Titanic: Chance of Survival

Vivian Do, Gabriella Rivera, Joel Day

Shiley-Marcos School of Engineering, University of San Diego

Masters of Applied Data Science

December 12, 2022

Table of Contents

Titanic: Chance of Survival 4

Overview and Description 4

Data Preparation..... 5

Attribute Deletion 5

Partitioning the Dataset..... 6

Title Extracted from Name 6

Age Predictor 7

Family Count: Combined SibSP and Pach 7

Exploratory Data Analysis..... 8

Title..... 8

Age and Family..... 9

Sex..... 10

Sex and Family 11

Embarked..... 12

Pclass..... 13

Ticket 13

Fare 13

Modeling..... 14

Logistic Regression..... 14

Remaining Three Models..... 16

Model Evaluation..... 17

Confusion Matrix..... 17

Establishing Baseline Model Performance 18

Confusion Matrix-based Performance Measures 18

ROC Curves 19

Conclusions 20

References 21

Supplemental 22

Titanic: Chance of Survival

Titanic is one of the most infamous commercial shipwrecks that has captured popular culture's sentiments especially when the movie starring Kate Winslet and Leonardo DiCaprio was released in 1977 (Ridding, 1998). Even about a century after its tragic fate on the 15th of April 1912, the Titanic has continued to be one of the most well-recognized and highly documented sunken ships in history. Although there are other deadlier maritime disasters such as SS Cap Arcona with 5,000 deaths and MV Goya with 7,000 deaths which both sank in 1945 wartime, the Titanic remains the most beloved shipwreck story of all time in the mainstream media (The Shipyard, 2022). People gravitate to the many retelling and new discoveries surrounding the Titanic time and again. The consumption of Titanic storytelling has proven to be appealing in not only drawing people's interest but as well as an enriching historical experience.

In this article, the Titanic dataset is subjected to exploratory data analysis in hopes to discover new information that may contribute to the survival chance of each passenger with the given attributes recorded at the time. To start with, it is reported that the Titanic had a total of 1,500 deaths and only 706 survivor counts (History.com Editors, 2009). With this, attributes such as age, sex, economic status, and the number of family members present are used to investigate key relationships as contributing factors to the passenger's survival chance. In addition, today's culture is progressively attentive to the presence of any disproportion between the attributes mentioned so predictive algorithms were created. This is to address future comparative analysis with similar shipwrecks throughout history to determine the progress of disparities to present-day application.

Overview and Description

The data was retrieved from Kaggle.com, as part of the competition 'Titanic - Machine Learning from Disaster'. Only the train.csv file from Kaggle was used; this file was later partitioned into the training and test set. The Titanic data frame, derived from the train.csv and used to train the model, contains 891 entries and 12 features. Figure 1 lists the names of all features as well as their data type. Survived is the target variable we are attempting to predict. In the training set there are 342 survivors and 549 deaths. Of the 11 remaining features, 6 are numeric and 5 categorical. Three features had null values. Cabin=687, Age=177, and Embarked=2. The eleven features, listed in the same order as Figure 1, represent PassengerID, Survival, Ticket Class, Sex, Age in years, # of siblings / spouses aboard the Titanic, # of parents / children aboard the Titanic, Ticket Number, Passenger Fare, Cabin Number, Port of Embarkment (Kaggle, 2022).

Figure 1

Features with their Corresponding Count and Data Type

0	PassengerId	891 non-null	int64
1	Survived	891 non-null	int64
2	Pclass	891 non-null	int64
3	Name	891 non-null	object
4	Sex	891 non-null	object
5	Age	714 non-null	float64
6	SibSp	891 non-null	int64
7	Parch	891 non-null	int64
8	Ticket	891 non-null	object
9	Fare	891 non-null	float64
10	Cabin	204 non-null	object
11	Embarked	889 non-null	object

Data Preparation

Attribute Deletion

The attribute Cabin will be deleted from the data set due to the high number of observations with null values (n=687, 77%). Additionally, passenger ID, an index starting from 1 randomly assigned to each passenger, will also be disregarded as it is not useful for analysis.

Partitioning the Dataset

The data set was partitioned using two-fold cross validation so that observations are randomly assigned to a training set and a test set. 67% (n=596) of the data is partitioned into the training set to learn the model and 33% (n=295) are assigned to the test set used for validation. After partitioning, the training set had 373 (63%) observations with Survived=0 and 223 (37%) observations with Survived=1. The training set was then balanced through resampling, the process of sampling at random with replacement (Larose & Larose, 2019). Equation 1 was used to calculate the number of observations required to be resampled in order to achieve a 1:1 ratio of passengers that survived (Survive=1) to passengers that did not survive (Survived=0). 150 records were resampled to obtain a balanced training set with 373 observations for each target class.

$$x = \frac{p(records) - rare}{1-p} \tag{1}$$

x = required number of resampled records

p = desired proportion of rare values in the balanced training set

records = the number of records in the unbalanced set

rare = current number of rare target value (i.e. Survived=1)

The test set represents holdout data that the models have not seen and should represent real-world data conditions as closely as possible. For this reason, only the training set was rebalanced.

Title Extracted from Name

The name attribute contains each passenger's full name and title in the following format: Last name, Title. First Name. Title was created as a new attribute to the data set. While the name of each passenger is unique to the individual and too varied for analysis, titles could give us an indication of marital status and/or occupation. For example, ‘Miss’ refers to a young, unmarried woman while ‘Mrs’ indicates a married woman.

Age Predictor

Before data exploration and model creation, the age attribute was cleaned with its missing values imputed and transformed into a categorical variable. There were 177 missing age values in the original Titanic training dataset and about 146 missing age values after the Titanic training dataset was partitioned as well as rebalanced into Titanic train_rebal and test datasets. To address missing data imputation, the overall age predictor distribution was assessed and is determined to be positively skewed with its mean at 30 which is greater than the value of its median at 28. With this, the age predictors in the train_rebal and test Titanic datasets were imputed with the median value to preserve the overall shape and behavior of the data distribution. For variable transformation, the age continuous values were categorized into 4 bins to represent Baby/Toddler for ages 0-3 years old, Child for ages 4 to 17 years old, Adult for 18 to 63 years old, and Elderly for ages 64-99 years old. Note that age transformation was performed to accommodate predictive models that function best with categorical variables as well as to visualize age distribution during data exploration.

Family Count: Combined SibSP and Pach

The Fam attribute was created by combining the total count of SibSP and Pach attributes in both the Titanic training and test datasets. This was decided to avoid multicollinearity as the number of siblings and spouses together with the number of parent and child onboard the ship with the passenger fall into the same umbrella of family members. To condense the number of predictors and to make the models simpler, the Fam attribute was created to represent the total number of family members present together with the passenger on the Titanic ship at the time.

Exploratory Data Analysis

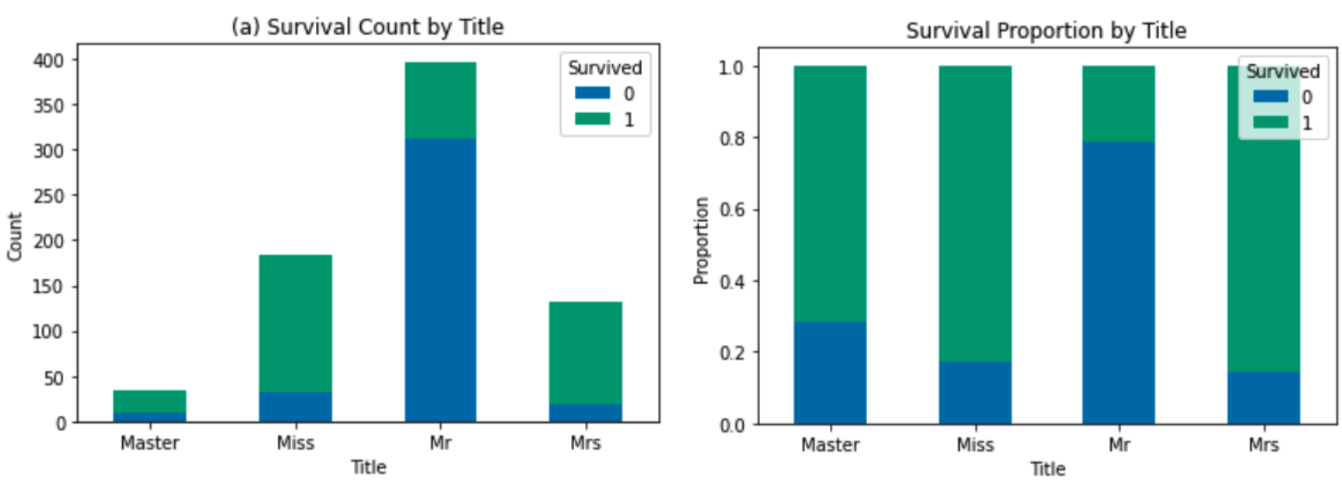
Title

The following titles were extracted from passenger names: Mr (379), Miss (180), Mrs (127), Master (35), Rev (6), Dr (4), Mme (2), Col (2), Sir (2), Ms (2), the Countess (2), with one count of Don, Major, Mlle, Jonkheer, and Capt each. Titles such as 'Mme' (Madame) and 'the Countess' (wife of an Earl) were combined with 'Mrs' to refer to married women; similarly, 'Mlle' (Madamoiselle) and 'Ms' (short for Miss) were combined with 'Miss' to specify unmarried women. 'Mr' refers to adult males while 'Master' connotes underaged boys (roughly under 18 years old). Analysis between title and sex showed that all other occupational titles were held by men and were combined with 'Mr'.

Recombination resulted in four title categories: Master (n=35, 4.69%), Mr (n=397, 53.22%), Miss (n=183, 24.53%), and Mrs (n=131, 17.56%). Young boys (71.43%, n=25), unmarried women (82.51%, n=151), and married women (85.50%, n=112) were proportionately more likely to survive; on the other hand, adult males (21.41%, n=85) were less likely to survive (Figure 2).

Figure 2

Survival by Title

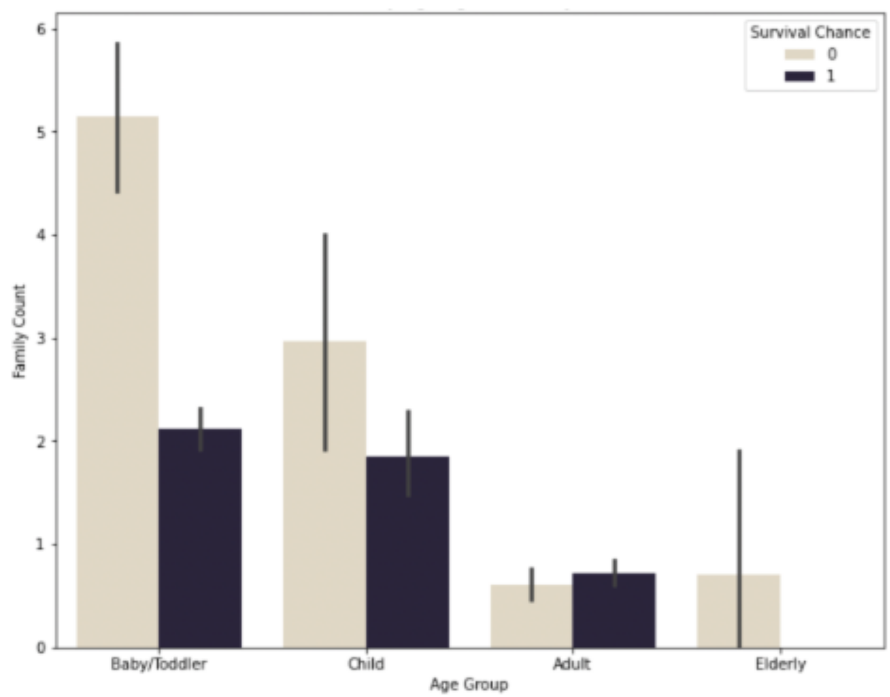


Age and Family

For age and family attributes, the patterns in distribution were first determined as shown in Supplemental Figures 1 and 2. In Supplemental Figure 1, adults have the highest body count at 632 for the age groups followed by children (n=71), babies (n=33), and then elderly people (n=10). In the contingency Supplemental Table 1, the counts of adults that survived show a similar pattern with adults having the highest body count followed by children, babies, and the elderly. In Supplemental Figure 2, people with no family members have the highest body count at 427 and the pattern decreases with increasing family member count. Similarly, contingency Supplemental Table 2 shows that the count of survival increases as the family member count decreases. However, looking at Figure 3 for the overall proportionality of survival based on each class within the attribute, babies have the highest survival chance with having 2 family members as well as the highest death chance with having 5 family members. This pattern is followed by children having 2 family members as well for survival chances and having 3 family members for those who did not survive the shipwreck. Adults have noticeably more survival proportions compared to non-survivors while having less than 1 family member. The elderly had the most tragic survival proportion with no survivors and less than 1 family member.

Figure 3

Survival by Age Group and Family Count

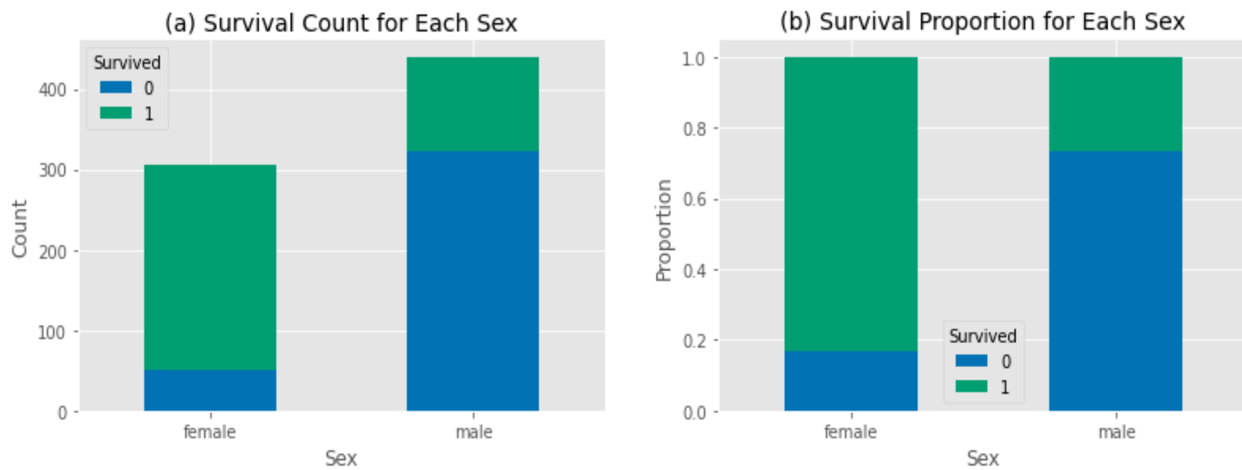


Sex

Sex is a binary, nominal variable with two values: Male or Female. In total, there were 314 (42.10%) females and 432 (57.91%) males. Proportionately more females (83.76%, n=263) survived than males (25.46%, n=110). Figure 4 shows the bar charts for the count and proportion of survival statistics for each sex. Sex, along with title, confirm what is commonly known about the escape effort made by passengers of the Titanic, as women and children were given priority for boarding onto lifeboats.

