This notebook was prepared by Donne Martin (http://donnemartin.com). Source and license info is on GitHub (https://github.com/donnemartin/data-science-ipython-notebooks).

Kaggle Machine Learning Competition: Predicting Titanic Survivors

Competition Site

Description, Evaluation, and Data Set taken from the competition site (https://www.kaggle.com/c/titanic-gettingStarted).

Description

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.

Evaluation

The historical data has been split into two groups, a 'training set' and a 'test set'. For the training set, we provide the outcome ('ground truth') for each passenger. You will use this set to build your model to generate predictions for the test set.

For each passenger in the test set, you must predict whether or not they survived the sinking (0 for deceased, 1 for survived). Your score is the percentage of passengers you correctly predict.

The Kaggle leaderboard has a public and private component. 50% of your predictions for the test set have been randomly assigned to the public leaderboard (the same 50% for all users). Your score on this public portion is what will appear on the leaderboard. At the end of the contest, we will reveal your score on the private 50% of the data, which will determine the final winner. This method prevents users from 'overfitting' to the leaderboard.

Data Set

File Name	Available Formats
train	.csv (59.76 kb)
gendermodel	.csv (3.18 kb)
genderclassmodel	.csv (3.18 kb)
test	.csv (27.96 kb)
gendermodel	.py (3.58 kb)
genderclassmodel	.py (5.63 kb)
myfirstforest	.py (3.99 kb)

```
VARIABLE DESCRIPTIONS:
survival        Survival
                (0 = No; 1 = Yes)
pclass          Passenger Class
                (1 = 1st; 2 = 2nd; 3 = 3rd)
name            Name
sex             Sex
age             Age
sibsp           Number of Siblings/Spouses Aboard
parch           Number of Parents/Children Aboard
ticket          Ticket Number
fare            Passenger Fare
cabin           Cabin
embarked        Port of Embarkation
                (C = Cherbourg; Q = Queenstown; S = Southampton)

SPECIAL NOTES:
Pclass is a proxy for socio-economic status (SES)
 1st ~ Upper; 2nd ~ Middle; 3rd ~ Lower

Age is in Years; Fractional if Age less than One (1)
 If the Age is Estimated, it is in the form xx.5

With respect to the family relation variables (i.e. sibsp and parch)
some relations were ignored.  The following are the definitions used
for sibsp and parch.

Sibling:  Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard
Titanic
Spouse:   Husband or Wife of Passenger Aboard Titanic (Mistresses and Fia
nces Ignored)
Parent:   Mother or Father of Passenger Aboard Titanic
Child:    Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Tit
anic

Other family relatives excluded from this study include cousins,
nephews/nieces, aunts/uncles, and in-laws.  Some children travelled
only with a nanny, therefore parch=0 for them.  As well, some
travelled with very close friends or neighbors in a village, however,
the definitions do not support such relations.
```

Setup Imports and Variables

In [1]:

```
import pandas as pd
import numpy as np
import pylab as plt

# Set the global default size of matplotlib figures
plt.rc('figure', figsize=(10, 5))

# Size of matplotlib figures that contain subplots
fizsize_with_subplots = (10, 10)

# Size of matplotlib histogram bins
bin_size = 10
```

Explore the Data

Read the data:

In [2]:

```
df_train = pd.read_csv('../data/titanic/train.csv')
df_train.head()
```

Out[2]:

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

In [3]:

```
1 df_train.tail()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	N
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	N
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	N

View the data types of each column:

In [4]:

```
1 df_train.dtypes
```

Out[4]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Type 'object' is a string for pandas, which poses problems with machine learning algorithms. If we want to use these as features, we'll need to convert these to number representations.

Get some basic information on the DataFrame:

In [5]:

```
1 df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Age, Cabin, and Embarked are missing values. Cabin has too many missing values, whereas we might be able to infer values for Age and Embarked.

Generate various descriptive statistics on the DataFrame:

In [6]:

```
1 df_train.describe()
```

Out[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Now that we have a general idea of the data set contents, we can dive deeper into each column. We'll be doing exploratory data analysis and cleaning data to setup 'features' we'll be using in our machine learning algorithms.

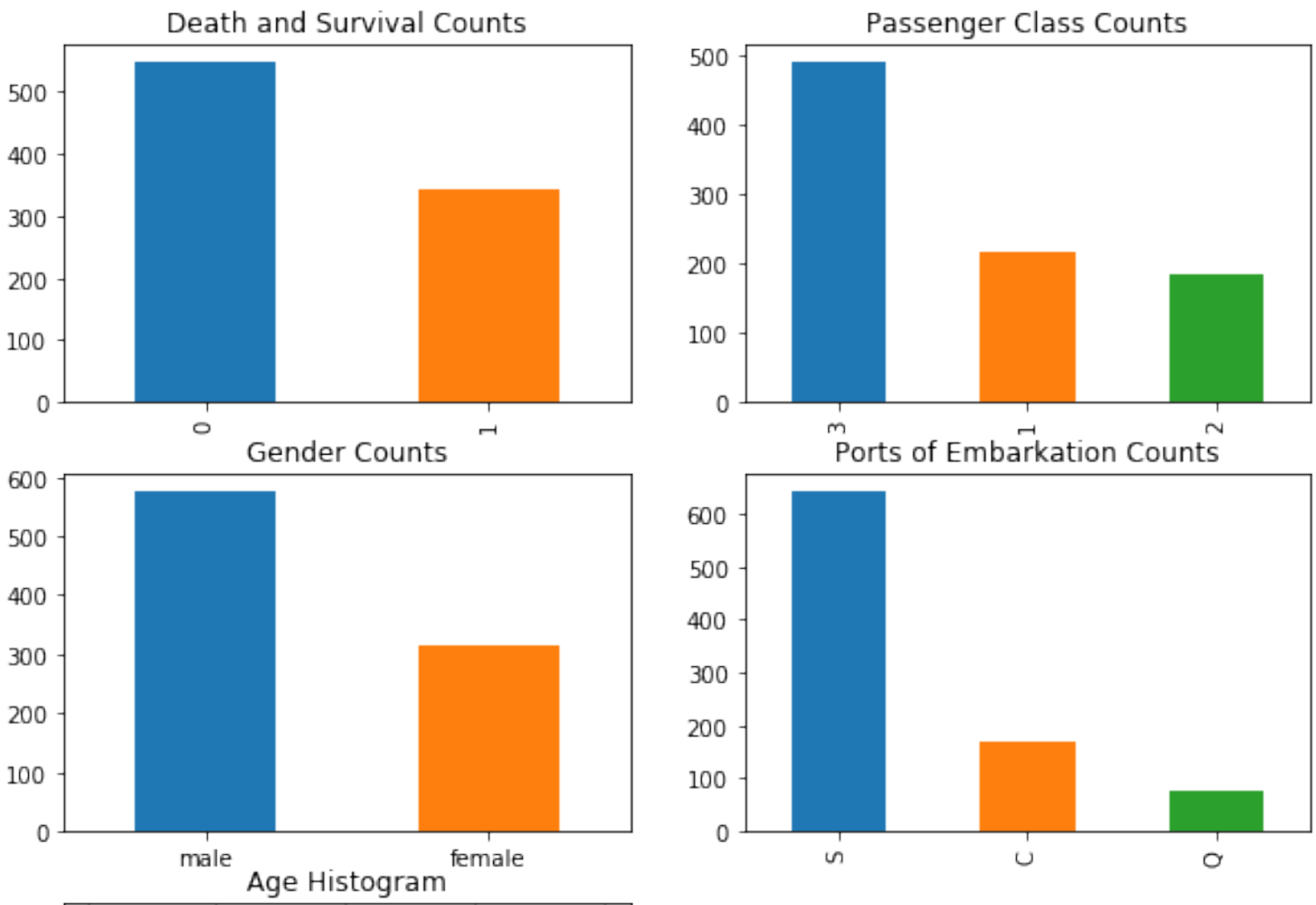
Plot a few features to get a better idea of each:

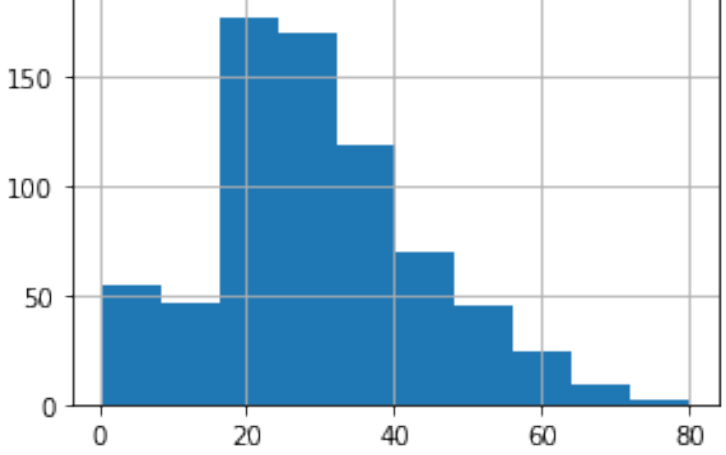
In [7]:

```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 2)

# Plot death and survival counts
plt.subplot2grid(fig_dims, (0, 0))
df_train['Survived'].value_counts().plot(kind='bar',
                                         title='Death and Survival Counts')

# Plot Pclass counts
plt.subplot2grid(fig_dims, (0, 1))
df_train['Pclass'].value_counts().plot(kind='bar',
                                       title='Passenger Class Counts')

# Plot Sex counts
plt.subplot2grid(fig_dims, (1, 0))
df_train['Sex'].value_counts().plot(kind='bar',
                                    title='Gender Counts')
plt.xticks(rotation=0)

# Plot Embarked counts
plt.subplot2grid(fig_dims, (1, 1))
df_train['Embarked'].value_counts().plot(kind='bar',
                                         title='Ports of Embarkation Counts')

# Plot the Age histogram
plt.subplot2grid(fig_dims, (2, 0))
df_train['Age'].hist()
plt.title('Age Histogram')
```

Out[7]:

```
Text(0.5,1,'Age Histogram')
```





Next we'll explore various features to view their impact on survival rates.

Feature: Passenger Classes

From our exploratory data analysis in the previous section, we see there are three passenger classes: First, Second, and Third class. We'll determine which proportion of passengers survived based on their passenger class.

Generate a cross tab of Pclass and Survived:

In [8]:

```
1 pclass_xt = pd.crosstab(df_train['Pclass'], df_train['Survived'])
2 pclass_xt
```

Out[8]:

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

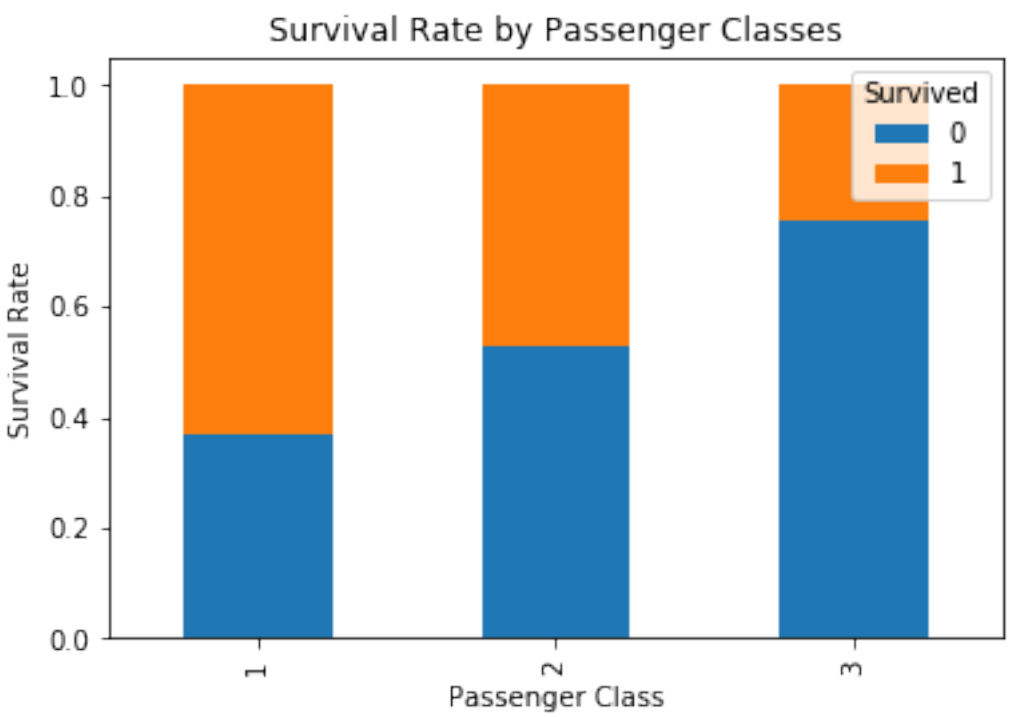
Plot the cross tab:

In [9]:

```
# Normalize the cross tab to sum to 1:
pclass_xt_pct = pclass_xt.div(pclass_xt.sum(1).astype(float), axis=0)

pclass_xt_pct.plot(kind='bar',
                   stacked=True,
                   title='Survival Rate by Passenger Classes')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

Out[9]:

```
Text(0,0.5,'Survival Rate')
```



We can see that passenger class seems to have a significant impact on whether a passenger survived. Those in First Class the highest chance for survival.

Feature: Sex

Gender might have also played a role in determining a passenger's survival rate. We'll need to map Sex from a string to a number to prepare it for machine learning algorithms.

Generate a mapping of Sex from a string to a number representation:

In [10]:

```
sexes = sorted(df_train['Sex'].unique())
genders_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))
genders_mapping
```

Out[10]:

```
{'female': 0, 'male': 1}
```

Transform Sex from a string to a number representation:

In [11]:

```
df_train['Sex_Val'] = df_train['Sex'].map(genders_mapping).astype(int)
df_train.head()
```

Out[11]:

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

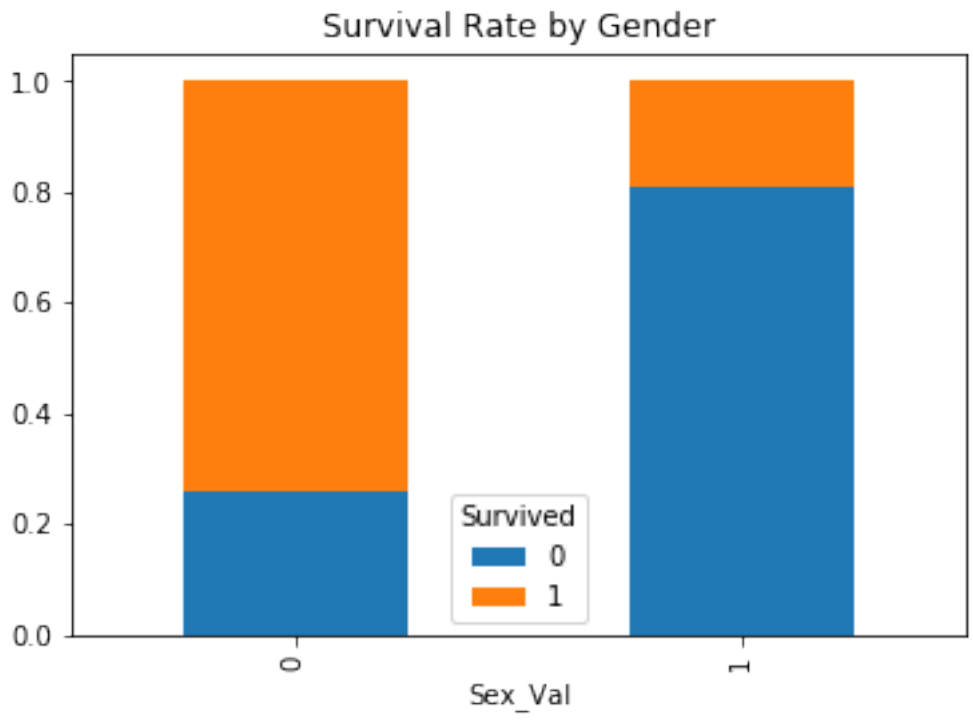
Plot a normalized cross tab for Sex_Val and Survived:

In [12]:

```
sex_val_xt = pd.crosstab(df_train['Sex_Val'], df_train['Survived'])
sex_val_xt_pct = sex_val_xt.div(sex_val_xt.sum(1).astype(float), axis=0)
sex_val_xt_pct.plot(kind='bar', stacked=True, title='Survival Rate by Gender'
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a1ceba8>
```



The majority of females survived, whereas the majority of males did not.

Next we'll determine whether we can gain any insights on survival rate by looking at both Sex and Pclass.

Count males and females in each Pclass:

In [14]:

```
# Get the unique values of Pclass:
passenger_classes = sorted(df_train['Pclass'].unique())

for p_class in passenger_classes:
    print('M: ', p_class, len(df_train[(df_train['Sex'] == 'male') &
                               (df_train['Pclass'] == p_class)]))
    print('F: ', p_class, len(df_train[(df_train['Sex'] == 'female') &
                               (df_train['Pclass'] == p_class)]))
```

```
M:  1 122
F:  1 94
M:  2 108
F:  2 76
M:  3 347
F:  3 144
```

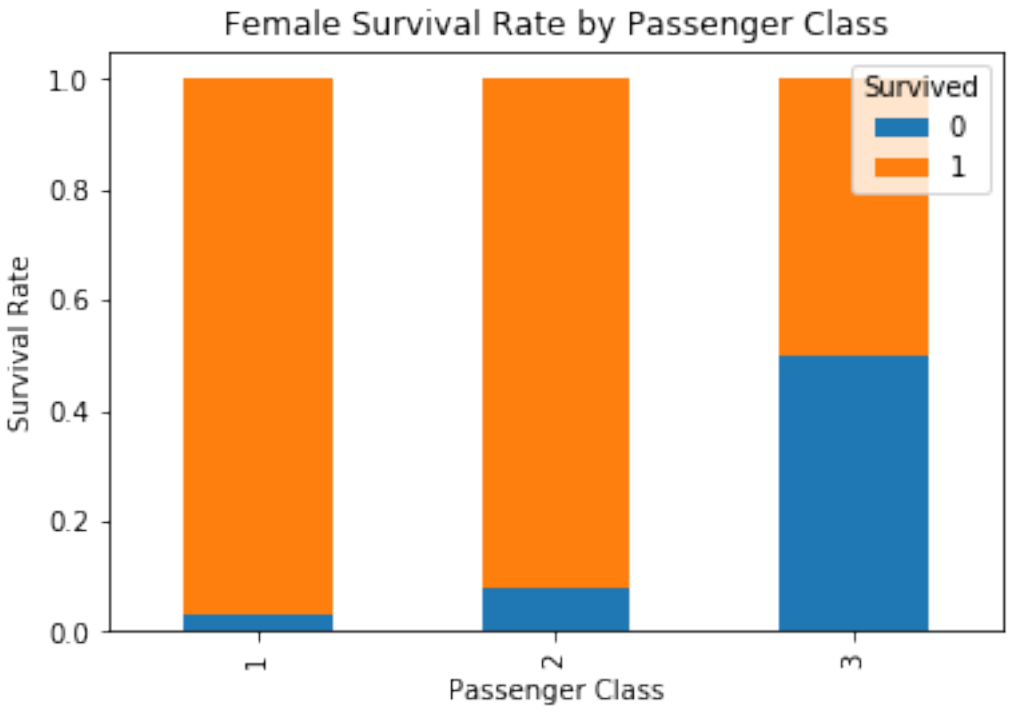
Plot survival rate by Sex and Pclass:

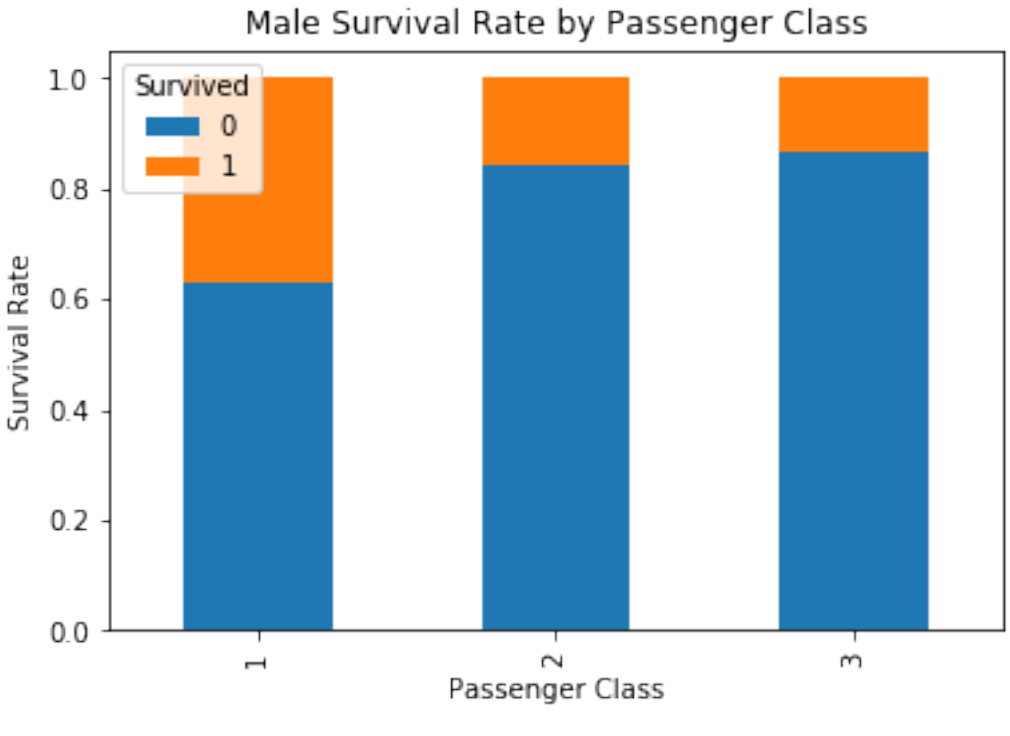
In [15]:

```
# Plot survival rate by Sex
females_df = df_train[df_train['Sex'] == 'female']
females_xt = pd.crosstab(females_df['Pclass'], df_train['Survived'])
females_xt_pct = females_xt.div(females_xt.sum(1).astype(float), axis=0)
females_xt_pct.plot(kind='bar',
                    stacked=True,
                    title='Female Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')

# Plot survival rate by Pclass
males_df = df_train[df_train['Sex'] == 'male']
males_xt = pd.crosstab(males_df['Pclass'], df_train['Survived'])
males_xt_pct = males_xt.div(males_xt.sum(1).astype(float), axis=0)
males_xt_pct.plot(kind='bar',
                  stacked=True,
                  title='Male Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

Out[15]:

```
Text(0,0.5,'Survival Rate')
```





The vast majority of females in First and Second class survived. Males in First class had the highest chance for survival.

Feature: Embarked

The Embarked column might be an important feature but it is missing a couple data points which might pose a problem for machine learning algorithms:

In [15]:

```
1 df_train[df_train['Embarked'].isnull()]
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
61	62	1	1	Icard, Miss. Amelie	female	38	0	0	113572	80	B2
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62	0	0	113572	80	B2

Prepare to map Embarked from a string to a number representation:

In [21]:

```
1 # Get the unique values of Embarked
2 embarked_locs=sorted(df_train['Embarked'].unique())
3
4 embarked_locs_mapping = dict(zip(embarked_locs,range(0, len(embarked_locs) +
5 print(embarked_locs_mapping)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-21-4305ad16072b> in <module>()
      1 # Get the unique values of Embarked
----> 2 embarked_locs=sorted(df_train['Embarked'].unique())
      3
      4 #embarked_locs_mapping = dict(zip(embarked_locs,range(0, len
(embarked_locs) + 1)))
      5 #print(embarked_locs_mapping)

TypeError: '<' not supported between instances of 'float' and 'str'
```

Transform Embarked from a string to a number representation to prepare it for machine learning algorithms:

In [22]:

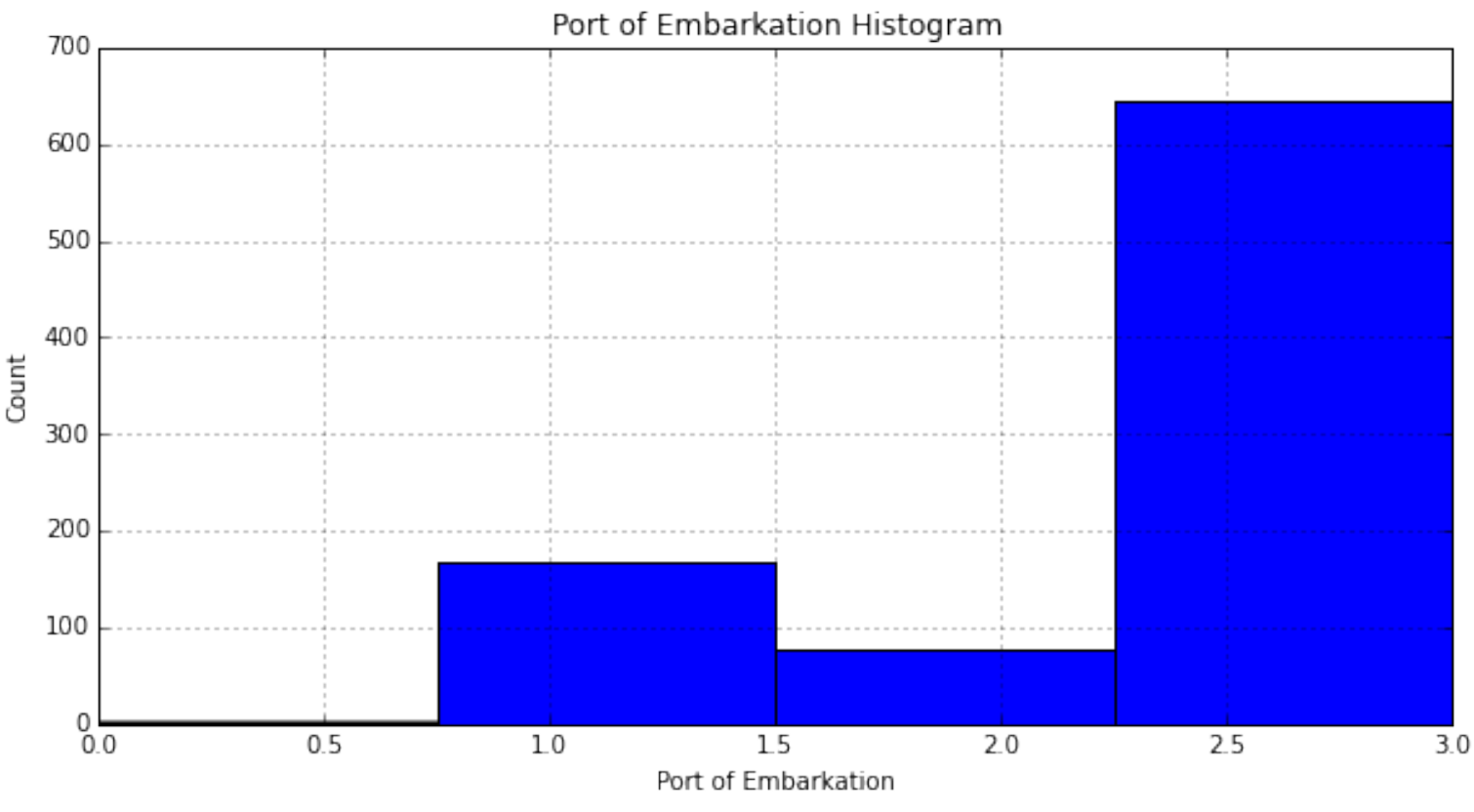
```
df_train['Embarked_Val'] = df_train['Embarked'] \
                                .map(embarked_locs_mapping) \
                                .astype(int)
df_train.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-22-7afd02dfe9dd> in <module>()
----> 1 df_train['Embarked_Val'] = df_train['Embarked']
.map(embarked_locs_mapping)                                .astype(i
nt)
      2 df_train.head()

NameError: name 'embarked_locs_mapping' is not defined
```

Plot the histogram for Embarked_Val:

In [18]:

```
df_train['Embarked_Val'].hist(bins=len(embarked_locs), range=(0, 3))
plt.title('Port of Embarkation Histogram')
plt.xlabel('Port of Embarkation')
plt.ylabel('Count')
plt.show()
```



Since the vast majority of passengers embarked in 'S': 3, we assign the missing values in Embarked to 'S':

In [19]:

```
if len(df_train[df_train['Embarked'].isnull()] > 0):
    df_train.replace({'Embarked_Val' :
                      { embarked_locs_mapping[nan] : embarked_locs_mapping['S']
                      }
                  },
                  inplace=True)
```

Verify we do not have any more NaNs for Embarked_Val:

In [20]:

```
embarked_locs = sorted(df_train['Embarked_Val'].unique())
embarked_locs
```

Out[20]:

```
array([1, 2, 3])
```

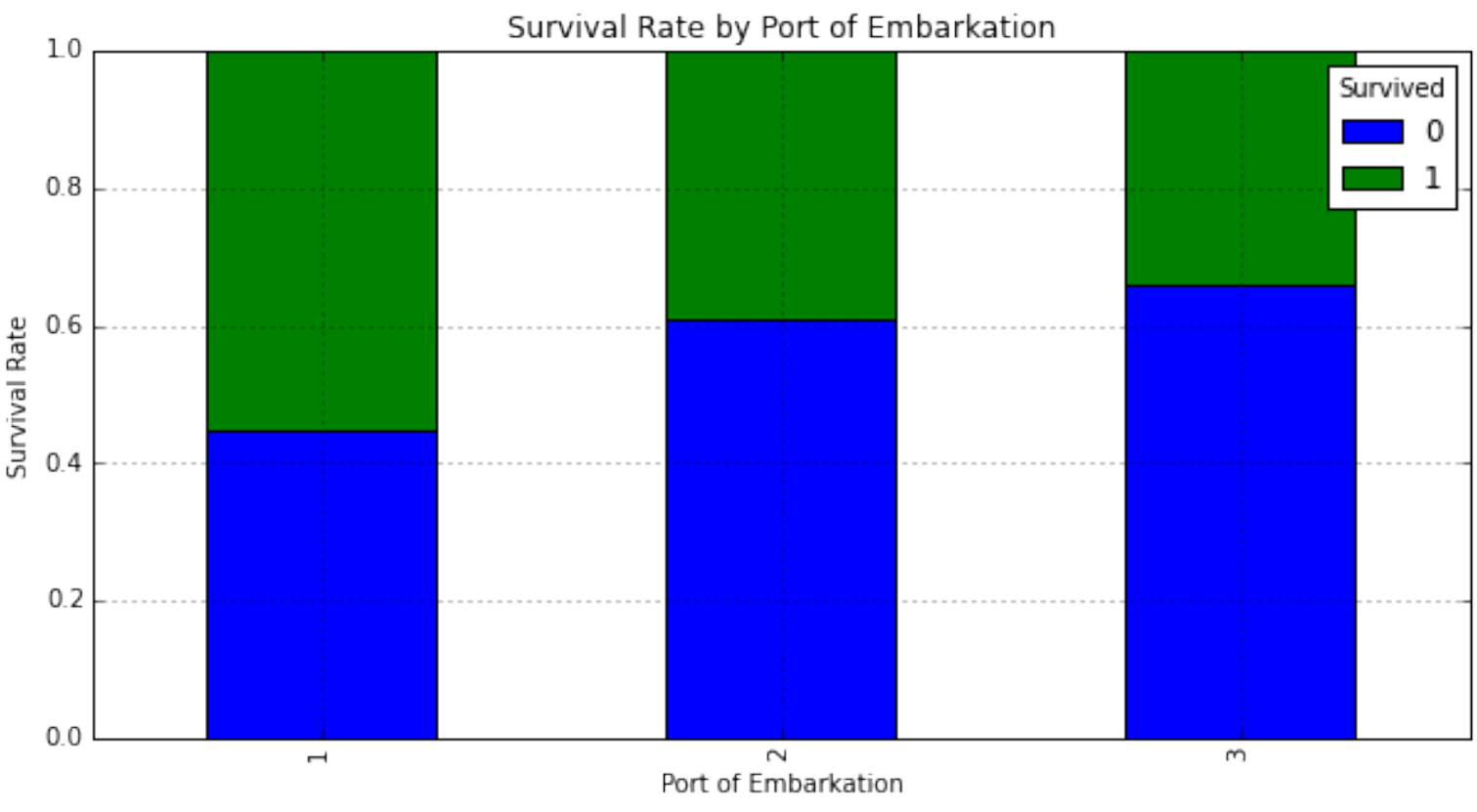
Plot a normalized cross tab for Embarked_Val and Survived:

In [21]:

```
embarked_val_xt = pd.crosstab(df_train['Embarked_Val'], df_train['Survived'])
embarked_val_xt_pct = \
    embarked_val_xt.div(embarked_val_xt.sum(1).astype(float), axis=0)
embarked_val_xt_pct.plot(kind='bar', stacked=True)
plt.title('Survival Rate by Port of Embarkation')
plt.xlabel('Port of Embarkation')
plt.ylabel('Survival Rate')
```

Out[21]:

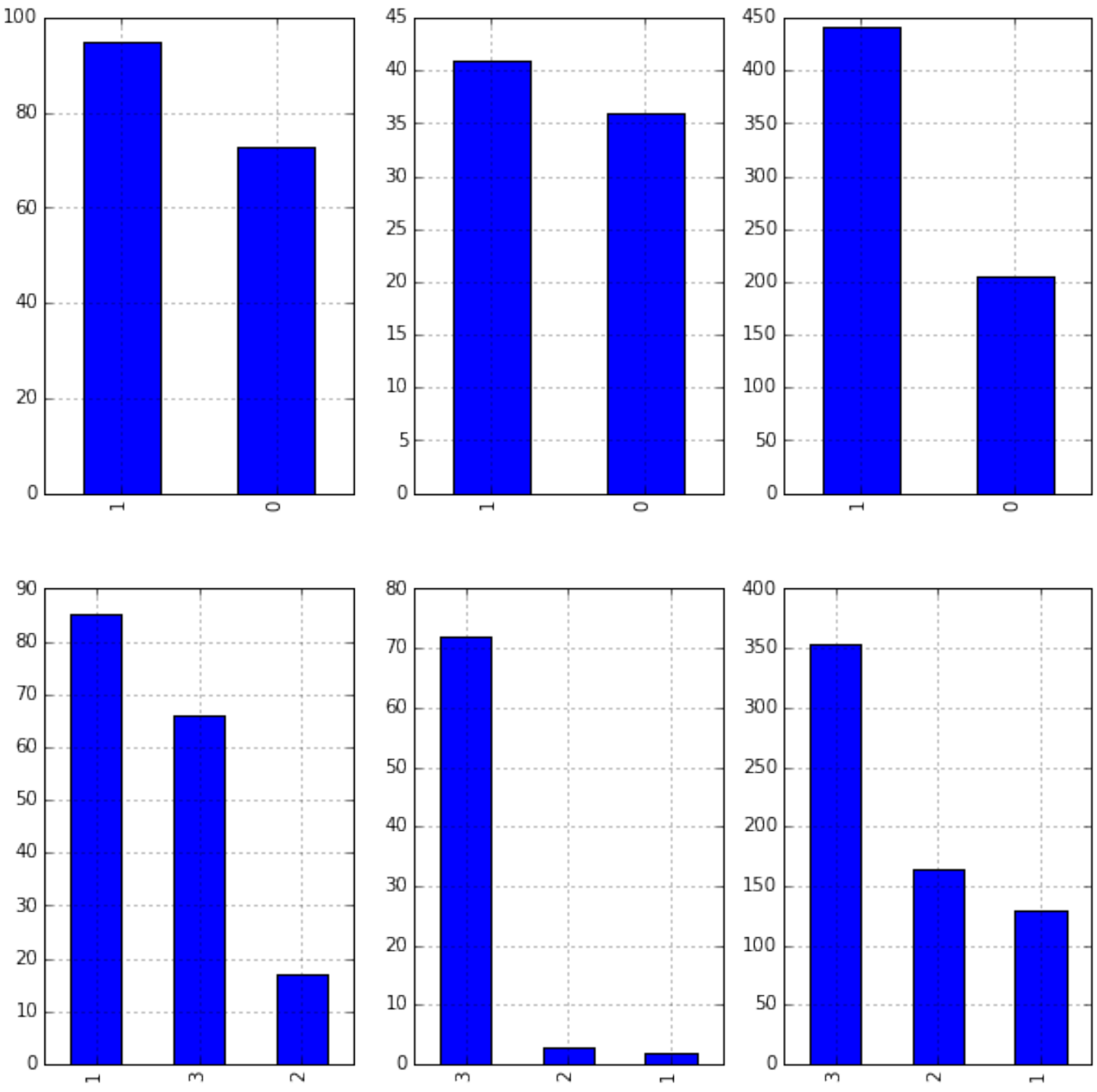
```
<matplotlib.text.Text at 0x10b7e28d0>
```



It appears those that embarked in location 'C': 1 had the highest rate of survival. We'll dig in some more to see why this might be the case. Below we plot a graphs to determine gender and passenger class makeup for each port:

In [22]:

```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)

rows = 2
cols = 3
col_names = ('Sex_Val', 'Pclass')

for portIdx in embarked_locs:
    for colIdx in range(0, len(col_names)):
        plt.subplot2grid((rows, cols), (colIdx, portIdx - 1))
        df_train[df_train['Embarked_Val'] == portIdx][col_names[colIdx]] \
            .value_counts().plot(kind='bar')
```



Leaving Embarked as integers implies ordering in the values, which does not exist. Another way to represent Embarked without ordering is to create dummy variables:

In [23]:

```
df_train = pd.concat([df_train, pd.get_dummies(df_train['Embarked_Val'], pref
```

Feature: Age

The Age column seems like an important feature--unfortunately it is missing many values. We'll need to fill in the missing values like we did with Embarked.

Filter to view missing Age values:

In [24]:

```
df_train[df_train['Age'].isnull()][['Sex', 'Pclass', 'Age']].head()
```

Out[24]:

	Sex	Pclass	Age
5	male	3	NaN
17	male	2	NaN
19	female	3	NaN
26	male	3	NaN
28	female	3	NaN

Determine the Age typical for each passenger class by Sex_Val. We'll use the median instead of the mean because the Age histogram seems to be right skewed.

In [25]:

```
# To keep Age in tact, make a copy of it called AgeFill
# that we will use to fill in the missing ages:
df_train['AgeFill'] = df_train['Age']

# Populate AgeFill
df_train['AgeFill'] = df_train['AgeFill'] \
                        .groupby([df_train['Sex_Val'], df_train['Pclass']]) \
                        .apply(lambda x: x.fillna(x.median()))
```

Ensure AgeFill does not contain any missing values:

In [26]:

```
len(df_train[df_train['AgeFill'].isnull()])
```

Out[26]:

```
0
```

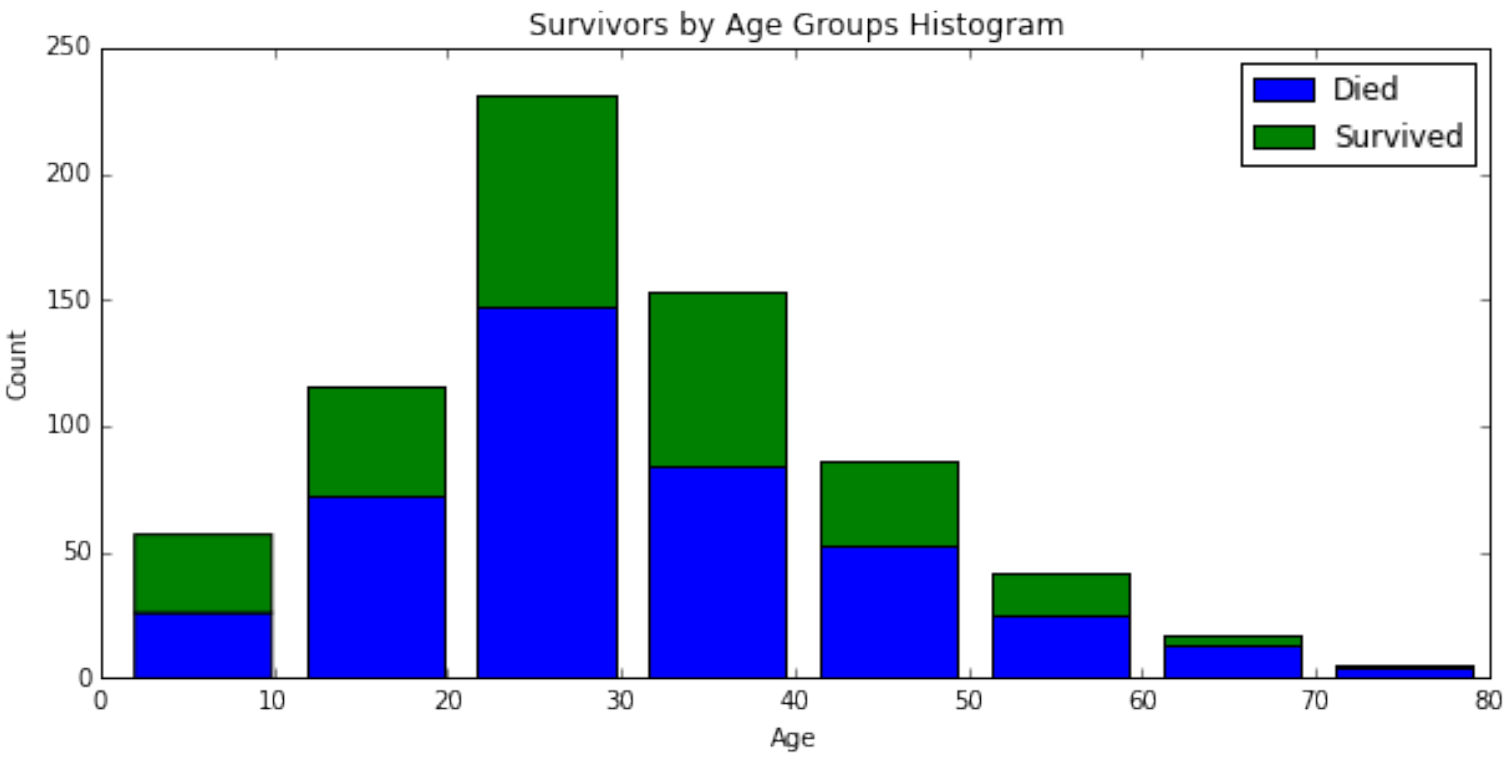
Plot a normalized cross tab for AgeFill and Survived:

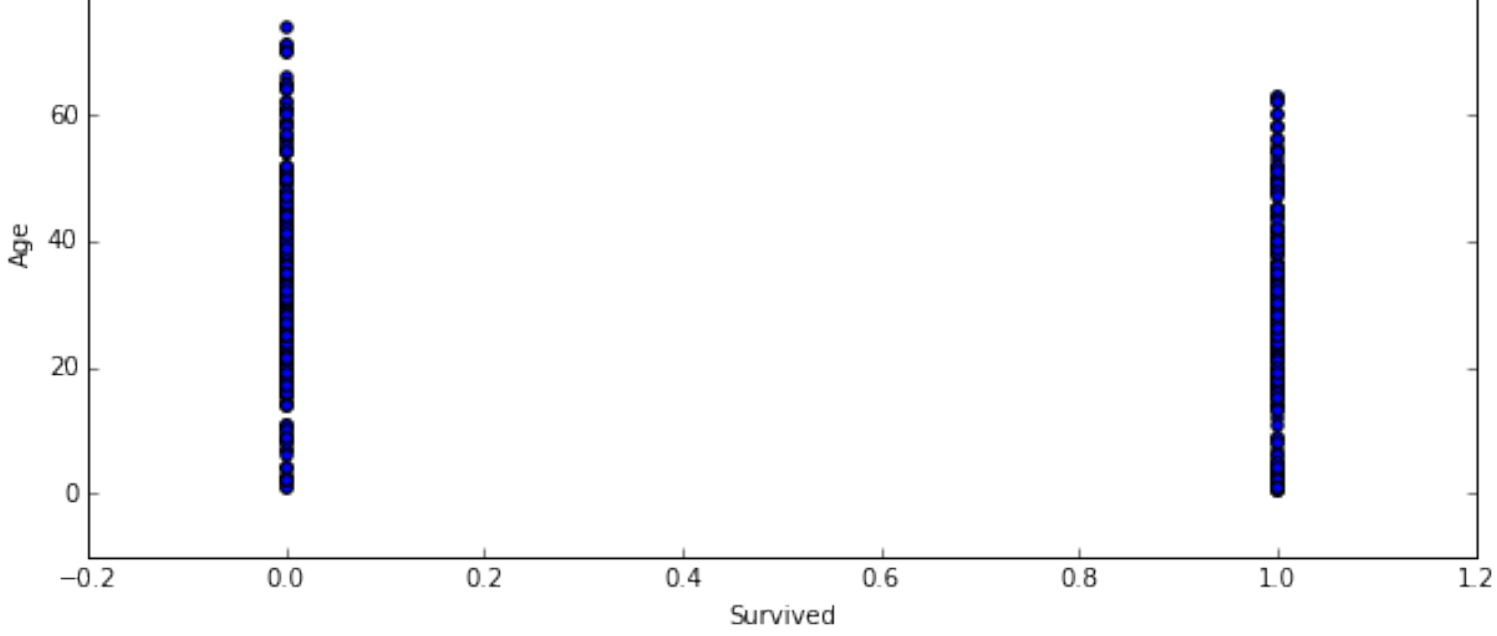
In [27]:

```
# Set up a grid of plots
fig, axes = plt.subplots(2, 1, figsize=fizsize_with_subplots)

# Histogram of AgeFill segmented by Survived
df1 = df_train[df_train['Survived'] == 0]['Age']
df2 = df_train[df_train['Survived'] == 1]['Age']
max_age = max(df_train['AgeFill'])
axes[0].hist([df1, df2],
             bins=max_age / bin_size,
             range=(1, max_age),
             stacked=True)
axes[0].legend(('Died', 'Survived'), loc='best')
axes[0].set_title('Survivors by Age Groups Histogram')
axes[0].set_xlabel('Age')
axes[0].set_ylabel('Count')

# Scatter plot Survived and AgeFill
axes[1].scatter(df_train['Survived'], df_train['AgeFill'])
axes[1].set_title('Survivors by Age Plot')
axes[1].set_xlabel('Survived')
axes[1].set_ylabel('Age')
```

Out[27]:

```
<matplotlib.text.Text at 0x10c4125d0>
```





Unfortunately, the graphs above do not seem to clearly show any insights. We'll keep digging further.

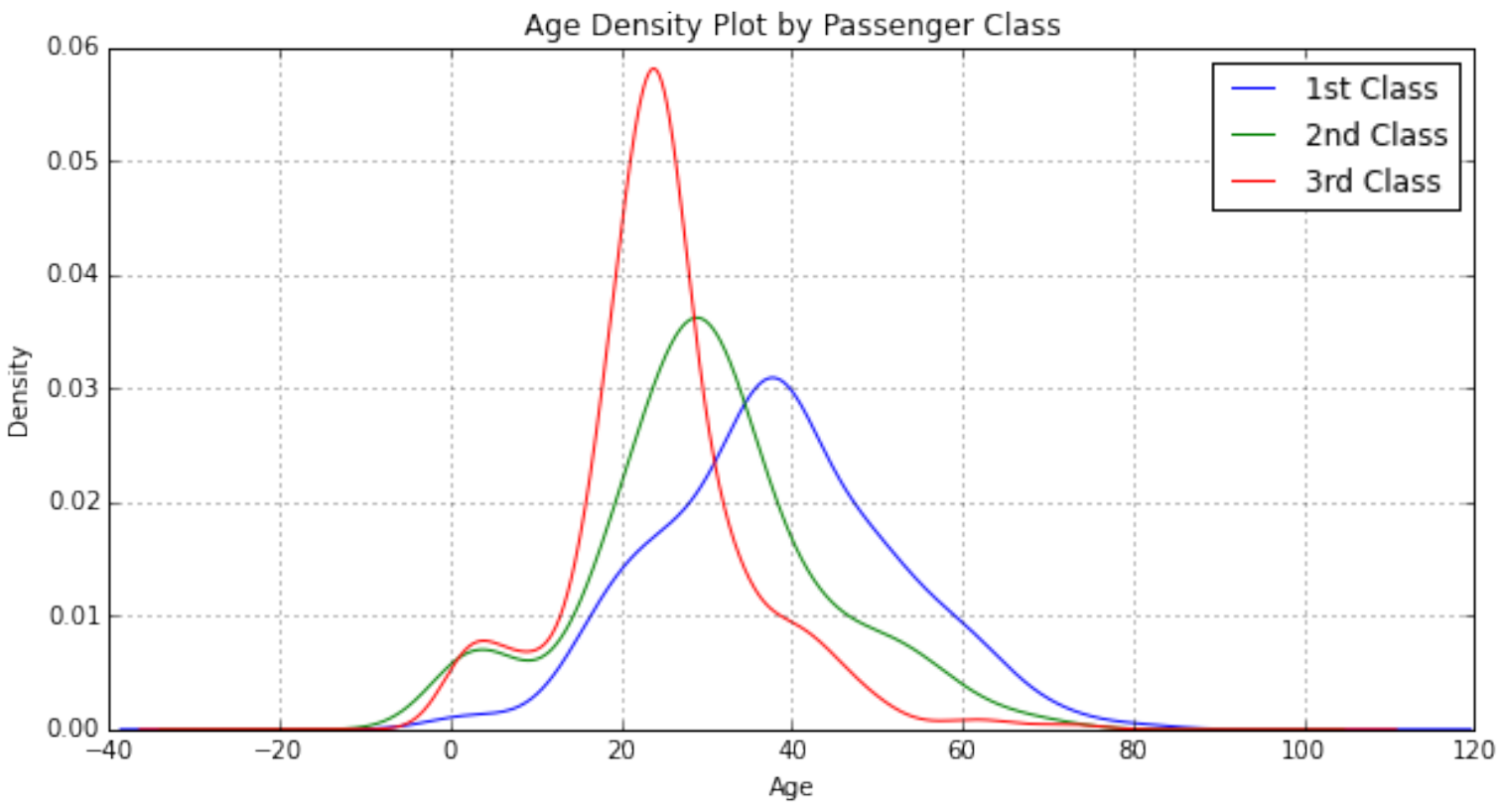
Plot AgeFill density by Pclass:

In [28]:

```
for pclass in passenger_classes:
    df_train.AgeFill[df_train.Pclass == pclass].plot(kind='kde')
plt.title('Age Density Plot by Passenger Class')
plt.xlabel('Age')
plt.legend(('1st Class', '2nd Class', '3rd Class'), loc='best')
```

Out[28]:

```
<matplotlib.legend.Legend at 0x10be093d0>
```



When looking at AgeFill density by Pclass, we see the first class passengers were generally older then second class passengers, which in turn were older than third class passengers. We've determined that first class passengers had a higher survival rate than second class passengers, which in turn had a higher survival rate than third class passengers.

In [29]:

```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 1)

# Plot the AgeFill histogram for Survivors
plt.subplot2grid(fig_dims, (0, 0))
survived_df = df_train[df_train['Survived'] == 1]
survived_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))

# Plot the AgeFill histogram for Females
plt.subplot2grid(fig_dims, (1, 0))
females_df = df_train[(df_train['Sex_Val'] == 0) & (df_train['Survived'] == 1
females_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))

# Plot the AgeFill histogram for first class passengers
plt.subplot2grid(fig_dims, (2, 0))
class1_df = df_train[(df_train['Pclass'] == 1) & (df_train['Survived'] == 1)]
class1_df['AgeFill'].hist(bins=max_age / bin_size, range=(1, max_age))
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10d6c22d0>
```

In the first graph, we see that most survivors come from the 20's to 30's age ranges and might be explained by the following two graphs. The second graph shows most females are within their 20's. The third graph shows most first class passengers are within their 30's.

Feature: Family Size

Feature enginering involves creating new features or modifying existing features which might be advantageous to a machine learning algorithm.

Define a new feature FamilySize that is the sum of Parch (number of parents or children on board) and SibSp (number of siblings or spouses):

In [30]:

```
df_train['FamilySize'] = df_train['SibSp'] + df_train['Parch']
df_train.head()
```

Out[30]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500

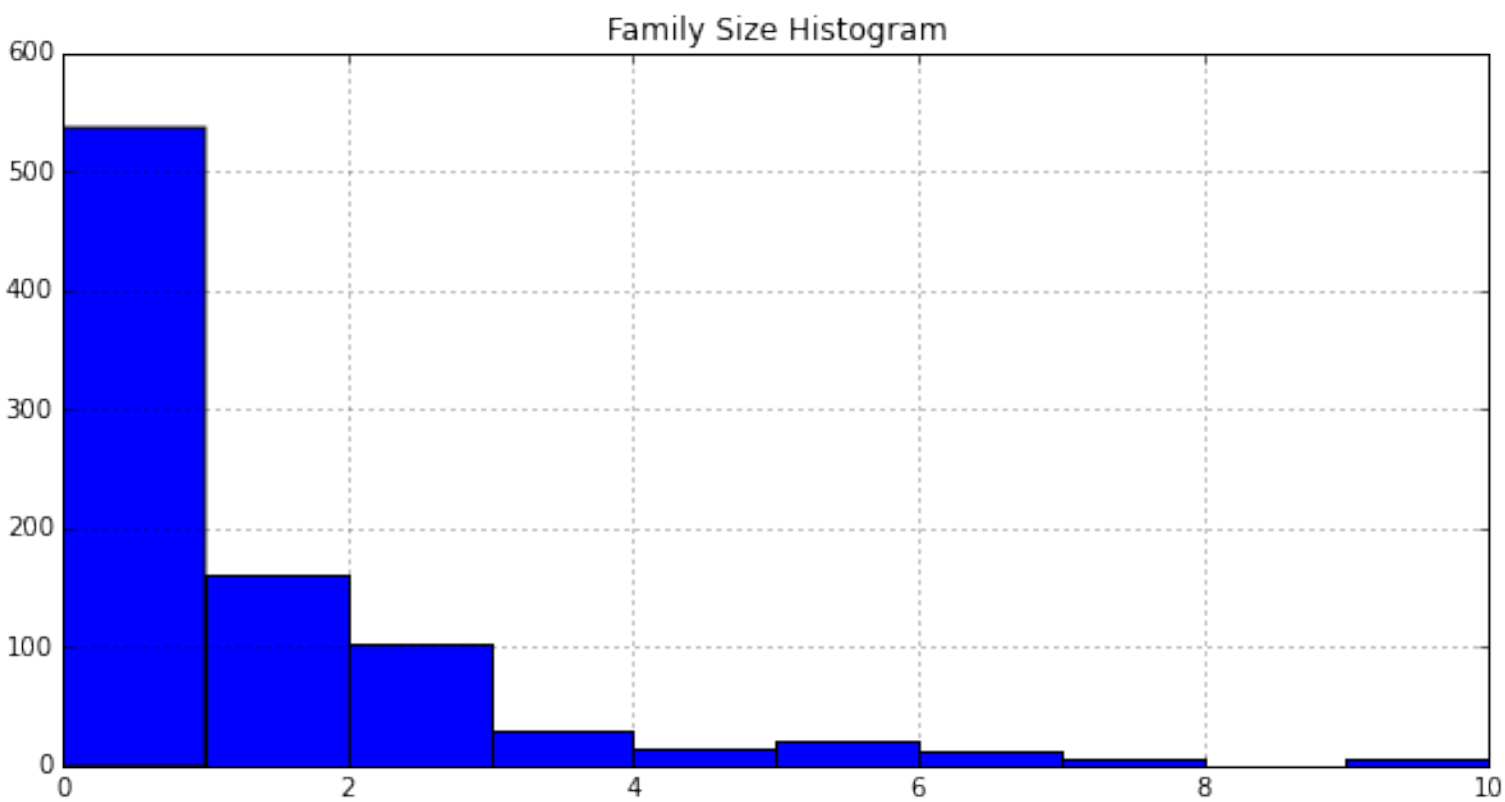
Plot a histogram of FamilySize:

In [31]:

```
1  df_train['FamilySize'].hist()
2  plt.title('Family Size Histogram')
```

Out[31]:

```
<matplotlib.text.Text at 0x10db78590>
```



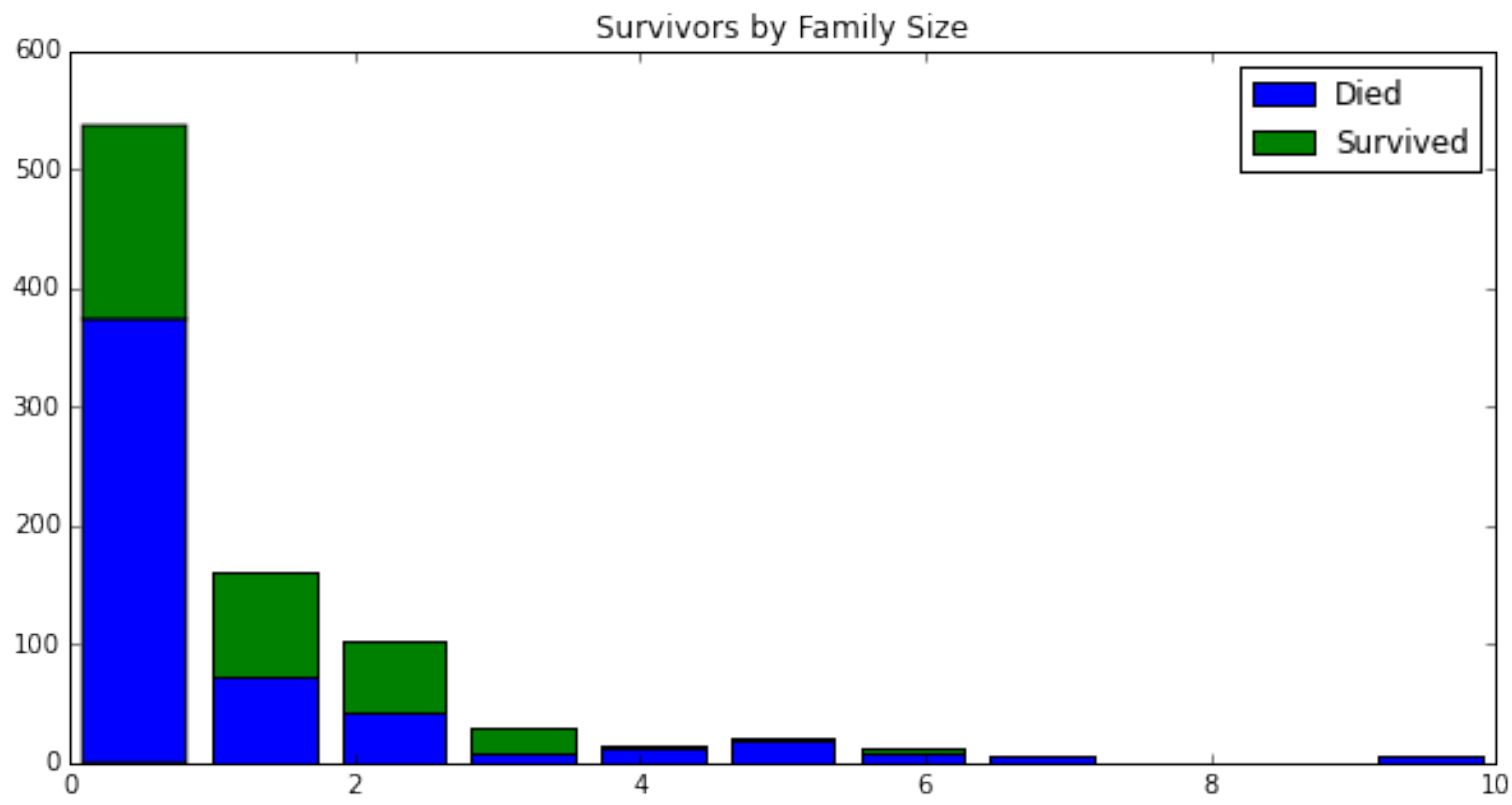
Plot a histogram of AgeFill segmented by Survived:

In [32]:

```
# Get the unique values of Embarked and its maximum
family_sizes = sorted(df_train['FamilySize'].unique())
family_size_max = max(family_sizes)

df1 = df_train[df_train['Survived'] == 0]['FamilySize']
df2 = df_train[df_train['Survived'] == 1]['FamilySize']
plt.hist([df1, df2],
         bins=family_size_max + 1,
         range=(0, family_size_max),
         stacked=True)
plt.legend(('Died', 'Survived'), loc='best')
plt.title('Survivors by Family Size')
```

Out[32]:

```
<matplotlib.text.Text at 0x10dd85bd0>
```



Based on the histograms, it is not immediately obvious what impact FamilySize has on survival. The machine learning algorithms might benefit from this feature.

Additional features we might want to engineer might be related to the Name column, for example honorrary or pedestrian titles might give clues and better predictive power for a male's survival.

Final Data Preparation for Machine Learning

Many machine learning algorithms do not work on strings and they usually require the data to be in an array, not a DataFrame.

Show only the columns of type 'object' (strings):

In [33]:

```
df_train.dtypes[df_train.dtypes.map(lambda x: x == 'object')]
```

Out[33]:

```
Name        object
Sex         object
Ticket      object
Cabin       object
Embarked    object
dtype: object
```

Drop the columns we won't use:

In [34]:

```
df_train = df_train.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'],
                         axis=1)
```

Drop the following columns:

- The Age column since we will be using the AgeFill column instead.
- The SibSp and Parch columns since we will be using FamilySize instead.
- The PassengerId column since it won't be used as a feature.
- The Embarked_Val as we decided to use dummy variables instead.

In [35]:

```
df_train = df_train.drop(['Age', 'SibSp', 'Parch', 'PassengerId', 'Embarked_V
df_train.dtypes
```

Out[35]:

```
Survived            int64
Pclass              int64
Fare              float64
Sex_Val             int64
Embarked_Val_1    float64
Embarked_Val_2    float64
Embarked_Val_3    float64
AgeFill           float64
FamilySize          int64
dtype: object
```

Convert the DataFrame to a numpy array:

In [36]:

```
train_data = df_train.values
train_data
```

Out[36]:

```
array([[  0.    ,   3.    ,   7.25  , ...,   1.    ,  22.    ,   1.
],
       [  1.    ,   1.    ,  71.2833, ...,   0.    ,  38.    ,   1.
],
       [  1.    ,   3.    ,   7.925 , ...,   1.    ,  26.    ,   0.
],
       ...,
       [  0.    ,   3.    ,  23.45  , ...,   1.    ,  21.5   ,   3.
],
       [  1.    ,   1.    ,  30.    , ...,   0.    ,  26.    ,   0.
],
       [  0.    ,   3.    ,   7.75  , ...,   0.    ,  32.    ,   0.
]])
```

Data Wrangling Summary

Below is a summary of the data wrangling we performed on our training data set. We encapsulate this in a function since we'll need to do the same operations to our test set later.

In [37]:

```
def clean_data(df, drop_passenger_id):

    # Get the unique values of Sex
    sexes = sorted(df['Sex'].unique())

    # Generate a mapping of Sex from a string to a number representation
    genders_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))

    # Transform Sex from a string to a number representation
    df['Sex_Val'] = df['Sex'].map(genders_mapping).astype(int)

    # Get the unique values of Embarked
    embarked_locs = sorted(df['Embarked'].unique())

    # Generate a mapping of Embarked from a string to a number representation
    embarked_locs_mapping = dict(zip(embarked_locs,
                                     range(0, len(embarked_locs) + 1)))

    # Transform Embarked from a string to dummy variables
    df = pd.concat([df, pd.get_dummies(df['Embarked'], prefix='Embarked_Val')

    # Fill in missing values of Embarked
    # Since the vast majority of passengers embarked in 'S': 3,
    # we assign the missing values in Embarked to 'S':
    if len(df[df['Embarked'].isnull()] > 0):
        df.replace({'Embarked_Val' :
                        { embarked_locs_mapping[nan] : embarked_locs_mapping['
```

```
                    }
                },
                inplace=True)

    # Fill in missing values of Fare with the average Fare
    if len(df[df['Fare'].isnull()] > 0):
        avg_fare = df['Fare'].mean()
        df.replace({ None: avg_fare }, inplace=True)

    # To keep Age in tact, make a copy of it called AgeFill
    # that we will use to fill in the missing ages:
    df['AgeFill'] = df['Age']

    # Determine the Age typical for each passenger class by Sex_Val.
    # We'll use the median instead of the mean because the Age
    # histogram seems to be right skewed.
    df['AgeFill'] = df['AgeFill'] \
                        .groupby([df['Sex_Val'], df['Pclass']]) \
                        .apply(lambda x: x.fillna(x.median()))

    # Define a new feature FamilySize that is the sum of
    # Parch (number of parents or children on board) and
    # SibSp (number of siblings or spouses):
    df['FamilySize'] = df['SibSp'] + df['Parch']

    # Drop the columns we won't use:
    df = df.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], axis=1)

    # Drop the Age column since we will be using the AgeFill column instead.
    # Drop the SibSp and Parch columns since we will be using FamilySize.
    # Drop the PassengerId column since it won't be used as a feature.
    df = df.drop(['Age', 'SibSp', 'Parch'], axis=1)

    if drop_passenger_id:
        df = df.drop(['PassengerId'], axis=1)

    return df
```

Random Forest: Training

Create the random forest object:

In [38]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)
```

Fit the training data and create the decision trees:

In [39]:

```
# Training data features, skip the first column 'Survived'
train_features = train_data[:, 1:]

# 'Survived' column values
train_target = train_data[:, 0]

# Fit the model to our training data
clf = clf.fit(train_features, train_target)
score = clf.score(train_features, train_target)
"Mean accuracy of Random Forest: {0}".format(score)
```

Out[39]:

```
'Mean accuracy of Random Forest: 0.980920314254'
```

Random Forest: Predicting

Read the test data:

In [40]:

```
df_test = pd.read_csv('../data/titanic/test.csv')
df_test.head()
```

Out[40]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

Note the test data does not contain the column 'Survived', we'll use our trained model to predict these values.

In [41]:

```
# Data wrangle the test set and convert it to a numpy array
df_test = clean_data(df_test, drop_passenger_id=False)
test_data = df_test.values
```

Take the decision trees and run it on the test data:

In [42]:

```
# Get the test data features, skipping the first column 'PassengerId'
test_x = test_data[:, 1:]

# Predict the Survival values for the test data
test_y = clf.predict(test_x)
```

Random Forest: Prepare for Kaggle Submission

Create a DataFrame by combining the index from the test data with the output of predictions, then write the results to the output:

In [43]:

```
df_test['Survived'] = test_y
df_test[['PassengerId', 'Survived']] \
    .to_csv('../data/titanic/results-rf.csv', index=False)
```

Evaluate Model Accuracy

Submitting to Kaggle will give you an accuracy score. It would be helpful to get an idea of accuracy without submitting to Kaggle.

We'll split our training data, 80% will go to "train" and 20% will go to "test":

In [44]:

```
from sklearn import metrics
from sklearn.cross_validation import train_test_split

# Split 80-20 train vs test data
train_x, test_x, train_y, test_y = train_test_split(train_features,
                                                    train_target,
                                                    test_size=0.20,
                                                    random_state=0)
print (train_features.shape, train_target.shape)
print (train_x.shape, train_y.shape)
print (test_x.shape, test_y.shape)
```

```
((891, 8), (891,))
((712, 8), (712,))
((179, 8), (179,))
```

Use the new training data to fit the model, predict, and get the accuracy score:

In [45]:

```
clf = clf.fit(train_x, train_y)
predict_y = clf.predict(test_x)

from sklearn.metrics import accuracy_score
print ("Accuracy = %.2f" % (accuracy_score(test_y, predict_y)))
```

```
Accuracy = 0.83
```

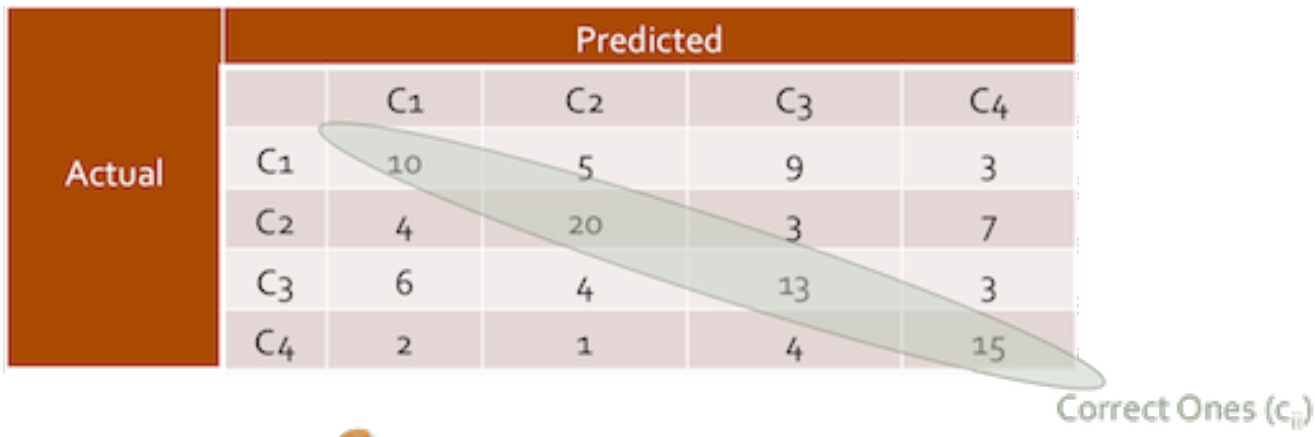
View the Confusion Matrix:

	condition True	**condition false**
prediction true	True Positive	False positive
Prediction False	False Negative	True Negative

In [46]:

```
from IPython.core.display import Image
Image(filename='../data/confusion_matrix.png', width=800)
```

Out[46]:



$$\text{Precision} = \frac{C_{ii}}{\sum_{\text{Columns } i} C_{ij}} \qquad \text{Recall} = \frac{C_{ii}}{\sum_{\text{Rows } j} C_{ij}}$$

Get the model score and confusion matrix:

In [47]:

```
model_score = clf.score(test_x, test_y)
print ("Model Score %.2f \n" % (model_score))

confusion_matrix = metrics.confusion_matrix(test_y, predict_y)
print ("Confusion Matrix ", confusion_matrix)

print ("          Predicted")
print ("         |  0  |  1  |")
print ("         |-----|-----|")
print ("       0 | %3d | %3d |" % (confusion_matrix[0, 0],
                                   confusion_matrix[0, 1]))
print ("Actual   |-----|-----|")
print ("       1 | %3d | %3d |" % (confusion_matrix[1, 0],
                                   confusion_matrix[1, 1]))
print ("         |-----|-----|")
```

```
Model Score 0.83

('Confusion Matrix ', array([[98, 12],
       [19, 50]]))
          Predicted
         |  0  |  1  |
         |-----|-----|
       0 |  98 |  12 |
Actual   |-----|-----|
       1 |  19 |  50 |
         |-----|-----|
```

Display the classification report:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2TP}{2TP + FP + FN}$$

In [48]:

```
from sklearn.metrics import classification_report
print(classification_report(test_y,
                            predict_y,
                            target_names=['Not Survived', 'Survived']))
```

```
             precision    recall  f1-score   support

Not Survived       0.84      0.89      0.86       110
    Survived       0.81      0.72      0.76        69

 avg / total       0.83      0.83      0.82       179
```

In [48]:

```

```