Research 2-3: Adaptive Bin-width Regressogram Fitting and Model Selection using Recursive Residual Minimization

Nahyun Lee

05/24/2025

1. Introduction

In this study, we revisit the crash-test dataset used in Assignment 1 to develop an **adaptive bin-width regressogram** model. Unlike regular regressograms with fixed-width bins, this approach allows variable bin boundaries based on the structure of the data.

Our objective is to build a model that recursively partitions the data to minimize prediction error, quantified by the **Residual Sum of Squares (RSS)**. This adaptive approach is a direct analogue to the recursive partitioning used in **decision trees (e.g., CART)**. Whereas decision trees split nodes to maximize purity, our method splits bins to minimize the **Residual Sum of Squares (RSS)**. This allows the model to grow in stages based on verifiable performance improvements.

The modeling process incorporates: - Recursive binary splits based on RSS minimization, - A stopping rule that avoids overfitting by limiting leaf size, - Model selection via a complexity-penalized risk criterion.

This method is particularly well-suited for datasets where underlying structures are not evenly distributed along the input domain.

2. Methodology

2.1 Recursive Partitioning for Adaptive Regressogram

The dataset is recursively split using a greedy search procedure to minimize the **Residual Sum of Squares (RSS)**. At each step, the algorithm evaluates potential split points between observed `xx` values and selects the one that yields the largest reduction in RSS without backtracking or global optimization.

Key properties of the procedure:

- **Split Candidates**: Only midpoints between sorted `xx` values are considered.
- **Stopping Rules**:
  - A bin is only split if it contains `min_split` observations.
  - The process terminates when all resulting bins contain `max_leaf` points.
- **Model Complexity**: The number of bins `k` increases by 1 with each valid split.

The final output is a sequence of candidate models with increasing bin counts, along with their corresponding RSS values.

2.2 Risk Estimation with Complexity Penalty

To select the optimal model, we define a complexity-penalized risk function inspired by Efron's covariance penalty:

$$\text{Risk}(k) = \frac{\text{RSS}(k)}{n} + \frac{2\hat{\sigma}^2 k \log(k)}{n}$$

- The first term reflects the in-sample prediction error.
- The second term penalizes models with higher complexity (more bins) : $\log(k)$ factor imposes a stronger penalty on model complexity than standard criteria like Mallows' C_p or AIC. This can be particularly effective in preventing overfitting when the number of potential splits is large, as is the case in our recursive procedure.

We estimate the residual variance $\hat{\sigma}^2$ using a regular 20-bin regressogram, which serves as a baseline for penalty calibration.

3. Analysis and Results

3.1. Data Loading and Preparation

We begin by loading the `mct.csv` dataset.

```
# Import dataset
crash_data <- read.csv("/Users/nahyunlee/Desktop/mct.csv")

# Extract x and y columns
x <- crash_data$xx
y <- crash_data$yy
```

3.2 Recursive Partitioning Output

To construct an adaptive bin-width regressogram, we recursively partition the data by minimizing the **Residual Sum of Squares (RSS)** at each step.

The process starts with all data in a single bin and iteratively selects the best split point that leads to the greatest reduction in RSS.
Splits are only made if the resulting bins meet the minimum size criteria.

The output below summarizes the first few splits:

- `bins_k`: Total number of bins after the split (starting from 2).
- `split_location`: The `x`-value where the best split occurred.
- `rss_after_split`: The total RSS after the split was applied.

This output provides the foundation for later risk estimation and model selection.

```
# Calculates the RSS for a vector of y-values within a single bin
calc_rss <- function(y) {
  # If there are no points in this bin, return 0 (no contribution to error)
  if (length(y) == 0) {
    return(0)
```

```
  }
  # Calculate the mean of y-values in the bin
  mean_y <- mean(y)
  # Return the sum of squared differences from the mean
  sum((y - mean_y)^2)
}
```

```
# Adaptive bin-width regressogram fitting function
# This function iteratively splits data to minimize RSS, under bin size constraints.
fit_regressogram <- function(x_all, # Numeric vector of all xx values
                             y_all, # Numeric vector of all yy values
                             min_split = 5, # Minimum observation parent to split
                             max_leaf = 5) { # Overall stopping condition

  # Combine x and y into a single data frame
  full_data <- data.frame(xx = x_all, yy = y_all)

  # Initialize with all data in one bin
  active_bins <- list(full_data)

  # Track split history and RSS after each split
  split_records <- data.frame(
    bins_k = integer(),
    split_location = numeric(),
    rss_after_split = numeric()
  )

  # Iterative splitting process
  while (TRUE) {
    # Store best split candidate in current iteration
    best_split <- list(
      rss_k_plus_1 = Inf,  # Best RSS so far
      parent_bin_idx = NA,           # Index of bin to be split
      split_value = NA,              # Location of split
      left_child = NULL,         # Left bin data
      right_child = NULL         # Right bin data
    )

    # Flag to check if any progress can be made
    can_split <- FALSE

    # Iterate through all current bins to find the best split
    for (i in 1:length(active_bins)) {
      parent_data <- active_bins[[i]]

      # Stopping condition: Don't split a bin with too small observations
      if (nrow(parent_data) < min_split) {
        next
      }

      # To split, we need at least two unique xx values in the bin
      unique_x <- sort(unique(parent_data$xx))
      if (length(unique_x) < 2) {
```

```
      next # Cannot find a split point in this bin
    }

    # Found at least one bin that could be split
    can_split <- TRUE

    # Compute RSS from all other bins
    # This sum will be part of the total RSS for the new (k+1)-bin model
    rss_other_bins <- 0
    for (j in 1:length(active_bins)) {
      if (j != i) { # If it's not the parent bin currently being considered for splitting
        rss_other_bins <- rss_other_bins + calc_rss(active_bins[[j]]$yy)
      }
    }

    # Midpoints between sorted unique x-values are potential split points
    mids <- (unique_x[-length(unique_x)] + unique_x[-1]) / 2

    # Try each split point for current bin
    for (sp_value in mids) {
      left_data <- parent_data[parent_data$xx < sp_value, ]
      right_data <- parent_data[parent_data$xx > sp_value, ]

      # Skip if either side is empty
      if (nrow(left_data) == 0 || nrow(right_data) == 0) {
        next
      }

      # Compute RSS for this split
      rss_children <- calc_rss(left_data$yy) + calc_rss(right_data$yy)

      # Total RSS if this parent bin (i) is split at sp_value
      total_rss <- rss_other_bins + rss_children

      # Update best split if this one is better
      if (total_rss < best_split$rss_k_plus_1) {
        best_split$rss_k_plus_1 <- total_rss
        best_split$parent_bin_idx <- i
        best_split$split_value <- sp_value
        best_split$left_child <- left_data
        best_split$right_child <- right_data
      }
    }
  }

  # Stop if no valid split improves the RSS
  if (!can_split || is.infinite(best_split$rss_k_plus_1)) {
    break
  }

  # Record the selected split
  split_records <- rbind(split_records, data.frame(
    bins_k = length(active_bins) + 1, # k+1 bins
```

```
      split_location = best_split$split_value,
      rss_after_split = best_split$rss_k_plus_1
    ))

    # Remove parent bin and add its children
    parent_idx <- best_split$parent_bin_idx

    # Create a new list for all old bins except the one split
    new_active_bins <- list()

    for (j in 1:length(active_bins)) {
      if (j != parent_idx) {
        new_active_bins[[length(new_active_bins) + 1]] <- active_bins[[j]]
      }
    }
    new_active_bins[[length(new_active_bins) + 1]] <- best_split$left_child
    new_active_bins[[length(new_active_bins) + 1]] <- best_split$right_child

    active_bins <- new_active_bins

    # Stop if no bin exceeds max allowed size
    if (length(active_bins) > 0) { # Ensure there's at least one bin
        max_in_bin <- max(sapply(active_bins, nrow))
        if (max_in_bin <= max_leaf) {
            break # Exit the while(TRUE) loop
        }
    } else { # Should not happen if logic is correct and n > 0
        break
    }
  } # End while(TRUE) loop for iterative splitting

  return(split_records)
}
```

```
# Run the partitioning routine
# Using min_split = 5, max_leaf = 5
split_results <- fit_regressogram(x, y)

# Display the results
print("Selected Splits and Corresponding RSS:")
```

```
## [1] "Selected Splits and Corresponding RSS:"
```

```
print(head(split_results))
```

```
##   bins_k split_location rss_after_split
## 1      2         0.8915       1112.5196
## 2      3         0.5615        473.5953
## 3      4         0.8430        322.2475
## 4      5         0.6605        256.7557
## 5      6         0.4725        228.8524
## 6      7         0.9815        201.9790
```

3.3 Risk Estimation Using Complexity-Penalized Criterion

To select the best adaptive regressogram model, we estimate the **prediction risk** for each model with k bins using a penalty-adjusted criterion inspired by Efron's covariance penalty.

It performs the following steps:

- Estimates the residual variance $\hat{\sigma}^2$ using a baseline 20-bin regressogram.
- Computes a risk score for each candidate model based on:

$$\text{Risk}(k) = \frac{\text{RSS}(k)}{n} + \frac{2\hat{\sigma}^2 k \log(k)}{n}$$

- The first term measures in-sample fit, while the second penalizes model complexity.

