Data Visualization Techniques - Bar and Box Plot

WE'LL COVER THE FOLLOWING

Visualization Techniques

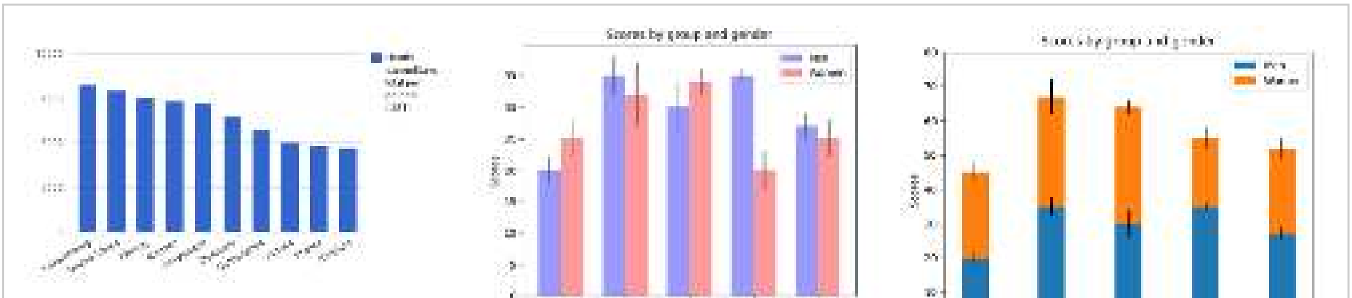
4. Bar Plots

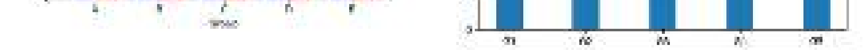
Bar plots are an effective visualization technique when we have categorical data that consists of various categories or groups.

For example, we can use bar plots to view test scores categorized by gender. These plots allow us to easily see the difference between categories (gender) because the size of the bar varies with the magnitude of the represented variable (score), and categories are clearly divided and color-coded.

However, there is a catch. Bar plots work well when we are dealing with a few categories. If there are too many categories though, the bars can get very cluttered and the plots can quickly become hard to understand.

We can have several variations of bar plots. The image below shows three different types of bar plots: regular, grouped, and stacked:





> *Notice that bar plots look very similar to histograms. How do we distinguish between the two? What are the differences between the two?*
>
> *A histogram represents the frequency distribution of continuous variables while a bar graph is a diagrammatic comparison of discrete variables. The histogram presents numerical data whereas the bar graph shows categorical data. The histogram is drawn in such a way that there is no gap between the bars.*

Say we have data in the form of lists for the number of users for different programming languages:

```
# Input data
labels = ('Python', 'C++', 'Java', 'Perl', 'C#')
num_users = [12,8,15,4,6]
```

To make a bar plot from this data, we first need to convert the labels from string data into numerical data that be used for plotting purposes. We can use NumPy's *arange* method to generate an array of sequential numbers of the same length as the *labels* array like this:

```
index = np.arange(len(label))
# index: [0,1,2,3,4]
```

Now we can easily represent languages on the *x-axis* and *num_users* on the *y-axis* using the `plt.bar()` method:

```
import numpy as np
import matplotlib.pyplot as plt

# Input data
labels = ('Python', 'C++', 'Java', 'Perl', 'C#')
num_users = [12,8,15,4,6]
index = np.arange(len(labels))


# Plotting
plt.bar(index, num_users, align='center', alpha=0.5)
plt.xticks(index, labels)
plt.xlabel('Language')
plt.ylabel('Num of Users')
plt.title('Programming language usage')

plt.savefig('output/barplot.png')
```

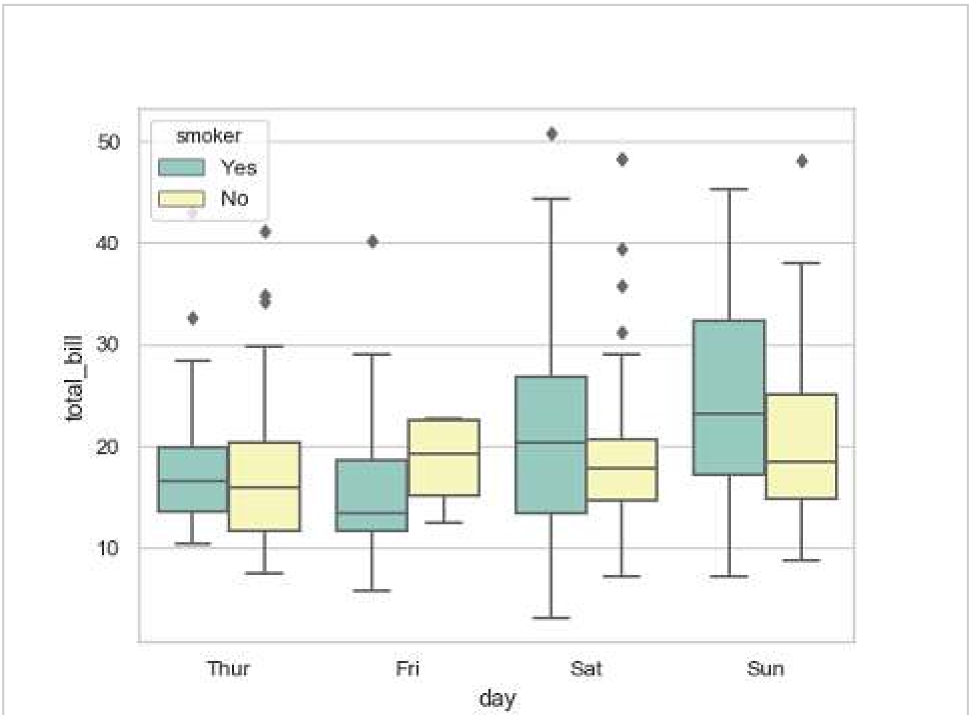
5. Box Plots

We previously looked at histograms which were great for visualizing the distribution of variables. But what if we need more information than that? Box plots provide us a statistical summary of the data. They allow us to answer questions like:

- Is the data skewed, when all the values are concentrated on one side?
- Is the median different from the mean?
- What does the standard deviation look like?

> ***Note: If all of these terms sound alien to you at the moment, don't worry! We will cover these terms, and get back to box plots in more detail, in the Statistics Lessons.***

In a box plot, the bottom and top of the box are always the 1st and 3rd quartiles, 25% and 75% of the data, and the band inside the box is always the 2nd quartile, the median. The dashed lines with the horizontal bars on the end, or whiskers, extend from the box to show the range of the data:



Example of Box plot Analysis

We can use the `boxplot()` method to create our insight-rich box plots:

```
plt.boxplot(data_to_plot)
```

This method also takes various parameters as input to customize the color and appearance of the boxes, whiskers, caps, and median. Let's see an example based on some randomly generated data:

```
# numpy is used for creating fake data
import numpy as np
import matplotlib.pyplot as plt

# Create data
data_group1 = np.random.normal(100, 10, 200)
data_group2 = np.random.normal(80, 30, 200)
data_group3 = np.random.normal(90, 20, 200)

# combine these different datasets into a list
data_to_plot = [data_group1, data_group2, data_group3]

# Create the boxplot
# patch_artist must be True to control box fill (color-filled boxes)
plt.boxplot(data_to_plot, patch_artist = True)

plt.xlabel('Group')
plt.ylabel('Scores')
plt.title('Some Group Scores')

# Save the figure
plt.savefig('output/boxplot.png')
```

Final Thoughts

These were examples of some *must-know* Data Visualization techniques using Python's most popular visualization library, Matplotlib. These are ***simple yet powerful visualization techniques which you can use them to extract rich insights from your datasets.***

Of course, the visualization story doesn't end here: not only are there other more elaborate visualizations to extract even deeper information from your data, like Heat Maps and Density Plots, but there are many ***other useful libraries*** for cool visualizations too. Here is a brief introduction of the most popular ones:

popular ones:

- Pandas Visualization: easy to use interface, built on Matplotlib
- Seaborn: high-level interface, based on Matplotlib, for attractive and informative plots
- ggplot: based on R's ggplot2
- Plotly: can create interactive plots
- Bokeh: interactive visualization library for beautiful visual presentation of data in web browsers

Again, data science is an extensive field, so you cannot, and should not, expect to learn and remember all the possible libraries, methods, and their details!

We will get back to creating more rich visualizations, **in the Projects Section**. We will also work with more libraries, like *seaborn*, and build upon what we have learned so far. **Stay Tuned!** 👊