Introduction to Survival Analysis

Reza Rashetnia, PhD · Follow

Published in Nerd For Tech · 34 min read · Jan 20, 2021

Survival analysis is a statistical method for predicting the expected duration of time until certain events happen. Such events could be death, removal, churn, damage, failure, redemption, expiration, etc. The most common

parameter in all these event is time which is the prominent part of any survival analysis. Survival analysis always try to measure probability, detect portion of population or evaluate effect of particular circumstances on **survival** over the event at certain **time.** This post try to cover most important details, mathematical intuition and theories, and whole process for **survival curves** and **survival regressions.**

Programming preference

LIFELINES: Survival Analysis in Python

LifeLines library in Python is used in this post. Lifelines is a complete survival analysis library, written in pure Python with benefits of:

1. easy installation
2. internal plotting methods
3. simple and intuitive API
4. handles right, left and interval censored data
5. contains the most popular parametric, semi-parametric and non-parametric models

Ch 0. Introduction

0.0 Application

When studying the occurrence of some event in a population of subjects, if the time until the occurrence of the event were unimportant, the event could be analyzed as a binary outcome using the logistic regression model. For some other events, the time until the event is important. Survival analysis is used to analyze data in which the time until the event is of interest. The response variable is the time until that event and is often called a failure

time, survival time, or event time. Customer churn is the event of a customer opting out of his subscription which time is matter. Businesses want to understand how and why their customers churn to improve their profits. Here, after investigating **survival analysis**, I'll explore it over customer churn and renewal.

0.1 Censoring

At the time you want to make inferences about durations, it is possible that not all the events have occurred yet. For example, a medical professional will not wait 50 years for each individual in the study to pass away before investigating — he or she is interested in making decisions after only a few years, or months possibly. The individuals in a population who have not been subject to the death event are labeled as right-censored, i.e., we did not (or can not) view the rest of their life history due to some external circumstances. All the information we have on these individuals are their current lifetime durations (which is naturally less than their actual lifetimes). A common mistake data analysts make is choosing to ignore the right-censored individuals. We will see why this is a mistake next.

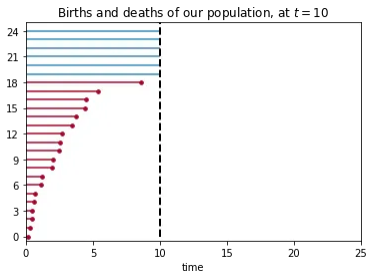
Consider a case where the population is actually made up of two subpopulations, A and B. Population A has a very small lifespan, say 2 months on average, and population B enjoys a much larger lifespan, say 12 months on average. We don't know this distinction beforehand. At t=10, we wish to investigate the average lifespan for the entire population.

In the figure below, the red lines denote the lifespan of individuals where the death event has been observed, and the blue lines denote the lifespan of the right-censored individuals (deaths have not been observed). If we are asked to estimate the average lifetime of our population, and we naively decided to

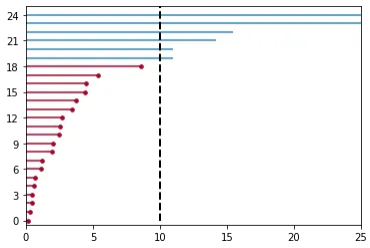
not included the right-censored individuals, it is clear that we would be severely underestimating the true average lifespan.

In [1]:
```
from lifelines.plotting import plot_lifetimes
import numpy as np
from numpy.random import uniform, exponential

N = 25

CURRENT_TIME = 10

actual_lifetimes = np.array([
    exponential(12) if (uniform() < 0.5) else exponential(2) for i in range(N)
])
observed_lifetimes = np.minimum(actual_lifetimes, CURRENT_TIME)
death_observed = actual_lifetimes < CURRENT_TIME

ax = plot_lifetimes(observed_lifetimes, event_observed=death_observed)

ax.set_xlim(0, 25)
ax.vlines(10, 0, 30, lw=2, linestyles='--')
ax.set_xlabel("time")
ax.set_title("Births and deaths of our population, at $t=10$")
print("Observed lifetimes at time %d:\n" % (CURRENT_TIME), observed_lifetimes)
```

```
Observed lifetimes at time 10:
 [ 1.1947688  10.          2.44345568  5.36683806  3.70008185
0.48173596
  0.13144326  0.45511328 10.          0.69205379 10.         10.
  0.59919553  1.97439122  3.43325156  1.09599444  4.38248954
4.47287191
  8.59646432 10.          2.66794281  2.5696101   0.27496312 10.
  2.006956  ]
```



```
In [2]: ax = plot_lifetimes(actual_lifetimes, event_observed=death_observed)
        ax.vlines(10, 0, 30, lw=2, linestyles='--')
        ax.set_xlim(0, 25)
```



Originally, Survival analysis was developed to solved censoring problems which is to estimate when our data is right_censored. However, even in the case where all events have been observed, i.e. there is no censoring, survival analysis is still a very useful tool to understand durations and rates.

The observations need not always start at zero, either. This was done only for understanding in the above example. Consider the example where a customer entering a store is a birth: a customer can enter at any time, and not necessarily at time zero. In survival analysis, durations are relative: individuals may start at different times. (We actually only need the duration of the observation, and not necessarily the start and end time.)

0.2 Mathematical intuition

- What is the ultimate survival input/output?
- **Kaplan-Meier curve (Survival curves):**
- Input: duration (start date, end date if available) for each customer and whether they have failed/churned/died.
- Output: aggregate survival curve
- **Survival regression: Cox proportional hazard model:**

- Input: covariates (features) + Kaplan-Meier curve input data
- Output: survival probability for individual customers
- Use cases of survival analysis
- Customer Analytics (Customer Retention): With the help of Survival Analysis we can focus on churn prevention efforts of high-value customers with low survival time. This analysis also helps us to calculate Customer Life Time Value. In this use case, event is defined as the time at which the customer churns/unsubscribe.
- Marketing Analytics (Cohort Analysis): Survival Analysis evaluates the retention rates of each marketing channel. In this case, event is defined as the time at which the customer unsubscribe a marketing channel.
- Actuaries: given the risks of a population, survival analysis evaluates the probability of the population to die in a particular time range. This analysis helps the insurance companies to evaluate the insurance premiums.

Lets assume a **non-negative continuous random variable** T, representing the time until some event of interest. For example:

- the time from the customer's subscription to the customer churn, the time from start of a machine to its breakdown, the time from diagnosis of a disease until death. (T is the time from customer's (a randomly selected customer) subscription to the customer churn; T is the time from start of a randomly selected machine to its breakdown; T is the time from diagnosis of a disease until death of a randomly selected patient.)
- T is continuous, non-negative random variable ($T \geq 0$).

- For such random variables, probability density function (pdf) and cumulative distribution function (cdf) are commonly used to characterize their distribution. Thus, we will assume that this random variable has a probability density function f(t) , and cumulative distribution function F(t):

$$F(t) = P(T < t) = \int_0^t f(t')dt'$$

<> Survival Function S(t)

- Gives us the probability that the event has not occurred by the time t. In simple words, S(t) gives us the proportion of population with the time to event value more than t.

$$S(t) = P(T \geq t) = 1 - F(t) = \int_t^\infty f(t')dt'$$

T_q: q th quantile is the time by which a fraction q of the subjects will fail. It is the value t such that S(t)=1-q or T_q=S^{-1}(1-q).

- T_{0.5}: the median life length is the time by which half the subjects will fail, S(t)=0.5 or T_{0.5}=S^{-1}(0.5).
- Let T_i denote the response for the ith subject. This response is the time until the event of interest, and it may be censored if the subject is not

followed long enough for the event to be observed.

- Let C_i denote the censoring time for the ith subject, and define the event indicator as
- e_i=1 if the event was observed (T_i ≤C_i),
- e_i=0 if the response was censored (T_i>C_i)
- The observed response is Y_i=\min(T_i, C_i), which is the time that occurred first: the failure time or the censoring time.
- The pair of values (Y_i, e_i) contains all the response information for most purposes (i.e., the potential censoring time C_i is not usually of interest if the event occurred before C_i).

<> Hazard Function h(t)

- It is the rate at which event is taking place, out of the surviving population at any given time t. In medical terms, we can define it as "**out of the people who survived at time t, what is the rate of dying of those people**":

$$h(t) = \frac{1}{S(t)} \lim_{dt \to 0} \frac{S(t) - S(t+dt)}{dt} = \frac{f(t)}{S(t)} = -\frac{S'(t)}{S(t)}$$

- The hazard at time t is related to the probability that the event will occur in a small interval around t, given that the event has not occurred before time t. The hazard function is not a density or a probability. However, we can think of it as the probability of failure in an infinitesimal small time period between t and t+dt given that the subject has survived up till time t. In this sense, the hazard is a measure of risk:

the greater the hazard between times t_1 and t_2, the greater the risk of failure in this time interval.

<> Cumulative hazard function

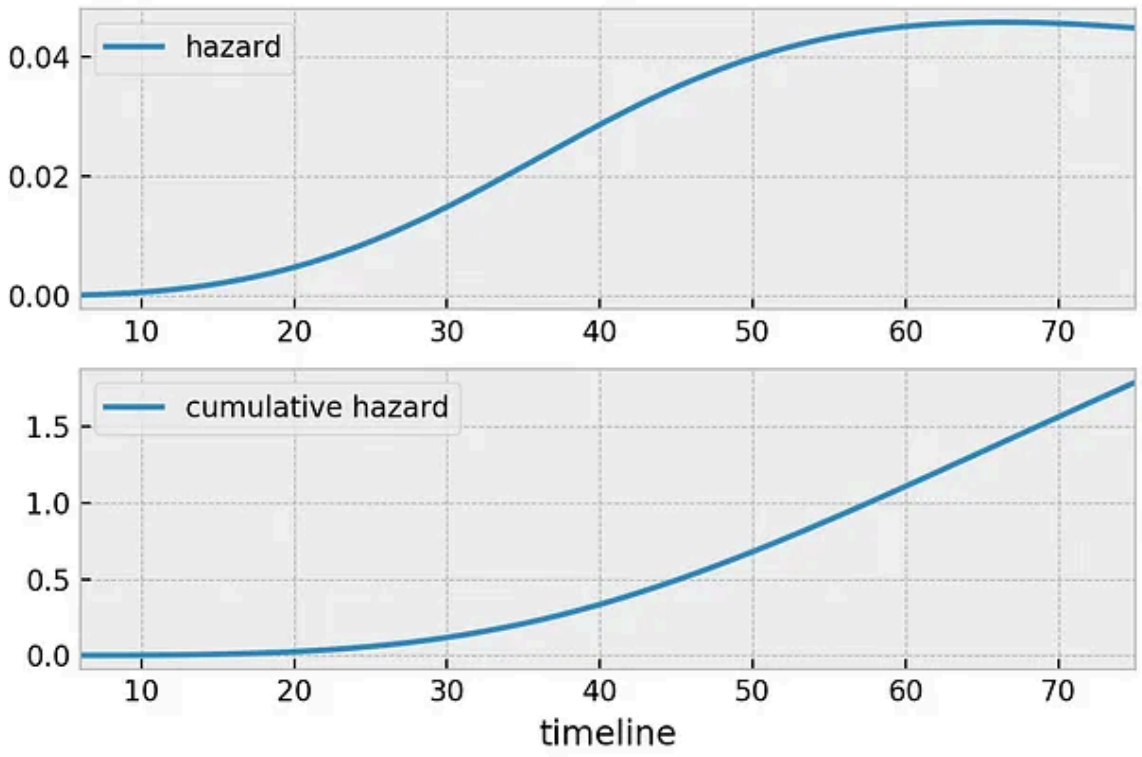
- *H(t)* and S(t): H(t) = \int_0^t h(t')dt', therefore:

$$S(t) = e^{-\int_0^t h(s)ds} = e^{-H(t)}$$

In Cox model

$$H(t|X) = e^{x^T\beta} \int_0^{t_0} h(s)ds = e^{x^T\beta} H_0(t)$$

- The expected value of H(t) is unity.



Ch 1. Estimating univariate models (Survival Curves)

The most common approach to estimate the survival function S(t) in univariate models is the Non-parametric Kaplan-Meier estimator. Let's start with Kaplan-Meier estimator then.

1.0 Non-parametric Kaplan-Meier estimator

Since we don't have the true survival curve of the population, thus we will estimate the survival curve from the data using Non-parametric Kaplan-Meier estimator

$$\hat{S}(t) = \prod_{i:t_i \leq t} \frac{n_i - d_i}{n_i}$$

- n_i: population at risk at time just prior to time t_i
- d_i: number of events occurred at time t_i.

Lets bring an example:

we will be investigating the lifetimes of political leaders around the world. A political leader, in this case, is defined by a single individual's time in office who controls the ruling regime. This political leader could be an elected president, unelected dictator, monarch, etc. **The birth event is the start of the individual's tenure, and the death event is the retirement of the individual. Censoring can occur if they are a) still in offices at the time of dataset compilation (2008), or b) die while in power (this includes assassinations).**

For example, the Bush regime began in 2000 and officially ended in 2008 upon his retirement, thus the regime's lifespan was eight years, and there was a "death" event observed. On the other hand, the JFK regime lasted 2 years, from 1961 and 1963, and the regime's official death event was not observed — JFK died before his official retirement.

(This is an example that has gladly redefined the birth and death events, and in fact completely flips the idea upside down by using deaths as the censoring event. This is also an example where the current time is not the only cause of censoring; there are the alternative events (e.g., death in office) that can be the cause of censoring.

In [3]:
```
from lifelines.datasets import load_dd

data = load_dd()
data.head()
```

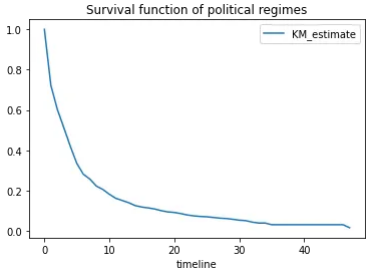
	ctryname	cowcode2	politycode	un_region_name	un_continent_name	ehead	leaderspellreg	democracy	regime	start_year	duration	observed
0	Afghanistan	700	700.0	Southern Asia	Asia	Mohammad Zahir Shah	Mohammad Zahir Shah.Afghanistan.1946.1952.Mona...	Non-democracy	Monarchy	1946	7	1
1	Afghanistan	700	700.0	Southern Asia	Asia	Sardar Mohammad Daoud	Sardar Mohammad Daoud.Afghanistan.1953.1962.Ci...	Non-democracy	Civilian Dict	1953	10	1
2	Afghanistan	700	700.0	Southern Asia	Asia	Mohammad Zahir Shah	Mohammad Zahir Shah.Afghanistan.1963.1972.Mona...	Non-democracy	Monarchy	1963	10	1
3	Afghanistan	700	700.0	Southern Asia	Asia	Sardar Mohammad Daoud	Sardar Mohammad Daoud.Afghanistan.1973.1977.Ci...	Non-democracy	Civilian Dict	1973	5	0
4	Afghanistan	700	700.0	Southern Asia	Asia	Nur Mohammad Taraki	Nur Mohammad Taraki.Afghanistan.1978.1978.Civi...	Non-democracy	Civilian Dict	1978	1	0

From the lifelines library, we'll need the KaplanMeierFitter for this exercise:

```
In [4]: from lifelines import KaplanMeierFitter
        kmf = KaplanMeierFitter()

        T = data["duration"]
        E = data["observed"]

        kmf.fit(T, event_observed=E)
```

```
<lifelines.KaplanMeierFitter:"KM_estimate", fitted with 1808 total
observations, 340 right-censored observations>
```

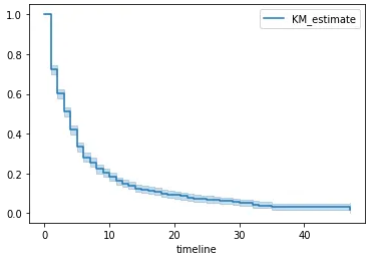
```
from matplotlib import pyplot as plt


kmf.survival_function_.plot()
plt.title('Survival function of political regimes');
```



Interpretation: The y-axis represents the probability a leader is still around after t years, where t years is on the x-axis.

We see that very few leaders make it past 20 years in office. Of course, we need to report how uncertain we are about these point estimates, i.e., we need confidence intervals. They are computed in the call to fit(), and located under the confidence_interval_ property. (The method uses exponential Greenwood confidence interval. The mathematics are found in these notes.) We can call plot() on the KaplanMeierFitter itself to plot both the KM estimate and its confidence intervals:

```
kmf.plot_survival_function()
<matplotlib.axes._subplots.AxesSubplot at 0x13a17cac0>
```



The median time in office, which defines the point in time where on average 50% of the population has expired, is a property:

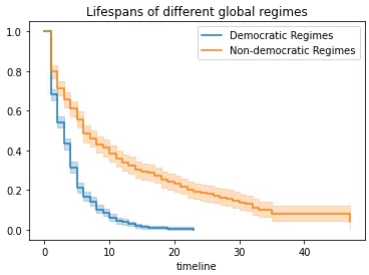
```
kmf.median_survival_time_
4.0
```

Interesting that it is only four years. That means, around the world, elected leaders have a 50% chance of cessation in four years or less! To get the confidence interval of the median, you can use:

```
In [8]: from lifelines.utils import median_survival_times
        median_ci = median_survival_times(kmf.confidence_interval_)
```

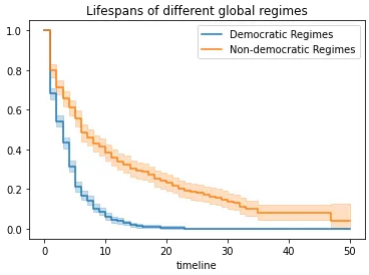
Let's segment on democratic regimes vs non-democratic regimes. Calling plot on either the estimate itself or the fitter object will return an axis object, that can be used for plotting further estimates:

```
In [9]: ax = plt.subplot(111)

        dem = (data["democracy"] == "Democracy")

        kmf.fit(T[dem], event_observed=E[dem], label="Democratic Regimes")
        kmf.plot_survival_function(ax=ax)

        kmf.fit(T[~dem], event_observed=E[~dem], label="Non-democratic Regimes")
        kmf.plot_survival_function(ax=ax)

        plt.title("Lifespans of different global regimes");
```



We might be interested in estimating the probabilities in between some points. We can do that with the timeline argument. We specify the times we are interested in and are returned a DataFrame with the probabilities of survival at those points:

In [10]:
```
import numpy as np

ax = plt.subplot(111)

t = np.linspace(0, 50, 51)
kmf.fit(T[dem], event_observed=E[dem], timeline=t, label="Democratic Regimes")
ax = kmf.plot_survival_function(ax=ax)

kmf.fit(T[~dem], event_observed=E[~dem], timeline=t, label="Non-democratic Regimes")
ax = kmf.plot_survival_function(ax=ax)

plt.title("Lifespans of different global regimes");
```



It is incredible how much longer these non-democratic regimes exist for. A democratic regime does have a natural bias towards death though: both via elections and natural limits (the US imposes a strict eight-year limit). The median of a non-democratic is only about twice as large as a democratic regime, but the difference is apparent in the tails: if you're a non-democratic leader, and you've made it past the 10 year mark, you probably have a long life ahead. Meanwhile, a democratic leader rarely makes it past ten years, and then have a very short lifetime past that.

Here the difference between survival functions is very obvious, and performing a statistical test seems pedantic. If the curves are more similar, or we possess less data, we may be interested in performing a statistical test. In this case, lifelines contains routines in lifelines.statistics to compare two survival functions. Below we demonstrate this routine. The function lifelines.statistics.logrank_test() is a common statistical test in survival analysis that compares two event series' generators. If the value returned

exceeds some pre-specified value, then we rule that the series have different generators.

In [11]:
```
from lifelines.statistics import logrank_test

results = logrank_test(T[dem], T[~dem], E[dem], E[~dem], alpha=.99)

results.print_summary()
```

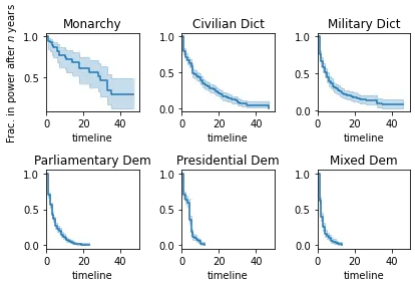
	t_0	-1
	null_distribution	chi squared
	degrees_of_freedom	1
	alpha	0.99
	test_name	logrank_test

	test_statistic	p	-log2(p)
0	260.47	<0.005	192.23

There are alternative (and sometimes better) tests of survival functions, and we explain more here: Statistically compare two populations

Lets compare the different types of regimes present in the dataset:

In [12]:
```
regime_types = data['regime'].unique()

for i, regime_type in enumerate(regime_types):
    ax = plt.subplot(2, 3, i + 1)

    ix = data['regime'] == regime_type
    kmf.fit(T[ix], E[ix], label=regime_type)
    kmf.plot_survival_function(ax=ax, legend=False)

    plt.title(regime_type)
    plt.xlim(0, 50)

    if i==0:
        plt.ylabel('Frac. in power after $n$ years')

plt.tight_layout()
```

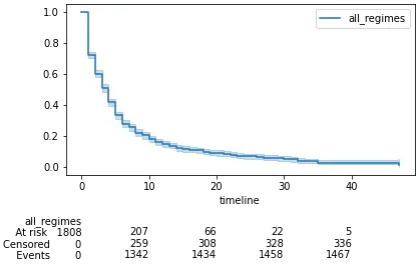


<> Best practices for presenting Kaplan Meier plots

A recent survey of statisticians, medical professionals, and other stakeholders suggested that the addition of two pieces of information, summary tables and confidence intervals, greatly increased the effectiveness of Kaplan Meier plots, see “Morris TP, Jarvis CI, Cragg W, et al. Proposals on Kaplan–Meier plots in medical research and a survey of stakeholder views: KMunicate. BMJ Open 2019;9:e030215. doi:10.1136/bmjopen-2019–030215”.

In lifelines, confidence intervals are automatically added, but there is the at_risk_counts kwarg to add summary tables as well:

```
In [13]: kmf = KaplanMeierFitter().fit(T, E, label="all_regimes")
         kmf.plot_survival_function(at_risk_counts=True)
         plt.tight_layout()
```



<> Getting data into the right format

lifelines data format is consistent across all estimator class and functions: an array of individual durations, and the individuals event observation (if any). These are often denoted T and E respectively. For example:

```
T = [0, 3, 3, 2, 1, 2]
E = [1, 1, 0, 0, 1, 1]
kmf.fit(T, event_observed=E)
```

```
<lifelines.KaplanMeierFitter:"all_regimes", fitted with 6 total
observations, 2 right-censored observations>
```

The raw data is not always available in this format — lifelines includes some helper functions to transform data formats to lifelines format. These are located in the lifelines.utils sub-library. For example, the function datetimes_to_durations() accepts an array or Pandas object of start times/dates, and an array or Pandas objects of end times/dates (or None if not observed):

```
In [15]: from lifelines.utils import datetimes_to_durations

         start_date = ['2013-10-10 0:00:00', '2013-10-09', '2013-10-10']
         end_date = ['2013-10-13', '2013-10-10', None]
         T, E = datetimes_to_durations(start_date, end_date, fill_date='2013-10-15')
         print('T (durations): ', T)
         print('E (event_observed): ', E)
```

```
T (durations):  [3. 1. 5.]
E (event_observed):  [ True  True False]
```

The function datetimes_to_durations() is very flexible, and has many keywords to tinker with.

1.1 Estimating hazard rates using Nelson-Aalen

The survival function (S) is a great way to summarize and visualize the survival dataset, however it is not the only way. If we are curious about the hazard function h(t) of a population, we unfortunately cannot transform the Kaplan Meier estimate — statistics doesn't work quite that well. Fortunately, there is a proper non-parametric estimator of the cumulative hazard function:

$$H(t) = \int_0^t \lambda(z)dz$$

The estimator for this quantity is called the Nelson Aalen estimator (Altschuler–Nelson estimator):

$$\hat{H}(t) = \sum_{i:t_i<t} \frac{d_i}{n_i}$$

where d_{i} is the number of deaths at time t_{i} and n_{i} is the number of susceptible individuals.

$$S(t) = \exp(-\hat{H}(t))$$

This estimator has advantages over Kaplan-Meier's.