The printed table displays the first few combinations of k, RSS, and corresponding estimated risk. This result is used in the next step to identify the optimal number of bins $k^\star$.

```
# Divide x into 20 equal-width bins (as in Question 1 of Assignment One)
bins_20 <- cut(x, breaks = 20, include.lowest = TRUE)

# Compute the mean y value for each bin (used as predicted value)
y_hat_means <- tapply(y, bins_20, mean, na.rm = TRUE)

# Assign predicted y values based on bin membership
predicted_y <- y_hat_means[as.character(bins_20)]

# Calculate residuals (difference between observed and predicted y)
resid <- y - predicted_y

# Estimate sigma^2 using unbiased estimator: RSS / (n - k)
k <- nlevels(bins_20) # number of bins actually used
n <- length(y)        # total number of observations
sigma_sq_hat <- sum(resid^2, na.rm = TRUE) / (n - k)

# Extract k (number of bins) and RSS values from previous regressogram fitting
k_values <- split_results$bins_k
rss_values <- split_results$rss_after_split

# Calculate penalty: 2 * sigma_sq_hat * k * log(k) / n
penalties <- (2 * sigma_sq_hat * k_values * log(k_values)) / n

# Calculate estimated risk for each model
est_risk <- (rss_values / n) + penalties

# Store results in a data frame
risk_data <- data.frame(
  k = k_values,
  rss = rss_values,
  risk = est_risk
)

# Print the first few rows of the result
print("Calculated Risk Data (first few rows):")
```

```
## [1] "Calculated Risk Data (first few rows):"
```

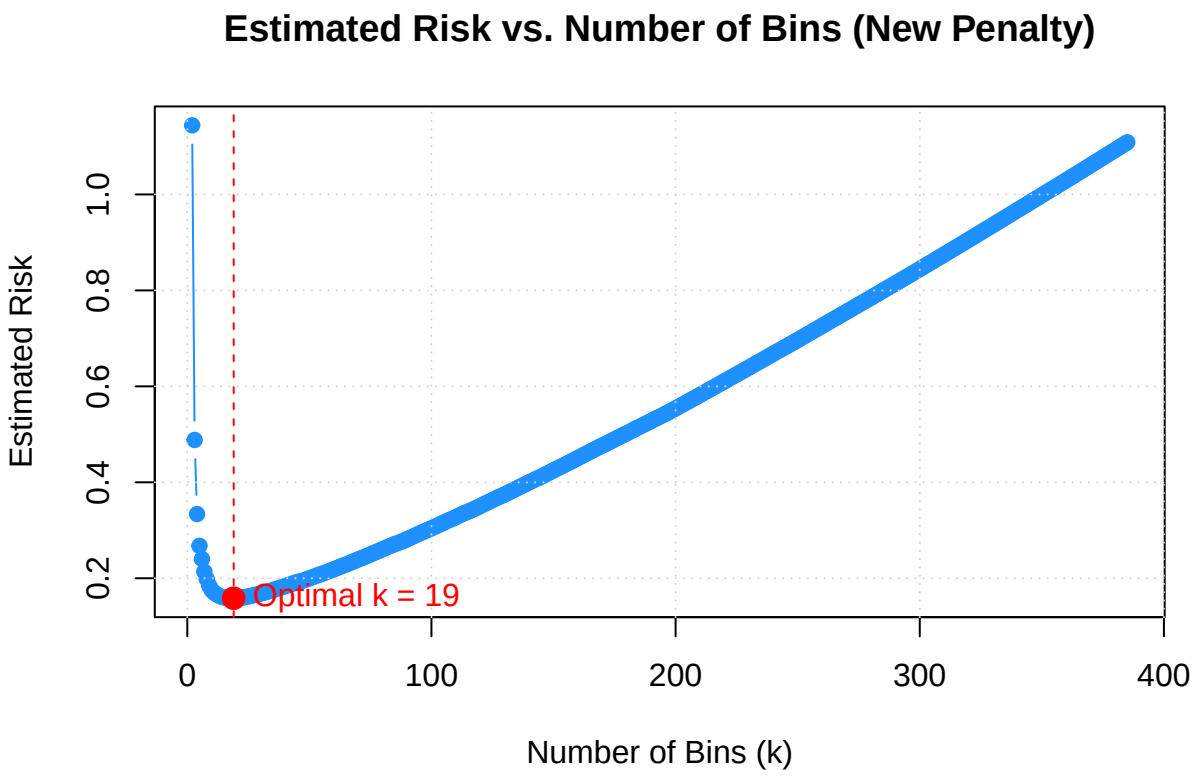
```
print(head(risk_data))
```

```
##   k       rss      risk
## 1 2 1112.5196 1.1440269
## 2 3  473.5953 0.4882487
## 3 4  322.2475 0.3337327
## 4 5  256.7557 0.2675710
## 5 6  228.8524 0.2401332
## 6 7  201.9790 0.2138307
```

3.4 Model Selection: Complexity-Penalized Risk

The estimated risk decreases as the number of bins increases, but a complexity penalty prevents overfitting. The optimal number of bins is selected where the penalized risk curve reaches its minimum. The red dot in the figure below highlights this point, while the dashed line indicates its corresponding value of k. This approach allows objective model selection based on a data-driven trade-off between fit and complexity.

```
# Check if 'risk_data' is available and contains valid (non-NA) risk values
if (nrow(risk_data) > 0 && any(!is.na(risk_data$risk))) {
  # Plot estimated risk vs. number of bins (k)
  plot(risk_data$k, risk_data$risk,
       type = "b", # "b" for both points and lines
       xlab = "Number of Bins (k)",
       ylab = "Estimated Risk",
       main = "Estimated Risk vs. Number of Bins (New Penalty)",
       pch = 19, col = "dodgerblue")
  grid()

  # Identify the index and value of minimum risk
  min_risk_idx <- which.min(risk_data$risk)
  k_opt <- risk_data$k[min_risk_idx]           # Optimal number of bins (k)
  minRiskVal <- risk_data$risk[min_risk_idx]  # Corresponding minimum risk value

  # Highlight the minimum risk point on the plot
  points(k_opt, minRiskVal, col = "red", pch = 19, cex = 1.5)
  abline(v = k_opt, col = "red", lty = 2)
  text(k_opt, minRiskVal, labels = paste("Optimal k =", k_opt), pos = 4, col = "red")

} else {
  print("No valid risk data to plot. Check 'sigma_hat_sq' and 'split_results'.")
}
```



The figure shows that the risk decreases rapidly as k increases, then stabilizes.
The plotted curve illustrates the estimated risk for each candidate bin count k.
While adding more bins typically reduces RSS, the penalty term increases — leading to a trade-off.
Overfitting is avoided due to the complexity penalty, and the selected value $k^\star$, marked in red, minimizes this total risk.

```
# Check if the variable k_opt exists and is not NA
# This means the optimal k was successfully found during the plotting step
if (exists("k_opt") && !is.na(k_opt)) {
  # Retrieve the Residual Sum of Squares (RSS) for the optimal k
  rss_opt <- risk_data$rss[risk_data$k == k_opt]

  # Report the selected optimal k and its corresponding RSS
  cat("Selected optimal k (minimizing estimated risk with new penalty):", k_opt, "\n")
  cat("Corresponding RSS for optimal k:", round(rss_opt, 4), "\n")

} else {
  # If k_opt was not found or risk_data is invalid
  cat("Optimal k could not be determined from the risk data.\n")

  # Ensure k_opt is NA if not found, for the next chunk's conditional execution
  if(!exists("k_opt")) k_opt <- NA
}
```

```
## Selected optimal k (minimizing estimated risk with new penalty): 19
## Corresponding RSS for optimal k: 129.0847
```

We now report the selected model parameters:
- The optimal number of bins $k^\star$ that minimizes the penalized risk
- The corresponding residual sum of squares (RSS) at that point

These values serve as the basis for constructing the final fitted model in the next section.

3.5 Best-Fit Adaptive Regressogram

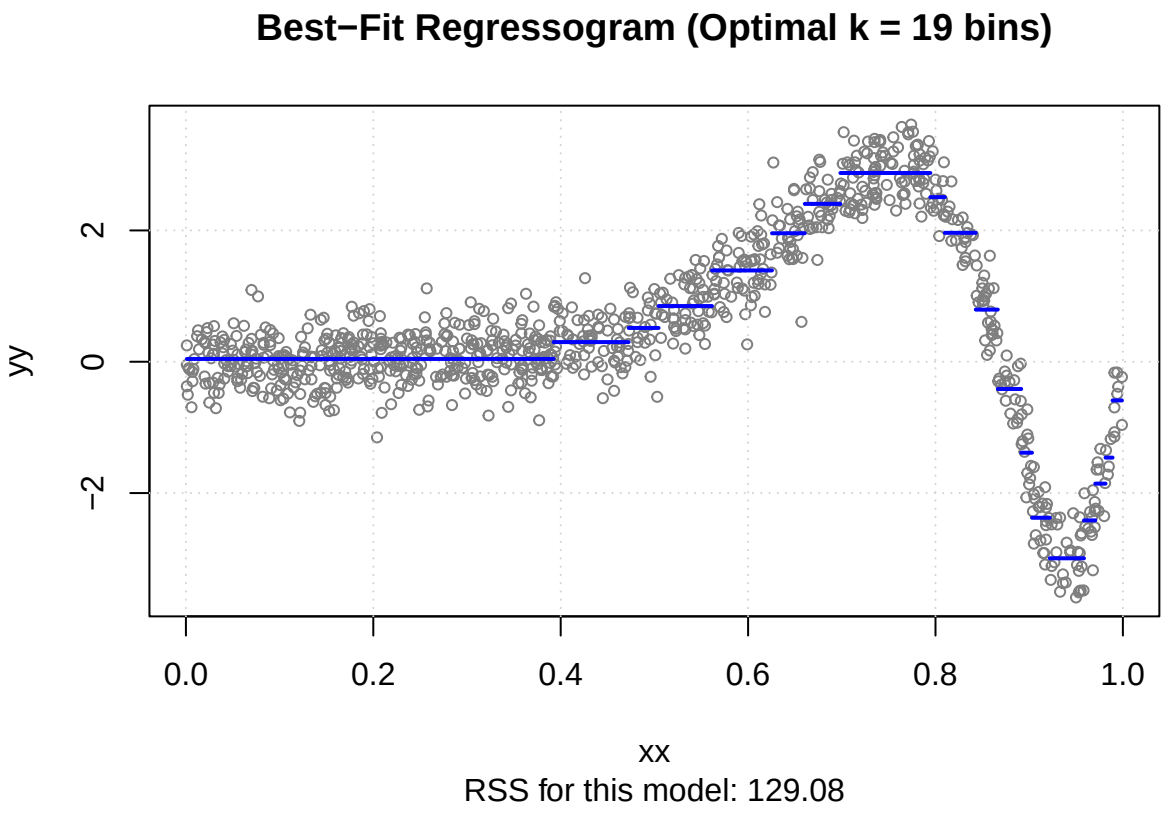
We now visualize the final fitted model using the optimal bin count $k^\star = 19$,
as selected by the complexity-penalized risk minimization in the previous section.

