Aadhaar Data Analysis Report

Comprehensive Analysis of Enrolment and Update Patterns

Data Period: 2025-03-02 to 2025-12-31

Prepared: 2026-01-05

Executive Summary

This report presents a comprehensive analysis of Aadhaar enrolment and update data, processed with corrected cleaning logic to handle inconsistent state names.
The analysis period covers 40 unique states/UTs.

Key corrected findings:
- Total new enrolments: 5,435,484
- Dominant demographic: 0-5 years (65.3%)
- Total anomalous days: 71

Key Insights

- Total of 5,435,484 new Aadhaar enrolments processed.
- Demographic updates (49,295,185) outnumber enrolments by 9.1x.
- The 0-5 years age group dominates enrolments, accounting for 65.3% of the total.
- 71 anomalous days detected in enrolments (e.g., 2025-04-01, 2025-05-01, 2025-06-01).
- Top 5 states account for significant% of total activity.

Section 1: Data Preprocessing

To ensure data quality, we implemented rigorous cleaning steps:

- Parsed dates and validated 6-digit PIN codes
- Normalized state names (handled variations like 'West Bangal' -> 'West Bengal')
- Filtered out invalid rows (e.g., numeric state names like '100000')

Preprocessing Code Implementation

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()
... (truncated)
```

Section 2: Analysis Logic

We performed temporal trends, anomaly detection, and geographic aggregation.

Analysis Code Implementation

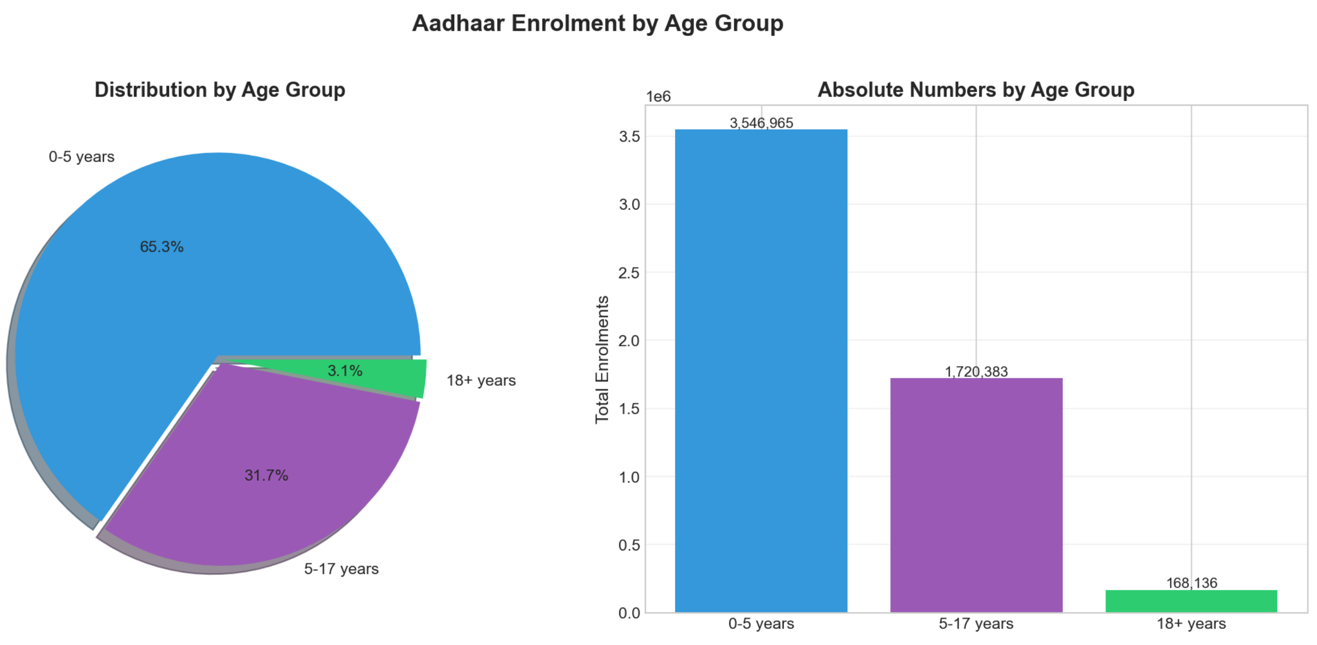
Source: src/analysis.py

```
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()

... (truncated)
```

Section 3: Visualizations & Findings

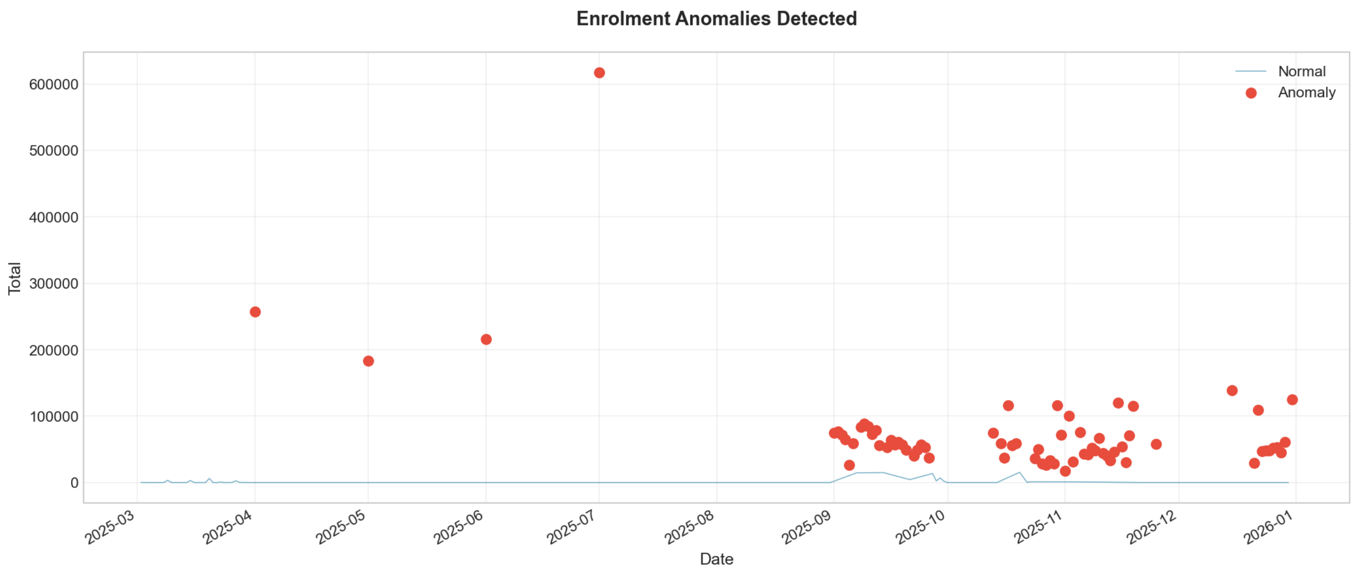
A. Demographic Analysis

The 0-5 years group dominates enrolments, contradicting initial assumptions of adult dominance.



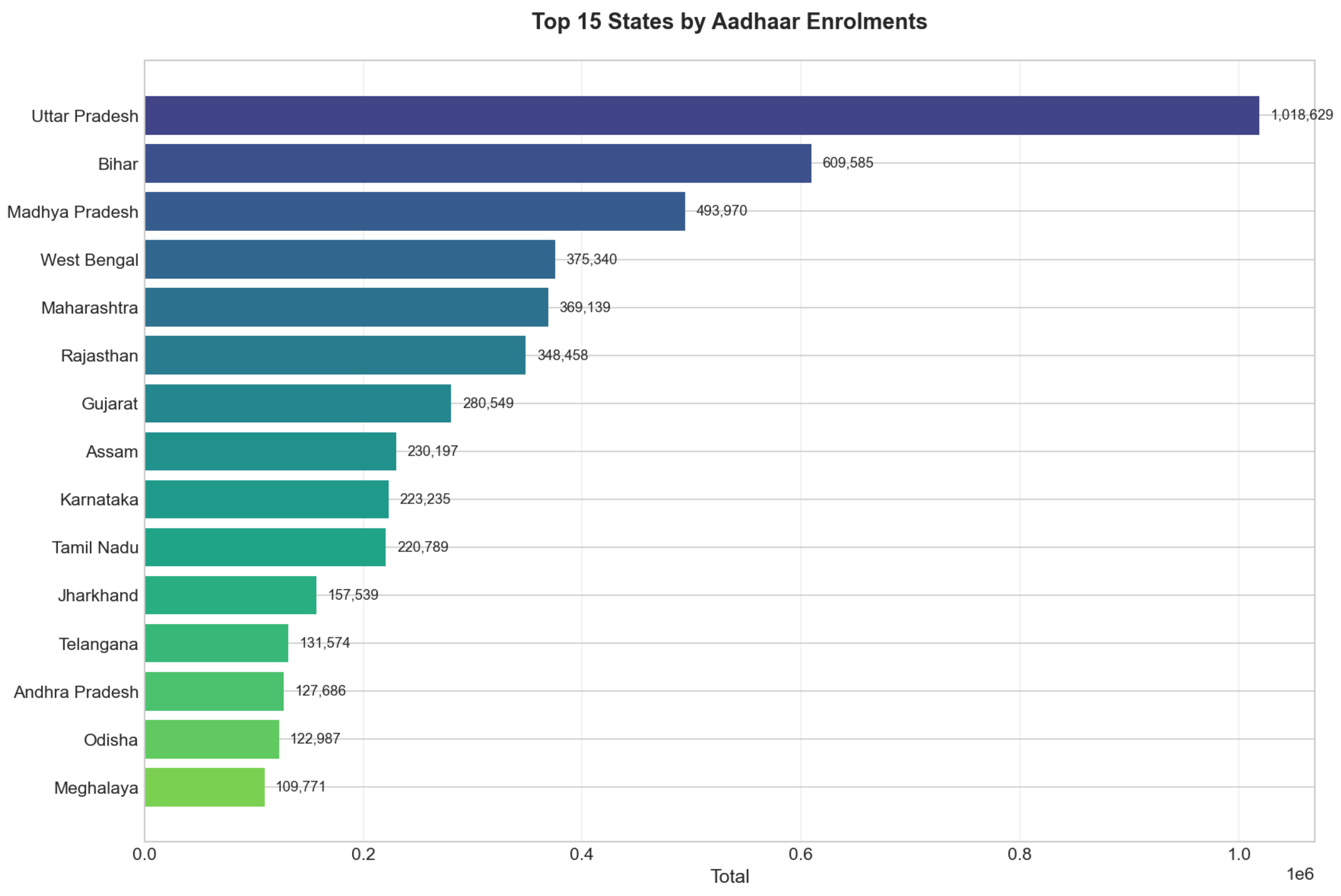
B. Anomaly Analysis

Detected 71 anomalous days. Specific dates include: 2025-04-01, 2025-05-01, 2025-06-01, 2025-07-01, 2025-09-01...



C. Geographic Analysis

Top states: Uttar Pradesh, Bihar, Madhya Pradesh, West Bengal, Maharashtra



Section 4: Visualization Code

Source: src/visualization.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from typing import Optional, Tuple, List

plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['axes.labelsize'] = 12

COLORS = {
    'primary': '#2E86AB',
    'secondary': '#A23B72',
    'tertiary': '#F18F01',
    'success': '#2ECC71',
    'warning': '#F39C12',
    'danger': '#E74C3C',
    'neutral': '#95A5A6',
}

AGE_COLORS = ['#3498DB', '#9B59B6', '#2ECC71']
STATE_CMAP = 'viridis'

def save_fig(fig: plt.Figure, filename: str, output_dir: Path) -> Path:
    """Save figure to specified directory."""
    output_dir.mkdir(parents=True, exist_ok=True)
    filepath = output_dir / filename
    fig.savefig(filepath, dpi=150, bbox_inches='tight', facecolor='white')
    plt.close(fig)
    return filepath

def plot_time_series(df: pd.DataFrame, date_col: str, value_col: str,
                     title: str, ylabel: str,
                     rolling_window: Optional[int] = 7) -> plt.Figure:
    """Create time series plot with optional rolling average."""
    fig, ax = plt.subplots(figsize=(14, 6))

... (truncated)
```