Survival Analysis with R: Cheat Sheet

Packages

The **tidyverse**, **survival**, and **survminer** packages from CRAN are required for this lesson. For looking at TCGA data, you'll also need **RTCGA**, **RTCGA.clinical**, and **RTCGA.mRNA** from Bioconductor. Here's how to install them (do this once).

```r
# Install from CRAN -------------------------
install.packages("tidyverse")
install.packages("survminer")

# Install from Bioconductor -----------------
# Install Bioconductor core packages first
source("http://bioconductor.org/biocLite.R")
biocLite()
# Next install RTCGA and RTCGA data packages
biocLite("RTCGA")
biocLite("RTCGA.clinical")
biocLite("RTCGA.mRNA")
```

After installing these packages, load them in each session you use them.

```r
# CRAN Packages needed
library(tidyverse)   # loads dplyr, ggplot2, readr, etc.
library(survival)    # core survival analysis functions
library(survminer)   # recommended for visualizing survival curves

# Bioconductor packages
library(RTCGA)
library(RTCGA.clinical)
library(RTCGA.mRNA)
```

Functions

Function	Description
`head(df)` ; `tail(df)`	Print first and last few rows of data frame `df`
`View(df)`	View tabular data frame `df` in a graphical viewer
`filter(df, ..,)`	Filters data according to condition ... (dplyr)
`Surv(df$time, df$event)`	Creates a survival object w/ right-censored data
`survfit(Surv(time, status)~x, data=df)`	Creates a survival curve against variable `x`
`summary(sfit, times=c(0,10,50))`	Shows life table for `sfit` object at specified times
`survdiff(Surv(time, status)~x, data=df)`	Log-rank test of differential survival by groups in `x`
`coxph(Surv(time, status)~x1+x2, data=df)`	Run a Cox PH model on variables `x1` and `x2`
`tidy()` ; `augment()` ; `glance()`	Model tidying functions in the broom package
`survivalTCGA(..., extract.cols=...)`	Extract survival data from 1+ (R)TCGA clinical datasets
`expressionsTCGA(..., extract.cols=...)`	Extract gene expression data (R)TCGA mRNA datasets

The pipe: %>%

When you load the **dplyr** or **tidyverse** library you can use **%>%**, the *pipe*. Running **x %>% f(args)** is the same as **f(x, args)**. If you wanted to run function `f()` on data `x`, then run function `g()` on that, then run function `h()` on that result: instead of nesting multiple functions, **h(g(f(x)))**, it's preferable and more readable to create a chain or pipeline of functions: **x %>% f %>% g %>% h**. Pipelines can be spread across multiple lines, with each line ending in **%>%** until the pipeline terminates. The keyboard shortcut for inserting **%>%** is Cmd+Shift+M on Mac, Ctrl+Shift+M on Windows.