Anish Hiranandani MATH4432 20233794 Project2

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
```

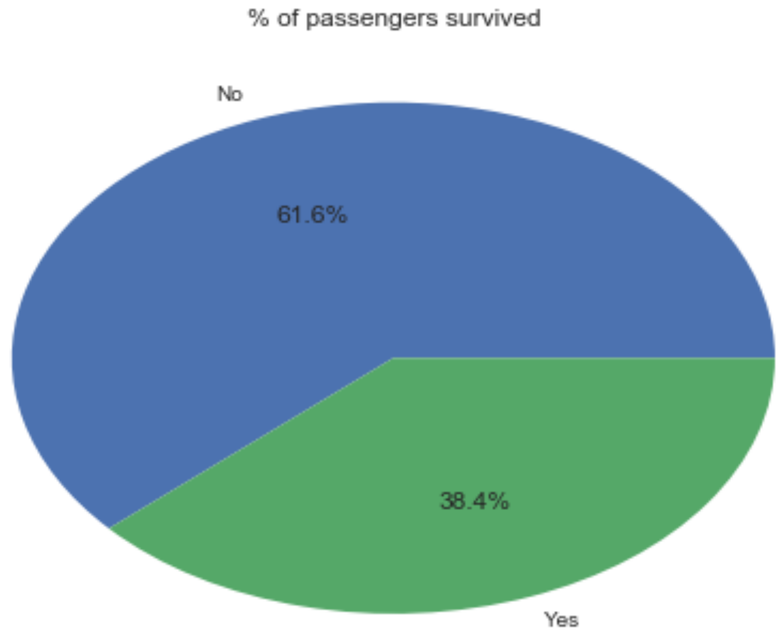
In [3]:
```
dataset = pd.read_csv('train.csv')
```

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Survived         0
dtype: int64
```

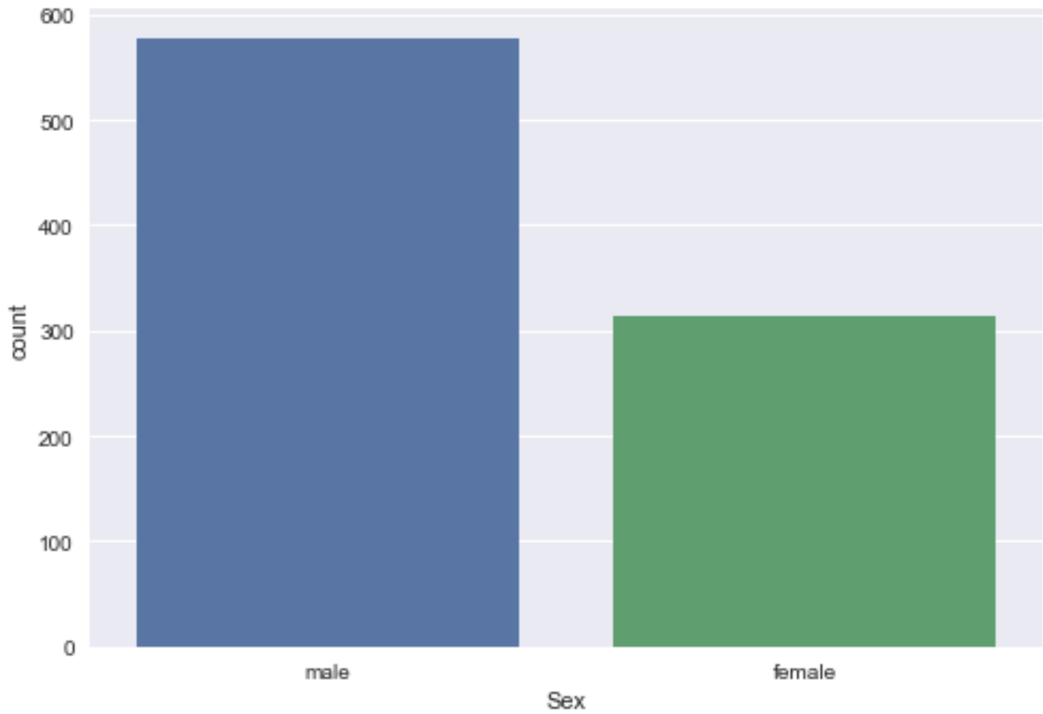
Make pie chart for class label to see distribution in dataset

In [5]:
```
plt.pie(dataset['Survived'].value_counts(), labels = ['No', 'Yes'],aut
plt.title('% of passengers survived')
plt.show()
```



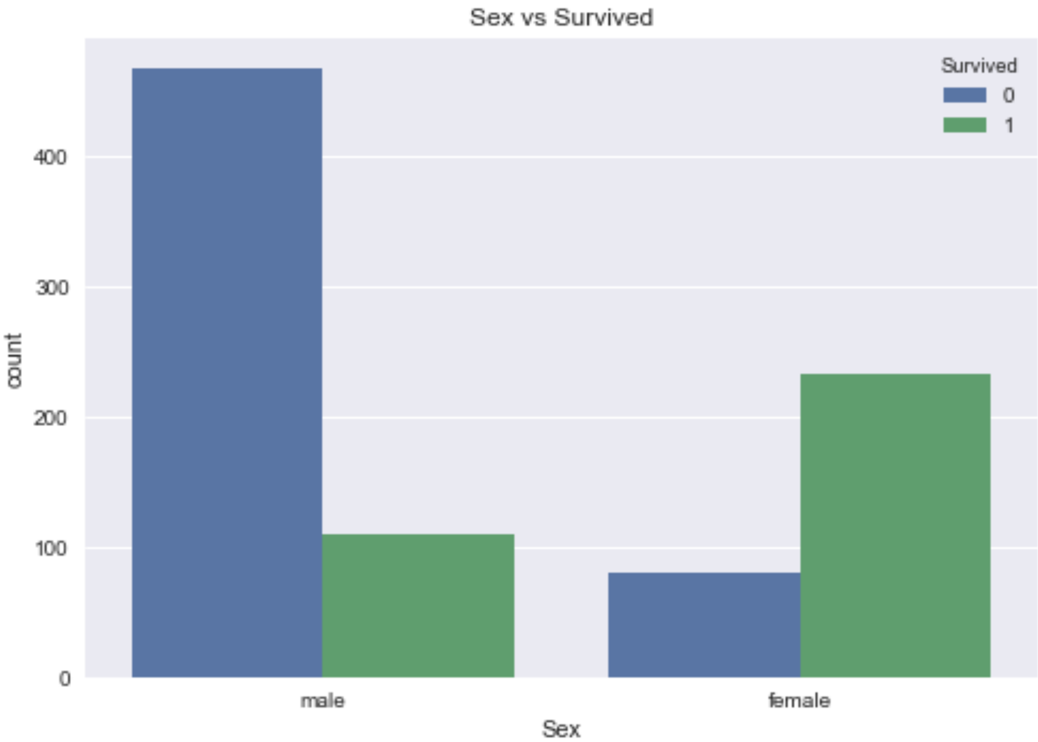
Draw countplot of Sex attribute in the dataset. Clearly, there are many more males than females. ~570 males ~320 females

