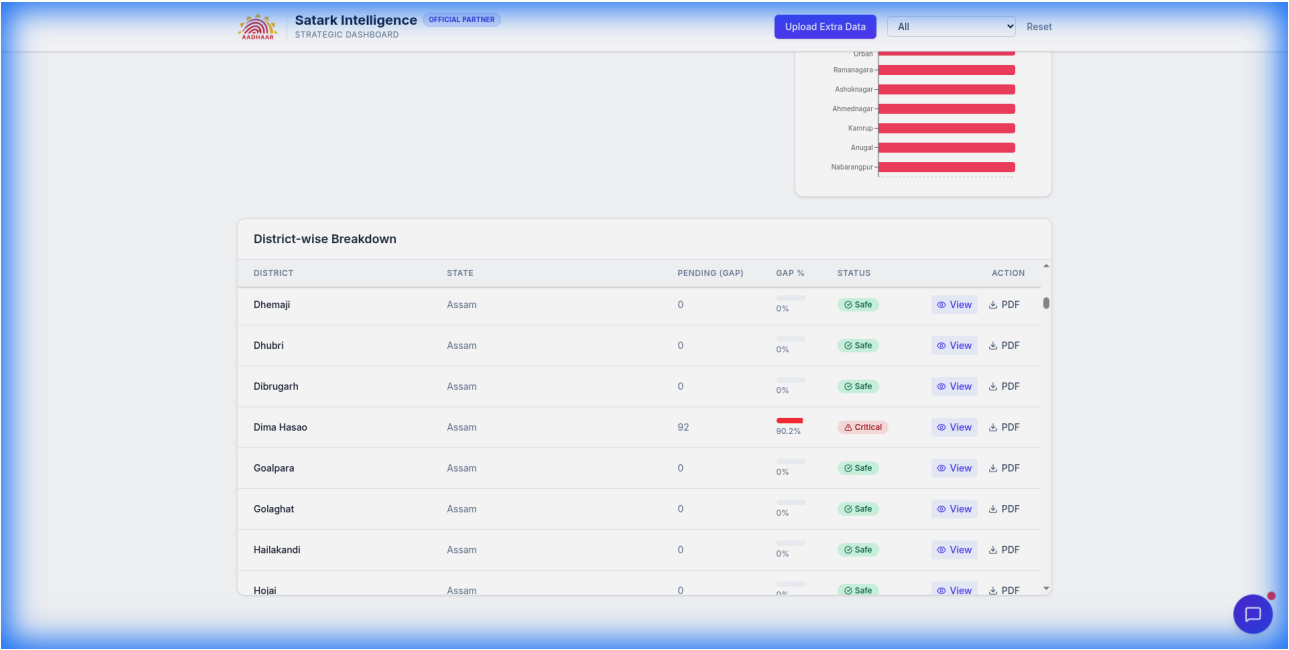
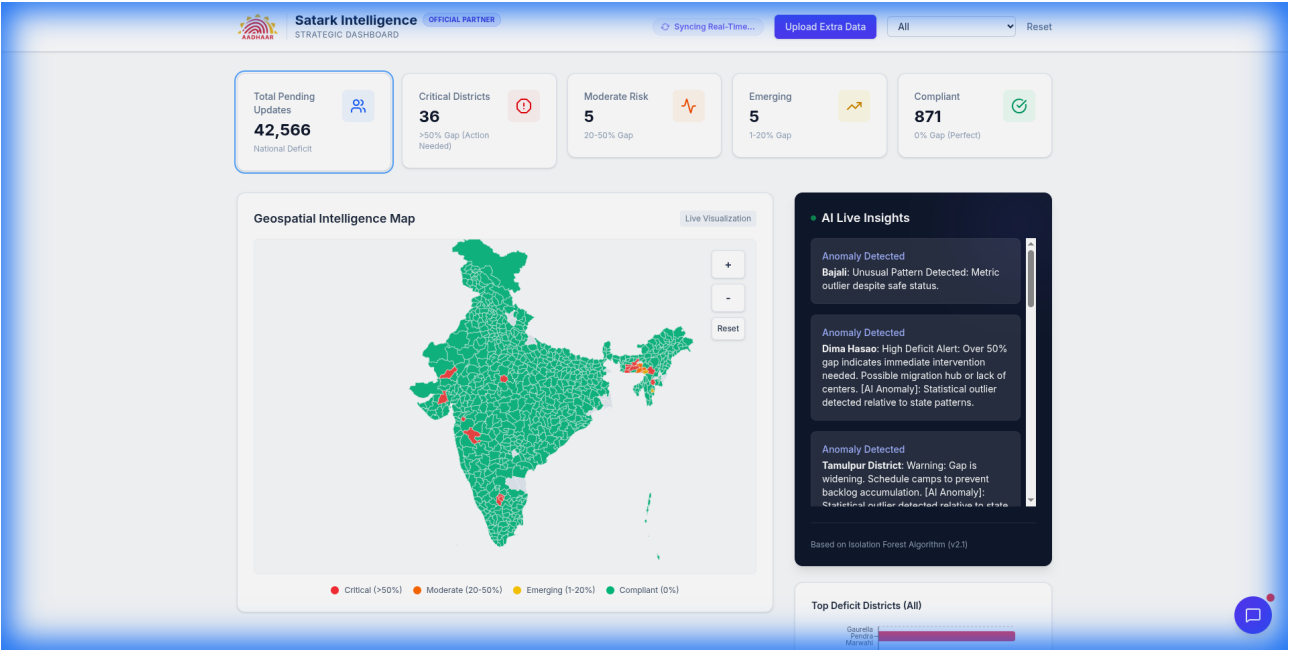