This model is a **step-function regressogram**, where the input domain is partitioned using **adaptive, data-driven splits**
chosen to minimize the **Residual Sum of Squares (RSS)** at each step.

- When $k = 1$, the model simplifies to a constant — the grand mean across all data.
- When $k > 1$, the data is segmented into k intervals, with each bin assigned its own constant prediction.

```
# Check if optimal k (k_opt) exists and is valid (non-NA and >= 1)
if (!is.na(k_opt) && k_opt >= 1) {
  # Base plot: raw (x, y) data with title and RSS as subtitle
  plot(x, y,
       main = paste("Best-Fit Regressogram (Optimal k =", k_opt, "bins)"),
       xlab = "xx", ylab = "yy", pch = 1, cex = 0.7, col = "grey50",
       sub = paste("RSS for this model:", round(ifelse(exists("rss_opt")
            && !is.na(rss_opt), rss_opt, NA), 2)))
  grid()

  # Case 1: If the best k is 1, draw a horizontal line at the overall mean
  if (k_opt == 1) {
    # If optimal k is 1, the prediction is the grand mean
    abline(h = mean(y), col = "blue", lwd = 2)
    legend("topright", legend = "Grand Mean (k=1)", col = "blue", lwd = 2, bty = "n")

  # Case 2: If optimal k > 1 and can be constructed from available split results
  } else if (k_opt > 1 && k_opt <= (nrow(split_results) + 1) ) {
    # We need (k_opt - 1) splits to define k_opt bins.
    # These splits are the first k_opt - 1 entries in split_results$split_location
    num_splits <- k_opt - 1

    # Ensure we have enough splits to draw the model
    if (num_splits > 0 && num_splits <= nrow(split_results)) {
      opt_splits <- sort(split_results$split_location[1:num_splits])

      # Define bin boundaries using min/max of x and the optimal splits
      bin_bounds <- c(min(x, na.rm = TRUE), opt_splits, max(x, na.rm = TRUE))

      # Add a tiny offset to min and max to prevent edge-case bin assignment errors
      bin_bounds[length(bin_bounds)] <- bin_bounds[length(bin_bounds)] + 1e-9
      bin_bounds[1] <- bin_bounds[1] - 1e-9

      # Assign each x to its optimal bin
      opt_bin_assign <- cut(x, breaks = bin_bounds, include.lowest = TRUE,
                            right = FALSE, dig.lab = 4)
      # Compute the mean y for each bin
      opt_bin_means <- tapply(y, opt_bin_assign, mean, na.rm = TRUE)
```

```
      # Draw horizontal segments for each bin
      for (i in 1:length(opt_bin_means)) {
        if (!is.na(opt_bin_means[i])) {
          x_start <- bin_bounds[i]
          x_end <- bin_bounds[i+1]
          y_mean <- opt_bin_means[i]
          segments(x_start, y_mean, x_end, y_mean, col = "blue", lwd = 2)
        }
      }
    } else {
      print(paste("Cannot plot best-fit model for k_opt =", k_opt))
    }
  }
} else {
  print("Best-fit model cannot be plotted as optimal k was not determined.")
}
```



The figure below presents the final adaptive regressogram fitted using the optimal number of bins, $k = 19$, as determined in the previous step. Each **gray circle** represents an observed (x, y) data point, while the **blue horizontal segments** indicate the model's predicted values within each bin. These segments form a piecewise-constant approximation of the underlying function.

The model automatically adjusts its bin widths based on data complexity: it uses narrower bins in regions with rapid changes (e.g., near the initial acceleration peak around xx = 0.015), and wider bins in smoother areas (e.g., the deceleration phase after xx = 0.04). This adaptiveness helps the model capture meaningful local patterns without overfitting to noise.

Overall, the fitted step-function model achieves a balance between flexibility and simplicity. The reported **residual sum of squares (RSS)** confirms that this choice of k yields a statistically justifiable and visually

coherent fit.

4. Comparative Analysis: Histogram Risk Estimation on `qer.csv`

To validate our risk-based model selection framework on a different problem, we apply a similar procedure to a second dataset, `qer.csv`. Here, the goal is to find the optimal number of bins for a density histogram, a related but distinct task. For this, we employ a penalized risk criterion specifically designed for histogram density estimation by **Birgé and Rozenholc (2006)**.

$$\text{Risk}'(k) = -\sum_{N_j>0} N_j \log(N_j) - n\log(k) + \text{pen}(k)$$

$$\text{pen}(k) = (k-1) + (\log k)^{2.5}$$

This comparative analysis serves to demonstrate the flexibility and effectiveness of the general risk-minimization approach across different models and datasets.

```
# Load the qer.csv dataset
qer_data <- read.csv("/Users/nahyunlee/Desktop/qer.csv")

# Extract x variable
x_qer <- qer_data$xx
n_qer <- length(x_qer)

# Determine maximum number of bins to try: up to floor(n/5), capped at 100
k_max_hist <- floor(min(100, n_qer / 5))

# Ensure k_max_hist is valid even for small n_qer
if (k_max_hist < 1 && n_qer > 0) { # If n_qer is very small (e.g., < 5)
    k_max_hist <- 1 # Ensure at least k=1 is considered
} else if (k_max_hist < 1) { # If n_qer is 0
    k_max_hist <- 0 # No k values to test
}

# Create sequence of k values to test (1 to k_max_hist)
k_hist_vals <- if(k_max_hist > 0) 1:k_max_hist else integer(0)

# Initialize result storage
hist_risk <- data.frame(k = integer(), risk = numeric())

# Determine the range of x values
min_qer <- min(x_qer, na.rm = TRUE)
max_qer <- max(x_qer, na.rm = TRUE)
range_qer <- max_qer - min_qer

# Only compute if we have data and at least one k value to test
if (n_qer > 0 && length(k_hist_vals) > 0) {
  # Handle the special case where all x values are identical
  if (range_qer == 0) {
    warning("All data points in x_qer are identical.
            Risk calculated for k=1 only, assuming it makes sense.",
            call. = FALSE)
    if (1 %in% k_hist_vals) {
```

```
      # If all data are identical and k=1, risk simplifies to -n log(n) + 1,
      # assuming the main term is -sum(Nj*log(Nj)) and log(1) = 0
      NlogN_k1 <- if (n_qer > 0) -n_qer * log(n_qer) else 0
      NlogK_k1 <- -n_qer * log(1) # This is 0
      Kpen_k1 <- (1 - 1) + (log(1))^2.5
      risk_k1 <- NlogN_k1 + NlogK_k1 + Kpen_k1
      hist_risk <- rbind(hist_risk, data.frame(k = 1, risk = risk_k1))
    }
  } else { # range_qer > 0
    # Regular case where x values vary across range
    for (kh in k_hist_vals) {
      if (kh == 0) next # k must be at least 1

      # Create equal-width bin edges from min to max of x
      hist_breaks <- seq(min_qer, max_qer + 1e-9, length.out = kh + 1)

      # Compute histogram without plotting
      hist_output <- hist(x_qer, breaks = hist_breaks, plot = FALSE,
                          right = FALSE, include.lowest = TRUE)
      nj_counts <- hist_output$counts # Bin counts

      # Calculate risk components:
      # Term 1: -sum(Nj*log(Nj) for Nj>0)
      NlogN <- -sum(nj_counts[nj_counts > 0] * log(nj_counts[nj_counts > 0]))

      # Term 2: -n*log(k)
      NlogK <- -n_qer * log(kh)

      # Term 3 (penalty): (k-1) + (log k)^2.5
      if (kh == 1) {
        k_pen <- (kh - 1) + (0)^2.5 # (1-1) + 0 = 0, log(1)=0
      } else {
        k_pen <- (kh - 1) + (log(kh))^2.5
      }

      # Total risk
      current_risk <- NlogN + NlogK + k_pen

      # Store result
      hist_risk <- rbind(hist_risk, data.frame(k = kh, risk = current_risk))
    }
  }
}

# Display the top of the result
print("Calculated Histogram Risk Data (first few rows):")
```

```
## [1] "Calculated Histogram Risk Data (first few rows):"
```

```
print(head(hist_risk))
```

```
##   k      risk
```

```
## 1 1 -7097.827
## 2 2 -7103.012
## 3 3 -7121.667
## 4 4 -7172.404
## 5 5 -7230.767
## 6 6 -7265.169
```

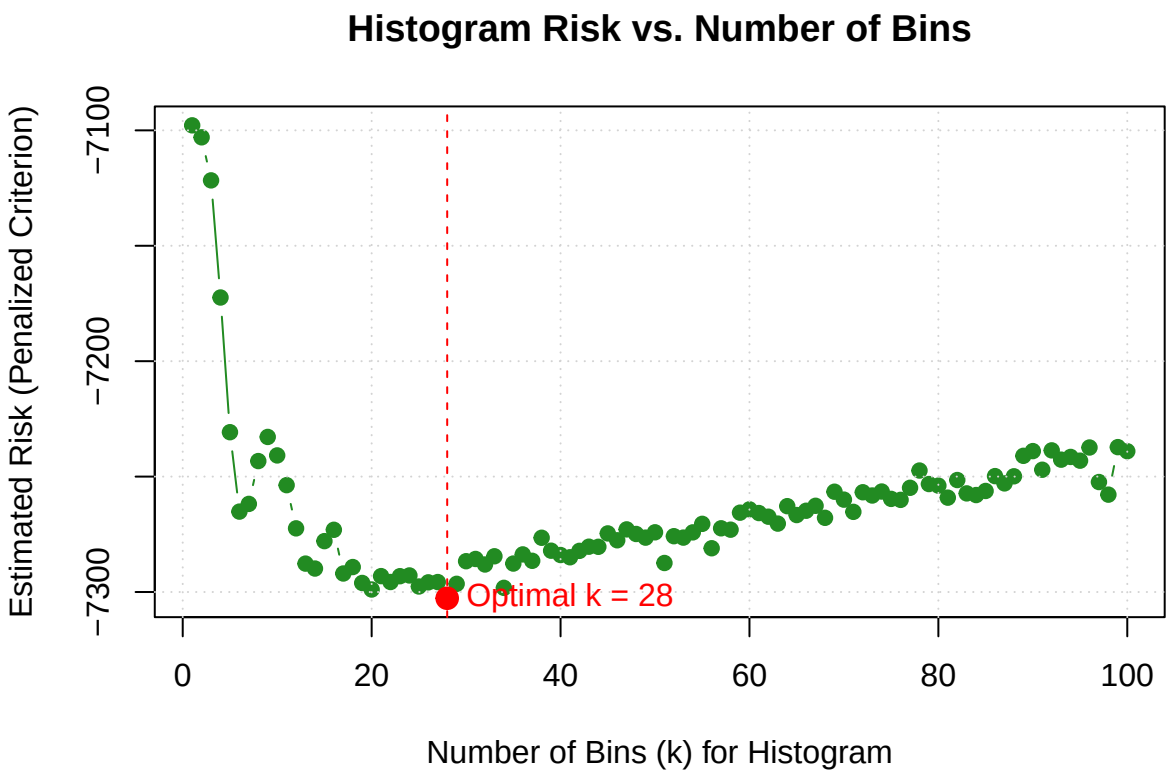
The table above shows the estimated risk values for different bin counts.
As expected, the risk decreases initially with increased flexibility, but grows again when the model becomes overly complex.

The optimal bin count (minimizing the penalized risk) is highlighted in the plot below.

4.1 Optimal Bin Selection using Penalized Histogram Risk

We now plot the penalized risk values computed for varying numbers of bins k,
based on Birgé and Rozenholc's risk formulation.

```r
# Check that histogram risk data exists and contains valid (non-NA) values
if (nrow(hist_risk) > 0 && any(!is.na(hist_risk$risk))) {
  # Plot estimated histogram-based risk vs. number of bins
  plot(hist_risk$k, hist_risk$risk,
       type = "b",
       xlab = "Number of Bins (k) for Histogram",
       ylab = "Estimated Risk (Penalized Criterion)",
       main = "Histogram Risk vs. Number of Bins",
       pch = 19, col = "forestgreen")
  grid()

  # Highlight the minimum risk
  if(any(!is.na(hist_risk$risk))){
      min_hist_risk <- which.min(hist_risk$risk)
      k_hist_opt <- hist_risk$k[min_hist_risk]
      min_hist_risk_val <- hist_risk$risk[min_hist_risk]

      points(k_hist_opt, min_hist_risk_val, col = "red", pch = 19, cex = 1.5)
      abline(v = k_hist_opt, col = "red", lty = 2)
      text(k_hist_opt, min_hist_risk_val,
           labels = paste("Optimal k =", k_hist_opt), pos = 4, col = "red")
  }
} else {
  # If risk data is empty or invalid
  print("No valid histogram risk data to plot.")
}
```




```
# Check if the optimal histogram bin count (k_hist_opt) exists and is valid
if (exists("k_hist_opt") && !is.na(k_hist_opt)) {
  # Report the optimal number of bins selected by minimizing histogram risk
  cat("Optimal number of bins (k):", k_hist_opt, "\n")
} else {
  # If optimal k could not be determined
  cat("Optimal k for the histogram could not be determined from the risk data.\n")
}
```

```
## Optimal number of bins (k): 28
```

The plot shows the estimated penalized risk across different histogram bin counts.
Initially, as k increases, the model captures more structure in the data, and the risk drops.
However, overly fine partitions lead to increased complexity, which is penalized by the Birgé–Rozenholc criterion:

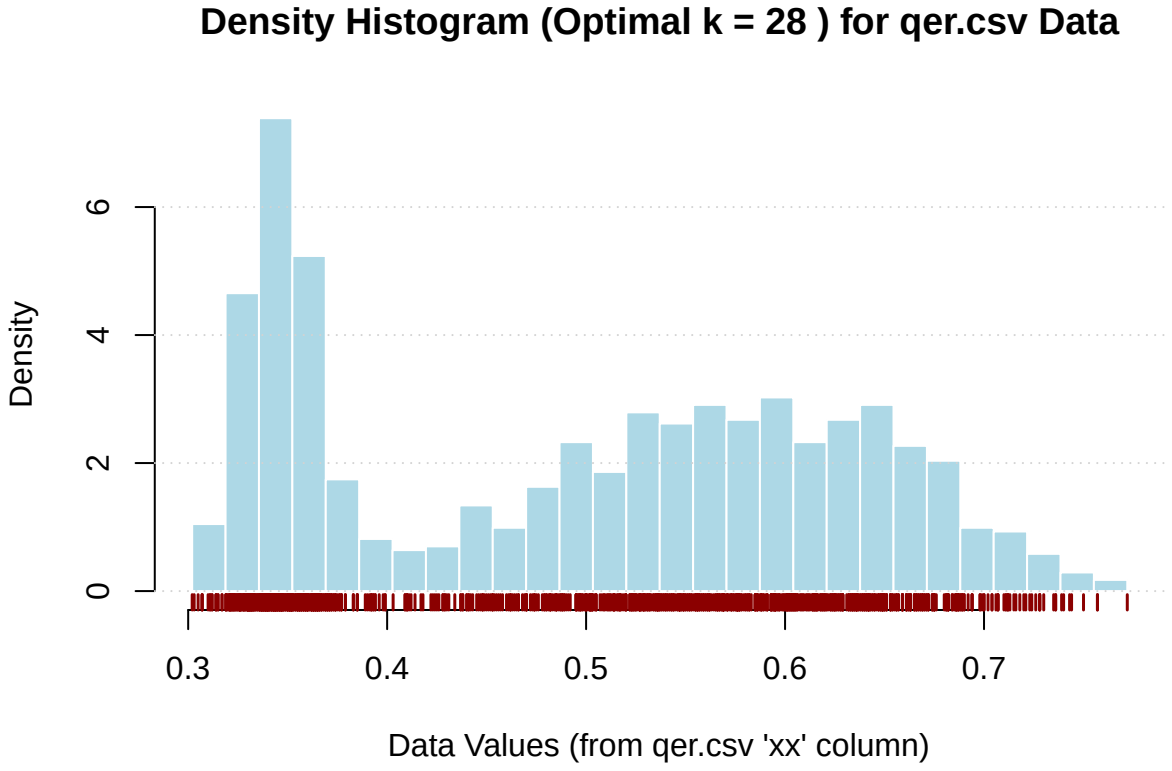
$$\text{Risk}'(k) = -\sum N_j \log(N_j) - n \log(k) + (k-1) + (\log k)^{2.5}$$

The **red dot and dashed line** indicate the optimal number of bins ($k = 28$),
where the trade-off between data fit and model complexity is best balanced.

4.2 Visualization of the Optimal Histogram

Using the optimal bin count $k = 28$ selected from the penalized risk criterion,
we visualize the density histogram of the `qer.csv` dataset.

```r
# Plot the histogram using the optimal number of bins (k_hist_opt)
# Check if optimal k is valid, and x_qer has data
if (!is.na(k_hist_opt) && k_hist_opt > 0 && length(x_qer) > 0) {
  # Define histogram bin edges using the optimal number of bins (k_hist_opt)
  # Add a small epsilon to max_qer to ensure the maximum data point is included
  # when hist() uses right=FALSE (intervals like [a,b)).
  hist_breaks_opt <- seq(min_qer, max_qer + 1e-9, length.out = k_hist_opt + 1)

  # Plot the histogram of x_qer with density (normalized) scale
  hist(x_qer,
       breaks = hist_breaks_opt,
       freq = FALSE,              # Plot as density
       main = paste("Density Histogram (Optimal k =", k_hist_opt, ") for qer.csv Data"),
       xlab = "Data Values (from qer.csv 'xx' column)",
       ylab = "Density",
       col = "lightblue",
       border = "white",
       right = FALSE,             # Intervals are [a,b)
       include.lowest = TRUE      # Include the smallest value in the first bin
      )

  # Add a rug plot to show individual data points along the x-axis.
  # 'lwd' controls the line width of the rug "whiskers".
  rug(x_qer, col = "darkred", lwd = 1.5) # lwd=1.5 makes whiskers clearly visible

  # Add horizontal grid lines for easier reading of the y-axis (density)
  grid(nx = NA, ny = NULL)
}
```



Each bin has equal width and shows the normalized frequency (density) of data points falling into that range. The rug plot at the bottom marks individual data points, providing a sense of granularity and data concentration.

This visualization allows us to verify that the selected k balances resolution and interpretability — capturing the multimodal structure without overfitting noise.

5. Conclusion

In this research, we successfully designed and implemented a method for constructing an adaptive-width regressogram via recursive RSS minimization. Crucially, we demonstrated that an objective, data-driven model selection is possible using a custom penalized risk criterion. The final model effectively balanced complexity and goodness-of-fit, adapting its structure to the local features of the crash-test data.

Furthermore, a comparative analysis on a second dataset using a specialized histogram risk penalty reinforced the validity and flexibility of this general framework. Both analyses confirm that model selection based on penalized risk is a powerful and essential tool for building robust, interpretable non-parametric models. This approach provides a reliable template for signal extraction in other complex, real-world systems.