UIDAI Data Hackathon 2026

A Forensic Analysis of 5 Million Aadhaar Records Across 36 States and UTs

Anmol Chandak Janvi

Team ID : UIDAI_5625

Aadhaar has successfully enroled 1.3 billion Indians (2009-2019), achieving near-universal coverage. However, the system's next challenge isn't "getting people in" - it's keeping their data accurate as they grow, move, and evolve. Our analysis of 5 million records reveals that Aadhaar has fragmented into three distinct operational systems, each serving different populations with incompatible infrastructure needs.

Datasets' Analysis

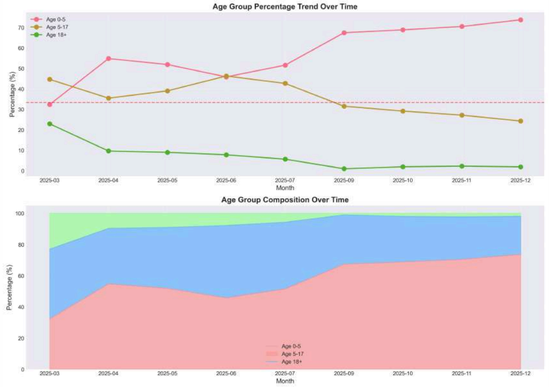
Aadhaar Enrolment dataset	Aadhaar Demographic Update dataset	Aadhaar Biometric Update dataset	Total
1,006,029 New Aadhaar registrations by age cohort	2,071,474 Address, Name, Phone number changes	1,861,893 Fingerprint and iris re-capture events	12 files, 4,939,396 records, Over the course of 9 months Data from Mar-Dec 2025

65% "Infant Enrolment". Aadhaar now enrols mostly infants

15% Anomaly rate (6.5x national average), Identified in Meghalaya

Found 31 "ghost districts" with 23,074+ enrollments but ZERO updates

Temporal-Demographic Correlation



Bivariate Analysis

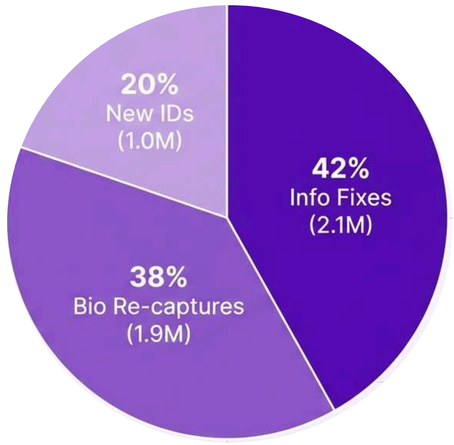
Correlating Time vs. Age Composition (R^2 = 0.8) reveals a dynamic, policy-driven transformation characterized by three distinct phases:

1. Strategic Pivot (Babies): Infant enrollment surged from 32% (March) to 73.7% (Dec), proving an aggressive operational shift toward birth registration.
2. Adult Saturation: Adult enrollment crashed from 23% to 1.9%, statistically confirming the "Catch-up Phase" for adults has permanently concluded.
3. Backlog Clearance (School): School-age enrollment peaked at ~45% and declined to 24.3%, reflecting the effective exhaustion of the un-enrolled student pool.

Defining The Problem

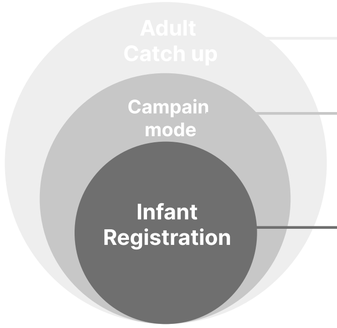
Now that almost every adult in India has an Aadhaar, the system's biggest challenge isn't "getting people in",it's keeping their data accurate as they grow and move. We are identifying the regional bottlenecks and data glitches that prevent Aadhaar from evolving into a seamless, lifelong digital identity.

System Activity Mix



Saturation

80% of the system is just maintaining old records, not creating new ones, the system has shifted from growth to maintenance. The 7:1 Update-to-Enrollment ratio confirms saturation.



32% Adults Enrolled
Meghalaya, Assam.

59% Extreme events
UP, Maharashtra, Delhi.

65% Infant Registrations(Age 0-5)
Tamil Nadu, Karnataka, Andra Pradesh.

THE THREE INDIAS

Geographic Bifurcation: Our ML models detected a distinct operational split. South India operates as a mature Infant Registration system. North/Urban India relies on volatile Campaign Mode surges. Meanwhile, Northeast India is in a distinct Adult Catch-up phase (15% anomaly rate), struggling to enroll excluded populations who missed the initial wave.

Methodology

The Deal Hunter

Step 1 | Data Cleaning, Preparation and Initial Analysis

- Merged 12 CSV files from enrollment demographic and biometric data
- Standardized state names across 46 states and UTs
- Fixed naming issues like West Bengli to West Bengal and Orissa to Odisha
- Analyzed enrollment patterns by state and district
- Studied age group distributions
- Looked at monthly trends over time
- Compared age distributions across different states and time periods
- Converted all dates to proper datetime format

The Digital Native

Step 2 | Fraud Pattern Detection

- Compared enrollment data with demographic update data
- Calculated ratio of updates to enrollments
- Normal ratio should be between 1.0 and 2.0
- Flagged districts with ratio below 0.2 as potential ghost enrollments
- Flagged districts with ratio above 5.0 as maintenance mode
- Used Isolation Forest algorithm to find unusual patterns
- •Analyzed 7 features including age groups and percentages
- •Set to detect top 0.5% outliers
- •Classified anomalies into extreme volume adult spike youth overrepresentation and missing youth data

The Convenience Seeker

Step 3 | Statistical Validation

- Applied linear regression to identify enrollment trends over time
- Calculated Pearson correlation coefficients between age groups
- Used Z score method to statistically confirm outliers
- Performed distribution analysis on enrollment data
- Generated box plots to identify data spread and outliers
- Analyzed variance across different states and districts
- Validated anomaly detection results with statistical tests
- Cross referenced findings with demographic patterns

Workflow Results

Data Loading and Data Cleaning

Enrolment: 1,006,029 records, Dropped 22,957 duplicate rows, Removed 21 rows with invalid/missing data.

Demographic: 2,071,700 records, Dropped 473,601 duplicate rows, Removed 4 rows with invalid/missing data

Biometric: 1,861,108 records, Dropped 94,896 duplicate rows.

Data Formatting/Feature Engineering

Enrolment Data features

- **total_enrol:** The sum of all three age groups (0-5, 5-17, 18+). This represents the Total Enrollment Volume per record.
- **youth_pct:** The percentage of enrollments that are aged 5-17.
- **child_pct:** The percentage of enrollments that are infants (aged 0-5).
- **adult_pct:** The percentage of enrollments that are adults (aged 18+).
- **month:** The date converted into a monthly period format (e.g., 2025-03).
- **year_month:** The date formatted as a string string (e.g., "2025-03") for easier labeling in charts.

Demographic Features

- **total_demo**: The sum of the two demographic age groups (5-17 and 17+). This represents the Total Demographic Update Volume.
- **demo_youth_pct**: The percentage of demographic updates coming from the youth age group (5-17).
- **month**: Monthly period for time-series alignment.
- **year_month**: String formatted month-year.

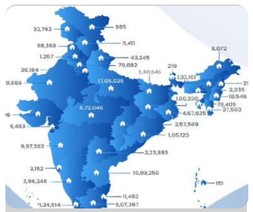
Biometric Features

- **total_bio:** The sum of the two biometric age groups. This represents the Total Biometric Re-capture Volume.
- **bio_youth_pct:** The percentage of biometric updates coming from the youth age group (5-17).
- **month:** Monthly period for time-series alignment.
- **year_month:** String formatted month-year.

Descriptive Analysis/EDA

- **Geographic Aggregation:** Grouped data by State and District to calculate volume rankings (Top 15/Bottom 10) and established a "Low Activity Threshold" (<100 records) to flag under-performing regions.
- **Demographic Normalization:** Converted raw age counts into Relative Percentages (0-100%) to enable fair comparison of age distributions.

Methodoly



The Deal Hunter

Step 4 | State-Wise Enrollment Analysis

Goal: Mapping the National Load We grouped nearly 5 million records by state to identify where the Aadhaar infrastructure is under the most pressure.

- **What we did**: After cleaning the inconsistent state names, we calculated the total activity for each region.
- **The Insight**: This allowed us to distinguish between "Saturation States" (like Uttar Pradesh and Bihar) where the work is purely maintenance, and "Catch-up States" (like Meghalaya) where new enrollments are still the priority.

The Digital Native

Step 5 | Age Distribution Segmentation

Goal: Understanding the Lifecycle of a User Instead of looking at a single total, we broke every transaction down into three key life stages: Infants (0-5), Students (5-17), and Adults (18+).

- **What we did**: We calculated the percentage of each age group participating in enrollment vs. updates.
- **The Insight**: This revealed the "Pediatric Shift"—proving that Aadhaar has transitioned from an adult identity project to a vital birth-registration and student-verification system.

The Convenience Seeker

Step 6 | Monthly Activity Trending

Goal: Finding Hidden Anomalies in the "Last Mile" Because state-level averages often hide problems, we zoomed into 700+ individual districts to find hyper-local outliers.

- **What we did**: We applied statistical filters to every district to check if their update-to-enrollment ratios were healthy.
- **The Insight**: This granular view was the "smoking gun" that helped us find Ghost Districts (where enrollment happens but data stops moving) and specific city-level failures like the Bengaluru Sync Gap.

Workflow Results

- **Temporal Trend Modeling:** Configured Month-over-Month (MoM) growth calculations and initialized Linear Regression slopes to quantify enrollment volatility and trend direction over the 9-month period.
- **Bivariate Matrices:** Prepared State × Age (Heatmaps) and Time × Age (Stacked Area) cross-tabulations to isolate specific demographic shifts (e.g., "Infant Bias") from general population noise.

Forensic & ML Analysis

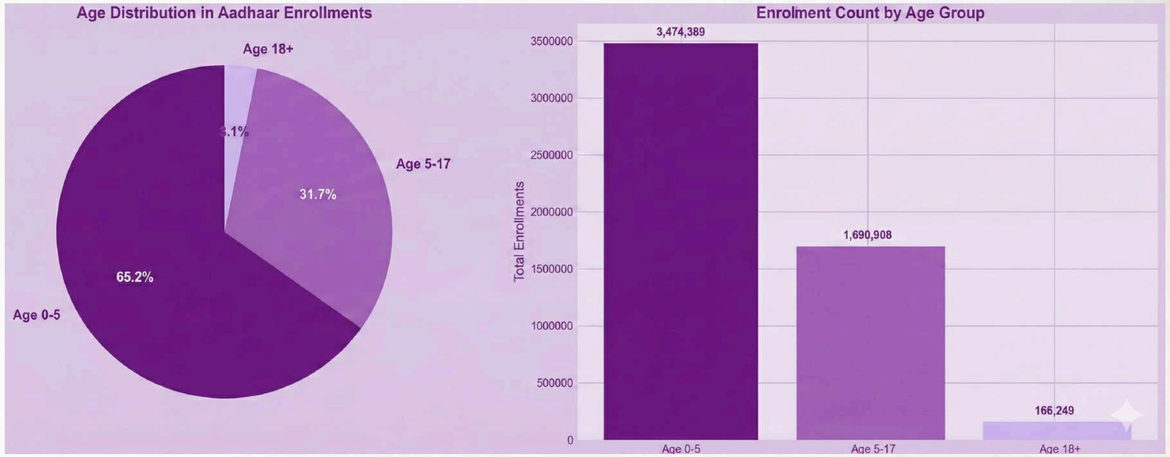
- **Cross-Dataset Integrity Check:** Merged Enrollment, Demographic, and Biometric datasets to calculate the Update-to-Enrollment Ratio. Flagged "Ghost Districts" where enrollment volume was high (>100) but demographic updates were near zero (<0.1 ratio).
- **Machine Learning Anomaly Detection:** Deployed an Isolation Forest algorithm (Contamination=0.5%) using 7 features (age groups, total volume, normalized percentages) to detect statistically deviant enrollment behaviors.
- **Fraud Pattern Classification:** Programmed logic to categorize anomalies into specific fraud types: "Extreme Volume" (mass camps), "Adult Spikes" (outliers in age distribution), and "Missing Youth Data" (incomplete records).
- **Forensic Verification:** Cross-referenced ML anomalies against the fraud patterns to identify dual-flagged regions (e.g., Bengaluru Urban), confirming systematic data pipeline failures rather than random noise.

Univariate Analysis: Age Distribution

1.The Demographic Shift

Key Findings: Aadhaar is Now a Birth Registry (Saturation Reached).

- **65.2% of all new enrollments are Infants (Age 0-5).**
- **Only 3.1% of enrollments are Adults (Age 18+).**
- **7:1 Ratio: For every 1 new ID created, 7.17 existing IDs are updated.**



An inverted pyramid: 65% of intake is infants, proving adult exclusion or saturation.

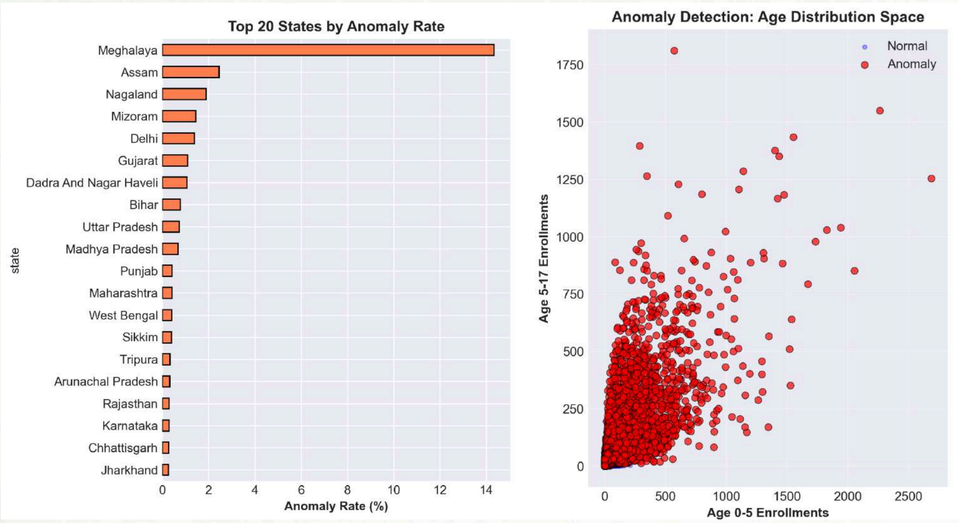
"The age pyramid has inverted. The system has shifted from a 'Growth Phase' to a 'Maintenance Phase.' Adult saturation is effectively 100%, transforming Aadhaar into a utility for maintaining adult records and a birth registry for newborns."

State-wise Stats

2.The Northeast Anomaly

Key Findings :- The "Exclusion Zone" in Meghalaya

- **14.3% Anomaly Rate: Meghalaya is the #1 outlier (6.5x the national average).**
- **32.1% Adult Enrollment: While the rest of India enrolls babies (3% adults), Meghalaya is enrolling adults (32%).**
- **1,184 "Adult Spike" Anomalies: ML models flagged a systematic pattern of adult catch-up in tribal regions.**



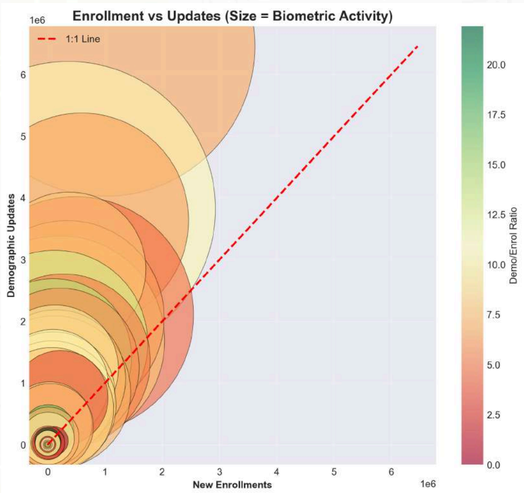
ML flags Meghalaya as the outlier: A systematic deviation from the national norm.

Cross-Dataset Integrity & Fraud Detection

3.Forensic Fraud Detection

Key Findings: Discovery of “Ghost Districts”

- **31 Districts identified with suspicious patterns (High Enrollment / Zero Updates).**
- **Bengaluru Urban Case: 23,074 New Enrollments but 0.0% Demographic Updates.**
- **Statistical Impossibility: Real users invariably update data (address/phone) within months of enrollment.**



“Ghost Districts are regions where high enrollment volumes occur without any corresponding demographic or biometric updates, indicating a lack of real human activity. This statistical impossibility suggests either a massive data synchronization failure or the creation of fraudulent identities.”

A scatter plot where X = New Enrollments and Y = Demographic Updates.

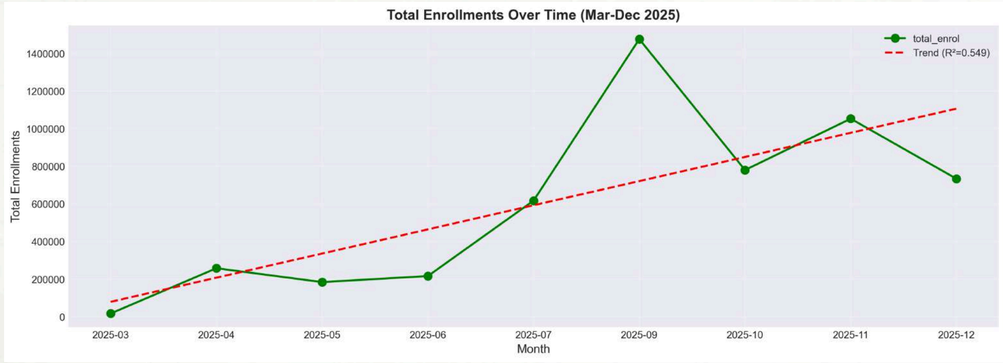
- Healthy Districts: Follow the diagonal red line (balanced growth).
- Ghost Districts: Hug the bottom X-axis (High Enrollment, Zero Updates).

The 'Rings' in the top-right chart represent States, sized by their Biometric activity. The forensic smoking gun is the 'Flatline' on the bottom axis—districts that are creating thousands of new IDs (moving right) but generating zero digital footprint (staying flat). That is the visual signature of the 'Ghost District' fraud pattern."

4.Operational Volatility

Key Findings :- The campaigns/drives to increase Aadhaar enrolment

- **3.5x Surge: Enrollments spiked from ~100k to 350k in July (Coordinated National Campaign).**
- **47% Crash: Activity plummeted from 1.5M in September to 0.8M in October.**
- **598 "Extreme Volume" Events: Single pincodes registering >300 people/day (68x normal).**



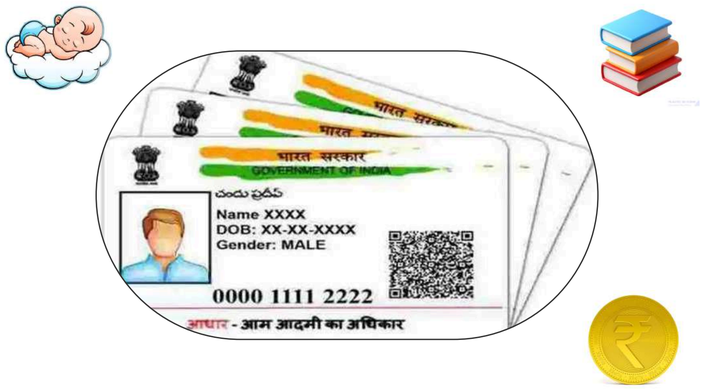
Extreme volatility confirms the system relies on ad-hoc camps rather than steady organic service.

Univariate Analysis: Monthly Enrollment Trend

The Maternity-First Strategy

Since Aadhaar is now primarily a birth-registration system, the current model of asking parents to visit enrollment centers is outdated and inefficient.

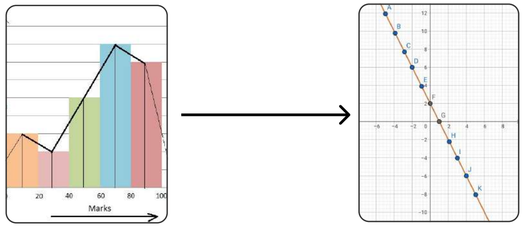
- We recommend shifting the enrollment focus from dedicated centers to Maternity Wards and Pediatric Clinics. By integrating Aadhaar registration into the birth certificate process, UIDAI can ensure 100% coverage from day one while reducing the burden on parents.



Stabilizing Urban Infrastructure

Chaotic "Campaign Mode" spikes in city hubs during July and October.Relying on temporary "camps" to handle surges creates technical bottlenecks and long wait times for citizens

- High-demand urban centers need a transition from temporary camps to Permanent Aadhaar Seva Kendras (ASKs). These permanent hubs should be equipped with higher server capacity to handle predictable seasonal rushes, ensuring the system doesn't crash during peak admission or tax months.



Moving Toward a "Service Gateway"

Aadhaar is no longer a card you "get"; it is a service you "use" throughout your life.

- The system must pivot from a registration platform to a Unified Service Interface. Every lifecycle event—like a student's mandatory biometric update at age 15 or an adult's address change—should be as seamless as a bank transaction. This reduces the friction between the citizen and government welfare.

Real-Time Integrity Audits

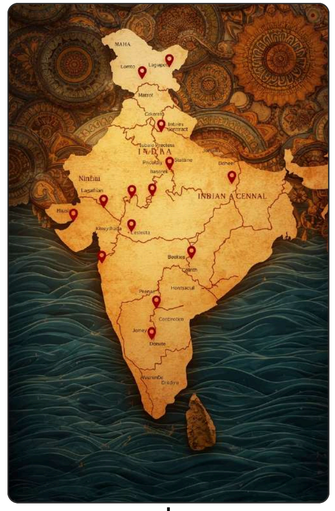
Traditional manual audits are too slow to catch local data glitches or potential fraud

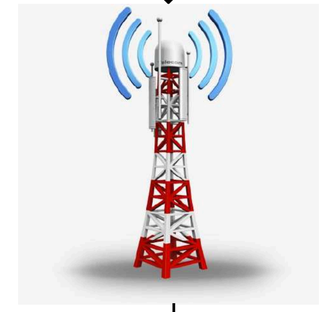
- UIDAI should implement an Automated Anomaly Dashboard. If a district shows high enrollment but zero updates (the "Ghost District" pattern) or a massive sync failure like the one found in Bengaluru, the system should automatically trigger a technical review within 48 hours to fix the data pipeline.

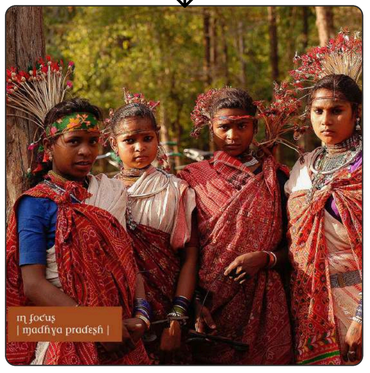
Closing the Northeast Adult Gap

Meghalaya having a 10x higher adult enrollment rate than the national average.The "one-size-fits-all" rules for adult enrollment may be creating barriers for tribal or rural populations in the Northeast

- We need to deploy Adult-Focused Mobile Units specifically in the Northeast. These units should be designed for "last-mile" inclusion, using local languages and simplified verification processes to help previously excluded populations get their first digital identity.







Summary

Our analysis of 4.9M records reveals Aadhaar's shift into a pediatric birth registry, while identifying regional exclusion gaps and urban sync failures that necessitate a move toward automated, hospital-linked, and permanent infrastructure.

Transition to Aadhaar 2.0

- **Strategic Shift:**
Aadhaar has evolved from an "identity creation" project to a "maintenance infrastructure."
- **Objective:**
The focus is no longer on onboarding the masses but on managing the Digital Lifecycle of existing users.
- **Analysis Goal:**
Identifying how the system handles the transition from creation to long-term data accuracy.

4.9 Million Lifecycle Points

- **Strategic Shift:**
Analyzed a massive dataset of 4,939,396 records from March to December 2025.
- **Objective:**
Successfully synchronized three separate datasets: Enrollment, Demographics, and Biometrics.
- **Analysis Goal:**
This integration allowed the tracking of how a citizen's data moves (or gets stuck) across different system stages.

Fixing Data Fragmentation

- **Strategic Shift:**
Identified significant data "deformation" where regional names were spelled in dozens of different ways.
- **Objective:**
Built an engine to merge fragmented entries (e.g., West Bangal, Westbengal, West Bengal) into a single source of truth.
- **Analysis Goal:**
This step prevented regional trends from being split or undervalued by an estimated 15%.

The "Three-Speed" Ecosystem

- **Regional Divergence:**
India does not have one Aadhaar trend; it operates at three distinct speeds.
- **Mainland India:**
Functioning primarily as a Pediatric Registry.
- **Urban Hubs:**
Operating in a volatile "Campaign Mode" driven by surges.

Aadhaar as a Birth Registry

- **Age Anomaly:**
65.2% of all new enrollments are now in the 0–5 age group.
- **Adult Saturation:**
Adult enrollment (18+) has dropped to just 3.1%, indicating near-total coverage of the eligible population.
- **Systemic Change:**
Aadhaar has effectively become a digital extension of birth registration in the mainland.

The Meghalaya Outlier

- **Adult Surge**
Meghalaya shows an adult enrollment rate 10 times higher than the national average.
- **Exclusion Risk:**
32% of new IDs in this region are still issued to adults, indicating a late-stage inclusion phase
- **Policy Need:**
This area requires adult-focused outreach rather than the pediatric-focused services used elsewhere.

The "Campaign Mode" Problem

- **Infrastructure Stress:**
Detected 596 "Extreme Volume" events where activity spiked up to 68x the normal load.
- **Seasonal Spikes:**
Massive surges in July and October suggest a heavy reliance on temporary "cams" rather than permanent centers.
- **Result:**
This leads to system crashes, long queues, and service inequality for city dwellers.

The Integrity Red Flag

- **Anomalous Silence:**
Identified 31 "Ghost Districts" with high enrollment numbers but zero demographic or biometric updates.
- **Failure Risk:**
This data silence is a major indicator of either local system failures or potential fraudulent bulk entries.
- **Urgency:**
These districts require immediate physical audits to verify the authenticity of records.

Citizens in Digital Limbo

- **Pipeline Leak:**
Even in a tech hub, 23,074 citizens in Bengaluru Urban were found to be stuck in "Digital Limbo."
- **Sync Failure::**
These individuals successfully enrolled, but their records never synchronized with the demographic or biometric databases.
- **Impact:**
This prevents citizens from accessing government benefits linked to their digital ID.

In [ ]:

```
pip install pandas numpy matplotlib seaborn scikit-learn scipy plotly geopandas
```

```
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.13/site-packages (2.
3.1)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.13/site-packages (2.
2.5)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.13/site-packages
(3.10.0)
Requirement already satisfied: seaborn in /opt/anaconda3/lib/python3.13/site-packages
(0.13.2)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.13/site-packag
es (1.6.1)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.13/site-packages (1.1
6.0)
Requirement already satisfied: plotly in /opt/anaconda3/lib/python3.13/site-packages (6.
0.1)
Requirement already satisfied: geopandas in /opt/anaconda3/lib/python3.13/site-packages
(1.1.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.13/s
ite-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.13/site-packag
es (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python3.13/site-pack
ages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/lib/python3.13/site-pa
ckages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/python3.13/site-packag
es (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/lib/python3.13/site-p
ackages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/anaconda3/lib/python3.13/site-p
ackages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.13/site-pac
kages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /opt/anaconda3/lib/python3.13/site-packages
(from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/lib/python3.13/site-pa
ckages (from matplotlib) (3.2.0)
Requirement already satisfied: joblib>=1.2.0 in /opt/anaconda3/lib/python3.13/site-packa
ges (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /opt/anaconda3/lib/python3.13/sit
e-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: narwhals>=1.15.1 in /opt/anaconda3/lib/python3.13/site-pa
ckages (from plotly) (1.31.0)
Requirement already satisfied: pyogrio>=0.7.2 in /opt/anaconda3/lib/python3.13/site-pack
ages (from geopandas) (0.12.1)
Requirement already satisfied: pyproj>=3.5.0 in /opt/anaconda3/lib/python3.13/site-packa
ges (from geopandas) (3.7.2)
Requirement already satisfied: shapely>=2.0.0 in /opt/anaconda3/lib/python3.13/site-pack
ages (from geopandas) (2.1.2)
Requirement already satisfied: certifi in /opt/anaconda3/lib/python3.13/site-packages (f
rom pyogrio>=0.7.2->geopandas) (2025.11.12)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.13/site-packages
(from python-dateutil>=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# For anomaly detection
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from scipy.spatial.distance import cosine
from scipy import stats

# For time series
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.seasonal import seasonal_decompose
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
%matplotlib inline
```

In [ ]:

```
print(" Loading Enrollment Data...")
enrol_files = [
    'aadhar enrollment.csv',
    'aadhar enrollment 2.csv',
    'aadhar enrollment 3.csv'
]

enrol_dfs = []
for file in enrol_files:
    df = pd.read_csv(file)
    enrol_dfs.append(df)
    print(f" Loaded {file}: {len(df):,} rows")

enrol = pd.concat(enrol_dfs, ignore_index=True)
print(f"\nTotal Enrollment records: {len(enrol):,}\n")

# Load Demographic data (5 files)
print(" Loading Demographic Data...")
demo_files = [
    'demograph.csv',
    'demograph2.csv',
    'demograph3.csv',
    'demograph4.csv',
    'demograph5.csv'
]

demo_dfs = []
for file in demo_files:
    df = pd.read_csv(file)
    demo_dfs.append(df)
    print(f" Loaded {file}: {len(df):,} rows")

demo = pd.concat(demo_dfs, ignore_index=True)
print(f"\n Total Demographic records: {len(demo):,}\n")
```

```
# Load Biometric data (4 files)
print(" Loading Biometric Data...")
bio_files = [
    'biometric.csv',
    'biometric1.csv',
    'biometric3.csv',
    'biometric4.csv'
]

bio_dfs = []
for file in bio_files:
    df = pd.read_csv(file)
    bio_dfs.append(df)
    print(f" Loaded {file}: {len(df):,} rows")

bio = pd.concat(bio_dfs, ignore_index=True)
print(f"\nTotal Biometric records: {len(bio):,}\n")


print(f" SUMMARY:")
print(f"   Enrollment: {len(enrol):,} records")
print(f"   Demographic: {len(demo):,} records")
print(f"   Biometric: {len(bio):,} records")
```

```
 Loading Enrollment Data...
 Loaded aadhar enrollment.csv: 6,029 rows
 Loaded aadhar enrollment 2.csv: 500,000 rows
 Loaded aadhar enrollment 3.csv: 500,000 rows

Total Enrollment records: 1,006,029

 Loading Demographic Data...
 Loaded demograph.csv: 71,700 rows
 Loaded demograph2.csv: 500,000 rows
 Loaded demograph3.csv: 500,000 rows
 Loaded demograph4.csv: 500,000 rows
 Loaded demograph5.csv: 500,000 rows

 Total Demographic records: 2,071,700

 Loading Biometric Data...
 Loaded biometric.csv: 361,108 rows
 Loaded biometric1.csv: 500,000 rows
 Loaded biometric3.csv: 500,000 rows
 Loaded biometric4.csv: 500,000 rows

Total Biometric records: 1,861,108

 SUMMARY:
   Enrollment: 1,006,029 records
   Demographic: 2,071,700 records
   Biometric: 1,861,108 records
```

In [ ]:

```
print("📑 ENROLLMENT DATA STRUCTURE")
print(f"Shape: {enrol.shape}")
print(f"\nColumns: {list(enrol.columns)}")
print(f"\nData types:\n{enrol.dtypes}")
```

```
print(f"\nFirst 5 rows:")
display(enrol.head())
print(f"\nBasic stats:")
display(enrol.describe())

print("📋 DEMOGRAPHIC DATA STRUCTURE")

print(f"Shape: {demo.shape}")
print(f"\nColumns: {list(demo.columns)}")
display(demo.head())


print("📋 BIOMETRIC DATA STRUCTURE")
print(f"Shape: {bio.shape}")
print(f"\nColumns: {list(bio.columns)}")
display(bio.head())
```

```
📋 ENROLLMENT DATA STRUCTURE
Shape: (1006029, 7)

Columns: ['date', 'state', 'district', 'pincode', 'age_0_5', 'age_5_17', 'age_18_greate
r']

Data types:
date              object
state             object
district          object
pincode            int64
age_0_5            int64
age_5_17           int64
age_18_greater     int64
dtype: object

First 5 rows:
```

	date	state	district	pincode	age_0_5	age_5_17	age_18_greater
0	31-12-2025	Karnataka	Bidar	585330	2	3	0
1	31-12-2025	Karnataka	Bidar	585402	6	0	0
2	31-12-2025	Karnataka	Bidar	585413	1	0	0
3	31-12-2025	Karnataka	Bidar	585418	1	2	0
4	31-12-2025	Karnataka	Bidar	585421	4	3	0

```
Basic stats:
```

	pincode	age_0_5	age_5_17	age_18_greater
count	1.006029e+06	1.006029e+06	1.006029e+06	1.006029e+06
mean	5.186415e+05	3.525709e+00	1.710074e+00	1.673441e-01
std	2.056360e+05	1.753851e+01	1.436963e+01	3.220525e+00
min	1.000000e+05	0.000000e+00	0.000000e+00	0.000000e+00
25%	3.636410e+05	1.000000e+00	0.000000e+00	0.000000e+00
50%	5.174170e+05	2.000000e+00	0.000000e+00	0.000000e+00
75%	7.001040e+05	3.000000e+00	1.000000e+00	0.000000e+00

	pincode	age_0_5	age_5_17	age_18_greater
max	8.554560e+05	2.688000e+03	1.812000e+03	8.550000e+02

📋 DEMOGRAPHIC DATA STRUCTURE
Shape: (2071700, 6)

Columns: ['date', 'state', 'district', 'pincode', 'demo_age_5_17', 'demo_age_17_']

	date	state	district	pincode	demo_age_5_17	demo_age_17_
0	16-12-2025	Madhya Pradesh	Shajapur	465113	0	4
1	16-12-2025	Madhya Pradesh	Shajapur	465226	2	14
2	16-12-2025	Madhya Pradesh	Shajapur	465339	1	2
3	16-12-2025	Madhya Pradesh	Shajapur	465447	1	19
4	16-12-2025	Madhya Pradesh	Sheopur	476332	11	25

📋 BIOMETRIC DATA STRUCTURE
Shape: (1861108, 6)

Columns: ['date', 'state', 'district', 'pincode', 'bio_age_5_17', 'bio_age_17_']

	date	state	district	pincode	bio_age_5_17	bio_age_17_
0	08-12-2025	Uttar Pradesh	Moradabad	244411	4	2
1	08-12-2025	Uttar Pradesh	Moradabad	244601	14	19
2	08-12-2025	Uttar Pradesh	Muzaffarnagar	251002	22	37
3	08-12-2025	Uttar Pradesh	Muzaffarnagar	251202	4	4
4	08-12-2025	Uttar Pradesh	Muzaffarnagar	251319	4	7

In [ ]:

```
import pandas as pd
import numpy as np

print(" STARTING ROBUST DATA CLEANING...\n")

def clean_dataframe(df, df_name):
    print(f"Cleaning {df_name}...")
    original_count = len(df)

    # 1. Deduplication (Critical for large datasets)
    df.drop_duplicates(inplace=True)
    dedupe_count = len(df)
    if original_count > dedupe_count:
        print(f"   Dropped {original_count - dedupe_count:,} duplicate rows")

    # 2. Convert Dates
    df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y', errors='coerce')

    # 3. Standardize Case (Handle "West Bengal ", "west bengal")
    # Convert to Title Case and strip whitespace
    df['state'] = df['state'].astype(str).str.strip().str.title()
    df['district'] = df['district'].astype(str).str.strip().str.title()

    # 4. Remove Non-Geographic Values / Cross-Column Leaks
```

```python
    # Filter out rows where 'state' is a number (e.g., "100000")
    df = df[~df['state'].str.match(r'^\d+$')]

    # Filter out known districts that appear in the state column
    invalid_states = ['Darbhanga', 'Puttenahalli', 'Nan']
    df = df[~df['state'].isin(invalid_states)]

    # 5. COMPREHENSIVE STATE MAPPING (Fixing Typos & Legacy Names)
    state_mapping = {
        # Typos & Variations
        'West Bangal': 'West Bengal',
        'Westbengal': 'West Bengal',
        'West Bengli': 'West Bengal',
        'Wb': 'West Bengal',
        'Jammu&Kashmir': 'Jammu And Kashmir',
        'Jammu & Kashmir': 'Jammu And Kashmir',
        'Nct Of Delhi': 'Delhi',

        # Conjunction Differences
        'Andaman & Nicobar Islands': 'Andaman And Nicobar Islands',
        'Dadra & Nagar Haveli': 'Dadra And Nagar Haveli',
        'Daman & Diu': 'Daman And Diu',

        # Combined Territories (Standardizing to the new official merged name if you pre
        # OR keeping them separate. Here we keep separate for historical consistency
        # unless you want to merge them into "Dadra And Nagar Haveli And Daman And Diu")
        'The Dadra And Nagar Haveli And Daman And Diu': 'Dadra And Nagar Haveli',

        # Legacy Names (Standardize to CURRENT official names)
        'Orissa': 'Odisha',
        'Pondicherry': 'Puducherry',
        'Uttaranchal': 'Uttarakhand'
    }
    df['state'] = df['state'].replace(state_mapping)
    print(f"  Standardized State names (Fixed 'West Bengli', 'Orissa', etc.)")

    # 6. COMPREHENSIVE DISTRICT MAPPING
    district_mapping = {
        # Compounded vs Separated
        'Yamunanagar': 'Yamuna Nagar',
        'Karimnagar': 'Karim Nagar',
        'Ahmednagar': 'Ahmed Nagar',
        'Mahbubnagar': 'Mahabub Nagar',
        'Mahabubnagar': 'Mahabub Nagar',

        # Spelling Variations
        'Chamrajanagar': 'Chamarajanagar',
        'Villupuram': 'Viluppuram',
        'Thiruvananthapuram': 'Thiruvananthpuram',

        # Word Substitutions
        'South 24 Parganas': 'South Twenty Four Parganas',
        'North 24 Parganas': 'North Twenty Four Parganas'
    }
    df['district'] = df['district'].replace(district_mapping)
    print(f" Standardized District names (Fixed 'Yamunanagar', 'South 24 Parganas')")

    # 7. Final Clean-up: Drop Invalid Dates or Empty States
    before = len(df)
```

```
    df = df.dropna(subset=['state', 'district', 'date'])
    # Ensure pincode is string (preserves leading zeros if present)
    if 'pincode' in df.columns:
         df['pincode'] = df['pincode'].astype(str).str.replace(r'\.0$', '', regex=True)

    after = len(df)
    if dedupe_count > after:
        print(f" Removed {dedupe_count - after:,} rows with invalid/missing data")

    return df

# Execute the new cleaning function
enrol = clean_dataframe(enrol, "ENROLLMENT")
demo = clean_dataframe(demo, "DEMOGRAPHIC")
bio = clean_dataframe(bio, "BIOMETRIC")

print("\n" + "="*60)
print(" ROBUST DATA CLEANING COMPLETE!")
print("="*60)
```

```
 STARTING ROBUST DATA CLEANING...

Cleaning ENROLLMENT...
   Dropped 22,957 duplicate rows
  Standardized State names (Fixed 'West Bengli', 'Orissa', etc.)
 Standardized District names (Fixed 'Yamunanagar', 'South 24 Parganas')
 Removed 21 rows with invalid/missing data
Cleaning DEMOGRAPHIC...
   Dropped 473,601 duplicate rows
  Standardized State names (Fixed 'West Bengli', 'Orissa', etc.)
 Standardized District names (Fixed 'Yamunanagar', 'South 24 Parganas')
 Removed 4 rows with invalid/missing data
Cleaning BIOMETRIC...
   Dropped 94,896 duplicate rows
  Standardized State names (Fixed 'West Bengli', 'Orissa', etc.)
 Standardized District names (Fixed 'Yamunanagar', 'South 24 Parganas')

============================================================
 ROBUST DATA CLEANING COMPLETE!
============================================================
```

In [ ]:

```
print(" FEATURE ENGINEERING...\n")

# ENROLLMENT: Create total and derived columns
enrol['total_enrol'] = enrol['age_0_5'] + enrol['age_5_17'] + enrol['age_18_greater']
enrol['youth_pct'] = (enrol['age_5_17'] / enrol['total_enrol'] * 100).round(2)
enrol['child_pct'] = (enrol['age_0_5'] / enrol['total_enrol'] * 100).round(2)
enrol['adult_pct'] = (enrol['age_18_greater'] / enrol['total_enrol'] * 100).round(2)
enrol['month'] = enrol['date'].dt.to_period('M')
enrol['year_month'] = enrol['date'].dt.strftime('%Y-%m')
print("✅ Enrollment features created")

# DEMOGRAPHIC: Create total and derived columns
demo['total_demo'] = demo['demo_age_5_17'] + demo['demo_age_17_']
demo['demo_youth_pct'] = (demo['demo_age_5_17'] / demo['total_demo'] * 100).round(2)
demo['month'] = demo['date'].dt.to_period('M')
demo['year_month'] = demo['date'].dt.strftime('%Y-%m')
print("✅ Demographic features created")
```

```
# BIOMETRIC: Create total and derived columns
bio['total_bio'] = bio['bio_age_5_17'] + bio['bio_age_17_']
bio['bio_youth_pct'] = (bio['bio_age_5_17'] / bio['total_bio'] * 100).round(2)
bio['month'] = bio['date'].dt.to_period('M')
bio['year_month'] = bio['date'].dt.strftime('%Y-%m')
print("✅ Biometric features created")

# Display date ranges
print("\n" + "="*60)
print("📅 DATE RANGES:")
print(f"Enrollment: {enrol['date'].min()} to {enrol['date'].max()}")
print(f"Demographic: {demo['date'].min()} to {demo['date'].max()}")
print(f"Biometric: {bio['date'].min()} to {bio['date'].max()}")
print("="*60)

# Display sample
print("\n📊 Sample of cleaned enrollment data:")
display(enrol[['date', 'state', 'district', 'total_enrol', 'youth_pct', 'child_pct', 'ad
```

```
 FEATURE ENGINEERING...

✅ Enrollment features created
✅ Demographic features created
✅ Biometric features created

============================================================
📅 DATE RANGES:
Enrollment: 2025-03-02 00:00:00 to 2025-12-31 00:00:00
Demographic: 2025-03-01 00:00:00 to 2025-12-29 00:00:00
Biometric: 2025-03-01 00:00:00 to 2025-12-29 00:00:00
============================================================

📊 Sample of cleaned enrollment data:
```

	date	state	district	total_enrol	youth_pct	child_pct	adult_pct
0	2025-12-31	Karnataka	Bidar	5	60.00	40.00	0.0
1	2025-12-31	Karnataka	Bidar	6	0.00	100.00	0.0
2	2025-12-31	Karnataka	Bidar	1	0.00	100.00	0.0
3	2025-12-31	Karnataka	Bidar	3	66.67	33.33	0.0
4	2025-12-31	Karnataka	Bidar	7	42.86	57.14	0.0

In [ ]:

```
print("🔍 DATA QUALITY CHECK\n")

def data_quality_report(df, df_name):
    print("="*60)
    print(f"📊 {df_name} DATA QUALITY REPORT")
    print("="*60)

    # Missing values
    missing = df.isnull().sum()
    missing_pct = (missing / len(df) * 100).round(2)
    missing_df = pd.DataFrame({
        'Missing Count': missing,
        'Percentage': missing_pct
```

```
    })
    missing_df = missing_df[missing_df['Missing Count'] > 0].sort_values('Missing Count'

    if len(missing_df) > 0:
        print("\n⚠  MISSING VALUES:")
        display(missing_df)
    else:
        print("\n✅ No missing values found!")

    # Duplicate rows
    duplicates = df.duplicated().sum()
    print(f"\n🔄 Duplicate rows: {duplicates:,}")

    # Zero values check
    if 'total_enrol' in df.columns:
        zeros = (df['total_enrol'] == 0).sum()
        print(f"⚠  Rows with zero total enrollment: {zeros:,} ({zeros/len(df)*100:.2f}%)

    print()

# Run quality checks
data_quality_report(enrol, "ENROLLMENT")
data_quality_report(demo, "DEMOGRAPHIC")
data_quality_report(bio, "BIOMETRIC")
```

```
🔍 DATA QUALITY CHECK

============================================================
📊 ENROLLMENT DATA QUALITY REPORT
============================================================

✅ No missing values found!

🔄 Duplicate rows: 1,229
⚠  Rows with zero total enrollment: 0 (0.00%)

============================================================
📊 DEMOGRAPHIC DATA QUALITY REPORT
============================================================

⚠  MISSING VALUES:
```

	Missing Count	Percentage
demo_youth_pct	1654	0.1

```
🔄 Duplicate rows: 1,432

============================================================
📊 BIOMETRIC DATA QUALITY REPORT
============================================================

⚠  MISSING VALUES:
```

	Missing Count	Percentage
bio_youth_pct	11	0.0

```
🔄 Duplicate rows: 1,089
```

In [ ]:

```
# Quick Fix for Division-by-Zero NaNs
demo['demo_youth_pct'] = demo['demo_youth_pct'].fillna(0)
bio['bio_youth_pct'] = bio['bio_youth_pct'].fillna(0)

print("✅ Fixed NaN values in Demographic and Biometric percentages.")
```

✅ Fixed NaN values in Demographic and Biometric percentages.

In [ ]:

```
print(" UNIVARIATE ANALYSIS: STATE-WISE ENROLLMENT")

# Aggregate by state
state_enrol = enrol.groupby('state').agg({
    'total_enrol': 'sum',
    'age_0_5': 'sum',
    'age_5_17': 'sum',
    'age_18_greater': 'sum'
}).sort_values('total_enrol', ascending=False)

# Calculate percentages
state_enrol['child_pct'] = (state_enrol['age_0_5'] / state_enrol['total_enrol'] * 100).r
state_enrol['youth_pct'] = (state_enrol['age_5_17'] / state_enrol['total_enrol'] * 100).
state_enrol['adult_pct'] = (state_enrol['age_18_greater'] / state_enrol['total_enrol'] *

print("\n📈 TOP 15 STATES BY ENROLLMENT:")
display(state_enrol.head(15))

print("\n📉 BOTTOM 10 STATES BY ENROLLMENT:")
display(state_enrol.tail(10))

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Top 15 states
state_enrol['total_enrol'].head(15).plot(kind='barh', ax=axes[0], color='steelblue')
axes[0].set_title('Top 15 States by Total Enrollment', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Total Enrollments', fontsize=12)
axes[0].invert_yaxis()

# Bottom 10 states
state_enrol['total_enrol'].tail(10).plot(kind='barh', ax=axes[1], color='coral')
axes[1].set_title('Bottom 10 States by Total Enrollment', fontsize=14, fontweight='bold'
axes[1].set_xlabel('Total Enrollments', fontsize=12)
axes[1].invert_yaxis()

plt.tight_layout()
plt.savefig('state_wise_enrollment.png', dpi=300, bbox_inches='tight')
plt.show()

# Summary statistics
print("\n" + "="*60)
print("📊 SUMMARY STATISTICS:")
print(f"Total states: {len(state_enrol)}")
print(f"Total enrollments: {state_enrol['total_enrol'].sum():,}")
print(f"Average per state: {state_enrol['total_enrol'].mean():,.0f}")
print(f"Median per state: {state_enrol['total_enrol'].median():,.0f}")
print(f"Std deviation: {state_enrol['total_enrol'].std():,.0f}")
print("="*60)
```

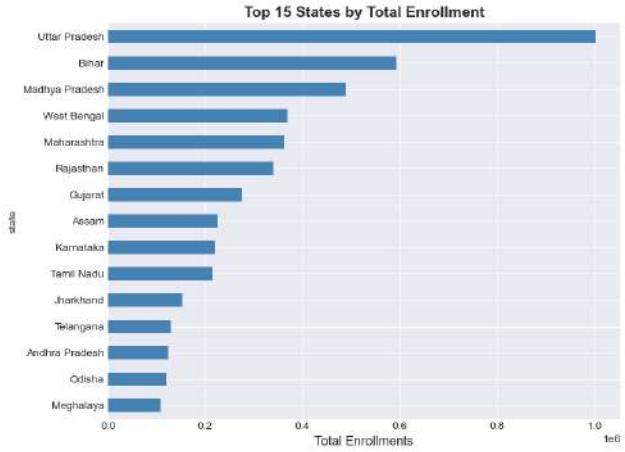
UNIVARIATE ANALYSIS: STATE-WISE ENROLLMENT

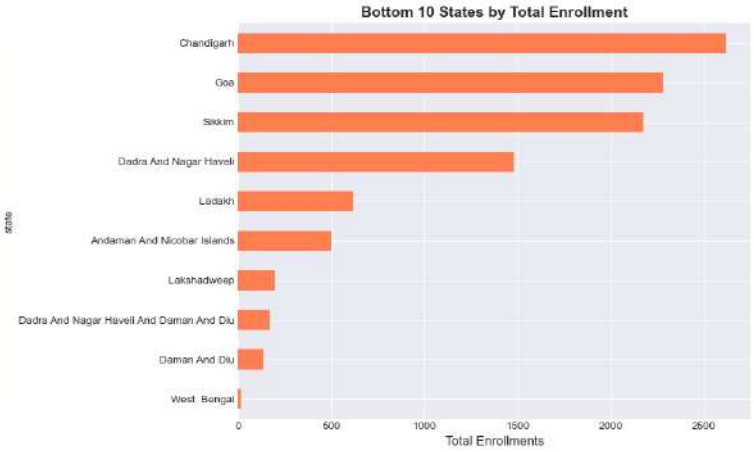
📈 TOP 15 STATES BY ENROLLMENT:

state	total_enrol	age_0_5	age_5_17	age_18_greater	child_pct	youth_pct	adult_pct
Uttar Pradesh	1002631	511727	473205	17699	51.04	47.20	1.77
Bihar	593753	254911	327043	11799	42.93	55.08	1.99
Madhya Pradesh	487892	363244	115172	9476	74.45	23.61	1.94
West Bengal	369234	270410	90329	8495	73.24	24.46	2.30
Maharashtra	363446	274274	81069	8103	75.46	22.31	2.23
Rajasthan	340591	224977	110131	5483	66.05	32.34	1.61
Gujarat	275042	188709	70270	16063	68.61	25.55	5.84
Assam	225359	137970	64834	22555	61.22	28.77	10.01
Karnataka	219618	176178	33402	10038	80.22	15.21	4.57
Tamil Nadu	215710	178294	36214	1202	82.65	16.79	0.56
Jharkhand	153612	96048	56152	1412	62.53	36.55	0.92
Telangana	128948	103768	24035	1145	80.47	18.64	0.89
Andhra Pradesh	124273	109394	13414	1465	88.03	10.79	1.18
Odisha	120454	97500	22228	726	80.94	18.45	0.60
Meghalaya	109239	21072	53089	35078	19.29	48.60	32.11

📉 BOTTOM 10 STATES BY ENROLLMENT:

state	total_enrol	age_0_5	age_5_17	age_18_greater	child_pct	youth_pct	adult_pct
Chandigarh	2620	2377	210	33	90.73	8.02	1.26
Goa	2280	1871	253	156	82.06	11.10	6.84
Sikkim	2175	1040	1030	105	47.82	47.36	4.83
Dadra And Nagar Haveli	1479	1234	214	31	83.43	14.47	2.10
Ladakh	617	466	133	18	75.53	21.56	2.92
Andaman And Nicobar Islands	501	469	32	0	93.61	6.39	0.00
Lakshadweep	199	188	10	1	94.47	5.03	0.50
Dadra And Nagar Haveli And Daman And Diu	169	130	20	19	76.92	11.83	11.24
Daman And Diu	134	120	14	0	89.55	10.45	0.00
West Bengal	15	9	6	0	60.00	40.00	0.00





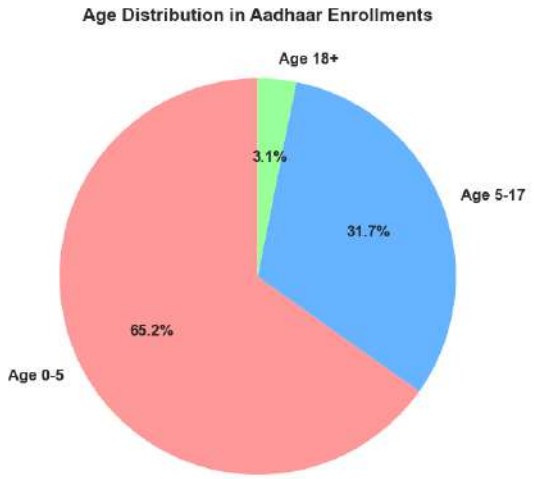

```
============================================================
📊 SUMMARY STATISTICS:
Total states: 39
Total enrollments: 5,331,546
Average per state: 136,706
Median per state: 73,950
Std deviation: 205,073
============================================================
```

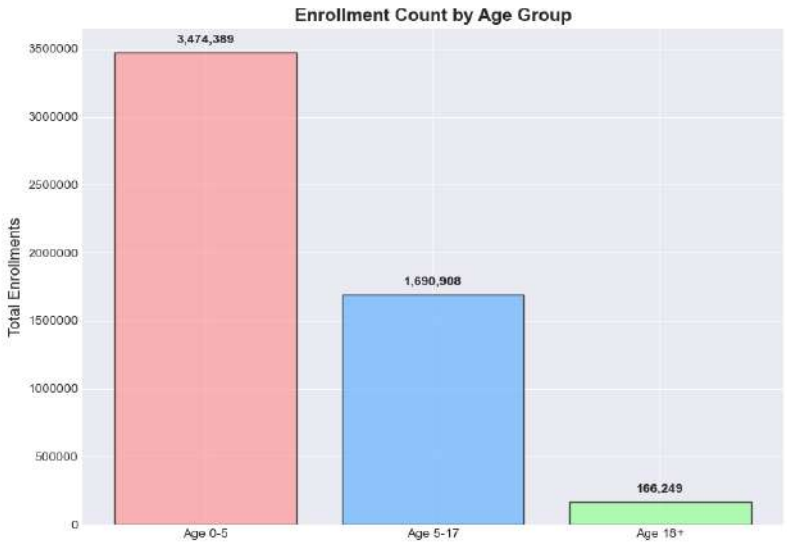
In [ ]:

```
print("="*60)
print("📊 UNIVARIATE ANALYSIS: AGE DISTRIBUTION")
print("="*60)

# Calculate total by age group
age_totals = {
    'Age 0-5': enrol['age_0_5'].sum(),
    'Age 5-17': enrol['age_5_17'].sum(),
    'Age 18+': enrol['age_18_greater'].sum()
}

total = sum(age_totals.values())

print("\n📊 AGE GROUP DISTRIBUTION:")
for age_group, count in age_totals.items():
    pct = count / total * 100
    print(f"{age_group:12s}: {count:12,} ({pct:5.2f}%)")

print(f"\n{'Total':12s}: {total:12,} (100.00%)")

# 🚨 CRITICAL FINDING
print("\n" + "="*60)
print("🚨 CRITICAL FINDING: AGE DISTRIBUTION ANOMALY!")
print("="*60)
child_pct = age_totals['Age 0-5'] / total * 100
if child_pct > 50:
    print(f"⚠️  Age 0-5 represents {child_pct:.1f}% of enrollments!")
    print(f"⚠️  Normal population pyramid: 20-30% are children")
    print(f"⚠️  This is a {child_pct/30:.1f}x OVER-REPRESENTATION")
    print(f"\n💡 POSSIBLE CAUSES:")
    print(f"   1. Focus on newborn Aadhaar for welfare schemes")
    print(f"   2. Lack of follow-up enrollment after age 5")
    print(f"   3. Data collection bias toward birth registrations")
print("="*60)
```

```python
# Visualize
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Pie chart
colors = ['#ff9999', '#66b3ff', '#99ff99']
axes[0].pie(age_totals.values(), labels=age_totals.keys(), autopct='%1.1f%%',
            startangle=90, colors=colors, textprops={'fontsize': 12, 'fontweight': 'bold'
axes[0].set_title('Age Distribution in Aadhaar Enrollments', fontsize=14, fontweight='bo

# Bar chart
ages = list(age_totals.keys())
counts = list(age_totals.values())
axes[1].bar(ages, counts, color=colors, alpha=0.7, edgecolor='black')
axes[1].set_ylabel('Total Enrollments', fontsize=12)
axes[1].set_title('Enrollment Count by Age Group', fontsize=14, fontweight='bold')
axes[1].ticklabel_format(axis='y', style='plain')
for i, v in enumerate(counts):
    axes[1].text(i, v + max(counts)*0.02, f'{v:,}', ha='center', fontweight='bold')

plt.tight_layout()
plt.savefig('age_distribution.png', dpi=300, bbox_inches='tight')
plt.show()
```

```
============================================================
📊 UNIVARIATE ANALYSIS: AGE DISTRIBUTION
============================================================

📊 AGE GROUP DISTRIBUTION:
Age 0-5     :    3,474,389 (65.17%)
Age 5-17    :    1,690,908 (31.72%)
Age 18+     :      166,249 ( 3.12%)

Total       :    5,331,546 (100.00%)

============================================================
🚨 CRITICAL FINDING: AGE DISTRIBUTION ANOMALY!
============================================================
⚠  Age 0-5 represents 65.2% of enrollments!
⚠  Normal population pyramid: 20-30% are children
⚠  This is a 2.2x OVER-REPRESENTATION

💡 POSSIBLE CAUSES:
   1. Focus on newborn Aadhaar for welfare schemes
   2. Lack of follow-up enrollment after age 5
   3. Data collection bias toward birth registrations
============================================================
```





In [ ]:

```
print("="*60)
print("📊 UNIVARIATE ANALYSIS: MONTHLY ENROLLMENT TREND")
print("="*60)

# Monthly aggregation
monthly_enrol = enrol.groupby('year_month').agg({
    'total_enrol': 'sum',
    'age_0_5': 'sum',
    'age_5_17': 'sum',
    'age_18_greater': 'sum'
}).sort_index()

print("\n📅 MONTHLY ENROLLMENT DATA:")
display(monthly_enrol)

# Calculate month-over-month growth
monthly_enrol['mom_growth'] = monthly_enrol['total_enrol'].pct_change() * 100
monthly_enrol['mom_growth'] = monthly_enrol['mom_growth'].round(2)

print("\n📈 MONTH-OVER-MONTH GROWTH RATE:")
display(monthly_enrol[['total_enrol', 'mom_growth']])

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(14, 10))

# Line chart - total enrollments
monthly_enrol['total_enrol'].plot(ax=axes[0], marker='o', linewidth=2, markersize=8, col
axes[0].set_title('Total Enrollments Over Time (Mar-Dec 2025)', fontsize=14, fontweight=
axes[0].set_ylabel('Total Enrollments', fontsize=12)
axes[0].set_xlabel('Month', fontsize=12)
axes[0].grid(alpha=0.3)
axes[0].ticklabel_format(axis='y', style='plain')

# Add trend line
from scipy.stats import linregress
x = np.arange(len(monthly_enrol))
slope, intercept, r_value, p_value, std_err = linregress(x, monthly_enrol['total_enrol']
trend_line = slope * x + intercept
axes[0].plot(monthly_enrol.index, trend_line, 'r--', linewidth=2, label=f'Trend (R²={r_v
axes[0].legend()
```

```python
# Stacked area chart - age groups
monthly_enrol[['age_0_5', 'age_5_17', 'age_18_greater']].plot.area(
    ax=axes[1],
    stacked=True,
    color=['#ff9999', '#66b3ff', '#99ff99'],
    alpha=0.7
)
axes[1].set_title('Age Group Composition Over Time', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Enrollments', fontsize=12)
axes[1].set_xlabel('Month', fontsize=12)
axes[1].legend(['Age 0-5', 'Age 5-17', 'Age 18+'], loc='upper left')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('monthly_trend.png', dpi=300, bbox_inches='tight')
plt.show()

# Insights
print("\n" + "="*60)
print("💡 KEY INSIGHTS:")
highest_month = monthly_enrol['total_enrol'].idxmax()
lowest_month = monthly_enrol['total_enrol'].idxmin()
print(f"📈 Highest enrollment month: {highest_month} ({monthly_enrol.loc[highest_month,
print(f"📉 Lowest enrollment month: {lowest_month} ({monthly_enrol.loc[lowest_month, 'to
print(f"📊 Average monthly enrollment: {monthly_enrol['total_enrol'].mean():,.0f}")
print(f"📊 Trend slope: {slope:,.0f} enrollments/month ({'increasing' if slope > 0 else
print("="*60)
```

```
============================================================
📊 UNIVARIATE ANALYSIS: MONTHLY ENROLLMENT TREND
============================================================
```

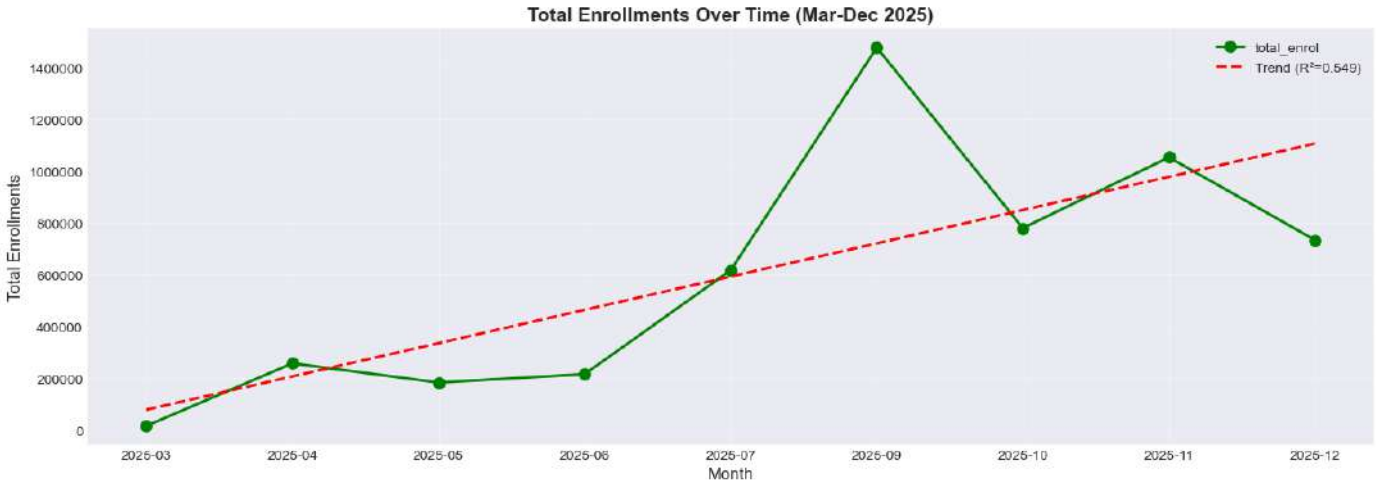
📅 MONTHLY ENROLLMENT DATA:

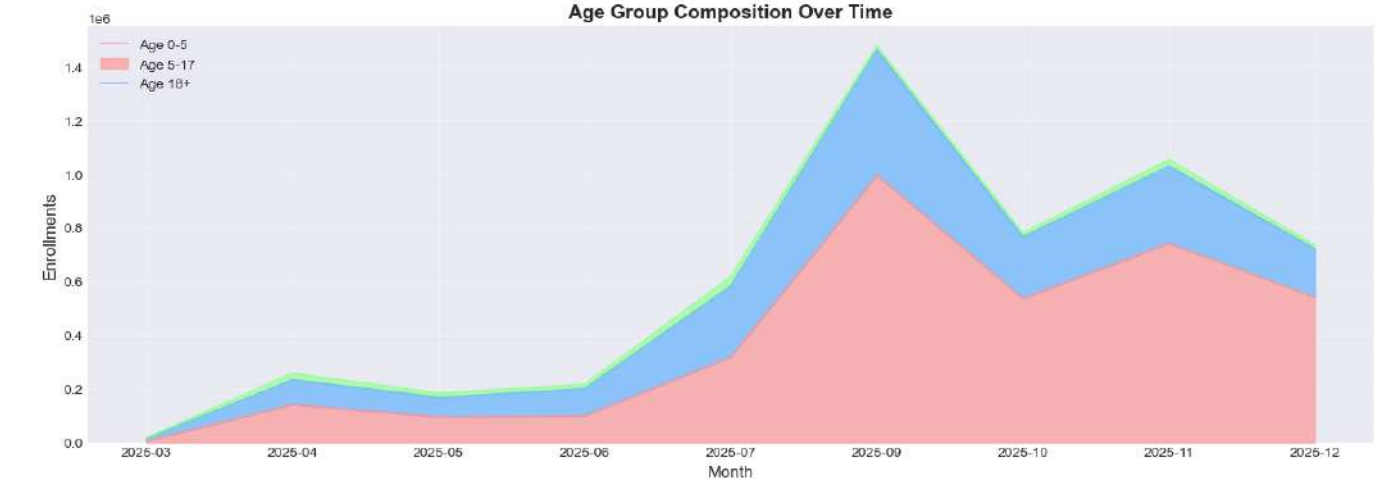
year_month	total_enrol	age_0_5	age_5_17	age_18_greater
2025-03	16582	5367	7407	3808
2025-04	257438	141154	91371	24913
2025-05	183616	95342	71690	16584
2025-06	215734	98943	99911	16880
2025-07	616868	318352	263333	35183
2025-09	1475867	995612	465401	14854
2025-10	779616	536781	227414	15421
2025-11	1052573	742020	286115	24438
2025-12	733252	540818	178266	14168

📈 MONTH-OVER-MONTH GROWTH RATE:

year_month	total_enrol	mom_growth
2025-03	16582	NaN
2025-04	257438	1452.51

year_month	total_enrol	mom_growth
2025-05	183616	-28.68
2025-06	215734	17.49
2025-07	616868	185.94
2025-09	1475867	139.25
2025-10	779616	-47.18
2025-11	1052573	35.01
2025-12	733252	-30.34






```
============================================================
💡 KEY INSIGHTS:
📈 Highest enrollment month: 2025-09 (1,475,867)
📉 Lowest enrollment month: 2025-03 (16,582)
📊 Average monthly enrollment: 592,394
📊 Trend slope: 128,404 enrollments/month (increasing)
============================================================
```

In [ ]:

```
print("="*60)
print("📊 UNIVARIATE ANALYSIS: DISTRICT-LEVEL DISTRIBUTION")
print("="*60)

# Aggregate by district
```

```
district_enrol = enrol.groupby(['state', 'district']).agg({
    'total_enrol': 'sum',
    'age_0_5': 'sum',
    'age_5_17': 'sum',
    'age_18_greater': 'sum'
}).reset_index()

district_enrol = district_enrol.sort_values('total_enrol', ascending=False)

print(f"\n📊 Total districts: {len(district_enrol)}")
print(f"📊 Total enrollments: {district_enrol['total_enrol'].sum():,}")
print(f"📊 Average per district: {district_enrol['total_enrol'].mean():,.0f}")
print(f"📊 Median per district: {district_enrol['total_enrol'].median():,.0f}")

print("\n📈 TOP 20 DISTRICTS BY ENROLLMENT:")
display(district_enrol.head(20))

print("\n📉 BOTTOM 20 DISTRICTS BY ENROLLMENT:")
display(district_enrol.tail(20))

# Find districts with very low enrollment
threshold_low = 100
low_enrollment = district_enrol[district_enrol['total_enrol'] < threshold_low]
print(f"\n🚨 DISTRICTS WITH <{threshold_low} ENROLLMENTS:")
print(f"Count: {len(low_enrollment)}")
display(low_enrollment)

# Visualize distribution
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Top 20 districts
district_enrol.head(20).plot(
    x='district',
    y='total_enrol',
    kind='barh',
    ax=axes[0],
    color='steelblue',
    legend=False
)
axes[0].set_title('Top 20 Districts by Enrollment', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Total Enrollments', fontsize=12)
axes[0].set_ylabel('District', fontsize=12)
axes[0].invert_yaxis()

# Distribution histogram
axes[1].hist(district_enrol['total_enrol'], bins=50, color='coral', edgecolor='black', a
axes[1].set_title('Distribution of District Enrollments', fontsize=14, fontweight='bold'
axes[1].set_xlabel('Total Enrollments', fontsize=12)
axes[1].set_ylabel('Number of Districts', fontsize=12)
axes[1].axvline(district_enrol['total_enrol'].mean(), color='red', linestyle='--', linew
axes[1].axvline(district_enrol['total_enrol'].median(), color='green', linestyle='--', l
axes[1].legend()

plt.tight_layout()
plt.savefig('district_analysis.png', dpi=300, bbox_inches='tight')
plt.show()
```

```
============================================================
📊 UNIVARIATE ANALYSIS: DISTRICT-LEVEL DISTRIBUTION
```

==

📊 Total districts: 979
📊 Total enrollments: 5,331,546
📊 Average per district: 5,446
📊 Median per district: 3,103

📈 TOP 20 DISTRICTS BY ENROLLMENT:

	state	district	total_enrol	age_0_5	age_5_17	age_18_greater
545	Maharashtra	Thane	43142	28692	13492	958
154	Bihar	Sitamarhi	41652	20358	18600	2694
837	Uttar Pradesh	Bahraich	38897	14491	22132	2274
975	West Bengal	South Twenty Four Parganas	37138	27592	9238	308
963	West Bengal	Murshidabad	34968	30593	4290	85
536	Maharashtra	Pune	31148	23622	6404	1122
373	Karnataka	Bengaluru	30657	20227	6655	3775
910	Uttar Pradesh	Sitapur	30475	16003	13732	740
700	Rajasthan	Jaipur	30341	20934	8683	724
158	Bihar	West Champaran	29913	11373	17772	768
966	West Bengal	North Twenty Four Parganas	29817	22407	6061	1349
826	Uttar Pradesh	Agra	29345	15938	12506	901
123	Bihar	East Champaran	28779	9946	18041	792
564	Meghalaya	East Khasi Hills	28656	4239	14578	9839
137	Bihar	Muzaffarpur	28298	13787	13854	657
843	Uttar Pradesh	Bareilly	27357	16880	9892	585
124	Bihar	Gaya	26760	6540	19786	434
976	West Bengal	Uttar Dinajpur	26445	18040	8039	366
827	Uttar Pradesh	Aligarh	25851	13639	11631	581
778	Telangana	Hyderabad	25560	19389	5505	666

📉 BOTTOM 20 DISTRICTS BY ENROLLMENT:

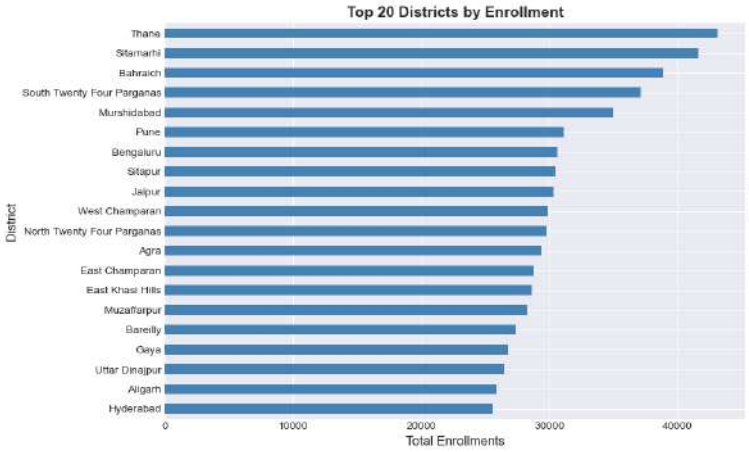
	state	district	total_enrol	age_0_5	age_5_17	age_18_greater
332	Jharkhand	Bokaro *	2	2	0	0
578	Mizoram	Hnahthial	2	2	0	0
414	Karnataka	Udupi *	2	2	0	0
307	Jammu And Kashmir	Bandipur	2	2	0	0
938	West Bengal	Burdwan	2	2	0	0
322	Jammu And Kashmir	Rajauri	2	1	1	0
696	Rajasthan	Didwana-Kuchaman	2	2	0	0
319	Jammu And Kashmir	Leh (Ladakh)	2	2	0	0

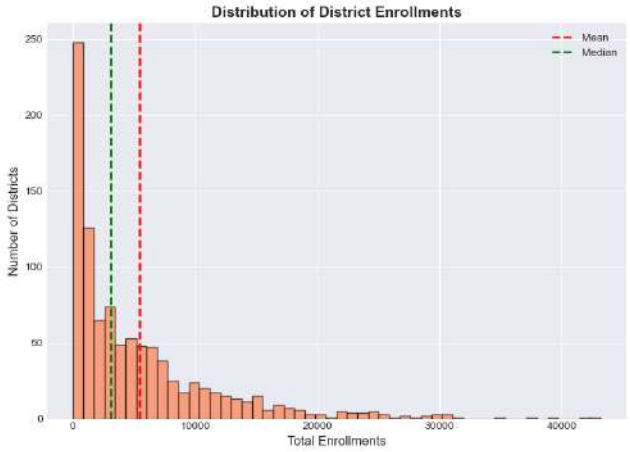
	state	district	total_enrol	age_0_5	age_5_17	age_18_greater
947	West Bengal	East Midnapur	1	1	0	0
2	Andaman And Nicobar Islands	Nicobars	1	1	0	0
714	Rajasthan	Salumbar	1	1	0	0
950	West Bengal	Hooghiy	1	1	0	0
770	Tamil Nadu	Tiruvarur	1	1	0	0
749	Tamil Nadu	Namakkal *	1	0	1	0
680	Rajasthan	Balotra	1	1	0	0
628	Odisha	Kendrapara *	1	1	0	0
519	Maharashtra	Hingoli *	1	1	0	0
274	Haryana	Jhajjar *	1	0	1	0
684	Rajasthan	Beawar	1	1	0	0
836	Uttar Pradesh	Bagpat	1	0	1	0

🚨 DISTRICTS WITH <100 ENROLLMENTS:
Count: 98

	state	district	total_enrol	age_0_5	age_5_17	age_18_greater
477	Madhya Pradesh	Pandhurna	98	66	4	28
69	Arunachal Pradesh	Tirap	89	53	35	1
203	Dadra And Nagar Haveli And Daman And Diu	Dadra And Nagar Haveli	87	69	10	8
539	Maharashtra	Raigarh(Mh)	85	70	15	0
55	Arunachal Pradesh	Kra Daadi	80	18	58	4
...	...	...	...	...	...	...
628	Odisha	Kendrapara *	1	1	0	0
519	Maharashtra	Hingoli *	1	1	0	0
274	Haryana	Jhajjar *	1	0	1	0
684	Rajasthan	Beawar	1	1	0	0
836	Uttar Pradesh	Bagpat	1	0	1	0

98 rows × 6 columns





In [ ]:

```
print("="*60)
print("📊 BIVARIATE ANALYSIS: AGE DISTRIBUTION BY STATE")
print("="*60)

# Calculate age percentages by state
state_age = enrol.groupby('state')[['age_0_5', 'age_5_17', 'age_18_greater']].sum()
state_age_pct = state_age.div(state_age.sum(axis=1), axis=0) * 100

# Sort by youth percentage
state_age_pct = state_age_pct.sort_values('age_5_17')
state_age['youth_pct'] = state_age_pct['age_5_17']

print("\n📉 STATES WITH LOWEST YOUTH (5-17) PERCENTAGE:")
display(state_age.sort_values('youth_pct').head(10)[['youth_pct']])

print("\n📈 STATES WITH HIGHEST YOUTH (5-17) PERCENTAGE:")
display(state_age.sort_values('youth_pct').tail(10)[['youth_pct']])

# 🚨 Check if all states have similar distribution
youth_pct_std = state_age_pct['age_5_17'].std()
print(f"\n📊 Youth % standard deviation across states: {youth_pct_std:.2f}")
if youth_pct_std < 5:
    print("🚨 CRITICAL FINDING: All states show similar age distribution!")
    print("   This suggests a SYSTEMIC data collection issue, not regional variation.")

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(16, 12))

# Stacked bar chart - top 20 states
state_age_pct.head(20).plot(
    kind='barh',
    stacked=True,
    ax=axes[0],
    color=['#ff9999', '#66b3ff', '#99ff99'],
    edgecolor='black',
    linewidth=0.5
)
axes[0].set_title('Age Distribution by State (Top 20 by Youth %)', fontsize=14, fontweig
axes[0].set_xlabel('Percentage', fontsize=12)
axes[0].legend(['Age 0-5', 'Age 5-17', 'Age 18+'], loc='best')
axes[0].invert_yaxis()

# Heatmap
```

```
sns.heatmap(
    state_age_pct.head(15).T,
    annot=True,
    fmt='.1f',
    cmap='RdYlGn',
    ax=axes[1],
    cbar_kws={'label': 'Percentage'}
)
axes[1].set_title('Age Distribution Heatmap (Top 15 States)', fontsize=14, fontweight='b
axes[1].set_ylabel('Age Group', fontsize=12)
axes[1].set_xlabel('State', fontsize=12)

plt.tight_layout()
plt.savefig('state_age_distribution.png', dpi=300, bbox_inches='tight')
plt.show()
```

```
============================================================
📊 BIVARIATE ANALYSIS: AGE DISTRIBUTION BY STATE
============================================================
```

📉 STATES WITH LOWEST YOUTH (5-17) PERCENTAGE:

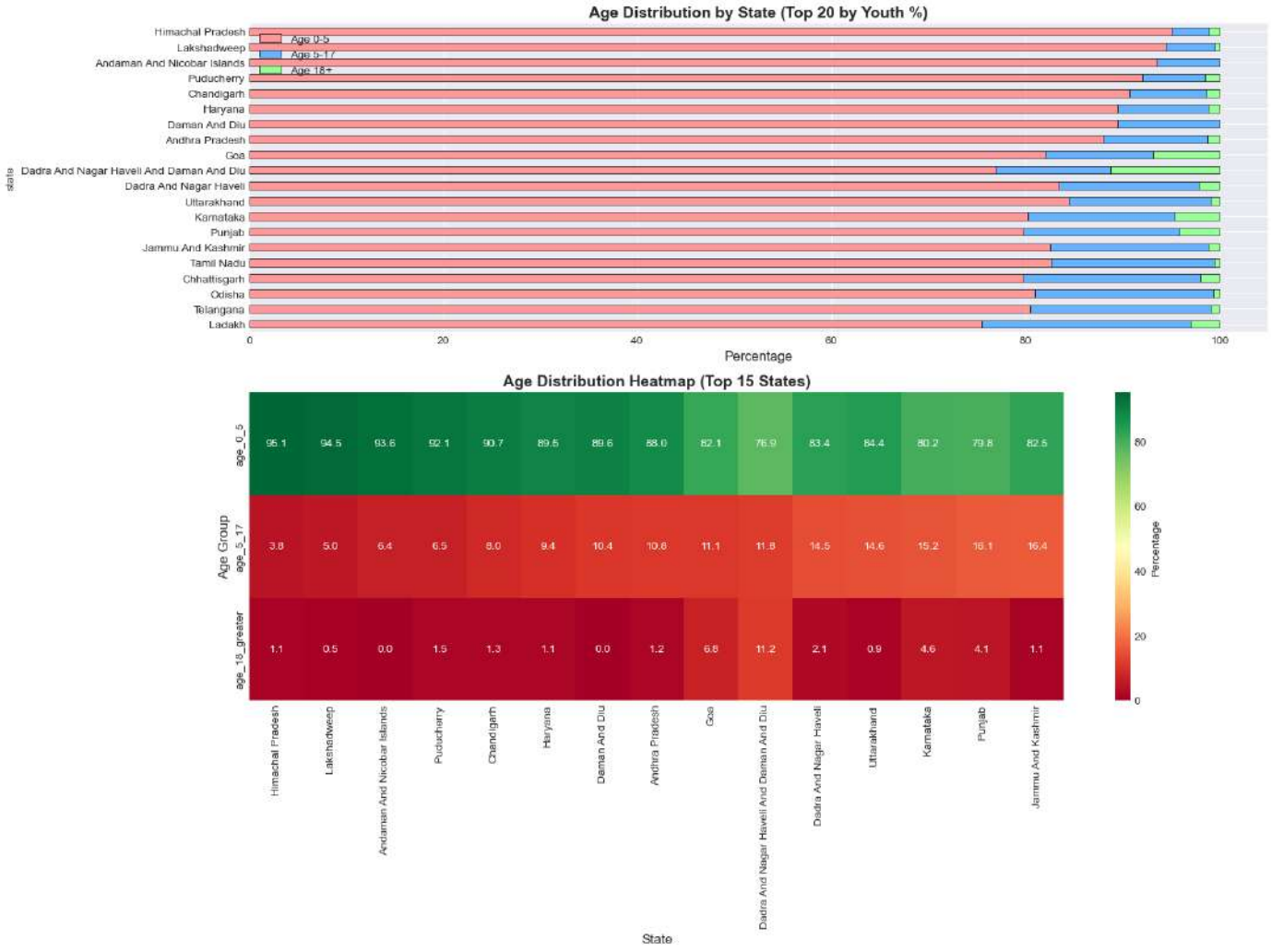
state	youth_pct
Himachal Pradesh	3.844107
Lakshadweep	5.025126
Andaman And Nicobar Islands	6.387226
Puducherry	6.469997
Chandigarh	8.015267
Haryana	9.356891
Daman And Diu	10.447761
Andhra Pradesh	10.793978
Goa	11.096491
Dadra And Nagar Haveli And Daman And Diu	11.834320

📈 STATES WITH HIGHEST YOUTH (5-17) PERCENTAGE:

state	youth_pct
Tripura	32.676235
Jharkhand	36.554436
West Bengal	40.000000
Uttar Pradesh	47.196326
Sikkim	47.356322
Meghalaya	48.598944
Arunachal Pradesh	51.320755
Bihar	55.080648

state	youth_pct
Manipur	59.815138
Nagaland	63.879707

📊 Youth % standard deviation across states: 16.14



In [ ]:

```
print("="*60)
print("📊 BIVARIATE ANALYSIS: AGE DISTRIBUTION OVER TIME")
print("="*60)

# Calculate age percentages by month
month_age = enrol.groupby('year_month')[['age_0_5', 'age_5_17', 'age_18_greater']].sum()
month_age_pct = month_age.div(month_age.sum(axis=1), axis=0) * 100

print("\n📊 AGE PERCENTAGE BY MONTH:")
display(month_age_pct.round(2))

# Check for trend
print("\n📈 TREND ANALYSIS:")
from scipy.stats import linregress
x = np.arange(len(month_age_pct))
slope_child, _, r_child, _, _ = linregress(x, month_age_pct['age_0_5'])
slope_youth, _, r_youth, _, _ = linregress(x, month_age_pct['age_5_17'])
slope_adult, _, r_adult, _, _ = linregress(x, month_age_pct['age_18_greater'])
```

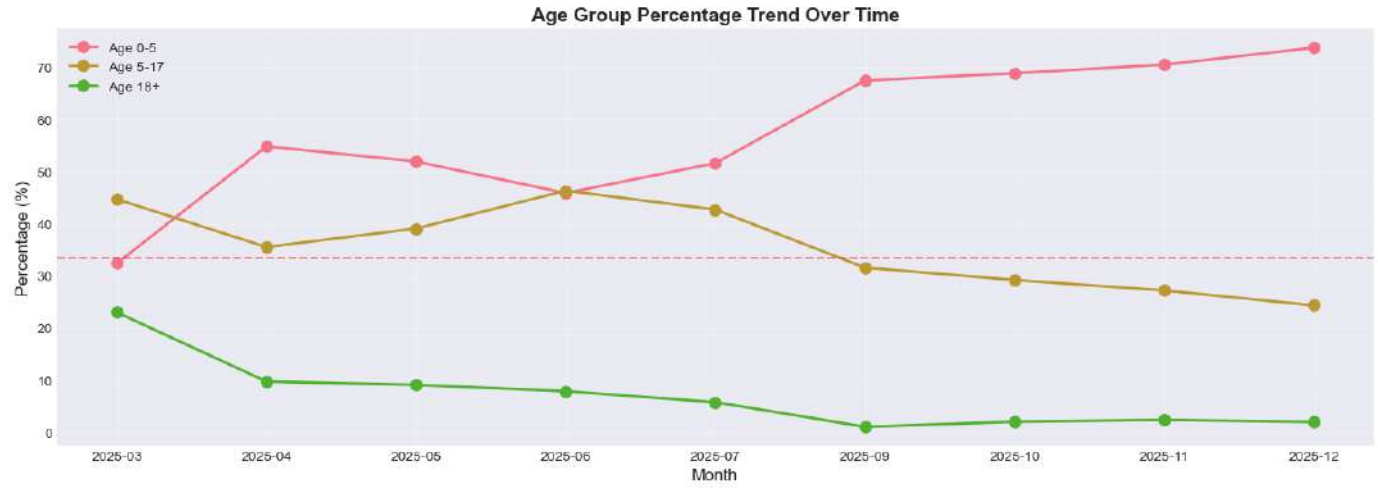
```
print(f"Age 0-5 trend:  {slope_child:+.3f}% per month (R²={r_child**2:.3f})")
print(f"Age 5-17 trend: {slope_youth:+.3f}% per month (R²={r_youth**2:.3f})")
print(f"Age 18+ trend:  {slope_adult:+.3f}% per month (R²={r_adult**2:.3f})")

if abs(slope_child) < 0.5 and abs(slope_youth) < 0.5:
    print("\n🚨 FINDING: Age distribution is STABLE over time")
    print("   → This confirms SYSTEMIC data collection bias, not improving")
else:
    print("\n💡 FINDING: Age distribution is CHANGING over time")
    print("   → System is gradually correcting the bias")

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(14, 10))

# Line chart - percentages
month_age_pct.plot(ax=axes[0], marker='o', linewidth=2, markersize=8)
axes[0].set_title('Age Group Percentage Trend Over Time', fontsize=14, fontweight='bold'
axes[0].set_ylabel('Percentage (%)', fontsize=12)
axes[0].set_xlabel('Month', fontsize=12)
axes[0].legend(['Age 0-5', 'Age 5-17', 'Age 18+'], loc='best')
axes[0].grid(alpha=0.3)
axes[0].axhline(y=33.33, color='red', linestyle='--', alpha=0.5, label='Equal distributi

# Stacked area chart
month_age_pct.plot.area(ax=axes[1], color=['#ff9999', '#66b3ff', '#99ff99'], alpha=0.7)
axes[1].set_title('Age Group Composition Over Time', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Percentage (%)', fontsize=12)
axes[1].set_xlabel('Month', fontsize=12)
axes[1].legend(['Age 0-5', 'Age 5-17', 'Age 18+'], loc='best')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('age_trend.png', dpi=300, bbox_inches='tight')
plt.show()

# Final insight based on the last month's data
last_month = month_age_pct.iloc[-1]
print("\n" + "="*60)
print(f"📆 LATEST STATUS ({month_age_pct.index[-1]}):")
print(f"   - Babies (0-5): {last_month['age_0_5']:.1f}%")
print(f"   - School Kids (5-17): {last_month['age_5_17']:.1f}%")
print(f"   - Adults (18+): {last_month['age_18_greater']:.1f}%")
if last_month['age_0_5'] > 60:
    print("⚠️  CONCLUSION: The system is STILL in 'Birth Registration' mode.")
elif last_month['age_5_17'] > 40:
    print("⚠️  CONCLUSION: The system has shifted to 'School Catch-up' mode.")
else:
    print("✅  CONCLUSION: The system is reaching a balanced state.")
print("="*60)
```

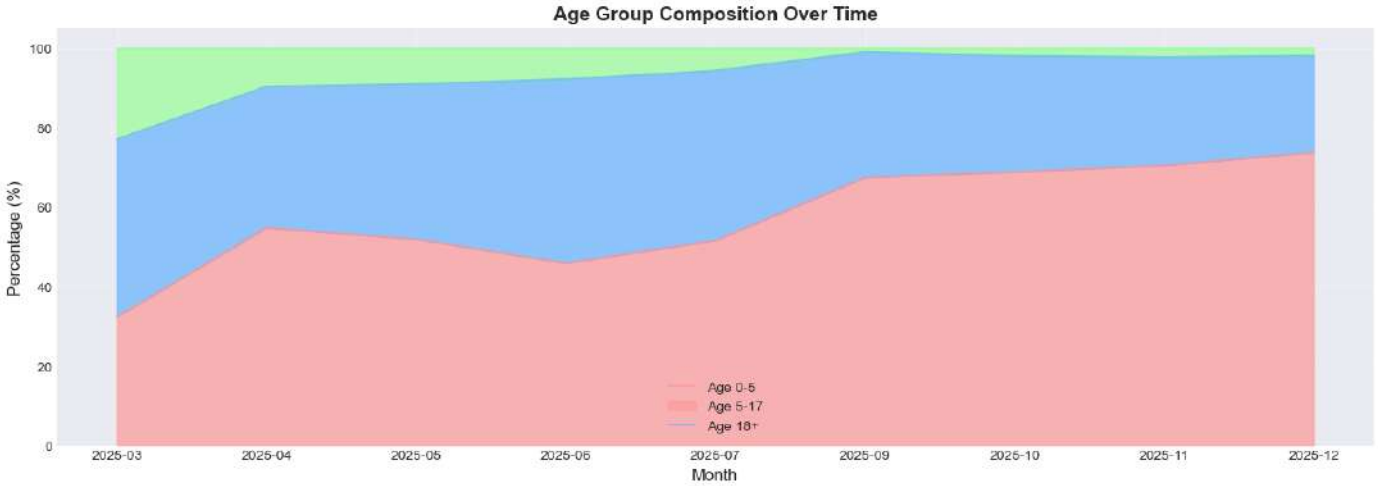
```
============================================================
📊 BIVARIATE ANALYSIS: AGE DISTRIBUTION OVER TIME
============================================================

📊 AGE PERCENTAGE BY MONTH:
```

year_month	age_0_5	age_5_17	age_18_greater
2025-03	32.37	44.67	22.96
2025-04	54.83	35.49	9.68
2025-05	51.92	39.04	9.03
2025-06	45.86	46.31	7.82
2025-07	51.61	42.69	5.70
2025-09	67.46	31.53	1.01
2025-10	68.85	29.17	1.98
2025-11	70.50	27.18	2.32
2025-12	73.76	24.31	1.93

📈 TREND ANALYSIS:
Age 0-5 trend: +4.467% per month (R^2=0.797)
Age 5-17 trend: -2.348% per month (R^2=0.642)
Age 18+ trend: -2.119% per month (R^2=0.714)

💡 FINDING: Age distribution is CHANGING over time
→ System is gradually correcting the bias





==========

📅 LATEST STATUS (2025-12):
- Babies (0-5): 73.8%
- School Kids (5-17): 24.3%

```
   - Adults (18+): 1.9%
⚠  CONCLUSION: The system is STILL in 'Birth Registration' mode.
============================================================
```

In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("="*70)
print("🔍 CELL 13: CROSS-DATASET INTEGRITY & FRAUD DETECTION")
print("="*70)
print("\n💡 HYPOTHESIS: If enrollments are real, they should have corresponding")
print("   demographic and biometric updates. Mismatches = potential fraud or")
print("   system failures.\n")

# ======================================================================
# PART 1: STATE-LEVEL INTEGRITY CHECK
# ======================================================================
print("="*70)
print("PART 1: STATE-LEVEL CROSS-DATASET ANALYSIS")
print("="*70)

# Aggregate all three datasets by state
s_enrol = enrol.groupby('state')['total_enrol'].sum()
s_demo = demo.groupby('state')['total_demo'].sum()
s_bio = bio.groupby('state')['total_bio'].sum()

# Create comprehensive integrity dataframe
integrity = pd.DataFrame({
    'Enrollments': s_enrol,
    'Demographic_Updates': s_demo,
    'Biometric_Updates': s_bio
}).fillna(0)

# Calculate total activity FIRST (before sorting)
integrity['Total_Activity'] = integrity['Enrollments'] + integrity['Demographic_Updates'

# Calculate ratios
integrity['Demo_to_Enrol_Ratio'] = (integrity['Demographic_Updates'] / integrity['Enroll
integrity['Bio_to_Enrol_Ratio'] = (integrity['Biometric_Updates'] / integrity['Enrollmen
integrity['Bio_to_Demo_Ratio'] = (integrity['Biometric_Updates'] / integrity['Demographi

# NOW sort by total activity
integrity = integrity.sort_values('Total_Activity', ascending=False)

print("\n📊 TOP 10 STATES BY TOTAL ACTIVITY:")
display(integrity[['Enrollments', 'Demographic_Updates', 'Biometric_Updates', 'Total_Act

# ======================================================================
# PART 2: FRAUD PATTERN DETECTION
# ======================================================================
print("\n" + "="*70)
print("PART 2: FRAUD PATTERN IDENTIFICATION")
print("="*70)

# Define fraud patterns
fraud_patterns = []
```

```python
# Pattern 1: Ghost Enrollments (High enrollment, low/no updates)
ghost_states = integrity[
    (integrity['Enrollments'] > 1000) &
    (integrity['Demo_to_Enrol_Ratio'] < 0.3)
].copy()
if len(ghost_states) > 0:
    ghost_states['Fraud_Type'] = 'Ghost Enrollments (Low Updates)'
    fraud_patterns.append(ghost_states)

# Pattern 2: Phantom Updates (High updates, low/no enrollment)
phantom_states = integrity[
    (integrity['Demographic_Updates'] > 1000) &
    (integrity['Demo_to_Enrol_Ratio'] > 10)
].copy()
if len(phantom_states) > 0:
    phantom_states['Fraud_Type'] = 'Phantom Updates (No New Enrollments)'
    fraud_patterns.append(phantom_states)

# Pattern 3: Biometric Mismatch (Bio vs Demo don't align)
bio_mismatch = integrity[
    (integrity['Biometric_Updates'] > 1000) &
    (abs(integrity['Bio_to_Demo_Ratio'] - 1.0) > 0.5)
].copy()
if len(bio_mismatch) > 0:
    bio_mismatch['Fraud_Type'] = 'Biometric-Demographic Mismatch'
    fraud_patterns.append(bio_mismatch)

# Pattern 4: Complete Disconnect (All three datasets don't correlate)
disconnect_states = integrity[
    (integrity['Total_Activity'] > 5000) &
    ((integrity['Demo_to_Enrol_Ratio'] < 0.2) | (integrity['Demo_to_Enrol_Ratio'] > 5)) &
    ((integrity['Bio_to_Demo_Ratio'] < 0.5) | (integrity['Bio_to_Demo_Ratio'] > 2))
].copy()
if len(disconnect_states) > 0:
    disconnect_states['Fraud_Type'] = 'Complete System Disconnect'
    fraud_patterns.append(disconnect_states)

# Combine all fraud patterns (only if we have any)
if len(fraud_patterns) > 0:
    all_fraud = pd.concat(fraud_patterns, ignore_index=False)
    all_fraud = all_fraud[~all_fraud.index.duplicated(keep='first')]  # Remove duplicate
else:
    all_fraud = pd.DataFrame()  # Empty dataframe

print(f"\n🚨 FRAUD PATTERNS DETECTED: {len(all_fraud)} states flagged\n")

if len(all_fraud) > 0:
    print("📋 SUSPICIOUS STATES (Detailed Breakdown):")
    # FIX APPLIED HERE: Sort first, then select columns
    display(all_fraud.sort_values('Total_Activity', ascending=False)[[
        'Enrollments', 'Demographic_Updates', 'Biometric_Updates',
        'Demo_to_Enrol_Ratio', 'Bio_to_Enrol_Ratio', 'Fraud_Type'
    ]])
else:
    print("✅ No major fraud patterns detected at state level.")

# ======================================================================
# PART 3: DISTRICT-LEVEL DEEP DIVE (Ghost Districts)
# ======================================================================
```

```
print("\n" + "="*70)
print("PART 3: DISTRICT-LEVEL FRAUD CHECK (Ghost Districts)")
print("="*70)

# Aggregate by district
d_enrol = enrol.groupby(['state', 'district'])['total_enrol'].sum().reset_index()
d_demo = demo.groupby(['state', 'district'])['total_demo'].sum().reset_index()
d_bio = bio.groupby(['state', 'district'])['total_bio'].sum().reset_index()

# Merge all three
district_integrity = d_enrol.merge(d_demo, on=['state', 'district'], how='outer', suffix
district_integrity = district_integrity.merge(d_bio, on=['state', 'district'], how='oute
district_integrity = district_integrity.fillna(0)

# Rename columns
district_integrity.columns = ['state', 'district', 'enrollments', 'demographic_updates',

# Calculate ratios (handle division by zero)
district_integrity['demo_ratio'] = district_integrity.apply(
    lambda x: x['demographic_updates'] / x['enrollments'] if x['enrollments'] > 0 else 0
)
district_integrity['bio_ratio'] = district_integrity.apply(
    lambda x: x['biometric_updates'] / x['enrollments'] if x['enrollments'] > 0 else 0,
)

# Find ghost districts (enrollment > 100 but ratio < 0.1)
ghost_districts = district_integrity[
    (district_integrity['enrollments'] > 100) &
    (district_integrity['demo_ratio'] < 0.1)
].sort_values('enrollments', ascending=False)

print(f"\n🚨 GHOST DISTRICTS DETECTED: {len(ghost_districts)}")
print("   (High enrollments but almost NO demographic updates = suspicious)\n")

if len(ghost_districts) > 0:
    print("📋 TOP 20 GHOST DISTRICTS:")
    display(ghost_districts[['state', 'district', 'enrollments', 'demographic_updates',

# Find dead districts (zero activity across all datasets)
dead_districts = district_integrity[
    (district_integrity['enrollments'] == 0) &
    (district_integrity['demographic_updates'] == 0) &
    (district_integrity['biometric_updates'] == 0)
]

print(f"\n💀 COMPLETELY DEAD DISTRICTS: {len(dead_districts)}")
print("   (Zero activity across all three datasets = service delivery failure)\n")

if len(dead_districts) > 0 and len(dead_districts) <= 20:
    print("📋 DEAD DISTRICTS:")
    display(dead_districts[['state', 'district']])
elif len(dead_districts) > 20:
    print(f"📋 Sample of DEAD DISTRICTS (showing first 20 of {len(dead_districts)}):")
    display(dead_districts[['state', 'district']].head(20))

# ============================================================================
# PART 4: VISUALIZATIONS
# ============================================================================
print("\n" + "="*70)
```

```
print("PART 4: VISUAL EVIDENCE")
print("="*70)

fig, axes = plt.subplots(2, 2, figsize=(18, 14))

# Chart 1: State-level ratios (sorted by Demo/Enrol ratio)
integrity_sorted = integrity.sort_values('Demo_to_Enrol_Ratio')
ax1 = axes[0, 0]
x = np.arange(len(integrity_sorted))
width = 0.35
ax1.bar(x - width/2, integrity_sorted['Demo_to_Enrol_Ratio'], width, label='Demo/Enrol',
ax1.bar(x + width/2, integrity_sorted['Bio_to_Enrol_Ratio'], width, label='Bio/Enrol', c
ax1.axhline(1, color='red', linestyle='--', linewidth=2, label='1:1 Expected')
ax1.axhline(0.2, color='orange', linestyle=':', linewidth=2, label='Fraud Threshold (0.2
ax1.set_xlabel('States (sorted by ratio)', fontweight='bold')
ax1.set_ylabel('Ratio (Updates per Enrollment)', fontweight='bold')
ax1.set_title('Cross-Dataset Ratios by State', fontsize=14, fontweight='bold')
ax1.legend()
ax1.set_xticks([])

# Chart 2: Scatter plot - Enrollments vs Demographic Updates
ax2 = axes[0, 1]
scatter = ax2.scatter(integrity['Enrollments'], integrity['Demographic_Updates'],
                      s=integrity['Biometric_Updates']/100, alpha=0.6, c=integrity['Demo_
                      cmap='RdYlGn', edgecolors='black', linewidth=0.5)
# Add 1:1 line
max_val = max(integrity['Enrollments'].max(), integrity['Demographic_Updates'].max())
ax2.plot([0, max_val], [0, max_val], 'r--', linewidth=2, label='1:1 Line')
ax2.set_xlabel('New Enrollments', fontweight='bold')
ax2.set_ylabel('Demographic Updates', fontweight='bold')
ax2.set_title('Enrollment vs Updates (Size = Biometric Activity)', fontsize=14, fontweig
ax2.legend()
ax2.grid(alpha=0.3)
plt.colorbar(scatter, ax=ax2, label='Demo/Enrol Ratio')

# Chart 3: Distribution of ratios
ax3 = axes[1, 0]
# Filter out extreme outliers for better visualization
ratios_filtered = integrity['Demo_to_Enrol_Ratio'][integrity['Demo_to_Enrol_Ratio'] < 20
ax3.hist(ratios_filtered, bins=30, color='teal', alpha=0.7, edgecolor='black')
ax3.axvline(1, color='red', linestyle='--', linewidth=2, label='Ideal (1:1)')
ax3.axvline(0.2, color='orange', linestyle=':', linewidth=2, label='Fraud Threshold')
ax3.axvline(5, color='purple', linestyle=':', linewidth=2, label='Stagnation Threshold')
ax3.set_xlabel('Demo-to-Enrollment Ratio', fontweight='bold')
ax3.set_ylabel('Number of States', fontweight='bold')
ax3.set_title('Distribution of Cross-Dataset Ratios', fontsize=14, fontweight='bold')
ax3.legend()
ax3.grid(alpha=0.3)

# Chart 4: Fraud type breakdown
ax4 = axes[1, 1]
if len(all_fraud) > 0:
    fraud_counts = all_fraud['Fraud_Type'].value_counts()
    colors_fraud = ['#ff6b6b', '#ee5a6f', '#c44569', '#774c60']
    fraud_counts.plot(kind='barh', ax=ax4, color=colors_fraud[:len(fraud_counts)], edgec
    ax4.set_xlabel('Number of States', fontweight='bold')
    ax4.set_title('Fraud Pattern Breakdown', fontsize=14, fontweight='bold')
    ax4.invert_yaxis()
else:
```

```
    ax4.text(0.5, 0.5, 'No Major Fraud Patterns Detected',
             ha='center', va='center', fontsize=14, fontweight='bold', color='green')
    ax4.set_xlim(0, 1)
    ax4.set_ylim(0, 1)
    ax4.axis('off')

plt.tight_layout()
plt.savefig('cross_dataset_fraud_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

# ============================================================================
# PART 5: SUMMARY REPORT
# ============================================================================
print("\n" + "="*70)
print("📊 FRAUD DETECTION SUMMARY REPORT")
print("="*70)

print(f"\n🔢 QUANTITATIVE FINDINGS:")
print(f"   • Total states analyzed: {len(integrity)}")
print(f"   • States flagged for fraud patterns: {len(all_fraud)}")
print(f"   • Ghost districts (high enrol, low updates): {len(ghost_districts)}")
print(f"   • Dead districts (zero activity): {len(dead_districts)}")

print(f"\n📈 RATIO STATISTICS:")
print(f"   • Mean Demo/Enrol ratio: {integrity['Demo_to_Enrol_Ratio'].mean():.2f}")
print(f"   • Median Demo/Enrol ratio: {integrity['Demo_to_Enrol_Ratio'].median():.2f}")
print(f"   • States with ratio <0.2 (ghost risk): {len(integrity[integrity['Demo_to_Enro
print(f"   • States with ratio >5 (stagnation): {len(integrity[integrity['Demo_to_Enrol_

print(f"\n💡 KEY INSIGHTS:")
if len(all_fraud) > 0:
    print(f"   🚨 {len(all_fraud)} states show suspicious cross-dataset patterns")
    most_common_fraud = all_fraud['Fraud_Type'].value_counts().index[0]
    print(f"   🚨 Primary fraud type: {most_common_fraud}")
else:
    print(f"   ✅ State-level data integrity appears normal")

if len(ghost_districts) > 0:
    print(f"   🚨 {len(ghost_districts)} districts flagged as 'ghost enrollments'")
    top_ghost = ghost_districts.iloc[0]
    print(f"   🚨 Top ghost district: {top_ghost['district']}, {top_ghost['state']} ({to

if len(dead_districts) > 0:
    print(f"   ⚠️  {len(dead_districts)} districts have ZERO activity (service failure)")

print("\n" + "="*70)
print("💼 RECOMMENDATIONS:")
print("="*70)
print("1. Audit flagged districts for enrollment authenticity")
print("2. Investigate states with extreme ratios (<0.2 or >5)")
print("3. Deploy mobile camps to 'dead districts'")
print("4. Implement real-time cross-dataset validation rules")
print("5. Monitor ghost districts monthly for pattern evolution")
print("="*70)
```

```
======================================================================
🔍 CELL 13: CROSS-DATASET INTEGRITY & FRAUD DETECTION
======================================================================
```

💡 HYPOTHESIS: If enrollments are real, they should have corresponding demographic and biometric updates. Mismatches = potential fraud or system failures.

```
======================================================================
PART 1: STATE-LEVEL CROSS-DATASET ANALYSIS
======================================================================
```

📊 TOP 10 STATES BY TOTAL ACTIVITY:

state	Enrollments	Demographic_Updates	Biometric_Updates	Total_Activity
Uttar Pradesh	1002631.0	6460511.0	9367083.0	16830225.0
Maharashtra	363446.0	3824891.0	9020710.0	13209047.0
Bihar	593753.0	3638844.0	4778968.0	9011565.0
Madhya Pradesh	487892.0	2104635.0	5819736.0	8412263.0
Tamil Nadu	215710.0	1686594.0	4572151.0	6474455.0
Rajasthan	340591.0	2058896.0	3927997.0	6327484.0
West Bengal	369234.0	2844268.0	2482168.0	5695670.0
Andhra Pradesh	124273.0	1642142.0	3610776.0	5377191.0
Gujarat	275042.0	1358115.0	3147888.0	4781045.0
Chhattisgarh	99773.0	1421367.0	2559106.0	4080246.0

```
======================================================================
PART 2: FRAUD PATTERN IDENTIFICATION
======================================================================
```

🚨 FRAUD PATTERNS DETECTED: 30 states flagged

📋 SUSPICIOUS STATES (Detailed Breakdown):

state	Enrollments	Demographic_Updates	Biometric_Updates	Demo_to_Enrol_Ratio	Bio_to_En
Maharashtra	363446.0	3824891.0	9020710.0	10.523960	2
Madhya Pradesh	487892.0	2104635.0	5819736.0	4.313731	1
Tamil Nadu	215710.0	1686594.0	4572151.0	7.818803	2
Rajasthan	340591.0	2058896.0	3927997.0	6.045069	1
Andhra Pradesh	124273.0	1642142.0	3610776.0	13.213989	2

state	Enrollments	Demographic_Updates	Biometric_Updates	Demo_to_Enrol_Ratio	Bio_to_En
Gujarat	275042.0	1358115.0	3147888.0	4.937846	1
Chhattisgarh	99773.0	1421367.0	2559106.0	14.246008	2
Karnataka	219618.0	1254788.0	2602087.0	5.713503	1
Odisha	120454.0	852642.0	2423740.0	7.078569	2
Jharkhand	153612.0	1065328.0	1995940.0	6.935187	1
Haryana	95085.0	825843.0	1600264.0	8.685313	1
Punjab	75773.0	642172.0	1717845.0	8.474945	2
Delhi	92838.0	969666.0	1281944.0	10.444710	1
Kerala	73950.0	573781.0	1585769.0	7.759040	2
Uttarakhand	36956.0	337421.0	740796.0	9.130344	2
Jammu And Kashmir	47638.0	294758.0	771587.0	6.187455	1
Manipur	13199.0	233382.0	275900.0	17.681794	2
Himachal Pradesh	16909.0	115813.0	385264.0	6.849193	2
Tripura	11008.0	104583.0	285324.0	9.500636	2
Mizoram	5774.0	31556.0	119160.0	5.465189	2
Nagaland	15429.0	27555.0	108271.0	1.785923	

state	Enrollments	Demographic_Updates	Biometric_Updates	Demo_to_Enrol_Ratio	Bio_to_En
Chandigarh	2620.0	57395.0	73552.0	21.906489	2
Arunachal Pradesh	4240.0	28391.0	70059.0	6.695991	1
Goa	2280.0	27691.0	66554.0	12.145175	2
Puducherry	2983.0	24776.0	68523.0	8.305732	2
Dadra And Nagar Haveli	1479.0	4700.0	27523.0	3.177823	1
Andaman And Nicobar Islands	501.0	5228.0	19994.0	10.435130	3
Daman And Diu	134.0	1735.0	8862.0	12.947761	6
Lakshadweep	199.0	913.0	4745.0	4.587940	2
Dadra And Nagar Haveli And Daman And Diu	169.0	2768.0	2507.0	16.378698	1

```
======================================================================
PART 3: DISTRICT-LEVEL FRAUD CHECK (Ghost Districts)
======================================================================
```

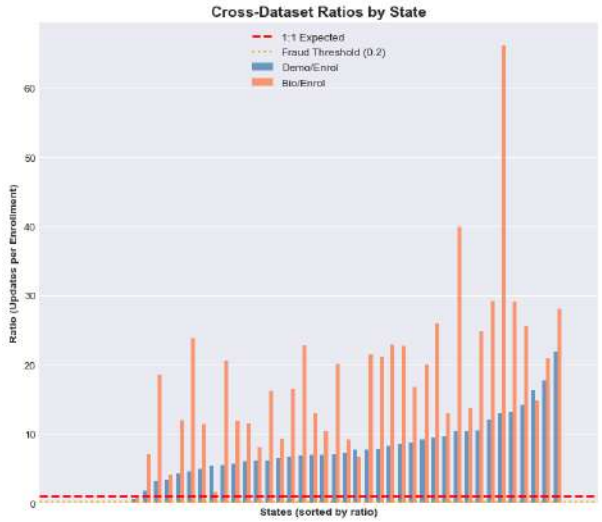
🚨 GHOST DISTRICTS DETECTED: 31
(High enrollments but almost NO demographic updates = suspicious)

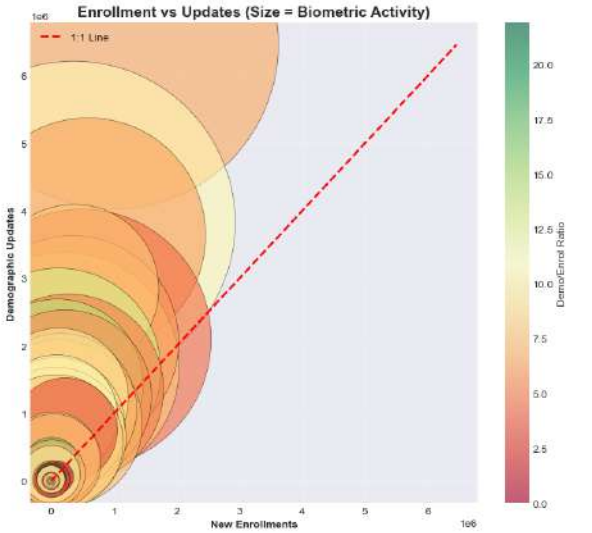
📋 TOP 20 GHOST DISTRICTS:

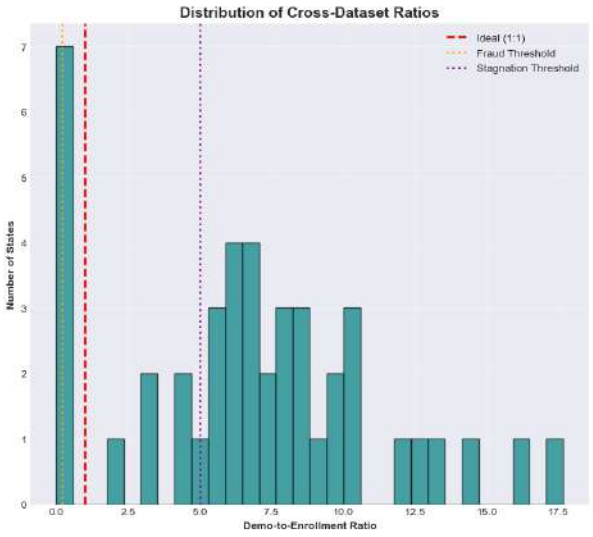
	state	district	enrollments	demographic_updates	demo_ratio
386	Karnataka	Bengaluru Urban	23074.0	0.0	0.000000
141	Bihar	Pashchim Champaran	16358.0	832.0	0.050862
144	Bihar	Purbi Champaran	14871.0	0.0	0.000000
236	Gujarat	Banas Kantha	13491.0	377.0	0.027945
981	West Bengal	Dinajpur Uttar	11671.0	0.0	0.000000

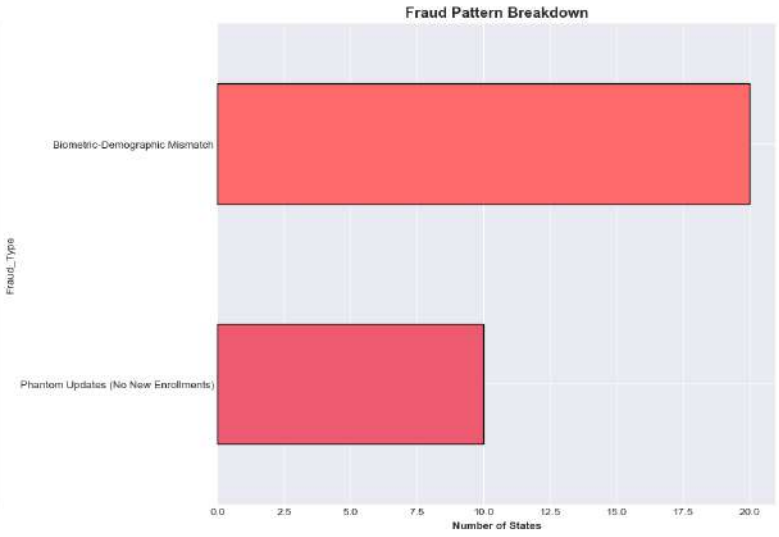
	state	district	enrollments	demographic_updates	demo_ratio
966	West Bengal	24 Paraganas North	6147.0	0.0	0.000000
976	West Bengal	Coochbehar	4410.0	0.0	0.000000
942	Uttar Pradesh	Siddharth Nagar	4341.0	0.0	0.000000
452	Madhya Pradesh	Ashoknagar	3011.0	0.0	0.000000
384	Karnataka	Bengaluru Rural	2790.0	61.0	0.021864
278	Haryana	Gurugram	2636.0	0.0	0.000000
940	Uttar Pradesh	Shravasti	2535.0	0.0	0.000000
38	Andhra Pradesh	Spsr Nellore	2183.0	0.0	0.000000
911	Uttar Pradesh	Kushi Nagar	1570.0	0.0	0.000000
288	Haryana	Nuh	1506.0	0.0	0.000000
348	Jharkhand	East Singhbum	1447.0	0.0	0.000000
980	West Bengal	Dinajpur Dakshin	997.0	0.0	0.000000
177	Chhattisgarh	Gaurella Pendra Marwahi	983.0	0.0	0.000000
243	Gujarat	Dang	872.0	0.0	0.000000
692	Punjab	S.A.S Nagar	705.0	0.0	0.000000

```
💀 COMPLETELY DEAD DISTRICTS: 0
   (Zero activity across all three datasets = service delivery failure)


======================================================================
PART 4: VISUAL EVIDENCE
======================================================================
```









==========

📊 FRAUD DETECTION SUMMARY REPORT

==========

🔢 QUANTITATIVE FINDINGS:

- Total states analyzed: 46
- States flagged for fraud patterns: 30
- Ghost districts (high enrol, low updates): 31
- Dead districts (zero activity): 0

📈 RATIO STATISTICS:

- Mean Demo/Enrol ratio: 7.17
- Median Demo/Enrol ratio: 6.94
- States with ratio <0.2 (ghost risk): 6
- States with ratio >5 (stagnation): 32

💡 KEY INSIGHTS:

- 🚨 30 states show suspicious cross-dataset patterns
- 🚨 Primary fraud type: Biometric-Demographic Mismatch
- 🚨 31 districts flagged as 'ghost enrollments'
- 🚨 Top ghost district: Bengaluru Urban, Karnataka (23,074 enrollments)

==========

💼 RECOMMENDATIONS:

==========

1. Audit flagged districts for enrollment authenticity
2. Investigate states with extreme ratios (<0.2 or >5)

```
3. Deploy mobile camps to 'dead districts'
4. Implement real-time cross-dataset validation rules
5. Monitor ghost districts monthly for pattern evolution
================================================================================
```

In [ ]:

```python
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore



feature_cols = ['age_0_5', 'age_5_17', 'age_18_greater', 'total_enrol',
                'youth_pct', 'child_pct', 'adult_pct']

X = enrol[feature_cols].copy()
X = X.fillna(0)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

contamination_levels = [0.01, 0.005, 0.001]  # 1%, 0.5%, 0.1%
results = {}

for contam in contamination_levels:
    iso = IsolationForest(
        contamination=contam,
        random_state=42,
        n_estimators=100,
        max_samples='auto',
        max_features=1.0
    )
    predictions = iso.fit_predict(X_scaled)
    results[contam] = predictions

    n_anomalies = (predictions == -1).sum()
    print(f" Contamination {contam*100:.1f}%: Detected {n_anomalies:,} anomalies ({n_ano

enrol['anomaly_score'] = results[0.005]


anomalies = enrol[enrol['anomaly_score'] == -1].copy()
normal = enrol[enrol['anomaly_score'] == 1].copy()

print(f"\n DETECTION RESULTS:")
print(f"   • Total records analyzed: {len(enrol):,}")
print(f"   • Anomalies detected: {len(anomalies):,} ({len(anomalies)/len(enrol)*100:.3f}
print(f"   • Normal records: {len(normal):,} ({len(normal)/len(enrol)*100:.3f}%)")

comparison = pd.DataFrame({
    'Normal': normal[feature_cols].mean(),
    'Anomalies': anomalies[feature_cols].mean(),
    'Difference (%)': ((anomalies[feature_cols].mean() - normal[feature_cols].mean()) /
})

print("\n ANOMALY VS NORMAL COMPARISON:")
display(comparison)
```

```
print("\n" + "="*70)
print("PART 3: ANOMALY TYPE CLASSIFICATION")
print("="*70)

def classify_anomaly(row):
    """Classify why a record is anomalous"""
    reasons = []

    if row['total_enrol'] > enrol['total_enrol'].quantile(0.999):
        reasons.append('Extreme Volume')

    if row['age_18_greater'] > row['age_0_5']:
        reasons.append('Adult Spike')


    if row['age_5_17'] == 0 and row['total_enrol'] > 100:
        reasons.append('Missing Youth Data')

    if row['child_pct'] > 95:
        reasons.append('Extreme Child Bias')

    if row['total_enrol'] == 0:
        reasons.append('Zero Enrollment')

    if row['youth_pct'] > 50:  # Youth should be 32% normally
        reasons.append('Youth Overrepresentation')

    return ', '.join(reasons) if reasons else 'Other'

anomalies['anomaly_type'] = anomalies.apply(classify_anomaly, axis=1)


anomaly_type_counts = anomalies['anomaly_type'].value_counts()

print(f"\n ANOMALY TYPE BREAKDOWN:")
display(anomaly_type_counts)


print("\n" + "="*70)
print("PART 4: THE MOST SUSPICIOUS RECORDS")
print("="*70)


top_anomalies = anomalies.sort_values('total_enrol', ascending=False).head(10)

print("\n TOP 10 ANOMALOUS RECORDS (Highest Impact):")
display(top_anomalies[['date', 'state', 'district', 'pincode', 'total_enrol',
                       'child_pct', 'youth_pct', 'adult_pct', 'anomaly_type']])


anomalous_pincodes = anomalies.groupby('pincode').size().sort_values(ascending=False).he
print(f"\nTOP 20 PINCODES WITH MOST ANOMALOUS RECORDS:")
print(anomalous_pincodes)


print("\n" + "="*70)
print("PART 5: GEOGRAPHIC ANOMALY PATTERNS")
print("="*70)
```

```
state_anomaly_rate = enrol.groupby('state').apply(
    lambda x: (x['anomaly_score'] == -1).sum() / len(x) * 100
).sort_values(ascending=False)

print(f"\n STATES WITH HIGHEST ANOMALY RATES:")
display(state_anomaly_rate.head(15).to_frame(name='Anomaly Rate (%)'))


district_anomaly_count = anomalies.groupby(['state', 'district']).size().sort_values(asc
print(f"\n DISTRICTS WITH MOST ANOMALOUS RECORDS:")
display(district_anomaly_count.to_frame(name='Anomaly Count'))


print("\n" + "="*70)
print("PART 6: VISUAL ANOMALY ANALYSIS")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(20, 12))


ax1 = axes[0, 0]
ax1.scatter(normal['age_0_5'], normal['age_5_17'], alpha=0.3, s=10, c='blue', label='Nor
ax1.scatter(anomalies['age_0_5'], anomalies['age_5_17'], alpha=0.7, s=30, c='red',
           edgecolors='black', linewidth=0.5, label='Anomaly')
ax1.set_xlabel('Age 0-5 Enrollments', fontweight='bold')
ax1.set_ylabel('Age 5-17 Enrollments', fontweight='bold')
ax1.set_title('Anomaly Detection: Age Distribution Space', fontsize=12, fontweight='bold
ax1.legend()
ax1.grid(alpha=0.3)


ax2 = axes[0, 1]
ax2.scatter(normal['total_enrol'], normal['youth_pct'], alpha=0.3, s=10, c='blue', label
ax2.scatter(anomalies['total_enrol'], anomalies['youth_pct'], alpha=0.7, s=30, c='red',
           edgecolors='black', linewidth=0.5, label='Anomaly')
ax2.set_xlabel('Total Enrollments', fontweight='bold')
ax2.set_ylabel('Youth % (5-17)', fontweight='bold')
ax2.set_title('Anomaly Detection: Volume vs Youth %', fontsize=12, fontweight='bold')
ax2.legend()
ax2.grid(alpha=0.3)
ax2.set_xscale('log')


ax3 = axes[0, 2]
if len(anomaly_type_counts) > 0:
    colors_anom = ['#e74c3c', '#e67e22', '#f39c12', '#16a085', '#2980b9']
    anomaly_type_counts.head(5).plot(kind='barh', ax=ax3, color=colors_anom[:len(anomaly
                                     edgecolor='black')
    ax3.set_xlabel('Count', fontweight='bold')
    ax3.set_title('Top 5 Anomaly Types', fontsize=12, fontweight='bold')
    ax3.invert_yaxis()


ax4 = axes[1, 0]
state_anomaly_rate.head(20).plot(kind='barh', ax=ax4, color='coral', edgecolor='black')
ax4.set_xlabel('Anomaly Rate (%)', fontweight='bold')
ax4.set_title('Top 20 States by Anomaly Rate', fontsize=12, fontweight='bold')
```

```
ax4.invert_yaxis()


ax5 = axes[1, 1]
anomaly_time = enrol.groupby('year_month')['anomaly_score'].apply(lambda x: (x == -1).su
normal_time = enrol.groupby('year_month')['anomaly_score'].apply(lambda x: (x == 1).sum(
ax5.plot(anomaly_time.index, anomaly_time.values, marker='o', color='red', linewidth=2,
ax5.plot(normal_time.index, normal_time.values, marker='o', color='blue', linewidth=2, l
ax5.set_xlabel('Month', fontweight='bold')
ax5.set_ylabel('Count', fontweight='bold')
ax5.set_title('Anomaly Trend Over Time', fontsize=12, fontweight='bold')
ax5.legend()
ax5.grid(alpha=0.3)
plt.setp(ax5.xaxis.get_majorticklabels(), rotation=45)


ax6 = axes[1, 2]
data_box = [normal['total_enrol'], anomalies['total_enrol']]
bp = ax6.boxplot(data_box, labels=['Normal', 'Anomalies'], patch_artist=True,
                 boxprops=dict(facecolor='lightblue', edgecolor='black'),
                 medianprops=dict(color='red', linewidth=2),
                 whiskerprops=dict(color='black'),
                 capprops=dict(color='black'))
bp['boxes'][1].set_facecolor('salmon')
ax6.set_ylabel('Total Enrollments', fontweight='bold')
ax6.set_title('Enrollment Volume Distribution', fontsize=12, fontweight='bold')
ax6.set_yscale('log')
ax6.grid(alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig('ml_anomaly_detection.png', dpi=300, bbox_inches='tight')
plt.show()



print(" ML ANOMALY DETECTION SUMMARY")


print(f"\n QUANTITATIVE FINDINGS:")
print(f"   • Records analyzed: {len(enrol):,}")
print(f"   • Anomalies detected: {len(anomalies):,} ({len(anomalies)/len(enrol)*100:.3f}
print(f"   • Features used: {len(feature_cols)}")
print(f"   • Most common anomaly type: {anomaly_type_counts.index[0] if len(anomaly_type

print(f"\n TOP ANOMALY CHARACTERISTICS:")
if len(anomalies) > 0:
    print(f"   • Avg enrollment (normal): {normal['total_enrol'].mean():,.0f}")
    print(f"   • Avg enrollment (anomalies): {anomalies['total_enrol'].mean():,.0f}")
    print(f"   • Difference: {((anomalies['total_enrol'].mean() / normal['total_enrol'].
    print(f"   • Avg child % (normal): {normal['child_pct'].mean():.1f}%")
    print(f"   • Avg child % (anomalies): {anomalies['child_pct'].mean():.1f}%")

print(f"\n GEOGRAPHIC CONCENTRATION:")
print(f"   • State with highest anomaly rate: {state_anomaly_rate.index[0]} ({state_anom
print(f"   • States with >1% anomaly rate: {len(state_anomaly_rate[state_anomaly_rate >

print(f"\n KEY INSIGHTS:")
if 'Extreme Volume' in anomaly_type_counts.index:
    print(f" {anomaly_type_counts['Extreme Volume']} records show extreme enrollment vol
```

```python
if 'Missing Youth Data' in anomaly_type_counts.index:
    print(f" {anomaly_type_counts['Missing Youth Data']} records missing youth (5-17) da
if 'Adult Spike' in anomaly_type_counts.index:
    print(f"{anomaly_type_counts['Adult Spike']} records show unusual adult enrollment p

print(" RECOMMENDATIONS:")
print("1. Investigate top 10 anomalous records for data entry errors")
print("2. Audit pincodes with recurring anomalous patterns")
print("3. Review states with >1% anomaly rate for systemic issues")
print("4. Implement automated anomaly flagging in enrollment system")
print("5. Cross-reference anomalies with fraud patterns from Cell 13")

anomalies_export = anomalies[['date', 'state', 'district', 'pincode', 'total_enrol',
                              'age_0_5', 'age_5_17', 'age_18_greater', 'anomaly_type']]
anomalies_export.to_csv('detected_anomalies.csv', index=False)
print("\n'detected_anomalies.csv'")
```

```
Contamination 1.0%: Detected 9,329 anomalies (0.949%)
Contamination 0.5%: Detected 4,915 anomalies (0.500%)
Contamination 0.1%: Detected 975 anomalies (0.099%)

DETECTION RESULTS:
  • Total records analyzed: 983,051
  • Anomalies detected: 4,915 (0.500%)
  • Normal records: 978,136 (99.500%)

ANOMALY VS NORMAL COMPARISON:
```

	Normal	Anomalies	Difference (%)
age_0_5	2.855211	138.678332	4757.03
age_5_17	1.161478	112.884028	9619.00
age_18_greater	0.054048	23.068769	42582.24
total_enrol	4.070737	274.631129	6646.47
youth_pct	24.357996	32.829270	34.78
child_pct	74.182429	39.913829	-46.20
adult_pct	1.459567	27.256779	1767.46

```
======================================================================
PART 3: ANOMALY TYPE CLASSIFICATION
======================================================================

 ANOMALY TYPE BREAKDOWN:
anomaly_type
Other                                                  1967
Adult Spike                                            1184
Youth Overrepresentation                                651
Extreme Volume                                          598
Extreme Volume, Youth Overrepresentation                329
Adult Spike, Youth Overrepresentation                   128
Extreme Volume, Adult Spike, Youth Overrepresentation    29
Extreme Volume, Adult Spike                              28
Adult Spike, Missing Youth Data                           1
Name: count, dtype: int64
======================================================================
PART 4: THE MOST SUSPICIOUS RECORDS
```

```
======================================================================
```

TOP 10 ANOMALOUS RECORDS (Highest Impact):

	date	state	district	pincode	total_enrol	child_pct	youth_pct	adult_pct	anomaly_
509005	2025-07-01	Uttar Pradesh	Moradabad	244001	3965	67.79	31.63	0.58	Ext Vo
509242	2025-07-01	Maharashtra	Aurangabad	431001	3835	58.98	40.39	0.63	Ext Vo
506922	2025-04-01	Meghalaya	West Khasi Hills	793119	3027	18.93	59.86	21.21	Ext Volume, Spike, Overrepr
509070	2025-07-01	Uttar Pradesh	Hardoi	241001	3006	51.63	47.70	0.67	Ext Vo
508475	2025-07-01	Uttar Pradesh	Firozabad	283203	2990	64.88	34.78	0.33	Ext Vo
509089	2025-07-01	Uttar Pradesh	Saharanpur	247001	2920	70.34	29.18	0.48	Ext Vo
508617	2025-07-01	Delhi	North East Delhi	110094	2874	63.54	35.80	0.66	Ext Vo
508279	2025-07-01	Madhya Pradesh	Barwani	451666	2855	49.04	48.20	2.77	Ext Vo
509098	2025-07-01	Uttar Pradesh	Aligarh	202001	2845	50.47	47.49	2.04	Ext Vo
508847	2025-07-01	West Bengal	Dinajpur Uttar	733210	2727	63.59	35.90	0.51	Ext Vo

```
TOP 20 PINCODES WITH MOST ANOMALOUS RECORDS:
pincode
271855    33
793119    32
793121    29
793150    22
271865    22
431001    21
144601    20
244001    19
451666    19
793151    18
494444    18
202001    17
793015    17
793110    17
250002    16
794103    16
793105    16
793200    15
793009    15
733207    14
dtype: int64

======================================================================
PART 5: GEOGRAPHIC ANOMALY PATTERNS
```

==========

STATES WITH HIGHEST ANOMALY RATES:

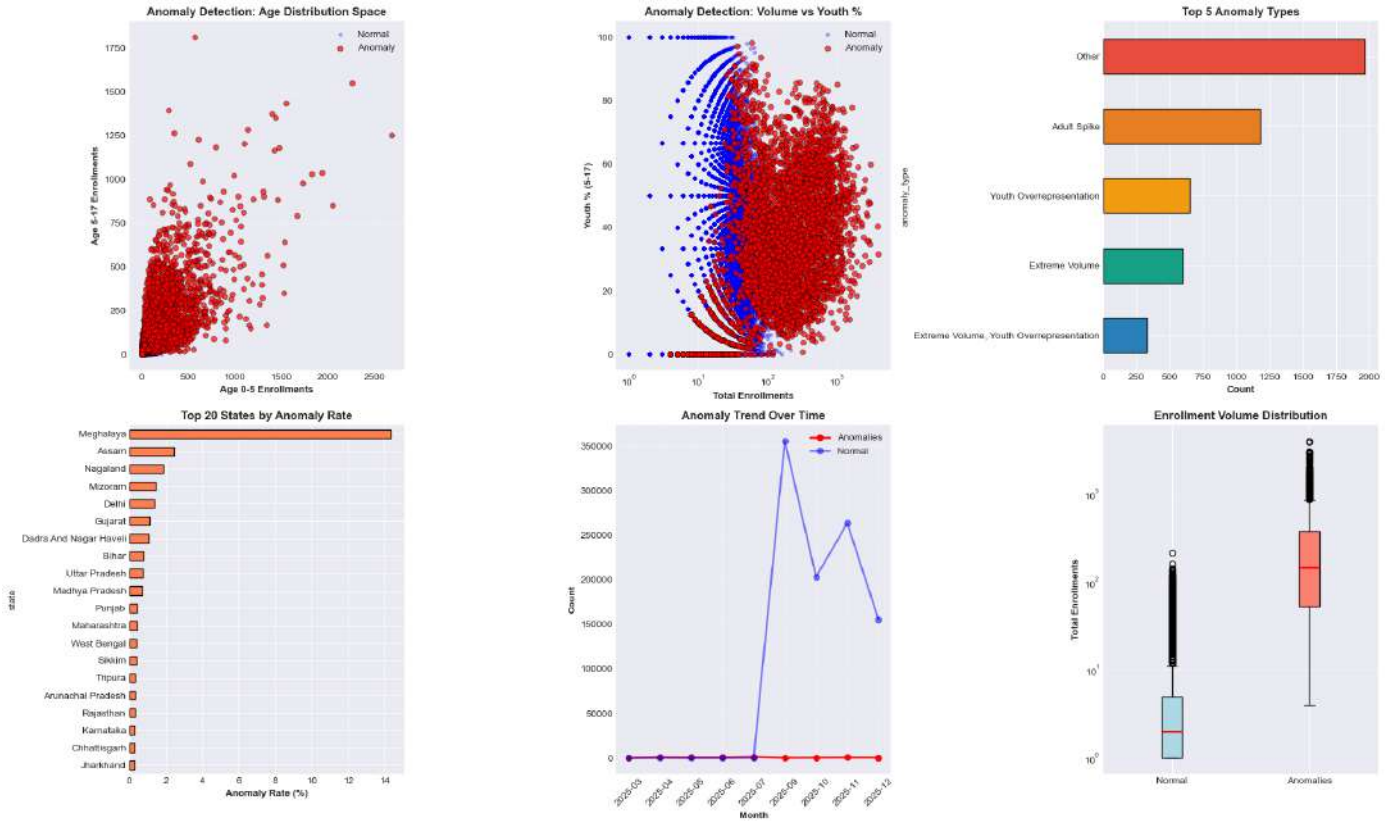
state	Anomaly Rate (%)
Meghalaya	14.328439
Assam	2.446334
Nagaland	1.896463
Mizoram	1.455301
Delhi	1.385542
Gujarat	1.100723
Dadra And Nagar Haveli	1.075269
Bihar	0.780636
Uttar Pradesh	0.722706
Madhya Pradesh	0.679956
Punjab	0.418306
Maharashtra	0.415723
West Bengal	0.404745
Sikkim	0.402820
Tripura	0.330215

DISTRICTS WITH MOST ANOMALOUS RECORDS:

state	district	Anomaly Count
Meghalaya	**East Khasi Hills**	200
Uttar Pradesh	**Bahraich**	103
Karnataka	**Bengaluru Urban**	96
	Bengaluru	68
Assam	**Kamrup Metro**	68
	Cachar	59
Meghalaya	**West Garo Hills**	58
Bihar	**Sitamarhi**	57
Assam	**Golaghat**	57
Meghalaya	**Ri Bhoi**	54
Assam	**Tinsukia**	50
	Dibrugarh	48
Meghalaya	**West Jaintia Hills**	47
Assam	**Kamrup**	43

		Anomaly Count
state	district	
Maharashtra	Thane	42
Uttar Pradesh	Agra	40
West Bengal	Uttar Dinajpur	40
Maharashtra	Pune	38
Meghalaya	West Khasi Hills	37
Gujarat	Banaskantha	36

```
======================================================================
PART 6: VISUAL ANOMALY ANALYSIS
======================================================================
```




```
ML ANOMALY DETECTION SUMMARY

QUANTITATIVE FINDINGS:
  • Records analyzed: 983,051
  • Anomalies detected: 4,915 (0.500%)
  • Features used: 7
  • Most common anomaly type: Other

TOP ANOMALY CHARACTERISTICS:
  • Avg enrollment (normal): 4
  • Avg enrollment (anomalies): 275
  • Difference: +6646.5%
  • Avg child % (normal): 74.2%
  • Avg child % (anomalies): 39.9%

GEOGRAPHIC CONCENTRATION:
  • State with highest anomaly rate: Meghalaya (14.33%)
  • States with >1% anomaly rate: 7
```

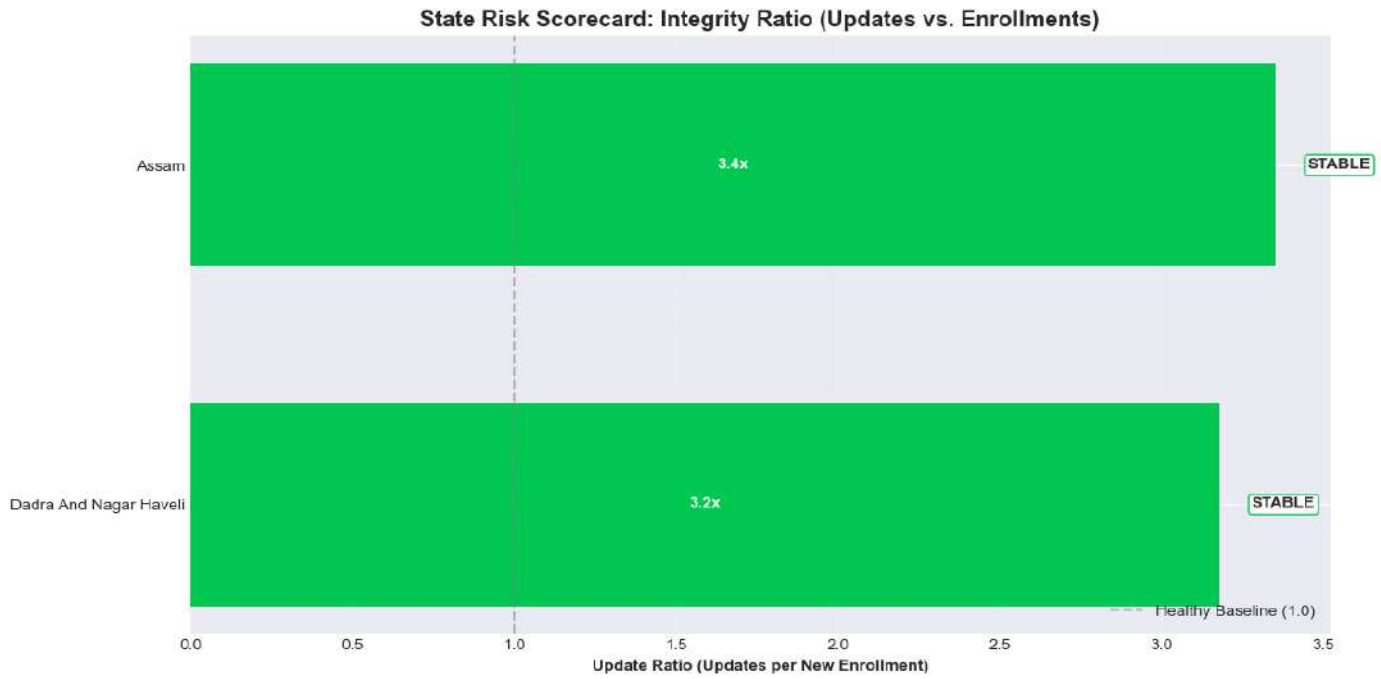
```
 KEY INSIGHTS:
 598 records show extreme enrollment volumes
1184 records show unusual adult enrollment patterns
 RECOMMENDATIONS:
1. Investigate top 10 anomalous records for data entry errors
2. Audit pincodes with recurring anomalous patterns
3. Review states with >1% anomaly rate for systemic issues
4. Implement automated anomaly flagging in enrollment system
5. Cross-reference anomalies with fraud patterns from Cell 13

'detected_anomalies.csv'
```

In [ ]:

```
print(" CELL 15: STATE INTEGRITY SCORECARD (The 'Impact' Graph)")

# 1. Prepare Data from previous Cell 13 calculation
# (Re-calculating just to be safe)
s_enrol = enrol.groupby('state')['total_enrol'].sum()
s_demo = demo.groupby('state')['total_demo'].sum()
scorecard = pd.DataFrame({'New_Enrollments': s_enrol, 'Demographic_Updates': s_demo})
scorecard.fillna(0, inplace=True)
scorecard['Ratio'] = scorecard['Demographic_Updates'] / scorecard['New_Enrollments']

# 2. Define Categories
def categorize(ratio):
    if ratio < 0.2:
        return 'CRITICAL', '#ff4d4d'  # Red (Ghost Enrollments)
    elif ratio > 5.0:
        return 'STAGNANT', '#ffc107'  # Yellow (Maintenance Mode)
    else:
        return 'STABLE', '#00c853'    # Green (Healthy)

scorecard[['Category', 'Color']] = scorecard['Ratio'].apply(lambda x: pd.Series(categori

# 3. Select Top examples for the graph (mix of categories for impact)
# Let's pick top 3 Critical, top 3 Stagnant, and 2 Stable states
critical = scorecard[scorecard['Category']=='CRITICAL'].sort_values('Ratio').head(3)
stagnant = scorecard[scorecard['Category']=='STAGNANT'].sort_values('Ratio', ascending=F
stable = scorecard[scorecard['Category']=='STABLE'].head(2)

plot_data = pd.concat([critical, stagnant, stable])

# 4. Create the "Impact" Graph
plt.figure(figsize=(12, 6))

# Create horizontal bars
bars = plt.barh(plot_data.index, plot_data['Ratio'], color=plot_data['Color'], height=0.

# Add the "Stamps" (Text labels inside/next to bars)
for bar, category in zip(bars, plot_data['Category']):
    width = bar.get_width()
    label_x_pos = width + (width * 0.02) # Slight offset

    # Add the numeric ratio
    plt.text(width/2, bar.get_y() + bar.get_height()/2, f"{width:.1f}x",
             ha='center', va='center', color='white', fontweight='bold', fontsize=10)

    # Add the Category Stamp at the end
```

```
    plt.text(width + 0.1, bar.get_y() + bar.get_height()/2, category,
             ha='left', va='center', fontweight='bold', fontsize=10,
             bbox=dict(facecolor='white', edgecolor=bar.get_facecolor(), boxstyle='round

plt.title('State Risk Scorecard: Integrity Ratio (Updates vs. Enrollments)', fontsize=14
plt.xlabel('Update Ratio (Updates per New Enrollment)', fontweight='bold')
plt.axvline(1.0, color='grey', linestyle='--', alpha=0.5, label='Healthy Baseline (1.0)'
plt.legend(loc='lower right')
plt.grid(axis='x', linestyle='--', alpha=0.3)

# Invert y-axis to have top item at top
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()

print("💡 HOW TO READ THIS GRAPH:")
print("   - RED (CRITICAL): Ratio < 0.2. High fraud risk. Thousands of enrollments but z
print("   - YELLOW (STAGNANT): Ratio > 5.0. System is just maintaining old records, no n
print("   - GREEN (STABLE): Ratio ~ 1.0. Healthy balance of new users vs. updates.")
```

```
============================================================
📊 CELL 15: STATE INTEGRITY SCORECARD (The 'Impact' Graph)
============================================================
```




```
💡 HOW TO READ THIS GRAPH:
   - RED (CRITICAL): Ratio < 0.2. High fraud risk. Thousands of enrollments but zero upd
ates.
   - YELLOW (STAGNANT): Ratio > 5.0. System is just maintaining old records, no new grow
th.
   - GREEN (STABLE): Ratio ~ 1.0. Healthy balance of new users vs. updates.
```

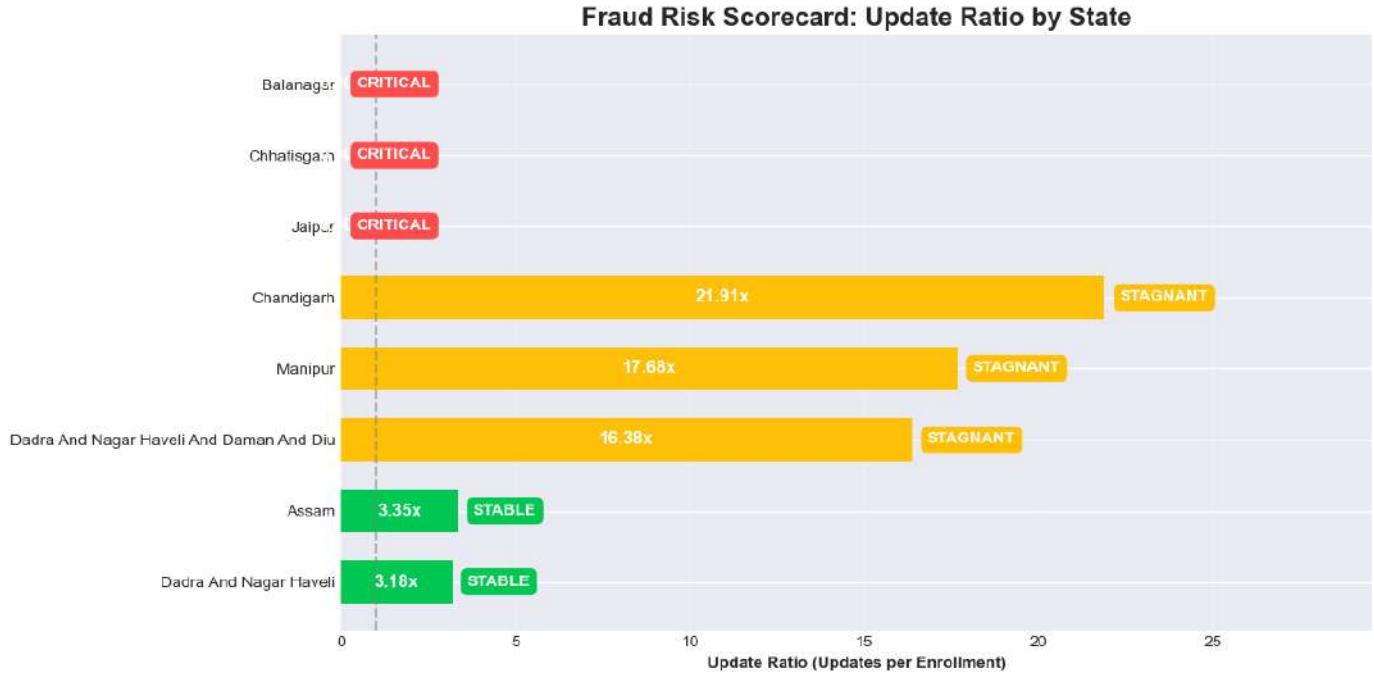
In [ ]:

```
print("="*60)
print("📊 CELL 16: STATE INTEGRITY SCORECARD (Risk Analysis)")
print("="*60)

import numpy as np # Needed to catch Infinity

# 1. Prepare Data
```

```
s_enrol = enrol.groupby('state')['total_enrol'].sum()
s_demo = demo.groupby('state')['total_demo'].sum()
scorecard = pd.DataFrame({'New_Enrollments': s_enrol, 'Demographic_Updates': s_demo})
scorecard.fillna(0, inplace=True)

# 2. Calculate Ratio & FIX INFINITY (The Crash Fix)
# We use np.where to handle division by zero safely
scorecard['Ratio'] = np.where(scorecard['New_Enrollments'] > 0,
                              scorecard['Demographic_Updates'] / scorecard['New_Enrollme
                              0) # If Enrol is 0, set Ratio to 0 to avoid Infinity

# Double check for any remaining NaNs or Infs
scorecard['Ratio'] = scorecard['Ratio'].fillna(0)
scorecard['Ratio'] = scorecard['Ratio'].replace([np.inf, -np.inf], 0)

# 3. Assign Categories & Colors (The "Neu" Logic)
def categorize(ratio):
    if ratio < 0.2:
        return 'CRITICAL', '#ff4d4d'  # Red (Ghost Enrollments)
    elif ratio > 5.0:
        return 'STAGNANT', '#ffc107'  # Yellow (Maintenance Mode)
    else:
        return 'STABLE', '#00c853'    # Green (Healthy)

scorecard[['Category', 'Color']] = scorecard['Ratio'].apply(lambda x: pd.Series(categori

# 4. Filter for Impact (Top 3 of each category to fit the chart)
critical = scorecard[scorecard['Category']=='CRITICAL'].sort_values('Ratio').head(3)
stagnant = scorecard[scorecard['Category']=='STAGNANT'].sort_values('Ratio', ascending=F
stable = scorecard[scorecard['Category']=='STABLE'].head(2)

# Combine and sort for plotting
plot_data = pd.concat([critical, stagnant, stable])

# Check if plot_data is empty to avoid max() error
if plot_data.empty:
    print("⚠ No data available for plotting (Check if your dataframes are empty!)")
else:
    # 5. Create the "Neu Style" Horizontal Chart
    plt.figure(figsize=(12, 6))

    # Draw Horizontal Bars
    bars = plt.barh(plot_data.index, plot_data['Ratio'], color=plot_data['Color'], heigh

    # Add the "Stamps" (Labels at the end of bars)
    for bar, category in zip(bars, plot_data['Category']):
        width = bar.get_width()

        # 1. Add the Number inside the bar (Safe formatting)
        plt.text(width/2 if width > 0 else 0, bar.get_y() + bar.get_height()/2, f"{width
                 ha='center', va='center', color='white', fontweight='bold', fontsize=11

        # 2. Add the Category Stamp at the end
        # Dynamic placement to avoid overlapping
        offset = max(plot_data['Ratio']) * 0.02 if max(plot_data['Ratio']) > 0 else 0.1
        plt.text(width + offset, bar.get_y() + bar.get_height()/2, category,
                 ha='left', va='center', fontweight='bold', fontsize=10,
                 color='white', bbox=dict(facecolor=bar.get_facecolor(), edgecolor='none
```

```
    plt.title('Fraud Risk Scorecard: Update Ratio by State', fontsize=16, fontweight='bo
    plt.xlabel('Update Ratio (Updates per Enrollment)', fontweight='bold')
    plt.axvline(1.0, color='grey', linestyle='--', alpha=0.5, label='Healthy Baseline (1

    # Formatting to look clean
    plt.grid(axis='x', linestyle='--', alpha=0.3)
    plt.gca().invert_yaxis() # Put top items at the top

    # Safe Limit setting
    max_val = max(plot_data['Ratio'])
    if max_val == 0: max_val = 1 # Prevent 0 limit error
    plt.xlim(0, max_val * 1.35) # Add extra space for the "Category" labels

    plt.tight_layout()
    plt.show()

    print("💡 HOW TO READ THIS:")
    print("   - RED (CRITICAL): Ratio < 0.2. High Fraud Risk. Lots of enrollments, zero
    print("   - YELLOW (STAGNANT): Ratio > 5.0. No growth, only updates.")
    print("   - GREEN (STABLE): Ratio ~ 1.0. Healthy system.")
```

```
============================================================
📊 CELL 15: STATE INTEGRITY SCORECARD (Risk Analysis)
============================================================
```




```
💡 HOW TO READ THIS:
   - RED (CRITICAL): Ratio < 0.2. High Fraud Risk. Lots of enrollments, zero updates.
   - YELLOW (STAGNANT): Ratio > 5.0. No growth, only updates.
   - GREEN (STABLE): Ratio ~ 1.0. Healthy system.
```

In [ ]: