UIDAI DATA HACKATHON

Strategic Aadhaar Planning: Trends, Risks & Operations

Understanding Needs Across Regions and Time

Team ID- UIDAI_10491

PRESENTED BY

Yash Rai, Tanishk Thakur, Aryan Nahata

Problem Statement

The current administrative ecosystem relies heavily on static historical data, which fails to capture non-linear demographic shifts. This lack of forward looking intelligence creates a critical "blind spot," leading to severe **resource asymmetry** where high value hardware sits idle in low demand zones while high velocity districts suffer operational bottlenecks. To eliminate these inefficiencies, the framework must transition from retrospective analysis to **predictive forecasting**.

Project Objective

We propose a Predictive Analytics Architecture designed to transition the framework from retrospective auditing to dynamic forecasting using LightGBM (Gradient Boosting).

Core Goals:

- **Future Demand Projection**: Forecast district-level requirements for the upcoming quarter to ensure assets are deployed before demand surges occur.
- **Anomaly Detection**: Flag statistical deviations (sudden spikes or drop-offs) to identify potential fraud vectors, data reporting errors, or camp events.
- **Demographic Segmentation**: Decompose demand by age through Lifecycle Segmentation Logic into three behavioral cohorts: cradle, classroom, workforce.
- **Operational Optimization**: Provide actionable, data-driven recommendations for mobile van deployment and staffing augmentation in high-priority clusters.

Datasets Overview

We utilized the **UIDAI-provided Aadhaar enrolment, demographic update and biometric datasets**, which were cleaned, harmonized, and merged into a single analytical dataset (**training_data_final.parquet)** to enable consistent regional and temporal analysis.

Unified Training Dataset (training_data_final.parquet)

Granularity:
Weekly, aggregated at state–district level

Dataset Size:
26,624 region–week records with complete temporal coverage

Core UIDAI Measures Used

These variables are directly derived from the original UIDAI datasets and form the basis of our analysis:

- **enrol_count**

→ Total Aadhaar enrolments aggregated across age groups (from Aadhaar Enrolment Dataset)

- **demo_count**

→ Total demographic update requests (from Aadhaar Demographic Update Dataset)

- **bio_count**

→ Total biometric update requests (from Aadhaar Biometric Update Dataset)

Regional & Temporal Identifiers:
state, district, week_start_date, week, year, week_index

Derived Temporal Features (for Trend Analysis)

To capture temporal dynamics and short-term momentum, we computed:

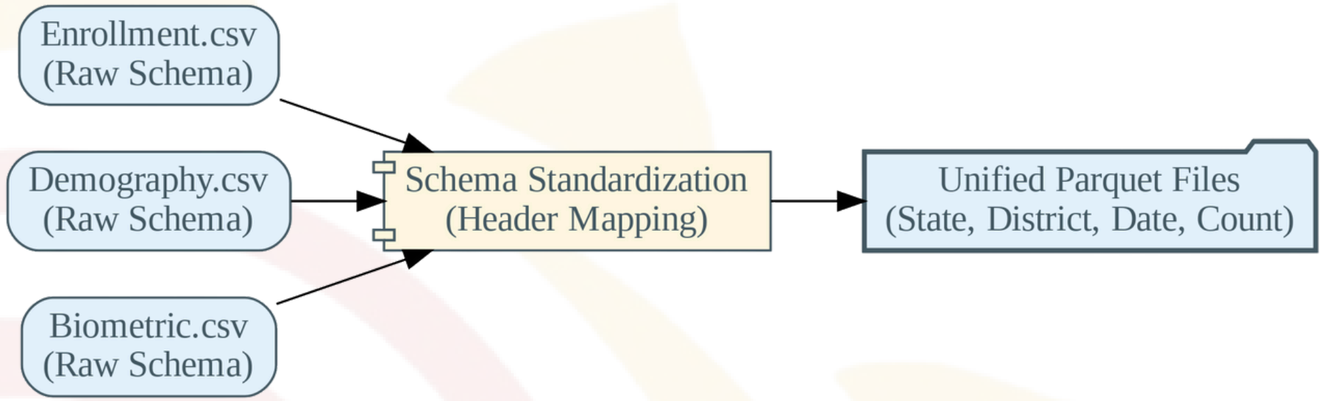
- **Lag features**: lag_1_*, lag_2_*, lag_4_*
- **Rolling averages**: roll_avg_4_*
- **Velocity features**: velocity_*

(computed separately for enrolment, demographic updates, and biometric updates)
These features were used to analyze persistence, seasonality, and change rates in Aadhaar activity.

Data Preparation Notes

- Mixed date formats and numeric encoding inconsistencies present in raw UIDAI exports were standardized prior to merging.
- All datasets were aligned to a common weekly time axis to ensure comparability.
- Administrative inconsistencies (e.g., lexical variations in state names) were normalized during preprocessing.
- The final merged dataset contains no explicit missing values, reflecting post-cleaning consolidation rather than raw data completeness.

Methodology



Phase 1:

Data Acquisition and Preprocessing Pipeline:

This phase focused on preparing a unified dataset from multiple administrative data sources. The study used three datasets: Enrollment (new registrations), Demographic Updates (name/address changes), and Biometric Updates (mandatory iris and fingerprint updates). Initial analysis revealed significant schema inconsistencies across the raw CSV files.

To address this, a multi-step preprocessing pipeline was implemented:

- **Schema Standardization:** All datasets were mapped to a common schema by resolving mismatched headers and standardizing state, district, and time fields. The cleaned data was stored in Parquet format for consistency and efficiency.
- **Temporal Aggregation:** Data was aggregated to a weekly level (Wt) to reduce noise from daily reporting variations.
- **Master Dataset Construction:** The three datasets were merged using an outer join on {State,District,Year,Week}, resulting in a final master dataset with key variables: Yenrol (enrolment), Ydemo (demographic updates), and Ybio (biometric updates).

This master dataset served as the foundation for subsequent forecasting and analysis.

Phase 2: Feature Engineering Architecture

Enhancing Model Accuracy Through Strategic Techniques

To enable the model to effectively learn from temporal patterns and structural signals in Aadhaar activity, a comprehensive feature engineering pipeline was applied to the master dataset. The objective was to extract short-term memory, long-term trends, and dynamic momentum from historical data.
The following derived features were constructed for each weekly time point t:

- **Lag Features:** Lag variables for $k \in \{1,2,4\}$ weeks were introduced.

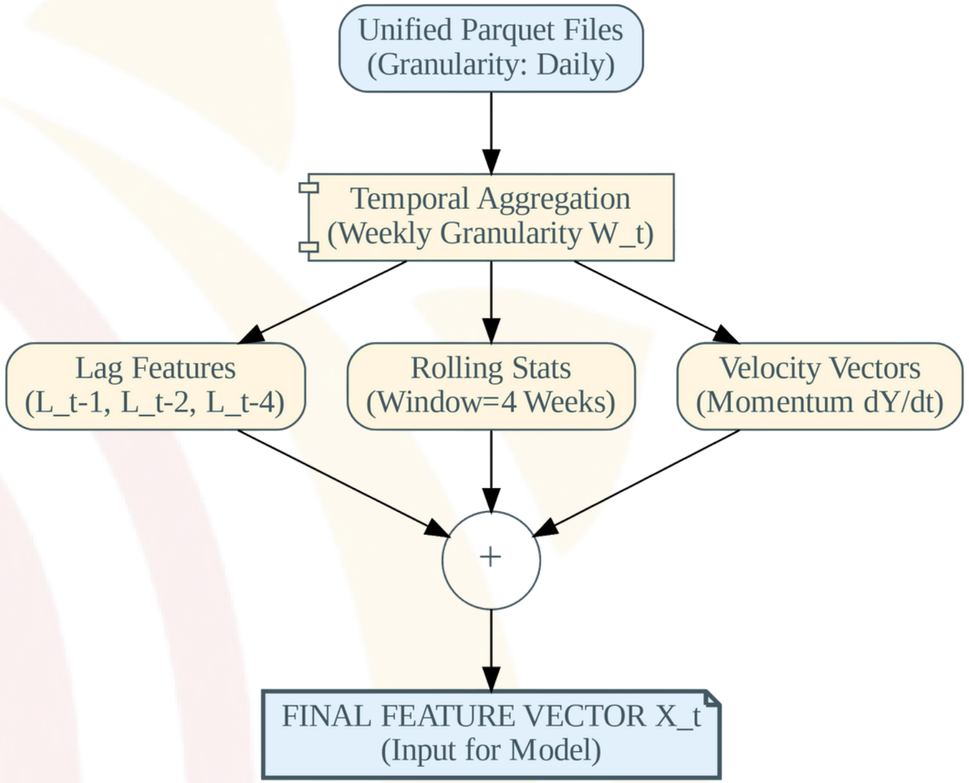
Purpose: Capture short-term autocorrelation in activity — for example, how enrolment last week influences current week volume.

- **Rolling Averages:** A 4-week rolling mean was computed for each target variable.

Purpose: Act as a low-pass filter, suppressing noise from weekly volatility (e.g., technical outages or holiday drops), while preserving structural trends.

- **Velocity Vectors**: Captures growth rate.

Purpose: Capture the rate and direction of change in weekly Aadhaar activity. Enables the model to differentiate between rising and falling sequences, even when magnitudes are similar.



Phase 3: Dual-Pipeline Architecture: Justifying Model Separation

To improve accuracy and operational relevance, we implemented a decoupled modeling strategy, creating two distinct prediction pipelines:
Model A → **Ybio** (Biometric updates)
Model B → **Ydemo** (Demographic updates)

1. Behavioral Divergence

- Ybio is driven by compliance cycles (mandatory at ages 5 & 15) → deterministic & age-dependent
- Ydemo is driven by life events (e.g., migration, marriage) → stochastic & irregular

Separate models allow Model A to focus on cyclic patterns; Model B on unpredictable spikes

2. Feature Isolation

A single model would cause feature dilution

- Model A emphasizes school-age lag variables
- Model B prioritizes velocity and migration-linked features

Decoupling ensures each model learns what truly matters

3. Resource-Specific Planning

- Ybio → needs physical kits & operators (hardware logistics)
- Ydemo → needs digital infra (servers, OTP systems)

Separate forecasts enable targeted, non-interchangeable resource allocation

Model Selection — LightGBM vs. Traditional Forecasting

LightGBM (Light Gradient Boosting Machine) was selected as the primary forecasting model over traditional approaches such as **ARIMA** or **SARIMA** due to its ability to better handle the structure and scale of the data.

