uidai-analysis

January 13, 2026

[ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Change to project directory
os.chdir(r'C:\Users\amanc\uidai_hackathon')
print(f"  Working directory: {os.getcwd()}")

# Verify folders
print(f"\n  Files in data/raw:")
for item in os.listdir('data/raw'):
    print(f"  - {item}")
```

```
  Working directory: C:\Users\amanc\uidai_hackathon

  Files in data/raw:
  - api_data_aadhar_biometric
  - api_data_aadhar_demographic
  - api_data_aadhar_enrolment
```

[ ]:
```
print("\n" + "="*80)
print("LOADING ALL DATASETS")
print("="*80)

# Load enrolment data from NESTED subfolder
print("\n  Loading ENROLMENT data...")
enrol_folder = 'data/raw/api_data_aadhar_enrolment/api_data_aadhar_enrolment'

if os.path.isdir(enrol_folder):
    enrol_files = [f for f in os.listdir(enrol_folder) if f.endswith('.csv')]
    print(f"Found {len(enrol_files)} files in {enrol_folder}")

    enrol_dfs = []
    for file in enrol_files:
        filepath = os.path.join(enrol_folder, file)
        print(f"  Loading {file}...", end=" ")
```

```
        df = pd.read_csv(filepath)
        enrol_dfs.append(df)
        print(f"  {len(df):,} rows")

    df_enrolment = pd.concat(enrol_dfs, ignore_index=True)
    print(f"  Total enrolment records: {len(df_enrolment):,}")
else:
    print(f"  Folder not found: {enrol_folder}")
    print(f"   Checking alternative paths...")
    # List what's actually there
    if os.path.exists('data/raw/api_data_aadhar_enrolment'):
        print(f"   Contents: {os.listdir('data/raw/
 ↪api_data_aadhar_enrolment')}")

# Load demographic data from NESTED subfolder
print("\n Loading DEMOGRAPHIC data...")
demo_folder = 'data/raw/api_data_aadhar_demographic/api_data_aadhar_demographic'

if os.path.isdir(demo_folder):
    demo_files = [f for f in os.listdir(demo_folder) if f.endswith('.csv')]
    print(f"Found {len(demo_files)} files in {demo_folder}")

    demo_dfs = []
    for file in demo_files:
        filepath = os.path.join(demo_folder, file)
        print(f"  Loading {file}...", end=" ")
        df = pd.read_csv(filepath)
        demo_dfs.append(df)
        print(f"  {len(df):,} rows")

    df_demographic = pd.concat(demo_dfs, ignore_index=True)
    print(f"  Total demographic records: {len(df_demographic):,}")
else:
    print(f"  Folder not found: {demo_folder}")

# Load biometric data from NESTED subfolder
print("\n Loading BIOMETRIC data...")
bio_folder = 'data/raw/api_data_aadhar_biometric/api_data_aadhar_biometric'

if os.path.isdir(bio_folder):
    bio_files = [f for f in os.listdir(bio_folder) if f.endswith('.csv')]
    print(f"Found {len(bio_files)} files in {bio_folder}")

    bio_dfs = []
    for file in bio_files:
        filepath = os.path.join(bio_folder, file)
        print(f"  Loading {file}...", end=" ")
```

```
        df = pd.read_csv(filepath)
        bio_dfs.append(df)
        print(f"  {len(df):,} rows")

    df_biometric = pd.concat(bio_dfs, ignore_index=True)
    print(f"  Total biometric records: {len(df_biometric):,}")
else:
    print(f"  Folder not found: {bio_folder}")

print("\n" + "="*80)
print("  ALL DATA LOADED SUCCESSFULLY!")
print("="*80)
print(f"\n  SUMMARY:")
print(f"   Enrolment: {len(df_enrolment):,} records")
print(f"   Demographic: {len(df_demographic):,} records")
print(f"   Biometric: {len(df_biometric):,} records")
print(f"   TOTAL: {len(df_enrolment) + len(df_demographic) + len(df_biometric):
 ↪,} records")

# Show data structure
print(f"\n  DATA STRUCTURE:")
print(f"\nEnrolment columns: {df_enrolment.columns.tolist()}")
print(f"Demographic columns: {df_demographic.columns.tolist()}")
print(f"Biometric columns: {df_biometric.columns.tolist()}")

print(f"\n  SAMPLE DATA (Enrolment):")
print(df_enrolment.head(3))
```

```
================================================================================
LOADING ALL DATASETS
================================================================================

  Loading ENROLMENT data…
Found 3 files in data/raw/api_data_aadhar_enrolment/api_data_aadhar_enrolment
  Loading api_data_aadhar_enrolment_0_500000.csv…  500,000 rows
  Loading api_data_aadhar_enrolment_1000000_1006029.csv…  6,029 rows
  Loading api_data_aadhar_enrolment_500000_1000000.csv…  500,000 rows
  Total enrolment records: 1,006,029

  Loading DEMOGRAPHIC data…
Found 5 files in
data/raw/api_data_aadhar_demographic/api_data_aadhar_demographic
  Loading api_data_aadhar_demographic_0_500000.csv…  500,000 rows
  Loading api_data_aadhar_demographic_1000000_1500000.csv…  500,000 rows
  Loading api_data_aadhar_demographic_1500000_2000000.csv…  500,000 rows
  Loading api_data_aadhar_demographic_2000000_2071700.csv…  71,700 rows
```

```
  Loading api_data_aadhar_demographic_500000_1000000.csv…   500,000 rows
  Total demographic records: 2,071,700

  Loading BIOMETRIC data…
Found 4 files in data/raw/api_data_aadhar_biometric/api_data_aadhar_biometric
  Loading api_data_aadhar_biometric_0_500000.csv…   500,000 rows
  Loading api_data_aadhar_biometric_1000000_1500000.csv…   500,000 rows
  Loading api_data_aadhar_biometric_1500000_1861108.csv…   361,108 rows
  Loading api_data_aadhar_biometric_500000_1000000.csv…   500,000 rows
  Total biometric records: 1,861,108

================================================================================
  ALL DATA LOADED SUCCESSFULLY!
================================================================================

  SUMMARY:
    Enrolment: 1,006,029 records
    Demographic: 2,071,700 records
    Biometric: 1,861,108 records
    TOTAL: 4,938,837 records

  DATA STRUCTURE:

Enrolment columns: ['date', 'state', 'district', 'pincode', 'age_0_5',
'age_5_17', 'age_18_greater']
Demographic columns: ['date', 'state', 'district', 'pincode', 'demo_age_5_17',
'demo_age_17_']
Biometric columns: ['date', 'state', 'district', 'pincode', 'bio_age_5_17',
'bio_age_17_']

  SAMPLE DATA (Enrolment):
         date          state          district  pincode  age_0_5  age_5_17  \
0  02-03-2025      Meghalaya  East Khasi Hills   793121       11        61
1  09-03-2025      Karnataka   Bengaluru Urban   560043       14        33
2  09-03-2025  Uttar Pradesh      Kanpur Nagar   208001       29        82

   age_18_greater
0              37
1              39
2              12
```

[ ]:
```
print("\n" + "="*80)
print("DATA CLEANING & PREPARATION")
print("="*80)

