Introduction to the R “Survival” Library and Other Useful Libraries to Run Advanced Survival Analysis Models

**This document is part of a resource ‘Advanced Topics in Survival Analysis’ on** [**https://www.ncrm.ac.uk/resources/online/all/?id=20860**](https://www.ncrm.ac.uk/resources/online/all/?id=20860)

I have used R to create Survival Analysis outputs during the presentations for these resources, and I have also used R to develop exercises. In this document I will provide a short overview of R libraries and commands that I have used.

# The “Survival” Library

The main library is “survival”, which you can install and load using the following commands:

install.package(“survival”)

library(survival)

You can check this repository for more information:

<https://cran.r-project.org/web/packages/survival/index.html>

# Create Survival Objects

Once installed, you can use the package to create a *survival object*, i.e. ensure that the dataset you are using is read as a dataset for survival analysis.

For example, using the dataset employed as an exemplar in Chapter 10 of Singer and Willett’s (2003) book “*Applied Longitudinal Data Analysis”*, you can get the “teachers” dataset and prepare it by using the Surv command from the *survival* library:

library(survival)

*# Preliminary steps to create Table 10.1 on page 327.*

teachers<-read.table("https://stats.idre.ucla.edu/stat/examples/alda/teachers.csv", sep=",", header=T)

*# Create a variable to indicate whether the event took place or not, based on censoring:*

teachers$event <- ifelse(teachers$censor == 1, 0, 1)

*# check the new variable is correctly specified*

table(teachers$event, teachers$censor)

*#create a survival object:*

**survobj<-Surv(teachers$t, teachers$event)**

The final line creates a survival object I called “survobj” using the command Surv : the first argument specifies the time variable in the survival analysis dataset, while the second argument specifies the variable indicating the event.

See the UCLA Advanced Research Computing web-pages for a slightly different example:

<https://stats.oarc.ucla.edu/r/examples/alda/r-applied-longitudinal-data-analysis-ch-10/>

If you want to check what the survival object looks like you can use:

survobj[1:15]

which will show this output:

[1] 1 2 1 1 12+ 1 12+ 1 2 2 7 12+ 1 12+ 12+

The output lists the first 15 cases in the dataset. The variable reported is the timing of the event of interest. In some cases, the timing is followed by the “+” sign: this indicates the case is censored.

# Estimate Key Survival Analysis Statistics

Once you have created a survival object, you can use this as an argument to create survival curves and collate other relevant information. To this end, the “survival” library offers the survfit function. Following from the previous commands, you can use the survival object created as follows:

ts <- survfit(survobj ~ 1, data = teachers)

The line above creates the “ts” object. The ~ 1 indicates that the survival curve should be estimated for the entire dataset as a whole, without stratifying by any covariates or groups. (Covariates can be included by substituting 1 with the covariate(s) of interest.)

You will see that the object “ts” contains a set of information about the model:

A screenshot of a computer program

Description automatically generated

For example, the number of cases in the dataset (ts$n), the time intervals (ts$time), the risk set within each time interval (ts$n.risk), the number of events observed in each time interval (ts$n.event), the number of censored cases within each time interval (ts$n.censor), the survival function for each time interval (ts$surv) and other information (e.g., the 95% confidence intervals of the survival function based on the Kaplan-Meier formula).

When we want to create a life table or plot the survival function, we can extract the information obtained through the survfit function.

However, the survfit function does not estimate the hazard function, but just the cumulative hazard function. To calculate the hazard function we can resort to its formula, where for each time interval *j*, the hazard function is estimated as:

Therefore, we can simply calculate the hazard function using the information in the survival function estimate created above:

h<-ts$n.event/ts$n.risk

Once we estimate the hazard function, we can include it in a dataset to report it in the life tables or to plot it.

