de-mini-project-india-2

October 14, 2024

DE Mini Project By:-Hrishit Madhavi

[ ]:
```
# Import necessary libraries
import pandas as pd
import numpy as np

# Load the dataset
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving india.csv to india.csv
```

Data Cleaning

[ ]:
```
#Data Cleaning
data = pd.read_csv('india.csv')

# Step 3: Display dataset info and check missing values (null values)
print("Dataset Info:")
print(data.info())

# Display null values per column
print("\nMissing Values Count Before Cleaning:")
print(data.isna().sum())

# Step 4: Display rows containing null values
print("\nRows with Missing Values:")
print(data[data.isna().any(axis=1)])

# Step 5: Data Cleaning - Fill missing values using forward fill method
data.fillna(method='ffill', inplace=True)

# Step 6: Check and confirm that missing values are filled
print("\nMissing Values Count After Cleaning:")
print(data.isna().sum())

# Optional: Display rows that originally had missing values to confirm filling
```

```
print("\nData After Filling Missing Values:")
print(data[data.isna().any(axis=1)])
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 35 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   state     607 non-null    object
 1   1990      235 non-null    float64
 2   1991      235 non-null    float64
 3   1992      226 non-null    float64
 4   1993      279 non-null    float64
 5   1994      279 non-null    float64
 6   1995      279 non-null    float64
 7   1996      279 non-null    float64
 8   1997      279 non-null    float64
 9   1998      279 non-null    float64
 10  1999      279 non-null    float64
 11  2000      288 non-null    float64
 12  2001      288 non-null    float64
 13  2002      288 non-null    float64
 14  2003      288 non-null    float64
 15  2004      577 non-null    float64
 16  2005      596 non-null    float64
 17  2006      596 non-null    float64
 18  2007      596 non-null    float64
 19  2008      596 non-null    float64
 20  2009      596 non-null    float64
 21  2010      596 non-null    float64
 22  2011      596 non-null    float64
 23  2012      596 non-null    float64
 24  2013      596 non-null    float64
 25  2014      607 non-null    float64
 26  2015      607 non-null    float64
 27  2016      607 non-null    float64
 28  2017      607 non-null    float64
 29  2018      607 non-null    float64
 30  2019      607 non-null    float64
 31  2020      607 non-null    float64
 32  2021      598 non-null    float64
 33  2022      190 non-null    float64
 34  CATEGORY  607 non-null    object
dtypes: float64(33), object(2)
memory usage: 166.1+ KB
None
```

```
Missing Values Count Before Cleaning:
state         0
1990        372
1991        372
1992        381
1993        328
1994        328
1995        328
1996        328
1997        328
1998        328
1999        328
2000        319
2001        319
2002        319
2003        319
2004         30
2005         11
2006         11
2007         11
2008         11
2009         11
2010         11
2011         11
2012         11
2013         11
2014          0
2015          0
2016          0
2017          0
2018          0
2019          0
2020          0
2021          9
2022        417
CATEGORY      0
dtype: int64

Rows with Missing Values:
                        state    1990    1991    1992     1993     1994  \
0    Andaman & Nicobar Island  2580.0  2302.0  2884.0  15192.0  16191.0
2            Arunachal Pradesh  2709.0  3013.0  3015.0   8733.0   8342.0
3                        Assam  1544.0  1575.0  1557.0   5715.0   5737.0
5                   Chandigarh     NaN     NaN     NaN  19761.0  21021.0
6                 Chhattisgarh     NaN     NaN     NaN   6539.0   6445.0
..                         …     …       …       …       …        …
602             Uttar Pradesh     NaN     NaN     NaN      NaN      NaN
```

```
603              Uttarakhand     NaN     NaN     NaN      NaN      NaN
604              West Bengal     NaN     NaN     NaN      NaN      NaN
605                    Delhi     NaN     NaN     NaN      NaN      NaN
606               Puducherry     NaN     NaN     NaN      NaN      NaN

        1995     1996     1997     1998  …      2014      2015      2016  \
0    15354.0  15896.0  16350.0  14502.0  …   98735.0  106711.0  114660.0
2     9352.0   8590.0   8634.0   8712.0  …   79004.0   91034.0   88768.0
3     5760.0   5793.0   5796.0   5664.0  …   43002.0   44809.0   50642.0
5    22524.0  24855.0  25470.0  26718.0  …  180615.0  182867.0  195205.0
6     6474.0   6654.0   6810.0   6873.0  …   61409.0   61122.0   61433.0
..       …        …        …        …  …         …         …
602      NaN      NaN      NaN      NaN  …   86322.0  108196.0  129756.0
603      NaN      NaN      NaN      NaN  …   12994.0   13402.0   14509.0
604      NaN      NaN      NaN      NaN  …   57264.0   67837.0   74698.0
605      NaN      NaN      NaN      NaN  …   16061.0   17636.0   19608.0
606      NaN      NaN      NaN      NaN  …    2260.0    2465.0    2359.0

         2017      2018      2019      2020      2021  2022  \
0    129532.0  145562.0  154233.0  161564.0       NaN   NaN
2     91319.0   94008.0   99580.0  113110.0  108706.0   NaN
3     53575.0   57835.0   59943.0   61519.0   57227.0   NaN
5    208231.0  218201.0  227231.0  234350.0  203180.0   NaN
6     67139.0   68374.0   72537.0   75278.0   72236.0   NaN
..        …         …         …         …         …   …
602  117089.0  134473.0  142288.0  159679.0  211661.0   NaN
603   15423.0   17508.0   17475.0   21785.0   24293.0   NaN
604   87845.0   94698.0   96429.0  111635.0  141751.0   NaN
605   22285.0   25264.0   27357.0   32563.0   37925.0   NaN
606    2499.0    2748.0    2821.0    3162.0    3381.0   NaN

                     CATEGORY
0            Per Capita Income
2            Per Capita Income
3            Per Capita Income
5            Per Capita Income
6            Per Capita Income
..                         …
602  Social Sector Expenditure
603  Social Sector Expenditure
604  Social Sector Expenditure
605  Social Sector Expenditure
606  Social Sector Expenditure

[460 rows x 35 columns]

Missing Values Count After Cleaning:
state       0
```

```
1990        0
1991        0
1992        0
1993        0
1994        0
1995        0
1996        0
1997        0
1998        0
1999        0
2000        0
2001        0
2002        0
2003        0
2004        0
2005        0
2006        0
2007        0
2008        0
2009        0
2010        0
2011        0
2012        0
2013        0
2014        0
2015        0
2016        0
2017        0
2018        0
2019        0
2020        0
2021        1
2022        1
CATEGORY    0
dtype: int64

Data After Filling Missing Values:
                      state    1990    1991    1992     1993     1994  \
0  Andaman & Nicobar Island  2580.0  2302.0  2884.0  15192.0  16191.0

      1995     1996     1997     1998  …     2014      2015      2016  \
0  15354.0  15896.0  16350.0  14502.0  …  98735.0  106711.0  114660.0

       2017      2018      2019      2020  2021  2022           CATEGORY
0  129532.0  145562.0  154233.0  161564.0   NaN   NaN  Per Capita Income

[1 rows x 35 columns]
```

```
<ipython-input-5-dd8d10f7f70e>:17: FutureWarning: DataFrame.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  data.fillna(method='ffill', inplace=True)
```

Data Integration

[ ]:
```
# Data Integration: Dropping non-numeric columns like 'state' and 'CATEGORY'
df = data.drop(columns=['state', 'CATEGORY'])

# Checking if the integration was successful
print("Data after dropping non-numeric columns:")
print(df.head())
```

```
Data after dropping non-numeric columns:
     1990    1991    1992     1993     1994     1995     1996     1997  \
0  2580.0  2302.0  2884.0  15192.0  16191.0  15354.0  15896.0  16350.0
1  2060.0  2134.0  2039.0   7416.0   7711.0   8071.0   8514.0   8191.0
2  2709.0  3013.0  3015.0   8733.0   8342.0   9352.0   8590.0   8634.0
3  1544.0  1575.0  1557.0   5715.0   5737.0   5760.0   5793.0   5796.0
4  1197.0  1105.0  1017.0   3037.0   3306.0   2728.0   3338.0   3100.0

      1998     1999  …     2013     2014      2015      2016      2017  \
0  14502.0  15293.0  …  92644.0  98735.0  106711.0  114660.0  129532.0
1   9144.0   9445.0  …  68865.0  72254.0   79174.0   88609.0   94115.0
2   8712.0   8890.0  …  73960.0  79004.0   91034.0   88768.0   91319.0
3   5664.0   5785.0  …  41609.0  43002.0   44809.0   50642.0   53575.0
4   3210.0   3282.0  …  22201.0  22776.0   23223.0   24064.0   25455.0

       2018      2019      2020      2021      2022
0  145562.0  154233.0  161564.0       NaN       NaN
1  103177.0  108853.0  115344.0  114324.0  126587.0
2   94008.0   99580.0  113110.0  108706.0  126587.0
3   57835.0   59943.0   61519.0   57227.0  126587.0
4   26719.0   29092.0   29794.0   28127.0   30779.0

[5 rows x 33 columns]
```

Data Transformation

[ ]:
```
# Data Transformation: Convert the dataset into a usable numeric format␣
 ↪(already done by dropping non-numeric columns)

# Checking summary statistics to verify transformation
print("Summary statistics of transformed data:")
print(df.describe())

# You can also check for further transformation like normalization or scaling␣
 ↪if needed (Optional)
```

Summary statistics of transformed data:

	1990	1991	1992	1993	1994
count	607.000000	607.000000	607.000000	607.000000	607.000000
mean	4545.615783	4686.730774	4963.434662	15024.110264	16106.061977
std	3714.012024	3709.881807	3917.894233	12180.441100	12995.414035
min	2.090000	2.300000	2.480000	5.390000	5.990000
25%	1285.370000	1305.000000	1440.520000	3187.790000	3502.685000
50%	5983.480000	6313.900000	6610.920000	20711.660000	22192.690000
75%	5983.480000	6313.900000	6610.920000	20711.660000	22192.690000
max	29899.270000	29986.090000	34281.560000	113319.640000	116213.270000

	1995	1996	1997	1998
count	607.000000	607.000000	607.000000	607.000000
mean	17419.110313	18913.467298	20502.519077	22824.611005
std	14066.736411	15176.339163	16202.047241	17722.661631
min	7.370000	9.080000	10.030000	10.660000
25%	3791.785000	3957.225000	4683.720000	5074.120000
50%	24310.680000	26531.600000	29065.710000	33019.560000
75%	24310.680000	26531.600000	29065.710000	33019.560000
max	129566.830000	136148.960000	143723.220000	148548.330000

	1999	…	2013	2014	2015
count	607.000000	…	6.070000e+02	6.070000e+02	6.070000e+02
mean	24677.069572	…	5.360240e+04	5.692582e+04	6.080881e+04
std	19207.975514	…	1.293404e+05	1.375710e+05	1.459789e+05
min	12.490000	…	-1.006700e+04	-2.239400e+04	-1.434000e+04
25%	5223.525000	…	1.898520e+03	2.409500e+03	2.623000e+03
50%	35997.920000	…	9.970330e+03	1.117800e+04	1.306300e+04
75%	35997.920000	…	4.243444e+04	4.419770e+04	4.971770e+04
max	163022.870000	…	1.357942e+06	1.451615e+06	1.543165e+06

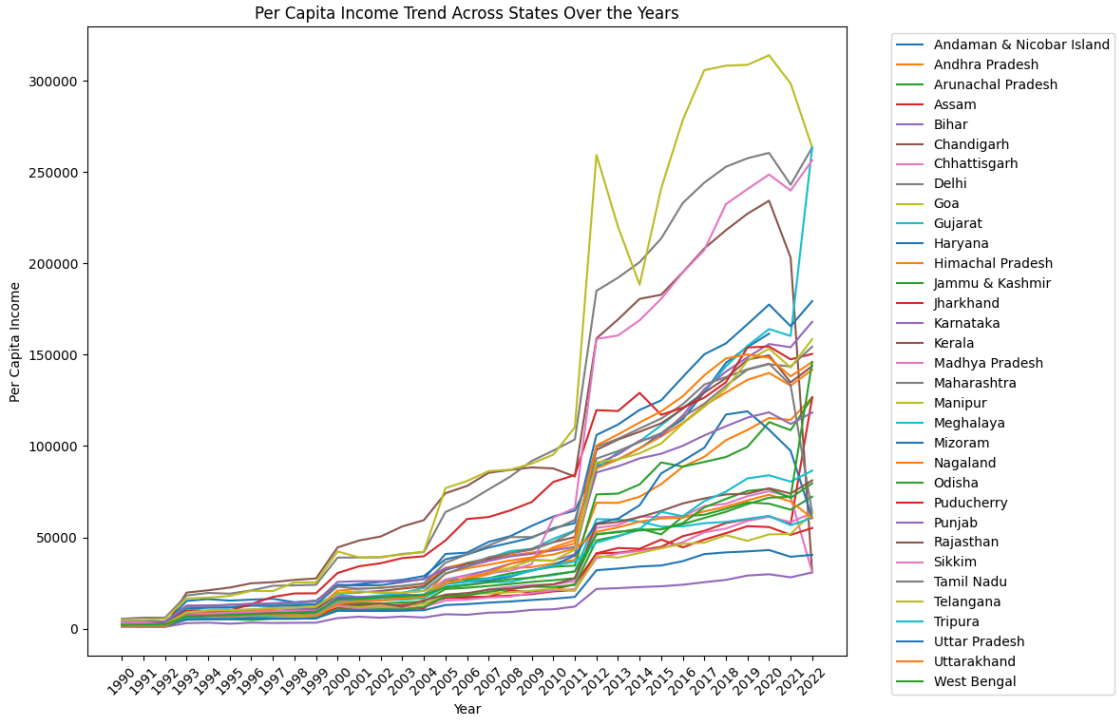
	2016	2017	2018	2019	2020
count	6.070000e+02	6.070000e+02	6.070000e+02	6.070000e+02	6.070000e+02
mean	6.659086e+04	7.193034e+04	7.705965e+04	8.181138e+04	8.726165e+04
std	1.586020e+05	1.736718e+05	1.846451e+05	1.953316e+05	2.034943e+05
min	-2.028300e+04	-1.482300e+04	-2.825000e+04	-6.756000e+04	-1.341800e+04
25%	2.749000e+03	2.638500e+03	3.161560e+03	3.217095e+03	5.029000e+03
50%	1.503250e+04	1.539727e+04	1.643983e+04	1.766600e+04	2.199730e+04
75%	5.290513e+04	5.421721e+04	6.020550e+04	6.242465e+04	6.793846e+04
max	1.654284e+06	1.807046e+06	1.888706e+06	1.972960e+06	2.043982e+06

	2021	2022
count	6.060000e+02	6.060000e+02
mean	8.849832e+04	1.109285e+05
std	1.989462e+05	1.705305e+05
min	-2.833800e+04	6.592000e+01
25%	5.481000e+03	6.208428e+04
50%	2.313128e+04	6.208428e+04

```
75%    7.123035e+04  6.208428e+04
max    1.889307e+06  1.345108e+06

[8 rows x 33 columns]
```

Graph 1:- Per Capita Income over the years for all states(Trend Graph)

[ ]:
```python
# Filter the data specifically for "Per Capita Income"
# Assuming 'CATEGORY' is a column in your 'data' DataFrame
filtered_data = data[data['CATEGORY'] == 'Per Capita Income']

# Drop unnecessary columns like 'CATEGORY' to focus on numeric data for plotting
per_capita_data = filtered_data.drop(columns=['CATEGORY'])

# Transpose the data to have states as rows and years as columns for plotting
per_capita_data.set_index('state', inplace=True)
per_capita_data = per_capita_data.T

# Plotting per capita income trend for all states
plt.figure(figsize=(12, 8))
for state in per_capita_data.columns:
    plt.plot(per_capita_data.index, per_capita_data[state], label=state)

plt.title('Per Capita Income Trend Across States Over the Years')
plt.xlabel('Year')
plt.ylabel('Per Capita Income')
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



Graph 2:- Highest Per Capita Income for the Year 2022 (SunBurst graph)

[ ]:
```
import pandas as pd
import plotly.express as px

# Manually create a DataFrame with per capita income values for 2022
data = pd.DataFrame({
    'state': [
        'Goa', 'Sikkim', 'Delhi', 'Haryana', 'Karnataka', 'Telangana',
        'Kerala', 'Gujarat', 'Tamil Nadu', 'Uttarakhand', 'Maharashtra',
        'Himachal Pradesh', 'Punjab', 'Andhra Pradesh', 'Rajasthan',
        'Mizoram', 'Odisha', 'Chhattisgarh', 'Madhya Pradesh', 'Jammu &␣
 ↪Kashmir',
        'Tripura', 'Meghalaya', 'Uttar Pradesh', 'Assam', 'Manipur', 'Bihar',
        'Arunachal Pradesh', 'Jharkhand', 'Nagaland', 'West Bengal',
        'Puducherry', 'Andaman & Nicobar Islands', 'Chandigarh', 'Lakshadweep',
        'Dadra and Nagar Haveli and Daman and Diu', 'Ladakh'
    ],
    'per_capita_income_2022': [
        500309, 256507, 263477, 179367, 168050, 158561,
        143816, 161016, 154427, 146047, 141913, 141830,
        118341, 126587, 81231, 80659, 79607, 76804,
        63345, 72287, 86539, 60606, 40432, 39434,
        36474, 30779, 143373, 55126, 92470, 115748,
```

```
        150454, 161564, 234350, 102430, 176447, 72000
    ]
})

# Add a column for region (you can adjust these as needed)
data['region'] = 'India'

# Create the sunburst chart
fig = px.sunburst(
    data,
    path=['region', 'state'],
    values='per_capita_income_2022',
    title='Per Capita Income of Indian States and UTs (2022)',
    color='per_capita_income_2022',
    color_continuous_scale='Viridis',
    hover_data={'per_capita_income_2022': ':,.0f'}
)

# Update layout
fig.update_layout(
    title_font_size=24,
    font_size=12,
    title_x=0.5,  # Center the title
)

# Show the figure
fig.show()
```

Graph 3:- Highest tax revenue for year 2021 (TreeMap)

[ ]:
```
import pandas as pd
import plotly.express as px

# List of all Indian states and union territories with their 2021 tax revenue␣
 ↪data
data = pd.DataFrame({
    'state': [
        'Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh',
        'Goa', 'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jammu and Kashmir',
        'Jharkhand', 'Karnataka', 'Kerala', 'Madhya Pradesh', 'Maharashtra',
        'Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha',
        'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana',
        'Tripura', 'Uttar Pradesh', 'Uttarakhand', 'West Bengal',
        'Delhi', 'Puducherry'
    ],
    'tax_revenue': [
        85265, 1900, 21178, 35050, 25750,
```

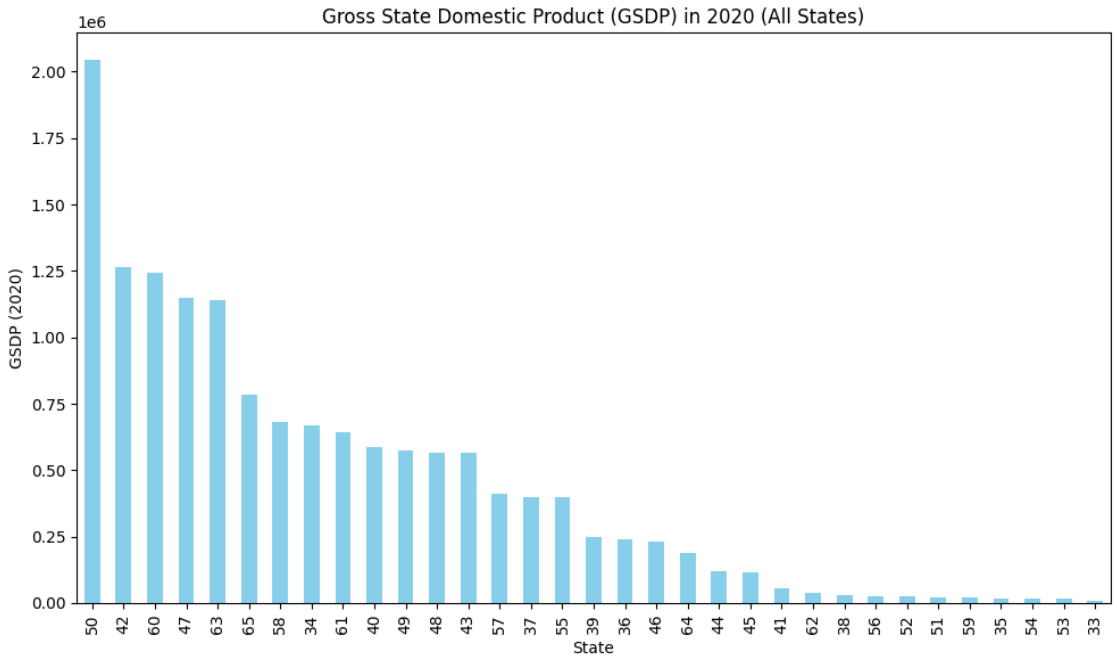
```
        5473, 111693, 52887, 9282, 16276,
        23256, 111494, 71833, 64914, 243490,
        2055, 2579, 720, 1272, 37500,
        37434, 90050, 1195, 126644, 92910,
        2412, 186345, 12754, 75416,
        43000, 2639
    ]
})

# Add a parent column for all states
data['parent'] = 'India'

# Create the treemap
fig = px.treemap(
    data,
    path=['parent', 'state'],
    values='tax_revenue',
    title='Tax Revenue of Indian States and Union Territories (2021)',
    color='tax_revenue',
    color_continuous_scale='Viridis',
    hover_data={'tax_revenue': ':,.0f'},
)

# Update layout
fig.update_layout(
    title_font_size=24,
    font_size=12,
    title_x=0.5,  # Center the title
)

# Update traces
fig.update_traces(
    textinfo="label+value",
    textfont_size=12,
)

# Show the figure
fig.show()
```

[ ]:
```
def prepare_data_for_plotting(data, category):
    """Prepares data for plotting by filtering by category and transposing."""
    # Filter data by category - This is the fix for the error!
    category_data = data[data['CATEGORY'] == category]
    return category_data.drop(columns=['CATEGORY']).T

# Prepare data for each plot type
pci_data = prepare_data_for_plotting(data, 'Per Capita Income')
```

```
gdp_data = prepare_data_for_plotting(data, 'Gross State Domestic Product')
nsdp_data = prepare_data_for_plotting(data, 'Net State Domestic Product')
fiscal_deficit_data = prepare_data_for_plotting(data, 'Gross Fiscal Deficit')
revenue_deficit_data = prepare_data_for_plotting(data, 'Revenue Deficit')
```

Graph 4:- Gross State Domestic Product for all state in 2020 (Bar Graph)

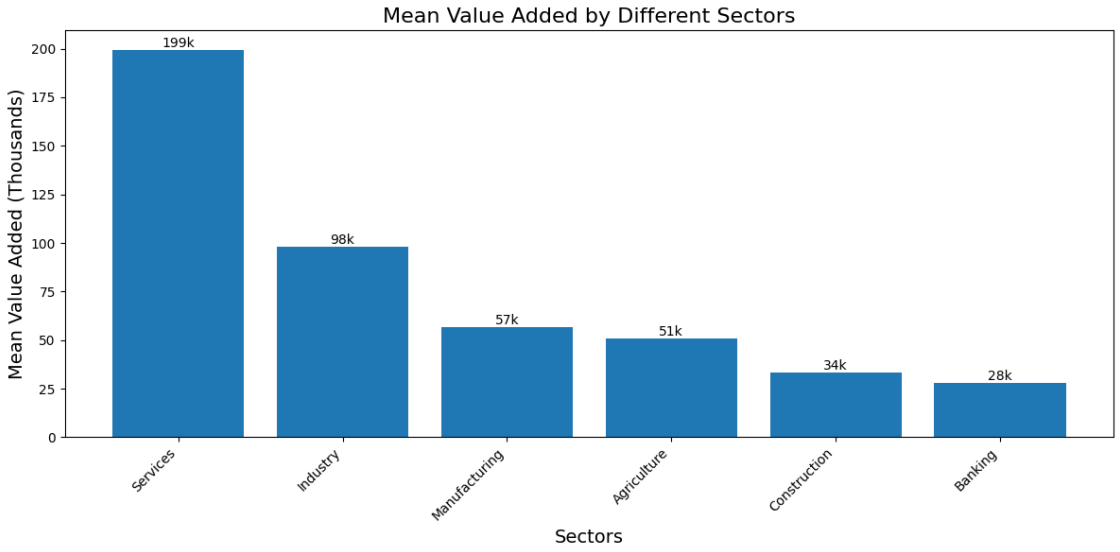
[ ]:
```
plt.figure(figsize=(10, 6))
gdp_2020 = gdp_data.loc['2020']
gdp_2020.sort_values(ascending=False).plot(kind='bar', color='skyblue')
plt.title('Gross State Domestic Product (GSDP) in 2020 (All States)')
plt.xlabel('State')
plt.ylabel('GSDP (2020)')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Graph 5:-Which Sectors gives more contribution over the years

[ ]:
```
import matplotlib.pyplot as plt

# Data
sectors = ['Services', 'Industry', 'Manufacturing', 'Agriculture',␣
 ↪'Construction', 'Banking']
means = [199408.73, 98159.12, 56831.25, 50625.78, 33578.21, 27991.56]
```

```
# Create bar plot
plt.figure(figsize=(12, 6))
plt.bar(sectors, means)

# Customize the plot
plt.title('Mean Value Added by Different Sectors', fontsize=16)
plt.xlabel('Sectors', fontsize=14)
plt.ylabel('Mean Value Added', fontsize=14)
plt.xticks(rotation=45, ha='right')

# Format y-axis to display values in thousands
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: format(int(x/
 ↪1000), ',')))
plt.ylabel('Mean Value Added (Thousands)', fontsize=14)

# Add value labels on top of each bar
for i, v in enumerate(means):
    plt.text(i, v, f'{v/1000:.0f}k', ha='center', va='bottom')

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

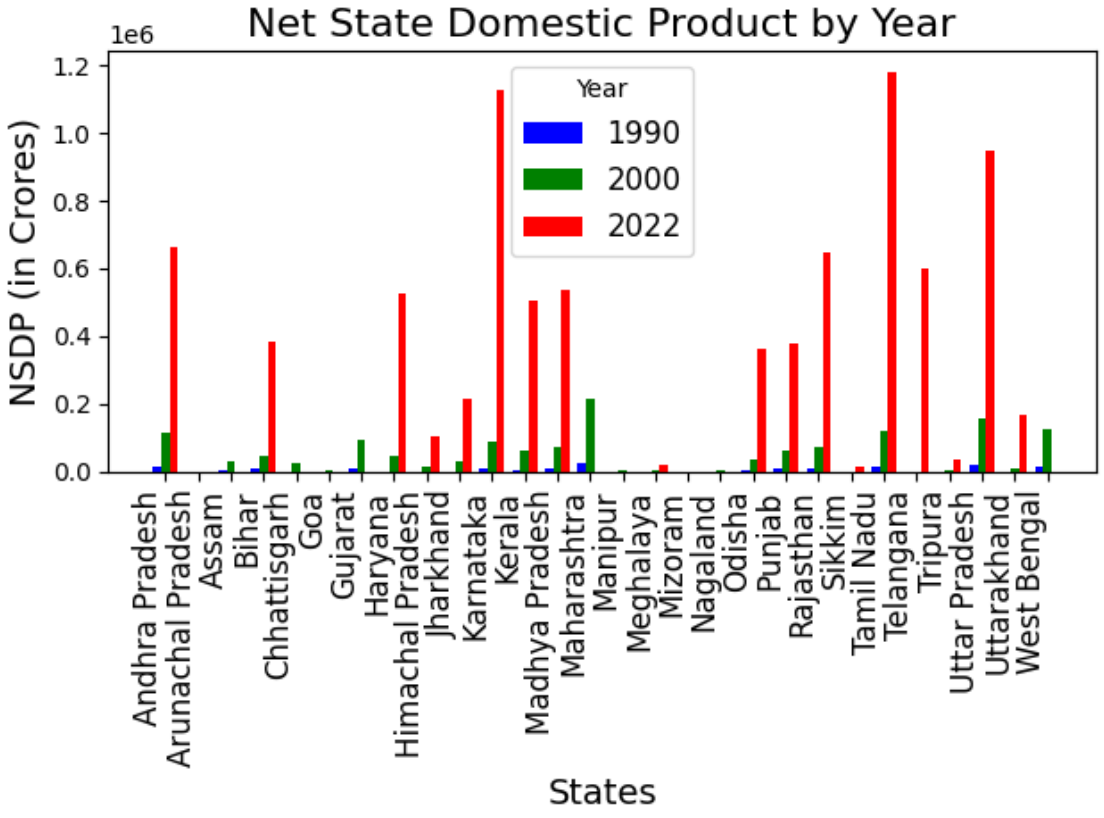


Graph 6:- Net State Domestic Product over the Decade for all state(MultiBar Graph)

```
[ ]: import matplotlib.pyplot as plt
     import pandas as pd
     import numpy as np

     # Data (replace with actual data from your dataset)
```

```
data = {
    'State': ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',␣
 ↪'Chhattisgarh',
              'Goa', 'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jharkhand',
              'Karnataka', 'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur',
              'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha', 'Punjab',
              'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Tripura',
              'Uttar Pradesh', 'Uttarakhand', 'West Bengal'],
    '1990': [13580.12, 231.11, 3425.55, 10253.27, 0, 568.25, 10839.15, 0, 1150.
 ↪8, 0,
             9112.1, 5262.34, 11107.21, 27223.82, 316.22, 303.99, 0, 238.1,␣
 ↪4344.7, 7504.93,
             8472.6, 135.32, 12422.99, 0, 446.86, 22779.65, 0, 14457.81],
    '2000': [116360.25, 1496.96, 32010.66, 46070.78, 23839.65, 5570.43, 92541,␣
 ↪47329.37, 12467, 30228.93,
             90531.92, 61359.33, 72655.36, 217197.93, 2954.11, 3211.3, 1409.51,␣
 ↪2609.8, 38398.89, 61139.23,
             74173.85, 765.37, 119703.94, 0, 4495.57, 156809.34, 11187.19,␣
 ↪125298.82],
    '2022': [661432.01, 0, 0, 382274.49, 0, 0, 0, 527733.12, 106003.79, 213680.
 ↪99,
             1127480.45, 506107.34, 539180.46, 0, 0, 22527.35, 0, 0, 365115.7,␣
 ↪376780.82,
             648141.63, 17442.45, 1181922.77, 599614.01, 35394.6, 948838.57,␣
 ↪167486.57, 0]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Replace zeros with NaN for better representation of missing data
df.replace(0, np.nan, inplace=True)

# Function to create grouped bar chart
def create_grouped_bar_chart():
    bar_width = 0.25  # Width of each bar
    index = np.arange(len(df['State']))  # The label locations

    # Create the bars for each year
    plt.bar(index, df['1990'], bar_width, label='1990', color='blue')
    plt.bar(index + bar_width, df['2000'], bar_width, label='2000',␣
 ↪color='green')
    plt.bar(index + 2 * bar_width, df['2022'], bar_width, label='2022',␣
 ↪color='red')

    # Add labels, title, and legend
```

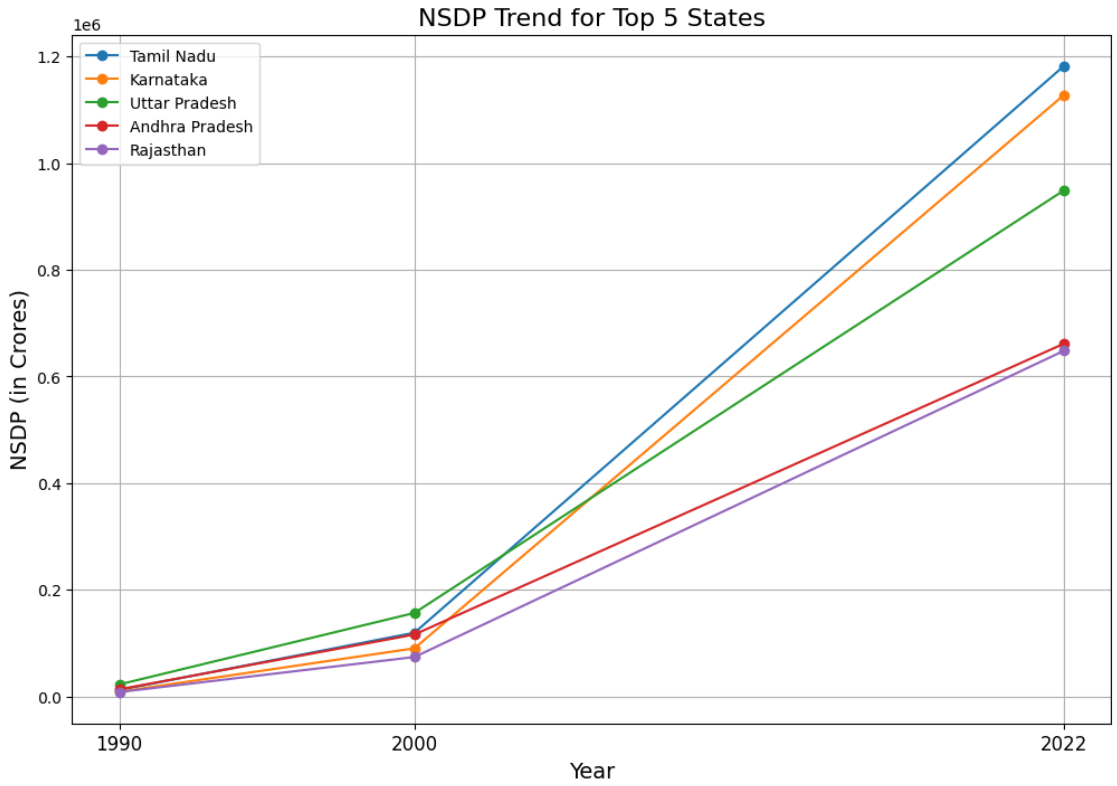
```
    plt.title('Net State Domestic Product by Year', fontsize=16)
    plt.xlabel('States', fontsize=14)
    plt.ylabel('NSDP (in Crores)', fontsize=14)
    plt.xticks(index + bar_width, df['State'], rotation=90, ha='right',␣
 ↪fontsize=12)

    # Add legend
    plt.legend(title='Year', fontsize=12)

    # Adjust layout
    plt.tight_layout()

    # Show plot
    plt.show()

# Create the grouped bar chart
create_grouped_bar_chart()
```



Graph 7:- NSDP of Top 5 state

[ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Data (replace with actual data from your dataset)
data = {
    'State': ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',␣
 ↪'Chhattisgarh',
              'Goa', 'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jharkhand',
              'Karnataka', 'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur',
              'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha', 'Punjab',
              'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Tripura',
              'Uttar Pradesh', 'Uttarakhand', 'West Bengal'],
    '1990': [13580.12, 231.11, 3425.55, 10253.27, 0, 568.25, 10839.15, 0, 1150.
 ↪8, 0,
             9112.1, 5262.34, 11107.21, 27223.82, 316.22, 303.99, 0, 238.1,␣
 ↪4344.7, 7504.93,
             8472.6, 135.32, 12422.99, 0, 446.86, 22779.65, 0, 14457.81],
    '2000': [116360.25, 1496.96, 32010.66, 46070.78, 23839.65, 5570.43, 92541,␣
 ↪47329.37, 12467, 30228.93,
             90531.92, 61359.33, 72655.36, 217197.93, 2954.11, 3211.3, 1409.51,␣
 ↪2609.8, 38398.89, 61139.23,
             74173.85, 765.37, 119703.94, 0, 4495.57, 156809.34, 11187.19,␣
 ↪125298.82],
    '2022': [661432.01, 0, 0, 382274.49, 0, 0, 0, 527733.12, 106003.79, 213680.
 ↪99,
             1127480.45, 506107.34, 539180.46, 0, 0, 22527.35, 0, 0, 365115.7,␣
 ↪376780.82,
             648141.63, 17442.45, 1181922.77, 599614.01, 35394.6, 948838.57,␣
 ↪167486.57, 0]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Replace zeros with NaN for better representation of missing data
df.replace(0, np.nan, inplace=True)

# 2. To create a line plot showing the trend for top 5 states:
# Calculate the total NSDP for each state across all years
state_totals = df.set_index('State').sum(axis=1)

# Get the top 5 states based on total NSDP
top_5_states = state_totals.nlargest(5).index

plt.figure(figsize=(12, 8))
```

```
for state in top_5_states:
    # Select data for the current state
    state_data = df[df['State'] == state].iloc[:, 1:].values[0]

    # Plot the data with markers and label
    plt.plot([1990, 2000, 2022], state_data, marker='o', label=state)

plt.title('NSDP Trend for Top 5 States', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('NSDP (in Crores)', fontsize=14)
plt.xticks([1990, 2000, 2022], fontsize=12)  # Set x-tick labels
plt.legend()
plt.grid(True)
plt.show()
```

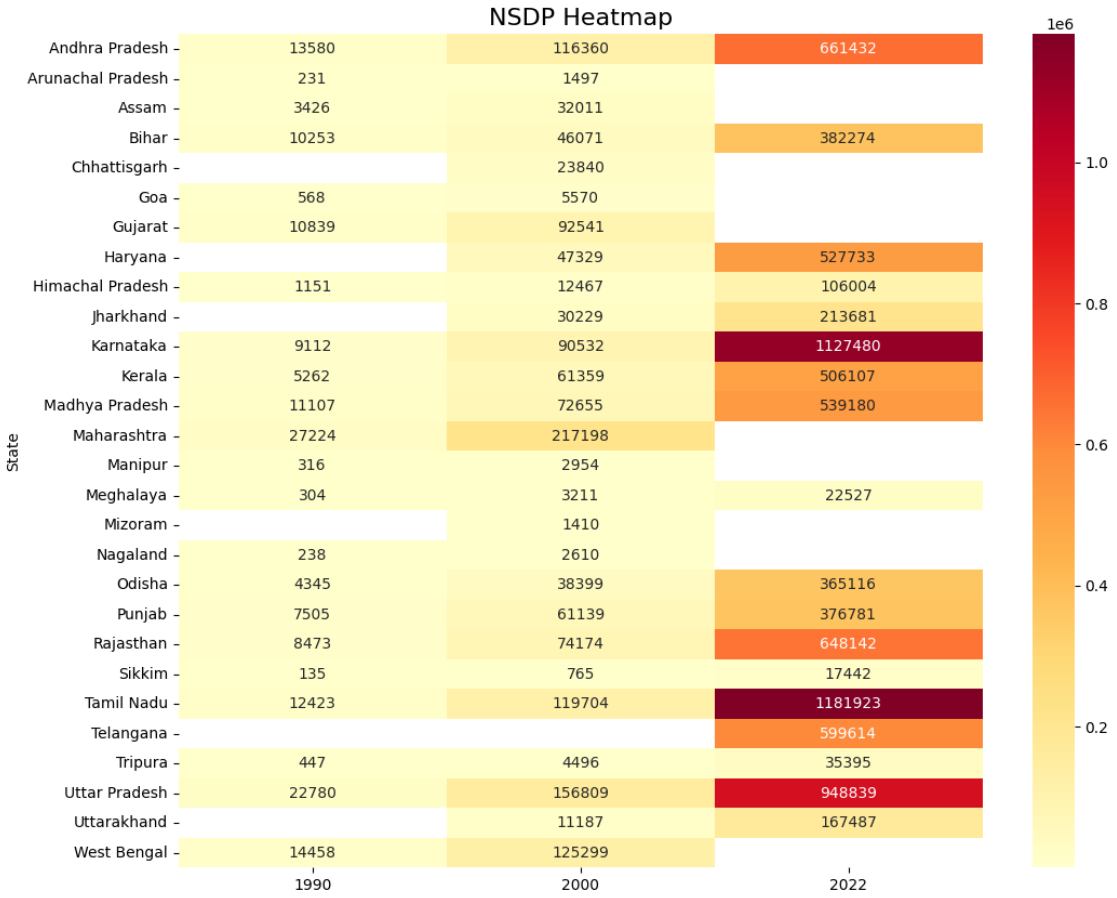


Graph 8:- Net State Domestic Product Heat Map

[ ]:
```
df['Growth_Rate_1990_2000'] = (df['2000'] - df['1990']) / df['1990'] * 100
print(df[['State', 'Growth_Rate_1990_2000']].
  ↪sort_values('Growth_Rate_1990_2000', ascending=False))
```

```
# 4. To create a heatmap of the data:
import seaborn as sns

plt.figure(figsize=(12, 10))
sns.heatmap(df.set_index('State').iloc[:, :3], annot=True, fmt='.0f',␣
 ↪cmap='YlOrRd')
plt.title('NSDP Heatmap', fontsize=16)
plt.show()
```

	State	Growth_Rate_1990_2000
11	Kerala	1066.008468
17	Nagaland	996.094078
8	Himachal Pradesh	983.333333
15	Meghalaya	956.383434
24	Tripura	906.035447
10	Karnataka	893.535189
5	Goa	880.278047
22	Tamil Nadu	863.567869
2	Assam	834.467750
14	Manipur	834.194548
18	Odisha	783.809929
20	Rajasthan	775.455586
27	West Bengal	766.651450
0	Andhra Pradesh	756.842576
6	Gujarat	753.766209
19	Punjab	714.654234
13	Maharashtra	697.823120
25	Uttar Pradesh	588.374668
12	Madhya Pradesh	554.127904
1	Arunachal Pradesh	547.726191
21	Sikkim	465.600059
3	Bihar	349.327678
4	Chhattisgarh	NaN
7	Haryana	NaN
9	Jharkhand	NaN
16	Mizoram	NaN
23	Telangana	NaN
26	Uttarakhand	NaN



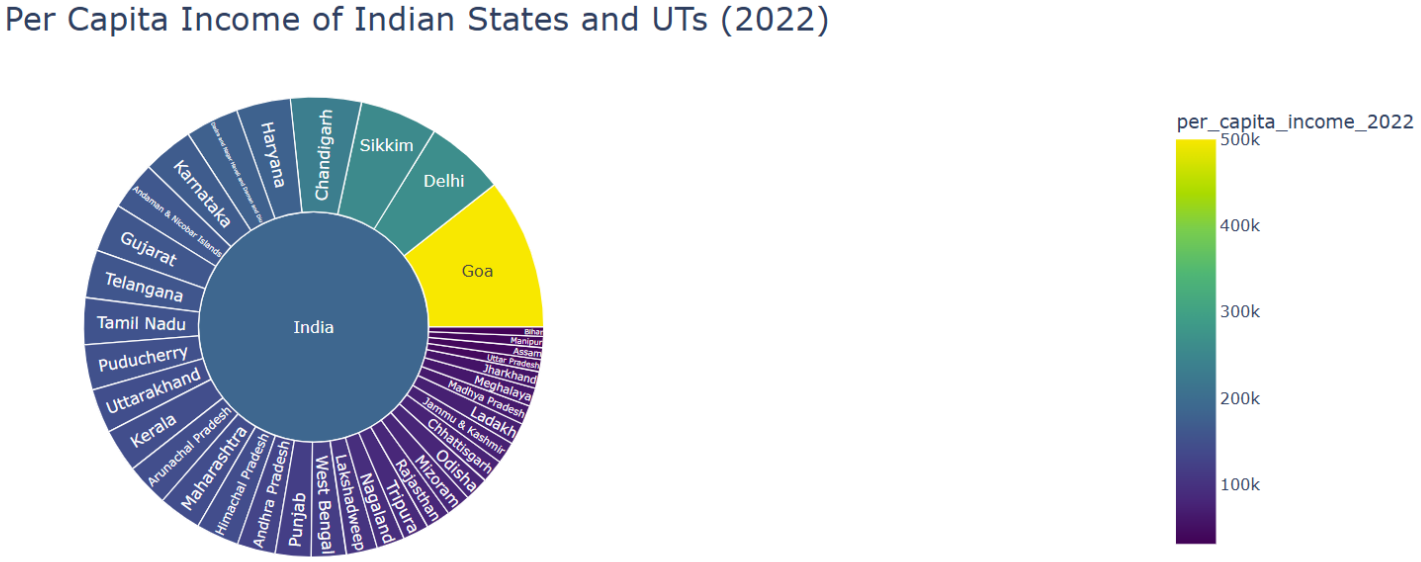
Applying Linear Regression algorithm on the dataset

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, f1_score
from sklearn.preprocessing import StandardScaler

# Load the data
df = pd.read_csv('imputed_india (1).csv')

# Prepare the data for per capita income prediction
years = df.columns[1:-2].astype(int).values.reshape(-1, 1)
per_capita = df.iloc[:, 1:-2].values

# Prepare the data for tax revenue prediction
tax_revenue = df[df['CATEGORY'] == 'Tax Revenue'].iloc[:, 1:-2].values
```

```python
def predict_and_evaluate(X, y, feature_name):
    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

    # Scale the features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Create and train the model
    model = LinearRegression()
    model.fit(X_train_scaled, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train_scaled)
    y_test_pred = model.predict(X_test_scaled)

    # Calculate accuracy metrics
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    # Calculate F1 score (for regression, we need to bin the values)
    y_train_bin = pd.cut(y_train, bins=2, labels=[0, 1])
    y_train_pred_bin = pd.cut(y_train_pred, bins=2, labels=[0, 1])
    y_test_bin = pd.cut(y_test, bins=2, labels=[0, 1])
    y_test_pred_bin = pd.cut(y_test_pred, bins=2, labels=[0, 1])

    train_f1 = f1_score(y_train_bin, y_train_pred_bin, average='weighted')
    test_f1 = f1_score(y_test_bin, y_test_pred_bin, average='weighted')

    print(f"\n{feature_name} Prediction Results:")
    print(f"Training Set R-squared: {train_r2:.4f}")
    print(f"Testing Set R-squared: {test_r2:.4f}")
    print(f"Training Set F1 Score: {train_f1:.4f}")
    print(f"Testing Set F1 Score: {test_f1:.4f}")

# Predict and evaluate per capita income
predict_and_evaluate(years, per_capita.mean(axis=0), 'Per Capita Income')

# Predict and evaluate tax revenue
predict_and_evaluate(years, tax_revenue.mean(axis=0), 'Tax Revenue')
```

```
Per Capita Income Prediction Results:
Training Set R-squared: 0.7479
Testing Set R-squared: 0.6314
```

```
Training Set F1 Score: 0.7518
Testing Set F1 Score: 1.0000

Tax Revenue Prediction Results:
Training Set R-squared: 0.0683
Testing Set R-squared: -0.0950
Training Set F1 Score: 0.4842
Testing Set F1 Score: 0.7024
```



```
import pandas as pd
import plotly.express as px

# List of all Indian states and union territories with their 2021 tax revenue data
data = pd.DataFrame({
    'state': [
        'Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh',
        'Goa', 'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jammu and Kashmir',
        'Jharkhand', 'Karnataka', 'Kerala', 'Madhya Pradesh', 'Maharashtra',
        'Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha',
        'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana',
        'Tripura', 'Uttar Pradesh', 'Uttarakhand', 'West Bengal',
        'Delhi', 'Puducherry'
    ],
    'tax_revenue': [
        85265, 1900, 21178, 35050, 25750,
        5473, 111693, 52887, 9282, 16276,
        23256, 111494, 71833, 64914, 243490,
        2055, 2579, 720, 1272, 37500,
        37434, 90050, 1195, 126644, 92910,
        2412, 186345, 12754, 75416,
        43000, 2639
    ]
})

# Add a parent column for all states
data['parent'] = 'India'

# Create the treemap
fig = px.treemap(
    data,
    path=['parent', 'state'],
    values='tax_revenue',
    title='Tax Revenue of Indian States and Union Territories (2021)',
    color='tax_revenue',
    color_continuous_scale='Viridis',
    hover_data={'tax_revenue': ':,.0f'},
)

# Update layout
fig.update_layout(
    title_font_size=24,
    font_size=12,
    title_x=0.5,  # Center the title
)

# Update traces
fig.update_traces(
    textinfo="label+value",
    textfont_size=12,
)

# Show the figure
fig.show()
```