Gurjus Singh

October $4^{th}$, 2020

MSDS 422 – Practical Machine Learning

Assignment #3 Evaluating Classification Models

**Data Preparation, Exploration and Visualisation**

The dataset that we did our data analysis on this week was on the Titanic. Before diving into the dataset, it is important to understand the historical context of the Titanic. The Titanic was a ship that departed from South Hampton, England in the year 1912 [1]. It sank hours before reaching North America and Newfoundland by hitting an Iceberg [1]. With this in mind, it is time to prepare our dataset for our algorithms to learn on.

As before in other assignments, I had to load our dataset which will allow us to manipulate the data using Python. The data was already split into train and training sets which was convenient for this assignment. After loading, I wanted to look at the initial data to get a feel for the data types and variables. I first noticed that this data set had missing values specifically in "Cabin" which I saw in 1-1 and 1-2. I also noticed there was a "Passenger ID" which was used to distinguish passengers. I also noticed a "Name" Column, and a "Sex" Column which shows gender of each passenger. I also noticed "Age" Colum, age of each passenger, "Fare" which is how much each passenger, "Cabin" which is where they resided on the ship and Embarked which is where they got on the ship. I looked up what "SibSp" and "Parch" meant in on Kaggle which was the source of my dataset and it mentioned that "SibSp" represents count of spouses and siblings of passenger while "Parch" was basically a count of the number of Parents and children per passenger [2].

After looking at the initial data set, I then decided to make sure that all the data column types matched from what I inferred from looking at the head of each data set.

| | 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 |

*Training Set Table Head 1-1*

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

*Test Set Table Head 1-2*

```
Out[63]: PassengerId      int64
         Survived         int64
         Pclass           int64
         Name            object
         Sex             object
         Age            float64
         SibSp            int64
         Parch            int64
         Ticket          object
         Fare           float64
         Cabin           object
         Embarked        object
         dtype: object
```

```
ut[64]: PassengerId      int64
        Pclass           int64
        Name            object
        Sex             object
        Age            float64
        SibSp            int64
        Parch            int64
        Ticket          object
        Fare           float64
        Cabin           object
        Embarked        object
        dtype: object
```

*Data Types 1-3*

I noticed that most of the types was exactly what I inferred, but the columns that contained strings were actually of object types as seen in 1-3. I then look at the shape of each data set as seen in 1-4 and one column was missing which was “Survived” column from our test set. This is the column which will be the column we want to predict using our classification models. Specifically, this is our response variable.

```
Out[11]: (891, 12)
```

```
Out[10]: (418, 11)
```

*Training and Test Data Shapes 1-4*

```
In [14]: train_df.describe()
Out[14]:
```

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

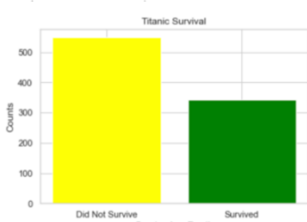
```
In [15]: test_df.describe()
Out[15]:
```

| | PassengerId | Pclass | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|
| count | 418.000000 | 418.000000 | 332.000000 | 418.000000 | 418.000000 | 417.000000 |
| mean | 1100.500000 | 2.265550 | 30.272590 | 0.447368 | 0.392344 | 35.627188 |
| std | 120.810458 | 0.841838 | 14.181209 | 0.896760 | 0.981429 | 55.907576 |
| min | 892.000000 | 1.000000 | 0.170000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 996.250000 | 1.000000 | 21.000000 | 0.000000 | 0.000000 | 7.895800 |
| 50% | 1100.500000 | 3.000000 | 27.000000 | 0.000000 | 0.000000 | 14.454200 |
| 75% | 1204.750000 | 3.000000 | 39.000000 | 1.000000 | 0.000000 | 31.500000 |
| max | 1309.000000 | 3.000000 | 76.000000 | 8.000000 | 9.000000 | 512.329200 |

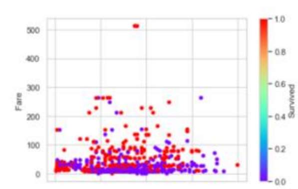
*Basic Stats for Each Data Set 1-5*

I then wanted to find out initial statistics on the data. I know the stats for “Pclass”, “Survived”, “Passenger Id” were not useful in our case. The “Fare”, “SibSp”, “Parch”, and “Age” column were useful as we could do statistical computations on them. The amazing thing that I found was that most of the people were around age 30 on the Titanic. I thought most of the people would be older around 50-80. This was some good insight.

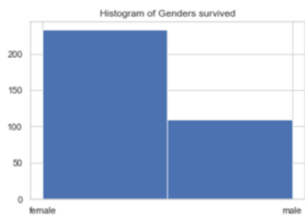
I then put together some initial EDAs of the training set to see if there was anything useful. I did a barplot to see if there was big difference between survived class and not survived class. What I found is more people died than survived in my results as shown in 1-6. Next I wanted to make a scatter plot to see if there was any connection between variables such as “sex”, “pclass”, “age”, “fare” and “Survival”. What I found was there was a trend between “fare” and survival which shows that the majority of the people who were paid higher fares survived while majority of lower fares did not survive in scatter plot 1-7. In 1-8 I found out that more females than males survived which was an interesting find through the data. I then wanted to see one last EDA to see if there was any finding between “pclass” and survival. What I noticed out of all those people that survived Pclass1 and Pclass3 had the most passengers that survived as shown in 1-9.



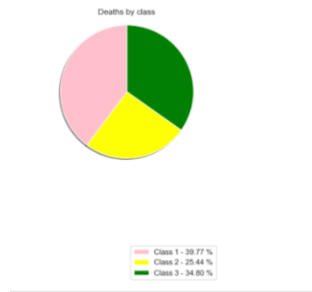

*Barplot 1-6*




*Scatterplot 1-7*

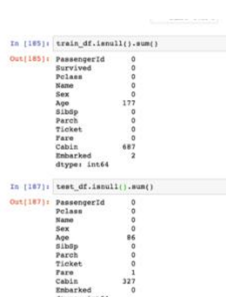


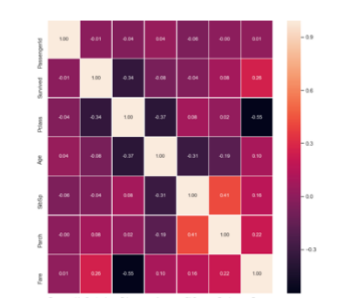

*Histogram 1-8*




*Piechart of Classes and Survival Rate 1-9*

After doing initial EDA, I had to make sure everything was ready for classification modeling. What I noticed is that some columns in the training set and test set had NA values as shown in 1-10. What I noticed was “Age” was missing a huge chunk of data, as well as “Cabin”. One way to deal with to deal with age is to use K Nearest Neighbors. This involves finding a grouping of variables that are correlated with “Age”. In this case I found using the heat map 1-11 that there was a somewhat large negative correlation between “Age” and “Sibsp”, “Pclass variables. With this in mind I found median Age of each subgroup of “Age”, “Sibsp” and “Pclass” and I imputed the age based on that. This took care of Age imputation. Next to impute Cabin, I made up a new class which I called “O”. I then transformed the Cabin column by only storing the first character which is the letter. I then saw that there missing values in “Embarked” in the train data and “Fare” in the test data. I took care of Embarked by finding the mode of the column which was “S” and imputed using mode, while for Fare I took the mean imputation method.



*Missing Values 1-10* *Heat Map 1-11*

After imputation, it was time to convert the object columns to numerical. This was done by creating separate functions to deal with encoding. For each column except for "Sex" such as "Embarked", "Cabin" I made N-1 columns. After doing encoding, I did feature engineering creating a relevant variable that could be better use. This column was created by taking "Sibsp" and "Parch" and summing them to find the total family members on board for a given passenger. Once I did all this it was time to start model preparation phase.

## Review research design and modeling methods

The three models we will be using for prediction is the Logistic Regression, Support Vector Machines, and KNeighbors Classifiers. Logistic Regression is used to estimate the probability that a given instance belongs to a class [3]. The way this works is a sigmoid function is used to give a probability between 0 and 1 based on the features from the training set. From there the data scientist uses a cutoff to classify each instance [3]. In Support Vector Machines, we have a margin which can be thought as of the flexibility to errors allowed [3]. This can be thought of as overfitting in linear regression if the margin is small as possible [3]. Support Vector Machines try to find the maximum margin that can correctly classify instances [3]. Support Vector Machines can also be used for nonlinear regression [3]. Lastly KNeighbors Classifier is an unsupervised classifier compared to the other two [4]. The way it works is the programmer inputs a particular parameter to the method which will then find the smallest distance between a certain number of defined instances in certain classes and the new instance [4]. Whichever defined instance is closest in distance/features to the new instance will be the class of the new instance [4].

I think it is important to define several models/algorithms, so a data scientist knows how they work. It is also important to use several models to get an idea of one being better over the other. I also think that several of these methods can be combined to get one score known as Ensembled methods.

Before running each model it is important to drop features that are unimportant in prediction or cannot be used and in this case “Passenger ID”, “Name”, “Parch”, “SibSp” which were used in feature transformation, “Cabin” and “Embarked” which we converted to numerical. Once we have dropped these features, we can then split the train dataset to 80% train and 20% test. I cannot use the original test data in my model since it did not come with the response column.

