ABSTRACT: This report explains the process of building a survival model selection function in Python that could be used in further analysis and illustrates it with a few datasets. Since model selection applied the same way to Cox proportional hazards models as accelerated failure time models, this function performs its methods according to which regression method is appropriate for a given dataset. After checking assumptions, three examples with different datasets illustrated the selection process with two different AFT models and a Cox proportional hazard model.

Cox Proportional Hazards and AFT Model Selection in Python

By Julian Gomez

Statistical modeling applies model selection in several fields. When medical research focuses on examining relationships between baseline explanatory variables and clinical outcomes, factors that are considered significant are termed prognosis. In other words, prognosis plays a substantial role in patient treatment and decision making when it comes to survival analysis (1). Most of the time, a lot of covariates are included in data sets. So in practice, researchers build large parametric models in order to reduce possible modeling bias. At first glance, it is often difficult to determine which subset of variables is significant for a hazard function. Therefore, there are some techniques, among many, that have been extended to the context of survival analysis. Model selections comprise of, for instance, subset variable selection and stepwise deletion (2).

Background

One of the most popular models in survival regression is the Cox proportional hazards model. Like all survival analysis models, this regression model attempts to represent the hazard rate $h(t|x)$ as a function of time t and some covariates. The Cox proportional hazards model asserts that the log-hazard of an individual is a linear function of their static covariates and a population-level baseline hazard, $h_0(t)$, that changes over time. The equation is:

$$h_t(t) = \mathrm{e}^{\beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{pi}} h_0(t)$$

Where the only time component is in $h_0(t)$. The partial hazard is a time-invariant scalar factor that only increases or decreases the baseline hazard. Moreover, changes in the covariates will only increase or decrease the baseline hazard (3).

Another type of model that encompasses a wider range of survival time distributions is the accelerated failure time model, commonly abbreviated as AFT. These models are used when the Cox Proportional model assumptions are not met. Like the Cox PH model, the AFT model has an intuitive linear regression interpretation (3). In a log-logistic model, the survival times follow a log-logistic distribution while in a log-normal model, the survival times follow a log-normal distribution (4). The log-logistic hazard function in python is,

$$H_i(t) = -\log\left(1 + \left[\frac{t}{exp\left(\alpha_1 x_{1i} + \cdots + \alpha_p x_{pi} + \alpha_{int}\right)}\right]^{exp(\beta)}\right)$$

and the log-normal hazard function in python is,

$$H_i(t) = -\log\left(1 - \Phi\left[\frac{1}{e^{\sigma}}\left(log\, t - e^{\alpha_1 x_{1i} + \cdots + \alpha_p x_{pi} + \mu_{int}}\right)\right]\right).$$

The survival model selection process consists of 5 steps where it adds or drops variables based on their significance to the aspiring model. The first step is to fit the model with no variables except the time and event variables. The second is to add one variable at a time if it is significant at $\chi^2(0.1, 1)$. The third step is to drop one variable from step 2 at a time if it is not significant. The fourth is to add each of the remaining terms at a time to the model from step 3 if one is significant. The fifth and final step is to drop one variable from the step 4 model at a time if it is not significant. The result of this step is the final model to use for analysis and this progress is applied equally on Cox proportional models and AFT models. In this project, I am going to present five model selection functions for Cox and AFT models in Python programming and depending on model assumptions, present examples of this function using three datasets.

Methods

The individual function from step 1, named mod_sel1(), has four arguments: data frame, time variable, event variable, and regression fitter function. This last argument lets the function know which regression method to use when fitting the models [i.e. CoxPHFitter() for Cox proportional models, LogLogisticAFTFitter() for log-logistic models]. The first thing that the mod_sel1 does is set up the provided fitter. Then, it fits the model accordingly with the dataset as well as the time and status variables. And finally, it prints the model summary.

The individual function for step 2 tends to add one covariate at a time and test whether the likelihood ratio of the null model and the model with one term is significant. The arguments of mod_sel2 are data frame, time, status, list of covariates to fit, regression function, alpha parameter set to 0.1, degrees of freedom set to 1, and print_model set to false. First, the function starts by setting the imputed regression function. Then, it sets the chi-square critical value for likelihood ratio tests. Next, the function fits the first model, the null model, and extracts its log likelihood. In order to fit the second model, I defined two empty lists, one for the likelihood ratio test statistics and another for their respective p-values. So I built a for loop that would iterate through the list of the provided terms. Inside the loop, the function fits the second model with time, status, and the potential term. Next, it extracts the log likelihood and calculates the likelihood ratio and p-value with 1 degree of freedom. Finally, the loop appends the likelihood ratio and p-value to the lists defined before the loop. After the loop, I defied an empty list for the significant terms and created a second for loop that iterates through the list of the likelihood ratio statistics from the previous loop by index. Inside the loop, I defined an if-ten statement that would test whether each likelihood ratio is greater than $\chi^2(0.1, 1)$. If so, then the function

appends the term to the defined empty list, prints its name, likelihood ratio statistic, p-value, and a statement saying, ‘Add term’. Otherwise, he function prints the term, statistic, p-value and a statement saying ‘do not add term.’ After the loop, the function prints the list of added terms as well as the model with these terms if the user inputs print_model = True in the function.

The individual function for step 3 tends to drop one covariate at a time and test whether the likelihood ratio of the step 2 model and the model without one of the terms is significant. Like the previous function, mod_sel3 has arguments of data frame, time, status, list of terms from step 2, regression function, alpha, degrees of freedom, and print_model. First, the function starts by setting the imputed regression function as well as the chi-square critical value. Then, it fits the first model with the terms from step 2 and extracts its log likelihood. The terms for the second model are the same as those of the first model except one that could potentially be dropped. Therefore, built a for loop that creates a list of lists where each one does not include one item at a time. Specifically, this loop iterates through the list of terms from step 2 by index and removes the item corresponding to that index with the .pop(index) function and appends that list inside an empty list. After this, I fit the second model the same way as step 2, except that the loop would go through each list created by the previous loop. In other words, the loop fits the second model without one particular term, extracts its log likelihood, computes the likelihood ratio statistic and p-value, and appends them to their respective defined lists. After this, like step 2, a third for loop tests whether each of the likelihood ratios is greater than the critical value. If greater, then the term not included in the second model is significant and is added an empty list. Otherwise, the term not included is dropped and is not added to the list. The result after the for loop is the list of terms that were not dropped as well as the model with this term.

The step 4 function is basically the same structure as that of the second step. It consists of testing whether adding any of the omitted terms so far improves the model. In other words, the potential terms will be the ones not included in the models from steps 2 and 3. The function mod_sel4 has arguments of data frame, time, status, list of terms from step 3, list of terms not in step 3, regression function, alpha, degrees of freedom, and print model. It starts by defining the regression function as well as the chi-squared threshold. Then, it defines the first model with the terms from step 3 and extracts its log likelihood estimate. Like in step 2, the for loop for the second model will add one item from the ‘list of terms not in step 3’ argument at a time. Then it appends the log likelihood ratios as well as the p-values for the next loop. This next loop tests

whether any of the likelihood ratios from the previous loop are greater than the critical value. If any term is significant, it is added to an empty list defined before the loop. If any terms are added, the resulting list is combined with that of step 3. If the list is empty, that means that no terms were added, and the result would be the same as step 3.

The fifth step consists of testing whether dropping any of the step 4 covariates improves the model. The body of this function is the same as that of step 3, which means that if no terms were added on step 4, then this step would not be necessary and will print the same outputs as step 3 and 4. If any terms were added in step 4, on the other hand, then the function will execute just like step 3 where the second function will iteratively fit a list without one particular term and later test whether the reduced model is significant. In the end, whether any terms are dropped or not, the function will print the final list of terms as well as the final model for Cox proportional hazard or AFT models.

The selection function, mod_sel(), combines all of the five steps discussed above. This function contains the body for each step where the resulting lists of terms are passed on to the next steps. For instance, the resulting list of terms from step two is passed on to the first model of step 3, where it is treated the same way as the list of terms provided in this step's individual function. The output of this function is the progress of which variables are added and which ones are removed from the very beginning to the final model.

In order to determine the optimal regression function for a dataset, I first check whether a dataset satisfies the Cox proportional hazards assumption. If it does, I perform model selection using the CoxPHFitter() in the function argument. Otherwise, I proceed by checking the suitability of an AFT model to the dataset. For this, I calculate a transformation of the survival function of the log-logistic or log-normal distribution using the Kaplan-Meier estimate. For this, a line plot comparing the log transformation of the time variable vs. the log-odds of survival beyond time *t* is produced. For log-logistic and log-normal, the log-odds of discontinuation equations are

$$\log\left\{\frac{S(t)}{1-S(t)}\right\} \text{ and } \Phi^{-1}\{1-S(t)\}$$

respectively. A plot resulting in a straight line is thus suitable for any of the mentioned AFT models (4). And therefore, I perform the model selection for whichever AFT model gives the highest log likelihood, even when the line plots for log-logistic and log-normal seem suitable.

Results

The first I used to apply my model selection functions is named veteran, which is about a lung cancer study by the Veterans' Administration (5). This dataset comes from the R survival package and contains 8 variables and 137 observations. The variables listed are treatment (standard or test), cell type, time, status, Karnofski performance score (karno), months from diagnosis to randomization, age, and prior therapy. I modified this dataset by adding indicators for the cell type variable, which are small, squamous, or large. For this dataset **figure 1** shows that the Cox Proportional assumption is not met at the 0.05 level. So, therefore, I proceeded with testing which AFT models are appropriate for this dataset. **Figure 2a** shows the log-logistic plot and **figure 2b** shows the log-normal plot. At first glance, the log-logistic plot seems more linear than that of the log-normal plot. Moreover, when comparing the log-likelihoods of both models, we can see that the one corresponding to the log-logistic model is the highest of the three AFT models. Hence, I am going to apply the model selection of this dataset using the log-logistic fitter. To get the optimal AFT model, I imputed the dataset, time, status, list of terms treatment, karno, diagtime, age, prior, cell_small, cell_squamous, and cell_large and LogLogisticFitter() in the mod_sel() function. With an alpha of 0.1 and 1 degree of freedom, step 1 adds karno, small sell, squamous cell, and large cell. In step 3, since the small cell variable is not significant, only karno, squamous cell, and large cells are left. Step 4 does not add any of the omitted terms and step 5 also does not change the model. Hence, the final log-logistic model here contains the karnofski performance score, the squamous cell, and the large cell indicator variables. The final model is in **table1a.**

The second dataset is about a German breast cancer study group (6). It contains 10 variables and 686 observations. The original variables are hormonal therapy, age, menopausal status, tumor size, tumor grade, number of positive nodes, progesterone receptor, estrogen receptor, time, and censoring indicator (status). To fit this dataset, I converted the hormonal therapy and menopausal variables to dummy variables. Then, I created two indicator variables from the tumor grade levels II and III. After I dropped the original tumor grade variable, I tested whether the dataset meets the Cox proportional hazard assumption and it turns out from **figure 3** that the assumption is not met with the tumor grade III indicator. Therefore, it was better to use model selection of an AFT model for this dataset as well. The Kaplan Meier plot in **figure 4a** tests whether a log-logistic model is appropriate for this dataset and **figure 4b** tests whether a log-normal model is appropriate. While both seem to be quite linear, the log-likelihood for the log-normal model for this dataset is higher than that of the log-logistic and the Weibull models.

