Survival and longitudinal data analysis

Lab 4

Etienne Gaucher

08/11/2021

Introduction

Le but de ce TP est d'utiliser les outils de l'analyse de survie pour développer un outil de maintenance prédictive pour une machine. "La maintenance prévisionnelle (aussi appelée « maintenance prédictive », ou encore « maintenance anticipée ») est une maintenance conditionnelle basée sur l'anticipation [...] qui permet de donner l'état de dégradation du bien avant sa détérioration complète." (voir https://fr.wikipedia.org/wiki/Maintenance_prévisionnelle).

L'intérêt de la maintenance prédictive est donc de pouvoir faire des réparations avant la panne, qui pourrait endommager l'équipement. Cela permet aussi d'éviter les accidents qui pourraient être dus à la panne, un autre but est d'éviter les contrôles trop rapprochés qui coûtent chers à l'entreprise. Cela est rendu possible par le développement de l'internet of things (IoT), qui permet de collecter facilement et à moindre coût des données enregistrées par des capteurs placés dans ou à proximité de la machine.

Les données contiennent des mesures sur 1 000 machines, l'évènement d'intérêt est la panne. On veut prédire avec précision l'évènement `panne dans les 2 semaines` afin d'alerter à l'avance l'équipe de maintenance. Ce sont des données d'entraînement où on suppose que les valeurs enregistrées par les capteurs ne varient pas avec le temps. En situation réelle, ces données dépendraient évidemment du temps.

```
# import des librairies
library(tidyverse, quietly = T)
library(survival, quietly = T)
library(corrplot, quietly =T)
library(ggfortify, quietly =T)
library(caret, quietly=T)
library(riskRegression, quietly =T)
library(randomForestSRC, quietly =T)
source("./utils.R")
```

Import des données

On importe les données `predictive_maintenance.csv` qui contiennent les colonnes :

Nom	Type	Description
lifetime	numérique	Nombre de semaines de fonctionnement de la machine
broken	numérique	Indicatrice de censure : 1 si la machine est en panne, 0 sinon
pressureInd	numérique	Mesure de pression dans les tuyaux
moistureInd	numérique	Mesure de l'humidité ambiante
temperatureInd	numérique	Mesure de la température ambiante
team	catégorique	Nom de l'équipe utilisant la machine

Nom	Type	Description
provider	catégorique	Nom du constructeur

```
# import des données
raw_dataset = read_csv("./predictive_maintenance.csv")
```

```
## New names:
## * `` -> ...1
```

```
## Rows: 1000 Columns: 8
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): team, provider
## dbl (6): ...1, lifetime, broken, pressureInd, moistureInd, temperatureInd
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(raw_dataset)
```

```
## Rows: 1,000
## Columns: 8
## $ ...1           <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ lifetime       <dbl> 56, 81, 60, 86, 34, 30, 68, 65, 23, 81, 38, 29, 65, 65,~
## $ broken         <dbl> 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1~
## $ pressureInd    <dbl> 92.17885, 72.07594, 96.27225, 94.40646, 97.75290, 87.67~
## $ moistureInd    <dbl> 104.23020, 103.06570, 77.80138, 108.49361, 99.41349, 11~
## $ temperatureInd <dbl> 96.51716, 87.27106, 112.19617, 72.02537, 103.75627, 89.~
## $ team           <chr> "TeamA", "TeamC", "TeamA", "TeamC", "TeamB", "TeamA", "~
## $ provider       <chr> "Provider4", "Provider4", "Provider1", "Provider2", "Pr~
```

```
# recodage de la variable id
raw_dataset = raw_dataset %>% rename(id = "...1" )
glimpse(raw_dataset)
```

```
## Rows: 1,000
## Columns: 8
## $ id             <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ lifetime       <dbl> 56, 81, 60, 86, 34, 30, 68, 65, 23, 81, 38, 29, 65, 65,~
## $ broken         <dbl> 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1~
## $ pressureInd    <dbl> 92.17885, 72.07594, 96.27225, 94.40646, 97.75290, 87.67~
## $ moistureInd    <dbl> 104.23020, 103.06570, 77.80138, 108.49361, 99.41349, 11~
## $ temperatureInd <dbl> 96.51716, 87.27106, 112.19617, 72.02537, 103.75627, 89.~
## $ team           <chr> "TeamA", "TeamC", "TeamA", "TeamC", "TeamB", "TeamA", "~
## $ provider       <chr> "Provider4", "Provider4", "Provider1", "Provider2", "Pr~
```

Les variables *provider* et *team* doivent être recodées pour qu'elles soient considérées en tant que *factor* et non *character*. Les autres variables sont bien numériques.

```
# recodage de provider et team
raw_dataset = raw_dataset %>% mutate(team = factor(team),
                                     provider = factor(provider))
glimpse(raw_dataset)
```

```
## Rows: 1,000
```

```
## Columns: 8
## $ id             <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ lifetime       <dbl> 56, 81, 60, 86, 34, 30, 68, 65, 23, 81, 38, 29, 65, 65,~
## $ broken         <dbl> 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1~
## $ pressureInd    <dbl> 92.17885, 72.07594, 96.27225, 94.40646, 97.75290, 87.67~
## $ moistureInd    <dbl> 104.23020, 103.06570, 77.80138, 108.49361, 99.41349, 11~
## $ temperatureInd <dbl> 96.51716, 87.27106, 112.19617, 72.02537, 103.75627, 89.~
## $ team           <fct> TeamA, TeamC, TeamA, TeamC, TeamB, TeamA, TeamB, TeamB,~
## $ provider       <fct> Provider4, Provider4, Provider1, Provider2, Provider1, ~
```

Vérification des données

On doit ensuite vérifier si le dataset contient des valeurs NA ou des lignes dupliquées.

```
# vérification de présence de NA
anyNA(raw_dataset)
```

```
## [1] FALSE
```

Aucune valeur NA n'est présente dans le dataset.

```
# vérification de lignes dupliquées
anyDuplicated(raw_dataset)
```

```
## [1] 0
```

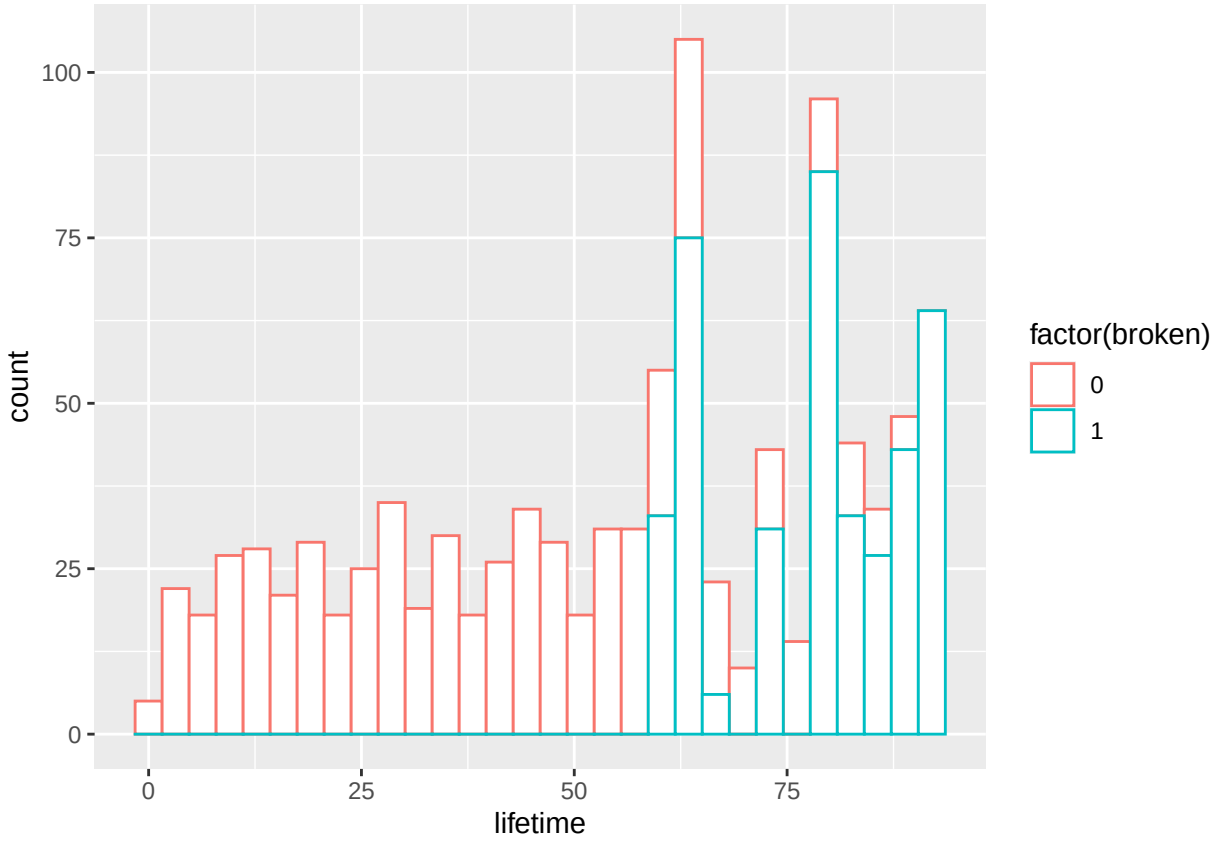
Aucune ligne dupliquée n'est présente dans le dataset.

