Titanic Survival Prediction

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.

Contents:

1) Import Necessary Libraries

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

In [3]:
```
#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 [4]:
```
#import train and test CSV files
train = pd.read_csv(r"C:\Users\Dell\Documents\Data Science, Machine Learning\Datasets\train.csv")
test = pd.read_csv(r"C:\Users\Dell\Documents\Data Science, Machine Learning\Datasets\test.csv")

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

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Cumings, Mrs. John Bradley (Florence Briggs Th...	male	NaN	NaN	NaN	CA. 2343	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

3) Data Analysis

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

In [5]:
```
#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 [6]:

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

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	NaN	S
620	621	0	3	Yasbeck, Mr. Antoni	male	27.0	1	0	2659	14.4542	NaN	C
239	240	0	2	Hunt, Mr. George Henry	male	33.0	0	0	SCO/W 1585	12.2750	NaN	S
448	449	1	3	Baclini, Miss. Marie Catherine	female	5.0	2	1	2666	19.2583	NaN	C
66	67	1	2	Nye, Mrs. (Elizabeth Ramell)	female	29.0	0	0	C.A. 29395	10.5000	F33	S

1. Numerical Features: Age (Continuous), Fare (Continuous), SibSp (Discrete), Parch (Discrete)
2. Categorical Features: Survived, Sex, Embarked, Pclass
3. Alphanumeric Features: Ticket, Cabin

What are the data types for each feature?

1. Survived: int
2. Pclass: int
3. Name: string
4. Sex: string
5. Age: float
6. SibSp: int
7. Parch: int
8. Ticket: string
9. Fare: float
10. Cabin: string
11. 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 [7]:

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

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Cumings, Mrs. John Bradley (Florence Briggs Th...	male	NaN	NaN	NaN	CA. 2343	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

Some Observations:

1. There are a total of 891 passengers in our training set.
2. 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.
3. 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.
4. The Embarked feature is missing 0.22% of its values, which should be relatively harmless.

In [8]:

```python
#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:

1. Sex: Females are more likely to survive.
2. SibSp/Parch: People traveling alone are more likely to survive.
3. Age: Young children are more likely to survive.
4. 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!

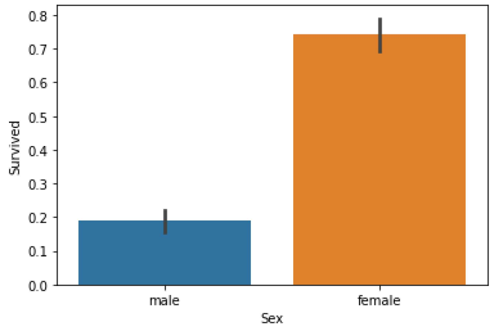
Sex Feature

In [9]:
```
#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'].value_counts(normalize = True)[1]*100)
```

```
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

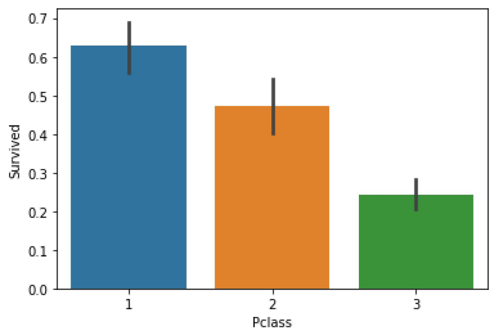
In [10]:
```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#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
```



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

SibSp Feature

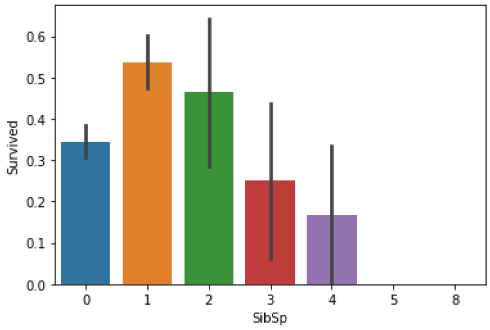
In [11]:
```python
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

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

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

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

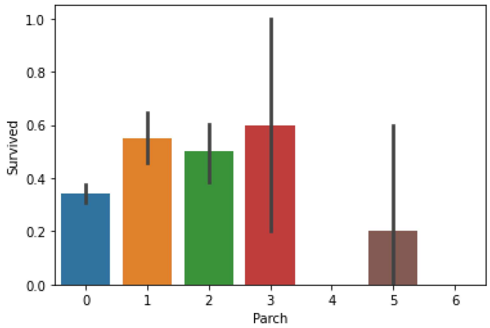
```
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```



In 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 [12]:
```python
#draw a bar plot for Parch vs. survival
sns.barplot(x="Parch", y="Survived", data=train)
plt.show()
```

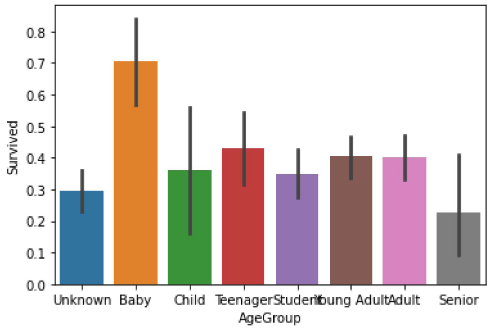



```
People with less than four parents or children aboard are more likely to survive than those with four or more. Again, people
traveling alone are less likely to survive than those with 1-3 parents or children.
```

Age Feature

In [13]:
```
#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', 'Senior']
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

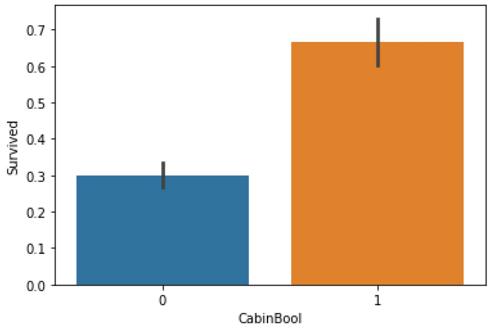
I think the idea here is that people with recorded cabin numbers are of higher socioeconomic class, and thus more likely to survive. Thanks for the tips, @salvus82 and Daniel Ellis!

In [14]:
```
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]*10

print("Percentage of CabinBool = 0 who survived:", train["Survived"][train["CabinBool"] == 0].value_counts(normalize = True)[1]*10
#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!

In [15]:
```
test.describe(include="all")
```

Out[15]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	AgeGroup	CabinBool
count	418.000000	418.000000	418	418	418.000000	418.000000	418.000000	418	417.000000	91	418	418	418.000000
unique	NaN	NaN	418	2	NaN	NaN	NaN	363	NaN	76	3	8	NaN
top	NaN	NaN	Thomas, Mr. Tannous	male	NaN	NaN	NaN	PC 17608	NaN	B57 B59 B63 B66	S	Young Adult	NaN
freq	NaN	NaN	1	266	NaN	NaN	NaN	5	NaN	3	270	96	NaN
mean	1100.500000	2.265550	NaN	NaN	23.941388	0.447368	0.392344	NaN	35.627188	NaN	NaN	NaN	0.217703
std	120.810458	0.841838	NaN	NaN	17.741080	0.896760	0.981429	NaN	55.907576	NaN	NaN	NaN	0.413179
min	892.000000	1.000000	NaN	NaN	-0.500000	0.000000	0.000000	NaN	0.000000	NaN	NaN	NaN	0.000000
25%	996.250000	1.000000	NaN	NaN	9.000000	0.000000	0.000000	NaN	7.895800	NaN	NaN	NaN	0.000000
50%	1100.500000	3.000000	NaN	NaN	24.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN	NaN	0.000000
75%	1204.750000	3.000000	NaN	NaN	35.750000	1.000000	0.000000	NaN	31.500000	NaN	NaN	NaN	0.000000
max	1309.000000	3.000000	NaN	NaN	76.000000	8.000000	9.000000	NaN	512.329200	NaN	NaN	NaN	1.000000

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.

In [16]:
```
#we'll start off by dropping the Cabin feature since not a lot more useful information can be extracted from it.
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

In [17]:
```
#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 [18]:
```
#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 [19]:
```
#replacing the missing values in the Embarked feature with S
train = train.fillna({"Embarked": "S"})
```

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 [20]:

```
#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[20]:

Sex Title	female	male
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 [21]:

```
#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[21]:

	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 [22]:

```
#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[22]:

	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 May Peel)	female	35.0	1	0	53.1000	S	Young Adult	1	3
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	Young Adult	0	1

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 [23]:
```python
# 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]]
```

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 [24]:
```python
#map each Age value to a numerical value
age_mapping = {'Baby': 1, 'Child': 2, 'Teenager': 3, 'Student': 4, 'Young Adult': 5, 'Adult': 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 [25]:
```python
#drop the name feature since it contains no more useful information.
train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)
```

In [26]:
```python
#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[26]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
0	1	0	3	0	1	0	7.2500	S	4.0	0	1
1	2	1	1	1	1	0	71.2833	C	6.0	1	3
2	3	1	3	1	0	0	7.9250	S	5.0	0	2
3	4	1	1	1	1	0	53.1000	S	5.0	1	3
4	5	0	3	0	0	0	8.0500	S	5.0	0	1

In [27]:
```python
#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[27]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeGroup	CabinBool	Title
0	1	0	3	0	1	0	7.2500	1	4.0	0	1
1	2	1	1	1	1	0	71.2833	2	6.0	1	3
2	3	1	3	1	0	0	7.9250	1	5.0	0	2
3	4	1	1	1	1	0	53.1000	1	5.0	1	3
4	5	0	3	0	0	0	8.0500	1	5.0	0	1

In [28]:
```python
#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 [29]:
```python
#check train data
train.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Embarked	AgeGroup	CabinBool	Title	FareBand
0	1	0	3	0	1	0	1	4.0	0	1	1
1	2	1	1	1	1	0	2	6.0	1	3	4
2	3	1	3	1	0	0	1	5.0	0	2	2
3	4	1	1	1	1	0	1	5.0	1	3	4
4	5	0	3	0	0	0	1	5.0	0	1	2

In [30]:
```python
#check test data
test.head()
```

Out[30]:

	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

6) Choosing the Best Model

Splitting the Training Data

We will use part of our training data (22% in this case) to test the accuracy of our different models.

In [31]:
```python
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)
```

Testing Different Models

I will be testing the following models with my training data (got the list from here):

1. Gaussian Naive Bayes
2. Logistic Regression
3. Support Vector Machines
4. Perceptron
5. Decision Tree Classifier
6. Random Forest Classifier
7. KNN or k-Nearest Neighbors
8. Stochastic Gradient Descent
9. Gradient Boosting Classifier

For each model, we set the model, fit it with 80% of our training data, predict for 20% of the training data and check the accuracy.

In [32]:
```python
# 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 [33]:
```
# 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)
```

```
79.7
```

In [34]:
```
# Support Vector Machines
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_val)
acc_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_svc)
```

```
82.74
```

In [35]:
```
# Linear SVC
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(x_train, y_train)
y_pred = linear_svc.predict(x_val)
acc_linear_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_linear_svc)
```

```
78.68
```

In [38]:
```
# Perceptron
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_val)
acc_perceptron = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_perceptron)
```

```
78.68
```

In [39]:
```
#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)
```

```
81.73
```

In [40]:
```
# Random Forest
from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_val)
acc_randomforest = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_randomforest)
```

```
83.25
```

In [41]:
```
# KNN or k-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_pred = knn.predict(x_val)
acc_knn = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_knn)
```

```
77.66
```

In [42]:
```
# Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(x_train, y_train)
y_pred = sgd.predict(x_val)
acc_sgd = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_sgd)
```

```
79.7
```

In [43]:
```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
y_pred = gbk.predict(x_val)
acc_gbk = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gbk)
```

```
84.77
```

Let's compare the accuracies of each model!

In [44]:
```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron', 'Linear SVC',
              'Decision Tree', 'Stochastic Gradient Descent', 'Gradient Boosting Classifier'],
    'Score': [acc_svc, acc_knn, acc_logreg,
              acc_randomforest, acc_gaussian, acc_perceptron,acc_linear_svc, acc_decisiontree,
              acc_sgd, acc_gbk]})
models.sort_values(by='Score', ascending=False)
```

Out[44]:

	Model	Score
9	Gradient Boosting Classifier	84.77
3	Random Forest	83.25
0	Support Vector Machines	82.74
7	Decision Tree	81.73
2	Logistic Regression	79.70
8	Stochastic Gradient Descent	79.70
4	Naive Bayes	78.68
5	Perceptron	78.68
6	Linear SVC	78.68
1	KNN	77.66

I decided to use the Gradient Boosting Classifier model for the testing data.

In [ ]: