Tidy Survival Analysis: Applying R's Tidyverse to Survival Data

Module 1. Introduction

Lu Mao

lmao@biostat.wisc.edu

Department of Biostatistics & Medical Informatics

University of Wisconsin-Madison

Aug 3, 2025

Table of contents

Basics of Survival Analysis

Time-to-Event Data

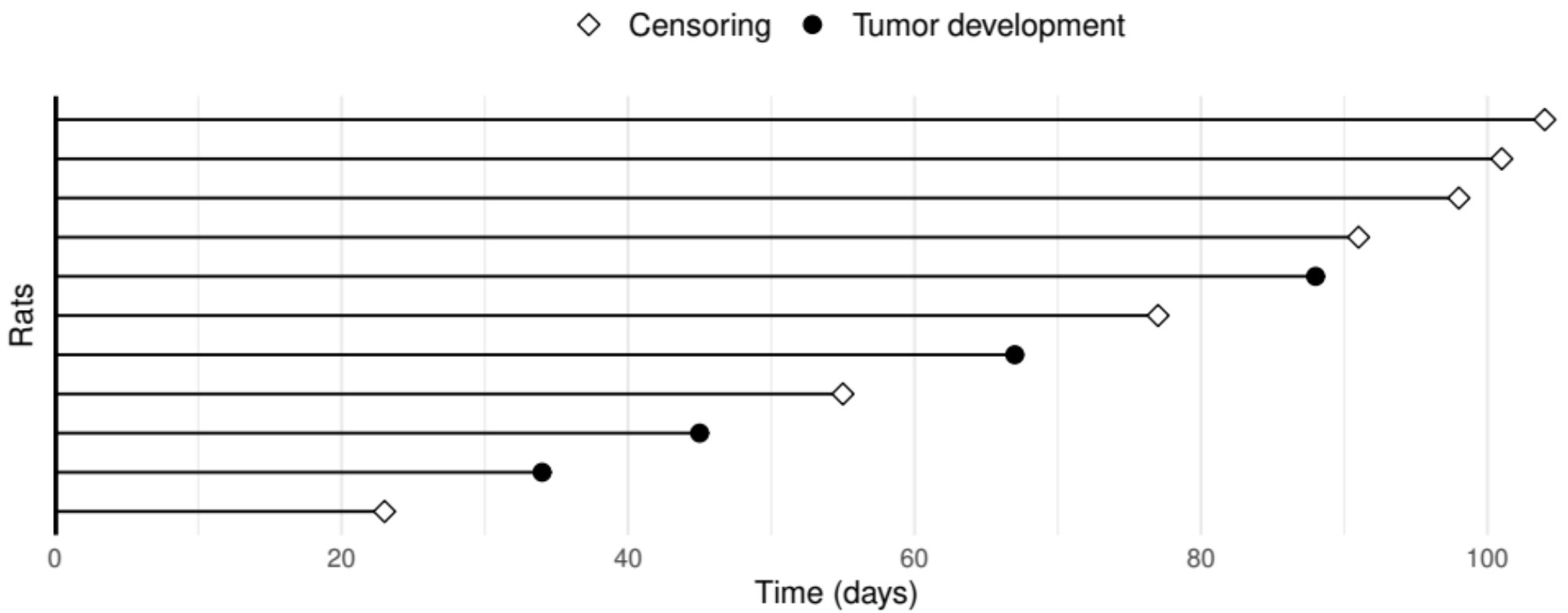
- A common type of outcome in medical and clinical studies
  - **Starting point**: Randomization, diagnosis, enrollment, birth, etc.
  - **Endpoint (Event of interest)**: Death, disease onset, hospitalization, etc.
    - **Engineering**: Failure times of machines or components (reliability)
    - **Social sciences**: Time to job change, dropout, or event occurrence
- **Right censoring**:
  - Event not observed within the follow-up period
  - Due to study ending, dropout, or loss to follow-up
  - We only know:

    $$T > C$$

    where T is event time and C is censoring time

Follow-up (Swimmer) Plot

- A rat tumorigenicity study



Basic Estimands

- **Survival function:** $S(t) = \Pr(T > t)$
  - Probability subject survives beyond time t
- **Hazard function:**

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{\Pr(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t}$$

  - Instantaneous risk of failure at time t
- **Relationship**

$$S(t) = \exp\left(- \int_0^t \lambda(u) \mathrm{d}u \right)$$

  - **Cumulative hazard function:** $\Lambda(t) = \int_0^t \lambda(u) \mathrm{d}u$

Observed (Censored) Data

- **Notation**: (X, δ)
  - $X = \min(T, C)$: observation time (event or censoring)
  - $\delta = I(T \leq C)$: event indicator (1 for event, 0 for censoring)
- **Data format**

```
# time = X, status = delta (tidy format)
    id    time status
1    1     5      1
2    2     3      0
3    3     8      1
4    4     2      0
5    5     6      1
# Alternatively
    id    time
1    1     5
2    2     3+
3    3     8
4    4     2+
5    5     6
```

German Breast Cancer Study: A Working Example

German Breast Cancer (GBC) Study

- **Study Information**
  - **Population**: 686 patients with node-positive breast cancer
  - **Objective**: Assess if tamoxifen + chemo reduces mortality/relapse
  - **Baseline info**: Age, tumor size, hormone levels, menopausal status, etc.
  - **Follow-up**: Median 44 months
    - 171 deaths $\rightarrow$ exact times known
    - 515 censored $\rightarrow$ survival time $>$ censoring time
- **Data sets**:
  - Mortality data: https://lmaowisc.github.io/tidysurv/data/gbc_mort.txt
  - Mortality + relapse: https://lmaowisc.github.io/tidysurv/data/gbc.txt
  - Download and save in a `data` folder under your root directory

Data Format (I)

- **Death only**

```r
# Load mortality data
gbc_mort <- read.table("data/gbc_mort.txt", header = TRUE)
# Check the first few rows of the data frame
head(gbc_mort)
```

```
  id      time status hormone age meno size grade nodes prog estrg
1  1 74.819672      0       1  38    1   18     3     5  141   105
2  2 65.770492      0       1  52    1   20     1     1   78    14
3  3 47.737705      1       1  47    1   30     2     1  422    89
4  4  4.852459      0       1  40    1   24     1     3   25    11
5  5 61.081967      0       2  64    2   19     2     1   19     9
6  6 63.377049      0       2  49    2   56     1     3  356    64
```

```r
# The data frame 'gbc_mort' contains:
#  time:   time (months) to death or censoring
#  status: event indicator (1 = death, 0 = censoring)
# hormone: Hormone therapy (1 = no, 2 = yes); age: Age at diagnosis (years);
# meno: Menopausal status (1 = no, 2 = yes); size: Tumor size (mm); grade: Tumor grade (1–3);
# nodes: Number of positive lymph nodes; prog: Progesterone receptor level (fmol/mg); estrg:
# Estrogen receptor level (fmol/mg).
```

Data Format (II)

- **Mortality + relapse**

```r
# Load mortality + relapse data
gbc <- read.table("data/gbc.txt", header = TRUE)
# Check the first few rows of the data frame
head(gbc)
```

```
  id     time status hormone age meno size grade nodes prog estrg
1  1 43.83607      1       1  38    1   18     3     5  141   105
2  1 74.81967      0       1  38    1   18     3     5  141   105
3  2 46.55738      1       1  52    1   20     1     1   78    14
4  2 65.77049      0       1  52    1   20     1     1   78    14
5  3 41.93443      1       1  47    1   30     2     1  422    89
6  3 47.73770      2       1  47    1   30     2     1  422    89
```

```r
# The data frame 'gbc' contains:
#  time:   time (months) to death, relapse, or censoring
#  status: event indicator (1 = relapse, 2 = death, 0 = censoring)
#  other covariates the same as in gbc_mor.
```

Analysis Goals

- **Descriptive**
  - Summarize patient characteristics
  - Visualize survival distributions
- **Inferential**
  - Compare survival curves (e.g., hormone therapy vs. no hormone therapy)
  - Assess impact of covariates on survival (e.g., age, tumor size, etc.)
  - Model competing risks (e.g., relapse vs. death)
- **Predictive**
  - Develop risk prediction models
  - Evaluate model performance (e.g., concordance index, calibration)

Standard Analysis with `survival` Package

Survival Package Overview

- **Key Functions**
  - `Surv()`: Create survival object
  - `survfit()`: Fit Kaplan-Meier survival curves
  - `survdiff()`: Compare survival curves (log-rank test)
  - `coxph()`: Fit Cox proportional hazards regression models
  - `survreg()`: Fit parametric survival regression models

Kaplan-Meier Survival Curves

- **Create dataset for relapse-free survival**

```r
# Sort by subject id, then time
o <- order(gbc$id, gbc$time)
gbc <- gbc[o,]
# Keep only first row per subject => first event
df <- gbc[!duplicated(gbc$id), ]
# Convert status > 0 to 1 if it is either relapse or death
df$status <- ifelse(df$status > 0, 1, 0)
head(df)
```

```
  id      time status hormone age meno size grade nodes prog estrg
1  1 43.836066      1       1  38    1   18     3     5  141   105
3  2 46.557377      1       1  52    1   20     1     1   78    14
5  3 41.934426      1       1  47    1   30     2     1  422    89
7  4  4.852459      0       1  40    1   24     1     3   25    11
8  5 61.081967      0       2  64    2   19     2     1   19     9
9  6 63.377049      0       2  49    2   56     1     3  356    64
```

Kaplan-Meier Curves (I)

- **Fit Kaplan-Meier survival curves**

```r
library(survival)
# Fit KM curves by hormone treatment group
km_fit <- survfit(Surv(time, status) ~ hormone, data = df)
km_fit
```

```
Call: survfit(formula = Surv(time, status) ~ hormone, data = df)

            n events median 0.95LCL 0.95UCL
hormone=1 440    205   50.1    42.5    59.5
hormone=2 246     94   66.2    62.9      NA
```

Kaplan-Meier Curves (II)

- **Summarize survival estimates at specified time points**
  - For example, at 6, 12, 24, and 36 months

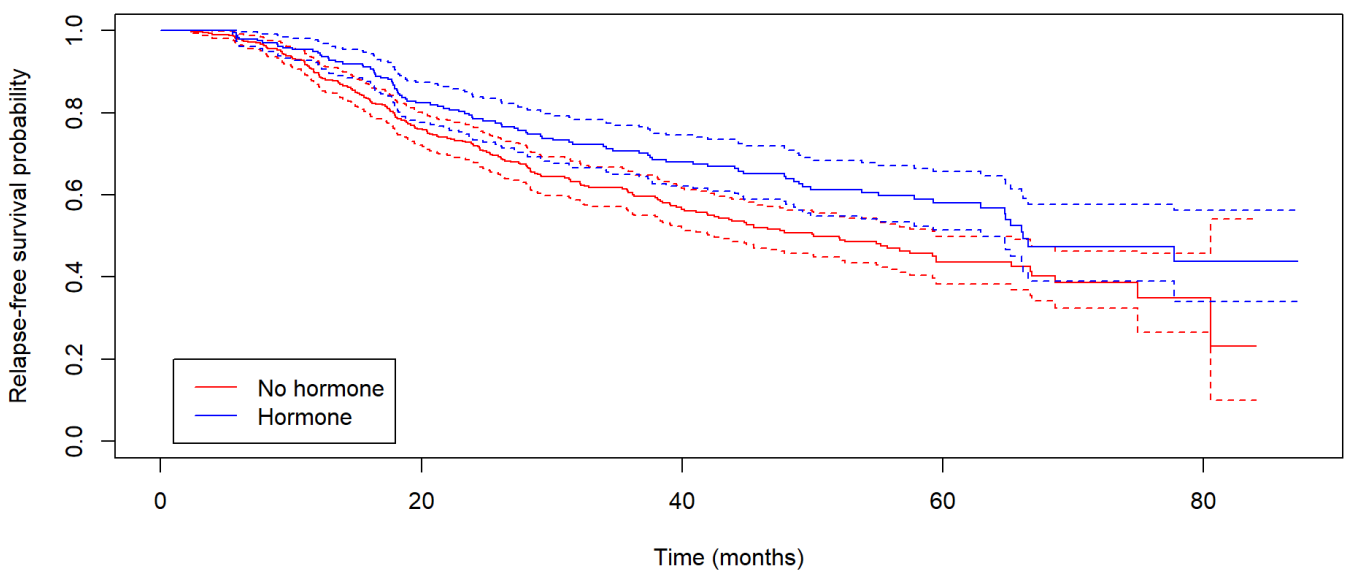
```r
summary(km_fit, times = c(6, 12, 24, 36))
```

```
Call: survfit(formula = Surv(time, status) ~ hormone, data = df)

                hormone=1
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    6    419       9    0.979 0.00691        0.966        0.993
   12    379      35    0.897 0.01476        0.868        0.926
   24    280      73    0.720 0.02203        0.678        0.764
   36    195      41    0.606 0.02475        0.559        0.656

                hormone=2
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    6    236       4    0.983 0.00826        0.967        1.000
   12    223       8    0.950 0.01418        0.922        0.978
   24    177      38    0.785 0.02701        0.733        0.839
   36    136      16    0.708 0.03047        0.650        0.770
```

Kaplan-Meier Curves (III)

- **Plot Kaplan-Meier survival curves by group**

```r
plot(km_fit, ylim = c(0,1), xlab = "Time (months)", ylab = "Relapse-free survival probability",
     col = c("red", "blue"), conf.int = TRUE)
# Add legend
legend(1, 0.2, col=c("red", "blue"), lty = 1,
       c("No hormone", "Hormone")) # Legend text
```



Log-Rank Test

- **Compare survival curves between groups**

```
lgr_obj <- survdiff(Surv(time, status) ~ hormone, data = df)
lgr_obj # Print log-rank test results
```

```
Call:
survdiff(formula = Surv(time, status) ~ hormone, data = df)

            N Observed Expected (O-E)^2/E (O-E)^2/V
hormone=1 440      205      180      3.37      8.56
hormone=2 246       94      119      5.12      8.56

 Chisq= 8.6  on 1 degrees of freedom, p= 0.003
```

```
lgr_obj$pvalue # Extract p-value
```

```
[1] 0.003427282
```

> **Exercise**
>
> Perform a log-rank test on treatment stratified by patient menopausal status `meno`.
>
> ▶ Solution

Cox Model - Model Specification

- **Cox proportional hazards model**

$$\lambda(t \mid Z) = \lambda_0(t) \exp(\beta_1 Z_1 + \beta_2 Z_2 + \ldots + \beta_p Z_p)$$

  - $\lambda_0(t)$: baseline hazard function
  - $Z = (Z_1, \ldots, Z_p)^{\mathrm{T}}$: covariates (e.g., hormone therapy, age, tumor size)
  - $\beta = (\beta_1 \ , \ \ldots \ , \beta_p)^{\mathrm{T}}$: regression coefficients
  - $\exp(\beta_j)$: hazard ratio for covariate Z_j

- **Proportional hazards (PH) assumption**

$$\frac{\lambda(t \mid Z)}{\lambda(t \mid Z^*)} = \exp\{\beta^{\mathrm{T}}(Z - Z^*)\}$$

  - HR constant over time, i.e., $\beta(t) \equiv \beta$ (for each covariate)

Cox Model - Model Fitting (I)

- **Model fitting:** `survival::coxph()`

```r
cox_fit <- coxph(Surv(time, status) ~ hormone + meno + age + grade + size + prog + estrg,
                 data = df)
summary(cox_fit) # Print model summary
```

```
Call:
coxph(formula = Surv(time, status) ~ hormone + meno + age + grade +
    size + prog + estrg, data = df)

  n= 686, number of events= 299

              coef  exp(coef)   se(coef)      z Pr(>|z|)
hormone -0.3422139  0.7101963  0.1290669 -2.651  0.00801 **
meno     0.2765637  1.3185909  0.1837781  1.505  0.13236
age     -0.0087813  0.9912572  0.0093375 -0.940  0.34700
grade    0.2785797  1.3212519  0.1051531  2.649  0.00807 **
size     0.0152793  1.0153966  0.0036877  4.143 3.42e-05 ***
prog    -0.0023307  0.9976720  0.0005803 -4.016 5.91e-05 ***
estrg    0.0001678  1.0001679  0.0004669  0.359  0.71923
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

        exp(coef) exp(-coef) lower .95 upper .95
```

Cox Model - Model Fitting (II)

- Extracting $\hat{\beta}$ and $\hat{\text{var}}(\hat{\beta})$

```r
beta <- cox_fit$coefficients # Estimated coefficients
vbeta <- vcov(cox_fit) # Estimated variance-covariance matrix
# Extract regression table (as data frame)
coef(summary(cox_fit))
```

```
                coef exp(coef)     se(coef)          z     Pr(>|z|)
hormone -0.3422138547 0.7101963 0.1290669350 -2.6514448 8.014821e-03
meno     0.2765636858 1.3185909 0.1837780595  1.5048787 1.323553e-01
age     -0.0087812621 0.9912572 0.0093375120 -0.9404285 3.469978e-01
grade    0.2785796730 1.3212519 0.1051531448  2.6492757 8.066449e-03
size     0.0152793172 1.0153966 0.0036877471  4.1432660 3.423945e-05
prog    -0.0023307288 0.9976720 0.0005803186 -4.0162919 5.912102e-05
estrg    0.0001678465 1.0001679 0.0004669057  0.3594870 7.192308e-01
```

- **Conclusion**
  - Hormone therapy significantly reduces the risk of relapse or death by $1 - 0.710 = 29\%$ (p = 0.008)

Cox Model - Prediction (I)

- **Predicted survival function**

$$\hat{S}(t \mid z) = \exp\Big\{-\exp(\hat{\beta}^{\mathrm{T}} z)\hat{\Lambda}_0(t)\Big\}$$

- Prepare new data for prediction

```r
# Create new data for prediction
# specify all covariate values
new_data <- data.frame(hormone = 1, meno = 1,
                       age = 45, grade = 2,
                       size = 20, prog = 100,
                       estrg = 100)
new_data
```

```
  hormone meno age grade size prog estrg
1       1    1  45     2   20  100   100
```

Cox Model - Prediction (II)

- **Predict survival probabilities at specified time points**

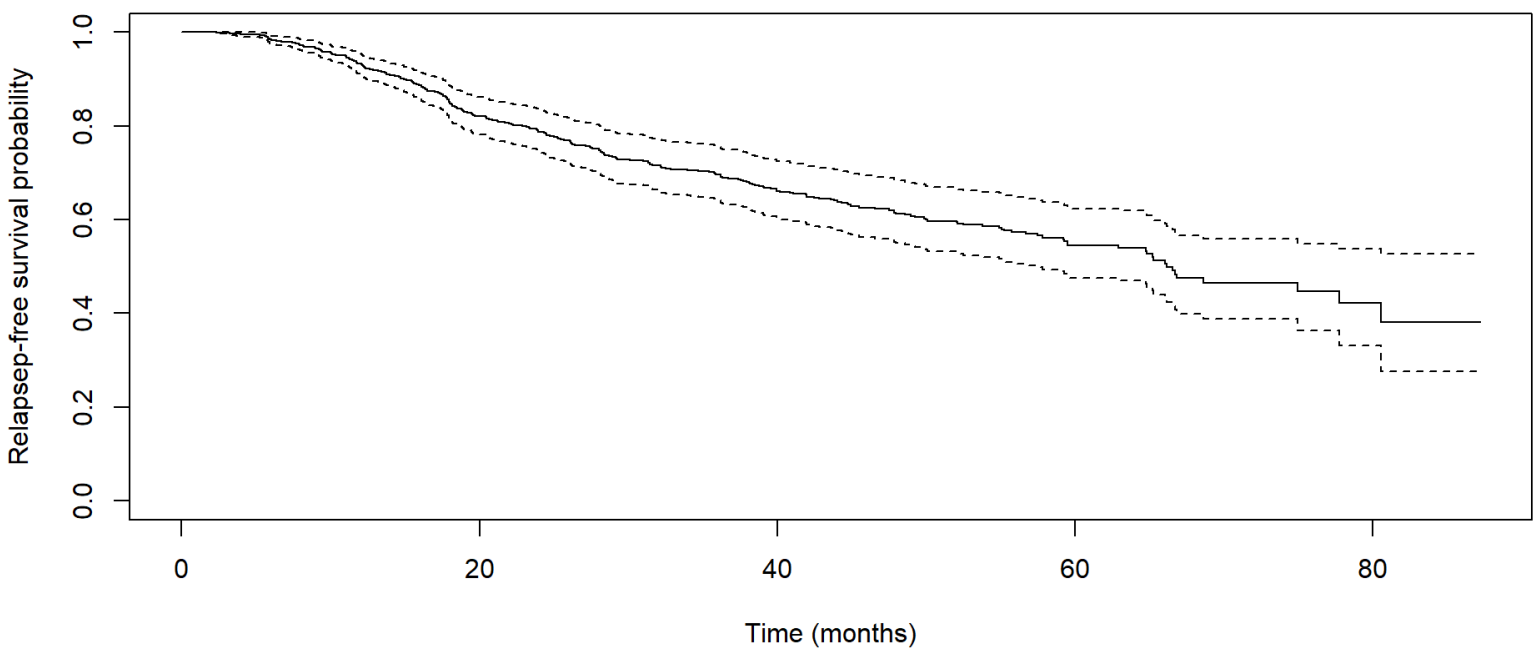
```r
# Predict survival probabilities at 6, 12, 24, 26 months
predicted_survival <- survfit(cox_fit, newdata = new_data[1, ], times = c(6, 12, 24, 36))
summary(predicted_survival, times = c(6, 12, 24, 36))
```

```
Call: survfit(formula = cox_fit, newdata = new_data[1, ], times = c(6,
    12, 24, 36))

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    6    655      13    0.985 0.00441        0.976        0.994
   12    602      43    0.933 0.01059        0.913        0.954
   24    457     111    0.786 0.02304        0.743        0.833
   36    331      57    0.696 0.02925        0.641        0.755
```

