Promising Methods to Handling Numerical Ties in PCPs

Determine the numerical ties in the dataset

Let's determine the number of numerical ties in the dataset.

Attempt 1: Incremental Offset (Sequential Tie Handling)

Equation:

$$value'_i = value_i + \Delta \times k$$

Where:

- Δ is a small offset.
- k is the relative position within the group of ties.

Description:

Adds a small incremental offset to each tied value based on its relative position in the tie group. Ensures that ties are visually and numerically distinct while maintaining an order.

Advantages:

- Maintains the tie order naturally.
- Suitable for ordered or sequential data.

Implementation:

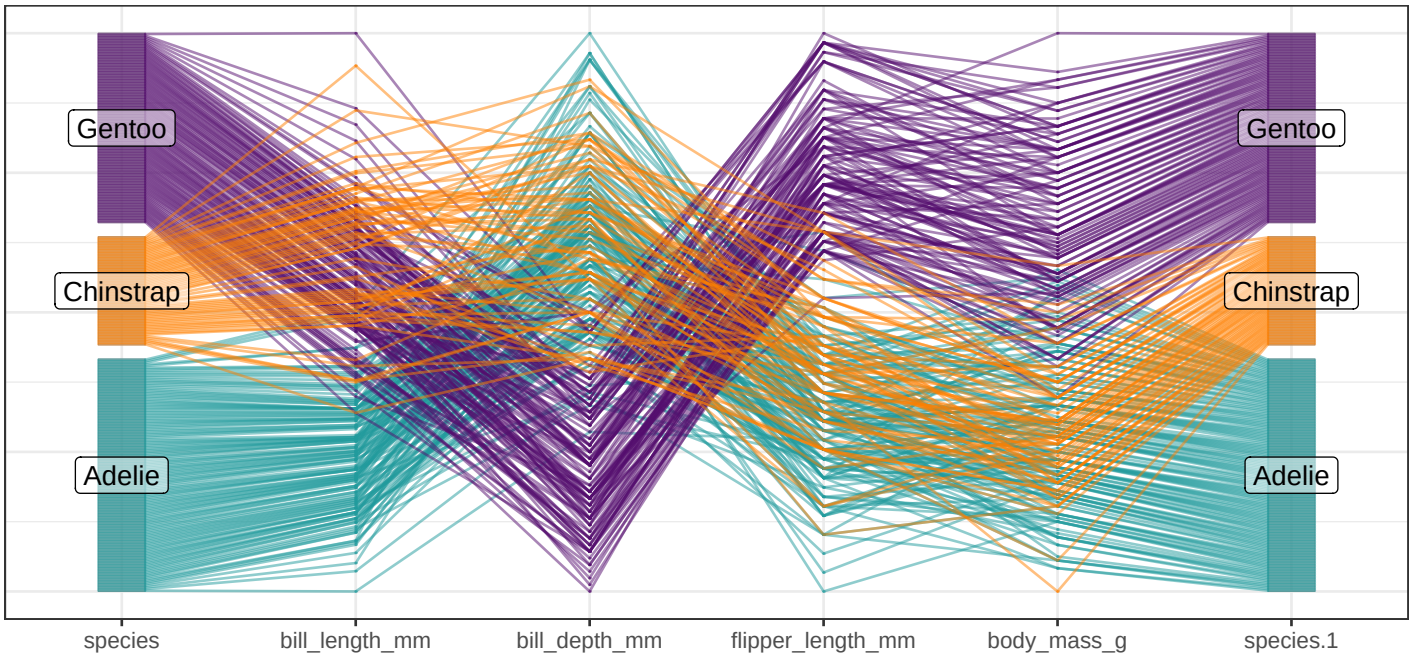
Ideal when you want to preserve some form of sequential relationship among tied values without introducing randomness.

```r
add_space_to_ties <- function(data, tie_spacing = 0.01) {
  # Identify numeric columns
  num_cols <- sapply(data, is.numeric)

  # Adjust numerical ties by adding incremental spacing using
  # Group Averages Over Level Combinations of Factors

  data[num_cols] <- lapply(data[num_cols], function(col) {
    tie_indices <- ave(col, col, FUN = seq_along)
    col + (tie_indices - 1) * tie_spacing
  })

  return(data)
}
```

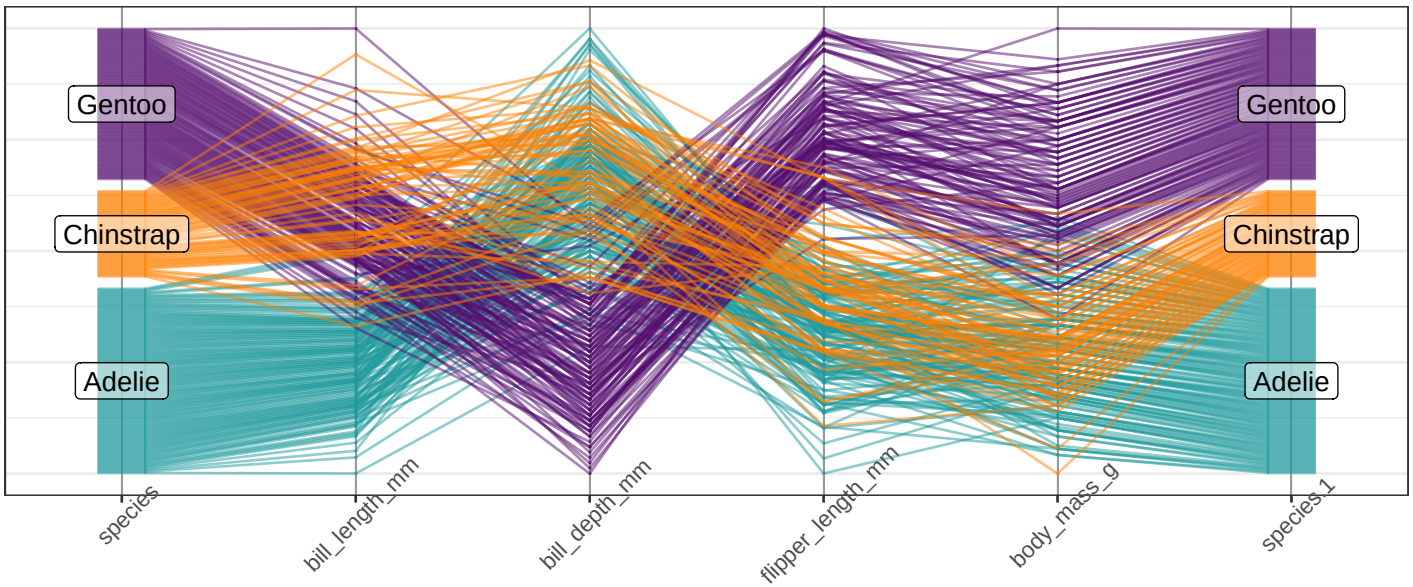
cat_name	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	year
Tie Counts	170.0	254.0	279.0	240.0	330.0
Tie Percentages	51.1	76.3	83.8	72.1	99.1

PCP Implementation

Unadjusted PCP of Palmer Penguins



Incremental Offset PCP of Palmer Penguins



Attempt 2: Jitter Spacing (Numerical Tie Handling)

Equation:

$$value'_i = value_i + \epsilon_i$$

where:

- $value'_i$ is the jittered value of $value_i$,
- $\epsilon_i \sim \mathcal{U}(-\delta, \delta)$, a small random uniform jitter.

Description:

This method breaks numerical ties by adding a tiny random value (ϵ) within a specified range $(-\delta, \delta)$. It ensures that tied values are slightly separated, preserving the overall data structure while improving visualization clarity.

Advantages:

- Simple and intuitive.

- Works well for small datasets with occasional ties.

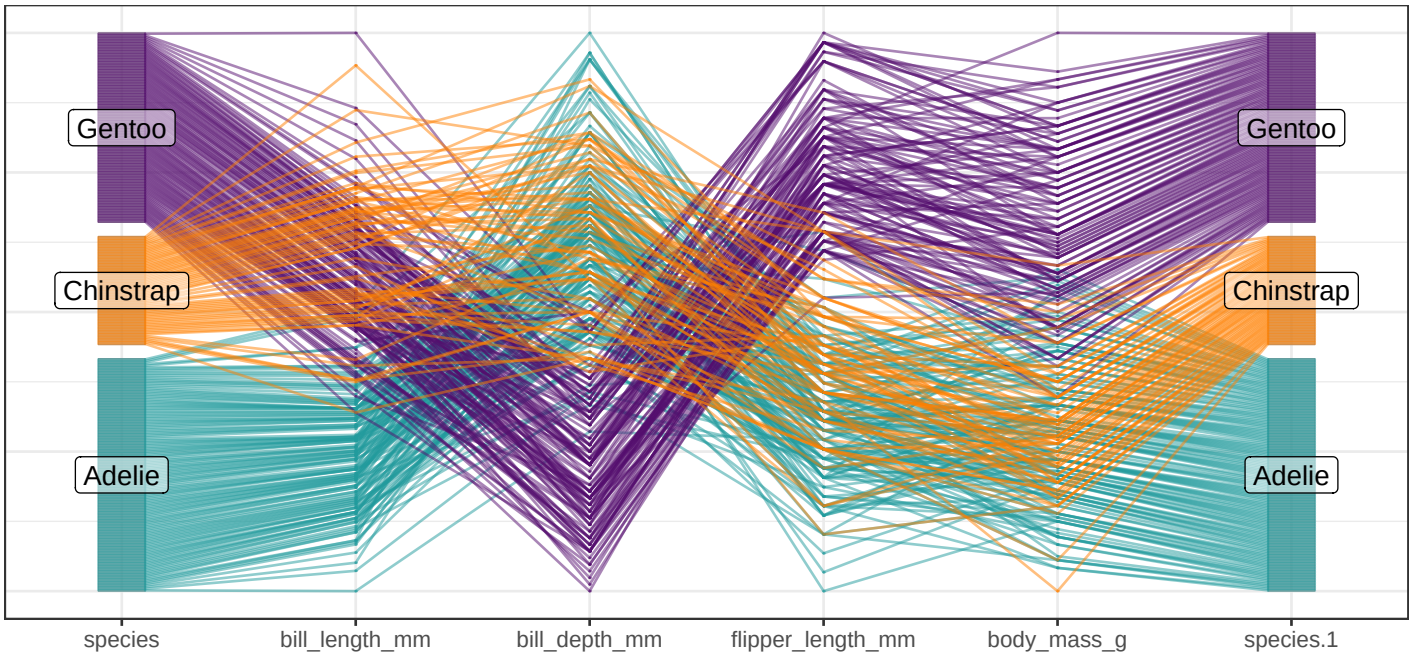
Implementation:

Applied to numeric columns. This is typically done before visualizing parallel coordinate plots or when preparing data for algorithms sensitive to ties.

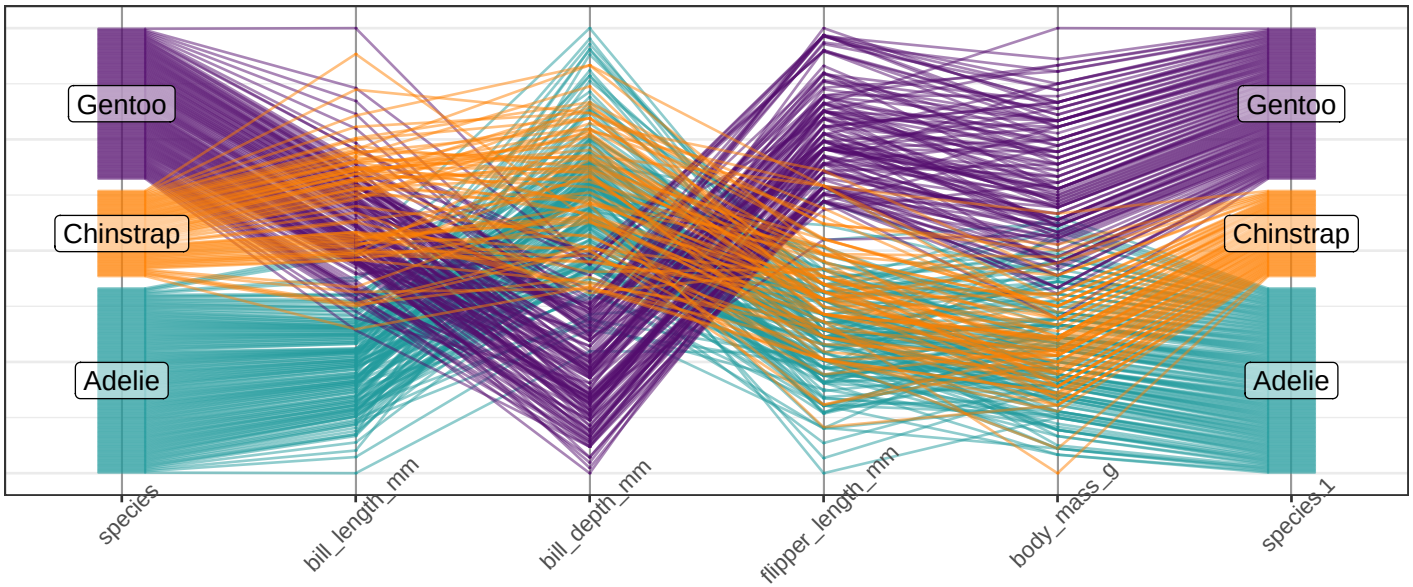
```r
adjust_ties <- function(df, cols_to_adjust, epsilon = 1e-5) {
  df_adjusted <- df

  for (col in cols_to_adjust) {
    if (is.numeric(df[[col]])) {
      ties <- duplicated(df[[col]]) | duplicated(df[[col]], fromLast = TRUE)
      unique_ties <- unique(df[[col]][ties])

      for (tie in unique_ties) {
        tie_indices <- which(df[[col]] == tie)
        adjustment <- seq(-epsilon, epsilon, length.out = length(tie_indices))
        df_adjusted[tie_indices, col] <- df[[col]][tie_indices] + adjustment
      }
    }
  }

  return(df_adjusted)
}
```

PCP Implementation

Unadjusted PCP of Palmer Penguins



Jitter Spacing PCP of Palmer Penguins



Attempt 3: Random Rank Shuffle (Numerical and Categorical Tie Handling)

Equation:

$$Rank_i = rank(Value_i, ties.method = "random")$$

Description:

Tied values are assigned random ranks, ensuring that the order of ties is resolved randomly while maintaining the distribution. For categorical data, this is applied to ordinal values derived from categories, which are then shuffled if tied.

Advantages:

- Guarantees tie resolution without adding noise to the original data.
- Compatible with both numerical and categorical data.

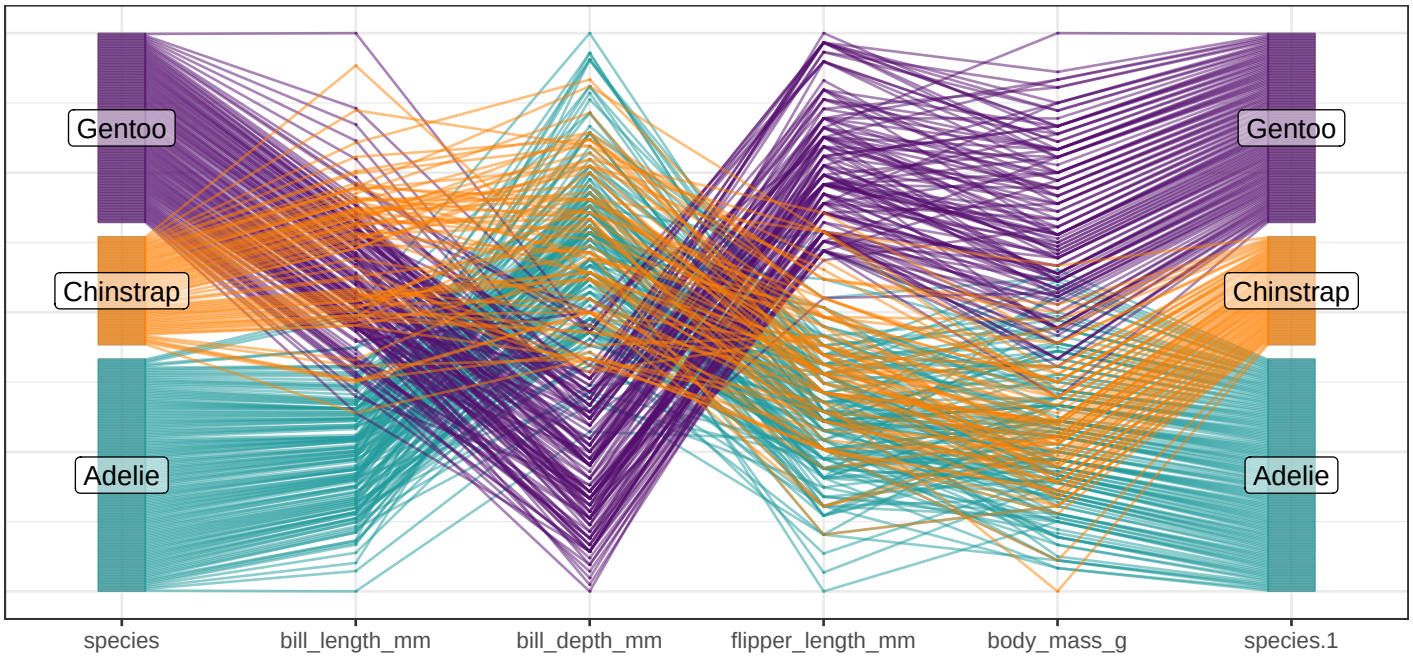
Implementation:

Useful for both numeric and categorical data before plotting or modeling to minimize ambiguity in ties. This method aligns well with the ranking logic used in ggpcp.

```r
apply_tie_breaking <- function(df, cols_to_adjust) {
  df_adjusted <- df

  for (col in cols_to_adjust) {
    if (is.numeric(df[[col]])) {
      # Rank the values with ties.method = "random"
      ranks <- rank(df[[col]], ties.method = "first")
      # Scale ranks back to the original range
      min_val <- min(df[[col]], na.rm = TRUE)
      max_val <- max(df[[col]], na.rm = TRUE)
      df_adjusted[[col]] <- scales::rescale(ranks, to = c(min_val, max_val))
    }
  }

  return(df_adjusted)
}
```

PCP Implementation

Unadjusted PCP of Palmer Penguins



Random Rank Shuffle PCP of Palmer Penguins