Logistic regression on titanic dataset

In this noterbook there is an attemt to analyze the data incurred from the disaster of titanic in the motive of the creation of a
machine learning model to use the logistic regression primarly for the identification of the survival status of the
passengers of the ship.

Libraries are being imported below

In [60]:

```
import numpy as np
import pandas as pd

from sklearn import preprocessing

import matplotlib.pyplot as plt

plt.rc("font", size=14)

import seaborn as sns

sns.set(style="white") #white background style for seaborn plots
sns.set(style="whitegrid", color_codes=True)

import warnings
warnings.simplefilter(action='ignore' )
```

The libraries used are being used in this are as follows -

Numpy - It means numeric python that mainly is used for handling the mathematical operations in python

Sklearn - it is a massive library with several kinds of algorithms presented for various needs such as learning, data management, preprocessing, data visualization etc.

There is the usage of the preprocessing set from the library - sklearn that will deal with the splitting the data that we have taken in form of a csv file

Matplotlib - this library is a massive and widely used data visualization tool that has various types of plots to make the layman understand the
data with ease.

Pandas - It is a very powerful library used for data manipulation making it easier to import and perform basic operation
with ease.

Seaborn - this is a powerful library that is based upon the matplotlib, that is having access to various highlevel visualizations

There is a attempt to set the backgrounds of the plots to white and a grid pattern is followed.

In [61]:

```
# Read CSV train data file into DataFrame
train_df = pd.read_csv("C:/Users/ABHISHEK/Desktop/train.csv")

# Read CSV test data file into DataFrame
test_df = pd.read_csv("C:/Users/ABHISHEK/Desktop/test.csv")

# preview train data
train_df.head()
```

Out[61]:

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

Here there is usage of the pandas library's built in function to import a csv file into the present code and assigning it to the variables & using the head() function of the pandas, attempting to see the top 5 records the training data and get to know it in a brief manner.

In [65]:

```
print('The number of samples into the train data is {}.'.format(train_df.shape[0]))
```

The number of samples into the train data is 891.

Usage of the shape function on the training data to see the dimensions of the data and from that only the no of rows is taken.

In [66]:

```
test_df.head()
```

Out[66]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

Usage of the head() function of the pandas we are tyring to see the top 5 records the test data and get to know it in a brief manner.

In [67]:

```
print('The number of samples into the test data is {}.'.format(test_df.shape[0]))
```

The number of samples into the test data is 418.

using the shape function on test data to see the dimensions of the data and from that only the no of rows is taken

In [68]:

```
train_df.isnull().sum()
```

Out[68]:

```
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
```

Usage of the isnull() function to find the number of empty values in the total data set so based on that we can get an idea on how to infer the data and found that only in the 3 coulumns whether there are missing values

In [70]:

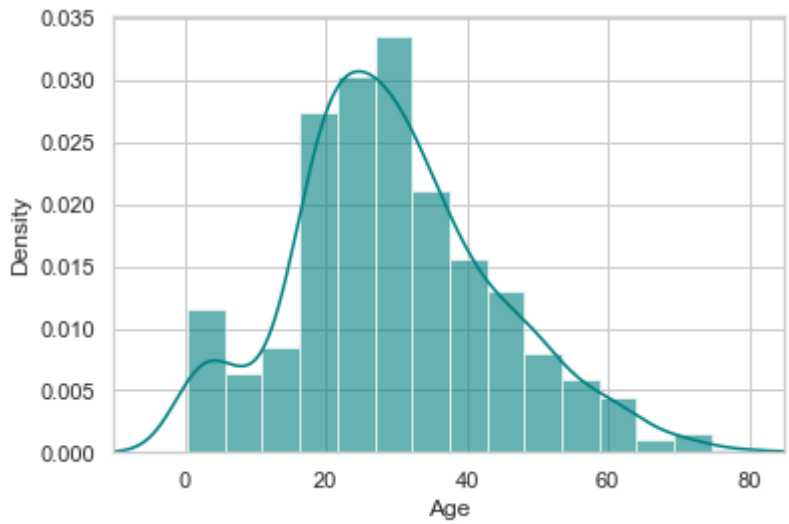
```
# percent of missing "Age"
print('Percent of missing "Age" records is %.2f%%' %((train_df['Age'].isnull().sum()/train_df.shape[0])*10
```

Percent of missing "Age" records is 19.87%

The calculations of the percentage of missing records of age column with respect to the total number of samples present in the data set

In [73]:

```
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



By the usage of this histogram plot there can be the finding of the distribution of the age column based on which one can get an idea of how to deal with the missing data in "age" colum.

In [75]:

```
# mean age
print('The mean of "Age" is %.2f' %(train_df["Age"].mean(skipna=True)))
# median age
print('The median of "Age" is %.2f' %(train_df["Age"].median(skipna=True)))
```

```
The mean of "Age" is 29.70
The median of "Age" is 28.00
```

There is the usage of the basic functions of mean and median on the "age" column

In [77]:

```
# percent of missing "Cabin"
print('Percent of missing "Cabin" records is %.2f%%' %((train_df['Cabin'].isnull().sum()/train_df.shape[0])
```

```
Percent of missing "Cabin" records is 77.10%
```

There is the calculation of the percentage of the missing records of cabin column with respect to the total number of samples present in the data set

In [78]:

```
# percent of missing "Embarked"
print('Percent of missing "Embarked" records is %.2f%%' %((train_df['Embarked'].isnull().sum()/train_df.
```

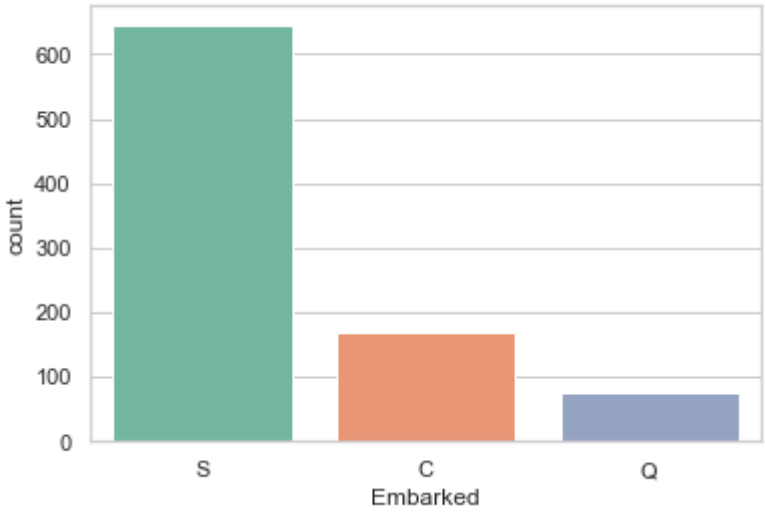
```
Percent of missing "Embarked" records is 0.22%
```

There is the calculation of the percentage of missing records of "embarked" column with respect to the total number of samples that are present in the data set

In [81]:

```
print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southamp
print(train_df['Embarked'].value_counts())
sns.countplot(x='Embarked', data=train_df, palette='Set2')
plt.show()
```

```
Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S =
Southampton):
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```



By the usage of a count plot that is in the seaborn library's built in function, there is determination of the the total count of the embarked column based on all the categories that are present are given in the numeric format and a plot based representation

In [82]:

```
print('The most common boarding port of embarkation is %s.' %train_df['Embarked'].value_counts().idxma
```

```
The most common boarding port of embarkation is S.
```

By the usage of the Count function along with the id function on the "embarked" column there is the determination of the most common part of the data set which help in the retieval of this attribute data

In [83]:

```
train_data = train_df.copy()
train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
train_data["Embarked"].fillna(train_df['Embarked'].value_counts().idxmax(), inplace=True)
train_data.drop('Cabin', axis=1, inplace=True)
```

When we copy the data set into the new variable there is a usage of Fillna() function that finds the missing values of the age column with a median values of the age column that in turn fills the most used value of the embarked column in place of the missing values in embarked column dropping the missing values of cabin column also.

In [85]:

```
# check missing values in adjusted train data
train_data.isnull().sum()
```

Out[85]:

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

Now, There is the usage of the Null() function for checking whether there are any missing values in the data set or not.

In [87]:

```
# preview adjusted train data
train_data.head()
```

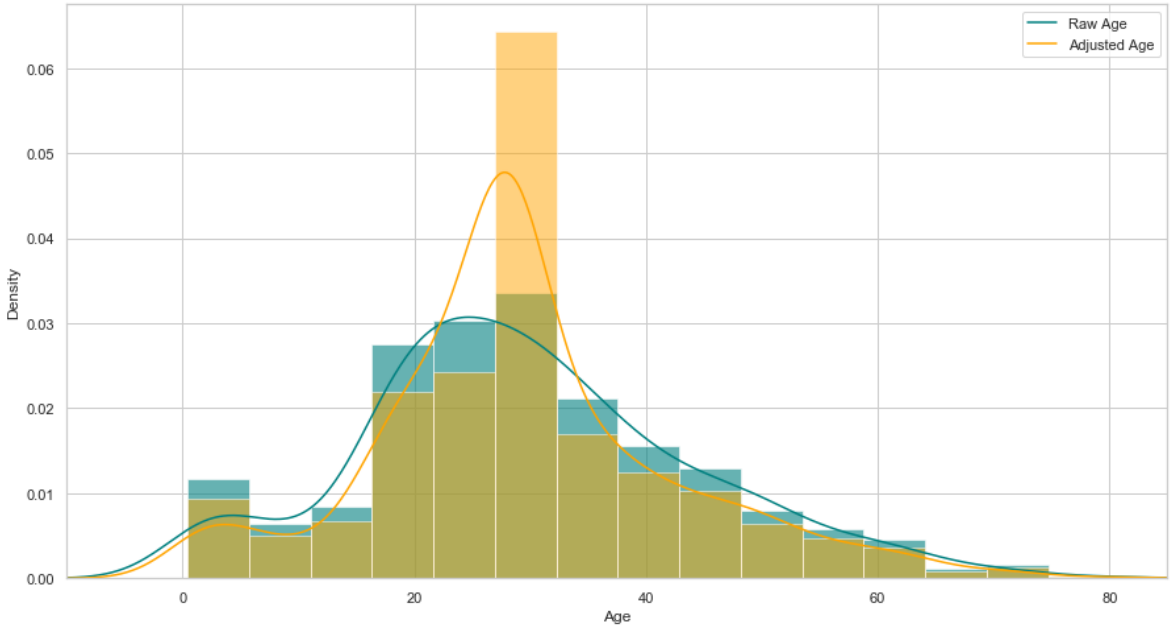
Out[87]:

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

Now, we use the head() function to get the top five records of the training data set

In [91]:

```
plt.figure(figsize=(15,8))
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax = train_data["Age"].hist(bins=15, density=True, stacked=True, color='orange', alpha=0.5)
train_data["Age"].plot(kind='density', color='orange')
ax.legend(['Raw Age', 'Adjusted Age'])
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



There is the plotting both the data sets after imputation of the age column the plot shows the distribution that doesn't have a huge difference in it.

In [ ]:

```
## Create categorical variable for traveling alone
train_data['TravelAlone']=np.where((train_data["SibSp"]+train_data["Parch"])>0, 0, 1)
train_data.drop('SibSp', axis=1, inplace=True)
train_data.drop('Parch', axis=1, inplace=True)
```

This is for the reduction of the multi-co-linearity and over-fitting of the data that are being combined with the data available in the sibsp and parch col into one and droping the other two columns for the manipulation with ease.

In [35]:

```
#create categorical variables and drop some variables
training=pd.get_dummies(train_data, columns=["Pclass","Embarked","Sex"])
training.drop('Sex_female', axis=1, inplace=True)
training.drop('PassengerId', axis=1, inplace=True)
training.drop('Name', axis=1, inplace=True)
training.drop('Ticket', axis=1, inplace=True)

final_train = training
final_train.head()
```

Out[35]:

	Survived	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked
0	0	22.0	7.2500	0	0	0	1	0	
1	1	38.0	71.2833	0	1	0	0	1	
2	1	26.0	7.9250	1	0	0	1	0	
3	1	35.0	53.1000	0	1	0	0	0	
4	0	35.0	8.0500	1	0	0	1	0	

There is the creation of the categorical variables for the columns of sex, pclass, Embarked and there is a dropping of the original cols

In [37]:

```
test_df.isnull().sum()
```

Out[37]:

```
PassengerId     0
Pclass          0
Name            0
Sex             0
Age            86
SibSp           0
Parch           0
Ticket          0
Fare            1
Cabin         327
Embarked        0
dtype: int64
```

Whethere there is any missing value or not are being checked

In [38]:

```
test_data = test_df.copy()
test_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
test_data["Fare"].fillna(train_df["Fare"].median(skipna=True), inplace=True)
test_data.drop('Cabin', axis=1, inplace=True)

test_data['TravelAlone']=np.where((test_data["SibSp"]+test_data["Parch"])>0, 0, 1)

test_data.drop('SibSp', axis=1, inplace=True)
test_data.drop('Parch', axis=1, inplace=True)

testing = pd.get_dummies(test_data, columns=["Pclass","Embarked","Sex"])
testing.drop('Sex_female', axis=1, inplace=True)
testing.drop('PassengerId', axis=1, inplace=True)
testing.drop('Name', axis=1, inplace=True)
testing.drop('Ticket', axis=1, inplace=True)

final_test = testing
final_test.head()
```

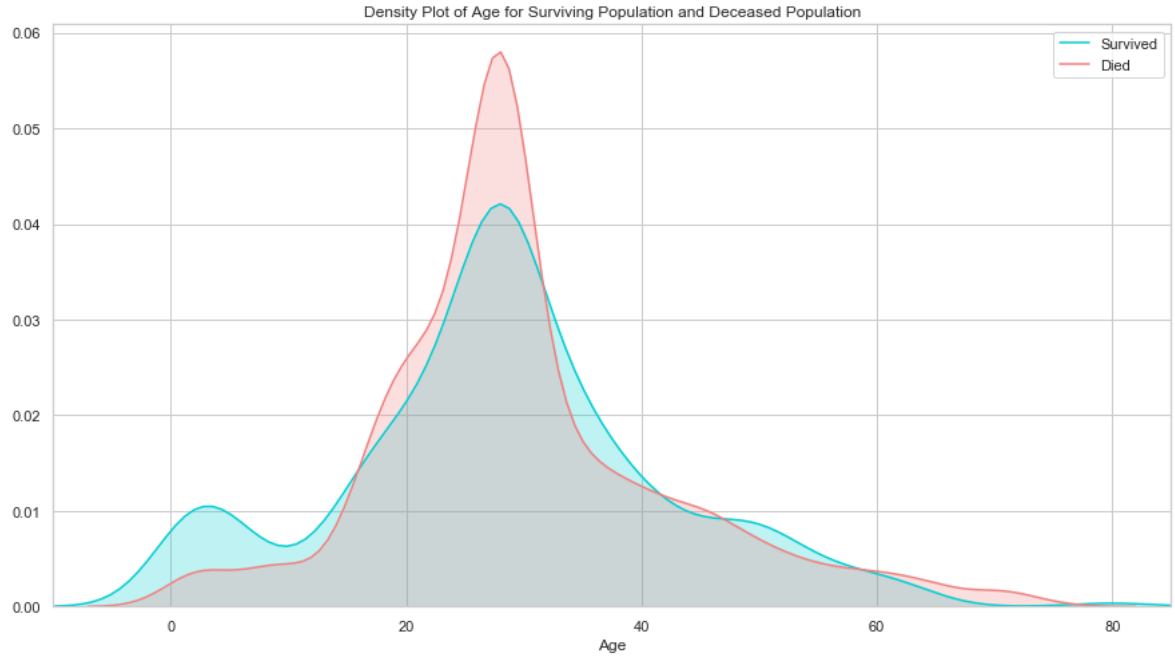
Out[38]:

	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	Embar
0	34.5	7.8292	1	0	0	1	0	1	
1	47.0	7.0000	0	0	0	1	0	0	
2	62.0	9.6875	1	0	1	0	0	1	
3	27.0	8.6625	1	0	0	1	0	0	
4	22.0	12.2875	0	0	0	1	0	0	

The test dataset is being imputed in the same way of the training dataset and comparing the sibsp and parch into one and dropping the other two columns and in the position of the missing vals in age and fare are being assigned the median values of the columns and there is also the assigning of the categorical variables for columns

In [39]:

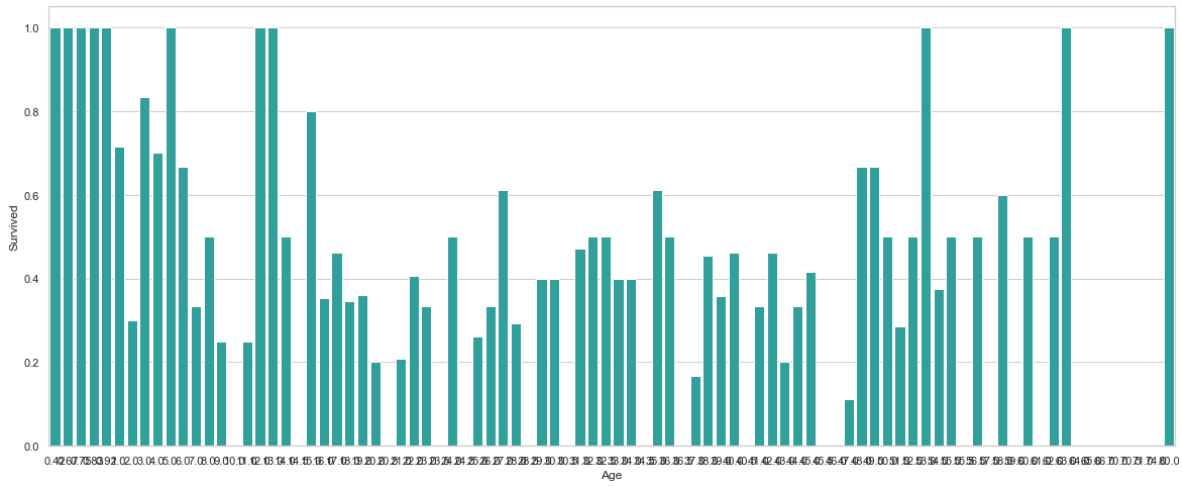
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Age for Surviving Population and Deceased Population')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



There is the usage of the kernel density plot of the seaborn to find the how the passengers are distributed across age and survived status and the distribution is almost similar for both survived and deseased are also found. Out of the survived state there has been a major density in the children so that's why it shows there is an efficient effort that has been made for the saving of the children.

In [40]:

```
plt.figure(figsize=(20,8))
avg_survival_byage = final_train[["Age", "Survived"]].groupby(['Age'], as_index=False).mean()
g = sns.barplot(x='Age', y='Survived', data=avg_survival_byage, color="LightSeaGreen")
plt.show()
```



The survival rate in the total dataset is being checked and we can observe the age under 16 has a lot of difference that's why there is a new column as minor for all the people under 16 are made

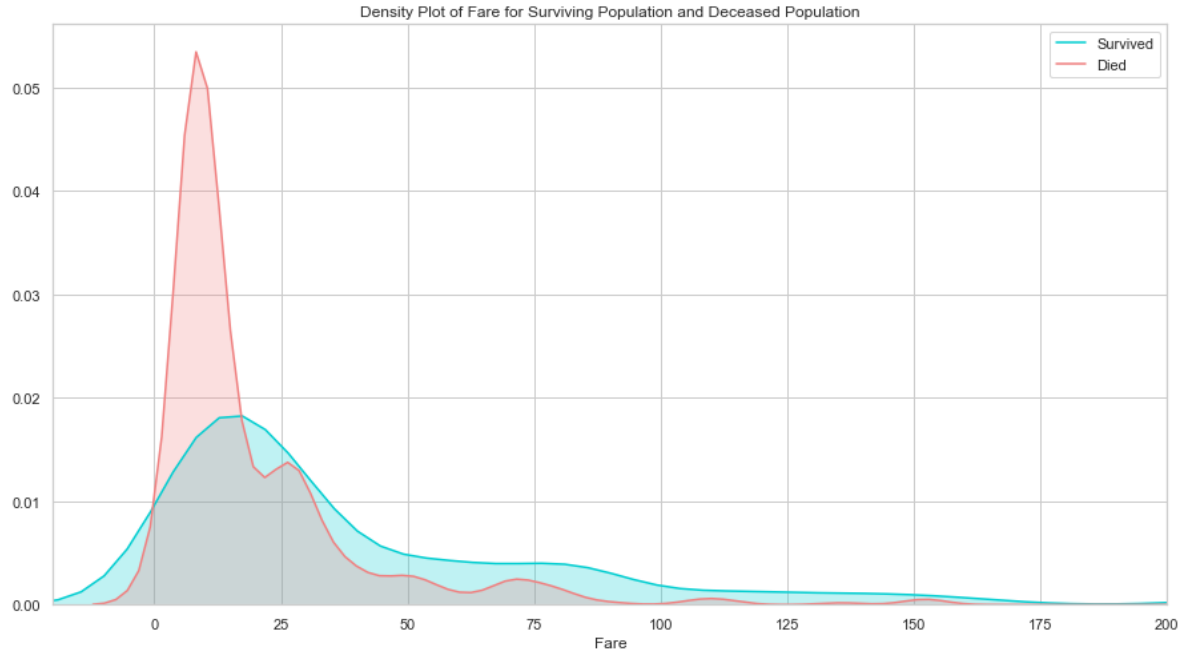
In [42]:

```
final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)

final_test['IsMinor']=np.where(final_test['Age']<=16, 1, 0)
```

The minor column has been assigned for both the train as well as test dataset in a categorical way

In [43]:

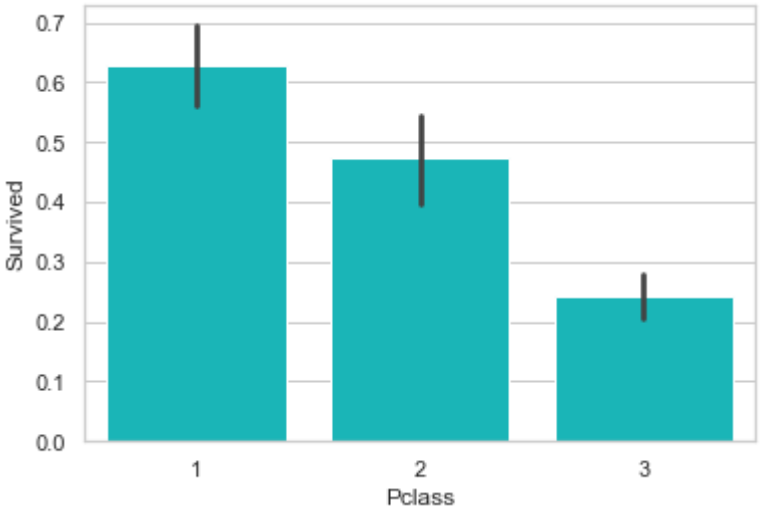
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Fare"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Fare"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Fare for Surviving Population and Deceased Population')
ax.set(xlabel='Fare')
plt.xlim(-20,200)
plt.show()
```



There is again the usage of the kernel density plot to observe the distribution of the people across the fare for the survived and deseased state and it shows that the people who paid higher fares had a good probability for being survived.

In [44]:

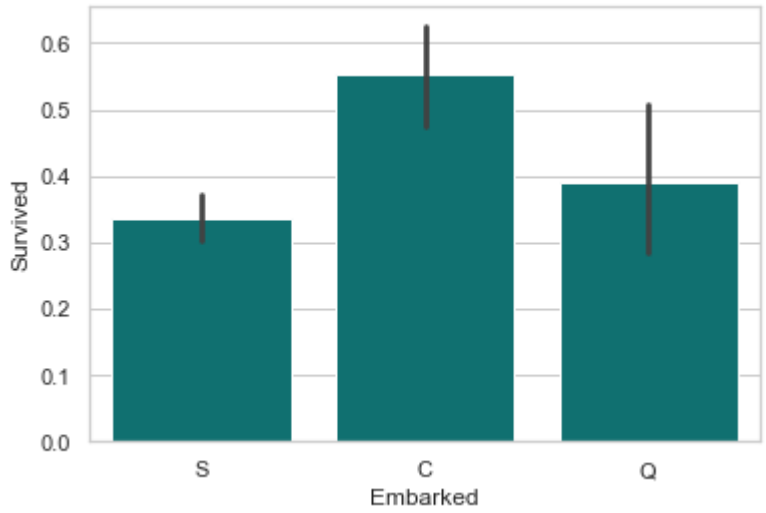
```
sns.barplot('Pclass', 'Survived', data=train_df, color="darkturquoise")
plt.show()
```



There is the usage of the bar plot which will help in to find the the number people who survived acrosses different passenger classes & it shows that there is a significant chance for the people of the 1st class to have a good chance of survival

In [45]:

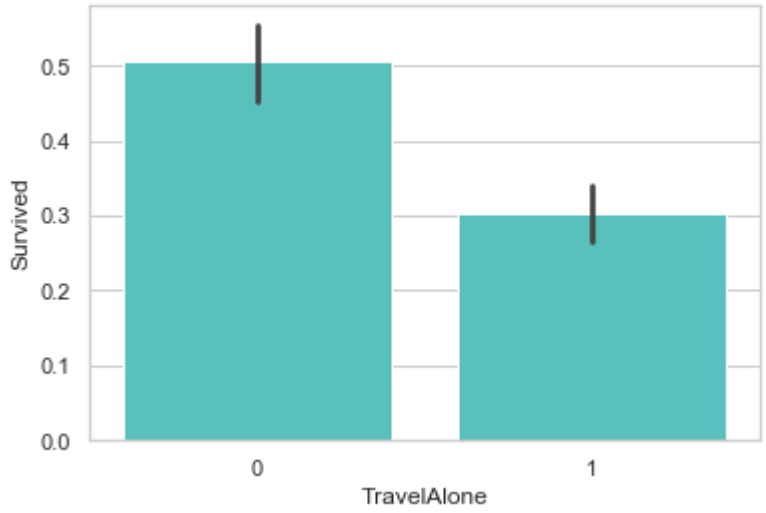
```
sns.barplot('Embarked', 'Survived', data=train_df, color="teal")
plt.show()
```



According to the bar graph we can incurr the detail of the data and we infer that the passengers who boarded in Cherbourg, France, appear to have the highest survival rate.& for the passengers who boarded in Southhampton were marginally less likely to survive than those who boarded in Queenstown. This has depicted a relation between the passenger class, or maybe even the order of the room assignments (e.g. maybe earlier passengers were more likely to have rooms closer to deck). the size of the whiskers in these plotsalso has some information on the number of passengers who boarded at Southhampton was highest, the confidence around the survival rate is the highest. The whisker of the Queenstown plot includes the Southhampton average, as well as the lower bound of its whisker. It's possible that Queenstown passengers were equally, or even
more, ill-fated than their Southhampton counterparts.

In [46]:

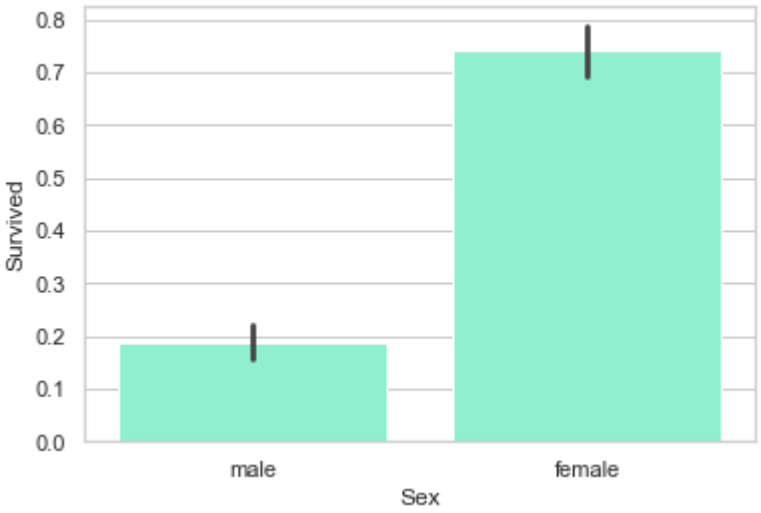
```
sns.barplot('TravelAlone', 'Survived', data=final_train, color="mediumturquoise")
plt.show()
```



We can incurr from the bar graph that the individuals that are travelling alone are more likely to die than those of the ones who are with family as the ones with the family have a good survival rate and based on which the process of the journey is more probable that the majority of the individual travellers would be of male gender.

In [48]:

```
sns.barplot('Sex', 'Survived', data=train_df, color="aquamarine")
plt.show()
```



In this bar plot,there is an depiction that the femal passenges have a better chance of survival than that compared with the male that was depicted in the prior bar plot

In [49]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMino
X = final_train[cols]
y = final_train['Survived']
# Build a logreg and compute the feature importances
model = LogisticRegression()
# create the RFE model and select 8 attributes
rfe = RFE(model, 8)
rfe = rfe.fit(X, y)
# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

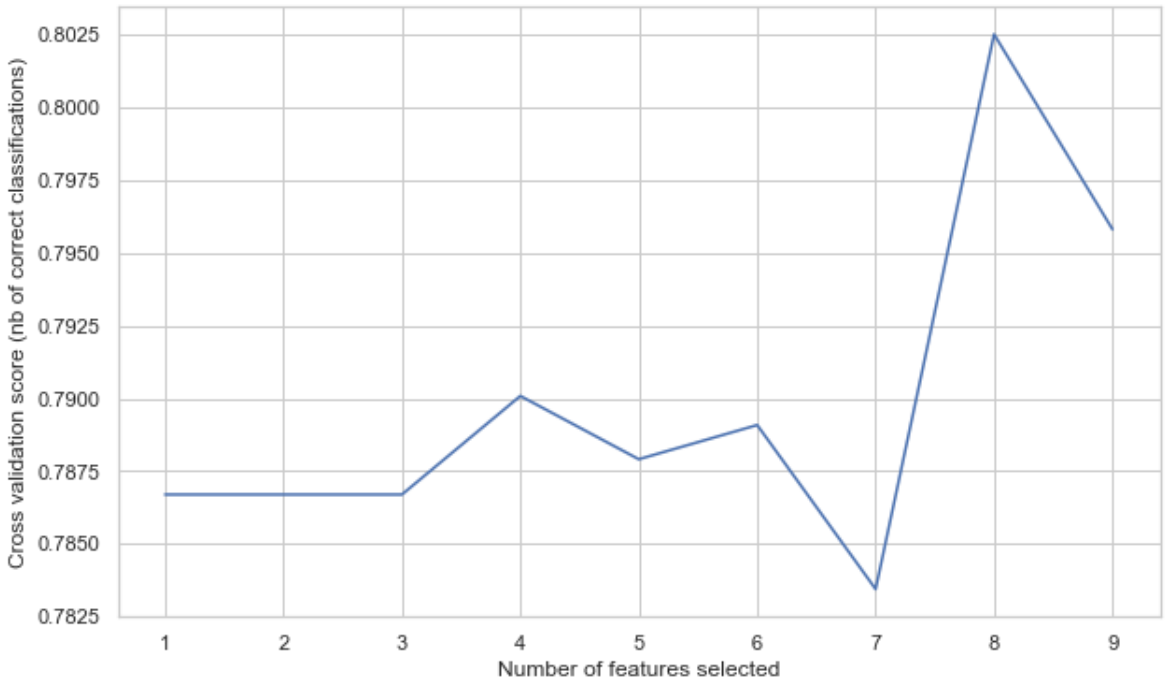
```
Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarke
d_S', 'Sex_male', 'IsMinor']
```

By using the library of sklearn there is an importing that is done int he two modules that is a logistic regression for finding the survival status in which there is a categorical way and another module has been the feature selection used to mark the important feature or attribute in the fataset thus loosing the unwanted data and making it easier for the model to computer and train and decrease overfitting using the RFE(Recursive Feature Elimination) we recursively train a model to finds the best performing feature or attribute in accordance with the model and there s a selection of only the best features.

In [50]:

```
from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=10, scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(X.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(10,6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

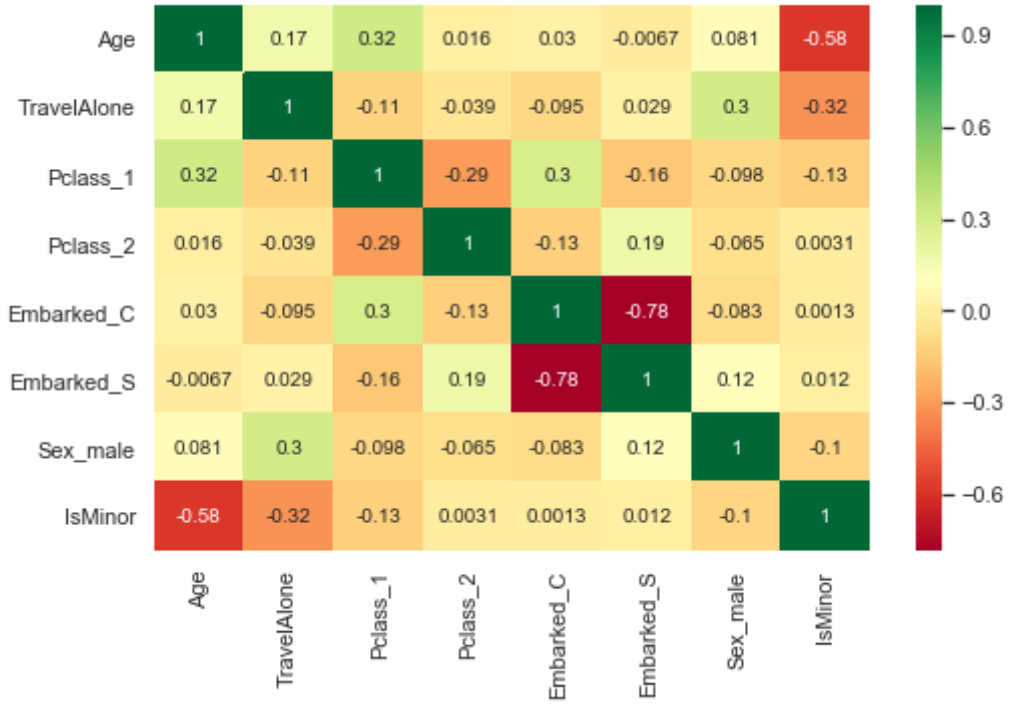
```
Optimal number of features: 8
Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarke
d_S', 'Sex_male', 'IsMinor']
```



There is the usage of the RFECV(Recursive Feature Elimination and Cross-Validated) in the module that will evaluate the features in a recursive manner along with the cross validation applied each time and providing a support vector for a better indepth understanding of the feature importance after which, the above plot shows the no of features along with the cross validation score that is also helpfil in auto tuning the no of feature required for the logestic regression afterwards the module opted for 9 features as optimal consideration.

In [51]:

```
Selected_features = ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C',
                     'Embarked_S', 'Sex_male', 'IsMinor']
X = final_train[Selected_features]

plt.subplots(figsize=(8, 5))
sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
plt.show()
```

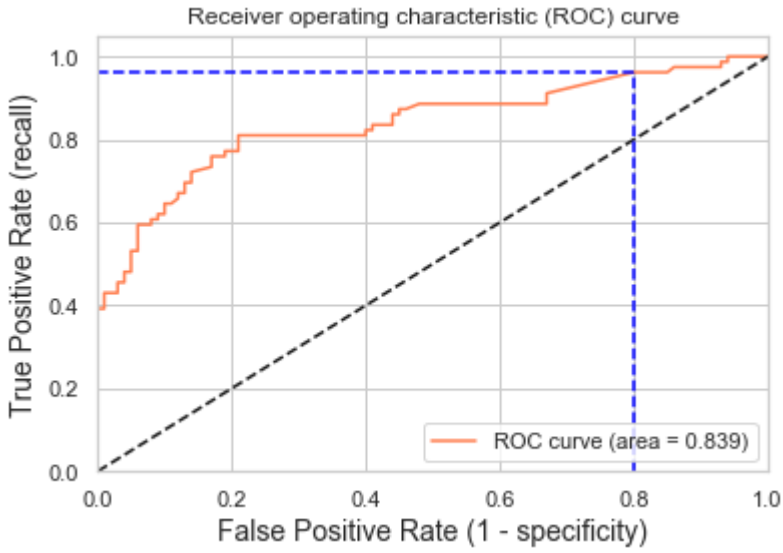


There is the application of the heatmap to show the co-relation of the features of the choice and it is done just to dive down much deeper insignts for a better understanding of all the feature well so we can isolate the less cooperating features like age and isminor doesn't go well compared to others.

In [52]:

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss

# create X (features) and y (response)
X = final_train[Selected_features]
y = final_train['Survived']

# use train/test split with different random_state values
# we can change the random_state values that changes the accuracy scores
# the scores change a lot, this is why testing scores is a high-variance estimate
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

# check classification scores of logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_proba = logreg.predict_proba(X_test)[:, 1]
[fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
print('Train/Test split results:')
print(logreg.__class__.__name__+" accuracy is %2.3f" % accuracy_score(y_test, y_pred))
print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_proba))
print(logreg.__class__.__name__+" auc is %2.3f" % auc(fpr, tpr))

idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95

plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--', color='blue')
plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--', color='blue')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
plt.ylabel('True Positive Rate (recall)', fontsize=14)
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend(loc="lower right")
plt.show()

print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +
      "and a specificity of %.3f" % (1-fpr[idx]) +
      ", i.e. a false positive rate of %.2f%%." % (np.array(fpr[idx])*100))
```

```
Train/Test split results:
LogisticRegression accuracy is 0.782
LogisticRegression log_loss is 0.504
LogisticRegression auc is 0.839
```



Using a threshold of 0.071 guarantees a sensitivity of 0.962 and a specificity of 0.200, i.e. a false positive rate of 80.00%.

There is the usage of the train test split in which there is a split of the training data into 80-20 subsets also using the sklearn metrics modules for performance calculation in various ways while there is a fitting the logistic regression model the prediction probablity is also measured and both
these outputs are sent for calculation of the roc curves which is usually used to find the performance of the classifier and the curves also shows the threshold of the performance metrics between the true positive & false positives nature of the model used in the area under the roc curve which also depicts a good idea how efficient the model is being performing as there is the range of the area that should be 0.5-1 and higher the number, the much better is the performance of the module.

In [53]:

```
# 10-fold cross-validation logistic regression
logreg = LogisticRegression()
# Use cross_val_score function
# We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - althought they are many
scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')
print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())
```

K-fold cross-validation results:
LogisticRegression average accuracy is 0.802
LogisticRegression average log_loss is 0.454
LogisticRegression average auc is 0.850

There is the passing of the model for the k-fold where k=10 is there in the above case that is done for the cross validation based on the score where the score is set for accuracy and the algorithm that will choose an higher accuracy as a primary constraint to check for any better performance with the whole data at once without splitting it and then passing it and the accuracy has just increased by .02 percent and no huge difference between the process and results.

In [54]:

```
from sklearn.model_selection import cross_validate

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['
                          if list(scoring.values())[sc]=='neg_log_loss'
                          else results['test_%s' % list(scoring.values())[sc]].mean(),
                          results['test_%s' % list(scoring.values())[sc]].std())
```

K-fold cross-validation results:
LogisticRegression average accuracy: 0.802 (+/-0.025)
LogisticRegression average log_loss: 0.454 (+/-0.034)
LogisticRegression average auc: 0.850 (+/-0.025)

There is the passing of the model for the k-fold where k=10 is there in the above case that is done for the cross validation without the score
parameter to be checked for a much better performing model using the whole data at once without splitting it and then passing it and the accuracy has not changed from the previous value and both of the algorithms are found at the same level of accuracy.

In [55]:

```
cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMino
X = final_train[cols]

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, final_train[cols], y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['
                          if list(scoring.values())[sc]=='neg_log_loss'
                          else results['test_%s' % list(scoring.values())[sc]].mean(),
                          results['test_%s' % list(scoring.values())[sc]].std())
```

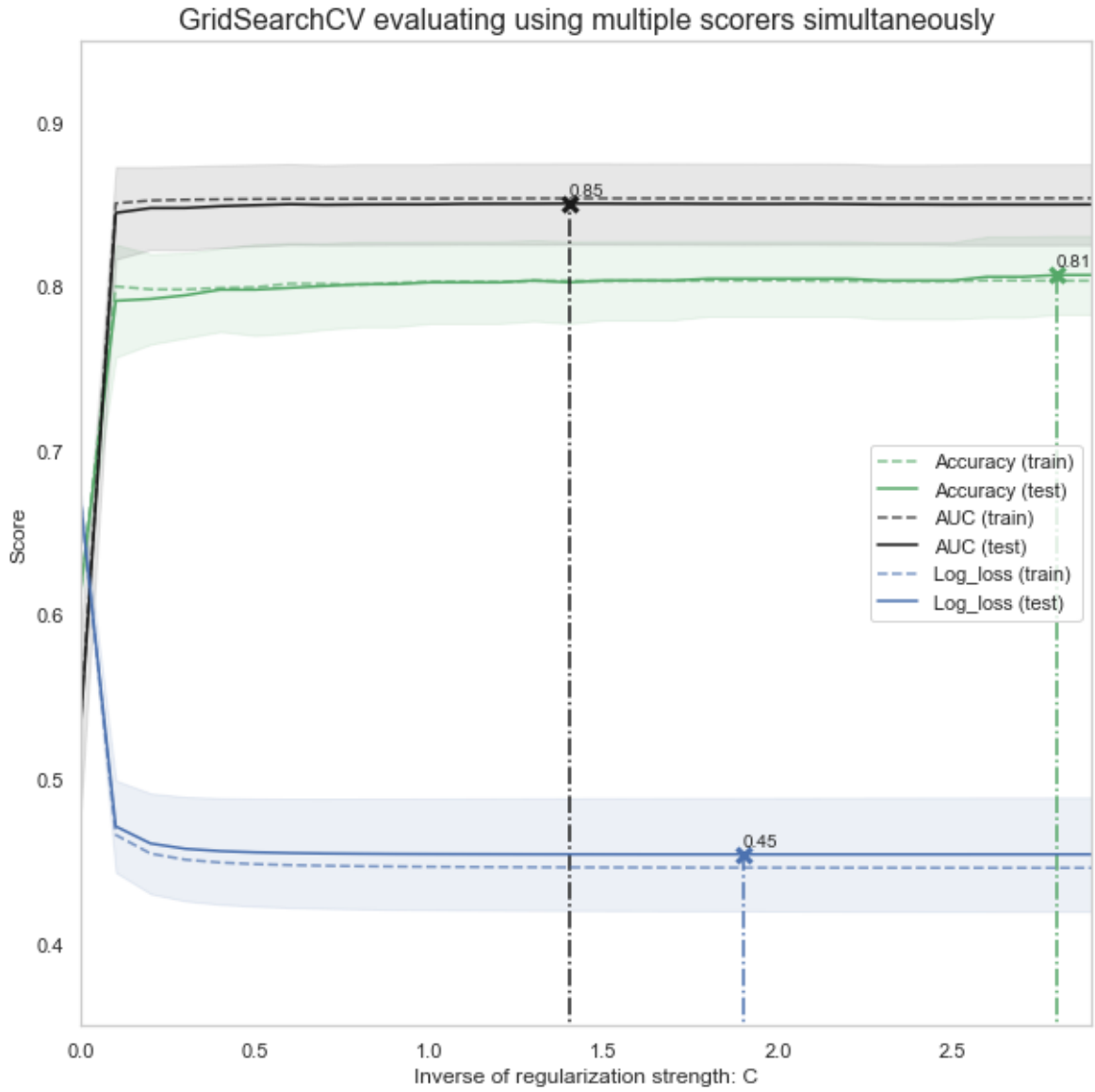
K-fold cross-validation results:
LogisticRegression average accuracy: 0.796 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.034)
LogisticRegression average auc: 0.849 (+/-0.025)

There is a supply of the model with an already pre-fixed set of features or attributes that is worked with and there is a proper finding of the
performance of it and it has been found that the fare feature has no much use and the model performance has slightly detoriorated but by a small quantity only

In [56]:

```
from sklearn.model_selection import GridSearchCV

X = final_train[Selected_features]

param_grid = {'C': np.arange(1e-05, 3, 0.1)}
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}

gs = GridSearchCV(LogisticRegression(), return_train_score=True,
                  param_grid=param_grid, scoring=scoring, cv=10, refit='Accuracy')

gs.fit(X, y)
results = gs.cv_results_

print('='*20)
print("best params: " + str(gs.best_estimator_))
print("best params: " + str(gs.best_params_))
print('best score:', gs.best_score_)
print('='*20)

plt.figure(figsize=(10, 10))
plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)

plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, param_grid['C'].max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss'
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else r

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

```
====================
best params: LogisticRegression(C=2.8000100000000003, class_weight=None, dual=False,
        fit_intercept=True, intercept_scaling=1, max_iter=100,
        multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
        solver='warn', tol=0.0001, verbose=0, warm_start=False)
best params: {'C': 2.8000100000000003}
best score: 0.8069584736251403
====================
```



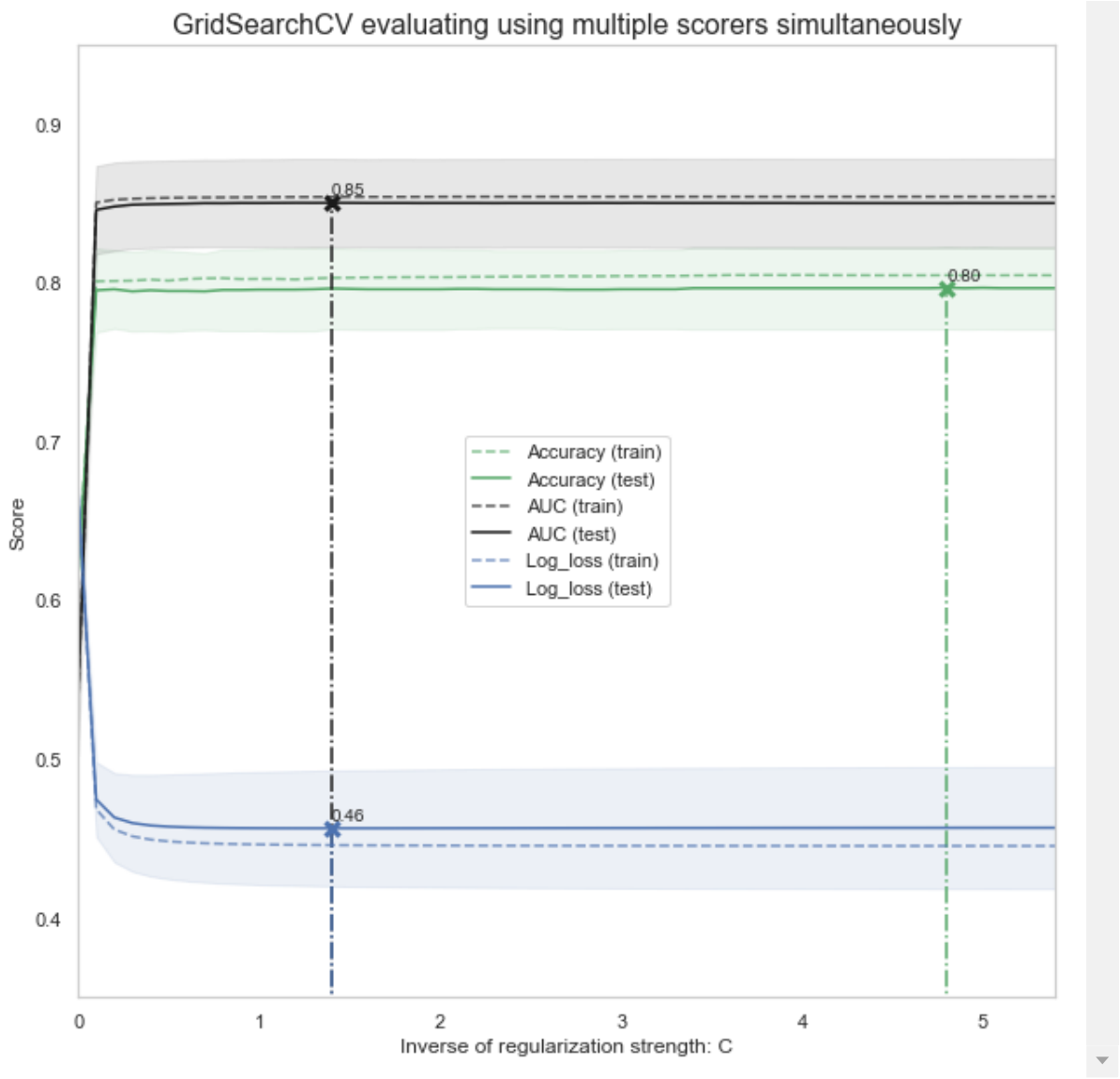
Here there is a application of the Gridsearchcv that has been used for tuning the hyper-parameters in a loop for the finding of the best way possible for the model to better perform and the above plot shows the training of the algorithms on the different sets of data and the accuracy score and the loss of the model and this shows the train and test data both perform in a simimlar way and the optimal parameters has been given as output of the given function in the above assigned cells.

In [58]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline

#Define simple model
##############################################################################
C = np.arange(1e-05, 5.5, 0.1)
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
log_reg = LogisticRegression()

#Simple pre-processing estimators
##############################################################################
std_scale = StandardScaler(with_mean=False, with_std=False)
#std_scale = StandardScaler()

#Defining the CV method: Using the Repeated Stratified K Fold
##############################################################################

n_folds=5
n_repeats=5

rskfold = RepeatedStratifiedKFold(n_splits=n_folds, n_repeats=n_repeats, random_state=2)

#Creating simple pipeline and defining the gridsearch
##############################################################################

log_clf_pipe = Pipeline(steps=[('scale',std_scale), ('clf',log_reg)])

log_clf = GridSearchCV(estimator=log_clf_pipe, cv=rskfold,
          scoring=scoring, return_train_score=True,
          param_grid=dict(clf__C=C), refit='Accuracy')

log_clf.fit(X, y)
results = log_clf.cv_results_

print('='*20)
print("best params: " + str(log_clf.best_estimator_))
print("best params: " + str(log_clf.best_params_))
print('best score:', log_clf.best_score_)
print('='*20)

plt.figure(figsize=(10, 10))
plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)

plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, C.max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_clf__C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss'
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
```

```
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else r

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

```
====================
best params: Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=False, with_std=False)), ('clf',
LogisticRegression(C=4.80001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False))])
best params: {'clf__C': 4.80001}
best score: 0.7966329966329966
====================
```



Here there is a grid search & a crossvalidation is used together many times in the loop to tune the model even on a much deeper level & also on a scaled data all the previous function is taken to another milestone further by preparing a pipline to scale the data & then send it into the classifier for the prediction of the data the data that is scaled to remove the mean and bring them in an order of variance for the reduction of the the calculation time on the model and to predict the pipline is again sent into the RepeatedStratifiedKFold which is a similar method to the kfold cross validation but the simple change here is the with each iteration the sampling is randomised for better fitting the data and tuning the parameters and the model here also is performing similar to the old one and thus we can Say that the basic exploratory analysis of the data has solved a huge portion of underfitting of the mkodel for better performance and the hyperparameter tuning is only an end fix when the model performance is too much deviated for the expected figures the plot also shows that the model has performed equally good on both for the training and testing datasets.

In [59]:

```
final_test['Survived'] = log_clf.predict(final_test[Selected_features])
final_test['PassengerId'] = test_df['PassengerId']

submission = final_test[['PassengerId','Survived']]

submission.to_csv("submission.csv", index=False)

submission.tail()
```

Out[59]:

	PassengerId	Survived
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

The model is being used for the prediction of the survival status of all the passengers of whose the data model has never been observed and there is a combination of the data for onserving the the output for a much better understanding * the output is exported into a csv file in the working directory of the program with the tail function that is being useed to see the last five records of the data for a quick peek.