- The estimator gives the correct expected number of events.
- There is a wealth of asymptotic theory based on the Altschuler–Nelson estimator.

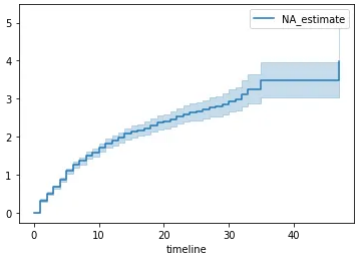
In [16]:
```
T = data["duration"]
E = data["observed"]

from lifelines import NelsonAalenFitter
naf = NelsonAalenFitter()

naf.fit(T,event_observed=E)
```

```
<lifelines.NelsonAalenFitter:"NA_estimate", fitted with 1808 total
observations, 340 right-censored observations>
```

```
print(naf.cumulative_hazard_.head())
naf.plot_cumulative_hazard()
```

```
NA_estimate
timeline
0.0          0.000000
1.0          0.325912
2.0          0.507356
3.0          0.671251
4.0          0.869867
```

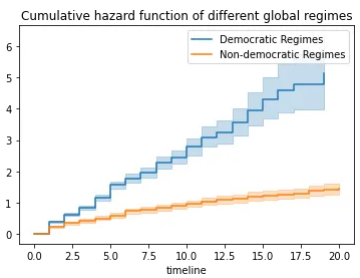
```
<matplotlib.axes._subplots.AxesSubplot at 0x139cd3a60>
```



The cumulative hazard has less obvious understanding than the survival functions, but the hazard functions is the basis of more advanced techniques in survival analysis. Recall that we are estimating cumulative hazard functions, H(t). (Why? The sum of estimates is much more stable than the point-wise estimates.) Thus we know the rate of change of this curve is an estimate of the hazard function.

Looking at figure above, it looks like the hazard starts of high and gets smaller (as seen by the decreasing rate of change). Let's break the regimes down between democratic and non-democratic, during the first 20 years:

In [18]:
```
naf.fit(T[dem], event_observed=E[dem], label="Democratic Regimes")
ax = naf.plot_cumulative_hazard(loc=slice(0, 20))

naf.fit(T[~dem], event_observed=E[~dem], label="Non-democratic Regimes")
naf.plot_cumulative_hazard(ax=ax, loc=slice(0, 20))

plt.title("Cumulative hazard function of different global regimes");
```



Looking at the rates of change, I would say that both political philosophies have a constant hazard, albeit democratic regimes have a much higher constant hazard.

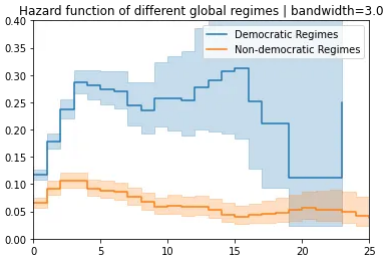
<> Smoothing the hazard function

Interpretation of the cumulative hazard function can be difficult — it is not how we usually interpret functions. On the other hand, most survival analysis is done using the cumulative hazard function, so understanding it is recommended.

Alternatively, we can derive the more interpretable hazard function, but there is a catch. The derivation involves a kernel smoother (to smooth out the differences of the cumulative hazard function) , and this requires us to specify a bandwidth parameter that controls the amount of smoothing. This functionality is in the smoothed_hazard_() and smoothed_hazard_confidence_intervals_() methods. Why methods? They require an argument representing the bandwidth.

There is also a plot_hazard() function (that also requires a bandwidth keyword) that will plot the estimate plus the confidence intervals, similar to the traditional plot() functionality.

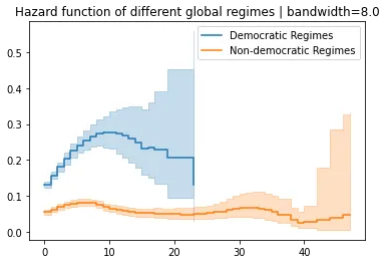
In [19]:
```
bandwidth = 3.

naf.fit(T[dem], event_observed=E[dem], label="Democratic Regimes")
ax = naf.plot_hazard(bandwidth=bandwidth)

naf.fit(T[~dem], event_observed=E[~dem], label="Non-democratic Regimes")
naf.plot_hazard(ax=ax, bandwidth=bandwidth)

plt.title("Hazard function of different global regimes | bandwidth=%.1f" % bandwidth);
plt.ylim(0, 0.4)
plt.xlim(0, 25);
```



It is more clear here which group has the higher hazard, and Non-democratic regimes appear to have a constant hazard.

There is no obvious way to choose a bandwidth, and different bandwidths produce different inferences, so it's best to be very careful here. My advice: stick with the cumulative hazard function.

```
In [20]: bandwidth = 8.0

         naf.fit(T[dem], event_observed=E[dem], label="Democratic Regimes")
         ax = naf.plot_hazard(bandwidth=bandwidth)

         naf.fit(T[~dem], event_observed=E[~dem], label="Non-democratic Regimes")
         naf.plot_hazard(ax=ax, bandwidth=bandwidth)

         plt.title("Hazard function of different global regimes | bandwidth=%.1f" % bandwidth);
```



1.2 Estimating cumulative hazards using parametric models

Parametric approaches are based on certain distributions such as exponential distribution, Weibull, Log-Normal, Log-Logistic, and more. Then we estimate the parameter, and then finally form the estimator of the survival function.

<> Fitting to a Weibull model

Another very popular model for survival data is the Weibull model. In contrast the the Nelson-Aalen estimator, this model is a parametric model, meaning it has a functional form with parameters that we are fitting the data to. (The Nelson-Aalen estimator has no parameters to fit to). The survival function looks like:

$$S(t) = \exp\left(-\left(\frac{t}{\lambda}\right)^{\rho}\right), \lambda > 0, \rho > 0$$

A priori, we do not know what λ and ρ are, but we use the data on hand to estimate these parameters. We model and estimate the cumulative hazard rate instead of the survival function (this is different than the Kaplan-Meier estimator):

$$H(t) = \left(\frac{t}{\lambda}\right)^{\rho}$$

- It is a generalization of exponential hazard function. The Weibull distribution with $\gamma = 1$ is an exponential distribution.
- When $\gamma > 1$, its hazard is increasing with t, and when $\gamma < 1$ its hazard is decreasing.
- $h(t|x) = \alpha\gamma t^{\gamma-1}$
- $H(t|X) = \alpha t^{\gamma}$
- $S(t|X) = \exp(-\alpha t^{\gamma})$
- $T_{0.5} = (\frac{ln(2)}{\alpha})^{1/\gamma}$

In [21]:

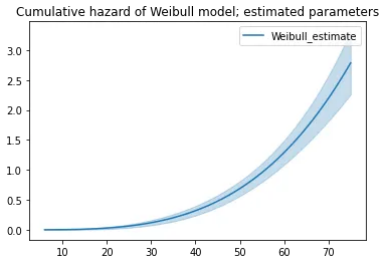
```
from lifelines import WeibullFitter
from lifelines.datasets import load_waltons

data = load_waltons()

T = data['T']
E = data['E']

wf = WeibullFitter().fit(T, E)

wf.print_summary()
ax = wf.plot_cumulative_hazard()
ax.set_title("Cumulative hazard of Weibull model; estimated parameters")
```

model	lifelines.WeibullFitter
number of observations	163
number of events observed	156
log-likelihood	-672.06
hypothesis	lambda_ != 1, rho_ != 1

	coef	se(coef)	coef lower 95%	coef upper 95%	z	p	-log2(p)
lambda_	55.73	1.33	53.13	58.33	41.26	<0.005	inf
rho_	3.45	0.24	2.97	3.93	10.07	<0.005	76.83

AIC	1348.12

```
Text(0.5, 1.0, 'Cumulative hazard of Weibull model; estimated
parameters')
```



<> Other parametric models: Exponential, Log-Logistic, Log-Normal and Splines

Similarly, there are other parametric models in lifelines. Generally, which parametric model to choose is determined by either knowledge of the distribution of durations, or some sort of model goodness-of-fit. Below are the built-in parametric models, and the Nelson-Aalen non-parametric model, of the same data.

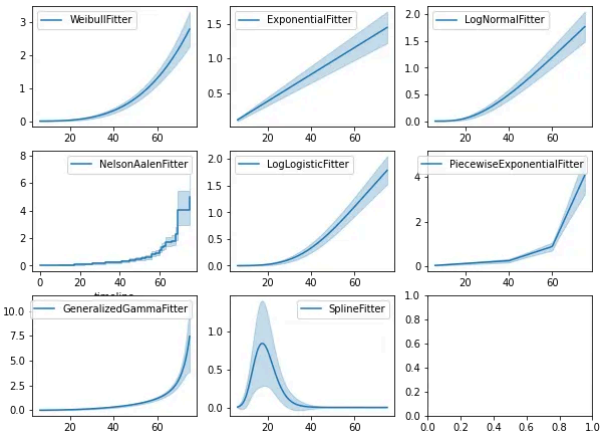
Exponential Model

- is the simple one where it assumes the hazard function is constant: $h(t) = h$
- $h(t|x)=h$
- $H(t|X)= h t$
- $S(t|X)=\exp(-h t)$
- $T_{0.5}=\ln(2)/\lambda$
- It has the property that the future lifetime of a subject is the same, no matter how “old” it is, or Prob{T>t0 + t|T >t0} = Prob{T>t}. This “ageless”

property also makes the exponential distribution a poor choice for modeling human survival except over short time periods.

In [22]:
```
from lifelines import (WeibullFitter, ExponentialFitter,
LogNormalFitter, LogLogisticFitter, NelsonAalenFitter,
PiecewiseExponentialFitter, GeneralizedGammaFitter, SplineFitter)

from lifelines.datasets import load_waltons
data = load_waltons()

fig, axes = plt.subplots(3, 3, figsize=(10, 7.5))

T = data['T']
E = data['E']

wbf = WeibullFitter().fit(T, E, label='WeibullFitter')
exf = ExponentialFitter().fit(T, E, label='ExponentialFitter')
lnf = LogNormalFitter().fit(T, E, label='LogNormalFitter')
naf = NelsonAalenFitter().fit(T, E, label='NelsonAalenFitter')
llf = LogLogisticFitter().fit(T, E, label='LogLogisticFitter')
pwf = PiecewiseExponentialFitter([40, 60]).fit(T, E, label='PiecewiseExponentialFitter')
gg = GeneralizedGammaFitter().fit(T, E, label='GeneralizedGammaFitter')
spf = SplineFitter([6, 20, 40, 75]).fit(T, E, label='SplineFitter')

wbf.plot_cumulative_hazard(ax=axes[0][0])
exf.plot_cumulative_hazard(ax=axes[0][1])
lnf.plot_cumulative_hazard(ax=axes[0][2])
naf.plot_cumulative_hazard(ax=axes[1][0])
llf.plot_cumulative_hazard(ax=axes[1][1])
pwf.plot_cumulative_hazard(ax=axes[1][2])
gg.plot_cumulative_hazard(ax=axes[2][0])
spf.plot_cumulative_hazard(ax=axes[2][1])
```

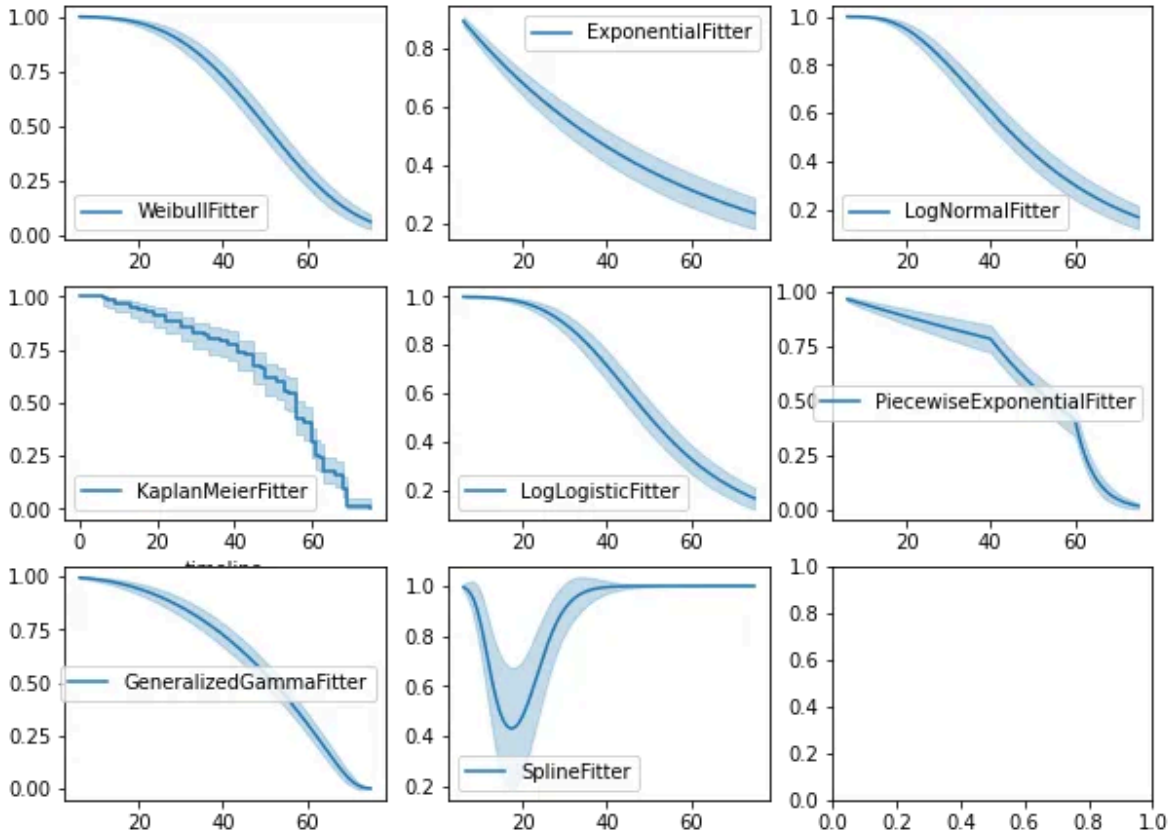
```
<matplotlib.axes._subplots.AxesSubplot at 0x13a494f70>
```



Parametric models can also be used to create and plot the survival function, too. Below we compare the parametric models versus the non-parametric Kaplan-Meier estimate:

In [23]:
```
from lifelines import KaplanMeierFitter

fig, axes = plt.subplots(3, 3, figsize=(10, 7.5))

T = data['T']
E = data['E']

kmf = KaplanMeierFitter().fit(T, E, label='KaplanMeierFitter')
wbf = WeibullFitter().fit(T, E, label='WeibullFitter')
exf = ExponentialFitter().fit(T, E, label='ExponentialFitter')
lnf = LogNormalFitter().fit(T, E, label='LogNormalFitter')
llf = LogLogisticFitter().fit(T, E, label='LogLogisticFitter')
pwf = PiecewiseExponentialFitter([40, 60]).fit(T, E, label='PiecewiseExponentialFitter')
gg = GeneralizedGammaFitter().fit(T, E, label='GeneralizedGammaFitter')
spf = SplineFitter([6, 20, 40, 75]).fit(T, E, label='SplineFitter')

wbf.plot_survival_function(ax=axes[0][0])
exf.plot_survival_function(ax=axes[0][1])
lnf.plot_survival_function(ax=axes[0][2])
kmf.plot_survival_function(ax=axes[1][0])
llf.plot_survival_function(ax=axes[1][1])
pwf.plot_survival_function(ax=axes[1][2])
gg.plot_survival_function(ax=axes[2][0])
spf.plot_survival_function(ax=axes[2][1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x139aad460>
```

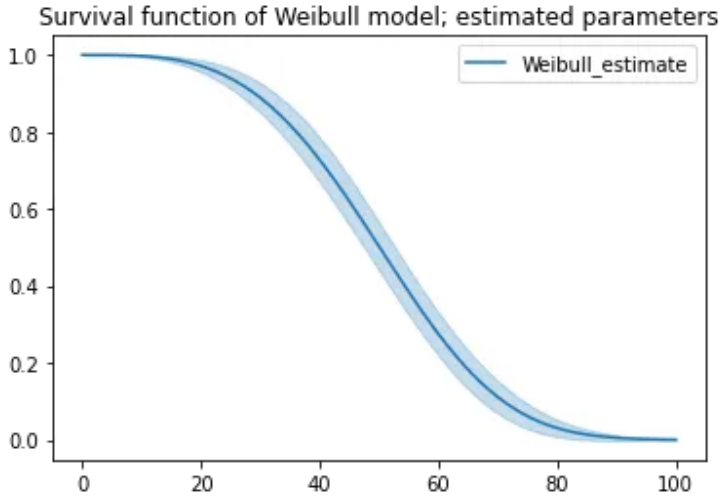


With parametric models, we have a functional form that allows us to extend the survival function (or hazard or cumulative hazard) past our maximum observed duration. This is called extrapolation. We can do this in a few ways.

In [24]:
```
timeline = np.linspace(0, 100, 200)

# directly compute the survival function, these return a pandas Series
wbf = WeibullFitter().fit(T, E)
wbf.survival_function_at_times(timeline)
wbf.hazard_at_times(timeline)
wbf.cumulative_hazard_at_times(timeline)

# use the `timeline` kwarg in `fit`
# by default, all functions and properties will use
# these values provided
wbf = WeibullFitter().fit(T, E, timeline=timeline)

ax = wbf.plot_survival_function()
ax.set_title("Survival function of Weibull model; estimated parameters")
```

```
Text(0.5, 1.0, 'Survival function of Weibull model; estimated
```

```
parameters')
```



Ch 2. Survival Regression

Often we have additional data aside from the duration that we want to use. The technique is called survival regression — the name implies we regress covariates (e.g., age, country, etc.) against another variable — in this case **durations**. There are a few popular models in survival regression such as **Cox Proportional Hazard Model**, **accelerated failure models** and **Aalen's additive model.** All models attempt to represent the hazard rate $h(t|x)$ as a function of t and some covariates x. Here, We explore Cox Proportional Hazards Regression Model models and Survival Regression is short for **Cox Proportional Hazards Regression Model.**

Survival analysis consitently is used in various industries such as:

- Insurance — time to lapsing on policy.
- Mortgages — time to mortgage redemption

- Mail Order Catalogue — time to next purchase
- Retail — time till food customer starts purchasing non-food
- Manufacturing — lifetime of a machine component
- Public Sector — time intervals to critical events

Notes:

1. We cannot use traditional methods like linear regression because of **censoring.**
2. Why do we say "Proportional Hazards" here? Because it is the most important assumption in this model which will be considered later.

2.0 Dataset to use

To review Cox Proportional Hazards Regression Model, the Rossi recidivism dataset (available in lifelines as load_rossi()) is used. This data set is originally from Rossi et al. (1980), and is used as an example in Allison (1995). The data pertain to 432 convicts who were released from Maryland state prisons in the 1970s and who were followed up for one year after release. Half the released convicts were assigned at random to an experimental treatment in which they were given financial aid; half did not receive aid. The week column is the duration, the arrest column denotes if the event (a re-arrest) occurred, and the other columns represent variables we wish to regress against.

Following is descriptions of Columns:

1. Week: Duration in weeks.
2. arrest: The event; 1 if arrested, 0 if not arrested.

3. fin: Financial aid: no yes.
4. Age: Age in years at time of release.
5. Race: Race; black or other.
6. wexp: full-time work experience before incarceration: no or yes.
7. mar: marital status at time of release; married or not married.
8. paro: released on parole? no or yes.
9. prio: number of convictions prior to current incarceration.

```
In [25]: from lifelines.datasets import load_rossi
         rossi = load_rossi()
         rossi.head()
```

	week	arrest	fin	age	race	wexp	mar	paro	prio
0	20	1	0	27	1	0	0	1	3
1	17	1	0	18	1	0	0	1	8
2	25	1	0	19	0	1	0	1	13
3	52	0	1	23	1	1	1	1	1
4	52	0	0	19	0	1	0	1	3

2.1 Cox Proportional Hazard Model

The idea behind Cox's proportional hazard model is that the log-hazard of an individual is a linear function of their covariates and a population-level baseline hazard that changes over time. Mathematically:

$$h(t|\mathbf{x}) = b_0(t)\exp\left(\sum_{i=1}^{n} b_i(\mathbf{x}_i - \bar{\mathbf{x}}_i)\right)$$

where,

- $h(t|\mathbf{x})$: hazard,
- $b_0(t)$: **baseline hazard**, no assumption about the shape of it.
- b_i: These are the regression parameters.
- $\sum_{i=1}^{n} b_i(\mathbf{x}_i - \bar{\mathbf{x}}_i)$ is log-partial hazard and
- $\exp\left(\sum_{i=1}^{n} b_i(\mathbf{x}_i - \bar{\mathbf{x}}_i)\right)$ is partial hazard.

Notes:

1. b_{0}(t) is the only time component. In the above equation, the partial hazard is a time-invariant scalar factor that only increases or decreases the baseline hazard. Thus changes in covariates will only inflate or deflate the baseline hazard.
2. A time-independent variable is defined to be any variable whose value for a given individual does not change over time. Examples are SEX and smoking status (SMK). Note, however, that a person's smoking status may actually change over time, but for purposes of the analysis, the SMK variable is assumed not to change once it is measured, so that only one value per individual is used.
3. Also note that although variables like AGE and weight change over time, it may be appropriate to treat such variables as time- independent in the analysis if their values do not change much over time or if the effect of such variables on survival risk depends essentially on the value at only one measurement.
4. The effects of covariates are additive and linear on the log-rate scale [If you take a log from both side you will see].

<> Hazard ratio

- In general, a hazard ratio (HR) is defined as the hazard for one individual divided by the hazard for a different individual. The two individuals being compared can be distinguished by their values for the set of covariates, that is, the x's

$$\mathrm{HR} = \frac{h(t|\mathbf{x}^*)}{h(t|\mathbf{x})} = \exp\left[\sum_{i=1}^{n} b_i(x_i^* - x_i)\right]$$

- a hazard ratio greater than 1 means the event is more likely to occur, and a ratio less than one means an event is less likely to occur. A hazard ratio of 1 means the predictor has no effect on the hazard of the event.
- Baseline hazard: In estimating the baseline hazard function, a Cox model uses the so-called Aalen-Breslow estimator, which is a generalization of the non-parametric Nelson-Aalen estimator of the cumulative hazard function.

<> Cox's partial likelihood function

Cox's partial likelihood function for i = 1, ..., n is:

$$L(\beta) = \prod_{i=1}^{n} \left(\frac{\exp(\beta x_i)}{\sum_{j \in R(t_i)} \exp(\beta x_j)} \right)^{D_i}$$

R(t_{i}) is the risk set at time t_{i}, i.e. the set of individuals at risk of just before time t_{i}.

- In presence of ties for failure times, one should use Breslow or Efron formula for approximating log-likelihood.
- Once a partial log likelihood function is derived, it is used as if it were an ordinary log likelihood function to estimate β, estimate standard errors of β, obtain confidence limits, and make statistical tests.

<> Fitting the regression

The implementation of the Cox model in lifelines is under **CoxPHFitter**. We fit the model to the dataset using fit(). It has a print_summary() function that prints a tabular view of coefficients and related stats.

