HarvardX Data Science
Capstone Project II
Surviving the Titanic

Guido D'Alessandro

June 12, 2019

Contents

1 Introduction

As a choice for the *Choose Your Own (CYO)* Data Science Capstone project, I selected predicting surviving the R.M.S. Titanic (Royal Mail SHip Titanic) tragedy from Kaggle because it is considered to be a good machine learning exercise, as well as a good prediction starting point for novice data scientists.

The data exhibits a significant number of *NA* values and empty character strings that need to be replaced (or dealt with some way), many of the given columns require some type of conversion, classification, or mapping before they can be used as predictors—for example, cabin numbers need conversion to deck numbers, passenger ages need mapping to age groups, and so on. Because of the diversity of the work for data preparation, analysis, model setup and prediction, the exercise has an increased learning value appraised by many.

In addition, the challenge carries a certain amount of appeal because it makes one wonder about the chances that one may have had, given one's own current socio-economic contitions, to survive such a disaster.

The data for this challenge comes in two *comma-separated values (CSV)* files:

- The training set (**train.csv**)
- The testing set (**test.csv**)

The *training set* consists of 891 rows over 12 columns of which 10 columns have the potential to be used as predictors. The remaining 2 columns serve other purposes. The *Passenger ID* is used for passenger identification and table indexing, while the *Survived* column, which is the target of prediction, is included in this table for training purposes, and specifies if a given passenger survived (by a value of 1), or not (by a value of 0) the ship's sinking.

In contrast to the *training set*, the *testing set* consists of 418 rows over 11 columns where the *Survived* column, the target of prediction, is explicitly left out.

To summarize, the object of the challenge is to achieve the highest possible level of accuracy ($Accuracy = \frac{CorrectPredictions}{TotalPredictions}$) in **predicting which of the *testing set* passengers survived the ship's sinking**. When the results are submitted by a participant, Kaggle rates and rank the predictions according to the level of accuracy they reach and posts the result in the competition's leaderboard, which is periodically revolved. You can read the complete listing of the competition's rules here. (^Note that Kaggle may require a Kaggle account and login before allowing you to view any of their competion pages.*)

2 Importing the Dataset

As mentioned above, the data for this challenge comes in two *CSV* files, *train.csv* and *test.csv*, offered individually for download. The files can be manually dowloaded from Kaggle as a 34KB zipped folder called *titanic.zip*, from here. The individual files can also be downloaded manually from here, or directly from here for the *training set*, and here for the *testing set*.

If the individual *CSV* file versions are chosen for download, they can be:

- Downloaded manually from Kaggle to some local folder and read from R
- Read remotely directly from R

For the sake of simplicity, it would be normal to prefer reading the individual files remotely like this:

```
# Read URL files

# Training dataset
if (!f_csvFileExists("train.csv")) train_set <- read.csv(url(kaggle_train_url), sep = ",",
    quote = "\"")

# Testing dataset
if (!f_csvFileExists("test.csv")) test_set <- read.csv(url(kaggle_test_url), sep = ",",
    quote = "\"")
```

but in order to prevent site abuse, Kaggle requires a login for direct downloads, otherwise the download consists of an *HTML file moved notice page* demanding login. So, I decided to carry the manual downloads into the project's ./data/csv directory.

Once downloaded, I proceeded to load the files like this:

```
# For the training dataset
if (!f_rdsFileExists("train.rds")) {

    # Read the CSV
    train_set <- f_csvRead("train.csv")

    # Save as compact RDS
    f_rdsSave(train_set, "train.rds")

} else {
    # RDS Exists

    # Read the RDS
    train_set <- f_rdsRead("train.rds")
}

# Do the same for the testing dataset
if (!f_rdsFileExists("test.rds")) {
    test_set <- f_csvRead("test.csv")
    f_rdsSave(test_set, "test.rds")
} else {
    test_set <- f_rdsRead("test.rds")
}
```

3 Data Inspections

Once loaded, the datasets should be inspected to verify that the have no errors, at the very least in structure and dimensions for the following:

- Verify the number of rows and columns coincide with the dimensions described by Kaggle, which you can read here.
- Verify the column names coincide with the descriptions posted by Kaggle.
- Enumerate columns with missing values.
- Enumerate duplicated rows.

3.1 Data Structure

The first step in this section is to print a description of the files we have loaded:

```
# For the Training set
str(train_set)
```

```
'data.frame':   891 obs. of  12 variables:
 $ PassengerId: int  1 2 ...
 $ Survived   : int  0 1 ...
 $ Pclass     : int  3 1 ...
 $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 ...
 $ Age        : num  22 38 ...
 $ SibSp      : int  1 1 ...
 $ Parch      : int  0 0 ...
 $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 ...
 $ Fare       : num  7.25 ...
 $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 ...
 $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 ...
```

```
# For the Testing set
str(test_set)
```

```
'data.frame':   418 obs. of  11 variables:
 $ PassengerId: int  892 893 ...
 $ Pclass     : int  3 3 ...
 $ Name       : Factor w/ 418 levels "Abbott, Master. Eugene Joseph",..: 210 409 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 ...
 $ Age        : num  34.5 47 ...
 $ SibSp      : int  0 1 ...
 $ Parch      : int  0 0 ...
 $ Ticket     : Factor w/ 363 levels "110469","110489",..: 153 222 ...
 $ Fare       : num  7.83 ...
 $ Cabin      : Factor w/ 77 levels "","A11","A18",..: 1 1 ...
 $ Embarked   : Factor w/ 3 levels "C","Q","S": 2 3 ...
```

Followed by verifying that the returned descriptions match with Kaggle's:

```
# Verify the read sets match the kaggle documented dimensions:

# For training
if (kaggle_train_rows != nrow(train_set) | kaggle_train_cols != ncol(train_set)) {
```

```
    noquote("The dimensions of the downloaded training file do not match Kaggle's....")
    noquote("Aborting, repair this issue and try again")
} else {
    noquote(paste("Downloaded train file and Kaggle's file dimensions (", kaggle_train_rows,
        " rows x ", kaggle_train_cols, " columns) match!", sep = ""))
}
```

```
[1] Downloaded train file and Kaggle's file dimensions (891 rows x 12 columns) match!
```

```
# For testing
if (kaggle_test_rows != nrow(test_set) | kaggle_test_cols != ncol(test_set)) {
    noquote("The dimensions of the downloaded testing file do not match Kaggle's....")
    noquote("Aborting, repair this issue and try again")
} else {
    noquote(paste("Downloaded test file and Kaggle's file dimensions (", kaggle_test_rows,
        " rows x ", kaggle_test_cols, " columns)match!", sep = ""))
}
```

```
[1] Downloaded test file and Kaggle's file dimensions (418 rows x 11 columns)match!
```

That matches the advertized dimensions.

3.2 Column Names

Yet another sanity check on the datasets, this time to verify the datasets column names match the ones described by Kaggle:

```
# Verify the column names in the read sets match the kaggle documented names

# For training
if (sum(kaggle_train_col_names != colnames(train_set)) > 0) {
    noquote("The column names of the downloaded test file do not match Kaggle's...")
    noquote("Aborting, repair this issue and try again")
} else {
    noquote("Downloaded train file column names match Kaggle's!")
}
```

```
[1] Downloaded train file column names match Kaggle's!
```

```
# For testing (if here, col_match still TRUE)
if (sum(kaggle_test_col_names != colnames(test_set)) > 0) {
    noquote("The column names of the downloaded test file do not match Kaggle's...")
    noquote("Aborting, repair this issue and try again")
} else {
    noquote("Downloaded test file column names match Kaggle's!")
}
```

```
[1] Downloaded test file column names match Kaggle's!
```

which also checks OK.

Note: you can find a reference to Kaggle's dataset description here, under "Data Dictionary" where some column names may not agree in order and case with the ones in the downloaded files and may also include spaces, however,

it is easy to discern they refer to the same column. Finally, later on the page, under "Columns", you may scroll all the values of each column starting with the column's name. I Used this name to construct 2 global variables, ***PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked*** *and* ***PassengerId, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked****, which are the names you see in the code above*

3.3 Missing Values

Knowledge of missing values is particularly important because some of the columns suffering this condition may later prove to be strong predictors in the training area, where we may just simply exclude rows with missing values from training, or define a bucket value to class them if possible, but this could lead to some additional problems as well. However, it becomes even more problematic when the missing values exist in the test area because any replacement there could lead to disastrous results. In that case, we would either require finding a reasonable strategy and justification for replacement or, defer to use the predictor on rows with missing values. Therefore, the earlier we know about these conditions, the better to start planning the incorporation of these shortfalls early in the analysis and design of the models we will use. Both tables, train_set and test_set were checked for NA values like this:

```
# In training file
train_na <- colnames(train_set)[colSums(is.na(train_set)) > 0]
train_na
```

```
[1] "Age"
```

```
# In testing file
test_na <- colnames(test_set)[colSums(is.na(test_set)) > 0]
test_na
```

```
[1] "Age"  "Fare"
```

As well like this for empty cells:

```
# MAKE SURE NA.RM = TRUE colSums return NA if found and na.rm = FALSE

# For training file
train_em <- colnames(train_set)[colSums(train_set == "", na.rm = TRUE) > 0]
train_em
```

```
[1] "Cabin"    "Embarked"
```

```
# In testing file
test_em <- colnames(test_set)[colSums(test_set == "", na.rm = TRUE) > 0]
test_em
```

```
[1] "Cabin"
```

In short, this is the summary of what we found:

Table	NA.	Empty
train_set	Age	Cabin Embarked
test_set	Age Fare	Cabin

where we see that there are several missing value columns in both, the training and testing sets.

3.4 Duplicated Rows

It is important to establish if there are duplicated rows, or more than one entry per PassengerId, in the data files because some of the analyses and computations that follow presume that. In that case, if necessary, we would be required to determine which of the duplicates to remove.

This is how we checked for the existence of duplicated rows:

```
# train_set
tr_dup <- nrow(train_set[duplicated(train_set["PassengerId"]), ])

# test_set
ts_dup <- nrow(test_set[duplicated(test_set["PassengerId"]), ])

bind_rows(data.frame(Table = "train_set", Duplicates = tr_dup), data.frame(Table = "train_set",
    Duplicates = ts_dup)) %>% knitr::kable(caption = "Duplicated Rows")
```

Table 2: Duplicated Rows

Table	Duplicates
train_set	0
train_set	0

```
if (tr_dup | ts_dup) {
    noquote("There are duplicates, a determination of row removal is needed.")
} else {
    noquote("There are no duplicates, so a determination of row removal is not needed.")
}
```

```
[1] There are no duplicates, so a determination of row removal is not needed.
```

Finally, we checked for cross-duplicated rows that is, for survival entries in the train dataset of a passenger whose fate is asked in the test dataset like this:

```
tbl <- inner_join(test_set, train_set, by = "PassengerId")
nr <- nrow(tbl)
noquote(paste("There are ", nr, " passengers whose fate is known", ifelse(nr > 0,
    "--these can be excluded from prediction.", "--nothing left to do here."), sep = ""))
```

```
[1] There are 0 passengers whose fate is known--nothing left to do here.
```

3.5 Summary

We wrap this section with a summary of the datasets:

```
[1] -- TRAINING DATASET SUMMARY:

  PassengerId       Survived          Pclass
 Min.   :  1.0   Min.   :0.0000   Min.   :1.000
 1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000
```

```
Median :446.0   Median :0.0000   Median :3.000
Mean   :446.0   Mean   :0.3838   Mean   :2.309
3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000
Max.   :891.0   Max.   :1.0000   Max.   :3.000

                                    Name         Sex           Age
Abbing, Mr. Anthony                    :  1   female:314   Min.   : 0.42
Abbott, Mr. Rossmore Edward            :  1   male  :577   1st Qu.:20.12
Abbott, Mrs. Stanton (Rosa Hunt)       :  1                Median :28.00
Abelson, Mr. Samuel                    :  1                Mean   :29.70
Abelson, Mrs. Samuel (Hannah Wizosky):  1                3rd Qu.:38.00
Adahl, Mr. Mauritz Nils Martin         :  1                Max.   :80.00
(Other)                                :885                NA's   :177
    SibSp           Parch             Ticket         Fare
Min.   :0.000   Min.   :0.0000   1601    :  7   Min.   :  0.00
1st Qu.:0.000   1st Qu.:0.0000   347082  :  7   1st Qu.:  7.91
Median :0.000   Median :0.0000   CA. 2343:  7   Median : 14.45
Mean   :0.523   Mean   :0.3816   3101295 :  6   Mean   : 32.20
3rd Qu.:1.000   3rd Qu.:0.0000   347088  :  6   3rd Qu.: 31.00
Max.   :8.000   Max.   :6.0000   CA 2144 :  6   Max.   :512.33
                                 (Other) :852
        Cabin     Embarked
           :687    :  2
B96 B98    :  4   C:168
C23 C25 C27:  4   Q: 77
G6         :  4   S:644
C22 C26    :  3
D          :  3
(Other)    :186

[1] -- TESTING DATASET SUMMARY:

 PassengerId         Pclass
Min.   : 892.0   Min.   :1.000
1st Qu.: 996.2   1st Qu.:1.000
Median :1100.5   Median :3.000
Mean   :1100.5   Mean   :2.266
3rd Qu.:1204.8   3rd Qu.:3.000
Max.   :1309.0   Max.   :3.000

                                        Name         Sex           Age
Abbott, Master. Eugene Joseph             :  1   female:152   Min.   : 0.17
Abelseth, Miss. Karen Marie               :  1   male  :266   1st Qu.:21.00
Abelseth, Mr. Olaus Jorgensen             :  1                Median :27.00
Abrahamsson, Mr. Abraham August Johannes :  1                Mean   :30.27
Abrahim, Mrs. Joseph (Sophie Halaut Easu):  1                3rd Qu.:39.00
Aks, Master. Philip Frank                 :  1                Max.   :76.00
(Other)                                   :412                NA's   :86
    SibSp           Parch            Ticket         Fare
Min.   :0.0000   Min.   :0.0000   PC 17608:  5   Min.   :  0.000
1st Qu.:0.0000   1st Qu.:0.0000   113503  :  4   1st Qu.:  7.896
Median :0.0000   Median :0.0000   CA. 2343:  4   Median : 14.454
Mean   :0.4474   Mean   :0.3923   16966   :  3   Mean   : 35.627
3rd Qu.:1.0000   3rd Qu.:0.0000   220845  :  3   3rd Qu.: 31.500
Max.   :8.0000   Max.   :9.0000   347077  :  3   Max.   :512.329
                                  (Other) :396   NA's   :1
            Cabin     Embarked
```

```
               :327   C:102
B57 B59 B63 B66:  3   Q: 46
A34            :  2   S:270
B45            :  2
C101           :  2
C116           :  2
(Other)        : 80
```

4 Variable Analysis

4.1 Introduction

This section is dedicated to the investigation of the relationships between the target column, *Survived*, and all the other columns. In other words, we will be looking at the possible contributions of every v_i:

Eq. 1: $Survived = f(v_1, v_2, ..., v_n) + \varepsilon$

where v_i are the different predictors or independent variables, or columns, yet to be discovered, ε, is the independent zero-centered residual explained by random variation.