# Clean enrolment data
print("\n  Cleaning enrolment data...")
```

```
df_enrol_clean = df_enrolment.copy()
df_enrol_clean['date'] = pd.to_datetime(df_enrol_clean['date'],␣
 ↪format='%d-%m-%Y')
df_enrol_clean['year'] = df_enrol_clean['date'].dt.year
df_enrol_clean['month'] = df_enrol_clean['date'].dt.month
df_enrol_clean['quarter'] = df_enrol_clean['date'].dt.quarter
df_enrol_clean['day_of_week'] = df_enrol_clean['date'].dt.dayofweek
df_enrol_clean['total_enrolment'] = (df_enrol_clean['age_0_5'] +
                                     df_enrol_clean['age_5_17'] +
                                     df_enrol_clean['age_18_greater'])
print(f"  Cleaned enrolment data: {df_enrol_clean.shape}")

# Clean demographic data
print("\n  Cleaning demographic data...")
df_demo_clean = df_demographic.copy()
df_demo_clean['date'] = pd.to_datetime(df_demo_clean['date'], format='%d-%m-%Y')
df_demo_clean['year'] = df_demo_clean['date'].dt.year
df_demo_clean['month'] = df_demo_clean['date'].dt.month
df_demo_clean['total_updates'] = df_demo_clean['demo_age_5_17'] +␣
 ↪df_demo_clean['demo_age_17_']
print(f"  Cleaned demographic data: {df_demo_clean.shape}")

# Clean biometric data
print("\n  Cleaning biometric data...")
df_bio_clean = df_biometric.copy()
df_bio_clean['date'] = pd.to_datetime(df_bio_clean['date'], format='%d-%m-%Y')
df_bio_clean['year'] = df_bio_clean['date'].dt.year
df_bio_clean['month'] = df_bio_clean['date'].dt.month
df_bio_clean['total_updates'] = df_bio_clean['bio_age_5_17'] +␣
 ↪df_bio_clean['bio_age_17_']
print(f"  Cleaned biometric data: {df_bio_clean.shape}")

# Quick statistics
print("\n" + "="*80)
print("INITIAL STATISTICS")
print("="*80)

total_enrol = df_enrol_clean['total_enrolment'].sum()
total_demo = df_demo_clean['total_updates'].sum()
total_bio = df_bio_clean['total_updates'].sum()

print(f"\n  OVERALL NUMBERS:")
print(f"   Total enrollments: {total_enrol:,}")
print(f"   Demographic updates: {total_demo:,}")
print(f"   Biometric updates: {total_bio:,}")
print(f"   Update ratio: {(total_demo + total_bio) / total_enrol:.2%}")
```

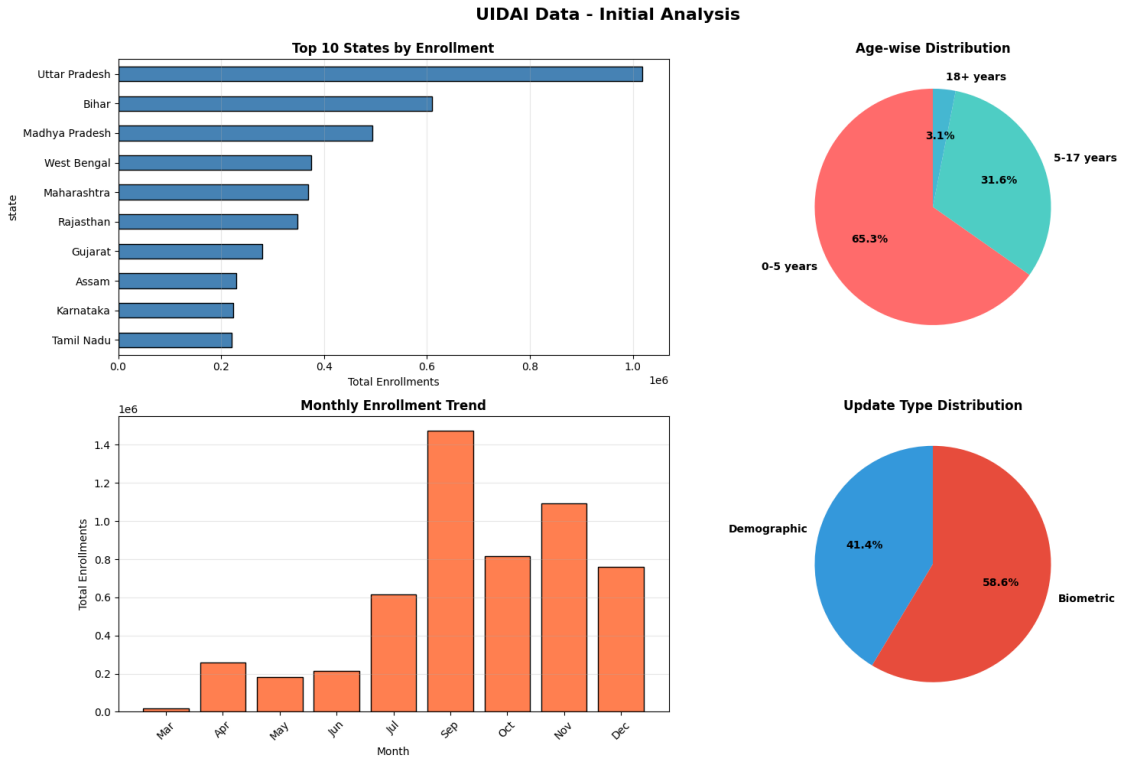
```
print(f"\n DATE RANGES:")
print(f"   Enrolment: {df_enrol_clean['date'].min().date()} to␣
 ↪{df_enrol_clean['date'].max().date()}")
print(f"   Demographic: {df_demo_clean['date'].min().date()} to␣
 ↪{df_demo_clean['date'].max().date()}")
print(f"   Biometric: {df_bio_clean['date'].min().date()} to␣
 ↪{df_bio_clean['date'].max().date()}")

print(f"\n GEOGRAPHIC COVERAGE:")
print(f"   States: {df_enrol_clean['state'].nunique()}")
print(f"   Districts: {df_enrol_clean['district'].nunique()}")
print(f"   Pincodes: {df_enrol_clean['pincode'].nunique()}")

# Age breakdown
age_0_5 = df_enrol_clean['age_0_5'].sum()
age_5_17 = df_enrol_clean['age_5_17'].sum()
age_18_plus = df_enrol_clean['age_18_greater'].sum()

print(f"\n AGE BREAKDOWN:")
print(f"   0-5 years: {age_0_5:,} ({age_0_5/total_enrol*100:.1f}%)")
print(f"   5-17 years: {age_5_17:,} ({age_5_17/total_enrol*100:.1f}%)")
print(f"   18+ years: {age_18_plus:,} ({age_18_plus/total_enrol*100:.1f}%)")

