Titanic Exploratory Data Analysis and Prediction

The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during its maiden voyage, the Titanic sank after colliding with an iceberg, resulting in the loss of 1,502 lives out of the 2,224 passengers and crew on board.

The Titanic cost approximately $7.5 million to build and met its fate due to a collision. The Titanic Dataset is an excellent resource for beginners looking to embark on a journey into data science.

The objective of this notebook is to provide an idea of the workflow in a predictive modeling problem. We will explore how to assess features, introduce new ones, and delve into some machine learning concepts.

Contents of the Notebook:

Part1: Exploratory Data Analysis(EDA):

1)Analysis of the features.

2)Finding any relations or trends considering multiple features.

Part2: Feature Engineering and Data Cleaning:

1)Adding any few features.

2)Removing redundant features.

3)Converting features into suitable form for modeling.

Part3: Predictive Modeling

1)Running Basic Machine Learning Algorithms.

2)Cross Validation.

3)Ensembling.

4)Important Features Extraction.

Part1: Exploratory Data Analysis(EDA)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
data=pd.read_csv('../input/train.csv')
```

In [3]:
```
data.head()
```

Out[3]:

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

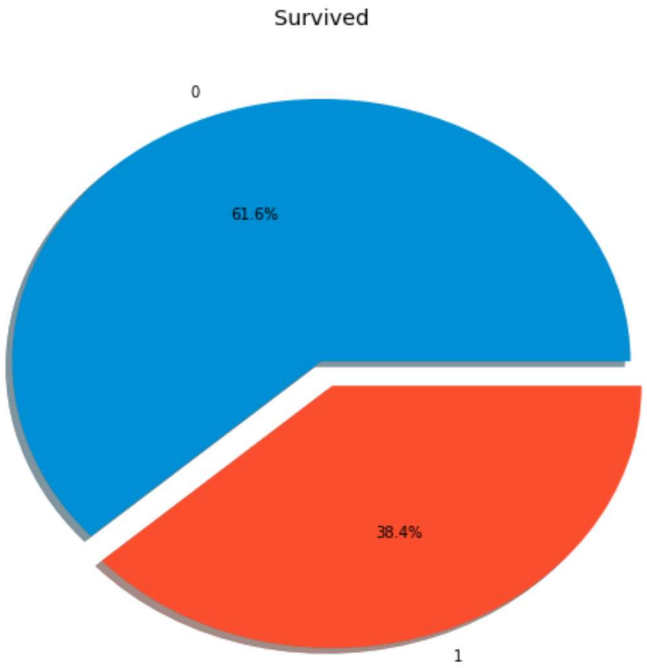
In [4]:
```
data.isnull().sum() #checking for total null values
```

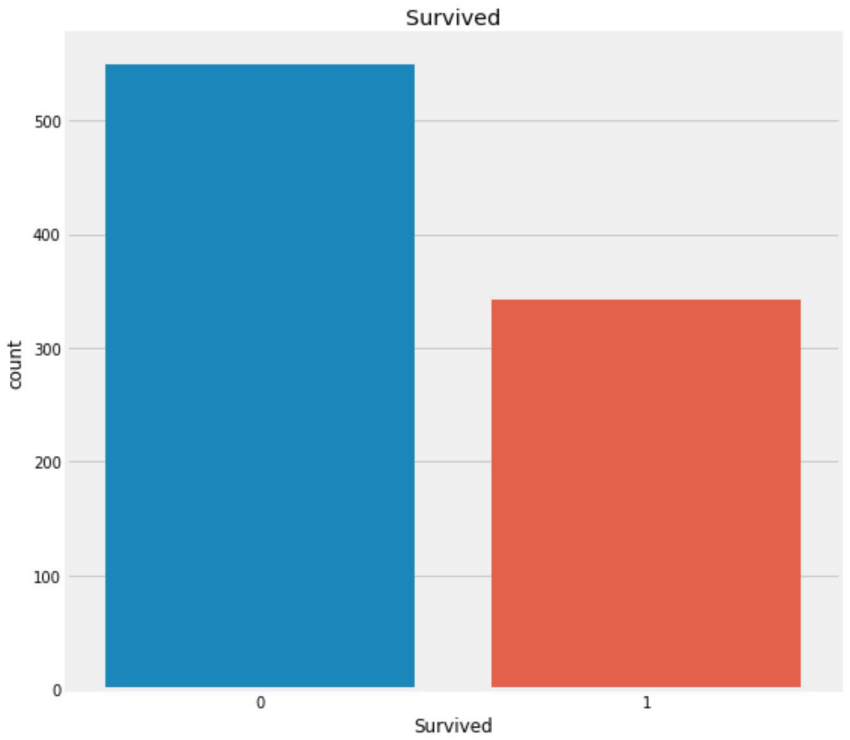
Out[4]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

The features **Age, Cabin and Embarked** have null values.

How many People Survived?

In [5]:
```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Survived')
ax[0].set_ylabel('')
sns.countplot('Survived',data=data,ax=ax[1])
ax[1].set_title('Survived')
plt.show()
```





It is evident that not many passengers survived the accident.

Out of the 891 passengers in the training set, only approximately 350 survived, which amounts to 38.4% of the total training set population who survived the crash. We need to delve deeper into the data to gain better insights and determine which categories of passengers survived and which did not.

We will attempt to assess the survival rate using various features of the dataset, such as Sex, Port of Embarkation, Age, etc.

First, let us understand the different types of features.

Types Of Features

Categorical Feature:

A categorical variable is one that comprises two or more categories, and each value within that feature can be categorized into one of these groups. For instance, gender is a categorical variable with two categories: male and female. Such variables cannot be sorted or

assigned any specific order, and they are commonly referred to as Nominal Variables.

Categorical features in the dataset: Sex, Embarked.

Ordinal Feature:

An ordinal variable is similar to categorical values, but what sets them apart is the presence of relative ordering or sorting between the values. For example, consider a feature like Height with values Tall, Medium, Short; in this case, Height is an ordinal variable. Here, we can establish a relative ranking within the variable.

Ordinal feature in the dataset: PClass

Continous Feature:

A feature is considered continuous if it can take values between any two points or within the range of the minimum and maximum values in the feature's column.

Continuous feature in the dataset: Age"

Analysing The Features

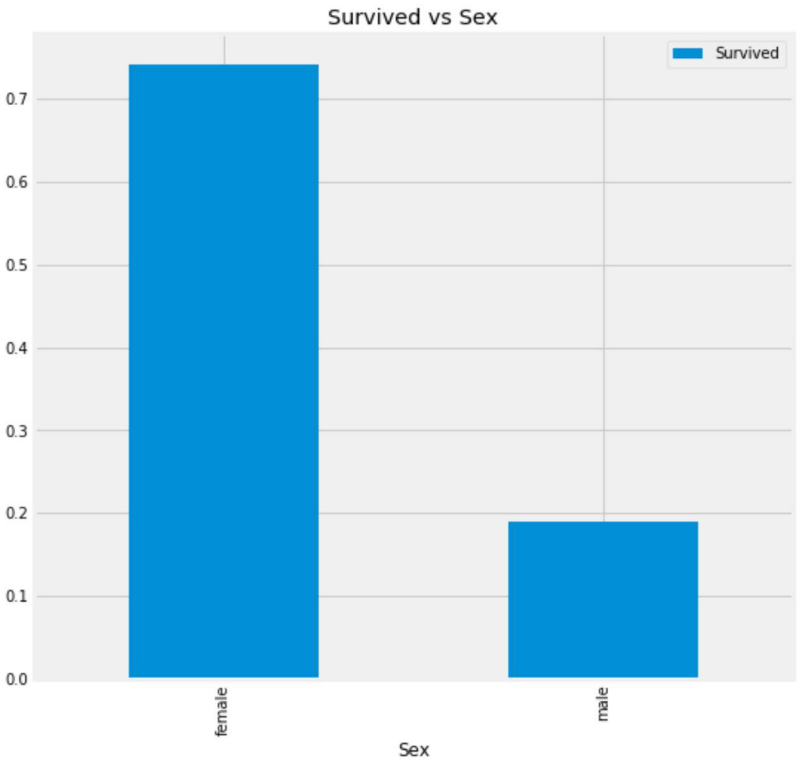
Sex--> Categorical Feature

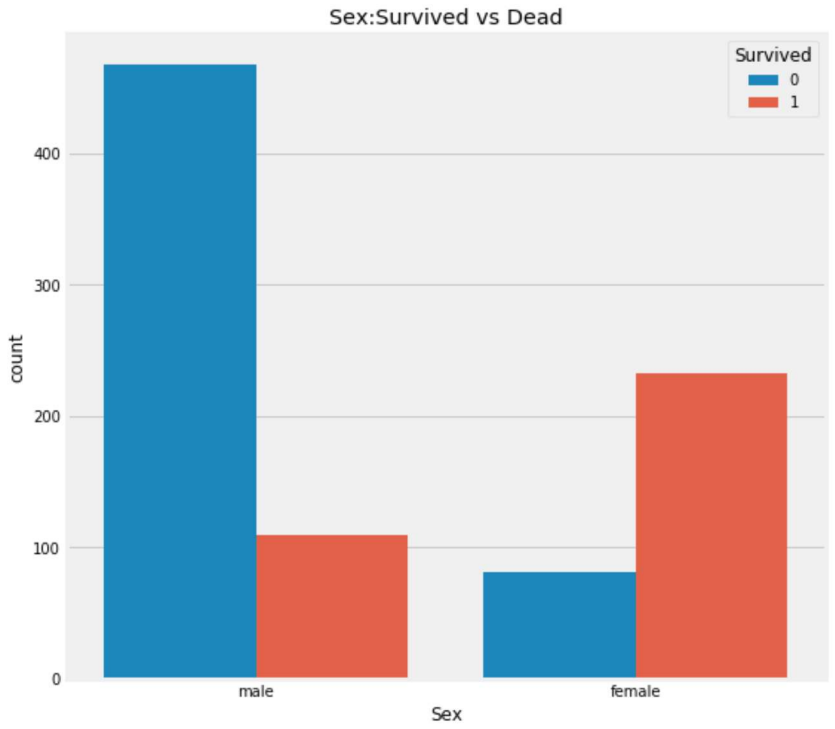
In [6]:
```
data.groupby(['Sex','Survived'])['Survived'].count()
```

Out[6]:
```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

In [7]:
```
f,ax=plt.subplots(1,2,figsize=(18,8))
data[['Sex','Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survived vs Sex')
sns.countplot('Sex',hue='Survived',data=data,ax=ax[1])
```

```
ax[1].set_title('Sex:Survived vs Dead')
plt.show()
```





The number of men on the ship is significantly greater than the number of women. However, the number of women saved is nearly twice the number of males saved. The survival rates for women on the ship are approximately 75%, whereas for men, it's around 18-19%.

This appears to be a highly significant feature for modeling.

Pclass --> Ordinal Feature

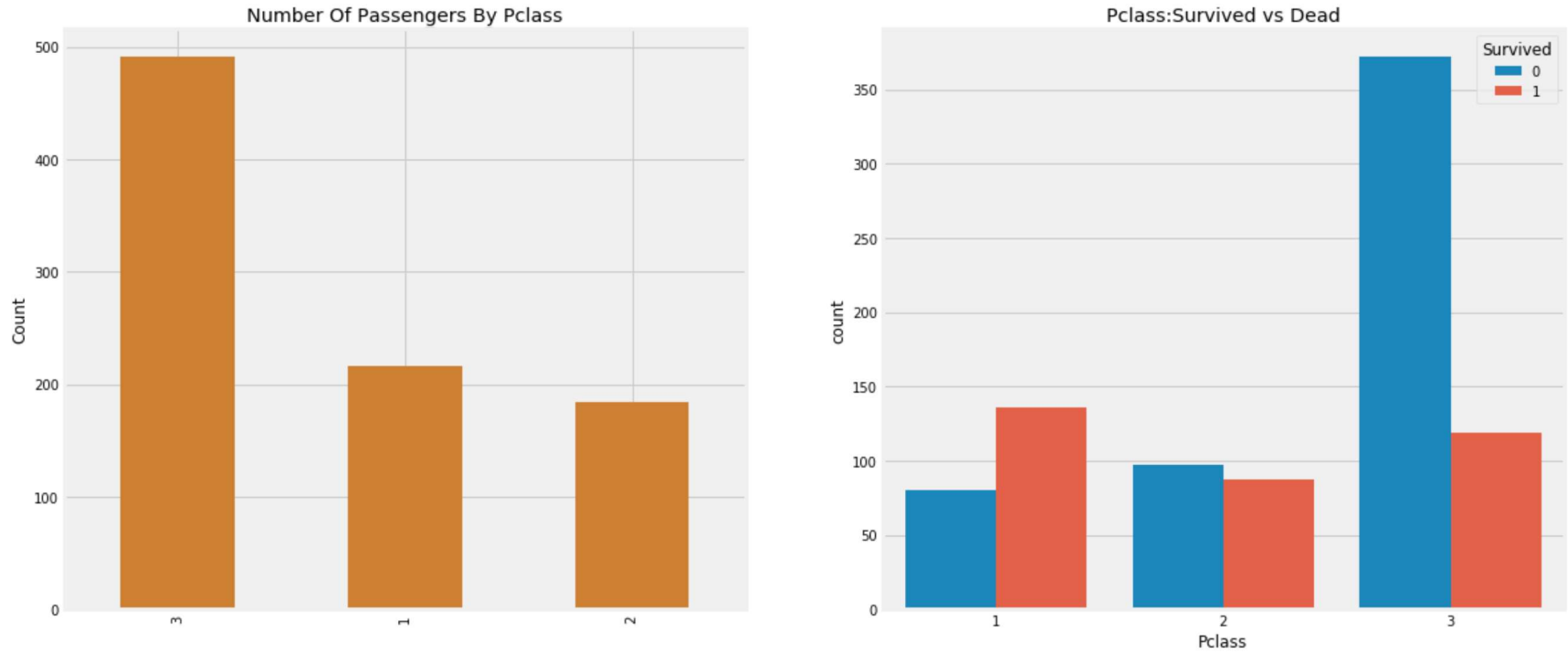
In [8]:
```
pd.crosstab(data.Pclass,data.Survived,margins=True).style.background_gradient(cmap='summer_r')
```

Out[8]:

Survived	0	1	All
Pclass			
1	80	136	216
2	97	87	184
3	372	119	491
All	549	342	891

In [9]:

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Pclass'].value_counts().plot.bar(color=['#CD7F32','#FFDF00','#D3D3D3'],ax=ax[0])
ax[0].set_title('Number Of Passengers By Pclass')
ax[0].set_ylabel('Count')
sns.countplot('Pclass',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Pclass:Survived vs Dead')
plt.show()
```



It is evident that passengers of Pclass 1 were given a significantly higher priority during the rescue. Even though the number of passengers in Pclass 3 was much larger, the survival rate for them is quite low, approximately 25%.

For Pclass 1, the survival rate is around 63%, while for Pclass 2, it's about 48%. This suggests that socioeconomic factors play a significant role. It's a reminder of the materialistic nature of the world.

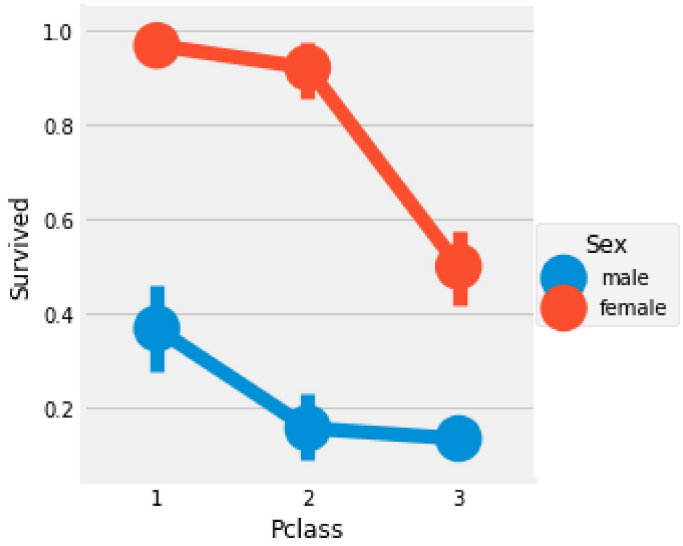
Let's delve deeper and explore other intriguing observations by examining the survival rate with Sex and Pclass together.

In [10]:
```
pd.crosstab([data.Sex,data.Survived],data.Pclass,margins=True).style.background_gradient(cmap='summer_r')
```

Out[10]:

Sex	Survived	Pclass 1	2	3	All
female	0	3	6	72	81
	1	91	70	72	233
male	0	77	91	300	468
	1	45	17	47	109
All		216	184	491	891

In [11]:
```
sns.factorplot('Pclass','Survived',hue='Sex',data=data)
plt.show()
```



We use FactorPlot in this case because it simplifies the separation of categorical values.

Looking at the CrossTab and the FactorPlot, we can readily infer that the survival rate for Women from Pclass 1 is approximately 95-96%, as only 3 out of 94 women from Pclass 1 perished.

It is evident that, regardless of Pclass, women were given the highest priority during the rescue. Even men from Pclass 1 have a significantly lower survival rate.

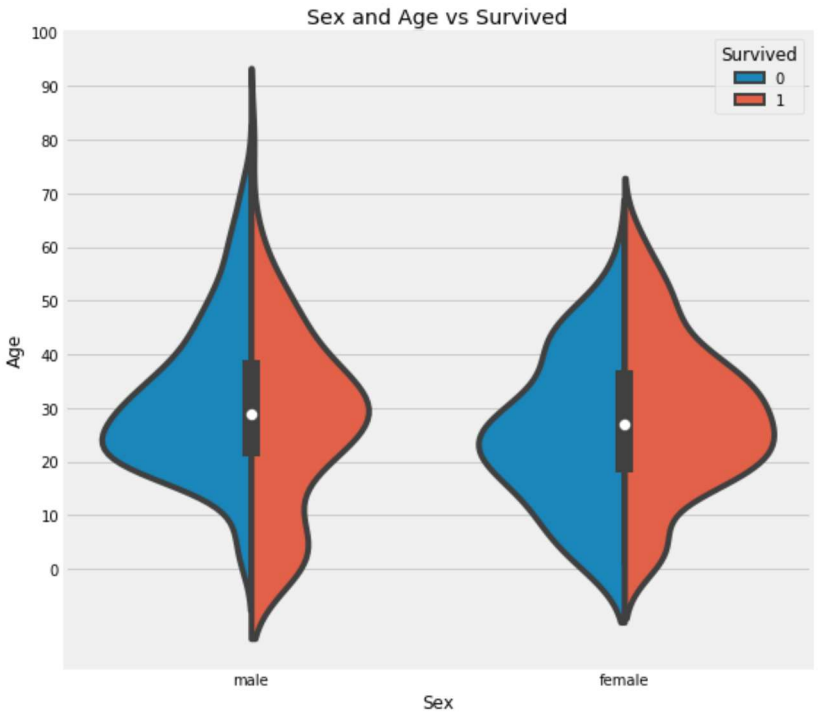
It appears that Pclass is also a crucial feature. Let's analyze other features.

Age--> Continous Feature

In [12]:
```
print('Oldest Passenger was of:',data['Age'].max(),'Years')
print('Youngest Passenger was of:',data['Age'].min(),'Years')
print('Average Age on the ship:',data['Age'].mean(),'Years')
```

```
Oldest Passenger was of: 80.0 Years
Youngest Passenger was of: 0.42 Years
Average Age on the ship: 29.69911764705882 Years
```

In [13]:
```
f,ax=plt.subplots(1,2,figsize=(18,8))
sns.violinplot("Pclass","Age", hue="Survived", data=data,split=True,ax=ax[0])
ax[0].set_title('Pclass and Age vs Survived')
ax[0].set_yticks(range(0,110,10))
sns.violinplot("Sex","Age", hue="Survived", data=data,split=True,ax=ax[1])
ax[1].set_title('Sex and Age vs Survived')
ax[1].set_yticks(range(0,110,10))
plt.show()
```



Observations:

1)The number of children increases with Pclass and the survival rate for passenegers below Age 10(i.e children) looks to be good irrespective of the Pclass.

2)Survival chances for Passenegers aged 20-50 from Pclass1 is high and is even better for Women.

3)For males, the survival chances decreases with an increase in age.

As we've previously observed, the Age feature contains 177 null values. To address these NaN values, we can impute them with the mean age of the dataset.

However, the challenge lies in the fact that passengers have various ages, and it wouldn't make sense to assign a 4-year-old child the mean age of 29 years. Is there a way to determine the age group to which the passenger belongs?

One approach is to examine the Name feature. Upon inspection, we notice that names include salutations like Mr. or Mrs. This allows us to assign the mean values of Mr. and Mrs. to their respective groups.

In [14]:
```
data['Initial']=0
for i in data:
    data['Initial']=data.Name.str.extract('([A-Za-z]+)\.') #lets extract the Salutations
```

We are using the regular expression (regex): [A-Za-z]+).. This regex searches for strings between A-Z or a-z followed by a (dot), effectively allowing us to extract the initials from the name.

In [15]:
```
pd.crosstab(data.Initial,data.Sex).T.style.background_gradient(cmap='summer_r') #Checking the Initials with the Sex
```

Out[15]:

Initial	Capt	Col	Countess	Don	Dr	Jonkheer	Lady	Major	Master	Miss	Mlle	Mme	Mr	Mrs	Ms	Rev	Sir
Sex																	
female	0	0	1	0	1	0	1	0	0	182	2	1	0	125	1	0	0
male	1	2	0	1	6	1	0	2	40	0	0	0	517	0	0	6	1

Some initials, like Mlle or Mme, are misspelled and actually stand for Miss. I will replace them accordingly and make similar corrections for other values.

In [16]:
```
data['Initial'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt','Sir','Don'],['
```

In [17]:
```
data.groupby('Initial')['Age'].mean() #lets check the average age by Initials
```

Out[17]:
```
Initial
Master     4.574167
Miss      21.860000
Mr        32.739609
Mrs       35.981818
Other     45.888889
Name: Age, dtype: float64
```

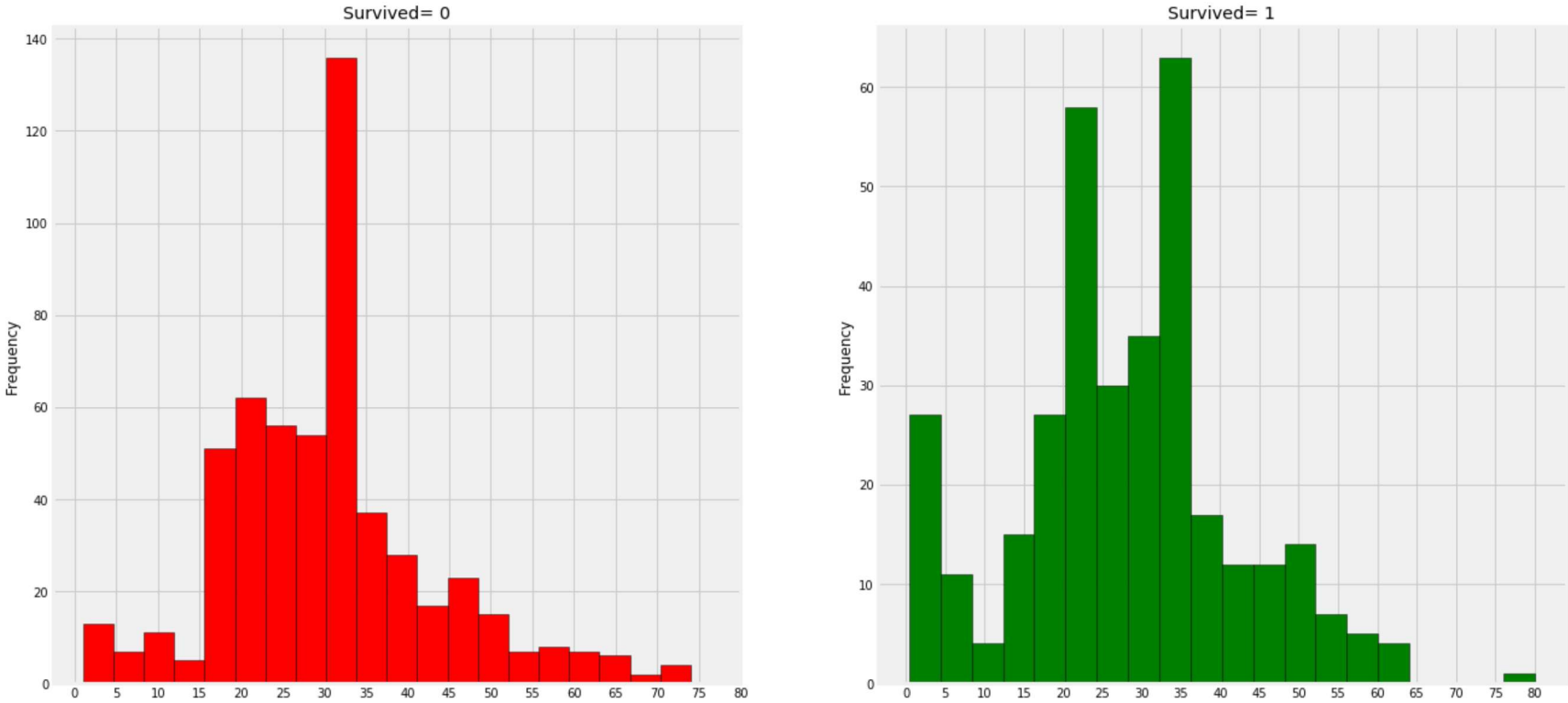
Filling NaN Ages

In [18]:
```
## Assigning the NaN Values with the Ceil values of the mean ages
data.loc[(data.Age.isnull())&(data.Initial=='Mr'),'Age']=33
data.loc[(data.Age.isnull())&(data.Initial=='Mrs'),'Age']=36
data.loc[(data.Age.isnull())&(data.Initial=='Master'),'Age']=5
data.loc[(data.Age.isnull())&(data.Initial=='Miss'),'Age']=22
data.loc[(data.Age.isnull())&(data.Initial=='Other'),'Age']=46
```

In [19]:
```
data.Age.isnull().any() #No null values left finally
```

Out[19]:
```
False
```

In [20]:
```
f,ax=plt.subplots(1,2,figsize=(20,10))
data[data['Survived']==0].Age.plot.hist(ax=ax[0],bins=20,edgecolor='black',color='red')
ax[0].set_title('Survived= 0')
x1=list(range(0,85,5))
ax[0].set_xticks(x1)
data[data['Survived']==1].Age.plot.hist(ax=ax[1],color='green',bins=20,edgecolor='black')
ax[1].set_title('Survived= 1')
x2=list(range(0,85,5))
ax[1].set_xticks(x2)
plt.show()
```



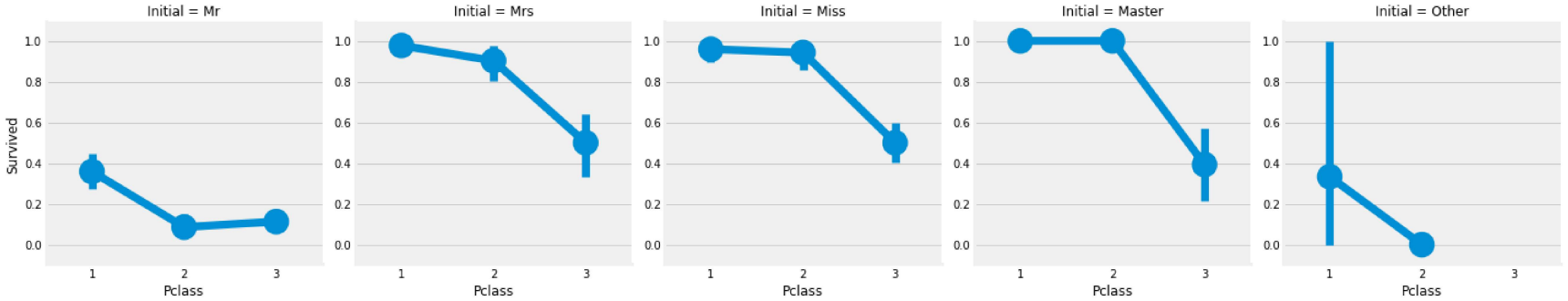
Observations:

Toddlers (age < 5) were saved in large numbers due to the "Women and Child First Policy."

The oldest passenger, aged 80, was saved.

The age group of 30-40 experienced the highest number of deaths.

In [21]:
```
sns.factorplot('Pclass','Survived',col='Initial',data=data)
plt.show()
```



The Women and Child First policy holds true regardless of class.

Embarked--> Categorical Value

In [22]:
```
pd.crosstab([data.Embarked,data.Pclass],[data.Sex,data.Survived],margins=True).style.background_gradient(cmap='summer_r
```

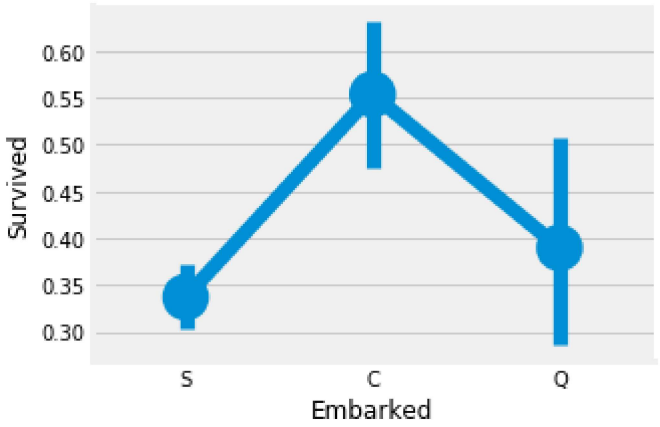
Out[22]:

	Sex	female		male		All
	Survived	0	1	0	1	
Embarked	Pclass					
C	1	1	42	25	17	85
	2	0	7	8	2	17
	3	8	15	33	10	66
Q	1	0	1	1	0	2
	2	0	2	1	0	3
	3	9	24	36	3	72
S	1	2	46	51	28	127
	2	6	61	82	15	164
	3	55	33	231	34	353
All		81	231	468	109	889

Chances for Survival by Port Of Embarkation

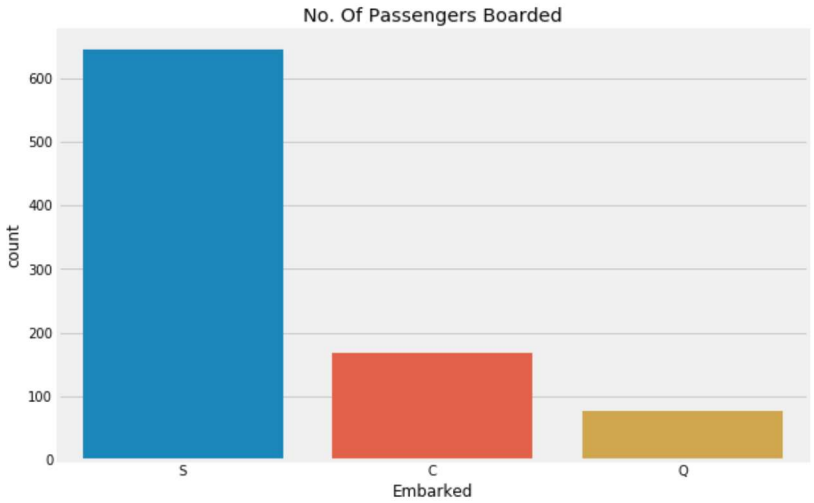
In [23]:

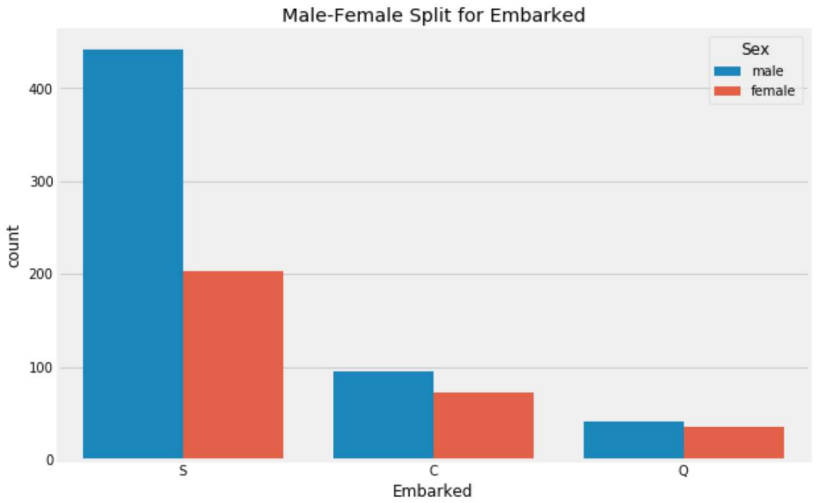
```
sns.factorplot('Embarked','Survived',data=data)
fig=plt.gcf()
fig.set_size_inches(5,3)
plt.show()
```

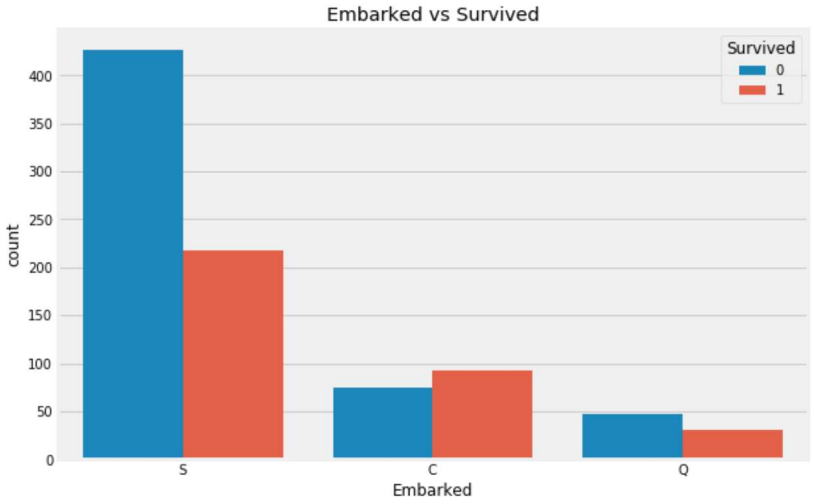


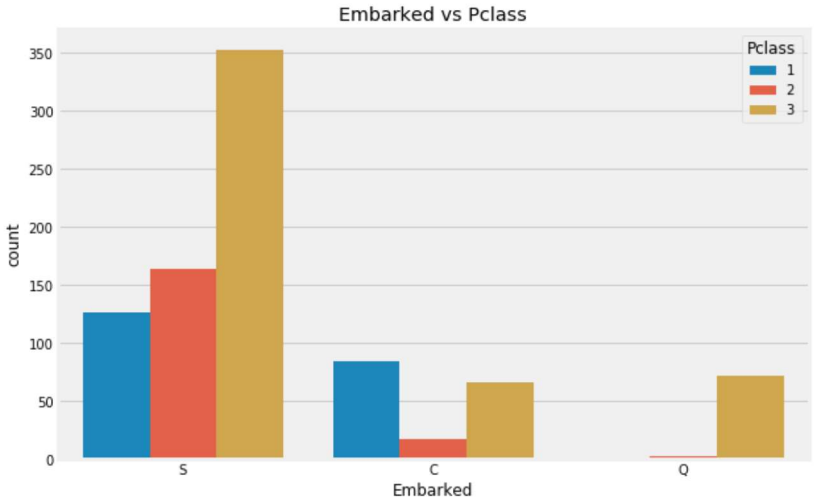
The highest survival rate is in Port C at approximately 0.55, while the lowest is in S.

In [24]:
```
f,ax=plt.subplots(2,2,figsize=(20,15))
sns.countplot('Embarked',data=data,ax=ax[0,0])
ax[0,0].set_title('No. Of Passengers Boarded')
sns.countplot('Embarked',hue='Sex',data=data,ax=ax[0,1])
ax[0,1].set_title('Male-Female Split for Embarked')
sns.countplot('Embarked',hue='Survived',data=data,ax=ax[1,0])
ax[1,0].set_title('Embarked vs Survived')
sns.countplot('Embarked',hue='Pclass',data=data,ax=ax[1,1])
ax[1,1].set_title('Embarked vs Pclass')
plt.subplots_adjust(wspace=0.2,hspace=0.5)
plt.show()
```









Observations:

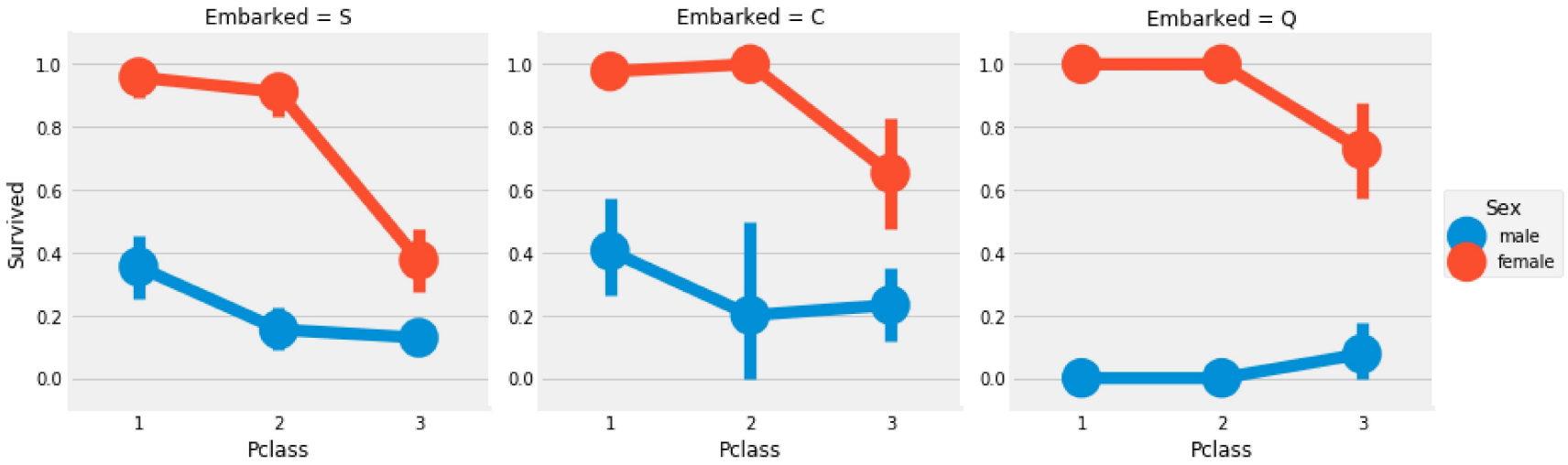
The majority of passengers boarded from S, with most of them belonging to Pclass 3.

Passengers from C had a higher survival rate, possibly due to the rescue of all Pclass 1 and Pclass 2 passengers.

Despite being the port where many rich passengers boarded, the chances of survival from Port S are low. This is because approximately 81% of Pclass 3 passengers did not survive.

Port Q had almost 95% of passengers from Pclass 3.

In [25]:
```
sns.factorplot('Pclass','Survived',hue='Sex',col='Embarked',data=data)
plt.show()
```



Observations:

Women in Pclass 1 and Pclass 2 have nearly a 100% survival rate, regardless of the Pclass.

Port S appears to be unfortunate for Pclass 3 passengers, as the survival rate is low for both men and women. (Money Matters)

Port Q seems to be the least fortunate for men, as almost all of them were from Pclass 3.

Filling Embarked NaN

Since the majority of passengers boarded from Port S, we will replace the NaN values with S.

In [26]:
```
data['Embarked'].fillna('S',inplace=True)
```

In [27]:
```
data.Embarked.isnull().any()# Finally No NaN values
```

Out[27]: False

SibSip-->Discrete Feature

This feature represents whether a person is alone or with his family members.

Sibling = brother, sister, stepbrother, stepsister

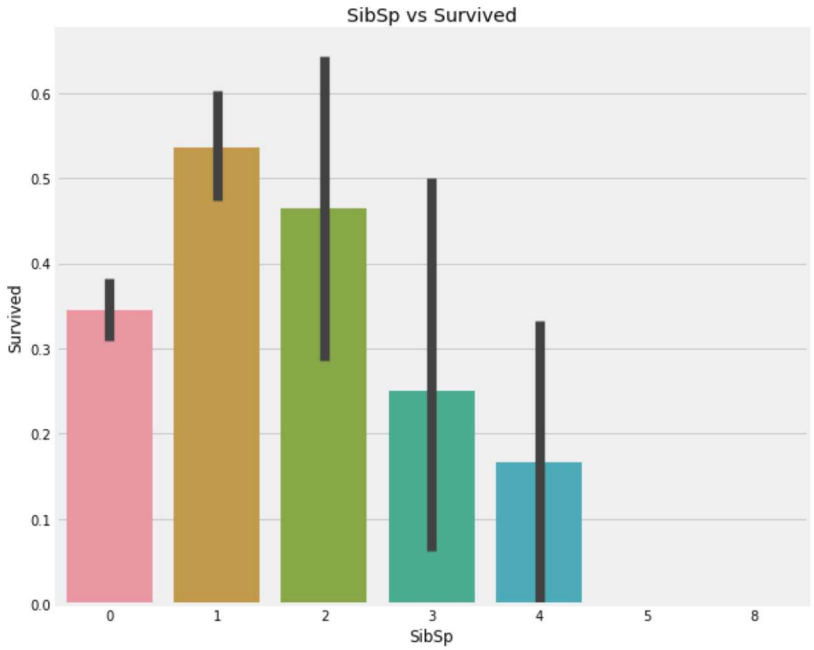
Spouse = husband, wife

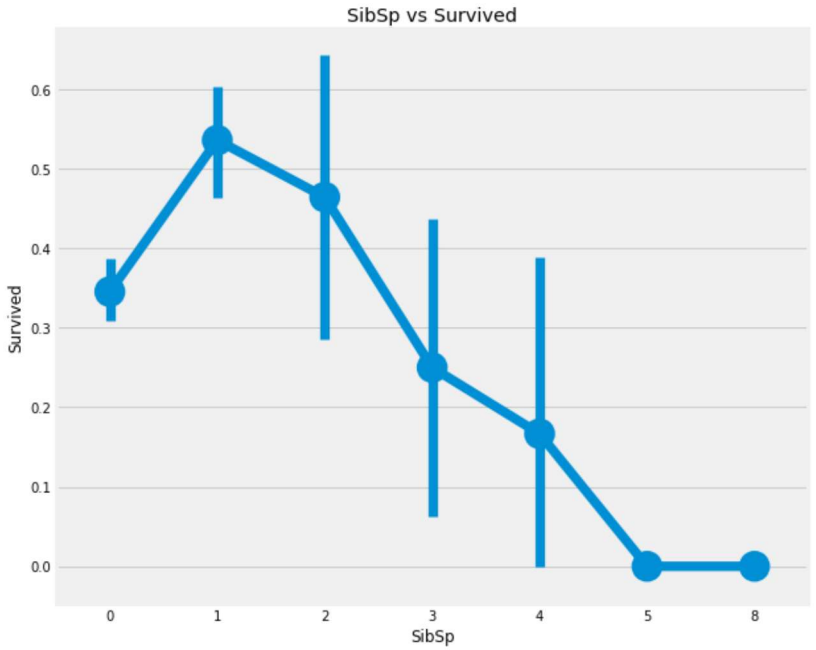
In [28]:
```
pd.crosstab([data.SibSp],data.Survived).style.background_gradient(cmap='summer_r')
```

Out[28]:

Survived	0	1
SibSp		
0	398	210
1	97	112
2	15	13
3	12	4
4	15	3
5	5	0
8	7	0

In [29]:
```
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot('SibSp','Survived',data=data,ax=ax[0])
ax[0].set_title('SibSp vs Survived')
sns.factorplot('SibSp','Survived',data=data,ax=ax[1])
ax[1].set_title('SibSp vs Survived')
plt.close(2)
plt.show()
```





In [30]:
```
pd.crosstab(data.SibSp,data.Pclass).style.background_gradient(cmap='summer_r')
```

Out[30]:

SibSp	Pclass 1	2	3
0	137	120	351
1	71	55	83
2	5	8	15
3	3	1	12
4	0	0	18
5	0	0	5
8	0	0	7

Observations:

The barplot and factorplot illustrate that if a passenger is alone onboard with no siblings, they have a 34.5% survival rate. This rate decreases as the number of siblings increases, which is logical—people may prioritize saving their family members over themselves.

Interestingly, the survival rate for families with 5-8 members is 0%. This might be due to the influence of Pclass.

Indeed, the reason is Pclass. The crosstab reveals that individuals with SibSp>3 were all in Pclass 3. It's evident that all the large families in Pclass 3 (>3) did not survive.

Parch

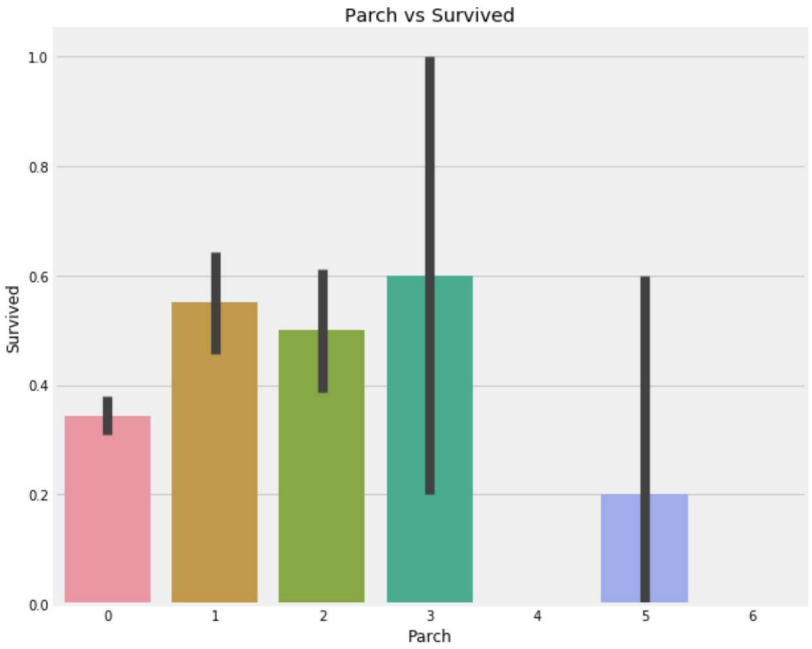
In [31]:
```
pd.crosstab(data.Parch,data.Pclass).style.background_gradient(cmap='summer_r')
```

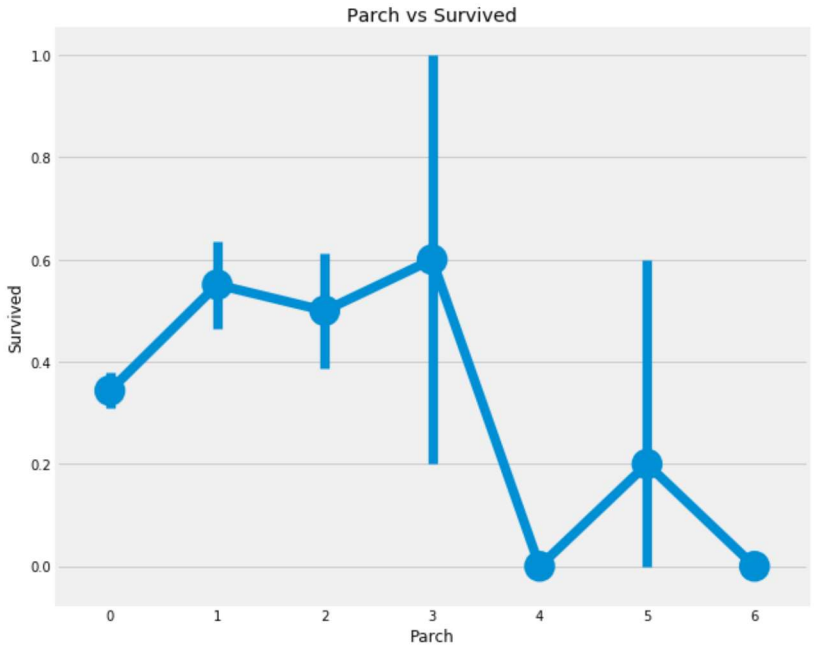
Out[31]:

Pclass	1	2	3
Parch			
0	163	134	381
1	31	32	55
2	21	16	43
3	0	2	3
4	1	0	3
5	0	0	5
6	0	0	1

The crosstab again shows that larger families were in Pclass3.

In [32]:
```
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot('Parch','Survived',data=data,ax=ax[0])
ax[0].set_title('Parch vs Survived')
sns.factorplot('Parch','Survived',data=data,ax=ax[1])
ax[1].set_title('Parch vs Survived')
plt.close(2)
plt.show()
```





Observations:

Similarly, the results show that passengers with their parents onboard have a greater chance of survival, which gradually decreases as the number of parents goes up. The likelihood of survival is higher for those with 1-3 parents on the ship, but being alone also proves to be fatal. The chances of survival decrease when somebody has more than 4 parents on the ship.

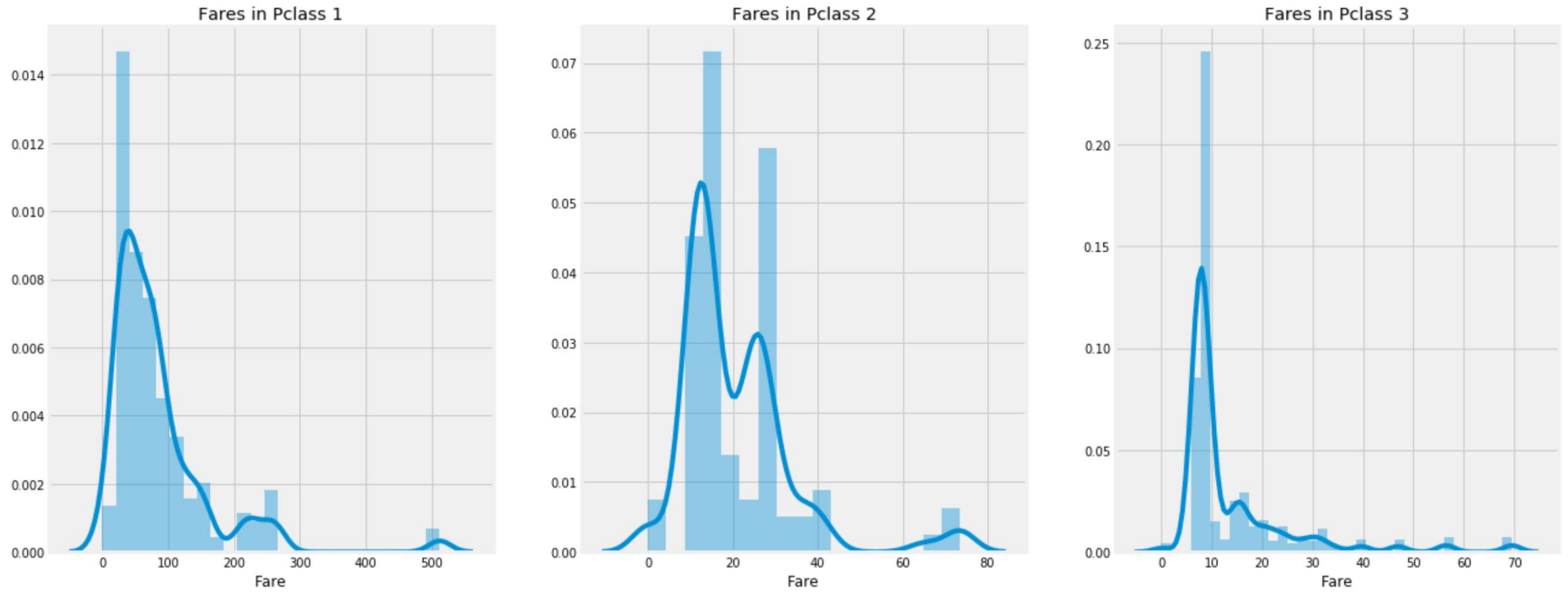
Fare--> Continous Feature

In [33]:
```
print('Highest Fare was:',data['Fare'].max())
print('Lowest Fare was:',data['Fare'].min())
print('Average Fare was:',data['Fare'].mean())
```

```
Highest Fare was: 512.3292
Lowest Fare was: 0.0
Average Fare was: 32.2042079685746
```

In [34]:
```
f,ax=plt.subplots(1,3,figsize=(20,8))
sns.distplot(data[data['Pclass']==1].Fare,ax=ax[0])
ax[0].set_title('Fares in Pclass 1')
sns.distplot(data[data['Pclass']==2].Fare,ax=ax[1])
```

```
ax[1].set_title('Fares in Pclass 2')
sns.distplot(data[data['Pclass']==3].Fare,ax=ax[2])
ax[2].set_title('Fares in Pclass 3')
plt.show()
```



There is a significant fare distribution among passengers in Pclass 1, and this distribution decreases as the class standards decline. Since this is a continuous variable, we can convert it into discrete values through binning.

Observations in a Nutshell for all features:

Sex: The chance of survival is higher for women compared to men.

Pclass: There is a noticeable trend where being a 1st class passenger increases your chances of survival. Conversely, the survival rate for Pclass3 is quite low. For women, the probability of survival in Pclass1 is almost 1, and it is also high for those in Pclass2.

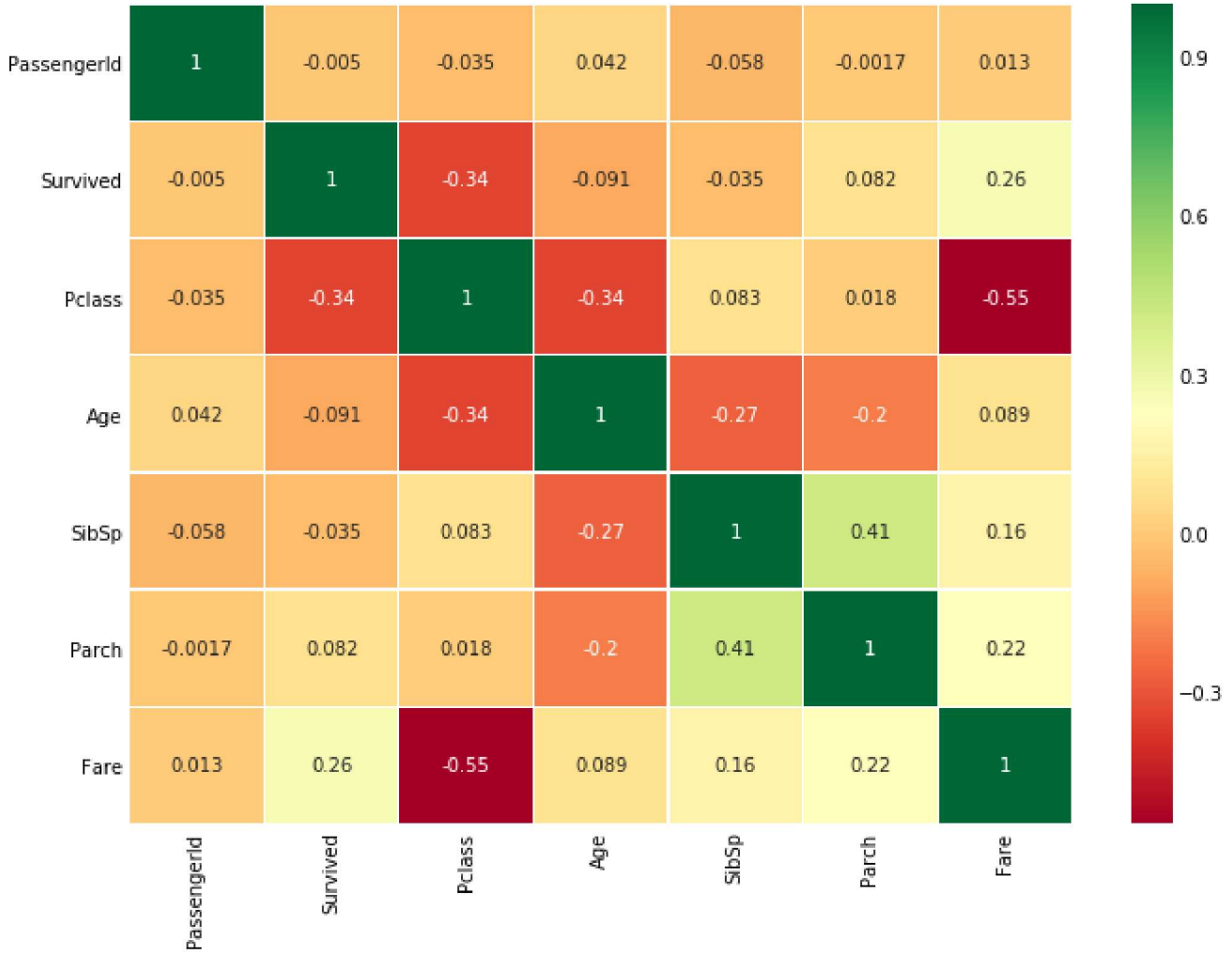
Age: Children under 5-10 years have a high chance of survival, while passengers in the age group 15 to 35 experienced higher mortality.

Embarked: This feature is particularly interesting. The chances of survival in C appear better, despite the majority of Pclass1 passengers boarding at S. Passengers at Q were all from Pclass3.

Parch+SibSp: Having 1-2 siblings or a spouse on board, or 1-3 parents, shows a higher probability of survival compared to being alone or traveling with a large family.

Correlation Between The Features

In [35]:
```
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-->correlation matrix
fig=plt.gcf()
fig.set_size_inches(10,8)
plt.show()
```



Interpreting The Heatmap

First, it's essential to note that we only compare numeric features because we cannot correlate alphabets or strings. Before diving into the plot, let's understand what correlation means.

POSITIVE CORRELATION: When an increase in feature A leads to an increase in feature B, they are positively correlated. A value of 1 indicates a perfect positive correlation.

NEGATIVE CORRELATION: When an increase in feature A leads to a decrease in feature B, they are negatively correlated. A value of -1 indicates a perfect negative correlation.

If two features are highly or perfectly correlated, it means that they contain almost the same information, leading to multicollinearity. In such cases, using both features is redundant. When building or training models, it's a good practice to eliminate redundant features, as it reduces training time and offers other advantages.

Now, looking at the heatmap, we can observe that the features are not strongly correlated. The highest correlation is 0.41 between SibSp and Parch. Therefore, it seems reasonable to proceed with all the features.

Part2: Feature Engineering and Data Cleaning

Whenever we are given a dataset with features, it is not necessary that all the features will be important. There may be many redundant features that should be eliminated. Additionally, we can acquire or add new features by observing or extracting information from other features.

An example of this would be obtaining the 'Initials' feature using the 'Name' feature. Let's explore if we can discover new features and eliminate unnecessary ones. We will also transform the existing relevant features into a suitable form for predictive modeling.

Age_band

Problem With Age Feature:

As I mentioned earlier, Age is a continuous feature, which poses a challenge in Machine Learning Models.

Example: If I ask you to group or categorize sportspeople by Sex, you can easily separate them into Male and Female.

Now, if I ask you to group them by their Age, how would you do it? With 30 persons, you could have 30 different age values, which is problematic.

To address this issue, we need to convert these continuous values into categorical values through either binning or normalization. I will be using binning, which involves grouping a range of ages into a single bin or assigning them a single value.

So, the maximum age of a passenger was 80. Let's divide the range from 0 to 80 into 5 bins. This results in 80/5 = 16, so we'll create bins of size 16.

In [36]:
```
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
data.head(2)
```

Out[36]:

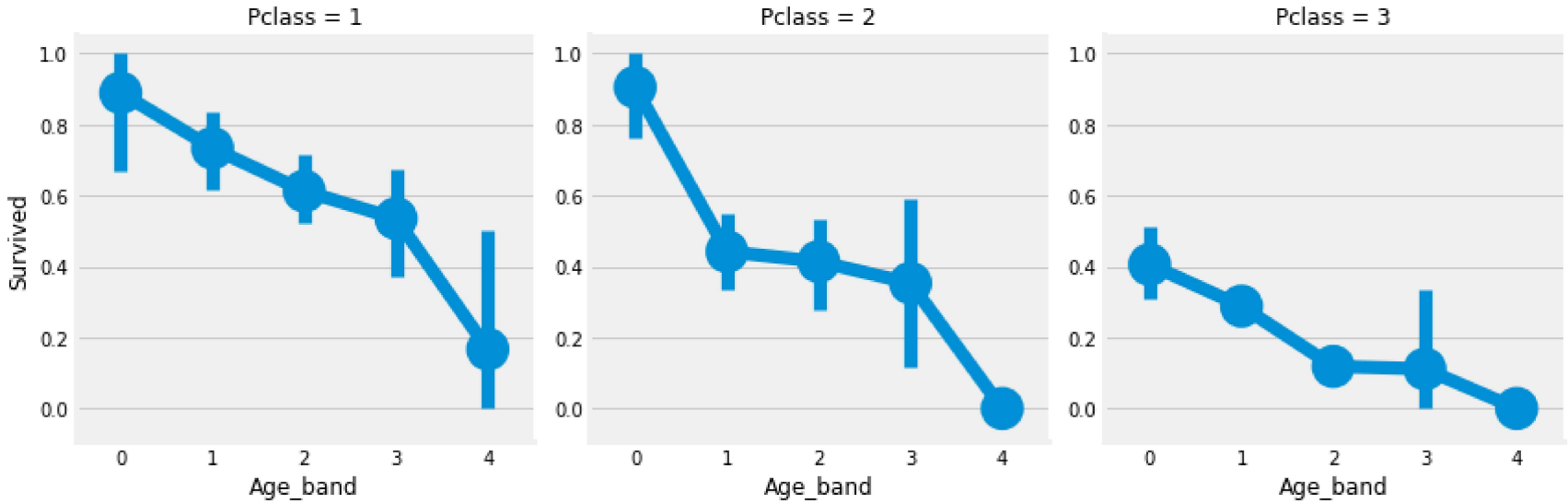
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Initial	Age_band
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	Mr	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs	2

In [37]:
```
data['Age_band'].value_counts().to_frame().style.background_gradient(cmap='summer')#checking the number of passenegers
```

Out[37]:

	Age_band
1	382
2	325
0	104
3	69
4	11

In [38]:
```
sns.factorplot('Age_band','Survived',data=data,col='Pclass')
plt.show()
```

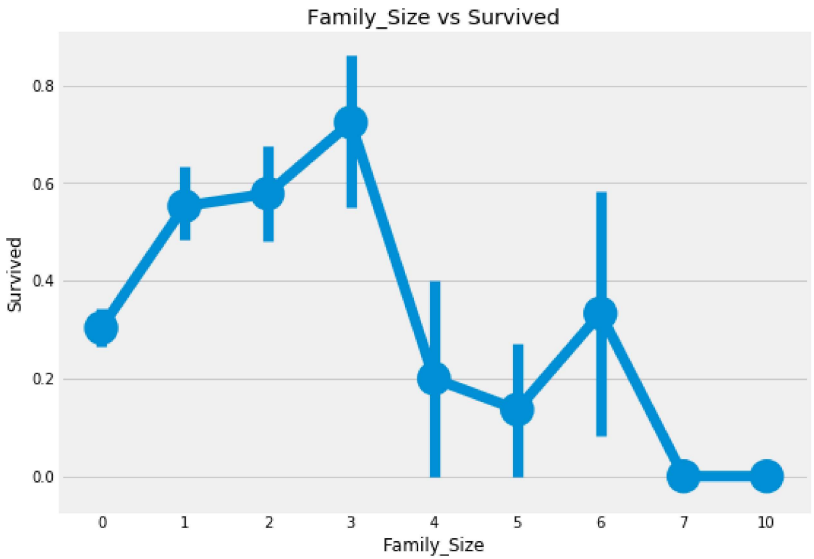


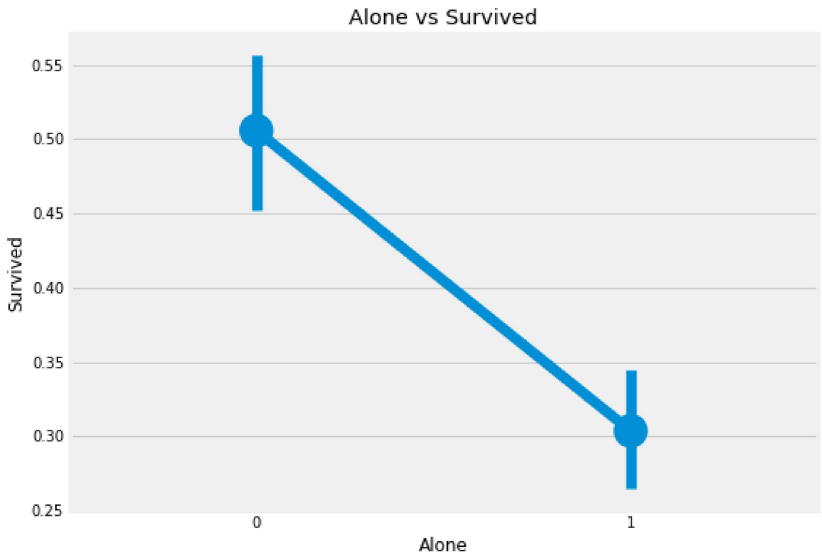
Survival rates decrease with increasing age, regardless of the passenger's class.

Family_Size and Alone

At this point, we can create two new features: 'Family_size' and 'Alone' for analysis. 'Family_size' is the sum of Parch and SibSp, providing combined data to assess whether the survival rate correlates with the passengers' family size. 'Alone' will indicate whether a passenger is traveling alone or not.

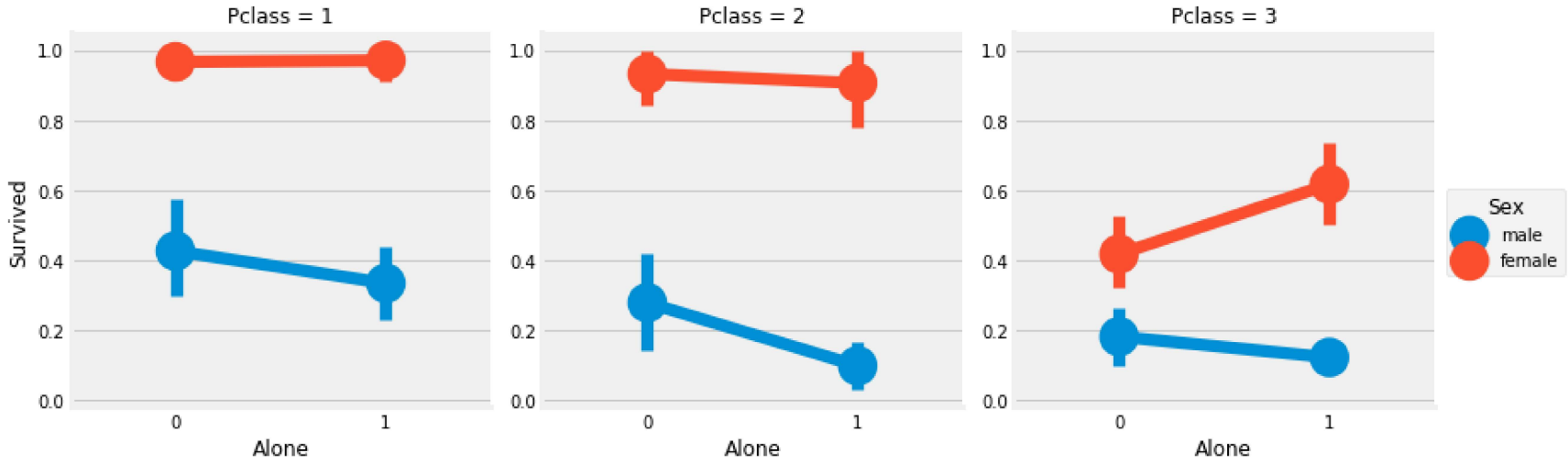
In [39]:
```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']#family size
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1#Alone

f,ax=plt.subplots(1,2,figsize=(18,6))
sns.factorplot('Family_Size','Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.factorplot('Alone','Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```





If 'Family_Size' equals 0, it means that the passenger is traveling alone. Evidently, if you are alone or have a 'Family_Size' of 0, the chances of survival are very low. Additionally, for 'Family_Size' greater than 4, the chances of survival decrease as well. This also appears to be an important feature for the model. Let's explore this further.

In [40]:
```
sns.factorplot('Alone','Survived',data=data,hue='Sex',col='Pclass')
plt.show()
```



It's evident that being alone is disadvantageous, regardless of sex or Pclass, except for Pclass 3, where the chances of survival for females who are alone are higher than those with family.

Fare_Range

Since fare is also a continuous feature, we need to convert it into an ordinal value. For this, we will use pandas.qcut.

qcut splits or arranges the values according to the number of bins we specify. If we choose 5 bins, it will evenly distribute the values into 5 separate bins or value ranges.

In [41]:
```
data['Fare_Range']=pd.qcut(data['Fare'],4)
data.groupby(['Fare_Range'])['Survived'].mean().to_frame().style.background_gradient(cmap='summer_r')
```

Out[41]:

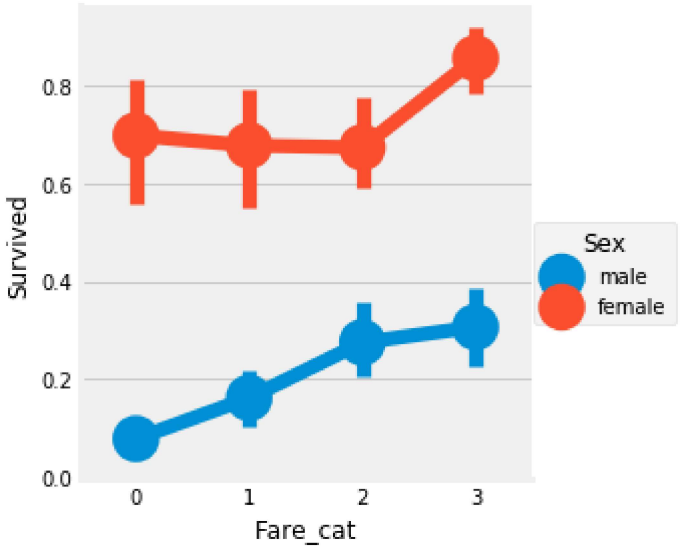
Fare_Range	Survived
(-0.001, 7.91]	0.197309
(7.91, 14.454]	0.303571
(14.454, 31.0]	0.454955
(31.0, 512.329]	0.581081

As discussed earlier, we can observe that as the fare_range increases, the chances of survival also increase.

Now, we cannot use the Fare_Range values as they are. We should convert them into single values, just as we did with Age_Band.

In [42]:
```
data['Fare_cat']=0
data.loc[data['Fare']<=7.91,'Fare_cat']=0
data.loc[(data['Fare']>7.91)&(data['Fare']<=14.454),'Fare_cat']=1
data.loc[(data['Fare']>14.454)&(data['Fare']<=31),'Fare_cat']=2
data.loc[(data['Fare']>31)&(data['Fare']<=513),'Fare_cat']=3
```

In [43]:
```
sns.factorplot('Fare_cat','Survived',data=data,hue='Sex')
plt.show()
```



Clearly, as the 'Fare_cat' increases, the chances of survival increase. This feature may become important during modeling, along with 'Sex.

Converting String Values into Numeric

Since we cannot pass strings to a machine learning model, we need to convert features like 'Sex,' 'Embarked,' etc. into numeric values.

In [44]:
```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
```

Dropping UnNeeded Features

Name: We don't need the 'Name' feature as it cannot be converted into a categorical value.

Age: Since we have the 'Age_band' feature, there's no need for this.

Ticket: This is a random string that cannot be categorized.

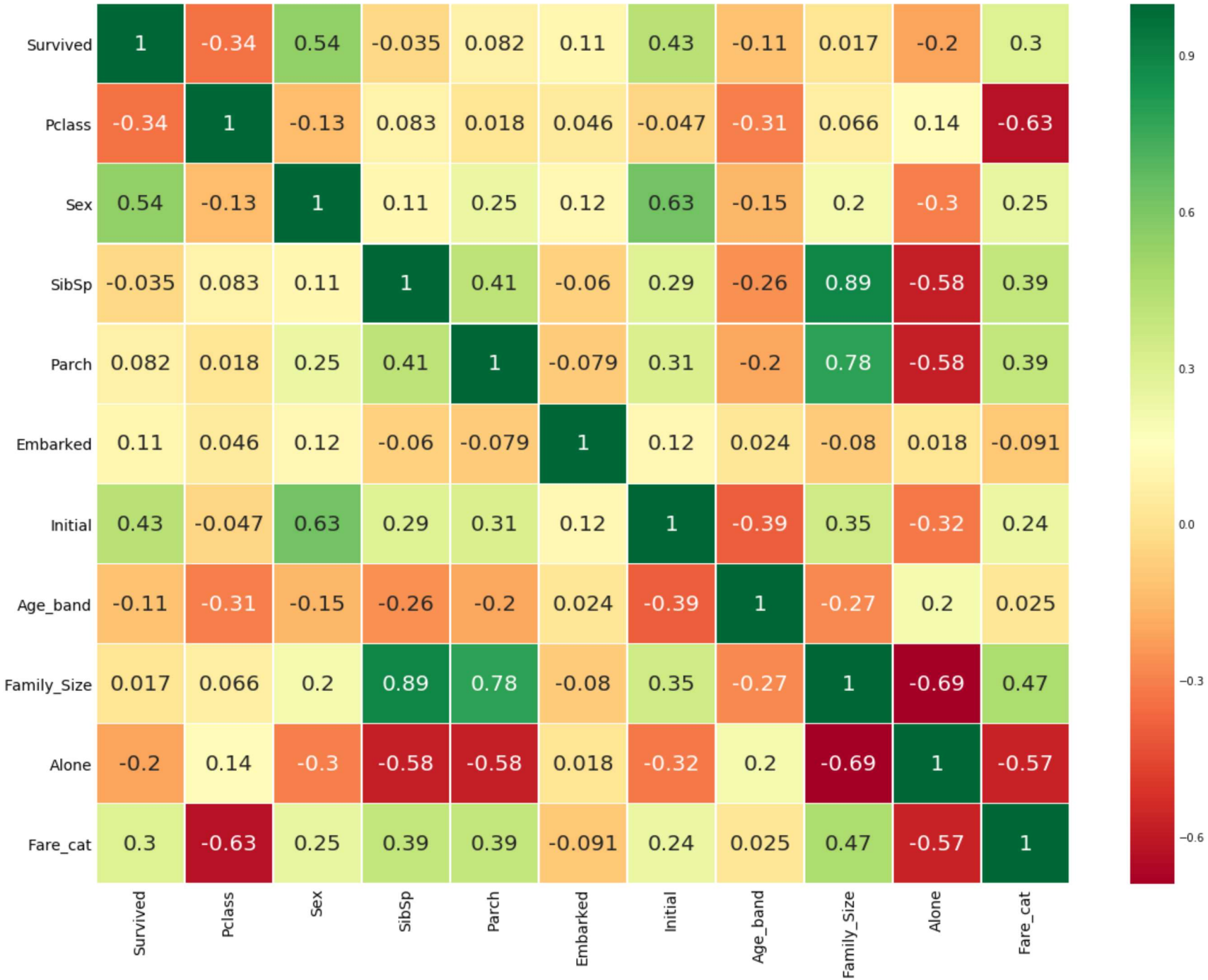
Fare: We have the 'Fare_cat' feature, so it's not needed.

Cabin: It has many NaN values, and many passengers have multiple cabins, making it a useless feature.

Fare_Range: We have the 'fare_cat' feature.

PassengerId: It cannot be categorized.

In [45]:
```
data.drop(['Name','Age','Ticket','Fare','Cabin','Fare_Range','PassengerId'],axis=1,inplace=True)
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':20})
fig=plt.gcf()
fig.set_size_inches(18,15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```


Survived 1 -0.34 0.54 -0.035 0.082 0.11 0.43 -0.11 0.017 -0.2 0.3
Pclass -0.34 1 -0.13 0.083 0.018 0.046 -0.047 -0.31 0.066 0.14 -0.63
Sex 0.54 -0.13 1 0.11 0.25 0.12 0.63 -0.15 0.2 -0.3 0.25
SibSp -0.035 0.083 0.11 1 0.41 -0.06 0.29 -0.26 0.89 -0.58 0.39
Parch 0.082 0.018 0.25 0.41 1 -0.079 0.31 -0.2 0.78 -0.58 0.39
Embarked 0.11 0.046 0.12 -0.06 -0.079 1 0.12 0.024 -0.08 0.018 -0.091
Initial 0.43 -0.047 0.63 0.29 0.31 0.12 1 -0.39 0.35 -0.32 0.24
Age_band -0.11 -0.31 -0.15 -0.26 -0.2 0.024 -0.39 1 -0.27 0.2 0.025
Family_Size 0.017 0.066 0.2 0.89 0.78 -0.08 0.35 -0.27 1 -0.69 0.47
Alone -0.2 0.14 -0.3 -0.58 -0.58 0.018 -0.32 0.2 -0.69 1 -0.57
Fare_cat 0.3 -0.63 0.25 0.39 0.39 -0.091 0.24 0.025 0.47 -0.57 1
Survived
Pclass
Sex
SibSp
Parch
Embarked
Initial
Age_band
Family_Size
Alone
Fare_cat
0.9
0.6
0.3
0.0
−0.3
−0.6

Now, looking at the correlation plot above, we can see some positively related features. For example, SibSp and Family_Size and Parch and Family_Size, as well as some negatively related ones like Alone and Family_Size.

Part3: Predictive Modeling

We have gained some insights from the EDA part, but these insights alone cannot accurately predict whether a passenger will survive or not. To make the prediction, I will use various classification algorithms. The following are the algorithms I plan to use to build the model:

1)Logistic Regression

2)Support Vector Machines(Linear and radial)

3)Random Forest

4)K-Nearest Neighbours

5)Naive Bayes

6)Decision Tree

In [46]:
```
#importing all the required ML packages
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

In [47]:
```
train,test=train_test_split(data,test_size=0.3,random_state=0,stratify=data['Survived'])
train_X=train[train.columns[1:]]
train_Y=train[train.columns[:1]]
test_X=test[test.columns[1:]]
test_Y=test[test.columns[:1]]
```

```
X=data[data.columns[1:]]
Y=data['Survived']
```

Radial Support Vector Machines(rbf-SVM)

In [48]:
```
model=svm.SVC(kernel='rbf',C=1,gamma=0.1)
model.fit(train_X,train_Y)
prediction1=model.predict(test_X)
print('Accuracy for rbf SVM is ',metrics.accuracy_score(prediction1,test_Y))
```

```
Accuracy for rbf SVM is  0.835820895522
```

Linear Support Vector Machine(linear-SVM)

In [49]:
```
model=svm.SVC(kernel='linear',C=0.1,gamma=0.1)
model.fit(train_X,train_Y)
prediction2=model.predict(test_X)
print('Accuracy for linear SVM is',metrics.accuracy_score(prediction2,test_Y))
```

```
Accuracy for linear SVM is 0.817164179104
```

Logistic Regression

In [50]:
```
model = LogisticRegression()
model.fit(train_X,train_Y)
prediction3=model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction3,test_Y))
```

```
The accuracy of the Logistic Regression is 0.817164179104
```

Decision Tree

In [51]:
```
model=DecisionTreeClassifier()
model.fit(train_X,train_Y)
prediction4=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction4,test_Y))
```

```
The accuracy of the Decision Tree is 0.805970149254
```

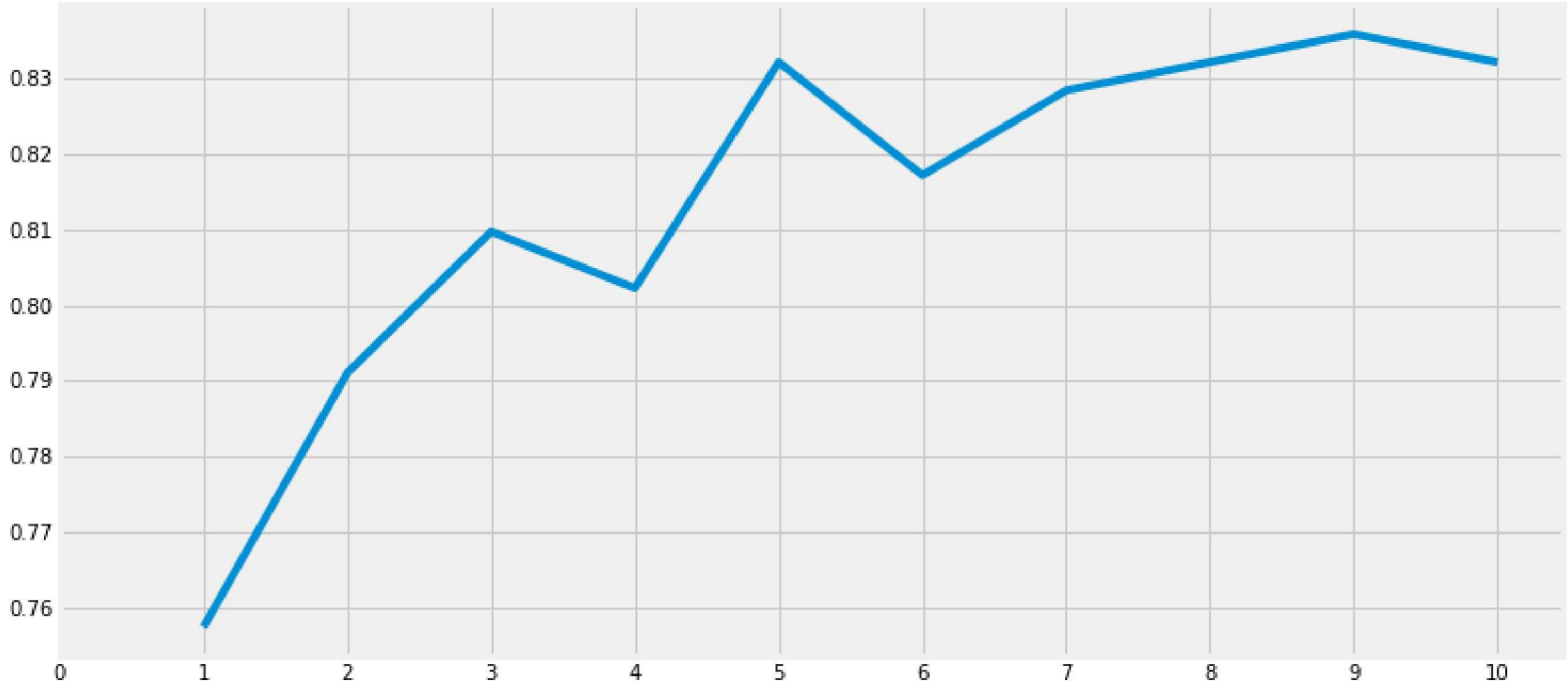
K-Nearest Neighbours(KNN)

In [52]:
```
model=KNeighborsClassifier()
model.fit(train_X,train_Y)
prediction5=model.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction5,test_Y))
```

```
The accuracy of the KNN is 0.832089552239
```

Now, the accuracy of the KNN model changes as we adjust the values for the n_neighbors attribute. The default value is 5. Let's assess the accuracies for various values of n_neighbors.

In [53]:
```
a_index=list(range(1,11))
a=pd.Series()
x=[0,1,2,3,4,5,6,7,8,9,10]
for i in list(range(1,11)):
    model=KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X,train_Y)
    prediction=model.predict(test_X)
    a=a.append(pd.Series(metrics.accuracy_score(prediction,test_Y)))
plt.plot(a_index, a)
plt.xticks(x)
fig=plt.gcf()
fig.set_size_inches(12,6)
plt.show()
print('Accuracies for different values of n are:',a.values,'with the max value as ',a.values.max())
```




```
Accuracies for different values of n are: [ 0.75746269  0.79104478  0.80970149  0.80223881  0.83208955  0.81716418
  0.82835821  0.83208955  0.8358209   0.83208955] with the max value as  0.835820895522
```

Gaussian Naive Bayes

In [54]:
```
model=GaussianNB()
model.fit(train_X,train_Y)
prediction6=model.predict(test_X)
print('The accuracy of the NaiveBayes is',metrics.accuracy_score(prediction6,test_Y))
```

```
The accuracy of the NaiveBayes is 0.813432835821
```

Random Forests

In [55]:
```
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_Y)
prediction7=model.predict(test_X)
print('The accuracy of the Random Forests is',metrics.accuracy_score(prediction7,test_Y))
```

```
The accuracy of the Random Forests is 0.813432835821
```

The accuracy of a model is not the sole factor determining the classifier's robustness. For instance, if a classifier is trained on one dataset and achieves a 90% accuracy, it doesn't guarantee that it will consistently achieve 90% accuracy on all new test sets. The accuracy may vary because we can't control which instances the classifier uses for training. This variability is known as model variance.

To address this and build a more generalized model, we use Cross Validation.

Cross Validation

Many times, data is imbalanced, meaning there may be a high number of instances in one class and fewer instances in another class. To address this, we should train and test our algorithm on each instance of the dataset and take an average of the accuracies.

Here's how K-Fold Cross Validation works:

Divide the dataset into k-subsets.

In each iteration, reserve one subset for testing and train the algorithm on the other k-1 subsets.

Repeat this process, changing the testing subset in each iteration, and average the accuracies to obtain the algorithm's overall accuracy.

K-Fold Cross Validation helps prevent underfitting or overfitting and leads to a more generalized model.

In [56]:
```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
kfold = KFold(n_splits=10, random_state=22) # k=10, split the data into 10 equal parts
xyz=[]
accuracy=[]
std=[]
classifiers=['Linear Svm','Radial Svm','Logistic Regression','KNN','Decision Tree','Naive Bayes','Random Forest']
models=[svm.SVC(kernel='linear'),svm.SVC(kernel='rbf'),LogisticRegression(),KNeighborsClassifier(n_neighbors=9),Decisio
for i in models:
    model = i
    cv_result = cross_val_score(model,X,Y, cv = kfold,scoring = "accuracy")
    cv_result=cv_result
    xyz.append(cv_result.mean())
    std.append(cv_result.std())
    accuracy.append(cv_result)
```

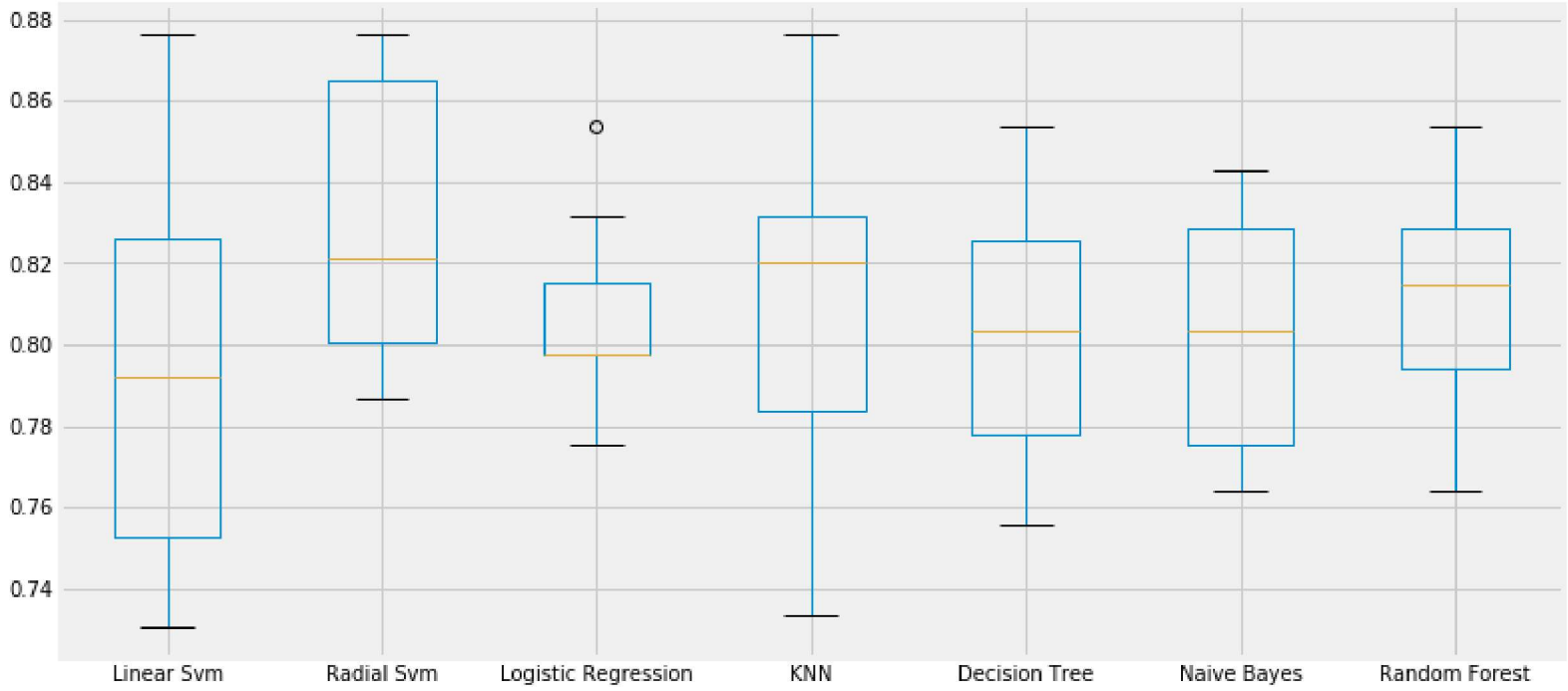
```
new_models_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
new_models_dataframe2
```

Out[56]:

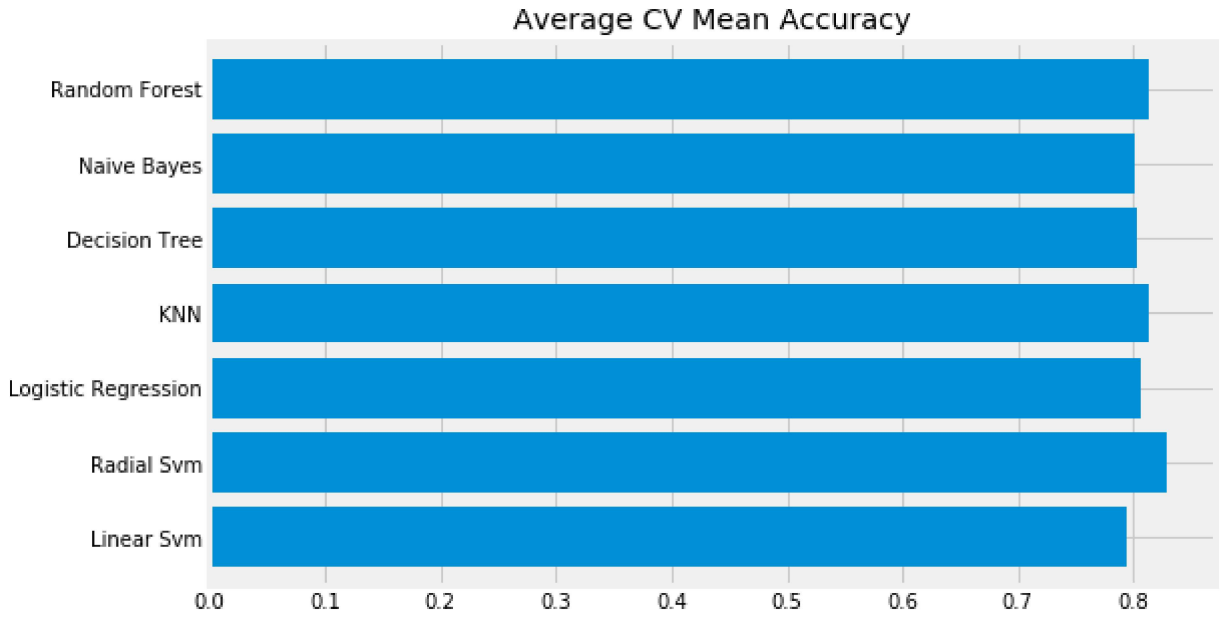
	CV Mean	Std
Linear Svm	0.793471	0.047797
Radial Svm	0.828290	0.034427
Logistic Regression	0.805843	0.021861
KNN	0.813783	0.041210
Decision Tree	0.802522	0.031441
Naive Bayes	0.801386	0.028999
Random Forest	0.812597	0.028683

In [57]:
```
plt.subplots(figsize=(12,6))
box=pd.DataFrame(accuracy,index=[classifiers])
box.T.boxplot()
```

Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x788e0fd23780>
```



In [58]:
```
new_models_dataframe2['CV Mean'].plot.barh(width=0.8)
plt.title('Average CV Mean Accuracy')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```



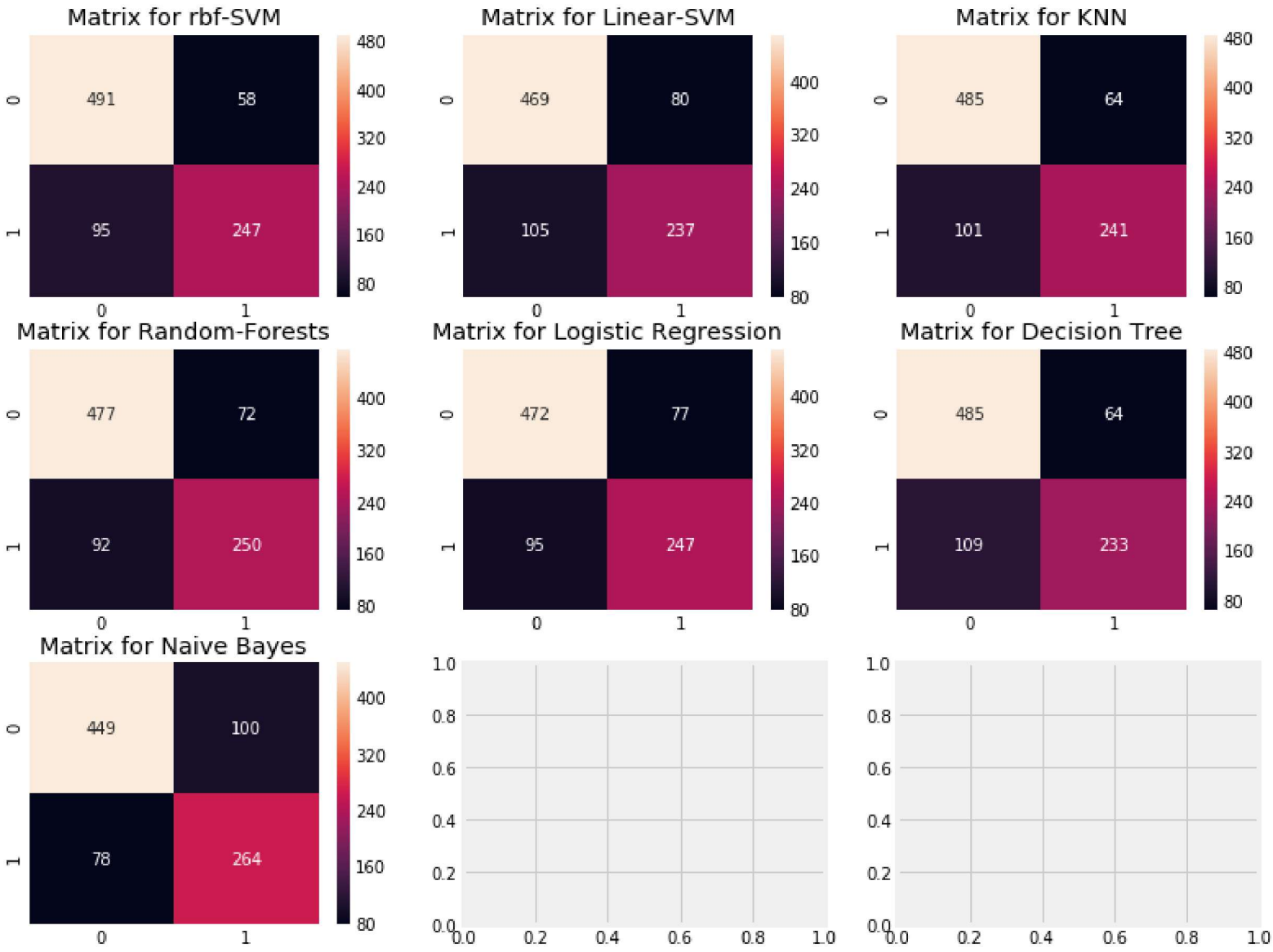
The classification accuracy can be misleading, especially in cases of class imbalance. To obtain a more informative summary, we use a confusion matrix. This matrix shows where the model went wrong and which class it predicted incorrectly.

Confusion Matrix

It provides the count of both correct and incorrect classifications made by the classifier.

In [59]:
```
f,ax=plt.subplots(3,3,figsize=(12,10))
y_pred = cross_val_predict(svm.SVC(kernel='rbf'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,0],annot=True,fmt='2.0f')
ax[0,0].set_title('Matrix for rbf-SVM')
y_pred = cross_val_predict(svm.SVC(kernel='linear'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,1],annot=True,fmt='2.0f')
ax[0,1].set_title('Matrix for Linear-SVM')
y_pred = cross_val_predict(KNeighborsClassifier(n_neighbors=9),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,2],annot=True,fmt='2.0f')
ax[0,2].set_title('Matrix for KNN')
y_pred = cross_val_predict(RandomForestClassifier(n_estimators=100),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,0],annot=True,fmt='2.0f')
ax[1,0].set_title('Matrix for Random-Forests')
y_pred = cross_val_predict(LogisticRegression(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,1],annot=True,fmt='2.0f')
```

```
ax[1,1].set_title('Matrix for Logistic Regression')
y_pred = cross_val_predict(DecisionTreeClassifier(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,2],annot=True,fmt='2.0f')
ax[1,2].set_title('Matrix for Decision Tree')
y_pred = cross_val_predict(GaussianNB(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[2,0],annot=True,fmt='2.0f')
ax[2,0].set_title('Matrix for Naive Bayes')
plt.subplots_adjust(hspace=0.2,wspace=0.2)
plt.show()
```



Interpreting Confusion Matrix

In the confusion matrix, the left diagonal represents the number of correct predictions for each class, while the right diagonal shows the number of incorrect predictions. Let's consider the first plot for rbf-SVM:

The number of correct predictions is 491 (for dead) + 247 (for survived), with the mean cross-validation accuracy being (491+247)/891 = 82.8%, which matches our earlier calculation.

Errors --> It wrongly classified 58 dead people as survivors and 95 survivors as deceased. This indicates more errors in predicting dead as survivors.

Analyzing all the matrices, we can conclude that rbf-SVM has a higher chance of correctly predicting deceased passengers, while Naive Bayes has a higher chance of correctly predicting passengers who survived.

Hyper-Parameters Tuning

Machine learning models are often considered as black boxes. These black boxes have default parameter values that we can adjust or tune to improve the model's performance. For example, in the SVM model, we can tune parameters like C and gamma. These parameters, known as hyper-parameters, enable us to adjust the learning rate of the algorithm and obtain a better model. This process is called Hyper-Parameter Tuning.

In this context, we will focus on tuning the hyper-parameters of the two best classifiers: SVM and RandomForests.

SVM

In [60]:
```
from sklearn.model_selection import GridSearchCV
C=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
gamma=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
kernel=['rbf','linear']
hyper={'kernel':kernel,'C':C,'gamma':gamma}
gd=GridSearchCV(estimator=svm.SVC(),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
0.828282828283
SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
[Parallel(n_jobs=1)]: Done 720 out of 720 | elapsed:   15.4s finished
```

Random Forests

In [61]:
```
n_estimators=range(100,1000,100)
hyper={'n_estimators':n_estimators}
gd=GridSearchCV(estimator=RandomForestClassifier(random_state=0),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   19.9s finished
0.817059483726
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=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=900, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

The best score for Rbf-SVM is 82.82% with C=0.05 and gamma=0.1. For RandomForest, the score is about 81.8% with n_estimators=900.

Ensembling

Ensembling is an effective method for improving the accuracy and performance of a model. In simple terms, it involves combining various simple models to create a single, more powerful model.

To illustrate, imagine you want to buy a phone and ask many people for their opinions based on various parameters. By analyzing different parameters and opinions, you can make a more informed decision about a single product. This process of combining multiple sources of information is called Ensembling, and it enhances the model's stability.

Ensembling can be implemented in several ways, including:

1)Voting Classifier

2)Bagging

3)Boosting.

Voting Classifier

Voting is the simplest way of combining predictions from various simple machine learning models. It provides an average prediction result based on the predictions of all the submodels. These submodels, or base models, can be of different types.

In [62]:
```
from sklearn.ensemble import VotingClassifier
ensemble_lin_rbf=VotingClassifier(estimators=[('KNN',KNeighborsClassifier(n_neighbors=10)),
                                              ('RBF',svm.SVC(probability=True,kernel='rbf',C=0.5,gamma=0.1)),
                                              ('RFor',RandomForestClassifier(n_estimators=500,random_state=0)),
                                              ('LR',LogisticRegression(C=0.05)),
                                              ('DT',DecisionTreeClassifier(random_state=0)),
                                              ('NB',GaussianNB()),
                                              ('svm',svm.SVC(kernel='linear',probability=True))
                                             ],
                        voting='soft').fit(train_X,train_Y)
print('The accuracy for ensembled model is:',ensemble_lin_rbf.score(test_X,test_Y))
cross=cross_val_score(ensemble_lin_rbf,X,Y, cv = 10,scoring = "accuracy")
print('The cross validated score is',cross.mean())
```

```
The accuracy for ensembled model is: 0.824626865672
The cross validated score is 0.822654919986
```

Bagging

Bagging is a general ensemble method that operates by applying similar classifiers to small partitions of the dataset and then averaging their predictions. This averaging process helps reduce variance. Unlike the Voting Classifier, Bagging uses similar classifiers.

Bagged KNN

Bagging works best with models that have high variance. Examples of such models include Decision Trees and Random Forests. Another suitable model for Bagging is KNN with a small value of n_neighbors, as a small value of n_neighbors can introduce variance.

In [63]:
```
from sklearn.ensemble import BaggingClassifier
model=BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3),random_state=0,n_estimators=700)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged KNN is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged KNN is:',result.mean())
```

```
The accuracy for bagged KNN is: 0.835820895522
The cross validated score for bagged KNN is: 0.814889342867
```

Bagged DecisionTree

In [64]:
```
model=BaggingClassifier(base_estimator=DecisionTreeClassifier(),random_state=0,n_estimators=100)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged Decision Tree is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged Decision Tree is:',result.mean())
```

```
The accuracy for bagged Decision Tree is: 0.824626865672
The cross validated score for bagged Decision Tree is: 0.820482635342
```

Boosting

Boosting is an ensemble technique that employs sequential learning of classifiers. It involves step-by-step improvement of a weak model. Boosting works as follows:

Initially, a model is trained on the complete dataset. This model will make some correct predictions and some incorrect ones.

In the next iteration, the learner focuses more on the instances that were previously predicted incorrectly, assigning them greater weight. The aim is to predict these instances correctly.

This iterative process continues, with new classifiers added to the model until a predefined accuracy limit is reached.

AdaBoost(Adaptive Boosting)

The weak learner or estimator in this case is a Decision Tree. However, we can change the default base_estimator to any algorithm of our choice.

In [65]:
```
from sklearn.ensemble import AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
result=cross_val_score(ada,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for AdaBoost is:',result.mean())
```

```
The cross validated score for AdaBoost is: 0.824952616048
```

Stochastic Gradient Boosting

In [66]:
```
from sklearn.ensemble import GradientBoostingClassifier
grad=GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
result=cross_val_score(grad,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for Gradient Boosting is:',result.mean())
```

```
The cross validated score for Gradient Boosting is: 0.818286233118
```

XGBoost

In [67]:
```
import xgboost as xg
xgboost=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
result=cross_val_score(xgboost,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for XGBoost is:',result.mean())
```

```
The cross validated score for XGBoost is: 0.810471002156
```

We achieved the highest accuracy with AdaBoost. We will now attempt to further increase it through hyper-parameter tuning.

Hyper-Parameter Tuning for AdaBoost

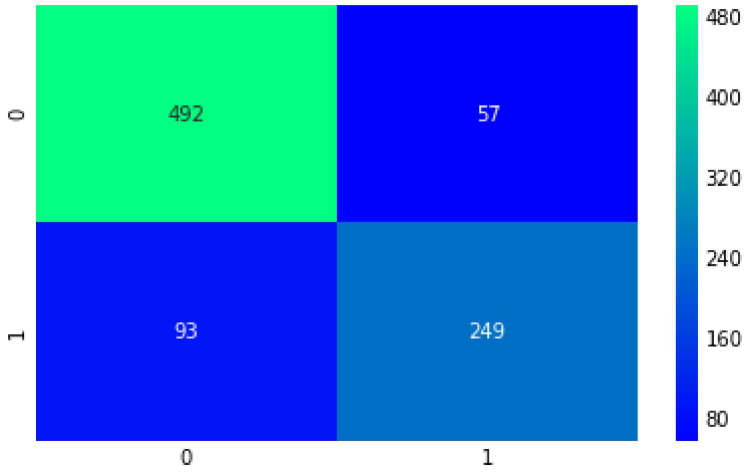
In [68]:
```
n_estimators=list(range(100,1100,100))
learn_rate=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
hyper={'n_estimators':n_estimators,'learning_rate':learn_rate}
gd=GridSearchCV(estimator=AdaBoostClassifier(),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits
[Parallel(n_jobs=1)]: Done 360 out of 360 | elapsed:  5.1min finished
0.83164983165
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=0.05, n_estimators=200, random_state=None)
```

The maximum accuracy we can achieve with AdaBoost is 83.16% with n_estimators=200 and learning_rate=0.05.

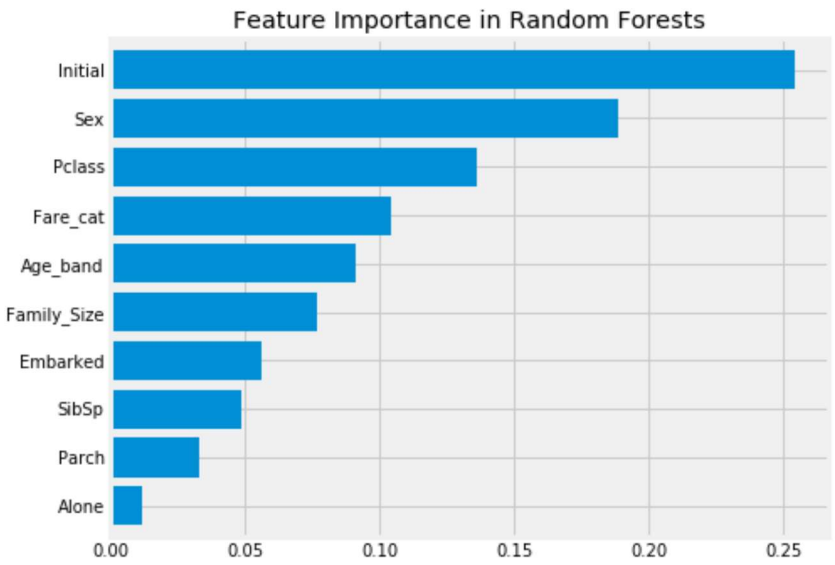
Confusion Matrix for the Best Model

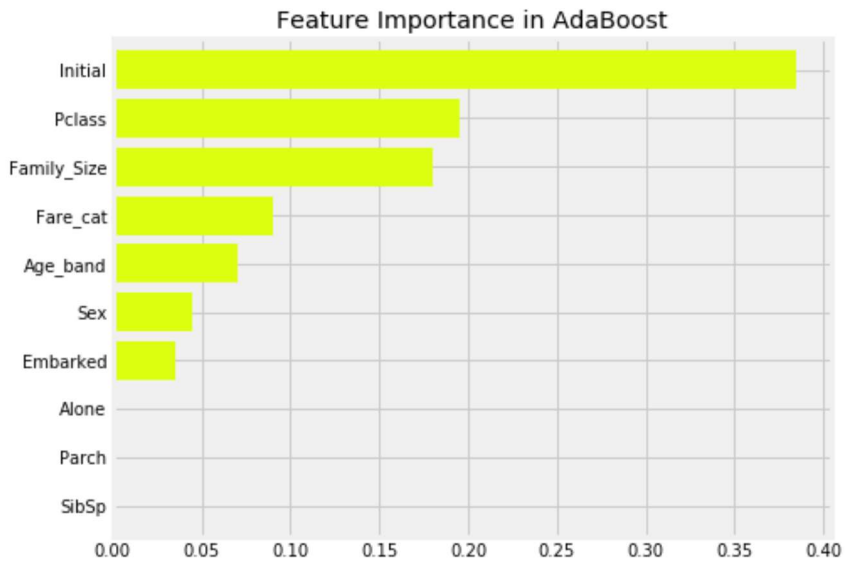
In [69]:
```
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.05)
result=cross_val_predict(ada,X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,result),cmap='winter',annot=True,fmt='2.0f')
plt.show()
```

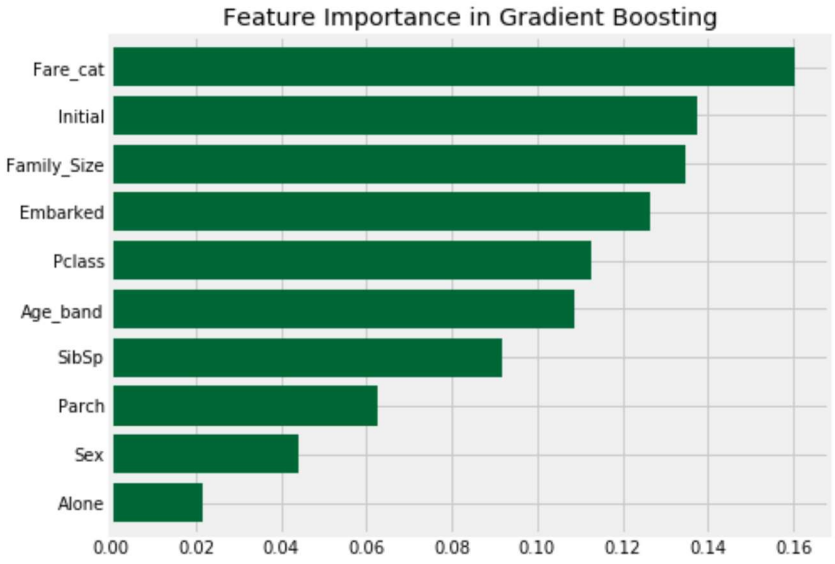


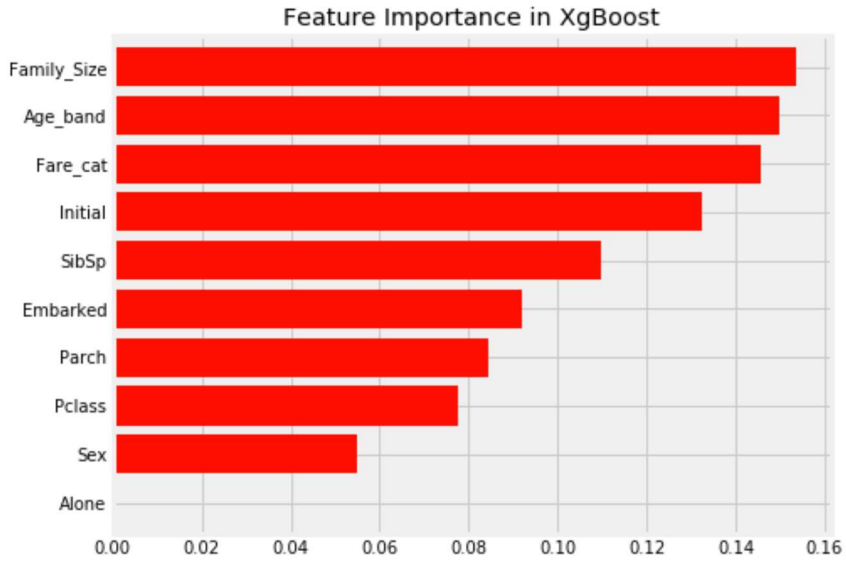
Feature Importance

In [70]:
```
f,ax=plt.subplots(2,2,figsize=(15,12))
model=RandomForestClassifier(n_estimators=500,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0,0])
ax[0,0].set_title('Feature Importance in Random Forests')
model=AdaBoostClassifier(n_estimators=200,learning_rate=0.05,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0,1],color='#ddff
ax[0,1].set_title('Feature Importance in AdaBoost')
model=GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[1,0],cmap='RdYlGn
ax[1,0].set_title('Feature Importance in Gradient Boosting')
model=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[1,1],color='#FD0F
ax[1,1].set_title('Feature Importance in XgBoost')
plt.show()
```









We can identify the important features for various classifiers such as RandomForests, AdaBoost, etc.

Observations:

"Common important features include Initial, Fare_cat, Pclass, and Family_Size."

"Surprisingly, the Sex feature does not appear to be important in most classifiers, even though we previously observed that Sex combined with Pclass was a differentiating factor. Sex is only important in RandomForests. On the other hand, Initial, which correlates with Sex, is a top feature in many classifiers, indicating its significance."

"Similarly, Pclass and Fare_cat reflect the passengers' status, while Family_Size is related to Alone, Parch, and SibSp."