Titanic Lab

Aaron C Cochran

September 12, 2017

Intro

Our dataset for working on the lab exercises is a list of passengers on the Titanic. This is a classic dataset for teaching machine learning and binary classification, and will serve our purposes well here. This lab is derived from work by **Megan L. Risdal** on the machine learning data science site Kaggle.

In this lab, we'll be tackling 3 main areas:

- Feature engineering
- Missing value imputation
- Prediction using logistic regression

This will require 5 packages.

```
library(tidyverse) # basically, use this all of the time
library(mice) # for imputation
library(scales) # for visualization
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(ggthemes) # for visualization
```

Dataset

We can obtain the data directly from Kaggle. However, for this lab, I've already downloaded the data and saved it in my working directory. The data are in the `data/titanic/` folder. There are two files called `test.csv` and `train.csv` and denote test and training datasets respectively.

The training set is used to build the predictive model, and contains 1 extra variable: the outcome. The test set is what we'll use to see how well our model works, and does not contain the outcome variable. It's our job to predict this outcome. In this case, it's predicting whether or not the passenger survived the sinking of the HMS Titanic.

```
test <- read_csv('data/titanic/test.csv')
```

```
## Parsed with column specification:
## cols(
##   PassengerId = col_integer(),
##   Pclass = col_integer(),
##   Name = col_character(),
##   Sex = col_character(),
##   Age = col_double(),
##   SibSp = col_integer(),
##   Parch = col_integer(),
##   Ticket = col_character(),
##   Fare = col_double(),
##   Cabin = col_character(),
##   Embarked = col_character()
## )
```

```
train <- read_csv('data/titanic/train.csv')
```

```
## Parsed with column specification:
## cols(
##   PassengerId = col_integer(),
##   Survived = col_integer(),
##   Pclass = col_integer(),
##   Name = col_character(),
##   Sex = col_character(),
##   Age = col_double(),
##   SibSp = col_integer(),
##   Parch = col_integer(),
##   Ticket = col_character(),
##   Fare = col_double(),
##   Cabin = col_character(),
##   Embarked = col_character()
## )
```

```
# combined dataset of both test and train data
full <- bind_rows(train, test)
```

Explore the dataset

```
str(full)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1309 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
##  $ Sex        : chr  "male" "female" "female" "female" ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  NA "C85" NA "C123" ...
##  $ Embarked   : chr  "S" "C" "S" "S" ...
```

Some of these variables are intuitively named, others are not. `SibSp` denotes the number of siblings or spouses the passenger has aboard. `Parch` is the number of parents or children the passenger has aboard. For passengers only traveling with a nanny, the `SibSp` and `Parch` variables would both be 0. `Embarked` is the port they embarked from. It is coded as a single letter: C = Cherbourg, Q = Queenstown, S = Southampton.

Feature engineering

The **passenger name** variable contains some extra information beyond the passenger's first and last name. Particularly, it contains the title each passenger went by. Additionally, the surname can be used to represent families. Let's create some variables, or as they say, do some **feature engineering.**

This section uses something known as *regular expressions*, or *regex* to parse the text and extract portions to use in the creation of new variables. Regex is not meant to be easy for humans to read, but for machines it is a snap. This makes learning it difficult, and implementing it often involves some trial and error (even after years of R usage). A good resource to test regular expressions is the website RegExr, available at www.regexr.com.

```
# grab the title from the passenger name using regular expressions (regex)

full$Title <- gsub('(.*, )|(\\..*)', '', full$Name)

# Show the title counts by sex
table(full$Sex, full$Title)
```

```
##
##          Capt Col Don Dona  Dr Jonkheer Lady Major Master Miss Mlle Mme
##   female    0   0   0    1   1        0    1     0      0  260    2   1
##   male      1   4   1    0   7        1    0     2     61    0    0   0
##
##           Mr Mrs  Ms Rev Sir the Countess
##   female   0 197   2   0   0            1
##   male   757   0   0   8   1            0
```

There are some columns with really low counts. We should consider combining these into a "Rare Title" category. Then, there are titles that mean the same thing as others: `Mlle` is Mademoiselle, the French equivalent of `Miss`. `Mme` is Madame, which is analogous to `Mrs`.

```
rare_title <- c("Dona", "Lady", "the Countess",
                "Capt", "Col", "Don", "Dr", "Major",
                "Rev", "Sir", "Jonkheer")

# reassigning french titles and rare titles
full$Title[full$Title == 'Mlle'] <- 'Miss'
full$Title[full$Title == 'Ms'] <- 'Miss'
full$Title[full$Title == 'Mme'] <- 'Mrs'
full$Title[full$Title %in% rare_title] <- 'Rare Title'

# look at the same table again
table(full$Sex, full$Title)
```

```
##
##          Master Miss  Mr Mrs Rare Title
##   female      0  264   0 198          4
##   male       61    0 757   0         25
```

Finally, let's grab the passenger's surname from the passenger name field.

```
# This uses a function from the apply family of functions.
# We may not have had time to teach this in this short course, but the gist is
# it applies a function, the part after function(x), over an entire object.
# In this case, it splits the character string for Name and returns only the
# surname portion. In the tidyverse functions, this is done faster using the
# purrr package command called map(). For more info, help(package="purrr") and ?map.

full$Surname <- sapply(full$Name, function(x) strsplit(x, split = '[,.]')[[1]][1])
```

We have 875 unique surnames. You can include that count in-line in this document by calling the value directly from the dataset. In this case, I used `nlevels(factor(full$Surname))` as a command to ask how many (`nlevels`) unique surnames (`factor(full$Surname)`) were in the full dataset.

Do families sink or swim together?

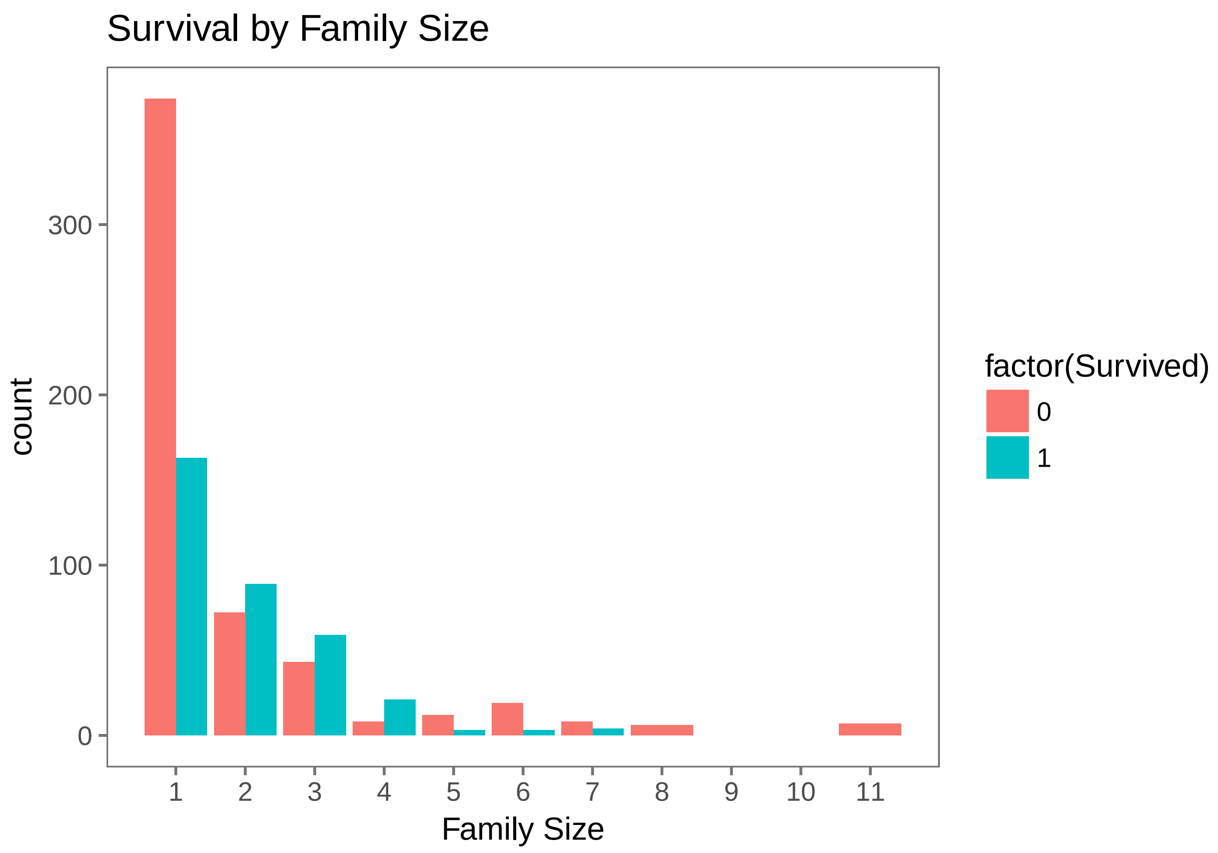
One more feature to engineer here... **family size**. We can create this by adding the number of siblings/spouses (`SibSp`) and the number of parents/children (`Parch`) together.

```
# Create a family size variable including the passenger in the count
full$Fsize <- full$SibSp + full$Parch + 1

# Create a family variable
full$Family <- paste(full$Surname, full$Fsize, sep="_")
```

What does this family size variable look like? Let's examine the variable in just the training data.

```
ggplot(full[1:891,], aes(x=Fsize, fill=factor(Survived))) +
  geom_bar(stat='count', position = 'dodge') +
  scale_x_continuous(breaks = c(1:11)) +
  labs(x='Family Size') +
  ggtitle('Survival by Family Size') +
  theme_few()
```



What stands out here? It looks like passengers travelling alone have a significant penalty to survival. Let's call these people 'singletons.' Then, things change for family sizes of 2, 3, and 4. After that, there is a survival penalty again. There seem to be three distinct typologies of families in terms of their size and survival. Let's take advantage of that and reduce family size into 3 discrete levels.

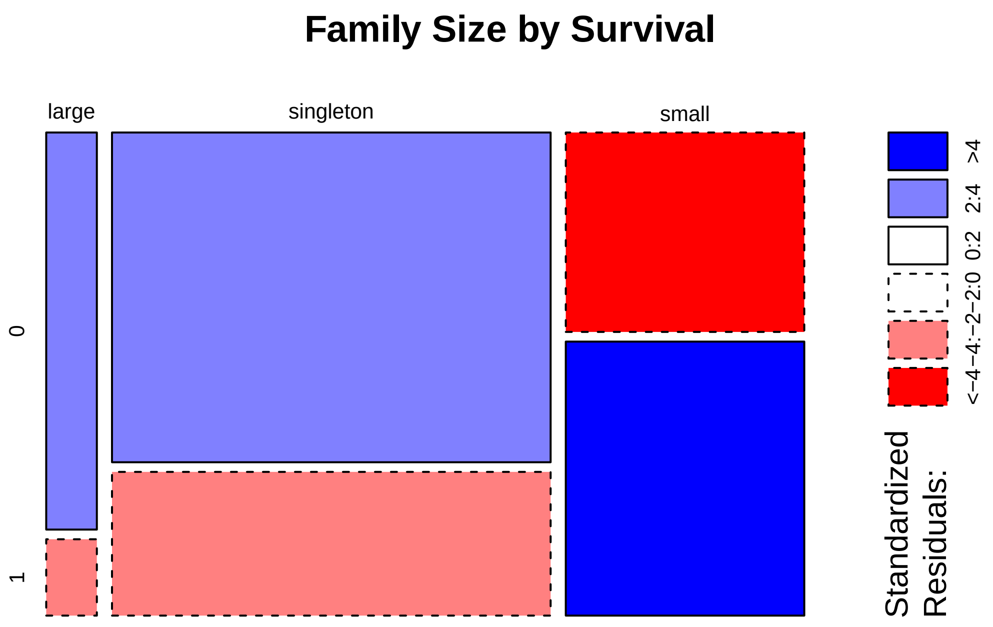
```
# discretize family size
full$FsizeD[full$Fsize==1] <- 'singleton'
```

```
## Warning: Unknown or uninitialised column: 'FsizeD'.
```

```
full$FsizeD[full$Fsize < 5 & full$Fsize > 1] <- 'small'
full$FsizeD[full$Fsize > 4] <- 'large'

# visualize this variable and the survival in a mosaic plot

mosaicplot(table(full$FsizeD, full$Survived),
           main = 'Family Size by Survival', shade = TRUE)
```



Cabin number and deck

On the Titanic, the cabin numbers were listed as a letter (Deck) and the number (Cabin). Let's create a deck variable here. You'll notice this column is missing a lot of values. We'll have to address that later during the imputation section. For now, we'll just create the deck variable from what we do have.

```
# how many missing values
length(is.na(full$Cabin))
```

```
## [1] 1309
```

```
full$Deck <- factor(sapply(full$Cabin, function(x) strsplit(x, NULL)[[1]][1]))

full$Deck[1:28] # check the first 28 rows to see what we have...
```

```
##  [1] <NA> C    <NA> C    <NA> <NA> E    <NA> <NA> <NA> G    C    <NA> <NA>
## [15] <NA> <NA> <NA> <NA> <NA> <NA> <NA> D    <NA> A    <NA> <NA> <NA> C
## Levels: A B C D E F G T
```

Missing Values and Imputation

There are a number of ways we can handle this. Casewise deletion is a bad idea given how small our dataset is, so we'll try to impute sensible values for the missing ones, or predict them based on the distribution of the values.

Sensible value imputation

Let's look at the `embarked` variable.

```
# check for missing values.
# This command will return the unique values in the Embarked variable
subset(full$Embarked, !duplicated(full$Embarked))
```

```
## [1] "S" "C" "Q" NA
```

```
# there are NAs among the other 3 embarkation points

# who is NA?

full %>% filter(is.na(Embarked)) # is.na is R's way of asking
```

```
## # A tibble: 2 x 18
##   PassengerId Survived Pclass                                      Name
##         <int>    <int>  <int>                                     <chr>
## 1          62        1      1                       Icard, Miss. Amelie
## 2         830        1      1 Stone, Mrs. George Nelson (Martha Evelyn)
## # ... with 14 more variables: Sex <chr>, Age <dbl>, SibSp <int>,
## #   Parch <int>, Ticket <chr>, Fare <dbl>, Cabin <chr>, Embarked <chr>,
## #   Title <chr>, Surname <chr>, Fsize <dbl>, Family <chr>, FsizeD <chr>,
## #   Deck <fctr>
```

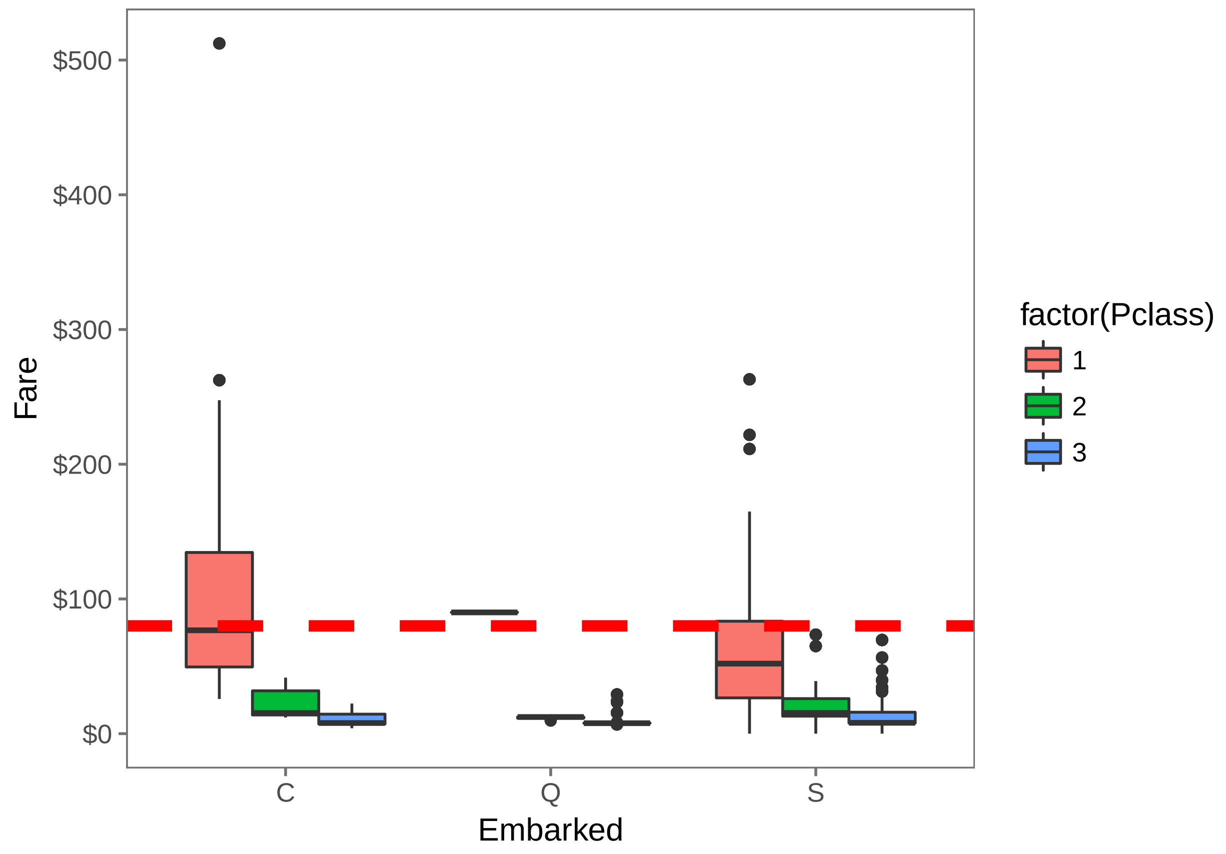
```
# if a value in a field is equal to NA (ie, missing)
```

Looks like Passenger 62 and Passenger 830 are missing their embarkation points. Let's try to infer the embark values for each passenger using other data we deem relevant: **passenger class** and **fare**.

```
# Get rid of our missing passenger IDs
embark_fare <- full %>%
  filter(PassengerId != 62 & PassengerId != 830)

# Use ggplot2 to visualize embarkment, passenger class, & median fare
ggplot(embark_fare, aes(x = Embarked, y = Fare, fill = factor(Pclass))) +
  geom_boxplot() +
  geom_hline(aes(yintercept=80),
    colour='red', linetype='dashed', lwd=2) +
  scale_y_continuous(labels=dollar_format()) +
  theme_few()
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



It looks like $80 was the fare for 1st Class passengers departing from Cherbourg (‘C’). We can probably safely replace these NA values with ‘C’.

```
full$Embarked[c(62, 830)] <- 'C'
```

```
# check for PassengerId's where the Fare is NA.
full$PassengerId[is.na(full$Fare)]
```

```
## [1] 1044
```

Passenger 1044 has a NA fare. Let’s see if we can impute the missing value based on the distribution of fare costs for passengers like this one. It looks like he embarked from Southampton (‘S’) and has a `Pclass` equal to 3, meaning he’s a 3rd class passenger.

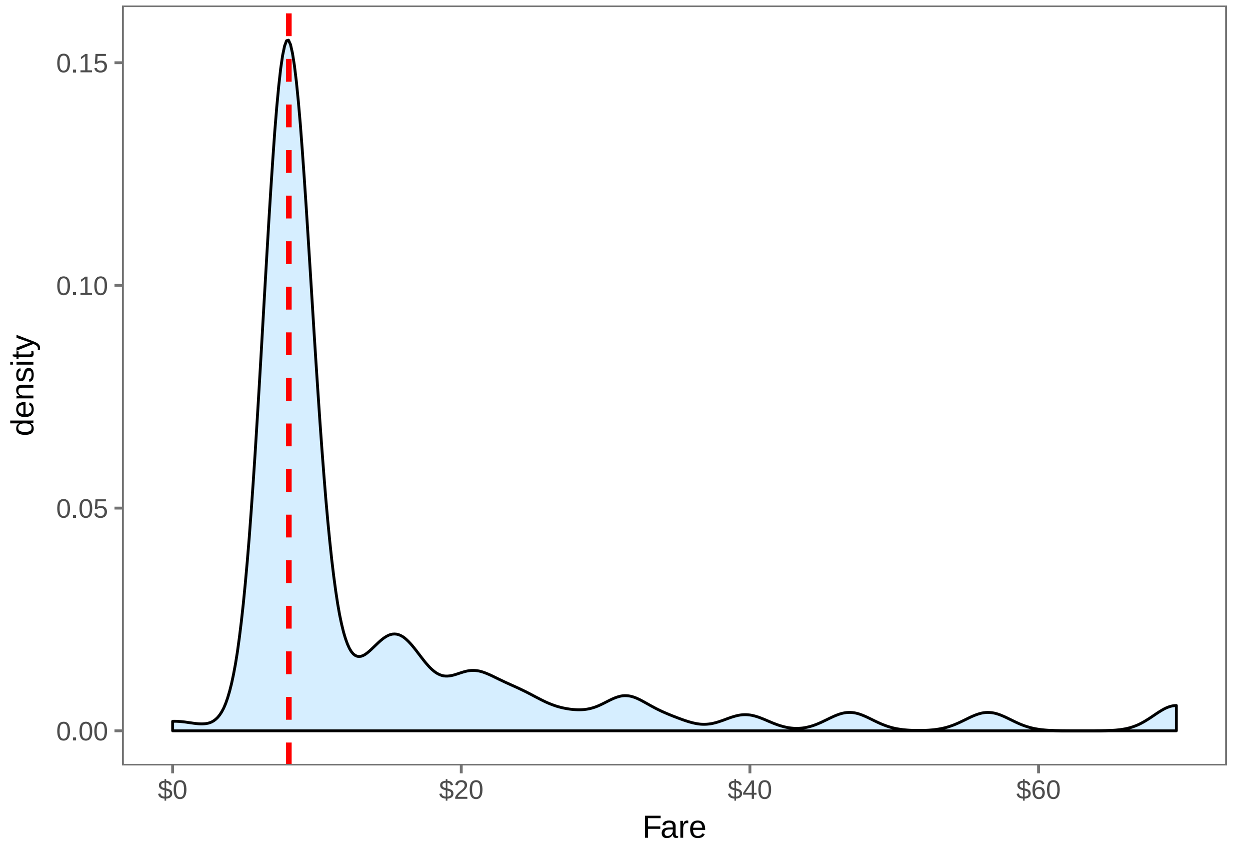
```
full[1044,] # visualize the row for this passenger
```

```
## # A tibble: 1 x 18
##   PassengerId Survived Pclass                Name   Sex   Age SibSp Parch
##         <int>    <int>  <int>               <chr> <chr> <dbl> <int> <int>
## 1        1044       NA      3 Storey, Mr. Thomas  male  60.5     0     0
## # ... with 10 more variables: Ticket <chr>, Fare <dbl>, Cabin <chr>,
## #   Embarked <chr>, Title <chr>, Surname <chr>, Fsize <dbl>, Family <chr>,
## #   FsizeD <chr>, Deck <fctr>
```

Looking at other 3rd class passengers from Southampton…

```
ggplot(full[full$Pclass == '3' & full$Embarked == 'S', ],
  aes(x = Fare)) +
  geom_density(fill = '#99d6ff', alpha=0.4) +
  geom_vline(aes(xintercept=median(Fare, na.rm=T)),
    colour='red', linetype='dashed', lwd=1) +
  scale_x_continuous(labels=dollar_format()) +
  theme_few()
```

```
## Warning: Removed 1 rows containing non-finite values (stat_density).
```



It looks like the fare is 8.05. I determined that value not from the line on the graph, but from the command `median(subset(full$Fare, full$Pclass == '3' & full$Embarked == 'S'), na.rm=TRUE)` which we used in the graph code to make the line.

From this, we can safely assume that the fare price for passenger 1044 is the median fare for people who departed from the same port and bought tickets in the same class.

```
# replace missing fare with median fare for class/embarkment

full$Fare[1044] <- median(full[full$Pclass == '3' &
                                 full$Embarked == 'S', ]$Fare,
                          na.rm=TRUE)
```

Predictive imputation

One area in our data with a lot of missing values is `age`. It has 263 missing values.

```
# show missing values
sum(is.na(full$Age))
```

```
## [1] 263
```

To fix this, we're going to use the `mice` package, which stands for "multiple imputation using chained equations." You can get more info by using the command `help(package="mice")`.

First, we need to change the class of variables from character strings to factors. This is a requirement for working in `mice`.

```r
# make variables into factors
factor_vars <- c('PassengerId', 'Pclass', 'Sex',
                 'Embarked', 'Title', 'Surname',
                 'Family', 'FsizeD')

full[factor_vars] <- lapply(full[factor_vars], function(x) as.factor(x))

# set a random seed. This allows us all to use the same randomness and get the same values.

set.seed(129)

# perform mice imputation

mice_mod <- mice(full[, !names(full) %in%
                        c('PassengerId',
                          'Name',
                          'Ticket',
                          'Cabin',
                          'Family',
                          'Surname',
                          'Survived')],
                 method = 'rf')
```

```
##
##  iter imp variable
##   1   1  Age  Deck
##   1   2  Age  Deck
##   1   3  Age  Deck
##   1   4  Age  Deck
##   1   5  Age  Deck
##   2   1  Age  Deck
##   2   2  Age  Deck
##   2   3  Age  Deck
##   2   4  Age  Deck
##   2   5  Age  Deck
##   3   1  Age  Deck
##   3   2  Age  Deck
##   3   3  Age  Deck
##   3   4  Age  Deck
##   3   5  Age  Deck
##   4   1  Age  Deck
##   4   2  Age  Deck
##   4   3  Age  Deck
##   4   4  Age  Deck
##   4   5  Age  Deck
##   5   1  Age  Deck
##   5   2  Age  Deck
##   5   3  Age  Deck
##   5   4  Age  Deck
##   5   5  Age  Deck
```

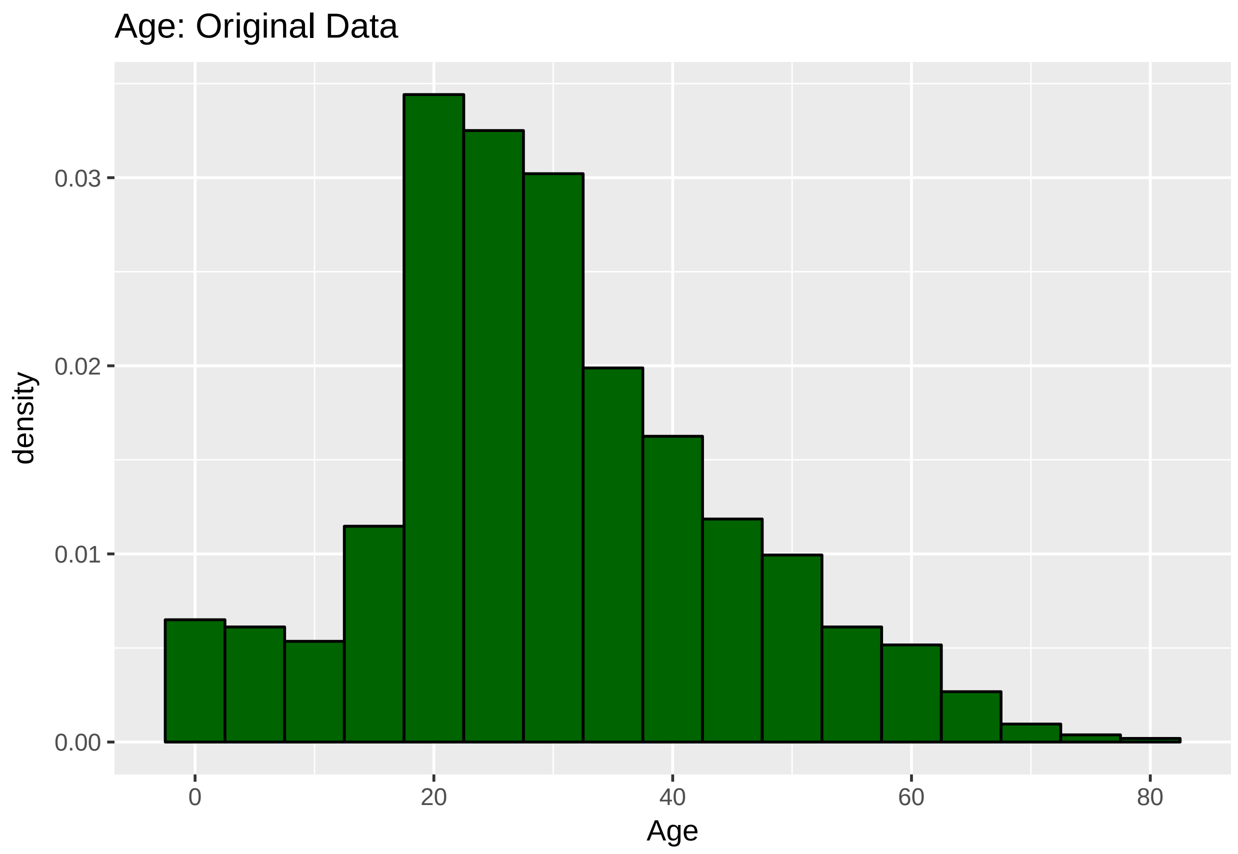
Then save the output

```
mice_output <- complete(mice_mod)
```

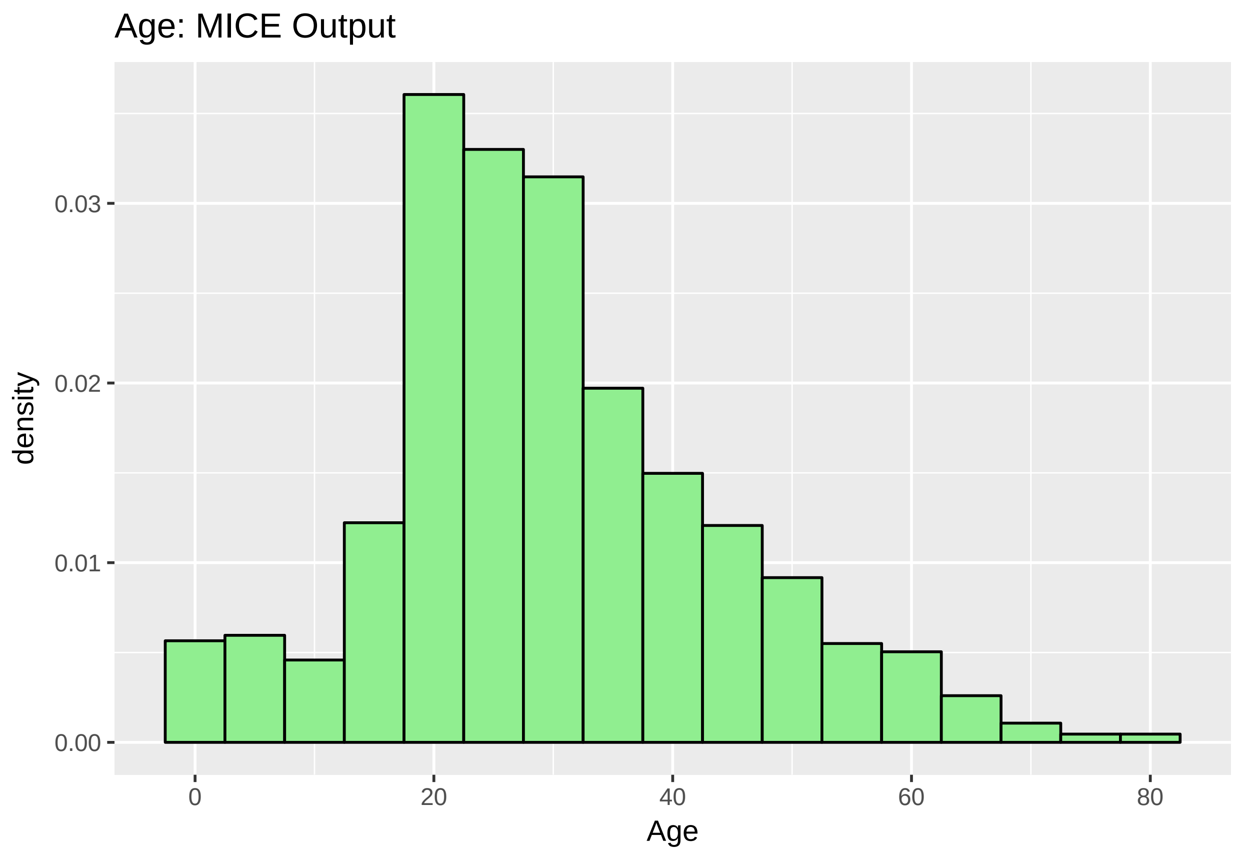
Now let's compare the imputed results with the original distribution of `age` to make sure everything looks okay.

```
# plot age distributions
ggplot(data=full) +
  geom_histogram(aes(x=Age, y=..density..),
                 binwidth = 5, color = 'black',
                 fill = 'darkgreen') +
  ggtitle('Age: Original Data')
```

```
## Warning: Removed 263 rows containing non-finite values (stat_bin).
```



```
ggplot(data=mice_output) +
  geom_histogram(aes(x=Age, y=..density..),
                 binwidth = 5, color = 'black',
                 fill = 'lightgreen') +
  ggtitle('Age: MICE Output')
```



That looks pretty close, so we'll replace our age vector in the original data with the output from the `mice` model.

```
# replace age variable from the mice model
full$Age <- mice_output$Age

# show new number of missing (NA) values
sum(is.na(full$Age))
```

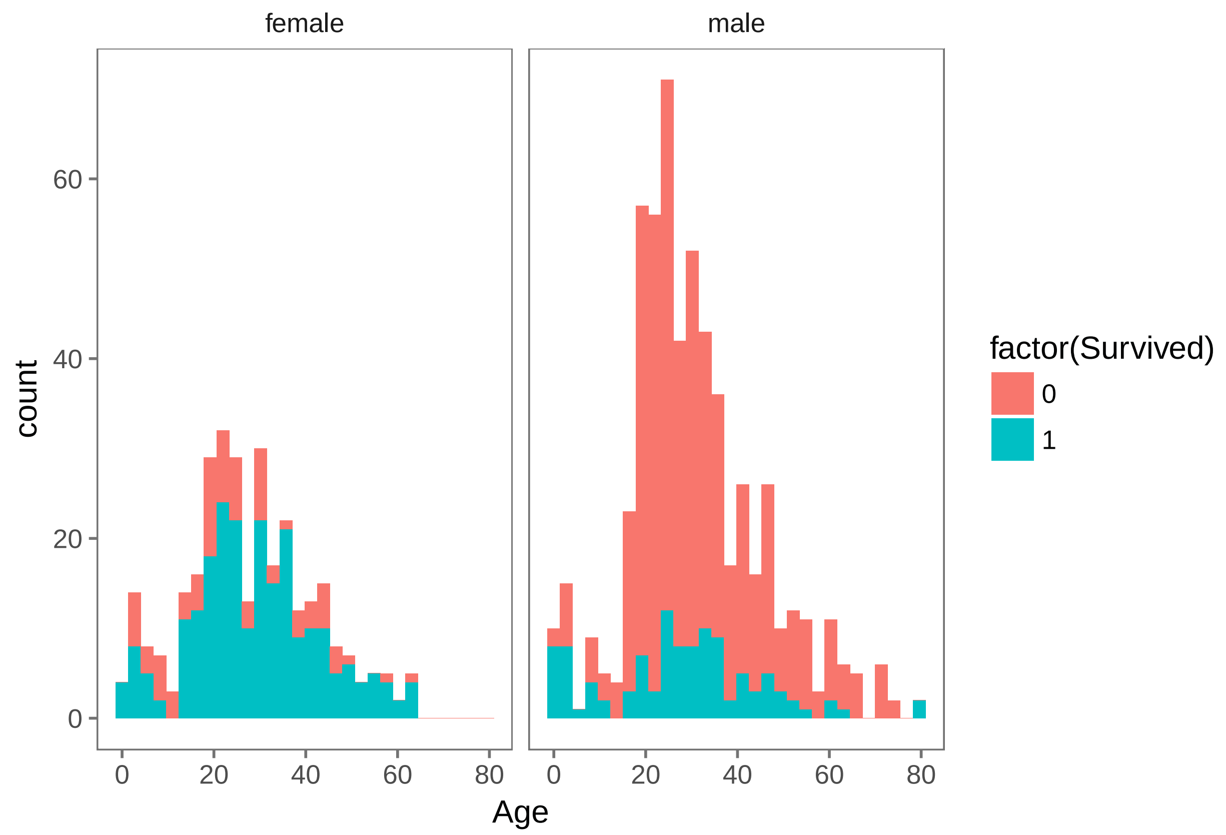
```
## [1] 0
```

Feature Engineering Part Deux

Now that we have a complete set of ages, we can compute some age-dependent variables: **Child** and **Mother**. A child will simply be someone under 18 years old, and a mother is someone who is both female and over 18, with more than 0 children, and does not have the title 'Miss.' The last criterion is something that is unique to the timing of the dataset. It is much less likely in 1912 that an unmarried woman would be traveling with children of her own than it would be in, say, 2017.

```
# first we'll look at age vs. survival
ggplot(full[1:891,], aes(Age, fill = factor(Survived))) +
  geom_histogram() +
  # sex is being included since we know, a priori, that it is a significant indicator.
  facet_grid(.~Sex) +
  theme_few()
```

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



```
# create a column child, and indicate whether or not the passenger is a child or adult
full$Child[full$Age < 18] <- 'Child'
full$Child[full$Age >=18] <- 'Adult'

# show counts
table(full$Child, full$Survived)
```

```
##
##           0   1
##   Adult 484 274
##   Child  65  68
```

So, it looks like being a child has some influence on survival, but it does not guarantee it. Now let's make the **mother** variable.

```
full$Mother <- 'Not mother' # first everyone is just assumed to not be a mother
full$Mother[full$Sex == 'female' &
              full$Parch > 0 &
              full$Age > 18 &
              full$Title !='Miss'] <- 'Mother' # if all 4 criteria are TRUE...

# show counts
table(full$Mother, full$Survived)
```

```
##
##                0   1
##   Mother      16  39
##   Not mother 533 303
```

Now let's make those new variables into factors, instead of character strings.

```
full$Child <- factor(full$Child)
full$Mother <- factor(full$Mother)
```

Let's use the `mice` command `md.pattern()` to see if there is any other missing data we need to be aware of.

```
md.pattern(full)
```

```
##     PassengerId Pclass Sex Age SibSp Parch Fare Embarked Title Surname
## 150           1      1   1   1     1     1    1        1     1       1
##  61           1      1   1   1     1     1    1        1     1       1
##  54           1      1   1   1     1     1    1        1     1       1
## 511           1      1   1   1     1     1    1        1     1       1
##  30           1      1   1   1     1     1    1        1     1       1
## 235           1      1   1   1     1     1    1        1     1       1
## 176           1      1   1   1     1     1    1        1     1       1
##  92           1      1   1   1     1     1    1        1     1       1
##               0      0   0   0     0     0    0        0     0       0
##     Fsize Family FsizeD Child Mother Ticket Survived Deck Name Cabin
## 150     1      1      1     1      1      1        1    1    0     0    2
##  61     1      1      1     1      1      1        0    1    0     0    3
##  54     1      1      1     1      1      0        1    1    0     0    3
## 511     1      1      1     1      1      1        1    0    0     0    3
##  30     1      1      1     1      1      0        0    1    0     0    4
## 235     1      1      1     1      1      1        0    0    0     0    4
## 176     1      1      1     1      1      0        1    0    0     0    4
##  92     1      1      1     1      1      0        0    0    0     0    5
##         0      0      0     0      0    352      418 1014 1309  1309 4402
```

Building the model

First we need to split our data up into training and test sets.

```
# split data back into training and test sets
train <- full[1:891,]
test <- full[892:1309,]
```

Now let's build our regression regression model.

```
# Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked


fit <- glm(Survived~Age+Pclass+Sex+SibSp+Parch+Fare+Embarked,
           data=train, family =binomial(link="logit"))

summary(fit)
```

```
##
## Call:
## glm(formula = Survived ~ Age + Pclass + Sex + SibSp + Parch +
##     Fare + Embarked, family = binomial(link = "logit"), data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6219  -0.6101  -0.4007   0.6008   2.5175
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.176801   0.483871   8.632  < 2e-16 ***
## Age         -0.037599   0.007506  -5.009 5.47e-07 ***
## Pclass2     -1.012118   0.304130  -3.328 0.000875 ***
## Pclass3     -2.285076   0.309096  -7.393 1.44e-13 ***
## Sexmale     -2.711495   0.201966 -13.425  < 2e-16 ***
## SibSp       -0.361328   0.109625  -3.296 0.000981 ***
## Parch       -0.086299   0.118763  -0.727 0.467441
## Fare         0.001817   0.002444   0.743 0.457282
## EmbarkedQ   -0.026795   0.384372  -0.070 0.944424
## EmbarkedS   -0.438481   0.238588  -1.838 0.066090 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1186.66  on 890  degrees of freedom
## Residual deviance:  783.65  on 881  degrees of freedom
## AIC: 803.65
##
## Number of Fisher Scoring iterations: 5
```

What if we replaced the Sex field with our engineered field, “Title” and re-ran the model? Would we have any improvement in the model?

```r
fit2 <- glm(Survived~Age+Pclass+SibSp+Parch+Fare+Embarked+Title,
            data=train, family =binomial(link="logit"))
summary(fit2)
```

```
##
## Call:
## glm(formula = Survived ~ Age + Pclass + SibSp + Parch + Fare +
##     Embarked + Title, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3877  -0.5516  -0.3772   0.5442   2.4670
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      4.420415   0.642833   6.876 6.14e-12 ***
## Age             -0.026189   0.008796  -2.977  0.00291 **
## Pclass2         -1.161432   0.327444  -3.547  0.00039 ***
## Pclass3         -2.255423   0.329069  -6.854 7.18e-12 ***
## SibSp           -0.568765   0.126451  -4.498 6.86e-06 ***
## Parch           -0.354851   0.134236  -2.643  0.00821 **
## Fare             0.003136   0.002625   1.195  0.23226
## EmbarkedQ       -0.065337   0.395671  -0.165  0.86884
## EmbarkedS       -0.411544   0.249509  -1.649  0.09906 .
## TitleMiss       -0.564950   0.501507  -1.127  0.25995
## TitleMr         -3.520730   0.540593  -6.513 7.38e-11 ***
## TitleMrs         0.240031   0.559382   0.429  0.66785
## TitleRare Title -3.124556   0.725942  -4.304 1.68e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1186.66  on 890  degrees of freedom
## Residual deviance:  730.26  on 878  degrees of freedom
## AIC: 756.26
##
## Number of Fisher Scoring iterations: 5
```

```r
anova(fit, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                       890    1186.66
## Age       1    2.236       889    1184.42    0.1348
## Pclass    2  140.261       887    1044.16 < 2.2e-16 ***
## Sex       1  236.687       886     807.47 < 2.2e-16 ***
## SibSp     1   17.936       885     789.53 2.284e-05 ***
## Parch     1    0.441       884     789.09    0.5064
## Fare      1    1.150       883     787.94    0.2836
## Embarked  2    4.294       881     783.65    0.1168
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(fit2, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                       890    1186.66
## Age       1    2.236       889    1184.42   0.13483
## Pclass    2  140.261       887    1044.16 < 2.2e-16 ***
## SibSp     1    4.555       886    1039.60   0.03282 *
## Parch     1    6.424       885    1033.18   0.01126 *
## Fare      1    4.478       884    1028.70   0.03433 *
## Embarked  2   18.801       882    1009.90 8.268e-05 ***
## Title     4  279.642       878     730.26 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It looks like `Title` outpreformed the more general `Sex` field and reduced our residual deviance (and AIC).

We can use the `prcl` package to compute McFadden's pseudo-R^2 which is roughly equivalent to R^2 in OLS.

```
pR2(fit)
```

```
##          llh      llhNull           G2     McFadden         r2ML
## -391.8246081 -593.3275684  403.0059207    0.3396150    0.3638414
##         r2CU
##    0.4943466
```

```
pR2(fit2)
```

```
##          llh      llhNull           G2     McFadden         r2ML
## -365.1286420 -593.3275684  456.3978529    0.3846087    0.4008427
##         r2CU
##    0.5446197
```

We could continue tweaking the model but for the purposes of this lab let's accept our `fit2` model and move on to prediction.

```
fitted.results <- predict(fit2,newdata=
                          subset(test,select=c(3,6,7,8,10,12,13)),
                        type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)

test2 <- test %>% select(-Survived) %>%
  mutate(PassengerId = as.numeric(PassengerId)) %>%
  left_join(results, by="PassengerId")

misClasificError <- mean(fitted.results != test2$Survived)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.916267942583732"
```

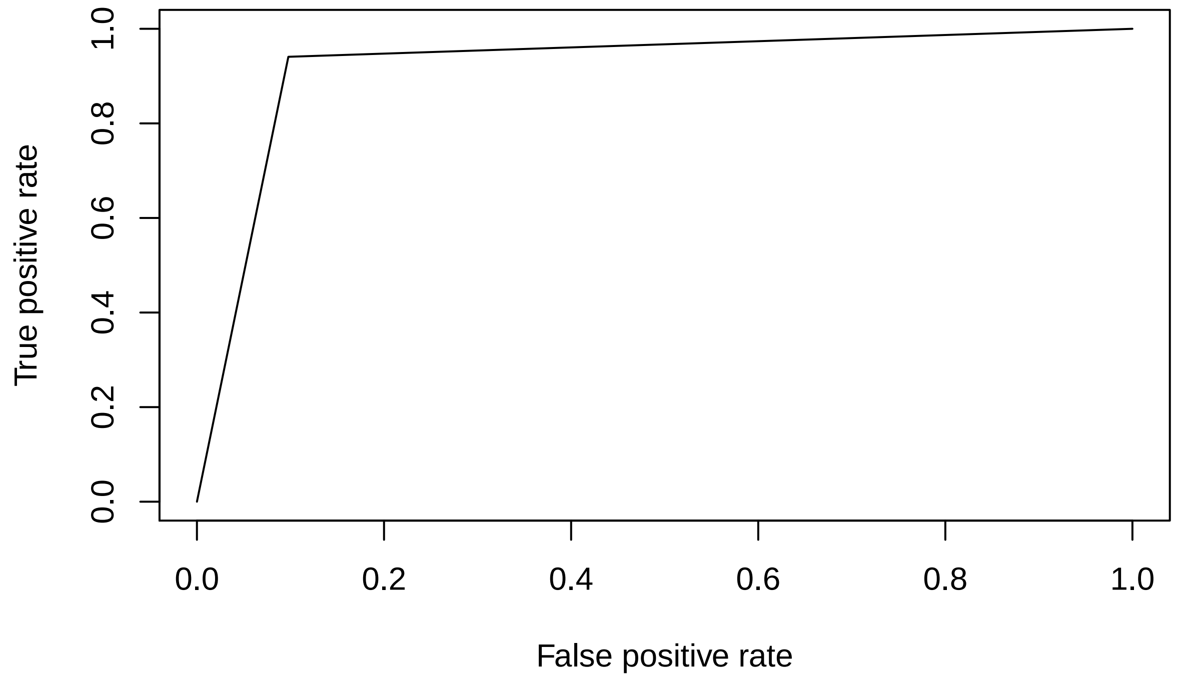
Wow. 91.6% predictive accuracy. It's almost as if this tutorial was *engineered* for a class lab exercise that produced great results. Let's take one last moment and look at the other prediction results using `caret` and `ROCR` pacakges to visualize a confusion matrix and graph the area under the ROC.

```
# caret package
confusionMatrix(data=fitted.results, reference=test2$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 240   9
##          1  26 143
##
##                Accuracy : 0.9163
##                  95% CI : (0.8855, 0.941)
##     No Information Rate : 0.6364
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                    Kappa : 0.8233
##  Mcnemar's Test P-Value : 0.006841
##
##             Sensitivity : 0.9023
##             Specificity : 0.9408
##          Pos Pred Value : 0.9639
##          Neg Pred Value : 0.8462
##              Prevalence : 0.6364
##          Detection Rate : 0.5742
##    Detection Prevalence : 0.5957
##       Balanced Accuracy : 0.9215
##
##        'Positive' Class : 0
##
```

```
# ROCR package
pr <- prediction(fitted.results, test2$Survived)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```



```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.9215226
```