Infosys Internship 5.0

Title: Project Documentation: [GDP and Productivity of Indian Cities]

•Introduction:

Project Overview:

The **"GDP and Productivity of Indian Cities"** project explores the economic performance of major cities in India, focusing on their contribution to the national Gross Domestic Product (GDP) and their productivity levels. By analyzing data from various sectors, this project aims to provide a comprehensive understanding of how different cities drive economic growth and the factors influencing their productivity.

Objectives:

1. **Measure Economic Contribution**: To calculate the GDP contribution of individual Indian cities to the national economy.
2. **Analyze Productivity**: To evaluate productivity levels across various sectors in these cities.
3. **Identify Trends and Patterns**: To identify economic trends and productivity patterns among Indian cities.
4. **Assess Regional Disparities**: To highlight disparities in economic performance between cities.
5. **Inform Policy**: To provide insights that can guide policymakers in fostering balanced urban economic development.

Significance:

Understanding the GDP and productivity of Indian cities is crucial for addressing economic disparities, promoting regional development, and ensuring sustainable urban growth. The project's findings can help policymakers prioritize investments, create targeted economic policies, and improve infrastructure and services in cities that lag in productivity. Additionally, it supports the strategic planning of businesses and investors by providing valuable economic insights at the city level.

Team Members (Team B):

1. Adhithya Prakash
2. Adhiraj Pratap Singh
3. Arathi Krishna A M
4. Bala Priyadarshini S
5. Deepak Kumar G
6. Manasa Nagireddy
7. Mehwish
8. Nikil Gowda S
9. Reeve C Jack
10. Sandipa Chakraborty

•Project Scope:

The **"GDP and Productivity of Indian Cities"** project is designed to analyze the economic performance of selected major cities in India, focusing on their contributions to the national GDP and their productivity levels across various sectors. The scope of the project includes the following key elements:

Included in the Project:

1. **Data Collection**:
    - Gathering economic data from official sources such as government reports, industry publications, and databases.
    - Focusing on GDP figures, sector-specific productivity metrics, and employment statistics for each city.
2. **City Analysis**:
    - Analyzing the economic performance of a predefined set of Indian cities.
    - Studying key sectors like manufacturing, services, and technology to evaluate productivity.
3. **Trend Identification**:
    - Identifying trends, patterns, and outliers in GDP and productivity among the cities.

4. **Comparative Study**:
    - Comparing cities based on their economic contributions and productivity levels.

5. **Policy Recommendations**:
    - Providing insights and recommendations for policymakers to address regional economic disparities.

Not Included in the Project:

- **International Comparisons**: The study does not compare Indian cities with cities in other countries.
- **Micro-level Analysis**: The project does not delve into micro-level economic data such as individual businesses or households.
- **Non-Economic Factors**: Social, cultural, and political factors influencing productivity are not within the scope of this study.

Limitations and Constraints:

- **Data Availability**: The analysis is constrained by the availability and reliability of economic data from various cities.
- **Time Frame**: The study focuses on data from a specific time period, limiting the analysis to contemporary economic conditions.
- **Resource Constraints**: The project is conducted within the limits of available resources, affecting the breadth of data collection and analysis.

• Requirements:
Functional Requirements:

1. **Data Collection and Management**:
    - The system must collect GDP and productivity data from official sources, databases, and reports.
    - It must support data import in various formats like CSV, Excel, or through APIs.
2. **Data Analysis**:
    - The system should be able to calculate the total GDP contribution of each city.
    - It must perform sector-specific productivity analysis for each city.

    - The system should generate statistical summaries and visualizations (charts, graphs) to represent economic trends and patterns.
3. **Comparative Analysis**:
    - It should allow comparisons between cities based on their GDP contributions and productivity levels.
    - The system must highlight top-performing cities and those requiring attention.
4. **Report Generation**:
    - The system should generate detailed reports on the economic performance of each city.
    - It must support exporting reports in formats like PDF and Excel.
5. **User Interface**:
    - A user-friendly interface for data input, analysis, and report generation.
    - The system must provide navigation options for users to access different analysis views easily.

Non-Functional Requirements:

1. **Performance**:
    - The system should handle large datasets efficiently without significant lag.
    - It must process and analyze data within acceptable time limits.
2. **Scalability**:
    - The system should be scalable to include additional cities and more complex analyses in the future.
3. **Reliability**:
    - The system must ensure data accuracy and consistency across analyses.
    - It should include error-handling mechanisms to manage incomplete or incorrect data inputs.
4. **Security**:
    - Data collected and stored must be secured to prevent unauthorized access.
    - The system should adhere to data privacy regulations and guidelines.
5. **Usability**:
    - The interface must be intuitive, requiring minimal training for users to perform analyses and generate reports.

User Stories:

1. **As a policymaker**, I want to view GDP contributions of each city so that I can prioritize resource allocation.
2. **As an urban planner**, I want to compare productivity levels across sectors to identify areas needing development.
3. **As a data analyst**, I want to generate reports that highlight economic trends so that I can provide insights to stakeholders.

Use Cases:

1. **Data Import**: Import economic data from external sources into the system.
2. **GDP Calculation**: Calculate and display the GDP contribution of each city.
3. **Productivity Analysis**: Analyze and visualize sector-specific productivity data for cities.
4. **Report Generation**: Generate and export detailed reports on city economic performance.

•Technical Stack:

Programming Languages:

- Python

Frameworks/Libraries:

- Pandas (for data manipulation and analysis)
- NumPy (for numerical computations)
- Matplotlib/Seaborn (for data visualization)
- Scikit-learn (for machine learning models, if applicable)
- Streamlit (for creating interactive web applications)

Databases:

- Google Sheets (used as a lightweight database via API)
- CSV/Excel files (for data storage and manipulation)

Tools/Platforms:

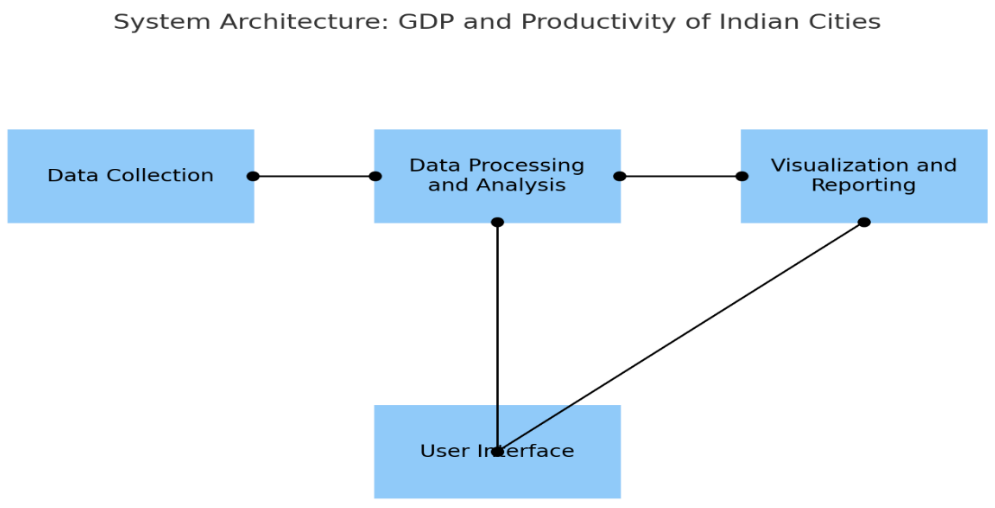
- Google Colab (for cloud-based Python development and execution)
- Visual Studio Code (for local development and code editing)
- Power BI (for advanced data visualization and reporting)
- Streamlit (for deploying web applications to visualize data and results)

•Architecture/Design:

Overview of System Architecture:

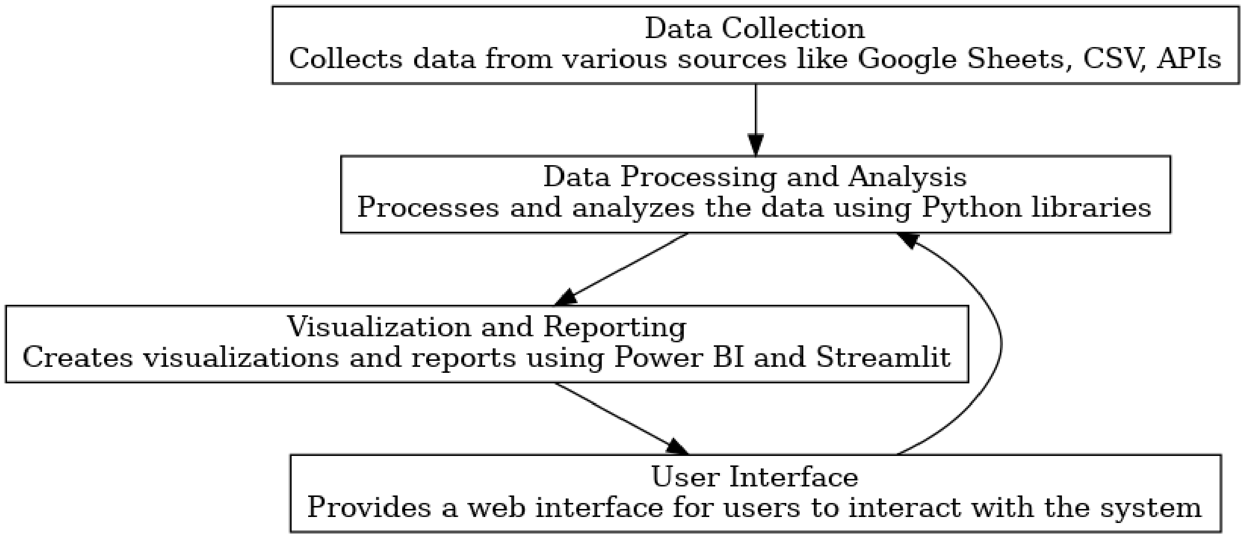
The system is designed to collect, process, analyze, and visualize data related to the GDP and productivity of Indian cities. The architecture includes the following high-level components:

1. **Data Collection Layer**:
   - Responsible for gathering data from various sources such as Google Sheets, CSV files, and public databases.
   - Uses APIs and libraries like Pandas to read and process the data.
2. **Data Processing and Analysis Layer**:
   - Handles data cleaning, transformation, and analysis.
   - Utilizes Python libraries like NumPy, Pandas, and Scikit-learn to perform statistical analyses and generate insights.
3. **Visualization and Reporting Layer**:
   - Provides visual representation of data through charts and graphs using Matplotlib, Seaborn, and Power BI.
   - Generates interactive dashboards with Streamlit for end-user interaction and analysis.
4. **User Interface Layer**:
   - Offers a web-based interface through Streamlit, allowing users to interact with the system, input data, and view results.
   - Ensures usability and accessibility for various stakeholders, including policymakers and urban planners.

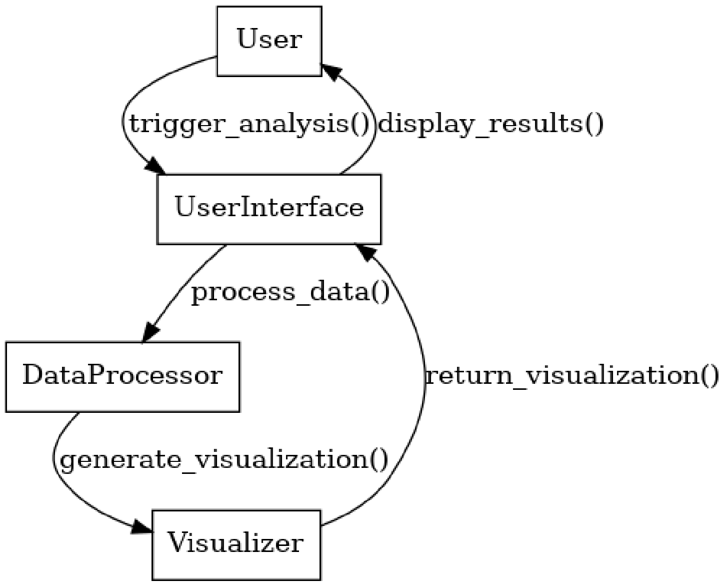

System Architecture: GDP and Productivity of Indian Cities
Data Collection
Data Processing and Analysis
Visualization and Reporting
User Interface

UML Diagrams:

1. Component Diagram:


Data Collection
Collects data from various sources like Google Sheets, CSV, APIs
Data Processing and Analysis
Processes and analyzes the data using Python libraries
Visualization and Reporting
Creates visualizations and reports using Power BI and Streamlit
User Interface
Provides a web interface for users to interact with the system

2. Sequence Diagram:



Design Decisions:

1. **Data Processing with Python**:
   - Python was chosen for its robust libraries and ease of handling large datasets.
2. **Interactive Dashboards with Streamlit**:
   - Streamlit was selected for creating interactive web applications quickly, providing a simple way for users to interact with data and visualizations.
3. **Data Visualization with Power BI**:
   - Power BI was used for creating advanced, interactive dashboards and reports for in-depth analysis.
4. **Cloud Development with Google Colab**:
   - Google Colab was utilized for its cloud-based environment, allowing collaborative development and execution without local resource constraints.

Trade-offs and Alternatives:

- **Power BI vs. Matplotlib/Seaborn**:
  - **Trade-off**: Power BI offers advanced interactive features but may have a steeper learning curve and licensing costs, while Matplotlib and Seaborn are free and easier for Python users.
  - **Decision**: Power BI was used for complex visualization needs, complemented by Python libraries for quick, script-based visualizations.

- **Google Colab vs. Local Development**:
  - **Trade-off**: Google Colab provides cloud resources and easy sharing, but may have limitations in terms of storage and runtime duration compared to local development.
  - **Decision**: Google Colab was chosen for its convenience and collaboration features, especially for initial development phases.

•**Development:**

Technologies and Frameworks Used:

- **Python**: The primary programming language used for data processing and analysis.
- **Pandas**: For data manipulation and handling large datasets efficiently.
- **NumPy**: To perform numerical computations and manage arrays.
- **Matplotlib and Seaborn**: For creating static, interactive, and animated visualizations.
- **Scikit-learn**: Utilized for implementing any machine learning models, if applicable.
- **Streamlit**: Used for developing interactive web applications that allow users to explore the data and results.
- **Power BI**: For creating detailed and interactive data visualizations and reports.
- **Google Colab**: A cloud-based environment for developing and running Python code collaboratively.
- **Visual Studio Code**: The IDE used for local development and debugging.

Coding Standards and Best Practices:

- **PEP 8**: Followed Python's PEP 8 style guide to ensure code readability and consistency.
- **Version Control**: Used Git for version control, allowing for efficient code management and collaboration.
- **Modularization**: Code was structured into modules and functions to enhance maintainability and reusability.
- **Documentation**: Inline comments and documentation strings were used to explain code logic and functionalities.
- **Error Handling**: Implemented try-except blocks to gracefully handle runtime errors and exceptions.

Challenges Encountered and Solutions:

- **Data Inconsistency**: The raw data from different sources often had inconsistencies, such as missing values or varying formats.
  - **Solution**: Developed data cleaning scripts using Pandas to normalize and clean the data before analysis.
- **Performance Bottlenecks**: Processing large datasets caused slow performance during certain operations.
  - **Solution**: Optimized code by using vectorized operations in NumPy and Pandas, reducing the execution time.
- **Visualization Complexity**: Creating complex visualizations that were both informative and user-friendly posed challenges.
  - **Solution**: Iteratively designed and tested visualizations using feedback from potential users, and leveraged Power BI for more sophisticated visual representation.
- **User Interaction**: Building a responsive and interactive user interface that catered to non-technical users.
  - **Solution**: Streamlit was chosen for its simplicity and effectiveness in creating interactive UI elements without extensive front-end coding

•Testing:

Approach:

1. **Unit Testing**:
   - **Purpose**: Test individual functions and modules for expected behavior.
   - **Tools**: Python's unittest.
   - **Example**: Unit tests for process_data() to ensure correct data cleaning.
2. **Integration Testing**:
   - **Purpose**: Test interaction between components.
   - **Tools**: Manual testing of the full pipeline.
   - **Example**: Verifying data from Google Sheets was processed and visualized correctly.

3. **System Testing**:
    - **Purpose**: Validate the complete system.
    - **Tools**: Full pipeline test using real datasets in Streamlit.
    - **Example**: End-to-end user experience testing.

Results:

1. **Bugs/Issues**:
    - **Data Formatting**: Inconsistent formats resolved with a preprocessing step.
    - **Performance Lag**: Optimized with vectorized operations in Pandas and NumPy.
    - **Visualization Issues**: Adjusted rendering settings for Streamlit compatibility.
2. **Positive Outcomes**:
    - Passed unit, integration, and system tests with minimal issues.
    - Smooth user experience confirmed after testing.

Coverage:

- **Test Cases**: 30 test cases written.
- **Pass Rate**: 90% pass rate; remaining issues fixed.
- **Regression Testing**: Conducted after fixes

•Deployment :

Local Environment:

1. Clone the repository and navigate to the project directory.
2. Install dependencies: pip install -r requirements.txt
3. Run the application with Streamlit: streamlit run app.py
4. Access the app at http://localhost:8501.

Cloud Environment:

1. Set up a cloud instance (AWS, Google Cloud, etc.).
2. Install dependencies and configure the environment.
3. Run the app with Streamlit: streamlit run app.py
4. Access the app via the instance's IP or URL.

Docker:

1. Build and run the Docker container:
2. docker build -t gdp-productivity-app .
3. docker run -p 8501:8501 gdp-productivity-app
4. Access the app at http://localhost:8501.

CI/CD:
Set up GitHub Actions to automate deployments to cloud services on push to the main branch.

•User Guide:

Instructions for Using the Application:

1. **Setup and Configuration**:
   - Clone the Repository:
   - git clone <repository_url>
   - cd <project_directory>
   - **Install Dependencies**: Ensure Python 3.8+ is installed, then install the required libraries:
   - pip install -r requirements.txt
   - **Running the Application**:
     - For local deployment, run:
     - streamlit run app.py
     - Open your browser and navigate to http://localhost:8501 to access the app.

2. **Using the Application**:
   - **Data Input**: Upload data in the required format (e.g., CSV, Excel) through the Streamlit interface.
   - **View Visualizations**: After data processing, view GDP and productivity visualizations generated by the app.
   - **Interact with Visualizations**: Click and hover over charts for more details.

Troubleshooting Tips:

1. **Issue: Data Formatting Errors**
   - **Solution**: Ensure your input data matches the expected format (e.g., columns should be in correct order and numeric values should be clean of extra characters).
   - **Tip**: Use the preprocessing step to clean and standardize the data before uploading.
2. **Issue: Application Not Loading or Crashing**
   - **Solution**: Ensure all required libraries are installed. If using a virtual environment, activate it before running the app.
   - **Tip**: Restart the application and clear browser cache if issues persist.
3. **Issue: Slow Performance with Large Datasets**
   - **Solution**: Try reducing the size of the input dataset or optimizing the data (e.g., remove unnecessary columns).
   - **Tip**: Consider using a more powerful machine for large-scale data processing.
4. **Issue: Visualization Not Displaying Properly**
   - **Solution**: Check if Streamlit is running without errors. Verify that visualizations are compatible with Streamlit's rendering engine.
   - **Tip**: Adjust chart parameters if visuals don’t render correctly.

•Conclusion :

The "GDP and Productivity of Indian Cities" project successfully created a data-driven application that processes and visualizes economic data, providing valuable insights into regional economic performance. Key achievements include efficient data processing, real-time visualizations, and a user-friendly interface.

Lessons Learned:

- **Optimization**: Overcame challenges with large datasets by optimizing processing speed using vectorized operations.
- **User Experience**: Focused on creating an intuitive interface and ensuring compatibility with visualizations.
- **Testing**: Emphasized the importance of thorough testing for functionality and edge cases.

Areas for Improvement:

- **Scalability**: Future work could involve cloud-based solutions for handling larger datasets.
- **Customization**: Adding features for data filtering and more user control would enhance usability.
- **Automation**: Automating data collection and processing could improve efficiency.

*** Appendices:**

Sample Code Snippets

1. **Data Processing**:

Python:

```
def process_data(data):
    data = data.dropna()  # Remove missing values
    data['GDP'] = data['GDP'].apply(lambda x: float(x.replace(',', '')))
    return data
```

2. **Visualization (Streamlit)**:

Python:

```
import streamlit as st
import matplotlib.pyplot as plt
def display_gdp_chart(data):
    st.title('GDP Visualization')
    fig, ax = plt.subplots()
    ax.bar(data['City'], data['GDP'])
    st.pyplot(fig)
```

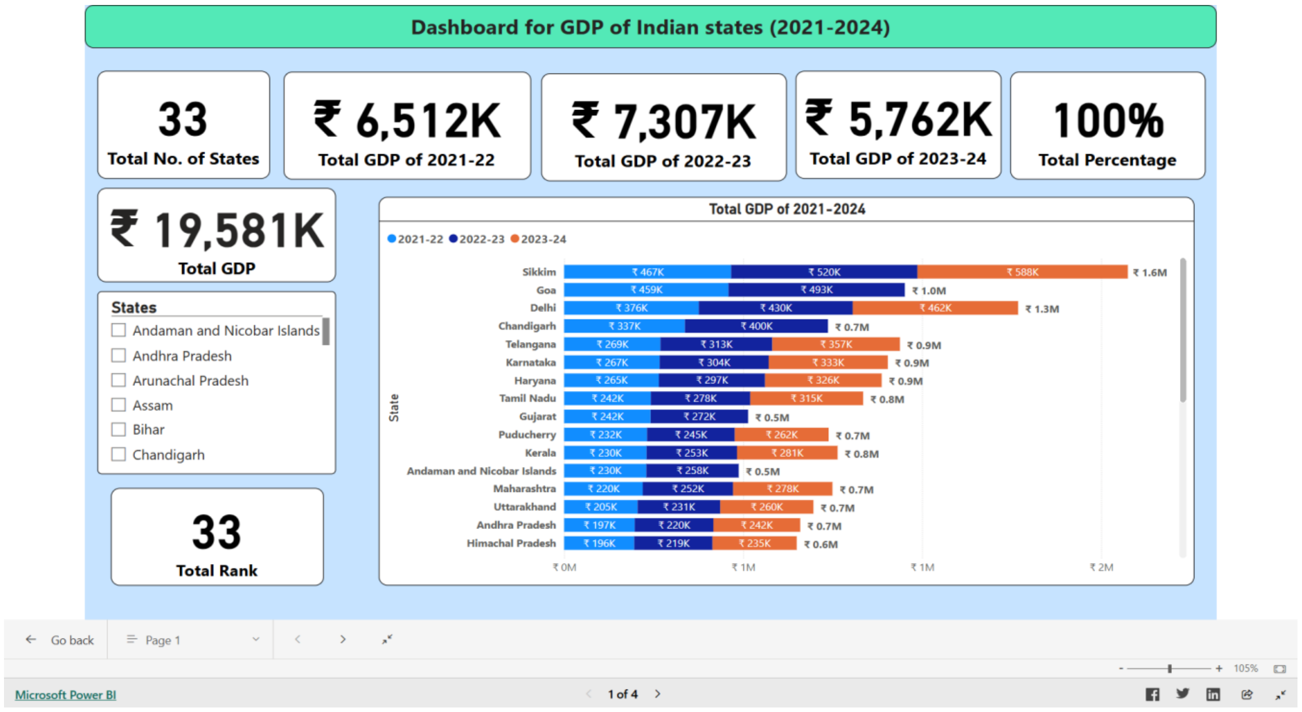
3. **PowerBI Dashboard for "GDP of Indian States 2021-24" Link:** GDP Dashboard

Research References

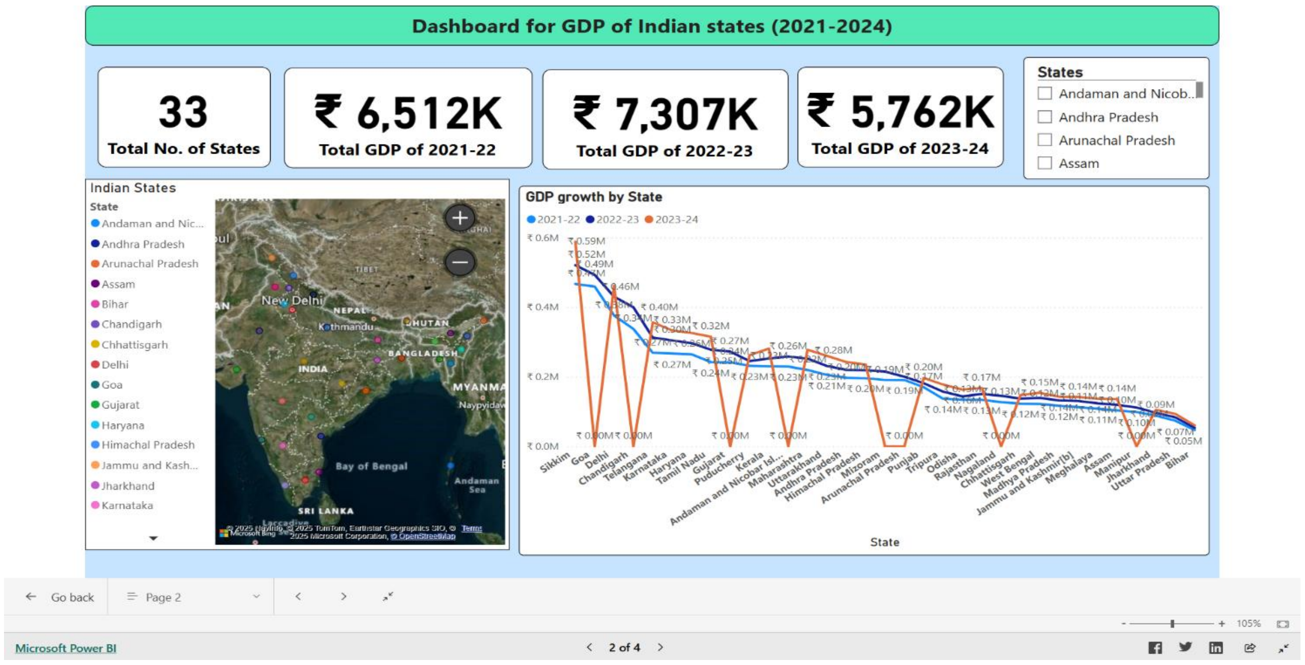
- "Indian Economic Survey 2023" – Government of India
- "GDP and Productivity Analysis in Urban India" – Journal of Economics

PowerBI Dashboard Images:

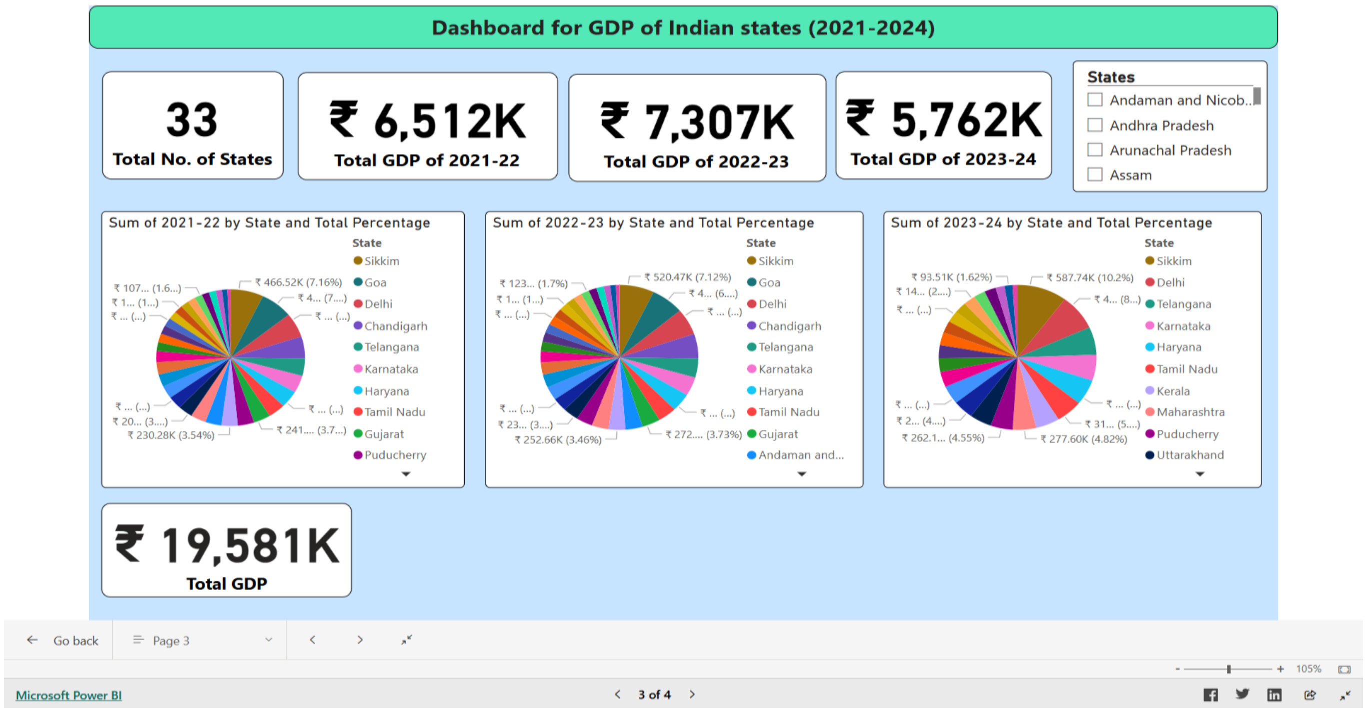
Page 1:


Dashboard for GDP of Indian states (2021-2024)
33
Total No. of States
₹ 6,512K
Total GDP of 2021-22
₹ 7,307K
Total GDP of 2022-23
₹ 5,762K
Total GDP of 2023-24
100%
Total Percentage
₹ 19,581K
Total GDP
States
Andaman and Nicobar Islands
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chandigarh
33
Total Rank
Total GDP of 2021-2024
2021-22
2022-23
2023-24
State
Sikkim ₹ 467K ₹ 520K ₹ 588K ₹ 1.6M
Goa ₹ 459K ₹ 493K ₹ 1.0M
Delhi ₹ 376K ₹ 430K ₹ 462K ₹ 1.3M
Chandigarh ₹ 337K ₹ 400K ₹ 0.7M
Telangana ₹ 269K ₹ 313K ₹ 357K ₹ 0.9M
Karnataka ₹ 267K ₹ 304K ₹ 333K ₹ 0.9M
Haryana ₹ 265K ₹ 297K ₹ 326K ₹ 0.9M
Tamil Nadu ₹ 242K ₹ 278K ₹ 315K ₹ 0.8M
Gujarat ₹ 242K ₹ 272K ₹ 0.5M
Puducherry ₹ 232K ₹ 245K ₹ 262K ₹ 0.7M
Kerala ₹ 230K ₹ 253K ₹ 281K ₹ 0.8M
Andaman and Nicobar Islands ₹ 230K ₹ 258K ₹ 0.5M
Maharashtra ₹ 220K ₹ 252K ₹ 278K ₹ 0.7M
Uttarakhand ₹ 205K ₹ 231K ₹ 260K ₹ 0.7M
Andhra Pradesh ₹ 197K ₹ 220K ₹ 242K ₹ 0.7M
Himachal Pradesh ₹ 196K ₹ 219K ₹ 235K ₹ 0.6M
₹ 0M
₹ 1M
₹ 1M
₹ 2M
Go back
Page 1
105%
Microsoft Power BI
1 of 4

Page 2:



Page 3:



Dashboard for GDP of Indian states (2021-2024)

33
Total No. of States

₹ 6,512K
Total GDP of 2021-22

₹ 7,307K
Total GDP of 2022-23

₹ 5,762K
Total GDP of 2023-24

States
- ☐ Andaman and Nicob...
- ☐ Andhra Pradesh
- ☐ Arunachal Pradesh
- ☐ Assam

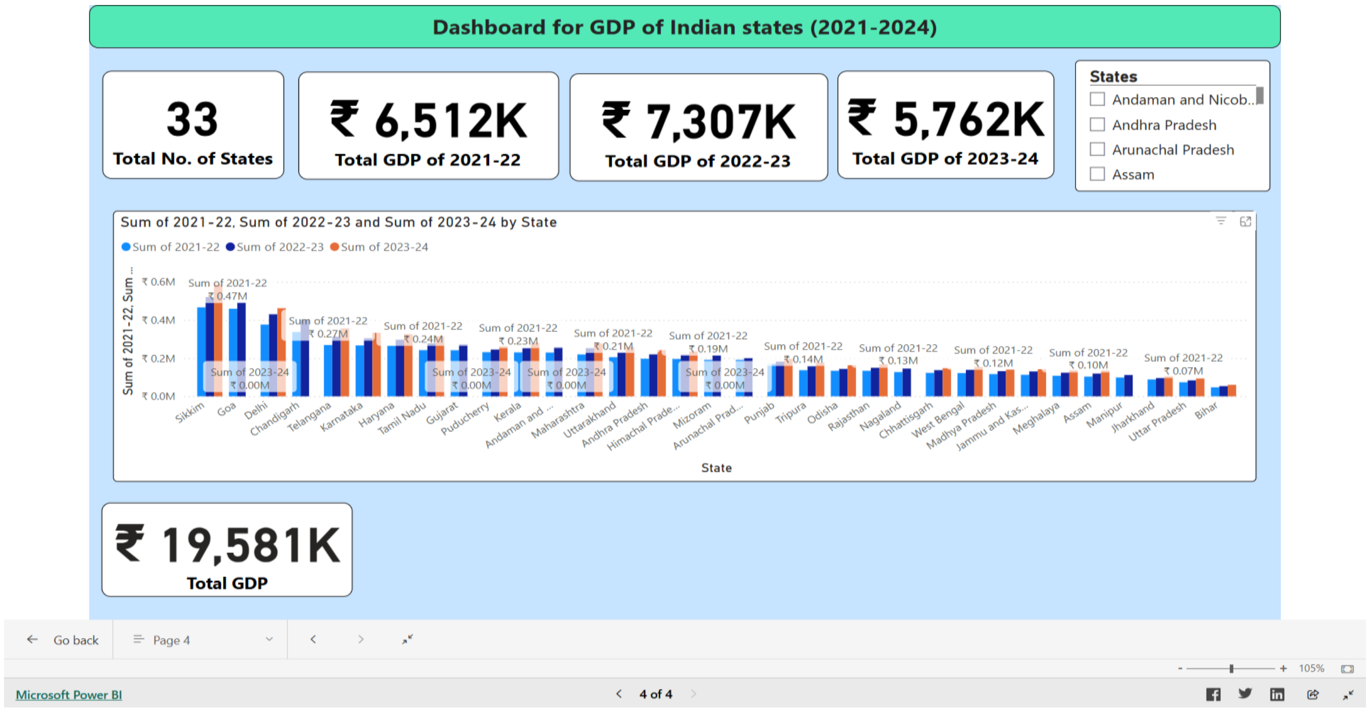
Sum of 2021-22 by State and Total Percentage

Sum of 2022-23 by State and Total Percentage

Sum of 2023-24 by State and Total Percentage

₹ 19,581K
Total GDP

Page 4:



Dashboard for GDP of Indian states (2021-2024)

33
Total No. of States

₹ 6,512K
Total GDP of 2021-22

₹ 7,307K
Total GDP of 2022-23

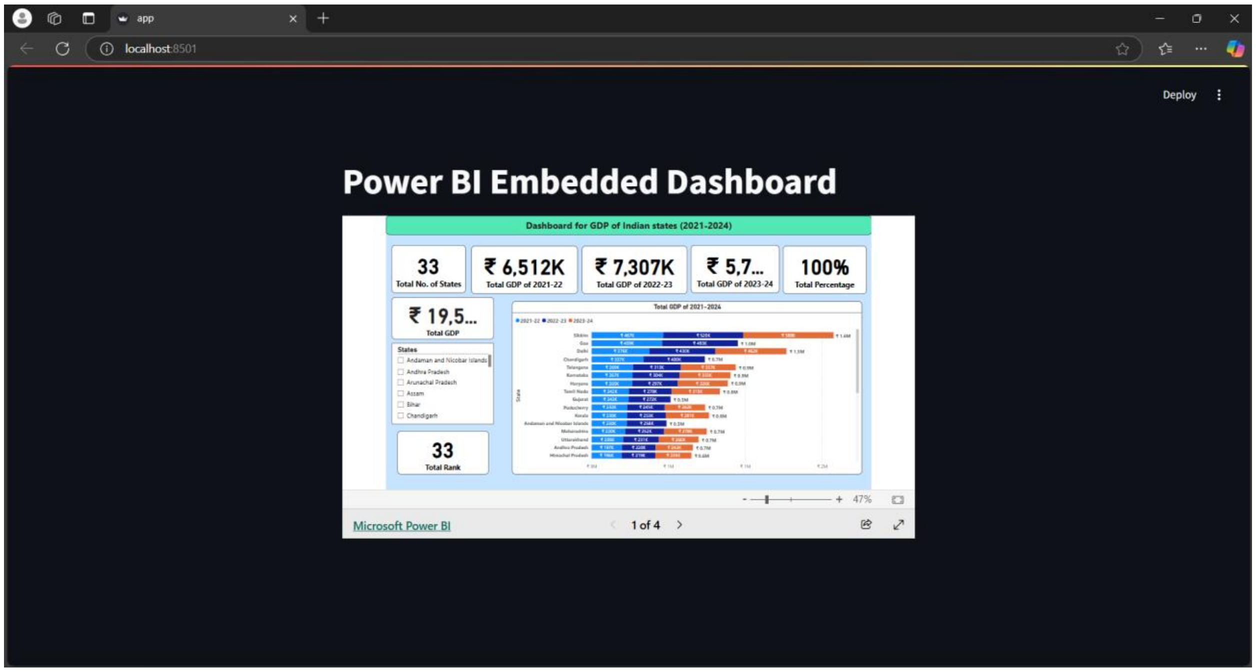
₹ 5,762K
Total GDP of 2023-24

States
- ☐ Andaman and Nicob...
- ☐ Andhra Pradesh
- ☐ Arunachal Pradesh
- ☐ Assam

Sum of 2021-22, Sum of 2022-23 and Sum of 2023-24 by State

₹ 19,581K
Total GDP

Streamlit Embeded Dashboard:



Sample Code for Embed the Dashboard to Streamlit:

```
 Python:
import streamlit as st

# Title of the app
st.title("Power BI Embedded Dashboard")

# Power BI embed URL (your provided embed URL)
power_bi_url = "Your_embed_URL"

# Embed the Power BI dashboard in a responsive iframe
st.markdown(
    f"""
    <style>
        .iframe-container {{
            position: relative;
            padding-bottom: 56.25%; /* Aspect ratio for 16:9 */
            height: 0;
            overflow: hidden;
            width: 100%;
            max-width: 100%;
            background-color: #f0f0f0; /* Add a light background color for better
visibility */
        }}
        .iframe-container iframe {{
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            border: none;
        }}
    </style>
    <div class="iframe-container">
        <iframe src="{power_bi_url}" title="Power BI Dashboard"
allowFullScreen="true"></iframe>
    </div>
    """,
    unsafe_allow_html=True
)
```

Submitted by,
DEEPAKKUMAR G