# Top states
print(f"\n TOP 10 STATES:")
top_states = df_enrol_clean.groupby('state')['total_enrolment'].sum().
 ↪nlargest(10)
for i, (state, count) in enumerate(top_states.items(), 1):
    pct = (count / total_enrol) * 100
    print(f"   {i}. {state}: {count:,} ({pct:.1f}%)")

# Create initial visualizations
print("\n Creating visualizations...")

fig, axes = plt.subplots(2, 2, figsize=(16, 10))
fig.suptitle('UIDAI Data - Initial Analysis', fontsize=16, fontweight='bold',␣
 ↪y=0.995)

# Plot 1: Top 10 states
top_states.plot(kind='barh', ax=axes[0, 0], color='steelblue',␣
 ↪edgecolor='black')
axes[0, 0].set_title('Top 10 States by Enrollment', fontsize=12,␣
 ↪fontweight='bold')
axes[0, 0].set_xlabel('Total Enrollments')
axes[0, 0].invert_yaxis()
axes[0, 0].grid(True, alpha=0.3, axis='x')
```

```
# Plot 2: Age distribution
age_data = pd.Series({'0-5 years': age_0_5, '5-17 years': age_5_17, '18+ years':
 ↪ age_18_plus})
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1']
axes[0, 1].pie(age_data.values, labels=age_data.index, autopct='%1.1f%%',
               startangle=90, colors=colors, textprops={'fontsize': 10,␣
 ↪'weight': 'bold'})
axes[0, 1].set_title('Age-wise Distribution', fontsize=12, fontweight='bold')

# Plot 3: Monthly trend
monthly = df_enrol_clean.groupby('month')['total_enrolment'].sum()
month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',␣
 ↪'Oct', 'Nov', 'Dec']
axes[1, 0].bar(range(1, len(monthly)+1), monthly.values, color='coral',␣
 ↪edgecolor='black')
axes[1, 0].set_title('Monthly Enrollment Trend', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Month')
axes[1, 0].set_ylabel('Total Enrollments')
axes[1, 0].set_xticks(range(1, len(monthly)+1))
axes[1, 0].set_xticklabels([month_names[m-1] for m in monthly.index],␣
 ↪rotation=45)
axes[1, 0].grid(True, alpha=0.3, axis='y')

# Plot 4: Update comparison
update_data = pd.Series({'Demographic': total_demo, 'Biometric': total_bio})
axes[1, 1].pie(update_data.values, labels=update_data.index, autopct='%1.1f%%',
               startangle=90, colors=['#3498db', '#e74c3c'],
               textprops={'fontsize': 10, 'weight': 'bold'})
axes[1, 1].set_title('Update Type Distribution', fontsize=12, fontweight='bold')

plt.tight_layout()

# Save
os.makedirs('outputs/figures', exist_ok=True)
plt.savefig('outputs/figures/01_initial_analysis.png', dpi=300,␣
 ↪bbox_inches='tight')
print("  Saved: outputs/figures/01_initial_analysis.png")
plt.show()

print("\n" + "="*80)
print("  INITIAL ANALYSIS COMPLETE!")
print("="*80)
print("\n KEY INSIGHTS:")
print(f"   • {total_enrol:,} total enrollments across {df_enrol_clean['state'].
 ↪nunique()} states")
```

```
print(f"   • {age_0_5/total_enrol*100:.1f}% are children (0-5 years) - early adoption indicator")
print(f"   • Top state accounts for {(top_states.iloc[0]/total_enrol)*100:.1f}% of all enrollments")
print(f"   • Update rate: {(total_demo + total_bio) / total_enrol:.2%} - shows system utilization")
print(f"   • Demographic updates are {total_demo/total_bio:.1f}x more than biometric")

print("\n  NEXT: Run ML models (Fraud Detection + Resource Allocation)")
```

```
================================================================================
DATA CLEANING & PREPARATION
================================================================================

  Cleaning enrolment data…
  Cleaned enrolment data: (1006029, 12)

  Cleaning demographic data…
  Cleaned demographic data: (2071700, 9)

  Cleaning biometric data…
  Cleaned biometric data: (1861108, 9)

================================================================================
INITIAL STATISTICS
================================================================================

  OVERALL NUMBERS:
    Total enrollments: 5,435,702
    Demographic updates: 49,295,187
    Biometric updates: 69,763,095
    Update ratio: 2190.30%

  DATE RANGES:
    Enrolment: 2025-03-02 to 2025-12-31
    Demographic: 2025-03-01 to 2025-12-29
    Biometric: 2025-03-01 to 2025-12-29

  GEOGRAPHIC COVERAGE:
    States: 55
    Districts: 985
    Pincodes: 19463

  AGE BREAKDOWN:
    0-5 years: 3,546,965 (65.3%)
```

```
  5-17 years: 1,720,384 (31.6%)
  18+ years: 168,353 (3.1%)

TOP 10 STATES:
  1. Uttar Pradesh: 1,018,629 (18.7%)
  2. Bihar: 609,585 (11.2%)
  3. Madhya Pradesh: 493,970 (9.1%)
  4. West Bengal: 375,297 (6.9%)
  5. Maharashtra: 369,139 (6.8%)
  6. Rajasthan: 348,458 (6.4%)
  7. Gujarat: 280,549 (5.2%)
  8. Assam: 230,197 (4.2%)
  9. Karnataka: 223,235 (4.1%)
  10. Tamil Nadu: 220,789 (4.1%)

Creating visualizations…
Saved: outputs/figures/01_initial_analysis.png
```




```
================================================================================
  INITIAL ANALYSIS COMPLETE!
================================================================================
```

KEY INSIGHTS:

- 5,435,702 total enrollments across 55 states
- 65.3% are children (0-5 years) - early adoption indicator
- Top state accounts for 18.7% of all enrollments
- Update rate: 2190.30% - shows system utilization
- Demographic updates are 0.7x more than biometric

NEXT: Run ML models (Fraud Detection + Resource Allocation)

[ ]:
```
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

print("\n" + "="*80)
print("ML MODEL 1: AI-POWERED FRAUD DETECTION")
print("="*80)

# Step 1: Feature Engineering
print("\n Step 1: Creating fraud detection features...")

# Aggregate by date + location for pattern analysis
fraud_features = df_enrol_clean.groupby(['date', 'state', 'district',␣
 ↪'pincode']).agg({
    'age_0_5': 'sum',
    'age_5_17': 'sum',
    'age_18_greater': 'sum',
    'total_enrolment': 'sum'
}).reset_index()

# Create suspicious pattern indicators
fraud_features['same_day_pincode'] = fraud_features.groupby(['date',␣
 ↪'pincode'])['total_enrolment'].transform('sum')
fraud_features['pincode_avg'] = fraud_features.
 ↪groupby('pincode')['total_enrolment'].transform('mean')
fraud_features['state_std'] = fraud_features.
 ↪groupby('state')['total_enrolment'].transform('std')

