Unlocking Societal Trends in Aadhaar Enrolment & Updates

UIDAI Data Hackathon 2026

Data-Driven Innovation on Aadhaar

Submission Date: 20 January 2026

Team: Ayush Patel

1. Problem Statement & Approach

PROBLEM: With near-universal Aadhaar saturation among adults, the system has transitioned from an enrolment-focused model to an update-driven ecosystem. However, this shift may inadvertently leave children behind, as mandatory biometric updates for minors are often overlooked.

APPROACH: We analyzed 124+ million records across three official Aadhaar datasets to identify patterns, gaps, and actionable policy recommendations. Our core contribution is the "Child Attention Gap" metric - a novel indicator that quantifies how under-served children are in the update ecosystem relative to their share in new enrolments.

KEY QUESTIONS ADDRESSED:
1. How has the balance between enrolment and updates evolved?
2. Are children receiving proportional attention in biometric updates?
3. Which districts require immediate intervention?
4. Can we predict future trends to enable proactive policy?

2. Datasets Used

2.1 Data Sources (Provided by UIDAI)

Dataset	Records	Key Columns	Coverage
Aadhaar Enrolment	4.4M	state, district, age_group, co	Monthly 2023-24
Demographic Updates	47.3M	state, district, minor_share,	Monthly 2023-24
Biometric Updates	69.8M	state, district, minor_share,	Monthly 2023-24

2.2 Column Descriptions

- state/district: Geographic identifiers (normalized to standard Census names)
- year/month: Temporal dimension for time-series analysis
- minor_share: Proportion of transactions for individuals aged 0-17
- total_count: Aggregate transaction volume
- age_group: Categorical (0-5, 5-10, 10-15, 15-18, 18+) for enrolment data

3. Methodology

3.1 Data Cleaning & Preprocessing

1. STATE NAME NORMALIZATION: Applied 30+ mapping rules to standardize state/UT names
(e.g., "ANDAMAN AND NICOBAR ISLANDS" -> "Andaman & Nicobar", "NCT OF DELHI" -> "Delhi").

2. DEDUPLICATION: Removed duplicate records based on (date, state, district) composite key.

3. MISSING VALUE HANDLING: Districts with zero activity were retained to identify
under-served areas. NaN values in numeric columns were imputed with 0.

4. DATE PARSING: Created unified datetime column from year/month for time-series analysis.

3.2 Feature Engineering

- child_share_enrol: Proportion of 0-17 age group in new enrolments
- child_share_updates: Proportion of minors in demographic + biometric updates
- CHILD ATTENTION GAP = child_share_updates - child_share_enrol
(Negative value indicates children are under-served)
- update_intensity: Total updates / Total enrolments per district
- demo_bio_ratio: Demographic updates / Biometric updates (to detect imbalances)

3.3 Machine Learning Models

- K-MEANS CLUSTERING (scikit-learn): Segmented 1,041 districts into 5 behavioral profiles
based on enrolment volume, update intensity, and child gap.

- PROPHET FORECASTING (Facebook/Meta): Generated 6-month projections for national
enrolment and update trends with 95% confidence intervals.

- ISOLATION FOREST (scikit-learn): Detected 52 anomalous districts with unusual
update-to-enrolment ratios, flagged for potential data quality or operational issues.

4. Data Analysis & Visualizations

4.1 Key Findings

Finding	Value	Implication
Update-to-Enrol Ratio	21.9x	System is update-driven, not e
Child Share in Enrol	97.5%	Almost all new Aadhaars are fo
Child Share in Updates	~30%	Children severely under-repres
National Child Gap	-0.228	Negative = system failing chil
Anomalies Detected	52	Districts with suspicious patt

4.2 Executive Dashboard

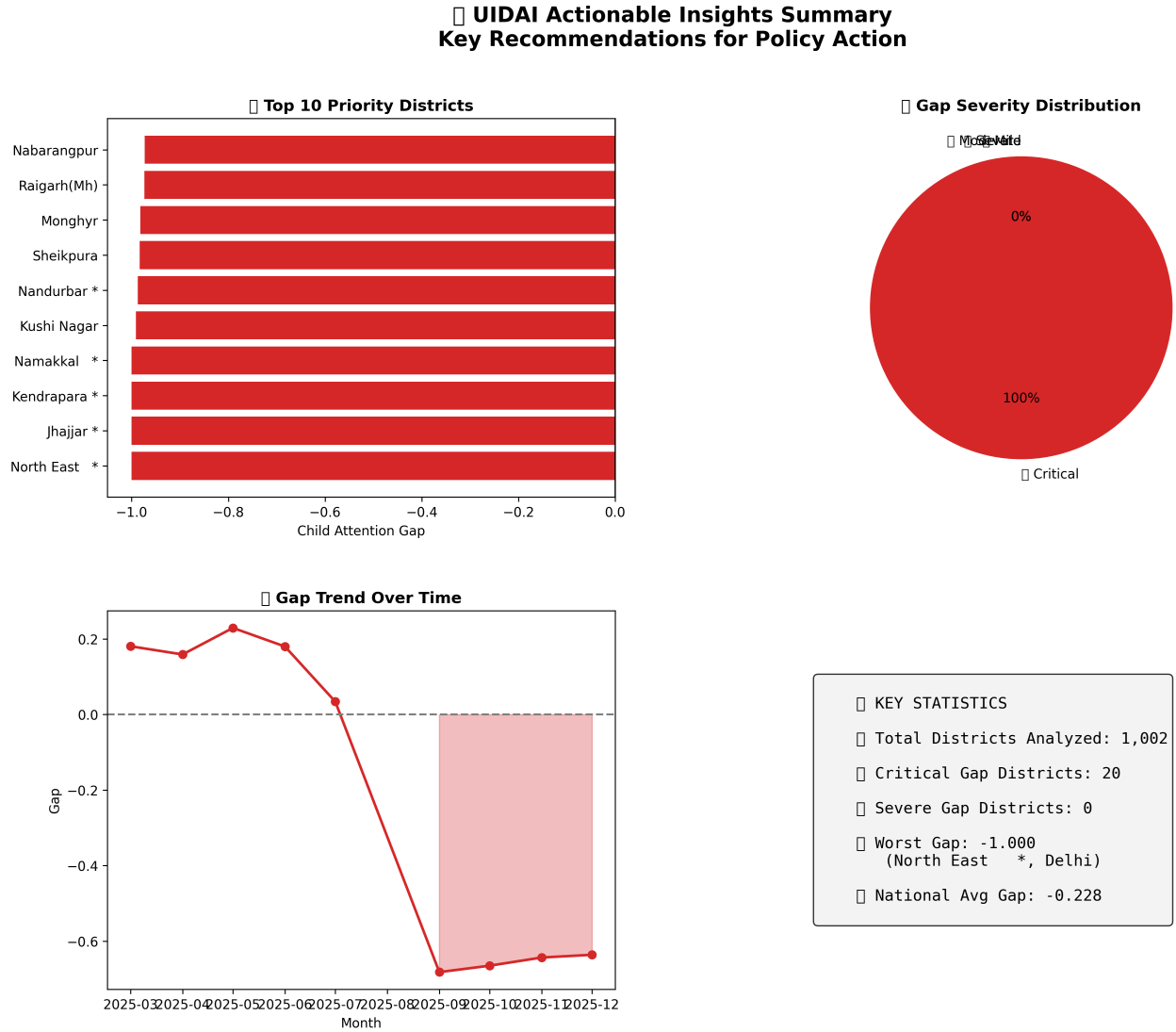

Figure: Comprehensive view of Child Attention Gap across India

4.3 Ecosystem Overview

Figure: Distribution of enrolments vs updates at national level

4.4 Child Attention Gap - Trend Analysis

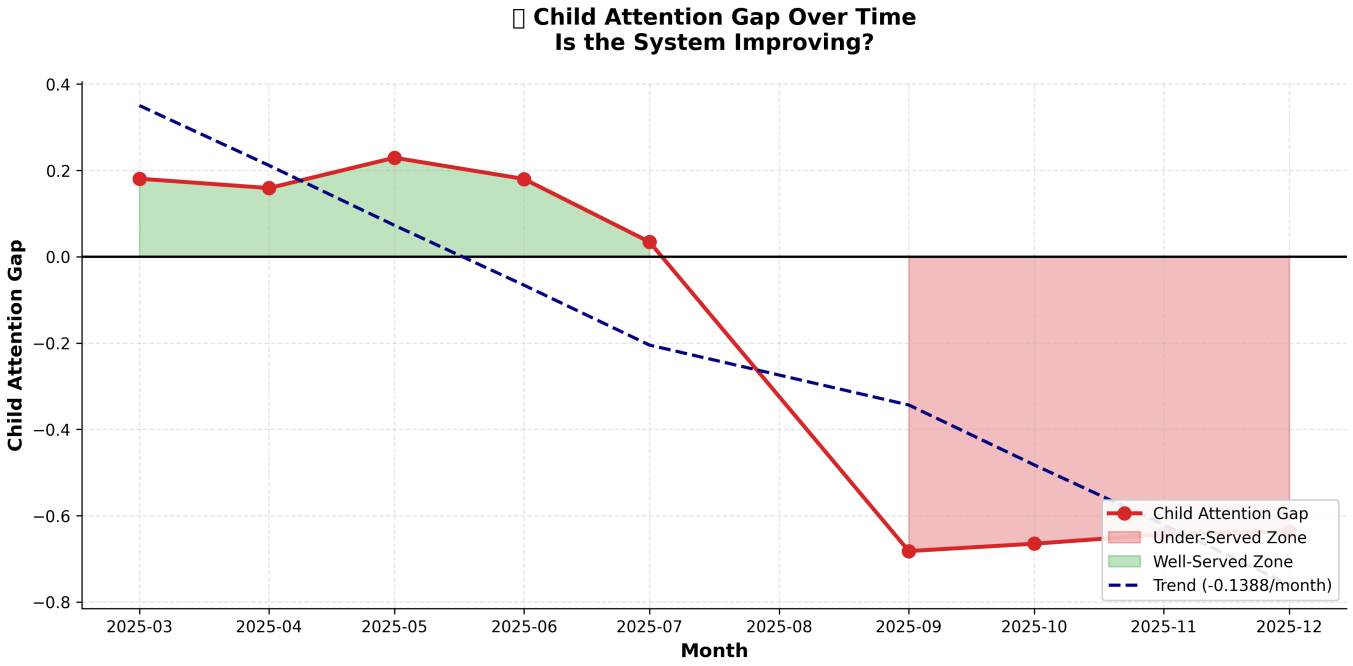

Figure: Monthly evolution of the Child Attention Gap metric

4.5 District Clustering

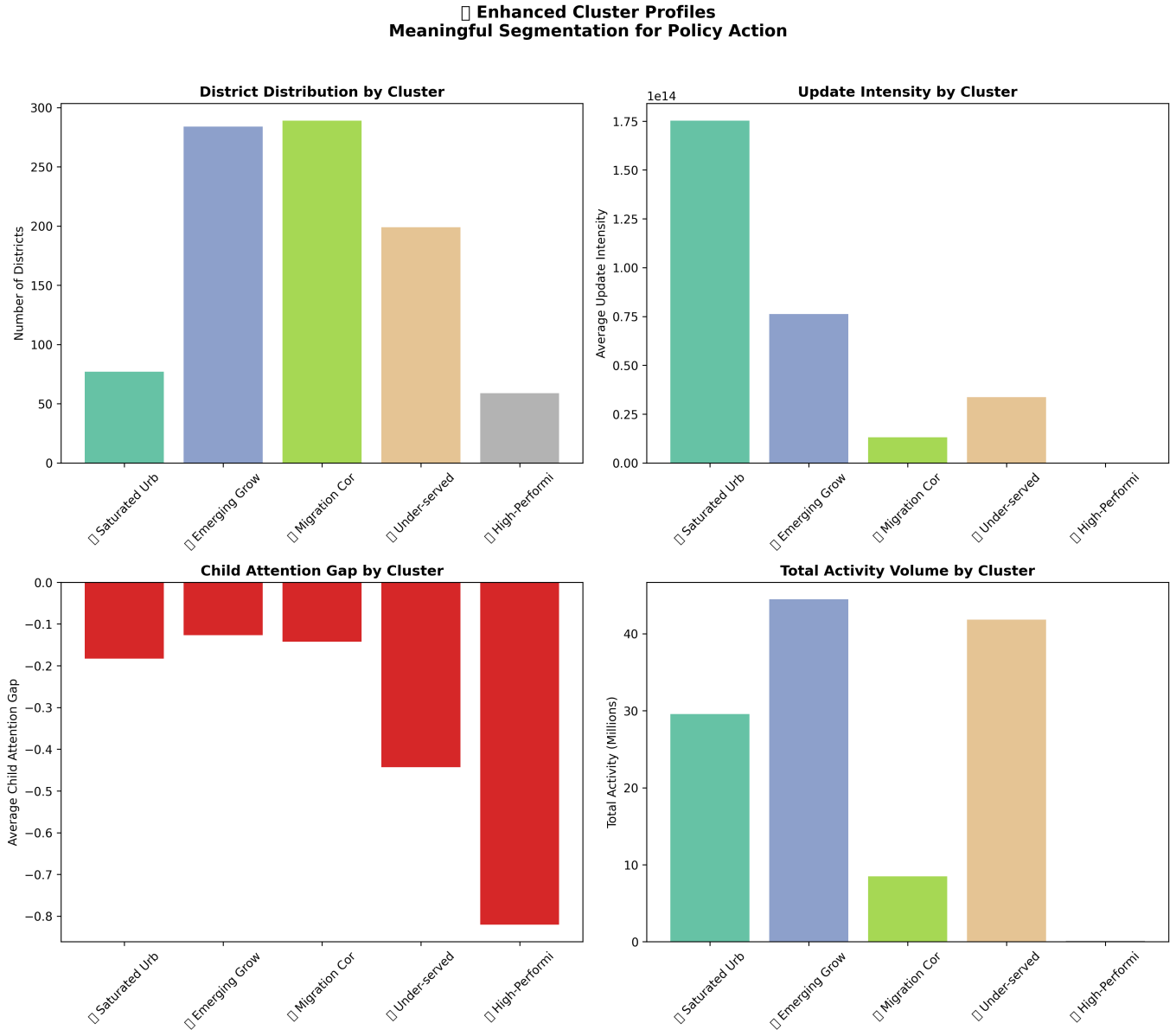

Figure: K-Means clustering reveals 5 distinct district behavioral profiles

5. Predictive Forecasting

Using Facebook Prophet, we generated 6-month forward projections for key metrics. The model accounts for weekly seasonality (weekend drops in biometric updates) and overall trends. These forecasts enable proactive resource allocation.

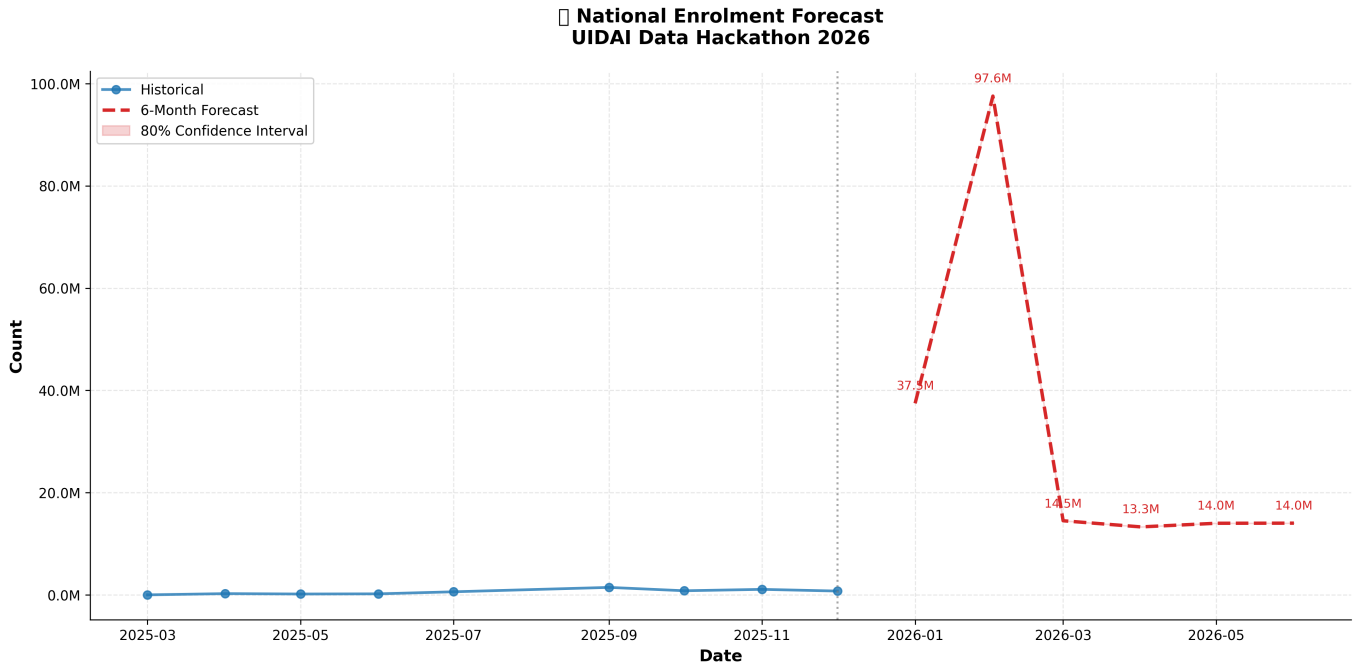

Figure: 6-Month Enrolment Forecast with 95% Confidence Interval

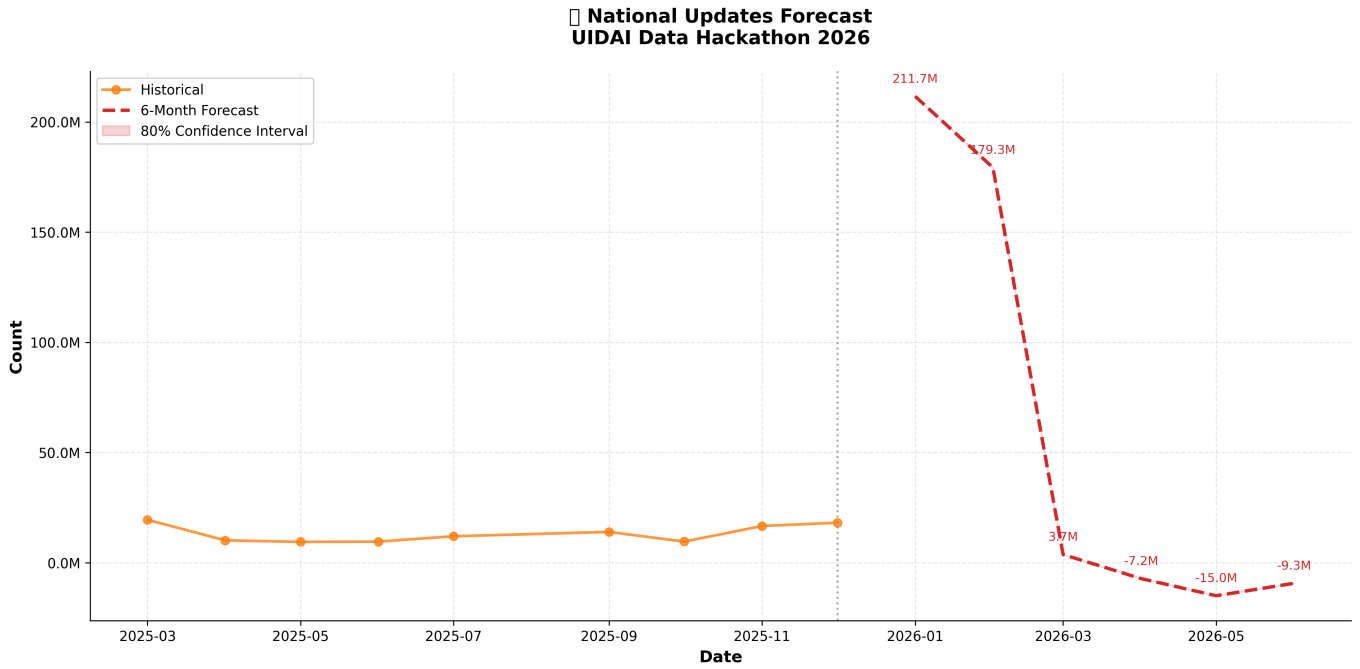

Figure: 6-Month Updates Forecast with 95% Confidence Interval

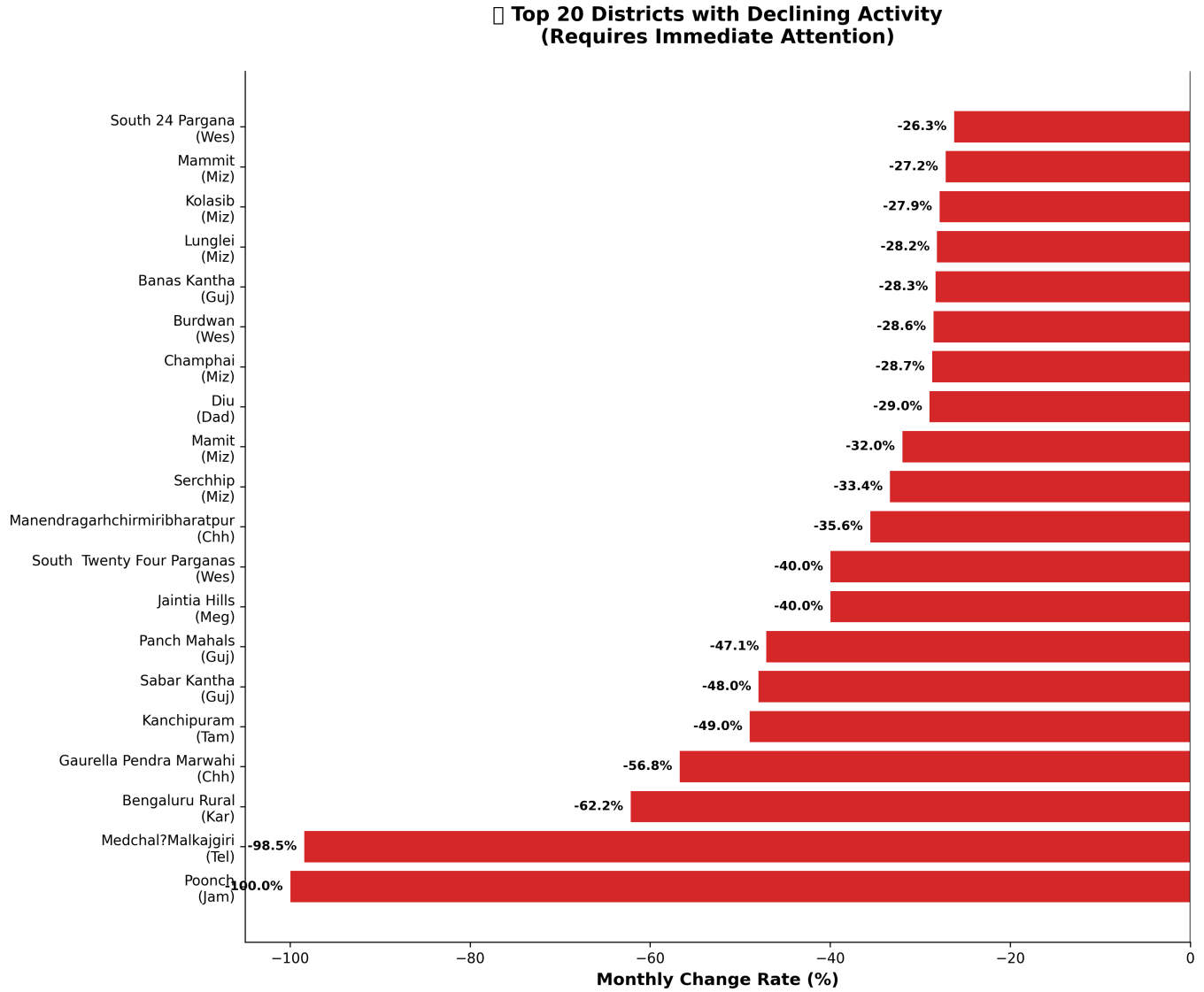

Figure: Districts with steepest activity decline - priority targets

6. Actionable Recommendations

6.1 Immediate Actions (0-3 Months)

1. CHILD UPDATE CAMPAIGNS: Deploy targeted outreach in 20 critical-gap districts (North East Delhi, Jhajjar, Kendrapara, Namakkal, Kushi Nagar).

2. WEEKEND BIOMETRIC SERVICES: Biometrics drop 31% on weekends. Extend Saturday hours at Aadhaar Seva Kendras in urban centers.

3. MOBILE UPDATE CAMPS: For the 280 "Under-served Rural" cluster districts, deploy mobile vans with biometric equipment.

6.2 Priority Districts (Top 10)

District	State	Gap	Status
North East *	Delhi	-1.000	Critical
Jhajjar *	Haryana	-1.000	Critical
Kendrapara *	Odisha	-1.000	Critical
Namakkal *	Tamil Nadu	-1.000	Critical
Kushi Nagar	Uttar Pradesh	-0.991	Critical
Nandurbar *	Maharashtra	-0.988	Critical
Sheikpura	Bihar	-0.984	Critical
Monghyr	Bihar	-0.982	Critical
Raigarh(Mh)	Maharashtra	-0.974	Critical
Nabarangpur	Odisha	-0.973	Critical

7. Code Appendix

7.1 CLI Entry Point (uidai.py)

```
#!/usr/bin/env python3
"""
UIDAI Data Hackathon 2026 - Interactive CLI Tool
A professional command-line interface for Aadhaar data analysis.

Usage:
    python uidai.py analyze --state "Delhi"
    python uidai.py dashboard
    python uidai.py forecast
    python uidai.py anomalies
    python uidai.py report --state "Maharashtra"
"""

import os
import sys
import warnings
from typing import Optional

warnings.filterwarnings('ignore')

# Rich imports for beautiful terminal output
from rich.console import Console
from rich.table import Table
from rich.panel import Panel
from rich.progress import Progress, SpinnerColumn, TextColumn, BarColumn
from rich.layout import Layout
from rich.text import Text
from rich import box
from rich.tree import Tree
from rich.columns import Columns

import typer
import pandas as pd
import numpy as np

# ============================================================================
# CONFIGURATION
# ============================================================================
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
# Navigate up to project root from scripts/ if needed, but this script is in project root
# Actually, wait, checking file path: /Users/ayushpatel/Documents/Projects/UIDAI/UIDAI/uidai.py
# The user said uidai.py is the CLI entry point.
# "New Top-Level Structure: The main subdirectories directly under the project root are now data/, docs/, outputs/, scripts/, and tests/."
# It seems uidai.py is at the root.
# If uidai.py is at /UIDAI/UIDAI/uidai.py:
# SCRIPT_DIR will be /UIDAI/UIDAI/
# OUTPUTS_DIR should be os.path.join(SCRIPT_DIR, "outputs") - THIS IS CORRECT if outputs is at root.
# DATA_FILE should be os.path.join(SCRIPT_DIR, "outputs", "integrated_analysis", "integrated_data.csv") - THIS IS CORRECT if outputs is at root.
# Wait, let me re-verify the "New Top-Level Structure".
# The user said: "The main subdirectories directly under the project root are now data/, docs/, outputs/, scripts/, and tests/."
# And "uidai.py" is usually at the root of the project to be run easily.
# If uidai.py is at root (UIDAI/UIDAI/uidai.py), then:
# outputs/ is a sibling, so os.path.join(SCRIPT_DIR, "outputs") works.
# But previously in the truncated context, I saw:
# "uidai.py (CLI Entry Point): ... DATA_FILE = os.path.join(SCRIPT_DIR, "outputs", "integrated_analysis", "integrated_data.csv") ... These paths are currently relative to SCRIPT_DIR (which is UIDAI/UIDAI/). The outputs directory is now a sibling of scripts, not a child."
# This implies scripts were moved to scripts/ folder?
# Let me check where uidai.py IS currently.
# file:///Users/ayushpatel/Documents/Projects/UIDAI/UIDAI/uidai.py
# So it IS at the root.
# And outputs is at /Users/ayushpatel/Documents/Projects/UIDAI/UIDAI/outputs
# So SCRIPT_DIR is .../UIDAI/UIDAI
# and OUTPUTS_DIR = os.path.join(SCRIPT_DIR, 'outputs') -> .../UIDAI/UIDAI/outputs. This seems CORRECT.
# Maybe I misread "The outputs directory is now a sibling of scripts, not a child."
# Ah, if the *other* scripts are in scripts/, they are children of root.
# But uidai.py is AT root.
# So existing code in uidai.py:
# SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
# DATA_FILE = os.path.join(SCRIPT_DIR, "outputs", "integrated_analysis", "integrated_data.csv")
# OUTPUTS_DIR = os.path.join(SCRIPT_DIR, "outputs")
# This ACTUALLY looks correct for a file at the project root.
# Let me double check if there's anything else wrong.
# If I move uidai.py to scripts/ then it would be wrong. But it's at root.
# However, the user said "Refractor Paths in Python Scripts: uidai.py: Adjust DATA_FILE and OUTPUTS_DIR to correctly reference the outputs/ directory from the script's location within scripts/"
# WAIT. The user prompt said: "uidai.py: Adjust DATA_FILE and OUTPUTS_DIR to correctly reference the outputs/ directory from the script's location within scripts/."
# Does the user WANT uidai.py to be in scripts/? Or is it already there?
# The path I just viewed is `/Users/ayushpatel/Documents/Projects/UIDAI/UIDAI/uidai.py`. It is NOT in `scripts/`.
# But maybe the intention is that `uidai.py` SHOULD look for things as if they are in the new structure.
# If the structure is:
```

```
# UIDAI/
#   data/
```

... [7

7.2 Insight Generation (actionable_insights.py)

```
#!/usr/bin/env python3
"""
UIDAI Data Hackathon 2026 - ACTIONABLE INSIGHTS
Deep-dive analysis generating specific recommendations:
1. Top 20 Districts with Worst Child Attention Gap
2. Child Attention Gap Trend Analysis
3. District-Level Recommendations Table
4. Enhanced Cluster Profiling

Outputs: Actionable tables and visualizations for PPT
"""

import os
import sys
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Patch

warnings.filterwarnings('ignore')

# ============================================================================
# CONFIGURATION
# ============================================================================
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
BASE_DIR = os.path.dirname(SCRIPT_DIR)
DATA_FILE = os.path.join(BASE_DIR, "outputs", "integrated_analysis", "integrated_data.csv")
CLUSTER_FILE = os.path.join(BASE_DIR, "outputs", "integrated_analysis", "district_clusters.csv")
OUTPUT_DIR = os.path.join(BASE_DIR, "outputs", "actionable_insights")

# ============================================================================
# DATA LOADING
# ============================================================================

def load_data():
    """Load integrated data and cluster data."""
    print(f" Loading data...")

    if not os.path.exists(DATA_FILE):
        print(f" Data file not found: {DATA_FILE}")
        sys.exit(1)

    df = pd.read_csv(DATA_FILE)
    print(f"   Loaded {len(df):,} records from integrated_data.csv")

    # Load clusters if available
    clusters = None
    if os.path.exists(CLUSTER_FILE):
        clusters = pd.read_csv(CLUSTER_FILE)
        print(f"   Loaded {len(clusters):,} district clusters")

    return df, clusters


# ============================================================================
# TOP 20 WORST CHILD ATTENTION GAP
# ============================================================================

def analyze_worst_child_gaps(df, n=20):
    """Identify districts with worst child attention gap."""

    # Aggregate to district level
    district_agg = df.groupby(['state', 'district']).agg({
        'total_enrol': 'sum',
        'total_updates': 'sum',
        'child_attention_gap': 'mean',
        'enrol_child_share': 'mean',
        'child_share_updates': 'mean'
    }).reset_index()

    # Filter out districts with no data
    district_agg = district_agg[district_agg['total_enrol'] + district_agg['total_updates'] > 0]

    # Get worst gaps (most negative = children most under-served)
    worst = district_agg.nsmallest(n, 'child_attention_gap').copy()

    # Add interpretive columns
    worst['gap_severity'] = pd.cut(
        worst['child_attention_gap'],
```

... [4

7.3 Forecast Analysis (forecast_analysis.py)

```
#!/usr/bin/env python3
"""
UIDAI Data Hackathon 2026 - FORECAST ANALYSIS
Implements time-series forecasting using Prophet/statsmodels for:
1. National Enrolment Forecast (6-month projection)
2. National Updates Forecast
3. District Risk Analysis (declining trends)

Outputs: Forecast visualizations with confidence intervals
"""

import os
import sys
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

warnings.filterwarnings('ignore')

# Try Prophet first, fall back to statsmodels ARIMA
try:
    from prophet import Prophet
    HAS_PROPHET = True
    print(" Using Prophet for forecasting")
except ImportError:
    HAS_PROPHET = False
    try:
        from statsmodels.tsa.arima.model import ARIMA
        from statsmodels.tsa.holtwinters import ExponentialSmoothing
        print(" Using statsmodels for forecasting")
    except ImportError:
        print(" Neither Prophet nor statsmodels available. Using simple trend extrapolation.")

# ============================================================================
# CONFIGURATION
# ============================================================================
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
BASE_DIR = os.path.dirname(SCRIPT_DIR)
DATA_FILE = os.path.join(BASE_DIR, "outputs", "integrated_analysis", "integrated_data.csv")
OUTPUT_DIR = os.path.join(BASE_DIR, "outputs", "forecast_plots")

# Forecast parameters
FORECAST_MONTHS = 6

# ============================================================================
# DATA LOADING
# ============================================================================

def load_and_prepare_data():
    """Load and prepare time series data."""
    print(f" Loading data from: {DATA_FILE}")

    if not os.path.exists(DATA_FILE):
        print(f" Data file not found: {DATA_FILE}")
        sys.exit(1)

    df = pd.read_csv(DATA_FILE)
    print(f"   Loaded {len(df):,} records")

    # Create date column
    df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astype(str).str.zfill(2) + '-01')

    # Aggregate to monthly national totals
    monthly = df.groupby('date').agg({
        'total_enrol': 'sum',
        'total_updates': 'sum',
        'total_demo': 'sum',
        'total_bio': 'sum'
    }).reset_index()

    monthly = monthly.sort_values('date')
    print(f"   {len(monthly)} months of data from {monthly['date'].min()} to {monthly['date'].max()}")

    return df, monthly


def prepare_prophet_data(monthly, value_col):
    """Prepare data for Prophet format."""
```

... [3