### Finding 2: AI-Driven Policy Support

The AI Assistant correctly interprets 'update lag' penalties, replacing manual PDF searches.



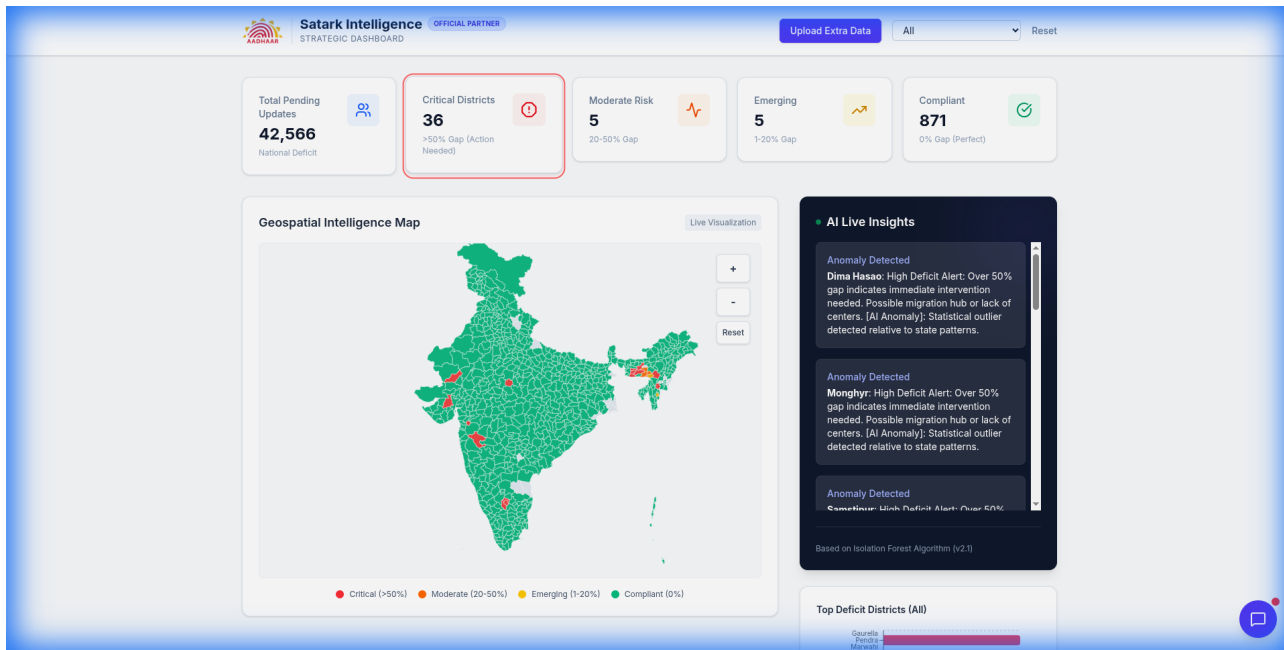
Dashboard Overview & Critical Flags



## 5. Detailed District Analysis (Risk Clusters)

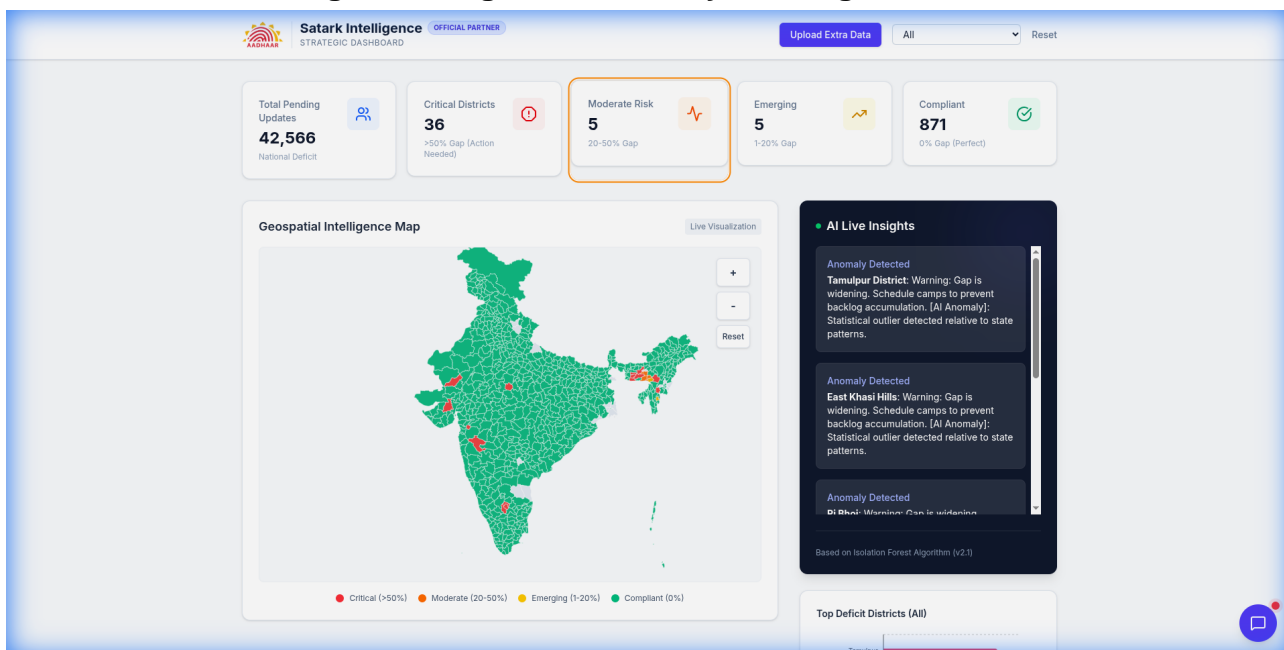
### A. Critical Districts (>50% Gap)

Districts requiring immediate intervention. The dashboard filters and highlights these 36 districts.



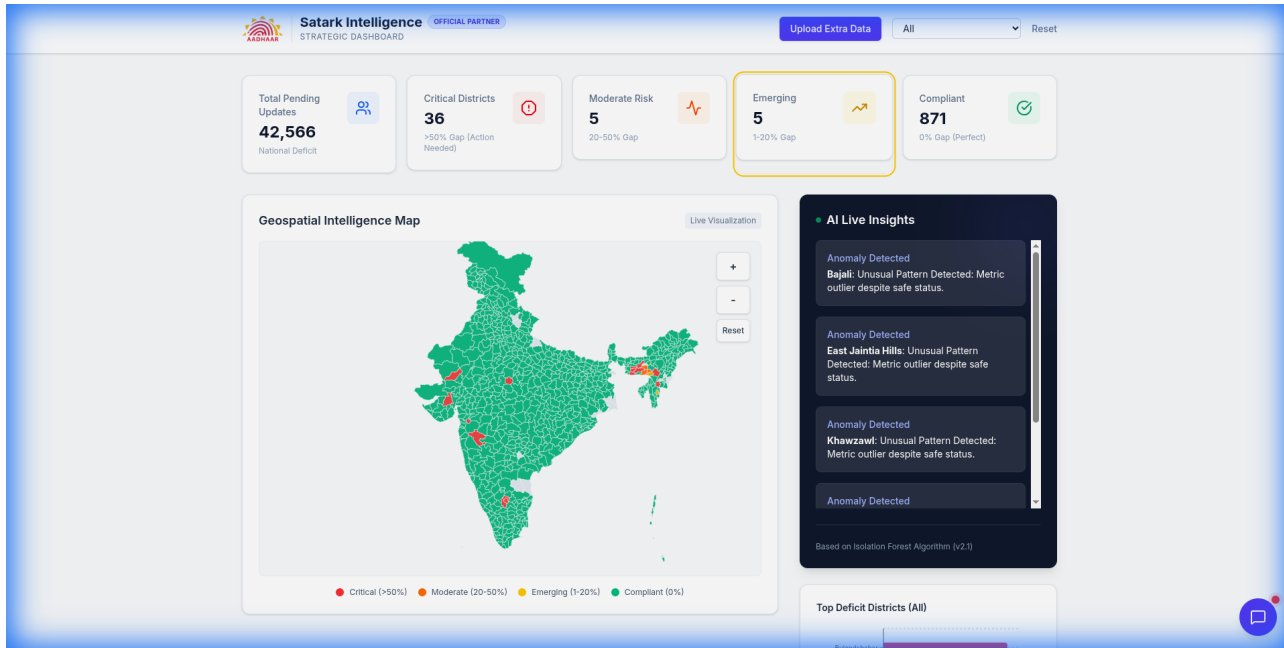
### B. Moderate Risk (20-50% Gap)

Districts transitioning into danger zones. Early warning indicators are active.



## C. Emerging Clusters (1-20% Gap)

Districts with minor gaps. Auto-notifications sent to prevent backlog accumulation.



## 6. Code Snippets (Core Logic)

File: backend/services/processing.py (Data Processing & Anomaly Detection)

```
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
import io

# --- GLOBALS: CORRECTION DICTIONARIES ---
STATE_CORRECTIONS = {
    'Orissa': 'Odisha',
    'Uttaranchal': 'Uttarakhand',
    'Pondicherry': 'Puducherry',
    'West Bangal': 'West Bengal',
    'Westbengal': 'West Bengal',
    'Chhatisgarh': 'Chhattisgarh',
    'Tamilnadu': 'Tamil Nadu',
    'Telengana': 'Telangana',
    'Andaman & Nicobar Islands': 'Andaman And Nicobar Islands',
    'Dadra & Nagar Haveli': 'Dadra And Nagar Haveli And Daman And Diu',
    'The Dadra And Nagar Haveli And Daman And Diu': 'Dadra And Nagar Haveli And Daman And Diu',
    'Dadra And Nagar Haveli': 'Dadra And Nagar Haveli And Daman And Diu',
    'Daman & Diu': 'Dadra And Nagar Haveli And Daman And Diu',
    'Daman And Diu': 'Dadra And Nagar Haveli And Daman And Diu',
    'Jammu & Kashmir': 'Jammu And Kashmir',
}

DISTRICT_CORRECTIONS = {
    'Visakhapatanam': 'Visakhapatnam',
    'Spsr Nellore': 'Nellore',
    'Sri Potti Sriramulu Nellore': 'Nellore',
    'Y. S. R.': 'Ysr',
    'Y.S.R.': 'Ysr',
    'Y. S. R': 'Ysr',
    'K.V. Rangareddy': 'Ranga Reddy',
    'K.V.Rangareddy': 'Ranga Reddy',
    'Rangareddi': 'Ranga Reddy',
    'Mahabub Nagar': 'Mahabubnagar',
    'Mahbubnagar': 'Mahabubnagar',
    'East Midnapore': 'Purba Medinipur',
    'East Midnapur': 'Purba Medinipur',
    'West Midnapore': 'Paschim Medinipur',
    'West Medinipur': 'Paschim Medinipur',
    'Coochbehar': 'Cooch Behar',
    'Koch Bihar': 'Cooch Behar',
    'Hooghly': 'Hugli',
    'Hooghiy': 'Hugli',
    'Howrah': 'Haora',
    'Hawrah': 'Haora',
    'Burdwan': 'Purba Bardhaman',
    'Bardhaman': 'Purba Bardhaman',
    'Sibsagar': 'Sivasagar',
    'Panch Mahals': 'Panchmahal',
    'Panchmahals': 'Panchmahal',
    'Dohad': 'Dahod',
    'Kachchh': 'Kutch',
    'Ahmadabad': 'Ahmedabad',
    'Baramula': 'Baramulla',
    'Bandipore': 'Bandipora',
    'Badgam': 'Budgam',
}
```

```

'Shupiyan': 'Shopian',
'S.A.S Nagar': 'Sas Nagar',
