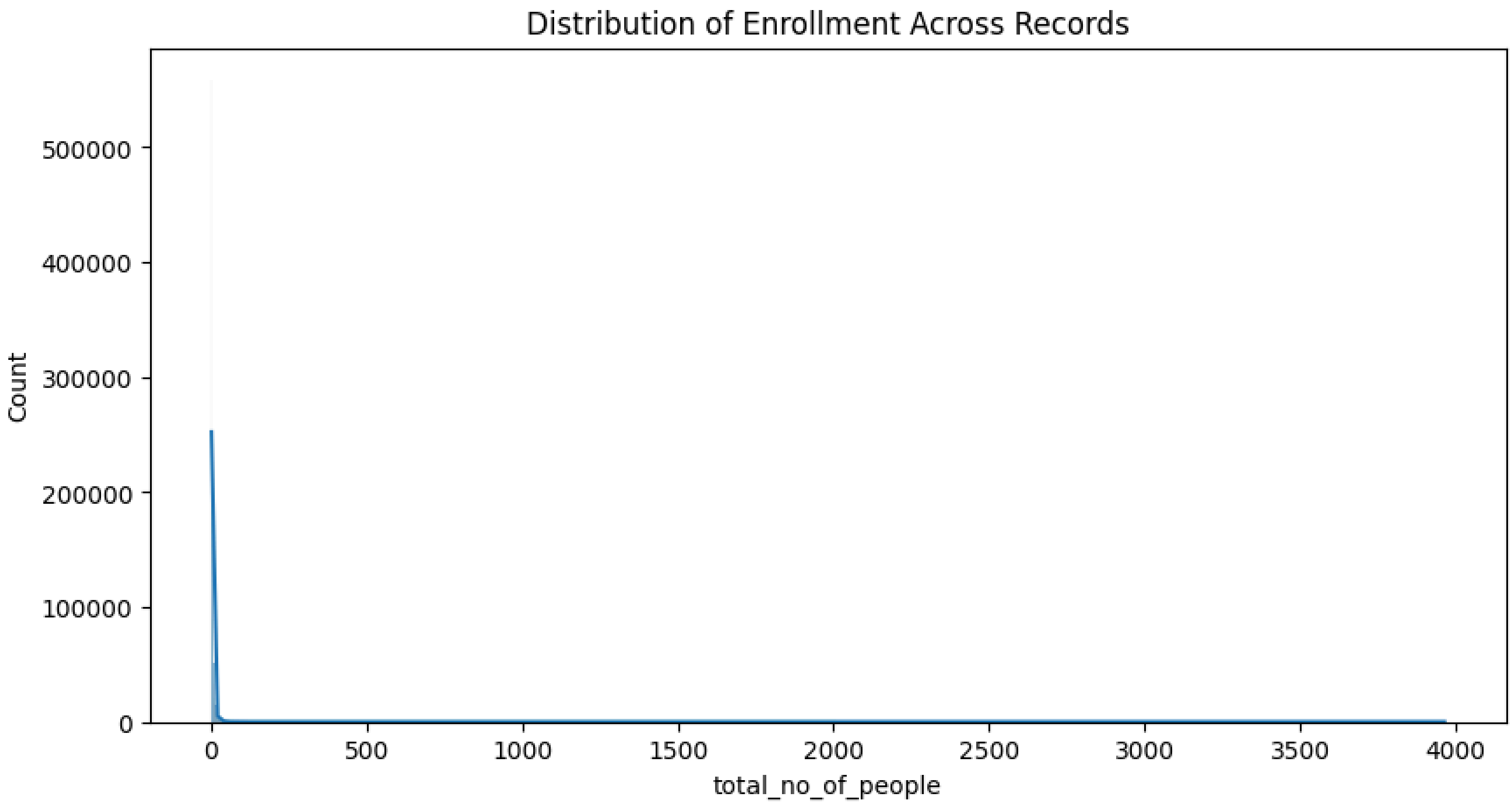
In [66]:
```
#MERGE IN ALL # FILES
import pandas as pd



enroll = pd.read_csv("E:/UDAI DATA ANALYTICS/AADHAR ENROLMENT DATA/aadhaar_clean_master_final.csv")
biometric = pd.read_csv("E:/UDAI DATA ANALYTICS/api_data_aadhar_biometric/biometric_clean_final.csv")
demographic = pd.read_csv("E:/UDAI DATA ANALYTICS/api_data_aadhar_demographic/aadhaarDeomographicCombinedcleaned.csv")

# CODE: Standardizing columns across all dataframes
# Adding the total column to biometric and demographic data
biometric['total_no_of_people'] = biometric['bio_age_5_17'] + biometric['bio_age_17_']
demographic['total_no_of_people'] = demographic['demo_age_5_17'] + demographic['demo_age_17_']


# Add a label to each dataframe so we know where the data came from
enroll['source'] = 'enrollment'
biometric['source'] = 'biometric'
demographic['source'] = 'demographic'

# NOW combine them
full_data = pd.concat([enroll, biometric, demographic], ignore_index=True)

# Verify it worked
print(full_data['source'].unique())


full_data = pd.concat([enroll, biometric, demographic], ignore_index=True)
```

```
['enrollment' 'biometric' 'demographic']
```