In [26]:
```
from lifelines import CoxPHFitter

cph = CoxPHFitter()
cph.fit(rossi, duration_col='week', event_col='arrest')
```

```
<lifelines.CoxPHFitter: fitted with 432 total observations, 318
right-censored observations>
```

We can also use formulas to handle the right-hand-side of the linear model. For example the linear model with interaction term:

$\beta 1\text{fin}+\beta 2\text{wexp}+\beta 3\text{age}+\beta 4\text{prio}+\beta 5\text{age}\cdot\text{prio}$

cph.fit(rossi, duration_col='week', event_col='arrest', formula="fin + wexp + age * prio")

In [27]:
```
cph.fit(rossi, duration_col='week', event_col='arrest', formula="fin + wexp + age * prio")
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-659.39
time fit was run	2021-01-19 22:43:51 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.33	0.72	0.19	-0.70	0.04	0.49	1.05	-1.73	0.08	3.57
wexp	-0.24	0.79	0.21	-0.65	0.17	0.52	1.19	-1.14	0.26	1.97
age	-0.03	0.97	0.03	-0.09	0.03	0.92	1.03	-0.93	0.35	1.51
prio	0.31	1.36	0.17	-0.03	0.64	0.97	1.90	1.80	0.07	3.80
age:prio	-0.01	0.99	0.01	-0.02	0.01	0.98	1.01	-1.28	0.20	2.32

Concordance	0.64
Partial AIC	1328.77
log-likelihood ratio test	31.99 on 5 df
-log2(p) of ll-ratio test	17.35

<> How to interpret the resutls

```
cph.print_summary()  # access the individual results using
cph.summary
```


model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-659.39
time fit was run	2021-01-19 22:43:51 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.33	0.72	0.19	-0.70	0.04	0.49	1.05	-1.73	0.08	3.57
wexp	-0.24	0.79	0.21	-0.65	0.17	0.52	1.19	-1.14	0.26	1.97
age	-0.03	0.97	0.03	-0.09	0.03	0.92	1.03	-0.93	0.35	1.51
prio	0.31	1.36	0.17	-0.03	0.64	0.97	1.90	1.80	0.07	3.80
age:prio	-0.01	0.99	0.01	-0.02	0.01	0.98	1.01	-1.28	0.20	2.32

Concordance	0.64
Partial AIC	1328.77
log-likelihood ratio test	31.99 on 5 df
-log2(p) of ll-ratio test	17.35

To access the coefficients and the baseline hazard directly, you can use *params_* and *baseline_hazard_* respectively. Taking a look at these coefficients for a moment, prio (the number of prior arrests) has a coefficient of about

0.09. Thus, a one unit increase in prio means the the baseline hazard will increase by a factor of exp(0.09)=1.10 — about a 10% increase. Recall, in the Cox proportional hazard model, a higher hazard means more at risk of the event occurring. The value exp(0.09) is the **hazard ratio**, a name that will be clear with another example.

- Example:

Consider the coefficient of mar (whether the subject is married or not). The values in the column are binary: 0 or 1, representing either unmarried or married. The value of the coefficient associated with mar, exp(−.43), is the value of ratio of hazards associated with being married, that is:

$$\exp(-0.43) = \frac{\text{hazard of married subjects at time t}}{\text{hazard of unmarried subjects at time t}}$$

Note that left-hand side is a constant (specifically, it's independent of time, *t*), but the right-hand side has two factors that may vary with time. The proportional hazard assumption is that relationship is true. That is, hazards can change over time, but their ratio between levels remains a constant. Later we will deal with checking this assumption. However, in reality, it's very common for the hazard ratio to change over the study duration. The hazard ratio then has the interpretation of some sort of weighted average of period-specific hazard ratios. As a result, the hazard ratio may critically depend on the duration of the follow-up.

```
cph.params_
```

```
covariate
fin         -0.330167
wexp        -0.238619
age         -0.028550
prio         0.307433
age:prio    -0.009743
Name: coef, dtype: float64
```

```
#cph.baseline_hazard_
```

<> Convergence

Fitting the Cox model to the data involves using iterative methods. lifelines is well developed in term of convergence by providing detailed warnings. Fixing any warnings will generally help convergence and decrease the number of iterative steps required. If you wish to see more information during fitting, there is a *show_progress* parameter in fit() function.

After fitting, the value of the **maximum log-likelihood** this available using log_likelihood_. The variance matrix of the coefficients is available under variance_matrix_.

Log-likelihood values cannot be used alone as an index of fit because they are a function of sample size but can be used to compare the fit of different coefficients. Because you want to maximize the log-likelihood, the higher value is better. The natural logarithm function is negative for values less than one and positive for values greater than one. So it is possible that you end up with a negative value for log-likelihood (for discrete variables it will always be so).

```
cph.log_likelihood_
```

```
-659.3864712540503
```

```
cph.variance_matrix_
```

covariate	fin	wexp	age	prio	age:prio
covariate					
fin	0.036502	0.000539	0.000084	0.002711	-0.000131
wexp	0.000539	0.044050	-0.001806	-0.000028	0.000067
age	0.000084	-0.001806	0.000934	0.003642	-0.000163
prio	0.002711	-0.000028	0.003642	0.029134	-0.001282
age:prio	-0.000131	0.000067	-0.000163	-0.001282	0.000058

<> Goodness of fit

After fitting, you may want to know how "good" of a fit your model was to the data. Following, 4 main approaches to evaluate the model fit are addressed:

1. inspect the survival probability calibration plot

- function to measure your fitted survival model against observed frequencies of events. We follow the advice in "Graphical calibration curves and the integrated calibration index (ICI) for survival models" by P. Austin and co., and use create a smoothed calibration curve using a flexible spline regression model (this avoids the traditional problem of binning the continuous-valued probability, and handles censored data).
- ICI: The ICI is a weighted average of the absolute difference between the calibration curve and the diagonal line of perfect calibration, where the absolute differences are weighted by the density function of the weights. This is equivalent to integrating f(x) over the distribution of the predicted probabilities.

1. look at the concordance-index, available as concordance_index_ or in the print_summary() as a measure of predictive accuracy.

- Concordance Index (CI) is censoring-sensitive measure, also known as the c-index. This measure evaluates the accuracy of the ranking of

predicted time. It is in fact a generalization of AUC, another common loss function, and is interpreted similarly:

1. 0.5 is the expected result from random predictions,
2. 1.0 is perfect concordance and,
3. 0.0 is perfect anti-concordance (multiply predictions with -1 to get 1.0)

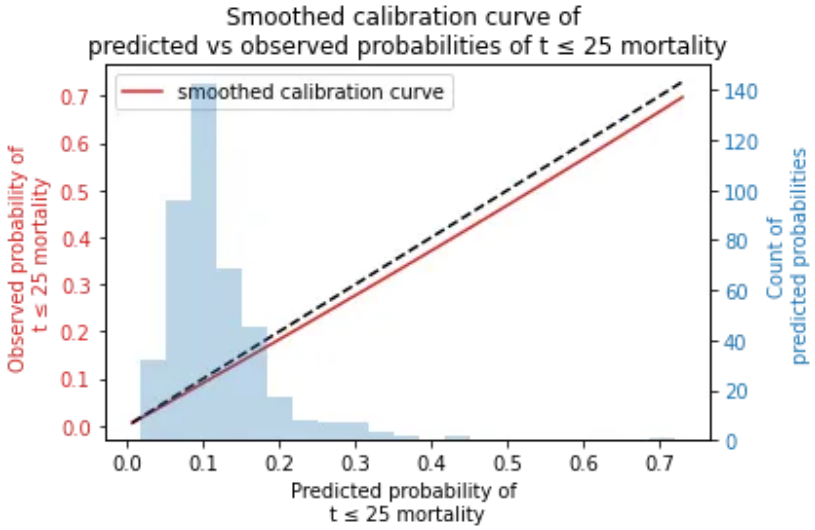
- Fitted survival models typically have a concordance index between 0.55 and 0.75 (this may seem bad, but even a perfect model has a lot of noise than can make a high score impossible). In lifelines, a fitted model's concordance-index is present in the output of score(), but also available under the concordance_index_ property. Generally, the measure is implemented in lifelines under lifelines.utils.concordance_index() and accepts the actual times (along with any censored subjects) and the predicted times.

1. look at the log-likelihood test result in the print_summary() or log_likelihood_ratio_test()
2. check the proportional hazards assumption with the check_assumptions() method. See section later on this page for more details.

```
from lifelines.calibration import survival_probability_calibration

survival_probability_calibration(cph, rossi, t0=25)
```

```
ICI =  0.010308090904927618
E50 =  0.00935319647035604
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x13a2974c0>,
 0.010308090904927618,
 0.009353196470356O4)
```



<> Prediction

After fitting, you can use the suite of prediction methods: predict_partial_hazard(), predict_survival_function(), and others.

For prediction on censored subjects please visit: Prediction on censored subjects

```
X = rossi

#cph.predict_survival_function(X)
#cph.predict_median(X)
cph.predict_partial_hazard(X)
```

```
0      1.157686
1      3.768718
2      4.919516
3      0.699116
4      1.447786
         ...
427    0.520233
428    1.385036
```

```
429    0.784224
430    0.812407
431    0.672850
Length: 432, dtype: float64
```

<> Penalties and sparse regression

It's possible to add a penalizer term to the Cox regression as well. One can use these to:

1. stabilize the coefficients
2. shrink the estimates to 0
3. encourages a Bayesian viewpoint
4. create sparse coefficients.

All regression models, including the Cox model, include both an L1 and L2 penalty:

$$ \frac{1}{2}\textrm{penalizer} ((1−l_{1_{ratio}})·||*β*||*{2}^{2}+*l*{1_{ratio}}·||*β*||_{1}) $$

To use this in lifelines, both the penalizer and l1_ratio can be specified in the class creation:

```
from lifelines import CoxPHFitter

cph = CoxPHFitter(penalizer=0.1, l1_ratio=0.001) # sparse solutions,
cph.fit(rossi, 'week', 'arrest')
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
penalizer	0.1
l1 ratio	0.001
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-662.86
time fit was run	2021-01-19 22:43:52 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.27	0.76	0.16	-0.59	0.04	0.56	1.04	-1.70	0.09	3.49
age	-0.04	0.96	0.02	-0.07	-0.01	0.94	0.99	-2.34	0.02	5.69
race	0.20	1.23	0.25	-0.29	0.69	0.75	2.00	0.81	0.42	1.26
wexp	-0.20	0.82	0.17	-0.54	0.13	0.59	1.14	-1.18	0.24	2.08
mar	-0.31	0.73	0.28	-0.86	0.23	0.42	1.26	-1.12	0.26	1.93
paro	-0.06	0.94	0.17	-0.38	0.27	0.68	1.30	-0.35	0.73	0.46
prio	0.07	1.07	0.03	0.02	0.12	1.02	1.13	2.76	0.01	7.42

Concordance	0.65
Partial AIC	1339.71
log-likelihood ratio test	25.05 on 7 df
-log2(p) of ll-ratio test	10.39

Instead of a float, an array can be provided that is the same size as the number of penalized parameters. The values in the array are specific penalty coefficients for each covariate. This is useful for more complicated covariate structure. Some examples:

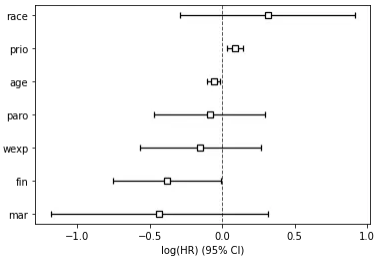
you have lots of confounders you wish to penalizer, but not the main treatment(s).

<> Plotting the coefficients

With a fitted model, an alternative way to view the coefficients and their ranges is to use the plot method.

```
from lifelines.datasets import load_rossi
from lifelines import CoxPHFitter

cph = CoxPHFitter()
cph.fit(rossi, duration_col='week', event_col='arrest')

cph.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13c75a9d0>
```

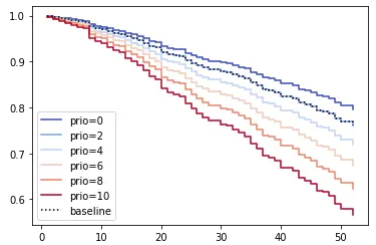


<> Plotting the effect of varying a covariate

After fitting, we can plot what the survival curves look like as we vary a single covariate while holding everything else equal. This is useful to understand the impact of a covariate, given the model. To do this, we use the plot_partial_effects_on_outcome() method and give it the covariate of interest, and the values to display.

```
from lifelines.datasets import load_rossi
from lifelines import CoxPHFitter

cph = CoxPHFitter()
cph.fit(rossi, duration_col='week', event_col='arrest')

cph.plot_partial_effects_on_outcome(covariates='prio', values=[0, 2,
4, 6, 8, 10], cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13c836070>
```



<> Checking the proportional hazards assumption

To make proper inferences, we should ask if our Cox model is appropriate for our dataset. Recall from above that when using the Cox model, we are implicitly applying the proportional hazard assumption. We should ask, does our dataset obey this assumption?

1. Non-proportional hazards: Survival curves for different strata must have hazard functions that are proportional over time. This is the proportional hazards assumption discussed above for a single explanatory variable and for the multivariate model.
2. There should be a linear relationship between the log hazard and each covariate. This is best checked using residual plots.
3. The mechanisms giving rise to censoring of individuals must not be related to the probability of an event occurring. This assumption is known as non-informative censoring and applies for nearly all forms of survival analysis.

CoxPHFitter has a check_assumptions() method that will output violations of the proportional hazard assumption. For a tutorial on how to fix violations, see Testing the Proportional Hazard Assumptions. Suggestions are to look for ways to stratify a column (see docs below), or use a time varying model.

Notes:

Checking assumptions like this is only necessary if your goal is inference or correlation. That is, you wish to understand the influence of a covariate on the survival duration & outcome. If your goal is prediction, checking model assumptions is less important since your goal is to maximize an accuracy metric, and not learn about how the model is making that prediction.

The proportional hazard assumption is that all individuals have the same hazard function, but a unique scaling factor infront. So the shape of the hazard function is the same for all individuals, and only a scalar multiple changes per individual.

$$h_i(t) = a_i h(t)$$

At the core of the assumption is that a_i is not time varying, that is, $a_i(t) = a_i$. Further more, if we take the ratio of this with another subject (called the hazard ratio):

$$\frac{h_i(t)}{h_j(t)} = \frac{a_i h(t)}{a_j h(t)} = \frac{a_i}{a_j}$$

is constant for all t. In this tutorial we will test this non-time varying assumption, and look at ways to handle violations.

In [38]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

from matplotlib import pyplot as plt
from lifelines import CoxPHFitter
import numpy as np
import pandas as pd

from lifelines.datasets import load_rossi
rossi = load_rossi()
cph = CoxPHFitter()

cph.fit(rossi, 'week', 'arrest')


cph.print_summary(model="untransformed variables", decimals=3)
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-658.748
time fit was run	2021-01-19 22:43:53 UTC
model	untransformed variables

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.379	0.684	0.191	-0.755	-0.004	0.470	0.996	-1.983	0.047	4.398
age	-0.057	0.944	0.022	-0.101	-0.014	0.904	0.986	-2.611	0.009	6.791
race	0.314	1.369	0.308	-0.290	0.918	0.748	2.503	1.019	0.308	1.698
wexp	-0.150	0.861	0.212	-0.566	0.266	0.568	1.305	-0.706	0.480	1.058
mar	-0.434	0.648	0.382	-1.182	0.315	0.307	1.370	-1.136	0.256	1.965
paro	-0.085	0.919	0.196	-0.469	0.299	0.626	1.348	-0.434	0.665	0.589
prio	0.091	1.096	0.029	0.035	0.148	1.036	1.159	3.194	0.001	9.476

Concordance	0.640
Partial AIC	1331.495
log-likelihood ratio test	33.266 on 7 df
-log2(p) of ll-ratio test	15.370

<> Checking assumptions with check_assumptions

New to lifelines 0.16.0 is the **CoxPHFitter.check_assumptions** method. This method will compute statistics that check the proportional hazard assumption, produce plots to check assumptions, and more. Also included is an option to display advice to the console.

- Presented first are the results of a statistical test to test for any time-varying coefficients. A time-varying coefficient imply a covariate's influence relative to the baseline changes over time. This implies a violation of the proportional hazard assumption. For each variable, we transform time four times (these are common transformations of time to perform). If lifelines rejects the null (that is, lifelines rejects that the coefficient is not time-varying), we report this to the user.
- Some advice is presented on how to correct the proportional hazard violation based on some summary statistics of the variable.
- As a compliment to the above statistical test, for each variable that violates the PH assumption, visual plots of the the scaled Schoenfeld residuals is presented against the four time transformations. A fitted

lowess is also presented, along with 10 bootstrapped lowess lines (as an approximation to the confidence interval of the original lowess line). Ideally, this lowess line is constant (flat). Deviations away from the constant line are violations of the PH assumption.

```
cph.check_assumptions(rossi, p_value_threshold=0.05, show_plots=True)

The ``p_value_threshold`` is set at 0.05. Even under the null hypothesis of no violations, some
covariates will be below the threshold by chance. This is compounded when there are many covariates.
Similarly, when there are lots of observations, even minor deviances from the proportional hazard
assumption will be flagged.

With that in mind, it's best to use a combination of statistical tests and visual tests to determine
the most serious violations. Produce visual plots using ``check_assumptions(..., show_plots=True)``
and looking for non-constant lines. See link [A] below for a full example.
```

		test_statistic	p	-log2(p)
age	km	11.03	<0.005	10.12
	rank	11.45	<0.005	10.45
fin	km	0.02	0.89	0.17
	rank	0.02	0.90	0.15
mar	km	0.60	0.44	1.19
	rank	0.71	0.40	1.32
paro	km	0.12	0.73	0.45
	rank	0.13	0.71	0.49
prio	km	0.02	0.88	0.18
	rank	0.02	0.89	0.17
race	km	1.44	0.23	2.12
	rank	1.43	0.23	2.11
wexp	km	7.48	0.01	7.32
	rank	7.31	0.01	7.19

null_distribution	chi squared
degrees_of_freedom	1
model	<lifelines.CoxPHFitter: fitted with 432 total ...
test_name	proportional_hazard_test

0.0007.

   Advice 1: the functional form of the variable 'age' might be incorrect. That is, there may be
non-linear terms missing. The proportional hazard test used is very sensitive to incorrect
functional forms. See documentation in link [D] below on how to specify a functional form.

   Advice 2: try binning the variable 'age' using pd.cut, and then specify it in `strata=['age',
...]` in the call in `.fit`. See documentation in link [B] below.

   Advice 3: try adding an interaction term with your time variable. See documentation in link [C]
below.

   Bootstrapping lowess lines. May take a moment...

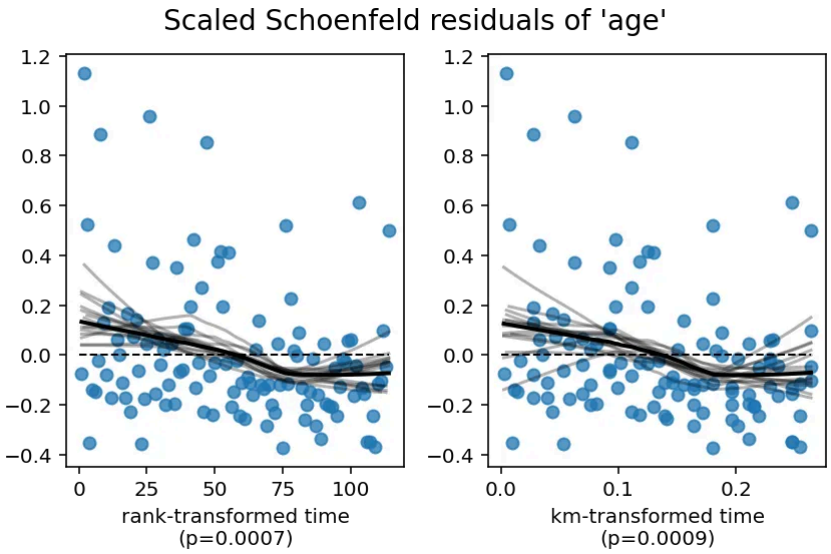
2. Variable 'wexp' failed the non-proportional test: p-value is 0.0063.

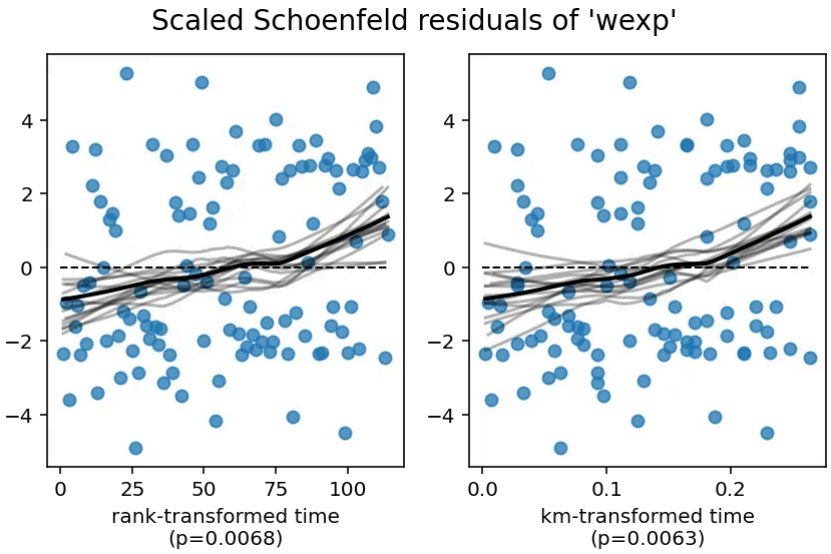
   Advice: with so few unique values (only 2), you can include `strata=['wexp', ...]` in the call in
`.fit`. See documentation in link [E] below.

   Bootstrapping lowess lines. May take a moment...

[A] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.html
[B] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.html#Bin-variable-and-stratify-on-it
[C] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.html#Introduce-time-varying-covariates
[D] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.html#Modify-the-functional-form
[E] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.html#Stratification

```
[[<matplotlib.axes._subplots.AxesSubplot at 0x13c869ee0>,
  <matplotlib.axes._subplots.AxesSubplot at 0x13c8eefa0>],
 [<matplotlib.axes._subplots.AxesSubplot at 0x13c991ca0>,
  <matplotlib.axes._subplots.AxesSubplot at 0x13c9d9dc0>]]
```






```
from lifelines.statistics import proportional_hazard_test
```

```
results = proportional_hazard_test(cph, rossi, time_transform='rank')
results.print_summary(decimals=3, model="untransformed variables")
```

time_transform	rank
null_distribution	chi squared
degrees_of_freedom	1
model	<lifelines.CoxPHFitter: fitted with 432 total ...
test_name	proportional_hazard_test

	test_statistic	p	-log2(p)
age	11.45	<0.005	10.45
fin	0.02	0.90	0.15
mar	0.71	0.40	1.32
paro	0.13	0.71	0.49
prio	0.02	0.89	0.17
race	1.43	0.23	2.11
wexp	7.31	0.01	7.19

<> Stratification

In the advice above, we can see that **wexp** has small cardinality, so we can easily fix that by specifying it in the strata. What does the strata do? Let's go back to the proportional hazard assumption.

In the introduction, we said that the proportional hazard assumption was that

$$h_{i}(t) = a_{i}h(t)$$

In a simple case, it may be that there are two subgroups that have very different baseline hazards. That is, we can split the dataset into subsamples based on some variable (we call this the stratifying variable), run the Cox model on all subsamples, and compare their baseline hazards. If these baseline hazards are very different, then clearly the formula above is wrong — the $h(t)$ is some weighted average of the subgroups' baseline hazards. This ill fitting average baseline can cause a_{i} to have time-dependent influence. A better model might be:

$$h_{i|i \in G(t)} = a_{i} h_{G}(t)$$

where now we have a unique baseline hazard per subgroup G. Because of the way the Cox model is designed, inference of the coefficients is identical (expect now there are more baseline hazards, and no variation of the stratifying variable within a subgroup G).

```
cph.fit(rossi, 'week', 'arrest', strata=['wexp'])
cph.print_summary(model="wexp in strata")
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
strata	[wexp]
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-580.89
time fit was run	2021-01-19 22:43:55 UTC
model	wexp in strata

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.38	0.68	0.19	-0.76	-0.01	0.47	0.99	-1.99	0.05	4.42
age	-0.06	0.94	0.02	-0.10	-0.01	0.90	0.99	-2.64	0.01	6.91
race	0.31	1.36	0.31	-0.30	0.91	0.74	2.49	1.00	0.32	1.65
mar	-0.45	0.64	0.38	-1.20	0.29	0.30	1.34	-1.19	0.23	2.09
paro	-0.08	0.92	0.20	-0.47	0.30	0.63	1.35	-0.42	0.67	0.57
prio	0.09	1.09	0.03	0.03	0.15	1.04	1.16	3.16	<0.005	9.33

Concordance	0.61
Partial AIC	1173.77
log-likelihood ratio test	23.77 on 6 df
-log2(p) of ll-ratio test	10.77

```
cph.check_assumptions(rossi, show_plots=True)
```

```
The ``p_value_threshold`` is set at 0.01. Even under the null
hypothesis of no violations, some
covariates will be below the threshold by chance. This is compounded
when there are many covariates.
Similarly, when there are lots of observations, even minor deviances
from the proportional hazard
assumption will be flagged.
```

With that in mind, it's best to use a combination of statistical tests and visual tests to determine
the most serious violations. Produce visual plots using ``check_assumptions(..., show_plots=True)``
and looking for non-constant lines. See link [A] below for a full example.

		test_statistic	p	-log2(p)
age	km	11.29	<0.005	10.32
	rank	4.62	0.03	4.99
fin	km	0.02	0.90	0.16
	rank	0.05	0.83	0.28
mar	km	0.53	0.47	1.10
	rank	1.31	0.25	1.99
paro	km	0.09	0.76	0.40
	rank	0.00	0.97	0.05
prio	km	0.02	0.89	0.16
	rank	0.02	0.90	0.16
race	km	1.47	0.23	2.15
	rank	0.64	0.42	1.23

null_distribution	chi squared
degrees_of_freedom	1
model	<lifelines.CoxPHFitter: fitted with 432 total ...
test_name	proportional_hazard_test

1. Variable 'age' failed the non-proportional test: p-value is 0.0008.

   Advice 1: the functional form of the variable 'age' might be incorrect. That is, there may be
non-linear terms missing. The proportional hazard test used is very sensitive to incorrect
functional forms. See documentation in link [D] below on how to specify a functional form.

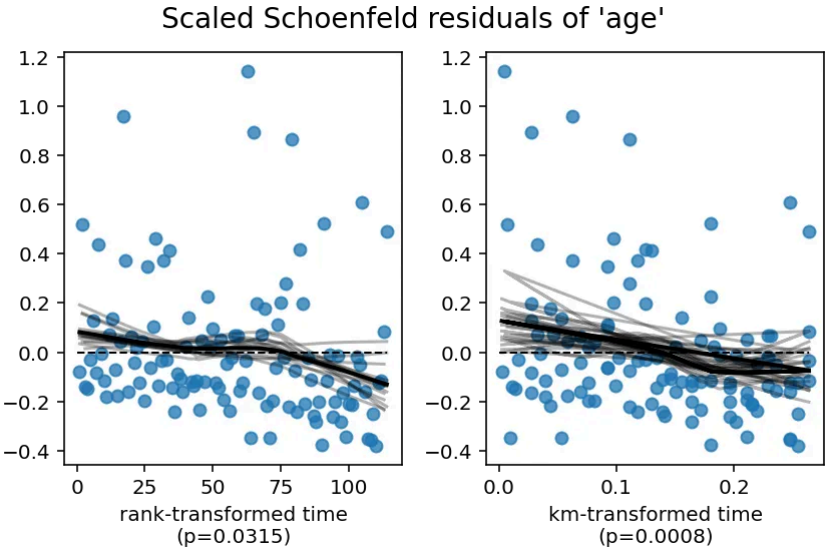
   Advice 2: try binning the variable 'age' using pd.cut, and then specify it in `strata=['age',
...]` in the call in `.fit`. See documentation in link [B] below.

   Advice 3: try adding an interaction term with your time variable. See documentation in link [C]
below.

   Bootstrapping lowess lines. May take a moment...

```
[A]
https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proporti
onal%20hazard%20assumption.html
[B]
https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proporti
onal%20hazard%20assumption.html#Bin-variable-and-stratify-on-it
[C]
https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proporti
onal%20hazard%20assumption.html#Introduce-time-varying-covariates
[D]
https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proporti
onal%20hazard%20assumption.html#Modify-the-functional-form
[E]
https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proporti
onal%20hazard%20assumption.html#Stratification
```

```
[[<matplotlib.axes._subplots.AxesSubplot at 0x13c71aac0>,
  <matplotlib.axes._subplots.AxesSubplot at 0x13caef580>]]
```



Since age is still violating the proportional hazard assumption, we need to model it better. From the residual plots above, we can see a the effect of age start to become negative over time. This will be relevant later. Below, we present three options to handle age.

<> Modify the functional form

The proportional hazard test is very sensitive (i.e. lots of false positives) when the functional form of a variable is incorrect. For example, if the association between a covariate and the log-hazard is non-linear, but the model has only a linear term included, then the proportional hazard test can raise a false positive.

The modeller can choose to add quadratic or cubic terms, i.e:

```
rossi['age**2'] = (rossi['age'] - rossi['age'].mean())**2
rossi['age**3'] = (rossi['age'] - rossi['age'].mean())**3
```

but I think a more correct way to include non-linear terms is to use basis splines:

```
cph.fit(rossi, 'week', 'arrest', strata=['wexp'], formula="bs(age,
df=4, lower_bound=10, upper_bound=50) + fin +race + mar + paro +
prio")
cph.print_summary(model="spline_model"); print()
cph.check_assumptions(rossi, show_plots=True, p_value_threshold=0.05)
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
strata	[wexp]
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-579.36
time fit was run	2021-01-19 22:43:56 UTC
model	spline_model

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
bs(age, df=4, lower_bound=10, upper_bound=50)[0]	-2.95	0.05	8.32	-19.26	13.37	0.00	6.39e+05	-0.35	0.72	0.47
bs(age, df=4, lower_bound=10, upper_bound=50)[1]	-5.48	0.00	6.23	-17.69	6.73	0.00	839.48	-0.88	0.38	1.40
bs(age, df=4, lower_bound=10, upper_bound=50)[2]	-3.69	0.03	8.88	-21.10	13.72	0.00	9.09e+05	-0.42	0.68	0.56
bs(age, df=4, lower_bound=10, upper_bound=50)[3]	-6.02	0.00	6.75	-19.26	7.21	0.00	1351.35	-0.89	0.37	1.43
fin	-0.37	0.69	0.19	-0.75	0.01	0.47	1.01	-1.93	0.05	4.22
race	0.35	1.42	0.31	-0.26	0.95	0.77	2.60	1.13	0.26	1.95
mar	-0.39	0.67	0.38	-1.15	0.36	0.32	1.43	-1.02	0.31	1.71
paro	-0.10	0.90	0.20	-0.49	0.28	0.61	1.33	-0.53	0.60	0.75
prio	0.09	1.10	0.03	0.04	0.15	1.04	1.16	3.22	<0.005	9.59

Concordance	0.62
Partial AIC	1176.72
log-likelihood ratio test	26.82 on 9 df
-log2(p) of ll-ratio test	9.38

```
Proportional hazard assumption looks okay.
```

```
[]
```

We see may still have potentially some violation, but it's a heck of a lot less. Also, interestingly, when we include these non-linear terms for age, the wexp proportionality violation disappears. It is not uncommon to see changing the functional form of one variable effects other's proportional tests, usually positively. So, we could remove the strata=['wexp'] if we wished.

<> Bin variable and stratify on it

The second option proposed is to bin the variable into equal-sized bins, and stratify like we did with wexp. There is a trade off here between estimation and information-loss. If we have large bins, we will lose information (since

different values are now binned together), but we need to estimate less new baseline hazards. On the other hand, with tiny bins, we allow the age data to have the most “wiggle room”, but must compute many baseline hazards each of which has a smaller sample size. Like most things, the optimial value is somewhere inbetween.

```
rossi_strata_age = rossi.copy()
rossi_strata_age['age_strata'] = pd.cut(rossi_strata_age['age'],
np.arange(0, 80, 3))

rossi_strata_age[['age', 'age_strata']].head()
```

	age	age_strata
0	27	(24, 27]
1	18	(15, 18]
2	19	(18, 21]
3	23	(21, 24]
4	19	(18, 21]

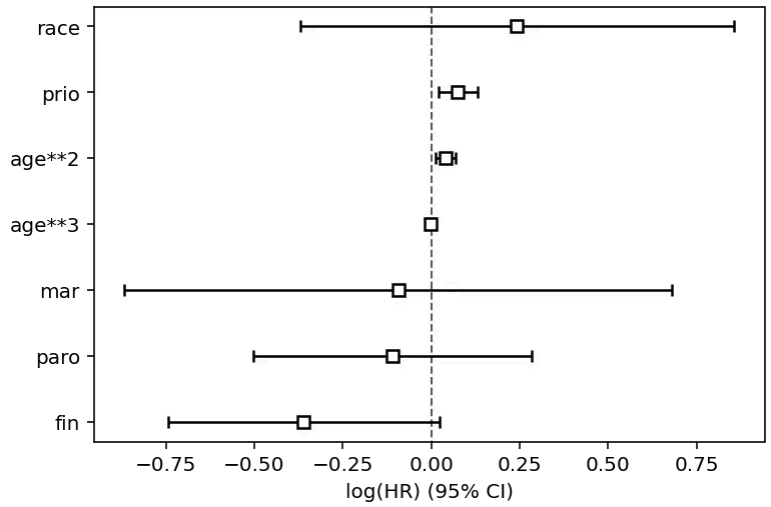
```
# drop the orignal, redundant, age column
rossi_strata_age = rossi_strata_age.drop('age', axis=1)
cph.fit(rossi_strata_age, 'week', 'arrest', strata=['age_strata',
'wexp'])

<lifelines.CoxPHFitter: fitted with 432 total observations, 318
right-censored observations>

cph.print_summary(3, model="stratified age and wexp")
cph.plot()
```

model	lifelines.CoxPHFitter
duration col	'week'
event col	'arrest'
strata	[age_strata, wexp]
baseline estimation	breslow
number of observations	432
number of events observed	114
partial log-likelihood	-386.834
time fit was run	2021-01-19 22:43:57 UTC
model	stratified age and wexp

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
fin	-0.361	0.697	0.196	-0.745	0.023	0.475	1.023	-1.843	0.065	3.935
race	0.242	1.274	0.313	-0.371	0.856	0.690	2.354	0.775	0.439	1.189
mar	-0.093	0.911	0.395	-0.867	0.682	0.420	1.977	-0.235	0.815	0.296
paro	-0.109	0.897	0.201	-0.503	0.284	0.605	1.329	-0.544	0.586	0.770
prio	0.075	1.078	0.028	0.020	0.130	1.020	1.139	2.667	0.008	7.032
age2**	0.040	1.041	0.015	0.012	0.069	1.012	1.071	2.746	0.006	7.374
age3**	-0.003	0.997	0.001	-0.005	-0.001	0.995	0.999	-2.684	0.007	7.101

Concordance	0.617
Partial AIC	787.667
log-likelihood ratio test	24.467 on 7 df
-log2(p) of ll-ratio test	10.051

```
<matplotlib.axes._subplots.AxesSubplot at 0x13c7027c0>
```




```
cph.check_assumptions(rossi_strata_age)
```

```
Proportional hazard assumption looks okay.
```

```
[]
```

<> Introduce time-varying covariates

Our second option to correct variables that violate the proportional hazard assumption is to model the time-varying component directly. This is done in two steps. The first is to transform your dataset into episodic format. This means that we split a subject from a single row into n new rows, and each new row represents some time period for the subject. It's okay that the variables are static over this new time periods — we'll introduce some time-varying covariates later.

See below for how to do this in lifelines:

```
from lifelines.utils import to_episodic_format

# the time_gaps parameter specifies how large or small you want the
periods to be.
rossi_long = to_episodic_format(rossi, duration_col='week',
event_col='arrest', time_gaps=1.)
rossi_long.head(25)
```

	stop	start	arrest	age	age**2	age**3	fin	id	mar	paro	prio	race	wexp
0	1.0	0.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
1	2.0	1.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
2	3.0	2.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
3	4.0	3.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
4	5.0	4.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
5	6.0	5.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
6	7.0	6.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
7	8.0	7.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
8	9.0	8.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
9	10.0	9.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
10	11.0	10.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
11	12.0	11.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
12	13.0	12.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
13	14.0	13.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
14	15.0	14.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
15	16.0	15.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
16	17.0	16.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
17	18.0	17.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
18	19.0	18.0	0	27	5.773341	13.872056	0	0	0	1	3	1	0
19	20.0	19.0	1	27	5.773341	13.872056	0	0	0	1	3	1	0
20	1.0	0.0	0	18	43.523341	-287.133153	0	1	0	1	8	1	0
21	2.0	1.0	0	18	43.523341	-287.133153	0	1	0	1	8	1	0
22	3.0	2.0	0	18	43.523341	-287.133153	0	1	0	1	8	1	0
23	4.0	3.0	0	18	43.523341	-287.133153	0	1	0	1	8	1	0
24	5.0	4.0	0	18	43.523341	-287.133153	0	1	0	1	8	1	0

Each subject is given a new id (but can be specified as well if already provided in the dataframe). This id is used to track subjects over time. Notice the arrest col is 0 for all periods prior to their (possible) event as well.

Above I mentioned there were two steps to correct age. The first was to convert to a episodic format. The second is to create an interaction term between age and stop. This is a time-varying variable.

Instead of CoxPHFitter, we must use CoxTimeVaryingFitter instead since we are working with a episodic dataset.

```
rossi_long['time*age'] = rossi_long['age'] * rossi_long['stop']

from lifelines import CoxTimeVaryingFitter
ctv = CoxTimeVaryingFitter()

ctv.fit(rossi_long,
        id_col='id',
        event_col='arrest',
        start_col='start',
        stop_col='stop',
        strata=['wexp'])
```

```
<lifelines.CoxTimeVaryingFitter: fitted with 19809 periods, 432
subjects, 114 events>
```

```
ctv.print_summary(3, model="age * time interaction")
```

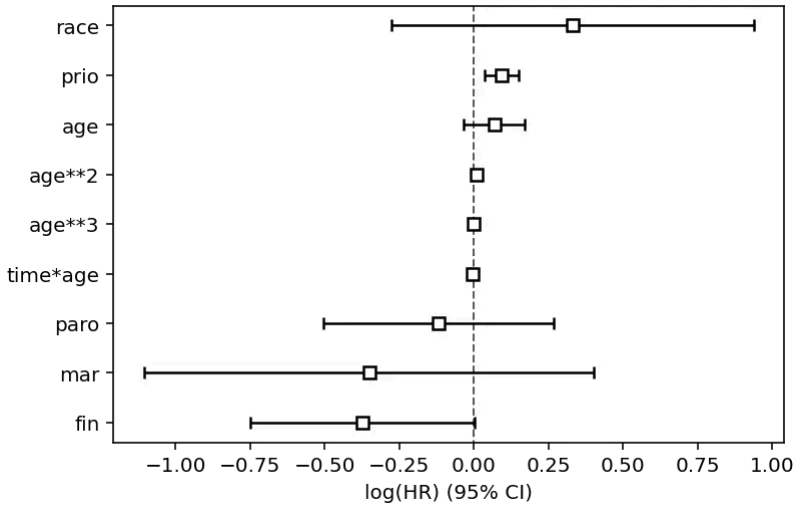
model	lifelines.CoxTimeVaryingFitter
event col	'arrest'
strata	[wexp]
number of subjects	432
number of periods	19809
number of events	114
partial log-likelihood	-573.878
time fit was run	2021-01-19 22:44:00 UTC
model	age * time interaction

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
age	0.068	1.070	0.053	-0.035	0.171	0.966	1.187	1.296	0.195	2.359
age2**	0.008	1.008	0.005	-0.002	0.017	0.998	1.017	1.542	0.123	3.023
age3**	-0.000	1.000	0.000	-0.001	0.000	0.999	1.000	-1.090	0.276	1.860
fin	-0.373	0.689	0.191	-0.748	0.002	0.473	1.002	-1.947	0.052	4.279
mar	-0.351	0.704	0.385	-1.105	0.403	0.331	1.497	-0.912	0.362	1.467
paro	-0.118	0.889	0.197	-0.504	0.268	0.604	1.307	-0.599	0.549	0.865
prio	0.092	1.097	0.029	0.035	0.149	1.036	1.161	3.183	0.001	9.424
race	0.331	1.392	0.309	-0.275	0.937	0.759	2.553	1.070	0.285	1.813
time*age	-0.005	0.995	0.001	-0.008	-0.002	0.992	0.998	-3.100	0.002	9.012

Partial AIC	1165.755
log-likelihood ratio test	37.791 on 9 df
-log2(p) of ll-ratio test	15.683

```
ctv.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13cc249a0>
```



In the above scaled Schoenfeld residual plots for age, we can see there is a slight negative effect for higher time values. This is confirmed in the output of the CoxTimeVaryingFitter: we see that the coefficient for time*age is -0.005.

<> Do I need to care about the proportional hazard assumption?

You may be surprised that often you don't need to care about the proportional hazard assumption. There are many reasons why not:

If your goal is survival prediction, then you don't need to care about proportional hazards. Your goal is to maximize some score, irrelevant of how predictions are generated. Given a large enough sample size, even very small violations of proportional hazards will show up. There are legitimate reasons to assume that all datasets will violate the proportional hazards assumption. This is detailed well in Stensrud & Hernán's "Why Test for Proportional Hazards?". "Even if the hazards were not proportional, altering the model to fit a set of assumptions fundamentally changes the scientific question. As Tukey said,"Better an approximate answer to the exact question, rather than an exact answer to the approximate question." If you were to fit the Cox model in the presence of non-proportional hazards, what is the net effect? Slightly less power. In fact, you can recover most of that power with robust

standard errors (specify robust=True). In this case the interpretation of the (exponentiated) model coefficient is a time-weighted average of the hazard ratio–I do this every single time." from AdamO, slightly modified to fit lifelines. Given the above considerations, the status quo is still to check for proportional hazards. So if you are avoiding testing for proportional hazards, be sure to understand and able to answer why you are avoiding testing.

References

The following resources were extremely helpful in making this post. Check them out if you want to learn more about survival analysis.

1. Harrell Jr., Frank E.. Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis
2. 2012, Survival Analysis, Self Learning Book
3. Assessing the Fit of the Cox Model
4. Hypothesis testing section of Survival package in R
5. The Cox Proportional Hazards Regression Model (slides)
6. Barry Leventhal's presentation with business insights
7. The Survival Package in R
8. The Lifelines Package in Python

Survival Analysis | Data Science | Analysis

Written by Reza Rashetnia, PhD

Follow

25 Followers · Writer for Nerd For Tech

Data Scientist @ Intel

More from Reza Rashetnia, PhD and Nerd For Tech

Reza Rashetnia, PhD in Nerd For Tech

Probabilistic Programming Using PyMC3: TESLA Stock Stochastic...

Probabilistic Programming allows to define probabilistic models and develop Bayesian...

5 min read · Mar 31, 2021

84 2

Madokai in Nerd For Tech

Will AI Replace DevOps Engineers?

Explore how the advent of AI affects the world of DevOps. Discussing AI's role in enhancing...

4 min read · Mar 13, 2024

282 6

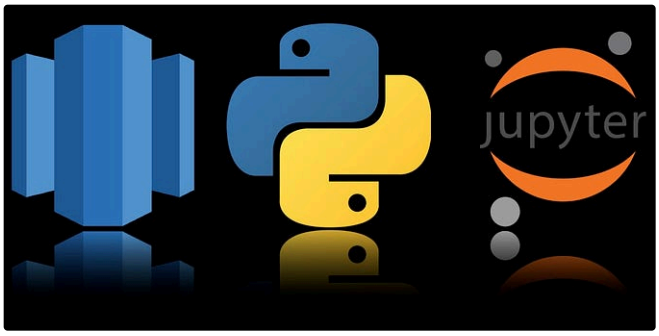
Dick Dowdell in Nerd For Tech

Is OOP Relevant Today?

Does object-oriented programming still have a purpose?

13 min read · Mar 15, 2024

36



Reza Rashetnia, PhD

Query Amazon Redshift Data Using Python (Pandas and NumPy)

Amazon Redshift data warehouse is part of the cloud-computing platform Amazon Web...

3 min read · Aug 28, 2020

See all from Reza Rashetnia, PhD

See all from Nerd For Tech

Recommended from Medium

Marco Cerliani in Towards Data Science

Rethinking Survival Analysis: How to Make your Model Produce...

Time to Event Forecasting with Simple ML Approaches

✨ · 6 min read · Nov 29, 2022

204

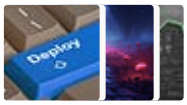
Rajeshneupane

Multinomial Logistic regression in python and statsmodels

Now, we can use the statsmodels api to run the multinomial logistic regression, the data...

3 min read · Mar 14, 2024

Lists



Predictive Modeling w/ Python

20 stories · 1141 saves

Practical Guides to Machine Learning

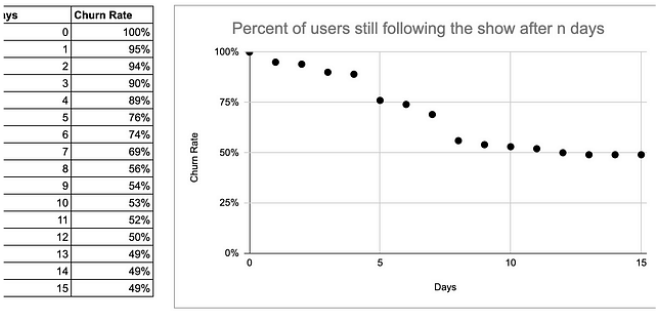
10 stories · 1371 saves

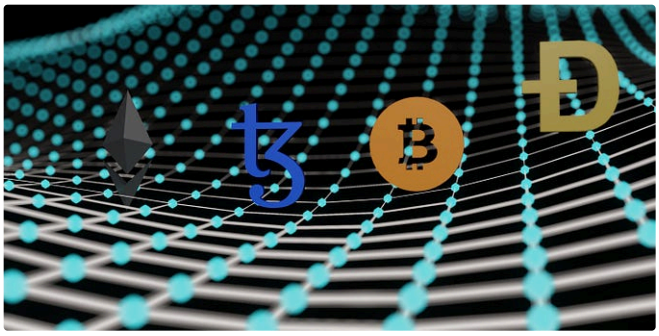
Coding & Development

11 stories · 586 saves

ChatGPT prompts

47 stories · 1492 saves





Tarek Amr in Python in Plain English

Kaplan-Meier Survival Analysis

And how this non-parametric approach is used to fit the survival function.

✦ · 6 min read · Apr 23, 2024

3

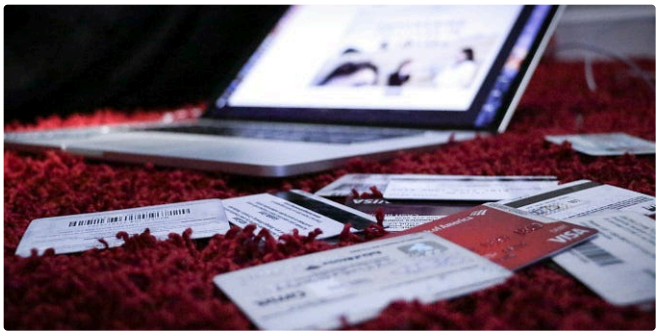
ChenDataBytes

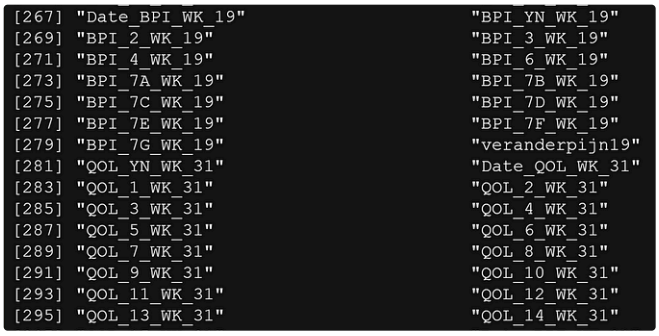
Customer Lifetime Value Prediction: Part 2—Survival Rate

CLV Prediction for New Customers

6 min read · Nov 8, 2023

17





Muhammad Arief Rachman

Credit Scoring Prediction

Scorecard modeling

15 min read · Nov 14, 2023

26 1

Dr. Marc Jacobs

Joint modelling of longitudinal and survival data

Dealing with MNAR data in a palliative pancreatic cancer setting

26 min read · Dec 11, 2023

19 1

See more recommendations