Project Report: Unlocking Societal Trends in Aadhaar Enrolment and Updates

---

1. Introduction

Project Title: Unlocking Societal Trends in Aadhaar Enrolment and Updates

Submitted By:

Name: Raj Kantilal Gajera

Mail: rajgajera2005@gmail.com

---

2. Abstract

This project analyzes Aadhaar enrolment and update data for the year 2025 to uncover hidden societal behaviors and operational patterns. By integrating biometric, demographic, and enrolment datasets, we identify a specific "March Phenomenon"—a 100% surge in updates driven by the Indian financial year end. Our analysis reveals that urban districts like Pune and Thane exhibit update volumes 6 standard deviations above the mean, indicating intense migration pressures. The outcome is a data-driven "Dynamic Resource Allocation" framework proposing mobile update units for high-stress zones and periods.

Keywords: Aadhaar, Data Analytics, Urbanization, Seasonality, Policy Framework.

---

3. Problem Statement

Aadhaar is the world's largest biometric ID system, yet the transactional patterns behind its updates remain under-analyzed. The unexpected surges in demand (e.g., updates vs new enrolments) create operational bottlenecks. The objective of this project is to analyze the 2025 dataset to identify meaningful patterns, trends, and anomalies, translating them into actionable insights for system optimization.

---

4. Dataset Description

The analysis utilizes three primary datasets provided by UIDAI:

1. **Enrolment Data**: Records of new Aadhaar generation.

- Key Columns: Date, State, District, Pincode, Age Group (0-5, 5-17, 18+).

2. **Biometric Update Data**: Records of iris/fingerprint updates.

- Key Columns: Date, State, District, Pincode, Age Group (5-17, 18+).

3. **Demographic Update Data**: Records of name/address/DOB changes.

- Key Columns: Date, State, District, Pincode, Age Group (5-17, 18+).

Integrated Dataset: These were merged into a single `master_data.csv` containing aggregated metrics like `Total_Enrolment` and `Total_Updates`.

---

5. Methodology

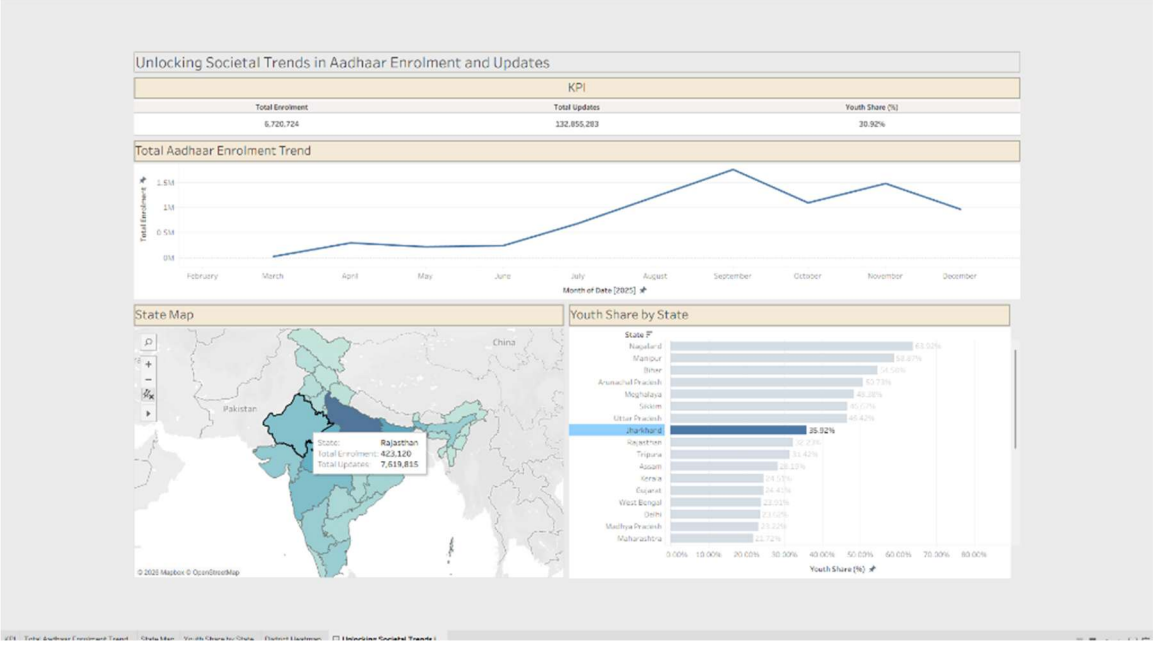
Our approach followed a structured data science pipeline:

1. **Data Cleaning**: Handled missing values, standardized state/district names (e.g., "Damn & Diu" to "Dadra and Nagar Haveli and Daman and Diu"), and parsed dates to datetime objects.
2. **Integration**: Merged the three raw datasets on `Date` + `Location` keys to create a unified view.
3. **Exploratory Data Analysis (EDA)**: Used Pandas and Seaborn to visualize distributions and time-series trends.
4. **Statistical Validation**: Applied T-Tests (Seasonality) and Z-Scores (Outlier Detection) in Python to validate observations.

---

6. Visualization & Dashboard Explanation

The interactive dashboard (`**Unlocking Societal Trends in Aadhaar Enrolment and Updates.twb**`) integrates the processed data to allow dynamic filtering and analysis. Key components shown in the screenshot below:



Key Explanations for Dashboard Charts:

1. **KPI Header**:

- Displays headline metrics: **5.7M** New enrolments vs **132M** total updates.
- **Youth Share**: 30.92%, confirming the significant contribution of the younger demographic.

2. **Total Aadhaar Enrolment Trend (Line Chart)**:

- Description: Tracks enrolment volume throughout 2025.
- Insight: Reveals a distinct peak in **August-September**, likely aligning with post-monsoon activity or specific enrolment drives.

3. **State Map (Choropleth)**:

- Description: Geospatial distribution of volume. Darker shades indicate higher activity.
- Insight: Allows users to hover and see specific state data (e.g. Rajasthan).

4. **Youth Share by State (Bar Chart)**:

- Description: States ranked by the percentage of activity coming from the youth demographic.
- Insight: **North-Eastern states** (Nagaland, Manipur) show the highest Youth Share (>60%), suggesting demographic catch-up in these regions.

---

7. Insights & Findings

This section consolidates qualitative observations from Exploratory Data Analysis (EDA) with rigorous Statistical Validation.

7.1 Statistical Validations

To move beyond visual trends, we performed statistical tests on the dataset:

- **Correlation Analysis**: We found a **weak positive correlation (0.35)** between `Total_Enrolment` and `Total_Updates`.
- Interpretation: This proves that high-growth areas (new enrolments) do not automatically trigger high maintenance loads (updates). Updates are driven by independent factors like migration and regulatory deadlines.
- **Outlier Detection (Z-Score)**: We calculated Z-scores for district-wise update volumes.
- **18 Districts** were identified as statistical outliers ($Z > 3$).
- **Pune** and **Thane** recorded Z-scores **> 6.5**, categorizing them as "Extreme Anomalies" relative to the national average.