**Review results, evaluate models**

After implementing the models, it was time to see their results. Using the train data called x_train and y_train the AUC score was highest for KNN at 0.872 and was lowest for Support Vector Machines at 0.733. The Accuracy was highest for Logistic Regression and lowest for Support Vector Machines as well as shown in 1-12. I wanted to see if the same model was best for the test data. As seen in 1-13 on the test data, the highest AUC score came from KNNs, while the lowest was from Support Vector Machines. The accuracy score was lowest for Support Vector Machines, while the highest was for Logistic Regression. In my opinion I believe Linear Regression was the best model as it had high Accuracy as well as a high AUC score. I have learned we cannot rely purely on AUC score so that is why Linear Regression performs the best overall.

Out[1144]:

| | Classifier | Accuracy | AUC |
|---|---|---|---|
| 0 | Logistic Regression | 0.796405 | 0.833150 |
| 1 | Support Vector Machine | 0.655865 | 0.733648 |
| 2 | K-Nearest Neighbours | 0.703753 | 0.872062 |

Out[1127]:

| | Classifier | Accuracy | AUC |
|---|---|---|---|
| 0 | Logistic Regression | 0.776190 | 0.892358 |
| 1 | Support Vector Machine | 0.703016 | 0.797760 |
| 2 | K-Nearest Neighbours | 0.764603 | 0.905599 |

*Train Data Results 1-12* *Test Data Results 1-13*

After going over the overall models, I looked at the parameters for logistic regression which helped fit the model to both the train data and test data. What I found was these two formulas computed for the train data from 1-14 and the test data from 1-15 :

```
[[-0.76052213 -2.53601257 -0.03843836  0.00352584  0.57820546  0.25173305
  -0.08023567  0.79841343  1.53569731  0.88820831 -0.01408867 -0.11883063
  -0.28490761  0.02850785 -0.21112623]]
```

```
[4.25970065]
```

*Train Data Coefficients and Intercept 1-14*

```
[[-0.65354275 -2.39527367 -0.01744501  0.00883051  0.          0.82555266
   0.53338634  1.25170458 -0.0621017   0.70826016 -0.3959501   0.
   0.30292944  1.017721   -0.21256538]]
```

```
[2.4272679]
```

*Test Data Coefficients and Intercept 1-15*

Train Data Formula: 4.26 – 0.76*Pclass-2.53*Sex – 0.04*Age + 0.004*Fare + 0.57*CabA + 0.25*CabB -0.08*CabC +0.79*CabD + 1.5*CabE + 0.89*CabF – 0.0141*CabG – 0.12*CabT – 0.285*EmbS + 0.0285*EmbC -0.211*FamilySize

Test Data Formula: 2.42 – 0.65*Pclass-2.4*Sex – 0.02*Age + 0.009*Fare + 0*CabA + 0.8*CabB + 0.53*CabC + 1.25*CabD -0.06*CabE + 0.71*CabF – 0.39*CabG – 0*CabT – 0.303*EmbS + 1.02*EmbC - 0.213*FamilySize

From looking at the formulas above, the  Logistic Regression model on the train data thinks that the most important feature is “Sex”. “Sex” has 1 for every passenger that is a male and 0 for every person that is a female. The coefficient is -2.53 which indicates that females have 8% more of surviving than males since by taking the exponential function of -2.53 one will get 0.08 which is 8 percent. Comparing the odds of survival by gender on the test set, the model says that males’ survival rate is 9 less than females. Secondly “Pclass” also has an influence on the model and it says for every increase in the variable the chance of survival decreases by 46.7

percent. While for the test set the odds of survival decreases by 52.2 percent. Lastly "Family Size" seems to be also contributing to the model. The model says for the training set that for every increase in family size there is an 80 percent decrease in odds of survival and similarly for the test set the odds of survival decreases by 80 percent for increase in family variable. With Fare, the odds of survival actually increase 100% with every increase in Fare.

Lastly, I want to look at results from the confusion matrices. What the matrices told me is that the logistic regression model seemed to perform better and predict less instances as false positives and negatives. I looked at this for both data sets, but usually we want to see performance on the test set.

```
In [1166]: confusion_matrix_model(log_reg, x_test, y_test)
Out[1166]:
```

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.89 | 0.11 |
| Actual Survived | 0.25 | 0.75 |

```
In [1167]: confusion_matrix_model(SVC_rbf, x_test, y_test)
Out[1167]:
```

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.95 | 0.05 |
| Actual Survived | 0.62 | 0.38 |

```
In [1168]: confusion_matrix_model(KNN, x_test, y_test)
Out[1168]:
```

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.87 | 0.13 |
| Actual Survived | 0.42 | 0.58 |

*Confusion matrix for Test set 1-16*

**Implementation and programming**

For implementation I first imported the packages as seen in 1-17 in the such as numpy for using the mean function, pandas for dataframe methods such as groupby, sklearn for the classification  models, matplotlib which I used to plot several of the EDA plots.

*Code 1-17 Imports*

```
import numpy as np
import pandas as pd

import statsmodels.formula.api as sm
from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import make_scorer, accuracy_score, roc_auc_score, roc_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

import scikitplot as skplt
import seaborn as sns
from matplotlib import pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')
sns.set(style='whitegrid', color_codes=True)
plt.rc('font', size=14)
```

I had to load the data using the **Pandas.read_csv()** function. After loading the packages and data, I used **.head(), describe(),** and **.dtypes** to get general information about the two datasets. One important thing to do in data prep is to also see if we have NA values. Specifically, the code to do this is **.isnull().sum()** to find the missing values in each column. I then used different EDA methods to show the data using matplotlib package. The methods I used were **.scatter()** which was used to create the scatter plot in 1-7. I used **.bar()** method for creating the barplot 1-6 and .**hist()** to create the histogram. The piechart was created was created by creating three ratio for each class and passengers that survived. I then used the **plt.pie** function to create the pie chart as shown below. For imputation techniques to handle NA values I had to use **groupby** in the Pandas package as it was important to group by columns which were highly correlated with age to find a trend. I then added **.median()** on to the group by to find the median of each subgroup for age. I then used the function **.fillna()** to fill in the specific age columns. This method was used in the other imputations. Before imputing Cabin, I did not need the whole sequence only the first character which was a letter, in order to get only the first character it could be done with the **lambda** function using indexing passed into the **map** method

For feature transformation, I used the add + expression to add two columns together which would be used to find family members of each passenger on board the Titanic. After Feature transformation, I encoded the columns, using **np.where()** to find columns that matched the specific Boolean expression such as for **cabin = = A, 1, 0.** This expression specifically meant to return 1 in the output encoded column if the value was equal to 1. This was done for columns such as Embarked, Cabin and I had only N-1 column for encoded N values. The Sex column was also encoded but did not need N-1 columns for N values since it was already binary between Male and Female. After imputation, feature transformation and encoding, I had to drop irrelevant columns using .**drop()** column. After dropping non-relevant columns such as nonnumerical columns, and untransformed columns, I then split the train set specifically using **.split().** I also had to save the target variable which was “Survived” in its separate dataframe. I then started the training using Logistic Regression, SVC and KNN. I had to first call the instance of each model and use the **.fit()** method on each. After using the **.fit()** method, I then computed the AUC and Accuracy scores for each method which each required a custom-made method. Accuracy is simply defined as predicted correctly/total predicted while AUC is the area under ROC curve which is the curve that is computed by True Positive rate by False Positive rate [3]. I used the confusion matrix custom made function to find the confusion matrix.

## Exposition, problem description, and management recommendations

The overall goal of this analysis was to find a way to predict the “Survived” response variable on several key features. When thinking about the features involved, I think the obvious relevant features in the dataset were “Sex” which had the biggest negative coefficient in 1-14, 1-15, then came “Pclass” and lastly “Family Size”. After evaluating each model, I would recommend Logistic Regression for predicting the survivors of the Titanic because when looking at the results from 1-12 and 1-13, Logistic has both AUC and Accuracy scores that are not too low and not too high which shows that the model is not overfitting such as “too good to be true” and not very bad. The other models specifically seem too high and too low specifically AUC socre for KNN is around 0.9 which is what you want, but I think that is too high to my liking as the Accuracy score is around 0.76, while for Support Vector Machine method the Accuracy score is too low at 0.70. For Logistic Regression, I think both are right where you want them not too high and not too low.  Overall in looking at ways to improve the model, I think we need to look at other features, maybe there are attributes that contributed to why more Females survived, an underlying correlation?

References


[1] https://www.history.com/topics/early-20th-century-us/titanic#:~:text=The%20RMS%20Titanic%2C%20a%20luxury,their%20lives%20in%20the%20disaster.