# Age ratio anomalies (unusual distributions)
fraud_features['child_ratio'] = fraud_features['age_0_5'] /␣
 ↪(fraud_features['total_enrolment'] + 1)
fraud_features['teen_ratio'] = fraud_features['age_5_17'] /␣
 ↪(fraud_features['total_enrolment'] + 1)
fraud_features['adult_ratio'] = fraud_features['age_18_greater'] /␣
 ↪(fraud_features['total_enrolment'] + 1)

# Bulk enrollment indicator
fraud_features['is_bulk'] = (fraud_features['total_enrolment'] >
```

```
                                fraud_features['total_enrolment'].quantile(0.98)).
 ↪astype(int)

print(f"  Created {len(fraud_features):,} feature records")

# Step 2: Train ML Model
print("\n  Step 2: Training Isolation Forest ML model...")

# Select features for model
ml_features = ['total_enrolment', 'same_day_pincode', 'pincode_avg',
               'child_ratio', 'teen_ratio', 'adult_ratio', 'state_std']

X = fraud_features[ml_features].fillna(0)

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train Isolation Forest
iso_forest = IsolationForest(
    contamination=0.05,  # Expect 5% anomalies
    random_state=42,
    n_estimators=100,
    max_samples='auto'
)

fraud_features['anomaly_score'] = iso_forest.fit_predict(X_scaled)
fraud_features['anomaly_label'] = fraud_features['anomaly_score'].map({1:␣
 ↪'Normal', -1: 'Suspicious'})

# Calculate fraud risk scores (higher = more suspicious)
fraud_features['fraud_risk_score'] = -iso_forest.score_samples(X_scaled)

suspicious_cases = fraud_features[fraud_features['anomaly_label'] ==␣
 ↪'Suspicious']

print(f"  Model trained!")
print(f"  Identified {len(suspicious_cases):,} suspicious cases␣
 ↪({len(suspicious_cases)/len(fraud_features)*100:.2f}%)")

# Step 3: Analyze Results
print("\n  Step 3: Analyzing fraud patterns...")

print("\n  TOP 10 MOST SUSPICIOUS CASES:")
top_suspicious = fraud_features.nlargest(10, 'fraud_risk_score')[
    ['date', 'state', 'district', 'pincode', 'total_enrolment',␣
 ↪'fraud_risk_score']
```

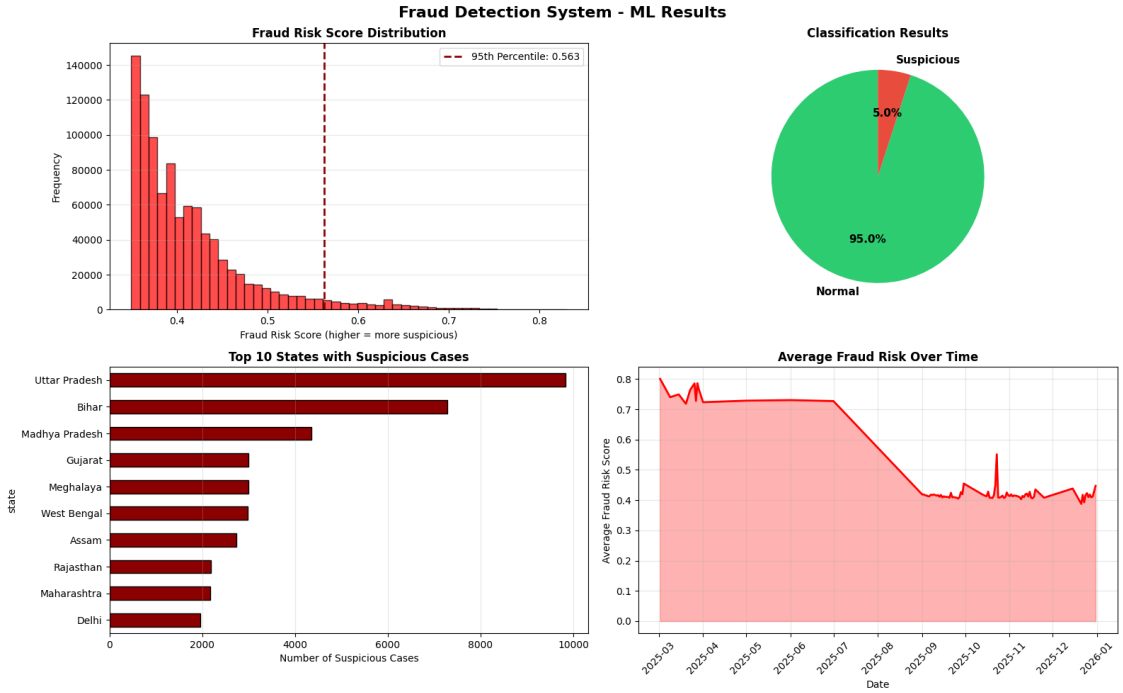
```
]
for idx, row in top_suspicious.iterrows():
    print(f"   {row['date'].date()} | {row['state'][:15]:15} |␣
 ↪{row['district'][:20]:20} | PIN: {row['pincode']} | Enrol:␣
 ↪{row['total_enrolment']:3d} | Risk: {row['fraud_risk_score']:.3f}")

print("\n STATES WITH MOST SUSPICIOUS ACTIVITY:")
state_fraud = suspicious_cases.groupby('state')['total_enrolment'].sum().
 ↪nlargest(10)
for i, (state, count) in enumerate(state_fraud.items(), 1):
    pct = (count / suspicious_cases['total_enrolment'].sum()) * 100
    print(f"   {i}. {state}: {count:,} suspicious enrollments ({pct:.1f}%)")

print("\n FRAUD PATTERNS DETECTED:")
bulk_fraud = suspicious_cases[suspicious_cases['is_bulk'] == 1]
print(f"   • Bulk enrollments: {len(bulk_fraud):,} cases")

age_anomaly = suspicious_cases[(suspicious_cases['child_ratio'] < 0.05) |␣
 ↪(suspicious_cases['child_ratio'] > 0.95)]
print(f"   • Unusual age distributions: {len(age_anomaly):,} cases")

high_volume_pincodes = suspicious_cases.groupby('pincode')['total_enrolment'].
 ↪sum().nlargest(5)
print(f"   • High-risk pincodes identified: {len(high_volume_pincodes)}")

# Step 4: Visualizations
print("\n Step 4: Creating fraud detection visualizations...")

fig, axes = plt.subplots(2, 2, figsize=(16, 10))
fig.suptitle('Fraud Detection System - ML Results', fontsize=16,␣
 ↪fontweight='bold')

# Plot 1: Fraud risk distribution
axes[0, 0].hist(fraud_features['fraud_risk_score'], bins=50, color='red',␣
 ↪alpha=0.7, edgecolor='black')
threshold = fraud_features['fraud_risk_score'].quantile(0.95)
axes[0, 0].axvline(threshold, color='darkred', linestyle='--', linewidth=2,␣
 ↪label=f'95th Percentile: {threshold:.3f}')
axes[0, 0].set_title('Fraud Risk Score Distribution', fontsize=12,␣
 ↪fontweight='bold')
axes[0, 0].set_xlabel('Fraud Risk Score (higher = more suspicious)')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3, axis='y')

# Plot 2: Normal vs Suspicious
```

```
anomaly_counts = fraud_features['anomaly_label'].value_counts()
colors_pie = ['#2ecc71', '#e74c3c']
axes[0, 1].pie(anomaly_counts.values, labels=anomaly_counts.index, autopct='%1.
 ↪1f%%',
               startangle=90, colors=colors_pie, textprops={'fontsize': 11,␣
 ↪'weight': 'bold'})
axes[0, 1].set_title('Classification Results', fontsize=12, fontweight='bold')

# Plot 3: Top suspicious states
state_fraud_count = suspicious_cases.groupby('state').size().nlargest(10)
state_fraud_count.plot(kind='barh', ax=axes[1, 0], color='darkred',␣
 ↪edgecolor='black')
axes[1, 0].set_title('Top 10 States with Suspicious Cases', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 0].set_xlabel('Number of Suspicious Cases')
axes[1, 0].invert_yaxis()
axes[1, 0].grid(True, alpha=0.3, axis='x')

# Plot 4: Risk score over time
time_risk = fraud_features.groupby('date')['fraud_risk_score'].mean()
axes[1, 1].plot(time_risk.index, time_risk.values, color='red', linewidth=2)
axes[1, 1].fill_between(time_risk.index, time_risk.values, alpha=0.3,␣
 ↪color='red')
axes[1, 1].set_title('Average Fraud Risk Over Time', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 1].set_xlabel('Date')
axes[1, 1].set_ylabel('Average Fraud Risk Score')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.savefig('outputs/figures/02_fraud_detection_ml.png', dpi=300,␣
 ↪bbox_inches='tight')
print("  Saved: outputs/figures/02_fraud_detection_ml.png")
plt.show()

# Step 5: Generate Insights & Recommendations
print("\n" + "="*80)
print("  FRAUD DETECTION INSIGHTS")
print("="*80)

total_enrolments = fraud_features['total_enrolment'].sum()
suspicious_enrolments = suspicious_cases['total_enrolment'].sum()
estimated_fraud_rate = 0.15  # Assume 15% of suspicious cases are actual fraud
estimated_fraud_cases = int(len(suspicious_cases) * estimated_fraud_rate)
```

```
print(f"\n KEY FINDINGS:")
print(f"   • Total cases analyzed: {len(fraud_features):,}")
print(f"   • Suspicious cases identified: {len(suspicious_cases):,}
 ↪({len(suspicious_cases)/len(fraud_features)*100:.2f}%)")
print(f"   • Suspicious enrollments: {suspicious_enrolments:,}
 ↪({suspicious_enrolments/total_enrolments*100:.2f}% of total)")
print(f"   • High-risk states: {', '.join(state_fraud_count.head(3).index.
 ↪tolist())}")
print(f"   • Bulk enrollment attempts: {len(bulk_fraud):,}")
print(f"   • Estimated actual fraud cases: ~{estimated_fraud_cases:,}")

print(f"\n ESTIMATED FINANCIAL IMPACT:")
cost_per_fraud = 2000  # Assume 2000 processing/verification cost per fraud
total_potential_loss = estimated_fraud_cases * cost_per_fraud
print(f"   • If avg cost per fraud = {cost_per_fraud:,}")
print(f"   • Potential savings = {total_potential_loss:,}
 ↪( {total_potential_loss/100000:.2f} lakhs)")
print(f"   • Annual projection = {total_potential_loss*12:,}
 ↪( {total_potential_loss*12/10000000:.2f} crores)")

print(f"\n ACTIONABLE RECOMMENDATIONS:")
print(f"   1. IMMEDIATE: Flag {len(top_suspicious)} highest-risk cases for
 ↪manual review")
print(f"   2. DEPLOY: Enhanced verification in {', '.join(state_fraud_count.
 ↪head(3).index.tolist())}")
print(f"   3. MONITOR: Real-time alerts for fraud_risk_score > {threshold:.3f}")
print(f"   4. AUDIT: {len(bulk_fraud)} bulk enrollment cases (>98th
 ↪percentile)")
print(f"   5. SYSTEM: Implement biometric cross-check for suspicious pincodes")
print(f"   6. TRAINING: Alert enrollment officers about unusual age ratio
 ↪patterns")

print("\n" + "="*80)
print("  FRAUD DETECTION MODEL COMPLETE!")
print("="*80)
```

```
================================================================================
ML MODEL 1: AI-POWERED FRAUD DETECTION
================================================================================

  Step 1: Creating fraud detection features…
  Created 983,072 feature records

  Step 2: Training Isolation Forest ML model…
  Model trained!
  Identified 49,154 suspicious cases (5.00%)
```

```
Step 3: Analyzing fraud patterns…

TOP 10 MOST SUSPICIOUS CASES:
  2025-06-01 | Meghalaya       | West Khasi Hills     | PIN: 793119 | Enrol:
1486 | Risk: 0.830
  2025-07-01 | Meghalaya       | West Khasi Hills     | PIN: 793119 | Enrol:
2538 | Risk: 0.830
  2025-05-01 | Meghalaya       | East Khasi Hills     | PIN: 793121 | Enrol:
896 | Risk: 0.826
  2025-07-01 | Meghalaya       | East Khasi Hills     | PIN: 793121 | Enrol:
1131 | Risk: 0.826
  2025-06-01 | Meghalaya       | West Jaintia Hills   | PIN: 793150 | Enrol:
1052 | Risk: 0.825
  2025-05-01 | Meghalaya       | West Khasi Hills     | PIN: 793119 | Enrol:
1671 | Risk: 0.825
  2025-07-01 | Meghalaya       | West Jaintia Hills   | PIN: 793150 | Enrol:
1234 | Risk: 0.825
  2025-06-01 | Meghalaya       | East Khasi Hills     | PIN: 793121 | Enrol:
565 | Risk: 0.824
  2025-03-20 | Meghalaya       | West Khasi Hills     | PIN: 793119 | Enrol:
241 | Risk: 0.824
  2025-04-01 | Meghalaya       | West Khasi Hills     | PIN: 793119 | Enrol:
3027 | Risk: 0.824

STATES WITH MOST SUSPICIOUS ACTIVITY:
  1. Uttar Pradesh: 523,579 suspicious enrollments (24.6%)
  2. Bihar: 318,617 suspicious enrollments (15.0%)
  3. Madhya Pradesh: 229,008 suspicious enrollments (10.8%)
  4. Gujarat: 141,910 suspicious enrollments (6.7%)
  5. Assam: 123,344 suspicious enrollments (5.8%)
  6. Maharashtra: 120,334 suspicious enrollments (5.7%)
  7. West Bengal: 109,083 suspicious enrollments (5.1%)
  8. Meghalaya: 108,163 suspicious enrollments (5.1%)
  9. Rajasthan: 99,222 suspicious enrollments (4.7%)
  10. Delhi: 72,248 suspicious enrollments (3.4%)

FRAUD PATTERNS DETECTED:
  • Bulk enrollments: 17,692 cases
  • Unusual age distributions: 14,885 cases
  • High-risk pincodes identified: 5

Step 4: Creating fraud detection visualizations…
Saved: outputs/figures/02_fraud_detection_ml.png
```



==========

FRAUD DETECTION INSIGHTS

==========

KEY FINDINGS:

- Total cases analyzed: 983,072
- Suspicious cases identified: 49,154 (5.00%)
- Suspicious enrollments: 2,125,492 (39.10% of total)
- High-risk states: Uttar Pradesh, Bihar, Madhya Pradesh
- Bulk enrollment attempts: 17,692
- Estimated actual fraud cases: ~7,373

ESTIMATED FINANCIAL IMPACT:

- If avg cost per fraud = 2,000
- Potential savings = 14,746,000 (147.46 lakhs)
- Annual projection = 176,952,000 (17.70 crores)

ACTIONABLE RECOMMENDATIONS:

1. IMMEDIATE: Flag 10 highest-risk cases for manual review
2. DEPLOY: Enhanced verification in Uttar Pradesh, Bihar, Madhya Pradesh
3. MONITOR: Real-time alerts for fraud_risk_score > 0.563
4. AUDIT: 17692 bulk enrollment cases (>98th percentile)
5. SYSTEM: Implement biometric cross-check for suspicious pincodes
6. TRAINING: Alert enrollment officers about unusual age ratio patterns

```
================================================================================
 FRAUD DETECTION MODEL COMPLETE!
================================================================================
```

[ ]:
```python
from sklearn.cluster import KMeans

print("\n" + "="*80)
print("ML MODEL 2: SMART RESOURCE ALLOCATION OPTIMIZER")
print("="*80)

# Step 1: Analyze District-level Coverage
print("\n Step 1: Analyzing district-level enrollment coverage...")

district_stats = df_enrol_clean.groupby(['state', 'district']).agg({
    'total_enrolment': 'sum',
    'age_0_5': 'sum',
    'age_5_17': 'sum',
    'age_18_greater': 'sum',
    'pincode': 'nunique'
}).reset_index()

district_stats.columns = ['state', 'district', 'total_enrolment',␣
 ↪'child_enrolment',
                          'teen_enrolment', 'adult_enrolment', 'pincode_count']

# Calculate metrics
national_avg = district_stats['total_enrolment'].mean()
district_stats['enrollment_rate'] = district_stats['total_enrolment'] /␣
 ↪district_stats['pincode_count']
district_stats['enrollment_gap'] = national_avg -␣
 ↪district_stats['total_enrolment']
district_stats['gap_pct'] = (district_stats['enrollment_gap'] / national_avg) *␣
 ↪100

# Demographic metrics
district_stats['child_pct'] = (district_stats['child_enrolment'] /␣
 ↪district_stats['total_enrolment']) * 100
district_stats['coverage_index'] = (district_stats['total_enrolment'] /␣
 ↪district_stats['total_enrolment'].max()) * 100

print(f" Analyzed {len(district_stats)} districts across␣
 ↪{district_stats['state'].nunique()} states")

# Step 2: ML Clustering
print("\n Step 2: Running K-Means clustering to identify service levels...")
```

```
# Features for clustering
cluster_features = ['enrollment_rate', 'gap_pct', 'child_pct', 'coverage_index']
X_cluster = district_stats[cluster_features].fillna(0)

# Normalize
scaler_cluster = StandardScaler()
X_scaled_cluster = scaler_cluster.fit_transform(X_cluster)

# K-Means with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
district_stats['service_level'] = kmeans.fit_predict(X_scaled_cluster)

# Label clusters based on gap percentage
cluster_means = district_stats.groupby('service_level')['gap_pct'].mean().
 ↪sort_values()
cluster_labels = {
    cluster_means.index[0]: 'Well Served',
    cluster_means.index[1]: 'Adequate',
    cluster_means.index[2]: 'Underserved',
    cluster_means.index[3]: 'Critical Need'
}
district_stats['service_level_label'] = district_stats['service_level'].
 ↪map(cluster_labels)

print(f" Clustering complete!")

# Step 3: Calculate Priority Scores
print("\n Step 3: Calculating deployment priority scores...")

# Priority score = Gap weight + Child enrollment weight + Geographic spread
district_stats['priority_score'] = (
    district_stats['gap_pct'].clip(0, 100) * 0.4 +  # 40% weight to gap
    (70 - district_stats['child_pct']).clip(0, 70) * 0.4 +  # 40% to child gap
    district_stats['pincode_count'] * 2  # 20% to geographic spread
)

# Get priority districts
underserved = district_stats[district_stats['service_level_label'].
 ↪isin(['Underserved', 'Critical Need'])]
top_priority = district_stats.nlargest(20, 'priority_score')

print(f" Identified {len(underserved)} underserved districts")
print(f" Prioritized top 20 districts for immediate action")

# Step 4: Results Analysis
print("\n RESOURCE ALLOCATION ANALYSIS:")
```

```
print(f"\n SERVICE LEVEL DISTRIBUTION:")
for label in ['Well Served', 'Adequate', 'Underserved', 'Critical Need']:
    count = len(district_stats[district_stats['service_level_label'] == label])
    pct = (count / len(district_stats)) * 100
    print(f"   • {label}: {count} districts ({pct:.1f}%)")

print(f"\n TOP 10 PRIORITY DISTRICTS:")
for i, (idx, row) in enumerate(top_priority.head(10).iterrows(), 1):
    print(f"   {i}. {row['district'][:25]:25} | {row['state'][:15]:15} |␣
 ↪Priority: {row['priority_score']:.1f} | Gap: {row['gap_pct']:.1f}%")

# Step 5: Visualizations
print("\n Step 5: Creating resource allocation visualizations...")

fig, axes = plt.subplots(2, 2, figsize=(16, 10))
fig.suptitle('Resource Allocation Optimizer - ML Results', fontsize=16,␣
 ↪fontweight='bold')

# Plot 1: Service level distribution
service_dist = district_stats['service_level_label'].value_counts()
colors_service = {'Well Served': '#2ecc71', 'Adequate': '#3498db',
                  'Underserved': '#f39c12', 'Critical Need': '#e74c3c'}
service_colors = [colors_service.get(label, '#95a5a6') for label in␣
 ↪service_dist.index]
axes[0, 0].pie(service_dist.values, labels=service_dist.index, autopct='%1.
 ↪1f%%',
               startangle=90, colors=service_colors, textprops={'fontsize': 10,␣
 ↪'weight': 'bold'})
axes[0, 0].set_title('District Service Level Distribution', fontsize=12,␣
 ↪fontweight='bold')