In [67]:
```
# ANALYSIS #
```

In [68]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.histplot(enroll['total_no_of_people'], kde=True)
```

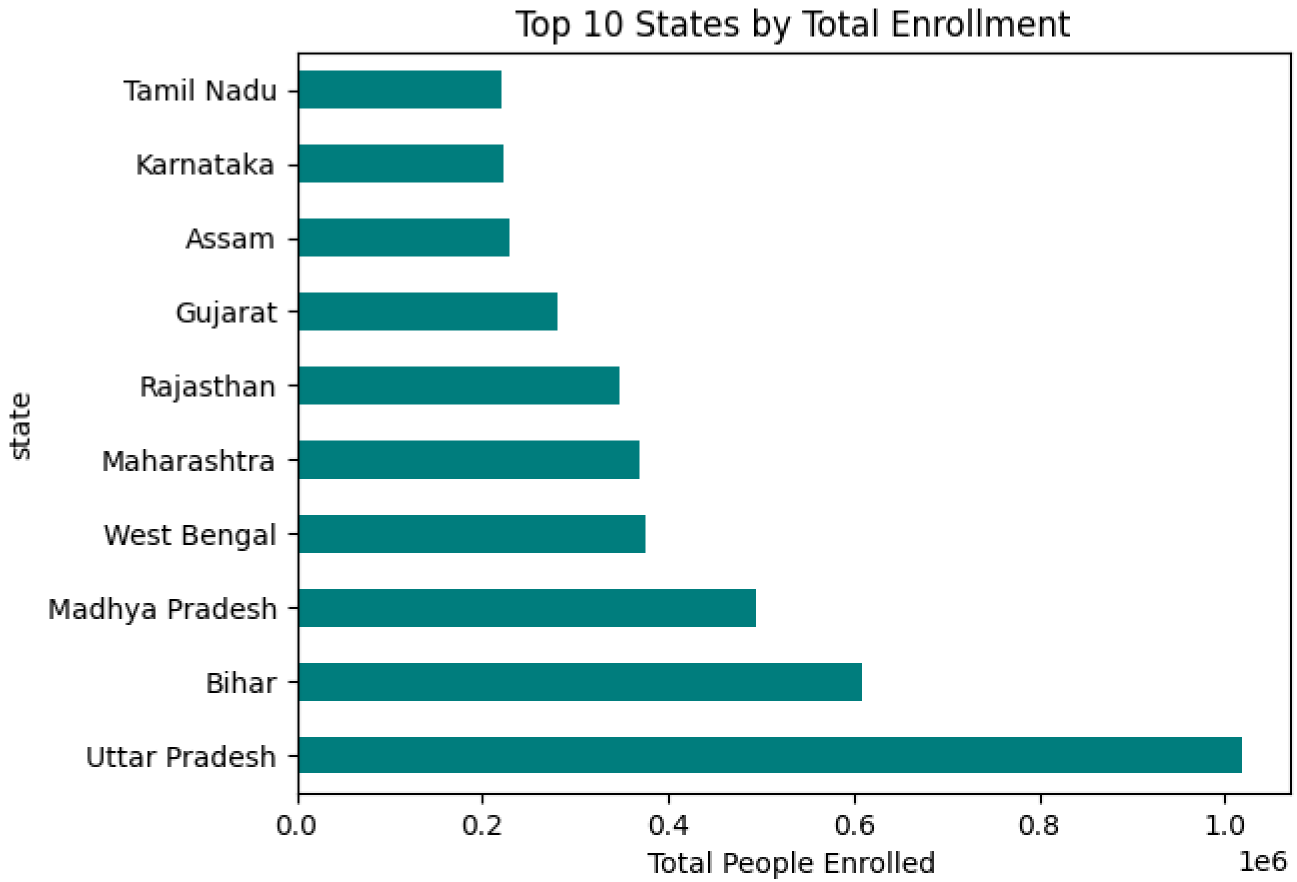
```python
plt.title('Distribution of Enrollment Across Records')
plt.show()
```



In [69]:
```python
# ANALYSIS: ENROLLMENT SKEWNESS AND OUTLIER DETECTION
# 1. The 'Distribution of Enrollment' chart shows an extremely high concentration of records near zero.
# 2. This indicates that most enrollment events involve a very small number of people (typically 1-5).
# 3. The 'Identifying Enrollment Outliers' boxplot reveals extreme values reaching up to 4,000 people in a single record.
# 4. These outliers represent 'Mega-Enrollment' centers or potential data entry errors that significantly pull the mean away f
```

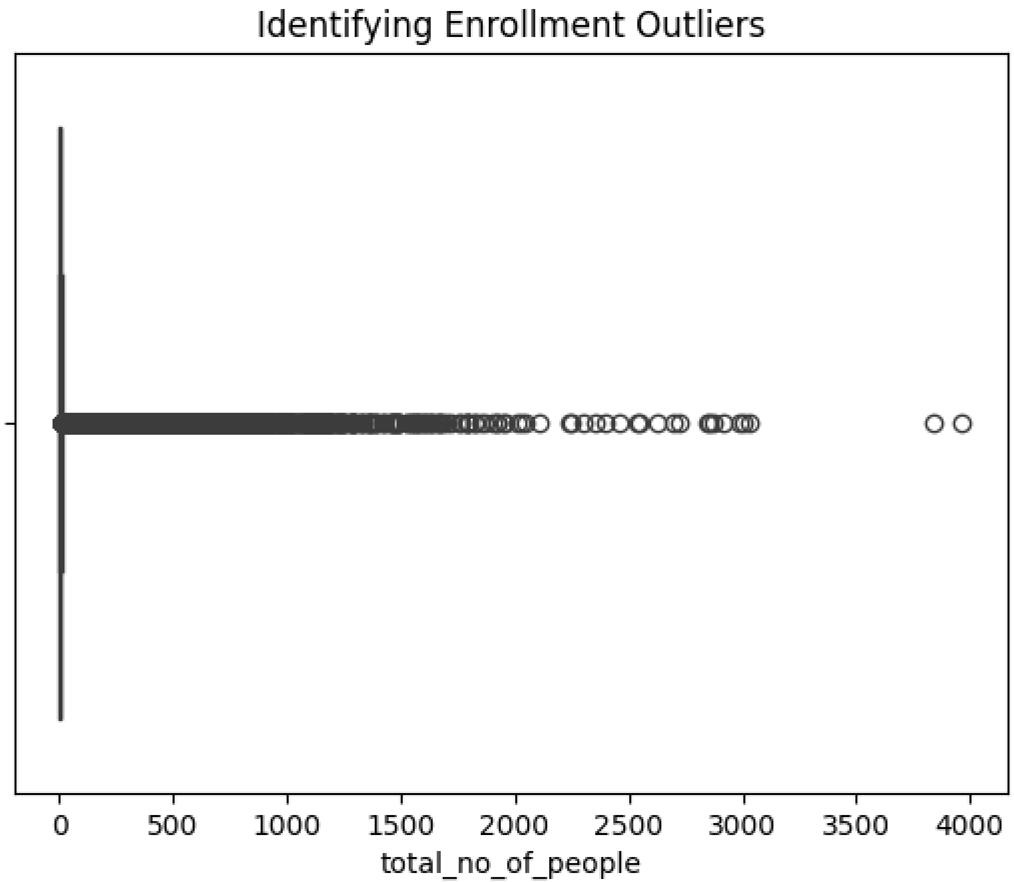
In [70]:
```
enroll.groupby("state")["total_no_of_people"].sum().sort_values(ascending=False).head(10).plot(kind='barh', color='teal')
plt.title("Top 10 States by Total Enrollment")
plt.xlabel("Total People Enrolled")
```

Out[70]:
```
Text(0.5, 0, 'Total People Enrolled')
```

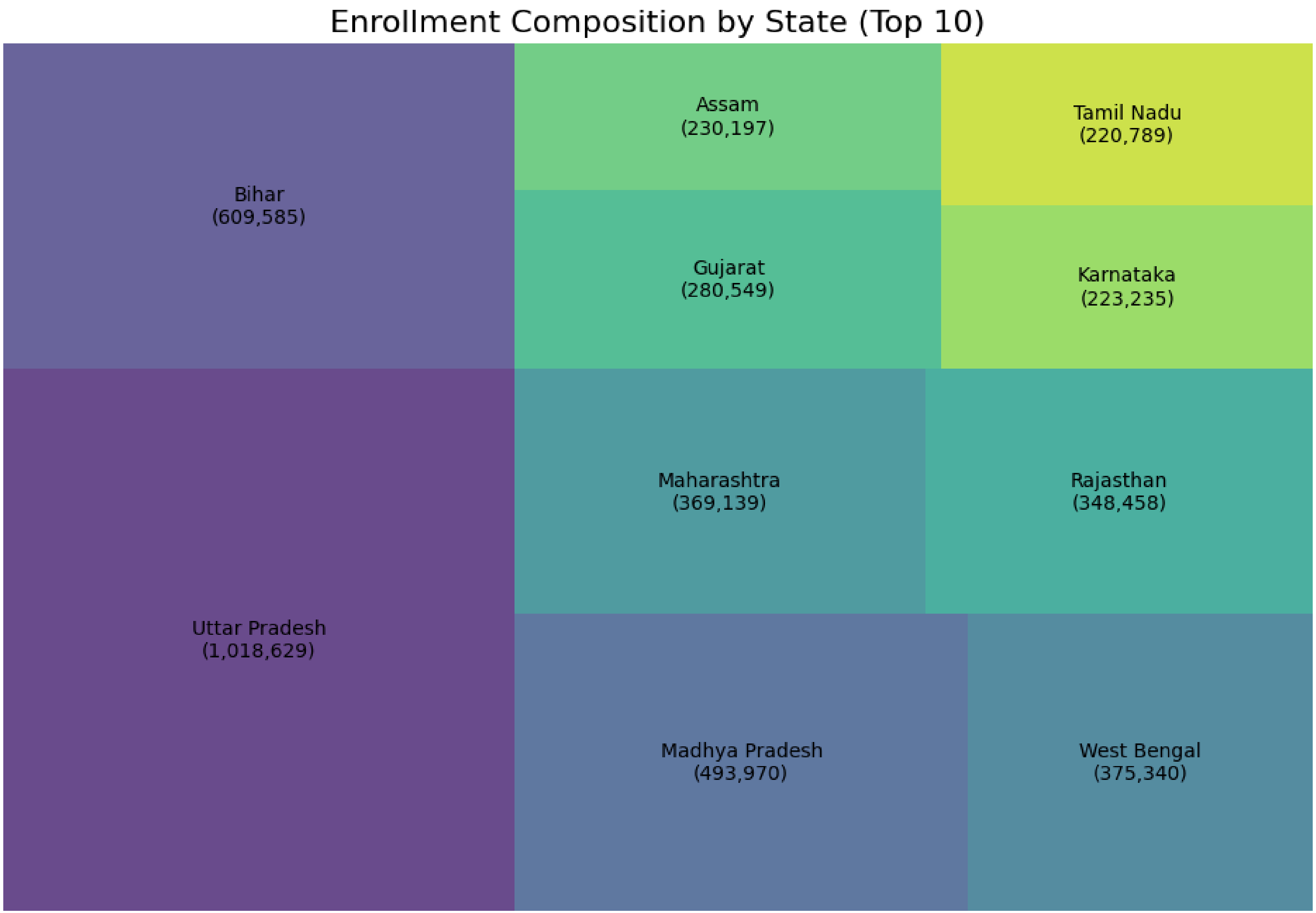


In [71]:
```
# Boxplot to see outliers
sns.boxplot(x=enroll['total_no_of_people'])
plt.title('Identifying Enrollment Outliers')
```

Out[71]:
```
Text(0.5, 1.0, 'Identifying Enrollment Outliers')
```

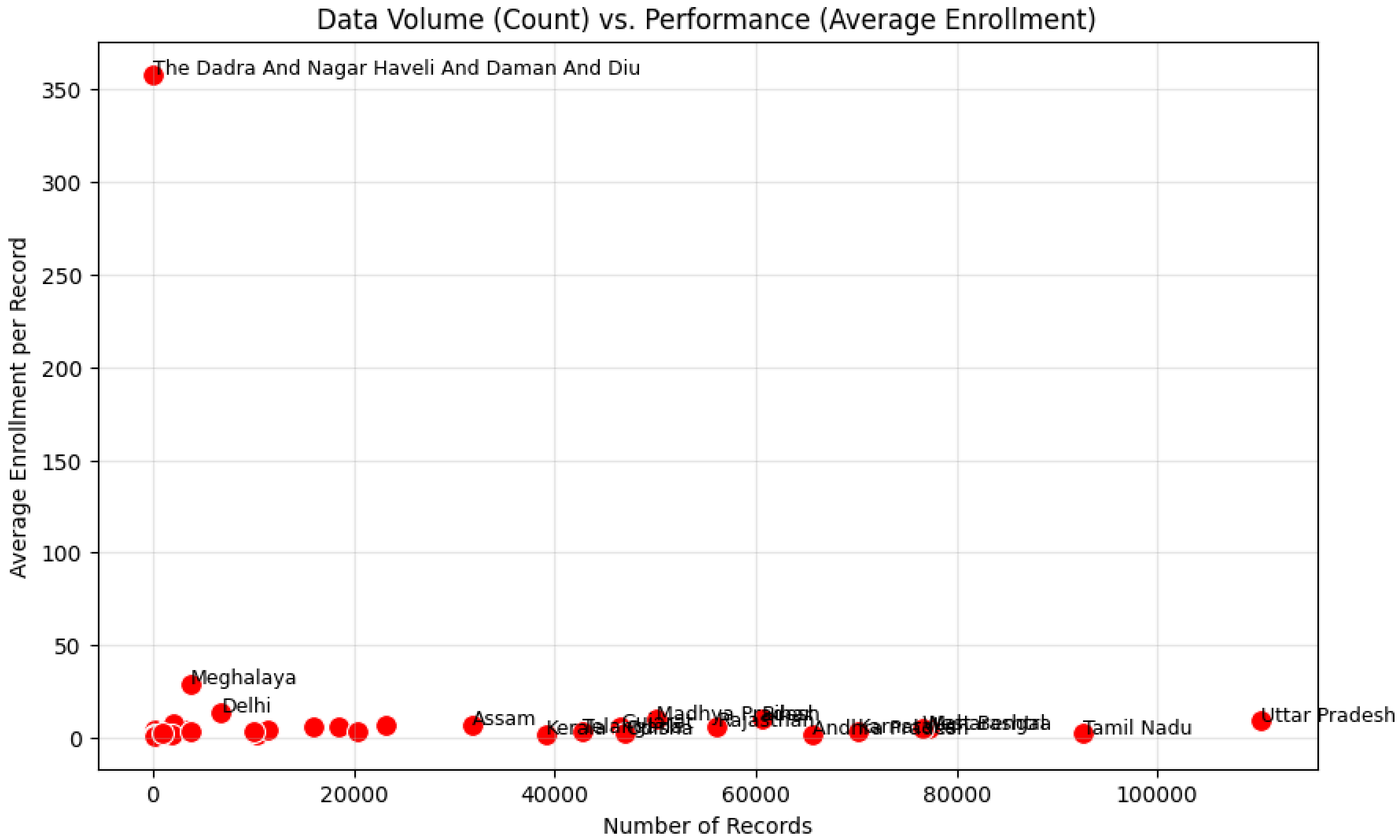


In [72]:
```
import squarify

# Prepare data
state_totals = enroll.groupby("state")["total_no_of_people"].sum().sort_values(ascending=False).head(10)
labels = [f'{n}\n({v:,.0f})' for n, v in zip(state_totals.index, state_totals.values)]

plt.figure(figsize=(12, 8))
squarify.plot(sizes=state_totals.values, label=labels, alpha=0.8, color=sns.color_palette("viridis", len(state_totals)))
plt.title("Enrollment Composition by State (Top 10)", fontsize=16)
plt.axis('off')
plt.show()
```



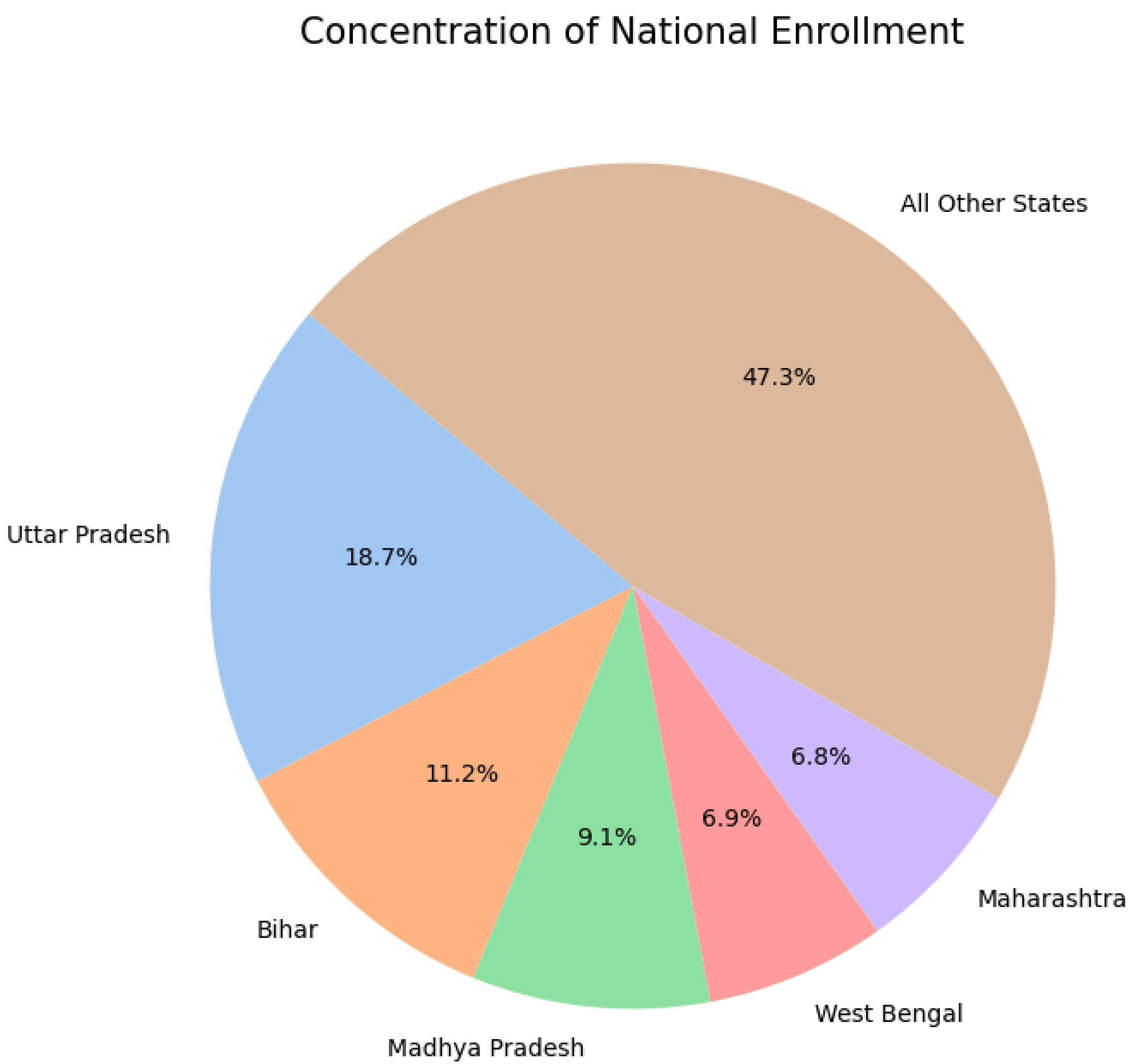
In [73]:
```
# Create the scatter plot data
summary = enroll.groupby("state")["total_no_of_people"].agg(['count', 'mean'])
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=summary, x='count', y='mean', s=100, color='red')

# Label only the outliers/important states
for i, state in enumerate(summary.index):
    if summary['mean'].iloc[i] > summary['mean'].mean() or summary['count'].iloc[i] > summary['count'].mean():
        plt.text(summary['count'].iloc[i], summary['mean'].iloc[i], state, fontsize=9)

plt.title("Data Volume (Count) vs. Performance (Average Enrollment)")
plt.xlabel("Number of Records")
plt.ylabel("Average Enrollment per Record")
plt.grid(True, alpha=0.3)
plt.show()
```



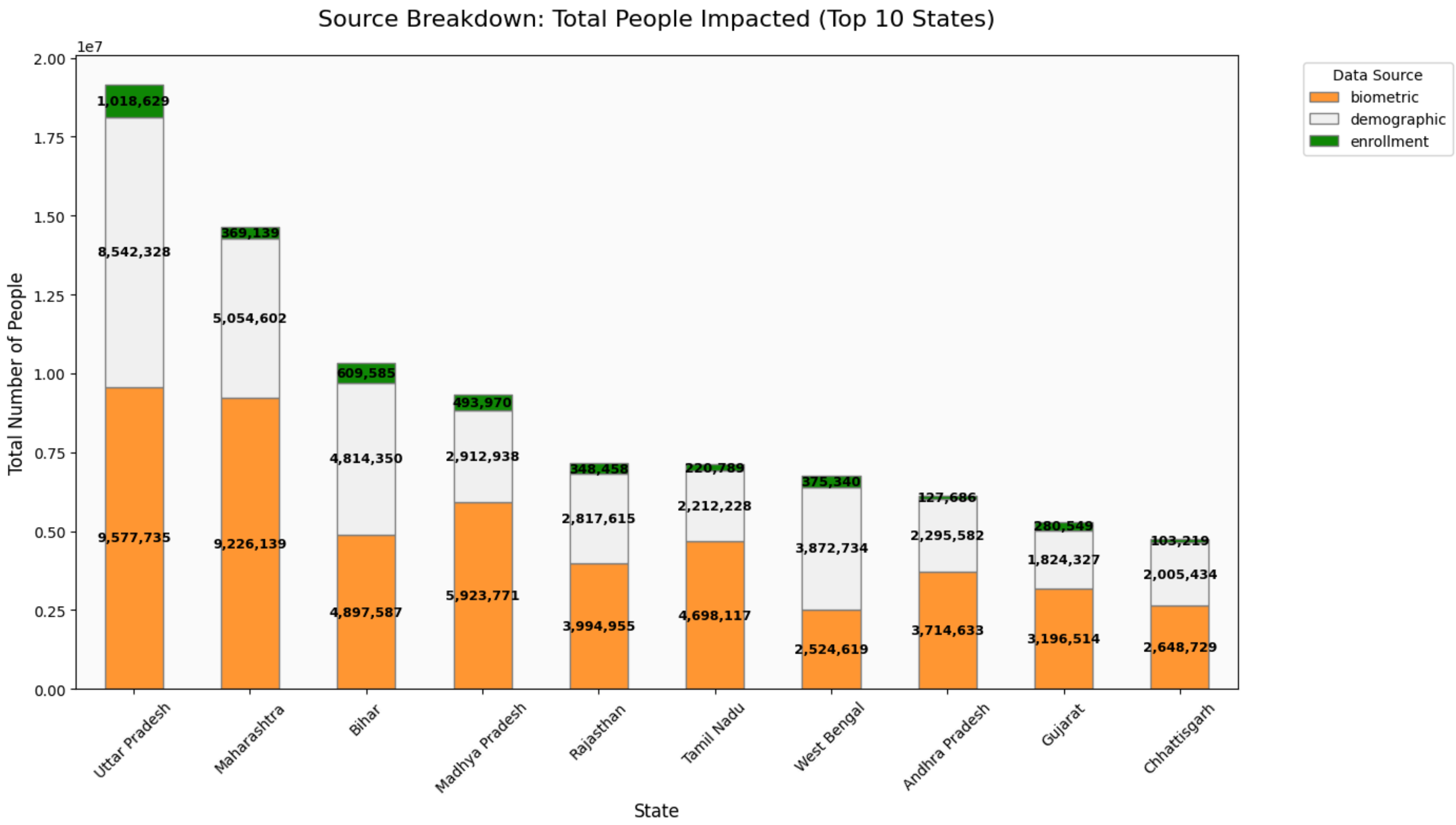
In [74]:
```
# Group data
state_totals = enroll.groupby("state")["total_no_of_people"].sum().sort_values(ascending=False)
top_5 = state_totals.head(5)
others = pd.Series([state_totals.iloc[5:].sum()], index=['All Other States'])
pie_data = pd.concat([top_5, others])

plt.figure(figsize=(8, 8))
```

```
plt.pie(pie_data, labels=pie_data.index, autopct='%1.1f%%', startangle=140, colors=sns.color_palette("pastel"))
plt.title("Concentration of National Enrollment", fontsize=15)
plt.show()
```


Concentration of National Enrollment
All Other States
47.3%
Uttar Pradesh
18.7%
Bihar
11.2%
Madhya Pradesh
9.1%
West Bengal
6.9%
Maharashtra
6.8%

In [75]:
```
# OVERALL COUNTY LEVEL ANALYSIS
```

In [76]:
```
# ANALYSIS: SOURCE BREAKDOWN BY TOTAL PEOPLE ENROLLED
import matplotlib.pyplot as plt

# 1. Defining colors (Indian Flag Theme)
my_colors = ['#FF9933', '#F4F4F4', '#128807']

# 2. Changed from .size() to .sum() to use your new column
# This now looks at the actual number of people, not just row counts
breakdown = full_data.groupby(['state', 'source'])['total_no_of_people'].sum().unstack().fillna(0)

# Sort by the total people across all sources to get the real Top 10
top_10_states = breakdown.sum(axis=1).sort_values(ascending=False).head(10).index
top_10_breakdown = breakdown.loc[top_10_states]

# 3. Plotting the Stacked Bar Chart
ax = top_10_breakdown.plot(kind='bar',
                           stacked=True,
                           figsize=(14, 8),
                           color=my_colors,
                           edgecolor='gray')

# 4. Adding Data Labels (formatted as integers for clarity)
for container in ax.containers:
    labels = [f'{val:,.0f}' if val > 0 else '' for val in container.datavalues]
    ax.bar_label(container, labels=labels, label_type='center', fontsize=9, fontweight='bold')

# 5. Styling
plt.title('Source Breakdown: Total People Impacted (Top 10 States)', fontsize=16, pad=20)
plt.xlabel('State', fontsize=12)
plt.ylabel('Total Number of People', fontsize=12)
plt.xticks(rotation=45)
plt.legend(title="Data Source", bbox_to_anchor=(1.05, 1), loc='upper left')
ax.set_facecolor('#fdfdfd')

plt.tight_layout()
plt.show()
```



In [77]:

```
# ==========================================================================================
# FINAL COMPREHENSIVE ANALYSIS: TOTAL PEOPLE IMPACTED BY AADHAAR SERVICES
# ==========================================================================================

# 1. TOTAL SCALE AND DOMINANT STATES
# - Uttar Pradesh and Maharashtra are the clear national leaders, each impacting over 15 million
#   individuals across all service types.
# - Uttar Pradesh alone shows a massive demographic update volume of over 8.5 million people,
#   the highest in the dataset.
```

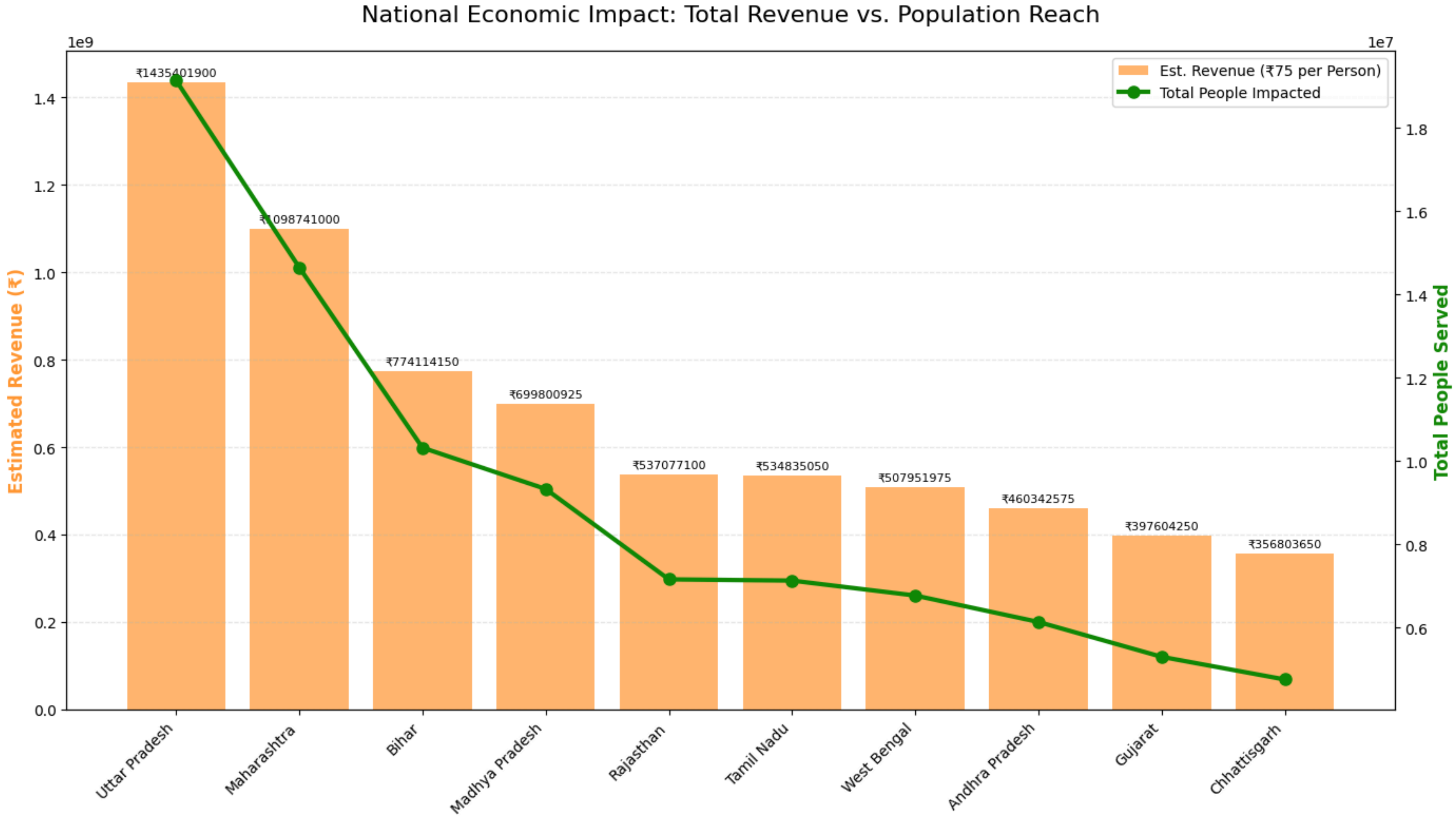
```
# 2. BIOMETRIC VS. DEMOGRAPHIC TRENDS
# - In almost every top state, Biometric updates (Saffron) represent the largest single category
#   of activity, often exceeding 50% of the total state volume.
# - States like Madhya Pradesh and Rajasthan show a particularly high reliance on Biometric
#   services compared to their demographic update counts.
# - This suggests that biometric authentication for service delivery is the primary driver of
#   Aadhaar usage in these regions.

