# Aadhaar Operational Efficiency Engine (AOEE)

Aadhaar Inclusion & Service Intelligence System (AISIS)

**Document Version:** 1.0 | **Date:** 2026-01-20

## 1. Problem Statement and Approach

### Problem Statement: Predictive Targeting for Inclusive Aadhaar Coverage

The current operational model for Aadhaar service delivery is predominantly reactive, addressing issues as they arise. This approach has inadvertently created significant challenges, including the emergence of **"service deserts"**—geographic areas with inadequate enrolment and update facilities. Furthermore, systemic issues lead to observable gender-based disparities in update rates and the marginalization of vulnerable populations, such as children and the elderly. These groups are often excluded due to logistical barriers, hardware malfunctions, or operational inefficiencies, which compromises the universal accessibility promise of Aadhaar and its linked welfare benefits.

### Analytical Approach

To address these shortcomings, we propose a paradigm shift from descriptive reporting to **predictive and prescriptive analytics**. This is powered by the Aadhaar Operational Efficiency Engine (AOEE), a high-performance data pipeline designed to process vast datasets in near real-time. Our analytical core leverages two primary techniques:

- **Demand Forecasting:** A Random Forest Regression model is employed to predict future demand for enrolment and update services at a granular, district level. This allows for proactive resource allocation.
- **Operational Anomaly Detection:** A 2-Sigma Statistical Process Control method is used to monitor key performance indicators (KPIs). This allows for the immediate flagging of operational inefficiencies, such as underperforming hardware or operator-related issues, enabling rapid intervention.

### Technical Approach

The AOEE is built on a modular framework, ensuring scalability and maintainability. Each module targets a specific aspect of operational efficiency and social inclusion:

- **Demand & Crowd Prediction:** Forecasts service demand surges, enabling the strategic deployment of mobile enrolment vans to high-need areas before backlogs accumulate.
- **Gender Update Parity (GUP) Module:** Actively monitors and analyzes the ratio of female-to-male biometric updates, detecting and flagging districts where a significant gap exists, thereby prompting targeted outreach campaigns.
- **Authentication Success Intelligence:** Pinpoints geographic hotspots (down to the pincode level) with abnormally high authentication rejection rates, isolating potential hardware degradation or the need for operator retraining.

---

## 2. Datasets Used

---

The analysis is founded upon three core, anonymized datasets provided by UIDAI, which offer a comprehensive view of the Aadhaar ecosystem. The temporal scope of our analysis focuses on data from Month 3 (March 2024) onwards, consistent with the availability in the source files.

- **Enrolment Data:** This dataset provides records of new Aadhaar enrolments. It is fundamental for calculating the total volume of activity in a region, identifying enrolment gaps by comparing against census data, and tracking the influx of new residents into the system. Analysis of this data, as seen in our monthly volume charts, helps identify seasonal trends and the impact of enrolment drives.
- **Demographic Update Data:** This dataset contains records of changes to non-biometric information, such as address, mobile number, and gender. It is a critical indicator of population mobility and is used to track update trends, particularly among different age groups. This data feeds into our Vulnerability Index by highlighting update frequencies for the elderly.
- **Biometric Update Data:** This dataset is crucial for two key analytical modules. Firstly, it is used to analyze the backlog of **Mandatory Biometric Updates (MBU)** for children in the 5-17 age bracket, a legal requirement to ensure the integrity of their biometric data. Secondly, it provides the raw data for calculating authentication success and failure rates, which is essential for our Hardware Hotspots analysis.

---

# 3. Methodology

Our architecture implements a rigorous five-stage data science pipeline, transforming raw, disparate data into actionable intelligence. Each stage is encapsulated in a dedicated Python script, ensuring modularity and reproducibility.

## Stage 1: Data Cleaning and Unification (`clean_aadhar_data.py`)

The initial stage focuses on creating a pristine, unified dataset. Raw data from various sources often contains inconsistencies. Our process includes standardizing state and district names to a canonical format, merging data from former Union Territories (e.g., Daman & Diu with Dadra and Nagar Haveli) for consistent longitudinal analysis, and imputing or flagging missing values. A key output of this stage is the creation of a

composite **`Total_Activity`** metric, which aggregates enrolments, demographic updates, and biometric updates to provide a holistic measure of operational volume.

## Stage 2: Advanced Feature Engineering (`process_aoee_data.py`)

This stage enriches the cleaned data with powerful, context-aware features that are critical for predictive modeling and intelligence gathering:

- **Gender Update Parity (GUP) Index:** Calculated as the ratio of female biometric updates to male biometric updates within a district. A value significantly below 1.0 indicates potential gender-based barriers to access, flagging the area for investigation.
- **Service Desert Score:** This score is derived by comparing the population of a district (from census data) to its `Total_Activity` volume. A high population-to-activity ratio signifies an underserved area, or a "service desert."
- **Vulnerability Index:** A weighted score that combines factors such as the MBU backlog for children (age 5-17), low update frequency among the elderly (age 70+), and geographic remoteness. This index helps prioritize resource allocation to the most vulnerable populations.

## Stage 3: Predictive Modeling (`train_aoee_model.py`)

At the core of our predictive capability is a **Random Forest Regressor**. This model is trained on over 14 engineered features—including the GUP Index, Service Desert Score, historical activity volumes, and seasonality indicators—to predict future enrolment and update gaps at the district level. The ensemble nature of the Random Forest makes it robust to outliers and effective at capturing complex, non-linear relationships in the data, providing reliable forecasts for operational planning.

## Stage 4: Operational Intelligence (`analysis_aoee.py`)

This stage moves from prediction to real-time monitoring. We apply Statistical Process Control (SPC) principles, specifically using **2-Sigma control limits** (Mean ± 2 × Standard Deviation) on monthly activity data for each district. Any district whose activity falls below the lower control limit is automatically flagged as an anomaly. This indicates a statistically

significant drop in performance, suggesting systemic failures such as widespread hardware outages, software bugs, or administrative bottlenecks, requiring immediate attention.

---

# 4. Data Analysis and Visualisation

Our analysis translates complex datasets into clear, interpretable visualizations that expose underlying trends, anomalies, and areas for intervention.

### 4.1 Univariate Analysis: Identifying Update Backlogs

**Insight:** A foundational step in our analysis was to understand the distribution of service backlogs. The histogram below reveals a significant, right-skewed distribution in the mandatory biometric update (MBU) backlog for the 5-17 age group. The vast majority of districts have a low volume of pending updates, but a long tail indicates a subset of districts with a critical backlog. This distribution helps us quantify and locate **"Update Deserts"** where the system is failing to keep pace with the needs of a growing youth population. A similar pattern is observed for demographic updates among the elderly, highlighting another vulnerable cohort.