Figure 4

Bar charts of Survival by Sex

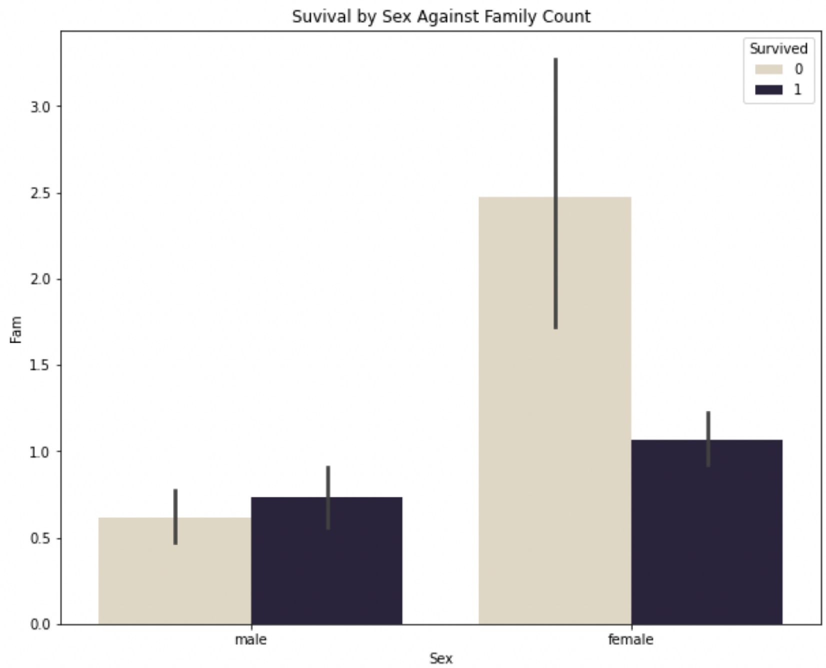


Sex and Family

Looking at Figure 5, the survival chance by sex and family member count was summarized. For males, there are proportionately more survivors than non-survivors with less than 1 family member for both instances. For females, there are proportionately more non-survivors compared to survivors. Female non-survivors have about 2 family members while female survivors have at least 1 family member. Overall, females have a slightly greater chance of survival compared to males proportionate to their class total.

Figure 5

Survival by Sex and Family Count

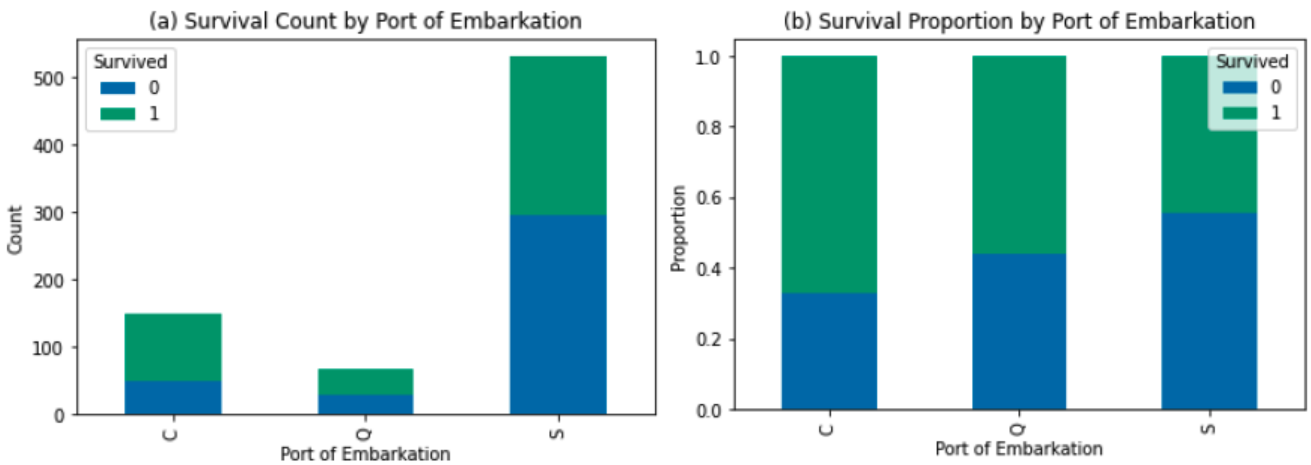


Embarked

Embarked is a nominal categorical variable that defines the port of embarkation for each passenger with three possible values: C (Port of Cherbourg), Q (Port of Queenstown), and S (Port of Southampton). Out of all passengers in the training set, 71.98% (n=537) boarded at Port S, 8.31% boarded at Port Q (n=62), and 19.71% (n=147) boarded at Port C (Figure 6a). The mode (Port S) was used to impute null values for two observations. Proportionately more passengers survived who boarded at port C (66.67%, n=98) and port Q (53.23%, n=33) while passengers who boarded at Port S were proportionately less likely to survive (45.07%, n=242) (Figure 6b).

Figure 6

Bar Charts of Survival by Ports of Embarkation



Pclass

Pclass is ordinal and split into three categories: 1, 2, & 3. There are 202 1st class, 158 2nd class, and 386 3rd class. Passengers had the highest chance of surviving in first class, and the worst survival odds in third class. This relationship suggests Pclass may be a good predictor.

Ticket

Ticket was removed due to being highly correlated with Fare. First, tickets were grouped together by the first value of the ticket number, resulting in 15 distinct ticket groups. Most tickets began with either 1, 2, or 3. A chi squared test was performed with the null hypothesis that the two variables Fare and Ticket are independent. The P-value was 6.05e-73, so the null hypothesis was rejected. This suggests that the two variables are highly correlated.

Fare

The last feature to discuss is Fare. It has a continuous distribution with a mean of 35 and a large rightward skew. The data was divided into 4 bins based on fare amount: Low, Mid, High, and Max. A crosstab table was formed with both Fare and Survived which suggests that as the

Fare amount increased so did the ticket holder's chance at surviving. Figure 7 shows the proportional chance of surviving for each ticket fare class.

Figure 7

Bar Charts of Survival by Ports of Embarkation

Modeling

Logistic Regression

For logistic regression, Ticket2 and Embarked were removed from the equation before creating the final equation. Ticket2 had NANs showing on the evaluation of the metric while embarked had high values under standard error, z-score, and p-value. With these attributes removed, the final equation uses continuous variables of Sex, Fare, Age, Fam, and Pclass attributes. Note that to make all the predictive models comparable and consistent with each other, Ticket 2 and Embarked attributes were also omitted out of CART, Random Forest, and Naive Bayes. Equations 3 and 4 are written below to demonstrate the algorithms used to calculate the predicted Titanic train and test datasets.

$$Survival\ Chance = \frac{\exp\left(b_0 + b_1 Sex + b_2 fare + b_3 Age + b_4 fam + b_5 Pclass\right)}{1 + \exp\left(b_0 + b_1 Sex + b_2 fare + b_3 Age + b_4 fam + b_5 Pclass\right)} + \varepsilon \tag{2}$$

$$Survival\ Chance_{train} = \frac{\exp\left(3.3 + 3.2 Sex + 0.001 fare - 0.05 Age - 0.3 fam - 1.3 Pclass\right)}{1 + \exp\left(3.3 + 3.2 Sex + 0.001 fare - 0.05 Age - 0.3 fam - 1.3 Pclass\right)} + \varepsilon \tag{3}$$

$$Survival\ Chance_{test} = \frac{\exp\left(0.4 + 2.2 Sex + 0.02 fare - 0.01 Age - 0.3 fam - 0.6 Pclass\right)}{1 + \exp\left(0.4 + 2.2 Sex + 0.02 fare - 0.01 Age - 0.3 fam - 0.6 Pclass\right)} + \varepsilon \tag{4}$$

Overall, the pseudo-r-squared score of 38.2% for the training dataset and 28.1% for the test dataset suggests that both models have a good fit as shown in Tables 1 and 2. Z-scores are within reasonable ranges and only 1 predictor variable in each case is not within a 0.5 significance level. This suggests that for the Titanic training dataset, the fare predictor failed to show a statistically significant relationship with survival response. The same with the Titanic test dataset, the age attribute failed to show a statistically significant relationship with survival response. The rest of the predictors are all within a 95% confidence interval. This is the same with the likelihood test (LLR) p-value at less than 0.5 level of significance which supports that both full models are performing well over the null model. For both models, the BIC is slightly higher than AIC and their values are about the same. Overall, the metrics suggest that both models are statistically significant to the response attribute.

Table 1

Logistic Model of Titanic Training Dataset

Model:	Logit	Pseudo R-squared:	0.382
Dependent Variable:	Survived	AIC:	650.9157
Date:	2022-12-09 22:55	BIC:	678.6040
No. Observations:	746	Log-Likelihood:	-319.46
Df Model:	5	LL-Null:	-517.09
Df Residuals:	740	LLR p-value:	3.1177e-83
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	3.2571	0.5057	6.4405	0.0000	2.2659	4.2483
Pclass	-1.2666	0.1502	-8.4350	0.0000	-1.5609	-0.9723
Fam	-0.2664	0.0726	-3.6709	0.0002	-0.4087	-0.1242
Age	-0.0503	0.0084	-5.9807	0.0000	-0.0668	-0.0338
Fare	0.0009	0.0021	0.4048	0.6856	-0.0033	0.0050
Sex	3.1643	0.2331	13.5721	0.0000	2.7073	3.6212

Table 2

Logistic Model of Titanic Test Dataset

Model:	Logit	Pseudo R-squared:	0.281
Dependent Variable:	Survived	AIC:	298.1912
Date:	2022-12-09 22:55	BIC:	320.3130
No. Observations:	295	Log-Likelihood:	-143.10
Df Model:	5	LL-Null:	-198.94
Df Residuals:	289	LLR p-value:	1.8066e-22
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	0.4190	0.8941	0.4686	0.6393	-1.3334	2.1714
Pclass	-0.6449	0.2470	-2.6113	0.0090	-1.1289	-0.1609
Fam	-0.2508	0.1225	-2.0473	0.0406	-0.4909	-0.0107
Age	-0.0147	0.0129	-1.1432	0.2530	-0.0400	0.0105
Fare	0.0175	0.0081	2.1679	0.0302	0.0017	0.0333
Sex	2.2155	0.3216	6.8896	0.0000	1.5852	2.8458

Remaining Three Models

Unlike Logistic Regression, CART, Random Forest, and Naive Bayes use the Embarked feature in their predictions. Like Age, Sex, and Pclass it is converted to an n-1 dummy variable.

Of the six features used in the generation of these three models, four are categorical (n-1) dummy variables: Sex, Embarked, Age, and Pclass; while two are continuous: Fare and Family.

Model Evaluation

Confusion Matrix

Confusion matrices were used to summarize the predicted results of each model compared to the actual distribution of the target class (Figure #). Values of 0 and 1 for the predicted and actual category refer to passengers who did not survive and passengers who did not survive, respectively (Did not Survive = 0, Survived = 1). True negatives (TN) refer to observations that were correctly classified as negative (i.e. passengers who did not survive were correctly identified); similarly, true positives (TP) refer to observations that were correctly classified as positive (i.e. passengers who survived were correctly identified). False negatives (FN) and false positives (FP) are observations where the model incorrectly classifies a passenger that survived or did not survive, respectively. For example, out of 176 passengers who did not survive, the logistic regression model correctly identified 149 passengers but misclassified 27 (Figure 8a).

Figure 8

Confusion Matrix for Each Model

(a) Logistic Regression

	Predicted	0	1	Total
Actual	0	149	27	176
	1	36	83	119
	Total	185	110	295

(b) CART

	Predicted	0	1	Total
Actual	0	151	25	176
	1	42	77	119
	Total	193	102	295

(c) Random Forest

	Predicted	0	1	Total
Actual	0	147	29	176
	1	43	76	119
	Total	190	105	295

(d) Naïve Bayes

	Predicted	0	1	All
Actual	0	159	17	176
	1	43	76	119
	All	202	93	295

Establishing Baseline Model Performance

Before evaluating each model's performance, the results need to be calibrated against a baseline model. We consider two possible options: an All Negative Model and an All Positive Model. The All Positive Model assigns all predictions as positive (i.e. all passengers are predicted to have survived) and the accuracy of this model will be *p,* the proportion of positive observations in the training set. On the other hand, the All Negative Model assigns all predictions as negative (i.e all passengers are predicted to have died) and the accuracy of this model will be *1-p*, the proportion of negative observations in the training set. The test set contains 176 survivors out of 295 passengers, so the All Positive Model will have an accuracy rate of 40.34% and the All Negative Model will have an accuracy rate of 59.66%. The All Negative Model will be used as a baseline model since it has a higher accuracy rate.

Accuracy is a measure of the proportion of correct predictions made by the model. An accuracy rate of 59.66% will be the lower threshold and only models that outperform the baseline model will be considered as useful.

Confusion Matrix-based Performance Measures

Table 3 summarizes the accuracy, error rate, sensitivity, and specificity for each model. All of the models outperformed the baseline model with Naive Bayes having the highest accuracy rate (79.66%), followed by Logistic Regression (78.64%), CART (77.29%) , and Random Forest (75.59%).

Additionally, all four models were more successful in correctly identifying passengers who died (specificity) compared to passengers who survived (sensitivity). The Naive Bayes model predicted the highest proportion of passengers who did not survive (i.e. the number of true negatives out of all possible negatives) with a specificity rate of 90.34%. In terms of sensitivity (i.e. the proportion of passengers correctly identified as survivors), the Logistic Regression model outperformed the other three with a sensitivity rate of 69.75%.

Table 3

Confusion Matrix-based Performance Measures

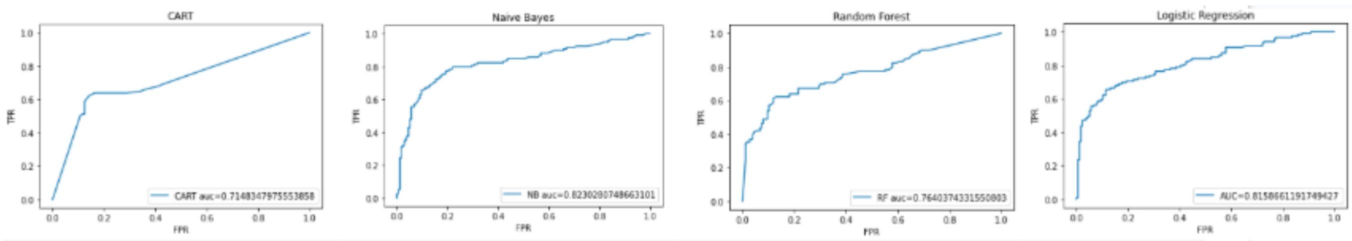
Model	Accuracy	Error Rate	Sensitivity	Specificity
Logistic Regression	78.64%	21.36%	69.75%	84.66%
CART	77.29%	22.71%	64.71%	85.80%
Random Forest	75.59%	24.41%	63.87%%	83.52%
Naïve Bayes	79.66%	20.34%	63.87%	90.34%

ROC Curves

ROC curves were created for all four models. The AUC is the area under this curve, and it was also calculated for each model. A high AUC represents a more accurate model (Brownlee, 2018). As shown in Figure 9, the AUCs listed in decreasing order are Naïve Bayes = .823, Logistic Regression = .816, Random Forest =.764, and CART = .714. Naive Bayes and Logistic regression both showed top AUC scores. This further suggests supporting Naive Bayes over Logistic Regression as the best performing model.

Figure 9

ROC Curves for the Naive Bayes, Random Forest, Logistic Regression, & CART Models



Conclusions

The Naive Bayes model outperformed the Logistic Regression, CART, and Random Forest models with an accuracy of 79.66%, error rate of 20.34%, sensitivity of 63.87%, specificity of 90.34%, and an area under the ROC curve of 0.8230. Given a passenger's age, sex, family size, boarding class, and fare price, the Naive Bayes model would correctly identify the survival status of that passenger 79.66% of the time. Additionally, the Naive Bayes model is able to identify 63.87% of all survivors and 90.34% of all deaths. The anticipated future application of data exploration is to shine awareness on how a ship's safety is engineered to account for and increase the survival of other disproportionately represented groups. It is also the intention of this article to produce a robust algorithm that can be used to predict the outcome of a similar shipwreck or gauge the survival outcome of a particular ship in question.

References

Brownlee, J. (2018, August 30). How to Use ROC Curves and Precision-Recall Curves for Classification in Python. *MachineLearningMastery.Com*. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classific tion-in-python/

History.com Editors. (2009, November 9). Titanic. History.com. Retrieved December 4, 2022, from https://www.history.com/topics/early-20th-century-us/titanic

Kaggle. *Titanic - Machine Learning from Disaster.* Retrieved November 7, 2022, from https://www.kaggle.com/competitions/titanic/overview

Larose, C. D. & Larose, D. T. (2019). *Data science Using Python and R*. John Wiley & Sons, Inc.

Riding, A. (1998, April 26). *Why 'titanic' conquered the world*. The New York Times. Retrieved December 9, 2022, from https://www.nytimes.com/1998/04/26/movies/why-titanic-conquered-the-world.html

The Shipyard. (2022, May 25). *13 maritime disasters more tragic than the Titanic*. The Shipyard. Retrieved December 4, 2022, from https://www.theshipyardblog.com/13-maritime-disasters-more-tragic-than-the-titanic/

Supplemental

Supplemental Figure 1

Age Attribute Distribution

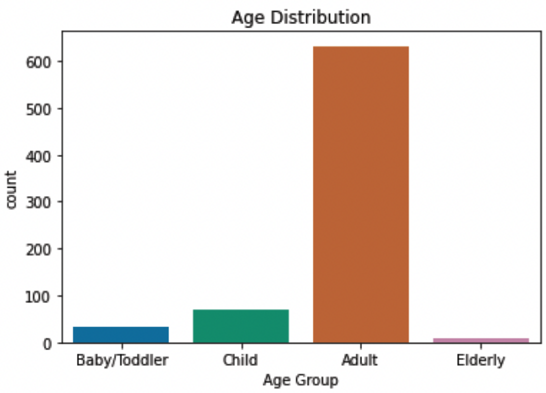
Distribution of age group:

```
train_rebal['Age_c'].value_counts()
```

```
Adult           632
Child            71
Baby/Toddler     33
Elderly          10
Name: Age_c, dtype: int64
```

```
sns.countplot(x=train_rebal["Age_c"])
plt.xlabel('Age Group')
plt.title('Age Distribution')
```

```
Text(0.5, 1.0, 'Age Distribution')
```



Supplemental Table 1

Age and Survival Contingency Table

Contingency table for age and Survived:

```
A1 = pd.crosstab(train_rebal['Survived'], train_rebal['Age_c'],
                 margins=True)
A1
```

Age_c / Survived	Baby/Toddler	Child	Adult	Elderly	All
0	7	27	329	10	373
1	26	44	303	0	373
All	33	71	632	10	746

Supplemental Figure 2

Family Attribute Distribution

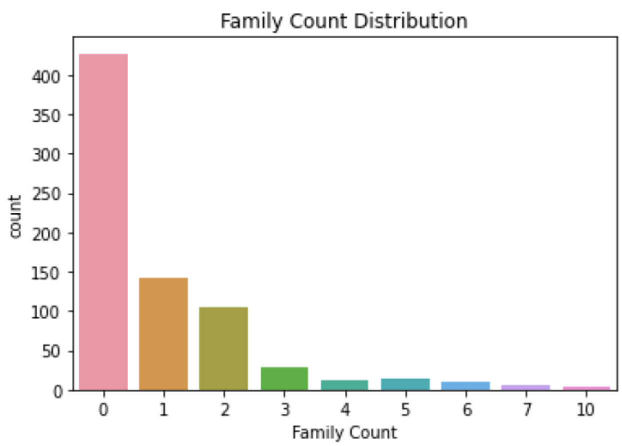
Distribution of family count:

```
train_rebal['Fam'].value_counts()
```

```
0     427
1     142
2     104
3      29
5      15
4      11
6       9
7       5
10      4
Name: Fam, dtype: int64
```

```
sns.countplot(x=train_rebal["Fam"])
plt.xlabel('Family Count')
plt.title('Family Count Distribution')
```

```
Text(0.5, 1.0, 'Family Count Distribution')
```



Supplemental Table 2

Family Count and Survival Contingency Table

Contingency table for family count and Survived:

```
F1 = pd.crosstab(train_rebal['Survived'], train_rebal['Fam'],
                 margins=True)
F1
```

Fam	0	1	2	3	4	5	6	7	10	All
Survived										
0	252	52	31	5	6	13	5	5	4	373
1	175	90	73	24	5	2	4	0	0	373
All	427	142	104	29	11	15	9	5	4	746

Data Import

In [199…]:
```
#import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
!pip install klib
import klib
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from scipy import stats
import statsmodels.api as sm
import statsmodels.tools.tools as stattools
from tabulate import tabulate
from sklearn.cluster import KMeans
from sklearn.naive_bayes import MultinomialNB
import statsmodels.tools.tools as stattools
import sklearn.metrics as metrics
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: klib in /usr/local/lib/python3.8/dist-packages (1.0.7)
Requirement already satisfied: numpy<2.0.0,>=1.16.3 in /usr/local/lib/python3.8/dist-packages (from klib) (1.21.6)
Requirement already satisfied: pandas<2.0.0,>=1.2.0 in /usr/local/lib/python3.8/dist-packages (from klib) (1.3.5)
Requirement already satisfied: matplotlib<4.0.0,>=3.0.3 in /usr/local/lib/python3.8/dist-packages (from klib) (3.2.2)
Requirement already satisfied: Jinja2<4.0.0,>=3.0.3 in /usr/local/lib/python3.8/dist-packages (from klib) (3.1.2)
Requirement already satisfied: scipy<2.0.0,>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from klib) (1.7.3)
Requirement already satisfied: seaborn<0.13.0,>=0.11.2 in /usr/local/lib/python3.8/dist-packages (from klib) (0.11.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.8/dist-packages (from Jinja2<4.0.0,>=3.0.3->klib) (2.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<4.0.0,>=3.0.3->klib) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib<4.0.0,>=3.0.3->klib) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<4.0.0,>=3.0.3->klib) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<4.0.0,>=3.0.3->klib) (3.0.9)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas<2.0.0,>=1.2.0->klib) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib<4.0.0,>=3.0.3->klib) (1.15.0)
```

In [200…]:
```
#I have been manually dragging the test and train files to the root folder each time i login
#This codes replces that task. -joel
#from google.colab import files
#uploaded = files.upload()
```

In [201…]:
```
#change color palette for graphs
plt.style.use('seaborn-colorblind')
```

In [202…]:
```
#read in dataset as 'titanic'
titanic = pd.read_csv("train.csv")
titanic.head()
```

Out[202]:

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

In [203…]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [204…]:
```
titanic.describe()
```

Out[204]:

	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 [205…]:
```
#show number of records and attributes
titanic.shape
```

Out[205]:
```
(891, 12)
```

In [206…]:
```
#show null values
titanic.isnull().sum()
```

Out[206]:
```
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
```

There is a significant number of null values for age (20%) and cabin (77%).

In [207…]:
```
#show counts for Survived
titanic['Survived'].value_counts()
```

Out[207]:
```
0    549
1    342
Name: Survived, dtype: int64
```

Cabin

Cabin will be deleted as an attribute due to the high volume of null values.

In [208…]:
```
#Drop Cabin as a column
titanic.drop('Cabin', axis=1, inplace=True)
#titanic.drop('Ticket', axis=1, inplace=True)
```

Partition the Dataset

We want to split the dataset so that 67% of records are in the training set and 33% are in test set.

In [209…]:
```
train, test = train_test_split(titanic, test_size=0.33, random_state=7)

#confirm that dataset has been partitioned correctly
print(titanic.shape)
print(train.shape)
print(test.shape)
```

```
(891, 11)
(596, 11)
(295, 11)
```

The dataset has now been partitioned into a training set ('train') and test set ('test'). Train has 596 (67%) records and test has 295 (33%) records.

In [210…]:
```
#show number of records that survived (Survived=1) or did not survive (Survive=0)
train['Survived'].value_counts()
```

Out[210]:
```
0    373
1    223
Name: Survived, dtype: int64
```

Currently, the training set is imbalanced, with a higher proportion of those that did not survive (Class=0). We will resample the training set so that the proportion of each class is 50/50 to avoid class imbalances during the modeling phase. To find the number of records with Survived=1 we need to resample to have a 50% Survived rate, we use the following equation:

> x = [p(records)-rare]/(1-p), where p=0.50, records=596, rare=223.

We get x=150, so we need to resample 150 records to obtain a Survived rate of 50%.

In [211…]:
```
#resampling of training data set
to_resample=train.loc[train['Survived'] == 1] #isolate all records of Survived=1
our_resample=to_resample.sample(n=150, replace=True) #sample w/ replacement
train_rebal=pd.concat([train, our_resample]) #combine original training set w/ resampled records
train_rebal['Survived'].value_counts()
```

Out[211]:
```
1    373
0    373
Name: Survived, dtype: int64
```

In [212…]:
```
print(titanic.shape)
print(train_rebal.shape)
print(test.shape)
```

```
(891, 11)
(746, 11)
(295, 11)
```

The training set now has 746 records, and the test set has 295 records.

In [213…]:
```
train_rebal.isnull().sum()
```

Out[213]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            146
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         0
dtype: int64
```

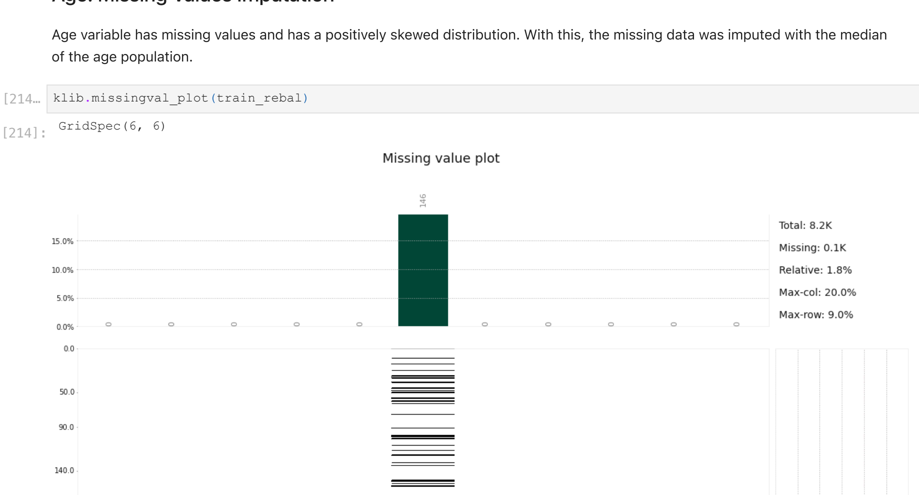
Data transformation

Age: Missing Values Imputation

Age variable has missing values and has a positively skewed distribution. With this, the missing data was imputed with the median of the age population.

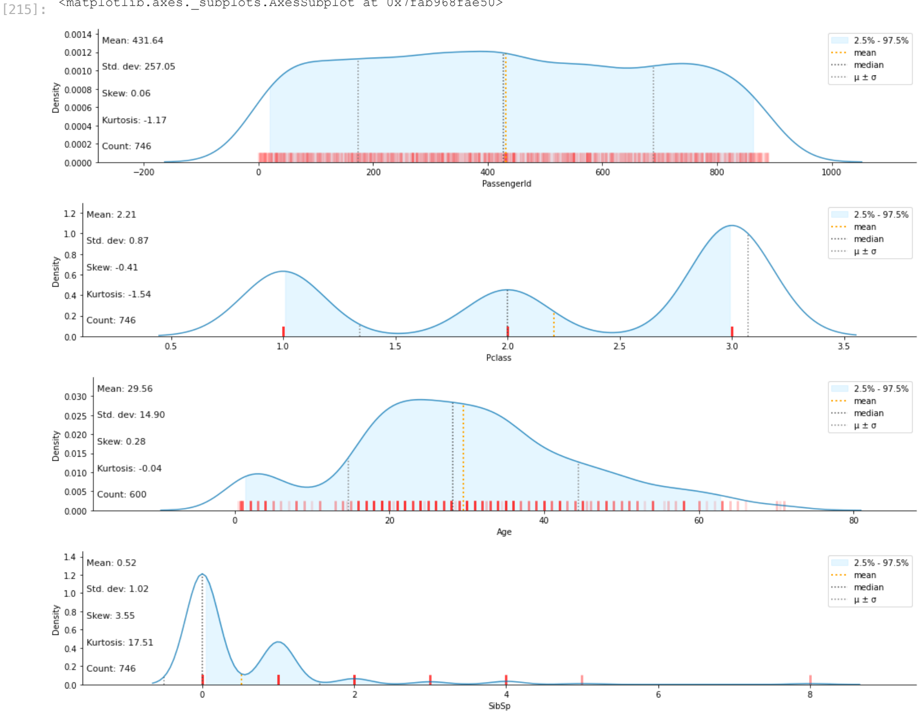
In [214…]:
```
klib.missingval_plot(train_rebal)
```

Out[214]:
```
GridSpec(6, 6)
```



In [215…]:
```
klib.dist_plot(train_rebal)
```

Out[215]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab968fae50>
```



