# Performance of Classification Models

## using the Logistic Regression

Philippe J.S. De Brouwer

October, 2019

# The Logistic Regression

# The Logistic Regression

$$\ln\left\{\frac{P[Y=1|X]}{P[Y=0|X]}\right\} = \alpha + \sum_{n=1}^{N} f_n(X_n)$$

with $X = (X_1, X_2, \ldots, X_N)$ the set of prognostic factors.

Assuming a linear model for $f_n$, the probability that $Y = 1$ is modelled as:

$$y = \frac{1}{1 + e^{-(b+a_1x_1+a_2x_2+a_3x_3+\ldots)}}$$

In R, this regression can be fitted with the function `glm()`.

# Titanic dataset

```
# if necessary: install.packages('titanic')
library(titanic)

# This provides a.o. two data-sets titanic_train and titanic_test.
# We will work further with the training-dataset.
t <- titanic_train
colnames(t)
```

```
##  [1] "PassengerId" "Survived"    "Pclass"
##  [4] "Name"        "Sex"         "Age"
##  [7] "SibSp"       "Parch"       "Ticket"
## [10] "Fare"        "Cabin"       "Embarked"
```

# A first, naive model

# A first -naive- model

Find some parameters that seem to be related and feed them in the logistic regresson

```
m1 <- glm(data = t,
          formula = Survived ~ Pclass + Sex + Age +
                               SibSp + Parch + Embarked,
          family = binomial)
```

# Summary of the model m1

```
summary(m1)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + SibSp + Parch +
##     Embarked, family = binomial, data = t)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6922  -0.6470  -0.3777   0.6260   2.4551
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  18.068902 607.683253   0.030  0.97628
## Pclass       -1.244100   0.145016  -8.579  < 2e-16 ***
## Sexmale      -2.639596   0.222169 -11.881  < 2e-16 ***
## Age          -0.043623   0.008218  -5.308 1.11e-07 ***
## SibSp        -0.354681   0.128001  -2.771  0.00559 **
## Parch        -0.044310   0.120675  -0.367  0.71348
```

# Improve model robustness

# Improve the model's robustness

Note that:

- Strings are automatically converted to factors and R introduces $n-1$ yes/no variables (eg. Sex is converted to Sexmale and the 3 ports of embarkment are in 2 variables)
- Variables that are factors but represented as numbers (eg. Pclass) are used as numerical
- All numerical values are treated as if they have a linear relationship with the dependent variable.

# Second model:

- leave out the parameters that have “no star”
- but still not normalize parameters, nor apply data-binning

```
m2 <- glm(data = t,
          formula = Survived ~ Pclass + Sex + Age + SibSp,
          family = binomial)
```

# Summary for m2:

```
summary(m2)
```

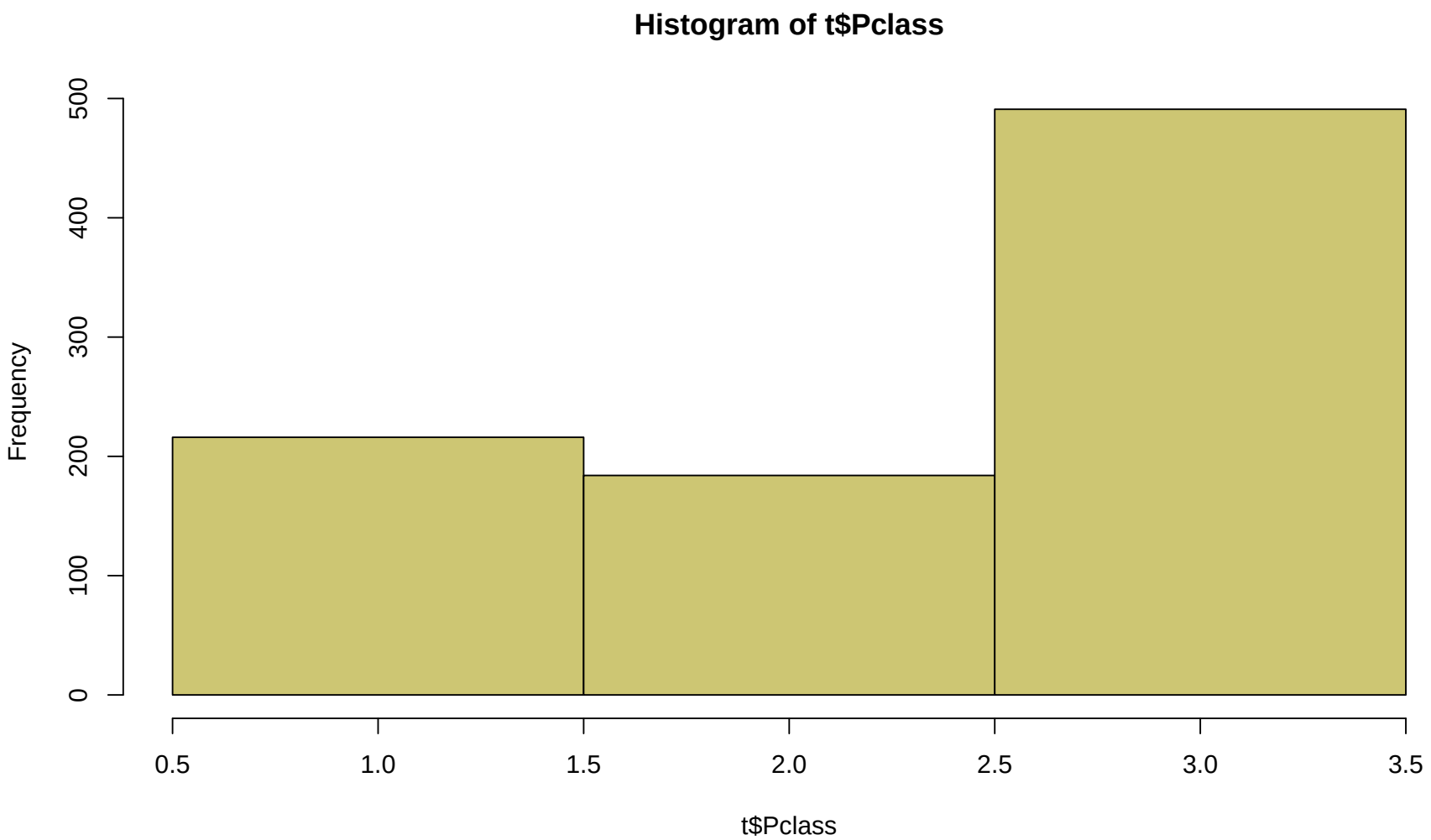
```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + SibSp, family = binomi
##     data = t)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7714  -0.6445  -0.3836   0.6276   2.4585
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.600846   0.543441  10.306  < 2e-16 ***
## Pclass      -1.317398   0.140900  -9.350  < 2e-16 ***
## Sexmale     -2.623483   0.214524 -12.229  < 2e-16 ***
## Age         -0.044385   0.008155  -5.442 5.26e-08 ***
## SibSp       -0.376119   0.121080  -3.106  0.00189 **
## ---
```

# Normalizing and binning data

# Data Exploration

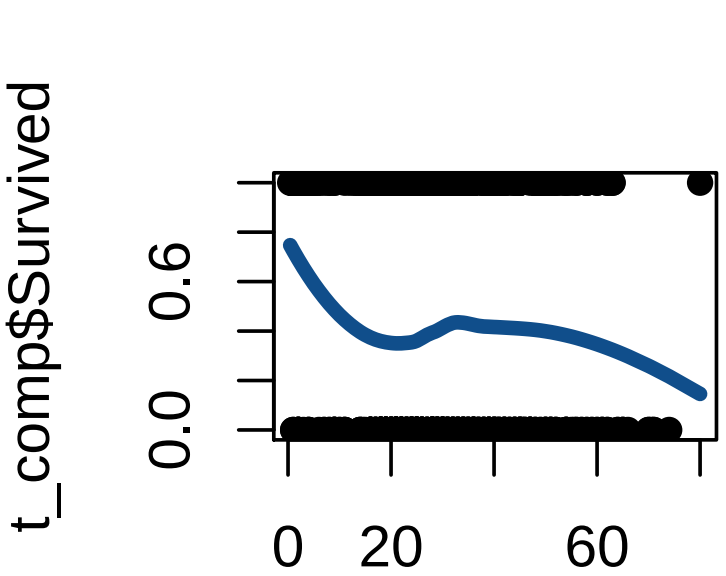
# Historgram of the class variable

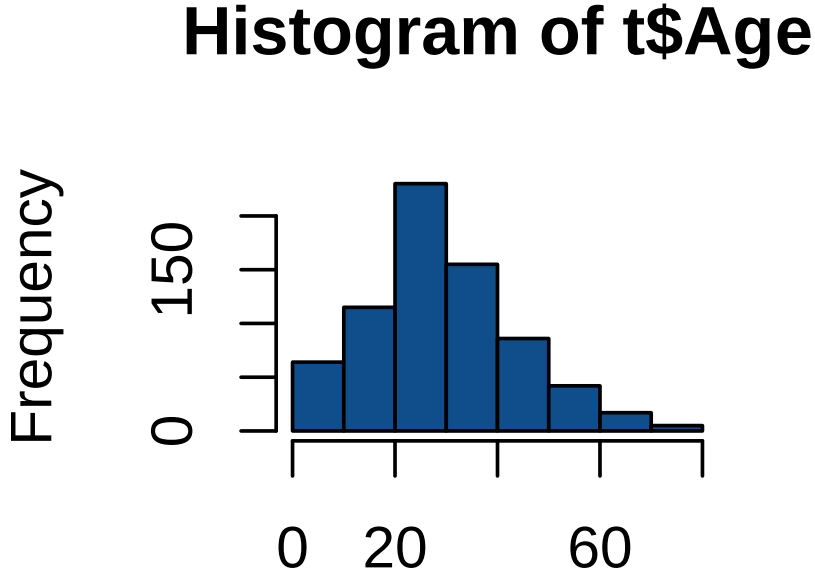
```
hist(t$Pclass, col = 'khaki3', breaks=c(0.5,1.5,2.5,3.5))
```



# The dependency relation with Age

```
t_comp <- t[complete.cases(t),]
t_comp <- t_comp[order(t_comp$Age),]  # order() returns sorted indices
tit_lo <- loess(Survived ~ Age, t_comp)
tit_pred_lo <- predict(tit_lo)
par(mfrow=c(1,2))
plot(t_comp$Age, t_comp$Survived, pch=16)
lines(t_comp$Age, tit_pred_lo, lwd=4, col='dodgerblue4')
hist(t$Age, col='dodgerblue4')
```



