CPTR330 – Final Project

Ryan Rabello

June 6, 2017

1 Titanic Data

1.1 Step 1 - Collecting the data

This dataset has been made available to us by kaggle. Full details to the dataset can be found here.

1.2 Importing data.

First things first we need to do is import our data to take a look at it.

```
# This function gets the target csv file from the url. If
# that fails it uses a copy of the files stored locally.
get <- function(fileName) {

    return(read.csv(paste0("https://cs.wallawalla.edu/~carmpr/cptr330/titanic/",
        fileName), stringsAsFactors = TRUE))

    # If the file is sucessfully found on the server download it,
    # otherwise use a local copy of the files. file <- tryCatch({
    # return(read.csv(paste0('https://cs.wallawalla.edu/~carmpr/cptr330/titanic/',
    # fileName), stringsAsFactors = TRUE)) }, error = function(e)
    # { return(read.csv(paste0('./data/titanic/', fileName),
    # stringsAsFactors = TRUE)) }) return(file)
}
# The train.csv contains all features and is used to train
# models.
raw_train <- get("train.csv")

# The test.csv is not going to change during the final time.
raw_test <- get("test.csv")

# The test_final.csv is what is going to be used (like a
# kaggle submission) to grade the performance of our
# algorithm.
raw_test_final <- get("test_final.csv")

# The test_results.csv contains the labels (or answers) to
# test.csv.
raw_test_results <- get("test_results.csv")

# The test_results_final.csv contains the labels for
# test_final.csv.
raw_test_results_final <- get("test_results_final.csv")

# Take a look at the data.
str(raw_train)
```

```
## 'data.frame':    891 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       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 520 629 417 58
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ 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     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

1.3 Step 2 - Exploring And Preparing The Data

The following list briefly explains what each variable is and what type (categorical or regression) it is.

- **Survived** A Boolean (0 or 1) indicating the survival of this particular passenger. (Categorical)
- **pclass** Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd (A numerical representation of the) (Categorical)
- **sex** Gender of the individual. (Categorical)
- **Age** Age in floating point years. (Regression)
- **sibsp** # of siblings / spouses aboard the Titanic (regression)
- **parch** # of parents / children aboard the Titanic (regression)
- **ticket** Ticket number (ID)
- **fare** Passenger fare (the cost of the ticket). (Regression)
- **cabin** Cabin number (String ex “A15” and “B12”) (Categorical)
- **embarked** Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton (Categorical)

Because `PassengerId`, `Name` and `ticket` are mostly unique we they will not be useful in any machine learning calculation and are therefore nullified. I’m also changing some of the names of variables so that everything works okay. Lastly I’m separating the `Cabin` variable into two variables `Floor` and `Room`.

```r
process <- function(titanic) {
    # Nullify unique feilds (they are still stored for later
    # use.)
    titanic$PassengerId <- NULL
    titanic$Name <- NULL
    titanic$Ticket <- NULL

    # Convert some variables to factors
    levels(titanic$Embarked) <- c("Missing", "Cherbourg", "Queenstown",
        "Southampton")
    titanic$Pclass <- as.factor(titanic$Pclass)
    if ("Survived" %in% names(titanic)) {
        titanic$Survived <- factor(titanic$Survived, c(0, 1),
            c("Died", "Survived"))
    }
    # Remove the suprise variable.
    if ("X" %in% names(titanic)) {
        titanic$X <- NULL
    }
```

```
    # Convert Cabin to `Room` (this line will introduce NAs)
    titanic$Room <- as.numeric(substring(as.character(titanic$Cabin),
        2))

    # Convert cabin to 'floor' (the first letter of each cabin)
    cabinLev <- levels(titanic$Cabin)
    cabinLev <- substr(cabinLev, 1, 1)
    levels(titanic$Cabin) <- cabinLev
    levels(titanic$Cabin) <- c("?", "A", "B", "C", "D", "E",
        "F", "G", "T")
    titanic$Floor <- titanic$Cabin

    # Remove Cabin becuase it is represented above.
    titanic$Cabin <- NULL

    return(titanic)
}

# Process the new data.
train <- process(raw_train)
```

```
## Warning in process(raw_train): NAs introduced by coercion
```

```
test <- process(raw_test)
```

```
## Warning in process(raw_test): NAs introduced by coercion
```

```
test_final <- process(raw_test_final)
```

```
## Warning in process(raw_test_final): NAs introduced by coercion
```

```
#
test_results <- factor(raw_test_results$Survived, c(0, 1), c("Died",
    "Survived"))
test_results_final <- factor(raw_test_results_final$Survived,
    c(0, 1), c("Died", "Survived"))

test_all <- cbind(Survived = test_results_final, test_final)

titanic <- rbind(train, test_all)
```

1.4 Global Functions

A global function for exporting data. `data` is a vector of answers and `name` the name of the csv that will be exported. (ex. name = "tree" -> "tree.csv")

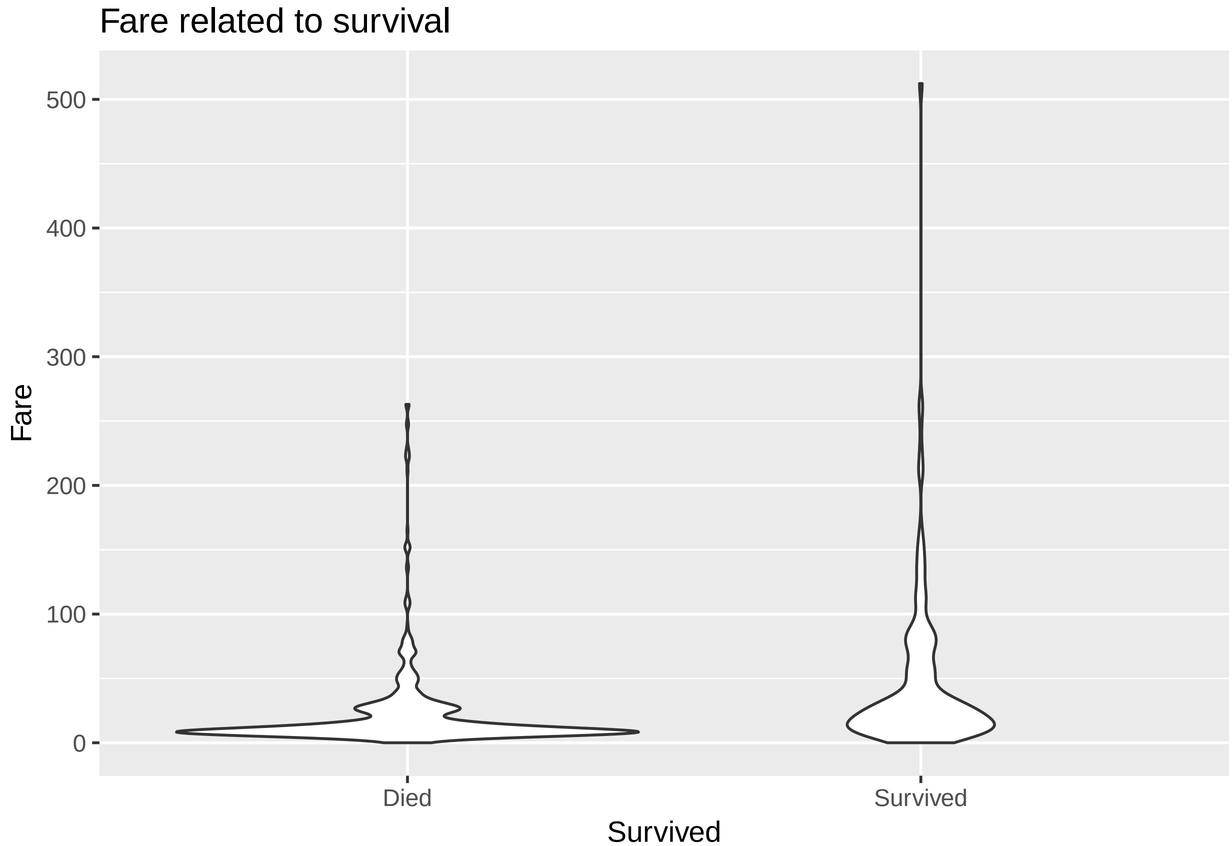
```
export <- function(data, name) {
    kaggle <- data.frame(PassengerId = raw_test_final$PassengerId,
        Survived = as.numeric(data) - 1)
    write.csv(x = kaggle, file = paste("./exports/", name, ".csv",
        sep = ""), row.names = F, quote = F)
}

# Dynamically install/import a package
dynInstall <- function(package) {
```

```
    if (package %in% rownames(installed.packages())) {
        do.call("library", list(package))
    } else {
        install.packages(package)
        do.call("library", list(package))
    }
}
```

1.5 Exploring the data

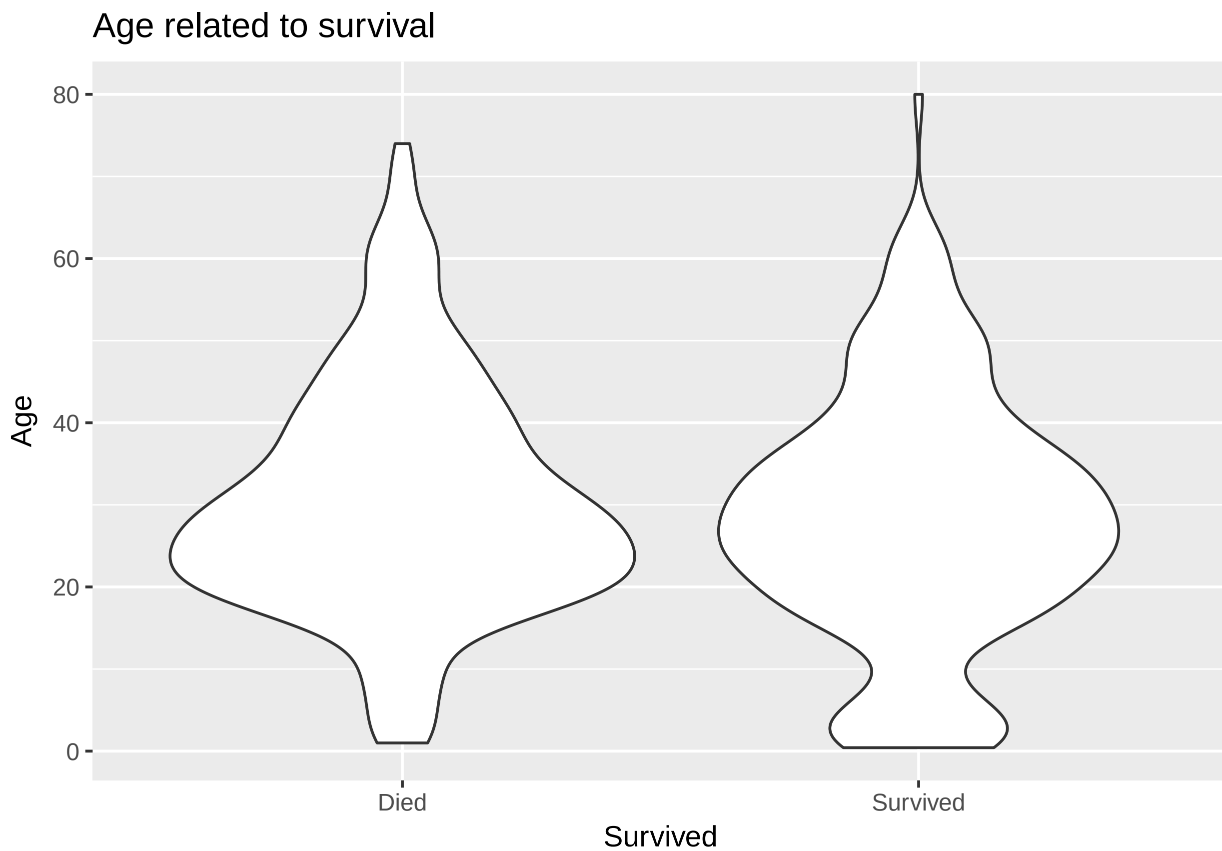
```
dynInstall("ggplot2")

dynInstall("ggthemes")

# Styles for the graphs.  ggplot <- function(...)
# ggplot2::ggplot(...) + scale_color_fivethirtyeight('cyl') +
# theme_fivethirtyeight() ggplot <- function(...)
# ggplot2::ggplot(...) + theme_hc() + scale_colour_hc()

# Graph some continuous variables in a 'Violin' graph.
ggplot(titanic, aes(Survived, Fare)) + geom_violin(scale = "area") +
    ggtitle("Fare related to survival")
```

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

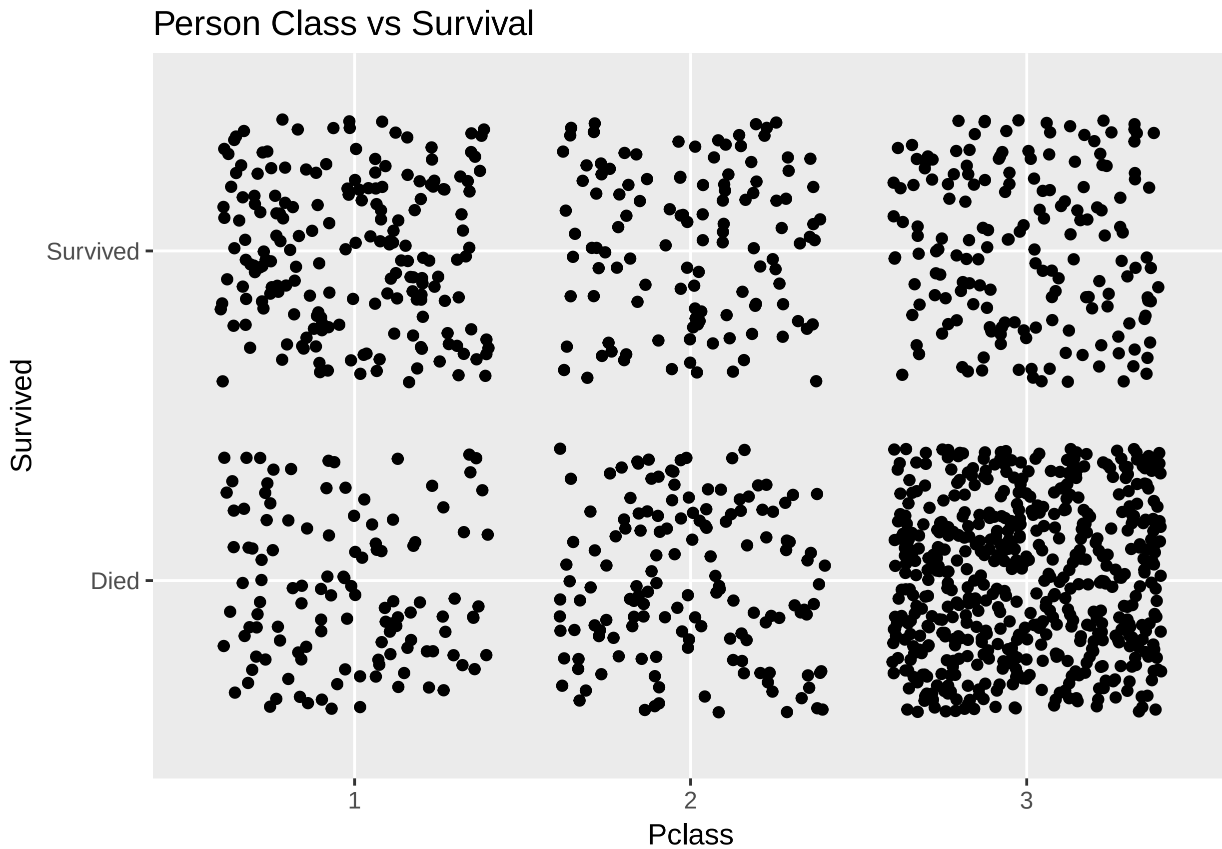


```r
ggplot(titanic, aes(Survived, Age)) + geom_violin(scale = "area") +
    ggtitle("Age related to survival")
```

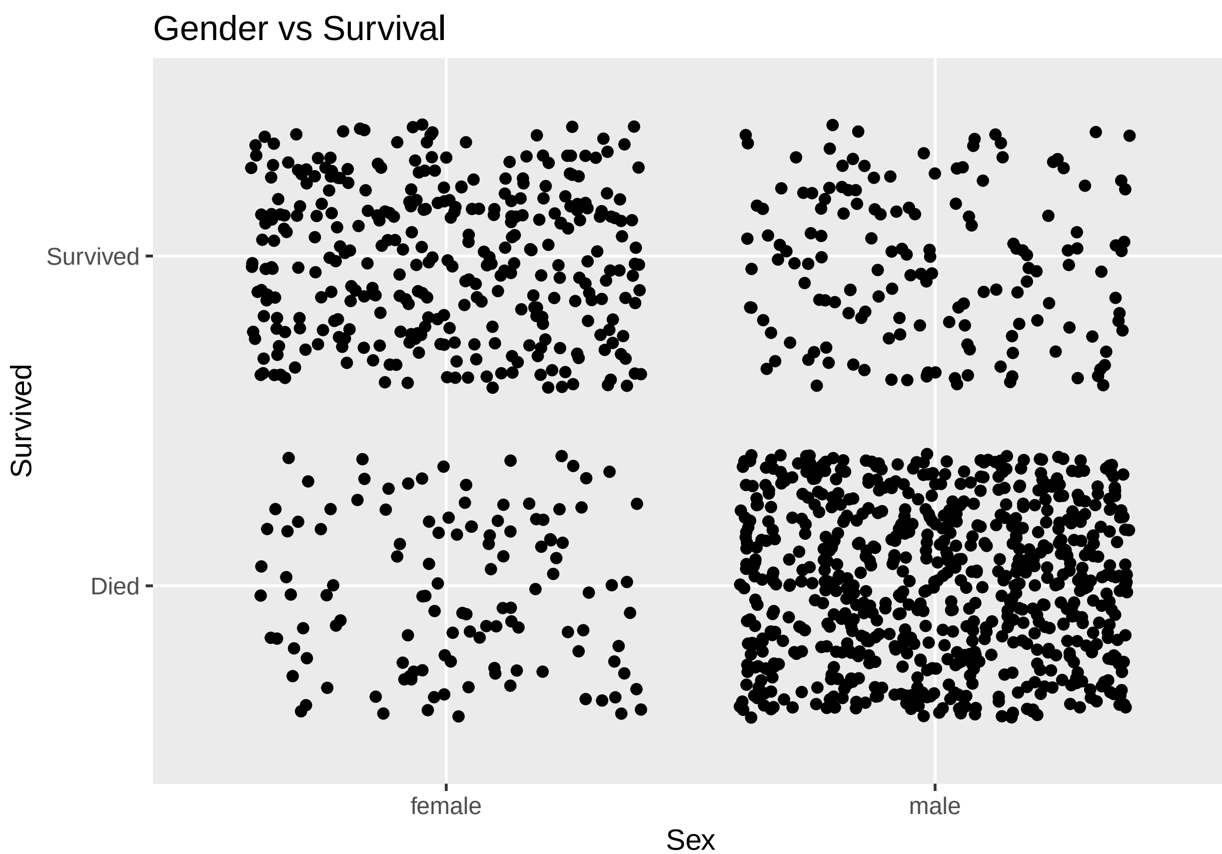
```
## Warning: Removed 266 rows containing non-finite values (stat_ydensity).
```




```r
# Plot a jitter graph for the discrete variables.
ggplot(titanic, aes(Pclass, Survived)) + geom_jitter() + ggtitle("Person Class vs Survival")
```

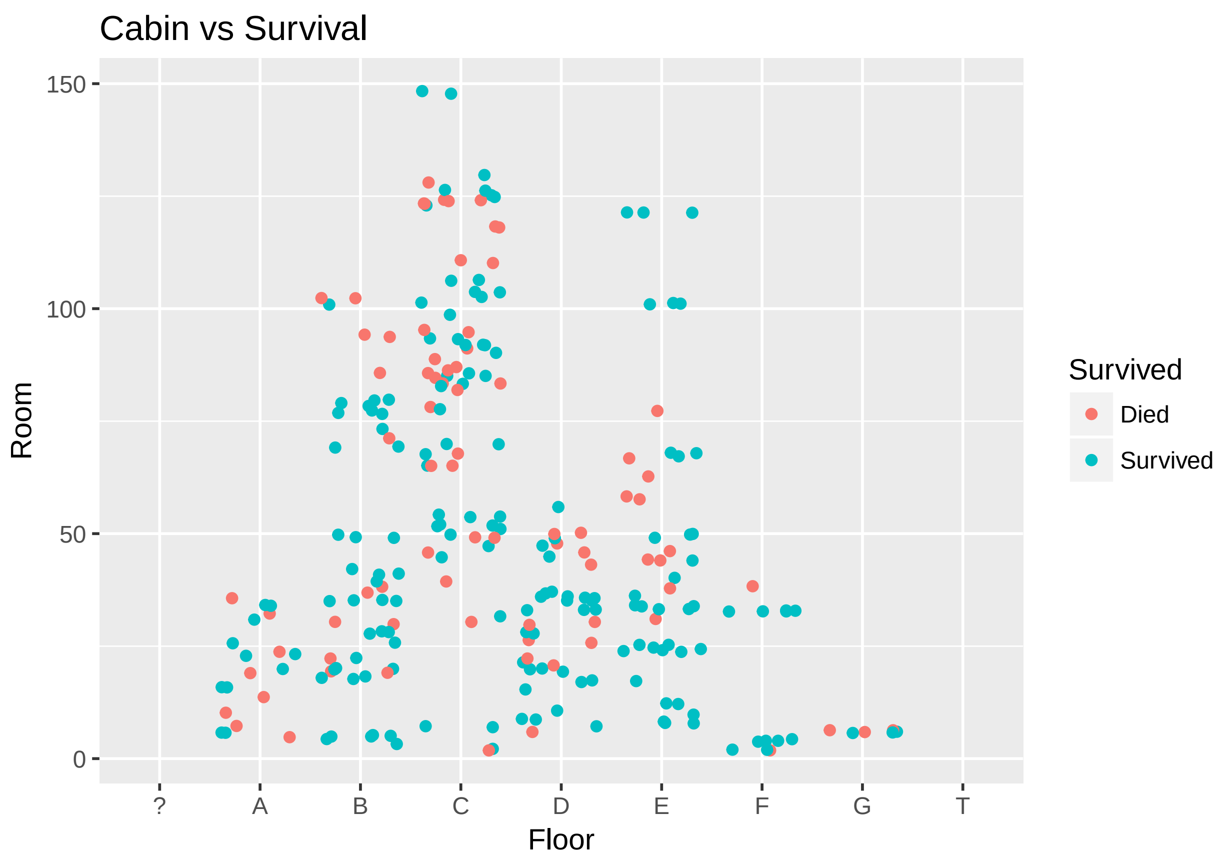



```
ggplot(titanic, aes(Sex, Survived)) + geom_jitter() + ggtitle("Gender vs Survival")
```

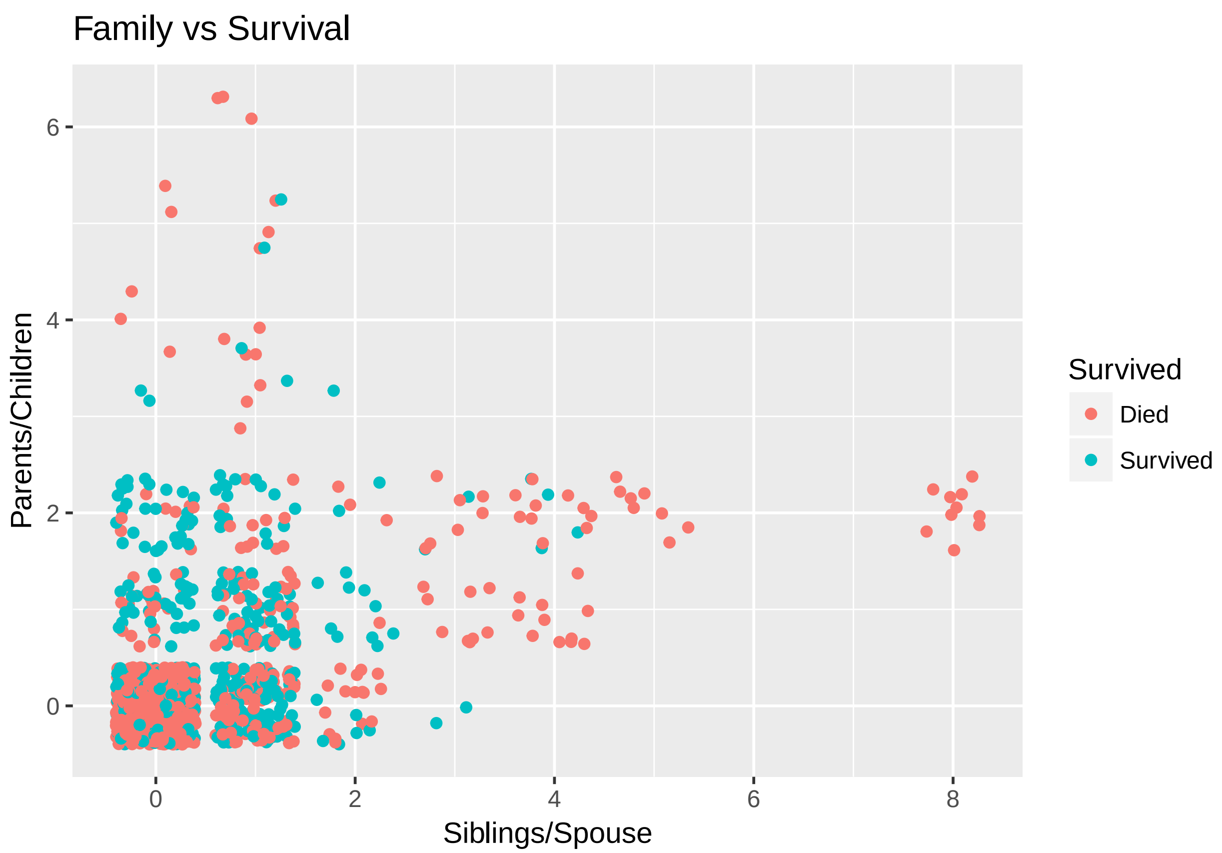



```r
# These are '3d' graphs that show two variables and the
# classificiation (survived or didn't survive) is shown as
# the color of the point.
ggplot(titanic, aes(x = Floor, y = Room, colour = Survived)) +
    geom_jitter() + ggtitle("Cabin vs Survival")
```

```
## Warning: Removed 1059 rows containing missing values (geom_point).
```

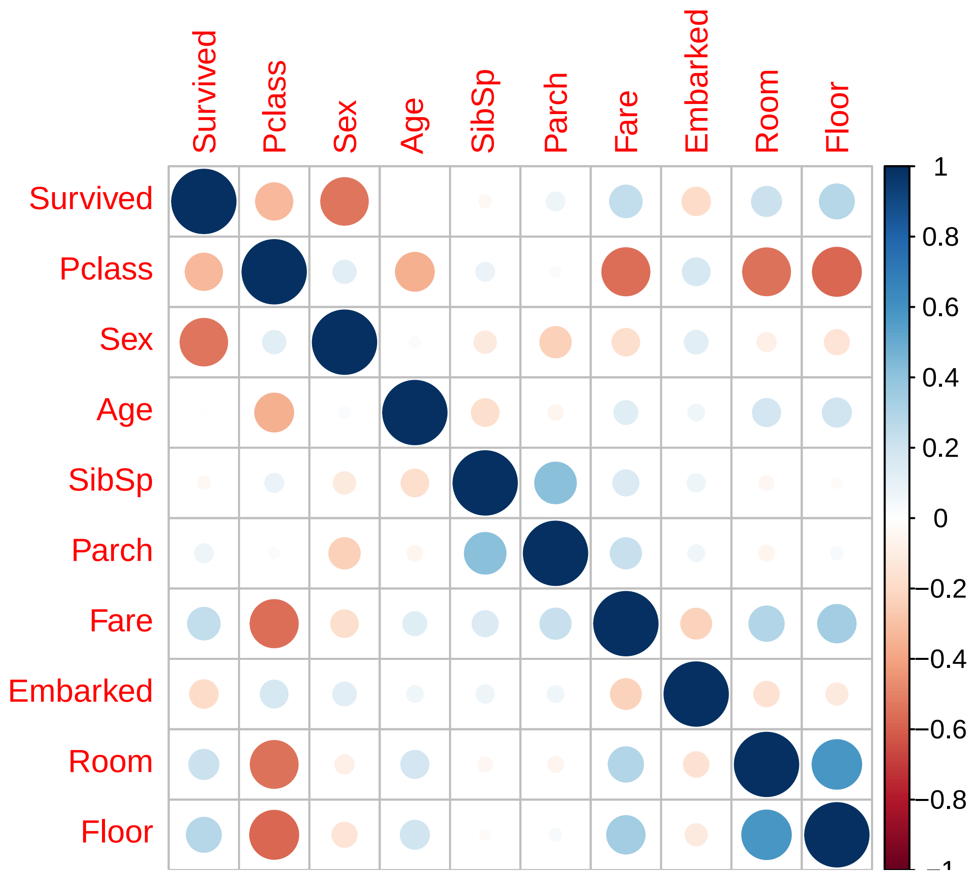



```
ggplot(titanic, aes(x = SibSp, y = Parch, colour = Survived)) +
    geom_jitter() + ggtitle("Family vs Survival") + labs(x = "Siblings/Spouse",
    y = "Parents/Children")
```



All the graphs (except the gender graph) above seem to indicate that the titanic data doesn't have a very recognizable pattern. In other words there is a lot of noise. The algorithms that will then work better are ones like Decision Trees, Neural Networks, and random forest. Also because so many variables in this dataset are discrete a feature ML algorithm like Naive Bayes should also be considered.

```
# Correlation
dynInstall("corrplot")

# Replace NA with the mean of that row.
matrix <- lapply(titanic, function(x) {
    x <- as.numeric(x)
    avg <- mean(x, na.rm = T)
    x[is.na(x)] <- avg
    return(x)
})

matrix <- data.matrix(titanic, rownames.force = T)

# Somehow NAs were introduced to lets set those to zero.
matrix[is.na(matrix)] <- 0

# Calculate the correlation matrix.
corr_titanic <- cor(matrix)

# Graph the correlation matrix.
corrplot(corr_titanic, method = "circle")
```



All of variables (excluding gender) are not linearly dependent on the graph. This means that a machine learning algorithm like a linear regression will not work very well for this dataset.

2 Step 3 - Training A Model On The Data

Of the models I tested out the most successful one was the Decision Tree. The other models are included here for clarity. These other models include: Naive bays, which performed the worse out of the four that I tested; Neural Networks, performed the second best right below decision trees; and Random Forest, which I wasn't able to predict any values for.

3 Descision Trees

3.1 Training

```
dynInstall("C50")

# Train the model.
tree <- C5.0(train[, -1], train$Survived, trials = 5)

summary(tree)
```

```
##
## Call:
```

```
## C5.0.default(x = train[, -1], y = train$Survived, trials = 5)
##
##
## C5.0 [Release 2.07 GPL Edition]      Tue Jun 06 00:01:03 2017
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 891 cases (10 attributes) from undefined.data
##
## -----  Trial 0:  -----
##
## Decision tree:
##
## Sex = male:
## :...Age > 9: Died (536.2/88.9)
## :   Age <= 9:
## :   :...SibSp > 2: Died (14.4/1)
## :       SibSp <= 2:
## :       :...Parch <= 0: Died (8.2/1)
## :           Parch > 0: Survived (18.2/0.1)
## Sex = female:
## :...Pclass in {1,2}: Survived (170/9)
##     Pclass = 3:
##     :...Embarked = Missing: Survived (0)
##         Embarked = Queenstown:
##         :...Parch <= 0: Survived (30/6)
##         :   Parch > 0: Died (3)
##         Embarked = Cherbourg:
##         :...Fare > 15.2458: Survived (7)
##         :   Fare <= 15.2458:
##         :   :...Fare <= 13.8625: Survived (6)
##         :       Fare > 13.8625: Died (10/2)
##         Embarked = Southampton:
##         :...Fare > 20.575: Died (29/3)
##             Fare <= 20.575:
##             :...Parch <= 0: Died (45/20)
##                 Parch > 0: Survived (14/4)
##
## -----  Trial 1:  -----
##
## Decision tree:
##
## Sex = female:
## :...Pclass in {1,2}: Survived (147.3/19.4)
## :   Pclass = 3:
## :   :...Fare <= 23.25: Survived (137.7/51.7)
## :       Fare > 23.25: Died (24.2/5.1)
## Sex = male:
## :...Pclass in {2,3}: Died (426.6/115.9)
##     Pclass = 1:
##     :...Fare <= 26: Died (7.9)
##         Fare > 26: Survived (147.3/53.2)
##
```

```
## -----  Trial 2:  -----
##
## Decision tree:
##
## Sex = female:
## :...Pclass in {1,2}: Survived (132.2/24.1)
## :   Pclass = 3:
## :   :...Parch <= 0: Survived (112.6/54.3)
## :       Parch > 0: Died (59.7/21.1)
## Sex = male:
## :...Age <= 13: Survived (50.6/17.7)
##     Age > 13:
##     :...Embarked in {Missing,Queenstown,Southampton}: Died (423.7/142.1)
##         Embarked = Cherbourg:
##         :...SibSp <= 0: Died (81.8/35.4)
##             SibSp > 0: Survived (30.4/11.9)
##
## -----  Trial 3:  -----
##
## Decision tree:
##
## Fare <= 10.5167: Died (285.4/44.8)
## Fare > 10.5167:
## :...SibSp > 2: Died (48.7/6)
##     SibSp <= 2:
##     :...Sex = female: Survived (184.1/41.2)
##         Sex = male:
##         :...Pclass = 1:
##             :...Age <= 43: Survived (121.9/37.1)
##             :   Age > 43: Died (87.5/37.3)
##             Pclass in {2,3}:
##             :...Age <= 14: Survived (26.1/4.4)
##                 Age > 14: Died (83.4/6.2)
##
## -----  Trial 4:  -----
##
## Decision tree:
##
## Sex = female:
## :...Pclass in {1,2}: Survived (99.7)
## :   Pclass = 3:
## :   :...Fare <= 23.25: Survived (204.5/65.6)
## :       Fare > 23.25: Died (30.1/1)
## Sex = male:
## :...Fare <= 8.4583: Died (115.6)
##     Fare > 8.4583:
##     :...SibSp > 2: Died (27.7)
##         SibSp <= 2:
##         :...Age <= 10: Survived (23.5/3.4)
##             Age > 10:
##             :...Pclass in {2,3}: Died (95/2.1)
##                 Pclass = 1:
##                 :...Age > 49: Died (35.1/5.2)
##                     Age <= 49:
```

```
##                       :...Room <= 54: Survived (107.2/46.6)
##                           Room > 54: Died (67.6/22.9)
##
##
## Evaluation on training data (891 cases):
##
## Trial        Decision Tree
## -----      ----------------
##    Size      Errors
##
##    0     13  135(15.2%)
##    1      6  191(21.4%)
##    2      7  175(19.6%)
##    3      7  182(20.4%)
##    4     10  143(16.0%)
## boost            137(15.4%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     484    65    (a): class Died
##      72   270    (b): class Survived
##
##
##  Attribute usage:
##
##  100.00% Pclass
##  100.00% Sex
##  100.00% Fare
##   76.77% Embarked
##   74.75% SibSp
##   50.84% Age
##   31.54% Parch
##    6.62% Room
##
##
## Time: 0.0 secs
```

```
# plot(tree)
```

3.2 Step 4 - Evaluating Model Performance

```
tree_pred <- function(test, answer) {
    prediction <- predict(tree, test)
    xtab <- table(prediction, answer)
    return(xtab)
}

tree_test <- tree_pred(test, test_results)
tree_test
```

```
##             answer
## prediction Died Survived
```

```
##   Died      242       7
##   Survived   24     145
```

```
tree_test_final <- tree_pred(test_final, test_results_final)
tree_test_final
```

```
##           answer
## prediction Died Survived
##   Died      239      35
##   Survived   37     107
```

The initial testing of the Decision tree was very good however, I was able to increase it's accuracy a little bit by increasing the number of trials and a couple other parameters as seen below.

3.3 Step 5 - Improving Model Performance

```
tree <- C5.0(train[, -1], train$Survived, trials = 10, control = C5.0Control(bands = 50),
    rules = T)

# summary(tree)


tree_pred <- function(test, answer) {
    prediction <- predict(tree, test)
    xtab <- table(prediction, answer)
    return(xtab)
}


tree2_test <- tree_pred(test, test_results)
tree2_test
```

```
##           answer
## prediction Died Survived
##   Died      247       9
##   Survived   19     143
```

```
tree2_test_final <- tree_pred(test_final, test_results_final)
tree2_test_final
```

```
##           answer
## prediction Died Survived
##   Died      241      35
##   Survived   35     107
```

3.4 Kaggle Exporting

```
tree_pred <- predict(tree, test_final)
export(tree_pred, "tree")
```

4 Final Scoring

```
# Kaggle Score
tree2_test
```

```
##           answer
## prediction Died Survived
##   Died      247        9
##   Survived   19      143
```

```
accuracy_kaggle <- sum(diag(tree2_test))/sum(tree2_test)
accuracy_kaggle
```

```
## [1] 0.9330144
```

```
# Final Score
tree2_test_final
```

```
##           answer
## prediction Died Survived
##   Died      241       35
##   Survived   35      107
```

```
accuracy_final <- sum(diag(tree2_test_final))/sum(tree2_test_final)
accuracy_final
```

```
## [1] 0.8325359
```

5 Other Algorithms

Much work was put into increasing the accuracy of these algorithms. However, they didn't perform better than decision trees so they are included here for clarity.

5.1 Naive Bayes

```
# Load e1071 if it's not installed install it.
dynInstall("e1071")

# Trains the data set.
nb1 <- naiveBayes(Survived ~ ., data = train)
# nb1
```

5.2 Testing

```
nb_guess <- predict(nb1, test[, c("Pclass", "Sex", "Age", "SibSp",
    "Parch", "Fare", "Floor", "Room")])
table(nb_guess, test_results_final)
```

```
##           test_results_final
## nb_guess   Died Survived
##   Died      195      102
##   Survived   81       40
```

```r
# Calculate the percentage of correct guesses.
nb_correct <- nb_guess == test_results_final
table(nb_correct)/length(nb_correct) * 100
```

```
## nb_correct
##   FALSE    TRUE
## 43.7799 56.2201
```

```r
export(nb_guess, "nb")
```

5.3 Neural Networks

Neural networks only work with numerical data so I'm going to for now, remove the categorical data. An option to do later would be to associate a category to a set of nodes and activate the corresponding input node for the specific category.

```r
nn_process <- function(titanic) {

    for (key in c("Embarked", "Pclass")) {
        # Convert those factors into boolean data.frames
        a <- NULL
        for (level in levels(titanic[[key]])) {
            a[[level]] <- as.numeric(titanic[key] == level)
        }
        # Add cbind all those data.frames together
        titanic <- cbind(titanic, as.data.frame(a))
        titanic[key] <- NULL
    }

    # Covert Everything to a numeric.
    titanic <- lapply(titanic, function(x) as.numeric(x))

    # Define our Normalize function
    normalize <- function(x) {
        if (anyNA(x) || max(x) == min(x)) {
            x <- scale(x, center = FALSE, scale = TRUE)
            x[is.na(x)] <- 0
            return(x)
        } else {
            return((x - min(x))/(max(x) - min(x)))
        }
    }

    titanic <- lapply(titanic, normalize)
    titanic <- data.frame(titanic)
    return(titanic)
}

nn_train <- nn_process(train)
nn_test <- nn_process(test)
```

5.3.1 Training the neural network

```r
# Dynamically load `neuralnet`
dynInstall("neuralnet")

set.seed(1234)

# Create a formula in the form 'Survived ~ V(1) + V(2) + ...
# + V(n-1) + V(n)'
fmla <- as.formula(paste("Survived ~ ", paste(names(nn_test),
    collapse = "+")))

# Train the model
nn <- neuralnet(fmla, data = nn_train, hidden = c(7, 7), threshold = 0.05)
plot(nn)
```

```r
# Predict some data
nn_predict <- compute(nn, subset(nn_train, select = -c(Survived)))
cor(nn_predict$net.result, as.numeric(nn_train$Survived) - 1)
```

```
##              [,1]
## [1,] 0.8238769981
```

5.4 Process and export data

```r
nn_predict <- compute(nn, nn_test)
results <- round(nn_predict$net.result)
results[results < 0] <- 0
results[results > 1] <- 1
results <- factor(x = results, c(0, 1), c("Dead", "Survived"))

export(results, "nn")
```

5.5 Random Forest

```r
dynInstall("randomForest")
```

```
## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

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

```r
rf_process <- function(titanic) {
    remove_na <- function(x) {
        if (class(x) == "factor") {
            return(x)
        } else {
```

```r
            x[is.na(x)] <- 0
            return(x)
        }
    }
    return(lapply(titanic, remove_na))

}

rf_train <- as.data.frame(rf_process(train))
rf_test <- as.data.frame(rf_process(test))

fmla <- as.formula(paste("Survived ~ ", paste(names(rf_train),
    collapse = "+")))

rand_forest <- randomForest(fmla, data = rf_train, na.action = na.omit)
print(rand_forest)
```

```
##
## Call:
##  randomForest(formula = fmla, data = rf_train, na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##          Died Survived class.error
## Died      549        0           0
## Survived    0      342           0
```

```r
# rf_predict <- predict(rand_forest, rf_test$Age)
```

Unfortunately random forests isn't working. Otherwise I'm would love to see the results it has.

6 Conclusion

```r
# Kaggle Score (based on `test`)
tree2_test
```

```
##             answer
## prediction Died Survived
##   Died      247        9
##   Survived   19      143
```

```r
accuracy_kaggle
```

```
## [1] 0.9330143541
```

```r
# Final Score (based on `test_final`)
tree2_test_final
```

```
##             answer
## prediction Died Survived
##   Died      241       35
```

```
##   Survived   35      107
```

```r
accuracy_final
```

```
## [1] 0.8325358852
```