Représentations graphiques

Histogramme de `lifetime` suivant la valeur de `broken`

```
ggplot(data = raw_dataset, aes(x = lifetime, col = factor(broken))) +
  geom_histogram(fill = 'white', alpha = 1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



On calcule ensuite le pourcentage de censure dans le jeu de données, c'est-à-dire le pourcentage d'observations avec $broken = 0$.

```
# pourcentage de censure
sum(raw_dataset$broken == 0)/nrow(raw_dataset)
```

```
## [1] 0.603
```

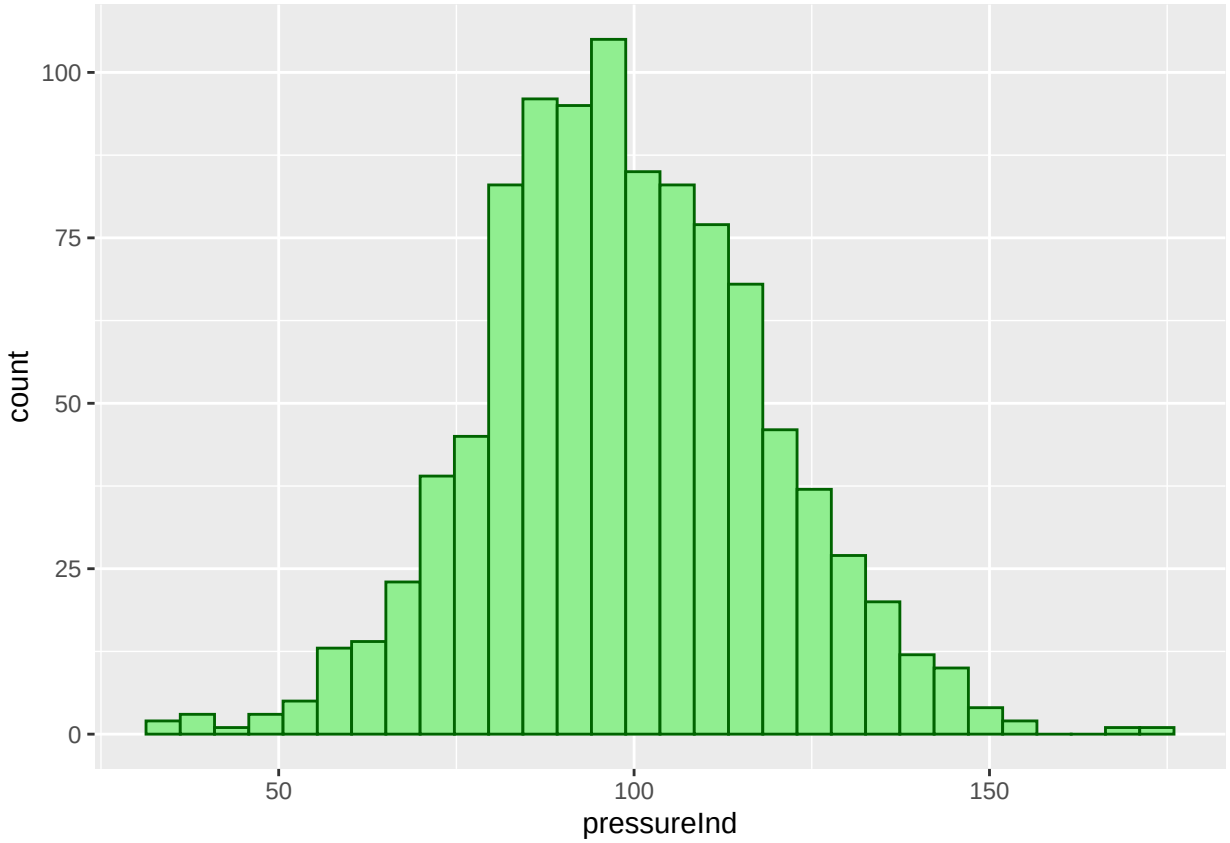
On peut voir sur l'histogramme que l'on a observé des pannes qu'à partir de 60 semaines de fonctionnement. Aucune panne n'a été détectée avant 60 semaines de fonctionnement.

Histogrammes pour les features continues

On trace les histogrammes des covariables continues (*pressureInd*, *moistureInd*, *temperatureInd*).

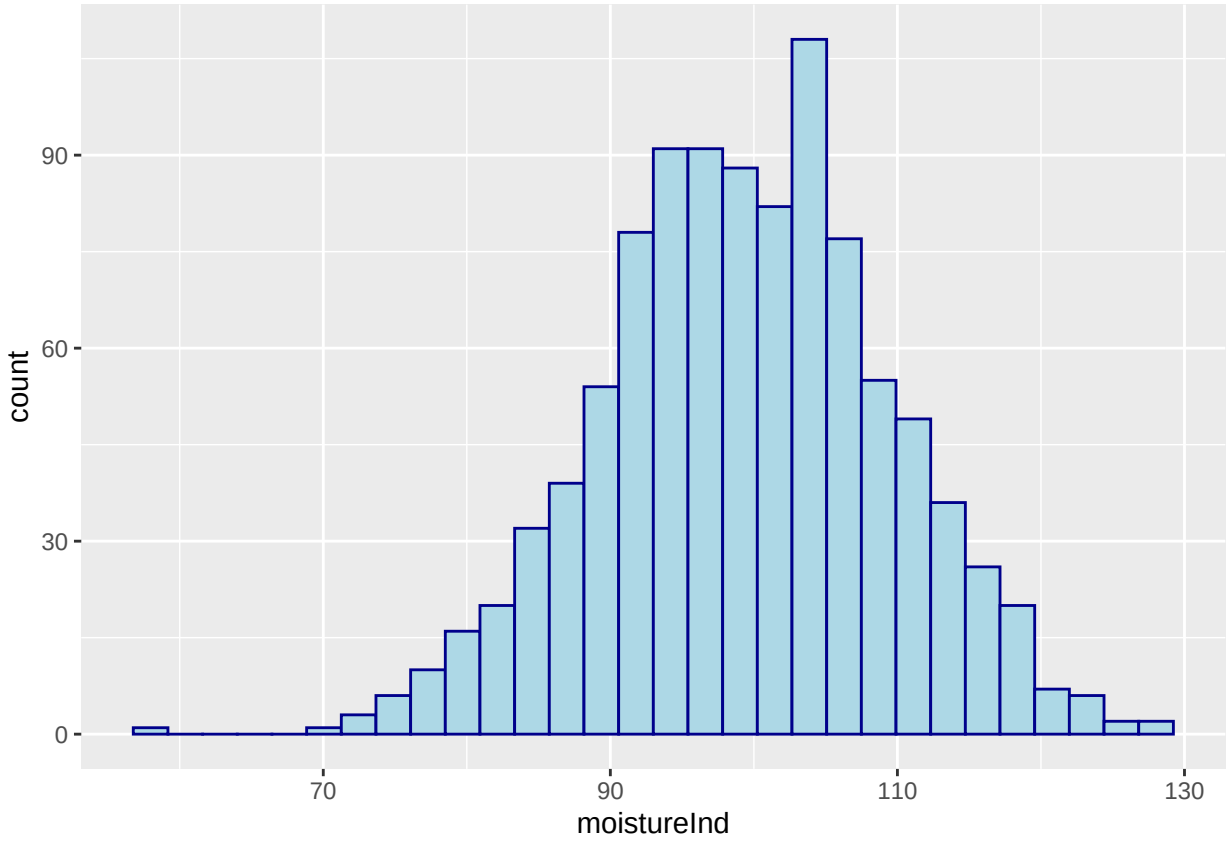
```
# Histogramme de pressureInd
ggplot(data = raw_dataset, aes(x = pressureInd)) +
  geom_histogram(color = "darkgreen", fill = "lightgreen")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



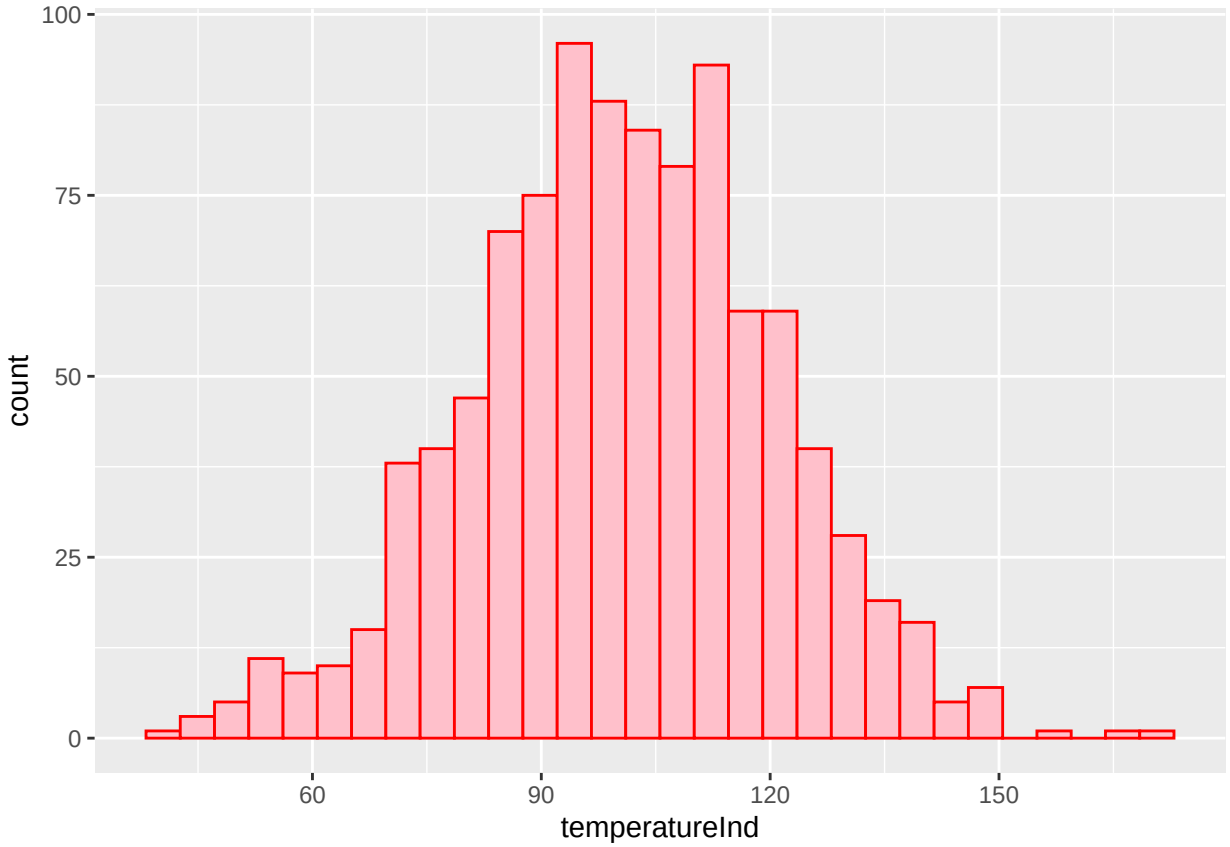

```
# Histogramme de moistureInd
ggplot(data = raw_dataset, aes(x = moistureInd)) +
  geom_histogram(color = "darkblue", fill = "lightblue")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Histogramme de temperatureInd
ggplot(data = raw_dataset, aes(x = temperatureInd)) +
  geom_histogram(color = "red", fill = "pink")
```

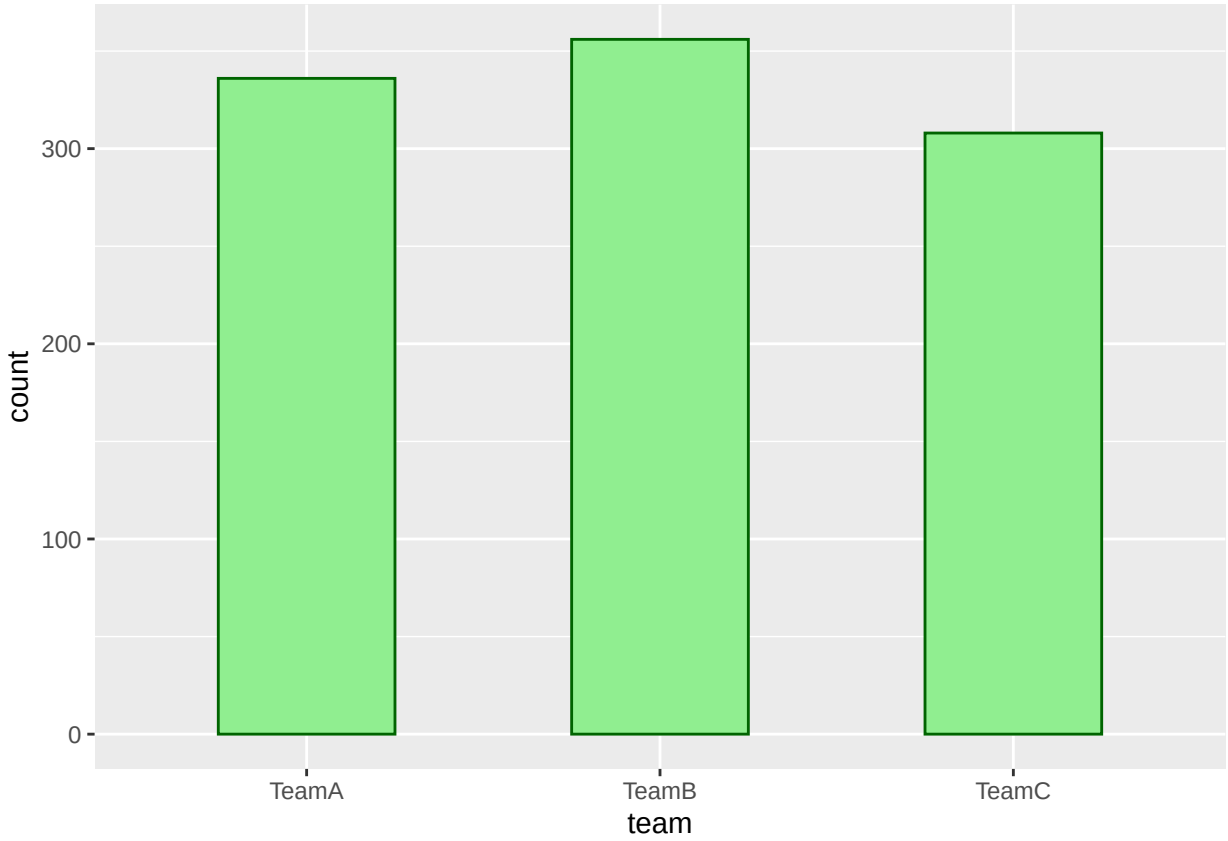
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Diagrammes en bâton pour les features discrètes