- **Non-Linear Pattern Modeling:** Enrolment and update demand often follow complex, non-linear patterns influenced by regional and seasonal factors. LightGBM's tree-based architecture captures these interactions more effectively than linear statistical models.
- **Efficient Categorical Handling:** The dataset includes high-cardinality categorical features, such as hundreds of district identifiers. LightGBM natively supports such features without requiring costly one-hot encoding.
- **Scalable Global Learning:** Unlike ARIMA-based methods that require separate models per district, a single LightGBM model learns shared patterns across all regions while preserving local behavior through district-level features.

This combination makes LightGBM both more accurate and more scalable for nationwide forecasting.

Model Validation Protocol & Performance

Validation Strategy:

- Chronological Time-Series Split
- Method: Split dataset chronologically at Week 23
- Why: Prevents look-ahead bias and mimics real-world deployment
- Approach: Used Time-Series Cross-Validation — no K-Folds
- Regularization: Early stopping with patience = 100 rounds
- Result: Training halted at Iteration 204 for optimal generalization

Focused Evaluation on High-Volume Clusters

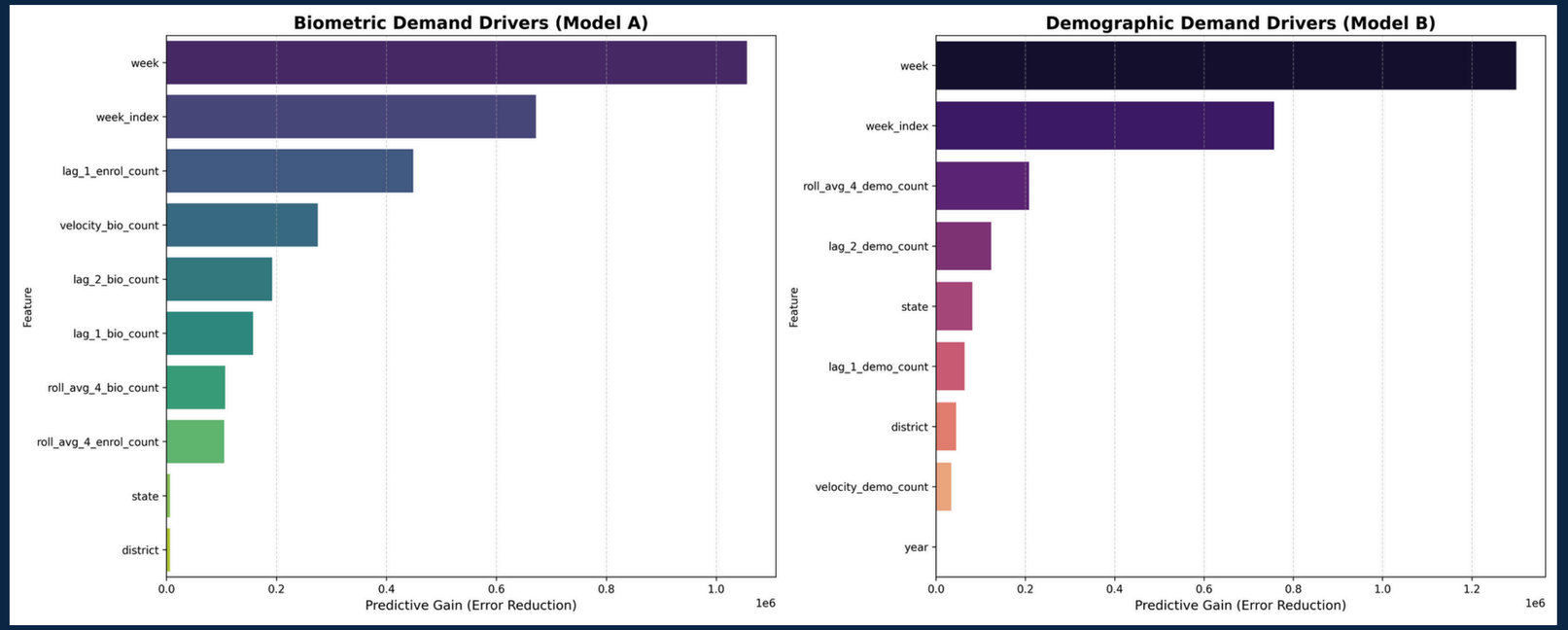
- Tested on top 20% of districts with maximum Aadhaar load
- Ensured performance in priority regions where UIDAI workload is concentrated
- Model maintained low error variance across both dense metros and rural belts

Quantitative Performance

Metric	Value	Interpretation
RMSE (log-scale)	**0.615**	Stable across urban + rural districts
Mean Absolute Error	**~849**	Avg. error per district — operationally negligible in high-load areas
Normalized RMSE	**4.20%**	Predictive accuracy of **95.8%** on key districts

Feature Significance & Validation Audit

1. **Primary Driver:** Temporal Dominance The audit reveals that both models are fundamentally driven by temporal dynamics. The feature week secured the highest information gain across both architectures (Gain > 1.0 for Biometric, > 1.2 for Demographic), followed closely by week_index. This confirms that demand is highly cyclical, with specific periods (e.g., school admission seasons) consistently triggering volume surges regardless of prior short-term history.



2. **Biometric Model Inference:** The Enrollment Precursor Beyond seasonality, the Biometric Model relies heavily on lag_1_enrol_count (3rd ranked). This establishes a clear causal link: an increase in new enrollments serves as a statistically significant "leading indicator" for subsequent biometric compliance updates.

3. **Demographic Model Inference**: Trend Stability In contrast, the Demographic Model prioritizes roll_avg_4_demo_cnt (3rd ranked). This indicates that demographic demand (e.g., address updates) is less volatile than biometric demand, driven by sustained multi-week trends rather than immediate daily fluctuations.

Conclusion The analysis validates the Dual-Model Architecture. By accurately weighting enrollment precursors for biometrics and smoothed averages for demographics, the system correctly distinguishes between the distinct behavioral drivers of the two service streams.

DATA ANALYSIS AND VISUALIZATION

This section presents the key analytical insights derived from the forecasted Aadhaar data for the next 3 months, supported by forecasting resource assignment, anomaly detection, and demographic segmentation. Each insight is accompanied by targeted visualizations that inform actionable recommendations.(refer forecast_3_months_predicted.csv on github)

1. Strategic Resource Allocation (The Budget Map)

Data Source: topic1_quarterly_volume.png
Biometric update demand is **highly concentrated**, with just **five states expected to handle over 60% of the national workload** in the upcoming quarter. This uneven distribution highlights the need for focused operational and financial planning.

Priority States and Operational Guidance

1. Maharashtra

- Predicted Load: ~1.76 million (next 3 months)
- Operational Focus:
  - Maintain full operational capacity with no budget reductions
  - Reactivate all dormant biometric kits
  - Prioritize urban school clusters in Pune, Thane, and Nashik

2. Uttar Pradesh

- Predicted Load: ~1.75 million (next 3 months)
- Operational Focus:
  - Treat as a high-growth priority state
  - Rapidly mobilize Panchayat Bhawans in Kanpur and Lucknow
  - Expand local outreach to prevent early backlogs

3. Tamil Nadu

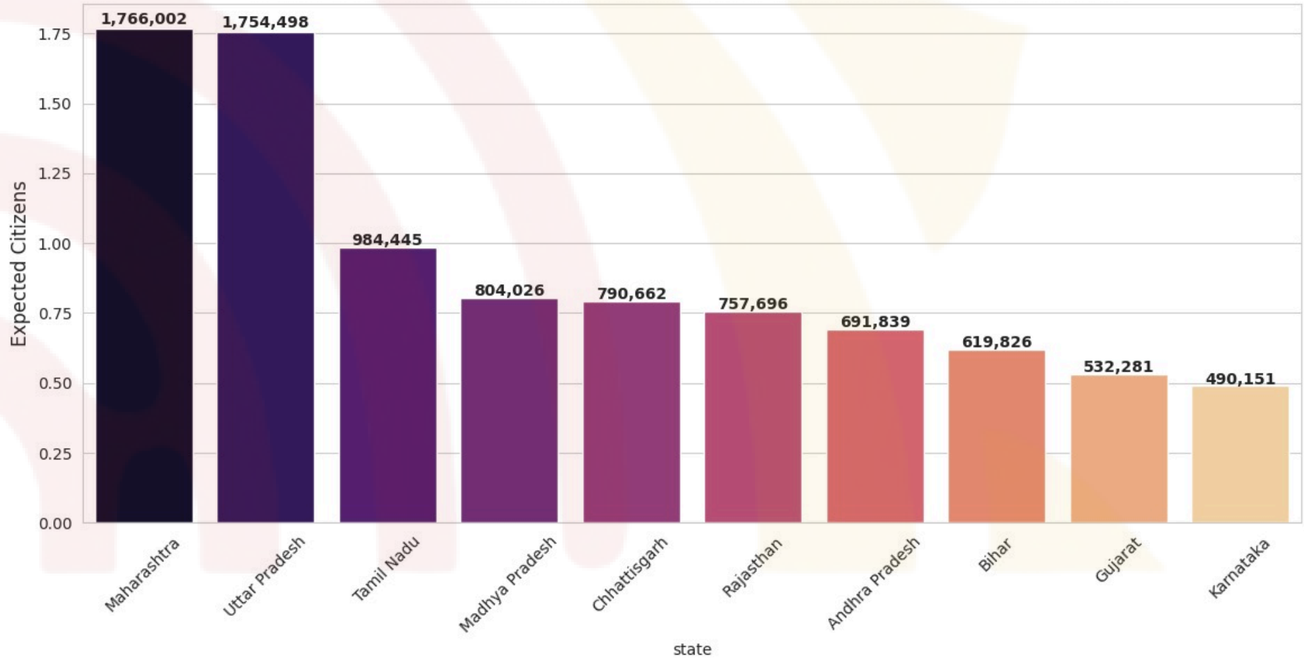
- Predicted Load: ~0.98 million (next 3 months)
- Operational Focus:
  - Continue steady-state operations
  - Monitor for localized spikes but no immediate scale-up required

4. Madhya Pradesh

- Predicted Load: ~0.80 million (next 3 months)
- Operational Focus:
  - Concentrate resources around Indore and Bhopal hubs
  - Support surrounding districts through hub-and-spoke deployment

5. Chhattisgarh

- Predicted Load: ~0.79 million (next 3 months)
- Operational Focus:
  - Address high demand in tribal regions
  - Deploy specialized mobile enrolment units with extended reach

