In [1]:

```
#data analysis libraries
import numpy as np
import pandas as pd

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#import train and test CSV files
train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")

#take a look at the training data
train.describe(include="all")
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147
top	NaN	NaN	NaN	Duff Gordon, Lady. (Lucille Christiana Sutherl...	male	NaN	NaN	NaN	1601	NaN	B96 B98
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN

In [3]:

```
#get a list of the features within the dataset
print(train.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [4]:

```
#see a sample of the dataset to get an idea of the variables
train.sample(5)
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
529	530	0	2	Hocking, Mr. Richard George	male	23.0	2	1	29104	11.5000	NaN	S
870	871	0	3	Balkic, Mr. Cerin	male	26.0	0	0	349248	7.8958	NaN	S
20	21	0	2	Fynney, Mr. Joseph J	male	35.0	0	0	239865	26.0000	NaN	S
152	153	0	3	Meo, Mr. Alfonzo	male	55.5	0	0	A.5. 11206	8.0500	NaN	S
477	478	0	3	Braund, Mr. Lewis Richard	male	29.0	1	0	3460	7.0458	NaN	S

In [5]:

```
#see a summary of the training dataset
train.describe(include = "all")
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147
top	NaN	NaN	NaN	Duff Gordon, Lady. (Lucille Christiana Sutherl...	male	NaN	NaN	NaN	1601	NaN	B96 B98
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN

In [6]:

```
#check for any other unusable values
print(pd.isnull(train).sum())
```

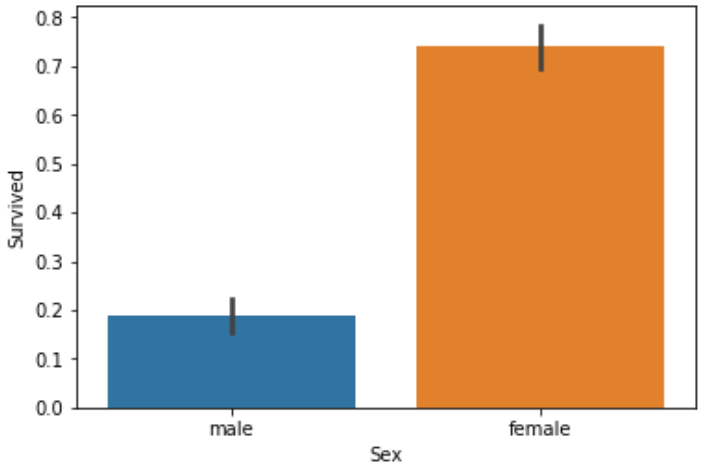
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [7]:

```
#draw a bar plot of survival by sex
sns.barplot(x="Sex", y="Survived", data=train)

#print percentages of females vs. males that survive
```

```
print("Percentage of females who survived:", train["Survived"][train["Sex"] == 'female'].
value_counts(normalize = True)[1]*100)

print("Percentage of males who survived:", train["Survived"][train["Sex"] == 'male'].val
ue_counts(normalize = True)[1]*100)
```

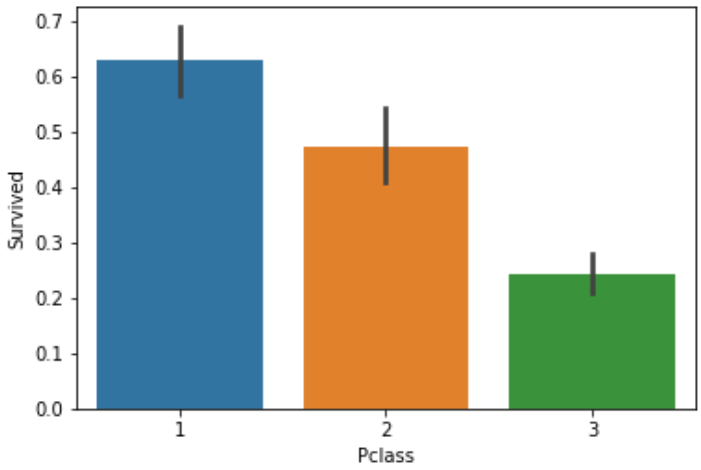
```
Percentage of females who survived: 74.20382165605095
Percentage of males who survived: 18.890814558058924
```



In [8]:

```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#print percentage of people by Pclass that survived
print("Percentage of Pclass = 1 who survived:", train["Survived"][train["Pclass"] == 1].
value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 2 who survived:", train["Survived"][train["Pclass"] == 2].
value_counts(normalize = True)[1]*100)

print("Percentage of Pclass = 3 who survived:", train["Survived"][train["Pclass"] == 3].
value_counts(normalize = True)[1]*100)
```

```
Percentage of Pclass = 1 who survived: 62.96296296296296
Percentage of Pclass = 2 who survived: 47.28260869565217
Percentage of Pclass = 3 who survived: 24.236252545824847
```

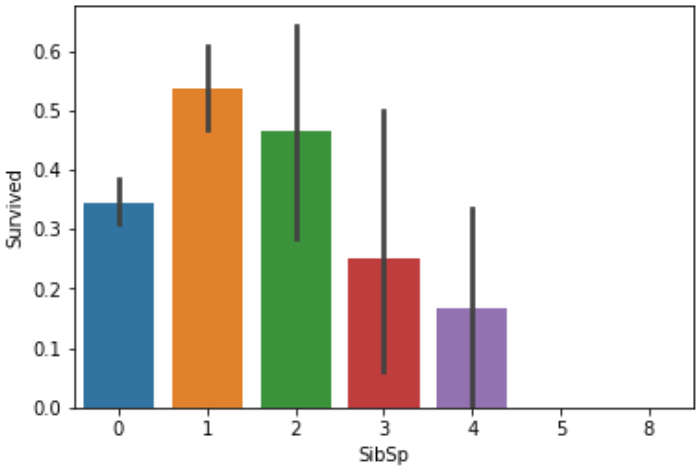


In [9]:

```
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

#I won't be printing individual percent values for all of these.
print("Percentage of SibSp = 0 who survived:", train["Survived"][train["SibSp"] == 0].va
lue_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 1 who survived:", train["Survived"][train["SibSp"] == 1].va
```

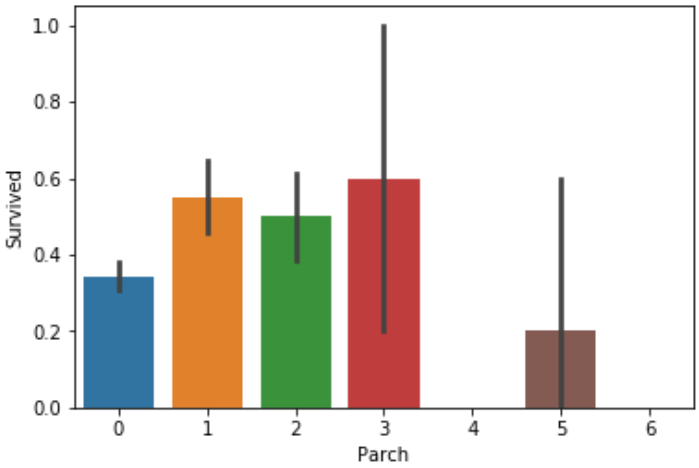
```
lue_counts(normalize = True)[1]*100)

print("Percentage of SibSp = 2 who survived:", train["Survived"][train["SibSp"] == 2].va
lue_counts(normalize = True)[1]*100)
```

```
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```



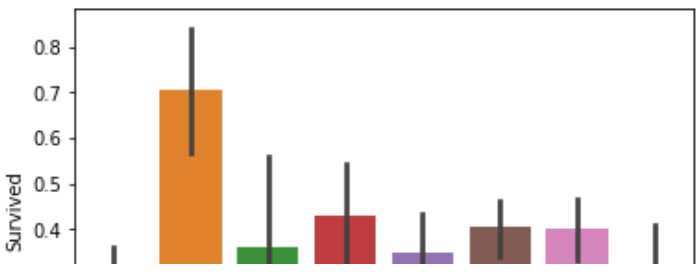
In [10]:

```
#draw a bar plot for Parch vs. survival
sns.barplot(x="Parch", y="Survived", data=train)
plt.show()
```



In [11]:

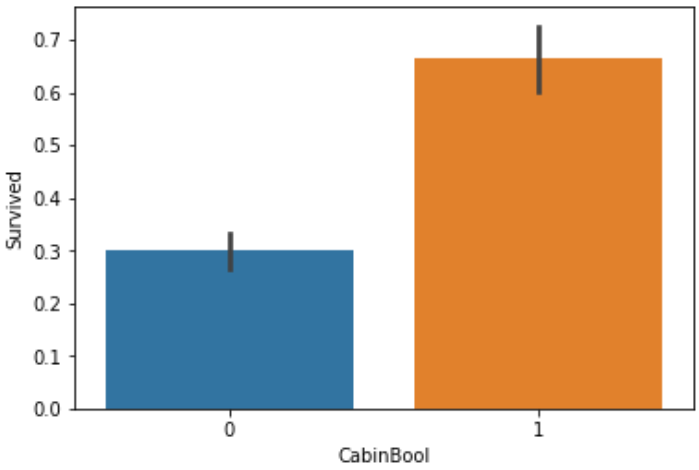
```
#sort the ages into logical categories
train["Age"] = train["Age"].fillna(-0.5)
test["Age"] = test["Age"].fillna(-0.5)
bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]
labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult', 'Se
nior']
train['AgeGroup'] = pd.cut(train["Age"], bins, labels = labels)
test['AgeGroup'] = pd.cut(test["Age"], bins, labels = labels)

#draw a bar plot of Age vs. survival
sns.barplot(x="AgeGroup", y="Survived", data=train)
plt.show()
```



In [12]:

```
train["CabinBool"] = (train["Cabin"].notnull().astype('int'))
test["CabinBool"] = (test["Cabin"].notnull().astype('int'))

#calculate percentages of CabinBool vs. survived
print("Percentage of CabinBool = 1 who survived:", train["Survived"][train["CabinBool"]
== 1].value_counts(normalize = True)[1]*100)

print("Percentage of CabinBool = 0 who survived:", train["Survived"][train["CabinBool"]
== 0].value_counts(normalize = True)[1]*100)
#draw a bar plot of CabinBool vs. survival
sns.barplot(x="CabinBool", y="Survived", data=train)
plt.show()
```

```
Percentage of CabinBool = 1 who survived: 66.66666666666666
Percentage of CabinBool = 0 who survived: 29.985443959243085
```



In [13]:

```
test.describe(include="all")
```

Out[13]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	418.000000	418.000000	418	418	418.000000	418.000000	418.000000	418	417.000000	91	418
unique	NaN	NaN	418	2	NaN	NaN	NaN	363	NaN	76	3
top	NaN	NaN	Svensson, Mr. Johan Cervin	male	NaN	NaN	NaN	PC 17608	NaN	B57 B59 B63 B66	S
freq	NaN	NaN	1	266	NaN	NaN	NaN	5	NaN	3	270
mean	1100.500000	2.265550	NaN	NaN	23.941388	0.447368	0.392344	NaN	35.627188	NaN	NaN
std	120.810458	0.841838	NaN	NaN	17.741080	0.896760	0.981429	NaN	55.907576	NaN	NaN
min	892.000000	1.000000	NaN	NaN	-0.500000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	996.250000	1.000000	NaN	NaN	9.000000	0.000000	0.000000	NaN	7.895800	NaN	NaN
50%	1100.500000	3.000000	NaN	NaN	24.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	1204.750000	3.000000	NaN	NaN	35.750000	1.000000	0.000000	NaN	31.500000	NaN	NaN
max	1309.000000	3.000000	NaN	NaN	76.000000	8.000000	9.000000	NaN	512.329200	NaN	NaN

In [14]:

```
#we'll start off by dropping the Cabin feature since not a lot more useful information ca
n be extracted from it.
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

In [15]:

```
#we can also drop the Ticket feature since it's unlikely to yield any useful information
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
```

In [16]:

```
#now we need to fill in the missing values in the Embarked feature
print("Number of people embarking in Southampton (S):")
southampton = train[train["Embarked"] == "S"].shape[0]
print(southampton)

print("Number of people embarking in Cherbourg (C):")
cherbourg = train[train["Embarked"] == "C"].shape[0]
print(cherbourg)

print("Number of people embarking in Queenstown (Q):")
queenstown = train[train["Embarked"] == "Q"].shape[0]
print(queenstown)
```

```
Number of people embarking in Southampton (S):
644
Number of people embarking in Cherbourg (C):
168
Number of people embarking in Queenstown (Q):
77
```

In [17]:

```
#replacing the missing values in the Embarked feature with S
train = train.fillna({"Embarked": "S"})
```

In [18]:

```
#create a combined group of both datasets
combine = [train, test]

#extract a title for each Name in the train and test datasets
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train['Title'], train['Sex'])
```

Out[18]:

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40

Sex	female	male
Miss	182	0
Title Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

In [19]:

```
#replace various titles with more common names
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Capt', 'Col',
    'Don', 'Dr', 'Major', 'Rev', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace(['Countess', 'Lady', 'Sir'], 'Royal')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

Out[19]:

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.285714
5	Royal	1.000000

In [20]:

```
#map each of the title groups to a numerical value
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Royal": 5, "Rare": 6}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train.head()
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	Student	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	Adult	1	3
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	Young Adult	0	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily	female	35.0	1	0	53.1000	S	Young Adult	1	3

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
4	**5**	**0**	**3**	**Allen, Mr. William Henry**	**male**	**35.0**	**0**	**0**	**8.0500**	**S**	**Young Adult**	**0**	**1**

In [21]:

```
# fill missing age with mode age group for each title
mr_age = train[train["Title"] == 1]["AgeGroup"].mode() #Young Adult
miss_age = train[train["Title"] == 2]["AgeGroup"].mode() #Student
mrs_age = train[train["Title"] == 3]["AgeGroup"].mode() #Adult
master_age = train[train["Title"] == 4]["AgeGroup"].mode() #Baby
royal_age = train[train["Title"] == 5]["AgeGroup"].mode() #Adult
rare_age = train[train["Title"] == 6]["AgeGroup"].mode() #Adult

age_title_mapping = {1: "Young Adult", 2: "Student", 3: "Adult", 4: "Baby", 5: "Adult",
6: "Adult"}

#I tried to get this code to work with using .map(), but couldn't.
#I've put down a less elegant, temporary solution for now.
#train = train.fillna({"Age": train["Title"].map(age_title_mapping)})
#test = test.fillna({"Age": test["Title"].map(age_title_mapping)})

for x in range(len(train["AgeGroup"])):
    if train["AgeGroup"][x] == "Unknown":
        train["AgeGroup"][x] = age_title_mapping[train["Title"][x]]

for x in range(len(test["AgeGroup"])):
    if test["AgeGroup"][x] == "Unknown":
        test["AgeGroup"][x] = age_title_mapping[test["Title"][x]]
```

In [22]:

```
#map each Age value to a numerical value
age_mapping = {'Baby': 1, 'Child': 2, 'Teenager': 3, 'Student': 4, 'Young Adult': 5, 'Ad
ult': 6, 'Senior': 7}
train['AgeGroup'] = train['AgeGroup'].map(age_mapping)
test['AgeGroup'] = test['AgeGroup'].map(age_mapping)

train.head()

#dropping the Age feature for now, might change
train = train.drop(['Age'], axis = 1)
test = test.drop(['Age'], axis = 1)
```

In [23]:

```
#drop the name feature since it contains no more useful information.
train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)
```

In [24]:

```
#map each Sex value to a numerical value
sex_mapping = {"male": 0, "female": 1}
train['Sex'] = train['Sex'].map(sex_mapping)
test['Sex'] = test['Sex'].map(sex_mapping)

train.head()
```

Out[24]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
0	**1**	**0**	**3**	**0**	**1**	**0**	**7.2500**	**S**	**4**	**0**	**1**
1	**2**	**1**	**1**	**1**	**1**	**0**	**71.2833**	**C**	**6**	**1**	**3**
2	**3**	**1**	**3**	**1**	**0**	**0**	**7.9250**	**S**	**5**	**0**	**2**
3	**4**	**1**	**1**	**1**	**1**	**0**	**53.1000**	**S**	**5**	**1**	**3**

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
4	5	0	3	0	0	0	8.0500	S	5	0	1

In [25]:

```
#map each Embarked value to a numerical value
embarked_mapping = {"S": 1, "C": 2, "Q": 3}
train['Embarked'] = train['Embarked'].map(embarked_mapping)
test['Embarked'] = test['Embarked'].map(embarked_mapping)

train.head()
```

Out[25]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
0	1	0	3	0	1	0	7.2500	1	4	0	1
1	2	1	1	1	1	0	71.2833	2	6	1	3
2	3	1	3	1	0	0	7.9250	1	5	0	2
3	4	1	1	1	1	0	53.1000	1	5	1	3
4	5	0	3	0	0	0	8.0500	1	5	0	1

In [26]:

```
#fill in missing Fare value in test set based on mean fare for that Pclass
for x in range(len(test["Fare"])):
    if pd.isnull(test["Fare"][x]):
        pclass = test["Pclass"][x] #Pclass = 3
        test["Fare"][x] = round(train[train["Pclass"] == pclass]["Fare"].mean(), 4)

#map Fare values into groups of numerical values
train['FareBand'] = pd.qcut(train['Fare'], 4, labels = [1, 2, 3, 4])
test['FareBand'] = pd.qcut(test['Fare'], 4, labels = [1, 2, 3, 4])

#drop Fare values
train = train.drop(['Fare'], axis = 1)
test = test.drop(['Fare'], axis = 1)
```

In [27]:

```
#check train data
train.head()
```

Out[27]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup	CabinBool	Title	FareBand
0	1	0	3	0	1	0	1	4	0	1	1
1	2	1	1	1	1	0	2	6	1	3	4
2	3	1	3	1	0	0	1	5	0	2	2
3	4	1	1	1	1	0	1	5	1	3	4
4	5	0	3	0	0	0	1	5	0	1	2

In [28]:

```
#check test data
test.head()
```

Out[28]:

	PassengerId	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup	CabinBool	Title	FareBand
0	892	3	0	0	0	3	5	0	1	1
1	893	3	1	1	0	1	6	0	3	1
2	894	2	0	0	0	3	7	0	1	2

	PassengerId	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup	CabinBool	Title	FareBand
3	895	3	0	0	0	1	5	0	1	2
4	896	3	1	1	1	1	4	0	3	2

In [29]:

```
from sklearn.model_selection import train_test_split

predictors = train.drop(['Survived', 'PassengerId'], axis=1)
target = train["Survived"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.22,
random_state = 0)
```

In [30]:

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_val)
acc_gaussian = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gaussian)
```

```
78.68
```

In [31]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_logreg)
```

```
79.19
```

In [32]:

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_decisiontree = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_decisiontree)
```

```
80.2
```