In [216…]:
```
train_rebal.describe()
```

Out[216]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	746.000000	746.000000	746.000000	600.000000	746.000000	746.000000	746.000000
mean	431.640751	0.500000	2.206434	29.556667	0.518767	0.399464	36.630791
std	257.054554	0.500335	0.865606	14.900709	1.024687	0.788304	55.696840
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	211.250000	0.000000	1.000000	20.000000	0.000000	0.000000	7.956250
50%	427.000000	0.500000	2.000000	28.250000	0.000000	0.000000	16.100000
75%	654.000000	1.000000	3.000000	39.000000	1.000000	1.000000	36.940650
max	888.000000	1.000000	3.000000	71.000000	8.000000	6.000000	512.329200

In [217…]:
```
train_rebal['Age']= train_rebal['Age'].fillna(train_rebal['Age'].median())
train_rebal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 746 entries, 65 to 383
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  746 non-null    int64
 1   Survived     746 non-null    int64
 2   Pclass       746 non-null    int64
 3   Name         746 non-null    object
 4   Sex          746 non-null    object
 5   Age          746 non-null    float64
 6   SibSp        746 non-null    int64
 7   Parch        746 non-null    int64
 8   Ticket       746 non-null    object
 9   Fare         746 non-null    float64
 10  Embarked     746 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 69.9+ KB
```

In [218…]:
```
test['Age']= test['Age'].fillna(test['Age'].median())
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 295 entries, 725 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  295 non-null    int64
 1   Survived     295 non-null    int64
 2   Pclass       295 non-null    int64
 3   Name         295 non-null    object
 4   Sex          295 non-null    object
 5   Age          295 non-null    float64
 6   SibSp        295 non-null    int64
 7   Parch        295 non-null    int64
 8   Ticket       295 non-null    object
 9   Fare         295 non-null    float64
 10  Embarked     293 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 27.7+ KB
```

Age: Transform to categorical

Convert Age variable to categorical as follows:

- 'Baby/Toddler' :- 0 to 3 years
- 'Child' :- 4 to 17 years
- 'Adult' :- 18 to 63 years
- 'Elderly' :- 64 to 99 years

In [219…]:
```
train_rebal['Age_c'] = pd.cut(x=train_rebal['Age'], bins=[0, 3, 17, 63, 99],
                    labels=['Baby/Toddler', 'Child', 'Adult',
                            'Elderly'])
train_rebal.head()
```

Out[219]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult

In [220…]:
```
test['Age_c'] = pd.cut(x=test['Age'], bins=[0, 3, 17, 63, 99],
                    labels=['Baby/Toddler', 'Child', 'Adult',
                            'Elderly'])
test.head()
```

Out[220]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c
725	726	0	3	Oreskovic, Mr. Luka	male	20.0	0	0	315094	8.6625	S	Adult
861	862	0	2	Giles, Mr. Frederick Edward	male	21.0	1	0	28134	11.5000	S	Adult
528	529	0	3	Salonen, Mr. Johan Werner	male	39.0	0	0	3101296	7.9250	S	Adult
46	47	0	3	Lennon, Mr. Denis	male	28.0	1	0	370371	15.5000	Q	Adult
627	628	1	1	Longley, Miss. Gretchen Fiske	female	21.0	0	0	13502	77.9583	S	Adult

Family Count: Combined SibSp and Pach

Combined the count of siblings, spouses, parents, and children under Fam column.

In [221…]:
```
train_rebal.eval('Fam = SibSp + Parch', inplace=True)
train_rebal.head()
```

Out[221]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult	2
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child	2
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult	2
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult	1
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult	1

In [222…]:
```
test.eval('Fam = SibSp + Parch', inplace=True)
test.head()
```

Out[222]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam
725	726	0	3	Oreskovic, Mr. Luka	male	20.0	0	0	315094	8.6625	S	Adult	0
861	862	0	2	Giles, Mr. Frederick Edward	male	21.0	1	0	28134	11.5000	S	Adult	1
528	529	0	3	Salonen, Mr. Johan Werner	male	39.0	0	0	3101296	7.9250	S	Adult	0
46	47	0	3	Lennon, Mr. Denis	male	28.0	1	0	370371	15.5000	Q	Adult	1
627	628	1	1	Longley, Miss. Gretchen Fiske	female	21.0	0	0	13502	77.9583	S	Adult	0

Fare: Transform to Catagorcial

In [223…]:
```
train_rebal['Fare2'] = pd.cut(x=train_rebal['Fare'], bins=[0, 8, 18, 50, 300],
                    labels=['Low', 'Mid', 'High','Max'])
train_rebal.head()
```

Out[223]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam	Fare2
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult	2	Mid
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child	2	Max
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult	2	Max
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult	1	Max
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult	1	Max

Ticket: Reduce Catagorical Vairety

In [224…]:
```
train_rebal['Ticket2'] = train_rebal['Ticket'].astype(str).str[0]

#train_rebal['Ticket2'] = pd.cut(x=train_rebal['Ticket'], bins=[0, 8, 18, 50, 300],
                    #labels=['Low', 'Mid', 'High','Max'])
train_rebal['Ticket2'].value_counts()
```

Out[224]:
```
3    241
2    149
1    148
P     58
S     52
C     42
A     16
W      9
4      8
F      7
7      6
6      6
5      2
9      1
L      1
Name: Ticket2, dtype: int64
```

In [225…]:
```
test['Ticket2'] = test['Ticket'].astype(str).str[0]

#test['Ticket2'] = pd.cut(x=train_rebal['Ticket'], bins=[0, 8, 18, 50, 300],
                    #labels=['Low', 'Mid', 'High','Max'])
test['Ticket2'].value_counts()
```

Out[225]:
```
3    88
2    67
1    50
P    22
S    22
C    13
A    13
W     4
4     4
L     3
7     3
6     3
8     2
5     1
Name: Ticket2, dtype: int64
```

Data Exploration

Age and Family

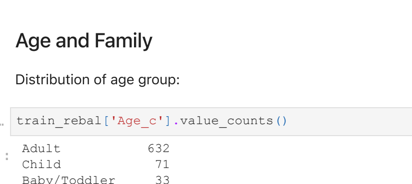
Distribution of age group:

In [226…]:
```
train_rebal['Age_c'].value_counts()
```

Out[226]:
```
Adult           632
Child            71
Baby/Toddler     33
Elderly          10
Name: Age_c, dtype: int64
```

In [227…]:
```
sns.countplot(x=train_rebal["Age_c"])
plt.xlabel('Age Group')
plt.title('Age Distribution')
```

Out[227]:
```
Text(0.5, 1.0, 'Age Distribution')
```



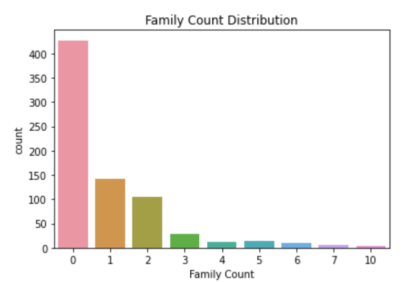
Distribution of family count:

In [228…]:
```
train_rebal['Fam'].value_counts()
```

Out[228]:
```
0     427
1     142
2     104
3      29
5      15
4      11
6       9
7       5
10      4
Name: Fam, dtype: int64
```

In [229…]:
```
sns.countplot(x=train_rebal["Fam"])
plt.xlabel('Family Count')
plt.title('Family Count Distribution')
```

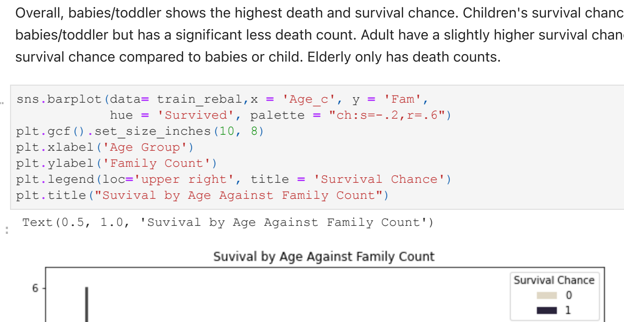
Out[229]:
```
Text(0.5, 1.0, 'Family Count Distribution')
```



Overall, babies/toddler shows the highest death and survival chance. Children's survival chance is not much difference from babies/toddler but has a significant less death count. Adult have a slightly higher survival chance compared to death but less survival chance compared to babies or child. Elderly only has death counts.

In [230…]:
```
sns.barplot(data= train_rebal,x = 'Age_c', y = 'Fam',
            hue = 'Survived', palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.xlabel('Age Group')
plt.ylabel('Family Count')
plt.legend(loc='upper right', title = 'Survival Chance')
plt.title("Suvival by Age Against Family Count")
```

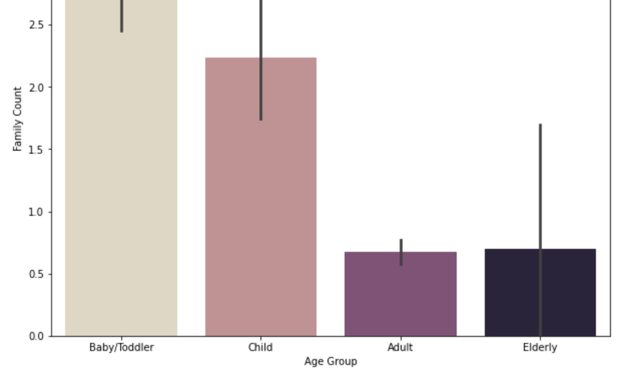
Out[230]:
```
Text(0.5, 1.0, 'Suvival by Age Against Family Count')
```



Plot below shows the relationship between age and number of family members there are with.

In [231…]:
```
sns.barplot(data= train_rebal,x = 'Age_c', y = 'Fam'
            , palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.xlabel('Age Group')
plt.ylabel('Family Count')
plt.title("Age Agaisnt Family Count")
```

Out[231]:
```
Text(0.5, 1.0, 'Age Agaisnt Family Count')
```



Contingency table for age and Survived:

In [232…]:
```
A1 = pd.crosstab(train_rebal['Survived'], train_rebal['Age_c'],
                 margins=True)
A1
```

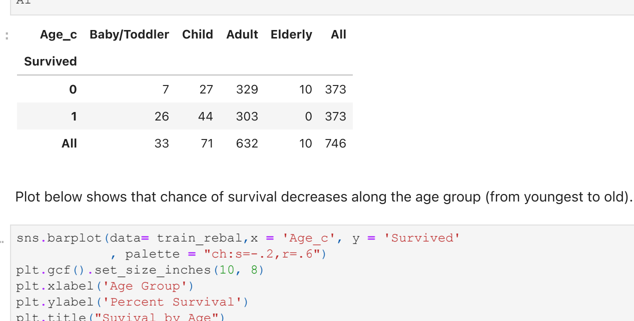
Out[232]:

Age_c	Baby/Toddler	Child	Adult	Elderly	All
Survived					
0	7	27	329	10	373
1	26	44	303	0	373
All	33	71	632	10	746

Plot below shows that chance of survival decreases along the age group (from youngest to old).

In [233…]:
```
sns.barplot(data= train_rebal,x = 'Age_c', y = 'Survived'
            , palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.xlabel('Age Group')
plt.ylabel('Percent Survival')
plt.title("Suvival by Age")
```

Out[233]:
```
Text(0.5, 1.0, 'Suvival by Age')
```



Contingency table for family count and Survived:

In [234…]:
```
F1 = pd.crosstab(train_rebal['Survived'], train_rebal['Fam'],
                 margins=True)
F1
```

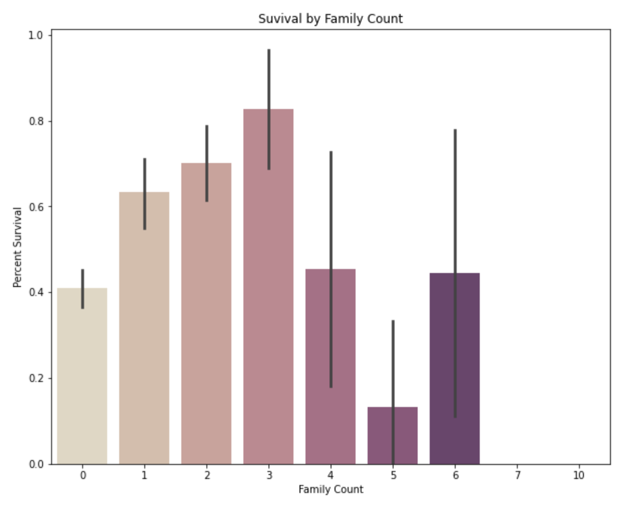
Out[234]:

Fam	0	1	2	3	4	5	6	7	10	All
Survived										
0	252	52	31	5	6	13	5	5	4	373
1	175	90	73	24	5	2	4	0	0	373
All	427	142	104	29	11	15	9	5	4	746

Plot below shows that there is higher chance of survival from having 3 family members.

In [235…]:
```
sns.barplot(data= train_rebal,x = 'Fam', y = 'Survived'
            , palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.xlabel('Family Count')
plt.ylabel('Percent Survival')
plt.title("Suvival by Family Count")
```

Out[235]:
```
Text(0.5, 1.0, 'Suvival by Family Count')
```

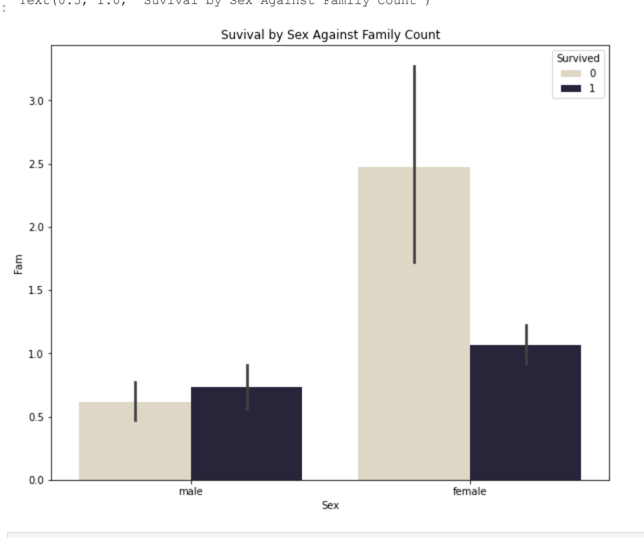


Sex and Family

Overall view, females and having a family count of 3 are attributes that shows higher chance of survival.

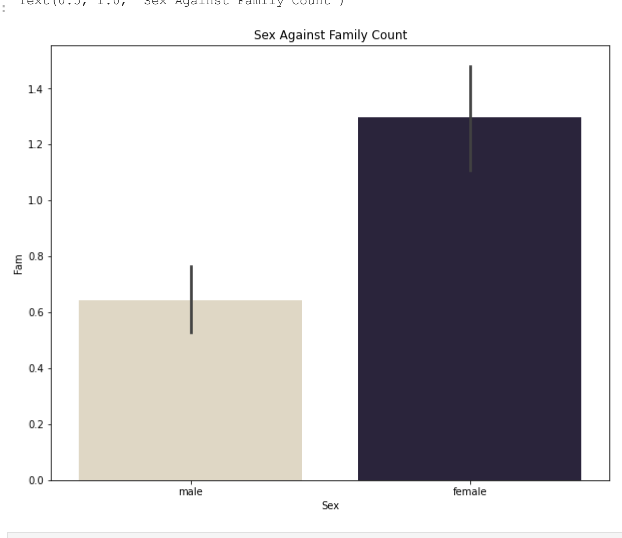
In [236…
```
sns.barplot(data= train_rebal,x = 'Sex', y = 'Fam',
            hue = 'Survived', palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.title("Suvival by Sex Against Family Count")
```

Out[236]: Text(0.5, 1.0, 'Suvival by Sex Against Family Count')



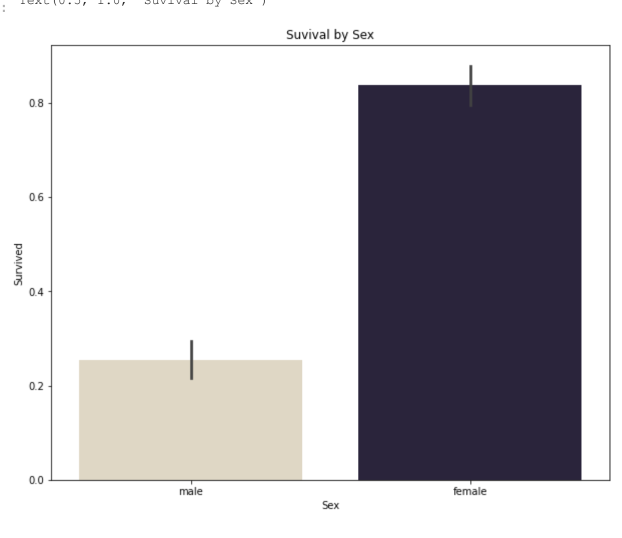
In [237…
```
sns.barplot(data= train_rebal,x = 'Sex', y = 'Fam', palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.title("Sex Against Family Count")
```

Out[237]: Text(0.5, 1.0, 'Sex Against Family Count')



In [238…
```
sns.barplot(data= train_rebal,x = 'Sex', y = 'Survived',
            palette = "ch:s=-.2,r=.6")
plt.gcf().set_size_inches(10, 8)
plt.title("Suvival by Sex")
```

Out[238]: Text(0.5, 1.0, 'Suvival by Sex')



Extract Titles from Names

Titles could give us an indication of marital status and occupation.

In [239…
```
train_rebal['Title'] = (train_rebal['Name'].str.split(',').str[1])
train_rebal['Title'] = (train_rebal['Title'].str.split('.').str[0])
train_rebal.head(10)
```

Out[239]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam	Fare2	Tic
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult	2	Mid	
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child	2	Max	
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult	2	Max	
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult	1	Max	
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult	1	Max	
397	398	0	2	McKane, Mr. Peter David	male	46.00	0	0	28403	26.0000	S	Adult	0	High	
810	811	0	3	Alexander, Mr. William	male	26.00	0	0	3474	7.8875	S	Adult	0	Low	
273	274	0	1	Natsch, Mr. Charles H	male	37.00	0	1	PC 17596	29.7000	C	Adult	1	High	
587	588	1	1	Frolicher-Stehli, Mr. Maxmillian	male	60.00	1	1	13567	79.2000	C	Adult	2	Max	
673	674	1	2	Wilhelms, Mr. Charles	male	31.00	0	0	244270	13.0000	S	Adult	0	Mid	

In [240…
```
#show all titles
train_rebal['Title'].value_counts()
```

Out[240]:
```
Mr              379
Miss            180
Mrs             127
Master           35
Rev               6
Dr                4
Mme               2
Col               2
Sir               2
Ms                2
the Countess      2
Don               1
Major             1
Mlle              1
Jonkheer          1
Capt              1
Name: Title, dtype: int64
```

In [241…
```
#show titles vs sex
train_rebal['Title'] = train_rebal.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
pd.crosstab(train_rebal['Title'], train_rebal['Sex'])
```

Out[241]:

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	2	0
Don	0	1
Dr	0	4
Jonkheer	0	1
Major	0	1
Master	0	35
Miss	180	0
Mlle	1	0
Mme	2	0
Mr	0	379
Mrs	127	0
Ms	2	0
Rev	0	6
Sir	0	2

There are four main titles: Mr., Miss., Mrs., Master. as well as others that denote occupation (e.g Dr., Rev., Capt.).

'Mlle' (Madamoiselle) and 'Ms'(short for Miss) can be combined with 'Miss' to specify unmarried women.

'Mme' (Madame) and 'Countess' (wife of an Earl) can be combined with 'Mrs' to specify married women.

'Master' is usually used for underage boys < 18 y.o

Other titles are all held by men, and will be combined with 'Mr'.

In [242…
```
#combine all titles into four main categories (Miss, Mrs, Mr, Master)
train_rebal['Title'] = train_rebal['Title'].replace(['Capt', 'Col','Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkhee

train_rebal['Title'] = train_rebal['Title'].replace('Mlle', 'Miss')
train_rebal['Title'] = train_rebal['Title'].replace('Ms', 'Miss')
train_rebal['Title'] = train_rebal['Title'].replace('Countess', 'Mrs')
train_rebal['Title'] = train_rebal['Title'].replace('Mme', 'Mrs')
```

In [243…
```
#show new value counts
train_rebal['Title'].value_counts()
```

Out[243]:
```
Mr        397
Miss      183
Mrs       131
Master     35
Name: Title, dtype: int64
```

In [244…
```
#create a contingency table of survival counts according to title
crosstab_title=pd.crosstab(train_rebal['Survived'], train_rebal['Title'])
crosstab_title
```

Out[244]:

Title	Master	Miss	Mr	Mrs
Survived				
0	10	32	312	19
1	25	151	85	112

In [245…
```
#show likelihood of survival
train_rebal[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

Out[245]:

	Title	Survived
0	Master	0.714286
1	Miss	0.825137
2	Mr	0.214106
3	Mrs	0.854962

In [246…
```
#show proportions
round(crosstab_title.div(crosstab_title.sum(0), axis=1)*100, 1)
```

Out[246]:

Title	Master	Miss	Mr	Mrs
Survived				
0	28.6	17.5	78.6	14.5
1	71.4	82.5	21.4	85.5

Most likely to survive:

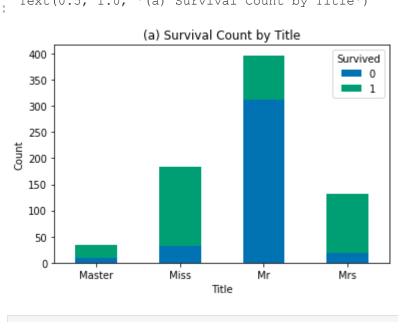
```
Unmarried women, Miss

Married women, Mrs

Young boys, Master
```

Least likely to surive:

```
Men, Mr
```

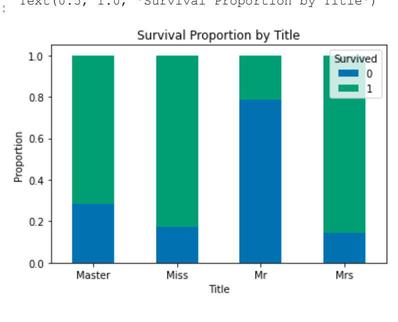
In [247…
```
#barchart of Title vs. Survived

pd.crosstab(train_rebal['Title'], train_rebal['Survived']).plot(kind='bar',
                                                                stacked=True)
plt.xticks(rotation=360)
plt.ylabel('Count')
plt.title('(a) Survival Count by Title')
```

Out[247]: Text(0.5, 1.0, '(a) Survival Count by Title')



In [248…
```
#normalized barchart for survival by title
norm_crosstab_title=pd.crosstab(train_rebal['Title'], train_rebal['Survived']).div(pd.crosstab(train_rebal['Tit
norm_crosstab_title.plot(kind='bar', stacked=True)
plt.xticks(rotation=360)
plt.ylabel('Proportion')
plt.title('Survival Proportion by Title')
```

Out[248]: Text(0.5, 1.0, 'Survival Proportion by Title')



Sex

In [249…
```
train_rebal['Sex'].value_counts()
```

Out[249]:
```
male      432
female    314
Name: Sex, dtype: int64
```

Sex is a binary, nominal variable with two values: Male or Female.

In [250…
```
#create a contingency table survial counts according to sex
crosstab_sex=pd.crosstab(train_rebal['Sex'], train_rebal['Survived'])
#crosstab_sex['Total']=crosstab_sex.sum(axis=1)
#crosstab_sex.loc['Total'] = crosstab_sex.sum()
crosstab_sex
```

Out[250]:

Survived	0	1
Sex		
female	51	263
male	322	110

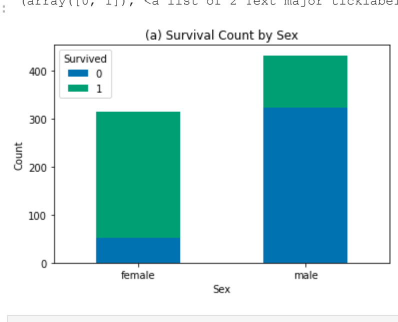
In [251…
```
#show proportions
crosstab_sex2=pd.crosstab(train_rebal['Survived'], train_rebal['Sex'])
round(crosstab_sex2.div(crosstab_sex2.sum(0), axis=1)*100, 1)
```

Out[251]:

Sex	female	male
Survived		
0	16.2	74.5
1	83.8	25.5

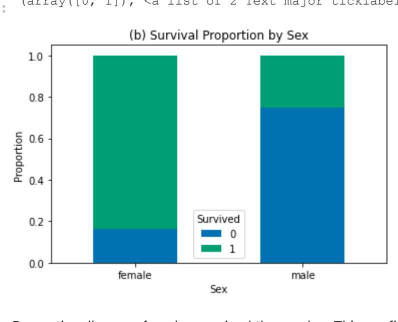
In [252…
```
#barchart of Sex vs. Survived
crosstab_sex.plot(kind='bar', stacked=True)
plt.ylabel('Count')
plt.title('(a) Survival Count by Sex')
plt.xticks(rotation=360)
```

Out[252]: (array([0, 1]), <a list of 2 Text major ticklabel objects>)



In [253…
```
#normalized bar graph of survival proportions according to sex
norm_crosstab_sex=crosstab_sex.div(crosstab_sex.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_sex.plot(kind='bar', stacked=True)
plt.ylabel('Proportion')
plt.title('(b) Survival Proportion by Sex')
plt.xticks(rotation=360)
```

Out[253]: (array([0, 1]), <a list of 2 Text major ticklabel objects>)



Proportionally more females survived than males. This confirms what we already know, since it is common knowledge that women and children were the first to board the rescue boats.

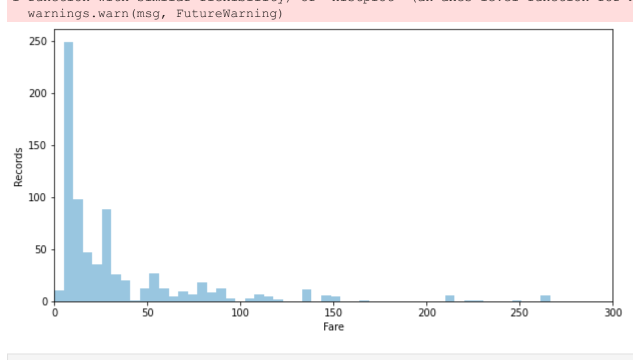
Fare

In [254…
```
train_rebal['Fare'].describe()
```

Out[254]:
```
count    746.000000
mean      36.630791
std       55.696840
min        0.000000
25%        7.956250
50%       16.100000
75%       36.940650
max      512.329200
Name: Fare, dtype: float64
```

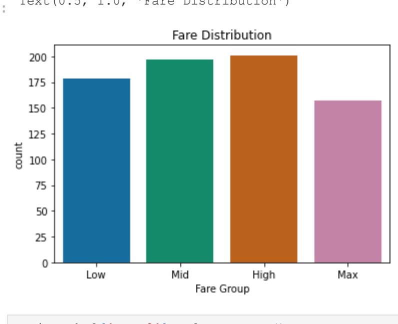
In [255…
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.xlim(0,300)
plt.xlabel('Fare')
plt.ylabel('Records')
sns.distplot(train_rebal['Fare'],bins=100,kde=False)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [256…
```
sns.countplot(x=train_rebal["Fare2"])
plt.xlabel('Fare Group')
plt.title('Fare Distribution')
```

Out[256]: Text(0.5, 1.0, 'Fare Distribution')

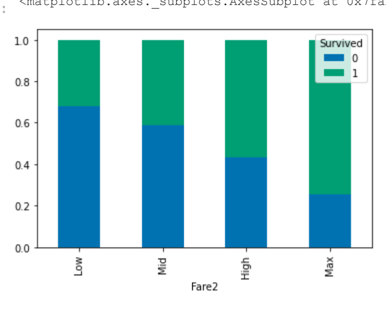


In [257…
```
train_rebal['Fare2'].value_counts()
```

Out[257]:
```
High    201
Mid     197
Low     178
Max     157
Name: Fare2, dtype: int64
```

In [258…
```
crosstab_fare=pd.crosstab(train_rebal['Fare2'], train_rebal['Survived'])
norm_crosstab_fare=crosstab_fare.div(crosstab_fare.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_fare.plot(kind='bar', stacked=True)
```

Out[258]: <matplotlib.axes._subplots.AxesSubplot at 0x7fab91ae7670>



Embarked

In [259…
```
train_rebal['Embarked'].value_counts()
```

Out[259]:
```
S    537
C    147
Q     62
Name: Embarked, dtype: int64
```

    `# This is formatted as code`

Embarked is a nominal categorical variable that defines the port of embarkation for each passenger. There are three possible values: C (Cherbourg), Q (Queenstown), and S (South Hampton).

The mode (S) is used to impute for null values.

In [260…
```
#impute for missing values
train_rebal['Embarked']=train_rebal['Embarked'].fillna('S')
train_rebal['Embarked'].isnull().sum()
```

Out[260]: 0

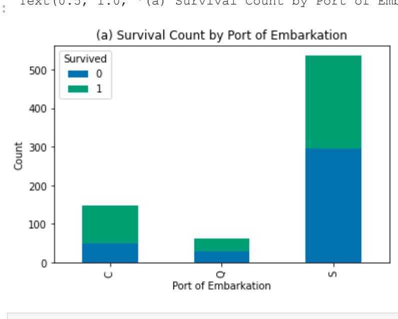
In [261…
```
#create a contingency table of survival counts for each port of embarkation
crosstab_embk=pd.crosstab(train_rebal['Embarked'], train_rebal['Survived'])
#crosstab_embk['Total']=crosstab_embk.sum(axis=1)
#crosstab_embk.loc['Total'] = crosstab_embk.sum()
crosstab_embk
```

Out[261]:

Survived	0	1
Embarked		
C	49	98
Q	29	33
S	295	242

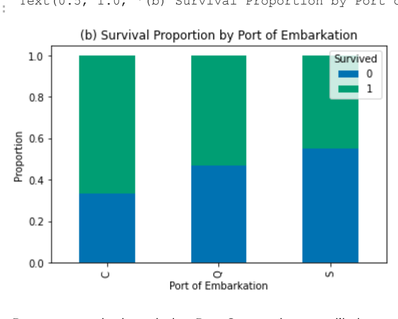
In [262…
```
#barchart of Sex vs. Survived
crosstab_embk.plot(kind='bar', stacked=True)
plt.xlabel('Port of Embarkation')
plt.ylabel('Count')
plt.title('(a) Survival Count by Port of Embarkation')
```

Out[262]: Text(0.5, 1.0, '(a) Survival Count by Port of Embarkation')



In [263…
```
#show proportions
crosstab_embk = round(crosstab_embk.div(crosstab_embk.sum(0), axis=1)*100, 1)
norm_crosstab_embk=crosstab_embk.div(crosstab_embk.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_embk.plot(kind='bar', stacked=True)
plt.xlabel('Port of Embarkation')
plt.ylabel('Proportion')
plt.title('(b) Survival Proportion by Port of Embarkation')
```

Out[263]: Text(0.5, 1.0, '(b) Survival Proportion by Port of Embarkation')



Passengers who boarded at Port C were the most likely to survive, followed by Q, and then S.

Pclass

In [264…
```
train_rebal['Pclass'].value_counts()
```

Out[264]:
```
3    372
1    218
2    156
Name: Pclass, dtype: int64
```

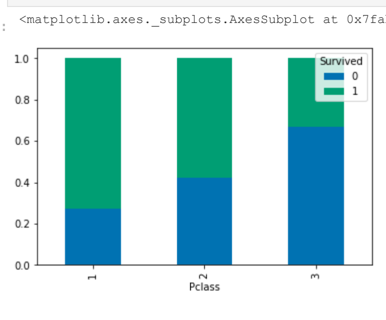
In [265…
```
crosstab_Pclass = pd.crosstab(train_rebal['Pclass'], train_rebal['Survived'])
crosstab_Pclass
```

Out[265]:

Survived	0	1
Pclass		
1	59	159
2	66	90
3	248	124

In [266…
```
crosstab_Pclass = round(crosstab_Pclass.div(crosstab_Pclass.sum(0), axis=1)*100, 1)
norm_crosstab_Pclass= crosstab_Pclass.div(crosstab_Pclass.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_Pclass.plot(kind='bar', stacked=True)
```

Out[266]: <matplotlib.axes._subplots.AxesSubplot at 0x7fab91eee7f0>



Ticket

In [267…
```
train_rebal['Ticket2'].value_counts()
```

Out[267]:
```
3    241
2    149
1    148
P     58
S     52
C     42
A     16
W      9
4      8
F      7
7      6
6      6
5      2
9      1
L      1
Name: Ticket2, dtype: int64
```

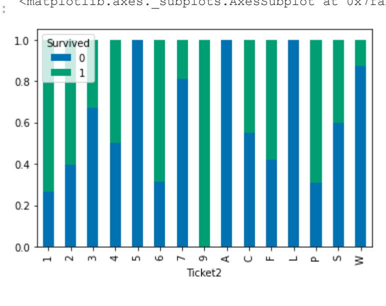
In [268…
```
crosstab_tick=pd.crosstab(train_rebal['Ticket2'], train_rebal['Survived'])
crosstab_tick
```

Out[268]:

Survived	0	1
Ticket2		
1	39	109
2	59	90
3	162	79
4	4	4
5	2	0
6	2	4
7	5	1
9	0	1
A	16	0
C	23	19
F	3	4
L	1	0
P	18	40
S	31	21
W	8	1

In [269…
```
crosstab_tick = round(crosstab_tick.div(crosstab_tick.sum(0), axis=1)*100, 1)
norm_crosstab_tick=crosstab_tick.div(crosstab_tick.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_tick.plot(kind='bar', stacked=True)
```

Out[269]: <matplotlib.axes._subplots.AxesSubplot at 0x7fab91b87040>



Fare2 vs Ticket2

In [270…
```
crossa =pd.crosstab(train_rebal['Fare2'], train_rebal['Ticket2'])
crossa
```

Out[270]:

Ticket2	1	2	3	4	5	6	7	9	A	C	F	P	S	W
Fare2														
Low	8	21	116	0	0	0	0	1	6	3	1	0	21	1
Mid	1	59	71	7	1	4	6	0	9	13	1	3	21	1
High	53	56	37	1	1	2	0	0	1	22	4	14	6	4
Max	82	9	17	0	0	0	0	0	0	4	1	37	4	3

In [270…

In [271…
```
resa = chi2_contingency(crossa)
print('The P-Value is: ', resa[1])
```

```
The P-Value is:  4.103614231972349e-72
```

A very low p value indicates that we should reject the null hypothesis and accept the sad truth that these two are highly ocrrelated.

Pclass vs Embark

In [272…
```
cross =pd.crosstab(train_rebal['Embarked'], train_rebal['Pclass'])
cross
```

Out[272]:

Pclass	1	2	3
Embarked			
C	82	11	54
Q	3	3	56
S	133	142	262

In [273…
```
res = chi2_contingency(cross)
print('The P-Value is: ', res[1])
```

```
The P-Value is:  6.63083673802121e-22
```

A very low p value indicates that we should reject the null hypothesis and accept the sad truth that these two are highly ocrrelated.

Prepare Data for Modeling

Convert Categorical Variables into Dummies

Training Set

In [274…
```
sex_dummies = pd.get_dummies(train_rebal['Sex'], prefix = 'Sex', drop_first=False)
sex_dummies
```

Out[274]:

	Sex_female	Sex_male
65	0	1
445	0	1
659	0	1
591	1	0
690	0	1
...	...	...
275	1	0
839	0	1
577	1	0
400	0	1
383	1	0

746 rows × 2 columns

In [274…

In [275…
```
fare_dummies = pd.get_dummies(train_rebal['Sex'], prefix = 'Sex', drop_first=False)
sex_dummies
```

Out[275]:

	Sex_female	Sex_male
65	0	1
445	0	1
659	0	1
591	1	0
690	0	1
...	...	...
275	1	0
839	0	1
577	1	0
400	0	1
383	1	0

746 rows × 2 columns

In [276…
```
pclass_dummies = pd.get_dummies(train_rebal['Pclass'], prefix = 'Pclass', drop_first=False)
pclass_dummies
```

Out[276]:

	Pclass_1	Pclass_2	Pclass_3
65	0	0	1
445	1	0	0
659	1	0	0
591	1	0	0
690	1	0	0
...	...	...	...
275	1	0	0
839	1	0	0
577	1	0	0
400	0	0	1
383	1	0	0

746 rows × 3 columns

In [277…
```
embarked_dummies = pd.get_dummies(train_rebal['Embarked'], prefix = 'Embarked', drop_first=False)
embarked_dummies
```

Out[277]:

	Embarked_C	Embarked_Q	Embarked_S
65	1	0	0
445	0	0	1
659	1	0	0
591	1	0	0
690	0	0	1
...	...	...	...
275	0	0	1
839	1	0	0
577	0	0	1
400	0	0	1
383	0	0	1

746 rows × 3 columns

In [278…
```
age_dummies = pd.get_dummies(train_rebal['Age_c'], prefix = 'Age', drop_first=False)
age_dummies
```

Out[278]:

	Age_Baby/Toddler	Age_Child	Age_Adult	Age_Elderly
65	0	0	1	0
445	0	1	0	0
659	0	0	1	0
591	0	0	1	0
690	0	0	1	0
...	...	...	...	...
275	0	0	1	0
839	0	0	1	0
577	0	0	1	0
400	0	0	1	0
383	0	0	1	0

746 rows × 4 columns

In [279…
```
ticket2_dummies = pd.get_dummies(train_rebal['Ticket2'], prefix = 'Ticket2', drop_first=False)
ticket2_dummies
```

Out[279]:

	Ticket2_1	Ticket2_2	Ticket2_3	Ticket2_4	Ticket2_5	Ticket2_6	Ticket2_7	Ticket2_9	Ticket2_A	Ticket2_C	Ticket2_F	Ticke
65	0	1	0	0	0	0	0	0	0	0	0	
445	0	0	1	0	0	0	0	0	0	0	0	
659	0	0	1	0	0	0	0	0	0	0	0	
591	0	0	1	0	0	0	0	0	0	0	0	
690	1	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	
275	1	0	0	0	0	0	0	0	0	0	0	
839	1	0	0	0	0	0	0	0	0	0	0	
577	1	0	0	0	0	0	0	0	0	0	0	
400	0	0	0	0	0	0	0	0	0	0	0	
383	1	0	0	0	0	0	0	0	0	0	0	

746 rows × 15 columns

Test Set

In [280…
```
sex_dummies_test = pd.get_dummies(test['Sex'], prefix = 'Sex', drop_first=False)
pclass_dummies_test = pd.get_dummies(test['Pclass'], prefix = 'Pclass', drop_first=False)
embarked_dummies_test = pd.get_dummies(test['Embarked'], prefix = 'Embarked', drop_first=False)
age_dummies_test = pd.get_dummies(test['Age_c'], prefix = 'Age', drop_first=False)
ticket2_dummies_test = pd.get_dummies(test['Ticket2'], prefix = 'Ticket2', drop_first=False)
```

Standardize Numeric Variables

Numeric variables are normalized using Min-Max scaling

Numeric variables include: Age, Fare, Fam

In [281…
```
#normalized variables are added back into original df as new columns 'Var_mm'
train_rebal['Age_mm'] = MinMaxScaler().fit_transform(train_rebal[['Age']])
#train_rebal['Fare_mm'] = MinMaxScaler().fit_transform(train_rebal[['Fare']])
#train_rebal['Fam_mm'] = MinMaxScaler().fit_transform(train_rebal[['Fam']])
train_rebal.head(10)
```

Out[281]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam	Fare2	Tic
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult	2	Mid	
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child	2	Max	
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult	2	Max	
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult	1	Max	
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult	1	Max	
397	398	0	2	McKane, Mr. Peter David	male	46.00	0	0	28403	26.0000	S	Adult	0	High	
810	811	0	3	Alexander, Mr. William	male	26.00	0	0	3474	7.8875	S	Adult	0	Low	
273	274	0	1	Natsch, Mr. Charles H	male	37.00	0	1	PC 17596	29.7000	C	Adult	1	High	
587	588	1	1	Frolicher-Stehli, Mr. Maxmillian	male	60.00	1	1	13567	79.2000	C	Adult	2	Max	
673	674	1	2	Wilhelms, Mr. Charles	male	31.00	0	0	244270	13.0000	S	Adult	0	Mid	

Models

Logistic Regression

In [282…
```
train_rebal.head()
```

Out[282]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam	Fare2	Tick
65	66	1	3	Moubarek, Master. Gerios	male	28.25	1	1	2661	15.2458	C	Adult	2	Mid	
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	S	Child	2	Max	
659	660	0	1	Newell, Mr. Arthur Webster	male	58.00	0	2	35273	113.2750	C	Adult	2	Max	
591	592	1	1	Stephenson, Mrs. Walter Bertram (Martha Eustis)	female	52.00	1	0	36947	78.2667	C	Adult	1	Max	
690	691	1	1	Dick, Mr. Albert Adrian	male	31.00	1	0	17474	57.0000	S	Adult	1	Max	

In [283…
```
train_rebal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 746 entries, 65 to 383
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  746 non-null    int64
 1   Survived     746 non-null    int64
 2   Pclass       746 non-null    int64
 3   Name         746 non-null    object
 4   Sex          746 non-null    object
 5   Age          746 non-null    float64
 6   SibSp        746 non-null    int64
 7   Parch        746 non-null    int64
 8   Ticket       746 non-null    object
 9   Fare         746 non-null    float64
 10  Embarked     746 non-null    object
 11  Age_c        746 non-null    category
 12  Fam          746 non-null    int64
 13  Fare2        733 non-null    category
 14  Ticket2      746 non-null    object
 15  Title        746 non-null    object
 16  Age_mm       746 non-null    float64
dtypes: category(2), float64(3), int64(6), object(6)
memory usage: 111.3+ KB
```

Correlation Matrix:

*Note that corr() applies to numeric variables only.

In [284…
```
train_rebal.corr()
```

Out[284]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Fam	Age_mm
PassengerId	1.000000	-0.035589	0.010040	0.001134	-0.013229	0.095082	0.006657	0.040526	0.001134
Survived	-0.035589	1.000000	-0.347120	-0.074136	-0.032727	0.108903	0.233654	0.034530	-0.074136
Pclass	0.010040	-0.347120	1.000000	-0.348372	0.062213	0.012752	-0.552360	0.048712	-0.348372
Age	0.001134	-0.074136	-0.348372	1.000000	-0.211754	-0.205865	0.129484	-0.250335	1.000000
SibSp	-0.013229	-0.032727	0.062213	-0.211754	1.000000	0.386197	0.131973	0.877294	-0.211754
Parch	0.095082	0.108903	0.012752	-0.205865	0.386197	1.000000	0.194591	0.781525	-0.205865
Fare	0.006657	0.233654	-0.552360	0.129484	0.131973	0.194591	1.000000	0.190509	0.129484
Fam	0.040526	0.034530	0.048712	-0.250335	0.877294	0.781525	0.190509	1.000000	-0.250335
Age_mm	0.001134	-0.074136	-0.348372	1.000000	-0.211754	-0.205865	0.129484	-0.250335	1.000000

Prepare separate predictors from response:

In [285…
```
#Predictors
X_logi = pd.DataFrame(train_rebal[['Pclass', 'Fam', 'Age', 'Fare', 'Sex']])