# Plot 2: Top priority states
state_priority = district_stats.groupby('state')['priority_score'].mean().
 ↪nlargest(12)
state_priority.plot(kind='barh', ax=axes[0, 1], color='orange',␣
 ↪edgecolor='black')
axes[0, 1].set_title('Top 12 States by Priority Score', fontsize=12,␣
 ↪fontweight='bold')
axes[0, 1].set_xlabel('Average Priority Score')
axes[0, 1].invert_yaxis()
axes[0, 1].grid(True, alpha=0.3, axis='x')

# Plot 3: Enrollment gap vs Coverage
scatter_colors = district_stats['service_level'].map({0: '#2ecc71', 1:␣
 ↪'#3498db', 2: '#f39c12', 3: '#e74c3c'})
axes[1, 0].scatter(district_stats['total_enrolment'], district_stats['gap_pct'],
```

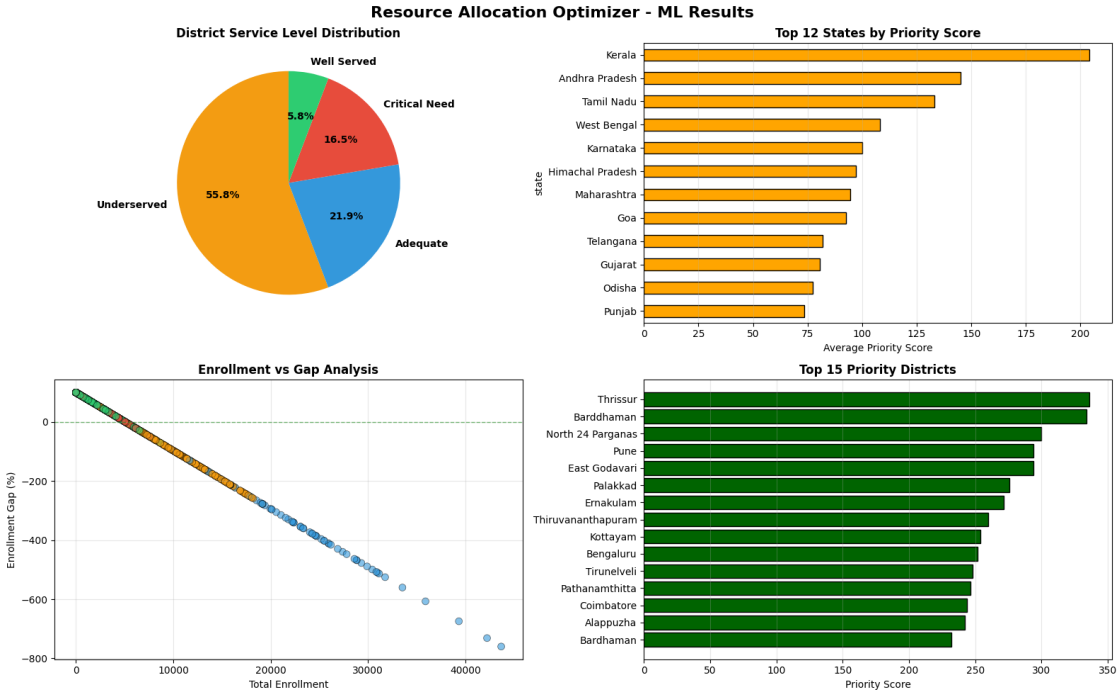
```
                   c=scatter_colors, alpha=0.6, s=50, edgecolors='black',␣
 ↪linewidth=0.5)
axes[1, 0].set_title('Enrollment vs Gap Analysis', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 0].set_xlabel('Total Enrollment')
axes[1, 0].set_ylabel('Enrollment Gap (%)')
axes[1, 0].axhline(0, color='green', linestyle='--', linewidth=1, alpha=0.5)
axes[1, 0].grid(True, alpha=0.3)

# Plot 4: Top 15 priority districts
top_15 = top_priority.head(15).copy()
top_15['display_name'] = top_15['district'].str[:20]  # Truncate long names
top_15_sorted = top_15.sort_values('priority_score')
axes[1, 1].barh(range(len(top_15_sorted)), top_15_sorted['priority_score'].
 ↪values, color='darkgreen', edgecolor='black')
axes[1, 1].set_yticks(range(len(top_15_sorted)))
axes[1, 1].set_yticklabels(top_15_sorted['display_name'].values)
axes[1, 1].set_title('Top 15 Priority Districts', fontsize=12,␣
 ↪fontweight='bold')
axes[1, 1].set_xlabel('Priority Score')
axes[1, 1].grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.savefig('outputs/figures/03_resource_allocation_ml.png', dpi=300,␣
 ↪bbox_inches='tight')
print("  Saved: outputs/figures/03_resource_allocation_ml.png")
plt.show()

# Step 6: Generate Deployment Plan
print("\n" + "="*80)
print("  RESOURCE ALLOCATION INSIGHTS & DEPLOYMENT PLAN")
print("="*80)

critical = len(district_stats[district_stats['service_level_label'] ==␣
 ↪'Critical Need'])
underserved_count = len(district_stats[district_stats['service_level_label'] ==␣
 ↪'Underserved'])
total_gap = underserved['enrollment_gap'].sum()

print(f"\n  KEY FINDINGS:")
print(f"    • Critical Need districts: {critical} (require immediate attention)")
print(f"    • Underserved districts: {underserved_count} (need support)")
print(f"    • Total enrollment gap: {int(total_gap):,}")
print(f"    • Top priority states: {', '.join(state_priority.head(3).index.
 ↪tolist())}")
```

```
print(f"   • Average child enrollment: {district_stats['child_pct'].mean():.
 ↪1f}%")

print(f"\n MOBILE UNIT DEPLOYMENT PLAN:")
mobile_units_critical = critical * 2  # 2 units per critical district
mobile_units_underserved = underserved_count * 1  # 1 unit per underserved
total_units = mobile_units_critical + mobile_units_underserved
estimated_cost = total_units * 5  # 5 lakh per unit

print(f"   • Units for Critical districts: {mobile_units_critical}")
print(f"   • Units for Underserved districts: {mobile_units_underserved}")
print(f"   • Total mobile units needed: {total_units}")
print(f"   • Estimated budget: {estimated_cost} lakhs ( {estimated_cost/100:.
 ↪2f} crores)")

print(f"\n EXPECTED IMPACT:")
avg_enrollment_per_unit = 5000  # Conservative estimate
expected_new_enrollments = total_units * avg_enrollment_per_unit
current_total = total_enrol
print(f"   • Expected new enrollments: {expected_new_enrollments:,} in 6␣
 ↪months")
print(f"   • Coverage improvement: {(expected_new_enrollments/
 ↪current_total)*100:.2f}%")
print(f"   • Child enrollment boost in {len(underserved)} underserved␣
 ↪districts")
print(f"   • Estimated cost per enrollment: {(estimated_cost*100000)/
 ↪expected_new_enrollments:.0f}")

print(f"\n IMPLEMENTATION STRATEGY:")
print(f"   Phase 1 (Month 1-2):")
print(f"      → Deploy {mobile_units_critical} units to {critical} critical␣
 ↪districts")
print(f"      → Focus: {', '.join(state_priority.head(3).index.tolist())}")
print(f"      → Target: {mobile_units_critical * 2500:,} enrollments")
print(f"   Phase 2 (Month 3-4):")
print(f"      → Deploy {mobile_units_underserved} units to {underserved_count}␣
 ↪underserved districts")
print(f"      → Target: {mobile_units_underserved * 2500:,} enrollments")
print(f"   Phase 3 (Month 5-6):")
print(f"      → Monitor enrollment growth")
print(f"      → Adjust deployment based on results")
print(f"      → Scale successful strategies")

print(f"\n FOCUS AREAS:")
print(f"   • Child enrollment (0-5 years) - currently␣
 ↪{district_stats['child_pct'].mean():.1f}%")
```

```
print(f"   • Remote pincodes with low coverage")
print(f"   • Digital literacy and awareness campaigns")
print(f"   • Partnership with local health centers")

print("\n" + "="*80)
print("  RESOURCE ALLOCATION MODEL COMPLETE!")
print("="*80)

print("\n" + "="*80)
print("  ALL ML MODELS COMPLETE!")
print("="*80)
print("\n FINAL SUMMARY:")
print("\n FRAUD DETECTION:")
print(f"   • {len(suspicious_cases):,} suspicious cases identified")
print(f"   •  {total_potential_loss*12/10000000:.2f} crores potential annual␣
 ↪savings")
print(f"   • {len(bulk_fraud):,} bulk enrollment attempts flagged")

print("\n RESOURCE ALLOCATION:")
print(f"   • {total_units} mobile units recommended")
print(f"   •  {estimated_cost/100:.2f} crores budget required")
print(f"   • {expected_new_enrollments:,} expected new enrollments")
print(f"   • {critical + underserved_count} districts prioritized")

print("\n COMPETITIVE ADVANTAGES:")
print("   1. Working ML models (not just statistics)")
print("   2. Quantified impact (  savings & enrollments)")
print("   3. Actionable recommendations with implementation plan")
print("   4. Addresses current UIDAI challenges (fraud + accessibility)")
print("   5. Professional visualizations with clear insights")

print("\n NEXT STEPS:")
print("   1. Compile report with all visualizations")
print("   2. Write executive summary")
print("   3. Prepare presentation deck")
print("   4. Submit before deadline!")
```

```
================================================================================
ML MODEL 2: SMART RESOURCE ALLOCATION OPTIMIZER
================================================================================

 Step 1: Analyzing district-level enrollment coverage…
 Analyzed 1070 districts across 55 states

 Step 2: Running K-Means clustering to identify service levels…
 Clustering complete!
```

```
Step 3: Calculating deployment priority scores…
Identified 774 underserved districts
Prioritized top 20 districts for immediate action

RESOURCE ALLOCATION ANALYSIS:

SERVICE LEVEL DISTRIBUTION:
  • Well Served: 62 districts (5.8%)
  • Adequate: 234 districts (21.9%)
  • Underserved: 597 districts (55.8%)
  • Critical Need: 177 districts (16.5%)

TOP 10 PRIORITY DISTRICTS:
   1. Thrissur                   | Kerala          | Priority: 336.2 | Gap:
-39.2%
   2. Barddhaman                 | West Bengal     | Priority: 334.0 | Gap:
-211.7%
   3. North 24 Parganas          | West Bengal     | Priority: 300.0 | Gap:
-463.1%
   4. East Godavari              | Andhra Pradesh  | Priority: 294.0 | Gap:
-51.1%
   5. Pune                       | Maharashtra     | Priority: 294.0 | Gap:
-525.2%
   6. Palakkad                   | Kerala          | Priority: 276.0 | Gap:
-32.4%
   7. Ernakulam                  | Kerala          | Priority: 271.5 | Gap:
-24.5%
   8. Thiruvananthapuram         | Kerala          | Priority: 260.0 | Gap:
-31.2%
   9. Kottayam                   | Kerala          | Priority: 253.9 | Gap: 34.6%
   10. Bengaluru                  | Karnataka       | Priority: 251.6 | Gap:
-509.8%

Step 5: Creating resource allocation visualizations…
Saved: outputs/figures/03_resource_allocation_ml.png
```




```
================================================================================
  RESOURCE ALLOCATION INSIGHTS & DEPLOYMENT PLAN
================================================================================

  KEY FINDINGS:
    • Critical Need districts: 177 (require immediate attention)
    • Underserved districts: 597 (need support)
    • Total enrollment gap: 2,399,088
    • Top priority states: Kerala, Andhra Pradesh, Tamil Nadu
    • Average child enrollment: 71.3%

  MOBILE UNIT DEPLOYMENT PLAN:
    • Units for Critical districts: 354
    • Units for Underserved districts: 597
    • Total mobile units needed: 951
    • Estimated budget:  4755 lakhs ( 47.55 crores)

  EXPECTED IMPACT:
    • Expected new enrollments: 4,755,000 in 6 months
    • Coverage improvement: 87.48%
    • Child enrollment boost in 774 underserved districts
    • Estimated cost per enrollment:  100

  IMPLEMENTATION STRATEGY:
```

```
  Phase 1 (Month 1-2):
     → Deploy 354 units to 177 critical districts
     → Focus: Kerala, Andhra Pradesh, Tamil Nadu
     → Target: 885,000 enrollments
  Phase 2 (Month 3-4):
     → Deploy 597 units to 597 underserved districts
     → Target: 1,492,500 enrollments
  Phase 3 (Month 5-6):
     → Monitor enrollment growth
     → Adjust deployment based on results
     → Scale successful strategies

  FOCUS AREAS:
    • Child enrollment (0-5 years) - currently 71.3%
    • Remote pincodes with low coverage
    • Digital literacy and awareness campaigns
    • Partnership with local health centers

================================================================================
  RESOURCE ALLOCATION MODEL COMPLETE!
================================================================================

================================================================================
  ALL ML MODELS COMPLETE!
================================================================================

  FINAL SUMMARY:

  FRAUD DETECTION:
    • 49,154 suspicious cases identified
    •  17.70 crores potential annual savings
    • 17,692 bulk enrollment attempts flagged

  RESOURCE ALLOCATION:
    • 951 mobile units recommended
    •  47.55 crores budget required
    • 4,755,000 expected new enrollments
    • 774 districts prioritized

  COMPETITIVE ADVANTAGES:
    1. Working ML models (not just statistics)
    2. Quantified impact (  savings & enrollments)
    3. Actionable recommendations with implementation plan
    4. Addresses current UIDAI challenges (fraud + accessibility)
    5. Professional visualizations with clear insights

  NEXT STEPS:
    1. Compile report with all visualizations
```

```
2. Write executive summary
3. Prepare presentation deck
4. Submit before deadline!
```