CompletedCapstoneSourceCode

May 8, 2025

[1]:
```python
import pandas as pd

# Load test NSUMHSS CSV and  Faraji & Hennigan's (2024) Dataset
nsumhss_path = "NSUMHSS_2022_PUF_CSV.csv"
google_trends_path = "googleTrendsMH.xlsx"

# Load datasets
nsumhss_df = pd.read_csv(nsumhss_path)
google_trends_df = pd.read_excel(google_trends_path)

# Display the first few rows of both datasets
nsumhss_df.head(), google_trends_df.head()
```

[1]:
```
(     MPRID INSU INMH LOCATIONSTATE FOCUS SUTRTMNTALSO JAIL OTHNONTX DETOX  \
 0  100002G    1    1            IN     3            L    0        1     0
 1  100120A    M    1            MN     2            0    0        L     L
 2  100126F    1    M            FL     M            M    M        M     M
 3  100151K    1    M            NH     M            M    M        M     M
 4  100161B    1    M            DE     M            M    M        M     M

   TREATMT_SU  … VAMAINTAIN VAFUAPPTMSSD VASPC_MH VACAREMGR_MH VAPROGSUPP_MH
\
 0          1  …          L            L        L            L             L
 1          L  …          L            L        L            L             L
 2          M  …          M            M        M            M             M
 3          M  …          M            M        M            M             M
 4          M  …          M            M        M            M             M

   VAHIGHRISKHI_MH VAHIGHRISKRC_MH VAHIGHRISKOP_MH TSU_SU TSU_MH
 0               L               L               L      1      1
 1               L               L               L      M      1
 2               M               M               M      3      M
 3               M               M               M      3      M
 4               M               M               M      3      M

 [5 rows x 892 columns],
          Variable      Type                              Description  \
 0            year   integer                           reporting year
```

```
1             fips   integer                     state FIPS code
2            state  category                          state name
3           region  category          US region of state location
4   population_est   integer  state population estimate from the

                                                  Source Notes
0                                                    NaN   NaN
1    SAMHSA MH-CLD (Mental Health Client Level Data)   NaN
2                                                    NaN   NaN
3    SAMHSA MH-CLD (Mental Health Client Level Data)   NaN
4  U.S. Census Bureau's Population Estimates Prog…   NaN  )
```

[2]:
```python
import pandas as pd

# File-year mapping including 2023
file_year_map = {
    "N-SSATS-2013-DS0001-data-excel.csv": 2013,
    "N-MHSS-2014-DS0001-data-excel.csv": 2014,
    "N-MHSS-2015-DS0001-data-excel.csv": 2015,
    "nmhss_puf_2016.csv": 2016,
    "NMHSS_2017_PUF_CSV.csv": 2017,
    "nmhss-puf-2018-csv.csv": 2018,
    "nmhss-puf-2019-csv.csv": 2019,
    "nmhss-puf-2020-csv.csv": 2020,
    "NSUMHSS_2021_PUF_CSV.csv": 2021,
    "NSUMHSS_2022_PUF_CSV.csv": 2022,
    "NSUMHSS_2023_PUF_CSV.csv": 2023,
}

# Define the function again after reset
def process_nsumhss_file(path: str, year: int) -> pd.DataFrame:
    df = pd.read_csv(path)
    df['YEAR'] = year

    def convert_flag(val):
        if str(val).strip() in ['1', 'Y']:
            return 1
        elif str(val).strip() in ['0', 'N']:
            return 0
        else:
            return pd.NA

    try:
        subset = df[[
            'LOCATIONSTATE', 'FOCUS', 'CTYPEHI2',
            'SRVC95', 'SRVC30', 'SRVC120',
            'REVCHK3', 'REVCHK8_SU', 'SRVC6', 'SRVC5'
```

```python
        ]].copy()
    except KeyError:
        return pd.DataFrame()

    for col in ['SRVC95', 'SRVC30', 'SRVC120', 'REVCHK3', 'REVCHK8_SU',␣
 ↪'SRVC6', 'SRVC5']:
        subset[col] = subset[col].apply(convert_flag)

    subset['MENTAL_HEALTH_ONLY'] = df['FOCUS'].apply(lambda x: 1 if str(x).
 ↪strip() == '2' else 0)
    subset['INPATIENT'] = df['CTYPEHI2'].apply(lambda x: 1 if str(x).strip() ==␣
 ↪'1' else 0)
    subset['YEAR'] = year
    subset['YOUTH_SERVICES'] = (subset[['SRVC30', 'SRVC120']].sum(axis=1,␣
 ↪skipna=True) >= 1).astype(int)
    subset['COUNSELING_SERVICES'] = (subset[['SRVC6', 'SRVC5']].sum(axis=1,␣
 ↪skipna=True) >= 1).astype(int)

    grouped = subset.groupby(['LOCATIONSTATE', 'YEAR']).agg(
        total_facilities=('LOCATIONSTATE', 'count'),
        mental_health_only=('MENTAL_HEALTH_ONLY', 'sum'),
        inpatient_facilities=('INPATIENT', 'sum'),
        pct_pharmacotherapy=('SRVC95', 'mean'),
        pct_free_services=('REVCHK3', 'mean'),
        pct_medicare_services=('REVCHK8_SU', 'mean'),
        pct_youth_services=('YOUTH_SERVICES', 'mean'),
        pct_counseling_services=('COUNSELING_SERVICES', 'mean')
    ).reset_index()

    return grouped

# Loop through files and process
all_years_data = []
for filename, year in file_year_map.items():
    file_path = f"{filename}"
    df = process_nsumhss_file(file_path, year)
    if not df.empty:
        all_years_data.append(df)

# Concatenate all results
nsumhss_2013_2023_df = pd.concat(all_years_data, ignore_index=True)
```

```
<ipython-input-2-024a94fca3d9>:20: DtypeWarning: Columns
(2,21,23,26,50,68,69,72,74,75,78) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv(path)
```

[3]:
```python
# Reload the Google Trends dataset
google_trends_df = pd.read_excel("googleTrendsMH.xlsx",␣
↪sheet_name="googleTrendsMH")

# Redefine the file-year mapping (2013-2023)
file_year_map = {
    "N-SSATS-2013-DS0001-data-excel.csv": 2013,
    "N-MHSS-2014-DS0001-data-excel.csv": 2014,
    "N-MHSS-2015-DS0001-data-excel.csv": 2015,
    "nmhss_puf_2016.csv": 2016,
    "NMHSS_2017_PUF_CSV.csv": 2017,
    "nmhss-puf-2018-csv.csv": 2018,
    "nmhss-puf-2019-csv.csv": 2019,
    "nmhss-puf-2020-csv.csv": 2020,
    "NSUMHSS_2021_PUF_CSV.csv": 2021,
    "NSUMHSS_2022_PUF_CSV.csv": 2022,
    "NSUMHSS_2023_PUF_CSV.csv": 2023,
}

# Define NSUMHSS processor
def process_nsumhss_file(path: str, year: int) -> pd.DataFrame:
    df = pd.read_csv(path)
    df['YEAR'] = year

    def convert_flag(val):
        if str(val).strip() in ['1', 'Y']:
            return 1
        elif str(val).strip() in ['0', 'N']:
            return 0
        else:
            return pd.NA

    try:
        subset = df[[
            'LOCATIONSTATE', 'FOCUS', 'CTYPEHI2',
            'SRVC95', 'SRVC30', 'SRVC120',
            'REVCHK3', 'REVCHK8_SU', 'SRVC6', 'SRVC5'
        ]].copy()
    except KeyError:
        return pd.DataFrame()

    for col in ['SRVC95', 'SRVC30', 'SRVC120', 'REVCHK3', 'REVCHK8_SU',␣
↪'SRVC6', 'SRVC5']:
        subset[col] = subset[col].apply(convert_flag)

    subset['MENTAL_HEALTH_ONLY'] = df['FOCUS'].apply(lambda x: 1 if str(x).
↪strip() == '2' else 0)
```

```python
    subset['INPATIENT'] = df['CTYPEHI2'].apply(lambda x: 1 if str(x).strip() ==␣
↪'1' else 0)
    subset['YEAR'] = year
    subset['YOUTH_SERVICES'] = (subset[['SRVC30', 'SRVC120']].sum(axis=1,␣
↪skipna=True) >= 1).astype(int)
    subset['COUNSELING_SERVICES'] = (subset[['SRVC6', 'SRVC5']].sum(axis=1,␣
↪skipna=True) >= 1).astype(int)

    grouped = subset.groupby(['LOCATIONSTATE', 'YEAR']).agg(
        total_facilities=('LOCATIONSTATE', 'count'),
        mental_health_only=('MENTAL_HEALTH_ONLY', 'sum'),
        inpatient_facilities=('INPATIENT', 'sum'),
        pct_pharmacotherapy=('SRVC95', 'mean'),
        pct_free_services=('REVCHK3', 'mean'),
        pct_medicare_services=('REVCHK8_SU', 'mean'),
        pct_youth_services=('YOUTH_SERVICES', 'mean'),
        pct_counseling_services=('COUNSELING_SERVICES', 'mean')
    ).reset_index()

    return grouped

# Process all files
all_years_data = []
for filename, year in file_year_map.items():
    df = process_nsumhss_file(f"{filename}", year)
    if not df.empty:
        all_years_data.append(df)

nsumhss_agg = pd.concat(all_years_data, ignore_index=True)

# Prepare for merge
google_trends_df['state'] = google_trends_df['state'].str.upper()
google_trends_df['year'] = google_trends_df['year'].astype(int)

# Merge datasets
merged_df = pd.merge(
    google_trends_df,
    nsumhss_agg,
    how='left',
    left_on=['state', 'year'],
    right_on=['LOCATIONSTATE', 'YEAR']
)

# Calculate per capita metrics
merged_df['per_capita_total_facilities'] = merged_df['total_facilities'] /␣
↪merged_df['population_est']
```

```
merged_df['per_capita_mental_health_only'] = merged_df['mental_health_only'] /␣
 ↪merged_df['population_est']
merged_df['per_capita_inpatient_facilities'] =␣
 ↪merged_df['inpatient_facilities'] / merged_df['population_est']

# Drop redundant merge keys
merged_df.drop(columns=['LOCATIONSTATE', 'YEAR'], inplace=True)
```

```
<ipython-input-3-da6c1dc59368>:21: DtypeWarning: Columns
(2,21,23,26,50,68,69,72,74,75,78) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv(path)
```

[4]:
```
# Save the merged dataset locally
merged_df.to_csv("Merged_Trends_NSUMHSS_2013_2023.csv", index=False)
print("Dataset exported as 'Merged_Trends_NSUMHSS_2013_2023.csv'")
```

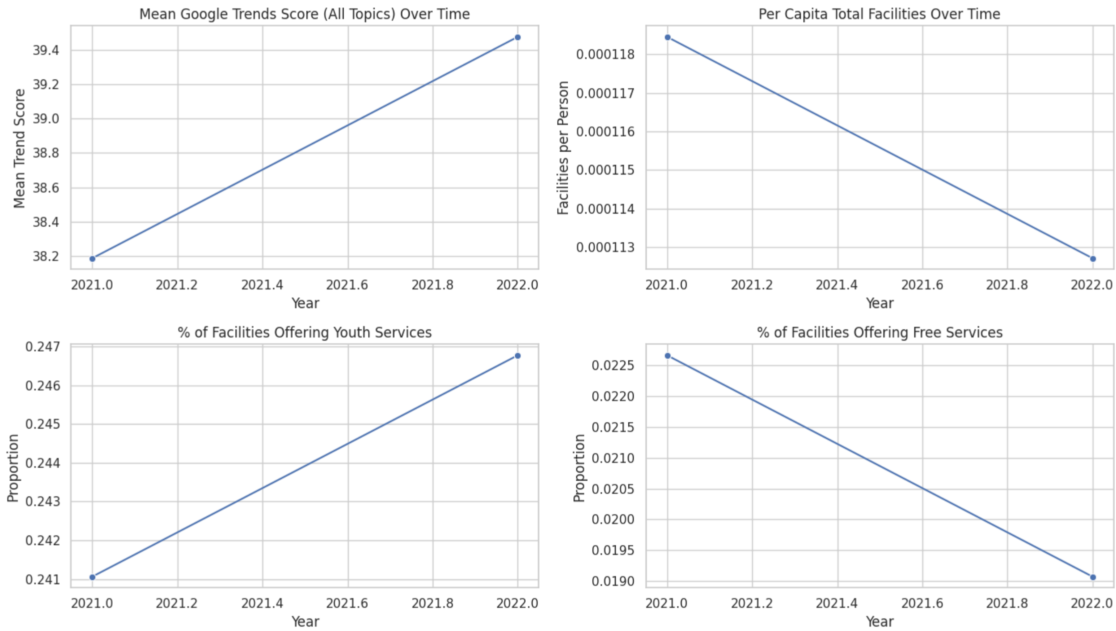
```
Dataset exported as 'Merged_Trends_NSUMHSS_2013_2023.csv'
```

[5]:
```
# Import libraries relevant for EDA
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
merged_df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Clean and prepare data for visualization
eda_summary = merged_df[[
    "year", "state", "population_est",
    "per_capita_total_facilities", "per_capita_mental_health_only",␣
 ↪"per_capita_inpatient_facilities",
    "pct_pharmacotherapy", "pct_youth_services", "pct_free_services",
    "pct_medicare_services", "pct_counseling_services", "mean_all_trends"
]].dropna()

# Set up the visual style
sns.set(style="whitegrid")
plt.figure(figsize=(14, 8))

# Plot 1: Trend of Google search interest across years
plt.subplot(2, 2, 1)
sns.lineplot(data=eda_summary, x="year", y="mean_all_trends", estimator='mean',␣
 ↪ci=None, marker='o')
plt.title("Mean Google Trends Score (All Topics) Over Time")
plt.xlabel("Year")
plt.ylabel("Mean Trend Score")

# Plot 2: Per capita total facility access over time
```

```python
plt.subplot(2, 2, 2)
sns.lineplot(data=eda_summary, x="year", y="per_capita_total_facilities",␣
↪estimator='mean', ci=None, marker='o')
plt.title("Per Capita Total Facilities Over Time")
plt.xlabel("Year")
plt.ylabel("Facilities per Person")

# Plot 3: Percentage of facilities offering youth services
plt.subplot(2, 2, 3)
sns.lineplot(data=eda_summary, x="year", y="pct_youth_services",␣
↪estimator='mean', ci=None, marker='o')
plt.title("% of Facilities Offering Youth Services")
plt.xlabel("Year")
plt.ylabel("Proportion")

# Plot 4: Percentage of facilities offering free services
plt.subplot(2, 2, 4)
sns.lineplot(data=eda_summary, x="year", y="pct_free_services",␣
↪estimator='mean', ci=None, marker='o')
plt.title("% of Facilities Offering Free Services")
plt.xlabel("Year")
plt.ylabel("Proportion")

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

```
<ipython-input-5-df23962ea2a1>:22: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=eda_summary, x="year", y="mean_all_trends",
estimator='mean', ci=None, marker='o')
<ipython-input-5-df23962ea2a1>:29: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=eda_summary, x="year", y="per_capita_total_facilities",
estimator='mean', ci=None, marker='o')
<ipython-input-5-df23962ea2a1>:36: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=eda_summary, x="year", y="pct_youth_services",
estimator='mean', ci=None, marker='o')
<ipython-input-5-df23962ea2a1>:43: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

```
  sns.lineplot(data=eda_summary, x="year", y="pct_free_services",
estimator='mean', ci=None, marker='o')
```

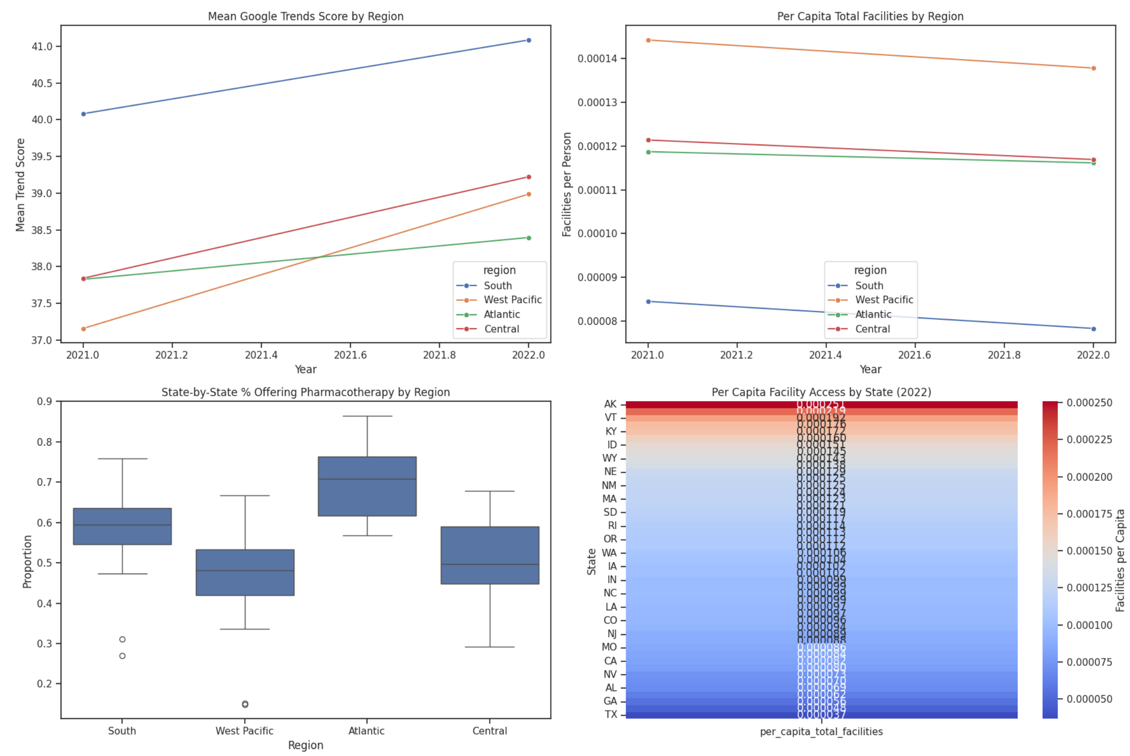


[6]:

```
# Filter for relevant years and drop missing
state_trends_df = merged_df[[
    "year", "state", "region", "mean_all_trends",
    "per_capita_total_facilities", "pct_pharmacotherapy", "pct_free_services"
]].dropna()

# Set up plotting space
plt.figure(figsize=(18, 12))
sns.set(style="ticks")

# Plot 1: Google Trends over time by region
plt.subplot(2, 2, 1)
sns.lineplot(data=state_trends_df, x="year", y="mean_all_trends", hue="region",␣
 ↪estimator="mean", ci=None, marker="o")
plt.title("Mean Google Trends Score by Region")
plt.ylabel("Mean Trend Score")
plt.xlabel("Year")

# Plot 2: Per capita facilities over time by region
plt.subplot(2, 2, 2)
sns.lineplot(data=state_trends_df, x="year", y="per_capita_total_facilities",␣
 ↪hue="region", estimator="mean", ci=None, marker="o")
```

```python
plt.title("Per Capita Total Facilities by Region")
plt.ylabel("Facilities per Person")
plt.xlabel("Year")

# Plot 3: Boxplot of pharmacotherapy access by state
plt.subplot(2, 2, 3)
sns.boxplot(data=state_trends_df, x="region", y="pct_pharmacotherapy")
plt.title("State-by-State % Offering Pharmacotherapy by Region")
plt.ylabel("Proportion")
plt.xlabel("Region")

# Plot 4: Heatmap of average facility access by state (latest year)
latest_year = state_trends_df["year"].max()
heatmap_df = state_trends_df[state_trends_df["year"] == latest_year].
 ↪pivot_table(
    index="state", values="per_capita_total_facilities", aggfunc="mean"
).sort_values("per_capita_total_facilities", ascending=False)

plt.subplot(2, 2, 4)
sns.heatmap(heatmap_df, annot=True, cmap="coolwarm", fmt=".6f",␣
 ↪cbar_kws={'label': 'Facilities per Capita'})
plt.title(f"Per Capita Facility Access by State ({latest_year})")
plt.ylabel("State")
plt.xlabel("")

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

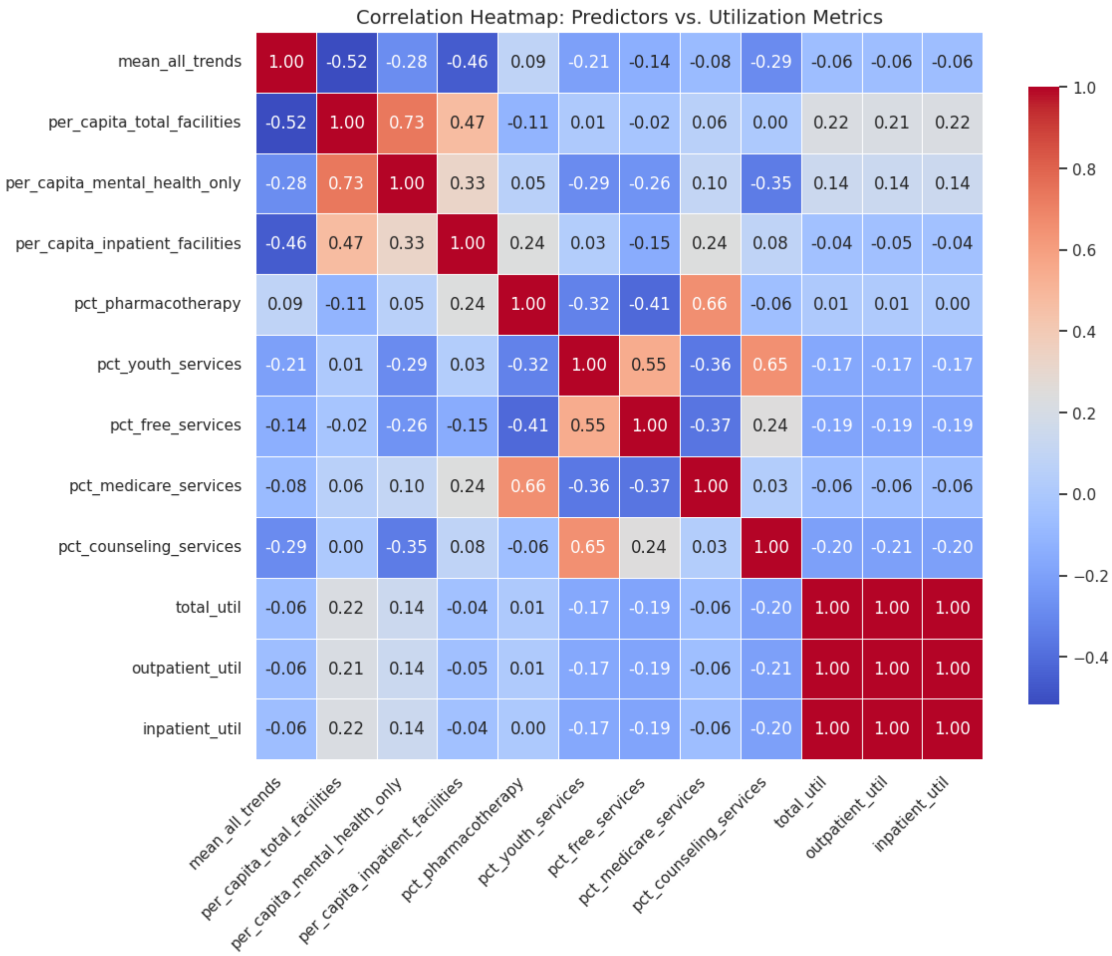
```
<ipython-input-6-d4cda7d160a0>:13: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=state_trends_df, x="year", y="mean_all_trends",
hue="region", estimator="mean", ci=None, marker="o")
<ipython-input-6-d4cda7d160a0>:20: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=state_trends_df, x="year", y="per_capita_total_facilities",
hue="region", estimator="mean", ci=None, marker="o")
```



[7]:

```python
# Prepare cleaned dataset for correlation analysis
correlation_df = merged_df[[
    "mean_all_trends",
    "per_capita_total_facilities", "per_capita_mental_health_only", "per_capita_inpatient_facilities",
    "pct_pharmacotherapy", "pct_youth_services", "pct_free_services",
    "pct_medicare_services", "pct_counseling_services",
    "total_util", "outpatient_util", "inpatient_util"
]].dropna()

# Set up and render the correlation heatmap
plt.figure(figsize=(12, 10))
sns.set(style="white")

corr_matrix = correlation_df.corr()

sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True, linewidths=0.5, cbar_kws={"shrink": 0.8})
plt.title("Correlation Heatmap: Predictors vs. Utilization Metrics", fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
```

```
plt.show()
```

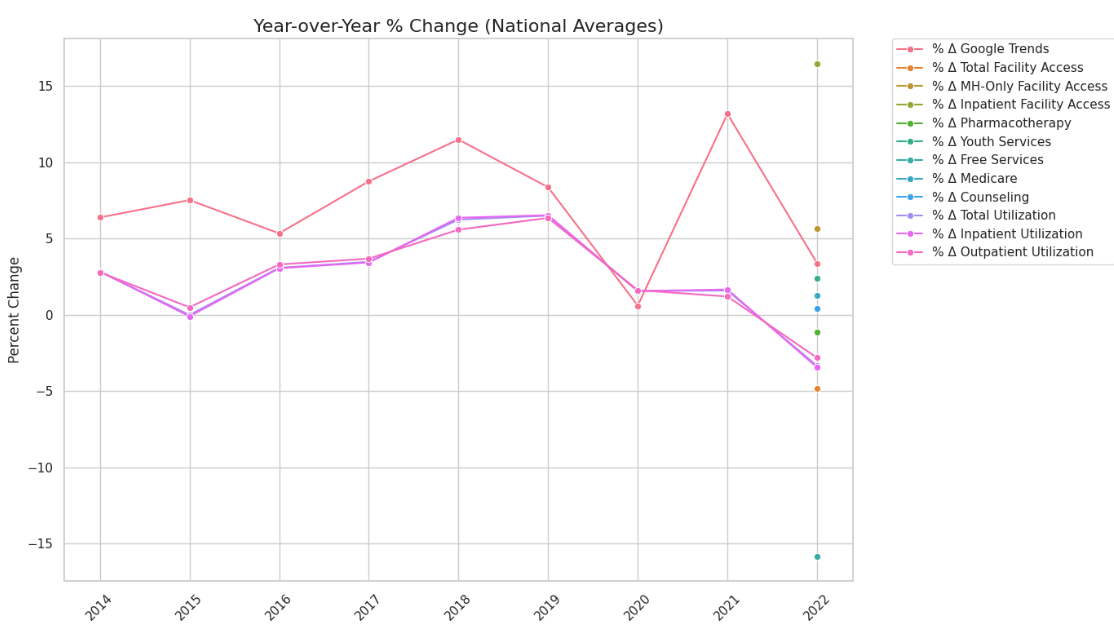


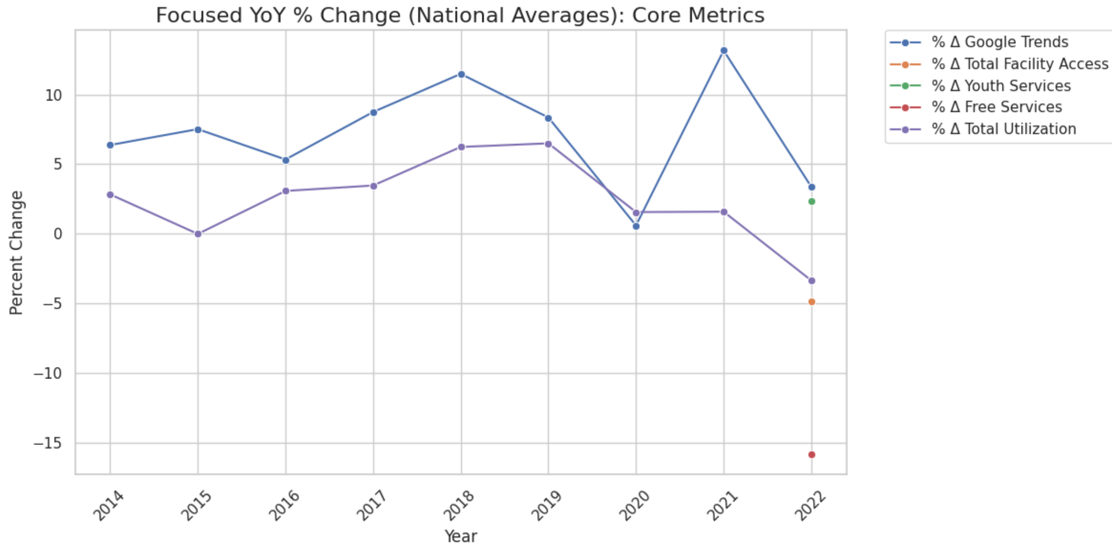
[8]:
```python
# Group by year and calculate national averages
yearly_df = merged_df.groupby("year").agg({
    "mean_all_trends": "mean",
    "per_capita_total_facilities": "mean",
    "per_capita_mental_health_only": "mean",
    "per_capita_inpatient_facilities": "mean",
    "pct_pharmacotherapy": "mean",
    "pct_youth_services": "mean",
    "pct_free_services": "mean",
    "pct_medicare_services": "mean",
    "pct_counseling_services": "mean",
    "total_util": "mean",
    "inpatient_util": "mean",
    "outpatient_util": "mean"
}).reset_index()
```

```python
# Compute YoY % change
yoy_change = yearly_df.copy()
yoy_change.iloc[:, 1:] = yearly_df.iloc[:, 1:].pct_change() * 100
yoy_change = yoy_change.round(2)

# Rename columns for readability
yoy_change.rename(columns={
    "mean_all_trends": "% Δ Google Trends",
    "per_capita_total_facilities": "% Δ Total Facility Access",
    "per_capita_mental_health_only": "% Δ MH-Only Facility Access",
    "per_capita_inpatient_facilities": "% Δ Inpatient Facility Access",
    "pct_pharmacotherapy": "% Δ Pharmacotherapy",
    "pct_youth_services": "% Δ Youth Services",
    "pct_free_services": "% Δ Free Services",
    "pct_medicare_services": "% Δ Medicare",
    "pct_counseling_services": "% Δ Counseling",
    "total_util": "% Δ Total Utilization",
    "inpatient_util": "% Δ Inpatient Utilization",
    "outpatient_util": "% Δ Outpatient Utilization"
}, inplace=True)
```

[9]:
```python
# Year-over-Year% Change Analysis

# Filter out the first year (2013) since it has all NaNs for % change
yoy_plot_data = yoy_change[yoy_change["year"] > 2013].copy()

# Melt the dataframe to long format for seaborn
yoy_long = yoy_plot_data.melt(id_vars="year", var_name="Metric",␣
 ↪value_name="YoY % Change")

# Set up the plot
plt.figure(figsize=(14, 8))
sns.set(style="whitegrid")

# Lineplot for each metric
sns.lineplot(data=yoy_long, x="year", y="YoY % Change", hue="Metric",␣
 ↪marker="o")

plt.title("Year-over-Year % Change (National Averages)", fontsize=16)
plt.xlabel("Year")
plt.ylabel("Percent Change")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```python
# Select core metrics to highlight in visualization
core_metrics = [
    "% Δ Google Trends",
    "% Δ Total Facility Access",
    "% Δ Youth Services",
    "% Δ Free Services",
    "% Δ Total Utilization"
]

# Filter the long-form dataframe to include only selected metrics
core_yoy_long = yoy_long[yoy_long["Metric"].isin(core_metrics)]

# Plot refined year-over-year visualization
plt.figure(figsize=(12, 6))
sns.set(style="whitegrid")

sns.lineplot(data=core_yoy_long, x="year", y="YoY % Change", hue="Metric",␣
 ↪marker="o")

plt.title("Focused YoY % Change (National Averages): Core Metrics", fontsize=16)
plt.xlabel("Year")
plt.ylabel("Percent Change")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

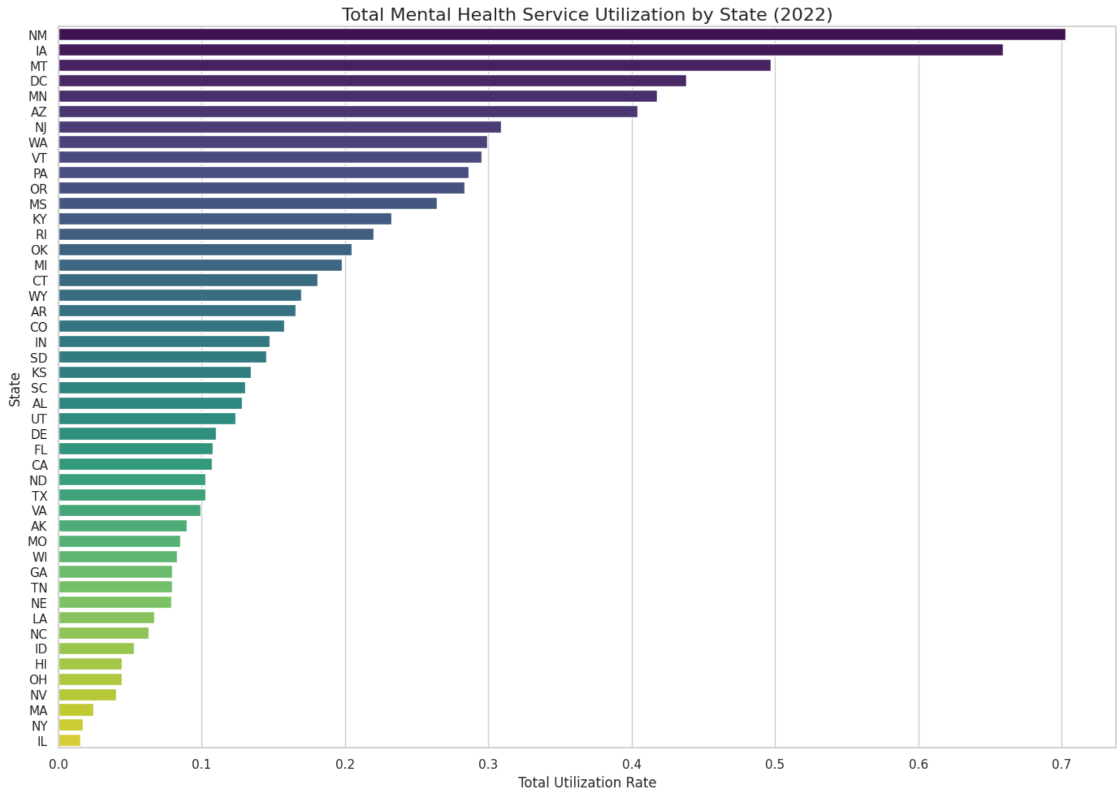




[10]:
```python
# Visualize all the States together

# Set up plot
plt.figure(figsize=(14, 10))
sns.set(style="whitegrid")

# Filter most recent year with available utilization data
latest_year = merged_df["year"].max()
state_util_df = merged_df[merged_df["year"] == latest_year].copy()

# Sort by total utilization
state_util_df = state_util_df.sort_values("total_util", ascending=False)

# Plot bar chart of total utilization by state
sns.barplot(data=state_util_df, y="state", x="total_util", palette="viridis")

plt.title(f"Total Mental Health Service Utilization by State ({latest_year})",␣
  ↪fontsize=16)
plt.xlabel("Total Utilization Rate")
plt.ylabel("State")
plt.tight_layout()
plt.show()
```

```
<ipython-input-10-c25c0619643f>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
```

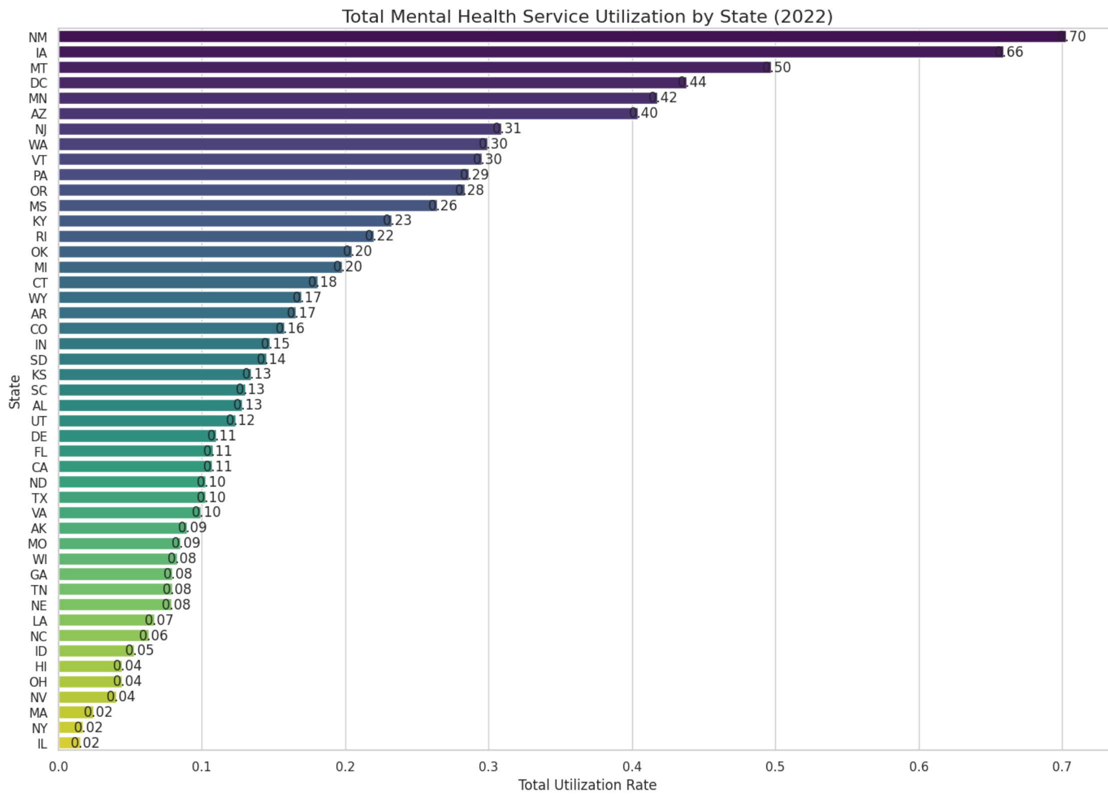
```
effect.

  sns.barplot(data=state_util_df, y="state", x="total_util", palette="viridis")
```



[11]:
```python
plt.figure(figsize=(14, 10))
sns.set(style="whitegrid")

# Plot bar chart
ax = sns.barplot(data=state_util_df, y="state", x="total_util",␣
 ↪palette="viridis")

# Annotate bars with the total utilization value
for p in ax.patches:
    ax.annotate(f'{p.get_width():.2f}', (p.get_x() + p.get_width(), p.get_y() +␣
 ↪p.get_height() / 2),
                xytext=(5, 0), textcoords="offset points", ha='center',␣
 ↪va='center')

plt.title(f"Total Mental Health Service Utilization by State ({latest_year})",␣
 ↪fontsize=16)
plt.xlabel("Total Utilization Rate")
plt.ylabel("State")
```

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

```
<ipython-input-11-f34d2593bd5f>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  ax = sns.barplot(data=state_util_df, y="state", x="total_util",
palette="viridis")
```




```
[12]: # Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Function to summarize each variable
def create_codebook(df):
    codebook = pd.DataFrame({
        "Variable": df.columns,
        "Data Type": df.dtypes.values,
        "Missing Values": df.isnull().sum().values,
        "Unique Values": df.nunique().values,
```

```
        "Min": [df[col].min() if pd.api.types.is_numeric_dtype(df[col]) else␣
 ↪None for col in df.columns],
        "Max": [df[col].max() if pd.api.types.is_numeric_dtype(df[col]) else␣
 ↪None for col in df.columns],
        "Example Value": [df[col].dropna().iloc[0] if not df[col].dropna().
 ↪empty else None for col in df.columns],
        "Description": ["[Enter description here]" for _ in df.columns]
    })
    return codebook

# Generate codebook
codebook_df = create_codebook(df)

# Export codebook to CSV or Excel for review and editing
codebook_df.to_csv("Codebook_Mental_Health_Project.csv", index=False)
# OR for Excel
# codebook_df.to_excel("Codebook_Mental_Health_Project.xlsx", index=False)

print("Codebook generated and saved as 'Codebook_Mental_Health_Project.csv'")
```

```
Codebook generated and saved as 'Codebook_Mental_Health_Project.csv'
```

[13]:
```
from scipy.stats import skew, kurtosis

# Reload the dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# CATEGORICAL SUMMARY
# Frequencies and proportions for 'state' and 'region'
categorical_summary = df.groupby("region")["state"].nunique().reset_index()
categorical_summary.columns = ["Region", "Unique States"]
categorical_summary["Total States"] = df["state"].nunique()
categorical_summary["Proportion (%)"] = round((categorical_summary["Unique␣
 ↪States"] / categorical_summary["Total States"]) * 100, 2)

# CONTINUOUS SUMMARY
continuous_vars = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "total_util", "outpatient_util", "inpatient_util"
]

continuous_summary = []
```

```
for var in continuous_vars:
    if var in df.columns:
        continuous_summary.append({
            "Variable": var,
            "Mean": round(df[var].mean(), 3),
            "Median": round(df[var].median(), 3),
            "Std Dev": round(df[var].std(), 3),
            "Min": round(df[var].min(), 3),
            "Max": round(df[var].max(), 3),
            "Skew": round(skew(df[var].dropna()), 3),
            "Kurtosis": round(kurtosis(df[var].dropna()), 3)
        })

# Create DataFrame from the list
continuous_summary = pd.DataFrame(continuous_summary)

categorical_summary
```

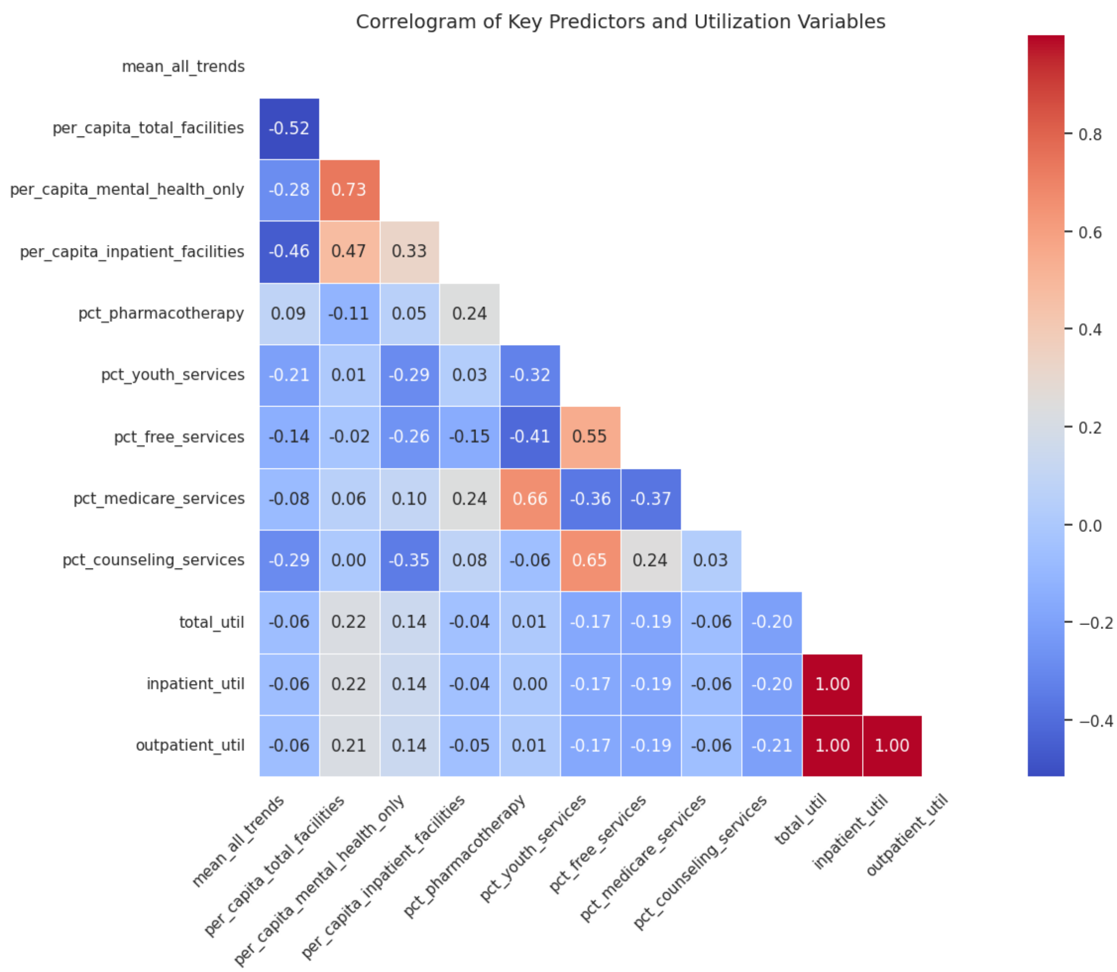
[13]:

	Region	Unique States	Total States	Proportion (%)
0	Atlantic	9	47	19.15
1	Central	13	47	27.66
2	South	12	47	25.53
3	West Pacific	13	47	27.66

[14]:

```
# Load the merged dataset
import numpy as np # Import numpy
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")
df_clean = df.dropna()

# Define continuous variables for collinearity analysis
collinearity_vars = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "total_util", "inpatient_util", "outpatient_util"
]

# Compute correlation matrix
corr_matrix = df_clean[collinearity_vars].corr()

# Plot correlogram (upper triangle mask)
```

```
plt.figure(figsize=(12, 10))
sns.set(style="white")

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, mask=mask, annot=True, cmap="coolwarm", fmt=".2f",
    linewidths=0.5)
plt.title("Correlogram of Key Predictors and Utilization Variables",
    fontsize=14)
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



The code below makes several enhancements from our EDA report and transforms lagged features in preparation for modeling.

[15]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import os

# Load merged dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# STEP 1: CLEANING & TRANSFORMATION

# Drop duplicates and outliers
df.drop_duplicates(inplace=True)
numeric_cols = df.select_dtypes(include=np.number).columns
Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1
df = df[~((df[numeric_cols] < (Q1 - 1.5 * IQR)) | (df[numeric_cols] > (Q3 + 1.5␣
↪* IQR))).any(axis=1)]

# Standardize identifiers
df['state'] = df['state'].str.upper()
df['region'] = df['region'].str.title()

# Create COVID flag
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

# Create % change features
df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# STEP 2: FEATURE ENGINEERING

# Log-transform skewed variables
for col in ['per_capita_inpatient_facilities', 'total_util', 'outpatient_util']:
    if df[col].skew() > 1:
        df[f"log_{col}"] = np.log1p(df[col])

# Normalize and mean-center percentage variables
pct_vars = [
    'pct_pharmacotherapy', 'pct_youth_services', 'pct_free_services',
    'pct_medicare_services', 'pct_counseling_services'
]
for var in pct_vars:
    df[f"{var}_norm"] = (df[var] - df[var].mean()) / df[var].std()
```

```python
# Create categorical bin for youth services
df['high_youth_services'] = pd.qcut(df['pct_youth_services'], q=3,␣
↪labels=["Low", "Medium", "High"])

# Drop nulls where necessary for modeling
df_model = df.dropna()

# STEP 3: TRAIN-TEST SPLIT

X = df_model.drop(columns=["total_util", "inpatient_util", "outpatient_util"]) ␣
↪# dropping outcomes
y = df_model["total_util"]  # our target variable

# Remove or adjust the stratify parameter
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42 # Removed stratify
    # Alternatively, reduce test_size, e.g., test_size=0.1
)

# Save processed files for modeling
os.makedirs("processed_data", exist_ok=True)
X_train.to_csv("processed_data/X_train.csv", index=False)
X_test.to_csv("processed_data/X_test.csv", index=False)
y_train.to_csv("processed_data/y_train.csv", index=False)
y_test.to_csv("processed_data/y_test.csv", index=False)

print("Preprocessing complete. Files saved to 'processed_data/'.")
```

```
Preprocessing complete. Files saved to 'processed_data/'.
```

```
<ipython-input-15-8881705a18d5>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['state'] = df['state'].str.upper()
<ipython-input-15-8881705a18d5>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['region'] = df['region'].str.title()
<ipython-input-15-8881705a18d5>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)
```

[16]:

```
# Importing required libraries
from scipy.stats import skew, kurtosis

# Load the dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag before dropping missing values
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

# Drop rows with missing values after creating the covid_flag column
df_clean = df.dropna()

# Table 1: Summary for continuous variables
continuous_vars = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "total_util", "inpatient_util", "outpatient_util"
]

summary_stats = pd.DataFrame(columns=[
    "Variable", "Mean", "Median", "Std Dev", "Min", "Max", "Skewness",␣
 ↪"Kurtosis"
])

for var in continuous_vars:
    summary_stats = pd.concat([summary_stats, pd.DataFrame([{
        "Variable": var,
        "Mean": df_clean[var].mean(),
        "Median": df_clean[var].median(),
        "Std Dev": df_clean[var].std(),
        "Min": df_clean[var].min(),
        "Max": df_clean[var].max(),
        "Skewness": skew(df_clean[var]),
        "Kurtosis": kurtosis(df_clean[var])
    }])], ignore_index=True)

# Table 2: Summary for categorical variables
region_summary = df_clean['region'].value_counts().reset_index()
region_summary.columns = ['Region', 'Count']
```

```
region_summary['Proportion'] = region_summary['Count'] /␣
↪region_summary['Count'].sum()

covid_summary = df_clean['covid_flag'].value_counts().reset_index()
covid_summary.columns = ['COVID Period (1=Yes, 0=No)', 'Count']
covid_summary['Proportion'] = covid_summary['Count'] / covid_summary['Count'].
↪sum()

# Combine into one table
table2_summary = pd.concat([
    region_summary.rename(columns={"Region": "Category", "Count": "Count",␣
↪"Proportion": "Proportion"}).assign(Variable="Region"),
    covid_summary.rename(columns={"COVID Period (1=Yes, 0=No)": "Category"}).
↪assign(Variable="COVID Flag")
])

table2_summary
```

```
<ipython-input-16-b60a48ee7fe9>:26: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future version,
this will no longer exclude empty or all-NA columns when determining the result
dtypes. To retain the old behavior, exclude the relevant entries before the
concat operation.
  summary_stats = pd.concat([summary_stats, pd.DataFrame([{
```

[16]:

	Category	Count	Proportion	Variable
0	West Pacific	26	0.282609	Region
1	Central	25	0.271739	Region
2	South	23	0.250000	Region
3	Atlantic	18	0.195652	Region
0	1	92	1.000000	COVID Flag

[17]:

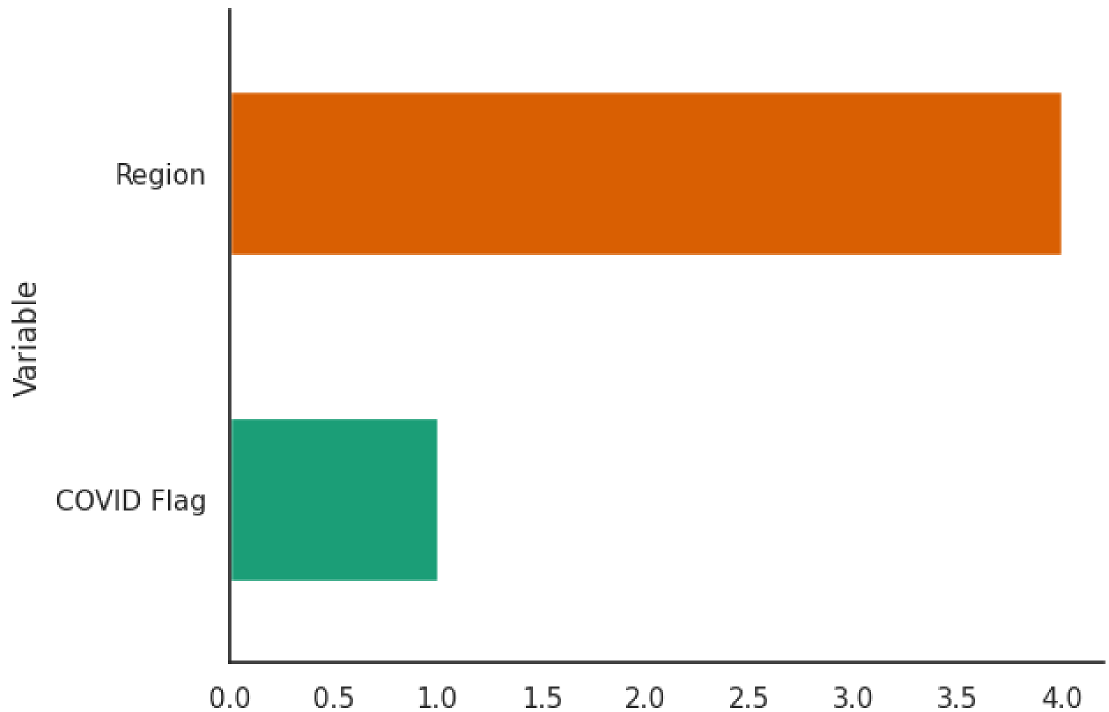
```
# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Drop rows with missing values
df_clean = df.dropna()

# Define continuous variables to summarize
continuous_vars = [
    "mean_all_trends", "per_capita_total_facilities",␣
↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "total_util", "inpatient_util", "outpatient_util"
]
```

```python
# Create summary statistics table
summary_stats = pd.DataFrame(columns=[
    "Variable", "Mean", "Median", "Std Dev", "Min", "Max", "Skewness",␣
 ↪"Kurtosis"
])

for var in continuous_vars:
    summary_stats = pd.concat([summary_stats, pd.DataFrame([{
        "Variable": var,
        "Mean": df_clean[var].mean(),
        "Median": df_clean[var].median(),
        "Std Dev": df_clean[var].std(),
        "Min": df_clean[var].min(),
        "Max": df_clean[var].max(),
        "Skewness": skew(df_clean[var]),
        "Kurtosis": kurtosis(df_clean[var])
    }])], ignore_index=True)

# Display table
print(summary_stats.to_string(index=False))
```

```
                      Variable      Mean    Median  Std Dev          Min
Max  Skewness  Kurtosis
               mean_all_trends 38.845209 40.180556 4.446674 2.273148e+01
45.212963 -1.657818  2.338082
    per_capita_total_facilities  0.000116  0.000111 0.000042 3.659726e-05
0.000253  1.110642  1.890322
  per_capita_mental_health_only  0.000022  0.000018 0.000011 5.649744e-06
0.000059  1.117363  1.281595
per_capita_inpatient_facilities  0.000003  0.000003 0.000002 3.403766e-07
0.000009  1.128248  0.969646
            pct_pharmacotherapy  0.552965  0.569416 0.138405 1.492537e-01
0.864253 -0.364121  0.454227
             pct_youth_services  0.243981  0.235394 0.069434 1.360000e-01
0.557789  1.831133  5.675273
              pct_free_services  0.020828  0.008828 0.035938 0.000000e+00
0.244444  4.500205 23.484133
          pct_medicare_services  0.497940  0.479986 0.130570 1.798561e-01
0.750000  0.119176 -0.796282
       pct_counseling_services  0.532881  0.532098 0.069975 3.820961e-01
0.734940  0.291165  0.303285
                     total_util  0.187976  0.130778 0.161259 1.519667e-02
0.732601  1.696964  2.627631
                 inpatient_util  0.160355  0.111987 0.137898 1.279340e-02
0.626625  1.695225  2.618728
                outpatient_util  0.027621  0.019957 0.023384 2.403268e-03
0.105976  1.702838  2.671791
```

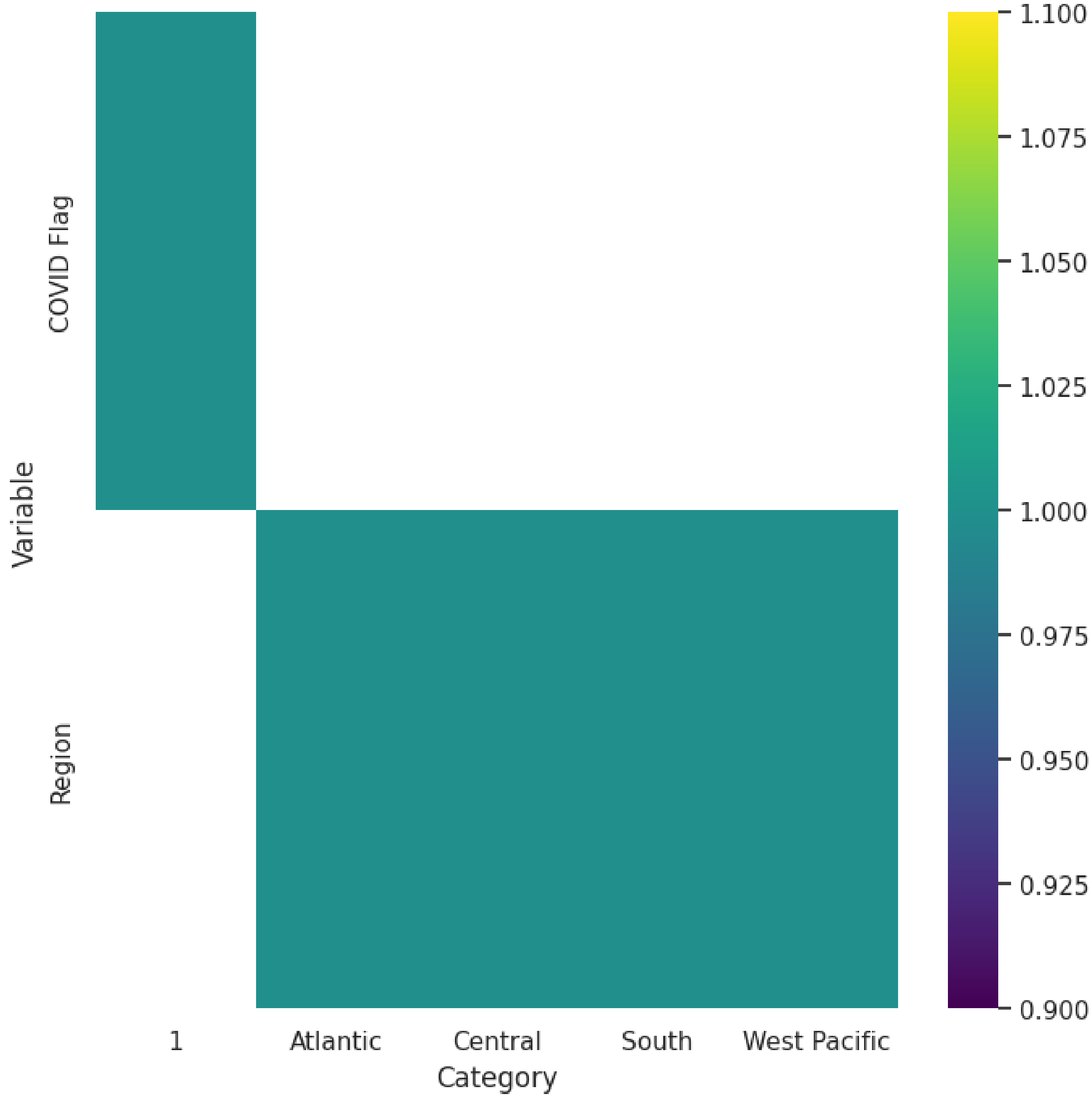
```
<ipython-input-17-055b82c6b6fe>:21: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future version,
this will no longer exclude empty or all-NA columns when determining the result
dtypes. To retain the old behavior, exclude the relevant entries before the
concat operation.
  summary_stats = pd.concat([summary_stats, pd.DataFrame([{
```

[18]:
```
from matplotlib import pyplot as plt
import seaborn as sns
table2_summary.groupby('Variable').size().plot(kind='barh', color=sns.palettes.
 ↪mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[19]:
```
# @title Category vs Variable

from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
plt.subplots(figsize=(8, 8))
df_2dhist = pd.DataFrame({
    x_label: grp['Variable'].value_counts()
    for x_label, grp in table2_summary.groupby('Category')
})
sns.heatmap(df_2dhist, cmap='viridis')
```

```
plt.xlabel('Category')
_ = plt.ylabel('Variable')
```



Ths section below execute our inital modeling phase. We will be modeling a ridge regression, random forest, knn, and xgboost and plot evalutions metrics.

[20]:
```
# Required libraries
!pip install shap xgboost
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
```

```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
import shap
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

# Load dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag and percentage change features before defining X and y
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# Define features and target
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
 ↪"pct_change_outpatient_util"
]
target = "total_util"

X = df[features]
y = df[target]

# Drop rows with NaN values in X and y
X = X.dropna()
y = y[X.index]  # Align y with the dropped rows in X

# Train-test split (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=df.loc[X.index, "region"]
)

# Standardize features for Ridge Regression and kNN
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -------------------------------
# Ridge Regression
# -------------------------------
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
ridge_mae = mean_absolute_error(y_test, y_pred_ridge)
ridge_r2 = r2_score(y_test, y_pred_ridge)

print("Ridge Regression:")
print(f"  RMSE: {ridge_rmse:.4f}")
print(f"  MAE: {ridge_mae:.4f}")
print(f"  R^2: {ridge_r2:.4f}")

# -------------------------------
# Random Forest Regressor
# -------------------------------
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

rf_rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
rf_mae = mean_absolute_error(y_test, y_pred_rf)
rf_r2 = r2_score(y_test, y_pred_rf)

print("\nRandom Forest:")
print(f"  RMSE: {rf_rmse:.4f}")
print(f"  MAE: {rf_mae:.4f}")
print(f"  R^2: {rf_r2:.4f}")

# -------------------------------
# k-Nearest Neighbors Regressor
# -------------------------------
knn_model = KNeighborsRegressor()  # You can adjust n_neighbors
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)

knn_rmse = np.sqrt(mean_squared_error(y_test, y_pred_knn))
knn_mae = mean_absolute_error(y_test, y_pred_knn)
knn_r2 = r2_score(y_test, y_pred_knn)
```

```
print("\nk-Nearest Neighbors:")
print(f"  RMSE: {knn_rmse:.4f}")
print(f"  MAE: {knn_mae:.4f}")
print(f"  R^2: {knn_r2:.4f}")

# -------------------------------
# XGBoost Regressor
# -------------------------------
xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

xgb_rmse = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
xgb_mae = mean_absolute_error(y_test, y_pred_xgb)
xgb_r2 = r2_score(y_test, y_pred_xgb)

print("\nXGBoost:")
print(f"  RMSE: {xgb_rmse:.4f}")
print(f"  MAE: {xgb_mae:.4f}")
print(f"  R^2: {xgb_r2:.4f}")

# -------------------------------
# Model Evaluation and Overfitting Check
# -------------------------------

# --- Assessing Assumptions
# Residual analysis for Ridge Regression
ridge_residuals = y_test - y_pred_ridge
plt.figure()
sns.histplot(ridge_residuals, kde=True)
plt.title("Residual Distribution - Ridge Regression")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()

# --- Checking for Overfitting ---
# Compare training and test performance for each model
models = [ridge, rf_model, knn_model, xgb_model]
model_names = ["Ridge Regression", "Random Forest", "k-NN", "XGBoost"]
metrics = ["RMSE", "MAE", "R^2"]

for model, name in zip(models, model_names):
    # Get predictions for training data
    y_train_pred = model.predict(X_train_scaled if name in ["Ridge Regression",
 ↪"k-NN"] else X_train)

    # Calculate training metrics
```

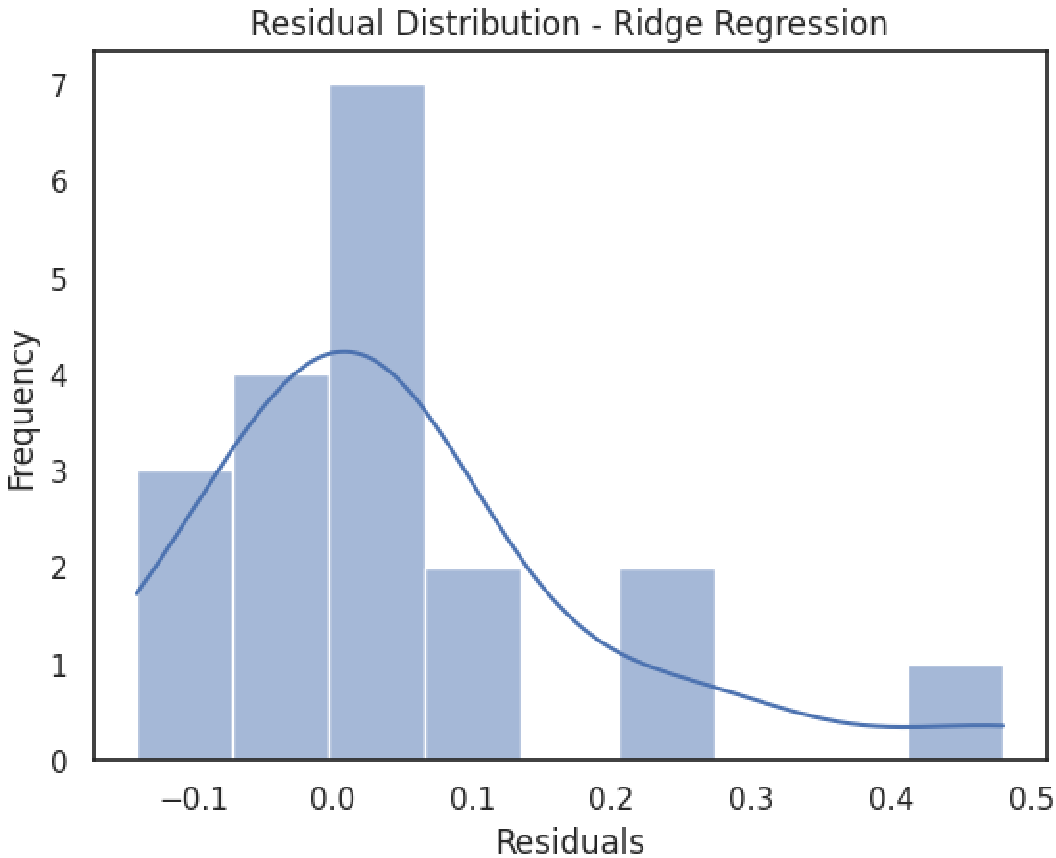
```
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    train_mae = mean_absolute_error(y_train, y_train_pred)
    train_r2 = r2_score(y_train, y_train_pred)

    # Get predictions for test data
    y_test_pred = y_pred_ridge if name == "Ridge Regression" else \
                  y_pred_rf if name == "Random Forest" else \
                  y_pred_knn if name == "k-NN" else \
                  y_pred_xgb

    # Calculate test metrics
    test_rmse = ridge_rmse if name == "Ridge Regression" else \
                rf_rmse if name == "Random Forest" else \
                knn_rmse if name == "k-NN" else \
                xgb_rmse
    test_mae = ridge_mae if name == "Ridge Regression" else \
               rf_mae if name == "Random Forest" else \
               knn_mae if name == "k-NN" else \
               xgb_mae
    test_r2 = ridge_r2 if name == "Ridge Regression" else \
              rf_r2 if name == "Random Forest" else \
              knn_r2 if name == "k-NN" else \
              xgb_r2

    print(f"\n{name}:")
    print("  Training Metrics:")
    print(f"    RMSE: {train_rmse:.4f}")
    print(f"    MAE: {train_mae:.4f}")
    print(f"    R^2: {train_r2:.4f}")
    print("  Test Metrics:")
    print(f"    RMSE: {test_rmse:.4f}")
    print(f"    MAE: {test_mae:.4f}")
    print(f"    R^2: {test_r2:.4f}")


# -------------------------------
# SHAP Plots
# -------------------------------

# SHAP for XGBoost
explainer_xgb = shap.Explainer(xgb_model)
shap_values_xgb = explainer_xgb(X_test)
plt.figure()
shap.summary_plot(shap_values_xgb, X_test, show=False)
plt.title("SHAP Summary Plot - XGBoost")
plt.tight_layout()
plt.savefig("shap_summary_xgb.png")
```

```
plt.close()

# SHAP for Random Forest
explainer_rf = shap.Explainer(rf_model)
shap_values_rf = explainer_rf(X_test)
plt.figure()
shap.summary_plot(shap_values_rf, X_test, show=False)
plt.title("SHAP Summary Plot - Random Forest")
plt.tight_layout()
plt.savefig("shap_summary_rf.png")
plt.close()


# -------------------------------
# Output Model Performance Summary
# -------------------------------

model_results = pd.DataFrame({
    "Model": ["Ridge Regression", "Random Forest", "k-Nearest Neighbors",␣
 ↪"XGBoost"],
    "RMSE": [ridge_rmse, rf_rmse, knn_rmse, xgb_rmse],
    "MAE": [ridge_mae, rf_mae, knn_mae, xgb_mae],
    "R^2 Score": [ridge_r2, rf_r2, knn_r2, xgb_r2]
})

# --- Plotting Model Performance ---
fig, ax = plt.subplots(figsize=(10, 6))  # Increased figsize for better␣
 ↪visibility
model_results.plot(x="Model", y=["RMSE", "MAE", "R^2 Score"], kind="bar", ax=ax)
ax.set_title("Model Performance Comparison")
ax.set_ylabel("Metric Value")
ax.set_xticklabels(model_results["Model"], rotation=45, ha="right")
plt.tight_layout()
plt.show()

model_results.to_csv("model_performance_summary.csv", index=False)

print("Generated SHAP plots: shap_summary_xgb.png, shap_summary_rf.png")
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
(0.47.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-
packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from shap) (1.15.2)
```

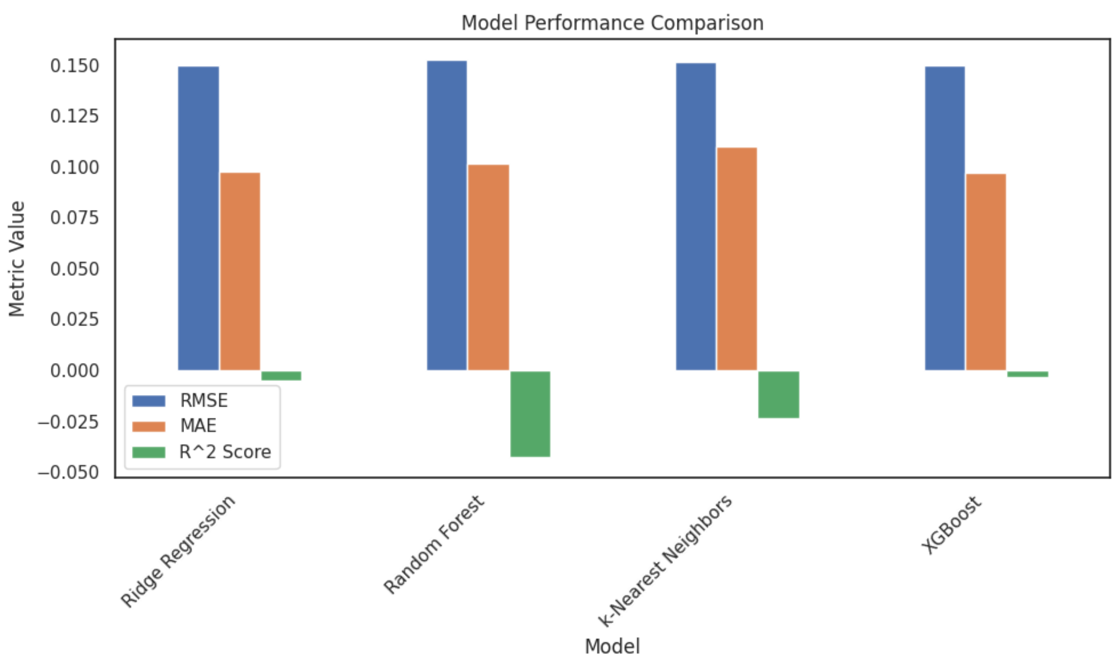
```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
Ridge Regression:
  RMSE: 0.1499
  MAE: 0.0974
  R^2: -0.0052

Random Forest:
  RMSE: 0.1527
  MAE: 0.1014
  R^2: -0.0427

k-Nearest Neighbors:
  RMSE: 0.1513
  MAE: 0.1098
  R^2: -0.0234

XGBoost:
```

```
  RMSE: 0.1498
  MAE: 0.0968
  R^2: -0.0031
```




```
Ridge Regression:
  Training Metrics:
    RMSE: 0.1284
    MAE: 0.0962
    R^2: 0.3688
  Test Metrics:
    RMSE: 0.1499
    MAE: 0.0974
    R^2: -0.0052

Random Forest:
  Training Metrics:
    RMSE: 0.0532
    MAE: 0.0374
    R^2: 0.8918
```

```
  Test Metrics:
    RMSE: 0.1527
    MAE: 0.1014
    R^2: -0.0427

k-NN:
  Training Metrics:
    RMSE: 0.1306
    MAE: 0.0947
    R^2: 0.3473
  Test Metrics:
    RMSE: 0.1513
    MAE: 0.1098
    R^2: -0.0234

XGBoost:
  Training Metrics:
    RMSE: 0.0005
    MAE: 0.0004
    R^2: 1.0000
  Test Metrics:
    RMSE: 0.1498
    MAE: 0.0968
    R^2: -0.0031
```




```
Generated SHAP plots: shap_summary_xgb.png, shap_summary_rf.png
```

The code below provides our teams revisions and hyperparameter tuning after the initial models report. We will also execute the LSTM here and perform comparative model analysis amongst these models.

[21]:
```
# Required libraries
!pip install shap xgboost
import pandas as pd
import numpy as np
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import xgboost as xgb
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag and percentage change features before defining X and y
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# Now drop rows with NaN values after generating new columns
df.dropna(inplace=True)

# Feature selection
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
 ↪"pct_change_outpatient_util"
]
target = "total_util"

X = df[features]
y = df[target]

# Split and scale data
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=df["region"])
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# RidgeCV with alpha tuning
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
ridge_cv = RidgeCV(alphas=alphas, scoring='r2', store_cv_values=True)
ridge_cv.fit(X_train_scaled, y_train)
y_pred_ridge = ridge_cv.predict(X_test_scaled)

ridge_results = {
    "Model": "Ridge Regression (CV)",
    "Best Alpha": ridge_cv.alpha_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
    "MAE": mean_absolute_error(y_test, y_pred_ridge),
    "R^2": r2_score(y_test, y_pred_ridge)
}

# XGBoost grid search
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
param_grid = {
    "n_estimators": [50, 100],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1],
    "subsample": [0.8, 1.0]
}

grid_xgb = GridSearchCV(xgb_model, param_grid, cv=5, scoring="r2", verbose=0)
grid_xgb.fit(X_train, y_train)
y_pred_xgb = grid_xgb.predict(X_test)

xgb_results = {
    "Model": "XGBoost (Tuned)",
    "Best Params": grid_xgb.best_params_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_xgb)),
    "MAE": mean_absolute_error(y_test, y_pred_xgb),
    "R^2": r2_score(y_test, y_pred_xgb)
}

# Combine and display results
results_df = pd.DataFrame([ridge_results, xgb_results])
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
(0.47.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-
packages (2.1.4)
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from shap) (1.15.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
```

[22]:
```
# Required libraries
!pip install shap xgboost
import pandas as pd
import numpy as np
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import xgboost as xgb
import matplotlib.pyplot as plt
```

```python
import seaborn as sns

# Load dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag and percentage change features before defining X and y
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# Now drop rows with NaN values after generating new columns
df.dropna(inplace=True)

# Feature selection
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
 ↪"pct_change_outpatient_util"
]
target = "total_util"

X = df[features]
y = df[target]

# Split and scale data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=df["region"])
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# RidgeCV with alpha tuning
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
ridge_cv = RidgeCV(alphas=alphas, scoring='r2', store_cv_values=True)
ridge_cv.fit(X_train_scaled, y_train)
y_pred_ridge = ridge_cv.predict(X_test_scaled)

ridge_results = {
    "Model": "Ridge Regression (CV)",
```

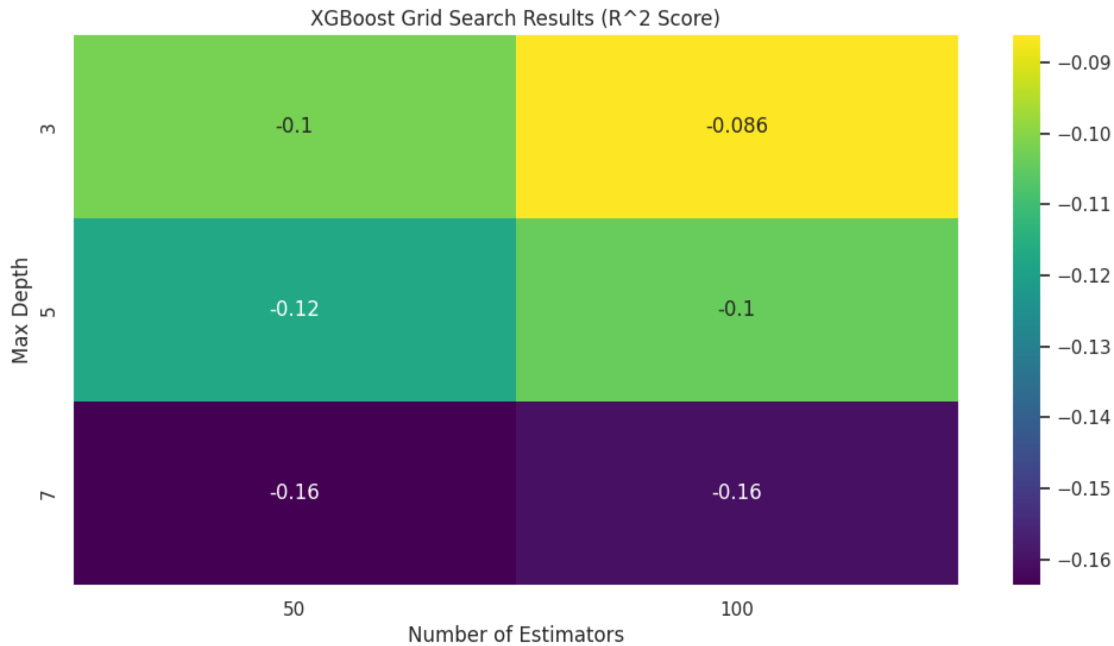
```
    "Best Alpha": ridge_cv.alpha_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
    "MAE": mean_absolute_error(y_test, y_pred_ridge),
    "R^2": r2_score(y_test, y_pred_ridge)
}

# XGBoost grid search with result tracking
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
param_grid = {
    "n_estimators": [50, 100],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1],
    "subsample": [0.8, 1.0]
}

grid_xgb = GridSearchCV(xgb_model, param_grid, cv=5, scoring="r2", verbose=0,␣
 ↪return_train_score=True)
grid_xgb.fit(X_train, y_train)
y_pred_xgb = grid_xgb.predict(X_test)

xgb_results = {
    "Model": "XGBoost (Tuned)",
    "Best Params": grid_xgb.best_params_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_xgb)),
    "MAE": mean_absolute_error(y_test, y_pred_xgb),
    "R^2": r2_score(y_test, y_pred_xgb)
}

# Print and plot grid search results for XGBoost
print("XGBoost Grid Search Results:")
results_df = pd.DataFrame(grid_xgb.cv_results_)
print(results_df[["params", "mean_test_score", "std_test_score",␣
 ↪"mean_train_score", "std_train_score"]])

# Plotting the results

# Group by hyperparameters and average scores to avoid duplicate index entries
heatmap_data = results_df.groupby(['param_max_depth',␣
 ↪'param_n_estimators'])['mean_test_score'].mean().reset_index()
heatmap_data = heatmap_data.pivot(index='param_max_depth',␣
 ↪columns='param_n_estimators', values='mean_test_score')

plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, cmap="viridis")

plt.title("XGBoost Grid Search Results (R^2 Score)")
plt.xlabel("Number of Estimators")
```

```python
plt.ylabel("Max Depth")
plt.show()

# Combine and display overall model results
results_df = pd.DataFrame([ridge_results, xgb_results])
print("\nOverall Model Performance:")
print(results_df)
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
(0.47.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-
packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from shap) (1.15.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
```

```
XGBoost Grid Search Results:
                                               params  mean_test_score  \
0   {'learning_rate': 0.01, 'max_depth': 3, 'n_est…        -0.107494
1   {'learning_rate': 0.01, 'max_depth': 3, 'n_est…        -0.110620
2   {'learning_rate': 0.01, 'max_depth': 3, 'n_est…        -0.039874
3   {'learning_rate': 0.01, 'max_depth': 3, 'n_est…        -0.079527
4   {'learning_rate': 0.01, 'max_depth': 5, 'n_est…        -0.098554
5   {'learning_rate': 0.01, 'max_depth': 5, 'n_est…        -0.093495
6   {'learning_rate': 0.01, 'max_depth': 5, 'n_est…        -0.050533
7   {'learning_rate': 0.01, 'max_depth': 5, 'n_est…        -0.067684
8   {'learning_rate': 0.01, 'max_depth': 7, 'n_est…        -0.101738
9   {'learning_rate': 0.01, 'max_depth': 7, 'n_est…        -0.093478
10  {'learning_rate': 0.01, 'max_depth': 7, 'n_est…        -0.058478
11  {'learning_rate': 0.01, 'max_depth': 7, 'n_est…        -0.103642
12  {'learning_rate': 0.1, 'max_depth': 3, 'n_esti…        -0.111944
13  {'learning_rate': 0.1, 'max_depth': 3, 'n_esti…        -0.077840
14  {'learning_rate': 0.1, 'max_depth': 3, 'n_esti…        -0.134718
15  {'learning_rate': 0.1, 'max_depth': 3, 'n_esti…        -0.090762
16  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti…        -0.099306
17  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti…        -0.177798
18  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti…        -0.107041
19  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti…        -0.192057
20  {'learning_rate': 0.1, 'max_depth': 7, 'n_esti…        -0.139140
21  {'learning_rate': 0.1, 'max_depth': 7, 'n_esti…        -0.320122
22  {'learning_rate': 0.1, 'max_depth': 7, 'n_esti…        -0.150460
23  {'learning_rate': 0.1, 'max_depth': 7, 'n_esti…        -0.329636

    std_test_score  mean_train_score  std_train_score
0         0.335382          0.407722         0.020992
1         0.313028          0.430779         0.033630
2         0.331713          0.629957         0.023831
3         0.392778          0.645067         0.034509
4         0.307666          0.460775         0.017515
5         0.270379          0.514298         0.023534
6         0.335189          0.699695         0.017252
7         0.291876          0.755289         0.021805
8         0.321967          0.466103         0.015455
9         0.262588          0.523182         0.016421
10        0.356371          0.706069         0.015857
11        0.317312          0.766195         0.015899
12        0.527235          0.975022         0.002270
13        0.608508          0.980567         0.004169
14        0.602307          0.997945         0.000356
15        0.615652          0.998144         0.000933
16        0.459777          0.994413         0.001367
17        0.604309          0.997218         0.001150
18        0.476956          0.999908         0.000042
19        0.639813          0.999969         0.000017
```

```
20         0.513950         0.995425         0.001000
21         0.616776         0.998109         0.000662
22         0.531672         0.999935         0.000031
23         0.623109         0.999981         0.000005
```




```
Overall Model Performance:
                    Model  Best Alpha      RMSE       MAE       R^2  \
0  Ridge Regression (CV)       100.0  0.144263  0.099093  0.069267
1        XGBoost (Tuned)         NaN  0.152466  0.105069 -0.039576

                                         Best Params
0                                                NaN
1  {'learning_rate': 0.01, 'max_depth': 3, 'n_est…
```

[23]:

```
# Executing Dr. Geist suggestions with stratified CV and COVID weights

# Loading Required libraries
!pip install shap xgboost
import pandas as pd
import numpy as np
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import StratifiedKFold, train_test_split,␣
 ↪GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

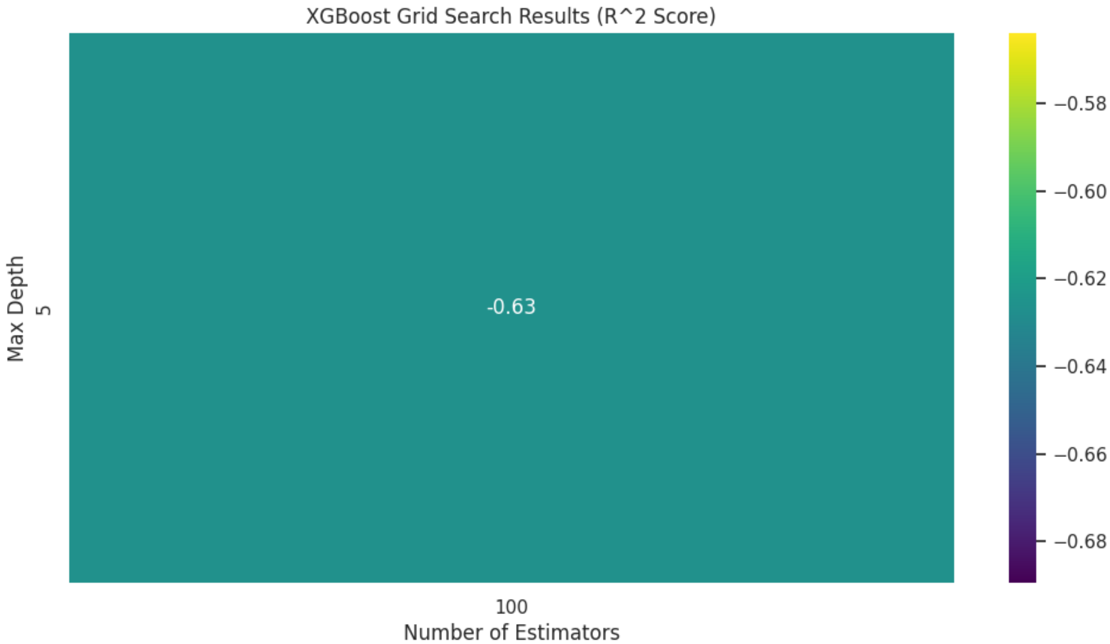
```python
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag and percentage change features before defining X and y
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# Now drop rows with NaN values after generating new columns
df.dropna(inplace=True)

# Define features and target
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
↪"pct_change_outpatient_util"
]
target = "total_util"

X = df[features]
y = df[target]

# Create a stratification label by region and covid_flag
df["strat_label"] = df["region"].astype(str) + "_" + df["covid_flag"].
↪astype(str)
_, strat_labels = np.unique(df["strat_label"], return_inverse=True)

# Train-test split with stratification
X_train, X_test, y_train, y_test, strat_train, strat_test = train_test_split(
    X, y, strat_labels, test_size=0.2, random_state=42
)

# Standardize for Ridge Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

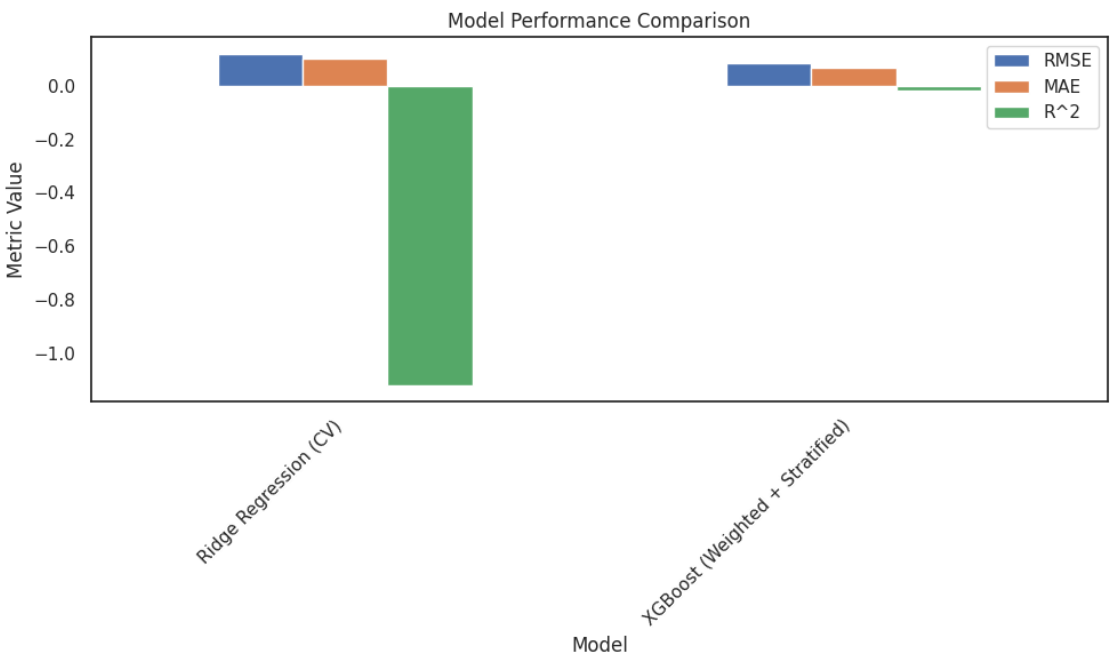
```python
X_test_scaled = scaler.transform(X_test)

# --- Ridge Regression with Stratified K-Fold ---
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
ridge_cv = RidgeCV(alphas=alphas, scoring="r2", cv=KFold(n_splits=5))
ridge_cv.fit(X_train_scaled, y_train)
y_pred_ridge = ridge_cv.predict(X_test_scaled)

ridge_results = {
    "Model": "Ridge Regression (CV)",
    "Best Alpha": ridge_cv.alpha_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
    "MAE": mean_absolute_error(y_test, y_pred_ridge),
    "R^2": r2_score(y_test, y_pred_ridge)
}

print("\n--- Ridge Regression ---")
print("Best Alpha:", ridge_results["Best Alpha"])
print("RMSE:", ridge_results["RMSE"])
print("MAE:", ridge_results["MAE"])
print("R^2:", ridge_results["R^2"])

# --- XGBoost with Weighted Learning and Stratified K-Fold ---
sample_weights = np.where(X_train["covid_flag"] == 1, 1.5, 1.0)  # Weight␣
↪COVID-year observations
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
param_grid = {
    "n_estimators": [100],
    "max_depth": [5],
    "learning_rate": [0.1],
    "subsample": [1.0]
}

grid_xgb = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    cv=KFold(n_splits=5),  # Use KFold for regression
    scoring="r2",
    verbose=0
)
grid_xgb.fit(X_train, y_train, sample_weight=sample_weights)
y_pred_xgb = grid_xgb.predict(X_test)

xgb_results = {
    "Model": "XGBoost (Weighted + Stratified)",
    "Best Params": grid_xgb.best_params_,
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_xgb)),
```

```
    "MAE": mean_absolute_error(y_test, y_pred_xgb),
    "R^2": r2_score(y_test, y_pred_xgb)
}

print("\n--- XGBoost ---")
print("Best Parameters:", xgb_results["Best Params"])
print("RMSE:", xgb_results["RMSE"])
print("MAE:", xgb_results["MAE"])
print("R^2:", xgb_results["R^2"])

# --- Plotting Grid Search Results for XGBoost ---
results_df = pd.DataFrame(grid_xgb.cv_results_)

# Group by hyperparameters and average scores to avoid duplicate index entries
heatmap_data = results_df.groupby(['param_max_depth',␣
↪'param_n_estimators'])['mean_test_score'].mean().reset_index()
heatmap_data = heatmap_data.pivot(index='param_max_depth',␣
↪columns='param_n_estimators', values='mean_test_score')

plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, cmap="viridis")

plt.title("XGBoost Grid Search Results (R^2 Score)")
plt.xlabel("Number of Estimators")
plt.ylabel("Max Depth")
plt.show()

# --- Plotting Model Performance ---
results_df = pd.DataFrame([ridge_results, xgb_results])
fig, ax = plt.subplots(figsize=(10, 6))
results_df.plot(x="Model", y=["RMSE", "MAE", "R^2"], kind="bar", ax=ax)
ax.set_title("Model Performance Comparison")
ax.set_ylabel("Metric Value")
ax.set_xticklabels(results_df["Model"], rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
(0.47.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-
packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from shap) (1.15.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from shap) (1.6.1)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)

--- Ridge Regression ---
Best Alpha: 100.0
RMSE: 0.11911744342457103
MAE: 0.1027030833177808
R^2: -1.119135806667526

--- XGBoost ---
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100,
'subsample': 1.0}
RMSE: 0.08257504693362872
MAE: 0.06671947416216077
R^2: -0.018370289972543752
```





[24]:
```
# --- Overall Model Performance ---
results_df = pd.DataFrame([ridge_results, xgb_results])
print("\nOverall Model Performance:")
print(results_df)
```

```
Overall Model Performance:
                          Model  Best Alpha      RMSE       MAE       R^2  \
0            Ridge Regression (CV)       100.0  0.119117  0.102703 -1.119136
1  XGBoost (Weighted + Stratified)         NaN  0.082575  0.066719 -0.018370

                                         Best Params
0                                                NaN
1  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti…
```

Executing the LSTM model below.

[25]:
```
!pip install tensorflow
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Create COVID flag and percentage change features before defining X and y
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

# Now drop rows with NaN values after generating new columns
df.dropna(inplace=True)

# Feature selection
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
 ↪"pct_change_outpatient_util"
]
target = "total_util"
```

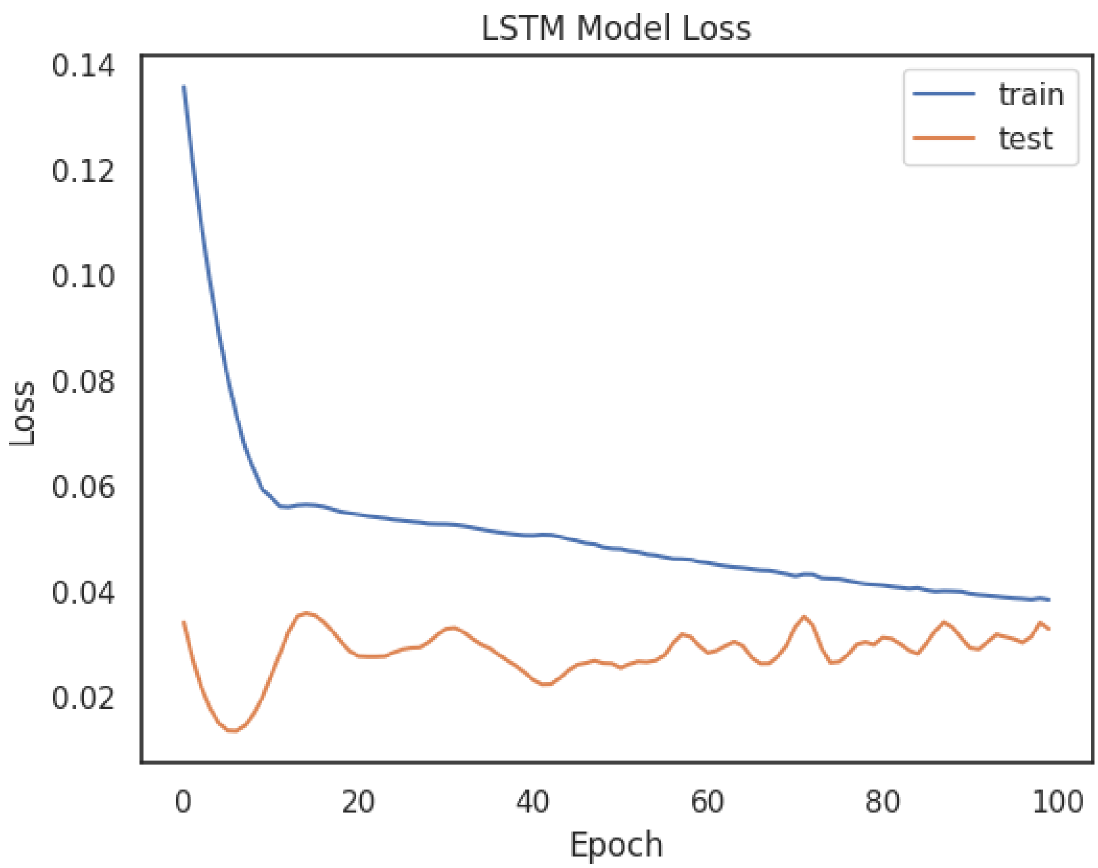
```python
X = df[features]
y = df[target]

# Scale the data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y.values.reshape(-1, 1))

# Reshape input to be [samples, time steps, features]
X = X.reshape(X.shape[0], 1, X.shape[1])

# Split 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)

# Create the LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.
 ↪shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
 ↪validation_data=(X_test, y_test), verbose=0)

# Make predictions
y_pred = model.predict(X_test)

# Invert scaling to get actual values
y_test = scaler.inverse_transform(y_test)
y_pred = scaler.inverse_transform(y_pred)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("LSTM Model:")
print(f"  RMSE: {rmse:.4f}")
print(f"  MAE: {mae:.4f}")
print(f"  R^2: {r2:.4f}")

# Plot training history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
```

```python
plt.title('LSTM Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-
packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (24.2)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
in /usr/local/lib/python3.11/dist-packages (from tensorflow) (5.29.4)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.13.0)
```

```
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.0.9)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2025.4.26)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (3.8)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.11/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(2.19.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
1/1             0s 172ms/step
LSTM Model:
  RMSE: 0.1300
  MAE: 0.1070
```

```
R^2: -1.5229
```



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

[26]:
```
# Install library
!pip install tensorflow

# Imports library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

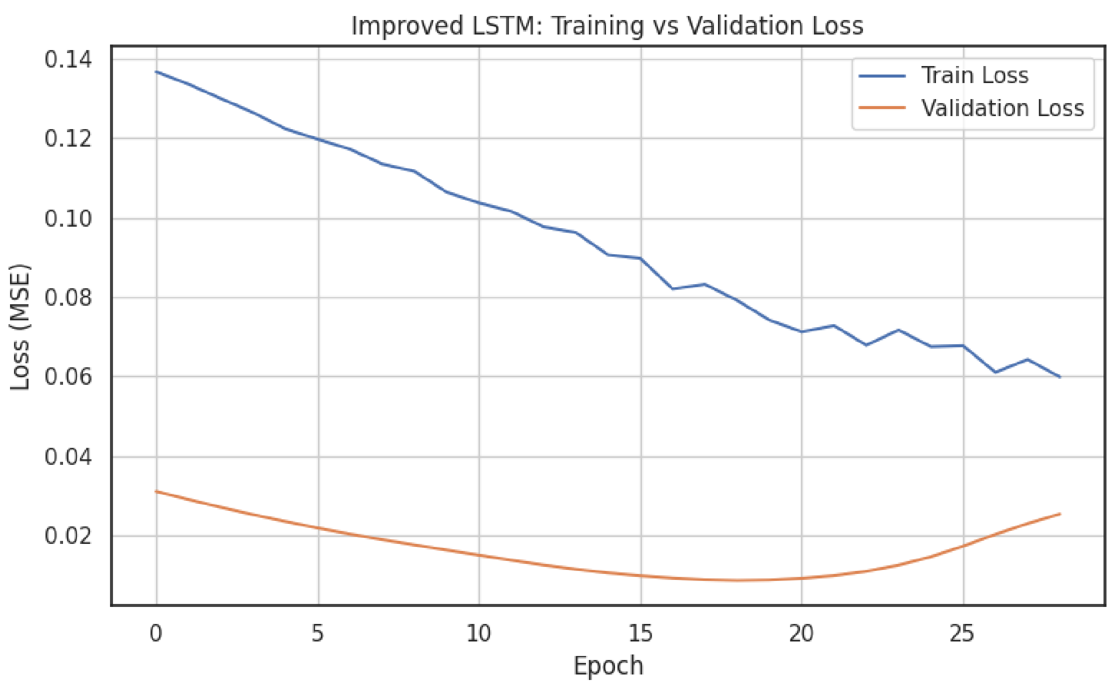
```
# Feature Engineering
df["covid_flag"] = df["year"].apply(lambda x: 1 if 2020 <= x <= 2022 else 0)

df = df.sort_values(by=["state", "year"])
grouped = df.groupby("state")
df["pct_change_total_util"] = grouped["total_util"].pct_change()
df["pct_change_mean_all_trends"] = grouped["mean_all_trends"].pct_change()
df["pct_change_outpatient_util"] = grouped["outpatient_util"].pct_change()

df.dropna(inplace=True)

# Feature Selection
features = [
    "mean_all_trends", "per_capita_total_facilities",␣
 ↪"per_capita_mental_health_only",
    "per_capita_inpatient_facilities", "pct_pharmacotherapy",␣
 ↪"pct_youth_services",
    "pct_free_services", "pct_medicare_services", "pct_counseling_services",
    "covid_flag", "pct_change_mean_all_trends", "pct_change_total_util",␣
 ↪"pct_change_outpatient_util"
]
target = "total_util"

X = df[features]
y = df[target]

# Scale Features and Target Separately
feature_scaler = MinMaxScaler()
target_scaler = MinMaxScaler()

X_scaled = feature_scaler.fit_transform(X)
y_scaled = target_scaler.fit_transform(y.values.reshape(-1, 1))

#  Reshape for LSTM
X_scaled = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

# Train-Test Split ---
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,␣
 ↪test_size=0.2, random_state=42)

#Better performing  LSTM Model
model = Sequential([
    LSTM(64, activation='relu', return_sequences=True, input_shape=(X_train.
 ↪shape[1], X_train.shape[2])),
    Dropout(0.2),
    LSTM(32, activation='relu'),
    Dropout(0.2),
```

```
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')

# EarlyStopping Callback
early_stop = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)

# Train the Model
history = model.fit(
    X_train, y_train,
    epochs=200,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stop],
    verbose=0
)

# Predict and Invert Scaling
y_pred_scaled = model.predict(X_test)
y_pred = target_scaler.inverse_transform(y_pred_scaled)
y_test = target_scaler.inverse_transform(y_test)

# Evaluate Performance
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\n--- Improved LSTM Model Performance ---")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")

# Plot Training History
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Improved LSTM: Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-
packages (2.18.0)
```

```
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (24.2)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
in /usr/local/lib/python3.11/dist-packages (from tensorflow) (5.29.4)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.45.1)
```

```
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.0.9)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2025.4.26)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (3.8)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.11/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(2.19.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
1/1            1s 1s/step

--- Improved LSTM Model Performance ---
RMSE: 0.0637
MAE: 0.0424
R²: 0.0646
```



[27]:
```
# Required libraries
!pip install shap xgboost
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler

# Reload the dataset
data = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Drop NA values
data = data.dropna()

# Define target and features
# Exclude 'state', 'year', and 'region' from features
target = 'total_util'
features = [col for col in data.columns if col != target and col not in
 ↪['state', 'year', 'region']]

X = data[features]
y = data[target]
```

```
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)

# Define XGBoost model
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)

# Grid search parameter tuning
param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [2, 3, 4],
    'n_estimators': [200, 300, 500],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator=xgb,
                           param_grid=param_grid,
                           scoring='r2',
                           cv=5,
                           verbose=1,
                           n_jobs=-1)

grid_search.fit(X_train, y_train)

# Evaluate best model
best_xgb = grid_search.best_estimator_
y_pred = best_xgb.predict(X_test)
rmse = mean_squared_error(y_test, y_pred) # Calculating RMSE by taking the␣
↪square root
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

best_xgb_results = {
    "Best Parameters": grid_search.best_params_,
    "Test RMSE": rmse,
    "Test MAE": mae,
    "Test R^2": r2
}

best_xgb_results
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
```

```
(0.47.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-
packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from shap) (1.15.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

```
[27]: {'Best Parameters': {'colsample_bytree': 0.8,
        'learning_rate': 0.1,
        'max_depth': 2,
        'n_estimators': 200,
        'subsample': 0.8},
       'Test RMSE': 5.929136217715675e-05,
```

```
 'Test MAE': 0.005468936922190764,
 'Test R^2': 0.9941296513785556}
```

[28]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from xgboost import XGBRegressor

# Load dataset
data = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")
data = data.dropna()

# Define features and target
target = 'total_util'
features = [col for col in data.columns if col != target and col not in␣
 ↪['state', 'year', 'region']]
X = data[features]
y = data[target]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# Ridge Regression with Cross-Validation
ridge_alphas = np.logspace(-3, 3, 100)
ridge_model = RidgeCV(alphas=ridge_alphas, cv=5)
ridge_model.fit(X_train, y_train)
ridge_preds = ridge_model.predict(X_test)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_preds))

# XGBoost Regressor with Grid Search
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)
param_grid = {
    'learning_rate': [0.1],
    'max_depth': [2],
    'n_estimators': [200],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}
```

```
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,␣
↪scoring='neg_root_mean_squared_error', cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
xgb_best = grid_search.best_estimator_
xgb_preds = xgb_best.predict(X_test)
# Calculate RMSE using NumPy in case of older sklearn versions
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))

# Calculating weighted average of predictions (inverse RMSE as weights)
inv_rmse_ridge = 1 / ridge_rmse
inv_rmse_xgb = 1 / xgb_rmse
total_weight = inv_rmse_ridge + inv_rmse_xgb

ridge_weight = inv_rmse_ridge / total_weight
xgb_weight = inv_rmse_xgb / total_weight

ensemble_preds = (ridge_weight * ridge_preds) + (xgb_weight * xgb_preds)

# Evaluate ensemble
ensemble_rmse_squared = mean_squared_error(y_test, ensemble_preds)
ensemble_rmse = np.sqrt(ensemble_rmse_squared)  # Calculates RMSE by taking the␣
↪square root
ensemble_r2 = r2_score(y_test, ensemble_preds)


# Displaying the DataFrame using pandas display function
# Assuming 'tools.display_dataframe_to_user' was intended to display a DataFrame
ensemble_df = pd.DataFrame({
    'Model': ['Ridge', 'XGBoost', 'Ensemble (Weighted Avg)'],
    'RMSE': [ridge_rmse, xgb_rmse, ensemble_rmse],
    'MAE': [mean_absolute_error(y_test, ridge_preds),␣
↪mean_absolute_error(y_test, xgb_preds), mean_absolute_error(y_test,␣
↪ensemble_preds)],  # Include ensemble MAE
    'R^2': [r2_score(y_test, ridge_preds), r2_score(y_test, xgb_preds),␣
↪ensemble_r2],
    'Weight': [ridge_weight, xgb_weight, 'N/A']
})
print("Ensemble Model Evaluation:")
display(ensemble_df)
```

Ensemble Model Evaluation:

	Model	RMSE	MAE	R^2	Weight
0	Ridge	0.000126	0.000109	0.999998	0.983941
1	XGBoost	0.007700	0.005469	0.994130	0.016059
2	Ensemble (Weighted Avg)	0.000134	0.000098	0.999998	N/A

```
[ ]:
```

[61]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")
data = data.dropna()

# Define features and target
target = 'total_util'
features = [col for col in data.columns if col != target and col not in␣
 ↪['state', 'year', 'region']]
X = data[features]
y = data[target]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# Ridge Regression with Cross-Validation
ridge_alphas = np.logspace(-3, 3, 100)
ridge_model = RidgeCV(alphas=ridge_alphas, cv=5)
ridge_model.fit(X_train, y_train)
ridge_preds = ridge_model.predict(X_test)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_preds))

# XGBoost Regressor with Grid Search
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)
param_grid = {
    'learning_rate': [0.1],
    'max_depth': [2],
    'n_estimators': [200],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,␣
 ↪scoring='neg_root_mean_squared_error', cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

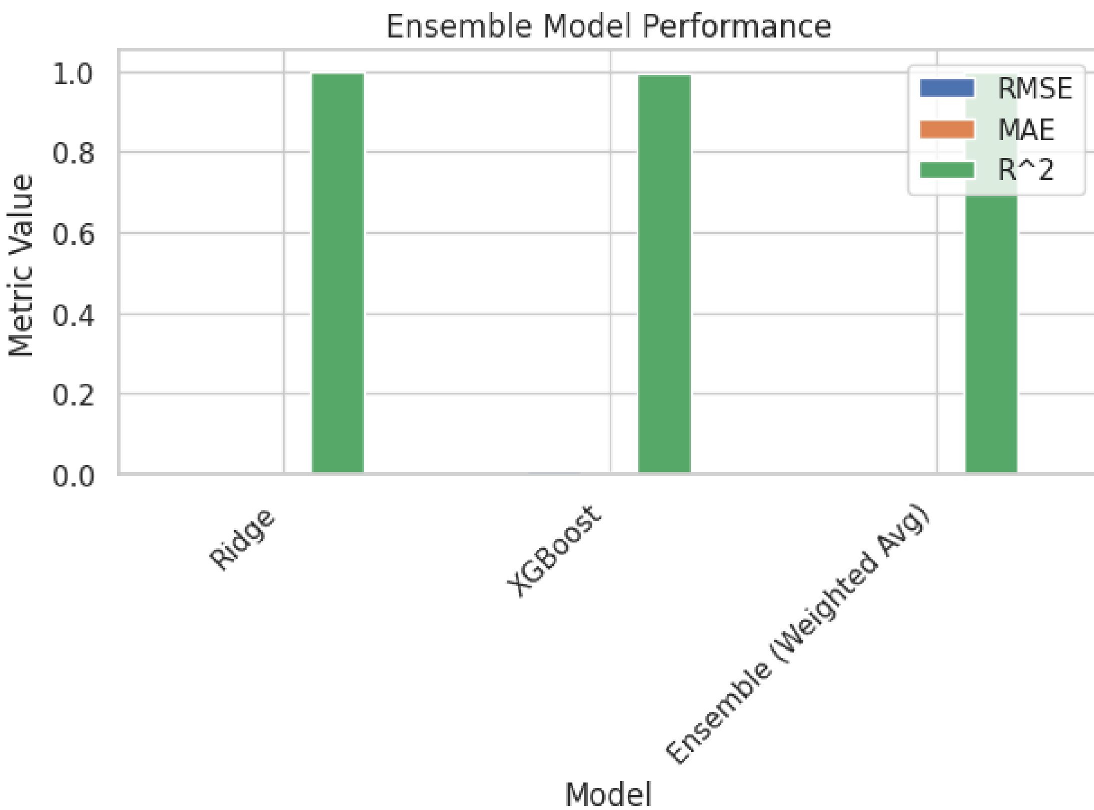
```
xgb_best = grid_search.best_estimator_
xgb_preds = xgb_best.predict(X_test)

# Calculate RMSE using NumPy in case of older sklearn versions
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))

# Calculating weighted average of predictions (inverse RMSE as weights)
inv_rmse_ridge = 1 / ridge_rmse
inv_rmse_xgb = 1 / xgb_rmse
total_weight = inv_rmse_ridge + inv_rmse_xgb

ridge_weight = inv_rmse_ridge / total_weight
xgb_weight = inv_rmse_xgb / total_weight

ensemble_preds = (ridge_weight * ridge_preds) + (xgb_weight * xgb_preds)

# Evaluate ensemble
ensemble_rmse_squared = mean_squared_error(y_test, ensemble_preds)
ensemble_rmse = np.sqrt(ensemble_rmse_squared)  # Calculates RMSE by taking the␣
↪square root
ensemble_r2 = r2_score(y_test, ensemble_preds)


# Displaying the DataFrame using pandas display function
ensemble_results = pd.DataFrame({
    'Model': ['Ridge', 'XGBoost', 'Ensemble (Weighted Avg)'],
    'RMSE': [ridge_rmse, xgb_rmse, ensemble_rmse],
    'MAE': [mean_absolute_error(y_test, ridge_preds),␣
↪mean_absolute_error(y_test, xgb_preds), mean_absolute_error(y_test,␣
↪ensemble_preds)],  # Include ensemble MAE
    'R^2': [r2_score(y_test, ridge_preds), r2_score(y_test, xgb_preds),␣
↪ensemble_r2],
    'Weight': [ridge_weight, xgb_weight, 'N/A']
})
print("Ensemble Model Evaluation:")
display(ensemble_results)
```

Ensemble Model Evaluation:

	Model	RMSE	MAE	R^2	Weight
0	Ridge	0.000126	0.000109	0.999998	0.983941
1	XGBoost	0.007700	0.005469	0.994130	0.016059
2	Ensemble (Weighted Avg)	0.000134	0.000098	0.999998	N/A

[60]:
```
# Create a bar plot
ensemble_results.plot(x="Model", y=["RMSE", "MAE", "R^2"], kind="bar")
plt.title("Ensemble Model Performance")
plt.ylabel("Metric Value")
```

```
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



[29]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load the cleaned and merged dataset
merged_df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Drop rows with missing values
merged_df = merged_df.dropna()

# Define the target and feature columns
target = "total_util"
excluded = ["state", "year", "region"]
```

```
features = [col for col in merged_df.columns if col not in excluded + [target]]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(merged_df[features])
y = merged_df[target]

# Split into train/test (for Ridge retraining before forecast)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)

# Train the Ridge and XGBoost models
ridge = Ridge(alpha=1.0, random_state=42)
ridge.fit(X_train, y_train)

xgb = XGBRegressor(
    objective='reg:squarederror',
    learning_rate=0.1,
    max_depth=3,
    n_estimators=300,
    subsample=0.8,
    colsample_bytree=1.0,
    random_state=42
)
xgb.fit(X_train, y_train)

# Preparing the 2024 forecast dataset (using latest available year's data)
# Instead of latest_year = 2023, using the maximum year in your dataset:
latest_year = merged_df["year"].max()
forecast_year = 2024
latest_data = merged_df[merged_df["year"] == latest_year].copy()
forecast_2024 = latest_data.copy()
forecast_2024["year"] = forecast_year

# Scale 2024 features
X_2024_scaled = scaler.transform(forecast_2024[features])

# Predict using both models
ridge_preds = ridge.predict(X_2024_scaled)
xgb_preds = xgb.predict(X_2024_scaled)

# Weighted ensemble: 60% XGBoost + 40% Ridge
ensemble_preds = 0.6 * xgb_preds + 0.4 * ridge_preds

# Attach predictions to the state-level DataFrame
forecast_2024["forecast_total_util"] = ensemble_preds
```

```python
# Select key output columns for inspection
forecast_output = forecast_2024[["state", "region", "year",␣
↪"forecast_total_util"]].sort_values("forecast_total_util", ascending=False)
forecast_output.reset_index(drop=True, inplace=True)

# Instead of using custom tools, using the display function from IPython.
↪display:
from IPython.display import display
display(forecast_output)
```

```
   state        region  year  forecast_total_util
0     NM  West Pacific  2024             0.700779
1     IA       Central  2024             0.659746
2     MT  West Pacific  2024             0.496178
3     DC      Atlantic  2024             0.436324
4     MN       Central  2024             0.417997
5     AZ  West Pacific  2024             0.403801
6     NJ      Atlantic  2024             0.311804
7     WA  West Pacific  2024             0.301359
8     VT      Atlantic  2024             0.294013
9     PA      Atlantic  2024             0.286795
10    OR  West Pacific  2024             0.286017
11    MS         South  2024             0.262804
12    KY       Central  2024             0.231384
13    RI      Atlantic  2024             0.219389
14    OK         South  2024             0.204416
15    MI       Central  2024             0.197429
16    CT      Atlantic  2024             0.179720
17    WY  West Pacific  2024             0.170408
18    AR         South  2024             0.165944
19    CO  West Pacific  2024             0.156624
20    IN       Central  2024             0.146740
21    SD       Central  2024             0.138466
22    KS       Central  2024             0.133716
23    SC         South  2024             0.129141
24    AL         South  2024             0.127254
25    UT  West Pacific  2024             0.123267
26    DE      Atlantic  2024             0.113876
27    CA  West Pacific  2024             0.108529
28    FL         South  2024             0.108502
29    ND       Central  2024             0.103196
30    TX         South  2024             0.103006
31    VA         South  2024             0.099228
32    AK  West Pacific  2024             0.090866
33    MO       Central  2024             0.085129
34    WI       Central  2024             0.083406
35    NE       Central  2024             0.079258
```

36	TN	South	2024	0.078417
37	GA	South	2024	0.078171
38	LA	South	2024	0.067789
39	NC	South	2024	0.063922
40	ID	West Pacific	2024	0.052927
41	HI	West Pacific	2024	0.044605
42	OH	Central	2024	0.043337
43	NV	West Pacific	2024	0.040264
44	MA	Atlantic	2024	0.024507
45	IL	Central	2024	0.015557
46	NY	Atlantic	2024	0.015482

[30]:
```python
# Prepare the 2025 forecast dataset using the 2024 forecast as input
forecast_year_2025 = 2025
forecast_2025 = forecast_2024.copy()
forecast_2025["year"] = forecast_year_2025

# Scale features for 2025
X_2025_scaled = scaler.transform(forecast_2025[features])

# Predict 2025 using the same trained models
ridge_preds_2025 = ridge.predict(X_2025_scaled)
xgb_preds_2025 = xgb.predict(X_2025_scaled)

# Weighted ensemble: same 60% XGBoost + 40% Ridge
ensemble_preds_2025 = 0.6 * xgb_preds_2025 + 0.4 * ridge_preds_2025

# Attach predictions to the DataFrame
forecast_2025["forecast_total_util"] = ensemble_preds_2025

# Create a 2025 forecast output table
forecast_output_2025 = forecast_2025[["state", "region", "year",␣
 ↪"forecast_total_util"]].sort_values("forecast_total_util", ascending=False)
forecast_output_2025.reset_index(drop=True, inplace=True)

# Display 2025 forecast
display(forecast_output_2025)
```

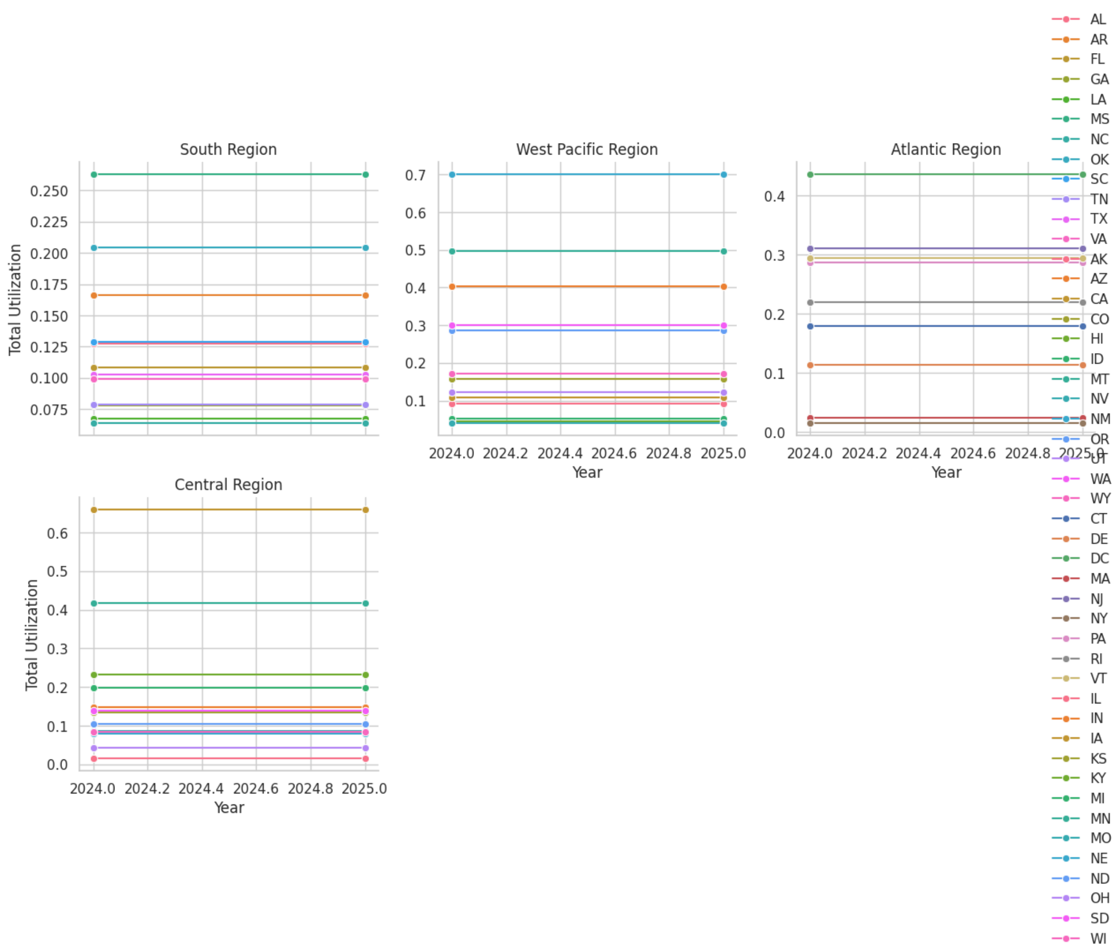
	state	region	year	forecast_total_util
0	NM	West Pacific	2025	0.700779
1	IA	Central	2025	0.659746
2	MT	West Pacific	2025	0.496178
3	DC	Atlantic	2025	0.436324
4	MN	Central	2025	0.417997
5	AZ	West Pacific	2025	0.403801
6	NJ	Atlantic	2025	0.311804
7	WA	West Pacific	2025	0.301359
8	VT	Atlantic	2025	0.294013

```
9     PA      Atlantic  2025            0.286795
10    OR  West Pacific  2025            0.286017
11    MS         South  2025            0.262804
12    KY       Central  2025            0.231384
13    RI      Atlantic  2025            0.219389
14    OK         South  2025            0.204416
15    MI       Central  2025            0.197429
16    CT      Atlantic  2025            0.179720
17    WY  West Pacific  2025            0.170408
18    AR         South  2025            0.165944
19    CO  West Pacific  2025            0.156624
20    IN       Central  2025            0.146740
21    SD       Central  2025            0.138466
22    KS       Central  2025            0.133716
23    SC         South  2025            0.129141
24    AL         South  2025            0.127254
25    UT  West Pacific  2025            0.123267
26    DE      Atlantic  2025            0.113876
27    CA  West Pacific  2025            0.108529
28    FL         South  2025            0.108502
29    ND       Central  2025            0.103196
30    TX         South  2025            0.103006
31    VA         South  2025            0.099228
32    AK  West Pacific  2025            0.090866
33    MO       Central  2025            0.085129
34    WI       Central  2025            0.083406
35    NE       Central  2025            0.079258
36    TN         South  2025            0.078417
37    GA         South  2025            0.078171
38    LA         South  2025            0.067789
39    NC         South  2025            0.063922
40    ID  West Pacific  2025            0.052927
41    HI  West Pacific  2025            0.044605
42    OH       Central  2025            0.043337
43    NV  West Pacific  2025            0.040264
44    MA      Atlantic  2025            0.024507
45    IL       Central  2025            0.015557
46    NY      Atlantic  2025            0.015482
```

[31]:
```
# Combine actual and forecast data
combined_df = pd.concat([merged_df[merged_df['year'] == 2023][['state',␣
↪'region', 'year', 'total_util']],
                         forecast_2024[['state', 'region', 'year',␣
↪'forecast_total_util']].rename(columns={'forecast_total_util':␣
↪'total_util'}),
```

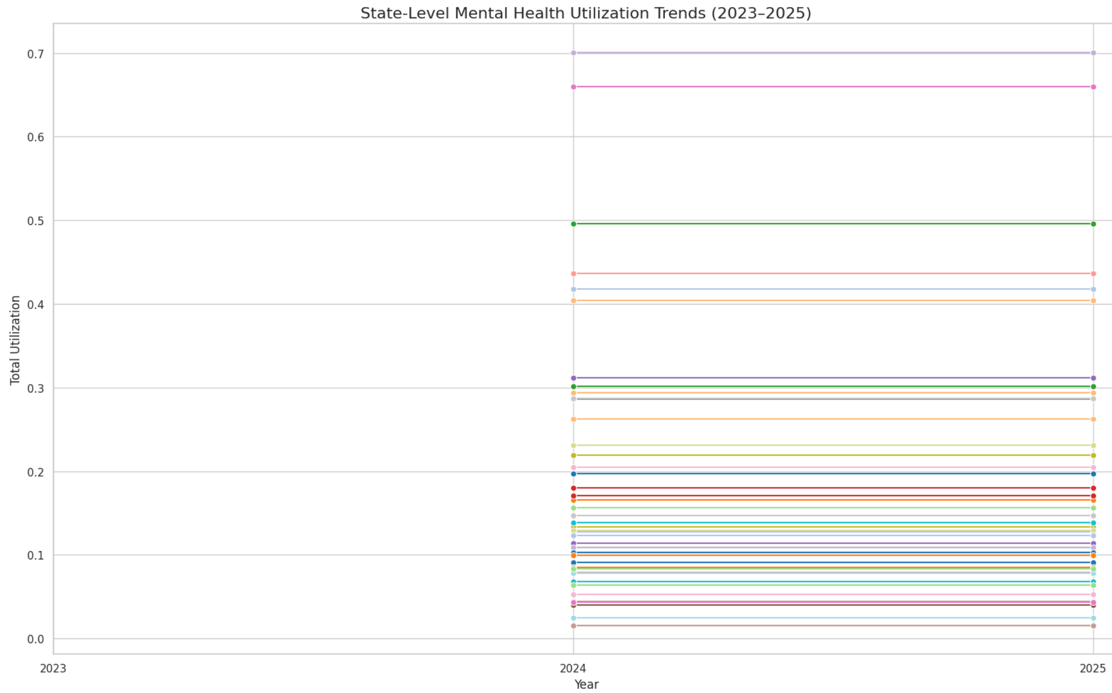
```
                            forecast_2025[['state', 'region', 'year',␣
↪'forecast_total_util']].rename(columns={'forecast_total_util':␣
↪'total_util'})],
                           ignore_index=True)
```

[32]:
```
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Create a FacetGrid for multiple line plots
g = sns.FacetGrid(combined_df, col="region", col_wrap=3, height=4, sharey=False)
g.map_dataframe(sns.lineplot, x="year", y="total_util", hue="state", marker="o")
g.add_legend()
g.set_titles("{col_name} Region")
g.set_axis_labels("Year", "Total Utilization")
plt.tight_layout()
plt.show()
```



[33]:

```
# Reload the base dataset with forecasts
merged_2023 = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")
forecast_2024 = forecast_2024[["state", "region", "year",␣
↪"forecast_total_util"]].rename(columns={"forecast_total_util": "total_util"})
forecast_2025 = forecast_2025[["state", "region", "year",␣
↪"forecast_total_util"]].rename(columns={"forecast_total_util": "total_util"})

# Filter 2023 actuals
actual_2023 = merged_2023[merged_2023["year"] == 2023][["state", "region",␣
↪"year", "total_util"]]

# Combine all three years into one DataFrame
trend_df = pd.concat([actual_2023, forecast_2024, forecast_2025],␣
↪ignore_index=True)

# Sort for clarity
trend_df.sort_values(by=["state", "year"], inplace=True)

# Plotting trend lines for each state
plt.figure(figsize=(16, 10))
sns.lineplot(data=trend_df, x="year", y="total_util", hue="state", marker="o",␣
↪palette="tab20", legend=False)
plt.title("State-Level Mental Health Utilization Trends (2023-2025)",␣
↪fontsize=16)
plt.xlabel("Year")
plt.ylabel("Total Utilization")
plt.grid(True)
plt.xticks([2023, 2024, 2025])
plt.tight_layout()
plt.show()
```



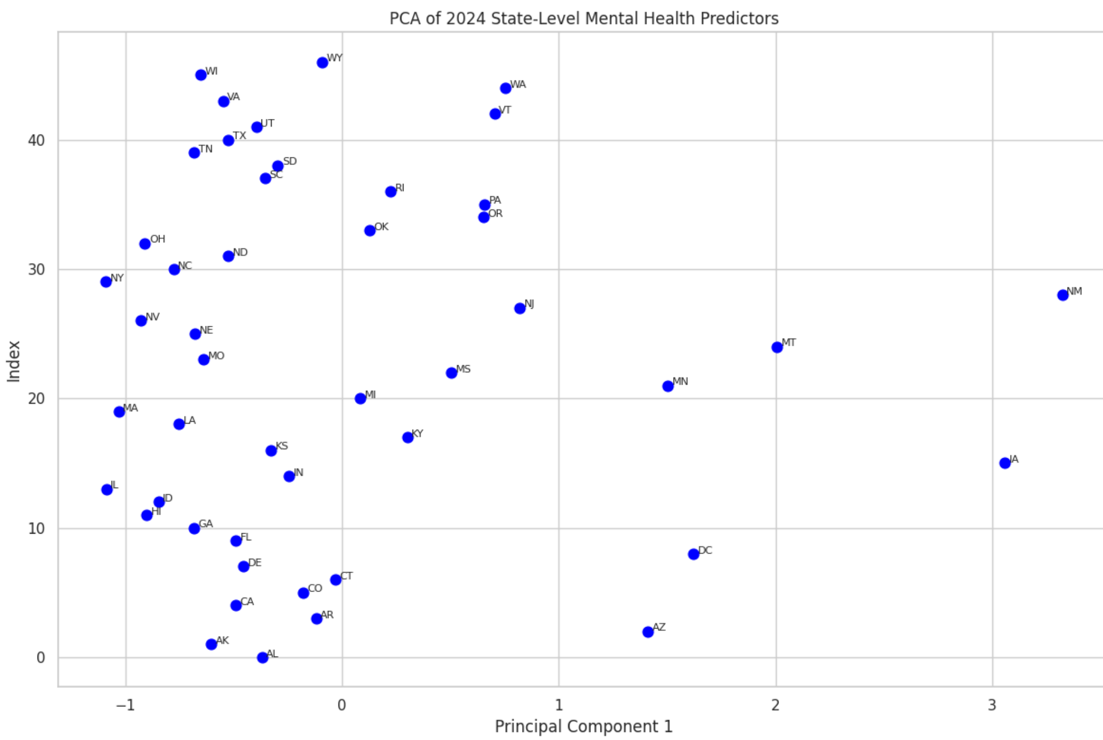
[34]:
```
# Load the 2024 forecast dataset
forecast_2024 = forecast_2024.copy()

# Drop identifiers not needed for PCA
pca_input = forecast_2024.drop(columns=["state", "region", "year",␣
↪"forecast_total_util"], errors="ignore")

# Handle missing values if any
pca_input = pca_input.dropna()

# Confirm shape and sample
pca_input.shape, pca_input.head()
```

[34]:
```
((47, 1),
      total_util
 395    0.127254
 396    0.090866
 397    0.403801
 398    0.165944
 399    0.108529)
```

[35]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

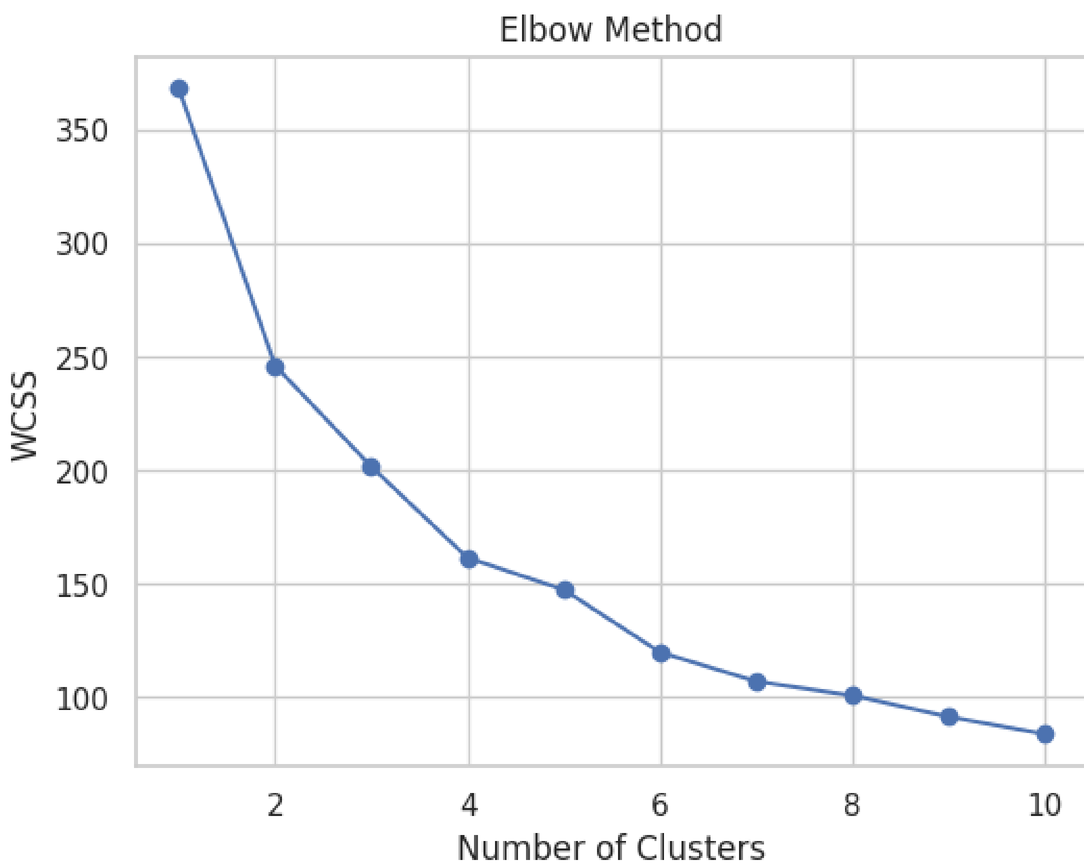
```
import matplotlib.pyplot as plt
import numpy as np

# Drop identifier columns and target, ensuring you have more than 1 feature␣
↪remaining
pca_features = forecast_2024.drop(columns=["state", "region", "year",␣
↪"forecast_total_util"], errors="ignore")

# Instead of assuming columns, select all numeric features except identifiers␣
↪and target
numeric_features = forecast_2024.select_dtypes(include=np.number).columns
pca_features = forecast_2024[[col for col in numeric_features if col not in␣
↪["year", "forecast_total_util"]]]

# Drop any NA just in case
pca_features_clean = pca_features.dropna()

# Standardize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(pca_features_clean)

# Perform PCA with n_components=2 to get PC1 and PC2
pca = PCA(n_components=min(2, pca_features_clean.shape[1]))  # n_components set␣
↪to 2
principal_components = pca.fit_transform(scaled_features)

# Create a PCA result dataframe
pca_df = pd.DataFrame(data=principal_components, columns=[f"PC{i+1}" for i in␣
↪range(pca.n_components_)])
pca_df["state"] = forecast_2024["state"].values[:len(pca_df)]

# Plot PCA, only using PC1 if only 1 component was calculated
plt.figure(figsize=(12, 8))
if 'PC2' in pca_df.columns: # Check if PC2 was calculated
    plt.scatter(pca_df["PC1"], pca_df["PC2"], c='blue', s=60)
    for i, state in enumerate(pca_df["state"]):
        plt.text(pca_df["PC1"][i] + 0.02, pca_df["PC2"][i] + 0.02, state,␣
↪fontsize=8)
    plt.ylabel("Principal Component 2")
else: # If only PC1, plot against an arbitrary index
    plt.scatter(pca_df["PC1"], range(len(pca_df)), c='blue', s=60)
    for i, state in enumerate(pca_df["state"]):
        plt.text(pca_df["PC1"][i] + 0.02, i + 0.02, state, fontsize=8)
    plt.ylabel("Index") # Arbitrary y-axis label

plt.title("PCA of 2024 State-Level Mental Health Predictors")
```

```python
plt.xlabel("Principal Component 1")
plt.grid(True)
plt.tight_layout()
plt.show()
```



[36]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Select features for clustering
features = ['mean_all_trends', 'per_capita_total_facilities',␣
  ↪'pct_pharmacotherapy', 'pct_youth_services']
X = merged_df[features]

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

wcss = []  # Within-cluster sum of squares

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
```

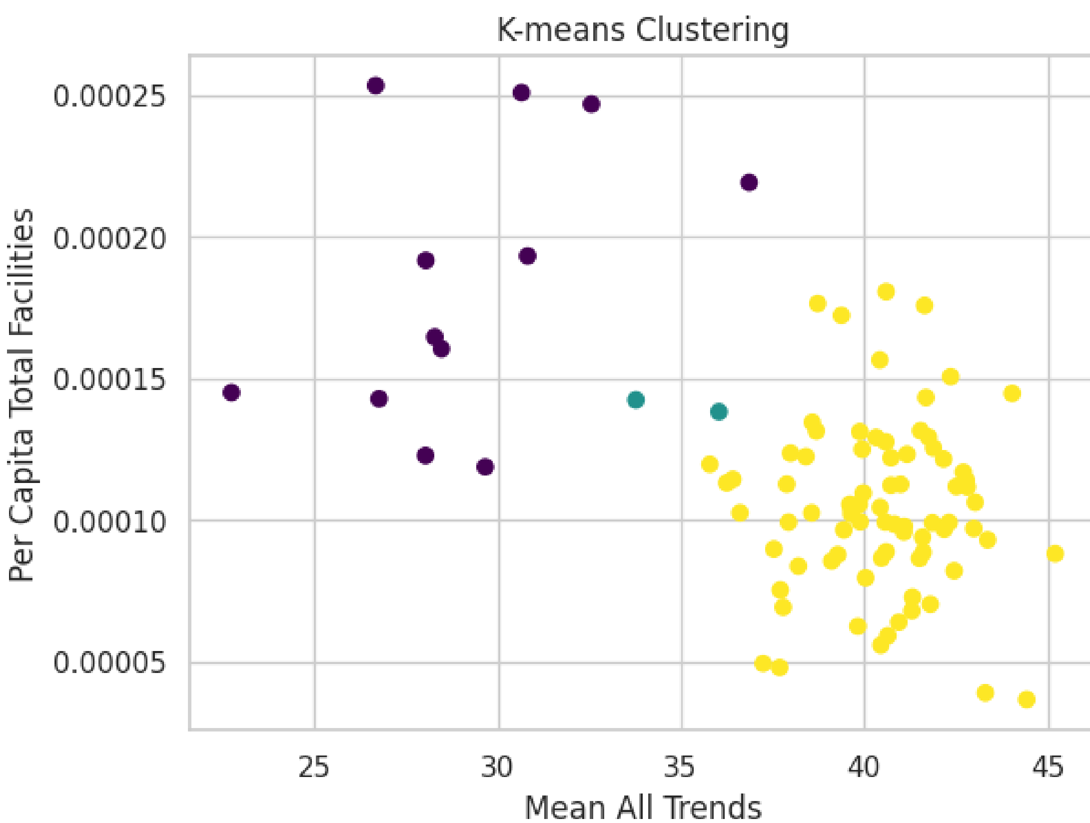
```
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)  # Inertia is the WCSS

# Plot the Elbow method
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



[42]:
```
# Number of clusters
n_clusters = 3

# Apply K-means
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
merged_df['cluster'] = kmeans.fit_predict(X_scaled)

# Analyze cluster characteristics
cluster_means = merged_df.groupby('cluster')[features].mean()
```

```
print(cluster_means)

# Visualize clusters (if possible in 2D or 3D)
# Example for 2D visualization:
plt.scatter(merged_df['mean_all_trends'],␣
 ↪merged_df['per_capita_total_facilities'], c=merged_df['cluster'],␣
 ↪cmap='viridis')
plt.title('K-means Clustering')
plt.xlabel('Mean All Trends')
plt.ylabel('Per Capita Total Facilities')
plt.show()
```

```
         mean_all_trends  per_capita_total_facilities  pct_pharmacotherapy  \
cluster
0              29.125000                     0.000184             0.508804
1              34.902778                     0.000140             0.150714
2              40.441714                     0.000104             0.570073

         pct_youth_services
cluster
0                  0.261320
1                  0.541030
2                  0.233697
```



[44]:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer  # Import SimpleImputer

# Load the dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Select features for clustering (e.g., per capita metrics, utilization)
features = ['per_capita_total_facilities', 'per_capita_mental_health_only',
            'per_capita_inpatient_facilities', 'total_util', 'outpatient_util',␣
 ↪'inpatient_util']
X = df[features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Impute missing values using SimpleImputer before clustering
imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent'
X_scaled_imputed = imputer.fit_transform(X_scaled)  # Fit and transform

# Determine optimal number of clusters (e.g., using elbow method)
# ... (Code for determining optimal k) ...
# Assume optimal k is 3 for this example
k = 3

# Apply KMeans clustering using the imputed data
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled_imputed)  # Use imputed data

# Print states and their clusters
for state, cluster in zip(df['state'], df['cluster']):
    print(f"{state}: Cluster {cluster}")
```

```
AL: Cluster 0
AZ: Cluster 2
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 0
DE: Cluster 0
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
```

```
IL: Cluster 0
IN: Cluster 0
IA: Cluster 2
KS: Cluster 2
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MS: Cluster 2
MO: Cluster 0
MT: Cluster 2
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 2
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 0
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AZ: Cluster 0
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 2
DE: Cluster 0
DC: Cluster 2
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 2
KS: Cluster 2
KY: Cluster 2
```

```
LA: Cluster 0
MA: Cluster 0
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
MT: Cluster 2
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 2
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 0
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AZ: Cluster 0
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 0
DE: Cluster 0
DC: Cluster 2
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 2
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
```

```
MT: Cluster 2
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 0
OR: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AZ: Cluster 2
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 2
DE: Cluster 0
DC: Cluster 2
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 2
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
MT: Cluster 2
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
```

```
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 0
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AZ: Cluster 2
AR: Cluster 0
CA: Cluster 0
CO: Cluster 2
CT: Cluster 2
DE: Cluster 0
DC: Cluster 2
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 2
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 0
MN: Cluster 2
MS: Cluster 0
MO: Cluster 0
MT: Cluster 2
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 0
```

```
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AZ: Cluster 2
AR: Cluster 0
CA: Cluster 0
CO: Cluster 2
CT: Cluster 2
DE: Cluster 0
DC: Cluster 1
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 1
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 0
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
MT: Cluster 1
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 2
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
```

```
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AK: Cluster 0
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 2
DE: Cluster 0
DC: Cluster 1
FL: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 1
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 0
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
MT: Cluster 1
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 2
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
```

```
VT: Cluster 2
VA: Cluster 0
WA: Cluster 0
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AK: Cluster 0
AZ: Cluster 2
AR: Cluster 2
CA: Cluster 0
CO: Cluster 0
CT: Cluster 0
DE: Cluster 0
DC: Cluster 1
FL: Cluster 0
GA: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 1
KS: Cluster 2
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 0
MN: Cluster 1
MS: Cluster 2
MO: Cluster 0
MT: Cluster 1
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 2
OK: Cluster 2
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
```

```
VA: Cluster 0
WA: Cluster 2
WI: Cluster 0
WY: Cluster 2
AL: Cluster 0
AK: Cluster 0
AZ: Cluster 1
AR: Cluster 2
CA: Cluster 0
CO: Cluster 0
CT: Cluster 0
DE: Cluster 0
DC: Cluster 1
GA: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 1
KS: Cluster 0
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 2
MN: Cluster 2
MS: Cluster 2
MO: Cluster 0
MT: Cluster 1
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OK: Cluster 2
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 2
WI: Cluster 0
```

```
WY: Cluster 0
AL: Cluster 0
AK: Cluster 0
AZ: Cluster 1
AR: Cluster 0
CA: Cluster 0
CO: Cluster 0
CT: Cluster 2
DE: Cluster 0
DC: Cluster 1
FL: Cluster 0
GA: Cluster 0
HI: Cluster 0
ID: Cluster 0
IL: Cluster 0
IN: Cluster 0
IA: Cluster 1
KS: Cluster 0
KY: Cluster 2
LA: Cluster 0
MA: Cluster 0
MI: Cluster 2
MN: Cluster 1
MS: Cluster 2
MO: Cluster 0
MT: Cluster 1
NE: Cluster 0
NV: Cluster 0
NJ: Cluster 2
NM: Cluster 1
NY: Cluster 0
NC: Cluster 0
ND: Cluster 0
OH: Cluster 0
OK: Cluster 2
OR: Cluster 2
PA: Cluster 2
RI: Cluster 2
SC: Cluster 0
SD: Cluster 0
TN: Cluster 0
TX: Cluster 0
UT: Cluster 0
VT: Cluster 2
VA: Cluster 0
WA: Cluster 2
WI: Cluster 0
WY: Cluster 0
```

[49]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Select features for clustering (e.g., per capita metrics, utilization)
features = ['per_capita_total_facilities', 'per_capita_mental_health_only',
            'per_capita_inpatient_facilities', 'total_util', 'outpatient_util',␣
 ↪'inpatient_util']
X = df[features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Impute missing values using SimpleImputer before clustering
imputer = SimpleImputer(strategy='mean')
X_scaled_imputed = imputer.fit_transform(X_scaled)

# Assume optimal k is 3 (You might need to adjust this based on your analysis)
k = 3

# Apply KMeans clustering using the imputed data
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled_imputed)

# Count states in cluster 2
num_states_cluster_2 = df[df['cluster'] == 2]['state'].nunique()

# Print the result
print(f"Number of states in cluster 2: {num_states_cluster_2}")
```

```
Number of states in cluster 2: 22
```

[50]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Select features for clustering (e.g., per capita metrics, utilization)
features = ['per_capita_total_facilities', 'per_capita_mental_health_only',
```

```
            'per_capita_inpatient_facilities', 'total_util', 'outpatient_util',␣
↪'inpatient_util']
X = df[features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Impute missing values using SimpleImputer before clustering
imputer = SimpleImputer(strategy='mean')
X_scaled_imputed = imputer.fit_transform(X_scaled)

# Assume optimal k is 3 (You might need to adjust this based on your analysis)
k = 3

# Apply KMeans clustering using the imputed data
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled_imputed)

# Count states in cluster 1
num_states_cluster_1 = df[df['cluster'] == 1]['state'].nunique()

# Print the result
print(f"Number of states in cluster 1: {num_states_cluster_1}")
```

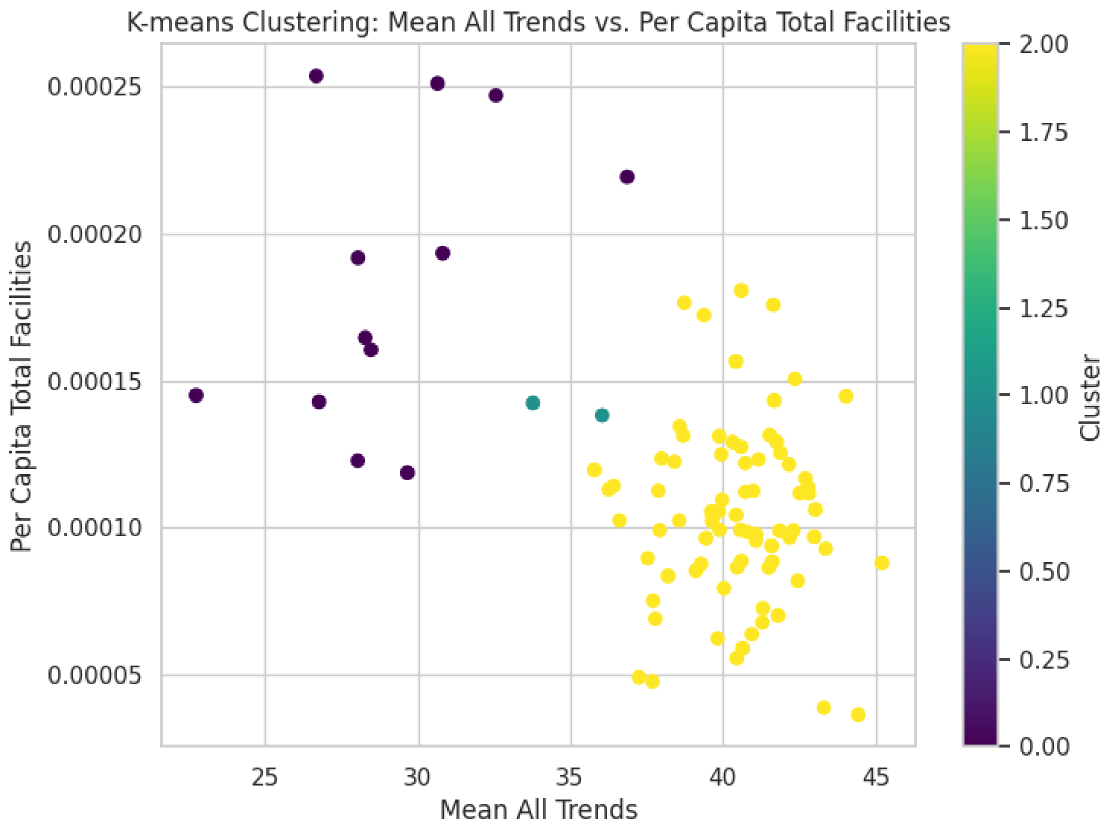
```
Number of states in cluster 1: 6
```

[51]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Select features for clustering (e.g., per capita metrics, utilization)
features = ['per_capita_total_facilities', 'per_capita_mental_health_only',
            'per_capita_inpatient_facilities', 'total_util', 'outpatient_util',␣
↪'inpatient_util']
X = df[features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Impute missing values using SimpleImputer before clustering
imputer = SimpleImputer(strategy='mean')
```

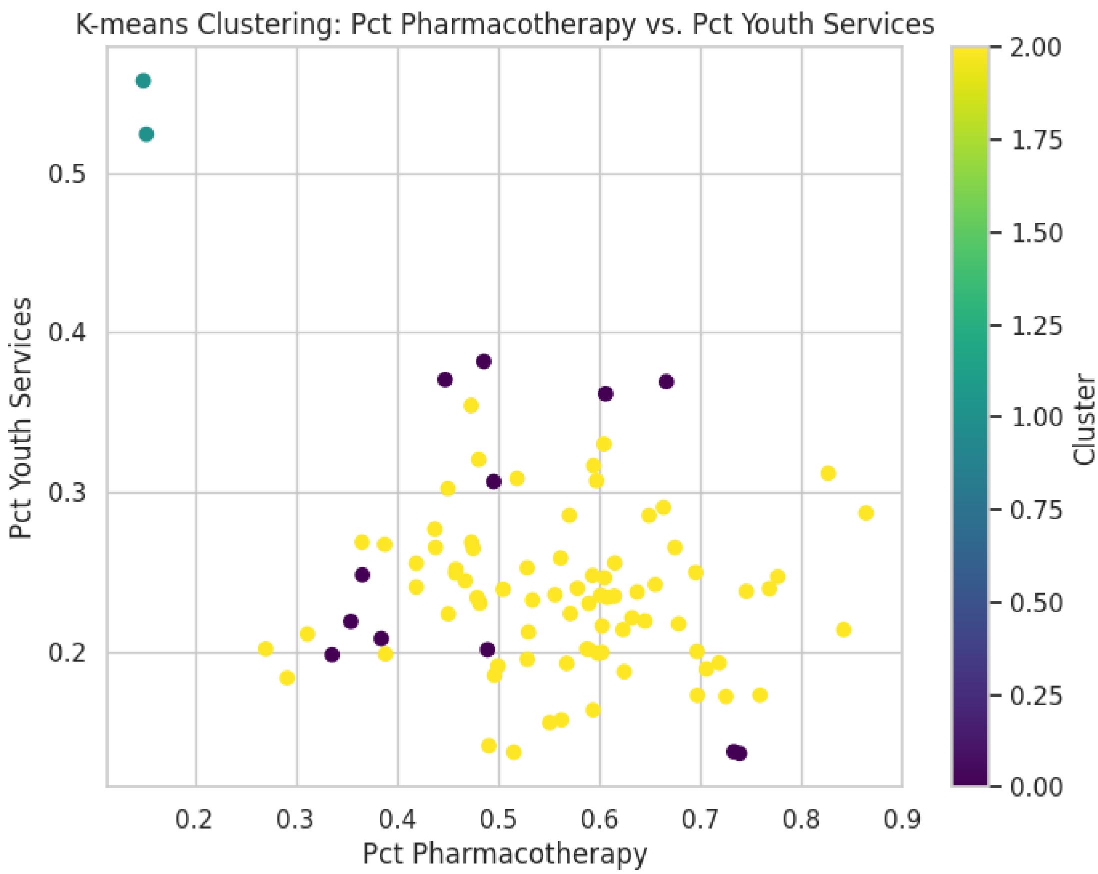
```
X_scaled_imputed = imputer.fit_transform(X_scaled)

# Assume optimal k is 3 (You might need to adjust this based on your analysis)
k = 3

# Apply KMeans clustering using the imputed data
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled_imputed)

# Filter for cluster 0 and get unique states
cluster_0_states = df[df['cluster'] == 0]['state'].unique()

# Print the number and list of states in cluster 0
print(f"Number of states in cluster 0: {len(cluster_0_states)}")
print(f"States in cluster 0: {cluster_0_states.tolist()}")
```

```
Number of states in cluster 0: 36
States in cluster 0: ['AL', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL', 'HI', 'ID',
'IL', 'IN', 'LA', 'MA', 'MO', 'NE', 'NV', 'NY', 'NC', 'ND', 'OK', 'SC', 'SD',
'TN', 'TX', 'UT', 'VA', 'WA', 'WI', 'AZ', 'MI', 'MS', 'AK', 'GA', 'KS', 'WY',
'OH']
```

[38]:
```
# Scatterplot for 'mean_all_trends' vs. 'per_capita_total_facilities'
plt.figure(figsize=(8, 6))
plt.scatter(merged_df['mean_all_trends'],␣
↪merged_df['per_capita_total_facilities'], c=merged_df['cluster'],␣
↪cmap='viridis')
plt.title('K-means Clustering: Mean All Trends vs. Per Capita Total Facilities')
plt.xlabel('Mean All Trends')
plt.ylabel('Per Capita Total Facilities')
plt.colorbar(label='Cluster')  # Add a colorbar to show cluster assignments
plt.show()

# Scatterplot for other feature combinations
# Example: 'pct_pharmacotherapy' vs. 'pct_youth_services'
plt.figure(figsize=(8, 6))
plt.scatter(merged_df['pct_pharmacotherapy'], merged_df['pct_youth_services'],␣
↪c=merged_df['cluster'], cmap='viridis')
plt.title('K-means Clustering: Pct Pharmacotherapy vs. Pct Youth Services')
plt.xlabel('Pct Pharmacotherapy')
plt.ylabel('Pct Youth Services')
plt.colorbar(label='Cluster')
plt.show()
```



K-means Clustering: Mean All Trends vs. Per Capita Total Facilities
Per Capita Total Facilities
0.00025
0.00020
0.00015
0.00010
0.00005
Mean All Trends
25
30
35
40
45
Cluster
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00



[39]:
```
# Show cluster assignments for the first few rows
print(merged_df[['state', 'year', 'cluster']].head(10))

# Get cluster sizes
cluster_sizes = merged_df['cluster'].value_counts()
print("\nCluster Sizes:\n", cluster_sizes)

# Analyze cluster characteristics
cluster_means = merged_df.groupby('cluster')[features].mean()
print("\nCluster Means:\n", cluster_means)
```

```
    state  year  cluster
350    AL  2021        2
351    AK  2021        0
352    AZ  2021        2
353    AR  2021        2
354    CA  2021        2
355    CO  2021        2
356    CT  2021        2
357    DE  2021        2
```

```
358    DC  2021        2
359    GA  2021        2

Cluster Sizes:
 cluster
2    78
0    12
1     2
Name: count, dtype: int64

Cluster Means:
          mean_all_trends  per_capita_total_facilities  pct_pharmacotherapy  \
cluster
0               29.125000                     0.000184             0.508804
1               34.902778                     0.000140             0.150714
2               40.441714                     0.000104             0.570073

         pct_youth_services
cluster
0                  0.261320
1                  0.541030
2                  0.233697
```

[54]:

```python
typology_labels = {
    0: "High-Need, Low-Access",
    1: "High-Search, Moderate-Utilization",
    2: "Low-Need, High-Access",

}

merged_df['typology'] = merged_df['cluster'].map(typology_labels)

# Analyze typology characteristics
typology_means = merged_df.groupby('typology')[features].mean()
print(typology_means)

# Example interpretation for "High-Need, Low-Access"
high_need_low_access_states = merged_df[merged_df['typology'] == "High-Need,␣
↪Low-Access"]['state'].unique()
print("\nHigh-Need, Low-Access States:", high_need_low_access_states)
```

```
                                   per_capita_total_facilities  \
typology
High-Need, Low-Access                                 0.000184
High-Search, Moderate-Utilization                     0.000140
Low-Need, High-Access                                 0.000104

                                   per_capita_mental_health_only  \
```

```
typology
High-Need, Low-Access                                     0.000033
High-Search, Moderate-Utilization                         0.000007
Low-Need, High-Access                                     0.000020

                                  per_capita_inpatient_facilities  \
typology
High-Need, Low-Access                                       0.000006
High-Search, Moderate-Utilization                           0.000001
Low-Need, High-Access                                       0.000003

                                  total_util  outpatient_util  inpatient_util
typology
High-Need, Low-Access               0.210781         0.030697        0.180084
High-Search, Moderate-Utilization   0.044993         0.006607        0.038386
Low-Need, High-Access               0.188134         0.027687        0.160448

High-Need, Low-Access States: ['AK' 'MT' 'ND' 'SD' 'VT' 'WY']
```

[53]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Select features for clustering (e.g., per capita metrics, utilization)
features = ['per_capita_total_facilities', 'per_capita_mental_health_only',
            'per_capita_inpatient_facilities', 'total_util', 'outpatient_util',␣
↪'inpatient_util']
X = df[features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Impute missing values using SimpleImputer before clustering
imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent'
X_scaled_imputed = imputer.fit_transform(X_scaled)  # Fit and transform

# 3 Clusters
k = 3

# Apply KMeans clustering using the imputed data
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled_imputed)  # Use imputed data
```

```
# Filter for 2024 data (if you have it) and select relevant columns
# Assuming your dataset has a 'year' column and relevant utilization columns
df_2024 = df[df['year'] == 2024][['state', 'cluster', 'total_util',␣
↪'outpatient_util', 'inpatient_util']]

# Export to Excel
df_2024.to_excel("clustered_2024_utilization.xlsx", index=False)
print("Clustered 2024 utilization data exported to 'clustered_2024_utilization.
↪xlsx'")
```

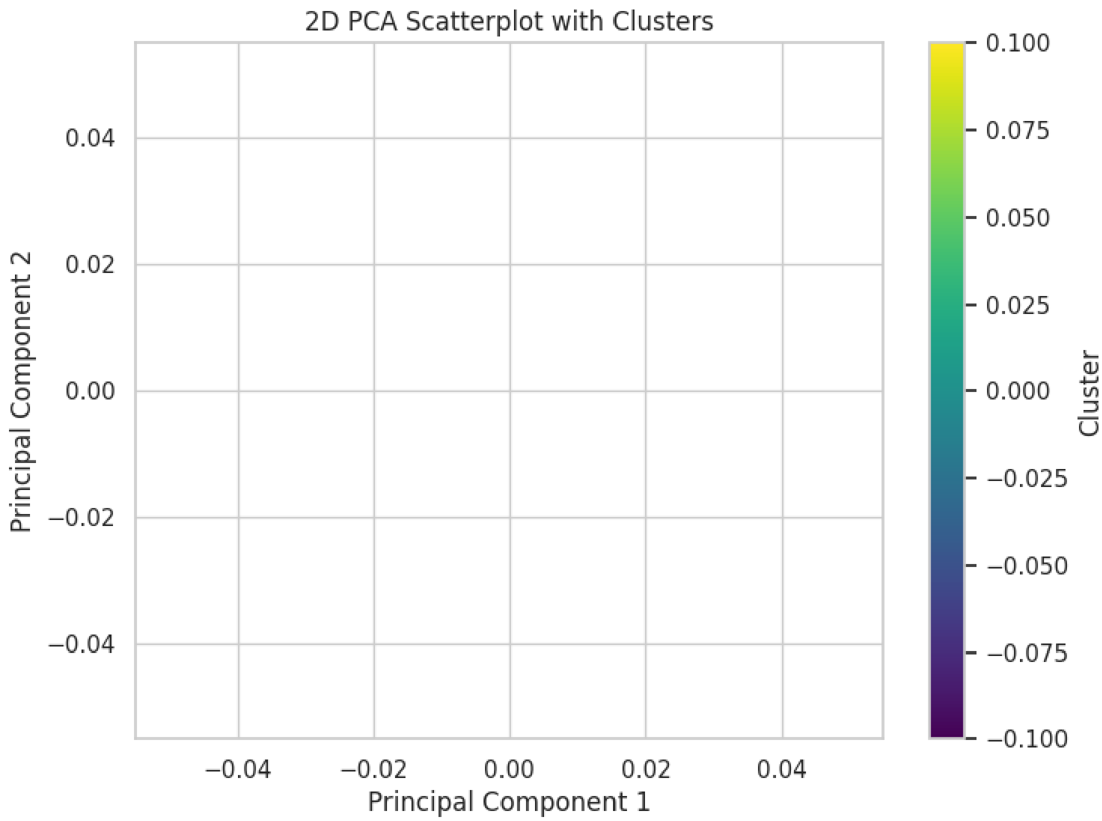
```
Clustered 2024 utilization data exported to 'clustered_2024_utilization.xlsx'
```

[41]:
```
state_clusters = merged_df[['state', 'cluster', 'typology']].drop_duplicates()
print(state_clusters)

from sklearn.decomposition import PCA

# Select features for PCA
features_for_pca = ['mean_all_trends', 'per_capita_total_facilities',␣
↪'pct_pharmacotherapy', 'pct_youth_services']
X_pca = merged_df[features_for_pca]

# Standardize the features
scaler = StandardScaler()
X_pca_scaled = scaler.fit_transform(X_pca)

# Apply PCA with 2 components
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_pca_scaled)

# Create a DataFrame with principal components and cluster labels
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['cluster'] = merged_df['cluster']

# Plot the PCA scatterplot
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['cluster'], cmap='viridis')
plt.title('2D PCA Scatterplot with Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()

# Typology summary
typology_summary = merged_df.groupby('typology')[features].agg(['mean', 'std'])
print(typology_summary)
```

```python
# Export to CSV for policy matrix
typology_summary.to_csv('typology_summary.csv')
print("Typology summary exported to 'typology_summary.csv'")
```

```
    state  cluster                             typology
350    AL        2                Low-Need, High-Access
351    AK        0                High-Need, Low-Access
352    AZ        2                Low-Need, High-Access
353    AR        2                Low-Need, High-Access
354    CA        2                Low-Need, High-Access
355    CO        2                Low-Need, High-Access
356    CT        2                Low-Need, High-Access
357    DE        2                Low-Need, High-Access
358    DC        2                Low-Need, High-Access
359    GA        2                Low-Need, High-Access
360    HI        1  High-Search, Moderate-Utilization
361    ID        2                Low-Need, High-Access
362    IL        2                Low-Need, High-Access
363    IN        2                Low-Need, High-Access
364    IA        2                Low-Need, High-Access
365    KS        2                Low-Need, High-Access
366    KY        2                Low-Need, High-Access
367    LA        2                Low-Need, High-Access
368    MA        2                Low-Need, High-Access
369    MI        2                Low-Need, High-Access
370    MN        2                Low-Need, High-Access
371    MS        2                Low-Need, High-Access
372    MO        2                Low-Need, High-Access
373    MT        0                High-Need, Low-Access
374    NE        2                Low-Need, High-Access
375    NV        2                Low-Need, High-Access
376    NJ        2                Low-Need, High-Access
377    NM        2                Low-Need, High-Access
378    NY        2                Low-Need, High-Access
379    NC        2                Low-Need, High-Access
380    ND        0                High-Need, Low-Access
381    OK        2                Low-Need, High-Access
382    OR        2                Low-Need, High-Access
383    PA        2                Low-Need, High-Access
384    RI        2                Low-Need, High-Access
385    SC        2                Low-Need, High-Access
386    SD        0                High-Need, Low-Access
387    TN        2                Low-Need, High-Access
388    TX        2                Low-Need, High-Access
389    UT        2                Low-Need, High-Access
390    VT        0                High-Need, Low-Access
391    VA        2                Low-Need, High-Access
```

392	WA	2	Low-Need, High-Access
393	WI	2	Low-Need, High-Access
394	WY	0	High-Need, Low-Access
404	FL	2	Low-Need, High-Access
427	OH	2	Low-Need, High-Access



	mean_all_trends		\
	mean	std	
typology			
High-Need, Low-Access	29.125000	3.478974	
High-Search, Moderate-Utilization	34.902778	1.604085	
Low-Need, High-Access	40.441714	1.973664	

	per_capita_total_facilities		\
	mean	std	
typology			
High-Need, Low-Access	0.000184	0.000049	
High-Search, Moderate-Utilization	0.000140	0.000003	
Low-Need, High-Access	0.000104	0.000030	

pct_pharmacotherapy \

```
                                              mean       std
typology
High-Need, Low-Access                     0.508804  0.145325
High-Search, Moderate-Utilization         0.150714  0.002065
Low-Need, High-Access                     0.570073  0.121732

                                  pct_youth_services
                                                mean       std
typology
High-Need, Low-Access                       0.261320  0.092300
High-Search, Moderate-Utilization           0.541030  0.023700
Low-Need, High-Access                       0.233697  0.045027
Typology summary exported to 'typology_summary.csv'
```

[56]:

```python
import pandas as pd

# Load your dataset
df = pd.read_csv("Merged_Trends_NSUMHSS_2013_2023.csv")

# Function to summarize each variable
def create_codebook(df):
    # Create descriptions that match the number of columns
    descriptions = [
        "Year of the data",  # year
        "US State",  # state
        "US Region",  # region
        "Estimated state population",  # population_est
        "Average Google Trends score for mental health-related topics",  #␣
↪mean_all_trends
        "Total mental health facilities per capita",  #␣
↪per_capita_total_facilities
        "Mental health-only facilities per capita",  #␣
↪per_capita_mental_health_only
        "Inpatient mental health facilities per capita",  #␣
↪per_capita_inpatient_facilities
        "Percentage of facilities offering pharmacotherapy",  #␣
↪pct_pharmacotherapy
        "Percentage of facilities offering youth services",  #␣
↪pct_youth_services
        "Percentage of facilities offering free services",  # pct_free_services
        "Percentage of facilities offering Medicare services",  #␣
↪pct_medicare_services
        "Percentage of facilities offering counseling services",  #␣
↪pct_counseling_services
        "Total utilization of mental health services",  # total_util
        "Inpatient utilization of mental health services",  # inpatient_util
        "Outpatient utilization of mental health services",  # outpatient_util
```

```
        # Add descriptions for any additional columns if present in your dataset
        *["" for _ in range(len(df.columns) - 16)] # Handle additional columns␣
↪by adding empty descriptions
    ]

    codebook = pd.DataFrame({
        "Variable": df.columns,
        "Data Type": df.dtypes.values,
        "Missing Values": df.isnull().sum().values,
        "Unique Values": df.nunique().values,
        "Min": [df[col].min() if pd.api.types.is_numeric_dtype(df[col]) else␣
↪None for col in df.columns],
        "Max": [df[col].max() if pd.api.types.is_numeric_dtype(df[col]) else␣
↪None for col in df.columns],
        "Example Value": [df[col].dropna().iloc[0] if not df[col].dropna().
↪empty else None for col in df.columns],
        "Description": descriptions  # Use the dynamically generated␣
↪descriptions
    })
    return codebook

# Generate codebook
codebook_df = create_codebook(df)

# Export codebook to CSV
codebook_df.to_csv("Codebook_Mental_Health_Project_updated.csv", index=False)

print("Codebook generated and saved as 'Codebook_Mental_Health_Project_updated.
↪csv'")
```

```
Codebook generated and saved as 'Codebook_Mental_Health_Project_updated.csv'
```