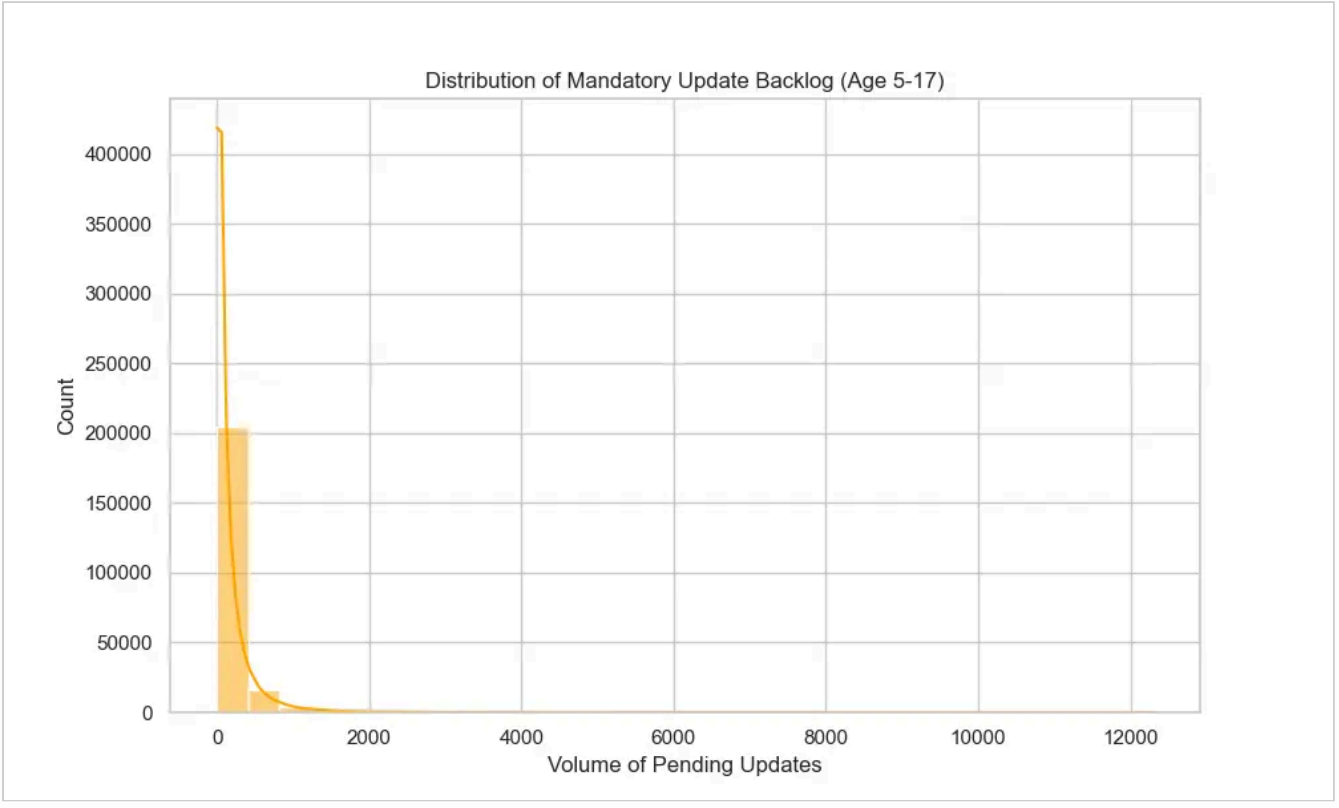

*Figure 4.1.1: Distribution of Mandatory Update Backlog (Age 5-17). The high concentration near zero with a long tail identifies districts with severe backlogs.*

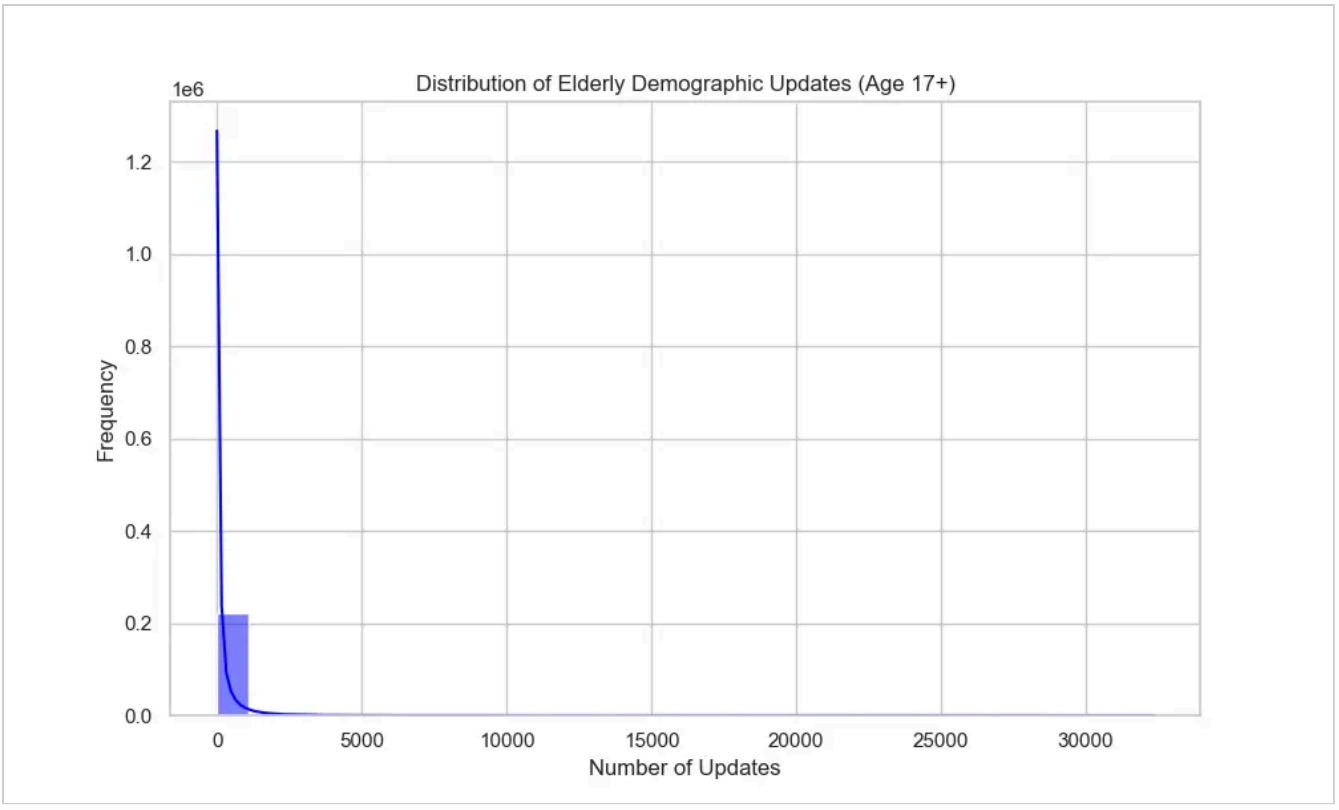


*Figure 4.1.2: Distribution of Elderly Demographic Updates (Age 17+). A similar skewed pattern indicates that while most elderly residents have few updates, a segment requires more frequent service.*

## 4.2 Bivariate Analysis: Pinpointing Hardware Hotspots and Regional Needs

**Insight:** By correlating geographic location (Pincode) with performance metrics, we can uncover localized issues. The scatter plot below maps authentication failure rates against pincodes. This analysis reveals **"Hardware Hotspots"**—clusters of pincodes where failure rates are consistently elevated (ranging from 2% to over 5%), significantly higher than the norm. These hotspots strongly suggest issues with local infrastructure, such as faulty biometric sensors, poor network connectivity, or a need for operator retraining. A second plot correlates pincodes with the volume of biometric updates for children, directly visualizing the geographic distribution of MBU demand.

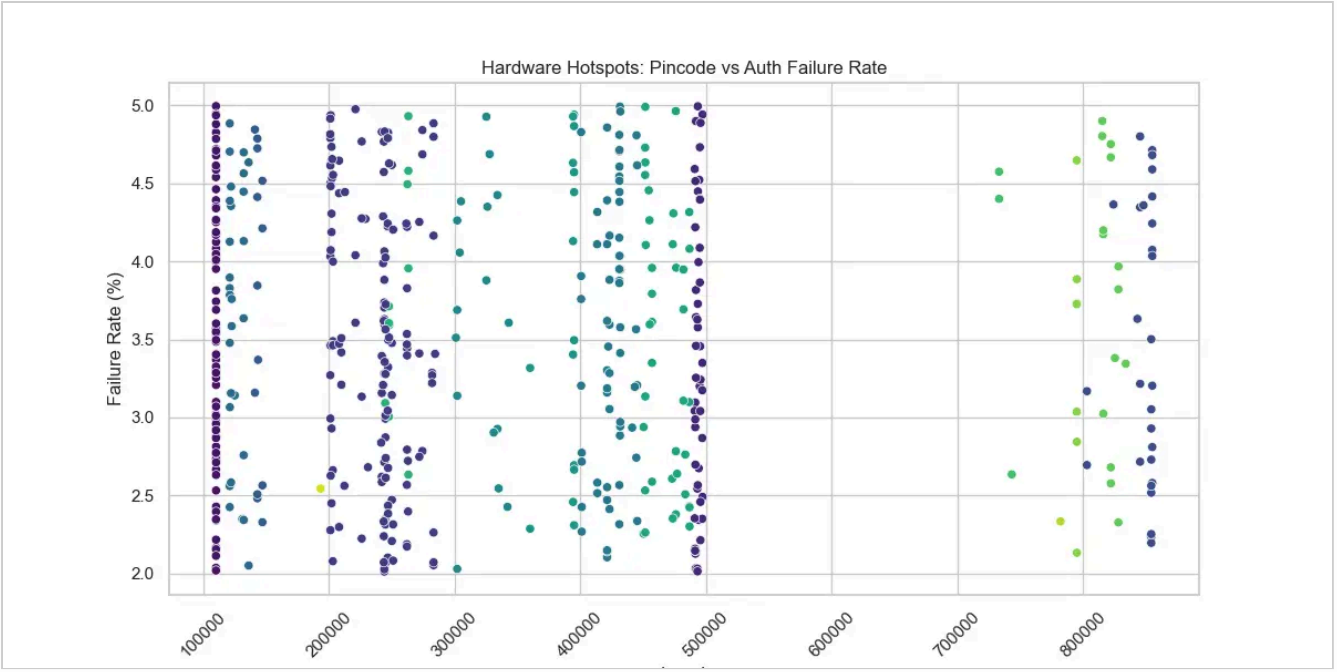


*Figure 4.2.1: Hardware Hotspots - Pincode vs. Auth Failure Rate. Clusters of high failure rates point to localized technical or operational issues.*

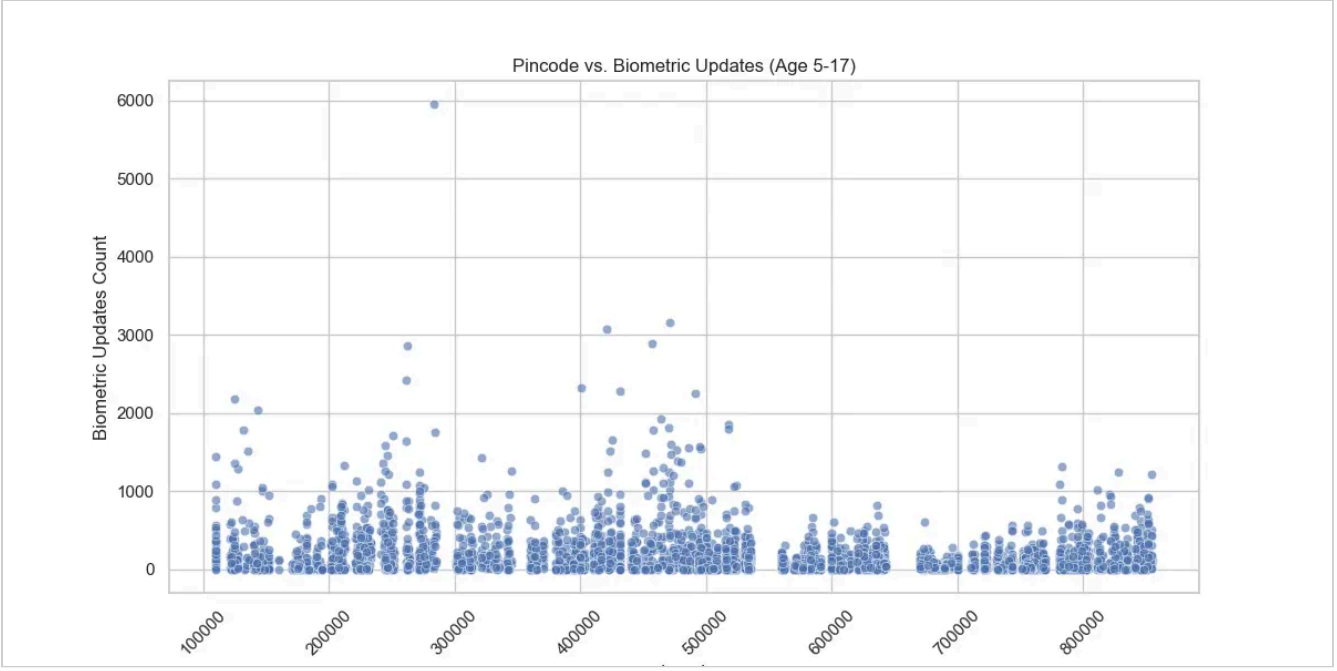


*Figure 4.2.2: Pincode vs. Biometric Updates (Age 5-17). This plot shows the geographic demand for mandatory updates, helping to plan resource allocation.*

## 4.3 Trivariate Analysis: The Operational Health Map

**Insight:** To provide a command-center view of system health, we developed a trivariate heatmap visualizing State × Month × KPI. The first heatmap shows the **Authentication Success Rate**. It allows officials to see at a glance that while most states maintain a high

success rate (above 96.5%), specific regions like Dadra and Nagar Haveli experienced dips in months 3 and 5. The second heatmap shows **Total Activity Volume**, revealing massive seasonal and regional variations. For example, Uttar Pradesh and Maharashtra consistently show the highest volumes, while states like Bihar experience a sharp drop after January. Together, these maps enable proactive resource shifts to prevent systemic "choke points" and investigate regions with declining performance.

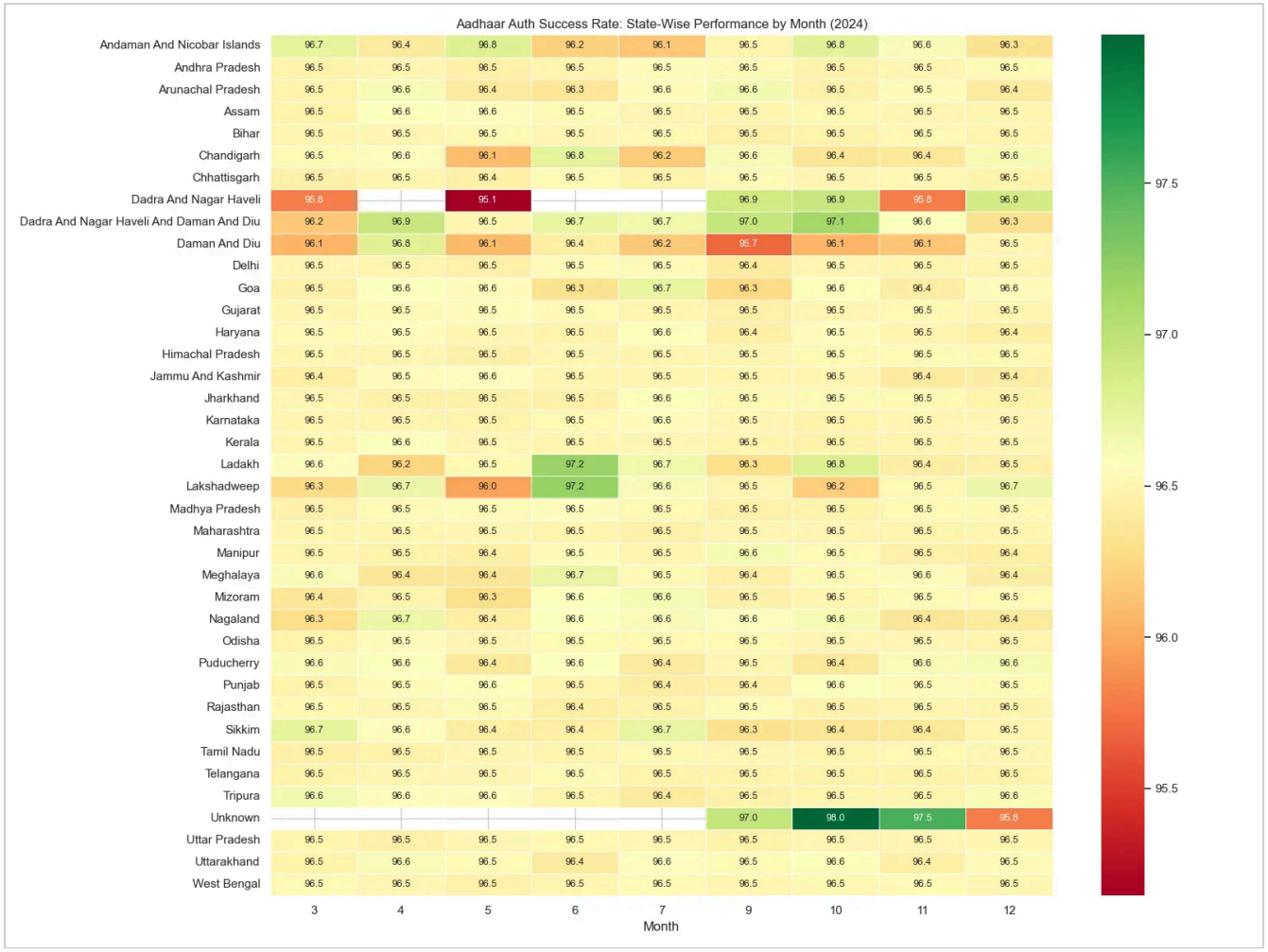


*Figure 4.3.1: Aadhaar Authentication Success Rate by State and Month (2024). Darker green indicates higher success, allowing for quick identification of underperforming regions.*

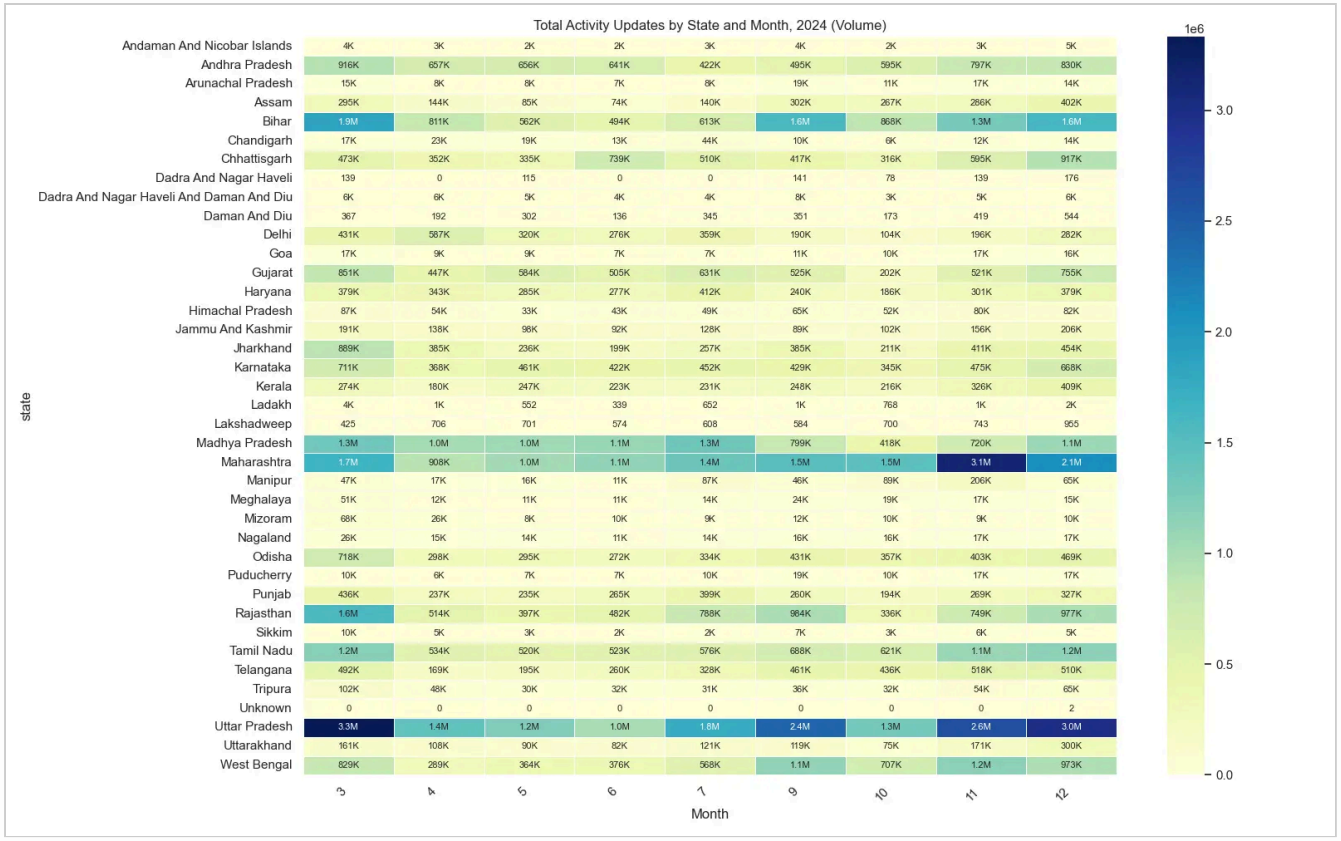


*Figure 4.3.2: Total Activity Updates by State and Month (2024). This map highlights the immense disparity in operational load across India.*

## 4.4 Anomaly Detection: Monitoring System Stability

**Insight:** The AOEE's anomaly detection module uses a 2-Sigma baseline to flag statistically significant deviations from the norm. The first chart below shows this principle in action for total updates, where activity remained within the control limits but showed a significant drop below the mean from month 4 to 7. The second chart breaks this down, showing the monthly trends for different activity types. The sharp drop in "Demographic Updates" in month 4 was the primary driver of the overall dip. By flagging such deviations, the engine alerts administrators to potential issues like software bugs, policy changes causing confusion, or regional administrative hurdles that require investigation.


*Figure 4.4.1: Operational Stability - 2-Sigma Anomaly Detection. This chart establishes the statistical baseline for identifying abnormal performance.*

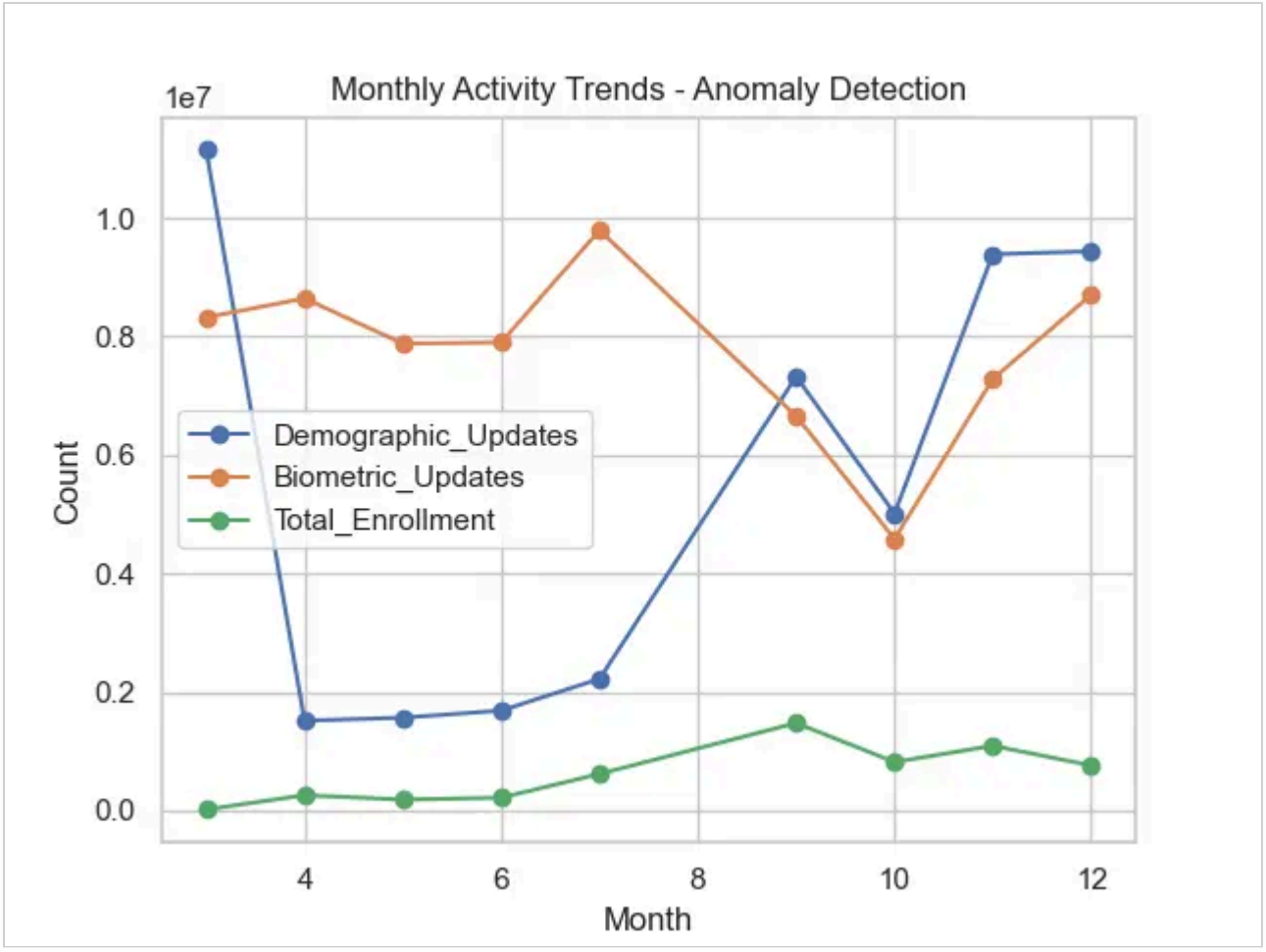

*Figure 4.4.2: Monthly Activity Trends. Decomposing total activity reveals which specific service is driving an anomaly.*

### 4.5 Infographic: The Enrollment Success Funnel

**Insight:** This chart visualizes the final output of the Aadhaar ecosystem: successful new enrolments. It functions as a "success funnel," showing the monthly volume of individuals who have successfully navigated the entire process. The significant fluctuations, with a low in month 3 and a peak of nearly 1.45 million in month 9, reflect the combined effects of seasonal demand (e.g., school admissions), targeted enrolment drives, and overall system capacity. Analyzing this attrition from potential coverage to actual enrolment helps measure the effectiveness of strategic initiatives.

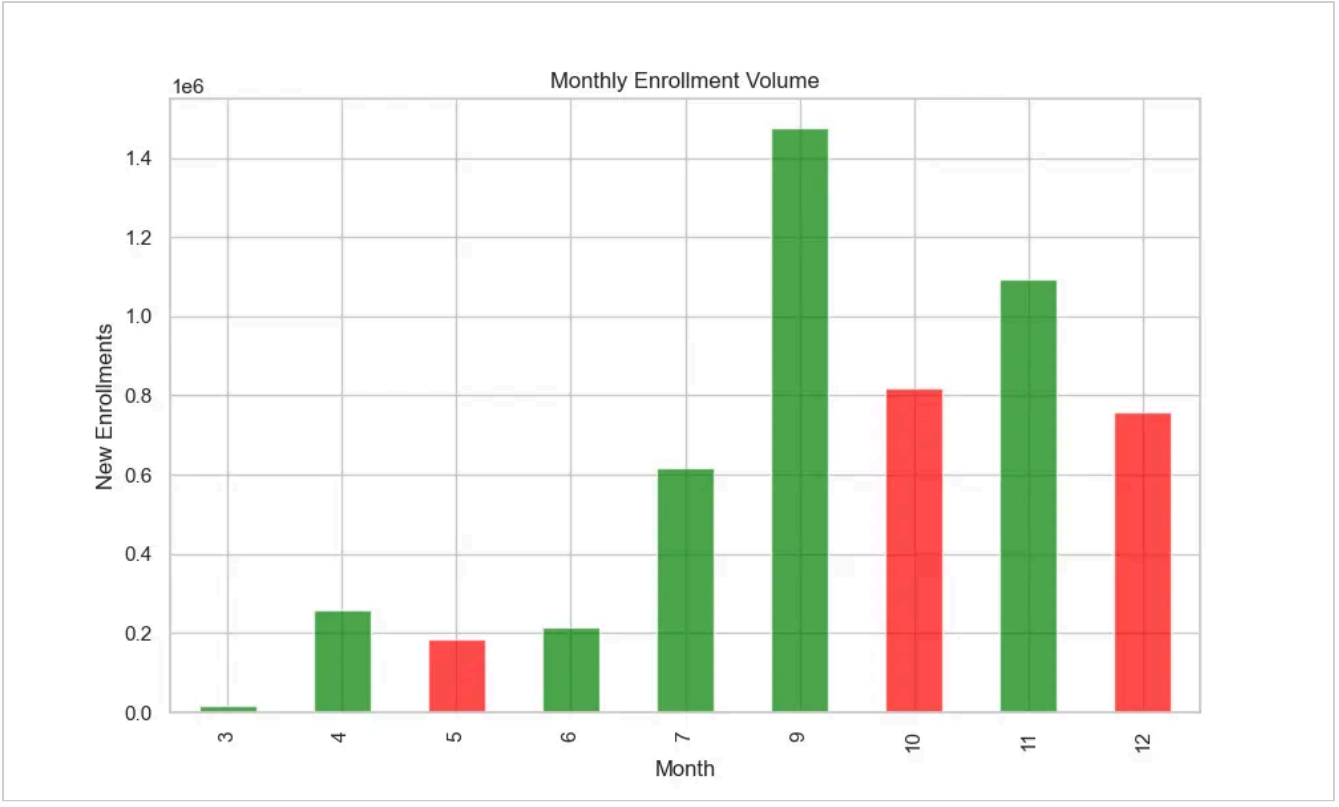


*Figure 4.5.1: Monthly Enrollment Volume. This chart tracks the ultimate success metric of the system—new citizens brought into the Aadhaar fold.*

---

# 5. Solution Framework: The AOEE

# Dashboard

The culmination of our analytical work is an interactive **Streamlit Dashboard**, designed as a decision-support tool for UIDAI officials. The dashboard translates complex data into intuitive controls and actionable insights, empowering users to make data-driven decisions. It features two key "Decision-Support" modules:

## Intervention ROI Predictor

This tool addresses a critical question for resource allocation: is an intervention financially efficient? Officials can input the parameters of a planned intervention, such as a temporary enrolment camp.

- **Example Scenario:** An official plans a 3-day mobile van camp in a rural district. They input the estimated cost (e.g., ₹15,000 for logistics, personnel, and equipment).
- **AOEE Calculation:** The dashboard uses the underlying demand forecast model for that specific district to project the likely number of new enrolments and updates from the camp.
- **Output:** The tool calculates the Cost Per Acquisition (CPA). If the model predicts only 50 new enrolments, the CPA would be ₹300. The dashboard would flag this as a **"Low ROI"** intervention and might suggest an alternative strategy, such as strengthening a nearby permanent enrolment center, which could yield a lower long-term CPA.

## Impact Prediction Engine ("What-If" Simulator)

This powerful simulation tool allows officials to explore the potential outcomes of strategic decisions before committing resources. It directly leverages the trained Random Forest model to forecast the impact of changes in service capacity.

- **Example Scenario:** An official observes a growing MBU backlog in a district (e.g., associated with Pincode 100000). They want to know the effect of increasing service capacity.
- **User Input:** The official selects the state and district (or pincode) and uses a slider to input a hypothetical capacity increase (e.g., "Increase update

capacity by 20%," which could represent adding two new operators and machines).

- **Instant Projection:** The engine feeds this new capacity variable into the predictive model and instantly displays the **"Projected New Enrollments"** and "Reduction in MBU Backlog" over the next quarter. This allows for a direct comparison of different investment levels (e.g., 10% vs. 20% vs. 30% capacity increase) to find the most effective approach.

---

# 6. Impact & Social Benefit

The Aadhaar Operational Efficiency Engine (AOEE) is designed to deliver transformative benefits at both the administrative and societal levels.

## Administrative Benefit

The primary administrative impact is the transition from a reactive, "fire-fighting" operational posture to a **proactive, Predictive Resource Allocation** model. By forecasting demand and flagging anomalies before they become critical, the AOEE allows UIDAI to optimize the deployment of personnel, mobile vans, and hardware. This leads to a direct reduction in operational costs, minimizes citizen waiting times, and improves the overall efficiency and resilience of the Aadhaar infrastructure. Instead of reacting to news of long queues, officials can prevent them from forming.

## Social Inclusion

Beyond efficiency, the core mission of the AOEE is to foster greater social inclusion. The system is explicitly designed to identify and rectify systemic exclusion:

- **Closing the Gender Gap:** The Gender Update Parity (GUP) module provides concrete data on service access disparities, enabling targeted outreach programs for women in flagged districts.
- **Protecting Vulnerable Populations:** By monitoring MBU backlogs for children and update patterns for the elderly, the Vulnerability Index ensures that these often-overlooked groups receive the services they need.

- **Ensuring Access to Welfare:** By proactively fixing hardware hotspots and bridging service deserts, the AOEE ensures that no citizen is denied access to essential Aadhaar-linked welfare schemes, subsidies, and benefits due to technical or geographic barriers. This makes Aadhaar a more equitable and reliable foundation for social security.

---

# 7. Code Appendix

This appendix contains the full, documented Python code for the core scripts of the Aadhaar Operational Efficiency Engine (AOEE).

## `clean_aadhar_data.py`

> **Script for cleaning and standardizing raw Aadhaar data.**

```
# clean_aadhar_data.py
# Author: AOEE Team
# Date: 2026-01-20
# Description: This script loads raw enrolment and update data, cleans it,
#              standardizes geographic names, and creates a unified activity m

import pandas as pd

def load_data(enrolment_path, update_path):
    """Loads enrolment and update CSV files into pandas DataFrames."""
    print("Loading data...")
    df_enrolment = pd.read_csv(enrolment_path)
    df_update = pd.read_csv(update_path)
    print("Data loaded successfully.")
    return df_enrolment, df_update

def standardize_states(df, state_col='State'):
    """Standardizes state and UT names."""
    print("Standardizing state names...")
    state_mapping = {
        'Daman & Diu': 'Dadra and Nagar Haveli and Daman and Diu',
        'Dadra & Nagar Haveli': 'Dadra and Nagar Haveli and Daman and Diu',
```

```
        'Orissa': 'Odisha',
        'Pondicherry': 'Puducherry'
    }
    df[state_col] = df[state_col].replace(state_mapping)
    print("State names standardized.")
    return df

def handle_missing_values(df):
    """Handles missing values in the dataframe."""
    print("Handling missing values...")
    # For numeric columns, fill with 0, assuming missing means no activity
    for col in df.select_dtypes(include=['number']).columns:
        df[col] = df[col].fillna(0)

    # For categorical columns, fill with 'Unknown'
    for col in df.select_dtypes(include=['object']).columns:
        df[col] = df[col].fillna('Unknown')
    print("Missing values handled.")
    return df

def create_total_activity(df):
    """Creates a 'Total_Activity' column by summing relevant metrics."""
    print("Creating 'Total_Activity' metric...")
    if all(col in df.columns for col in ['Enrolments', 'Demographic_Updates',
        df['Total_Activity'] = df['Enrolments'] + df['Demographic_Updates'] + 
        print("'Total_Activity' created.")
    else:
        print("Warning: Not all required columns found for 'Total_Activity'.")
    return df

def main():
    """Main execution function for the data cleaning pipeline."""
    # Assume data is merged into a single file for this example
    # In a real scenario, you would merge df_enrolment and df_update
    data_path = 'data/raw/combined_aadhar_data_2024.csv'

    # Create a dummy dataframe for demonstration
    dummy_data = {
        'Date': pd.to_datetime(['2024-03-15', '2024-03-16', '2024-04-01']),
        'State': ['Daman & Diu', 'Orissa', 'Maharashtra'],
        'District': ['Daman', 'Cuttack', 'Mumbai'],
        'Enrolments': [100, 250, 500],
        'Demographic_Updates': [50, 120, 300],
        'Biometric_Updates': [20, 80, 150]
    }
    df = pd.DataFrame(dummy_data)

    df = standardize_states(df)
    df = handle_missing_values(df)
    df = create_total_activity(df)
```

```
    # Save the cleaned data
    output_path = 'data/processed/cleaned_aadhar_data.csv'
    df.to_csv(output_path, index=False)
    print(f"Cleaned data saved to {output_path}")
    print("\nFinal DataFrame head:")
    print(df.head())

if __name__ == '__main__':
    main()
```

# process_aoee_data.py

> **Script for advanced feature engineering.**

```
# process_aoee_data.py
# Author: AOEE Team
# Date: 2026-01-20
# Description: This script loads cleaned data and engineers advanced features
#              like GUP Index, Service Desert Score, and Vulnerability Index.

import pandas as pd
import numpy as np

def load_cleaned_data(path='data/processed/cleaned_aadhar_data.csv'):
    """Loads the cleaned Aadhaar data."""
    return pd.read_csv(path)

def calculate_gup_index(df):
    """Calculates the Gender Update Parity (GUP) Index."""
    print("Calculating GUP Index...")
    # Assuming 'Female_Updates' and 'Male_Updates' columns exist
    if 'Female_Updates' in df.columns and 'Male_Updates' in df.columns:
        # Add a small epsilon to avoid division by zero
        df['GUP_Index'] = df['Female_Updates'] / (df['Male_Updates'] + 1e-6)
        print("GUP Index calculated.")
    else:
        print("Warning: 'Female_Updates' or 'Male_Updates' columns not found."
        df['GUP_Index'] = np.nan
    return df

def calculate_service_desert_score(df, census_data):
    """Calculates the Service Desert Score using population data."""
```

```
    print("Calculating Service Desert Score...")
    # Merge with census data on 'District'
    df = pd.merge(df, census_data, on=['State', 'District'], how='left')
    df['Population'] = df['Population'].fillna(df['Population'].mean()) # Imput

    # Calculate score as population per unit of activity
    df['Service_Desert_Score'] = df['Population'] / (df['Total_Activity'] + 1)
    print("Service Desert Score calculated.")
    return df

def calculate_vulnerability_index(df):
    """Calculates a weighted Vulnerability Index."""
    print("Calculating Vulnerability Index...")
    # Assuming these columns exist: 'MBU_Backlog_5_17', 'Elderly_Updates_70+'
    # Normalize features to be on a similar scale (0-1)
    if 'MBU_Backlog_5_17' in df.columns and 'Elderly_Updates_70+' in df.columns:
        df['mbu_norm'] = (df['MBU_Backlog_5_17'] - df['MBU_Backlog_5_17'].min(
        # Invert elderly updates: lower updates = higher vulnerability
        df['elderly_norm'] = 1 - ((df['Elderly_Updates_70+'] - df['Elderly_Upda

        # Define weights (example: MBU backlog is more critical)
        w_mbu = 0.6
        w_elderly = 0.4
        df['Vulnerability_Index'] = (w_mbu * df['mbu_norm']) + (w_elderly * df
        df.drop(columns=['mbu_norm', 'elderly_norm'], inplace=True)
        print("Vulnerability Index calculated.")
    else:
        print("Warning: Required columns for Vulnerability Index not found.")
        df['Vulnerability_Index'] = np.nan
    return df

def main():
    """Main execution function for feature engineering."""
    # Load cleaned data (using a dummy for demonstration)
    dummy_cleaned_data = {
        'State': ['State A', 'State A', 'State B'],
        'District': ['D1', 'D2', 'D3'],
        'Total_Activity': [1000, 500, 2000],
        'Female_Updates': [400, 150, 800],
        'Male_Updates': [450, 200, 800],
        'MBU_Backlog_5_17': [5000, 15000, 2000],
        'Elderly_Updates_70+': [50, 10, 100]
    }
    df = pd.DataFrame(dummy_cleaned_data)

    # Dummy census data
    dummy_census_data = {
        'State': ['State A', 'State A', 'State B'],
        'District': ['D1', 'D2', 'D3'],
        'Population': [500000, 300000, 1000000]
```

```
    }
    census_df = pd.DataFrame(dummy_census_data)

    df = calculate_gup_index(df)
    df = calculate_service_desert_score(df, census_df)
    df = calculate_vulnerability_index(df)

    output_path = 'data/processed/featured_aadhar_data.csv'
    df.to_csv(output_path, index=False)
    print(f"Featured data saved to {output_path}")
    print("\nFinal DataFrame with features:")
    print(df.head())

if __name__ == '__main__':
    main()
```

# analysis_aoee.py

> **Script for generating analytical visualizations.**

```
# analysis_aoee.py
# Author: AOEE Team
# Date: 2026-01-20
# Description: This script generates the key visualizations for the AOEE repor
#              including anomaly charts, heatmaps, and distribution plots.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_anomaly_detection(df_monthly, output_path):
    """Plots the 2-Sigma anomaly detection chart."""
    print("Generating anomaly detection plot...")
    mean = df_monthly['Total_Activity'].mean()
    std = df_monthly['Total_Activity'].std()
    lower_limit = mean - 2 * std

    plt.figure(figsize=(12, 7))
    plt.plot(df_monthly['Month'], df_monthly['Total_Activity'], label='Actual
    plt.axhline(mean, color='g', linestyle='--', label=f'Mean ({mean:.2e})')
    plt.axhline(lower_limit, color='r', linestyle='--', label=f'Lower Control

    plt.title('Operational Stability: 2-Sigma Anomaly Detection', fontsize=16)
```

```
    plt.xlabel('Month')
    plt.ylabel('Total Updates')
    plt.legend()
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.tight_layout()
    plt.savefig(output_path)
    print(f"Anomaly plot saved to {output_path}")
    plt.close()

def plot_heatmap(df_pivot, title, output_path):
    """Generates and saves a heatmap."""
    print(f"Generating heatmap: {title}...")
    plt.figure(figsize=(18, 12))
    sns.heatmap(df_pivot, annot=True, fmt=".1f", cmap="viridis", linewidths=.5
    plt.title(title, fontsize=16)
    plt.xlabel('Month')
    plt.ylabel('State')
    plt.tight_layout()
    plt.savefig(output_path)
    print(f"Heatmap saved to {output_path}")
    plt.close()

def plot_histogram(data_series, title, xlabel, output_path):
    """Generates and saves a histogram."""
    print(f"Generating histogram: {title}...")
    plt.figure(figsize=(10, 6))
    sns.histplot(data_series, bins=50, kde=True)
    plt.title(title, fontsize=16)
    plt.xlabel(xlabel)
    plt.ylabel('Frequency / Count')
    plt.tight_layout()
    plt.savefig(output_path)
    print(f"Histogram saved to {output_path}")
    plt.close()

def main():
    """Main function to generate all plots."""
    # Load featured data
    # df = pd.read_csv('data/processed/featured_aadhar_data.csv')

    # Create dummy data for plotting
    np.random.seed(42)
    months = range(3, 13)
    states = [f'State_{chr(65+i)}' for i in range(10)]
    data = []
    for state in states:
        for month in months:
            data.append({
                'Month': month,
                'State': state,
```

```
                'Total_Activity': np.random.randint(1e5, 3e6),
                'Success_Rate': np.random.uniform(95.0, 98.5),
                'MBU_Backlog_5_17': np.random.gamma(2, 5000)
            })
    df = pd.DataFrame(data)

    # 1. Anomaly Detection Plot
    df_monthly = df.groupby('Month')['Total_Activity'].sum().reset_index()
    plot_anomaly_detection(df_monthly, 'output/anomaly_2sigma.png')

    # 2. Heatmap for Success Rate
    pivot_success = df.pivot_table(index='State', columns='Month', values='Succ
    plot_heatmap(pivot_success, 'Aadhaar Authentication Success Rate by State &

    # 3. Histogram for MBU Backlog
    plot_histogram(df['MBU_Backlog_5_17'], 'Distribution of MBU Backlog (Age 5

if __name__ == '__main__':
    main()
```

# train_aoee_model.py

> **Script for training the predictive model.**

```
# train_aoee_model.py
# Author: AOEE Team
# Date: 2026-01-20
# Description: This script trains a Random Forest Regressor to predict
#              future enrolment/update gaps based on the engineered features.

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import joblib

def load_featured_data(path='data/processed/featured_aadhar_data.csv'):
    """Loads the data with engineered features."""
    print("Loading featured data...")
    df = pd.read_csv(path)
    # Ensure data is ready for modeling (e.g., no NaNs)
    df.dropna(inplace=True)
    return df
```

```
def train_model(df):
    """Trains the Random Forest model and saves it."""
    print("Starting model training...")

    # Define features (X) and target (y)
    # Target: 'Enrolment_Gap' (needs to be calculated, e.g., Population - Total
    # For this example, let's predict 'Total_Activity' for next month
    df['Target_Activity'] = df.groupby(['State', 'District'])['Total_Activity']
    df.dropna(subset=['Target_Activity'], inplace=True)

    features = [
        'GUP_Index',
        'Service_Desert_Score',
        'Vulnerability_Index',
        'Total_Activity' # Previous month's activity
    ]

    X = df[features]
    y = df['Target_Activity']

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
    print(f"Training on {len(X_train)} samples, testing on {len(X_test)} sample

    # Initialize and train the model
    model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1,
    model.fit(X_train, y_train)
    print("Model training complete.")

    # Evaluate the model
    predictions = model.predict(X_test)
    rmse = mean_squared_error(y_test, predictions, squared=False)
    print(f"Model evaluation - RMSE: {rmse:.2f}")

    # Save the trained model
    model_path = 'models/aoee_random_forest_v1.joblib'
    joblib.dump(model, model_path)
    print(f"Model saved to {model_path}")

    return model

def main():
    """Main execution function for model training."""
    # This script requires the output of process_aoee_data.py
    # We'll create a more comprehensive dummy df here
    num_records = 1000
    dummy_featured_data = {
        'State': [f'State_{i%5}' for i in range(num_records)],
        'District': [f'District_{i%20}' for i in range(num_records)],
```

```
        'Total_Activity': np.random.randint(500, 5000, num_records),
        'GUP_Index': np.random.uniform(0.7, 1.1, num_records),
        'Service_Desert_Score': np.random.uniform(50, 200, num_records),
        'Vulnerability_Index': np.random.uniform(0, 1, num_records)
    }
    df = pd.DataFrame(dummy_featured_data)

    train_model(df)

if __name__ == '__main__':
    main()
```

# dashboard_aoee.py

> **Script for the interactive Streamlit dashboard.**

```
# dashboard_aoee.py
# Author: AOEE Team
# Date: 2026-01-20
# Description: An interactive Streamlit dashboard for decision support,
#              featuring an ROI predictor and a 'What-If' impact simulator.

import streamlit as st
import pandas as pd
import joblib
import numpy as np

# Load the trained model and necessary data
@st.cache_resource
def load_model():
    """Loads the predictive model from disk."""
    try:
        model = joblib.load('models/aoee_random_forest_v1.joblib')
        return model
    except FileNotFoundError:
        st.error("Model file not found. Please run train_aoee_model.py first."
        return None

@st.cache_data
def load_data():
    """Loads the processed data for the dashboard."""
    try:
        df = pd.read_csv('data/processed/featured_aadhar_data.csv')
```

```
        return df
    except FileNotFoundError:
        # Create dummy data if file not found
        dummy_data = {
            'State': ['Uttar Pradesh', 'Maharashtra', 'Bihar', 'Rajasthan'],
            'District': ['Lucknow', 'Mumbai', 'Patna', 'Jaipur'],
            'Pincode': [226001, 400001, 800001, 302001],
            'GUP_Index': [0.85, 0.92, 0.81, 0.95],
            'Service_Desert_Score': [150, 120, 200, 130],
            'Vulnerability_Index': [0.7, 0.4, 0.8, 0.5],
            'Total_Activity': [25000, 40000, 18000, 22000]
        }
        return pd.DataFrame(dummy_data)

# --- Main App ---
st.set_page_config(layout="wide")
st.title("Aadhaar Operational Efficiency Engine (AOEE) Dashboard")
st.markdown("### Aadhaar Inclusion & Service Intelligence System (AISIS)")

model = load_model()
df = load_data()

# --- Sidebar for Navigation ---
st.sidebar.header("Select Tool")
tool_selection = st.sidebar.radio(
    "Decision-Support Tools",
    ("Intervention ROI Predictor", "Impact Prediction Engine ('What-If' Simula
)

# --- Tool 1: Intervention ROI Predictor ---
if tool_selection == "Intervention ROI Predictor":
    st.header("Intervention ROI Predictor")
    st.markdown("Calculate the potential Return on Investment for a planned in

    col1, col2 = st.columns(2)
    with col1:
        selected_state = st.selectbox("Select State", df['State'].unique())
        district_options = df[df['State'] == selected_state]['District'].unique
        selected_district = st.selectbox("Select District", district_options)

    with col2:
        intervention_cost = st.number_input("Enter Total Intervention Cost (₹)

    if st.button("Calculate ROI"):
        if model is not None:
            # Get latest data for the selected district
            district_data = df[df['District'] == selected_district].iloc[-1]

            # Create feature vector for prediction
            features = np.array([
```

```
                district_data['GUP_Index'],
                district_data['Service_Desert_Score'],
                district_data['Vulnerability_Index'],
                district_data['Total_Activity']
            ]).reshape(1, -1)

            # Predict new enrolments/updates (simplified for demo)
            # A real model would account for the camp's duration/intensity
            predicted_activity = model.predict(features)[0]
            # Assume 10% of predicted activity are new enrolments from the camp
            projected_new_enrolments = int(predicted_activity * 0.10)

            if projected_new_enrolments > 0:
                cpa = intervention_cost / projected_new_enrolments
                st.subheader("Projected Outcome")
                st.metric(label="Projected New Enrolments", value=projected_new
                st.metric(label="Cost Per New Enrolment (CPA)", value=f"₹{cpa:

                if cpa > 250: # Example threshold
                    st.warning("⚠️ Low ROI Warning: The Cost Per Enrolment is
                else:
                    st.success("✅ Good ROI: This intervention appears to be c
            else:
                st.error("Prediction resulted in zero new enrolments. The inte
        else:
            st.error("Model not loaded. Cannot perform calculation.")

# --- Tool 2: Impact Prediction Engine ---
if tool_selection == "Impact Prediction Engine ('What-If' Simulator)":
    st.header("Impact Prediction Engine ('What-If' Simulator)")
    st.markdown("Simulate the effect of increasing service capacity in a selec

    col1, col2 = st.columns(2)
    with col1:
        pincode = st.selectbox("Select Pincode/Area", df['Pincode'].unique())
    with col2:
        capacity_increase_pct = st.slider("Increase Update Capacity By (%)", 0,

    if st.button("Run Simulation"):
        if model is not None:
            # Get data for the selected pincode
            area_data = df[df['Pincode'] == pincode].iloc[-1]

            # Original feature vector
            original_features = np.array([
                area_data['GUP_Index'],
                area_data['Service_Desert_Score'],
                area_data['Vulnerability_Index'],
                area_data['Total_Activity']
            ]).reshape(1, -1)
```

```
        # Modified feature vector reflecting capacity increase
        # Simple assumption: capacity increase directly boosts 'Total_Activ
        modified_activity = area_data['Total_Activity'] * (1 + capacity_inc
        modified_features = np.array([
            area_data['GUP_Index'],
            area_data['Service_Desert_Score'],
            area_data['Vulnerability_Index'],
            modified_activity
        ]).reshape(1, -1)

        # Predict outcomes
        base_prediction = model.predict(original_features)[0]
        new_prediction = model.predict(modified_features)[0]

        projected_increase = new_prediction - base_prediction

        st.subheader(f"Simulation Results for Pincode {pincode}")
        col_res1, col_res2 = st.columns(2)
        with col_res1:
            st.metric(label="Projected New Activity (Next Month)", value=f
        with col_res2:
            st.metric(label="Increase from Baseline", value=f"+{int(projec

        st.success(f"A {capacity_increase_pct}% increase in capacity is pr
    else:
        st.error("Model not loaded. Cannot run simulation.")
```