```
#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')

#import train and test CSV files
train = pd.read_csv("/Users/nivedha/Downloads/train.csv")
test = pd.read_csv("/Users/nivedha/Downloads/test.csv")

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

```
        PassengerId    Survived      Pclass                     Name
Sex  \
count    891.000000  891.000000  891.000000                      891
891
unique          NaN         NaN         NaN                      891
2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris
male
freq            NaN         NaN         NaN                        1
577
mean     446.000000    0.383838    2.308642                      NaN
NaN
std      257.353842    0.486592    0.836071                      NaN
NaN
min        1.000000    0.000000    1.000000                      NaN
NaN
25%      223.500000    0.000000    2.000000                      NaN
NaN
50%      446.000000    0.000000    3.000000                      NaN
NaN
75%      668.500000    1.000000    3.000000                      NaN
NaN
max      891.000000    1.000000    3.000000                      NaN
NaN

               Age       SibSp       Parch  Ticket        Fare
Cabin  \
count   714.000000  891.000000  891.000000     891  891.000000
204
unique         NaN         NaN         NaN     681         NaN
```

```
147
top            NaN         NaN         NaN  347082         NaN  B96
B98
freq           NaN         NaN         NaN       7         NaN
4
mean     29.699118    0.523008    0.381594     NaN   32.204208
NaN
std      14.526497    1.102743    0.806057     NaN   49.693429
NaN
min       0.420000    0.000000    0.000000     NaN    0.000000
NaN
25%      20.125000    0.000000    0.000000     NaN    7.910400
NaN
50%      28.000000    0.000000    0.000000     NaN   14.454200
NaN
75%      38.000000    1.000000    0.000000     NaN   31.000000
NaN
max      80.000000    8.000000    6.000000     NaN  512.329200
NaN

       Embarked
count       889
unique        3
top           S
freq        644
mean        NaN
std         NaN
min         NaN
25%         NaN
50%         NaN
75%         NaN
max         NaN
```

```
print(train.columns)
```

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

```
train.sample(5)
```

```
     PassengerId  Survived  Pclass                             Name
Sex  \
796          797         1       1        Leader, Dr. Alice (Farnham)
female
265          266         0       2                  Reeves, Mr. David
male
197          198         0       3   Olsen, Mr. Karl Siegwart Andreas
male
```

```
520          521         1       1              Perreault, Miss. Anne
female
149          150         0       2  Byles, Rev. Thomas Roussel Davids
male

      Age  SibSp  Parch      Ticket     Fare Cabin Embarked
796  49.0      0      0       17465  25.9292   D17        S
265  36.0      0      0  C.A. 17248  10.5000   NaN        S
197  42.0      0      1        4579   8.4042   NaN        S
520  30.0      0      0       12749  93.5000   B73        S
149  42.0      0      0      244310  13.0000   NaN        S
```

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

```
        PassengerId    Survived      Pclass                     Name
Sex  \
count    891.000000  891.000000  891.000000                      891
891
unique          NaN         NaN         NaN                      891
2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris
male
freq            NaN         NaN         NaN                        1
577
mean     446.000000    0.383838    2.308642                      NaN
NaN
std      257.353842    0.486592    0.836071                      NaN
NaN
min        1.000000    0.000000    1.000000                      NaN
NaN
25%      223.500000    0.000000    2.000000                      NaN
NaN
50%      446.000000    0.000000    3.000000                      NaN
NaN
75%      668.500000    1.000000    3.000000                      NaN
NaN
max      891.000000    1.000000    3.000000                      NaN
NaN

               Age       SibSp       Parch  Ticket        Fare
Cabin  \
count   714.000000  891.000000  891.000000     891  891.000000
204
unique         NaN         NaN         NaN     681         NaN
147
top            NaN         NaN         NaN  347082         NaN  B96
B98
freq           NaN         NaN         NaN       7         NaN
4
mean     29.699118    0.523008    0.381594     NaN   32.204208
```

```
NaN
std      14.526497    1.102743    0.806057     NaN   49.693429
NaN
min       0.420000    0.000000    0.000000     NaN    0.000000
NaN
25%      20.125000    0.000000    0.000000     NaN    7.910400
NaN
50%      28.000000    0.000000    0.000000     NaN   14.454200
NaN
75%      38.000000    1.000000    0.000000     NaN   31.000000
NaN
max      80.000000    8.000000    6.000000     NaN  512.329200
NaN