Cox Model - Prediction (III)

- **Plot predicted survival function**

```r
# Plot predicted survival function
plot(predicted_survival, ylim = c(0, 1), xlab = "Time (months)",
     ylab = "Relapsep-free survival probability", conf.int = TRUE)
```



Cox Model - Check PH Assumptions (I)

- **Schoenfeld residuals**
  - Difference between observed and expected covariate values at each event time
  - Use `cox.zph()` to test PH assumption

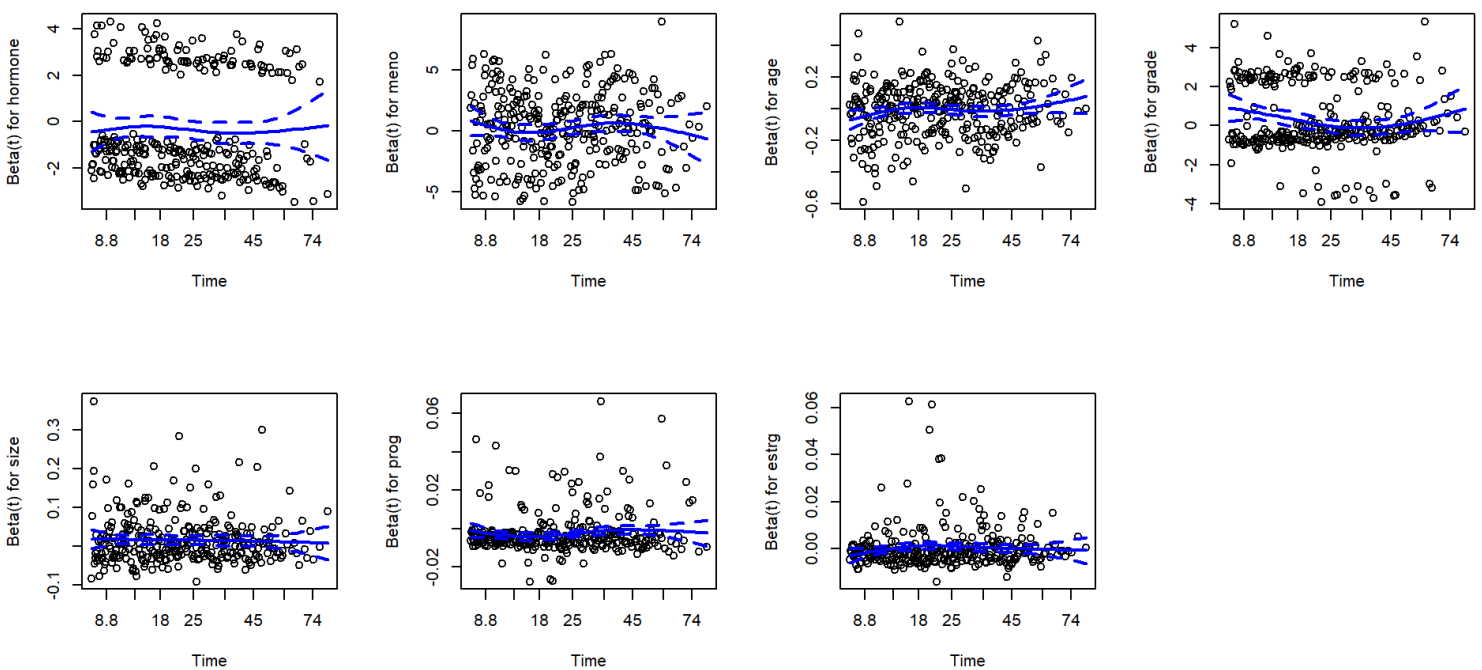
```
ph_test <- cox.zph(cox_fit)
ph_test # Print test results
```

```
         chisq df      p
hormone  0.272  1 0.6017
meno     5.514  1 0.0189
age      9.430  1 0.0021
grade    8.490  1 0.0036
size     0.872  1 0.3505
prog     4.881  1 0.0272
estrg    5.403  1 0.0201
GLOBAL  20.636  7 0.0043
```

Cox Model - Check PH Assumptions (II)

- **Graphical check of PH assumptions**
  - Plot Schoenfeld residuals against time

```r
par(mfrow= c(2, 4)) # Set up 2x4 plotting area for 7 covariates
plot(ph_test, se = TRUE, col = "blue", lwd = 2) # Plot Schoenfeld residuals
```

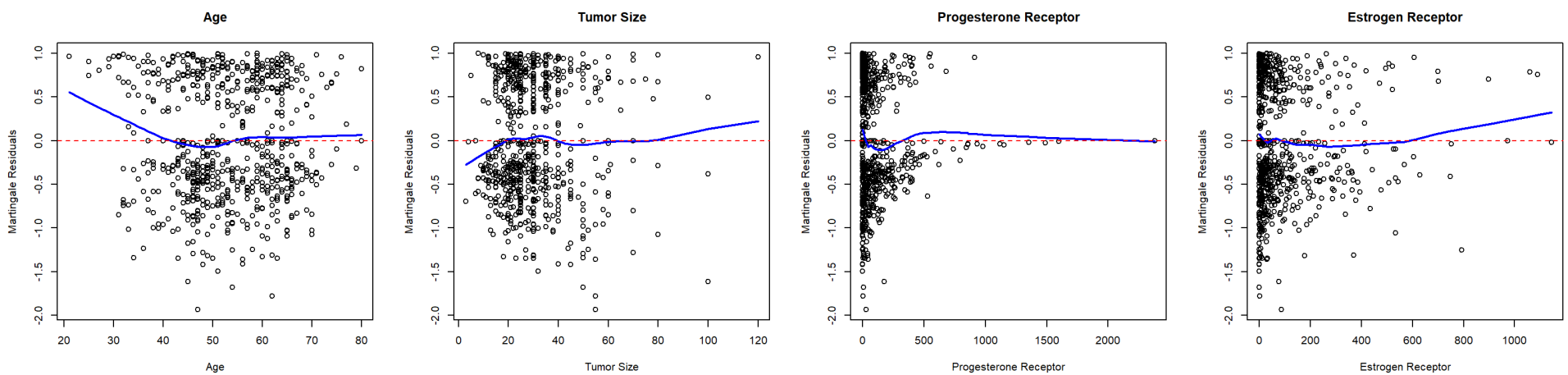


Cox Model - Check Covariate Forms

- **Check linearity of covariate effects**
  - Plot martingale residuals against (quantitative) covariates

```r
# Extract martingale residuals
mart_resid <- residuals(cox_fit, type = 'martingale')
```

▶ Plotting



Coding Exercise

Exercise

Residual analyses show that the proportional hazards assumption is violated for tumor grade, and that the effect of age is not linear.

Fit a different model to address these issues.

▶ Sample solution

Summary

Key Takeaways

- Survival analysis is essential for (often censored) time-to-event data
- Key estimands: survival function, hazard function, cumulative hazard
- Standard analysis tools
  - Kaplan-Meier curves (`survfit()`)
  - Log-rank test (`survdiff()`)
  - Cox proportional hazards model (`coxph()`)

Open Questions

- Efficient/effective presentation of survival probabilities
  - Point estimates, confidence intervals
- Customizable survival curves
  - Add at risk table below graph
- Presentation of regression results
  - Hazard ratios, confidence intervals, p-values
  - Visualize regression results (e.g., forest plots)

Tidy Survival Analysis: Applying R's Tidyverse to Survival Data

Module 2. Data Manipulation with Tidyverse

Lu Mao

lmao@biostat.wisc.edu

Department of Biostatistics & Medical Informatics

University of Wisconsin-Madison

Aug 3, 2025

Table of contents

Overview of Tidyverse

The `tidyverse` Ecosystem

- **Motivation**: tidy data for reproducible analysis
- **Key packages**
  - `dplyr` (filtering, mutating, grouping, summarizing)
  - `tidyr` (pivoting, nesting, reshaping)
  - `tibble` (modern data frames)
  - `readr` / `haven` (importing .csv or .sas7bdat)
  - `lubridate` (handling time variables)
  - `ggplot2` (visualization)

```
# Load core tidyverse packages
library(tidyverse)
```

Basic Functionalities

- **Data manipulation**: using `dplyr` verbs
  - `mutate()` to create new variables (e.g., age group, log-transformed labs)
  - `filter()` to subset by treatment or age
  - `select()` and `rename()` for variable formatting
  - `arrange()` to sort
  - `group_by()` and `summarize()` for descriptive summaries by arm
- **Data reshaping**: using `tidyr` functions
  - `pivot_longer()` to convert wide to long format
  - `pivot_wider()` to convert long to wide format
  - `nest()` and `unnest()` for hierarchical data

A Simple Example

- **Example dataset**

```r
# Simulated data example
df1 <- tibble(
  id = 1:6,
  trt = c("A", "A", "B", "B", "A", "B"),
  age = c(65, 70, 58, 60, 64, 59),
  time = c(5, 8, 12, 3, 2, 6),
  status = c(1, 0, 1, 1, 0, 0)  # 1 = event, 0 = censored
)
df1
```

```
# A tibble: 6 × 5
     id trt     age  time status
  <int> <chr> <dbl> <dbl>  <dbl>
1     1 A        65     5      1
2     2 A        70     8      0
3     3 B        58    12      1
4     4 B        60     3      1
5     5 A        64     2      0
6     6 B        59     6      0
```

Native Pipe Operator: `|>`

- **What is `|>`**
  - Introduced in **R 4.1** (hot key: `Ctrl + Shift + M`)
  - Passes the result of one expression into the first argument of the next
  - Same idea as `%>%`, but **built into base R**
- **Example**

```
df1 |> # passes tibble data frame df1 to the next function
  mutate(age_group = if_else(age >= 65, "older", "younger")) |> # create age group
  filter(trt == "A") |> # filter for treatment A
  arrange(time) # sort by time
```

```
# A tibble: 3 × 6
     id trt     age  time status age_group
  <int> <chr> <dbl> <dbl>  <dbl> <chr>
1     5 A        64     2      0 younger
2     1 A        65     5      1 older
3     2 A        70     8      0 older
```

Summarizing and Grouping

- Survival-specific summaries (e.g., number of events)
  - `group_by()` and `summarize()` for descriptive summaries by arm

```
df1 |>
  group_by(trt) |> # group by treatment arm
  summarize( # summarize each group
    n = n(), # count number of rows (subjects)
    events = sum(status), # sum of events (status = 1)
    median_time = median(time) # median survival time
  )
```

```
# A tibble: 2 × 4
  trt       n events median_time
  <chr> <int>  <dbl>       <dbl>
1 A         3      1           5
2 B         3      2           6
```

What Does "Tidy" Mean?

> A dataset is tidy if:
>
> - Each **variable** is a column
> - Each **observation** is a row
> - Each **type of observational unit** is a table
>
> — Hadley Wickham, *Tidy Data* (2014)
> https://www.jstatsoft.org/article/view/v059i10

Why Tidy Data?

- **Tidy data principles**
  - Easy to reshape and transform
  - Compatible with `ggplot2`, `dplyr`, `tidyr`, and modeling tools
  - Encourages modular and reproducible code
- **Messy data challenges:**
  - Time in rows, covariates in columns
  - Multiple data types in one column
  - Separte randomization and event/censoring dates
  - Missing/censored values inconsistently coded

Tidy Survival Data

- **Possible pre-processing steps**
  - Calculate survival time from start to event/censoring
  - Creating the (X, δ) structure expected by `Surv()`
  - Reshaping data to long format in case of multiple events
- **An Example**

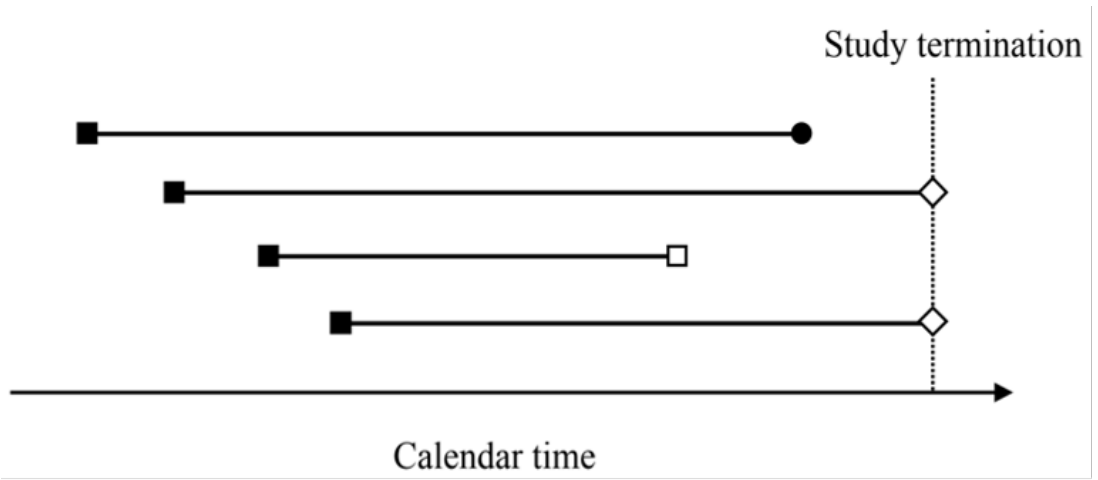
```r
# Load GBC data
gbc <- read.table("data/gbc.txt", header = TRUE)
head(gbc)
```

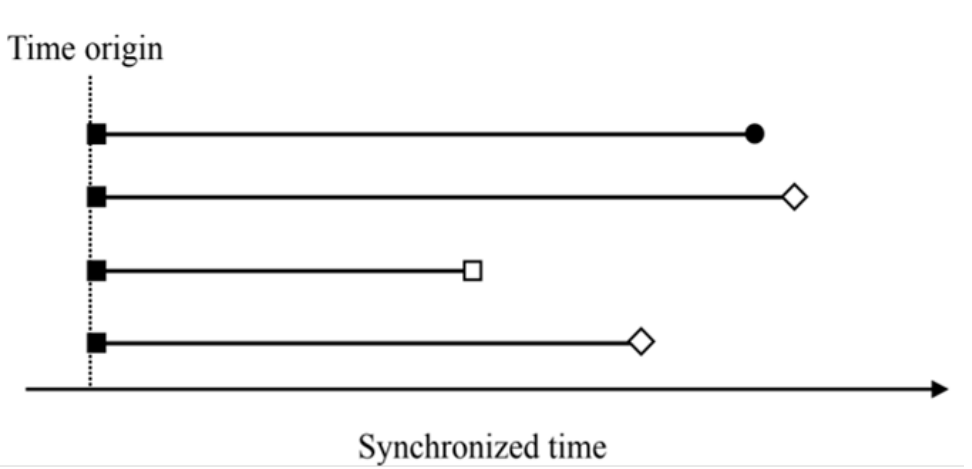
```
  id     time status hormone age meno size grade nodes prog estrg
1  1 43.83607      1       1  38    1   18     3     5  141   105
2  1 74.81967      0       1  38    1   18     3     5  141   105
3  2 46.55738      1       1  52    1   20     1     1   78    14
4  2 65.77049      0       1  52    1   20     1     1   78    14
5  3 41.93443      1       1  47    1   30     2     1  422    89
6  3 47.73770      2       1  47    1   30     2     1  422    89
```

Tidying Survival Data

Calendar vs. Event Times

- **Time from start to event/censoring ()**





■ Study entry (randomization)
◇ Administrative censoring
● Outcome event
□ Loss to follow-up

Dates to Time Difference

- **A data example**

```r
# Example: raw dates as character strings
df2 <- tibble(
  id = 1:3,
  rand_date = c("2022-01-01", "2022-01-15", "2022-01-20"),
  end_date = c("2022-04-01", "2022-06-01", "2022-03-15"),
  status = c("dead", "censored", "dead")
)
df2
```

```
# A tibble: 3 × 4
     id rand_date  end_date   status
  <int> <chr>      <chr>      <chr>
1     1 2022-01-01 2022-04-01 dead
2     2 2022-01-15 2022-06-01 censored
3     3 2022-01-20 2022-03-15 dead
```

Parsing Dates and Calculating Time

- **Using `lubridate` to parse dates**
  - `ymd()` for "year-month-day" format
  - `mdy()` for "month-day-year" format

```
# Parse dates and calculate time/status
df2 |>
  mutate(
    rand_date = ymd(rand_date), # convert character to Date
    end_date = ymd(end_date), # convert character to Date
    time = as.numeric(end_date - rand_date), # calculate time in days
    status = if_else(status == "dead", 1, 0) # convert status to 1/0
  )
```

```
# A tibble: 3 × 5
     id rand_date  end_date   status  time
  <int> <date>     <date>      <dbl> <dbl>
1     1 2022-01-01 2022-04-01      1    90
2     2 2022-01-15 2022-06-01      0   137
3     3 2022-01-20 2022-03-15      1    54
```

Exercise: Calculate Survival Time (I)

- Calculate `time` and `status` variables for `df3`:

```r
# create a df3 with dates in the form of month-day-year
df3 <- tibble(
  id = 1:3,
  rand_date = c("Jan-01-2022", "01-15-2022", "01-20-2022"),
  end_date = c("04-01-2022", "Jun-01-2022", "03-15-2022"),
  status = c("dead", "censored", "dead")
)
df3
```

```
# A tibble: 3 × 4
     id rand_date   end_date    status
  <int> <chr>       <chr>       <chr>
1     1 Jan-01-2022 04-01-2022  dead
2     2 01-15-2022  Jun-01-2022 censored
3     3 01-20-2022  03-15-2022  dead
```

Exercise: Calculate Survival Time (II)

- **Hint**: use `mdy()` to parse dates

▶ Solution

- More about manipulating dates
  - lubridate official documentation
  - R for Data Science: Dates and times

Parsing Censored Observations

- **Alternative formats for censored times**
  - `"32+"`, `">17"`, etc
  - `parse_number()` for get `time`; `str_detect()` for `status`

```
# Example data: relapse times with "+" indicating censoring
MP <- c(10,  "32+", 23, "25+")
# Convert to (time, status) format
df4 <- tibble(
  MP = MP,                                # Original data
  time = parse_number(MP),                # Extract numeric part
  status = 1 - str_detect(MP, "\\+")      # Censored if "+" detected
)
df4
```

```
# A tibble: 4 × 3
  MP     time status
  <chr> <dbl>  <dbl>
1 10       10      1
2 32+      32      0
3 23       23      1
4 25+      25      0
```

Exercise: Parse Censored Times

- **Task**: Parse `MP` in `df5` to create `time` and `status`

```
df5 <- tibble(
  MP = c(10, "32+", 23, ">25")
)
```

▶ Solution

- More on string operation
  - stringr official documentation
  - R for Data Science: Strings

Reshaping Data

- **Why reshape?**
  - Multiple events per subject
  - Wide format (multiple columns) long format (one row per event)

```r
# Example: wide format with multiple events
df6 <- tibble(
  id = 1:3,
  prog_time = c(10, 20, 30),
  prog_status = c(1, 0, 1), # 1 = progression, 0 = censored
  death_time = c(15, 20, 35),
  death_status = c(0, 1, 1) # 1 = dead, 0 = censored
)
# 1: progression at 10, censored at 15
# 2: dead at 20 without progression
# 3: progression at 30, dead at 35
df6
```

```
# A tibble: 3 × 5
     id prog_time prog_status death_time death_status
  <int>     <dbl>       <dbl>      <dbl>        <dbl>
1     1        10           1         15            0
2     2        20           0         20            1
3     3        30           1         35            1
```

Wide to Long

- **Using `pivot_longer()`**
  - Convert wide format to long format
  - Specify `names_to` and `values_to` for new columns

```r
df7 <- df6 |>
  pivot_longer(
    cols = c(prog_time, prog_status, death_time, death_status), # columns to reshape
    names_to = c("event", ".value"), # .value keeps the variable name, event is the new column
    names_pattern = "(.*)_(.*)" # split by underscore
  )
df7
```

```
# A tibble: 6 × 4
     id event  time status
  <int> <chr> <dbl>  <dbl>
1     1 prog     10      1
2     1 death    15      0
3     2 prog     20      0
4     2 death    20      1
5     3 prog     30      1
6     3 death    35      1
```

Exercise: Clean Up

- **Task**: Clean up `df7` to create a tidy survival dataset
  - Remove rows with `event = prog` and `status = 0` (non-terminal event)
  - Recode `status = 2` for death events

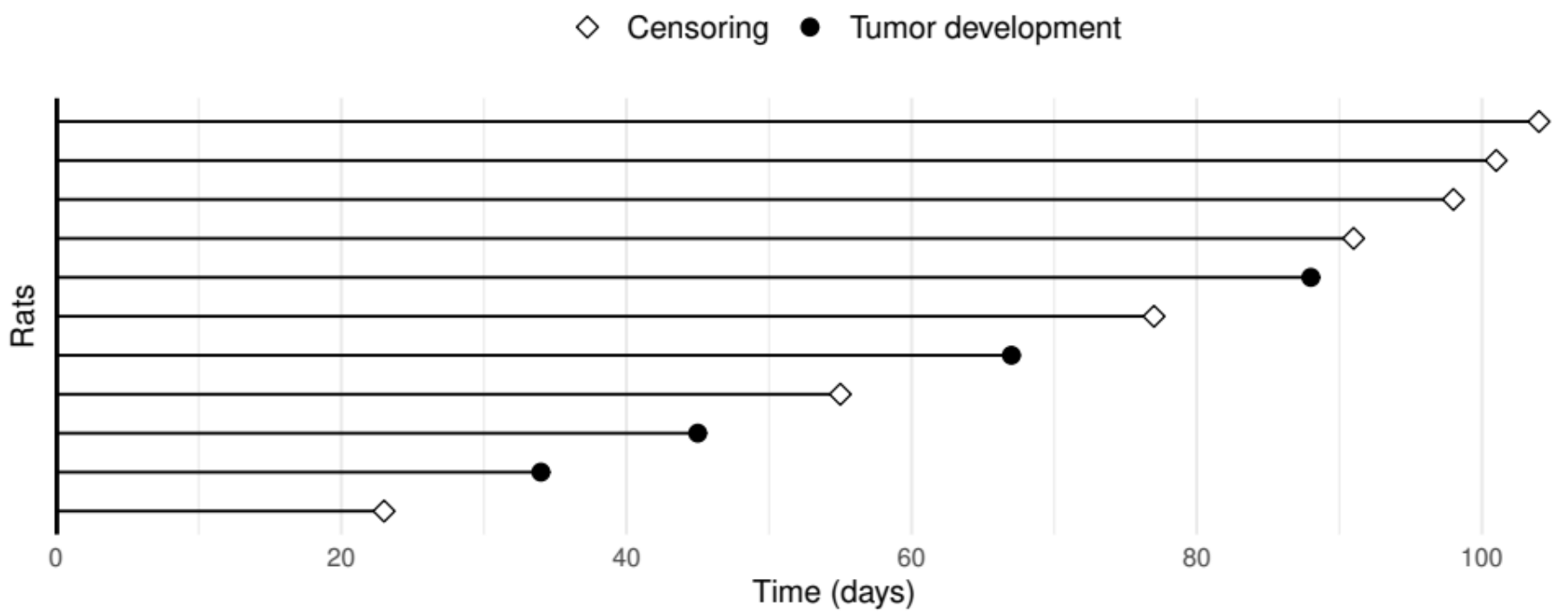
▶ Solution

- More on reshaping data
  - tidyr official documentation
  - R for Data Science: Data tidying

Visualizing Subject Follow-Up

Swimmer Plot

- **What is a swimmer plot?**
  - Visualizes subject follow-up
  - Each row represents a subject
  - Horizontal lines show time to event/censoring



Swimmer Plot Basics

- **Using `ggplot2`**
  - `geom_linerange()` for horizontal lines
  - `geom_point()` for events
  - `facet_wrap()` for treatment arms (optional)
- **A data example**

```
# Example data: rat survival times
df8 <- tibble(
  time = c(101, 55, 67, 23, 45, 98, 34, 77, 91, 104, 88),
  status = c(0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1),
  group = c("A", "A", "A", "B", "B", "B", "A", "B", "B", "A", "B")
) |>
  mutate(
    id = row_number(), # create id column using row number
    .before = 1 # place id before time
  )
```

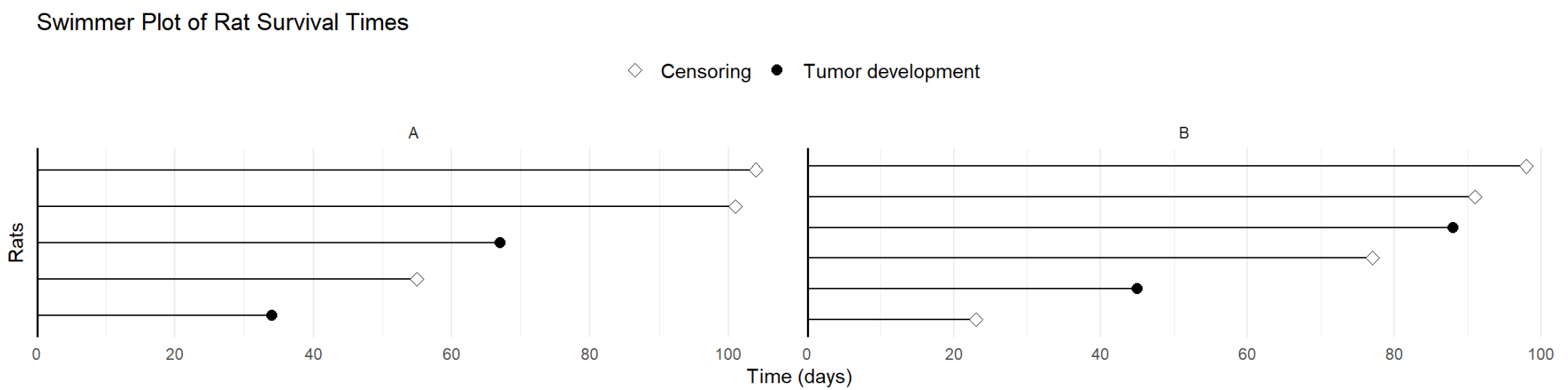
Creating a Swimmer Plot

- Code to reproduce previous plot

```r
# Specify the plot
fig8 <- df8 |>
  # Set-up: id on the y-axis, time on the x-axis
  ggplot(aes(x = time, y = reorder(id, time))) + # reorder id by time
  # Add geometric objects
  geom_linerange(aes(xmin = 0, xmax = time)) + # horizontal lines from 0 to time
  # Add points for events/censoring, distinguish by status
  geom_point(aes(shape = factor(status)), size = 2.5, fill = "white") +
  # Add vertical line at x = 0
  geom_vline(xintercept = 0, linewidth = 1) +
  theme_minimal() + # use minimal theme
  # Format y axis
  scale_y_discrete(name = "Rats") + # y-axis label
  # Format x axis (label, breaks, no expansion on left, 0.05 expansion on right)
  scale_x_continuous(name = "Time (days)", breaks = seq(0, 100, by = 20),
                     expand = expansion(c(0, 0.05))) +
  # Format point shape (pch = 23 for censoring, pch = 19 for event; label shape)
  scale_shape_manual(values = c(23, 19), labels = c("Censoring", "Tumor development")) +
  # Further formatting using theme()
  theme(
    legend.position = "top", # place legend at the top
    legend.title = element_blank(), # no legend title
```

Exercise: Swimmer Plot by Group

- **Task**: Create a swimmer plot for `df8` by `group`
  - Use `facet_wrap()` to create separate panels for each group
  - Add a title “Swimmer Plot of Rat Survival Times”



▶ Solution

Creating “Table 1”

Descriptive Statistics

- **Importance of Table 1**
  - Summarizes baseline characteristics
  - Provides context for formal analysis
- **Using `gtsummary`**
  - `tbl_summary()` for descriptive statistics
  - `add_p()` for p-values comparing groups (not recommended for randomized trials)
  - `add_overall` to add overall summary
  - `modify_header()` to customize table headers

Basic Syntax of `tbl_summary()`

- **Common arguments**
  - `by = "group"` to summarize by group
  - `include = c("variable1", "variable2")` to include specific variables
  - `label = list(variable = "Label")` to customize variable labels
  - `statistic = list(variable ~ "statistic")` to specify statistics
    - `statistic = list(all_continuous() ~ "{mean} ({sd})")` for mean and SD
  - `digits = list(variable ~ 2)` to set decimal places

A Simple Example

- **Example dataset**

```r
# Example data: 10 subjects with treatment, age, and sex
df9 <- tibble(
  id = 1:10,
  time = c(101, 55, 67, 23, 45, 98, 34, 77, 91, 104),
  status = c(0, 1, 1, 0, 1, 0, 1, 0, 1, 0), # 0 = censored, 1 = event
  trt = c("A", "A", "B", "B", "A", "B", "A", "B", "A", "B"),
  sex = c("M", "F", "M", "F", "M", "F", "M", "F", "M", "F"),
  age = c(65, 70, 58, 60, 64, 59, 66, 62, 68, 61)
)
head(df9)
```

```
# A tibble: 6 × 6
     id  time status trt   sex     age
  <int> <dbl>  <dbl> <chr> <chr> <dbl>
1     1   101      0 A     M        65
2     2    55      1 A     F        70
3     3    67      1 B     M        58
4     4    23      0 B     F        60
5     5    45      1 A     M        64
6     6    98      0 B     F        59
```

Creating a Summary Table

```
library(gtsummary) # load package
df9 |>
  tbl_summary(
    by = trt,                                # summarize by treatment arm
    include = c(sex, age, time, status), # include specific variables
    label = list(                            # label variables
      time = "Follow-up time (months)",
      status = "Events"
    )
  )
```

Characteristic	**A**, N = 5[1]	**B**, N = 5[1]
sex		
F	1 (20%)	4 (80%)
M	4 (80%)	1 (20%)
age	66.0 (65.0, 68.0)	60.0 (59.0, 61.0)
Follow-up time (months)	55 (45, 91)	77 (67, 98)
Events	4 (80%)	1 (20%)

[1] n (%); Median (IQR)

Exercise: Summarize GBC Data (I)

- **Task**: Summarize the GBC mortality data (`gbc_mort.txt`) like below

Characteristic	**Hormone**, N = 246[1]	**No Hormone**, N = 440[1]	**Overall**, N = 686[1]
Follow-up time (months)	48 (29, 61)	41 (25, 57)	44 (26, 60)
Death	56 (23%)	115 (26%)	171 (25%)
Age (years)	58 (50, 63)	50 (45, 59)	53 (46, 61)
Menopausal status	187 (76%)	209 (48%)	396 (58%)
Tumor size (mm)	25 (20, 35)	25 (20, 35)	25 (20, 35)
Tumor grade			
1	33 (13%)	48 (11%)	81 (12%)
2	163 (66%)	281 (64%)	444 (65%)
3	50 (20%)	111 (25%)	161 (23%)
Number of nodes	3 (1, 7)	3 (1, 7)	3 (1, 7)
Progesterone (fmol/mg)	35 (7, 133)	32 (7, 130)	33 (7, 132)
Estrogen (fmol/mg)	46 (9, 183)	32 (8, 92)	36 (8, 114)

[1] Median (IQR); n (%)

Exercise: Summarize GBC Data (II)

- **Points to note**
  - Summarize by hormone therapy (`hormone`)
  - Include variables: `time`, `status`, `age`, `meno`, `size`, `grade`, `nodes`, `prog`, `estrg`
  - Label variables appropriately
  - Add overall summary column at the end

Exercise: Summarize GBC Data (III)

- Solution

Exercise: Summarize GBC Data (IV)

- **Task**: summarize *relapse* and death data from `gbc.txt`
  - Hint: `group_by(id)` and `summarize()`

Characteristic	**Hormone**, N = 246[1]	**No Hormone**, N = 440[1]	**Overall**, N = 686[1]
Relapse	94 (38%)	205 (47%)	299 (44%)
Death	56 (23%)	115 (26%)	171 (25%)
Composite	94 (38%)	205 (47%)	299 (44%)
Relapse then death	56 (23%)	115 (26%)	171 (25%)

[1] n (%)

Exercise: Summarize GBC Data (V)

- ▶ Solution

Summary

Key Takeaways

- **Tidyverse** provides powerful tools for data manipulation and visualization
- **Tidy data** principles simplify analysis and visualization
- **Survival data** may require pre-processing steps (`dplyr`, `tidyr`,`lubridate`)
- **Swimmer plots** effectively visualize subject follow-up (`ggplot2`)
- **Descriptive statistics** can be easily summarized using `gtsummary::tbl_summary()`

Next Steps

- Format analysis results from the `survival` package:
  - **Nonparametric estimates** with `survfit()`
  - **Regression models** with `coxph()`
- Explore advanced visualization techniques:
  - Kaplan–Meier curves with `ggsurvfit` or `survminer`
  - Layered plots using `ggplot2`
  - Annotated plots for publications

Tidy Survival Analysis: Applying R's Tidyverse to Survival Data

Module 3. Nonparametric Survival Analysis

Lu Mao

lmao@biostat.wisc.edu

Department of Biostatistics & Medical Informatics

University of Wisconsin-Madison

Aug 3, 2025

Table of contents

Tabulating Survival Estimates

GBC: Relapse-Free Survival

- Use `dplyr` to get time-to-first event

```
library(tidyverse) # Load tidyverse packages
# Load mortality + relapse data
gbc <- read.table("data/gbc.txt", header = TRUE)
df <- gbc |>  # calculate time to first event (relapse or death)
  group_by(id) |> # group by id
  arrange(time) |> # sort rows by time
  slice(1) |>      # get the first row within each id
  ungroup()     # remove grouping
# Display the first few rows of the data
head(df)
```

```
# A tibble: 6 × 11
     id  time status hormone   age  meno  size grade nodes  prog estrg
  <int> <dbl>  <int>   <int> <int> <int> <int> <int> <int> <int> <int>
1     1 43.8       1       1    38     1    18     3     5   141   105
2     2 46.6       1       1    52     1    20     1     1    78    14
3     3 41.9       1       1    47     1    30     2     1   422    89
4     4  4.85      0       1    40     1    24     1     3    25    11
5     5 61.1       0       2    64     2    19     2     1    19     9
6     6 63.4       0       2    49     2    56     1     3   356    64
```

Raw Output from `survfit()`

- **KM estimates by hormone therapy**

```r
library(survival) # Load survival package
# Fit KM estimates by hormone group
km_fit <- survfit(Surv(time, status > 0) ~ hormone, data = df)
# summarize the KM fit object
summary(km_fit, times = c(6, 12, 24, 36))
```

```
Call: survfit(formula = Surv(time, status > 0) ~ hormone, data = df)

                hormone=1
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    6    419       9    0.979 0.00691        0.966        0.993
   12    379      35    0.897 0.01476        0.868        0.926
   24    280      73    0.720 0.02203        0.678        0.764
   36    195      41    0.606 0.02475        0.559        0.656

                hormone=2
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    6    236       4    0.983 0.00826        0.967        1.000
   12    223       8    0.950 0.01418        0.922        0.978
   24    177      38    0.785 0.02701        0.733        0.839
   36    136      16    0.708 0.03047        0.650        0.770
```

Extracting Survival Estimates

- **Elements in `survfit` object**
  - `time`: time points of the survival estimates
  - `surv`: survival probabilities at the time points
  - `lower, upper`: confidence intervals for the survival estimates
  - `strata`: stratification information (if applicable)

> **Exercise**
>
> Create a table of survival estimates with 95% confidence intervals at 6, 12, 24, and 36 months for each hormone therapy group using `dplyr` and `tibble`.

Tidying `survfit()` Output

- **Use `broom` package to tidy `survfit` objects**
  - `broom::tidy()` converts the `survfit` object into a tidy data frame
  - Useful for further analysis or visualization

```
1 library(broom) # Load broom package
2 tidy(km_fit) # Tidy the KM fit object
```

```
# A tibble: 613 × 9
    time n.risk n.event n.censor estimate std.error conf.high conf.low strata
   <dbl>  <dbl>   <dbl>    <dbl>    <dbl>     <dbl>     <dbl>    <dbl> <chr>
 1 0.262    440       0        1    1       0                 1    1     hormone=1
 2 0.525    439       0        1    1       0                 1    1     hormone=1
 3 0.557    438       0        2    1       0                 1    1     hormone=1
 4 0.590    436       0        1    1       0                 1    1     hormone=1
 5 0.951    435       0        1    1       0                 1    1     hormone=1
 6 1.87     434       0        1    1       0                 1    1     hormone=1
 7 2.13     433       0        1    1       0                 1    1     hormone=1
 8 2.20     432       0        1    1       0                 1    1     hormone=1
 9 2.33     431       0        1    1       0                 1    1     hormone=1
10 2.36     430       1        0    0.998   0.00233           1    0.993 hormone=1
# ℹ 603 more rows
```

Tabulation with `gtsummary`

- **Main function: `tbl_survfit()`**
  - Takes on `survfit` object
  - Creates a table of survival estimates with confidence intervals
  - Automatically handles stratification and time points

```
library(gtsummary) # Load gtsummary package
# Create a table of survival estimates
km_fit |> tbl_survfit(                    # Pass `survfit` object
             label = "Hormone",           # Row label: "Hormone"
             times = c(6, 12, 24, 36),     # Time points for estimates
             label_header = "Month {time}" # Column label: "Month xx"
             )
```

Characteristic	Month 6	Month 12	Month 24	Month 36
Hormone				
1	98% (97%, 99%)	90% (87%, 93%)	72% (68%, 76%)	61% (56%, 66%)
2	98% (97%, 100%)	95% (92%, 98%)	78% (73%, 84%)	71% (65%, 77%)

Grouping by Multiple Variables

- **Pass raw data to `tbl_survfit()`**

```
df |>                                      # Use raw data
  tbl_survfit(y = Surv(time, status),      # Survival object
              include = c(meno, grade),    # Include variables: menopause, grade
              label = list(meno = "Menopause", # Row labels
                           grade = "Tumor grade"),
              times = c(6, 12, 24, 36),    # Time points for estimates
              label_header = "Month {time}" # Column label: "Month xx"
             )
```

Characteristic	Month 6	Month 12	Month 24	Month 36
Menopause				
1	98% (96%, 99%)	90% (86%, 93%)	73% (68%, 78%)	65% (59%, 71%)
2	98% (97%, 100%)	93% (90%, 96%)	75% (71%, 80%)	64% (59%, 69%)
Tumor grade				
1	100% (100%, 100%)	100% (100%, 100%)	93% (87%, 99%)	84% (75%, 93%)
2	98% (97%, 100%)	92% (90%, 95%)	75% (71%, 80%)	64% (60%, 69%)
3	96% (93%, 99%)	85% (80%, 91%)	63% (55%, 71%)	55% (48%, 64%)

Tabulating Quantile Estimates

- **Quantile estimates**: Median survival time, quartiles, etc.
  - Specify `probs` argument in `tbl_survfit()`

```
# Create a table of quantile estimates
km_fit |>                          # Pass `survfit` object
  tbl_survfit(
    label = "Hormone",             # Row label: "Hormone"
    probs = c(0.25, 0.5, 0.75),  # Quantiles: 25%, 50%, 75%
              label_header = "{100 * prob}% quantile") # Column label: "xx quantile"
```

Characteristic	25% quantile	50% quantile	75% quantile
Hormone			
1	21 (18, 25)	50 (42, 59)	81 (81, —)
2	28 (23, 39)	66 (63, —)	— (—, —)

Exercise: Tabulating Quantiles

- **Create the following table**

Characteristic	25% quantile	50% quantile	75% quantile
Menopause			
1	21 (18, 27)	66 (52, —)	— (—, —)
2	24 (21, 28)	56 (49, 65)	— (81, —)
Tumor grade			
1	48 (38, —)	— (65, —)	— (—, —)
2	24 (21, 28)	57 (49, 67)	— (81, —)
3	16 (13, 19)	44 (31, —)	— (67, —)

▶ Solution

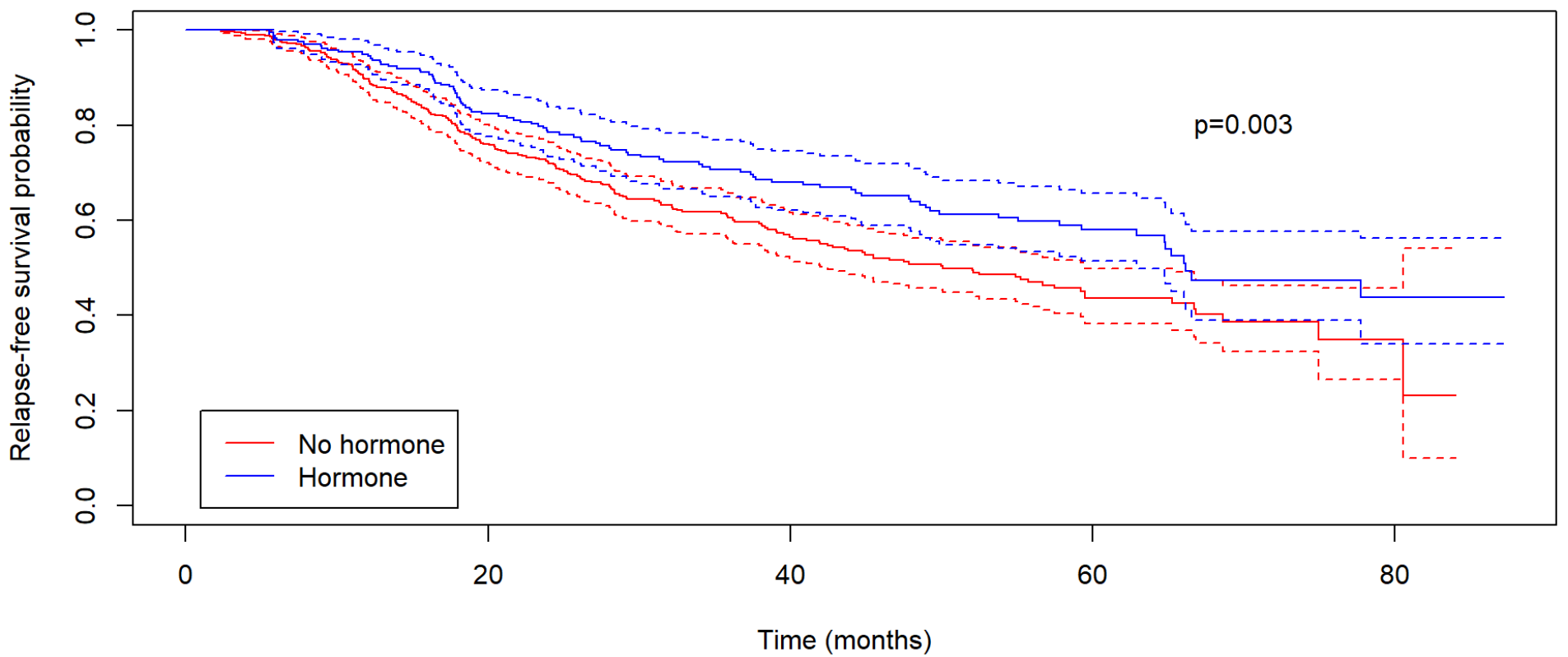
Customizing the Table

- **Customize table appearance**
  - `label_header`: change column names
  - `label`: change row labels
  - `statistic`: customize statistics displayed
    - `statistic = "{estimate} ({conf.low}, {conf.high})"` for confidence intervals
- More about `gtsummary`
  - gtsummary website
  - tbl_survfit documentation

Visualizing Kaplan-Meier Curves

Base Plot

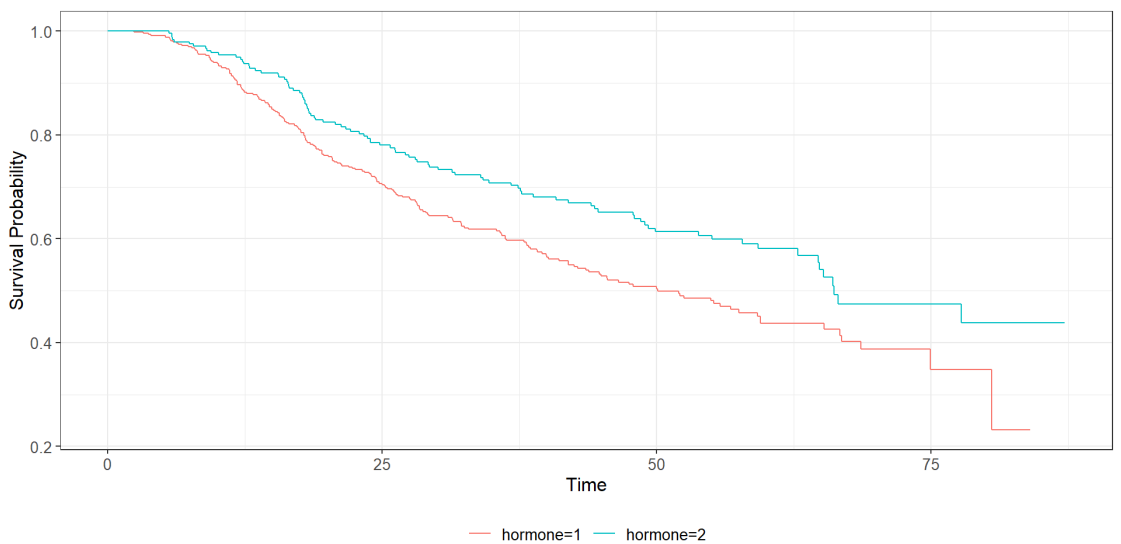
- Plot KM curves by hormone group



Enhanced Graphics with `ggsurvfit`

- `ggsurvfit`: Provides a `ggplot2` interface for survival curves
  - Takes on `survfit` object or raw data
  - `add_risktable()` adds a risk table below graph
  - Allows for more customization and aesthetics

```
1 library(ggsurvfit) # Load ggsurvfit package)
2 km_fit |> ggsurvfit()  # Pass `survfit` object
```



Customization (I)

- **Customize the plot with `ggsurvfit`**
  - `add_risktable()`: Adds a risk table below the survival curve
  - `add_confidence_interval()`: Adds confidence intervals to the survival curve
  - `add_pvalue()`: Adds p-value for log-rank test
  - Other `ggplot2` functions to further customize the plot
    - `scale_x_continuous()`, `scale_y_continuous()`, `theme()`, etc.

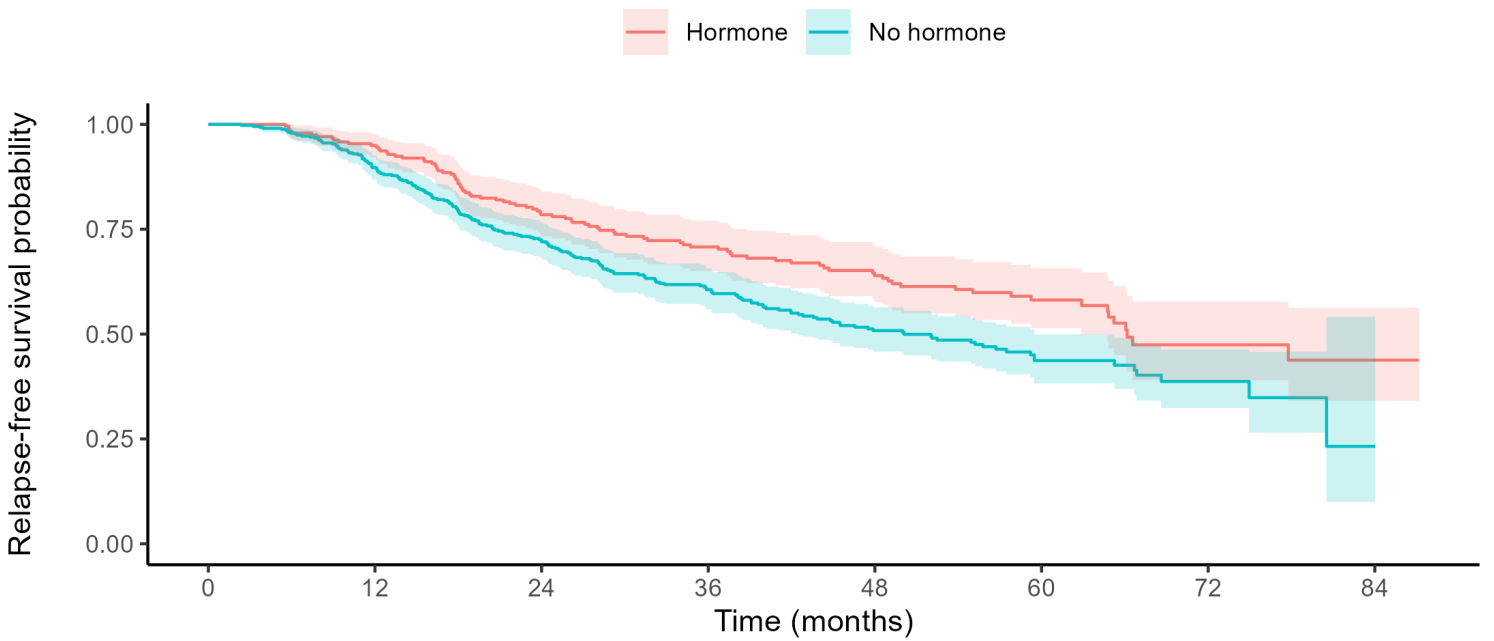
Customization (II)

- Code

```r
# survfit2() fits better with `ggsurvfit`
km_fit2 <- survfit2(Surv(time, status > 0) ~ hormone,
                    data = df |> # Relabel hormone variable
                      mutate(hormone = if_else(hormone == 1, "No hormone", "Hormone"))
                      )
km_fig <- km_fit2 |>     # Plot KM curves with customization
  ggsurvfit() +          # Pass `survfit2` object
  add_risktable() +      # Add risk table below the graph
  add_confidence_interval() +   # Add confidence intervals
  add_pvalue(caption = "Log-rank {p.value}") +  # Add p-value for log-rank test
  scale_x_continuous("Time (months)", breaks = seq(0, 84, 12)) + # x-axis format
  scale_y_continuous("Relapse-free survival probability", limits = c(0, 1)) + # y-axis format
  theme_classic() + # Use classic theme for this ggplot
  theme(legend.position = "top") # Position legend at the top

ggsave("images/km_fig.png", km_fig, width = 7.5, height = 5) # Save the plot
```

Customization (III)

- **Result**



	Hormone							
At Risk	246	223	177	136	103	58	17	2
Events	0	12	50	66	78	86	93	94
	No hormone							
At Risk	440	379	280	195	124	63	17	1
Events	0	44	117	158	186	199	203	205

Risk Table Exercise

- **Task**: Display only numbers at risk in the risk table
  - Hint: Add `risktable_stats = "n.risk"` argument in `add_risktable()`

▶ Solution

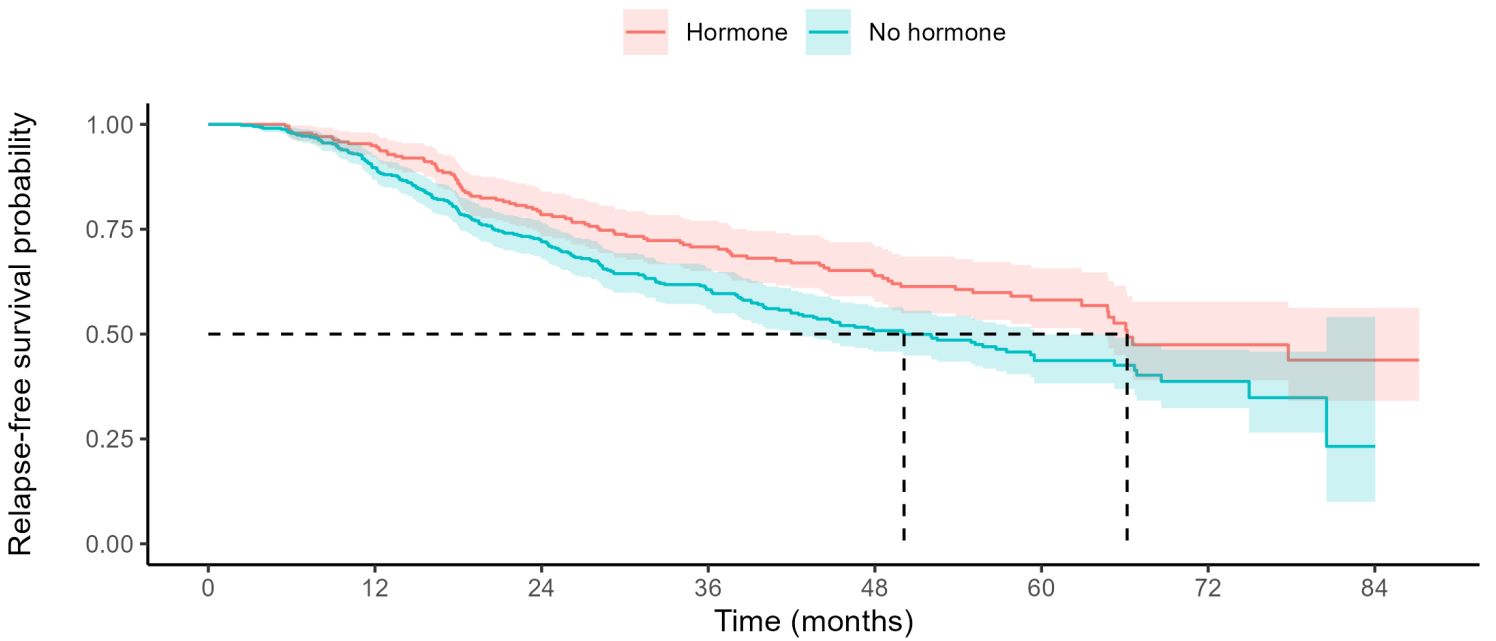
Add Quantiles (I)

- **Add quantile estimates to the plot**
  - Use `add_quantile()` to add median survival time and other quantiles
  - Specify `y_value` or `x_value` arguments for desired quantiles or time points

```
km_fig + add_quantile(
    y_value = 0.5, # Add median survival time
  )
```

Add Quantiles (II)

- **Result**



	Hormone							
At Risk	246	223	177	136	103	58	17	2
Events	0	12	50	66	78	86	93	94
	No hormone							
At Risk	440	379	280	195	124	63	17	1
Events	0	44	117	158	186	199	203	205

Exercise: Add Time Points

- **Task**: Add reference lines at 72 months
  - Use `add_quantile()` with `x_value` argument

▶ Solution

Further Customizations

- **Customize the plot further**
  - `add_risktable_strata_symbol(...)`: Use symbols for strata in the risk table
    - `symbol = NULL`, `size = 15`, `face = "bold"`, etc.
  - `add_censor_mark(...)`: Add censor marks to the survival curve
    - `size = 3`, `shape = 3`, `color = "black"`, etc.
- More about `ggsurvfit`
  - ggsurvfit website
  - Webinar by Daniel D. Sjoberg

Tidy Analysis of Competing Risks

Competing Risks Overview

- **Competing risks**
  - Subject may experience at most one of multiple distinct types of event
  - E.g., death from different causes; relapse vs. death in remission (before relapse)
- **Notation**: (T, Δ)
  - T: time to event
  - Δ: event type indicator (e.g., 1 for relapse, 2 for death)
- **Quantity of interest**
  - cumulative incidence function (CIF), or sub-distribution

$$F_k(t) = P(T \leq t, \Delta = k)$$

    - Cumulative probability of event type k by time t

`tidycmprsk` Package

- **Analysis of CIF** implemented in `cmprsk` package
  - A "tidy" version is available in `tidycmprsk` package
  - Simple interface, plays nicely with `gtsummary` and `ggsurvfit`
  - Input data: `status`: must be a factor, with the first level indicating censoring and subsequent levels the competing risks

```
1 library(tidycmprsk) # Load tidycmprsk package
2 data("trial", package = "tidycmprsk") # Load trial data from tidycmprsk package
3 head(trial) # Display the first few rows of the data
```

```
# A tibble: 6 × 9
  trt      age marker stage grade response death death_cr           ttdeath
  <chr>  <dbl>  <dbl> <fct> <fct>    <int> <int> <fct>                <dbl>
1 Drug A    23  0.16  T1    II           0     0 censor                24
2 Drug B     9  1.11  T2    I            1     0 censor                24
3 Drug A    31  0.277 T1    II           0     0 censor                24
4 Drug A    NA  2.07  T3    III          1     1 death other causes    17.6
5 Drug A    51  2.77  T4    III          1     1 death other causes    16.4
6 Drug B    39  0.613 T4    I            0     1 death from cancer     15.6
```

Nonpametric Inference

- **Gray's estimator and test**

```
# Fit cumulative incidence function (CIF) for competing risks
cif_fit <- cuminc(Surv(ttdeath, death_cr) ~ trt, trial)
cif_fit # print results
#> • Failure type "death from cancer"
#> strata   time   n.risk   estimate   std.error   95% CI
#> Drug A   5.00   97       0.000      0.000       NA, NA
#> Drug A   10.0   94       0.020      0.014       0.004, 0.065
#> Drug A   15.0   83       0.071      0.026       0.031, 0.134
#> Drug A   20.0   61       0.173      0.039       0.106, 0.255
#> Drug B   5.00   102      0.000      0.000       NA, NA
#> Drug B   10.0   95       0.039      0.019       0.013, 0.090
#> Drug B   15.0   75       0.167      0.037       0.102, 0.246
#> Drug B   20.0   55       0.255      0.043       0.175, 0.343
```

Raw Output

- **Raw output from `cuminc()` continued**

```
#> • Failure type "death other causes"
#> strata   time   n.risk   estimate   std.error   95% CI
#> Drug A   5.00   97       0.010      0.010       0.001, 0.050
#> Drug A   10.0   94       0.020      0.014       0.004, 0.065
#> Drug A   15.0   83       0.082      0.028       0.038, 0.147
#> Drug A   20.0   61       0.204      0.041       0.131, 0.289
#> Drug B   5.00   102      0.000      0.000       NA, NA
#> Drug B   10.0   95       0.029      0.017       0.008, 0.077
#> Drug B   15.0   75       0.098      0.030       0.050, 0.165
#> Drug B   20.0   55       0.206      0.040       0.133, 0.289
#>
#> • Tests
#> outcome              statistic   df     p.value
#> death from cancer    1.99        1.00   0.16
#> death other causes   0.089       1.00   0.77
```

Tidy Output in `tibble`

- **Use `broom` to tidy `cuminc` object**
  - Useful for further analysis or visualization

```
1 tidy_cif <- tidy(cif_fit) # Tidy the CIF fit object
2 head(tidy_cif) # Display the first few rows of the tidy data
```

```
# A tibble: 6 × 12
   time outcome      strata estimate std.error conf.low conf.high n.risk n.event
  <dbl> <chr>        <fct>     <dbl>     <dbl>    <dbl>     <dbl>  <int>   <int>
1  0    death from … Drug A   0         0      NA          NA           98       0
2  3.53 death from … Drug A   0         0      NA          NA           98       0
3  5.33 death from … Drug A   0         0      NA          NA           97       0
4  6.32 death from … Drug A   0         0      NA          NA           97       0
5  7.27 death from … Drug A   0.0102    0.0102  8.84e-4     0.0503      97       1
6  7.38 death from … Drug A   0.0204    0.0144  3.90e-3     0.0652      96       1
# ℹ 3 more variables: n.censor <int>, cum.event <int>, cum.censor <int>
```

> **Exercise**
>
> Tabulate CIF estimates with 95% confidence intervals at 5, 10, 15, and 20 months for each risk.

Tabulating CIF Estimates (I)

- **Use `tbl_cuminc()` to create a table of CIF estimates**
  - Similar syntax to `tbl_survfit()`
  - `times`: time points for estimates
  - `outcomes`: specify outcomes to include in the table (Default is the first outcome)

```
# Tabulate CIF estimates with 95% confidence intervals
cif_fit |> # Pass `tidycuminc` object
  tbl_cuminc(
    outcomes = c("death from cancer", "death other causes"), # Specify outcomes
    times = c(10, 15, 20), # Time points for estimates
    label_header = "Month {time}" # Column label: "Month xx"
  )|>
  add_p() # Add p-values from Gray's test
```

Tabulating CIF Estimates (II)

- **Result**

Characteristic	Month 10	Month 15	Month 20	**p-value**[1]
death from cancer				
Chemotherapy Treatment				0.2
Drug A	2.0% (0.39%, 6.5%)	7.1% (3.1%, 13%)	17% (11%, 26%)	
Drug B	3.9% (1.3%, 9.0%)	17% (10%, 25%)	25% (17%, 34%)	
death other causes				
Chemotherapy Treatment				0.8
Drug A	2.0% (0.39%, 6.5%)	8.2% (3.8%, 15%)	20% (13%, 29%)	
Drug B	2.9% (0.79%, 7.7%)	9.8% (5.0%, 17%)	21% (13%, 29%)	

[1] Gray's Test

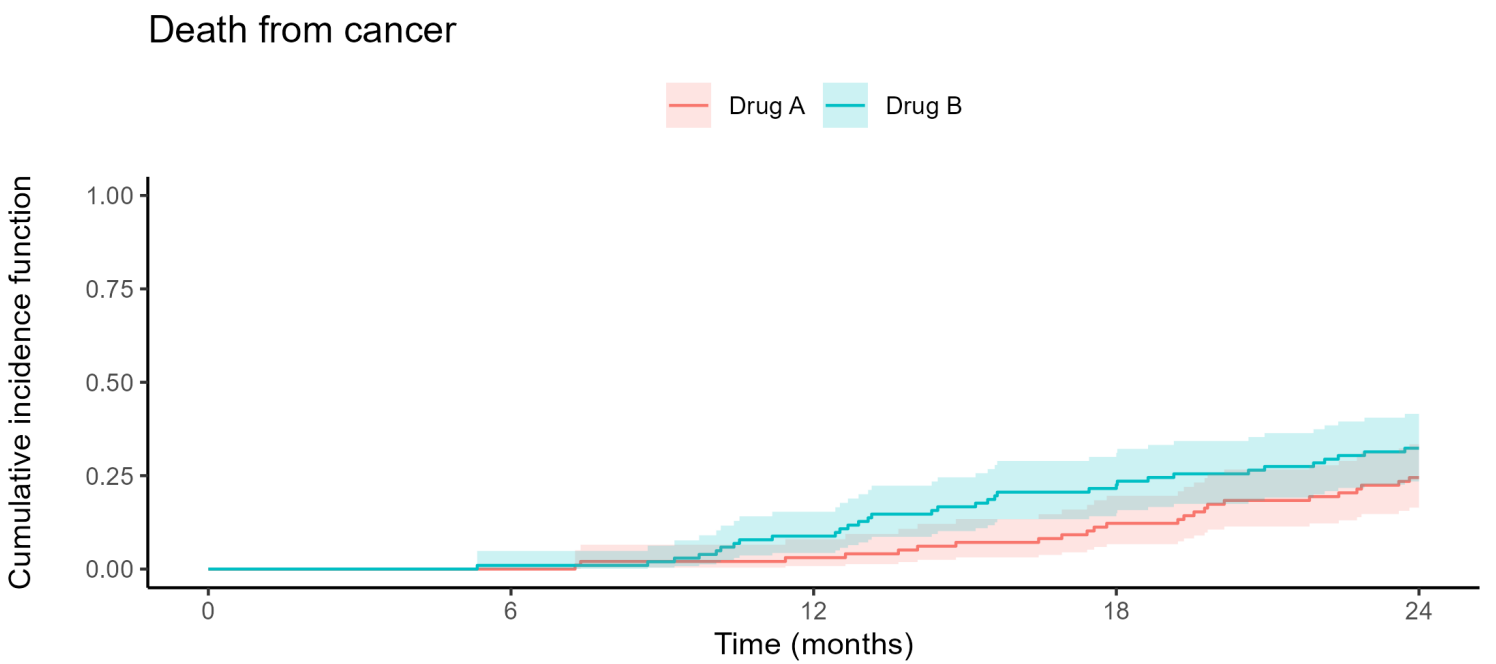
CIF Graphics (I)

- **Plot CIF estimates with `ggsurvfit::ggcuminc()`**
  - Similar syntax to `ggsurvfit()`
  - `outcome`: specify outcome to plot

```
cif_fit |> # Pass `tidycuminc` object
  ggcuminc(outcome = "death from cancer") + # Plot CIF for "death from cancer"
  add_confidence_interval() + # Add confidence intervals
  add_risktable() + # Add risk table below the graph
  add_pvalue(caption = "Gray's test {p.value}") + # Add p-value for Gray's test
  scale_x_continuous("Time (months)", breaks = seq(0, 24, 6)) + # x-axis format
  scale_y_continuous("Cumulative incidence function", limits = c(0, 0.5)) + # y-axis format
  ggtitle("Death from cancer") + # Title
  theme_classic() + # Use classic theme for this ggplot
  theme(legend.position = "top") # Position legend at the top
```

CIF Graphics (II)

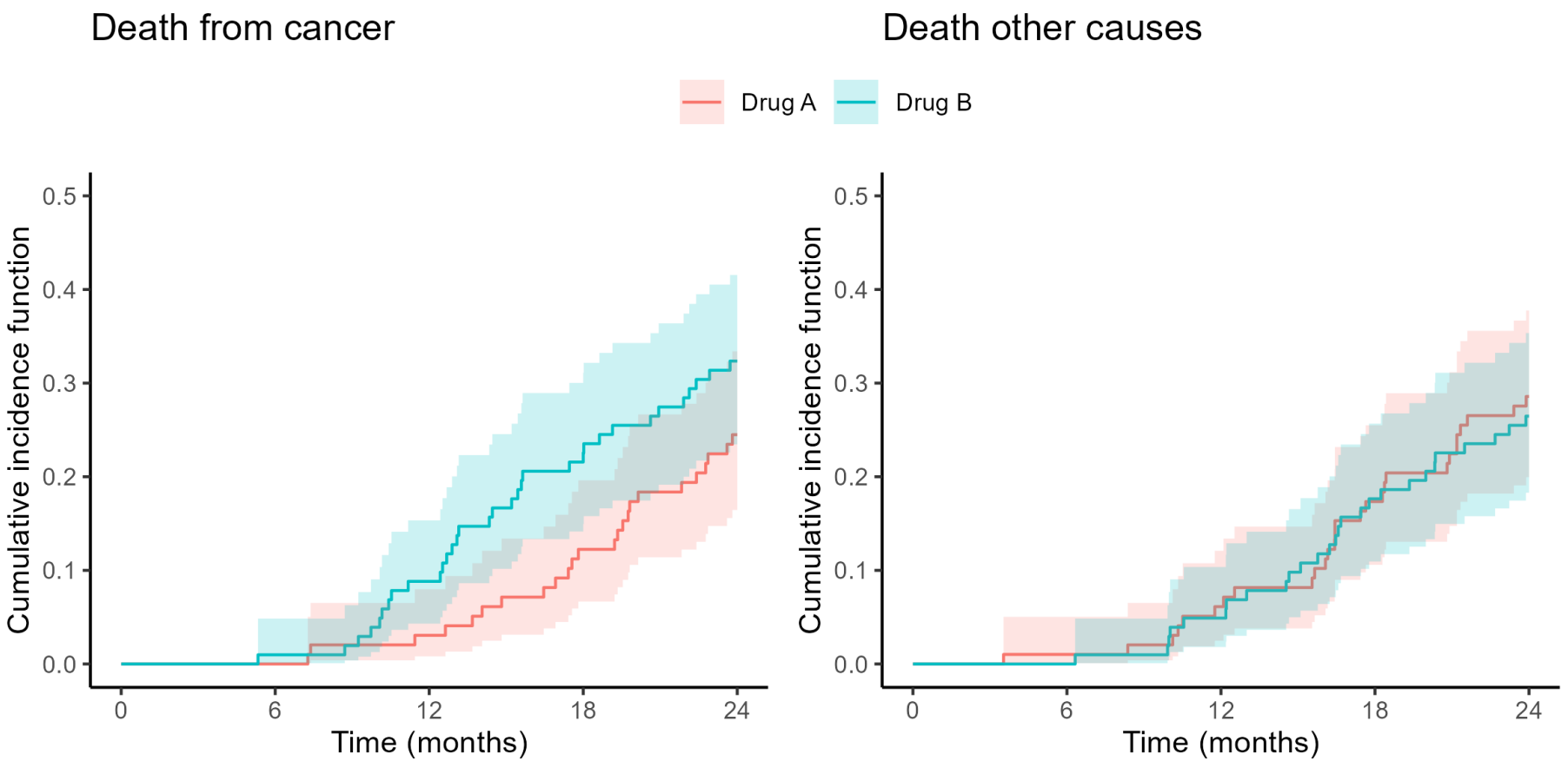
- **Result**



	Drug A				
At Risk	98	97	89	69	46
Events	0	0	3	12	24
	Drug B				
At Risk	102	101	88	62	42
Events	0	1	9	23	33

CIF Graphics Exercise (I)

- **Task**: create the figure below
  - **Hint**: plot separate figures for each outcome and use `patchwork` to combine them



CIF Graphics Exercise (II)

▶ Solution

- More about `tidycmprsk`
  - tidycmprsk website

Summary

Key Takeaways

- **Nonparametric survival analysis**
  - Use `survival::survfit()` for Kaplan-Meier estimates
  - Use `tidycmprsk::cuminc()` for CIF of competing risks
- **Tidy outputs**
  - Use `broom::tidy()` to convert `survfit` and `tidycuminc` objects into tidy data frames
- **Tabulation and visualization**
  - Use `gtsummary::tbl_survfit()` and `tidycumprsk::tbl_cuminc()` for tabulating survival estimates
  - Use `ggsurvfit::ggsurvfit()` and `ggsurvfit::ggcuminc()` for visualizing survival curves and CIF

Next Steps

- **Cox regression analysis**
  - Tidy and format results from `survival::coxph()`
  - Visualize prediction results
- **Competing risks**
  - Proportional sub-distribution hazards (Fine-Gray) regression
  - Tabulation and graphics

Tidy Survival Analysis: Applying R's Tidyverse to Survival Data

Module 4. Semiparametric Regression Analysis

Lu Mao

lmao@biostat.wisc.edu

Department of Biostatistics & Medical Informatics

University of Wisconsin-Madison

Aug 3, 2025

Table of contents

Presenting Regression Results

Cox PH Regression

- **Model specification**

$$\lambda(t \mid Z) = \lambda_0(t) \exp(\beta_1 Z_1 + \beta_2 Z_2 + \ldots + \beta_p Z_p)$$

  - $\lambda_0(t)$: baseline hazard function
  - $\exp(\beta_j)$: hazard ratio for covariate Z_j

- **GBC data: relapse-free survival**

```
1 library(tidyverse) # Load tidyverse packages
2 gbc <- read.table("data/gbc.txt", header = TRUE) # Load GBC dataset
```

GBC Data: a Running Example

- **Reformat the data**

```r
df <- gbc |>  # calculate time to first event (relapse or death)
  group_by(id) |> # group by id
  arrange(time) |> # sort rows by time
  slice(1) |>      # get the first row within each id
  ungroup() |>     # remove grouping
  mutate(
    age40 = ifelse(age >= 40, 1, 0), # create binary variable for age >= 40
    grade = factor(grade), # convert grade to factor
    prog = prog / 100, # rescale progesterone receptor
    estrg = estrg / 100 # rescale estrogen receptor
    )
```

Analysis in Base R

- **Model fitting:** `survival::coxph()`

```r
library(survival) # Load survival package
cox_fit <- coxph(Surv(time, status) ~ hormone + meno + age40 + grade + size + prog + estrg,
                 data = df)
summary(cox_fit) # Print model summary
```

```
Call:
coxph(formula = Surv(time, status) ~ hormone + meno + age40 +
    grade + size + prog + estrg, data = df)

  n= 686, number of events= 299

            coef exp(coef) se(coef)      z Pr(>|z|)
hormone -0.37432   0.68776  0.12917 -2.898 0.003758 **
meno     0.28450   1.32909  0.13973  2.036 0.041748 *
age40   -0.55127   0.57622  0.20243 -2.723 0.006463 **
grade2   0.71547   2.04514  0.24854  2.879 0.003993 **
grade3   0.77465   2.16982  0.26970  2.872 0.004075 **
size     0.01606   1.01619  0.00368  4.365 1.27e-05 ***
prog    -0.22400   0.79932  0.05776 -3.878 0.000105 ***
estrg    0.01204   1.01212  0.04680  0.257 0.796895
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tidy `coxph()` Output

- **Using `broom` package:** `broom::tidy()`
  - Provides a tidy data frame for easy manipulation and visualization

```
1 library(broom) # Load broom package
2 tidy_cox <- tidy(cox_fit) # Tidy the coxph output
3 tidy_cox # Display the tidy output
```

```
# A tibble: 8 × 5
  term    estimate std.error statistic   p.value
  <chr>      <dbl>     <dbl>     <dbl>     <dbl>
1 hormone  -0.374    0.129      -2.90  0.00376
2 meno      0.284    0.140       2.04  0.0417
3 age40    -0.551    0.202      -2.72  0.00646
4 grade2    0.715    0.249       2.88  0.00399
5 grade3    0.775    0.270       2.87  0.00408
6 size      0.0161   0.00368     4.37  0.0000127
7 prog     -0.224    0.0578     -3.88  0.000105
8 estrg     0.0120   0.0468      0.257 0.797
```

Tabulating Results with `gtsummary` (I)

- **Using `gtsummary` package:** `tbl_regression()`
  - Automatically formats regression results into a publication-ready table

```
library(gtsummary) # Load gtsummary package
cox_tbl <- cox_fit |> tbl_regression( # Create a regression table
                exponentiate = TRUE, # Exponentiate coefficients to get hazard ratios
                label = list(hormone ~ "Hormone Therapy", # Custom labels
                             meno ~ "Menopausal",
                             age40 ~ "Older than 40",
                             grade ~ "Tumor Grade",
                             size ~ "Tumor Size (mm)",
                             prog ~ "Progesterone Receptor (100 fmol/ml)",
                             estrg ~ "Estrogen Receptor  (100 fmol/ml)")
              ) |>
          add_global_p() # Add global p-value for categorical variables
cox_tbl # Display the regression table
```

Tabulating Results with `gtsummary` (II)

- **Result**

Characteristic	HR[1]	95% CI[1]	p-value
Hormone Therapy	0.69	0.53, 0.89	0.003
Menopausal	1.33	1.01, 1.75	0.039
Older than 40	0.58	0.39, 0.86	0.009
Tumor Grade			0.004
1	—	—	
2	2.05	1.26, 3.33	
3	2.17	1.28, 3.68	
Tumor Size (mm)	1.02	1.01, 1.02	<0.001
Progesterone Receptor (100 fmol/ml)	0.80	0.71, 0.90	<0.001
Estrogen Receptor (100 fmol/ml)	1.01	0.92, 1.11	0.8

[1] HR = Hazard Ratio, CI = Confidence Interval

Further Customization

- **Styling functions**
  - `modify_header()`: update column headers
  - `modify_footnote_header()`: update column header footnote
  - `modify_footnote_body()`: update table body footnote
  - `modify_caption()`: update table caption/title
  - `bold_labels()`: bold variable labels
  - `bold_levels()`: bold variable levels
  - `italicize_labels()`: italicize variable labels
  - `italicize_levels()`: italicize variable levels
  - `bold_p()`: bold significant p-values
- More about `tbl_regression()`
  - `gtsummary` documentation

Table Customization Exercise

- **Task**: Customize the regression table
  - Add a caption: “Cox regression analysis of the German breast cancer study”
  - Bold significant p-values
  - Italicize tumor grade levels

▶ Solution

Other Regression Models

- **Accelerated failure time (AFT) models**

$$\log T = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \ldots + \beta_p Z_p + \epsilon$$

  - $\epsilon \sim$ Weibull, lognormal, etc. (parametric models)
  - $\exp(\beta_j)$: acceleration factor for covariate Z_j
  - **Model fitting**: `survival::survreg()`

```
# Fit a Weibull AFT model
aft_fit <- survreg(Surv(time, status) ~ hormone + meno + age + grade + size + prog + estrg,
                   data = df, dist = "weibull") # specify the Weibull model
```

Exercise

- Tidy up the `survreg` object `aft_fit` using `broom::tidy()`
- Create a regression table using `gtsummary::tbl_regression()`

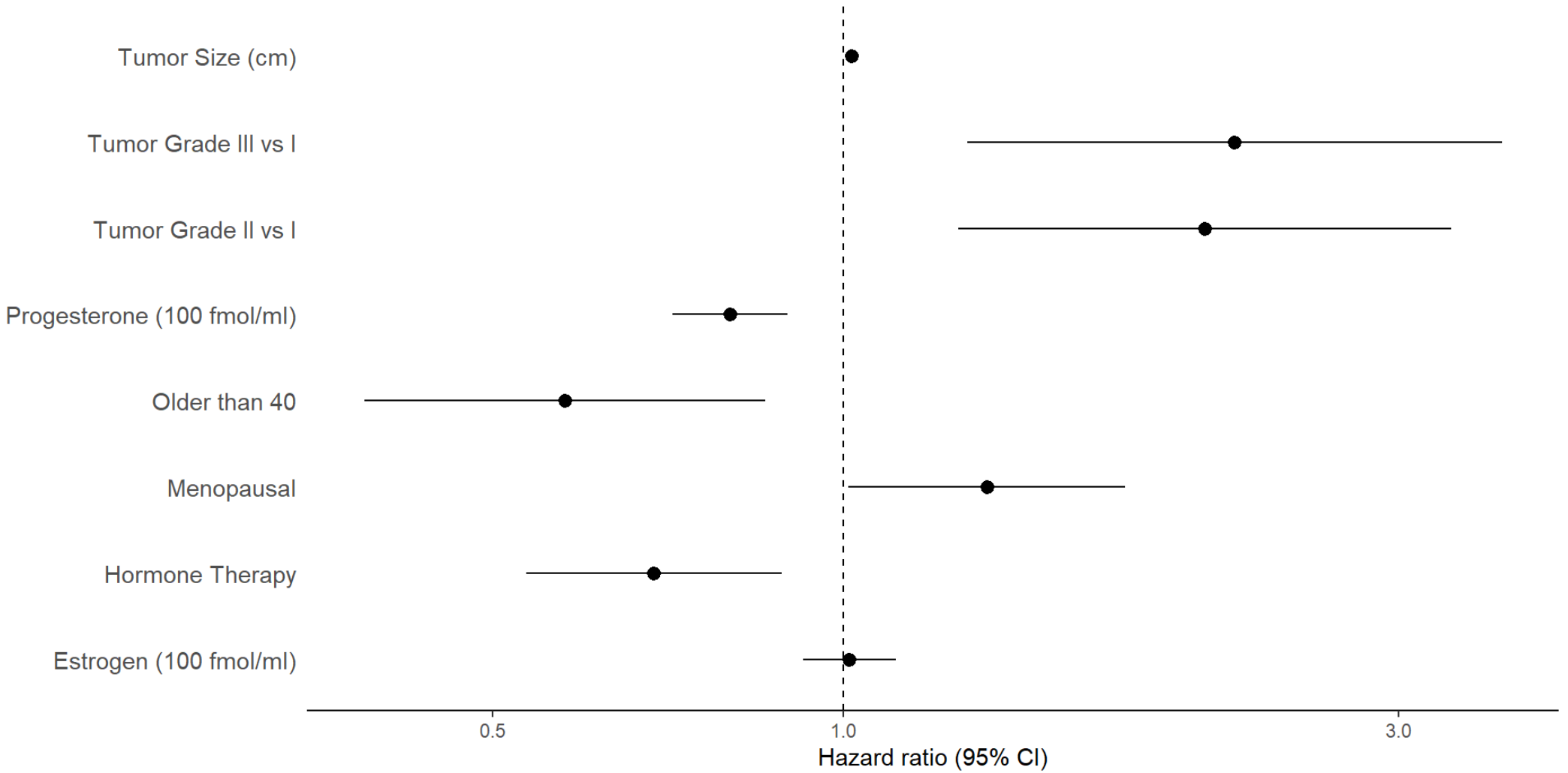
Visualizing Hazard Ratios (I)

- **Forest plot**: Visualize hazard ratios and confidence intervals

```r
# Tidy with exponentiated coeffs (HR) and CI
tidy_cox <- tidy(cox_fit, exponentiate = TRUE, conf.int = TRUE)
tidy_cox$term <- recode(tidy_cox$term,            # Relabel the variables
                 hormone = "Hormone Therapy",
                 meno = "Menopausal",
                 age40 = "Older than 40",
                 grade2 = "Tumor Grade II vs I",
                 grade3 = "Tumor Grade III vs I",
                 size = "Tumor Size (mm)",
                 prog = "Progesterone (100 fmol/ml)",
                 estrg = "Estrogen (100 fmol/ml)")

tidy_cox |>  # plot of hazard ratios and 95% CIs
  ggplot(aes(y=term, x=estimate, xmin=conf.low, xmax=conf.high)) +
  geom_pointrange() +  # plots center point (x) and range (xmin, xmax)
  geom_vline(xintercept=1, linetype = 2) + # vertical line at HR=1
  scale_x_log10("Hazard ratio (95% CI)") + # log scale for x-axis
  theme_classic() + # classic theme for clean look
  theme(
    axis.line.y = element_blank(),          # remove y-axis line
    axis.ticks.y = element_blank(),         # remove y-axis ticks
    axis.text.y = element_text(size = 11),  # set variable label size
```

Visualizing Hazard Ratios (II)

- **Result**



Forest Plot Exercise

- **Task**: customize the forest plot
  - Use square rather than default circle for point estimates
  - Set x-axis ticks at 0.5, 1, 2.0, and 4.0
  - Add a title: “Cox Regression Results for GBC Data”

▶ Solution

Cox Model Prediction and Diagnostics

Model-Based Prediction

- **Predicted survival function**

$$\hat{S}(t \mid z) = \exp\left\{ - \exp(\hat{\beta}^{\mathrm{T}} z) \hat{\Lambda}_0(t) \right\}$$

- Prepare new data for prediction
  - A post-menpausal woman older than 40, undergoing hormone therapy, with tumor grade II, tumor size 20 mm, and progesterone and estrogen receptor levels both 100 fmol/ml.

```r
# Create new data for prediction
# specify all covariate values
new_data <- data.frame(hormone = 2, meno = 2, age40 = 1, grade = factor(2),
                       size = 20, prog = 1, estrg = 1)

new_data
```

```
  hormone meno age40 grade size prog estrg
1       2    2     1     2   20    1     1
```

Tidy Survival Prediction

- Use `survival::survfit()` to predict survival probabilities
  - `newdata`: new data for prediction
  - `times`: time points for prediction
  - `broom::tidy()` to tidy the output

```
# Predict survival probabilities for `newdata`
pred_surv <- survfit(cox_fit, newdata = new_data[1, ])
tidy_pred_surv <- tidy(pred_surv) # Tidy the survival prediction output
head(tidy_pred_surv) # Display the first few rows of the tidy output
```

```
# A tibble: 6 × 8
   time n.risk n.event n.censor estimate std.error conf.high conf.low
  <dbl>  <dbl>   <dbl>    <dbl>    <dbl>     <dbl>     <dbl>    <dbl>
1 0.262    686       0        1        1         0         1        1
2 0.492    685       0        1        1         0         1        1
3 0.525    684       0        1        1         0         1        1
4 0.557    683       0        2        1         0         1        1
5 0.590    681       0        1        1         0         1        1
6 0.951    680       0        1        1         0         1        1
```

Visualizing Predicted Survival (I)

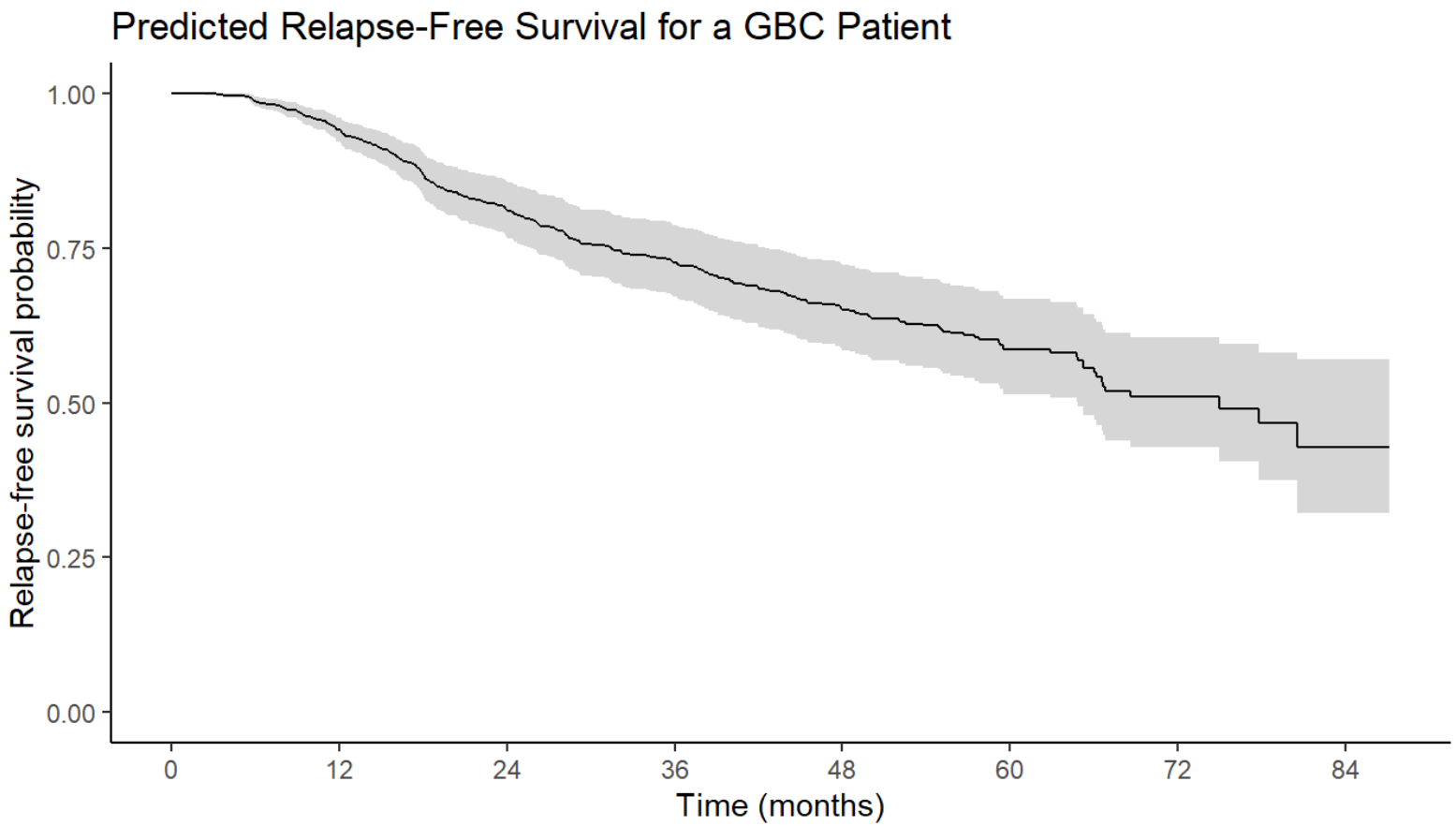
- **Using `ggsurvfit` package:** `ggsurvfit()`
  - Pass `survfit` object to `ggsurvfit()`
  - Similar customization to KM curves

```
library(ggsurvfit) # Load ggsurvfit package
pred_fig <- pred_surv |>  # Pass the survfit object
  ggsurvfit() + # Main function
  add_confidence_interval() + # Add confidence interval
  scale_x_continuous("Time (months)", breaks = seq(0, 84, by = 12)) + # x-axis format
  scale_y_continuous("Relapse-free survival probability", limits = c(0, 1)) + # y-axis format
  ggtitle("Predicted Relapse-Free Survival for a GBC Patient") + # Add title
  theme_classic() # Classic theme for clean look
```

Visualizing Predicted Survival (II)

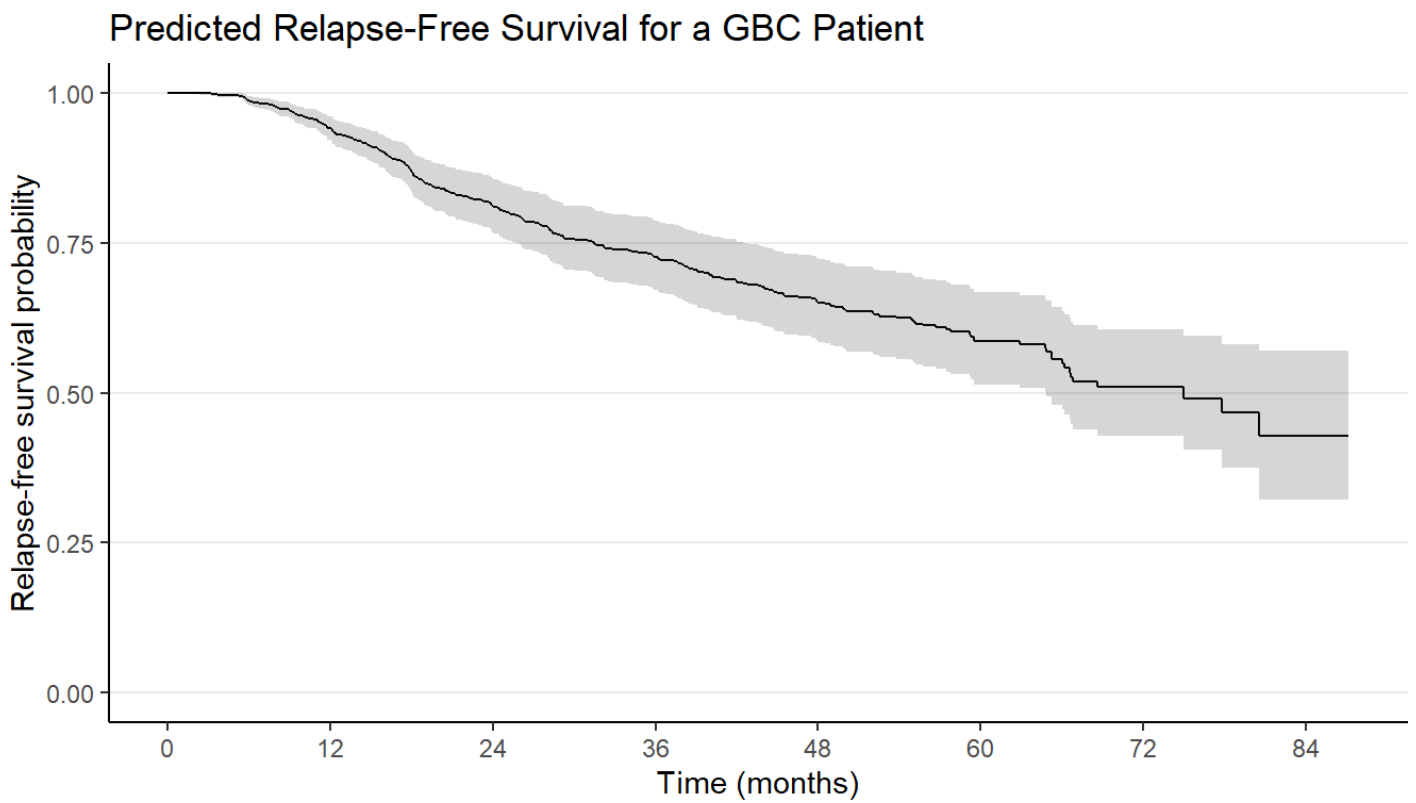
- **Result**

```
1 pred_fig # print figure
```



Prediction Graphics Exercise

- **Task**: Add horizontal grid lines



▶ Solution

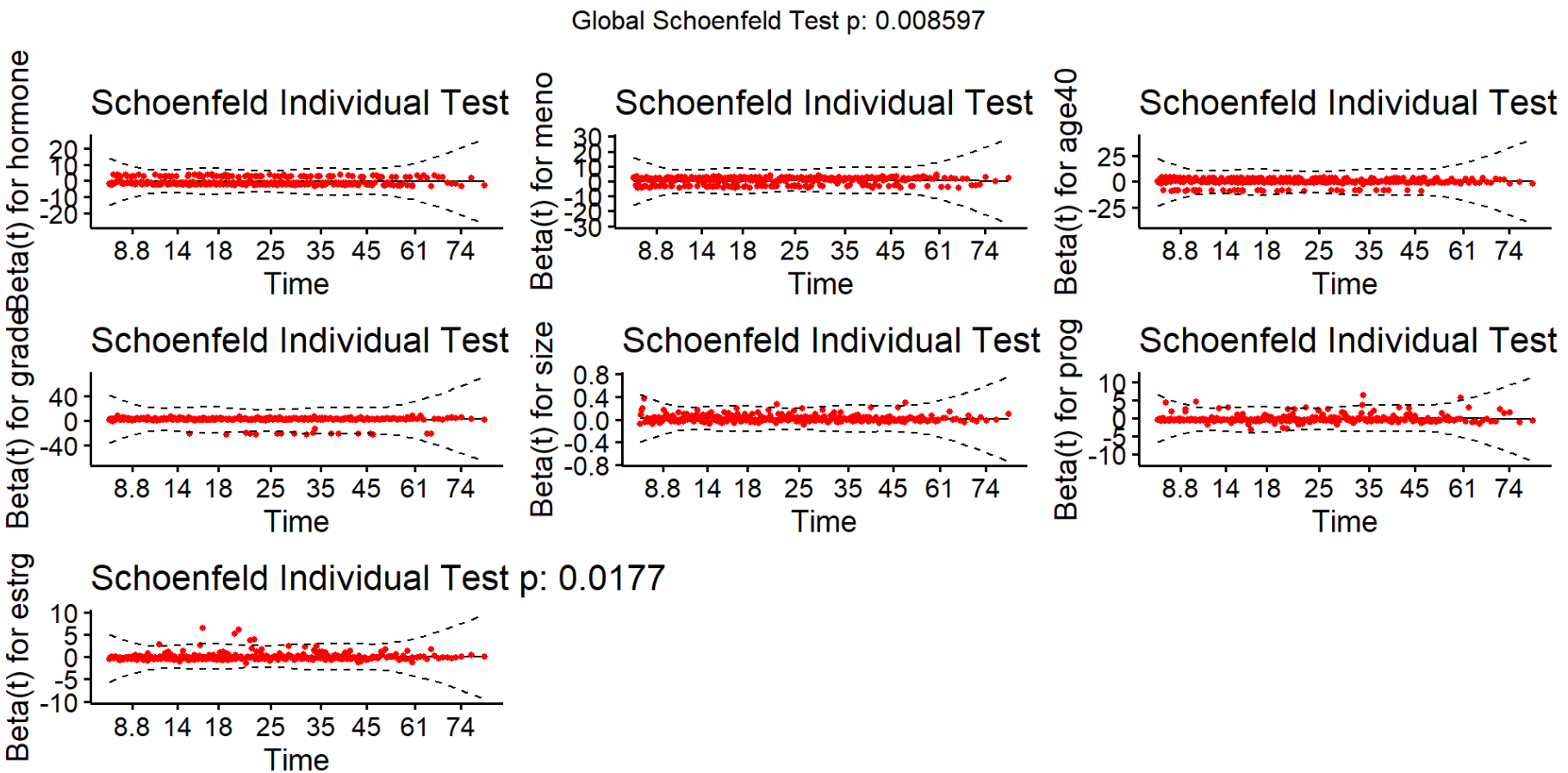
Cox Model Diagnostics

- **Ph assumptions: Schoenfeld residuals**
  - Difference between observed and expected covariate values at each event time
  - Use `cox.zph()` to test PH assumption
  - Use `survminer::ggcoxzph()` on `cox.zph` object to visualize Schoenfeld residuals
- **Functional form of covariates**
  - Plot martingale residuals against (quantitative) covariates
  - Use `residuals(cox_fit, type = "martingale")` to get martingale residuals`
- **Other aspects**
  - Appropriateness of exponential link function
  - Influential points/outliers
  - `survminer::ggcoxdiagnostics()`

Schoenfeld Residuals

- **Check proportionality**
  - Focus on graphics; use p-value only as guideline

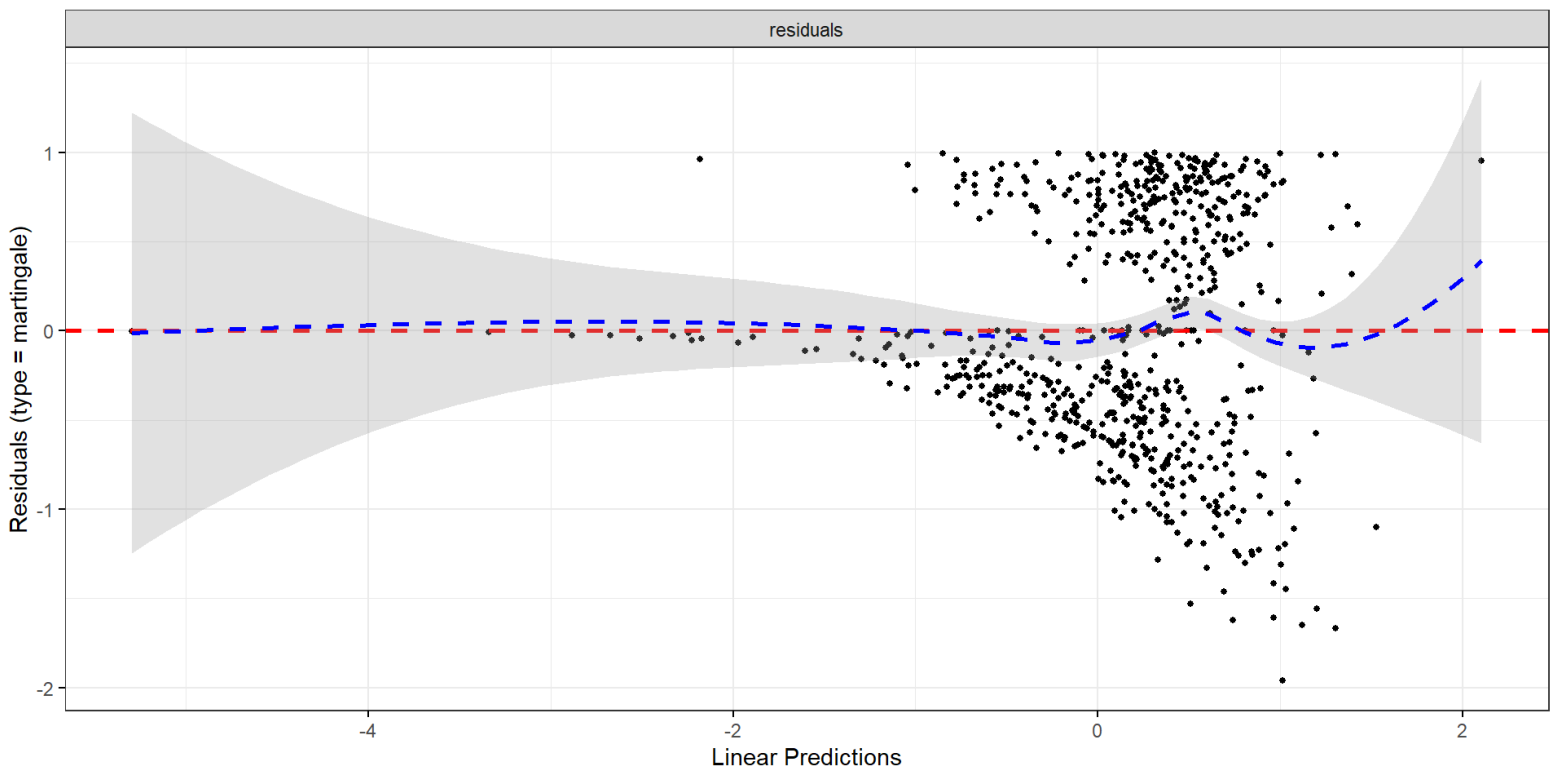
```
1 library(survminer) # Load survminer package
2 ph_test <- cox.zph(cox_fit) # Test proportional hazards assumption
3 ggcoxzph(ph_test) # Visualize Schoenfeld residuals
```



Exponential Link Function

- **Martingale vs.** $\hat{\beta}^{\mathrm{T}} Z_i$

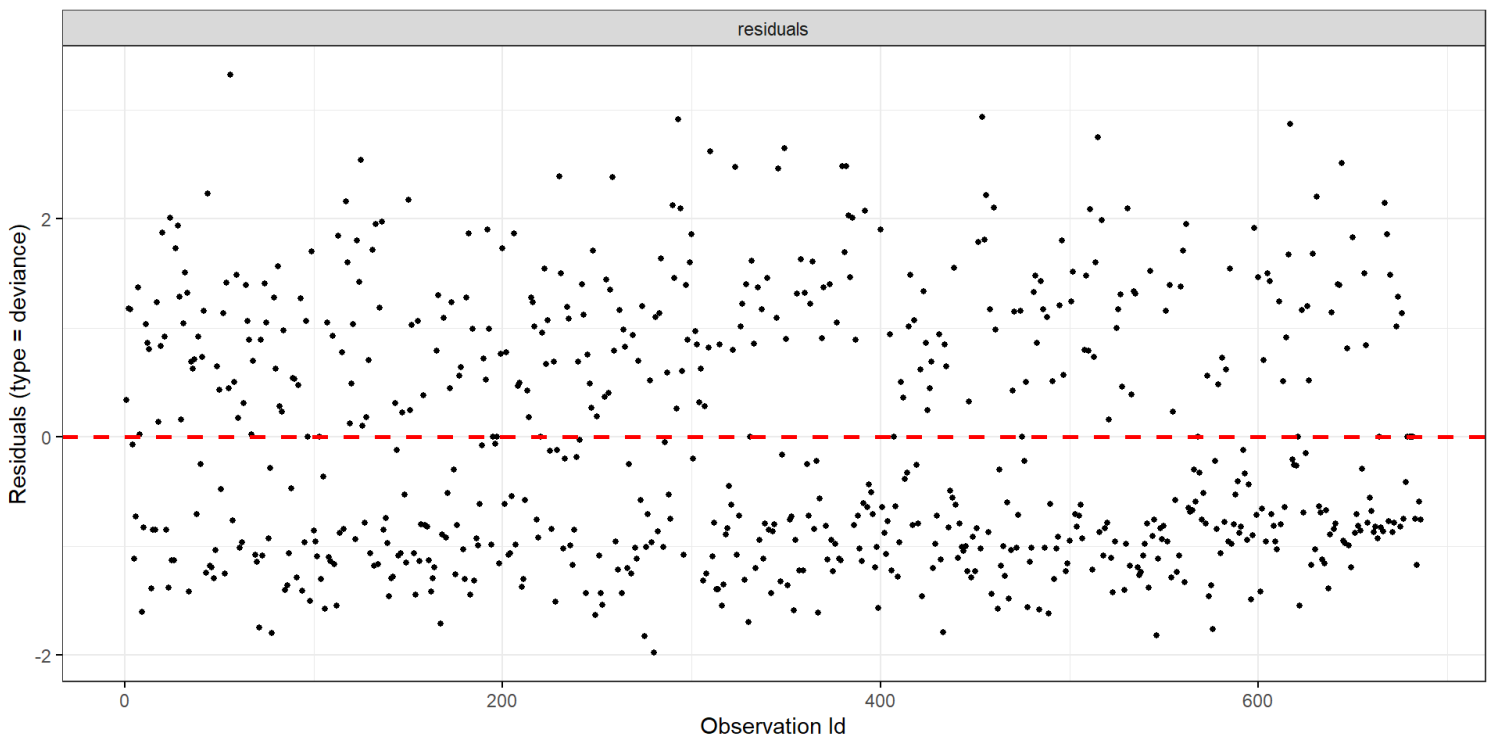
```
# Martingale residuals vs linear predictor
ggcoxdiagnostics(cox_fit, type = "martingale",    # martingale on y-axi
                 ox.scale = "linear.predictions") # linear predictor on x-axis
```



Influential Points

- **Deviance residuals**

```r
# Deviance residuals vs linear predictor
ggcoxdiagnostics(cox_fit, type = "deviance",  # deviance on y-axis
                 ox.scale = "observation.id", # observation ID on x-axis
                 sline = FALSE)               # no smoothed line
```



General Residual Graphics

- **Basic arguments of `ggcoxdiagnostics()`**
  - `coxph` object
  - `type`: Residual type (“martingale”, “deviance”, “score”, “schoenfeld”, “dfbeta”, “dfbetas”, and “scaledsch”)`
  - `ox.scale`: Scale for x-axis (“linear.predictions”, “observation.id”, “time”)
  - `point.col`: Color of points
  - `point.size`: Size of points
  - etc.
- More about `survminer`
  - survminer website

Competing Risks Regression

Sub-Distribution Hazard

- **Definition**

$$\Lambda_k(t \mid Z) = -\log\{1 - F_k(t \mid Z)\}$$

  - $F_k(t \mid Z)$: cumulative incidence function (CIF) of the k-th cause
  - $\lambda_k(t \mid Z) = \Lambda_k'(t)$: risk of the k-th cause in presence of other competing events in the whole population

- **Different from cause-specific hazard**
  - Cause-specific hazard

$$\lambda_k^{\mathrm{c}}(t \mid Z) = \Pr(t \le T < t + \mathrm{d}t \mid T \ge t, Z)/\mathrm{d}t$$

  - Risk of the k-th cause in *survivors*

Fine-Gray Model

- **Proportional sub-distribution hazards**

$$\lambda_k(t \mid Z) = \lambda_0(t) \exp(\beta_1 Z_1 + \beta_2 Z_2 + \ldots + \beta_p Z_p)$$

  - $\lambda_0(t)$: baseline sub-distribution hazard function
  - $\exp(\beta_j)$: sub-distribution hazard ratio for covariate Z_j

```r
library(tidycmprsk) # Load tidycmprsk package
data("trial", package = "tidycmprsk") # Load trial data from tidycmprsk package
head(trial) # Display the first few rows of the data
```

```
# A tibble: 6 × 9
  trt      age marker stage grade response death death_cr           ttdeath
  <chr>  <dbl>  <dbl> <fct> <fct>    <int> <int> <fct>                <dbl>
1 Drug A    23  0.16  T1    II           0     0 censor                24
2 Drug B     9  1.11  T2    I            1     0 censor                24
3 Drug A    31  0.277 T1    II           0     0 censor                24
4 Drug A    NA  2.07  T3    III          1     1 death other causes    17.6
5 Drug A    51  2.77  T4    III          1     1 death other causes    16.4
6 Drug B    39  0.613 T4    I            0     1 death from cancer     15.6
```

Fitting Fine-Gray Model

- Using `cmprsk::crr()`
  - `formula`: `Surv(time, status) ~ covariates`
    - `status`: a factor with first level indicating censoring and subsequent levels the competing risks
  - `failcode`: event code for the cause of interest

```
fg_fit <- crr(Surv(ttdeath, death_cr) ~ trt + age + marker + stage, # fit FG model
              failcode = "death from cancer", trial) # for death from cancer
```

```
21 cases omitted due to missing values
```

```
fg_fit # print the Fine-Gray model fit summary
```

```
Variable    Coef     SE      HR     95% CI       p-value
trtDrug B   0.396    0.283   1.49   0.85, 2.59   0.16
age         0.009    0.011   1.01   0.99, 1.03   0.42
marker      -0.002   0.159   1.00   0.73, 1.36   0.99
stageT2     0.140    0.475   1.15   0.45, 2.92   0.77
stageT3     0.500    0.460   1.65   0.67, 4.06   0.28
stageT4     0.959    0.418   2.61   1.15, 5.91   0.022
```

Parameter Estimates and Variance

- Extracting $\hat{\beta}$ and $\hat{\text{var}}(\hat{\beta})$

```
1 coef(fg_fit) # Extract coefficients
```

```
   trtDrug B          age       marker      stageT2      stageT3      stageT4
 0.396486121  0.008956935 -0.002006717  0.140251967  0.500394141  0.958640247
```

```
1 vcov(fg_fit) |> head() # Extract variance-covariance matrix
```

```
              [,1]          [,2]          [,3]         [,4]          [,5]
[1,]  0.0800665101  0.0001535045 -0.0051801922  0.011605373  0.0094601803
[2,]  0.0001535045  0.0001239790 -0.0005094795  0.001111245 -0.0009368412
[3,] -0.0051801922 -0.0005094795  0.0251827414 -0.028697647 -0.0037297926
[4,]  0.0116053732  0.0011112447 -0.0286976466  0.225187822  0.1101888313
[5,]  0.0094601803 -0.0009368412 -0.0037297926  0.110188831  0.2111942725
[6,]  0.0219362509  0.0010459739 -0.0176113331  0.124589145  0.1064088264
              [,6]
[1,]  0.021936251
[2,]  0.001045974
[3,] -0.017611333
[4,]  0.124589145
[5,]  0.106408826
[6,]  0.174446817
```

Tidy Fine-Gray Model Output

- **Using `broom` package:** `broom::tidy()`
  - Provides a tidy data frame for easy manipulation and visualization

```r
tidy_fg <- tidy(fg_fit, exponentiate = TRUE, conf.int = TRUE) # Tidy model output
tidy_fg # Display the tidy output
```

```
# A tibble: 6 × 7
  term      estimate std.error statistic conf.low conf.high p.value
  <chr>        <dbl>     <dbl>     <dbl>    <dbl>     <dbl>   <dbl>
1 trtDrug B    1.49      0.283     1.40      0.854      2.59   0.16
2 age          1.01      0.0111    0.804     0.987      1.03   0.42
3 marker       0.998     0.159    -0.0126    0.731      1.36   0.99
4 stageT2      1.15      0.475     0.296     0.454      2.92   0.77
5 stageT3      1.65      0.460     1.09      0.670      4.06   0.28
6 stageT4      2.61      0.418     2.30      1.15       5.91   0.022
```

Forest Plot Exercise

- **Task**: Visualize sub-distribution hazard ratios and confidence intervals

▸ Solution

FG Regression Table (I)

- **Using `gtsummary` package:** `tbl_regression()`
  - Similarly to tabulating fitted `coxph` object

```
library(gtsummary) # Load gtsummary package
fg_tbl <- fg_fit |> tbl_regression(exponentiate = TRUE) |>  # Create a regression table
          add_global_p() # Add global p-value for categorical variables
```

FG Regression Table (II)

- **Result**

Characteristic	HR[1]	95% CI[1]	p-value
Chemotherapy Treatment			0.2
Drug A	—	—	
Drug B	1.49	0.85, 2.59	
Age	1.01	0.99, 1.03	0.4
Marker Level (ng/mL)	1.00	0.73, 1.36	>0.9
T Stage			0.058
T1	—	—	
T2	1.15	0.45, 2.92	
T3	1.65	0.67, 4.06	
T4	2.61	1.15, 5.91	

[1] HR = Hazard Ratio, CI = Confidence Interval

Model-Based Prediction

- **Predicted cumulative incidence function (CIF)**

$$\hat{F}_k(t \mid z) = 1 - \exp\Big\{-\hat{\Lambda}_k(t \mid z)\Big\}$$

```r
# Predict cumulative incidence function for first 10 patients
fg_pred <- predict(fg_fit, newdata= trial[1:10, ], times = c(6, 12, 18)) # Predict CIF
fg_pred # Display the predicted CIF
```

```
$`time 6`
 [1] 0.002279375 0.003430702 0.002447917          NA 0.007579470 0.010150002
 [7] 0.002582498 0.002462688 0.002448570 0.006155050

$`time 12`
 [1] 0.02093164 0.03135502 0.02246374         NA 0.06809922 0.09023665
 [7] 0.02368558 0.02259790 0.02246967 0.05562630

$`time 18`
 [1] 0.07090010 0.10483563 0.07594466         NA 0.21744137 0.28018810
 [7] 0.07995368 0.07638548 0.07596414 0.18042196
```

Summary

Key Takeaways

- **Cox proportional hazards regression**
  - Tidy output with `broom` and `gtsummary`
  - Visualize hazard ratios with forest plots with `ggplot2`
  - Model-based prediction with `survival::survfit()` and `ggsurvfit()`
  - Model diagnostics with `survminer`
- **Fine-Gray model for competing risks regression**
  - Tidy output with `broom` and `gtsummary`
  - Visualize sub-distribution hazard ratios with forest plots

Next Steps

- **Machine learning**: build best predictive model with many predictors
  - Regularized Cox regression
  - Parametric AFT models
  - Survival trees
  - `tidymodels` packages (`censored`)

Tidy Survival Analysis: Applying R's Tidyverse to Survival Data

Module 5. Machine Learning

Lu Mao

lmao@biostat.wisc.edu

Department of Biostatistics & Medical Informatics

University of Wisconsin-Madison

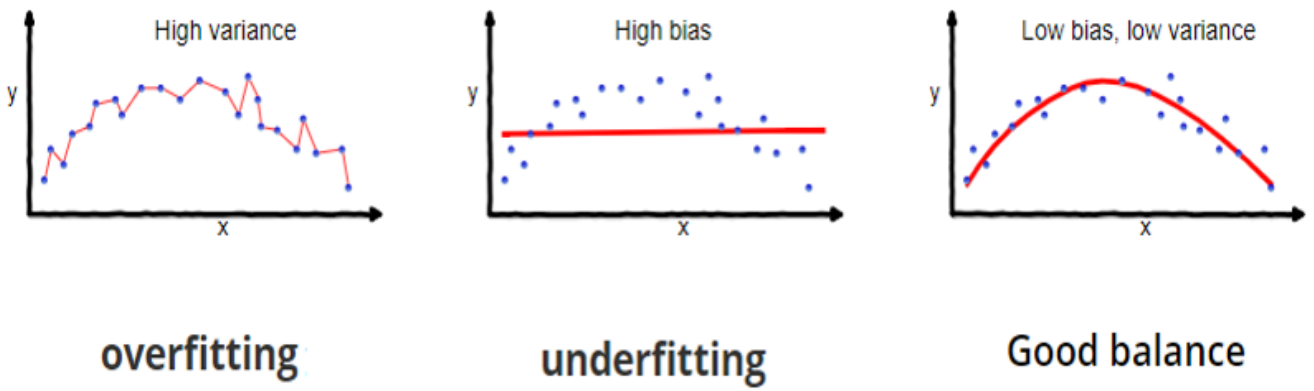
Aug 3, 2025

Table of contents

Machine Learning Survival Models

Setting

- **With many covariates**
  - **Prediction accuracy**: under- vs over-fitting



    - Too many predictors → overfitting
  - **Interpretation**: easier with fewer predictors

Regularized Cox Regression

- **Idea**
  - Penalize the magnitude of coefficients (-norm) to avoid overfitting
- **Elastic net**: minimize objective function
  - : tuning parameter that controls the strength of penalty
    - Determined by *cross-validation*
  - : controls the type of penalty
    - ridge regression: handles correlated predictors better
    - lasso regression: performs variable selection
  - **Implementation**: `glmnet` package

Survival Trees

- **Decision trees**
  - *Classification and Regression Trees* (CART; Breiman et al., 1984)
  - Root node (all sample) split into (more homogeneous) daughter nodes split recursively
- **Growing the tree**
  - Starting with root node, search partition criteria for one that minimizes “impurity” (e.g., mean squared deviance residuals) within daughter nodes
  - Recursive splitting until terminal nodes sufficiently “pure” in outcome

Complexity Control and Prediction

- **Pruning the tree**
  - Cut overgrown branches to prevent overfitting
  - Penalize number of terminal nodes
  - Tune complexity parameter (or minimum size of terminal node)
- **Prediction**
  - New terminal node KM estimates (or median survival)
- **Implementation**: `rpart` package

Random Forests

- **Limitation of a single tree**
  - High variance: small changes in data can lead to large changes in predictions
- **Random forests**
  - Bootstrap samples from training data
  - Take a random subset of covariates to split on (decorrelate the trees)
  - Tune the number of covariates to split on
- **Implementation**: `aorsf` package

Model Evaluation

- **Brier score**
  - Mean squared error between observed survival status and predicted survival probability
  - Inverse probability censoring weighting (IPCW) to account for censoring
  - Integrated Brier score: average Brier score over a time interval
- **ROC AUC**
  - Area under the receiver operating characteristic (ROC) curve for survival status
  - IPCW to handle censoring
  - Concordance index: overall AUC over time

tidymodels Workflows

Overview of `tidymodels` and `censored`

- `tidymodels`: a collection of packages for modeling and machine learning in R
  - Provides a *consistent interface* for model training, tuning, and evaluation
    - Key package `parsnip`
  - Supports various model types, including regression, classification, and survival analysis
- `censored`: a `parsnip` extension package for survival data
  - Implements parametric, semiparametric, and tree-based survival models

Data Preparation and Splitting

- **Create a `Surv` object** as response
  - `Surv(time, event)`

```
library(tidymodels)
library(censored)
df <- df |>
  mutate(
    surv_obj = Surv(time, event), # create the Surv object as response variable
    .keep = "unused"              # discard original time and event columns
  )
```

- **Data splitting**
  - `initial_split()`: splits data into training and testing sets

```
df_split <- initial_split(flight_data, prop = 3/4) # default ratio 3:1
df_train <- training(df_split) # obtain training set
```

Model Specification

- **Model type**
  - `survival_reg()`: parametric AFT models
  - `proportional_hazards(penalty = tune())`: (regularized) Cox PH models
  - `decision_tree(complexity = tune())`: decision trees
  - `rand_forest(mtry = tune())`: random forests
- **Set engine and mode**
  - `set_engine("survival")`: for AFT models
  - `set_engine("glmnet")`: for Cox PH models
  - `set_engine("aorsf")`: for random forests
  - `set_mode("censored regression")`: for survival models

```
model_spec <- proportional_hazards(penalty = tune()) |>  # regularized Cox model (tune lambda)
  set_engine("glmnet") |>  # set engine to glmnet
  set_mode("censored regression") # set mode to censored regression
```

Recipe and Workflow

- **Recipe**: a series of preprocessing steps for the data
  - `recipe(response ~ ., data = df)`: specify response and predictors
  - `step_mutate()`: standardize numeric predictors
  - `step_dummy()`: convert categorical variables to dummy variables
- **Workflow**: combines model specification and recipe
  - `workflow() |> add_model(model_spec) |> add_recipe(recipe)`

```
# Create a recipe
model_recipe <- recipe(surv_obj ~ ., data = df_train) |> # specify formula
  step_mutate(z1 = z1 / 1000) |>  # standardize z1
  step_other(z2, z3, threshold = 0.02) |> # group levels with prop < .02 into "other"
  step_dummy(all_nominal_predictors())  # convert categorical variables to dummy variables
# Create a workflow by combining model and recipe
model_wflow <- workflow() |>
  add_model(model_spec) |>   # add model specification
  add_recipe(model_recipe)   # add recipe
```

Tune Hyperparameters

- **Cross-validation**
  - `df_train_folds <- vfold_cv(df_train, v = k)`: create *k*-folds on training data (default 10)
  - `tune_grid(model_wflow, resamples = df_train_folds)`: tune hyperparameters using cross-validation

```
# k-fold cross-validation
df_train_folds <- vfold_cv(df_train, v = 10) # 10-fold cross-validation
# Tune hyperparameters
model_res <- tune_grid(
  model_wflow,
  resamples = df_train_folds,
  grid = 10, # number of hyperparameter combinations to try
  metrics = metric_set(brier_survival, brier_survival_integrated,  # specify metrics
                       roc_auc_survival, concordance_survival),
  eval_time = seq(0, 84, by = 12) # evaluation time points
)
```

Finalize Workflow

- **Examine validation results**
  - `collect_metrics(model_res)`: collect metrics from tuning results
  - `show_best(model_res, metric = "brier_survival_integrated", n = 5)`: show top 5 models based on Brier score
- **Workflow for best model**
  - `param_best <- select_best(model_res, metric = "brier_survival_integrated")`: select best hyperparameters based on Brier score
  - `final_wl <- finalize_workflow(model_wflow, param_best)`: finalize workflow with best hyperparameters

```
# Extract the best hyperparameters based on Brier score
param_best <- select_best(model_res, metric = "brier_survival_integrated")
# Finalize the workflow with the best hyperparameters
final_wl <- model_wflow |> finalize_workflow(param_best)
```

Fit Final Model

- **Fit the finalized workflow**
  - `final_mod <- last_fit(final_wl, split = df_split)`: fit the finalized workflow on the testing set
  - `collect_metrics(final_mod)`: collect metrics of final model on test data
- **Make predictions**
  - `predict(final_mod, new_data = new_data, type = "time")`: predict survival times on new data

```
# Fit the finalized workflow on the testing set
final_mod <- last_fit(final_wl, split = df_split)
# Collect metrics of final model on test data
collect_metrics(final_mod) %>%
  filter(.metric == "brier_survival_integrated")
# Make predictions on new data
new_data <- testing(df_split) |>  slice(1:5) # take first 5 rows of test data
predict(final_mod, new_data = new_data, type = "time")
```

A Case Study

GBC: Relapse-Free Survival

- **Time to first event**

```r
library(tidymodels) # load tidymodels
library(censored)
gbc <- read.table("data/gbc.txt", header = TRUE) # Load GBC dataset
df <- gbc |>  # calculate time to first event (relapse or death)
  group_by(id) |> # group by id
  arrange(time) |> # sort rows by time
  slice(1) |>      # get the first row within each id
  ungroup() |>
  mutate(
    surv_obj = Surv(time, status), # create the Surv object as response variable
    .after = id, # keep id column after surv_obj
    .keep = "unused" # discard original time and status columns
  )
```

Data Preparation

- **Analysis dataset**

```
1 head(df) # show the first few rows of the dataset
```

```
# A tibble: 6 × 10
     id   surv_obj hormone   age  meno  size grade nodes  prog estrg
  <int>      <Surv>   <int> <int> <int> <int> <int> <int> <int> <int>
1     1 43.836066        1    38     1    18     3     5   141   105
2     2 46.557377        1    52     1    20     1     1    78    14
3     3 41.934426        1    47     1    30     2     1   422    89
4     4  4.852459+       1    40     1    24     1     3    25    11
5     5 61.081967+       2    64     2    19     2     1    19     9
6     6 63.377049+       2    49     2    56     1     3   356    64
```

- **Data splitting**

```
1 set.seed(123) # set seed for reproducibility
2 gbc_split <- initial_split(df) # split data into training and testing sets
3 gbc_split
```

```
<Training/Testing/Total>
<514/172/686>
```

Models to be Trained

- **Regularized Cox model**
  - `proportional_hazards(penalty = tune())`
  - Default: (lasso)
  - Tune penalty parameter
  - Use `glmnet` engine for fitting
- **Random forest**
  - `rand_forest(mtry = tune(), min_n = tune())`
  - Tune number of predictors to split on and minimum size of terminal node
  - Use `aorsf` engine for fitting

```
# Training data
gbc_train <- training(gbc_split) # obtain training set
```

Common Recipe

- **Recipe for both models**

```
gbc_recipe <- recipe(surv_obj ~ ., data = gbc_train) |> # specify formula
  step_mutate(
    grade = factor(grade),
    age40 = as.numeric(age >= 40), # create a binary variable for age >= 40
    prog = prog / 100, # rescale prog
    estrg = estrg / 100 # rescale estrg
  ) |>
  step_dummy(grade) |>
  step_rm(id) # remove id
# gbc_recipe # print recipe information
```

Regularized Cox Model

- **Cox model specification and workflow**

```r
# Regularized Cox model specification
cox_spec <- proportional_hazards(penalty = tune()) |>  # tune lambda
  set_engine("glmnet") |>  # set engine to glmnet
  set_mode("censored regression") # set mode to censored regression
cox_spec # print model specification
```

```
Proportional Hazards Model Specification (censored regression)

Main Arguments:
  penalty = tune()

Computational engine: glmnet
```

```r
# Create a workflow by combining model and recipe
cox_wflow <- workflow() |>
  add_model(cox_spec) |>   # add model specification
  add_recipe(gbc_recipe)   # add recipe
```

Model Tuning

- **Cross-validation set-up**
  - For both models

```r
set.seed(123) # set seed for reproducibility
gbc_folds <- vfold_cv(gbc_train, v = 10) # 10-fold cross-validation
# Set evaulation metrics
gbc_metrics <- metric_set(brier_survival, brier_survival_integrated,
                          roc_auc_survival, concordance_survival)
gbc_metrics # evaluation metrics info
```

```
A metric set, consisting of:
- `brier_survival()`, a dynamic survival metric              | direction:
minimize
- `brier_survival_integrated()`, a integrated survival metric | direction:
minimize
- `roc_auc_survival()`, a dynamic survival metric            | direction:
maximize
- `concordance_survival()`, a static survival metric         | direction:
maximize
```

```r
# Set evaluation time points
time_points <- seq(0, 84, by = 12) # evaluation time points
```

Cox Model Tuning

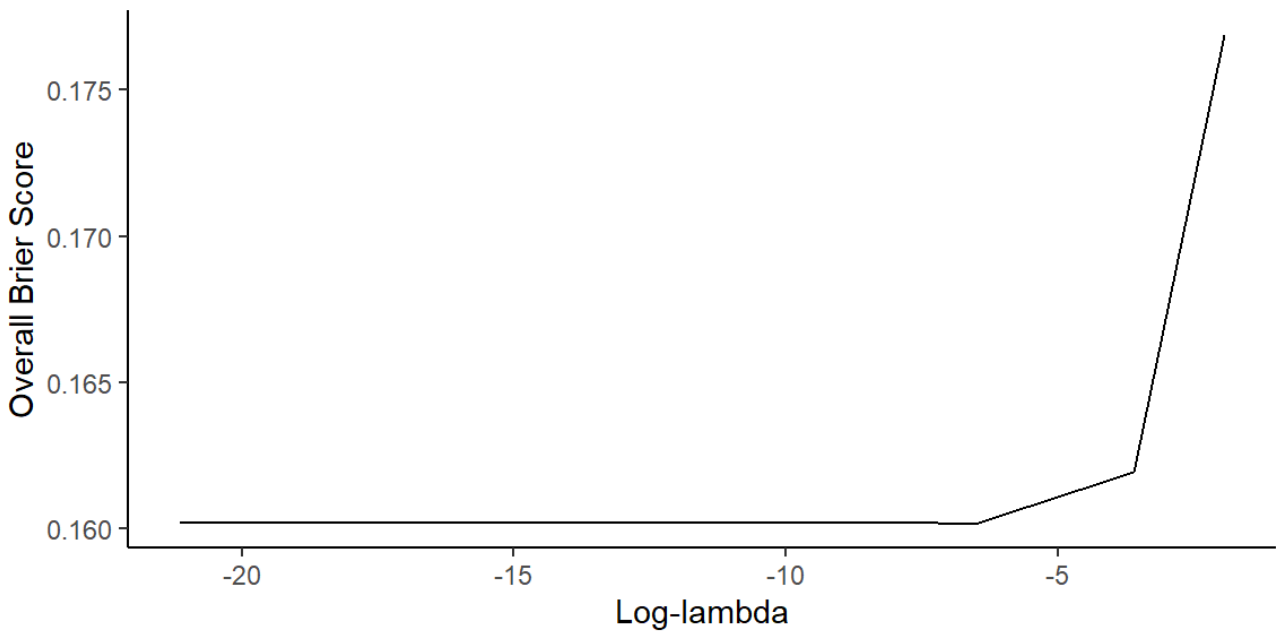
- **Tune the regularized Cox model**
  - Use `tune_grid()` to perform hyperparameter tuning
  - Evaluate performance using Brier score and ROC AUC

```
set.seed(123) # set seed for reproducibility
# Tune the regularized Cox model (this will take some time)
cox_res <- tune_grid(
  cox_wflow,
  resamples = gbc_folds,
  grid = 10, # number of hyperparameter combinations to try
  metrics = gbc_metrics, # evaluation metrics
  eval_time = time_points, # evaluation time points
  control = control_grid(save_workflow = TRUE) # save workflow
)
```

Cox Model Tuning Results

- Plot Brier score as function of

```
collect_metrics(cox_res) |>  # collect metrics from tuning results
  filter(.metric == "brier_survival_integrated") |>  # filter for Brier score
  ggplot(aes(log(penalty), mean)) + # plot log-lambda vs Brier score
  geom_line() +  # plot line
  labs(x = "Log-lambda", y = "Overall Brier Score") + # labels
  theme_classic() # classic theme
```



Best Cox Models

- **Show best models**
  - Based on Brier score

```
1 show_best(cox_res, metric = "brier_survival_integrated", n = 5) # top 5 models
```

```
# A tibble: 5 × 8
       penalty .metric          .estimator .eval_time  mean     n std_err .config
         <dbl> <chr>            <chr>           <dbl> <dbl> <int>   <dbl> <chr>
1 0.00147      brier_survival… standard           NA 0.160    10 0.00775 Prepro…
2 0.0000127    brier_survival… standard           NA 0.160    10 0.00775 Prepro…
3 0.0000000105 brier_survival… standard           NA 0.160    10 0.00775 Prepro…
4 0.000754     brier_survival… standard           NA 0.160    10 0.00775 Prepro…
5 0.00000409   brier_survival… standard           NA 0.160    10 0.00775 Prepro…
```

Random Forest Model

- **Random forest specification and workflow**

```
# Random forest model specification
rf_spec <- rand_forest(mtry = tune(), min_n = tune()) |>  # tune mtry and min_n
  set_engine("aorsf") |>  # set engine to aorsf
  set_mode("censored regression") # set mode to censored regression
rf_spec # print model specification
```

```
Random Forest Model Specification (censored regression)

Main Arguments:
  mtry = tune()
  min_n = tune()

Computational engine: aorsf
```

```
# Create a workflow by combining model and recipe
rf_wflow <- workflow() |>
  add_model(rf_spec) |>   # add model specification
  add_recipe(gbc_recipe)   # add recipe
```

Random Forest Tuning

- **Tune the random forest model**
  - Similar to Cox model tuning

```
set.seed(123) # set seed for reproducibility
# Tune the random forest model (this will take some time)
rf_res <- tune_grid(
  rf_wflow,
  resamples = gbc_folds,
  grid = 10, # number of hyperparameter combinations to try
  metrics = gbc_metrics, # evaluation metrics
  eval_time = time_points # evaluation time points
)
```

Random Forest Tuning Results

- **View validation results**

```
1  collect_metrics(rf_res) |> head()   # collect metrics from tuning results
```

```
# A tibble: 6 × 9
   mtry min_n .metric         .estimator .eval_time   mean     n std_err .config
  <int> <int> <chr>           <chr>           <dbl>  <dbl> <int>   <dbl> <chr>
1     3    30 brier_survival  standard            0 0         10 0       Prepro…
2     3    30 roc_auc_surviv… standard            0 0.5       10 0       Prepro…
3     3    30 brier_survival  standard           12 0.0635    10 0.00706 Prepro…
4     3    30 roc_auc_surviv… standard           12 0.827     10 0.0314  Prepro…
5     3    30 brier_survival  standard           24 0.163     10 0.0114  Prepro…
6     3    30 roc_auc_surviv… standard           24 0.747     10 0.0475  Prepro…
```

Best Random Forest Models

- **Show best models**
  - Based on Brier score

```
1  show_best(rf_res, metric = "brier_survival_integrated", n = 5) # top 5 models
```

```
# A tibble: 5 × 9
   mtry min_n .metric          .estimator .eval_time  mean     n std_err .config
  <int> <int> <chr>            <chr>           <dbl> <dbl> <int>   <dbl> <chr>
1     6    24 brier_survival_… standard           NA 0.155    10 0.00765 Prepro…
2     5    27 brier_survival_… standard           NA 0.155    10 0.00782 Prepro…
3     4    20 brier_survival_… standard           NA 0.156    10 0.00777 Prepro…
4     9    36 brier_survival_… standard           NA 0.156    10 0.00743 Prepro…
5     2     7 brier_survival_… standard           NA 0.156    10 0.00829 Prepro…
```

- **Conclusion**
  - Best RF model has lower Brier score than best Cox model

Finalize and Fit Best Model

- **Fit final RF model**

```r
# Select best RF hyperparameters (mtry, min_n) based on Brier score
param_best <- select_best(rf_res, metric = "brier_survival_integrated")
param_best # view results
```

```
# A tibble: 1 × 3
   mtry min_n .config
  <int> <int> <chr>
1     6    24 Preprocessor1_Model07
```

```r
# Finalize the workflow with the best hyperparameters
rf_final_wflow <- finalize_workflow(rf_wflow, param_best) # finalize workflow
# Fit the finalized workflow on the testing set
set.seed(123) # set seed for reproducibility
final_rf_fit <- last_fit(
  rf_final_wflow,
  split = gbc_split, # use the original split
  metrics = gbc_metrics, # evaluation metrics
  eval_time = time_points # evaluation time points
)
```

Test Performance (I)

- **Collect metrics on test data**

```r
collect_metrics(final_rf_fit) |> # collect overall performance metrics
  filter(.metric %in% c("concordance_survival", "brier_survival_integrated"))
```

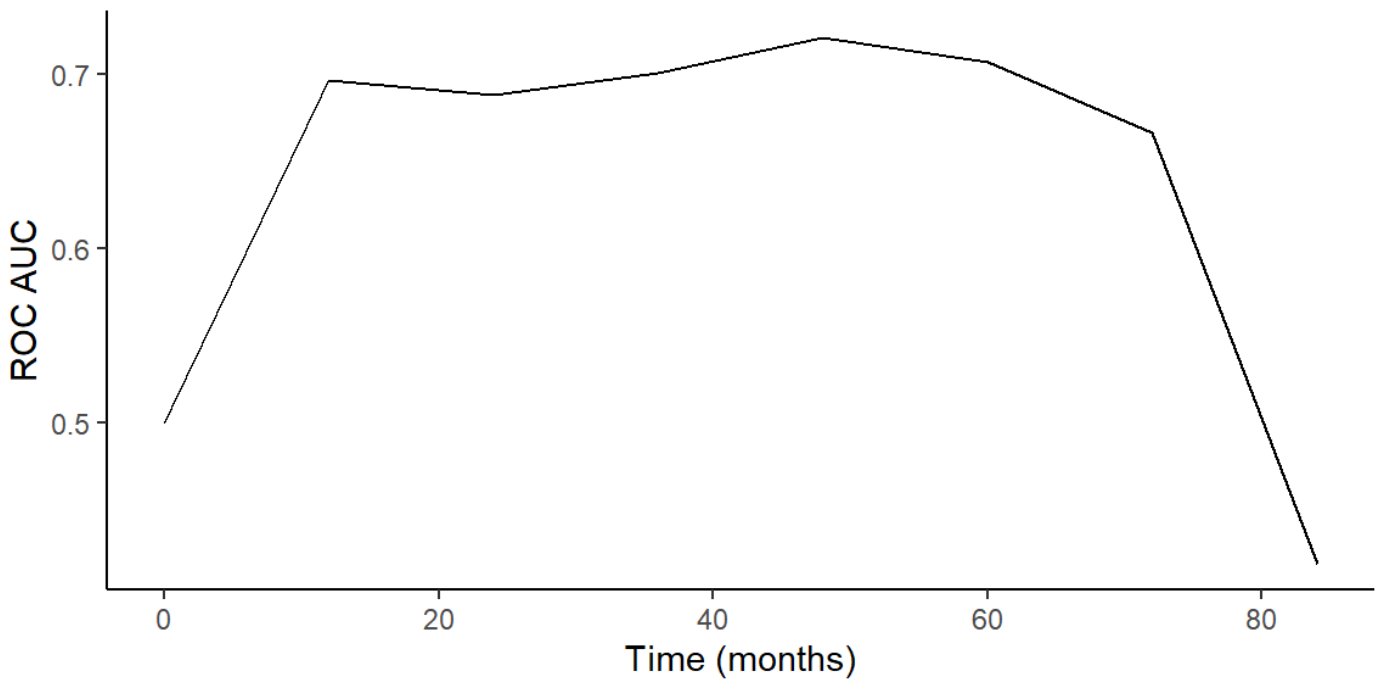
```
# A tibble: 2 × 5
  .metric                   .estimator .eval_time .estimate .config
  <chr>                     <chr>           <dbl>     <dbl> <chr>
1 brier_survival_integrated standard           NA     0.237 Preprocessor1_Model1
2 concordance_survival      standard           NA     0.655 Preprocessor1_Model1
```

```r
# Extract test ROC AUC over time
roc_test <- collect_metrics(final_rf_fit) |>
  filter(.metric == "roc_auc_survival") |>  # filter for ROC AUC
  rename(mean = .estimate) # rename mean column
```

Test Performance (II)

- **Plot test ROC AUC over time**

```
roc_test |>  # pass the test ROC AUC data
  ggplot(aes(.eval_time, mean)) +  # plot evaluation time vs mean ROC AUC
  geom_line() + # plot line
  labs(x = "Time (months)", y = "ROC AUC") + # labels
  theme_classic()
```



Prediction by Final RF Model

- **Extract the fitted workflow**
  - Use `extract_workflow()` to get the final model

```
gbc_rf <- extract_workflow(final_rf_fit) # extract the fitted workflow
# Predict on new data
gbc_5 <- testing(gbc_split) |>  slice(1:5) # take first 5 rows of test data
predict(gbc_rf, new_data = gbc_5, type = "time") # predict survival times
```

```
# A tibble: 5 × 1
  .pred_time
       <dbl>
1       47.6
2       67.1
3       67.2
4       36.1
5       49.7
```

Cox Model Exercise (I)

- **Task**: extract the best Cox model from `cox_res` and fit it to test data

▶ Solution

Cox Model Exercise (II)

- Solution - continued

Cox Model Exercise (III)

- **Task**: find the parameter estimates of final Cox model
  - Hint: use `tidy()` function from `broom` package

▶ Solution

Survival Tree Exercise

- **Task**: fit a survival tree model to the GBC data
  - Use `decision_tree()` with `set_engine("rpart")`
  - Tune complexity parameter `cp` using `tune()`
  - Use the same recipe as for Cox and RF models
  - Evaluate performance using Brier score and ROC AUC

Summary

Key Takeaways

- **Machine learning**: powerful tools for survival analysis with many covariates
  - Regularized Cox regression, survival trees, and random forests
- `tidymodels`: a consistent interface for modeling and machine learning
  - `parsnip` for model specification and tuning
  - `censored` packages for survival data
  - **Model evaluation**: Brier score and ROC AUC for survival models