BMI 881 - HWK 1 - TRANSFORMING A TABLE INTO A GRAPH

Aurod Ounsinegad

October 17^{th}, 2024

1 Visualization of Mortality Rates

Utilizing the Matplotlib library in Python, I produced a multi-panel line plot to display the mortality rate data from Feigin et al. (2014) Lancet 383:245-255. See the plot my code provided me below:

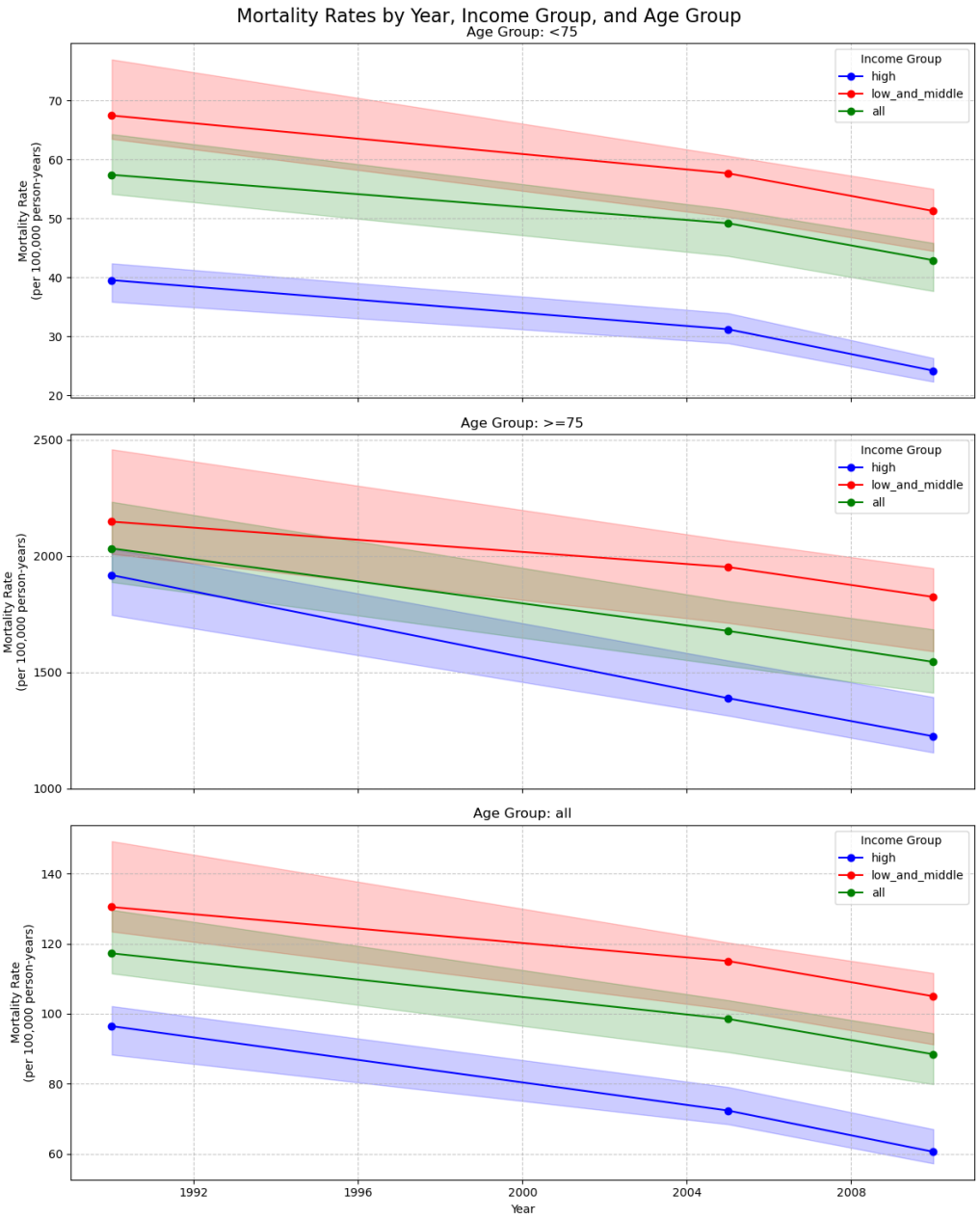

Figure 1: Mortality Rates by Year, Income Group, and Age Group

2 Design Choices

The design choices for this visualization were made to convey the intricate relationships between age groups, years, income groups, and mortality rates:

1. **Multi-panel layout:** I utilized a 3x1 grid of subplots, with each subplot (<75, ≥ 75, all) indicating a distinct age group. As a result, comparisons within and between age groups can be made clearly and without scale problems.
2. **Consistent color scheme:** High-income countries are shown in blue, low- and middle-income countries are shown in red, and the world average is shown in green. This uniformity facilitates visual comparison between panels.
3. **Linear scale:** I first thought about using a logarithmic scale but ultimately decided to use a linear scale with unique y-axis ticks for every age group. This allows for clear trend representation and a more intuitive interpretation of the data.
4. **Confidence intervals:** The 95% confidence intervals are represented by the shaded areas surrounding each line, giving the data uncertainty a visual expression.
5. **Clear labeling:** Each panel is labeled with its age group, and for ease of interpretation, the legend is the same on every panel.
6. **Time series representation:** The x-axis represents years from 1990 to 2010, clearly showing trends over time.
7. **Grid lines:** Light grid lines make it easier to read particular data and make comparisons between various income levels and years.
8. **Custom y-axis scaling:** Custom y-axis scales for each panel that are suitable for the data range they represent enhance readability without sacrificing the ability to compare trends.

These design choices produce a thorough and simple-to-understand representation that conveys the intricacy of the data effectively and permits many degrees of comparison.

3 Comprehensive Graph Concept

Below are my suggestions on how you could combine all the data from the original table into a single, comprehensive graph:

1. **Base structure:** Keep the current multi-panel arrangement as it allows for easy comparison and efficiently divides age groups.
2. **Additional metrics:** Add other graphical components, such as bar charts overlaid on the current lines, or use supplementary y-axes to incorporate additional metrics, such as incidence, prevalence, etc.
3. **Interactive elements:** Implement interactivity using tools like Plotly or Bokeh, allowing users to:
    - Toggle between different metrics
    - Zoom in on specific time periods or data ranges
    - Hover over data points to see exact values and additional information
4. **Annotations:** Add text annotations to highlight key findings or significant changes in trends.
5. **Color coding:** Use a consistent color scheme across all metrics to represent income groups, enhancing comparability.
6. **Layered information:** Develop a system that allows users to gradually add layers of data, beginning with the fundamental trends in mortality rates and progressing to increasingly intricate comparisons between indicators.

With the above method, all the data from the original table would be captured in a rich, interactive display that would still be understandable and easy to use. Users would be able to examine the data at different degrees of detail thanks to the interactivity, which makes it appropriate for both a brief summary and a thorough examination.

A Python Code

The following Python code was used to generate the visualization:

```
######Import Packages, Read CSV, & Prepare for Plotting######

import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file
df = pd.read_csv('feigin2014_table1_mortality.csv')

# Convert year to datetime for better plotting
df['year'] = pd.to_datetime(df['year'], format='%Y')

######Create Plot to Transform Table to Graph######

# Create the plot
fig, axes = plt.subplots(3, 1, figsize=(12, 15), sharex=True)
fig.suptitle('Mortality Rates by Year, Income Group, and Age Group', fontsize=16)

age_groups = ['<75', '>=75', 'all']
colors = {'high': 'blue', 'low_and_middle': 'red', 'all': 'green'}

for i, age in enumerate(age_groups):
    ax = axes[i]
    data = df[df['age_group'] == age]

    for income in ['high', 'low_and_middle', 'all']:
        income_data = data[data['income_group'] == income]
        ax.plot(income_data['year'], income_data['mortality_rate'],
                label=income, color=colors[income], marker='o')

        # Add confidence interval
        ax.fill_between(income_data['year'],
                        income_data['interval_low'],
                        income_data['interval_high'],
                        alpha=0.2, color=colors[income])

    ax.set_title(f'Age Group: {age}')
    ax.set_ylabel('Mortality Rate\n(per 100,000 person-years)')
    ax.legend(title='Income Group')
    ax.grid(True, linestyle='--', alpha=0.7)

    # # Uncomment this to set y-axis to logarithmic scale with custom tick
      locations
    # ax.set_yscale('log')

    # Custom y-axis ticks based on the age group
    if age == '<75':
        yticks = [20, 30, 40, 50, 60, 70]
    elif age == '>=75':
        yticks = [1000, 1500, 2000, 2500]
    else:  # 'all'
        yticks = [60, 80, 100, 120, 140]

    ax.set_yticks(yticks)
    ax.set_yticklabels([str(y) for y in yticks])

plt.xlabel('Year')
plt.tight_layout()
plt.show()
```