Further examples of life tables and plotting the estimated functions based on Singer and Willatt’s (2003) book are available here:

<https://stats.oarc.ucla.edu/r/examples/alda/r-applied-longitudinal-data-analysis-ch-10/>

and here:

<https://stats.oarc.ucla.edu/other/examples/alda/>

# Estimate Key Survival Analysis Statistics: Kaplan-Meier Method

When the time variable used to measure event occurrence is continuous (e.g., hours, days, etc.), the survfit function does provide statistics using the Kaplan-Meier method. See the following page for more details:

<https://www.rdocumentation.org/packages/survival/versions/2.11-4/topics/survfit>

To provide an example, you can refer to Chapter 13 of Singer & Willett’s book, see the link:

<https://stats.oarc.ucla.edu/r/examples/alda/r-applied-longitudinal-data-analysis-ch-13/>

and run the following:

library(survival)

surv\_obj <- Surv(honk$SECONDS, honk$event)

km\_fit <- survfit(Surv(SECONDS, event) ~ 1, data = honk)

The “km\_fit” object created will include these variables:

A computer screen shot of a number

AI-generated content may be incorrect.

The survfit function therefore provides survivor function estimates using the Kaplan-Meier method (called “surv”), as well as their standard errors (“std.err”, lower and upper bound 95% CIs. It also provides the cumulative hazard function estimates, called “cumhaz”: these are estimated using the Nelson-Aalen method by default.

# Plot Survival Analysis Statistics

While the “survival” library allows to plot key statistics, another library, called “survminer” allows more options based on the ggplot library. Using the example from Chapter 13 just described above, the “km\_fit” can be taken to create a customised graph of the Kaplan-Meier survivor function estimate with 95% CIs:

library(survminer)

# Plot survival with custom CI color and line size

gsurv<-ggsurvplot(

km\_fit,

conf.int = TRUE,

conf.int.fill = "lightblue", # Your custom CI color (e.g., orange)

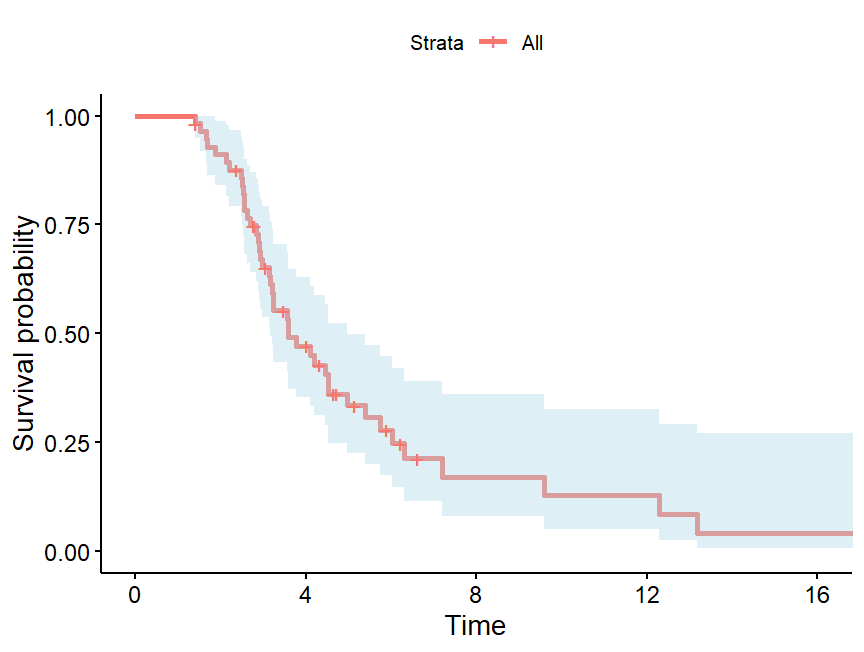
conf.int.alpha = 0.4, # Transparency of CI

palette = "#2E9Fdfex", # Line color (e.g., blue)

size = 1.2 # Line thickness

)

print(gsurv)



The “survminer” library can also be used to plot other functions, e.g. the Nelson-Aalen cumulative hazard function:

# Plot cumulative hazard with custom CI color and line size

gcumhaz<-ggsurvplot(

km\_fit,

**fun = "cumhaz", #invokes the estimated cumulative hazards**

conf.int = TRUE,

conf.int.fill = "#FFB347", # Your custom CI color (e.g., orange)

conf.int.alpha = 0.4, # Transparency of CI

palette = "#2E9Fdfex", # Line color (e.g., blue)

size = 1.5 # Line thickness

)

# Estimate and Plot Kernel-Smoothed Hazard Functions

The “muhaz” library in R is a package designed for hazard function estimation in survival analysis, specifically from right-censored data. It produces a smooth estimate of the hazard function using kernel-based methods.

