Seaborn Demystified: Visual Data Analysis for Actionable Intelligence

Minor in AI, IIT Ropar

6th March, 2025

Welcome, future data explorers! This notes is your guide to mastering Seaborn, a powerful Python library for creating insightful and visually appealing data visualizations. We'll go beyond just the syntax, focusing on how to extract meaningful insights from your data and use those insights to make strategic decisions.

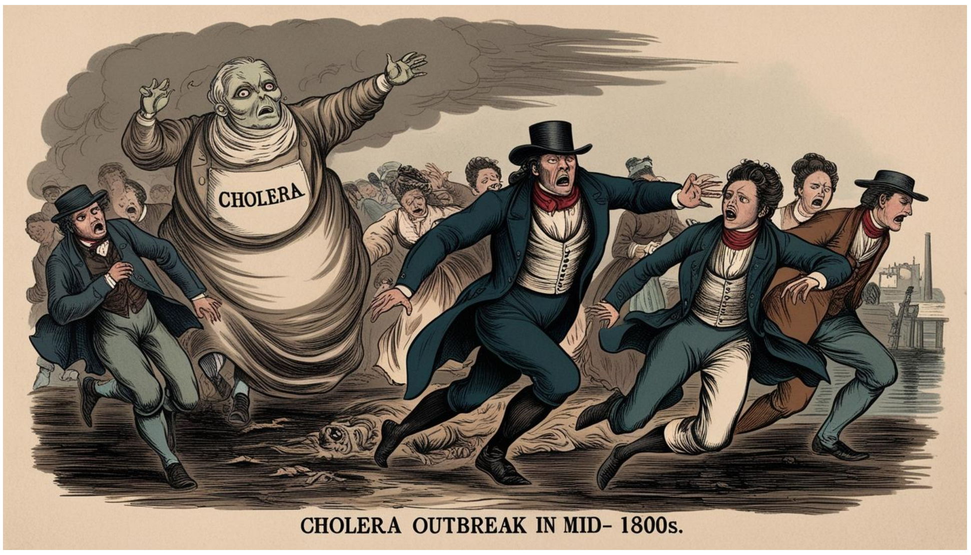

Figure 1: Scary scary Cholera back in the day!

1 The Power of Seeing: The John Snow Story

Imagine a bustling London in the mid-1800s. A terrifying outbreak of cholera grips the city, leaving residents panicked and desperate. Doctors and officials are baffled, unable to pinpoint the source of the deadly disease. Meetings are held, theories are debated, but the deaths continue.

Enter Dr. John Snow. He wasn't content with just discussing the problem. He decided to *visualize* it. He meticulously plotted each cholera death on a map of London. He then marked the locations of the water pumps throughout the city.

Suddenly, a pattern emerged. A dense cluster of deaths centered around a particular water pump on Broad Street. Dr. Snow hypothesized that the pump was contaminated and responsible for the outbreak. He convinced the authorities to shut down the pump. The result? The cholera outbreak slowed dramatically.

This story demonstrates the incredible power of visualization. Sometimes, a simple graph can reveal insights that are invisible in raw data. It can transform confusion into clarity, and ultimately, lead to life-saving decisions. This is the power of Seaborn, and this is what we'll be learning to harness in this book.

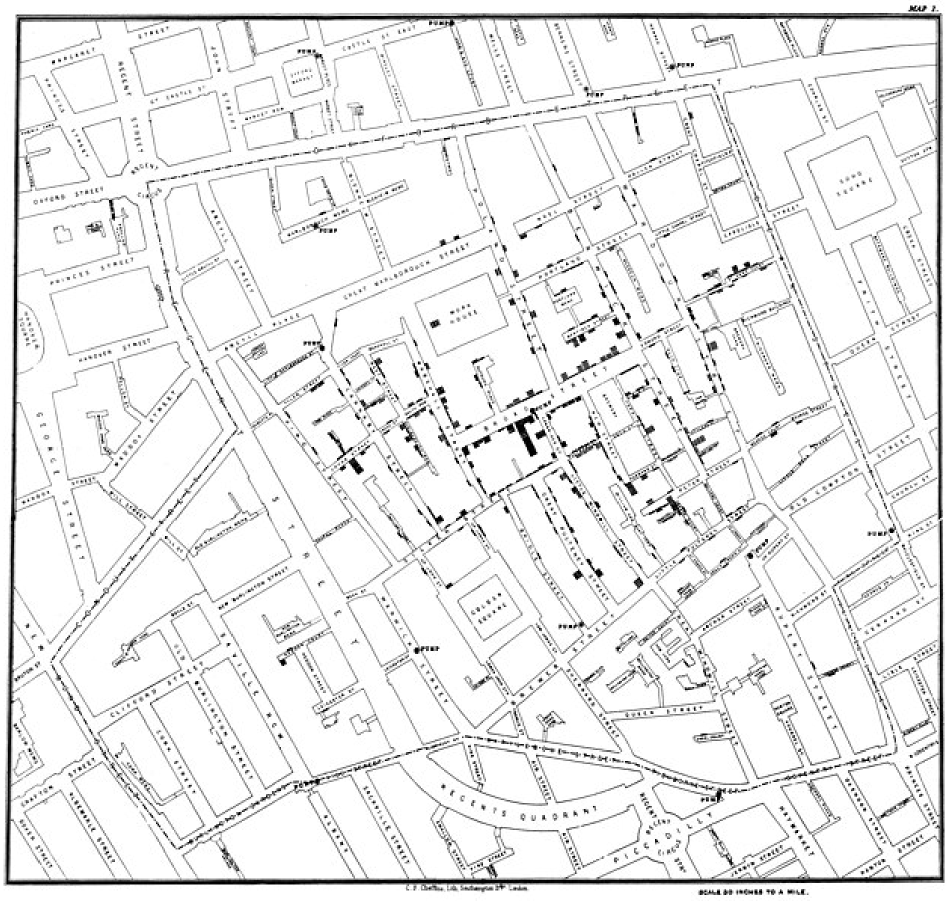

Figure 2: John Snow's cholera map showing clusters of deaths around the Broad Street pump

2 Setting the Stage: Python, Pandas, and Seaborn

Before we dive into the visual wonders of Seaborn, let's make sure our stage is set. You'll need a basic understanding of Python and the Pandas library.

- **Python:** The programming language we'll be using. If you're new to Python, there are countless online resources to get you started.
- **Pandas:** A powerful library for data manipulation and analysis. Pandas provides data structures like DataFrames that make working with tabular data a breeze.
- **Seaborn:** Our star player! Seaborn is built on top of Matplotlib and provides a high-level interface for creating informative and aesthetically pleasing statistical graphics.

3 Count Plots: Tallying the Votes

Let's start with a simple yet effective visualization: the count plot. Imagine you've conducted a survey asking people about their favorite programming language. You have a list of responses, and you want to quickly see which languages are the most popular. That's where a count plot comes in.

Here's how you can create a count plot using Seaborn:

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

# Create a simple dataset
data = pd.DataFrame({
    "Language": ["C", "Python", "Python", "Java", "JS", "C", "C++", "JS", "
    Python", "R"]
})

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

# Create countplot
sns.countplot(data=data, x="Language", hue="Language", palette="coolwarm",
    legend=False)

# Add title
plt.title("Favorite Programming Languages")

# Show plot
plt.show()
```

Listing 1: Creating a Count Plot with Seaborn

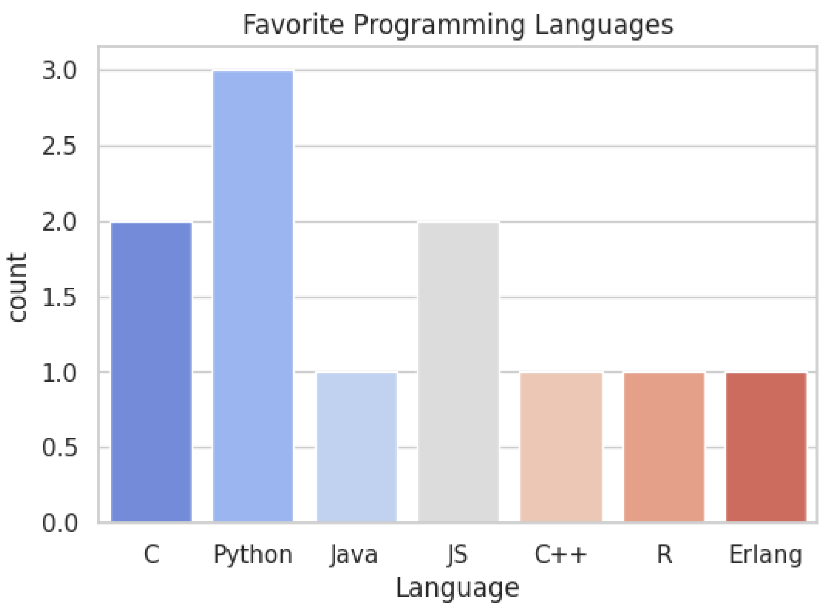

Figure 3: A count plot showing the frequency of different programming languages

Code Breakdown:

- We import the necessary libraries: `seaborn`, `matplotlib.pyplot`, and `pandas`.
- We create a Pandas DataFrame with a column called ”Language” containing the survey responses.
- `sns.set_theme(style="whitegrid")` sets a visually appealing theme for our plot.
- `sns.countplot(data=data, x="Language")` creates the count plot. We specify the DataFrame (`data`) and the column we want to count (`Language`).
- `plt.title("Favorite Programming Languages")` adds a title to our plot.
- `plt.show()` displays the plot.

A count plot essentially counts the occurrences of each category in a column and displays them as bars. Think of it like a histogram for categorical data.

4 Scatter Plots and Kernel Density Estimation (KDE): Unveiling Relationships

Now, let's move on to a more sophisticated visualization: the scatter plot. Scatter plots are perfect for exploring the relationship between two numerical variables. Imagine you're studying penguins and want to see if there's a relationship between their bill length and bill depth.

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

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

# Drop NaN values
penguins = penguins.dropna()

# Set figure size
plt.figure(figsize=(10, 6))

# Create a scatter plot with KDE overlay
sns.kdeplot(
    data=penguins,
    x="bill_length_mm",
    y="bill_depth_mm",
    hue="species",
    fill=True,
    alpha=0.4,
)
sns.scatterplot(
    data=penguins,
    x="bill_length_mm",
    y="bill_depth_mm",
    hue="species",
    edgecolor="black"
)

# Titles and labels
plt.title("Penguin Bill Dimensions: KDE and Scatter Plot", fontsize=14)
plt.xlabel("Bill Length (mm)", fontsize=12)
plt.ylabel("Bill Depth (mm)", fontsize=12)
plt.legend(title="Species")
plt.grid(True)

# Show plot
plt.show()
```

Listing 2: Creating a Scatter Plot with KDE Overlay

Insights and Strategic Decision-Making:
Now, this is where things get interesting. Looking at the scatter plot, you might observe:

- **Clustering:** Penguins of the same species tend to cluster together. This suggests that bill dimensions can be used to distinguish between species.
- **Overlapping:** There is some overlap between the species, especially in the middle. This means that relying solely on bill length and depth might not be enough to accurately classify all penguins. You might need to consider other features, such as flipper length or body mass.
- **Outliers:** There may be a few outliers – penguins that fall far from the main clusters. These outliers could be due to measurement errors, misidentification, or simply represent unusual individuals. Further investigation might be needed to understand these outliers.

By visualizing the data, you gain valuable insights that can inform your research and decision-making.

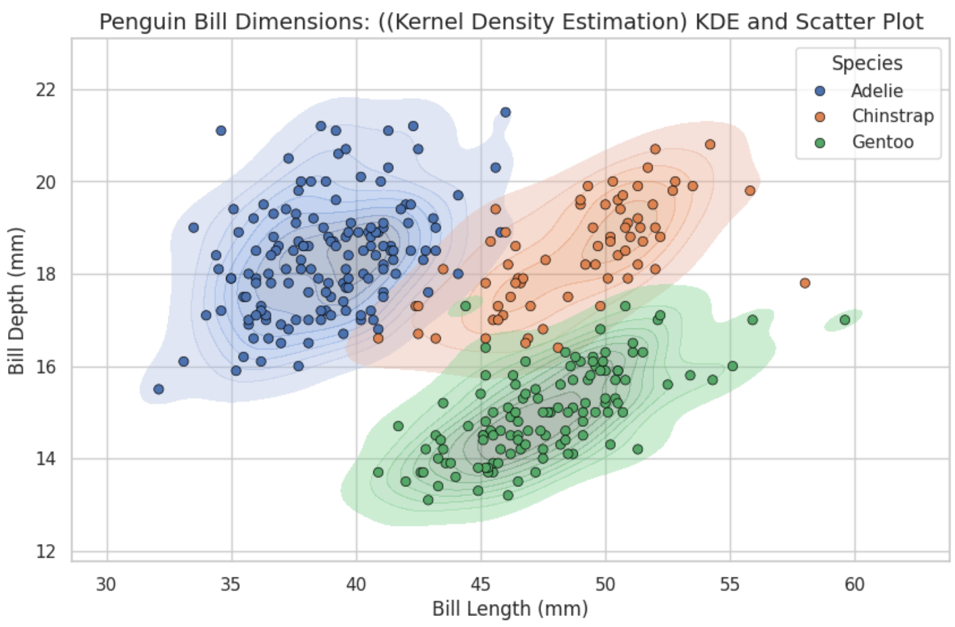

Figure 4: A scatter plot of penguin bill length vs. bill depth, with KDE overlay

5 Regression Plots: Spotting Trends

Let's explore trend analysis with regression plots. Let's use the example of customer age versus total spending.

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

# Generate sample customer sales data
np.random.seed(42)
data = pd.DataFrame({
    "Customer_Age": np.random.randint(18, 65, 200),
    "Total_Spend": np.random.uniform(100, 5000, 200)
})


plt.figure(figsize=(10, 6))

# Scatter plot with trend line
sns.regplot(data=data, x="Customer_Age", y="Total_Spend", scatter_kws={'alpha'
    :0.6}, line_kws={'color':'red'})

plt.title("Customer Age vs Total Spending (with Trend Line)", fontsize=14)
plt.xlabel("Customer Age", fontsize=12)
plt.ylabel("Total Spend (in $)", fontsize=12)

plt.show()
```

Listing 3: Creating a Regression Plot

In the example plot generated, The trend line is flat, suggesting that there is no correlation between customer age and the total amount spent.

6 Bar Plots: Payment preference

Let's explore analyzing payment methods in the customer payment preferences example:

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

# Generate sample payment method data
np.random.seed(42)
payment_methods = ["Credit Card", "Debit Card", "UPI", "Cash", "Net Banking", "
    Wallet"]
transactions = np.random.randint(100, 1000, len(payment_methods))

data = pd.DataFrame({
    "Payment_Method": payment_methods,
    "Number_of_Transactions": transactions
})

plt.figure(figsize=(10, 6))

# Fix: Assign `hue` to the x variable and disable the legend
sns.barplot(data=data, x="Payment_Method", y="Number_of_Transactions",
            hue="Payment_Method", palette="coolwarm", legend=False)

# Customize plot
plt.title("Customer Payment Preferences", fontsize=14)
plt.xlabel("Payment Method", fontsize=12)
plt.ylabel("Number of Transactions", fontsize=12)

# Rotate x-axis labels for readability
plt.xticks(rotation=20)

# Show plot
plt.show()
```

Listing 4: Creating a Bar Plot for Payment Preferences

Strategic Decision-Making:
When opening a market you should consider the top three payment modes. From the dataset provided and the plot, UPI, Debit Card and Cash emerge as the top three payments.

7 Pair Plots: The Big Picture

Now, let's unleash the power of the pair plot. This plot is your go-to tool when you want to explore the relationships between multiple numerical variables in your dataset. It creates a matrix of scatter plots, showing the relationship between each pair of variables.

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

# Generate sample employee performance data
np.random.seed(42)
num_employees = 100

data = pd.DataFrame({
    "Years_of_Experience": np.random.randint(1, 20, num_employees),
    "Salary": np.random.randint(30, 150, num_employees),  # Salary in $1000s
    "Projects_Completed": np.random.randint(1, 50, num_employees),
    "Work_Hours_per_Week": np.random.randint(30, 60, num_employees)
})


# Pair plot to visualize relationships between numerical variables
sns.pairplot(data, diag_kind="kde")

# Show plot
plt.show()
```

Listing 5: Creating a Pair Plot

In pair plots, all relationships are scattered in a plot. The data is scattered across the plot, and insights can be gathered from the plots. In the diagonal elements, you are free to pick the kind of data you want. You can go for KDE, histogram, based on what you want to see.

8 Swarm Plots

To look at the salary distributions across the department, swarm plots could be really helpful:

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

# Set seed for reproducibility
np.random.seed(42)
num_employees = 150

# Define departments
departments = ["HR", "Engineering", "Marketing", "Sales", "Finance"]

# Generate synthetic salary data
data = pd.DataFrame({
    "Department": np.random.choice(departments, num_employees),
    "Salary": np.concatenate([
        np.random.randint(40, 80, 30),    # HR salaries
        np.random.randint(70, 150, 40),   # Engineering salaries
        np.random.randint(50, 100, 30),   # Marketing salaries
        np.random.randint(45, 90, 25),    # Sales salaries
        np.random.randint(60, 120, 25)    # Finance salaries
    ])
})

# Create the swarm plot with the corrected syntax
plt.figure(figsize=(10, 6))
sns.swarmplot(data=data, x="Department", y="Salary", hue="Department",
              palette="coolwarm", legend=False)

# Add labels and title
plt.xlabel("Department")
plt.ylabel("Salary ($1000s)")
plt.title("Swarm Plot of Salaries by Department")

# Show the plot
plt.show()
```

Listing 6: Creating a Swarm Plot for Salary Distribution

9 Beyond the Basics: Customization

Seaborn offers endless possibilities for customization. You can control colors, fonts, labels, titles, and much more to create visualizations that are both informative and visually appealing. With custom charts, you can leverage different customizations. The customization charts code is available in the shared collab link.

10 Visualizing Your Way to Success

Seaborn is more than just a plotting library. It's a powerful tool for exploring data, uncovering hidden patterns, and making strategic decisions. By mastering Seaborn, you'll be able to transform raw data into actionable insights and communicate your findings effectively. So, dive in, experiment, and unleash the power of visualization!