Explaining the update to `pcp_arrange()` to include numerical ties

Motivating Example

In the Figure 1, the left panel (*Iris Data no adjustment*) has many lines with identical or nearly identical numeric values, which are superimposed, causing overplotting. In contrast, the right panel (*Iris Data with adjustment*) has assigned incremental vertical offsets to tied numeric values based on the following equations:

$$y_i' = y + \Delta_i, \quad \Delta_i = \tau \left(i - \frac{k+1}{2} \right)$$

where k is the number of ties and τ is a small positive constant derived from data dispersion (using standard deviation or interquartile range). This yields symmetrical spreading above and below the original value y, thereby minimizing overlap. Furthermore, for categorical (factor) variables, both plots order the observations by pivoting them into a wide format, sorting them by the values of another column, and assigning new position with:

$$y_i' = \frac{i - 0.5}{n} \times max(y),$$

where n is the size of the group.

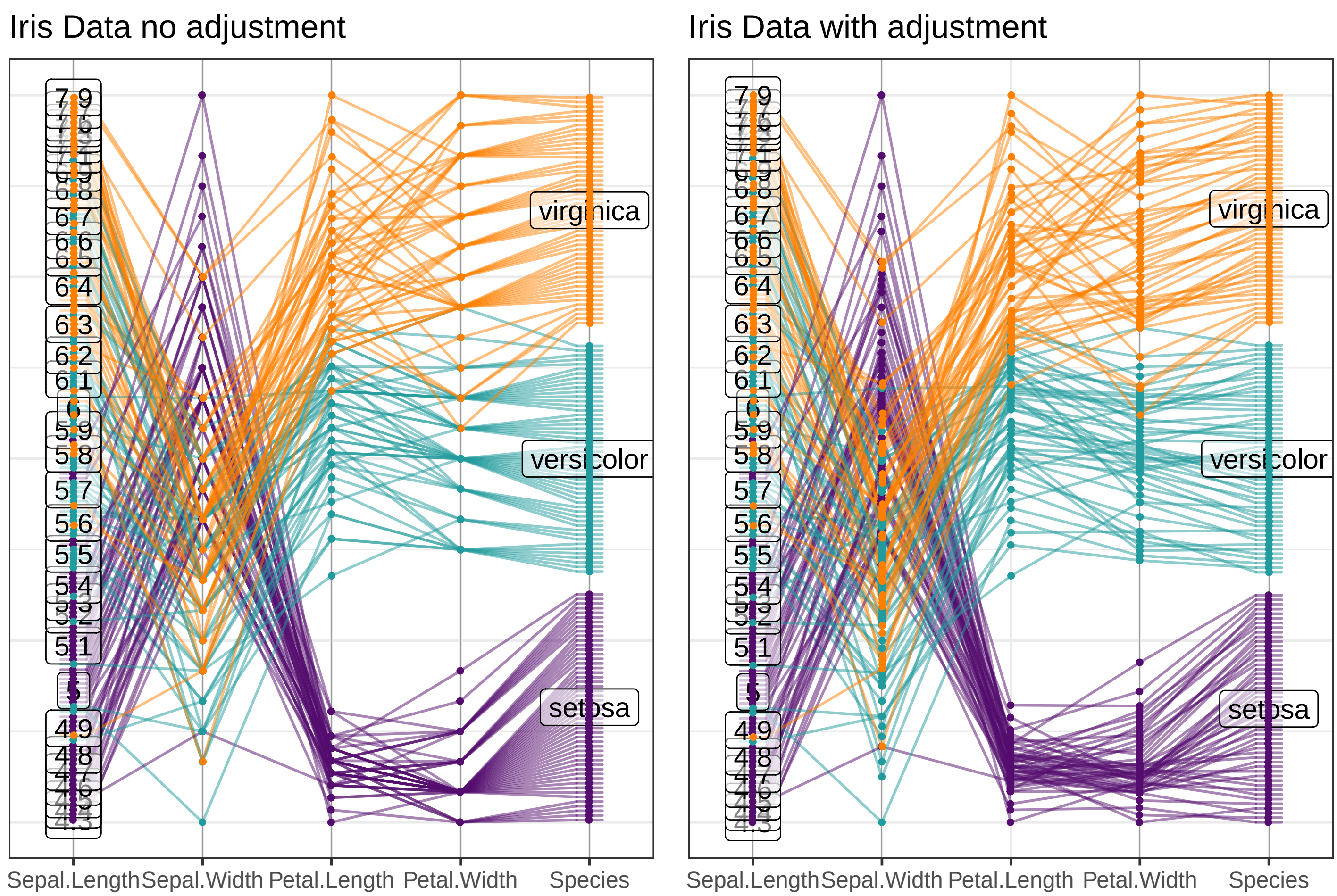

Figure 1

The Latest Version of `pcp_arrange` that will be called `pcp_arrange_with_ties` for Numerical Ties

The enhanced function `pcp_arrange_with_ties` extends the approach to handle both categorical and numerical variables. For numerical ties, it employs helper routines, `numerical_tie_breaker` and `adjusted_numerical_tie_breaker`, which introduce symmetric, statistically informed offsets.

Step 1: Computing the Scaling Factor

The helper function first determines a tie-band τ based on the characteristics of the numeric data:

- **Whole Numbers**: If all values are integers with a small tolerance ϵ

$$|v_i - round(v_i)| < \epsilon \quad \forall i$$

then

$$\tau = \ base\ fraction \quad (default\ 0.01)$$

- **Continuous Numbers**: Otherwise, if the sample is approximately normal (assessed via a Shapiro–Wilk test with $p > 0.05$) and free of outliers, using the bounds

$$L = Q_1 - 1.5 \times IQR, \quad U = Q_3 + 1.5 \times IQR,$$

where $IQR = Q_1 - Q_3$, the scaling factor is set as

$$\tau = \ base\ fraction \ \times \sigma$$

which σ is the standard deviation. If these conditions are not met, the function uses

$$\tau = \ base\ fraction \ \times IQR$$

Step 2: Symmetric Adjustment of Tied Values

Once τ is established, the function sorts the numeric values while preserving their original order (using a secondary key based on observation order). For a block of k tied observations, it assigns offsets symmetrically by computing:

$$\Delta_i = \tau \left(i - \frac{k+1}{2} \right) \ \ for \ \ i = 1, ..., k.$$

Each tied observation is then adjusted as:

$$y'_i = y + \Delta_i$$

this approach ensures that the offsets are distributed both below and above the original value. The first observation receives the most negative adjustment, and the last the most positive, thereby preserving the original order in a manner similar to how `pcp_arrange` treats categorical data.

```r
numerical_tie_breaker <- function(values, base_fraction = 0.01) {
  is_whole_number <- all(abs(values - round(values)) < .Machine$double.eps^0.5)
  if (is_whole_number) {
    tie_band <- base_fraction
  } else {
    is_normal <- FALSE
    if (length(values) >= 3) {
      shapiro_p <- shapiro.test(values)$p.value
      is_normal <- (shapiro_p > 0.05)
    }
    q <- stats::quantile(values, probs = c(0.25, 0.75))
    iqr_value <- q[2] - q[1]
    lower_bound <- q[1] - 1.5 * iqr_value
```

```r
    upper_bound <- q[2] + 1.5 * iqr_value
    has_outliers <- any(values < lower_bound) || any(values > upper_bound)
    if (is_normal && !has_outliers) {
      tie_band <- base_fraction * stats::sd(values)
    } else {
      tie_band <- base_fraction * iqr_value
    }
  }

  # Use a secondary key (original order) to ensure stability.
  sorted_indices <- order(values, seq_along(values))
  sorted_values <- values[sorted_indices]
  adjusted_sorted <- sorted_values

  # Identify consecutive runs of tied values.
  rle_vals <- rle(sorted_values)
  cum_lengths <- cumsum(rle_vals$lengths)
  start_indices <- c(1, head(cum_lengths, -1) + 1)

  # For each tied group, assign symmetric offsets preserving the order.
  for (j in seq_along(rle_vals$lengths)) {
    len <- rle_vals$lengths[j]
    if (len > 1) {
      offsets <- tie_band * (seq_len(len) - (len + 1) / 2)
      indices <- start_indices[j]:cum_lengths[j]
      adjusted_sorted[indices] <- sorted_values[indices] + offsets
    }
  }
}
```

Step 3: Reintegration into the Dataset

The helper routine **`adjusted_numerical_tie_breaker`** applies the above process and then returns values in the original order of observations.

```r
adjusted_numerical_tie_breaker <- function(values, base_fraction = 0.01) {
  idx <- order(values, seq_along(values))
  adjusted_sorted <- numerical_tie_breaker(values[idx], base_fraction)

  result <- numeric(length(values))
  result[idx] <- adjusted_sorted

  result
}
```

In doing so, **`pcp_arrange_with_ties`** provides a unified method that refines the placement of tied numeric values while maintaining consistency with the categorical tie resolution strategy.

```r
pcp_arrange_with_ties <- function(data, method = "from-right", space = 0.05, .by_group = TRUE) {
  pcp_id <- pcp_x <- pcp_y <- pcp_yend <- pcp_class <- NULL
  pcp_right <- pcp_left <- replace_y <- replace_yend <- NULL

  if (.by_group == FALSE) {
    data <- data %>% ungroup()
    }

  groups <- names(attr(data, "groups"))

  if (length(groups) > 0) {
    groups <- setdiff(groups, ".rows")
```

```
}

add_space <- function(subdata, space, replace_var, y_var) {
  subdata %>% mutate(
    "{{ replace_var }}" := (1 - space) * {{ replace_var }} + {{ y_var }} * space
  )
}

from_right <- function(subdata, index_start, index_last, var_y, replace_var) {
  select1 <- setdiff(unique(c(groups, "pcp_id",
                              paste0(".pcp.", vars$pcp_x[index_start:index_last]))), "pcp_x")
  select2 <- setdiff(unique(c(groups, "pcp_id",
                              paste0(".pcp.", vars$pcp_x[index_start]))), "pcp_x")
  vary <- rlang::enquo(var_y)

  subdata <- subdata %>%
    tidyr::pivot_wider(names_from = .data$pcp_x, values_from = {{ var_y }},
                       names_prefix = ".pcp.", names_repair = "unique") %>%
    select(select1) %>%
    arrange(across(-.data$pcp_id)) %>%
    dplyr::select(select2) %>%
    tidyr::pivot_longer(cols = paste0(".pcp.", vars$pcp_x[index_start]),
                        names_to = "pcp_x", values_to = as_label(vary), names_prefix = ".pcp.")

  subdata %>%
    mutate(
    "{{ replace_var }}" := (1:n() - 0.5) / n() * max({{ var_y }}, na.rm = TRUE)
  )
}

vars <- data %>%
  group_by(pcp_x, .add = FALSE) %>%
  summarize(pcp_class = pcp_class[1]) %>%
  mutate(pcp_x = as.character(pcp_x))

factor_targets <- which(vars$pcp_class == "factor")

numvars <- nrow(vars)
selects <- unique(c(groups, "pcp_id", "pcp_x", "pcp_y"))

if (method == "from-right") {
  sapply(rev(factor_targets), function(index_start) {
    index_end <- ifelse(index_start < numvars, numvars, 1)

    block <- data %>%
      filter(pcp_x %in% vars$pcp_x[index_start:index_end]) %>%
      dplyr::select(selects)

    b1 <- from_right(block, index_start, index_end, pcp_y, replace)

    b1 <- add_space(b1, space = space, replace, pcp_y)

    bys <- unique(c(groups, "pcp_id", "pcp_x"))
    selects2 <- unique(c(groups, "pcp_id", "pcp_x", "replace"))

    data <<- data %>%
      left_join(b1 %>% dplyr::select(selects2), by = bys)

    data <<- data %>%
      mutate(
      pcp_y = ifelse(!is.na(replace) & !is.na(pcp_y), replace, pcp_y),
      pcp_x = factor(pcp_x, levels = vars$pcp_x)
```

```
    ) %>%
      dplyr::select(-replace)
  })
}

if (method == "from-left") {
  sapply(factor_targets, function(index_start) {
    index_end <- ifelse(index_start == 1, numvars, 1)

    block <- data %>%
      filter(pcp_x %in% vars$pcp_x[index_start:index_end]) %>%
      dplyr::select(selects)

    b1 <- from_right(block, index_start, index_end, pcp_y, replace)
    b1 <- add_space(b1, space = space, replace, pcp_y)

    bys <- unique(c(groups, "pcp_id", "pcp_x"))

    selects2 <- unique(c(groups, "pcp_id", "pcp_x", "replace"))

    data <<- data %>%
      left_join(b1 %>% dplyr::select(selects2), by = bys)

    data <<- data %>%
      mutate(
      pcp_y = ifelse(!is.na(replace), replace, pcp_y),
      pcp_x = factor(pcp_x, levels = vars$pcp_x)
    ) %>%
      dplyr::select(-replace)
  })
}

if (method == "from-both") {
  right <- data %>%
    dplyr::select(selects, "pcp_yend") %>%
    dplyr::select(-pcp_y) %>%
    tidyr::pivot_wider(names_from = "pcp_x", values_from = "pcp_yend")

  left <- data %>%
    dplyr::select(selects, "pcp_yend") %>%
    dplyr::select(-pcp_yend) %>%
    tidyr::pivot_wider(names_from = "pcp_x", values_from = "pcp_y")

  sapply(factor_targets, function(index_start) {
    index_right <- ifelse(index_start < numvars, index_start + 1, index_start)

    block_right <- data.frame(
      pcp_id = left$pcp_id,
      pcp_yend = right[[vars$pcp_x[index_start]]],
      left %>%
        dplyr::select(vars$pcp_x[index_right:numvars]),
      pcp_y = left %>%
        dplyr::select(vars$pcp_x[index_start])
    )

    block_right <- block_right %>%
      arrange(across(-pcp_id)) %>%
      mutate(replace_yend = (1:n() - 0.5) / n() * max(pcp_yend, na.rm = TRUE)) %>%
      arrange(pcp_id)

    block_right <- add_space(block_right, space = space, replace_yend, pcp_yend)
```

```r
    right[[vars$pcp_x[index_start]]] <<- block_right$replace_yend

    if (index_start == 1) {
      left[[vars$pcp_x[index_start]]] <<- block_right$replace_yend
    } else {
      block_left <- data.frame(
        pcp_id = right$pcp_id,
        pcp_y = left[[vars$pcp_x[index_start]]],
        right %>%
          dplyr::select(vars$pcp_x[(index_start-1):1]) %>%
          dplyr::select(-pcp_id)
      )
      block_left <- block_left %>%
        arrange(across(-pcp_id)) %>%
        mutate(replace_y = (1:n() - 0.5) / n() * max(pcp_y, na.rm = TRUE)) %>%
        arrange(pcp_id)

      block_left <- add_space(block_left, space = space, replace_y, pcp_y)
      left[[vars$pcp_x[index_start]]] <<- block_left$replace_y
    }
  })

  replace_yend <- right %>%
    tidyr::pivot_longer(-pcp_id, names_to = "pcp_x", values_to = "replace_yend")

  replace_y <- left %>%
    tidyr::pivot_longer(-pcp_id, names_to = "pcp_x", values_to = "replace_y")

  data <- data %>%
    left_join(replace_yend %>% dplyr::select("pcp_id", "pcp_x", starts_with("replace_")),
                         by = c("pcp_x", "pcp_id"))

  data <- data %>%
    left_join(replace_y %>% dplyr::select("pcp_id", "pcp_x", starts_with("replace_")),
                         by = c("pcp_x", "pcp_id"))

  if (!is.null(data$replace_y)) {
    data <- data %>%
    mutate(pcp_y = ifelse(!is.na(replace_y), replace_y, pcp_y))
  }

  if (!is.null(data$replace_yend)) {
    data <- data %>%
    mutate(pcp_yend = ifelse(!is.na(replace_yend), replace_yend, pcp_yend))
  }

  data <- data %>%
    dplyr::select(-starts_with("replace_")) %>%
    mutate(pcp_x = factor(pcp_x, levels = vars$pcp_x))
}

if (method == "from-both-keep") {
  sapply(factor_targets, function(index_start) {
    index_left <- ifelse(index_start == 1, index_start, index_start - 1)

    index_right <- ifelse(index_start == numvars, index_start, index_start + 1)

    block <- data %>%
      filter(pcp_x %in% vars$pcp_x[index_start])

    right <- data %>%
      filter(pcp_x %in% vars$pcp_x[index_right]) %>%
```

```r
      ungroup()

    left <- data %>%
      filter(pcp_x %in% vars$pcp_x[index_left]) %>%
      ungroup()

    block <- block %>%
      left_join(
      right %>% dplyr::select(pcp_y, pcp_id) %>% rename(pcp_right = pcp_y),
      by = "pcp_id"
    )

    block <- block %>%
      arrange(pcp_yend, pcp_right) %>%
      mutate(replace_yend = (1:n() - 0.5) / n() * max(pcp_yend, na.rm = TRUE))

    if (index_start == 1) {
      block$replace_y <- block$replace_yend
    } else {
      block <- block %>%
        left_join(
        left %>% select(pcp_yend, pcp_id) %>% rename(pcp_left = pcp_yend),
        by = "pcp_id"
      ) %>%
        arrange(pcp_y, pcp_left) %>%
        mutate(replace_y = (1:n() - 0.5) / n() * max(pcp_y, na.rm = TRUE))
    }

    block <- add_space(block, space = space, replace_y, pcp_y)

    block <- add_space(block, space = space, replace_yend, pcp_y)

    data <- data %>%
      left_join(block %>% dplyr::select("pcp_id", "pcp_x", starts_with("replace_")),
                              by = c("pcp_x", "pcp_id"))

    if (!is.null(data$replace_y)){
      data <- data %>%
        mutate(pcp_y = ifelse(!is.na(replace_y), replace_y, pcp_y))
    }

    if (!is.null(data$replace_yend)){
      data <- data %>%
        mutate(pcp_yend = ifelse(!is.na(replace_yend), replace_yend, pcp_yend))
    }

    data <<- data %>% select(-starts_with("replace_"))
  })
}

if (method != "from-both")
  data$pcp_yend <- data$pcp_y

# Apply numerical tie-breaking to numeric variables
numeric_vars <- vars %>% filter(pcp_class == "numeric") %>% pull(pcp_x)
if (length(numeric_vars) > 0) {
  data <- data %>% group_by(pcp_x) %>%
    mutate(
      pcp_y = if (unique(pcp_x) %in% numeric_vars){
        adjusted_numerical_tie_breaker(pcp_y)
      } else{
        pcp_y
```

```
        },

      pcp_yend = if (unique(pcp_x) %in% numeric_vars){
        adjusted_numerical_tie_breaker(pcp_yend)
      } else{
        pcp_yend
        }
    ) %>%
    ungroup()
  }

  data
}
```