Introduction to this Notebook

Here I will be performing the analysis for the final project. It includes looking at the e-way bill data (a proxy for merchandise trade) and then inferring things about states based on that data.

This data will also be conbined with employment datasets to see if there is a commensurate rise in employment with an increase in trade, as represented by the E-Way Bill data.

All the data processing work is done in this notebook itself - which makes it longer and slightly more onerous to read. However, it allows better visualization of the process and so I've chosen not to separate it.

1. Setting up the Imports

In [1]:
```
import polars as pl
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.dates as mdates
import ipywidgets as widgets
from ipywidgets import interact
import requests
import io
from shapely import wkt
import numpy as np
from datetime import datetime
import textwrap
```

2. Reading the E-way bill data (contained in the Datasets folder)

In [2]:
```
e_way_bill_data = pl.read_csv("Datasets/India_EWay_Bills.csv")
```

The dataset contains 7 unique values for e-way bill data in the column "Type of Supply". This can be broken down into 3 categories: Incoming Trade, Outgoing Trade, and Trade within a state

These 3 types of trade patterns will be studied in my analysis

I will also be combining Incoming and Outgoing Trade later to form a field called External Trade. This, I will contrast with Trade within a State - which I will call Internal Trade

Note that E-way bills are recording merchandise trade of value over Rs. 50,000

3. First Generating some All India Level Analysis

There is no "All-India" value in the State/UT name column in the E-way bill dataset

Thus, I will aggregate the data over a month for all states and union territories, sum up the values, and generate all India level insights based on that

In [3]:
```
"""
Below, I do the following
1. Choosing the required columns and renaming the ones with long and complicated na
2. Grouping the columns by month
3. Aggregating the value of required columns
4. Creating a proper date column to sort the data and use for visualizations later
5. Sorting the dataframe based on the new date column
6. Displaying the result at the end to verify if the steps worked
"""

e_way_all_india = e_way_bill_data.select(
    pl.col("Month"),
    pl.col("State/Ut Name"),
    pl.col("E-Way Suppliers (UOM:Number), Scaling Factor:1").alias("Total E-way Bil
    pl.col("E-Way Bills (UOM:Number), Scaling Factor:1").alias("Count of E-way Bill
    pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000").ali
).group_by(
    pl.col("Month")
).agg(
    pl.col("Total E-way Bill Suppliers").sum(),
    pl.col("Count of E-way Bill Generated").sum(),
    pl.col("Total E-way Bill Value").sum()
).with_columns(
    Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
).select(
    ["Formatted_Date", "Total E-way Bill Suppliers", "Count of E-way Bill Generated
).sort(
    pl.col("Formatted_Date")
)

e_way_all_india
```

Out[3]: shape: (84, 4)

Formatted_Date	Total E-way Bill Suppliers	Count of E-way Bill Generated	Total E-way Bill Value
date	i64	i64	f64
2018-07-01	2785209	60068938	1.8275e6
2018-08-01	2984687	67813434	2.0581e6
2018-09-01	2995804	66976167	2.0537e6
2018-10-01	3254616	73984944	2.2093e6
2018-11-01	3041802	61277703	1.9384e6
…	…	…	…
2025-02-01	4711826	148624718	3.9423e6
2025-03-01	4927584	165391900	4.6115e6
2025-04-01	4725923	158629324	4.0815e6
2025-05-01	4779080	162174573	4.2123e6
2025-06-01	4669897	157695640	4.0715e6

3.1 Analyzing E-Way Bill Suppliers and seeing its evolution over time

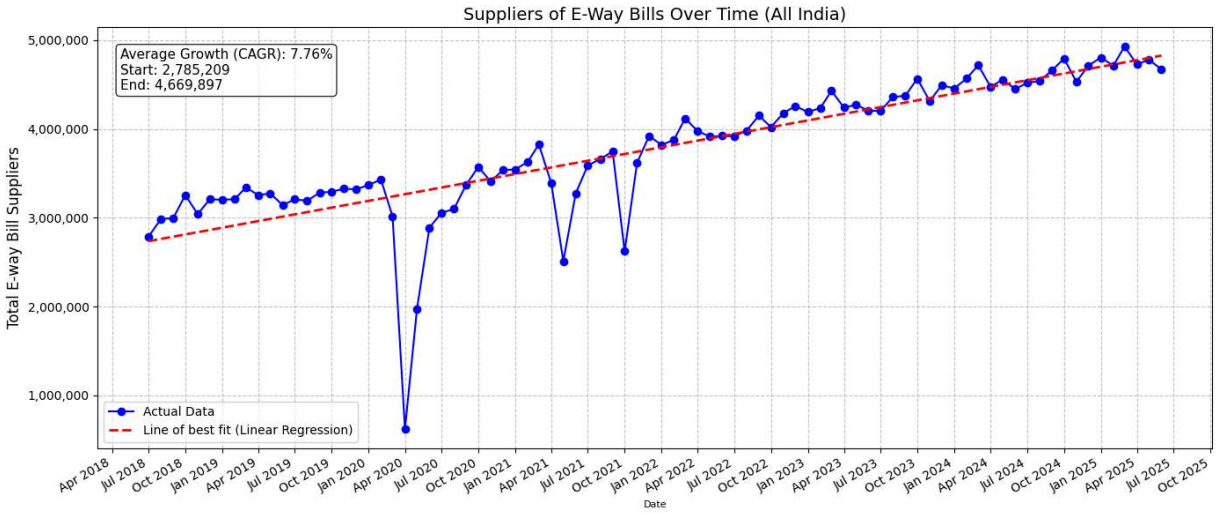
The e-way bill suppliers indicate the adoption of e-way bill across the country (by counting the number of businesses using the system)

Seeing the evolution of usage rate over time will help me discern if the system is seeing more or less usage over time.

In [4]:
```
"""
Doing the following steps below:
1. Getting the x and y values to generate a simple time-series graph from the dataf

2. Converting the dates into numbers to do a linear regression
3. Fitting a degree 1 polynomial `y=mx+c` to the data points
4. Creating a function `p` that represents the straight line that I genarated.
5. Feeding the date numbers back into the equation to generate the ideal Y-values -

6. Using the x and y values that I extracted in step 1 to do a calculation of CAGR
- My x axis is currently structured as 1-month-year
- Thus I subtract the first and last x dates and divide the number by 365.25 to get
- Then I feed the number of years as an argument into the equation for cagr

7. Finally, I perform the visualization
"""
```

```
x_dates = e_way_all_india["Formatted_Date"].to_list()
y_values = e_way_all_india["Total E-way Bill Suppliers"].to_list()

x_nums = mdates.date2num(x_dates)
z = np.polyfit(x_nums, y_values, 1)
p = np.poly1d(z)
trend_line_y = p(x_nums)

# CAGR Calculation
start_val = y_values[0]
end_val = y_values[-1]
num_years = (x_dates[-1] - x_dates[0]).days / 365.25
cagr = (end_val / start_val) ** (1 / num_years) - 1



# ------------------------------------ Visualization Starts Here ----------------

"""
VISUALIZATION STEPS:

1. Plot Actual Data:
- Uses plt.plot to graph 'Formatted_Date' (X) vs 'Suppliers of E-way Bills' (Y).
- Styles the line with blue color ('b'), solid style ('-'), and circle markers ('o'

2. Plot Trend Line:
- Plots the regression line calculated earlier (x_dates vs trend_line_y).
- Styles it as a red dashed line ('--') to distinguish it from actual data.

3. Add Statistics Box:
- Formats a string with the calculated CAGR (to 2 decimal places), Start Value, and
- Places a text box at coordinates (0.02, 0.95) (top-left relative to axes).
- Adds a white, semi-transparent background box for readability.

4. Format X-Axis (Dates):
- Formats tick labels as 'Month Year'.
- Sets the locator to show a label only every 3 months to reduce clutter.
- Auto-rotates dates (autofmt_xdate) to prevent overlapping.

5. Format Y-Axis (Values):
- Applies a string formatter to add commas to large numbers (e.g., 1,000,000).

6. Final Layout & Labels:
- Sets the chart Title and X/Y axis labels.
- Adds a grid for easier readability of values.
- Adds a legend to identify the two lines.
- Uses tight_layout() to prevent labels from being cut off before showing the plot.
"""

plt.figure(figsize=(14, 6))

plt.plot(e_way_all_india["Formatted_Date"], e_way_all_india["Total E-way Bill Suppl

# Plot Trend Line
plt.plot(x_dates, trend_line_y, color='red', linestyle='--', linewidth=2, label='Li
```

```
# Add Text Box with Stats
text_str = f"Average Growth (CAGR): {cagr:.2%}\nStart: {start_val:,.0f}\nEnd: {end_
plt.gca().text(0.02, 0.95, text_str, transform=plt.gca().transAxes, fontsize=11,
               verticalalignment='top', bbox=dict(boxstyle='round', facecolor='whit

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gcf().autofmt_xdate()

plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.title("Suppliers of E-Way Bills Over Time (All India)", fontsize=14)
plt.xlabel("Date", fontsize=8)
plt.ylabel("Total E-way Bill Suppliers", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()

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



The above result clearly shows a consistent rise in the number of entities using the E-way bill system since 2018, even when considering the shocks to the system due to Covid-19

The CAGR for this period is higher than the rate of GDP growth. Therefore, entities engaged in registered-bulk-merchandise-movement grew faster than the overall economy, indicating greater formalization of the system

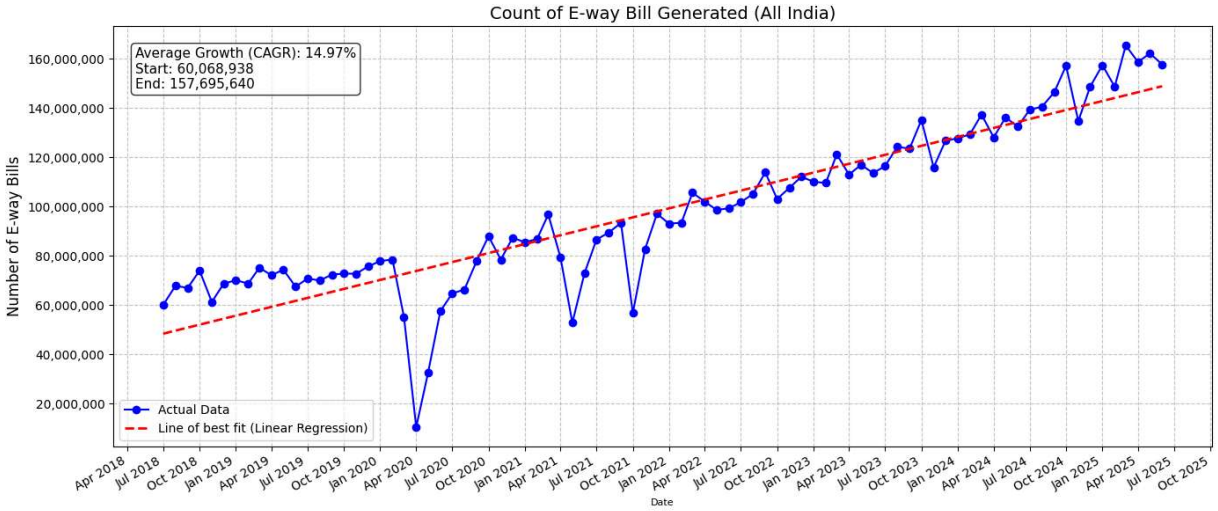
3.2 Analyzing count of E-Way Bills and its evolution over time

This analysis, coupled with the one above, would help me understand if there was growing usage of the system within entities that were on the E-way platform

In other words, we saw the rate of growth of entities on the E-way bill platform above. If the rate of growth of bills generated outpaces entity registration on the platform, then I would be able to conclude that there is deepening usage of the system among businesses

In [5]:
```
"""
Doing the following steps below:
1. Getting the x and y values to generate a simple time-series graph from the dataf

2. Converting the dates into numbers to do a linear regression
3. Fitting a degree 1 polynomial `y=mx+c` to the data points
4. Creating a function `p` that represents the straight line that I genarated.
5. Feeding the date numbers back into the equation to generate the ideal Y-values -

6. Using the x and y values that I extracted in step 1 to do a calculation of CAGR
- My x axis is currently structured as "1-month-year"
- Thus I subtract the first and last x dates and divide the number by 365.25 to get
- Then I feed the number of years as an argument into the equation for cagr

7. Finally, I perform the visualization
"""

x_dates = e_way_all_india["Formatted_Date"].to_list()
y_values = e_way_all_india["Count of E-way Bill Generated"].to_list()

x_nums = mdates.date2num(x_dates)
z = np.polyfit(x_nums, y_values, 1)
p = np.poly1d(z)
trend_line_y = p(x_nums)

# CAGR Calculation
start_val = y_values[0]
end_val = y_values[-1]
num_years = (x_dates[-1] - x_dates[0]).days / 365.25
cagr = (end_val / start_val) ** (1 / num_years) - 1



# ------------------------------------ Visualization Starts Here ----------------

"""
VISUALIZATION STEPS:

1. Plot Actual Data:
- Uses plt.plot to graph 'Formatted_Date' (X) vs 'Count of E-way Bills' (Y).
- Styles the line with blue color ('b'), solid style ('-'), and circle markers ('o'

2. Plot Trend Line:
- Plots the regression line calculated earlier (x_dates vs trend_line_y).
- Styles it as a red dashed line ('--') to distinguish it from actual data.

3. Add Statistics Box:
```

```
- Formats a string with the calculated CAGR (to 2 decimal places), Start Value, and
- Places a text box at coordinates (0.02, 0.95) (top-left relative to axes).
- Adds a white, semi-transparent background box for readability.

4. Format X-Axis (Dates):
- Formats tick labels as 'Month Year'.
- Sets the locator to show a label only every 3 months to reduce clutter.
- Auto-rotates dates (autofmt_xdate) to prevent overlapping.

5. Format Y-Axis (Values):
- Applies a string formatter to add commas to large numbers (e.g., 1,000,000).

6. Final Layout & Labels:
- Sets the chart Title and X/Y axis labels.
- Adds a grid for easier readability of values.
- Adds a legend to identify the two lines.
- Uses tight_layout() to prevent labels from being cut off before showing the plot.
"""

plt.figure(figsize=(14, 6))

plt.plot(e_way_all_india["Formatted_Date"], e_way_all_india["Count of E-way Bill Ge

# Plot Trend Line
plt.plot(x_dates, trend_line_y, color='red', linestyle='--', linewidth=2, label='Li

# Add Text Box with Stats
text_str = f"Average Growth (CAGR): {cagr:.2%}\nStart: {start_val:,.0f}\nEnd: {end_
plt.gca().text(0.02, 0.95, text_str, transform=plt.gca().transAxes, fontsize=11,
               verticalalignment='top', bbox=dict(boxstyle='round', facecolor='whit

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gcf().autofmt_xdate()

plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.title("Count of E-way Bill Generated (All India)", fontsize=14)
plt.xlabel("Date", fontsize=8)
plt.ylabel("Number of E-way Bills", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()

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



The above graph and the CAGR value shows that the count of e-way bills generated per month has grown much faster than the entities registered on the platform

This shows that the evidence for greater usage of the E-way bill system among registered entities is quite strong

3.3 Analyzing Value of E-way Bill Value over time

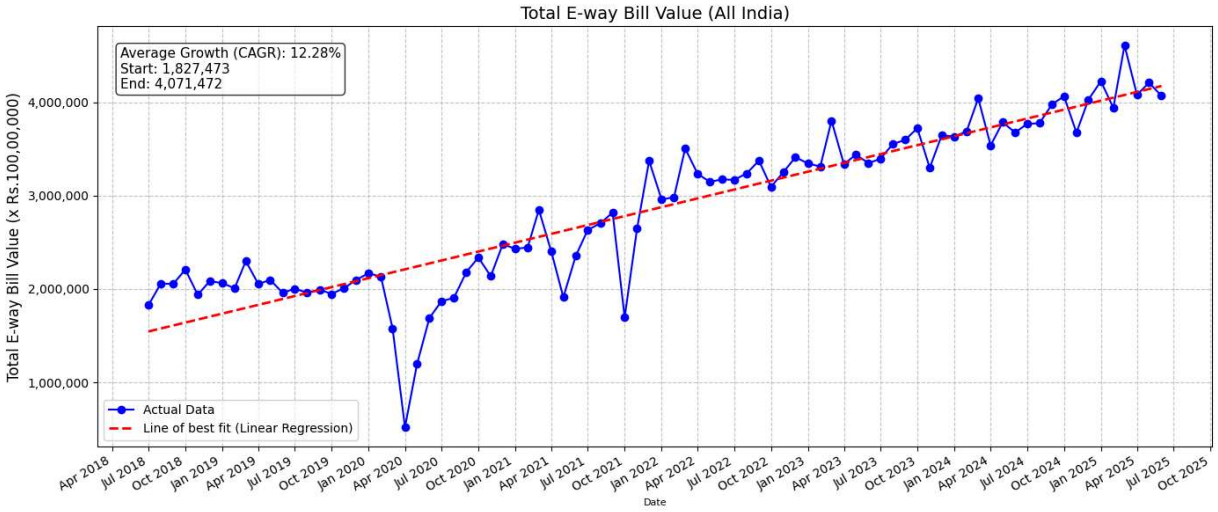
Now that I can infer that the number of E-way bills generated has grown very fast, I wanted to see if the ticket size (value of each bill) is growing at the same pace

Difference in the count of e-way bill growth vs the value of e-way bill growth will indicate that the ticket size is either decreasing (businesses are generating bills for smaller consignments) or increasing (businesses are generating bills for larger consignments)

In [6]:
```
"""
Doing the following steps below:
1. Getting the x and y values to generate a simple time-series graph from the dataf

2. Converting the dates into numbers to do a linear regression
3. Fitting a degree 1 polynomial `y=mx+c` to the data points
4. Creating a function `p` that represents the straight line that I genarated.
5. Feeding the date numbers back into the equation to generate the ideal Y-values -

6. Using the x and y values that I extracted in step 1 to do a calculation of CAGR
- My x axis is currently structured as "1-month-year"
- Thus I subtract the first and last x dates and divide the number by 365.25 to get
- Then I feed the number of years as an argument into the equation for cagr

7. Finally, I perform the visualization
"""

x_dates = e_way_all_india["Formatted_Date"].to_list()
y_values = e_way_all_india["Total E-way Bill Value"].to_list()

x_nums = mdates.date2num(x_dates)
```

```
z = np.polyfit(x_nums, y_values, 1)
p = np.poly1d(z)
trend_line_y = p(x_nums)

# CAGR Calculation
start_val = y_values[0]
end_val = y_values[-1]
num_years = (x_dates[-1] - x_dates[0]).days / 365.25
cagr = (end_val / start_val) ** (1 / num_years) - 1



# ------------------------------------ Visualization Starts Here ---------------

"""
VISUALIZATION STEPS:

1. Plot Actual Data:
- Uses plt.plot to graph 'Formatted_Date' (X) vs 'Total E-way Bill Value' (Y).
- Styles the line with blue color ('b'), solid style ('-'), and circle markers ('o'

2. Plot Trend Line:
- Plots the regression line calculated earlier (x_dates vs trend_line_y).
- Styles it as a red dashed line ('--') to distinguish it from actual data.

3. Add Statistics Box:
- Formats a string with the calculated CAGR (to 2 decimal places), Start Value, and
- Places a text box at coordinates (0.02, 0.95) (top-left relative to axes).
- Adds a white, semi-transparent background box for readability.

4. Format X-Axis (Dates):
- Formats tick labels as 'Month Year'.
- Sets the locator to show a label only every 3 months to reduce clutter.
- Auto-rotates dates (autofmt_xdate) to prevent overlapping.

5. Format Y-Axis (Values):
- Applies a string formatter to add commas to large numbers (e.g., 1,000,000).

6. Final Layout & Labels:
- Sets the chart Title and X/Y axis labels.
- Adds a grid for easier readability of values.
- Adds a legend to identify the two lines.
- Uses tight_layout() to prevent labels from being cut off before showing the plot.
"""

plt.figure(figsize=(14, 6))

plt.plot(e_way_all_india["Formatted_Date"], e_way_all_india["Total E-way Bill Value

# Plot Trend Line
plt.plot(x_dates, trend_line_y, color='red', linestyle='--', linewidth=2, label='Li

# Add Text Box with Stats
text_str = f"Average Growth (CAGR): {cagr:.2%}\nStart: {start_val:,.0f}\nEnd: {end_
plt.gca().text(0.02, 0.95, text_str, transform=plt.gca().transAxes, fontsize=11,
               verticalalignment='top', bbox=dict(boxstyle='round', facecolor='whit
```

```
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gcf().autofmt_xdate()

plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.title("Total E-way Bill Value (All India)", fontsize=14)
plt.xlabel("Date", fontsize=8)
plt.ylabel("Total E-way Bill Value (x Rs.100,00,000)", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()

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



We can see here that the value of e-way bills has grown slightly slower than the count of e-way bills over time

This indicates that the e-way bills are slowly increasing in their ticket size (slightly larger bills are getting generated).

Transitioning to State Level Analysis

The most important initial requirement for this section is to extract state level data and store it in a data structure in memory for quick use

The code blocks below will focus on gathering this state level data - directly from datasets + derived metrics from calculations

1. Getting the names of States from the E-way bill Dataset

This step is crucial as it helps us store the data in memory + gives us an idea of how states and union territory names are represented in this dataset.

Other datasets used later on have different names for the same regions, and so obtaining this information is crucial for creating a mapping

In [7]:
```
indian_states = e_way_bill_data["State/Ut Name"].str.strip_chars().unique().sort().
indian_states = [state.strip() for state in indian_states]
```

2. Calculating Important Metrics for Analysis

2.1. Creating functions for finding CAGR (Compound Annual Growth Rate) of Merchandise Trade across various dimensions including

a. Incoming Trade

b. Outgoing Trade

c. Internal Trade (Within Trade)

d. Total Trade

Methodology of finding Trade CAGR in the function below

i) *First the overall dataset is taken and for each subcomponent (incoming / outgoing / within), a filtering of states is done.*

ii) *Then only the required columns are taken (month, count of e-way bills, value of e-way bills)*

iii) *The month column is then cleaned and converted to datetime format. This is stored in a new column called Formatted Date*

iv) *3 columns are then chosen (Formatted Date, count of e-way bills, value of e-way bills)*

v) *The dataframe is then sorted by the Date column in ascending order*

vi) *Using the sorted dataframe, the first and last 12 months of data is taken for value and count of e-way bills*

vii) *These first and last 12 months of data is then summed up and stored for calculation of CAGR*

viii) *The length of the data series is then calculated. CAGR calculation uses this dynamic field.*

ix) *CAGR is then calculated for count and value of e-way bills, and represented as a percentage*

x) *These calculated values are returned at the end of the function.*

In the function below, the total Trade CAGR `find_cagr_total` (combining internal + external + within trade) is calculated as follows:

i) *Filter overall dataset to find all the trade data for the given state*

ii) *Select the columns of month, count and value of e-way bills*

iii) *Create a Formatted Date column (datetime) using the Month column*

iv) *Format the dataframe using the Formatted Date column*

v) *Group the dataframe using the Formatted Date column, and sum the value of count and value for each month. This will allow us to calculate the sum of count and value for incoming + outgoing + within trade*

vi) *Calculate the CAGR as above, using the first and last 12 months of data, and return the result*

In [8]:
```
def calculate_robust_cagr(df, value_col):
    # Sort by date to ensure head/tail are actually first/last years
    df = df.sort("Formatted_Date")

    # Needs at least 24 months of data to calculate a reliable multi-year CAGR
    if len(df) < 24:
        return 0.0

    sum_first_year = df[value_col].head(12).sum()
    sum_last_year = df[value_col].tail(12).sum()

    # Prevent division by zero or extreme low-base effect
    if sum_first_year < 1.0:
        return 0.0

    # Calculate years, clamping to a minimum of 1 to prevent exponent explosion
    n_years = max(1, (len(df) - 12) / 12)

    cagr = (((sum_last_year / sum_first_year) ** (1/n_years)) - 1) * 100
    return cagr


def find_incoming_CAGR(state_name):
    df_incoming = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
        pl.col("Type Of Supply").is_in(["INCOMING FROM OTHER STATES", "Inter State
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    )

    val_cagr = calculate_robust_cagr(df_incoming, "Assessable Value (UOM:INR(Indian
    count_cagr = calculate_robust_cagr(df_incoming, "E-Way Bills (UOM:Number), Scal

    return (val_cagr, count_cagr)

def find_outgoing_CAGR(state_name):
    df_outgoing = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
```

```
        pl.col("Type Of Supply").is_in(["OUTGOING TO OTHER STATES", "Inter State Ou
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    )

    val_cagr = calculate_robust_cagr(df_outgoing, "Assessable Value (UOM:INR(Indian
    count_cagr = calculate_robust_cagr(df_outgoing, "E-Way Bills (UOM:Number), Scal

    return (val_cagr, count_cagr)

def find_within_CAGR(state_name):
    df_within = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
        pl.col("Type Of Supply").is_in(["WITHIN-STATE", "Intra State Supplies", "IN
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    )

    val_cagr = calculate_robust_cagr(df_within, "Assessable Value (UOM:INR(IndianRu
    count_cagr = calculate_robust_cagr(df_within, "E-Way Bills (UOM:Number), Scalin

    return (val_cagr, count_cagr)

def find_cagr_total(state_name):
    # 1. Filter for the State (All Supply Types)
    df_total = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    )

    # 2. Group by Date to sum Incoming + Outgoing + Within for each month
    # This is critical so that 'head(12)' represents 12 months, not 12 rows
    df_grouped = df_total.group_by("Formatted_Date").agg(
        pl.col("E-Way Bills (UOM:Number), Scaling Factor:1").sum().alias("Total_Cou
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).sort("Formatted_Date")

    # 3. Calculate CAGR using the Aggregated Columns
    val_cagr = calculate_robust_cagr(df_grouped, "Total_Value")
    count_cagr = calculate_robust_cagr(df_grouped, "Total_Count")

    return (val_cagr, count_cagr)
```

The above code block allows us to obtain the CAGR for all forms of merchandise trade - incoming, outgoing, internal, external. Finally the CAGR for the overall merchandise trade (summing over all these parameters) is also calculated.

2.2 Finding the Absolute Value of E-way bills for a particular state

Since the E-way bill dataset is month by month, if graphs used just latest month of data, the comparison across states would not be fair due to the seasonality of

merchandise trade

To counter this, I am going to take the data for the last 12 months, and generate an average. This will remnove seasonality and allow a clearer cross-state comparison.

The function that I have defined below will calculate absolute values over the previous 12 months for external trade (incoming + outgoing), internal trade, total trade

The calculated metrics are then returned by the function at the end

In [9]:
```
"""
Implementation of absolute_value_of_eway_bills(state)

Calculates the External Trade Value, Internal Trade Value in the following manner:
- Filters 'e_way_bill_data' for the given state and "Type Of Supply" (Incoming or O
- Selects 'Month' and 'Assessable Value' columns.
- Converts 'Month' string to a standard Date object ('Formatted_Date').
- Sorts the dataframe using Formatted_Date
- Takes the last 12 months of data, sums their values, and divides by 12 to get the
- Returns a tuple containing three float values:
"""

def absolute_value_of_eway_bills(state):

    absolute_value_external = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state,
        pl.col("Type Of Supply").is_in(["OUTGOING TO OTHER STATES", "Inter State Ou
    ).select(
        pl.col("Month"),
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).sort("Formatted_Date").select(
        pl.col("Formatted_Date"),
        pl.col("Value of E-Way Bills")
    ).group_by("Formatted_Date").agg(
        pl.col("Value of E-Way Bills").sum()
    )

    e_way_external_value_12mo_avg = (absolute_value_external.tail(12).select(
        pl.col("Value of E-Way Bills")
    ).sum().item())/12


    absolute_value_internal = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state,
        pl.col("Type Of Supply").is_in(["WITHIN-STATE", "Intra State Supplies", "IN
    ).select(
        pl.col("Month"),
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).sort("Formatted_Date").select(
        pl.col("Formatted_Date"),
```

```
        pl.col("Value of E-Way Bills")
    ).group_by("Formatted_Date").agg(
        pl.col("Value of E-Way Bills").sum()
    )

    e_way_internal_value_12mo_avg = (absolute_value_internal.tail(12).select(
        pl.col("Value of E-Way Bills")
    ).sum().item())/12

    absolute_value_total = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state,
        pl.col("Type Of Supply").is_in(["OUTGOING TO OTHER STATES", "Inter State Ou
    ).select(
        pl.col("Month"),
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).sort("Formatted_Date").select(
        pl.col("Formatted_Date"),
        pl.col("Value of E-Way Bills")
    ).group_by("Formatted_Date").agg(
        pl.col("Value of E-Way Bills").sum()
    )

    e_way_total_value_12mo_avg = (absolute_value_total.tail(12).select(
        pl.col("Value of E-Way Bills")
    ).sum().item())/12


    return (e_way_external_value_12mo_avg, e_way_internal_value_12mo_avg, e_way_tot
```

2.3 Calculating all the remaining metrics for the states

The metrics that will be calculated below get used in the state dashboard, as well as the maps that are used for cross-state comparisons

`generate_state_e_way_bill_stats` will calculate and obtain the following metrics for a given state, storing them in a results dictionary:

1. *Extracting the **Incoming**, **Outgoing**, and **Within-State** trade data and storing them in separate dataframes containing the Formatted Date, Count, and Value of E-Way Bills.* This will be used later for granular analysis and specific visualizations below.

2. *Calculating the `correlation between the value of Incoming and Outgoing trade`. This helps determine if a state's import and export activities are synchronized (high correlation) or divergent (low correlation).*

3. *Calculating the `correlation between External trade (Incoming + Outgoing) and Internal (Within) trade`. This reveals if a state's domestic economic activity is growing in tandem with its cross-border trade.*

4. *Calculating the `correlation between the Value and Count` of E-Way bills for all three categories (Incoming, Outgoing, Within). This indicates whether the average "ticket size" of orders is consistent or fluctuating.*

5. *Obtaining the **12-month average absolute value** for External, Internal, and Total trade from the function above. This provides a baseline metric for the current scale of economic activity in the state.*

6. *Calculating the **CAGR (Compound Annual Growth Rate)** for both the **Value** and **Count** of E-Way bills across all categories (Incoming, Outgoing, Within, and Total). This data quantifies the growth trajectory of the state's trade and will be essential for visualization.*

In [10]:
```
"""
Implementation of generate_state_e_way_bill_stats(state_name)

1. Extracts 'Incoming', 'Outgoing', and 'Within-State' trade data. Cleans and forma
2. Correlation Analysis:
- Calculates correlation between Import (Incoming) and Export (Outgoing) trade valu
- Calculates correlation between External Trade (Incoming + Outgoing) and Internal
- Calculates correlation between the 'Count' and 'Value' of bills for each trade ca

3. Calls the function `absolute_value_of_eway_bills(state)` to get the 12-month ave
4. Computes the Compound Annual Growth Rate (CAGR) for both Value and Count across
5. Result Compilation: Aggregates all dataframes and calculated metrics into a sing
"""

def generate_state_e_way_bill_stats(state_name):

    # Extract Incoming Trade Data
    incoming_trade = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
        pl.col("Type Of Supply").is_in(["INCOMING FROM OTHER STATES", "Inter State
    ).select(
        pl.col(["Month", "E-Way Bills (UOM:Number), Scaling Factor:1", "Assessable
        pl.col("E-Way Bills (UOM:Number), Scaling Factor:1").alias("Count of E-Way
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).select(
        pl.col("Formatted_Date"),
        pl.col("Count of E-Way Bills"),
        pl.col("Value of E-Way Bills")
    ).sort(
        pl.col("Formatted_Date")
    )

    # Extract Outgoing Trade Data
    outgoing_trade = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
        pl.col("Type Of Supply").is_in(["OUTGOING TO OTHER STATES", "Inter State Ou
    ).select(
        pl.col(["Month", "E-Way Bills (UOM:Number), Scaling Factor:1", "Assessable
        pl.col("E-Way Bills (UOM:Number), Scaling Factor:1").alias("Count of E-Way
```

```
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).select(
        pl.col("Formatted_Date"),
        pl.col("Count of E-Way Bills"),
        pl.col("Value of E-Way Bills")
    ).sort(
        pl.col("Formatted_Date")
    )

    # Extract Within-State Trade Data
    within_trade = e_way_bill_data.filter(
        pl.col("State/Ut Name") == state_name,
        pl.col("Type Of Supply").is_in(["WITHIN-STATE", "Intra State Supplies", "IN
    ).select(
        pl.col(["Month", "E-Way Bills (UOM:Number), Scaling Factor:1", "Assessable
        pl.col("E-Way Bills (UOM:Number), Scaling Factor:1").alias("Count of E-Way
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
    ).with_columns(
        Formatted_Date = (pl.lit("1 ") + pl.col("Month")).str.to_date("%d %B, %Y")
    ).select(
        pl.col("Formatted_Date"),
        pl.col("Count of E-Way Bills"),
        pl.col("Value of E-Way Bills")
    ).sort(
        pl.col("Formatted_Date")
    )

    # Prepare data for Correlation: Incoming vs. Outgoing
    incoming_trade_new = incoming_trade.select(
        ["Formatted_Date", "Value of E-Way Bills"]
    ).with_columns(
        pl.col("Value of E-Way Bills").alias("Value of Incoming Trade")
    )

    outgoing_trade_new = outgoing_trade.select(
        ["Formatted_Date", "Value of E-Way Bills"]
    ).with_columns(
        pl.col("Value of E-Way Bills").alias("Value of Outgoing Trade")
    )

    # Merge Incoming and Outgoing data on Date
    incoming_plus_outgoing = incoming_trade_new.join(outgoing_trade_new, on="Format

    # Calculate Correlation: Import (Incoming) vs Export (Outgoing)
    correlation_bw_incoming_outgoing = incoming_plus_outgoing.select(
        pl.corr("Value of Incoming Trade", "Value of Outgoing Trade")
    ).item()

    # Prepare data for Correlation: External vs. Internal
    # First, combine Incoming and Outgoing into a single "External Trade" column
    incoming_plus_outgoing = incoming_plus_outgoing.with_columns(
        External_Trade =  pl.col("Value of Incoming Trade") + pl.col("Value of Outg
    ).select(
        ["Formatted_Date", "External_Trade"]
```

```
    )

    # Merge External Trade with Internal (Within) Trade data
    within_plus_external = within_trade.join(
        incoming_plus_outgoing,
        on="Formatted_Date",
        how="inner"
    ).with_columns(
        pl.col("Value of E-Way Bills").alias("Value of Within Trade")
    ).select(
        ["Formatted_Date", "External_Trade", "Value of Within Trade"]
    )

    # Calculate Correlation: External Trade vs Internal Trade
    correlation_bw_external_trade_and_within_trade = within_plus_external.select(
        pl.corr("External_Trade", "Value of Within Trade")
    ).item()

    # Calculate correlation between Count and Value of bills (Ticket Size Analysis)
    correlation_bw_count_and_value_incoming = incoming_trade.select(
        pl.corr("Count of E-Way Bills", "Value of E-Way Bills")
    ).item()

    correlation_bw_count_and_value_outgoing = outgoing_trade.select(
        pl.corr("Count of E-Way Bills", "Value of E-Way Bills")
    ).item()

    correlation_bw_count_and_value_within = within_trade.select(
        pl.corr("Count of E-Way Bills", "Value of E-Way Bills")
    ).item()

    # Obtain 12-Month Average Absolute Values
    (avg_absolute_value_12_mo_external, avg_absolute_value_12_mo_internal, avg_abso

    # Use function to calculate CAGR from above to compute it for Value and Count a
    (cagr_incoming_value, cagr_incoming_count) = find_incoming_CAGR(state_name)
    (cagr_outgoing_value, cagr_outgoing_count) = find_outgoing_CAGR(state_name)
    (cagr_within_value, cagr_within_count) = find_within_CAGR(state_name)
    (cagr_total_value, cagr_total_count) = find_cagr_total(state_name)

    # Compile and Return Results
    results = {
        f"{state_name}_incoming": incoming_trade,
        f"{state_name}_outgoing": outgoing_trade,
        f"{state_name}_within": within_trade,
        f"{state_name}_correlation_bw_incoming_outgoing_trade": correlation_bw_inco
        f"{state_name}_correlation_bw_external_trade_and_within_trade": correlation
        f"{state_name}_correlation_bw_count_and_value_incoming": correlation_bw_cou
        f"{state_name}_correlation_bw_count_and_value_outgoing": correlation_bw_cou
        f"{state_name}_correlation_bw_count_and_value_within": correlation_bw_count
        f"{state_name}_cagr_incoming_value": cagr_incoming_value,
        f"{state_name}_cagr_incoming_count": cagr_incoming_count,
        f"{state_name}_cagr_outgoing_value": cagr_outgoing_value,
        f"{state_name}_cagr_outgoing_count": cagr_outgoing_count,
        f"{state_name}_cagr_within_value": cagr_within_value,
        f"{state_name}_cagr_within_count": cagr_within_count,
```

```
        f"{state_name}_cagr_total_value": cagr_total_value,
        f"{state_name}_cagr_total_count": cagr_total_count,
        f"{state_name}_avg_absolute_value_12_mo_external": avg_absolute_value_12_mo
        f"{state_name}_avg_absolute_value_12_mo_internal": avg_absolute_value_12_mo
        f"{state_name}_avg_absolute_value_12_mo_total": avg_absolute_value_12_mo_to
    }
    return results

all_state_stats = {}

for state in indian_states:
    state_results = generate_state_e_way_bill_stats(state)
    all_state_stats[state] = state_results

# print(all_state_stats["KARNATAKA"])
```

2.4 Getting all the statistics ready for the State Level Dashboard

While the map based analysis helps compare a particular metric across states, having a state level dashboard is crucial to obtaining all the state level insights at a glance.

The dashboard is crucial for any granular analysis on a state.

In [11]:
```
"""
PREPARING DATA FOR DASHBOARD BENCHMARKING

This code block aggregates state-level metrics to calculate national-level statisti
These statistics will serve as benchmarks in the dashboard, allowing users to compa

The code repeats a standardized process for three categories of metrics:

1. Trade Flow Correlations:
   - Aggregates the correlation between 'Incoming vs Outgoing' trade to see if stat
   - Aggregates the correlation between 'External vs Internal' trade to see if dome
   - Calculates the National Average, P25, and P75 for these correlations.