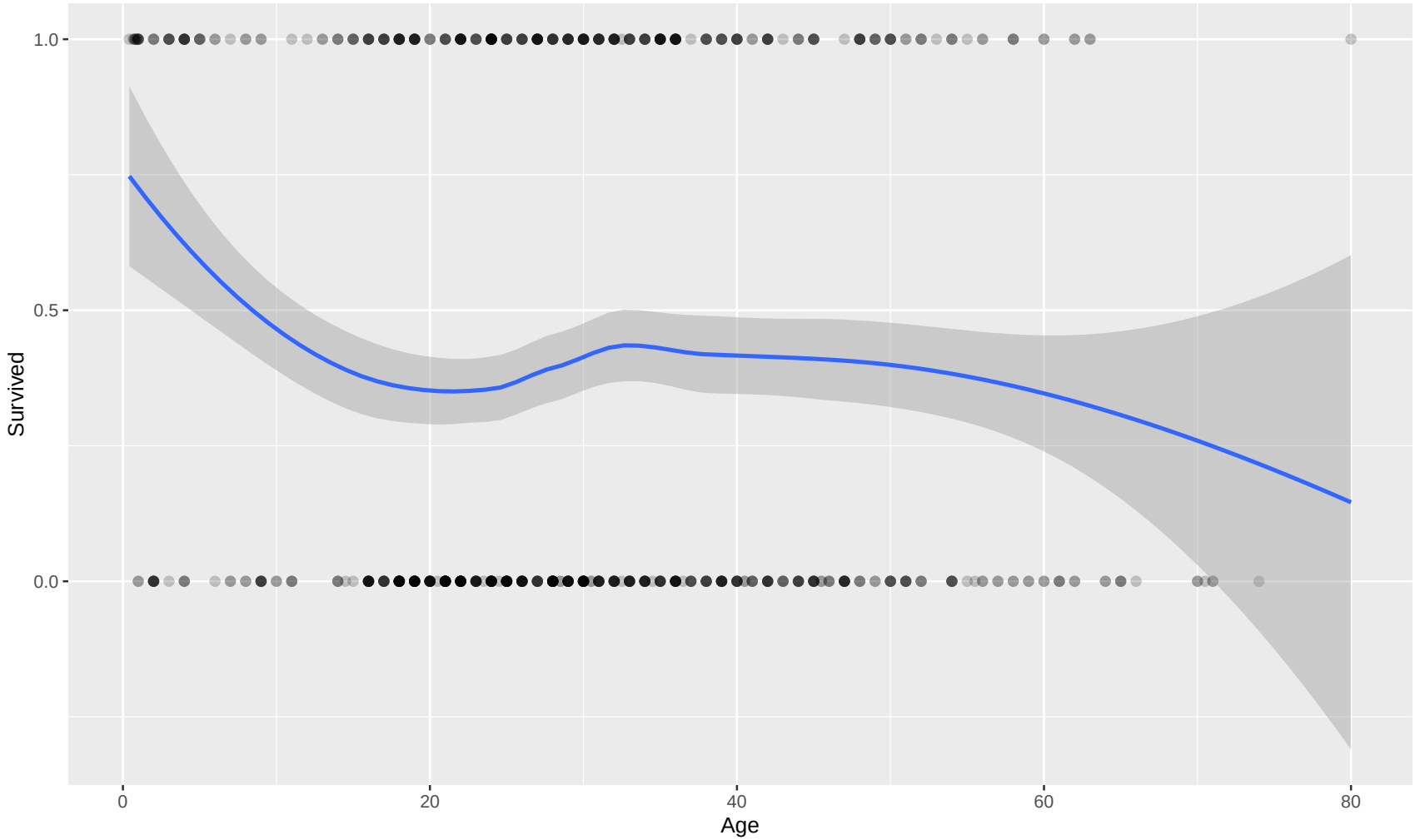



# The same information with ggplot2 – code

```
library(ggplot2)
p <- ggplot(t, mapping = aes(x = Age, y = Survived)) +
     geom_point(alpha=0.1) +
     geom_smooth(method = "loess")

# to visualize the plot, simply print it
p
```

# Similar information with ggplot2 – plot



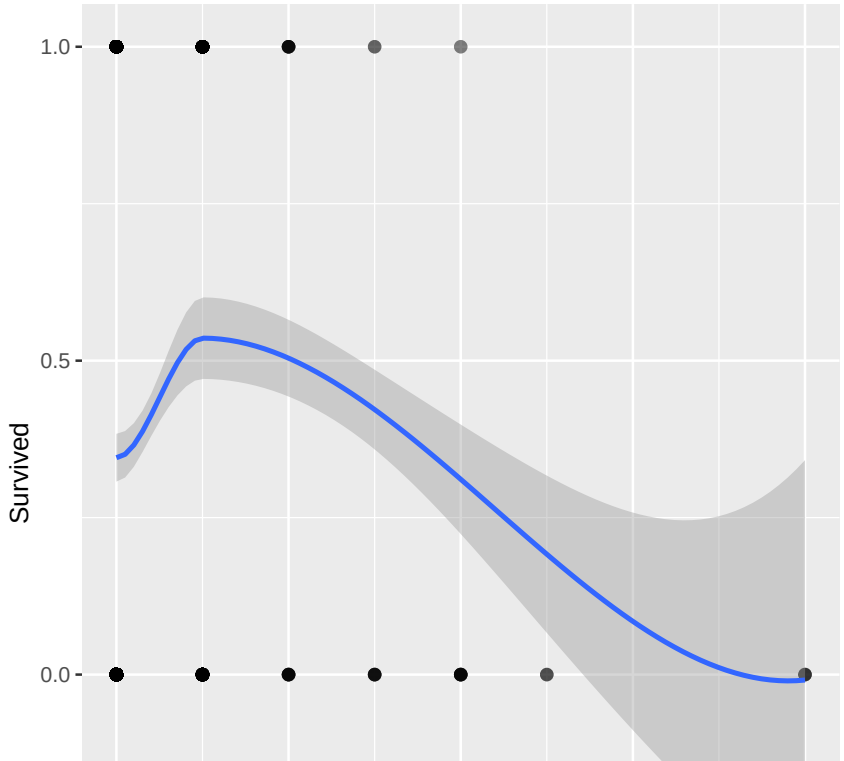
# The same analysis for the siblings variable – code

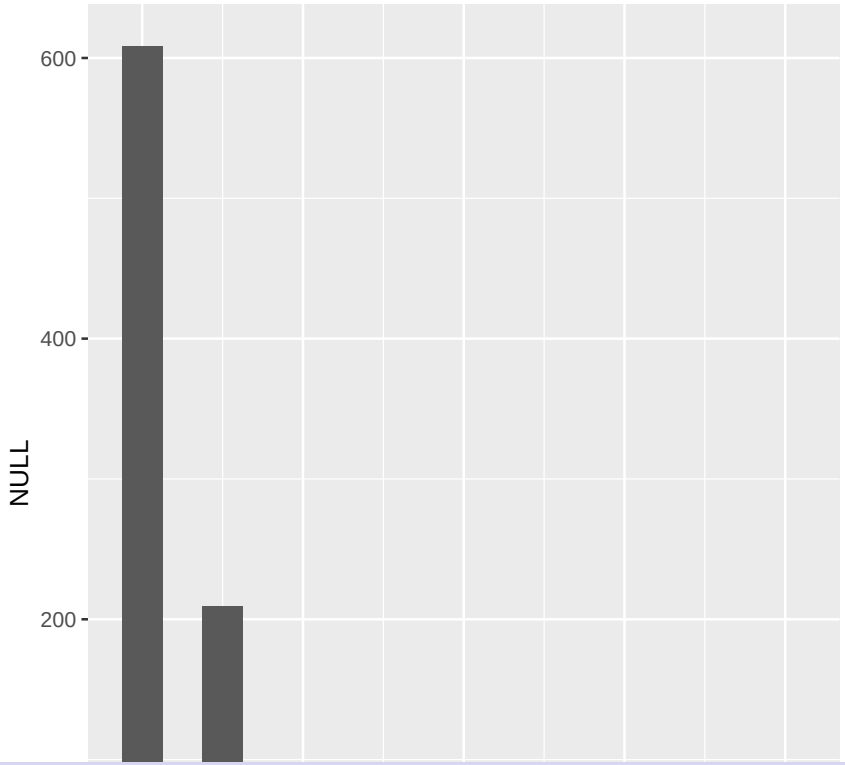
```
library(gridExtra)
p1 <- ggplot(t, mapping = aes(x = SibSp, y = Survived)) +
     geom_point(alpha=0.2, size = 2) +
     geom_smooth(method = "loess")
p2 <- qplot(t$SibSp, geom="histogram", binwidth=0.5)
grid.arrange(p1, p2, ncol=2)
```

# The same analysis for the sibings variable – plot

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```



# Normalizing and binning data

# Normalize and bin the data

```
## function to normalize d
normalize <- function(d) {
  data.frame(
    survived  = d$Survived,
    class_1   = ifelse(d$Pclass == 1, 1, 0),
    class_2   = ifelse(d$Pclass == 2, 1, 0),
    male      = ifelse(d$Sex    == 'male', 1, 0),
    age_child = ifelse(d$Age    <= 15, 1, 0),
    age_youth = ifelse(d$Age    > 15 & d$Age <= 25, 1, 0),
    age_old   = ifelse(d$Age    > 50, 1, 0),
    has_sib   = ifelse(d$SibSp  == 0, 0, 1)
  )
}
```

# Normalizing and binning data and then the logistic regression

```
t_norm <- normalize(t)
m3 <- glm(data = t_norm,
          formula = survived ~ .,
          family = binomial)
```

# Summary of m3

```
summary(m3)
```

```
##
## Call:
## glm(formula = survived ~ ., family = binomial, data = t_norm)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7120  -0.6786  -0.4192   0.6320   2.3937
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.25534    0.23861   1.070  0.28458
## class_1      2.43207    0.27167   8.952  < 2e-16 ***
## class_2      1.25414    0.24698   5.078 3.82e-07 ***
## male        -2.57096    0.21217 -12.118  < 2e-16 ***
## age_child    1.38275    0.33879   4.081 4.48e-05 ***
## age_youth   -0.07215    0.23932  -0.301  0.76305
## age_old     -0.94290    0.36337  -2.595  0.00946 **
```