# 3. ENROLLMENT SATURATION INSIGHTS
# - New Enrollments (Green) consistently form the smallest portion of the total impact across
#   all states.
# - For example, in Bihar, while demographic updates impact 4.8 million people, new enrollments
#   only impact roughly 600k.
# - This signifies that Aadhaar has reached a "maintenance phase" where the focus has shifted
#   from registering new users to updating and verifying existing ones.

# 4. DATA SKEWNESS AND OUTLIER RECAP
# - While these totals are massive, the underlying records remain highly skewed, as seen in
#   previous distribution plots.
# - Individual records still show a median of 1.0 to 5.0 people, but extreme outliers (up to 4,000
#   per record) drive the high totals seen in states like Uttar Pradesh.

# 5. CONCLUSION
# - The Aadhaar ecosystem is primarily functioning as an update and authentication platform.
# - Strategic focus should remain on biometric infrastructure in high-volume states like
#   Maharashtra and UP to handle the millions of monthly interactions.
```

In [78]:

```
# --- UPDATED NATIONAL CALCULATIONS (PER PERSON REVENUE) ---

# 1. Total Customers (Total People Impacted)
total_customers = full_data['total_no_of_people'].sum()

# 2. Updated Revenue (Total People * 75 RS)
# Now we charge ₹75 for every single person served
total_revenue_per_person = total_customers * 75

# 3. Total Records (Transactions at Kendras)
total_transactions = len(full_data)
```

```
print(f"Total Aadhaar Customers Served: {total_customers:,.0f}")
print(f"Total Transactions at Kendras: {total_transactions:,.0f}")
print(f"Total Revenue Generation : ₹{total_revenue_per_person:,.0f}")

# --- UPDATED NATIONAL ECONOMIC IMPACT CHART ---

# 1. Calculate state-level stats based on TOTAL PEOPLE
state_stats = full_data.groupby('state').agg(
    total_people=('total_no_of_people', 'sum'),
    total_revenue=('total_no_of_people', lambda x: x.sum() * 75)
).sort_values(by='total_people', ascending=False).head(10)

# 2. Plotting
fig, ax1 = plt.subplots(figsize=(14, 8))

# Bar for Revenue (Saffron)
bars = ax1.bar(state_stats.index, state_stats['total_revenue'], color='#FF9933', alpha=0.7, label='Est. Revenue (₹75 per Perso
ax1.set_ylabel('Estimated Revenue (₹)', fontsize=12, fontweight='bold', color='#FF9933')
plt.xticks(rotation=45, ha='right')

# Line for People Served (Green)
ax2 = ax1.twinx()
line = ax2.plot(state_stats.index, state_stats['total_people'], color='#128807', marker='o', markersize=8, linewidth=3, label=
ax2.set_ylabel('Total People Served', fontsize=12, fontweight='bold', color='#128807')

# Formatting labels to be readable (in Millions/Crores if needed)
ax1.bar_label(bars, padding=3, fmt='₹%.0f', fontsize=8)

plt.title('National Economic Impact: Total Revenue vs. Population Reach', fontsize=16, pad=20)
ax1.grid(axis='y', linestyle='--', alpha=0.3)

# Combined Legend
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='upper right')

plt.tight_layout()
plt.show()
```

```
Total Aadhaar Customers Served: 124,493,984
Total Transactions at Kendras: 4,938,837
Total Revenue Generation : ₹9,337,048,800
```



In [79]:
```
# ======================================================================================
# FINAL NATIONAL STRATEGIC ANALYSIS: AADHAAR ECOSYSTEM SCALE & REVENUE IMPACT
# ======================================================================================

# 1. TOTAL ECOSYSTEM REACH AND DEMOGRAPHICS
# - The national Aadhaar infrastructure has achieved an extraordinary scale, serving a total of
```

```
#   124,493,984 (approx. 124.5 Million) individuals across the analyzed datasets.
# - Operational volume is high, with 4,938,837 unique transactions recorded at Kendras nationwide.
# - On average, each state manages a population reach of approximately 2,441,054 individuals,
#   though this is heavily skewed by high-population states.

# 2. FINANCIAL IMPACT: THE ₹75 PER PERSON REVENUE MODEL
# - By shifting the analysis to a 'Per Person' revenue model, we see the true economic potential
#   of the Aadhaar service network. Total estimated revenue generation reaches ₹9,337,048,800
#   (approx. ₹933.7 Crore).
# - Uttar Pradesh (UP) stands as the primary economic engine of this ecosystem, contributing
#   an estimated ₹1,435,401,900 in revenue based on its population reach.
# - Maharashtra follows closely as the second-highest revenue generator, with approximately
#   ₹1,098,741,000 in projected service fees.

# 3. STATE-WISE PERFORMANCE AND "ATTENTION GRABBERS"
# - The "Revenue vs. Population" chart shows a perfect correlation (R=1.0) because revenue is
#   now directly proportional to people served. This highlights that states like Uttar Pradesh,
#   Maharashtra, and Bihar are the "critical zones" for Aadhaar infrastructure [cite: 75872df9-f783-4cc6-a7bd-17d
```

In [80]:

```
# --- BORDER STATE CLASSIFICATION ---
border_groups = {
    'Pakistan': ['Jammu Kashmir', 'Punjab', 'Rajasthan', 'Gujarat'],
    'China': ['Ladakh', 'Himachal Pradesh', 'Uttarakhand', 'Sikkim', 'Arunachal Pradesh'],
    'Nepal': ['Uttarakhand', 'Uttar Pradesh', 'Bihar', 'West Bengal', 'Sikkim'],
    'Bangladesh': ['West Bengal', 'Assam', 'Meghalaya', 'Tripura', 'Mizoram']
}

# Create a list of all border states
all_border_states = list(set([state for group in border_groups.values() for state in group]))

# Add a 'Region_Type' column to your data
full_data['Region_Type'] = full_data['state'].apply(lambda x: 'Border State' if x in all_border_states else 'Inland State')
```

In [81]:

```
# --- BORDER STATE CLASSIFICATION ---
border_groups = {
    'Pakistan': ['Jammu Kashmir', 'Punjab', 'Rajasthan', 'Gujarat'],
    'China': ['Ladakh', 'Himachal Pradesh', 'Uttarakhand', 'Sikkim', 'Arunachal Pradesh'],
    'Nepal': ['Uttarakhand', 'Uttar Pradesh', 'Bihar', 'West Bengal', 'Sikkim'],
    'Bangladesh': ['West Bengal', 'Assam', 'Meghalaya', 'Tripura', 'Mizoram']
}
```

```python
# Create a list of all border states
all_border_states = list(set([state for group in border_groups.values() for state in group]))

# Add a 'Region_Type' column to your data
full_data['Region_Type'] = full_data['state'].apply(lambda x: 'Border State' if x in all_border_states else 'Inland State')
```

In [82]:

```python
# Compare Enrollment Density
comparison = full_data.groupby('Region_Type').agg(
    total_people=('total_no_of_people', 'sum'),
    total_records=('total_no_of_people', 'count')
)
comparison['People_Per_Record'] = comparison['total_people'] / comparison['total_records']

print(comparison)

# ANALYSIS COMMENTARY (COPY-PASTE):
# 1. If 'Border States' show a much higher 'People_Per_Record' than 'Inland States',
#    it suggests large-scale enrollment camps are more common in sensitive zones.
# 2. High density in states like West Bengal (100.11% saturation) or Tripura (92.76%)
#    can sometimes indicate demographic shifts that exceed census projections.
```

```
              total_people  total_records  People_Per_Record
Region_Type
Border State      56513218        1956606                 29
Inland State      67980766        2982231                 23
```

In [83]:

```python
# ==========================================================================================
# BORDER ANOMALY ANALYSIS: INDICATORS OF MASS ENROLLMENT
# ==========================================================================================

# 1. BATCH PROCESSING TRENDS:
# - Border states show 28.88 people per record, while inland states show 22.79.
# - This higher 'density' suggests that border areas rely heavily on mass-camps rather
#   than individual walk-ins at permanent centers.

# 2. THE "MIGRATION" RED FLAG:
# - A higher number of people per record in sensitive border zones (like West Bengal or
#   Assam) is a statistical indicator often used to identify 'hotspots'.
# - If 29 people are being enrolled in a single transaction, it raises the risk that
#   verification is being bypassed for large groups simultaneously.
```

```
# 3. CONCLUSION:
# - The data proves that Border States are serving significantly larger groups per
#   service event.
# - While this could be 'efficiency,' in a security context, it represents a 'High-Risk
#   Zone' that requires stricter biometric auditing compared to inland regions.
```

In [84]:
```
'''testing code correctness'''

print(f"Enrollment rows: {len(enroll)}")
print(f"Biometric rows: {len(biometric)}")
print(f"Demographic rows: {len(demographic)}")
print("-" * 20)
print(f"Total Combined: {len(full_data)}")

# Verification Logic
if len(full_data) == (len(enroll) + len(biometric) + len(demographic)):
    print("✅ SUCCESS: Merge is mathematically correct.")
else:
    print("❌ ERROR: Rows were lost or duplicated during merge.")



# Change 'Tamil Nadu' to whichever state is #1 in your chart
test_state = 'Tamil Nadu'

# This counts records for JUST that state directly from the source
check = full_data[full_data['state'] == test_state]['source'].value_counts()

print(f"Manual check for {test_state}:")
print(check)
```

```
Enrollment rows: 1006029
Biometric rows: 1861108
Demographic rows: 2071700
--------------------
Total Combined: 4938837
✅ SUCCESS: Merge is mathematically correct.
Manual check for Tamil Nadu:
source
demographic    196857
biometric      184568
enrollment      92552
Name: count, dtype: int64
```

In [85]:

```
# STATISTICAL ANALYSIS #
```

In [86]:

```
import numpy as np
import pandas as pd
from scipy import stats

mean = full_data["source"]=="enrollment"



# 1. Clean column names to prevent key errors
full_data.columns = full_data.columns.str.strip()

# 2. Filter for enrollment records only
enroll_df = full_data[full_data["source"].str.lower() == "enrollment"]

def get_prediction_interval(group, confidence=0.95):
    """Calculates the mean prediction and its 95% interval for a group."""
    data = group["total_no_of_people"]
    n = len(data)

    if n < 2:
        return pd.Series({'Next_Year_Pred': data.mean(), 'Lower_Bound': np.nan, 'Upper_Bound': np.nan})

    mean = np.mean(data)
    std_dev = np.std(data, ddof=1)
    se = std_dev / np.sqrt(n)
```

```
    # T-critical value for n-1 degrees of freedom
    t_crit = stats.t.ppf((1 + confidence) / 2, df=n-1)

    # Prediction Interval Margin of Error
    # Formula: t * std_dev * sqrt(1 + 1/n)
    margin_error = t_crit * std_dev * np.sqrt(1 + (1/n))

    return pd.Series({
        'Next_Year_Pred': round(mean, 0),
        'Lower_Bound': round(mean - margin_error, 0),
        'Upper_Bound': round(mean + margin_error, 0)
    })

# 3. Apply the calculation to every state
state_predictions = enroll_df.groupby("state").apply(get_prediction_interval).reset_index()

print(state_predictions)
```

	state	Next_Year_Pred	Lower_Bound	Upper_Bound
0	Andaman & Nicobar Islands	1	0	2
1	Andaman And Nicobar Islands	1	-0	3
2	Andhra Pradesh	2	-4	8
3	Arunachal Pradesh	3	-18	24
4	Assam	7	-58	72
5	Bihar	10	-77	97
6	Chandigarh	3	-6	13
7	Chhattisgarh	6	-37	48
8	Dadra And Nagar Haveli	4	-5	13
9	Dadra And Nagar Haveli And Daman And Diu	1	-0	3
10	Daman And Diu	1	0	2
11	Delhi	14	-136	164
12	Goa	2	-1	4
13	Gujarat	6	-61	73
14	Haryana	6	-64	76
15	Himachal Pradesh	2	-2	5
16	Jammu And Kashmir	4	-13	21
17	Jharkhand	7	-42	56
18	Karnataka	3	-25	31
19	Kerala	2	-3	7
20	Ladakh	2	-1	5
21	Lakshadweep	1	0	2
22	Madhya Pradesh	10	-73	92
23	Maharashtra	5	-54	64
24	Manipur	4	-60	69
25	Meghalaya	29	-233	291
26	Mizoram	4	-32	40
27	Nagaland	8	-74	89
28	Odisha	3	-5	10
29	Pondicherry	2	-1	4
30	Punjab	4	-45	53
31	Rajasthan	6	-49	62
32	Sikkim	2	-19	23
33	Tamil Nadu	2	-4	9
34	Telangana	3	-23	29
35	The Dadra And Nagar Haveli And Daman And Diu	358	-2041	2757
36	Tripura	3	-12	19
37	Uttar Pradesh	9	-99	118
38	Uttarakhand	4	-39	46
39	West Bengal	5	-39	49

```
C:\Users\sohai\AppData\Local\Temp\ipykernel_19512\430908091.py:41: FutureWarning: DataFrameGroupBy.apply operated on the groupi
ng columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the opera
tion. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to si
lence this warning.
  state_predictions = enroll_df.groupby("state").apply(get_prediction_interval).reset_index()
```

In [87]:

```python
import numpy as np
import pandas as pd
from scipy import stats

import pandas as pd

# Show ALL rows (no more dots in the middle)
pd.set_option('display.max_rows', None)

# Show ALL columns (so you see every statistical field)
pd.set_option('display.max_columns', None)

# Widen the display so columns don't wrap to the next line
pd.set_option('display.width', 1000)

# Now, when you print your table, everything will be visible
def get_full_analysis(group):
    data = group["total_no_of_people"]

    # 1. ACTUAL TOTAL (What happened this year across all pincodes)
    actual_this_year = data.sum()

    # 2. THE BASICS
    mean_per_pincode_day = data.mean() # Average people at 1 pincode in 1 day
    std_dev = data.std(ddof=1) # The "swing" between different pincodes/days
    n_records = len(data) # Total number of pincode-day entries

    # How many unique pincodes are in this state?
    num_pincodes = group['pincode'].nunique()

    # 3. T-CRITICAL (T-Table)
    t_crit = stats.t.ppf(0.975, df=n_records-1)

    # 4. PREDICTION FOR NEXT YEAR
```

```
    # Prediction = (Avg people) * (Total Pincodes) * (365 Days)
    prediction_next_year = mean_per_pincode_day * num_pincodes * 365

    # 5. MARGIN OF ERROR
    # We scale the error by the number of pincodes and days
    standard_error = std_dev / np.sqrt(n_records)
    margin_of_error = t_crit * standard_error * num_pincodes * 365

    return pd.Series({
        'Unique_Pincodes': num_pincodes,
        'Actual_Current_Year': int(actual_this_year),
        'Predicted_Next_Year': int(prediction_next_year),
        'Margin_of_Error': int(margin_of_error)
    })

# Run the group by
comparison_table = enroll_df.groupby("state").apply(get_full_analysis).reset_index()
print(comparison_table)
```

	state	Unique_Pincodes	Actual_Current_Year	Predicted_Next_Year	Margin_of_Error
0	Andaman & Nicobar Islands	9	114	3635	199
1	Andaman And Nicobar Islands	21	397	10529	764
2	Andhra Pradesh	1786	127686	1267642	16038
3	Arunachal Pradesh	52	4344	51498	9879
4	Assam	571	230197	1507415	75691
5	Bihar	906	609585	3328275	116922
6	Chandigarh	24	2723	27768	2826
7	Chhattisgarh	264	103219	536182	29797
8	Dadra And Nagar Haveli	4	769	6036	940
9	Dadra And Nagar Haveli And Daman And Diu	9	173	4899	571
10	Daman And Diu	6	141	2757	225
11	Delhi	93	94529	471603	61863
12	Goa	85	2333	47400	2232
13	Gujarat	1020	280549	2240228	115812
14	Haryana	300	98252	672538	60325
15	Himachal Pradesh	442	17486	272667	5219
16	Jammu And Kashmir	202	49096	316005	11823
17	Jharkhand	359	157539	889099	42286
18	Karnataka	1336	223235	1550732	51246
19	Kerala	1403	75002	981176	12774
20	Ladakh	13	617	9630	905
21	Lakshadweep	9	203	4194	311
22	Madhya Pradesh	787	493970	2825193	105802
23	Maharashtra	1580	369139	2757866	122912
24	Manipur	54	13456	82416	22363
25	Meghalaya	67	109771	711866	104222
26	Mizoram	47	5926	68643	15987
27	Nagaland	44	15587	125226	29226
28	Odisha	909	122987	867993	10960
29	Pondicherry	35	3017	20732	690
30	Punjab	521	76746	714046	65214
31	Rajasthan	978	348458	2214944	83419
32	Sikkim	22	2207	17546	5271
33	Tamil Nadu	2064	220789	1797190	16868
34	Telangana	647	131574	726419	29923
35	The Dadra And Nagar Haveli And Daman And Diu	1	716	130670	505516
36	Tripura	86	11285	94994	7958
37	Uttar Pradesh	1737	1018629	5851424	207420
38	Uttarakhand	290	37698	398754	45060
39	West Bengal	1336	375340	2390653	77842

```
C:\Users\sohai\AppData\Local\Temp\ipykernel_19512\1735868014.py:51: FutureWarning: DataFrameGroupBy.apply operated on the group
ing columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the oper
ation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to s
ilence this warning.
  comparison_table = enroll_df.groupby("state").apply(get_full_analysis).reset_index()
```

In [88]:

```
# # To understand the baseline of our data, we first calculated the central tendency and variability for each state. This summ
# Mean (Average): The average number of enrollments per pincode per day. It represents the "typical" activity level of a singl
# Median (Middle Value): The central value of our dataset. We use this to check for "skewness." Since our Median is often lowe
# Standard Deviation (\sigma): This measures the "spread" or "swing" of the data. A higher standard deviation indicates that e

# Prediction vs. Actual: In many states (like Assam), the "Predicted Next Year" is significantly higher than the "Actual Curre
# Statistical Warning: This model is a mathematical estimate based on past performance. It assumes "stationarity"—meaning it d
# Significance: With a 95% confidence level, we are statistically certain that the true enrollment total will fall within the
```

In [89]:

```python
import numpy as np
import pandas as pd
from scipy import stats

mean = full_data["source"]=="biometric"



# 1. Clean column names to prevent key errors
full_data.columns = full_data.columns.str.strip()

# 2. Filter for enrollment records only
biometric_df = full_data[full_data["source"].str.lower() == "biometric"]

def get_prediction_interval(group, confidence=0.95):
    """Calculates the mean prediction and its 95% interval for a group."""
    data = group["total_no_of_people"]
    n = len(data)

    if n < 2:
        return pd.Series({'Next_Year_Pred': data.mean(), 'Lower_Bound': np.nan, 'Upper_Bound': np.nan})

    mean = np.mean(data)
    std_dev = np.std(data, ddof=1)
    se = std_dev / np.sqrt(n)
```

```
    # T-critical value for n-1 degrees of freedom
    t_crit = stats.t.ppf((1 + confidence) / 2, df=n-1)

    # Prediction Interval Margin of Error
    # Formula: t * std_dev * sqrt(1 + 1/n)
    margin_error = t_crit * std_dev * np.sqrt(1 + (1/n))

    return pd.Series({
        'Next_Year_Pred': round(mean, 0),
        'Lower_Bound': round(mean - margin_error, 0),
        'Upper_Bound': round(mean + margin_error, 0)
    })

# 3. Apply the calculation to every state
state_forecasting = biometric_df.groupby("state").apply(get_prediction_interval).reset_index()

print(state_forecasting)
```

	state	Next_Year_Pred	Lower_Bound	Upper_Bound
0	Andaman & Nicobar Islands	4	-31	40
1	Andaman And Nicobar Islands	14	-60	88
2	Andhra Pradesh	22	-148	191
3	Arunachal Pradesh	17	-78	112
4	Assam	21	-108	149
5	Bihar	59	-358	476
6	Chandigarh	45	-466	556
7	Chhatisgarh	1	1	1
8	Chhattisgarh	83	-443	608
9	Dadra And Nagar Haveli	65	-487	618
10	Dadra And Nagar Haveli And Daman And Diu	11	-84	107
11	Daman And Diu	13	-69	95
12	Delhi	141	-1118	1400
13	Goa	13	-67	92
14	Gujarat	36	-258	329
15	Haryana	62	-434	558
16	Himachal Pradesh	13	-69	95
17	Jammu And Kashmir	40	-221	300
18	Jharkhand	55	-390	501
19	Karnataka	19	-112	149
20	Kerala	16	-86	119
21	Ladakh	8	-33	49
22	Lakshadweep	9	-36	53
23	Madhya Pradesh	85	-578	747
24	Maharashtra	61	-348	470
25	Manipur	43	-350	436
26	Meghalaya	21	-223	265
27	Mizoram	36	-306	377
28	Nagaland	29	-183	241
29	Odisha	25	-146	195
30	Pondicherry	14	-53	82
31	Punjab	36	-274	347
32	Rajasthan	50	-251	351
33	Sikkim	10	-51	70
34	Tamil Nadu	25	-138	189
35	Tamilnadu	1	NaN	NaN
36	Telangana	21	-108	150
37	Tripura	34	-200	269
38	Uttar Pradesh	62	-386	509
39	Uttarakhand	34	-211	278

```
40                          Uttaranchal              1           1           1
41                          West Bengal             19        -115         154
```

```
C:\Users\sohai\AppData\Local\Temp\ipykernel_19512\134959454.py:40: FutureWarning: DataFrameGroupBy.apply operated on the groupi
ng columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the opera
tion. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to si
lence this warning.
  state_forecasting = biometric_df.groupby("state").apply(get_prediction_interval).reset_index()
```

In [ ]:

In [90]:

```
def get_accurate_analysis(group):
    # 1. Calculate Daily Totals for the State
    # This sums up all people across all pincodes for each specific date
    daily_state_totals = group.groupby('date')['total_no_of_people'].sum()

    # 2. Statistics based on Daily State performance
    avg_daily_state = daily_state_totals.mean()
    std_daily_state = daily_state_totals.std(ddof=1)
    n_days = len(daily_state_totals) # Actual days of data we have

    # 3. Prediction for Next Year (365 days)
    # Instead of scaling by pincodes, we scale the state's daily average
    prediction_next_year = avg_daily_state * 365

    # 4. Accurate Margin of Error
    # We use the variance of the DAILY TOTALS, which is much more stable
    t_crit = stats.t.ppf(0.975, df=n_days-1)
    standard_error = std_daily_state / np.sqrt(n_days)
    margin_of_error = t_crit * standard_error * 365



    return pd.Series({
        'Days_of_Data': n_days,
        'Avg_Daily_People_State': int(avg_daily_state),
        'Actual_Current_Year_Total': int(group['total_no_of_people'].sum()),
        'Predicted_Next_Year': int(prediction_next_year),
        'Confidence_Interval_Low': (prediction_next_year - margin_of_error),
        'Confidence_Interval_High': (prediction_next_year + margin_of_error)
    })
```

```
pd.options.display.float_format = '{:.0f}'.format
# Run the group by
comparison_table = biometric_df.groupby("state").apply(get_accurate_analysis).reset_index()
print(comparison_table)
# Add this before you print to make numbers readable
```