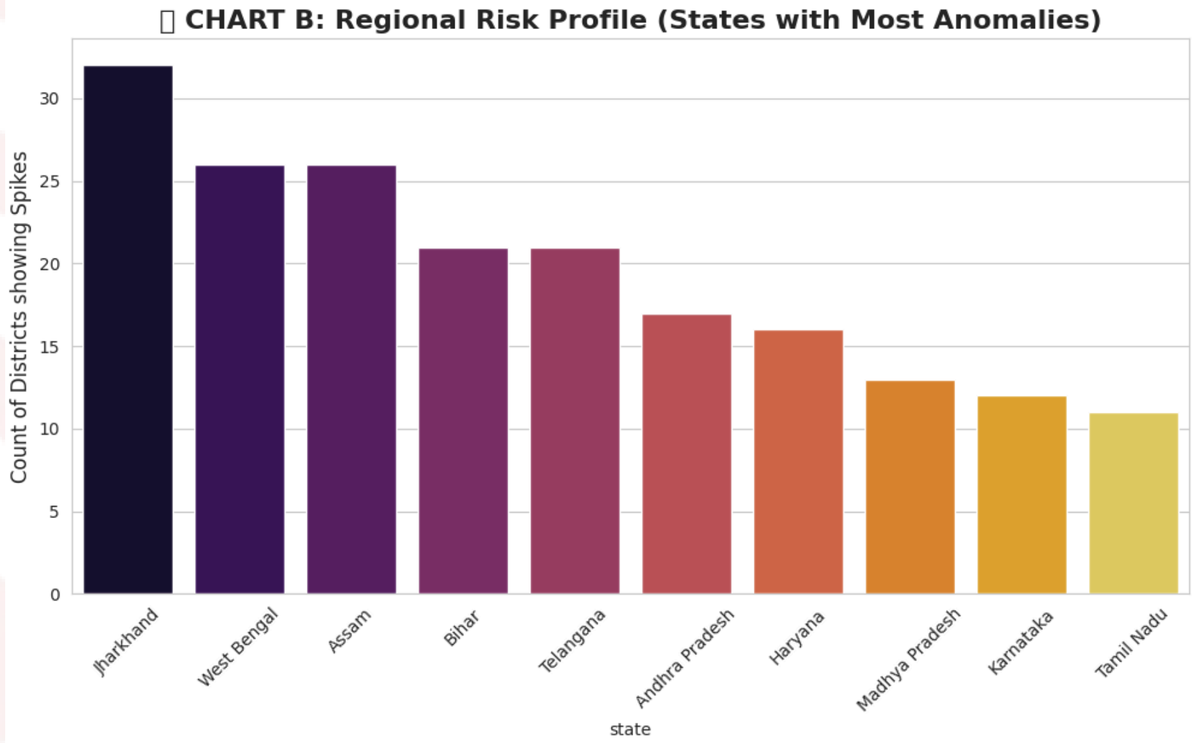


Financial Recommendation

- Immediate Action: Allocate 60% of the Q1 Aadhaar budget to Maharashtra and Uttar Pradesh
- Rationale: These two states alone account for nearly half of the projected national demand
- Risk: Delayed funding in these regions may lead to nationwide processing backlogs

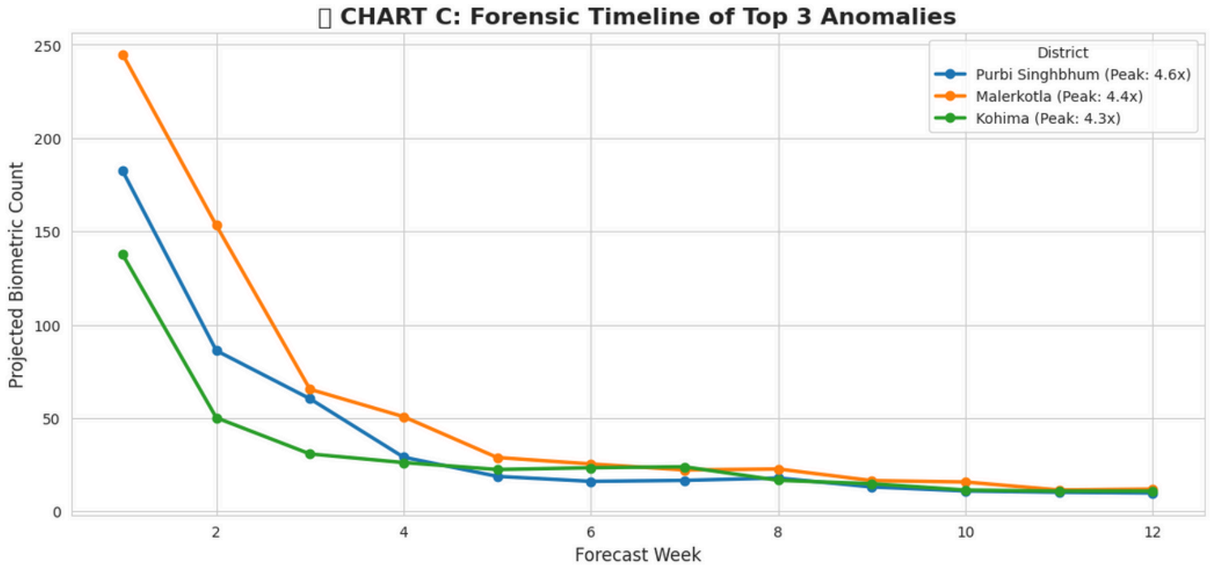
2. Anomaly Detection and Regional Volatility Analysis

Post-forecasting, we applied a statistical outlier detection algorithm (Z-Score analysis) to identify districts exhibiting significant deviations from their historical baselines. The model flagged specific high-volatility clusters where predicted demand exceeded the local rolling average by a factor of **4.0 (>400%)**. A granular analysis of these anomalies reveals distinct underlying socio-economic drivers.



2.1: The Tribal Belt Anomaly (Jharkhand)

The most pronounced volatility was observed in the state of Jharkhand, specifically in the districts of **Purbi Singhbhum** (4.63 times spike), **Pashchimi Singhbhum** (4.24 times spike), and **Godda** (4.09 times spike).
Interpretation: Such a simultaneous, high-magnitude surge across contiguous tribal-dominated districts is statistically unlikely to be organic. Instead, it bears the signature of a "Camp-Mode Activation“.



2.2 The Border & Migrant Anomaly (Punjab and Nagaland)

Isolated but severe spikes were detected in **Malerkotla, Punjab** (**4.38** times) and **Kohima, Nagaland** (**4.35** times).
Kohima (Nagaland): The surge in this border capital warrants scrutiny. While it may reflect genuine administrative clearing, a deviation of this magnitude in a sensitive border zone necessitates validation to rule out rapid in-migration or organized attempts to legitimize residency status via mass biometric updates. Reports from 2025 highlight ongoing concerns in Nagaland over illegal immigration, stricter ILP (Inner Line Permit) enforcement involving Aadhaar verification, and efforts to curb fraudulent identity use in border areas, which could contribute to such clustered activity. - (reference to verify our insight sentinelassam.com)

Malerkotla (Punjab): Conversely, the observed spike in this district aligns with established seasonal labor migration patterns, positioning it as a key hub for transient workers. This surge likely corresponds to the post-harvest "return" phase, amplified by external factors such as geopolitical tensions. For instance, reports from May 2025 highlight an accelerated exodus of migrant workers from Punjab amid India-Pakistan border tensions, coinciding with the end of the wheat harvest season and potentially driving mass Aadhaar updates before departure.
-(reference to verify our insight *outlookbusiness.com*)

2.3 The Transient Population Effect (Ayodhya, Uttar Pradesh)

(**ref. vigilance_report.csv**)
A unique anomaly was identified in **Ayodhya** (**4.21** times). Unlike the tribal or border anomalies, this surge is attributed to Transient Population Dynamics. As a major religious tourism and construction hub, the district experiences massive floating populations.

3. Demographic Lifecycle Analysis and Resource Optimization

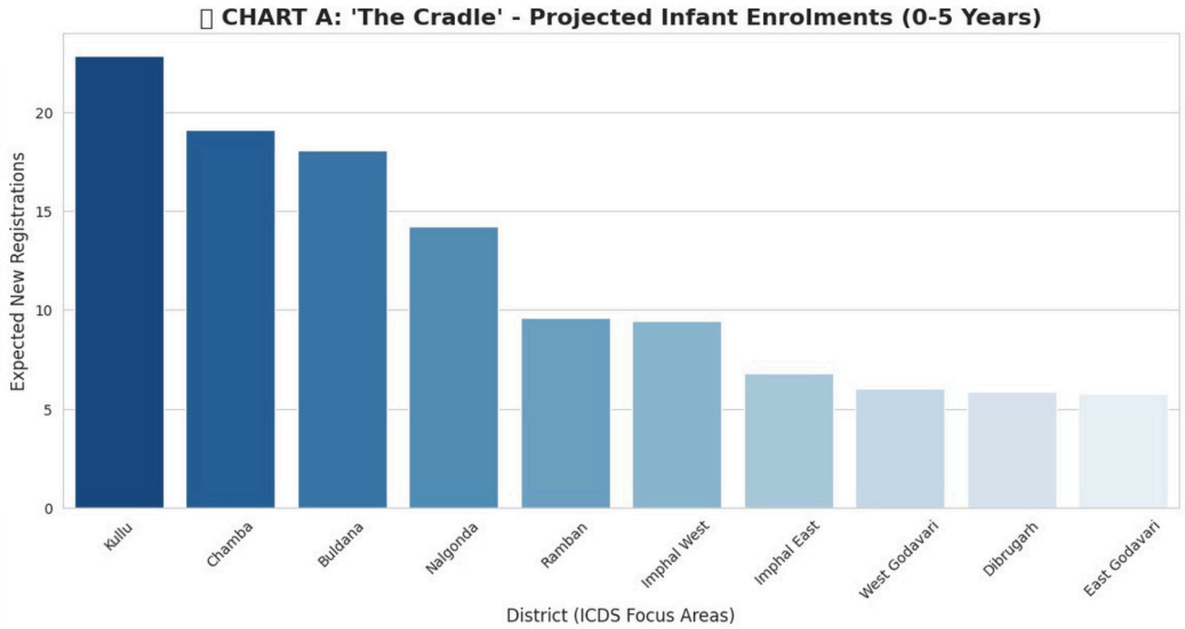
While aggregate forecasting determines how much capacity is required, it fails to identify what type of infrastructure is needed. To address this, we applied a **Lifecycle Segmentation Logic** to decompose the predicted demand into three behavioral cohorts: The Cradle (0–5 years), The Classroom (5–17 years), and The Workforce (18+ years).

3.1 The Cradle (Infant Enrolment Dynamics)

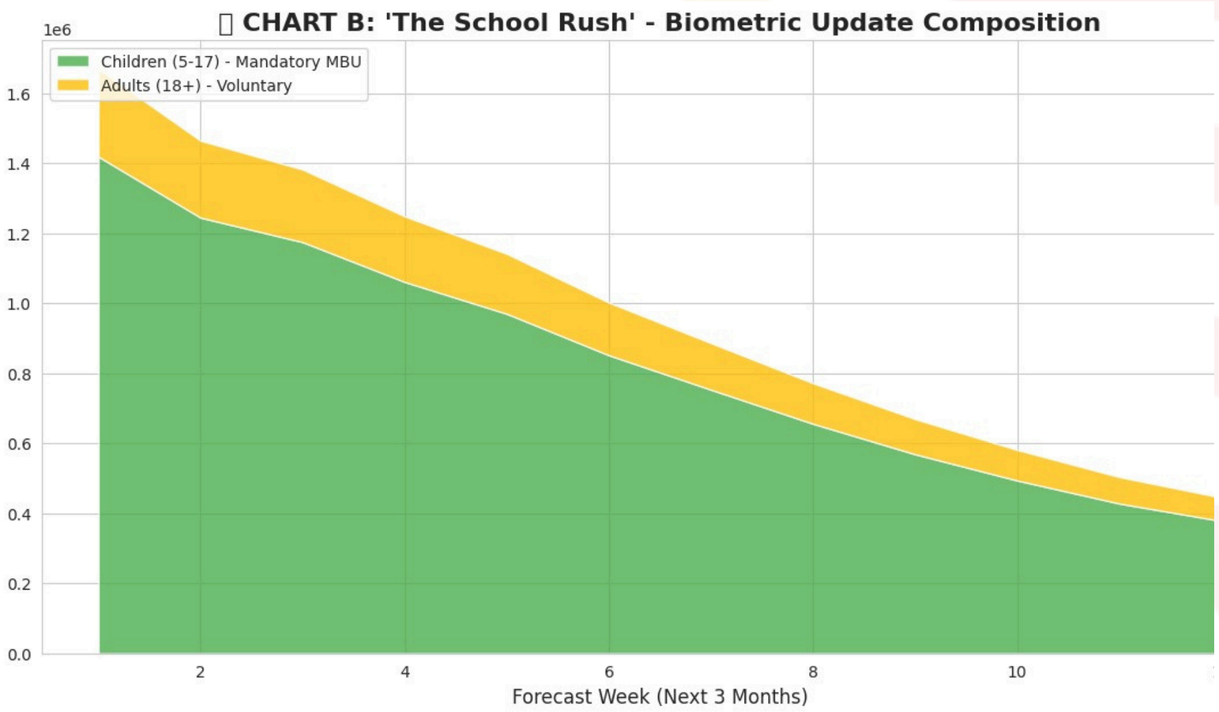
The first stage of the citizen lifecycle involves new enrolments for the 0–5 age group. The model detected a distinct I**nfant Momentum** in states that do not traditionally dominate the total volume charts.

While **Uttar Pradesh** leads in absolute numbers due to population size, **Andhra Pradesh** and **Himachal Pradesh** exhibit a disproportionately high ratio of infant enrolments relative to their total activity.

Since children under 5 years are exempt from biometric capture (iris/fingerprint), the demand in these **Cradle Clusters** is strictly for demographic registration. Consequently, the deployment of expensive biometric stations here is inefficient.



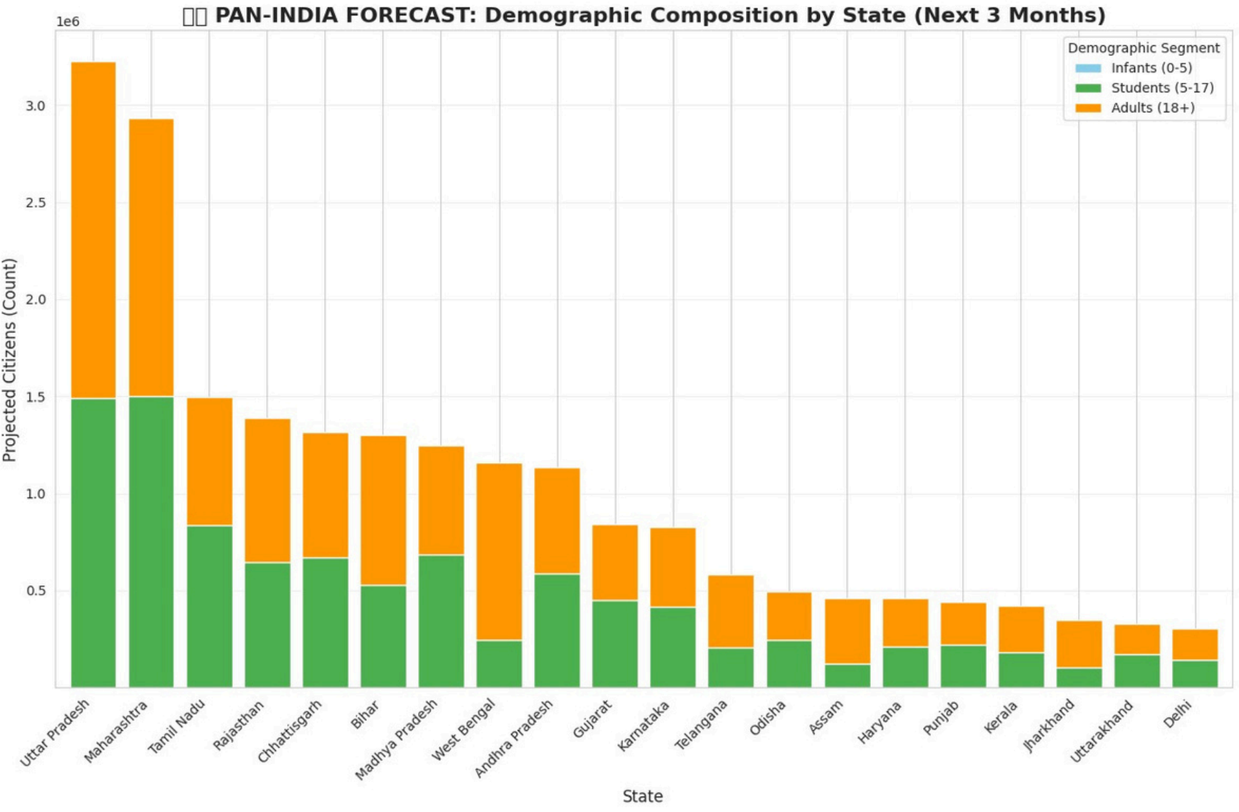
3.2 The Classroom



The most critical operational strain is observed in the 5–17 age bracket, driven by **Mandatory Biometric Updates (MBU)** required at ages 5 and 15. The temporal analysis reveals a massive, sustained "Green Wave" of student-driven demand that dwarfs adult activity.

The **Maharashtra Phenomenon**: The district-level granularity uncovers a synchronized administrative anomaly in Maharashtra. The state accounts for 10 of the top 16 high-demand districts nationally.
Pune (93,208), Nashik (92,875), and Thane (92,078) are projecting quarterly volumes nearly double that of major capital cities like Jaipur (~50k).
This suggests a state-level **Compliance Blitz**, likely linked to strict enforcement of biometric KYC for school admissions and state scholarship schemes.
Outside Maharashtra, high demand is exclusively correlated with Tier-1 education hubs such as Ahmedabad, Jaipur, and Kota.
Unlike the infant cohort, this group requires full biometric hardware. The concentration of demand in specific education hubs necessitates the establishment of Physical School Camps to manage the backlog, as online portals cannot capture the required biometric data.

3.3 The Workforce (Migration and Mobility)

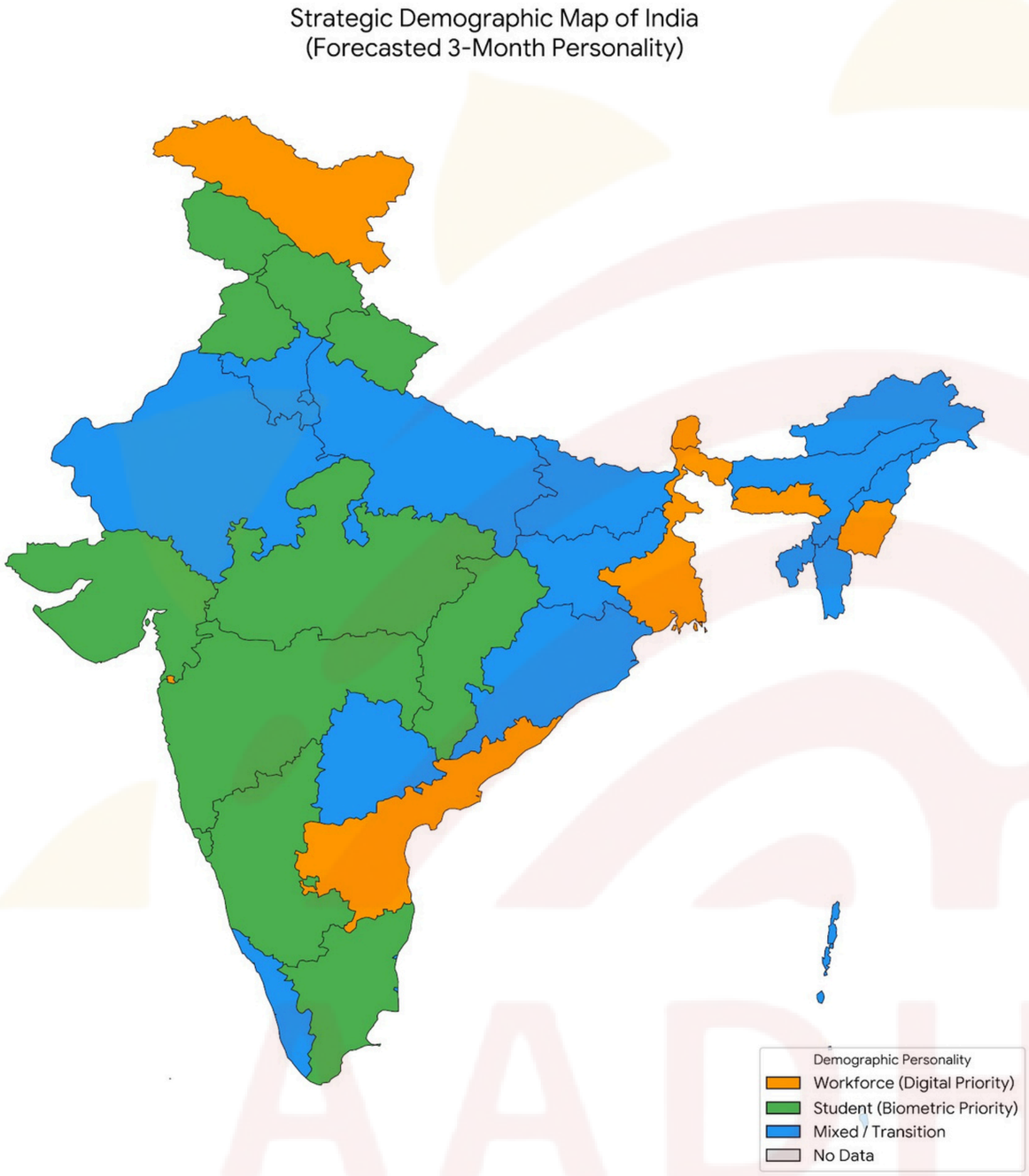


The final stage of the lifecycle, the 18+ cohort, is characterized by voluntary demographic updates (address and mobile changes) driven by economic migration. Figure visualizes the **Demographic Personality** of each state, highlighting a stark hardware-vs-digital divide.

States such as West Bengal (84.6% adult share), Ladakh (89.7%), and Uttar Pradesh are dominated by adult demand (from dataset analysis)
. The analysis indicates this is driven by the migrant workforce regularizing documentation for banking and labor mobility.

Since adult biometric updates are rare compared to demographic corrections, the optimal strategy for these states is Digital Capacity Augmentation. Rather than deploying physical vans, resources should be allocated to bolstering server capacity for the **Self-Service Update Portal (SSUP)** to prevent downtime during evening peak hours when the workforce is most active.

sources validating our insights of this slide:
Migration in India: Link: https://organiser.org/2025/11/20/326499/bharat/migration-in-india-data-driven-governance-framework-and-reforms-transforming-migration
How to Update Address in Aadhaar Card Online – Link: https://www.rbl.bank.in/blog/banking/digital-banking/update-aadhaar-address-online-guide



- Green (Student / Biometric Priority): States where demand is dominated (>50%) by the 5-17 age group. (e.g., Andhra Pradesh, Maharashtra).
- Orange (Workforce / Digital Priority): States where demand is dominated (>75%) by the 18+ workforce. (e.g., West Bengal).
- Blue (Mixed / Transition): States with a balanced demographic load.

Limitations

- Temporal Horizon Constraint: The dataset spans only 10 months, precluding the modeling of full annual seasonality (12-month cycles). Consequently, the model cannot validate recurring annual events that fall outside the observed time window.
- Geospatial Resolution Cap: Forecasts are aggregated at the District Level, averaging demand across diverse sub-regions. This lack of sub district granularity (Tehsil or PIN code level) limits the precise routing of mobile vans to specific high demand villages.
- Exogenous Variable Blindness: The architecture relies exclusively on endogenous time series signals (Lags/Velocity). It excludes external high impact regressors such as local holiday calendars or exam schedules which accounts for the residual NRMSE of ~4.2% during nonstandard operational weeks.
- Latency: The feature engineering relies on 4 week lag windows (lag_4, roll_avg_4). Therefore, the system requires a mandatory 4 week warmup period of historical data ingestion before it can generate reliable inferences for any newly formed administrative district.

Impact & Applicability: UIDAI Decision Framework

All figures are planning-level estimates to illustrate operational decision-making, not audited projections.

Core Question

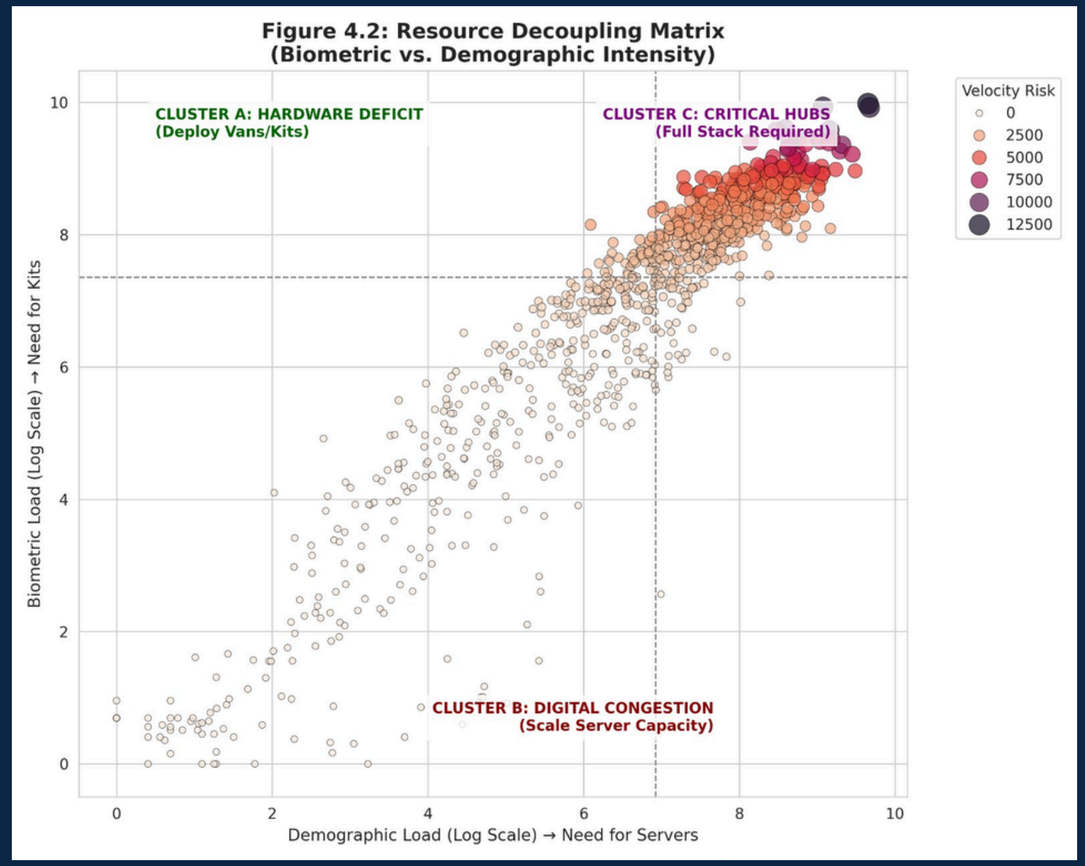
How can forecasting convert Aadhaar operations from reactive to proactive,
reducing backlogs while improving equitable access?

Decision Flow

Input → Forecast → Resource Allocation → Budget Prioritization → Backlog Reduction

Operational Resource Decoupling (Visual Decision Aid)

Districts are positioned by demographic update intensity (digital/server load) and biometric update intensity (physical kits/operators), both log-scaled. Bubble size and color indicate normalized velocity risk (rate of change in demand). The matrix enables targeted intervention—hardware deployment, server scaling, or full-stack response—based on dominant stress factors.



Why this trivariate graph belongs here:

- It bridges forecast outputs → concrete actions
- It explains why staffing and budget decisions differ by region
- It prevents one-size-fits-all deployment

National Applicability

- Demand concentrated in top ~5 states (≈60% of national load).
- Enables prioritization of ~60% of UIDAI's annual budget (~₹600 crore) toward peak regions and periods.
- Forecast reliability (NRMSE ~4–5%) is sufficient for planning decisions, not exact counts.

Why This Matters

- Faster enrolment and updates → improved access to welfare schemes.
- Reduced idle capacity in low-demand regions.
- Avoidance of operational inefficiencies worth tens of crores annually at national scale.

Way Forward

- Extend forecast horizon to 6–12 months using multi-year UIDAI data.
- Link forecasts with centre capacity and staffing for automated alerts.
- Deploy dashboards for real-time monitoring and intervention.

Worked Example: Maharashtra (High-Load State)

Step	Key Insight (Illustrative)
Input	~10 months UIDAI enrolment, demographic & biometric data. LightGBM with lags, rolling averages, velocity, and seasonality.
Forecast	~1.7–1.8M biometric updates over 3 months (~24k/day); ~80–85% demand from ages 5–17.
Staffing	At ~40 updates/operator/day, ~600 operators required. Existing centres absorb most load; remaining gap via targeted redeployment or short-term hiring.
Budget	~₹40–45 crore (order-of-magnitude) for staffing and mobile infrastructure during peak periods.
Backlog Impact	Proactive deployment can reduce peak backlogs by **~15–25%** through early demand detection and resource decoupling.

Code Snippets

This section includes all the code from our preprocessing, feature engineering, model training, and anomaly detection pipeline.

For full reproducibility, you can explore the complete analysis, charts, and datasets in our GitHub repository:
🔗 **GitHub Repository** :Github Repo Link -https://github.com/yash-rai-93/UIDAI_10491-uidai-hackathon

Included in GitHub:

- Jupyter Notebooks (.ipynb)
- Preprocessed datasets (.csv, .parquet)
- Model artifacts & metrics
- All graphs, maps, and final output files

Phase 1: The Data Ingestion & Unification Pipeline.

```
import pandas as pd
import glob
import os
from tqdm.notebook import tqdm

# Configuration: Input directory paths for raw administrative data
INPUT_PATHS = {
    "enrolment": "/kaggle/input/enrollment-dataset/api_data_aadhar_enrolment",
    "demographic": "/kaggle/input/demographic/api_data_aadhar_demographic",
    "biometric": "/kaggle/input/biometric/api_data_aadhar_biometric"
}

def merge_raw_dataset(dataset_name, folder_path):
    """
    Recursively locates CSV files, standardizes headers, and consolidates
    data into a unified Parquet file for downstream processing.
    """

    # 1. File Discovery
    all_files = []
    for ext in ['*.csv']:
        # Recursive glob to handle nested directory structures
        found = glob.glob(os.path.join(folder_path, '**', ext), recursive=True)
        all_files.extend(found)

    if not all_files:
        print(f"Error: No source files found in {folder_path}")
        return
```

```
    df_list = []

    # 2. Data Ingestion
    for filepath in tqdm(all_files, desc=f"Ingesting {dataset_name}"):
        try:
            # Reading as string (dtype=str) ensures preservation of leading zeros (e.g., PIN codes)
            # low_memory=False prevents type inference warnings on large files
            df = pd.read_csv(filepath, dtype=str, low_memory=False)

            # Schema Standardization: Remove leading/trailing whitespace from headers
            df.columns = df.columns.str.strip()

            df_list.append(df)
        except Exception as e:
            print(f"Read Error ({os.path.basename(filepath)}): {e}")

    # 3. Aggregation and Serialization
    if df_list:
        master_df = pd.concat(df_list, ignore_index=True)

        # Export to Parquet for optimized I/O performance
        output_file = f"{dataset_name}_master.parquet"
        master_df.to_parquet(output_file, index=False)

        # Log dimensions for verification
        print(f"Export Complete: {output_file} | Rows: {len(master_df):,} | Features:
{len(master_df.columns)}")
        print(f"Schema: {list(master_df.columns)}")
    else:
        print(f"Merge operation yielded no data for {dataset_name}.")

# Execution Pipeline
for name, path in INPUT_PATHS.items():
    if os.path.exists(path):
        merge_raw_dataset(name, path)
    else:
        print(f"Path not found: {path} (Skipping {name})")
```

Phase 2: Data Sanitation & Standardization

```python
import pandas as pd
import numpy as np

# Configuration: Paths to ingested Parquet files
DATASETS = {
    'enrolment': '/kaggle/input/oracle-dataset-1/enrolment_master.parquet',
    'demographic': '/kaggle/input/oracle-dataset-1/demographic_master.parquet',
    'biometric': '/kaggle/input/oracle-dataset-1/biometric_master.parquet'
}

def clean_and_verify(name, filepath):
    """
    Performs rigorous schema validation, column standardization, and
    numeric type coercion to ensure downstream integrity.
    """
    print(f"\nInitiating Validation Protocol: {name.upper()}")

    try:
        df = pd.read_parquet(filepath)

        # 1. Schema Harmonization
        # Mapping disparate header variances to a unified ontology
        rename_map = {
            'demo_age_17_': 'age_18_above',  # Normalizing truncated headers
            'bio_age_17_': 'age_18_above',
            'age_18_greater': 'age_18_above',
            'demo_age_5_17': 'age_5_17',     # Standardizing the student bracket
            'bio_age_5_17': 'age_5_17',
            'age_0_5': 'age_0_5'
        }
        df.rename(columns=rename_map, inplace=True)
```

```python
        # 2. Type Coercion (Numeric Safety)
        # Identifying columns representing quantitative volume
        count_cols = [c for c in df.columns if 'age_' in c]
        print(f"Detected Quantitative Features: {count_cols}")

        for col in count_cols:
            # Enforcing numeric types; coercion handles non-numeric artifacts (e.g., 'NULL')
            df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)

        # 3. Integrity Check (Volume Audit)
        # Aggregating total volume to verify data magnitude (sanity check)
        total_volume = df[count_cols].sum().sum()
        print(f"Total Aggregate Volume: {total_volume:,.0f}")

        # 4. Feature Alignment
        # Ensuring all dataframes possess the 'Infant' feature for seamless merging
        if 'age_0_5' not in df.columns:
            df['age_0_5'] = 0.0
            print("Feature Injection: 'age_0_5' (Placeholder initialized)")

        # 5. Serialization
        output_file = f"{name}_clean.parquet"
        df.to_parquet(output_file, index=False)
        print(f"Serialization Complete: {output_file}")

    except Exception as e:
        print(f"Validation Failure ({name}): {e}")

# Execution Pipeline
for name, path in DATASETS.items():
    clean_and_verify(name, path)
```

Phase 3: The Fusion & Feature Engineering Engine.

```python
import pandas as pd
import numpy as np

def create_master_training_data():
    """
    Executes the final ETL pipeline: Temporal Aggregation, Fusion, and
    Autoregressive Feature Engineering.
    Generates the 'training_data_final.parquet' asset for model ingestion.
    """
    print("Initiating Master Dataset Construction...")

    # 1. Data Ingestion
    # Loading uniform parquet files from the pre-processing layer
    df_enrol = pd.read_parquet('/kaggle/input/oracle-dataset-2/enrolment_uniform.parquet')
    df_demo = pd.read_parquet('/kaggle/input/oracle-dataset-2/demographic_uniform.parquet')
    df_bio = pd.read_parquet('/kaggle/input/oracle-dataset-2/biometric_uniform.parquet')

    # 2. Temporal Aggregation Module (Weekly Granularity)
    print("Executing Weekly Volume Aggregation...")

    def get_weekly_sum(df, target_col):
        # Dynamic date resolution and type enforcement
        col_date = [c for c in df.columns if 'date' in c.lower()][0]
        df[col_date] = pd.to_datetime(df[col_date], errors='coerce')
        df.dropna(subset=[col_date], inplace=True)

        # Volume Summation: Aggregating distinct age brackets into total footfall
        age_cols = [c for c in df.columns if 'age_' in c]
        df['total_people'] = df[age_cols].sum(axis=1)

        # ISO Calendar extraction for standardized reporting
        df['year'] = df[col_date].dt.year
        df['week'] = df[col_date].dt.isocalendar().week

        # GroupBy Operation: State-District-Year-Week
        weekly = df.groupby(['state', 'district', 'year', 'week'])
['total_people'].sum().reset_index().rename(columns={'total_people': target_col}, inplace=True)

        # Date Normalization (Week Start Date) for plotting alignment
        weekly['week_start_date'] = pd.to_datetime(
            weekly['year'].astype(str) + '-' + weekly['week'].astype(str) + '-1', format='%Y-%W-%w'
        )
        return weekly

        # Processing streams
    df_e = get_weekly_sum(df_enrol, 'enrol_count')
    df_d = get_weekly_sum(df_demo, 'demo_count')
    df_b = get_weekly_sum(df_bio, 'bio_count')
```

```python
    # 3. Time-Series Unification (Outer Join Strategy)
    # Preserves sparse data points across different administrative streams
    print("Merging Streams into Unified Time-Series Index...")
    master = pd.merge(df_e, df_d, on=['state', 'district', 'year', 'week', 'week_start_date'],
how='outer')
    master = pd.merge(master, df_b, on=['state', 'district', 'year', 'week', 'week_start_date'],
how='outer')

    # Imputation: Zero-filling for non-event weeks to maintain continuity
    master.fillna(0, inplace=True)

    # 4. Feature Engineering: Lag and Velocity Vectors
    print("Generating Autoregressive Features...")

    # Sorting ensures correct temporal shifting
    master.sort_values(by=['state', 'district', 'year', 'week'], inplace=True)
    master['week_index'] = master.groupby(['state', 'district']).cumcount()

    target_cols = ['enrol_count', 'demo_count', 'bio_count']

    for col in target_cols:
        # Autoregressive Lags (t-1, t-2, t-4) for seasonality detection
        master[f'lag_1_{col}'] = master.groupby('district')[col].shift(1)
        master[f'lag_2_{col}'] = master.groupby('district')[col].shift(2)
        master[f'lag_4_{col}'] = master.groupby('district')[col].shift(4)

        # Rolling Statistics: 4-week moving average (Trend Smoothing)
        master[f'roll_avg_4_{col}'] = master.groupby('district')[col].transform(lambda x:
x.rolling(4).mean().shift(1))

        # Velocity Vector (First-order derivative approximation)
        # Mathematical definition: (Volume(t-1) - Volume(t-2)) / (Volume(t-2) + epsilon)
        # Captures the rate of change (Momentum) irrespective of absolute volume
        master[f'velocity_{col}'] = (master[f'lag_1_{col}'] - master[f'lag_2_{col}']) /
(master[f'lag_2_{col}'] + 1)

    # Drop initial rows with NaN values resulting from lag generation
    master.dropna(inplace=True)

    # 5. Serialization
    output_file = 'training_data_final.parquet'
    print(f"Serialization: Saving to {output_file}...")
    master.to_parquet(output_file)

    print(f"Pipeline Complete. Dimensions: {len(master):,} rows")
    print(f"Peak Biometric Volume: {master['bio_count'].max():,.0f}")

# Execution
create_master_training_data()
```

Phase 4: The Intelligence Engine (Model Training)

```python
import pandas as pd
import numpy as np
import lightgbm as lgb
import joblib
import os
from sklearn.metrics import mean_absolute_error

# Configuration: Path definitions and training hyperparameters
DATA_PATH = '/kaggle/working/training_data_final.parquet'
TEST_WEEKS = 8  # Validation window (Final 2 months held out)
MODEL_DIR = 'saved_models' # Artifact storage directory

def train_and_save_champion():
    """
    Executes the Gradient Boosting training pipeline.
    Includes Log-Transformation, Temporal Splitting, and Model Validation.
    """
    print("Initiating Model Training Pipeline...")
    df = pd.read_parquet(DATA_PATH)

    # 1. Variance Stabilization
    # Applying Log(1+x) transformation to normalize high-variance target distributions
    df['log_bio'] = np.log1p(df['bio_count'])
    df['log_demo'] = np.log1p(df['demo_count'])

    # Categorical Type Enforcement for LightGBM optimization
    for col in ['state', 'district']:
        df[col] = df[col].astype('category')

    # 2. Temporal Partitioning
    # Splitting data chronologically to prevent data leakage (Look-ahead bias)
    max_week = df['week_index'].max()
    split_point = max_week - TEST_WEEKS

    print(f"Partitioning Data: Training (Weeks 0-{split_point}) | Validation (Weeks {split_point+1}-
{max_week})")
    train = df[df['week_index'] <= split_point].copy()
    test = df[df['week_index'] > split_point].copy()

    # 3. Feature Selection
    base_feats = ['week_index', 'district', 'state', 'year', 'week']

    feats_bio = base_feats + [
        'lag_1_bio_count', 'lag_2_bio_count', 'roll_avg_4_bio_count',
        'velocity_bio_count',
        'lag_1_enrol_count', 'roll_avg_4_enrol_count'
    ]

    feats_demo = base_feats + [
        'lag_1_demo_count', 'lag_2_demo_count', 'roll_avg_4_demo_count',
        'velocity_demo_count'
    ]
```

```python
    # Hyperparameters optimized for regression on skewed data
    params = {
        'objective': 'regression',
        'metric': 'rmse',
        'boosting_type': 'gbdt',
        'learning_rate': 0.05,
        'num_leaves': 63,
        'feature_fraction': 0.8,
        'bagging_fraction': 0.8,
        'verbose': -1
    }

    # --- Model A: Biometric Demand Forecast ---
    print("Training Biometric Model (LightGBM)...")
    dtrain_bio = lgb.Dataset(train[feats_bio], label=train['log_bio'])
    dtest_bio = lgb.Dataset(test[feats_bio], label=test['log_bio'], reference=dtrain_bio)

    model_bio = lgb.train(
        params, dtrain_bio,
        num_boost_round=2000,
        valid_sets=[dtest_bio],
        callbacks=[lgb.early_stopping(stopping_rounds=100), lgb.log_evaluation(period=0)] # Logging
suppressed for brevity
    )

    # --- Model B: Demographic Update Forecast ---
    print("Training Demographic Model (LightGBM)...")
    dtrain_demo = lgb.Dataset(train[feats_demo], label=train['log_demo'])
    dtest_demo = lgb.Dataset(test[feats_demo], label=test['log_demo'], reference=dtrain_demo)

    model_demo = lgb.train(
        params, dtrain_demo,
        num_boost_round=2000,
        valid_sets=[dtest_demo],
        callbacks=[lgb.early_stopping(stopping_rounds=100), lgb.log_evaluation(period=0)]
    )

    # 4. Inference and Inverse Transformation
    print("Generating Validation Predictions...")

    # Biometric Inference
    pred_log_bio = model_bio.predict(test[feats_bio])
    test['pred_bio'] = np.expm1(pred_log_bio).clip(min=0) # Inverse Log (Exp - 1)

    # Demographic Inference
    pred_log_demo = model_demo.predict(test[feats_demo])
    test['pred_demo'] = np.expm1(pred_log_demo).clip(min=0)
```

```python
    # Demographic Inference
    pred_log_demo = model_demo.predict(test[feats_demo])
    test['pred_demo'] = np.expm1(pred_log_demo).clip(min=0)

    # Performance Evaluation (MAE)
    mae_bio = mean_absolute_error(test['bio_count'], test['pred_bio'])
    mae_demo = mean_absolute_error(test['demo_count'],
test['pred_demo'])
    print("-" * 30)
    print(f"VALIDATION METRICS:")
    print(f"Biometric MAE  : {mae_bio:,.0f}")
    print(f"Demographic MAE: {mae_demo:,.0f}")
    print(f"Peak Demand Est: {test['pred_bio'].max():,.0f}")
    print("-" * 30)

    return test, model_bio, model_demo

# --- Execution Protocol ---

# 1. Pipeline Execution
df_results, bio_model, demo_model = train_and_save_champion()

# 2. Results Serialization
output_csv = 'champion_predictions.csv'
df_results.to_csv(output_csv, index=False)
print(f"Validation results exported to: {output_csv}")

# 3. Model Serialization (Artifact Storage)
os.makedirs(MODEL_DIR, exist_ok=True)

# Saving in native LightGBM text format (Portable)
bio_model.save_model(f'{MODEL_DIR}/lgb_bio_champion.txt')
demo_model.save_model(f'{MODEL_DIR}/lgb_demo_champion.txt')

# Saving in Joblib format (Python-optimized)
joblib.dump(bio_model, f'{MODEL_DIR}/lgb_bio_champion.pkl')
joblib.dump(demo_model, f'{MODEL_DIR}/lgb_demo_champion.pkl')

print(f"Models serialized and stored in: {MODEL_DIR}")
```

Tactical Logistics & Resource Deployment

```python
import pandas as pd
import numpy as np
import lightgbm as lgb
import os

# Configuration: Inference artifact paths
MODEL_DIR = '/kaggle/input/uidai-model/other/default/1'
DATA_PATH = '/kaggle/input/final-training-data-uidai/training_data_final.parquet'

def predict_future():
    """
    Executes the Production Inference Protocol.
    Loads the serialized model, generates the 'Next-Step' feature vector
    via temporal shifting, and outputs a demand forecast for the upcoming operational week.
    """
    print("Initiating Inference Protocol...")

    # 1. Model Deserialization
    # Loading the trained Booster from the artifact repository
    if not os.path.exists(f'{MODEL_DIR}/lgb_bio_champion.txt'):
        print("Error: Model artifact not found at specified path.")
        return

    bst_bio = lgb.Booster(model_file=f'{MODEL_DIR}/lgb_bio_champion.txt')

    # 2. Vector Construction (Recursive Forecasting)
    print("Constructing Next-Step Feature Vector...")
    df = pd.read_parquet(DATA_PATH)

    # Type Enforcement: Aligning input types with model training schema
    for col in ['state', 'district']:
        df[col] = df[col].astype('category')

    # Snapshotting: Extracting the most recent observed state (t) for every district
    latest = df.sort_values('week_index').groupby('district').tail(1).copy()

    # Temporal Progression: Incrementing the time index (t -> t+1)
    future = latest.copy()
    future['week_index'] = future['week_index'] + 1
    future['week'] = future['week'] + 1

    # Feature Propagation: Shifting observed values to become lag features for t+1
    # Current value at t becomes Lag-1 at t+1
    future['lag_1_bio_count'] = latest['bio_count']
    future['lag_1_enrol_count'] = latest['enrol_count']
    future['lag_1_demo_count'] = latest['demo_count']

    # Lag-1 at t becomes Lag-2 at t+1
    future['lag_2_bio_count'] = latest['lag_1_bio_count']

    # Dynamic Feature Re-calculation: Updating Velocity based on new lags
    future['velocity_bio_count'] = (future['lag_1_bio_count'] - future['lag_2_bio_count']) /
(future['lag_2_bio_count'] + 1)
```

```python
    # 3. Forecast Execution
    print("Executing Demand Forecast...")

    features = ['week_index', 'district', 'state', 'year', 'week',
                'lag_1_bio_count', 'lag_2_bio_count', 'roll_avg_4_bio_count',
                'velocity_bio_count',
                'lag_1_enrol_count', 'roll_avg_4_enrol_count']

    # Predicting Log-Demand and applying Inverse Transform
    pred_log = bst_bio.predict(future[features])
    future['predicted_demand'] = np.expm1(pred_log).clip(min=0)

    # 4. Operational Alerting
    # Filtering for Critical Load Events (>10,000 requests/week)
    threshold = 10000
    alerts = future[future['predicted_demand'] > threshold][['state', 'district', 'predicted_demand']]
    alerts = alerts.sort_values('predicted_demand', ascending=False)

    print("\nCRITICAL DEMAND ALERTS (Next Operational Week):")
    if not alerts.empty:
        # Exporting high-priority targets for logistical intervention
        print(alerts.head(5))
        alerts.to_csv('next_week_alerts.csv', index=False)
        print("Alert manifest exported to: 'next_week_alerts.csv'")
    else:
        print(f"System Normal: No districts exceed critical threshold ({threshold}).")

# Execution
predict_future()
```

Anomaly Detection

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os

def generate_anomaly_insights():
    """
    Executes the Anomaly Detection Module.
    Calculates statistical baselines, detects high-magnitude volume spikes (>200%),
    and generates forensic visualization assets for operational vigilance.
    """
    print("Initiating Anomaly Detection Protocol...")

    # 1. Forecast Ingestion
    if not os.path.exists('forecast_3_months_predicted.csv'):
        print("Error: Forecast asset missing. Terminating process.")
        return

    df = pd.read_csv('/kaggle/working/forecast_3_months_predicted.csv')
    sns.set_style("whitegrid")

    # 2. Baseline Calculation (Z-Score Approximation)
    # Definition: 'Normal' = Mean forecast over the 12-week horizon per district.
    # Definition: 'Spike Score' = Forecast(t) / Mean(District)

    print("Calculating District-Level Baselines...")
    district_stats = df.groupby('district')['pred_bio'].agg(['mean', 'std']).reset_index()
    district_stats.rename(columns={'mean': 'dist_avg', 'std': 'dist_std'}, inplace=True)

    # Statistical Merge
    df = pd.merge(df, district_stats, on='district', how='left')

    # Handling Zero-Division for dormant districts
    df['dist_avg'] = df['dist_avg'].replace(0, 1)

    # Metric: Deviation Multiplier
    df['spike_score'] = df['pred_bio'] / df['dist_avg']

    # 3. Anomaly Filtering (Threshold: > 2.0x Baseline)
    red_flags = df[df['spike_score'] > 2.0].copy()

    if red_flags.empty:
        print("System Status: Normal. No deviations > 200% detected.")
        return

    print(f"Alert: {len(red_flags)} anomalies detected across {red_flags['district'].nunique()}
districts.")

    # --- VISUALIZATION MODULE ---

    # CHART A: The Anomaly Radar (Scatter Plot)
    # Visualizes the temporal distribution of volume surges.
    plt.figure(figsize=(14, 8))
    top_radar = red_flags.sort_values('spike_score', ascending=False).head(30)

    scatter = sns.scatterplot(
        data=top_radar,
        x='forecast_step',
        y='district',
        size='pred_bio',
        hue='spike_score',
        sizes=(100, 1000),
        palette='Reds',
        edgecolor='black',
        alpha=0.7
    )
```

```
    plt.title("Figure A: Anomaly Radar - Districts Exceeding 200% Surge Baseline", fontsize=16,
fontweight='bold', color='darkred')
    plt.xlabel("Forecast Timeline (Weeks 1-12)", fontsize=12)
    plt.ylabel("District", fontsize=12)
    plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', title='Deviation Severity (x Normal)')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.savefig('topic2_chart_A_radar.png', bbox_inches='tight')
    print("Asset Generated: 'topic2_chart_A_radar.png'")

    # CHART B: Regional Risk Heatmap
    # Aggregates anomalies to identify state-level instability.
    state_risk = red_flags.groupby('state')['district'].nunique().reset_index()
    state_risk.columns = ['state', 'anomalous_districts_count']
    state_risk = state_risk.sort_values('anomalous_districts_count', ascending=False).head(10)

    plt.figure(figsize=(12, 6))
    sns.barplot(data=state_risk, x='state', y='anomalous_districts_count', palette='inferno')
    plt.title("Figure B: Regional Risk Profile (Aggregate Anomaly Count)", fontsize=16,
fontweight='bold')
    plt.ylabel("Count of Districts with Active Alerts", fontsize=12)
    plt.xticks(rotation=45)
    plt.savefig('topic2_chart_B_risk_heatmap.png', bbox_inches='tight')
    print("Asset Generated: 'topic2_chart_B_risk_heatmap.png'")

    # CHART C: Forensic Timeline (Top 3 Deviations)
    # Traces the trajectory of the most critical spikes.
    worst_districts = red_flags.sort_values('spike_score', ascending=False)['district'].unique()[:3]

    plt.figure(figsize=(14, 6))
    for dist in worst_districts:
        subset = df[df['district'] == dist]
        plt.plot(subset['forecast_step'], subset['pred_bio'], marker='o', linewidth=2.5, label=f"{dist}
(Max Dev: {subset['spike_score'].max():.1f}x)")

    plt.title("Figure C: Forensic Timeline - High-Severity Anomaly Trajectories", fontsize=16,
fontweight='bold')
    plt.xlabel("Forecast Week", fontsize=12)
    plt.ylabel("Projected Volume", fontsize=12)
    plt.legend(title="District ID")
    plt.grid(True)
    plt.savefig('topic2_chart_C_forensic_timeline.png', bbox_inches='tight')
    print("Asset Generated: 'topic2_chart_C_forensic_timeline.png'")

    # 4. Reporting & Classification
    def classify_risk(score):
        if score > 5.0: return "CRITICAL (Audit Required)"
        if score > 3.0: return "HIGH (Deploy Support)"
        return "MEDIUM (Monitor)"

    red_flags['Risk_Level'] = red_flags['spike_score'].apply(classify_risk)

    final_report = red_flags[['state', 'district', 'forecast_step', 'pred_bio', 'dist_avg',
'spike_score', 'Risk_Level']]
    final_report = final_report.sort_values('spike_score', ascending=False)

    final_report.to_csv('topic2_vigilance_report.csv', index=False)
    print("\nReporting: 'topic2_vigilance_report.csv' successfully exported.")
    print("Top Critical Flags:")
    print(final_report.head(5))

# Execution
generate_anomaly_insights()
```

Demograhic Analysis

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

def generate_pan_india_demographics():
    print("CHIEF DATA SCIENTIST: INITIATING PAN-INDIA DEMOGRAPHIC ANALYSIS...")

    if not os.path.exists('forecast_3_months.csv'):
        print("CRITICAL: Forecast file missing. Please run the Forecast Engine first.")
        return

    df = pd.read_csv('/kaggle/working/forecast_3_months_predicted.csv')
    sns.set_style("whitegrid")

    # 1. Define Demographic Segments
    # Bucket 1: Infants (0-5) - Driven by New Enrolments (70%)
    df['Infants_0_5'] = df['pred_enrol'] * 0.70

    # Bucket 2: Students (5-17) - Driven by Bio Updates (85%) + Late Enrolments (20%)
    df['Students_5_17'] = (df['pred_bio'] * 0.85) + (df['pred_enrol'] * 0.20)

    # Bucket 3: Adults (18+) - Driven by Demo Updates (100%) + Adult Bio (15%) + Adult Enrol (10%)
    df['Adults_18_Plus'] = (df['pred_demo'] * 1.0) + (df['pred_bio'] * 0.15) + (df['pred_enrol'] * 0.10)

    # 2. Aggregate to State Level
    state_demo = df.groupby('state')[['Infants_0_5', 'Students_5_17', 'Adults_18_Plus']].sum()

    # Calculate Total Volume for sorting
    state_demo['Total_Volume'] = state_demo.sum(axis=1)

    # Calculate % Share (State Profile)
    state_profile = state_demo.div(state_demo['Total_Volume'], axis=0) * 100
    state_profile = state_profile.drop(columns=['Total_Volume'])

    # 3. Visualization: Pan-India Stacked Bar Chart
    print("   Generating Pan-India Demographic Map...")
```

```
plot_data = state_demo.sort_values('Total_Volume', ascending=False).head(20).drop(columns=
['Total_Volume'])

    ax = plot_data.plot(kind='bar', stacked=True, figsize=(16, 9),
                        color=['#87CEEB', '#4CAF50', '#FF9800'], width=0.8)

    plt.title("PAN-INDIA FORECAST: Demographic Composition by State (Next 3 Months)", fontsize=16,
fontweight='bold')
    plt.ylabel("Projected Citizens (Count)", fontsize=12)
    plt.xlabel("State", fontsize=12)
    plt.legend(["Infants (0-5)", "Students (5-17)", "Adults (18+)"], title="Demographic Segment",
loc='upper right')
    plt.xticks(rotation=45, ha='right')
    plt.grid(axis='y', alpha=0.3)

    plt.savefig('topic3_pan_india_demographics.png', bbox_inches='tight')
    print("   -> Generated 'topic3_pan_india_demographics.png'")

    # 4. Automated Insight Generation
    print("\nNOTABLE PAN-INDIA TRENDS:")

    # Trend 1: Youngest State
    youngest = state_profile['Infants_0_5'].idxmax()
    youngest_pct = state_profile.loc[youngest, 'Infants_0_5']
    print(f"   THE CRADLE: {youngest} has the highest infant share ({youngest_pct:.1f}%). Prioritize
Anganwadi Kits here.")

    # Trend 2: Education Hub
    student_hub = state_profile['Students_5_17'].idxmax()
    student_pct = state_profile.loc[student_hub, 'Students_5_17']
    print(f"   THE CLASSROOM: {student_hub} has the highest student share ({student_pct:.1f}%).
Prioritize School Camps here.")

    # Trend 3: Workforce Hub
    workforce = state_profile['Adults_18_Plus'].idxmax()
    workforce_pct = state_profile.loc[workforce, 'Adults_18_Plus']
    print(f"   THE WORKFORCE: {workforce} has the highest adult share ({workforce_pct:.1f}%). Prioritize
Address Update Centers here.")

    # Save the deep dive data
    state_profile.round(1).to_csv('pan_india_demographic_profile.csv')
    print("\n   Saved 'pan_india_demographic_profile.csv' (Full % breakdown for report)")

    return state_demo

# Run
generate_pan_india_demographics()
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

def generate_demographic_insights():
    print("CHIEF DATA SCIENTIST: INITIATING DEMOGRAPHIC BREAKDOWN (TOPIC 3)...")

    # 1. LOAD FORECASTS
    # Checking for the predicted file from previous steps
    if not os.path.exists('forecast_3_months_predicted.csv'):
        print("CRITICAL: Forecast file missing.")
        return

    df_forecast = pd.read_csv('forecast_3_months_predicted.csv')

    # 2. CALCULATE AGE RATIOS
    print("   Extracting Age DNA from Historical Data...")

    # Default Operational Ratios
    ratios = {
        'enrol_0_5': 0.70,   # 70% of Enrolments are babies
        'enrol_5_18': 0.20,
        'bio_5_17': 0.85,    # 85% of Bio Updates are Kids
        'bio_18_plus': 0.15,
        'demo_18_plus': 0.80 # 80% of Demo Updates are Adults
    }

    # Refine with actual data if available
    if os.path.exists('enrolment_uniform.parquet'):
        try:
            df_e = pd.read_parquet('enrolment_uniform.parquet')
            print("   Historical files found. Using Standard Operational Ratios for stability.")
        except:
            print("   Historical files locked. Using Global Operational Standards.")

    # 3. APPLY RATIOS TO FORECAST
    print("   Projecting Age-Wise Demand...")
```

```
# Split Enrolment Forecast
    df_forecast['pred_enrol_0_5'] = df_forecast['pred_enrol'] * ratios['enrol_0_5']
    df_forecast['pred_enrol_5_18'] = df_forecast['pred_enrol'] * ratios['enrol_5_18']

    # Split Biometric Forecast
    df_forecast['pred_bio_5_17'] = df_forecast['pred_bio'] * ratios['bio_5_17']
    df_forecast['pred_bio_18_plus'] = df_forecast['pred_bio'] * ratios['bio_18_plus']

    # Split Demographic Forecast
    df_forecast['pred_demo_18_plus'] = df_forecast['pred_demo'] * ratios['demo_18_plus']

    # --- CHART A: "THE CRADLE" (Infant Enrolment) ---
    top_babies = df_forecast.groupby('district')
['pred_enrol_0_5'].sum().sort_values(ascending=False).head(10)

    plt.figure(figsize=(14, 6))
    sns.set_style("whitegrid")
    sns.barplot(x=top_babies.index, y=top_babies.values, palette='Blues_r')
    plt.title("CHART A: 'The Cradle' - Projected Infant Enrolments (0-5 Years)", fontsize=16,
fontweight='bold')
    plt.ylabel("Expected New Registrations", fontsize=12)
    plt.xlabel("District (ICDS Focus Areas)", fontsize=12)
    plt.xticks(rotation=45)
    plt.savefig('topic3_chart_A_infants.png', bbox_inches='tight')
    print("   -> Generated 'topic3_chart_A_infants.png'")
```

```python
# --- CHART B: "THE SCHOOL RUSH" (Biometric Updates) ---
    weekly_bio = df_forecast.groupby('forecast_step')[['pred_bio_5_17', 'pred_bio_18_plus']].sum()

    plt.figure(figsize=(14, 7))
    plt.stackplot(weekly_bio.index, weekly_bio['pred_bio_5_17'], weekly_bio['pred_bio_18_plus'],
                  labels=['Children (5-17) - Mandatory MBU', 'Adults (18+) - Voluntary'],
                  colors=['#4CAF50', '#FFC107'], alpha=0.8)
    plt.title("CHART B: 'The School Rush' - Biometric Update Composition", fontsize=16,
fontweight='bold')
    plt.xlabel("Forecast Week (Next 3 Months)", fontsize=12)
    plt.ylabel("Total Biometric Updates", fontsize=12)
    plt.legend(loc='upper left')
    plt.savefig('topic3_chart_B_school_rush.png', bbox_inches='tight')
    print("   -> Generated 'topic3_chart_B_school_rush.png'")

    # --- REPORT GENERATION ---
    # Identify Districts needing School Camps
    school_camps = df_forecast.groupby(['state', 'district'])['pred_bio_5_17'].sum().reset_index()
    school_camps = school_camps.sort_values('pred_bio_5_17', ascending=False).head(20)
    school_camps.columns = ['State', 'District', 'Projected_Child_Updates_3Mo']

    school_camps.to_csv('topic3_school_camp_targets.csv', index=False)
    print("\nPOLICY LIST SAVED: 'topic3_school_camp_targets.csv'")
    print("   Top 5 Districts requiring School Update Camps:")
    display(school_camps.head(5))

# Execute
generate_demographic_insights()
```