Hence, I used the log-normal fitter for the model selection. The null model's log likelihood for this dataset is -2618.89. Step 2 adds hormonal therapy, tumor size, positive nodes, progesterone receptor, estrogen receptor, and tumor grade 3. Step 3 only drops estrogen receptor from this list of variables. Next, step 4 adds back tumor grade 2, while step 5 does not drop anything. Hence, the final model has the terms hormonal therapy, tumor size, positive node count, progesterone receptor, tumor grade III and tumor grade II. The final model is in **table 1b.**

The third dataset is about survival in advanced lung cancer patients from the North Central Cancer Treatment Group from the R survival package (7). This dataset has 10 variables and 228 observations. After importing this dataset, I converted the status variable to a dummy variable and removed the missing values, which reduced the number of observations to 167. The lung cancer data contains the variables institution code, survival time in days, status, age, sex, ECOG performance score on a scale 0 to 5, Karnofsky performance score rated by physician, Karnofsky score rated by patient, calories consumed at meals, and weight loss in the last six months. Unlike the previous two datasets, the cox proportional hazards assumption is met for this dataset. Therefore, I applied model selection with the Cox Proportional Hazards fitter. Step 1 prints the null model log likelihood of -508.12. Step 2 adds all of the variables except physician calories consumed, weight loss, and ECOG score of 1. Step 2 drops age, physician Karnofsky score, and patient Karnofsky score, leaving sex and ECOG score of 2 and 3. Step 4 does not add any of the previously dropped terms and step 5 does not drop either of the two variables left. Hence, the final model contains sex and ECOG score of 2 and 3. The result is in **table1c.**

Conclusion

Since there was no function or algorithm that performs model selection in Python, the goal of this project was to create five separate functions for each step and combine them together into one function so that I would not only get an optimal model in seconds, but also demonstrate how to choose the right regression method before performing the selection itself. The complete survival model selection thus helped select the best model for each of the three datasets according to Cox proportional hazard models or advanced failure time models. Since some datasets were unable to meet the proportional hazards assumption, I alternatively fitted with an AFT model, depending on visualizations log likelihood coefficient of the Weibull, log-

logistic, and log-normal models. Therefore, in order to fit an appropriate model, there are assumptions to be met and variables to omit in order to compute results needed for analysis.

Works Cited:

1 - Lin, Chen-Yen, and Susan Halabi. "On Model Specification and Selection of the Cox Proportional Hazards Model." *Statistics in Medicine*, U.S. National Library of Medicine, 20 Nov. 2013, www.ncbi.nlm.nih.gov/pmc/articles/PMC3795916/.

2 – Fan, Jianqing, and Runze Li. "VARIABLE SELECTION FOR COX'S PROPORTIONAL HAZARDS MODEL AND FRAILTY MODEL." *The Annals of Statistics*, vol. 30, no. 1, 2002, pp. 74–99., projecteuclid.org/euclid.aos/1015362185.

3 - "Survival Regression" *Survival Regression - Lifelines 0.23.3 Documentation*, lifelines.readthedocs.io/en/latest/Survival%20Regression.html.

4 - Collett, D. *Modelling Survival Data in Medical Research*. Chapman & Hall/CRC, 2015.

5 - Therneau, Terry M, and Thomas Lumley. "Veteran: Veterans' Administration Lung Cancer Study in Survival: Survival Analysis." *Veteran: Veterans' Administration Lung Cancer Study in Survival: Survival Analysis*, 9 Nov. 2019, rdrr.io/cran/survival/man/veteran.html.

6 - Hothorn, Torsten. "GBSG2: German Breast Cancer Study Group 2 in TH.data: TH's Data Archive." *GBSG2: German Breast Cancer Study Group 2 in TH.data: TH's Data Archive*, 2 May 2019, rdrr.io/cran/TH.data/man/GBSG2.html.

7 - Therneau, Terry M, and Thomas Lumley. "Lung: NCCTG Lung Cancer Data in Survival: Survival Analysis." *Lung: NCCTG Lung Cancer Data in Survival: Survival Analysis*, 9 Nov. 2019, rdrr.io/cran/survival/man/lung.html.

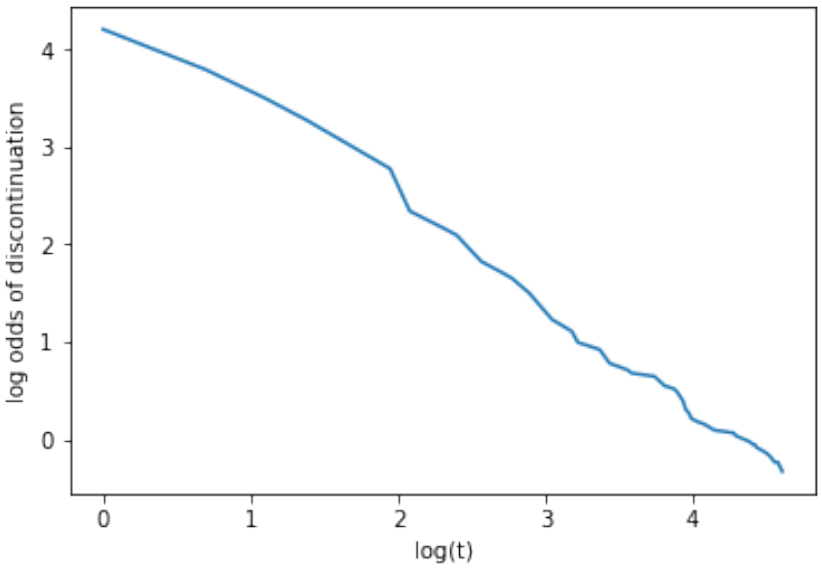
Appendix

Figure 1 - Cox proportional assumption test for the veteran dataset with null hypothesis set as 'no violations' for each variable. If a p-value is low, that means that the variable does not meet the Cox proportional hazards assumption.

```
---
                    test_statistic      p  -log2(p)
age           km              5.42   0.02      5.65
              rank            5.53   0.02      5.74
cell_large    km              0.03   0.87      0.20
              rank            0.02   0.89      0.16
cell_small    km              2.97   0.08      3.56
              rank            2.90   0.09      3.49
cell_squamous km              3.06   0.08      3.64
              rank            3.22   0.07      3.78
diagtime      km              3.08   0.08      3.66
              rank            2.88   0.09      3.48
karno         km             13.19 <0.005     11.80
              rank           13.65 <0.005     12.15
prior         km              4.63   0.03      4.99
              rank            4.63   0.03      4.99
trt           km              0.11   0.74      0.43
              rank            0.08   0.77      0.37
```

Figure 2 – Line plots for AFT model suitability for the veteran dataset

a) Log-logistic distribution seems quite linear



b) Log-linear seems quite linear

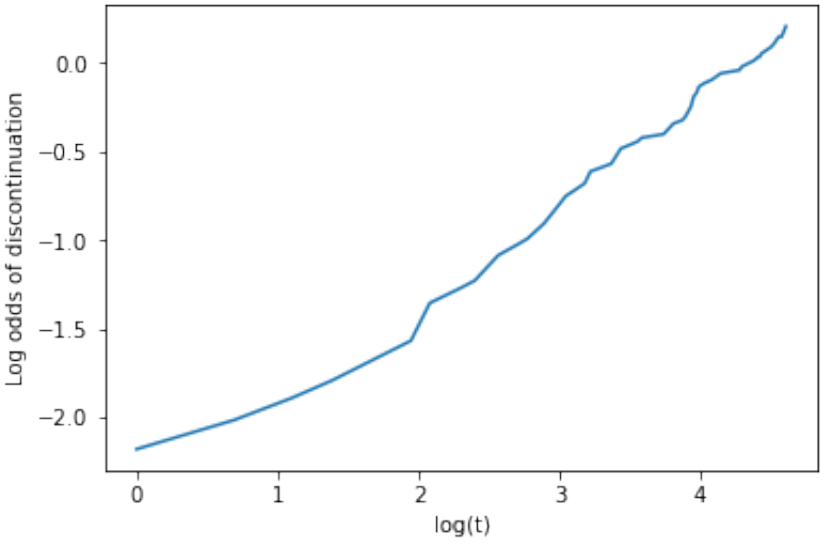
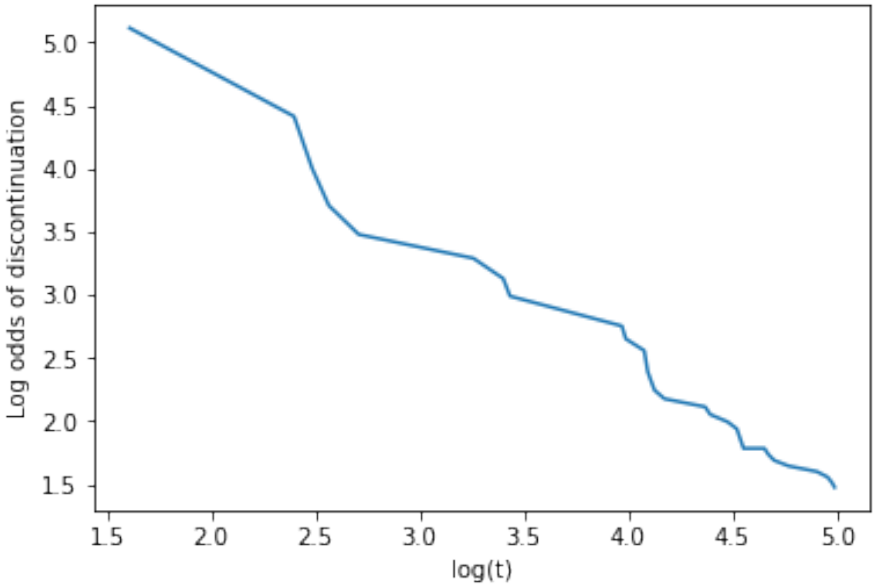


Figure 3 – Cox proportional hazards assumption test for the German breast cancer study data with null hypothesis set as 'no violations' for each variable

		test_statistic	p	-log2(p)
age	km	2.94	0.09	3.53
	rank	2.80	0.09	3.41
estrec	km	1.00	0.32	1.66
	rank	1.41	0.23	2.09
horTh	km	0.20	0.65	0.62
	rank	0.21	0.65	0.62
menostat	km	0.00	0.99	0.01
	rank	0.00	0.94	0.08
pnodes	km	0.59	0.44	1.18
	rank	0.58	0.45	1.16
progrec	km	1.22	0.27	1.89
	rank	1.09	0.30	1.75
tgrade2	km	1.45	0.23	2.13
	rank	1.84	0.17	2.52
tgrade3	km	4.74	0.03	5.08
	rank	5.87	0.02	6.02
tsize	km	0.18	0.67	0.58
	rank	0.18	0.67	0.58