Of course other models are possible, for example:

Eq. 2: $Survived = f_i(v_i, ..., v_{ni}) + \varepsilon_i$

where every $f_i()$ can include a prescribed number of variables from the data as needed, although their number can increase as a function of the number of missing values. **Eq. 1** would be the ideal, but there is always wrong with collected data–missing values, erroneous entris, and so on, imagine back then, during the early 1900's.

Let us start by looking at the possible predictors individually and independently of each other to assess their importance to the model. After the selection of the most promising variables completes, we can move into testing the multivariate model in cross-validation.

We will keep track of selected variables (or columns–if you prefer to think in terms of tables), as we go along, in a table summarizing the following information:

Column: The column names of the selected variables.

Type: The data type of the variable values (i.e. Number, Integer, etc.)

Use As: The data type of the variable values as will be used.

Requires Work: Yes or no.

Work Type: A one word description of the type of work needed (e.g. "Type Conversion", etc.).

Work Description: A short description of the work to be done (e.g. "To Number", etc.).

Please note that we have not decided how we are going to solve $f(v_1, v_2, ..., v_n)$. We will make this decision later after comparing different methods throughout the model building sections of this study.

4.2 Pclass

This column is described by Kaggle as *"Ticket class"*, with the following key: *"1 = 1st, 2 = 2nd, 3 = 3rd"*

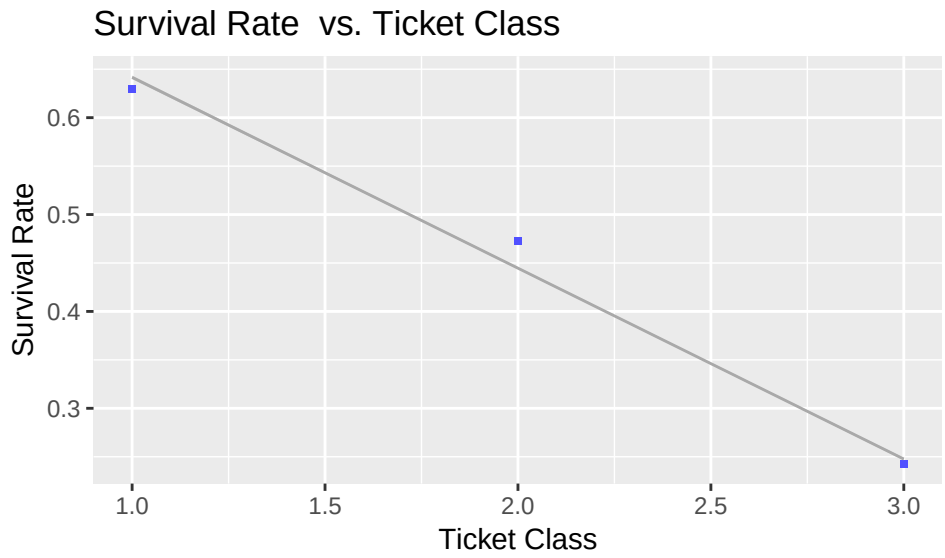
The *Pclass* column is defined in the dataset as an integer, described by Kaggle to range from 1 through 3 (references here under *Data Description–>Data Dictionary* and under *Data Sources–>file-name–>Columns*). In short, like this:

Table 3: Ticket Class Values

First	Second	Third
1	2	3

we know there are no missing values in this column from the missing values section, so let us continue to inspect the

data:

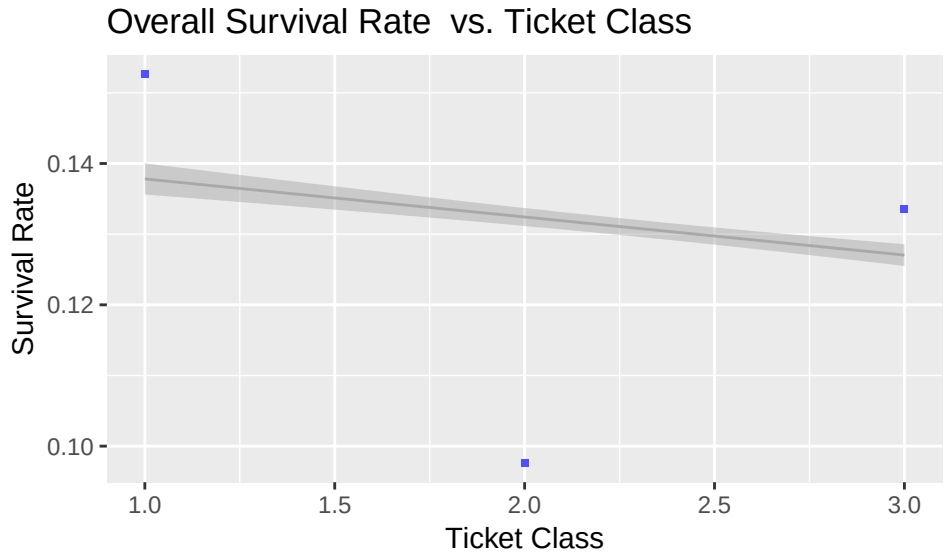


Where a negative correlation is obvious. In numbers, we see that a passenger in first class had 63%chances of survival among all first class passengers, 47% in second among all second class passengers, and 24% in third among all third class passengers. Fair or not, we need to quantize this dependency by calculating the correlation coefficient like this:

```
noquote(paste("Correlation =", round(cor(x = temp$Pclass, y = temp$pc_avg), rnd_detail)))
```

```
[1] Correlation = -0.9961
```

Almost one. We are keeping this column as a predictor because of its high correlation coefficient, but keep in mind that these numbers reflect group probabilities, that is, the probability of survival among first class passengers, otherwise, those numbers would have looked like this:




```
[1] Survival Rates: First=15.3%, second=9.8%, and third=13.4%.
```

Which were, at that time, the overall chances of survival for passengers traveling in those classes, that is 15.3% for those traveling in first class, and so on. Of course, we can also have a number of group-based models, each acting solely on their own group types to take advantage of the high correlations.

Since we have decided to keep this column, we add it to our selection table:

```
sel_table <- data.frame(Column = "Pclass", Type = "Integer", `Use As` = "Factor",
    Req.Work = "Y", `Work Type` = "TC", Description = "To factor", `Req Alt` = "N")
sel_table %>% knitr::kable(caption = "Variable Analysis Table")
```

Table 4: Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N

Since we are going to be carrying this table all the way to the data modification and model implementation sections of this report, I have taken the time to classify the types of operations that I forsee executing on the datasets.

Work Types:

Classification: (CL) Conversion of integer or continuous values into groupable bins.

Data Separation: (DS) Never executed, except for submission.

Model Separation: (MS) Produce the same model but using a different number of variables working on the same datasets.

Numerical Transforms: (NT) Scaling, raising to a power, etc.

Table Modifications: (TM) Addition of trandformed columns only.

Type Conversions: (TC) Conversion between types.

Values Exclussions: (VE) Through model separation.

Abbreviations: (Other than the ones listed above)

Req.Alt: Requires alternate model.

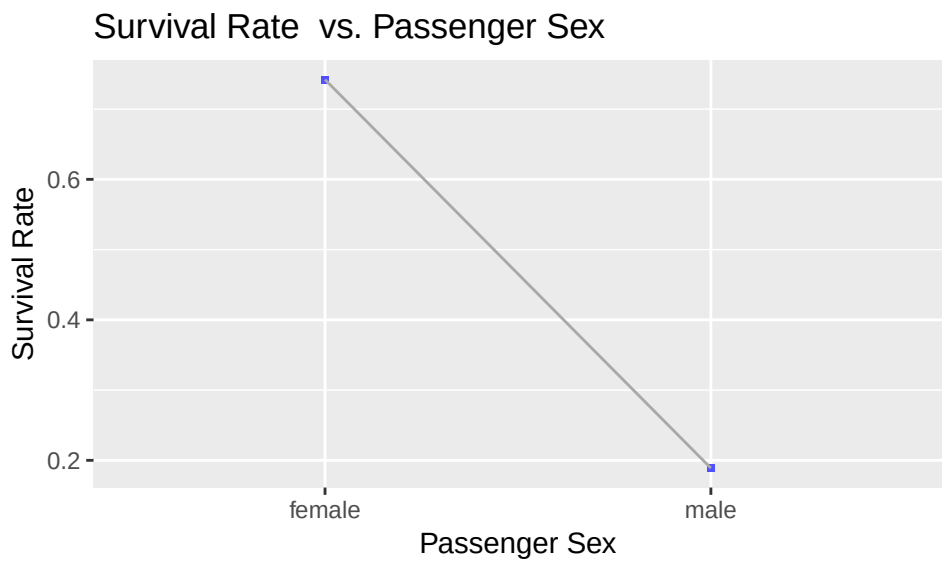
Req.Work: Requires work.

Y: Yes

N: No

4.3 Sex

Kaggle describes this column as *"Sex."* We saw earlier, in the data structure section of this study, that this column is a factor with two levels, "male" and "female," and (in the missing values section) that the data is complete, not missing any values, therefore I selected this column for further investigation.

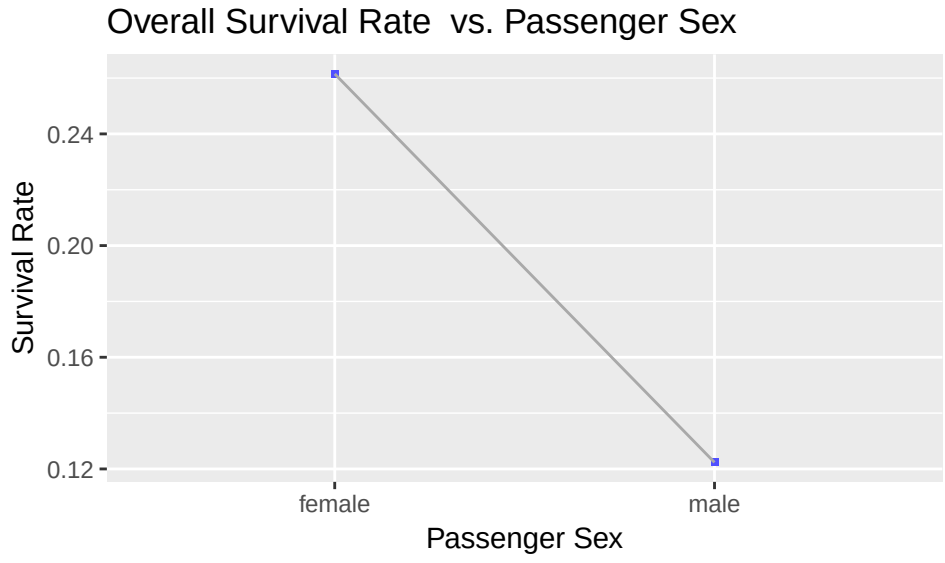


The above graph shows a strong correlation between sex and survival rate. If you were male you had a 19% change of survival among all males, and 74% if you were female among all females, exhibiting a correlation coefficient of:

```
noquote(paste("Correlation =", round(cor(x = as.integer(temp$Sex), y = temp$sex_avg),
    rnd_detail)))
```

```
[1] Correlation = -1
```

Because of its high correlation coefficient, this is another variable that we will incorporate into our model. As with the ticket class variable calculations, these numbers reflect group probabilities, that is, the probability of a female surviving among females, for example. These probability numbers change when comparing the rates against the whole population to this:




```
[1] Correlation = -1
```

Despite this, this variable continues to show a strong correlation, but survival rates are lower–26% for females and 12% for males among all passengers, as expected. Since we decided to incorporate this column into our model, we add it to our selection table like this:

```r
sel_table <- bind_rows(sel_table, data.frame(Column = "Sex", Type = "Factor", `Use As` = "As is",
    `Req Work` = "N", `Work Type` = "", Description = "", `Req Alt` = "N"))

sel_table %>% knitr::kable(caption = "Updated Variable Analysis Table")
```

Table 5: Updated Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N

4.4 Age

Kaggle describes this column as *"the age in years"* with the following footnote:

> **age:** Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5.

We know from the missing values section that there are several missing values in the *Age* column. We found NA's in both, the *train_set* and *test_set* tables.