# Considering Options

# Review the normalisation process (unbin age)

No binning of the variable age (linear scaling between 0 and 1)

```
normalizeNoBinAge <- function(d) {
  rngAge <- range(t$Age, na.rm = TRUE)
  dd <- data.frame(
    survived   = d$Survived,
    class_1    = ifelse(d$Pclass == 1, 1, 0),
    class_2    = ifelse(d$Pclass == 2, 1, 0),
    male       = ifelse(d$Sex    == 'male', 1, 0),
    age = (d$Age - rngAge[1]) / rngAge[2]
  )
  dd <- dd[complete.cases(dd), ]  # remove NAs
  dd
}
t_normNoBinAge <- normalizeNoBinAge(t)
mNoBinAge <- glm(data = t_normNoBinAge,
          formula = survived ~  .,
          family = binomial)
```

# Summary of m4 (normalized and optimized - Age not binned)

```
summary(mNoBinAge)
```

```
##
## Call:
## glm(formula = survived ~ ., family = binomial, data = t_normNoBinAge
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7303  -0.6780  -0.3953   0.6485   2.4657
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.1809     0.2504   4.716 2.40e-06 ***
## class_1       2.5806     0.2814   9.169  < 2e-16 ***
## class_2       1.2708     0.2440   5.207 1.92e-07 ***
## male         -2.5228     0.2074 -12.164  < 2e-16 ***
## age          -2.9588     0.6125  -4.831 1.36e-06 ***
## ---
## Signif. codes:
```

# Try the same but *bin Age anyhow*

Mix teenages and children, leave out siblings and remove intercept

```
normalizeBinAge <- function(d) {
  dd <- data.frame(
    survived   = d$Survived,
    class_1    = ifelse(d$Pclass == 1, 1, 0),
    class_2    = ifelse(d$Pclass == 2, 1, 0),
    male       = ifelse(d$Sex    == 'male', 1, 0),
    age_youth  = ifelse(d$Age    <= 20, 1, 0),
    age_old    = ifelse(d$Age    > 50, 1, 0)
  )
  dd <- dd[complete.cases(dd), ]  # remove NAs
  dd
}
t_normBinAge <- normalizeBinAge(t)
mBinAge <- glm(data = t_normBinAge,
          formula = survived ~ 0 +  ., # Intercept has no stars
          family = binomial)
```

# Summary of m4 (normalized and optimized)

```
summary(mNoBinAge)
```

```
##
## Call:
## glm(formula = survived ~ ., family = binomial, data = t_normNoBinAge
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7303  -0.6780  -0.3953   0.6485   2.4657
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.1809     0.2504   4.716 2.40e-06 ***
## class_1       2.5806     0.2814   9.169  < 2e-16 ***
## class_2       1.2708     0.2440   5.207 1.92e-07 ***
## male         -2.5228     0.2074 -12.164  < 2e-16 ***
## age          -2.9588     0.6125  -4.831 1.36e-06 ***
## ---
## Signif. codes:
```

# Weight of Evidence and Information Value

# Weight of Evidence (WOE)

## Weight of Evidence (WOE)

for each bin $i$ of variable $j$ (or for each binary variable $j$) is defined as:

$$WOE_{ij} = log\left\{\frac{\frac{\#G_{ij}}{\#G}}{\frac{\#B_{ij}}{\#B}}\right\}$$

This is a measure of predicting power of a binned variable

# Information Value

## Information Value (IV)

of bin $i$ for variable $j$ is:

$$IV_{ij} = \left( \frac{\#G_{ij}}{\#G} - \frac{\#B_{ij}}{\#B} \right) .WOE_{ij}$$

## Rule of thumb:

| IV | predictability |
|---|---|
| $< 0.02$ | not predictive |
| $0.02 - 0.3$ | weak |
| $0.1 - 0.3$ | medium |
| $0.3 - 0.5$ | strong |
| $> 0.5$ | suspicious |

# WOE and IV in R

```
#install.packages("InformationValue")
library(InformationValue)
WOETable(X = factor(t$Pclass), Y = t$Survived, valueOfGood=1) %>%
   kable
```

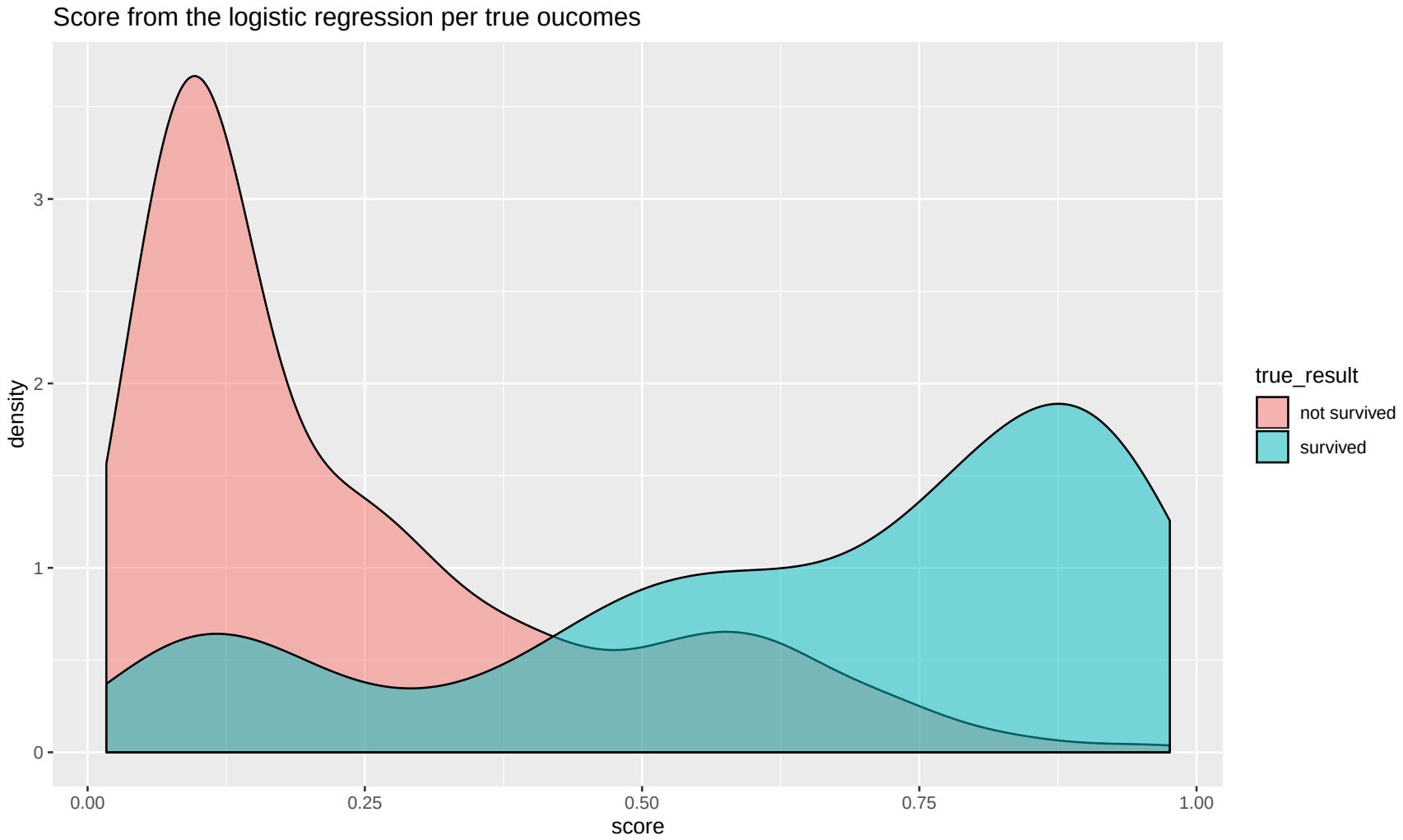
| CAT | GOODS | BADS | TOTAL | PCT_G | PCT_B | WOE | |
|---|---|---|---|---|---|---|---|
| 1 | 136 | 80 | 216 | 0.3976608 | 0.1457195 | 1.0039160 | 0.252 |
| 2 | 87 | 97 | 184 | 0.2543860 | 0.1766849 | 0.3644848 | 0.028 |
| 3 | 119 | 372 | 491 | 0.3479532 | 0.6775956 | -0.6664827 | 0.219 |

```
## also functions WOE() and IV(), eg.
# IV of a categorical variable is the sum of IV of its categories
IV(X = factor(t$Pclass), Y = t$Survived, valueOfGood=1)
```

```
## [1] 0.5009497
## attr(,"howgood")
## [1] "Highly Predictive"
```

# Model Performance for Classification Models

# Whichmodel is best? What is "best" anyhow?



# Adding predictions to the model

# predictions

```
# scores between 0 and 1 (odds)
predicScore <- predict(object=mNoBinAge,type="response")
head(predicScore)
```

```
##         1         2         3         4         5
## 0.10526285 0.91463372 0.55842450 0.92290788 0.06780678
##         7
## 0.32235446
```

```
# introduce a cut-off level above which we assume survival
predic <- ifelse(predicScore > 0.7, 1, 0)
head(predic)
```

```
## 1 2 3 4 5 7
## 0 1 0 1 0 0
```

# Performance Measures

# Some definitions

- *Accuracy*: the proportion of predictions that were correctly identified.
- Positive Predictive Value or *Precision*: the proportion of positive cases that correct.
- Negative Predictive Value : the proportion of negative cases that were correctly identified.
- *Sensitivity* or Recall : the proportion of actual positive cases which are correctly identified.
- *Specificity* : the proportion of actual negative cases which are correctly identified.

# Confusion Matrix

| | observed pos. | observed neg. | |
|---|---|---|---|
| model pos. | TP | FP | pos.pred.val=TP/(TP+FP) |
| model neg. | FN | TN | neg.pred.val=TN/(FN+TN) |
| | sensitivity =TP/(TP+FN) | specificity =TN/(FP+TN) | accuracy = (TP+TN)/(TP+FN+FP+T |

- TPR = True Positive Rate = Sensitivity = TP/(TP+FN)
- FPR = False Positive Rate = 1 - Specificity = FP/(FP+TN)

# Confusion matrix

## the confusion matrix

```
confusion_matrix <- table(predic, t_normNoBinAge$survived)
rownames(confusion_matrix) <- c("predicted_death",
                                "predicted_survival")
colnames(confusion_matrix) <- c("observed_death",
                                "observed_survival")
confusion_matrix
```

```
##
## predic                observed_death observed_survival
##   predicted_death                407                134
##   predicted_survival              17                156
```

as a precentage:

```
confusion_matrixPerc <- sweep(confusion_matrix, 2,
                        margin.table(confusion_matrix,2),"/")
round(confusion_matrixPerc,2)
```

```
##
## predic                observed_death observed_survival
##   predicted_death               0.96              0.46
##   predicted_survival            0.04              0.54
```

# Performance measurement in R

# ROCR

```
library(ROCR)
```

```
## Loading required package: gplots
```

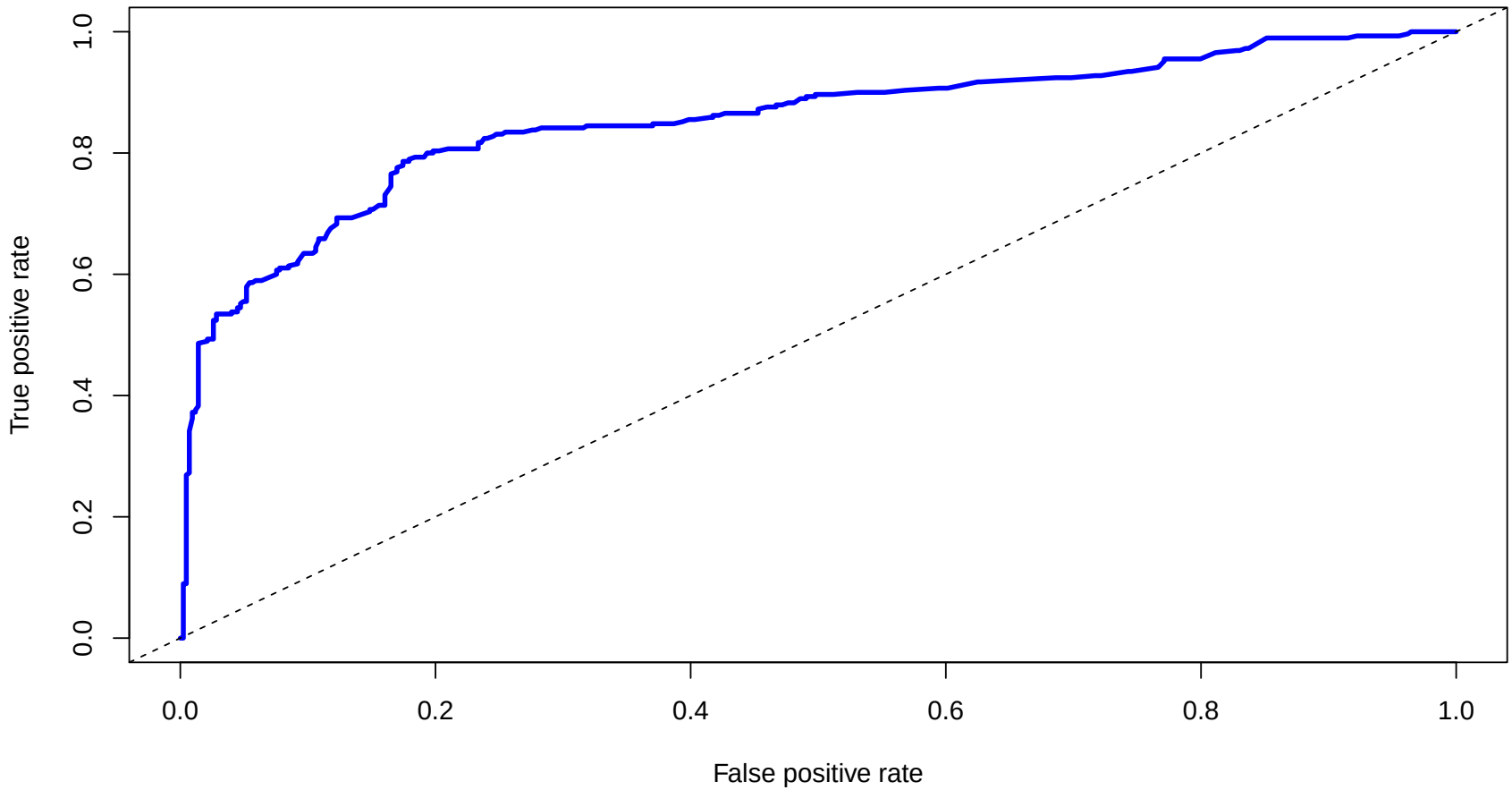
```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
pred <- prediction(predict(mNoBinAge,type="response"),
                   t_normNoBinAge$survived)
```

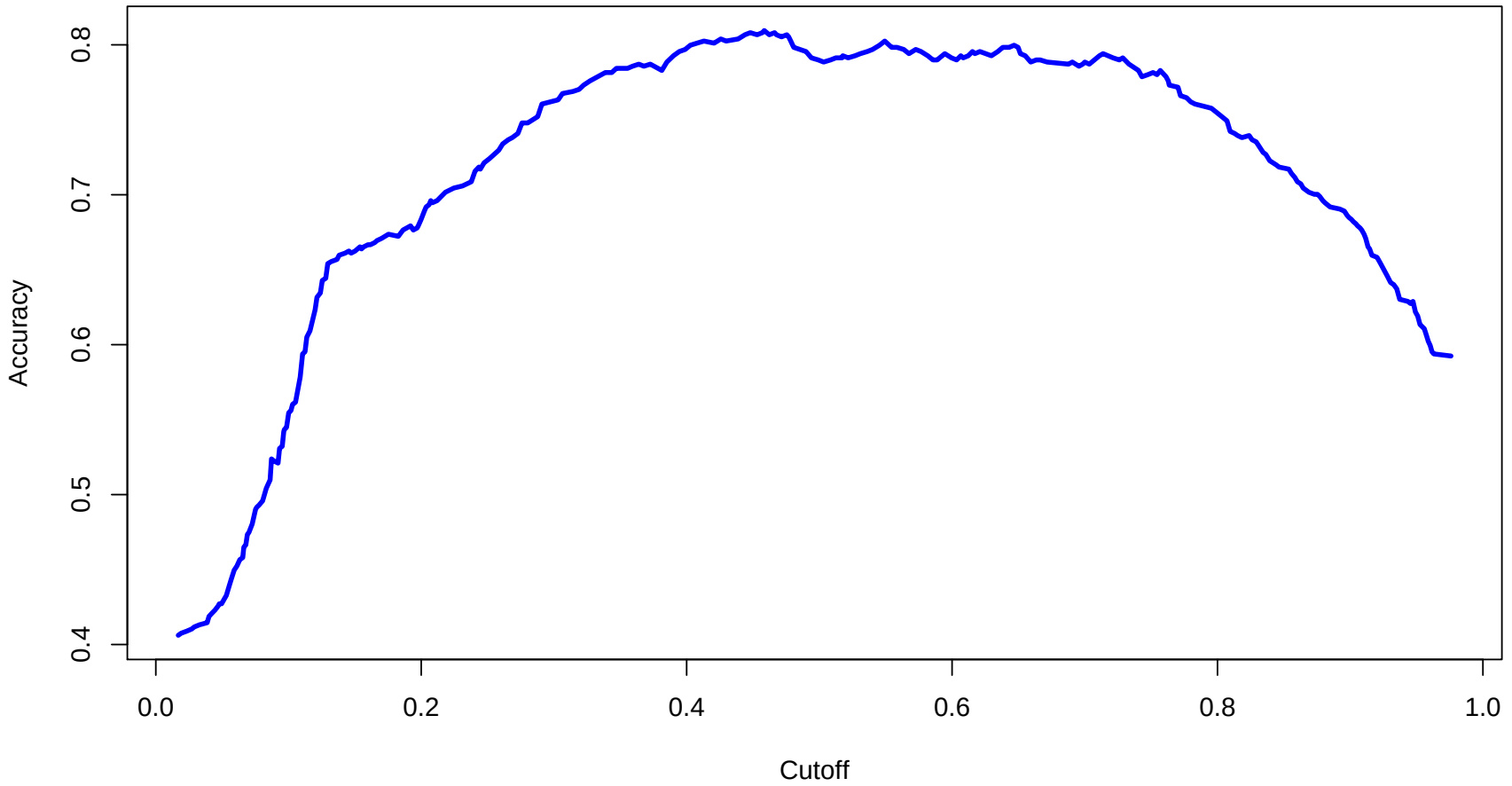
# Visualize the ROC curve

```
plot(performance(pred, "tpr", "fpr"), col="blue", lwd=3)
abline(0,1,lty=2)
```

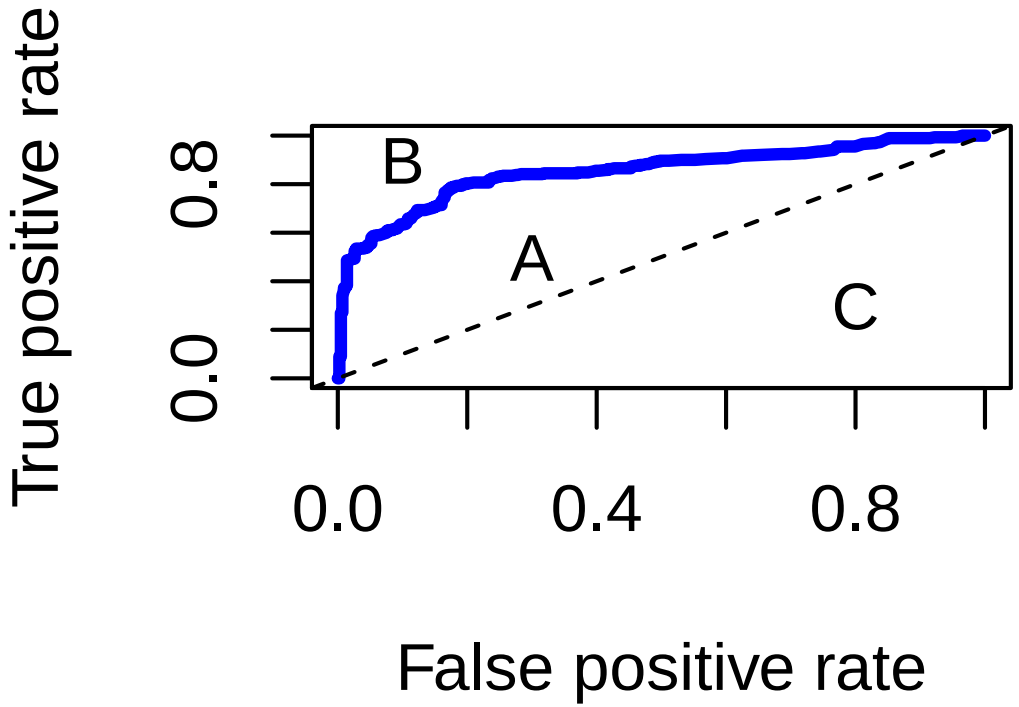


# Plotting the accuracy (in function of the cut-off)

```
plot(performance(pred, "acc"), col="blue", lwd=3)
```



# AUC



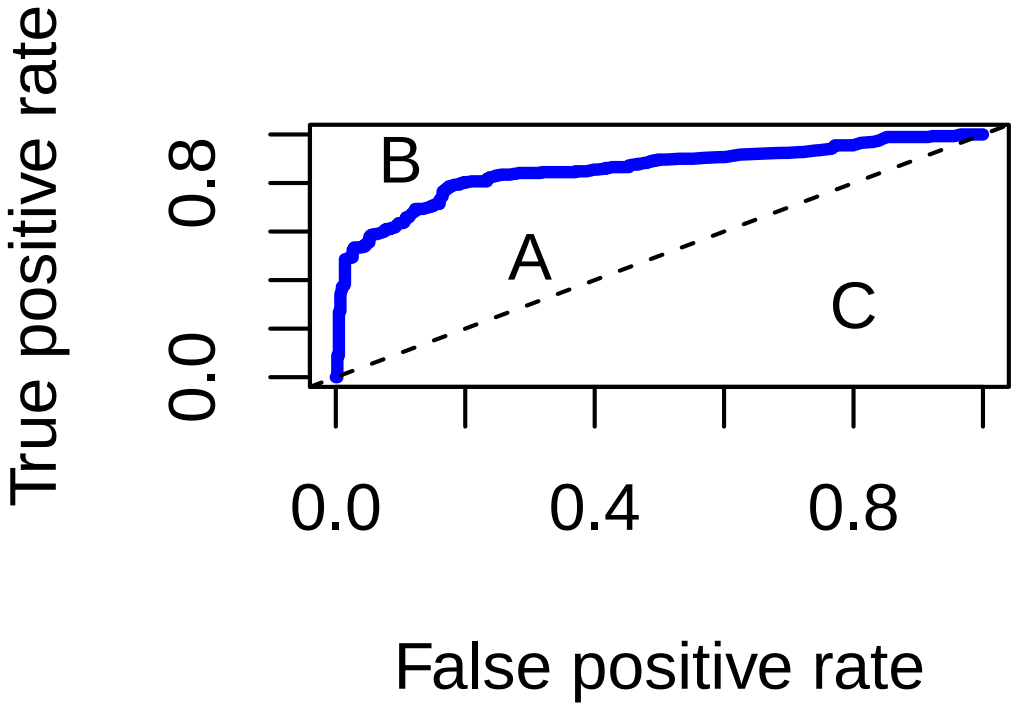
# AUC

AUC is the area under the ROC curve ($A + C = A + \frac{1}{2}$ in the figure)

```
AUC <- attr(performance(pred, "auc"), "y.values")[[1]]
AUC
```

```
## [1] 0.8523219
```

# GINI




## Gini

Gini is the area under the ROC curve compared to the aread above ($\frac{A}{A+B} = \frac{A}{0.5} = 2A$ in the figure and since $A + B = C = 0.5$, related to AUC)

```
paste("the Gini is:",round(2 * AUC - 1, 2))
```

```
## [1] "the Gini is: 0.7"
```

# KS

## KS

The Kolmogorov-Smirnov test is the largest distance between two cummulative distribution functions. For the performance of a binary classification model (such as logistic regression), it is the largest difstance between the cdf of the score of the true positives compared to the distribution of the true negatives.

# KS Visualized – code 1/2

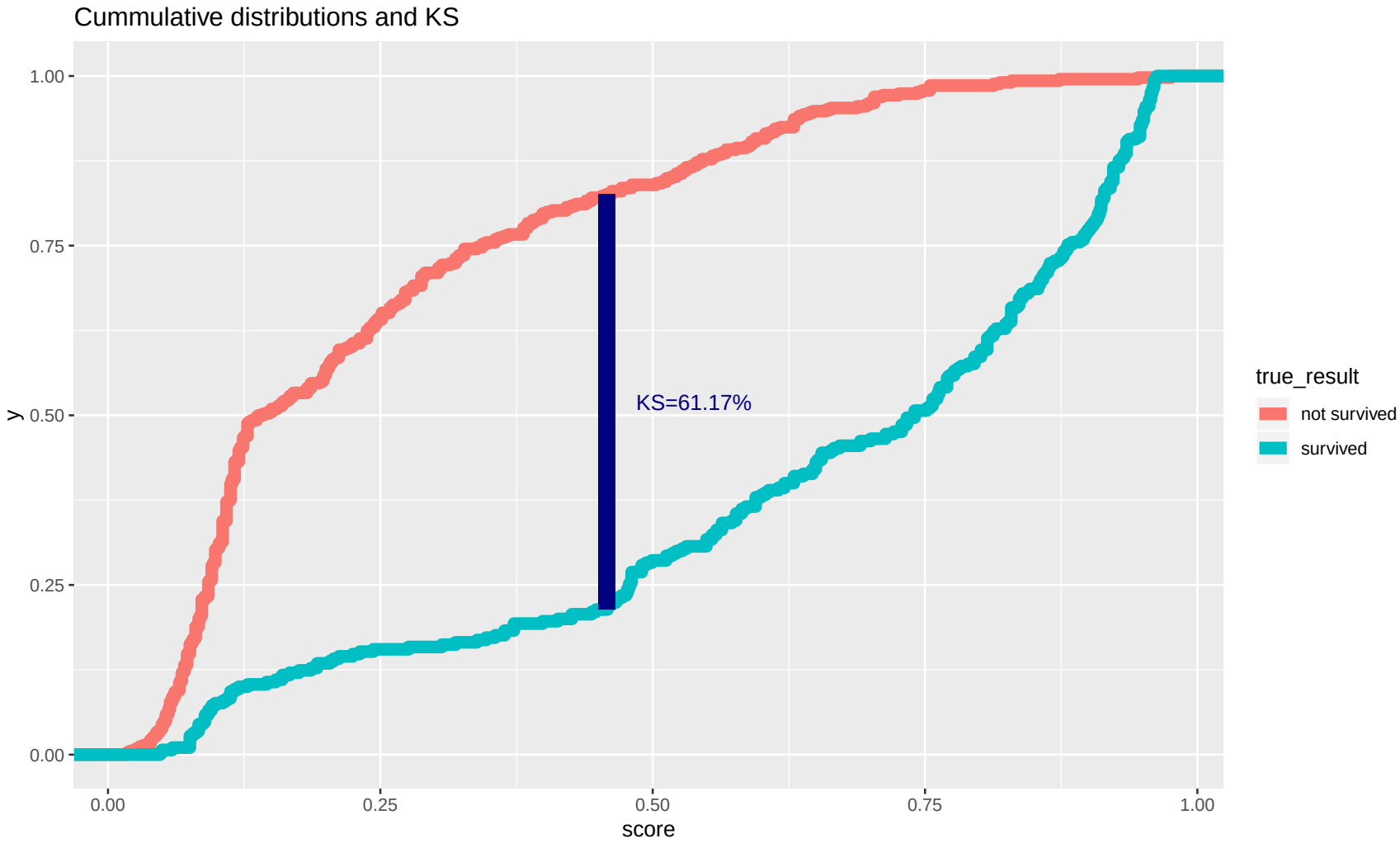
```
predicScore <- predict(object=mNoBinAge,type="response")
d0 <- data.frame(
       score = as.vector(predicScore)[t_normBinAge$survived == 0],
       true_result = 'not survived')
d1 <- data.frame(
       score = as.vector(predicScore)[t_normBinAge$survived == 1],
       true_result = 'survived')
d <- rbind(d0, d1)
d <- d[complete.cases(d),]

cumDf0 <- ecdf(d0$score)
cumDf1 <- ecdf(d1$score)
x <- sort(d$score)
cumD0 <- cumDf0(x)
cumD1 <- cumDf1(x)
diff <- cumD0 - cumD1
y1  <- gdata::first(cumD0[diff == max(diff)])
y2  <- gdata::first(cumD1[diff == max(diff)])
x1  <- x2 <- quantile(d0$score, probs=y1, na.rm=TRUE)
```

# KS Visualized – code 2/2

```
library(ggplot2)
p <- ggplot(d, aes(x = score)) +
     stat_ecdf(geom = "step", aes(col = true_result), lwd=3) +
     ggtitle('Cummulative distributions and KS') +
     geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
                  color='navy', lwd=4) +
     annotate("text",
              label = paste0("KS=",round((y1-y2)*100,2),"%"),
              x = x1 + 0.08, y = y2+(y1-y2)/2, color = "navy")
```

# KS Visualized – code 2/2



# KS using the function ks.test() from the package {stats}

```
ks.test(attr(pred,"predictions")[[1]],
        t_normNoBinAge$survived,
        alternative = 'greater')
```

```
## Warning in ks.test(attr(pred, "predictions")[[1]],
## t_normNoBinAge$survived, : p-value will be approximate in
## the presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  attr(pred, "predictions")[[1]] and t_normNoBinAge$survived
## D^+ = 0.40616, p-value < 2.2e-16
## alternative hypothesis: the CDF of x lies above that of y
```

# This standard test does not work well in our example.

```
# Here is an alternative:
perf <- performance(pred, "tpr", "fpr")
ks<-max(attr(perf,'y.values')[[1]]-attr(perf,'x.values')[[1]])
ks
```

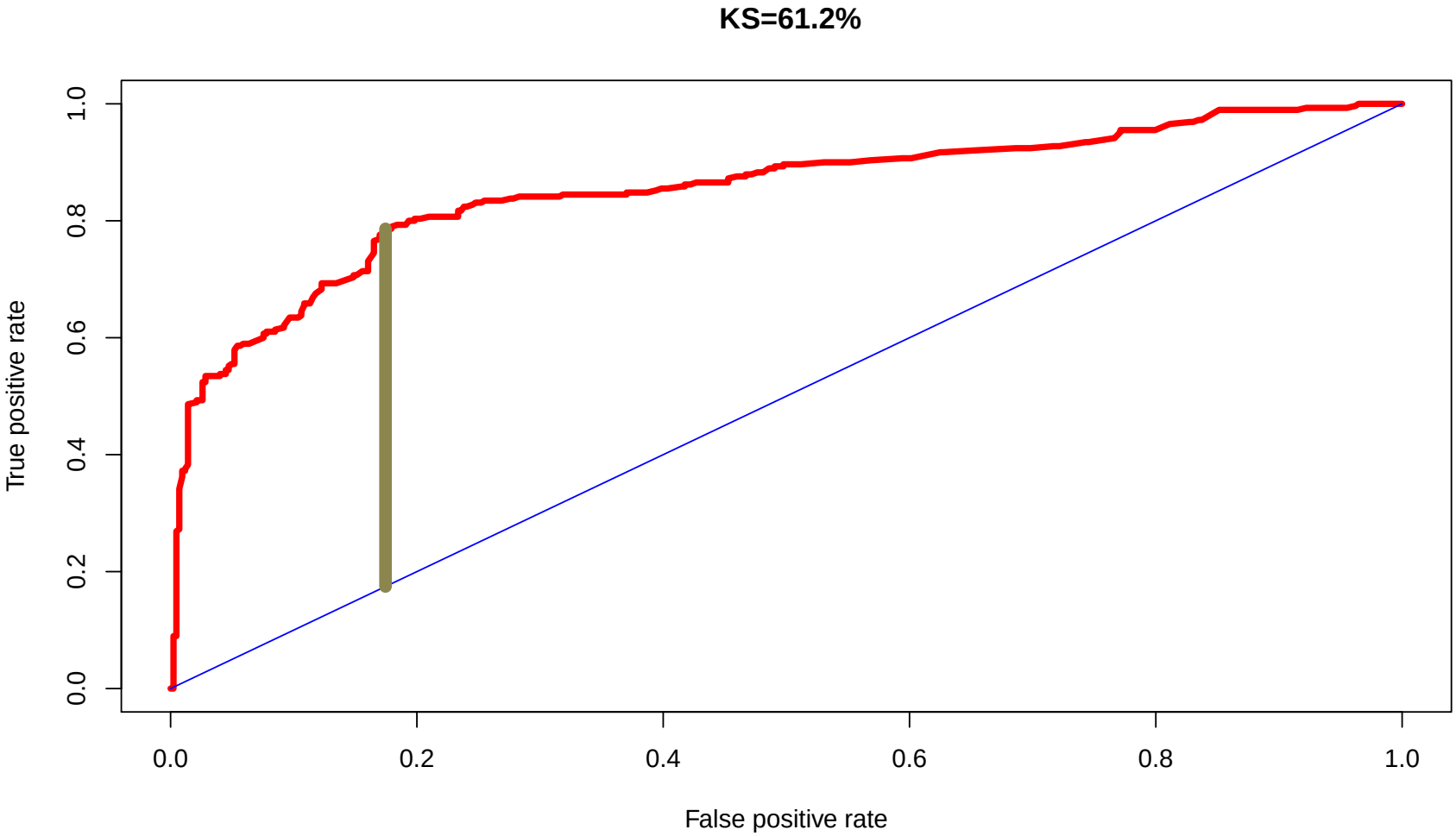
```
## [1] 0.6116786
```

```
# note: the following also works:
ks <- max(perf@y.values[[1]] - perf@x.values[[1]])
ks
```

```
## [1] 0.6116786
```

```
# or use the cumulative distribution functions (as we did to plot
# the visualization of the KS 4 slides back)
```

# The visualization of the KS statistic



# visualize the KS in the plot: the code

This code generates the plot from previous slide:

```
plot(perf,main=paste0(' KS=',round(ks*100,1),'%'),
     lwd=4, col='red')
lines(x = c(0,1),y=c(0,1), col='blue')
# the KS line
diff <- perf@y.values[[1]] - perf@x.values[[1]]
xVal <- attr(perf,'x.values')[[1]][diff == max(diff)]
yVal <- attr(perf,'y.values')[[1]][diff == max(diff)]
lines(x = c(xVal, xVal), y = c(xVal, yVal),
      col = 'khaki4', lwd=8)
```

# Compare the two models (1): extracting function

```
library(ROCR)

# define a function that returns an overview
get_perf <- function(model, data, label) {
  pred <- prediction(predict(model,type="response", newdat=data),
                     data$survived)
  perf <- performance(pred, "tpr", "fpr")
  auc <- attr(performance(pred, "auc"), "y.values")[[1]]
  df <- data.frame(
              model = label,
              AUC   = round(auc, 4),
              gini = round(2 * auc - 1, 4),
              ks   = round(max(attr(perf,'y.values')[[1]] -
                               attr(perf,'x.values')[[1]]),
                           4)
              )
 df
}
```

# The result for the 2 models

```
rbind(
  get_perf(mNoBinAge, t_normNoBinAge, 'No Age Binning'),
  get_perf(mBinAge,   t_normBinAge,   'Age Binning'   )
  )
```

```
##              model    AUC   gini     ks
## 1 No Age Binning 0.8523 0.7046 0.6117
## 2    Age Binning 0.8457 0.6915 0.5680
```

# Sampling and Cross Validation

# Is it really better not to bin age?

we would like to know if that makes the model more robust ...

# Sampling

```
library(tidyverse)
library(modelr)
library(titanic)
library(ROCR)
# also needed: our functions normalizeBinAge and normalizeNoBinAge
set.seed(20181)
set_norm_ti_data <- function() {
  # Sample once
  smpl <- t         %>%
         resample_partition(c(train = 0.6, test = 0.4))
  # Use that selection to make all data-sets
  d_NB_train <<- as_tibble(smpl$train)  %>% normalizeNoBinAge
  d_NB_test  <<- as_tibble(smpl$test)   %>% normalizeNoBinAge
  d_B_train  <<- as_tibble(smpl$train)  %>% normalizeBinAge
  d_B_test   <<- as_tibble(smpl$test)   %>% normalizeBinAge
  }
set_norm_ti_data()  # sets d_NB_train, d_NB_test, d_B_train and d_B_tes
```

# Fit the two models

```
mNoBinAge <- glm(data = d_NB_train, formula = survived ~  .,
                 family = binomial)
mBinAge <- glm(data = d_B_train,
               formula = survived ~ 0 +  ., family = binomial)
rbind(
  get_perf(mNoBinAge, d_NB_test, 'No Age Binning'),
  get_perf(mBinAge,   d_B_test,   'Age Binning'   )
  )
```

```
##             model    AUC   gini     ks
## 1 No Age Binning 0.8603 0.7206 0.6107
## 2    Age Binning 0.8464 0.6927 0.5599
```

# How to be really sure? – multiple samples

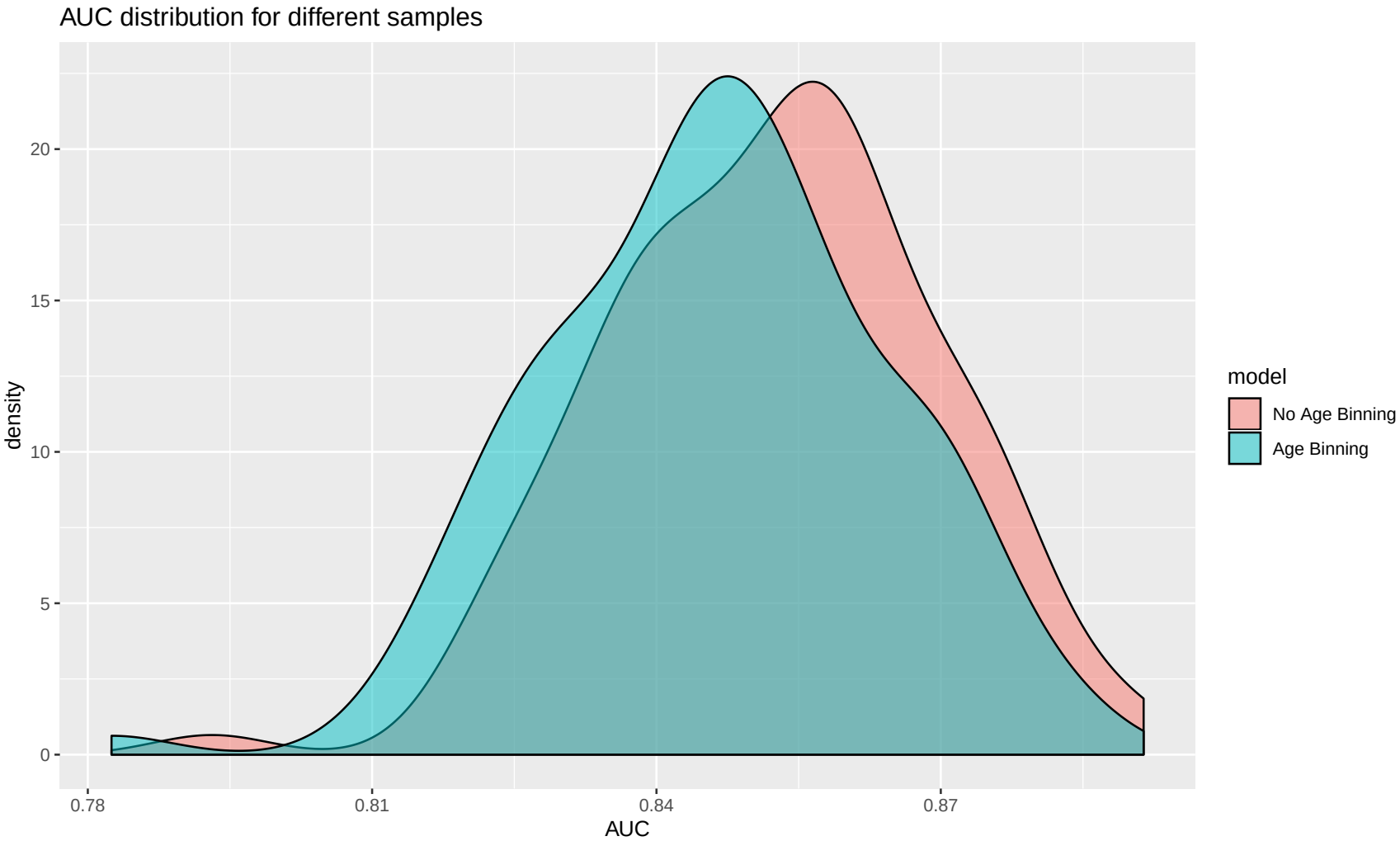
```
# prepare an empty data frame
d <- data.frame(matrix(ncol = 4, nrow = 0))
colnames (d) <- c('model', 'AUC', 'gini', 'ks')

# Create eg. 100 times the performance for different training
# and test data; then put this in a dataframe "d".
for (n in 1:100) {
  set_norm_ti_data() # sets d_NB_train, d_NB_test, d_B_train and d_B_te
  mNoBinAge <- glm(data = d_NB_train,
                   formula = survived ~  ., family = binomial)
  mBinAge <- glm(data = d_B_train,
                 formula = survived ~ 0 +  ., family = binomial)
  dtmp <- rbind(get_perf(mNoBinAge, d_NB_test, 'No Age Binning'),
                get_perf(mBinAge,   d_B_test,   'Age Binning' ))
  d <- rbind(d,dtmp)
  }
```

# Visualize AUC distribution – code

```
library(ggplot2)
p <- ggplot(d, aes(x = AUC)) +
     geom_density(aes(fill = model), alpha = 0.5) +
     ggtitle('AUC distribution for different samples')
p
```

# Visualize AUC distribution – plot



# The Optimal Cut-Off score

# Finding an Optimal Cut-off

So far we have assessed how good the model is able to separate the good outcomes from the bad outcomes. It is still an entirely different issue to determine an ideal cutoff level. To do so we have to understand what the model is trying to do and what the impact is of a type one or type two error. For example if we are making analysis of medical data and try to assess if a person has cancer and needs treatment, then it is bad to provide treatment when it is not really necessary, but it is much worse to provide no treatment when the patient needs it.

IF the model tries to predict if a customer would default on this new loan then we should be able to compare expected costs of such default with expected profit of a successful loan. In such case we can actually calculate and find an optimal cut-off.

# We will use the same example:

```
m4 <- glm(data = t_normNoBinAge,
          formula = survived ~ 0 + .,
          family = binomial)
summary(m4)
```

```
##
## Call:
## glm(formula = survived ~ 0 + ., family = binomial, data = t_normNoBi
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2633  -0.7743  -0.3891   0.7287   2.3533
##
## Coefficients:
##         Estimate Std. Error z value Pr(>|z|)
## class_1   2.5005     0.2653   9.426  < 2e-16 ***
## class_2   1.5199     0.2310   6.579 4.75e-11 ***
## male     -2.1515     0.1922 -11.192  < 2e-16 ***
## age      -0.9918     0.4276  -2.319   0.0204 *
```

# Performanc and Predictions

## the predictions:

```
pred <- prediction(predict(m4,type="response"),
                   t_normNoBinAge$survived)
```

## the performance object:

```
perf <- performance(pred, "tpr", "fpr")
```

# Create a function that determines the optimal cut-off value

```
get_best_cutoff <- function(x, y, cutoff){
        d = (x - 0)^2 + (y - 1)^2
        idx = which(d == min(d))
        c(sensitivity = y[[idx]],
          specificity = 1 - x[[idx]],
          cutoff = cutoff[[idx]])
    }
opt_cut_off = function(perf, pred){
    mapply(FUN=get_best_cutoff,
           perf@x.values,
           perf@y.values,
           pred@cutoffs)
   }
opt_cut_off(perf, pred)
```

```
##                    [,1]
## sensitivity 0.7758621
## specificity 0.7948113
## cutoff      0.4183574
```

# Cost functions

The previous only works when false positives and false negatives have the same "cost". In general they will have different cost.

```
# eg. cost.fp = 1 x cost.fn
perf_cst1 <- performance(pred, "cost", cost.fp = 1)

# the optimal cut-off is then the same as in previous code sample
pred@cutoffs[[1]][which.min(perf_cst1@y.values[[1]])]
```

```
##       742
## 0.4769701
```

```
# eg. cost.fp = 5 x cost.fn
perf_cst2 <- performance(pred, "cost", cost.fp = 5)

# the optimal cut-off is now:
pred@cutoffs[[1]][which.min(perf_cst2@y.values[[1]])]
```

```
##        16
## 0.6991535
```

# Cross Validation

# Types of cross validation

# Types of cross validation

## Overview

1. Simple: split data in training and testing set
2. Monte Carlo: random splits between training and test-data
3. k-fold: k samples, with k-1 in training and 1 block in test

## Workflow

1. get data
2. split data
3. compare results of training and testing set

# The example data (repeat)

```
library(titanic)
t <- titanic_train

# consider binning age anyhow.
normalizeBinAge <- function(d) {
  dd <- data.frame(
    survived   = d$Survived,
    class_1    = ifelse(d$Pclass == 1, 1, 0),
    class_2    = ifelse(d$Pclass == 2, 1, 0),
    male       = ifelse(d$Sex    == 'male', 1, 0),
    age_youth  = ifelse(d$Age    <= 20, 1, 0),
    age_old    = ifelse(d$Age    > 50, 1, 0)
  )
  dd <- dd[complete.cases(dd), ]  # remove NAs
  dd
}
t_normBinAge <- normalizeBinAge(t)
mBinAge <- glm(data = t_normBinAge,
          formula = survived ~ . + .*., # Intercept has no stars
```

# model m2

```
frm2 <- survived ~ 0 + class_1*male + class_2*male + class_2*age_youth
m2 <- glm(data = t_normBinAge,
          formula = frm2,
          family = binomial)
summary(m2)
```

```
##
## Call:
## glm(formula = frm2, family = binomial, data = t_normBinAge)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6591  -0.6137  -0.3554   0.4631   2.3637
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## class_1             3.2731     0.5883   5.564 2.64e-08
## male               -1.8070     0.1909  -9.467  < 2e-16
## class_2             2.1788     0.4305   5.061 4.17e-07
```

# Elementary Cross Validation

# Cross-Validation in the tidyverse

```
library(modelr)
set.seed(2323)
rs <- t_normBinAge %>%
      resample_partition(c(train = 0.6, test = 0.4))
# address the datasets with: as.data.frame(rs$train)
#
dim(as.data.frame(rs$train))
```

```
## [1] 428   6
```

```
dim(as.data.frame(rs$test))
```

```
## [1] 286   6
```

# get model performance

```
library(ROCR)

# define a function that returns an overview
get_perf <- function(model, data, label) {
  pred <- prediction(predict(model,type="response", newdat=data),
                     data$survived)
  perf <- performance(pred, "tpr", "fpr")
  auc <- attr(performance(pred, "auc"), "y.values")[[1]]
  df <- data.frame(
              model = label,
              AUC   = round(auc, 4),
              gini = round(2 * auc - 1, 4),
              ks   = round(max(attr(perf,'y.values')[[1]] -
                               attr(perf,'x.values')[[1]]),
                           4)
              )
 df
}
```

# Cross validation for training and test data

```
# we used this before for 2 different models and the same data
rbind(get_perf(m2, as.data.frame(rs$train), 'train'),
      get_perf(m2,  as.data.frame(rs$test),  'test'))
```

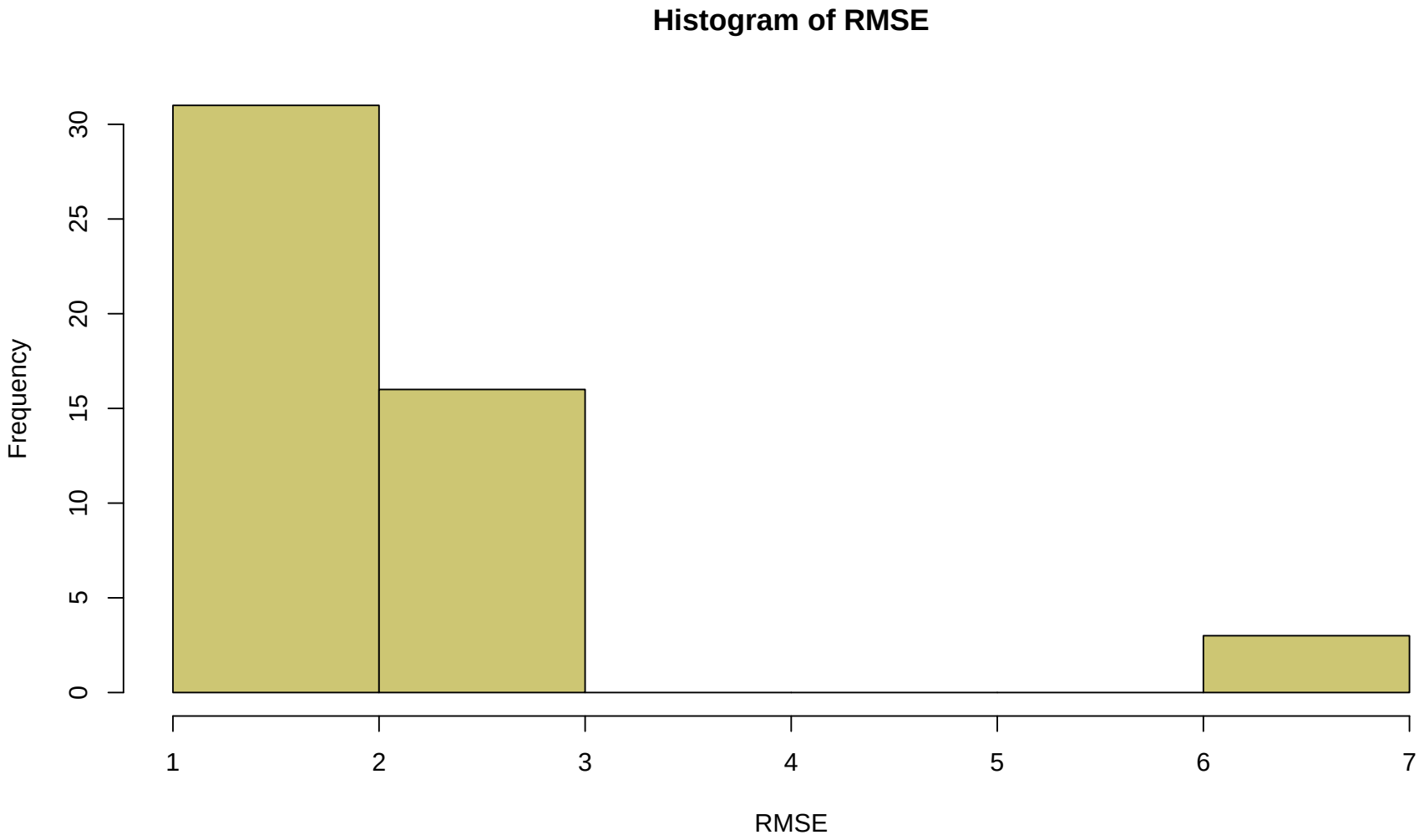
```
##   model    AUC   gini     ks
## 1 train 0.8529 0.7059 0.5376
## 2  test 0.8701 0.7402 0.5858
```

# Monte Carlo Cross Validation

# Monte Carlo Cross Validation – code

```
library(magrittr) # to access the %T>% pipe
crossv <- t_normBinAge                                  %>%
          crossv_mc(n = 50, test = 0.40)
RMSE <- crossv  %$%
        map(train, ~ glm(frm2, data = ., family = binomial))  %>%
        map2_dbl(crossv$test, rmse) %T>%
        hist(col = "khaki3", main ="Histogram of RMSE", xlab = "RMSE")
```

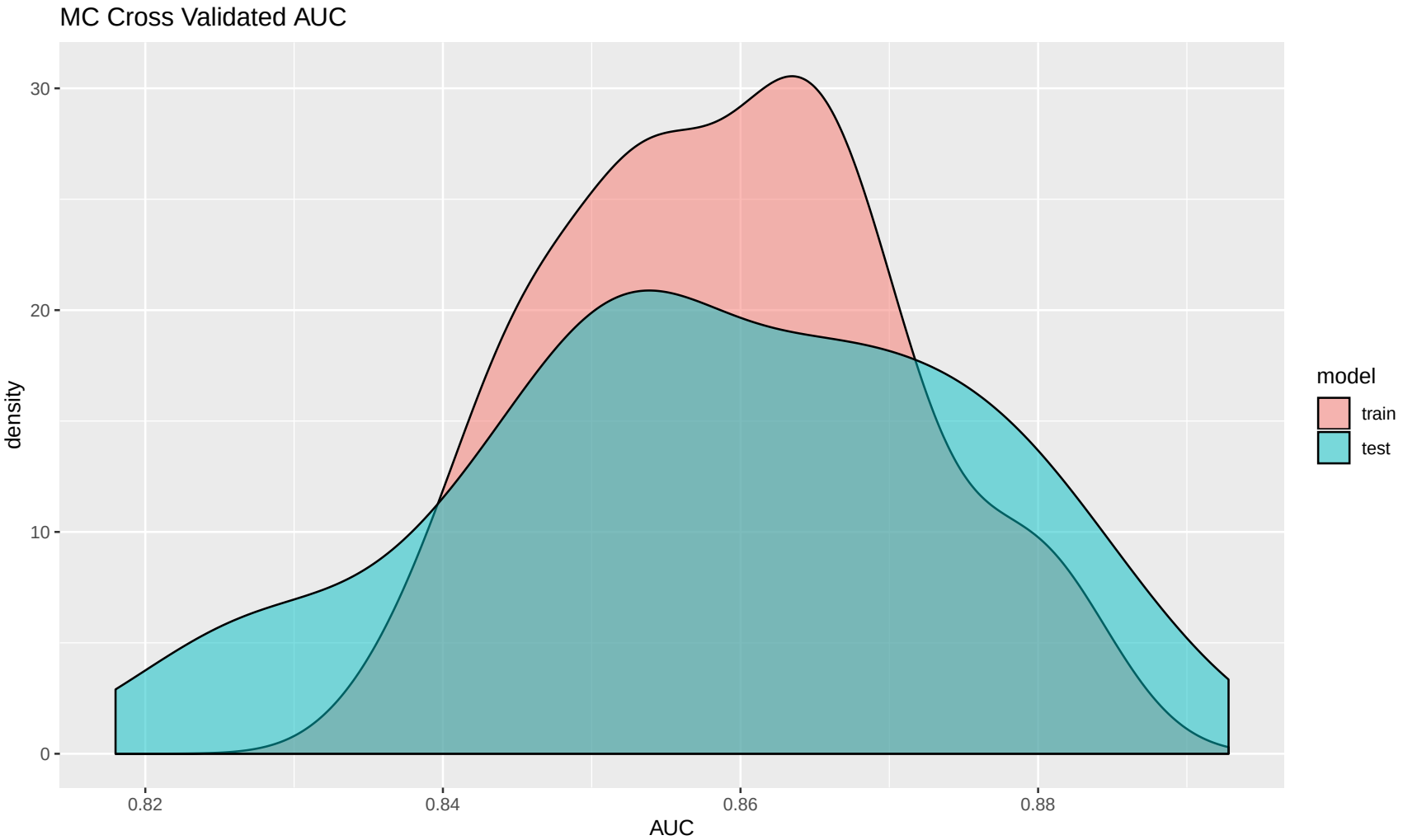
# Monte Carlo Cross Validation – plot



# Monte Carlo Cross Validation – own measures – code

```
# do once
Nbr <- 100
crossv <- t_normBinAge %>% crossv_mc(n = Nbr, test = 0.40)
perf <- data.frame(model = character(0), AUC = numeric(0), gini = numer
for(k in 1:Nbr) {
   perf <- rbind(perf,
                 get_perf(m2, data.frame(crossv$train[k]), 'train'),
                 get_perf(m2,  as.data.frame(crossv$test[k]),  'test'))
}
library(ggplot2)
p <- ggplot(perf, aes(x = AUC)) +
     geom_density(aes(fill = model), alpha = 0.5) +
     ggtitle('MC Cross Validated AUC')
p
```

# Monte Carlo Cross Validation – own measures – plot

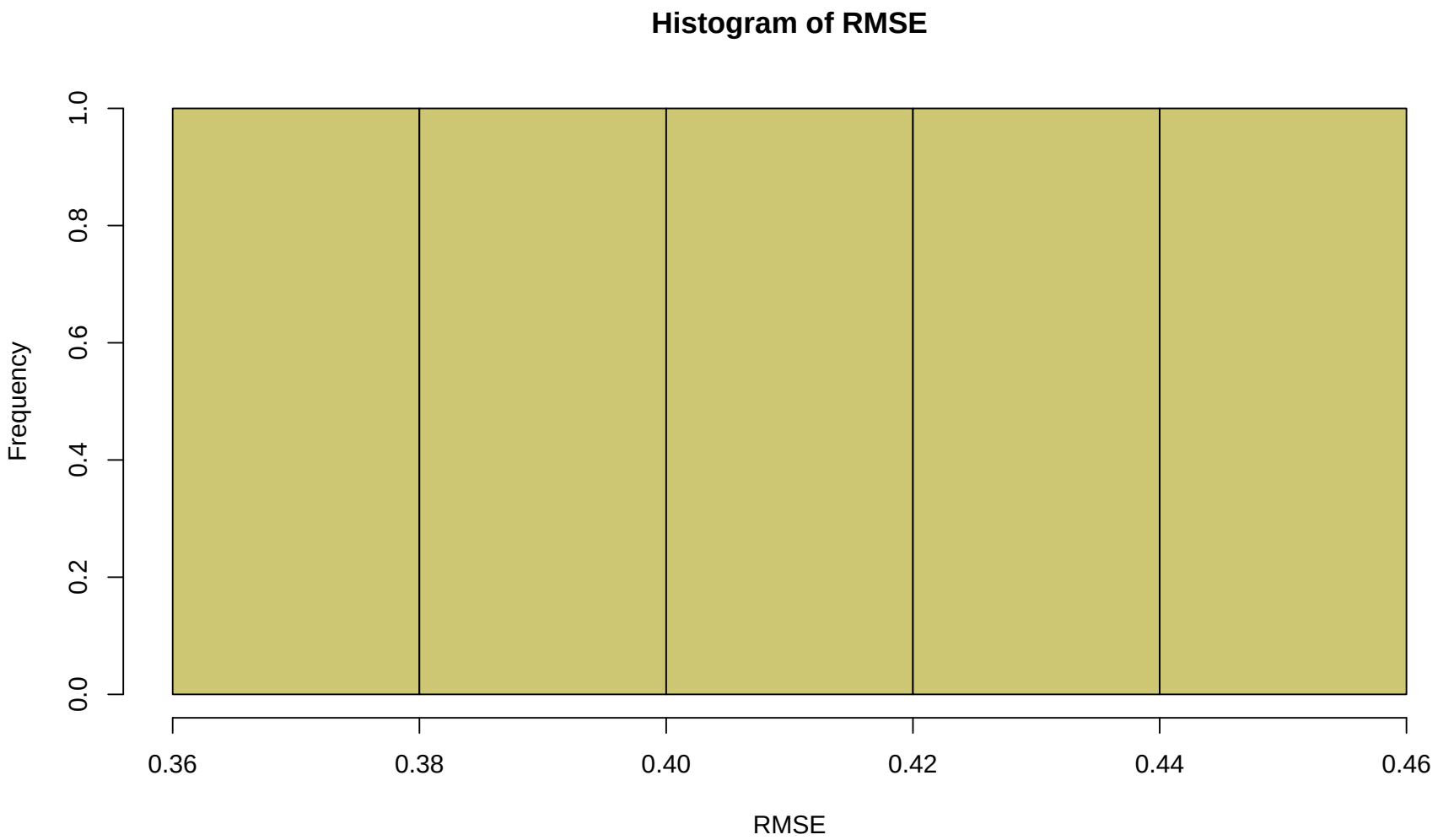


# K-fold Cross Validation

# k-fold Cross Validation – tidyverse – code

```
library(modelr)
library(magrittr) # to access the %T>% pipe
crossv <- t_normBinAge    %>%
          crossv_kfold(k = 5)
RMSE <- crossv %$%
        map(train,~ glm(frm2, data = .)) %>%
        map2_dbl(crossv$test, rmse) %T>%
        hist(col = "khaki3", main ="Histogram of RMSE", xlab = "RMSE")
```

# k-fold Cross Validation – tidyverse – plot



# K-fold Cross Validation – own measures – code

```
# do once
Nbr <- 10
crossv <- t_normBinAge %>% crossv_kfold(k = Nbr)
perf <- data.frame(model = character(0), AUC = numeric(0),
                   gini = numeric(0), ks = numeric(0))
for(k in 1:Nbr) {
  d_train <- data.frame(crossv$train[k])
  d_test <- as.data.frame(crossv$test[k])
  colnames(d_train) <- colnames(d_test) <- c('survived', 'class_1',
                          'class_2', 'male', 'age_youth', 'age_old')
 perf <- rbind(perf,
               get_perf(m2, d_train, 'train'),
               get_perf(m2,  d_test,  'test'))
}
library(ggplot2)
p <- ggplot(perf, aes(x = AUC)) +
     geom_density(aes(fill = model), alpha = 0.5) +
     ggtitle(paste0('k-fold Cross Validated AUC (k=',Nbr,')'))
p
```

# K-fold Cross Validation – own measures – plot