Titanic Analysis and Prediction

Table of Contents:

In [1]:
```
#Setup
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import plotly.express as px
```

1. Light Analysis

This will be a high level obseration of the data to understand the general shape and correlations. We can also start developing some ideas for potential feature engineering.

In [2]:
```
# Import training and test data
train = pd.read_csv("train.csv")
test= pd.read_csv("test.csv")
```

In [3]:
```
# Look at catagorical and numerical columns
print('Catagorical columns: {}'.format(train.describe(include = 'object').columns))
print('Numerical columns: {}'.format(train.describe(include = 'number').columns))

#General overview
train.describe(include = 'all')
```

```
Catagorical columns: Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='objec
t')
Numerical columns: Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch',
'Fare'], dtype='object')
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681
top	NaN	NaN	NaN	Jenkin, Mr. Stephen Curnow	male	NaN	NaN	NaN	347082
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN

In [4]:

```
#Create new df for numerical and categoric values
numeric = train[['Age', 'SibSp','Parch', 'Fare']]
categoric = train[['Survived','Sex', 'Pclass','Embarked']]  #'Cabin','Ticket',

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Observations and Prediction

General:

- There are 819 passengers in the training dataset

Numeric and categorical features:

- Numeric: Age (continuous) , SibSp (discrete), Parch (discrete), Fare (continuous)
- Categorical: Survived, Sex, Ticket, Cabin, Embarked, Pclass (discrete)

Data types:

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

Missing data:

- Age seems like an important factor and it is missing 177 values or 19.86% - we should address this
- Cabin is missing 687 values or 77.1%. This does not seem reasonably salvagable - we should drop this
- Embarked is only missing 2 values which should not affect the model too much - we can ignore this

Prediction

Now that we know what types of values we are working with let's make a few predictions:

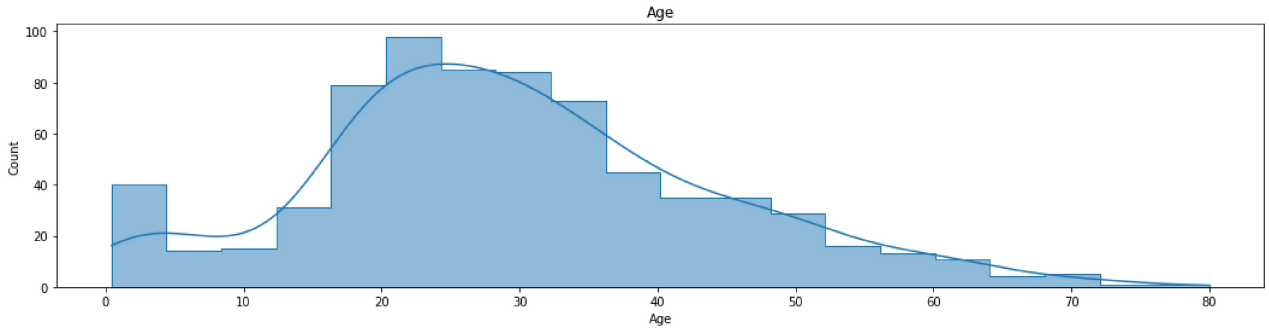
- Sex: Female survival rate will be much higher than males
- Pclass: People of higher class will have a higher survival rate
- Age: Younger individuals will have a higher survival rate

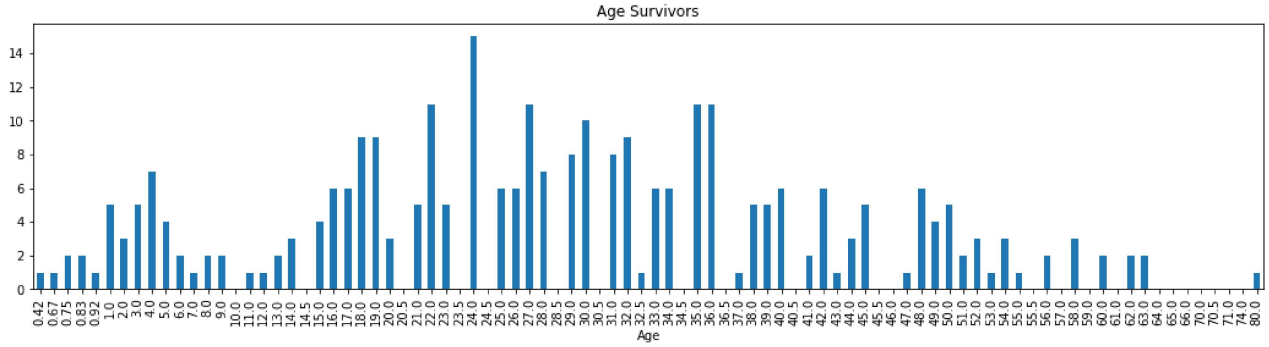
Let's create a few plots to see if we are close. We will plot the count for both numeric and categoric columns as well as the number of survivors.

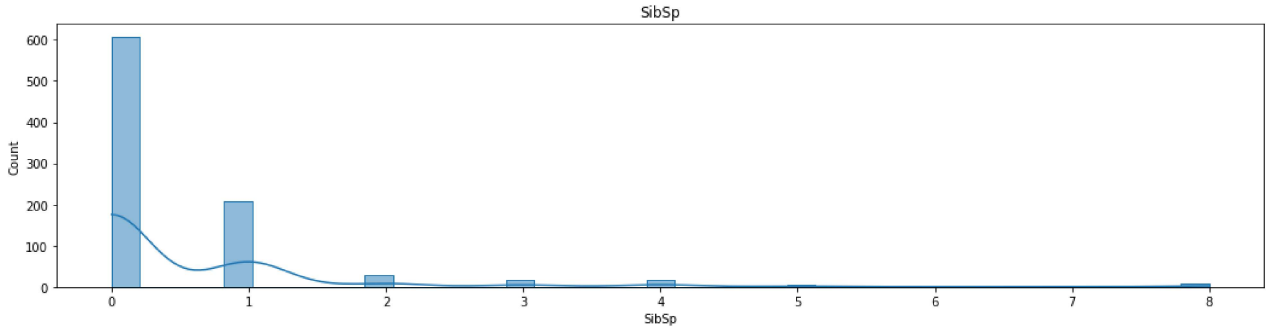
2. Visualizations

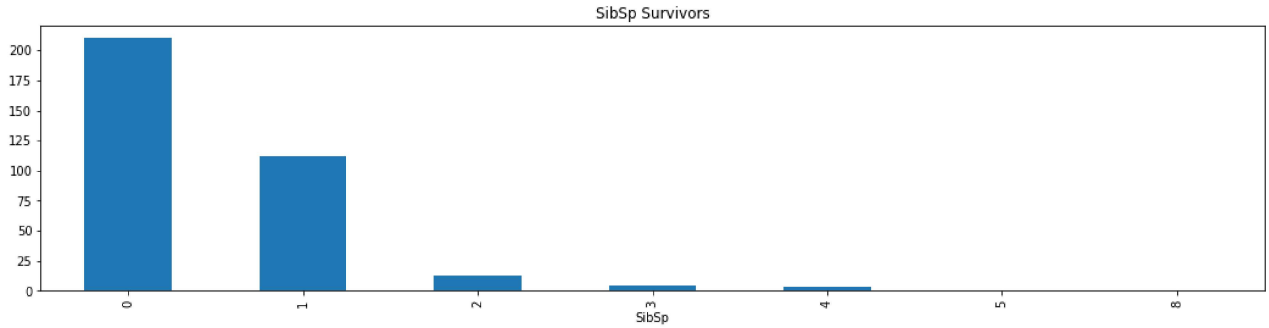
I will plot the distributions/ counts of each column, followed by the survivors.

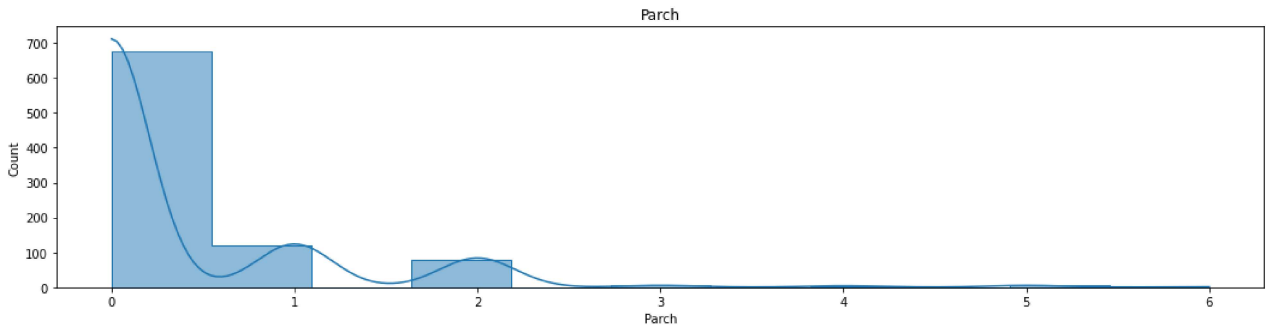
In [5]:
```
for i in numeric.columns:
    plt.figure(figsize=(18,4))

    #Plot distributions for numeric values
    sns.histplot(data = numeric, x = i,element="step",kde=True)
    plt.title(i)
    plt.show()

    #Plot survivors
    plt.figure(figsize=(18,4))
    z = train.groupby(i).Survived.sum()
    z.plot(kind='bar')

#     df['Survived']=z
#     df['Age']=numeric.Age.value_counts()
#     sns.barplot(x='Survived', y="Age", data=df,errwidth=0) #Why doesn't this work?

    plt.title(i+ " Survivors")
    plt.show()
```

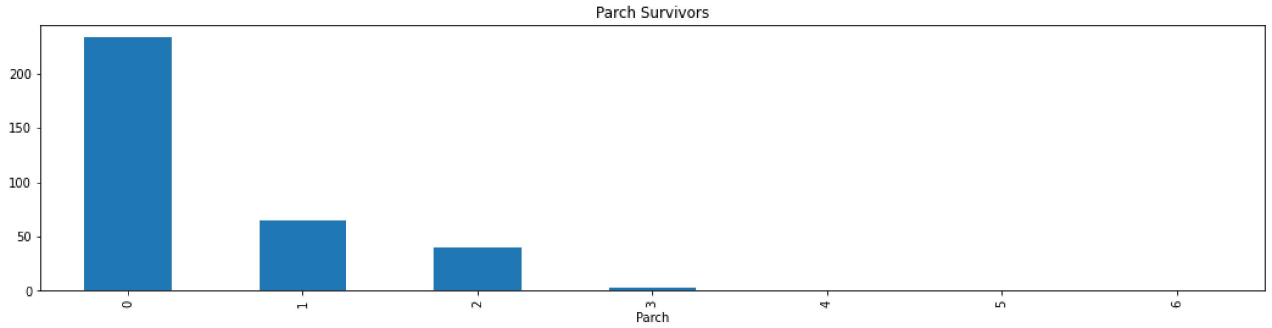

Age
Count
Age

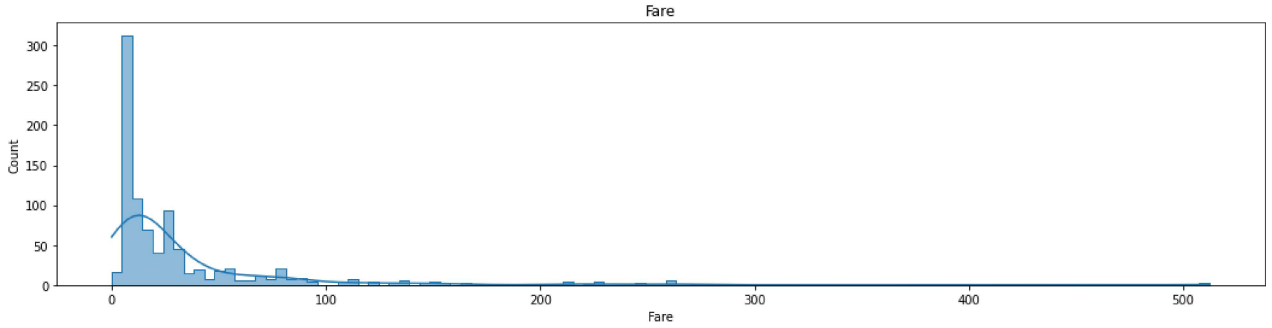

Age Survivors
Age

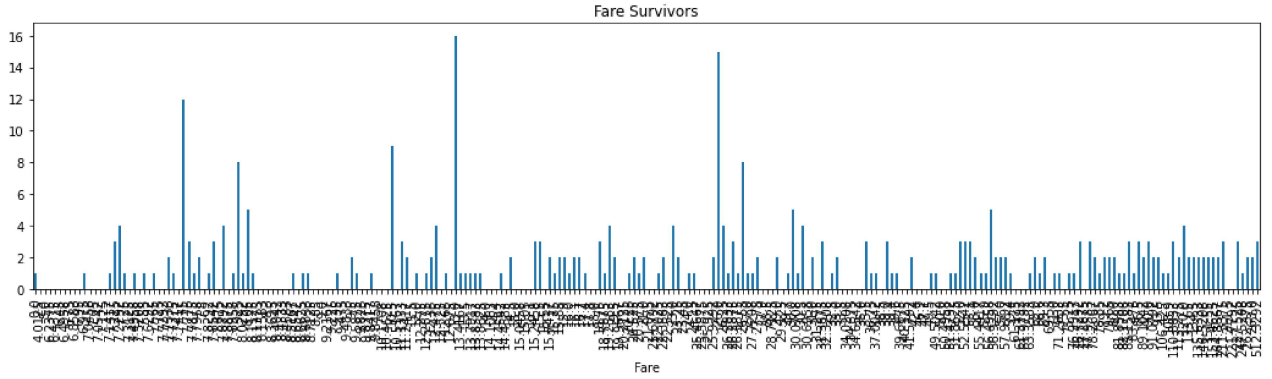

SibSp
Count
SibSp


SibSp Survivors
SibSp


Parch
Count
Parch

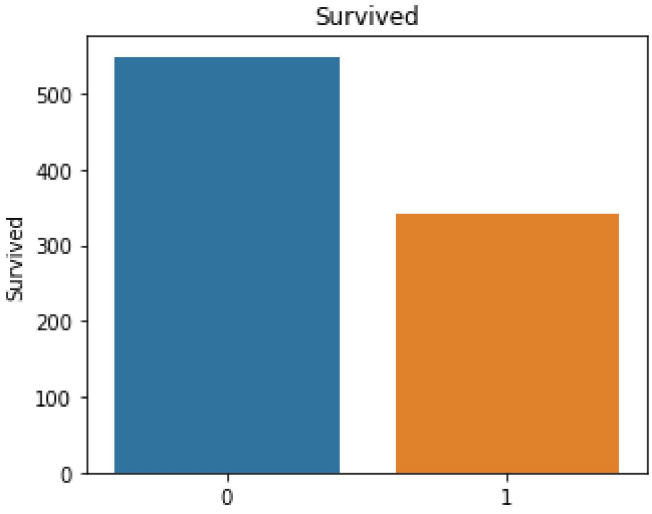


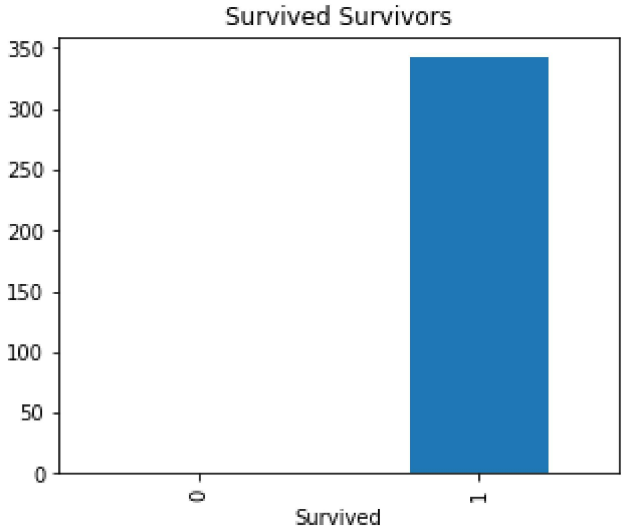


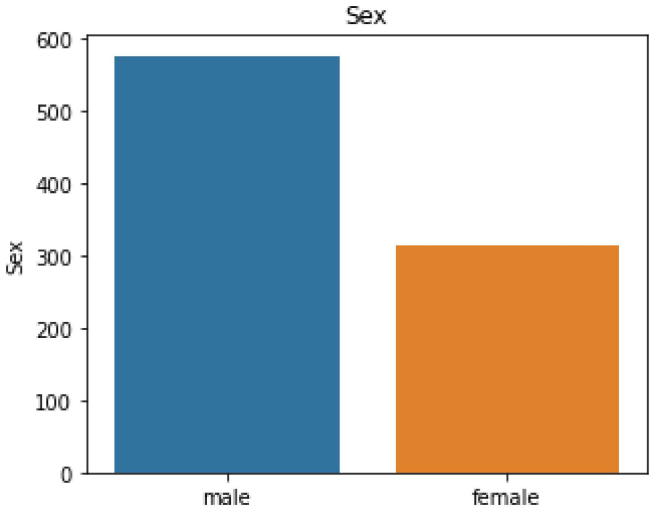


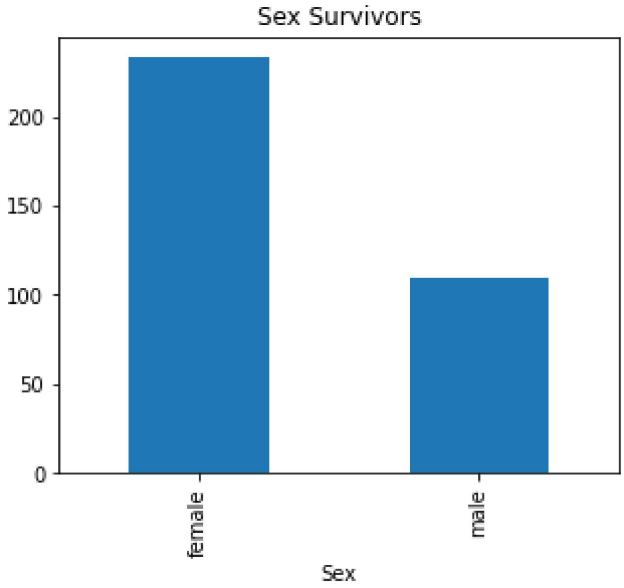
In [6]:

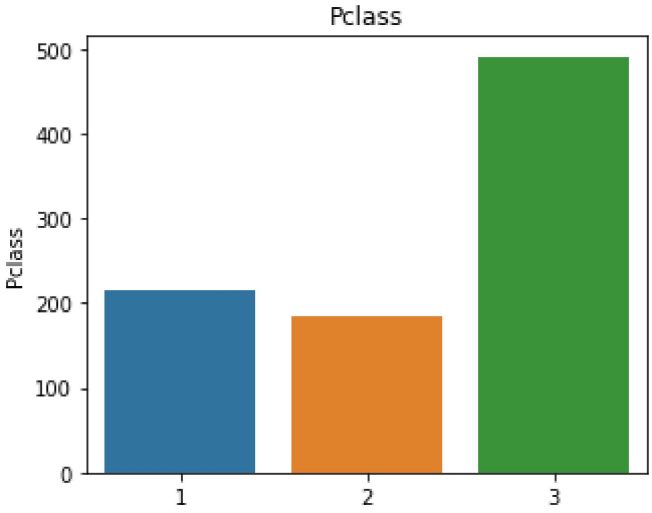
```
for i in categoric.columns:
    plt.figure(figsize=(5,4))
    #Plot distributions for numeric values
    sns.barplot(categoric[i].value_counts().index,categoric[i].value_counts())
    plt.title(i)
    plt.show()

    #Plot number of people that survived
    plt.figure(figsize=(5,4))
    z = train.groupby(i).Survived.sum()
    z.plot(kind='bar',ax=None)
    plt.title(i+ " Survivors")
    plt.show()
```

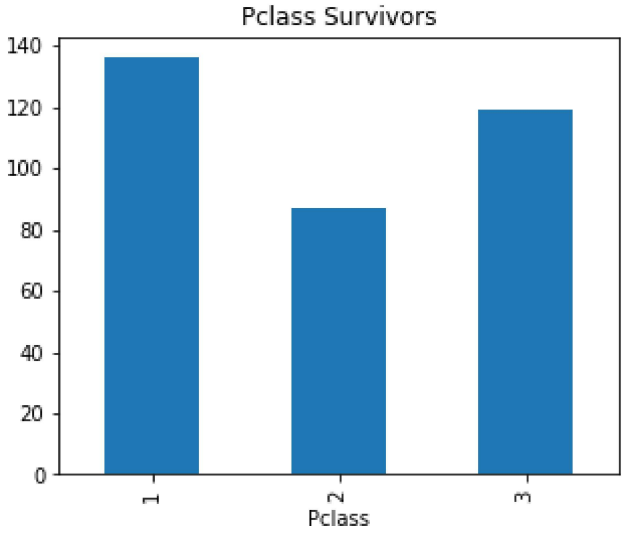

Survived
Survived
500
400
300
200
100
0
0
1

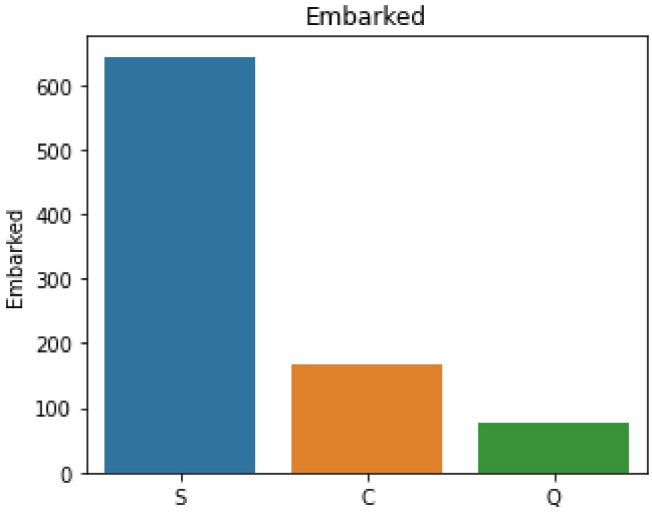

Survived Survivors
350
300
250
200
150
100
50
0
0
1
Survived

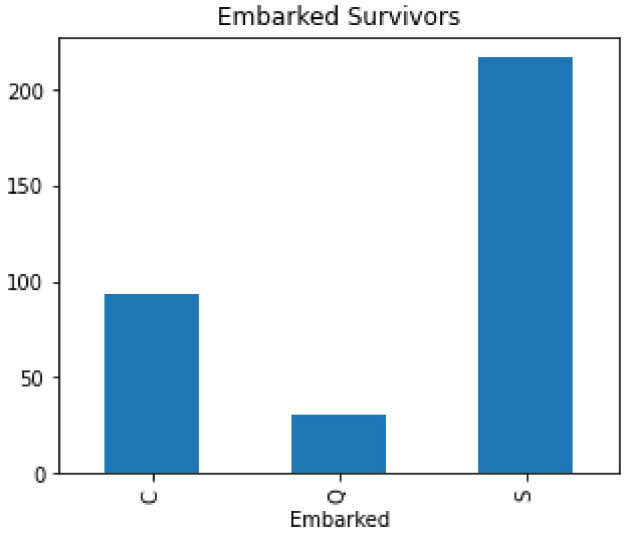

Sex
600
500
400
300
200
100
0
Sex
male
female


Sex Survivors
200
150
100
50
0
female
male
Sex


Pclass
500
400
300
200
100
0
Pclass
1
2
3

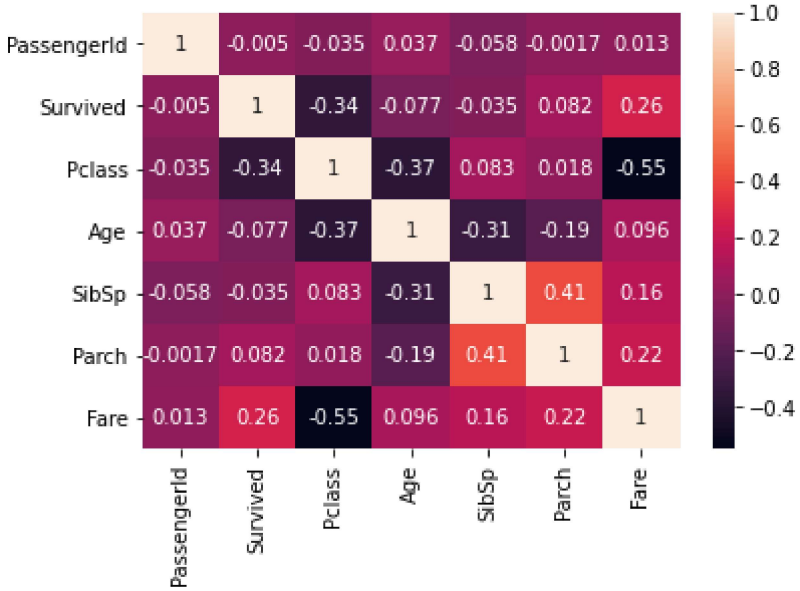

Pclass Survivors
140
120
100
80
60
40
20
0
1
2
3
Pclass





In [7]:
```
sns.heatmap(train.corr(),annot=True)
```

Out[7]: <AxesSubplot:>



General Observations: It looks like for the most part, the number of survivors is highly dependent

on the sheer volume of people in a specific category. Just looking at survival numbers of course is not fair as volume inflates survival so we will take a look at percentages later on.

Noteworthy Categories: Sex and class are the only two categories that have a strong correlation to survival rate. It also looks like babies have a fairly high survival rate. We will take a more in depth look in the next section.

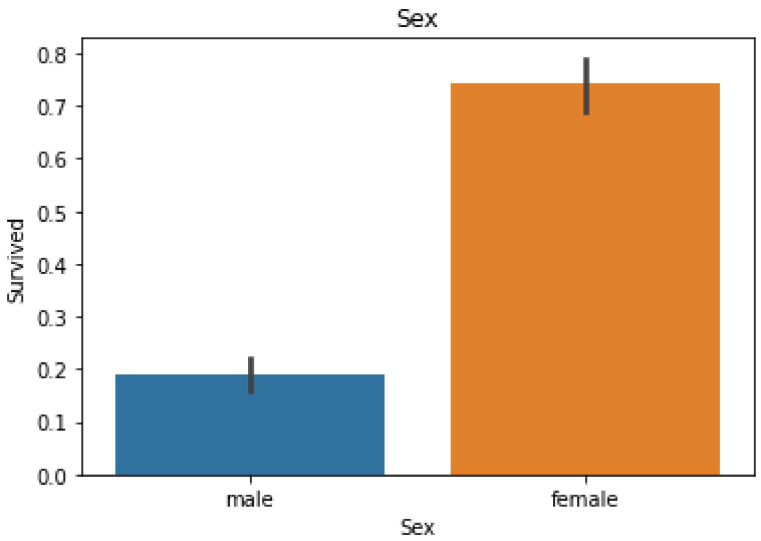
3. Feature Engineering

1. Sex: Is sex the largest determining factor?
2. Pclass: How strong is the correlation between class and survival?
3. Age: What ages are the most likely to survive
4. SibSp and Parch: Does travelling alone or with a group impact survival rate?

We will validate our original predictions.

1. Sex

In [8]:
```
#Plot Survival percentages of predictions

sns.barplot(x='Sex', y="Survived", data=train)
plt.title('Sex')
plt.show()

print("Male survival rate:", train["Survived"][train["Sex"] == 'male'].value_counts(nor
print("Female survival rate:", train["Survived"][train["Sex"] == 'female'].value_counts

#     Women = (train['Sex']=='female')                                    #Alternate s
#             w_survived = Women &(train['Survived'])
#             rate_women = sum(w_survived)/sum(Women)

#      women = train_data.loc[train_data.Sex == 'female']["Survived"]
#      rate_women = sum(women)/len(women)

          #normalize returns relative frequencies of unique values
```

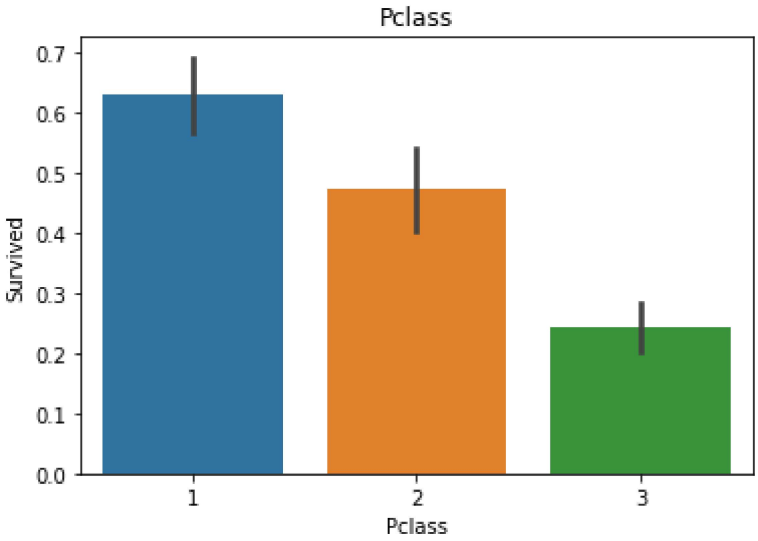



```
Male survival rate: 0.18890814558058924
Female survival rate: 0.7420382165605095
```

The survival rate discrepancies between genders are massive. It is highly likely this is a key determining factor for a passenger's survival.

2. Pclass

In [9]:
```
sns.barplot(x='Pclass', y="Survived", data=train)
plt.title('Pclass')
plt.show()

print("Class 1 survival rate:", train["Survived"][train["Pclass"] == 1].value_counts(no
print("Class 2 survival rate:", train["Survived"][train["Pclass"] == 2].value_counts(no
print("Class 3 survival rate:", train["Survived"][train["Pclass"] == 3].value_counts(no
```

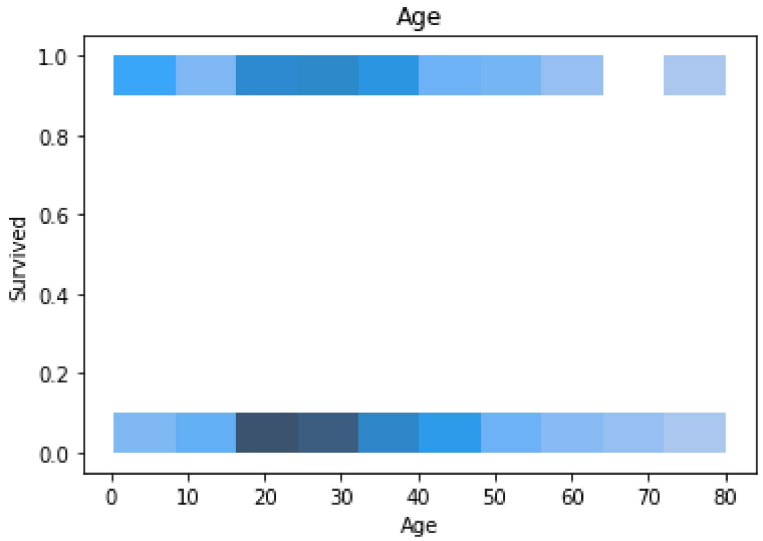



```
Class 1 survival rate: 0.6296296296296297
Class 2 survival rate: 0.47282608695652173
Class 3 survival rate: 0.24236252545824846
```

The correlation between class and survival is quite strong and completely linear. I believe that class and sex are the two main factors that determine survival rate.

3. Age

In [10]:
```
sns.histplot(x='Age', y="Survived", data=train,bins=10)
plt.title('Age')
plt.show()
```

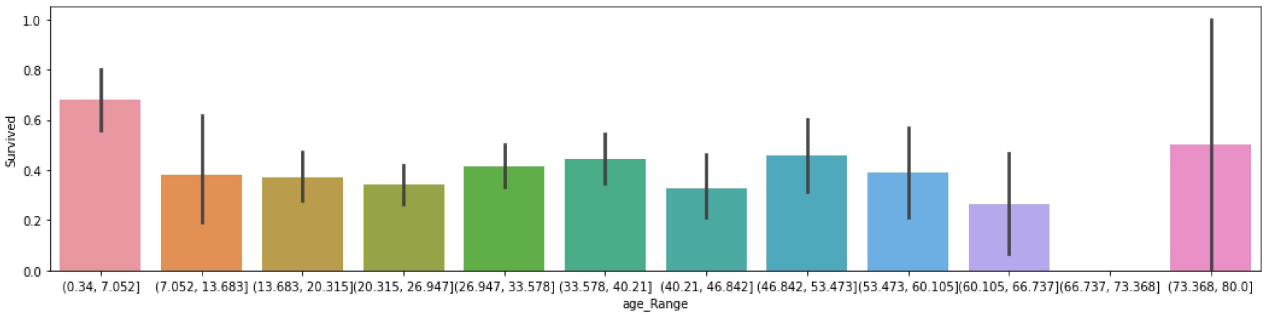


We can see that the most concentrated survival areas are people between 15 and 30. However, this is also the category where most people die. This does not tell us much as this seems very dependent on the volume of people in each age category. To combat this, we need to sort everyone into different age categories and determine the survival %.

Other observations: all death rates are at least a shade darker than the survival rates except for ages 0-10. That means that more people died than survived in every category except for little kids. Let's look into this.

In [11]:
```
plt.figure(figsize=(18,4))
#ages = pd.cut(train['Age'],bins=12)

#Add column to train and test dfs
train['age_Range']= pd.cut(train['Age'],bins=12)
test['age_Range']= pd.cut(test['Age'],bins=12)

sns.barplot(x='age_Range',y='Survived',data = train)
```

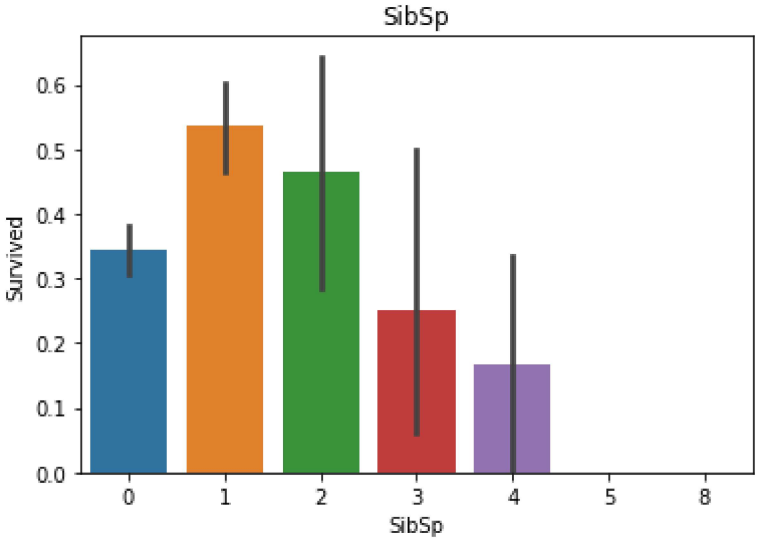
Out[11]: <AxesSubplot:xlabel='age_Range', ylabel='Survived'>



It seems like there is not much of a correlation between age and survival rate; about 50% foreach age range. However, we can see that the survival rate for children was very high. The rate for ages 73-80 is also high but looking at our initial age plot, it can be seen that there were a very low number of data points making the value look inflated. By looking at the error bar, it is clear that the rate for ages 73-80 is not reliable. We will drop this later on when cleaning data.

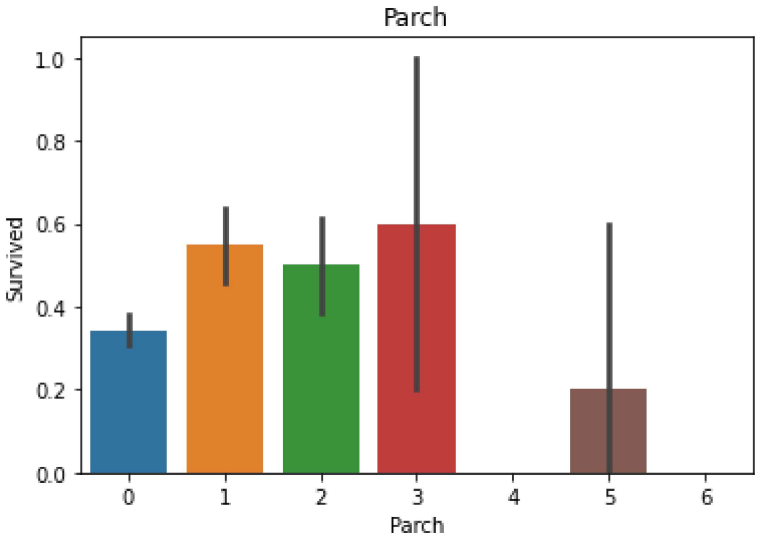
4. SibSp & Parch

In [12]:
```
sns.barplot(x='SibSp', y="Survived", data=train)
plt.title('SibSp')
plt.show()

for i in range(5):
    print("{} SibSp survival rate: {}".format(i, train["Survived"][train["SibSp"] == i]
```




```
0 SibSp survival rate: 0.34539473684210525
1 SibSp survival rate: 0.5358851674641149
2 SibSp survival rate: 0.4642857142857143
3 SibSp survival rate: 0.25
4 SibSp survival rate: 0.16666666666666666
```

In [13]:
```
sns.barplot(x='Parch', y="Survived", data=train)
plt.title('Parch')
plt.show()

for i in [0,1,2,3,5]:
    print("{} Parch survival rate: {}".format(i, train["Survived"][train["Parch"] == i]
```




```
0 Parch survival rate: 0.34365781710914456
1 Parch survival rate: 0.5508474576271186
2 Parch survival rate: 0.5
3 Parch survival rate: 0.6
5 Parch survival rate: 0.2
```

These results are very interesting and unexpected. It looks like travelling with 1 or 2 people increases the survival rate as all 3 people (passenger and two others) would be able to work together and help each other. This is further proven by the small error bars. Travelling with more than that however becomes a problem as we see the survival rate drop from 46% - 25%.

4. Cleaning Data

In this section we will address missing values and prepare our training data for modelling.

In [14]:
```
#Combine datasets to see missing values
all_data=pd.concat([train,test],ignore_index=True)
all_data.describe(include='all')
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
count	1309.000000	891.000000	1309.000000	1309	1309	1046.000000	1309.000000	1309.000000
unique	NaN	NaN	NaN	1307	2	NaN	NaN	NaN
top	NaN	NaN	NaN	Connolly, Miss. Kate	male	NaN	NaN	NaN
freq	NaN	NaN	NaN	2	843	NaN	NaN	NaN
mean	655.000000	0.383838	2.294882	NaN	NaN	29.881138	0.498854	0.385027
std	378.020061	0.486592	0.837836	NaN	NaN	14.413493	1.041658	0.865560
min	1.000000	0.000000	1.000000	NaN	NaN	0.170000	0.000000	0.000000
25%	328.000000	0.000000	2.000000	NaN	NaN	21.000000	0.000000	0.000000
50%	655.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000
75%	982.000000	1.000000	3.000000	NaN	NaN	39.000000	1.000000	0.000000
max	1309.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	9.000000

Across the train and test data, there are 1309 entries. 265 (20%) age values are missing and 2 (0.15%) embarked values are missing.

Age: Age doesn't seem to have a very high correlation with survival rate so for now we can just fill in the missing data with the median age.

#TODO: Predict age based on Title, SibSp, Parch

Embarked: Fill with S

In [15]:
```
for i in [train,test,all_data]:
    i['Age'] = i['Age'].fillna(i['Age'].median())
    i = i.fillna({'Embarked':'S'})

all_data.describe(include='all')
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
count	1309.000000	891.000000	1309.000000	1309	1309	1309.000000	1309.000000	1309.000000
unique	NaN	NaN	NaN	1307	2	NaN	NaN	NaN
top	NaN	NaN	NaN	Connolly, Miss. Kate	male	NaN	NaN	NaN
freq	NaN	NaN	NaN	2	843	NaN	NaN	NaN
mean	655.000000	0.383838	2.294882	NaN	NaN	29.503186	0.498854	0.385027
std	378.020061	0.486592	0.837836	NaN	NaN	12.905241	1.041658	0.865560
min	1.000000	0.000000	1.000000	NaN	NaN	0.170000	0.000000	0.000000
25%	328.000000	0.000000	2.000000	NaN	NaN	22.000000	0.000000	0.000000
50%	655.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000
75%	982.000000	1.000000	3.000000	NaN	NaN	35.000000	1.000000	0.000000
max	1309.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	9.000000

Nice all our values are filled. We can finally start modelling.

5. Modelling

We will test an assortment of different machine learning models from simplest (at least in my mind) to most complex.

1. Decision Tree
2. Random Forest
3. Naive Bayes
4. Logistic Regression
5. KNN (K Nearest Neighbor)
6. Support Vector Machines
7. Perceptron
8. Stochastic Gradient Descent
9. Xtreme Gradient Boosting Classifier

In [16]:

```
train.head()
test.head()
```

Out[16]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	

In [17]:
```
#We will need to save a copy of the passenger ids for our output file later on
passenger_id = test['PassengerId']

#Create dummy variables for so that Sex can be numerically categorized
train = pd.get_dummies(train[['Pclass','Sex','Age','SibSp','Parch','Embarked','Survived
test = pd.get_dummies(test[['Pclass','Sex','Age','SibSp','Parch','Embarked']])

train.head()
```

Out[17]:

	Pclass	Age	SibSp	Parch	Survived	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	3	22.0	1	0	0	0	1	0	0	1
1	1	38.0	1	0	1	1	0	1	0	0
2	3	26.0	0	0	1	1	0	0	0	1
3	1	35.0	1	0	1	1	0	0	0	1
4	3	35.0	0	0	0	0	1	0	0	1

In [18]:
```
# Split data
from sklearn.model_selection import train_test_split
X_train = train.drop(['Survived'],axis=1)
y_train = train['Survived']

train_X, val_X, train_y, val_y = train_test_split(X_train,y_train,test_size = 0.2, rand

train_X.shape,train_y.shape,val_X.shape, val_y.shape
```

Out[18]: ((712, 9), (712,), (179, 9), (179,))

In [19]:

```
#Scale data
from mlxtend.preprocessing import minmax_scaling
minmax_scaling(X_train,columns=['Pclass','Age'])
```

Out[19]:

	Pclass	Age
0	1.0	0.271174
1	0.0	0.472229
2	1.0	0.321438
3	0.0	0.434531
4	1.0	0.434531
...	...	...
886	0.5	0.334004
887	0.0	0.233476
888	1.0	0.346569
889	0.0	0.321438
890	1.0	0.396833

891 rows × 2 columns

In [20]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score

# Decision Tree
# Random Forest
# Naive Bayes
# Logistic Regression
# KNN (K Nearest Neighbor)
# Support Vector Machines
# Perceptron
# Stochastic Gradient Descent
# Xtreme Gradient Boosting Classifier

from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()

from sklearn.ensemble import RandomForestClassifier
random_Forest = RandomForestClassifier()

from sklearn.naive_bayes import GaussianNB
naive_bayes = RandomForestClassifier()

from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression()

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

from sklearn.svm import SVC
svc = SVC()
```

```
from sklearn.linear_model import Perceptron
perceptron = Perceptron()

from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()

from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
```

In [21]:

```
models = [decision_tree,random_Forest,naive_bayes,logReg,knn,svc,perceptron,sgd,gbc]

for i in models:
    i.fit(train_X,train_y)
    pred_y = i.predict(val_X)

    print(i)
    print ('Mean absolute error: ',mean_absolute_error(pred_y,val_y))
    print ('Accuracy score: ', accuracy_score(pred_y,val_y),'\n')
```

```
DecisionTreeClassifier()
Mean absolute error:  0.19553072625698323
Accuracy score:  0.8044692737430168

RandomForestClassifier()
Mean absolute error:  0.1787709497206704
Accuracy score:  0.8212290502793296

RandomForestClassifier()
Mean absolute error:  0.18435754189944134
Accuracy score:  0.8156424581005587

LogisticRegression()
Mean absolute error:  0.18435754189944134
Accuracy score:  0.8156424581005587

KNeighborsClassifier()
Mean absolute error:  0.19553072625698323
Accuracy score:  0.8044692737430168

SVC()
Mean absolute error:  0.3575418994413408
Accuracy score:  0.6424581005586593

Perceptron()
Mean absolute error:  0.3016759776536313
Accuracy score:  0.6983240223463687

SGDClassifier()
Mean absolute error:  0.3128491620111732
Accuracy score:  0.6871508379888268

