IE7275 12159 Data Mining in Engineering SEC 02 Fall 2024

Assignment -2

Group 11

Aayush Amrute - amrute.a@northeastern.edu
(NUID : 002838262)

Paritosh Vyawahare - vyawahare.p@northeastern.edu
(NUID : 002416079)

Saurabh Chavan - chavan.sau@northeastern.edu
(NUID : 002479083)

Simran Sinha - sinha.sim@northeastern.edu
(NUID : 002475433)

Submission Date: 23-Oct-2024

Problem 1

Dataset- Activity
Dataset Description-Activity dataset shows how has global plastic waste disposal method changed over time.

TODO 1: Create an animated bar chart to illustrate how waste disposal methods have evolved over the years.

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

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

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

```
# Here we will import the data first and and then drop the null values
waste_data=pd.read_excel('/content/drive/MyDrive/Colab
Notebooks/Datasets/activity.xlsx') cleaned_data = waste_data.dropna() cleaned_data =
cleaned_data[cleaned_data['Type'] != 'Generation']
```

```
cleaned_data
```

	year	Value	Type
2	1960	5.6	Recycling
3	1960	0.0	Combustion with energy recovery
4	1960	82.5	Landfilling and other disposal
7	1970	8.0	Recycling
8	1970	0.5	Combustion with energy recovery
9	1970	112.6	Landfilling and other disposal
12	1980	14.5	Recycling

```
13   1980      2.8      Combustion with energy
     recovery

14   1980      134.3    Landfilling and other disposal

18   1990      29.8     Combustion with energy
     recovery

19   1990      145.3    Landfilling and other disposal

 17 1990      29.0                          Recycling

 16 1990       4.2                          Composting*

21   2000      16.5     Composting*

22   2000      53.0     Recycling

23   2000      33.7     Combustion with energy
     recovery

24   2000      140.3    Landfilling and other disposal

 29 2005     142.2          Landfilling and other disposal

 28 2005      31.7    Combustion with energy recovery

 27 2005      59.2                          Recycling

 26 2005      20.6                          Composting*

31   2010      20.2     Composting*

32   2010      65.3     Recycling

33   2010      29.3     Combustion with energy
     recovery

34   2010      136.3    Landfilling and other disposal

38   2015      33.5     Combustion with energy
     recovery

39   2015      137.6    Landfilling and other disposal

36   2015      23.4     Composting*

37   2015      67.6     Recycling

41 2016        25.1     Composting* 42 2016   68.6
       Recycling

43   2016      33.9     Combustion with energy
     recovery

44   2016      139.2    Landfilling and other disposal

46   2017      27.0     Composting*
```

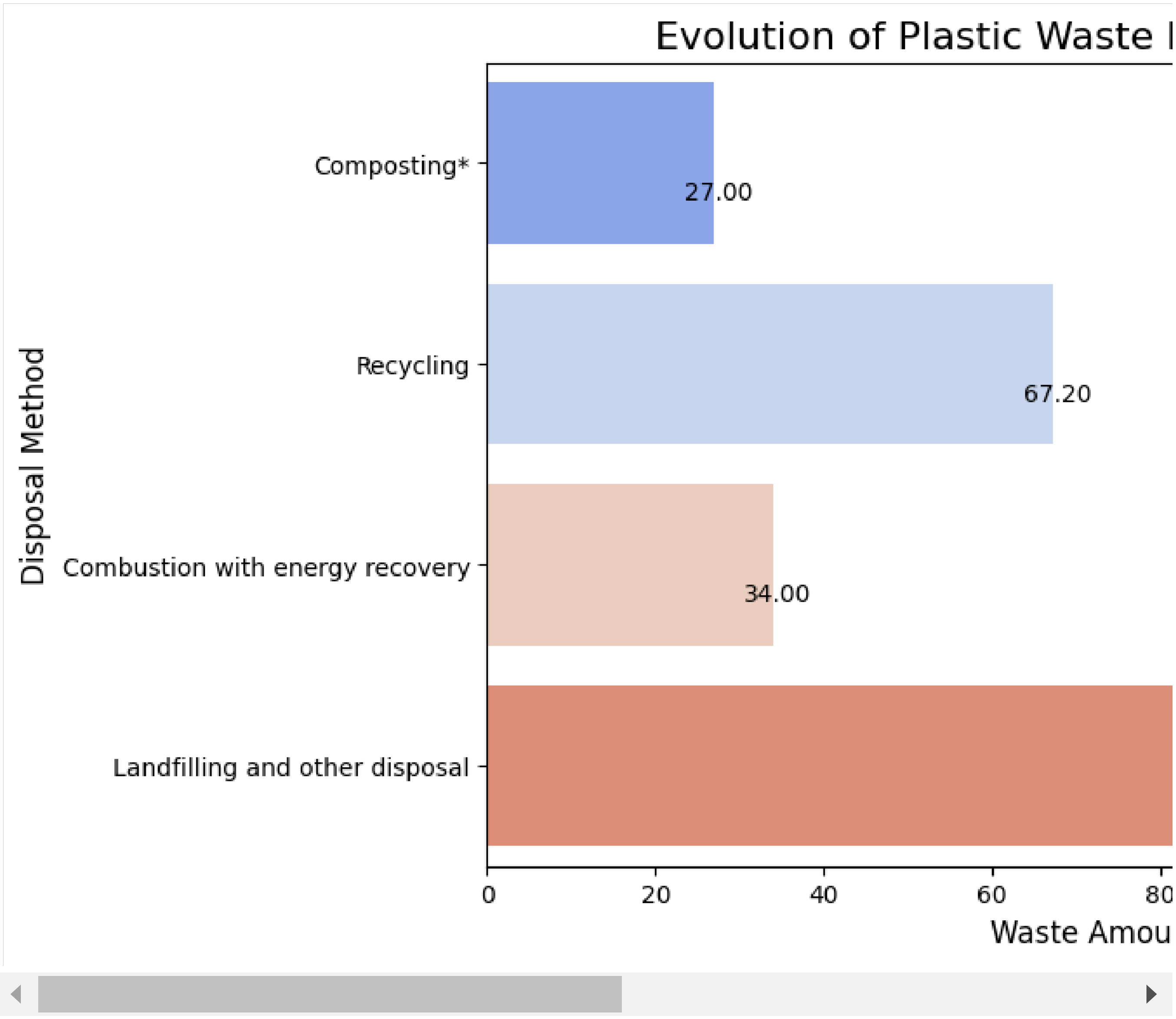
47	2017	67.2	Recycling
48	2017	34.0	Combustion with energy recovery
49	2017	139.6	Landfilling and other disposal

Next steps: Generate code with `cleaned_data` | View recommended plots | New interactive sheet

```
cleaned_data.sort_values(by='year', inplace=True)
figure, axis = plt.subplots(figsize=(10, 6))


def animate(current_year):
    axis.clear()     data_for_current_year = cleaned_data[cleaned_data['year'] ==
current_year]     sns.barplot(x='Value', y='Type', data=data_for_current_year,
palette='coolwarm', ax=axis     axis.set_title(f'Evolution of Plastic Waste Disposal
Methods in {current_year}', fontsiz     axis.set_xlabel('Waste Amount (in tons)',
fontsize=12)     axis.set_ylabel('Disposal Method', fontsize=12)     axis.set_xlim(0,
cleaned_data['Value'].max() * 1.1)     for bar in axis.patches:
axis.text(bar.get_width() + 0.5, bar.get_y() + 0.55, f'{bar.get_width():.2f}', ha='c


all_years = sorted(cleaned_data['year'].unique()) animation_plot =
animation.FuncAnimation(figure, animate, frames=all_years, repeat=False)
animation_plot.save('plastic_waste_animation.gif', writer='pillow', fps=1) plt.show()
```

```
<ipython-input-62-bec2a27c8b39>:8: FutureWarning:



  Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.


  <ipython-input-62-bec2a27c8b39>:8: FutureWarning:



  Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.


  <ipython-input-62-bec2a27c8b39>:8: FutureWarning:



  Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.


  <ipython-input-62-bec2a27c8b39>:8: FutureWarning:



  Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```

```
<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

<ipython-input-62-bec2a27c8b39>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```



TODO 2: What key insights can you draw from the visualization

1) **1960 Trends:** In 1960, landfilling was the predominant method for disposing of plastic waste, with approximately 60 tons generated, indicating that it was the most widely used disposal method at the time. Only three methods were utilized in that year: landfilling, recycling, and energy recovery.

2) **2000 Overview:** By the year 2000, the composting method was introduced as a waste disposal option. However, landfilling continued to be the most commonly used method for disposing of plastic waste.

3) **2000-2010 Developments:** Between 2000 and 2010, there was a notable increase in the use of both combustion and recycling methods for plastic waste disposal, indicating a shift in waste management practices.

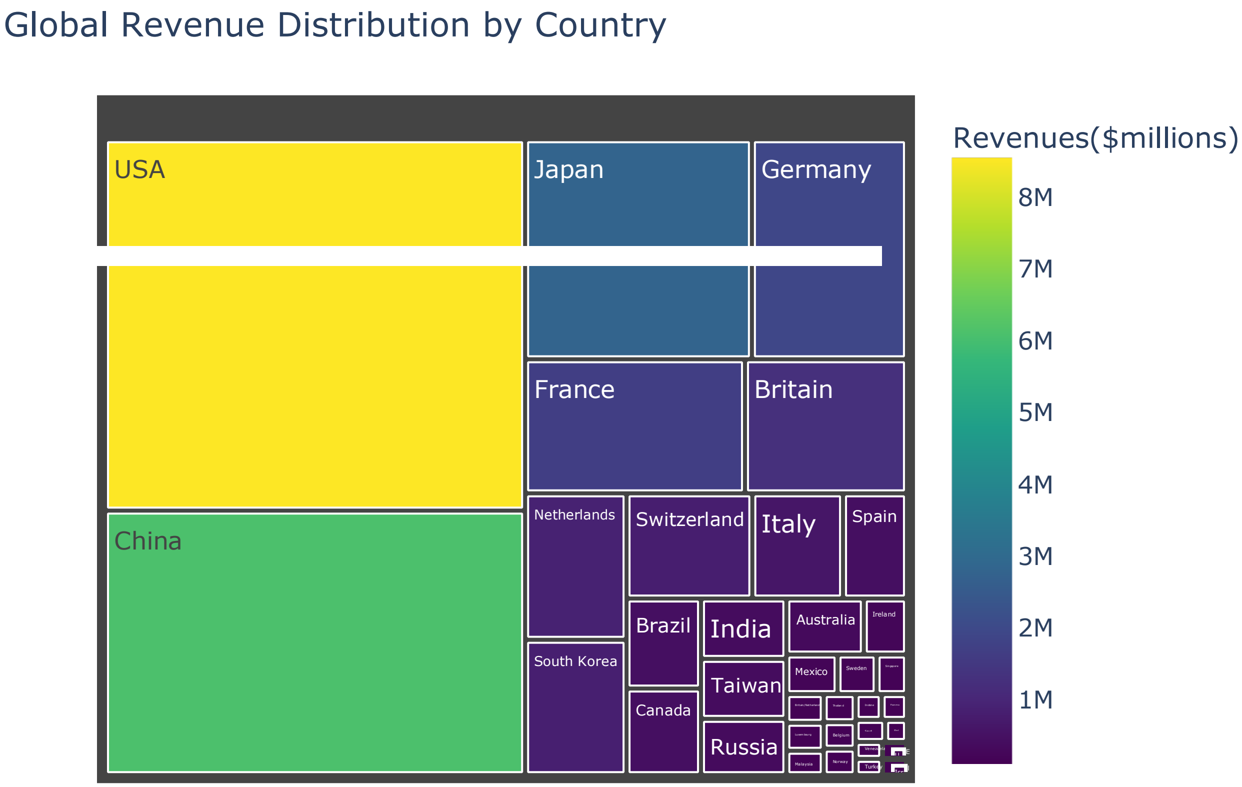
4)**Long-Term Trends:** Over the years, the reliance on landfilling has declined, while the use of combustion and recycling methods has increased. This trend is a very positive development for waste management, reflecting an improvement in sustainable practices and a move towards more environmentally friendly disposal methods.

Problem 2

Dataset - Global 500
Dataset Description - Fortune Global 500 List is a list of largest corporations worldwide which are measured by their total fiscal year revenues. Companies rankings sorted by total revenues for their respective fiscal years ended on or before March 31 of the relevant year.

TODO 1: Using treemap infer which countries dominate the global revenue landscape Interpret your key findings from the map
Hint: Perform data preprocessing before plotting the map

```
import plotly.express as px global_data = pd.read_excel('/content/drive/MyDrive/Colab
Notebooks/Datasets/Global 500.xlsx

# We need to remove the '$' and unneccessary commasfrom revenue column and convert it to num
global_data['Revenues($millions)'] = global_data['Revenues($millions)'].replace({'\$': '', '

# Aggregate total revenue by country revenue_by_country =

global_data.groupby('Country')['Revenues($millions)'].sum().reset_index # Sorting values

revenue_by_country = revenue_by_country.sort_values('Revenues($millions)', ascending=False)

# treemap fig = px.treemap(revenue_by_country,
path=['Country'],
values='Revenues($millions)',                  title='Global
Revenue Distribution by Country',
color='Revenues($millions)',
color_continuous_scale='Viridis') fig.show()
```



Top Countries by Revenue: The plot clearly shows that the USA and China are the largest countries in terms of revenue, with the USA generating approximately $8.48 trillion and China about 6.04 trillion. The USA is home to major companies like Amazon and Walmart, contributing significantly to its overall revenue.

Japan and Germany's Position: Following the USA and China, Japan and Germany rank third and fourth globally in revenue. Notable companies in these countries include Toyota and Mitsubishi, which play a crucial role in their economic output.

Similar Revenue Levels: Countries such as India, Brazil, Canada, and Taiwan have revenues that are nearly equal, each hovering around $2 trillion. This indicates a competitive economic landscape among these natio

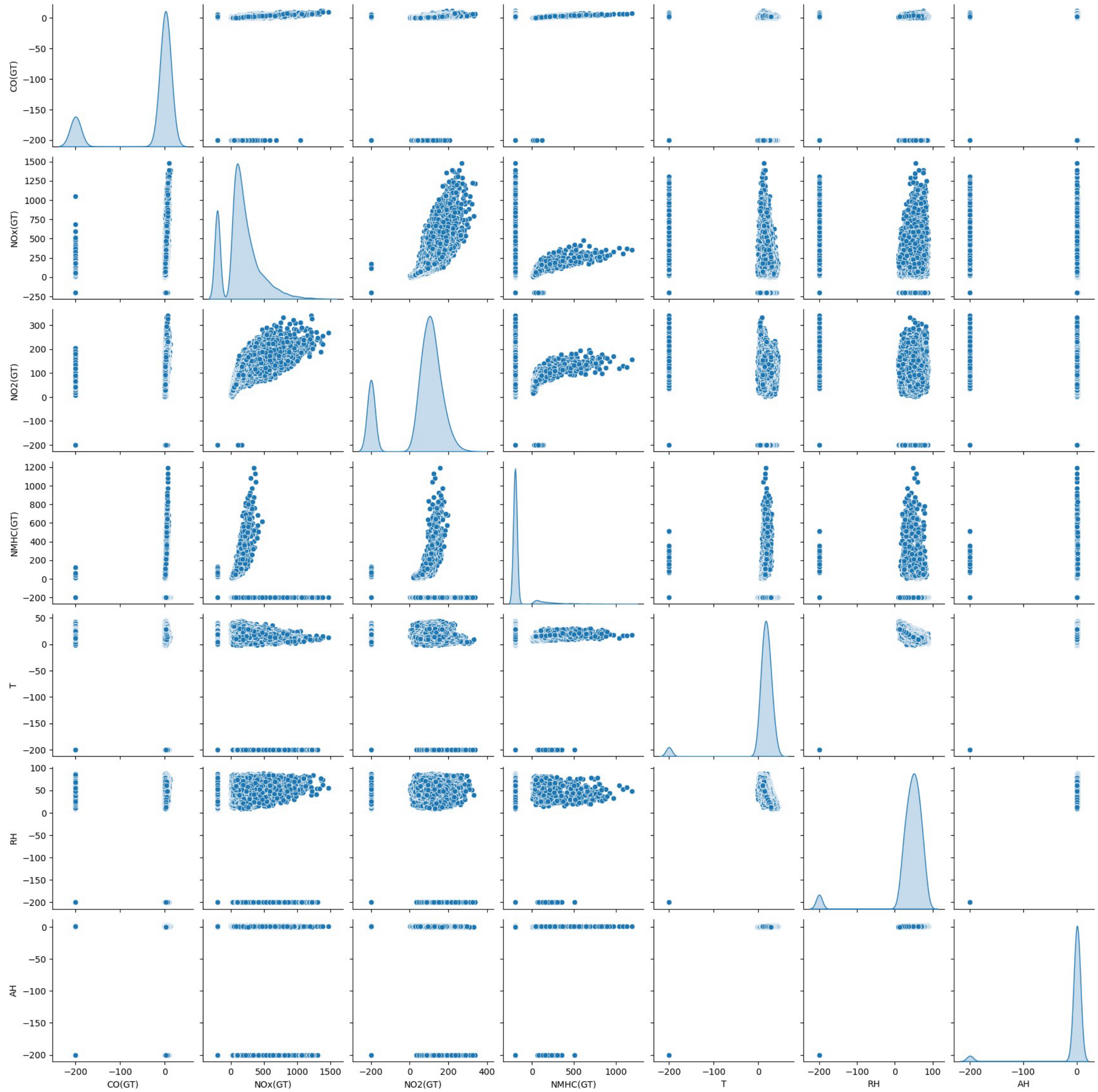
□ Problem 3

Dataset - AirQualityUCI
Dataset Description - The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device.

The device was located on the field in a significantly polluted area, at road level,within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.

TODO 1: Analyze the relationships between pollutants and environmental factors (T, RH, AH) using a scatter matrix (pair plot). Interpret your findings from the data

```
#Importing data air_quality=pd.read_excel('/content/drive/MyDrive/Colab
Notebooks/Datasets/AirQualityUCI.xls


#Taking only environmental factors which are needed
environmental_factor=['CO(GT)', 'NOx(GT)', 'NO2(GT)', 'NMHC(GT)', 'T', 'RH','AH']


sns.pairplot(air_quality[environmental_factor],diag_kind='kde')
plt.show()
```



Distribution of Pollutants: Pollutants such as CO(GT), NOx(GT), and NO2(GT) display right-skewed distributions, suggesting that lower concentrations are prevalent, while there are occasional instances of significantly higher levels.

Temperature Effects: There is a noticeable positive correlation between temperature (T) and pollutants like CO(GT), NMHC(GT), and NOx(GT). As temperatures rise, so do the levels of these pollutants, which could be attributed to heightened emissions from various human activities during warmer weather.

Humidity Relationships: Relative Humidity (RH) appears to have a slight negative correlation with certain pollutants, including CO(GT) and NOx(GT). This implies that higher humidity levels may be associated with lower concentrations of these pollutants, possibly because moisture in the air helps disperse them.

Humidity and Temperature Connection: Absolute Humidity (AH) exhibits a strong positive correlation with Temperature (T). This relationship is intuitive, as warmer air can hold more moisture, thereby directly influencing absolute humidity levels.

Problem 4

Dataset: [Wine Classification Dataset]
Dataset Description : These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

TODO 1: Calculate the cumulative variance explained by each of the first two principal components using the raw data (without standardization). Explain how much of the total variance is captured by these two components.

```
# ucimlrepo package
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```
#Import the dataset into your code
from ucimlrepo import fetch_ucirepo
  # fetch dataset wine =
fetch_ucirepo(id=109)

# data (as pandas
dataframes) X =
wine.data.features y =
wine.data.targets
```

```
wine.metadata
```

```
{'uci_id': 109,
 'name': 'Wine',
 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine',
 'data_url': 'https://archive.ics.uci.edu/static/public/109/data.csv',
 'abstract': 'Using chemical analysis to determine the origin of wines',
 'area': 'Physics and Chemistry',
 'tasks': ['Classification'],
 'characteristics': ['Tabular'],
 'num_instances': 178,
 'num_features': 13,
 'feature_types': ['Integer', 'Real'],
 'demographics': [],
 'target_col': ['class'],
 'index_col': None,
 'has_missing_values': 'no',
 'missing_values_symbol': None,
 'year_of_dataset_creation': 1992,
 'last_updated': 'Mon Aug 28 2023',
 'dataset_doi': '10.24432/C5PC7J',
 'creators': ['Stefan Aeberhard', 'M. Forina'],
 'intro_paper': {'ID': 246,


  'type': 'NATIVE',
  'title': 'Comparative analysis of statistical pattern recognition methods in high
dimensional settings',
  'authors': 'S. Aeberhard, D. Coomans, O. Vel',
  'venue': 'Pattern Recognition',
  'year': 1994,
  'journal': None,
  'DOI': '10.1016/0031-3203(94)90145-7',
  'URL':
'https://www.semanticscholar.org/paper/83dc3e4030d7b9fbdbb4bde03ce12ab70ca10528',
  'sha': None,
  'corpus': None,
  'arxiv': None,
  'mag': None,
  'acl': None,
  'pmid': None,
  'pmcid': None},
 'additional_info': {'summary': 'These data are the results of a chemical analysis
of wines grown in the same region in Italy but derived from three different
cultivars. The analysis determined the quantities of 13 constituents found in each
of the three types of wines. \r\n\r\nI think that the initial data set had around 30
variables, but for some reason I only have the 13 dimensional version. I had a list
of what the 30 or so variables were, but a.)  I lost it, and b.), I would not know
which 13 variables are included in the set.\r\n\r\nThe attributes are (dontated by
Riccardo Leardi, riclea@anchem.unige.it )\r\n1) Alcohol\r\n2) Malic acid\r\n3)
Ash\r\n4) Alcalinity of ash  \r\n5) Magnesium\r\n6) Total phenols\r\n7)
Flavanoids\r\n8) Nonflavanoid phenols\r\n9) Proanthocyanins\r\n10)Color
intensity\r\n11)Hue\r\n12)OD280/OD315 of diluted wines\r\n13)Proline \r\n\r\nIn a
classification context, this is a well posed problem with "well behaved" class
structures. A good data set for first testing of a new classifier, but not very
challenging.           ',
  'purpose': 'test',
  'funded_by': None,
  'instances_represent': None,
  'recommended_data_splits': None,
  'sensitive_data': None,
wine.variables
```

```
                          name     role         type  demographic  description  units  mis
0                        class   Target  Categorical         None         None   None
1                      Alcohol  Feature   Continuous         None         None   None
2                    Malicacid  Feature   Continuous         None         None   None
3                          Ash  Feature   Continuous         None         None   None
4            Alcalinity_of_ash  Feature   Continuous         None         None   None
5                    Magnesium  Feature      Integer         None         None   None
6                Total_phenols  Feature   Continuous         None         None   None
7                   Flavanoids  Feature   Continuous         None         None   None
8         Nonflavanoid_phenols  Feature   Continuous         None         None   None
9              Proanthocyanins  Feature   Continuous         None         None   None
10             Color_intensity  Feature   Continuous         None         None   None
11                         Hue  Feature   Continuous         None         None   None
12 0D280_0D315_of_diluted_wines  Feature   Continuous         None         None   None
13                     Proline  Feature      Integer         None         None   None
```

```
#PCA without standardizing
pca_raw = PCA(n_components=2)
pca_raw.fit(X)
```

```
explained_variance_raw = pca_raw.explained_variance_ratio_
cumulative_variance_raw = np.cumsum(explained_variance_raw)
```

```
print("Explained variance (raw data):", explained_variance_raw)
print("Cumulative variance (raw data):", cumulative_variance_raw)
```

```
Explained variance (raw data): [0.99809123 0.00173592]
 Cumulative variance (raw data): [0.99809123 0.99982715]
```

TODO 2: Use the function PCA() on the centered but not scaled data to calculate the principal components. Compare how the components differ when using centered-only data versus raw data. Discuss any shifts in the proportion of variance explained.

```
#Finding centered values

centered_data = X -

np.mean(X,axis=0) pca_centered =

PCA(n_components=2)

pca_centered.fit(centered_data)


explained_variance_centered = pca_centered.explained_variance_ratio_
cumulative_variance_centered = np.cumsum(explained_variance_centered)

print("Explained variance (centered data):", explained_variance_centered)
print("Cumulative variance (centered data):", cumulative_variance_centered)
```

```
Explained variance (centered data): [0.99809123 0.00173592]
 Cumulative variance (centered data): [0.99809123 0.99982715]
```

TODO 3: Use PCA on the standardized data but compute the top three principal components instead of the first two. Compare how much more variance is captured by including the third principal component.

```
from sklearn.preprocessing import StandardScaler


#Standardizing Data scaler =
StandardScaler() standardized_data =
scaler.fit_transform(X)

pca_standardized = PCA(n_components=3)
pca_standardized.fit(standardized_data)


explained_variance_standardized = pca_standardized.explained_variance_ratio_
cumulative_variance_standardized = np.cumsum(explained_variance_standardized)
```

```
print("Explained variance (standardized data):", explained_variance_standardized)
print("Cumulative variance (standardized data):",
cumulative_variance_standardized)
```

```
Explained variance (standardized data): [0.36198848 0.1920749  0.11123631]
Cumulative variance (standardized data): [0.36198848 0.55406338 0.66529969]
```

TODO 4: Compare the results of PCA on standardized vs. min-max normalized data. Discuss the impact of these two techniques on the PCA outcomes.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler



# z-score normalization scaler_standard =
StandardScaler() standardized_data =
scaler_standard.fit_transform(X)


pca_standardized = PCA(n_components=3)
pca_standardized.fit(standardized_data)

explained_variance_standardized = pca_standardized.explained_variance_ratio_
cumulative_variance_standardized = np.cumsum(explained_variance_standardized)

print("Explained variance (standardized data):", explained_variance_standardized)
print("Cumulative variance (standardized data):",
cumulative_variance_standardized)

# Normalizing data scaler_minmax =
MinMaxScaler() normalized_data =
scaler_minmax.fit_transform(X)


pca_normalized = PCA(n_components=3)
pca_normalized.fit(normalized_data)

explained_variance_normalized = pca_normalized.explained_variance_ratio_
cumulative_variance_normalized = np.cumsum(explained_variance_normalized)

print("Explained variance (min-max normalized data):", explained_variance_normalized)
print("Cumulative variance (min-max normalized data):", cumulative_variance_normalized)
```

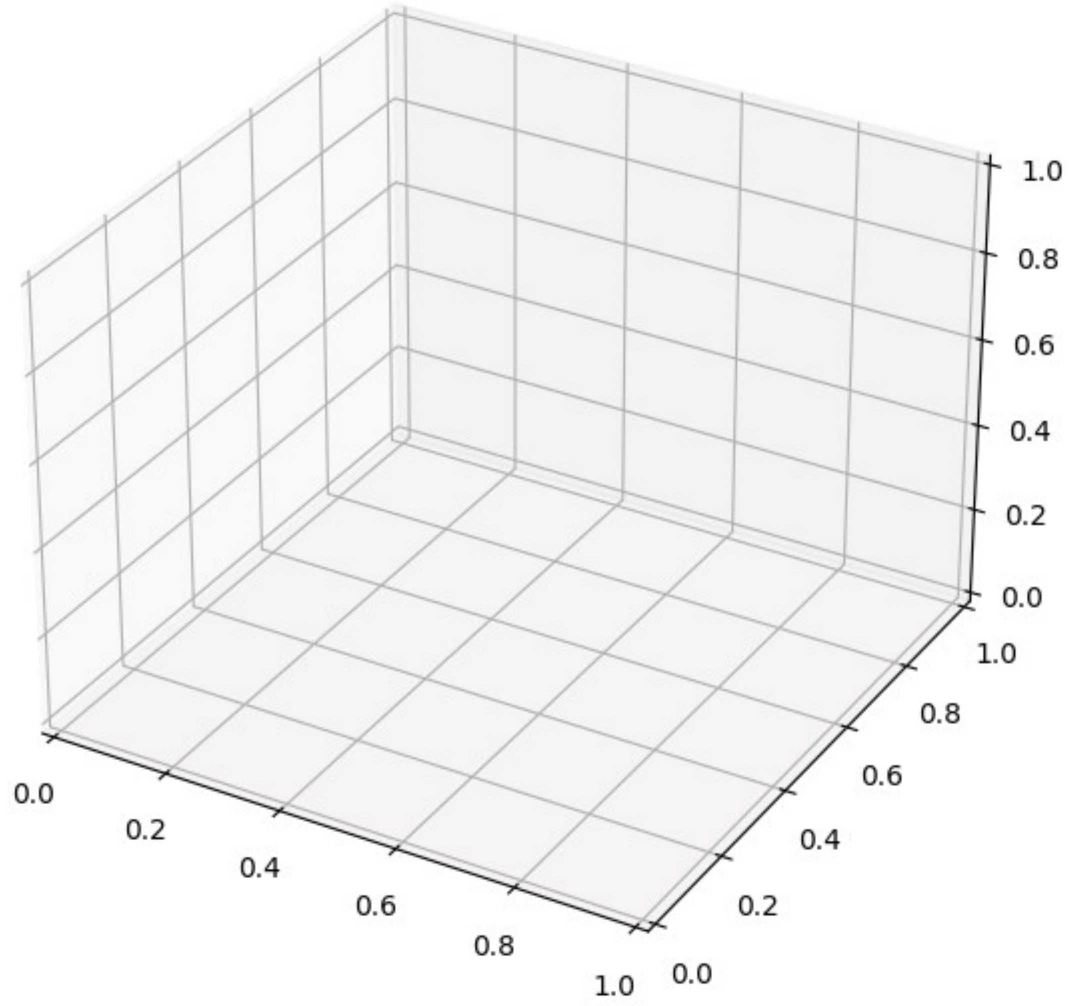
```
Explained variance (standardized data): [0.36198848 0.1920749  0.11123631]
Cumulative variance (standardized data): [0.36198848 0.55406338 0.66529969]
Explained variance (min-max normalized data): [0.40749485 0.18970352 0.08561671]
Cumulative variance (min-max normalized data): [0.40749485 0.59719836 0.68281507]
```

TODO 5: Instead of plotting only the first two PCs, generate a 3D plot using the first three principal components. Use color to differentiate wine classes and interpret the additional insights from the third component. python

```
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
```

```
X_pca_std = pca_standardized.fit_transform(standardized_data)
pc1 = X_pca_std[:, 0] pc2 = X_pca_std[:, 1] pc3 =
X_pca_std[:, 2]
```

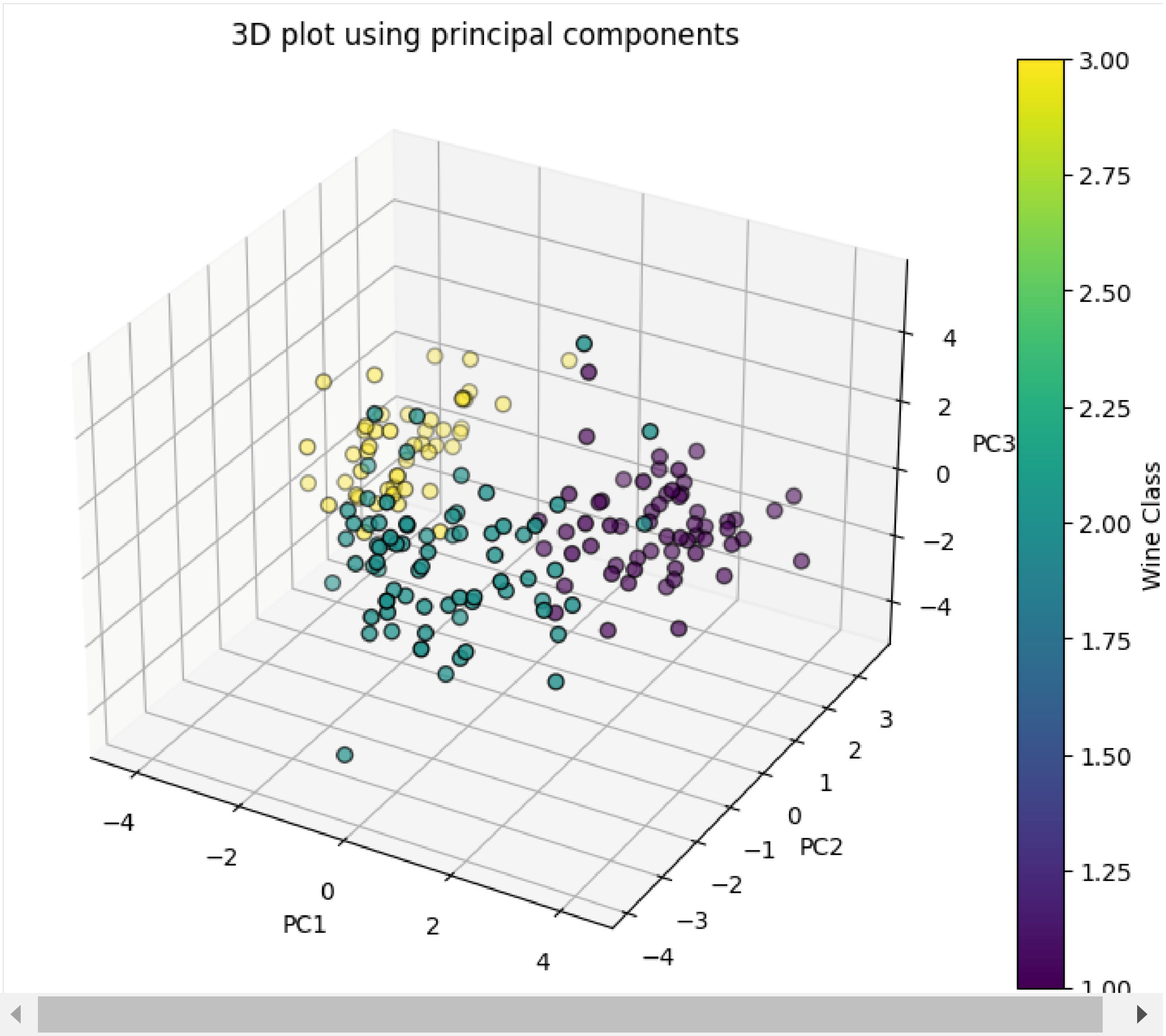
```
fig = plt.figure(figsize=(10, 7)) ax =
fig.add_subplot(111, projection='3d')
```



```
scatter = ax.scatter(pc1, pc2, pc3, c=y, cmap='viridis', edgecolor='k', s=40)
```

```
#plotting the data
```

```
ax.set_title("3D plot using principal components")
ax.set_xlabel("PC1") ax.set_ylabel("PC2")
ax.set_zlabel("PC3") color = fig.colorbar(scatter,
ax=ax, label='Wine Class') plt.show() fig
```



Principal Component 1 (PC1): Represents the highest amount of variance in the dataset, highlighting the most pronounced differences among the wine samples.

Principal Component 2 (PC2): Accounts for the second largest variance, capturing variations that are independent of those represented by PC1.

Principal Component 3 (PC3): Captures additional variance that may not be explained by PC1 and PC2, allowing for more detailed differentiation of the samples.

Separation of Wine Classes: The analysis reveals a distinct separation among the three wine categories, illustrating the effectiveness of PCA in distinguishing between them.

Class Distinction: PC1 is particularly effective in differentiating between the various wine classes.

Further Differentiation: Both PC2 and PC3 contribute to refining the separation of overlapping classes, with PC3 revealing subtle distinctions in a three-dimensional representation.

Problem 5

Dataset: Life Expectancy

Introduction: The above dataset gives life expectancy related data for 37 countries in 2014.

Consider only the following variables in your analysis: 'GDP', 'Income composition of resources', 'Schooling', and 'Total expenditure'.

TODO 1: Perform Z-score normalization on the numeric variables to scale the data. Compare the distribution of features before and after Z-score normalization and discuss the effect on variance.

```
#Importing Data df=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Datasets/Life
Expectancy.csv')
```

```
df.head()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hep
0	Afghanistan	2014	Developing	59.9	271	64	0.01	73.523582	
1	Australia	2014	Developed	82.7	6	1	9.71	10769.363050	
2	Austria	2014	Developed	81.4	66	0	12.32	8350.193523	
3	Bangladesh	2014	Developing	71.4	132	98	0.01	10.446403	
4	Belgium	2014	Developed	89.0	76	0	12.60	7163.348923	

5 rows × 22 columns

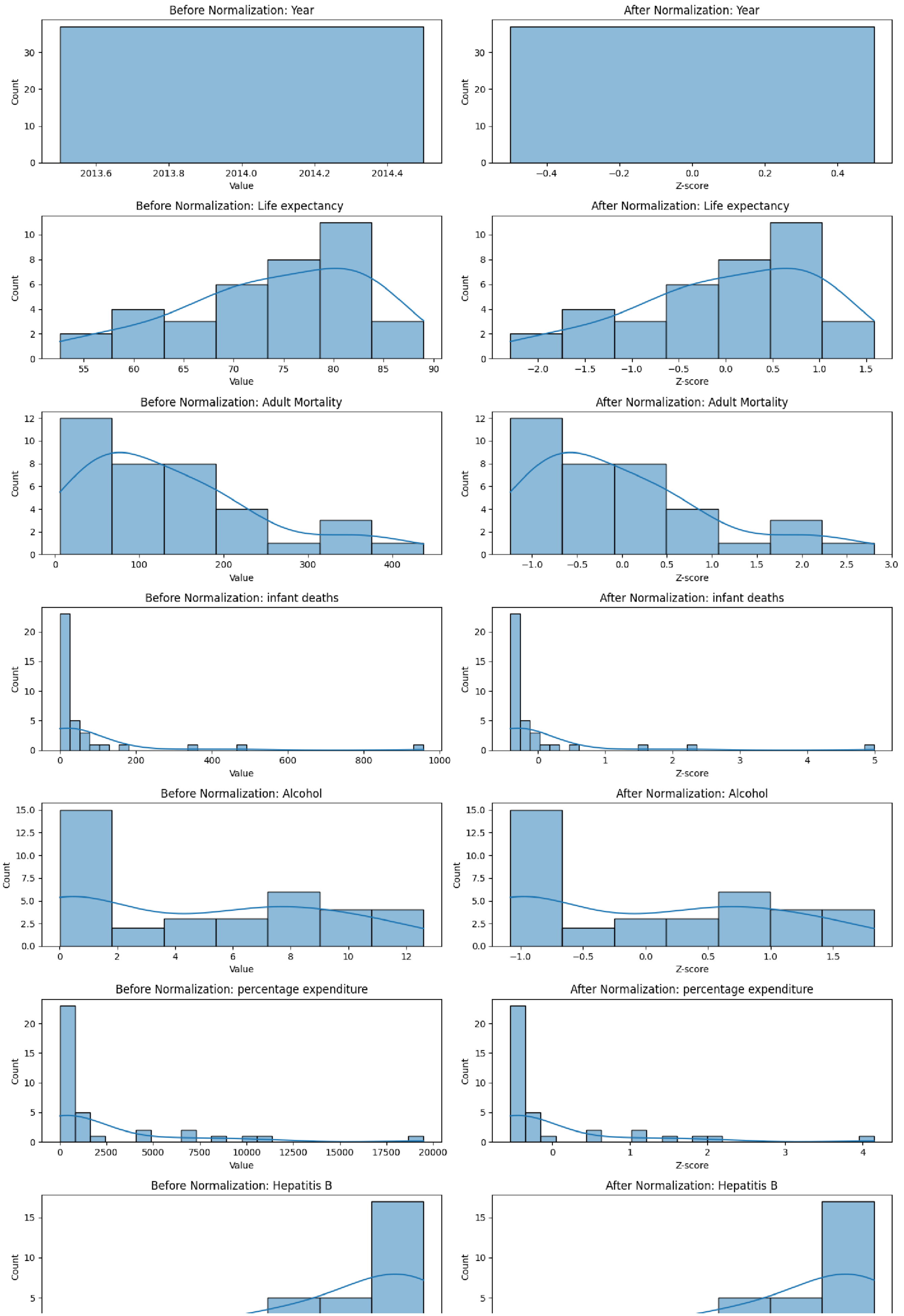
```
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
```

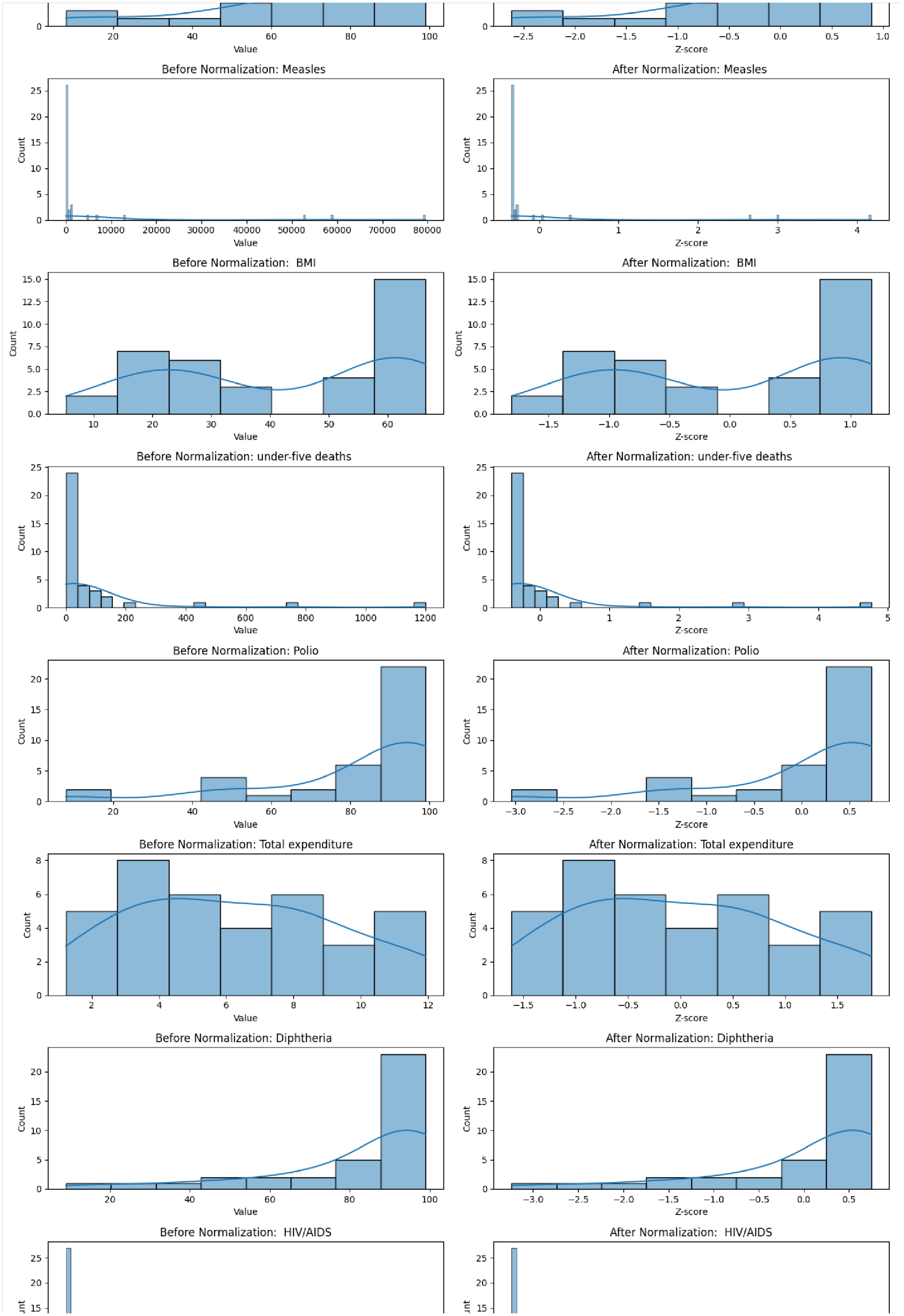
```
df_numeric = df[numeric_cols] scaler = StandardScaler() df_numeric_normalized =
pd.DataFrame(scaler.fit_transform(df_numeric), columns=numeric_cols) print("Z-score
Normalized Data:") print(df_numeric_normalized.head())
```

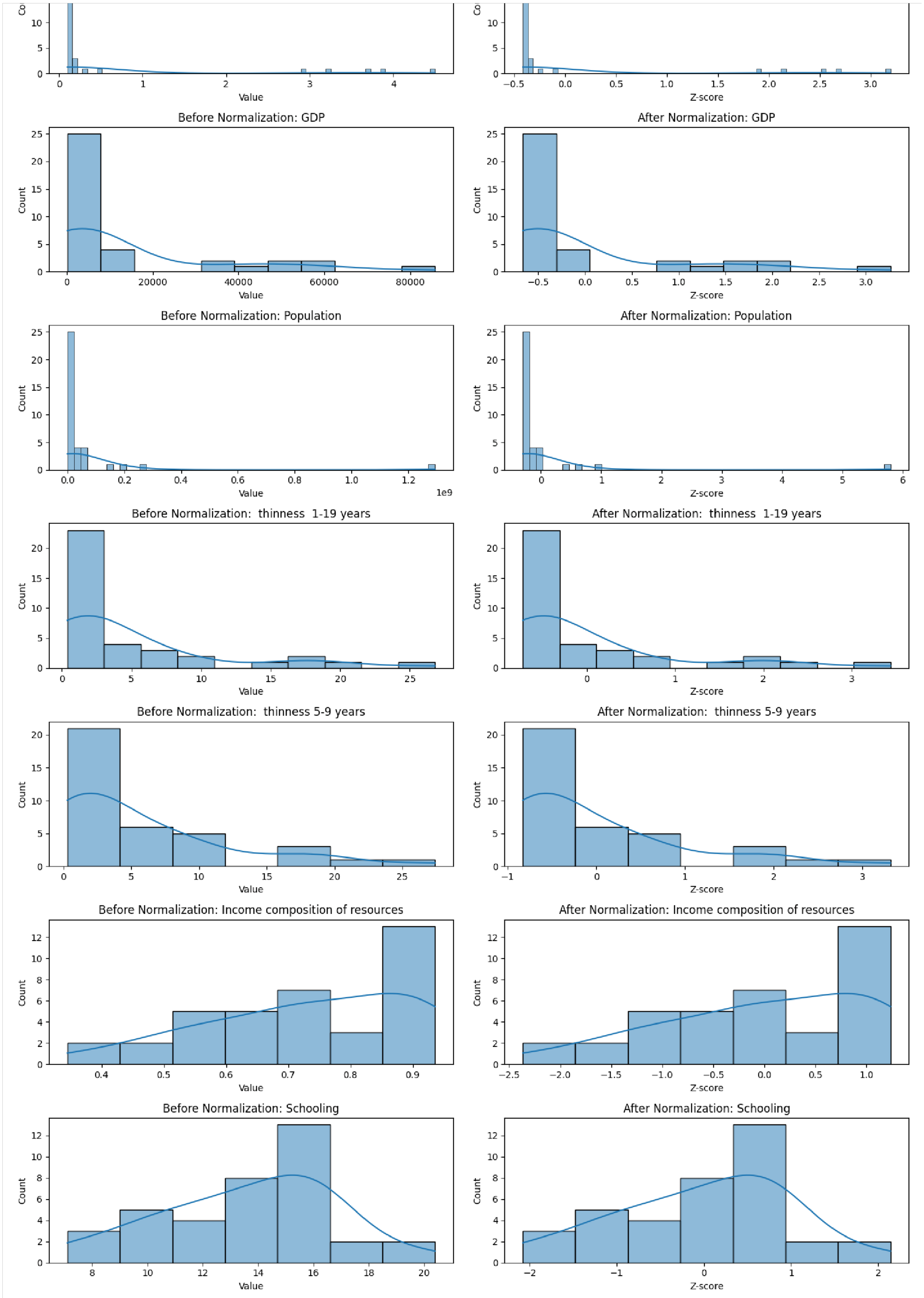
```
Z-score Normalized Data:
   Year  Life expectancy   Adult Mortality  infant deaths   Alcohol  \
0   0.0         -1.517367          1.243871      -0.046452 -1.082462
1   0.0          0.911111         -1.242096      -0.401464  1.159808
2   0.0          0.772645         -0.679236      -0.407099  1.763140
```

```
3   0.0         -0.292477        -0.060089       0.145142 -1.082462   4   0.0
    1.582138        -0.585425      -0.407099  1.827866

   percentage expenditure  Hepatitis B  Measles       BMI   \
0              -0.526543    -0.538579 -0.317693 -1.158468
1               2.048225     0.577049 -0.326338  1.155703
2               1.465868     0.846338 -0.339019  0.717228
3               -0.541727     0.807868 -0.329238 -1.202315   4
                1.180163     0.846338 -0.341692  1.024160

   under-five deaths      Polio  Total expenditure  Diphtheria    HIV/AIDS  \
0          -0.035392 -0.970515           0.619128    -0.891875  -0.411981
1           -0.401541  0.437683           1.017244     0.436959  -0.411981
2           -0.405849  0.686189           1.591945     0.702725  -0.411981
3           0.115375  0.644771          -1.101762     0.658431  -0.411981
4           -0.401541  0.727607          -1.496667     0.747020  -0.411981

        GDP  Population   thinness  1-19 years   thinness 5-9 years  \
0  -0.643117   -0.297465               1.971357             1.801683
1  2.180926   -0.287950             -0.692190           -0.781652
2  1.681597   -0.258756             -0.503062           -0.567648
3  -0.662744   -0.223865               2.065921             1.969830
4  1.503576   -0.298476             -0.629147           -0.720508

   Income composition of resources  Schooling
0                        -1.567660  -1.158371
1                        1.236204   2.140971
2                        0.968009   0.713371
3                        -0.994696  -1.158371
4                        0.955818   0.840269
```

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

# Create a copy for comparison df_before_norm = df_numeric.copy()
n_cols = 2 n_rows = len(numeric_cols) fig, axes =
plt.subplots(n_rows, n_cols, figsize=(14, n_rows * 3))



for i, col in enumerate(numeric_cols):
    sns.histplot(df_before_norm[col], ax=axes[i, 0], kde=True)
axes[i, 0].set_title(f'Before Normalization: {col}')
axes[i, 0].set_xlabel('Value')
            sns.histplot(df_numeric_normalized[col], ax=axes[i, 1],
kde=True)     axes[i, 1].set_title(f'After Normalization: {col}')
axes[i, 1].set_xlabel('Z-score')
plt.tight_layout()
plt.show()



print("Variance before normalization:")
print(df_before_norm.var())
print("\nVariance after normalization (should be close to 1):")
print(df_numeric_normalized.var())
```


Before Normalization: Year
Count
0
10
20
30
2013.6
2013.8
2014.0
2014.2
2014.4
Value
After Normalization: Year
Count
0
10
20
30
−0.4
−0.2
0.0
0.2
0.4
Z-score
Before Normalization: Life expectancy
Count
0
2
4
6
8
10
55
60
65
70
75
80
85
90
Value
After Normalization: Life expectancy
Count
0
2
4
6
8
10
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Z-score
Before Normalization: Adult Mortality
Count
0
2
4
6
8
10
12
0
100
200
300
400
Value
After Normalization: Adult Mortality
Count
0
2
4
6
8
10
12
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Z-score
Before Normalization: infant deaths
Count
0
5
10
15
20
0
200
400
600
800
1000
Value
After Normalization: infant deaths
Count
0
5
10
15
20
0
1
2
3
4
5
Z-score
Before Normalization: Alcohol
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0
2
4
6
8
10
12
Value
After Normalization: Alcohol
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
−1.0
−0.5
0.0
0.5
1.0
1.5
Z-score
Before Normalization: percentage expenditure
Count
0
5
10
15
20
0
2500
5000
7500
10000
12500
15000
17500
20000
Value
After Normalization: percentage expenditure
Count
0
5
10
15
20
0
1
2
3
4
Z-score
Before Normalization: Hepatitis B
Count
5
10
15
After Normalization: Hepatitis B
Count
5
10
15


0
20
40
60
80
100
Value
0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Z-score
Before Normalization: Measles
Count
0 5 10 15 20 25
0
10000
20000
30000
40000
50000
60000
70000
80000
Value
After Normalization: Measles
Count
0 5 10 15 20 25
0 1 2 3 4
Z-score
Before Normalization: BMI
Count
0.0 2.5 5.0 7.5 10.0 12.5 15.0
10 20 30 40 50 60
Value
After Normalization: BMI
Count
0.0 2.5 5.0 7.5 10.0 12.5 15.0
−1.5 −1.0 −0.5 0.0 0.5 1.0
Z-score
Before Normalization: under-five deaths
Count
0 5 10 15 20 25
0 200 400 600 800 1000 1200
Value
After Normalization: under-five deaths
Count
0 5 10 15 20 25
0 1 2 3 4 5
Z-score
Before Normalization: Polio
Count
0 5 10 15 20
20 40 60 80 100
Value
After Normalization: Polio
Count
0 5 10 15 20
−3.0 −2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5
Z-score
Before Normalization: Total expenditure
Count
0 2 4 6 8
2 4 6 8 10 12
Value
After Normalization: Total expenditure
Count
0 2 4 6 8
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
Z-score
Before Normalization: Diphtheria
Count
0 5 10 15 20
20 40 60 80 100
Value
After Normalization: Diphtheria
Count
0 5 10 15 20
−3.0 −2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5
Z-score
Before Normalization: HIV/AIDS
unt
15 20 25
After Normalization: HIV/AIDS
unt
15 20 25



```
Variance before normalization:
```

```
Year                           0.000000e+00 Life expectancy                 9.059422e+01
Adult Mortality                 1.167886e+04 infant deaths                  3.236630e+04
Alcohol                       1.923388e+01 percentage expendituture          1.773588e+07
Hepatitis B                        6.961818e+02
Measles                            3.177893e+08
BMI                               4.330079e+02
under-five deaths                  5.538873e+04
Polio                              5.991411e+02
Total expenditure                  9.970631e+00
Diphtheria                         5.238423e+02
 HIV/AIDS                          1.530270e+00
GDP                                4.890421e+08
Population                         4.627788e+16
thinness  1-19 years              4.137632e+01
thinness 5-9 years                4.398565e+01
Income composition of resources    2.766310e-02
Schooling                          1.021201e+01
dtype: float64

Variance after normalization (should be close to 1):
Year                               0.000000
Life expectancy                    1.027778
Adult Mortality                    1.027778
infant deaths                      1.027778
Alcohol                            1.027778
percentage expenditure             1.027778
Hepatitis B                        1.030303
Measles                            1.027778
BMI                               1.027778
under-five deaths                  1.027778
Polio                              1.027778
Total expenditure                  1.027778
Diphtheria                         1.027778
 HIV/AIDS                          1.027778
GDP                                1.027778
Population                         1.027778
thinness  1-19 years              1.027778
thinness 5-9 years                1.027778
Income composition of resources    1.027778
Schooling                          1.027778
dtype: float64
```

□ Z-Score Normalization and Its Effects on Variance

In this analysis, we performed **Z-score normalization** on all numeric variables in the Life Expectancy dataset. The goal of Z-score normalization is to standardize the data so that each feature has a mean of 0 and a variance of 1. This process allows us to scale features that were originally on different scales and variances, ensuring that no single feature dominates the analysis due to its scale.

Before Normalization:

The dataset showed significant variation in the scales of different features. For instance, **GDP** had a variance of **489,042,100**, while **Schooling** had a variance of only **10.21**. Such disparities suggest that the larger variance features (like **GDP** or **Population**) would heavily influence any statistical model or analysis, potentially leading to biased results.

After Normalization:

After applying **Z-score normalization**, the variance of every feature was approximately **1**. This means that all features now contribute equally in terms of their variability. Features with larger scales, like **GDP** or **Population**, no longer overpower features with smaller scales like **Schooling** or **Alcohol**.

Conclusion:

Z-score normalization effectively standardizes the dataset, ensuring that each feature has equal weight in the analysis. This is crucial for models that are sensitive to feature scaling, as it prevents certain features from dominating due to their original measurement units or variances.

As a result, this step ensures a **fair comparison** between features and is an important preprocessing step for improving **model accuracy** and **interpretability**.

TODO 2: Covariance Matrix and Eigen Decomposition (PCA) This step involves performing Principal

Component Analysis (PCA) for dimensionality reduction. You’ll compute the covariance matrix and

use eigen decomposition to find principal components. `import numpy as np`

```
df_numeric_filled = df_numeric.copy() df_numeric_filled[numeric_cols] =
df_numeric[numeric_cols].fillna(df_numeric[numeric_cols].m cov_matrix =
np.cov(df_numeric_filled[numeric_cols].values.T) print("Covariance Matrix:\n", cov_matrix)
```

```
                    4.33007853e+02 -1.80325075e+03  2.00815165e+02  2.40947583e+01
   1.98708559e+02 -1.25132733e+01  2.30915741e+05 -1.06166176e+09   -
7.32665691e+01 -9.00499775e+01  2.62613198e+00  4.65111411e+01]
                   [ 0.00000000e+00 -9.08668544e+02  7.52349324e+03  4.21408626e+04
                    -2.71811712e+02 -2.03191396e+05 -5.31416667e+02  2.68754745e+06
```

```
 [ 0.00000000e+00  1.48216884e+00 -1.37235233e+01 -1.04882695e+01
   5.09668086e-01  3.77915738e+02  1.50252778e+00 -4.56036791e+02
   2.62613198e+00 -1.42910420e+01  2.11619369e+00  1.90357740e-01
   2.07483483e+00 -1.28022748e-01  2.18514181e+03 -5.56137952e+06
  -6.33703979e-01 -7.55105405e-01  2.76631021e-02  4.97145571e-01]
 [ 0.00000000e+00  2.67222973e+01 -2.49982508e+02 -2.05957282e+02
   9.90652402e+00  7.28781678e+03  2.67055556e+01 -9.80454077e+03
   4.65111411e+01 -2.77105856e+02  3.64327327e+01  3.97888664e+00
   3.57595345e+01 -2.08569820e+00  4.26293777e+04 -1.00249753e+08
  -1.21140165e+01 -1.43221021e+01  4.97145571e-01  1.02120120e+01]]
```

TODO 3: Variance Explained by Each Principal Component This step involves calculating how much variance each principal component explains.

```
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

print("\nEigenvalues:\n", eigenvalues)
print("\nEigenvectors:\n", eigenvectors)

sorted_indices = np.argsort(eigenvalues)[::-1]  # Sort in descending order
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

print("\nSorted Eigenvalues:\n", sorted_eigenvalues)
print("\nSorted Eigenvectors:\n", sorted_eigenvectors)
```

```
   1.02692962e-01  5.26680475e-02 -2.51510452e-03  2.16148284e-01
1.40854750e-01  9.56314387e-01  4.32371900e-02  0.00000000e+00]  [
1.38788452e-05  9.78531329e-01  9.59132447e-02 -1.82261356e-01
   5.25842295e-03  5.17929844e-03 -1.58832046e-03 -8.46815938e-04
3.05430760e-04  1.56770982e-04  1.24238512e-04  8.19569723e-05
   9.28291143e-06 -6.22241511e-05 -1.35593715e-04 -3.46286336e-05
7.52048781e-05 -3.04829392e-05 -2.32112543e-06  0.00000000e+00]  [-
9.99999998e-01  1.93825755e-05 -5.42276669e-05 -6.42325891e-07
  -9.09363513e-07  2.44011421e-07 -2.03792944e-09  5.86180670e-09
   6.07056929e-09 -5.93683057e-08 -4.78823493e-08  1.42408197e-08   -
5.36176179e-09  4.47670129e-09  6.52630360e-10 -1.72095896e-09
   5.46391728e-09 -1.31696236e-09 -8.24247600e-11  0.00000000e+00]  [-
1.74735559e-08 -7.96207523e-05 -6.41863048e-05  4.97051747e-04
   6.80726508e-03  8.09031109e-03  3.99476087e-02 -1.57359552e-01   -
5.36465872e-02  3.40280559e-01 -7.55128931e-02  1.99584781e-01
  -4.50409802e-01 -3.29287535e-01  1.14012396e-01  6.88670783e-01
   4.47100369e-02 -9.35612146e-02 -2.59819520e-03  0.00000000e+00]  [-
1.69767823e-08 -1.02740967e-04  5.29427858e-06  8.00759789e-04
   1.30252501e-02  1.08782171e-02  4.21893852e-02 -1.64542258e-01   -
5.27998888e-02  2.47977926e-01 -1.35659733e-01  1.97119062e-01
  -4.84449719e-01 -3.19848122e-01  1.32573278e-02 -6.77150314e-01
  -4.45345465e-02  2.27010771e-01  9.84842844e-03  0.00000000e+00]
 [ 1.20173603e-10  4.28851535e-06  3.57819859e-07 -3.09954686e-05
  -3.59639870e-04 -7.95460957e-04  3.04115001e-04  3.13636223e-03
   3.11323734e-03 -3.44911764e-03  3.37303034e-03 -7.67789173e-03   -
5.51382593e-03 -1.86395910e-03 -6.08903568e-03  4.04106678e-04
  -1.05307640e-02 -4.27850330e-02  9.98941850e-01  0.00000000e+00]
 [ 2.16625600e-09  8.41269342e-05 -9.48582249e-07 -7.12535131e-04
  -6.93318308e-03 -1.29028486e-02  1.30302989e-03  4.91811768e-02
   5.08961198e-02 -9.47519258e-02  2.99295592e-02 -1.53693154e-01   -
7.31950392e-02 -1.11428700e-01 -1.17781462e-01  1.07990085e-01
  -9.48786040e-01  1.33107880e-01 -7.61025589e-03  0.00000000e+00]]
```

```
explained_variance_ratio = eigenvalues/np.sum(eigenvalues) print("Explained

Variance by each Principal Component:", explained_variance_ratio)
```

```
Explained Variance by each Principal Component: [9.99999985e-01 1.07990784e-08 3.6833716
 5.90284440e-13 1.77075055e-13 2.21080425e-14 5.33867498e-15  3.80609095e-
15 1.51743076e-15 1.12772442e-15 3.69438102e-16
 2.53712985e-16 1.62128802e-16 8.27702013e-17 3.36695749e-17
 2.31379426e-17 5.59021191e-18 9.29203283e-21 0.00000000e+00]
```

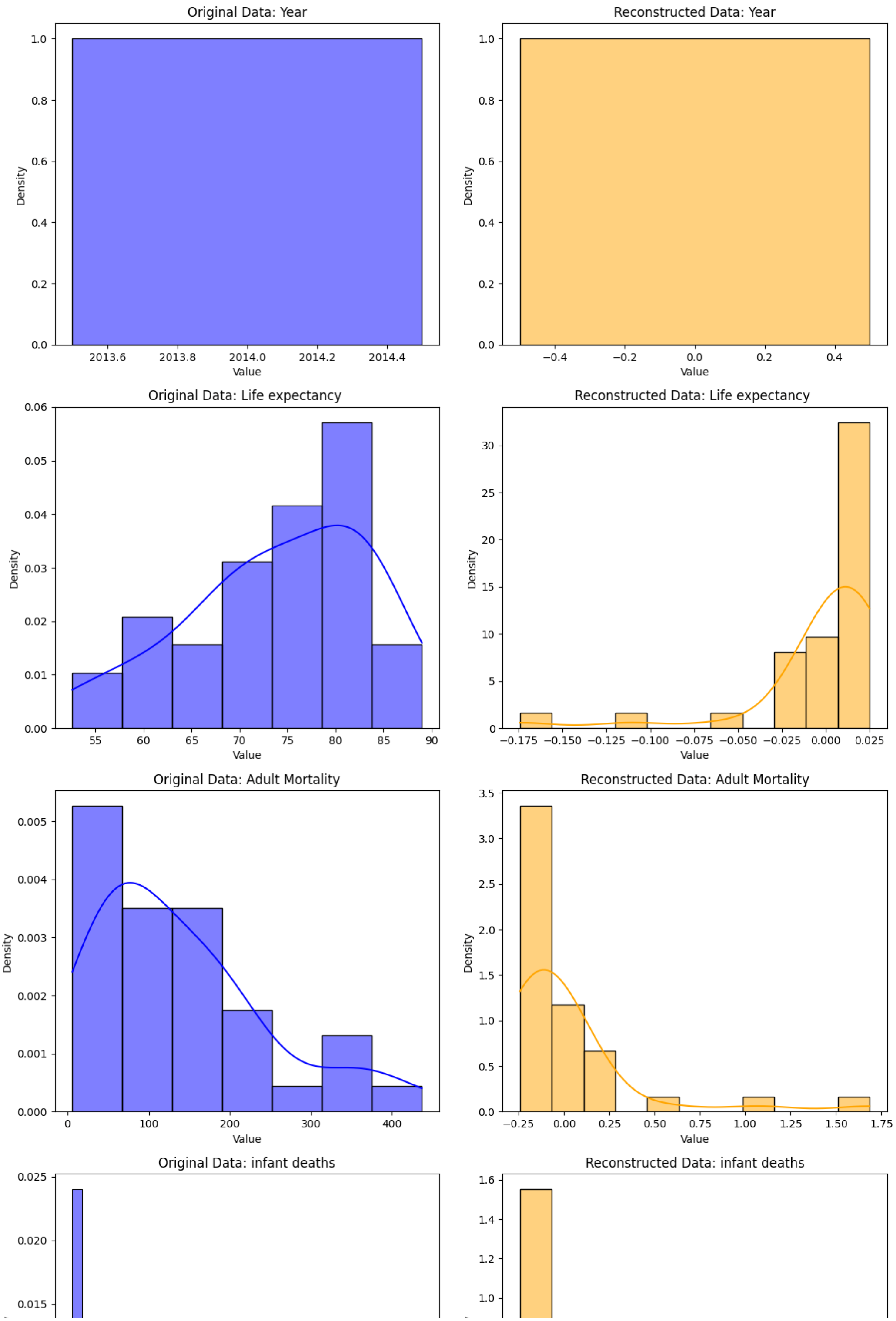
TODO 4: Reconstructing the Data Using Principal Components This task would involve using the principal components to approximate the original data by reconstructing it from the top few principal components.

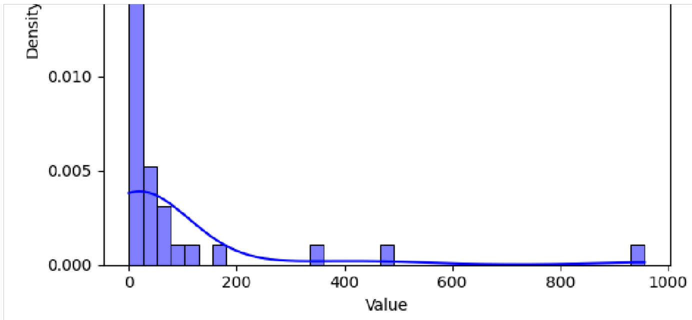
```
k = 5 top_k_components =
sorted_eigenvectors[:, :k]


projected_data = np.dot(df_numeric_normalized, top_k_components)


reconstructed_data = np.dot(projected_data, top_k_components.T)

reconstructed_df = pd.DataFrame(reconstructed_data, columns=numeric_cols)


print("Reconstructed Data (using top 5 principal components):")
print(reconstructed_df.head())
```

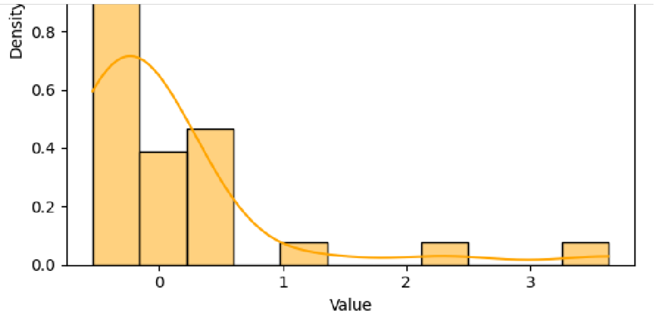
```
Reconstructed Data (using top 5 principal components):
   Year  Life expectancy   Adult Mortality  infant deaths   Alcohol  \
0   0.0         -0.011230         0.106865       0.240210 -0.003006
1   0.0          0.024098        -0.225267      -0.536373  0.006616
2   0.0          0.020992        -0.197572      -0.460648  0.005702
3   0.0         -0.004820         0.044533       0.108837 -0.001365   4   0.0
    0.021196        -0.200790      -0.458546  0.005700

   percentage expenditure  Hepatitis B  Measles       BMI   \
0              -0.483548    -0.007132 -0.318708 -0.010868
1               2.014013     0.009474 -0.325682  0.021145
2               1.439908     0.010120 -0.338405  0.019218
3               -0.548349    -0.001260 -0.328882 -0.003885   4
                1.150983     0.012050 -0.341098  0.020176

   under-five deaths      Polio  Total expenditure  Diphtheria    HIV/AIDS  \
0           0.356724 -0.009975          -0.001941    -0.012211   0.001519
1           -0.796178  0.018773           0.004853     0.023441  -0.003286
2           -0.683713  0.017161           0.004004     0.021311  -0.002849
3           0.161736 -0.003532          -0.001055    -0.004387   0.000665
4           -0.680554  0.018127           0.003825     0.022391  -0.002864

        GDP  Population   thinness  1-19 years   thinness 5-9 years  \
0  -0.654970   -0.297465              0.002969            0.005650
1  2.190484   -0.287949             -0.006301           -0.012178
2  1.688719   -0.258755             -0.005490           -0.010592
3  -0.661805   -0.223865              0.001287            0.002439
4  1.511349   -0.298476             -0.005548           -0.010679

   Income composition of resources  Schooling
0                        -0.000154  -0.002929
1                        0.000326   0.006110
2                        0.000286   0.005383
3                        -0.000065  -0.001201
4                        0.000290   0.005495
```

TODO 5: Compare Original Data with Reconstructed Data The task here is to compare the original data with the reconstructed data to see the difference between the two.

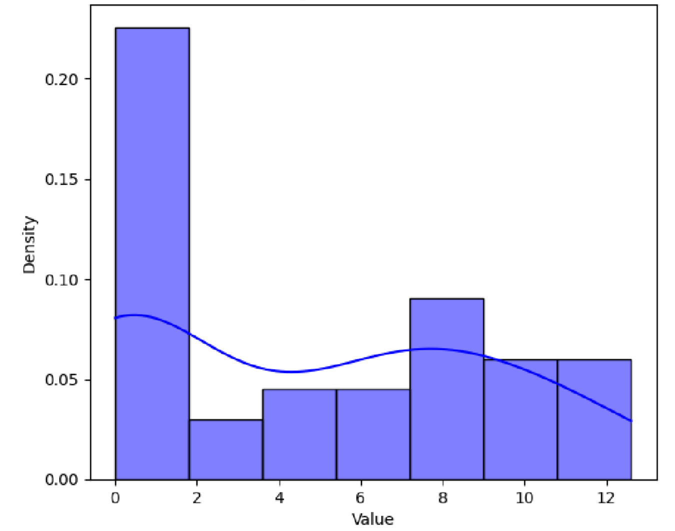
```
columns_to_compare = numeric_cols


n_cols = len(columns_to_compare) fig, axes =
plt.subplots(n_cols, 2, figsize=(12, 5 * n_cols))


for i, col in enumerate(columns_to_compare):
    sns.histplot(df_numeric_filled[col], ax=axes[i, 0], kde=True, color='blue', label='Origi
sns.histplot(reconstructed_df[col], ax=axes[i, 1], kde=True, color='orange', label='Reco
axes[i, 0].set_title(f'Original Data: {col}')     axes[i, 0].set_xlabel('Value')
axes[i, 0].set_ylabel('Density')     axes[i, 1].set_title(f'Reconstructed Data: {col}')
axes[i, 1].set_xlabel('Value')     axes[i, 1].set_ylabel('Density')


plt.tight_layout()
plt.show()


difference = df_numeric_filled[columns_to_compare] - reconstructed_df
print("\nDifference between Original and Reconstructed Data (first 5 rows):")
print(difference.head())
```


Original Data: Year
Density
1.0
0.8
0.6
0.4
0.2
0.0
2013.6
2013.8
2014.0
2014.2
2014.4
Value
Reconstructed Data: Year
Density
1.0
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
Value
Original Data: Life expectancy
Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
55
60
65
70
75
80
85
90
Value
Reconstructed Data: Life expectancy
Density
30
25
20
15
10
5
0
−0.175
−0.150
−0.125
−0.100
−0.075
−0.050
−0.025
0.000
0.025
Value
Original Data: Adult Mortality
Density
0.005
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
Value
Reconstructed Data: Adult Mortality
Density
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Value
Original Data: infant deaths
0.025
0.020
0.015
Reconstructed Data: infant deaths
1.6
1.4
1.2
1.0

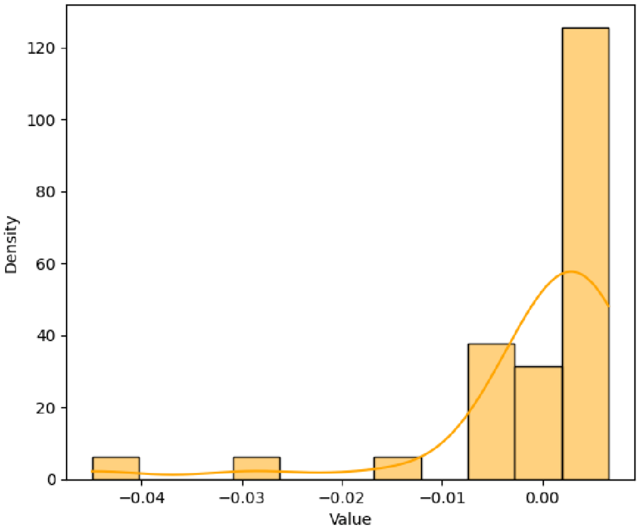




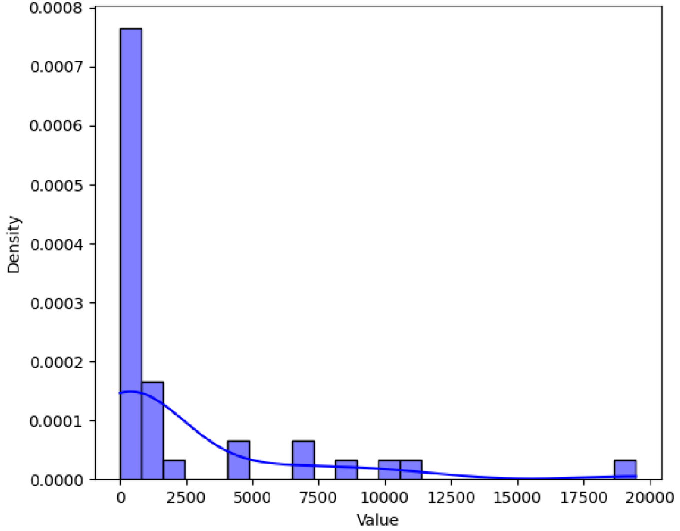
Original Data: Alcohol



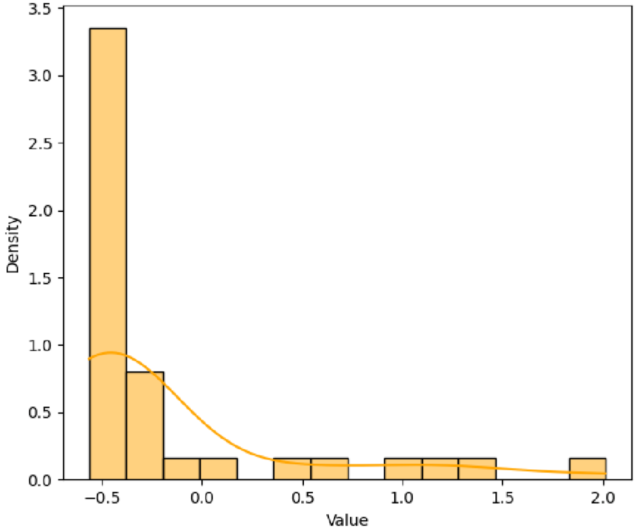
Reconstructed Data: Alcohol



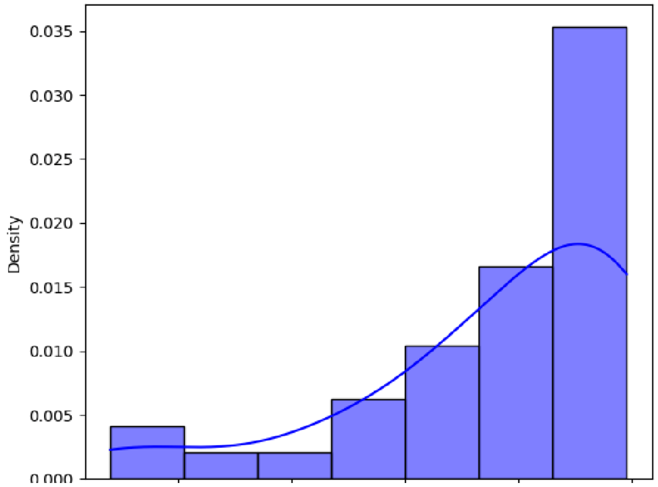
Original Data: percentage expenditure



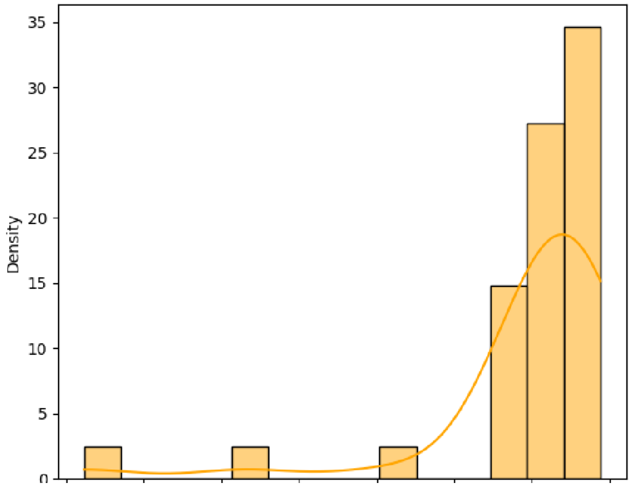
Reconstructed Data: percentage expenditure



Original Data: Hepatitis B



Reconstructed Data: Hepatitis B



```
Difference between Original and Reconstructed Data (first 5 rows):
     Year  Life expectancy   Adult Mortality  infant deaths    Alcohol  \
0  2014.0         59.911230       270.893135      63.759790   0.013006
1  2014.0         82.675902         6.225267       1.536373   9.703384
2  2014.0         81.379008        66.197572       0.460648  12.314298
3  2014.0         71.404820       131.955467      97.891163   0.011365
4  2014.0         88.978804        76.200790       0.458546  12.594300

   percentage expenditure  Hepatitis B    Measles        BMI   \
0               74.007130    62.007132  492.318708  18.610868
1            10767.349037    90.990526  340.325682  66.078855
2             8348.753615    97.989880  117.338405  57.080782
3               10.994752    97.001260  289.328882  17.703885
4             7162.197940    97.987950   70.341098  63.379824

   under-five deaths       Polio  Total expenditure  Diphtheria    HIV/AIDS  \
0           85.643276  58.009975           8.181941    62.012211   0.098481
1            1.796178  91.981227           9.415147    91.976559   0.103286
2            0.683713  97.982839          11.205996    97.978689   0.102849
3          120.838264  97.003532           2.821055    97.004387   0.099335
4            1.680554  98.981873           1.586175    98.977609   0.102864

            GDP    Population   thinness  1-19 years   thinness 5-9 years  \
0    613.351484  3.275823e+05              17.497031            17.494350
...
1                         0.935674  20.393890
2                         0.891714  15.894617
3                         0.570065  10.001201
4                         0.889710  16.294505
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...