TITANIC

Sankeerthana Satini

Create a model that predicts which passengers survived the Titanic shipwreck.

This would be a classification problem as either the person survives or does not survive. Another question from this would be feature importance, as to which features are important in the prediction. The passengers can be filtered out based on the predictions.

"...which passengers survived the Titanic Shipwreck", which classes of which features have a higher probability to get 1 under the 'Survived' feature.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
```

In [2]:

```
train = pd.read_csv("train.csv")
```

In [3]:

```
train.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

In [4]:

```
train.columns
```

Out[4]:

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

Looking from this, I can say that there are 12 columns, out of which the label is the column Survived. Therefore, there are 11 Features and 1 Label.

In [5]:

```
pass_id = train[["PassengerId"]]
```

Description of the Variables

Variable Definition Key

survival Survival 0 = No, 1 = Yes pclass Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd

sex Sex
Age Age in years
sibsp # of siblings / spouses aboard the Titanic
parch # of parents / children aboard the Titanic
ticket Ticket number
fare Passenger fare
cabin Cabin number
embarked Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes pclass: A proxy for socio-economic status (SES) 1st = Upper 2nd = Middle 3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way... Sibling = brother, sister, stepbrother, stepsister Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way... Parent = mother, father Child = daughter, son, stepdaughter, stepson Some children travelled only with a nanny, therefore parch=0 for them

In [6]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

In [7]:

```
label = train["Survived"]
```

In [8]:

```
type(label)
```

Out[8]:

```
pandas.core.series.Series
```

In [9]:

```
#Converting from Series to DataFrame
label = pd.DataFrame(data=label)
```

In [10]:

```
#Dropping 'survived' from the dataset as it is the label, and better to have it as the last
#Random Forest
train = train.drop(["Survived"], axis=1)
```

In [11]:

```
train.head()
```

Out[11]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

Since PassengerId and Name are unique features, it is not very useful in the EDA and the prediction of the model and can be omitted

In [12]:

```
#Dropping the PassengerId and Name from the dataset
train = train.drop(["PassengerId","Name"], axis=1)
```

In [13]:

```
train.head()
```

Out[13]:

	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	1	female	35.0	1	0	113803	53.1000	C123	S
4	3	male	35.0	0	0	373450	8.0500	NaN	S

In [14]:

```
#Finding the number of NaN values to carry out NA Imputation
train.isnull().sum()
```

Out[14]:

```
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Ticket        0
Fare          0
Cabin       687
Embarked      2
dtype: int64
```

Age is float64 data type and therefore can be imputed using NA Imputation using Mean value.

Cabin is of the object data type.

In [15]:

```
#Importing the relevant library
from sklearn.impute import SimpleImputer
```

In [16]:

```
#Setting the imputer to replace NaN values with the mean
imp_1 = SimpleImputer(missing_values=np.nan, strategy='mean')
```

Age

In [17]:

```
#age is a Series
age = train["Age"]
```

In [18]:

```
#Converting from Series to DataFrame
age = pd.DataFrame(data=age)
```

In [19]:

```
age.head()
```

Out[19]:

	Age
0	22.0
1	38.0
2	26.0
3	35.0
4	35.0

In [20]:

```
#Need to do this step as simple imputer takes in 2D array - age and fare
fare = train["Fare"]
```

In [21]:

```
fare = pd.DataFrame(data=fare)
```

In [22]:

```
age_fare = pd.concat([age,fare],axis=1)
```

In [23]:

```
age_fare.head()
```

Out[23]:

	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500

In [24]:

```
age_fare.isnull().sum()
```

Out[24]:

```
Age     177
Fare      0
dtype: int64
```

In [25]:

```
age_fare = imp_1.fit_transform(age_fare)
```

In [26]:

```
age_fare
```

Out[26]:

```
array([[22.        ,  7.25      ],
       [38.        , 71.2833    ],
       [26.        ,  7.925     ],
       ...,
       [29.69911765, 23.45      ],
       [26.        , 30.        ],
       [32.        ,  7.75      ]])
```

In [27]:

```
age_fare = pd.DataFrame(data=age_fare,columns=["Age", "Fare"])
```

In [28]:

```
age_fare.head()
```

Out[28]:

	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500

In [29]:

```
age_fare.isnull().sum()
```

Out[29]:

```
Age     0
Fare    0
dtype: int64
```

In [30]:

```
age = age_fare["Age"]
```

In [31]:

```
age = pd.DataFrame(data=age)
```

In [32]:

```
age.head()
```

Out[32]:

	Age
0	22.0
1	38.0
2	26.0
3	35.0
4	35.0

In [33]:

```
train = train.drop(["Age"],axis=1)
```

In [34]:

```
train.head()
```

Out[34]:

	Pclass	Sex	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	3	male	1	0	A/5 21171	7.2500	NaN	S
1	1	female	1	0	PC 17599	71.2833	C85	C
2	3	female	0	0	STON/O2. 3101282	7.9250	NaN	S
3	1	female	1	0	113803	53.1000	C123	S
4	3	male	0	0	373450	8.0500	NaN	S

In [35]:

```
train = pd.concat([train,age],axis=1)
```

In [36]:

```
train.head()
```

Out[36]:

	Pclass	Sex	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age
0	3	male	1	0	A/5 21171	7.2500	NaN	S	22.0
1	1	female	1	0	PC 17599	71.2833	C85	C	38.0
2	3	female	0	0	STON/O2. 3101282	7.9250	NaN	S	26.0
3	1	female	1	0	113803	53.1000	C123	S	35.0
4	3	male	0	0	373450	8.0500	NaN	S	35.0

Cabin

In [37]:

```
cabin = train["Cabin"]
```

In [38]:

```
cabin = pd.DataFrame(data=cabin)
```

In [39]:

```
fare = train["Fare"]
```

In [40]:

```
fare = pd.DataFrame(data=fare)
```

In [41]:

```
cab_fare = pd.concat([cabin,fare],axis=1)
```

In [42]:

```
cab_fare.head()
```

Out[42]:

	Cabin	Fare
0	NaN	7.2500
1	C85	71.2833
2	NaN	7.9250
3	C123	53.1000
4	NaN	8.0500

In [43]:

```
cab_fare.isnull().sum()
```

Out[43]:

```
Cabin    687
Fare       0
dtype: int64
```

In [44]:

```
imp_2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

In [45]:

```
cab_fare = imp_2.fit_transform(cab_fare)
```

In [46]:

```
cab_fare
```

Out[46]:

```
array([['B96 B98', 7.25],
       ['C85', 71.2833],
       ['B96 B98', 7.925],
       ...,
       ['B96 B98', 23.45],
       ['C148', 30.0],
       ['B96 B98', 7.75]], dtype=object)
```

In [47]:

```
cab_fare = pd.DataFrame(data=cab_fare, columns=["Cabin","Fare"])
```

In [48]:

```
cab_fare.isnull().sum()
```

Out[48]:

```
Cabin    0
Fare     0
dtype: int64
```

In [49]:

```
cabin = cab_fare["Cabin"]
```

In [50]:

```
train = train.drop(["Cabin"],axis=1)
```

In [51]:

```
train = pd.concat([train,cabin],axis=1)
```

In [52]:

```
train.head()
```

Out[52]:

	Pclass	Sex	SibSp	Parch	Ticket	Fare	Embarked	Age	Cabin
0	3	male	1	0	A/5 21171	7.2500	S	22.0	B96 B98
1	1	female	1	0	PC 17599	71.2833	C	38.0	C85
2	3	female	0	0	STON/O2. 3101282	7.9250	S	26.0	B96 B98
3	1	female	1	0	113803	53.1000	S	35.0	C123
4	3	male	0	0	373450	8.0500	S	35.0	B96 B98

Embarked

In [53]:

```
emb = train["Embarked"]
```

In [54]:

```
emb = pd.DataFrame(data=emb)
```

In [55]:

```
emb.head()
```

Out[55]:

	Embarked
0	S
1	C
2	S
3	S
4	S

In [56]:

```
emb_fare = pd.concat([emb,fare],axis=1)
```

In [57]:

```
emb_fare.isnull().sum()
```

Out[57]:

```
Embarked    2
Fare        0
dtype: int64
```

In [58]:

```
emb_fare = imp_2.fit_transform(emb_fare)
```

In [59]:

```
emb_fare
```

Out[59]:

```
array([['S', 7.25],
       ['C', 71.2833],
       ['S', 7.925],
       ...,
       ['S', 23.45],
       ['C', 30.0],
       ['Q', 7.75]], dtype=object)
```

In [60]:

```
emb_fare = pd.DataFrame(data=emb_fare,columns=["Embarked","Fare"])
```

In [61]:

```
emb_fare.isnull().sum()
```

Out[61]:

```
Embarked    0
Fare        0
dtype: int64
```

In [62]:

```
emb = emb_fare["Embarked"]
```

In [63]:

```
emb = pd.DataFrame(data=emb)
```

In [64]:

```
train = train.drop(["Embarked"],axis=1)
```

In [65]:

```
train = pd.concat([train,emb],axis=1)
```

In [66]:

```
train.isnull().sum()
```

Out[66]:

```
Pclass      0
Sex         0
SibSp       0
Parch       0
Ticket      0
Fare        0
Age         0
Cabin       0
Embarked    0
dtype: int64
```

In [67]:

```
train.head()
```

Out[67]:

	Pclass	Sex	SibSp	Parch	Ticket	Fare	Age	Cabin	Embarked
0	3	male	1	0	A/5 21171	7.2500	22.0	B96 B98	S
1	1	female	1	0	PC 17599	71.2833	38.0	C85	C
2	3	female	0	0	STON/O2. 3101282	7.9250	26.0	B96 B98	S
3	1	female	1	0	113803	53.1000	35.0	C123	S
4	3	male	0	0	373450	8.0500	35.0	B96 B98	S

Exploratory Data Analysis

In [68]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
Pclass      891 non-null int64
Sex         891 non-null object
SibSp       891 non-null int64
Parch       891 non-null int64
Ticket      891 non-null object
Fare        891 non-null float64
Age         891 non-null float64
Cabin       891 non-null object
Embarked    891 non-null object
dtypes: float64(2), int64(3), object(4)
memory usage: 62.7+ KB
```

In [69]:

```
#Only considers the numeric values
train.describe()
```

Out[69]:

	Pclass	SibSp	Parch	Fare	Age
count	891.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	0.523008	0.381594	32.204208	29.699118
std	0.836071	1.102743	0.806057	49.693429	13.002015
min	1.000000	0.000000	0.000000	0.000000	0.420000
25%	2.000000	0.000000	0.000000	7.910400	22.000000
50%	3.000000	0.000000	0.000000	14.454200	29.699118
75%	3.000000	1.000000	0.000000	31.000000	35.000000
max	3.000000	8.000000	6.000000	512.329200	80.000000

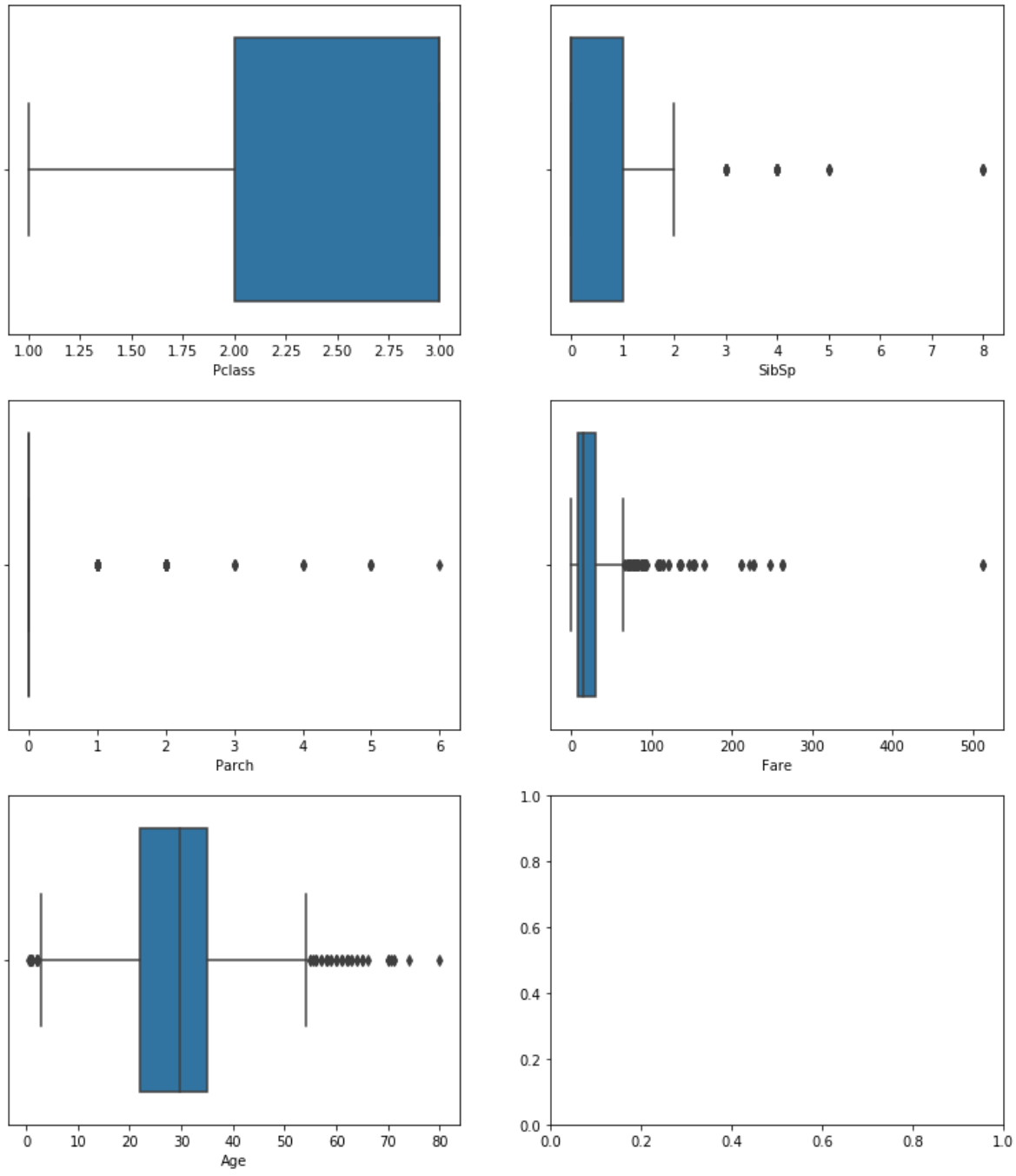
In [70]:

```
f, axes = plt.subplots(3,2 ,figsize=(13,15))

sb.boxplot(train["Pclass"], ax = axes[0, 0])
sb.boxplot(train["SibSp"], ax = axes[0, 1])
sb.boxplot(train["Parch"], ax = axes[1, 0])
sb.boxplot(train["Fare"], ax = axes[1, 1])
sb.boxplot(train["Age"], ax = axes[2, 0])
```

Out[70]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bb45e5c0>
```



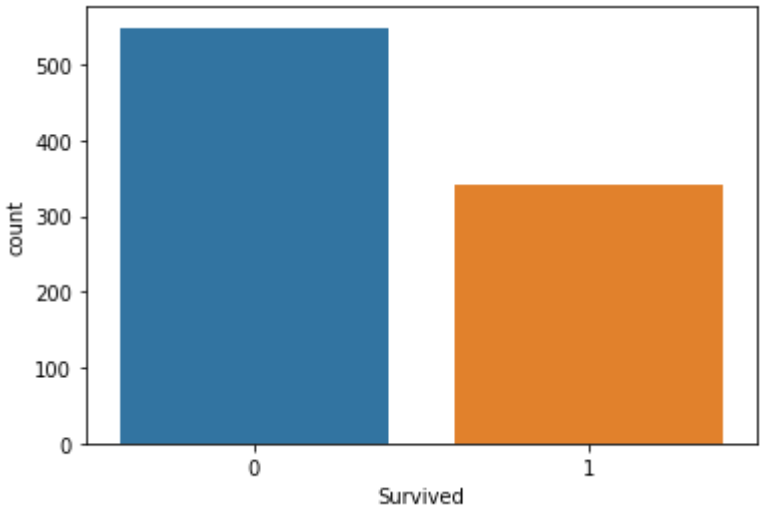
Would not be removing outliers as they might show interesting trends. In addition, since Random Forest will be used, the outliers will be taken care of.

In [71]:

```
#Checking the countplot of the label to look for any unbalanced classes
sb.countplot(x="Survived", data=label)
#sns.countplot(x="geo_level_1_id", data=df, hue='damage_grade')
```

Out[71]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bb9cbdd8>
```



In [72]:

```
#TO convert into series form df, just extract that row out
label = label["Survived"]
```

In [73]:

```
type(label)
```

Out[73]:

```
pandas.core.series.Series
```

In [74]:

```
label.value_counts()
```

Out[74]:

```
0    549
1    342
Name: Survived, dtype: int64
```

In [75]:

```
549/342
```

Out[75]:

```
1.605263157894737
```

Not going to balance the data first as the ratio is somewhat close 1.60:1

In [76]:

```
label = pd.DataFrame(data=label)
```

In [77]:

```
label.isnull().sum()
```

Out[77]:

```
Survived    0
dtype: int64
```

In [78]:

```
#Joining the label with the feature dataset
train = pd.concat([train,label],axis=1)
```

In [79]:

```
train.head()
```

Out[79]:

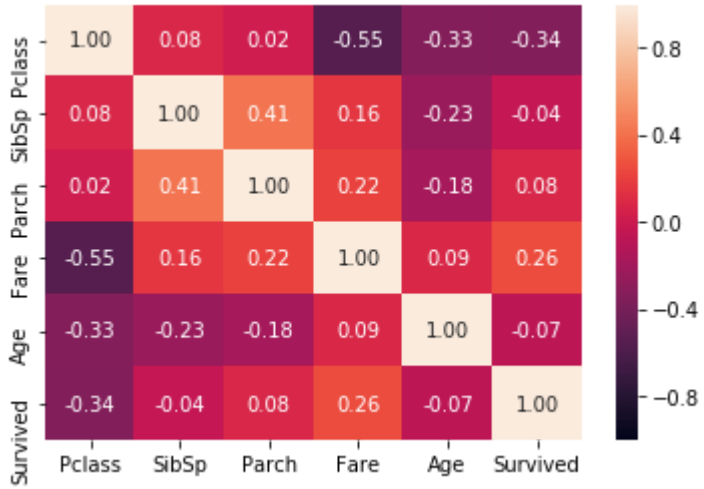
	Pclass	Sex	SibSp	Parch	Ticket	Fare	Age	Cabin	Embarked	Survived
0	3	male	1	0	A/5 21171	7.2500	22.0	B96 B98	S	0
1	1	female	1	0	PC 17599	71.2833	38.0	C85	C	1
2	3	female	0	0	STON/O2. 3101282	7.9250	26.0	B96 B98	S	1
3	1	female	1	0	113803	53.1000	35.0	C123	S	1
4	3	male	0	0	373450	8.0500	35.0	B96 B98	S	0

In [80]:

```
#Only considers the numeric features
sb.heatmap(train.corr(), vmin=-1, vmax=1, annot = True, fmt=".2f")
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bba40f98>
```



The highest negative correlation with survived is -0.55, between Pclass and Survived and the highest positive correlation is 0.26 which is in between Fare and Survived. However, this only includes the numeric features and not the features of the datatype object. Therefore, we would need to use feature importance after Random Forest Classifier.

FYI

–1. A perfect negative (downward sloping) linear relationship

–0.70. A strong negative (downward sloping) linear relationship

–0.50. A moderate negative (downhill sloping) relationship

–0.30. A weak negative (downhill sloping) linear relationship

Comparing Gender and Survived

In [81]:

```
sex = train["Sex"]
```

In [82]:

```
sex = pd.DataFrame(data=sex)
```

In [83]:

```
gender = pd.concat([sex,label],axis=1)
```

In [84]:

```
gender.head()
```

Out[84]:

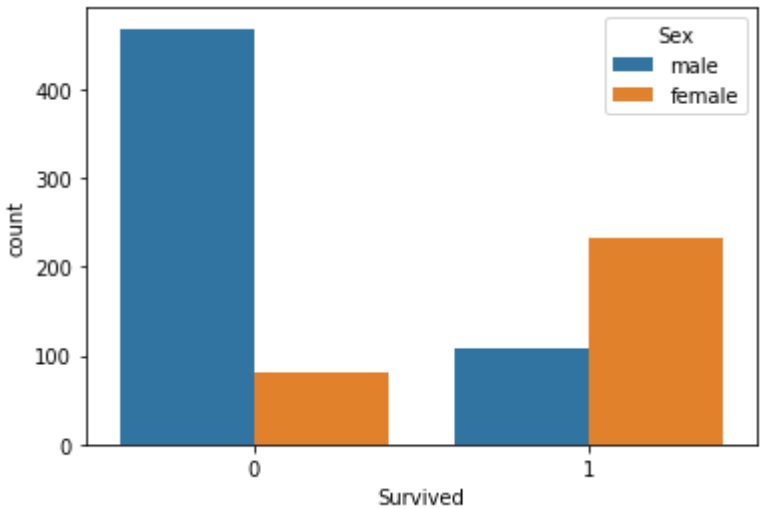
	Sex	Survived
0	male	0
1	female	1
2	female	1
3	female	1
4	male	0

In [85]:

```
sb.countplot(x="Survived", data=gender, hue="Sex")
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bb6b9e80>
```



From this graph, we can see that out of those who survived, most were Female. A surprising number of men did not survive. This would be because from prior knowledge we know that women and children were first sent out of the ship, before men, as an act of chivalry.

Comparing Age and Survived

In [86]:

```
age = train["Age"]
```

In [87]:

```
age.value_counts()
```

Out[87]:

```
29.699118    177
24.000000     30
22.000000     27
18.000000     26
28.000000     25
30.000000     25
19.000000     25
21.000000     24
25.000000     23
36.000000     22
29.000000     20
35.000000     18
26.000000     18
32.000000     18
27.000000     18
31.000000     17
16.000000     17
23.000000     15
34.000000     15
33.000000     15
20.000000     15
39.000000     14
40.000000     13
42.000000     13
17.000000     13
45.000000     12
38.000000     11
4.000000      10
50.000000     10
2.000000      10
             ...
0.830000       2
30.500000      2
0.750000       2
57.000000      2
55.000000      2
70.000000      2
10.000000      2
32.500000      2
71.000000      2
63.000000      2
28.500000      2
45.500000      2
40.500000      2
59.000000      2
14.500000      1
0.670000       1
12.000000      1
0.920000       1
74.000000      1
34.500000      1
70.500000      1
36.500000      1
24.500000      1
66.000000      1
```

```
80.000000      1
55.500000      1
53.000000      1
20.500000      1
23.500000      1
0.420000       1
Name: Age, Length: 89, dtype: int64
```

In [88]:

```
age = pd.DataFrame(data=age)
```

In [89]:

```
age = pd.concat([age,label],axis=1)
```

In [90]:

```
age.head()
```

Out[90]:

	Age	Survived
0	22.0	0
1	38.0	1
2	26.0	1
3	35.0	1
4	35.0	0

In [91]:

```
f, axes = plt.subplots(1,1 ,figsize=(100,5))
sb.countplot(x="Age", data=age, hue="Survived")
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bace9320>
```

As seen from the graph, most of the people were of the age 29.69918, out of which, 127 did not survive and 50 did.

Comparing Cabin and Survived

In [92]:

```
cabin = train["Cabin"]
```

In [93]:

```
cabin = pd.DataFrame(data=cabin)
```

In [94]:

```
cabin = pd.concat([cabin,label],axis=1)
```

In [95]:

```
cabin.head()
```

Out[95]:

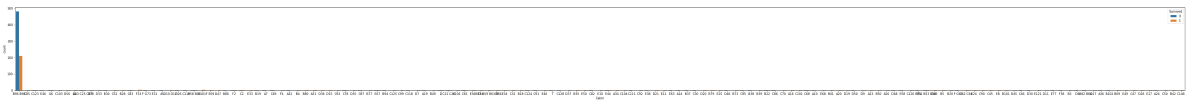
	Cabin	Survived
0	B96 B98	0
1	C85	1
2	B96 B98	1
3	C123	1
4	B96 B98	0

In [96]:

```
f, axes = plt.subplots(1,1 ,figsize=(70,5))
sb.countplot(x="Cabin", data=cabin, hue="Survived")
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a9bc7f0400>
```



Most of the people who survived was from B96 Cabin, and most of the people who did not survive were B98. However, this could have been due to NA Imputation where the NA Values were replaced with the Most Frequently occuring String.

Overall, from the EDA, we can say that the features which prove to have a strong impact on whether the passengers survived is Sex, Age, PClass and Fare.

In [97]:

```
train.head()
```

Out[97]:

	Pclass	Sex	SibSp	Parch	Ticket	Fare	Age	Cabin	Embarked	Survived
0	3	male	1	0	A/5 21171	7.2500	22.0	B96 B98	S	0
1	1	female	1	0	PC 17599	71.2833	38.0	C85	C	1
2	3	female	0	0	STON/O2. 3101282	7.9250	26.0	B96 B98	S	1
3	1	female	1	0	113803	53.1000	35.0	C123	S	1
4	3	male	0	0	373450	8.0500	35.0	B96 B98	S	0

One-Hot Encoding

In [98]:

```
#Finding out the categorical variables of the data type object
train.select_dtypes(['object']).columns
```

Out[98]:

```
Index(['Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

In [99]:

```
#Getting the 1s and 0s
dummies = pd.get_dummies(train[['Sex', 'Ticket', 'Cabin', 'Embarked']], drop_first=False)
```

In [100]:

```
#Viewing the dummies
dummies.head()
```

Out[100]:

	Sex_female	Sex_male	Ticket_110152	Ticket_110413	Ticket_110465	Ticket_110564	Ticket_11
0	0	1	0	0	0	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	0	0	
4	0	1	0	0	0	0	

5 rows × 833 columns

In [101]:

```
#Dropping the original object data type features to make way for the new ones
train = train.drop(['Sex', 'Ticket', 'Cabin', 'Embarked', 'Survived'], axis=1)
```

In [102]:

```
#Concatenating the dummies and the original dataset
train = pd.concat([train, dummies], axis=1)
```

In [103]:

```
train.head()
```

Out[103]:

	Pclass	SibSp	Parch	Fare	Age	Sex_female	Sex_male	Ticket_110152	Ticket_110413	1
0	3	1	0	7.2500	22.0	0	1	0	0	
1	1	1	0	71.2833	38.0	1	0	0	0	
2	3	0	0	7.9250	26.0	1	0	0	0	
3	1	1	0	53.1000	35.0	1	0	0	0	
4	3	0	0	8.0500	35.0	0	1	0	0	

5 rows × 838 columns

In [104]:

```
train.isnull().sum()
```

Out[104]:

```
Pclass           0
SibSp            0
Parch            0
Fare             0
Age              0
Sex_female       0
Sex_male         0
Ticket_110152    0
Ticket_110413    0
Ticket_110465    0
Ticket_110564    0
Ticket_110813    0
Ticket_111240    0
Ticket_111320    0
Ticket_111361    0
Ticket_111369    0
Ticket_111426    0
Ticket_111427    0
Ticket_111428    0
Ticket_112050    0
Ticket_112052    0
Ticket_112053    0
Ticket_112058    0
Ticket_112059    0
Ticket_112277    0
Ticket_112379    0
Ticket_113028    0
Ticket_113043    0
Ticket_113050    0
Ticket_113051    0
                ..
Cabin_E24        0
Cabin_E25        0
Cabin_E31        0
Cabin_E33        0
Cabin_E34        0
Cabin_E36        0
Cabin_E38        0
Cabin_E40        0
Cabin_E44        0
Cabin_E46        0
Cabin_E49        0
Cabin_E50        0
Cabin_E58        0
Cabin_E63        0
Cabin_E67        0
Cabin_E68        0
Cabin_E77        0
Cabin_E8         0
Cabin_F E69      0
Cabin_F G63      0
Cabin_F G73      0
Cabin_F2         0
Cabin_F33        0
Cabin_F38        0
```

```
Cabin_F4          0
Cabin_G6          0
Cabin_T           0
Embarked_C        0
Embarked_Q        0
Embarked_S        0
Length: 838, dtype: int64
```

Random Forest

Training the Model

In [105]:

```
label.head()
```

Out[105]:

	Survived
0	0
1	1
2	1
3	1
4	0

In [106]:

```
train.shape
```

Out[106]:

```
(891, 838)
```

In [107]:

```
label.shape
```

Out[107]:

```
(891, 1)
```

In [108]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [109]:

```
#Splitting the data randomly into train set and test set
X_train, X_test, y_train, y_test = train_test_split(train, label, test_size=0.3)
```

In [110]:

```
#TRAINING THE RANDOM FOREST ALGORITHM
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier() #All the parameters are of its default value

#FEW IMPORTANT PARAMETERS OF RANDOM FOREST CLASSIFIER:

#n_estimators --> no of decision trees in a forest - a forest is a collection of decision t

#bootstrap --> whether bootstrap samples are used when building trees.
#If False then, the whole dataset is used to build each tree which is not
#what we want, as then the Trees will be identical to each other

#random_state --> Controls both the randomness of the bootstrapping of the
#samples when buiding trees if bootstrap = True, AND the sampling of the
#features to consider when looking for the best split at each node.

# There are 3 instances:
# 1. None --> default - use global random state numpy.random
# 2. int --> most popular seeded values are 0 and 42
# 3. numpy.random.RandomState instance

#FITTING THE ALGORITHM
model = classifier.fit(X_train, y_train.values.ravel())

#PREDICTING THE DAMAGE GRADE ON THE TEST DATA
predictions = classifier.predict(X_test)
```

In [111]:

```
model
```

Out[111]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [112]:

```
predictions = classifier.predict(X_test)
```

Obtaining Evaluation Metrics

In [113]:

```
#The Metrics used to evaluate classifiation problems are:
# 1. Accuracy
# 2. Confusion Matrix
# 3. Precision Recall
# 4. F1 Values

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_scc

print("Confusion Matrix:")
print(confusion_matrix(y_test,predictions))
print("\n")

print("CLassification Report:")
print(classification_report(y_test,predictions))
print("\n")

print("Accuracy Score:")
print(accuracy_score(y_test,predictions))
print("\n")

print("f1_score: ")
print(f1_score(y_test,predictions, average='micro'))
```

```
Confusion Matrix:
[[156   9]
 [ 34  69]]


CLassification Report:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       165
           1       0.88      0.67      0.76       103

    accuracy                           0.84       268
   macro avg       0.85      0.81      0.82       268
weighted avg       0.85      0.84      0.83       268



Accuracy Score:
0.8395522388059702


f1_score:
0.8395522388059702
```

Overall accuracy is 0.839

In [114]:

```
a_Xtr = X_train
```

In [115]:

```
a_ytr = y_train
```

In [116]:

```
X_test.head()
```

Out[116]:

	Pclass	SibSp	Parch	Fare	Age	Sex_female	Sex_male	Ticket_110152	Ticket_1
411	3	0	0	6.8583	29.699118	0	1	0	
49	3	1	0	17.8000	18.000000	1	0	0	
324	3	8	2	69.5500	29.699118	0	1	0	
149	2	0	0	13.0000	42.000000	0	1	0	
429	3	0	0	8.0500	32.000000	0	1	0	

5 rows × 838 columns

In [117]:

```
y_train.head()
```

Out[117]:

	Survived
890	0
597	0
781	1
742	1
361	0

In [118]:

```
y_test.head()
```

Out[118]:

	Survived
411	0
49	0
324	0
149	0
429	1

Feature Importance

In [119]:

```
from sklearn.feature_selection import SelectFromModel
```

In [120]:

```
#Finding the Importance of each Feature
model.feature_importances_
```

Out[120]:

```
array([5.24299877e-02, 2.34209603e-02, 2.63407185e-02, 9.00719508e-02,
       9.26652391e-02, 1.02134854e-01, 1.13025082e-01, 4.50905635e-04,
       6.68633307e-04, 3.33189849e-04, 1.06276434e-03, 0.00000000e+00,
       2.45980197e-04, 1.86377896e-04, 5.23575271e-04, 9.00546705e-04,
       2.65680487e-03, 2.96707408e-03, 2.49325916e-03, 1.84503494e-04,
       1.42546661e-04, 0.00000000e+00, 1.46185179e-04, 7.44822710e-05,
       0.00000000e+00, 0.00000000e+00, 5.39638156e-04, 0.00000000e+00,
       1.97574802e-04, 0.00000000e+00, 0.00000000e+00, 4.51433178e-04,
       5.42540049e-04, 3.53864820e-04, 5.64355293e-04, 2.23929857e-04,
       5.13882894e-04, 0.00000000e+00, 4.99104109e-04, 0.00000000e+00,
       3.56007686e-03, 3.64561581e-04, 3.19499493e-04, 2.87835282e-04,
       2.15167595e-03, 3.42498241e-04, 0.00000000e+00, 1.54613445e-03,
       2.02791117e-04, 0.00000000e+00, 1.67002944e-04, 6.04550484e-04,
       2.58633276e-03, 0.00000000e+00, 3.20032290e-04, 0.00000000e+00,
       6.00715802e-04, 1.97011947e-03, 1.45520443e-03, 3.41822957e-04,
       2.45826562e-03, 1.29261817e-03, 0.00000000e+00, 0.00000000e+00,
       1.65930078e-04, 0.00000000e+00, 6.61419998e-04, 1.13589334e-04,
       3.66028521e-04, 1.11511973e-03, 0.00000000e+00, 1.04655543e-04,
       1.04238618e-04, 1.07431426e-04, 4.57692063e-04, 4.94378962e-04,
       1.66552587e-03, 1.12504585e-03, 4.23490683e-04, 2.74267435e-04,
       2.42620999e-04, 1.13384123e-03, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.30079737e-03, 2.11021349e-05, 6.74358729e-03,
       2.72519280e-04, 1.72005242e-03, 1.36275772e-03, 2.09847527e-05,
       0.00000000e+00, 3.28956649e-04, 0.00000000e+00, 1.34487266e-04,
       1.31812426e-03, 0.00000000e+00, 4.89758739e-04, 2.72563173e-04,
       8.43783230e-04, 1.70353727e-03, 5.96828742e-04, 1.19467081e-03,
       1.05951646e-04, 0.00000000e+00, 4.59263093e-06, 3.97330653e-04,
       0.00000000e+00, 8.34876821e-05, 0.00000000e+00, 1.08235037e-04,
       9.36914024e-05, 9.90915140e-04, 4.93017801e-05, 0.00000000e+00,
       0.00000000e+00, 3.71460370e-04, 2.75747592e-04, 1.22478978e-05,
       7.96089450e-04, 5.18271442e-04, 2.69240610e-04, 4.24731710e-04,
       1.74766682e-04, 7.70571547e-05, 1.36726240e-04, 6.07094033e-05,
       0.00000000e+00, 6.16737346e-04, 1.10817220e-04, 4.65179303e-04,
       6.68849435e-05, 0.00000000e+00, 4.47353730e-04, 2.31748901e-05,
       0.00000000e+00, 0.00000000e+00, 2.33810558e-03, 4.69390169e-04,
       5.07198049e-04, 0.00000000e+00, 2.23532983e-04, 5.02296363e-05,
       4.24121463e-05, 0.00000000e+00, 1.53769007e-04, 3.40358234e-04,
       6.43540828e-04, 0.00000000e+00, 6.48252159e-04, 2.52006844e-03,
       4.82923429e-03, 1.66126201e-05, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 6.50141539e-04, 0.00000000e+00, 1.97197318e-03,
       0.00000000e+00, 1.81526461e-05, 1.05688824e-03, 2.31333244e-05,
       0.00000000e+00, 4.26183612e-04, 1.80692871e-05, 0.00000000e+00,
       0.00000000e+00, 1.62759675e-03, 4.91847165e-05, 4.54049116e-05,
       0.00000000e+00, 8.87824159e-04, 0.00000000e+00, 3.65909280e-04,
       1.07197833e-05, 0.00000000e+00, 3.60072937e-03, 1.38962752e-04,
       0.00000000e+00, 1.27120032e-03, 1.18985796e-03, 1.74492731e-03,
       1.01379164e-04, 3.58577046e-04, 2.70128351e-04, 1.02734682e-03,
       0.00000000e+00, 2.52421787e-04, 0.00000000e+00, 1.13320065e-03,
       8.98531855e-04, 2.07560965e-03, 2.64533894e-03, 7.72446383e-04,
       2.43641306e-03, 2.64054346e-04, 0.00000000e+00, 1.21231360e-04,
       1.55607156e-03, 1.02511212e-03, 4.02548073e-04, 1.93497198e-03,
       2.20310841e-04, 0.00000000e+00, 1.86172895e-04, 0.00000000e+00,
       0.00000000e+00, 4.07750802e-03, 2.67443304e-03, 3.16025888e-04,
       1.08977573e-04, 0.00000000e+00, 2.71556907e-04, 0.00000000e+00,
```

```
1.71514791e-03, 6.90499007e-04, 8.94189059e-04, 2.46717034e-04,
0.00000000e+00, 4.64474588e-04, 1.38118656e-04, 2.85281820e-03,
1.90602319e-04, 1.47460989e-03, 2.29400309e-04, 4.58311317e-04,
1.75481868e-04, 1.89439507e-05, 2.33451715e-05, 5.14787267e-04,
0.00000000e+00, 1.78037576e-04, 7.00798935e-05, 0.00000000e+00,
2.21630285e-04, 2.41893983e-04, 0.00000000e+00, 2.68575492e-05,
3.86506857e-03, 9.82164220e-05, 2.53045856e-04, 9.03655653e-04,
2.37822585e-03, 1.39933039e-04, 5.79636576e-04, 4.14934152e-04,
7.50320854e-05, 9.86374764e-05, 1.21074410e-03, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.08586237e-03,
1.02211543e-03, 0.00000000e+00, 0.00000000e+00, 8.75174130e-04,
3.13411168e-05, 4.53122199e-03, 4.48819540e-05, 4.26317627e-05,
5.96963208e-04, 1.59314408e-05, 3.11099447e-05, 6.48958267e-04,
7.28235241e-05, 9.55661875e-05, 6.69998201e-05, 0.00000000e+00,
0.00000000e+00, 3.87194927e-05, 1.53086715e-03, 0.00000000e+00,
4.67875012e-03, 1.46959747e-04, 0.00000000e+00, 0.00000000e+00,
3.77633630e-05, 1.87343421e-05, 0.00000000e+00, 1.79262180e-03,
1.08401202e-03, 6.43080762e-04, 1.03907509e-03, 1.00795130e-03,
1.39630368e-03, 5.65663677e-04, 0.00000000e+00, 0.00000000e+00,
0.00000000e+00, 6.91767743e-05, 0.00000000e+00, 1.39955972e-03,
4.33600687e-04, 0.00000000e+00, 3.09453553e-04, 2.39149399e-05,
1.56991454e-04, 4.75643858e-05, 1.09412277e-03, 1.28597910e-03,
1.25163325e-05, 3.34410257e-05, 3.17383278e-05, 1.25132700e-03,
1.76062761e-03, 1.08595011e-03, 0.00000000e+00, 1.27079278e-05,
2.01948804e-05, 1.13062350e-04, 2.47994534e-03, 4.49687717e-03,
6.36453115e-05, 8.37426180e-05, 4.23328512e-03, 1.00058216e-04,
2.19674253e-05, 6.19076753e-05, 9.25949032e-05, 2.07361612e-03,
5.94533347e-05, 3.30711643e-05, 4.89288913e-05, 6.53551675e-05,
8.75817699e-05, 5.57665300e-05, 0.00000000e+00, 4.31814520e-05,
8.95805030e-04, 9.62728559e-04, 1.18087206e-04, 2.42183389e-04,
4.50803568e-03, 0.00000000e+00, 1.51607849e-03, 1.67561152e-03,
3.35615824e-03, 4.18866992e-03, 0.00000000e+00, 1.65666205e-03,
1.92754515e-03, 4.22487205e-03, 0.00000000e+00, 0.00000000e+00,
0.00000000e+00, 6.14487585e-05, 2.15125071e-03, 2.60269597e-03,
0.00000000e+00, 0.00000000e+00, 1.68961575e-04, 1.32093575e-05,
0.00000000e+00, 6.53709433e-05, 3.88387175e-05, 2.87086737e-05,
1.22204387e-05, 1.43736927e-05, 0.00000000e+00, 4.39685125e-05,
2.04299774e-05, 0.00000000e+00, 6.65278007e-05, 3.43461957e-05,
1.48491125e-05, 2.50901439e-05, 4.56009755e-05, 1.34207069e-05,
2.38363756e-05, 2.83283688e-05, 0.00000000e+00, 1.78601654e-05,
0.00000000e+00, 0.00000000e+00, 1.45776325e-05, 1.19286181e-04,
0.00000000e+00, 9.21072572e-06, 0.00000000e+00, 7.77630340e-04,
1.02213573e-04, 8.02703663e-05, 3.61895975e-03, 1.47606689e-04,
6.84308553e-05, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
2.96705883e-05, 0.00000000e+00, 2.67512311e-05, 3.10946243e-05,
2.60180779e-05, 2.46124502e-05, 0.00000000e+00, 1.07614348e-04,
3.80984002e-04, 1.81236450e-05, 1.13964708e-03, 5.29521261e-05,
2.70340637e-05, 1.26381192e-04, 3.88545419e-05, 2.45269573e-05,
0.00000000e+00, 0.00000000e+00, 2.68750941e-05, 0.00000000e+00,
4.86187988e-03, 1.26317016e-03, 7.32146270e-05, 1.82829688e-04,
1.80957905e-05, 0.00000000e+00, 1.08167683e-05, 9.85061684e-05,
0.00000000e+00, 8.42135737e-04, 0.00000000e+00, 3.68785233e-04,
9.35987858e-04, 1.05345791e-03, 1.09492564e-03, 8.35579399e-05,
1.14148055e-04, 1.94076325e-05, 1.98835068e-03, 0.00000000e+00,
3.46636825e-05, 0.00000000e+00, 4.22448157e-05, 5.08970627e-05,
4.20918772e-05, 0.00000000e+00, 2.09136391e-05, 8.57227418e-04,
1.42727854e-03, 1.31080525e-03, 6.08285664e-04, 1.76872023e-03,
1.76391355e-04, 3.99174125e-04, 1.72296847e-03, 9.39483150e-05,
0.00000000e+00, 5.16136278e-03, 0.00000000e+00, 8.37763321e-04,
4.07073543e-04, 8.48972967e-05, 8.84252286e-05, 8.66702966e-05,
0.00000000e+00, 5.28565061e-05, 1.31384732e-03, 2.29151312e-04,
```

```
0.00000000e+00, 2.74152611e-04, 0.00000000e+00, 2.64227605e-04,
4.23534452e-04, 0.00000000e+00, 8.73269118e-04, 0.00000000e+00,
1.17245229e-03, 0.00000000e+00, 3.05164732e-04, 8.29001243e-04,
1.17757301e-03, 7.40163259e-05, 0.00000000e+00, 1.54007871e-04,
3.57471964e-04, 2.00483841e-04, 1.29537831e-04, 3.00611248e-05,
1.78192523e-05, 5.82505187e-05, 2.47759765e-05, 1.00592064e-03,
0.00000000e+00, 2.08835905e-03, 3.24398269e-03, 8.97188044e-04,
1.57793643e-04, 1.87354070e-04, 1.54906289e-03, 1.28250709e-03,
4.56133530e-05, 1.45530041e-03, 0.00000000e+00, 1.85868110e-03,
1.67827931e-03, 9.47161322e-04, 0.00000000e+00, 1.43673314e-03,
1.31917225e-03, 2.16679169e-04, 1.58990529e-04, 2.22067598e-04,
1.10726433e-04, 0.00000000e+00, 5.22288657e-03, 1.77661018e-05,
2.50122285e-04, 1.45696529e-04, 4.16125496e-05, 1.37912228e-04,
7.65774274e-05, 7.33622734e-05, 7.94752991e-04, 1.39005906e-03,
1.10540117e-03, 4.45740382e-03, 4.88934466e-05, 1.88231998e-05,
9.44509788e-04, 3.06775137e-05, 1.87781640e-05, 1.54475018e-04,
0.00000000e+00, 0.00000000e+00, 5.58694594e-05, 1.24542403e-05,
0.00000000e+00, 1.47709102e-05, 0.00000000e+00, 2.02069444e-05,
2.43903809e-05, 2.63572015e-05, 2.04015405e-05, 1.54471550e-05,
3.33730258e-05, 4.70476863e-03, 0.00000000e+00, 2.98560367e-05,
5.17544023e-03, 0.00000000e+00, 3.33758777e-05, 1.64233978e-03,
0.00000000e+00, 3.27626459e-04, 3.77167115e-05, 2.17459242e-05,
4.21683580e-03, 1.52106302e-04, 6.18203942e-05, 0.00000000e+00,
0.00000000e+00, 1.17436504e-04, 0.00000000e+00, 1.51624022e-03,
0.00000000e+00, 1.76231607e-05, 2.38384940e-04, 2.48113634e-05,
2.58495969e-05, 6.31423007e-05, 4.55962824e-04, 5.93379356e-04,
7.75327149e-05, 7.15435980e-04, 9.84391775e-04, 0.00000000e+00,
8.21177933e-04, 1.20381219e-03, 3.19108050e-05, 2.05171398e-05,
1.24448477e-04, 2.34937291e-03, 1.24213108e-03, 4.22025384e-04,
0.00000000e+00, 6.89004829e-04, 6.49778525e-04, 0.00000000e+00,
4.33028308e-05, 2.01448225e-03, 3.21501876e-04, 2.41784590e-04,
1.50734847e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
2.91278292e-05, 2.40592599e-04, 6.73926621e-04, 9.33237431e-04,
4.88841258e-04, 3.77738402e-04, 1.10633749e-03, 5.48750803e-04,
3.76796257e-04, 4.76563875e-04, 5.25987448e-05, 9.28403519e-04,
0.00000000e+00, 6.75461423e-04, 0.00000000e+00, 0.00000000e+00,
3.46742030e-04, 2.96140876e-04, 0.00000000e+00, 6.57090235e-04,
4.21002522e-04, 7.43780387e-05, 3.17627974e-04, 0.00000000e+00,
2.08921738e-04, 1.93203030e-04, 3.87096797e-04, 1.62897070e-03,
1.22678785e-04, 1.26051059e-03, 5.75491702e-05, 0.00000000e+00,
2.79991262e-04, 6.01646691e-04, 1.48901922e-05, 1.31179902e-03,
0.00000000e+00, 1.32857750e-03, 1.53149637e-04, 2.74727362e-05,
1.37858168e-03, 1.40101390e-05, 0.00000000e+00, 0.00000000e+00,
6.88203186e-04, 2.14430907e-04, 5.24869787e-04, 2.65752890e-05,
1.23210513e-04, 3.42629909e-04, 3.01935391e-03, 0.00000000e+00,
3.89826850e-04, 5.70838876e-04, 6.07338137e-04, 0.00000000e+00,
5.31668801e-04, 0.00000000e+00, 0.00000000e+00, 4.82067851e-05,
0.00000000e+00, 2.19584397e-05, 5.64263200e-05, 0.00000000e+00,
1.26873135e-03, 0.00000000e+00, 0.00000000e+00, 1.91281378e-05,
3.25636292e-05, 8.70294656e-05, 1.36152984e-05, 2.61836300e-05,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.08207609e-05,
0.00000000e+00, 6.09538295e-05, 2.42551251e-05, 0.00000000e+00,
5.31134615e-03, 0.00000000e+00, 5.19792083e-03, 1.13229784e-04,
6.39185051e-05, 0.00000000e+00, 8.79500680e-04, 1.45632822e-03,
2.20516613e-03, 0.00000000e+00, 1.15891945e-03, 5.16305041e-03,
0.00000000e+00, 0.00000000e+00, 1.16004214e-03, 1.87212200e-03,
1.60797188e-03, 0.00000000e+00, 2.04249631e-05, 3.25729617e-04,
6.21759051e-04, 2.77474695e-04, 6.67593972e-05, 3.07664391e-04,
1.30632949e-03, 1.52397429e-03, 4.63336858e-04, 1.38705414e-03,
0.00000000e+00, 3.47492480e-04, 5.83780995e-04, 1.24087060e-04,
4.79274861e-04, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
```

```
2.12653391e-04, 2.64783889e-04, 1.21553668e-04, 1.86079077e-03,
2.12912336e-04, 0.00000000e+00, 3.02032327e-04, 3.92237945e-04,
2.09853869e-04, 3.55731913e-04, 2.01228387e-04, 0.00000000e+00,
0.00000000e+00, 5.19615850e-04, 0.00000000e+00, 2.60674530e-04,
7.19726967e-04, 9.43503963e-04, 1.10908530e-03, 4.54059449e-04,
4.23719590e-04, 1.55686255e-04, 0.00000000e+00, 2.16841838e-04,
3.87928890e-05, 3.43475592e-05, 3.99821313e-04, 2.40214988e-04,
3.70128551e-04, 5.54735984e-04, 3.53293610e-05, 2.69909050e-02,
9.93485099e-05, 3.73006857e-04, 1.08144629e-03, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 4.89139214e-04, 3.51121772e-04,
2.59776992e-04, 4.61438205e-04, 4.35083744e-05, 0.00000000e+00,
1.21409764e-03, 4.58898568e-04, 1.79807813e-03, 4.15703910e-04,
4.21281935e-04, 2.08359619e-04, 8.00864930e-05, 5.50627582e-04,
7.64104474e-04, 0.00000000e+00, 3.19917178e-04, 3.65101971e-03,
9.01296834e-05, 7.81354166e-05, 1.63152691e-05, 7.29988929e-04,
0.00000000e+00, 8.37061804e-04, 4.09824003e-04, 6.92932388e-04,
1.96318260e-04, 0.00000000e+00, 6.20618576e-04, 4.43295454e-04,
5.80756536e-05, 0.00000000e+00, 3.33363591e-04, 1.21441801e-03,
4.12599384e-04, 1.43850237e-04, 4.63815347e-04, 0.00000000e+00,
1.60934405e-04, 0.00000000e+00, 3.86080360e-04, 0.00000000e+00,
5.19125854e-04, 1.75041919e-04, 9.20106822e-04, 1.31168141e-04,
5.55452371e-04, 6.43605603e-04, 1.09433120e-03, 1.00955263e-04,
0.00000000e+00, 1.10938202e-03, 2.73953090e-04, 0.00000000e+00,
4.12832854e-04, 0.00000000e+00, 0.00000000e+00, 2.03824972e-03,
4.26110317e-04, 0.00000000e+00, 2.00325609e-04, 0.00000000e+00,
7.49755120e-04, 1.68820047e-03, 1.79709395e-03, 0.00000000e+00,
0.00000000e+00, 1.19514248e-03, 0.00000000e+00, 1.13532722e-04,
8.81247438e-05, 0.00000000e+00, 2.30453966e-04, 1.63135503e-04,
3.02160083e-04, 0.00000000e+00, 2.45294695e-05, 0.00000000e+00,
2.24302111e-04, 2.79853537e-04, 4.48829160e-04, 1.75134216e-04,
0.00000000e+00, 1.11990410e-03, 4.55643909e-04, 0.00000000e+00,
4.14843265e-04, 1.35817613e-03, 3.69491831e-04, 4.16559194e-05,
4.54298165e-04, 1.21063331e-03, 0.00000000e+00, 9.88325826e-03,
5.37436399e-03, 1.12742976e-02])
```

In [121]:

```
len(model.feature_importances_)
```

Out[121]:

838

In [122]:

```
feature_importances = pd.DataFrame(model.feature_importances_,
                                   index = train.columns,
                                    columns=['importance']).sort_values('importance', ascer
feature_importances
```

Out[122]:

	importance
Sex_male	0.113025
Sex_female	0.102135
Age	0.092665
Fare	0.090072
Pclass	0.052430
Cabin_B96 B98	0.026991
Parch	0.026341
SibSp	0.023421
Embarked_S	0.011274
Embarked_C	0.009883
Ticket_1601	0.006744
Embarked_Q	0.005374
Ticket_STON/O 2. 3101286	0.005311
Ticket_65306	0.005223
Ticket_STON/O 2. 3101289	0.005198
Ticket_A/5. 10482	0.005175
Ticket_SW/PP 751	0.005163
Ticket_367228	0.005161
Ticket_350043	0.004862
Ticket_244270	0.004829
Ticket_A/5 3540	0.004705
Ticket_315098	0.004679
Ticket_312991	0.004531
Ticket_347077	0.004508
Ticket_345774	0.004497
Ticket_7598	0.004457
Ticket_345779	0.004233
Ticket_347089	0.004225
Ticket_C 17369	0.004217
Ticket_347083	0.004189
...	...
Ticket_SOTON/O.Q. 3101306	0.000000

	importance
Ticket_SOTON/O.Q. 3101305	0.000000
Ticket_PC 17595	0.000000
Ticket_PC 17597	0.000000
Ticket_PC 17599	0.000000
Ticket_250651	0.000000
Ticket_335097	0.000000
Ticket_PC 17603	0.000000
Ticket_250648	0.000000
Ticket_350042	0.000000
Ticket_PC 17610	0.000000
Ticket_350035	0.000000
Ticket_250643	0.000000
Ticket_350034	0.000000
Ticket_248747	0.000000
Ticket_PC 17759	0.000000
Ticket_248733	0.000000
Ticket_S.C./A.4. 23567	0.000000
Ticket_S.P. 3464	0.000000
Ticket_S.W./PP 752	0.000000
Ticket_248706	0.000000
Ticket_244373	0.000000
Ticket_244361	0.000000
Ticket_244358	0.000000
Ticket_SC/PARIS 2149	0.000000
Ticket_244310	0.000000
Ticket_243847	0.000000
Ticket_SCO/W 1585	0.000000
Ticket_336439	0.000000
Ticket_239856	0.000000

838 rows × 1 columns

As seen from the feature_importance dataframe, The Gender of the passengers tends to be the most important factor. Women are more likely to survive the crash, the other factors are Fare, Age, Sex_male.

In [123]:

```
predictions = pd.DataFrame(data=predictions)
```

In [124]:

```
predictions.head()
```

Out[124]:

	0
0	0
1	1
2	0
3	0
4	0

In [125]:

```
model_pred = pd.concat([pass_id,predictions],axis=1)
```

In [126]:

```
model_pred.head()
```

Out[126]:

	PassengerId	0
0	1	0.0
1	2	1.0
2	3	0.0
3	4	0.0
4	5	0.0

In [127]:

```
model_pred = pd.concat([model_pred,label],axis=1)
```

In [128]:

```
model_pred.head()
```

Out[128]:

	PassengerId	0	Survived
0	1	0.0	0
1	2	1.0	1
2	3	0.0	1
3	4	0.0	1
4	5	0.0	0

The NaN Values in the 0 column are because those values were not a part of the training set.