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:

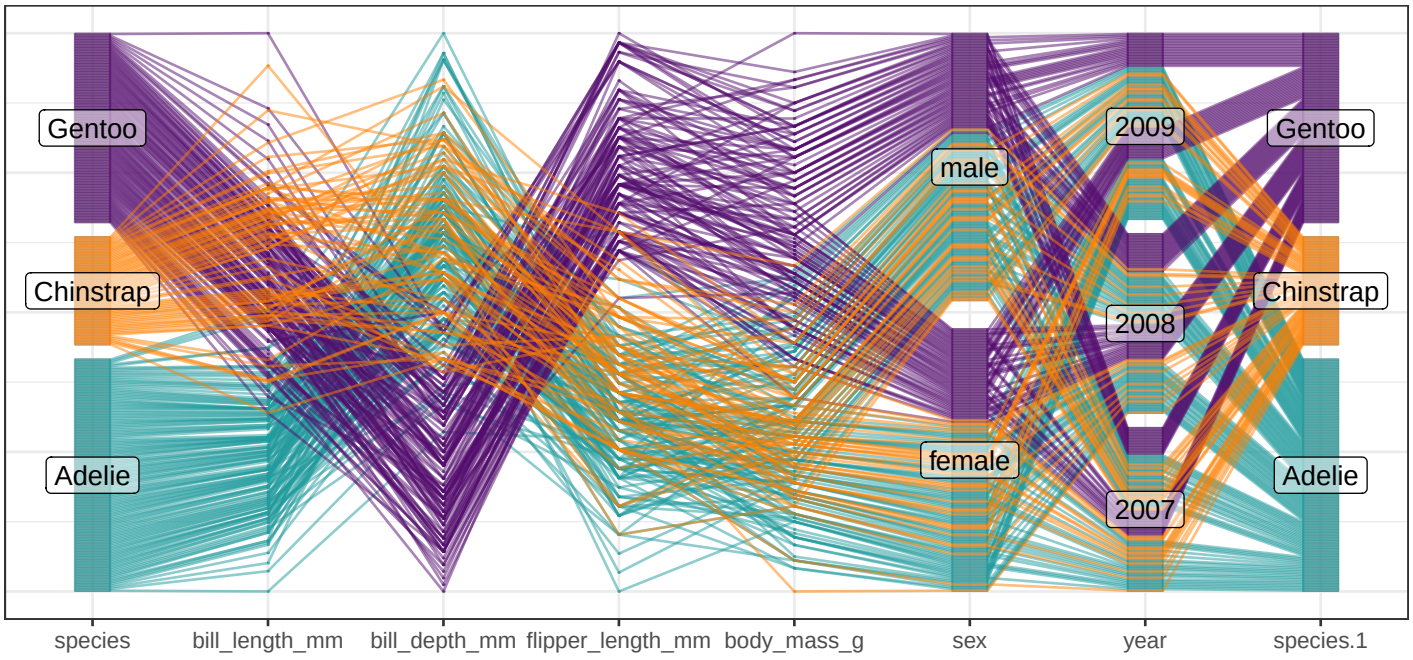
Ideal when you want to preserve some form of sequential relationship among tied values without introducing randomness.

```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)
}
```

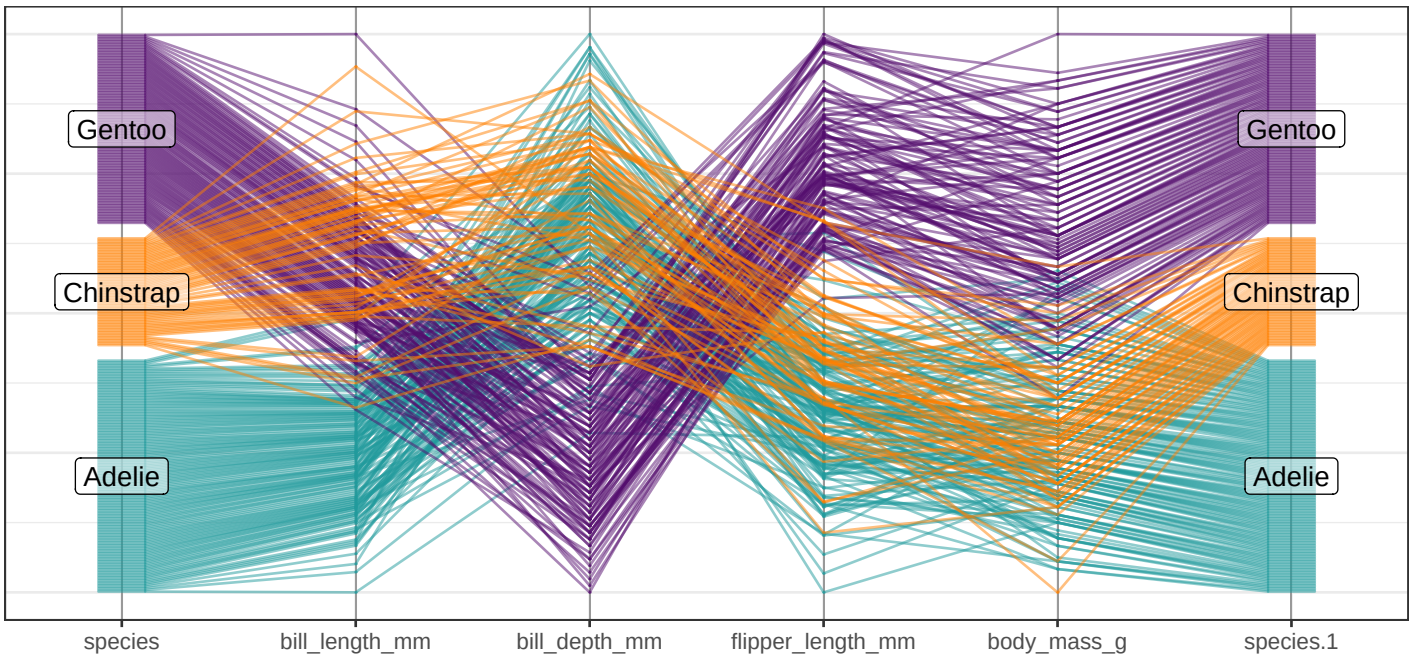
cat_name	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	year
Tie Counts	170.0	254.0	279.0	240.0	330.0
Tie Percentages	51.1	76.3	83.8	72.1	99.1

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.

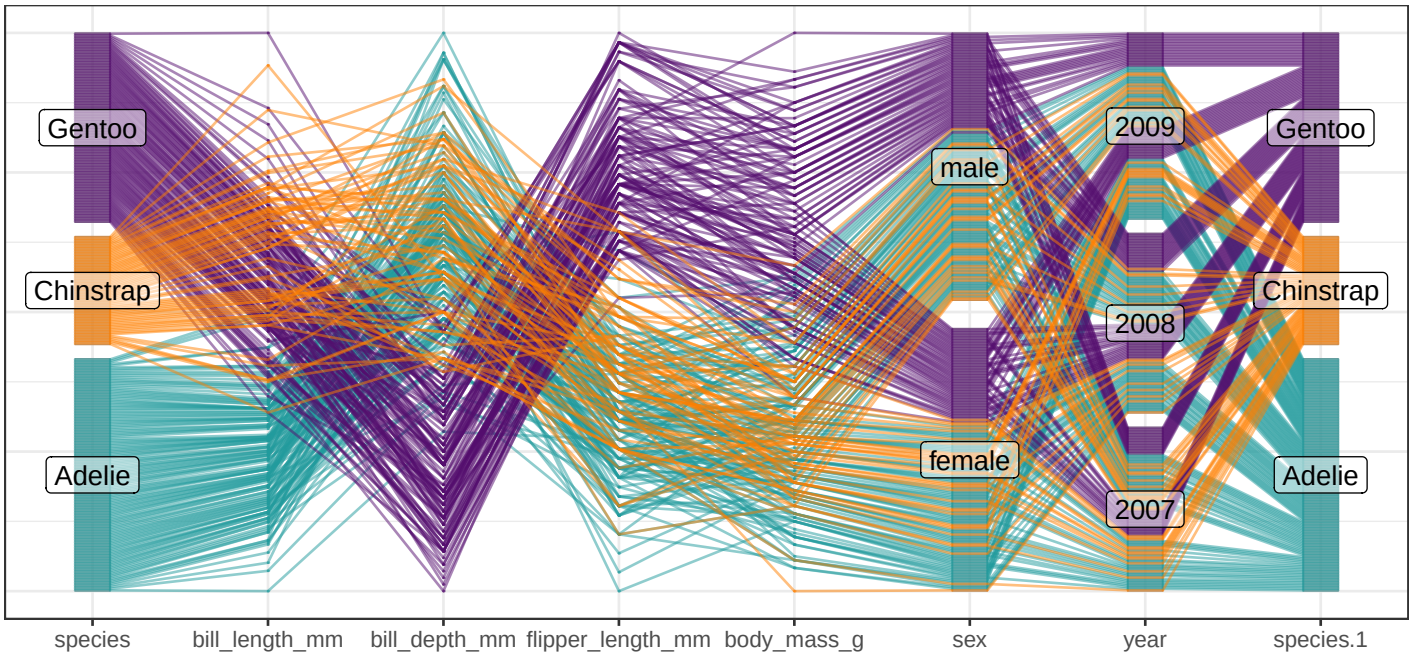
Implementation:

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

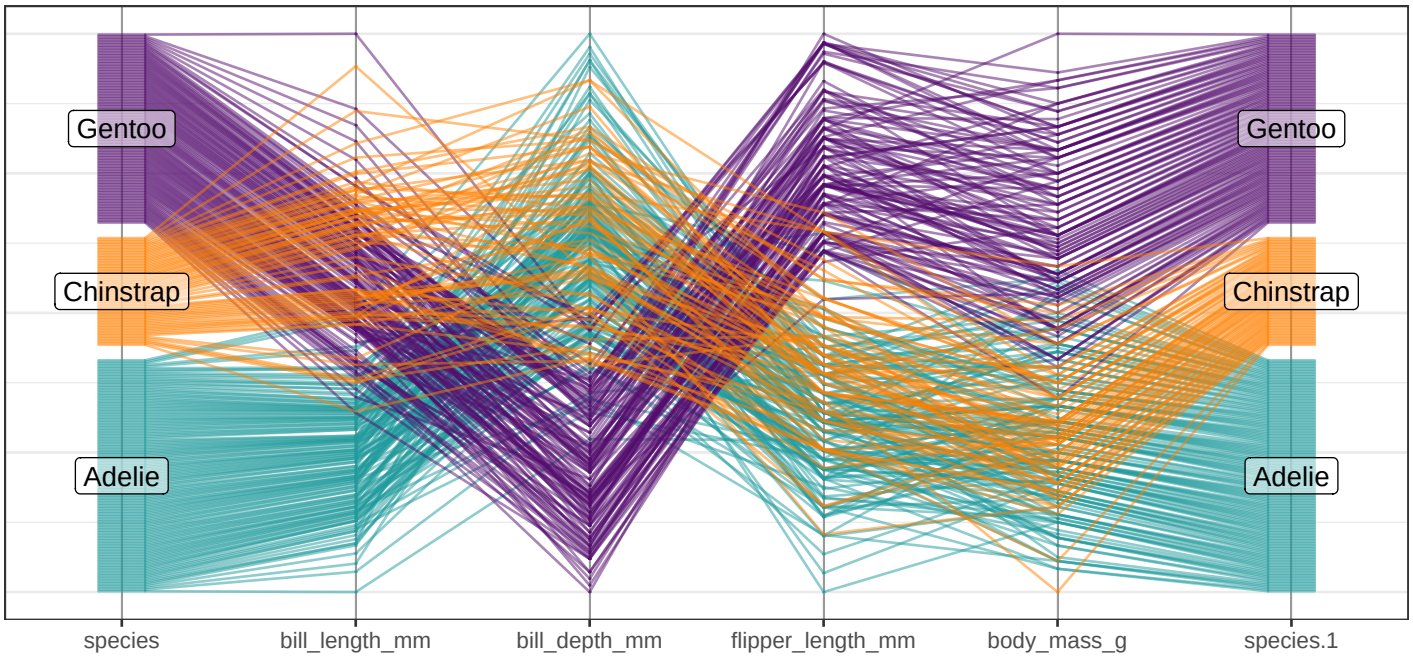
```r
adjust_ties <- function(df, cols_to_adjust, epsilon = 1e-5) {
  df_adjusted <- df

  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.

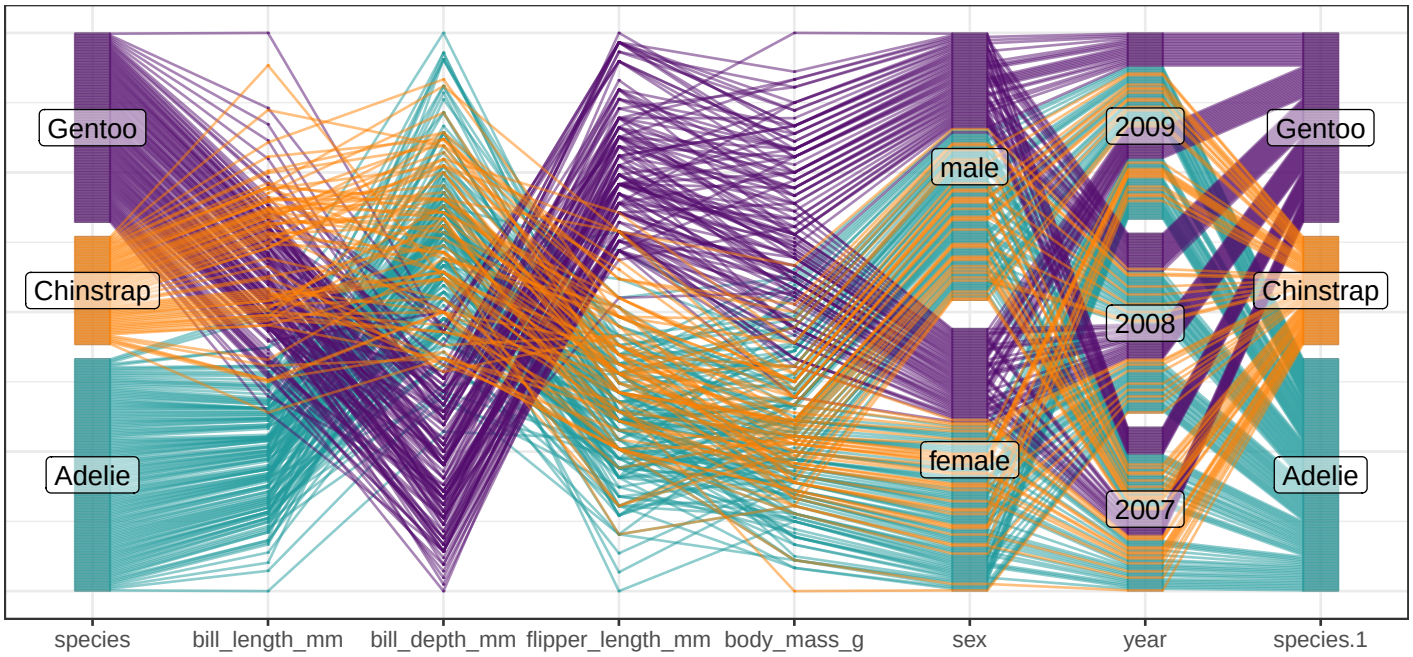
Implementation:

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.

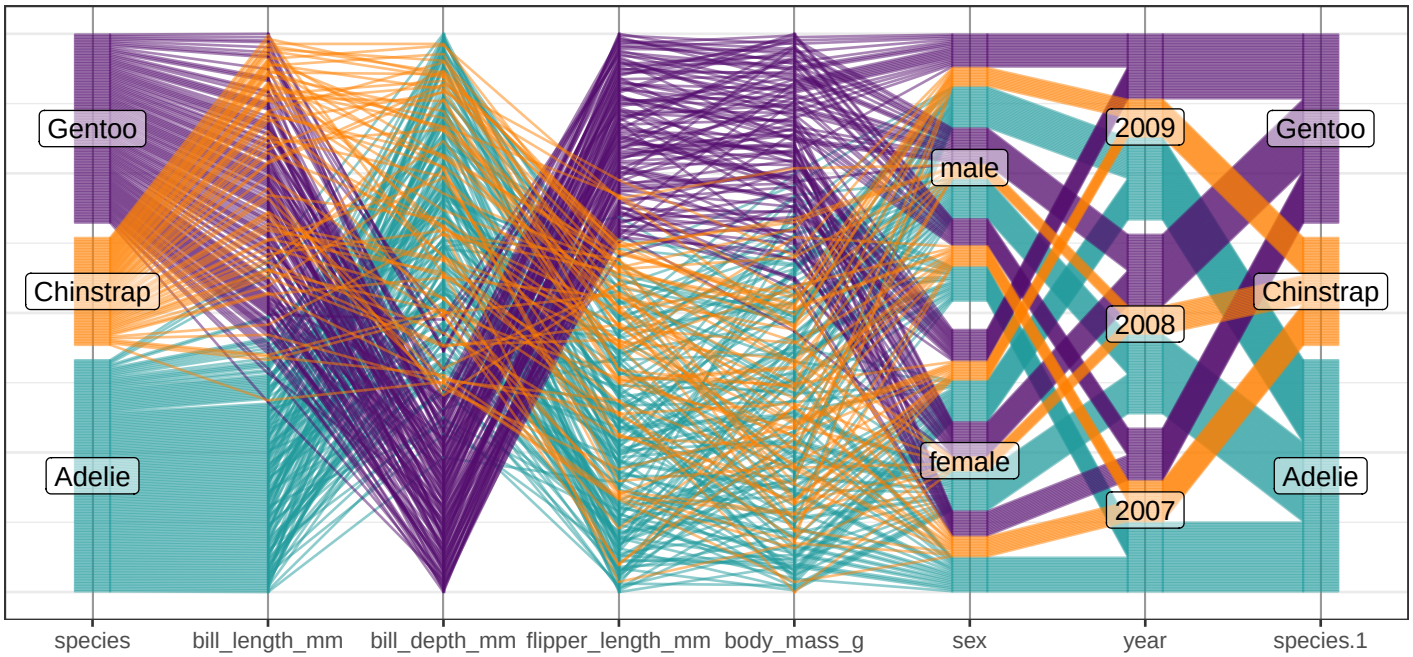
```r
apply_tie_breaking <- function(df, cols_to_adjust) {
  df_adjusted <- df

  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



Attempt 4: "Cascade Rank Tie-Breaker" Method

Key Equations:

1. Initial Rank Assignment:

Assign initial ranks to each variable, handling ties using a user-specified method.

$$R_{i,j} = \begin{cases} rank(x_{i,j}, \text{ties.method}) & \text{if } x_{i,j} \text{ is numeric} \\ as.numeric(foctor(x_{i,j})) & \text{if } x_{i,j} \text{ is categorical} \end{cases}$$

2. Sequential Rank Adjustment Based on Previous Variables:

Refine ranks by incorporating information from preceding variables to break ties and maintain a hierarchical influence.

$$NormPrev_{i,j-1} = \begin{cases} \frac{R_{i,j-1} - min(R_{i,j-1})}{max(R_{i,j-1}) - min(R_{i,j-1})} & \text{if method = "range"} \\ \frac{R_{i,j-1} - min(R_{i,j-1})}{\sigma(R_{i,j-1})} & \text{if method = "sd"} \end{cases}$$

3. Adjusted Current Rank:

Incorporate the normalized previous rank into the current rank:

$$R'_{i,j} = \begin{cases} R_{i,j} & \text{if } j = 1 \\ R'_{i,j} + \epsilon \times NormPrev_{i,j-1} & \text{if } j > 1 \end{cases}$$

- ϵ: A small constant (e.g., 0.01) to control the influence of the previouse rank.

4. Final Data Arrangement for Visualization

Combine the original data with adjusted ranks and arrange the dataset to reflect the unique ordering derived from the "Cascade Rank-Tie Breaker" method.

$$Ranked\ data = original\ data \cup R'\ and\ sorted\ by\ R'_1, R'_2, ..., R'_n$$

Description:

The "Cascade Rank-Tie Breaker" method systematically assigns ranks to each variable in a dataset, ensuring that ties are resolved by leveraging information from previous variables. This hierarchical approach maintains the integrity and relative importance of variables, making it particularly useful for data visualization techniques like Parallel Coordinate Plots.

Advantages:

- **Hierarchical Influence**: Earlier variables in the dataset have a stronger impact on the ranking, preserving the logical flow and importance of variables.
- **Flexible Tie Handling**: Users can specify different ties.method options (e.g., "min", "average") to control how ties are resolved during the initial ranking.
- **Scalability**: Efficiently handles multiple variables and large datasets by sequentially adjusting ranks. Enhanced Visualization: Facilitates clearer and more informative multivariate visualizations by ensuring a unique and meaningful ordering of data points.
- **Customization**: Allows users to choose between different normalization methods ("range", "sd") and to define custom normalization functions, tailoring the method to specific analytical needs.

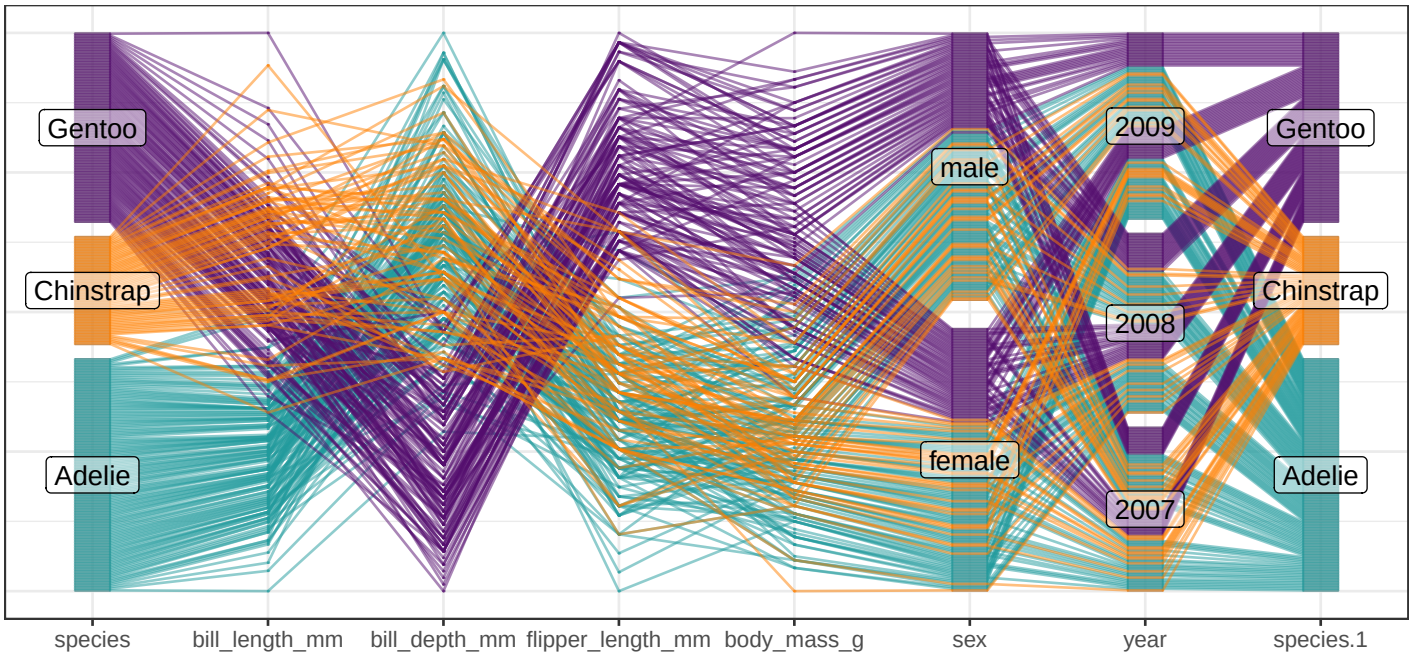
Implementation:

```r
cascade_tie_break <- function(data, method = c("range", "sd"), ties_method = "min") {
  # Match the 'method' argument to ensure it's one of the allowed options
  method <- match.arg(method)

  # Validate the 'ties_method' argument to ensure it's a valid option for the rank function
  valid_ties_methods <- c("average", "first", "random", "max", "min")
  if (!ties_method %in% valid_ties_methods) {
    stop(paste("Invalid ties_method. Choose one of:", paste(valid_ties_methods, collapse = ", ")))
  }

  # Initialize a dataframe to store ranks, excluding any existing 'rank' columns
  rank_df <- data.frame(id = 1:nrow(data))

  # Iterate over each column to assign ranks
  for (i in seq_along(data)) {
    var <- data[[i]]

    # Determine if the variable is numeric or categorical
    if (is.numeric(var)) {
      # Apply the specified ties.method in the rank function
      current_rank <- rank(var, ties.method = ties_method)
```

```
    } else {
      # For categorical variables, convert factors to numeric ranks based on their levels
      current_rank <- as.numeric(as.factor(var))
    }

    # If not the first variable, adjust ranks based on the previous variable
    if (i > 1) {
      prev_rank <- rank_df[[i - 1]]

      # Normalize the previous rank based on the selected method
      if (method == "range") {
        norm_prev_rank <- (prev_rank - min(prev_rank)) / (max(prev_rank) - min(prev_rank))
      } else if (method == "sd") {
        norm_prev_rank <- (prev_rank - min(prev_rank)) / sd(prev_rank)
      }

      # Adjust the current rank by adding a small fraction of the normalized previous rank
      # This helps in breaking ties based on the previous variable
      current_rank <- current_rank + norm_prev_rank * 0.01
    }

    # Assign the adjusted rank to the rank dataframe
    rank_df[[i]] <- current_rank
  }

  # Combine the original data with the rank dataframe, excluding the 'id' column
  ranked_data <- bind_cols(data, rank_df[ , -1])

  # Arrange the data based on the cumulative adjusted ranks to establish a unique ordering
  ranked_data <- ranked_data %>%
    arrange(across(starts_with("rank")))

  return(ranked_data)
}
```

PCP Implementation

Unadjusted PCP of Palmer Penguins



Cascadde Ranked PCP of Palmer Penguins