Assignment 2: Understanding Data Distributions Through Visualization (100 points)

In this assignment, you'll explore different ways to visualize the same dataset. Once again, you'll learn why different visualization types are useful and how they complement each other in understanding data distributions.

```
# read data from BoxPlots.tsv into a dataframe that we'll call df
import pandas as pd
import matplotlib.pyplot as plt

raise NotImplementedError("Please write code to load and parse BoxPlots.csv into a df")
df = pd.read_csv()#### modify and elaborate on this code
```

Part 1: Basic Box Plot (15 points)

First, let's create a simple box plot to understand the basic distribution of our data.

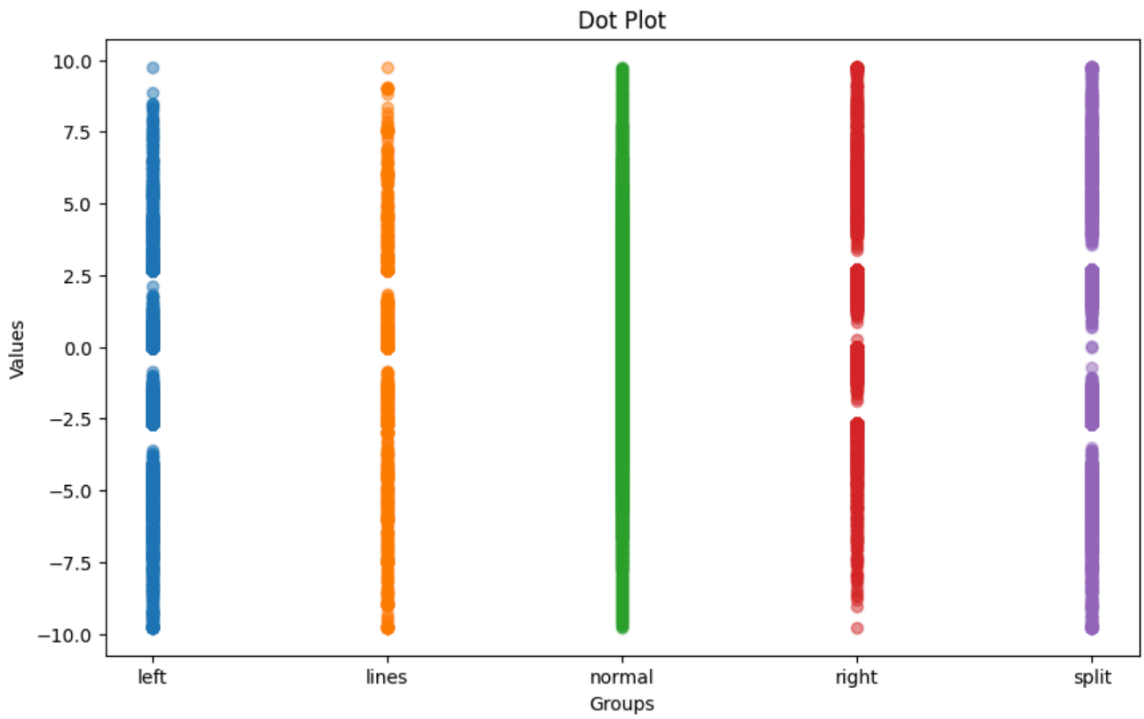
```
# Create your first boxplot here
# YOUR CODE HERE
raise NotImplementedError()

"""
Expected steps:
1. Create a figure with appropriate size
2. Create a single figure using plt.boxplot() or df.boxplot(),
HINT: it should plot all the columns in the dataframe and they should be identical!
3. Add proper labels and title
4. Show the plot
"""
```

Part 2a: Dot Plot - Raw Data Points (10 points)

Box plots are really useful for seeing visualizing the interquartile range of data, but they can hide the actual distribution of data. Let's take a deeper look at the data by creating a dot plot to see every data point.

Adapt what you know about scatter plots to create a dot plot. Instead of letting the x-axis vary with data values, assign a fixed x-coordinate for each categorical group. Use the data for each group to determine the y-coordinates. Think about how to use loops to handle multiple groups efficiently.

this might look like something like this:

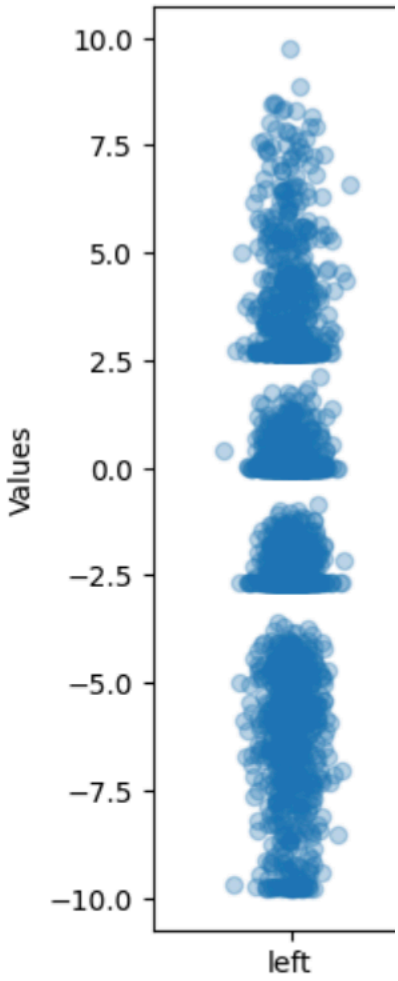



```
# Create a strip plot (dots aligned vertically for each group)
# YOUR CODE HERE
raise NotImplementedError()

"""
Expected output:
- use the scatterplot capability of matplotlib
- One column of points for each group - think of how to generate an x-position based on category
- Each point represents one data value
- Points should be small and slightly transparent
"""
```

Part 2b: Jittered Dot Plot (15 points)

The Dot plot started to reveal important differences between the dataset that were not apparent in the boxplot. We can gain even more insight by adding a 'jitter', or small random element of noise to the dotplot code that we made in the prior section.

Aim for something like the following




```
# Create a jittered strip plot
# Hint: Use np.random.normal() to create jitter in the x-dimension within each category
# Reflection: why do we only use jitter in x and not in y as well?
# YOUR CODE HERE
raise NotImplementedError()
```

Part 3: Explore the Distribution Shape (15 points)

By now we have demonstrated that, just as in Assignment 1, there are significant differences between these groups despite the fact that each group has very similar summary stats. Let's gain further insight by visualizing the actual shape of the data distribution by creating a histogram. Use it to understand how the values are spread across different ranges.

```
# Create subplots with a histogram for each group
# YOUR CODE HERE
raise NotImplementedError()

"""
Expected steps:
1. Create a subplot grid appropriate for your number of group
2. For each group:
   - Create a histogram
```

```
   - Add labels
   - Set consistent bins across all plots
"""
```

Part 4: Kernel Density Estimation (20 points)

Histograms are great for understanding the overall shape of a distribution, but they can be influenced by choices like the number of bins or bin edges. To get a smoother and more continuous view of the data's distribution, we can use Kernel Density Estimation (KDE). KDE estimates the underlying probability density function of the data, giving us a clearer picture of how values are distributed without relying on fixed bins.

In the code cell below create a KDE plot for each group. Compare the KDE curves to the histograms from Part 3. What does KDE reveal about the data that might have been harder to see in the histograms?

```
# Create KDE plots for each group
# Hint: Use scipy.stats.gaussian_kde
# YOUR CODE HERE
raise NotImplementedError()
```

Reflection (Ungraded): Compare the KDE curves to the histograms from Part 3. What does KDE reveal about the data that might have been harder to see in the histograms?

Part 5 Violin Plots (20 points)

So far, we've explored the distribution boxplots, dot, plots, histograms and KDE, but comparing multiple groups side by side can still be tricky. This is where violin plots come in! Violin plots combine the strengths of KDE with the easy side-by-side group comparison in box or dot plots; Violin plots work by mirroring the density curve for each category. They also incorporate a marker for the median and other summary statistics, giving a more holistic view of the data.

In the cell below create a violin plot to visualize the distribution of each group. Compare the violin plot to the histograms and KDE plots. How does the violin plot enhance your ability to compare the groups?

```
# Create violin plots
# YOUR CODE HERE
raise NotImplementedError()
```

Reflection (Ungraded): How does the violin plot enhance your ability to compare the groups?

Part 6 Subplots (25 points)

Now let's bring together three of our visualization techniques in a way that allows direct comparison. Create a figure with three subplots (1 row, 3 columns) that shows:

1. A box plot
2. A jittered dot plot
3. A violin plot

Requirements:

- All three plots should share the same y-axis scale
- All plots should have consistent styling (colors, labels, etc.)
- Each subplot should have an appropriate title