X_logi
```

Out[285]:

	Pclass	Fam	Age	Fare	Sex
65	3	2	28.25	15.2458	male
445	1	2	4.00	81.8583	male
659	1	2	58.00	113.2750	male
591	1	1	52.00	78.2667	female
690	1	1	31.00	57.0000	male
...	...	...	...	...	...
275	1	1	63.00	77.9583	female
839	1	0	28.25	29.7000	male
577	1	1	39.00	55.9000	female
400	3	0	39.00	7.9250	male
383	1	1	35.00	52.0000	female

746 rows × 5 columns

In [286…
```
#Response
y_logi = pd.DataFrame(train_rebal[['Survived']])
y_logi
```

Out[286]:

	Survived
65	1
445	1
659	0
591	1
690	1
...	...
275	1
839	1
577	1
400	1
383	1

746 rows × 1 columns

Convert catergorical variable to numerical:

In [287…
```
X_logi['Sex'].replace(['male', 'female'],
                        [0, 1], inplace=True)
```

Convert predictors to dummy variables:

In [288…
```
X_logi_d = pd.get_dummies(X_logi)


X_logi_d.head()
```

Out[288]:

	Pclass	Fam	Age	Fare	Sex
65	3	2	28.25	15.2458	0
445	1	2	4.00	81.8583	0
659	1	2	58.00	113.2750	0
591	1	1	52.00	78.2667	1
690	1	1	31.00	57.0000	0

Logistic Regression Model:

In [289…
```
#Add contant to X_logi
X_logi_d = sm.add_constant(X_logi_d)


logreg01 = sm.Logit(y_logi, X_logi_d).fit()
logreg01.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.428228
         Iterations 6
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of p
andas all arguments of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[289]:

Model:	Logit	Pseudo R-squared:	0.382
Dependent Variable:	Survived	AIC:	650.9157
Date:	2022-12-09 22:55	BIC:	678.6040
No. Observations:	746	Log-Likelihood:	-319.46
Df Model:	5	LL-Null:	-517.09
Df Residuals:	740	LLR p-value:	3.1177e-83
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	3.2571	0.5057	6.4405	0.0000	2.2659	4.2483
Pclass	-1.2666	0.1502	-8.4350	0.0000	-1.5609	-0.9723
Fam	-0.2664	0.0726	-3.6709	0.0002	-0.4087	-0.1242
Age	-0.0503	0.0084	-5.9807	0.0000	-0.0668	-0.0338
Fare	0.0009	0.0021	0.4048	0.6856	-0.0033	0.0050
Sex	3.1643	0.2331	13.5721	0.0000	2.7073	3.6212

Logistic Regression Shaving:

Removed Ticket2 because it's showing NANs for the metrics.

Embarked is showing multicollinearity with very high Std Error, 0 values for z score, and p-value of 1.

Logistic Regression Interpreattion:

Model has a pseudo R^2 of 36.3% which suggest that the model is a good fit. LLR p-value is less than 0.5 significance level. BIC is slightly higher than AIC but they are around the same ball park. All the varible metrics look reasonable.

In [290…
```
lg_pred_tr = logreg01.predict(X_logi_d)
lg_pred_tr_logis = np.where (lg_pred_tr > 0.5, 1, 0)
```

Test Dataset Validation

In [291…
```
test.head()
```

Out[291]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Age_c	Fam	Ticket2
725	726	0	3	Oreskovic, Mr. Luka	male	20.0	0	0	315094	8.6625	S	Adult	0	3
861	862	0	2	Giles, Mr. Frederick Edward	male	21.0	1	0	28134	11.5000	S	Adult	1	2
528	529	0	3	Salonen, Mr. Johan Werner	male	39.0	0	0	3101296	7.9250	S	Adult	0	3
46	47	0	3	Lennon, Mr. Denis	male	28.0	1	0	370371	15.5000	Q	Adult	1	3
627	628	1	1	Longley, Miss. Gretchen Fiske	female	21.0	0	0	13502	77.9583	S	Adult	0	1

In [292…
```
X_logi_test = pd.DataFrame(test[['Pclass', 'Fam', 'Age', 'Fare', 'Sex']])
y_logi_test = pd.DataFrame(test[['Survived']])
```

In [293…
```
X_logi_test['Sex'].replace(['male', 'female'],
                        [0, 1], inplace=True)
X_logi_d_test = pd.get_dummies(X_logi_test)
```

In [294…
```
X_logi_d_test
```

Out[294]:

	Pclass	Fam	Age	Fare	Sex
725	3	0	20.0	8.6625	0
861	2	1	21.0	11.5000	0
528	3	0	39.0	7.9250	0
46	3	1	28.0	15.5000	0
627	1	0	21.0	77.9583	1
...	...	...	...	...	...
360	3	5	40.0	27.9000	0
856	1	2	45.0	164.8667	1
199	2	0	24.0	13.0000	1
451	3	1	28.0	19.9667	0
417	2	2	18.0	13.0000	1

295 rows × 5 columns

In [295…
```
X_logi_d_test = sm.add_constant(X_logi_d_test)

logreg01_test = sm.Logit(y_logi_test, X_logi_d_test).fit()
logreg01_test.summary2()
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of p
andas all arguments of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
Optimization terminated successfully.
         Current function value: 0.485070
         Iterations 7
```

Out[295]:

Model:	Logit	Pseudo R-squared:	0.281
Dependent Variable:	Survived	AIC:	298.1912
Date:	2022-12-09 22:55	BIC:	320.3130
No. Observations:	295	Log-Likelihood:	-143.10
Df Model:	5	LL-Null:	-198.94
Df Residuals:	289	LLR p-value:	1.8066e-22
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	0.4190	0.8941	0.4686	0.6393	-1.3334	2.1714
Pclass	-0.6449	0.2470	-2.6113	0.0090	-1.1289	-0.1609
Fam	-0.2508	0.1225	-2.0473	0.0406	-0.4909	-0.0107
Age	-0.0147	0.0129	-1.1432	0.2530	-0.0400	0.0105
Fare	0.0175	0.0081	2.1679	0.0302	0.0017	0.0333
Sex	2.2155	0.3216	6.8896	0.0000	1.5852	2.8458

Contingency Table

Predictions:

In [296…
```
predictions_prob = logreg01_test.predict(X_logi_d_test)
predictions_prob.head()
```

Out[296]:
```
725    0.159946
861    0.226244
528    0.124425
46     0.129246
627    0.954487
dtype: float64
```

In [297…
```
cutoff = 0.5
```

In [298…
```
ypred_logis = np.where (predictions_prob > cutoff, 1, 0)
#ypred_c['Survived'] = pd.DataFrame(ypred_logis)
#ypred_c
```

In [299…
```
conf_matrix = pd.crosstab(test['Survived'], ypred_logis,
                          rownames = ["Actual"],
                          colnames = ["Predicted"],
                          margins = True)
conf_matrix
```

Out[299]:

Predicted	0	1	All
Actual			
0	149	27	176
1	36	83	119
All	185	110	295

Evaluation Metrics

In [300…
```
#Baseline
#Accuracy (all negative model) = TAN/GT

logis_baseline = round((176/296)*100, 2)
logis_baseline
```

Out[300]:
```
59.46
```

In [301…
```
#Accuracy = (TN+TP) / GT

logis_a = round(((150+82)/295)*100, 2)
logis_a
```

Out[301]:
```
78.64
```

In [302…
```
#Error rate = 1-Accuracy

logis_e = round(100-78.38, 2)
logis_e
```

Out[302]:
```
21.62
```

In [303…
```
#Sensitivity: Recall = TP/TAP

logis_r = round((82/119)*100, 2)
logis_r
```

Out[303]:
```
68.91
```

In [304…
```
#Specificity: Specificity = TN/TAN

logis_s = round((150/176)*100, 2)
logis_s
```

Out[304]:
```
85.23
```

In [305…
```
logis_t = [ ['Metrics', 'Score, %'],
          ['Accuracy, base', logis_baseline],
          ['Accuracy', logis_a],
          ['Error rate', logis_e],
          ['Sensitivity', logis_r],
          ['Specificity', logis_s],
          ]

print(tabulate(logis_t, headers= 'firstrow'))
```

```
Metrics           Score, %
--------------  ----------
Accuracy, base       59.46
Accuracy             78.64
Error rate           21.62
Sensitivity          68.91
Specificity          85.23
```

K-means Clustering

In [306…
```
#Predictors
X_kmeans = train_rebal[['Pclass', 'Fam',
                        'Age', 'Fare']]
X_kmeans
```

Out[306]:

	Pclass	Fam	Age	Fare
65	3	2	28.25	15.2458
445	1	2	4.00	81.8583
659	1	2	58.00	113.2750
591	1	1	52.00	78.2667
690	1	1	31.00	57.0000
...	...	...	...	...
275	1	1	63.00	77.9583
839	1	0	28.25	29.7000
577	1	1	39.00	55.9000
400	3	0	39.00	7.9250
383	1	1	35.00	52.0000

746 rows × 4 columns

Standardize predictors using Z-score transformation:

*Note: only numeric data

In [307…
```
X_kmns = pd.DataFrame(stats.zscore(X_kmeans),
                      columns=['Pclass', 'Fam', 'Age','Fare'])
```

K-means clustering model:

In [308…
```
kmeans01 = KMeans(n_clusters = 4).fit(X_kmns)

kmeans01
```

Out[308]:
```
KMeans(n_clusters=4)
```

Investigate:

In [309…
```
cluster = kmeans01.labels_
```

In [310…
```
Cluster1 = X_kmeans.loc[cluster == 0]

Cluster2 = X_kmeans.loc[cluster == 1]

Cluster3 = X_kmeans.loc[cluster == 2]

Cluster4 = X_kmeans.loc[cluster == 3]
```

In [311…
```
Cluster1.describe()
```

Out[311]:

	Pclass	Fam	Age	Fare
count	103.000000	103.000000	103.000000	103.000000
mean	2.611650	3.533981	11.902913	32.175321
std	0.564176	2.066614	11.782431	19.788354
min	1.000000	1.000000	0.420000	7.854200
25%	2.000000	2.000000	2.000000	19.258300
50%	3.000000	3.000000	8.000000	27.750000
75%	3.000000	5.000000	19.500000	36.877100
max	3.000000	10.000000	48.000000	120.000000

In [312…
```
Cluster2.describe()
```

Out[312]:

	Pclass	Fam	Age	Fare
count	232.000000	232.000000	232.000000	232.000000
mean	1.168103	0.741379	39.114224	59.230280
std	0.374767	0.854027	12.762562	38.327296
min	1.000000	0.000000	16.000000	0.000000
25%	1.000000	0.000000	28.250000	26.550000
50%	1.000000	1.000000	36.000000	52.000000
75%	1.000000	1.000000	48.250000	83.475000
max	2.000000	4.000000	71.000000	153.462500

In [313…
```
Cluster3.describe()
```

Out[313]:

	Pclass	Fam	Age	Fare
count	390.000000	390.000000	390.000000	390.000000
mean	2.782051	0.276923	28.014744	11.188694
std	0.413383	0.591483	7.940711	6.942283
min	2.000000	0.000000	11.000000	0.000000
25%	3.000000	0.000000	23.250000	7.750000
50%	3.000000	0.000000	28.250000	8.050000
75%	3.000000	0.000000	30.000000	13.000000
max	3.000000	2.000000	70.500000	73.500000

In [314…
```
Cluster4.describe()
```

Out[314]:

	Pclass	Fam	Age	Fare
count	21.0	21.000000	21.000000	21.000000
mean	1.0	1.952381	30.107143	281.309333
std	0.0	1.935877	11.629319	118.925800
min	1.0	0.000000	15.000000	151.550000
25%	1.0	0.000000	23.000000	211.337500
50%	1.0	1.000000	28.250000	247.520800
75%	1.0	4.000000	35.000000	263.000000
max	1.0	5.000000	64.000000	512.329200

Test Dataset Validation

In [315…
```
X_kmeans_test = test[['Pclass', 'Fam',
                      'Age', 'Fare']]
X_kmeans_test
```

Out[315]:

	Pclass	Fam	Age	Fare
725	3	0	20.0	8.6625
861	2	1	21.0	11.5000
528	3	0	39.0	7.9250
46	3	1	28.0	15.5000
627	1	0	21.0	77.9583
...	...	...	...	...
360	3	5	40.0	27.9000
856	1	2	45.0	164.8667
199	2	0	24.0	13.0000
451	3	1	28.0	19.9667
417	2	2	18.0	13.0000

295 rows × 4 columns

In [316…
```
X_kmns_test = pd.DataFrame(stats.zscore(X_kmeans_test),
                      columns=['Pclass', 'Fam', 'Age','Fare'])

kmeans_test = KMeans(n_clusters = 4).fit(X_kmns_test)

cluster_test = kmeans_test.labels_
```

In [317…
```
Cluster1_test = X_kmeans_test.loc[cluster_test == 0]

Cluster2_test = X_kmeans_test.loc[cluster_test == 1]

Cluster3_test = X_kmeans_test.loc[cluster_test == 2]

Cluster4_test = X_kmeans_test.loc[cluster_test == 3]
```

In [318…
```
Cluster1_test.describe()
```

Out[318]:

	Pclass	Fam	Age	Fare
count	179.000000	179.000000	179.000000	179.000000
mean	2.793296	0.402235	26.642458	13.014968
std	0.406077	0.691193	9.255007	10.661127
min	2.000000	0.000000	1.000000	0.000000
25%	3.000000	0.000000	21.500000	7.750000
50%	3.000000	0.000000	28.000000	8.662500
75%	3.000000	1.000000	29.000000	14.477100
max	3.000000	2.000000	61.000000	73.500000

In [319…
```
Cluster2_test.describe()
```

Out[319]:

	Pclass	Fam	Age	Fare
count	76.000000	76.000000	76.000000	76.000000
mean	1.289474	0.447368	39.677632	41.242270
std	0.484858	0.640723	13.868599	24.971153
min	1.000000	0.000000	16.000000	0.000000
25%	1.000000	0.000000	28.000000	26.000000
50%	1.000000	0.000000	36.750000	30.500000
75%	2.000000	1.000000	49.250000	60.044800
max	3.000000	3.000000	80.000000	91.079200

In [320…
```
Cluster3_test.describe()
```

Out[320]:

	Pclass	Fam	Age	Fare
count	26.000000	26.000000	26.000000	26.000000
mean	2.846154	5.307692	20.653846	35.033019
std	0.367946	2.035077	12.699425	14.440364
min	2.000000	3.000000	1.000000	18.750000
25%	3.000000	4.000000	8.250000	25.850025
50%	3.000000	5.000000	24.500000	31.275000
75%	3.000000	6.000000	28.000000	39.687500
max	3.000000	10.000000	40.000000	69.550000

In [321…
```
Cluster4_test.describe()
```

Out[321]:

	Pclass	Fam	Age	Fare
count	14.0	14.000000	14.000000	14.000000
mean	1.0	1.500000	29.351429	165.804164
std	0.0	1.224745	16.176330	46.532749
min	1.0	0.000000	0.920000	110.883300
25%	1.0	0.250000	19.000000	133.650000
50%	1.0	1.500000	32.000000	151.550000
75%	1.0	2.750000	39.750000	211.860425
max	1.0	3.000000	50.000000	247.520800

CART

Predictor Variables: Embarked, Age_c, Sex, Fam, Fare/Fare2, Pclass, Ticket

Build Model

```
In [322…  #save target variable as y_CART
y_dt = train_rebal[['Survived']]

#specify levels of target variable
y_dt_names = ["Did not survive", "Survived"]
```

```
In [323…  #combine all predictor variables, with dummies for categorical attributes
X_dt = pd.concat([sex_dummies,
                  embarked_dummies,
                  age_dummies,
                  pclass_dummies,
                  train_rebal['Fare'],
                  train_rebal['Fam']], axis=1)
X_dt
```

Out[323]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Age_Baby/Toddler	Age_Child	Age_Adult	Age_Elderly	Pclass_
65	0	1	1	0	0	0	0	1	0	
445	0	1	0	0	1	0	1	0	0	
659	0	1	1	0	0	0	0	1	0	
591	1	0	1	0	0	0	0	1	0	
690	0	1	0	0	1	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	.
275	1	0	0	0	1	0	0	1	0	
839	0	1	1	0	0	0	0	1	0	
577	1	0	0	0	1	0	0	1	0	
400	0	1	0	0	1	0	0	1	0	
383	1	0	0	0	1	0	0	1	0	

746 rows × 14 columns

```
In [323…
```

```
In [324…  #build model
cart01 = DecisionTreeClassifier(criterion = "gini").fit(X_dt, y_dt)
export_graphviz(cart01, out_file = "cart01.dot", class_names=y_dt_names)
```

```
In [325…  #save predictions
pred_CART = cart01.predict(X_dt)
```

Test Dataset Validation

```
In [326…  #test dataset prep
#combine all predictor variables, with dummies for categorical attributes
X_dt_test = pd.concat([sex_dummies_test,
                  embarked_dummies_test,
                  age_dummies_test,
                  pclass_dummies_test,
                  test['Fare'],
                  test['Fam']], axis=1)
X_dt_test
```

Out[326]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Age_Baby/Toddler	Age_Child	Age_Adult	Age_Elderly	Pclass_
725	0	1	0	0	1	0	0	1	0	
861	0	1	0	0	1	0	0	1	0	
528	0	1	0	0	1	0	0	1	0	
46	0	1	0	1	0	0	0	1	0	
627	1	0	0	0	1	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	.
360	0	1	0	0	1	0	0	1	0	
856	1	0	0	0	1	0	0	1	0	
199	1	0	0	0	1	0	0	1	0	
451	0	1	0	0	1	0	0	1	0	
417	1	0	0	0	1	0	0	1	0	

295 rows × 14 columns

```
In [327…  #use CART model to make predictions on test dataset
pred_CART_test = cart01.predict(X_dt_test)
```

Model Performance

```
In [328…  #Show contingency table for actual vs. predicted
ypredCART = pd.crosstab(test['Survived'], pred_CART_test,
                   rownames= ['Actual'],
                   colnames = ['Predicted'])
ypredCART['Total']=ypredCART.sum(axis=1)
ypredCART.loc['Total'] = ypredCART.sum()
ypredCART
```

Out[328]:

Predicted	0	1	Total
Actual			
0	151	25	176
1	42	77	119
Total	193	102	295

Compared to baseline model: (1) An All Negative Model would have an accuracy rate of 60% and (2) An All Positive Model would have an accuracy rate of 40%.

Accuracy: (153+74)/295 = 77%

Error Rate: 100% - 77% = 23%

Sensitivity: TP/Total Positive = 74/119 = 62%

Specificity: TN/Total Negative = 153/176 = 87%

Random Forest

Build Model

```
In [329…  # re-format target variable as a 1D-array
rfy_dt = np.ravel(y_dt)
```

```
In [330…  #run Random Forest algorithm
rf01 = RandomForestClassifier(n_estimators = 100, criterion = "gini").fit(X_dt, rfy_dt)
```

Test Dataset Validation

```
In [331…  #use Random Forest model to make predictions on test dataset
pred_rf_test = rf01.predict(X_dt_test)
```

Model Performance

```
In [332…  #Show contingency table for actual vs. predicted
ypredRF = pd.crosstab(test['Survived'], pred_rf_test,
                   rownames= ['Actual'],
                   colnames = ['Predicted'])
ypredRF['Total']=ypredRF.sum(axis=1)
ypredRF.loc['Total'] = ypredRF.sum()
ypredRF
```

Out[332]:

Predicted	0	1	Total
Actual			
0	147	29	176
1	43	76	119
Total	190	105	295

Accuracy: (148+75)/295 = 76%

Error Rate: 100% - 76% = 24%

Sensitivity: TP/Total Positive = 74/119 = 62%

Specificity: TN/Total Negative = 148/176 = 84%

Naive Bayes

Build Model

```
In [333…  ynb2 = train_rebal[['Survived']]
yyy = test[['Survived']]
ynb = np.ravel(ynb2)
xnb = pd.concat([sex_dummies,
                  embarked_dummies,
                  age_dummies,
                  pclass_dummies,
                  train_rebal['Fare'],
                  train_rebal['Fam']], axis=1)
xnbtest = pd.concat([sex_dummies_test,
                  embarked_dummies_test,
                  age_dummies_test,
                  pclass_dummies_test,
                  test['Fare'],
                  test['Fam']], axis=1)
nb_01 = MultinomialNB().fit(xnb, ynb)
ypred nb2 = nb_01.predict(xnbtest)
yprednb = pd.crosstab(test['Survived'], ypred nb2,
                   rownames= ['Actual'],
                   colnames = ['Predicted'])
yprednb['Total']=yprednb.sum(axis=1)
yprednb.loc['Total'] = yprednb.sum()
```

Model Performance

```
In [334…  ypred nb
```

Out[334]:

Predicted	0	1	Total
Actual			
0	159	17	176
1	43	76	119
Total	202	93	295

Accuracy: 160+77/295 = 80.3%

Error Rate: 100% - 77% = 19.7%

Sensitivity: TP/Total Positive = 77/119 = 65%

Specificity: TN/Total Negative = 160/176 = 91%

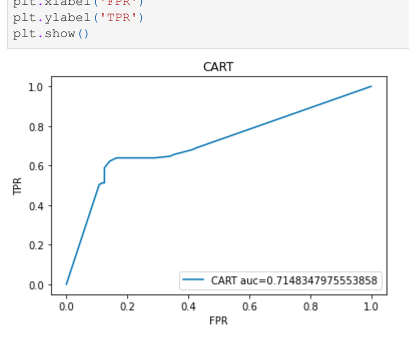
ROC

CART ROC

```
In [337…  print("Accuracy", metrics.accuracy_score(yyy, pred_CART_test))
```

```
Accuracy 0.7728813559322034
```

```
In [338…  y_pred_proba1 = DecisionTreeClassifier(criterion = "gini").fit(X_dt, y_dt).predict_proba(X_dt_test)[::,1]
fpr1, tpr1, _ = metrics.roc_curve(yyy,  y_pred_proba1)
auc1 = metrics.roc_auc_score(yyy, y_pred_proba1)
plt.plot(fpr1,tpr1,label="CART auc="+str(auc1))
plt.legend(loc=4)
plt.title('CART')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

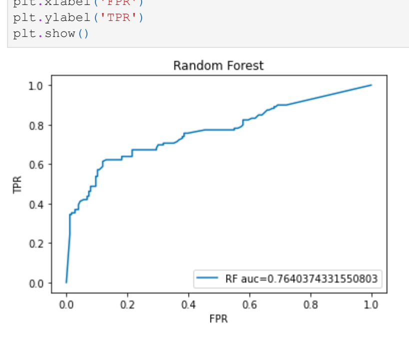


RF ROC

```
In [339…  print("Accuracy", metrics.accuracy_score(yyy, pred_rf_test))
```

```
Accuracy 0.7559322033898305
```

```
In [340…  y_pred_proba2 = RandomForestClassifier(n_estimators = 100, criterion = "gini").fit(X_dt, rfy_dt).predict_proba(
fpr2, tpr2, _ = metrics.roc_curve(yyy,  y_pred_proba2)
auc2 = metrics.roc_auc_score(yyy, y_pred_proba2)
plt.plot(fpr2,tpr2,label="RF auc="+str(auc2))
plt.legend(loc=4)
plt.title('Random Forest')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

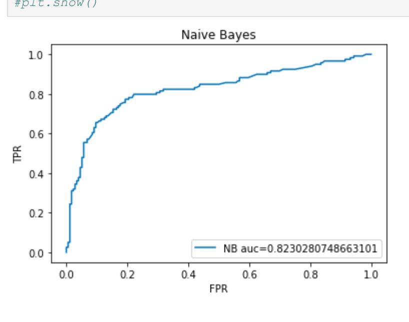


NB ROC

```
In [341…  print("Accuracy", metrics.accuracy_score(yyy, ypred nb2))
```

```
Accuracy 0.7966101694915254
```

```
In [342…  y_pred_proba = MultinomialNB().fit(xnb, ynb).predict_proba(xnbtest)[::,1]
fpr, tpr, _ = metrics.roc_curve(yyy,  y_pred_proba)
auc = metrics.roc_auc_score(yyy, y_pred_proba)
plt.plot(fpr,tpr,label="NB auc="+str(auc))
plt.legend(loc=4)
plt.title('Naive Bayes')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()

#fpr, tpr, _ = metrics.roc_curve(yyy,  y_pred_proba)
#plt.plot(fpr, tpr)
#plt.show()
```

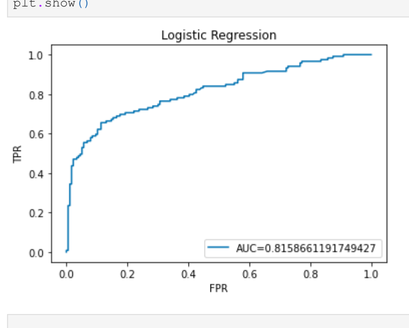


Logistic Regression ROC

```
In [344…  print("Accuracy", metrics.accuracy_score(yyy, ypred_logis))
```

```
Accuracy 0.7864406779661017
```

```
In [343…  #create ROC curve for Logistic regression model
lg_pred_prob = logreg01.predict(X_logi_d_test)
fpr9, tpr9, _ = metrics.roc_curve(yyy,  lg_pred_prob)
auc9 = metrics.roc_auc_score(yyy, lg_pred_prob)
plt.plot(fpr9,tpr9,label="AUC="+str(auc9))
plt.legend(loc=4)
plt.title('Logistic Regression')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```




```
In [343…
```