Titanic_Classification

March 17, 2024

#**BHARAT INTERN - DATA SCIENCE INTERNSHIP**

##BY: M POOJA

###**TASK 2: Titanic Classification**: Build a predictive model to determine the likelihood of survival for passengers on the Titanic using data science techniques in Python.

##**Importing necessary Libraries**:

Numpy – Perform array manipulation and mathematical operations.

Pandas – Data manipulation and analysis library.

Matplotlib – Plotting library for creating visualizations.

Seaborn – Statistical data visualization library based on Matplotlib.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

##**Reading the dataset**:

The **Titanic survival** dataset taken from **Kaggle** is a popular dataset used for machine learning and statistical analysis projects. It contains information about passengers aboard the RMS Titanic, including details such as their age, sex, ticket class, fare, cabin, and whether they survived the sinking of the ship or not.

This dataset is often used for predictive modeling tasks, where the goal is to predict whether a passenger would survive based on their attributes.

[3]:
```
train = pd.read_csv("/content/train.csv")
test = pd.read_csv("/content/test.csv")
```

##**Data Analysis**

[4]:
```
train.describe()
```

[4]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
```

```
min        1.000000    0.000000    1.000000    0.420000    0.000000
25%      223.500000    0.000000    2.000000   20.125000    0.000000
50%      446.000000    0.000000    3.000000   28.000000    0.000000
75%      668.500000    1.000000    3.000000   38.000000    1.000000
max      891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

[40]:

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

[40]:

```
        PassengerId    Survived      Pclass         Sex       SibSp  \
count    891.000000  891.000000  891.000000  891.000000  891.000000
unique          NaN         NaN         NaN         NaN         NaN
top             NaN         NaN         NaN         NaN         NaN
freq            NaN         NaN         NaN         NaN         NaN
mean     446.000000    0.383838    2.308642    0.352413    0.523008
std      257.353842    0.486592    0.836071    0.477990    1.102743
min        1.000000    0.000000    1.000000    0.000000    0.000000
25%      223.500000    0.000000    2.000000    0.000000    0.000000
50%      446.000000    0.000000    3.000000    0.000000    0.000000
75%      668.500000    1.000000    3.000000    1.000000    1.000000
max      891.000000    1.000000    3.000000    1.000000    8.000000

             Parch    Embarked    AgeGroup   CabinBool       Title  FareBand
count   891.000000  891.000000  891.000000  891.000000  891.000000     891.0
unique         NaN         NaN         NaN         NaN         NaN       4.0
top            NaN         NaN         NaN         NaN         NaN       2.0
freq           NaN         NaN         NaN         NaN         NaN     224.0
mean      0.381594    1.361392    4.636364    0.228956    1.751964       NaN
std       0.806057    0.635673    1.353390    0.420397    1.112838       NaN
min       0.000000    1.000000    1.000000    0.000000    1.000000       NaN
25%       0.000000    1.000000    4.000000    0.000000    1.000000       NaN
50%       0.000000    1.000000    5.000000    0.000000    1.000000       NaN
75%       0.000000    2.000000    6.000000    0.000000    2.000000       NaN
max       6.000000    3.000000    7.000000    1.000000    6.000000       NaN
```

###**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.

[5]:
```
print(train.columns)
```

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

[6]:
```
train.sample(5)
```

[6]:
```
     PassengerId  Survived  Pclass  \
824          825         0       3
279          280         1       3
146          147         1       3
71            72         0       3
595          596         0       3

                                              Name     Sex   Age  SibSp  Parch  \
824                        Panula, Master. Urho Abraham    male   2.0      4      1
279                    Abbott, Mrs. Stanton (Rosa Hunt)  female  35.0      1      1
146  Andersson, Mr. August Edvard ("Wennerstrom")    male  27.0      0      0
71                          Goodwin, Miss. Lillian Amy  female  16.0      5      2
595                         Van Impe, Mr. Jean Baptiste    male  36.0      1      1

        Ticket     Fare Cabin Embarked
824    3101295  39.6875   NaN        S
279  C.A. 2673  20.2500   NaN        S
146     350043   7.7958   NaN        S
71     CA 2144  46.9000   NaN        S
595     345773  24.1500   NaN        S
```

Numerical Features: Age (Continuous), Fare (Continuous), SibSp (Discrete), Parch (Discrete)

Categorical Features: Survived, Sex, Embarked, Pclass

Alphanumeric Features: Ticket, Cabin

[7]:
```
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
```

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.