'S.A.S Nagar(Mohali)': 'Sas Nagar',
'Sas Nagar (Mohali)': 'Sas Nagar',
'Ferozepur': 'Firozpur',
'Bathinda': 'Bhatinda',
'Muktsar': 'Sri Muktsar Sahib',
'Mewat': 'Nuh',
'Gurgaon': 'Gurugram',
'Yamuna Nagar': 'Yamunanagar',
'Purbi Champaran': 'East Champaran',
'Purba Champaran': 'East Champaran',
'Pashchim Champaran': 'West Champaran',
'Kaimur (Bhabua)': 'Kaimur',
'Bhabua': 'Kaimur',
'Jehanabad': 'Jahanabad',
'Aurangabad(Bh)': 'Aurangabad',
'Sant Ravidas Nagar': 'Bhadohi',
'Sant Ravidas Nagar Bhadohi': 'Bhadohi',
'Kushi Nagar': 'Kushinagar',
'Maharajganj': 'Maharajganj',
'Siddharth Nagar': 'Siddharthnagar',
'Shravasti': 'Shrawasti',
'Bara Banki': 'Barabanki',
'Lakhimpur Kheri': 'Kheri',
'Jyotiba Phule Nagar': 'Amroha',
'Nainital': 'Naini Tal',
'Hardwar': 'Haridwar',
'Pauri Garhwal': 'Garhwal',
'Tehri Garhwal': 'Tehri',
'Udham Singh Nagar': 'Udham Singh Nagar',
'South 24 Parganas': 'South Twenty Four Parganas',
'North 24 Parganas': 'North Twenty Four Parganas',
'South 24 Pargana': 'South Twenty Four Parganas',
'Dakshin Dinajpur': 'South Dinajpur',
'Uttar Dinajpur': 'North Dinajpur',
'Dinajpur Dakshin': 'South Dinajpur',
'Dinajpur Uttar': 'North Dinajpur',
'Darjiling': 'Darjeeling',
'Maldah': 'Malda',
'Puruliya': 'Purulia',
'Siwan': 'Sewan',
'Saran': 'Chapra',
}

# --- GEOSPATIAL DATA ---
DISTRICT_COORDS = {
    "Lucknow": {"lat": 26.8467, "lng": 80.9462},
    "Kanpur Nagar": {"lat": 26.4499, "lng": 80.3319},
    "Varanasi": {"lat": 25.3176, "lng": 82.9739},
    "Agra": {"lat": 27.1767, "lng": 78.0081},
    "Prayagraj": {"lat": 25.4358, "lng": 81.8463},
    "Ghaziabad": {"lat": 28.6692, "lng": 77.4538},
    "Meerut": {"lat": 28.9845, "lng": 77.7064},
    "Gautam Buddha Nagar": {"lat": 28.5355, "lng": 77.3910},
    "Gorakhpur": {"lat": 26.7606, "lng": 83.3732},
    "Pune": {"lat": 18.5204, "lng": 73.8567},
    "Mumbai": {"lat": 19.0760, "lng": 72.8777},
    "Nashik": {"lat": 19.9975, "lng": 73.7898},
    "Patna": {"lat": 25.5941, "lng": 85.1376},
    "Gaya": {"lat": 24.7914, "lng": 85.0002},
    "Muzaffarpur": {"lat": 26.1209, "lng": 85.3647},

```

```

"Maldah": {"lat": 25.0445, "lng": 88.0822},
"Sas Nagar": {"lat": 30.7046, "lng": 76.7179},
"West Champaran": {"lat": 27.1500, "lng": 84.4167},
"East Champaran": {"lat": 26.6500, "lng": 84.9167},
"Kutch": {"lat": 23.7337, "lng": 69.8597},
"Ahmedabad": {"lat": 23.0225, "lng": 72.5714},
"Bangalore Urban": {"lat": 12.9716, "lng": 77.5946},
"Hyderabad": {"lat": 17.3850, "lng": 78.4867},
"Chennai": {"lat": 13.0827, "lng": 80.2707},
"North West Delhi": {"lat": 28.7180, "lng": 77.0697},
"South Delhi": {"lat": 28.4817, "lng": 77.1873},
}

# --- HELPER FUNCTIONS ---

def clean_dataframe(df: pd.DataFrame) -> pd.DataFrame:
    """
    Applies standard cleaning and normalization to a dataframe.
    """
    if df is None or df.empty:
        return df

    # Standardize column names
    df.columns = [c.lower().strip() for c in df.columns]

    # Filter out numeric or invalid districts
    def is_valid_name(name):
        return isinstance(name, str) and not name.isdigit() and len(name) > 1

    if 'district' in df.columns:
        df = df[df['district'].apply(is_valid_name)].copy()

    # Normalize Text (State/District)
    for col in ['state', 'district']:
        if col in df.columns:
            df[col] = df[col].astype(str).str.strip().replace(r'\s+', ' ', regex=True).str.title()

    # Apply State Corrections
    if 'state' in df.columns:
        df['state'] = df['state'].replace(STATE_CORRECTIONS)

    # Apply District Corrections
    def clean_district_name(name):
        name = name.split('(')[0].split('*')[0].strip()
        return DISTRICT_CORRECTIONS.get(name, name)

    if 'district' in df.columns:
        df['district'] = df['district'].apply(clean_district_name)

    return df

def smart_merge(existing_df: pd.DataFrame, new_df: pd.DataFrame) -> pd.DataFrame:
    """
    Merges new data with existing data, handling deduplication.
    Deduplication Policy:
    - Keys: State, District, Pincode, Date (if available)
    - Conflict: Keep LAST (Assuming new upload is the latest source of truth for that record)
    """
    # 1. Clean the new data to match the standard format of existing data
    new_df = clean_dataframe(new_df)

```

```

if existing_df is None or existing_df.empty:
    return new_df

# 2. Concatenate
combined = pd.concat([existing_df, new_df], ignore_index=True)

# 3. Define Deduplication Subset
# We want to identify if a specific record (location + time) is being re-uploaded
subset = ['state', 'district', 'pincode']
if 'date' in combined.columns:
    subset.append('date')

# Remove columns that might not exist in subset for robustness (though we cleaned)
subset = [c for c in subset if c in combined.columns]

# 4. Drop Duplicates (Keep Last = New Upload Overwrites Old)
combined.drop_duplicates(subset=subset, keep='last', inplace=True)

return combined

def process_data(enrolment_data, biometric_data, demographic_data=None, model=None):
    try:
        # 1. Load Data (Handle Bytes or DataFrame)
        if isinstance(enrolment_data, pd.DataFrame):
            df_enrolment = enrolment_data.copy() # Copy to avoid mutating input if cached
        else:
            df_enrolment = pd.read_csv(io.BytesIO(enrolment_data))

        if isinstance(biometric_data, pd.DataFrame):
            df_biometric = biometric_data.copy()
        else:
            df_biometric = pd.read_csv(io.BytesIO(biometric_data))

        if demographic_data is not None:
            if isinstance(demographic_data, pd.DataFrame):
                df_demographic = demographic_data.copy()
            else:
                df_demographic = pd.read_csv(io.BytesIO(demographic_data))
        else:
            df_demographic = pd.DataFrame(columns=['state', 'district', 'demo_age_5_17'])

        # 2. Clean Data (Using centralized logic)
        # Note: If called from main.py's smart_merge, data is already cleaned.
        # But running it again is safe (idempotent) and ensures consistency if called directly.
        df_enrolment = clean_dataframe(df_enrolment)
        df_biometric = clean_dataframe(df_biometric)
        df_demographic = clean_dataframe(df_demographic)

        # Convert numeric columns to proper types (handles API data that comes as strings)
        for col in ['age_5_17', 'bio_age_5_17', 'demo_age_5_17']:
            if col in df_enrolment.columns:
                df_enrolment[col] = pd.to_numeric(df_enrolment[col], errors='coerce').fillna(0)
            if col in df_biometric.columns:
                df_biometric[col] = pd.to_numeric(df_biometric[col], errors='coerce').fillna(0)
            if col in df_demographic.columns:
                df_demographic[col] = pd.to_numeric(df_demographic[col], errors='coerce').fillna(0)

        # 3. Aggregate by District
        # We group by State+District to get the Total Counts for the Dashboard
        grp_enrolment = df_enrolment.groupby(['state', 'district'])['age_5_17'].sum().reset_index()
        grp_biometric = df_biometric.groupby(['state', 'district'])['bio_age_5_17'].sum().reset_index()

