Gurjus Singh

October 11th, 2020

MSDS 422 – Practical Machine Learning

Assignment #4 Random Forests and Gradient Boosting PART B

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. Our goal was to see if our algorithms could classify who survived based on several features in our data.

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

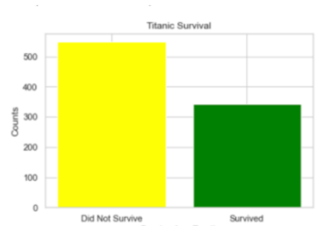
```
In [15]: test_df.describe()
Out[15]:
```

	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

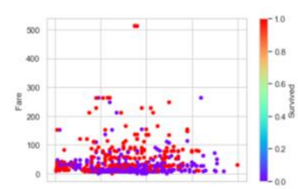
Basic Stats for Each Data Set 1-5

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.

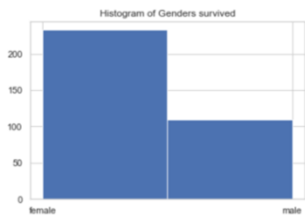
I then put together some initial EDAs of the training set to see if there was anything useful. I did a barplot to see if there was big difference between survived class and not survived class. What I found is more people died than survived in my results as shown in 1-6. Next I wanted to make a scatter 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.



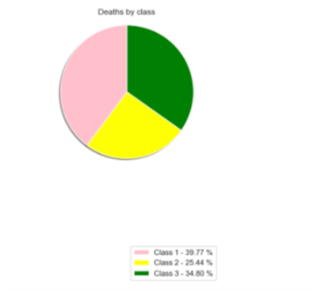
Barplot 1-6



Scatterplot 1-7

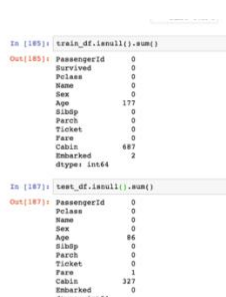


Histogram 1-8



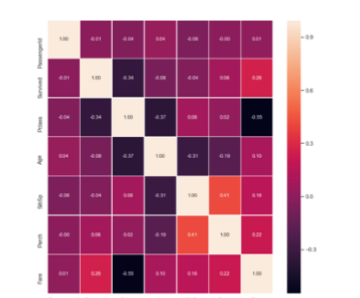
Piechart of Classes and Survival Rate 1-9

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.



```
In [185]: train_df.isnull().sum()
Out[185]: 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 [187]: test_df.isnull().sum()
Out[187]: PassengerId      0
          Pclass           0
          Name             0
          Sex              0
          Age             86
          SibSp            0
          Parch            0
          Ticket           0
          Fare             1
          Cabin          327
          Embarked         0
          dtype: int64
```



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 dummy columns with 1 and 0s. 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. For Forests, I did not need to make N-1 columns instead I needed to convert “Cabin” and “Embarked” to int category type. 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 were Logistic Regression, Random Forest and Extra Randomized Trees. 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]. Random Forest is another special algorithm which involves decision trees [3]. Decision trees are trees that use feature cutoffs to decide on the final value of the prediction [3]. With Random Forests it involves an ensembled method which involves different trees which all aggregate together to form a predicted result [3]. Each tree uses a method to collect training data which is known as “Bagging”. “Bagging” involves sampling which collects n data points from the training set to train on. Thirdly another Tree learning algorithm used is Extra Randomized Trees [3]. This is similar to Random Forests in a sense because they both pick a random number of features to train on, but this also involves random thresholds to use as a cutoff for each feature chosen [3]. I think these algorithms are important because Linear Regression as mentioned before is mostly used to predict numerical values. Trees are kind of different in that one can see how the algorithm works and it is not a "black-box" like Linear Regression is. Trees do not assume linearity like Regression does and does not need transformation and scaling [3]. 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. For the tree algorithms I only needed to drop "Passenger ID", "Name", "Parch", and "SibSp" as the other columns were just converted to numeric.

Review results, evaluate models

After training the models, I noticed from the results of AUC, and accuracy score, that the Random Forests model was the best model. The scores were both pretty high were Accuracy and AUC for the Test Data in 1-13 while score while the Accuracy was lower for Logistic Regression and was too high for AUC score at 0.89. I also checked the OOB score for Random Forests and what I saw was that the OOB score increased for the test set from the training set from 0.80 to 0.866.

I also wanted to how the linear regression model was doing by looking at the coefficients and which features the extra trees classifier was marking as important. The formulas for logistic regression from 1-14 and 1-15 are:

1.71 – 1.53*Pclass -2.4*Sex -0.37*Age + 0.29*Fare + 0*CabA + 0.74*CabB +0.63*CabC + 1.12*CabC – 0.0079*CabD + 0.62*CabE – 0 .394*CabF + 0*CabT – 0.049*EmbS + 0.733*EmbC – 0.205*Fammemb

Train Data Formula

3.15 – 1.67*Pclass -2.51*Sex -2.13*Age + 0.56*Fare + 0.26*CabA + 0.20*CabB - 0.26*CabC + 0.51*CabC + 1.17*CabD + 0.81*CabE + 0.017*CabF – 0.245*CabT – 0.362*EmbS + 0.312*EmbC – 1.287*Fammemb

Test Data Formula

From looking at the train and test data formula, Pclass and Sex are the most heavily weighted which suggests that they contribute the most to the survival rate. What Pclass of -1.53 says for the train data that for every increase in Pclass the odds of survival decrease by 21.7%. With Sex for every increase, the odds of survival decrease by 9%. In the test data, what Pclass of -1.67 says for the train data that for every increase in Pclass the odds of survival decrease by 18.8%. With Sex for every increase, the odds of survival decrease by 8.1%. I also notice the weight of Age has gone up which says for every increase in age the odds of survival decrease by 11.89%. I also went ahead to look at what the Extra Trees said about importance features for both train data and test data in 1-16 and 1-17. What it said is that the importance of “Sex” was at 28%, the importance of “age” was at 24% and the importance of “Fare” was at 22.4% for the train data. I was a little surprised at how it marked these important features. For the test data, it said that 10% feature importance to “Pclass”, 27% importance was to “Sex”, 15.3% importance to “age”, 19.34% to “Fare” and 12% feature importance was to “Cabin”. I thought this was surprising as well considering the EDA findings.

	Classifier	Accuracy	AUC
0	Logistic Regression	0.793598	0.838932
1	Random Forrest Classifier	0.790860	0.790830
2	Extra Trees Classifier	0.779573	0.775367

Training Set Results 1-12

Out[814]:

	Classifier	Accuracy	AUC
0	Logistic Regression	0.787460	0.895652
1	Random Forrest Classifier	0.865397	0.804905
2	Extra Trees Classifier	0.843492	0.774008

Test Set Results 1-13

```
[[-1.53119647 -2.40533073 -0.3662788   0.2950648   0.          0.74364169
   0.63108398  1.12790299 -0.00792574  0.62149717 -0.39438881  0.
  -0.04948702  0.73264367 -0.20547853]]
[1.71264417]
```

Test Data Coefficients 1-14

```
[794]: #print formula coefficients as well as intercept
       print(log_reg.coef_)

       [[-1.6696145  -2.51302151 -2.1370189   0.56786685  0.25773991  0.20372611
         -0.26103434  0.50392596  1.17361218  0.80805185  0.01662876 -0.24454108
         -0.36161555  0.03115022 -1.28660895]]

[795]: print(log_reg.intercept_)

       [3.15140194]
```

Train Data Coefficients 1-15

```
array([0.0854828 , 0.28002679, 0.24329833, 0.22477831, 0.06475646,
       0.03132265, 0.07033466])
```

Train Data Feature Importance 1-16

```
Out[83]: array([0.10913612, 0.27002161, 0.1530056 , 0.19342465, 0.1208983 ,
                0.04691237, 0.10660134])
```

Test Data Feature Importance 1-17

Implementation and programming

For implementation I first imported the packages as seen in 1-18 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 Imports 1-18

```
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, RandomForests and Extra Trees. I had to first call the instance of each model and use the **.fit()** method on each. The way to call the instance is writing

LogisticRegression(), RandomForestClassifier() and **ExtraTreesClassifier().** 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. I also used **ExtraTreesClassifier().feature_importances_** to find which features Extra Trees put more weight in percentages on. I also did something similar to get coefficeints an y intercept for logistic regression. See Sklearn package for more details on the attributes to apply.

Exposition, problem description, and management recommendations

After reviewing the above formulas, results from each algorithm, **I recommend** the Random Forests classifier for this data set as it seems to have performed the best by looking at the table for the test data 1-13. I chose this algorithm as Logistic Regression had too low of an accuracy score for the test-data at 0.787 while Extra Trees had too low of a AUC score for the test data at 0.774. I think by looking at the formulas and feature importance in 1-14-1-17 it made sense to me why those were not performing well as those algorithms though features such as "Cabin", "Age" were important or heavily weighted. Another reason why Random Forests was a great algorithm, is because the OOB score was around 0.86 for the test data. This was also a big factor in my interpretation. Therefore, I recommend the Random Forests algorithm to management. In the future I also want to explore how I can make Extra Trees classifier, and Logistic Regression perform better. I think it will take some hyperparameter tuning which may help them both perform better for this type of problem.

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.

Appendix

Import packages

In [1]:

```
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.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

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


from collections import OrderedDict


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 [2]:

```
%matplotlib inline
```

In [3]:

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

In [4]:

```
#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 [5]:

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

Out[5]:

	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 [6]:

```
train_df.head()
```

Out[6]:

	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 [7]:

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

Out[7]:

```
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
```

In [8]:

```
test_df.dtypes
```

Out[8]:

```
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 [9]:

