How to Create a Population Pyramid in Python?

Authored by

stats writer

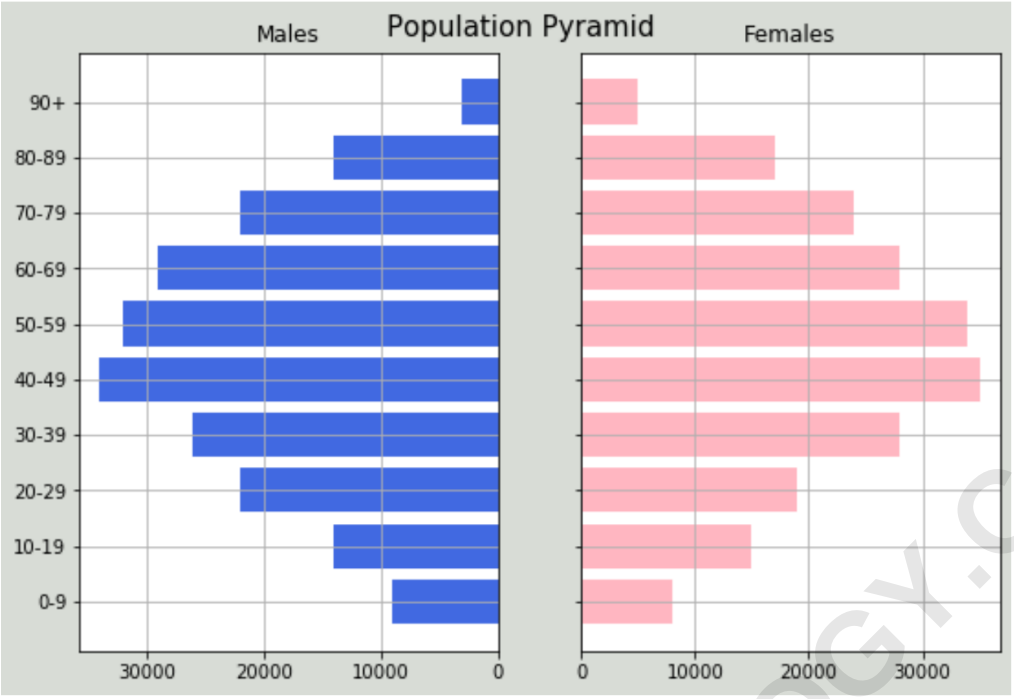
December 25, 2025

Creating a Population Pyramid in Python is a fundamental skill for data scientists and demographers alike. This powerful visualization technique uses the widely adopted matplotlib library to generate a clear, graphical representation of the age and gender distribution within a specific population. The resulting chart provides immediate insights into population structure, growth trends, and potential future demographic challenges.

The core methodology relies on the `matplotlib.pyplot.barh()` function, which is essential for plotting horizontal bar charts. By manipulating parameters such as color, size, axis labeling, and alignment, we can create a complex, bifurcated graph that efficiently displays demographic realities. This technique extends beyond simple reporting; it is crucial for analyzing phenomena such as age-related mortality rates, regional differences in birth rates, and the overall impact of migration on population composition.

A **population pyramid**, often referred to as an age-sex pyramid, is a graphical illustration that displays the distribution of various age groups in a population, which is typically divided by gender. The shape of the pyramid--whether broad at the base (indicating high birth rates) or narrower (indicating aging populations)--is highly useful for understanding the composition of a population and predicting future demographic trends, particularly concerning workforce capacity and social security needs.

This comprehensive tutorial will guide you through the process of setting up your Python environment, preparing the required dataset using the pandas library, and employing advanced Matplotlib plotting techniques to successfully generate a professional-grade population pyramid. By the end of this guide, you will be able to replicate and customize the visualization shown below.



Introduction to Population Pyramids

The population pyramid is not just a chart; it is a critical tool in demographic analysis. Its design, featuring horizontal bars representing population counts across specific age brackets (usually 5-year intervals) for males and females, allows for rapid interpretation of a country's demographic history. Typically, males are plotted on the left side (represented by negative values in the horizontal axis calculation, though plotted positively) and females on the right.

Analyzing the shape of the pyramid provides key insights into the demographic transition model a country is experiencing. A triangular shape indicates a young, rapidly growing population (common in developing nations), while a rectangular or "beehive" shape suggests a slow-growth or stable population with higher life expectancy (typical of developed nations). Conversely, an inverted pyramid signals population decline, where the older population surpasses the younger cohorts.

In this tutorial, we focus on utilizing the robust scientific computing libraries available in Python to automate the creation of this complex visualization, ensuring accuracy and reproducibility in demographic reporting. Our goal is to transform raw demographic data into an immediately understandable visual narrative.

Preparing Demographic Data in Python

To successfully generate the visualization, we first require a structured dataset detailing the population distribution by age group and gender. For this example, we will assume we have

aggregated data for a hypothetical country. This data must be organized efficiently, which is best handled using a Dataframe provided by the pandas library in Python.

The dataset structure is straightforward, consisting of three columns: the categorical age groups, and the population counts for 'Male' and 'Female' corresponding to those age groups. The use of pandas ensures that data manipulation, filtering, and preparation steps are executed with high efficiency before the data is passed to the plotting library.

Below is the initial code snippet demonstrating the required library imports--NumPy for numerical operations, pandas for data structure management, and Matplotlib for visualization--followed by the creation and display of our sample Dataframe.

Analyzing the Sample Dataset Structure

The sample dataset provides a clear breakdown of the population across ten age categories, ranging from '0-9' up to '90+'. Notice that the population figures are expressed in thousands, representing counts for males and females within each corresponding age bracket. This structure is ideal for direct input into the horizontal bar plotting function.

#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#create dataframe
df = pd.DataFrame({'Age': ,
'Male': ,
'Female': })
#view dataframe
df

Age Male Female
0 0-9 9000 8000
1 10-19 14000 15000
2 20-29 22000 19000
3 30-39 26000 28000
4 40-49 34000 35000
5 50-59 32000 34000
6 60-69 29000 28000
7 70-79 22000 24000
8 80-89 14000 17000

9 90+ 3000 5000

A quick inspection of the displayed Dataframe reveals that the population peaks in the middle-age groups (40-49 and 50-59), which is characteristic of countries with stabilizing populations and high life expectancy. We observe a slightly higher male population in the younger brackets but a notably higher female population in the older brackets (70+), reflecting typical biological longevity differences.

This organized structure is critical because the plotting process requires two separate, symmetrically arranged horizontal bar charts that share a common vertical axis (the Age categories). The next step involves configuring the Matplotlib environment to facilitate this dual plotting design.

Constructing the Visualization Framework

The population pyramid requires plotting two distinct bar charts side-by-side using the same Y-axis labels. In Matplotlib, this is achieved efficiently by using subplots. We define a figure (`fig`) and an array of axes (`axes`), specifying that we want two columns (`ncols=2`) that share the Y-axis (`sharey=True`). The figure size is also defined to ensure proper aspect ratio.

The critical step in creating the pyramid effect is inverting the X-axis for the male population data. While the population counts are positive values, inverting the axis visually places the male bars extending to the left from the central Y-axis, creating the characteristic pyramid shape.

We begin the plotting script by defining the data limits and setting up the figure environment, including a custom background color and a centered title, which significantly improves the visual presentation.

Plotting the Male and Female Distributions

With the framework established, we proceed to plot the data using the `barh()` function for horizontal bars. This function takes the Y coordinates (the index range of the age groups) and the X values (the population counts) as primary inputs. Separate plotting calls are necessary for the male and female populations.

For the male population (plotted on `axes`), we explicitly set the title to 'Males' and assign a distinct, professional color, such as 'royalblue'. For the female population (plotted on `axes`), we use 'lightpink' and assign the title 'Females'. The `align='center'` parameter ensures the bars are centered on the ticks.

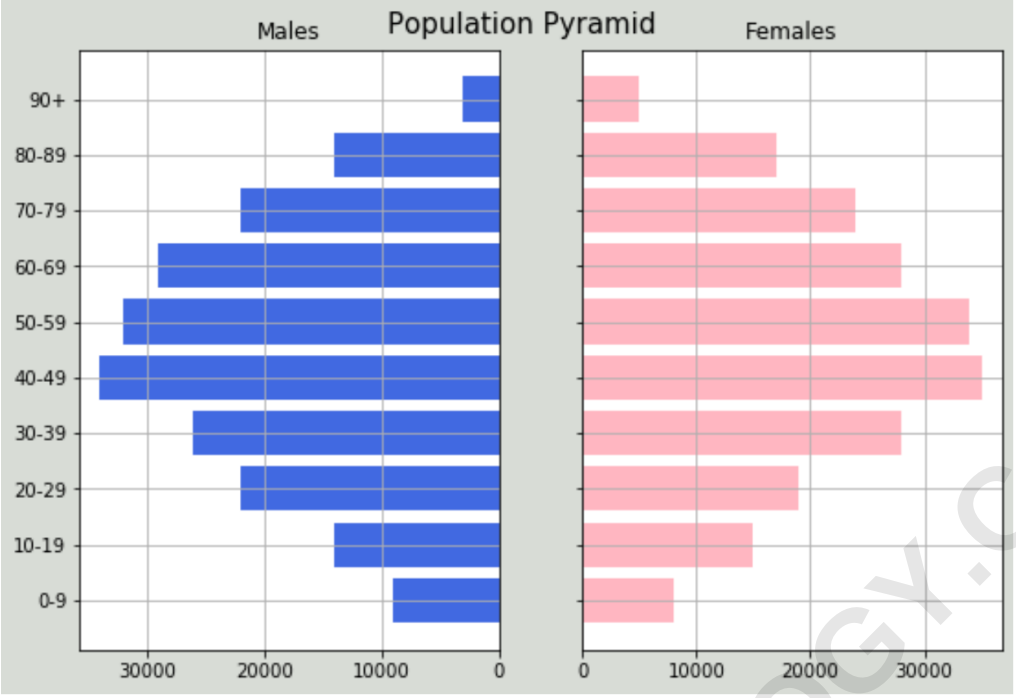
The final steps involve rigorous customization of the axes. We use `axes.invert_xaxis()` to flip

the X-axis for the male data, achieving the desired leftward extension. We then define the Y-axis ticks and labels using the age categories from the Dataframe, ensuring both subplots share these labels correctly. Adding a grid to both plots enhances readability by allowing easy comparison of bar lengths across the population counts.

The complete, highly documented code for generating the standard population pyramid is presented below:

```
#define x and y limits
y = range(0, len(df))
x_male = df
x_female = df

#define plot parameters
fig, axes = plt.subplots(ncols=2, sharey=True, figsize=(9, 6))

#specify background color and plot title
fig.patch.set_facecolor('xkcd:light grey')
plt.figtext(.5,.9,"Population Pyramid ", fontsize=15, ha='center')

#define male and female bars
axes.barh(y, x_male, align='center', color='royalblue')
axes.set(title='Males')
axes.barh(y, x_female, align='center', color='lightpink')
axes.set(title='Females')

#adjust grid parameters and specify labels for y-axis
axes.grid()
axes.set(yticks=y, yticklabels=df)
axes.invert_xaxis()
axes.grid()

#display plot
plt.show()
```

Executing this script produces the high-quality demographic visualization that accurately reflects the input data distribution. The separation of the male and female data onto two mirrored axes is the definitive visual characteristic of a population pyramid.



Interpreting the Generated Population Pyramid

Upon reviewing the resulting visualization, several key demographic observations become immediately apparent. The distribution of males and females in this specific population is largely symmetrical, suggesting balanced gender ratios across most age groups, which is a common feature in many industrialized nations lacking significant gender-specific migration patterns.

Furthermore, the widest bars appear prominently in the 40-49 and 50-59 age brackets. This middle-heavy structure indicates a large cohort of individuals approaching retirement age, often referred to as a "bulge" generation. This pattern implies that the country experienced a period of high birth rates approximately 40 to 60 years prior, followed by a slowdown in recent decades, as evidenced by the slightly narrower base (age 0-9).

By simply analyzing the shape of this single plot, we gain a decent and comprehensive understanding of the country's demographic profile, including evidence of increasing life expectancy (indicated by significant populations in the 70+ categories) and the future challenges related to supporting an aging populace.

Advanced Customization: Adjusting Aesthetics and Color Schemes

A crucial advantage of using Matplotlib is the extensive flexibility it offers in customizing the appearance of the output. While the default colors chosen ('royalblue' and 'lightpink') offer professional contrast, demographic charts often need to adhere to specific corporate branding or

publication standards.

The appearance of the background, the titles, and the individual bars can all be modified easily by altering the color arguments passed to the relevant Matplotlib functions. Colors can be defined using standard names (like 'red', 'blue'), hexadecimal codes, or specific named colors available through the `xkcd` color library, as demonstrated in the background setting.

For instance, we can dramatically change the visual impact by specifying a warmer palette, perhaps utilizing 'hotpink' and 'dodgerblue' for the bars against a softer 'beige' background. The code snippet below illustrates how to implement these changes by overriding the previous color specifications without altering the fundamental structure of the plot:

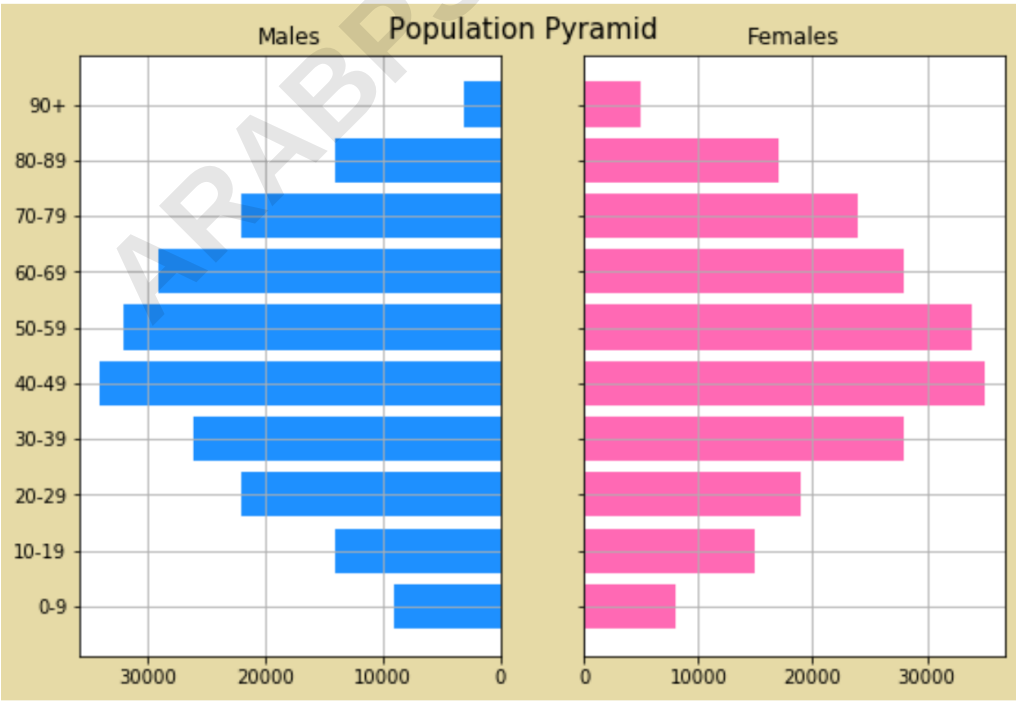
fig.patch.set_facecolor('xkcd:beige')

axes.barh(y, x_male, align='center', color='dodgerblue')

axes.barh(y, x_female, align='center', color='hotpink')

plt.show()

This simple modification demonstrates the ease with which presentation aesthetics can be tailored. Selecting the appropriate color scheme is vital, as it can affect how quickly and accurately the audience interprets the demographic data, ensuring that the chart remains visually appealing while serving its analytical purpose.



Conclusion: The Power of Demographic Visualization

The ability to programmatically generate complex visualizations like the population pyramid using Python and Matplotlib offers significant advantages in data analysis workflows. This method ensures that large datasets can be processed and visualized efficiently, providing dynamic tools for demographers, public health researchers, and policy makers.

We have successfully walked through the entire process: from loading and structuring population data using pandas, setting up a dual-axis visualization framework, implementing the critical axis inversion necessary for the pyramid shape, and finally, customizing the resulting plot for optimal aesthetic impact. The principles demonstrated here can be applied to any dataset structured by categorical groups and two comparative numerical metrics.

We encourage users to experiment freely with different color palettes and figure parameters to optimize the visual communication of their unique demographic data. Mastering this technique is a powerful addition to any data scientist's toolkit, enabling clearer communication of vital population statistics.