Article from
Actuary of the Future
November 2019
Issue 45

Python Libraries for Mortality Modeling

By Carlos Brioso

Python is the leading programming language for data science. Python programming is one skill that actuaries should acquire. It facilitates analytic work, and the available visualization tools help to communicate ideas. In this article, I use a traditional actuarial problem, mortality modeling, as an example.

Until now, R language has been the preferred tool for survival analysis since there are well-known packages, such as Survival and Survminer. Python packages, like Lifelines and Scikit-survival, are newer and powerful. Lifelines, originally developed by Shopify, is an implementation of survival analysis in Python. This library is built on top of Pandas, the most used library for data manipulation.

I will use the Old-Age Mortality Scania dataset for our analyses. This data consists of old-age life histories from Jan. 1, 1813, through Dec. 31, 1894. Only life histories above age 50 are considered. The information recorded in this dataset is shown in Figure 1.

Figure 1
Old-Age Mortality Scania Dataset

ID	Enter	Exit	Event	Birthdate	Sex	Parish	Ses	Immigrant	Duration
1	50	59.242	1	1781.454	male	1	lower	no	9.242
2	50	53.539	0	1821.350	male	1	lower	yes	3.539
...	...	...	...	...	...	...	...	...	...
4	50	51.280	0	1822.713	male	1	lower	yes	1.280
5	50	70.776	1	1813.547	female	1	lower	yes	20.776

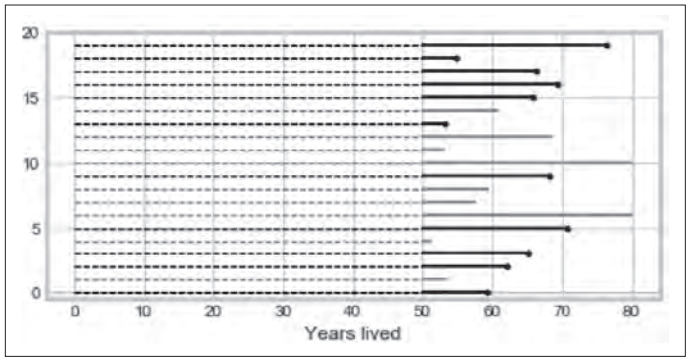
Data Dictionary

Enter: Start age for the interval.
Exit: Stop age for the interval.
Event: Indicator of death; equals TRUE if the person died at the end of the interval, FALSE otherwise.
Birthdate: Birthdate as a real number (i.e., "1765-06-27" is 1765.490).
Sex: Gender, a factor with levels male female.
Parish: One of five parishes in Scania, coded 1, 2, 3, 4, 5. Factor.
Ses: Socio-economic status at age 50, a factor with levels upper and lower.
Immigrant: a factor with levels no region and yes.

Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

To understand the data, it is useful to visualize the events and censored observations. In Figure 2, the light lines represent people who have not died, while dark lines show people who died in the observation period.

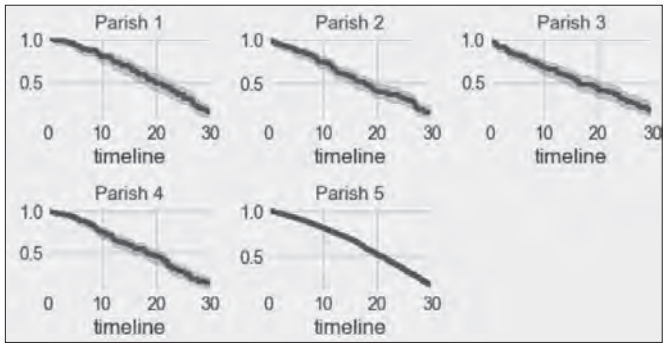
Figure 2
Events and Censored Observations



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

We can also plot the Kaplan-Meier estimator for the whole population or for specific subpopulations. Figure 3 shows survival curves for the five parishes in our dataset as well as their confidence intervals.

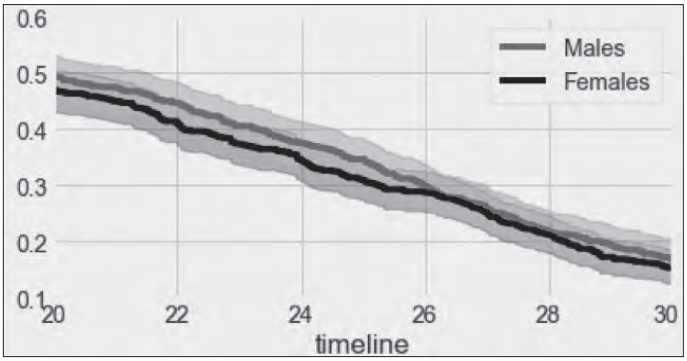
Figure 3
Survival by Parish



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

We may be also interested in looking at survival curves of two different groups in one plot. For example, we can plot the survival curve by gender for times 20 to 30 (see Figure 4). We observe that the survival of men is higher than for women, but this difference becomes less significant over time. However, there is a significant overlap between the confidence intervals of these curves.

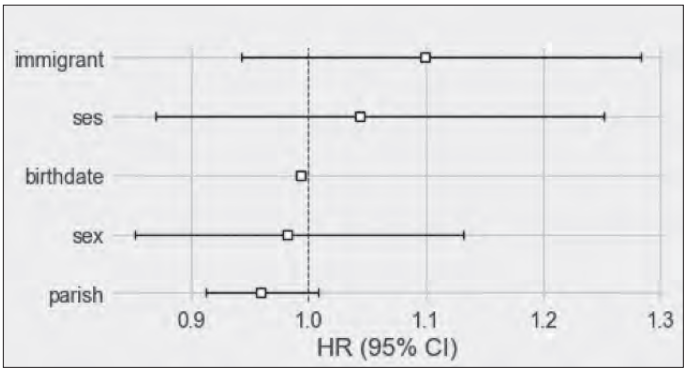
Figure 4
Lifespans by Gender



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

After performing univariate analysis, we may be interested in multivariate analysis. The most traditional multivariate regression for survival data is the Cox's Proportional Hazard (PH) Model. In this model, the log hazard of an individual is a linear function of static covariates and a population-level baseline hazard that changes over time. Figure 5 shows the values of the PH coefficients and their confidence intervals.

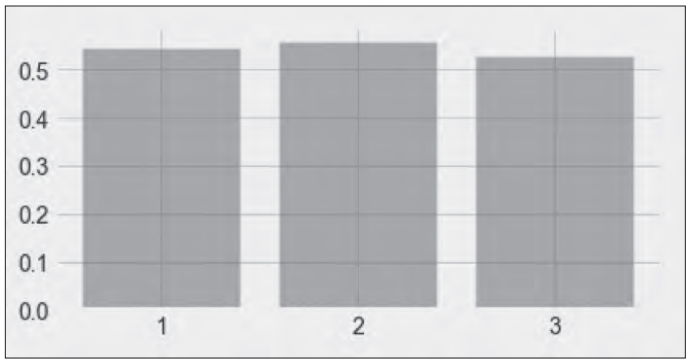
Figure 5
Cox PH Coefficient



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

Lifelines also offers functions to perform cross-validation. Cross-validation is one of the most important techniques to validate the stability of model performance. In Figure 6, we observe that the Concordance-Index, the global index for validating the predictive ability of a survival model (the higher the better), is stable across the (three) cross-validation samples.

Figure 6
Concordance Index for Three-fold Cross-Validation



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

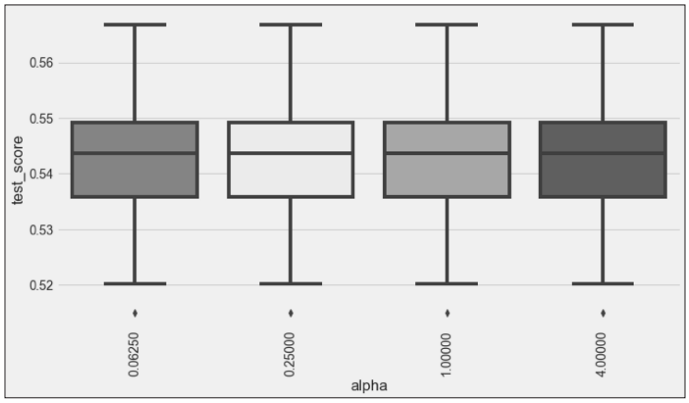
Lifelines offers several parametric and nonparametric models: Aalen Additive, Cox Time Varying, Cox PH, Weibull Accelerated Failure Time (AFT), Log-Normal AFT, Log-Logistic AFT, and Piecewise Exponential Regression. However, Lifelines does not offer the more recent machine learning algorithms.

Scikit-survival is a Python module for survival analysis built on top of Scikit-learn, the most popular library for training machine learning models. Therefore, one can perform survival analysis while utilizing the Scikit-learn utilities for preprocessing and doing cross-validation. Scikit-survival supports the following models: Survival Support Vector Machine, Gradient Boosting Survival Analysis, Cox PH Model and Cox PH Model with elastic net penalty. In addition, this library implements additional model metrics: Concordance Index for right-censored data based on inverse probability of censoring weights and cumulative/dynamic Area Under Curve for right-censored time-to-event data.

Training machine learning models require finding a set of parameters that optimize their performance (hyperparameter optimization). One can use Scikit-learn functions to achieve that.

For example, for the Survival Support Vector Machine Model (see Figure 7), the alpha parameter needs to be optimized. We used the Grid Search Cross-Validation technique and estimated the optimal alpha parameter (0.0625, corresponding to a Concordance Index of 0.542).

Figure 7
Concordance Index—Alpha Selection



Source: Data from *https://www.rdocumentation.org/packages/eha/versions/2.6.0/topics/scania*

One additional package to consider is DeepSurv (by Jared Katzman), which implements a deep learning generalization of the Cox PH Model.

The choice of the appropriate models and methods is not a trivial problem. The example used in this article does not help to appreciate the differences in performance among the models at our disposal because of the small sample size and the limited number of covariates. If we have a larger dataset with more covariates, our analysis will become more complex and iterative; let's use Python to simplify this challenging but interesting work. ■

Carlos Brioso, FSA, CERA, is a director with the Center for Data Science and AI at New York Life. He can be reached at *carlos_brioso@newyorklife.com.*