Step 1: Define the Problem

For this project, the problem statement is given to us on a golden plater, develop an algorithm to predict the survival outcome of passengers on the Titanic.

......

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, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Step 2: Gather the Data

The dataset is also given to us on a golden plater with test and train data at [Kaggle's Titanic: Machine Learning from Disaster (https://www.kaggle.com/c/titanic/data)](https://www.kaggle.com/c/titanic/data)

Step 3: Prepare Data for Consumption

Since step 2 was provided to us on a golden plater, so is step 3. Therefore, normal processes in data wrangling, such as data architecture, governance, and extraction are out of scope. Thus, only data cleaning is in scope.

3.1 Import Libraries

The following code is written in Python 3.x. Libraries provide pre-written functionality to perform necessary tasks. The idea is why write ten lines of code, when you can write one line.

In [1]:
```
#Load packages
import sys
import pandas as pd
import matplotlib
import numpy as np
import scipy as sp
import sklearn

#misc libraries
import random
import time
```

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. Below are common classes to load.

In [30]:
```
#Common Model Algorithms
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, di
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

#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
%config InlineBackend.figure_format='retina'
```

3.2 Meet and Greet Data

This is the meet and greet step. Get to know your data and learn a little bit about it. What does it look like (datatype and values), what makes it tick (independent/feature variables(s)), what's its goals (dependent/target variable(s)).

1- To begin this step, we first import our data.

2- Next we use the info() and sample() function, to get a quick and dirty overview of variable datatypes (i.e. qualitative vs quantitative).

3- Try to understand the features (take a look at the data dictionary and understand the meaning of each one)

In [46]:
```
data_raw = pd.read_csv('data/train.csv')

#a dataset should be broken into 3 splits: train, test, and (final) validation
#the test file provided is the validation file for competition submission
#we will split the train set into train and test data in future sections
data_val  = pd.read_csv('data/test.csv')
data1 = data_raw.copy(deep = True)

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


#preview data
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: 83.7+ KB
```

Out[46]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [4]:
```
data_raw[~(data_raw['Pclass']==1) & ~(data_raw['Cabin'].isnull())]
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000
66	67	1	2	Nye, Mrs. (Elizabeth Ramell)	female	29.0	0	0	C.A. 29395	10.5000
75	76	0	3	Moen, Mr. Sigurd Hansen	male	25.0	0	0	348123	7.6500
123	124	1	2	Webber, Miss. Susan	female	32.5	0	0	27267	13.0000
128	129	1	3	Peter, Miss. Anna	female	NaN	1	1	2668	22.3583
148	149	0	2	Navratil, Mr. Michel ("Louis M Hoffman")	male	36.5	0	2	230080	26.0000
183	184	1	2	Becker, Master. Richard F	male	1.0	2	1	230136	39.0000
193	194	1	2	Navratil, Master. Michel M	male	3.0	1	1	230080	26.0000
205	206	0	3	Strom, Miss. Telma Matilda	female	2.0	0	1	347054	10.4625
251	252	0	3	Strom, Mrs. Wilhelm (Elna Matilda Persson)	female	29.0	1	1	347054	10.4625
292	293	0	2	Levy, Mr. Rene Jacques	male	36.0	0	0	SC/Paris 2163	12.8750
303	304	1	2	Keane, Miss. Nora A	female	NaN	0	0	226593	12.3500
327	328	1	2	Ball, Mrs. (Ada E Hall)	female	36.0	0	0	28551	13.0000
340	341	1	2	Navratil, Master. Edmond Roger	male	2.0	1	1	230080	26.0000

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
345	346	1	2	Brown, Miss. Amelia "Mildred"	female	24.0	0	0	248733	13.0000
394	395	1	3	Sandstrom, Mrs. Hjalmar (Agnes Charlotta Bengt...	female	24.0	0	2	PP 9549	16.7000
429	430	1	3	Pickard, Mr. Berk (Berk Trembisky)	male	32.0	0	0	SOTON/O.Q. 392078	8.0500
473	474	1	2	Jerwan, Mrs. Amin S (Marie Marthe Thuillard)	female	23.0	0	0	SC/AH Basle 541	13.7917
516	517	1	2	Lemore, Mrs. (Amelia Milley)	female	34.0	0	0	C.A. 34260	10.5000
618	619	1	2	Becker, Miss. Marion Louise	female	4.0	2	1	230136	39.0000
699	700	0	3	Humblen, Mr. Adolf Mathias Nicolai Olsen	male	42.0	0	0	348121	7.6500
715	716	0	3	Soholt, Mr. Peter Andreas Lauritz Andersen	male	19.0	0	0	348124	7.6500
717	718	1	2	Troutt, Miss. Edwina Celia "Winnie"	female	27.0	0	0	34218	10.5000
751	752	1	3	Moor, Master. Meier	male	6.0	0	1	392096	12.4750
772	773	0	2	Mack, Mrs. (Mary)	female	57.0	0	0	S.O./P.P. 3	10.5000
776	777	0	3	Tobin, Mr. Roger	male	NaN	0	0	383121	7.7500
823	824	1	3	Moor, Mrs. (Beila)	female	27.0	0	1	392096	12.4750

In [5]:
```
data_raw[~data_raw['Sex'].isin(['male','female'])]
```

Out[5]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark

In [6]:
```
data_raw.groupby('Sex').count()
```

Out[6]:

Sex	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
female	314	314	314	314	261	314	314	314	314	97	312
male	577	577	577	577	453	577	577	577	577	107	577

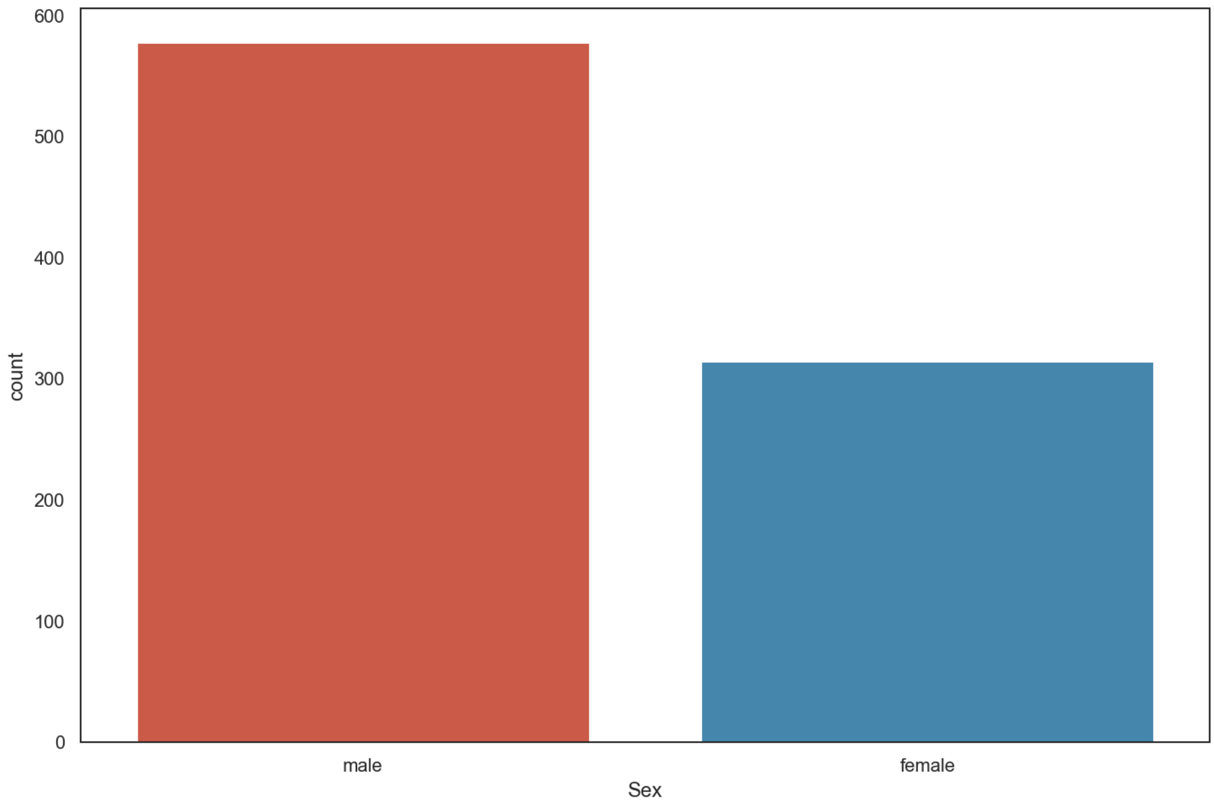
In [7]:
```
data_raw[data_raw['Age'].isnull() & data_raw['Cabin'].isnull()]
```

Out[7]:

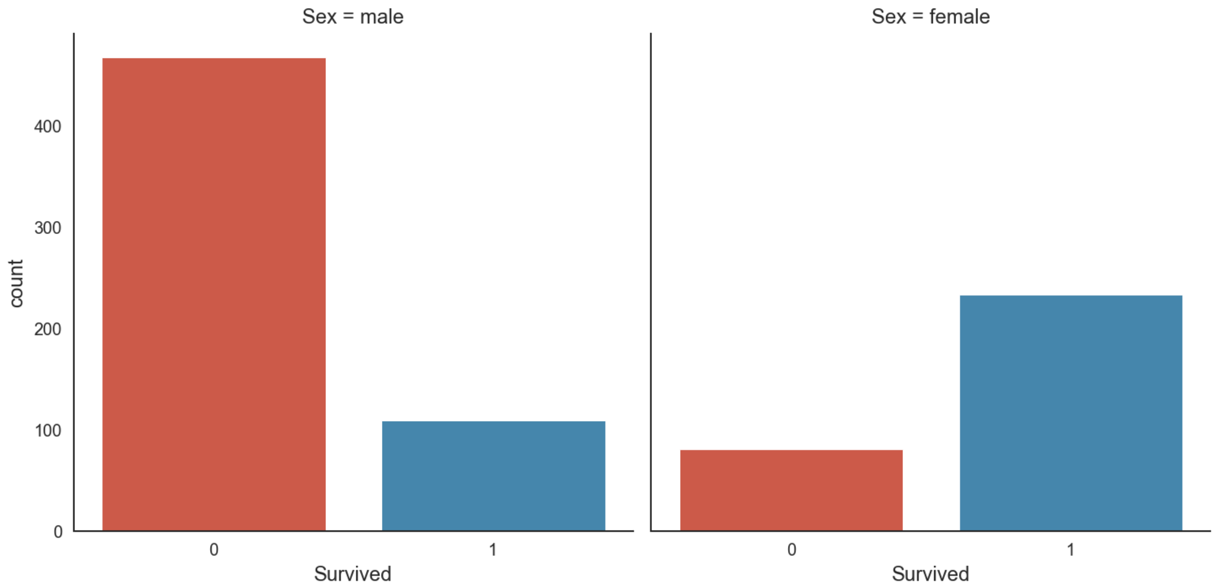
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	N
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	N
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	N
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.2250	N
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	N
...	...	...	...	...	...	...	...	...	...	...	
859	860	0	3	Razi, Mr. Raihed	male	NaN	0	0	2629	7.2292	N
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	N
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5000	N
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958	N
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	N

158 rows × 12 columns

In [31]:
```
sns.countplot(x='Sex', data=data_raw);
```

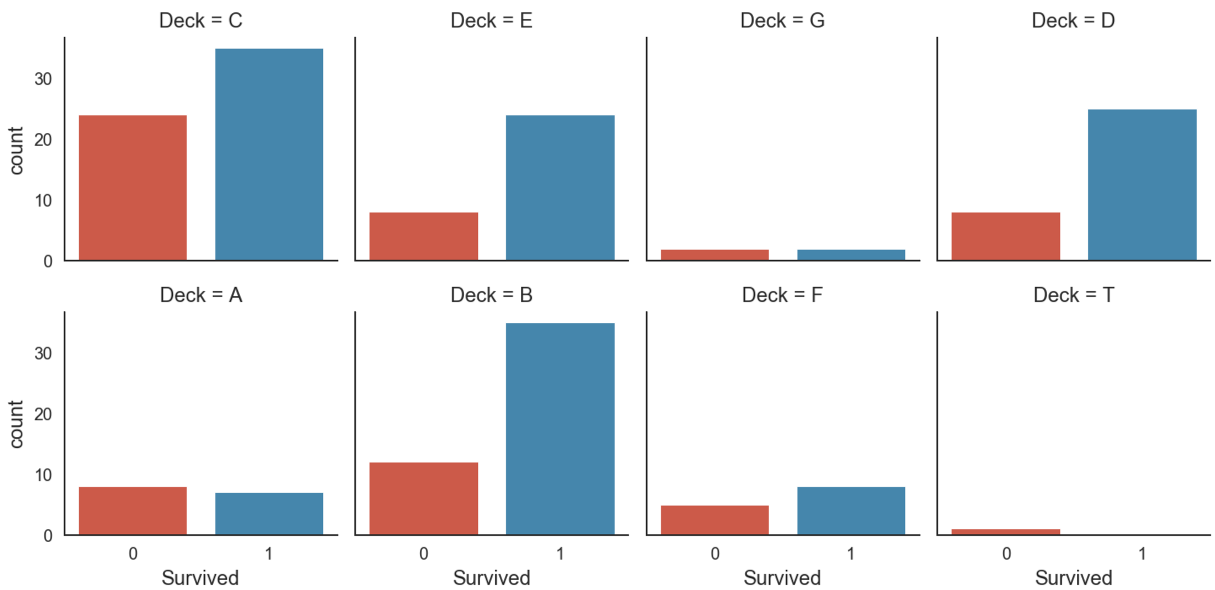


In [9]:
```
g=sns.catplot(x='Survived', col='Sex', kind='count', data=data_raw);
```



In [10]:
```
data2=data_raw.copy(deep=True)
data2['Deck']=data2['Cabin'].str[0]
# data2
sns.catplot(x="Survived", col="Deck", col_wrap=4, data=data2, kind="count", heigl
```

Out[10]: `<seaborn.axisgrid.FacetGrid at 0x173d709edc8>`



In [11]:
```
data2[data2['Deck']=='T']
```

Out[11]:

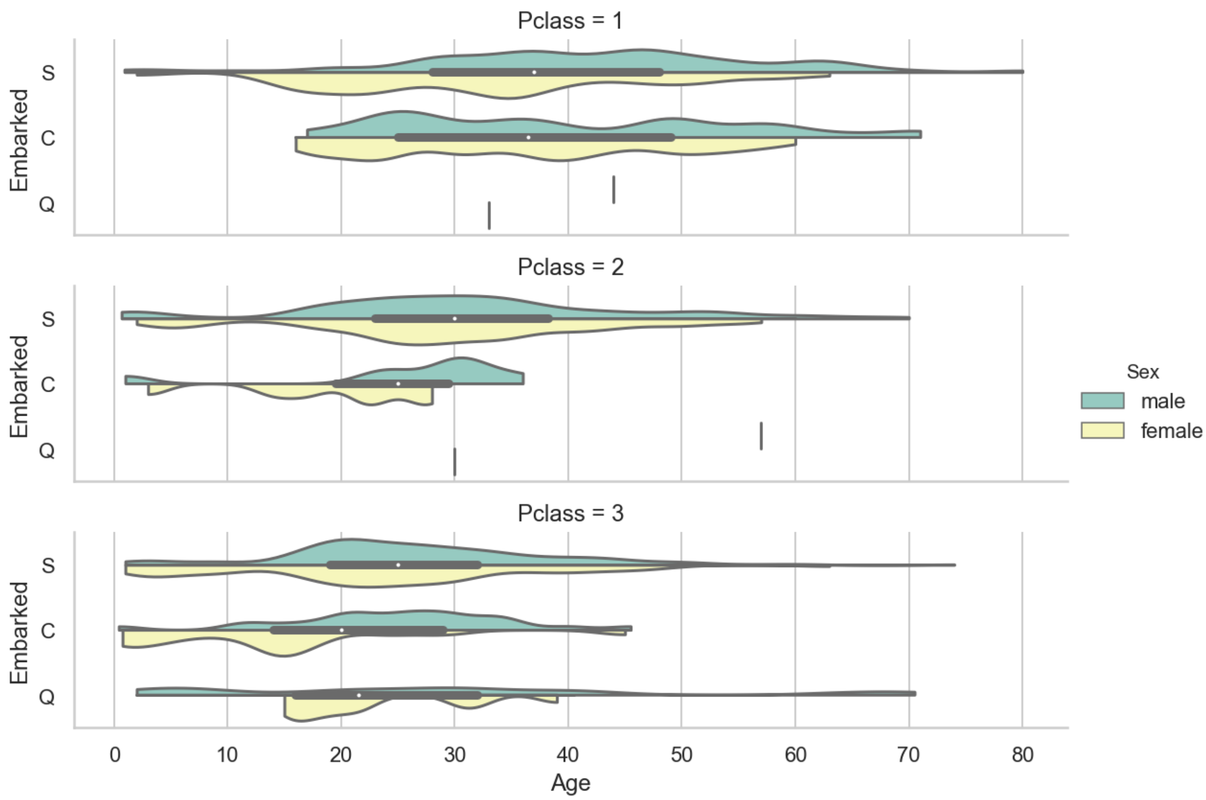
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Er
339	340	0	1	Blackwell, Mr. Stephen Weart	male	45.0	0	0	113784	35.5	T	

In [12]:
```
import re

data3 = data2[data2['Name'].str.contains('Jack' , regex=True)]
data3
# It seems Titanic movie is not real!
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
766	767	0	1	Brewe, Dr. Arthur Jackson	male	NaN	0	0	112379	39.6	NaN	

In [13]:
```
sns.set(style="whitegrid")
g= sns.catplot(x="Age", y="Embarked", hue="Sex", row="Pclass",
                data=data2,
                orient="h", height=2, aspect=4, palette="Set3",
                kind="violin",cut=0,  bw=0.2, split=True) #cut =0 limits the gra

# see params here: https://seaborn.pydata.org/generated/seaborn.violinplot.html
```



Let $(x_1, \ldots, x_n)$ be the observation. The following function is the estimator for Violin diagram Kernel Density Estimation:

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big),$$

where K is the kernel function (a non-negative function) and $h > 0$ is a smoothing parameter called the bandwidth. A kernel with subscript h is called the scaled kernel.

See here for more details: https://en.wikipedia.org/wiki/Kernel_density_estimation (https://en.wikipedia.org/wiki/Kernel_density_estimation)

A typical kernel function is the Normal distribution, with mean on each data point and standard deviation of 1: $K(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

In [14]:
```
data3=data2[['Survived','Pclass','Sex','Age','SibSp','Parch','Fare']]
sns.pairplot(data=data3, hue='Survived')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:48
7: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetools.p
y:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:44
7: RuntimeWarning: invalid value encountered in greater
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:44
7: RuntimeWarning: invalid value encountered in less
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x173d7518588>

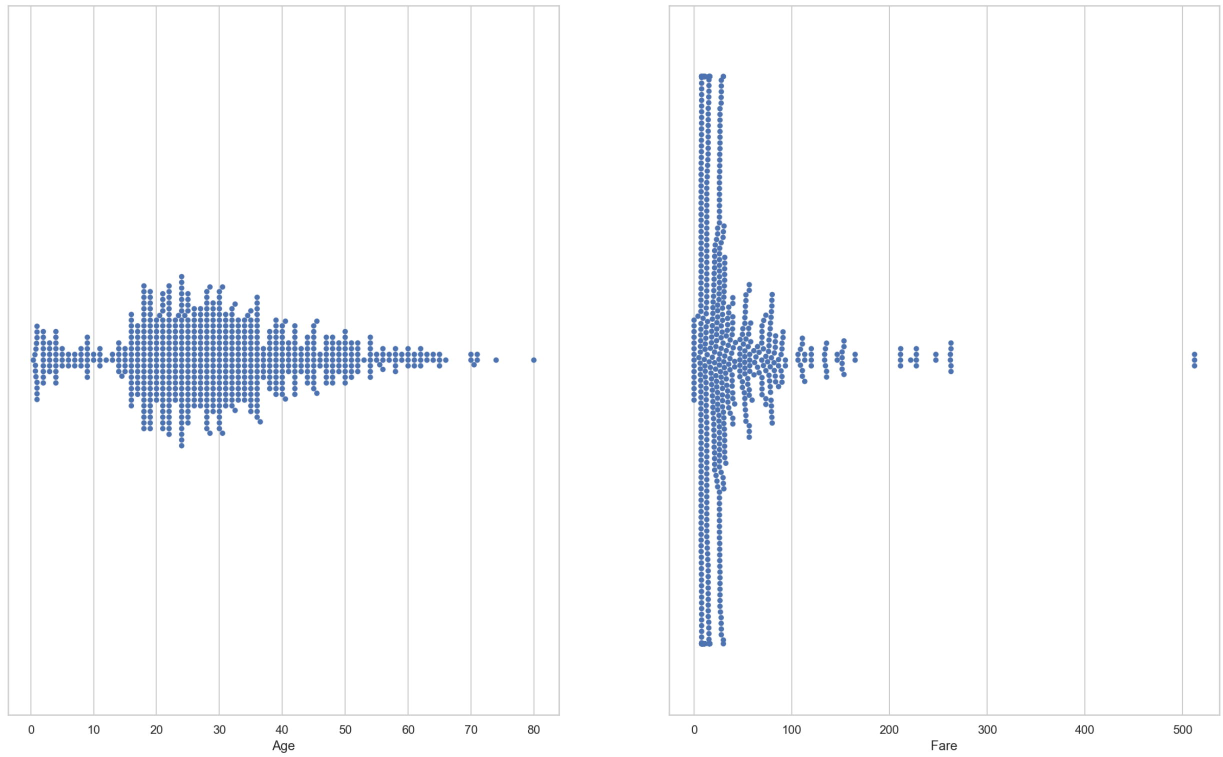
3.3 The 4 C's of Data Cleaning: Correcting, Completing, Creating, and Converting

In this stage, we will clean our data by 1) correcting bad 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.

1. **Correcting:** Reviewing the data, there does not appear to be any aberrant or non-acceptable data inputs. In addition, we see we may have potential outliers in age and fare. However, since they are reasonable values, we will wait until after we complete our exploratory analysis to determine if we should include or exclude from the dataset. It should be noted, that if they were unreasonable values, for example age = 800 instead of 80, then it's probably a safe decision to fix now. However, we want to use caution when we modify data from its original value, because it may be necessary to create an accurate model.
2. **Completing:** There are null values or missing data in the age, cabin, and embarked field. Missing values can be bad, because some algorithms don't know how-to handle null values and will fail. While others, like decision trees, can handle null values. Thus, it's important to fix before we start modeling, because we will compare and contrast several models. There are two common methods, either delete the record or populate the missing value using a reasonable input. It is not recommended to delete the record, especially a large percentage of records, unless it truly represents an incomplete record. Instead, it's best to impute missing values. A basic methodology for qualitative data is impute using mode. A basic methodology for quantitative data is impute using mean, median, or mean + randomized standard deviation. An intermediate methodology is to use the basic methodology based on specific criteria; like the average age by class or embark port by fare and SES. There are more complex methodologies, however before deploying, it should be compared to the base model to determine if complexity truly adds value. **For this dataset, age will be imputed with the median, the cabin attribute will be dropped, and embark will be imputed with mode.** Subsequent model iterations may modify this decision to determine if it improves the model’s accuracy.
3. **Creating:** Feature engineering is when we use existing features to create new features to determine if they provide new signals to predict our outcome. For this dataset, we will create a title feature to determine if it played a role in survival.
4. **Converting:** Last, but certainly not least, we'll deal with formatting. There are no date or currency formats, but datatype formats. Our categorical data imported as objects, which makes it difficult for mathematical calculations. For this dataset, we will convert object datatypes to categorical dummy variables.

In [34]:
```
import warnings
warnings.filterwarnings('ignore')

pylab.rcParams['figure.figsize'] = 20,12
sns.set(style="whitegrid")
```

In [35]:
```
fig, ax =plt.subplots(1,2)
sns.swarmplot(x=data_raw['Age'], ax=ax[0])
sns.swarmplot(x=data_raw['Fare'], ax=ax[1])
fig.show()
```



In [36]:
```
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[36]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN
top	NaN	NaN	NaN	Flynn, Mr. John	male	NaN	NaN	NaN
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000

In [49]:

```
###COMPLETING: complete or delete missing values in train and test/validation dat
for dataset in data_cleaner:
    dataset['Age'].fillna(dataset['Age'].median(), inplace = True)     #complete
    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace = True)
    dataset['Fare'].fillna(dataset['Fare'].median(), inplace = True)    #complet

    #delete the cabin feature/column and others previously stated to exclude in t
    drop_column = ['PassengerId','Cabin', 'Ticket']
    dataset.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
----------
Pclass      0
Name        0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [60]:
```
###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 i

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

    #Fare Bins using quantile-cut or frequency bins (euqal number of data points
    dataset['FareBin'] = pd.qcut(dataset['Fare'], 4) #the ouput has a Category t

    #Age Bins using cut or value bins: https://pandas.pydata.org/pandas-docs/stal
    dataset['AgeBin'] = pd.cut(dataset['Age'].astype(int), 5)

data1
```

Out[60]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	2
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	2
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	1
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	2
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1
...	...	...	...	...	...	...	...	...	...	...
886	0	2	Montvila, Rev. Juozas	male	27.0	0	0	13.0000	S	1
887	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	30.0000	S	1
888	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	28.0	1	2	23.4500	S	4

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize
889	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	30.0000	C	1
890	0	3	Dooley, Mr. Patrick	male	32.0	0	0	7.7500	Q	1

891 rows × 14 columns

sample size determination in statistics (to remove small groups of samples):

https://en.wikipedia.org/wiki/Sample_size_determination (https://en.wikipedia.org/wiki/Sample_size_determination)

https://en.wikipedia.org/wiki/Effect_size#Cohen's_d (https://en.wikipedia.org/wiki/Effect_size#Cohen's_d)

http://nicholasjjackson.com/2012/03/08/sample-size-is-10-a-magic-number/ (http://nicholasjjackson.com/2012/03/08/sample-size-is-10-a-magic-number/)

For confidence level 95% you can use the following table to determine the sample size:

Power	Cohen's d: 0.2	0.5	0.8
0.25	84	14	6
0.50	193	32	13
0.60	246	40	16
0.70	310	50	20
0.80	393	64	26
0.90	526	85	34
0.95	651	105	42
0.99	920	148	58

In [61]:
```
#cleanup rare title names
# print(data1['Title'].value_counts())
stat_min = 13
title_names = (data1['Title'].value_counts() < stat_min) #create mask

data1['Title'] = data1['Title'].apply(lambda x: 'Misc' if title_names.loc[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: 85.8+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 13 columns):
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
Fare          418 non-null float64
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(5), object(4)
memory usage: 37.3+ KB
```

Out[61]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	I
673	1	2	Wilhelms, Mr. Charles	male	31.0	0	0	13.0000	S	1	
62	0	1	Harris, Mr. Henry Birkhardt	male	45.0	1	0	83.4750	S	2	
639	0	3	Thorneycroft, Mr. Percival	male	28.0	1	0	16.1000	S	2	
626	0	2	Kirkland, Rev. Charles Leonard	male	57.0	0	0	12.3500	Q	1	
154	0	3	Olsen, Mr. Ole Martin	male	28.0	0	0	7.3125	S	1	
415	0	3	Meek, Mrs. Thomas (Annie Louise Rowley)	female	28.0	0	0	8.0500	S	1	
502	0	3	O'Sullivan, Miss. Bridget Mary	female	28.0	0	0	7.6292	Q	1	
481	0	2	Frost, Mr. Anthony Wood "Archie"	male	28.0	0	0	0.0000	S	1	
422	0	3	Zimmerman, Mr. Leo	male	29.0	0	0	7.8750	S	1	
427	1	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...	female	19.0	0	0	26.0000	S	1	

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.

** Developer Documentation: **

- [Categorical Encoding (http://pbpython.com/categorical-encoding.html)](http://pbpython.com/categorical-encoding.html)
- [Sklearn LabelEncoder (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)

- [Sklearn OneHotEncoder (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)
- [Pandas Categorical dtype (https://pandas.pydata.org/pandas-docs/stable/categorical.html)](https://pandas.pydata.org/pandas-docs/stable/categorical.html)
- [pandas.get_dummies (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)

Also, read about dummy variables trap here: [https://towardsdatascience.com/one-hot-encoding-multicollinearity-and-the-dummy-variable-trap-b5840be3c41a (https://towardsdatascience.com/one-hot-encoding-multicollinearity-and-the-dummy-variable-trap-b5840be3c41a)](https://towardsdatascience.com/one-hot-encoding-multicollinearity-and-the-dummy-variable-trap-b5840be3c41a)

In [63]:
```
#CONVERT: convert objects to category using Label Encoder for train and test/val

#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'])

data1.head()
```

Out[63]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	Is
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	2	
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	2	
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	1	
3	1	1	Futrelle, Mrs. Jacques Heath	female	35.0	1	0	53.1000	S	2	

In [193]:

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

#define x variables for original features
data1_x = ['Sex','Pclass', 'Embarked', 'Title','SibSp', 'Parch', 'Age', 'Fare',
data1_x_calc = ['Sex_Code','Pclass', 'Embarked_Code', 'Title_Code','SibSp', 'Par
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',
data1_xy_bin = Target + data1_x_bin
print('Bin X Y: ', data1_xy_bin, '\n')
```

```
Original X Y:  ['Survived', 'Sex', 'Pclass', 'Embarked', 'Title', 'SibSp', 'Par
ch', 'Age', 'Fare', 'FamilySize', 'IsAlone']

Bin X Y:  ['Survived', 'Sex_Code', 'Pclass', 'Embarked_Code', 'Title_Code', 'Fa
milySize', 'AgeBin_Code', 'FareBin_Code']
```

In [196]:

```

#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()
```

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

Out[196]:

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

In [47]:
```
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)

data1.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
----------
<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: 83.7+ KB
None
----------
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
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
None
----------
```

Out[47]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	
top	NaN	NaN	NaN	Flynn, Mr. John	male	NaN	NaN	NaN	34
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	

Split Training and Testing Data

[*sklearn's* train_test_split function (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

[*sklearn's* cross validation functions (http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation)](http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation)

In [179]:
```
#split train and test data with function defaults
#random_state -> seed or control random number generator: https://www.quora.com/W
train1_x, test1_x, train1_y, test1_y = model_selection.train_test_split(data1[da
train1_x_bin, test1_x_bin, train1_y_bin, test1_y_bin = model_selection.train_tes
train1_x_dummy, test1_x_dummy, train1_y_dummy, test1_y_dummy = model_selection.tr


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[179]:

	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 Analysis with Statistics

In [180]:

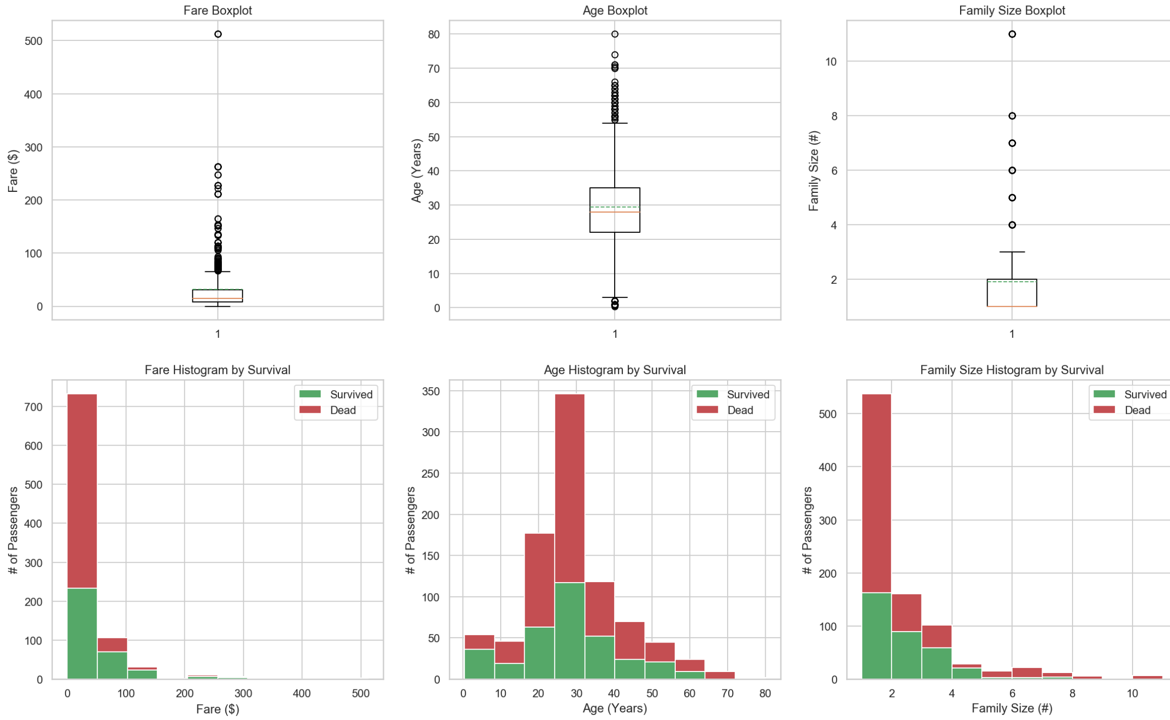
```
#Discrete Variable Correlation by Survival using
#group by aka pivot table: https://pandas.pydata.org/pandas-docs/stable/generatec
for x in data1_x:
    if data1[x].dtype != 'float64' :
        print('Survival Correlation by:', x)
        print(data1[[x, Target[0]]].groupby(x, as_index=False).mean())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


#using crosstabs: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.c
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
```

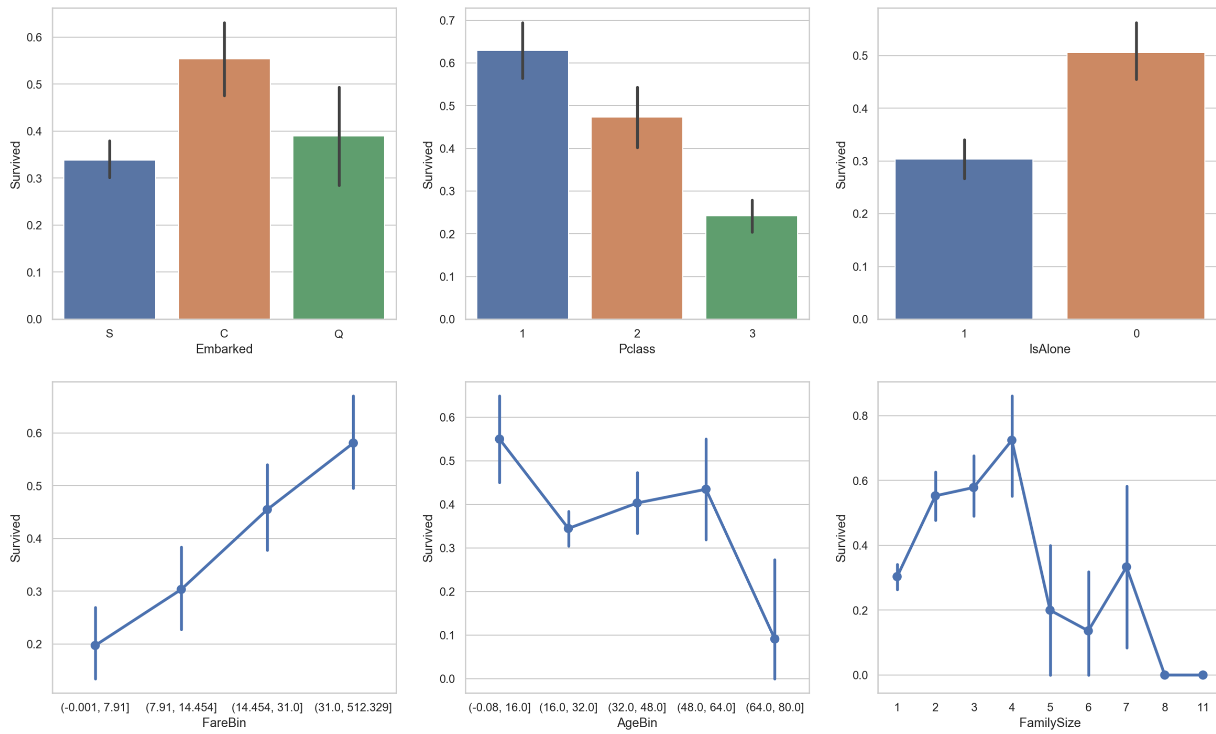
In [64]:
```
#IMPORTANT: Intentionally plotted different ways for learning purposes only.

#to organize our graphics will use figure: https://matplotlib.org/api/_as_gen/mat
#subplot: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matp
#and subplotS: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.htm

#graph distribution of quantitative data

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]['
         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]['A
         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']
         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[64]: <matplotlib.legend.Legend at 0x173e0e05548>


Fare Boxplot
Fare ($)
Age Boxplot
Age (Years)
Family Size Boxplot
Family Size (#)
Fare Histogram by Survival
Survived
Dead
# of Passengers
Fare ($)
Age Histogram by Survival
Age (Years)
Family Size Histogram by Survival
Family Size (#)

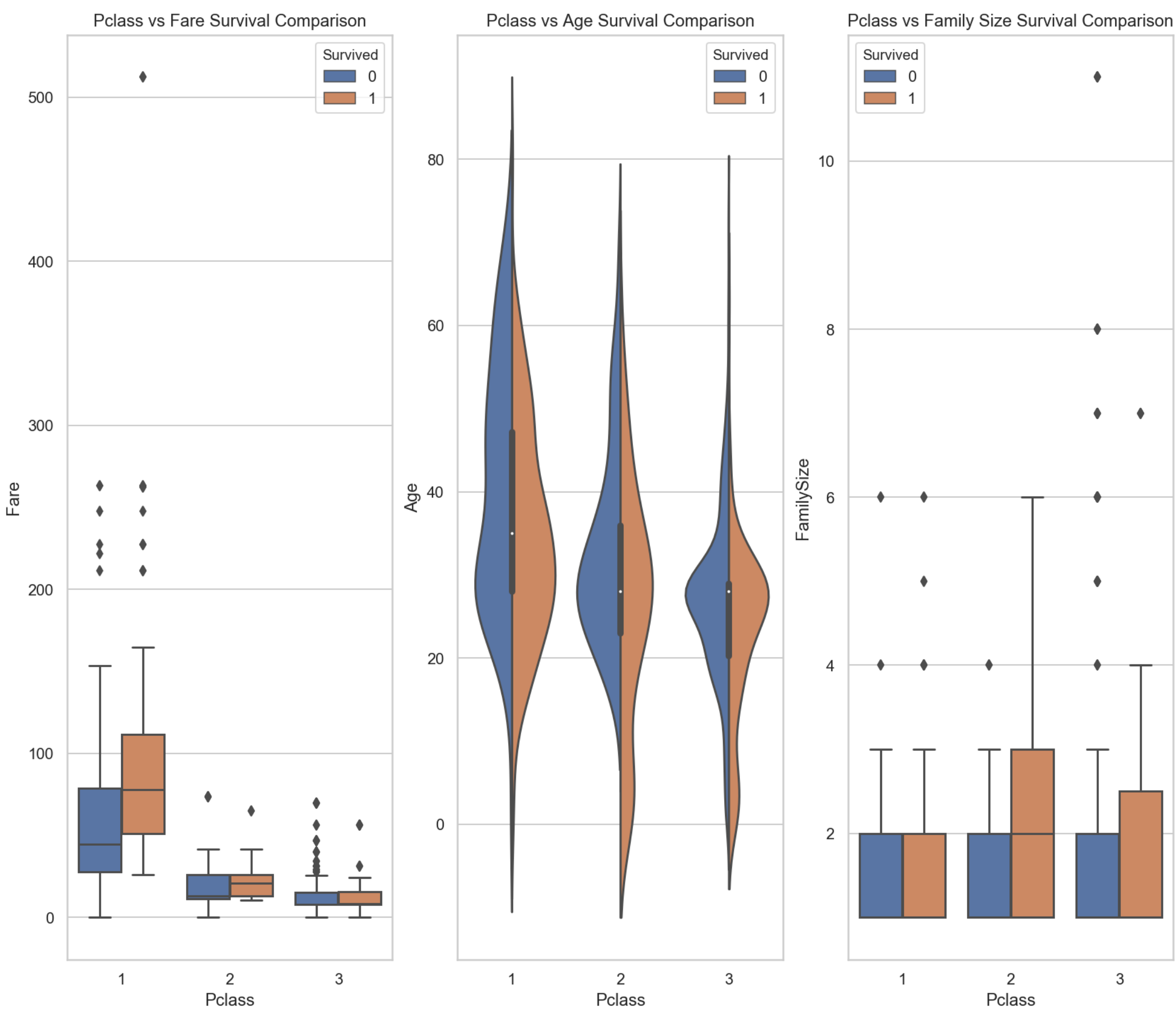
In [78]:
```
#graph individual features by survival
fig, saxis = plt.subplots(2, 3)

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[(
sns.barplot(x = 'IsAlone', y = 'Survived', order=[1,0], data=data1, ax = saxis[0,

sns.pointplot(x = 'FareBin', y = 'Survived',  data=data1, ax = saxis[1,0], ci=99
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[78]: <matplotlib.axes._subplots.AxesSubplot at 0x173df0345c8>



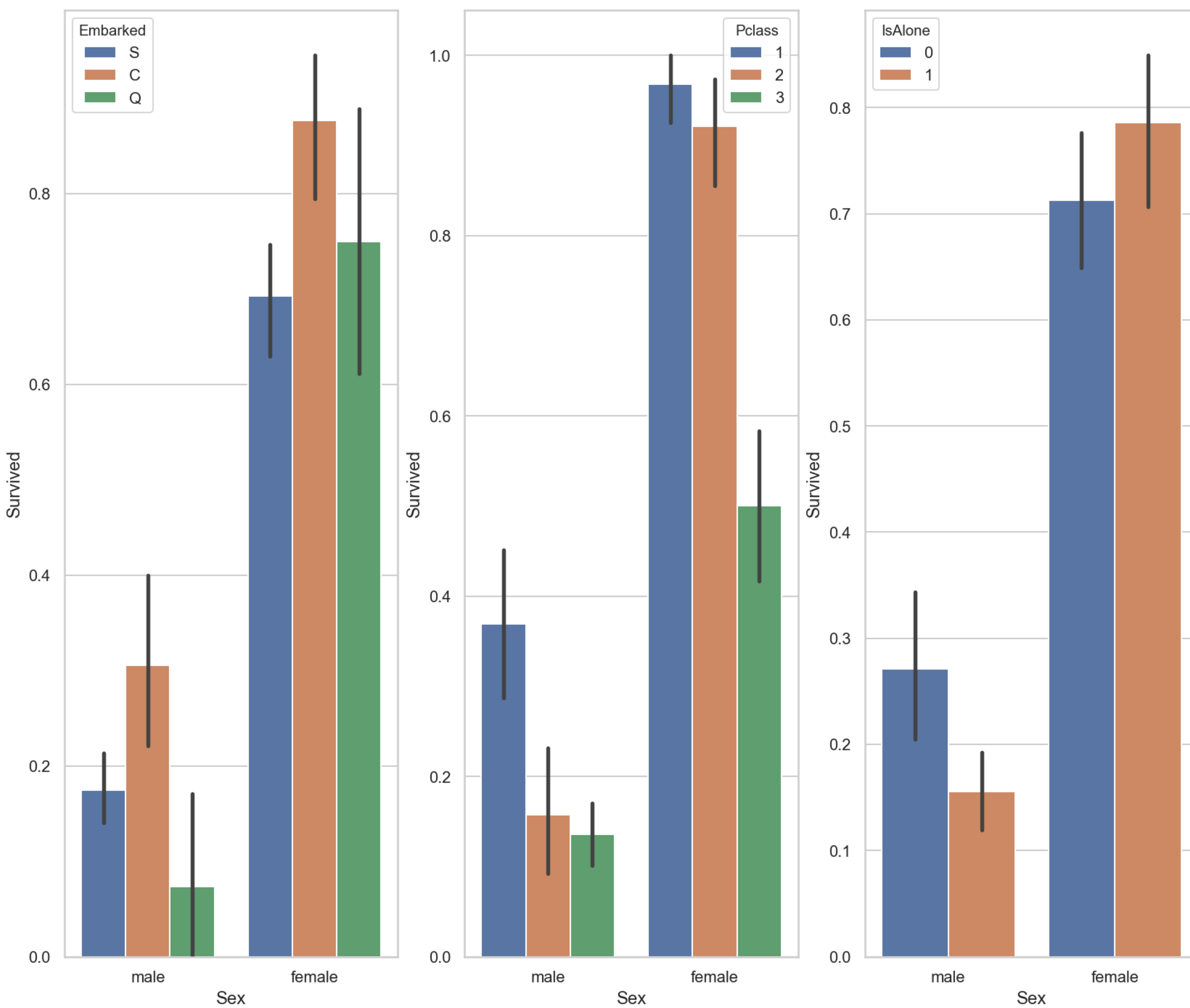
In [79]:
```
#graph distribution of qualitative data: Pclass
#we know class mattered in survival, now let's compare class and a 2nd feature
fig, (axis1,axis2,axis3) = plt.subplots(1,3,figsize=(14,12))

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 = 
axis2.set_title('Pclass vs Age Survival Comparison')

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

Out[79]: `Text(0.5, 1.0, 'Pclass vs Family Size Survival Comparison')`



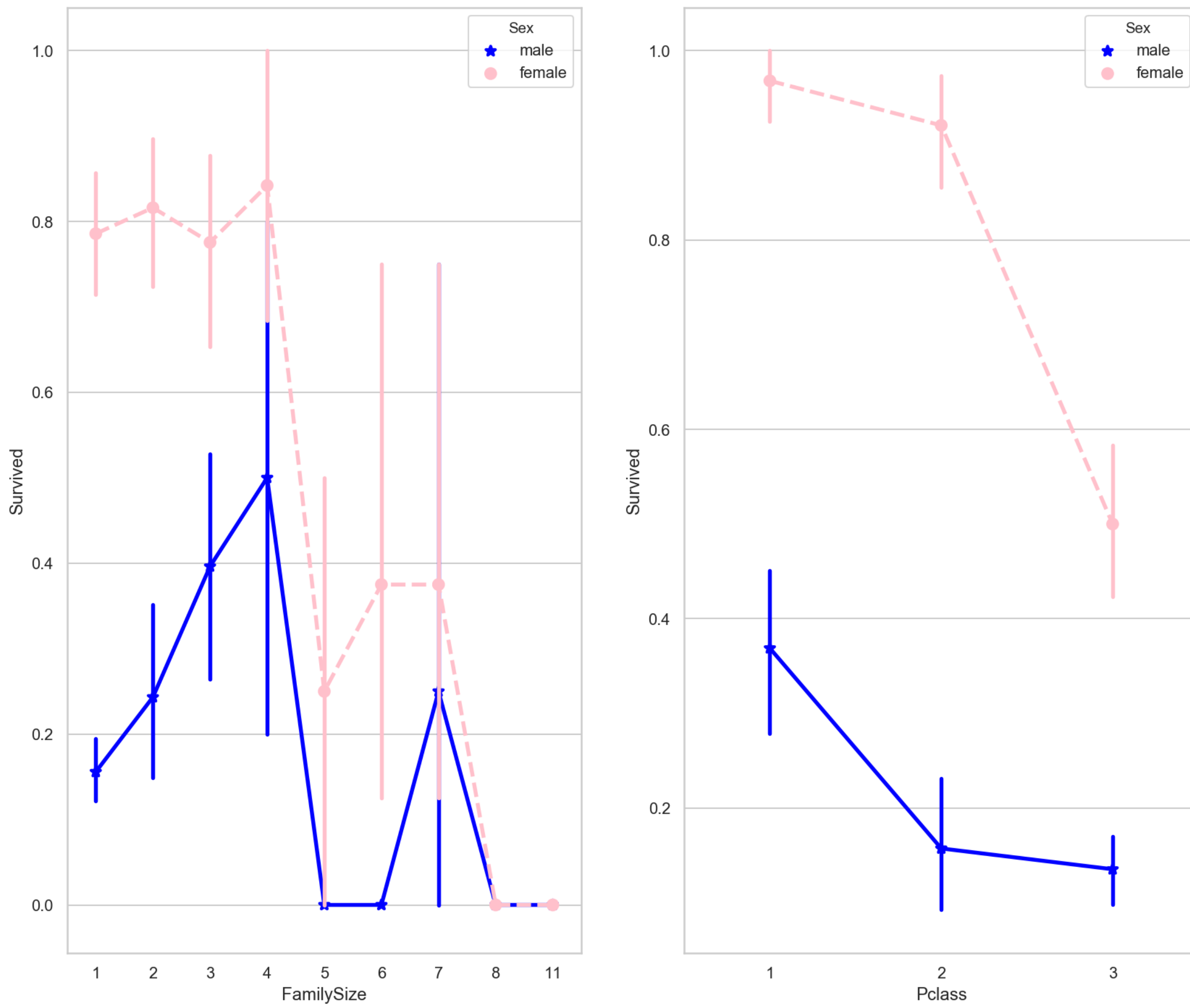
In [81]:
```
#graph distribution of qualitative data: Sex
#we know sex mattered in survival, now let's compare sex and a 2nd feature
fig, qaxis = plt.subplots(1,3,figsize=(14,12))

sns.barplot(x = 'Sex', y = 'Survived', hue = 'Embarked', data=data1, ax = qaxis[(
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[81]: Text(0.5, 1, 'Sex vs IsAlone Survival Comparison')



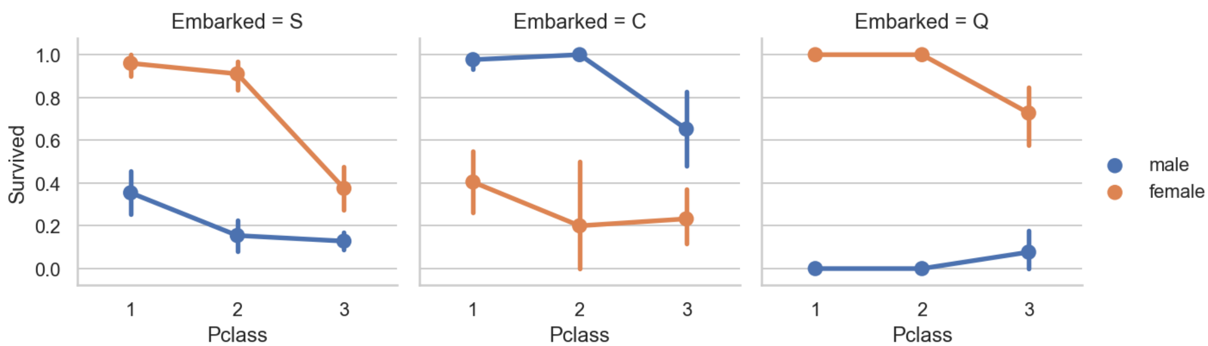
In [82]:
```
#more side-by-side comparisons
fig, (maxis1, maxis2) = plt.subplots(1, 2,figsize=(14,12))

#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[82]: <matplotlib.axes._subplots.AxesSubplot at 0x173e126b2c8>



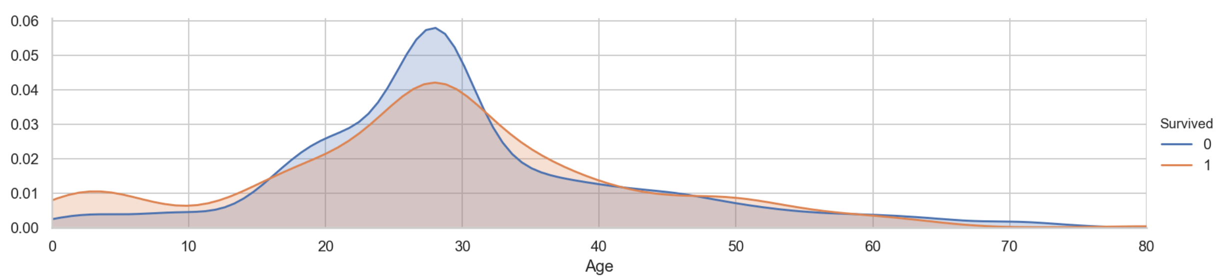
In [83]:
```
#how does embark port factor with class, sex, and survival compare
#facetgrid: https://seaborn.pydata.org/generated/seaborn.FacetGrid.html
e = sns.FacetGrid(data1, col = 'Embarked')
e.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', ci=95.0, palette = 'deep')
e.add_legend()
```

Out[83]: <seaborn.axisgrid.FacetGrid at 0x173e1334548>



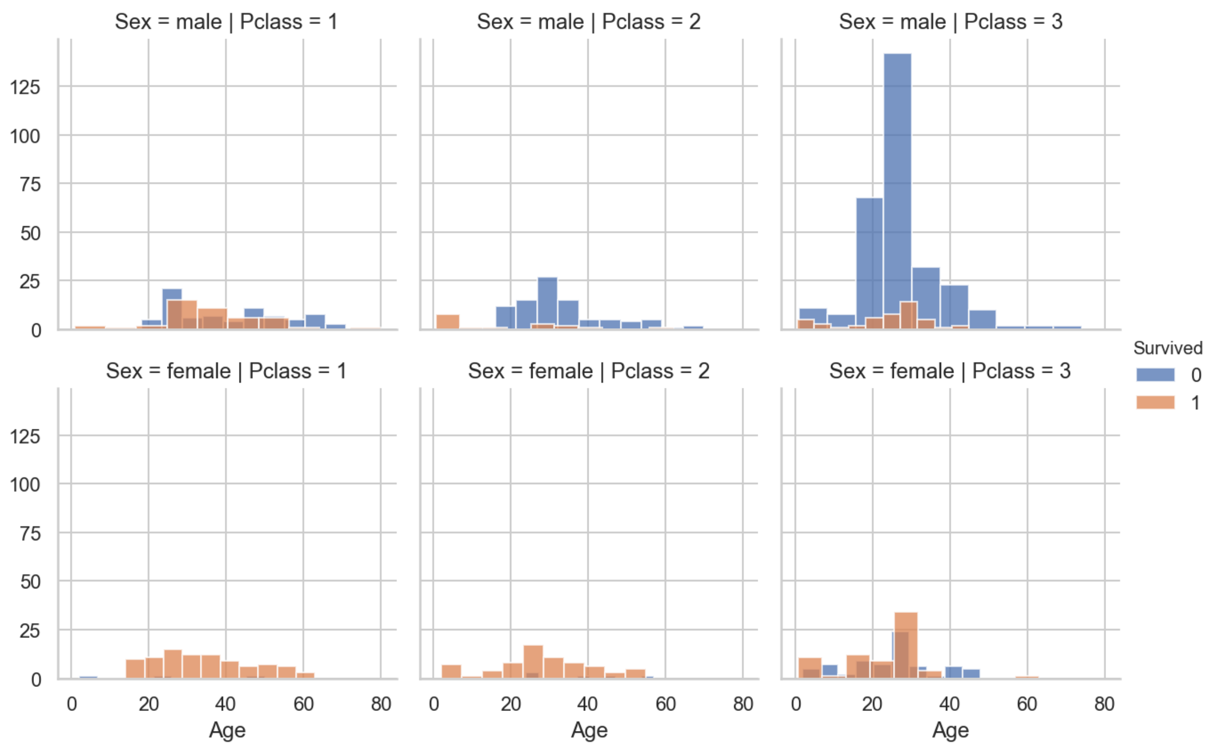
In [84]:
```
#plot distributions of age of passengers who survived or did not survive
a = sns.FacetGrid( data1, hue = 'Survived', aspect=4 )
a.map(sns.kdeplot, 'Age', shade= True )
a.set(xlim=(0 , data1['Age'].max()))
a.add_legend()
```

Out[84]: <seaborn.axisgrid.FacetGrid at 0x173e0547508>

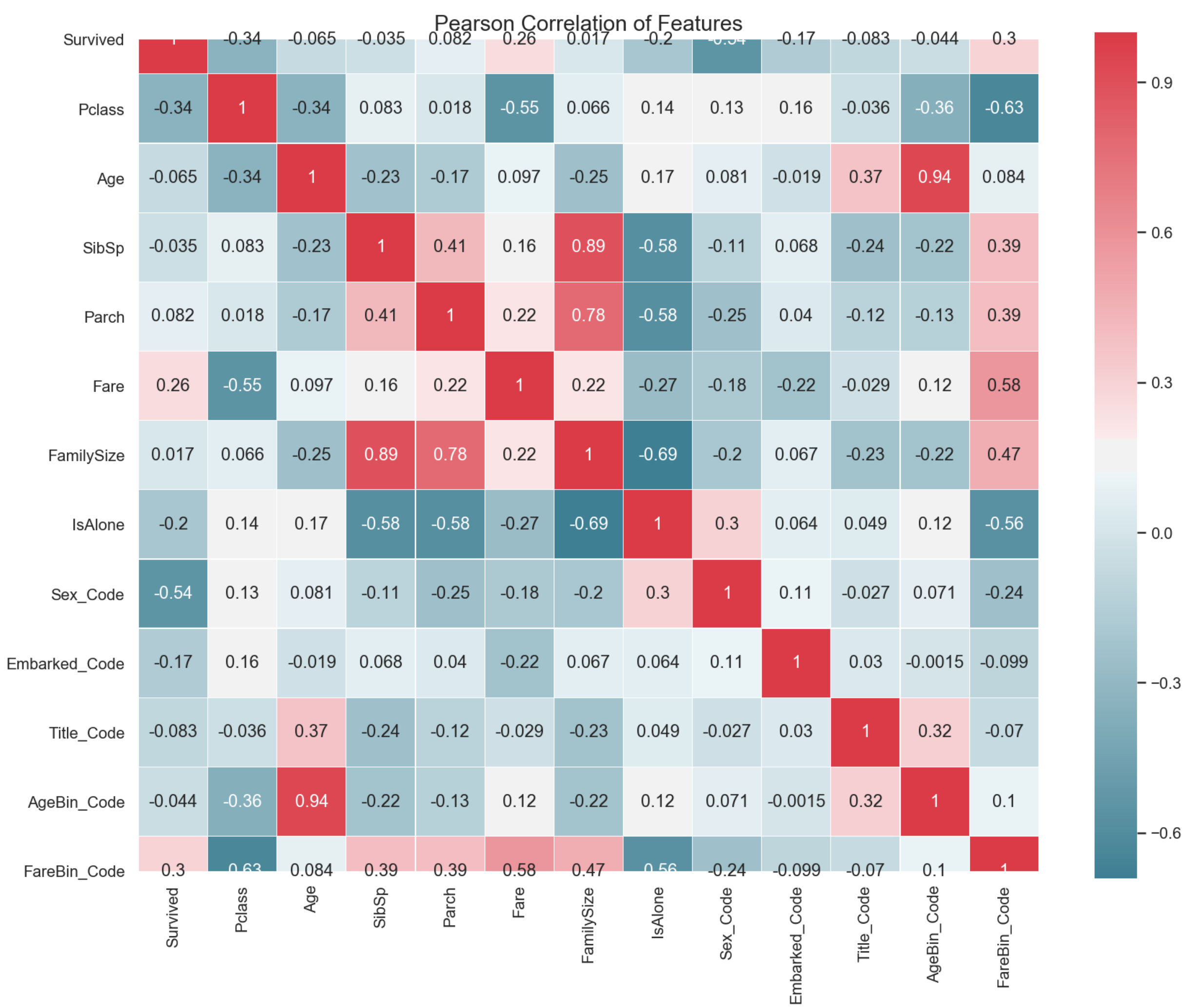


In [85]:
```
#histogram comparison of sex, class, and age by survival
h = sns.FacetGrid(data1, row = 'Sex', col = 'Pclass', hue = 'Survived')
h.map(plt.hist, 'Age', alpha = .75)
h.add_legend()
```

Out[85]: <seaborn.axisgrid.FacetGrid at 0x173e190f0c8>



In [87]:
```python
#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)
```



Step 5: Model Data

Data Science is a multi-disciplinary field between mathematics (i.e. statistics, linear algebra, etc.), computer science (i.e. programming languages, computer systems, etc.) and business management (i.e. communication, subject-matter knowledge, etc.). Most data scientist come from one of the three fields, so they tend to lean towards that discipline. However, data science is like a three-legged stool, with no one leg being more important than the other. So, this step will require advanced knowledge in mathematics. But don't worry, we only need a high-level overview, which we'll cover in this Kernel. Also, thanks to computer science, a lot of the heavy lifting is done for you. So, problems that once required graduate degrees in mathematics or statistics, now only take a few lines of code. Last, we'll need some business acumen to think through the problem. After all, like training a sight-seeing dog, it's learning from us and not the other way around.

Machine Learning (ML), as the name suggest, is teaching the machine how-to think and not what to think. While this topic and big data has been around for decades, it is becoming more popular than ever because the barrier to entry is lower, for businesses and professionals alike. This is both good and bad. It's good because these algorithms are now accessible to more people that can solve more problems in the real-world. It's bad because a lower barrier to entry means, more people will not know the tools they are using and can come to incorrect conclusions. That's why I focus on teaching you, not just what to do, but why you're doing it. Previously, I used the analogy of asking someone to hand you a Philip screwdriver, and they hand you a flathead screwdriver or worst a hammer. At best, it shows a complete lack of understanding. At worst, it makes completing the project impossible; or even worst, implements incorrect actionable intelligence. So now that I've hammered (no pun intended) my point, I'll show you what to do and most importantly, WHY you do it.

First, you must understand, that the purpose of machine learning is to solve human problems. Machine learning can be categorized as: supervised learning, unsupervised learning, and reinforced learning. Supervised learning is where you train the model by presenting it a training dataset that includes the correct answer. Unsupervised learning is where you train the model using a training dataset that does not include the correct answer. And reinforced learning is a hybrid of the previous two, where the model is not given the correct answer immediately, but later after a sequence of events to reinforce learning. We are doing supervised machine learning, because we are training our algorithm by presenting it with a set of features and their corresponding target. We then hope to present it a new subset from the same dataset and have similar results in prediction accuracy.

There are many machine learning algorithms, however they can be reduced to four categories: classification, regression, clustering, or dimensionality reduction, depending on your target variable and data modeling goals. We'll save clustering and dimension reduction for another day, and focus on classification and regression. We can generalize that a continuous target variable requires a regression algorithm and a discrete target variable requires a classification algorithm. One side note, logistic regression, while it has regression in the name, is really a classification algorithm. Since our problem is predicting if a passenger survived or did not survive, this is a discrete target variable. We will use a classification algorithm from the *sklearn* library to begin our analysis. We will use cross validation and scoring metrics, discussed in later sections, to rank and compare our algorithms' performance.

Machine Learning Selection:

- Sklearn Estimator Overview (http://scikit-learn.org/stable/user_guide.html)
- Sklearn Estimator Detail (http://scikit-learn.org/stable/modules/classes.html)
- Choosing Estimator Mind Map (http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
- Choosing Estimator Cheat Sheet (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Scikit_Learn_Cheat_Sheet_Pytl

Now that we identified our solution as a supervised learning classification algorithm. We can narrow our list of choices.

Machine Learning Classification Algorithms:

- Ensemble Methods (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble)
- Generalized Linear Models (GLM) (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)
- Naive Bayes (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes)
- Nearest Neighbors (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors)
- Support Vector Machines (SVM) (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm)
- Decision Trees (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.tree)
- Discriminant Analysis (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.discriminant_analysis)

Data Science 101: How to Choose a Machine Learning Algorithm (MLA)

IMPORTANT: When it comes to data modeling, the beginner's question is always, "what is the best machine learning algorithm?" To this the beginner must learn, the No Free Lunch Theorem (NFLT) (http://robertmarks.org/Classes/ENGR5358/Papers/NFL_4_Dummies.pdf) of Machine Learning. In short, NFLT states, there is no super algorithm, that works best in all situations, for all datasets. So the best approach is to try multiple MLAs, tune them, and compare them for your specific scenario. With that being said, some good research has been done to compare algorithms, such as Caruana & Niculescu-Mizil 2006 (https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf) watch video lecture here (http://videolectures.net/solomon_caruana_wslmw/) of MLA comparisons, Ogutu et al. 2011 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3103196/) done by the NIH for genomic selection, Fernandez-Delgado et al. 2014 (http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf) comparing 179 classifiers from 17 families, Thoma 2016 sklearn comparison (https://martin-thoma.com/comparing-classifiers/), and there is also a school of thought that says, more data beats a better algorithm (https://www.kdnuggets.com/2015/06/machine-learning-more-data-better-algorithms.html).

So with all this information, where is a beginner to start? I recommend starting with [Trees, Bagging, Random Forests, and Boosting (http://jessica2.msri.org/attachments/10778/10778-boost.pdf)](http://jessica2.msri.org/attachments/10778/10778-boost.pdf). They are basically different implementations of a decision tree, which is the easiest concept to learn and understand. They are also easier to tune, discussed in the next section, than something like SVC. Below, I'll give an overview of how-to run and compare several MLAs, but the rest of this Kernel will focus on learning data modeling via decision trees and its derivatives.

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(),

    #Navies 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: http://xgboost.readthedocs.io/en/latest/model.html
    XGBClassifier()
    ]



#split dataset in cross-validation with this splitter class: http://scikit-learn
#note: this is an alternative to train_test_split
cv_split = model_selection.ShuffleSplit(n_splits = 10, test_size = .3, train_siz

#create table to compare MLA metrics
MLA_columns = ['MLA Name', 'MLA Parameters','MLA Train Accuracy Mean', 'MLA Test
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: http://scikit-learn.org/stable/modules/ge
    cv_results = model_selection.cross_validate(alg, data1[data1_x_bin], data1[Ta

    MLA_compare.loc[row_index, 'MLA Time'] = cv_results['fit_time'].mean()
    MLA_compare.loc[row_index, 'MLA Train Accuracy Mean'] = cv_results['train_sco
    MLA_compare.loc[row_index, 'MLA Test Accuracy Mean'] = cv_results['test_score
    #if this is a non-bias random sample, then +/-3 standard deviations (std) fro
    MLA_compare.loc[row_index, 'MLA Test Accuracy 3*STD'] = cv_results['test_scor


    #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: https://pandas.pydata.org/pandas-docs/stable/generated/pan
MLA_compare.sort_values(by = ['MLA Test Accuracy Mean'], ascending = False, inpla
MLA_compare
#MLA_predict
```

...

In [22]:

```
#barplot using https://seaborn.pydata.org/generated/seaborn.barplot.html
sns.barplot(x='MLA Test Accuracy Mean', y = 'MLA Name', data = MLA_compare, colo

#prettify using pyplot: https://matplotlib.org/api/pyplot_api.html
plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

...

5.1 Evaluate Model Performance

Let's recap, with some basic data cleaning, analysis, and machine learning algorithms (MLA), we are able to predict passenger survival with ~82% accuracy. Not bad for a few lines of code. But the question we always ask is, can we do better and more importantly get an ROI (return on investment) for our time invested? For example, if we're only going to increase our accuracy by

1/10th of a percent, is it really worth 3-months of development. If you work in research maybe the answer is yes, but if you work in business mostly the answer is no. So, keep that in mind when improving your model.

Data Science 101: Determine a Baseline Accuracy

Before we decide how-to make our model better, let's determine if our model is even worth keeping. To do that, we have to go back to the basics of data science 101. We know this is a binary problem, because there are only two possible outcomes; passengers survived or died. So, think of it like a coin flip problem. If you have a fair coin and you guessed heads or tail, then you have a 50-50 chance of guessing correct. So, let's set 50% as the worst model performance; because anything lower than that, then why do I need you when I can just flip a coin?

Okay, so with no information about the dataset, we can always get 50% with a binary problem. But we have information about the dataset, so we should be able to do better. We know that 1,502/2,224 or 67.5% of people died. Therefore, if we just predict the most frequent occurrence, that 100% of people died, then we would be right 67.5% of the time. So, let's set 68% as bad model performance, because again, anything lower than that, then why do I need you, when I can just predict using the most frequent occurrence.

Data Science 101: How-to Create Your Own Model

Our accuracy is increasing, but can we do better? Are there any signals in our data? To illustrate this, we're going to build our own decision tree model, because it is the easiest to conceptualize and requires simple addition and multiplication calculations. When creating a decision tree, you want to ask questions that segment your target response, placing the survived/1 and dead/0 into homogeneous subgroups. This is part science and part art, so let's just play the 21-question game to show you how it works. If you want to follow along on your own, download the train dataset and import into Excel. Create a pivot table with survival in the columns, count and % of row count in the values, and the features described below in the rows.

Remember, the name of the game is to create subgroups using a decision tree model to get survived/1 in one bucket and dead/0 in another bucket. Our rule of thumb will be the majority rules. Meaning, if the majority or 50% or more survived, then everybody in our subgroup survived/1, but if 50% or less survived then if everybody in our subgroup died/0. Also, we will stop if the subgroup is less than 10 and/or our model accuracy plateaus or decreases. Got it? Let's go!

Question 1: Were you on the Titanic? If Yes, then majority (62%) died. Note our sample survival is different than our population of 68%. Nonetheless, if we assumed everybody died, our sample accuracy is 62%.

Question 2: Are you male or female? Male, majority (81%) died. Female, majority (74%) survived. Giving us an accuracy of 79%.

Question 3A (going down the female branch with count = 314): Are you in class 1, 2, or 3? Class 1, majority (97%) survived and Class 2, majority (92%) survived. Since the dead subgroup is less than 10, we will stop going down this branch. Class 3, is even at a 50-50 split. No new information to improve our model is gained.

Question 4A (going down the female class 3 branch with count = 144): Did you embark from port C, Q, or S? We gain a little information. C and Q, the majority still survived, so no change. Also, the dead subgroup is less than 10, so we will stop. S, the majority (63%) died. So, we will change females, class 3, embarked S from assuming they survived, to assuming they died. Our model accuracy increases to 81%.

Question 5A (going down the female class 3 embarked S branch with count = 88): So far, it looks like we made good decisions. Adding another level does not seem to gain much more information. This subgroup 55 died and 33 survived, since majority died we need to find a signal to identify the 33 or a subgroup to change them from dead to survived and improve our model accuracy. We can play with our features. One I found was fare 0-8, majority survived. It's a small sample size 11-9, but one often used in statistics. We slightly improve our accuracy, but not much to move us past 82%. So, we'll stop here.

Question 3B (going down the male branch with count = 577): Going back to question 2, we know the majority of males died. So, we are looking for a feature that identifies a subgroup that majority survived. Surprisingly, class or even embarked didn't matter like it did for females, but title does and gets us to 82%. Guess and checking other features, none seem to push us past 82%. So, we'll stop here for now.

You did it, with very little information, we get to 82% accuracy. On a worst, bad, good, better, and best scale, we'll set 82% to good, since it's a simple model that yields us decent results. But the question still remains, can we do better than our handmade model?

Before we do, let's code what we just wrote above. Please note, this is a manual process created by "hand." You won't have to do this, but it's important to understand it before you start working with MLA. Think of MLA like a TI-89 calculator on a Calculus Exam. It's very powerful and helps you with a lot of the grunt work. But if you don't know what you're doing on the exam, a calculator, even a TI-89, is not going to help you pass. So, study the next section wisely.

Reference: [Cross-Validation and Decision Tree Tutorial (http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/tutorial3_CrossVal-DTs.pdf)](http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/tutorial3_CrossVal-DTs.pdf)

In [23]:
```
#IMPORTANT: This is a handmade model for learning purposes only.
#However, it is possible to create your own predictive model without a fancy algo

#coin flip model with random 1/survived 0/died

#iterate over dataFrame rows as (index, Series) pairs: https://pandas.pydata.org/
for index, row in data1.iterrows():
    #random number generator: https://docs.python.org/2/library/random.html
    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 #sc
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
#http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score
print('Coin Flip Model Accuracy w/SciKit: {:.2f}%'.format(metrics.accuracy_score
```

...

In [24]:
```
#group by or pivot table: https://pandas.pydata.org/pandas-docs/stable/generated/
pivot_female = data1[data1.Sex=='female'].groupby(['Sex','Pclass', 'Embarked','Fa
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)
```

...

In [25]:
```
#handmade data model using brain power (and Microsoft Excel Pivot Tables for quic
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 informat

        #Question 5B Female - FareBin; set anything less than .5 in female node d
        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
        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.ac


#Accuracy Summary Report with http://scikit-learn.org/stable/modules/generated/sk
#Where recall score = (true positives)/(true positive + false negative) w/1 being
#And F1 score = weighted average of precision and recall w/1 being best: http://s
print(metrics.classification_report(data1['Survived'], Tree_Predict))
```

...

In [26]:
```
#Plot Accuracy Summary
#Credit: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confu
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')
```

...

5.11 Model Performance with Cross-Validation (CV)

In step 5.0, we used sklearn cross_validate (http://scikit-learn.org/stable/modules/cross_validation.html#multimetric-cross-validation) function to train, test, and score our model performance.

Remember, 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. It's like cheating on a school quiz to get 100%, but then when you go to take the exam, you fail because you never truly learned anything. The same is true with machine learning.

CV 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. This is helpful in a Kaggle Competition or any use case where consistency matters and surprises should be avoided.

In addition to CV, we used a customized sklearn train test splitter (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection), to allow a little more randomness in our test scoring. Below is an image of the default CV split.

CV

5.12 Tune Model with Hyper-Parameters

When we used sklearn Decision Tree (DT) Classifier (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTr
we accepted all the function defaults. This leaves opportunity to see how various hyper-parameter settings will change the model accuracy. (Click here to learn more about parameters vs hyper-parameters.) (https://www.youtube.com/watch?v=EJtTNboTsm8)

However, in order to tune a model, we need to actually understand it. That's why I took the time in the previous sections to show you how predictions work. Now let's learn a little bit more about our DT algorithm.

Credit: sklearn (http://scikit-learn.org/stable/modules/tree.html#classification)

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Below are available hyper-parameters and defintions (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTr

> class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

We will tune our model using ParameterGrid (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html#sklearn.model_sel GridSearchCV (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_sel and customized sklearn scoring (http://scikit-learn.org/stable/modules/model_evaluation.html); click here to learn more about ROC_AUC scores (http://www.dataschool.io/roc-curves-and-auc-explained/). We will then visualize our tree with graphviz (http://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html#sklearn.tree.export_graphviz) Click here to learn more about ROC_AUC scores (http://www.dataschool.io/roc-curves-and-auc-explained/).

In [27]:
```
#base model
dtree = tree.DecisionTreeClassifier(random_state = 0)
base_results = model_selection.cross_validate(dtree, data1[data1_x_bin], data1[Ta
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_s
print("BEFORE DT Test w/bin score mean: {:.2f}". format(base_results['test_score
print("BEFORE DT Test w/bin score 3*std: +/- {:.2f}". format(base_results['test_s
#print("BEFORE DT Test w/bin set score min: {:.2f}". format(base_results['test_sc
print('-'*10)


#tune hyper-parameters: http://scikit-learn.org/stable/modules/generated/sklearn
param_grid = {'criterion': ['gini', 'entropy'],  #scoring methodology; two suppor
              #'splitter': ['best', 'random'], #splitting methodology; two suppor
              'max_depth': [2,4,6,8,10,None], #max depth tree can grow; default i
              #'min_samples_split': [2,5,10,.03,.05], #minimum subset size BEFORE
              #'min_samples_leaf': [1,5,10,.03,.05], #minimum subset size AFTER n
              #'max_features': [None, 'auto'], #max features to consider when per
              'random_state': [0] #seed or control random number generator: https
             }

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

#choose best model with grid_search: #http://scikit-learn.org/stable/modules/grid
#http://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_di
tune_model = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_gr
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_
#print(tune_model.cv_results_['mean_test_score'])
print("AFTER DT Test w/bin score mean: {:.2f}". format(tune_model.cv_results_['me
print("AFTER DT Test w/bin score 3*std: +/- {:.2f}". format(tune_model.cv_results
print('-'*10)


#duplicates gridsearchcv
#tune_results = model_selection.cross_validate(tune_model, data1[data1_x_bin], da

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

...

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. [Sklearn (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_selection)](http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_selection) has several options, we will use [recursive feature elimination (RFE) with cross validation (CV) (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html#sklearn.feature_selectior](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html#sklearn.feature_selection)

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['tra
print("BEFORE DT RFE Test w/bin score mean: {:.2f}". format(base_results['test_sc
print("BEFORE DT RFE Test w/bin score 3*std: +/- {:.2f}". format(base_results['te
print('-'*10)



#feature selection
dtree_rfe = feature_selection.RFECV(dtree, step = 1, scoring = 'accuracy', cv = c
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: http://scikit-l
X_rfe = data1[data1_x_bin].columns.values[dtree_rfe.get_support()]
rfe_results = model_selection.cross_validate(dtree, data1[X_rfe], data1[Target],

#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['trair
print("AFTER DT RFE Test w/bin score mean: {:.2f}". format(rfe_results['test_scor
print("AFTER DT RFE Test w/bin score 3*std: +/- {:.2f}". format(rfe_results['test
print('-'*10)


#tune rfe model
rfe_tune_model = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), para
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_moc
#print(rfe_tune_model.cv_results_['mean_test_score'])
print("AFTER DT RFE Tuned Test w/bin score mean: {:.2f}". format(rfe_tune_model.c
print("AFTER DT RFE Tuned Test w/bin score 3*std: +/- {:.2f}". format(rfe_tune_mc
print('-'*10)
```

. . .

In [29]:
```
#Graph MLA version of Decision Tree: http://scikit-learn.org/stable/modules/gener
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
```

...

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 =
#there are some 1's, but enough blues and light reds to create a "super algorithm
correlation_heatmap(MLA_predict)
```

...

In [31]:
```
#why choose one model, when you can pick them all with voting classifier
#http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassif
#removed models w/o attribute 'predict_proba' required for vote classifier and mo
vote_est = [
    #Ensemble Methods: http://scikit-learn.org/stable/modules/ensemble.html
    ('ada', ensemble.AdaBoostClassifier()),
    ('bc', ensemble.BaggingClassifier()),
    ('etc',ensemble.ExtraTreesClassifier()),
    ('gbc', ensemble.GradientBoostingClassifier()),
    ('rfc', ensemble.RandomForestClassifier()),

    #Gaussian Processes: http://scikit-learn.org/stable/modules/gaussian_process
    ('gpc', gaussian_process.GaussianProcessClassifier()),

    #GLM: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
    ('lr', linear_model.LogisticRegressionCV()),

    #Navies Bayes: http://scikit-learn.org/stable/modules/naive_bayes.html
    ('bnb', naive_bayes.BernoulliNB()),
    ('gnb', naive_bayes.GaussianNB()),

    #Nearest Neighbor: http://scikit-learn.org/stable/modules/neighbors.html
    ('knn', neighbors.KNeighborsClassifier()),

    #SVM: http://scikit-learn.org/stable/modules/svm.html
    ('svc', svm.SVC(probability=True)),

    #xgboost: http://xgboost.readthedocs.io/en/latest/model.html
   ('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], data
vote_hard.fit(data1[data1_x_bin], data1[Target])

print("Hard Voting Training w/bin score mean: {:.2f}". format(vote_hard_cv['trai
print("Hard Voting Test w/bin score mean: {:.2f}". format(vote_hard_cv['test_sco
print("Hard Voting Test w/bin score 3*std: +/- {:.2f}". format(vote_hard_cv['tes
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], data
vote_soft.fit(data1[data1_x_bin], data1[Target])

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

...

In [32]:
```
#IMPORTANT: THIS SECTION IS UNDER CONSTRUCTION!!!! 12.24.17
#UPDATE: This section was scrapped for the next section; as it's more computatio

#WARNING: Running is very computational intensive and time expensive
#code is written for experimental/developmental purposes and not production read


#tune each estimator before creating a super model
#http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe
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 = [{
#             #http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
            'ada__n_estimators': grid_n_estimator,
            'ada__learning_rate': grid_ratio,
            'ada__algorithm': ['SAMME', 'SAMME.R'],
            'ada__random_state': grid_seed,

            #http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.B
            'bc__n_estimators': grid_n_estimator,
            'bc__max_samples': grid_ratio,
            'bc__oob_score': grid_bool,
            'bc__random_state': grid_seed,

            #http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.E
            'etc__n_estimators': grid_n_estimator,
            'etc__criterion': grid_criterion,
            'etc__max_depth': grid_max_depth,
            'etc__random_state': grid_seed,


            #http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.G
            '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,

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

            #http://scikit-learn.org/stable/modules/generated/sklearn.naive_baye
            'bnb__alpha': grid_ratio,
            'bnb__prior': grid_bool,
            'bnb__random_state': grid_seed,

            #http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.
            '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,

            #http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.htm
            #http://blog.hackerearth.com/simple-tutorial-svm-parameter-tuning-py
            '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,


            #http://xgboost.readthedocs.io/en/latest/parameter.html
            '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 = vo
#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
#print(grid_soft.cv_results_['mean_test_score'])
#print("Soft Vote Tuned Test w/bin set score mean: {:.2f}". format(grid_soft.cv_
#print("Soft Vote Tuned Test w/bin score 3*std: +/- {:.2f}". format(grid_soft.cv
#print('-'*10)


#credit: https://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClass
```

```
#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]:
```
#WARNING: Running is very computational intensive and time expensive.
#Code is written for experimental/developmental purposes and not production ready


#Hyperparameter Tune with GridSearchCV: http://scikit-learn.org/stable/modules/ge
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 - http://scikit-learn.org/stable/modules/generate
            '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 - http://scikit-learn.org/stable/modules/generated
            'n_estimators': grid_n_estimator, #default=10
            'max_samples': grid_ratio, #default=1.0
            'random_state': grid_seed
             }],


            [{
            #ExtraTreesClassifier - http://scikit-learn.org/stable/modules/genera
            'n_estimators': grid_n_estimator, #default=10
            'criterion': grid_criterion, #default=”gini”
            'max_depth': grid_max_depth, #default=None
            'random_state': grid_seed
             }],


            [{
            #GradientBoostingClassifier - http://scikit-learn.org/stable/modules/
            #'loss': ['deviance', 'exponential'], #default=’deviance’
            'learning_rate': [.05], #default=0.1 -- 12/31/17 set to reduce runtin
            'n_estimators': [300], #default=100 -- 12/31/17 set to reduce runtime
            #'criterion': ['friedman_mse', 'mse', 'mae'], #default=”friedman_mse
            'max_depth': grid_max_depth, #default=3
            'random_state': grid_seed
             }],


            [{
            #RandomForestClassifier - http://scikit-learn.org/stable/modules/gene
            '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 runtim
'random_state': grid_seed
 }],

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


[{
#LogisticRegressionCV - http://scikit-learn.org/stable/modules/gener
'fit_intercept': grid_bool, #default: True
#'penalty': ['l1','l2'],
'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], #defau
'random_state': grid_seed
 }],


[{
#BernoulliNB - http://scikit-learn.org/stable/modules/generated/skle
'alpha': grid_ratio, #default: 1.0
 }],


#GaussianNB -
[{}],

[{
#KNeighborsClassifier - http://scikit-learn.org/stable/modules/gener
'n_neighbors': [1,2,3,4,5,6,7], #default: 5
'weights': ['uniform', 'distance'], #default = 'uniform'
'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}],


[{
#SVC - http://scikit-learn.org/stable/modules/generated/sklearn.svm.
#http://blog.hackerearth.com/simple-tutorial-svm-parameter-tuning-py
#'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 - http://xgboost.readthedocs.io/en/latest/parameter.h
'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() #https://docs.python.org/3/library/time.html#t
for clf, param in zip (vote_est, grid_param): #https://docs.python.org/3/library/

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


    start = time.perf_counter()
    best_search = model_selection.GridSearchCV(estimator = clf[1], param_grid = p
    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.'.for
    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)
```

...

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], dat
grid_hard.fit(data1[data1_x_bin], data1[Target])

print("Hard Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". fo
print("Hard Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". forma
print("Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". 
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], dat
grid_soft.fit(data1[data1_x_bin], data1[Target])

print("Soft Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". fo
print("Soft Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". forma
print("Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". 
print('-'*10)


#12/31/17 tuned with data1_x_bin
#The best parameter for AdaBoostClassifier is {'learning_rate': 0.1, 'n_estimato
#The best parameter for BaggingClassifier is {'max_samples': 0.25, 'n_estimators
#The best parameter for ExtraTreesClassifier is {'criterion': 'entropy', 'max_de
#The best parameter for GradientBoostingClassifier is {'learning_rate': 0.05, 'm
#The best parameter for RandomForestClassifier is {'criterion': 'entropy', 'max_
#The best parameter for GaussianProcessClassifier is {'max_iter_predict': 10, 'r
#The best parameter for LogisticRegressionCV is {'fit_intercept': True, 'random_
#The best parameter for BernoulliNB is {'alpha': 0.1} with a runtime of 0.19 sec
#The best parameter for GaussianNB is {} with a runtime of 0.04 seconds.
#The best parameter for KNeighborsClassifier is {'algorithm': 'brute', 'n_neighb
#The best parameter for SVC is {'C': 2, 'decision_function_shape': 'ovo', 'gamma
#The best parameter for XGBClassifier is {'learning_rate': 0.01, 'max_depth': 4,
#Total optimization time was 5.56 minutes.
```

...

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
#submit_dt = tree.DecisionTreeClassifier()
#submit_dt = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_gr
#submit_dt.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_dt.best_params_) #Best Parameters:  {'criteri
#data_val['Survived'] = submit_dt.predict(data_val[data1_x_bin])


#bagging w/full dataset modeling submission score: defaults= 0.75119, tuned= 0.77
#submit_bc = ensemble.BaggingClassifier()
#submit_bc = model_selection.GridSearchCV(ensemble.BaggingClassifier(), param_gr
#submit_bc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_bc.best_params_) #Best Parameters:  {'max_samp
#data_val['Survived'] = submit_bc.predict(data_val[data1_x_bin])


#extra tree w/full dataset modeling submission score: defaults= 0.76555, tuned= 0
#submit_etc = ensemble.ExtraTreesClassifier()
#submit_etc = model_selection.GridSearchCV(ensemble.ExtraTreesClassifier(), param
#submit_etc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_etc.best_params_) #Best Parameters:  {'criter
#data_val['Survived'] = submit_etc.predict(data_val[data1_x_bin])


#random foreset w/full dataset modeling submission score: defaults= 0.71291, tune
#submit_rfc = ensemble.RandomForestClassifier()
#submit_rfc = model_selection.GridSearchCV(ensemble.RandomForestClassifier(), par
#submit_rfc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_rfc.best_params_) #Best Parameters:  {'criter
#data_val['Survived'] = submit_rfc.predict(data_val[data1_x_bin])



#ada boosting w/full dataset modeling submission score: defaults= 0.74162, tuned=
#submit_abc = ensemble.AdaBoostClassifier()
#submit_abc = model_selection.GridSearchCV(ensemble.AdaBoostClassifier(), param_g
#submit_abc.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_abc.best_params_) #Best Parameters:  {'algori
#data_val['Survived'] = submit_abc.predict(data_val[data1_x_bin])


#gradient boosting w/full dataset modeling submission score: defaults= 0.75119, t
#submit_gbc = ensemble.GradientBoostingClassifier()
#submit_gbc = model_selection.GridSearchCV(ensemble.GradientBoostingClassifier(),
#submit_gbc.fit(data1[data1_x_bin], data1[Target])
```

```
#print('Best Parameters: ', submit_gbc.best_params_) #Best Parameters:  {'learnin
#data_val['Survived'] = submit_gbc.predict(data_val[data1_x_bin])

#extreme boosting w/full dataset modeling submission score: defaults= 0.73684, tu
#submit_xgb = XGBClassifier()
#submit_xgb = model_selection.GridSearchCV(XGBClassifier(), param_grid= {'learnin
#submit_xgb.fit(data1[data1_x_bin], data1[Target])
#print('Best Parameters: ', submit_xgb.best_params_) #Best Parameters:  {'learnin
#data_val['Survived'] = submit_xgb.predict(data_val[data1_x_bin])


#hard voting classifier w/full dataset modeling submission score: defaults= 0.75
#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= 0.73
#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(norma
submit.sample(10)
```

...

Step 7: Optimize and Strategize

Conclusion

Iteration one of the Data Science Framework, seems to converge on 0.77990 submission accuracy. Using the same dataset and different implementation of a decision tree (adaboost, random forest, gradient boost, xgboost, etc.) with tuning does not exceed the 0.77990 submission accuracy. Interesting for this dataset, the simple decision tree algorithm had the best default submission score and with tuning achieved the same best accuracy score.

While no general conclusions can be made from testing a handful of algorithms on a single dataset, there are several observations on the mentioned dataset.

1. The train dataset has a different distribution than the test/validation dataset and population. This created wide margins between the cross validation (CV) accuracy score and Kaggle submission accuracy score.
2. Given the same dataset, decision tree based algorithms, seemed to converge on the same accuracy score after proper tuning.
3. Despite tuning, no machine learning algorithm, exceeded the homemade algorithm. The author will theorize, that for small datasets, a manmade algorithm is the bar to beat.