[2] https://www.kaggle.com/c/titanic/data

[3] Géron, A. *Hands-On Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2d Edition. Sebastopol, Calif.: O'Reilly. [ISBN 9781492032649], 2019.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

# Appendix

# Import packages

In [996]:

```
import numpy as np
import pandas as pd

import statsmodels.formula.api as sm
from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import make_scorer, accuracy_score, ro
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

import scikitplot as skplt
import seaborn as sns
from matplotlib import pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")
sns.set(style="whitegrid", color_codes=True)
plt.rc("font", size=14)
```

In [997]:

```
%matplotlib inline
```

In [998]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [999]:

```
#read in datasets
urltest = 'https://raw.githubusercontent.com/djp840/MSDS_42
test_df=pd.read_csv(urltest)

urltrain = 'https://raw.githubusercontent.com/djp840/MSDS_4
train_df=pd.read_csv(urltrain)
```

In [1000]:

```
#Check Heads of Both Datasets
test_df.head()
```

Out[1000]:

| | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch |
|---|---|---|---|---|---|---|---|
| 0 | 892 | 3 | Kelly, Mr. James | male | 34.5 | 0 | 0 |
| 1 | 893 | 3 | Wilkes, Mrs. James (Ellen Needs) | female | 47.0 | 1 | 0 |
| 2 | 894 | 2 | Myles, Mr. Thomas Francis | male | 62.0 | 0 | 0 |
| 3 | 895 | 3 | Wirz, Mr. Albert | male | 27.0 | 0 | 0 |
| 4 | 896 | 3 | Hirvonen, Mrs. Alexander (Helga E Lindqvist) | female | 22.0 | 1 | 1 |

In [1001]:

```
train_df.head()
```

Out[1001]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibS |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | |

In [1156]:

```
#check types
train_df.dtypes
```

Out[1156]:

```
Survived         int64
Pclass           int64
Sex              int64
Age            float64
Fare           float64
Cabin A          int64
Cabin B          int64
Cabin C          int64
Cabin D          int64
Cabin E          int64
Cabin F          int64
Cabin G          int64
Cabin T          int64
Embarked S       int64
Embarked C       int64
Fammemb          int64
dtype: object
```

In [1003]:

```
test_df.dtypes
```

Out[1003]:

```
PassengerId      int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [1004]:

```
#see shape
test_df.shape
```

Out[1004]:

```
(418, 11)
```

In [1005]:

```
train_df.shape
```

Out[1005]:

```
(891, 12)
```

In [1006]:

```
#see if there are NA values for both test and train
train_df.isnull().sum()
```

Out[1006]:

```
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
```

In [1007]:

```
#see if there are NA values for both test and train
test_df.isnull().sum()
```

Out[1007]:

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

In [1008]:

```
#check summary stats for both
train_df.describe()
```

Out[1008]:

| | PassengerId | Survived | Pclass | Age | S |
|---|---|---|---|---|---|
| count | 891.000000 | 891.000000 | 891.000000 | 714.000000 | 891.00 |
| mean | 446.000000 | 0.383838 | 2.308642 | 29.699118 | 0.52 |
| std | 257.353842 | 0.486592 | 0.836071 | 14.526497 | 1.10 |
| min | 1.000000 | 0.000000 | 1.000000 | 0.420000 | 0.00 |
| 25% | 223.500000 | 0.000000 | 2.000000 | 20.125000 | 0.00 |
| 50% | 446.000000 | 0.000000 | 3.000000 | 28.000000 | 0.00 |
| 75% | 668.500000 | 1.000000 | 3.000000 | 38.000000 | 1.00 |
| max | 891.000000 | 1.000000 | 3.000000 | 80.000000 | 8.00 |

In [1009]:

```
test_df.describe()
```

Out[1009]:

| | PassengerId | Pclass | Age | SibSp | P |
|---|---|---|---|---|---|
| count | 418.000000 | 418.000000 | 332.000000 | 418.000000 | 418.00 |
| mean | 1100.500000 | 2.265550 | 30.272590 | 0.447368 | 0.39 |
| std | 120.810458 | 0.841838 | 14.181209 | 0.896760 | 0.98 |
| min | 892.000000 | 1.000000 | 0.170000 | 0.000000 | 0.00 |
| 25% | 996.250000 | 1.000000 | 21.000000 | 0.000000 | 0.00 |
| 50% | 1100.500000 | 3.000000 | 27.000000 | 0.000000 | 0.00 |
| 75% | 1204.750000 | 3.000000 | 39.000000 | 1.000000 | 0.00 |
| max | 1309.000000 | 3.000000 | 76.000000 | 8.000000 | 9.00 |

In [1010]:

```
#make barplot
plt.bar(['Did Not Survive','Survived'],[train_df['Survived'
                                         train_df['Survived'
       color = ['yellow', 'green'])
plt.title('Titanic Survival')
plt.xlabel('Survived vs Death')
plt.ylabel('Counts')
```

Out[1010]:

```
Text(0, 0.5, 'Counts')
```

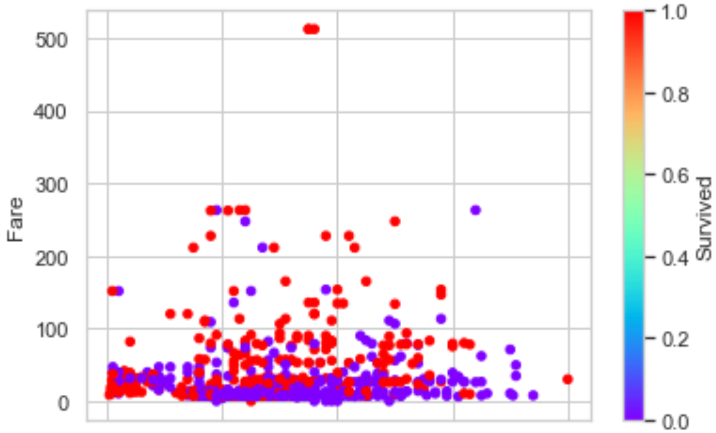
In [1011]:

```
#make scatterplot

train_df.plot.scatter('Age', 'Fare', c='Survived', cmap='ra
plt.xlabel('Age')
plt.ylabel('Fare')
```

Out[1011]:

```
Text(0, 0.5, 'Fare')
```

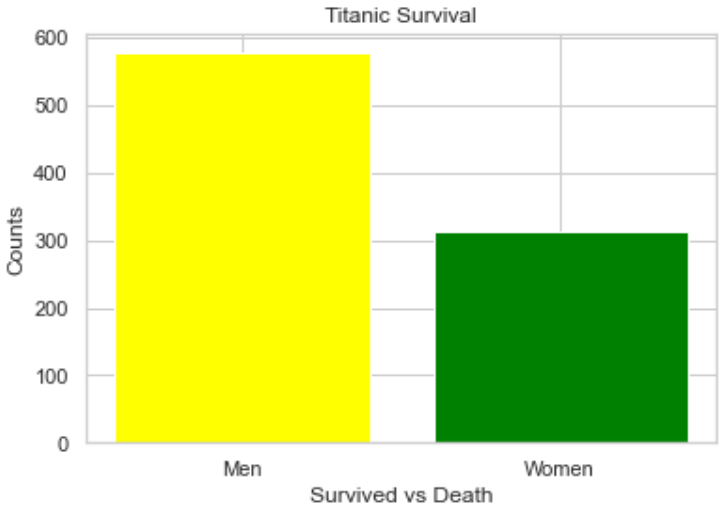


In [1012]:

```
#make barplot
plt.bar(['Men','Women'],[train_df.groupby('Sex').count()['S
                          train_df.groupby('Sex').count()['S
       color = ['yellow', 'green'])
plt.title('Titanic Survival')
plt.xlabel('Survived vs Death')
plt.ylabel('Counts')
```

Out[1012]:

```
Text(0, 0.5, 'Counts')
```

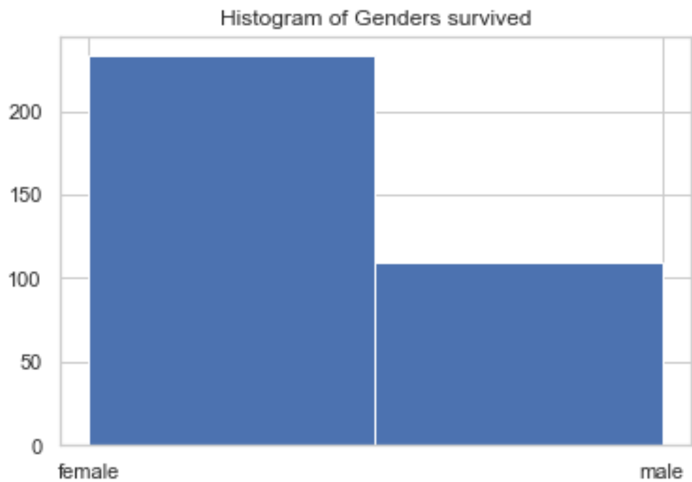


In [1013]:

```
#make histogram
plt.hist(train_df[train_df['Survived'] == 1]['Sex'], bins =
plt.title('Histogram of Genders survived')
```

Out[1013]:

```
Text(0.5, 1.0, 'Histogram of Genders survive
d')
```

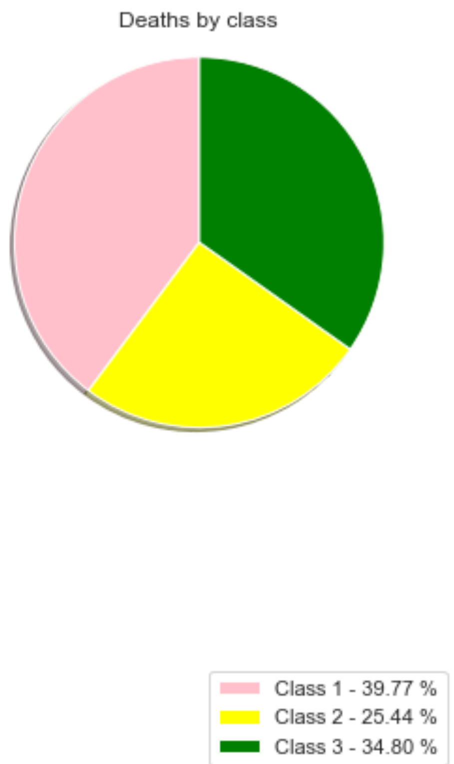



In [1014]:

```
#make pie chart/make ratios
class1 = train_df.groupby(['Survived', 'Pclass']).count()['
class2 = train_df.groupby(['Survived', 'Pclass']).count()['
class3 = train_df.groupby(['Survived', 'Pclass']).count()['
```

In [1015]:

```
x =  ['Class 1', 'Class 2', 'Class 3']
sizes = [class1, class2, class3]
percent = [class1*100, class2*100, class3*100]
colors = ['pink', 'yellow', 'green', 'blue', 'purple', 'red
explode = (0, 0, 0, 0, 0, 0,0,0,0,0)  # explode 1st slice
labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(x, 
# Plot
plt.title("Deaths by class")
patches, texts = plt.pie(sizes, colors=colors, shadow=True,
plt.legend(patches, labels, loc="lower left", bbox_to_ancho
plt.axis('equal')
plt.show()
```



In [1016]:

```
#check for NA values
train_df.isnull().sum()
```

Out[1016]:

```
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
```

In [1017]:

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

Out[1017]:

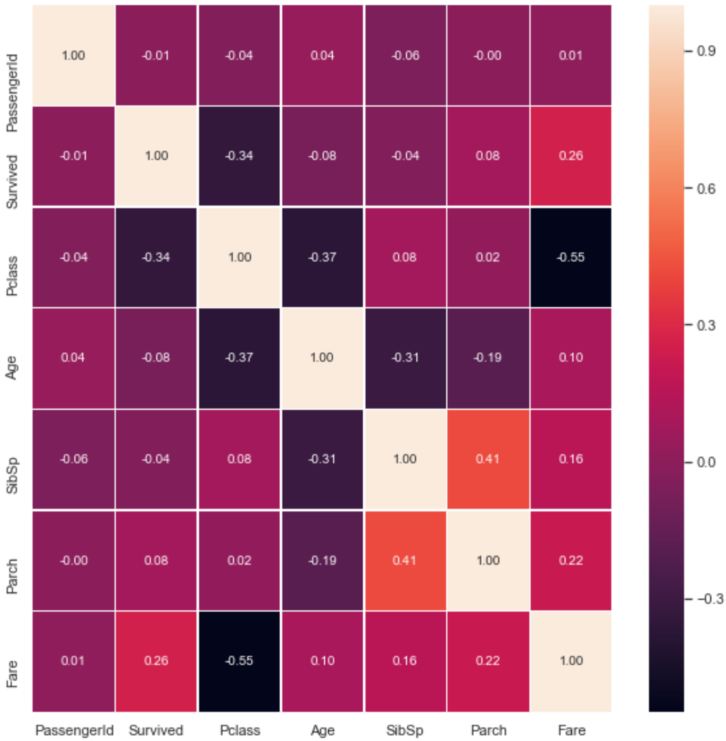
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [1018]:

```
#imputing age
f,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(train_df.corr(), annot=True, linewidths=0.5, fm
```

Out[1018]:

<AxesSubplot:>



In [1019]:

```
#compute median
medagetrain = train_df.groupby(['Pclass', 'SibSp']).median(
medagetrain
```

Out[1019]:

| Pclass | SibSp | PassengerId | Survived | Age | Parch | Fare |
|---|---|---|---|---|---|---|
| 1 | 0 | 476.0 | 1.0 | 37.0 | 0.0 | 39.6000 |
| | 1 | 485.0 | 1.0 | 38.0 | 0.0 | 79.2000 |
| | 2 | 572.0 | 1.0 | 44.0 | 0.0 | 133.6500 |
| | 3 | 89.0 | 1.0 | 23.0 | 2.0 | 263.0000 |
| 2 | 0 | 407.0 | 0.0 | 30.0 | 0.0 | 13.0000 |
| | 1 | 451.0 | 1.0 | 29.0 | 1.0 | 26.0000 |
| | 2 | 565.5 | 0.5 | 23.5 | 1.0 | 39.0000 |
| | 3 | 727.0 | 1.0 | 30.0 | 0.0 | 21.0000 |
| 3 | 0 | 472.0 | 0.0 | 26.0 | 0.0 | 7.8958 |
| | 1 | 372.0 | 0.0 | 25.0 | 0.0 | 15.5500 |
| | 2 | 334.0 | 0.0 | 19.5 | 0.0 | 19.2583 |
| | 3 | 302.5 | 0.0 | 6.0 | 1.0 | 25.4667 |
| | 4 | 264.5 | 0.0 | 6.5 | 1.5 | 31.2750 |
| | 5 | 387.0 | 0.0 | 11.0 | 2.0 | 46.9000 |
| | 8 | 325.0 | 0.0 | NaN | 2.0 | 69.5500 |

In [1020]:

```
#compute median of every column
medagetest = test_df.groupby(['Pclass', 'SibSp']).median()
medagetest
```

Out[1020]:

| Pclass | SibSp | PassengerId | Age | Parch | Fare |
|---|---|---|---|---|---|
| 1 | 0 | 1088.0 | 39.0 | 0.0 | 42.50000 |
| | 1 | 1109.5 | 46.0 | 0.0 | 82.06250 |
| | 2 | 969.0 | 55.0 | 0.0 | 51.47920 |
| | 3 | 945.0 | 28.0 | 2.0 | 263.00000 |
| 2 | 0 | 1117.5 | 27.0 | 0.0 | 13.00000 |
| | 1 | 1139.0 | 29.0 | 0.0 | 26.00000 |
| | 2 | 1077.5 | 21.0 | 0.5 | 31.50000 |
| 3 | 0 | 1095.5 | 24.0 | 0.0 | 7.82920 |
| | 1 | 1084.0 | 20.0 | 1.0 | 15.24580 |
| | 2 | 1059.0 | 19.5 | 0.0 | 21.67920 |
| | 3 | 1281.0 | 29.0 | 1.0 | 21.07500 |
| | 4 | 1076.0 | 11.5 | 2.0 | 30.25625 |
| | 5 | 1032.0 | 10.0 | 2.0 | 46.90000 |
| | 8 | 1166.0 | 14.5 | 2.0 | 69.55000 |

In [1021]:

```
#This function is a case of if else's to impute age by medi
#functions
def impute_age(dataset,dataset_med):
    for x in range(len(dataset)):
        if dataset["Pclass"][x]==1:
            if dataset["SibSp"][x]==0:
                return dataset_med.loc[1,0]["Age"]
            elif dataset["SibSp"][x]==1:
                return dataset_med.loc[1,1]["Age"]
            elif dataset["SibSp"][x]==2:
                return dataset_med.loc[1,2]["Age"]
            elif dataset["SibSp"][x]==3:
                return dataset_med.loc[1,3]["Age"]
        elif dataset["Pclass"][x]==2:
            if dataset["SibSp"][x]==0:
                return dataset_med.loc[2,0]["Age"]
            elif dataset["SibSp"][x]==1:
                return dataset_med.loc[2,1]["Age"]
            elif dataset["SibSp"][x]==2:
                return dataset_med.loc[2,2]["Age"]
            elif dataset["SibSp"][x]==3:
                return dataset_med.loc[2,3]["Age"]
        elif dataset["Pclass"][x]==3:
            if dataset["SibSp"][x]==0:
                return dataset_med.loc[3,0]["Age"]
            elif dataset["SibSp"][x]==1:
                return dataset_med.loc[3,1]["Age"]
            elif dataset["SibSp"][x]==2:
                return dataset_med.loc[3,2]["Age"]
            elif dataset["SibSp"][x]==3:
                return dataset_med.loc[3,3]["Age"]
            elif dataset["SibSp"][x]==4:
                return dataset_med.loc[3,4]["Age"]
            elif dataset["SibSp"][x]==5:
                return dataset_med.loc[3,5]["Age"]
            elif dataset["SibSp"][x]==8:
                return dataset_med.loc[3]["Age"].median()
```

In [1022]:

```
#Fill in NA for Age and
train_df['Age'] = train_df['Age'].fillna(impute_age(train_d
test_df['Age'] = test_df['Age'].fillna(impute_age(test_df, i
```

In [1023]:

```
#Check missing values again for both data sets; we see age
train_df.isnull().sum()
```

Out[1023]:

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

In [1024]:

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

Out[1024]:

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

In [1025]:

```
#Next we have to fill in missing values for cabin; we can f
train_df['Cabin'] = train_df['Cabin'].fillna('O')
test_df['Cabin'] = test_df['Cabin'].fillna('O')
```

In [1026]:

```
#We check again and we see Cabin has been filled
train_df.isnull().sum()
```

Out[1026]:

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       2
dtype: int64
```

In [1027]:

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

Out[1027]:

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

In [1028]:

```
#We can deal with Fare by using mean imputation to fill NA
test_df['Fare'] = test_df['Fare'].fillna(np.mean(test_df['F
```

In [1029]:

```
#test is all fixed
test_df.isnull().sum()
```

Out[1029]:

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

In [1030]:

```
#Now we deal with Embarked; we fill it with the mode of the
train_df['Embarked'] = train_df['Embarked'].fillna('S')
```

In [1031]:

```
#Next we want to fix Column Cabin to only show the number;
train_df["Cabin"]=train_df["Cabin"].map(lambda x: x[0])
test_df["Cabin"]=test_df["Cabin"].map(lambda x: x[0])
```

In [1032]:

```
train_df['Cabin'].head()
```

Out[1032]:

```
0    O
1    C
2    O
3    C
4    O
Name: Cabin, dtype: object
```

In [1033]:

```
#We see that has been fixed; encode features
def cabin_assignment(dataset):
    dataset["Cabin A"]=np.where(dataset["Cabin"]=="A",1,0)
    dataset["Cabin B"]=np.where(dataset["Cabin"]=="B",1,0)
    dataset["Cabin C"]=np.where(dataset["Cabin"]=="C",1,0)
    dataset["Cabin D"]=np.where(dataset["Cabin"]=="D",1,0)
    dataset["Cabin E"]=np.where(dataset["Cabin"]=="E",1,0)
    dataset["Cabin F"]=np.where(dataset["Cabin"]=="F",1,0)
    dataset["Cabin G"]=np.where(dataset["Cabin"]=="G",1,0)
    dataset["Cabin T"]=np.where(dataset["Cabin"]=="T",1,0)

def embark_assignment(dataset):
    dataset["Embarked S"]=np.where(dataset["Embarked"]=="S"
    dataset["Embarked C"]=np.where(dataset["Embarked"]=="C"


sex_map={"male":1,"female":0}
train_df["Sex"]=train_df["Sex"].map(sex_map)
test_df["Sex"]=test_df["Sex"].map(sex_map)
```

In [1034]:

```
#use functions on both test and train
cabin_assignment(train_df)
embark_assignment(train_df)
```

In [1035]:

```
cabin_assignment(test_df)
embark_assignment(test_df)
```

In [1036]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 22 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null int64
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          891 non-null object
Embarked       891 non-null object
Cabin A        891 non-null int64
Cabin B        891 non-null int64
Cabin C        891 non-null int64
Cabin D        891 non-null int64
Cabin E        891 non-null int64
Cabin F        891 non-null int64
Cabin G        891 non-null int64
Cabin T        891 non-null int64
Embarked S     891 non-null int64
Embarked C     891 non-null int64
dtypes: float64(2), int64(16), object(4)
memory usage: 153.3+ KB
```

In [1037]:

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 21 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null int64
Age            418 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           418 non-null float64
Cabin          418 non-null object
Embarked       418 non-null object
Cabin A        418 non-null int64
Cabin B        418 non-null int64
Cabin C        418 non-null int64
Cabin D        418 non-null int64
Cabin E        418 non-null int64
Cabin F        418 non-null int64
Cabin G        418 non-null int64
Cabin T        418 non-null int64
Embarked S     418 non-null int64
Embarked C     418 non-null int64
dtypes: float64(2), int64(15), object(4)
memory usage: 68.7+ KB
```

In [1038]:

```
#Make new feature which is total siblings, spouse, parents
train_df['Fammemb'] = train_df['SibSp'] + train_df['Parch']
test_df['Fammemb'] = test_df['SibSp'] + test_df['Parch'] +
```

In [1039]:

```
#We are now ready for training; We should drop off features
train_df.drop(["Name","Ticket","PassengerId","Embarked","Ca
test_df.drop(["Name","Ticket","Embarked","Cabin","SibSp","P
```

In [1040]:

```
#see end of data set
train_df.tail()
```

Out[1040]:

| | Survived | Pclass | Sex | Age | Fare | Cabin A | Cabin B | Cabin C |
|---|---|---|---|---|---|---|---|---|
| 886 | 0 | 2 | 1 | 27.0 | 13.00 | 0 | 0 | 0 |
| 887 | 1 | 1 | 0 | 19.0 | 30.00 | 0 | 1 | 0 |
| 888 | 0 | 3 | 0 | 25.0 | 23.45 | 0 | 0 | 0 |
| 889 | 1 | 1 | 1 | 26.0 | 30.00 | 0 | 0 | 1 |
| 890 | 0 | 3 | 1 | 32.0 | 7.75 | 0 | 0 | 0 |

In [1041]:

```
#make copy
training_df1=train_df.copy()
test_df1=test_df.copy()
```

In [1042]:

```
#Next save response variable and drop from training set
x = training_df1.drop(['Survived'], 1)
y = training_df1["Survived"]
```

In [1043]:

```
x.shape, y.shape
```

Out[1043]:

```
((891, 15), (891,))
```

In [1044]:

```
#we want to split training data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_siz

x_train.shape, x_test.shape
```

Out[1044]:

```
((712, 15), (179, 15))
```

In [1045]:

```
#check head after split
x_train.head()
```

Out[1045]:

| | Pclass | Sex | Age | Fare | Cabin A | Cabin B | Cabin C | Cabin D |
|---|---|---|---|---|---|---|---|---|
| 140 | 3 | 0 | 25.0 | 15.2458 | 0 | 0 | 0 | 0 |
| 439 | 2 | 1 | 31.0 | 10.5000 | 0 | 0 | 0 | 0 |
| 817 | 2 | 1 | 31.0 | 37.0042 | 0 | 0 | 0 | 0 |
| 378 | 3 | 1 | 20.0 | 4.0125 | 0 | 0 | 0 | 0 |
| 491 | 3 | 1 | 21.0 | 7.2500 | 0 | 0 | 0 | 0 |

In [1046]:

```
k_fold = KFold(n_splits=5, shuffle=True, random_state=0)
```

In [1088]:

```
#custom functions
def acc_score(model, x_train, y_train):
    return np.mean(cross_val_score(model,x_train,y_train,cv

def confusion_matrix_model(model_used, x_test, y_test):
    cm=confusion_matrix(y_test,model_used.predict(x_test))
    col=["Predicted Dead","Predicted Survived"]
    cm=pd.DataFrame(cm)
    cm.columns=["Predicted Dead","Predicted Survived"]
    cm.index=["Actual Dead","Actual Survived"]
    cm[col]=np.around(cm[col].div(cm[col].sum(axis=1),axis=
    return cm
def importance_of_features(model):
    features = pd.DataFrame()
    features['feature'] = x_train.columns
    features['importance'] = model.feature_importances_
    features.sort_values(by=['importance'], ascending=True,
    features.set_index('feature', inplace=True)
    return features.plot(kind='barh', figsize=(10,10))
```

In [1101]:

```
def aucscore(model,x_test, y_test, has_proba=True):
    if has_proba:
        fpr,tpr,thresh=skplt.metrics.roc_curve(y_test,model
    else:
        fpr,tpr,thresh=skplt.metrics.roc_curve(y_test,model
    x=fpr
    y=tpr
    auc= skplt.metrics.auc(x,y)
    return auc
def plt_roc_curve(name,model,x_test, y_test, has_proba=True
    if has_proba:
        fpr,tpr,thresh=skplt.metrics.roc_curve(y_test,model
    else:
        fpr,tpr,thresh=skplt.metrics.roc_curve(y_test,model
    x=fpr
    y=tpr
    auc= skplt.metrics.auc(x,y)
    plt.plot(x,y,label='ROC curve for %s (AUC = %0.2f)' % (
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim((0,1))
    plt.ylim((0,1))
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.legend(loc="lower right")
    plt.show()
```

In [1102]:

```
#time for training
log_reg=LogisticRegression()
log_reg.fit(x_train,y_train)
print("Accuracy: " + str(acc_score(log_reg, x_train, y_trai
confusion_matrix_model(log_reg, x_train, y_train)
```

```
Accuracy: 0.7964050034472571
```

Out[1102]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.86 | 0.14 |
| Actual Survived | 0.28 | 0.72 |

In [1103]:

```
#print formula coefficients as well as intercept
print(log_reg.coef_)
```

```
[[-0.76052213 -2.53601257 -0.03843836  0.00352
584   0.57820546  0.25173305
  -0.08023567  0.79841343  1.53569731  0.88820
831 -0.01408867 -0.11883063
  -0.28490761  0.02850785 -0.21112623]]
```

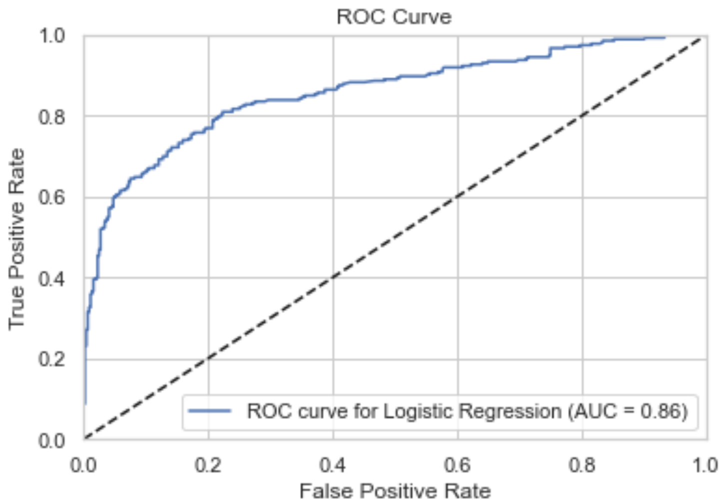
In [1104]:

```
print(log_reg.intercept_)
```

```
[4.25970065]
```

In [1109]:

```
#see ROC curve
plt_roc_curve("Logistic Regression", log_reg, x_train, y_tr
```




In [1110]:

```
print(log_reg.coef_)
```

```
[[-0.76052213 -2.53601257 -0.03843836  0.00352
584  0.57820546  0.25173305
  -0.08023567  0.79841343  1.53569731  0.88820
831 -0.01408867 -0.11883063
  -0.28490761  0.02850785 -0.21112623]]
```

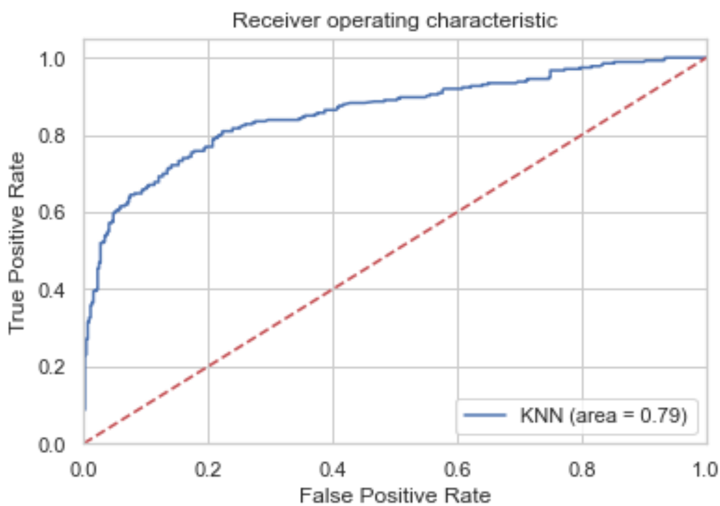
In [1111]:

```
print(log_reg.intercept_)
```

```
[4.25970065]
```

In [1113]:

```
logit_roc_auc = roc_auc_score(y_train, log_reg.predict(x_tr
fpr, tpr, thresholds = roc_curve(y_train, log_reg.predict_p
plt.figure()
plt.plot(fpr, tpr, label='KNN (area = %0.2f)' % logit_roc_a
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('KNN')
plt.show()
```

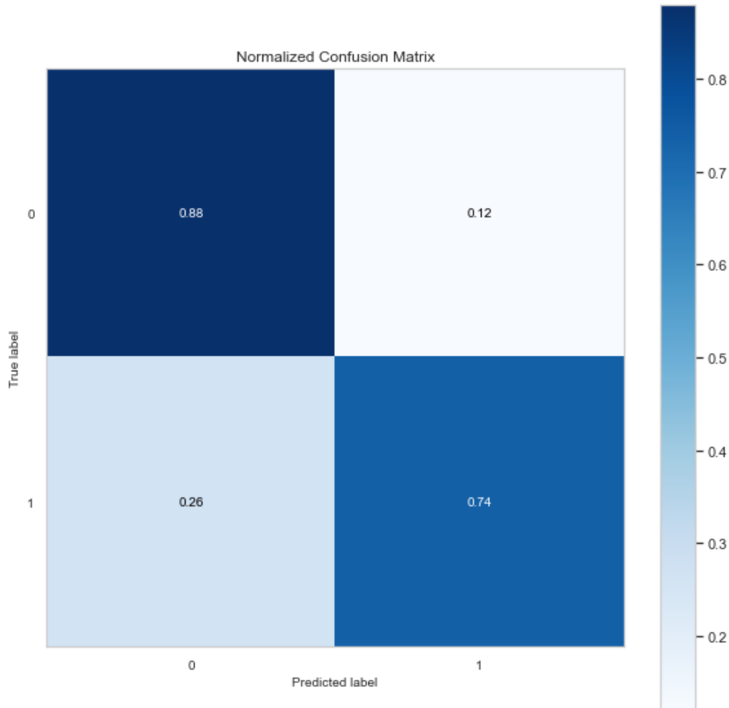


In [1057]:

```
skplt.metrics.plot_confusion_matrix(y_test, log_reg.predict
```

Out[1057]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```



In [1155]:

```
SVC_rbf=SVC(kernel="rbf")
SVC_rbf.fit(x_train,y_train)
print(aucscore(SVC_rbf, x_train, y_train,has_proba=False))
print("Accuracy: " + str(acc_score(SVC_rbf, x_train, y_trai
confusion_matrix_model(SVC_rbf, x_train, y_train)
```

```
0.7336479010738692
Accuracy: 0.6558652614990643
```

Out[1155]:

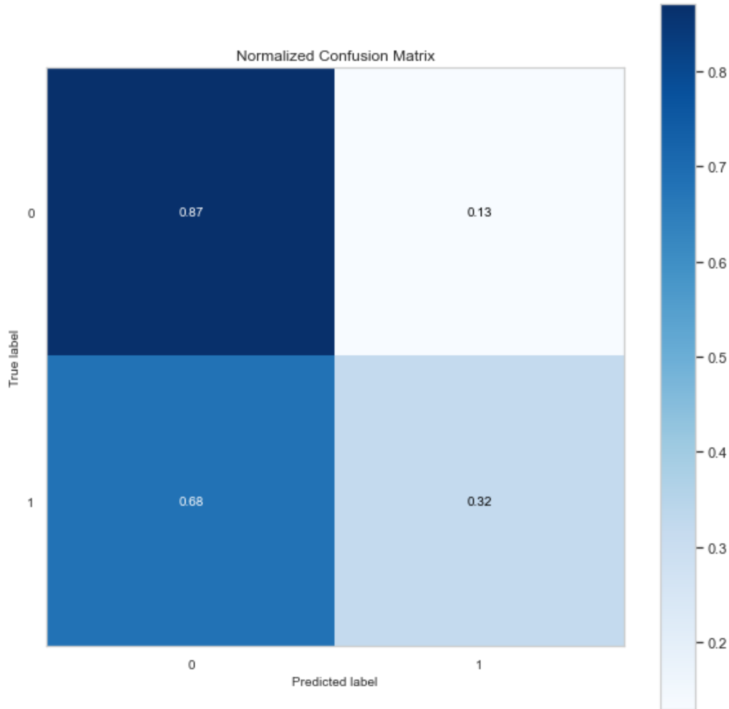
| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.92 | 0.08 |
| Actual Survived | 0.74 | 0.26 |

In [1136]:

```
skplt.metrics.plot_confusion_matrix(y_train, SVC_rbf.predic
```

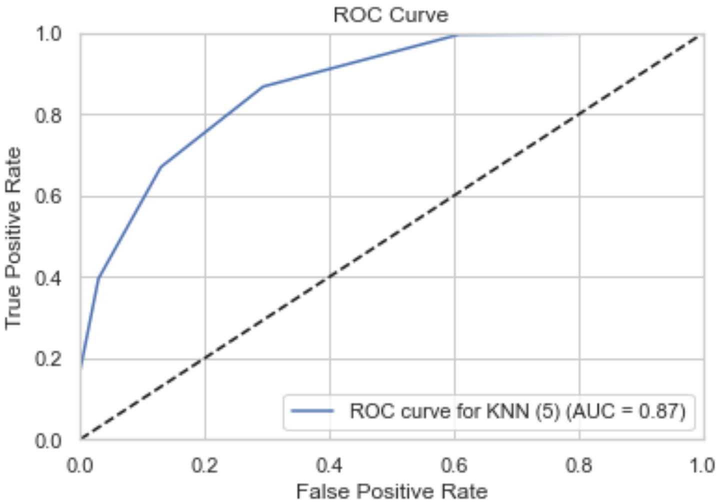
Out[1136]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```



In [1141]:

```
plt_roc_curve("KNN (5)" ,KNN,x_train, y_train,has_proba=Tru
```

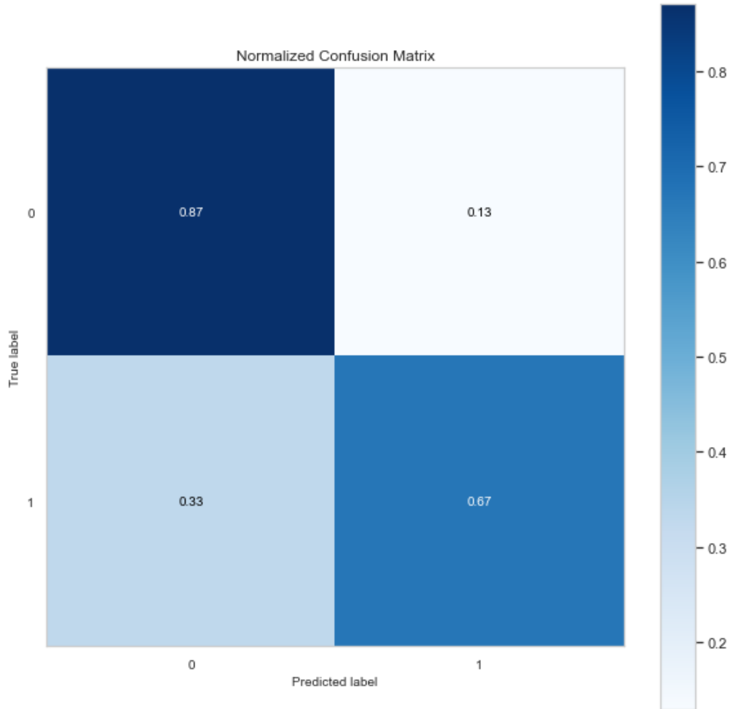


In [1134]:

```
skplt.metrics.plot_confusion_matrix(y_train, KNN.predict(x_
```

Out[1134]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```



In [1132]:

```
KNN=KNeighborsClassifier(n_neighbors=5)
KNN.fit(x_train,y_train)
print(aucscore(KNN, x_train, y_train, has_proba=True))
print("Accuracy: " + str(acc_score(KNN, x_train, y_train)))
confusion_matrix_model(KNN, x_train, y_train)
```

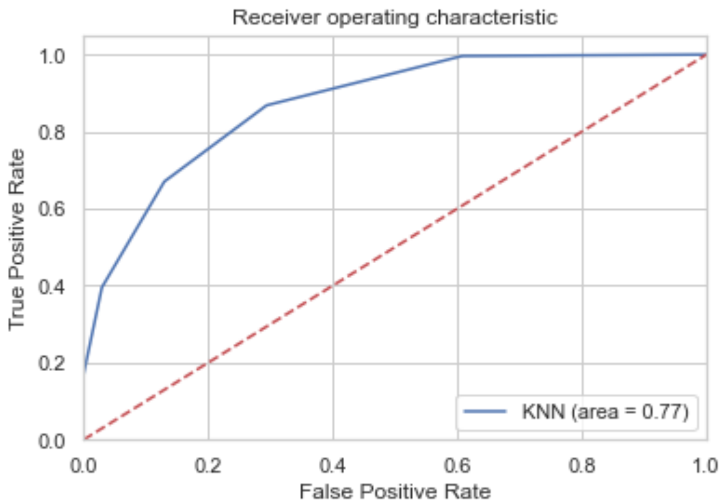
```
0.8720618788955918
Accuracy: 0.7037525854427262
```

Out[1132]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.87 | 0.13 |
| Actual Survived | 0.33 | 0.67 |

In [1143]:

```
logit_roc_auc = roc_auc_score(y_train, KNN.predict(x_train)
fpr, tpr, thresholds = roc_curve(y_train, KNN.predict_proba
plt.figure()
plt.plot(fpr, tpr, label='KNN (area = %0.2f)' % logit_roc_a
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('KNN')
plt.show()
```



In [1144]:

```
Classifiers=["Logistic Regression","Support Vector Machine"
Acc=[acc_score(x, x_train, y_train) for x in [log_reg,SVC_r
auc_scores_prob=[aucscore(x, x_train, y_train, has_proba=Tr
auc_scores_noprob=[aucscore(x, x_train, y_train, has_proba=
auc_scores= [auc_scores_prob[0],auc_scores_noprob[0], auc_s
cols=["Classifier","Accuracy","AUC"]
results = pd.DataFrame(columns=cols)
results["Classifier"]=Classifiers
results["Accuracy"]=Acc
results["AUC"]=auc_scores
results
```

Out[1144]:

| | Classifier | Accuracy | AUC |
|---|---|---|---|
| 0 | Logistic Regression | 0.796405 | 0.833150 |
| 1 | Support Vector Machine | 0.655865 | 0.733648 |
| 2 | K-Nearest Neighbours | 0.703753 | 0.872062 |

In [1127]:

```
#same thing for test data; see confusion matrix etc.
log_reg=LogisticRegression()
log_reg.fit(x_test,y_test)
print("Accuracy: " + str(acc_score(log_reg, x_test, y_test)
print(aucscore(log_reg, x_test, y_test, has_proba=True))
print(log_reg.coef_)
print(log_reg.intercept_)

SVC_rbf=SVC(kernel="rbf")
SVC_rbf.fit(x_test,y_test)
print(aucscore(SVC_rbf, x_test, y_test,has_proba=False))
print("Accuracy: " + str(acc_score(SVC_rbf, x_test, y_test)


KNN=KNeighborsClassifier(n_neighbors=5)
KNN.fit(x_test,y_test)
print(aucscore(KNN,x_test, y_test,has_proba=True))
print("Accuracy: " + str(acc_score(KNN, x_test, y_test)))


Classifiers=["Logistic Regression","Support Vector Machine"
Acc=[acc_score(x, x_test, y_test) for x in [log_reg,SVC_rbf
auc_scores_prob=[aucscore(x, x_test, y_test, has_proba=True
auc_scores_noprob=[aucscore(x,x_test, y_test, has_proba=Fal
auc_scores= [auc_scores_prob[0],auc_scores_noprob[0], auc_s
cols=["Classifier","Accuracy","AUC"]
results = pd.DataFrame(columns=cols)
results["Classifier"]=Classifiers
results["Accuracy"]=Acc
results["AUC"]=auc_scores
results
```

```
Accuracy: 0.7761904761904762
0.892358366271409
[[-0.65354275 -2.39527367 -0.01744501  0.00883
051  0.          0.82555266
   0.53338634  1.25170458 -0.0621017   0.70826
016 -0.3959501   0.
   0.30292944  1.017721   -0.21256538]]
[2.4272679]
0.797760210803689
Accuracy: 0.703015873015873
```

```
0.9055994729907773
Accuracy: 0.7646031746031746
```

Out[1127]:

| | Classifier | Accuracy | AUC |
|---|---|---|---|
| 0 | Logistic Regression | 0.776190 | 0.892358 |
| 1 | Support Vector Machine | 0.703016 | 0.797760 |
| 2 | K-Nearest Neighbours | 0.764603 | 0.905599 |

In [1130]:

```
print(log_reg.coef_)
```

```
[[-0.65354275 -2.39527367 -0.01744501  0.00883
051  0.          0.82555266
   0.53338634  1.25170458 -0.0621017   0.70826
016 -0.3959501   0.
   0.30292944  1.017721   -0.21256538]]
```

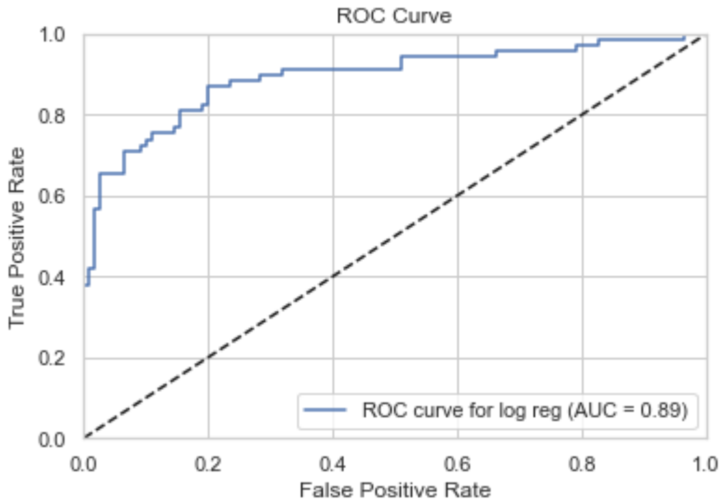
In [1131]:

```
print(log_reg.intercept_)
```

```
[2.4272679]
```

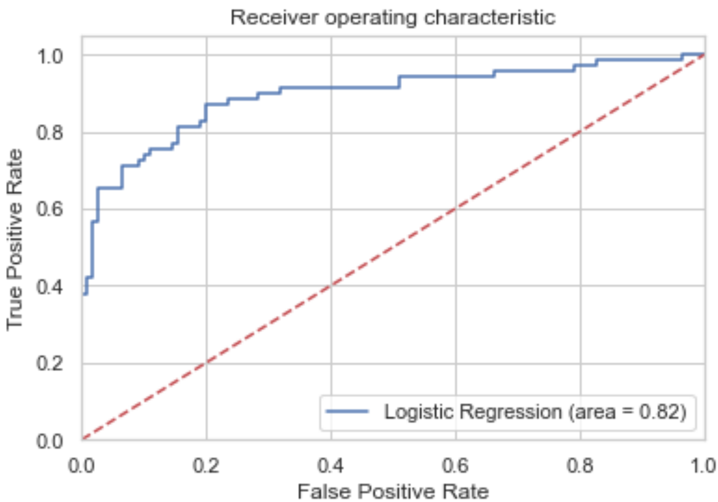
In [1148]:

```
plt_roc_curve("log reg",log_reg, x_test, y_test, has_proba=
```



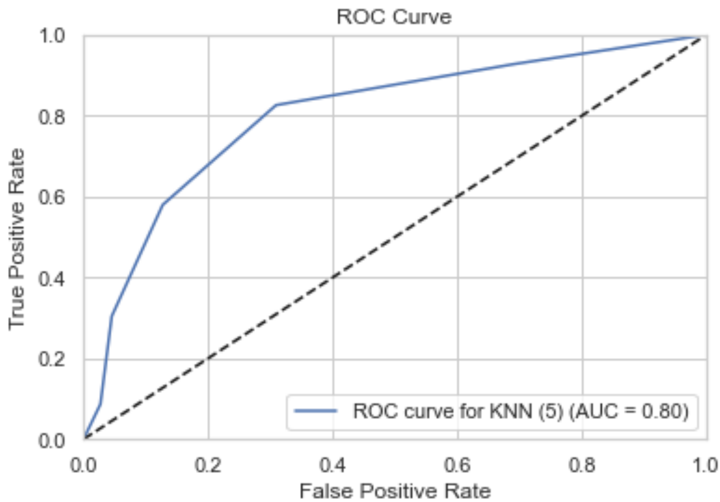
In [1149]:

```
logit_roc_auc = roc_auc_score(y_test, log_reg.predict(x_tes
fpr, tpr, thresholds = roc_curve(y_test, log_reg.predict_pr
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



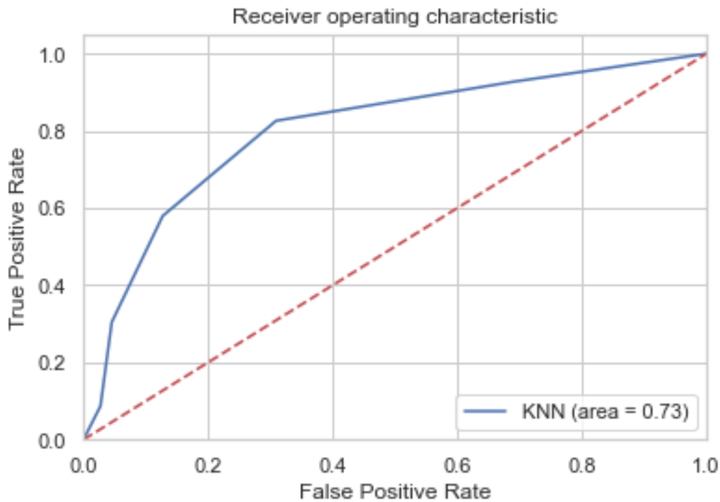
In [1145]:

```
#test ROC curve
plt_roc_curve("KNN (5)",KNN, x_test, y_test,has_proba=True)
```



In [1147]:

```
logit_roc_auc = roc_auc_score(y_test, KNN.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, KNN.predict_proba(:
plt.figure()
plt.plot(fpr, tpr, label='KNN (area = %0.2f)' % logit_roc_a
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('KNN')
plt.show()
```



In [1166]:

```
confusion_matrix_model(log_reg, x_test, y_test)
```

Out[1166]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.89 | 0.11 |
| Actual Survived | 0.25 | 0.75 |

In [1167]:

```
confusion_matrix_model(SVC_rbf, x_test, y_test)
```

Out[1167]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.95 | 0.05 |
| Actual Survived | 0.62 | 0.38 |

In [1168]:

```
confusion_matrix_model(KNN, x_test, y_test)
```

Out[1168]:

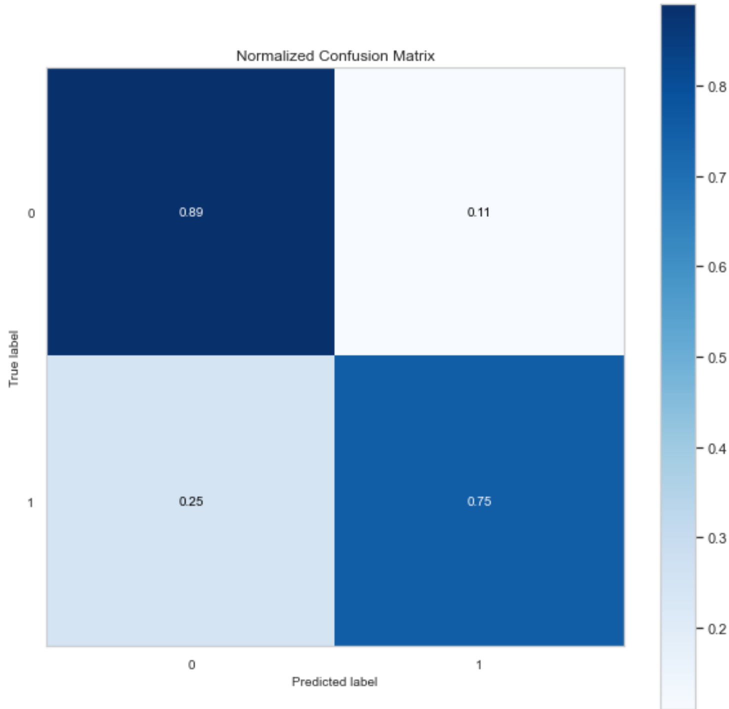
| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.87 | 0.13 |
| Actual Survived | 0.42 | 0.58 |

In [1076]:

```
skplt.metrics.plot_confusion_matrix(y_test, log_reg.predict
```

Out[1076]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```

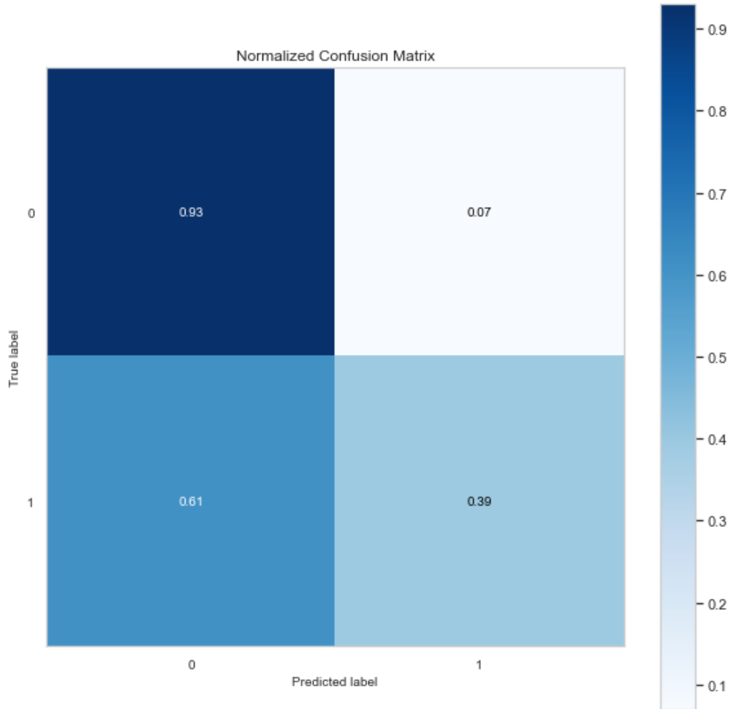


In [1077]:

```
skplt.metrics.plot_confusion_matrix(y_test, SVC_rbf.predict
```

Out[1077]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```

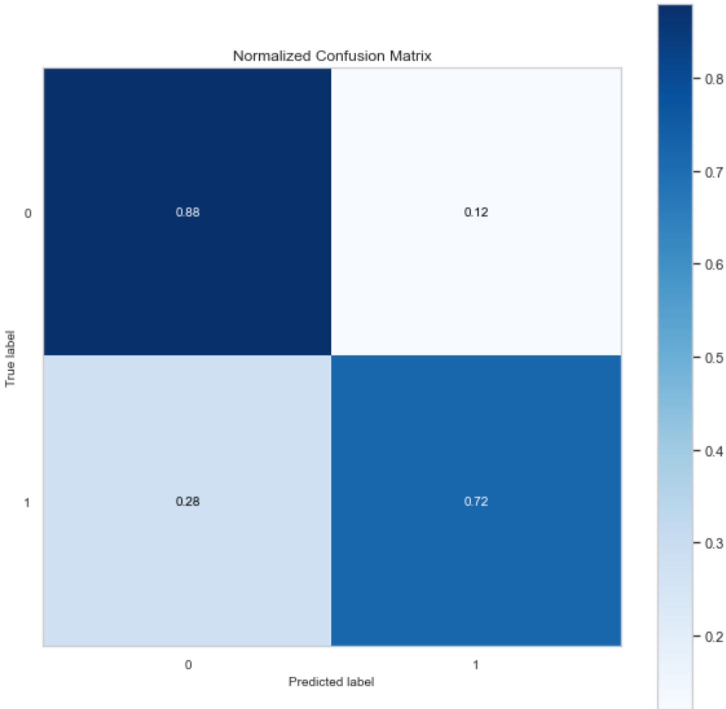


In [1078]:

```
skplt.metrics.plot_confusion_matrix(y_test, KNN.predict(x_t
```

Out[1078]:

```
<AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```



In [1157]:

```
#find right parameters for best results for lunear regressi
parameters = {"class_weight": ["None", "balanced"],
              "max_iter": [25,50,75,100],
              "penalty": ["l1", "l2", "elasticnet", "none"]
              "solver": ["newton-cg", "lbfgs", "liblinear",
             }
```

In [1080]:

```
grid_cv = GridSearchCV(log_reg, parameters, scoring = make_
grid_cv = grid_cv.fit(x_train, y_train)
```

In [1081]:

```
print("Our optimized Logistic Regression model is:")
grid_cv.best_estimator_
```

```
Our optimized Logistic Regression model is:
```

Out[1081]:

```
LogisticRegression(class_weight='None', max_it
er=50)
```

In [1082]:

```
logreg_clf_GSCV = LogisticRegression(C=1.0, class_weight='N
                   intercept_scaling=1, l1_ratio=None, max_
                   multi_class='auto', n_jobs=None, penalty
                   random_state=None, solver='lbfgs', tol=0
                   warm_start=False)
logreg_clf_GSCV.fit(x_train, y_train)
```

Out[1082]:

```
LogisticRegression(class_weight='None', max_it
er=50, penalty='none')
```

In [1159]:

```
print("Accuracy: " + str(acc_score(logreg_clf_GSCV, x_train
confusion_matrix_model(logreg_clf_GSCV, x_train, y_train)
```

Accuracy: 0.7851669457303261

Out[1159]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.86 | 0.14 |
| Actual Survived | 0.30 | 0.70 |

In [1161]:

```
grid_cv = GridSearchCV(log_reg, parameters, scoring = make_
grid_cv = grid_cv.fit(x_test, y_test)
```

In [1162]:

```
print("Our optimized Logistic Regression model is:")
grid_cv.best_estimator_
```

Our optimized Logistic Regression model is:

Out[1162]:

```
LogisticRegression(class_weight='None', max_it
er=75, penalty='none')
```

In [1163]:

```
logreg_clf_GSCV = LogisticRegression(C=1.0, class_weight='N
                    intercept_scaling=1, l1_ratio=None, max_
                    multi_class='auto', n_jobs=None, penalty
                    random_state=None, solver='lbfgs', tol=0
                    warm_start=False)
logreg_clf_GSCV.fit(x_train, y_train)
```

Out[1163]:

```
LogisticRegression(class_weight='None', max_it
er=75, penalty='none')
```

In [1165]:

```
print("Accuracy: " + str(acc_score(logreg_clf_GSCV, x_test,
confusion_matrix_model(logreg_clf_GSCV, x_test, y_test)
```

```
Accuracy: 0.7763492063492063
```

Out[1165]:

| | Predicted Dead | Predicted Survived |
|---|---|---|
| Actual Dead | 0.85 | 0.15 |
| Actual Survived | 0.26 | 0.74 |

In [ ]: