Survival Prediction on Titanic Dataset Using Decision Trees

Importing important libraries

In [1]:
```
# Warnings
import warnings
warnings.filterwarnings('ignore')

# Python imports
import math, time, random, datetime

# Data manipulation
import numpy as np
import pandas as pd

# Visualization
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno

# Pre-processing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn import model_selection, tree, preprocessing, metrics, linear_model
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

Project Outline

- Understanding the nature of data
- Visualization of null values
- Checking relevance of every feature with target feature "Survival"
- Handling missing data using appropriate strategies
- Correlation between the metrices
- Explore interesting themes:
    - Did the wealthy had stronger chances of survival?
    - Which age group had a stronger chances of survival?
    - Did passengers travelling with family had a stronger chances of survival?
- Survival prediction model using Decision Trees Classification algorithm
- Improving the model performance through Hyperparameter Tuning using GridSearchCV to obtain an improved accuracy.

Data

Let's have a look at the data dictionary to understand what information do the features contain.

Features

In [2]:
```
dictionary = pd.read_csv("Dictionary.csv")
dictionary
```

Out[2]:

	Variable	Definition	Key
0	survival	Survival	0 = No, 1 = Yes
1	pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
2	sex	Sex	-
3	Age	Age in years	-
4	sibsp	# of siblings / spouses aboard the Titanic	-
5	parch	# of parents / children aboard the Titanic	-
6	ticket	Ticket number	-
7	fare	Passenger fare	-
8	cabin	Cabin number	-
9	embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Loading the data

In [3]:
```
train = pd.read_csv("train.csv")
```

Understanding data

In [4]:
```
train.head()
```

Out[4]:

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
				Heath (Lily May Peel)								
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [5]:

```
train.shape
```

Out[5]: (891, 12)

In [6]:

```
train.describe()
```

Out[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

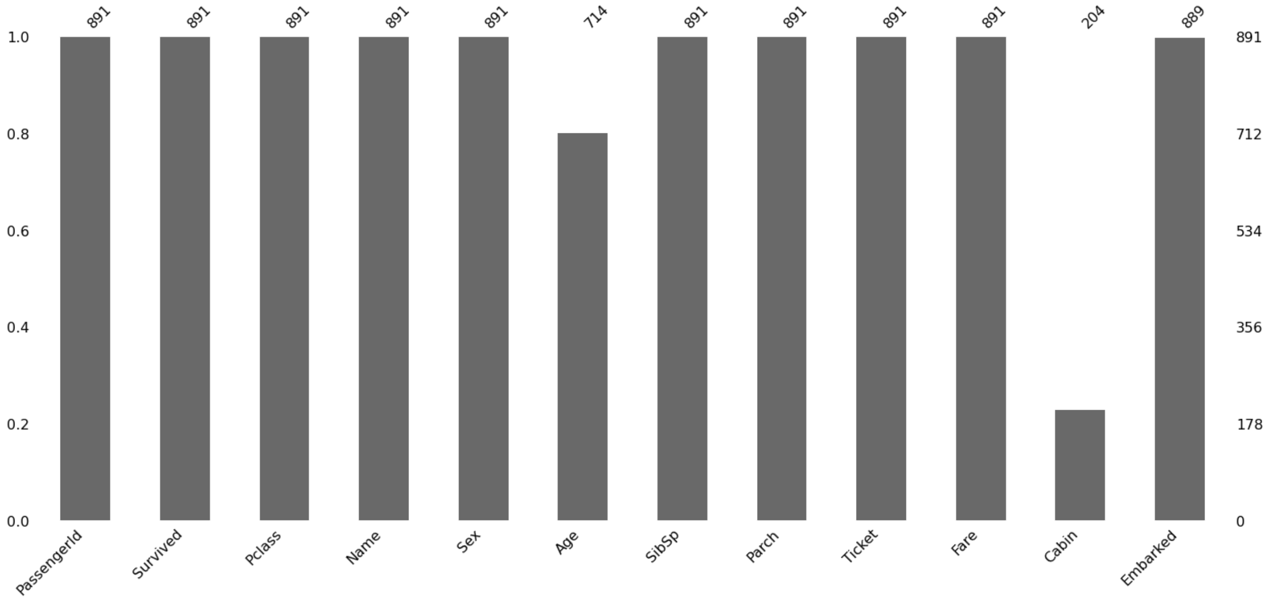
Checking missing values

Using the Missingno library

In [7]:

```
# Bar Chart
msno.bar(train)
```

Out[7]: <AxesSubplot:>



In [8]:
```
train.isnull().sum()
```

Out[8]:
```
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
```

Exploratory Data Analysis

We will make a new dataset called df_bin which will eventually have all the features converted from numerical to categorical and bins of data.

In [9]:
```
train.head()
df_bin = pd.DataFrame()
```

In [10]:
```
# Checking datatypes
train.dtypes
```

Out[10]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
```

```
Cabin           object
Embarked        object
dtype: object
```

Let's explore these features!

In [11]:

```
train.head()
```

Out[11]:

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

Target Feature: Survived

Description: Whether a person survived or not

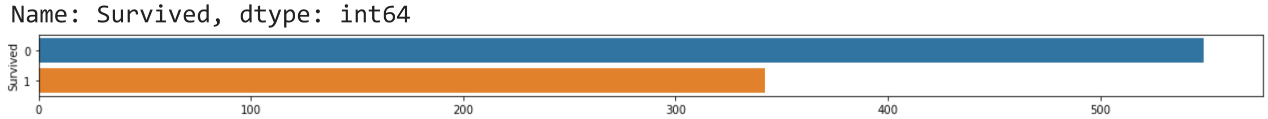
0: Did not survive; **1:** Survived

How many people survived?

In [12]:

```
fig = plt.figure(figsize = (20,1))
sns.countplot(y = "Survived", data = train)
print(train.Survived.value_counts())
```

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



In [13]:
```
df_bin['Survived'] = train["Survived"]
```

Feature: Pclass

Description: Ticket class of the passenger.

Key: 1 = 1st, 2 = 2nd, 3 = 3rd

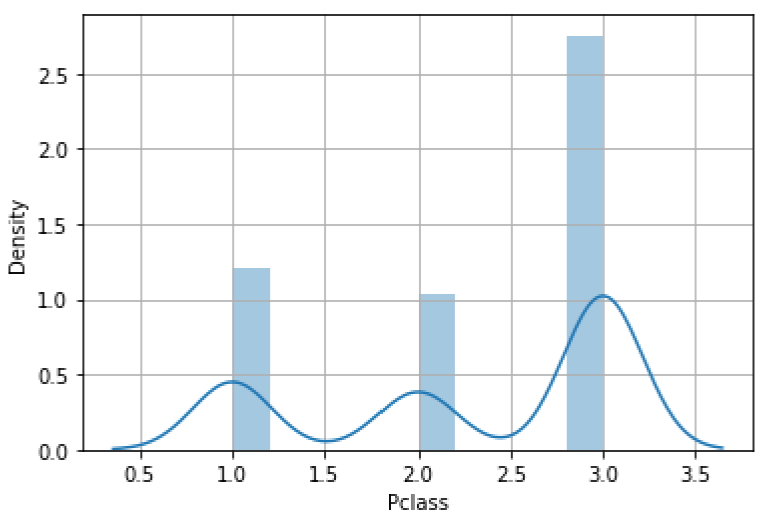
Relavance of the feature:

In [14]:
```
train.groupby(["Pclass"])["Survived"].mean()
```

Out[14]:
```
Pclass
1    0.629630
2    0.472826
3    0.242363
Name: Survived, dtype: float64
```

We can see that as the classes change, the percentage people surviving in going dowm. Therefore, this feature has a mathematical correlation (negative in this case) with the target feature "Survived".

In [15]:
```
# Pclass
g = sns.distplot(train.Pclass)
g.grid()
```



Passangers travelling in Pclass 3 is the major chunk, as evident from the probability distribution function above.

In [16]:
```
print("Null values:", train.Pclass.isnull().sum())
```

```
Null values: 0
```

In [17]:
```
df_bin['Pclass'] = train['Pclass']
```

In [18]:
```
train.head()
```

Out[18]:

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

NOTE: Although the feature "Pclass" is numerical (1,2 and 3) but they are categories. Therefore, this is a categorical feature

In [19]:
```
train.Pclass.dtype
```

Out[19]: dtype('int64')

In [20]:
```
train.Pclass = train.Pclass.astype('object')
```

In [21]:
```
train.Pclass.dtype
```

Out[21]: dtype('O')

Feature: Name

Relavance of the feature:

Logically, the name of a person wouldn't have an effect on the chances of their survival, but the important information in this feature is the salutations (Mr., Mrs. etc.). Therefore, we will keep this feature for further analysis.

Checking for duplicate names:

In [22]:
```
train.Name.duplicated().sum()
```

Out[22]:
```
0
```

Therefore, all names are unique.

In [23]:
```
train.Name.head(10)
```

Out[23]:
```
0                               Braund, Mr. Owen Harris
1     Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                Heikkinen, Miss. Laina
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                              Allen, Mr. William Henry
5                                      Moran, Mr. James
6                               McCarthy, Mr. Timothy J
7                        Palsson, Master. Gosta Leonard
8     Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
9                   Nasser, Mrs. Nicholas (Adele Achem)
Name: Name, dtype: object
```

Let's make columns with salutations:

- Mr.
- Mrs.
- Miss.
- Master.
- Dr.

In [24]:
```
train.columns
```

Out[24]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [25]:
```
train['Title'] = ''
for i in range(len(train)):
    if ("Mr." in train['Name'][i]):
        train['Title'][i] = 'Mr'
    elif ("Mrs." in train['Name'][i]):
        train['Title'][i] = "Mrs"
    elif ("Miss." in train['Name'][i]):
        train['Title'][i] = "Miss"
    elif ("Master." in train['Name'][i]):
        train['Title'][i] = "Master"
    elif ("Dr." in train['Name'][i]):
        train['Title'][i] = "Dr"
```

In [26]:
```
train.head()
```

Out[26]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
0	1	0	3	Braund, Mr. Owen	male	22.0	1	0	A/5 21171	7.2500	NaN	

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

As we can see above, the column "Title" has been added to the dataframe with respective salutations. This will help us categorize the names further as we proceed.

Let's check if every name has one of these salutations or not.

In [27]:

```
L = []
for i in range(len(train)):
    if ("Mr." in train['Name'][i] or
        "Mrs." in train['Name'][i] or
        "Miss." in train['Name'][i] or
        "Master." in train['Name'][i] or
        "Dr." in train['Name'][i]):
        continue
    else:
        L.append(train['Name'][i])
print ("Total Names with Other Titles:",len(L),"\n\nNames with other titles:")
L
```

```
Total Names with Other Titles: 20

Names with other titles:
```

Out[27]:

```
['Uruchurtu, Don. Manuel E',
 'Byles, Rev. Thomas Roussel Davids',
 'Bateman, Rev. Robert James',
 'Carter, Rev. Ernest Courtenay',
 'Aubart, Mme. Leontine Pauline',
 'Reynaldo, Ms. Encarnacion',
 'Peuchen, Major. Arthur Godfrey',
 'Butt, Major. Archibald Willingham',
 'Duff Gordon, Lady. (Lucille Christiana Sutherland) ("Mrs Morgan")',
 'Duff Gordon, Sir. Cosmo Edmund ("Mr Morgan")',
 'Kirkland, Rev. Charles Leonard',
 'Sagesser, Mlle. Emma',
 'Simonius-Blumer, Col. Oberst Alfons',
```

```
 'Weir, Col. John',
 'Mayne, Mlle. Berthe Antonine ("Mrs de Villiers")',
 'Crosby, Capt. Edward Gifford',
 'Rothes, the Countess. of (Lucy Noel Martha Dyer-Edwards)',
 'Reuchlin, Jonkheer. John George',
 'Harper, Rev. John',
 'Montvila, Rev. Juozas']
```

Following observations have been found:

- **Mme.** is a *french title* equivalent to **"Mrs."**. We will change the name 'Aubart, Mme. Leontine Pauline' to 'Aubart, Mrs. Leontine Pauline'
- **Ms.** is an abbreviation for **"Miss."**. Therefore we will change the name 'Reynaldo, Ms. Encarnacion' to 'Reynaldo, Miss. Encarnacion'
- **Mlle.** is a *french title* equivalent to **"Miss."**. We will change the names:
    - 'Sagesser, Mlle. Emma' to 'Sagesser, Miss. Emma'
    - 'Mayne, Mlle. Berthe Antonine ("Mrs de Villiers")' to Mayne, Miss. Berthe Antonine ("Mrs de Villiers")'

In [28]:

```
train["Name"] = train["Name"].str.replace("Ms.","Miss.", regex = True)
train["Name"] = train["Name"].str.replace("Mme.","Mrs.", regex = True)
train["Name"] = train["Name"].str.replace("Mlle.","Miss.", regex = True)
```

In [29]:

```
L = []
for i in range(len(train)):
    if ("Mr." in train['Name'][i] or
        "Mrs." in train['Name'][i] or
        "Miss." in train['Name'][i] or
        "Master." in train['Name'][i] or
        "Dr." in train['Name'][i]):
        continue
    else:
        L.append(train['Name'][i])
print ("Total Names with Other Titles:",len(L),"\n\nNames with other titles:")
L
```

```
Total Names with Other Titles: 16

Names with other titles:
```

Out[29]:

```
['Uruchurtu, Don. Manuel E',
 'Byles, Rev. Thomas Roussel Davids',
 'Bateman, Rev. Robert James',
 'Carter, Rev. Ernest Courtenay',
 'Peuchen, Major. Arthur Godfrey',
 'Butt, Major. Archibald Willingham',
 'Duff Gordon, Lady. (Lucille Christiana Sutherland) ("Mrs Morgan")',
 'Duff Gordon, Sir. Cosmo Edmund ("Mr Morgan")',
 'Kirkland, Rev. Charles Leonard',
 'Simonius-Blumer, Col. Oberst Alfons',
 'Weir, Col. John',
 'Crosby, Capt. Edward Gifford',
 'Rothes, the Countess. of (Lucy Noel Martha Dyer-Edwards)',
 'Reuchlin, Jonkheer. John George',
 'Harper, Rev. John',
 'Montvila, Rev. Juozas']
```

Let's put these as **"Other_Titles"** in the title column.

In [30]:
```
for i in range(len(train)):
    if ("Mr." in train['Name'][i] or
        "Mrs." in train['Name'][i] or
        "Miss." in train['Name'][i] or
        "Master." in train['Name'][i] or
        "Dr." in train['Name'][i]):
        continue
    else:
        train['Title'][i] = "Other"
```

In [31]:
```
train.head()
```

Out[31]:

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

In [32]:
```
train.Title.describe()
```

Out[32]:
```
count     891
unique      7
top        Mr
freq      517
Name: Title, dtype: object
```

In [33]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
```

```
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    object 
 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 
 12  Title        891 non-null    object 
dtypes: float64(2), int64(4), object(7)
memory usage: 90.6+ KB
```

In [34]:
```
df_bin["Title"] = train["Title"]
```

Feature: Sex

Relavance of the feature:

In [35]:
```
train.groupby(["Sex"])["Survived"].mean()
```

Out[35]:
```
Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

There is a 74% chance that a female would survive and an 18% chance that a male survives.

Therefore, the feature "Sex" has a huge role to play in survival chances.

In [36]:
```
train["Sex"].isnull().sum()
```

Out[36]:
```
0
```

In [37]:
```
train["Sex"].describe()
```

Out[37]:
```
count      891
unique       2
top       male
freq       577
Name: Sex, dtype: object
```

Let's change the "male" and "female" to "1" and "0" respectively, for analysis purposes.

