DATA SCIENCE WITH PYTHON

PROJECT REPORT

(Project Semester January-April 2025)

Turning Real-Time Government Scheme Data into Actionable Insights

Submitted by

(KISHAN NATH)

Registration No: 12300755

Programme and Section CSE K23GR

Course Code INT375

Under the Guidance of

(Ms. Gargi Sharma)

Discipline of CSE/IT

Lovely School of Computer Science and Engineering

Lovely Professional University, Phagwara

CERTIFICATE

This is to certify that Kishan Nath bearing Registration no. 12300755 has completed INT375 project titled, **“Turning Real-Time Government Scheme Data into Actionable Insights”** under my guidance and supervision. To the best of my knowledge, the present work is the result of his/her original development, effort and study.

Signature and Name of the Supervisor

Designation of the Supervisor

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab.

Date: 12th April, 2025

DECLARATION

I, Kishan Nath, student of Computer Science and Engineering under CSE/IT Discipline at, Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own intensive work and is genuine.

Date: 12th April 2025 Signature

Registration No. 12300755 Kishan nath

Acknowledgement

I would like to express my sincere gratitude to all those who contributed to the successful completion of this project.

First and foremost, I am deeply thankful to **Gargi Sharma**, my mentor and guide, for their constant support, valuable suggestions, and encouragement throughout the duration of this project. Their insights helped me explore the dataset more meaningfully and present my findings effectively.

I would also like to thank the Department of Computer Science and Engineering, **LOVELY PROFESSIONAL UNIVERSITY**, for providing me with the necessary resources and a learning environment that made this project possible.

My heartfelt thanks to my friends and classmates who supported me during the various stages of this analysis, offering valuable feedback and motivation.

Lastly, I am grateful to the creators and maintainers of the dataset, as well as the open-source community for tools like Python, Pandas, Seaborn, and Matplotlib, which played an integral role in my data analysis journey.

This project has been a great learning experience, and I truly appreciate everyone who helped make it possible.

Exploratory Data Analysis on Turning Real-Time Government Scheme Data into Actionable Insights

Introduction.

In the era of digital governance, real-time data collection and dissemination have become pivotal tools for evaluating and enhancing the effectiveness of government schemes. With an increasing focus on transparency, accountability, and citizen-centric service delivery, various government platforms now provide live updates on scheme implementation across multiple dimensions such as total beneficiaries, Aadhaar linkage, mobile seeding, and inclusion scores.

This Exploratory Data Analysis (EDA) project aims to delve into a comprehensive dataset containing real-time metrics from government welfare schemes. The objective is to uncover patterns, detect anomalies, and derive actionable insights that can inform decision-making for policymakers, program managers, and other stakeholders.

Source of Dataset

The dataset used for this Exploratory Data Analysis was obtained from a real-time government dashboard, which tracks the implementation and outreach of various welfare schemes across states and districts in India. The data includes key indicators such as:

- *Total number of beneficiaries*
- *Aadhaar linkage status*
- *Mobile number seeding*
- *Inclusion scores*

The data was downloaded in CSV format and reflects live metrics as published by the respective government portals. While the exact source URL is not provided here, such datasets are typically available on official Indian government platforms like:

- **data.gov.in**
- **pmayg.nic.in**
- **nrega.nic.in**
- **uidai.gov.in**

EDA Process

Exploratory Data Analysis (EDA) is the process of analyzing datasets to summarize their main characteristics, often using visual methods. This section describes the step-by-step EDA process followed in this project.

3.1 Importing Required Libraries

The analysis was performed using Python with the help of libraries such as:

- `pandas` – for data manipulation

- `numpy` – for numerical operations

- `matplotlib.pyplot` and `seaborn` – for visualization

3.2 Data Loading

The dataset was loaded using `pandas` from a `.csv` file. An initial overview was done using:

```
Python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_excel('File.xlsx')
df.head(1)
df.head()  # Displays the first 5 rows
df.info()  # Gives data types and null values
df.describe()  # Statistical summary
```

3.3 Data Cleaning

- Removed rows with null or missing values.
- Standardized column names (removed trailing spaces).
- Converted relevant columns to numeric data types.

3.4 Data Transformation

- Melted the DataFrame for advanced plotting (e.g., for box plots):

  ```
  df_melted = df.melt(id_vars='srcStateName', var_name='Area Type', value_name='Area')
  ```

- Log transformation was applied to columns with large numeric ranges to reduce skewness.

  ```
  df['Net area sown_log'] = np.log1p(df['Net area sown'])
  ```

3.5 Data Visualization Setup

- Subplots were used for comparative histogram and bar plot analysis.
- Visual styles were customized using Seaborn themes and custom color palettes.

4. Analysis on Dataset

4.1 BAR PLOT

1. Introduction

A **bar plot** (also called a bar chart) is a graphical representation used to display and compare the frequency, count, or other measures (e.g., mean, total) across **categorical variables**. Bar plots are widely used in data analysis due to their simplicity and clarity in showing comparisons among discrete categories.

2. Purpose of Using a Bar Plot

- To **visually compare** values across categories.
- To identify **patterns**, **trends**, or **outliers** in categorical data.
- To present **summarized** and **aggregated** data in an easily understandable format.

3. Data Requirements

Bar plots require two main components:

- A **categorical variable** (e.g., product name, country, gender)
- A **numerical variable** (e.g., sales, counts, percentages)

4. Tools and Libraries Used

In Python, bar plots can be generated using:

- **Matplotlib**: for basic plotting
- **Seaborn**: for aesthetically pleasing and enhanced bar charts
- **Pandas**: for quick plotting with DataFrame support

5. Methodology

a. Data Preparation:

- Clean and preprocess the data
- Aggregate values by categories (e.g., sum or count)

b. Interpretation:

- Categories with **taller bars** represent higher values.
- Categories with **shorter bars** indicate lower values.
- Variations in height show the **differences across categories**.

6. Advantages of Bar Plots

- Simple to create and interpret
- Useful for both small and large datasets
- Easy to identify **comparative differences**

7. Limitations

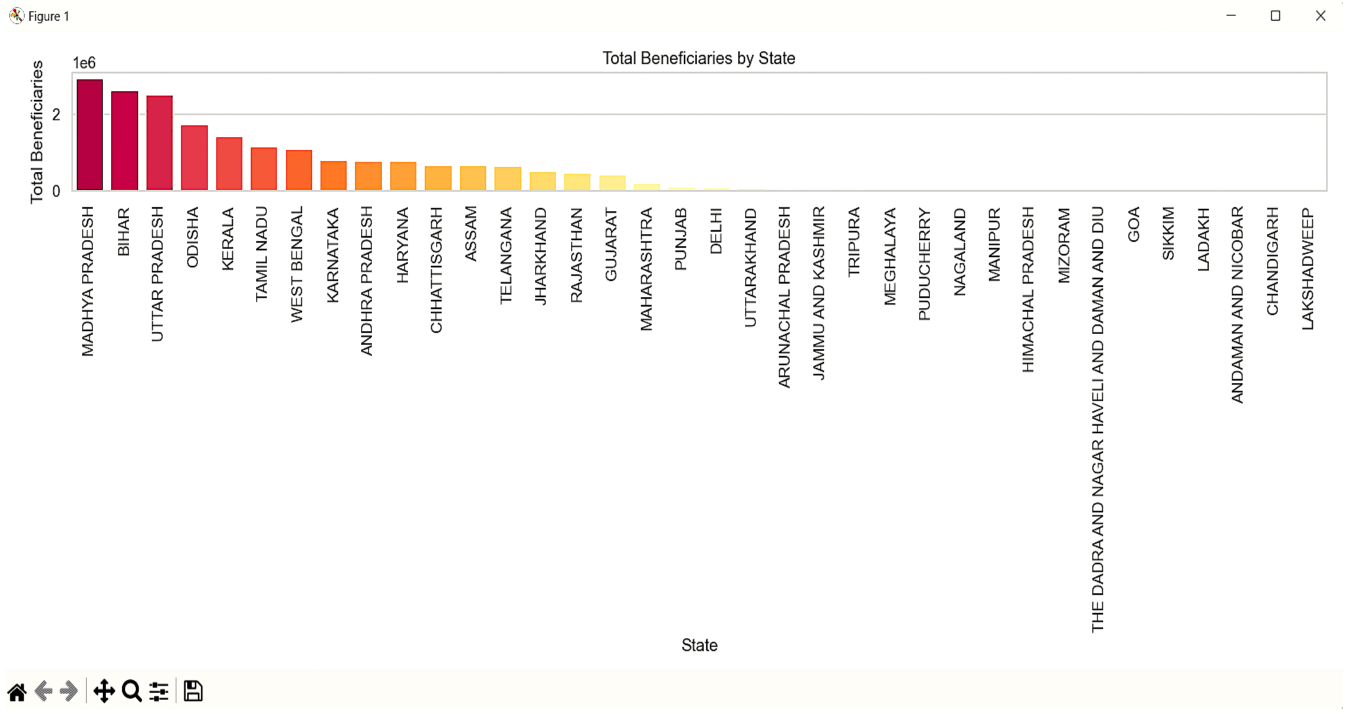
- Not suitable for showing **continuous data**
- Can become **cluttered** with too many categories
- Doesn't show **distribution within categories** (like boxplots or histograms)

8. Conclusion

Bar plots are a fundamental tool in data visualization. They effectively communicate differences among categorical groups, making them ideal for reports, dashboards, and presentations. By pairing them with proper preprocessing and customization, analysts can deliver impactful visual insights.

```python
# Bar plot - Total Beneficiaries by State
plt.figure(figsize=(14, 6))
#sns.barplot(data=state_beneficiaries, x='state_name', y='total_beneficiaries', palette='Spectral', dodge=False)
sns.barplot(data=state_beneficiaries, x='state_name', y='total_beneficiaries', hue='state_name', palette='Spectral', dodge=False, legend=False)

plt.xticks(rotation=90)
plt.title("Total Beneficiaries by State")
plt.xlabel("State")
plt.ylabel("Total Beneficiaries")
plt.tight_layout()
plt.show()
```



4.2 <u>Heat map</u>

1. Introduction

A heat map is a data visualization technique that represents the magnitude of values in a matrix or 2D grid using colour coding. Heat maps are especially useful for visualizing correlations, comparisons, or density patterns in large datasets.

2. Purpose of Using a Heat Map

- To quickly detect patterns, correlations, or anomalies in a matrix or tabular dataset.
- To visualize **numerical relationships** (e.g., correlation matrices).
- To enhance understanding of **multivariate** data.

3. Data Requirements

A heat map typically requires:

- A **2D matrix of numerical values** (e.g., correlation matrix, pivot table, confusion matrix)
- Labels for **rows and columns** (e.g., feature names, time, categories)

4. Tools and Libraries in Python

The most commonly used libraries for creating heat maps in Python are:

Library	Purpose
Seaborn	High-level interface for heat maps and statistical plots
Matplotlib	Underlying plotting engine
Pandas	Data manipulation and formatting

5. Methodology

a. Data Preparation

- Load dataset using Pandas.
- Create a 2D matrix or correlation matrix.

6. Interpretation

- Colour represent the strength of the relationship:
  - **Dark red** or **dark blue** indicates strong correlation (positive or negative).
  - **Lighter shades** indicate weaker or no correlation.
- The **annot=True** parameter overlays actual correlation values for clarity.

- Diagonal values are always 1 (perfect correlation with itself).

7. Advantages of Heat Maps

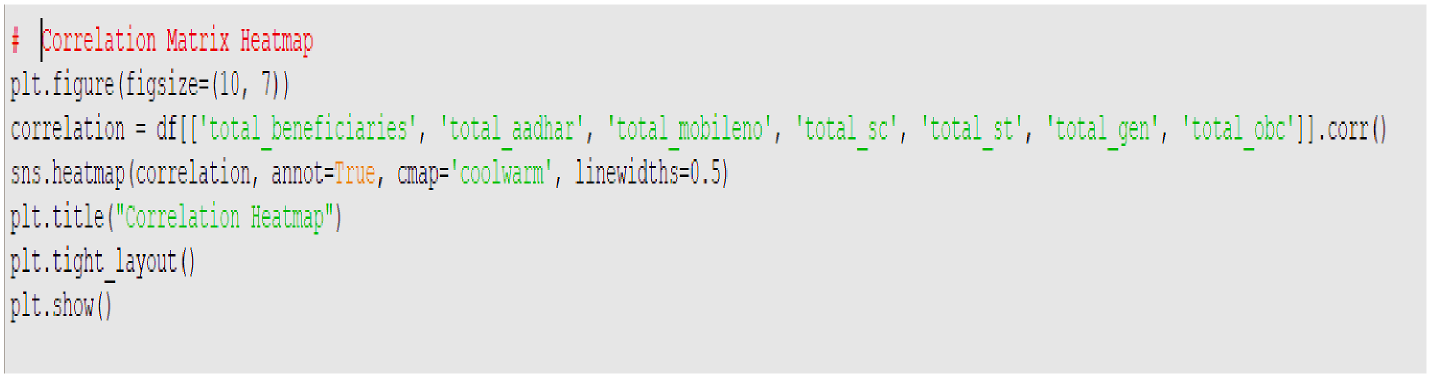
- Easy to identify **strong relationships or clusters** in large data.
- Visually appealing and compact.
- Can reveal **hidden patterns** not easily seen in tables.

8. Limitations

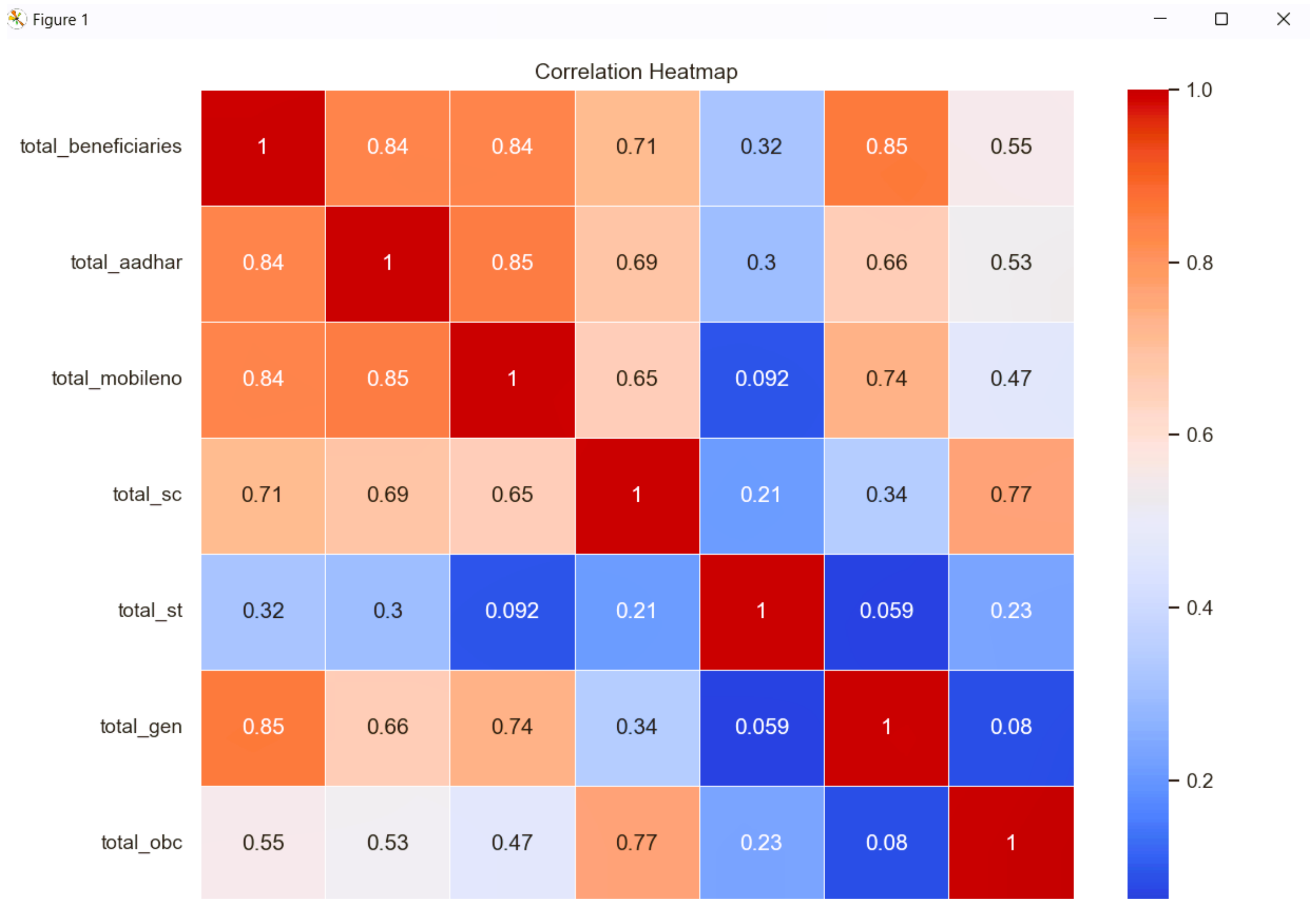
- **May become hard to read with too many rows/columns.**
- **Interpretation can be misleading without proper scaling.**
- **Colour may be ambiguous if not clearly labled or if colourblind -friendly palettes aren't used.**

9. Conclusion

Heat maps are powerful visual tools in Python for and interpreting complex data. They are especially valuable in exploratory data analysis (EDA), feature selection, and correlation analysis. When implemented properly using libraries like Seaborn, they can transform raw numbers into intuitive visual stories.



```
# Correlation Matrix Heatmap
plt.figure(figsize=(10, 7))
correlation = df[['total_beneficiaries', 'total_aadhar', 'total_mobileno', 'total_sc', 'total_st', 'total_gen', 'total_obc']].corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()
```



total_beneficiaries	1	0.84	0.84	0.71	0.32	0.85	0.55
total_aadhar	0.84	1	0.85	0.69	0.3	0.66	0.53
total_mobileno	0.84	0.85	1	0.65	0.092	0.74	0.47
total_sc	0.71	0.69	0.65	1	0.21	0.34	0.77
total_st	0.32	0.3	0.092	0.21	1	0.059	0.23
total_gen	0.85	0.66	0.74	0.34	0.059	1	0.08
total_obc	0.55	0.53	0.47	0.77	0.23	0.08	1

4.3 Scatter plot

Introduction:

A scatter plot is a type of data visualization that displays values for two numerical variables as points on a 2D plane. It's used to identify relationships, trends, clusters, and outliers in data. In Python, scatter plots are widely used during exploratory data analysis (EDA).

2. Purpose of a Scatter Plot

- ***To visualize correlation between two continuous variables.***
- ***To detect trends, linear/non-linear relationships, and outliers.***

- ***To highlight clusters or groups in data.***
- ***To evaluate the strength and direction of relationships.***

3. Data Requirements

A scatter plot requires:

- ***Two numerical variables (X and Y).***
- ***Optional categorical variable (for colouring or grouping points).***
- ***Optional size/shape/hue attributes for enhanced visualization.***

4. Tools and Libraries in Python

Library	***Use Case***
Matplotlib	***Basic and custom scatter plots***
Seaborn	***High-level, attractive scatter plots***
Ploty	***Interactive and web-based plots***
Pandas	***Quick scatter plot using .plot()***

5. Methodology

a. Data Preparation

- ***Clean and load data using Pandas.***
- ***Select two numerical columns for X and Y axes.***
-

6. Interpretation

- ***Direction: If the points trend from lower-left to upper-right, the correlation is positive. If they trend downwards, the correlation is negative.***
- ***Strength: A tighter cluster around a line suggests a stronger relationship.***
- ***Outliers: Points far from the general trend.***
- ***Clusters: Groups of similar data points may indicate categories or groupings.***

7. Advantages of Scatter Plots

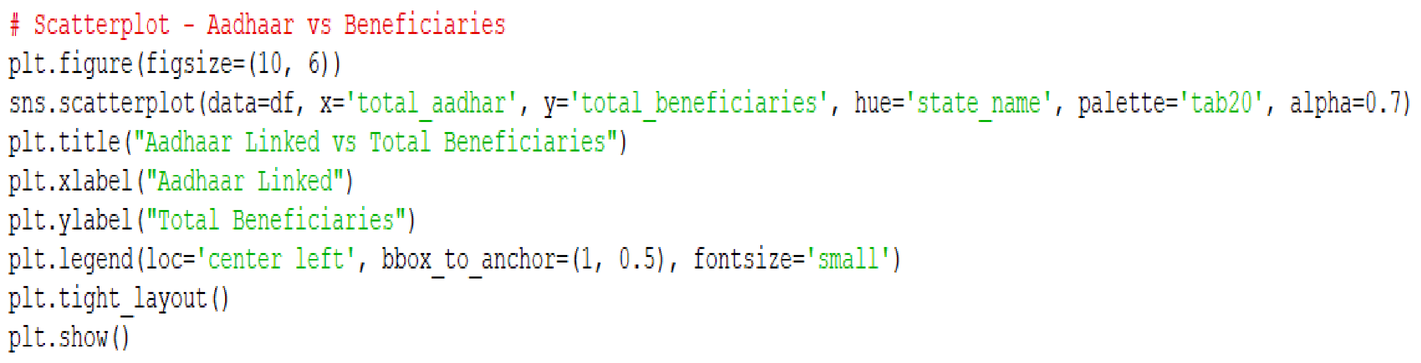
- ***Easy to detect correlation and trends.***
- ***Can highlight patterns in large datasets.***
- ***Can encode additional dimensions using colour, size, or shape.***

8. Limitations

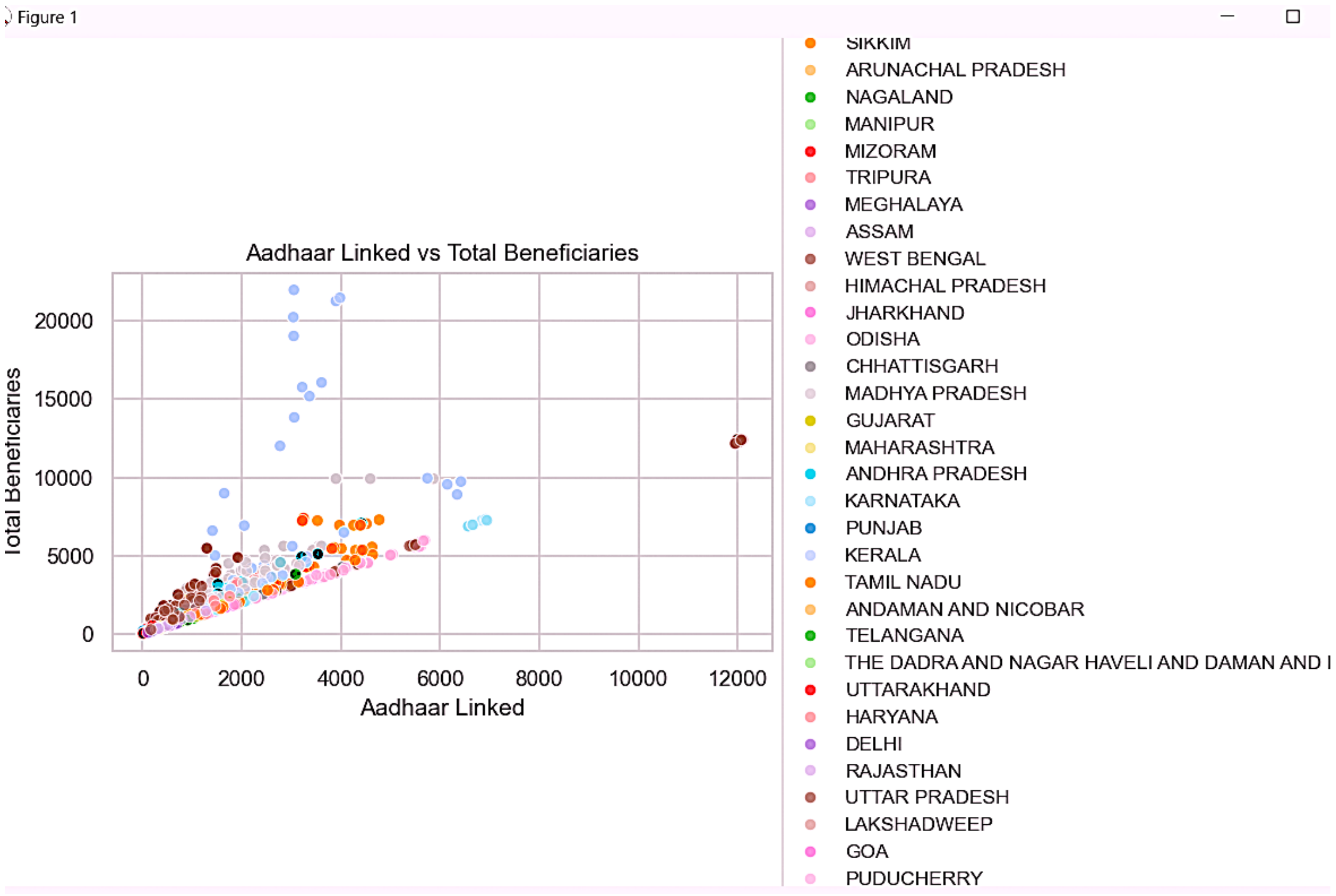
- ***Only shows relationships between two variables at a time.***
- ***Can become cluttered with too many points.***
- ***May require additional techniques to analyze non-linear relationships or multiple variables.***

9. Conclusion

Scatter plots are a core part of any data analysis workflow. They are intuitive, flexible, and effective for identifying correlations, trends, and anomalies in data. With Python libraries like Matplotlib and Seaborn, scatter plots can be easily customized and extended to show multidimensional data insights.



```
# Scatterplot - Aadhaar vs Beneficiaries
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='total_aadhar', y='total_beneficiaries', hue='state_name', palette='tab20', alpha=0.7)
plt.title("Aadhaar Linked vs Total Beneficiaries")
plt.xlabel("Aadhaar Linked")
plt.ylabel("Total Beneficiaries")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='small')
plt.tight_layout()
plt.show()
```



4.4 Line plot

1. Introduction

A line plot (or line chart) is a fundamental type of data visualization used to display data points connected by straight lines. It is primarily used to show trends over time, continuous data progression, or relationships between two numerical variables.

2. Purpose of a Line Plot

- **To show trends or patterns over time (time series).**
- **To visualize changes or progression in values.**
- **To compare multiple variables across the same axis.**
- **To detect spikes, drops, cycles, or seasonality in data.**

3. Data Requirements

To create a line plot, you typically need:

- **A continuous variable on the X-axis (often time-based).**
- **A dependent variable (Y-axis) representing the changing values.**
- **Optional: multiple lines (e.g., grouped by category).**

4. Python Libraries for Line Plots

Library	Use Case
Matplotlib	Standard for simple line plots
Seaborn	Stylish line plots with ease
Pandas	Quick .plot() method for DataFrames
Ploty	Interactive and web-based line charts

5. Methodology

a. Data Preparation

Prepare a dataset with at least two numerical columns (e.g., date and value).

b. Example: Line Plot Using Matplotlib

python

6. Interpretation

- **An upward slope indicates an increase in the Y variable.**
- **A downward slope shows a decrease.**
- **Flat lines suggest no significant change.**

- **Peaks and valleys can indicate important events, seasonality, or anomalies.**

7. Advantages of Line Plots

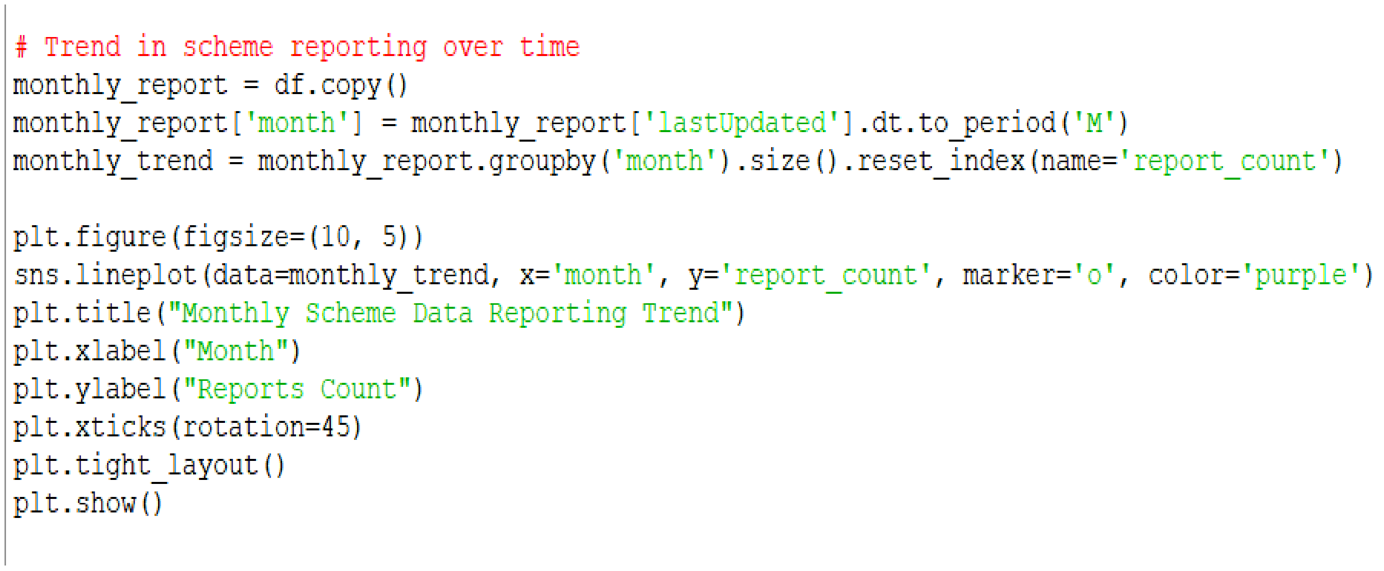
- **Excellent for visualizing trends over time.**
- **Can show multiple data series on the same graph.**
- **Good for spotting patterns, fluctuations, or cycles.**

8. Limitations

- **Less effective for categorical or discrete data.**
- **Can become cluttered with too many lines or data points.**
- **Doesn't show distribution (use histograms or boxplots for that).**

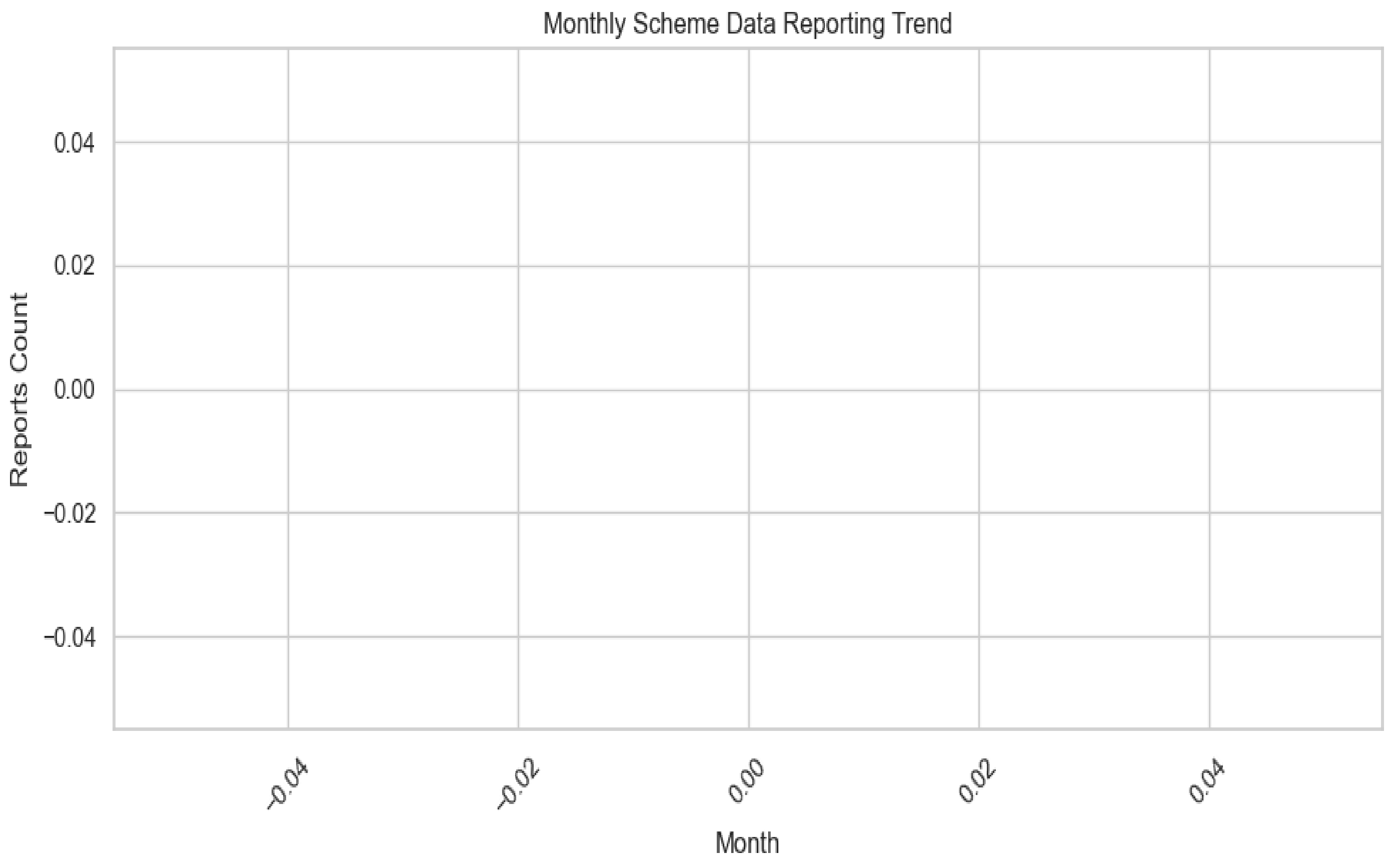
9. Conclusion

Line plots are essential for understanding how values change over time or across continuous domains. Python makes it easy to create and customize these plots using libraries like Matplotlib, Seaborn, and Pandas. They are ideal for trend analysis, forecasting, and data storytelling.



```python
# Trend in scheme reporting over time
monthly_report = df.copy()
monthly_report['month'] = monthly_report['lastUpdated'].dt.to_period('M')
monthly_trend = monthly_report.groupby('month').size().reset_index(name='report_count')

plt.figure(figsize=(10, 5))
sns.lineplot(data=monthly_trend, x='month', y='report_count', marker='o', color='purple')
plt.title("Monthly Scheme Data Reporting Trend")
plt.xlabel("Month")
plt.ylabel("Reports Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



4.5 Bar Plot Analysis

i. Introduction

Bar plots are useful for comparing numerical values across categorical groups. In this context, we visualized how different area types vary across states.

ii. General Description

Bar plots were created to compare total land features like *Net area sown*, *Net area cultivated*, *Area under current fallows*, and *Uncultivated area* across selected states.

iii. Specific Requirements, Functions and Formulas

- **Data Aggregation: Grouped by srcStateName and used .sum()**
- **Function:**

  sns.barplot(data=df, x='srcStateName', y='Net area sown')

- **Horizontal Bar Plot: Used y for categorical axis and x for numerical**

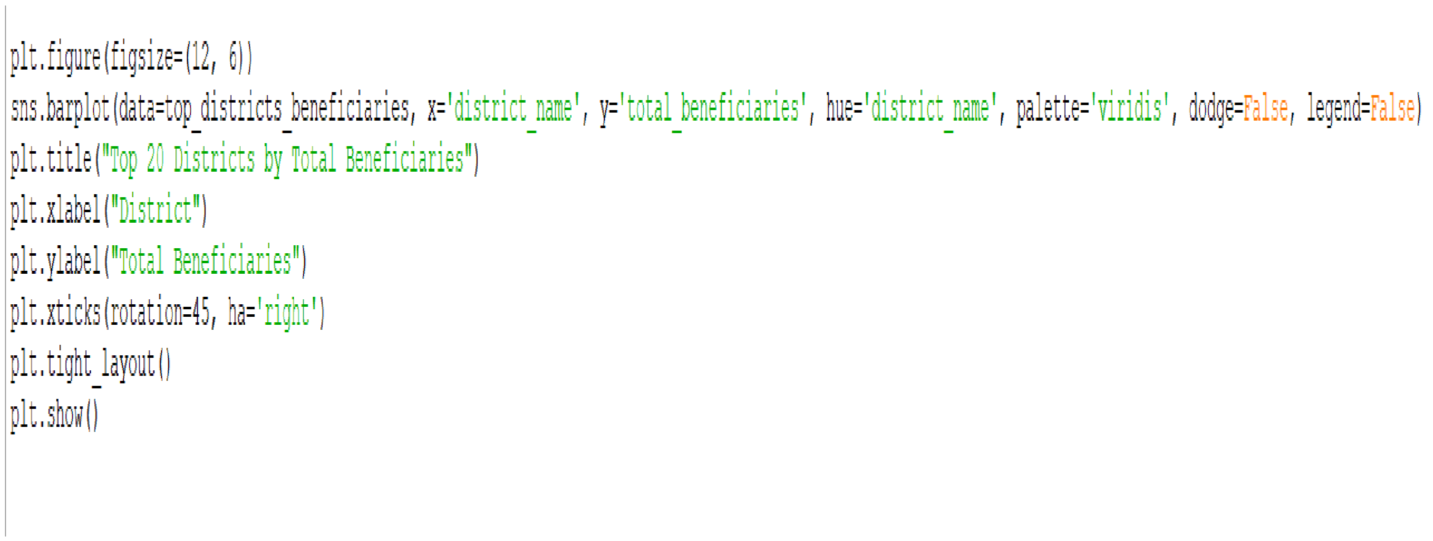
sns.barplot(data=df, y='srcStateName', x='Net area sown')

- **Subplot Layout: Created 2x2 grid of horizontal bar plots**

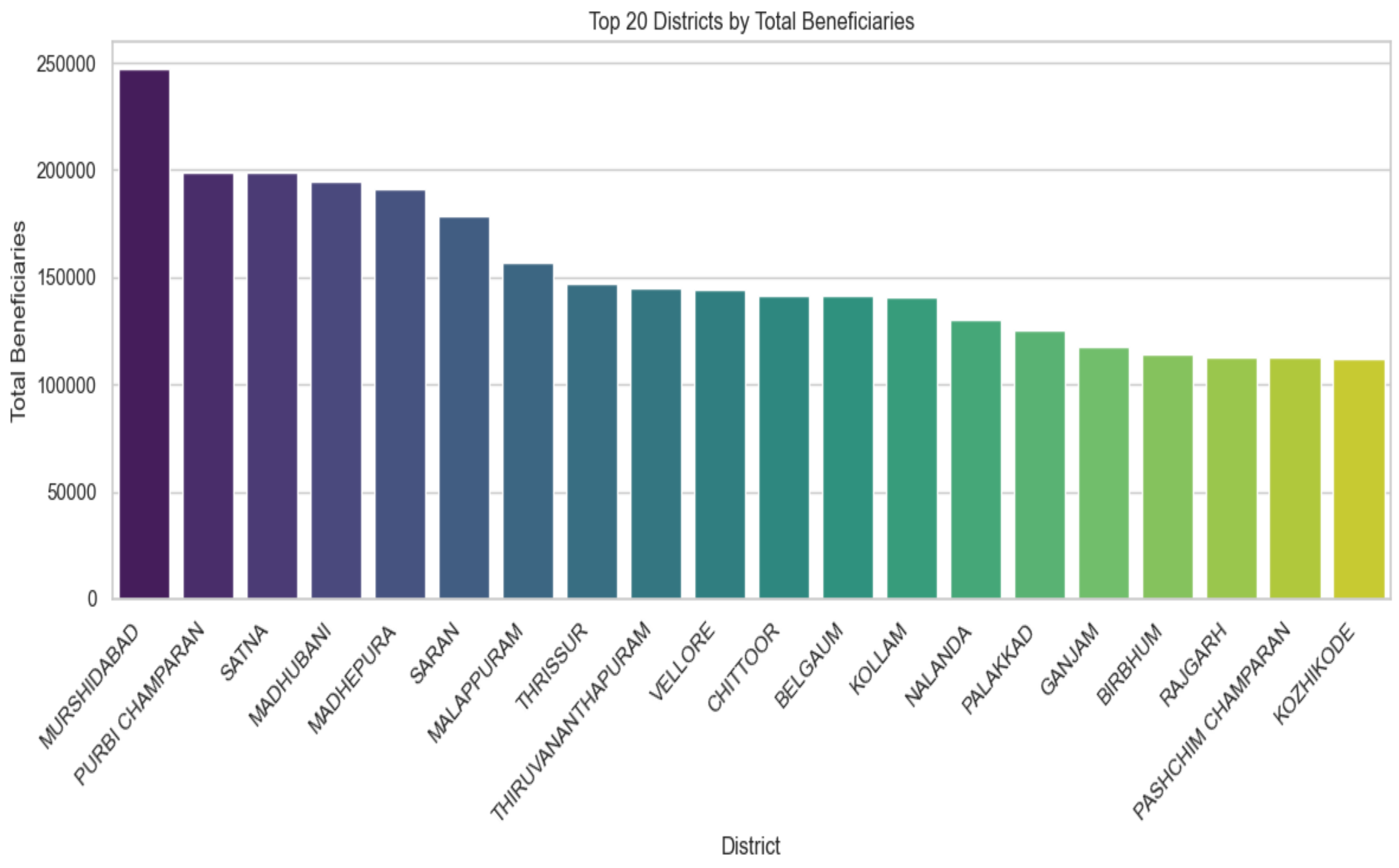
  fig, axes = plt.subplots(2, 2, figsize=(14, 10))

iv. Analysis Results

- **States like Uttar Pradesh and Maharashtra showed higher values for net area sown and cultivated.**
- **Some states had larger uncultivated areas, indicating scope for agricultural expansion.**
- **Bar plots clearly showed the distribution gaps between states.**



```
plt.figure(figsize=(12, 6))
sns.barplot(data=top_districts_beneficiaries, x='district_name', y='total_beneficiaries', hue='district_name', palette='viridis', dodge=False, legend=False)
plt.title("Top 20 Districts by Total Beneficiaries")
plt.xlabel("District")
plt.ylabel("Total Beneficiaries")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



4.6 box plot

1. Introduction

A box plot (also known as a box-and-whisker plot) is a powerful statistical chart used to visualize the distribution, central tendency, and spread of a dataset. It provides a clear summary of five-number statistics: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum — and helps to detect outliers.

2. Purpose of a Box Plot

- **To summarize the spread and skewness of numerical data.**
- **To compare distributions across groups or categories.**
- **To identify outliers and variability in datasets.**

3. Key Components

Element	Meaning
Box	Interquartile Range (IQR = Q3 - Q1)
Line inside the box	Median (Q2)
Whiskers	Extend to the smallest and largest non-outlier values
Outliers	Individual points beyond 1.5 × IQR from Q1 or Q3

5. Interpretation

- Median Line: Shows the middle value of the data.
- Box Size (IQR): Indicates data spread; wider box = more variability.
- Whiskers: Represent the range of non-outlier values.
- Outliers: Marked as dots, may suggest unusual behavior or errors.
- Group Comparison: When using categorical X-axes, it's easy to compare data distributions (e.g., sales by region).

6. Advanced Uses

- Grouped box plots: Compare distributions between multiple categories.
- Horizontal orientation: Useful for long category names.
- Colour coding (hue): Distinguish subgroups within categories.

7. Advantages

- Excellent for spotting outliers.
- Compact visualization of distribution and spread.
- Easily compares multiple datasets side by side.

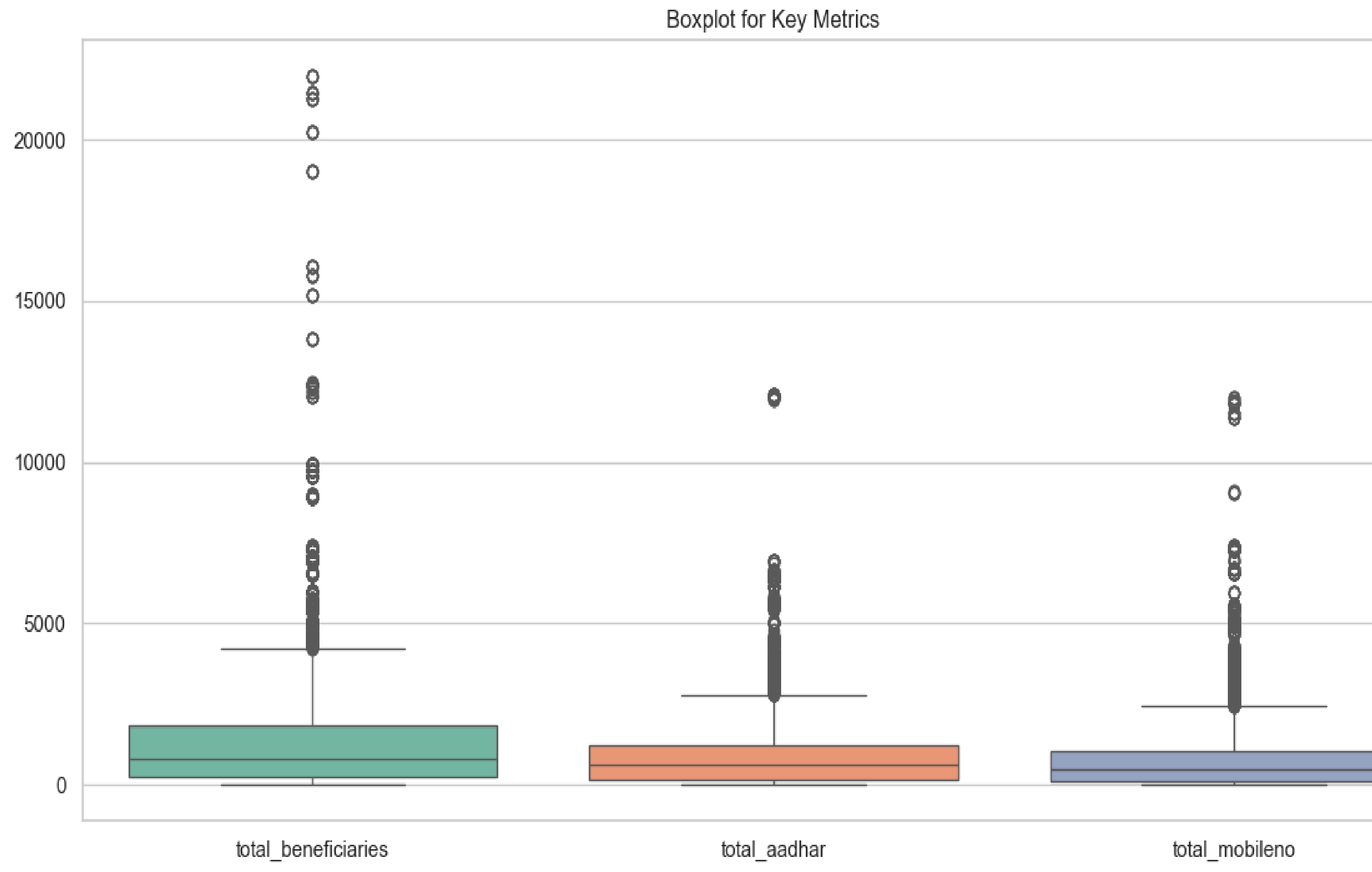
8. Limitations

- Doesn't show distribution shape (e.g., bimodality) as clearly as histograms.
- Might be less intuitive for non-statistical audiences.
- Less effective for small datasets.

9. Conclusion

Box plots are robust and efficient tools for statistical data visualization. They give a quick overview of the data's spread, central value, and outliers. In Python, libraries like Seaborn and Matplotlib make it easy to create insightful box plots with just a few lines of code.

```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['total_beneficiaries', 'total_aadhar', 'total_mobileno']], palette='Set2')
plt.title("Boxplot for Key Metrics")
plt.tight_layout()
#plt.savefig("key_metrics_boxplot.pdf")
plt.show()
```


Boxplot for Key Metrics
total_beneficiaries
total_aadhar
total_mobileno