Titanic Survival Predictions (Beginner)

I am a newbie to data science and machine learning, and will be attempting to work my way through the Titanic: Machine Learning from Disaster dataset. Please consider upvoting if this is useful to you! :)

Contents:

1) Import Necessary Libraries

First off, we need to import several Python libraries such as numpy, pandas, matplotlib and seaborn.

This code block imports necessary data analysis and visualization libraries for Python:

```
numpy (abbreviated as np) is a library for numerical
computing with Python. It provides powerful tools for working
with arrays and matrices, as well as various mathematical
functions.
pandas (abbreviated as pd) is a library for data manipulation
and analysis in Python. It provides data structures for
efficiently storing and analyzing data, such as data frames
and series.
matplotlib.pyplot (abbreviated as plt) is a plotting library
for Python that provides a variety of visualizations,
including line charts, scatter plots, histograms, and more.
seaborn is a data visualization library based on matplotlib.
It provides a higher-level interface for creating statistical
graphics.
%matplotlib inline is a magic command for Jupyter notebooks
that enables displaying of plots inline within the notebook.
```

The last code block warnings.filterwarnings('ignore') is used to suppress warning messages in the output.

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')
```

2) Read in and Explore the Data

It's time to read in our training and testing data using `pd.read_csv`, and take a first look at the training data using the `describe()` function.

In [2]:
```
#import train and test CSV files
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

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

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parc
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.00000
unique	NaN	NaN	NaN	891	2	NaN	NaN	Na
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	Na
freq	NaN	NaN	NaN	1	577	NaN	NaN	Na
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.38159
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.80605
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.00000
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.00000
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.00000
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.00000
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.00000

3) Data Analysis

We're going to consider the features in the dataset and how complete they are.

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]:
```python
#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	C
841	842	0	2	Mudd, Mr. Thomas Charles	male	16.0	0	0	S.O./P.P. 3	10.5000	
182	183	0	3	Asplund, Master. Clarence Gustaf Hugo	male	9.0	4	2	347077	31.3875	
234	235	0	2	Leyson, Mr. Robert William Norman	male	24.0	0	0	C.A. 29566	10.5000	
97	98	1	1	Greenfield, Mr. William Bertram	male	23.0	0	1	PC 17759	63.3583	
743	744	0	3	McNamee, Mr. Neal	male	24.0	1	0	376566	16.1000	

- **Numerical Features:** Age (Continuous), Fare (Continuous), SibSp (Discrete), Parch (Discrete)
- **Categorical Features:** Survived, Sex, Embarked, Pclass
- **Alphanumeric Features:** Ticket, Cabin

What are the data types for each feature?

- Survived: int
- Pclass: int
- Name: string
- Sex: string
- Age: float
- SibSp: int
- Parch: int
- Ticket: string
- Fare: float
- Cabin: string
- Embarked: string

Now that we have an idea of what kinds of features we're working with, we can see how much information we have about each of them.

In [5]:
```python
#see a summary of the training dataset
```

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

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parc
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.00000
unique	NaN	NaN	NaN	891	2	NaN	NaN	Na
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	Na
freq	NaN	NaN	NaN	1	577	NaN	NaN	Na
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.38159
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.80605
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.00000
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.00000
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.00000
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.00000
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.00000

Some Observations:

- There are a total of 891 passengers in our training set.
- The Age feature is missing approximately 19.8% of its values. I'm guessing that the Age feature is pretty important to survival, so we should probably attempt to fill these gaps.
- The Cabin feature is missing approximately 77.1% of its values. Since so much of the feature is missing, it would be hard to fill in the missing values. We'll probably drop these values from our dataset.
- The Embarked feature is missing 0.22% of its values, which should be relatively harmless.

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
```

We can see that except for the abovementioned missing values, no NaN values exist.

Some Predictions:

- Sex: Females are more likely to survive.
- SibSp/Parch: People traveling alone are more likely to survive.
- Age: Young children are more likely to survive.
- Pclass: People of higher socioeconomic class are more likely to survive.

4) Data Visualization

It's time to visualize our data so we can see whether our predictions were accurate!

Sex Feature

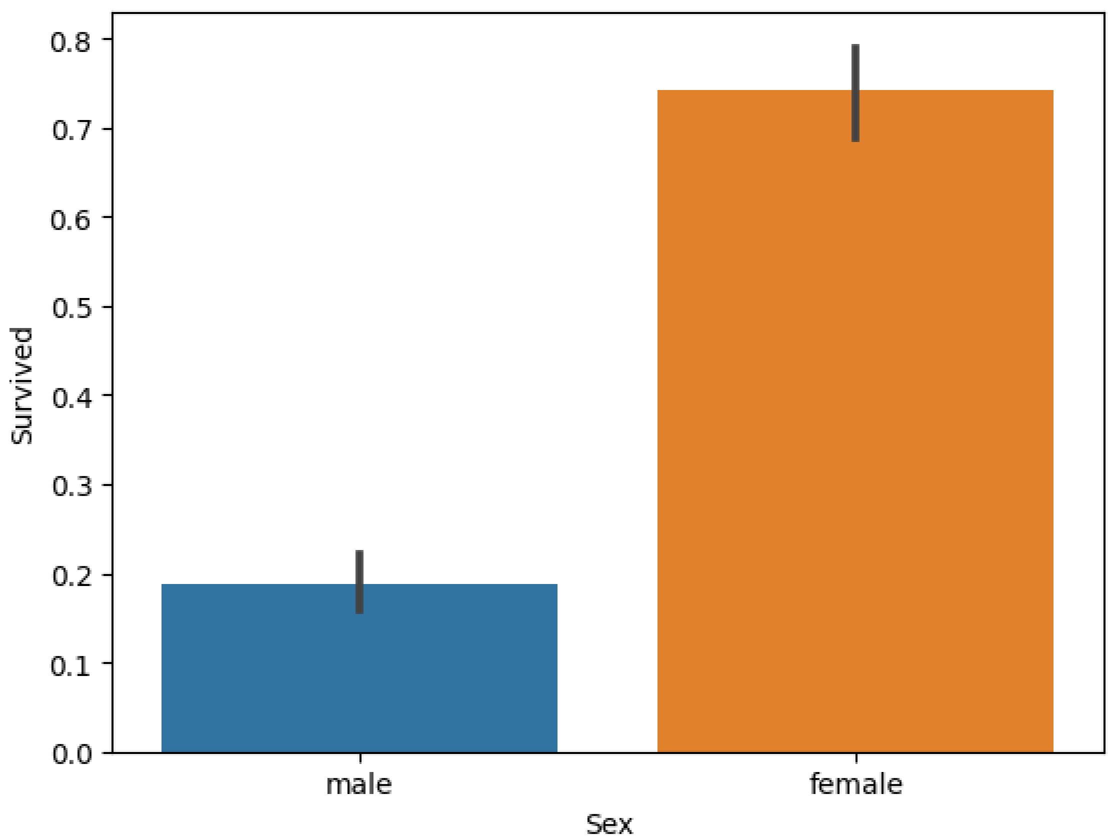
In [7]:

```
#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"]

print("Percentage of males who survived:", train["Survived"][train["Sex"] ==
```

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



As predicted, females have a much higher chance of survival than males. The Sex feature is essential in our predictions.

Pclass Feature

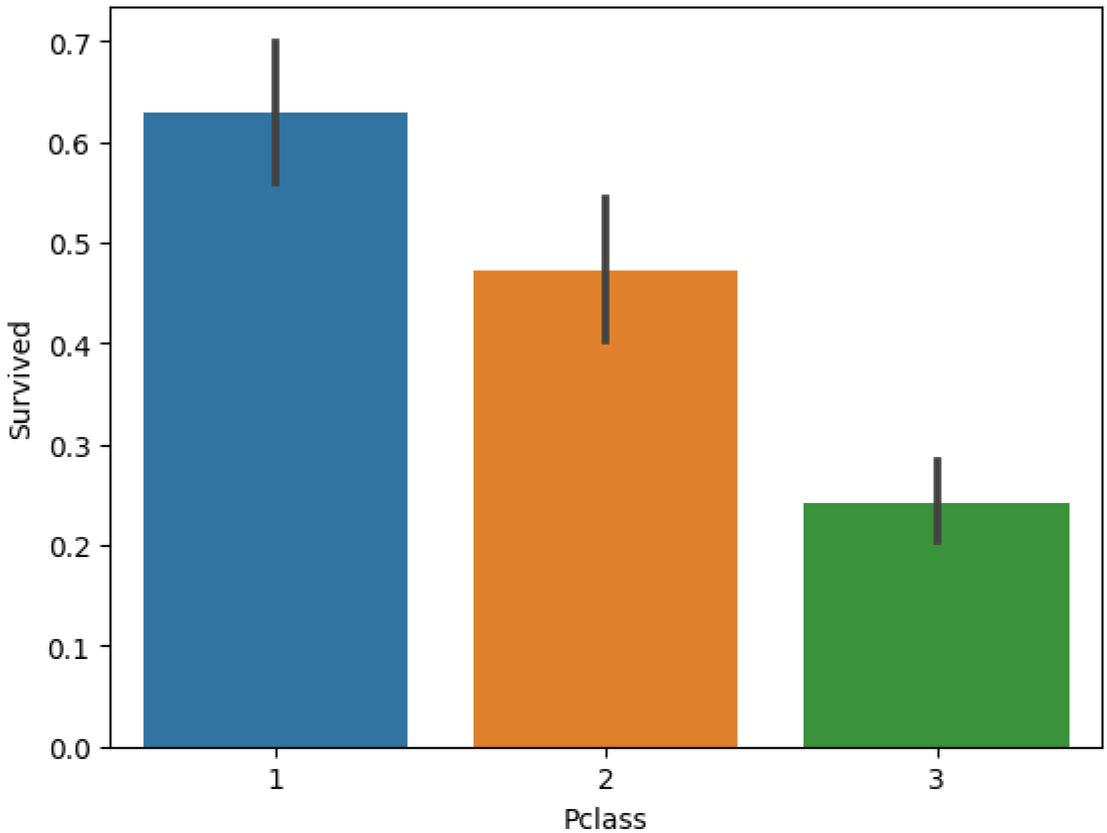
In [8]:
```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#print percentage of people by Pclass that survived
print("Percentage of Pclass = 1 who survived:", train["Survived"][train["Pcl

print("Percentage of Pclass = 2 who survived:", train["Survived"][train["Pcl

print("Percentage of Pclass = 3 who survived:", train["Survived"][train["Pcl
```

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



As predicted, people with higher socioeconomic class had a higher rate of survival. (62.9% vs. 47.3% vs. 24.2%)

SibSp Feature

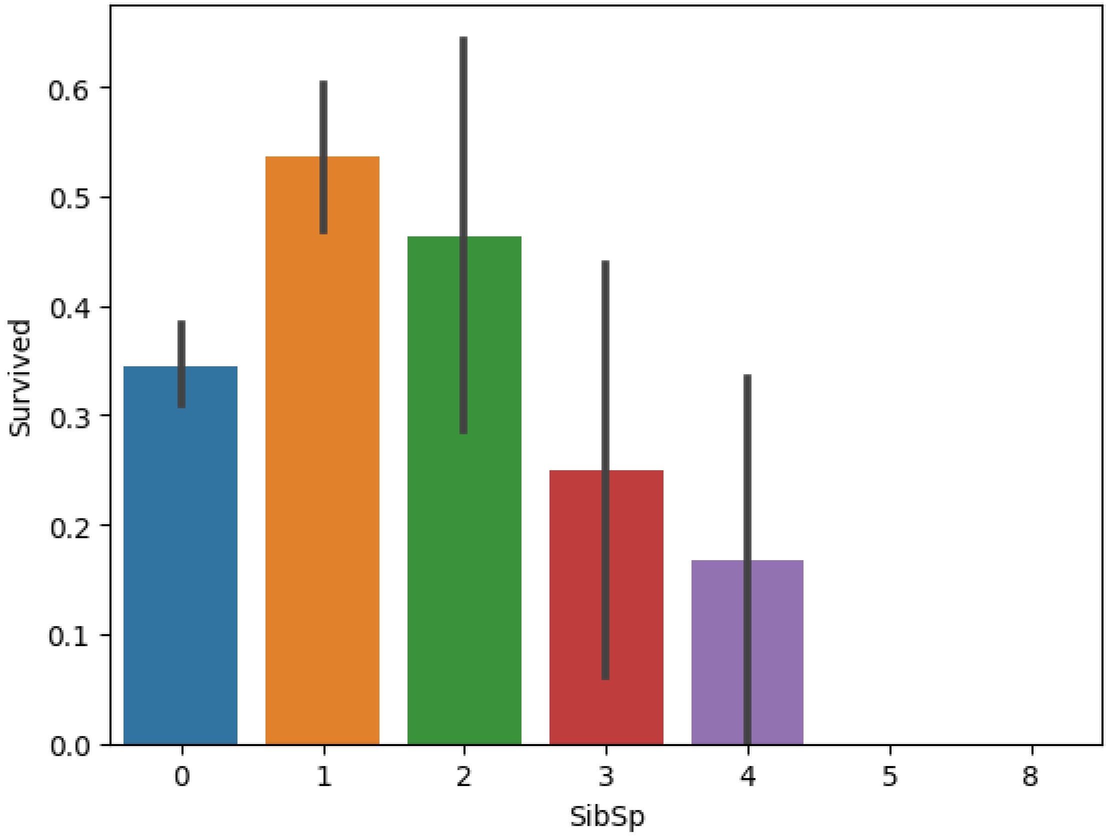
In [9]:
```
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

#I won't be printing individual percent values for all of these.
print("Percentage of SibSp = 0 who survived:", train["Survived"][train["SibS

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

```
print("Percentage of SibSp = 2 who survived:", train["Survived"][train["SibS
```

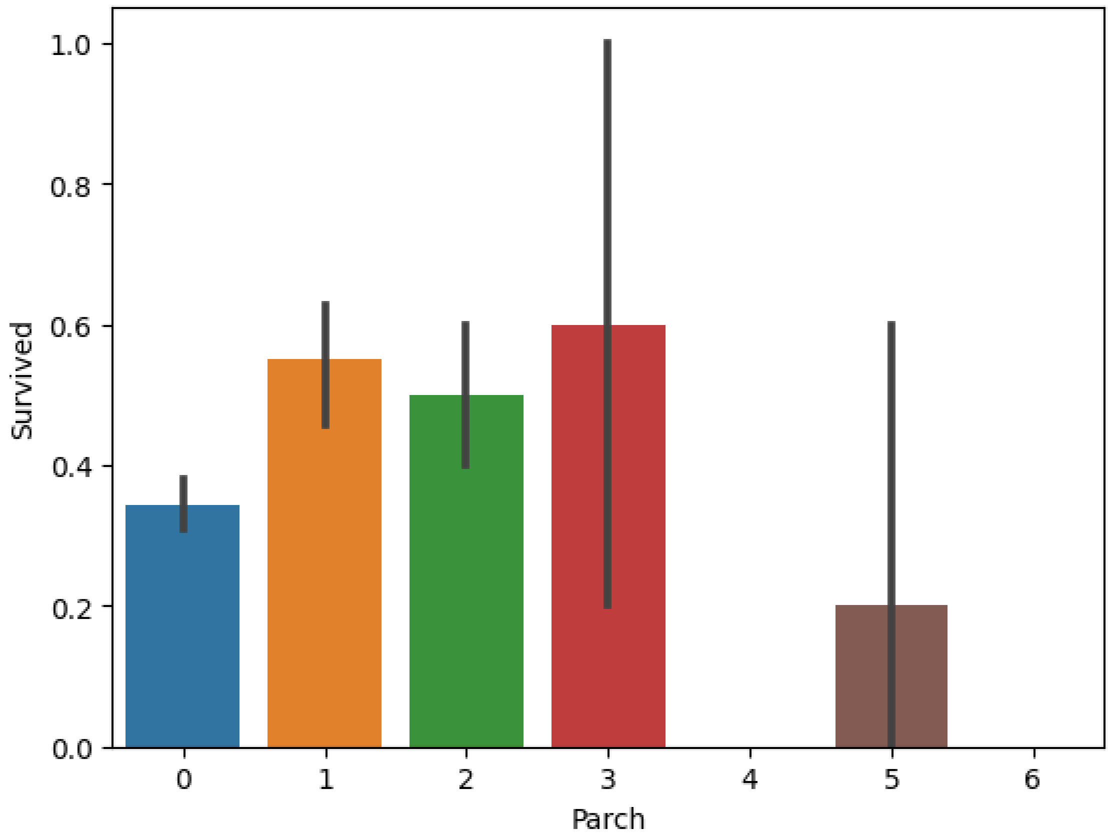
```
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```



In general, it's clear that people with more siblings or spouses aboard were less likely to survive. However, contrary to expectations, people with no siblings or spouses were less to likely to survive than those with one or two. (34.5% vs 53.4% vs. 46.4%)

Parch Feature

In [10]:
```
#draw a bar plot for Parch vs. survival
sns.barplot(x="Parch", y="Survived", data=train)
plt.show()
```

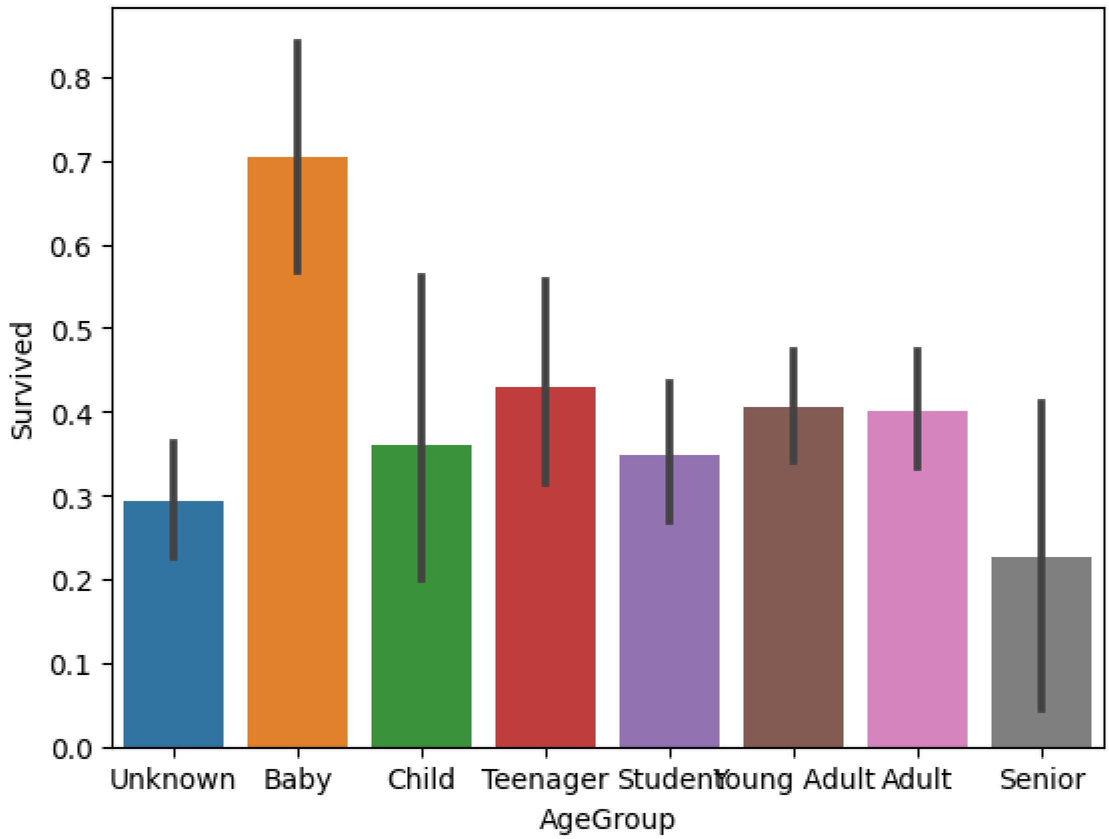


People with less than four parents or children aboard are more likely to survive than those with four or more. Again, people traveling alone are less likely to survive than those with 1-3 parents or children.

Age Feature

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',
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()
```



Babies are more likely to survive than any other age group.

Cabin Feature

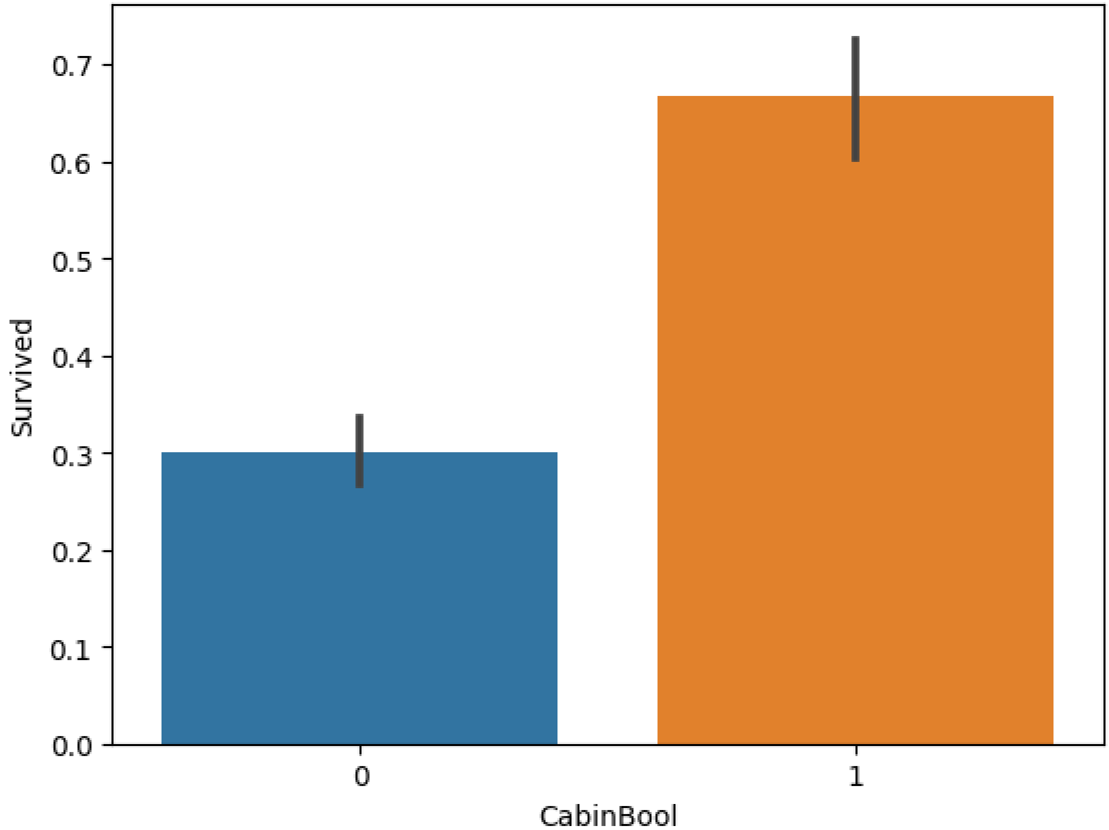
I think the idea here is that people with recorded cabin numbers are of higher socioeconomic class, and thus more likely to survive. Thanks for the tips, @salvus82 and Daniel Ellis!

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["

print("Percentage of CabinBool = 0 who survived:", train["Survived"][train["
#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
```



People with a recorded Cabin number are, in fact, more likely to survive. (66.6% vs 29.9%)

5) Cleaning Data

Time to clean our data to account for missing values and unnecessary information!

Looking at the Test Data

Let's see how our test data looks!

In [13]:
```
test.describe(include="all")
```

Out[13]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
count	418.000000	418.000000	418	418	418.000000	418.000000	418.000000	418	417
unique	NaN	NaN	418	2	NaN	NaN	NaN	363	
top	NaN	NaN	Kelly, Mr. James	male	NaN	NaN	NaN	PC 17608	
freq	NaN	NaN	1	266	NaN	NaN	NaN	5	
mean	1100.500000	2.265550	NaN	NaN	23.941388	0.447368	0.392344	NaN	35
std	120.810458	0.841838	NaN	NaN	17.741080	0.896760	0.981429	NaN	55
min	892.000000	1.000000	NaN	NaN	-0.500000	0.000000	0.000000	NaN	0
25%	996.250000	1.000000	NaN	NaN	9.000000	0.000000	0.000000	NaN	7
50%	1100.500000	3.000000	NaN	NaN	24.000000	0.000000	0.000000	NaN	14
75%	1204.750000	3.000000	NaN	NaN	35.750000	1.000000	0.000000	NaN	31
max	1309.000000	3.000000	NaN	NaN	76.000000	8.000000	9.000000	NaN	512

- We have a total of 418 passengers.
- 1 value from the Fare feature is missing.
- Around 20.5% of the Age feature is missing, we will need to fill that in.

Cabin Feature

In [14]:
```
#we'll start off by dropping the Cabin feature since not a lot more useful i
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

Ticket Feature

In [15]:
```
#we can also drop the Ticket feature since it's unlikely to yield any useful
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
```

Embarked Feature

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
```

It's clear that the majority of people embarked in Southampton (S). Let's go ahead and fill in the missing values with S.

In [17]:
```
#replacing the missing values in the Embarked feature with S
train = train.fillna({"Embarked": "S"})
```

Age Feature

Next we'll fill in the missing values in the Age feature. Since a higher percentage of values are missing, it would be illogical to fill all of them with the same value (as we did with Embarked). Instead, let's try to find a way to predict the missing ages.

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=Fal

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
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

In [19]:

```python
#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'],
    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]:

```python
#map each of the title groups to a numerical value
```

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Royal": 5, "Rar
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
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S

The code I used above is from here. Next, we'll try to predict the missing Age values from the most common age for their Title.

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",

#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]]
```

Now that we've filled in the missing values at least *somewhat* accurately (I will work on a better way for predicting missing age values), it's time to map each age group to a numerical value.

In [22]:

```
#map each Age value to a numerical value
age_mapping = {'Baby': 1, 'Child': 2, 'Teenager': 3, 'Student': 4, 'Young Ac
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)
```

Name Feature

We can drop the name feature now that we've extracted the titles.

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)
```

Sex Feature

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	CabinBo
0	1	0	3	0	1	0	7.2500	S	4.0	
1	2	1	1	1	1	0	71.2833	C	6.0	
2	3	1	3	1	0	0	7.9250	S	5.0	
3	4	1	1	1	1	0	53.1000	S	5.0	
4	5	0	3	0	0	0	8.0500	S	5.0	

Embarked Feature

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	CabinBo
0	1	0	3	0	1	0	7.2500	1	4.0	
1	2	1	1	1	1	0	71.2833	2	6.0	
2	3	1	3	1	0	0	7.9250	1	5.0	
3	4	1	1	1	1	0	53.1000	1	5.0	
4	5	0	3	0	0	0	8.0500	1	5.0	

Fare Feature

It's time separate the fare values into some logical groups as well as filling in the single missing value in the test dataset.

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"].mea

#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
0	1	0	3	0	1	0	1	4.0	0	1
1	2	1	1	1	1	0	2	6.0	1	3
2	3	1	3	1	0	0	1	5.0	0	2
3	4	1	1	1	1	0	1	5.0	1	3
4	5	0	3	0	0	0	1	5.0	0	1

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	0	1	1
1	893	3	1	1	0	1	6.0	0	3	1
2	894	2	0	0	0	3	7.0	0	1	2
3	895	3	0	0	0	1	5.0	0	1	2
4	896	3	1	1	1	1	4.0	0	3	2

Sources:

- Titanic Data Science Solutions
- Scikit-Learn ML from Start to Finish