Table 6: Age Missing Values

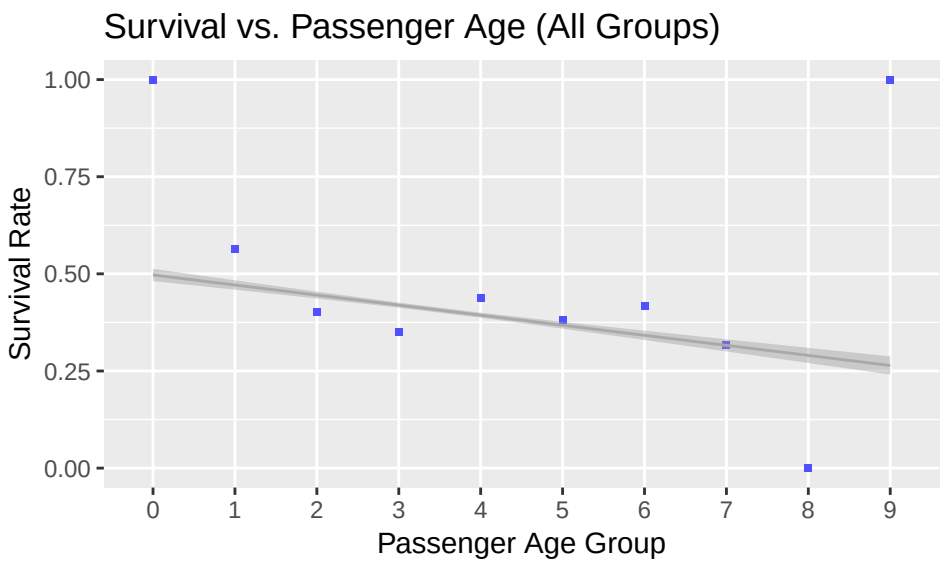
Train.NA	Train.Empty	Train.Total	Test.NA	Test.Empty	Test.Total
177	0	177	86	0	86

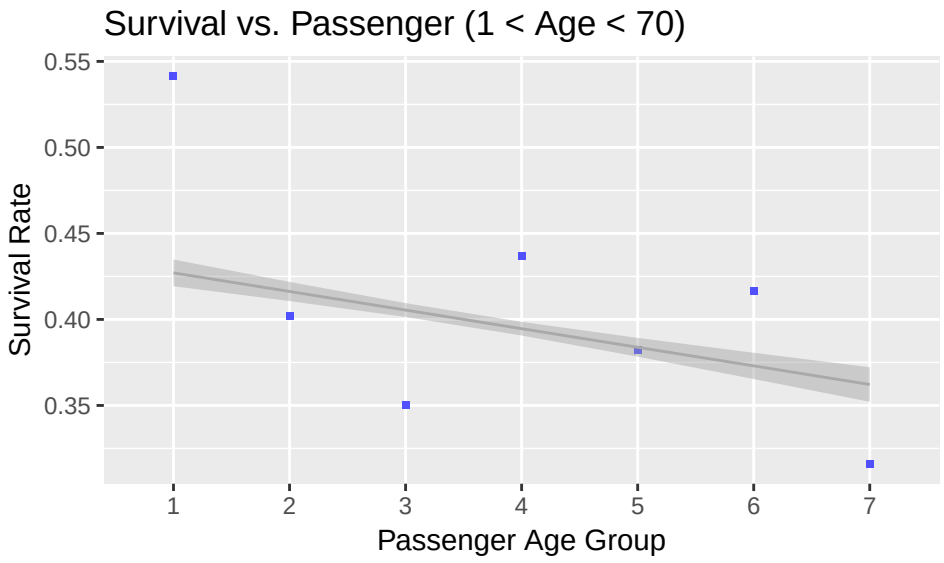
Despite this, it might be worth the work incorporating this variable into our models just because it may have a great effect on survival when used in conjunction with the other variables.

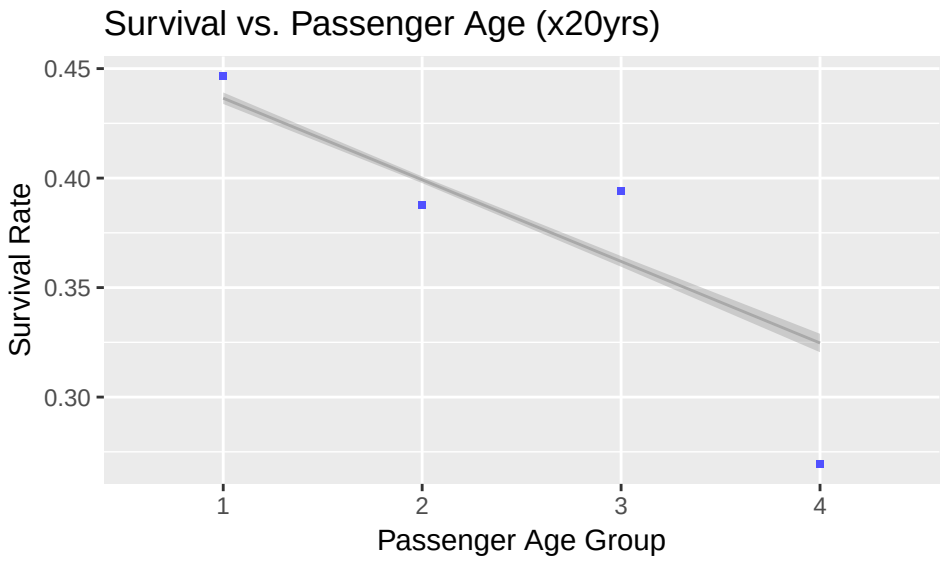
Table 7: Age Groups in train_set

Age.Group	Age	Survival.Rate
0	0-12 mos	100.00
1	1-9 yrs	56.36
2	10-19 yrs	40.20
3	20-29 yrs	35.00
4	30-39 yrs	43.71
5	40-49 yrs	38.20
6	50-59 yrs	41.67
7	60-69 yrs	31.58
8	70-79 yrs	0.00
9	80-89 yrs	100.00

Where all infants, 7, and 1 8x-year-old, survived. There were some, 6, 70+-years-olds that did not survive. I will exclude group 0 from the computations and treat it as an independent class, and ignore groups 8 and 9 unless I find passengers within these age groups in the test dataset, in which case I would join the two groups to avoid ill-posed regressions.


Survival vs. Passenger Age (All Groups)
Survival Rate
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8
9
Passenger Age Group


Survival vs. Passenger (1 < Age < 70)
Survival Rate
0.55
0.50
0.45
0.40
0.35
1
2
3
4
5
6
7
Passenger Age Group


Survival vs. Passenger Age (x20yrs)
Survival Rate
0.45
0.40
0.35
0.30
1
2
3
4
Passenger Age Group

```
[1] Correlation = -0.8066
```

The third graph and the displayed correlation coefficient above excludes the infant group and presents the results in groups of 20 years. I will incorporate this column into the model because I suspect it has a great influence in the outcome when acting together with other variables.

Let us add it to our selection table:

Table 8: Updated Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y

4.5 SibSp

This column is described by Kaggle as *"the number of siblings/spouses aboard the Titanic"* with the following footnote:

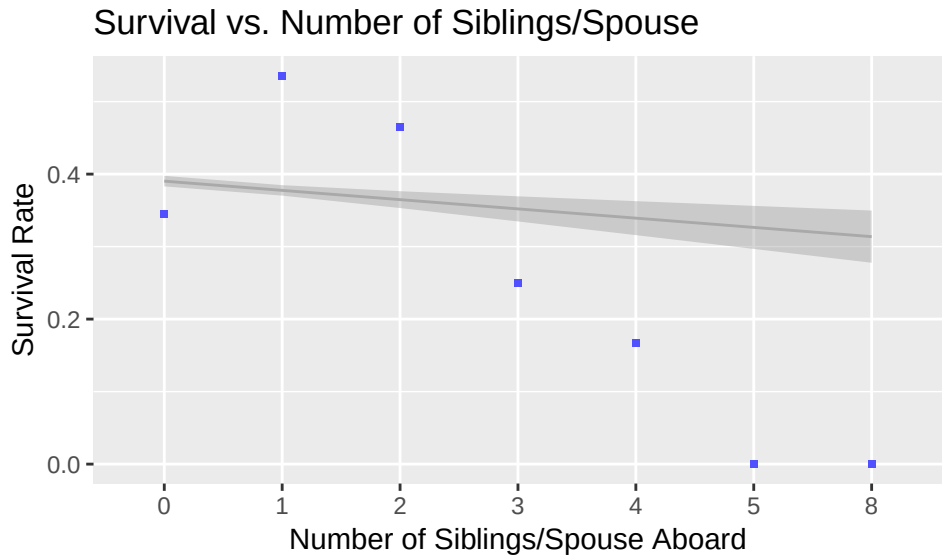
> **sibs**: The dataset defines family relations in this way...
>
> Sibling = brother, sister, stepbrother, stepsister.
>
> Spouse = husband, wife (mistresses and fiancés were ignored).

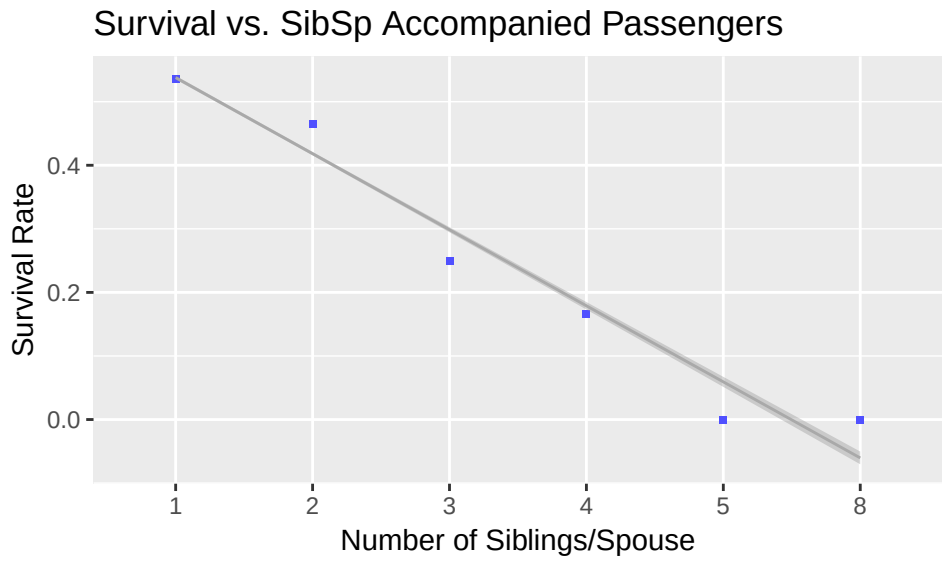
Kaggle's literature does not specify if the number in this column is a sum of all those relatives, though it should be safe to presume so because this is an integer type with no room for text representations with separators. Here a few listed entries:

Table 9: Random sample of rows to show SibSp values

	PassengerId	Name	SibSp
1	1	Braund, Mr. Owen Harris	1
62	62	Icard, Miss. Amelie	0
113	113	Barton, Mr. David John	0
227	227	Mellors, Mr. William John	0
238	238	Collyer, Miss. Marjorie "Lottie"	0
435	435	Silvey, Mr. William Baird	1
466	466	Goncalves, Mr. Manuel Estanslas	0
511	511	Daly, Mr. Eugene Patrick	0
640	640	Thorneycroft, Mr. Percival	1
696	696	Chapman, Mr. Charles Henry	0
707	707	Kelly, Mrs. Florence "Fannie"	0
766	766	Hogeboom, Mrs. John C (Anna Andrews)	1

In addition, this column does not exhibit missing values. Let us check if it can have an influence on survival:






```
[1] Correlation = -0.9879
```

We see the trend after 1 sibling/spouse. It seems that passengers traveling alone were a class to themselves. If we incorporate this column into our model, I will separate group 0, passengers traveling alone, from the fitting algorithms. The second graph and the correlation coefficient above excludes it.

We are selecting this column, so let us add it to our selection table:

```
sel_table <- bind_rows(sel_table, data.frame(Column = "SibSp", Type = "Integer",
    `Use As` = "As factor", `Req Work` = "Y", `Work Type` = "TC", Description = "To factor",
    `Req Alt` = ""))
sel_table <- bind_rows(sel_table, data.frame(Column = "", Type = "Factor", `Use As` = "As is",
    `Req Work` = "Y", `Work Type` = "VE", Description = "Exclude group", `Req Alt` = "N"))
sel_table %>% knitr::kable(caption = "Updated Variable Analysis Table")
```

Table 10: Updated Variable Analysis Table

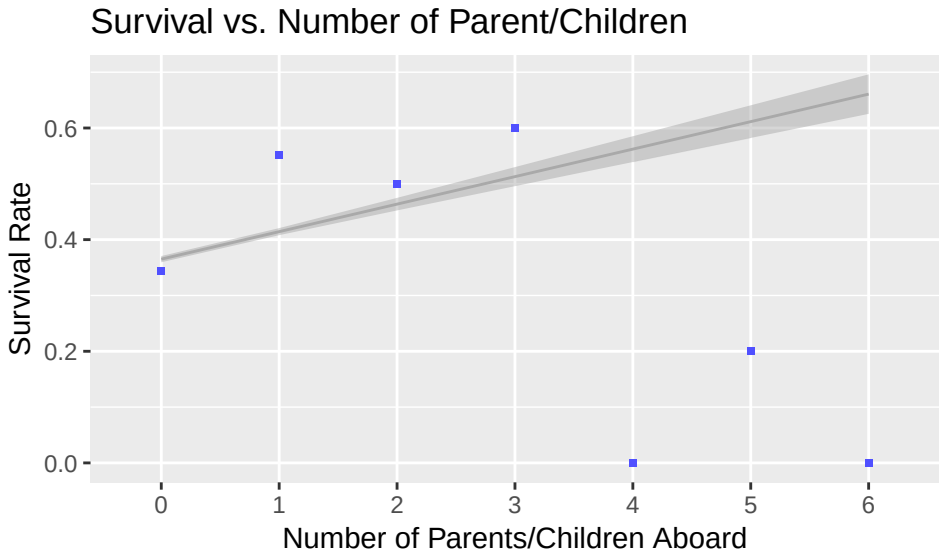
Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y
SibSp	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	N

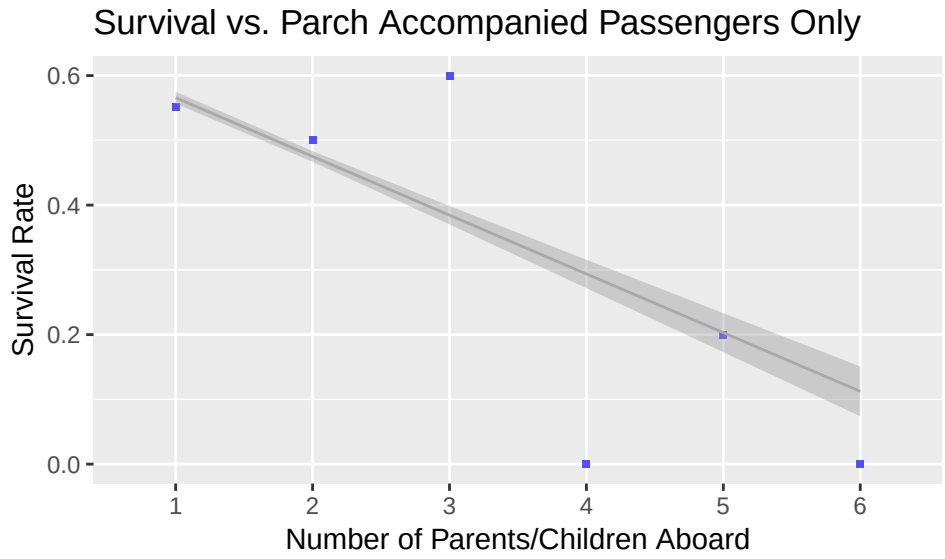
4.6 Parch

This column is described by Kaggle as the *"number of parents/children aboard the Titanic"* with the following footnote:

> **parch:** The dataset defines family relations in this way...
>
> Parent = mother, father.
>
> Child = daughter, son, stepdaughter, stepson.
>
> Some children travelled only with a nanny, therefore parch=0 for them.

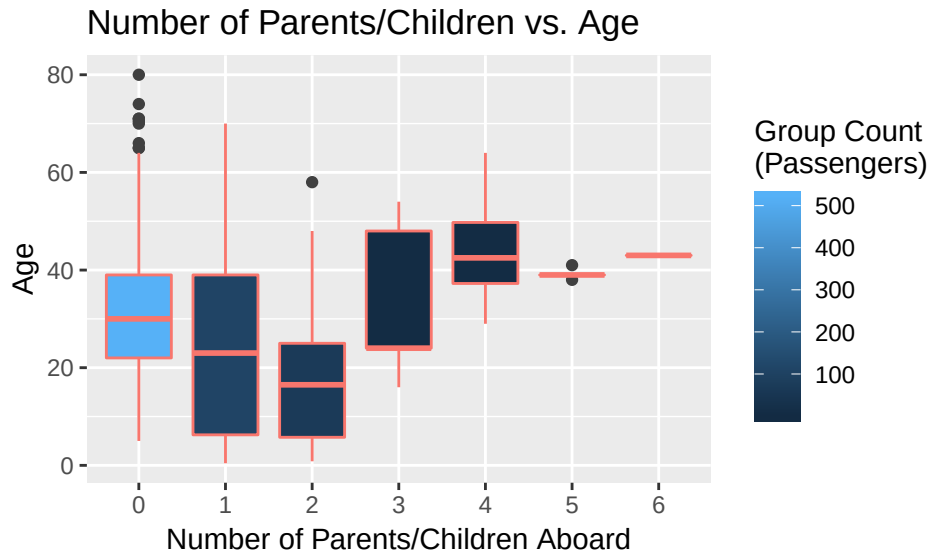
Again, Kaggle's literature does not specify if the number in this column is a sum of all those relatives (parents and children), though it should be safe to presume so because this is an integer typed column with no room for text representations with separators. Let us see how this column can contribute to our model:






```
[1] Correlation = -0.1242
```

These are mixing results, because the survival rates here are somehow inter-related with the *Age* column findings, that is for example, if you have a count of 2 in this column, it means that you are likely traveling with either 2 parents, so most likely you are a child, or 1 parent and 1 child, so you must of child-bearing age or older. I will keep this column as a predictor because it might be supportive in the multivariate model described in the survival dependencies section, especially in instances when the *Age* column is missing values. Lets see if there is a cross relationship with *Age*:



Not much structure to make sense of. However, we will keep it just because it may serve to provide some useful information for prediction in the multivariate model described earlier:

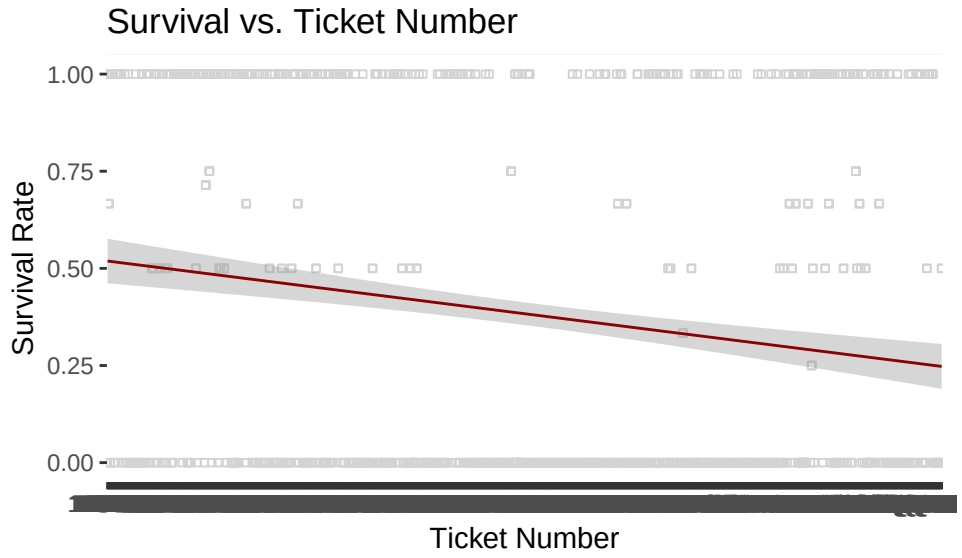
```
sel_table <- bind_rows(sel_table, data.frame(Column = "Parch", Type = "Integer",
    `Use As` = "As factor", Req.Work = "", `Work Type` = "TC", Description = "To factor",
    `Req Alt` = ""))
sel_table <- bind_rows(sel_table, data.frame(Column = "", Type = "Factor", `Use As` = "As is",
    Req.Work = "", `Work Type` = "VE", Description = "Exclude group", `Req Alt` = "N"))
sel_table %>% knitr::kable(caption = "Updated Variable Analysis Table")
```

Table 11: Updated Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y
SibSp	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	N
Parch	Integer	As factor		TC	To factor	
	Factor	As is		VE	Exclude group	N

4.7 Ticket

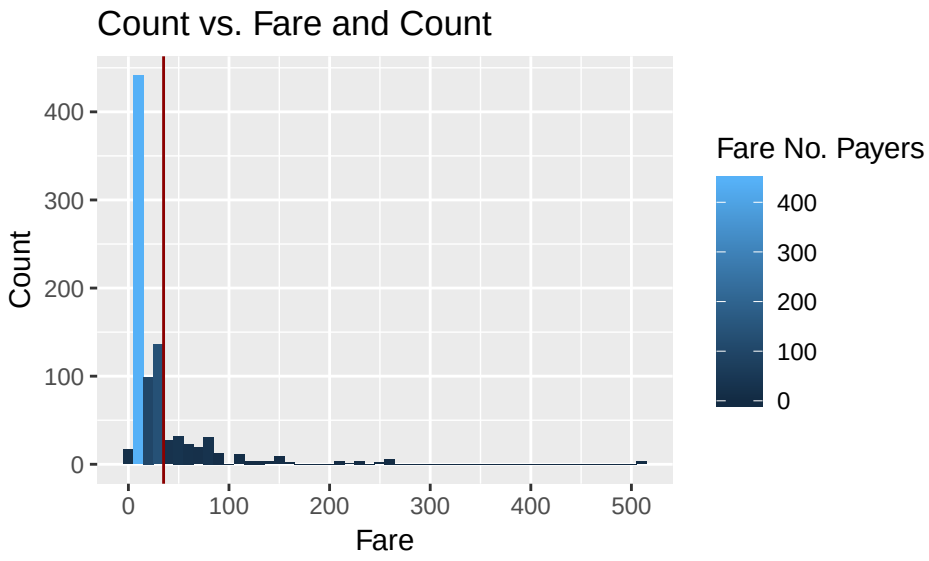
This column is described by Kaggle as *"Ticket number"*. It has no missing values, but I think there will be little dependency of survival on this, except that low ticket numbers may imply early embarkation, which could have made some passangers better acquiented with the crew in charge of filling the safe-boats during sinking, which may have increased the odds of survival. Let's see.



So there seems to be a negative correlation, comfirming my suspitions. However, I will not use this column in our model, particularly because I believe that all the information contributed to survival from the tickret number is implicitly conveyed by the contribution of the *Embarked* column, which we will examine later.

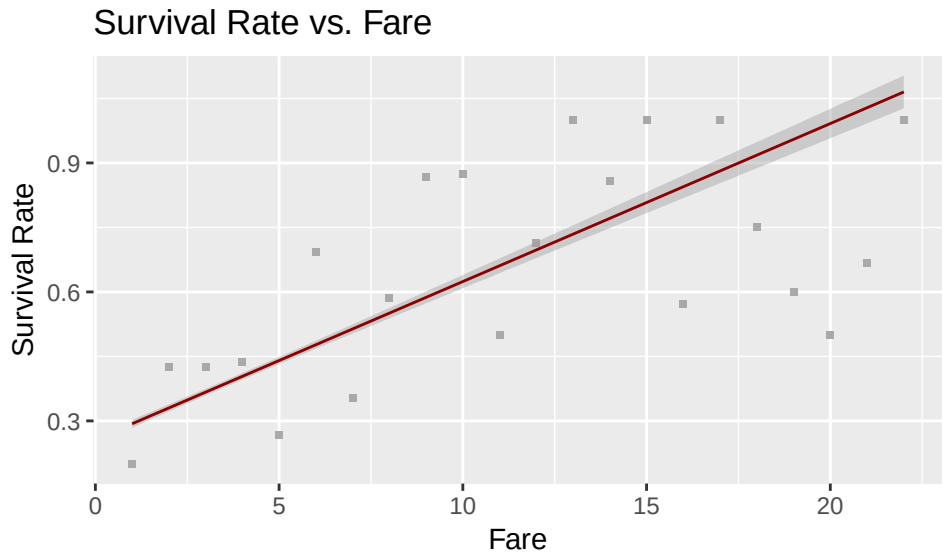
4.8 Fare

This column is described by Kaggle as *"Passenger fare"*. From our missing values section, we know this column contains a missing value, 1 row(s). However it might be worth looking at the effect of this variable on the survival rate. So let us take a quick look at that:



Notice that the majority of passengers paid an average of \$32.2 (dark-red vertical line) for their tickets, while others (very few of them) paid more than \$500.00. Also, notice that the lowest fares correspond to third class in their majority and the highest ones to first class in their entirety. In the second graph we can see that 76.32% of the passengers paid less than average (dark-red vertical line).

Now let us examine the impact of fares, when binned, say in increments of \$10.00, to compute their associated survival rates like this:




```
[1] Missing Fare values in the test dataset = 1
```

Even though the error from the generalized least square regression fit appears at sight comparatively large and considering that the contribution of this column may be implicitly provided by the Pclass column, we will keep this column in spite its missing values in the training set, for it may contribute in the final prediction, at the very least in predicting those instances where there are 1 missing values in the test set.

Table 12: Updated Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y
SibSp	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	N
Parch	Integer	As factor		TC	To factor	
	Factor	As is		VE	Exclude group	N
Fare	Number	As is		CL	To bins	
		As factor		TC	To factor	Y

4.9 Cabin

This column is described by Kaggle as *"Cabin number"*. It has many missing values in both the training and testing datasets, which we count like this:

```
# For Training set, count missing values in Cabin column
tmp1 <- train_set %>% filter(is.na(Cabin) | Cabin == "")
mv1 <- nrow(tmp1)
noquote(paste("Train set missing values in Cabin column =", mv1))
```

```
[1] Train set missing values in Cabin column = 687
```

```r
# For testing set, count missing values in Cabin column
tmp2 <- test_set %>% filter(is.na(Cabin) | Cabin == "")
mv2 <- nrow(tmp2)
noquote(paste("Test set missing values in Cabin column =", mv2))
```

```
[1] Test set missing values in Cabin column = 327
```

There are 687 missing values in the Cabin column, that leaves 0.2289562% usable columns for training to contribute in the prediction of 0.2177033% of the test rows. The contribution that this column could have provided to the model is greatly reduced because this fact. In addition, there are another number of entries that consist of multiple cabin numbers, sometimes across different decks, which could make it difficult to extract valuable model fitting information, as we can see here:

Table 13: Train set: Sample Cabin Values

PassengerId	Name	Cabin
2	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	C85
4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	C123
7	McCarthy, Mr. Timothy J	E46
11	Sandstrom, Miss. Marguerite Rut	G6
12	Bonnell, Miss. Elizabeth	C103
22	Beesley, Mr. Lawrence	D56
24	Sloper, Mr. William Thompson	A6
28	Fortune, Mr. Charles Alexander	C23 C25 C27
32	Spencer, Mrs. William Augustus (Marie Eugenie)	B78
53	Harper, Mrs. Henry Sleeper (Myna Haxtun)	D33

Table 14: Test set: Sample Cabin Values

PassengerId	Name	Cabin
904	Snyder, Mrs. John Pillsbury (Nelle Stevenson)	B45
906	Chaffee, Mrs. Herbert Fuller (Carrie Constance Toogood)	E31
916	Ryerson, Mrs. Arthur Larned (Emily Maria Borie)	B57 B59 B63 B66
918	Ostby, Miss. Helene Ragnhild	B36
920	Brady, Mr. John Bertram	A21
926	Mock, Mr. Philipp Edmund	C78
933	Franklin, Mr. Thomas Parham	D34
936	Kimball, Mrs. Edwin Nelson Jr (Gertrude Parsons)	D19
938	Chevre, Mr. Paul Romaine	A9
940	Bucknell, Mrs. William Robert (Emma Eliza Ward)	D15

There is also the possibility that deck number information is implicitly included in the ticket class if we consider that ship passenger classes are usually placed in distinct deck levels, where the lower the class the deeper the level. I will not include this column in the prediction models.

4.10 Embarked

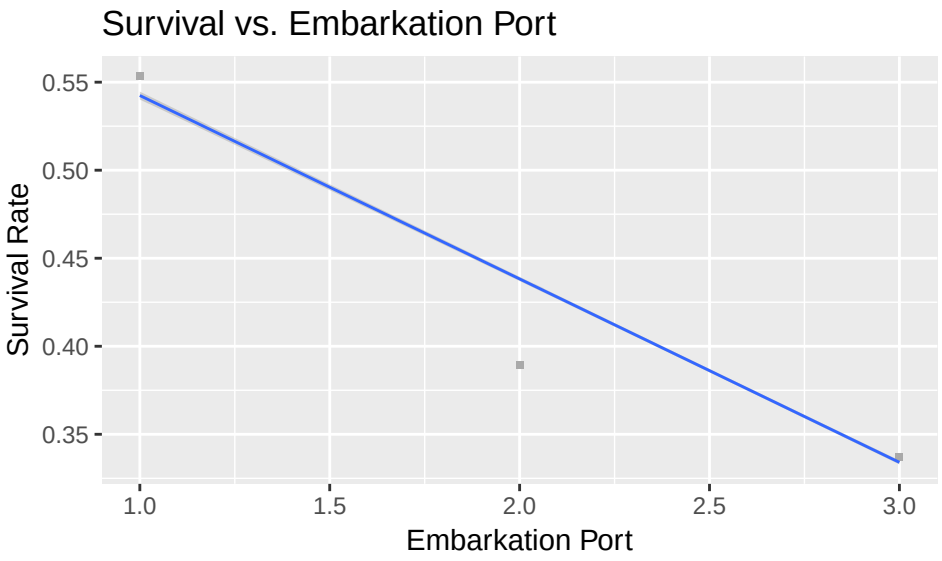
This column is described by Kaggle as *"Port of Embarkation"*with the following comment (key):

C = Cherbourg, Q = Queenstown, S = Southamton.

It shows in the datasets as a factor with 4 levels–the three already mentioned in Kaggle's description plus one for missing values in the training dataset, as we were able to verify in the missing values section of this study. The number of missing entries in the training set is computed like this:

```
[1] Embarked Column Missing Values in Training Set = 2
```

where there are only 2 missing value entries in the *Embarked* column of the training dataset, a number I can live with. To check for dependencies of our model in this variable, let us do our usual plots:



Which confirms my suspicions on the effect of the order of port stops in the survival rate because earlier boarding passengers had the more time to acquaint with the crew that later loaded the safe-boats. Also, we can presume that a good number of crew members and initial passengers were native to the originating port. I am including this column in the model and therefore add the column to the selection table:

Table 15: Updated Variable Analysis Table

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y
SibSp	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	N
Parch	Integer	As factor		TC	To factor	
	Factor	As is		VE	Exclude group	N
Fare	Number	As is		CL	To bins	
		As factor		TC	To factor	Y
Embarked	Factor	As is	N			Y

4.11 Summary

We spent the last few sections of this study examining all the variables that came with the datasets, deciding whether or not to include them for prediction and in that case, whether or not to used them as found or required additional work before use. We also kept track of our findings in the selection table, whose final version we show here:

Table 16: Final Variable Analysis Table Results

Column	Type	Use.As	Req.Work	Work.Type	Description	Req.Alt
Pclass	Integer	Factor	Y	TC	To factor	N
Sex	Factor	As is	N			N
Age	Integer	As is	Y	CL	To bins	
	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	Y
SibSp	Integer	As factor	Y	TC	To factor	
	Factor	As is	Y	VE	Exclude group	N
Parch	Integer	As factor		TC	To factor	
	Factor	As is		VE	Exclude group	N
Fare	Number	As is		CL	To bins	
		As factor		TC	To factor	Y
Embarked	Factor	As is	N			Y

Not that this list represents the final word in the composition of our model, but that it serves as a good starting point from which to carry testing and cross-validations against the data. It is possible that through this process we decide to add (or drop) a column from the model (or maybe even change the format in which it is used). With a list like this at hand, however, the work becomes comprehensible, manageable, and straight forward.

5 Data Modifications

5.1 Introduction

The truth is that other than adding additional columns to the tables to hold transformed values we are not going to changethe data as provided by Kaggle.

This section will be devoted to the pruduction of those extra columns, as described by the section table we have been carrying along.

5.2 Missing Values

To avoid residuals larger than necessary, I decided not to attempt the replacement or reconstruction of missing values (the process by which the value of a variable instance is predicted from the other independent variables in the system, replaced with a good guess, or simply placed into a bin of unknowns). Instead, I decided to exclude rows with missing values from main training and create additional models to handle prediction for the few rows where the variables have a missing value. I know this decision will cause an abundance of models, each with its own number of independent variables, but it will prevent the model from introducing unnecessary artifacts to the system.

If there are $N = 7$ variables available for main training, and there are, say $mv = 3$ columns with missing values, we would have:

$$NumberOfModels = MainModel + 3ReducedModels = 4Models$$

We can represent this system like this:

$$Survived_{main} = f_{main}(v_1, v_2, ..., v_7) + \varepsilon_{main}$$

$$Surviced_1 = f_1(v_1, v_2, ..., v_6) + \varepsilon_1$$

$$Surviced_2 = f_2(v_1, v_2, ..., v_6) + \varepsilon_2$$

$$Surviced_3 = f_3(v_1, v_2, ..., v_6) + \varepsilon_3$$

$$Survived = OneOf[Survived_{main}, Surviced_1, Surviced_2, Surviced_3]$$

Where the $f_i()$ are simply dimished versions of $f_{main}()$

The total count of models can end up being more or less than 4, more if there are rows missing more than one value, and less depending on which of those variables have missing values only in the training set, since a dimished model would not be needed unless it performs better than the main under validation.

In any event, do not let yourself into believing tthese issues are a large cause of complication—once the models are determined, we can automate the complete validation process and make the task simple.

Because we will be adding columns to out training and testing datasets, we will call these datasets tables and files **tr_workset**, for the training set (*train_set*) and **ts_workset**, for the testing set (*test_set*), which we will save after every modification, starting now like this:

```
# For workset the training set
tr_workset <- train_set
f_rdsSave(tr_workset, "tr_workset.rds")

# For workset the training set
ts_workset <- test_set
f_rdsSave(ts_workset, "ts_workset.rds")
```

In addition, we will be skipping table modifications for columns that we are not using or that were found good for use as provided.

5.3 Pclass

We had determined in the Pclass analysis section of this report that we were going to convert *Pclass* to a factor type (see here). We are going to change our minds in favor of keeping things as simple as possible, because this step was not necessary. It just felt like Pclass should have been a factor.

So we will leave Pclass unchanged. That is all.

5.4 Age

This was found to be onr of the most problematic variables in the datasets. I had decided to bin the age of passengers into age groups and exclude group 0 (infants <= 1 year-old) from the training algorithms and simply set survived to 1 if an infant was found in the test set. Are there any?

```
ts_infants <- ts_workset %>% filter(Age <= 1)
ts_infants %>% dplyr::select(PassengerId, Pclass, Sex, Age, SibSp, Parch, Embarked)
```

```
  PassengerId Pclass    Sex  Age SibSp Parch Embarked
1        1009      3 female 1.00     1     1        S
2        1093      3   male 0.33     0     2        S
3        1142      2 female 0.92     1     2        S
4        1155      3 female 1.00     1     1        S
5        1173      3   male 0.75     1     1        S
6        1188      2 female 1.00     1     2        C
7        1199      3   male 0.83     0     1        S
8        1246      3 female 0.17     1     2        S
```

```
no_ts_infants <- nrow(ts_infants)

tr_infants <- tr_workset %>% filter(Age <= 1)
tr_infants %>% dplyr::select(PassengerId, Pclass, Sex, Age, SibSp, Parch, Embarked)
```

```
   PassengerId Pclass    Sex  Age SibSp Parch Embarked
1           79      2   male 0.83     0     2        S
2          165      3   male 1.00     4     1        S
3          173      3 female 1.00     1     1        S
4          184      2   male 1.00     2     1        S
5          306      1   male 0.92     1     2        S
6          382      3 female 1.00     0     2        C
7          387      3   male 1.00     5     2        S
8          470      3 female 0.75     2     1        C
9          645      3 female 0.75     2     1        C
10         756      2   male 0.67     1     1        S
11         789      3   male 1.00     1     2        S
12         804      3   male 0.42     0     1        C
13         828      2   male 1.00     0     2        C
14         832      2   male 0.83     1     1        S
```

```
no_tr_infants <- nrow(tr_infants)
```

There are 8 infants in the training set, a bit more than half the trainind ones (57.14%). In second thought, I will not do that, I will keep this group in the equations. What I might do is remove the one 80 year old in the training section because it is a suspicious entry–PassengerId=1:

```
tr70 <- tr_workset %>% filter(Age > 69) %>% arrange(desc(Age), Pclass, Embarked)
tr70 %>% dplyr::select(PassengerId, Pclass, Sex, Age, SibSp, Parch, Embarked)
```

```
  PassengerId Pclass  Sex  Age SibSp Parch Embarked
1         631      1 male 80.0     0     0        S
2         852      3 male 74.0     0     0        S
3          97      1 male 71.0     0     0        C
4         494      1 male 71.0     0     0        C
5         117      3 male 70.5     0     0        Q
6         746      1 male 70.0     1     1        S
7         673      2 male 70.0     0     0        S
```

```
tr70 %>% dplyr::select(PassengerId, Name)
```

```
  PassengerId                                 Name
1         631 Barkworth, Mr. Algernon Henry Wilson
2         852                   Svensson, Mr. Johan
3          97              Goldschmidt, Mr. George B
4         494                Artagaveytia, Mr. Ramon
5         117                   Connors, Mr. Patrick
6         746           Crosby, Capt. Edward Gifford
7         673            Mitchell, Mr. Henry Michael
```

and the only condition that seems equal for all is their sex– they are all males. Are there any of these older passengers in the test set?

```
# 70+-years-old in test set
ts70 <- ts_workset %>% filter(Age > 69) %>% arrange(desc(Age), Pclass, Embarked)

ts70 %>% dplyr::select(PassengerId, Pclass, Sex, Age, SibSp, Parch, Embarked)
```

```
  PassengerId Pclass    Sex Age SibSp Parch Embarked
1         988      1 female  76     1     0        S
```

```
# 70+-year-ols in test set
ts70 %>% dplyr::select(PassengerId, Age, Name)
```

```
  PassengerId Age                                                 Name
1         988  76 Cavendish, Mrs. Tyrell William (Julia Florence Siegel)
```

```
# Create variables for text
ts70_cnt <- nrow(ts70)
ts70_age <- "NA"
ts70_sex <- "NA"
if (ts70_cnt > 1) {
    ts70_age <- "70+"
    if (("male" %in% ts70$Sex) & ("female" %in% ts70$Sex)) {
        ts70_sex <- "mixed-sex"
    } else if ("male" %in% ts70$Sex) {
        ts70_sex <- "male(s)"
    } else {
        ts70_sex <- "female(s)"
    }
```

```
} else if (ts70_cnt == 1) {
    ts70_age <- ts70$Age[1]
    ts70_sex <- ts70$Sex[1]
}

# full-join of 70+-years-olds... looking for Mrs Tyrell William Cavendish (Julia
# Florence Siegel) SibSp
full <- ts_workset %>% full_join(tr_workset)
full %>% filter("Cavendish" %in% Name)
```

```
 [1] PassengerId Pclass      Name        Sex         Age         SibSp
 [7] Parch       Ticket      Fare        Cabin       Embarked    Survived
<0 rows> (or 0-length row.names)
```

```
full %>% filter("Siegel" %in% Name)
```

```
 [1] PassengerId Pclass      Name        Sex         Age         SibSp
 [7] Parch       Ticket      Fare        Cabin       Embarked    Survived
<0 rows> (or 0-length row.names)
```

There is/are 1 76-years-old female.

Because it is a female probably travelling with a half-sibling or married sister using her husband's last name (SibSp=1 and no matches were found in either database for her married name–Cavendish, or her maiden name–Siegel.) I am inclined now not to fiddle with these age groups and let the equations solve the problem.

However, we need to create the groups like this:

```
## Notice we are placing Age NA's into bin 10

# For the training set
tr_workset <- tr_workset %>% mutate(age_group = as.factor(ifelse(is.na(Age), 10,
    ifelse(Age < 1, 0, floor(Age/10) + 1))))

# And save it
f_rdsSave(tr_workset, "tr_workset.rds")


# For the test set
ts_workset <- ts_workset %>% mutate(age_group = as.factor(ifelse(is.na(Age), 10,
    ifelse(Age < 1, 0, floor(Age/10) + 1))))

# And save it
f_rdsSave(ts_workset, "ts_workset.rds")
```

We have created a new factor column called **age_group** where we have classified ages in groups of 10 years, as we observed previously in the age analysis section. We have also placed all NA's into a 10th, unexistant group, which we can ignore by value when training the missing values diminished models.

Note that I have not been keeping track of additions to the data sets. We will run a structural and data summary at the end of this section.

5.5 SibSp

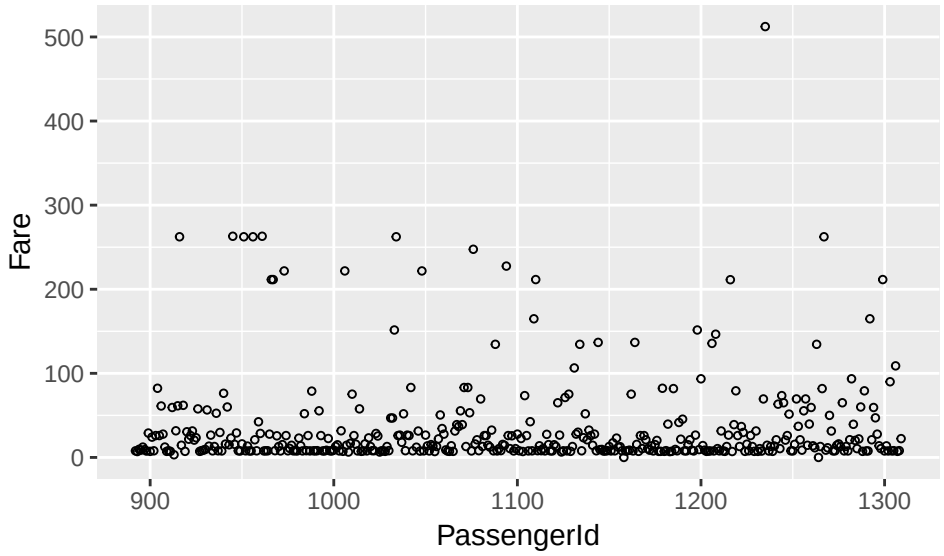
We had decided to convert *SibSp* to factor, but in retrospect it is really a number, so I will leave it like that.

5.6 Parch

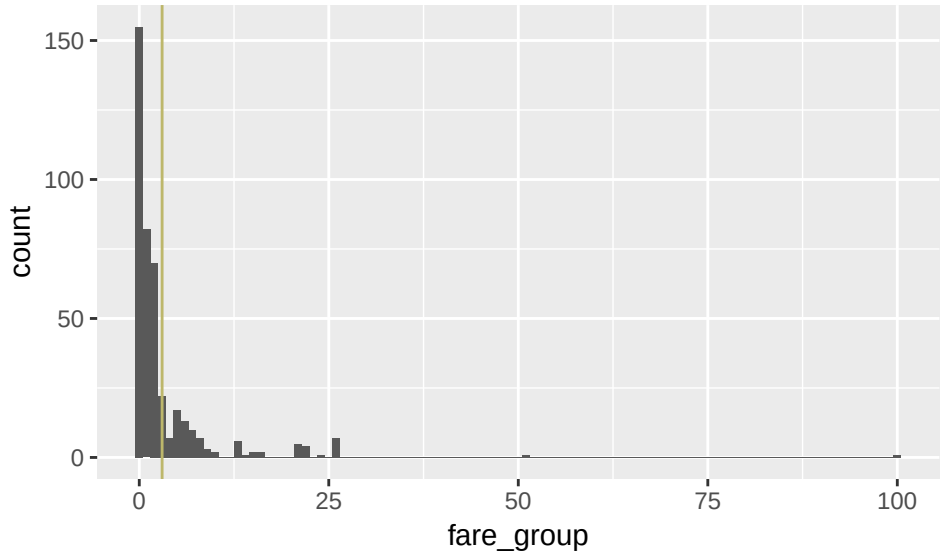
We will also leave *Parch* as is.

5.7 Fare

There is so much difference in fares ranging from $0 to $500+, that I decided to bin it into %10.00 increments. However, I will like to know what kind of fares the test set passengers paid before I decide this. So let us see:



Yes, same range of dispersed amounts as in the train set, so let us do the binning:



Where the average group Fare is of ≈ 3.01 (dark khaki vertical line)–excluding NA's and empty values. So yes, let us go ahead and bin the fares like this:

```r
# -- Train set$10 binned Fare groups
tr_workset <- tr_workset %>% mutate(fare_group = as.factor(ifelse(is.na(Fare), 100,
    floor(Fare/10))))

# And save it
f_rdsSave(tr_workset, "tr_workset.rds")

# -- Test set$10 binned groups
ts_workset <- ts_workset %>% mutate(fare_group = as.factor(ifelse(is.na(Fare), 100,
    floor(Fare/10))))

# And save it
f_rdsSave(ts_workset, "ts_workset.rds")
```

We have created a new factor column called **fare_group** where we have classified the fare in groups of $10, as we observed previously in the fare analysis section. We have also placed all NA's into a 100th, unexistant group, which we can ignore by value when training the missing values diminished models.

5.8 Embarked

We are really not doing anything to this variable other than keep it present to develop an alternate model for missing values.

5.9 Modified Worksets

```
[1] -- TRAINING WORKSET STRUCTURE:

'data.frame':   891 obs. of  14 variables:
 $ PassengerId: int  1 2 ...
 $ Survived   : int  0 1 ...
 $ Pclass     : int  3 1 ...
 $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 ...
 $ Age        : num  22 38 ...
 $ SibSp      : int  1 1 ...
 $ Parch      : int  0 0 ...
 $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 ...
 $ Fare       : num  7.25 ...
 $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 ...
 $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 ...
 $ age_group  : Factor w/ 11 levels "0","1","2","3",..: 4 5 ...
 $ fare_group : Factor w/ 22 levels "0","1","2","3",..: 1 8 ...

[1] -- TESTING WORKSET STRUCTURE:

'data.frame':   418 obs. of  13 variables:
 $ PassengerId: int  892 893 ...
 $ Pclass     : int  3 3 ...
 $ Name       : Factor w/ 418 levels "Abbott, Master. Eugene Joseph",..: 210 409 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 ...
 $ Age        : num  34.5 47 ...
 $ SibSp      : int  0 1 ...
 $ Parch      : int  0 0 ...
 $ Ticket     : Factor w/ 363 levels "110469","110489",..: 153 222 ...
```

```
 $ Fare       : num  7.83 ...
 $ Cabin      : Factor w/ 77 levels "","A11","A18",..: 1 1 ...
 $ Embarked   : Factor w/ 3 levels "C","Q","S": 2 3 ...
 $ age_group  : Factor w/ 10 levels "0","1","2","3",..: 5 6 ...
 $ fare_group : Factor w/ 21 levels "0","1","2","3",..: 1 1 ...

[1] ---------- END OF STRUCTURES -
```

5.10 Worksets Summaries

```
[1] -- TRAIN WORKSET SUMMARY:

  PassengerId       Survived          Pclass
 Min.   :  1.0   Min.   :0.0000   Min.   :1.000
 1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000
 Median :446.0   Median :0.0000   Median :3.000
 Mean   :446.0   Mean   :0.3838   Mean   :2.309
 3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000
 Max.   :891.0   Max.   :1.0000   Max.   :3.000

                                     Name         Sex           Age
 Abbing, Mr. Anthony                   :  1   female:314   Min.   : 0.42
 Abbott, Mr. Rossmore Edward           :  1   male  :577   1st Qu.:20.12
 Abbott, Mrs. Stanton (Rosa Hunt)      :  1                Median :28.00
 Abelson, Mr. Samuel                   :  1                Mean   :29.70
 Abelson, Mrs. Samuel (Hannah Wizosky):  1                3rd Qu.:38.00
 Adahl, Mr. Mauritz Nils Martin        :  1                Max.   :80.00
 (Other)                               :885                NA's   :177
     SibSp           Parch             Ticket         Fare
 Min.   :0.000   Min.   :0.0000   1601    :  7   Min.   :  0.00
 1st Qu.:0.000   1st Qu.:0.0000   347082  :  7   1st Qu.:  7.91
 Median :0.000   Median :0.0000   CA. 2343:  7   Median : 14.45
 Mean   :0.523   Mean   :0.3816   3101295 :  6   Mean   : 32.20
 3rd Qu.:1.000   3rd Qu.:0.0000   347088  :  6   3rd Qu.: 31.00
 Max.   :8.000   Max.   :6.0000   CA 2144 :  6   Max.   :512.33
                                  (Other) :852
         Cabin     Embarked
            :687    :  2
 B96 B98    :  4   C:168
 C23 C25 C27:  4   Q: 77
 G6         :  4   S:644
 C22 C26    :  3
 D          :  3
 (Other)    :186

[1] -- TEST WORKSET SUMMARY:

  PassengerId         Pclass
 Min.   : 892.0   Min.   :1.000
 1st Qu.: 996.2   1st Qu.:1.000
 Median :1100.5   Median :3.000
 Mean   :1100.5   Mean   :2.266
 3rd Qu.:1204.8   3rd Qu.:3.000
 Max.   :1309.0   Max.   :3.000
```

```
                                          Name         Sex           Age
Abbott, Master. Eugene Joseph               :  1   female:152   Min.   : 0.17
Abelseth, Miss. Karen Marie                 :  1   male  :266   1st Qu.:21.00
Abelseth, Mr. Olaus Jorgensen               :  1                Median :27.00
Abrahamsson, Mr. Abraham August Johannes    :  1                Mean   :30.27
Abrahim, Mrs. Joseph (Sophie Halaut Easu):  1                   3rd Qu.:39.00
Aks, Master. Philip Frank                   :  1                Max.   :76.00
(Other)                                     :412                NA's   :86
    SibSp           Parch             Ticket         Fare
Min.   :0.0000   Min.   :0.0000   PC 17608:  5   Min.   :  0.000
1st Qu.:0.0000   1st Qu.:0.0000   113503  :  4   1st Qu.:  7.896
Median :0.0000   Median :0.0000   CA. 2343:  4   Median : 14.454
Mean   :0.4474   Mean   :0.3923   16966   :  3   Mean   : 35.627
3rd Qu.:1.0000   3rd Qu.:0.0000   220845  :  3   3rd Qu.: 31.500
Max.   :8.0000   Max.   :9.0000   347077  :  3   Max.   :512.329
                                  (Other) :396   NA's   :1
             Cabin     Embarked
                :327   C:102
B57 B59 B63 B66:  3   Q: 46
A34             :  2   S:270
B45             :  2
C101            :  2
C116            :  2
(Other)         : 80

[1] ---------- END OF SUMMARIES -
```

5.11 Summary

In short, we have created two new columns in both the training and testing sets:

- **age_group** – Factor with 11 levels[1]: 0="0-12 mos", 1="1-9 yrs", ..., 9="80-89 yrs", 10="NAs"
- **fare_group** – Factor with 52 levels[2] : 0="$0-9", 1="$10-19", ..., 51="$510-519", 100="NAs"

(1) *May show as 9-10 levels in the data structure and summary of previous sections because of missing ages in some groups (refer to age analysis)*

(2) *May show as 21-22 levels in the data structure and summary of previous sections above because of missing fares in some groups (refer to fare analysis)*

6 Models

6.1 Introduction

So far we have analyzed all the variables in the provided data files. We have selected 7 of the 10 listed columns as predictors (**Pclass, Sex, Age, SibSp, Parch, Fare, and Embarked**.) We rejected **Name, Ticket and Cabin**. Some of the selected columns have missing values in the training set (**Age and Embarked**) and in the test set (**Age and Fare**). We added to columns to the datasets, namely **age_group** and **fare_group**. We have not checked for rows with more than one missing value, which we do here:

Where we get that the maximum number of missing values in any one row in the training set is **1**, and **1** in the testing set. Because there is a maximum of 1 missing value in any 1 row of the testing set, we don't need to build 5-variable models, we will need to handle *NA* Fares and *NA* Ages in the testing set, but they do not coincide in the same row, which we can also verify like this:

```
# Calculation for Test set only (remember we stuck na ages in group 10, and na
# fares in group 100)
head(ts %>% filter((is.na(Age) & is.na(Fare)) | (age_group == 10 & fare_group ==
    100)))
```

```
 [1] PassengerId Pclass      Name        Sex         Age         SibSp
 [7] Parch       Ticket      Fare        Embarked    age_group   fare_group
<0 rows> (or 0-length row.names)
```

Therefore, we need one model for all 7 variables (which we call the main model) and two additional 6-variable models to handle prediction in rows where either the Age or the Fare fields are missing, for a total of 3 systems of equations–not so bad after all.

To make the models easier to represent, I want to label the different variables like this:

$v_1 = \text{Age}$

$v_2 = \text{Pclass}$

$v_3 = \text{Sex}$

$v_4 = \text{SibSp}$

$v_5 = \text{Pclass}$

$v_6 = \text{Embarked}$

$v_7 = \text{Fare}$

6.2 The Main Model

The main model we explain earlier but resta here:

$$Survived = f_m(v_1, v_2, ..., v_7) + \varepsilon_m$$

where the $_m$ in f_m stands for *main*, v_i are the different predictors or independent variables as defined here, and ε, is the independent zero-centered residual explained by random variation corresponding to this model.

Later on during cross-validation we will introduce different methods to solve this system of equations. Here we are just defining the models.

6.3 The Age Model

The age model is designed to predict in the presence of Age missing values. All rows containg missing values in the Fare column must be ignored when training this model. The model is defined like this:

$$Survived = f_a(v_2, v_3, ..., v_7) + \varepsilon_a$$

where the $_a$ in f_a stands for *age*, v_i are the different predictors or independent variables as defined here (note how the subscripts start at 2), and ε_a, is the independent zero-centered residual explained by random variation corresponding to this model.

Later on during cross-validation we will introduce different methods to solve this system of equations. Here we are just defining the models.