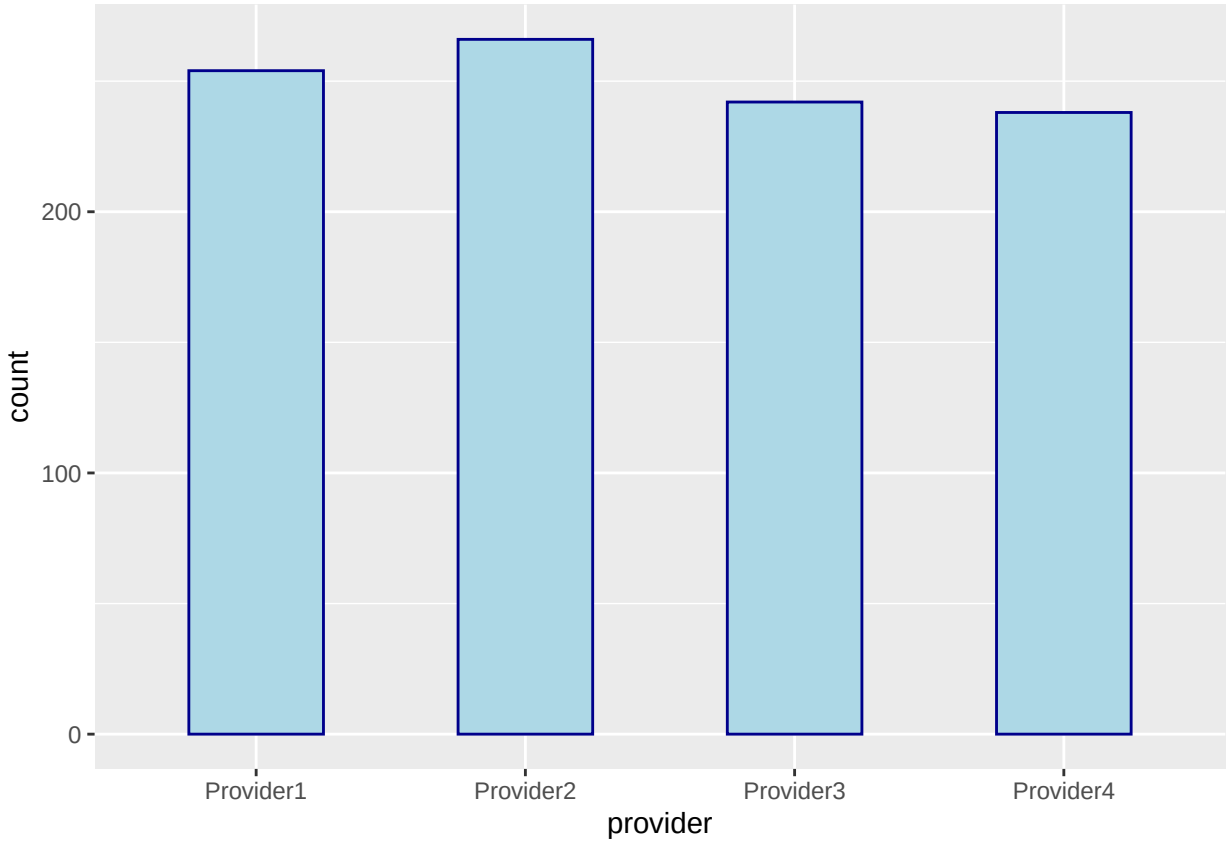
On trace les diagrammes en bâton des covariables discrètes (*team*, *provider*).

```
# Histogramme de team
ggplot(data = raw_dataset, aes(x = team)) +
  geom_bar(width = 0.5, color = "darkgreen", fill = "lightgreen")
```



Les 3 équipes sont présentes dans les mêmes proportions dans le dataset.

```
# Histogramme de provider
ggplot(data = raw_dataset, aes(x = provider)) +
  geom_bar(width = 0.5, color = "darkblue", fill = "lightblue")
```



Les 4 constructeurs sont présents dans les mêmes proportions dans le dataset.

Corrélation

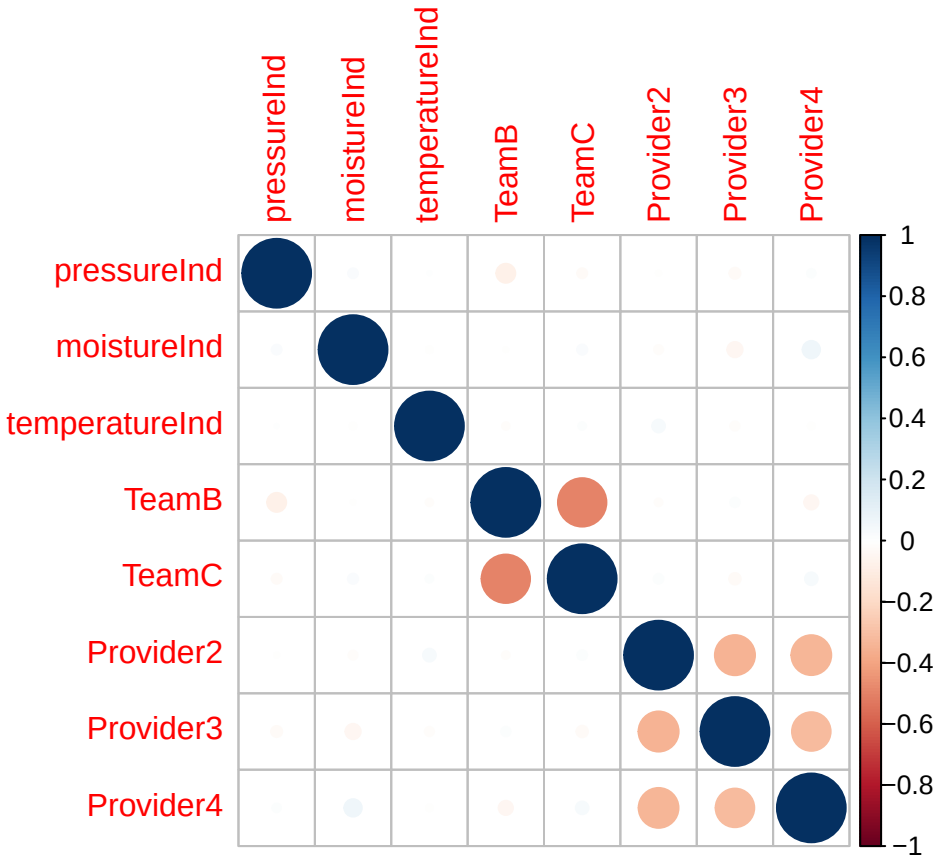
On commence par recoder les features discrètes avec le one_hot encoding.

```
one_hot_dataset = raw_dataset %>% mutate(value = 1) %>% spread(team, value, fill = 0)  %>%
                                  mutate(value = 1) %>% spread(provider, value, fill = 0) %>%
                                  select(-TeamA,-Provider1)
glimpse(one_hot_dataset)
```

```
## Rows: 1,000
## Columns: 11
## $ id             <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ lifetime       <dbl> 56, 81, 60, 86, 34, 30, 68, 65, 23, 81, 38, 29, 65, 65,~
## $ broken         <dbl> 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1~
## $ pressureInd    <dbl> 92.17885, 72.07594, 96.27225, 94.40646, 97.75290, 87.67~
## $ moistureInd    <dbl> 104.23020, 103.06570, 77.80138, 108.49361, 99.41349, 11~
## $ temperatureInd <dbl> 96.51716, 87.27106, 112.19617, 72.02537, 103.75627, 89.~
## $ TeamB          <dbl> 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1~
## $ TeamC          <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0~
## $ Provider2      <dbl> 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1~
## $ Provider3      <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0~
## $ Provider4      <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0~
```

On regarde maintenant les corrélations entre ces variables.

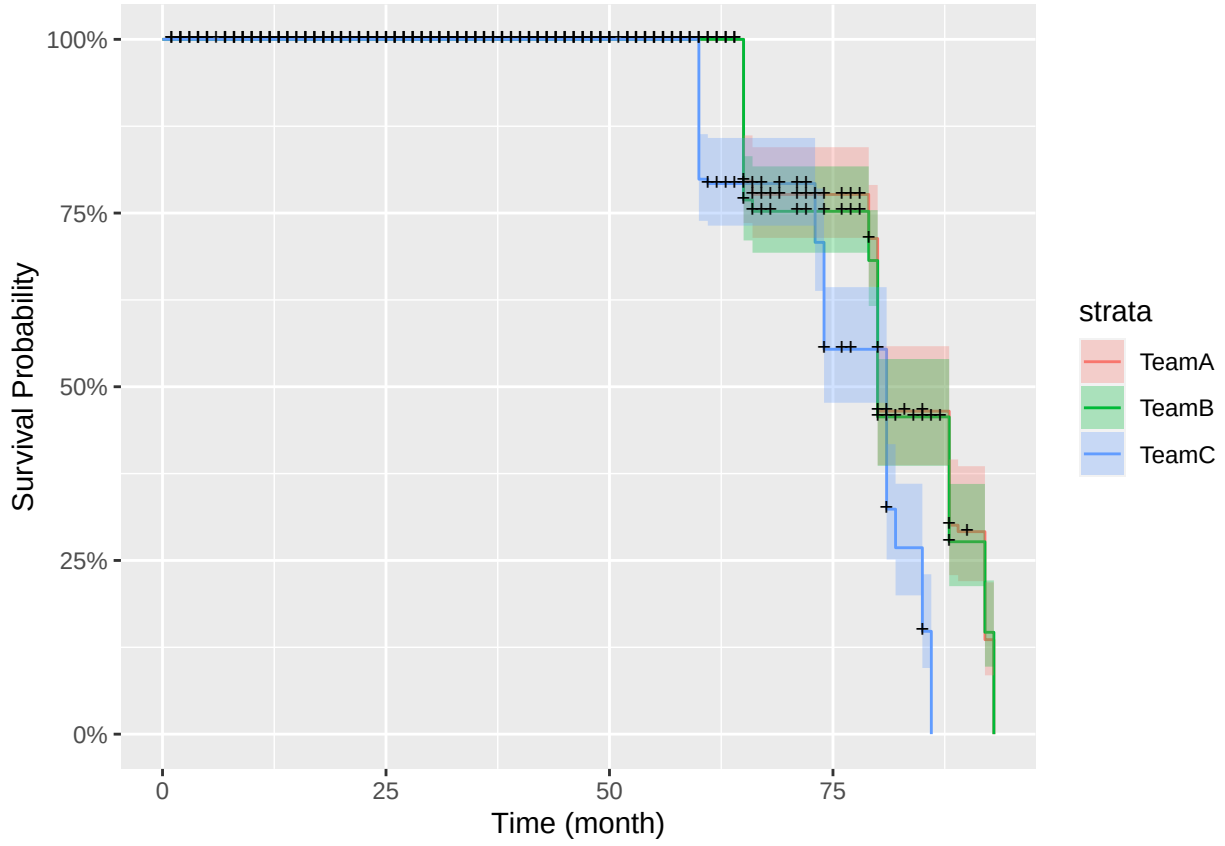
```
correlations = cor(one_hot_dataset[,-c(1:3)])
corrplot(correlations)
```



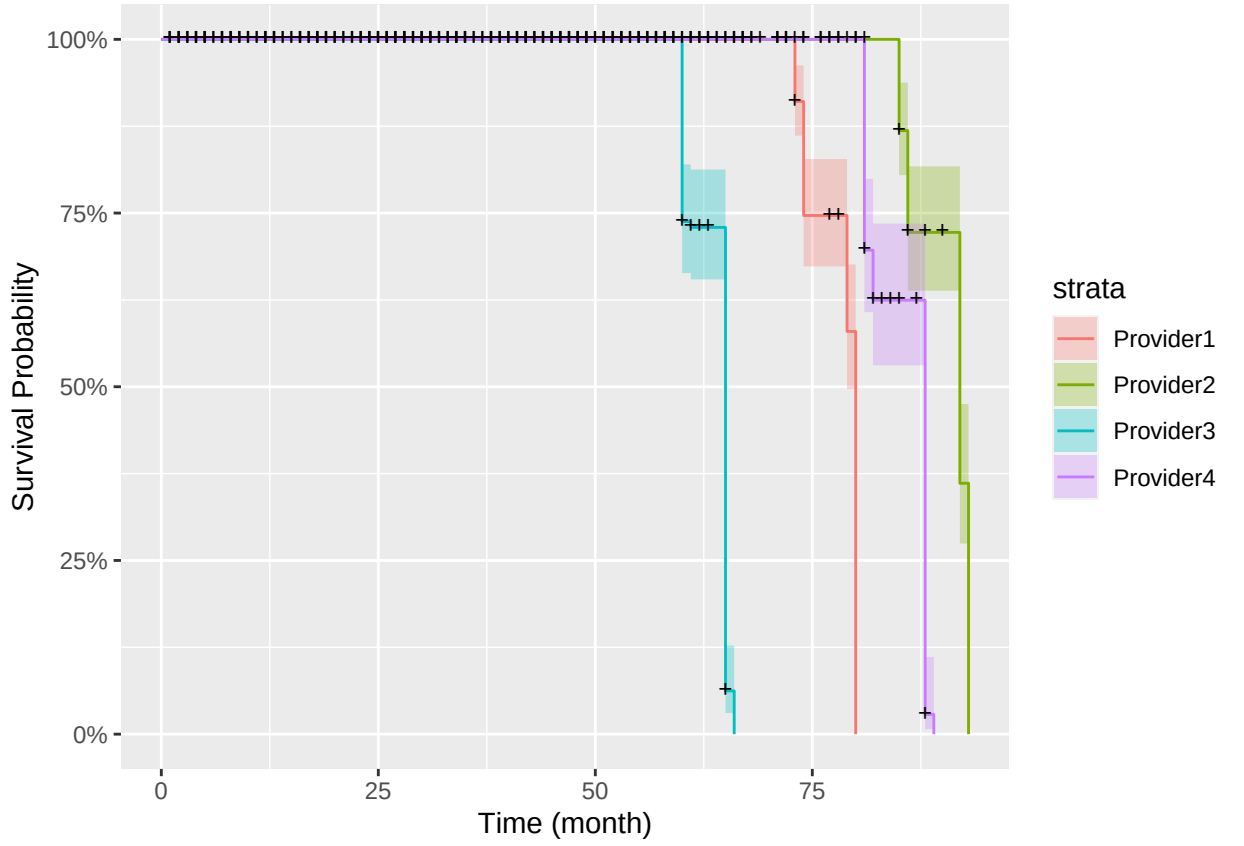
Il n'y manifestement pas de corrélations inquiétantes entre les features.

Représentations graphiques des fonctions de survie en fonction des variables discrètes

```
# Fonction de survie en fonction de l'équipe
autoplot(survfit(Surv(lifetime, broken) ~ team, data = raw_dataset),
         xlab = "Time (month)", ylab = "Survival Probability")
```




```
# Fonction de survie en fonction du constructeur
autoplot(survfit(Surv(lifetime, broken) ~ provider, data = raw_dataset),
         xlab = "Time (month)", ylab = "Survival Probability")
```



On remarque que les fonctions de survie des différentes équipes sont assez proches. En revanche, les fonctions de survie des 4 constructeurs sont extrêmement différentes. Par exemple, l'estimateur de la fonction de survie pour les machines du constructeur 3 vaut 0 après 65 semaines, alors qu'il vaut encore 1 pour tous les autres constructeurs.

Partition

On commence par séparer le jeu de données en 2 jeux de données, `train` et `test`, pour pouvoir comparer nos différents modèles. On stratifie sur la variable de censure *broken*.

```
# partition du dataset
set.seed(123)
trainIndex <- createDataPartition(raw_dataset$broken,
                                  p = 0.8,
                                  list = FALSE,
                                  times = 1)

train <- raw_dataset[ trainIndex,]
test <- raw_dataset[-trainIndex,]
```

On vérifie le pourcentage de censure dans les 2 jeux de données.

```
sum(train$broken == 0)/nrow(train)
```

```
## [1] 0.5925
```

```
sum(test$broken == 0)/nrow(test)
```

```
## [1] 0.645
```

Les 2 jeux de données ont quasiment le même pourcentage de censure, environ 60 %, ce qui correspond au pourcentage de censure que l'on a précédemment calculé pour le jeu de données complet.

Apprentissage

Modèle de Cox simple : benchmark

Modèle

```
# modèle de Cox avec toutes les covariables
cox_train_provider <- coxph(Surv(lifetime, broken) ~ .-id, data = train)
```

```
## Warning in coxph.fit(X, Y, istrat, offset, init, control, weights = weights, :
## Ran out of iterations and did not converge
```

On ne sélectionnera pas *provider* dans le modèle final car la régression ne fonctionne pas si on l'inclut, puisque les différents états de *provider* ont des fonctions de survie trop éloignées. C'est la conclusion que l'on a faite lors de l'estimation des fonctions de survie.

```
# modèle de Cox sans provider
cox_simple = coxph(Surv(lifetime, broken) ~ .-id - provider,
                   data = train,
                   x = TRUE, y = TRUE)
summary(cox_simple)
```

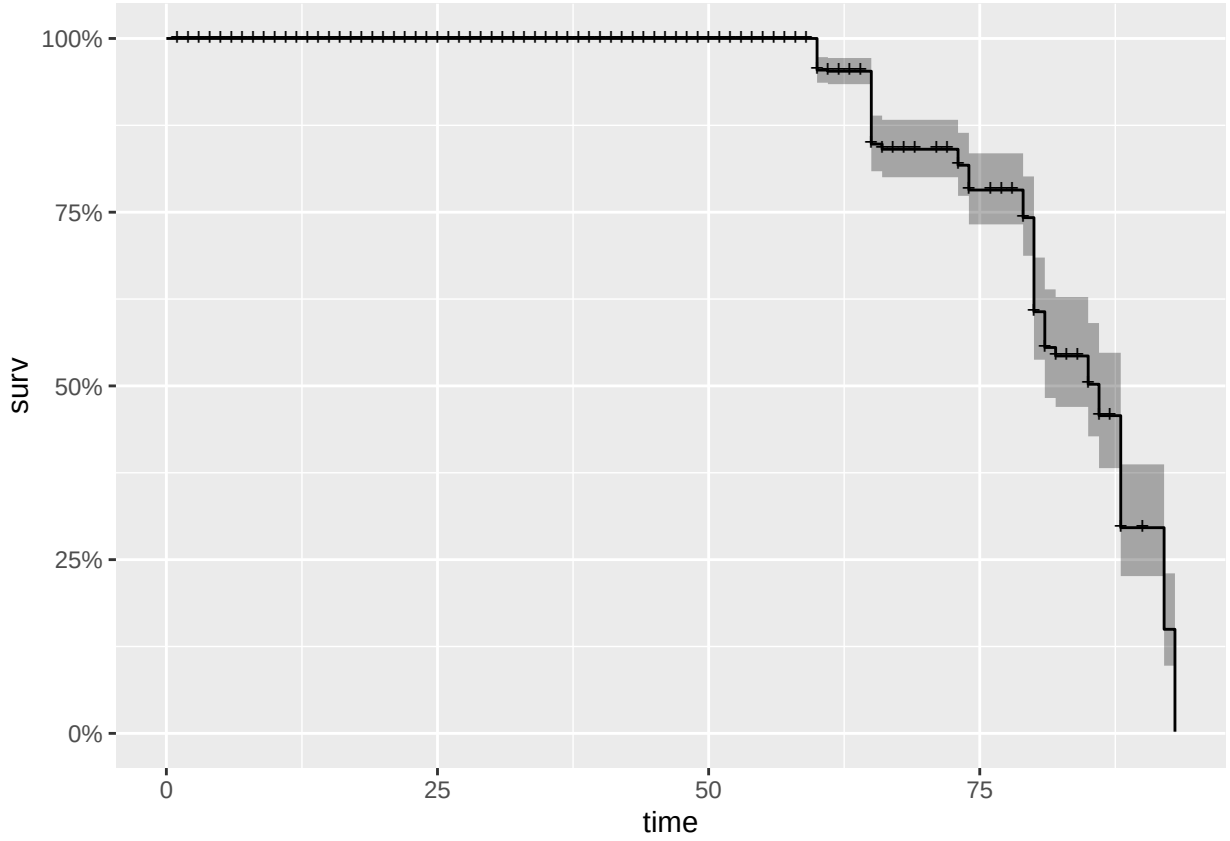
```
## Call:
## coxph(formula = Surv(lifetime, broken) ~ . - id - provider, data = train,
##     x = TRUE, y = TRUE)
##
##   n= 800, number of events= 326
##
##                       coef  exp(coef)   se(coef)      z Pr(>|z|)
## pressureInd     4.546e-05  1.000e+00  2.765e-03  0.016   0.9869
## moistureInd    -4.740e-03  9.953e-01  6.002e-03 -0.790   0.4297
## temperatureInd  7.092e-03  1.007e+00  2.768e-03  2.562   0.0104 *
## teamTeamB       7.096e-02  1.074e+00  1.386e-01  0.512   0.6086
## teamTeamC       9.261e-01  2.525e+00  1.548e-01  5.984 2.18e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                exp(coef) exp(-coef) lower .95 upper .95
## pressureInd       1.0000     1.0000    0.9946     1.005
## moistureInd       0.9953     1.0048    0.9836     1.007
## temperatureInd    1.0071     0.9929    1.0017     1.013
## teamTeamB         1.0735     0.9315    0.8182     1.409
## teamTeamC         2.5248     0.3961    1.8641     3.420
##
## Concordance= 0.598  (se = 0.021 )
## Likelihood ratio test= 46.11  on 5 df,   p=9e-09
## Wald test            = 48.17  on 5 df,   p=3e-09
## Score (logrank) test = 50.67  on 5 df,   p=1e-09
```

Performances

Brier score : implémentation à la main

Puisqu'on veut prédire, on va utiliser le score de Brier (voir https://square.github.io/pysurvival/metrics/brier_score.html) pour comparer les modèles. Pour cela, on récupère les prédictions de `cox_simple` sur le jeu de données de test.

```
surv_cox_simple = survfit(cox_simple)
autoplot(surv_cox_simple)
```



puis les prédicteurs linéaires

```
lp = predict(cox_simple, newdata = test, type = "lp")
```

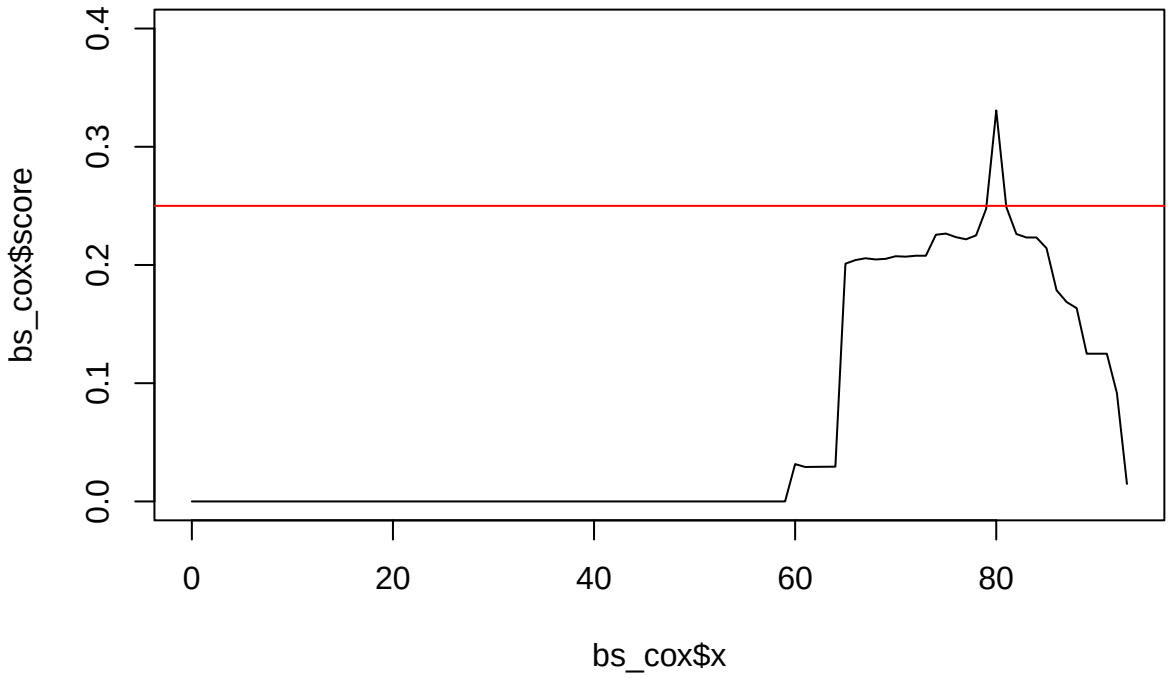
On peut alors construire une matrice qui contient les fonctions de survie estimées pour les individus du jeu de données test.

```
surv_cox_estimee = function(lp,Fbar_breslow){
  return(Fbar_breslow^exp(lp))
}
survival_matrix_cox_simple = t(sapply(lp,surv_cox_estimee,Fbar_breslow = surv_cox_simple$surv)) # je tr
times_cox_simple = surv_cox_simple$time
```

Attention, l'implémentation est différente de celle du package `riskRegression`, les scores sont donc différents.

```
x = seq(0,93,1)
bs_cox = Brier_score_vec(x,survival_matrix = survival_matrix_cox_simple, times = times_cox_simple,
                censored_times = test$lifetime, censoring_indicators = test$broken)
```

```r
plot(bs_cox$x,bs_cox$score, type = "l", ylim=c(0,0.4))
abline(h = 0.25, col="red")
```



Brier score "maison"

```r
IBrier_score(bs_cox)
```

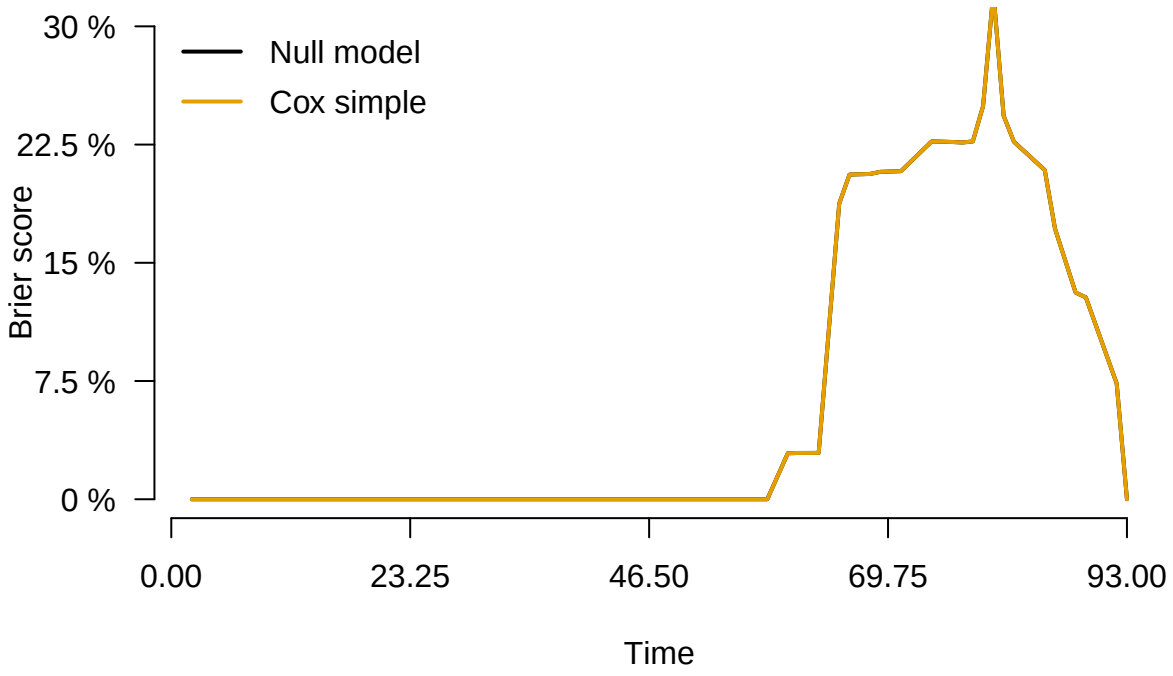
```
## [1] 0.06267107
```

On obtient un score de Brier "maison" de 0,06267. Puis on utilise l'implémentation du package `riskRegression` pour calculer le score de Brier d'une manière différente.

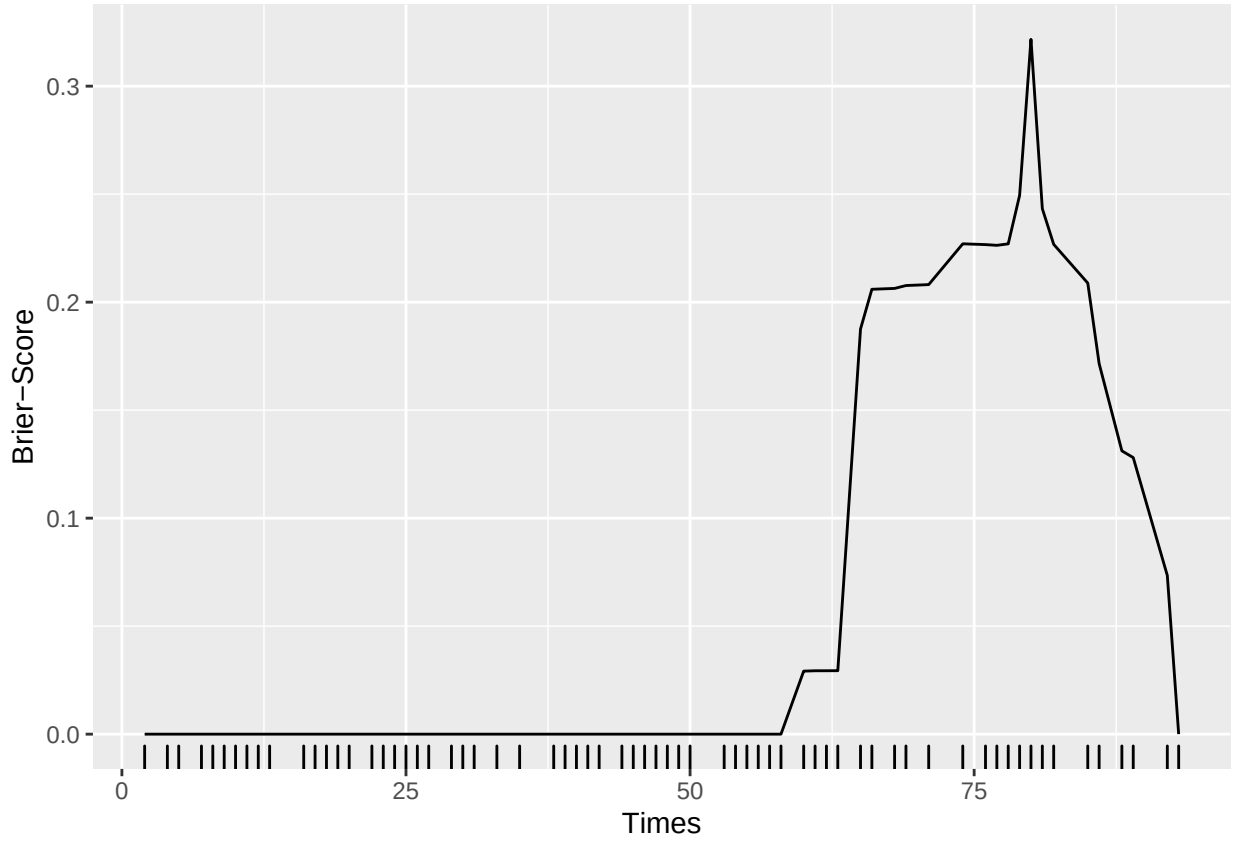
Brier score : implémentation du package `riskRegression`

```r
scores_cox_simple = Score(list("Cox simple" = cox_simple),
                          data = test,
                          formula = Surv(lifetime, broken) ~ 1,
                          cens.model = "km",
                          summary = "ipa",
                          se.fit = 0L,
                          metrics = "brier",
                          contrasts = FALSE,
                          times = sort(unique(test$lifetime)))

plotBrier(scores_cox_simple)
```




```
scores_cox_simple = tibble(scores_cox_simple$Brier$score)
scores_cox_simple = scores_cox_simple %>%
                          filter(model == "Cox simple")
performances <- tibble(brierscores_Cox_Simple =
                         scores_cox_simple$Brier,
                           times = scores_cox_simple$times)
ggplot(data = performances, mapping = aes(x = times, y = brierscores_Cox_Simple)) +
  geom_line() +
  geom_rug(sides = "b") +
  xlab("Times") +
  ylab("Brier-Score")
```



Brier score intégré

```
sum(scores_cox_simple$Brier * diff(c(0, scores_cox_simple$times)))/ max(scores_cox_simple$times)
```

```
## [1] 0.06238507
```

On obtient un score de Brier de 0,06238. Comme prévu, il y a une légère différence avec le score de Brier "maison".

Random Forest

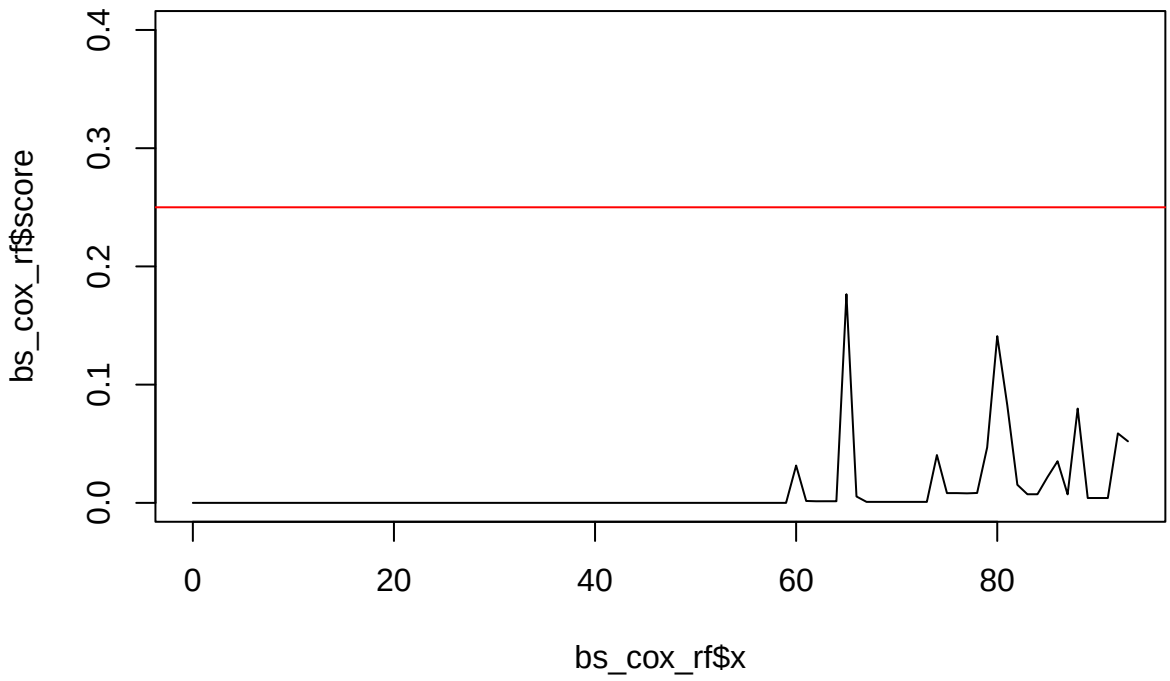
On utilise le modèle Random Forest pour essayer d'améliorer les prédictions et donc diminuer le score de Brier. On doit d'abord recoder les features discrètes de *train* et *test* avec le one_hot encoding, ce qui a déjà été réalisé sur le jeu de donnés complet lors du calcul de la corrélation.

```
one_hot_dataset_train = one_hot_dataset[trainIndex,]
one_hot_dataset_test = one_hot_dataset[-trainIndex,]
```

```
# modèle Random Forest
random_forest<-rfsrc(Surv(lifetime, broken) ~.-id, data = one_hot_dataset_train)
```

```
x = seq(0,93,1)
bs_cox_rf = Brier_score_vec(x,
                            survival_matrix = predict(random_forest, newdata = one_hot_dataset_test)$su
                            times = random_forest$time.interest,
                            censored_times = test$lifetime,
                            censoring_indicators = test$broken)
```

```
plot(bs_cox_rf$x,bs_cox_rf$score, type = "l", ylim = c(0,0.4))
abline(h = 0.25, col = "red")
```




```
# score de Brier
IBrier_score(bs_cox_rf)
```

```
## [1] 0.009308616
```

On obtient un score de Brier de 0.009, ce qui est beaucoup plus petit que le score obtenu avec le modèle de Cox. Plus le score de Brier est proche de 0, meilleur est le modèle. Par conséquent, le modèle RandomForest a de meilleures capacités de prédiction que le modèle de Cox.

Prédiction

Une machine dont les features sont données par les caractéristiques suivantes a déjà fonctionné 63 semaines. On souhaite estimer la probabilité qu'elle fonctionne encore 2 semaines.

```
# définition des covariables de la machine
new_data = tibble("id" = 2001,"pressureInd" = 96.4, "moistureInd" = 107.2, "temperatureInd" = 101.8,
                  "team" = "TeamA", "provider" = "Provider2")

# définition des covariables de la machine avec one_hot_encoding
one_hot_new_data = tibble("id" = 2001,"pressureInd" = 96.4, "moistureInd" = 107.2, "temperatureInd" = 1
                  "TeamB" = 0, "TeamC" = 0, "Provider2" = 1, "Provider3" = 0, "Provider4" = 0)
```

On calcule d'abord la probabilité avec le modèle de Cox.

```
# calcul de la probabilité
print(exp(-predict(cox_simple, newdata = c("lifetime" = 65, "broken" = 1, new_data),
                   type = "expected"))/
        exp(-predict(cox_simple, newdata = c("lifetime" = 63, "broken" = 1, new_data), type = "expected
```

```
## [1] 0.893081
```

La machine a une probabilité de 0,893 de fonctionner encore 2 semaines. Puis on calcule la probabilité avec le modèle Random Forest.

```
pred_rf <- predict(random_forest, one_hot_new_data)

# fonction de survie à la semaine 65
surv_65 <- pred_rf$survival[pred_rf$time.interest == 65]

print(pred_rf$time.interest)
```

```
##  [1] 60 61 65 66 73 74 79 80 81 82 85 86 88 92 93
```

Il n'y a pas d'évènement à la semaine 63, on prend donc la valeur de la fonction de survie à la semaine 61.

```
# fonction de survie à la semaine 63
surv_61 <- pred_rf$survival[pred_rf$time.interest == 61]
```

```
print(surv_65/surv_61)
```

```
## [1] 0.9955584
```

La machine a une probabilité de 0,99 de fonctionner encore 2 semaines avec le modèle Random Forest.

Autres modèles

On cherche un modèle qui améliore les performances du modèle Random Forest. On doit donc trouver un modèle dont le score de Brier est inférieur à 0,009. En modifiant les paramètres de base du modèle Random Forest, le score de Brier obtenu n'était pas meilleur. J'ai ensuite essayé le modèle Bagging implémenté dans la librarie `ipred`.

```
library(ipred)
bagg <- bagging(Surv(lifetime, broken) ~.-id, one_hot_dataset_train, nbagg = 30, coob = TRUE)
```

```
# score de Brier
sbrier(Surv(one_hot_dataset_test$lifetime, one_hot_dataset_test$broken),
       predict(bagg, newdata = one_hot_dataset_test))[1]
```

```
## integrated Brier score
##            0.004220315
```

Le score de Brier vaut 0,004. Ainsi, le modèle Bagging a de meilleures performances que le modèle Random Forest. On estime la probabilité que la machine fonctionne encore 2 semaines.

```
pred_bg <- predict(bagg, newdata = one_hot_new_data)
```

```
print(pred_bg[[1]]$surv[pred_bg[[1]]$time == 65] / pred_bg[[1]]$surv[pred_bg[[1]]$time == 63])
```

```
## [1] 1
```

La machine a une probabilité de 1 de fonctionner encore 2 semaines avec le modèle Bagging. Même si ce résultat semble absurde, il est logique car l'estimateur de la fonction de survie pour les machines du constructeur 2 montre qu'il n'y a jamais eu de panne avant la semaine 85. Le résultat paraît cohérent.

```
min(raw_dataset %>% filter(broken == 1, provider == "Provider2") %>% dplyr::select(lifetime))
```

```
## [1] 85
```