Figure 4 – Line plots for AFT model suitability for the German breast cancer study data

a) Log-logistic seems quite linear



c) Log-Normal also seems quite linear

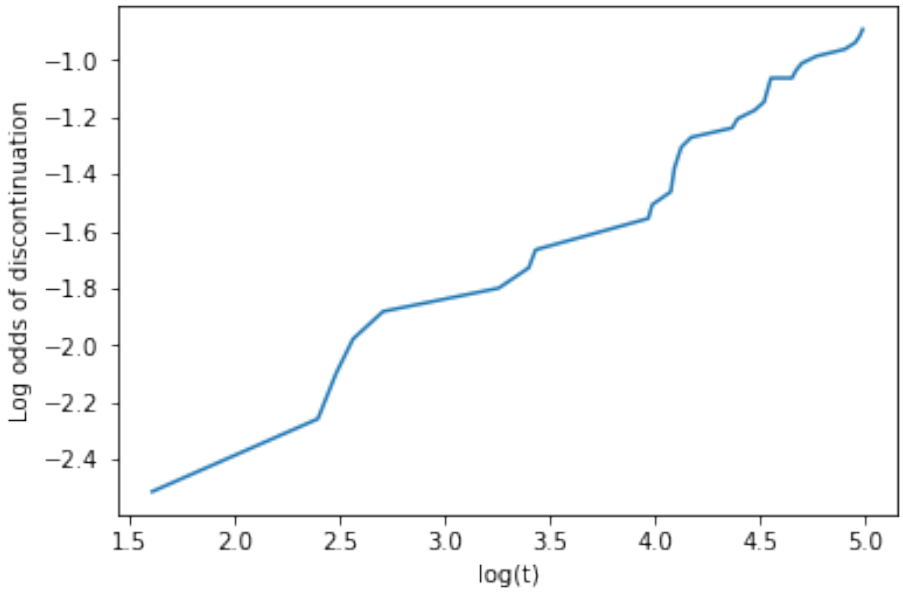


Figure 5 - Cox proportional hazards assumptions test for the lung cancer data

```
cph = CoxPHFitter()
cph.fit(lung, 'time', 'status')
cph.check_assumptions(lung, p_value_threshold= 0.05, show_plots= True)
```

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

Table 1 – Final model equations from the results section

a) Final log-logistic model for the Veteran Lung Cancer data

$$H_i(t) = -\log\left(1 + \left[\frac{t}{e^{0.04karno_i + 0.73cellsquamous_{i_i} + 0.7cellarge_i + 1.77}}\right]^{e^{0.54}}\right)$$

b) Final log-normal model for the German Breast cancer study group data

$$H_i(t) = -\log\left(1 - \Phi\left[\frac{1}{e^{-0.02}}\left(log\, t - e^{0.31horTh_i - 0.01tsize_i - 0.05pnodes_i - 0.65tgrade3_i - 0.5tgrade2 + 8.06}\right)\right]\right)$$

c) Final Cox Proportional Hazards model for the Lung Cancer data

$$h_t(t) = \mathrm{e}^{-0.49sex_i + 0.73exog2p_i} h_0(t)$$

Cox and AFT Model Selection

December 12, 2019

```
In [27]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from scipy.stats import chi2
         import math
         import matplotlib.pyplot as plt
         from scipy.stats import norm
         from lifelines import CoxPHFitter
         from lifelines import WeibullAFTFitter
         from lifelines import LogNormalAFTFitter
         from lifelines import LogLogisticAFTFitter
         from lifelines import KaplanMeierFitter
```

0.1 Step 1: Null model

```
In [279]: def mod_sel1(data, time_var, event_var, fun):
              fitter = fun
              mod1 = fitter.fit(data[[time_var] + [event_var]],
                                duration_col = time_var, event_col = event_var)
              mod1.print_summary()
```

0.2 Step 2: Add one variable at a time

```
In [280]: ## step 2: iter
          ## vars1: variables from the null model
          ## vars2: variables to test

          def mod_sel2(data, time_var, event_var, terms, fun, alpha = 0.1,
                       df = 1, print_model = False):

              print('Model Selection using ' + str(fun) + '\n')

              fitter = fun
              threshold = chi2.ppf(1 - alpha, df)

              mod1 = fitter.fit(data[[time_var] + [event_var]],
                                duration_col= time_var, event_col= event_var)
              ll1 = mod1.log_likelihood_
```

```
    print('Step 2\n')

    ll_diffs = []
    p_vals = []
    for term in terms:
        mod2 = fitter.fit(data[[time_var] + [event_var] + [term]],
                          duration_col= time_var, event_col= event_var)
        ll2 = mod2.log_likelihood_
        res = (2*ll2) - (2*ll1)
        p = chi2.sf(res, 1)
        ll_diffs.append(res)
        p_vals.append(p)

    step2_mod = []
    for j in range(len(ll_diffs)):
        if ll_diffs[j] > threshold:
            print('Variable Fitted: ' + terms[j])
            print('LL difference: ' + str(ll_diffs[j]))
            print('p-value: ' + str(p_vals[j]))
            print('Add ' + terms[j])
            step2_mod.append(terms[j])
            print(' ')
        else:
            print('Variable Fitted: ' + terms[j])
            print('LL difference: ' + str(ll_diffs[j]))
            print('p-value: ' + str(p_vals[j]))
            print('Do not add ' + terms[j])
            print(' ')
    print('Step 2 Variables: ' + str(step2_mod) + '\n')

    if print_model == True:
        model = fitter.fit(data[[time_var] + [event_var] + step2_mod],
                           duration_col = time_var, event_col= event_var)
        model.print_summary()
```

0.3 Step 3: Drop one variable at a time

```
In [281]: ## step 3: iter
          ## vars1: variables from the previous step
          ## vars2: variables to test

          def mod_sel3(data, time_var, event_var, terms, fun,
                       alpha = 0.1, df= 1, print_model=False):

              print('Step 3 Model Selection using ' + str(fun) + '\n')

              fitter = fun
              threshold = chi2.ppf(1 - alpha, df)
```

```
mod1 = fitter.fit(data[[time_var] + [event_var] + terms],
                  duration_col= time_var, event_col= event_var)
ll1 = mod1.log_likelihood_


term_lists = []
for x in range(len(terms)):
    term_l = list(terms)
    term_l.pop(x)
    term_lists.append(term_l)

ll_diffs = []
p_vals = []
for term_list in term_lists:
    mod2 = fitter.fit(data[[time_var] + [event_var] + term_list],
                      duration_col = time_var, event_col= event_var)
    ll2 = mod2.log_likelihood_
    res = (2*ll1) - (2*ll2)
    p = chi2.sf(res, 1)
    ll_diffs.append(res)
    p_vals.append(p)

step3_mod = []
for k in range(len(ll_diffs)):
    if ll_diffs[k] > threshold:
        print('Variables Fitted: ' + str(term_lists[k]))
        print('LL difference: ' + str(ll_diffs[k]))
        print('p-value: ' + str(p_vals[k]))
        print('Do not drop ' + terms[k])
        step3_mod.append(terms[k])
        print(' ')
    else:
        print('Variables Fitted: ' + str(term_lists[k]))
        print('LL difference: ' + str(ll_diffs[k]))
        print('p-value: ' + str(p_vals[k]))
        print('Drop ' + terms[k])
        print(' ')
print('Step 3 Variables: ' + str(step3_mod) + '\n')

if print_model == True:
    model = fitter.fit(data[[time_var] + [event_var] + step3_mod],
                       duration_col = time_var, event_col= event_var)
    model.print_summary()
```

0.4 Step 4: Add one the remaining variables at a time

In [282]:
```
## step 4: iter
## terms1: variables from the last step
```

```
## terms2: variables needed to test

def mod_sel4(data, time_var, event_var, terms1, terms2, fun,
             alpha = 0.1, df = 1, print_model=False):
    print('Step 4 Model Selection using ' + str(fun) + '\n')

    fitter = fun
    threshold = chi2.ppf(1 - alpha, df)

    mod1 = fitter.fit(data[[time_var] + [event_var] + terms1],
                      duration_col= time_var, event_col= event_var)
    ll1 = mod1.log_likelihood_

    ll_diffs = []
    p_vals = []
    for term in terms2:
        mod2 = fitter.fit(data[[time_var] + [event_var] + terms1 + [term]],
                          duration_col= time_var, event_col= event_var)
        ll2 = mod2.log_likelihood_
        res = (2*ll2) - (2*ll1)
        p = chi2.sf(res, 1)
        ll_diffs.append(res)
        p_vals.append(p)

    step4_mod = []
    for j in range(len(ll_diffs)):
        if ll_diffs[j] > threshold:
            print('Variable Fitted: ' + terms2[j])
            print('LL difference: ' + str(ll_diffs[j]))
            print('p-value: ' + str(p_vals[j]))
            print('Add ' + terms2[j])
            step4_mod.append(terms2[j])
            print(' ')
        else:
            print('Variable Fitted: ' + terms2[j])
            print('LL difference: ' + str(ll_diffs[j]))
            print('p-value: ' + str(p_vals[j]))
            print('Do not add ' + terms2[j])
            print(' ')
    print('Step 4 Variables: ' + str(terms1 + step4_mod) + '\n')

    if print_model == True:
        model = fitter.fit(data[[time_var] + [event_var] + terms1 + step4_mod],
                           duration_col= time_var, event_col= event_var)
        model.print_summary()
```

0.5 Step 5: Drop one variable at a time and print final model

In [283]:
```
## step 5: iter
## vars1: variables from the previous step
## vars2: variables to test drop

def mod_sel5(data, time_var, event_var, terms, fun, alpha = 0.1, df= 1):
    print('Step 4 Model Selection using ' + str(fun) + '\n')

    fitter = fun
    threshold = chi2.ppf(1 - alpha, df)
    mod1 = fitter.fit(data[[time_var] + [event_var] + terms],
                      duration_col= time_var, event_col= event_var)
    ll1 = mod1.log_likelihood_

    term_lists = []
    for x in range(len(terms)):
        term_list = list(terms)
        term_list.pop(x)
        term_lists.append(term_list)

    ll_diffs = []
    p_vals = []
    for term_l in term_lists:
        mod2 = fitter.fit(data[[time_var] + [event_var] + term_l],
                          duration_col = time_var, event_col= event_var)
        ll2 = mod2.log_likelihood_
        res = (2*ll1) - (2*ll2)
        p = chi2.sf(res, 1)
        ll_diffs.append(res)
        p_vals.append(p)

    final_mod = []
    for k in range(len(ll_diffs)):
        if ll_diffs[k] > threshold:
            print('Variables Fitted: ' + str(term_lists[k]))
            print('LL difference: ' + str(ll_diffs[k]))
            print('p-value: ' + str(p_vals[k]))
            print('Do not drop ' + terms[k])
            final_mod.append(terms[k])
            print(' ')

        else:
            print('Variables Fitted: ' + str(term_lists[k]))
            print('LL difference: ' + str(ll_diffs[k]))
            print('p-value: ' + str(p_vals[k]))
            print('Drop ' + terms[k])
            print(' ')
```

```
        print('Step 5 Variables: ' + str(final_mod) + '\n')
        mod_final = fitter.fit(data[[time_var] + [event_var] + final_mod],
                               duration_col = time_var, event_col= event_var)
        mod_final.print_summary()
```

0.6 Steps 1 - 5

```
In [278]: def mod_sel(data, time_var, event_var, terms, fun, alpha = 0.1, df= 1):
        print('Model Selection using ' + str(fun) + '\n')
        fitter = fun
        threshold = chi2.ppf(1 - alpha, df)
        print('Step 1\n')

        mod2_1 = fitter.fit(data[[time_var] + [event_var]],
                            duration_col= time_var, event_col= event_var)
        ll1 = mod2_1.log_likelihood_
        print('Null LL: ' + str(ll1) + '\n')

        print('Step 2\n')
        ## step 2: add one terma at a time

        ll_diffs = []
        p_vals = []
        for term in terms:
            mod2_2 = fitter.fit(data[[time_var] + [event_var] + [term]],
                                duration_col= time_var, event_col= event_var)
            ll2 = mod2_2.log_likelihood_
            res = (2*ll2) - (2*ll1)
            p = chi2.sf(res, 1)
            ll_diffs.append(res)
            p_vals.append(p)

        step2_mod = []
        for j in range(len(ll_diffs)):
            if ll_diffs[j] > threshold:
                print('Variable Fitted: ' + terms[j])
                print('LL Ratio: ' + str(ll_diffs[j]))
                print('p-value: ' + str(p_vals[j]))
                print('Add ' + terms[j])
                step2_mod.append(terms[j])
                print(' ')
            else:
                print('Variable Fitted: ' + terms[j])
                print('LL Ratio: ' + str(ll_diffs[j]))
                print('p-value: ' + str(p_vals[j]))
                print('Do not add ' + terms[j])
```

```
        print(' ')
print('Step 2 Variables: ' + str(step2_mod) + '\n')

## Step 3: drop one variable at a time

print('Step 3\n')
mod3_1 = fitter.fit(data[[time_var] + [event_var] + step2_mod],
                    duration_col= time_var, event_col= event_var)
ll1 = mod3_1.log_likelihood_


term_lists = []
for x in range(len(step2_mod)):
    term_l = list(step2_mod)
    term_l.pop(x)
    term_lists.append(term_l)

ll_diffs = []
p_vals = []
for term_list in term_lists:
    mod3_2 = fitter.fit(data[[time_var] + [event_var] + term_list],
                        duration_col = time_var, event_col= event_var)
    ll2 = mod3_2.log_likelihood_
    res = (2*ll1) - (2*ll2)
    p = chi2.sf(res, 1)
    ll_diffs.append(res)
    p_vals.append(p)

step3_mod = []
for k in range(len(ll_diffs)):
    if ll_diffs[k] > threshold:
        print('Variables Fitted: ' + str(term_lists[k]))
        print('LL Ratio: ' + str(ll_diffs[k]))
        print('p-value: ' + str(p_vals[k]))
        print('Do not drop ' + step2_mod[k])
        step3_mod.append(step2_mod[k])
        print(' ')
    else:
        print('Variables Fitted: ' + str(term_lists[k]))
        print('LL Ratio: ' + str(ll_diffs[k]))
        print('p-value: ' + str(p_vals[k]))
        print('Drop ' + step2_mod[k])
        print(' ')
print('Step 3 Variables: ' + str(step3_mod) + '\n')

## Step 4: drop one variable at a time
## vars1: varibales from previous step, vars2:variables to be tested
print('Step 4\n')
```

```
step4_terms = [x for x in terms if not x in step3_mod]

mod4_1 = fitter.fit(data[[time_var] + [event_var] + step3_mod],
                    duration_col= time_var, event_col= event_var)
ll1 = mod4_1.log_likelihood_

ll_diffs = []
p_vals = []
for term in step4_terms:
    mod4_2 = fitter.fit(data[[time_var] + [event_var] + step3_mod + [term]],
                        duration_col= time_var, event_col= event_var)
    ll2 = mod4_2.log_likelihood_
    res = (2*ll2) - (2*ll1)
    p = chi2.sf(res, 1)
    ll_diffs.append(res)
    p_vals.append(p)

step4_mod = []
for j in range(len(ll_diffs)):
    if ll_diffs[j] > threshold:
        print('Variable Fitted: ' + step4_terms[j])
        print('LL Ratio: ' + str(ll_diffs[j]))
        print('p-value: ' + str(p_vals[j]))
        print('Add ' + step4_terms[j])
        step4_mod.append(step4_terms[j])
        print(' ')
    else:
        print('Variable Fitted: ' + step4_terms[j])
        print('LL Ratio: ' + str(ll_diffs[j]))
        print('p-value: ' + str(p_vals[j]))
        print('Do not add ' + step4_terms[j])
        print(' ')
step4_mod2 = step3_mod + step4_mod
print('Step 4 Variables: ' + str(step4_mod2) + '\n')


## step 5: drop one variable at a time
print('Step 5\n')
mod5_1 = fitter.fit(data[[time_var] + [event_var] + step4_mod2],
                    duration_col= time_var, event_col= event_var)
ll1 = mod5_1.log_likelihood_

term_lists = []
for x in range(len(step4_mod2)):
    term_list = list(step4_mod2)
    term_list.pop(x)
    term_lists.append(term_list)
```

```
    ll_diffs = []
    p_vals = []
    for term_l in term_lists:
        mod5_2 = fitter.fit(data[[time_var] + [event_var] + term_l],
                            duration_col = time_var, event_col= event_var)
        ll2 = mod5_2.log_likelihood_
        res = (2*ll1) - (2*ll2)
        p = chi2.sf(res, 1)
        ll_diffs.append(res)
        p_vals.append(p)

    final_mod = []
    for k in range(len(ll_diffs)):
        if ll_diffs[k] > threshold:
            print('Variables Fitted: ' + str(term_lists[k]))
            print('LL Ratio: ' + str(ll_diffs[k]))
            print('p-value: ' + str(p_vals[k]))
            print('Do not drop ' + str(step4_mod2[k]))
            final_mod.append(step4_mod2[k])
            print(' ')

        else:
            print('Variables Fitted: ' + str(term_lists[k]))
            print('LL Ratio: ' + str(ll_diffs[k]))
            print('p-value: ' + str(p_vals[k]))
            print('Drop ' + str(step4_mod2[k]))
            print(' ')

    print('Step 5 Variables: ' + str(final_mod) + '\n')
    mod_final = fitter.fit(data[[time_var] + [event_var] + final_mod],
                           duration_col = time_var, event_col= event_var)
    mod_final.print_summary()
```

0.7 AFT Model Selection

```
In [284]: def aft_sel(data, time_var, event_var):

    llf = LogLogisticAFTFitter()
    llf_mod = llf.fit(data, time_var, event_var)

    lnf = LogNormalAFTFitter()
    lnf_mod = lnf.fit(data, time_var, event_var)

    wf = WeibullAFTFitter()
    wf_mod = wf.fit(data, time_var, event_var)

    print('Log-Logistic:' + str(llf_mod.log_likelihood_))
    print('Log-Normal:' + str(lnf_mod.log_likelihood_))
```

```
print('Weibull:' + str(wf_mod.log_likelihood_) + '\n')

print(max([llf_mod.log_likelihood_, lnf_mod.log_likelihood_, wf_mod.log_likelihoo
## select the model corresponding to the maximum log-likelihood
```

0.7.1 Veteran Data

```
In [268]: veteran = pd.read_csv('/Users/Julian/Documents/STAT 311/Data/veteran.csv')
          veteran.head()
```

Out[268]:

	trt	celltype	time	status	karno	diagtime	age	prior
0	1	squamous	72	1	60	7	69	0
1	1	squamous	411	1	70	5	64	10
2	1	squamous	228	1	60	3	38	0
3	1	squamous	126	1	60	9	63	10
4	1	squamous	118	1	70	11	65	10

```
In [269]: veteran['celltype'].value_counts()
```

```
Out[269]: smallcell    48
          squamous     35
          large        27
          adeno        27
          Name: celltype, dtype: int64
```

```
In [270]: ## indicator for cell type - adeno is baseline
          cell_small = []
          cell_squamous = []
          cell_large = []

          for i in veteran['celltype']:
              if i == 'smallcell':
                  cell_small.append(1)
                  cell_squamous.append(0)
                  cell_large.append(0)
              elif i == 'squamous':
                  cell_small.append(0)
                  cell_squamous.append(1)
                  cell_large.append(0)
              elif i == 'large':
                  cell_small.append(0)
                  cell_squamous.append(0)
                  cell_large.append(1)
              else:
                  cell_small.append(0)
                  cell_squamous.append(0)
                  cell_large.append(0)

          veteran['cell_small'] = cell_small
```

```
veteran['cell_squamous'] = cell_squamous
veteran['cell_large'] = cell_large

veteran['trt'] = veteran['trt'] - 1
veteran['prior'] = veteran['prior']/10

veteran = veteran[['trt', 'time', 'status', 'karno', 'diagtime', 'age',
       'prior', 'cell_small', 'cell_squamous', 'cell_large']]

veteran.head()
```

Out[270]:

	trt	time	status	karno	diagtime	age	prior	cell_small	cell_squamous	cell_large
0	0	72	1	60	7	69	0.0	0	1	0
1	0	411	1	70	5	64	1.0	0	1	0
2	0	228	1	60	3	38	0.0	0	1	0
3	0	126	1	60	9	63	1.0	0	1	0
4	0	118	1	70	11	65	1.0	0	1	0

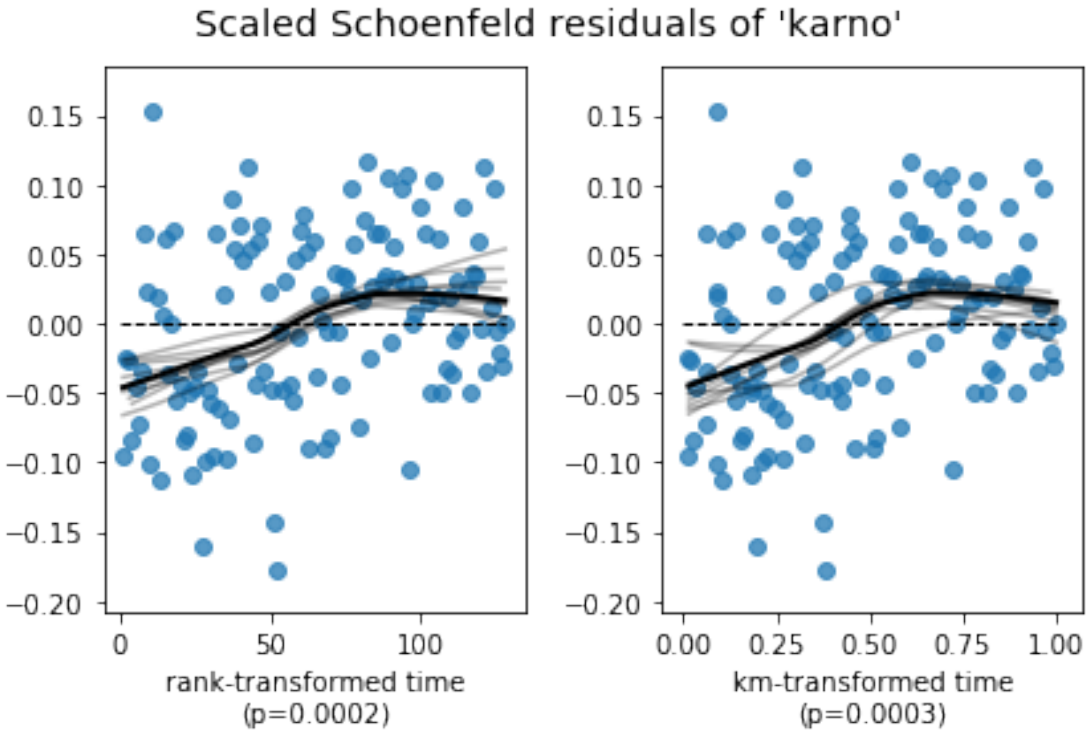
Check proportiona Hazards assumption

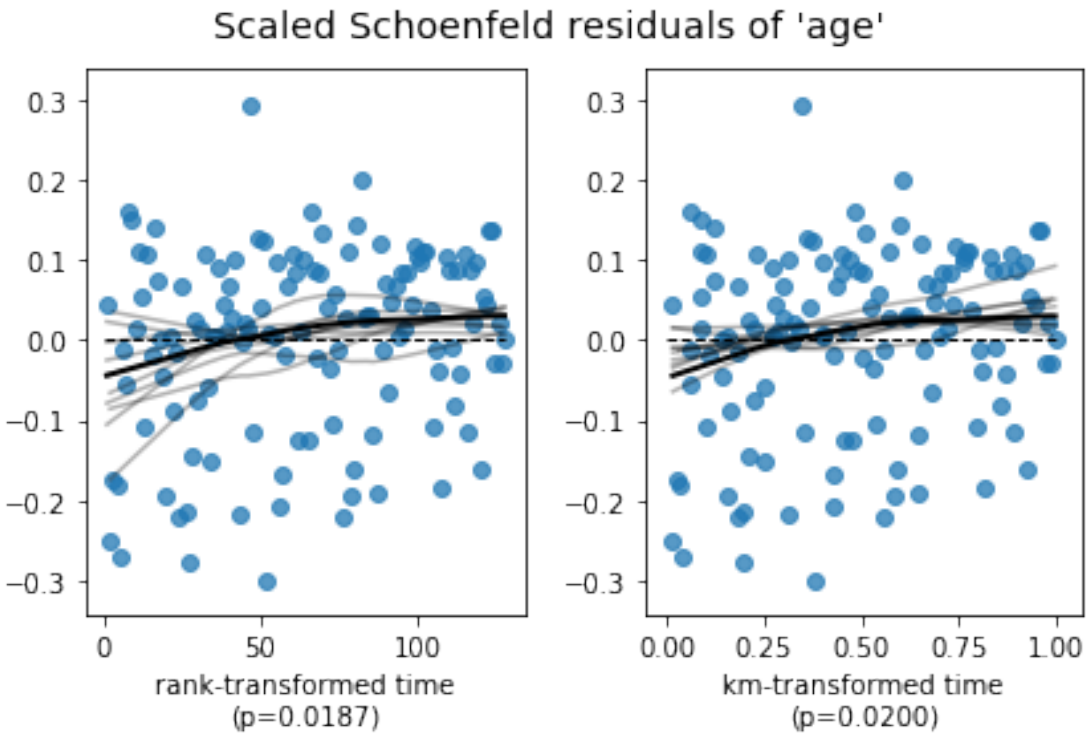
In [287]:
```
cph = CoxPHFitter()
cph.fit(veteran, 'time', 'status')
cph.check_assumptions(veteran, p_value_threshold= 0.05, show_plots= True)
```

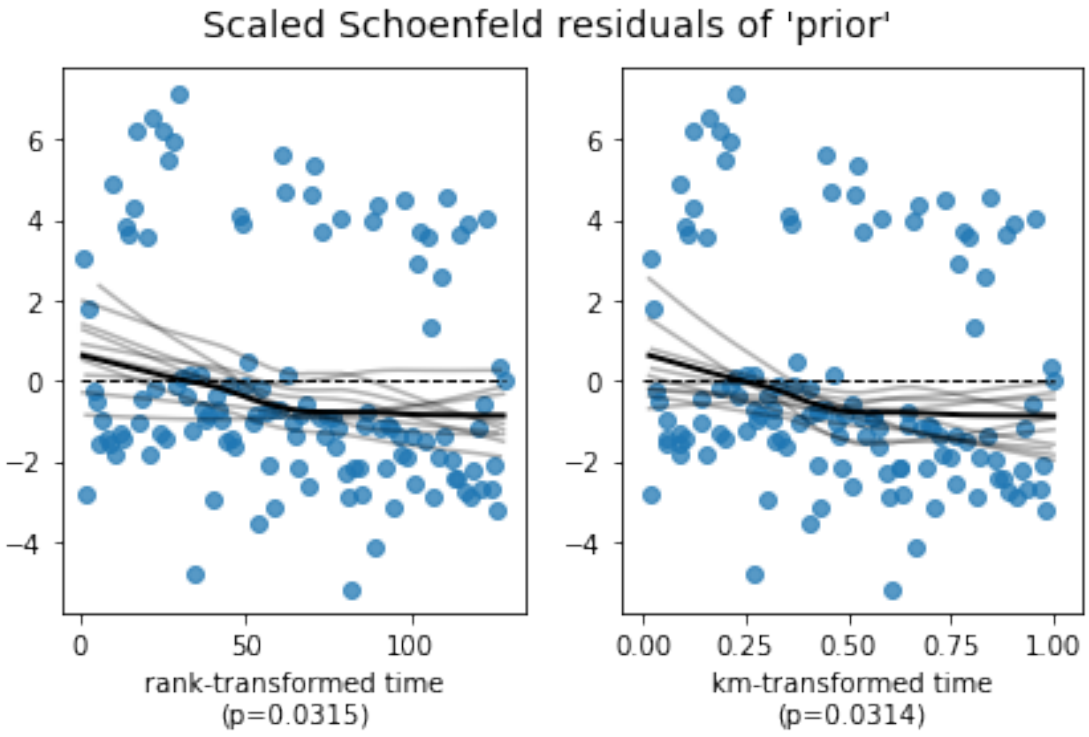
```
The ``p_value_threshold`` is set at 0.05. Even under the null hypothesis of no violations, some
covariates will be below the threshold by chance. This is compounded when there are many covari
Similarly, when there are lots of observations, even minor deviances from the proportional haza
assumption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

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

<lifelines.StatisticalResult>
         test_name = proportional_hazard_test
 null_distribution = chi squared
degrees_of_freedom = 1

---
                    test_statistic      p  -log2(p)
age           km              5.42   0.02      5.65
```

```
                rank            5.53   0.02      5.74
cell_large      km              0.03   0.87      0.20
                rank            0.02   0.89      0.16
cell_small      km              2.97   0.08      3.56
                rank            2.90   0.09      3.49
cell_squamous   km              3.06   0.08      3.64
                rank            3.22   0.07      3.78
diagtime        km              3.08   0.08      3.66
                rank            2.88   0.09      3.48
karno           km             13.19 <0.005     11.80
                rank           13.65 <0.005     12.15
prior           km              4.63   0.03      4.99
                rank            4.63   0.03      4.99
trt             km              0.11   0.74      0.43
                rank            0.08   0.77      0.37


1. Variable 'karno' failed the non-proportional test: p-value is 0.0002.

   Advice 1: the functional form of the variable 'karno' might be incorrect. That is, there may
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 'karno' using pd.cut, and then specify it in `strata=['ka
...]` 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
below.


2. Variable 'age' failed the non-proportional test: p-value is 0.0187.

   Advice 1: the functional form of the variable 'age' might be incorrect. That is, there may h
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
below.


3. Variable 'prior' failed the non-proportional test: p-value is 0.0314.

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

```
---
[A]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[B]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[C]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[D]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[E]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
```




Scaled Schoenfeld residuals of 'age'
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
50
100
rank-transformed time
(p=0.0187)
0.00
0.25
0.50
0.75
1.00
km-transformed time
(p=0.0200)

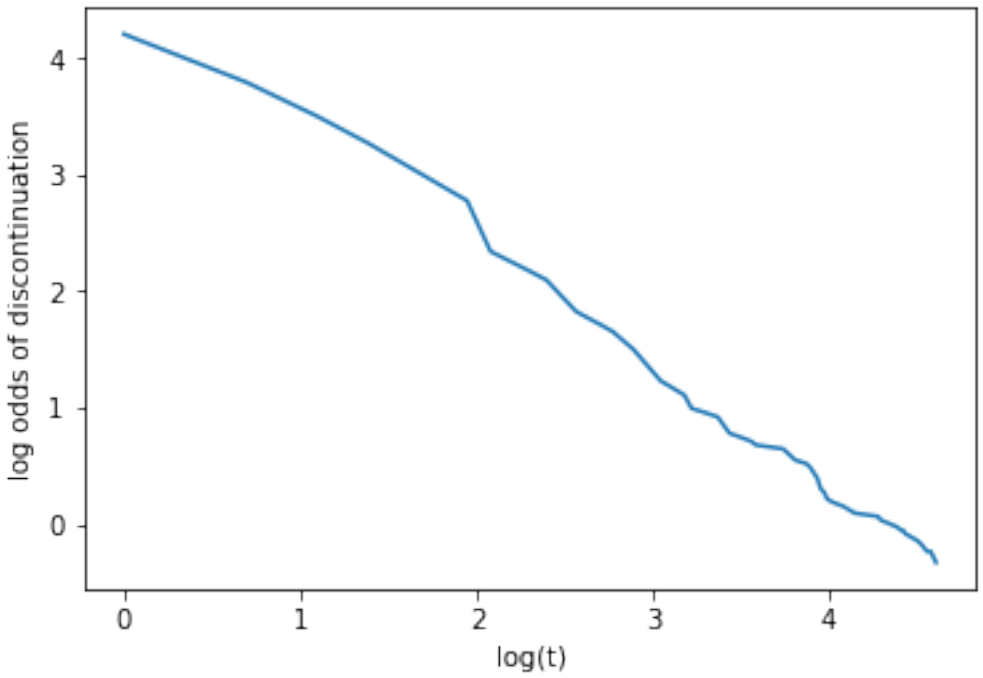

Scaled Schoenfeld residuals of 'prior'
6
4
2
0
−2
−4
0
50
100
rank-transformed time
(p=0.0315)
0.00
0.25
0.50
0.75
1.00
km-transformed time
(p=0.0314)

```
In [271]: kmf = KaplanMeierFitter()
          kmf.fit(veteran['time'], veteran['status'])
          kt_sf = kmf.survival_function_ ## veteran survival function
          kt_sf= kt_sf[1:len(kt_sf)-1]

          ls = []
          for x in kt_sf['KM_estimate']:
              ls.append(math.log(x/(1-x)))

          lt= []
          for x in kt_sf.index:
              lt.append(math.log(x))

          kt_sf['ln[s/(s-1)]'] = ls
          kt_sf['ln(t)'] = lt
          #kt_sf
```

```
/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:14: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:15: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  from ipykernel import kernelapp as app
```

```
In [272]: plt.xlabel('log(t)')
          plt.ylabel('log odds of discontinuation')
          plt.plot(kt_sf['ln(t)'], kt_sf['ln[s/(s-1)]']) ## this is log-logistic
```

```
Out[272]: [<matplotlib.lines.Line2D at 0x1a193f4320>]
```

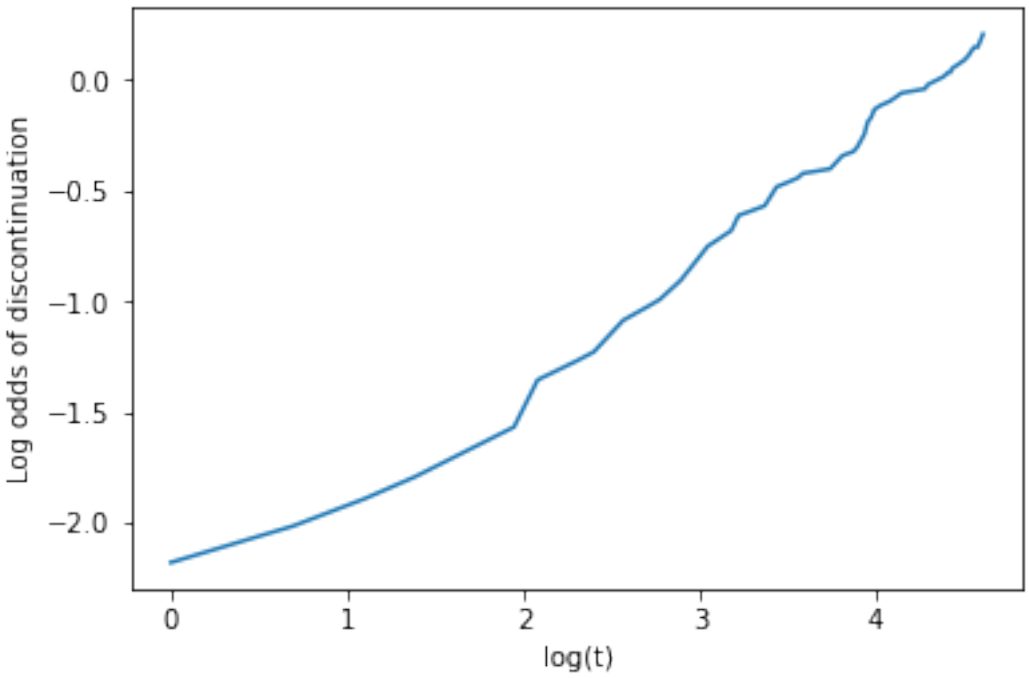



```
In [273]: lp= []
          for x in kt_sf['KM_estimate']:
              k = (1 - x)
              lp.append(norm.ppf(k))

          kt_sf['psi(t)'] = lp
          #kt_sf
```

```
/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [274]: ## ln-normal
          plt.xlabel('log(t)')
          plt.ylabel('Log odds of discontinuation')
          plt.plot(kt_sf['ln(t)'], kt_sf['psi(t)']) ## a log normal model can be used
```

```
Out[274]: [<matplotlib.lines.Line2D at 0x1a19592470>]
```




```
In [34]: aft_sel(veteran, 'time', 'status') ## use log logistic with the veteran data

Log-Logistic:-711.9425568192644
Log-Normal:-714.6349369510474
Weibull:-715.5513293540253

-711.9425568192644
```

0.8 Dataset 2

```
In [116]: from lifelines.datasets import load_gbsg2

In [196]: gbsg = load_gbsg2()
          gbsg.head()
```

Out[196]:

	horTh	age	menostat	tsize	tgrade	pnodes	progrec	estrec	time	cens
0	no	70	Post	21	II	3	48	66	1814	1
1	yes	56	Post	12	II	7	61	77	2018	1
2	yes	58	Post	35	II	9	52	271	712	1
3	yes	59	Post	17	II	4	60	29	1807	1
4	no	73	Post	35	II	1	26	65	772	1

```
In [190]: gbsg.describe()
```

```
Out[190]:              age       tsize      pnodes      progrec       estrec  \
          count  686.000000  686.000000  686.000000   686.000000   686.000000
          mean    53.052478   29.329446    5.010204   109.995627    96.252187
          std     10.120739   14.296217    5.475483   202.331552   153.083963
          min     21.000000    3.000000    1.000000     0.000000     0.000000
          25%     46.000000   20.000000    1.000000     7.000000     8.000000
          50%     53.000000   25.000000    3.000000    32.500000    36.000000
          75%     61.000000   35.000000    7.000000   131.750000   114.000000
          max     80.000000  120.000000   51.000000  2380.000000  1144.000000

                       time        cens
          count   686.000000  686.000000
          mean   1124.489796    0.435860
          std     642.791948    0.496231
          min       8.000000    0.000000
          25%     567.750000    0.000000
          50%    1084.000000    0.000000
          75%    1684.750000    1.000000
          max    2659.000000    1.000000

In [43]: gbsg['horTh'].value_counts()

Out[43]: no     440
         yes    246
         Name: horTh, dtype: int64

In [44]: gbsg['menostat'].value_counts()

Out[44]: Post    396
         Pre     290
         Name: menostat, dtype: int64

In [45]: gbsg['tgrade'].value_counts()

Out[45]: II     444
         III    161
         I       81
         Name: tgrade, dtype: int64

In [197]: ## create indicators for hormonal therapy

          horTh2 = []
          for x in gbsg['horTh']:
              if x == 'yes':
                  horTh2.append(1)
              else:
                  horTh2.append(0)

          gbsg['horTh'] = horTh2
```

In [198]:
```
## create indicators for menostat
## 1 if post, 0 for pre

menostat2 = []
for x in gbsg['menostat']:
    if x == 'Post':
        menostat2.append(1)
    else:
        menostat2.append(0)

gbsg['menostat'] = menostat2
```

In [199]:
```
## create indicators for tgrade
## grade 3 and 4

tgrade2 = []
tgrade3 = []
for x in gbsg['tgrade']:
    if x == 'II':
        tgrade2.append(1)
        tgrade3.append(0)
    elif x == 'III':
        tgrade2.append(0)
        tgrade3.append(1)
    else:
        tgrade2.append(0)
        tgrade3.append(0)

gbsg['tgrade2'] = tgrade2
gbsg['tgrade3'] = tgrade3
```

In [200]:
```
gbsg = gbsg.drop(['tgrade'], axis = 1)
gbsg.head()
```

Out[200]:

	horTh	age	menostat	tsize	pnodes	progrec	estrec	time	cens	tgrade2	\
0	0	70	1	21	3	48	66	1814	1	1	
1	1	56	1	12	7	61	77	2018	1	1	
2	1	58	1	35	9	52	271	712	1	1	
3	1	59	1	17	4	60	29	1807	1	1	
4	0	73	1	35	1	26	65	772	1	1	

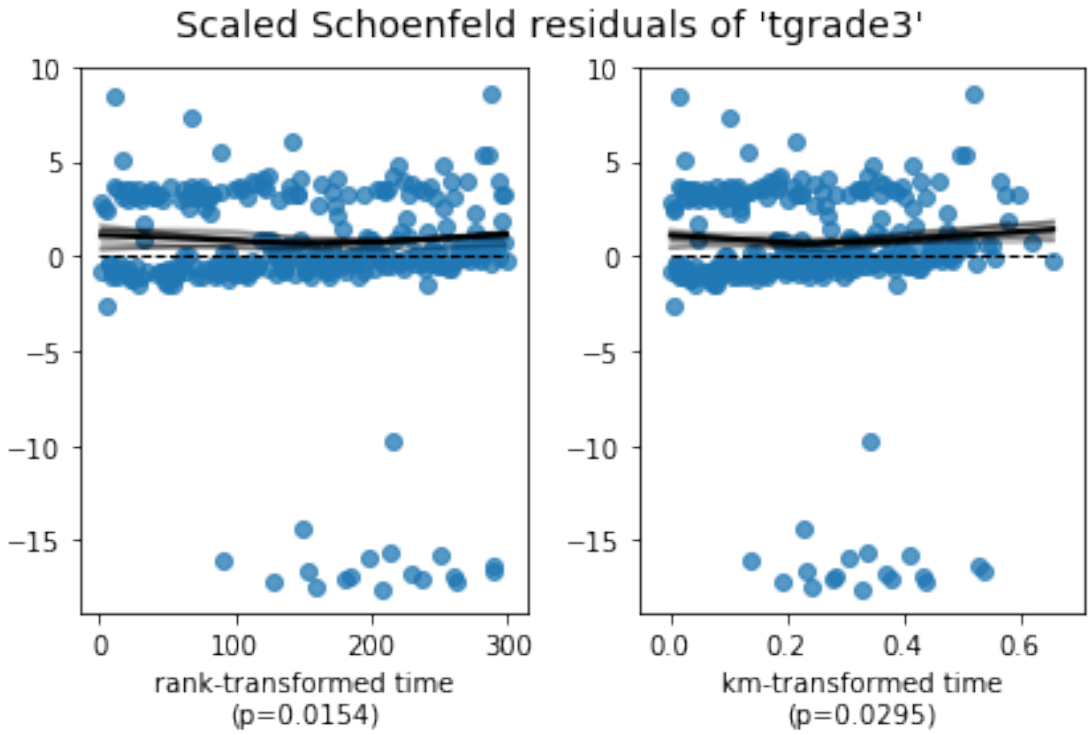
	tgrade3
0	0
1	0
2	0
3	0
4	0

Check proportional hazars assumption

```
In [286]: cph = CoxPHFitter()
          cph.fit(gbsg, 'time', 'cens')
          cph.check_assumptions(gbsg, 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 covari
Similarly, when there are lots of observations, even minor deviances from the proportional haza
assumption will be flagged.

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

<lifelines.StatisticalResult>
         test_name = proportional_hazard_test
 null_distribution = chi squared
degrees_of_freedom = 1

---
               test_statistic    p  -log2(p)
age      km              2.94 0.09      3.53
         rank            2.80 0.09      3.41
estrec   km              1.00 0.32      1.66
         rank            1.41 0.23      2.09
horTh    km              0.20 0.65      0.62
         rank            0.21 0.65      0.62
menostat km              0.00 0.99      0.01
         rank            0.00 0.94      0.08
pnodes   km              0.59 0.44      1.18
         rank            0.58 0.45      1.16
progrec  km              1.22 0.27      1.89
         rank            1.09 0.30      1.75
tgrade2  km              1.45 0.23      2.13
         rank            1.84 0.17      2.52
tgrade3  km              4.74 0.03      5.08
         rank            5.87 0.02      6.02
tsize    km              0.18 0.67      0.58
         rank            0.18 0.67      0.58


1. Variable 'tgrade3' failed the non-proportional test: p-value is 0.0154.

   Advice: with so few unique values (only 2), you can include `strata=['tgrade3', ...]` in the
in `.fit`. See documentation in link [E] below.

---
[A]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assum
[B]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assum
```

[C] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[D] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu
[E] https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assu



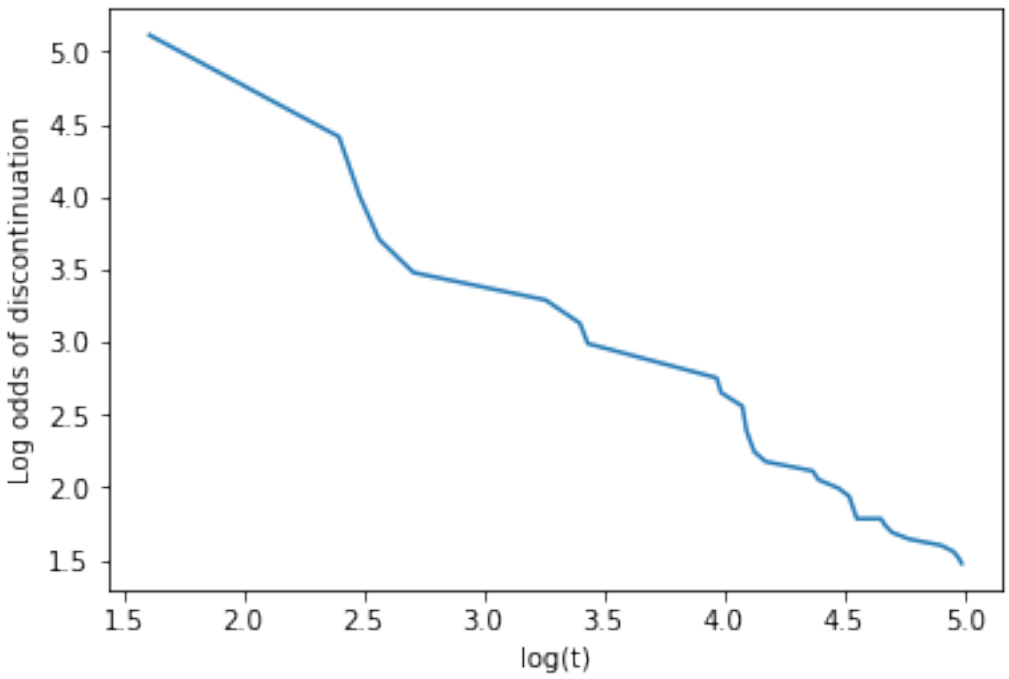
In [69]:
```
kmf = KaplanMeierFitter()
kmf.fit(gbsg['time'], gbsg['cens'])
kt_sf = kmf.survival_function_ ## veteran survival function
kt_sf= kt_sf[72:len(kt_sf)]

ls = []
for x in kt_sf['KM_estimate']:
    ls.append(math.log(x/(1-x)))

lt= []
for x in kt_sf.index:
    lt.append(math.log(x))

kt_sf['ln[s/(s-1)]'] = ls
kt_sf['ln(t)'] = lt
```

```
/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:14: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:15: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  from ipykernel import kernelapp as app
```

```
In [265]: plt.xlabel('log(t)')
          plt.ylabel('Log odds of discontinuation')
          plt.plot(kt_sf['ln(t)'], kt_sf['ln[s/(s-1)]']) ## this is quite log-logistic
```

```
Out[265]: [<matplotlib.lines.Line2D at 0x1a199aaf28>]
```

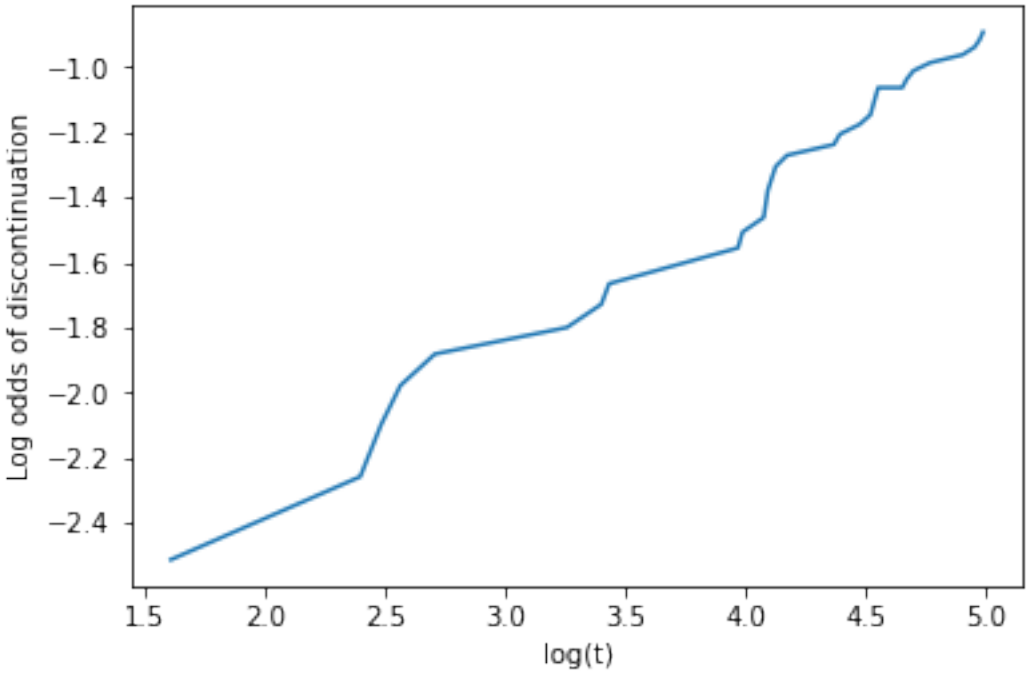



```
In [71]: lp= []
         for x in kt_sf['KM_estimate']:
             k = (1 - x)
             lp.append(norm.ppf(k))

         kt_sf['psi(t)'] = lp
         #kt_sf
```

```
/Users/Julian/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [266]: ## ln-normal
          plt.xlabel('log(t)')
          plt.ylabel('Log odds of discontinuation')
          plt.plot(kt_sf['ln(t)'], kt_sf['psi(t)']) ## a log normal model can be used
```

```
Out[266]: [<matplotlib.lines.Line2D at 0x1a19681a58>]
```




```
In [155]: aft_sel(gbsg, 'time', 'cens') ## use log normal here
```

```
Log-Logistic:-2565.5098360835404
Log-Normal:-2558.5813924298027
Weibull:-2579.6948362244207

-2558.5813924298027
```

0.9 Lung Cancer

```
In [248]: from lifelines.datasets import load_lung
          lung = load_lung()
          lung = lung.dropna()
          lung.shape

Out[248]: (167, 10)

In [249]: lung['status'] = lung['status'] - 1
          lung.head()

Out[249]:    inst  time  status  age  sex  ph.ecog  ph.karno  pat.karno  meal.cal  \
          1   3.0   455       1   68    1      0.0      90.0       90.0    1225.0
          3   5.0   210       1   57    1      1.0      90.0       60.0    1150.0
          5  12.0  1022       0   74    1      1.0      50.0       80.0     513.0
          6   7.0   310       1   68    2      2.0      70.0       60.0     384.0
          7  11.0   361       1   71    2      2.0      60.0       80.0     538.0

             wt.loss
          1     15.0
          3     11.0
          5      0.0
          6     10.0
          7      1.0

In [242]: lung['ph.ecog'].value_counts()

Out[242]: 1.0    81
          0.0    47
          2.0    38
          3.0     1
          Name: ph.ecog, dtype: int64

In [250]: ## create indicators for ECOG
          ecog1 = [] ## ECOG of 1
          ecog2p = [] ## ECOG 2+(2 and 3)
          for x in lung['ph.ecog']:
              if x == 1:
                  ecog1.append(1)
                  ecog2p.append(0)
              elif x == 2 or x == 3:
                  ecog1.append(0)
                  ecog2p.append(1)
              else:
                  ecog1.append(0)
                  ecog2p.append(0)

          lung['ecog1'] = ecog1
```

```
lung['ecog2p'] = ecog2p

lung = lung.drop(['ph.ecog'], axis = 1)
lung.head()
```

Out[250]:

	inst	time	status	age	sex	ph.karno	pat.karno	meal.cal	wt.loss	ecog1	ecog2p
1	3.0	455	1	68	1	90.0	90.0	1225.0	15.0	0	0
3	5.0	210	1	57	1	90.0	60.0	1150.0	11.0	1	0
5	12.0	1022	0	74	1	50.0	80.0	513.0	0.0	1	0
6	7.0	310	1	68	2	70.0	60.0	384.0	10.0	0	1
7	11.0	361	1	71	2	60.0	80.0	538.0	1.0	0	1

Check Cox Proportional Hazards Assumption'

```
In [285]: cph = CoxPHFitter()
          cph.fit(lung, 'time', 'status')
          cph.check_assumptions(lung, p_value_threshold= 0.05, show_plots= True)
```

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

0.10 Use Function with the datasets above

Veteran example

```
In [131]: veteran.columns
```

```
Out[131]: Index(['trt', 'time', 'status', 'karno', 'diagtime', 'age', 'prior',
                 'cell_small', 'cell_squamous', 'cell_large'],
                dtype='object')
```

```
In [277]: mod_sel(veteran, 'time', 'status', ['trt', 'karno', 'diagtime', 'age', 'prior',
                  'cell_small', 'cell_squamous', 'cell_large'], LogLogisticAFTFitter())
```

```
Model Selection using <lifelines.LogLogisticAFTFitter>

Step 1

Null LL: -750.2657888574508

Step 2
```

```
Variable Fitted: trt
LL Ratio: 0.8951726826337563
p-value: 0.34407939869021964
Do not add trt

Variable Fitted: karno
LL Ratio: 60.17566345044929
p-value: 8.675829579229974e-15
Add karno

Variable Fitted: diagtime
LL Ratio: 0.7307296780541037
p-value: 0.3926467736511239
Do not add diagtime

Variable Fitted: age
LL Ratio: 0.030998327875749965
p-value: 0.8602441185388373
Do not add age

Variable Fitted: prior
LL Ratio: 0.07883513928277353
p-value: 0.7788822439283708
Do not add prior

Variable Fitted: cell_small
LL Ratio: 9.71163460038224
p-value: 0.001831050784947594
Add cell_small

Variable Fitted: cell_squamous
LL Ratio: 5.7168503690013495
p-value: 0.016802846026119825
Add cell_squamous

Variable Fitted: cell_large
LL Ratio: 8.915634053476424
p-value: 0.002827396304339703
Add cell_large

Step 2 Variables: ['karno', 'cell_small', 'cell_squamous', 'cell_large']

Step 3

Variables Fitted: ['cell_small', 'cell_squamous', 'cell_large']
LL Ratio: 54.43639510696653
p-value: 1.6056145509714005e-13
Do not drop karno
```

```
Variables Fitted: ['karno', 'cell_squamous', 'cell_large']
LL Ratio: 0.13839415621077933
p-value: 0.7098826859256566
Drop cell_small

Variables Fitted: ['karno', 'cell_small', 'cell_large']
LL Ratio: 8.358231499193835
p-value: 0.003839440549322767
Do not drop cell_squamous

Variables Fitted: ['karno', 'cell_small', 'cell_squamous']
LL Ratio: 8.00100624478955
p-value: 0.004675136210790654
Do not drop cell_large

Step 3 Variables: ['karno', 'cell_squamous', 'cell_large']

Step 4

Variable Fitted: trt
LL Ratio: 0.13243693524418632
p-value: 0.7159186021465329
Do not add trt

Variable Fitted: diagtime
LL Ratio: -1.2785393437297898e-05
p-value: 1.0
Do not add diagtime

Variable Fitted: age
LL Ratio: 0.929552519786057
p-value: 0.3349792295292947
Do not add age

Variable Fitted: prior
LL Ratio: 0.24795917926917355
p-value: 0.618515762008417
Do not add prior

Variable Fitted: cell_small
LL Ratio: 0.13839415621077933
p-value: 0.7098826859256566
Do not add cell_small

Step 4 Variables: ['karno', 'cell_squamous', 'cell_large']

Step 5
```

```
Variables Fitted: ['cell_squamous', 'cell_large']
LL Ratio: 54.41160352101019
p-value: 1.6259990828547091e-13
Do not drop karno

Variables Fitted: ['karno', 'cell_large']
LL Ratio: 10.118893895479232
p-value: 0.0014675704732795896
Do not drop cell_squamous

Variables Fitted: ['karno', 'cell_squamous']
LL Ratio: 9.676704833036183
p-value: 0.001866190244411406
Do not drop cell_large

Step 5 Variables: ['karno', 'cell_squamous', 'cell_large']

<lifelines.LogLogisticAFTFitter: fitted with 137 observations, 9 censored>
         event col = 'status'
number of subjects = 137
  number of events = 128
    log-likelihood = -712.66
  time fit was run = 2019-12-12 18:49:02 UTC

---
                      coef exp(coef)  se(coef)  coef lower 95%  coef upper 95% exp(coef) lower
alpha_ karno          0.04      1.04      0.00            0.03            0.04                1
       cell_squamous  0.73      2.07      0.22            0.29            1.16                1
       cell_large     0.70      2.01      0.22            0.27            1.13                1
       _intercept     1.77      5.86      0.27            1.24            2.30                3
beta_  _intercept     0.54      1.72      0.07            0.40            0.69                1

                        z      p  -log2(p)
alpha_ karno         8.20 <0.005     51.86
       cell_squamous 3.25 <0.005      9.74
       cell_large    3.17 <0.005      9.36
       _intercept    6.50 <0.005     33.55
beta_  _intercept    7.31 <0.005     41.82
---
Concordance = 0.74
Log-likelihood ratio test = 75.21 on 3 df, -log2(p)=51.44
```

GBSG Example

```
In [202]: mod_sel(gbsg, 'time', 'cens', ['horTh', 'age', 'menostat', 'tsize', 'pnodes', 'progre
                                          'tgrade2', 'tgrade3'], LogNormalAFTFitter())
```

```
Model Selection using <lifelines.LogNormalAFTFitter>

Step 1

Null LL: -2618.8850036762237

Step 2

Variable Fitted: horTh
LL Ratio: 9.54051028127924
p-value: 0.0020098585780372598
Add horTh

Variable Fitted: age
LL Ratio: 1.878787867376559
p-value: 0.1704719784837989
Do not add age

Variable Fitted: menostat
LL Ratio: 0.0082452167325755
p-value: 0.927648986567506
Do not add menostat

Variable Fitted: tsize
LL Ratio: 17.6513230983046
p-value: 2.6533063333114965e-05
Add tsize

Variable Fitted: pnodes
LL Ratio: 58.36177985339509
p-value: 2.1808690347481885e-14
Add pnodes

Variable Fitted: progrec
LL Ratio: 34.72811507656388
p-value: 3.791166734127728e-09
Add progrec

Variable Fitted: estrec
LL Ratio: 6.518414370294522
p-value: 0.010676314825462125
Add estrec

Variable Fitted: tgrade2
LL Ratio: 0.04086326297147025
p-value: 0.8398020890830521
Do not add tgrade2
```

```
Variable Fitted: tgrade3
LL Ratio: 13.382660833534828
p-value: 0.000253961184335871
Add tgrade3

Step 2 Variables: ['horTh', 'tsize', 'pnodes', 'progrec', 'estrec', 'tgrade3']

Step 3

Variables Fitted: ['tsize', 'pnodes', 'progrec', 'estrec', 'tgrade3']
LL Ratio: 10.822856150936786
p-value: 0.0010025471234177744
Do not drop horTh

Variables Fitted: ['horTh', 'pnodes', 'progrec', 'estrec', 'tgrade3']
LL Ratio: 3.89340212875868
p-value: 0.048476131393563306
Do not drop tsize

Variables Fitted: ['horTh', 'tsize', 'progrec', 'estrec', 'tgrade3']
LL Ratio: 39.44237857357166
p-value: 3.378839087005479e-10
Do not drop pnodes

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'estrec', 'tgrade3']
LL Ratio: 24.010384227266513
p-value: 9.5817532431097e-07
Do not drop progrec

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade3']
LL Ratio: 0.00010987278074026108
p-value: 0.9916367098002019
Drop estrec

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'progrec', 'estrec']
LL Ratio: 3.4251127355764766
p-value: 0.06421185897535465
Do not drop tgrade3

Step 3 Variables: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade3']

Step 4

Variable Fitted: age
LL Ratio: 0.5005233406764091
p-value: 0.47927026106489745
Do not add age
```

```
Variable Fitted: menostat
LL Ratio: 0.44808744553756696
p-value: 0.5032445980123434
Do not add menostat

Variable Fitted: estrec
LL Ratio: 0.00010987277983076638
p-value: 0.9916367098348151
Do not add estrec

Variable Fitted: tgrade2
LL Ratio: 9.458736580515506
p-value: 0.0021014576619850576
Add tgrade2

Step 4 Variables: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade3', 'tgrade2']

Step 5

Variables Fitted: ['tsize', 'pnodes', 'progrec', 'tgrade3', 'tgrade2']
LL Ratio: 10.69176763896121
p-value: 0.001076133331785573
Do not drop horTh

Variables Fitted: ['horTh', 'pnodes', 'progrec', 'tgrade3', 'tgrade2']
LL Ratio: 3.979641013298533
p-value: 0.04605337732044584
Do not drop tsize

Variables Fitted: ['horTh', 'tsize', 'progrec', 'tgrade3', 'tgrade2']
LL Ratio: 36.8582979127259
p-value: 1.2703426463363542e-09
Do not drop pnodes

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'tgrade3', 'tgrade2']
LL Ratio: 21.08976547612292
p-value: 4.38262626840722e-06
Do not drop progrec

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade2']
LL Ratio: 12.795808397273504
p-value: 0.00034739683477462386
Do not drop tgrade3

Variables Fitted: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade3']
LL Ratio: 9.458736580515506
p-value: 0.0021014576619850576
Do not drop tgrade2
```

```
Step 5 Variables: ['horTh', 'tsize', 'pnodes', 'progrec', 'tgrade3', 'tgrade2']

<lifelines.LogNormalAFTFitter: fitted with 686 observations, 387 censored>
         event col = 'cens'
number of subjects = 686
  number of events = 299
    log-likelihood = -2560.41
  time fit was run = 2019-12-12 00:44:35 UTC

---
                    coef exp(coef)  se(coef)  coef lower 95%  coef upper 95% exp(coef) lower 95%
mu_    horTh        0.31      1.36      0.09            0.12            0.49                1.13
       tsize       -0.01      0.99      0.00           -0.01           -0.00                0.99
       pnodes      -0.05      0.95      0.01           -0.07           -0.03                0.94
       progrec      0.00      1.00      0.00            0.00            0.00                1.00
       tgrade3     -0.65      0.52      0.18           -1.01           -0.29                0.36
       tgrade2     -0.50      0.61      0.17           -0.83           -0.17                0.44
       _intercept   8.06   3165.24      0.19            7.68            8.44             2172.87
sigma_ _intercept  -0.02      0.98      0.04           -0.10            0.07                0.90

                       z      p  -log2(p)
mu_    horTh        3.26 <0.005      9.83
       tsize       -2.00   0.05      4.46
       pnodes      -6.13 <0.005     30.06
       progrec      4.20 <0.005     15.22
       tgrade3     -3.53 <0.005     11.25
       tgrade2     -3.01 <0.005      8.57
       _intercept 41.99 <0.005       inf
sigma_ _intercept -0.37   0.71      0.50
---
Concordance = 0.69
Log-likelihood ratio test = 174.79 on 6 df, -log2(p)=114.15


In [163]: aft_aic = -2*(-2560.41) + 2*(7)
          aft_aic

Out[163]: 5134.82
```

Lung Cancer Example

```
In [179]: lung.columns

Out[179]: Index(['inst', 'time', 'status', 'age', 'sex', 'ph.ecog', 'ph.karno',
                 'pat.karno', 'meal.cal', 'wt.loss'],
                dtype='object')

In [258]: mod_sel(lung, 'time', 'status', ['age', 'sex', 'ph.karno',
                  'pat.karno', 'meal.cal', 'wt.loss', 'ecog1', 'ecog2p'], CoxPHFitter())
```

```
Model Selection using <lifelines.CoxPHFitter>

Step 1

Null LL: -508.1167985890332

Step 2

Variable Fitted: age
LL Ratio: 3.523554509065775
p-value: 0.060502560783663596
Add age

Variable Fitted: sex
LL Ratio: 6.2467749525039835
p-value: 0.012441963653928915
Add sex

Variable Fitted: ph.karno
LL Ratio: 3.2102560389666905
p-value: 0.07317803130504656
Add ph.karno

Variable Fitted: pat.karno
LL Ratio: 8.878942650720319
p-value: 0.002884785695995085
Add pat.karno

Variable Fitted: meal.cal
LL Ratio: 0.24857422438799404
p-value: 0.6180808038571679
Do not add meal.cal

Variable Fitted: wt.loss
LL Ratio: 0.0005235070301523592
p-value: 0.9817457737437024
Do not add wt.loss

Variable Fitted: ecog1
LL Ratio: 0.38219251817440636
p-value: 0.536432167283543
Do not add ecog1

Variable Fitted: ecog2p
LL Ratio: 10.989125035158168
p-value: 0.000916480678017047
Add ecog2p
```

```
Step 2 Variables: ['age', 'sex', 'ph.karno', 'pat.karno', 'ecog2p']

Step 3

Variables Fitted: ['sex', 'ph.karno', 'pat.karno', 'ecog2p']
LL Ratio: 0.614962091819848
p-value: 0.43292558866073494
Drop age

Variables Fitted: ['age', 'ph.karno', 'pat.karno', 'ecog2p']
LL Ratio: 6.3374812959175415
p-value: 0.011821262494348186
Do not drop sex

Variables Fitted: ['age', 'sex', 'pat.karno', 'ecog2p']
LL Ratio: 0.7618814852447713
p-value: 0.3827403573168954
Drop ph.karno

Variables Fitted: ['age', 'sex', 'ph.karno', 'ecog2p']
LL Ratio: 1.235515546872989
p-value: 0.26633707089709946
Drop pat.karno

Variables Fitted: ['age', 'sex', 'ph.karno', 'pat.karno']
LL Ratio: 4.189748358454153
p-value: 0.040669133593181195
Do not drop ecog2p

Step 3 Variables: ['sex', 'ecog2p']

Step 4

Variable Fitted: age
LL Ratio: 0.49989959544654994
p-value: 0.47954424239562954
Do not add age

Variable Fitted: ph.karno
LL Ratio: 0.2851488675763676
p-value: 0.593346154564752
Do not add ph.karno

Variable Fitted: pat.karno
LL Ratio: 1.0541767421016175
p-value: 0.3045471029125563
Do not add pat.karno
```

```
Variable Fitted: meal.cal
LL Ratio: 0.05846796300807
p-value: 0.8089339804013056
Do not add meal.cal

Variable Fitted: wt.loss
LL Ratio: 1.9368651593678123
p-value: 0.1640093148281325
Do not add wt.loss

Variable Fitted: ecog1
LL Ratio: 1.9581464425234572
p-value: 0.16171169189084011
Do not add ecog1

Step 4 Variables: ['sex', 'ecog2p']

Step 5

Variables Fitted: ['ecog2p']
LL Ratio: 6.6323262635413585
p-value: 0.010014439588646555
Do not drop sex

Variables Fitted: ['sex']
LL Ratio: 11.374676346195542
p-value: 0.0007445219697542842
Do not drop ecog2p

Step 5 Variables: ['sex', 'ecog2p']

<lifelines.CoxPHFitter: fitted with 167 observations, 47 censored>
      duration col = 'time'
         event col = 'status'
number of subjects = 167
  number of events = 120
partial log-likelihood = -499.31
  time fit was run = 2019-12-12 02:07:11 UTC

---
        coef exp(coef)  se(coef)  coef lower 95%  coef upper 95% exp(coef) lower 95% exp(coef)
sex    -0.49      0.61      0.20           -0.88           -0.11                 0.41
ecog2p  0.73      2.07      0.20            0.33            1.13                 1.39

           z      p  -log2(p)
sex    -2.51   0.01      6.37
ecog2p  3.57 <0.005     11.47
---
```

```
Concordance = 0.63
Log-likelihood ratio test = 17.62 on 2 df, -log2(p)=12.71
```

In [ ]: