Predicting Hotel Booking Cancellations

Adeline Casali

2024-03-07

Data Preprocessing

# Loading Data  
data <- read.csv("project\_data.csv")  
  
# Loading packages  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(lubridate)

##   
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':  
##   
## date, intersect, setdiff, union

library(kableExtra)

##   
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':  
##   
## group\_rows

library(class)  
library(tree)  
library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

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

library(reticulate)  
library(tensorflow)

##   
## Attaching package: 'tensorflow'

## The following object is masked from 'package:caret':  
##   
## train

library(keras)  
library(MESS)  
  
  
# Explore the dataset  
na\_rows <- data[apply(is.na(data), 1, any), ]  
print(na\_rows)

## [1] Booking\_ID no\_of\_adults   
## [3] no\_of\_children no\_of\_weekend\_nights   
## [5] no\_of\_week\_nights type\_of\_meal\_plan   
## [7] required\_car\_parking\_space room\_type\_reserved   
## [9] lead\_time arrival\_date   
## [11] market\_segment\_type repeated\_guest   
## [13] no\_of\_previous\_cancellations no\_of\_previous\_bookings\_not\_canceled  
## [15] avg\_price\_per\_room no\_of\_special\_requests   
## [17] booking\_status   
## <0 rows> (or 0-length row.names)

unique(data$booking\_status)

## [1] "not\_canceled" "canceled"

# Remove unnecessary columns (Booking\_ID)  
data <- data[ , -1]  
  
# Convert booking\_status to 0 and 1  
data$booking\_status <- ifelse(data$booking\_status == "not\_canceled", 0, 1)  
  
# Calculate booking\_date based on arrival\_date and lead\_time  
data$arrival\_date <- as.Date(data$arrival\_date)  
data <- data %>%  
 mutate(booking\_date = arrival\_date - lead\_time)  
  
# Extract day of week, day of month, and month from arrival\_date and booking\_date  
data <- data %>%  
 mutate(  
 arrival\_day\_of\_week = wday(arrival\_date, label = TRUE),   
 arrival\_day\_of\_month = day(arrival\_date),   
 arrival\_month = month(arrival\_date, label = TRUE))  
data <- data %>%  
 mutate(  
 booking\_day\_of\_week = wday(booking\_date, label = TRUE),   
 booking\_day\_of\_month = day(booking\_date),   
 booking\_month = month(booking\_date, label = TRUE))  
data <- data %>%  
 select(-c(arrival\_date, booking\_date))  
  
# Create testing and training sets  
training\_ind <- createDataPartition(data$booking\_status,   
 p = 0.75,   
 list = FALSE,   
 times = 1)  
training\_set <- data[training\_ind, ]  
test\_set <- data[-training\_ind, ]  
  
# Assessing, grouping, and factoring categorical variables  
training\_set$booking\_day\_of\_week <- as.character(training\_set$booking\_day\_of\_week)  
training\_set$booking\_month <- as.character(training\_set$booking\_month)  
training\_set$arrival\_day\_of\_week <- as.character(training\_set$arrival\_day\_of\_week)  
training\_set$arrival\_month <- as.character(training\_set$arrival\_month)  
  
unique(training\_set$type\_of\_meal\_plan)

## [1] "meal\_plan\_1" "not\_selected" "meal\_plan\_2" "meal\_plan\_3"

unique(training\_set$room\_type\_reserved)

## [1] "room\_type1" "room\_type4" "room\_type2" "room\_type6" "room\_type5"  
## [6] "room\_type7" "room\_type3"

unique(training\_set$market\_segment\_type)

## [1] "offline" "online" "corporate" "aviation"   
## [5] "complementary"

unique(training\_set$booking\_day\_of\_week)

## [1] "Mon" "Thu" "Tue" "Sat" "Wed" "Sun" "Fri"

unique(training\_set$booking\_month)

## [1] "Feb" "Nov" "Oct" "Sep" "Mar" "Jul" "Jan" "May" "Aug" "Apr" "Dec" "Jun"

unique(training\_set$arrival\_day\_of\_week)

## [1] "Mon" "Tue" "Wed" "Sun" "Thu" "Fri" "Sat"

unique(training\_set$arrival\_month)

## [1] "Oct" "Nov" "Feb" "May" "Apr" "Sep" "Dec" "Jul" "Jun" "Jan" "Aug" "Mar"

training\_set$type\_of\_meal\_plan <- factor(training\_set$type\_of\_meal\_plan)  
training\_set$room\_type\_reserved <- factor(training\_set$room\_type\_reserved)  
training\_set$market\_segment\_type <- factor(training\_set$market\_segment\_type)  
training\_set$booking\_day\_of\_week <- factor(training\_set$booking\_day\_of\_week)  
training\_set$booking\_month <- factor(training\_set$booking\_month)  
training\_set$arrival\_day\_of\_week <- factor(training\_set$arrival\_day\_of\_week)  
training\_set$arrival\_month <- factor(training\_set$arrival\_month)  
  
class(training\_set$type\_of\_meal\_plan)

## [1] "factor"

class(training\_set$room\_type\_reserved)

## [1] "factor"

class(training\_set$market\_segment\_type)

## [1] "factor"

class(training\_set$booking\_day\_of\_week)

## [1] "factor"

class(training\_set$booking\_month)

## [1] "factor"

class(training\_set$arrival\_day\_of\_week)

## [1] "factor"

class(training\_set$arrival\_month)

## [1] "factor"

levels(training\_set$type\_of\_meal\_plan)

## [1] "meal\_plan\_1" "meal\_plan\_2" "meal\_plan\_3" "not\_selected"

levels(training\_set$room\_type\_reserved)

## [1] "room\_type1" "room\_type2" "room\_type3" "room\_type4" "room\_type5"  
## [6] "room\_type6" "room\_type7"

levels(training\_set$market\_segment\_type)

## [1] "aviation" "complementary" "corporate" "offline"   
## [5] "online"

levels(training\_set$booking\_day\_of\_week)

## [1] "Fri" "Mon" "Sat" "Sun" "Thu" "Tue" "Wed"

levels(training\_set$booking\_month)

## [1] "Apr" "Aug" "Dec" "Feb" "Jan" "Jul" "Jun" "Mar" "May" "Nov" "Oct" "Sep"

levels(training\_set$arrival\_day\_of\_week)

## [1] "Fri" "Mon" "Sat" "Sun" "Thu" "Tue" "Wed"

levels(training\_set$arrival\_month)

## [1] "Apr" "Aug" "Dec" "Feb" "Jan" "Jul" "Jun" "Mar" "May" "Nov" "Oct" "Sep"

# One-hot encoding the training set  
onehot\_encoder <- dummyVars(~ type\_of\_meal\_plan + room\_type\_reserved + market\_segment\_type + booking\_day\_of\_week + booking\_month + arrival\_day\_of\_week + arrival\_month,   
 training\_set[, c("type\_of\_meal\_plan", "room\_type\_reserved", "market\_segment\_type",   
 "booking\_day\_of\_week", "booking\_month", "arrival\_day\_of\_week", "arrival\_month")],   
 levelsOnly = FALSE,   
 fullRank = TRUE)  
  
onehot\_enc\_training <- predict(onehot\_encoder,   
 training\_set[, c("type\_of\_meal\_plan", "room\_type\_reserved", "market\_segment\_type",   
 "booking\_day\_of\_week", "booking\_month", "arrival\_day\_of\_week", "arrival\_month")])  
training\_set <- cbind(training\_set, onehot\_enc\_training)  
  
# One-hot encoding the test set  
test\_set$booking\_day\_of\_week <- as.character(test\_set$booking\_day\_of\_week)  
test\_set$booking\_month <- as.character(test\_set$booking\_month)  
test\_set$arrival\_day\_of\_week <- as.character(test\_set$arrival\_day\_of\_week)  
test\_set$arrival\_month <- as.character(test\_set$arrival\_month)  
  
test\_set$type\_of\_meal\_plan <- factor(test\_set$type\_of\_meal\_plan)  
test\_set$room\_type\_reserved <- factor(test\_set$room\_type\_reserved)  
test\_set$market\_segment\_type <- factor(test\_set$market\_segment\_type)  
test\_set$booking\_day\_of\_week <- factor(test\_set$booking\_day\_of\_week)  
test\_set$booking\_month <- factor(test\_set$booking\_month)  
test\_set$arrival\_day\_of\_week <- factor(test\_set$arrival\_day\_of\_week)  
test\_set$arrival\_month <- factor(test\_set$arrival\_month)  
  
onehot\_enc\_test <- predict(onehot\_encoder, test\_set[, c("type\_of\_meal\_plan", "room\_type\_reserved", "market\_segment\_type",   
 "booking\_day\_of\_week", "booking\_month", "arrival\_day\_of\_week", "arrival\_month")])  
test\_set <- cbind(test\_set, onehot\_enc\_test)  
  
# Scaling test and training sets  
test\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)] <- scale(test\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)],   
 center = apply(training\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)], 2, mean),   
 scale = apply(training\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)], 2, sd))  
training\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)] <- scale(training\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)])  
  
# Convert data sets to tensors  
training\_features <- array(data = unlist(training\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)]),   
 dim = c(nrow(training\_set), 42))  
training\_labels <- array(data = unlist(training\_set[, 15]),   
 dim = c(nrow(training\_set)))  
  
test\_features <- array(data = unlist(test\_set[, -c(5, 7, 9, 15, 16, 18, 19, 21)]),   
 dim = c(nrow(test\_set), 42))  
test\_labels <- array(data = unlist(test\_set[, 15]),   
 dim = c(nrow(test\_set)))  
  
# Remove unnecessary columns from training and test sets for use in linear models  
training\_set <- training\_set[ , -c(5, 7, 9, 16, 18, 19, 21)]  
test\_set <- test\_set[ , -c(5, 7, 9, 16, 18, 19, 21)]

Building and Evaluating Models

# Building and evaluating a logistic regression model  
# Model with all predictors  
lm <- glm(booking\_status ~ ., data = training\_set, family = binomial)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

summary(lm)

