Summary of Data Science Project Analysis Steps

Step 1: Define the Problem

Project Summary:

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, resulting in the loss of 1,502 out of 2,224 passengers and crew. This sensational tragedy shocked the international community and prompted the implementation of improved safety regulations for ships.

One of the reasons this shipwreck resulted in such a high loss of life was the insufficient number of lifeboats available for the passengers and crew. While luck played a role in surviving the sinking, certain groups of people were more likely to survive than others, including women, children, and the upper class.

In this challenge, we invite you to complete the analysis of the characteristics that influenced a person's likelihood of survival. Specifically, we request that you utilize machine learning tools to predict which passengers survived this tragedy.

Step 2: Gather the Data

Step 3: Prepare Data for Consumption

3.1 Import Libraries

In [1]:
```
#Load packages
import sys
print("Python version: {}". format(sys.version))

import pandas as pd
print("pandas version: {}". format(pd.__version__))
```

```
import matplotlib
print("matplotlib version: {}". format(matplotlib.__version__))

import numpy as np
print("NumPy version: {}". format(np.__version__))

import scipy as sp
print("SciPy version: {}". format(sp.__version__))

import IPython
from IPython import display
print("IPython version: {}". format(IPython.__version__))

import sklearn
print("scikit-learn version: {}". format(sklearn.__version__))

#misc libraries
import random
import time


#ignore warnings
import warnings
warnings.filterwarnings('ignore')
print('-'*25)

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))
```

```
Python version: 3.6.4 |Anaconda custom (64-bit)| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0]
pandas version: 0.20.3
matplotlib version: 2.1.0
NumPy version: 1.13.3
SciPy version: 1.0.0
IPython version: 6.1.0
scikit-learn version: 0.19.1
-------------------------
gender_submission.csv
test.csv
train.csv
```

3.11 Load Data Modelling Libraries

We will use the popular *scikit-learn* library to develop our machine learning algorithms. In *sklearn*, algorithms are called Estimators and implemented in their own classes. For data visualization, we will use the *matplotlib* and *seaborn* library.

In [2]:
```
#Common Model Algorithms
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, discriminant_analysis, gaussian_process
from xgboost import XGBClassifier

#Common Model Helpers
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics

#Visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
from pandas.tools.plotting import scatter_matrix

#Configure Visualization Defaults
#%matplotlib inline = show plots in Jupyter Notebook browser
%matplotlib inline
mpl.style.use('ggplot')
sns.set_style('white')
pylab.rcParams['figure.figsize'] = 12,8
```

3.2 Understanding the Data

Get to know the data, What does it look like (datatype and values), what makes it tick (independent/feature variables(s)), what's its goals in life (dependent/target variable(s)).

To begin this step, we first import the data. Next we use the info() and sample() function, to get a quick overview of variable datatypes (i.e. qualitative vs quantitative).

1. The *Survived* variable is our outcome or dependent variable. It is a binary nominal datatype of 1 for survived and 0 for did not survive. All other variables are potential predictor or independent variables. **It's important to note, more predictor variables do not make a better model, but the right variables.**

2. The *PassengerID* and *Ticket* variables are assumed to be random unique identifiers, that have no impact on the outcome variable. Thus, they will be excluded from analysis.
3. The *Pclass* variable is an ordinal datatype for the ticket class, a proxy for socio-economic status (SES), representing 1 = upper class, 2 = middle class, and 3 = lower class.
4. The *Name* variable is a nominal datatype. It could be used in feature engineering to derive the gender from title, family size from surname, and SES from titles like doctor or master. Since these variables already exist, we'll make use of it to see if title, like master, makes a difference.
5. The *Sex* and *Embarked* variables are a nominal datatype. They will be converted to dummy variables for mathematical calculations.
6. The *Age* and *Fare* variable are continuous quantitative datatypes.
7. The *SibSp* represents number of related siblings/spouse aboard and *Parch* represents number of related parents/children aboard. Both are discrete quantitative datatypes. This can be used for feature engineering to create a family size and is alone variable.
8. The *Cabin* variable is a nominal datatype that can be used in feature engineering for approximate position on ship when the incident occurred and SES from deck levels. However, since there are many null values, it does not add value and thus is excluded from analysis.

In [3]:
```
#import data from file
data_raw = pd.read_csv('../input/train.csv')


#a dataset should be broken into 3 splits: train, test, and (final) validation
#the test file provided is the validation file

data_val  = pd.read_csv('../input/test.csv')


#to play with our data we'll create a copy

data1 = data_raw.copy(deep = True)

#however passing by reference is convenient, because we can clean both datasets at once
data_cleaner = [data1, data_val]


#preview data
print (data_raw.info())
#data_raw.head()
#data_raw.tail()
data_raw.sample(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
388	389	0	3	Sadlier, Mr. Matthew	male	NaN	0	0	367655	7.7292	NaN	Q
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
332	333	0	1	Graham, Mr. George Edward	male	38.0	0	1	PC 17582	153.4625	C91	S
251	252	0	3	Strom, Mrs. Wilhelm (Elna Matilda Persson)	female	29.0	1	1	347054	10.4625	G6	S
358	359	1	3	McGovern, Miss. Mary	female	NaN	0	0	330931	7.8792	NaN	Q
229	230	0	3	Lefebre, Miss. Mathilde	female	NaN	3	1	4133	25.4667	NaN	S
117	118	0	2	Turpin, Mr. William John Robert	male	29.0	1	0	11668	21.0000	NaN	S
824	825	0	3	Panula, Master. Urho Abraham	male	2.0	4	1	3101295	39.6875	NaN	S
294	295	0	3	Mineff, Mr. Ivan	male	24.0	0	0	349233	7.8958	NaN	S
210	211	0	3	Ali, Mr. Ahmed	male	24.0	0	0	SOTON/O.Q. 3101311	7.0500	NaN	S

3.21 The 4 C's of Data Cleaning: Correcting, Completing, Creating, and Converting

In this stage, we will clean our data by 1) correcting aberrant values and outliers, 2) completing missing information, 3) creating new features for analysis, and 4) converting fields to the correct format for calculations and presentation.

Correcting: Upon reviewing the data, it appears that there are no aberrant or unacceptable data inputs. Additionally, we have identified potential outliers in age and fare. However, since these values seem reasonable, we will postpone any decisions regarding their inclusion or exclusion from the dataset until after we complete our exploratory analysis. It's important to note that if these values were truly unreasonable, such as an age of 800 instead of 80, it would be advisable to address them now. Nevertheless, we must exercise caution when modifying data from its original values, as this may impact the creation of an accurate model.

Completing: There are null values or missing data in the age, cabin, and embarked fields. Missing values can be problematic because some algorithms do not handle null values, resulting in failure. However, other algorithms, like decision trees, can manage null values. Therefore, it's crucial to address this issue before we commence modeling, considering that we will be comparing and contrasting several models. Two common methods for handling missing data are either deleting the record or populating the missing value using a reasonable input. Deleting the record is not recommended, especially when a large percentage of records are affected unless it genuinely represents an incomplete record. Instead, it's best to impute missing values. A basic methodology for qualitative data is to impute using the mode, while for quantitative data, it involves imputing using the mean, median, or the mean plus a randomized standard deviation. An intermediate approach is to apply the basic methodology based on specific criteria, such as the average age by class or embark port by fare and socioeconomic status (SES). There are more complex methodologies available, but before deployment, they should be compared to the baseline model to determine if complexity indeed adds value. For this dataset, age will be imputed with the median, the cabin attribute will be dropped, and embark will be imputed with the mode. Subsequent model iterations may modify this decision to assess its impact on the model's accuracy.

Creating: Feature engineering involves utilizing existing features to create new ones that can potentially provide new signals for predicting our outcome. For this dataset, we will introduce a "title" feature to investigate whether it played a role in survival.

Converting: Lastly, but certainly not least, we'll address formatting. In this context, we are dealing with datatype formats rather than date or currency formats. Our categorical data has been imported as objects, making it challenging for mathematical calculations. For this dataset, we will convert object datatypes into categorical dummy variables.

In [4]:
```
print('Train columns with null values:\n', data1.isnull().sum())
print("-"*10)
```

```
print('Test/Validation columns with null values:\n', data_val.isnull().sum())
print("-"*10)

data_raw.describe(include = 'all')
```

```
Train columns with null values:
 PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
----------
Test/Validation columns with null values:
 PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
----------
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Kilgannon, Mr. Thomas J	male	NaN	NaN	NaN	347082	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

3.22 Clean Data

In [5]:

```
###COMPLETING: complete or delete missing values in train and test/validation dataset
for dataset in data_cleaner:
    #complete missing age with median
    dataset['Age'].fillna(dataset['Age'].median(), inplace = True)

    #complete embarked with mode
    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace = True)

    #complete missing fare with median
    dataset['Fare'].fillna(dataset['Fare'].median(), inplace = True)

#delete the cabin feature/column and others previously stated to exclude in train dataset
drop_column = ['PassengerId','Cabin', 'Ticket']
data1.drop(drop_column, axis=1, inplace = True)

print(data1.isnull().sum())
```

```
print("-"*10)
print(data_val.isnull().sum())
```

```
Survived    0
Pclass      0
Name        0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
----------
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

In [6]:

```
###CREATE: Feature Engineering for train and test/validation dataset
for dataset in data_cleaner:
    #Discrete variables
    dataset['FamilySize'] = dataset ['SibSp'] + dataset['Parch'] + 1

    dataset['IsAlone'] = 1 #initialize to yes/1 is alone
    dataset['IsAlone'].loc[dataset['FamilySize'] > 1] = 0 # now update to no/0 if family size is greater than 1

    #quick and dirty code split title from name
    dataset['Title'] = dataset['Name'].str.split(", ", expand=True)[1].str.split(".", expand=True)[0]


    #Continuous variable bins; qcut vs cut
    #Fare Bins/Buckets using qcut or frequency bins
    dataset['FareBin'] = pd.qcut(dataset['Fare'], 4)

    #Age Bins/Buckets using cut or value bins
    dataset['AgeBin'] = pd.cut(dataset['Age'].astype(int), 5)
```

```
#cleanup rare title names
#print(data1['Title'].value_counts())
stat_min = 10 #while small is arbitrary, we'll use the common minimum in statistics
title_names = (data1['Title'].value_counts() < stat_min) #this will create a true false series with title name as index

#apply and lambda functions are quick and dirty code to find and replace with fewer lines of code
data1['Title'] = data1['Title'].apply(lambda x: 'Misc' if title_names.loc[x] == True else x)
print(data1['Title'].value_counts())
print("-"*10)


#preview data again
data1.info()
data_val.info()
data1.sample(10)
```

```
Mr        517
Miss      182
Mrs       125
Master     40
Misc       27
Name: Title, dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
Survived      891 non-null int64
Pclass        891 non-null int64
Name          891 non-null object
Sex           891 non-null object
Age           891 non-null float64
SibSp         891 non-null int64
Parch         891 non-null int64
Fare          891 non-null float64
Embarked      891 non-null object
FamilySize    891 non-null int64
IsAlone       891 non-null int64
Title         891 non-null object
FareBin       891 non-null category
AgeBin        891 non-null category
dtypes: category(2), float64(2), int64(6), object(4)
memory usage: 85.5+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 16 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            418 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           418 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
FamilySize     418 non-null int64
IsAlone        418 non-null int64
Title          418 non-null object
FareBin        418 non-null category
AgeBin         418 non-null category
```

```
dtypes: category(2), float64(2), int64(6), object(6)
memory usage: 46.8+ KB
```

Out[6]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	IsAlone	Title	FareBin	AgeBin
642	0	3	Skoog, Miss. Margit Elizabeth	female	2.0	3	2	27.9000	S	6	0	Miss	(14.454, 31.0]	(-0.08, 16.0]
489	1	3	Coutts, Master. Eden Leslie "Neville"	male	9.0	1	1	15.9000	S	3	0	Master	(14.454, 31.0]	(-0.08, 16.0]
848	0	2	Harper, Rev. John	male	28.0	0	1	33.0000	S	2	0	Misc	(31.0, 512.329]	(16.0, 32.0]
688	0	3	Fischer, Mr. Eberhard Thelander	male	18.0	0	0	7.7958	S	1	1	Mr	(-0.001, 7.91]	(16.0, 32.0]
781	1	1	Dick, Mrs. Albert Adrian (Vera Gillespie)	female	17.0	1	0	57.0000	S	2	0	Mrs	(31.0, 512.329]	(16.0, 32.0]
133	1	2	Weisz, Mrs. Leopold (Mathilde Francoise Pede)	female	29.0	1	0	26.0000	S	2	0	Mrs	(14.454, 31.0]	(16.0, 32.0]
822	0	1	Reuchlin, Jonkheer. John George	male	38.0	0	0	0.0000	S	1	1	Misc	(-0.001, 7.91]	(32.0, 48.0]
303	1	2	Keane, Miss. Nora A	female	28.0	0	0	12.3500	Q	1	1	Miss	(7.91, 14.454]	(16.0, 32.0]
874	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	24.0000	C	2	0	Mrs	(14.454, 31.0]	(16.0, 32.0]
386	0	3	Goodwin, Master. Sidney Leonard	male	1.0	5	2	46.9000	S	8	0	Master	(31.0, 512.329]	(-0.08, 16.0]

3.23 Convert Formats

We will convert categorical data to dummy variables for mathematical analysis. There are multiple ways to encode categorical variables; we will use the sklearn and pandas functions.

In this step, we will also define our x (independent/features/explanatory/predictor/etc.) and y (dependent/target/outcome/response/etc.) variables for data modeling.

In [7]:
```
#CONVERT: convert objects to category using Label Encoder for train and test/validation dataset

#code categorical data
label = LabelEncoder()
for dataset in data_cleaner:
    dataset['Sex_Code'] = label.fit_transform(dataset['Sex'])
    dataset['Embarked_Code'] = label.fit_transform(dataset['Embarked'])
    dataset['Title_Code'] = label.fit_transform(dataset['Title'])
    dataset['AgeBin_Code'] = label.fit_transform(dataset['AgeBin'])
    dataset['FareBin_Code'] = label.fit_transform(dataset['FareBin'])


#define y variable aka target/outcome
Target = ['Survived']

#define x variables for original features aka feature selection
data1_x = ['Sex','Pclass', 'Embarked', 'Title','SibSp', 'Parch', 'Age', 'Fare', 'FamilySize', 'IsAlone']
data1_x_calc = ['Sex_Code','Pclass', 'Embarked_Code', 'Title_Code','SibSp', 'Parch', 'Age', 'Fare']
data1_xy =  Target + data1_x
print('Original X Y: ', data1_xy, '\n')


#define x variables for original w/bin features to remove continuous variables
data1_x_bin = ['Sex_Code','Pclass', 'Embarked_Code', 'Title_Code', 'FamilySize', 'AgeBin_Code', 'FareBin_Code']
data1_xy_bin = Target + data1_x_bin
print('Bin X Y: ', data1_xy_bin, '\n')


#define x and y variables for dummy features original
data1_dummy = pd.get_dummies(data1[data1_x])
data1_x_dummy = data1_dummy.columns.tolist()
data1_xy_dummy = Target + data1_x_dummy
print('Dummy X Y: ', data1_xy_dummy, '\n')


data1_dummy.head()
```

```
Original X Y:  ['Survived', 'Sex', 'Pclass', 'Embarked', 'Title', 'SibSp', 'Parch', 'Age', 'Fare', 'FamilySize', 'IsAlo
ne']

Bin X Y:  ['Survived', 'Sex_Code', 'Pclass', 'Embarked_Code', 'Title_Code', 'FamilySize', 'AgeBin_Code', 'FareBin_Cod
e']

Dummy X Y:  ['Survived', 'Pclass', 'SibSp', 'Parch', 'Age', 'Fare', 'FamilySize', 'IsAlone', 'Sex_female', 'Sex_male',
'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Title_Master', 'Title_Misc', 'Title_Miss', 'Title_Mr', 'Title_Mrs']
```

Out[7]:

	Pclass	SibSp	Parch	Age	Fare	FamilySize	IsAlone	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Title_Master	Title
0	3	1	0	22.0	7.2500	2	0	0	1	0	0	1	0	
1	1	1	0	38.0	71.2833	2	0	1	0	1	0	0	0	
2	3	0	0	26.0	7.9250	1	1	1	0	0	0	1	0	
3	1	1	0	35.0	53.1000	2	0	1	0	0	0	1	0	
4	3	0	0	35.0	8.0500	1	1	0	1	0	0	1	0	

3.24 Double Check Cleaned Data

In [8]:

```
print('Train columns with null values: \n', data1.isnull().sum())
print("-"*10)
print (data1.info())
print("-"*10)

print('Test/Validation columns with null values: \n', data_val.isnull().sum())
print("-"*10)
print (data_val.info())
print("-"*10)

data_raw.describe(include = 'all')
```

```
Train columns with null values:
 Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Fare             0
Embarked         0
FamilySize       0
IsAlone          0
Title            0
FareBin          0
AgeBin           0
Sex_Code         0
Embarked_Code    0
Title_Code       0
AgeBin_Code      0
FareBin_Code     0
dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 19 columns):
Survived         891 non-null int64
Pclass           891 non-null int64
Name             891 non-null object
Sex              891 non-null object
Age              891 non-null float64
SibSp            891 non-null int64
Parch            891 non-null int64
Fare             891 non-null float64
Embarked         891 non-null object
FamilySize       891 non-null int64
IsAlone          891 non-null int64
Title            891 non-null object
FareBin          891 non-null category
AgeBin           891 non-null category
Sex_Code         891 non-null int64
Embarked_Code    891 non-null int64
Title_Code       891 non-null int64
AgeBin_Code      891 non-null int64
FareBin_Code     891 non-null int64
dtypes: category(2), float64(2), int64(11), object(4)
```

```
memory usage: 120.3+ KB
None
----------
Test/Validation columns with null values:
 PassengerId        0
Pclass              0
Name                0
Sex                 0
Age                 0
SibSp               0
Parch               0
Ticket              0
Fare                0
Cabin             327
Embarked            0
FamilySize          0
IsAlone             0
Title               0
FareBin             0
AgeBin              0
Sex_Code            0
Embarked_Code       0
Title_Code          0
AgeBin_Code         0
FareBin_Code        0
dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 21 columns):
PassengerId      418 non-null int64
Pclass           418 non-null int64
Name             418 non-null object
Sex              418 non-null object
Age              418 non-null float64
SibSp            418 non-null int64
Parch            418 non-null int64
Ticket           418 non-null object
Fare             418 non-null float64
Cabin            91 non-null object
Embarked         418 non-null object
FamilySize       418 non-null int64
IsAlone          418 non-null int64
Title            418 non-null object
FareBin          418 non-null category
```

```
AgeBin           418 non-null category
Sex_Code         418 non-null int64
Embarked_Code    418 non-null int64
Title_Code       418 non-null int64
AgeBin_Code      418 non-null int64
FareBin_Code     418 non-null int64
dtypes: category(2), float64(2), int64(11), object(6)
memory usage: 63.1+ KB
None
----------
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Kilgannon, Mr. Thomas J	male	NaN	NaN	NaN	347082	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

3.25 Split Training and Testing Data

In [9]:

```
#split train and test data with function defaults
#random_state -> seed or control random number generator
train1_x, test1_x, train1_y, test1_y = model_selection.train_test_split(data1[data1_x_calc], data1[Target], random_stat
train1_x_bin, test1_x_bin, train1_y_bin, test1_y_bin = model_selection.train_test_split(data1[data1_x_bin], data1[Targe
train1_x_dummy, test1_x_dummy, train1_y_dummy, test1_y_dummy = model_selection.train_test_split(data1_dummy[data1_x_dum
```

```
print("Data1 Shape: {}".format(data1.shape))
print("Train1 Shape: {}".format(train1_x.shape))
print("Test1 Shape: {}".format(test1_x.shape))

train1_x_bin.head()
```

```
Data1 Shape: (891, 19)
Train1 Shape: (668, 8)
Test1 Shape: (223, 8)
```

Out[9]:

	Sex_Code	Pclass	Embarked_Code	Title_Code	FamilySize	AgeBin_Code	FareBin_Code
105	1	3	2	3	1	1	0
68	0	3	2	2	7	1	1
253	1	3	2	3	2	1	2
320	1	3	2	3	1	1	0
706	0	2	2	4	1	2	1

Step 4: Perform Exploratory Data Analysis with Statistics

Now that our data is cleaned, we will explore our data with descriptive and graphical statistics to describe and summarize our variables. In this stage, classifying features and determining their correlation with the target variable and each other.

In [10]:

```
#Discrete Variable Correlation by Survival using
#group by aka pivot table
for x in data1_x:
    if data1[x].dtype != 'float64' :
        print('Survival Correlation by:', x)
        print(data1[[x, Target[0]]].groupby(x, as_index=False).mean())
        print('-'*10, '\n')


#using crosstabs
print(pd.crosstab(data1['Title'],data1[Target[0]]))
```

```
Survival Correlation by: Sex
      Sex  Survived
0  female  0.742038
1    male  0.188908
----------

Survival Correlation by: Pclass
   Pclass  Survived
0       1  0.629630
1       2  0.472826
2       3  0.242363
----------

Survival Correlation by: Embarked
  Embarked  Survived
0        C  0.553571
1        Q  0.389610
2        S  0.339009
----------

Survival Correlation by: Title
    Title  Survived
0  Master  0.575000
1    Misc  0.444444
2    Miss  0.697802
3      Mr  0.156673
4     Mrs  0.792000
----------

Survival Correlation by: SibSp
   SibSp  Survived
0      0  0.345395
1      1  0.535885
2      2  0.464286
3      3  0.250000
4      4  0.166667
5      5  0.000000
6      8  0.000000
----------

Survival Correlation by: Parch
   Parch  Survived
0      0  0.343658
1      1  0.550847
2      2  0.500000
```

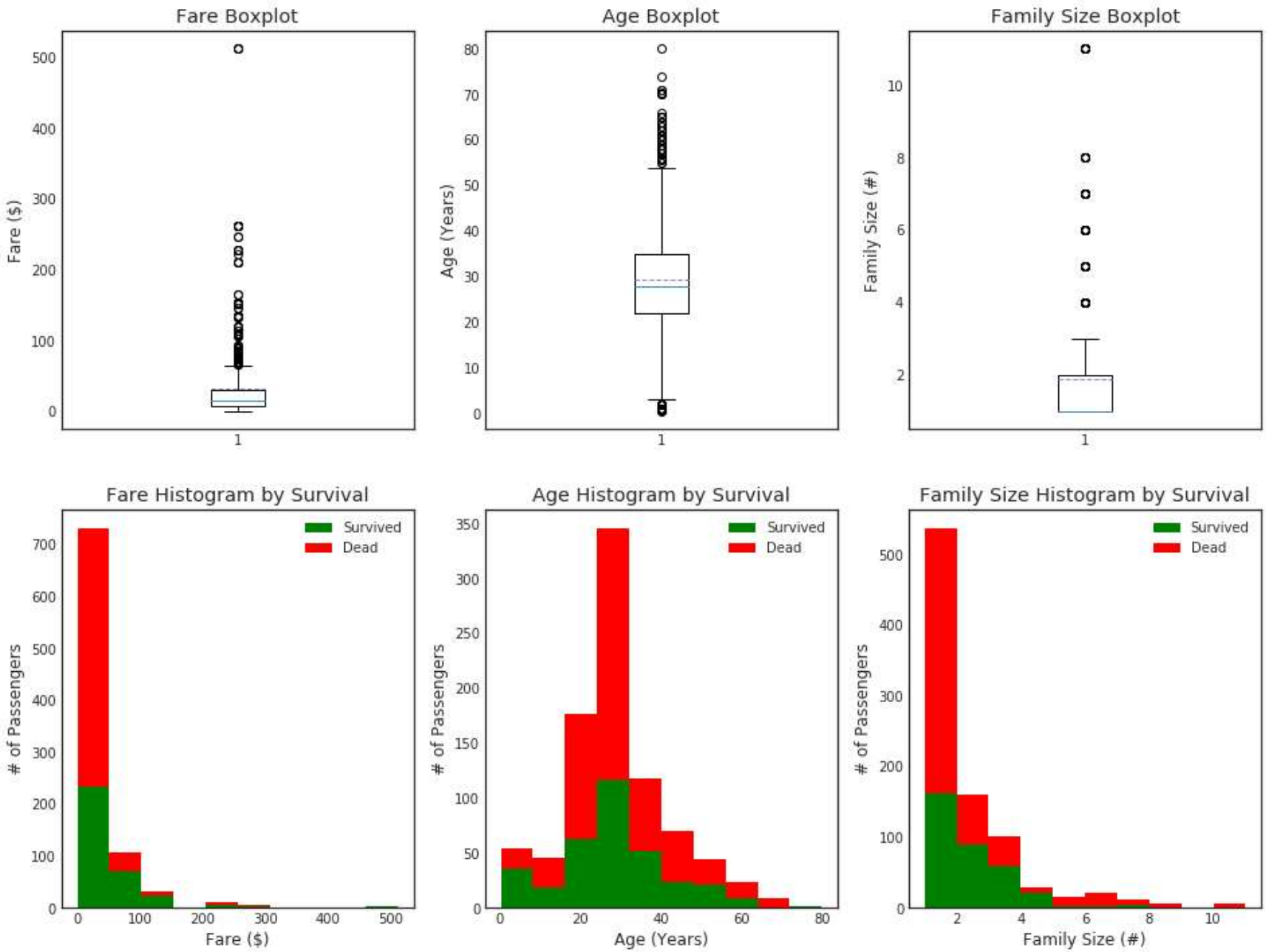
```
3      3  0.600000
4      4  0.000000
5      5  0.200000
6      6  0.000000
----------

Survival Correlation by: FamilySize
   FamilySize  Survived
0           1  0.303538
1           2  0.552795
2           3  0.578431
3           4  0.724138
4           5  0.200000
5           6  0.136364
6           7  0.333333
7           8  0.000000
8          11  0.000000
----------

Survival Correlation by: IsAlone
   IsAlone  Survived
0        0  0.505650
1        1  0.303538
----------

Survived    0    1
Title
Master     17   23
Misc       15   12
Miss       55  127
Mr        436   81
Mrs        26   99
```

In [11]:

```python
#graph distribution of quantitative data
plt.figure(figsize=[16,12])

plt.subplot(231)
plt.boxplot(x=data1['Fare'], showmeans = True, meanline = True)
plt.title('Fare Boxplot')
plt.ylabel('Fare ($)')

plt.subplot(232)
plt.boxplot(data1['Age'], showmeans = True, meanline = True)
plt.title('Age Boxplot')
plt.ylabel('Age (Years)')
```

```
plt.subplot(233)
plt.boxplot(data1['FamilySize'], showmeans = True, meanline = True)
plt.title('Family Size Boxplot')
plt.ylabel('Family Size (#)')

plt.subplot(234)
plt.hist(x = [data1[data1['Survived']==1]['Fare'], data1[data1['Survived']==0]['Fare']],
         stacked=True, color = ['g','r'],label = ['Survived','Dead'])
plt.title('Fare Histogram by Survival')
plt.xlabel('Fare ($)')
plt.ylabel('# of Passengers')
plt.legend()

plt.subplot(235)
plt.hist(x = [data1[data1['Survived']==1]['Age'], data1[data1['Survived']==0]['Age']],
         stacked=True, color = ['g','r'],label = ['Survived','Dead'])
plt.title('Age Histogram by Survival')
plt.xlabel('Age (Years)')
plt.ylabel('# of Passengers')
plt.legend()

plt.subplot(236)
plt.hist(x = [data1[data1['Survived']==1]['FamilySize'], data1[data1['Survived']==0]['FamilySize']],
         stacked=True, color = ['g','r'],label = ['Survived','Dead'])
plt.title('Family Size Histogram by Survival')
plt.xlabel('Family Size (#)')
plt.ylabel('# of Passengers')
plt.legend()
```

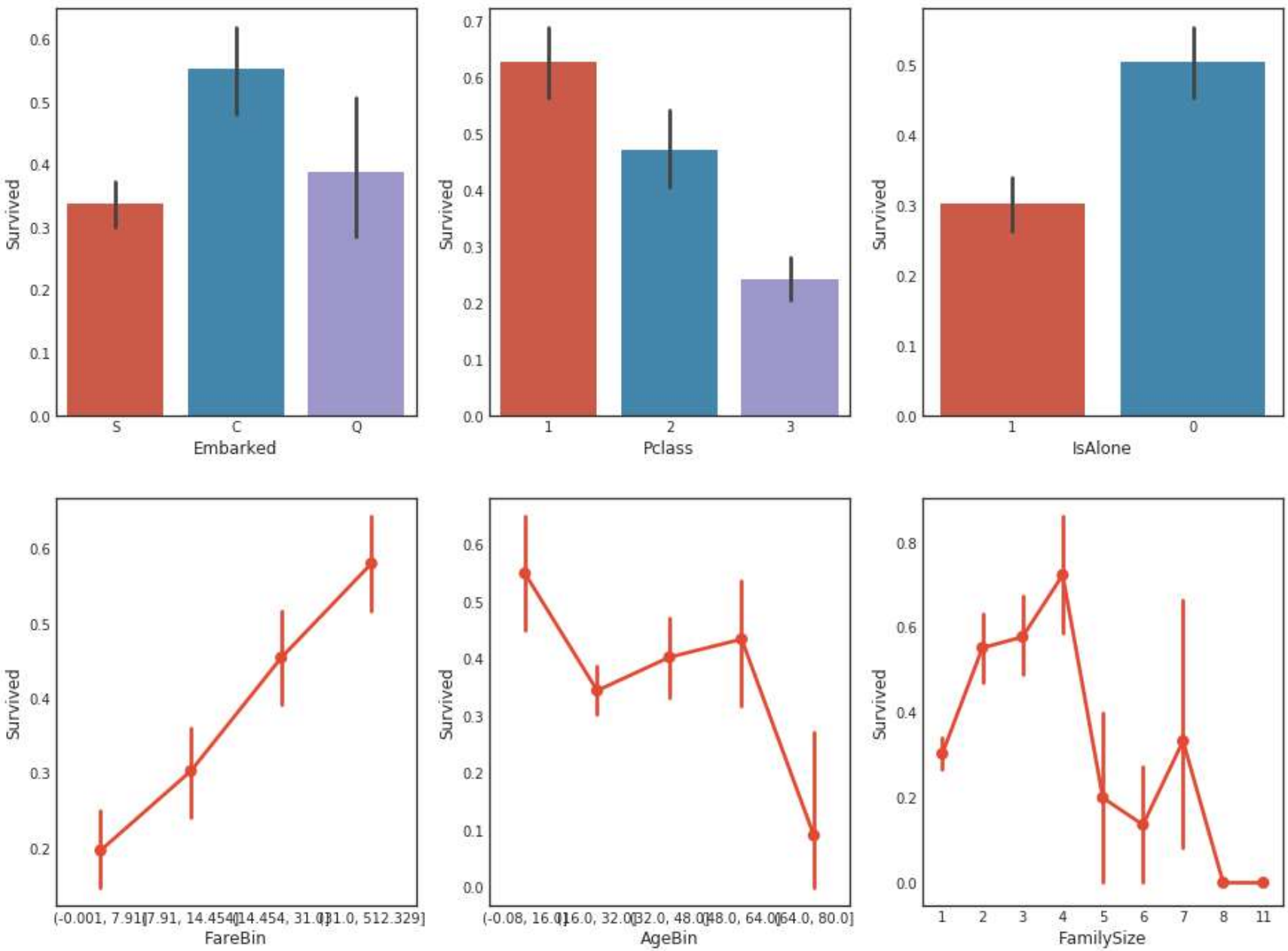
Out[11]:
```
<matplotlib.legend.Legend at 0x7b27c677c0f0>
```



In [12]:
```
#we will use seaborn graphics for multi-variable comparison

#graph individual features by survival
fig, saxis = plt.subplots(2, 3,figsize=(16,12))
```

```
sns.barplot(x = 'Embarked', y = 'Survived', data=data1, ax = saxis[0,0])
sns.barplot(x = 'Pclass', y = 'Survived', order=[1,2,3], data=data1, ax = saxis[0,1])
sns.barplot(x = 'IsAlone', y = 'Survived', order=[1,0], data=data1, ax = saxis[0,2])

sns.pointplot(x = 'FareBin', y = 'Survived',  data=data1, ax = saxis[1,0])
sns.pointplot(x = 'AgeBin', y = 'Survived',  data=data1, ax = saxis[1,1])
sns.pointplot(x = 'FamilySize', y = 'Survived', data=data1, ax = saxis[1,2])
```

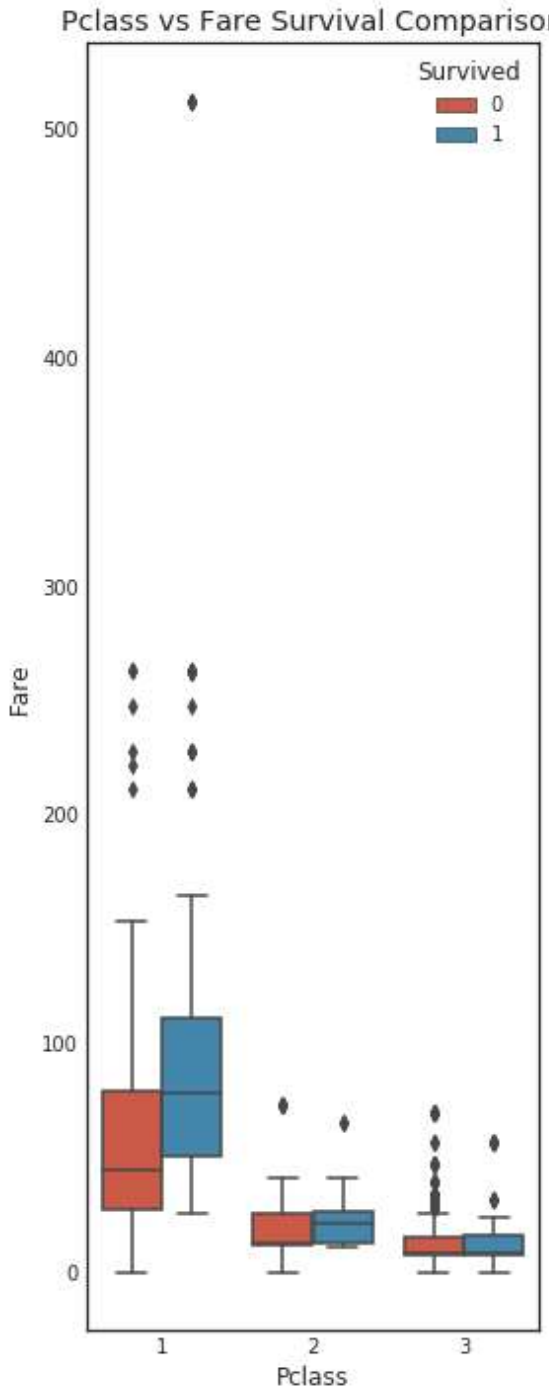
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7b27c64f96a0>

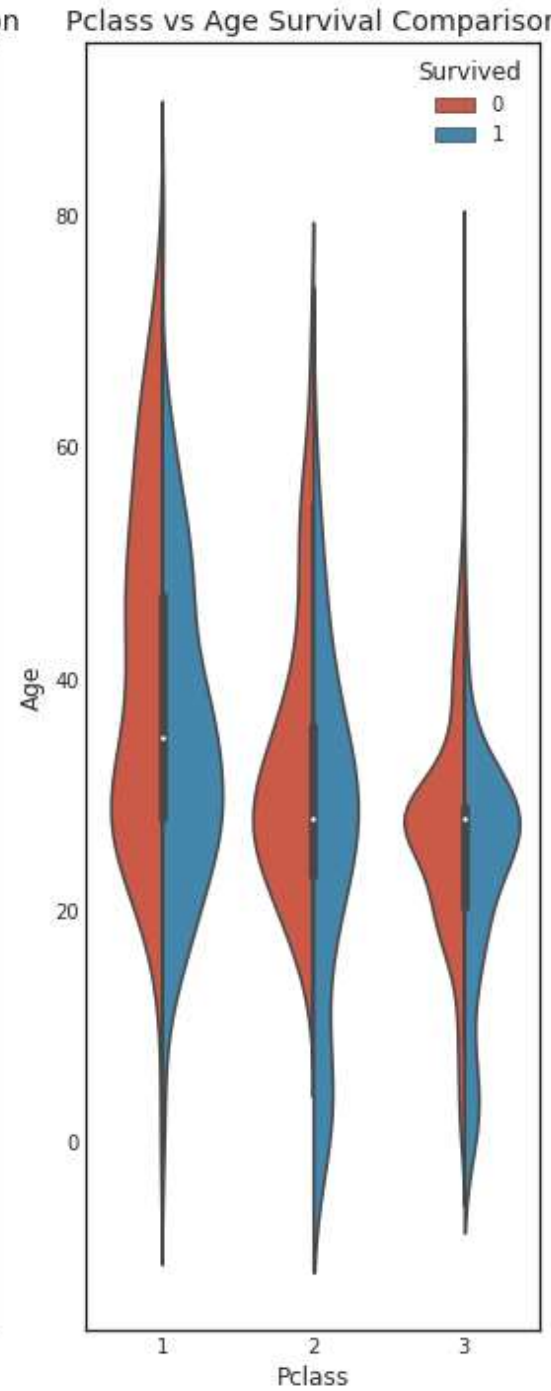


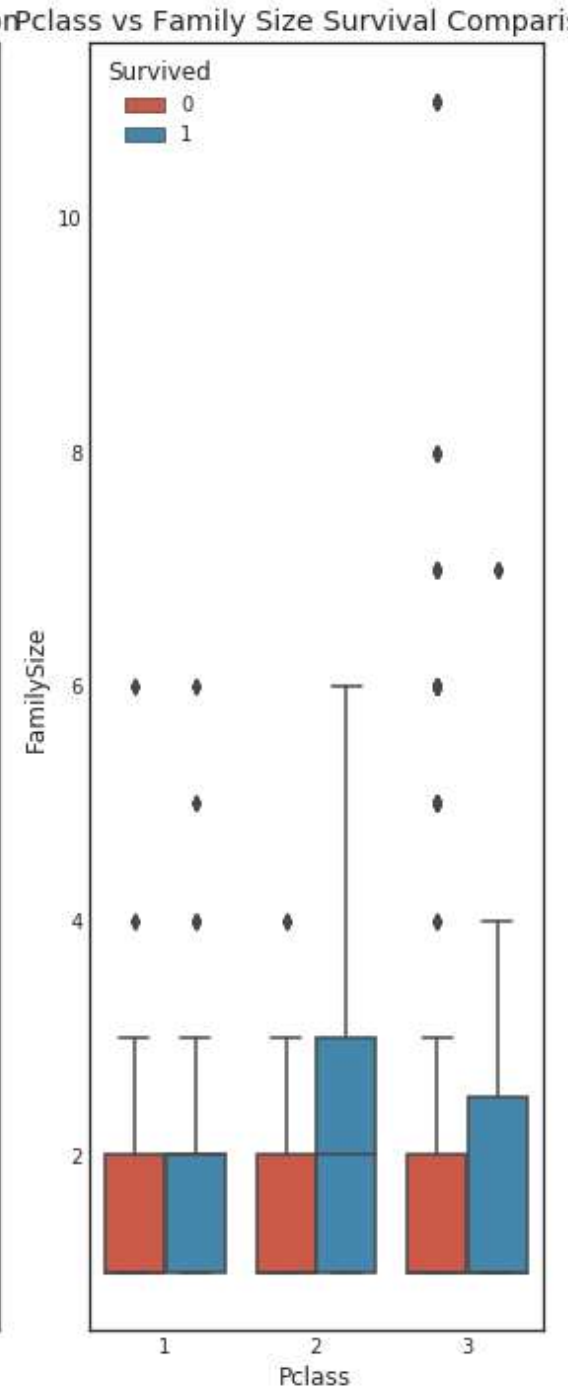
In [13]:
```
#graph distribution of qualitative data: Pclass
#we know class mattered in survival, now let's compare class and a 2nd feature
fig, (axis1,axis2,axis3) = plt.subplots(1,3,figsize=(14,12))

sns.boxplot(x = 'Pclass', y = 'Fare', hue = 'Survived', data = data1, ax = axis1)
```

```
axis1.set_title('Pclass vs Fare Survival Comparison')

sns.violinplot(x = 'Pclass', y = 'Age', hue = 'Survived', data = data1, split = True, ax = axis2)
axis2.set_title('Pclass vs Age Survival Comparison')

sns.boxplot(x = 'Pclass', y ='FamilySize', hue = 'Survived', data = data1, ax = axis3)
axis3.set_title('Pclass vs Family Size Survival Comparison')
```

Out[13]:
```
Text(0.5,1,'Pclass vs Family Size Survival Comparison')
```

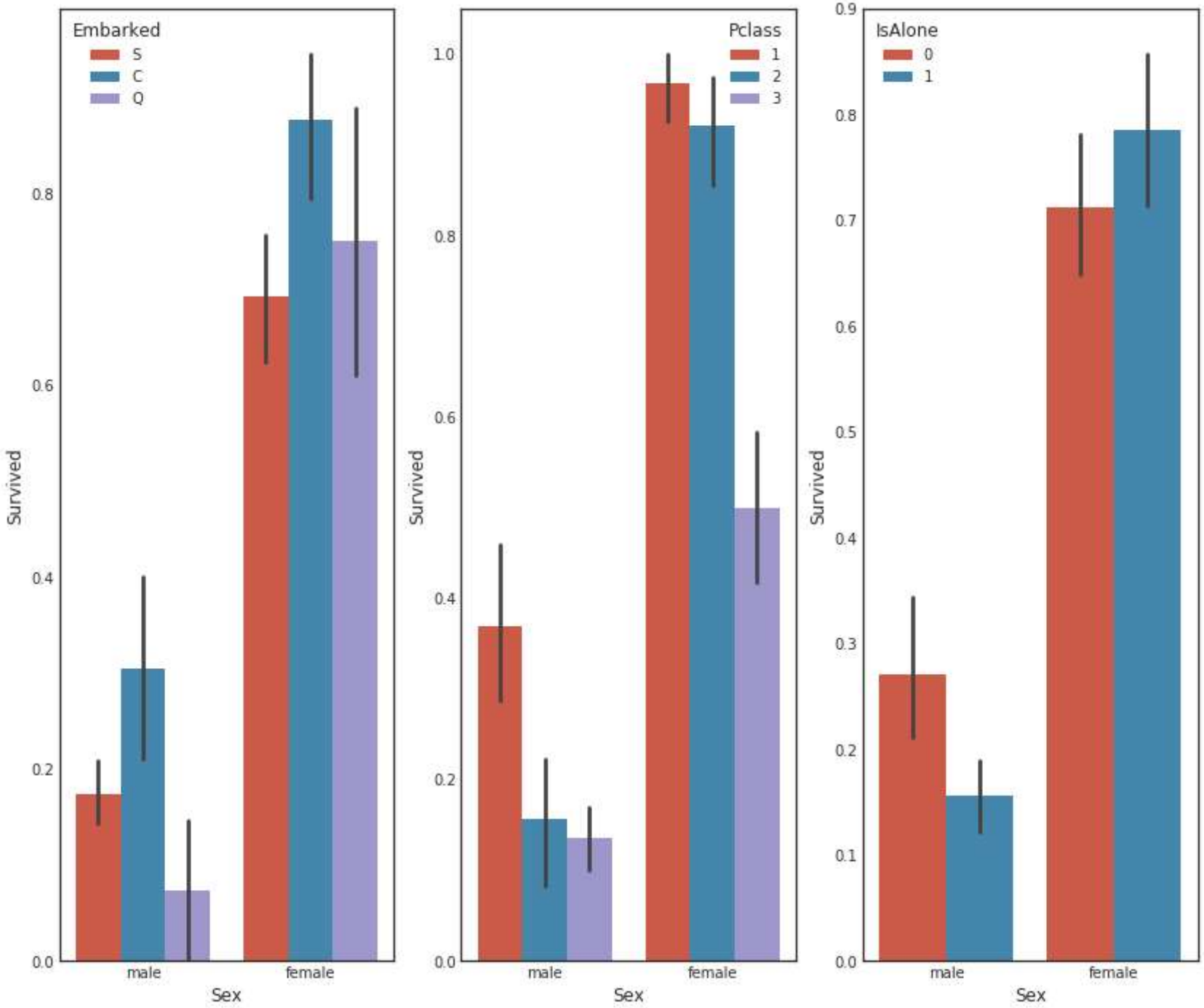

Pclass vs Fare Survival Comparison
Survived
0
1
Fare
500
400
300
200
100
0
1
2
3
Pclass


Pclass vs Age Survival Comparison
Survived
0
1
Age
80
60
40
20
0
1
2
3
Pclass


Pclass vs Family Size Survival Comparison
Survived
0
1
FamilySize
10
8
6
4
2
1
2
3
Pclass

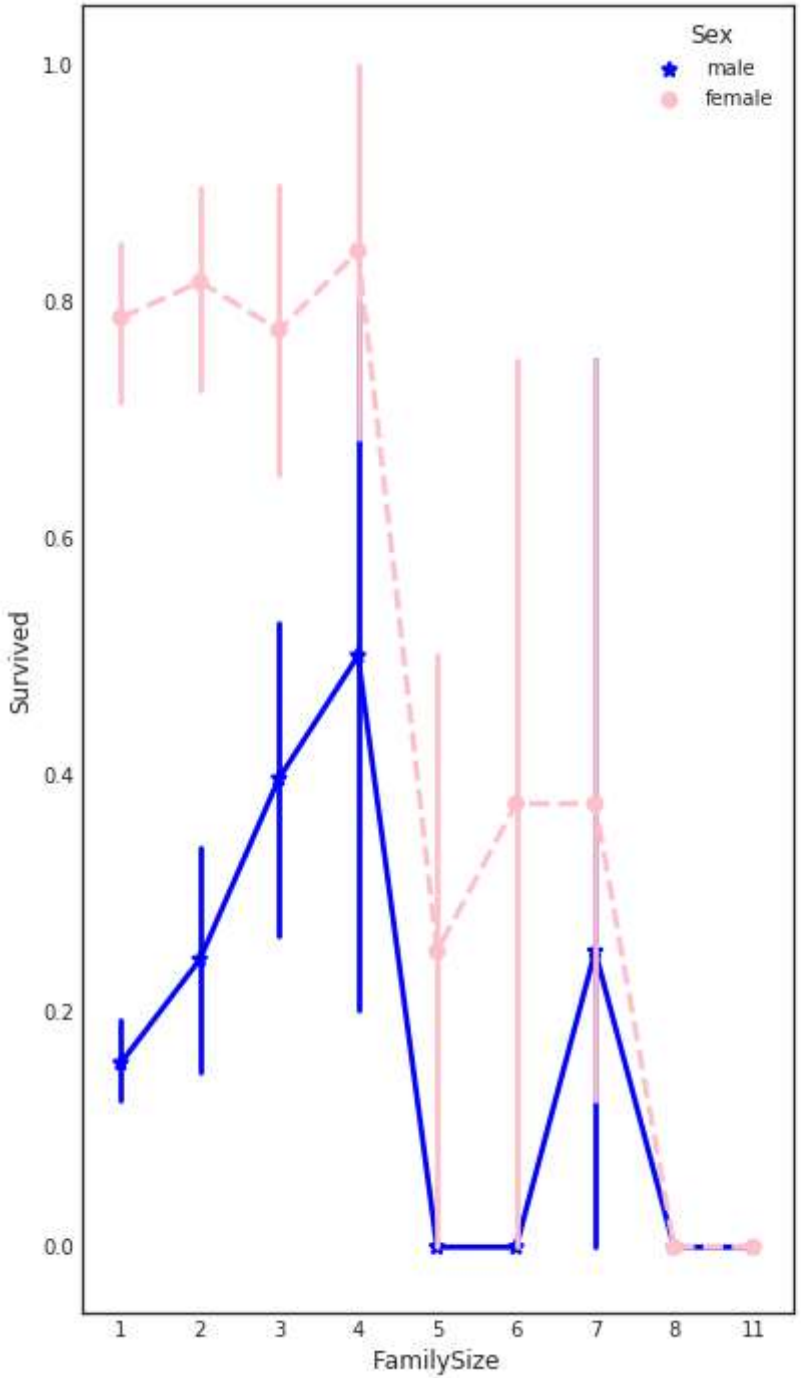
In [14]:
```
#graph distribution of qualitative data: Sex
#we know sex mattered in survival, now let's compare sex and a 2nd feature
fig, qaxis = plt.subplots(1,3,figsize=(14,12))

sns.barplot(x = 'Sex', y = 'Survived', hue = 'Embarked', data=data1, ax = qaxis[0])
axis1.set_title('Sex vs Embarked Survival Comparison')

sns.barplot(x = 'Sex', y = 'Survived', hue = 'Pclass', data=data1, ax  = qaxis[1])
axis1.set_title('Sex vs Pclass Survival Comparison')

sns.barplot(x = 'Sex', y = 'Survived', hue = 'IsAlone', data=data1, ax  = qaxis[2])
axis1.set_title('Sex vs IsAlone Survival Comparison')
```

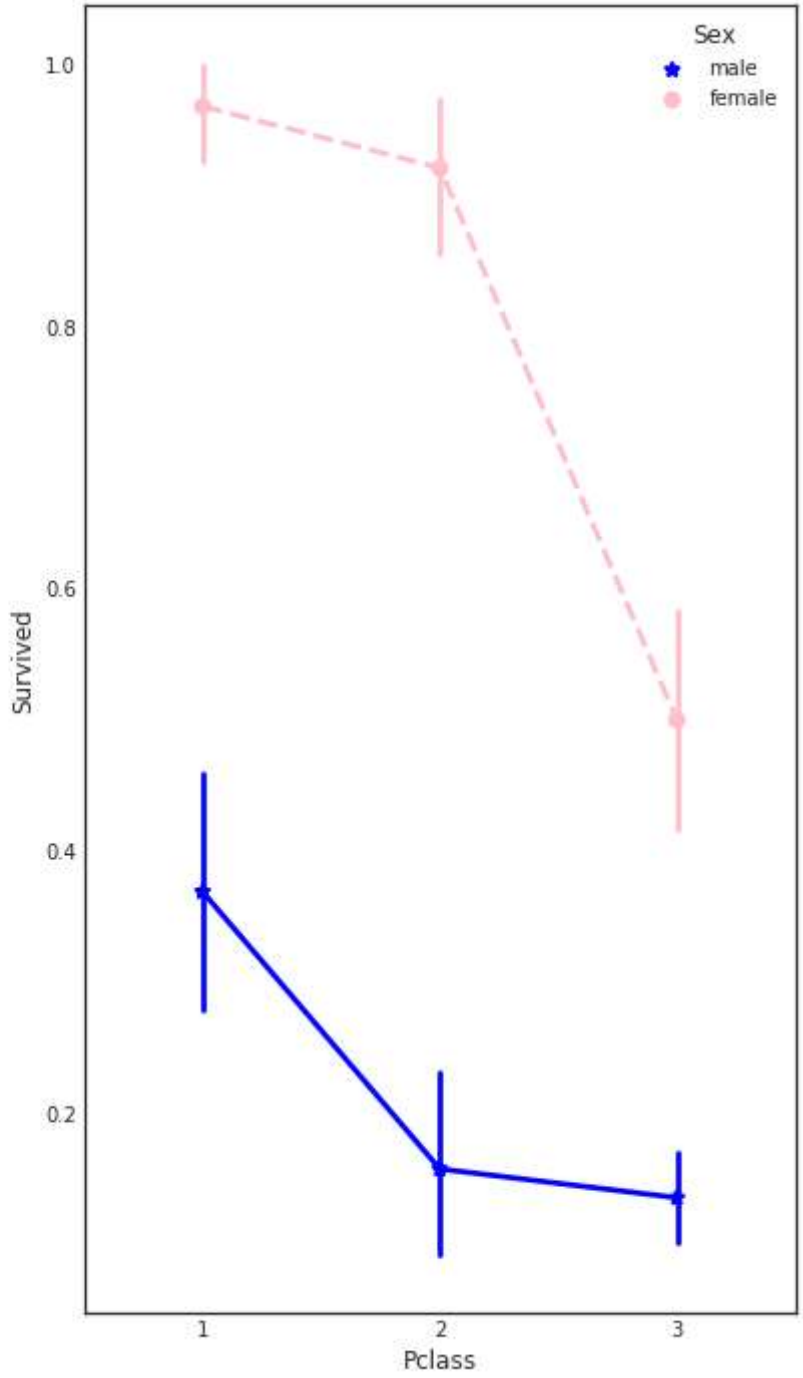
Out[14]:
```
Text(0.5,1,'Sex vs IsAlone Survival Comparison')
```


Embarked
S
C
Q
Survived
Sex
male
female
0.0
0.2
0.4
0.6
0.8
Pclass
1
2
3
1.0
IsAlone
0
1
0.1
0.3
0.5
0.7
0.9

In [15]:
```
#more side-by-side comparisons
fig, (maxis1, maxis2) = plt.subplots(1, 2,figsize=(14,12))

#how does family size factor with sex & survival compare
sns.pointplot(x="FamilySize", y="Survived", hue="Sex", data=data1,
              palette={"male": "blue", "female": "pink"},
              markers=["*", "o"], linestyles=["-", "--"], ax = maxis1)

#how does class factor with sex & survival compare
sns.pointplot(x="Pclass", y="Survived", hue="Sex", data=data1,
              palette={"male": "blue", "female": "pink"},
              markers=["*", "o"], linestyles=["-", "--"], ax = maxis2)
```

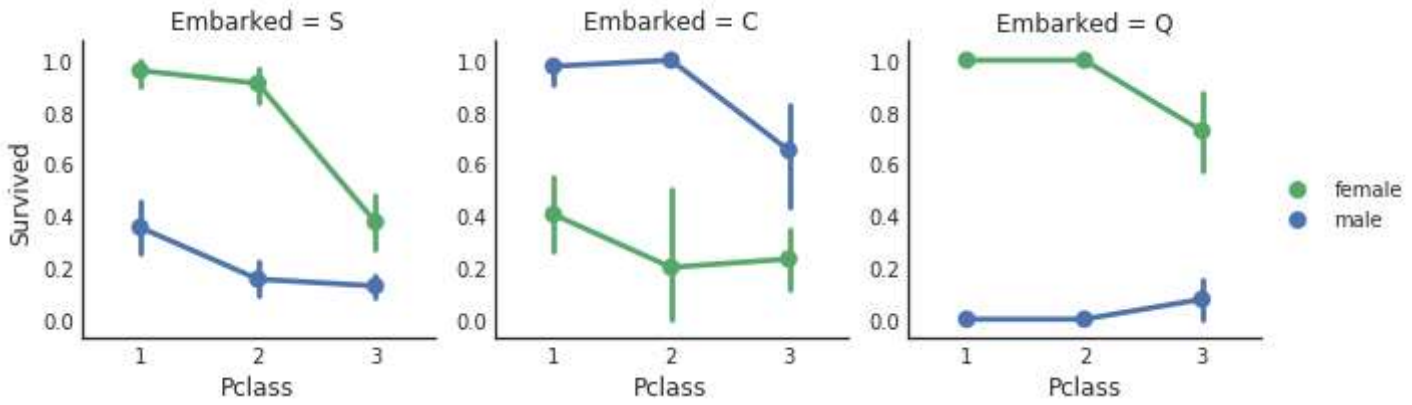
Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7b27c6082ac8>
```


Sex
male
female
Survived
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
7
8
11
FamilySize


Sex
male
female
Survived
1.0
0.8
0.6
0.4
0.2
1
2
3
Pclass

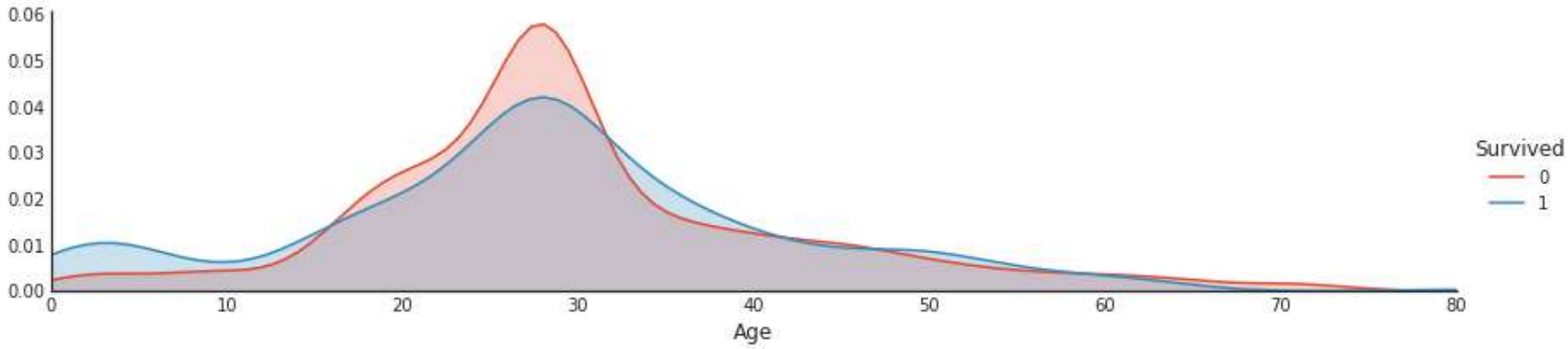
In [16]:
```
#how does embark port factor with class, sex, and survival compare
#facetgrid
e = sns.FacetGrid(data1, col = 'Embarked')
e.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', ci=95.0, palette = 'deep')
e.add_legend()
```

Out[16]:
```
<seaborn.axisgrid.FacetGrid at 0x7b27cdb7e2e8>
```



In [17]:
```
#plot distributions of age of passengers who survived or did not survive
a = sns.FacetGrid( data1, hue = 'Survived', aspect=4 )
a.map(sns.kdeplot, 'Age', shade= True )
a.set(xlim=(0 , data1['Age'].max()))
a.add_legend()
```

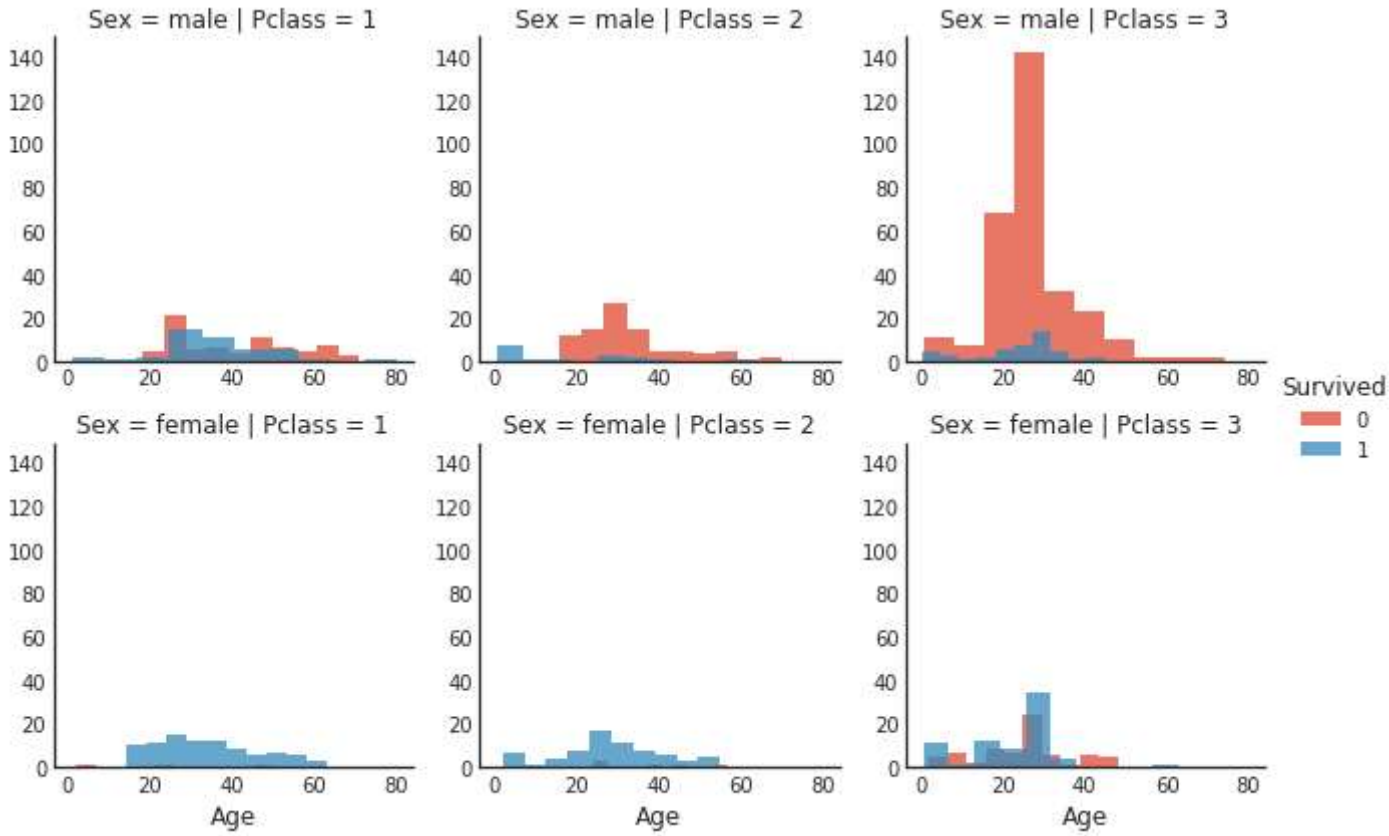
Out[17]:
```
<seaborn.axisgrid.FacetGrid at 0x7b27cdb7e748>
```



In [18]:
```
#histogram comparison of sex, class, and age by survival
h = sns.FacetGrid(data1, row = 'Sex', col = 'Pclass', hue = 'Survived')
```

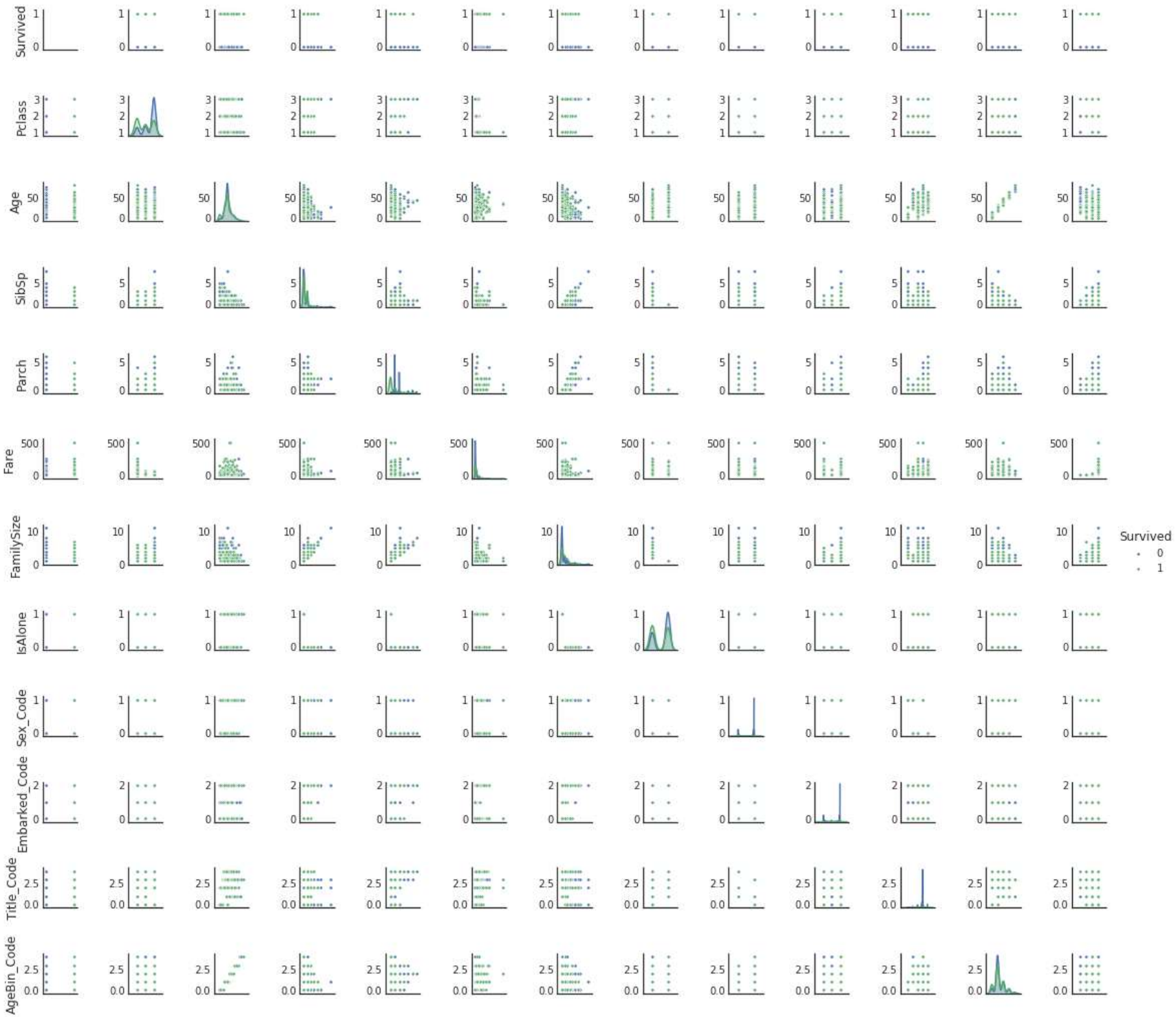
```
h.map(plt.hist, 'Age', alpha = .75)
h.add_legend()
```

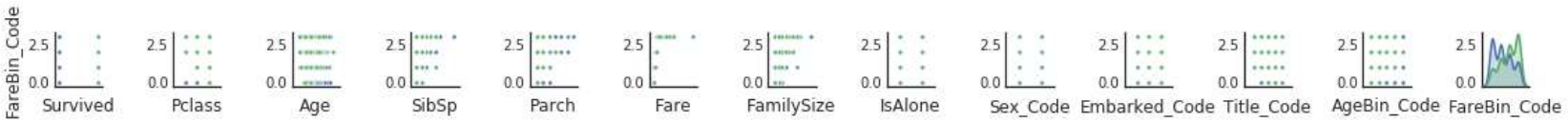
Out[18]: <seaborn.axisgrid.FacetGrid at 0x7b27c5e50588>



In [19]:
```
#pair plots of entire dataset
pp = sns.pairplot(data1, hue = 'Survived', palette = 'deep', size=1.2, diag_kind = 'kde', diag_kws=dict(shade=True), pl
pp.set(xticklabels=[])
```

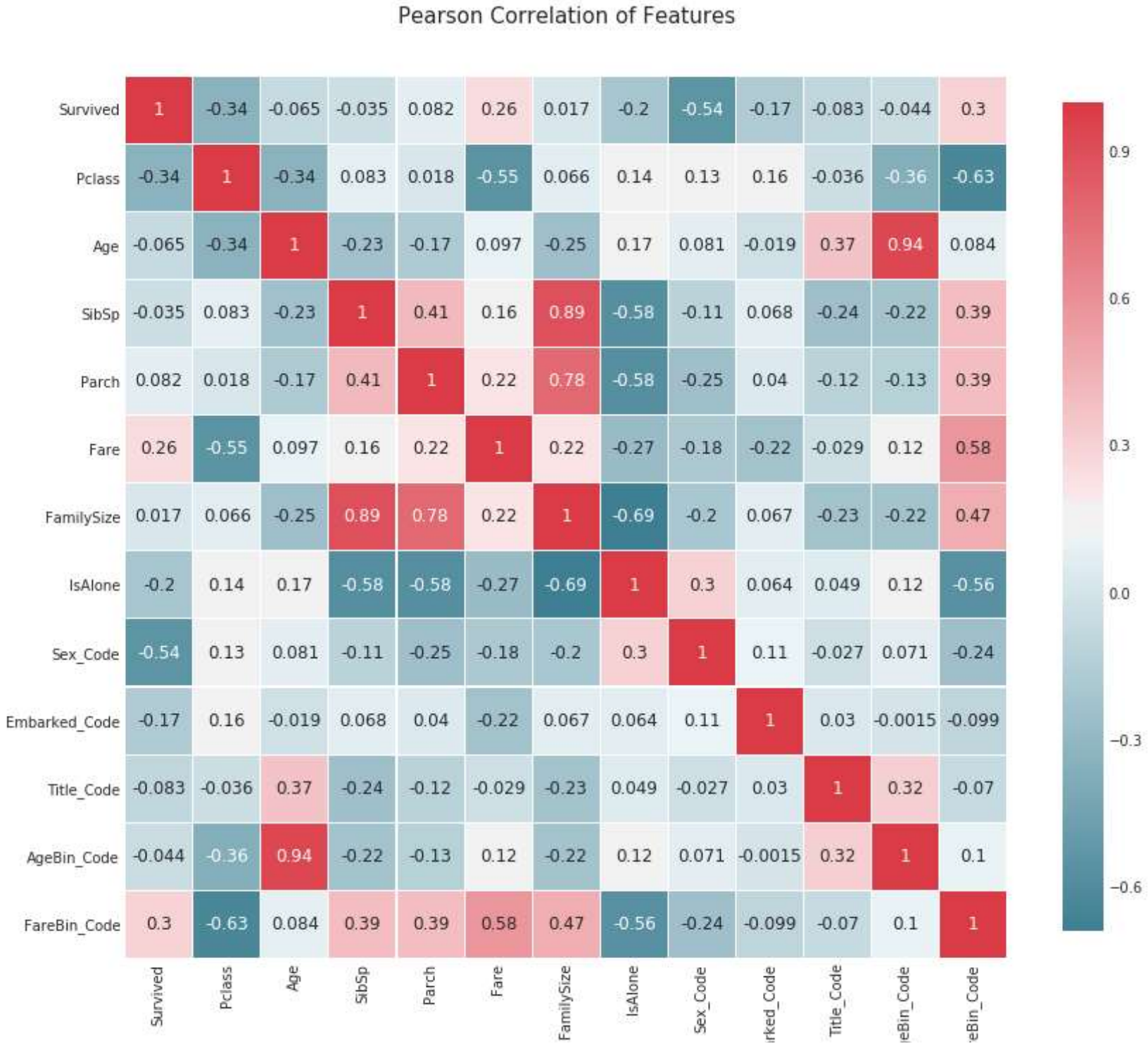
Out[19]: <seaborn.axisgrid.PairGrid at 0x7b27c5c3b630>


Survived
Pclass
Age
SibSp
Parch
Fare
FamilySize
IsAlone
Sex_Code
Embarked_Code
Title_Code
AgeBin_Code
Survived
0
1



In [20]:

```
#correlation heatmap of dataset
def correlation_heatmap(df):
    _ , ax = plt.subplots(figsize =(14, 12))
    colormap = sns.diverging_palette(220, 10, as_cmap = True)

    _ = sns.heatmap(
        df.corr(),
        cmap = colormap,
        square=True,
        cbar_kws={'shrink':.9 },
        ax=ax,
        annot=True,
        linewidths=0.1,vmax=1.0, linecolor='white',
        annot_kws={'fontsize':12 }
    )

    plt.title('Pearson Correlation of Features', y=1.05, size=15)

correlation_heatmap(data1)
```


Pearson Correlation of Features
Survived 1 -0.34 -0.065 -0.035 0.082 0.26 0.017 -0.2 -0.54 -0.17 -0.083 -0.044 0.3
Pclass -0.34 1 -0.34 0.083 0.018 -0.55 0.066 0.14 0.13 0.16 -0.036 -0.36 -0.63
Age -0.065 -0.34 1 -0.23 -0.17 0.097 -0.25 0.17 0.081 -0.019 0.37 0.94 0.084
SibSp -0.035 0.083 -0.23 1 0.41 0.16 0.89 -0.58 -0.11 0.068 -0.24 -0.22 0.39
Parch 0.082 0.018 -0.17 0.41 1 0.22 0.78 -0.58 -0.25 0.04 -0.12 -0.13 0.39
Fare 0.26 -0.55 0.097 0.16 0.22 1 0.22 -0.27 -0.18 -0.22 -0.029 0.12 0.58
FamilySize 0.017 0.066 -0.25 0.89 0.78 0.22 1 -0.69 -0.2 0.067 -0.23 -0.22 0.47
IsAlone -0.2 0.14 0.17 -0.58 -0.58 -0.27 -0.69 1 0.3 0.064 0.049 0.12 -0.56
Sex_Code -0.54 0.13 0.081 -0.11 -0.25 -0.18 -0.2 0.3 1 0.11 -0.027 0.071 -0.24
Embarked_Code -0.17 0.16 -0.019 0.068 0.04 -0.22 0.067 0.064 0.11 1 0.03 -0.0015 -0.099
Title_Code -0.083 -0.036 0.37 -0.24 -0.12 -0.029 -0.23 0.049 -0.027 0.03 1 0.32 -0.07
AgeBin_Code -0.044 -0.36 0.94 -0.22 -0.13 0.12 -0.22 0.12 0.071 -0.0015 0.32 1 0.1
FareBin_Code 0.3 -0.63 0.084 0.39 0.39 0.58 0.47 -0.56 -0.24 -0.099 -0.07 0.1 1
Survived
Pclass
Age
SibSp
Parch
Fare
FamilySize
IsAlone
Sex_Code
rked_Code
Title_Code
eBin_Code
eBin_Code
0.9
0.6
0.3
0.0
−0.3
−0.6

Emba Ag Far

Step 5: Data Modelling

In [21]:
```
#Machine Learning Algorithm (MLA) Selection and Initialization
MLA = [
    #Ensemble Methods
    ensemble.AdaBoostClassifier(),
    ensemble.BaggingClassifier(),
    ensemble.ExtraTreesClassifier(),
    ensemble.GradientBoostingClassifier(),
    ensemble.RandomForestClassifier(),

    #Gaussian Processes
    gaussian_process.GaussianProcessClassifier(),

    #GLM
    linear_model.LogisticRegressionCV(),
    linear_model.PassiveAggressiveClassifier(),
    linear_model.RidgeClassifierCV(),
    linear_model.SGDClassifier(),
    linear_model.Perceptron(),

    #Naives Bayes
    naive_bayes.BernoulliNB(),
    naive_bayes.GaussianNB(),

    #Nearest Neighbor
    neighbors.KNeighborsClassifier(),

    #SVM
    svm.SVC(probability=True),
    svm.NuSVC(probability=True),
    svm.LinearSVC(),

    #Trees
    tree.DecisionTreeClassifier(),
    tree.ExtraTreeClassifier(),

    #Discriminant Analysis
    discriminant_analysis.LinearDiscriminantAnalysis(),
```

```
    discriminant_analysis.QuadraticDiscriminantAnalysis(),


    #xgboost
    XGBClassifier()
    ]



#split dataset in cross-validation with this splitter class
#note: this is an alternative to train_test_split
cv_split = model_selection.ShuffleSplit(n_splits = 10, test_size = .3, train_size = .6, random_state = 0 ) # run model

#create table to compare MLA metrics
MLA_columns = ['MLA Name', 'MLA Parameters','MLA Train Accuracy Mean', 'MLA Test Accuracy Mean', 'MLA Test Accuracy 3*S
MLA_compare = pd.DataFrame(columns = MLA_columns)

#create table to compare MLA predictions
MLA_predict = data1[Target]

#index through MLA and save performance to table
row_index = 0
for alg in MLA:

    #set name and parameters
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index, 'MLA Name'] = MLA_name
    MLA_compare.loc[row_index, 'MLA Parameters'] = str(alg.get_params())

    #score model with cross validation
    cv_results = model_selection.cross_validate(alg, data1[data1_x_bin], data1[Target], cv  = cv_split)

    MLA_compare.loc[row_index, 'MLA Time'] = cv_results['fit_time'].mean()
    MLA_compare.loc[row_index, 'MLA Train Accuracy Mean'] = cv_results['train_score'].mean()
    MLA_compare.loc[row_index, 'MLA Test Accuracy Mean'] = cv_results['test_score'].mean()
    #if this is a non-bias random sample, then +/-3 standard deviations (std) from the mean, should statistically captur
    MLA_compare.loc[row_index, 'MLA Test Accuracy 3*STD'] = cv_results['test_score'].std()*3   #let's know the worst th


    #save MLA predictions - see section 6 for usage
    alg.fit(data1[data1_x_bin], data1[Target])
    MLA_predict[MLA_name] = alg.predict(data1[data1_x_bin])

    row_index+=1
```

```
#print and sort table
MLA_compare.sort_values(by = ['MLA Test Accuracy Mean'], ascending = False, inplace = True)
MLA_compare
#MLA_predict
```

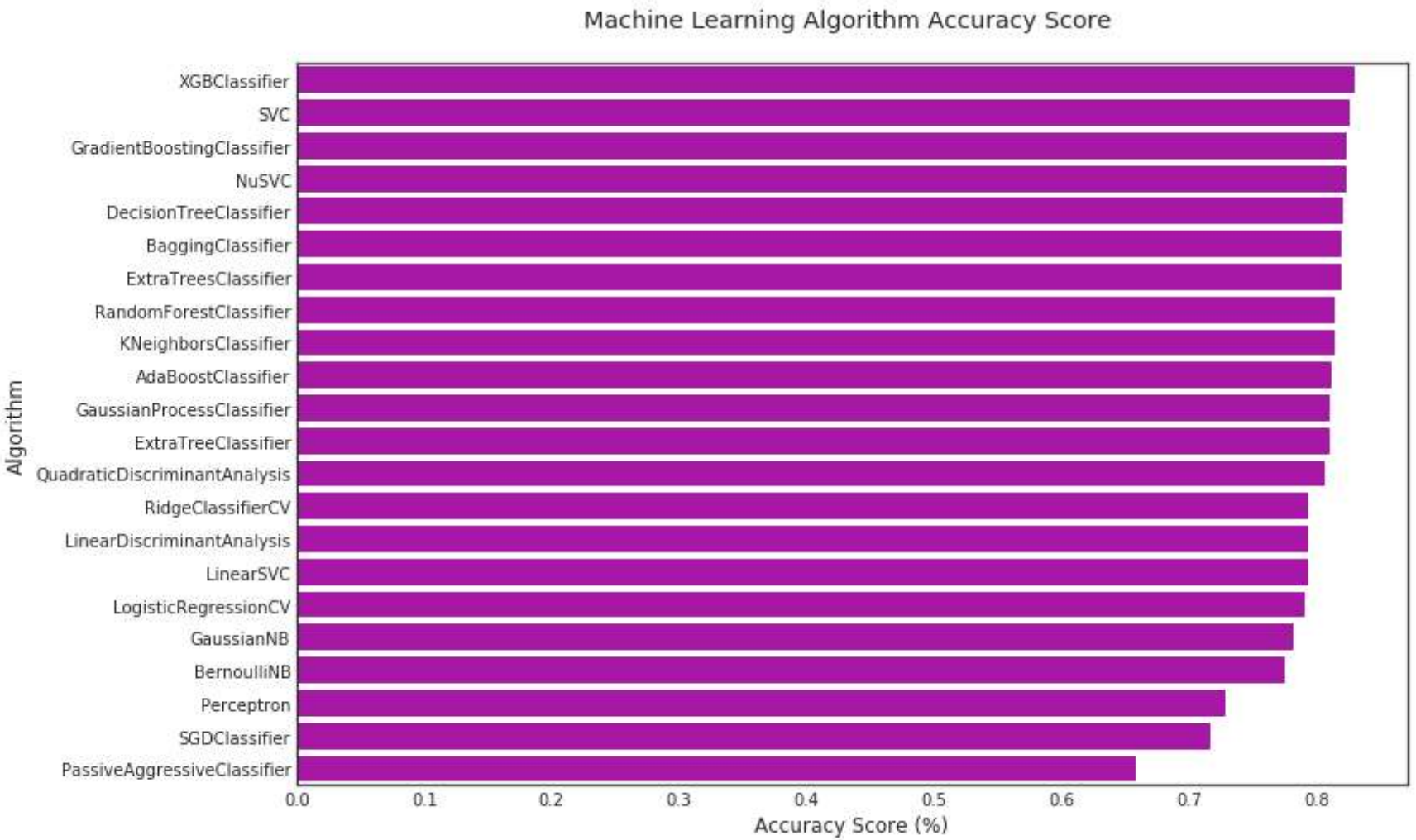
Out[21]:

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
21	XGBClassifier	{'base_score': 0.5, 'booster': 'gbtree', 'cols...	0.856367	0.829478	0.0527546	0.0182186
14	SVC	{'C': 1.0, 'cache_size': 200, 'class_weight': ...	0.837266	0.826119	0.0453876	0.0389952
3	GradientBoostingClassifier	{'criterion': 'friedman_mse', 'init': None, 'l...	0.866667	0.822761	0.0498731	0.0502059
15	NuSVC	{'cache_size': 200, 'class_weight': None, 'coe...	0.835768	0.822761	0.0493681	0.0475456
17	DecisionTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.895131	0.821269	0.0501737	0.00136428
1	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.8897	0.820149	0.0474394	0.01195
2	ExtraTreesClassifier	{'bootstrap': False, 'class_weight': None, 'cr...	0.895131	0.819403	0.0675734	0.0105573
4	RandomForestClassifier	{'bootstrap': True, 'class_weight': None, 'cri...	0.891948	0.814552	0.0690137	0.0112843
13	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.850375	0.813806	0.0690863	0.00132315
0	AdaBoostClassifier	{'algorithm': 'SAMME.R', 'base_estimator': Non...	0.820412	0.81194	0.0498606	0.0546636
5	GaussianProcessClassifier	{'copy_X_train': True, 'kernel': None, 'max_it...	0.871723	0.810448	0.0492537	0.389992
18	ExtraTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.895131	0.810448	0.0618818	0.00115268
20	QuadraticDiscriminantAnalysis	{'priors': None, 'reg_param': 0.0, 'store_cova...	0.821536	0.80709	0.0810389	0.00372462
8	RidgeClassifierCV	{'alphas': (0.1, 1.0, 10.0), 'class_weight': N...	0.796629	0.79403	0.0360302	0.00312555
19	LinearDiscriminantAnalysis	{'n_components': None, 'priors': None, 'shrink...	0.796816	0.79403	0.0360302	0.00179307

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
16	LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True,...	0.797378	0.79291	0.0410533	0.0243565
6	LogisticRegressionCV	{'Cs': 10, 'class_weight': None, 'cv': None, '...	0.797004	0.790672	0.0653582	0.083844
12	GaussianNB	{'priors': None}	0.794757	0.781343	0.0874568	0.0012383
11	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.785768	0.775373	0.0570347	0.00185876
10	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'eta0'...	0.740075	0.728731	0.162221	0.00125492
9	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.723596	0.717164	0.214318	0.00125179
7	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.656554	0.658209	0.430416	0.00134194

In [22]:
```
#barplot
sns.barplot(x='MLA Test Accuracy Mean', y = 'MLA Name', data = MLA_compare, color = 'm')

#prettify using pyplot
plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

Out[22]:
```
Text(0,0.5,'Algorithm')
```



5.1 Evaluate Model Performance

In [23]:
```
#iterate over dataFrame rows as (index, Series) pairs
for index, row in data1.iterrows():
    #random number generator
    if random.random() > .5:     # Random float x, 0.0 <= x < 1.0
        data1.set_value(index, 'Random_Predict', 1) #predict survived/1
    else:
        data1.set_value(index, 'Random_Predict', 0) #predict died/0
```

```
#score random guess of survival. Use shortcut 1 = Right Guess and 0 = Wrong Guess
#the mean of the column will then equal the accuracy
data1['Random_Score'] = 0 #assume prediction wrong
data1.loc[(data1['Survived'] == data1['Random_Predict']), 'Random_Score'] = 1 #set to 1 for correct prediction
print('Coin Flip Model Accuracy: {:.2f}%'.format(data1['Random_Score'].mean()*100))

#we can also use scikit's accuracy_score function to save us a few lines of code

print('Coin Flip Model Accuracy w/SciKit: {:.2f}%'.format(metrics.accuracy_score(data1['Survived'], data1['Random_Predi
```

```
Coin Flip Model Accuracy: 48.71%
Coin Flip Model Accuracy w/SciKit: 48.71%
```

In [24]:

```
#group by or pivot table
pivot_female = data1[data1.Sex=='female'].groupby(['Sex','Pclass', 'Embarked','FareBin'])['Survived'].mean()
print('Survival Decision Tree w/Female Node: \n',pivot_female)

pivot_male = data1[data1.Sex=='male'].groupby(['Sex','Title'])['Survived'].mean()
print('\n\nSurvival Decision Tree w/Male Node: \n',pivot_male)
```

```
Survival Decision Tree w/Female Node:
 Sex     Pclass  Embarked  FareBin
female  1       C         (14.454, 31.0]     0.666667
                          (31.0, 512.329]    1.000000
                Q         (31.0, 512.329]    1.000000
                S         (14.454, 31.0]     1.000000
                          (31.0, 512.329]    0.955556
        2       C         (7.91, 14.454]     1.000000
                          (14.454, 31.0]     1.000000
                          (31.0, 512.329]    1.000000
                Q         (7.91, 14.454]     1.000000
                S         (7.91, 14.454]     0.875000
                          (14.454, 31.0]     0.916667
                          (31.0, 512.329]    1.000000
        3       C         (-0.001, 7.91]     1.000000
                          (7.91, 14.454]     0.428571
                          (14.454, 31.0]     0.666667
                Q         (-0.001, 7.91]     0.750000
                          (7.91, 14.454]     0.500000
                          (14.454, 31.0]     0.714286
                S         (-0.001, 7.91]     0.533333
                          (7.91, 14.454]     0.448276
                          (14.454, 31.0]     0.357143
                          (31.0, 512.329]    0.125000
Name: Survived, dtype: float64


Survival Decision Tree w/Male Node:
 Sex   Title
male  Master    0.575000
      Misc      0.250000
      Mr        0.156673
Name: Survived, dtype: float64
```

In [25]:

```python
#handmade data model using brain power (and Microsoft Excel Pivot Tables for quick calculations)
def mytree(df):

    #initialize table to store predictions
    Model = pd.DataFrame(data = {'Predict':[]})
    male_title = ['Master'] #survived titles

    for index, row in df.iterrows():

        #Question 1: Were you on the Titanic; majority died
        Model.loc[index, 'Predict'] = 0
```

```
        #Question 2: Are you female; majority survived
        if (df.loc[index, 'Sex'] == 'female'):
                  Model.loc[index, 'Predict'] = 1

        #Question 3A Female - Class and Question 4 Embarked gain minimum information

        #Question 5B Female - FareBin; set anything less than .5 in female node decision tree back to 0
        if ((df.loc[index, 'Sex'] == 'female') &
            (df.loc[index, 'Pclass'] == 3) &
            (df.loc[index, 'Embarked'] == 'S')  &
            (df.loc[index, 'Fare'] > 8)

           ):
                  Model.loc[index, 'Predict'] = 0

        #Question 3B Male: Title; set anything greater than .5 to 1 for majority survived
        if ((df.loc[index, 'Sex'] == 'male') &
            (df.loc[index, 'Title'] in male_title)
            ):
            Model.loc[index, 'Predict'] = 1


    return Model


#model data
Tree_Predict = mytree(data1)
print('Decision Tree Model Accuracy/Precision Score: {:.2f}%\n'.format(metrics.accuracy_score(data1['Survived'], Tree_P


#Accuracy Summary Report
#Where recall score = (true positives)/(true positive + false negative) w/1 being best
#And F1 score = weighted average of precision and recall w/1 being best
print(metrics.classification_report(data1['Survived'], Tree_Predict))
```

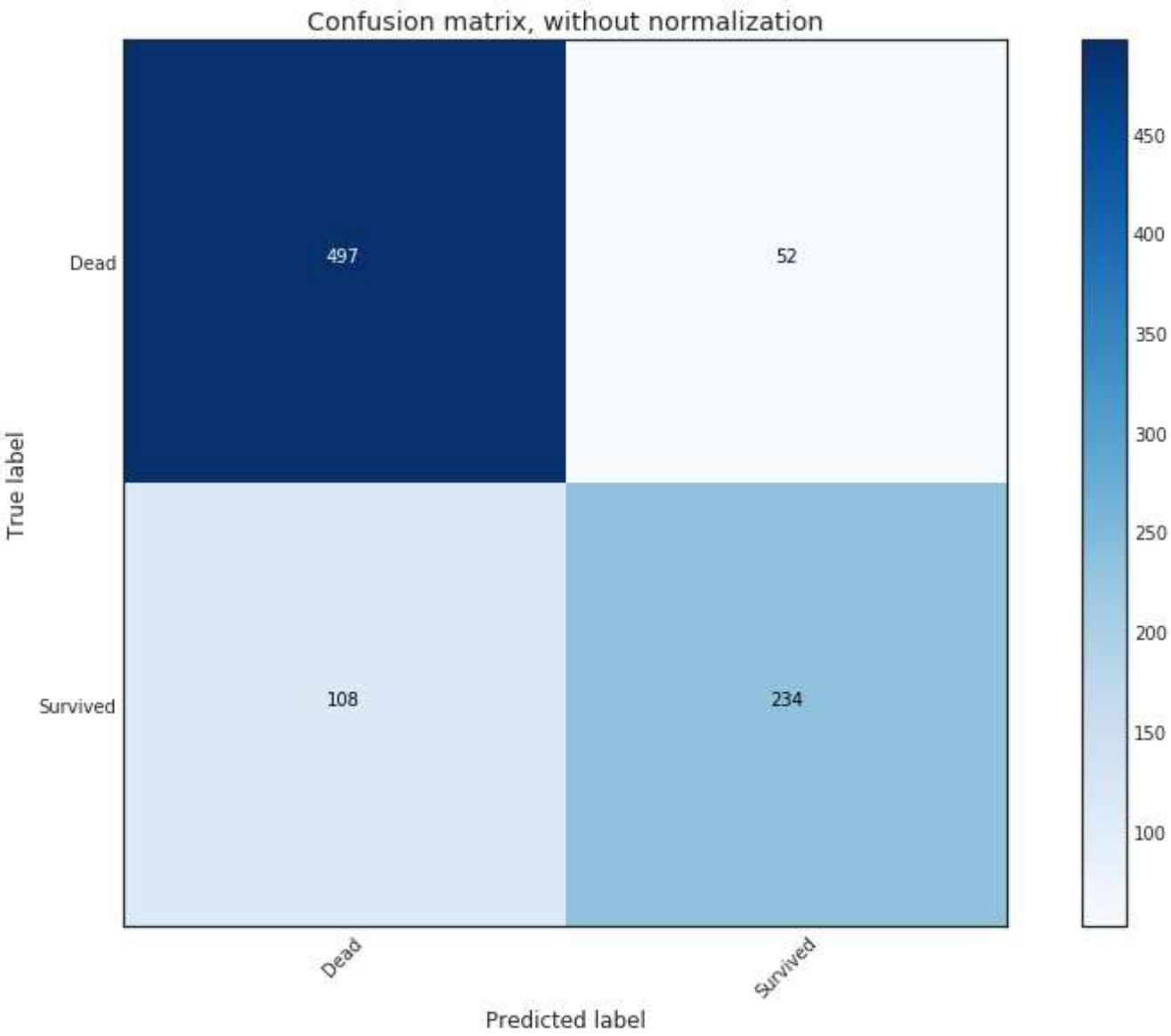
```
Decision Tree Model Accuracy/Precision Score: 82.04%

             precision    recall  f1-score   support

          0       0.82      0.91      0.86       549
          1       0.82      0.68      0.75       342

avg / total       0.82      0.82      0.82       891
```

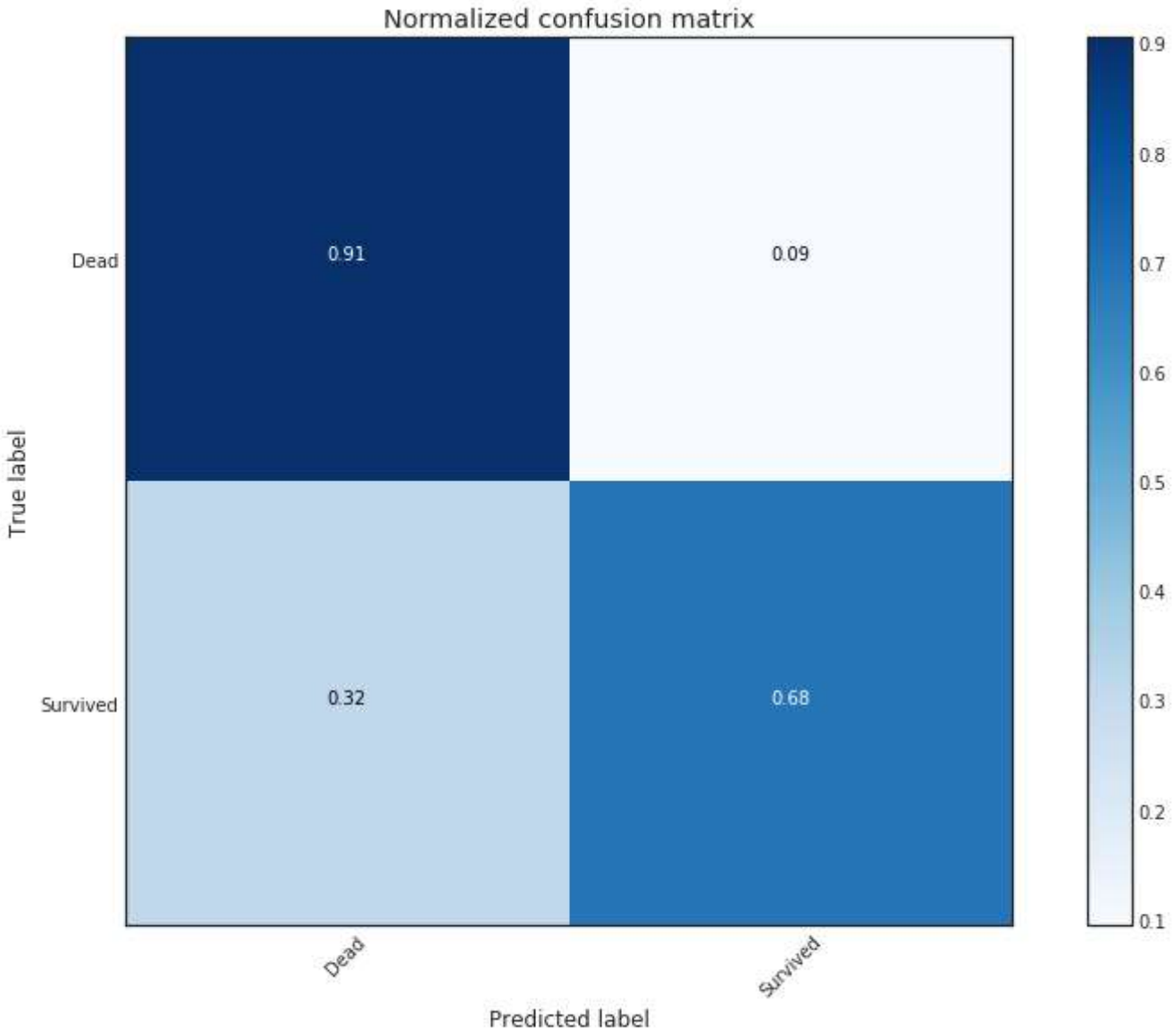
In [26]:

```
#Plot Accuracy Summary
#Credit
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cnf_matrix = metrics.confusion_matrix(data1['Survived'], Tree_Predict)
np.set_printoptions(precision=2)

class_names = ['Dead', 'Survived']
# Plot non-normalized confusion matrix
plt.figure()
```

```
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
```

```
Confusion matrix, without normalization
[[497  52]
 [108 234]]
Normalized confusion matrix
[[ 0.91  0.09]
 [ 0.32  0.68]]
```


Confusion matrix, without normalization
Dead
Survived
True label
497
52
108
234
Dead
Survived
Predicted label
450
400
350
300
250
200
150
100



5.11 Model Performance with Cross-Validation (CV)

It's important we use a different subset for train data to build our model and test data to evaluate our model. Otherwise, our model will be overfitted. Meaning it's great at "predicting" data it's already seen, but terrible at predicting data it has not seen; which is not prediction at all.

Cross-Validation is basically a shortcut to split and score our model multiple times, so we can get an idea of how well it will perform on unseen data. It's a little more expensive in computer processing, but it's important so we don't gain false confidence.

In addition to Cross-Validation we used a customized sklearn train test splitter to allow a little more randomness in our test scoring.

5.12 Tune Model with Hyper-Parameters

We will tune our model using [ParameterGrid], [GridSearchCV], and customized [sklearn scoring]. We will then visualize our tree with [graphviz].

In [27]:
```
#base model
dtree = tree.DecisionTreeClassifier(random_state = 0)
base_results = model_selection.cross_validate(dtree, data1[data1_x_bin], data1[Target], cv  = cv_split)
dtree.fit(data1[data1_x_bin], data1[Target])

print('BEFORE DT Parameters: ', dtree.get_params())
print("BEFORE DT Training w/bin score mean: {:.2f}". format(base_results['train_score'].mean()*100))
print("BEFORE DT Test w/bin score mean: {:.2f}". format(base_results['test_score'].mean()*100))
print("BEFORE DT Test w/bin score 3*std: +/- {:.2f}". format(base_results['test_score'].std()*100*3))
#print("BEFORE DT Test w/bin set score min: {:.2f}". format(base_results['test_score'].min()*100))
print('-'*10)


#tune hyper-parameters
param_grid = {'criterion': ['gini', 'entropy'],  #scoring methodology; two supported formulas for calculating informatic
              #'splitter': ['best', 'random'], #splitting methodology; two supported strategies - default is best
              'max_depth': [2,4,6,8,10,None], #max depth tree can grow; default is none
              #'min_samples_split': [2,5,10,.03,.05], #minimum subset size BEFORE new split (fraction is % of total); de
              #'min_samples_leaf': [1,5,10,.03,.05], #minimum subset size AFTER new split split (fraction is % of total)
              #'max_features': [None, 'auto'], #max features to consider when performing split; default none or all
              'random_state': [0] #seed or control random number generator:
             }

#print(list(model_selection.ParameterGrid(param_grid)))

#choose best model with grid_search:

tune_model = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_grid=param_grid, scoring = 'roc_auc', cv
tune_model.fit(data1[data1_x_bin], data1[Target])

#print(tune_model.cv_results_.keys())
```

```
#print(tune_model.cv_results_['params'])
print('AFTER DT Parameters: ', tune_model.best_params_)
#print(tune_model.cv_results_['mean_train_score'])
print("AFTER DT Training w/bin score mean: {:.2f}". format(tune_model.cv_results_['mean_train_score'][tune_model.best_i
#print(tune_model.cv_results_['mean_test_score'])
print("AFTER DT Test w/bin score mean: {:.2f}". format(tune_model.cv_results_['mean_test_score'][tune_model.best_index_
print("AFTER DT Test w/bin score 3*std: +/- {:.2f}". format(tune_model.cv_results_['std_test_score'][tune_model.best_in
print('-'*10)


#duplicates gridsearchcv
#tune_results = model_selection.cross_validate(tune_model, data1[data1_x_bin], data1[Target], cv  = cv_split)

#print('AFTER DT Parameters: ', tune_model.best_params_)
#print("AFTER DT Training w/bin set score mean: {:.2f}". format(tune_results['train_score'].mean()*100))
#print("AFTER DT Test w/bin set score mean: {:.2f}". format(tune_results['test_score'].mean()*100))
#print("AFTER DT Test w/bin set score min: {:.2f}". format(tune_results['test_score'].min()*100))
#print('-'*10)
```

```
BEFORE DT Parameters:  {'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_n
odes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': 0, 'splitter': 'best'}
BEFORE DT Training w/bin score mean: 89.51
BEFORE DT Test w/bin score mean: 82.09
BEFORE DT Test w/bin score 3*std: +/- 5.57
----------
AFTER DT Parameters:  {'criterion': 'gini', 'max_depth': 4, 'random_state': 0}
AFTER DT Training w/bin score mean: 89.35
AFTER DT Test w/bin score mean: 87.40
AFTER DT Test w/bin score 3*std: +/- 5.00
----------
```

5.13 Tune Model with Feature Selection

As stated in the beginning, more predictor variables do not make a better model, but the right predictors do. So another step in data modeling is feature selection. we will use [recursive feature elimination (RFE) with cross validation (CV)]

In [28]:

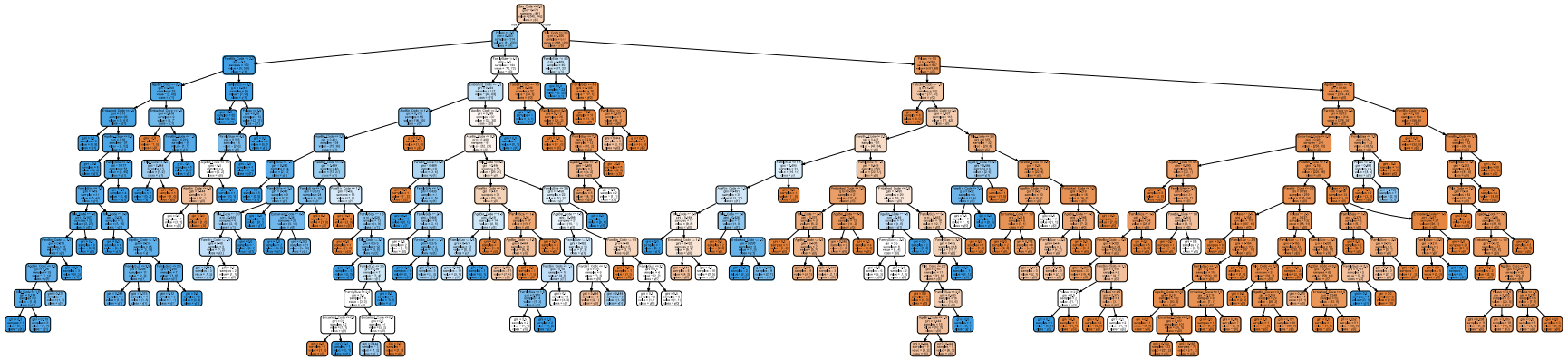
```
#base model
print('BEFORE DT RFE Training Shape Old: ', data1[data1_x_bin].shape)
print('BEFORE DT RFE Training Columns Old: ', data1[data1_x_bin].columns.values)

print("BEFORE DT RFE Training w/bin score mean: {:.2f}". format(base_results['train_score'].mean()*100))
print("BEFORE DT RFE Test w/bin score mean: {:.2f}". format(base_results['test_score'].mean()*100))
```

```
print("BEFORE DT RFE Test w/bin score 3*std: +/- {:.2f}". format(base_results['test_score'].std()*100*3))
print('-'*10)



#feature selection
dtree_rfe = feature_selection.RFECV(dtree, step = 1, scoring = 'accuracy', cv = cv_split)
dtree_rfe.fit(data1[data1_x_bin], data1[Target])

#transform x&y to reduced features and fit new model
#alternative: can use pipeline to reduce fit and transform steps
X_rfe = data1[data1_x_bin].columns.values[dtree_rfe.get_support()]
rfe_results = model_selection.cross_validate(dtree, data1[X_rfe], data1[Target], cv  = cv_split)

#print(dtree_rfe.grid_scores_)
print('AFTER DT RFE Training Shape New: ', data1[X_rfe].shape)
print('AFTER DT RFE Training Columns New: ', X_rfe)

print("AFTER DT RFE Training w/bin score mean: {:.2f}". format(rfe_results['train_score'].mean()*100))
print("AFTER DT RFE Test w/bin score mean: {:.2f}". format(rfe_results['test_score'].mean()*100))
print("AFTER DT RFE Test w/bin score 3*std: +/- {:.2f}". format(rfe_results['test_score'].std()*100*3))
print('-'*10)


#tune rfe model
rfe_tune_model = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_grid=param_grid, scoring = 'roc_auc'
rfe_tune_model.fit(data1[X_rfe], data1[Target])

#print(rfe_tune_model.cv_results_.keys())
#print(rfe_tune_model.cv_results_['params'])
print('AFTER DT RFE Tuned Parameters: ', rfe_tune_model.best_params_)
#print(rfe_tune_model.cv_results_['mean_train_score'])
print("AFTER DT RFE Tuned Training w/bin score mean: {:.2f}". format(rfe_tune_model.cv_results_['mean_train_score'][tun
#print(rfe_tune_model.cv_results_['mean_test_score'])
print("AFTER DT RFE Tuned Test w/bin score mean: {:.2f}". format(rfe_tune_model.cv_results_['mean_test_score'][tune_mod
print("AFTER DT RFE Tuned Test w/bin score 3*std: +/- {:.2f}". format(rfe_tune_model.cv_results_['std_test_score'][tune
print('-'*10)
```

```
BEFORE DT RFE Training Shape Old:  (891, 7)
BEFORE DT RFE Training Columns Old:  ['Sex_Code' 'Pclass' 'Embarked_Code' 'Title_Code' 'FamilySize'
 'AgeBin_Code' 'FareBin_Code']
BEFORE DT RFE Training w/bin score mean: 89.51
BEFORE DT RFE Test w/bin score mean: 82.09
BEFORE DT RFE Test w/bin score 3*std: +/- 5.57
----------
AFTER DT RFE Training Shape New:  (891, 6)
AFTER DT RFE Training Columns New:  ['Sex_Code' 'Pclass' 'Title_Code' 'FamilySize' 'AgeBin_Code' 'FareBin_Code']
AFTER DT RFE Training w/bin score mean: 88.16
AFTER DT RFE Test w/bin score mean: 83.06
AFTER DT RFE Test w/bin score 3*std: +/- 6.22
----------
AFTER DT RFE Tuned Parameters:  {'criterion': 'gini', 'max_depth': 4, 'random_state': 0}
AFTER DT RFE Tuned Training w/bin score mean: 89.39
AFTER DT RFE Tuned Test w/bin score mean: 87.34
AFTER DT RFE Tuned Test w/bin score 3*std: +/- 6.21
----------
```

In [29]:
```
#Graph MLA version of Decision Tree
import graphviz
dot_data = tree.export_graphviz(dtree, out_file=None,
                                feature_names = data1_x_bin, class_names = True,
                                filled = True, rounded = True)
graph = graphviz.Source(dot_data)
graph
```

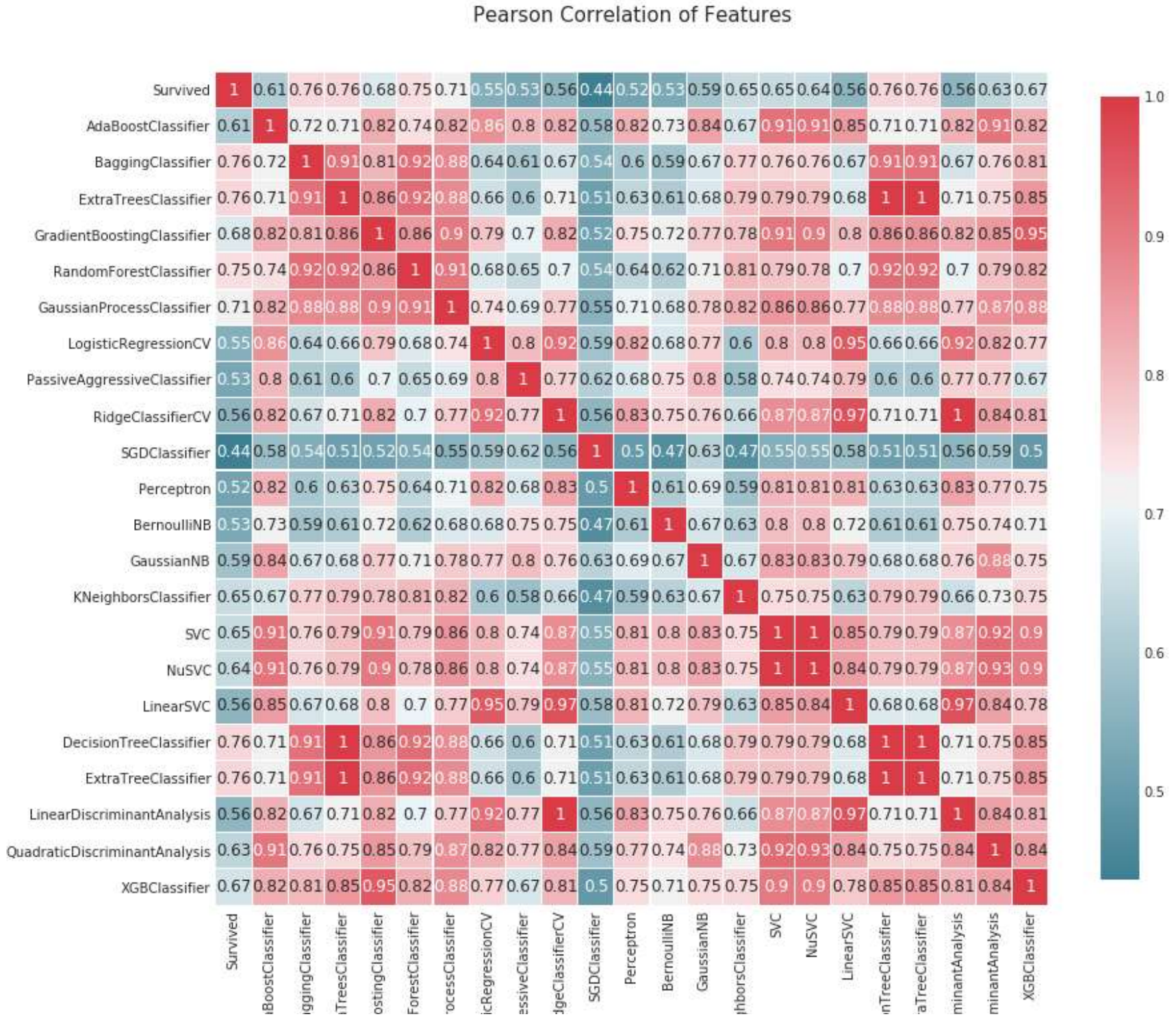
Out[29]:



Step 6: Validate and Implement

The next step is to prepare for submission using the validation data.

In [30]:
```
#compare algorithm predictions with each other, where 1 = exactly similar and 0 = exactly opposite
#there are some 1's, but enough blues and light reds to create a "super algorithm" by combining them
correlation_heatmap(MLA_predict)
```


Pearson Correlation of Features
Survived
AdaBoostClassifier
BaggingClassifier
ExtraTreesClassifier
GradientBoostingClassifier
RandomForestClassifier
GaussianProcessClassifier
LogisticRegressionCV
PassiveAggressiveClassifier
RidgeClassifierCV
SGDClassifier
Perceptron
BernoulliNB
GaussianNB
KNeighborsClassifier
SVC
NuSVC
LinearSVC
DecisionTreeClassifier
ExtraTreeClassifier
LinearDiscriminantAnalysis
QuadraticDiscriminantAnalysis
XGBClassifier
1.0
0.9
0.8
0.7
0.6
0.5

In [31]:
```
#why choose one model, when you can pick them all with voting classifier

#removed models w/o attribute 'predict_proba' required for vote classifier and models with a 1.0 correlation to another
vote_est = [
    #Ensemble Methods:
    ('ada', ensemble.AdaBoostClassifier()),
    ('bc', ensemble.BaggingClassifier()),
    ('etc',ensemble.ExtraTreesClassifier()),
    ('gbc', ensemble.GradientBoostingClassifier()),
    ('rfc', ensemble.RandomForestClassifier()),

    #Gaussian Processes:
    ('gpc', gaussian_process.GaussianProcessClassifier()),

    #GLM:
    ('lr', linear_model.LogisticRegressionCV()),

    #Navies Bayes:
    ('bnb', naive_bayes.BernoulliNB()),
    ('gnb', naive_bayes.GaussianNB()),

    #Nearest Neighbor:
    ('knn', neighbors.KNeighborsClassifier()),

    #SVM:
    ('svc', svm.SVC(probability=True)),

    #xgboost:
   ('xgb', XGBClassifier())

]


#Hard Vote or majority rules
vote_hard = ensemble.VotingClassifier(estimators = vote_est , voting = 'hard')
vote_hard_cv = model_selection.cross_validate(vote_hard, data1[data1_x_bin], data1[Target], cv  = cv_split)
vote_hard.fit(data1[data1_x_bin], data1[Target])

print("Hard Voting Training w/bin score mean: {:.2f}". format(vote_hard_cv['train_score'].mean()*100))
```

```
print("Hard Voting Test w/bin score mean: {:.2f}". format(vote_hard_cv['test_score'].mean()*100))
print("Hard Voting Test w/bin score 3*std: +/- {:.2f}". format(vote_hard_cv['test_score'].std()*100*3))
print('-'*10)


#Soft Vote or weighted probabilities
vote_soft = ensemble.VotingClassifier(estimators = vote_est , voting = 'soft')
vote_soft_cv = model_selection.cross_validate(vote_soft, data1[data1_x_bin], data1[Target], cv  = cv_split)
vote_soft.fit(data1[data1_x_bin], data1[Target])

print("Soft Voting Training w/bin score mean: {:.2f}". format(vote_soft_cv['train_score'].mean()*100))
print("Soft Voting Test w/bin score mean: {:.2f}". format(vote_soft_cv['test_score'].mean()*100))
print("Soft Voting Test w/bin score 3*std: +/- {:.2f}". format(vote_soft_cv['test_score'].std()*100*3))
print('-'*10)
```

```
Hard Voting Training w/bin score mean: 86.61
Hard Voting Test w/bin score mean: 82.46
Hard Voting Test w/bin score 3*std: +/- 4.42
----------
Soft Voting Training w/bin score mean: 87.15
Soft Voting Test w/bin score mean: 82.35
Soft Voting Test w/bin score 3*std: +/- 4.80
----------
```

In [32]:

```
#tune each estimator before creating a super model

grid_n_estimator = [50,100,300]
grid_ratio = [.1,.25,.5,.75,1.0]
grid_learn = [.01,.03,.05,.1,.25]
grid_max_depth = [2,4,6,None]
grid_min_samples = [5,10,.03,.05,.10]
grid_criterion = ['gini', 'entropy']
grid_bool = [True, False]
grid_seed = [0]

vote_param = [{
#
            'ada__n_estimators': grid_n_estimator,
            'ada__learning_rate': grid_ratio,
            'ada__algorithm': ['SAMME', 'SAMME.R'],
            'ada__random_state': grid_seed,


            'bc__n_estimators': grid_n_estimator,
            'bc__max_samples': grid_ratio,
```

```
            'bc__oob_score': grid_bool,
            'bc__random_state': grid_seed,


            'etc__n_estimators': grid_n_estimator,
            'etc__criterion': grid_criterion,
            'etc__max_depth': grid_max_depth,
            'etc__random_state': grid_seed,



            'gbc__loss': ['deviance', 'exponential'],
            'gbc__learning_rate': grid_ratio,
            'gbc__n_estimators': grid_n_estimator,
            'gbc__criterion': ['friedman_mse', 'mse', 'mae'],
            'gbc__max_depth': grid_max_depth,
            'gbc__min_samples_split': grid_min_samples,
            'gbc__min_samples_leaf': grid_min_samples,
            'gbc__random_state': grid_seed,


            'rfc__n_estimators': grid_n_estimator,
            'rfc__criterion': grid_criterion,
            'rfc__max_depth': grid_max_depth,
            'rfc__min_samples_split': grid_min_samples,
            'rfc__min_samples_leaf': grid_min_samples,
            'rfc__bootstrap': grid_bool,
            'rfc__oob_score': grid_bool,
            'rfc__random_state': grid_seed,

            #http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html#sklearn.li
            'lr__fit_intercept': grid_bool,
            'lr__penalty': ['l1','l2'],
            'lr__solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
            'lr__random_state': grid_seed,


            'bnb__alpha': grid_ratio,
            'bnb__prior': grid_bool,
            'bnb__random_state': grid_seed,


            'knn__n_neighbors': [1,2,3,4,5,6,7],
            'knn__weights': ['uniform', 'distance'],
            'knn__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
```

```
            'knn__random_state': grid_seed,



            'svc__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
            'svc__C': grid_max_depth,
            'svc__gamma': grid_ratio,
            'svc__decision_function_shape': ['ovo', 'ovr'],
            'svc__probability': [True],
            'svc__random_state': grid_seed,



            'xgb__learning_rate': grid_ratio,
            'xgb__max_depth': [2,4,6,8,10],
            'xgb__tree_method': ['exact', 'approx', 'hist'],
            'xgb__objective': ['reg:linear', 'reg:logistic', 'binary:logistic'],
            'xgb__seed': grid_seed

        }]



#Soft Vote with tuned models
#grid_soft = model_selection.GridSearchCV(estimator = vote_soft, param_grid = vote_param, cv = 2, scoring = 'roc_auc')
#grid_soft.fit(data1[data1_x_bin], data1[Target])

#print(grid_soft.cv_results_.keys())
#print(grid_soft.cv_results_['params'])
#print('Soft Vote Tuned Parameters: ', grid_soft.best_params_)
#print(grid_soft.cv_results_['mean_train_score'])
#print("Soft Vote Tuned Training w/bin set score mean: {:.2f}". format(grid_soft.cv_results_['mean_train_score'][tune_mo
#print(grid_soft.cv_results_['mean_test_score'])
#print("Soft Vote Tuned Test w/bin set score mean: {:.2f}". format(grid_soft.cv_results_['mean_test_score'][tune_model.b
#print("Soft Vote Tuned Test w/bin score 3*std: +/- {:.2f}". format(grid_soft.cv_results_['std_test_score'][tune_model.b
#print('-'*10)


#credit:
#cv_keys = ('mean_test_score', 'std_test_score', 'params')
#for r, _ in enumerate(grid_soft.cv_results_['mean_test_score']):
#    print("%0.3f +/- %0.2f %r"
#          % (grid_soft.cv_results_[cv_keys[0]][r],
#             grid_soft.cv_results_[cv_keys[1]][r] / 2.0,
```

```
#             grid_soft.cv_results_[cv_keys[2]][r]))


#print('-'*10)
```

In [33]:

```
#Hyperparameter Tune with GridSearchCV:
grid_n_estimator = [10, 50, 100, 300]
grid_ratio = [.1, .25, .5, .75, 1.0]
grid_learn = [.01, .03, .05, .1, .25]
grid_max_depth = [2, 4, 6, 8, 10, None]
grid_min_samples = [5, 10, .03, .05, .10]
grid_criterion = ['gini', 'entropy']
grid_bool = [True, False]
grid_seed = [0]


grid_param = [
            [{
            #AdaBoostClassifier
            'n_estimators': grid_n_estimator, #default=50
            'learning_rate': grid_learn, #default=1
            #'algorithm': ['SAMME', 'SAMME.R'], #default=’SAMME.R
            'random_state': grid_seed
            }],


            [{
            #BaggingClassifier
            'max_samples': grid_ratio, #default=1.0
            'random_state': grid_seed
             }],


            [{
            #ExtraTreesClassifier
            'n_estimators': grid_n_estimator, #default=10
            'criterion': grid_criterion, #default=”gini”
            'max_depth': grid_max_depth, #default=None
            'random_state': grid_seed
             }],


            [{
            #GradientBoostingClassifier
            #'loss': ['deviance', 'exponential'], #default=’deviance’
```

```
            'learning_rate': [.05], #default=0.1 -- 12/31/17 set to reduce runtime -- The best parameter for GradientBoo
            'n_estimators': [300], #default=100 -- 12/31/17 set to reduce runtime -- The best parameter for GradientBoos
            #'criterion': ['friedman_mse', 'mse', 'mae'], #default="friedman_mse"
            'max_depth': grid_max_depth, #default=3
            'random_state': grid_seed
             }],


            [{
            #RandomForestClassifier
            'n_estimators': grid_n_estimator, #default=10
            'criterion': grid_criterion, #default="gini"
            'max_depth': grid_max_depth, #default=None
            'oob_score': [True], #default=False -- 12/31/17 set to reduce runtime -- The best parameter for RandomFores
            'random_state': grid_seed
             }],

            [{
            #GaussianProcessClassifier
            'max_iter_predict': grid_n_estimator, #default: 100
            'random_state': grid_seed
            }],


            [{
            #LogisticRegressionCV
            'fit_intercept': grid_bool, #default: True
            #'penalty': ['l1','l2'],
            'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], #default: lbfgs
            'random_state': grid_seed
             }],


            [{
            #BernoulliNB
            'alpha': grid_ratio, #default: 1.0
             }],


            #GaussianNB -
            [{}],

            [{
            #KNeighborsClassifier
            'n_neighbors': [1,2,3,4,5,6,7], #default: 5
```

```
            'weights': ['uniform', 'distance'], #default = 'uniform'
            'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
            }],


            [{
            #SVC

            #'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
            'C': [1,2,3,4,5], #default=1.0
            'gamma': grid_ratio, #edfault: auto
            'decision_function_shape': ['ovo', 'ovr'], #default:ovr
            'probability': [True],
            'random_state': grid_seed
             }],


            [{
            #XGBClassifier
            'learning_rate': grid_learn, #default: .3
            'max_depth': [1,2,4,6,8,10], #default 2
            'n_estimators': grid_n_estimator,
            'seed': grid_seed
             }]
        ]



start_total = time.perf_counter()
for clf, param in zip (vote_est, grid_param):

    #print(clf[1]) #vote_est is a list of tuples, index 0 is the name and index 1 is the algorithm
    #print(param)


    start = time.perf_counter()
    best_search = model_selection.GridSearchCV(estimator = clf[1], param_grid = param, cv = cv_split, scoring = 'roc_au
    best_search.fit(data1[data1_x_bin], data1[Target])
    run = time.perf_counter() - start

    best_param = best_search.best_params_
    print('The best parameter for {} is {} with a runtime of {:.2f} seconds.'.format(clf[1].__class__.__name__, best_pa
    clf[1].set_params(**best_param)
```

```
run_total = time.perf_counter() - start_total
print('Total optimization time was {:.2f} minutes.'.format(run_total/60))

print('-'*10)
```

```
The best parameter for AdaBoostClassifier is {'learning_rate': 0.1, 'n_estimators': 300, 'random_state': 0} with a runt
ime of 30.55 seconds.
The best parameter for BaggingClassifier is {'max_samples': 0.25, 'n_estimators': 300, 'random_state': 0} with a runtim
e of 27.66 seconds.
The best parameter for ExtraTreesClassifier is {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 100, 'random_st
ate': 0} with a runtime of 60.09 seconds.
The best parameter for GradientBoostingClassifier is {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators': 300, 'rand
om_state': 0} with a runtime of 32.23 seconds.
The best parameter for RandomForestClassifier is {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 100, 'oob_sco
re': True, 'random_state': 0} with a runtime of 69.79 seconds.
The best parameter for GaussianProcessClassifier is {'max_iter_predict': 10, 'random_state': 0} with a runtime of 38.30
seconds.
The best parameter for LogisticRegressionCV is {'fit_intercept': True, 'random_state': 0, 'solver': 'liblinear'} with a
runtime of 7.15 seconds.
The best parameter for BernoulliNB is {'alpha': 0.1} with a runtime of 0.17 seconds.
The best parameter for GaussianNB is {} with a runtime of 0.04 seconds.
The best parameter for KNeighborsClassifier is {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'uniform'} with a ru
ntime of 4.42 seconds.
The best parameter for SVC is {'C': 2, 'decision_function_shape': 'ovo', 'gamma': 0.1, 'probability': True, 'random_sta
te': 0} with a runtime of 25.23 seconds.
The best parameter for XGBClassifier is {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 300, 'seed': 0} with a
runtime of 45.15 seconds.
Total optimization time was 5.68 minutes.
----------
```

In [34]:

```
#Hard Vote or majority rules w/Tuned Hyperparameters
grid_hard = ensemble.VotingClassifier(estimators = vote_est , voting = 'hard')
grid_hard_cv = model_selection.cross_validate(grid_hard, data1[data1_x_bin], data1[Target], cv  = cv_split)
grid_hard.fit(data1[data1_x_bin], data1[Target])

print("Hard Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_hard_cv['train_score'].mean(
print("Hard Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_hard_cv['test_score'].mean()*100
print("Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_hard_cv['test_score'].std()
print('-'*10)

#Soft Vote or weighted probabilities w/Tuned Hyperparameters
grid_soft = ensemble.VotingClassifier(estimators = vote_est , voting = 'soft')
grid_soft_cv = model_selection.cross_validate(grid_soft, data1[data1_x_bin], data1[Target], cv  = cv_split)
grid_soft.fit(data1[data1_x_bin], data1[Target])
```

```
print("Soft Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_soft_cv['train_score'].mean(
print("Soft Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_soft_cv['test_score'].mean()*100
print("Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_soft_cv['test_score'].std()
print('-'*10)

#The best parameter for AdaBoostClassifier is {'learning_rate': 0.1, 'n_estimators': 300, 'random_state': 0} with a runt
#The best parameter for BaggingClassifier is {'max_samples': 0.25, 'n_estimators': 300, 'random_state': 0} with a runtim
#The best parameter for ExtraTreesClassifier is {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 100, 'random_st
#The best parameter for GradientBoostingClassifier is {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators': 300, 'rand
#The best parameter for RandomForestClassifier is {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 100, 'oob_sco
#The best parameter for GaussianProcessClassifier is {'max_iter_predict': 10, 'random_state': 0} with a runtime of 6.01
#The best parameter for LogisticRegressionCV is {'fit_intercept': True, 'random_state': 0, 'solver': 'liblinear'} with a
#The best parameter for BernoulliNB is {'alpha': 0.1} with a runtime of 0.19 seconds.
#The best parameter for GaussianNB is {} with a runtime of 0.04 seconds.
#The best parameter for KNeighborsClassifier is {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'uniform'} with a ru
#The best parameter for SVC is {'C': 2, 'decision_function_shape': 'ovo', 'gamma': 0.1, 'probability': True, 'random_sta
#The best parameter for XGBClassifier is {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 300, 'seed': 0} with a
#Total optimization time was 5.56 minutes.
```

```
Hard Voting w/Tuned Hyperparameters Training w/bin score mean: 85.22
Hard Voting w/Tuned Hyperparameters Test w/bin score mean: 82.31
Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 5.26
----------
Soft Voting w/Tuned Hyperparameters Training w/bin score mean: 84.76
Soft Voting w/Tuned Hyperparameters Test w/bin score mean: 82.28
Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 5.42
----------
```

In [35]:

```
#prepare data for modeling
print(data_val.info())
print("-"*10)
#data_val.sample(10)
#handmade decision tree - submission score = 0.77990
data_val['Survived'] = mytree(data_val).astype(int)


#decision tree w/full dataset modeling submission score: defaults= 0.76555, tuned= 0.77990
#submit_dt = tree.DecisionTreeClassifier()
#submit_dt = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_grid=param_grid, scoring = 'roc_auc', cv
#submit_dt.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_dt.best_params_) #Best Parameters:  {'criterion': 'gini', 'max_depth': 4, 'random_sta
#data_val['Survived'] = submit_dt.predict(data_val[data1_x_bin])


#bagging w/full dataset modeling submission score: defaults= 0.75119, tuned= 0.77990
```

```
#submit_bc = ensemble.BaggingClassifier()
#submit_bc = model_selection.GridSearchCV(ensemble.BaggingClassifier(), param_grid= {'n_estimators':grid_n_estimator, 'n
#submit_bc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_bc.best_params_) #Best Parameters:  {'max_samples': 0.25, 'n_estimators': 500, 'oob_s
#data_val['Survived'] = submit_bc.predict(data_val[data1_x_bin])


#extra tree w/full dataset modeling submission score: defaults= 0.76555, tuned= 0.77990
#submit_etc = ensemble.ExtraTreesClassifier()
#submit_etc = model_selection.GridSearchCV(ensemble.ExtraTreesClassifier(), param_grid={'n_estimators': grid_n_estimator
#submit_etc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_etc.best_params_) #Best Parameters:  {'criterion': 'entropy', 'max_depth': 6, 'n_est
#data_val['Survived'] = submit_etc.predict(data_val[data1_x_bin])


#random foreset w/full dataset modeling submission score: defaults= 0.71291, tuned= 0.73205
#submit_rfc = ensemble.RandomForestClassifier()
#submit_rfc = model_selection.GridSearchCV(ensemble.RandomForestClassifier(), param_grid={'n_estimators': grid_n_estimat
#submit_rfc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_rfc.best_params_) #Best Parameters:  {'criterion': 'entropy', 'max_depth': 6, 'n_est
#data_val['Survived'] = submit_rfc.predict(data_val[data1_x_bin])



#ada boosting w/full dataset modeling submission score: defaults= 0.74162, tuned= 0.75119
#submit_abc = ensemble.AdaBoostClassifier()
#submit_abc = model_selection.GridSearchCV(ensemble.AdaBoostClassifier(), param_grid={'n_estimators': grid_n_estimator,
#submit_abc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_abc.best_params_) #Best Parameters:  {'algorithm': 'SAMME.R', 'learning_rate': 0.1,
#data_val['Survived'] = submit_abc.predict(data_val[data1_x_bin])


#gradient boosting w/full dataset modeling submission score: defaults= 0.75119, tuned= 0.77033
#submit_gbc = ensemble.GradientBoostingClassifier()
#submit_gbc = model_selection.GridSearchCV(ensemble.GradientBoostingClassifier(), param_grid={'learning_rate': grid_rat
#submit_gbc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_gbc.best_params_) #Best Parameters:  {'learning_rate': 0.25, 'max_depth': 2, 'n_estim
#data_val['Survived'] = submit_gbc.predict(data_val[data1_x_bin])

#extreme boosting w/full dataset modeling submission score: defaults= 0.73684, tuned= 0.77990
#submit_xgb = XGBClassifier()
#submit_xgb = model_selection.GridSearchCV(XGBClassifier(), param_grid= {'learning_rate': grid_learn, 'max_depth': [0,2,
#submit_xgb.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_xgb.best_params_) #Best Parameters:  {'learning_rate': 0.01, 'max_depth': 4, 'n_estim
#data_val['Survived'] = submit_xgb.predict(data_val[data1_x_bin])
```

```
#hard voting classifier w/full dataset modeling submission score: defaults= 0.75598, tuned = 0.77990
#data_val['Survived'] = vote_hard.predict(data_val[data1_x_bin])
data_val['Survived'] = grid_hard.predict(data_val[data1_x_bin])


#soft voting classifier w/full dataset modeling submission score: defaults= 0.73684, tuned = 0.74162
#data_val['Survived'] = vote_soft.predict(data_val[data1_x_bin])
#data_val['Survived'] = grid_soft.predict(data_val[data1_x_bin])


#submit file
submit = data_val[['PassengerId','Survived']]
submit.to_csv("../working/submit.csv", index=False)

print('Validation Data Distribution: \n', data_val['Survived'].value_counts(normalize = True))
submit.sample(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 21 columns):
PassengerId      418 non-null int64
Pclass           418 non-null int64
Name             418 non-null object
Sex              418 non-null object
Age              418 non-null float64
SibSp            418 non-null int64
Parch            418 non-null int64
Ticket           418 non-null object
Fare             418 non-null float64
Cabin            91 non-null object
Embarked         418 non-null object
FamilySize       418 non-null int64
IsAlone          418 non-null int64
Title            418 non-null object
FareBin          418 non-null category
AgeBin           418 non-null category
Sex_Code         418 non-null int64
Embarked_Code    418 non-null int64
Title_Code       418 non-null int64
AgeBin_Code      418 non-null int64
FareBin_Code     418 non-null int64
dtypes: category(2), float64(2), int64(11), object(6)
memory usage: 63.1+ KB
None
----------
Validation Data Distribution:
 0    0.633971
1    0.366029
Name: Survived, dtype: float64
```

Out[35]:

	PassengerId	Survived
240	1132	1
57	949	0
95	987	0
127	1019	1
293	1185	0
351	1243	0
119	1011	1
251	1143	0
278	1170	0
47	939	0

Step 7: Conclusion

Iteration one of the Data Science Framework appears to have converged with a submission accuracy of 0.77990. Repeated attempts using the same dataset and different implementations of decision trees (adaboost, random forest, gradient boost, xgboost, etc.) with tuning did not surpass the 0.77990 submission accuracy. It's interesting to note that, for this dataset, the simple decision tree algorithm achieved the best default submission score and, even with tuning, maintained the same high accuracy.

While it's essential to avoid making sweeping generalizations based on the testing of a few algorithms on a single dataset, there are several observations regarding this specific dataset:

The training dataset exhibits a different distribution compared to the test/validation dataset and the general population. This difference led to significant disparities between the cross-validation (CV) accuracy scores and the Kaggle submission accuracy scores. Decision tree-based algorithms, when applied to the same dataset, consistently reached a similar accuracy score following appropriate tuning. Despite extensive tuning efforts, no machine learning algorithm outperformed the homemade algorithm. The author's hypothesis is that, for small datasets, a manually crafted algorithm sets the performance benchmark. With these observations in mind, for iteration two, it would be beneficial to allocate more time to preprocessing and feature engineering. This effort aims to better align the CV score with the Kaggle score and enhance the overall accuracy.