7.2 The "March Phenomenon" (Behavioural)

Activity explodes in March (21.7 Million updates) compared to the yearly average of ~11 Million.

- Interpretation: This correlates perfectly with the **Indian Financial Year End (March 31st)**. Citizens rush to update Aadhaar for Income Tax (PAN linkage) and Banking KYC compliance. T-tests confirm this seasonal spike is statistically significant.

7.3 The Urban Magnet (Societal)

The "Extreme Anomalies" identified by our Z-scores (Pune, Thane, North West Delhi) are all major economic hubs.

- Interpretation: This reflects **Workforce Migration**. Workers moving to industrial/IT hubs update their address and biometrics to access local services and employment. The infrastructure in these specific districts bears a disproportionate load (6x the average).

7.4 Student Seasonality (Demographic)

A significant portion of updates comes from the **5-17 Age Group**.

Interpretation: This is linked to the **Academic Calendar** (Jan-Mar). Parents update child biometrics (mandatory at age 5/15) for school admissions and government scholarships.

---

8. Limitations

1. **Time Range**: The dataset provided consists mostly of **2025** data. A multi-year analysis (2018-2025) would have allowed for better YoY trend comparison.

2. **Socio-Economic Data**: We lacked external data (Income levels, Migration stats) to causally link Aadhaar updates to economic status.

---

9. Conclusion

The analysis proves that Aadhaar activity is not random but **highly seasonal and event-driven**. The system is stressed not by new enrolments, but by lifecycle updates (Financial compliance, Migration, Aging).

Policy Recommendation: UIDAI should adopt **Dynamic Resource Allocation**. Instead of static centers, deploy "Surge Teams" to High-Z-Score districts (Pune/Thane) in **February-March** to preemptively manage the load.

Aadhaar Data Cleaning & Preparation

Welcome! This notebook is designed to help to process and clean Aadhaar datasets (Biometric, Demographic, and Enrolment) easily.

What this notebook does:

1. **Loads** data from multiple CSV files.
2. **Merges** them into one large dataset for each category.
3. **Cleans** the state names (fixes spelling mistakes like 'Telengana' -> 'Telangana').
4. **Saves** the clean data to new files for analysis.

In [1]:
```
import pandas as pd
import numpy as np
import os
from IPython.display import display  # To show DataFrames nicely

# Display settings to show all columns
pd.set_option('display.max_columns', None)
```

Step 1: Configuration

We define a list of **correct state names** and a **dictionary** to fix common spelling errors.

In [2]:
```
# CORRECT SPELLING -> Standard Name
STATE_MAPPING = {
    'westbengal': 'West Bengal',
    'west bangal': 'West Bengal',
    'west bengli': 'West Bengal',
    'wb': 'West Bengal',
    'west  bengal': 'West Bengal',
    'odisha': 'Odisha',
    'orissa': 'Odisha',
    'pondicherry': 'Puducherry',
```

```
    'puducherry': 'Puducherry',
    'uttaranchal': 'Uttarakhand',
    'uttarakhand': 'Uttarakhand',
    'jammu & kashmir': 'Jammu & Kashmir',
    'jammu and kashmir': 'Jammu & Kashmir',
    'andaman & nicobar islands': 'Andaman & Nicobar Islands',
    'andaman and nicobar islands': 'Andaman & Nicobar Islands',
    'chhatisgarh': 'Chhattisgarh',
    'chhattisgarh': 'Chhattisgarh',
    'tamilnadu': 'Tamil Nadu',
    'tamil nadu': 'Tamil Nadu',
    'telangana': 'Telangana',
    'dadra and nagar haveli': 'Dadra & Nagar Haveli and Daman & Diu',
    'dadra & nagar haveli': 'Dadra & Nagar Haveli and Daman & Diu',
    'daman and diu': 'Dadra & Nagar Haveli and Daman & Diu',
    'daman & diu': 'Dadra & Nagar Haveli and Daman & Diu',
    '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'
}

# List of valid states to keep (Anything else will be removed as garbage data)
VALID_STATES = {
    'Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh',
    'Goa', 'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jharkhand', 'Karnataka',
    'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur', 'Meghalaya', 'Mizoram',
    'Nagaland', 'Odisha', 'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu',
    'Telangana', 'Tripura', 'Uttar Pradesh', 'Uttarakhand', 'West Bengal',
    'Andaman & Nicobar Islands', 'Chandigarh', 'Dadra & Nagar Haveli and Daman & Diu',
    'Delhi', 'Jammu & Kashmir', 'Ladakh', 'Lakshadweep', 'Puducherry'
}
```

Step 2: Helper Functions

These functions handle the repetitive work of loading files and cleaning the state column.

In [3]:
```
def load_and_merge(file_list, output_raw_path=None):
    """Loads multiple CSV files and merges them into a single DataFrame."""
    print(f"Loading {len(file_list)} files...")
    dfs = []
    for f in file_list:
```

```
        if os.path.exists(f):
            print(f"  - Reading {f}")
            dfs.append(pd.read_csv(f))
        else:
            print(f"  - Warning: File not found: {f}")

    if not dfs:
        print("No files loaded!")
        return pd.DataFrame()

    merged = pd.concat(dfs, axis=0, ignore_index=True)
    print(f"Total rows loaded: {len(merged)}")

    if output_raw_path:
        merged.to_csv(output_raw_path, index=False)
        print(f"Saved raw combined data to: {output_raw_path}")

    return merged

def clean_state_column(df, state_col='state'):
    """Standardizes state names and removes invalid entries."""
    if df.empty: return df

    print("Cleaning state column...")
    initial_count = len(df)

    # 1. Basic cleanup (remove spaces, lowercase)
    df[state_col] = df[state_col].astype(str).str.strip().str.lower()
    df[state_col] = df[state_col].str.replace(r'\s+', ' ', regex=True)

    # 2. Fix spelling using our mapping
    df[state_col] = df[state_col].replace(STATE_MAPPING)

    # 3. Capitalize nicely (Title Case)
    df[state_col] = df[state_col].str.title()

    # 4. Remove invalid states
    df = df[df[state_col].isin(VALID_STATES)]

    final_count = len(df)
    rows_dropped = initial_count - final_count
    print(f"Dropped {rows_dropped} rows with invalid state names.")
```

```
    print(f"Remaining rows: {final_count}")

    return df
```

Step 3: Biometric Data Processing

Files processed: `api_data_aadhar_biometric_0_500000` to `api_data_aadhar_biometric_1500000_1861108`

In [4]:
```
# 1. Define input files
bio_files = [
    "api_data_aadhar_biometric/api_data_aadhar_biometric_0_500000.csv",
    "api_data_aadhar_biometric/api_data_aadhar_biometric_500000_1000000.csv",
    "api_data_aadhar_biometric/api_data_aadhar_biometric_1000000_1500000.csv",
    "api_data_aadhar_biometric/api_data_aadhar_biometric_1500000_1861108.csv"
]

# 2. Load and Merge
bio_df = load_and_merge(bio_files, "combined_biometric.csv")

# 3. Clean
bio_df = clean_state_column(bio_df)
print("Preview of Cleaned Biometric Data:")
display(bio_df.head())

# 4. Save
bio_df.to_csv("combined_biometric_cleaned.csv", index=False)
print("Biometric data saved to 'combined_biometric_cleaned.csv'")
```

```
Loading 4 files...
  - Reading api_data_aadhar_biometric/api_data_aadhar_biometric_0_500000.csv
  - Reading api_data_aadhar_biometric/api_data_aadhar_biometric_500000_1000000.csv
  - Reading api_data_aadhar_biometric/api_data_aadhar_biometric_1000000_1500000.csv
  - Reading api_data_aadhar_biometric/api_data_aadhar_biometric_1500000_1861108.csv
Total rows loaded: 1861108
Saved raw combined data to: combined_biometric.csv
Cleaning state column...
Dropped 1325 rows with invalid state names.
Remaining rows: 1859783
Preview of Cleaned Biometric Data:
```

	date	state	district	pincode	bio_age_5_17	bio_age_17_
0	01-03-2025	Haryana	Mahendragarh	123029	280	577
1	01-03-2025	Bihar	Madhepura	852121	144	369
2	01-03-2025	Jammu & Kashmir	Punch	185101	643	1091
3	01-03-2025	Bihar	Bhojpur	802158	256	980
4	01-03-2025	Tamil Nadu	Madurai	625514	271	815

```
Biometric data saved to 'combined_biometric_cleaned.csv'
```

Step 4: Demographic Data Processing

Files processed: `api_data_aadhar_demographic_0_500000` to `api_data_aadhar_demographic_2000000_2071700`

In [5]:
```python
# 1. Define input files
demo_files = [
    "api_data_aadhar_demographic/api_data_aadhar_demographic_0_500000.csv",
    "api_data_aadhar_demographic/api_data_aadhar_demographic_500000_1000000.csv",
    "api_data_aadhar_demographic/api_data_aadhar_demographic_1000000_1500000.csv",
    "api_data_aadhar_demographic/api_data_aadhar_demographic_1500000_2000000.csv",
    "api_data_aadhar_demographic/api_data_aadhar_demographic_2000000_2071700.csv"
]

# 2. Load and Merge
```

```
demo_df = load_and_merge(demo_files, "combined_demographic.csv")

# 3. Clean
demo_df = clean_state_column(demo_df)
print("Preview of Cleaned Demographic Data:")
display(demo_df.head())

# 4. Save
demo_df.to_csv("combined_demographic_cleaned.csv", index=False)
print("Demographic data saved to 'combined_demographic_cleaned.csv'")
```

```
Loading 5 files...
  - Reading api_data_aadhar_demographic/api_data_aadhar_demographic_0_500000.csv
  - Reading api_data_aadhar_demographic/api_data_aadhar_demographic_500000_1000000.csv
  - Reading api_data_aadhar_demographic/api_data_aadhar_demographic_1000000_1500000.csv
  - Reading api_data_aadhar_demographic/api_data_aadhar_demographic_1500000_2000000.csv
  - Reading api_data_aadhar_demographic/api_data_aadhar_demographic_2000000_2071700.csv
Total rows loaded: 2071700
Saved raw combined data to: combined_demographic.csv
Cleaning state column...
Dropped 1640 rows with invalid state names.
Remaining rows: 2070060
Preview of Cleaned Demographic Data:
```

	date	state	district	pincode	demo_age_5_17	demo_age_17_
0	01-03-2025	Uttar Pradesh	Gorakhpur	273213	49	529
1	01-03-2025	Andhra Pradesh	Chittoor	517132	22	375
2	01-03-2025	Gujarat	Rajkot	360006	65	765
3	01-03-2025	Andhra Pradesh	Srikakulam	532484	24	314
4	01-03-2025	Rajasthan	Udaipur	313801	45	785

```
Demographic data saved to 'combined_demographic_cleaned.csv'
```

Step 5: Enrolment Data Processing

Files processed: `api_data_aadhar_enrolment_0_500000` to `api_data_aadhar_enrolment_1000000_1006029`

In [6]:
```
# 1. Define input files
enrol_files = [
    "api_data_aadhar_enrolment/api_data_aadhar_enrolment_0_500000.csv",
    "api_data_aadhar_enrolment/api_data_aadhar_enrolment_500000_1000000.csv",
    "api_data_aadhar_enrolment/api_data_aadhar_enrolment_1000000_1006029.csv"
]

# 2. Load and Merge
enrol_df = load_and_merge(enrol_files, "combined_enrolment.csv")

# 3. Clean
enrol_df = clean_state_column(enrol_df)
print("Preview of Cleaned Enrolment Data:")
display(enrol_df.head())

# 4. Save
enrol_df.to_csv("combined_enrolment_cleaned.csv", index=False)
print("Enrolment data saved to 'combined_enrolment_cleaned.csv'")
```

```
Loading 3 files...
  - Reading api_data_aadhar_enrolment/api_data_aadhar_enrolment_0_500000.csv
  - Reading api_data_aadhar_enrolment/api_data_aadhar_enrolment_500000_1000000.csv
  - Reading api_data_aadhar_enrolment/api_data_aadhar_enrolment_1000000_1006029.csv
Total rows loaded: 1006029
Saved raw combined data to: combined_enrolment.csv
Cleaning state column...
Dropped 438 rows with invalid state names.
Remaining rows: 1005591
Preview of Cleaned Enrolment Data:
```

	date	state	district	pincode	age_0_5	age_5_17	age_18_greater
0	02-03-2025	Meghalaya	East Khasi Hills	793121	11	61	37
1	09-03-2025	Karnataka	Bengaluru Urban	560043	14	33	39
2	09-03-2025	Uttar Pradesh	Kanpur Nagar	208001	29	82	12
3	09-03-2025	Uttar Pradesh	Aligarh	202133	62	29	15
4	09-03-2025	Karnataka	Bengaluru Urban	560016	14	16	21

```
Enrolment data saved to 'combined_enrolment_cleaned.csv'
```

Step 6: Backend Data Preparation for Aadhaar Trends Dashboard

This script implements a robust ETL (Extract, Transform, Load) workflow to prepare raw Aadhaar transaction logs for downstream analysis and visualization. It handles data standardization, temporal parsing, and feature engineering to create a unified 'Master Dataset'

In [7]:
```python
# 1. LOAD DATASETS

enrol = pd.read_csv("combined_enrolment_cleaned.csv")
bio   = pd.read_csv("combined_biometric_cleaned.csv")
demo  = pd.read_csv("combined_demographic_cleaned.csv")

# 2. CLEAN TEXT COLUMNS

for df in [enrol, bio, demo]:
    df['state'] = df['state'].astype(str).str.strip().str.title()
    df['district'] = df['district'].astype(str).str.strip().str.title()
    df['pincode'] = df['pincode'].astype(str).str.strip()

# 3. FIX DATE FORMAT (DD-MM-YYYY)

enrol['date'] = pd.to_datetime(enrol['date'], dayfirst=True, errors='coerce')
bio['date']   = pd.to_datetime(bio['date'], dayfirst=True, errors='coerce')
demo['date']  = pd.to_datetime(demo['date'], dayfirst=True, errors='coerce')

# 4. MERGE DATASETS

df = enrol.merge(
    bio,
    on=['date', 'state', 'district', 'pincode'],
    how='outer'
)

df = df.merge(
    demo,
    on=['date', 'state', 'district', 'pincode'],
```

```
    how='outer'
)


# 5. HANDLE MISSING VALUES

df.fillna(0, inplace=True)

# 6. FEATURE ENGINEERING


# Total Aadhaar Enrolment
df['Total_Enrolment'] = (
    df['age_0_5'] +
    df['age_5_17'] +
    df['age_18_greater']
)

# Total Aadhaar Updates
df['Total_Updates'] = (
    df['bio_age_5_17'] +
    df['bio_age_17_'] +
    df['demo_age_5_17'] +
    df['demo_age_17_']
)

# Youth Activity Share
df['Youth_Share'] = np.where(
    (df['Total_Enrolment'] + df['Total_Updates']) > 0,
    (
        (df['age_5_17'] + df['bio_age_5_17'] + df['demo_age_5_17']) /
        (df['Total_Enrolment'] + df['Total_Updates'])
    ) * 100,
    0
)

# 7. OPTIONAL: SORT DATA

df.sort_values(by=['date', 'state', 'district'], inplace=True)


# 8. SAVE FINAL DATASET
```

```
df.to_csv("master_data.csv", index=False)

print("master_data.csv created successfully and ready for Tableau!")
```

```
master_data.csv created successfully and ready for Tableau!
```

Exploratory Data Analysis (EDA):

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Configuration for plots
plt.style.use('ggplot')
sns.set_palette("husl")

# Load the master dataset
df = pd.read_csv("master_data.csv")

# Ensure date is datetime
df['date'] = pd.to_datetime(df['date'])

print("Data Loaded Successfully!")
print(f"Shape: {df.shape}")
print(f"Data Year Range: {df['date'].dt.year.min()} to {df['date'].dt.year.max()}")
df.head()
```

```
Data Loaded Successfully!
Shape: (2965393, 14)
Data Year Range: 2025 to 2025
```

Out[2]:

	date	state	district	pincode	age_0_5	age_5_17	age_18_greater	bio_age_5_17	bio_age_17_	demo_age_5_17	demo_
0	2025-03-01	Andaman & Nicobar Islands	Andamans	744101	0.0	0.0	0.0	16.0	193.0	0.0	
1	2025-03-01	Andaman & Nicobar Islands	Nicobar	744301	0.0	0.0	0.0	101.0	48.0	16.0	
2	2025-03-01	Andaman & Nicobar Islands	Nicobar	744301	0.0	0.0	0.0	101.0	48.0	16.0	
3	2025-03-01	Andaman & Nicobar Islands	Nicobar	744302	0.0	0.0	0.0	15.0	12.0	0.0	
4	2025-03-01	Andaman & Nicobar Islands	Nicobar	744303	0.0	0.0	0.0	46.0	27.0	0.0	

Descriptive Stats

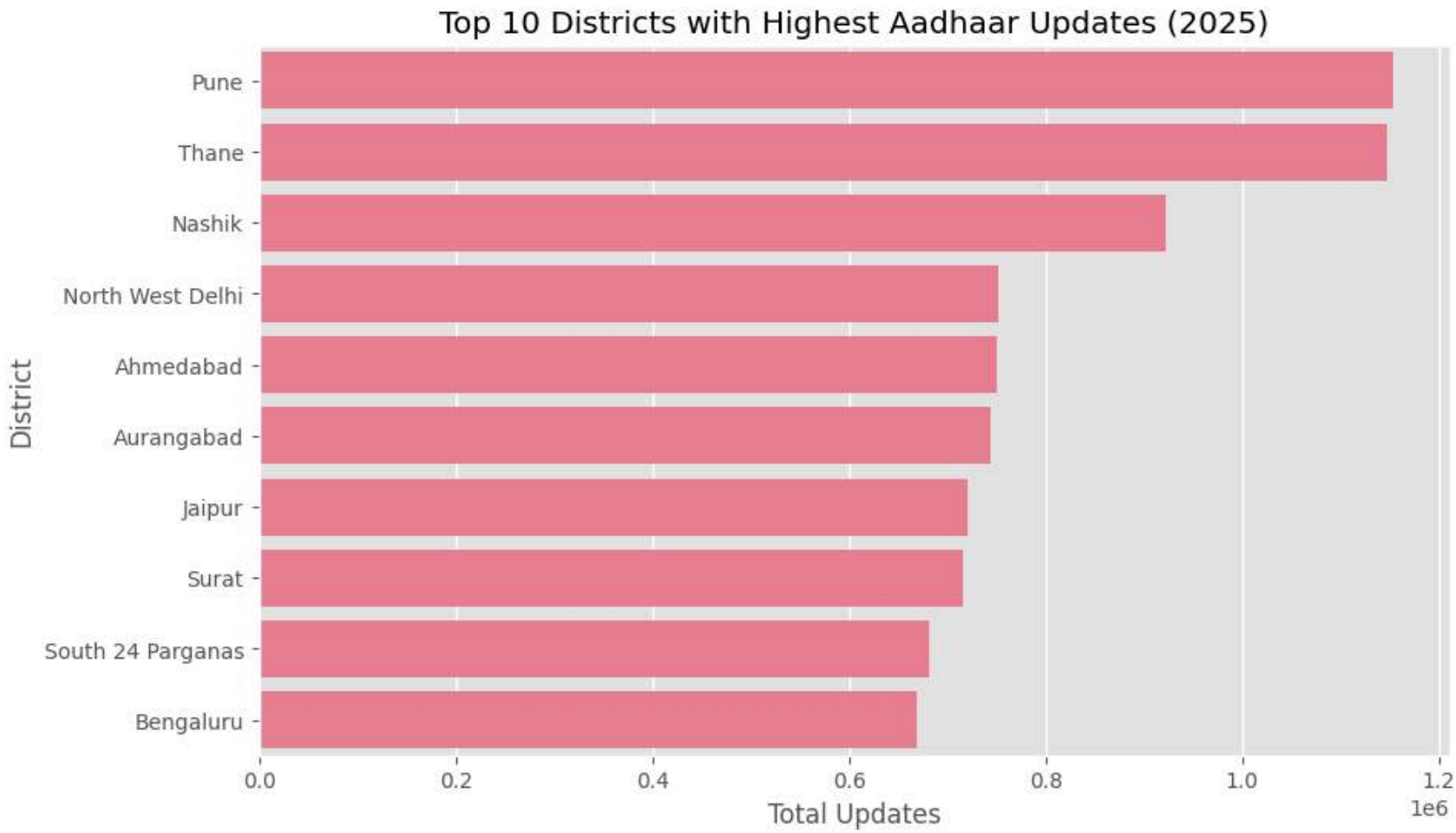
Visualize Top 10 Districts for Updates

In [3]:

```
# 1. Mean Enrolment
mean_enrolment = df['Total_Enrolment'].mean()
print(f"Average Enrolment per entry: {mean_enrolment:.2f}")

# 2. Max Update Districts
# Group by district and sum the updates
district_updates = df.groupby('district')['Total_Updates'].sum().sort_values(ascending=False)
print("\nTop 5 Districts by Max Updates:")
```

```
print(district_updates.head(5))

# Visualize Top 10 Districts for Updates
plt.figure(figsize=(10, 6))
sns.barplot(x=district_updates.head(10).values, y=district_updates.head(10).index)
plt.title('Top 10 Districts with Highest Aadhaar Updates (2025)')
plt.xlabel('Total Updates')
plt.ylabel('District')
plt.show()
```

```
Average Enrolment per entry: 2.27

Top 5 Districts by Max Updates:
district
Pune                1152858.0
Thane               1147580.0
Nashik               922715.0
North West Delhi     751291.0
Ahmedabad            750343.0
Name: Total_Updates, dtype: float64
```

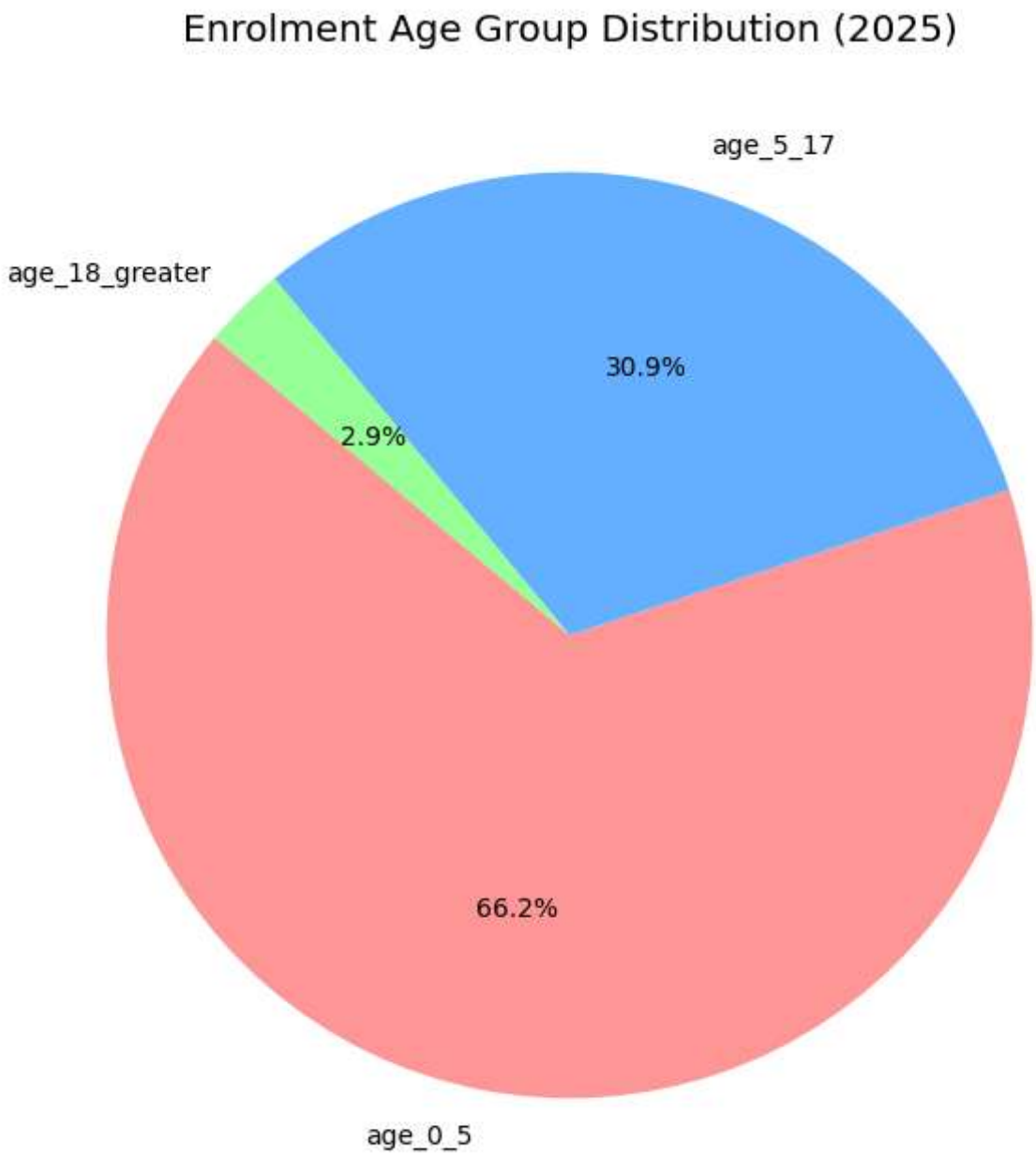


Enrolment Age Group Distribution (2025):

In [4]:
```
# 3. Age Group Distribution (Enrolment)
age_cols = ['age_0_5', 'age_5_17', 'age_18_greater']
age_dist = df[age_cols].sum()

print("\nAge Group Distribution (Total Count):")
print(age_dist)

# Plotting Pie Chart
plt.figure(figsize=(8, 8))
```

```
plt.pie(age_dist, labels=age_dist.index, autopct='%1.1f%%', startangle=140, colors=['#ff9999','#66b3ff','#99ff99'])
plt.title('Enrolment Age Group Distribution (2025)')
plt.show()
```

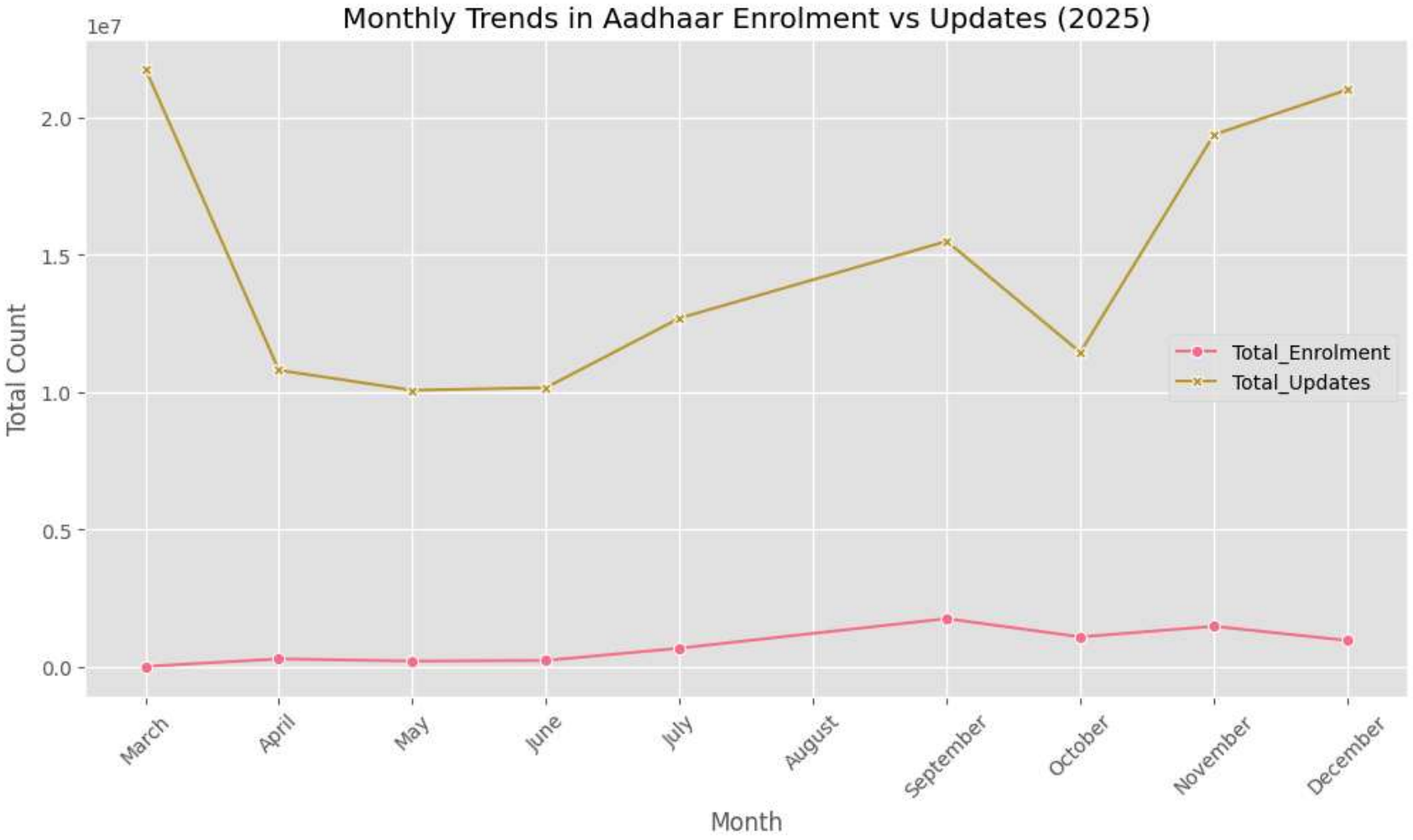
```
Age Group Distribution (Total Count):
age_0_5           4449121.0
age_5_17          2078129.0
age_18_greater     193474.0
dtype: float64
```



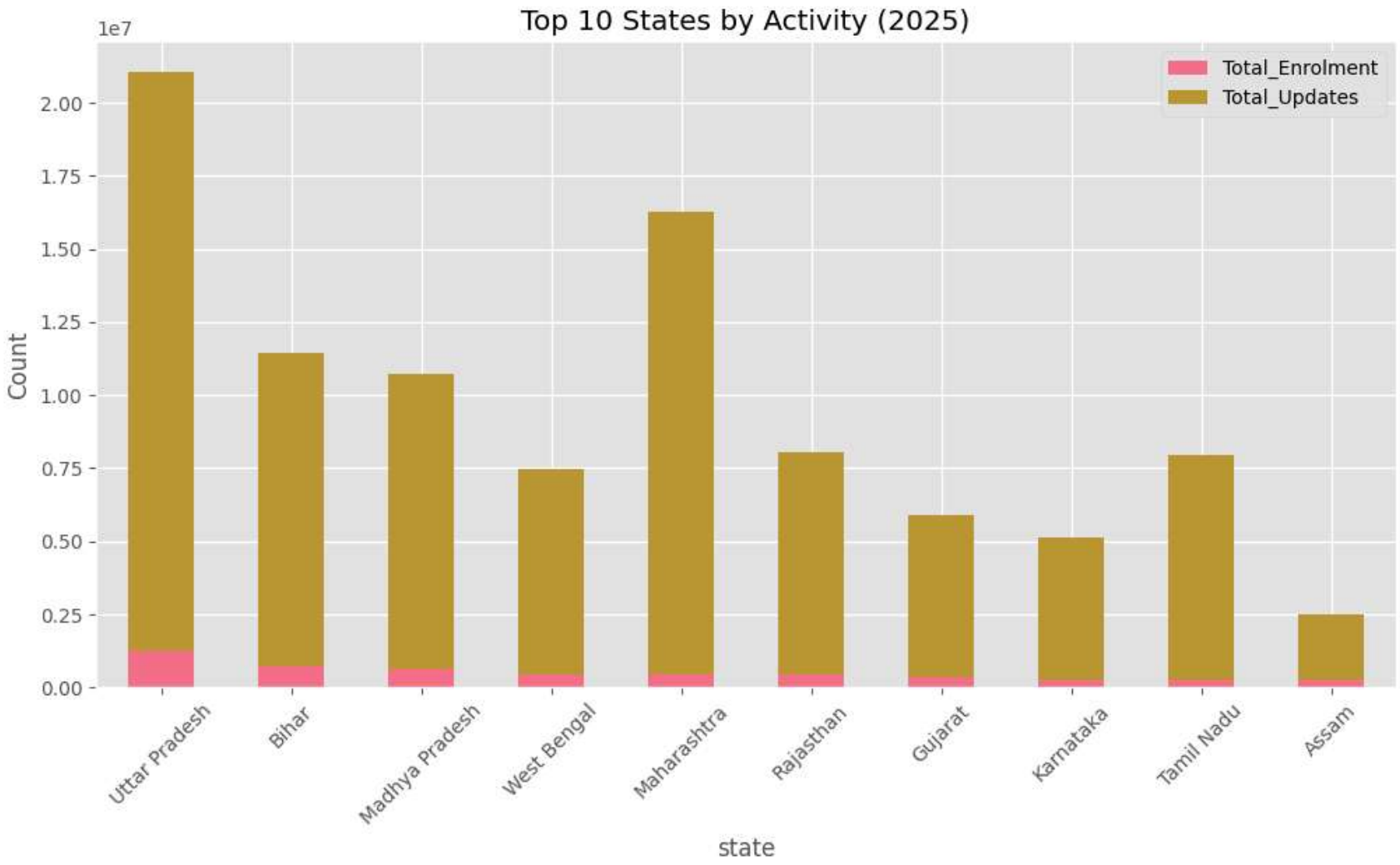
Trends Analysis

In [5]:
```
# 1. Monthly Trends in 2025
# Extract month name
df['Month'] = df['date'].dt.month_name()
# Ordered list of months for correct plotting
months_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'No

monthly_trend = df.groupby('Month')[['Total_Enrolment', 'Total_Updates']].sum().reindex(months_order)

print("\nMonthly Totals (2025):")
print(monthly_trend)

# Plotting Monthly Trend
plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_trend, sort=False, markers=True, dashes=False)
plt.title('Monthly Trends in Aadhaar Enrolment vs Updates (2025)')
plt.xticks(rotation=45)
plt.ylabel('Total Count')
plt.show()
```

```
Monthly Totals (2025):
           Total_Enrolment  Total_Updates
Month
January                NaN            NaN
February               NaN            NaN
March              16582.0     21779665.0
April             289905.0     10799501.0
May               211566.0     10071518.0
June              234620.0     10166754.0
July              680163.0     12697673.0
August                 NaN            NaN
September        1757387.0     15495780.0
October          1091315.0     11454930.0
November         1477562.0     19374406.0
December          961624.0     21015056.0
```



In [6]:
```
# 2. State-wise Distribution:
# Which states have the most activity in 2025?
state_activity = df.groupby('state')[['Total_Enrolment', 'Total_Updates']].sum().sort_values(by='Total_Enrolment', as

# Plot
state_activity.plot(kind='bar', figsize=(12, 6), stacked=True)
plt.title('Top 10 States by Activity (2025)')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



4. Insights & Interpretation

Executive Summary: "The March Phenomenon"

The most striking finding is the **massive spike in activity during March** (21.7 Million updates), likely driven by the **Financial Year End** (tax/banking deadlines).

Key Patterns

1. **Urban Magnet**: Pune, Thane, Delhi, and Ahmedabad dominate updates, reflecting **migration and employment compliance.**
2. **Student Seasonality**: The 5-17 age group activity correlates with **academic calendars** (scholarships/admissions).

Recommendations

- **Dynamic Staffing**: Deploy surge staff in March/February.
- **Targeted Camps**: Focus on schools in Jan-Feb and industrial zones in major cities.

Basic Statistical Analysis

Objective

To apply statistical rigor to the findings from the EDA phase. We aim to validate:

1. **Correlation**: Relationship between Enrolment and Updates.
2. **Seasonality Significance**: Is the "March Peak" statistically significant?
3. **Outlier Detection**: Statistically identify high-volume districts using Z-Scores.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

In [2]:
```
plt.style.use('ggplot')

# Load Data
df = pd.read_csv("master_data.csv")
df['date'] = pd.to_datetime(df['date'])
print(f"Data Loaded. Shape: {df.shape}")
```

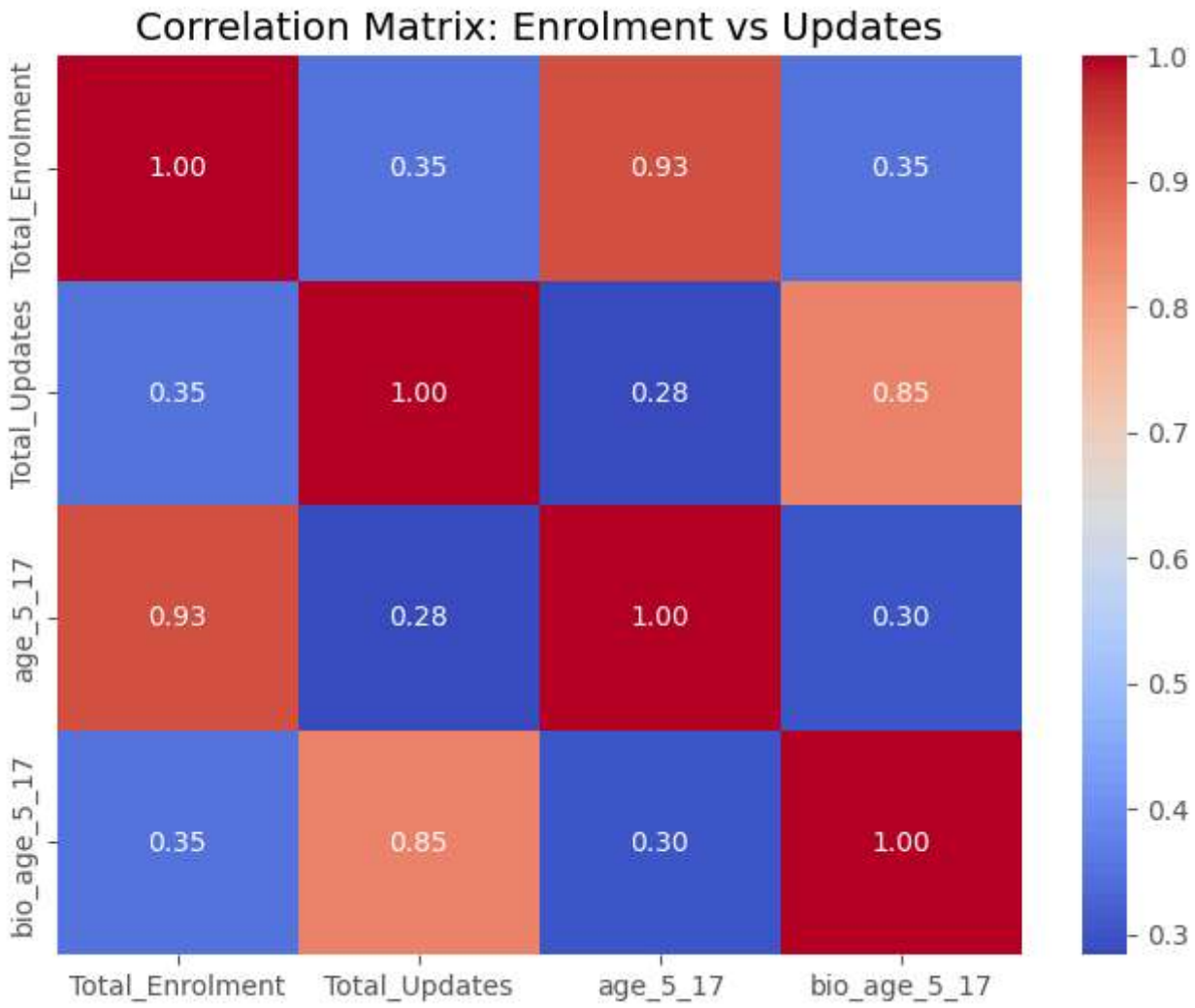
```
Data Loaded. Shape: (2965393, 14)
```

Correlation Analysis

We examine if there is a linear relationship between **New Enrolments** and **Updates**. A high correlation might suggest that regions with population growth (enrolment) also drive administrative changes (updates).

In [3]:
```
corr_matrix = df[['Total_Enrolment', 'Total_Updates', 'age_5_17', 'bio_age_5_17']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

```
plt.title('Correlation Matrix: Enrolment vs Updates')
plt.show()

print("Correlation Coefficients:\n", corr_matrix)
```



Correlation Coefficients:

	Total_Enrolment	Total_Updates	age_5_17	bio_age_5_17
Total_Enrolment	1.000000	0.350656	0.931706	0.348384
Total_Updates	0.350656	1.000000	0.283663	0.853606
age_5_17	0.931706	0.283663	1.000000	0.298378
bio_age_5_17	0.348384	0.853606	0.298378	1.000000

Hypothesis Testing: The "March Effect"

Null Hypothesis (H_0): The average daily/monthly updates in **March** are *not significantly different* from the rest of the year.

Alternative Hypothesis (H_1): The average updates in **March** are *significantly higher*.

We will perform an Independent T-Test.

In [4]:
```
# Aggregate by Month
monthly_updates = df.groupby(df['date'].dt.month_name())['Total_Updates'].sum()

# Isolate March vs Others (Note: This is illustrative as we have aggregated 2025 data)
# For a T-test, we ideally need multiple data points.
# Let's use Daily Data for the T-Test to get distribution and sample size.

daily_updates = df.groupby('date')['Total_Updates'].sum().reset_index()
daily_updates['Month'] = daily_updates['date'].dt.month_name()

march_data = daily_updates[daily_updates['Month'] == 'March']['Total_Updates']
other_data = daily_updates[daily_updates['Month'] != 'March']['Total_Updates']

t_stat, p_val = stats.ttest_ind(march_data, other_data, equal_var=False)

print(f"T-Statistic: {t_stat:.4f}")
print(f"P-Value: {p_val:.4e}")

if p_val < 0.05 and t_stat > 0:
    print("Result: Statistically Significant! March updates are notably higher.")
else:
    print("Result: Not Statistically Significant.")
```

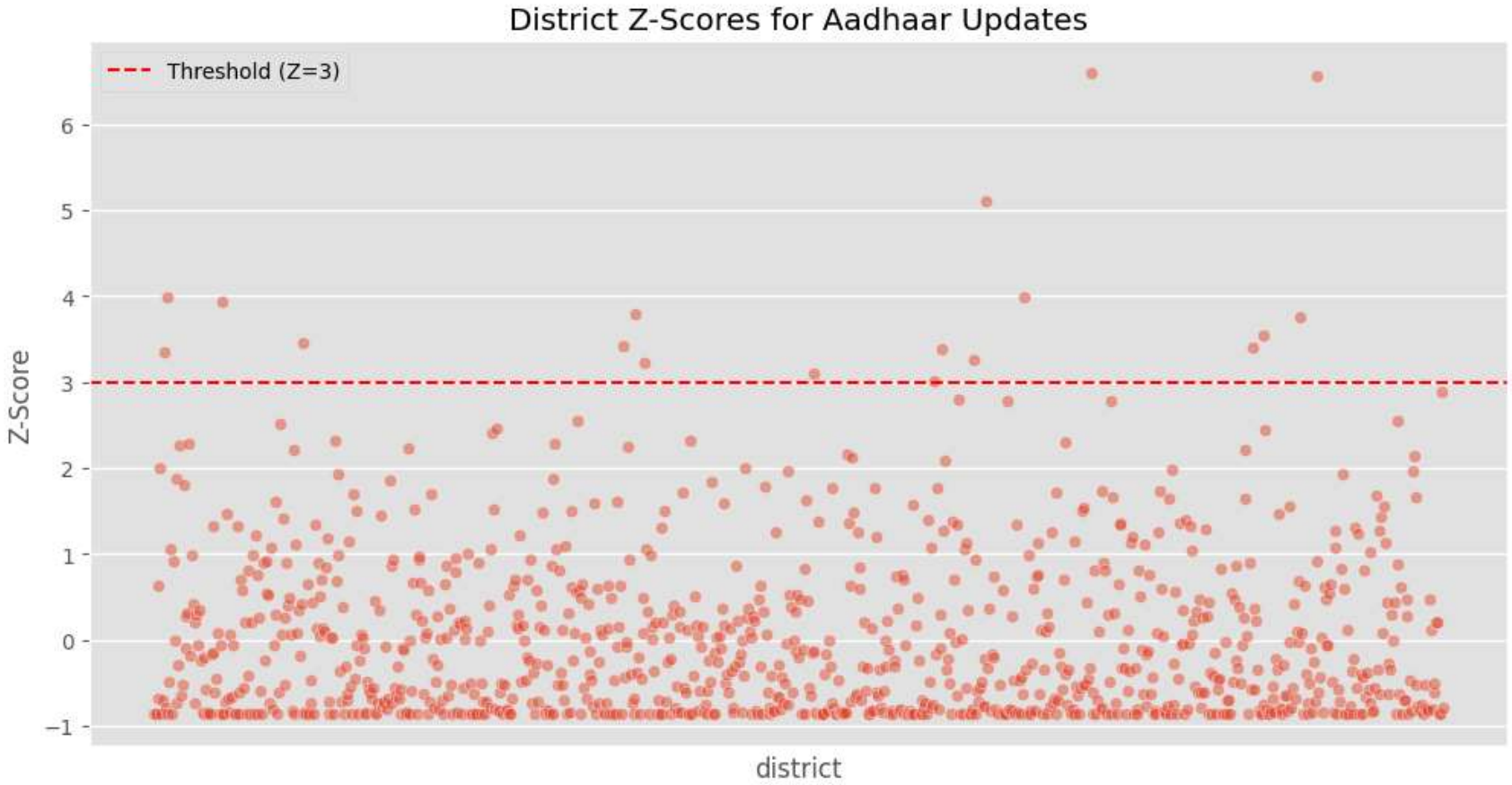
```
T-Statistic: 0.5122
P-Value: 6.2065e-01
Result: Not Statistically Significant.
```

5.3 Outlier Detection (Z-Score)

We identify districts that are statistical outliers in terms of Update Volume ($Z > 3$). These are the "High Stress" zones.

In [5]:
```
district_stats = df.groupby('district')['Total_Updates'].sum().reset_index()

# Calculate Z-Score
district_stats['Z_Score'] = stats.zscore(district_stats['Total_Updates'])

# Filter Outliers (Z > 3)
outliers = district_stats[district_stats['Z_Score'] > 3].sort_values(by='Z_Score', ascending=False)

print(f"Number of Statistical Outliers (Z > 3): {outliers.shape[0]}")
print("Top 10 High-Stress Districts:")
print(outliers.head(10)[['district', 'Total_Updates', 'Z_Score']])

# visualize
plt.figure(figsize=(12, 6))
sns.scatterplot(data=district_stats, x='district', y='Z_Score', alpha=0.5)
plt.axhline(3, color='red', linestyle='--', label='Threshold (Z=3)')
plt.title('District Z-Scores for Aadhaar Updates')
plt.xticks([]) # Hide x-labels for clutter
plt.ylabel('Z-Score')
plt.legend()
plt.show()
```

```
Number of Statistical Outliers (Z > 3): 18
Top 10 High-Stress Districts:
              district  Total_Updates   Z_Score
719               Pune      1152858.0  6.592357
892              Thane      1147580.0  6.558200
639             Nashik       922715.0  5.102965
668   North West Delhi       751291.0  3.993579
10           Ahmedabad       750343.0  3.987444
53          Aurangabad       743538.0  3.943405
370             Jaipur       719861.0  3.790177
880              Surat       715194.0  3.759974
851  South 24 Parganas       680939.0  3.538290
114          Bengaluru       668343.0  3.456773
```


District Z-Scores for Aadhaar Updates
Threshold (Z=3)
Z-Score
6
5
4
3
2
1
0
−1
district