##   
## Call:  
## glm(formula = booking\_status ~ ., family = binomial, data = training\_set)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.489048 2.335181 -0.638 0.523696   
## no\_of\_adults 0.050666 0.019833 2.555 0.010628 \*   
## no\_of\_children 0.080890 0.023830 3.394 0.000688 \*\*\*  
## no\_of\_weekend\_nights 0.154021 0.023724 6.492 8.45e-11 \*\*\*  
## no\_of\_week\_nights 0.050303 0.020289 2.479 0.013162 \*   
## required\_car\_parking\_space -0.275469 0.023373 -11.786 < 2e-16 \*\*\*  
## lead\_time 1.501856 0.027439 54.734 < 2e-16 \*\*\*  
## repeated\_guest -0.404247 0.100361 -4.028 5.63e-05 \*\*\*  
## no\_of\_previous\_cancellations 0.136546 0.044029 3.101 0.001927 \*\*   
## no\_of\_previous\_bookings\_not\_canceled -0.158636 0.168477 -0.942 0.346404   
## avg\_price\_per\_room 0.709436 0.027730 25.584 < 2e-16 \*\*\*  
## no\_of\_special\_requests -1.206945 0.023601 -51.140 < 2e-16 \*\*\*  
## arrival\_day\_of\_month 0.023039 0.017204 1.339 0.180511   
## booking\_day\_of\_month 0.037147 0.017416 2.133 0.032934 \*   
## type\_of\_meal\_plan.meal\_plan\_2 0.064878 0.019610 3.308 0.000938 \*\*\*  
## type\_of\_meal\_plan.meal\_plan\_3 0.122341 1.651124 0.074 0.940934   
## type\_of\_meal\_plan.not\_selected 0.123869 0.018180 6.814 9.52e-12 \*\*\*  
## room\_type\_reserved.room\_type2 -0.052132 0.018084 -2.883 0.003942 \*\*   
## room\_type\_reserved.room\_type3 -0.002139 0.020200 -0.106 0.915681   
## room\_type\_reserved.room\_type4 -0.098392 0.019591 -5.022 5.11e-07 \*\*\*  
## room\_type\_reserved.room\_type5 -0.084373 0.017397 -4.850 1.24e-06 \*\*\*  
## room\_type\_reserved.room\_type6 -0.183506 0.024180 -7.589 3.22e-14 \*\*\*  
## room\_type\_reserved.room\_type7 -0.091116 0.019980 -4.560 5.11e-06 \*\*\*  
## market\_segment\_type.complementary -1.967496 22.366419 -0.088 0.929903   
## market\_segment\_type.corporate -0.238485 0.060270 -3.957 7.59e-05 \*\*\*  
## market\_segment\_type.offline -0.969578 0.113395 -8.550 < 2e-16 \*\*\*  
## market\_segment\_type.online -0.095581 0.118577 -0.806 0.420204   
## booking\_day\_of\_week.Mon -0.048847 0.022023 -2.218 0.026556 \*   
## booking\_day\_of\_week.Sat 0.069050 0.021825 3.164 0.001558 \*\*   
## booking\_day\_of\_week.Sun -0.046110 0.022273 -2.070 0.038431 \*   
## booking\_day\_of\_week.Thu -0.038876 0.023036 -1.688 0.091493 .   
## booking\_day\_of\_week.Tue -0.023171 0.020295 -1.142 0.253565   
## booking\_day\_of\_week.Wed 0.035106 0.021473 1.635 0.102074   
## booking\_month.Aug 0.043070 0.024244 1.777 0.075649 .   
## booking\_month.Dec -0.114298 0.023302 -4.905 9.34e-07 \*\*\*  
## booking\_month.Feb -0.053311 0.022571 -2.362 0.018181 \*   
## booking\_month.Jan -0.109580 0.024763 -4.425 9.64e-06 \*\*\*  
## booking\_month.Jul -0.035184 0.022222 -1.583 0.113346   
## booking\_month.Jun -0.058902 0.019802 -2.974 0.002935 \*\*   
## booking\_month.Mar -0.095390 0.020616 -4.627 3.71e-06 \*\*\*  
## booking\_month.May -0.004275 0.019867 -0.215 0.829632   
## booking\_month.Nov -0.076688 0.023988 -3.197 0.001389 \*\*   
## booking\_month.Oct -0.110763 0.024731 -4.479 7.51e-06 \*\*\*  
## booking\_month.Sep -0.137486 0.027179 -5.059 4.22e-07 \*\*\*  
## arrival\_day\_of\_week.Mon -0.076303 0.025988 -2.936 0.003324 \*\*   
## arrival\_day\_of\_week.Sat -0.084521 0.023639 -3.575 0.000350 \*\*\*  
## arrival\_day\_of\_week.Sun -0.041258 0.024818 -1.662 0.096421 .   
## arrival\_day\_of\_week.Thu 0.010178 0.022535 0.452 0.651525   
## arrival\_day\_of\_week.Tue -0.062773 0.028908 -2.171 0.029893 \*   
## arrival\_day\_of\_week.Wed -0.042917 0.025052 -1.713 0.086687 .   
## arrival\_month.Aug -0.141140 0.025062 -5.632 1.79e-08 \*\*\*  
## arrival\_month.Dec -0.471019 0.029972 -15.715 < 2e-16 \*\*\*  
## arrival\_month.Feb 0.157092 0.020009 7.851 4.12e-15 \*\*\*  
## arrival\_month.Jan -0.392838 0.043772 -8.975 < 2e-16 \*\*\*  
## arrival\_month.Jul -0.084029 0.022344 -3.761 0.000169 \*\*\*  
## arrival\_month.Jun -0.031110 0.022804 -1.364 0.172501   
## arrival\_month.Mar 0.090249 0.020511 4.400 1.08e-05 \*\*\*  
## arrival\_month.May -0.115280 0.021338 -5.403 6.57e-08 \*\*\*  
## arrival\_month.Nov 0.079235 0.026115 3.034 0.002413 \*\*   
## arrival\_month.Oct -0.079987 0.029619 -2.701 0.006924 \*\*   
## arrival\_month.Sep -0.179362 0.028493 -6.295 3.08e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 34286 on 27178 degrees of freedom  
## Residual deviance: 22122 on 27118 degrees of freedom  
## AIC: 22244  
##   
## Number of Fisher Scoring iterations: 16

predict\_lm <- predict(lm, newdata = test\_set)  
binary\_predict\_lm <- ifelse(predict\_lm > 0.5, 1, 0)  
results <- data.frame(  
 Actual = test\_set$booking\_status,   
 Predicted = binary\_predict\_lm  
)  
results$Correct <- results$Actual == results$Predicted  
confusion\_matrix\_lm <- table(Predicted = results$Predicted, Actual = results$Actual)  
print(confusion\_matrix\_lm)

## Actual  
## Predicted 0 1  
## 0 5631 1493  
## 1 389 1546

accuracy\_lm <- (5714 + 1568) / (5714 + 1395 + 382 + 1568)  
error\_lm <- 1 - accuracy\_lm  
cat("Accuracy:", accuracy\_lm, "\n")

## Accuracy: 0.8038415

cat("Error Rate:", error\_lm, "\n")

## Error Rate: 0.1961585

# Model with only significant predictors  
sig\_lm <- glm(booking\_status ~ no\_of\_adults + no\_of\_children + no\_of\_weekend\_nights + no\_of\_week\_nights + required\_car\_parking\_space + lead\_time + repeated\_guest + no\_of\_previous\_cancellations + avg\_price\_per\_room + no\_of\_special\_requests + arrival\_day\_of\_month + type\_of\_meal\_plan.meal\_plan\_2 + type\_of\_meal\_plan.not\_selected + room\_type\_reserved.room\_type2 + room\_type\_reserved.room\_type4 + room\_type\_reserved.room\_type5 + room\_type\_reserved.room\_type6 + room\_type\_reserved.room\_type7 + market\_segment\_type.corporate + market\_segment\_type.offline + booking\_day\_of\_week.Mon + booking\_day\_of\_week.Sat + booking\_month.Dec + booking\_month.Feb + booking\_month.Jan + booking\_month.Jul + booking\_month.Jun + booking\_month.Mar + booking\_month.Nov + booking\_month.Oct + booking\_month.Sep + arrival\_day\_of\_week.Mon + arrival\_day\_of\_week.Sat + arrival\_month.Aug + arrival\_month.Dec + arrival\_month.Feb + arrival\_month.Jan + arrival\_month.Jul + arrival\_month.Jun + arrival\_month.Mar + arrival\_month.May + arrival\_month.Nov + arrival\_month.Oct + arrival\_month.Sep,  
 data = training\_set, family = binomial)  
summary(sig\_lm)

##   
## Call:  
## glm(formula = booking\_status ~ no\_of\_adults + no\_of\_children +   
## no\_of\_weekend\_nights + no\_of\_week\_nights + required\_car\_parking\_space +   
## lead\_time + repeated\_guest + no\_of\_previous\_cancellations +   
## avg\_price\_per\_room + no\_of\_special\_requests + arrival\_day\_of\_month +   
## type\_of\_meal\_plan.meal\_plan\_2 + type\_of\_meal\_plan.not\_selected +   
## room\_type\_reserved.room\_type2 + room\_type\_reserved.room\_type4 +   
## room\_type\_reserved.room\_type5 + room\_type\_reserved.room\_type6 +   
## room\_type\_reserved.room\_type7 + market\_segment\_type.corporate +   
## market\_segment\_type.offline + booking\_day\_of\_week.Mon + booking\_day\_of\_week.Sat +   
## booking\_month.Dec + booking\_month.Feb + booking\_month.Jan +   
## booking\_month.Jul + booking\_month.Jun + booking\_month.Mar +   
## booking\_month.Nov + booking\_month.Oct + booking\_month.Sep +   
## arrival\_day\_of\_week.Mon + arrival\_day\_of\_week.Sat + arrival\_month.Aug +   
## arrival\_month.Dec + arrival\_month.Feb + arrival\_month.Jan +   
## arrival\_month.Jul + arrival\_month.Jun + arrival\_month.Mar +   
## arrival\_month.May + arrival\_month.Nov + arrival\_month.Oct +   
## arrival\_month.Sep, family = binomial, data = training\_set)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.28177 0.02429 -52.768 < 2e-16 \*\*\*  
## no\_of\_adults 0.04615 0.01956 2.359 0.018318 \*   
## no\_of\_children 0.07933 0.02384 3.327 0.000876 \*\*\*  
## no\_of\_weekend\_nights 0.11480 0.01783 6.437 1.22e-10 \*\*\*  
## no\_of\_week\_nights 0.08034 0.01764 4.555 5.25e-06 \*\*\*  
## required\_car\_parking\_space -0.27446 0.02334 -11.757 < 2e-16 \*\*\*  
## lead\_time 1.48913 0.02625 56.722 < 2e-16 \*\*\*  
## repeated\_guest -0.44543 0.09267 -4.807 1.54e-06 \*\*\*  
## no\_of\_previous\_cancellations 0.11437 0.03588 3.188 0.001433 \*\*   
## avg\_price\_per\_room 0.73065 0.02716 26.902 < 2e-16 \*\*\*  
## no\_of\_special\_requests -1.20336 0.02352 -51.173 < 2e-16 \*\*\*  
## arrival\_day\_of\_month 0.02480 0.01710 1.450 0.147012   
## type\_of\_meal\_plan.meal\_plan\_2 0.05159 0.01924 2.681 0.007346 \*\*   
## type\_of\_meal\_plan.not\_selected 0.12797 0.01812 7.064 1.62e-12 \*\*\*  
## room\_type\_reserved.room\_type2 -0.04989 0.01801 -2.770 0.005604 \*\*   
## room\_type\_reserved.room\_type4 -0.10308 0.01952 -5.281 1.29e-07 \*\*\*  
## room\_type\_reserved.room\_type5 -0.08650 0.01735 -4.986 6.17e-07 \*\*\*  
## room\_type\_reserved.room\_type6 -0.18831 0.02414 -7.801 6.13e-15 \*\*\*  
## room\_type\_reserved.room\_type7 -0.09470 0.02002 -4.730 2.25e-06 \*\*\*  
## market\_segment\_type.corporate -0.18569 0.02377 -7.812 5.62e-15 \*\*\*  
## market\_segment\_type.offline -0.86860 0.02434 -35.680 < 2e-16 \*\*\*  
## booking\_day\_of\_week.Mon -0.03357 0.01783 -1.882 0.059772 .   
## booking\_day\_of\_week.Sat 0.08489 0.01699 4.996 5.86e-07 \*\*\*  
## booking\_month.Dec -0.12038 0.02153 -5.591 2.26e-08 \*\*\*  
## booking\_month.Feb -0.06308 0.01981 -3.185 0.001449 \*\*   
## booking\_month.Jan -0.11273 0.02201 -5.121 3.04e-07 \*\*\*  
## booking\_month.Jul -0.05526 0.01896 -2.914 0.003563 \*\*   
## booking\_month.Jun -0.06644 0.01774 -3.745 0.000180 \*\*\*  
## booking\_month.Mar -0.10163 0.01811 -5.613 1.99e-08 \*\*\*  
## booking\_month.Nov -0.08862 0.02212 -4.007 6.16e-05 \*\*\*  
## booking\_month.Oct -0.12463 0.02199 -5.668 1.44e-08 \*\*\*  
## booking\_month.Sep -0.15889 0.02341 -6.787 1.15e-11 \*\*\*  
## arrival\_day\_of\_week.Mon -0.04090 0.01734 -2.359 0.018325 \*   
## arrival\_day\_of\_week.Sat -0.06506 0.01806 -3.602 0.000316 \*\*\*  
## arrival\_month.Aug -0.13851 0.02453 -5.646 1.64e-08 \*\*\*  
## arrival\_month.Dec -0.46453 0.02955 -15.719 < 2e-16 \*\*\*  
## arrival\_month.Feb 0.14597 0.01980 7.373 1.66e-13 \*\*\*  
## arrival\_month.Jan -0.38618 0.04332 -8.915 < 2e-16 \*\*\*  
## arrival\_month.Jul -0.08699 0.02214 -3.929 8.53e-05 \*\*\*  
## arrival\_month.Jun -0.04064 0.02252 -1.805 0.071151 .   
## arrival\_month.Mar 0.08756 0.02044 4.283 1.84e-05 \*\*\*  
## arrival\_month.May -0.12051 0.02114 -5.701 1.19e-08 \*\*\*  
## arrival\_month.Nov 0.08671 0.02549 3.402 0.000670 \*\*\*  
## arrival\_month.Oct -0.07525 0.02880 -2.613 0.008987 \*\*   
## arrival\_month.Sep -0.17368 0.02745 -6.328 2.48e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 34286 on 27178 degrees of freedom  
## Residual deviance: 22187 on 27134 degrees of freedom  
## AIC: 22277  
##   
## Number of Fisher Scoring iterations: 7

predict\_sig\_lm <- predict(sig\_lm, newdata = test\_set)  
binary\_predict\_sig\_lm <- ifelse(predict\_sig\_lm > 0.5, 1, 0)  
results\_sig <- data.frame(  
 Actual = test\_set$booking\_status,   
 Predicted = binary\_predict\_sig\_lm  
)  
results\_sig$Correct <- results\_sig$Actual == results\_sig$Predicted  
confusion\_matrix\_sig\_lm <- table(Predicted = results\_sig$Predicted, Actual = results\_sig$Actual)  
print(confusion\_matrix\_sig\_lm)

## Actual  
## Predicted 0 1  
## 0 5634 1481  
## 1 386 1558

accuracy\_sig\_lm <- (5712 + 1568) / (5712 + 1395 + 384 + 1568)  
error\_sig\_lm <- 1 - accuracy\_sig\_lm  
cat("Accuracy:", accuracy\_sig\_lm, "\n")

## Accuracy: 0.8036207

cat("Error Rate:", error\_sig\_lm, "\n")

## Error Rate: 0.1963793

# Building and evaluating a K-Nearest Neighbors (KNN) model  
# Model with all predictors and K = 3  
predictors <- training\_set[, -which(names(training\_set) == "booking\_status")]  
label <- training\_set$booking\_status  
k <- 3  
knn\_model <- knn(train = predictors, test = predictors, cl = label, k = k)  
knn\_predictions <- knn(  
 train = training\_set[, -length(predictors)],  
 test = test\_set[, -length(predictors)],  
 cl = training\_set$booking\_status,  
 k = k  
)  
knn\_results <- data.frame(  
 Actual = test\_set$booking\_status,  
 Predicted = knn\_predictions  
)  
knn\_results$Correct <- knn\_results$Actual == knn\_results$Predicted  
knn\_confusion\_matrix <- table(Predicted = knn\_results$Predicted, Actual = knn\_results$Actual)  
print(knn\_confusion\_matrix)

## Actual  
## Predicted 0 1  
## 0 5430 903  
## 1 590 2136

accuracy\_knn <- (5519 + 2114) / (5519 + 849 + 577 + 2114)  
error\_knn <- 1 - accuracy\_knn  
cat("Accuracy:", accuracy\_knn, "\n")

## Accuracy: 0.8425875

cat("Error Rate:", error\_knn, "\n")

## Error Rate: 0.1574125

# Model with all predictors and K = 5  
k <- 5  
knn\_model <- knn(train = predictors, test = predictors, cl = label, k = k)  
knn\_predictions <- knn(  
 train = training\_set[, -length(predictors)],  
 test = test\_set[, -length(predictors)],  
 cl = training\_set$booking\_status,  
 k = k  
)  
knn\_results <- data.frame(  
 Actual = test\_set$booking\_status,  
 Predicted = knn\_predictions  
)  
knn\_results$Correct <- knn\_results$Actual == knn\_results$Predicted  
knn\_confusion\_matrix <- table(Predicted = knn\_results$Predicted, Actual = knn\_results$Actual)  
print(knn\_confusion\_matrix)

## Actual  
## Predicted 0 1  
## 0 5484 972  
## 1 536 2067

accuracy\_knn <- (5541 + 2087) / (5541 + 876 + 555 + 2087)  
error\_knn <- 1 - accuracy\_knn  
cat("Accuracy:", accuracy\_knn, "\n")

## Accuracy: 0.8420355

cat("Error Rate:", error\_knn, "\n")

## Error Rate: 0.1579645

# Model with all predictors and K = 10  
k <- 10  
knn\_model <- knn(train = predictors, test = predictors, cl = label, k = k)  
knn\_predictions <- knn(  
 train = training\_set[, -length(predictors)],  
 test = test\_set[, -length(predictors)],  
 cl = training\_set$booking\_status,  
 k = k  
)  
knn\_results <- data.frame(  
 Actual = test\_set$booking\_status,  
 Predicted = knn\_predictions  
)  
knn\_results$Correct <- knn\_results$Actual == knn\_results$Predicted  
knn\_confusion\_matrix <- table(Predicted = knn\_results$Predicted, Actual = knn\_results$Actual)  
print(knn\_confusion\_matrix)

## Actual  
## Predicted 0 1  
## 0 5530 1030  
## 1 490 2009

accuracy\_knn <- (5629 + 2001) / (5629 + 962 + 467 + 2001)  
error\_knn <- 1 - accuracy\_knn  
cat("Accuracy:", accuracy\_knn, "\n")

## Accuracy: 0.8422563

cat("Error Rate:", error\_knn, "\n")

## Error Rate: 0.1577437

# Building and evaluating a classification tree model  
set.seed(123)  
rf <- randomForest(booking\_status ~ ., data = training\_set, mtry = 4, importance = TRUE, ntree = 25, type = "classification")

## Warning in randomForest.default(m, y, ...): The response has five or fewer  
## unique values. Are you sure you want to do regression?

rf

##   
## Call:  
## randomForest(formula = booking\_status ~ ., data = training\_set, mtry = 4, importance = TRUE, ntree = 25, type = "classification")   
## Type of random forest: regression  
## Number of trees: 25  
## No. of variables tried at each split: 4  
##   
## Mean of squared residuals: 0.09739041  
## % Var explained: 55.62

importance(rf)

## %IncMSE IncNodePurity  
## no\_of\_adults 7.927026 73.0306269  
## no\_of\_children 6.234115 24.8052648  
## no\_of\_weekend\_nights 10.787022 99.7151491  
## no\_of\_week\_nights 8.184137 124.9780811  
## required\_car\_parking\_space 12.475792 24.6791120  
## lead\_time 19.002833 928.4891290  
## repeated\_guest 3.510462 15.9353678  
## no\_of\_previous\_cancellations 2.165194 3.2312633  
## no\_of\_previous\_bookings\_not\_canceled 3.718757 7.7746705  
## avg\_price\_per\_room 16.032148 375.2911457  
## no\_of\_special\_requests 17.669759 460.5682070  
## arrival\_day\_of\_month 10.989718 175.3078116  
## booking\_day\_of\_month 12.724026 201.7439848  
## type\_of\_meal\_plan.meal\_plan\_2 7.429973 48.8137128  
## type\_of\_meal\_plan.meal\_plan\_3 0.000000 0.1551703  
## type\_of\_meal\_plan.not\_selected 7.260338 31.7092582  
## room\_type\_reserved.room\_type2 3.949542 9.4425248  
## room\_type\_reserved.room\_type3 0.000000 0.2568388  
## room\_type\_reserved.room\_type4 6.020215 31.4044777  
## room\_type\_reserved.room\_type5 2.897606 5.3616064  
## room\_type\_reserved.room\_type6 7.669748 9.7664599  
## room\_type\_reserved.room\_type7 0.976384 1.8854604  
## market\_segment\_type.complementary 3.628969 7.1977798  
## market\_segment\_type.corporate 4.617043 21.1978926  
## market\_segment\_type.offline 6.928862 60.5027639  
## market\_segment\_type.online 8.211171 103.1254348  
## booking\_day\_of\_week.Mon 5.489808 29.0469587  
## booking\_day\_of\_week.Sat 8.248485 30.4602893  
## booking\_day\_of\_week.Sun 6.012867 27.5932136  
## booking\_day\_of\_week.Thu 6.527494 35.2018976  
## booking\_day\_of\_week.Tue 5.293054 23.5523321  
## booking\_day\_of\_week.Wed 9.598472 30.6898992  
## booking\_month.Aug 5.802052 29.0859907  
## booking\_month.Dec 7.764677 32.8375933  
## booking\_month.Feb 5.068197 31.2888602  
## booking\_month.Jan 6.002872 40.1776453  
## booking\_month.Jul 4.652396 25.3890705  
## booking\_month.Jun 5.187513 21.2151034  
## booking\_month.Mar 5.226285 23.6104222  
## booking\_month.May 6.170927 19.2687963  
## booking\_month.Nov 7.523172 22.5429196  
## booking\_month.Oct 6.786589 32.3883663  
## booking\_month.Sep 7.683706 60.7713143  
## arrival\_day\_of\_week.Mon 10.499418 31.2626992  
## arrival\_day\_of\_week.Sat 8.764416 28.0808517  
## arrival\_day\_of\_week.Sun 8.556843 32.5577393  
## arrival\_day\_of\_week.Thu 6.073320 26.2303183  
## arrival\_day\_of\_week.Tue 9.228698 28.1383910  
## arrival\_day\_of\_week.Wed 6.971482 30.5561734  
## arrival\_month.Aug 7.638099 28.5015399  
## arrival\_month.Dec 9.843865 66.6186510  
## arrival\_month.Feb 7.091746 18.7029076  
## arrival\_month.Jan 7.144443 31.6528026  
## arrival\_month.Jul 6.584324 31.2356620  
## arrival\_month.Jun 6.046398 28.1944896  
## arrival\_month.Mar 7.207560 17.7911175  
## arrival\_month.May 7.098190 27.0473554  
## arrival\_month.Nov 5.885317 29.4703988  
## arrival\_month.Oct 9.355388 33.7814847  
## arrival\_month.Sep 8.845893 29.6208384

rf\_predictions <- predict(rf, test\_set, type = "class")  
rf\_results <- data.frame(  
 Actual = test\_set$booking\_status,  
 Predicted = rf\_predictions  
)  
rf\_predictions <- factor(ifelse(rf\_predictions >= 0.5, 1, 0))  
test\_set$booking\_status <- factor(test\_set$booking\_status)  
levels(rf\_predictions) <- levels(test\_set$booking\_status)  
confusion\_mat <- confusionMatrix(rf\_predictions, test\_set$booking\_status)  
print(confusion\_mat)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 5738 830  
## 1 282 2209  
##   
## Accuracy : 0.8772   
## 95% CI : (0.8703, 0.8839)  
## No Information Rate : 0.6645   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7118   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9532   
## Specificity : 0.7269   
## Pos Pred Value : 0.8736   
## Neg Pred Value : 0.8868   
## Prevalence : 0.6645   
## Detection Rate : 0.6334   
## Detection Prevalence : 0.7250   
## Balanced Accuracy : 0.8400   
##   
## 'Positive' Class : 0   
##

accuracy\_tree\_rf <- (5893 + 2112) / (5893 + 804 + 250 + 2112)  
error\_tree\_rf <- 1 - accuracy\_tree\_rf  
cat("Accuracy:", accuracy\_tree\_rf, "\n")

## Accuracy: 0.8836516

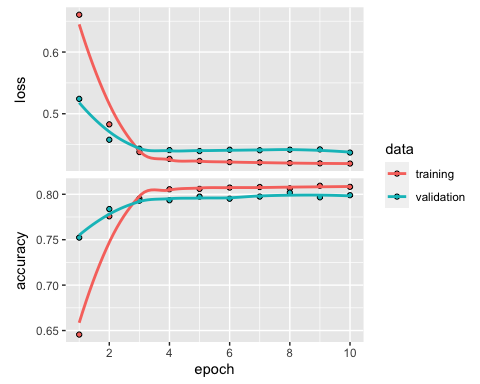
cat("Error Rate:", error\_tree\_rf, "\n")

## Error Rate: 0.1163484

# Building and evaluating a neural network model  
model <- keras\_model\_sequential(list(  
 layer\_dense(units = 40, activation = "relu"),   
 layer\_dense(units = 20, activation = "relu"),  
 layer\_dense(units = 1, activation = "sigmoid")  
))  
compile(model,   
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = "accuracy")  
  
# Training the model  
history <- fit(model, training\_features, training\_labels,   
 epochs = 10, batch\_size = 512, validation\_split = 0.33)

## Epoch 1/10  
## 36/36 - 2s - loss: 0.6605 - accuracy: 0.6456 - val\_loss: 0.5241 - val\_accuracy: 0.7523 - 2s/epoch - 50ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.4828 - accuracy: 0.7758 - val\_loss: 0.4578 - val\_accuracy: 0.7837 - 401ms/epoch - 11ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.4380 - accuracy: 0.7952 - val\_loss: 0.4433 - val\_accuracy: 0.7930 - 390ms/epoch - 11ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.4269 - accuracy: 0.8055 - val\_loss: 0.4411 - val\_accuracy: 0.7934 - 388ms/epoch - 11ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.4235 - accuracy: 0.8059 - val\_loss: 0.4395 - val\_accuracy: 0.7974 - 391ms/epoch - 11ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.4217 - accuracy: 0.8073 - val\_loss: 0.4416 - val\_accuracy: 0.7951 - 387ms/epoch - 11ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.4210 - accuracy: 0.8080 - val\_loss: 0.4408 - val\_accuracy: 0.7974 - 389ms/epoch - 11ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.4200 - accuracy: 0.8065 - val\_loss: 0.4417 - val\_accuracy: 0.8019 - 387ms/epoch - 11ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.4196 - accuracy: 0.8094 - val\_loss: 0.4422 - val\_accuracy: 0.7967 - 390ms/epoch - 11ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.4192 - accuracy: 0.8082 - val\_loss: 0.4370 - val\_accuracy: 0.7990 - 386ms/epoch - 11ms/step

plot(history)



# Using the model to make predictions  
predictions <- predict(model, test\_features)

## 284/284 - 0s - 423ms/epoch - 1ms/step

test\_set$p\_prob <- predictions[, 1]  
head(predictions, 10)

## [,1]  
## [1,] 0.07948324  
## [2,] 0.03747618  
## [3,] 0.10787603  
## [4,] 0.04403732  
## [5,] 0.52117199  
## [6,] 0.09139957  
## [7,] 0.04189007  
## [8,] 0.95274347  
## [9,] 0.03495034  
## [10,] 0.02473943

predicted\_class <- (predictions[, 1] >= 0.5) \* 1  
head(predicted\_class, 10)

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

# Calculating accuracy  
accuracy <- mean(predicted\_class == test\_labels)  
accuracy

## [1] 0.8001987

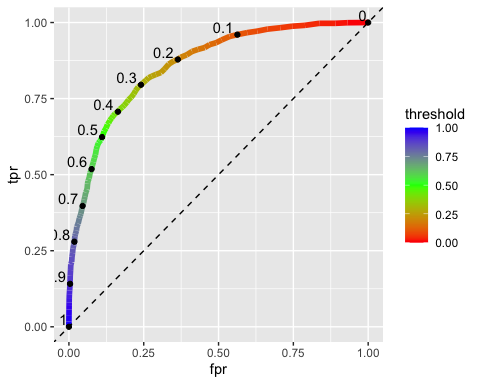
# Making predictions and calculating fpr and tpr rates at 0.5 threshold  
over\_threshold <- test\_set[test\_set$p\_prob >= 0.5, ]  
fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
fpr

## [1] 0.1106312

tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
tpr

## [1] 0.6235604

# Plotting ROC curve  
roc\_data <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)  
for (i in roc\_data$threshold) {  
 over\_threshold <- test\_set[test\_set$p\_prob >= i, ]  
 fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
 roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
 tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
 roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +   
 geom\_line(data = roc\_data,   
 aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +   
 scale\_color\_gradientn(colors = rainbow(3)) +   
 geom\_abline(intercept = 0, slope = 1, lty = 2) +   
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +   
 geom\_text(data = roc\_data[seq(1, 101, 10), ],   
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# Calculating the AUC  
auc <- auc(x = roc\_data$fpr, y = roc\_data$tpr, type = "spline")

## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique  
## 'x' values

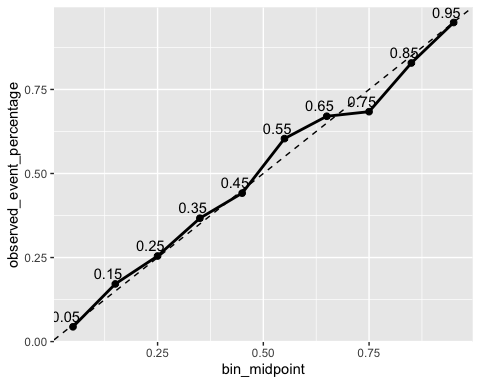
auc

## [1] 0.8594633

# Creating a calibration curve  
in\_interval <- test\_set[test\_set$p\_prob >= 0.7 & test\_set$p\_prob <= 0.8, ]  
nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)

## [1] 0.683908

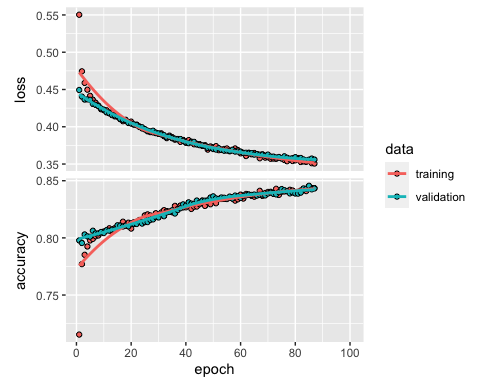
calibration\_data <- data.frame(bin\_midpoint=seq(0.05,0.95,0.1),  
 observed\_event\_percentage=0)  
for (i in seq(0.05,0.95,0.1)) {  
 in\_interval <- test\_set[test\_set$p\_prob >= (i-0.05) & test\_set$p\_prob <= (i+0.05), ]  
 oep <- nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)  
 calibration\_data[calibration\_data$bin\_midpoint==i, "observed\_event\_percentage"] <- oep  
}  
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
 geom\_line(linewidth = 1) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(size = 2) +  
 geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)



# Building another neural network model  
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 80, activation = "tanh") %>%  
 layer\_dropout(rate = 0.3) %>%   
 layer\_dense(units = 40, activation = "tanh") %>%  
 layer\_dropout(rate = 0.2) %>%   
 layer\_dense(units = 20, activation = "tanh") %>%  
 layer\_dropout(rate = 0.2) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
# Compile the model  
compile(model,   
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = "accuracy")  
  
# Define early stopping callback  
early\_stop <- callback\_early\_stopping(  
 monitor = "val\_loss",  
 patience = 3  
)  
  
# Training the model with early stopping  
history <- fit(  
 model,  
 training\_features,  
 training\_labels,  
 epochs = 100,  
 batch\_size = 512,  
 validation\_split = 0.33,  
 callbacks = list(early\_stop)  
)

## Epoch 1/100  
## 36/36 - 2s - loss: 0.5503 - accuracy: 0.7154 - val\_loss: 0.4493 - val\_accuracy: 0.7978 - 2s/epoch - 60ms/step  
## Epoch 2/100  
## 36/36 - 1s - loss: 0.4744 - accuracy: 0.7770 - val\_loss: 0.4404 - val\_accuracy: 0.7955 - 517ms/epoch - 14ms/step  
## Epoch 3/100  
## 36/36 - 1s - loss: 0.4589 - accuracy: 0.7852 - val\_loss: 0.4365 - val\_accuracy: 0.8030 - 505ms/epoch - 14ms/step  
## Epoch 4/100  
## 36/36 - 1s - loss: 0.4498 - accuracy: 0.7925 - val\_loss: 0.4367 - val\_accuracy: 0.8013 - 503ms/epoch - 14ms/step  
## Epoch 5/100  
## 36/36 - 1s - loss: 0.4415 - accuracy: 0.7977 - val\_loss: 0.4361 - val\_accuracy: 0.8010 - 503ms/epoch - 14ms/step  
## Epoch 6/100  
## 36/36 - 1s - loss: 0.4362 - accuracy: 0.7991 - val\_loss: 0.4307 - val\_accuracy: 0.8062 - 505ms/epoch - 14ms/step  
## Epoch 7/100  
## 36/36 - 1s - loss: 0.4323 - accuracy: 0.8014 - val\_loss: 0.4294 - val\_accuracy: 0.8039 - 503ms/epoch - 14ms/step  
## Epoch 8/100  
## 36/36 - 1s - loss: 0.4285 - accuracy: 0.8021 - val\_loss: 0.4276 - val\_accuracy: 0.8039 - 504ms/epoch - 14ms/step  
## Epoch 9/100  
## 36/36 - 1s - loss: 0.4235 - accuracy: 0.8047 - val\_loss: 0.4246 - val\_accuracy: 0.8051 - 504ms/epoch - 14ms/step  
## Epoch 10/100  
## 36/36 - 1s - loss: 0.4223 - accuracy: 0.8056 - val\_loss: 0.4230 - val\_accuracy: 0.8042 - 600ms/epoch - 17ms/step  
## Epoch 11/100  
## 36/36 - 1s - loss: 0.4221 - accuracy: 0.8065 - val\_loss: 0.4193 - val\_accuracy: 0.8067 - 502ms/epoch - 14ms/step  
## Epoch 12/100  
## 36/36 - 1s - loss: 0.4191 - accuracy: 0.8084 - val\_loss: 0.4180 - val\_accuracy: 0.8064 - 502ms/epoch - 14ms/step  
## Epoch 13/100  
## 36/36 - 1s - loss: 0.4193 - accuracy: 0.8053 - val\_loss: 0.4157 - val\_accuracy: 0.8071 - 503ms/epoch - 14ms/step  
## Epoch 14/100  
## 36/36 - 0s - loss: 0.4140 - accuracy: 0.8085 - val\_loss: 0.4149 - val\_accuracy: 0.8069 - 499ms/epoch - 14ms/step  
## Epoch 15/100  
## 36/36 - 1s - loss: 0.4135 - accuracy: 0.8098 - val\_loss: 0.4142 - val\_accuracy: 0.8110 - 500ms/epoch - 14ms/step  
## Epoch 16/100  
## 36/36 - 1s - loss: 0.4102 - accuracy: 0.8108 - val\_loss: 0.4106 - val\_accuracy: 0.8107 - 508ms/epoch - 14ms/step  
## Epoch 17/100  
## 36/36 - 1s - loss: 0.4075 - accuracy: 0.8142 - val\_loss: 0.4091 - val\_accuracy: 0.8099 - 508ms/epoch - 14ms/step  
## Epoch 18/100  
## 36/36 - 1s - loss: 0.4081 - accuracy: 0.8108 - val\_loss: 0.4093 - val\_accuracy: 0.8107 - 515ms/epoch - 14ms/step  
## Epoch 19/100  
## 36/36 - 1s - loss: 0.4055 - accuracy: 0.8133 - val\_loss: 0.4051 - val\_accuracy: 0.8100 - 508ms/epoch - 14ms/step  
## Epoch 20/100  
## 36/36 - 1s - loss: 0.4069 - accuracy: 0.8081 - val\_loss: 0.4043 - val\_accuracy: 0.8127 - 508ms/epoch - 14ms/step  
## Epoch 21/100  
## 36/36 - 1s - loss: 0.4029 - accuracy: 0.8137 - val\_loss: 0.4031 - val\_accuracy: 0.8122 - 507ms/epoch - 14ms/step  
## Epoch 22/100  
## 36/36 - 1s - loss: 0.4029 - accuracy: 0.8140 - val\_loss: 0.4021 - val\_accuracy: 0.8122 - 506ms/epoch - 14ms/step  
## Epoch 23/100  
## 36/36 - 1s - loss: 0.4005 - accuracy: 0.8156 - val\_loss: 0.3998 - val\_accuracy: 0.8142 - 506ms/epoch - 14ms/step  
## Epoch 24/100  
## 36/36 - 1s - loss: 0.3977 - accuracy: 0.8135 - val\_loss: 0.3993 - val\_accuracy: 0.8125 - 512ms/epoch - 14ms/step  
## Epoch 25/100  
## 36/36 - 1s - loss: 0.3965 - accuracy: 0.8197 - val\_loss: 0.3968 - val\_accuracy: 0.8138 - 510ms/epoch - 14ms/step  
## Epoch 26/100  
## 36/36 - 1s - loss: 0.3939 - accuracy: 0.8206 - val\_loss: 0.3947 - val\_accuracy: 0.8138 - 506ms/epoch - 14ms/step  
## Epoch 27/100  
## 36/36 - 1s - loss: 0.3928 - accuracy: 0.8196 - val\_loss: 0.3938 - val\_accuracy: 0.8157 - 507ms/epoch - 14ms/step  
## Epoch 28/100  
## 36/36 - 1s - loss: 0.3924 - accuracy: 0.8181 - val\_loss: 0.3947 - val\_accuracy: 0.8162 - 508ms/epoch - 14ms/step  
## Epoch 29/100  
## 36/36 - 1s - loss: 0.3915 - accuracy: 0.8216 - val\_loss: 0.3921 - val\_accuracy: 0.8183 - 507ms/epoch - 14ms/step  
## Epoch 30/100  
## 36/36 - 1s - loss: 0.3905 - accuracy: 0.8197 - val\_loss: 0.3912 - val\_accuracy: 0.8178 - 580ms/epoch - 16ms/step  
## Epoch 31/100  
## 36/36 - 1s - loss: 0.3912 - accuracy: 0.8222 - val\_loss: 0.3898 - val\_accuracy: 0.8206 - 502ms/epoch - 14ms/step  
## Epoch 32/100  
## 36/36 - 1s - loss: 0.3874 - accuracy: 0.8243 - val\_loss: 0.3888 - val\_accuracy: 0.8190 - 501ms/epoch - 14ms/step  
## Epoch 33/100  
## 36/36 - 1s - loss: 0.3848 - accuracy: 0.8234 - val\_loss: 0.3860 - val\_accuracy: 0.8223 - 503ms/epoch - 14ms/step  
## Epoch 34/100  
## 36/36 - 1s - loss: 0.3881 - accuracy: 0.8229 - val\_loss: 0.3866 - val\_accuracy: 0.8230 - 503ms/epoch - 14ms/step  
## Epoch 35/100  
## 36/36 - 1s - loss: 0.3860 - accuracy: 0.8225 - val\_loss: 0.3846 - val\_accuracy: 0.8242 - 506ms/epoch - 14ms/step  
## Epoch 36/100  
## 36/36 - 1s - loss: 0.3827 - accuracy: 0.8280 - val\_loss: 0.3840 - val\_accuracy: 0.8213 - 516ms/epoch - 14ms/step  
## Epoch 37/100  
## 36/36 - 1s - loss: 0.3836 - accuracy: 0.8254 - val\_loss: 0.3832 - val\_accuracy: 0.8243 - 506ms/epoch - 14ms/step  
## Epoch 38/100  
## 36/36 - 1s - loss: 0.3797 - accuracy: 0.8283 - val\_loss: 0.3835 - val\_accuracy: 0.8260 - 521ms/epoch - 14ms/step  
## Epoch 39/100  
## 36/36 - 1s - loss: 0.3823 - accuracy: 0.8266 - val\_loss: 0.3811 - val\_accuracy: 0.8293 - 535ms/epoch - 15ms/step  
## Epoch 40/100  
## 36/36 - 1s - loss: 0.3785 - accuracy: 0.8267 - val\_loss: 0.3792 - val\_accuracy: 0.8282 - 591ms/epoch - 16ms/step  
## Epoch 41/100  
## 36/36 - 1s - loss: 0.3821 - accuracy: 0.8287 - val\_loss: 0.3780 - val\_accuracy: 0.8311 - 605ms/epoch - 17ms/step  
## Epoch 42/100  
## 36/36 - 1s - loss: 0.3800 - accuracy: 0.8253 - val\_loss: 0.3793 - val\_accuracy: 0.8300 - 601ms/epoch - 17ms/step  
## Epoch 43/100  
## 36/36 - 1s - loss: 0.3765 - accuracy: 0.8272 - val\_loss: 0.3797 - val\_accuracy: 0.8298 - 601ms/epoch - 17ms/step  
## Epoch 44/100  
## 36/36 - 1s - loss: 0.3770 - accuracy: 0.8274 - val\_loss: 0.3762 - val\_accuracy: 0.8327 - 593ms/epoch - 16ms/step  
## Epoch 45/100  
## 36/36 - 1s - loss: 0.3768 - accuracy: 0.8294 - val\_loss: 0.3753 - val\_accuracy: 0.8309 - 597ms/epoch - 17ms/step  
## Epoch 46/100  
## 36/36 - 1s - loss: 0.3744 - accuracy: 0.8327 - val\_loss: 0.3751 - val\_accuracy: 0.8324 - 591ms/epoch - 16ms/step  
## Epoch 47/100  
## 36/36 - 1s - loss: 0.3753 - accuracy: 0.8279 - val\_loss: 0.3741 - val\_accuracy: 0.8326 - 603ms/epoch - 17ms/step  
## Epoch 48/100  
## 36/36 - 1s - loss: 0.3693 - accuracy: 0.8311 - val\_loss: 0.3716 - val\_accuracy: 0.8329 - 594ms/epoch - 17ms/step  
## Epoch 49/100  
## 36/36 - 1s - loss: 0.3736 - accuracy: 0.8294 - val\_loss: 0.3718 - val\_accuracy: 0.8353 - 523ms/epoch - 15ms/step  
## Epoch 50/100  
## 36/36 - 1s - loss: 0.3718 - accuracy: 0.8345 - val\_loss: 0.3702 - val\_accuracy: 0.8363 - 506ms/epoch - 14ms/step  
## Epoch 51/100  
## 36/36 - 1s - loss: 0.3741 - accuracy: 0.8303 - val\_loss: 0.3694 - val\_accuracy: 0.8346 - 510ms/epoch - 14ms/step  
## Epoch 52/100  
## 36/36 - 1s - loss: 0.3686 - accuracy: 0.8339 - val\_loss: 0.3713 - val\_accuracy: 0.8341 - 509ms/epoch - 14ms/step  
## Epoch 53/100  
## 36/36 - 1s - loss: 0.3694 - accuracy: 0.8340 - val\_loss: 0.3689 - val\_accuracy: 0.8353 - 517ms/epoch - 14ms/step  
## Epoch 54/100  
## 36/36 - 1s - loss: 0.3703 - accuracy: 0.8341 - val\_loss: 0.3678 - val\_accuracy: 0.8351 - 508ms/epoch - 14ms/step  
## Epoch 55/100  
## 36/36 - 1s - loss: 0.3709 - accuracy: 0.8340 - val\_loss: 0.3681 - val\_accuracy: 0.8343 - 517ms/epoch - 14ms/step  
## Epoch 56/100  
## 36/36 - 1s - loss: 0.3671 - accuracy: 0.8353 - val\_loss: 0.3677 - val\_accuracy: 0.8361 - 528ms/epoch - 15ms/step  
## Epoch 57/100  
## 36/36 - 1s - loss: 0.3682 - accuracy: 0.8345 - val\_loss: 0.3691 - val\_accuracy: 0.8343 - 536ms/epoch - 15ms/step  
## Epoch 58/100  
## 36/36 - 1s - loss: 0.3710 - accuracy: 0.8326 - val\_loss: 0.3667 - val\_accuracy: 0.8370 - 514ms/epoch - 14ms/step  
## Epoch 59/100  
## 36/36 - 1s - loss: 0.3687 - accuracy: 0.8338 - val\_loss: 0.3669 - val\_accuracy: 0.8363 - 514ms/epoch - 14ms/step  
## Epoch 60/100  
## 36/36 - 1s - loss: 0.3643 - accuracy: 0.8378 - val\_loss: 0.3667 - val\_accuracy: 0.8349 - 519ms/epoch - 14ms/step  
## Epoch 61/100  
## 36/36 - 1s - loss: 0.3651 - accuracy: 0.8344 - val\_loss: 0.3648 - val\_accuracy: 0.8366 - 512ms/epoch - 14ms/step  
## Epoch 62/100  
## 36/36 - 1s - loss: 0.3607 - accuracy: 0.8384 - val\_loss: 0.3651 - val\_accuracy: 0.8372 - 517ms/epoch - 14ms/step  
## Epoch 63/100  
## 36/36 - 1s - loss: 0.3652 - accuracy: 0.8378 - val\_loss: 0.3644 - val\_accuracy: 0.8379 - 561ms/epoch - 16ms/step  
## Epoch 64/100  
## 36/36 - 1s - loss: 0.3631 - accuracy: 0.8368 - val\_loss: 0.3646 - val\_accuracy: 0.8376 - 526ms/epoch - 15ms/step  
## Epoch 65/100  
## 36/36 - 1s - loss: 0.3648 - accuracy: 0.8360 - val\_loss: 0.3626 - val\_accuracy: 0.8379 - 515ms/epoch - 14ms/step  
## Epoch 66/100  
## 36/36 - 1s - loss: 0.3631 - accuracy: 0.8374 - val\_loss: 0.3634 - val\_accuracy: 0.8370 - 521ms/epoch - 14ms/step  
## Epoch 67/100  
## 36/36 - 1s - loss: 0.3577 - accuracy: 0.8412 - val\_loss: 0.3626 - val\_accuracy: 0.8377 - 522ms/epoch - 14ms/step  
## Epoch 68/100  
## 36/36 - 1s - loss: 0.3630 - accuracy: 0.8367 - val\_loss: 0.3626 - val\_accuracy: 0.8378 - 518ms/epoch - 14ms/step  
## Epoch 69/100  
## 36/36 - 1s - loss: 0.3586 - accuracy: 0.8393 - val\_loss: 0.3611 - val\_accuracy: 0.8371 - 517ms/epoch - 14ms/step  
## Epoch 70/100  
## 36/36 - 1s - loss: 0.3591 - accuracy: 0.8402 - val\_loss: 0.3627 - val\_accuracy: 0.8388 - 513ms/epoch - 14ms/step  
## Epoch 71/100  
## 36/36 - 1s - loss: 0.3571 - accuracy: 0.8403 - val\_loss: 0.3599 - val\_accuracy: 0.8396 - 514ms/epoch - 14ms/step  
## Epoch 72/100  
## 36/36 - 1s - loss: 0.3573 - accuracy: 0.8391 - val\_loss: 0.3600 - val\_accuracy: 0.8385 - 514ms/epoch - 14ms/step  
## Epoch 73/100  
## 36/36 - 1s - loss: 0.3570 - accuracy: 0.8429 - val\_loss: 0.3588 - val\_accuracy: 0.8380 - 514ms/epoch - 14ms/step  
## Epoch 74/100  
## 36/36 - 1s - loss: 0.3588 - accuracy: 0.8376 - val\_loss: 0.3594 - val\_accuracy: 0.8400 - 513ms/epoch - 14ms/step  
## Epoch 75/100  
## 36/36 - 1s - loss: 0.3577 - accuracy: 0.8404 - val\_loss: 0.3575 - val\_accuracy: 0.8427 - 534ms/epoch - 15ms/step  
## Epoch 76/100  
## 36/36 - 1s - loss: 0.3554 - accuracy: 0.8406 - val\_loss: 0.3604 - val\_accuracy: 0.8417 - 522ms/epoch - 14ms/step  
## Epoch 77/100  
## 36/36 - 1s - loss: 0.3538 - accuracy: 0.8410 - val\_loss: 0.3584 - val\_accuracy: 0.8408 - 512ms/epoch - 14ms/step  
## Epoch 78/100  
## 36/36 - 1s - loss: 0.3565 - accuracy: 0.8421 - val\_loss: 0.3572 - val\_accuracy: 0.8389 - 515ms/epoch - 14ms/step  
## Epoch 79/100  
## 36/36 - 1s - loss: 0.3540 - accuracy: 0.8415 - val\_loss: 0.3579 - val\_accuracy: 0.8397 - 512ms/epoch - 14ms/step  
## Epoch 80/100  
## 36/36 - 1s - loss: 0.3550 - accuracy: 0.8396 - val\_loss: 0.3592 - val\_accuracy: 0.8388 - 514ms/epoch - 14ms/step  
## Epoch 81/100  
## 36/36 - 1s - loss: 0.3580 - accuracy: 0.8402 - val\_loss: 0.3564 - val\_accuracy: 0.8387 - 517ms/epoch - 14ms/step  
## Epoch 82/100  
## 36/36 - 1s - loss: 0.3528 - accuracy: 0.8414 - val\_loss: 0.3581 - val\_accuracy: 0.8396 - 516ms/epoch - 14ms/step  
## Epoch 83/100  
## 36/36 - 1s - loss: 0.3534 - accuracy: 0.8419 - val\_loss: 0.3561 - val\_accuracy: 0.8433 - 517ms/epoch - 14ms/step  
## Epoch 84/100  
## 36/36 - 1s - loss: 0.3529 - accuracy: 0.8410 - val\_loss: 0.3548 - val\_accuracy: 0.8429 - 534ms/epoch - 15ms/step  
## Epoch 85/100  
## 36/36 - 1s - loss: 0.3542 - accuracy: 0.8451 - val\_loss: 0.3559 - val\_accuracy: 0.8456 - 515ms/epoch - 14ms/step  
## Epoch 86/100  
## 36/36 - 1s - loss: 0.3511 - accuracy: 0.8435 - val\_loss: 0.3573 - val\_accuracy: 0.8425 - 513ms/epoch - 14ms/step  
## Epoch 87/100  
## 36/36 - 1s - loss: 0.3506 - accuracy: 0.8435 - val\_loss: 0.3560 - val\_accuracy: 0.8433 - 521ms/epoch - 14ms/step

# Plot training history  
plot(history)



# Using the model to make predictions  
predictions <- predict(model, test\_features)

## 284/284 - 1s - 578ms/epoch - 2ms/step

test\_set$p\_prob <- predictions[, 1]  
head(predictions, 10)

## [,1]  
## [1,] 0.121874869  
## [2,] 0.048223857  
## [3,] 0.040033769  
## [4,] 0.020202033  
## [5,] 0.115720607  
## [6,] 0.023871146  
## [7,] 0.008270793  
## [8,] 0.955575347  
## [9,] 0.013673816  
## [10,] 0.008958214

predicted\_class <- (predictions[, 1] >= 0.5) \* 1  
head(predicted\_class, 10)

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

# Calculating accuracy  
accuracy <- mean(predicted\_class == test\_labels)  
accuracy

## [1] 0.8410421

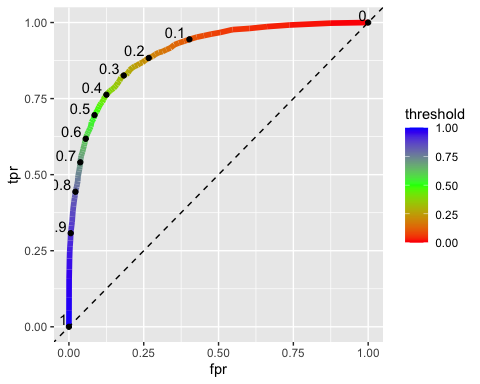
# Making predictions and calculating fpr and tpr rates at 0.5 threshold  
over\_threshold <- test\_set[test\_set$p\_prob >= 0.5, ]  
fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
fpr

## [1] 0.08554817

tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
tpr

## [1] 0.6956236

# Plotting ROC curve  
roc\_data <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)  
for (i in roc\_data$threshold) {  
 over\_threshold <- test\_set[test\_set$p\_prob >= i, ]  
 fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
 roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
 tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
 roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +   
 geom\_line(data = roc\_data,   
 aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +   
 scale\_color\_gradientn(colors = rainbow(3)) +   
 geom\_abline(intercept = 0, slope = 1, lty = 2) +   
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +   
 geom\_text(data = roc\_data[seq(1, 101, 10), ],   
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# Calculating the AUC  
auc <- auc(x = roc\_data$fpr, y = roc\_data$tpr, type = "spline")

## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique  
## 'x' values

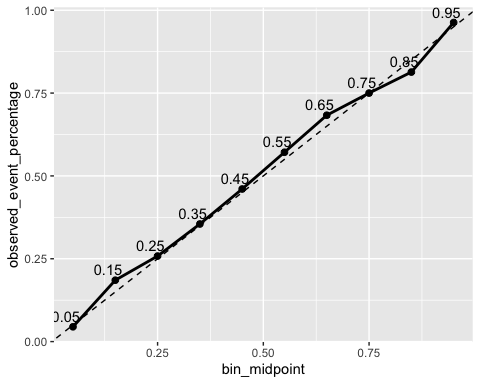
auc

## [1] 0.9041712

# Creating a calibration curve  
in\_interval <- test\_set[test\_set$p\_prob >= 0.7 & test\_set$p\_prob <= 0.8, ]  
nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)

## [1] 0.75

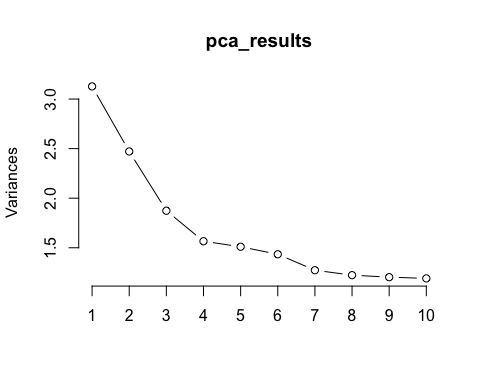
calibration\_data <- data.frame(bin\_midpoint=seq(0.05,0.95,0.1),  
 observed\_event\_percentage=0)  
for (i in seq(0.05,0.95,0.1)) {  
 in\_interval <- test\_set[test\_set$p\_prob >= (i-0.05) & test\_set$p\_prob <= (i+0.05), ]  
 oep <- nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)  
 calibration\_data[calibration\_data$bin\_midpoint==i, "observed\_event\_percentage"] <- oep  
}  
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
 geom\_line(linewidth = 1) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(size = 2) +  
 geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)



# Building another neural network model with PCA  
# Running PCA  
pca\_results <- prcomp(training\_features)  
summary(pca\_results)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 1.76853 1.57206 1.36882 1.25127 1.22894 1.19765 1.12853  
## Proportion of Variance 0.07447 0.05884 0.04461 0.03728 0.03596 0.03415 0.03032  
## Cumulative Proportion 0.07447 0.13331 0.17792 0.21520 0.25116 0.28531 0.31563  
## PC8 PC9 PC10 PC11 PC12 PC13 PC14  
## Standard deviation 1.10616 1.09701 1.09130 1.08001 1.06595 1.05940 1.04791  
## Proportion of Variance 0.02913 0.02865 0.02836 0.02777 0.02705 0.02672 0.02615  
## Cumulative Proportion 0.34477 0.37342 0.40178 0.42955 0.45660 0.48332 0.50947  
## PC15 PC16 PC17 PC18 PC19 PC20 PC21  
## Standard deviation 1.04244 1.03552 1.0288 1.02333 1.01268 1.00460 0.99967  
## Proportion of Variance 0.02587 0.02553 0.0252 0.02493 0.02442 0.02403 0.02379  
## Cumulative Proportion 0.53534 0.56087 0.5861 0.61101 0.63542 0.65945 0.68325  
## PC22 PC23 PC24 PC25 PC26 PC27 PC28  
## Standard deviation 0.98848 0.98315 0.97693 0.9743 0.96261 0.95443 0.94294  
## Proportion of Variance 0.02326 0.02301 0.02272 0.0226 0.02206 0.02169 0.02117  
## Cumulative Proportion 0.70651 0.72952 0.75225 0.7749 0.79691 0.81860 0.83977  
## PC29 PC30 PC31 PC32 PC33 PC34 PC35  
## Standard deviation 0.92878 0.89286 0.85892 0.8250 0.81097 0.7721 0.71775  
## Proportion of Variance 0.02054 0.01898 0.01757 0.0162 0.01566 0.0142 0.01227  
## Cumulative Proportion 0.86031 0.87929 0.89685 0.9131 0.92872 0.9429 0.95518  
## PC36 PC37 PC38 PC39 PC40 PC41 PC42  
## Standard deviation 0.67507 0.64417 0.58820 0.53778 0.43749 0.42251 0.08219  
## Proportion of Variance 0.01085 0.00988 0.00824 0.00689 0.00456 0.00425 0.00016  
## Cumulative Proportion 0.96603 0.97591 0.98415 0.99103 0.99559 0.99984 1.00000

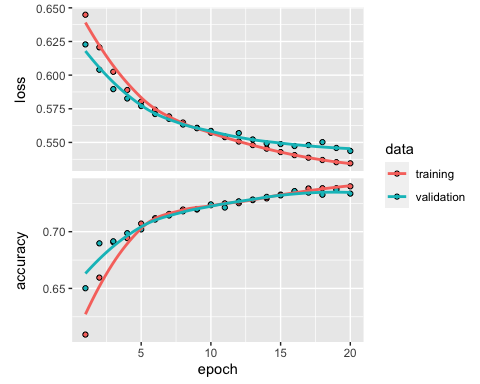
screeplot(pca\_results, type = "line")



# Reducing to 4 PCs  
n\_components <- 4  
reduced\_training\_features <- pca\_results$x[, 1:n\_components]  
reduced\_test\_features <- predict(pca\_results, newdata = test\_features)[, 1:n\_components]  
  
pca\_model <- keras\_model\_sequential(list(  
 layer\_dense(units = 10, activation = "relu"),   
 layer\_dense(units = 5, activation = "relu"),   
 layer\_dense(units = 5, activation = "tanh"),   
 layer\_dense(units = 1, activation = "sigmoid")  
))  
compile(pca\_model,   
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = "accuracy")  
  
# Training the model  
history <- fit(pca\_model, reduced\_training\_features, training\_labels,   
 epochs = 20, batch\_size = 512, validation\_split = 0.33)

## Epoch 1/20  
## 36/36 - 3s - loss: 0.6449 - accuracy: 0.6093 - val\_loss: 0.6228 - val\_accuracy: 0.6502 - 3s/epoch - 75ms/step  
## Epoch 2/20  
## 36/36 - 0s - loss: 0.6207 - accuracy: 0.6595 - val\_loss: 0.6040 - val\_accuracy: 0.6897 - 463ms/epoch - 13ms/step  
## Epoch 3/20  
## 36/36 - 0s - loss: 0.6025 - accuracy: 0.6908 - val\_loss: 0.5896 - val\_accuracy: 0.6915 - 462ms/epoch - 13ms/step  
## Epoch 4/20  
## 36/36 - 0s - loss: 0.5889 - accuracy: 0.6944 - val\_loss: 0.5826 - val\_accuracy: 0.6987 - 447ms/epoch - 12ms/step  
## Epoch 5/20  
## 36/36 - 0s - loss: 0.5803 - accuracy: 0.7071 - val\_loss: 0.5770 - val\_accuracy: 0.7020 - 449ms/epoch - 12ms/step  
## Epoch 6/20  
## 36/36 - 0s - loss: 0.5743 - accuracy: 0.7119 - val\_loss: 0.5711 - val\_accuracy: 0.7107 - 448ms/epoch - 12ms/step  
## Epoch 7/20  
## 36/36 - 0s - loss: 0.5693 - accuracy: 0.7156 - val\_loss: 0.5675 - val\_accuracy: 0.7143 - 444ms/epoch - 12ms/step  
## Epoch 8/20  
## 36/36 - 0s - loss: 0.5648 - accuracy: 0.7195 - val\_loss: 0.5633 - val\_accuracy: 0.7179 - 454ms/epoch - 13ms/step  
## Epoch 9/20  
## 36/36 - 0s - loss: 0.5608 - accuracy: 0.7195 - val\_loss: 0.5606 - val\_accuracy: 0.7200 - 445ms/epoch - 12ms/step  
## Epoch 10/20  
## 36/36 - 0s - loss: 0.5572 - accuracy: 0.7233 - val\_loss: 0.5585 - val\_accuracy: 0.7241 - 440ms/epoch - 12ms/step  
## Epoch 11/20  
## 36/36 - 0s - loss: 0.5540 - accuracy: 0.7250 - val\_loss: 0.5545 - val\_accuracy: 0.7213 - 447ms/epoch - 12ms/step  
## Epoch 12/20  
## 36/36 - 0s - loss: 0.5507 - accuracy: 0.7251 - val\_loss: 0.5569 - val\_accuracy: 0.7268 - 449ms/epoch - 12ms/step  
## Epoch 13/20  
## 36/36 - 0s - loss: 0.5480 - accuracy: 0.7278 - val\_loss: 0.5522 - val\_accuracy: 0.7283 - 449ms/epoch - 12ms/step  
## Epoch 14/20  
## 36/36 - 0s - loss: 0.5454 - accuracy: 0.7294 - val\_loss: 0.5492 - val\_accuracy: 0.7307 - 444ms/epoch - 12ms/step  
## Epoch 15/20  
## 36/36 - 0s - loss: 0.5429 - accuracy: 0.7327 - val\_loss: 0.5488 - val\_accuracy: 0.7320 - 449ms/epoch - 12ms/step  
## Epoch 16/20  
## 36/36 - 0s - loss: 0.5406 - accuracy: 0.7351 - val\_loss: 0.5473 - val\_accuracy: 0.7358 - 445ms/epoch - 12ms/step  
## Epoch 17/20  
## 36/36 - 0s - loss: 0.5386 - accuracy: 0.7381 - val\_loss: 0.5481 - val\_accuracy: 0.7344 - 444ms/epoch - 12ms/step  
## Epoch 18/20  
## 36/36 - 0s - loss: 0.5369 - accuracy: 0.7383 - val\_loss: 0.5502 - val\_accuracy: 0.7326 - 446ms/epoch - 12ms/step  
## Epoch 19/20  
## 36/36 - 0s - loss: 0.5353 - accuracy: 0.7383 - val\_loss: 0.5459 - val\_accuracy: 0.7369 - 451ms/epoch - 13ms/step  
## Epoch 20/20  
## 36/36 - 0s - loss: 0.5345 - accuracy: 0.7400 - val\_loss: 0.5436 - val\_accuracy: 0.7336 - 463ms/epoch - 13ms/step

plot(history)



# Using the model to make predictions  
pca\_predictions <- predict(pca\_model, reduced\_test\_features)

## 284/284 - 0s - 472ms/epoch - 2ms/step

test\_set$p\_prob <- pca\_predictions[, 1]  
head(pca\_predictions, 10)

## [,1]  
## [1,] 0.2031607  
## [2,] 0.1186430  
## [3,] 0.2716128  
## [4,] 0.2154604  
## [5,] 0.4540634  
## [6,] 0.2371480  
## [7,] 0.1043686  
## [8,] 0.7723917  
## [9,] 0.1244992  
## [10,] 0.1428141

pca\_predicted\_class <- (pca\_predictions[, 1] >= 0.5) \* 1  
head(pca\_predicted\_class, 10)

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

# Calculating accuracy  
pca\_accuracy <- mean(pca\_predicted\_class == test\_labels)  
pca\_accuracy

## [1] 0.7280053

# Tuning the model  
parameterGrid <- expand.grid(  
 units = c(5, 10, 15, 20),  
 activation = c("relu", "tanh", "sigmoid")  
)  
  
# Define a function to create a neural network model  
create\_model <- function(units, activation, learning\_rate) {  
 model <- keras\_model\_sequential() %>%  
 layer\_dense(units = units, activation = activation, input\_shape = ncol(reduced\_training\_features)) %>%  
 layer\_dense(units = units, activation = activation) %>%  
 layer\_dense(units = units, activation = activation) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
   
 compile(model, optimizer = "rmsprop", loss = "binary\_crossentropy", metrics = "accuracy")  
   
 return(model)  
}  
  
# Perform grid search  
results <- list()  
for (i in 1:nrow(parameterGrid)) {  
 model <- create\_model(parameterGrid$units[i], parameterGrid$activation[i], parameterGrid$learning\_rate[i])  
   
 history <- fit(model,   
 x = reduced\_training\_features,   
 y = training\_labels,   
 epochs = 10,   
 batch\_size = 512,   
 validation\_split = 0.33)  
   
 results[[i]] <- list(model = model, history = history)  
}

## Epoch 1/10  
## 36/36 - 2s - loss: 0.7330 - accuracy: 0.5925 - val\_loss: 0.6445 - val\_accuracy: 0.6338 - 2s/epoch - 42ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6316 - accuracy: 0.6425 - val\_loss: 0.6031 - val\_accuracy: 0.6670 - 450ms/epoch - 12ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.6057 - accuracy: 0.6693 - val\_loss: 0.5934 - val\_accuracy: 0.6776 - 450ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5981 - accuracy: 0.6821 - val\_loss: 0.5901 - val\_accuracy: 0.6876 - 452ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5958 - accuracy: 0.6893 - val\_loss: 0.5895 - val\_accuracy: 0.6844 - 448ms/epoch - 12ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5952 - accuracy: 0.6903 - val\_loss: 0.5901 - val\_accuracy: 0.6881 - 446ms/epoch - 12ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5954 - accuracy: 0.6921 - val\_loss: 0.5906 - val\_accuracy: 0.6890 - 441ms/epoch - 12ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5953 - accuracy: 0.6915 - val\_loss: 0.5913 - val\_accuracy: 0.6864 - 447ms/epoch - 12ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5952 - accuracy: 0.6925 - val\_loss: 0.5912 - val\_accuracy: 0.6873 - 443ms/epoch - 12ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5951 - accuracy: 0.6923 - val\_loss: 0.5905 - val\_accuracy: 0.6873 - 443ms/epoch - 12ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.7019 - accuracy: 0.5591 - val\_loss: 0.6436 - val\_accuracy: 0.6728 - 2s/epoch - 45ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6194 - accuracy: 0.6866 - val\_loss: 0.5992 - val\_accuracy: 0.6923 - 461ms/epoch - 13ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.5932 - accuracy: 0.6938 - val\_loss: 0.5833 - val\_accuracy: 0.7008 - 469ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5829 - accuracy: 0.6994 - val\_loss: 0.5786 - val\_accuracy: 0.7000 - 450ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5794 - accuracy: 0.7020 - val\_loss: 0.5761 - val\_accuracy: 0.7028 - 450ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5771 - accuracy: 0.7064 - val\_loss: 0.5746 - val\_accuracy: 0.7035 - 453ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5757 - accuracy: 0.7064 - val\_loss: 0.5750 - val\_accuracy: 0.7094 - 479ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5746 - accuracy: 0.7090 - val\_loss: 0.5760 - val\_accuracy: 0.7091 - 462ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5742 - accuracy: 0.7103 - val\_loss: 0.5756 - val\_accuracy: 0.6994 - 454ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5739 - accuracy: 0.7092 - val\_loss: 0.5730 - val\_accuracy: 0.7051 - 454ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 3s - loss: 0.6355 - accuracy: 0.6547 - val\_loss: 0.6055 - val\_accuracy: 0.6876 - 3s/epoch - 77ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6040 - accuracy: 0.6840 - val\_loss: 0.5908 - val\_accuracy: 0.6980 - 461ms/epoch - 13ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.5918 - accuracy: 0.6981 - val\_loss: 0.5836 - val\_accuracy: 0.6989 - 457ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5865 - accuracy: 0.6989 - val\_loss: 0.5808 - val\_accuracy: 0.6987 - 448ms/epoch - 12ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5831 - accuracy: 0.6990 - val\_loss: 0.5792 - val\_accuracy: 0.6998 - 436ms/epoch - 12ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5808 - accuracy: 0.7011 - val\_loss: 0.5806 - val\_accuracy: 0.6948 - 439ms/epoch - 12ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5799 - accuracy: 0.7016 - val\_loss: 0.5754 - val\_accuracy: 0.7071 - 441ms/epoch - 12ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5801 - accuracy: 0.7000 - val\_loss: 0.5882 - val\_accuracy: 0.6829 - 435ms/epoch - 12ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5790 - accuracy: 0.7010 - val\_loss: 0.5843 - val\_accuracy: 0.6849 - 447ms/epoch - 12ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5791 - accuracy: 0.6994 - val\_loss: 0.5750 - val\_accuracy: 0.7038 - 446ms/epoch - 12ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6619 - accuracy: 0.5869 - val\_loss: 0.6151 - val\_accuracy: 0.6581 - 2s/epoch - 47ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6034 - accuracy: 0.6724 - val\_loss: 0.5916 - val\_accuracy: 0.6776 - 477ms/epoch - 13ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.5862 - accuracy: 0.6939 - val\_loss: 0.5775 - val\_accuracy: 0.7001 - 451ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5831 - accuracy: 0.6966 - val\_loss: 0.5789 - val\_accuracy: 0.6940 - 452ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5807 - accuracy: 0.7000 - val\_loss: 0.5801 - val\_accuracy: 0.6906 - 448ms/epoch - 12ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5801 - accuracy: 0.6988 - val\_loss: 0.5760 - val\_accuracy: 0.6994 - 437ms/epoch - 12ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5845 - accuracy: 0.6909 - val\_loss: 0.5901 - val\_accuracy: 0.6808 - 446ms/epoch - 12ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5835 - accuracy: 0.6932 - val\_loss: 0.5924 - val\_accuracy: 0.6769 - 439ms/epoch - 12ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5870 - accuracy: 0.6853 - val\_loss: 0.5937 - val\_accuracy: 0.6760 - 442ms/epoch - 12ms/step  
## Epoch 10/10  
## 36/36 - 1s - loss: 0.5886 - accuracy: 0.6850 - val\_loss: 0.5854 - val\_accuracy: 0.6858 - 529ms/epoch - 15ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.7209 - accuracy: 0.4838 - val\_loss: 0.6875 - val\_accuracy: 0.5523 - 2s/epoch - 49ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6688 - accuracy: 0.6182 - val\_loss: 0.6470 - val\_accuracy: 0.6838 - 481ms/epoch - 13ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.6338 - accuracy: 0.6893 - val\_loss: 0.6181 - val\_accuracy: 0.6920 - 462ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.6093 - accuracy: 0.7000 - val\_loss: 0.5989 - val\_accuracy: 0.6978 - 471ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5938 - accuracy: 0.7001 - val\_loss: 0.5863 - val\_accuracy: 0.6979 - 486ms/epoch - 14ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5836 - accuracy: 0.6998 - val\_loss: 0.5787 - val\_accuracy: 0.7029 - 460ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5767 - accuracy: 0.7039 - val\_loss: 0.5731 - val\_accuracy: 0.7041 - 462ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5712 - accuracy: 0.7094 - val\_loss: 0.5686 - val\_accuracy: 0.7109 - 470ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5661 - accuracy: 0.7150 - val\_loss: 0.5642 - val\_accuracy: 0.7232 - 463ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 1s - loss: 0.5619 - accuracy: 0.7240 - val\_loss: 0.5617 - val\_accuracy: 0.7179 - 502ms/epoch - 14ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.7111 - accuracy: 0.5361 - val\_loss: 0.6608 - val\_accuracy: 0.6350 - 2s/epoch - 53ms/step  
## Epoch 2/10  
## 36/36 - 1s - loss: 0.6321 - accuracy: 0.6649 - val\_loss: 0.6091 - val\_accuracy: 0.6799 - 504ms/epoch - 14ms/step  
## Epoch 3/10  
## 36/36 - 1s - loss: 0.5966 - accuracy: 0.6908 - val\_loss: 0.5852 - val\_accuracy: 0.6952 - 501ms/epoch - 14ms/step  
## Epoch 4/10  
## 36/36 - 1s - loss: 0.5792 - accuracy: 0.7078 - val\_loss: 0.5728 - val\_accuracy: 0.7115 - 522ms/epoch - 14ms/step  
## Epoch 5/10  
## 36/36 - 1s - loss: 0.5688 - accuracy: 0.7140 - val\_loss: 0.5663 - val\_accuracy: 0.7143 - 509ms/epoch - 14ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5620 - accuracy: 0.7215 - val\_loss: 0.5638 - val\_accuracy: 0.7215 - 498ms/epoch - 14ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5577 - accuracy: 0.7257 - val\_loss: 0.5601 - val\_accuracy: 0.7246 - 473ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5542 - accuracy: 0.7287 - val\_loss: 0.5598 - val\_accuracy: 0.7205 - 472ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5515 - accuracy: 0.7302 - val\_loss: 0.5569 - val\_accuracy: 0.7231 - 475ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5490 - accuracy: 0.7310 - val\_loss: 0.5576 - val\_accuracy: 0.7224 - 469ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6199 - accuracy: 0.6623 - val\_loss: 0.5875 - val\_accuracy: 0.7023 - 2s/epoch - 54ms/step  
## Epoch 2/10  
## 36/36 - 1s - loss: 0.5750 - accuracy: 0.7144 - val\_loss: 0.5743 - val\_accuracy: 0.7099 - 502ms/epoch - 14ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.5634 - accuracy: 0.7232 - val\_loss: 0.5674 - val\_accuracy: 0.7162 - 467ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5592 - accuracy: 0.7262 - val\_loss: 0.5644 - val\_accuracy: 0.7195 - 493ms/epoch - 14ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5572 - accuracy: 0.7279 - val\_loss: 0.5628 - val\_accuracy: 0.7235 - 462ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5552 - accuracy: 0.7289 - val\_loss: 0.5609 - val\_accuracy: 0.7226 - 476ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5536 - accuracy: 0.7298 - val\_loss: 0.5647 - val\_accuracy: 0.7134 - 475ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5515 - accuracy: 0.7291 - val\_loss: 0.5644 - val\_accuracy: 0.7143 - 458ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5497 - accuracy: 0.7286 - val\_loss: 0.5591 - val\_accuracy: 0.7203 - 461ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5475 - accuracy: 0.7282 - val\_loss: 0.5587 - val\_accuracy: 0.7198 - 467ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6159 - accuracy: 0.6608 - val\_loss: 0.5827 - val\_accuracy: 0.7061 - 2s/epoch - 54ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.5700 - accuracy: 0.7165 - val\_loss: 0.5670 - val\_accuracy: 0.7156 - 490ms/epoch - 14ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.5608 - accuracy: 0.7250 - val\_loss: 0.5654 - val\_accuracy: 0.7197 - 493ms/epoch - 14ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.5579 - accuracy: 0.7268 - val\_loss: 0.5615 - val\_accuracy: 0.7194 - 470ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.5552 - accuracy: 0.7284 - val\_loss: 0.5605 - val\_accuracy: 0.7210 - 472ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.5526 - accuracy: 0.7308 - val\_loss: 0.5596 - val\_accuracy: 0.7196 - 462ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5499 - accuracy: 0.7327 - val\_loss: 0.5567 - val\_accuracy: 0.7229 - 459ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5477 - accuracy: 0.7339 - val\_loss: 0.5580 - val\_accuracy: 0.7212 - 460ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5448 - accuracy: 0.7339 - val\_loss: 0.5566 - val\_accuracy: 0.7227 - 469ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5414 - accuracy: 0.7366 - val\_loss: 0.5552 - val\_accuracy: 0.7202 - 456ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6596 - accuracy: 0.6769 - val\_loss: 0.6499 - val\_accuracy: 0.6706 - 2s/epoch - 55ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6405 - accuracy: 0.6769 - val\_loss: 0.6382 - val\_accuracy: 0.6706 - 485ms/epoch - 13ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.6316 - accuracy: 0.6769 - val\_loss: 0.6339 - val\_accuracy: 0.6706 - 461ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.6288 - accuracy: 0.6769 - val\_loss: 0.6331 - val\_accuracy: 0.6706 - 476ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.6280 - accuracy: 0.6769 - val\_loss: 0.6323 - val\_accuracy: 0.6706 - 457ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.6272 - accuracy: 0.6769 - val\_loss: 0.6315 - val\_accuracy: 0.6706 - 458ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.6265 - accuracy: 0.6769 - val\_loss: 0.6307 - val\_accuracy: 0.6706 - 472ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.6257 - accuracy: 0.6769 - val\_loss: 0.6300 - val\_accuracy: 0.6706 - 458ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.6249 - accuracy: 0.6769 - val\_loss: 0.6289 - val\_accuracy: 0.6706 - 458ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.6239 - accuracy: 0.6769 - val\_loss: 0.6279 - val\_accuracy: 0.6706 - 455ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 1.0609 - accuracy: 0.3231 - val\_loss: 0.9571 - val\_accuracy: 0.3294 - 2s/epoch - 57ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.8954 - accuracy: 0.3231 - val\_loss: 0.8254 - val\_accuracy: 0.3294 - 489ms/epoch - 14ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.7800 - accuracy: 0.3231 - val\_loss: 0.7325 - val\_accuracy: 0.3294 - 484ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 1s - loss: 0.7017 - accuracy: 0.4577 - val\_loss: 0.6733 - val\_accuracy: 0.6706 - 509ms/epoch - 14ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.6553 - accuracy: 0.6769 - val\_loss: 0.6425 - val\_accuracy: 0.6706 - 478ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.6335 - accuracy: 0.6769 - val\_loss: 0.6316 - val\_accuracy: 0.6706 - 474ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.6268 - accuracy: 0.6769 - val\_loss: 0.6296 - val\_accuracy: 0.6706 - 487ms/epoch - 14ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.6253 - accuracy: 0.6769 - val\_loss: 0.6286 - val\_accuracy: 0.6706 - 480ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.6241 - accuracy: 0.6769 - val\_loss: 0.6273 - val\_accuracy: 0.6706 - 471ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.6228 - accuracy: 0.6769 - val\_loss: 0.6253 - val\_accuracy: 0.6706 - 470ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6282 - accuracy: 0.6769 - val\_loss: 0.6317 - val\_accuracy: 0.6706 - 2s/epoch - 51ms/step  
## Epoch 2/10  
## 36/36 - 1s - loss: 0.6265 - accuracy: 0.6769 - val\_loss: 0.6298 - val\_accuracy: 0.6706 - 571ms/epoch - 16ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.6241 - accuracy: 0.6769 - val\_loss: 0.6267 - val\_accuracy: 0.6706 - 463ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.6206 - accuracy: 0.6769 - val\_loss: 0.6218 - val\_accuracy: 0.6706 - 468ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.6156 - accuracy: 0.6769 - val\_loss: 0.6162 - val\_accuracy: 0.6706 - 469ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.6098 - accuracy: 0.6769 - val\_loss: 0.6094 - val\_accuracy: 0.6706 - 455ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.6042 - accuracy: 0.6769 - val\_loss: 0.6030 - val\_accuracy: 0.6706 - 469ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5991 - accuracy: 0.6769 - val\_loss: 0.5981 - val\_accuracy: 0.6706 - 456ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5950 - accuracy: 0.6770 - val\_loss: 0.5927 - val\_accuracy: 0.6728 - 463ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5913 - accuracy: 0.6876 - val\_loss: 0.5888 - val\_accuracy: 0.6877 - 465ms/epoch - 13ms/step  
## Epoch 1/10  
## 36/36 - 2s - loss: 0.6462 - accuracy: 0.6578 - val\_loss: 0.6329 - val\_accuracy: 0.6706 - 2s/epoch - 53ms/step  
## Epoch 2/10  
## 36/36 - 0s - loss: 0.6272 - accuracy: 0.6769 - val\_loss: 0.6299 - val\_accuracy: 0.6706 - 495ms/epoch - 14ms/step  
## Epoch 3/10  
## 36/36 - 0s - loss: 0.6236 - accuracy: 0.6769 - val\_loss: 0.6249 - val\_accuracy: 0.6706 - 475ms/epoch - 13ms/step  
## Epoch 4/10  
## 36/36 - 0s - loss: 0.6183 - accuracy: 0.6769 - val\_loss: 0.6185 - val\_accuracy: 0.6706 - 473ms/epoch - 13ms/step  
## Epoch 5/10  
## 36/36 - 0s - loss: 0.6118 - accuracy: 0.6769 - val\_loss: 0.6110 - val\_accuracy: 0.6706 - 463ms/epoch - 13ms/step  
## Epoch 6/10  
## 36/36 - 0s - loss: 0.6048 - accuracy: 0.6769 - val\_loss: 0.6031 - val\_accuracy: 0.6706 - 482ms/epoch - 13ms/step  
## Epoch 7/10  
## 36/36 - 0s - loss: 0.5986 - accuracy: 0.6770 - val\_loss: 0.5963 - val\_accuracy: 0.6721 - 458ms/epoch - 13ms/step  
## Epoch 8/10  
## 36/36 - 0s - loss: 0.5937 - accuracy: 0.6824 - val\_loss: 0.5909 - val\_accuracy: 0.6806 - 456ms/epoch - 13ms/step  
## Epoch 9/10  
## 36/36 - 0s - loss: 0.5899 - accuracy: 0.6966 - val\_loss: 0.5868 - val\_accuracy: 0.6916 - 469ms/epoch - 13ms/step  
## Epoch 10/10  
## 36/36 - 0s - loss: 0.5869 - accuracy: 0.7006 - val\_loss: 0.5839 - val\_accuracy: 0.6924 - 464ms/epoch - 13ms/step

# Evaluate results and choose the best model  
best\_accuracy <- 0  
best\_model <- NULL  
for (i in 1:length(results)) {  
 accuracy <- max(results[[i]]$history$metrics$val\_accuracy)  
 if (accuracy > best\_accuracy) {  
 best\_accuracy <- accuracy  
 best\_model <- results[[i]]$model  
 }  
}  
summary(best\_model)

## Model: "sequential\_8"  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## Layer (type) Output Shape Param #   
## ================================================================================  
## dense\_34 (Dense) (None, 10) 50   
## dense\_33 (Dense) (None, 10) 110   
## dense\_32 (Dense) (None, 10) 110   
## dense\_31 (Dense) (None, 1) 11   
## ================================================================================  
## Total params: 281 (1.10 KB)  
## Trainable params: 281 (1.10 KB)  
## Non-trainable params: 0 (0.00 Byte)  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

str(best\_model)

## Model: "sequential\_8"  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## Layer (type) Output Shape Param #   
## ================================================================================  
## dense\_34 (Dense) (None, 10) 50   
## dense\_33 (Dense) (None, 10) 110   
## dense\_32 (Dense) (None, 10) 110   
## dense\_31 (Dense) (None, 1) 11   
## ================================================================================  
## Total params: 281 (1.10 KB)  
## Trainable params: 281 (1.10 KB)  
## Non-trainable params: 0 (0.00 Byte)  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_functions <- lapply(best\_model$layers, `[[`, "activation")  
print(activation\_functions)

## [[1]]  
## <function tanh at 0x2b3f0e430>  
##   
## [[2]]  
## <function tanh at 0x2b3f0e430>  
##   
## [[3]]  
## <function tanh at 0x2b3f0e430>  
##   
## [[4]]  
## <function sigmoid at 0x2b3f0e5e0>

# Use the best model for predictions  
predictions <- predict(best\_model, reduced\_test\_features)

## 284/284 - 1s - 526ms/epoch - 2ms/step

test\_set$p\_prob <- predictions[, 1]  
pca\_predicted\_class <- ifelse(predictions[, 1] >= 0.5, 1, 0)  
pca\_accuracy <- mean(pca\_predicted\_class == test\_labels)  
pca\_accuracy

## [1] 0.7210509

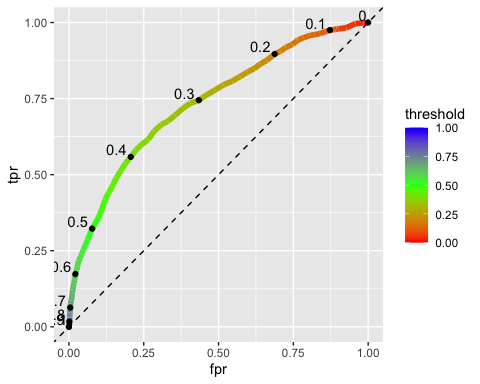
# Making predictions and calculating fpr and tpr rates at 0.5 threshold  
over\_threshold <- test\_set[test\_set$p\_prob >= 0.5, ]  
fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
fpr

## [1] 0.07774086

tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
tpr

## [1] 0.3224745

# Plotting ROC curve  
roc\_data <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)  
for (i in roc\_data$threshold) {  
 over\_threshold <- test\_set[test\_set$p\_prob >= i, ]  
 fpr <- sum(over\_threshold$booking\_status==0)/sum(test\_set$booking\_status==0)  
 roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
 tpr <- sum(over\_threshold$booking\_status==1)/sum(test\_set$booking\_status==1)  
 roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +   
 geom\_line(data = roc\_data,   
 aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +   
 scale\_color\_gradientn(colors = rainbow(3)) +   
 geom\_abline(intercept = 0, slope = 1, lty = 2) +   
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +   
 geom\_text(data = roc\_data[seq(1, 101, 10), ],   
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# Calculating the AUC  
pca\_auc <- auc(x = roc\_data$fpr, y = roc\_data$tpr, type = "spline")

## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique  
## 'x' values

pca\_auc

## [1] 0.7334001

# Creating a calibration curve  
in\_interval <- test\_set[test\_set$p\_prob >= 0.7 & test\_set$p\_prob <= 0.8, ]  
nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)

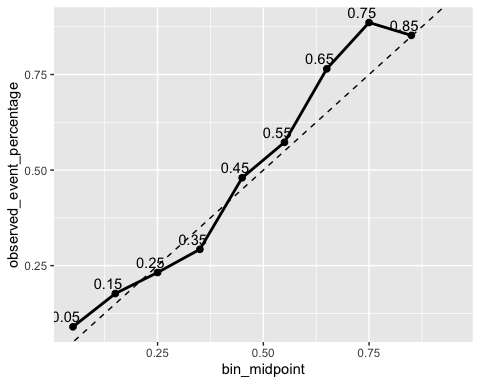
## [1] 0.8860759

calibration\_data <- data.frame(bin\_midpoint=seq(0.05,0.95,0.1),  
 observed\_event\_percentage=0)  
for (i in seq(0.05,0.95,0.1)) {  
 in\_interval <- test\_set[test\_set$p\_prob >= (i-0.05) & test\_set$p\_prob <= (i+0.05), ]  
 oep <- nrow(in\_interval[in\_interval$booking\_status==1, ])/nrow(in\_interval)  
 calibration\_data[calibration\_data$bin\_midpoint==i, "observed\_event\_percentage"] <- oep  
}  
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
 geom\_line(linewidth = 1) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(size = 2) +  
 geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)

## Warning: Removed 1 row containing missing values (`geom\_line()`).

## Warning: Removed 1 rows containing missing values (`geom\_point()`).

## Warning: Removed 1 rows containing missing values (`geom\_text()`).



# Table with models and relative accuracies  
classification\_overview <- data.frame(  
 Method = c("Logistic Regression", "kNN (k = 3)", "Random Forest", "Simple Neural Network", "Complex Neural Network", "Neural Network with PCA"),  
 Accuracy = c("80.38%", "84.23%", "88.37%", "80.02%", "84.10%", "72.11%")  
)  
classification\_table <- kable(classification\_overview, "markdown") %>%  
 kable\_styling(full\_width = FALSE) %>%  
 column\_spec(1, bold = TRUE)

## Warning in kable\_styling(., full\_width = FALSE): Please specify format in  
## kable. kableExtra can customize either HTML or LaTeX outputs. See  
## https://haozhu233.github.io/kableExtra/ for details.

## Warning in column\_spec(., 1, bold = TRUE): Please specify format in kable.  
## kableExtra can customize either HTML or LaTeX outputs. See  
## https://haozhu233.github.io/kableExtra/ for details.

classification\_table

| Method | Accuracy |
| --- | --- |
| Logistic Regression | 80.38% |
| kNN (k = 3) | 84.23% |
| Random Forest | 88.37% |
| Simple Neural Network | 80.02% |
| Complex Neural Network | 84.10% |
| Neural Network with PCA | 72.11% |