In [6]:
```
sns.countplot(x = 'Sex', data = dataset)
plt.show()
```



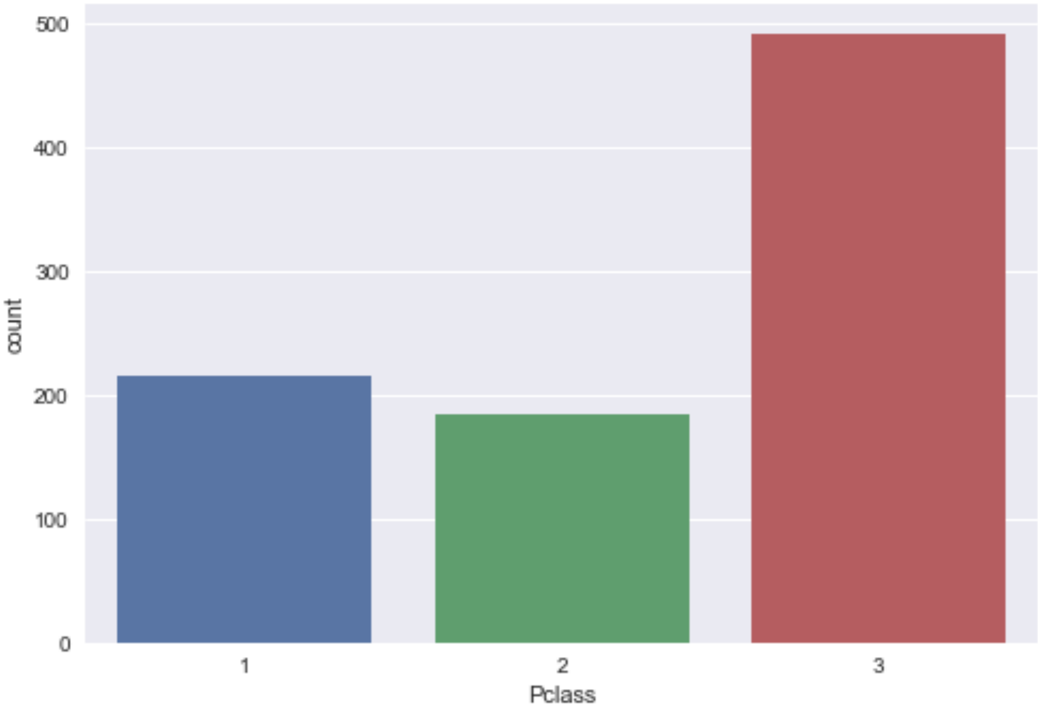
Draw counplot of Sex attribute splitting on Survived class. Clearly, women were given priority while rescuing. ~71% of feamles were saved as compared to ~17 % of males.

In [7]:
```
sns.countplot('Sex', hue = 'Survived', data = dataset)
plt.title('Sex vs Survived')
plt.show()
```



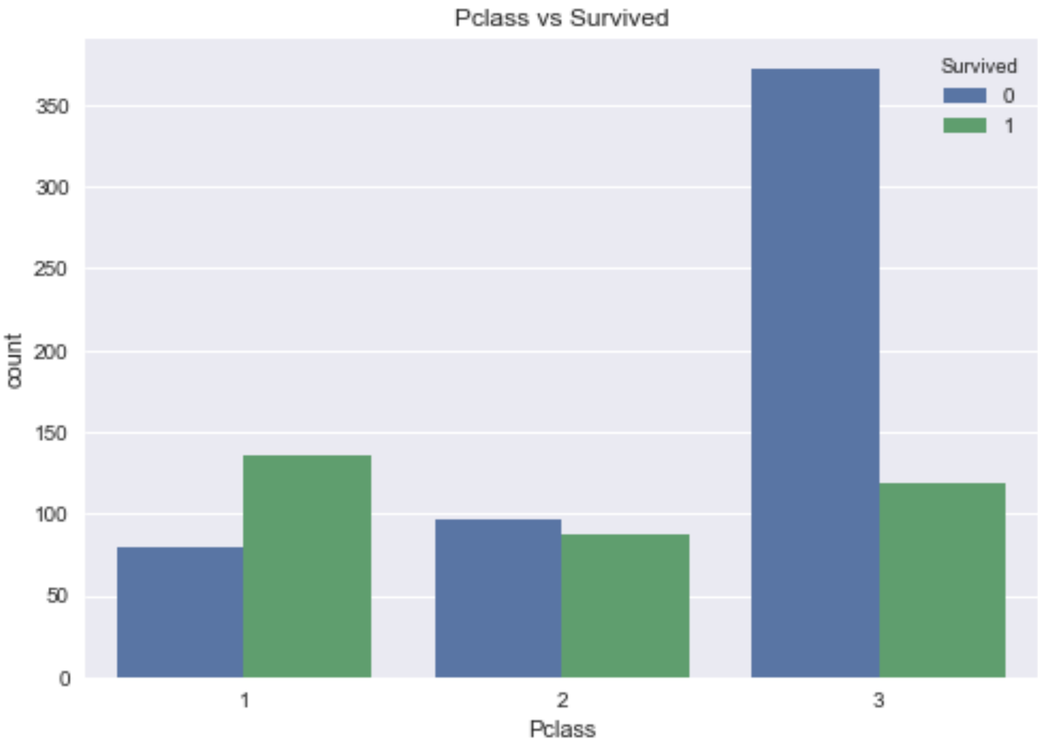
Draw countplot to show distribution of Passenger class in the dataset. Clearly, most tickets were class 3.

In [8]:
```
sns.countplot(x = 'Pclass', data = dataset)
plt.show()
```



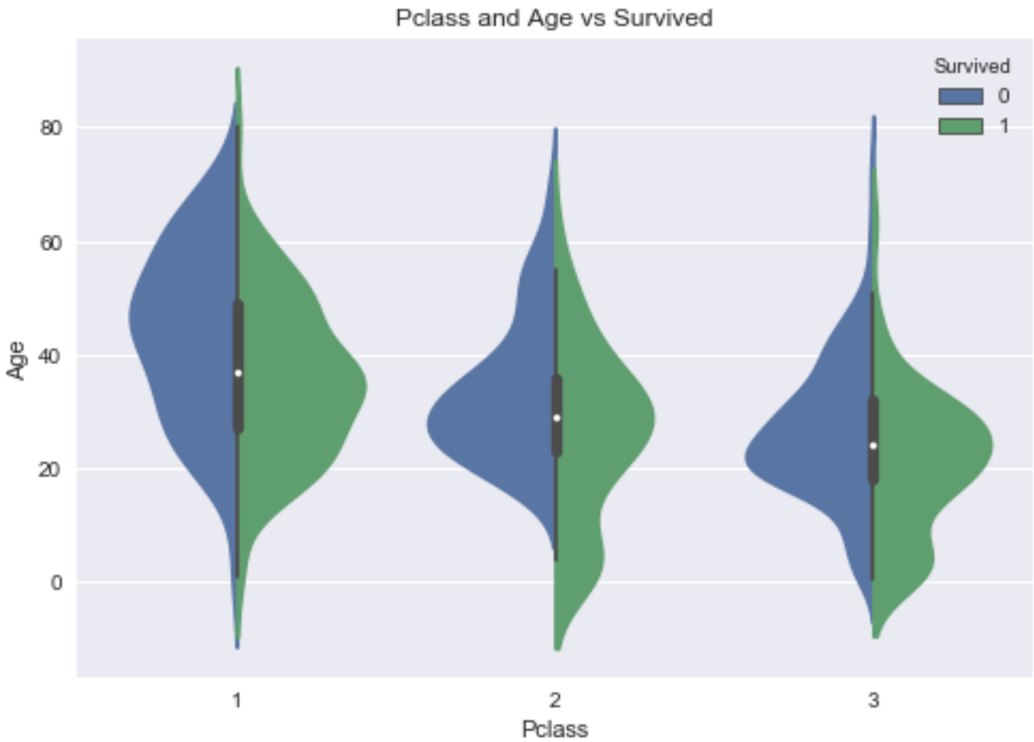
Draw countplot for passenger class splitting on survived. Clearly, class 1 passengers were given priority. Most class 3 passengers did not survive. Class 2 passengers were at a 50-50 chance of surviving.

In [9]:
```
sns.countplot('Pclass', hue = 'Survived', data = dataset)
plt.title('Pclass vs Survived')
plt.show()
```



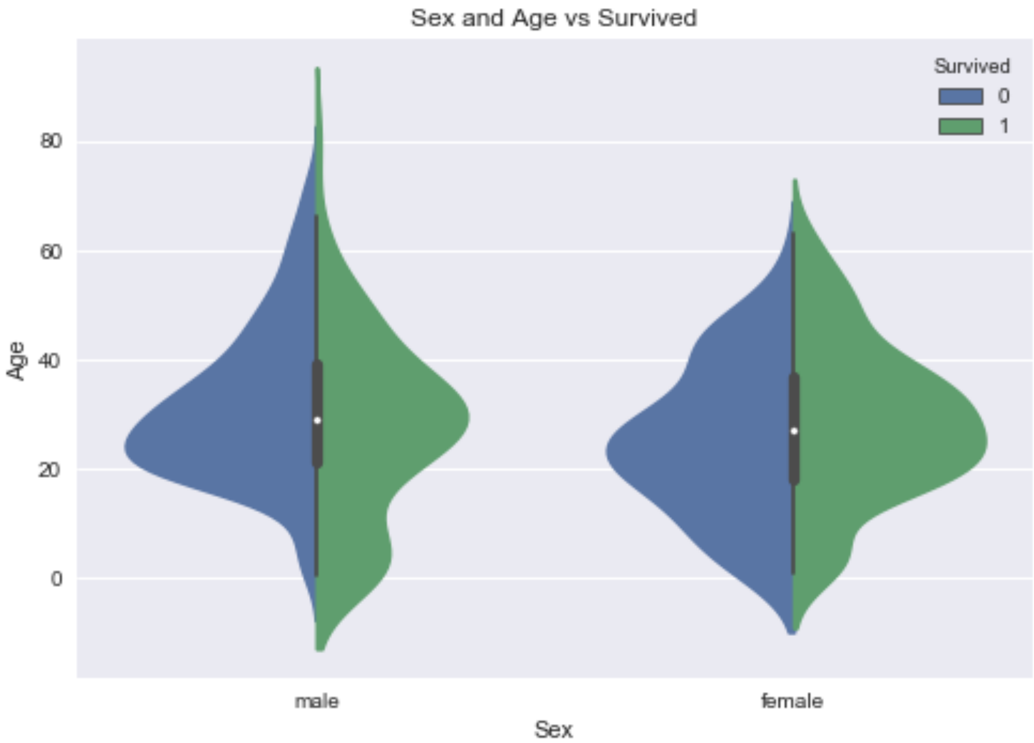
Draw violin plot of Age and passenger class splitting on survived. Most people survived in class 1 were middle-aged and most people survived in class 3 were young.

In [10]:
```
sns.violinplot("Pclass","Age", hue = "Survived", data = dataset, split
plt.title('Pclass and Age vs Survived')
plt.show()
```



Draw a violin plot of Sex and Age splitting on survived. Most men who survived were around ~30 years of age. and most women who survived were ~24 years of age. Lot of young men and women did not survive.

In [11]:
```
sns.violinplot("Sex","Age", hue = "Survived", data = dataset, split = '
plt.title('Sex and Age vs Survived')
plt.show()
```

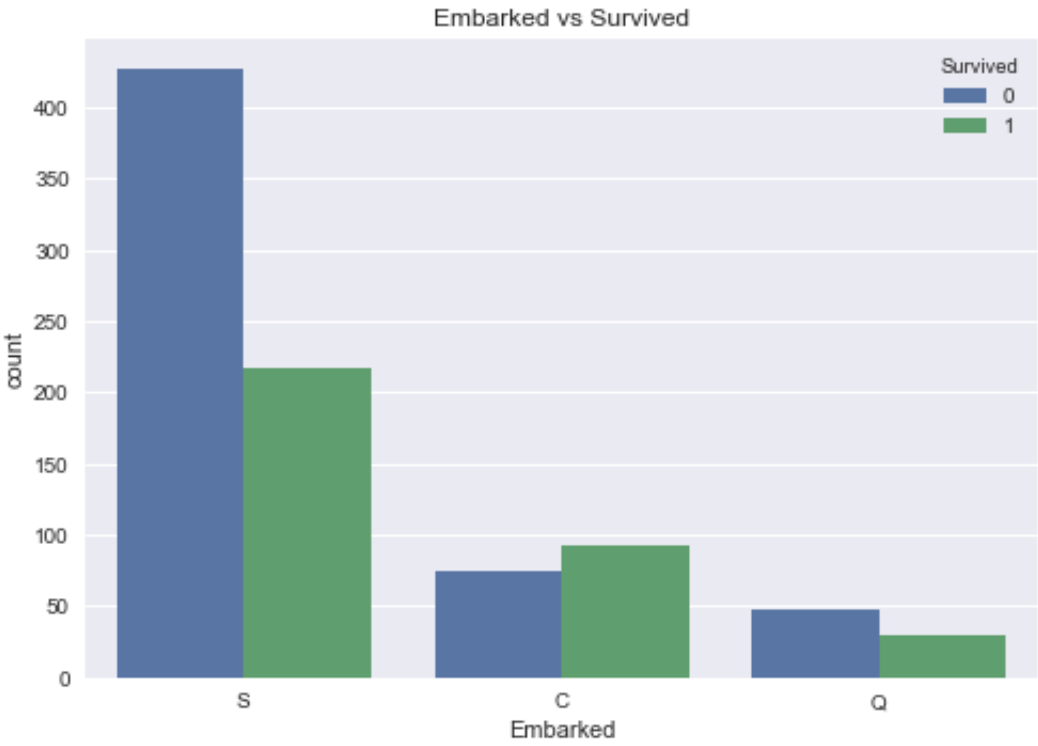


Fill the missing age values with the mean age in the dataset.

In [12]:
```
dataset['Age'] = dataset['Age'].fillna(dataset['Age'].mean())
```

Draw countplot of embarked attribute splitting on survived. Clearly, majority of people embarked at S and did not survive.

In [13]:
```
sns.countplot('Embarked',hue = 'Survived', data = dataset)
plt.title('Embarked vs Survived')
plt.show()
```

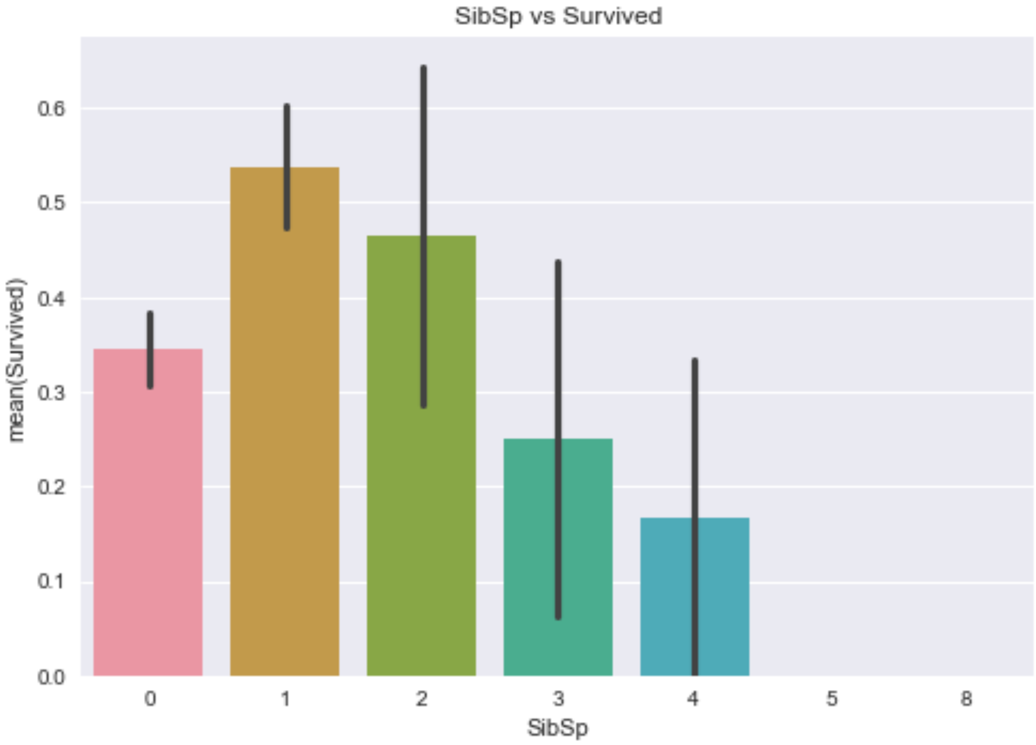


Fill missing embarked values with 'S' since most people embarked at S.

In [14]:
```
dataset['Embarked'].fillna('S', inplace = True)
```

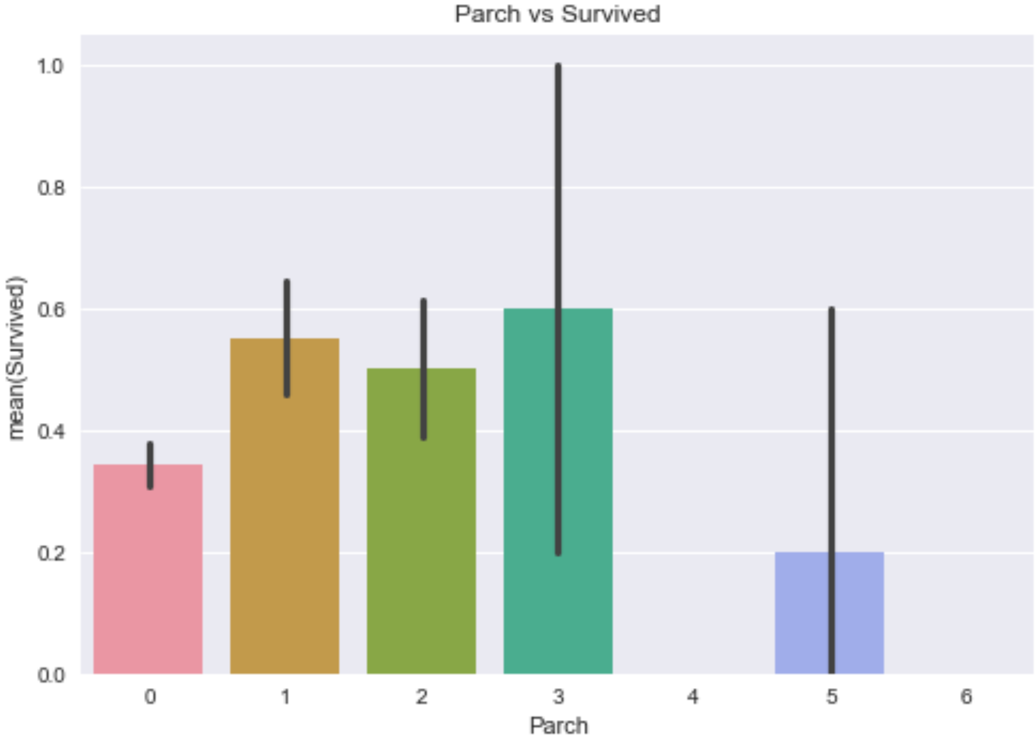
Bar chart for Siblings/Family splitting on survived. Interestingly, no people with a large family (>4) survived.

In [15]:
```
sns.barplot('SibSp', 'Survived', data = dataset)
plt.title('SibSp vs Survived')
plt.show()
```



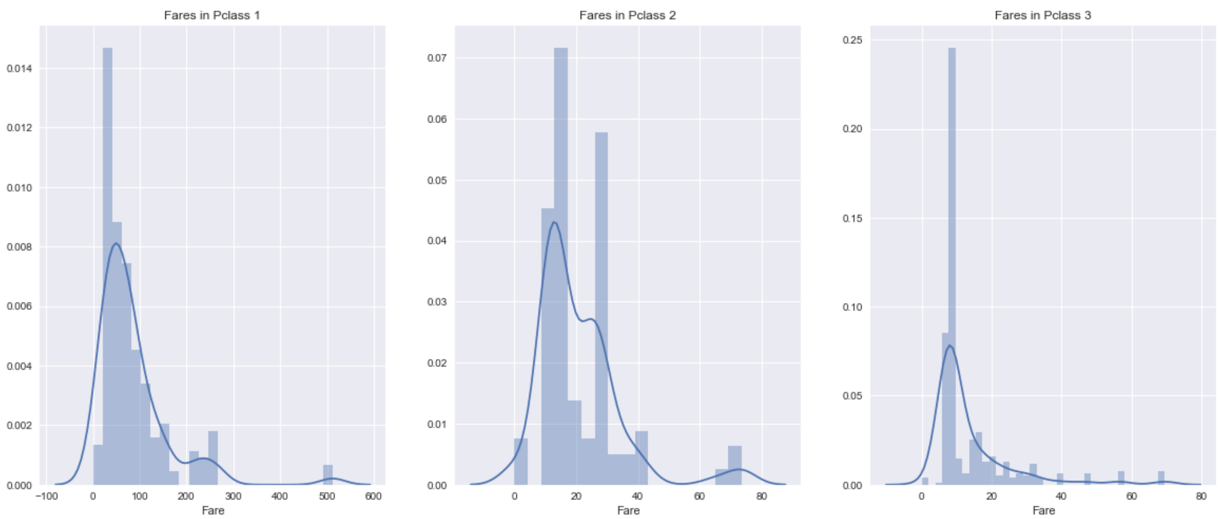
Draw Bar chart for Parent/Child splitting on survived. Similar to previous chart, people with higher number of family members did not survive.

In [16]:
```
sns.barplot('Parch', 'Survived', data = dataset)
plt.title('Parch vs Survived')
plt.show()
```



Draw Distribution chart to show fare distribution among passengers.

In [17]:
```
f, ax = plt.subplots(1, 3, figsize = (20, 8))
sns.distplot(dataset[dataset['Pclass'] == 1].Fare, ax = ax[0])
ax[0].set_title('Fares in Pclass 1')
sns.distplot(dataset[dataset['Pclass'] == 2].Fare, ax = ax[1])
ax[1].set_title('Fares in Pclass 2')
sns.distplot(dataset[dataset['Pclass'] == 3].Fare, ax = ax[2])
ax[2].set_title('Fares in Pclass 3')
plt.show()
```

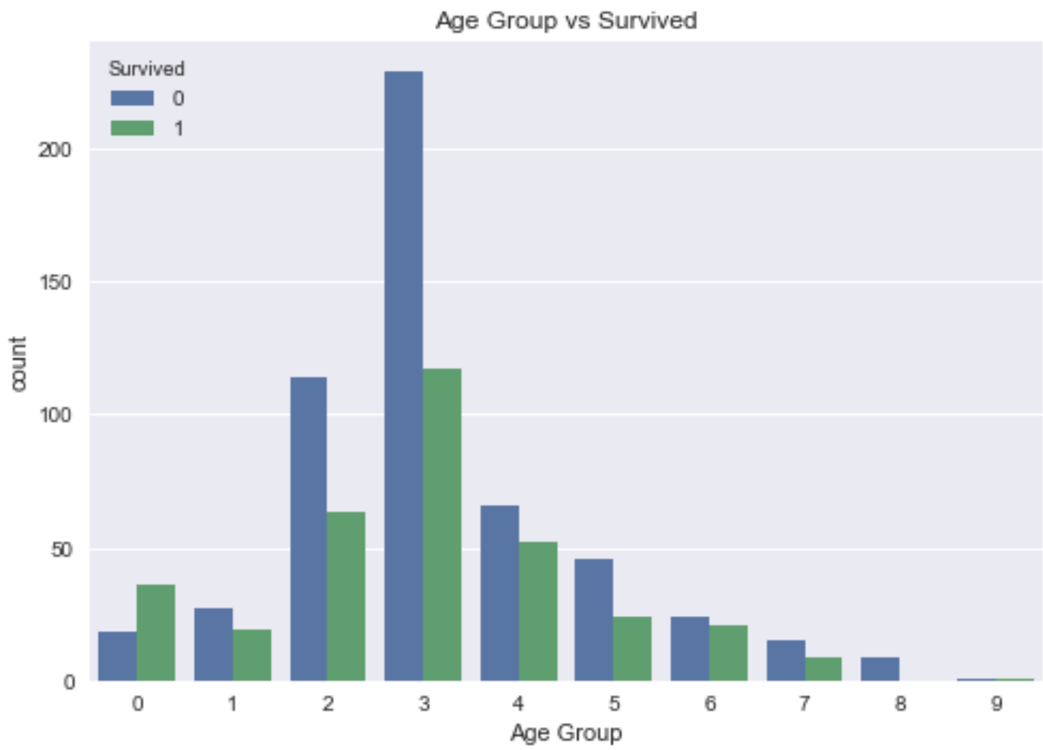


Since Age and Fare are continous attributes, it would be difficult to deal with them in a classification problem. Hence, split them into categories of 10 and 5 respectively. Since both age and fare will have a relative order, these is justified.

In [18]:
```
dataset['Age Group'] = pd.cut(dataset['Age'], 10, labels = range(0,10))
dataset['Fare Group'] = pd.cut(dataset['Fare'], 5, labels = range(0,5))
```

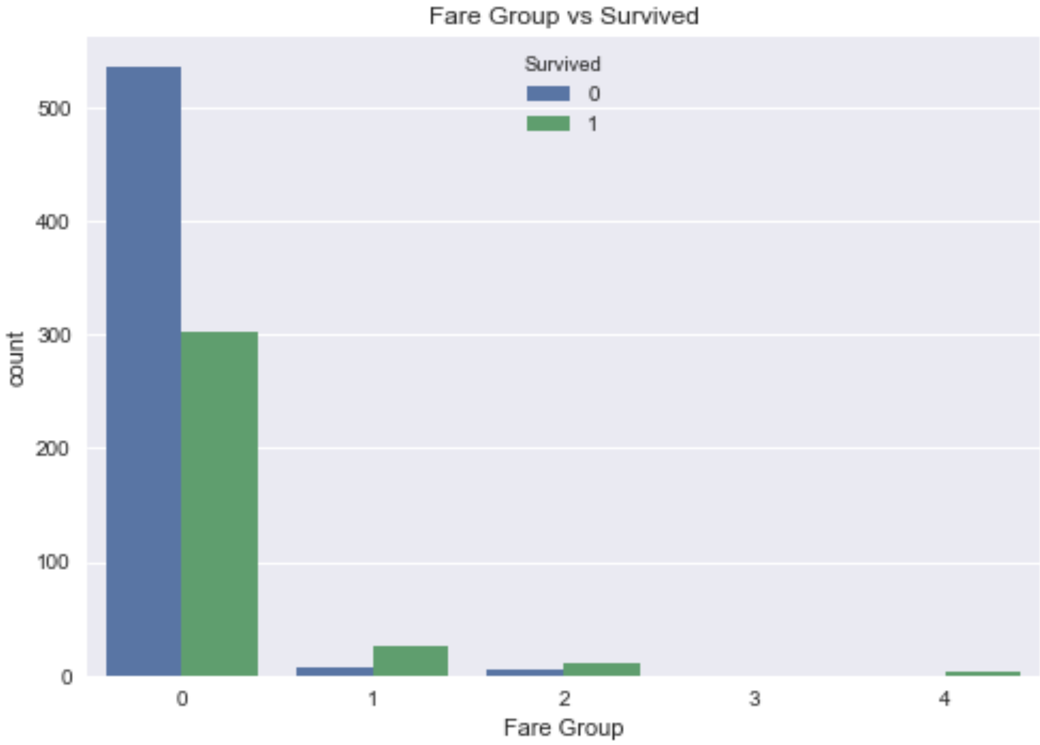
Most people in Age group 3 did not survive. A good estimate should tell us that age grouo 3 would correspond to young adults.

In [19]:
```
sns.countplot('Age Group',hue = 'Survived', data = dataset)
plt.title('Age Group vs Survived')
plt.show()
```



Since there was a relation between passenger class and survival rate, we see the same relation here. Most people in fare class 0 (most probably passenger class 3) did not survive.

In [20]:
```
sns.countplot('Fare Group',hue = 'Survived', data = dataset)
plt.title('Fare Group vs Survived')
plt.show()
```



Convert Sex and Embarked attributes to numeric/categorical for easy processing.

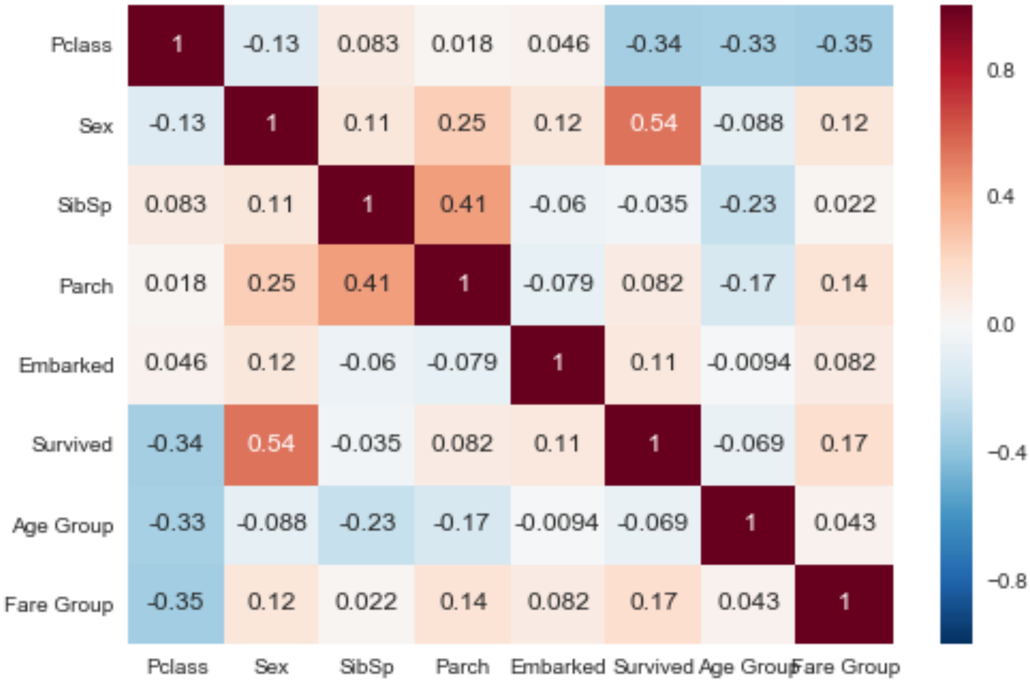
In [21]:
```
dataset['Sex'].replace(['male', 'female'], [0, 1], inplace = True)
dataset['Embarked'].replace(['S', 'C', 'Q'], [0, 1, 2], inplace = True
```

In [22]:
```
dataset['Fare Group'] = dataset['Fare Group'].astype(int)
dataset['Age Group'] = dataset['Age Group'].astype(int)
```

Name, Passenger ID, Cabin and Ticket are unimportant steing features which we cannot process. So, delete them. We have taken care of Age and Fare, so delete them.

In [23]:
```
del dataset['Name']
del dataset['Age']
del dataset['Fare']
del dataset['PassengerId']
del dataset['Ticket']
del dataset['Cabin']
```

In [24]:
```
sns.heatmap(dataset.corr(),annot = True)
plt.show()
```



In [25]:
```
train, test = train_test_split(dataset, test_size = 0.25, random_state
```

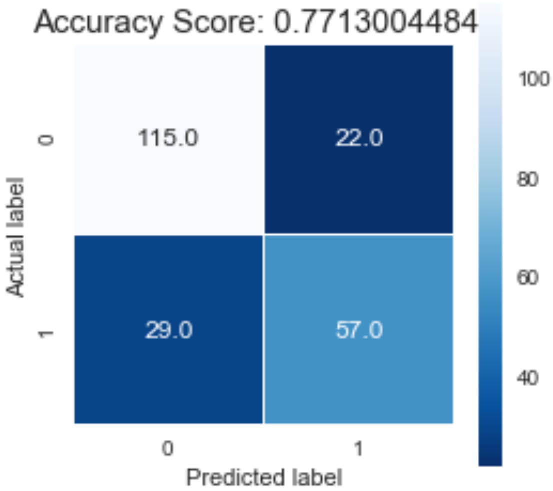
In [26]:
```
train_X = train[['Pclass', 'Sex', 'Age Group', 'Fare Group', 'SibSp',
train_y = train['Survived']
test_X = test[['Pclass', 'Sex', 'Age Group', 'Fare Group', 'SibSp', 'E
test_y = test['Survived']
```

Perform classification with Linear SVM, LogisticRegression, DecisionTreeClassifier, KNN and GaussianNB and draw resulting confusion matrix.

In [27]:
```
clf = SVC(kernel='linear',C = 0.2, gamma = 0.2)
clf.fit(train_X, train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy SVM: ' + str(metrics.accuracy_score(pred_clf, test_y))
```

```
Accuracy SVM: 0.77130044843
```

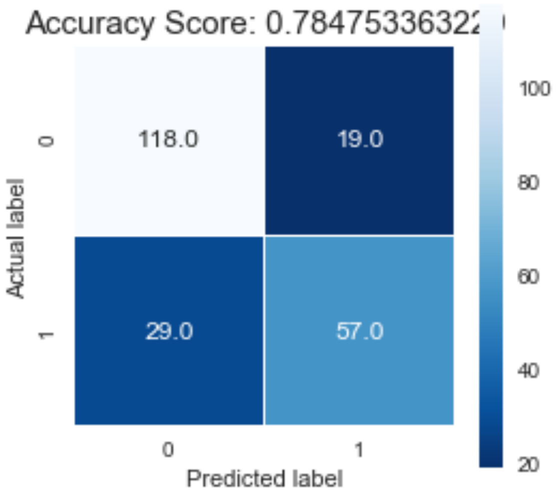
In [28]:
```
cm_SVC = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_SVC, annot = True, fmt = ".1f", linewidths = .5, square
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



In [29]:
```
clf = LogisticRegression()
clf.fit(train_X, train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy Logistic Regression: ' + str(metrics.accuracy_score(pre
```

```
Accuracy Logistic Regression: 0.784753363229
```

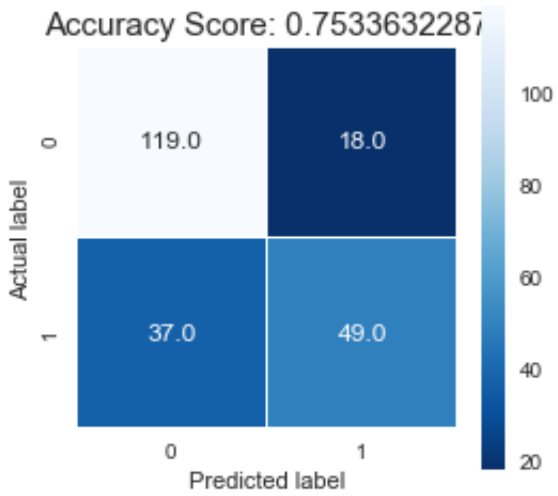
In [30]:
```
cm_LR = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_LR, annot = True, fmt = ".1f", linewidths = .5, square =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



In [31]:
```
clf = DecisionTreeClassifier()
clf.fit(train_X, train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy Decision Tree: ' + str(metrics.accuracy_score(pred_clf
```

```
Accuracy Decision Tree: 0.7533632287
```

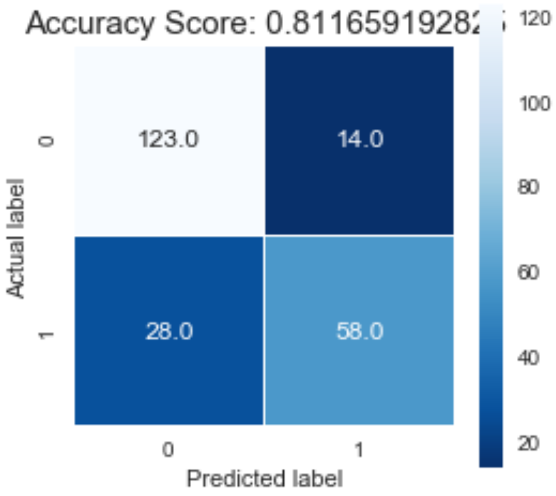
In [32]:
```
cm_DT = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_DT, annot = True, fmt = ".1f", linewidths = .5, square =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



In [33]:
```
clf = KNeighborsClassifier(5)
clf.fit(train_X,train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy KNN: ' + str(metrics.accuracy_score(pred_clf, test_y))
```

```
Accuracy KNN: 0.811659192825
```

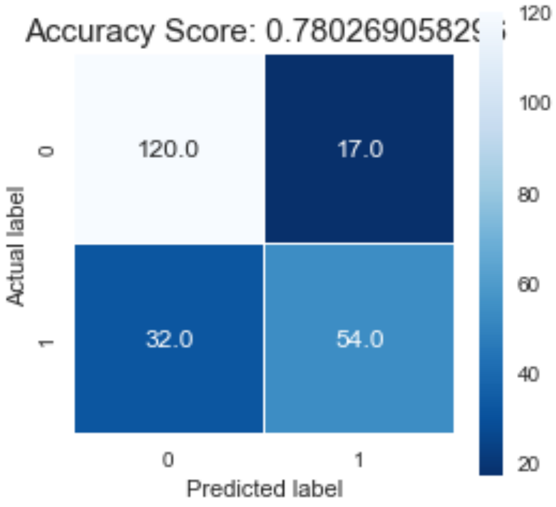
In [34]:
```
cm_KNN = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_KNN, annot = True, fmt = ".1f", linewidths = .5, square
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



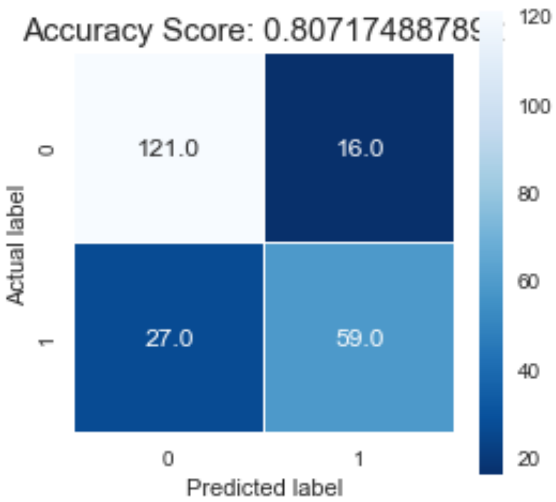
In [35]:
```
clf = GaussianNB()
clf.fit(train_X, train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy NaiveBayes: ' + str(metrics.accuracy_score(pred_clf,te
```

```
Accuracy NaiveBayes: 0.780269058296
```

In [36]:
```
cm_NB = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_NB, annot = True, fmt = ".1f", linewidths = .5, square =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



In [38]:
```
cm_P = metrics.confusion_matrix(test_y, pred_clf)
plt.figure(figsize = (4,4))
sns.heatmap(cm_P, annot = True, fmt = ".1f", linewidths = .5, square =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score
plt.title(all_sample_title, size = 15);
plt.show()
```



In [59]:
```
clf = MLPClassifier(hidden_layer_sizes=(200,200), activation='relu', ma
clf.fit(train_X, train_y)
pred_clf = clf.predict(test_X)
print 'Accuracy Perceptron: ' + str(metrics.accuracy_score(pred_clf,te
```

```
Accuracy Perceptron: 0.80269058296
```

Explored the dataset with machine learning. Now testing and reporting real prediction on Kaggle.

In [39]:
```
test_data = pd.read_csv('test.csv')
```

In [40]:
```
test_data.isnull().sum()
```

Out[40]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [41]:
```
del test_data['Name']
del test_data['Cabin']
del test_data['Ticket']
```

In [42]:
```
test_data['Age'] = test_data['Age'].fillna(test_data['Age'].mean())
test_data['Fare'] = test_data['Fare'].fillna(test_data['Fare'].mean())
```

In [43]:
```
test_data['Sex'].replace(['male', 'female'], [0, 1], inplace = True)
test_data['Embarked'].replace(['S', 'C', 'Q'], [0, 1, 2], inplace = Tr
```

In [44]:
```
test_data['Age Group'] = pd.cut(test_data['Age'], 10, labels = range(0
test_data['Fare Group'] = pd.cut(test_data['Fare'], 5, labels = range(
```

In [45]:
```
del test_data['Age']
del test_data['Fare']
```

In [46]:
```
X = test_data[['Pclass', 'Sex', 'Age Group', 'Fare Group', 'SibSp', 'Em
```

In [47]:
```
clf = MLPClassifier()
clf.fit(train_X,train_y)
```

Out[47]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', be
ta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=
0.1,
       verbose=False, warm_start=False)
```

In [48]:
```
test_data['Survived'] = clf.predict(X)
```

In [49]:
```
results = test_data[['PassengerId','Survived']]
```

In [50]:
```
results
```

Out[50]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	0
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0
10	902	0
11	903	0
12	904	1
13	905	0
14	906	1
15	907	1
16	908	0
17	909	0
18	910	0
19	911	1

20	912	0
21	913	0
22	914	1
23	915	0
24	916	1
25	917	0
26	918	1
27	919	0
28	920	0
29	921	0
...	...	...
388	1280	0
389	1281	0
390	1282	0
391	1283	1
392	1284	1
393	1285	0
394	1286	0
395	1287	1
396	1288	0
397	1289	1
398	1290	0
399	1291	0
400	1292	1
401	1293	0
402	1294	1
403	1295	0
404	1296	0
405	1297	0
406	1298	0
407	1299	0
408	1300	1
409	1301	1
410	1302	1

411	1303	1
412	1304	1
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [51]:
```
results.to_csv('results.csv')
```

Submitted predictions on Kaggle with an accuracy of 0.78468

In [ ]:
```
```