6.4 The Fare Model

The age model is designed to predict in the presence of Fare missing values. All rows containg missing values in the Age column must be ignored when training this model. The model is defined like this::

$$Survived = f_f(v_1, v_2, ..., v_6) + \varepsilon_f$$

where the $_f$ in f_f stands for *fare*, v_i are the different predictors or independent variables as defined here (note how the subscripts end at 6), and ε_f, is the independent zero-centered residual explained by random variation corresponding to this model.

Later on during cross-validation we will introduce different methods to solve this system of equations. Here we are just defining the models.

7 Methods

In a previous section we identified the inputs to our system (the independent variables) later described how we were going to use those variables to produce the desired result of predicting surviving the sinking of RMS Titanic in this fashion:

$$Inputs -> f_x() -> Output$$

We never mentioned how we were going to operate on the inputs to produce the desired result, that is, we never specified what were the $f_x()$'s above, or if we were going to test a handful of methods to chose the most suitable one, that is, the one with the smallest amount of error.

This is the moment to formally introduce the methods that we will use to transform the described *inputs* (variables) into the *output* (prediction) and indicate that we will test them against each other using *cross-validation* to select the one that maximizes most the *Accuracy* of our predictions. To that purpose, we have selected the following methods:

1. **knn**: classification
3. **gbm**: generalized boosted regression modeling
3. **glm**: multivariate logistic
4. **nn**: neural network
5. **rpart**: recursive partitioning and regression trees

7.1 Cross-Validation

Cross validation is a technique used for assessing the performance of machine learning models. It helps in knowing how a model would generalize an independent data set to predict, and to learn how accurate the model's predictions will be in practice.

During cross-validations we will be splitting the training set successively in two parts: the training part and the validation part, both of which change distinctively (in the elements they include) for every fold.

To assure that every solution method receives the same partitions for cross-validations, we placed a call to *set.seed(11, sample.kind = "Rounding")*.

7.2 Validation Preprocessing

In order to streamline our system and minimize computation time (there will be much passing the data sets around and partitioning during the extraction of the validation sets, we will remove all the unused columns from both the training and testing datasets like this*:

```
# For the training set
tr_final <- tr_workset
tr_final$Name <- NULL
tr_final$Ticket <- NULL
tr_final$Cabin <- NULL

# And Save it
f_rdsSave(tr_final, "tr_final.rds")

# For the testing set
ts_final <- ts_workset
ts_final$Name <- NULL
ts_final$Ticket <- NULL
```

```
ts_final$Cabin <- NULL

# And Save it
f_rdsSave(ts_final, "ts_final.rds")
```

(*) *We will actually not need that version of the test file until prediction time. The original **Age** and **Fare** columns have been left in the final datasets for prediction. There will still be a total of 3 models, but one of 9 independent variables, instead of 7; the other two of 7, instead of 6.*

In addition to that, we have to remove the rows containing missing values from the training set for training the main model, and convert the predicted column, *Survived*, to a factor so that the **accuracy** statistic is reported by the training algorithms:

```
# Remove rows with empty values for training main model We have to remove
# embarkment because it is one of the selected predictors for main, butt it is
# not missing values in the test set
tr_main <- tr_final %>% mutate(lived = as.factor(Survived)) %>% filter(!is.na(Age) &
    !is.na(Fare) & (Embarked != ""))
noquote(paste("After removing missing value rows, the main set was left with", nrow(tr_main),
    "rows."))
```

```
[1] After removing missing value rows, the main set was left with 712 rows.
```

```
# Remove rows with empty values for Age to train age model have to remove empty
# fare because it is used by age model
tr_age <- tr_final %>% mutate(lived = as.factor(Survived)) %>% filter(!is.na(Fare) &
    (Embarked != ""))
noquote(paste("After removing missing value rows, the age set was left with", nrow(tr_age),
    "rows for training."))
```

```
[1] After removing missing value rows, the age set was left with 889 rows for training.
```

```
# Remove rows with empty values for Fare to train fare model have to remove empty
# ages because it is used by fare model
tr_fare <- tr_final %>% mutate(lived = as.factor(Survived)) %>% filter(!is.na(Age) &
    (Embarked != ""))
noquote(paste("After removing missing value rows, the fare set was left with", nrow(tr_fare),
    "rows for training."))
```

```
[1] After removing missing value rows, the fare set was left with 712 rows for training.
```

```
# And Save them
f_rdsSave(tr_main, "tr_main.rds")
f_rdsSave(tr_age, "tr_age.rds")
f_rdsSave(tr_fare, "tr_fare.rds")
```

We have been saving milestone snapshots of the datasets in case we need to refer to them later without having to execute the entire set of instructions that let to their state at that time.

7.3 The Main Model

7.3.1 K-Nearest Neighbors (KNN)

We ran cross-validation using the KNN prediction method like this:

```
library(caret)

# define training control
t_ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10, savePredictions = TRUE)
if (R___DEBUG > 0) {
    # Quicker version for debuging
    t_ctrl <- trainControl(method = "repeatedcv", number = 3, repeats = 1, savePredictions = TRUE)
}

# train the model
library(gbm)
set.seed(11, sample.kind = "Rounding")
Model_knn <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
    age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "knn")
```

7.3.2 Generalized Boosted Regression Modeling (GBM)

We ran cross-validation using the KNN prediction method like this:

```
library(gbm)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_gbm <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
    age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "gbm", verbose = FALSE)
```

7.3.3 Generalized Linear Models (GLM) Logistic

We ran cross-validation using the GLM-logistic (family='binomial') prediction method like this:

```
# train the model
set.seed(11, sample.kind = "Rounding")
Model_glm <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
    age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "glm", family = "binomial")
```

7.3.4 Neural Networks (NN)

We ran cross-validation using the NN prediction method like this:

```
library(MASS)

# train the model
set.seed(11, sample.kind = "Rounding")
if (R___DEBUG > 0) {
    # Quicker Debug
    Model_nn <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
        age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "nnet",
        maxit = 100, trace = FALSE)
} else {
    # Slower release
    Model_nn <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
        age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "nnet",
        maxit = 1000, trace = FALSE)
}
```

7.3.5 Recursive Partitioning and Regression Trees (RPART)

We ran cross-validation using the RPART prediction method like this:

```
library(rpart)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_rpart <- train(lived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
    age_group + fare_group, data = tr_main, trControl = t_ctrl, method = "rpart")
```

7.4 The Age Model

7.4.1 K-Nearest Neighbors (KNN)

We ran cross-validation using the KNN prediction method like this:

```
library(caret)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_aknn <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
    data = tr_age, trControl = t_ctrl, method = "knn")
```

7.4.2 Generalized Boosted Regression Modeling (GBM)

We ran cross-validation using the KNN prediction method like this:

```
library(gbm)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_agbm <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
    data = tr_age, trControl = t_ctrl, method = "gbm", verbose = FALSE)
```

7.4.3 Generalized Linear Models (GLM) Logistic

We ran cross-validation using the GLM-logistic prediction method like this:

```
# train the model
set.seed(11, sample.kind = "Rounding")
Model_aglm <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
    data = tr_age, trControl = t_ctrl, method = "glm", family = "binomial")
```

7.4.4 Neural Networks (NN)

We ran cross-validation using the NN prediction method like this:

```
library(MASS)

# train the model
set.seed(11, sample.kind = "Rounding")
```

```
if (R___DEBUG > 0) {
    Model_ann <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
        data = tr_age, trControl = t_ctrl, method = "nnet", maxit = 100, trace = FALSE)
} else {
    Model_ann <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
        data = tr_age, trControl = t_ctrl, method = "nnet", maxit = 1000, trace = FALSE)
}
```

7.4.5 Recursive Partitioning and Regression Trees (RPART)

We ran cross-validation using the RPART prediction method like this:

```
library(rpart)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_arpart <- train(lived ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + fare_group,
    data = tr_age, trControl = t_ctrl, method = "rpart")
```

7.5 The Fare Model

7.5.1 K-Nearest Neighbors (KNN)

We ran cross-validation using the KNN prediction method like this:

```
library(caret)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_fknn <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
    data = tr_fare, trControl = t_ctrl, method = "knn")
```

7.5.2 Generalized Boosted Regression Modeling (GBM)

We ran cross-validation using the KNN prediction method like this:

```
library(gbm)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_fgbm <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
    data = tr_fare, trControl = t_ctrl, method = "gbm", verbose = FALSE)
```

7.5.3 Generalized Linear Models (GLM) Logistic

We ran cross-validation using the GLM-logistic prediction method like this:

```
# train the model
set.seed(11, sample.kind = "Rounding")
Model_fglm <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
    data = tr_fare, trControl = t_ctrl, method = "glm", family = "binomial")
```

7.5.4 Neural Networks (NN)

We ran cross-validation using the NN prediction method like this:

```
library(MASS)

# seed and train the model
set.seed(11, sample.kind = "Rounding")
if (R___DEBUG > 0) {
    Model_fnn <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
        data = tr_fare, trControl = t_ctrl, method = "nnet", maxit = 100, trace = FALSE)
} else {
    Model_fnn <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
        data = tr_fare, trControl = t_ctrl, method = "nnet", maxit = 1000, trace = FALSE)
}
```

7.5.5 Recursive Partitioning and Regression Trees (RPART)

We ran cross-validation using the RPART prediction method like this:

```
library(rpart)

# train the model
set.seed(11, sample.kind = "Rounding")
Model_frpart <- train(lived ~ Pclass + Sex + SibSp + Parch + Age + Embarked + age_group,
    data = tr_fare, trControl = t_ctrl, method = "rpart")
```

8 Results

In this section, we run and display the results of cross-validating our three models across their five solution methods. We have left the *caret* package function **train** handle cross-validations, by setting its train control object options *method*, *number*, and *repetitions*, to *repeatedcv* (repeated cross-validation), 10 k-folds and 10 repetitions, respectively (100 resamples.) If you are planning to run this script quickly, make sure to set the global variable R____DEBUG to *DBG_LEVEL_FST* to reduce training to three k-samples of two repetitions (6 resamples), among other time reductions, since the non-debug version will take many more minutes to complete.

I could have added more methods to these models but thought the number was a good sample for the extent of this exercise.

As mentioned earlier, *set.seed* is called with an identical seed before cross-validating the methods to provide the same k-samples to all. However, changing the number of k-samples and the number of repetitions to *train* may lead to different results from *knit* to *knit*, which I was able to experience from differences between the debug and release versions of this document. The debug version will also limit the number of maximum iterations (*maxit*)on the neural network methods to 100 from 1000 in the release version. So the neural network method will generalize better under the release version.

8.1 Main Model

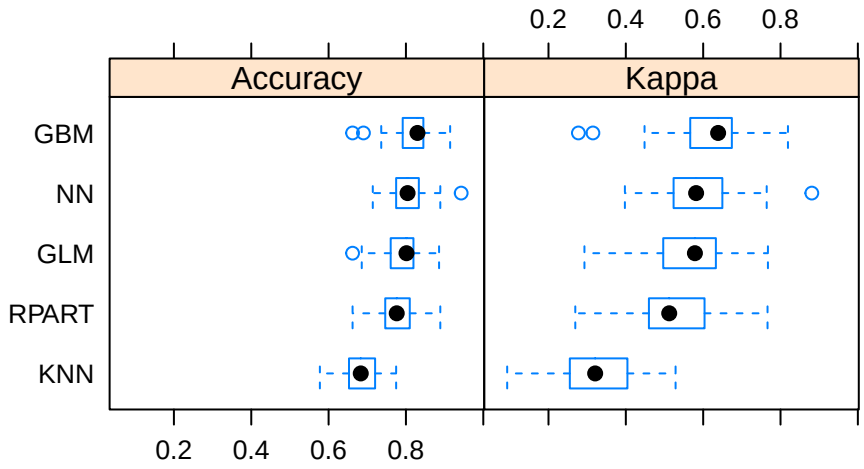
These are the results of cross-validating the main model methods:

```
Call:
summary.resamples(object = m_res)

Models: KNN, GBM, GLM, NN, RPART
Number of resamples: 100

Accuracy
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.5774648 0.6527778 0.6831349 0.6834783 0.7192879 0.7746479    0
GBM   0.6619718 0.7916667 0.8297787 0.8228886 0.8450704 0.9142857    0
GLM   0.6619718 0.7605634 0.8014085 0.7906955 0.8194444 0.8857143    0
NN    0.7142857 0.7746479 0.8041862 0.8040808 0.8333333 0.9428571    0
RPART 0.6619718 0.7464789 0.7762128 0.7807266 0.8077381 0.8888889    0

Kappa
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.09319899 0.2562877 0.3206842 0.3265232 0.4039695 0.5286307    0
GBM   0.27735369 0.5672081 0.6386155 0.6251944 0.6741761 0.8192771    0
GLM   0.29294606 0.4968945 0.5786197 0.5617933 0.6318723 0.7674419    0
NN    0.39759036 0.5234899 0.5817427 0.5863029 0.6496350 0.8809524    0
RPART 0.26929674 0.4605547 0.5121672 0.5285540 0.5998920 0.7664234    0
```



The boxplot above has taken care of sorting the methods according to median accuracy. In contrast, we will be selecting the highest mean accuracy, which order can be different at times.

8.2 Age Model

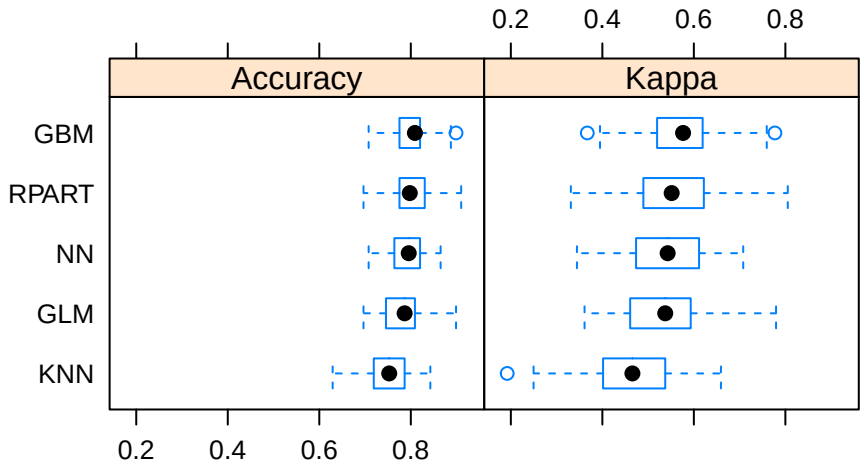
These are the results of cross-validating the age model methods:

```
Call:
summary.resamples(object = a_res)

Models: KNN, GBM, GLM, NN, RPART
Number of resamples: 100

Accuracy
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.6292135 0.7191011 0.7528090 0.7521885 0.7865169 0.8426966    0
GBM   0.7078652 0.7752809 0.8089888 0.8040513 0.8202247 0.8988764    0
GLM   0.6966292 0.7478933 0.7865169 0.7792837 0.8089888 0.8988764    0
NN    0.7078652 0.7640449 0.7954545 0.7938036 0.8202247 0.8651685    0
RPART 0.6966292 0.7752809 0.7977528 0.8003422 0.8300243 0.9101124    0

Kappa
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.1920220 0.4016528 0.4657012 0.4614234 0.5361290 0.6591904    0
GBM   0.3670678 0.5201876 0.5764131 0.5735741 0.6170652 0.7770665    0
GLM   0.3610742 0.4621155 0.5374574 0.5309554 0.5931702 0.7796424    0
NN    0.3444759 0.4736694 0.5427111 0.5481314 0.6109612 0.7078775    0
RPART 0.3311996 0.4896789 0.5515118 0.5552819 0.6206747 0.8052516    0
```



The boxplot above has taken care of sorting the methods according to median accuracy. In contrast, we will be selecting the highest mean accuracy, which order can be different at times.

8.3 Fare Model

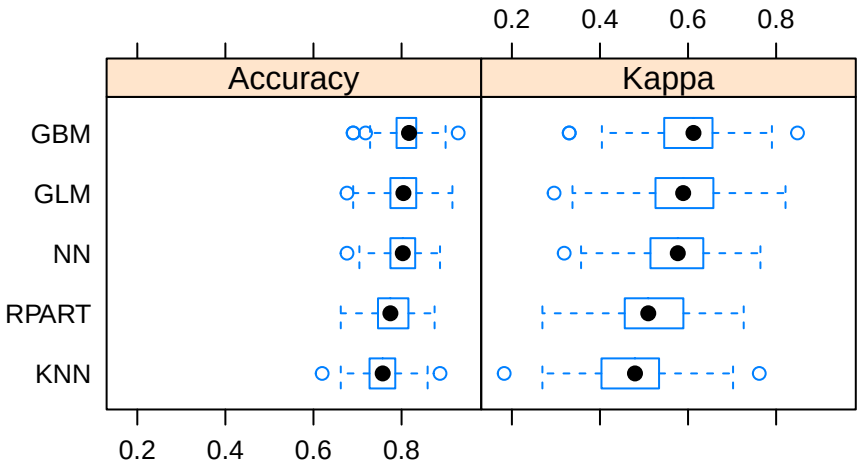
These are the results of cross-validating the fare model methods:

```
Call:
summary.resamples(object = f_res)

Models: KNN, GBM, GLM, NN, RPART
Number of resamples: 100

Accuracy
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.6197183 0.7298513 0.7571429 0.7565977 0.7857143 0.8873239    0
GBM   0.6901408 0.7887324 0.8169014 0.8155940 0.8333333 0.9285714    0
GLM   0.6760563 0.7746479 0.8041862 0.8047831 0.8333333 0.9154930    0
NN    0.6760563 0.7746479 0.8028169 0.8007261 0.8309859 0.8873239    0
RPART 0.6619718 0.7464789 0.7746479 0.7770762 0.8149396 0.8750000    0

Kappa
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
KNN   0.1825160 0.4038942 0.4795170 0.4743985 0.5341615 0.7617450    0
GBM   0.3301887 0.5458422 0.6122871 0.6083673 0.6545301 0.8484848    0
GLM   0.2958172 0.5273455 0.5888936 0.5915814 0.6557294 0.8213087    0
NN    0.3187317 0.5154957 0.5766610 0.5746723 0.6346484 0.7643154    0
RPART 0.2692967 0.4568549 0.5096669 0.5157471 0.5877039 0.7263514    0
```



The boxplot above has taken care of sorting the methods according to median accuracy. In contrast, we will be selecting the highest mean accuracy, which order can be different at times.

8.4 Predictions

At this time can now solve the task to predict survival with the passenger information in the *test dataset*. We will separate the test dataset for the different models from the prepared "ts_final" set we saved earlier and filter this set according to the model's requirement. Because we devised three models, we will need to bind the rows resulting from each model's predictions to produce the final result.

We can retrieve the best performing methods for our models like this:

```
# Find best model method
methods <- c("KNN", "GBM", "GLM", "NN", "RPART")
m_models <- c("Model_knn", "Model_gbm", "Model_glm", "Model_nn", "Model_rpart")
a_models <- c("Model_aknn", "Model_agbm", "Model_aglm", "Model_ann", "Model_arpart")
f_models <- c("Model_fknn", "Model_fgbm", "Model_fglm", "Model_fnn", "Model_frpart")

# Function to retrieve best method
f_bestMethod <- function(res) {

    acc <- sapply(methods, function(m) {
        stat <- paste("res$values$\"", m, "~", "Accuracy\"", sep = "")
        return(mean(eval(parse(text = stat))))
    })

    df <- data.frame(Method = methods, Accuracy = acc) %>% arrange(desc(Accuracy))
    return(df[1, ])
}

m_best <- f_bestMethod(m_res)
a_best <- f_bestMethod(a_res)
f_best <- f_bestMethod(f_res)

m_name <- m_models[which(methods == m_best$Method[1])]
a_name <- a_models[which(methods == a_best$Method[1])]
f_name <- f_models[which(methods == f_best$Method[1])]
```

Table 17: Best Performing Methods

	Method	Accuracy
Main Model	GBM	0.8228886
Age Model	GBM	0.8040513
Fare Model	GBM	0.8155940

It is worth mentioning that the best performing methods for all three models can be retrieved like this:

```
eval(parse(text = paste(m_name, "$finalModel", sep = "")))
```

```
A gradient boosted model with bernoulli loss function.
150 iterations were performed.
There were 40 predictors of which 20 had non-zero influence.
```

```
eval(parse(text = paste(a_name, "$finalModel", sep = "")))
```

```
A gradient boosted model with bernoulli loss function.
150 iterations were performed.
There were 29 predictors of which 14 had non-zero influence.
```

```
eval(parse(text = paste(f_name, "$finalModel", sep = "")))
```

```
A gradient boosted model with bernoulli loss function.
100 iterations were performed.
There were 18 predictors of which 12 had non-zero influence.
```

From which we can calculate the overall system accuracy for our task like this:

$$SystemAccuracy = \frac{\sum_i A_{mi} + \sum_i A_{ai} + \sum_i A_{fi}}{(481 * K)}$$

Where 418 is the number of rows in the test dataset, K are the number of k-folds, A_{mi} are the validation fold accuracies from the main model, A_{ai}, from the age model, and A_{fi}, from the fare model, which we calculate like this:

```
# The prediction data sets -- we need these numbers
ts_main <- (ts_final %>% filter(!is.na(Age) & !is.na(Fare)))
ts_age <- (ts_final %>% filter(is.na(Age)))
ts_fare <- (ts_final %>% filter(is.na(Fare)))

# nrow(ts_main) pedicted with main model, nrow(ts_age) with the age model and
# nrow(ts_fare) with the fare model such that nrow(test_set) == nrow(ts_main) +
# nrow(ts_age) + nrow(ts_fare)

sys_accuracy <- (nrow(ts_main) * mean(eval(parse(text = paste(m_name, "$results$Accuracy",
    sep = "")))) + nrow(ts_age) * mean(eval(parse(text = paste(a_name, "$results$Accuracy",
    sep = "")))) + nrow(ts_fare) * mean(eval(parse(text = paste(f_name, "$results$Accuracy",
    sep = "")))))/nrow(test_set)

noquote(paste("Expected overall accuracy predicting the test dataset = ", round(sys_accuracy,
    rnd_detail)))
```

```
[1] Expected overall accuracy predicting the test dataset =  0.8064
```

```r
# The predictions
p_main <- as.data.frame(predict(Model_gbm, ts_main))
p_age <- as.data.frame(predict(Model_agbm, ts_age))
p_fare <- as.data.frame(predict(Model_fgbm, ts_fare))

# For Main
ts_main$Survived <- p_main[, 1]
if (R___DEBUG > 1) {
    summarize(ts_main)
    head(ts_main %>% dplyr::select(PassengerId, Survived))
}

# For Age
ts_age$Survived <- p_age[, 1]
if (R___DEBUG > 1) {
    summarize(ts_age)
    head(ts_age %>% dplyr::select(PassengerId, Survived))
}

# For Fare
ts_fare$Survived <- p_fare[, 1]
if (R___DEBUG > 1) {
    summarize(ts_fare)
    head(ts_fare %>% dplyr::select(PassengerId, Survived))
}

# Final result
ts_res <- full_join(ts_main, ts_age) %>% full_join(ts_fare) %>% dplyr::select(PassengerId,
    Survived)

# Nice table for first 10 predictions
head(ts_res, n = 10) %>% knitr::kable(caption = "First 10 predictions")
```

Table 18: First 10 predictions

PassengerId	Survived
892	0
893	0
894	0
895	0
896	0
897	0
898	0
899	0
900	1
901	0

```r
str(ts_res)
```

```
'data.frame':   418 obs. of  2 variables:
 $ PassengerId: int  892 893 ...
 $ Survived   : Factor w/ 2 levels "0","1": 1 1 ...
```

Which shows we have predicted the 418 rows in the test dataset with an expected weighted overall accuracy of $\approx$ 0.8064.

9 Conclusion

Throughout the development of this report, we were able to examine all the information that was available on some of the passengers that were onboard the *RMS Titanic* at the time of its sinking. Many of them perished, others survived. It was our task to predict surviving and non-surviving passengers from another list on which the survival field was withheld, and which we received as the *test dataset*.

During the study of this information, we discovered irregularities in the data such as missing values and hard to decipher column field meanings with multiple entries that had little bearing on survival, in the other side of the spectrum, we identified other data characteristics, such as the influence that sex and age had on the outcome.

We took action to support the use of variables of great impact to the prediction system despite their shortcomings by designing models that could handle instances where the relevant variables were missing or disparate, such as it was the case with the *Age* and *Fare* columns.

The models were initially proposed as black boxes that we replaced with five methods each, compatible with the *caret package* function *train*, and integrated in a way that permitted their inclusion in our prediction assessment scheme under a single interface.

After testing the models' solution methods, we gained an idea, from cross-validation, of how would these methods perform on real data (see results from the reported accuracies), which then allowed us to select the most accurate ones and used them to predict the test dataset, which we detailed in the previous predictions section.

Perhaps the accuracy of the models we developed in this report is not enough to place this solution in Kaggle's leaderboard, but sufficient to shed a certain aura of confidence and as a mechanism to display our newly-found skills in data science.

---

10 Miscellaneous

10.1 Notes

You can execute the associated R scripts if you:

- Create a **titanic** directory under your home directory, to copy the **titanic.rmd** file provided with this distribution (you can use *knitr::purl* to generate the R scripts from the rmd file like this: *knitr::purl("titanic.rmd", output = "titanic.R", documentation = 1)*, or copy here the *titanic.R* file also included with this distribution.
- Create a **./data/cvs** directory under ~/**titanic**, where Kaggle's **train.csv** and **test.csv** files should be copied.
- Remember that the distributed RMD file was used to generate the final PDF and thus had R___DEBUG set to *DBG_LEVEL_RLS*, which takes longer than 10 minutes to execute. Setting R____DEBUG to *DBG_LEVEL_FST* within this file will reduce this time to under ≈ 40 seconds.

Thank you!

10.2 System Information

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 17763)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] forcats_0.4.0   stringr_1.4.0   dplyr_0.8.1     purrr_0.3.2
[5] readr_1.3.1     tidyr_0.8.3     tibble_2.1.1    ggplot2_3.1.1
[9] tidyverse_1.2.1

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.1       cellranger_1.1.0 pillar_1.4.0     compiler_3.6.0
 [5] plyr_1.8.4       tools_3.6.0      digest_0.6.18    lubridate_1.7.4
 [9] jsonlite_1.6     evaluate_0.13    nlme_3.1-139     gtable_0.3.0
[13] lattice_0.20-38  pkgconfig_2.0.2  rlang_0.3.4      cli_1.1.0
[17] rstudioapi_0.10  yaml_2.2.0       haven_2.1.0      xfun_0.7
[21] withr_2.1.2      xml2_1.2.0       httr_1.4.0       knitr_1.22
[25] hms_0.4.2        generics_0.0.2   grid_3.6.0       tidyselect_0.2.5
[29] glue_1.3.1       R6_2.4.0         readxl_1.3.1     rmarkdown_1.12
[33] modelr_0.1.4     magrittr_1.5     backports_1.1.4  scales_1.0.0
[37] htmltools_0.3.6  rvest_0.3.4      assertthat_0.2.1 colorspace_1.4-1
[41] stringi_1.4.3    lazyeval_0.2.2   munsell_0.5.0    broom_0.5.2
[45] crayon_1.3.4
```

10.3 Execution Time

```
[1] Total execution time: 13.18 mins.
```