Mastering Data Visualization with Seaborn: Case Studies, Categorical Data, and Practical Applications for AI Engineers

Minor In AI, IIT Ropar
5th March, 2025

Welcome!

Welcome, aspiring AI engineers, to the fascinating world of data visualization! In this book, we'll embark on a journey to master the art of transforming raw data into insightful visuals using Seaborn, a powerful Python library. We'll focus on practical applications particularly relevant to your future careers.

Figure 1: Hehe, penguin (look forward for penguin data)

Setting the Stage - Categorical vs. Continuous Data

Imagine you're building an AI model to understand user behavior on an online streaming platform (like Netflix, Hulu, etc). You have a lot of data. What do you do with it? You have *watch time*, *movie genre*, *subscription type*, and *ratings*.

Before we dive into creating visualizations, it's crucial to understand the different *types* of data we'll be working with. This will determine the most appropriate and effective ways to represent it visually.

Specifically, let's consider these distinctions:

- **Categorical Data:** Think of data that falls into distinct categories or groups. These categories are usually limited in number. In our streaming platform example, *movie genre* (action, comedy, drama, thriller) and *subscription type* (free, standard, premium) are examples of categorical data. The number of options are fixed, or at least finite.
- **Continuous Data:** This type of data can take on any value within a given range. It's often numerical and can have a very large (or infinite) number of possible values. *Watch time* (120 minutes, 98.3 minutes, etc.) and *ratings* (0.1 to 10) are prime examples of continuous data in our streaming scenario. There's theoretically a large number of values that the data points can take.

Now, let's ask ourselves a question: What about *years*? For instance, if we're looking at movies released between 1990 and 1995, does "year" become categorical, or continuous?

It Depends!

The key here is *context*.

- If you're only considering movies from 2000 to 2005, then "year" effectively becomes categorical. You have only five distinct categories.
- But if you're analyzing movies spanning from the 1800s to the present day, "year" transforms into continuous data because of the large range of possible values.

Understanding this distinction between categorical and continuous data is crucial because it affects which visualizations are most appropriate and interpretable. For example, using a scatter plot on categorical data often won't provide meaningful insights.

Introducing Seaborn - Visualizing the Penguin Data

Let's switch gears and consider a different dataset: Penguins.

Researchers are often interested in the physical characteristics of penguins, such as the length and depth of their bills (beaks). These measurements can help identify different species of penguins.

To explore this data visually, we'll use Seaborn, a Python library built on top of Matplotlib. Seaborn provides a higher-level interface for creating informative and aesthetically pleasing statistical graphics. It builds on top of Matplotlib, which is a more basic data plotting library.

`seaborn` comes pre-installed with many basic datasets for learning and playing around with.

Here's how to get started:

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

# Load the penguin dataset
penguins = sns.load_dataset("penguins")

# Create a scatter plot
sns.scatterplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species",
                style="species")

plt.show()
```

Listing 1: Creating a scatter plot with the Penguins dataset

Let's break down this code:

1. **Import Libraries:**
   - `import matplotlib.pyplot as plt`: Imports the Matplotlib library and assigns it the alias `plt`. This library provides the foundation for creating plots and visualizations in Python.
   - `import seaborn as sns`: Imports the Seaborn library and assigns it the alias `sns`. Seaborn builds on top of Matplotlib and provides a higher-level interface for creating statistical graphics.
2. **Load Dataset:**
   - `penguins = sns.load_dataset("penguins")`: Loads the "penguins" dataset from Seaborn's built-in datasets. This dataset contains information about different species of penguins and their physical characteristics.
3. **Create Scatter Plot:**
   - `sns.scatterplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species", style="species")`: Creates a scatter plot using Seaborn's `scatterplot` function.
     - `data=penguins`: Specifies the dataset to use for the plot. In this case, it's the "penguins" dataset.
     - `x="bill_length_mm"`: Specifies the column to use for the x-axis of the plot. Here, it's the "bill_length_mm" column, representing the bill length of the penguins.

- `y="bill_depth_mm"`: Specifies the column to use for the y-axis of the plot. It's the "bill_depth_mm" column, representing the bill depth of the penguins.
- `hue="species"`: Specifies the column to use for coloring the points in the plot. The points will be colored differently based on the species of the penguins, allowing us to distinguish between them visually.
- `style="species"`: Specifies the column to use for varying the style of the points in the plot. In this case, the style (e.g., shape) of the points will also vary based on the species of the penguins.

4. **Display Plot:**

   - `plt.show()`: Displays the plot that has been created using Matplotlib. This function opens a window or displays the plot inline, depending on the environment (e.g., Jupyter Notebook).

This code generates a scatter plot where each point represents a penguin. The x-axis shows the bill length, the y-axis shows the bill depth, and the color and shape of the points differentiate the penguin species.

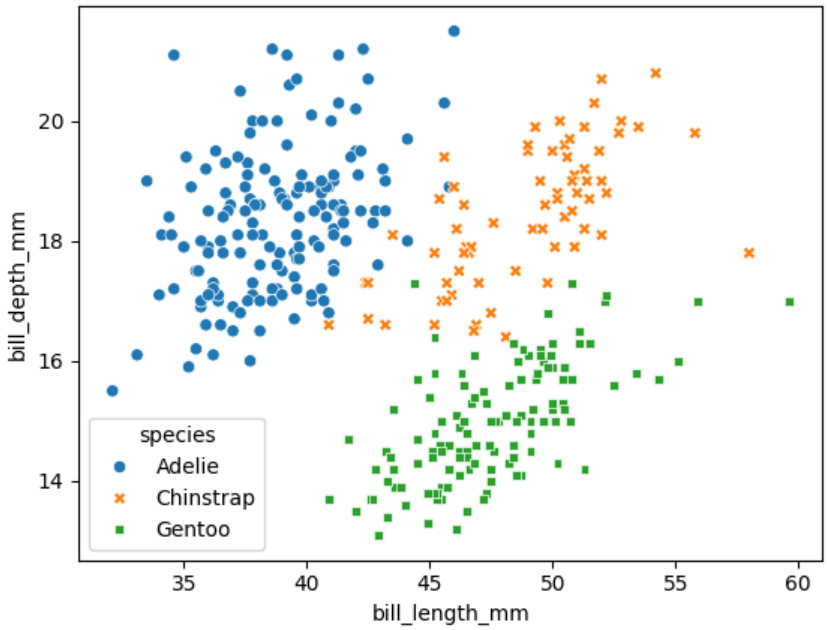

Figure 2: A scatter plot of penguin bill length vs. bill depth, colored by species.

Notice the `hue` and `style` parameters. These are where the *categorical* nature of "species" comes into play. Seaborn uses different colors (hue) and marker styles to visually distinguish the different penguin species.

- **What Happens If You Use Continuous Data for `hue`?**

  If you were to accidentally set `hue` to a continuous variable like "year" (assuming such a column existed), Seaborn would likely issue a warning. The library understands that `hue` is best suited for categorical data to create distinct visual groupings.

Exploring More Seaborn Datasets and Plot Types

Seaborn isn't just limited to penguins! It comes with several built-in datasets that allow you to explore different visualization techniques.

Let's dive into the "Titanic" dataset and see how we can use Seaborn to gain insights into the passengers and their survival rates.

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

# Load Titanic dataset
titanic = sns.load_dataset("titanic")

# Set Seaborn theme
sns.set_theme(style="whitegrid")

# Create a figure with subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# 1. Bar Plot: Survival Rate by Class
sns.barplot(data=titanic, x="class", y="survived", hue="class", estimator=lambda x: sum(x) / len(x), palette="Blues", legend=False, ax=axes[0, 0])
axes[0, 0].set_title("Survival Rate by Class")

# 2. Box Plot: Age Distribution by Class
sns.boxplot(data=titanic, x="class", y="age", hue="class", palette="Set2", legend=False, ax=axes[1, 0])
axes[1, 0].set_title("Age Distribution by Class")

# 3. Count Plot: Number of Passengers by Embark Town
sns.countplot(data=titanic, x="embark_town", hue="embark_town", palette="coolwarm", legend=False, ax=axes[0, 1])
axes[0, 1].set_title("Number of Passengers by Embark Town")

# 4. Violin Plot: Fare Distribution by Class
sns.violinplot(data=titanic, x="class", y="fare", hue="class", palette="muted", legend=False, ax=axes[1, 1])
axes[1, 1].set_title("Fare Distribution by Class")

# Adjust layout and display
plt.tight_layout()
plt.show()
```

Listing 2: Visualizing the Titanic dataset with multiple plot types

In this example, we're creating four different types of plots:

1. **Bar Plot:** Shows the survival rate for each passenger class.
2. **Box Plot:** Displays the distribution of ages within each passenger class. Box plots are useful for understanding the median, quartiles, and outliers in your data.
3. **Count Plot:** Visualizes the number of passengers who embarked from each town.
4. **Violin Plot:** Similar to a box plot, but provides a more detailed view of the data distribution. Violin plots are a combination of box plots and kernel density estimation (KDE) plots, displaying the probability density of the data at different values.

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

# Load the Flights dataset
flights = sns.load_dataset("flights")

# Set Seaborn theme
sns.set_theme(style="whitegrid")

# Create a figure with subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Line Plot: Yearly Trend of Airline Passengers
sns.lineplot(data=flights, x="year", y="passengers", estimator="sum", errorbar=None, marker="o", ax=axes[0, 0])
axes[0, 0].set_title("Total Passengers Per Year")

# 2. Box Plot: Monthly Passenger Distribution Over Years
sns.boxplot(data=flights, x="month", y="passengers", hue="month", palette="coolwarm", legend=False, ax=axes[0, 1])
```

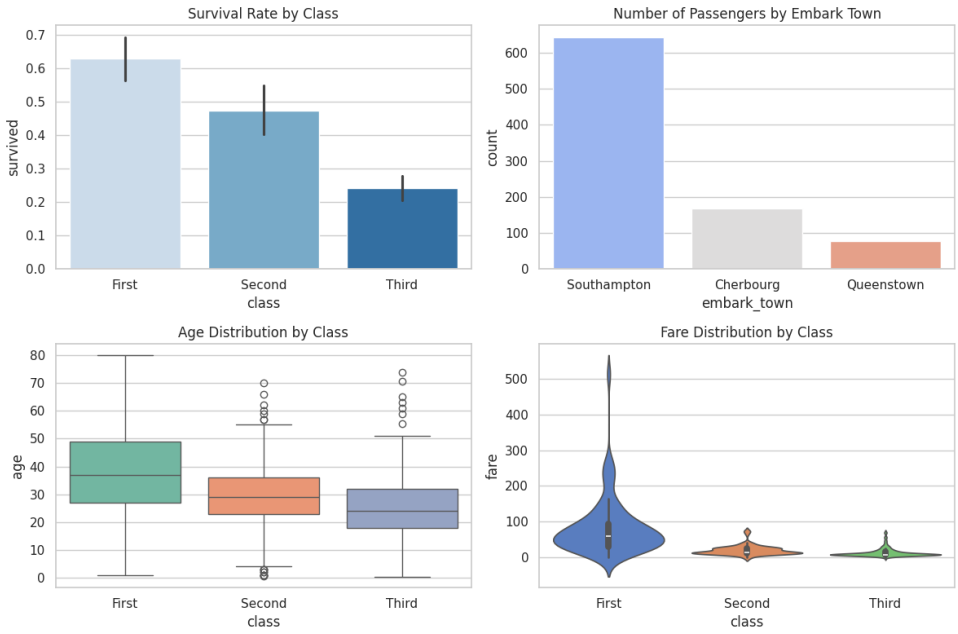

Figure 3: Multiple plots visualizing the Titanic dataset, including bar plot, box plot, count plot, and violin plot.

```
axes[0, 1].set_title("Passenger Distribution by Month")

# 3. Bar Plot: Average Monthly Passenger Count
sns.barplot(data=flights, x="month", y="passengers", hue="month", estimator="mean",
    palette="Blues", legend=False, ax=axes[1, 0])
axes[1, 0].set_title("Average Monthly Passengers")

# 4. Heatmap: Monthly Passenger Count Over Years
flights_pivot = flights.pivot(index="month", columns="year", values="passengers")
sns.heatmap(flights_pivot, annot=True, fmt="d", cmap="YlGnBu", linewidths=0.5, ax=axes
    [1, 1])
axes[1, 1].set_title("Heatmap of Monthly Passengers")

# Adjust layout and display
plt.tight_layout()
plt.show()
```

Listing 3: Visualizing the Flights dataset with multiple plot types

This example uses the "flights" dataset to illustrate how the distribution of air passengers has evolved over time. It creates four different plots:

1. **Line Plot**: Yearly Trend of Airline Passengers
2. **Box Plot**: Passenger Distribution by Month
3. **Bar Plot**: Average Monthly Passengers
4. **Heatmap**: Monthly Passenger Count Over Years

Heatmaps - Unveiling Correlations

Let's focus specifically on heatmaps because they're incredibly valuable for identifying relationships between variables.

A heatmap uses color intensity to represent the values in a matrix. In our flight data example, we can create a heatmap to visualize the number of passengers for each month and year.

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

```python

# Load dataset
flights = sns.load_dataset("flights")

# Create a pivot table
flights_pivot = flights.pivot(index="month", columns="year", values="passengers")

# Create a heatmap
sns.heatmap(flights_pivot, annot=True, fmt="d", cmap="YlGnBu", linewidths=0.5)

# Add labels
plt.xlabel("Year")
plt.ylabel("Month")
plt.title("Number of Passengers (Flights Dataset)")

plt.show()
```

Listing 4: Creating a heatmap of the Flights dataset

Key points about heatmaps:

- `annot=True`: Displays the values in each cell of the heatmap.
- `fmt="d"`: Specifies the format of the values (in this case, integers).
- `cmap="YlGnBu"`: Sets the color palette.
  - The color palette has to do with the colormap of choice for the heap map
  - Colormap has to do with what colors you want to use to represent the data points in the heatmap
  - In this case we are using the color palette 'YlGnBu'
- Lighter colors represent lower passenger counts, while darker colors represent higher passenger counts.

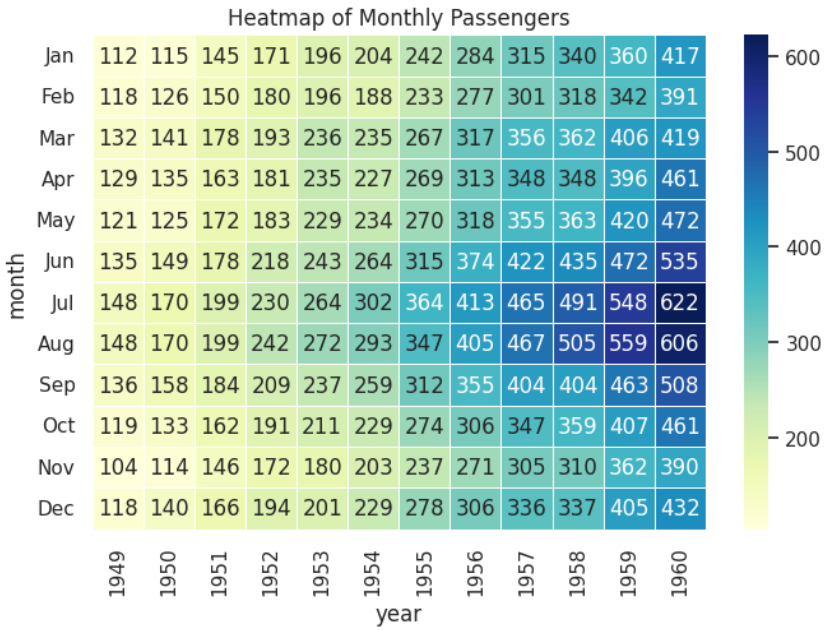

Figure 4: Heatmap visualizing monthly passenger counts over the years.

What if your Heatmap looks blank?

If your heatmap appears empty, the most likely reason is that your pivot table (the input to the heatmap) has missing values or incorrect indexing. Double-check that your data is properly structured before creating the heatmap.

We've only scratched the surface of Seaborn's capabilities in this brief introduction. As you continue your journey to mastering data visualization, remember to explore the documentation, experiment with different plot types, and always consider the type of data you're working with. With practice, you'll be able to create insightful visuals that drive data-driven decisions in your AI projects.

Happy visualizing!