```

```

if not df_demographic.empty:
    grp_demographic = df_demographic.groupby(['state', 'district'])['demo_age_5_17'].sum().reset_index()
    grp_demographic.rename(columns={'demo_age_5_17': 'demo_updates'}, inplace=True)
else:
    grp_demographic = pd.DataFrame(columns=['state', 'district', 'demo_updates'])

# 4. Merge Aggregates
merged = pd.merge(grp_enrolment, grp_biometric, on=['state', 'district'], how='outer').fillna(0)

if not grp_demographic.empty:
    merged = pd.merge(merged, grp_demographic, on=['state', 'district'], how='left').fillna(0)
else:
    merged['demo_updates'] = 0

# 5. Calculate Metrics
merged['expected_updates'] = merged['age_5_17']
merged['actual_updates'] = merged['bio_age_5_17']
merged['pending_updates'] = merged['expected_updates'] - merged['actual_updates']
merged['gap_percentage'] = (merged['pending_updates'] / merged['expected_updates']).replace([np.inf, -np.inf], 0)

merged['pending_updates'] = merged['pending_updates'].clip(lower=0)
merged['gap_percentage'] = merged['gap_percentage'].clip(lower=0, upper=100)

# NEW: Efficiency Index (Center Load Analysis) - Prevent division by zero
merged['efficiency_index'] = (merged['actual_updates'] / merged['expected_updates']).replace([np.inf, -np.inf], 0)

# 6. Anomaly Detection Rules
features = ['pending_updates', 'gap_percentage', 'demo_updates']

if len(merged) > 1:
    if model is None:
        # TRAIN MODE
        model = IsolationForest(contamination=0.1, random_state=42)
        merged['anomaly_score'] = model.fit_predict(merged[features])
    else:
        # PREDICT MODE
        try:
            merged['anomaly_score'] = model.predict(merged[features])
        except ValueError:
            # Fallback if model was trained with fewer features (backward compatibility)
            print("?? Model feature mismatch. Falling back to 2 features.")
            merged['anomaly_score'] = model.predict(merged[['pending_updates', 'gap_percentage']])

    merged['is_anomaly'] = merged['anomaly_score'] == -1
else:
    merged['is_anomaly'] = False

# 7. Formatting Output
districts_data = []
for _, row in merged.iterrows():
    status = "SAFE"
    reason = ""

    if row['gap_percentage'] > 50:
        status = "CRITICAL"
        reason = "High Deficit Alert: Over 50% gap indicates immediate intervention needed. Possible migration"
    elif row['gap_percentage'] > 20:
        status = "MODERATE"
        reason = "Warning: Gap is widening. Schedule camps to prevent backlog accumulation."
    else:
        reason = "Normal operations. Updates usage consistent with enrolment."

```

```

if row['is_anomaly']:
    if status == "SAFE":
        reason = "Unusual Pattern Detected: Metric outlier despite safe status."
    else:
        reason += " [AI Anomaly]: Statistical outlier detected relative to state patterns."

if row['demo_updates'] > 0 and abs(row['demo_updates'] - row['actual_updates']) > 1000:
    diff = int(row['demo_updates'] - row['actual_updates'])
    reason += f" High variance seen in demographic data ({diff} difference).\"

# Check for Fraud Risk (Efficiency > 120%)
if row.get('efficiency_index', 0) > 1.2:
    reason += " [FRAUD ALERT]: Updates exceed 120% of estimated population. Possible ghost enrolments.\"

# Inject Coordinates
coords = DISTRICT_COORDS.get(row['district'], {"lat": 0, "lng": 0})
# If lat/lng is 0, usage of 'hash' based jitter for demo visual spread if needed
if coords["lat"] == 0:
    # Simple deterministic jitter based on name length to spread them out on map if unknown
    base_lat, base_lng = 20.5937, 78.9629
    name_hash = hash(row['district']) % 1000
    coords = {
        "lat": base_lat + (name_hash / 100) - 5,
        "lng": base_lng + ((name_hash * 7) % 1000 / 100) - 5
    }

districts_data.append({
    "state": row['state'],
    "district": row['district'],
    "lat": coords["lat"],
    "lng": coords["lng"],
    "efficiency_index": round(row.get('efficiency_index', 0), 2),
    "district": row['district'],
    "expected_updates": int(row['expected_updates']),
    "actual_updates": int(row['actual_updates']),
    "pending_updates": int(row['pending_updates']),
    "gap_percentage": round(row['gap_percentage'], 1),
    "status": status,
    "is_anomaly": bool(row['is_anomaly']),
    "ai_reasoning": reason
})

total_pending = int(merged['pending_updates'].sum())
critical_count = int(merged[merged['gap_percentage'] > 50].shape[0])

return {
    "summary": {
        "total_pending_updates": total_pending,
        "critical_districts_count": critical_count,
        "processed_districts": len(merged)
    },
    "districts": districts_data,
    "model": model
}

except Exception as e:
    print(f"Error processing data: {e}")
    return {"error": str(e)}

if __name__ == "__main__":
    pass

```