2. Value vs. Count Correlations (Ticket Size Consistency):
   - Aggregates the correlation between the 'Number of Bills' and 'Total Value' for
   - A high correlation here implies consistent order sizes; low correlation implie
   - Computes national benchmarks (Avg, P25, P75) for these three categories.

3. Growth Rates (CAGR):
   - Aggregates the Compound Annual Growth Rate (CAGR) for Incoming, Outgoing, Inte
   - Computes the national benchmarks. This allows the dashboard to answer question

Key Pattern Used for Each Metric:
   - Initialize a dictionary to hold state-specific values.
   - Loop through `all_state_stats` to extract the specific metric for every state.
   - Sort the values and store them in a list.
   - Use NumPy (`np.mean`, `np.percentile`) to calculate the Mean, 25th Percentile,
   - Append these national stats back into the dictionary under keys 'avg', 'p25',
"""
```

```
"""
----------------------------------------
Section 1: Trade Flow Correlations
----------------------------------------
"""

# --- 1. Correlation: Incoming vs. Outgoing Trade ---
correlation_info_incoming_outgoing = {}

# Extract correlation metric for each state
for state in all_state_stats:
    correlation_info_incoming_outgoing[state] = all_state_stats[state][f"{state}_co

# Sort the correlations to easily inspect rank
correlation_info_incoming_outgoing_list = sorted(correlation_info_incoming_outgoing

# Create a list of values for statistical calculations
correlation_values_incoming_outgoing = []
for item in correlation_info_incoming_outgoing_list:
    correlation_values_incoming_outgoing.append(item[1])

# Calculate benchmarks
avg_correlation_values_incoming_outgoing = np.mean(correlation_values_incoming_outg
p25_correlation_values_incoming_outgoing = np.percentile(correlation_values_incomin
p75_correlation_values_incoming_outgoing = np.percentile(correlation_values_incomin

# Store benchmarks back into the dictionary
correlation_info_incoming_outgoing['avg'] = avg_correlation_values_incoming_outgoin
correlation_info_incoming_outgoing['p25'] = p25_correlation_values_incoming_outgoin
correlation_info_incoming_outgoing['p75'] = p75_correlation_values_incoming_outgoin


# --- 2. Correlation: External vs. Internal Trade ---
correlation_info_internal_external = {}

for state in all_state_stats:
    correlation_info_internal_external[state] = all_state_stats[state][f"{state}_co

correlation_info_internal_external_list = sorted(correlation_info_internal_external

correlation_values_internal_external = []
for item in correlation_info_internal_external_list:
    correlation_values_internal_external.append(item[1])

# Calculate benchmarks
avg_correlation_values_internal_external = np.mean(correlation_values_internal_exte
p25_correlation_values_internal_external = np.percentile(correlation_values_interna
p75_correlation_values_internal_external = np.percentile(correlation_values_interna

# Store benchmarks
correlation_info_internal_external['avg'] = avg_correlation_values_internal_externa
correlation_info_internal_external['p25'] = p25_correlation_values_internal_externa
correlation_info_internal_external['p75'] = p75_correlation_values_internal_externa
```

```
"""
----------------------------------------
Section 2: Ticket Size Consistency (Value vs. Count Correlation)
----------------------------------------
"""

# --- 1. Correlation: Value vs. Count (Incoming) ---
correlation_value_count_incoming = {}

for state in all_state_stats:
    correlation_value_count_incoming[state] = all_state_stats[state][f"{state}_corr

correlation_value_count_incoming_list = sorted(correlation_value_count_incoming.ite

correlation_values_value_count_incoming = []
for item in correlation_value_count_incoming_list:
    correlation_values_value_count_incoming.append(item[1])

# Calculate benchmarks
avg_correlation_values_value_count_incoming = np.mean(correlation_values_value_coun
p25_correlation_values_value_count_incoming = np.percentile(correlation_values_valu
p75_correlation_values_value_count_incoming = np.percentile(correlation_values_valu

# Store benchmarks
correlation_value_count_incoming['avg'] = avg_correlation_values_value_count_incomi
correlation_value_count_incoming['p25'] = p25_correlation_values_value_count_incomi
correlation_value_count_incoming['p75'] = p75_correlation_values_value_count_incomi


# --- 2. Correlation: Value vs. Count (Outgoing) ---
correlation_value_count_outgoing = {}

for state in all_state_stats:
    correlation_value_count_outgoing[state] = all_state_stats[state][f"{state}_corr

correlation_value_count_outgoing_list = sorted(correlation_value_count_outgoing.ite

correlation_values_value_count_outgoing = []
for item in correlation_value_count_outgoing_list:
    correlation_values_value_count_outgoing.append(item[1])

# Calculate benchmarks
avg_correlation_values_value_count_outgoing = np.mean(correlation_values_value_coun
p25_correlation_values_value_count_outgoing = np.percentile(correlation_values_valu
p75_correlation_values_value_count_outgoing = np.percentile(correlation_values_valu

# Store benchmarks
correlation_value_count_outgoing['avg'] = avg_correlation_values_value_count_outgoi
correlation_value_count_outgoing['p25'] = p25_correlation_values_value_count_outgoi
correlation_value_count_outgoing['p75'] = p75_correlation_values_value_count_outgoi


# --- 3. Correlation: Value vs. Count (Internal/Within) ---
correlation_value_count_within = {}

for state in all_state_stats:
```

```
    correlation_value_count_within[state] = all_state_stats[state][f"{state}_correl

correlation_value_count_within_list = sorted(correlation_value_count_within.items()

correlation_values_value_count_within = []
for item in correlation_value_count_within_list:
    correlation_values_value_count_within.append(item[1])

# Calculate benchmarks
avg_correlation_values_value_count_within = np.mean(correlation_values_value_count_
p25_correlation_values_value_count_within = np.percentile(correlation_values_value_
p75_correlation_values_value_count_within = np.percentile(correlation_values_value_

# Store benchmarks
correlation_value_count_within['avg'] = avg_correlation_values_value_count_within
correlation_value_count_within['p25'] = p25_correlation_values_value_count_within
correlation_value_count_within['p75'] = p75_correlation_values_value_count_within


"""
----------------------------------------
Section 3: Growth Rates (CAGR)
----------------------------------------
"""

# --- 1. CAGR: Incoming Trade Value ---
cagr_incoming_trade_value = {}

for state in all_state_stats:
    cagr_incoming_trade_value[state] = all_state_stats[state][f"{state}_cagr_incomi

cagr_incoming_trade_value_list = sorted(cagr_incoming_trade_value.items(), key=lamb

cagr_values_incoming_trade_value = []
for item in cagr_incoming_trade_value_list:
    cagr_values_incoming_trade_value.append(item[1])

# Calculate benchmarks
avg_cagr_incoming_trade_value = np.mean(cagr_values_incoming_trade_value)
p25_cagr_incoming_trade_value = np.percentile(cagr_values_incoming_trade_value, 25)
p75_cagr_incoming_trade_value = np.percentile(cagr_values_incoming_trade_value, 75)

# Store benchmarks
cagr_incoming_trade_value['avg'] = avg_cagr_incoming_trade_value
cagr_incoming_trade_value['p25'] = p25_cagr_incoming_trade_value
cagr_incoming_trade_value['p75'] = p75_cagr_incoming_trade_value


# --- 2. CAGR: Outgoing Trade Value ---
cagr_outgoing_trade_value = {}

for state in all_state_stats:
    cagr_outgoing_trade_value[state] = all_state_stats[state][f"{state}_cagr_outgoi

cagr_outgoing_trade_value_list = sorted(cagr_outgoing_trade_value.items(), key=lamb
```

```
cagr_values_outgoing_trade_value = []
for item in cagr_outgoing_trade_value_list:
    cagr_values_outgoing_trade_value.append(item[1])

# Calculate benchmarks
avg_cagr_outgoing_trade_value = np.mean(cagr_values_outgoing_trade_value)
p25_cagr_outgoing_trade_value = np.percentile(cagr_values_outgoing_trade_value, 25)
p75_cagr_outgoing_trade_value = np.percentile(cagr_values_outgoing_trade_value, 75)

# Store benchmarks
cagr_outgoing_trade_value['avg'] = avg_cagr_outgoing_trade_value
cagr_outgoing_trade_value['p25'] = p25_cagr_outgoing_trade_value
cagr_outgoing_trade_value['p75'] = p75_cagr_outgoing_trade_value


# --- 3. CAGR: Internal Trade Value ---
cagr_internal_trade_value = {}

for state in all_state_stats:
    cagr_internal_trade_value[state] = all_state_stats[state][f"{state}_cagr_within

cagr_internal_trade_value_list = sorted(cagr_internal_trade_value.items(), key=lamb

cagr_values_internal_trade_value = []
for item in cagr_internal_trade_value_list:
    cagr_values_internal_trade_value.append(item[1])

# Calculate benchmarks
avg_cagr_internal_trade_value = np.mean(cagr_values_internal_trade_value)
p25_cagr_internal_trade_value = np.percentile(cagr_values_internal_trade_value, 25)
p75_cagr_internal_trade_value = np.percentile(cagr_values_internal_trade_value, 75)

# Store benchmarks
cagr_internal_trade_value['avg'] = avg_cagr_internal_trade_value
cagr_internal_trade_value['p25'] = p25_cagr_internal_trade_value
cagr_internal_trade_value['p75'] = p75_cagr_internal_trade_value


# --- 4. CAGR: Total Trade Value ---
cagr_total_trade_value = {}

for state in all_state_stats:
    cagr_total_trade_value[state] = all_state_stats[state][f"{state}_cagr_total_val

cagr_total_trade_value_list = sorted(cagr_total_trade_value.items(), key=lambda x:

cagr_values_total_trade_value = []
for item in cagr_total_trade_value_list:
    cagr_values_total_trade_value.append(item[1])

# Calculate benchmarks
avg_cagr_total_trade_value = np.mean(cagr_values_total_trade_value)
p25_cagr_total_trade_value = np.percentile(cagr_values_total_trade_value, 25)
p75_cagr_total_trade_value = np.percentile(cagr_values_total_trade_value, 75)

# Store benchmarks
```

```
cagr_total_trade_value['avg'] = avg_cagr_total_trade_value
cagr_total_trade_value['p25'] = p25_cagr_total_trade_value
cagr_total_trade_value['p75'] = p75_cagr_total_trade_value
```

3. Generating country level plots now to look at the E-Way Bill Statistics

This section will finally incorporate all the data that we have gathered in the steps above and visualize it

3.1 Fetching the Geodata and storing it for reusability

In [12]:
```
url = "https://cdn.jsdelivr.net/gh/udit-001/india-maps-data@8d907bc/geojson/india.g
print("Downloading GeoJSON...")
response = requests.get(url)
gdf = gpd.read_file(io.BytesIO(response.content))
gdf.head(2)
```

```
Downloading GeoJSON...
```

Out[12]:

	id	district	dt_code	st_nm	st_code	year	geometry
0	None	Aizawl	261	Mizoram	15	2011_c	POLYGON ((93.04466 23.41052, 92.9468 23.51363,...
1	None	Champhai	262	Mizoram	15	2011_c	MULTIPOLYGON (((93.04619 23.66623, 93.04466 23...

Converting the Geometry column to a string using wkt. This will allow us to use the dataframe in Polars.

In [13]:
```
gdf_pandas = pd.DataFrame(gdf)
gdf_pandas['geometry'] = gdf_pandas['geometry'].apply(lambda x: x.wkt)
gdf_pandas.head(2)
gdf_pl = pl.from_pandas(gdf_pandas)
```

3.2 Mapping the state names in the geopandas dataset with the state names from the E-Way Bills dataset

In [14]:
```
state_mapping = {
    "ANDAMAN AND NICOBAR": "Andaman and Nicobar Islands",
    "DADRA AND NAGAR HAVELI": None,
    "DAMAN AND DIU": None,
    "DELHI": "Delhi",
    "Other Territory": None,
    "ANDHRA PRADESH": "Andhra Pradesh",
    "ARUNACHAL PRADESH": "Arunachal Pradesh",
    "ASSAM": "Assam",
    "BIHAR": "Bihar",
    "CHANDIGARH": "Chandigarh",
```

```
    "CHHATTISGARH": "Chhattisgarh",
    "GOA": "Goa",
    "GUJARAT": "Gujarat",
    "HARYANA": "Haryana",
    "HIMACHAL PRADESH": "Himachal Pradesh",
    "JAMMU AND KASHMIR": "Jammu and Kashmir",
    "JHARKHAND": "Jharkhand",
    "KARNATAKA": "Karnataka",
    "KERALA": "Kerala",
    "LADAKH": "Ladakh",
    "LAKSHADWEEP": "Lakshadweep",
    "MADHYA PRADESH": "Madhya Pradesh",
    "MAHARASHTRA": "Maharashtra",
    "MANIPUR": "Manipur",
    "MEGHALAYA": "Meghalaya",
    "MIZORAM": "Mizoram",
    "NAGALAND": "Nagaland",
    "ODISHA": "Odisha",
    "PUDUCHERRY": "Puducherry",
    "PUNJAB": "Punjab",
    "RAJASTHAN": "Rajasthan",
    "SIKKIM": "Sikkim",
    "TAMIL NADU": "Tamil Nadu",
    "TELANGANA": "Telangana",
    "TRIPURA": "Tripura",
    "UTTAR PRADESH": "Uttar Pradesh",
    "UTTARAKHAND": "Uttarakhand",
    "WEST BENGAL": "West Bengal",
    "avg": None,
    "p25": None,
    "p75": None
}
```


3.3 Preparing different polars dataframes for geospatial plotting

List of Dataframes will include:

1. Value of E-Way Bills (External) - This will be visualize the interconnectedness of states in terms of merchandise trade

2. Value of E-Way Bills (Internal) - This will be visualize the internal market development of states

3. Value of E-Way Bills (Total) - This will be visualize the total merchandise trade of states

4. CAGR of Value of E-Way Bills (Total) - Where has the merchandise trade most grown (in percentage terms)

5. CAGR of Value of E-Way Bills (Internal) - Where has the internal trade most grown (in percentage terms)

`convert_to_dataframe` Helper Function

This utility function is designed to transform a Python dictionary into a Polars DataFrame for easier plotting and analysis.

1. **Input:** A dictionary where keys represent `state` names and values represent a specific metric (e.g., CAGR, Correlation, etc.).
2. **Process:** It iterates through the dictionary items, separating keys and values into two distinct lists stored within a new dictionary structure.
3. **Output:** Returns a 2-column Polars DataFrame (`state`, `value`) ready for use with visualization libraries.

In [15]:
```
def convert_to_dataframe(dict_of_state_and_value):
    converted_dict = {"state": [], "value": []}
    for item in dict_of_state_and_value.items():
        converted_dict["state"].append(item[0])
        converted_dict["value"].append(item[1])
    return pl.DataFrame(converted_dict)
```

Using the Helper function above and generating the Dataframes for Plotting

The loop below calls upon the state level statistics generated above and stored in a dictionary called all_state_stats

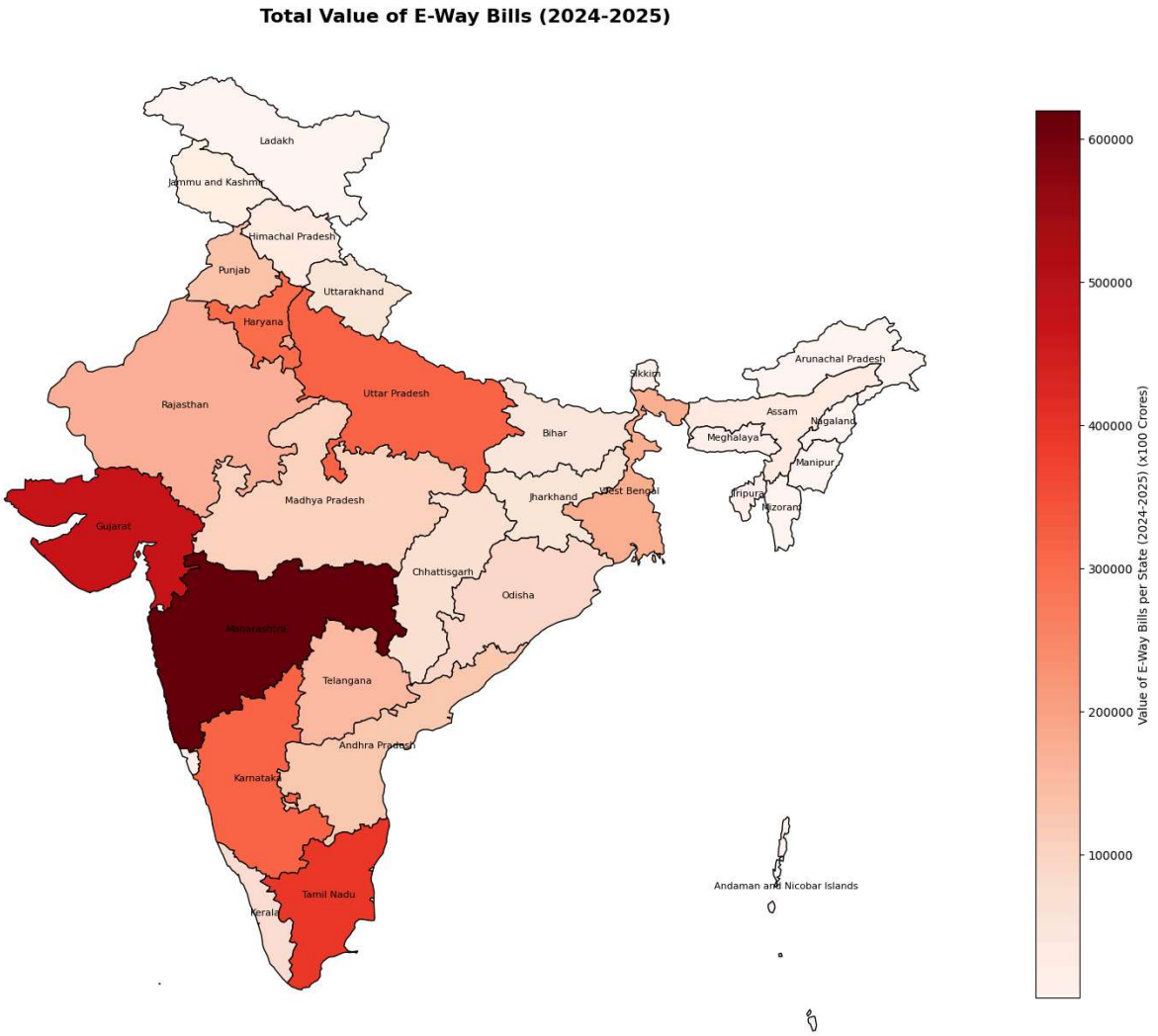
In [16]:
```
e_way_bill_value_external = {}
e_way_bill_value_internal = {}
e_way_bill_value_total = {}

for state in all_state_stats:
    e_way_bill_value_external[state] = all_state_stats[state][f"{state}_avg_absolut
    e_way_bill_value_internal[state] = all_state_stats[state][f"{state}_avg_absolut
    e_way_bill_value_total[state] = all_state_stats[state][f"{state}_avg_absolute_v


e_way_bill_value_external_df = convert_to_dataframe(e_way_bill_value_external)
e_way_bill_value_internal_df = convert_to_dataframe(e_way_bill_value_internal)
e_way_bill_value_total_df = convert_to_dataframe(e_way_bill_value_total)
cagr_total_trade_value_df = convert_to_dataframe(cagr_total_trade_value)
cagr_internal_trade_value_df = convert_to_dataframe(cagr_internal_trade_value)
```

3.3 Map 1: E-Way Bill Absolute Values (2024-2025, Total Merchandise Trade)

In [17]:
```
"""
This code block performs the final steps to create a choropleth map of India showin

1. Data Standardization: It replaces state names in the value dataframe to match th
2. Data Merging: It performs an inner join between the trade value data and the geo
3. GeoDataFrame Creation: It converts the Polars DataFrame to Pandas, reconstructs
4. Plotting: It renders the map using Matplotlib, color-coding states by their trad
```

```
"""

# Using the state mapping dictionary from above to match the names of the states an
e_way_bill_value_total_df = e_way_bill_value_total_df.with_columns(
    st_nm_new = pl.col("state").replace(state_mapping) # Create a new column with s
).select(
    pl.col("st_nm_new"),
    pl.col("value") # Keep only the new name and the value column
)

# Merge the trade value data with the geographic data (gdf_pl)
merged_pl = e_way_bill_value_total_df.join(
    gdf_pl,
    left_on="st_nm_new",
    right_on="st_nm",
    how="inner" # Inner join ensures we only keep states present in both datasets
).select(
    pl.col("st_nm_new").alias("st_nm"), # Rename for consistency
    pl.col("value"),
    pl.col("geometry") # Include the geometry column for mapping
)

# Convert Polars DataFrame to Pandas (Required for GeoPandas)
merged_pandas = merged_pl.to_pandas()

# Reconstruct Geometry from WKT (Well-Known Text) string format
merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)

# Create the GeoDataFrame
merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

# Dissolve boundaries to ensure one geometry per state (removes internal district l
merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

# Initialize the plot figure
fig, ax = plt.subplots(1, 1, figsize=(15, 15))

# Plot the choropleth map
merged_gdf.plot(column='value',
                ax=ax,
                legend=True, # Add a colorbar legend
                legend_kwds={
                    'label': "Value of E-Way Bills per State (2024-2025) (x100 Cror
                    'orientation': "vertical",
                    'shrink': 0.7 # Shrink the legend size slightly
                },
                cmap='Reds', # Use a red color scale
                edgecolor='black'
)

# Add Labels to the map
for idx, row in merged_gdf.iterrows():
    if row.geometry.area > 0.5: # Only label larger states to avoid clutter
        plt.annotate(text=row['st_nm'],
                     xy=row.geometry.centroid.coords[0], # Place label at the geomet
                     horizontalalignment='center',
```

```
                    fontsize=8,
                    color='black'
                    )

# Final formatting and display
ax.set_title('Total Value of E-Way Bills (2024-2025)', fontsize=16, fontweight='bol
ax.set_axis_off() # Remove X and Y axes (lat/long markings)
plt.tight_layout()
plt.show()
```

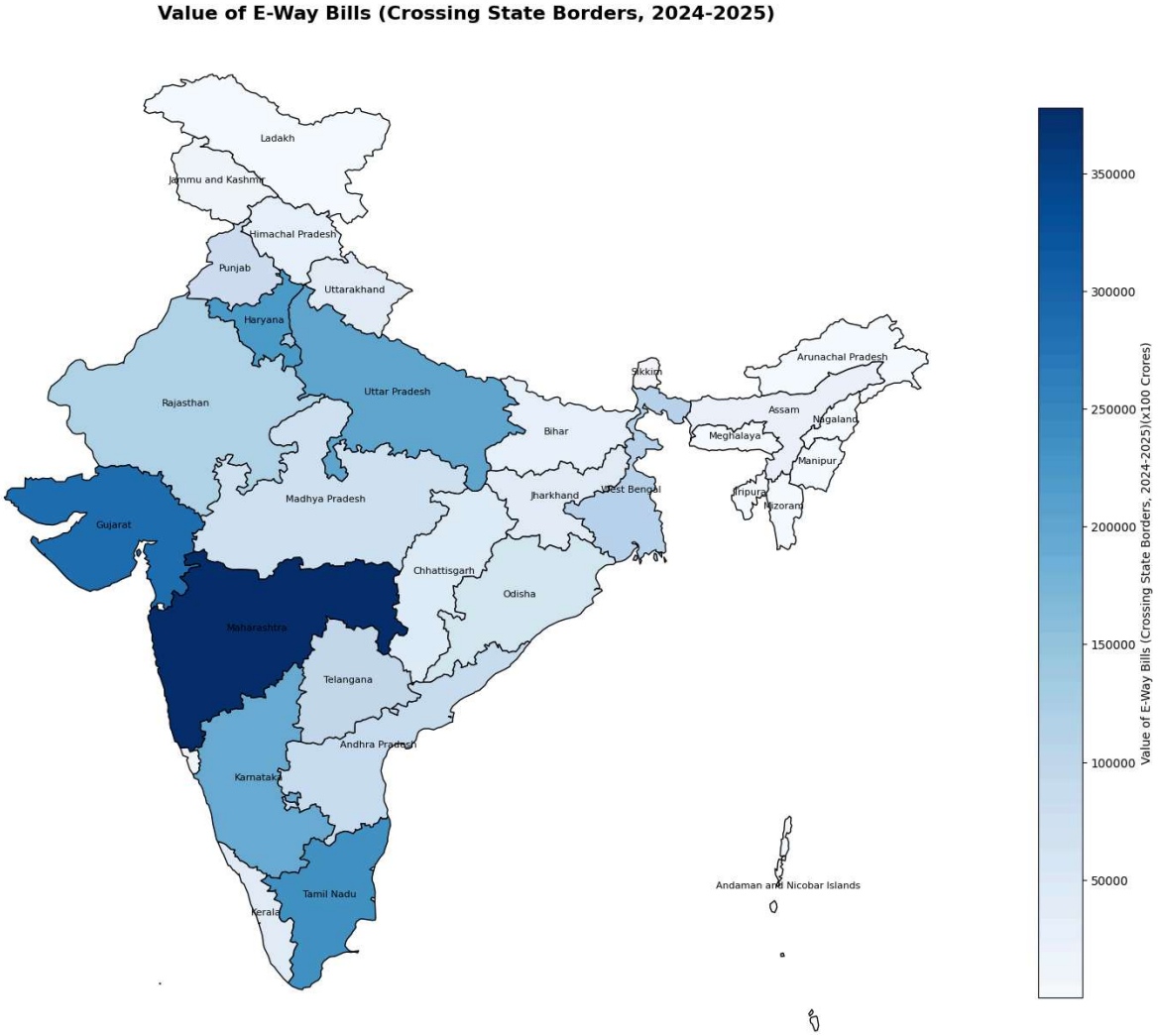


3.4 Map 2: Mapping the E-Way Bill Values crossing state boundaries (2024-2025)

In [18]:
```
"""
This code block creates a choropleth map specifically for External Trade (Inter-Sta

1. Data Standardization: Replaces state names to match the geo-data format.
2. Data Merging: Joins the trade value data with the geographic boundaries.
3. GeoDataFrame Creation: Converts the data to GeoPandas format and reconstructs ge
4. Plotting: Renders the map using a 'Blues' colormap to distinguish it from the To
"""
```

```
# Apply state name mapping to ensure consistency with the GeoJSON file
e_way_bill_value_external_df = e_way_bill_value_external_df.with_columns(
    st_nm_new = pl.col("state").replace(state_mapping)
).select(
    pl.col("st_nm_new"),
    pl.col("value")
)

# Merge the external trade data with the geographic data
merged_pl = e_way_bill_value_external_df.join(
    gdf_pl,
    left_on="st_nm_new",
    right_on="st_nm",
    how="inner"
).select(
    pl.col("st_nm_new").alias("st_nm"),
    pl.col("value"),
    pl.col("geometry")
)

# Convert to Pandas for GeoPandas compatibility
merged_pandas = merged_pl.to_pandas()

# Reconstruct geometry from WKT format
merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)

# Create the GeoDataFrame
merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

# Dissolve boundaries to handle any state fragmentation
merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

# Initialize the figure
fig, ax = plt.subplots(1, 1, figsize=(15, 15))

# Plot the map
merged_gdf.plot(column='value',
                ax=ax,
                legend=True,
                legend_kwds={
                    'label': "Value of E-Way Bills (Crossing State Borders, 2024-20
                    'orientation': "vertical",
                    'shrink': 0.7
                },
                cmap='Blues', # Using Blue scale for External/Inter-state trade
                edgecolor='black'
)

# Add Labels to larger states
for idx, row in merged_gdf.iterrows():
    if row.geometry.area > 0.5:
        plt.annotate(text=row['st_nm'],
                     xy=row.geometry.centroid.coords[0],
                     horizontalalignment='center',
                     fontsize=8,
                     color='black'
```

```
                    )

# Final display settings
ax.set_title('Value of E-Way Bills (Crossing State Borders, 2024-2025)', fontsize=1
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

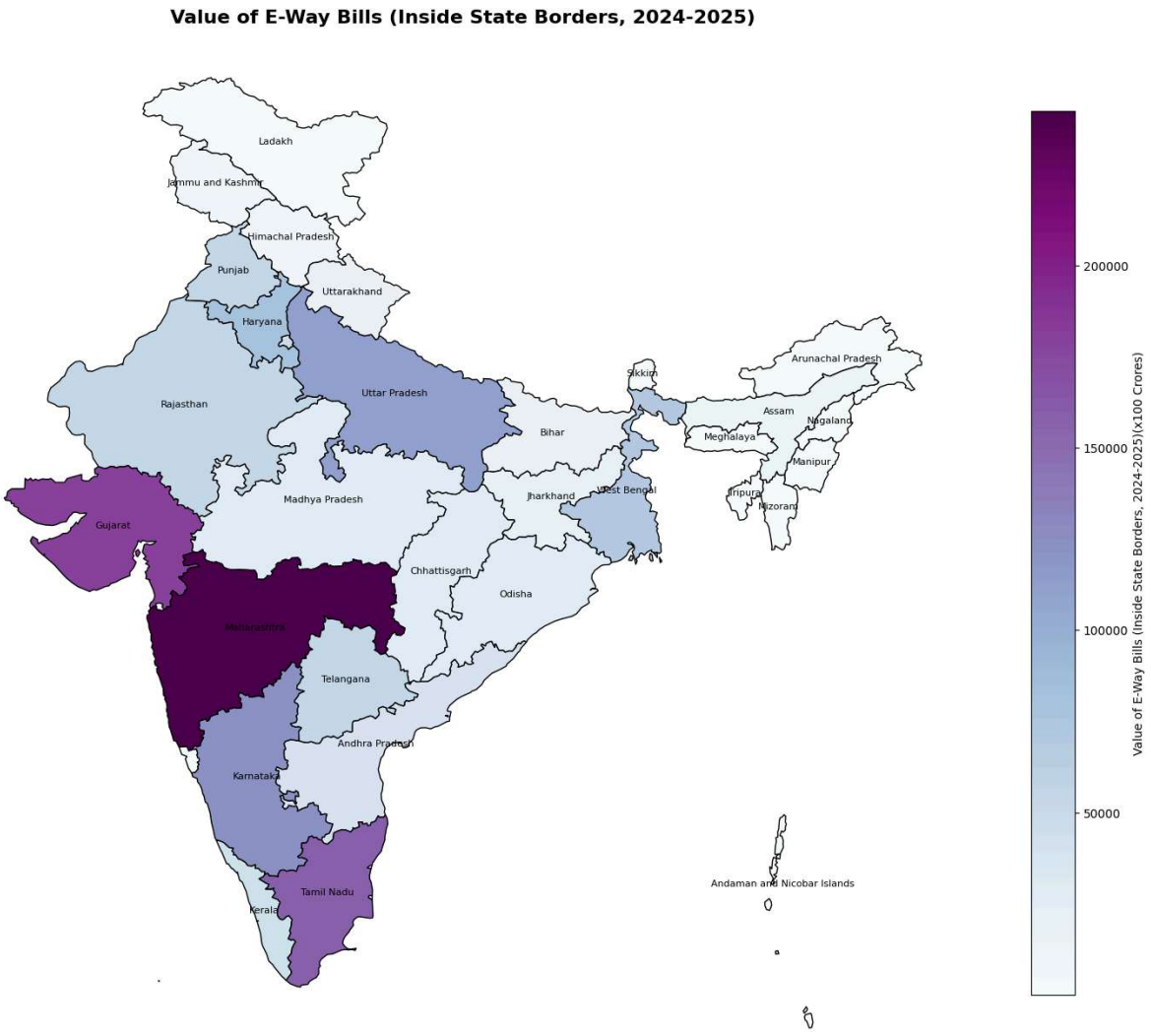


3.5 Map 3: Mapping the Merchandise Trade occuring Inside State Boundaries (2024-2025)

In [19]:
```
"""
This code block generates a choropleth map specifically for Internal Trade (Within-

1. Data Standardization: Maps state names in the internal trade dataset to the stan
2. Data Merging: Joins the internal trade values with state geometry.
3. GeoDataFrame Creation: converts the data into a GeoDataFrame and reconstructs th
4. Plotting: Renders the map using a 'BuPu' (Blue-Purple) colormap to distinctively
"""

# Apply state name mapping to match the standardized names in the GeoJSON
e_way_bill_value_internal_df = e_way_bill_value_internal_df.with_columns(
```

```
    st_nm_new = pl.col("state").replace(state_mapping)
).select(
    pl.col("st_nm_new"),
    pl.col("value")
)

# Merge the internal trade value data with the geographic boundaries
merged_pl = e_way_bill_value_internal_df.join(
    gdf_pl,
    left_on="st_nm_new",
    right_on="st_nm",
    how="inner"
).select(
    pl.col("st_nm_new").alias("st_nm"),
    pl.col("value"),
    pl.col("geometry")
)

# Convert Polars DataFrame to Pandas for GeoPandas compatibility
merged_pandas = merged_pl.to_pandas()

# Reconstruct geometry objects from WKT strings
merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)

# Create the GeoDataFrame
merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

# Dissolve boundaries to unify state polygons (removes internal divisions)
merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

# Initialize the figure
fig, ax = plt.subplots(1, 1, figsize=(15, 15))

# Plot the choropleth map
merged_gdf.plot(column='value',
                ax=ax,
                legend=True,
                legend_kwds={
                    'label': "Value of E-Way Bills (Inside State Borders, 2024-2025
                    'orientation': "vertical",
                    'shrink': 0.7
                },
                cmap='BuPu', # Using Blue-Purple scale for Internal trade
                edgecolor='black'
)

# Add Labels to the map for larger states
for idx, row in merged_gdf.iterrows():
    if row.geometry.area > 0.5:
        plt.annotate(text=row['st_nm'],
                     xy=row.geometry.centroid.coords[0],
                     horizontalalignment='center',
                     fontsize=8,
                     color='black'
                     )
```

```
# Final plot configuration
ax.set_title('Value of E-Way Bills (Inside State Borders, 2024-2025)', fontsize=16,
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

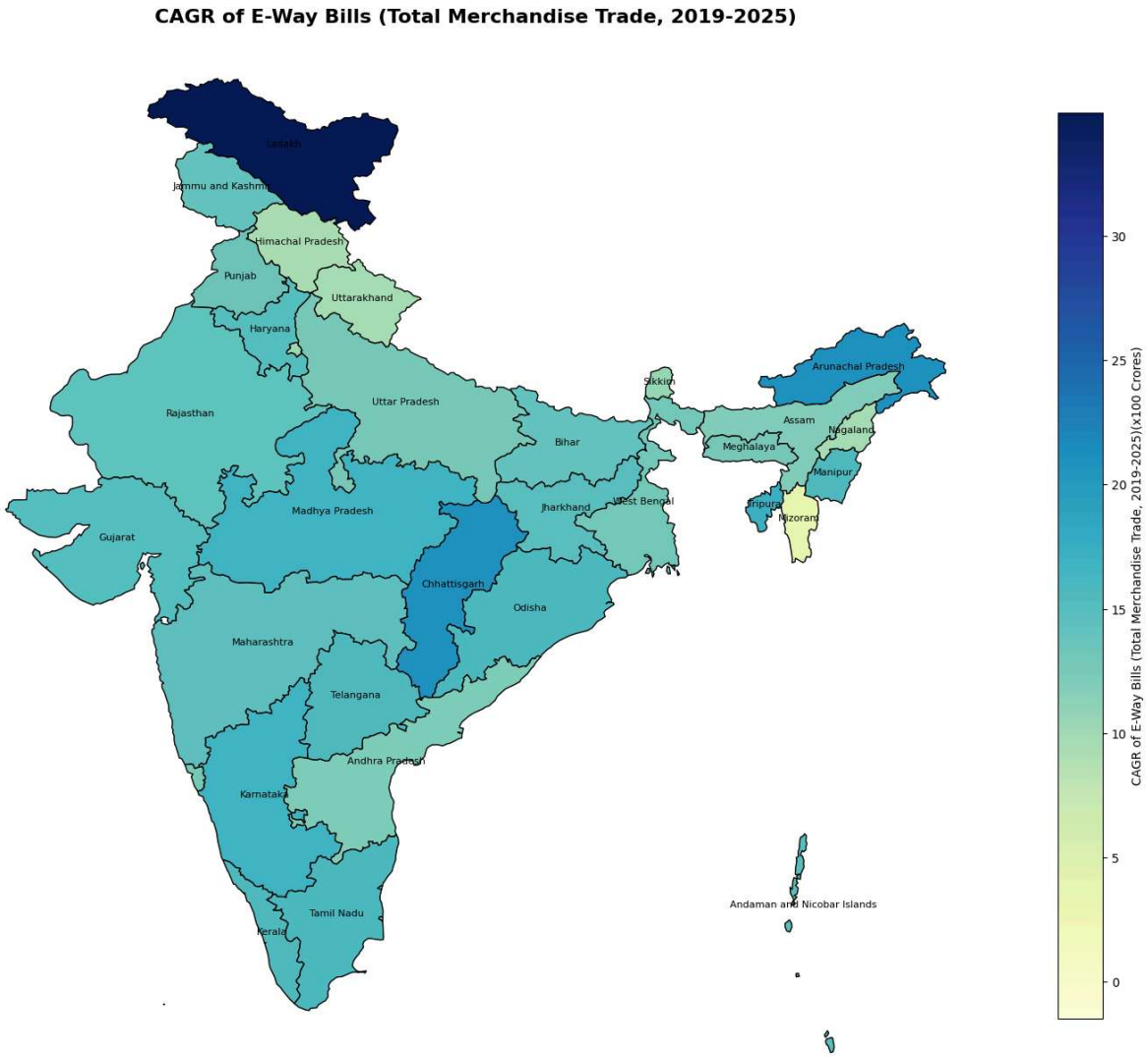


3.6 Map 4: Visualizing the CAGR of overall trade (2019-2025)

In [20]:
```
"""
This code block generates a choropleth map to visualize the Compound Annual Growth

1. Data Standardization: Maps state names in the CAGR dataframe to the standardized
2. Data Merging: Joins the calculated CAGR values with state geometry.
3. GeoDataFrame Creation: Converts the data into a GeoDataFrame and reconstructs th
4. Plotting: Renders the map using a 'YlGnBu' (Yellow-Green-Blue) colormap to highl
"""

# Apply state name mapping to match the standardized names in the GeoJSON
cagr_total_trade_value_df = cagr_total_trade_value_df.with_columns(
    st_nm_new = pl.col("state").replace(state_mapping)
).select(
    pl.col("st_nm_new"),
```

```
    pl.col("value")
)

# Merge the CAGR value data with the geographic boundaries
merged_pl = cagr_total_trade_value_df.join(
    gdf_pl,
    left_on="st_nm_new",
    right_on="st_nm",
    how="inner"
).select(
    pl.col("st_nm_new").alias("st_nm"),
    pl.col("value"),
    pl.col("geometry")
)

# Convert Polars DataFrame to Pandas for GeoPandas compatibility
merged_pandas = merged_pl.to_pandas()

# Reconstruct geometry objects from WKT strings
merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)

# Create the GeoDataFrame
merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

# Dissolve boundaries to unify state polygons
merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

# Initialize the figure
fig, ax = plt.subplots(1, 1, figsize=(15, 15))

# Plot the choropleth map
merged_gdf.plot(column='value',
                ax=ax,
                legend=True,
                legend_kwds={
                    'label': "CAGR of E-Way Bills (Total Merchandise Trade, 2019-20
                    'orientation': "vertical",
                    'shrink': 0.7
                },
                cmap='YlGnBu', # Using Yellow-Green-Blue scale for Growth Rates
                edgecolor='black'
)

# Add Labels to the map for larger states
for idx, row in merged_gdf.iterrows():
    if row.geometry.area > 0.5:
        plt.annotate(text=row['st_nm'],
                     xy=row.geometry.centroid.coords[0],
                     horizontalalignment='center',
                     fontsize=8,
                     color='black'
                     )

# Final plot configuration
ax.set_title('CAGR of E-Way Bills (Total Merchandise Trade, 2019-2025)', fontsize=1
ax.set_axis_off()
```

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

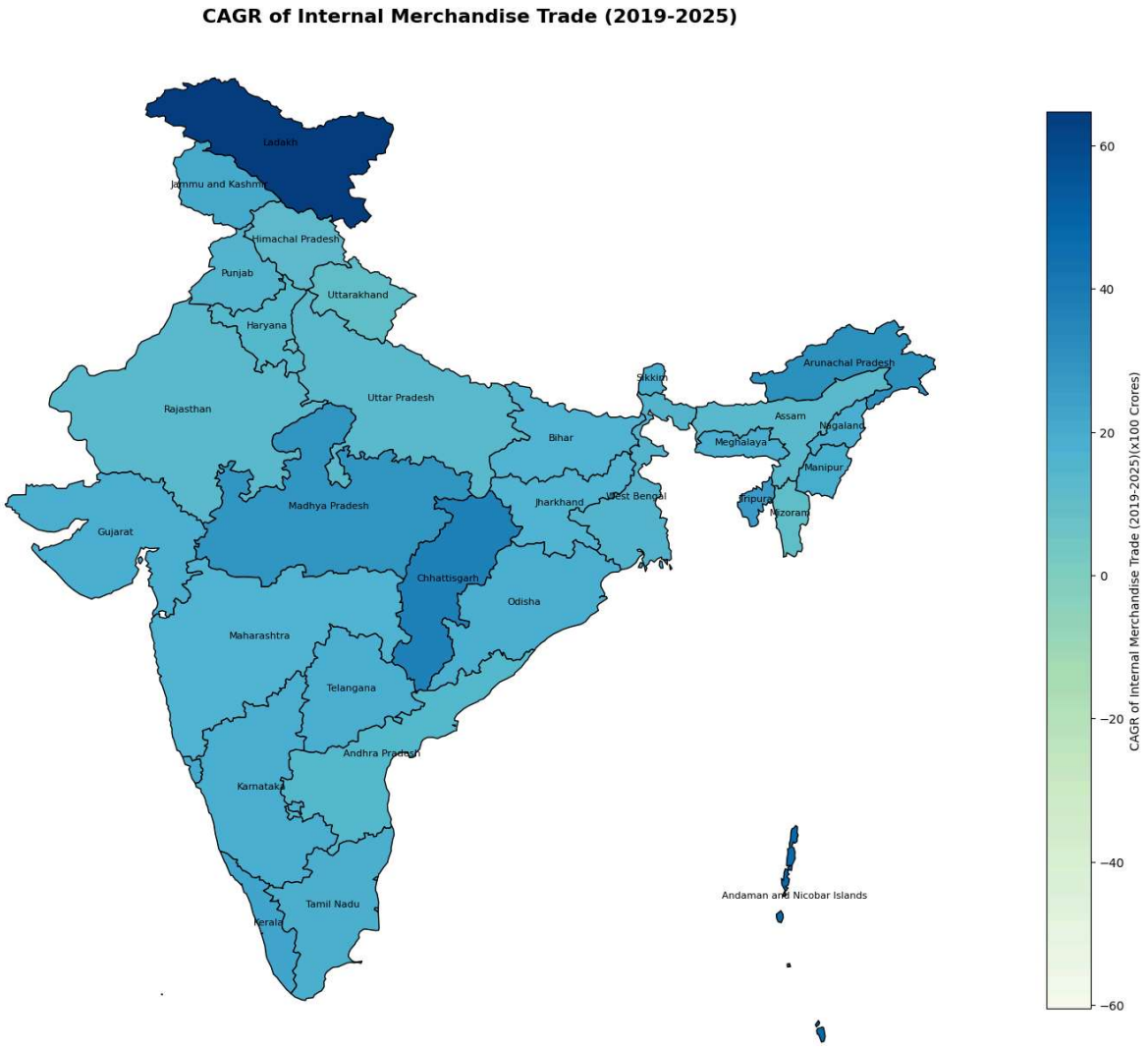


3.7 Map 5: Visualizing the CAGR of Internal Merchandise Trade (2019-2025)

In [21]:
```
"""
This code block generates a choropleth map to visualize the Compound Annual Growth
This highlights which states are seeing the fastest acceleration in their domestic

1. Data Standardization: standardized state names are applied to ensure they match
2. Data Merging: Joins the internal trade growth rates with the state boundaries.
3. GeoDataFrame Creation: Converts the combined data into a geospatial format suita
4. Plotting: Renders the map using a 'GnBu' (Green-Blue) colormap to indicate growt
"""

# Apply state name mapping to match the standardized names in the GeoJSON
cagr_internal_trade_value_df = cagr_internal_trade_value_df.with_columns(
    st_nm_new = pl.col("state").replace(state_mapping)
).select(
    pl.col("st_nm_new"),
```

```
    pl.col("value")
)

# Merge the Internal CAGR data with the geographic boundaries
merged_pl = cagr_internal_trade_value_df.join(
    gdf_pl,
    left_on="st_nm_new",
    right_on="st_nm",
    how="inner"
).select(
    pl.col("st_nm_new").alias("st_nm"),
    pl.col("value"),
    pl.col("geometry")
)

# Convert Polars DataFrame to Pandas for GeoPandas compatibility
merged_pandas = merged_pl.to_pandas()

# Reconstruct geometry objects from WKT strings
merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)

# Create the GeoDataFrame
merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

# Dissolve boundaries to ensure clean state polygons
merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

# Initialize the figure
fig, ax = plt.subplots(1, 1, figsize=(15, 15))

# Plot the choropleth map
merged_gdf.plot(column='value',
                ax=ax,
                legend=True,
                legend_kwds={
                    'label': "CAGR of Internal Merchandise Trade (2019-2025)(x100 C
                    'orientation': "vertical",
                    'shrink': 0.7
                },
                cmap='GnBu', # Using Green-Blue scale for Internal Growth
                edgecolor='black'
)

# Add Labels to the map for larger states
for idx, row in merged_gdf.iterrows():
    if row.geometry.area > 0.5:
        plt.annotate(text=row['st_nm'],
                     xy=row.geometry.centroid.coords[0],
                     horizontalalignment='center',
                     fontsize=8,
                     color='black'
                     )

# Final plot configuration
ax.set_title('CAGR of Internal Merchandise Trade (2019-2025)', fontsize=16, fontwei
ax.set_axis_off()
```

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



4. Preparing the State Level Dashboard

The section below will help to see the activity within a state at a glance and also comparisons with other states in terms of where it stands on a particular metric

The dashboard generated below is interactable and the state to be explored can be chosen from a dropdown menu.

The interactive dashboard will also contain some higher level analysis that can be applied to each state

I will start by first getting the correct state names from the E-way bill dataset once again

In [22]:
```
"""
The raw data has UPPERCASE names (e.g., "KARNATAKA"), but the
```

```
functions expect Title Case (e.g., "Karnataka").
This applies the mapping globally to the dataframe.
"""

state_name_mapping = {
    # Mapped to match Employment Data
    "ANDAMAN AND NICOBAR": "Andaman and Nicobar Islands",
    "ANDHRA PRADESH": "Andhra Pradesh",
    "ARUNACHAL PRADESH": "Arunachal Pradesh",
    "ASSAM": "Assam",
    "BIHAR": "Bihar",
    "CHANDIGARH": "Chandigarh",
    "CHHATTISGARH": "Chhattisgarh",
    "DELHI": "Delhi",
    "GOA": "Goa",
    "GUJARAT": "Gujarat",
    "HARYANA": "Haryana",
    "HIMACHAL PRADESH": "Himachal Pradesh",
    "JAMMU AND KASHMIR": "Jammu and Kashmir",
    "JHARKHAND": "Jharkhand",
    "KARNATAKA": "Karnataka",
    "KERALA": "Kerala",
    "LADAKH": "Ladakh",
    "LAKSHADWEEP": "Lakshadweep",
    "MADHYA PRADESH": "Madhya Pradesh",
    "MAHARASHTRA": "Maharashtra",
    "MANIPUR": "Manipur",
    "MEGHALAYA": "Meghalaya",
    "MIZORAM": "Mizoram",
    "NAGALAND": "Nagaland",
    "ODISHA": "Odisha",
    "PUNJAB": "Punjab",
    "RAJASTHAN": "Rajasthan",
    "SIKKIM": "Sikkim",
    "TAMIL NADU": "Tamilnadu",  # Note: Target DF uses concatenated spelling
    "TELANGANA": "Telangana",
    "TRIPURA": "Tripura",
    "UTTAR PRADESH": "Uttar Pradesh",
    "UTTARAKHAND": "Uttarakhand",
    "WEST BENGAL": "West Bengal",

    # Explicit Removals (Mapped to None)
    "DADRA AND NAGAR HAVELI": None,
    "DAMAN AND DIU": None,
    "PUDUCHERRY": None,
    "Other Territory": None
}

e_way_bill_data = e_way_bill_data.with_columns(
    pl.col("State/Ut Name").str.strip_chars().replace(state_name_mapping)
).filter(
    pl.col("State/Ut Name").is_not_null() # Remove states mapped to None
)
```

4.1 `build_state_summary_text` Helper Function

This function acts as the **formatting engine** for the textual component of the dashboard. Its primary purpose is to solve layout issues where detailed insights were previously overflowing the plot boundaries.

Key Functionalities:

1. **Text Wrapping:** Utilizes the `textwrap` library to strictly enforce a character limit (`width=92`) per line. This ensures that long, qualitative comments (e.g., explanations of "Structurally Decoupled" trade) wrap neatly within the dashboard's text panel without getting cut off.
2. **Safe Formatting:** Includes internal helper functions (`fmt_pct`, `fmt_corr`, `is_nan`) to handle missing data (`NaN` or `None`) gracefully, preventing the dashboard from crashing if a specific metric is unavailable for a state.
3. **Report Structuring:** Systematically assembles the report into four readable sections:
   - **Scale:** Absolute monthly trade values.
   - **Growth:** CAGR percentages with national context.
   - **Structure:** Correlation coefficients indicating trade alignment.
   - **Basket Stability:** Analysis of value-to-volume consistency.

In [23]:
```
def build_state_summary_text(
    State,
    abs_tot, abs_ext, abs_int,
    cagr_tot_val, cagr_int_val,
    corr_incoming_outgoing, corr_internal_external,
    corr_vc_in, corr_vc_out, corr_vc_int,
    bench_cagr_tot=None, bench_cagr_int=None,
    bench_corr_io=None, bench_corr_ie=None,
    bench_vc_in=None, bench_vc_out=None, bench_vc_int=None,
    comment_cagr_total="", comment_cagr_internal="",
    comment_io="", comment_ie="",
    comment_vc_in="", comment_vc_out="", comment_vc_int="",
    width=92
):
    def is_nan(x):
        return isinstance(x, float) and x != x

    def fmt_corr(val):
        if val is None or is_nan(val):
            return "N/A"
        return f"{val:.2f}"

    def fmt_pct(val):
        if val is None or is_nan(val):
            return "N/A"
        return f"{val:.2f}%"

    def bench_triplet_str(bench, kind="corr"):
        if not bench:
            return "Nat p25/avg/p75: N/A"
        p25 = bench.get("p25")
```

```
        avg = bench.get("avg")
        p75 = bench.get("p75")
        if p25 is None or avg is None or p75 is None:
            return "Nat p25/avg/p75: N/A"
        if kind == "pct":
            return f"Nat p25/avg/p75: {p25:.2f}% / {avg:.2f}% / {p75:.2f}%"
        return f"Nat p25/avg/p75: {p25:.2f} / {avg:.2f} / {p75:.2f}"

    def wrap_line(label, value_str, bench_str, comment):
        # One compact line + a wrapped insight line
        line1 = f"- {label}: {value_str}  ({bench_str})"
        line2 = textwrap.fill(
            comment.strip(),
            width=width,
            initial_indent="  ",
            subsequent_indent="  "
        ) if comment else ""
        return [line1] + ([line2] if line2 else [])

    lines = []
    lines.append(f"{State.upper()} — STATE TRADE SNAPSHOT")
    lines.append("=" * min(len(lines[-1]), 60))
    lines.append("")

    lines.append("Scale (12-month average)")
    lines.append("-" * 26)
    lines.append(f"- Total trade:      ₹ {abs_tot:,.0f} Cr / month")
    lines.append(f"- External trade:   ₹ {abs_ext:,.0f} Cr / month")
    lines.append(f"- Internal trade:   ₹ {abs_int:,.0f} Cr / month")
    lines.append("")

    lines.append("Growth (CAGR 2019–2025)")
    lines.append("-" * 23)
    lines += wrap_line(
        "Total trade growth",
        fmt_pct(cagr_tot_val),
        bench_triplet_str(bench_cagr_tot, "pct"),
        comment_cagr_total
    )
    lines += wrap_line(
        "Internal logistics growth",
        fmt_pct(cagr_int_val),
        bench_triplet_str(bench_cagr_int, "pct"),
        comment_cagr_internal
    )
    lines.append("")

    lines.append("Structure (correlations)")
    lines.append("-" * 25)
    lines += wrap_line(
        "Incoming vs outgoing alignment",
        fmt_corr(corr_incoming_outgoing),
        bench_triplet_str(bench_corr_io, "corr"),
        comment_io
    )
    lines += wrap_line(
```

```
        "Internal vs external coupling",
        fmt_corr(corr_internal_external),
        bench_triplet_str(bench_corr_ie, "corr"),
        comment_ie
    )
    lines.append("")

    lines.append("Basket stability (value vs count)")
    lines.append("-" * 34)
    lines += wrap_line(
        "Incoming basket stability",
        fmt_corr(corr_vc_in),
        bench_triplet_str(bench_vc_in, "corr"),
        comment_vc_in
    )
    lines += wrap_line(
        "Outgoing basket stability",
        fmt_corr(corr_vc_out),
        bench_triplet_str(bench_vc_out, "corr"),
        comment_vc_out
    )
    lines += wrap_line(
        "Internal basket stability",
        fmt_corr(corr_vc_int),
        bench_triplet_str(bench_vc_int, "corr"),
        comment_vc_int
    )

    return "\n".join(lines)
```

4.2 `plot_state_analysis` Visualization Function

This function handles the generation of the full **State-Level Dashboard**, combining visual time-series data with the textual analysis generated above.

1. **Layout Management:** Uses `gridspec_kw` to define a custom layout where the bottom subplot (Text Panel) is allocated significantly more vertical space (Ratio 1.6) than the charts. This specific adjustment is critical for accommodating the detailed text report.
2. **Time-Series Visualization:** Iterates through **Incoming**, **Outgoing**, and **Within-State** trade to generate three dual-axis charts:
   - **Left Axis (Red):** Transaction Volume (Count of E-Way Bills).
   - **Right Axis (Blue):** Monetary Value (In Crores).
3. **Automated Insight Generation:**
   - Defines logic (`analyze_cagr`, `analyze_corr`) to compare the specific state's metrics against the **National Benchmarks** (25th/75th percentiles) calculated earlier.
   - Automatically generates qualitative descriptors (e.g., "Hyper-Growth Phase", "Mature Domestic Distribution") based on where the state falls in the national distribution.

4. **Final Rendering:** Calls `build_state_summary_text` to generate the formatted string and renders it into the dedicated text subplot, applying manual spacing adjustments to ensure a clean final presentation.

In [24]:
```
def plot_state_analysis(State):
    results = all_state_stats[State]

    # Give the text panel more room and reduce layout conflicts
    fig, axes = plt.subplots(
        4, 1,
        figsize=(14, 24),
        gridspec_kw={"height_ratios": [1, 1, 1, 1.6]}
    )

    plot_types = [
        (f"{State}_incoming", "Incoming Trade (Buying from India)"),
        (f"{State}_outgoing", "Outgoing Trade (Selling to India)"),
        (f"{State}_within",   "Within State Trade (Internal Logistics)")
    ]

    # -------------------------
    # Charts
    # -------------------------
    for i, (key, title) in enumerate(plot_types):
        df = results[key]
        ax1 = axes[i]

        color_count = '#d62728'
        color_val = '#1f77b4'

        ax1.set_xlabel('Date', fontsize=10)
        ax1.set_ylabel('Count of E-Way Bills', color=color_count, fontsize=11, font
        ax1.plot(
            df['Formatted_Date'], df['Count of E-Way Bills'],
            color=color_count, marker='.', linestyle='-',
            linewidth=1.5, alpha=0.8, label='Count'
        )
        ax1.tick_params(axis='y', labelcolor=color_count)
        ax1.grid(True, alpha=0.3)

        ax2 = ax1.twinx()
        ax2.set_ylabel('Value (INR Crores)', color=color_val, fontsize=11, fontweig
        ax2.plot(
            df['Formatted_Date'], df['Value of E-Way Bills'],
            color=color_val, linestyle='--', linewidth=2, label='Value'
        )
        ax2.tick_params(axis='y', labelcolor=color_val)

        ax1.set_title(title, fontsize=14, fontweight='bold')

        lines_1, labels_1 = ax1.get_legend_handles_labels()
        lines_2, labels_2 = ax2.get_legend_handles_labels()
        ax1.legend(lines_1 + lines_2, labels_1 + labels_2, loc='upper left', fontsi

    # -------------------------
```

```
    # Metrics
    # -------------------------
    abs_ext = results.get(f"{State}_avg_absolute_value_12_mo_external", 0)
    abs_int = results.get(f"{State}_avg_absolute_value_12_mo_internal", 0)
    abs_tot = results.get(f"{State}_avg_absolute_value_12_mo_total", 0)

    cagr_tot_val = results.get(f"{State}_cagr_total_value", None)
    cagr_int_val = results.get(f"{State}_cagr_within_value", None)

    corr_incoming_outgoing = results.get(f"{State}_correlation_bw_incoming_outgoing
    corr_internal_external = results.get(f"{State}_correlation_bw_external_trade_an

    corr_vc_in  = results.get(f"{State}_correlation_bw_count_and_value_incoming", N
    corr_vc_out = results.get(f"{State}_correlation_bw_count_and_value_outgoing", N
    corr_vc_int = results.get(f"{State}_correlation_bw_count_and_value_within", Non

    # -------------------------
    # Benchmarks
    # -------------------------
    bench_corr_io  = globals().get("correlation_info_incoming_outgoing", {}) or {}
    bench_corr_ie  = globals().get("correlation_info_internal_external", {}) or {}

    bench_vc_in    = globals().get("correlation_value_count_incoming", {}) or {}
    bench_vc_out   = globals().get("correlation_value_count_outgoing", {}) or {}
    bench_vc_int   = globals().get("correlation_value_count_within", {}) or {}

    bench_cagr_tot = globals().get("cagr_total_trade_value", {}) or {}
    bench_cagr_int = globals().get("cagr_internal_trade_value", {}) or {}

    # -------------------------
    # High-level insight generators
    # -------------------------
    def is_nan(x):
        return isinstance(x, float) and x != x

    def analyze_cagr(value, bench, label):
        if value is None or is_nan(value):
            return f"{label} growth insight not available."
        p25, avg, p75 = bench.get("p25"), bench.get("avg"), bench.get("p75")
        if p25 is None or avg is None or p75 is None:
            if value < 0:
                return f"{label} appears to be contracting."
            elif value < 5:
                return f"{label} looks mature with slow expansion."
            elif value < 15:
                return f"{label} shows steady, incremental growth."
            else:
                return f"{label} is growing rapidly, potentially aided by a low bas
        if value < p25:
            return (
                f"{label} is in the bottom quartile. "
                "This can reflect a mature base, weaker momentum, or a shifting sup
            )
        elif value < avg:
            return (
                f"{label} is below average. "
```

```
                "Often consistent with stable but slower-moving trade cycles."
            )
        elif value < p75:
            return (
                f"{label} is above average. "
                "Suggests improving integration, competitiveness, or consumption st
            )
        else:
            return (
                f"{label} is in the top quartile. "
                "May indicate structural scale-up or accelerated catch-up from a sm
            )

    def analyze_corr(value, bench, label, low_meaning, high_meaning):
        if value is None or is_nan(value):
            return f"{label} insight not available."
        p25, avg, p75 = bench.get("p25"), bench.get("avg"), bench.get("p75")
        if p25 is None or avg is None or p75 is None:
            return low_meaning if value < 0.3 else high_meaning if value > 0.7 else
                   "The relationship is moderate, suggesting partial synchronizatio
        if value < p25:
            return f"{label} is bottom-quartile. {low_meaning}"
        elif value < avg:
            return f"{label} is below average. The linkage exists but is not domina
        elif value < p75:
            return f"{label} is above average. This suggests coherent trade rhythms
        else:
            return f"{label} is top-quartile. {high_meaning}"

    meaning_low_io = (
        "Lower alignment can signal specialization, a consumption-skewed profile, "
        "or a supply chain that is not tightly closed locally."
    )
    meaning_high_io = (
        "Higher alignment often reflects consistent procurement-to-dispatch cycles
        "and stable two-way integration."
    )

    meaning_low_ie = (
        "Weaker coupling can imply thinner internal diffusion of cross-border flows
        "or uneven market depth."
    )
    meaning_high_ie = (
        "Stronger coupling suggests external trade is being absorbed and redistribu
        "indicating deeper intra-state market connectivity."
    )

    meaning_low_vc = (
        "Weaker value-count linkage suggests volatility in shipment sizes or produc
        "often reflecting heterogeneous demand or episodic high-value movement."
    )
    meaning_high_vc = (
        "Stronger linkage suggests stable scaling of trade intensity and a more pre
    )

    comment_cagr_total = analyze_cagr(cagr_tot_val, bench_cagr_tot, "Total trade")
```

```
    comment_cagr_internal = analyze_cagr(cagr_int_val, bench_cagr_int, "Internal lo

    comment_io = analyze_corr(
        corr_incoming_outgoing, bench_corr_io,
        "Incoming vs outgoing alignment",
        meaning_low_io, meaning_high_io
    )
    comment_ie = analyze_corr(
        corr_internal_external, bench_corr_ie,
        "Internal vs external coupling",
        meaning_low_ie, meaning_high_ie
    )

    comment_vc_in = analyze_corr(
        corr_vc_in, bench_vc_in,
        "Incoming basket stability",
        meaning_low_vc, meaning_high_vc
    )
    comment_vc_out = analyze_corr(
        corr_vc_out, bench_vc_out,
        "Outgoing basket stability",
        meaning_low_vc, meaning_high_vc
    )
    comment_vc_int = analyze_corr(
        corr_vc_int, bench_vc_int,
        "Internal basket stability",
        meaning_low_vc, meaning_high_vc
    )

    # -------------------------
    # Build compact, wrapped text
    # -------------------------
    analysis_text = build_state_summary_text(
        State,
        abs_tot, abs_ext, abs_int,
        cagr_tot_val, cagr_int_val,
        corr_incoming_outgoing, corr_internal_external,
        corr_vc_in, corr_vc_out, corr_vc_int,
        bench_cagr_tot=bench_cagr_tot,
        bench_cagr_int=bench_cagr_int,
        bench_corr_io=bench_corr_io,
        bench_corr_ie=bench_corr_ie,
        bench_vc_in=bench_vc_in,
        bench_vc_out=bench_vc_out,
        bench_vc_int=bench_vc_int,
        comment_cagr_total=comment_cagr_total,
        comment_cagr_internal=comment_cagr_internal,
        comment_io=comment_io,
        comment_ie=comment_ie,
        comment_vc_in=comment_vc_in,
        comment_vc_out=comment_vc_out,
        comment_vc_int=comment_vc_int,
        width=92
    )

    ax_text = axes[3]
```

```
    ax_text.axis("off")
    ax_text.text(
        0.01, 0.99, analysis_text,
        transform=ax_text.transAxes,
        va="top", ha="left",
        fontsize=10,
        wrap=True
    )

    # Replace tight_layout with a safer manual spacing
    fig.subplots_adjust(hspace=0.35, top=0.95, bottom=0.05)
    plt.show()


# Trigger the interactive widget
interact(plot_state_analysis, State=indian_states);
```

```
interactive(children=(Dropdown(description='State', options=('ANDAMAN AND NICOBAR',
'ANDHRA PRADESH', 'ARUNACH…
```

5. Combining Employment Data with the Merchandise Trade Data and Generating Analysis

First we read the Employment Data

In [25]:
```
employment_data = pl.read_csv("Datasets/India_Employment.csv")
employment_data.head(5)
```

Out[25]: shape: (5, 7)

Country	Year	Gender	State Name	Type Of Area	Status Of Employment	Percentage Distribution Of Workers (UOM:% (Percentage)), Scaling Factor:1
str	str	str	str	str	str	f64
"India"	"Agriculture Year (Jul - Jun), ...	"Female"	"All India"	"Rural"	"All self employed"	73.5
"India"	"Agriculture Year (Jul - Jun), ...	"Female"	"All India"	"Rural"	"Casual labour"	18.7
"India"	"Agriculture Year (Jul - Jun), ...	"Female"	"All India"	"Rural"	"Regular wage/salary"	7.8
"India"	"Agriculture Year (Jul - Jun), ...	"Female"	"All India"	"Rural"	"Self-Employed helper in hourse...	42.3
"India"	"Agriculture Year (Jul - Jun), ...	"Female"	"All India"	"Rural"	"Self-Employed own account work...	31.2

Then the employment status names are identified, so that the exact values can be used in later analysis

In [26]:
```
# Find types of employment
employment_types = employment_data.select(
    pl.col("Status Of Employment").unique()
)
print(employment_types)
```

```
shape: (6, 1)
┌──────────────────────────────┐
│ Status Of Employment         │
│ ---                          │
│ str                          │
╞══════════════════════════════╡
│ Self-Employed helper in hourse… │
│ Self-Employed own account work… │
│ Casual labour                │
│ Regular wage/salary          │
│ All self employed            │
│ all                          │
└──────────────────────────────┘
```

Checking the years for which the data is available, to merge it appropriately with the E-Way Bill data later on

In [27]:
```
# Find unique years in the data
unique_years = employment_data.select(
    pl.col("Year").unique()
)
for values in unique_years.iter_rows():
    print(values)
```

```
('Agriculture Year (Jul - Jun), 2018',)
('Agriculture Year (Jul - Jun), 2017',)
('Agriculture Year (Jul - Jun), 2021',)
('Agriculture Year (Jul - Jun), 2019',)
('Agriculture Year (Jul - Jun), 2022',)
('Agriculture Year (Jul - Jun), 2020',)
('Agriculture Year (Jul - Jun), 2023',)
```

Generating the list of state names in the Employment Dataset and removing some values that we are not concerned about

This will be used to create a mapping between state names, and generate maps later on

In [28]:
```
# Get a list of states in the employment dataset
states_in_employment_dataset = employment_data.select(
    pl.col("State Name").unique()
).filter(
    ~pl.col("State Name").is_in(["All India", "Dadra and Nagar Haveli and Daman and
)
```

Checking the regions for which data is available

In [29]:
```
unique_area = employment_data.select(
    pl.col("Type Of Area").unique()
)
unique_area
```

Out[29]: shape: (3, 1)

Type Of Area
str
"Rural + Urban"
"Rural"
"Urban"

`Overall ambition`

1. See for each state how much CAGR in E-Way bill overall is linked to rise in regular wage/salary employment (men, women, persons)

2. See for each state how much CAGR in E-Way bill overall is linked to fall in casual labour (men, women, persons)

- This will require me to find correlations between both polars dataframes
- Years for analysis can only be 2019 to 2023 (as this is the common time period for which data is available)
- The time period to match should be from July to June (to match the agricultural year logic)
- State names need to be matched in both dataframes
- I will do the analysis at the Rural + Urban level only (otherwise there will be too many variables to consider) """

5.1 Obtaining only the data that we need

In [30]:
```
"""
This block performs the following data cleaning and filtering operations:
1. Filters for the specific Agriculture Years (2019-2023) relevant to the study.
2. Removes 'All India' aggregates and specific Union Territories (like Daman and Di
3. Selects the 'Rural + Urban' category to capture the comprehensive workforce data
4. Filters for the two key employment types: 'Regular wage/salary' and 'Casual labo
5. Selects only the necessary columns and transforms the 'Year' column from a descr
"""

employment_data_filtered = employment_data.filter(
    pl.col("Year").is_in(["Agriculture Year (Jul - Jun), 2023", "Agriculture Year (
    ~pl.col("State Name").is_in(["All India", "Dadra and Nagar Haveli and Daman and
    pl.col("Type Of Area").is_in(["Rural + Urban"]),
    pl.col("Status Of Employment").is_in(["Regular wage/salary", "Casual labour"])
).select(
    ["State Name", "Year", "Status Of Employment", "Gender", "Percentage Distributi
).with_columns(
    Year = pl.col("Year").str.slice(-4).cast(pl.Int32)
)

employment_data_filtered
```

Out[30]: shape: (1_026, 5)

State Name	Year	Status Of Employment	Gender	Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Factor:1
str	i32	str	str	f64
"Andaman and Nicobar Islands"	2023	"Casual labour"	"Female"	3.0
"Andaman and Nicobar Islands"	2023	"Regular wage/salary"	"Female"	47.2
"Andhra Pradesh"	2023	"Casual labour"	"Female"	34.3
"Andhra Pradesh"	2023	"Regular wage/salary"	"Female"	17.3
"Arunachal Pradesh"	2023	"Casual labour"	"Female"	0.6
…	…	…	…	…
"Uttar Pradesh"	2019	"Regular wage/salary"	"Persons"	15.2
"Uttarakhand"	2019	"Casual labour"	"Persons"	10.0
"Uttarakhand"	2019	"Regular wage/salary"	"Persons"	26.1
"West Bengal"	2019	"Casual labour"	"Persons"	28.3
"West Bengal"	2019	"Regular wage/salary"	"Persons"	23.5

Generating state names from the E-way bill dataset again to generate a mapping

In [31]:
```
state_names_eway_dataset = e_way_bill_data["State/Ut Name"].unique()
state_names_eway_dataset
```

Out[31]: shape: (34,)

State/Ut Name
str
"Arunachal Pradesh"
"Gujarat"
"Uttar Pradesh"
"Chhattisgarh"
"Nagaland"
…
"Ladakh"
"Karnataka"
"Assam"
"Telangana"
"Madhya Pradesh"

5.2 Creating the mapping

In [32]:
```
state_name_mapping = {
    # Mapped to match Employment Data
    "ANDAMAN AND NICOBAR": "Andaman and Nicobar Islands",
    "ANDHRA PRADESH": "Andhra Pradesh",
    "ARUNACHAL PRADESH": "Arunachal Pradesh",
    "ASSAM": "Assam",
    "BIHAR": "Bihar",
    "CHANDIGARH": "Chandigarh",
    "CHHATTISGARH": "Chhattisgarh",
    "DELHI": "Delhi",
    "GOA": "Goa",
    "GUJARAT": "Gujarat",
    "HARYANA": "Haryana",
    "HIMACHAL PRADESH": "Himachal Pradesh",
    "JAMMU AND KASHMIR": "Jammu and Kashmir",
    "JHARKHAND": "Jharkhand",
    "KARNATAKA": "Karnataka",
    "KERALA": "Kerala",
    "LADAKH": "Ladakh",
    "LAKSHADWEEP": "Lakshadweep",
    "MADHYA PRADESH": "Madhya Pradesh",
    "MAHARASHTRA": "Maharashtra",
    "MANIPUR": "Manipur",
    "MEGHALAYA": "Meghalaya",
    "MIZORAM": "Mizoram",
    "NAGALAND": "Nagaland",
    "ODISHA": "Odisha",
```

```
    "PUNJAB": "Punjab",
    "RAJASTHAN": "Rajasthan",
    "SIKKIM": "Sikkim",
    "TAMIL NADU": "Tamilnadu",  # Note: Target DF uses concatenated spelling
    "TELANGANA": "Telangana",
    "TRIPURA": "Tripura",
    "UTTAR PRADESH": "Uttar Pradesh",
    "UTTARAKHAND": "Uttarakhand",
    "WEST BENGAL": "West Bengal",

    # Explicit Removals (Mapped to None)
    "DADRA AND NAGAR HAVELI": None,
    "DAMAN AND DIU": None,
    "PUDUCHERRY": None,
    "Other Territory": None
}
```

5.3 Creating Helper Function to generate correlations between different employment types and E-way bill values

`give_correlation` calculates six different correlation values between E-way bill data and employment statistics for a specific state.

It takes a state name and two DataFrames (df1 for E-way bill data and df2 for employment data) as input.

First, it preprocesses both DataFrames by filtering for the given state and sorting by year.

For the employment data (df2), it creates six separate subsets based on employment status ("Casual labour" or "Regular wage/salary") and gender ("Male", "Female", or overall).

Then, for each of the six employment categories (Casual Male, Casual Female, Casual Overall, Formal Male, Formal Female, Formal Overall), it performs an inner join with the E-way bill data on the "Year" column.

After joining, it calculates the correlation between the "Total E-way Bill Value" and the "Percentage of Workers" for that specific category.

Finally, the function returns a tuple containing all six calculated correlation values.

In [33]:
```
def give_correlation(state_name, df1, df2):
    """
    Helper function for the function below
    Aim is to return a tuple with 6 values with all the correlations
    """
    df1_prepped = df1.filter(
        pl.col("State/UT") == state_name
    ).sort(
        pl.col("Year")
    )

    df2_casual_men_prepped = df2.filter(
```

```
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Casual labour",
        pl.col("Gender") == "Male"
    ).sort(
        pl.col("Year")
    )

    df2_casual_female_prepped = df2.filter(
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Casual labour",
        pl.col("Gender") == "Female"
    ).sort(
        pl.col("Year")
    )

    df2_casual_overall_prepped = df2.filter(
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Casual labour",
    ).sort(
        pl.col("Year")
    )

    df2_formal_men_prepped = df2.filter(
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Regular wage/salary",
        pl.col("Gender") == "Male"
    ).sort(
        pl.col("Year")
    )

    df2_formal_female_prepped = df2.filter(
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Regular wage/salary",
        pl.col("Gender") == "Female"
    ).sort(
        pl.col("Year")
    )

    df2_formal_overall_prepped = df2.filter(
        pl.col("State Name") == state_name,
        pl.col("Status Of Employment") == "Regular wage/salary",
    ).sort(
        pl.col("Year")
    )

    correlation_casual_male_labour = df2_casual_men_prepped.join(
        df1_prepped,
        on="Year",
        how="inner"
    ).select(
        pl.col("Year"),
        pl.col("Total E-way Bill Value"),
        pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
    ).select(
        pl.corr("Total E-way Bill Value", "Percentage of Workers")
    ).item()
```

```
correlation_casual_female_labour = df2_casual_female_prepped.join(
    df1_prepped,
    on="Year",
    how="inner"
).select(
    pl.col("Year"),
    pl.col("Total E-way Bill Value"),
    pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
).select(
    pl.corr("Total E-way Bill Value", "Percentage of Workers")
).item()

correlation_casual_overall_labour = df2_casual_overall_prepped.join(
    df1_prepped,
    on="Year",
    how="inner"
).select(
    pl.col("Year"),
    pl.col("Total E-way Bill Value"),
    pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
).select(
    pl.corr("Total E-way Bill Value", "Percentage of Workers")
).item()

correlation_formal_male_labour = df2_formal_men_prepped.join(
    df1_prepped,
    on="Year",
    how="inner"
).select(
    pl.col("Year"),
    pl.col("Total E-way Bill Value"),
    pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
).select(
    pl.corr("Total E-way Bill Value", "Percentage of Workers")
).item()

correlation_formal_female_labour = df2_formal_female_prepped.join(
    df1_prepped,
    on="Year",
    how="inner"
).select(
    pl.col("Year"),
    pl.col("Total E-way Bill Value"),
    pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
).select(
    pl.corr("Total E-way Bill Value", "Percentage of Workers")
).item()

correlation_formal_overall_labour = df2_formal_overall_prepped.join(
    df1_prepped,
    on="Year",
    how="inner"
).select(
    pl.col("Year"),
    pl.col("Total E-way Bill Value"),
```

```
        pl.col("Percentage Distribution Of Workers (UOM:%(Percentage)), Scaling Fac
    ).select(
        pl.corr("Total E-way Bill Value", "Percentage of Workers")
    ).item()

    return (correlation_casual_male_labour, correlation_casual_female_labour, corre
    correlation_formal_male_labour, correlation_formal_female_labour, correlation_f
```

5.4 Obtaining the Correlations that will be plotted

`find_correlation_bw_ewayvalue_and_employment`

This function calculates the correlation analysis between E-way bills and Employment stats across Indian states.

1. **Data Preparation (E-Way Bills):**
    - **Selection & Cleaning:** Extracts relevant columns (State, Month, Value) and standardizes state names using `state_name_mapping` to ensure they match the employment dataset.
    - **Time Filtering:** Extracts the year from the date string and filters the dataset to focus on the period **2019–2023**.
    - **Aggregation:** Groups the data by **State** and **Year** to calculate the total annual E-way bill value for each region.
2. **Correlation Computation:**
    - Iterates through each unique state in the processed dataset.
    - Calls the helper function `give_correlation` to compute six distinct correlation coefficients (Casual vs. Formal labor for Males, Females, and Overall).
3. **Result Compilation:**
    - Stores the calculated correlations in a dictionary structure.
    - Includes a specific mapping fix for **Tamil Nadu** to handle naming inconsistencies between datasets.
    - **Returns:** A dictionary where every state maps to its specific set of six employment-trade correlation metrics.

In [34]:
```
def find_correlation_bw_ewayvalue_and_employment(eway_bill_data, employment_data):
    """
    First I will find the state specific information from the e-way bill dataset.
    The total value of E-way bills (incoming + outgoing + internal) will be used

    The state names from that dataset need to be corrected so that they can be mapp

    Aim is to return a dictionary that contains the correlation value for each stat
    """
    eway_bill_data_cleaned = eway_bill_data.select(
        ["Month", "State/Ut Name", "Assessable Value (UOM:INR(IndianRupees)), Scali
    ).with_columns(
```

```
        pl.col("State/Ut Name").str.strip_chars().replace(state_name_mapping).alias
        pl.col("Month").str.slice(-4).cast(pl.Int32)
    ).filter(
        pl.col("Month") >= 2019,
        pl.col("Month") <= 2023
    ).select(
        pl.col("State/UT"),
        pl.col("Assessable Value (UOM:INR(IndianRupees)), Scaling Factor:10000000")
        pl.col("Month").alias("Year")
    ).group_by(
        pl.col("State/UT"),
        pl.col("Year")
    ).agg(
        pl.col("Total E-way Bill Value").sum()
    )

    correlation_data_for_states = {}

    for state_name in eway_bill_data_cleaned["State/UT"].unique():
        (correlation_casual_male_labour, correlation_casual_female_labour, correlat
        correlation_formal_male_labour, correlation_formal_female_labour, correlati

        # Need to take Tamil Nadu into special consideration (name mismatch in data
        if state_name == "Tamilnadu":
            correlation_data_for_states["Tamil Nadu"] = {
                "Casual Male Labour": correlation_casual_male_labour,
                "Casual Female Labour": correlation_casual_female_labour,
                "Casual Overall Labour": correlation_casual_overall_labour,
                "Formal Male Labour": correlation_formal_male_labour,
                "Formal Female Labour": correlation_formal_female_labour,
                "Formal Overall Labour": correlation_formal_overall_labour
            }
        else:
            correlation_data_for_states[state_name] = {
                "Casual Male Labour": correlation_casual_male_labour,
                "Casual Female Labour": correlation_casual_female_labour,
                "Casual Overall Labour": correlation_casual_overall_labour,
                "Formal Male Labour": correlation_formal_male_labour,
                "Formal Female Labour": correlation_formal_female_labour,
                "Formal Overall Labour": correlation_formal_overall_labour
            }


    return correlation_data_for_states

employment_correlations = find_correlation_bw_ewayvalue_and_employment(e_way_bill_d
```

5.5 Generating the Plots to see the correlations (which are analyzed in the report)

`plot_employment_correlation_map`

This function generates a series of choropleth maps to visualize the relationship between Merchandise Trade (E-Way Bill Value) and various Employment categories (Formal/Casual, Male/Female) across Indian states.

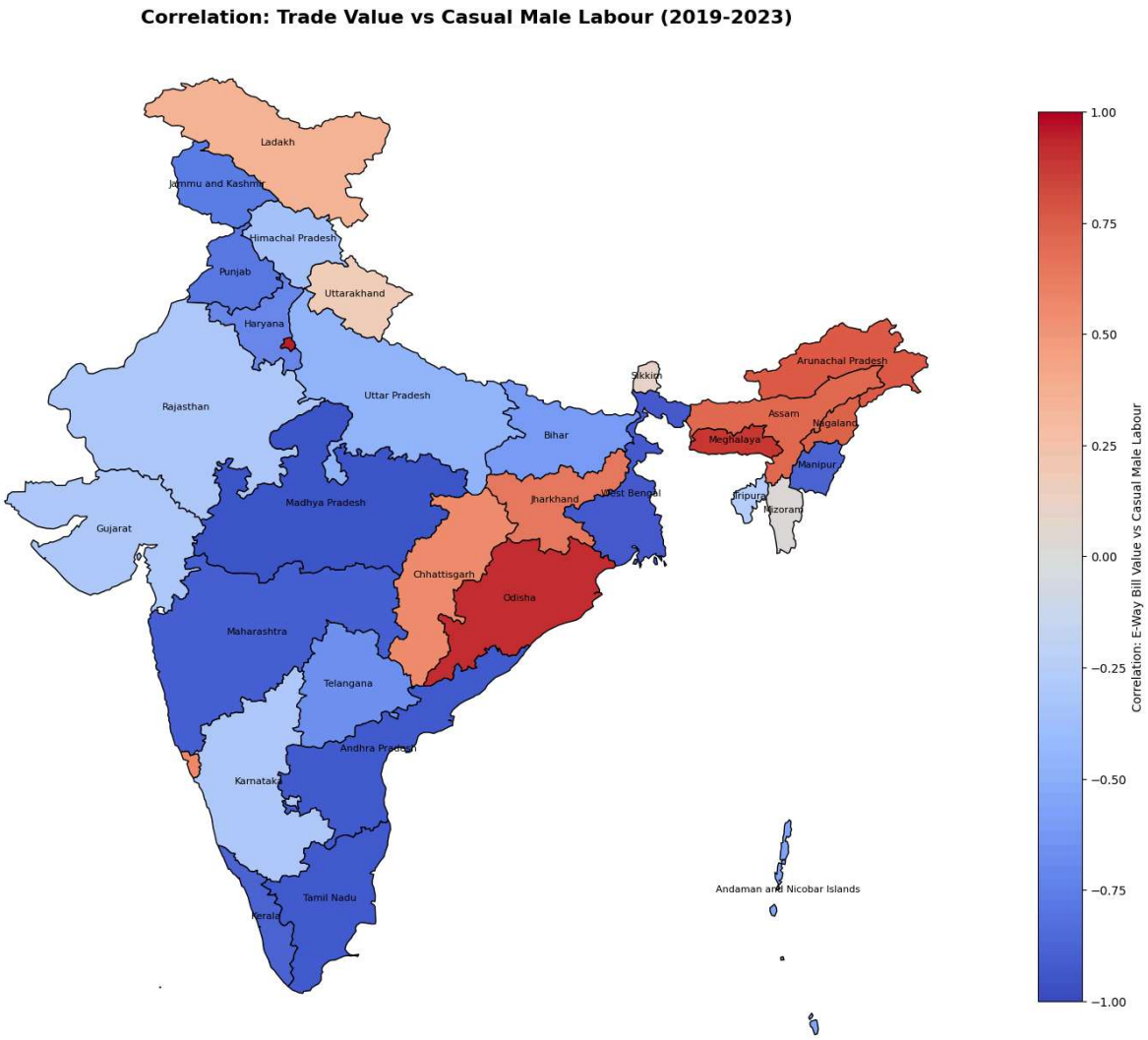
1. **Data Extraction:**
   - Iterates through the `employment_correlations` dictionary.
   - Extracts the specific correlation value (e.g., "Casual Male Labour") for each state and builds a temporary dictionary.
2. **Geospatial Integration:**
   - Converts the extracted data into a Polars DataFrame.
   - Performs an **inner join** with the `gdf_pl` (Geospatial Data) based on state names to attach geometry to the correlation values.
3. **Map Rendering:**
   - Converts the data to a GeoDataFrame and handles geometry parsing (WKT).
   - Plots the data on a map of India.
   - **Colormap:** Uses `'coolwarm'` (Red-Blue diverging scale). This is critical for correlation matrices because it clearly distinguishes between **positive correlations** (Red, value > 0) and **negative correlations** (Blue, value < 0).
   - **Scale:** Fixes the scale (`vmin=-1` , `vmax=1`) to ensure all 6 maps are directly comparable.
4. **Annotation & Formatting:**
   - Adds text labels (State Names) to the centroids of larger states.
   - Sets a dynamic title based on the specific employment type being analyzed.
5. **Execution:**
   - The loop at the bottom iterates through 6 distinct employment categories (Casual/Formal x Male/Female/Overall), generating a separate map for each to allow for comparative analysis.

In [35]:
```
# Generating the Plots for each of these correlations


def plot_employment_correlation_map(correlation_data, correlation_type, gdf_pl):
    # Extract data for the specific correlation type
    data = {"state": [], "value": []}
    for state, metrics in correlation_data.items():
        val = metrics.get(correlation_type)
        if val is not None:
            data["state"].append(state)
            data["value"].append(val)

    # Create Polars DataFrame
    df = pl.DataFrame(data)
```

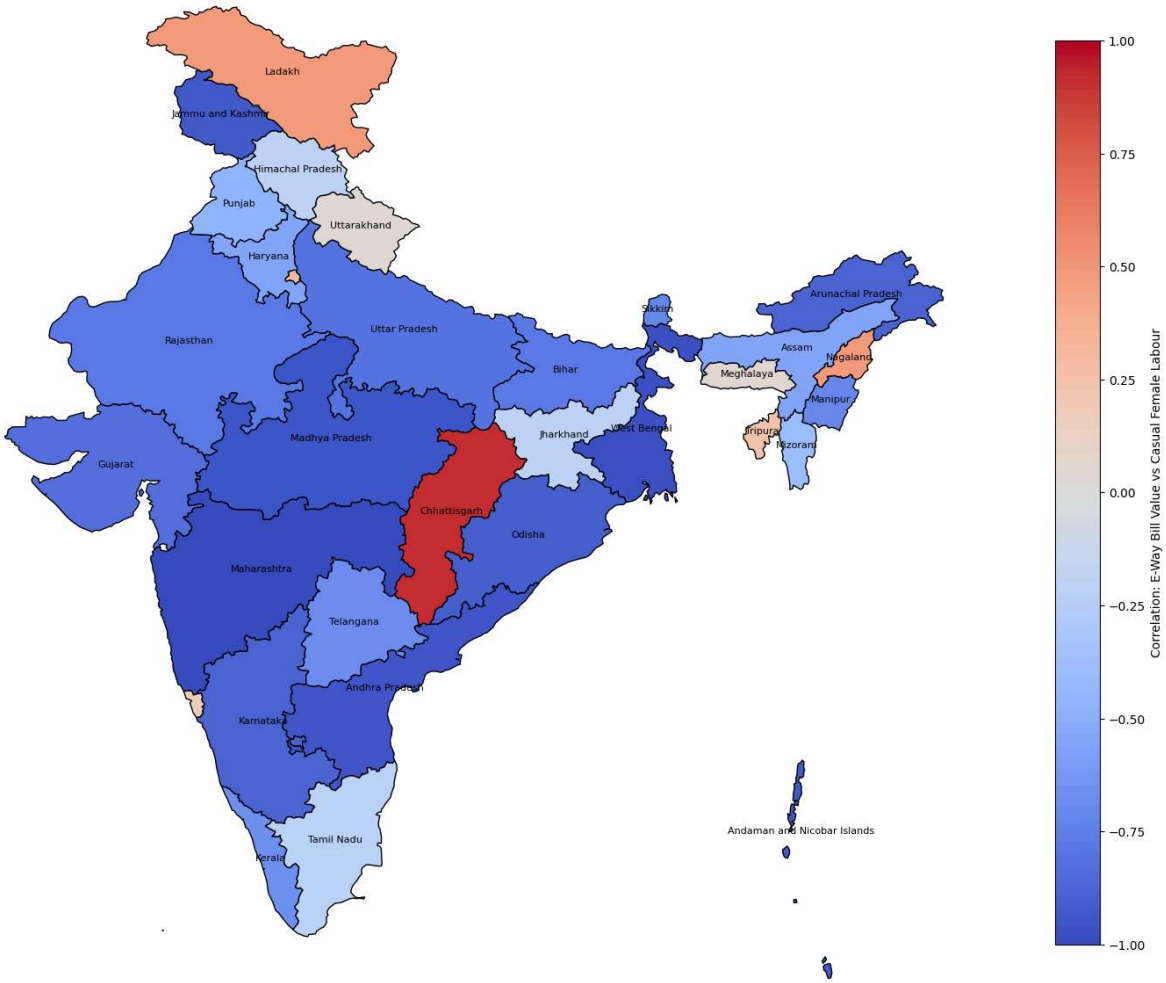
```
    # Join with GeoJSON data
    # Note: Assuming 'state' in df matches 'st_nm' in gdf_pl (both Title Case)
    merged_pl = df.join(
        gdf_pl,
        left_on="state",
        right_on="st_nm",
        how="inner"
    ).select(
        pl.col("state").alias("st_nm"),
        pl.col("value"),
        pl.col("geometry")
    )

    # Convert to Pandas and then GeoDataFrame
    merged_pandas = merged_pl.to_pandas()
    merged_pandas['geometry'] = merged_pandas['geometry'].apply(wkt.loads)
    merged_gdf = gpd.GeoDataFrame(merged_pandas, geometry='geometry')

    # Dissolve by state name to handle any duplicate geometries
    merged_gdf = merged_gdf.dissolve(by='st_nm', aggfunc='first').reset_index()

    # Plotting
    fig, ax = plt.subplots(1, 1, figsize=(15, 15))
    merged_gdf.plot(
        column='value',
        ax=ax,
        legend=True,
        legend_kwds={
            'label': f"Correlation: E-Way Bill Value vs {correlation_type}",
            'orientation': "vertical",
            'shrink': 0.7
        },
        cmap='coolwarm', # Diverging colormap is good for correlation (-1 to 1)
        edgecolor='black',
        vmin=-1, # Fix scale from -1 to 1 for consistent comparison
        vmax=1
    )

    # Add Labels
    for idx, row in merged_gdf.iterrows():
        if row.geometry.area > 0.5:
            plt.annotate(
                text=row['st_nm'],
                xy=row.geometry.centroid.coords[0],
                horizontalalignment='center',
                fontsize=8,
                color='black'
            )

    ax.set_title(f'Correlation: Trade Value vs {correlation_type} (2019-2023)', fon
    ax.set_axis_off()
    plt.tight_layout()
    plt.show()

# 4. Generate plots for all 6 types
correlation_types = [
```

```
    "Casual Male Labour",
    "Casual Female Labour",
    "Casual Overall Labour",
    "Formal Male Labour",
    "Formal Female Labour",
    "Formal Overall Labour"
]

for c_type in correlation_types:
    print(f"Generating plot for: {c_type}")
    plot_employment_correlation_map(employment_correlations, c_type, gdf_pl)
```

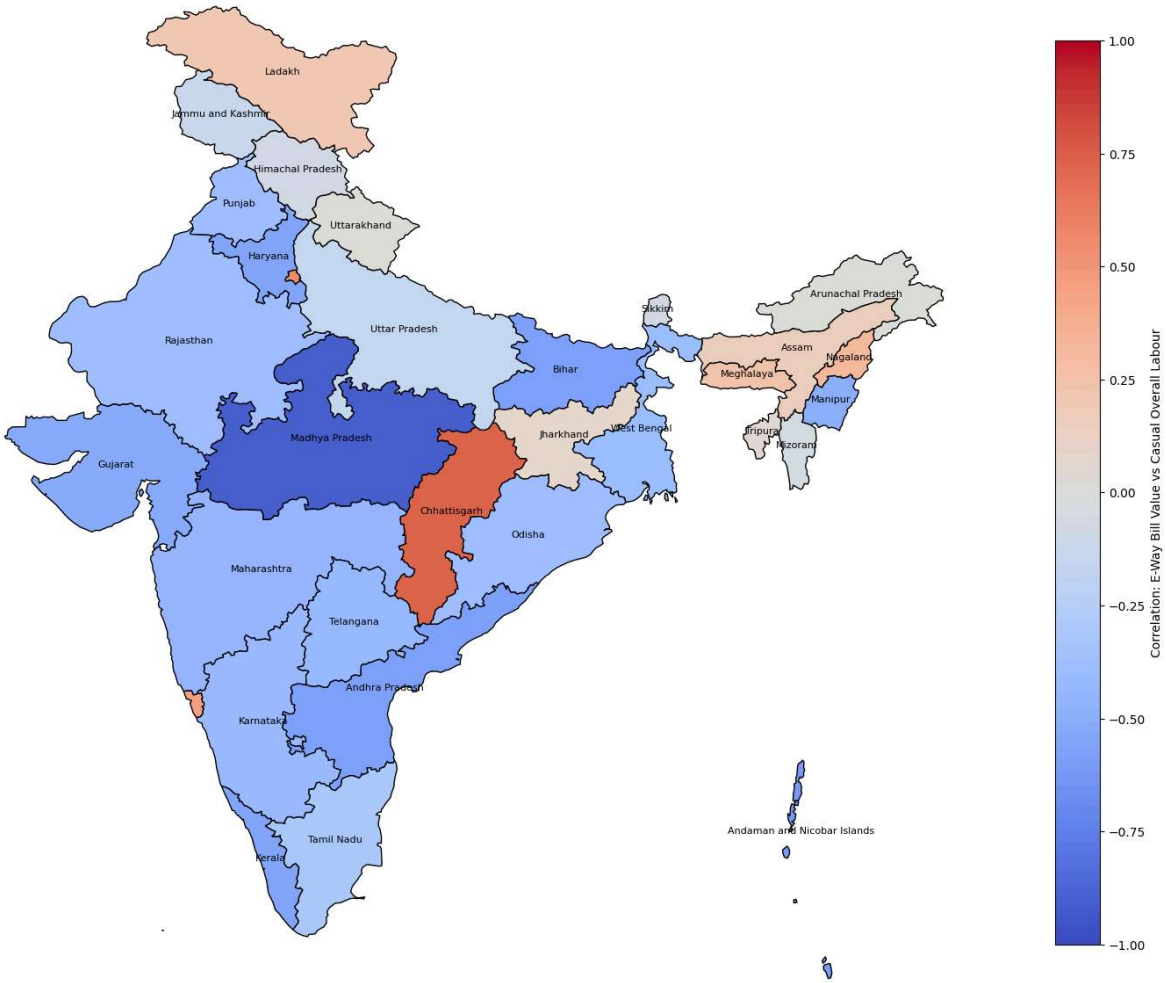
```
Generating plot for: Casual Male Labour
```



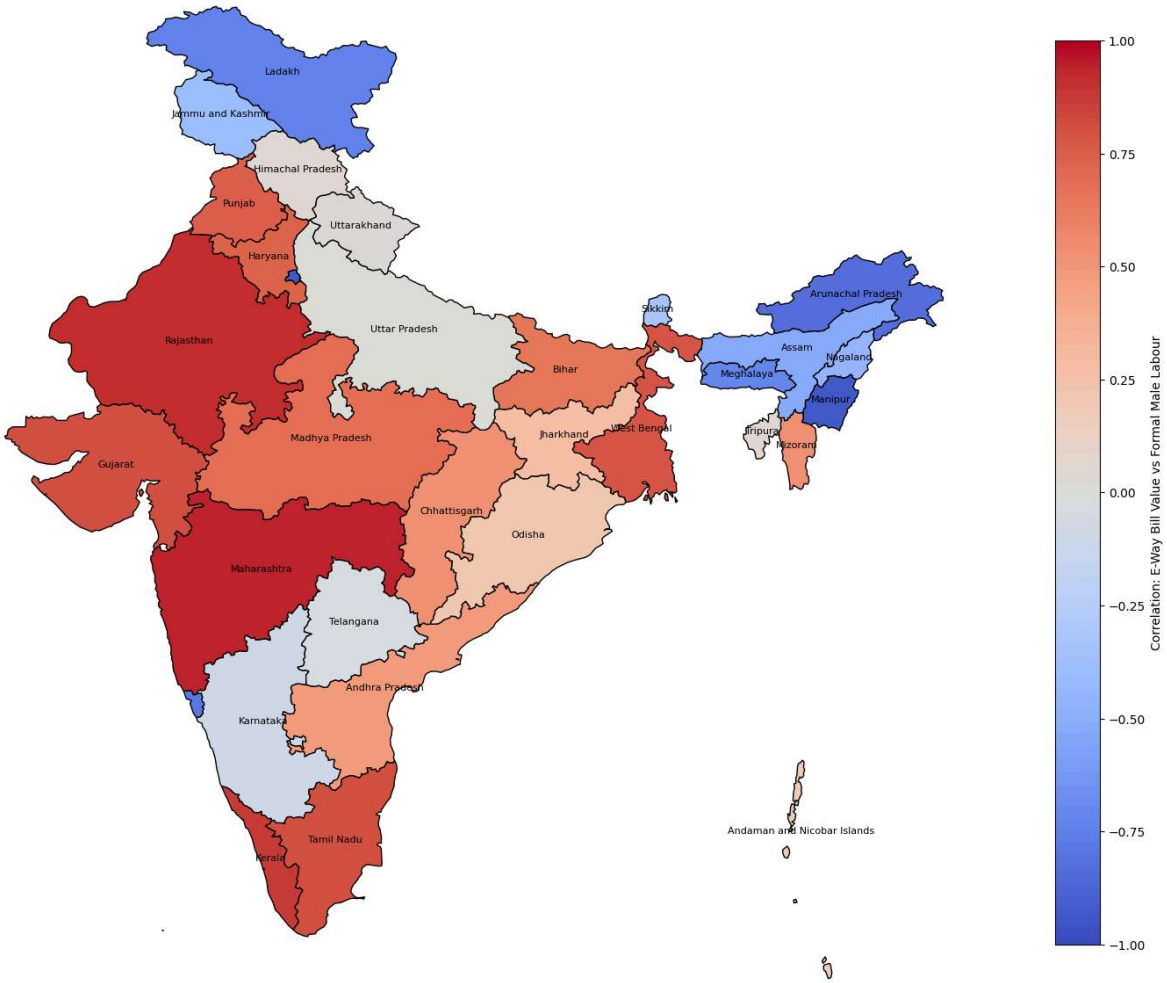

```
Generating plot for: Casual Female Labour
```



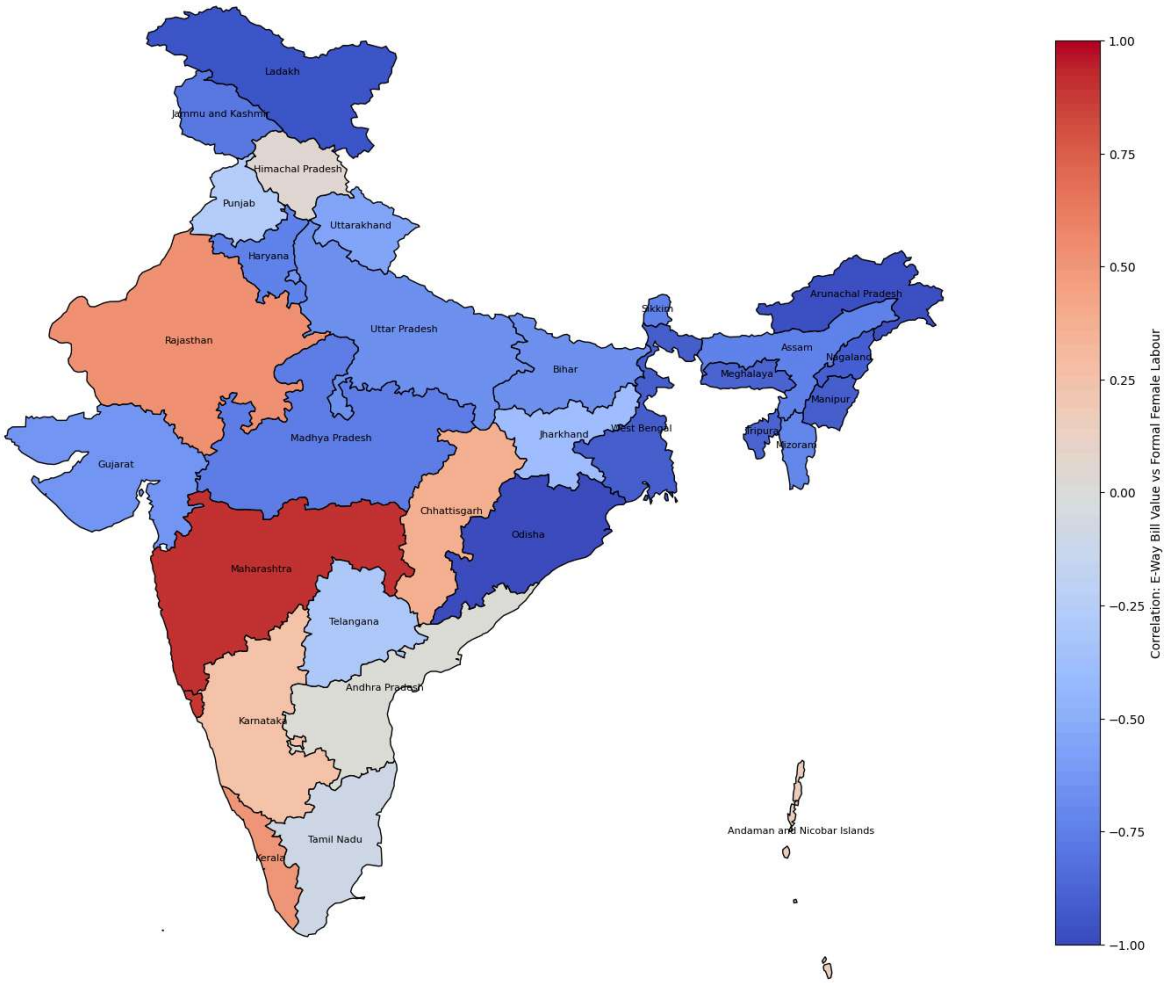

```
Generating plot for: Casual Overall Labour
```



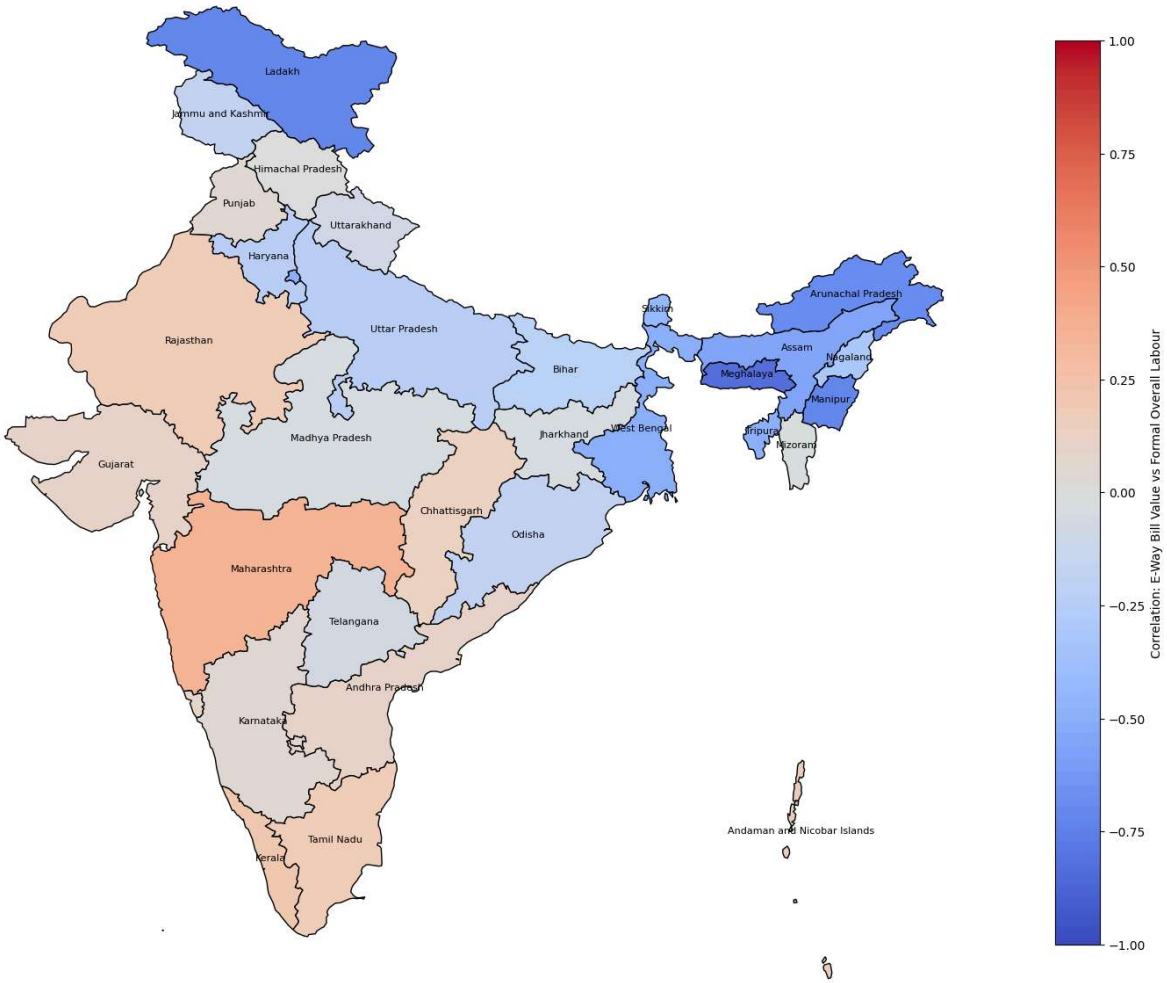

```
Generating plot for: Formal Male Labour
```




```
Generating plot for: Formal Female Labour
```




```
Generating plot for: Formal Overall Labour
```


Correlation: Trade Value vs Formal Overall Labour (2019-2023)
Ladakh
Jammu and Kashmir
Himachal Pradesh
Punjab
Uttarakhand
Haryana
Rajasthan
Uttar Pradesh
Bihar
Sikkim
Arunachal Pradesh
Assam
Nagaland
Meghalaya
Manipur
Tripura
Mizoram
Jharkhand
West Bengal
Madhya Pradesh
Gujarat
Chhattisgarh
Odisha
Maharashtra
Telangana
Andhra Pradesh
Karnataka
Tamil Nadu
Kerala
Andaman and Nicobar Islands
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Correlation: E-Way Bill Value vs Formal Overall Labour