GradientBoostingClassifier()
Mean absolute error:  0.1564245810055866
Accuracy score:  0.8435754189944135
```

Of all the models, Gradient Boosting Classifier has the highest score. Of course, that does not mean it is the best model as it may only work well for our specific test data however we will select this model to refine.

6. Refining Model

From the documentation, the two most important parameters for Gradient Boosting Classifier are max depth and the number of estimators. Let's test a few cases using values that are close to the default.

In [22]:
```
#Creates Gradient Boosting Classifier model based on number of depth and estiators

def gbc_accuracy (max_depth,n_estimators):
    gbc = GradientBoostingClassifier(max_depth = d, n_estimators = n)
    gbc.fit(train_X,train_y)
    pred_y = gbc.predict(val_X)
    accuracy = accuracy_score(pred_y,val_y)
    return accuracy
```

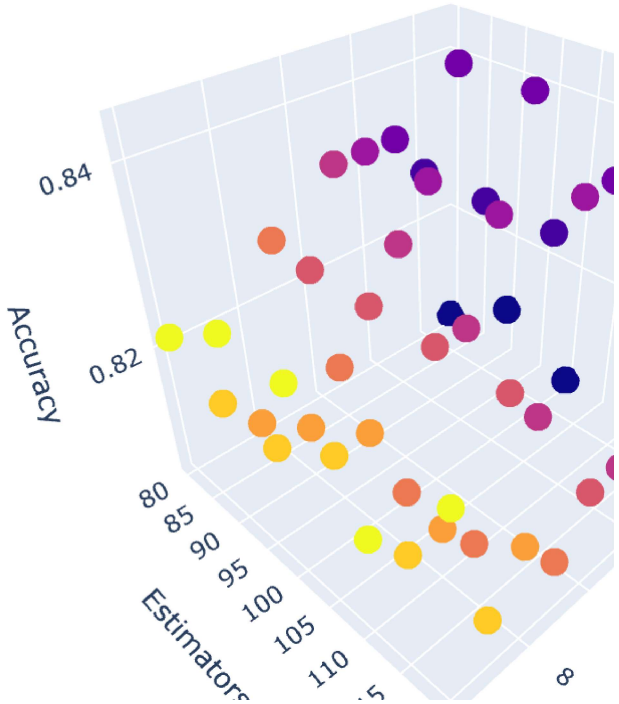
In [23]:
```
candidate_max_depth = [1,2,3,4,5,6,7,8,9,10]
candidate_n_estimators = [80,90,100,110,120]

gbc_df = pd.DataFrame(columns=['Depth','Estimators','Accuracy'])
gbc_df

for d in candidate_max_depth:
    for n in candidate_n_estimators:

        ser=pd.Series([d,n,gbc_accuracy(d,n)],index=gbc_df.columns)
        gbc_df = gbc_df.append(ser,ignore_index=True)

gbc_df.head()

# parameters = []
# for d in candidate_max_depth:
#     for n in candidate_n_estimators:
#         dictionary = {'Depth':d,'Estimators':n,'Accuracy':gbc_accuracy(d,n)}
#         list.append(dictionary)
# gbc_df = pd.DataFrame(parameters)

# best_parameters = max(scores,key=scores.get)
# best_parameters
```

Out[23]:

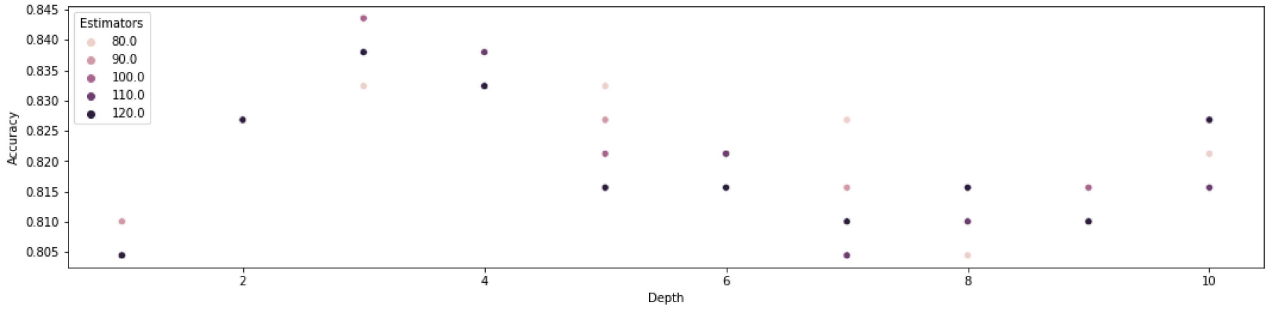
	Depth	Estimators	Accuracy
0	1.0	80.0	0.804469
1	1.0	90.0	0.810056
2	1.0	100.0	0.804469
3	1.0	110.0	0.804469
4	1.0	120.0	0.804469

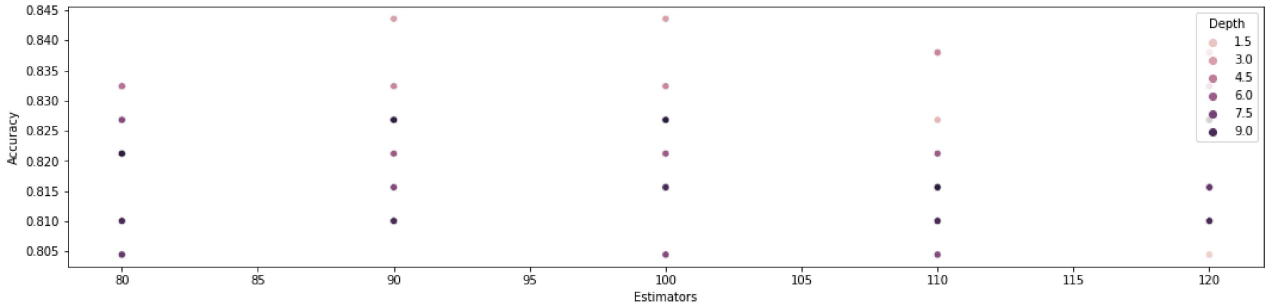
In [24]:
```
fig = px.scatter_3d(gbc_df, x='Depth', y='Estimators', z='Accuracy',color='Depth')
fig.show()

plt.figure(figsize=(18,4))
sns.scatterplot(x=gbc_df['Depth'],y=gbc_df['Accuracy'],hue=gbc_df['Estimators'])
```

```
plt.show

plt.figure(figsize=(18,4))
sns.scatterplot(x=gbc_df['Estimators'],y=gbc_df['Accuracy'],hue=gbc_df['Depth'])
plt.show
```



Out[24]: <function matplotlib.pyplot.show(close=None, block=None)>





As predicted, depth has a slightly higher affect on the model than the number of estimators. From all three charts, we can see that the ideal values are a depth of 3 and 100 estimators. With this in mind, lets build our final model and make a prediction!

In [25]:
```
gbc = GradientBoostingClassifier(max_depth = 3, n_estimators = 100)
gbc.fit(train_X,train_y)
```

Out[25]:
```
GradientBoostingClassifier()
```

7. Creating Submission File

In [26]:
```
prediction = gbc.predict(test)

#set the output as a dataframe and convert to csv file named submission.csv
#Remember that the passenger_id column was declared when creating the dummy variables
result = pd.DataFrame({ 'PassengerId' : passenger_id, 'Survived': prediction })
result.to_csv('submission.csv', index=False)
result
```

Out[26]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns