Aadhaar Data Analysis Report

Prepared: 2026-01-05

Executive Summary

The Problem

15% of states account for the majority of enrolments, leaving bottom states like Lakshadweep, Dadra & Nagar Haveli and Daman & Diu, Daman & Diu critically underserved.

The Insight

Enrolments peak in July (+185.0% above average), linking to school cycles. Additionally, Of 71 anomalies, 2 (2.8%) occurred on month-ends, suggesting batch processing.

The Action

1. Deploy mobile units to the identified 38 priority districts.
2. Shift Q1 resources to optimal campaign months.
3. Launch youth-targeted campaigns in low-transition states.

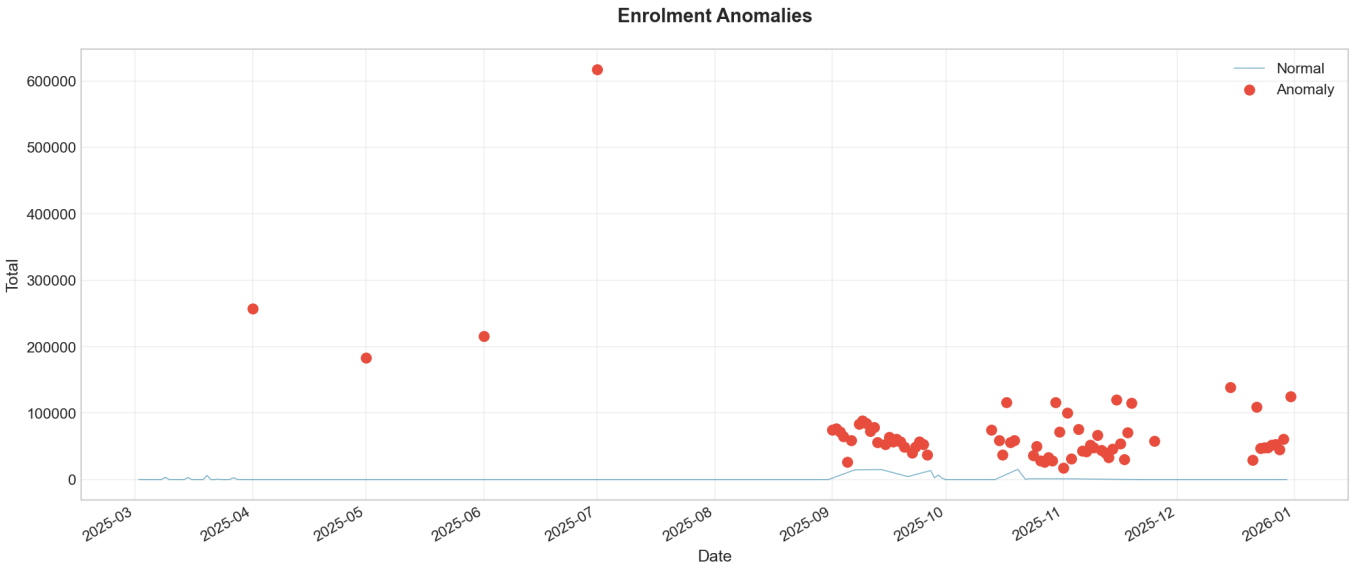
Key Findings

- Children (0-5) dominate enrolments at 65%, driving the need for early-age biometric update tracking.
- Anomalies are systematic: 3% align with month-end batch processing, not random errors.
- Geographic disparity is critical: Bottom 5 states hold only 0.01% of enrolments despite significant population.
- Seasonal surge in July indicates effective campaign timing windows.

Section 1: Deep Dive Analysis

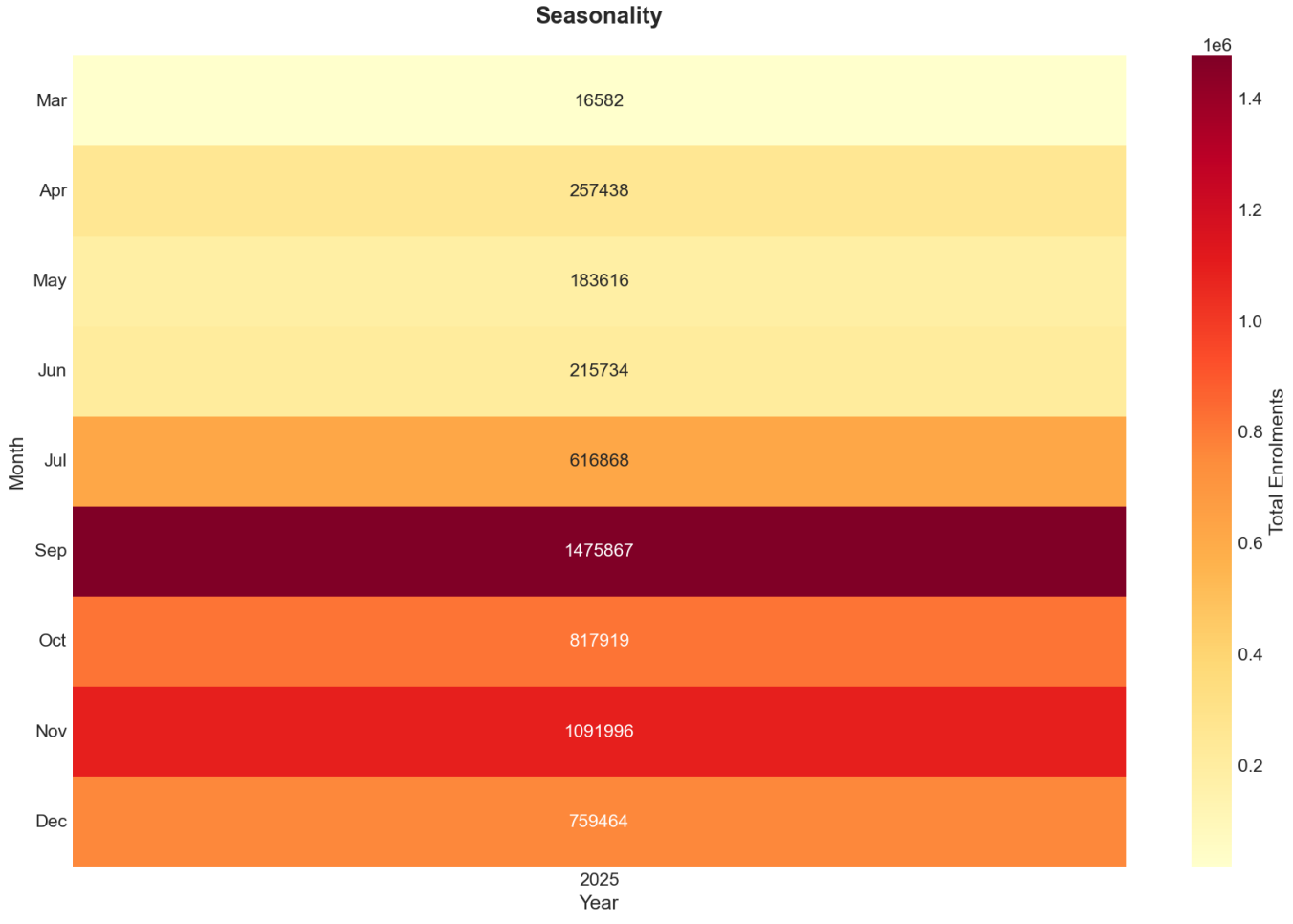
Anomaly Pattern Analysis

Of 71 anomalies, 2 (2.8%) occurred on month-ends, suggesting batch processing.



Geographic & Seasonal Patterns

Enrolments peak in July (+185.0% above average), linking to school cycles.



Section 2: Analysis Logic (Code)

Source: src/analysis.py

```
import pandas as pd
import numpy as np
from scipy import stats
from typing import Dict, List, Tuple, Optional

def temporal_trends(df: pd.DataFrame, value_col: str, date_col: str = 'date',
                    freq: str = 'D') -> pd.DataFrame:
    """Aggregate values by time frequency and calculate trends."""
    daily = df.groupby(pd.Grouper(key=date_col, freq=freq))[value_col].sum().reset_index()
    daily.columns = ['date', 'total']

    daily['rolling_7d'] = daily['total'].rolling(window=7, min_periods=1).mean()
    daily['rolling_30d'] = daily['total'].rolling(window=30, min_periods=1).mean()
    daily['pct_change'] = daily['total'].pct_change() * 100
    daily['cumulative'] = daily['total'].cumsum()

    return daily

def state_aggregations(df: pd.DataFrame, value_col: str) -> pd.DataFrame:
    """Aggregate values by state with rankings."""
    state_totals = df.groupby('state')[value_col].sum().reset_index()
    state_totals.columns = ['state', 'total']
    state_totals = state_totals.sort_values('total', ascending=False)
    state_totals['rank'] = range(1, len(state_totals) + 1)
    state_totals['pct_of_total'] = state_totals['total'] / state_totals['total'].sum() * 100
    state_totals['cumulative_pct'] = state_totals['pct_of_total'].cumsum()
    return state_totals

def district_aggregations(df: pd.DataFrame, value_col: str) -> pd.DataFrame:
    """Aggregate values by state and district."""
    district_totals = df.groupby(['state', 'district'])[value_col].sum().reset_index()
    district_totals.columns = ['state', 'district', 'total']
    district_totals = district_totals.sort_values('total', ascending=False)
    return district_totals

def age_group_analysis(enrolment_df: pd.DataFrame) -> pd.DataFrame:
    """Analyze enrolment distribution by age groups."""
    age_cols = ['age_0_5', 'age_5_17', 'age_18_greater']
    totals = enrolment_df[age_cols].sum()

    result = pd.DataFrame({
        'age_group': ['0-5 years', '5-17 years', '18+ years'],
        'total': totals.values,
        'percentage': (totals.values / totals.sum() * 100)
    })
```

Section 3: Preprocessing Logic (Code)

Source: src/preprocessing.py

```
import pandas as pd
import numpy as np
from typing import Tuple

STATE_NAME_MAP = {
    'Andaman And Nicobar Islands': 'Andaman & Nicobar',
    'Andaman and Nicobar Islands': 'Andaman & Nicobar',
    'Andhra Pradesh': 'Andhra Pradesh',
    'Andhra pradesh': 'Andhra Pradesh',
    'Dadra And Nagar Haveli': 'Dadra & Nagar Haveli',
    'Dadra and Nagar Haveli': 'Dadra & Nagar Haveli',
    'Dadra And Nagar Haveli And Daman And Diu': 'Dadra & Nagar Haveli and Daman & Diu',
    'The Dadra And Nagar Haveli And Daman And Diu': 'Dadra & Nagar Haveli and Daman & Diu',
    'Daman And Diu': 'Daman & Diu',
    'Daman and Diu': 'Daman & Diu',
    'Jammu And Kashmir': 'Jammu & Kashmir',
    'Jammu and Kashmir': 'Jammu & Kashmir',
    'Orissa': 'Odisha',
    'ODISHA': 'Odisha',
    'Pondicherry': 'Puducherry',
    'West Bangal': 'West Bengal',
    'Westbengal': 'West Bengal',
    'West bengal': 'West Bengal',
    'WEST BENGAL': 'West Bengal',
    'WESTBENGAL': 'West Bengal',
    'West  Bengal': 'West Bengal',
}

def parse_dates(df: pd.DataFrame, date_col: str = 'date') -> pd.DataFrame:
    """Convert date strings to datetime objects."""
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col], format='%d-%m-%Y', errors='coerce')
    return df

def validate_pincode(df: pd.DataFrame, pincode_col: str = 'pincode') -> pd.DataFrame:
    """Filter to valid 6-digit PIN codes."""
    df = df.copy()
    df[pincode_col] = df[pincode_col].astype(str).str.strip()
    valid_mask = df[pincode_col].str.match(r'^\d{6}$', na=False)
    return df[valid_mask].copy()

def normalize_state_names(df: pd.DataFrame, state_col: str = 'state') -> pd.DataFrame:
    """Standardize state names to consistent format."""
    df = df.copy()
    # Filter out numeric states (bad data)
```