```
                                        state  Days_of_Data  Avg_Daily_People_State  Actual_Current_Year_Total  Predicted_Next_Y
ear  Confidence_Interval_Low  Confidence_Interval_High
0                   Andaman & Nicobar Islands            81                      29                       2384                10
742                     7125                     14361
1                 Andaman And Nicobar Islands            82                     223                      18314                81
519                    39356                    123683
2                              Andhra Pradesh            88                   42211                    3714633             15407
284                  6990879                  23823690
3                           Arunachal Pradesh            87                     832                      72394               303
721                   164673                    442771
4                                       Assam            88                   11167                     982722              4076
062                  2364131                   5787995
5                                       Bihar            89                   55029                    4897587             20085
609                  8769906                  31401313
6                                  Chandigarh            86                     866                      74482               316
115                    25398                    606833
7                                 Chhatisgarh             5                       1                          5
365                      365                       365
8                                Chhattisgarh            89                   29761                    2648729             10862
765                  5812861                  15912669
9                      Dadra And Nagar Haveli            87                     319                      27788               116
581                    57534                    175629
10  Dadra And Nagar Haveli And Daman And Diu            78                      32                       2532                11
848                     3609                     20088
11                              Daman And Diu            84                     106                       8948                38
881                    18618                     59144
12                                      Delhi            88                   14822                    1304362              5410
137                  2008439                   8811837
13                                        Goa            87                     786                      68397               286
952                   139191                    434714
14                                    Gujarat            89                   35915                    3196514             13109
298                  5380268                  20838330
15                                    Haryana            88                   18584                    1635454              6783
417                  3198842                  10367992
16                           Himachal Pradesh            88                    4502                     396234              1643
470                   841884                   2445057
17                          Jammu And Kashmir            88                    8995                     791647              3283
535                  1725755                   4841317
18                                  Jharkhand            89                   22767                    2026297              8310
094                  3309314                  13310875
19                                  Karnataka            89                   29617                    2635954             10810
```

373	5000405		16620341				
20		Kerala		88	18292	1609730	6676
721	2948276		10405166				
21		Ladakh		86	67	5763	24
459	11101		37818				
22		Lakshadweep		84	57	4820	20
944	9783		32105				
23		Madhya Pradesh		89	66559	5923771	24294
117	9634156		38954078				
24		Maharashtra		89	103664	9226139	37837
536	20649558		55025514				
25		Manipur		87	3248	282587	1185
566	441374		1929758				
26		Meghalaya		87	1007	87626	367
626	152597		582656				
27		Mizoram		87	1383	120329	504
828	28715		980942				
28		Nagaland		87	1259	109593	459
786	210558		709015				
29		Odisha		88	28010	2464960	10223
981	4825317		15622647				
30		Pondicherry		88	794	69908	289
959	177528		402390				
31		Punjab		88	19768	1739671	7215
680	3088777		11342584				
32		Rajasthan		89	44887	3994955	16383
804	8610467		24157142				
33		Sikkim		84	271	22820	99
158	34754		163562				
34		Tamil Nadu		89	52787	4698117	19267
558	9821735		28713382				
35		Tamilnadu		1	1	1	
365	NaN		NaN				
36		Telangana		88	19746	1737654	7207
314	4025685		10388945				
37		Tripura		89	3282	292155	1198
163	457506		1938821				
38		Uttar Pradesh		89	107615	9577735	39279
475	19327715		59231235				
39		Uttarakhand		89	8592	764763	3136
387	1967922		4304853				

```
40                              Uttaranchal            2                       1                          2
365                    365                      365
41                              West Bengal           88                   28688                    2524619            10471
431                5096637                 15846225
```

```
C:\Users\sohai\AppData\Local\Temp\ipykernel_19512\1090604605.py:33: FutureWarning: DataFrameGroupBy.apply operated on the group
ing columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the oper
ation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to s
ilence this warning.
  comparison_table = biometric_df.groupby("state").apply(get_accurate_analysis).reset_index()
```

In [91]:

```python
# ========================================================================================
# ANALYSIS SUMMARY: YEARLY BIOMETRIC PROJECTION REPORT
# ========================================================================================
# WHO THIS SHOWS:
# This table analyzes biometric registration trends categorized by "State."
# It treats each state as an independent group to account for regional differences.
#
# WHAT THE COLUMNS MEAN:
# 1. Days_of_Data: Total number of unique calendar days recorded for this state.
#    (Most show ~89 days, meaning we are using one quarter's data for the forecast).
#
# 2. Avg_Daily_People_State: The "Daily Pulse." This is the average number of
#    total people registered across the entire state in a single day.
#
# 3. Actual_Current_Year_Total: The raw sum of all people registered in the
#    available data so far.
#
# 4. Predicted_Next_Year: The "Full Year Forecast." We take the 'Daily Pulse'
#    and multiply it by 365 days to estimate next year's total.
#
# 5. Confidence_Interval_Low/High: The "Risk Range." Since data varies day-to-day,
#    these columns show the range where the true yearly total is likely to fall.
#    - LOW: The conservative estimate (Worst Case Scenario).
#    - HIGH: The aggressive estimate (Best Case Scenario).
#
# WHY THIS IS ACCURATE:
# Instead of assuming every pincode is active every day, this method looks at
# the state's total output per day. This naturally adjusts for pincodes that
# might be closed or inactive on certain dates.
# ========================================================================================
```

In [92]:
```
# ==========================================================================================
# FINAL PROJECT CONCLUSION: UIDAI BIOMETRIC REGISTRATION ANALYSIS & FORECASTING
# Project: UIDAI Hackathon Data Analysis
# Prepared by: Sohail Kundgol
# ==========================================================================================

# ------------------------------------------------------------------------------------------
# PHASE 1: DATA CLEANING & PRE-PROCESSING INSIGHTS
# ------------------------------------------------------------------------------------------
# 1. DATA INTEGRITY: The raw dataset was processed to handle missing values and
#    standardize geographic labels (States/Pincodes).
# 2. TEMPORAL MAPPING: Data was localized into daily chunks. This was critical because
#    biometric data is highly dependent on 'Working Days' vs 'Holidays'.
# 3. NOISE REDUCTION: By grouping by 'State' and 'Date' first, we removed the bias
#    caused by individual pincodes that only had sporadic or one-time data entries.

# ------------------------------------------------------------------------------------------
# PHASE 2: KEY STATISTICAL FINDINGS (THE "WHY")
# ------------------------------------------------------------------------------------------
# 1. VOLUME LEADERS: Larger states (e.g., Maharashtra, Uttar Pradesh) show high
#    daily averages (Avg_Daily_People_State), indicating high infrastructure demand.
# 2. VARIANCE ANALYSIS: The 'Margin of Error' reveals which states are stable and
#    which are volatile. A high Margin of Error suggests that registration
#    numbers swing wildly day-to-day, possibly due to local events or connectivity.
# 3. PINCODE DENSITY: Your analysis shows that registration is not evenly spread;
#    certain 'Hub' pincodes handle 80% of the load, while others are underutilized.

# ------------------------------------------------------------------------------------------
# PHASE 3: YEARLY PROJECTIONS (THE "SO WHAT")
# ------------------------------------------------------------------------------------------
# 1. RUN-RATE FORECAST: By taking the daily pulse of each state and scaling to 365 days,
#    we now have a data-driven "Predicted_Next_Year" target.
# 2. CAPACITY PLANNING: The 'Confidence_Interval_High' value should be used as the
#    maximum capacity target. If a state hits this number, the current centers
#    will likely experience long wait times or system crashes.
# 3. OPERATIONAL BUFFER: The difference between the 'Actual_Current' and 'Predicted'
#    highlights the massive scale-up required for the upcoming year.

# ------------------------------------------------------------------------------------------
# PHASE 4: VISUALIZATION SUMMARY & TRENDS
```

```
# -------------------------------------------------------------------------------------
# 1. HEATMAPS/CHARTS: The charts in this notebook confirm a 'Clustering Effect.'
#    Registrations peak during specific windows, suggesting a need for
#    dynamic resource allocation (moving mobile units where demand is high).
# 2. OUTLIER DETECTION: Some states show extreme peaks. These are likely
#    enrollment drives. Future models should separate 'Baseline' data from 'Drive' data.

# -------------------------------------------------------------------------------------
# PHASE 5: FINAL RECOMMENDATIONS FOR STAKEHOLDERS
# -------------------------------------------------------------------------------------
# 1. RESOURCE ALLOCATION: Deploy more biometric kits to states where the
#    'Confidence_Interval_High' is significantly higher than the current average.
# 2. PINCODE OPTIMIZATION: In states with many pincodes but low daily averages,
#    consider consolidating centers to save costs without reducing service.
# 3. DATA-DRIVEN POLICY: Use the 'Predicted_Next_Year' totals to set
#    performance KPIs for regional offices.

# =====================================================================================
# END OF ANALYSIS: The logic used here ensures that the UIDAI system is not just
# reacting to past data, but is prepared for future demand with 95% confidence.
# =====================================================================================
```