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

Out[9]:

```
(418, 11)
```

In [10]:

```
train_df.shape
```

Out[10]:

```
(891, 12)
```

In [11]:

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

Out[11]:

```
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 [12]:

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

Out[12]:

```
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 [13]:

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

Out[13]:

	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 [14]:

```
test_df.describe()
```

Out[14]:

	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 [15]:

```
#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[15]:

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

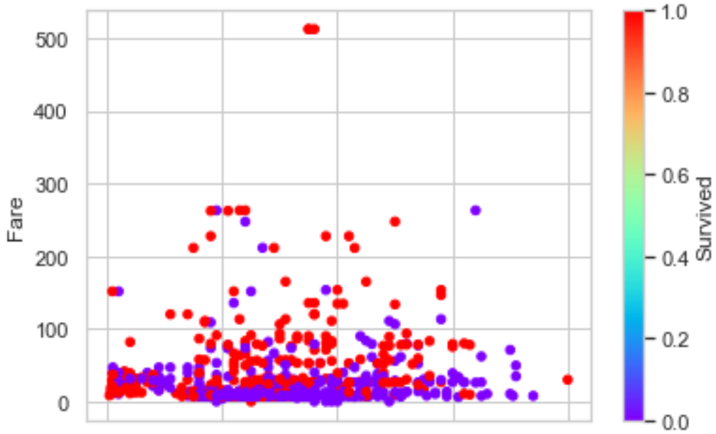
In [16]:

```
#make scatterplot

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

Out[16]:

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

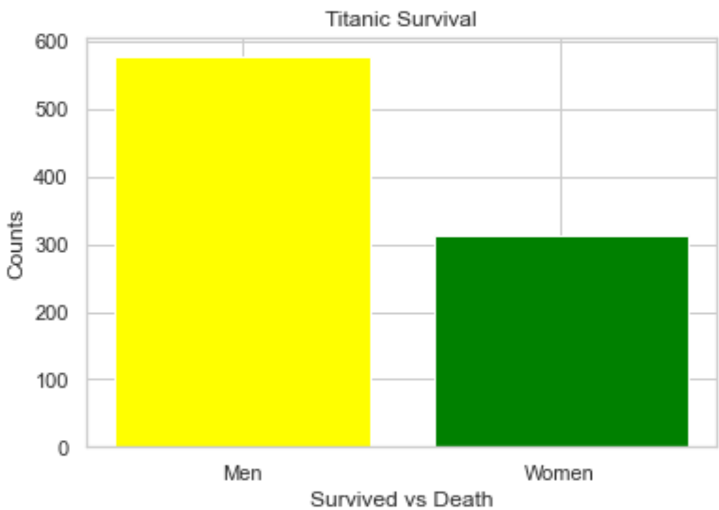


In [17]:

```
#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[17]:

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

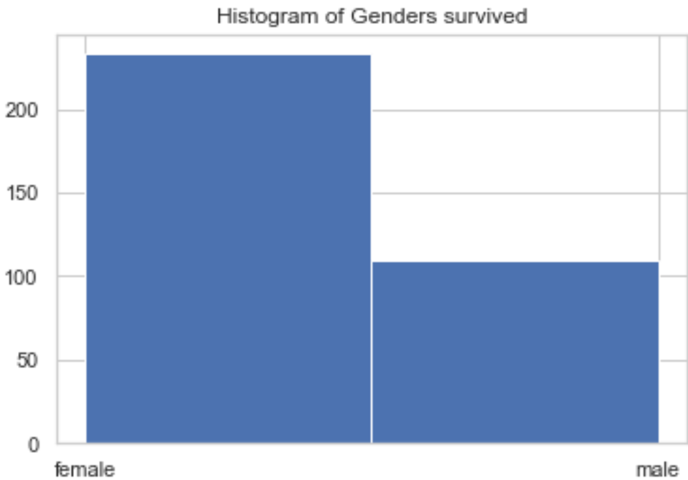


In [18]:

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

Out[18]:

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

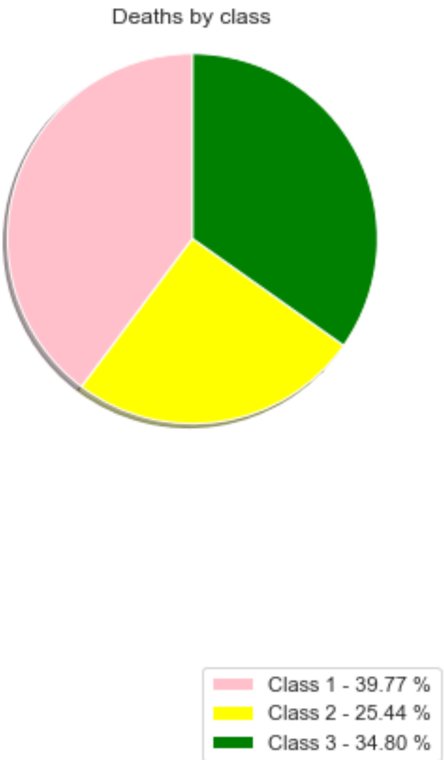


In [19]:

```
#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 [20]:

```
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 [21]:

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

Out[21]:

```
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 [22]:

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

Out[22]:

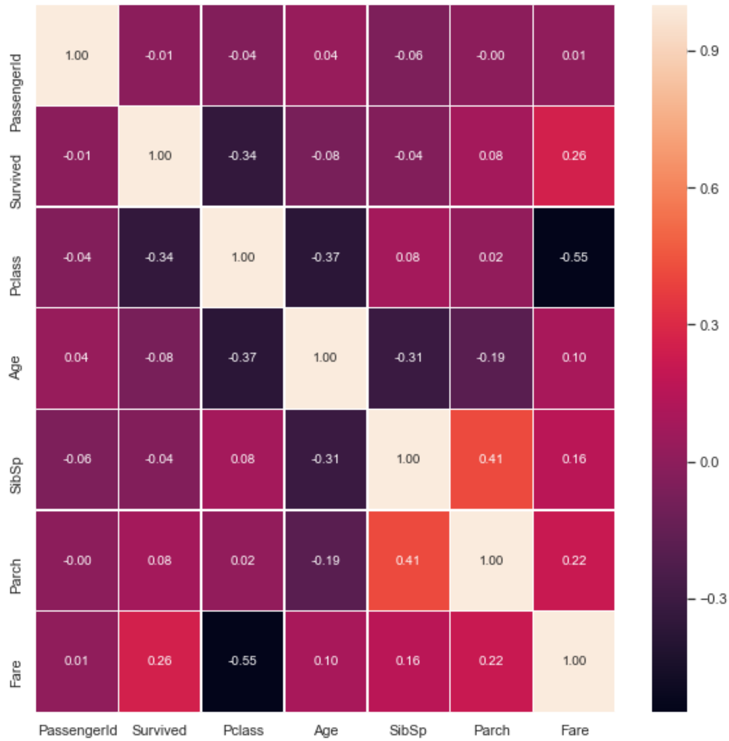
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [23]:

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

Out[23]:

```
<AxesSubplot:>
```



In [24]:

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

Out[24]:

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 [25]:

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

Out[25]:

		PassengerId	Age	Parch	Fare
Pclass	**SibSp**				
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 [26]:

```
#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 [27]:

```
#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, 
```

In [28]:

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

Out[28]:

```
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 [29]:

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

Out[29]:

```
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 [30]:

```
#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 [31]:

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

Out[31]:

```
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 [32]:

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

Out[32]:

```
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 [33]:

```
#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 [34]:

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

Out[34]:

```
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 [35]:

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

In [36]:

```
#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 [37]:

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

Out[37]:

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

In [38]:

```
training_df = train_df.copy()
```

In [39]:

```
training_df
```

Out[39]:

	PassengerId	Survived	Pclass	Name	Sex	Age	Si
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	
...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	25.0	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	

	PassengerId	Survived	Pclass	Name	Sex	Age	Si
890	891	0	3	Dooley, Mr. Patrick	male	32.0	

891 rows × 12 columns

In [40]:

```
#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)
training_df["Sex"]=training_df["Sex"].map(sex_map)
```

In [41]:

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

In [42]:

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

In [43]:

```
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 [44]:

```
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 [45]:

```
training_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 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
dtypes: float64(2), int64(6), object(4)
memory usage: 83.7+ KB
```

In [46]:

```
#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'] +
training_df['Fammemb'] = training_df['SibSp'] + training_df
```

In [47]:

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

In [48]:

```
#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
training_df.drop(["Name","Ticket","PassengerId","SibSp","Pa
training_df['Cabin'] = training_df['Cabin'].astype('categor
training_df['Embarked'] = training_df['Embarked'].astype('c
training_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
Survived    891 non-null int64
Pclass      891 non-null int64
Sex         891 non-null int64
Age         891 non-null float64
Fare        891 non-null float64
Cabin       891 non-null int8
Embarked    891 non-null int8
Fammemb     891 non-null int64
dtypes: float64(2), int64(4), int8(2)
memory usage: 43.6 KB
```

In [49]:

```
training_df.head(1000)
```

Out[49]:

	Survived	Pclass	Sex	Age	Fare	Cabin	Embarked	F
0	0	3	1	22.0	7.2500	7	2	
1	1	1	0	38.0	71.2833	2	0	
2	1	3	0	26.0	7.9250	7	2	
3	1	1	0	35.0	53.1000	2	2	
4	0	3	1	35.0	8.0500	7	2	
...	...	...	...	...	...	...	...	
886	0	2	1	27.0	13.0000	7	2	
887	1	1	0	19.0	30.0000	1	2	
888	0	3	0	25.0	23.4500	7	2	
889	1	1	1	26.0	30.0000	2	0	
890	0	3	1	32.0	7.7500	7	1	

891 rows × 8 columns

In [50]:

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

Out[50]:

	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 [51]:

```
#scale data by min max scaler
train_df=train_df.transform(lambda x: (x - x.min()) / (x.ma
```

In [52]:

```
#Next save response variable and drop from training set
x = train_df.drop(['Survived'], 1)
y = training_df1["Survived"]
x1 = training_df.drop(['Survived'], 1)
y1 = training_df1["Survived"]
```

In [53]:

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

Out[53]:

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

In [54]:

```
x1.shape, y.shape
```

Out[54]:

```
((891, 7), (891,))
```

In [55]:

```
#we want to split training data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_siz
x_train1,x_test1,y_train1,y_test1=train_test_split(x1,y1,te
x_train1.shape, x_test1.shape
```

Out[55]:

```
((712, 7), (179, 7))
```

In [56]:

```
x_train.shape, x_test.shape
```

Out[56]:

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

In [57]:

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

Out[57]:

	Pclass	Sex	Age	Fare	Cabin A	Cabin B	Cabin C	Ca
140	1.0	0.0	0.308872	0.029758	0.0	0.0	0.0	
439	0.5	1.0	0.384267	0.020495	0.0	0.0	0.0	
817	0.5	1.0	0.384267	0.072227	0.0	0.0	0.0	
378	1.0	1.0	0.246042	0.007832	0.0	0.0	0.0	
491	1.0	1.0	0.258608	0.014151	0.0	0.0	0.0	

In [58]:

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

In [59]:

```
#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 [60]:

```
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 [61]:

```
#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: 0.7935979513444302
```

Out[61]:

	Predicted Dead	Predicted Survived
Actual Dead	0.85	0.15
Actual Survived	0.29	0.71

In [62]:

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

```
[[-1.6696145  -2.51302151 -2.1370189   0.56786
685  0.25773991  0.20372611
  -0.26103434  0.50392596  1.17361218  0.80805
185  0.01662876 -0.24454108
  -0.36161555  0.03115022 -1.28660895]]
```

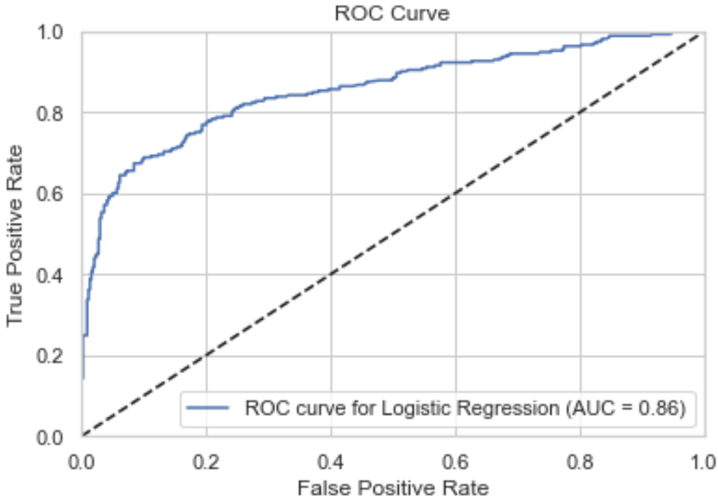
In [63]:

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

```
[3.15140194]
```

In [64]:

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

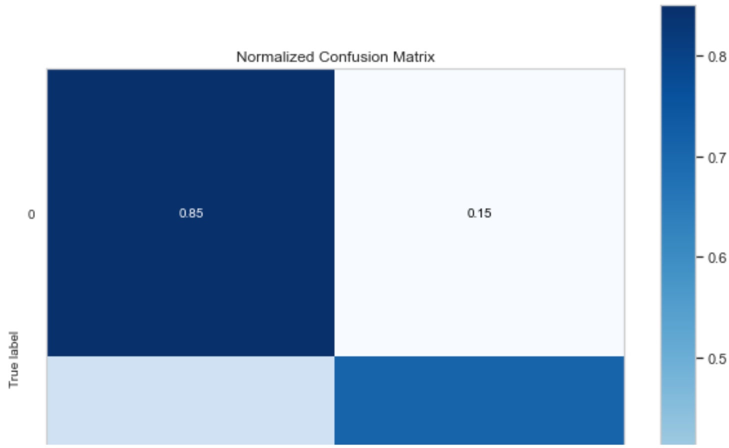


In [65]:

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

Out[65]:

```
&lt;AxesSubplot:title={'center':'Normalized Confu
sion Matrix'}, xlabel='Predicted label', ylabe
l='True label'&gt;
```



In [66]:

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

Accuracy: 0.7874603174603174

Out[66]:

	Predicted Dead	Predicted Survived
Actual Dead	0.87	0.13
Actual Survived	0.25	0.75

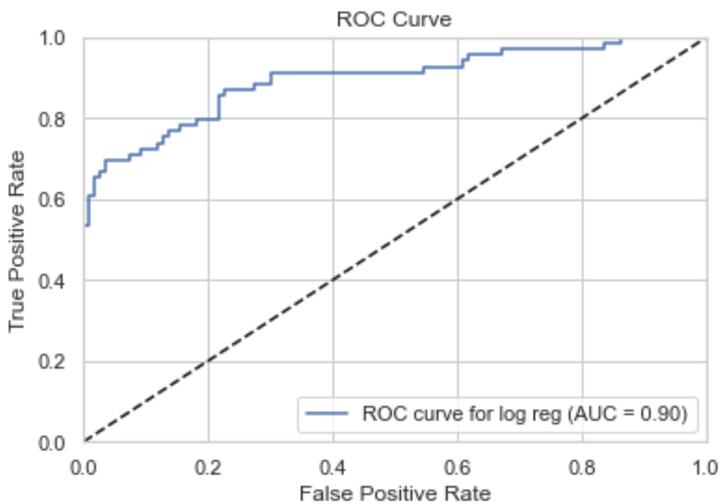
In [67]:

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

```
[[-1.53119647 -2.40533073 -0.3662788   0.29506
48   0.          0.74364169
   0.63108398  1.12790299 -0.00792574  0.62149
717 -0.39438881  0.
  -0.04948702  0.73264367 -0.20547853]]
[1.71264417]
```

In [68]:

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



In [69]:

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

Out[69]:

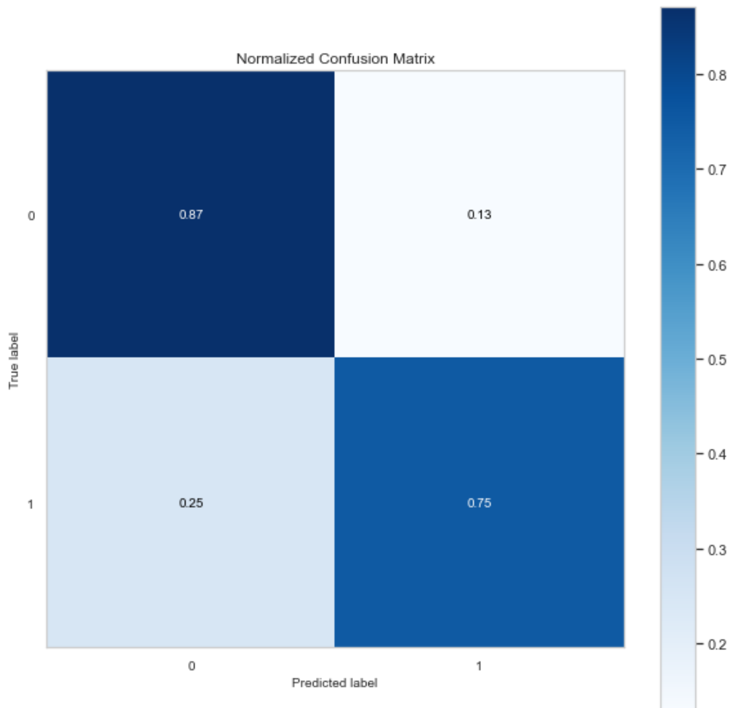
	Predicted Dead	Predicted Survived
Actual Dead	0.87	0.13
Actual Survived	0.25	0.75

In [70]:

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

Out[70]:

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



In [71]:

```
from sklearn.ensemble import RandomForestClassifier # Rando
from sklearn.ensemble import ExtraTreesClassifier # Extra T
Randreg = RandomForestClassifier(oob_score = True)

# Fit data on to the model
Randreg.fit(x_train1, y_train1)
print(Randreg.oob_score_)
# Predict
y_predicted_Randreg = Randreg.predict(x_test1)
```

```
0.800561797752809
```

In [72]:

```
print("Accuracy: " + str(acc_score(Randreg, x_train1, y_tra
print(aucscore(Randreg, x_train1, y_train1, has_proba=True)
confusion_matrix_model(Randreg, x_train1, y_train1)
```

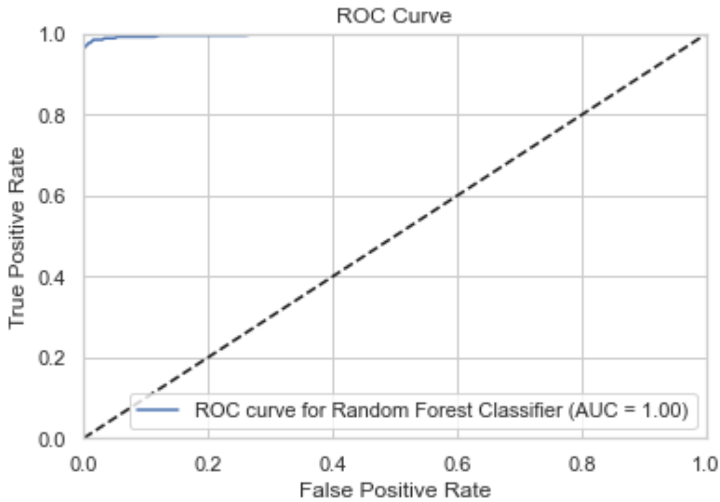
```
Accuracy: 0.7993203979119471
0.9981100903652157
```

Out[72]:

	Predicted Dead	Predicted Survived
Actual Dead	0.99	0.01
Actual Survived	0.02	0.98

In [73]:

```
plt_roc_curve("Random Forest Classifier",Randreg,x_train1, 
```

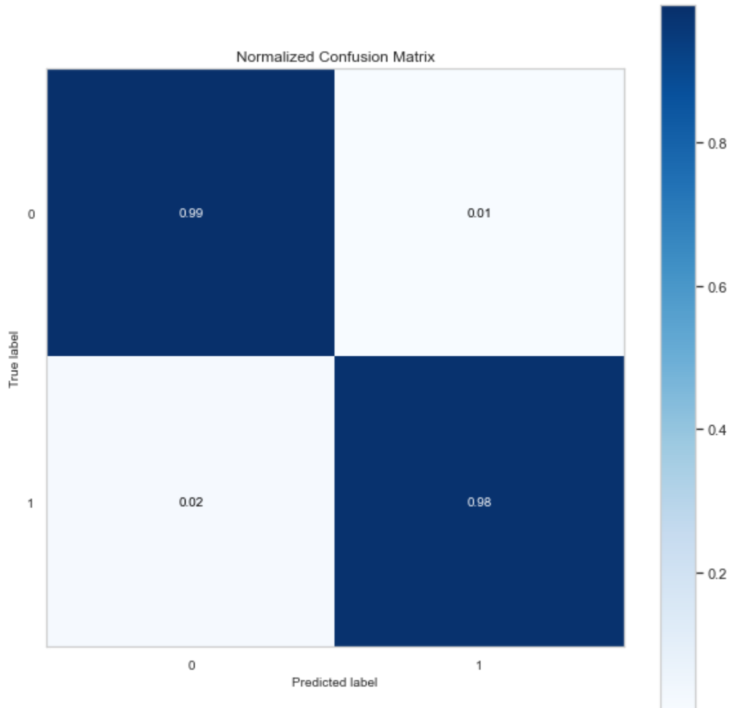


In [74]:

```
skplt.metrics.plot_confusion_matrix(y_train1, Randreg.predi
```

Out[74]:

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



In [75]:

```
ETreg = ExtraTreesClassifier()

# Fit data on to the model
ETreg.fit(x_train1, y_train1)

# Predict
y_predicted_ETreg = ETreg.predict(x_test1)
```

In [76]:

```
print("Accuracy: " + str(acc_score(ETreg, x_train1, y_train
ETacc=acc_score(ETreg, x_train1, y_train1)
confusion_matrix_model(ETreg, x_train1, y_train1)
```

```
Accuracy: 0.7810006894513937
```

Out[76]:

	Predicted Dead	Predicted Survived
Actual Dead	1.00	0.00
Actual Survived	0.04	0.96

In [77]:

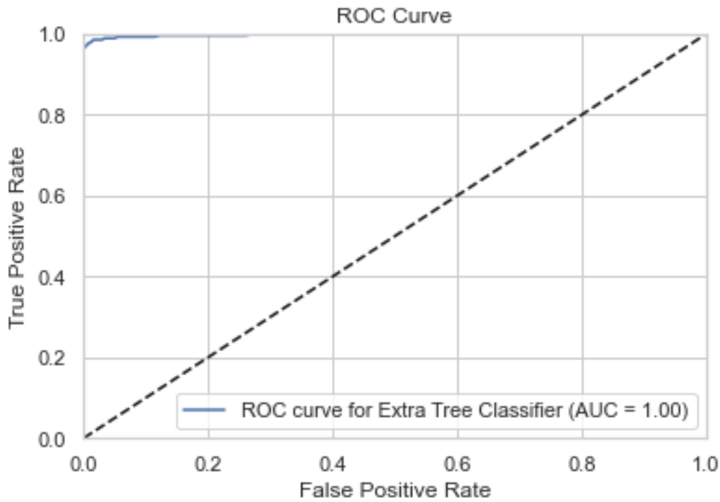
```
ETreg.feature_importances_
```

Out[77]:

```
array([0.0854828 , 0.28002679, 0.24329833, 0.2
2477831, 0.06475646,
       0.03132265, 0.07033466])
```

In [78]:

```
plt_roc_curve("Extra Tree Classifier",Randreg, x_train1, y_
```

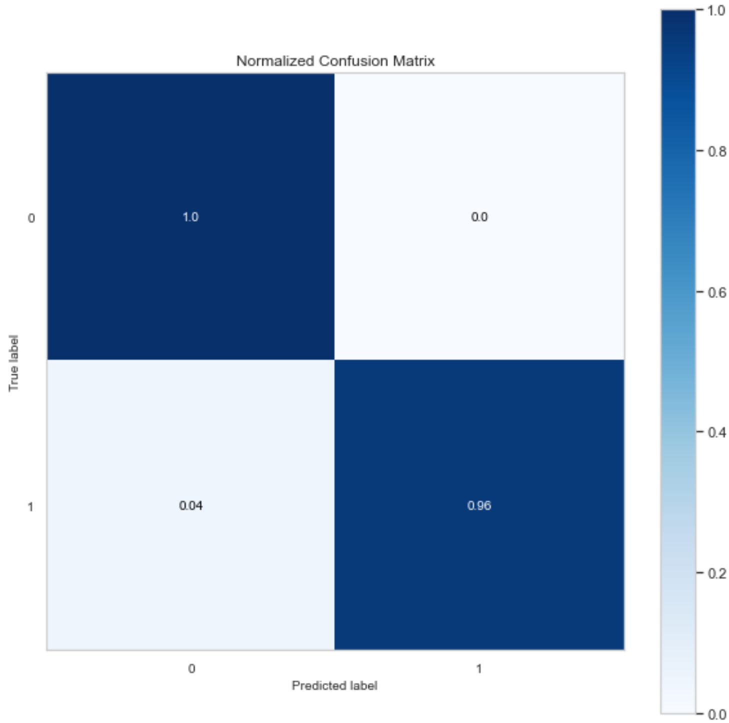


In [79]:

```
skplt.metrics.plot_confusion_matrix(y_train1, ETreg.predict
```

Out[79]:

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



In [80]:

```
Classifiers=["Logistic Regression","Random Forrest Classifi
Acc=[acc_score(x, x_train, y_train) for x in [log_reg]]
Acc.append(acc_score(Randreg, x_train1, y_train1))
Acc.append(acc_score(ETreg, x_train1, y_train1))
auc_scores_prob=[aucscore(x, x_train, y_train,has_proba=Tru
auc_scores_prob.append(acc_score(Randreg, x_train1, y_train
auc_scores_prob.append(acc_score(ETreg, x_train1, y_train1)
auc_scores=auc_scores_prob[:3] + auc_scores_prob[3:]
cols=["Classifier","Accuracy","AUC"]
results = pd.DataFrame(columns=cols)
results["Classifier"]=Classifiers
results["Accuracy"]=Acc
results["AUC"]=auc_scores
results
```

Out[80]:

	Classifier	Accuracy	AUC
0	Logistic Regression	0.793598	0.838932
1	Random Forrest Classifier	0.803546	0.795075
2	Extra Trees Classifier	0.783788	0.788004

In [81]:

```
Randreg = RandomForestClassifier(oob_score = True)

# Fit data on to the model
Randreg.fit(x_test1, y_test1)
print(Randreg.oob_score_)
print("Accuracy: " + str(acc_score(Randreg, x_test1, y_test
print(aucscore(Randreg, x_test1, y_test1, has_proba=True))
confusion_matrix_model(Randreg, x_test1, y_test1)
```

```
0.8603351955307262
Accuracy: 0.8820634920634921
1.0
```

Out[81]:

	Predicted Dead	Predicted Survived
Actual Dead	1.0	0.0
Actual Survived	0.0	1.0

In [82]:

```
ETreg = ExtraTreesClassifier()

# Fit data on to the model
ETreg.fit(x_test1, y_test1)

print("Accuracy: " + str(acc_score(ETreg, x_test, y_test)))
ETacc=acc_score(ETreg, x_test1, y_test1)
confusion_matrix_model(ETreg, x_test1, y_test1)
```

Accuracy: 0.8433333333333334

Out[82]:

	Predicted Dead	Predicted Survived
Actual Dead	1.0	0.0
Actual Survived	0.0	1.0

In [83]:

```
ETreg.feature_importances_
```

Out[83]:

```
array([0.10913612, 0.27002161, 0.1530056 , 0.1
9342465, 0.1208983 ,
       0.04691237, 0.10660134])
```

In [84]:

```
Classifiers=["Logistic Regression","Random Forrest Classifi
Acc=[acc_score(x, x_test, y_test) for x in [log_reg]]
Acc.append(acc_score(Randreg, x_test1, y_test1))
Acc.append(acc_score(ETreg, x_test1, y_test1))
auc_scores_prob=[aucscore(x, x_test, y_test,has_proba=True)
auc_scores_prob.append(acc_score(Randreg, x_train1, y_train
auc_scores_prob.append(acc_score(ETreg, x_train1, y_train1)
auc_scores=auc_scores_prob[:3] + auc_scores_prob[3:]
cols=["Classifier","Accuracy","AUC"]
results = pd.DataFrame(columns=cols)
results["Classifier"]=Classifiers
results["Accuracy"]=Acc
results["AUC"]=auc_scores
results
```

Out[84]:

	Classifier	Accuracy	AUC
0	Logistic Regression	0.787460	0.895652
1	Random Forrest Classifier	0.870794	0.806323
2	Extra Trees Classifier	0.849048	0.778203