See the link below for more information:

<https://cran.r-project.org/web/packages/muhaz/index.html>

The option bw.method = "global" means the same bandwidth will be applied for all grid points. If bw.method="global" and bw.grid has one component only, no MSE minimization is performed, and the hazard estimates are computed for the value of bw.grid. Thus, it is possible to specify different bandwidths (e.g., 2, 5, and 10 seconds in the dataset “honk” mentioned above). Once these are estimated, they can be plotted using ggplot or other libraries. Please see the link for more details on bandwidths:

<https://www.rdocumentation.org/packages/muhaz/versions/1.2.6.4/topics/muhaz>

# Estimate Key Survival Analysis Statistics by Groups

Once you have created a survival object, and invoke the survfit function, you can ask to estimate statistics by groups by including the predictor you want after the ~ symbol. I refer to the example from Singer & Willatt’s chapter 14 (see <https://stats.oarc.ucla.edu/r/examples/alda/r-applied-longitudinal-data-analysis-ch-14/>):

rearrest<-read.table("https://stats.idre.ucla.edu/stat/examples/alda/rearrest.csv", sep=",", header=T)

df <-

rearrest %>%

mutate(

status = recode(censor, `1` = 0, `0` = 1)

)

# check the dataset

head(df[, c("months", "status", "censor")])

#summary statistics of time

**km\_fit <- survfit(Surv(months, status) ~ personal, data = df)**

summary(km\_fit)

The “km\_fit” object now holds survival statistics for participants with different values of the predictor “personal”. The last line in the window above will show a summary by groups, which the software calls “strata” in order to indicate that the statistics are now stratified according to values of the predictor. Using the last line in the window above you can also retrieve the median lifetime for the different groups or strata.

Once the “km\_fit” object has been created, it can be used to plot the key statistics by group. For example, using the “survminer” library, we can display the survivor functions according to values of the “personal” predictor:

# Create a customized plot

library(survminer)

g1<-ggsurvplot(

km\_fit,

data = df,

conf.int = FALSE, # DO not Show 95% confidence intervals

palette = c("#E7B800", "#2E9FDF"), # Custom colors for lines

size = 1.2, # Change line thickness

xlab = "Days",

ylab = "Survival probability",

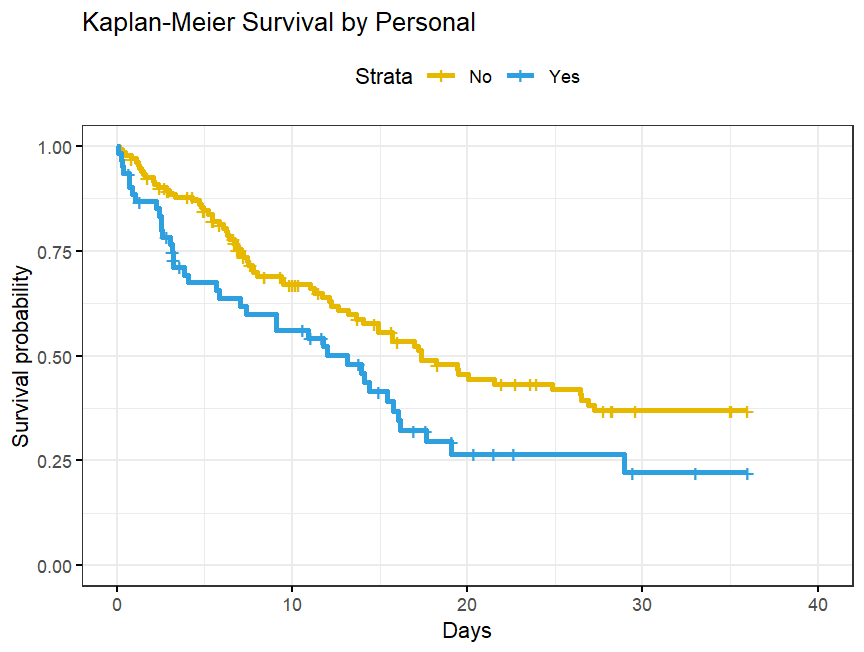
legend.labs = c("No", "Yes"), # Adjust labels as appropriate

ggtheme = theme\_bw(), # Clean background

title = "Kaplan-Meier Survival by Personal"

)

print(g1)



# Run Cox Regression Model

The Cox regression model can be run using the “coxph” option from the “survival” library, please see <https://cran.r-project.org/package=survival/survival.pdf>

Using the same example from Singer & Willatt’s Chapter 14, you can run a Cox regression model using predictor “personal” in this way:

rearrest<-read.table("https://stats.idre.ucla.edu/stat/examples/alda/rearrest.csv", sep=",", header=T)

df <-

rearrest %>%

mutate(

status = recode(censor, `1` = 0, `0` = 1)

)

# check the dataset

head(df[, c("months", "status", "censor")])

attach(rearrest)

tab14.1A <- coxph(Surv(months, status)~personal, data = df)

summary(tab14.1A)

The last line above will show the key coefficients estimated by the model. Note that “exp(coef)” is the Hazard Ratio (HR) estimate.

The command “coxph” allows to change default options. One issue to consider is, for example, the method used to deal with ties in the data. Ties are instances where two participants or more experience the occurrence of the target event in the same interval. If the dataset contains a large number of ties, this issue can create significant issues in model estimation. This problem is more likely when the time intervals are coarser. The “coxph” command uses the “Efron” approximation by default. However, this can be changed by other options. For example, option ties=”breslow” will involve the Breslow approximation. There is also another option, the “exact partial likelihood”, which is equivalent to a conditional logistic model, and is appropriate when the times are a small set of discrete values.

If we want to add other predictors to the Cox regression model, we can specify these after the ~ symbol. However, we also need to specify these predictors are representing additive effects: for example, if we wanted to use the example of the “rearrest” dataset above and add a further independent predictor like centred-age, we will specify this as follows:

tab14.1B <- coxph(Surv(months, status)~personal+cage, data = df)

summary(tab14.1B)

The summary command will return key statistics that show the effect of the predictor “personal” on the timing of rearrest, as well as the effect of the age predictor (centred at the participants’ mean age) on the timing of rearrest.

I will finally draw your attention on the summary statistics provided after running “coxph”. You may notice that these statistics include the likelihood ratio test and more. Indeed, when running models using “coxph”, the software will also estimate statistics that cane be used for model comparison. For example, we can retrieve the log-likelihood and Bayesian Information Criterion (BIC) of the last model with two predictors in this way:

tab14.1B <- coxph(Surv(months, status)~personal+cage, data = df)

summary(tab14.1B)

logLik(tab14.1B)

BIC(tab14.1B)

# Recovering the Baseline functions

The “coxph” command also allows to retrieve the baseline functions of a “prototypical” participant that displays values equal to zero in all the covariates in the model.

I will still use the example of the “rearrest” database from Singer & Willatt’s Chapter 14, I also used above. Assume we ran a model with three covariates, “personal”, “property”, and “cage”, i.e. age centred at the participants’ mean:

tab14.1D <- coxph(Surv(months, event))~personal + property + cage)

We then can create a “baseline” dataframe whereby we assign the values 0 to all the covariates in the model:

baseline <- data.frame(personal=0, property=0, cage=0)

Now we can use the coefficients from the estimated model tab14.D to estimate the statistics that a “baseline” participant will obtain:

s.baseline <- survfit(tab14.1D, newdata = baseline)

What the line above is basically doing is applying the coefficients from the estimated model to the “baseline” dataset, whereby all values of the covariates are set to be equal to zero.

Once we have the “s.baseline” object, we can report and plot key statistics like the survivor function, etc.

Using this template, we can create other “prototypical” participants. For example, if we wanted to represent the statistics expected (based on model tab14.D) for a participant that ticks the “personal” and “property” predictors and is 10 years older than the sample average, we could run these commands:

perpro10yrs <- data.frame(personal=1, property=1, cage=10)

s.perpro10yrs <- survfit(tab14.1D, newdata = perpro10yrs)