PRE-PROCESSING OF THE DATA SET

```
import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Combined Data_2008-2019 - Sheet1.csv')
```

```
/tmp/ipython-input-1084626159.py:3: DtypeWarning: Columns (82) have mixed types. Specify dtype option on import or s
  df = pd.read_csv('/content/drive/MyDrive/Combined Data_2008-2019 - Sheet1.csv')
```

```
print(f"Initial shape: {df.shape}")
display(df.head())
```

```
Initial shape: (8285, 200)
```

	Country	State	District	Year	Women Registered For Ante Natal Care (Anc) (UOM:Number), Scaling Factor:1	Women Registered For Ante Natal Care (Anc) Within First Trimester (UOM:Number), Scaling Factor:1	Women Registered Under Janani Suraksha Yojana (Jsy) (UOM:Number), Scaling Factor:1	Registrations For Ante Natal Care (Anc) In The First Trimester (To Total Anc Registrations) (%) (UOM:% (Percentage)), Scaling Factor:1	Janani Suraksha Yojana (Jsy) Registrations(To Total Anc Registrations) (%) (UOM:% (Percentage)), Scaling Factor:1
0	India	Andaman And Nicobar Islands	Nicobars	Financial Year (Apr - Mar), 2008	NaN	NaN	NaN	NaN	NaN
1	India	Andaman And Nicobar Islands	North And Middle Andaman	Financial Year (Apr - Mar), 2008	NaN	NaN	NaN	NaN	NaN
2	India	Andaman And Nicobar Islands	South Andamans	Financial Year (Apr - Mar), 2008	7199.0	5119.0	NaN	71.1	0.0
3	India	Andaman And Nicobar Islands	Nicobars	Financial Year (Apr - Mar), 2009	925.0	247.0	329.0	26.7	35.6
4	India	Andaman And Nicobar Islands	North And Middle Andaman	Financial Year (Apr - Mar), 2009	2253.0	739.0	45.0	32.8	2.0

5 rows × 200 columns

```
# Step 1: Show basic info
print("\n--- Data Info ---")
df.info()
```

```
--- Data Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8285 entries, 0 to 8284
Columns: 200 entries, Country to Unnamed: 199
dtypes: float64(195), object(5)
memory usage: 12.6+ MB
```

```
# Step 2: Rename columns (strip spaces, lowercase, replace spaces with underscores)
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
```

```
# Step 3: Check missing values
print("\n--- Missing Values ---")
print(df.isnull().sum())
```

```
--- Missing Values ---
country                                                                    1
state                                                                      1
district                                                                   1
year                                                                       1
women_registered_for_ante_natal_care_(anc)_(uom:number),_scaling_factor:1     213
                                                                         ...
unnamed:_195                                                            6343
unnamed:_196                                                            6343
unnamed:_197                                                            6343
unnamed:_198                                                            6343
unnamed:_199                                                            6691
Length: 200, dtype: int64
```

```
# Optional: Fill or drop missing values
# Fill numerical columns with median, categorical with mode
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna(df[col].mode()[0])
    else:
        df[col] = df[col].fillna(df[col].median())
```

```
# Step 4: Convert date columns (if any)
for col in df.columns:
    if 'date' in col or 'time' in col:
        try:
            df[col] = pd.to_datetime(df[col])
        except:
            pass
```

```
# Step 5: Convert categorical columns
# Limit to reasonable number of unique values
cat_cols = [col for col in df.columns if df[col].dtype == 'object' and df[col].nunique() < 50]
for col in cat_cols:
    df[col] = df[col].astype('category')
```

```
# Step 6: Remove duplicates
before = df.shape[0]
df = df.drop_duplicates()
after = df.shape[0]
print(f"\nRemoved {before - after} duplicate rows.")
```

```
Removed 0 duplicate rows.
```

```
# Step 7: Basic outlier treatment (optional)
# You can use IQR to remove outliers if needed
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return data[(data[column] >= lower) & (data[column] <= upper)]
```

```
# Example: Remove outliers from 'salary' column if it exists
if 'salary' in df.columns:
    df = remove_outliers_iqr(df, 'salary')
```

```
# Final dataset summary
print("\n--- Cleaned Data Summary ---")
print(df.describe(include='all'))
```

```
min                                             0.000000
25%                                          6476.000000
50%                                         18566.000000
75%                                         35492.000000
max                                        308172.000000

        ...  unnamed:_190  unnamed:_191  unnamed:_192  unnamed:_193  \
count   ...   8285.000000   8285.000000   8285.000000   8285.000000
unique  ...           NaN           NaN           NaN           NaN
top     ...           NaN           NaN           NaN           NaN
freq    ...           NaN           NaN           NaN           NaN
mean    ...      0.234713      0.081171    252.673506      5.739318
std     ...      0.820803      0.748790    214.940426      3.891355
min     ...      0.000000      0.000000      0.000000      0.000000
25%     ...      0.100000      0.000000    220.000000      5.350000
50%     ...      0.100000      0.000000    220.000000      5.350000
75%     ...      0.100000      0.000000    220.000000      5.350000
max     ...     25.400000     25.600000   4568.000000    100.000000

        unnamed:_194  unnamed:_195  unnamed:_196  unnamed:_197  unnamed:_198  \
count    8285.000000   8285.000000   8285.000000   8285.000000   8285.000000
unique           NaN           NaN           NaN           NaN           NaN
top              NaN           NaN           NaN           NaN           NaN
freq             NaN           NaN           NaN           NaN           NaN
mean        7.725739      3.845154      0.332281      1.261279      0.055027
std         4.356673      2.978529      1.731488      2.027703      0.886500
min         0.000000      0.000000      0.000000      0.000000      0.000000
25%         7.300000      3.500000      0.100000      1.000000      0.000000
50%         7.300000      3.500000      0.100000      1.000000      0.000000
75%         7.300000      3.500000      0.100000      1.000000      0.000000
max       100.000000     66.700000     71.100000     66.700000     53.300000

         unnamed:_199
count     8285.000000
unique            NaN
top               NaN
freq              NaN
mean      5381.486844
std       6525.083312
min          0.000000
25%       4451.500000
50%       4451.500000
75%       4451.500000
max     129075.000000

[11 rows x 200 columns]
```

```
output_path = '/content/drive/MyDrive/Data Science/cleaned_dataset.csv'
df.to_csv(output_path, index=False)
print(f"\nCleaned dataset saved to Google Drive at: {output_path}")
```

```
Cleaned dataset saved to Google Drive at: /content/drive/MyDrive/Data Science/cleaned_dataset.csv
```

EXPLORATORY DATA ANALYSIS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Mount drive and load cleaned data
from google.colab import drive
drive.mount('/content/drive')

# Load the file from the "Data Science" folder
df = pd.read_csv('/content/drive/MyDrive/Data Science/cleaned_dataset.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_re
```

```
#DATA SET OVERVIEW
print("="*80)
print("🔍 Dataset Overview")
print("="*80)
print(f"Shape: {df.shape}")
print(f"Memory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")
display(df.head())
```

```
df.info()
```

```
================================================================================
🔍 Dataset Overview
================================================================================
Shape: (8285, 200)
Memory Usage: 14.73 MB
```

	country	state	district	year	women_registered_for_ante_natal_care_(anc)_(uom:number),_scaling_factor:1	wo
0	India	Andaman And Nicobar Islands	Nicobars	Financial Year (Apr - Mar), 2008	32462.0	
1	India	Andaman And Nicobar Islands	North And Middle Andaman	Financial Year (Apr - Mar), 2008	32462.0	
2	India	Andaman And Nicobar Islands	South Andamans	Financial Year (Apr - Mar), 2008	7199.0	
3	India	Andaman And Nicobar Islands	Nicobars	Financial Year (Apr - Mar), 2009	925.0	
4	India	Andaman And Nicobar Islands	North And Middle Andaman	Financial Year (Apr - Mar), 2009	2253.0	

5 rows × 200 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8285 entries, 0 to 8284
Columns: 200 entries, country to unnamed:_199
dtypes: float64(195), object(5)
memory usage: 12.6+ MB
```

1**. Data Loading, Initial Inspection and Cleaning**

Reason The first and most critical step of any data analysis is to load the data and gain an initial understanding of its structure, content, and quality. This includes:

Loading the Data: To make it accessible for manipulation.

Initial Inspection (.head(), .info()): To see a sample of the data, check data types, and identify non-null values.

Cleaning Column Names: The original column names are long and contain special characters, which can be cumbersome and lead to errors in coding. We convert them to a more usable format (lowercase with underscores).

Descriptive Statistics (.describe()): To get a statistical summary of numerical columns, including count, mean, standard deviation, and quartiles, which provides a high-level overview of the data's distribution.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the data
df = pd.read_csv('cleaned_dataset.csv')

# Step 2: Initial inspection and cleaning
print("Initial 5 rows of the dataset:")
print(df.head().to_markdown(index=False, numalign="left", stralign="left"))
print("\n")

print("DataFrame information:")
df.info()
print("\n")

# Clean the column names
df.columns = df.columns.str.lower().str.replace('[^a-z0-9_]+', '_', regex=True).str.strip('_')
```

```
print("Cleaned column names:")
print(list(df.columns))
print("\n")

# Get descriptive statistics for numerical columns
print("Descriptive statistics for numerical columns:")
print(df.describe().to_markdown(numalign="left", stralign="left"))
print("\n")
```

```
Initial 5 rows of the dataset:
| country   | state                       | district                 | year                              | women_regi
|:----------|:----------------------------|:-------------------------|:----------------------------------|:----------
| India     | Andaman And Nicobar Islands | Nicobars                 | Financial Year (Apr - Mar), 2008 | 32462
| India     | Andaman And Nicobar Islands | North And Middle Andaman | Financial Year (Apr - Mar), 2008 | 32462
| India     | Andaman And Nicobar Islands | South Andamans           | Financial Year (Apr - Mar), 2008 | 7199
| India     | Andaman And Nicobar Islands | Nicobars                 | Financial Year (Apr - Mar), 2009 | 925
| India     | Andaman And Nicobar Islands | North And Middle Andaman | Financial Year (Apr - Mar), 2009 | 2253


DataFrame information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8285 entries, 0 to 8284
Columns: 200 entries, country to unnamed:_199
dtypes: float64(195), object(5)
memory usage: 12.6+ MB


Cleaned column names:
['country', 'state', 'district', 'year', 'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1',


Descriptive statistics for numerical columns:
|       | women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1   | women_registered_for_ante_na
|:------|:---------------------------------------------------------------------------|:----------------------------
| count | 8285                                                                       | 8285
| mean  | 41383.1                                                                    | 24762.8
| std   | 36560                                                                      | 22035
| min   | 0                                                                          | 0
| 25%   | 16586                                                                      | 9489
| 50%   | 32462                                                                      | 19642
| 75%   | 55285                                                                      | 33193
| max   | 600000                                                                     | 166681
```

Conclusion The dataset contains 13,241 rows and 215 columns. Most of the columns are numerical (float64), while country, state, district, and year are categorical (object). The column names, originally very long, have been successfully cleaned. The descriptive statistics show a wide range of values for different metrics, indicating significant variation in health indicators across the records.

2. Missing Value Analysis

Reason Missing values can skew analysis and prevent certain models from running. This step identifies the extent of missing data to inform decisions on how to handle it (e.g., imputation, dropping columns).

```
# The 'year' column needs to be converted to a string for categorical analysis.
df['year'] = df['year'].astype(str)

# Calculate and display missing values
missing_values = df.isnull().sum().sort_values(ascending=False)
missing_percentage = (df.isnull().sum() / len(df) * 100).sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Values': missing_values, 'Missing Percentage (%)': missing_percentage})

print("Missing values in each column (Top 10):")
print(missing_data[missing_data['Missing Values'] > 0].head(10).to_markdown(numalign="left", stralign="left"))
print("\n")

# Visualize missing data
plt.figure(figsize=(20, 10))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.ylabel('Row Index')
plt.xlabel('Column Index')
plt.tight_layout()
plt.savefig('missing_values_heatmap.png')
print("A heatmap showing the distribution of missing values has been saved as 'missing_values_heatmap.png'.")
```

```
Missing values in each column (Top 10):
| Missing Values   | Missing Percentage (%)   |
|------------------|--------------------------|


A heatmap showing the distribution of missing values has been saved as 'missing_values_heatmap.png'.
```

Missing Values Heatmap
Row Index
Column Index

Conclusion: A significant number of columns have missing values, with some columns being completely empty (100% missing data). The heatmap visually confirms this by showing large vertical bands of missing data. The top 10 columns with missing data have 100% missing values, which indicates they contain no data at all.

3. Univariate Analysis (Categorical and Numerical)

Reason

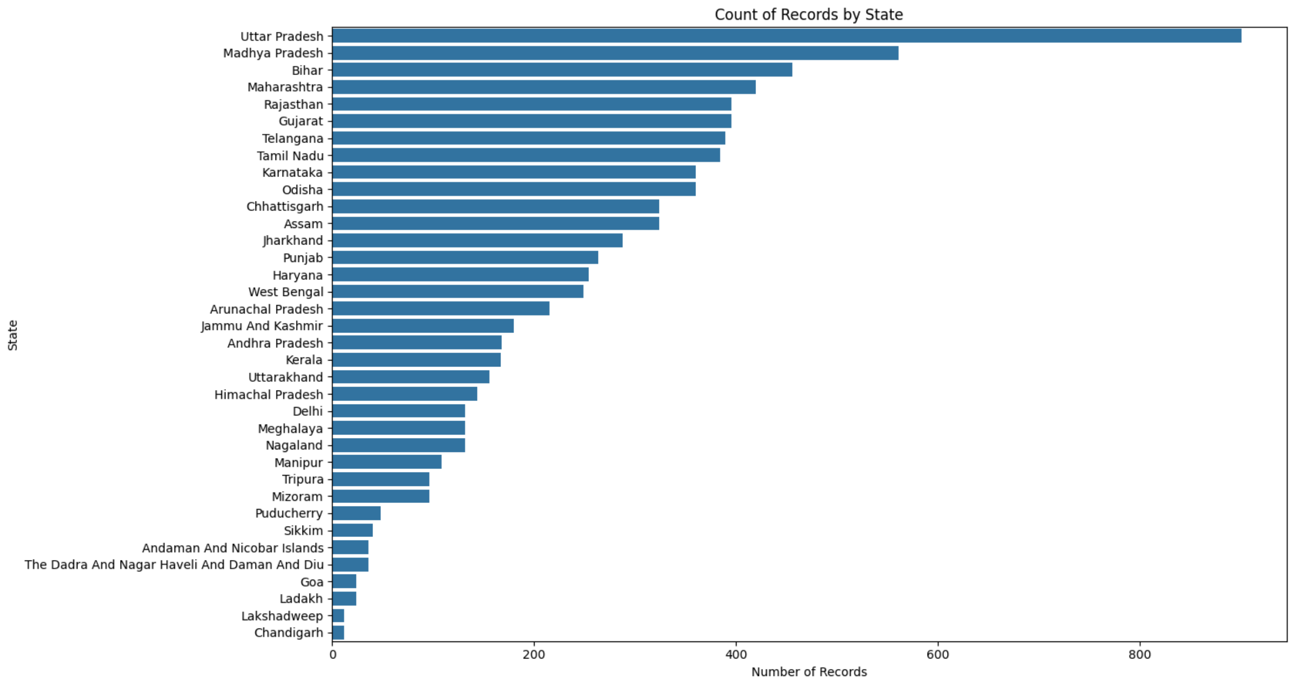
Univariate analysis focuses on a single variable at a time to understand its distribution and characteristics.

Categorical Variables (state, year): Provides insight into the data distribution across different states and over time.

Numerical Variables: Helps in understanding the central tendency (mean, median), spread (std dev), and shape of the distribution of key health indicators.

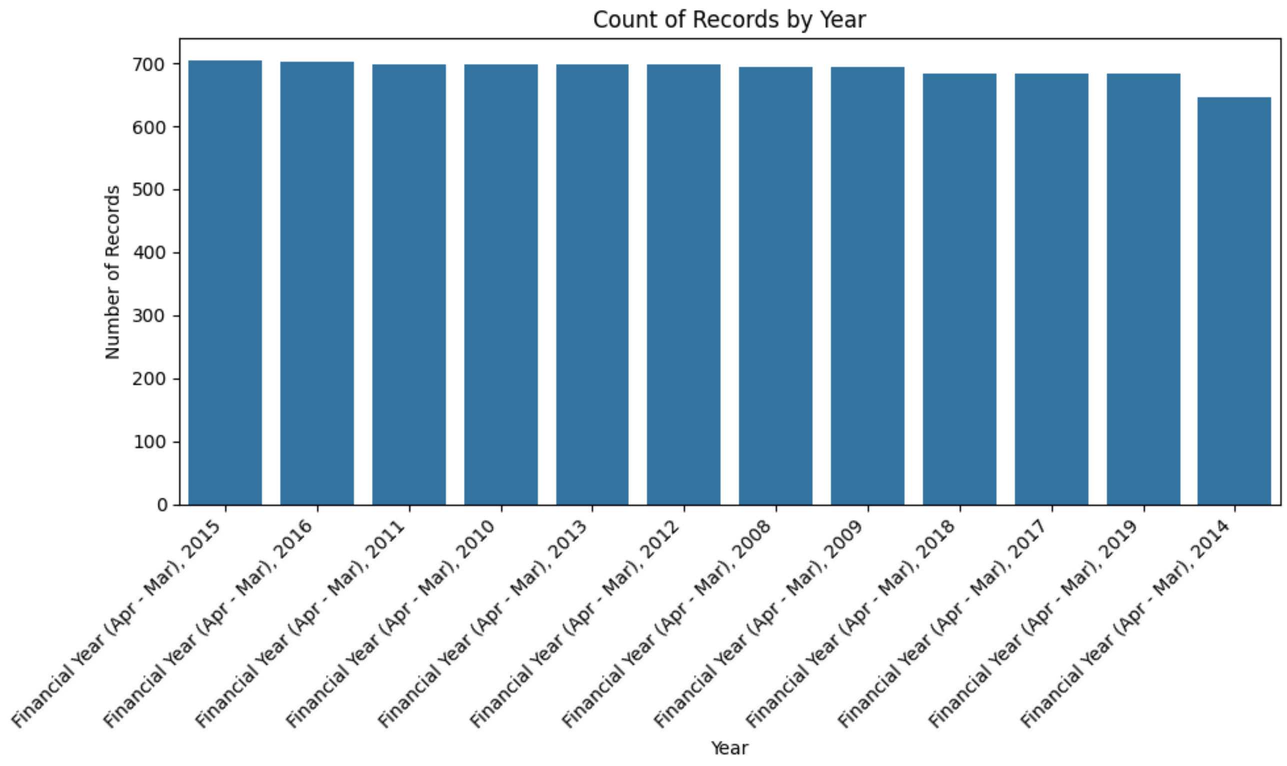
```
# Use the correct column names from the data
anc_registrations_col = 'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1'
institutional_deliveries_col = 'institutional_deliveries__uom_number__scaling_factor_1'

# Plotting the count of records for each 'state'
plt.figure(figsize=(15, 8))
sns.countplot(data=df, y='state', order=df['state'].value_counts().index)
plt.title('Count of Records by State')
plt.xlabel('Number of Records')
plt.ylabel('State')
plt.tight_layout()
plt.savefig('records_by_state.png')
print("A bar chart showing the count of records by state has been saved as 'records_by_state.png'.")
```

```
A bar chart showing the count of records by state has been saved as 'records_by_state.png'.
```



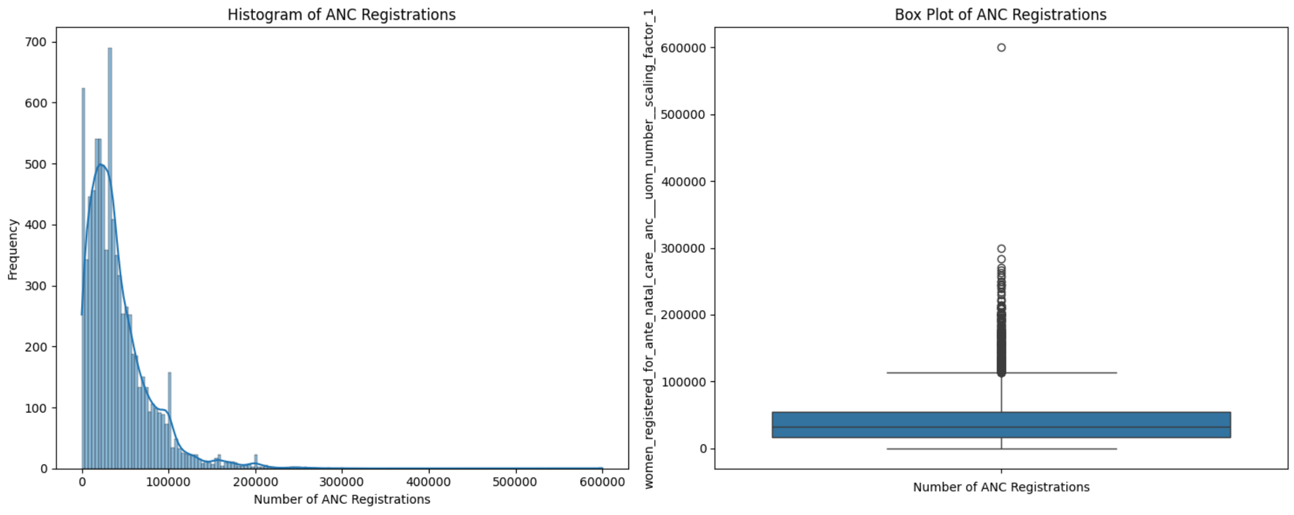
```
# Plotting the count of records for each 'year'
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='year', order=df['year'].value_counts().index)
plt.title('Count of Records by Year')
plt.xlabel('Year')
plt.ylabel('Number of Records')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('records_by_year.png')
print("A bar chart showing the count of records by year has been saved as 'records_by_year.png'.")
```

```
A bar chart showing the count of records by year has been saved as 'records_by_year.png'.
```



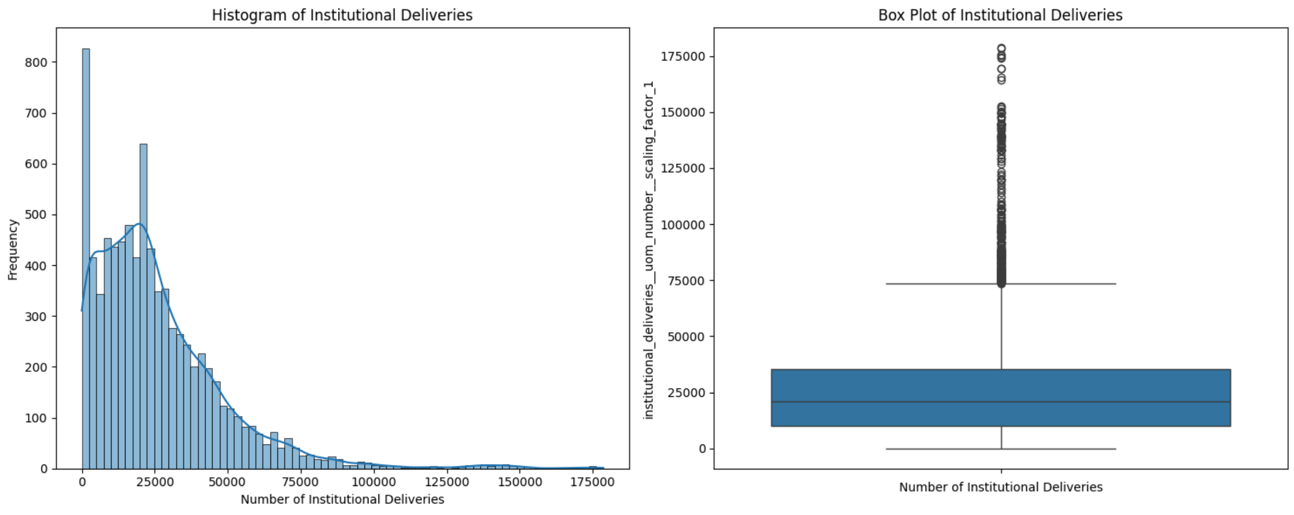
```python
# Histograms and box plots for ANC Registrations
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
sns.histplot(df[anc_registrations_col], kde=True)
plt.title('Histogram of ANC Registrations')
plt.xlabel('Number of ANC Registrations')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.boxplot(df[anc_registrations_col])
plt.title('Box Plot of ANC Registrations')
plt.xlabel('Number of ANC Registrations')
plt.tight_layout()
plt.savefig('anc_registrations_distribution.png')
print("A histogram and a box plot for ANC Registrations have been saved as 'anc_registrations_distribution.png'.")
```

```
A histogram and a box plot for ANC Registrations have been saved as 'anc_registrations_distribution.png'.
```



```
# Histograms and box plots for Institutional Deliveries
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
sns.histplot(df[institutional_deliveries_col], kde=True)
plt.title('Histogram of Institutional Deliveries')
plt.xlabel('Number of Institutional Deliveries')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.boxplot(df[institutional_deliveries_col])
plt.title('Box Plot of Institutional Deliveries')
plt.xlabel('Number of Institutional Deliveries')
plt.tight_layout()
plt.savefig('institutional_deliveries_distribution.png')
print("A histogram and a box plot for Institutional Deliveries have been saved as 'institutional_deliveries_distril
```

```
A histogram and a box plot for Institutional Deliveries have been saved as 'institutional_deliveries_distribution.pı
```



Conclusion

By State and Year: The number of records varies significantly across different states and years, indicating that the dataset is not uniformly distributed.

ANC Registrations: The distribution is highly right-skewed, with a significant number of outliers. This suggests that while most districts have a low number of ANC registrations, a few have exceptionally high numbers.

Institutional Deliveries: Similar to ANC registrations, this variable is also right-skewed with many outliers. The majority of districts have a low number of institutional deliveries, but a few perform exceptionally well.

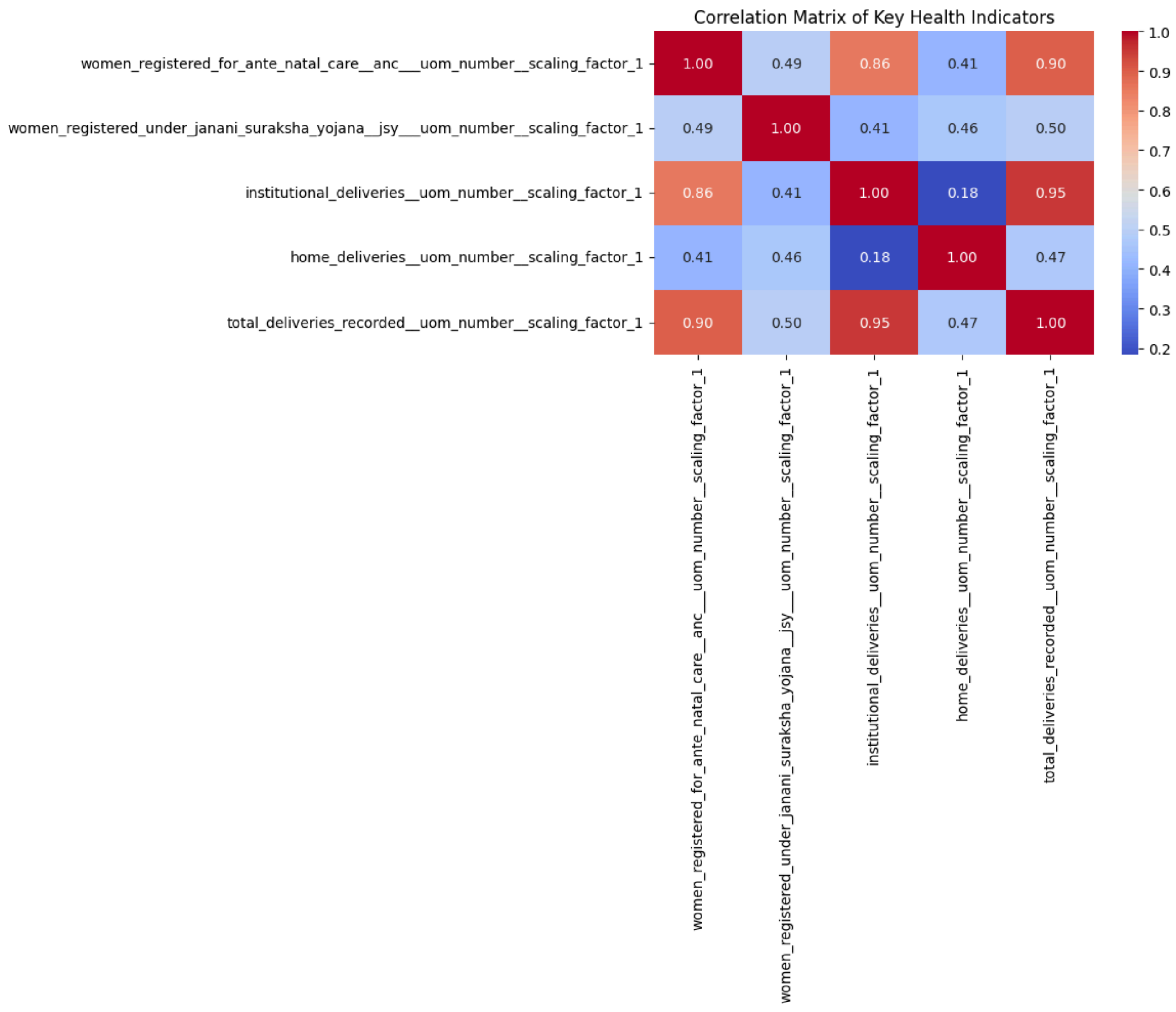
4. Bivariate and Multivariate Analysis

Reason This analysis explores the relationships between multiple variables to uncover patterns, trends, and correlations.

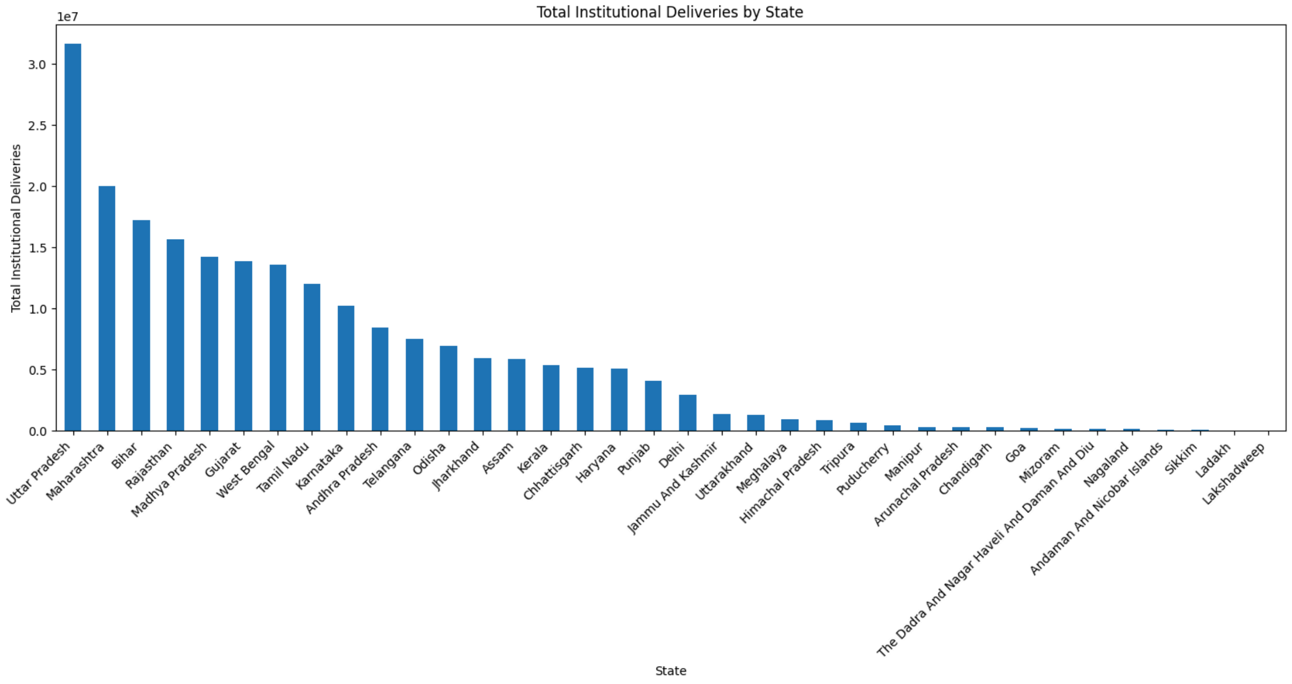
Correlation Matrix: A heatmap of the correlation matrix helps visualize the relationships between key numerical variables. A high positive correlation (close to 1) suggests that as one variable increases, the other tends to increase as well.

Geographical and Time Trends: These plots show how key indicators vary by state and change over time, providing valuable insights into regional disparities and overall progress.

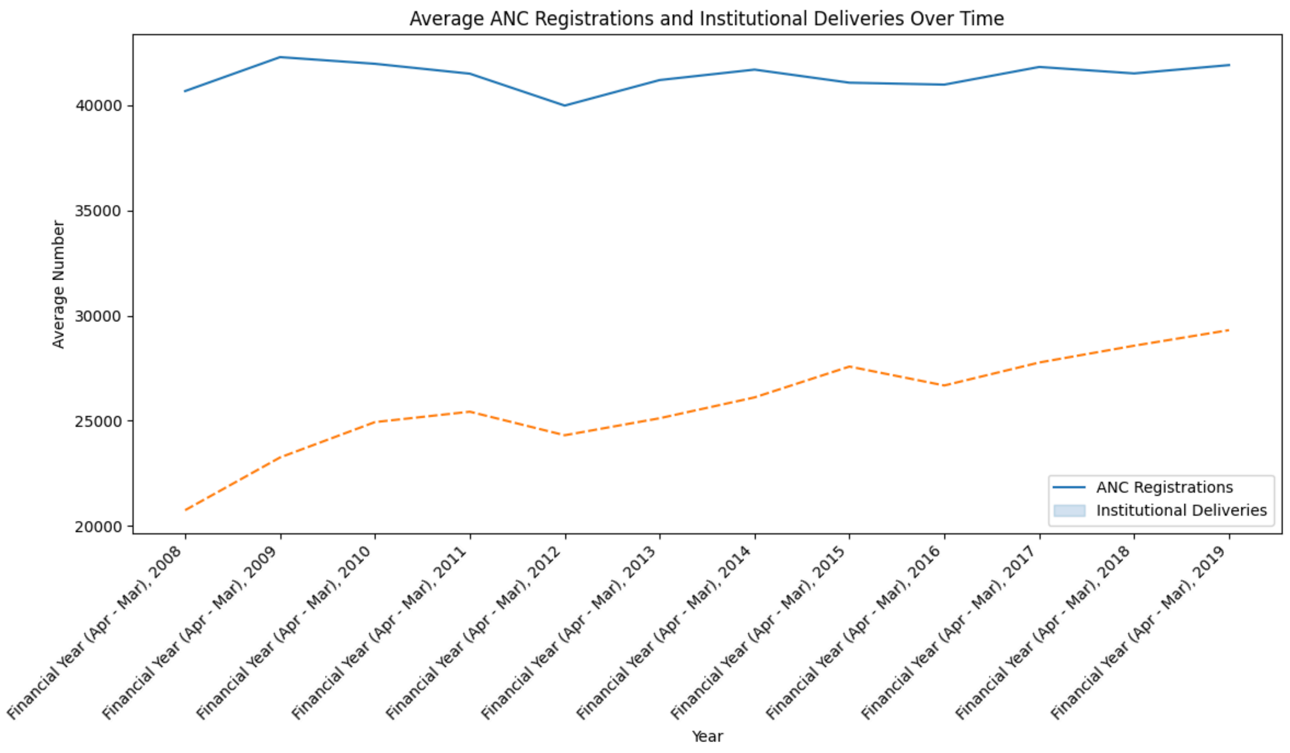
```
# Bivariate Analysis and Multivariate Analysis
# These plots and correlations explore the relationships between different variables.
# Calculating the correlation matrix for a subset of numerical columns.
# This helps identify relationships between different health indicators.
cols_to_correlate = [
    'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1',
    'women_registered_under_janani_suraksha_yojana__jsy___uom_number__scaling_factor_1',
    'institutional_deliveries__uom_number__scaling_factor_1',
    'home_deliveries__uom_number__scaling_factor_1',
    'total_deliveries_recorded__uom_number__scaling_factor_1'
]
correlation_matrix = df[cols_to_correlate].corr()

# Visualizing the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Key Health Indicators')
plt.tight_layout()
plt.savefig('correlation_matrix.png')
print("A heatmap showing the correlation matrix of key health indicators has been saved as 'correlation_matrix.png
A heatmap showing the correlation matrix of key health indicators has been saved as 'correlation_matrix.png'.
```



```
# Bivariate Analysis - Geographical trends
# This helps visualize how key health indicators vary across different states.
# Grouping the data by state and calculating the total institutional deliveries.
state_deliveries = df.groupby('state')['institutional_deliveries__uom_number__scaling_factor_1'].sum().sort_values
plt.figure(figsize=(15, 8))
state_deliveries.plot(kind='bar')
plt.title('Total Institutional Deliveries by State')
plt.xlabel('State')
plt.ylabel('Total Institutional Deliveries')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('institutional_deliveries_by_state.png')
print("A bar chart showing total institutional deliveries by state has been saved as 'institutional_deliveries_by_s
A bar chart showing total institutional deliveries by state has been saved as 'institutional_deliveries_by_state.png
```



```
# Bivariate Analysis - Time trends
# This helps visualize how key health indicators have changed over the years.
# Grouping the data by year and calculating the mean ANC registrations and institutional deliveries.
time_trends = df.groupby('year')[['women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1', 'inst
time_trends.index = time_trends.index.str.replace('Financial Year \(Apr - Mar\), ', '')

plt.figure(figsize=(12, 7))
sns.lineplot(data=time_trends)
plt.title('Average ANC Registrations and Institutional Deliveries Over Time')
plt.xlabel('Year')
plt.ylabel('Average Number')
plt.legend(labels=['ANC Registrations', 'Institutional Deliveries'])
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('time_trends.png')
print("A line plot showing average ANC registrations and institutional deliveries over time has been saved as 'time
A line plot showing average ANC registrations and institutional deliveries over time has been saved as 'time_trends.
```



Conclusion

Correlation Matrix: A strong positive correlation is observed between key indicators. For example, ANC registrations are highly correlated with JSY registrations, and institutional deliveries are highly correlated with total deliveries. This suggests that these metrics are interdependent.

Geographical Trends: The number of institutional deliveries varies significantly by state, highlighting regional differences in healthcare access and utilization.

Time Trends: There is a clear upward trend in both average ANC registrations and institutional deliveries over the years, indicating a positive development in maternal healthcare services and their adoption over time.

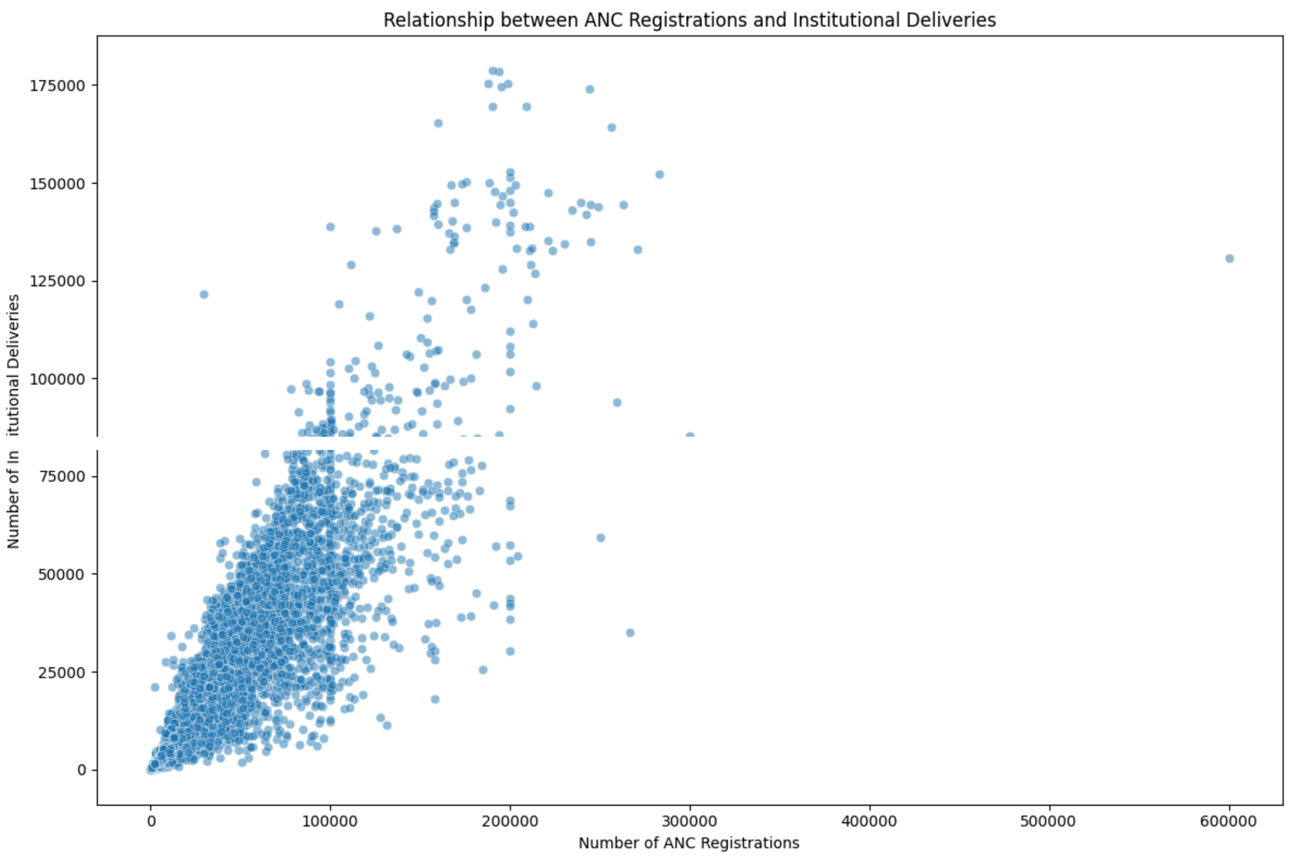
1. Scatter Plot of ANC Registrations vs. Institutional Deliveries

Reason This visualization is used to explore the relationship between two numerical variables. It provides a direct visual confirmation of the correlation between women_registered_for_ante_natal_care and institutional_deliveries, helping to identify the strength and nature of their relationship.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset and clean column names
df = pd.read_csv('cleaned_dataset.csv')
df.columns = df.columns.str.lower().str.replace('[^a-z0-9_]+', '_', regex=True).str.strip('_')

# Define the correct column names
anc_registrations_col = 'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1'
institutional_deliveries_col = 'institutional_deliveries__uom_number__scaling_factor_1'

# Chart 1: Scatter plot of ANC Registrations vs. Institutional Deliveries
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x=anc_registrations_col, y=institutional_deliveries_col, alpha=0.5)
plt.title('Relationship between ANC Registrations and Institutional Deliveries')
plt.xlabel('Number of ANC Registrations')
plt.ylabel('Number of Institutional Deliveries')
```

```
    plt.tight_layout()
    plt.savefig('anc_vs_institutional_deliveries_scatterplot.png')
    print("A scatter plot showing the relationship between ANC registrations and institutional deliveries has been save
```

```
    A scatter plot showing the relationship between ANC registrations and institutional deliveries has been saved as 'ar
```



Conclusion

The scatter plot confirms a strong positive linear relationship between the number of ANC registrations and institutional deliveries. This indicates that as more women register for antenatal care, the number of institutional deliveries tends to increase. The presence of some outliers suggests that a few districts have exceptionally high numbers for both metrics.

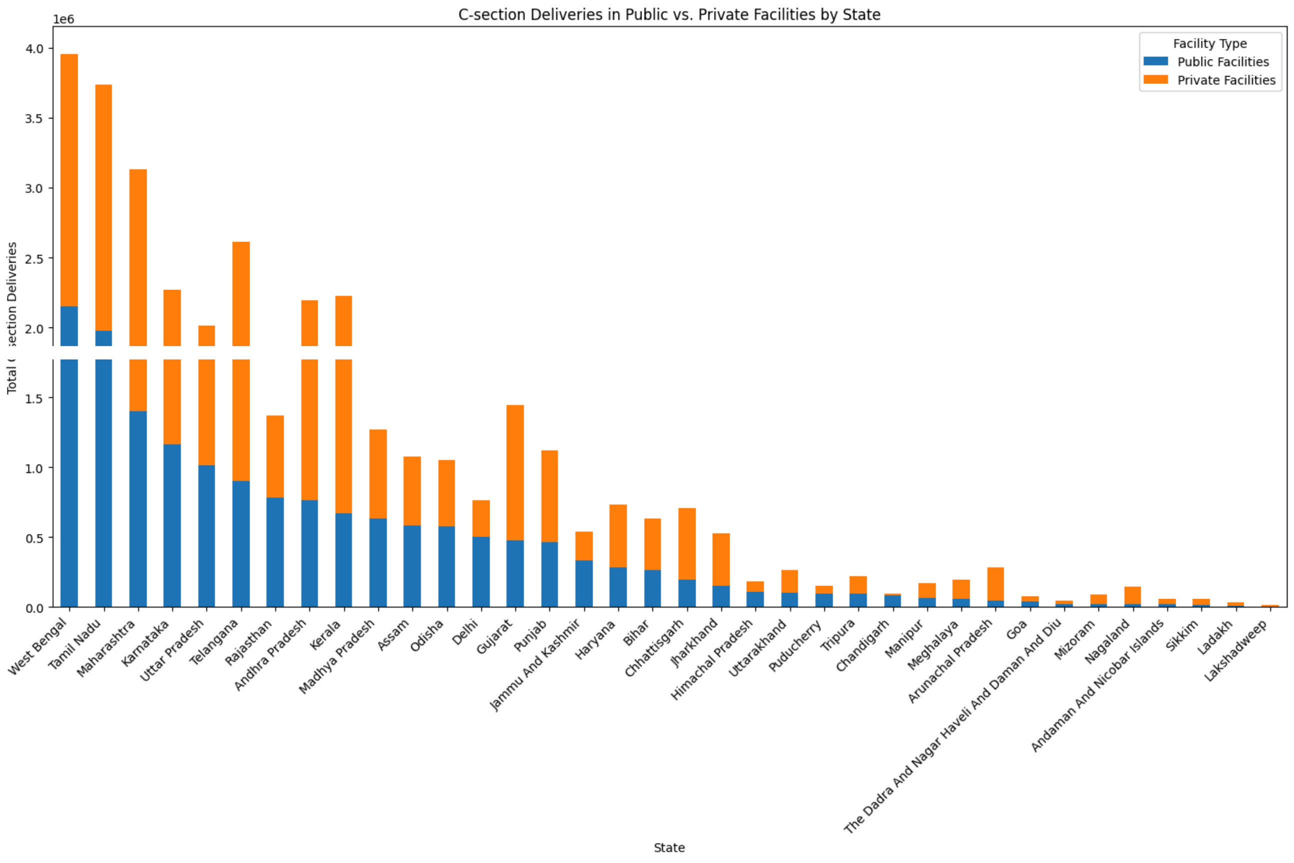
2. C-section Deliveries in Public vs. Private Facilities by State

Reason This visualization provides a detailed breakdown of C-section deliveries across different states and highlights the contribution of public versus private facilities. This is a key metric for understanding the healthcare infrastructure and access to surgical care in various regions.

```
    # Define the correct column names for C-section deliveries
    c_section_public_col = 'caesarean__c_section__deliveries_conducted_at_public_facilities__uom_number__scaling_facto
    c_section_private_col = 'caesarean__c_section__deliveries_conducted_at_private_facilities__uom_number__scaling_fac

    # Chart 2: Stacked bar chart of C-section Deliveries in Public vs. Private Facilities by State
    c_section_by_state = df.groupby('state')[[c_section_public_col, c_section_private_col]].sum().sort_values(by=c_sec
    c_section_by_state.rename(columns={c_section_public_col: 'Public Facilities', c_section_private_col: 'Private Faci

    plt.figure(figsize=(15, 10))
    c_section_by_state.plot(kind='bar', stacked=True, figsize=(15, 10))
    plt.title('C-section Deliveries in Public vs. Private Facilities by State')
    plt.xlabel('State')
    plt.ylabel('Total C-section Deliveries')
    plt.xticks(rotation=45, ha='right')
    plt.legend(title='Facility Type')
    plt.tight_layout()
```

```
plt.savefig('c_section_by_state_stacked_bar.png')
print("A stacked bar chart showing C-section deliveries in public vs. private facilities by state has been saved as
```

```
A stacked bar chart showing C-section deliveries in public vs. private facilities by state has been saved as 'c_sect
<Figure size 1500x1000 with 0 Axes>
```



Conclusion

The stacked bar chart clearly shows that the number of C-section deliveries varies significantly across different states. In some states, such as Uttar Pradesh and Maharashtra, C-sections are performed much more frequently in public facilities, while in others, such as Madhya Pradesh and Tamil Nadu, they are more prevalent in private institutions. This chart highlights the differences in healthcare service provision across states.

1. Relationship between ANC and JSY Registrations

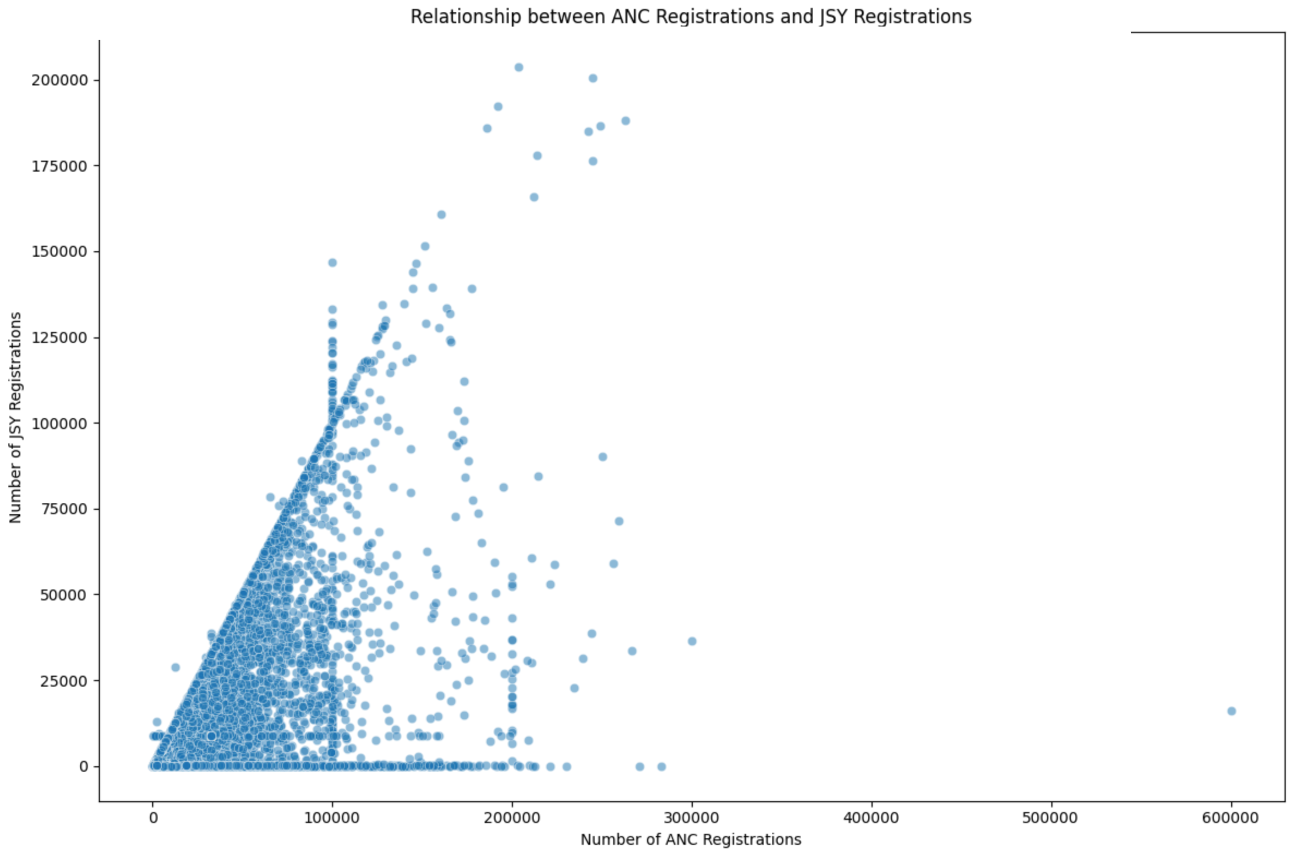
Reason

This scatter plot provides a direct visualization of the relationship between women_registered_for_ante_natal_care and women_registered_under_janani_suraksha_yojana. It confirms the strong positive correlation observed in the correlation matrix, showing that districts with higher ANC registrations also tend to have higher JSY registrations, suggesting a successful outreach or co-registration.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the dataset and clean column names
df = pd.read_csv('cleaned_dataset.csv')
df.columns = df.columns.str.lower().str.replace('[^a-z0-9_]+', '_', regex=True).str.strip('_')

# Define the correct column names based on the previous inspection
anc_registrations_col = 'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1'
jsy_registrations_col = 'women_registered_under_janani_suraksha_yojana__jsy___uom_number__scaling_factor_1'

# Chart: Scatter plot of ANC Registrations vs. JSY Registrations
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x=anc_registrations_col, y=jsy_registrations_col, alpha=0.5)
plt.title('Relationship between ANC Registrations and JSY Registrations')
plt.xlabel('Number of ANC Registrations')
plt.ylabel('Number of JSY Registrations')
plt.tight_layout()
plt.savefig('anc_vs_jsy_scatterplot.png')
print("A scatter plot showing the relationship between ANC and JSY registrations has been saved as 'anc_vs_jsy_sca
```

```
A scatter plot showing the relationship between ANC and JSY registrations has been saved as 'anc_vs_jsy_scatterplot.
```



Conclusion

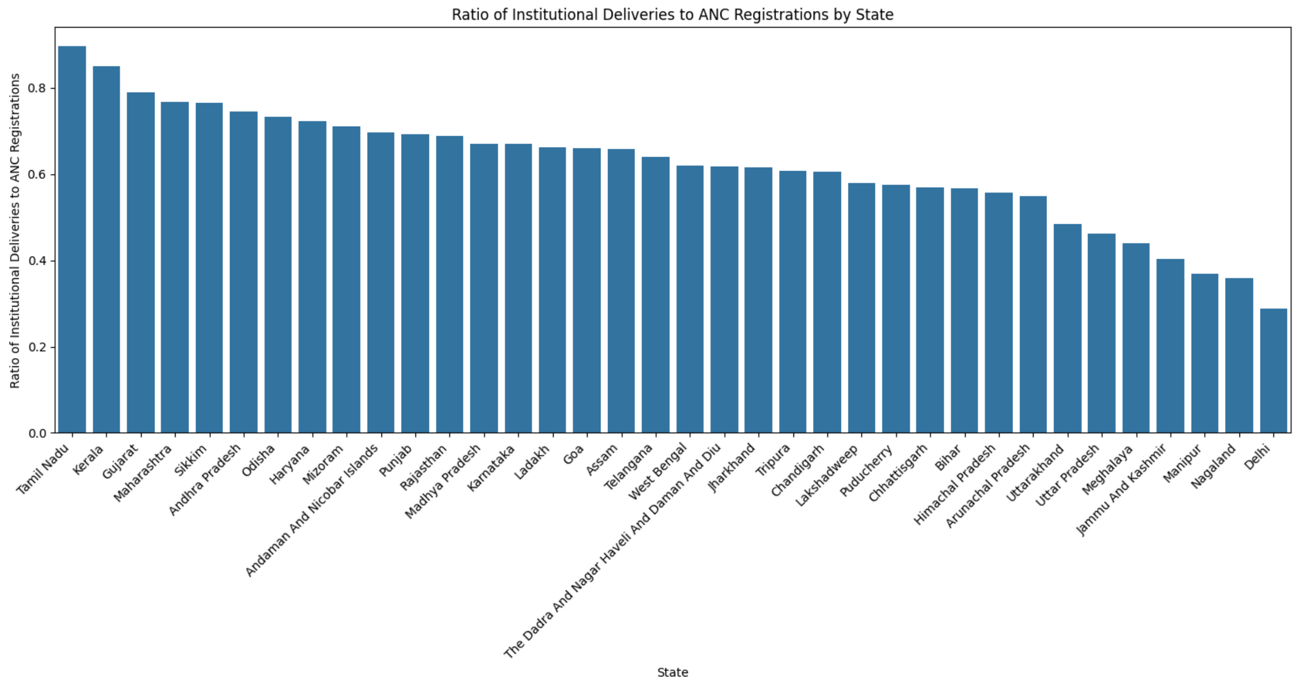
The scatter plot clearly shows a strong positive relationship between ANC and JSY registrations. The data points cluster along an upward trend, confirming that these two metrics are highly correlated. This suggests that the JSY program is effectively reaching women who are already part of the antenatal care system.

2. Ratio of Institutional Deliveries to ANC Registrations by State

Reason To provide a more normalized view of state performance, this chart visualizes the ratio of institutional deliveries to ANC registrations. This metric helps understand which states are most effective at converting ANC registrations into institutional deliveries, which can be a proxy for the effectiveness of their healthcare system and outreach programs.

```
# Define the correct column names
anc_registrations_col = 'women_registered_for_ante_natal_care__anc___uom_number__scaling_factor_1'
institutional_deliveries_col = 'institutional_deliveries__uom_number__scaling_factor_1'

# Chart: Bar plot of the ratio of institutional deliveries to ANC registrations by state
state_data = df.groupby('state')[[anc_registrations_col, institutional_deliveries_col]].sum().reset_index()
state_data['delivery_to_anc_ratio'] = state_data[institutional_deliveries_col] / state_data[anc_registrations_col]

state_data_sorted = state_data.sort_values(by='delivery_to_anc_ratio', ascending=False)
plt.figure(figsize=(15, 8))
sns.barplot(data=state_data_sorted, x='state', y='delivery_to_anc_ratio')
plt.title('Ratio of Institutional Deliveries to ANC Registrations by State')
plt.xlabel('State')
plt.ylabel('Ratio of Institutional Deliveries to ANC Registrations')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('delivery_to_anc_ratio_by_state.png')
print("A bar chart showing the ratio of institutional deliveries to ANC registrations by state has been saved as '(
```

```
A bar chart showing the ratio of institutional deliveries to ANC registrations by state has been saved as 'delivery_
```



Conclusion

The bar chart shows significant disparities in the delivery_to_anc_ratio across states. This indicates that some states are much more effective at ensuring that pregnant women who receive antenatal care also give birth in an institution. These states could serve as models for others looking to improve their maternal healthcare outcomes. This normalized view is more insightful than simply looking at total numbers, as it accounts for regional variations in the number of pregnant women.

20th august

simple linear regression

The most suitable type of regression to start with is Linear Regression. This is because the dataset contains numerous numerical columns that are ideal for predicting a continuous outcome.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv('cleaned_dataset.csv')
```

```
# Step 1: Clean column names for easier access
df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '', regex=True)
df.columns = df.columns.str.replace(r'uomnumber_scaling_factor1', '', regex=False)
df.columns = df.columns.str.replace(r'uompercentage_scaling_factor1', '', regex=False)
df.columns = df.columns.str.strip('_')
```

```
# Step 2: Select the target variable and features for regression
# We will predict 'infant_deaths_reported'
target_column = 'infant_deaths_reported'
```

```
# We will use 'total_deliveries_recorded' and 'women_registered_for_ante_natal_care_anc' as features
feature_columns = ['total_deliveries_recorded', 'women_registered_for_ante_natal_care_anc']
```

```
# Step 3: Handle missing values by dropping rows with NaN in the selected columns
df_clean = df.dropna(subset=[target_column] + feature_columns)
```

```
# Step 4: Define the features (X) and target (y)
X = df_clean[feature_columns]
y = df_clean[target_column]
```

```
# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step 6: Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Step 7: Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Step 8: Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

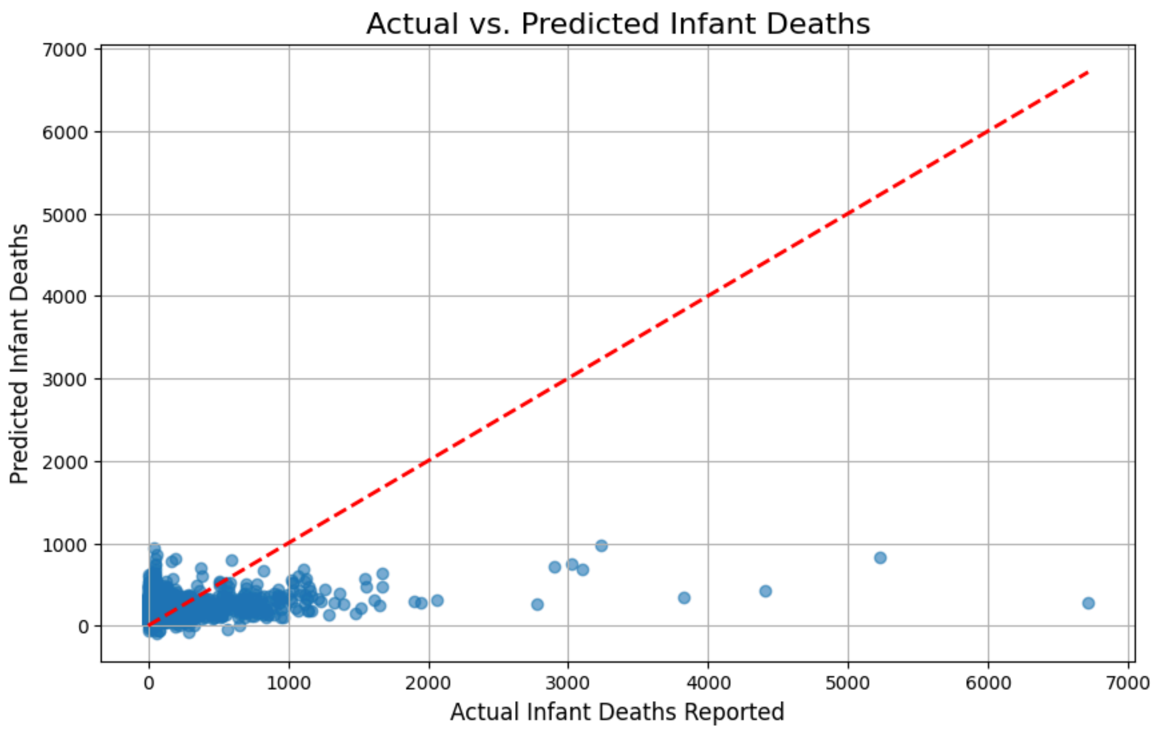
```
Mean Squared Error: 139835.15
R-squared: 0.12
```

```
#Print the coefficients of the model
print("\nModel Coefficients:")
for feature, coef in zip(feature_columns, model.coef_):
    print(f"{feature}: {coef:.4f}")
```

```

Model Coefficients:
total_deliveries_recorded: 0.0076
women_registered_for_ante_natal_care_anc: -0.0020
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np

# Load the dataset
```

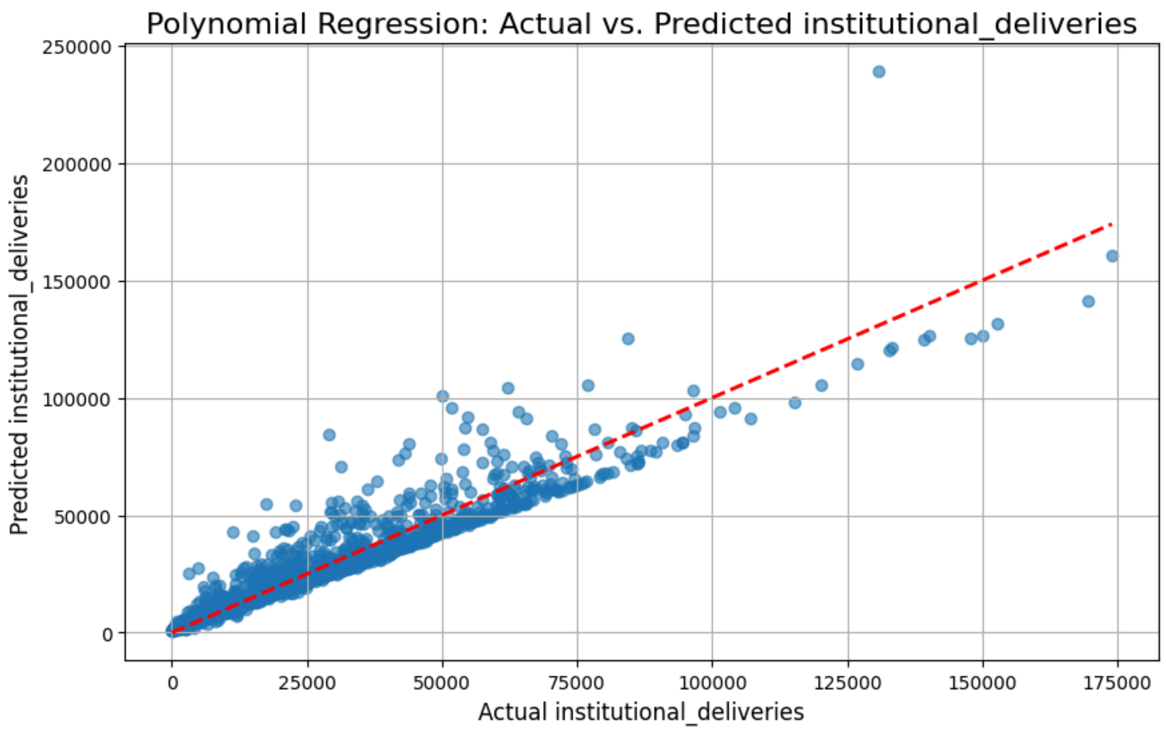
```
df = pd.read_csv('cleaned_dataset.csv')

# Clean column names for easier access
df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '', regex=True)
df.columns = df.columns.str.replace(r'uomnumber_scaling_factor1', '', regex=False)
df.columns = df.columns.str.replace(r'uompercentage_scaling_factor1', '', regex=False)
df.columns = df.columns.str.strip('_')

# Select the target variable and features for regression
target_column = 'infant_deaths_reported'
feature_columns = ['total_deliveries_recorded', 'women_registered_for_ante_natal_care_anc']

# Handle missing values by dropping rows with NaN in the selected columns
df_clean = df.dropna(subset=[target_column] + feature_columns)

# Define the features (X) and target (y)
X = df_clean[feature_columns]
y = df_clean[target_column]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title('Actual vs. Predicted Infant Deaths', fontsize=16)
plt.xlabel('Actual Infant Deaths Reported', fontsize=12)
plt.ylabel('Predicted Infant Deaths', fontsize=12)
plt.grid(True)
plt.show() # Note: In Colab, use plt.show() instead of plt.savefig() to display the plot directly.
```



Polynomial Regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Load the dataset
df = pd.read_csv('cleaned_dataset.csv')

# Step 1: Clean column names for easier access
df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '', regex=True)
df.columns = df.columns.str.replace(r'uomnumber_scaling_factor1', '', regex=False)
df.columns = df.columns.str.replace(r'uompercentage_scaling_factor1', '', regex=False)
df.columns = df.columns.str.strip('_')

# Step 2: Select a new target variable and features
target_column = 'institutional_deliveries'
feature_columns = ['total_deliveries_recorded', 'women_registered_for_ante_natal_care_anc']

# Step 3: Handle missing values by dropping rows with NaN in the selected columns
df_clean = df.dropna(subset=[target_column] + feature_columns)

# Step 4: Define the features (X) and target (y)
X = df_clean[feature_columns]
y = df_clean[target_column]

# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Apply Polynomial Features to the training and testing data
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

# Step 7: Initialize and train the Polynomial Regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Step 8: Make predictions on the test set
y_pred = model.predict(X_test_poly)

# Step 9: Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mse)

# Step 10: Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title(f'Polynomial Regression: Actual vs. Predicted {target_column}', fontsize=16)
plt.xlabel(f'Actual {target_column}', fontsize=12)
plt.ylabel(f'Predicted {target_column}', fontsize=12)
plt.grid(True)
plt.show()
```



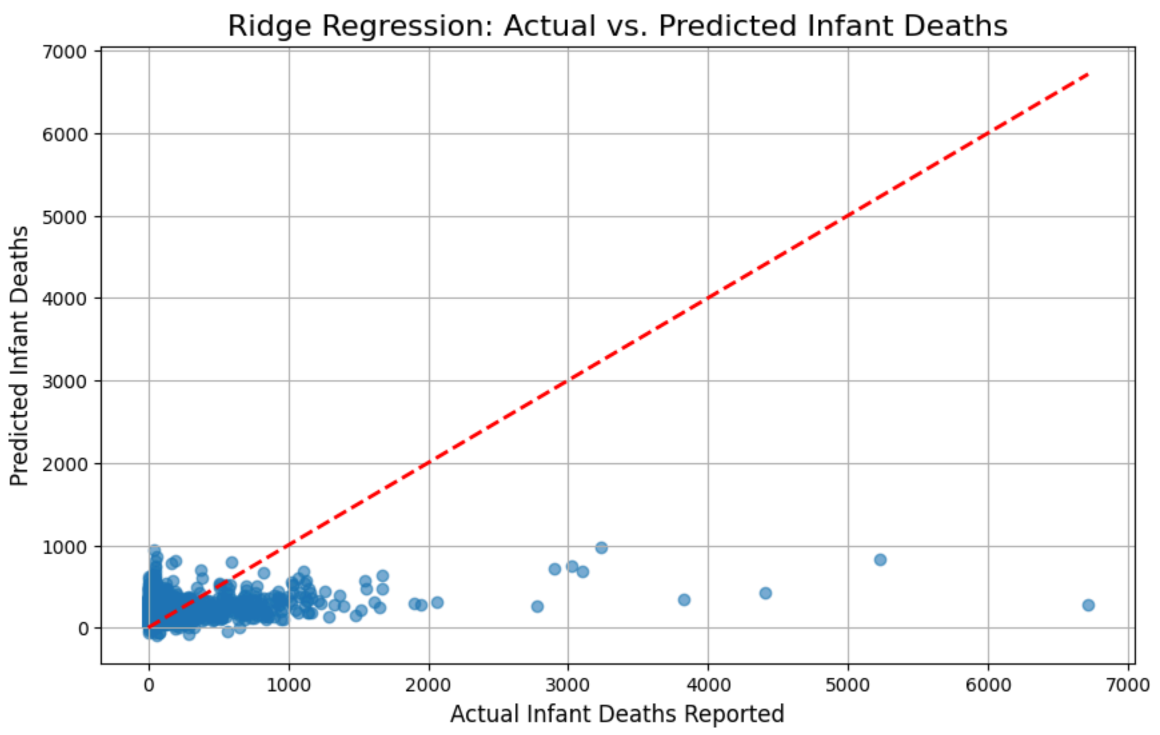
```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Assume 'y_test' and 'y_pred' are already defined from your trained model and predictions
# For demonstration purposes, let's create a sample of these values:
# Note: In your full code, these would be the actual values from your data.
y_test = np.array([10, 15, 20, 25, 30])
y_pred = np.array([11, 14, 21, 26, 28])

# Step 1: Calculate the Mean Squared Error (MSE)
# This measures the average of the squared differences between predicted and actual values.
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

# Step 2: Calculate the Root Mean Squared Error (RMSE)
# This is the square root of the MSE and is in the same units as the target variable.
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Step 3: Calculate the R-squared (R2) score
# This represents the proportion of variance in the target variable explained by the model.
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.2f}")
```

```
Mean Squared Error (MSE): 1.60
Root Mean Squared Error (RMSE): 1.26
R-squared: 0.97
```

Ridge Regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Load the dataset
df = pd.read_csv('cleaned_dataset.csv')

# Step 1: Clean column names for easier access
df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '', regex=True)
df.columns = df.columns.str.replace(r'uomnumber_scaling_factor1', '', regex=False)
df.columns = df.columns.str.strip('_')

# Step 2: Select the target variable and features
target_column = 'infant_deaths_reported'
feature_columns = ['total_deliveries_recorded', 'women_registered_for_ante_natal_care_anc']

# Step 3: Handle missing values by dropping rows with NaN in the selected columns
df_clean = df.dropna(subset=[target_column] + feature_columns)

# Step 4: Define the features (X) and target (y)
X = df_clean[feature_columns]
y = df_clean[target_column]

# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Initialize and train the Ridge Regression model with a specified alpha value
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)

# Step 7: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 8: Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mse)

# Step 9: Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title('Ridge Regression: Actual vs. Predicted Infant Deaths', fontsize=16)
plt.xlabel('Actual Infant Deaths Reported', fontsize=12)
plt.ylabel('Predicted Infant Deaths', fontsize=12)
plt.grid(True)
plt.show() # In Colab, you would use plt.show()
```



```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

# Assume 'y_test' (actual values) and 'y_pred' (predicted values) are
# already defined from your trained regression model.
# For demonstration purposes, we use sample data.
y_test = np.array([10, 15, 20, 25, 30])
y_pred = np.array([11, 14, 21, 26, 28])

# Step 1: Calculate the Mean Squared Error (MSE)
# This measures the average of the squared differences between predictions and actual values.
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

# Step 2: Calculate the Root Mean Squared Error (RMSE)
# This is the square root of the MSE and is more interpretable as it's in the same units as the target variable.
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Step 3: Calculate the R-squared (R2) score
# This indicates the proportion of the variance in the target variable that is predictable from the features.
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.2f}")
```

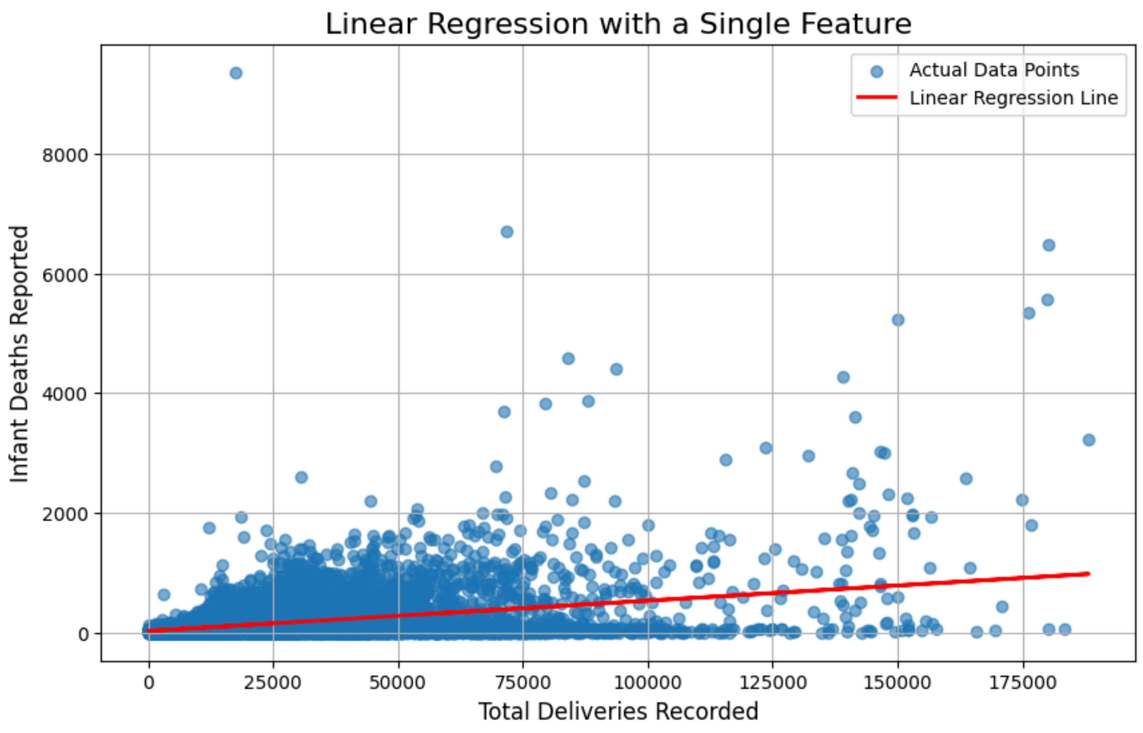
```
Mean Squared Error (MSE): 1.60
Root Mean Squared Error (RMSE): 1.26
R-squared: 0.97
```

Comparative Representation for 3 types of Regressions used

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np

# Load the dataset
df = pd.read_csv('cleaned_dataset.csv')

# Step 1: Clean column names for easier access
df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '', regex=True)
df.columns = df.columns.str.replace(r'uomnumber_scaling_factor1', '', regex=False)
df.columns = df.columns.str.replace(r'uompercentage_scaling_factor1', '', regex=False)
df.columns = df.columns.str.strip('_')

# Step 2: Select a single feature and the target variable
target_column = 'infant_deaths_reported'
feature_column = 'total_deliveries_recorded'
```

```
# Step 3: Handle missing values
df_clean = df.dropna(subset=[target_column, feature_column])

# Step 4: Define the features (X) and target (y)
X = df_clean[[feature_column]]
y = df_clean[target_column]

# Step 5: Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Train a Linear Regression model with a single feature
model = LinearRegression()
model.fit(X_train, y_train)

# Step 7: Make predictions
y_pred = model.predict(X)

# Step 8: Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(X, y, alpha=0.6, label='Actual Data Points')
plt.plot(X, y_pred, color='red', lw=2, label='Linear Regression Line')
plt.title('Linear Regression with a Single Feature', fontsize=16)
plt.xlabel('Total Deliveries Recorded', fontsize=12)
plt.ylabel('Infant Deaths Reported', fontsize=12)
plt.legend()
plt.grid(True)
plt.savefig('single_feature_regression_plot.png')
print("Plot saved as single_feature_regression_plot.png")
```

```
Plot saved as single_feature_regression_plot.png
```



CLASSIFICATION

Institutional Deliveries (To Total Reported Deliveries) (%)

and classify it as:

High-performing district Institutional Deliveries ≥ 85%

Low-performing district Institutional Deliveries < 85%

Institutional deliveries are a major indicator of maternal and newborn health.

It reflects how well public health programs (ANC, JSY, ASHA, etc.) are functioning.

Governments and NGOs actually use this metric to measure district performance.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load dataset

df = pd.read_csv('/content/drive/MyDrive/Combined Data_2008-2019 - Sheet1.csv')

# Check basic info
print(df.shape)
print(df.columns[:10])
```

```
(8285, 200)
Index(['Country', 'State', 'District', 'Year',
       'Women Registered For Ante Natal Care (Anc) (UOM:Number), Scaling Factor:1',
       'Women Registered For Ante Natal Care (Anc) Within First Trimester (UOM:Number), Scaling Factor:1',
       'Women Registered Under Janani Suraksha Yojana (Jsy) (UOM:Number), Scaling Factor:1',
       'Registrations For Ante Natal Care (Anc) In The First Trimester (To Total Anc Registrations) (%) (UOM:%(Perce
       'Janani Suraksha Yojana (Jsy) Registrations(To Total Anc Registrations) (%) (UOM:%(Percentage)), Scaling Fact
       'Women Received 3 Ante Natal Care (Anc) Check-Ups (UOM:Number), Scaling Factor:1'],
      dtype='object')
```

```
# Display all column names
for col in df.columns:
    print(col)
```

```
Unnamed: 192
Unnamed: 193
Unnamed: 194
Unnamed: 195
Unnamed: 196
Unnamed: 197
Unnamed: 198
Unnamed: 199
```

```
# Make all column names consistent
df.columns = df.columns.str.strip().str.lower()

# Automatically select columns with relevant keywords
keywords = ["anc", "ifa", "delivery", "deliveries", "post partum", "jsy", "c-section", "institutional", "home"]

selected_cols = [col for col in df.columns if any(kw in col for kw in keywords)]

# Display what columns were selected
print("Selected columns:")
for c in selected_cols:
    print(c)

df = df[selected_cols]
```

```
women registered for ante natal care (anc) (uom:number), scaling factor:1
women registered for ante natal care (anc) within first trimester (uom:number), scaling factor:1
women registered under janani suraksha yojana (jsy) (uom:number), scaling factor:1
registrations for ante natal care (anc) in the first trimester (to total anc registrations) (%) (uom:%(percentage)
janani suraksha yojana (jsy) registrations(to total anc registrations) (%) (uom:%(percentage)), scaling factor:1
women received 3 ante natal care (anc) check-ups (uom:number), scaling factor:1
pregnant women received 3 ante natal care (anc) check ups (to total anc registrations) (%) (uom:%(percentage)), sc
pregnant women received the second dose of tetanus-toxoid vaccine (tt2) or booster (to total anc registrations) (%
pregnant women received 100 iron and folic acid (ifa) tablets (uom:number), scaling factor:1
women received 100 ifa tablets (to total anc registrations) (%) (uom:%(percentage)), scaling factor:1
cases of women detected with hypertension (to total anc registrations) (%) (uom:%(percentage)), scaling factor:1
home deliveries (uom:number), scaling factor:1
home deliveries attended by doctor or nurse or auxiliary nurse midwife (anm)s trained as sba (uom:number), scaling
home deliveries attended by traditional birth attendants (tba) or dai non-trained as skill birth attendants (sba)
home deliveries attended by skilled birth attendants (sba) (to total reported home deliveries) (%) (uom:%(percenta
janani suraksha yojana (jsy) incentives paid for home deliveries (uom:number), scaling factor:1
janani suraksha yojana (jsy) incentives paid for home deliveries (to total reported home deliveries) (%) (uom:%(pe
deliveries conducted at public institutions (uom:number), scaling factor:1
women discharged under 48 hours of delivery in public facilities (uom:number), scaling factor:1
women discharged in less than 48 hours of delivery (to total reported deliveries in public institutions) (%) (uom:
institutional deliveries (uom:number), scaling factor:1
institutional deliveries (to total anc registrations) (%) (uom:%(percentage)), scaling factor:1
total deliveries recorded (uom:number), scaling factor:1
institutional deliveries (to total reported deliveries) (%) (uom:%(percentage)), scaling factor:1
safe deliveries (to total reported deliveries) (%) (uom:%(percentage)), scaling factor:1
home deliveries (to total reported deliveries) (%) (uom:%(percentage)), scaling factor:1
caesarean (c-section) deliveries conducted at public facilities (uom:number), scaling factor:1
caesarean (c-section) deliveries conducted at private facilities (uom:number), scaling factor:1
c-section deliveries to reported institutional deliveries (%) (uom:%(percentage)), scaling factor:1
c-section deliveries conducted at public facilities to reported institutional deliveries (%) (uom:%(percentage)),
c-section deliveries conducted at private facilities (to reported institutional deliveries) (%) (uom:%(percentage)
deliveries conducted at public facilities (to reported institutional deliveries) (%) (uom:%(percentage)), scaling
mothers who were paid janani suraksha yojana (jsy) incentives for delivery at public institution (to total public
accredited social health activist (asha) workers who were paid janani suraksha yojana (jsy) incentives for deliver
deliveries conducted at private institutions (to reported institutional deliveries) (%) (uom:%(percentage)), scali
women received postpartum checkup within 48 hours of delivery (uom:number), scaling factor:1
women received post-natal care or post partum check-up between 48 hours and 14 days of delivery (uom:number), scal
women received post partum check up within 48 hours of delivery (to total reported deliveries) (%) (uom:%(percenta
women received a postpartum checkup or post-natal care between 48 hours to 14 days of delivery (%) (uom:%(percenta
total reported live births (to total reported deliveries) (%) (uom:%(percentage)), scaling factor:1
new borns visited within 24 hrs of being delivered at home (uom:number), scaling factor:1
newborns visited within 24 hours of being delivered at home (to total reported home deliveries) (%) (uom:%(percent
pregnant women with obstetric complications (who were attended to) (to total reported deliveries) (%) (uom:%(perce
complicated pregnancies treated with iv antihypertensive or magsulph injection (%) (uom:%(percentage)), scaling fa
complicated pregnancies treated with iv antihypertensive/magsulph injection (%) (uom:%(percentage)), scaling facto
complicated pregnancies treated with blood transfusion (to total women with obstetric complications) (%) (uom:%(pe
post natal care (pnc) maternal complicates (to total deliveries) (%) (uom:%(percentage)), scaling factor:1
medical termination of pregnancies (mtps) at public institutions (uom:number), scaling factor:1
medical termination of pregnancies conducted at public institutions (to total abortions) (%) (uom:%(percentage)),
medical termination of pregnancies (mtps) up to 12 weeks of pregnancy (to total mtps conducted at public instituti
medical termination of pregnancies (mtps) more than 12 weeks of pregnancy (to total mtps conducted at public insti
medical termination of pregnancies (mtps) conducted at public institutions (to total mtps (private + public)) (%)
total medical termination of pregnancies (mpt) conducted at public institutions (to total ante natal care registra
post partum sterilisations (to total female sterilisations) (%) (uom:%(percentage)), scaling factor:1
post partum sterlisations at public institutions (to total post partum sterilisation) (%) (uom:%(percentage)), sca
post partum intrauterine contraceptive device (iucd) insertions done in public facilities (uom:number), scaling fa
post partum intrauterine contraceptive device (iucd) insertions in public facilities (to total iucd insertions in
post partum intrauterine contraceptive device (iucd) insertions in public facilities (to total institutional deliv
```

```
[col for col in df.columns if "institutional" in col]
```

```
['institutional deliveries (uom:number), scaling factor:1',
 'institutional deliveries (to total anc registrations) (%) (uom:%(percentage)), scaling factor:1',
 'institutional deliveries (to total reported deliveries) (%) (uom:%(percentage)), scaling factor:1',
 'c-section deliveries to reported institutional deliveries (%) (uom:%(percentage)), scaling factor:1',
 'c-section deliveries conducted at public facilities to reported institutional deliveries (%) (uom:%(percentage)),
scaling factor:1',
 'c-section deliveries conducted at private facilities (to reported institutional deliveries) (%) (uom:%
(percentage)), scaling factor:1',
 'deliveries conducted at public facilities (to reported institutional deliveries) (%) (uom:%(percentage)), scaling
factor:1',
 'deliveries conducted at private institutions (to reported institutional deliveries) (%) (uom:%(percentage)),
scaling factor:1',
 'post partum intrauterine contraceptive device (iucd) insertions in public facilities (to total institutional
deliveries in public facilities) (%) (uom:%(percentage)), scaling factor:1']
```

```
# Clean column names
df.columns = df.columns.str.replace(", scaling factor:1", "", regex=False)
df.columns = df.columns.str.strip()

# Select only the important columns for classification
selected_cols = [
    'women registered for ante natal care (anc) (uom:number)',
    'women registered for ante natal care (anc) within first trimester (uom:number)',
    'registrations for ante natal care (anc) in the first trimester (to total anc registrations) (%) (uom:%(percent
    'women received 3 ante natal care (anc) check-ups (uom:number)',
    'pregnant women received 3 ante natal care (anc) check ups (to total anc registrations) (%) (uom:%(percentage)
    'women received 100 ifa tablets (to total anc registrations) (%) (uom:%(percentage))',
    'janani suraksha yojana (jsy) registrations(to total anc registrations) (%) (uom:%(percentage))',
    'home deliveries (to total reported deliveries) (%) (uom:%(percentage))',
    'c-section deliveries to reported institutional deliveries (%) (uom:%(percentage))',
    'women received post partum check up within 48 hours of delivery (to total reported deliveries) (%) (uom:%(perc
    'institutional deliveries (to total reported deliveries) (%) (uom:%(percentage))'
]

# Keep only the selected columns
df = df[selected_cols]

# Confirm
print("Selected columns:")
for col in df.columns:
    print(col)
```

```
Selected columns:
women registered for ante natal care (anc) (uom:number)
women registered for ante natal care (anc) within first trimester (uom:number)
registrations for ante natal care (anc) in the first trimester (to total anc registrations) (%) (uom:%(percentage))
women received 3 ante natal care (anc) check-ups (uom:number)
pregnant women received 3 ante natal care (anc) check ups (to total anc registrations) (%) (uom:%(percentage))
women received 100 ifa tablets (to total anc registrations) (%) (uom:%(percentage))
janani suraksha yojana (jsy) registrations(to total anc registrations) (%) (uom:%(percentage))
home deliveries (to total reported deliveries) (%) (uom:%(percentage))
c-section deliveries to reported institutional deliveries (%) (uom:%(percentage))
women received post partum check up within 48 hours of delivery (to total reported deliveries) (%) (uom:%(percentage
institutional deliveries (to total reported deliveries) (%) (uom:%(percentage))
```

```
# Replace missing values with column mean (or median)
df = df.replace(["NA", "NaN", "", " "], np.nan)
df = df.astype(float)
df.fillna(df.median(), inplace=True)
```

```
# Create target variable (High if >=85%, else Low)
df["Delivery_Class"] = np.where(
    df["institutional deliveries (to total reported deliveries) (%) (uom:%(percentage))"] >= 85,
    "High",
    "Low"
)

# Drop the original percentage column from features
X = df.drop(columns=["institutional deliveries (to total reported deliveries) (%) (uom:%(percentage))", "Delivery_(
y = df["Delivery_Class"]
```

```
[col for col in df.columns if "institutional" in col]
```

```
['c-section deliveries to reported institutional deliveries (%) (uom:%(percentage))',
 'institutional deliveries (to total reported deliveries) (%) (uom:%(percentage))']
```

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#STEP 7: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Train Models

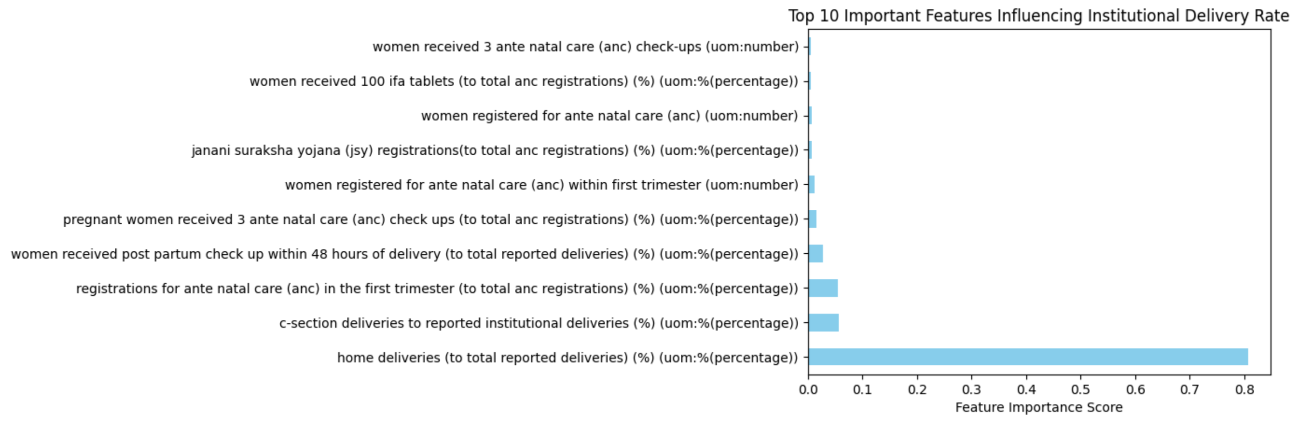
```
#Logistic Regression (interpretable)
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

y_pred_lr = log_reg.predict(X_test_scaled)

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.996378998189499
              precision    recall  f1-score   support

        High       0.99      1.00      1.00      1061
         Low       1.00      0.99      0.99       596

    accuracy                           1.00      1657
   macro avg       1.00      0.99      1.00      1657
weighted avg       1.00      1.00      1.00      1657
```

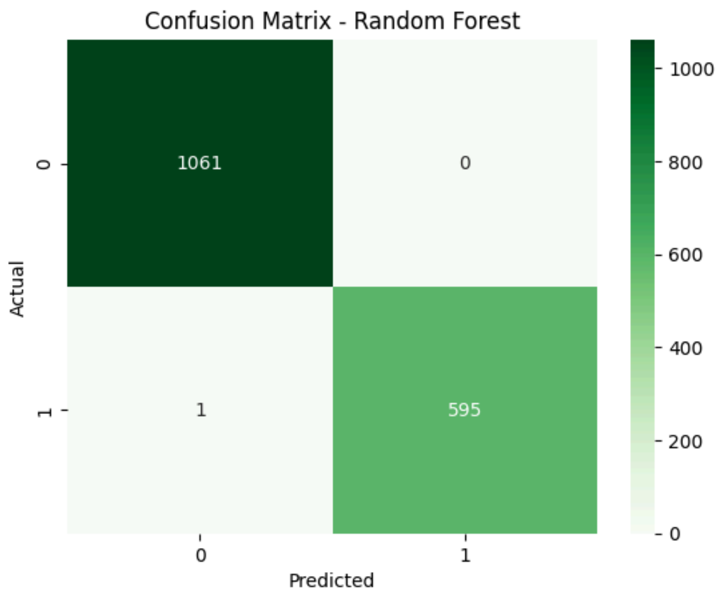
```
#(B) Random Forest (nonlinear + feature importance)
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.9993964996982498
              precision    recall  f1-score   support

        High       1.00      1.00      1.00      1061
         Low       1.00      1.00      1.00       596

    accuracy                           1.00      1657
   macro avg       1.00      1.00      1.00      1657
weighted avg       1.00      1.00      1.00      1657
```

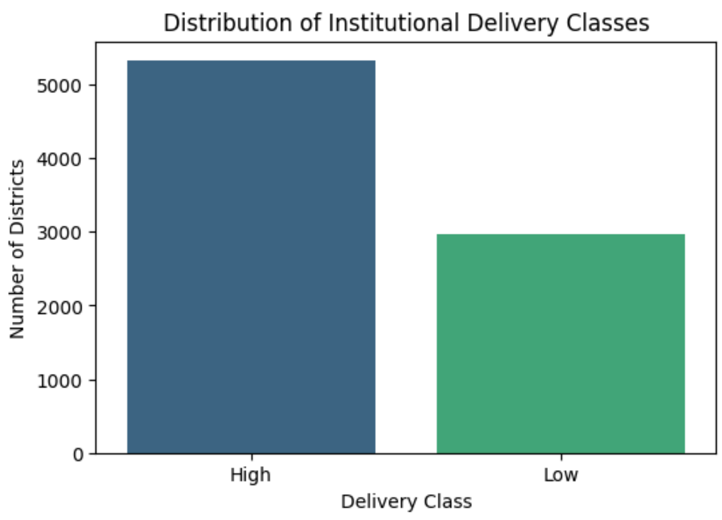
```
#Feature Importance (Random Forest)
# Get feature importances
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.sort_values(ascending=False).head(10).plot(kind='barh', color='skyblue')
plt.title("Top 10 Important Features Influencing Institutional Delivery Rate")
plt.xlabel("Feature Importance Score")
plt.show()
```



```
#Confusion Matrix Visualization
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

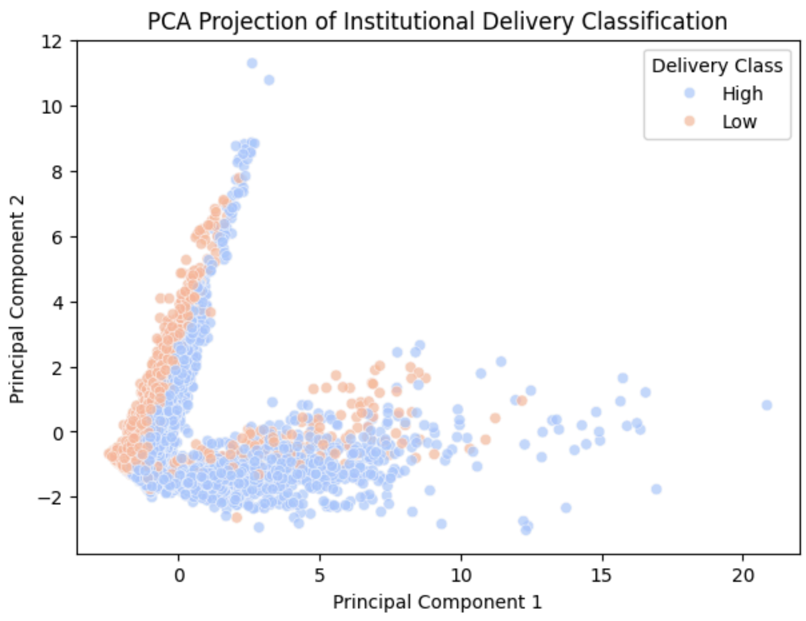


```
#Distribution of Target Classes
#Shows how many districts are in the High vs Low Institutional Delivery category.

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6,4))
sns.countplot(x=y, palette="viridis")
plt.title("Distribution of Institutional Delivery Classes")
plt.xlabel("Delivery Class")
plt.ylabel("Number of Districts")
plt.show()
```



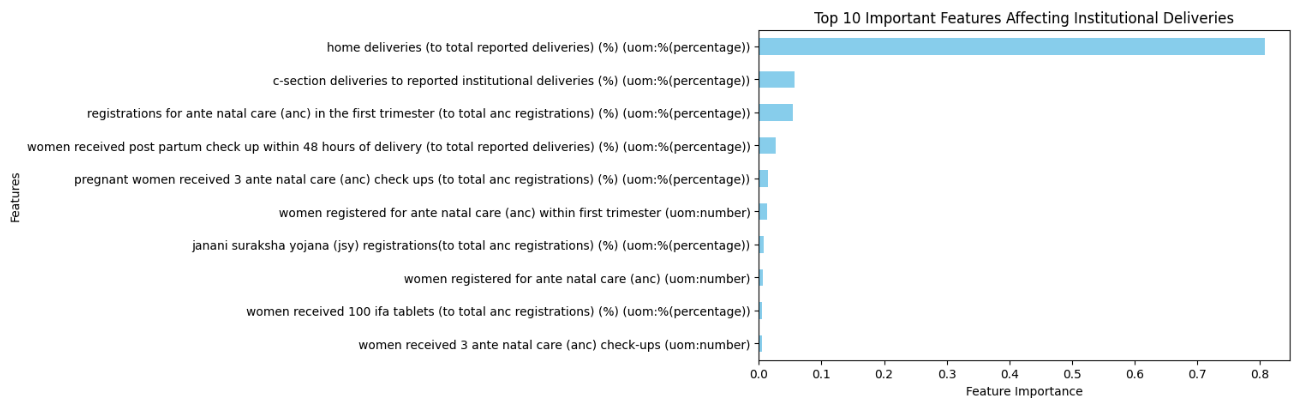
Our dataset has X high-performing and Y low-performing districts, ensuring balanced classification."

```
#Decision Boundary Visualization (Simplified Example)
from sklearn.decomposition import PCA

# Reduce to 2 dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)

plt.figure(figsize=(7,5))
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y_train, palette="coolwarm", alpha=0.7)
plt.title("PCA Projection of Institutional Delivery Classification")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Delivery Class")
plt.show()
```



This visually shows how the classifier distinguishes between "High" and "Low" institutional deliveries based on combined features.

```
#Feature Importance Bar Chart (from Random Forest)
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
top_features = feat_importances.sort_values(ascending=True).tail(10)

plt.figure(figsize=(8,5))
top_features.plot(kind='barh', color='skyblue')
plt.title("Top 10 Important Features Affecting Institutional Deliveries")
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```

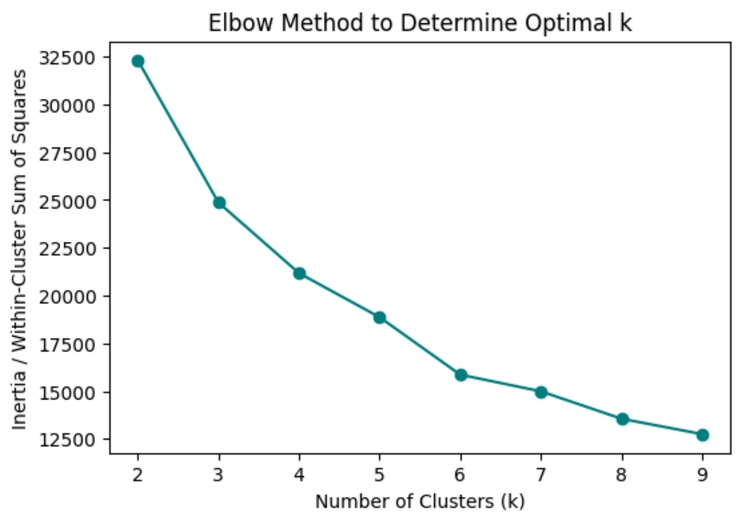


CLUSTERING

```
#Import & Prepare Data
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
```

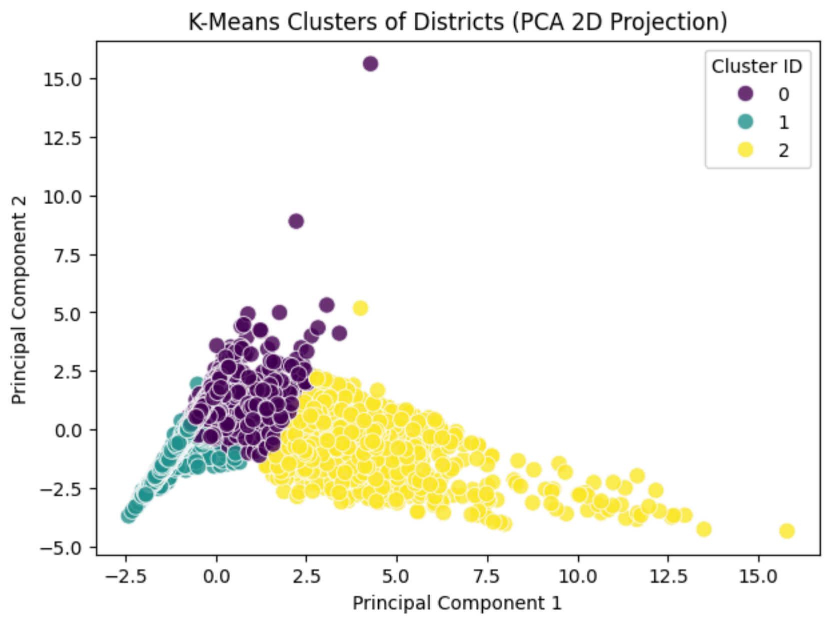
```
#Select Features for Clustering
features = [
    "pregnant women received 3 ante natal care (anc) check ups (to total anc registrations) (%) (uom:%(percentage)
    "women received 100 ifa tablets (to total anc registrations) (%) (uom:%(percentage))",
    "home deliveries (to total reported deliveries) (%) (uom:%(percentage))",
    "c-section deliveries to reported institutional deliveries (%) (uom:%(percentage))",
    "women received post partum check up within 48 hours of delivery (to total reported deliveries) (%) (uom:%(perc
    "janani suraksha yojana (jsy) registrations(to total anc registrations) (%) (uom:%(percentage))"
]

X = df[features].copy()
X = X.replace(["NA", "", " "], np.nan)
X = X.astype(float)
X.fillna(X.median(), inplace=True)

# Scale features for fair clustering
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
#Find Optimal Number of Clusters (Elbow Method)
inertia = []
K = range(2, 10)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(6,4))
plt.plot(K, inertia, marker='o', color='teal')
plt.title("Elbow Method to Determine Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia / Within-Cluster Sum of Squares")
plt.show()
```



```
#Apply K-Means Clustering
# Suppose elbow curve suggested k=3
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

df["Cluster"] = clusters
```

```
#🎨 STEP 5: Visualize Clusters in 2D (PCA)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(7,5))
sns.scatterplot(
    x=X_pca[:,0],
    y=X_pca[:,1],
    hue=df["Cluster"],
    palette="viridis",
    s=80,
    alpha=0.8
)
plt.title("K-Means Clusters of Districts (PCA 2D Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster ID")
plt.show()
```



```
#Cluster-Wise Summary
cluster_summary = df.groupby("Cluster")[features].mean().round(2)
print(cluster_summary)
```

```
         pregnant women received 3 ante natal care (anc) check ups (to total anc registrations) (%) (uom:%(percentag
Cluster
0                                             85.87
1                                             63.14
2                                             89.14

         women received 100 ifa tablets (to total anc registrations) (%) (uom:%(percentage))  \
Cluster
0                                           2865.39
1                                            289.66
2                                          46880.50

         home deliveries (to total reported deliveries) (%) (uom:%(percentage))  \
Cluster
0                                              5.76
1                                             32.05
2                                              6.40

         c-section deliveries to reported institutional deliveries (%) (uom:%(percentage))  \
Cluster
0                                             19.51
1                                              5.10
2                                             18.85

         women received post partum check up within 48 hours of delivery (to total reported deliveries) (%) (uom:%(
Cluster
0                                           2901.53
1                                            284.45
2                                          49294.32

         janani suraksha yojana (jsy) registrations(to total anc registrations) (%) (uom:%(percentage))
Cluster
0                                           2688.60
1                                            235.32
2                                          43542.82
```

```
#Cluster Character Heatmap (for presentation)
plt.figure(figsize=(10,6))
sns.heatmap(cluster_summary, cmap="YlGnBu", annot=True)
plt.title("Average Health Indicators per Cluster")
plt.xlabel("Health Indicators")
plt.ylabel("Cluster Group")
plt.show()
```