In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
ls
```

```
 Volume in drive C is Windows
 Volume Serial Number is F8D0-9F99

 Directory of C:\Users\Hajir\Machine Learning Course

02/21/2022  07:30 PM    <DIR>          .
02/21/2022  07:30 PM    <DIR>          ..
02/15/2022  05:24 PM    <DIR>          .ipynb_checkpoints
01/22/2022  08:50 PM             4,756 Advertising.csv
01/19/2022  05:44 PM             9,869 First ML program.ipynb
02/11/2022  02:55 PM           281,466 Linear Regression Excersise 1.ipynb
02/21/2022  07:30 PM           975,199 Logistic Regression.ipynb
02/15/2022  05:25 PM            23,480 SL_Logistics_Regression_+ Decision tree_RF_Practi
ce.ipynb
02/13/2022  09:39 AM            16,843 SL_Logistics_Regression_Practice.ipynb
02/11/2022  03:57 PM            60,302 titanic_train.csv
               7 File(s)      1,371,915 bytes
               3 Dir(s)  77,878,464,512 bytes free
```

In [3]:
```python
df = pd.read_csv('titanic_train.csv') # it will show us the index column and better to
df
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	I
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	
...	...	...	...	...	...	...	...	...	...	...	...	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	I
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	

891 rows × 12 columns

In [4]:

```
df = pd.read_csv('titanic_train.csv', index_col=0) # here we remove the index column be
df
```

Out[4]:

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

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embar
PassengerId											
...	...	...	...	...	...	...	...	...	...	...	
887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	
888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	
889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	
890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	
891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	

891 rows × 11 columns

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Name      891 non-null    object 
 3   Sex       891 non-null    object 
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64  
 6   Parch     891 non-null    int64  
 7   Ticket    891 non-null    object 
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object 
 10  Embarked  889 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
```

In [6]:

```
df.isnull().sum() # To detect null values
```

Out[6]:

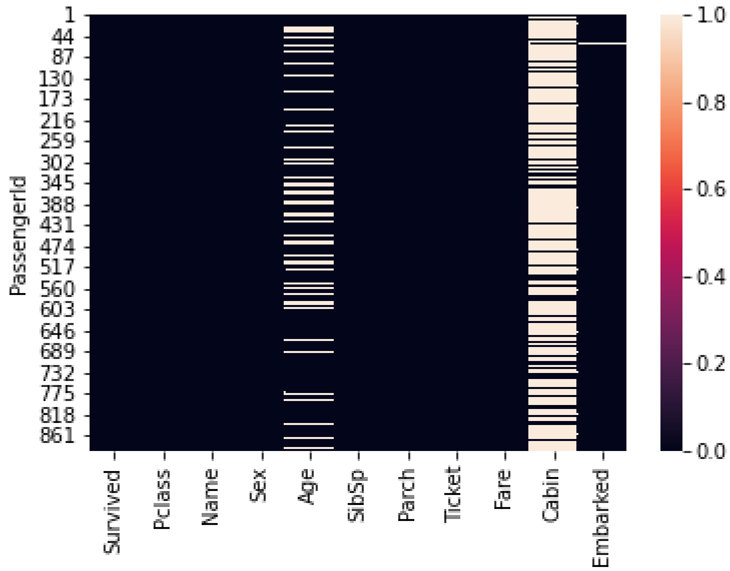
```
Survived      0
Pclass        0
Name          0
Sex           0
Age         177
```

```
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin        687
Embarked       2
dtype: int64
```

In [7]:

```python
#We can see the missing value in graph. Let us see missing values in graph.

sns.heatmap(df.isnull())
```

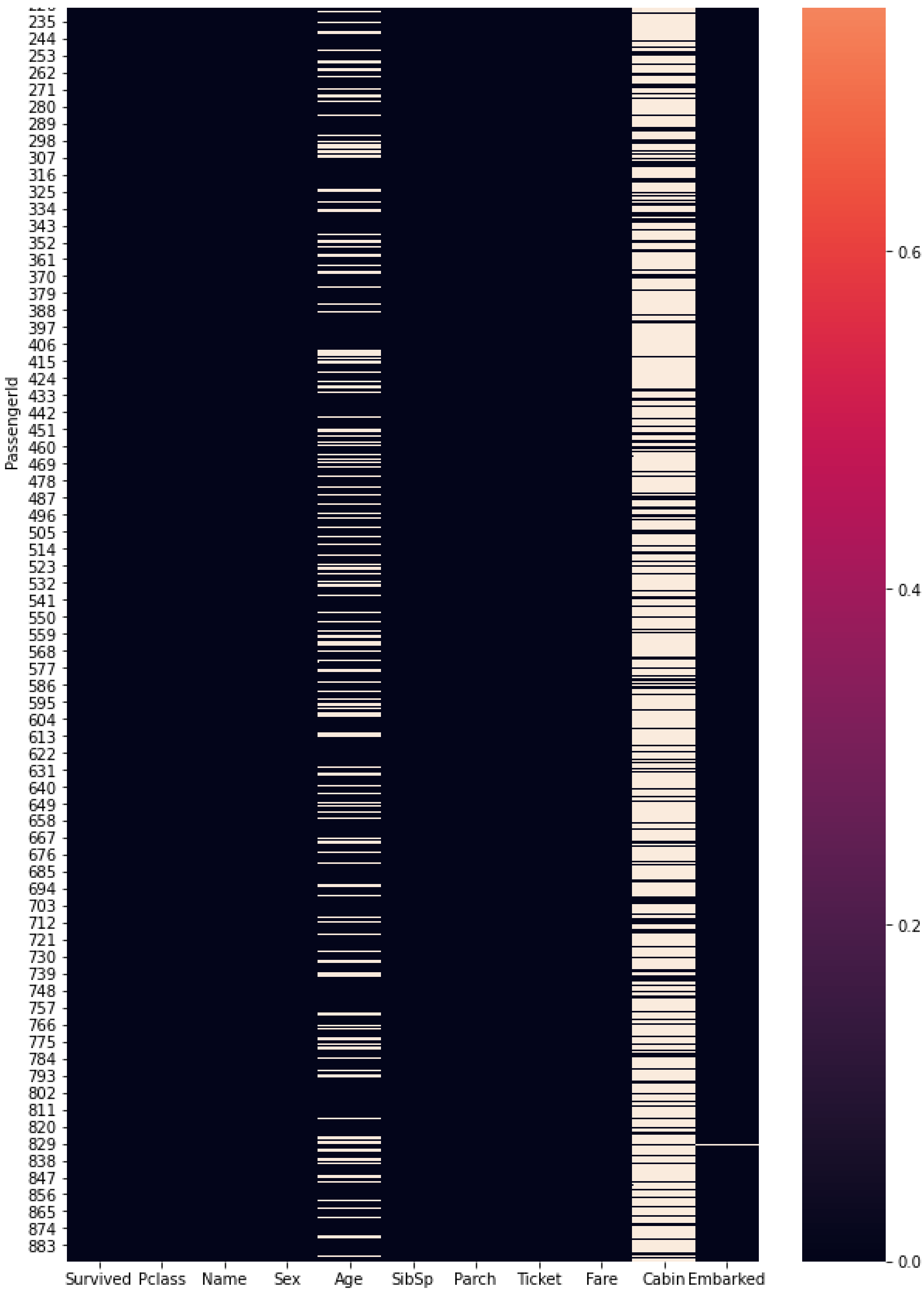
Out[7]: <AxesSubplot:ylabel='PassengerId'>



In [8]:

```python
# we notice from above graph is small and we cant see exactly how many values are missi
# Based on our data we decide which graph we choose.

plt.figure(figsize=(10,20)) #This is to increase the size of graph
sns.heatmap(df.isnull())
plt.show
```

Out[8]: <function matplotlib.pyplot.show(close=None, block=None)>



In [9]:

```
# Let us creat assumption based on data

df.columns
```

Out[9]:

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

Assumption iteration or creation 1

'Survived'-----> Target calss(y)--> Dependent Variable

Now will analyze and check column by column if the specific column has relation to survival or not.

'Pclass'---> Need to Check 'Name'-----> Exclude 'Sex'-----> Need to check 'Age'----> Need to check 'SibSp'---> Need to check 'Parch'-----> Need to check 'Ticket'---> Exclude 'Fare'----> Exclude 'Cabin'---> Exclude 'Embarked'-----> Need to check

In [10]:

```
## Let us create our data for analytics

df = df.drop(['Name','Ticket','Fare','Cabin'], axis=1) #Want to drop it frpm column
df
```

Out[10]:

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
1	0	3	male	22.0	1	0	S
2	1	1	female	38.0	1	0	C
3	1	3	female	26.0	0	0	S
4	1	1	female	35.0	1	0	S
5	0	3	male	35.0	0	0	S
...	...	...	...	...	...	...	...
887	0	2	male	27.0	0	0	S
888	1	1	female	19.0	0	0	S
889	0	3	female	NaN	1	2	S
890	1	1	male	26.0	0	0	C
891	0	3	male	32.0	0	0	Q

891 rows × 7 columns

Assumption iteration or creation 2

'Survived'-----> Target calss(y)--> Dependent Variable

Now will analyze and check column by column if the specific column has relation to survival or not.

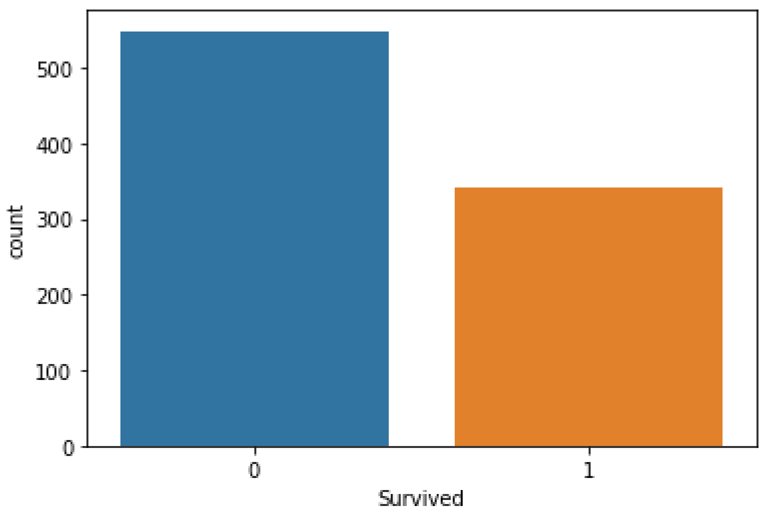
'Pclass'---> Need to Check

'Sex'-----> Need to check 'Age'----> Need to check 'SibSp'---> Need to check 'Parch'-----> Need to check 'Embarked'-----> Need to check

Assumption verification

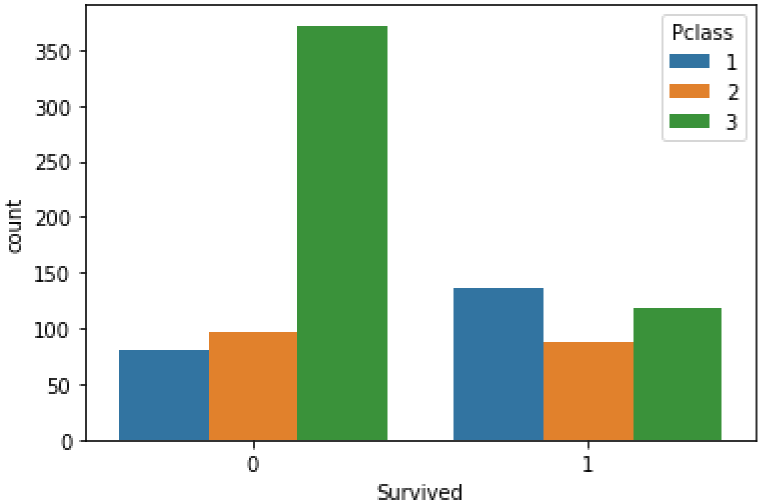
In [11]:

```
sns.countplot(x = 'Survived', data = df)
plt.show()

#what we understand from this graph is that; total of surviver is 300 and all remaining
```



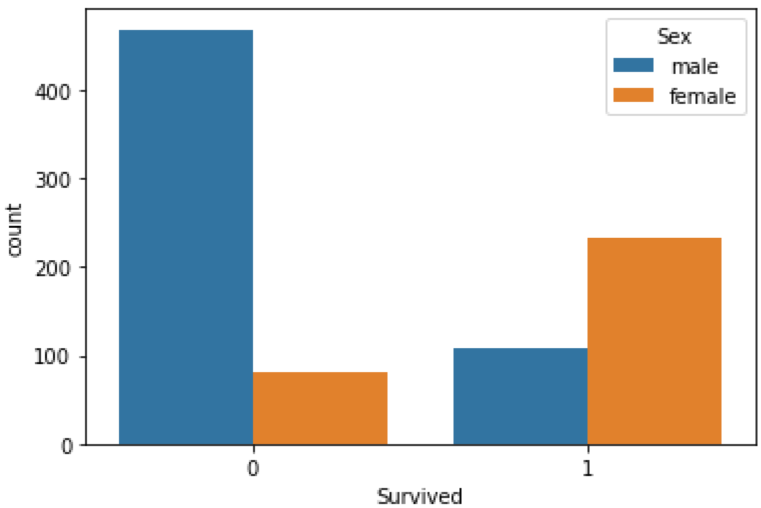
In [12]:

```
sns.countplot(x = 'Survived', data = df, hue ='Pclass')
plt.show()

#According to this graph we can see there is impacting btw survival and pclass. We have
#so will keep Pclass column in our data for logistic regression.
```



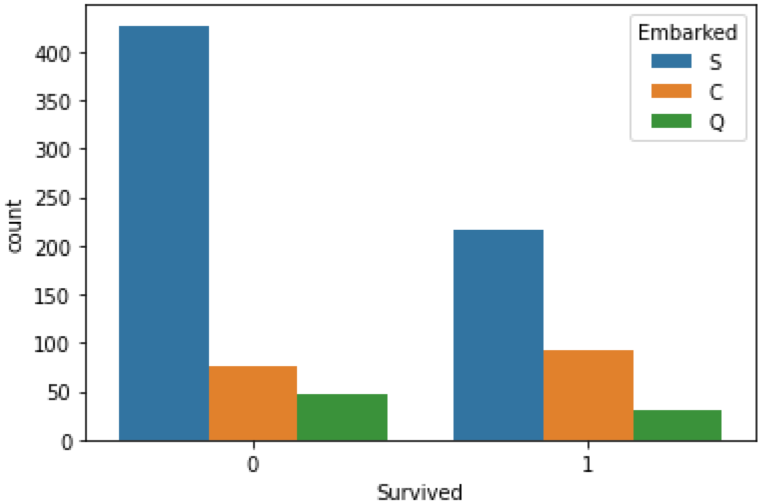
In [13]:

```
sns.countplot(x = 'Survived', data = df, hue ='Sex')
plt.show()

#We notice here the survival percentage for female much higher than male. So, There is
```



In [14]:

```
sns.countplot(x = 'Survived', data = df, hue ='Embarked')
plt.show()

#To understand the graph, first left side show the dead for each embarked and right sid
#let us analyze; First S embarked showed that; approxmatly 550 ppl died and around 220 p
```



Learning so far: We use Countplot for calculating object data. Object data means textual data. It will count how many numbers appears for particular segment or category in a coulmn like (Embarked, Sex and pclass) all these are object.

Assumption iteration or creation 3

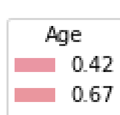
'Survived'-----> Target calss(y)--> Dependent Variable

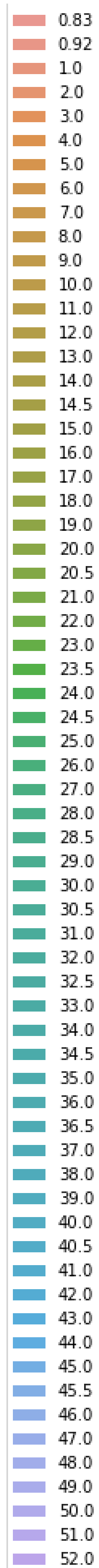
Now will analyze and check column by column if the specific column has relation to survival or not.

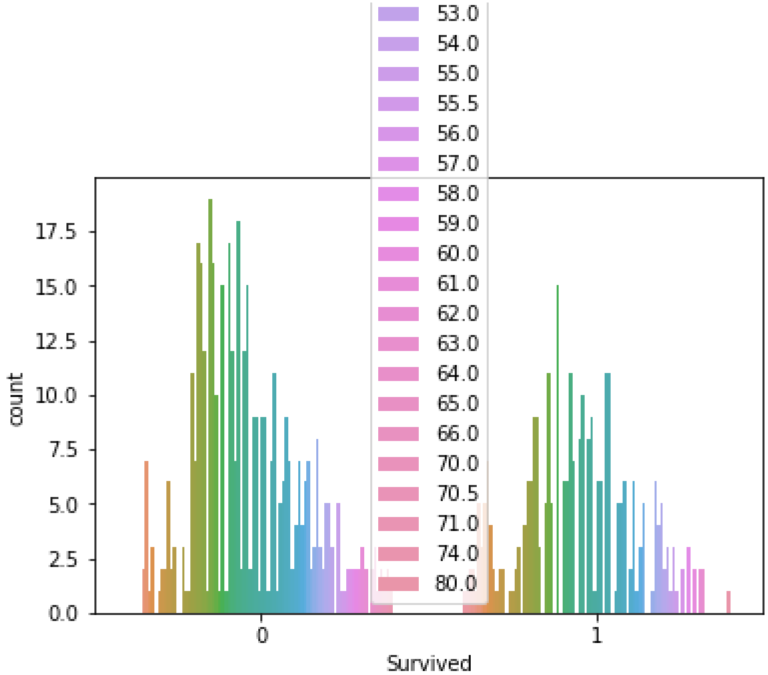
'Pclass'---> Include 'Sex'-----> Include 'Age'----> Need to check 'SibSp'---> Need to check 'Parch'-----> Need to check 'Embarked'-----> Include

In [15]:

```
#Now will check remaining column which are continous data

sns.countplot(x = 'Survived', data = df, hue = 'Age')
plt.show()


#We notice this is not good graph for Age column which is continous data. So, Will crea
```



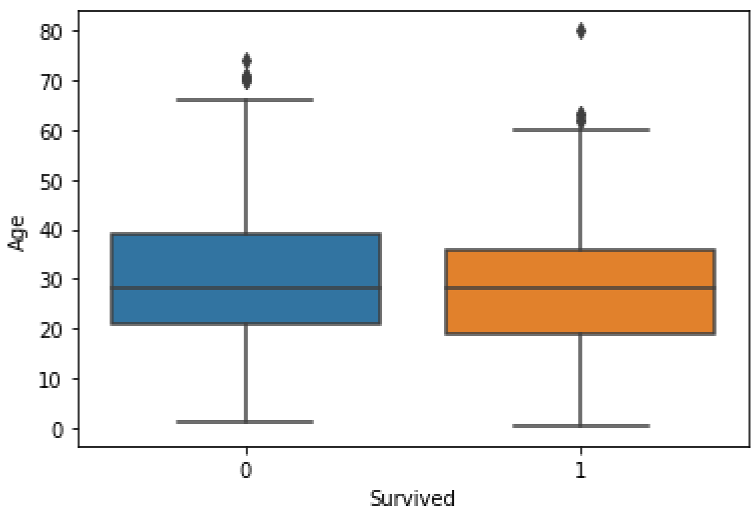

0.83
0.92
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
14.5
15.0
16.0
17.0
18.0
19.0
20.0
20.5
21.0
22.0
23.0
23.5
24.0
24.5
25.0
26.0
27.0
28.0
28.5
29.0
30.0
30.5
31.0
32.0
32.5
33.0
34.0
34.5
35.0
36.0
36.5
37.0
38.0
39.0
40.0
40.5
41.0
42.0
43.0
44.0
45.0
45.5
46.0
47.0
48.0
49.0
50.0
51.0
52.0



In [16]:

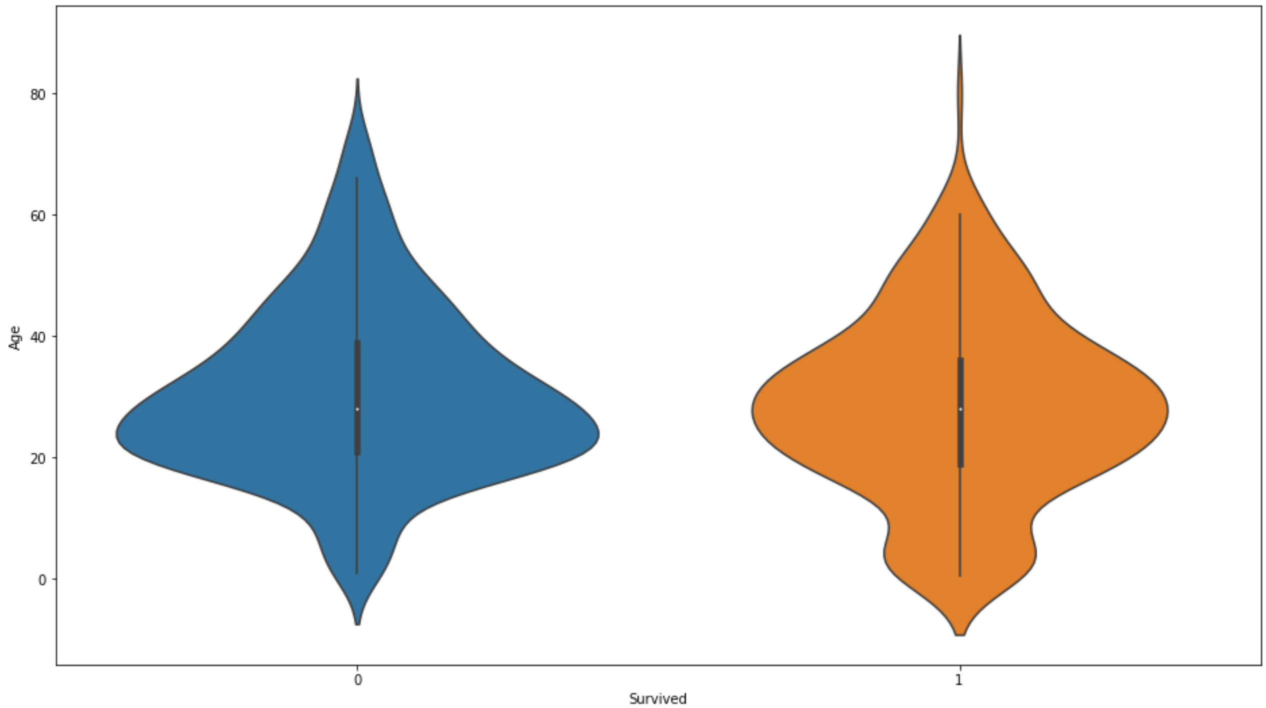
```
sns.boxplot(x = 'Survived', data = df, y= 'Age') #We create boxplot for age column inst
```

Out[16]: <AxesSubplot:xlabel='Survived', ylabel='Age'>

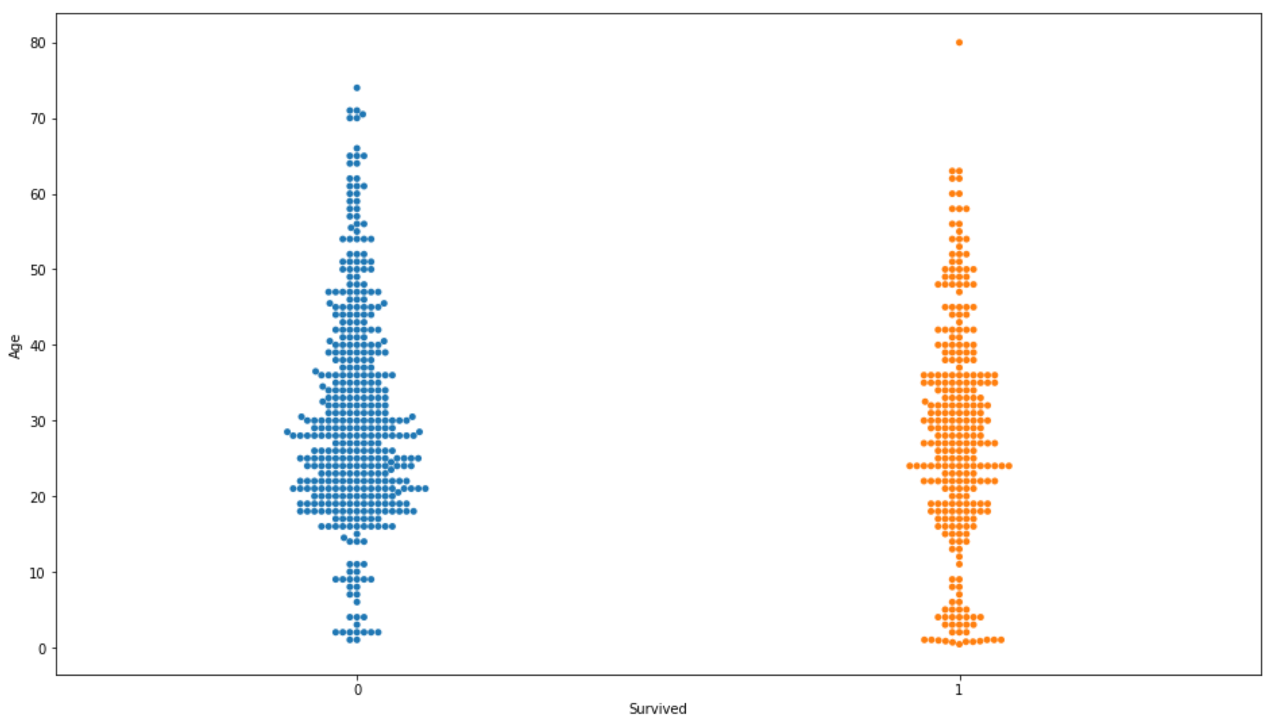


In [17]:

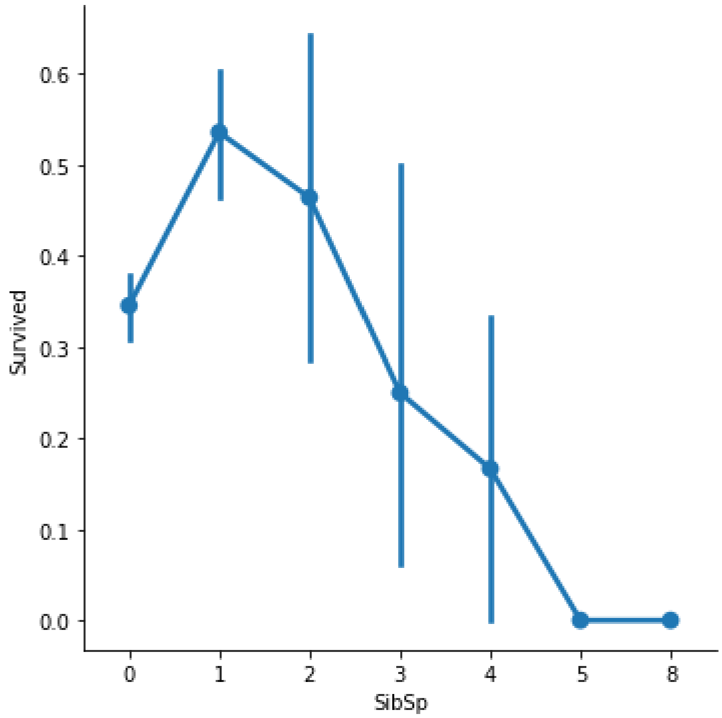
```
plt.figure(figsize=(16, 9))
sns.violinplot(x = 'Survived', data = df, y= 'Age')
plt.show()

#We notice almost same, we can evidence there is different. Will try to create another
```



In [18]:
```
plt.figure(figsize=(16, 9))
sns.swarmplot(x = 'Survived', data = df, y= 'Age')
plt.show()

#According to this graph we notice that who are age less than 10 survived more. We can
```

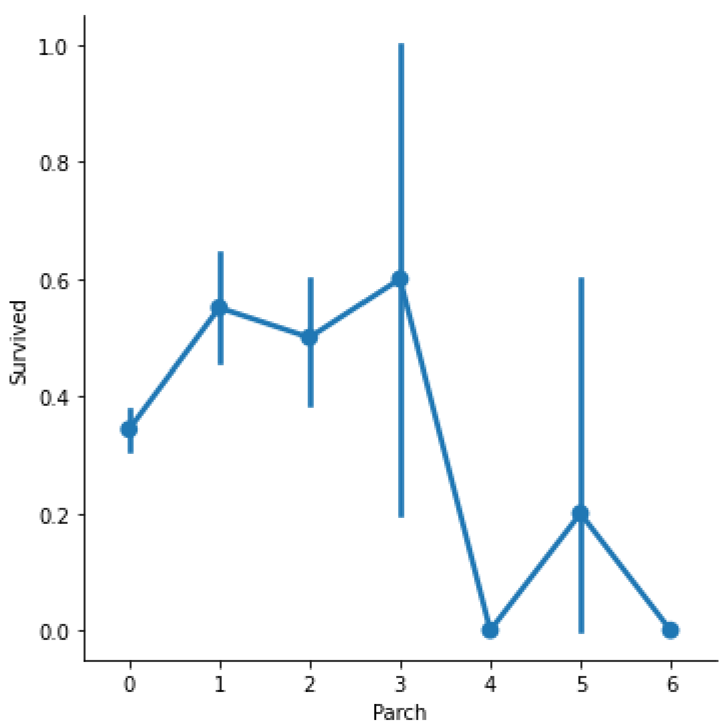


In [19]:
```
#Now will continue and check Sibsb

sns.catplot(x = 'SibSp', data = df, y= 'Survived', kind = 'point')
plt.show()

#According to the graph yes we can see the Sibsb impacting the survival but very little
```



In [20]:

```
sns.catplot(x = 'Parch', data = df, y= 'Survived', kind = 'point')
plt.show()


#We can notice from the graph the rate of survival from 0 sibsb to 3 are higher. when w
```



So, Assumption iteration or creation 4

'Survived'-----> Target calss(y)--> Dependent Variable

'Pclass'---> Include 'Sex'-----> Include 'Age'----> Include 'SibSp'---> Include 'Parch'-----> Include 'Embarked'-----> Include

So, Our df should include the above columns after we verify that, they are all impacting the suvival.

In [21]:
```
df
```

Out[21]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
PassengerId							
1	0	3	male	22.0	1	0	S
2	1	1	female	38.0	1	0	C
3	1	3	female	26.0	0	0	S
4	1	1	female	35.0	1	0	S
5	0	3	male	35.0	0	0	S
...	...	...	...	...	...	...	...
887	0	2	male	27.0	0	0	S
888	1	1	female	19.0	0	0	S
889	0	3	female	NaN	1	2	S
890	1	1	male	26.0	0	0	C
891	0	3	male	32.0	0	0	Q

891 rows × 7 columns

In [22]:
```
df.isnull().sum() #Here to check if we have null values in columns that we consider the
```

Out[22]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Embarked      2
dtype: int64
```

In [23]:
```
#Let us impute the values for the Embarked column
df['Embarked']

#we notice here The vlues in Embarked column are categorical data So, we cant replace t
```

Out[23]:
```
PassengerId
1       S
```

```
2      C
3      S
4      S
5      S
      ..
887    S
888    S
889    S
890    C
891    Q
Name: Embarked, Length: 891, dtype: object
```

In [24]:

```
df['Embarked'].mode()
```

Out[24]:

```
0    S
dtype: object
```

In [25]:

```
df['Embarked'].mode()[0] # we need to see the first value
```

Out[25]:

```
'S'
```

In [26]:

```
mode_Embarked = df['Embarked'].mode()[0]
mode_Embarked
```

Out[26]:

```
'S'
```

In [27]:

```
df['Embarked'] = df['Embarked'].fillna(mode_Embarked)
```

In [28]:

```
#We need to check if we fill the null values in embarked column.

df.isnull().sum()

#We notice there is 0 null value in Embarked column now.
```

Out[28]:

```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Embarked      0
dtype: int64
```

In [29]:

```
#Now will replace values for null values in Age column by mean

df['Age'].mean()
```

Out[29]:

```
29.69911764705882
```

In [30]:

```
#The age value should consider difference btw male and female, So it effect by {Sex}

df[df['Sex']== 'female']['Age'].mean()
```

Out[30]: 27.915708812260537

In [31]:
```
df[df['Sex']== 'female']['Age'].median()
```

Out[31]: 27.0

In [32]:
```
df[df['Sex']== 'male']['Age'].mean()
```

Out[32]: 30.72664459161148

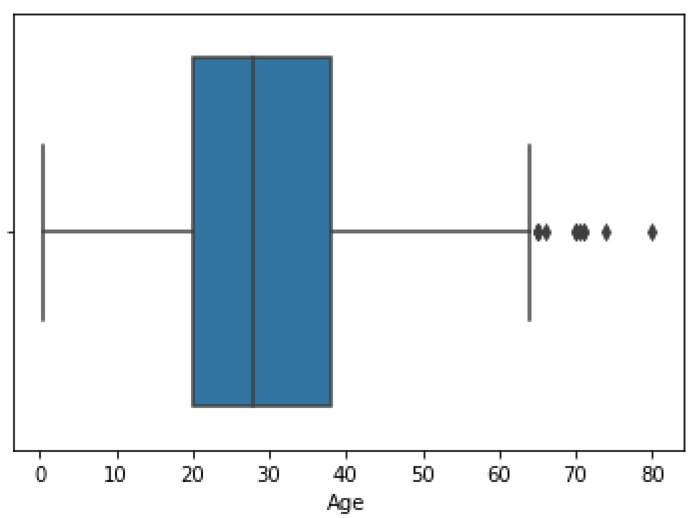
In [33]:
```
df[df['Sex']== 'male']['Age'].median()
```

Out[33]: 29.0

In [34]:
```
# We can see there is no big different btw mean and median. So the data here is not ske
```

In [35]:
```
# To view the mean for female and male

sns.boxplot(x= 'Age', data= df )
```

Out[35]: <AxesSubplot:xlabel='Age'>



In [36]:
```
# Will check if the Pclass can show someting with Age.

print(df[df['Pclass']== 1]['Age'].mean())
print(df[df['Pclass']== 2]['Age'].mean())
print(df[df['Pclass']== 3]['Age'].mean())

# We can see here pclass 1 has ppl who are older than pclass 2 and 3. The difference is
```

```
38.233440860215055
29.87763005780347
25.14061971830986
```

In [37]:
```
# Will create function..
```

```
def age_impute(col):
    Age = col[0]
    Pclass = col[1]


    if pd.isnull(Age):
        if Pclass == 1:
            return 38.23
        elif Pclass == 2:
            return 29.87
        else:
            return 25.14
    else:
        return Age
```

In [38]:

```
df['Age'] #To check the Null Values but here it does not show any NAN because I did fil
```

Out[38]:

```
PassengerId
1      22.0
2      38.0
3      26.0
4      35.0
5      35.0
       ...
887    27.0
888    19.0
889     NaN
890    26.0
891    32.0
Name: Age, Length: 891, dtype: float64
```

In [39]:

```
df['Age'] = df[['Age', 'Pclass']].apply(age_impute, axis = 1) # To impute and apply fil
df['Age']
```

Out[39]:

```
PassengerId
1      22.00
2      38.00
3      26.00
4      35.00
5      35.00
       ...
887    27.00
888    19.00
889    25.14
890    26.00
891    32.00
Name: Age, Length: 891, dtype: float64
```

In [40]:

```
df
```

Out[40]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
PassengerId							
1	0	3	male	22.00	1	0	S

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
PassengerId							
2	1	1	female	38.00	1	0	C
3	1	3	female	26.00	0	0	S
4	1	1	female	35.00	1	0	S
5	0	3	male	35.00	0	0	S
...	...	...	...	...	...	...	...
887	0	2	male	27.00	0	0	S
888	1	1	female	19.00	0	0	S
889	0	3	female	25.14	1	2	S
890	1	1	male	26.00	0	0	C
891	0	3	male	32.00	0	0	Q

891 rows × 7 columns

In [41]:

```
#And now I want to convert data in Sex column to numerical data. To be easy for compute

df['Sex'] = pd.get_dummies(df.Sex)['female']
df
```

Out[41]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
PassengerId							
1	0	3	0	22.00	1	0	S
2	1	1	1	38.00	1	0	C
3	1	3	1	26.00	0	0	S
4	1	1	1	35.00	1	0	S
5	0	3	0	35.00	0	0	S
...	...	...	...	...	...	...	...
887	0	2	0	27.00	0	0	S
888	1	1	1	19.00	0	0	S
889	0	3	1	25.14	1	2	S
890	1	1	0	26.00	0	0	C
891	0	3	0	32.00	0	0	Q

891 rows × 7 columns

In [42]:

```
df = pd.get_dummies(df, columns = ['Embarked']) #Will notice that get_dummies in column
df
```

Out[42]:

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked_C	Embarked_Q	Embarked_S
1	0	3	0	22.00	1	0	0	0	1
2	1	1	1	38.00	1	0	1	0	0
3	1	3	1	26.00	0	0	0	0	1
4	1	1	1	35.00	1	0	0	0	1
5	0	3	0	35.00	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...
887	0	2	0	27.00	0	0	0	0	1
888	1	1	1	19.00	0	0	0	0	1
889	0	3	1	25.14	1	2	0	0	1
890	1	1	0	26.00	0	0	1	0	0
891	0	3	0	32.00	0	0	0	1	0

891 rows × 9 columns

In [43]:

```
#Now We can delete column for Any Embarked. If we have Embarked_Q and Emabrked_S. The E

df = df.drop(['Embarked_C'], axis=1)
df
```

Out[43]:

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked_Q	Embarked_S
1	0	3	0	22.00	1	0	0	1
2	1	1	1	38.00	1	0	0	0
3	1	3	1	26.00	0	0	0	1
4	1	1	1	35.00	1	0	0	1
5	0	3	0	35.00	0	0	0	1
...	...	...	...	...	...	...	...	...
887	0	2	0	27.00	0	0	0	1
888	1	1	1	19.00	0	0	0	1
889	0	3	1	25.14	1	2	0	1
890	1	1	0	26.00	0	0	0	0
891	0	3	0	32.00	0	0	1	0

891 rows × 8 columns

In [44]:

```
#So All our data becaome now numeric data.
```

```
#It is time to split our data into two parts

y = df.Survived
y
```

Out[44]:
```
PassengerId
1      0
2      1
3      1
4      1
5      0
      ..
887    0
888    1
889    0
890    1
891    0
Name: Survived, Length: 891, dtype: int64
```

In [45]:
```
x =df.drop(['Survived'], axis=1) #X will ba all columns expect survive column which is
x
```

Out[45]:

PassengerId	Pclass	Sex	Age	SibSp	Parch	Embarked_Q	Embarked_S
1	3	0	22.00	1	0	0	1
2	1	1	38.00	1	0	0	0
3	3	1	26.00	0	0	0	1
4	1	1	35.00	1	0	0	1
5	3	0	35.00	0	0	0	1
...	...	...	...	...	...	...	...
887	2	0	27.00	0	0	0	1
888	1	1	19.00	0	0	0	1
889	3	1	25.14	1	2	0	1
890	1	0	26.00	0	0	0	0
891	3	0	32.00	0	0	1	0

891 rows × 7 columns

Let us split data into training and testing

In [46]:
```
from sklearn.model_selection import train_test_split
```

In [47]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, random_state = 123)
```

Let us apply Logistic Regression on the IV and DV

In [48]:
```
from sklearn.linear_model import LogisticRegression
```

In [49]:
```
#Will pass Logistic Regression to variable to use it
model = LogisticRegression()
model
```

Out[49]:
```
LogisticRegression()
```

In [50]:
```
#Now Will fit the data into the model for logistic regression

model.fit(x_train, y_train)
```

```
C:\Users\Hajir\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[50]:
```
LogisticRegression()
```

In [51]:
```
#Let us predict for the test data

y_pred = model.predict(x_test)
y_pred

#This code and the bottome one model---> predict_proba(x_test) \\\shows us the probabil
```

Out[51]:
```
array([1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0], dtype=int64)
```

In [52]:
```
model.predict_proba(x_test)

#How I read this: If the probability greater than 0.5 the predict will be 1 it mean wil
```

Out[52]:
```
array([[0.22746369, 0.77253631],
       [0.86071307, 0.13928693],
       [0.40747569, 0.59252431],
       [0.77524934, 0.22475066],
```

```
[0.90434765, 0.09565235],
[0.85592275, 0.14407725],
[0.25908179, 0.74091821],
[0.21744487, 0.78255513],
[0.2568596 , 0.7431404 ],
[0.4173324 , 0.5826676 ],
[0.37110807, 0.62889193],
[0.40737048, 0.59262952],
[0.20034738, 0.79965262],
[0.88482109, 0.11517891],
[0.11297453, 0.88702547],
[0.24299401, 0.75700599],
[0.15342189, 0.84657811],
[0.55831333, 0.44168667],
[0.67149016, 0.32850984],
[0.79828466, 0.20171534],
[0.46029411, 0.53970589],
[0.0784905 , 0.9215095 ],
[0.81354596, 0.18645404],
[0.89848307, 0.10151693],
[0.9182833 , 0.0817167 ],
[0.91714764, 0.08285236],
[0.87310324, 0.12689676],
[0.04022019, 0.95977981],
[0.85771225, 0.14228775],
[0.85070921, 0.14929079],
[0.37923103, 0.62076897],
[0.32999126, 0.67000874],
[0.7777807 , 0.2222193 ],
[0.89139112, 0.10860888],
[0.71788377, 0.28211623],
[0.76813803, 0.23186197],
[0.9432369 , 0.0567631 ],
[0.96110503, 0.03889497],
[0.14727353, 0.85272647],
[0.51661011, 0.48338989],
[0.04477935, 0.95522065],
[0.92468378, 0.07531622],
[0.4004571 , 0.5995429 ],
[0.76282273, 0.23717727],
[0.84079492, 0.15920508],
[0.70182258, 0.29817742],
[0.12384426, 0.87615574],
[0.67782522, 0.32217478],
[0.8630084 , 0.1369916 ],
[0.19577994, 0.80422006],
[0.32151318, 0.67848682],
[0.50670021, 0.49329979],
[0.86559373, 0.13440627],
[0.92426918, 0.07573082],
[0.69251517, 0.30748483],
[0.86071307, 0.13928693],
[0.95419494, 0.04580506],
[0.85224641, 0.14775359],
[0.54256104, 0.45743896],
[0.96411254, 0.03588746],
[0.09787655, 0.90212345],
[0.041957  , 0.958043  ],
[0.34877998, 0.65122002],
[0.0704615 , 0.9295385 ],
[0.88255542, 0.11744458],
[0.94268946, 0.05731054],
[0.86645892, 0.13354108],
[0.87633593, 0.12366407],
[0.55831333, 0.44168667],
```

```
[0.13891945, 0.86108055],
[0.14659842, 0.85340158],
[0.95942303, 0.04057697],
[0.55831333, 0.44168667],
[0.76276701, 0.23723299],
[0.86813773, 0.13186227],
[0.89963768, 0.10036232],
[0.96931675, 0.03068325],
[0.86813773, 0.13186227],
[0.09653287, 0.90346713],
[0.09653287, 0.90346713],
[0.47009675, 0.52990325],
[0.90161054, 0.09838946],
[0.9386079 , 0.0613921 ],
[0.86071307, 0.13928693],
[0.88255542, 0.11744458],
[0.77006749, 0.22993251],
[0.1403257 , 0.8596743 ],
[0.89731614, 0.10268386],
[0.55831333, 0.44168667],
[0.88704864, 0.11295136],
[0.30667253, 0.69332747],
[0.69525613, 0.30474387],
[0.89421437, 0.10578563],
[0.92147738, 0.07852262],
[0.48337076, 0.51662924],
[0.85771225, 0.14228775],
[0.03890361, 0.96109639],
[0.07754824, 0.92245176],
[0.91786175, 0.08213825],
[0.25533467, 0.74466533],
[0.91491342, 0.08508658],
[0.70235831, 0.29764169],
[0.63013414, 0.36986586],
[0.23256675, 0.76743325],
[0.85070921, 0.14929079],
[0.5643492 , 0.4356508 ],
[0.89139112, 0.10860888],
[0.87754228, 0.12245772],
[0.92426918, 0.07573082],
[0.22668998, 0.77331002],
[0.44742219, 0.55257781],
[0.5643492 , 0.4356508 ],
[0.06574459, 0.93425541],
[0.9144761 , 0.0855239 ],
[0.89139112, 0.10860888],
[0.88461937, 0.11538063],
[0.76747454, 0.23252546],
[0.92257054, 0.07742946],
[0.85224641, 0.14775359],
[0.43285349, 0.56714651],
[0.85771225, 0.14228775],
[0.94555178, 0.05444822],
[0.59881491, 0.40118509],
[0.85070921, 0.14929079],
[0.71994061, 0.28005939],
[0.73553853, 0.26446147],
[0.65141618, 0.34858382],
[0.55831333, 0.44168667],
[0.30409409, 0.69590591],
[0.8630084 , 0.1369916 ],
[0.08899402, 0.91100598],
[0.90354879, 0.09645121],
[0.88704864, 0.11295136],
[0.38655294, 0.61344706],
```

```
[0.12976968, 0.87023032],
[0.85224641, 0.14775359],
[0.11290002, 0.88709998],
[0.51395901, 0.48604099],
[0.88766559, 0.11233441],
[0.26748846, 0.73251154],
[0.97061667, 0.02938333],
[0.42892572, 0.57107428],
[0.85874125, 0.14125875],
[0.5331806 , 0.4668194 ],
[0.27909356, 0.72090644],
[0.89963768, 0.10036232],
[0.95336805, 0.04663195],
[0.9144761 , 0.0855239 ],
[0.8955863 , 0.1044137 ],
[0.03890361, 0.96109639],
[0.69136711, 0.30863289],
[0.95924459, 0.04075541],
[0.35542334, 0.64457666],
[0.84079492, 0.15920508],
[0.0626198 , 0.9373802 ],
[0.86071307, 0.13928693],
[0.18814311, 0.81185689],
[0.1995898 , 0.8004102 ],
[0.27636366, 0.72363634],
[0.13483025, 0.86516975],
[0.25908179, 0.74091821],
[0.55581132, 0.44418868],
[0.43743397, 0.56256603],
[0.07074954, 0.92925046],
[0.55831333, 0.44168667],
[0.04521675, 0.95478325],
[0.47689487, 0.52310513],
[0.8630084 , 0.1369916 ],
[0.30409409, 0.69590591],
[0.93922276, 0.06077724],
[0.87790806, 0.12209194],
[0.55831333, 0.44168667],
[0.62904365, 0.37095635],
[0.72859599, 0.27140401],
[0.85070921, 0.14929079],
[0.94991703, 0.05008297],
[0.38948288, 0.61051712],
[0.86071307, 0.13928693],
[0.71994061, 0.28005939],
[0.02764082, 0.97235918],
[0.88064238, 0.11935762],
[0.93812968, 0.06187032],
[0.82601616, 0.17398384],
[0.33542102, 0.66457898],
[0.88557887, 0.11442113],
[0.07003855, 0.92996145],
[0.61282533, 0.38717467],
[0.0631759 , 0.9368241 ],
[0.86813773, 0.13186227],
[0.22603488, 0.77396512],
[0.64370159, 0.35629841],
[0.88766559, 0.11233441],
[0.20269453, 0.79730547],
[0.81874539, 0.18125461],
[0.35627516, 0.64372484],
[0.37143802, 0.62856198],
[0.91714764, 0.08285236],
[0.56863601, 0.43136399],
[0.09024478, 0.90975522],
```

```
       [0.88766559, 0.11233441],
       [0.03507567, 0.96492433],
       [0.30409409, 0.69590591],
       [0.67782522, 0.32217478],
       [0.78283774, 0.21716226],
       [0.57745988, 0.42254012],
       [0.85771225, 0.14228775],
       [0.85070921, 0.14929079],
       [0.15280045, 0.84719955],
       [0.55831333, 0.44168667],
       [0.9432369 , 0.0567631 ],
       [0.90354879, 0.09645121],
       [0.16337614, 0.83662386],
       [0.38984065, 0.61015935],
       [0.92426918, 0.07573082],
       [0.90916022, 0.09083978],
       [0.86071307, 0.13928693],
       [0.7777807 , 0.2222193 ],
       [0.88928318, 0.11071682],
       [0.89694641, 0.10305359],
       [0.85997145, 0.14002855],
       [0.88766559, 0.11233441],
       [0.78427203, 0.21572797],
       [0.8844733 , 0.1155267 ]])
```

Let us check the model Performance.

In [53]:
```
from sklearn import metrics
```

In [54]:
```
matrix = metrics.confusion_matrix(y_test, y_pred)
print(matrix)
```

```
[[121  18]
 [ 23  61]]
```

In [55]:
```
metrics.accuracy_score(y_test, y_pred) #This is mean 81% of the time  my model is able
```

Out[55]:
```
0.8161434977578476
```

This is mean 81% of the time my model is able to predict if somebody has survived or not survived.

Applying Desision tree with same data

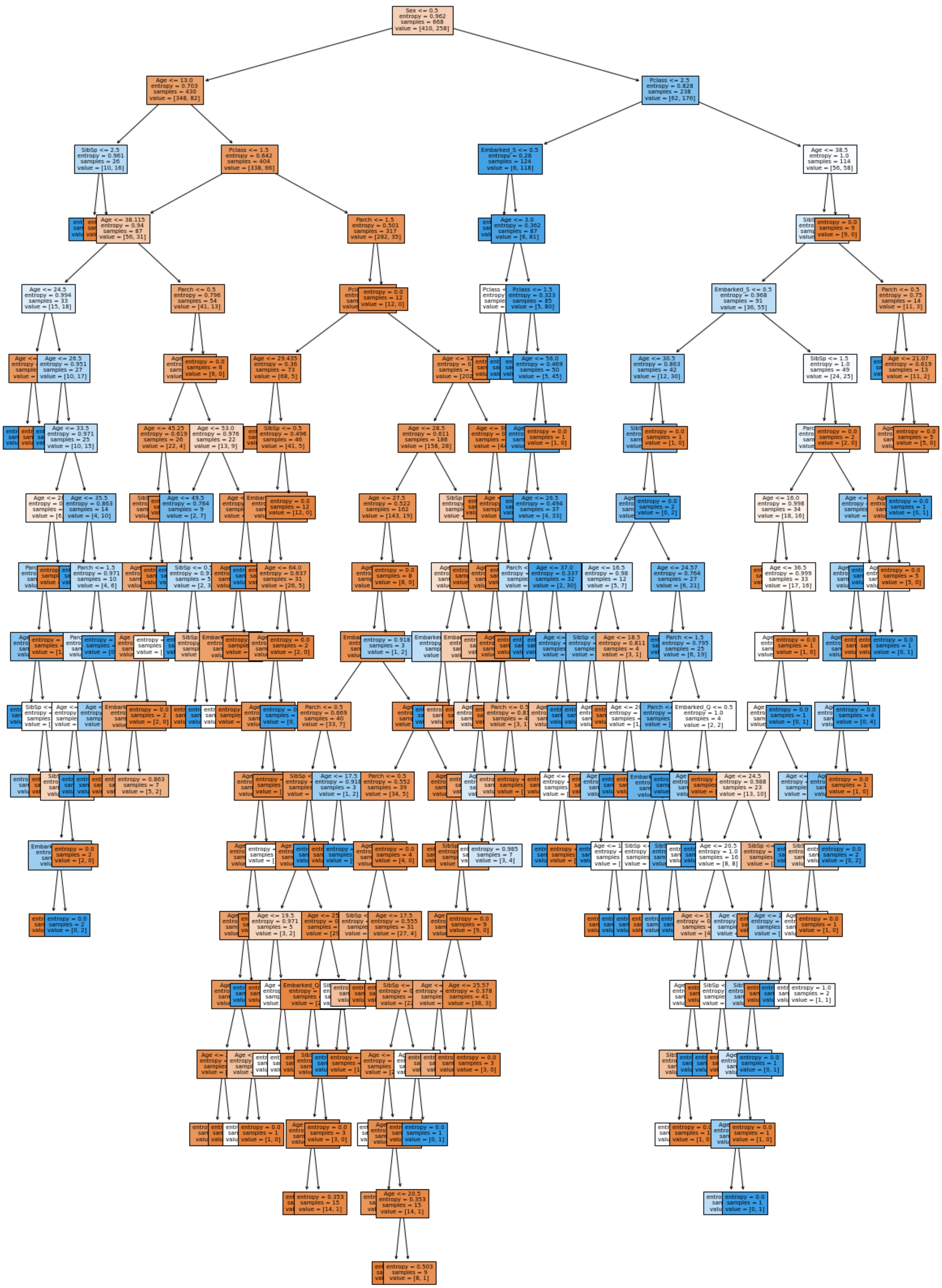
In [56]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [57]:
```
model_dt = DecisionTreeClassifier(criterion = 'entropy')
```

In [58]:
```
model_dt
```
Out[58]:
```
DecisionTreeClassifier(criterion='entropy')
```
In [59]:
```
model_dt.fit(x_train, y_train)
```
Out[59]:
```
DecisionTreeClassifier(criterion='entropy')
```
In [60]:
```
y_pred_dt = model_dt.predict(x_test)
y_pred_dt
```
Out[60]:
```
array([0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0], dtype=int64)
```
In [61]:
```
metrics.accuracy_score(y_test, y_pred_dt)
```
Out[61]:
```
0.7892376681614349
```
In [62]:
```
#The accuracy for decision tree model is 78% while the logistic regression model was 81
```

Let us plot the decision tree graph

In [63]:
```
from sklearn.tree import plot_tree
```
In [64]:
```
x_train.columns #This code because will need the names of column in next code.
```
Out[64]:
```
Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked_Q', 'Embarked_S'], dtype='obj
ect')
```
In [65]:
```
plt.figure(figsize= (20,30))
plot_tree(model_dt, feature_names= ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked
plt.show()
```



In [66]:

```
#It is better dont show the graph when we have alot of independent variables, it will t
# Decision tree is the best when we have small number of raws. it is very good to divid
# When we use Decision tree; When our data is Dichotomas or polynomic. Also, if we have
```

```
# For repeated values decision tree works better. When we have conditions and one condi
# If we have outlires, Abnormality and missing in the data, decision tree can exclude a
# Decision tree is part of classification and regression. Sometimes we use decision tre
```

In [67]:

```
# Let us create confusion matrix

pd.DataFrame(metrics.confusion_matrix(y_test, y_pred_dt),
            columns = ['Predicted not survival', 'Predicted survival'],
            index = ['True not survival', 'True Survival'])
```

Out[67]:

	Predicted not survival	Predicted survival
True not survival	122	17
True Survival	30	54

Overfitting/ underfitting/ just right models.

Checking with logistic regression.

In [68]:

```
y_pred_lr_train = model.predict(x_train)
y_pred_lr_train

print("training data Acuuracy for Log Reg is: ", metrics.accuracy_score(y_train, y_pred

y_pred_lr_test = model.predict(x_test)
print("Testing data Acuuracy for Log Reg is: ", metrics.accuracy_score(y_test, y_pred_l
```

```
training data Acuuracy for Log Reg is:  0.8038922155688623
Testing data Acuuracy for Log Reg is:  0.8161434977578476
```

Checking with decision tree model

In [69]:

```
y_pred_dt_train = model_dt.predict(x_train)
y_pred_dt_train

print("training data Acuuracy for Log Reg is: ", metrics.accuracy_score(y_train, y_pred

y_pred_dt_test = model_dt.predict(x_test)
print("Testing data Acuuracy for Log Reg is: ", metrics.accuracy_score(y_test, y_pred_d
```

```
training data Acuuracy for Log Reg is:  0.9401197604790419
Testing data Acuuracy for Log Reg is:  0.7892376681614349
```

We can see clearly that; Decision tree model is overfitted

To minimize the over fitted in decision tree, we use

Random Forest

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [71]:
```
model_rf = RandomForestClassifier(n_estimators=500, criterion='gini', random_state=123,
#min_samples_leaf clarify means for example random forest dont split after 30 split  be
```

In [72]:
```
model_rf
```

Out[72]:
```
RandomForestClassifier(min_samples_leaf=4, n_estimators=500, random_state=123)
```

In [73]:
```
model_rf.fit(x_train, y_train)
```

Out[73]:
```
RandomForestClassifier(min_samples_leaf=4, n_estimators=500, random_state=123)
```

In [74]:
```
y_pred_rf = model_rf.predict(x_test)
y_pred_rf
```

Out[74]:
```
array([1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0], dtype=int64)
```

In [75]:
```
metrics.accuracy_score(y_test, y_pred_rf)
```

Out[75]:
```
0.8385650224215246
```

Checking the performance of Random State

In [76]:
```
y_pred_rf_train = model_rf.predict(x_train)
print("Training Data Accuracy for RF is:", metrics.accuracy_score(y_train, y_pred_rf_tr

y_pred_rf_test = model_rf.predict(x_test)
print("Testing Data Accuracy for RF is :",metrics.accuracy_score(y_test, y_pred_rf_test
```

```
Training Data Accuracy for RF is: 0.8577844311377245
Testing Data Accuracy for RF is : 0.8385650224215246
```

In [ ]:
```

```