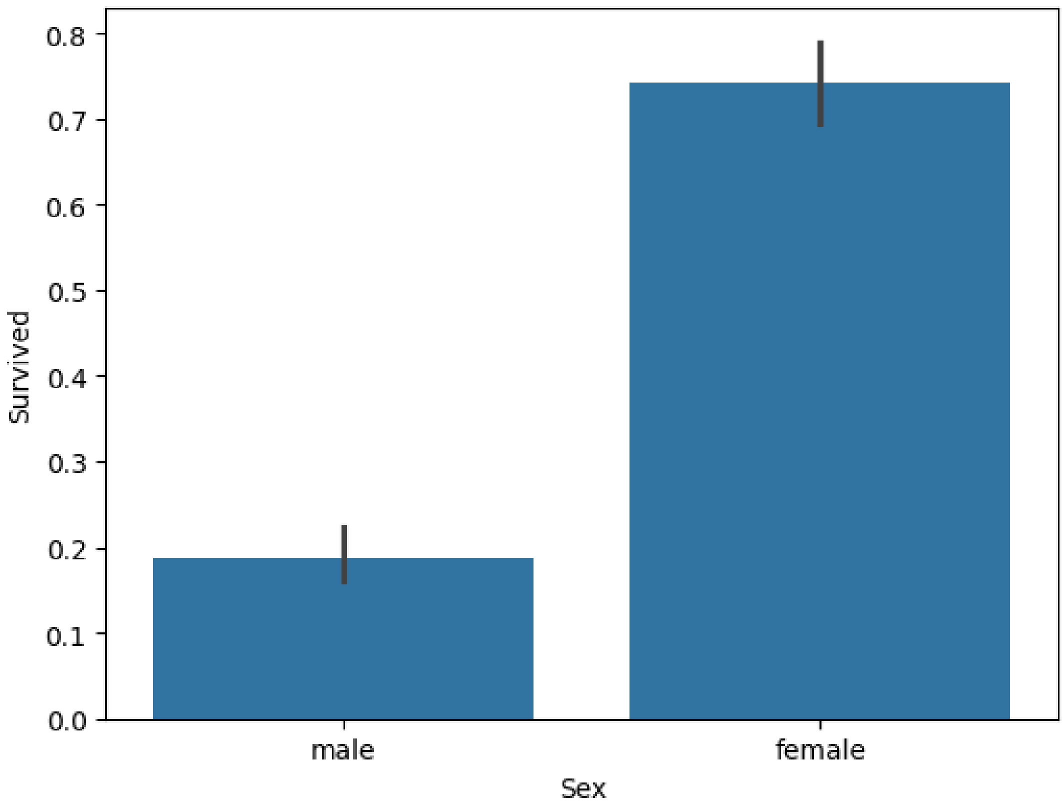
Pclass: People of higher socioeconomic class are more lik

#**Data Visualization**

[8]:

```
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
```



Inference:

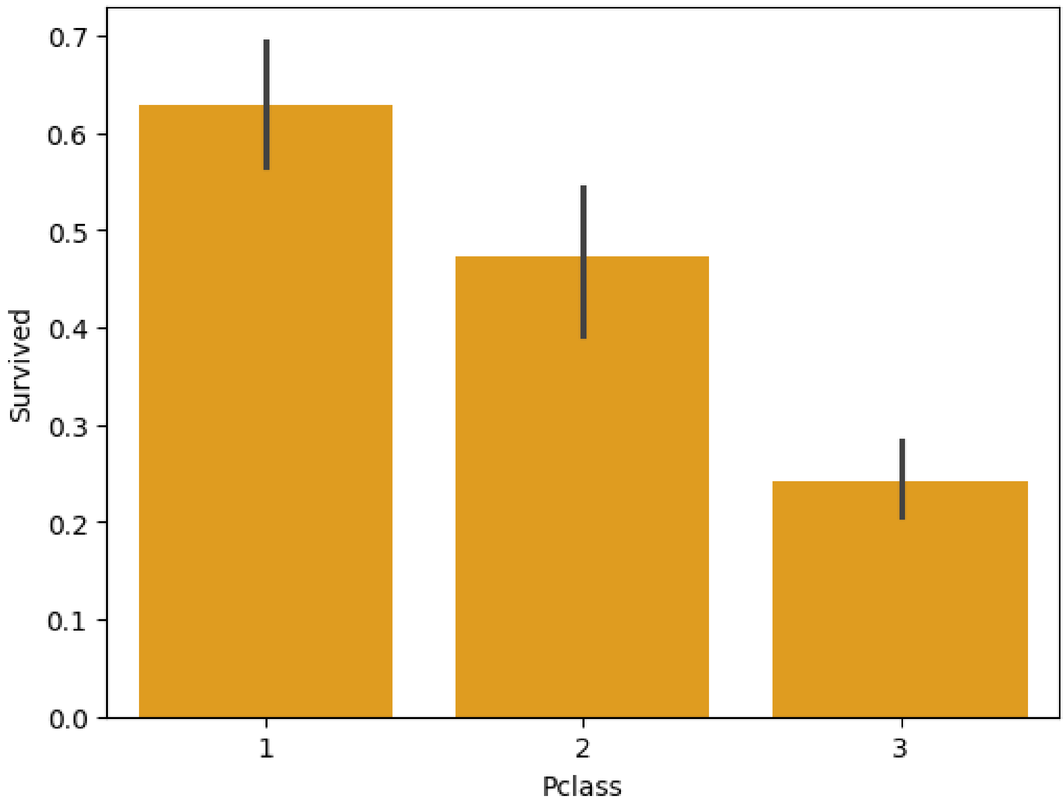
Percentage of females who survived: 74.20 Percentage of males who survived: 18.89

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

[11]:
```
sns.barplot(x="Pclass", y="Survived", data=train,color="Orange")

#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
```



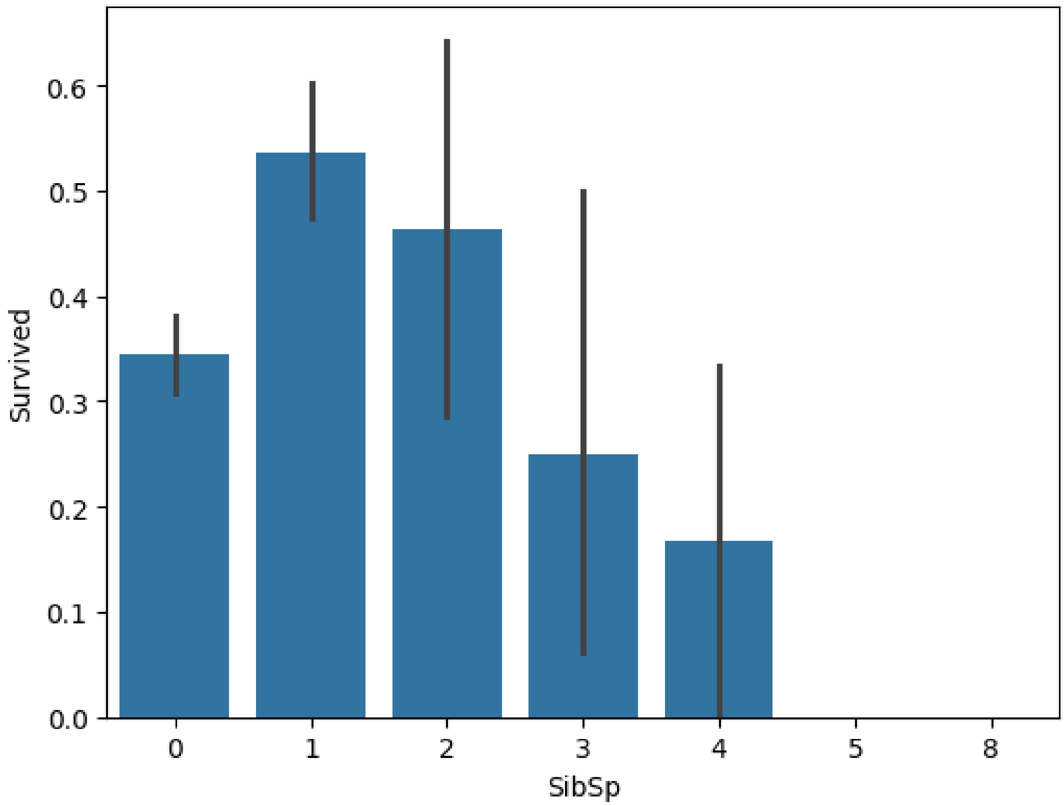
Inference:

People with higher socioeconomic class had a higher rate of survival. (62.9% vs. 47.3% vs. 24.2%)

[12]:
```
#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
```

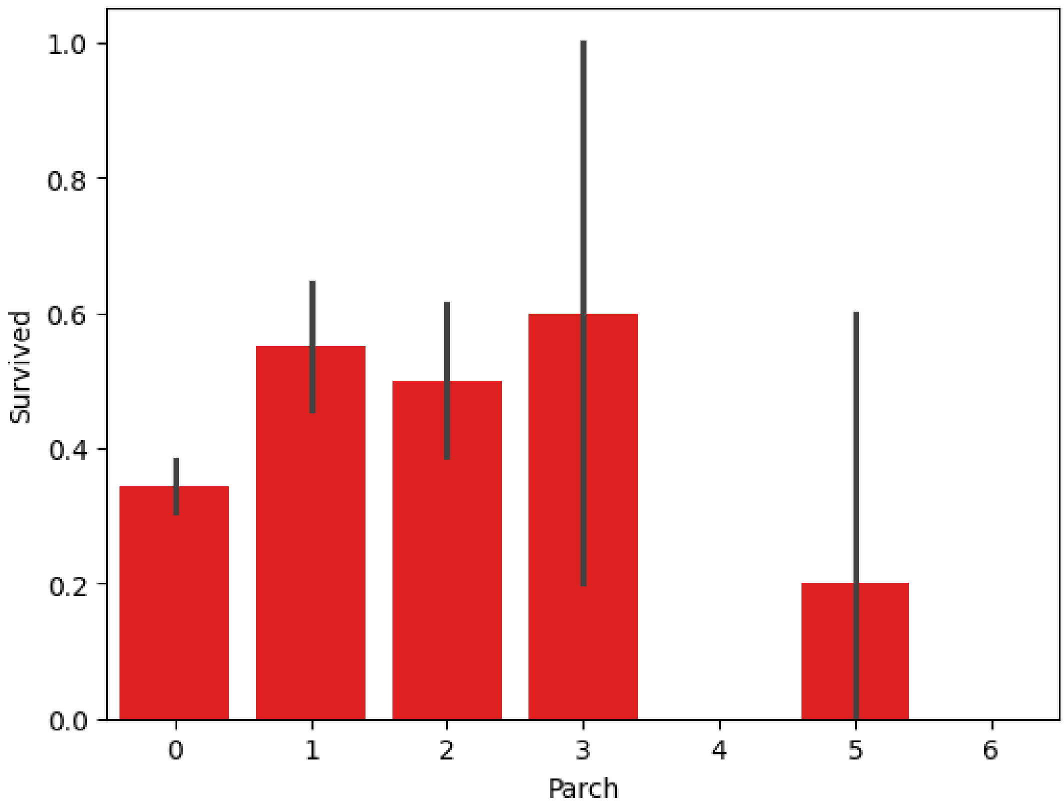
```
Percentage of SibSp = 2 who survived: 46.42857142857143
```



Inference:

People with more siblings or spouses aboard were less likely to survive.

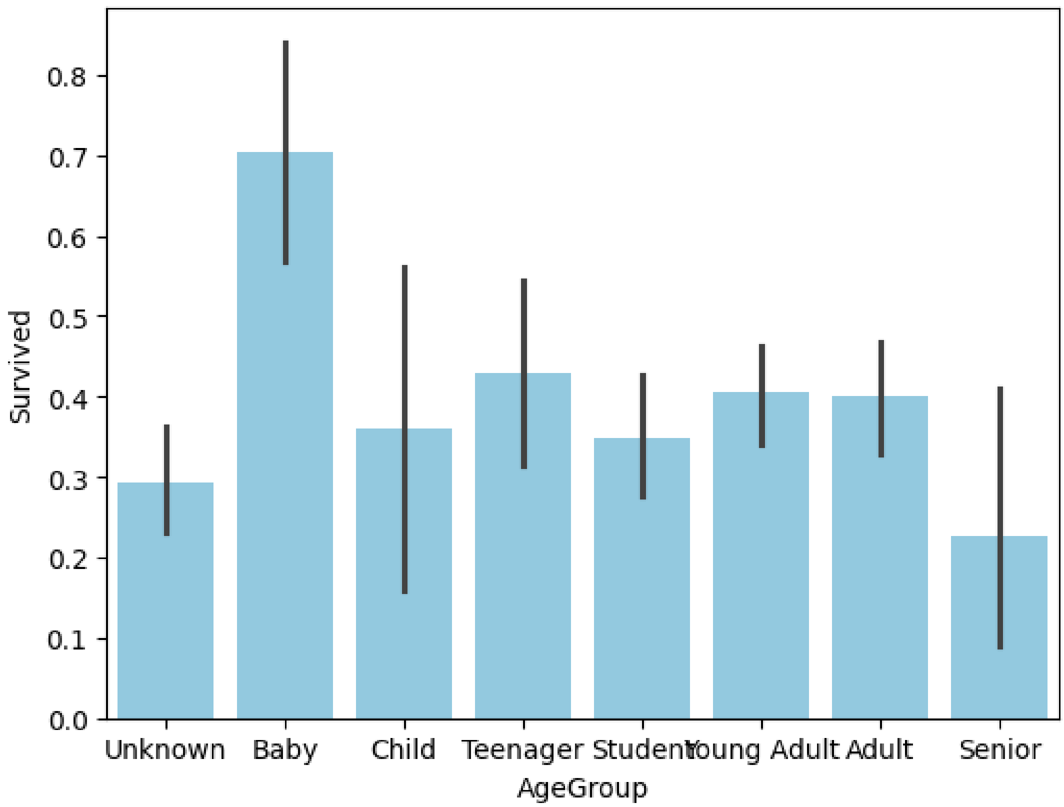
[14]:
```
#draw a bar plot for Parch vs. survival
sns.barplot(x="Parch", y="Survived", data=train,color='red')
plt.show()
```



Inference:

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.

[16]:
```
#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,color="skyblue")
plt.show()
```

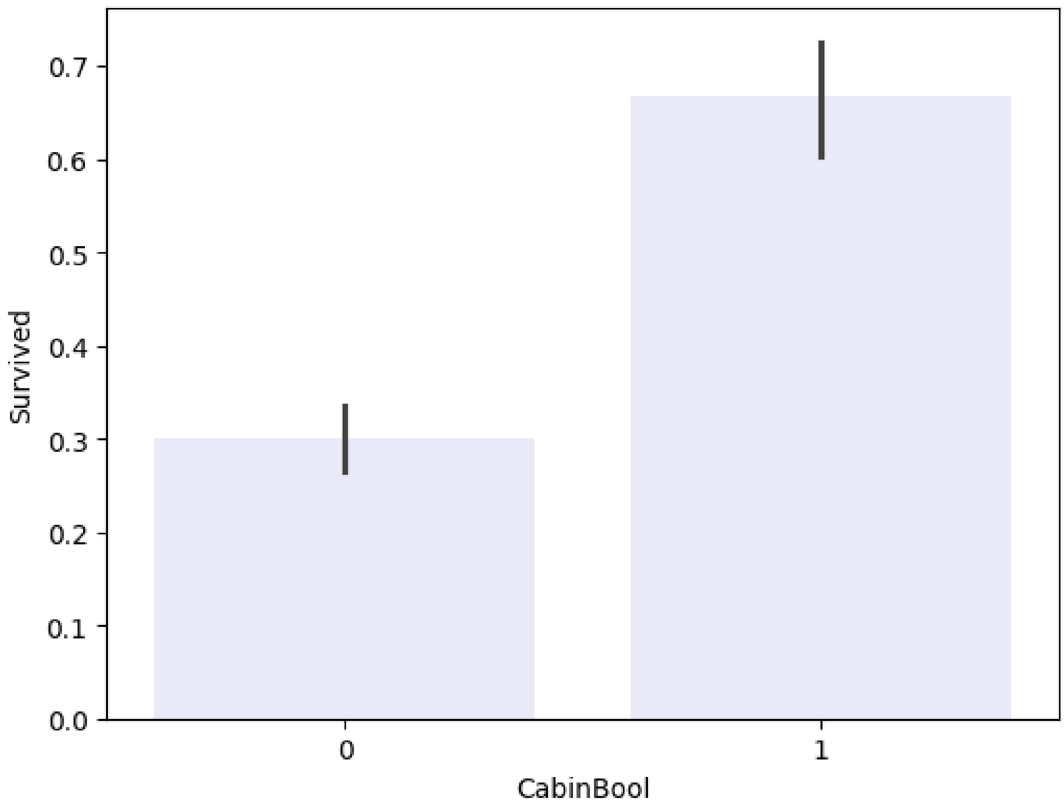


Inference:

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

[17]:
```
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,color="lavender")
plt.show()
```

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



Inference:

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

#**Cleaning Data**

[41]:
```
test.describe(include="all")
```

[41]:
```
        PassengerId      Pclass         Sex       SibSp       Parch  \
count    418.000000  418.000000  418.000000  418.000000  418.000000
unique          NaN         NaN         NaN         NaN         NaN
top             NaN         NaN         NaN         NaN         NaN
freq            NaN         NaN         NaN         NaN         NaN
mean    1100.500000    2.265550    0.363636    0.447368    0.392344
std      120.810458    0.841838    0.481622    0.896760    0.981429
min      892.000000    1.000000    0.000000    0.000000    0.000000
25%      996.250000    1.000000    0.000000    0.000000    0.000000
50%     1100.500000    3.000000    0.000000    0.000000    0.000000
75%     1204.750000    3.000000    1.000000    1.000000    0.000000
max     1309.000000    3.000000    1.000000    8.000000    9.000000

          Embarked    AgeGroup   CabinBool       Title   FareBand
```

```
count   418.000000  418.000000  418.000000  418.000000     418.0
unique         NaN         NaN         NaN         NaN       4.0
top            NaN         NaN         NaN         NaN       1.0
freq           NaN         NaN         NaN         NaN     114.0
mean      1.464115    4.696172    0.217703    1.755981       NaN
std       0.685516    1.286728    0.413179    1.058380       NaN
min       1.000000    1.000000    0.000000    1.000000       NaN
25%       1.000000    4.000000    0.000000    1.000000       NaN
50%       1.000000    5.000000    0.000000    1.000000       NaN
75%       2.000000    6.000000    0.000000    2.000000       NaN
max       3.000000    7.000000    1.000000    6.000000       NaN
```

Observations:

We have a total of 418 passengers.

1 value from the Fare feature is missing.

Around 20.5% of the Age feature is missing

[19]:
```
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

[20]:
```
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
```

[21]:
```
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
```

[22]:
```
train = train.fillna({"Embarked": "S"})
```

[23]:
```
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'])
```

[23]:

```
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
```

[24]:

```
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()
```

[24]:

```
    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
```

[25]:
```python
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()
```

[25]:
```
   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
```

[26]:
```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"}
```

[27]:
```python
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]]
```

We've filled in the missing values at least somewhat accurately, it's time to map each age group to a numerical value.

[28]:
```
#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)
```

[29]:
```
#drop the name feature since it contains no more useful information.
train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)
```

[30]:
```
#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()
```

[30]:
```
   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
```

[31]:
```
#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()
```

[31]:

```
   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
```

[32]:

```python
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)
```

```
<ipython-input-32-c4df96e03c7b>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test["Fare"][x] = round(train[train["Pclass"] == pclass]["Fare"].mean(), 4)
```

[33]:

```python
train.head()
```

[33]:

```
   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
```

[34]:
```
test.head()
```

[34]:
```
   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
```

#Choosing the 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.

[35]:
```
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:

Gaussian Naive Bayes

Logistic Regression

Support Vector Machines

Random Forest 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.

[36]:
```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```python
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

[37]:
```python
# 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

[38]:
```python
# 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

[39]:
```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)
```

85.28

##**Inferences:**

1) **Import Necessary Libraries:** Utilized essential libraries for data manipulation and visualization, facilitating subsequent analysis.
2) **Read In and Explore the Data:** Explored dataset structure and variables to formulate analysis strategies and gain initial insights.

3) **Data Analysis:** Examined descriptive statistics and identified missing values to understand dataset characteristics.

4) **Data Visualization:** Created visualizations to enhance comprehension of survival trends based on passenger attributes.

5) **Cleaning Data:** Addressed missing values and handled categorical variables to prepare data for modeling.

6) **Choosing Best Model**: Evaluated multiple machine learning algorithms, including ***Gaussian Naive Bayes, Logistic Regression, Support Vector Machine (SVM), and Random Forest***. By training and testing each model on the dataset, we assessed their performance in predicting Titanic survival.

Accuracy Rates:

Gaussian Naive Bayes: 78.68%

Logistic Regression: 79.7%

SVM: 82.74%

Random Forest: 85.28%

##**Conclusion**:

Made a significant progress in understanding factors influencing Titanic survival. Among the models tested, **Random Forest** exhibited the highest accuracy rate of 85.28%, indicating its effectiveness in predicting survival outcomes.

This highlights the importance of exploratory data analysis, feature engineering, and model selection in deriving meaningful insights from the Titanic survival dataset.