Cox Model Assessment, Diagnostics, Extensions

Matthew Austin

Institute for Advanced Analytics
MSA Class of 2019

Model assessment:
General things you can do

Introduction

- The obvious question once you build a model: “Is it any good?”
- We’ll briefly discuss a few ways to measure a model’s predictive ability (all of which you’ve seen before in different contexts) although many of these things are a bit trickier in survival analysis
    - Spread of predicted values
    - R^2
    - Concordance

Plots and R^2

- One way to assess the model is to look at how well it separates low and high risk subjects (instead of predicting everyone to have the same risk); to do this, you can look at the standard deviation of $\hat{\boldsymbol{\eta}}$ or make a histogram
- Another approach is R^2: in survival analysis, $R^2 = 1 - e^{-LR/n}$
  - **Important**: n is the number of **subjects**
  - This can be rescaled by its maximum value (both computed by `coxph()`, but not always correctly—more about this later), and we can use it to compute the Nagelkerke R^2 from logistic regression
  - Like in logistic regression, not easily interpretable
  - Shouldn't be used to compare models (use LRT instead)

Overfitting

- Measures like concordance and R^2 look at how well the model fits the data, but predicting future outcomes for survival analysis isn't really easy or intuitive, so methods of addressing overfitting/optimism aren't very clear
- One simple way to account for overfitting is to shrink $\hat{\boldsymbol{\eta}}$ by some amount $\hat{v} = 1 - \frac{\text{model df}}{LR}$ (the *shrinkage factor*), so $\tilde{\eta}_i = \hat{v}\hat{\eta}_i$
  - This is a simple implementation of *calibration*
- Ultimately, due to the presence of censoring as well as the fact that the Cox regression model makes relative predictions, it's much more difficult to use standard tools like cross-validation or hold-out samples to estimate the absolute accuracy of survival models

Model assessment:
Concordance

Concordance in survival analysis

- Recall concordance as you've learned it previously: for all possible (event, non-event) pairs, we want to assign the higher predicted value to the subject that had the event
- With survival data, it's mostly the same idea: we hope to assign a higher risk to the subject that had the event **first**
- The idea is that we want to see how well the model ranks who will have the event *sooner*
- Example:
  - Person 1 had the event at `time = 3`, and $\hat{\eta}_1 = 1.5$
  - Person 2 had the event at `time = 7`, and $\hat{\eta}_2 = 0.3$
  - This a concordant pair because the person with the higher risk score had the event first

Concordance with censoring

- Even with censoring, the concept still works as it did before
- Comparing two people again:
  - Person 1 had the event at `time = 3`, and $\hat{\eta}_1 = 1.5$
  - Person 2 was censored at `time = 7`, and $\hat{\eta}_2 = 0.3$
  - This is still a concordant pair because the person with the higher risk score had the event first
- If both people (1) have the event at the same time, **or** (2) are censored at the same time, **or** (3) have the same predicted risk, then these pairs are incomparable, and we don't count them

Indeterminate pairs

- However, censoring does change this in a couple ways
- Once again, comparing two people:
  - Person 1 is censored at `time = 3`, and $\hat{\eta}_1 = 1.5$
  - Person 2 had the event (or is censored) at `time = 7`, and $\hat{\eta}_2 = 0.3$
- This is an **indeterminate** pair: Person 1 is predicted to have the event first, but there's no way to know whether or not that actually happened

Ordering of event times

- The second consideration with censoring is how to "order" things that happen at the same time
- The convention in survival analysis is that everything happens at the end of the time interval, so if someone is censored at `time = 5`, we assume they made it through the entirety of time 5 and are lost after that
- The consequence of this assumption is that **censoring happens after events happen**
- Comparing two people:
  - Person 1 has the event at `time = 3`, and $\hat{\eta}_1 = 1.5$
  - Person 2 is censored at `time = 3`, and $\hat{\eta}_2 = 0.3$
  - This is a concordant pair because the person with the higher risk score had the event "first"

Concordance: R and SAS syntax

- In R, the concordance( ) function will compute this for you:
  `concordance(fit)`

```
## Call:
## concordance.coxph(object = fit)
##
## n= 432
## Concordance= 0.6403 se= 0.02666
## discordant concordant     tied.x     tied.y    tied.xy
##      27242      15291         49        111          0
```

- In SAS, use the concordance option in the proc phreg statement

Diagnostics:
Residuals

Model assumptions

- In the Cox regression model, predictors are restricted to following the proportional hazards, so we need to check for violations of the PH assumption
- We also assume a linear relationship between $\mathbf{x}$ and $\log(\text{hazard})$, so we need to check this as well
- Just like in any other regression model you've seen, we're about to do a ton of plotting a bunch of different residuals to check these assumptions

Residuals in survival analysis

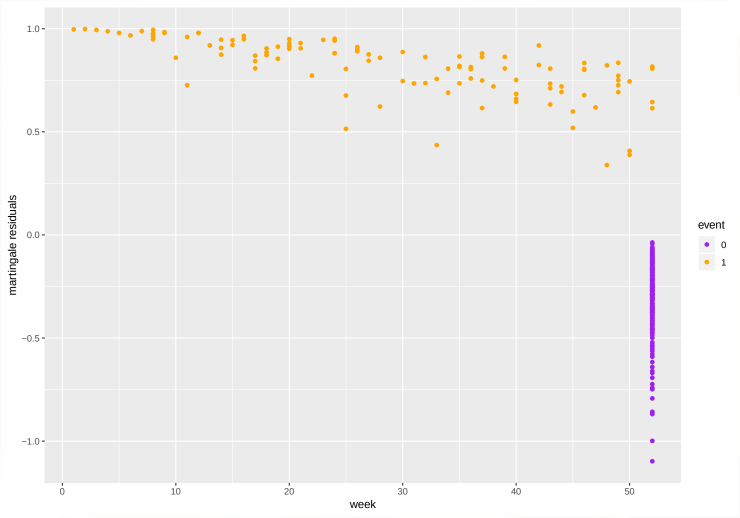
- There are four kinds of residuals for survival models, all with various uses:
  - Martingale (check linearity, detect outliers)
  - Deviance (check linearity, detect outliers)
  - Score (detect influential observations and compute robust SE)
  - Schoenfeld (check proportional hazards)
- `coxph()` and `proc phreg` can compute all of these for you
  - R: `residuals(..., type = c("martingale", "deviance", "score", "schoenfeld", "scaledsch"))`
  - SAS: `resmart=... resdev=... ressco=... ressch=... wtressch=...`

Diagnostics:
Influence & outliers

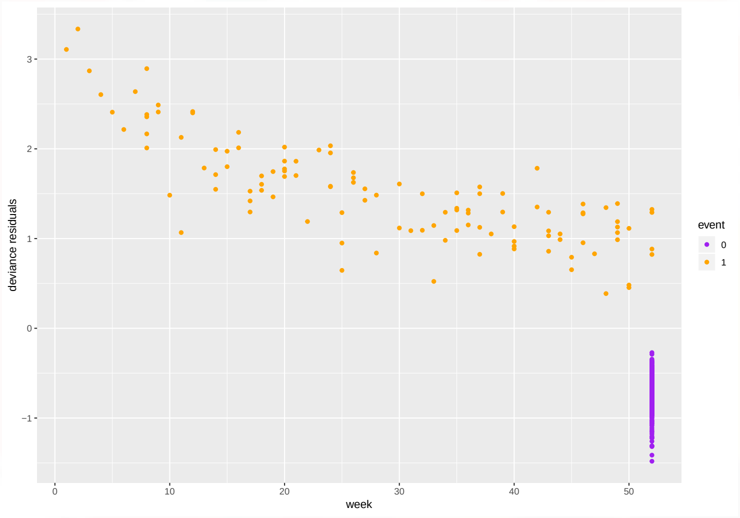
Martingale & deviance residuals

- Martingale residuals are the difference between a subject's observed and expected number of events for their entire tenure
    - Upper bound of 1 (for uncensored observations) or 0 (for censored observations); no lower bound
- Positive/negative martingale residuals indicate if the subject had the event sooner than expected (positive) or later than expected (negative)
- A plot of martingale residuals can help identify outliers
- Deviance residuals are more symmetric and sometimes preferable to martingale residuals