In [38]:
```
for i in range(len(train)):
    if (train["Sex"][i] == 'male'):
        train["Sex"][i] = train["Sex"][i].replace("male","1")
    else:
        train["Sex"][i] = train["Sex"][i].replace("female","0")
```

In [39]:
```
train.Sex.unique()
```

Out[39]: array(['1', '0'], dtype=object)

In [40]:
```
train.head()
```

Out[40]:

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

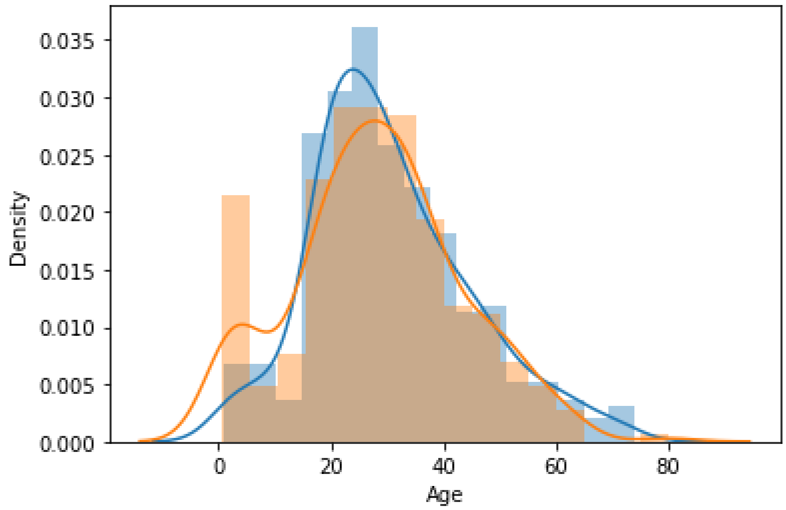
In [41]:
```
df_bin['Sex'] = train["Sex"]
```

Feature: Age

Relevance of the feature:

In [42]:
```
sns.distplot(train["Age"][train["Survived"]==0])
sns.distplot(train["Age"][train["Survived"]==1])
```

Out[42]: <AxesSubplot:xlabel='Age', ylabel='Density'>



Note: Blue graph is for **"Survived" = 0** and **Orange** is for **"Survived" = 1**

Following is evident from the graphs:

- The feature seems to be following a normal distribution.
- Most of the kids (age 0 to about 15 years) seem to have survived.
- More passengers between the age of 20 and 30 years to have died rather than survived.

The pattern we can see here is that passengers with very less age have survived more than died, and opposite is the case for passengers with ages between 20 and 30 years.

Therefore, age seems to have an impact on the survival probability.

In [43]:

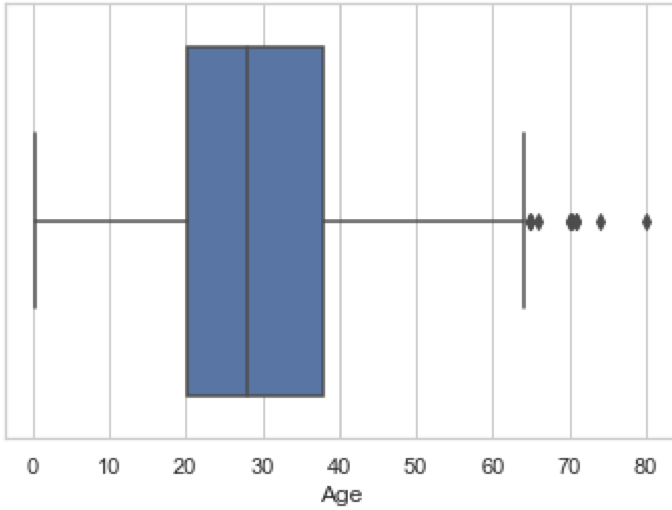
```
train.Age.describe()
```

Out[43]:

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

In [44]:

```
sns.set_theme(style="whitegrid")
sns.boxplot(train["Age"], width = 0.8)
plt.show()
```



In [45]:
```
train[train["Age"]>63].count()
```

Out[45]:
```
PassengerId    13
Survived       13
Pclass         13
Name           13
Sex            13
Age            13
SibSp          13
Parch          13
Ticket         13
Fare           13
Cabin           6
Embarked       13
Title          13
dtype: int64
```

In [46]:
```
train["Age"].isnull().sum()
```

Out[46]:
```
177
```

Since there are many null values in the column, let's impute them.

In order to have a finer estimation, we'll find out the **mean age according to the title** and impute values accordingly.

In [47]:
```
mr_age = []
for i in range(len(train)):
    if ("Mr." in train["Name"][i]):
        mr_age.append(train["Age"][i])

mrs_age = []
for i in range(len(train)):
    if ("Mrs." in train["Name"][i]):
        mrs_age.append(train["Age"][i])

miss_age = []
for i in range(len(train)):
    if ("Miss." in train["Name"][i]):
        miss_age.append(train["Age"][i])
```

```
master_age = []
for i in range(len(train)):
    if ("Master." in train["Name"][i]):
        master_age.append(train["Age"][i])

other_age = []
for i in range(len(train)):
    if ("Mr." in train['Name'][i] or
        "Mrs." in train['Name'][i] or
        "Miss." in train['Name'][i] or
        "Master." in train['Name'][i]):
        continue
    else:
        other_age.append(train['Age'][i])
```

In [48]:

```
mr_age_df = pd.Series(mr_age)
mrs_age_df = pd.Series(mrs_age)
miss_age_df = pd.Series(miss_age)
master_age_df = pd.Series(master_age)
other_age_df = pd.Series(other_age)
```

In [49]:

```
mr_age_df_mean = round((mr_age_df.mean()),1)
mrs_age_df_mean = round((mrs_age_df.mean()),1)
miss_age_df_mean = round((miss_age_df.mean()),1)
master_age_df_mean = round((master_age_df.mean()),1)
other_age_df_mean = round((other_age_df.mean()),1)
print("Mr. mean age:", mr_age_df_mean, "\nMrs. mean age:", mrs_age_df_mean,"\nMiss. mea
      "\nMaster. mean age:",master_age_df_mean,"\nOther mean age:",other_age_df_mean)
```

```
Mr. mean age: 32.4
Mrs. mean age: 35.8
Miss. mean age: 21.8
Master. mean age: 4.6
Other mean age: 45.5
```

In [50]:

```
train.Age.describe()
```

Out[50]:

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

In [51]:

```
train.head(20)
```

Out[51]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John	0	38.0	1	0	PC 17599	71.2833	C85	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
				Bradley (Florence Briggs Th...								
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	
5	6	0	3	Moran, Mr. James	1	NaN	0	0	330877	8.4583	NaN	
6	7	0	1	McCarthy, Mr. Timothy J	1	54.0	0	0	17463	51.8625	E46	
7	8	0	3	Palsson, Master. Gosta Leonard	1	2.0	3	1	349909	21.0750	NaN	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	0	27.0	0	2	347742	11.1333	NaN	
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	0	14.0	1	0	237736	30.0708	NaN	
10	11	1	3	Sandstrom, Miss. Marguerite Rut	0	4.0	1	1	PP 9549	16.7000	G6	
11	12	1	1	Bonnell, Miss. Elizabeth	0	58.0	0	0	113783	26.5500	C103	
12	13	0	3	Saundercock, Mr. William Henry	1	20.0	0	0	A/5. 2151	8.0500	NaN	
13	14	0	3	Andersson, Mr. Anders Johan	1	39.0	1	5	347082	31.2750	NaN	
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	0	14.0	0	0	350406	7.8542	NaN	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	0	55.0	0	0	248706	16.0000	NaN	
16	17	0	3	Rice, Master. Eugene	1	2.0	4	1	382652	29.1250	NaN	
17	18	1	2	Williams, Mr. Charles Eugene	1	NaN	0	0	244373	13.0000	NaN	
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	0	31.0	1	0	345763	18.0000	NaN	
19	20	1	3	Masselmani, Mrs. Fatima	0	NaN	0	0	2649	7.2250	NaN	

In [52]:

```
Age_null_list = list(train["Age"].isnull())

a = 0
b = 0
c = 0
d = 0
e = 0

for i in range(len(train)):
    if (("Mr." in train["Name"][i]) and (Age_null_list[i] is True)):
        a = a + 1
        train["Age"][i] = mr_age_df_mean
    elif (("Mrs." in train["Name"][i]) and (Age_null_list[i] is True)):
        b = b + 1
        train["Age"][i] = mrs_age_df_mean
    elif (("Miss." in train["Name"][i]) and (Age_null_list[i] is True)):
        c = c + 1
        train["Age"][i] = miss_age_df_mean
    elif (("Master." in train["Name"][i]) and (Age_null_list[i] is True)):
        d = d + 1
        train["Age"][i] = master_age_df_mean
    elif (Age_null_list[i] is True):
        e = e + 1
        train["Age"][i] = other_age_df_mean

print("Mr. null age total:",a,"\nMrs. null age total:",b,"\nMiss. null age total:",c,
      "\nMaster. null age total:",d,"\nOther null age total:",e)
```

```
Mr. null age total: 119
Mrs. null age total: 17
Miss. null age total: 36
Master. null age total: 4
Other null age total: 1
```

As there are many null values in this column, let us impute the values.

In [53]:

```
train.head(20)
```

Out[53]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	
5	6	0	3	Moran, Mr. James	1	32.4	0	0	330877	8.4583	NaN	
6	7	0	1	McCarthy, Mr. Timothy J	1	54.0	0	0	17463	51.8625	E46	
7	8	0	3	Palsson, Master. Gosta Leonard	1	2.0	3	1	349909	21.0750	NaN	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	0	27.0	0	2	347742	11.1333	NaN	
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	0	14.0	1	0	237736	30.0708	NaN	
10	11	1	3	Sandstrom, Miss. Marguerite Rut	0	4.0	1	1	PP 9549	16.7000	G6	
11	12	1	1	Bonnell, Miss. Elizabeth	0	58.0	0	0	113783	26.5500	C103	
12	13	0	3	Saundercock, Mr. William Henry	1	20.0	0	0	A/5. 2151	8.0500	NaN	
13	14	0	3	Andersson, Mr. Anders Johan	1	39.0	1	5	347082	31.2750	NaN	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	0	14.0	0	0	350406	7.8542	NaN	
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	0	55.0	0	0	248706	16.0000	NaN	
16	17	0	3	Rice, Master. Eugene	1	2.0	4	1	382652	29.1250	NaN	
17	18	1	2	Williams, Mr. Charles Eugene	1	32.4	0	0	244373	13.0000	NaN	
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	0	31.0	1	0	345763	18.0000	NaN	
19	20	1	3	Masselmani, Mrs. Fatima	0	35.8	0	0	2649	7.2250	NaN	

In [54]:

```
Age_null_list = list(train["Age"].isnull())

a = 0
b = 0
c = 0
d = 0
e = 0

for i in range(len(train)):
    if (("Mr." in train["Name"][i]) and (Age_null_list[i] is True)):
        a = a + 1
    elif (("Mrs." in train["Name"][i]) and (Age_null_list[i] is True)):
        b = b + 1
    elif (("Miss." in train["Name"][i]) and (Age_null_list[i] is True)):
        c = c + 1
    elif (("Master." in train["Name"][i]) and (Age_null_list[i] is True)):
        d = d + 1
    elif (Age_null_list[i] is True):
        e = e + 1

print("Mr. null age total:",a,"\nMrs. null age total:",b,"\nMiss. null age total:",c,
      "\nMaster. null age total:",d,"\nOther null age total:",e)
```

```
Mr. null age total: 0
Mrs. null age total: 0
Miss. null age total: 0
Master. null age total: 0
Other null age total: 0
```

No null values now!

In [55]:

```
train.Age.describe()
```

Out[55]:
```
count    891.000000
mean      29.762144
std       13.280454
min        0.420000
25%       21.800000
50%       30.000000
75%       35.800000
max       80.000000
Name: Age, dtype: float64
```

In [56]:
```
df_bin['Age'] = pd.cut(train["Age"],[0,18,30,50,80])
```

In [57]:
```
df_bin.head(30)
```

Out[57]:

	Survived	Pclass	Title	Sex	Age
0	0	3	Mr	1	(18, 30]
1	1	1	Mrs	0	(30, 50]
2	1	3	Miss	0	(18, 30]
3	1	1	Mrs	0	(30, 50]
4	0	3	Mr	1	(30, 50]
5	0	3	Mr	1	(30, 50]
6	0	1	Mr	1	(50, 80]
7	0	3	Master	1	(0, 18]
8	1	3	Mrs	0	(18, 30]
9	1	2	Mrs	0	(0, 18]
10	1	3	Miss	0	(0, 18]
11	1	1	Miss	0	(50, 80]
12	0	3	Mr	1	(18, 30]
13	0	3	Mr	1	(30, 50]
14	0	3	Miss	0	(0, 18]
15	1	2	Mrs	0	(50, 80]
16	0	3	Master	1	(0, 18]
17	1	2	Mr	1	(30, 50]
18	0	3	Mrs	0	(30, 50]
19	1	3	Mrs	0	(30, 50]
20	0	2	Mr	1	(30, 50]
21	1	2	Mr	1	(30, 50]
22	1	3	Miss	0	(0, 18]
23	1	1	Mr	1	(18, 30]

	Survived	Pclass	Title	Sex	Age
24	0	3	Miss	0	(0, 18]
25	1	3	Mrs	0	(30, 50]
26	0	3	Mr	1	(30, 50]
27	0	1	Mr	1	(18, 30]
28	1	3	Miss	0	(18, 30]
29	0	3	Mr	1	(30, 50]

Feature: SibSp

Relevance of the feature:

In [58]:
```
train.groupby(["SibSp"])["Survived"].mean()
```

Out[58]:
```
SibSp
0    0.345395
1    0.535885
2    0.464286
3    0.250000
4    0.166667
5    0.000000
8    0.000000
Name: Survived, dtype: float64
```

As the number of siblings/spouse increases, the chances of survival decreases. Therefore, these is a mathematical correlation between "SibSp" and "Survival".

In [59]:
```
train["SibSp"].isnull().sum()
```

Out[59]:
```
0
```

In [60]:
```
train["SibSp"].dtype
```

Out[60]:
```
dtype('int64')
```

In [61]:
```
train["SibSp"].value_counts()
```

Out[61]:
```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

Feature: Parch

Relevance of the feature:

In [62]:
```
train.groupby(["Parch"])["Survived"].mean()
```

Out[62]:
```
Parch
0    0.343658
1    0.550847
2    0.500000
3    0.600000
4    0.000000
5    0.200000
6    0.000000
Name: Survived, dtype: float64
```

For the feature "Parch", it is again evident that passengers with higher number of parents/children have lesser chances of survival. Therefore, there is a mathematical correlation between "Parch" and "Survival".

In [63]:
```
train.Parch.isnull().sum()
```

Out[63]:
```
0
```

In [64]:
```
train.Parch.value_counts()
```

Out[64]:
```
0    678
1    118
2     80
3      5
5      5
4      4
6      1
Name: Parch, dtype: int64
```

In [65]:
```
train.Parch.dtype
```

Out[65]:
```
dtype('int64')
```

Let's check some data sanity!

Passengers with **"Miss."** and **"Master."** salutations should not have "Parch" feature with value greater than 2.

In [66]:
```
n = 0
m = 0

for i in range(len(train)):
    if (("Miss." in train["Name"][i]) and (train["Parch"][i] > 2)):
        n = n + 1

for i in range(len(train)):
    if (("Master." in train["Name"][i]) and (train["Parch"][i] > 2)):
        m = m + 1

print('\"Miss." with \"Parch" > 2:',n,'\n\"Master." with \"Parch" > 2:',m)
```

```
"Miss." with "Parch" > 2: 0
"Master." with "Parch" > 2: 0
```

Merging "SibSp" and "Parch":

To simplify the model, let us merge the two feature into one: "Family".

In [67]:
```
train["Family"] = train["SibSp"] + train["Parch"]
```

In [68]:
```
train.head()
```

Out[68]:

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

In [69]:
```
train.groupby(["Family"])["Survived"].mean()
```

Out[69]:
```
Family
0     0.303538
1     0.552795
2     0.578431
3     0.724138
4     0.200000
5     0.136364
6     0.333333
7     0.000000
10    0.000000
Name: Survived, dtype: float64
```

Following conclusions can be made from the results above:

- If a person is travelling alone, there's a 30% chance of survival.

- People with 1, 2 or 3 family members have comparitively higher survival chances.
- People with 3, 4 or 5 family members have declining survival chances.
- People travelling with 7 or 10 family members have 0 survival chances.

As we can see categories here, let's make a new column called **"Family_Size"** and categorize the family into:

- Alone
- Small
- Medium
- Large

In [70]:

```
def calculate (number):
    if number == 0:
        return "Alone"
    elif number > 0 and number < 4:
        return "Small"
    elif number > 3 and number < 7:
        return "Medium"
    else:
        return "Large"

train["Family_Size"] = train["Family"].apply(calculate)
```

In [71]:

```
train.head()
```

Out[71]:

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

In [72]:
```
df_bin['Family_Size'] = train['Family_Size']
```

In [73]:
```
df_bin.head()
```

Out[73]:

	Survived	Pclass	Title	Sex	Age	Family_Size
0	0	3	Mr	1	(18, 30]	Small
1	1	1	Mrs	0	(30, 50]	Small
2	1	3	Miss	0	(18, 30]	Alone
3	1	1	Mrs	0	(30, 50]	Small
4	0	3	Mr	1	(30, 50]	Alone

Feature: Ticket

Relevance of the feature:

To check the relevance of the feature, let's categorize the tickets to get the best results.

In [74]:
```
print("Null values:", train.Ticket.isnull().sum())
print("Data type:",train.Ticket.dtype)
```

```
Null values: 0
Data type: object
```

In [75]:
```
train.Ticket.nunique()
```

Out[75]:
```
681
```

In [76]:
```
train.Ticket.unique()
```

Out[76]:
```
array(['A/5 21171', 'PC 17599', 'STON/O2. 3101282', '113803', '373450',
       '330877', '17463', '349909', '347742', '237736', 'PP 9549',
       '113783', 'A/5. 2151', '347082', '350406', '248706', '382652',
       '244373', '345763', '2649', '239865', '248698', '330923', '113788',
       '347077', '2631', '19950', '330959', '349216', 'PC 17601',
       'PC 17569', '335677', 'C.A. 24579', 'PC 17604', '113789', '2677',
       'A./5. 2152', '345764', '2651', '7546', '11668', '349253',
       'SC/Paris 2123', '330958', 'S.C./A.4. 23567', '370371', '14311',
       '2662', '349237', '3101295', 'A/4. 39886', 'PC 17572', '2926',
       '113509', '19947', 'C.A. 31026', '2697', 'C.A. 34651', 'CA 2144',
       '2669', '113572', '36973', '347088', 'PC 17605', '2661',
       'C.A. 29395', 'S.P. 3464', '3101281', '315151', 'C.A. 33111',
       'S.O.C. 14879', '2680', '1601', '348123', '349208', '374746',
       '248738', '364516', '345767', '345779', '330932', '113059',
       'SO/C 14885', '3101278', 'W./C. 6608', 'SOTON/OQ 392086', '343275',
       '343276', '347466', 'W.E.P. 5734', 'C.A. 2315', '364500', '374910',
       'PC 17754', 'PC 17759', '231919', '244367', '349245', '349215',
       '35281', '7540', '3101276', '349207', '343120', '312991', '349249',
       '371110', '110465', '2665', '324669', '4136', '2627',
       'STON/O 2. 3101294', '370369', 'PC 17558', 'A4. 54510', '27267',
       '370372', 'C 17369', '2668', '347061', '349241',
```

```
'SOTON/O.Q. 3101307', 'A/5. 3337', '228414', 'C.A. 29178',
'SC/PARIS 2133', '11752', '7534', 'PC 17593', '2678', '347081',
'STON/O2. 3101279', '365222', '231945', 'C.A. 33112', '350043',
'230080', '244310', 'S.O.P. 1166', '113776', 'A.5. 11206',
'A/5. 851', 'Fa 265302', 'PC 17597', '35851', 'SOTON/OQ 392090',
'315037', 'CA. 2343', '371362', 'C.A. 33595', '347068', '315093',
'363291', '113505', 'PC 17318', '111240', 'STON/O 2. 3101280',
'17764', '350404', '4133', 'PC 17595', '250653', 'LINE',
'SC/PARIS 2131', '230136', '315153', '113767', '370365', '111428',
'364849', '349247', '234604', '28424', '350046', 'PC 17610',
'368703', '4579', '370370', '248747', '345770', '3101264', '2628',
'A/5 3540', '347054', '2699', '367231', '112277',
'SOTON/O.Q. 3101311', 'F.C.C. 13528', 'A/5 21174', '250646',
'367229', '35273', 'STON/O2. 3101283', '243847', '11813',
'W/C 14208', 'SOTON/OQ 392089', '220367', '21440', '349234',
'19943', 'PP 4348', 'SW/PP 751', 'A/5 21173', '236171', '347067',
'237442', 'C.A. 29566', 'W./C. 6609', '26707', 'C.A. 31921',
'28665', 'SCO/W 1585', '367230', 'W./C. 14263',
'STON/O 2. 3101275', '2694', '19928', '347071', '250649', '11751',
'244252', '362316', '113514', 'A/5. 3336', '370129', '2650',
'PC 17585', '110152', 'PC 17755', '230433', '384461', '110413',
'112059', '382649', 'C.A. 17248', '347083', 'PC 17582', 'PC 17760',
'113798', '250644', 'PC 17596', '370375', '13502', '347073',
'239853', 'C.A. 2673', '336439', '347464', '345778', 'A/5. 10482',
'113056', '349239', '345774', '349206', '237798', '370373',
'19877', '11967', 'SC/Paris 2163', '349236', '349233', 'PC 17612',
'2693', '113781', '19988', '9234', '367226', '226593', 'A/5 2466',
'17421', 'PC 17758', 'P/PP 3381', 'PC 17485', '11767', 'PC 17608',
'250651', '349243', 'F.C.C. 13529', '347470', '29011', '36928',
'16966', 'A/5 21172', '349219', '234818', '345364', '28551',
'111361', '113043', 'PC 17611', '349225', '7598', '113784',
'248740', '244361', '229236', '248733', '31418', '386525',
'C.A. 37671', '315088', '7267', '113510', '2695', '2647', '345783',
'237671', '330931', '330980', 'SC/PARIS 2167', '2691',
'SOTON/O.Q. 3101310', 'C 7076', '110813', '2626', '14313',
'PC 17477', '11765', '3101267', '323951', 'C 7077', '113503',
'2648', '347069', 'PC 17757', '2653', 'STON/O 2. 3101293',
'349227', '27849', '367655', 'SC 1748', '113760', '350034',
'3101277', '350052', '350407', '28403', '244278', '240929',
'STON/O 2. 3101289', '341826', '4137', '315096', '28664', '347064',
'29106', '312992', '349222', '394140', 'STON/O 2. 3101269',
'343095', '28220', '250652', '28228', '345773', '349254',
'A/5. 13032', '315082', '347080', 'A/4. 34244', '2003', '250655',
'364851', 'SOTON/O.Q. 392078', '110564', '376564', 'SC/AH 3085',
'STON/O 2. 3101274', '13507', 'C.A. 18723', '345769', '347076',
'230434', '65306', '33638', '113794', '2666', '113786', '65303',
'113051', '17453', 'A/5 2817', '349240', '13509', '17464',
'F.C.C. 13531', '371060', '19952', '364506', '111320', '234360',
'A/S 2816', 'SOTON/O.Q. 3101306', '113792', '36209', '323592',
'315089', 'SC/AH Basle 541', '7553', '31027', '3460', '350060',
'3101298', '239854', 'A/5 3594', '4134', '11771', 'A.5. 18509',
'65304', 'SOTON/OQ 3101317', '113787', 'PC 17609', 'A/4 45380',
'36947', 'C.A. 6212', '350035', '315086', '364846', '330909',
'4135', '26360', '111427', 'C 4001', '382651', 'SOTON/OQ 3101316',
'PC 17473', 'PC 17603', '349209', '36967', 'C.A. 34260', '226875',
'349242', '12749', '349252', '2624', '2700', '367232',
'W./C. 14258', 'PC 17483', '3101296', '29104', '2641', '2690',
'315084', '113050', 'PC 17761', '364498', '13568', 'WE/P 5735',
'2908', '693', 'SC/PARIS 2146', '244358', '330979', '2620',
'347085', '113807', '11755', '345572', '372622', '349251',
'218629', 'SOTON/OQ 392082', 'SOTON/O.Q. 392087', 'A/4 48871',
'349205', '2686', '350417', 'S.W./PP 752', '11769', 'PC 17474',
'14312', 'A/4. 20589', '358585', '243880', '2689',
'STON/O 2. 3101286', '237789', '13049', '3411', '237565', '13567',
'14973', 'A./5. 3235', 'STON/O 2. 3101273', 'A/5 3902', '364848',
```

```
'SC/AH 29037', '248727', '2664', '349214', '113796', '364511',
'111426', '349910', '349246', '113804', 'SOTON/O.Q. 3101305',
'370377', '364512', '220845', '31028', '2659', '11753', '350029',
'54636', '36963', '219533', '349224', '334912', '27042', '347743',
'13214', '112052', '237668', 'STON/O 2. 3101292', '350050',
'349231', '13213', 'S.O./P.P. 751', 'CA. 2314', '349221', '8475',
'330919', '365226', '349223', '29751', '2623', '5727', '349210',
'STON/O 2. 3101285', '234686', '312993', 'A/5 3536', '19996',
'29750', 'F.C. 12750', 'C.A. 24580', '244270', '239856', '349912',
'342826', '4138', '330935', '6563', '349228', '350036', '24160',
'17474', '349256', '2672', '113800', '248731', '363592', '35852',
'348121', 'PC 17475', '36864', '350025', '223596', 'PC 17476',
'PC 17482', '113028', '7545', '250647', '348124', '34218', '36568',
'347062', '350048', '12233', '250643', '113806', '315094', '36866',
'236853', 'STON/O2. 3101271', '239855', '28425', '233639',
'349201', '349218', '16988', '376566', 'STON/O 2. 3101288',
'250648', '113773', '335097', '29103', '392096', '345780',
'349204', '350042', '29108', '363294', 'SOTON/O2 3101272', '2663',
'347074', '112379', '364850', '8471', '345781', '350047',
'S.O./P.P. 3', '2674', '29105', '347078', '383121', '36865',
'2687', '113501', 'W./C. 6607', 'SOTON/O.Q. 3101312', '374887',
'3101265', '12460', 'PC 17600', '349203', '28213', '17465',
'349244', '2685', '2625', '347089', '347063', '112050', '347087',
'248723', '3474', '28206', '364499', '112058', 'STON/O2. 3101290',
'S.C./PARIS 2079', 'C 7075', '315098', '19972', '368323', '367228',
'2671', '347468', '2223', 'PC 17756', '315097', '392092', '11774',
'SOTON/O2 3101287', '2683', '315090', 'C.A. 5547', '349213',
'347060', 'PC 17592', '392091', '113055', '2629', '350026',
'28134', '17466', '233866', '236852', 'SC/PARIS 2149', 'PC 17590',
'345777', '349248', '695', '345765', '2667', '349212', '349217',
'349257', '7552', 'C.A./SOTON 34068', 'SOTON/OQ 392076', '211536',
'112053', '111369', '370376'], dtype=object)
```

We see there are majorly 2 types of tickets.

- Alphanumeric
- Numeric

In [77]:

```python
# Alphanumeric tickets
L = []
for i in range(len(train)):
    if (train['Ticket'][i].isdigit() == False):
        L.append(train['Ticket'][i])

Li = pd.Series(L)
print("Total:",Li.nunique(),'\n',Li.unique())
```

```
Total: 167
 ['A/5 21171' 'PC 17599' 'STON/O2. 3101282' 'PP 9549' 'A/5. 2151'
 'PC 17601' 'PC 17569' 'C.A. 24579' 'PC 17604' 'A./5. 2152'
 'SC/Paris 2123' 'S.C./A.4. 23567' 'A/4. 39886' 'PC 17572' 'C.A. 31026'
 'C.A. 34651' 'CA 2144' 'PC 17605' 'C.A. 29395' 'S.P. 3464' 'C.A. 33111'
 'S.O.C. 14879' 'SO/C 14885' 'W./C. 6608' 'SOTON/OQ 392086' 'W.E.P. 5734'
 'C.A. 2315' 'PC 17754' 'PC 17759' 'STON/O 2. 3101294' 'PC 17558'
 'A4. 54510' 'C 17369' 'SOTON/O.Q. 3101307' 'A/5. 3337' 'C.A. 29178'
 'SC/PARIS 2133' 'PC 17593' 'STON/O2. 3101279' 'C.A. 33112' 'S.O.P. 1166'
 'A.5. 11206' 'A/5. 851' 'Fa 265302' 'PC 17597' 'SOTON/OQ 392090'
 'CA. 2343' 'C.A. 33595' 'PC 17318' 'STON/O 2. 3101280' 'PC 17595' 'LINE'
 'SC/PARIS 2131' 'PC 17610' 'A/5 3540' 'SOTON/O.Q. 3101311' 'F.C.C. 13528'
 'A/5 21174' 'STON/O2. 3101283' 'W/C 14208' 'SOTON/OQ 392089' 'PP 4348'
 'SW/PP 751' 'A/5 21173' 'C.A. 29566' 'W./C. 6609' 'C.A. 31921'
 'SCO/W 1585' 'W./C. 14263' 'STON/O 2. 3101275' 'A/5. 3336' 'PC 17585'
```

```
 'PC 17755' 'C.A. 17248' 'PC 17582' 'PC 17760' 'PC 17596' 'C.A. 2673'
 'A/5. 10482' 'SC/Paris 2163' 'PC 17612' 'A/5 2466' 'PC 17758' 'P/PP 3381'
 'PC 17485' 'PC 17608' 'F.C.C. 13529' 'A/5 21172' 'PC 17611' 'C.A. 37671'
 'SC/PARIS 2167' 'SOTON/O.Q. 3101310' 'C 7076' 'PC 17477' 'C 7077'
 'PC 17757' 'STON/O 2. 3101293' 'SC 1748' 'STON/O 2. 3101289'
 'STON/O 2. 3101269' 'A/5. 13032' 'A/4. 34244' 'SOTON/O.Q. 392078'
 'SC/AH 3085' 'STON/O 2. 3101274' 'C.A. 18723' 'A/5 2817' 'F.C.C. 13531'
 'A/S 2816' 'SOTON/O.Q. 3101306' 'SC/AH Basle 541' 'A/5 3594' 'A.5. 18509'
 'SOTON/OQ 3101317' 'PC 17609' 'A/4 45380' 'C.A. 6212' 'C 4001'
 'SOTON/OQ 3101316' 'PC 17473' 'PC 17603' 'C.A. 34260' 'W./C. 14258'
 'PC 17483' 'PC 17761' 'WE/P 5735' 'SC/PARIS 2146' 'SOTON/OQ 392082'
 'SOTON/O.Q. 392087' 'A/4 48871' 'S.W./PP 752' 'PC 17474' 'A/4. 20589'
 'STON/O 2. 3101286' 'A./5. 3235' 'STON/O 2. 3101273' 'A/5 3902'
 'SC/AH 29037' 'SOTON/O.Q. 3101305' 'STON/O 2. 3101292' 'S.O./P.P. 751'
 'CA. 2314' 'STON/O 2. 3101285' 'A/5 3536' 'F.C. 12750' 'C.A. 24580'
 'PC 17475' 'PC 17476' 'PC 17482' 'STON/O2. 3101271' 'STON/O 2. 3101288'
 'SOTON/O2 3101272' 'S.O./P.P. 3' 'W./C. 6607' 'SOTON/O.Q. 3101312'
 'PC 17600' 'STON/O2. 3101290' 'S.C./PARIS 2079' 'C 7075' 'PC 17756'
 'SOTON/O2 3101287' 'C.A. 5547' 'PC 17592' 'SC/PARIS 2149' 'PC 17590'
 'C.A./SOTON 34068' 'SOTON/OQ 392076']
```

In [78]:

```python
import re
L1 = []
for i in range(len(train)):
    if (train['Ticket'][i].isdigit() == False):
        train['Ticket'][i] = re.sub('A/5\.','A/5',train["Ticket"][i])
        train['Ticket'][i] = re.sub('A\./5\.','A/5',train["Ticket"][i])
        train['Ticket'][i] = re.sub('A\.5\.','A/5',train["Ticket"][i])
        train['Ticket'][i] = re.sub('C\.A\.','CA',train["Ticket"][i])
        train['Ticket'][i] = re.sub('A/4\.','A/4',train["Ticket"][i])
        train['Ticket'][i] = re.sub('A4\.','A/4',train["Ticket"][i])
        train['Ticket'][i] = re.sub('CA\.','CA',train["Ticket"][i])
        train['Ticket'][i] = re.sub('W\.E\.P\.','WEP',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.P\.','SP',train["Ticket"][i])
        train['Ticket'][i] = re.sub('SOTON/O2','STON/O2',train["Ticket"][i])
        train['Ticket'][i] = re.sub('STON/O2\.','STON/O2',train["Ticket"][i])
        train['Ticket'][i] = re.sub('SOTON/O\.Q\.','SOTON/OQ',train["Ticket"][i])
        train['Ticket'][i] = re.sub('F\.C\.C\.','FCC',train["Ticket"][i])
        train['Ticket'][i] = re.sub('W\./C\.','W/C',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.C\./PARIS','SC/PARIS',train["Ticket"][i])
        train['Ticket'][i] = re.sub('F\.C\.','FC',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.O\.C\.','SOC',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.O\.P\.','SOP',train["Ticket"][i])
        train['Ticket'][i] = re.sub('Fa','FA',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.O\./P\.P\.','SO/PP',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.W\./PP','SW/PP',train["Ticket"][i])
        train['Ticket'][i] = re.sub('S\.C\./A\.4\.','SC/A4',train["Ticket"][i])
        train['Ticket'][i] = re.sub('STON/O 2\.','STON/O2',train["Ticket"][i])
        L1.append(train['Ticket'][i])

Lii = pd.Series(L1)
print("Total:",Lii.nunique(),'\n',Lii.unique())
```

```
Total: 167
 ['A/5 21171' 'PC 17599' 'STON/O2 3101282' 'PP 9549' 'A/5 2151' 'PC 17601'
 'PC 17569' 'CA 24579' 'PC 17604' 'A/5 2152' 'SC/Paris 2123' 'SC/A4 23567'
 'A/4 39886' 'PC 17572' 'CA 31026' 'CA 34651' 'CA 2144' 'PC 17605'
 'CA 29395' 'SP 3464' 'CA 33111' 'SOC 14879' 'SO/C 14885' 'W/C 6608'
 'SOTON/OQ 392086' 'WEP 5734' 'CA 2315' 'PC 17754' 'PC 17759'
 'STON/O2 3101294' 'PC 17558' 'A/4 54510' 'C 17369' 'SOTON/OQ 3101307'
 'A/5 3337' 'CA 29178' 'SC/PARIS 2133' 'PC 17593' 'STON/O2 3101279'
```

```
 'CA 33112' 'SOP 1166' 'A/5 11206' 'A/5 851' 'FA 265302' 'PC 17597'
 'SOTON/OQ 392090' 'CA 2343' 'CA 33595' 'PC 17318' 'STON/O2 3101280'
 'PC 17595' 'LINE' 'SC/PARIS 2131' 'PC 17610' 'A/5 3540'
 'SOTON/OQ 3101311' 'FCC 13528' 'A/5 21174' 'STON/O2 3101283' 'W/C 14208'
 'SOTON/OQ 392089' 'PP 4348' 'SW/PP 751' 'A/5 21173' 'CA 29566' 'W/C 6609'
 'CA 31921' 'SCO/W 1585' 'W/C 14263' 'STON/O2 3101275' 'A/5 3336'
 'PC 17585' 'PC 17755' 'CA 17248' 'PC 17582' 'PC 17760' 'PC 17596'
 'CA 2673' 'A/5 10482' 'SC/Paris 2163' 'PC 17612' 'A/5 2466' 'PC 17758'
 'P/PP 3381' 'PC 17485' 'PC 17608' 'FCC 13529' 'A/5 21172' 'PC 17611'
 'CA 37671' 'SC/PARIS 2167' 'SOTON/OQ 3101310' 'C 7076' 'PC 17477'
 'C 7077' 'PC 17757' 'STON/O2 3101293' 'SC 1748' 'STON/O2 3101289'
 'STON/O2 3101269' 'A/5 13032' 'A/4 34244' 'SOTON/OQ 392078' 'SC/AH 3085'
 'STON/O2 3101274' 'CA 18723' 'A/5 2817' 'FCC 13531' 'A/S 2816'
 'SOTON/OQ 3101306' 'SC/AH Basle 541' 'A/5 3594' 'A/5 18509'
 'SOTON/OQ 3101317' 'PC 17609' 'A/4 45380' 'CA 6212' 'C 4001'
 'SOTON/OQ 3101316' 'PC 17473' 'PC 17603' 'CA 34260' 'W/C 14258'
 'PC 17483' 'PC 17761' 'WE/P 5735' 'SC/PARIS 2146' 'SOTON/OQ 392082'
 'SOTON/OQ 392087' 'A/4 48871' 'SW/PP 752' 'PC 17474' 'A/4 20589'
 'STON/O2 3101286' 'A/5 3235' 'STON/O2 3101273' 'A/5 3902' 'SC/AH 29037'
 'SOTON/OQ 3101305' 'STON/O2 3101292' 'SO/PP 751' 'CA 2314'
 'STON/O2 3101285' 'A/5 3536' 'FC 12750' 'CA 24580' 'PC 17475' 'PC 17476'
 'PC 17482' 'STON/O2 3101271' 'STON/O2 3101288' 'STON/O2 3101272'
 'SO/PP 3' 'W/C 6607' 'SOTON/OQ 3101312' 'PC 17600' 'STON/O2 3101290'
 'SC/PARIS 2079' 'C 7075' 'PC 17756' 'STON/O2 3101287' 'CA 5547'
 'PC 17592' 'SC/PARIS 2149' 'PC 17590' 'CA/SOTON 34068' 'SOTON/OQ 392076']
```

To categorize various types of tickets, we make a new column called "**Ticket_type"**.

In [79]:

```
train['Ticket_type'] = ""

for i in range(len(train)):
    if (train['Ticket'][i].isdigit() == False):
        train['Ticket_type'][i] = train['Ticket'][i].split(" ")[0]

print(train['Ticket_type'].unique(),'\n\nTotal: ',train['Ticket_type'].nunique())
```

```
['A/5' 'PC' 'STON/O2' '' 'PP' 'CA' 'SC/Paris' 'SC/A4' 'A/4' 'SP' 'SOC'
 'SO/C' 'W/C' 'SOTON/OQ' 'WEP' 'C' 'SC/PARIS' 'SOP' 'FA' 'LINE' 'FCC'
 'SW/PP' 'SCO/W' 'P/PP' 'SC' 'SC/AH' 'A/S' 'WE/P' 'SO/PP' 'FC' 'CA/SOTON']

Total:  31
```

As we can see here, there is a null unique element too. This is natural because we still have the numeric tickets to be added in this newly made column **Ticket_type**.

Till now, we have checked for tickets starting with a code or a string.

Let's now check for the other tickets too.

In [80]:

```
L2 = []
for i in range(len(train)):
    if (train['Ticket'][i].isdigit() == True):
        if (len(train['Ticket'][i]) == 6):
            train["Ticket_type"][i] = "6_Digit_Number"
        elif (len(train['Ticket'][i]) == 5):
            train["Ticket_type"][i] = "5_Digit_Number"
        elif (len(train['Ticket'][i]) == 4):
            train["Ticket_type"][i] = "4_Digit_Number"

train['Ticket_type'].unique()
```

Out[80]:
```
array(['A/5', 'PC', 'STON/O2', '6_Digit_Number', '5_Digit_Number', 'PP',
       '4_Digit_Number', 'CA', 'SC/Paris', 'SC/A4', '', 'A/4', 'SP',
       'SOC', 'SO/C', 'W/C', 'SOTON/OQ', 'WEP', 'C', 'SC/PARIS', 'SOP',
       'FA', 'LINE', 'FCC', 'SW/PP', 'SCO/W', 'P/PP', 'SC', 'SC/AH',
       'A/S', 'WE/P', 'SO/PP', 'FC', 'CA/SOTON'], dtype=object)
```

In [81]:
```
train.groupby(["Ticket_type"])["Survived"].mean()
```

Out[81]:
```
Ticket_type
                  0.222222
4_Digit_Number    0.371134
5_Digit_Number    0.618321
6_Digit_Number    0.320482
A/4               0.000000
A/5               0.095238
A/S               0.000000
C                 0.400000
CA                0.341463
CA/SOTON          0.000000
FA                0.000000
FC                0.000000
FCC               0.800000
LINE              0.250000
P/PP              0.500000
PC                0.650000
PP                0.666667
SC                1.000000
SC/A4             0.000000
SC/AH             0.666667
SC/PARIS          0.428571
SC/Paris          0.500000
SCO/W             0.000000
SO/C              1.000000
SO/PP             0.000000
SOC               0.000000
SOP               0.000000
SOTON/OQ          0.133333
SP                0.000000
STON/O2           0.400000
SW/PP             1.000000
W/C               0.100000
WE/P              0.500000
WEP               0.000000
Name: Survived, dtype: float64
```

Passengers with a **5-digit number ticket** and an **"SC/AH"** have about 60% chances of survival, whereas passengers with an **"A4"** and an **"FA"** ticket have zero chances of survival. Therefore, this feature seems to have a mathematical correlation with the feature "Survived".

In [82]:
```
train['Ticket_type'].describe()
```

Out[82]:
```
count                891
unique                34
top       6_Digit_Number
freq                 415
Name: Ticket_type, dtype: object
```

In [83]:
```
train["Ticket_type"].isnull().sum()
```

Out[83]: 0

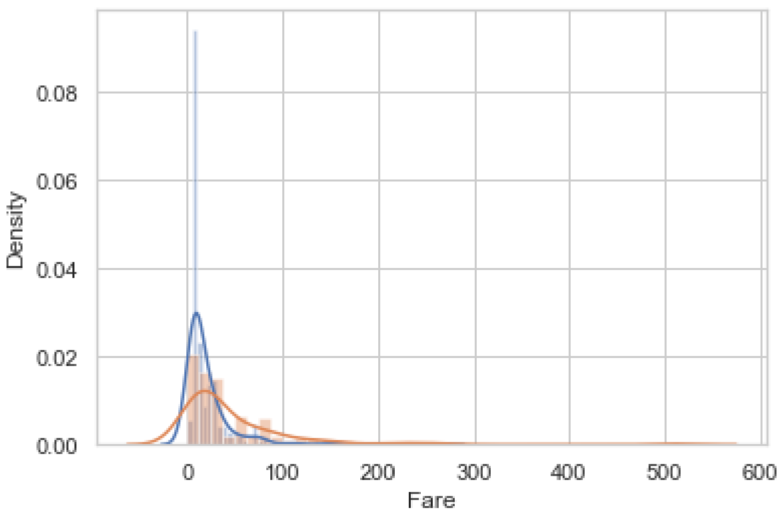
In [84]:
```
df_bin["Ticket_type"] = train["Ticket_type"]
```

Feature: Fare

Relevance of the feature:

In [85]:
```
sns.distplot(train["Fare"][train["Survived"]==0])
sns.distplot(train["Fare"][train["Survived"]==1])
```

Out[85]: <AxesSubplot:xlabel='Fare', ylabel='Density'>



As the fare **increases**, the orange curve (Survived = 1) is **dominating** the blue curve (Survived = 0). Therefore, there is a strong mathematical correlation between "Fare" and "Survived".

In [86]:
```
train.Fare.isnull().sum()
```

Out[86]: 0

In [87]:
```
train.Fare.dtype
```

Out[87]: dtype('float64')

In [88]:
```
train.Fare.describe()
```

Out[88]:
```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```

The mean fare for travelling on Titanic was 32 while the costilest ticket was sold at 512.

In [89]:
```
# For df_bin
bin_cut = pd.cut(train["Fare"], bins=10)
f_interval = bin_cut.cat.categories[0]
new_interval = pd.Interval(0,f_interval.right)
bin_cut = bin_cut.cat.rename_categories({f_interval:new_interval})
```

In [90]:
```
df_bin["Fare"] = bin_cut
```

In [91]:
```
df_bin.head()
```

Out[91]:

	Survived	Pclass	Title	Sex	Age	Family_Size	Ticket_type	Fare
0	0	3	Mr	1	(18, 30]	Small	A/5	(0.0, 51.233]
1	1	1	Mrs	0	(30, 50]	Small	PC	(51.233, 102.466]
2	1	3	Miss	0	(18, 30]	Alone	STON/O2	(0.0, 51.233]
3	1	1	Mrs	0	(30, 50]	Small	6_Digit_Number	(51.233, 102.466]
4	0	3	Mr	1	(30, 50]	Alone	6_Digit_Number	(0.0, 51.233]

Feature: Cabin

In [92]:
```
# Percentage null values
round(((train.Cabin.isnull().sum()/len(train))*100),0)
```

Out[92]: 77.0

The feature **Cabin** has 77% missing values. Let's drop this!

In [93]:
```
train = train.drop("Cabin", axis = 1)
```

In [94]:
```
train.head()
```

Out[94]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Titl
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	S	M
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C	Mr
2	3	1	3	Heikkinen, Miss.	0	26.0	0	0	STON/O2 3101282	7.9250	S	Mis

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Titl
				Laina								
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	S	Mr
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	S	M

Feature: Embarked

Relavance of the feature:

In [95]:
```
train.groupby(["Embarked"])["Survived"].mean()
```

Out[95]:
```
Embarked
C    0.553571
Q    0.389610
S    0.336957
Name: Survived, dtype: float64
```

Passengers who embarked at station "C" have 55% chances of survival. Therefore, this feature seems to play a role in determining the survival chances.

In [96]:
```
train.Embarked.isnull().sum()
```

Out[96]:
```
2
```

In [97]:
```
train.Embarked.describe()
```

Out[97]:
```
count     889
unique      3
top         S
freq      644
Name: Embarked, dtype: object
```

In [98]:
```
train.Embarked.fillna("S", inplace = True)
```

In [99]:
```
train.Embarked.isnull().sum()
```

Out[99]:
```
0
```

In [100…
```
df_bin['Embarked'] = train['Embarked']
```

In [101…
```
df_bin.Pclass = df_bin.Pclass.astype('object')
```

In [102…

```
df_bin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   Survived     891 non-null    int64   
 1   Pclass       891 non-null    object  
 2   Title        891 non-null    object  
 3   Sex          891 non-null    object  
 4   Age          891 non-null    category
 5   Family_Size  891 non-null    object  
 6   Ticket_type  891 non-null    object  
 7   Fare         891 non-null    category
 8   Embarked     891 non-null    object  
dtypes: category(2), int64(1), object(6)
memory usage: 51.3+ KB
```

Feature Encoding

For df_bin (One hot encoding)

In [103…

```
df_bin.head()
```

Out[103…

	Survived	Pclass	Title	Sex	Age	Family_Size	Ticket_type	Fare	Embarked
0	0	3	Mr	1	(18, 30]	Small	A/5	(0.0, 51.233]	S
1	1	1	Mrs	0	(30, 50]	Small	PC	(51.233, 102.466]	C
2	1	3	Miss	0	(18, 30]	Alone	STON/O2	(0.0, 51.233]	S
3	1	1	Mrs	0	(30, 50]	Small	6_Digit_Number	(51.233, 102.466]	S
4	0	3	Mr	1	(30, 50]	Alone	6_Digit_Number	(0.0, 51.233]	S

In [104…

```
df_bin_enc = pd.get_dummies(df_bin, columns = ["Pclass","Title","Sex","Age","Family_Siz
df_bin_enc.head()
```

Out[104…

	Survived	Pclass_2	Pclass_3	Title_Dr	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Other	Sex_1
0	0	0	1	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0	1	0	0
2	1	0	1	0	0	1	0	0	0	0
3	1	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	1	0	0	1

5 rows × 60 columns

Machine Learning Model - Decision Trees

Let's seperate the data

In [105…

```
# Select the dataframe
selected_df = df_bin_enc
```

In [106…

```
# Splitting the dataframe into data and output label
X = selected_df.drop("Survived", axis = 1) # data
y = selected_df["Survived"] # output target
```

In [107…

```
X.head()
```

Out[107…

	Pclass_2	Pclass_3	Title_Dr	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Other	Sex_1	Age_(18, 30]
0	0	1	0	0	0	1	0	0	1	1
1	0	0	0	0	0	0	1	0	0	0
2	0	1	0	0	1	0	0	0	0	1
3	0	0	0	0	0	0	1	0	0	0
4	0	1	0	0	0	1	0	0	1	0

5 rows × 59 columns

In [108…

```
y.head()
```

Out[108…

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [109…

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
classifier = DecisionTreeClassifier()
classifier.fit(X_train,y_train)
```

Out[109…

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [110…

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

In [111…

```
from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

Out[111…
```
0.7597765363128491
```

The model is able to predict the survival of passengers with 75% accuracy.

Since this is a base model, there is a scope of improvement in the performance of the model using hyperparameter tuning.

We will use **GridSearchCV** to tune the hyperparameters and try to achieve the best possible accuracy.

Hyperparameter Tuning

For the hyperparameter tuning, we will consider following Decision Tree parameters:

- Crierion
- Max Depth
- Minimum Sample Split

In [112…
```
param_dict = {
    "criterion":["gini","entropy"],
    "max_depth":[None,5,10,15],
    "min_samples_split":[2, 5, 10]
}
```

In [113…
```
grid = GridSearchCV(classifier,param_grid=param_dict,cv=10,n_jobs=-1)
```

In [114…
```
grid.fit(X_train,y_train)
```

Out[114…
```
GridSearchCV
estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

In [115…
```
grid.best_estimator_
```

Out[115…
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_split=10)
```

In [116…
```
grid.best_score_
```

Out[116…
```
0.8313967136150235
```

GridSearchCV suggests the following parameters:

- Criterion: Entropy
- Max Depth: 5
- Minimum Sample Split: 10

The accuracy promised when these hyperparameters are used is 83%.

Revised Decision Tree Model

In [117…

```
best_params = grid.best_params_
```

In [118…

```
best_classifier = DecisionTreeClassifier(**best_params)
best_classifier.fit(X_train,y_train)
```

Out[118…

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_split=10)
```

In [119…

```
y_pred_new = best_classifier.predict(X_test)
```

In [120…

```
accuracy_score(y_pred_new,y_test)
```

Out[120…

```
0.8156424581005587
```

After hyperparamter tuning using GridSearchCV, we have achieved an accuracy of 81.5%, an improvement on the initial baseline model which had the accuracy of 75.9%.