       Embarked
count       889
unique        3
top           S
freq        644
mean        NaN
std         NaN
min         NaN
25%         NaN
50%         NaN
75%         NaN
max         NaN
```

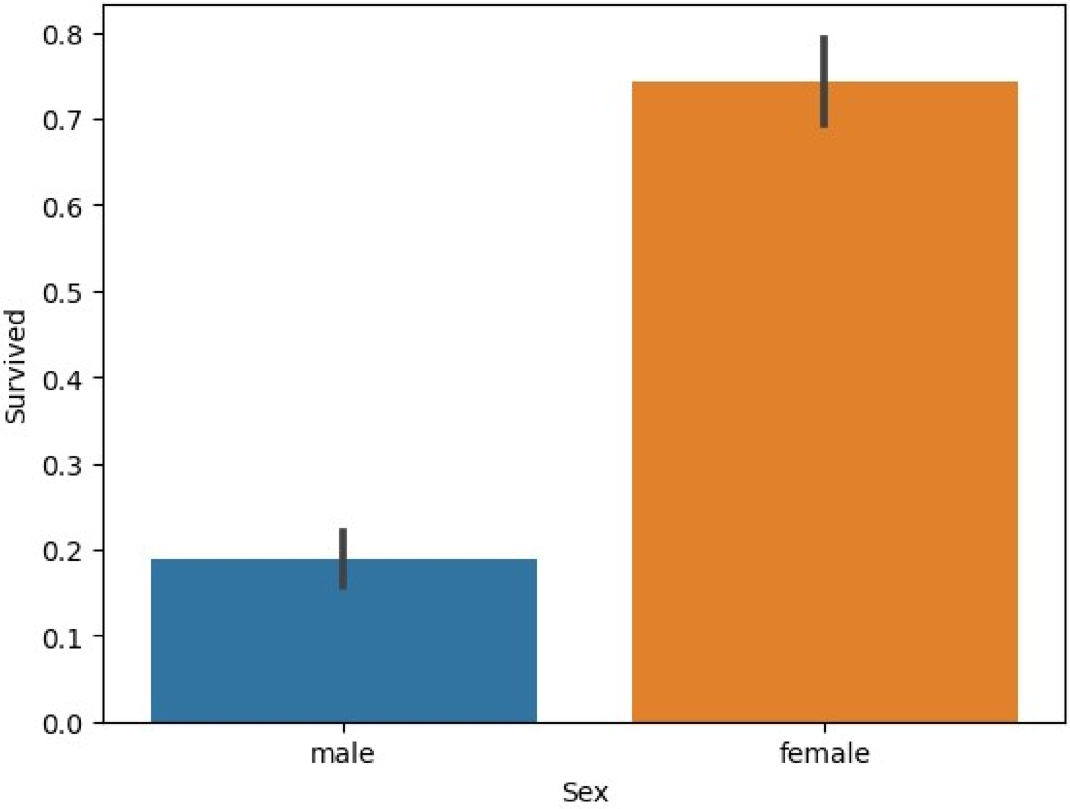
```
print(pd.isnull(train).sum())
```

```
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
```

```
#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)
```

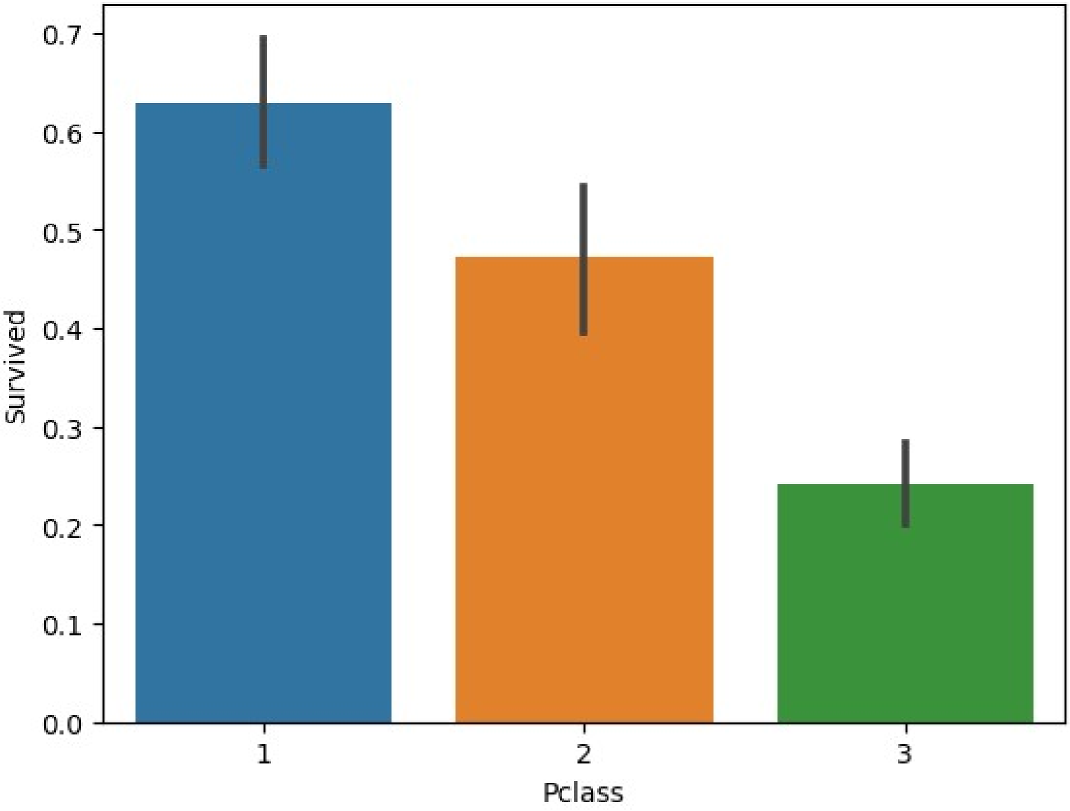
```
print("Percentage of males who survived:", train["Survived"]
[train["Sex"] == 'male'].value_counts(normalize = True)[1]*100)
```

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



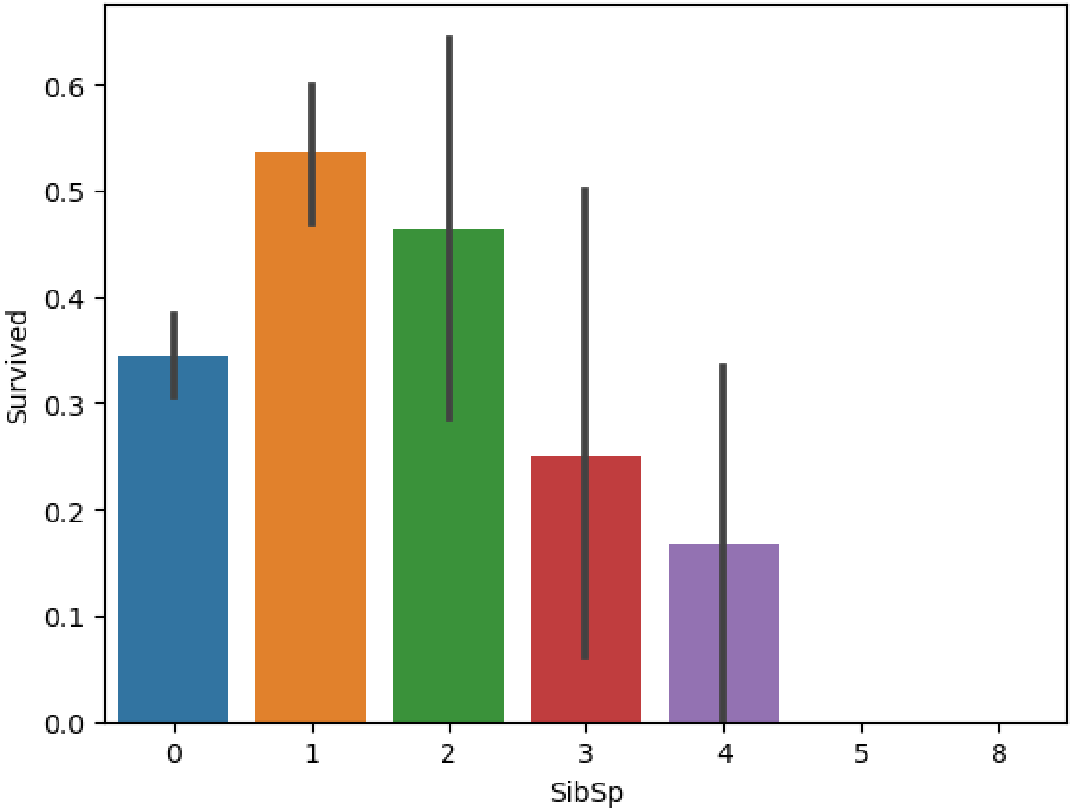
```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#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
```

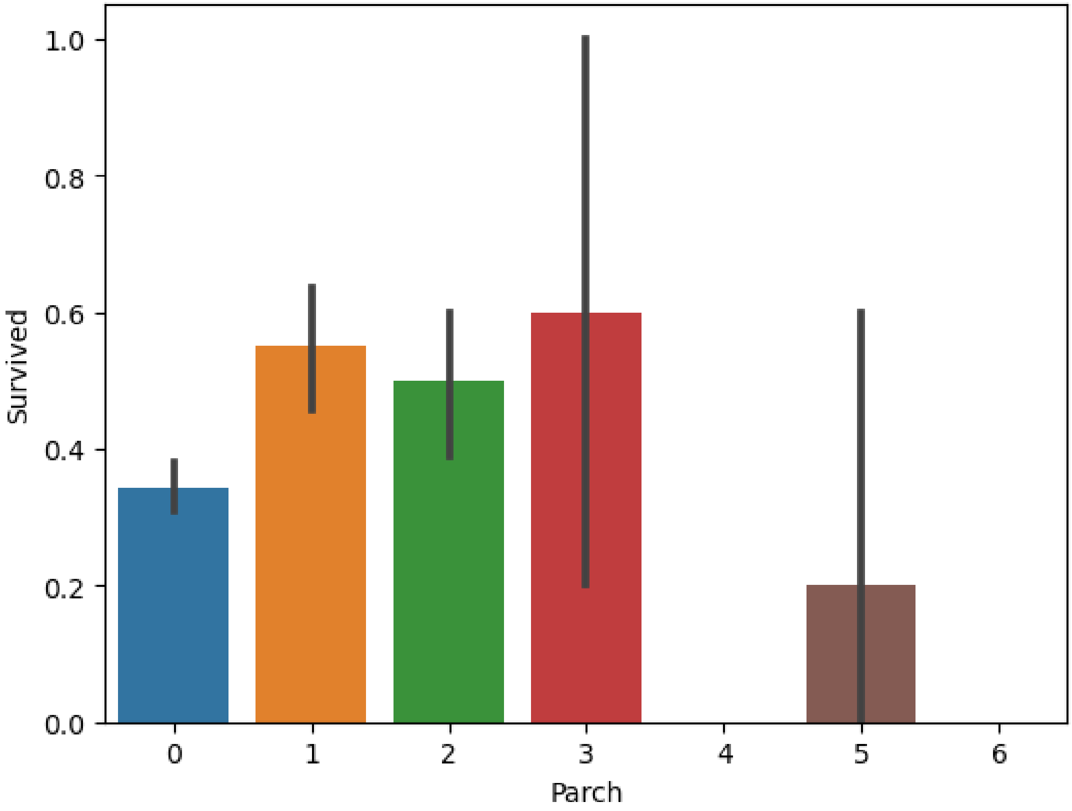


```
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)

#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)
```

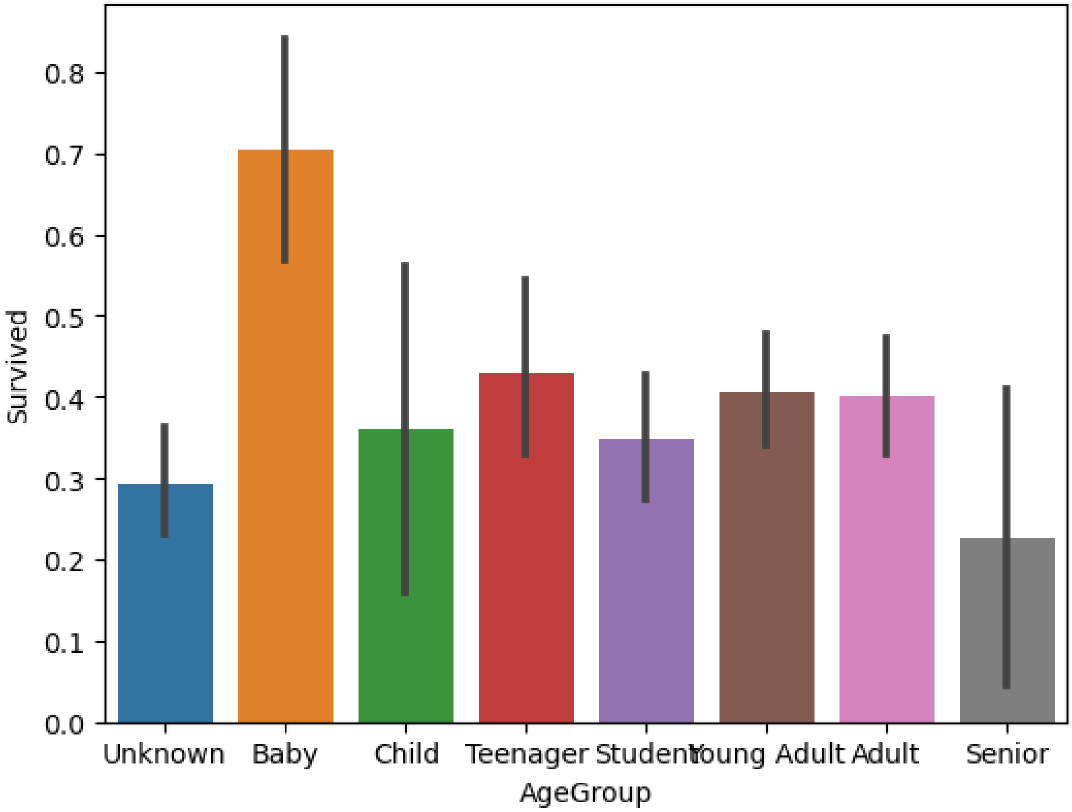
```
Percentage of SibSp = 0 who survived: 34.53947368421053
Percentage of SibSp = 1 who survived: 53.588516746411486
Percentage of SibSp = 2 who survived: 46.42857142857143
```



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

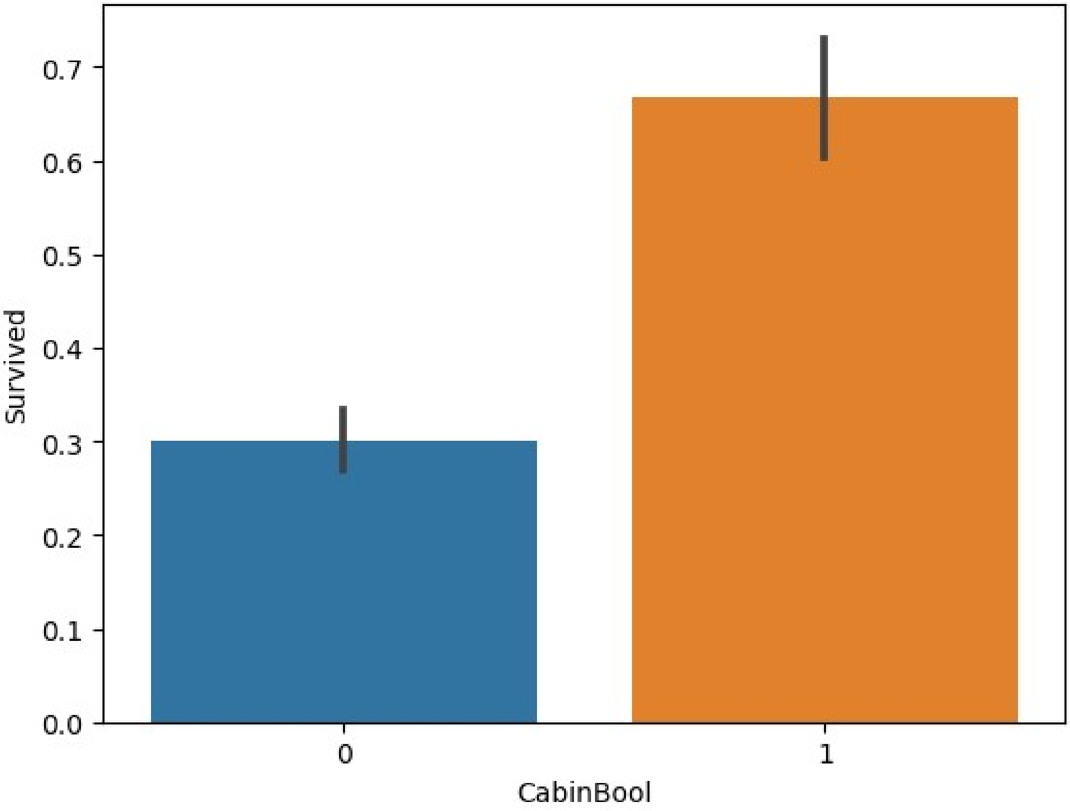


```
#sort the ages into logical categories
train["Age"] = train["Age"].fillna(-0.5)
test["Age"] = test["Age"].fillna(-0.5)
bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]
labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young
Adult', 'Adult', 'Senior']
train['AgeGroup'] = pd.cut(train["Age"],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



```
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
```



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

```
        PassengerId      Pclass              Name   Sex         Age  \
count    418.000000  418.000000               418   418  418.000000
unique          NaN         NaN               418     2         NaN
top             NaN         NaN  Kelly, Mr. James  male         NaN
freq            NaN         NaN                 1   266         NaN
mean    1100.500000    2.265550               NaN   NaN   23.941388
std      120.810458    0.841838               NaN   NaN   17.741080
min      892.000000    1.000000               NaN   NaN   -0.500000
25%      996.250000    1.000000               NaN   NaN    9.000000
50%     1100.500000    3.000000               NaN   NaN   24.000000
75%     1204.750000    3.000000               NaN   NaN   35.750000
max     1309.000000    3.000000               NaN   NaN   76.000000

             SibSp       Parch    Ticket        Fare            Cabin
\
count   418.000000  418.000000       418  417.000000               91

unique         NaN         NaN       363         NaN               76

top            NaN         NaN  PC 17608         NaN  B57 B59 B63 B66

freq           NaN         NaN         5         NaN                3
```

```
mean      0.447368    0.392344       NaN   35.627188             NaN

std       0.896760    0.981429       NaN   55.907576             NaN

min       0.000000    0.000000       NaN    0.000000             NaN

25%       0.000000    0.000000       NaN    7.895800             NaN

50%       0.000000    0.000000       NaN   14.454200             NaN

75%       1.000000    0.000000       NaN   31.500000             NaN

max       8.000000    9.000000       NaN  512.329200             NaN


       Embarked     AgeGroup   CabinBool
count       418          418  418.000000
unique        3            8         NaN
top           S  Young Adult         NaN
freq        270           96         NaN
mean        NaN          NaN    0.217703
std         NaN          NaN    0.413179
min         NaN          NaN    0.000000
25%         NaN          NaN    0.000000
50%         NaN          NaN    0.000000
75%         NaN          NaN    0.000000
max         NaN          NaN    1.000000
```

```
#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)

#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)

#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
```

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

#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'])
```

```
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
```

```
#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()
```

```
    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
```

```
#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()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch     Fare Embarked     AgeGroup  CabinBool  Title
0      0   7.2500        S      Student          0      1
1      0  71.2833        C        Adult          1      3
2      0   7.9250        S  Young Adult          0      2
3      0  53.1000        S  Young Adult          1      3
4      0   8.0500        S  Young Adult          0      1
```

```
# 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]]

#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)

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

#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()
```

```
   PassengerId  Survived  Pclass  Sex  SibSp  Parch     Fare Embarked
\
0            1         0       3    0      1      0   7.2500        S

1            2         1       1    1      1      0  71.2833        C

2            3         1       3    1      0      0   7.9250        S
```

```
3            4         1       1    1      1      0  53.1000        S

4            5         0       3    0      0      0   8.0500        S


   AgeGroup  CabinBool  Title
0       4.0          0      1
1       6.0          1      3
2       5.0          0      2
3       5.0          1      3
4       5.0          0      1
```

```
#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()
```

```
   PassengerId  Survived  Pclass  Sex  SibSp  Parch     Fare  Embarked
\
0            1         0       3    0      1      0   7.2500         1

1            2         1       1    1      1      0  71.2833         2

2            3         1       3    1      0      0   7.9250         1

3            4         1       1    1      1      0  53.1000         1

4            5         0       3    0      0      0   8.0500         1


   AgeGroup  CabinBool  Title
0       4.0          0      1
1       6.0          1      3
2       5.0          0      2
3       5.0          1      3
4       5.0          0      1
```

```
#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)

train.head()
```

```
   PassengerId  Survived  Pclass  Sex  SibSp  Parch  Embarked
AgeGroup  \
0            1         0       3    0      1      0         1
4.0
1            2         1       1    1      1      0         2
6.0
2            3         1       3    1      0      0         1
5.0
3            4         1       1    1      1      0         1
5.0
4            5         0       3    0      0      0         1
5.0

   CabinBool  Title FareBand
0          0      1        1
1          1      3        4
2          0      2        2
3          1      3        4
4          0      1        2
```

```
test.head()
```

```
   PassengerId  Pclass  Sex  SibSp  Parch  Embarked  AgeGroup
CabinBool  \
0          892       3    0      0      0         3       5.0
0
1          893       3    1      1      0         1       6.0
0
2          894       2    0      0      0         3       7.0
0
3          895       3    0      0      0         1       5.0
0
4          896       3    1      1      1         1       4.0
0

   Title FareBand
0      1        1
1      3        1
2      1        2
3      1        2
4      3        2
```

```
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)
```

```
# 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
```

```
# 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
```

```
# 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
```

```
#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)
```

```
81.22
```

```python
# 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
```

```python
# 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)
```

```
82.74
```

```python
from sklearn.linear_model import LinearRegression

linearregression = LinearRegression()
linearregression.fit(x_train, y_train)
y_pred = linearregression.predict(x_val)
from sklearn import metrics
import sklearn.metrics as sm
print("Mean absolute error =", round(sm.mean_absolute_error(y_val, 
y_pred), 2))
Mne =  round(sm.mean_squared_error(y_val, y_pred), 2)
print("Mean squared error =",Mne)
print("Median absolute error =", round(sm.median_absolute_error(y_val, 
y_pred), 2))
print("Explain variance score =", 
round(sm.explained_variance_score(y_val, y_pred), 2))
print("R2 score =", round(sm.r2_score(y_val, y_pred), 2))
```

```
Mean absolute error = 0.29
Mean squared error = 0.14
Median absolute error = 0.2
Explain variance score = 0.42
R2 score = 0.42
```

```python
#K-means clustering algorithm
from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters=2)
y_pred = kmeans_model.fit_predict(x_val)
acc_kmeans= round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_kmeans)
```

```
73.1
```

```
#PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
pca = PCA()
x_train = pca.fit_transform(x_train)
x_val = pca.transform(x_val)
```

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',

              'Random Forest', 'Naive Bayes','Linear Regression',
              'Decision Tree','K-means clustering' ],
    'Score': [acc_svc, acc_knn, acc_logreg,
              acc_randomforest, acc_gaussian, Mne, acc_decisiontree,
               acc_kmeans]})
models.sort_values(by='Score', ascending=False)
```

```
                     Model  Score
3            Random Forest  83.25
0  Support Vector Machines  82.74
1                      KNN  82.74
6            Decision Tree  81.22
2      Logistic Regression  79.70
4              Naive Bayes  78.68
7       K-means clustering  73.10
5        Linear Regression   0.14
```

```
#set ids as PassengerId and predict survival
ids = test['PassengerId']
predictions = randomforest.predict(test.drop('PassengerId', axis=1))

#set the output as a dataframe and convert to csv file named
submission.csv
output = pd.DataFrame({ 'PassengerId' : ids, 'Survived':
predictions })
output.to_csv('submission.csv', index=False)
```