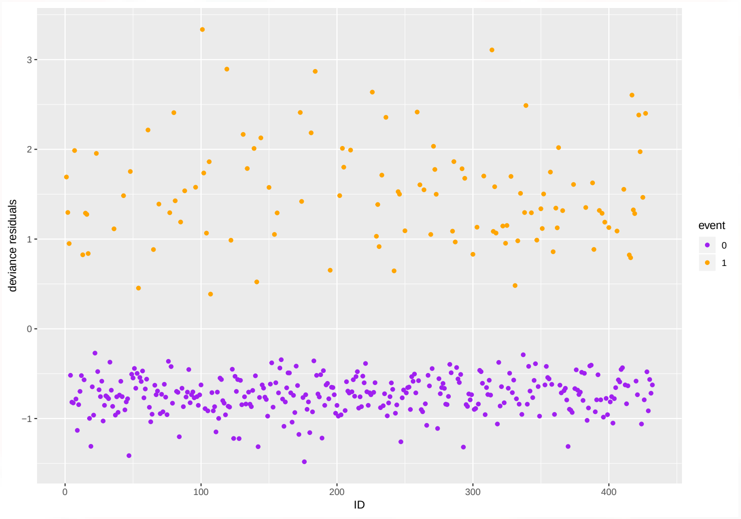
Martingale residuals vs. time



Deviance residuals vs. time



Deviance residuals by ID

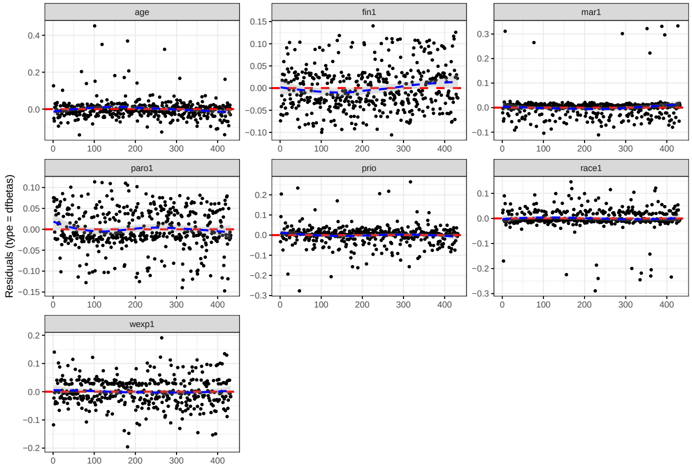


Influential observations

- As before, we'll use delta-beta (dfbeta) plots to detect influential points (remember that these show the difference in $\hat{\beta}_j$ after omitting each subject)
- In R: `residuals(..., type = "dfbetas")`
- In SAS: `dfbeta=...`

Delta-beta plots

```
ggcoxdiagnostics(fit, type = "dfbetas")
```



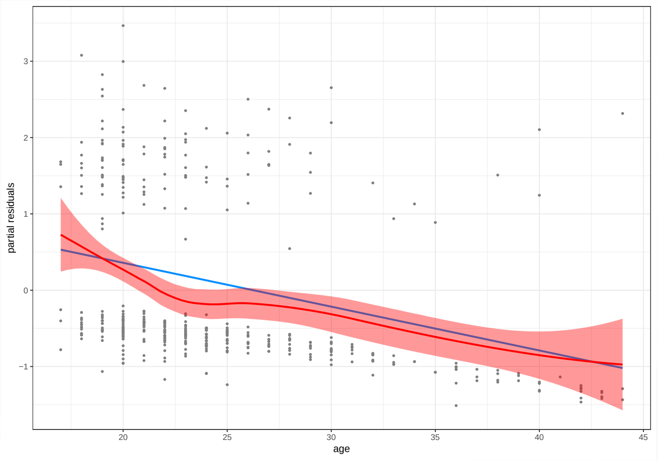
Diagnostics:
Linearity

Residual plots

- Martingale and deviance residuals are also useful for checking linearity of [continuous] predictors by plotting them vs. the predictor
    - There are varying approaches on how to do this, but they all seem to give similar results in my experience (but what do I know?)
- SAS can produce a cumulative residual plot; this process is supposed to show a random walk (starting and ending at 0) over time, so a mostly positive or negative walk in these plots indicates a violation of linearity, although I'm not sure this plot gives any indication of what the appropriate transformation is

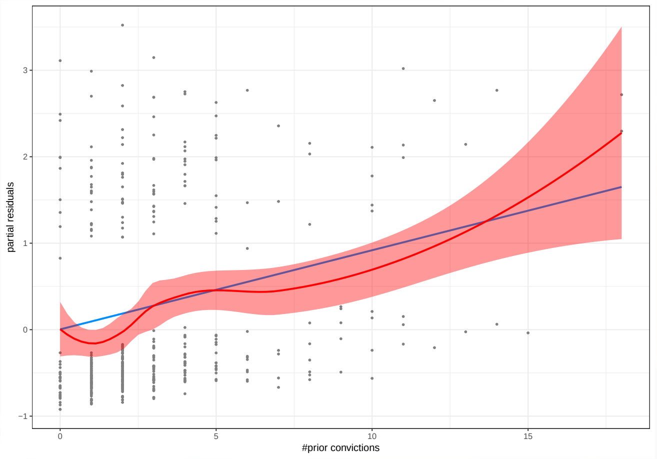
Functional form for `age`: `R` syntax

```
visreg(fit, ”age”)
```



Functional form for `prio`: R syntax

```
visreg(fit, "prio")
```



Functional form: SAS syntax

```
proc phreg data=survival.recid;
  ...
  assess var=(age prio) / resample;
run;
```

Proportional hazards & model extensions:
Stratification

Introduction

- In PH models, we assume effects are constant over time, so the hazard ratio is independent of time:

$$\frac{\lambda_1(t)}{\lambda_2(t)} = e^{(\mathbf{x}_1 - \mathbf{x}_2)^\mathsf{T} \boldsymbol{\beta}}$$

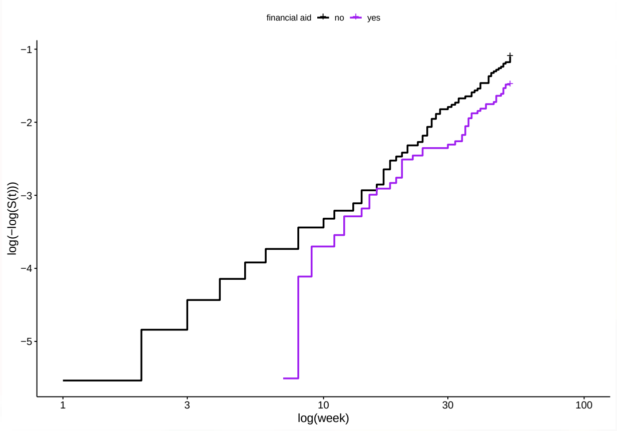
- This assumption can be violated for many reasons, but there are a number of ways to check it
- Fortunately, the Cox model can be easily extended in various ways to accomodate non-proportional hazards

Checking proportional hazards: stratification

- The simplest method of checking the proportional hazards is to ummm... plot the hazards to see if they're proportional...
- In a PH model, a plot of $\log(-\log S_i(t))$ vs. $\log t$ should have parallel lines
- This approach is best used for categorical variables with few levels
- There's no guide for how parallel is "parallel enough," and results from these plots may conflict with other checks for PH

Checking PH: R syntax

```
fit_strat <- coxph(Surv(...) ~ strata(fin) + ...)
ggsurvplot(survfit(fit_strat), data = recid, fun = ”cloglog”)
```



Checking PH: SAS syntax

- SAS can do a different check for PH using cumulative sums of Schoenfeld residuals (discussed later)
- Like the plots for linearity we saw earlier, this process is supposed to show a random walk (starting and ending at 0) over time, so a mostly positive or negative walk in these plots indicates a violation of PH

```
proc phreg data=survival.recid;
  ...
  assess ph / resample;
run;
```

The stratified Cox model

- If a predictor violates the PH assumption (in any way), the easiest solution is to just use a **stratified Cox model**—instead of including it in the model (where it's restricted to obey PH), just let each group g have its own separate baseline hazard $\lambda_{0g}(t)$:

$$\lambda_{ig}(t) = \lambda_{0g}(t)e^{\mathbf{x}_i^\top \boldsymbol{\beta}}$$

- Basically, the stratification variable is treated as categorical, and by pooling the information in each stratum, we estimate the effects $\boldsymbol{\beta}$ (which are common across strata)
- This can be extended even further by including strata×predictor interaction(s), allowing the effect(s) to differ across strata:

$$\lambda_{ig}(t) = \lambda_{0g}(t)e^{\mathbf{x}_i^\top \boldsymbol{\beta}_g}$$

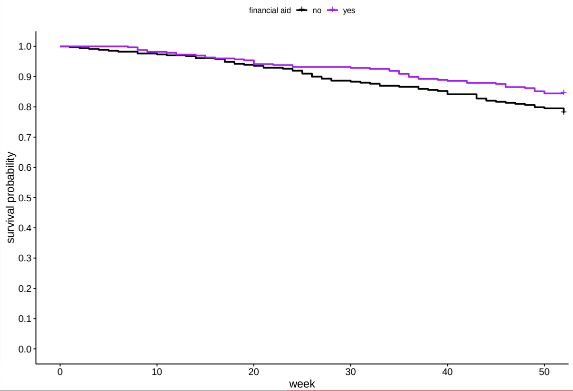
Stratification: R syntax

```
fit_strat <- coxph(Surv(...) ~ strata(fin) + ...)
```

	coef	exp(coef)	se(coef)	z	Pr(>|z|)
age	-0.06	0.94	0.02	-2.59	0.01
race1	0.31	1.36	0.31	1.00	0.32
wexp1	-0.15	0.86	0.21	-0.71	0.48
mar1	-0.43	0.65	0.38	-1.14	0.26
paro1	-0.08	0.92	0.20	-0.42	0.68
prio	0.09	1.10	0.03	3.19	0.00

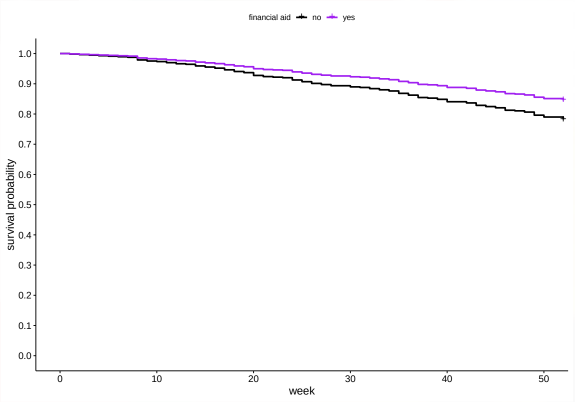
Survival curves: `fin` as strata

```
newdata1 <- data.frame(age = 30, race = "0", wexp = "0",
                       mar = "0", paro = "0", prio = 4)
ggsurvplot(survfit(fit_strat, newdata1), data = newdata1)
```



Survival curves: `fin` as predictor

```
ggsurvplot(survfit(fit, newdata2), data = newdata2)
```



Stratification with interactions: R syntax

```
fit_strat2 <- coxph(Surv(...) ~ age:strata(fin) + race +
                      wexp + mar + paro + prio,
                    data = recid)
```

	coef	exp(coef)	se(coef)	z	Pr(>|z|)
race1	0.32	1.37	0.31	1.03	0.30
wexp1	-0.16	0.85	0.21	-0.76	0.45
mar1	-0.45	0.64	0.38	-1.18	0.24
paro1	-0.08	0.92	0.20	-0.41	0.68
prio	0.09	1.10	0.03	3.30	0.00
age:strata(fin)0	-0.02	0.98	0.03	-0.85	0.40
age:strata(fin)1	-0.11	0.90	0.04	-2.88	0.00

Stratification: SAS syntax

```
proc phreg data=survival.recid;
  model week*arrest(0) = age race wexp mar paro prio / ...;
  strata fin;
run;
```

Stratification with interactions: SAS syntax

```
proc phreg data=survival.recid;
  model week*arrest(0) = age_fin0 age_fin1 race wexp mar
                         paro prio / ...;
  age_fin0 = age*(fin=0);
  age_fin1 = age*(fin=1);
  strata fin;
run;
```

Advantages and disadvantages

- The biggest advantage of stratification is never having to specify how the stratifying variable interacts with time!
  - No PH assumption (or any other assumption) is made about how the stratifying variable affects survival
- But there are a few disadvantages:
  - Since we don't specify anything about the stratifying variable, we cannot get estimates, SEs, etc. for it
  - Quickly gets complicated if stratifying variable is continuous (although you could bin it) or for multiple stratification variables
- Other considerations:
  - Stratification works best when the number of strata isn't too large compared to the total number of events
  - Be careful of strata with no events—this contributes no information, so try to avoid it when possible

Proportional hazards & model extensions:
Time-dependent coefficients

Introduction

- One alternative to stratification is allowing effects to vary with time
  - Does age have a constant effect throughout the study?
  - Is being married equally helpful/harmful over time?
- So far, all of our models assume predictors have a constant effect, β, on the response, but the effect could depend on time: $\beta(t)$
- This is called a **time-dependent coefficient**

Modeling time-dependent coefficients

$$\log \lambda_i(t) = \log \lambda_0(t) + x_i \beta(t)$$

- Here, $\beta(t)$ is a function of time; e.g., it could be a linear function:
$$\beta(t) = \beta + b \times \texttt{time}$$
- If $b = 0$, then $\beta(t) = \beta$, and the effect of x doesn't depend on time (meaning the PH assumption is satisfied)
- Otherwise, if $b \neq 0$, then $\beta(t) \neq \beta$, and the effect of x will change over time
- So instead of each observation having a constant predicted value or risk score η_i, it will be $\eta_i(t)$ recalculated at every distinct time t

Schoenfeld residuals

- Schoenfeld residuals are best used for investigating relationships with time; the reason being that, when appropriately scaled/weighted, $\beta(t) \approx \hat{\beta} + \mathbb{E}(s^{*})$
- So we can plot the scaled Schoenfeld residuals vs. [some function of] time to see if they have random scatter around 0, as you're used to doing
- There's also another familiar option: testing the correlation between these residuals and [some function of] time
- Both of these options are available via ZPH plots and tests in `R` and `SAS`

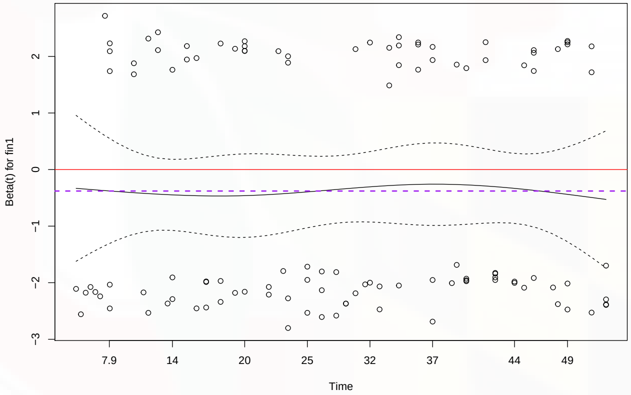
ZPH tests: Kaplan-Meier transformation

```
fit_zph <- cox.zph(fit, transform = "km")
```

```
##                 rho    chisq        p
## fin1    0.00646  0.00502 0.943519
## age    -0.26455 11.27897 0.000784
## race1  -0.11224  1.41652 0.233977
## wexp1   0.22976  7.14021 0.007537
## mar1    0.07295  0.68627 0.407435
## paro1  -0.03618  0.15496 0.693841
## prio   -0.01366  0.02304 0.879353
## GLOBAL       NA 17.65862 0.013609
```

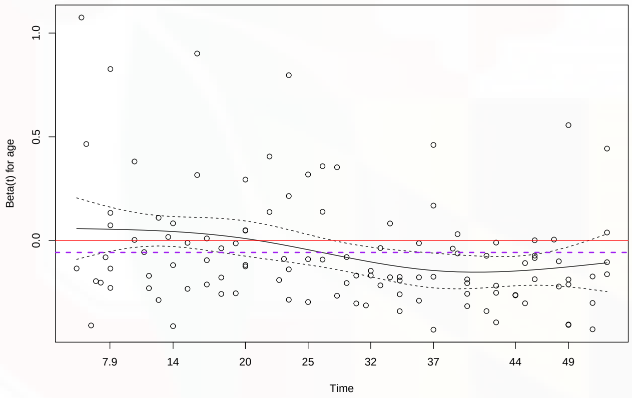
ZPH plots: Kaplan-Meier transformation

```
plot(fit_zph, var = "fin1")
```



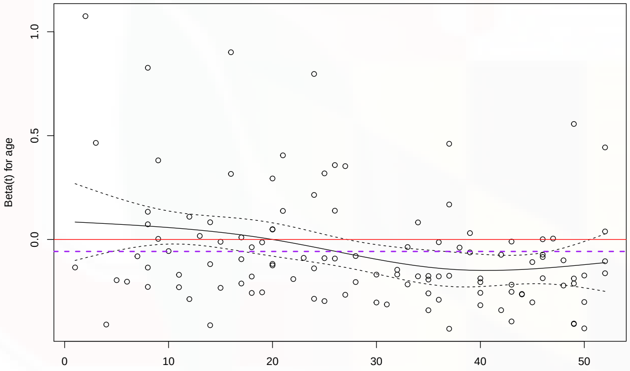
ZPH plots: Kaplan-Meier transformation

```r
plot(fit_zph, var = "age")
```



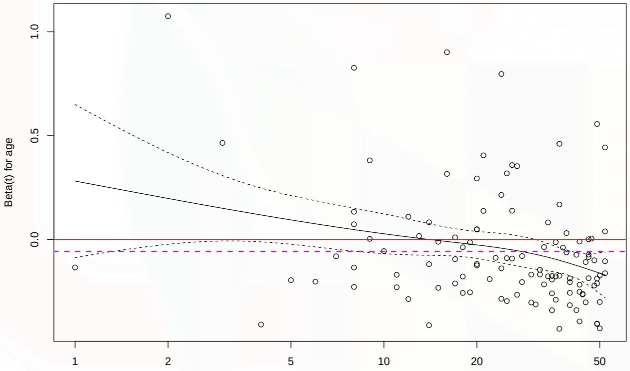
ZPH plots: no transformation

```r
fit_zph_i <- cox.zph(fit, transform = "identity")
plot(fit_zph_i, var = "age")
```



ZPH plots: log transformation

```r
fit_zph_log <- cox.zph(fit, transform = "log")
plot(fit_zph_log, var = "age")
```



ZPH plots: SAS syntax

```
proc phreg data=survival.recid zph(global transform=km
                                              fit=loess);
  ...
run;
```

Time transformations

- There are a few commonly used choices for $\beta(t)$: step functions, t, $\log t$, rank(t), Kaplan-Meier estimate
- This may seem like a standard interaction with time, but the important thing to remember is that $\eta_i(t)$ is updated at each time, so the "interaction" must be constructed in a way that does so
- Creating an $x \times$ `time` variable as you're used to doing **is incorrect**: this produces just one score $\eta_i = x_i \times \texttt{time} \times \beta$ that only counts the effect at the single `time` recorded for subject i
    - `coxph()` actually prints a warning if you try it this way!
    - `proc phreg` implements this correctly as long as you define the interaction within `proc phreg`

Time-dependent coefficients: R syntax

```
fit_tdc <- coxph(Surv(...) ~ ... + age + tt(age), data = recid,
                 tt = function(x, time, ...){x*log(time)})
```

	coef	exp(coef)	se(coef)	z	Pr(>|z|)
fin1	-0.38	0.68	0.19	-1.98	0.05
race1	0.32	1.38	0.31	1.04	0.30
wexp1	-0.13	0.88	0.21	-0.59	0.55
mar1	-0.41	0.66	0.38	-1.08	0.28
paro1	-0.09	0.91	0.20	-0.47	0.64
prio	0.09	1.10	0.03	3.26	0.00
age	0.12	1.13	0.07	1.86	0.06
tt(age)	-0.06	0.94	0.02	-2.73	0.01

The "coefficient" for age is now $\beta_{\texttt{age}} = 0.12 - 0.06(t)$, where t is the current time

Time-dependent coefficients: SAS syntax

```
proc phreg data=survival.recid;
  model week*arrest(0) = ... agelogweek / ...;
  agelogweek = age*log(week);
run;
```

Proportional hazards & model extensions:
Time-dependent variables

Introduction

- A similar idea to time-dependent coefficients is a **time-dependent variable**, where the actual **value** of the variable (rather than its effect) changes over time
- Time-*independent* variables: age (at entry), height, race, etc.
- Time-*dependent* variables: employment status, blood pressure, etc.

Modeling time-dependent variables

$$\log \lambda_i(t) = \log \lambda_0(t) + x_i(t)\beta$$

- Now the predictor itself depends on time, but its effect does not
- So once again, we will have $\eta_i(t)$ recalculated/updated at the times where the value of x_i changes
- Example: employment status can change each week—employed/unemployed

Coding time-dependent variables

- The most important thing to remember with time-dependent variables: **future data cannot be used to predict the past**
- That seems obvious, but this mistake is frequent enough to have a name: "immortal time bias"
- The way to avoid this is to actually recognize when this can occur and to structure the data appropriately, and then we can just use the same fitting procedures we've already learned

The counting process structure

- For time-dependent variables, it is necessary to split the `time` column of your dataset into separate `(start` and `stop]` columns
- This is known as the **counting process** structure/layout
- Essentially, we're moving from "wide" to "long" formatting

Counting process example

- A subject has an event at `time = 9`. If the value of x_i changes after `time = 5`, then we observe subject i until the end of time 5, at which point they are censored:

ID	(start	stop]	x	event
1	0	5	3	0

- Then, we create a "new" subject starting after `time = 5` who is the *exact same* as subject i but with x_i replaced by its new value. We observe this "new" subject until either x_i changes again or their tenure ends (whichever comes first):

ID	(start	stop]	x	event
1	0	5	3	0
1	5	9	7	1

Baseball example

- For our MLB playoff study, our original dataset looks like this:

team	games	event
COL	4	1
ATL	4	1
MIL	3	0
...	...	...

- ...and we want this:

team	start	stop	day	event
COL	0	1	0	0
COL	1	2	1	0
COL	2	3	1	0
COL	3	4	1	1
ATL	0	1	0	0
ATL	1	2	0	0
...	...	...	...	...

Fitting the model

- The most difficult part of modeling time-dependent variables is formatting the data correctly; once that's done, there's little difference from what you've seen before
  - In R: `Surv(time, event)` $\longrightarrow$ `Surv(start, stop, event)`
  - In SAS: `model time*event()` $\longrightarrow$ `model (start,stop)*event()`
- Estimates and inference aren't affected, but since we have a different "sample size," things like c or R^2 might be

“Four steps to effective use of the counting process”

From Therneau and Grambsch (2000), ch. 3.7:

1. “Think through the problem. ... Create appropriate dummy variables for interactions, choose the time scale, and so on.”
2. “Create the (start, stop] data set. This is usually tedious but straightforward.”
3. “Check the data set for sanity.”
    - “PRINT OUT some or all of the cases.”
    - “Read the printout carefully.”
4. “Fit the model (trivial), and think again about what the results appear to be saying.”

Time-dependent variables: `R` syntax

- We are interested in the effect of employment on recidivism
- The file `recid_long.csv` is the `recid` data formatted in the counting process with an additional variable `employed` indicating weekly employment status
- I created `employed` (and the corresponding `(start, stop]` columns) using the information from the `emp1--emp52` variables of the `recid` data set

Time-dependent variables: R syntax (con't)

```
fit_long <- coxph(Surv(start, stop, arrested == 1) ~ ... +
                  employed, data = recid_long)
```

	coef	exp(coef)	se(coef)	z	Pr(>|z|)
fin1	-0.36	0.70	0.19	-1.87	0.06
age	-0.05	0.95	0.02	-2.13	0.03
race1	0.34	1.40	0.31	1.09	0.27
wexp1	-0.03	0.97	0.21	-0.12	0.90
mar1	-0.29	0.75	0.38	-0.77	0.44
paro1	-0.06	0.94	0.19	-0.33	0.74
prio	0.09	1.09	0.03	2.94	0.00
employed	-1.33	0.26	0.25	-5.30	0.00

- The interpretation of a time-dependent variable doesn't change: the recidivism rate for employed men is $\beta_{\texttt{employed}} = e^{-1.33} = 0.26$ times that for unemployed men
- However, something still isn't quite right…

Time intervals for time-dependent variables

- Remember that in survival analysis, the convention is that things happen at the end of the time period, so if `emp4 = 1`, we assume they're employed at the end of the interval `(3, 4]`
- But if someone was arrested before the end of the week, maintaining employment throughout the week is unlikely
- Thus, by using the current week's employment status, our estimate is a bit confounded: does employment predict recidivism, or does recidivism predict employment?
- So let's try something else: the file `recid_lag.csv` is the same as `recid_long`, except the variable `employed` now indicates employment status during the **previous** week

Time-dependent variables: R syntax (con't)

```
fit_lag <- coxph(Surv(start, stop, arrested == 1) ~ ... +
                  employed, data = recid_lag)
```

	coef	exp(coef)	se(coef)	z	Pr(>|z|)
fin1	-0.35	0.70	0.19	-1.83	0.07
age	-0.05	0.95	0.02	-2.27	0.02
race1	0.32	1.38	0.31	1.04	0.30
wexp1	-0.05	0.95	0.21	-0.22	0.82
mar1	-0.34	0.71	0.38	-0.90	0.37
paro1	-0.05	0.95	0.20	-0.24	0.81
prio	0.09	1.10	0.03	3.19	0.00
employed	-0.79	0.46	0.22	-3.61	0.00

- Using the lagged version of `employed`, employment status is far less protective ($\beta_{\texttt{employed}} = e^{-0.79} = 0.46$) than previously thought, although the effect still seems substantial

Time-dependent variables: SAS syntax

```
proc phreg data=survival.recid;
  model week*arrest(0) = ... employed / ...;
  array emp(*) emp1-emp52;
  employed = emp[week];
run;

proc phreg data=survival.recid;
  where week>1;
  model week*arrest(0) = ... employed / ...;
  array emp(*) emp1-emp52;
  employed = emp[week-1];
run;
```

Model assessment with time-dependent variables

- In `proc phreg`, neither the `model (start,stop)*event(0)` syntax nor programming statements will compute the concordance, even with the `concordance` option specified
- In `R`, `concordance()` works correctly, but the R^2 returned by `coxph()` **is incorrect** because it thinks n is now the "sample size" that we've artificially increased by using the counting process data
  - (I'm almost done making a function that should do it correctly, which you are free to use at your own risk)
- When creating survival curves with time-dependent variables, your reference data set **must** contain an ID column which needs to be passed to `survfit()` using the `id` argument: `survfit(..., newdata, id = ...)`