MODULE 4: TITANIC: MACHINE LEARNING THROUGH DISASTER

Claire Markey, Julia Granito, Manny Hurtado, and Steve Desilets

MSDS 422: Practical Machine Learning

April 23rd, 2023

Introduction

The sinking of the Titanic remains an infamous tragedy that continues to be studied today. Survival data from this event has led to significant changes in maritime safety and ship design and raised ethical questions that have challenged societal norms (Elinder and Erixson, 2012). For this assignment, an exploratory data analysis (EDA) and modeling techniques were employed to understand how factors (gender, age, socioeconomic status, cabin, ticket class, and originating port) influenced survival.

Method

Kaggle data about Titanic passengers were downloaded and analyzed using Jupyter Notebooks (Li and Cukierski, 2016). An EDA explored feature characteristics (associations, distributions, missingness, and outliers). Classification methods such as Logistic Regression, Linear Discrimination Analysis (LDA), Quadratic Discrimination Analysis (QDA), and K-Nearest Neighbor (KNN) were also used. Model assumptions and metrics (confusion matrices, ROC and Precision-Recall plots) were also examined.

Results and Insights

First, descriptive statistics about the numeric variables were examined by constructing histograms and calculating summary statistics. With regard to survival, out of 891 data points available, 61.6% of passengers died. Next, we cleaned the data and visually inspected the distributions via bar charts. While the numeric variables' distributions were skewed right, the values were realistic, so no values were removed. An examination of missing values revealed that three variables (cabin, age, embarked) contained missingness. Missing values for 'age' and 'embarked' were replaced with the median value and mode values, respectively. Two new features were created to address missing 'cabin' values: a dichotomous 'cabin' indicator where missing values were computed as 0, and a categorical variable of the first letters of Cabin values (to reflect the cabin's deck), and missing values were updated to 'None'. A dichotomous variable (child indicator) was created by utilizing age to classify passengers under 18 as children (set to '1').

Pearson and Point Biserial correlations and box plots were used to understand each variable's relationship with passenger survival. The boxplots suggested that cabin indicator, child indicator, passenger class, and sex were strong predictors of survival, but the correlation analyses suggested there were only weak associations between the predictors and survival (r values ranging from -.03 to .26).

A multiple logistic regression was employed to examine the association between survival and the child indicator variable (adult reference category), ticket class, location where they embarked (Queenstown ref.), and sex (female ref.). The overall model was statistically significant when compared to the null model, explained 32% of the variance (Nagelkerke R^2) and correctly predicted 78.0% of cases. While none of the predictors were strongly, significantly associated with each other, an inspection of variance inflation factor scores indicated that some multicollinearity existed among predictors. The remainder of the logistic regression assumptions were met. ROC and Precision-Recall curves were examined, with areas under the curve (AUC) of 0.85 and 0.82, respectively. The classification score of the model was 0.78, meaning that 78% of passenger survival outcomes were correctly identified by the model. This model also had high specificity (0.81) and precision (0.71). Survival classification accuracy on the Kaggle test data was 0.748, indicating that 74.8% of passengers survival outcomes in the test dataset were correctly predicted by the logistic model. Passenger class, being male, and embarking in Queenstown were significantly, negatively associated with survival. Those with a lower class status were about 63% less likely to survive. Relative to females, males were 93% less likely to have survived. Compared to those that embarked in Queenstown, those that embarked at Cherbourg were 39% less likely to have survived. Being a child was significantly, positively associated with survival. Children were 2.7 times more likely to have survived compared to adults.

Then, LDA and QDA models were constructed using the same five predictors as the logistic model and performance metrics were examined. The predictors were scaled prior to fitting the models. The coefficients and means of the LDA model indicate that the passenger sex contributes the most to discriminating between the two classes (survival and no survival), while passenger ticket class contributes the second highest. The other features had smaller coefficients and differences between means for each class of the outcome, indicating that they had less impact on the discriminant function. Precision-Recall and ROC curves were also examined, with AUC scores of 0.82 and 0.81, respectively. The classification score of the model was 0.79 meaning that the model correctly identified 79% of passenger survival outcomes. The LDA model correctly classified 76.8% of passengers survival outcomes in the Kaggle test data. The means for each feature in the QDA model within each class indicate that passenger sex and passenger ticket class contribute the most to class separation (survival vs dying). This aligns with the LDA and logistic regression models. ROC and Precision-Recall curves were examined, with AUC scores of 0.81 and 0.80, respectively. The classification score of the model was 0.77, meaning that 77% of passenger survival outcomes in the training set were correctly identified by the model. This model also had the highest sensitivity (0.80) and recall (0.80) scores. For the Kaggle test data, this model correctly classified 75.8% of passengers survival outcomes.

Next, we created a KNN model to predict survival. First, min-max scaling was applied to the fare and passenger class variables. We used these two scaled predictor variables, as well as three other predictors (sex, child indicator, and cabin data indicator) to create a KNN model. The training dataset was split into training and validation datasets to find the optimal value for k. ROC and Precision-Recall curves corresponding to this model's performance on the validation dataset were constructed. Then, we leveraged the model's performance on the validation dataset to identify the cutoff boundary for predictions that maximized the percent of passengers whose survival was correctly predicted in the validation dataset. At this point, the recall, specificity, precision, and false positive rate, and accuracy for the validation data were 0.82, 0.72, 0.67, 0.28, and 0.78, respectively. We applied this KNN Model, using the identified optimal boundary, to the Kaggle test dataset and correctly predicted survival for 72% of passengers.

This analysis provided insights about factors that may have contributed to passenger survival on the Titanic. The classification models support the notion that men or passengers with lower ticket classes were less likely to survive. Results from the LDA model indicate that the linear decision boundary assumed by the model was appropriate for the data and that LDA may represent the best classification model to predict survival if accuracy is the goal. Accuracy scores by the QDA model were high as well, indicating that the quadratic boundary assumed by the model performed well at classifying the data.

References

Elinder, Mikael, and Oscar Erixson. 2012. "Gender, Social Norms, and Survival in Maritime Disasters." *Proceedings of the National Academy of Sciences* 109, no. 33: 13220–24. https://doi.org/10.1073/pnas.1207156109.

Li, Jessica and Will Cukierski. 2012. "Titanic - Machine Learning from Disaster." *Kaggle*. https://www.kaggle.com/competitions/titanic/overview

Appendix 1 - Python Code and Outputs

Data Preparation

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Import Data

In [2]:
```
import pandas as pd
titanic_training_data = pd.read_csv('train.csv')

# show first five rows of the data
titanic_training_data.head(100)
# show number of columns and rows
titanic_training_data.shape
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN
...	...	...	...	...	...	...	...	...	...	...	...
95	96	0	3	Shorney, Mr. Charles Joseph	male	NaN	0	0	374910	8.0500	NaN
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5
97	98	1	1	Greenfield, Mr. William Bertram	male	23.0	0	1	PC 17759	63.3583	D10 D12
98	99	1	2	Doling, Mrs. John T (Ada Julia Bone)	female	34.0	0	1	231919	23.0000	NaN
99	100	0	2	Kantor, Mr. Sinai	male	34.0	1	0	244367	26.0000	NaN

100 rows × 12 columns

Out[2]:

```
(891, 12)
```

Variable Key Guide:

Variable name	Variable label	Variable value and value label	Variable type
PassengerID	Passenger ID		Numerical
Survived	Did the passenger survive?	0 = No, 1 = Yes	Indicator, dichotomous
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd	Categorical, or ordinal

Variable name	Variable label	Variable value and value label	Variable type
Name	Passenger name		String
Sex	Passenger's sex	Male, Female	Categorical or indicator
Age	Passenger's age		Numerical
Sibsp	# of siblings / spouses aboard the Titanic		Numerical
Parch	# of parents / children aboard the Titanic		Nmerical
Ticket	Ticket number		String
Fare	Passenger fare		Numerical, continuous
Cabin	Cabin number		Categorical
Embarked	Port of embarkation	C = Cherbourg, Q = Queenstown, S = Southampton	Categorical

Exploratory Data Analysis

First, we can categorize each variable as either an indicator variable, multi-category categorical variable, or numeric variable. Then we can proceed in our exploratory data analysis by constructing the appropriate visualization for each type of variable.

In [3]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

numeric_variables = ['Age', 'Fare', 'SibSp', 'Parch']

indicator_variables = ['Survived']

categorical_variables = ['Pclass', 'Embarked', 'Cabin', 'Sex']

# Numeric Variable Visualizations
titanic_training_data[numeric_variables].describe()

titanic_training_data[numeric_variables].hist(edgecolor = 'black',
                                              bins = 15, figsize = (15, 10),
                                              layout = (2, 2), grid = False)


# Indicator Variable Visualizations
sns.catplot(x = 'Survived', kind = 'count', data = titanic_training_data)


# Categorical Variable Visualizations
fig, ax = plt.subplots(2, 2, figsize = (15, 15))
for var, subplot in zip(categorical_variables, ax.flatten()):
    titanic_training_data[var].value_counts().plot(kind = 'bar', ax = subplot, title =
```

```
fig.tight_layout()
```

Out[3]:

	Age	Fare	SibSp	Parch
count	714.000000	891.000000	891.000000	891.000000
mean	29.699118	32.204208	0.523008	0.381594
std	14.526497	49.693429	1.102743	0.806057
min	0.420000	0.000000	0.000000	0.000000
25%	20.125000	7.910400	0.000000	0.000000
50%	28.000000	14.454200	0.000000	0.000000
75%	38.000000	31.000000	1.000000	0.000000
max	80.000000	512.329200	8.000000	6.000000

Out[3]:

```
array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Fare'}>],
       [<Axes: title={'center': 'SibSp'}>,
        <Axes: title={'center': 'Parch'}>]], dtype=object)
```

Out[3]:

```
<seaborn.axisgrid.FacetGrid at 0x1a63350f100>
```

Out[3]:

```
<Axes: title={'center': 'Pclass'}>
```

Out[3]:

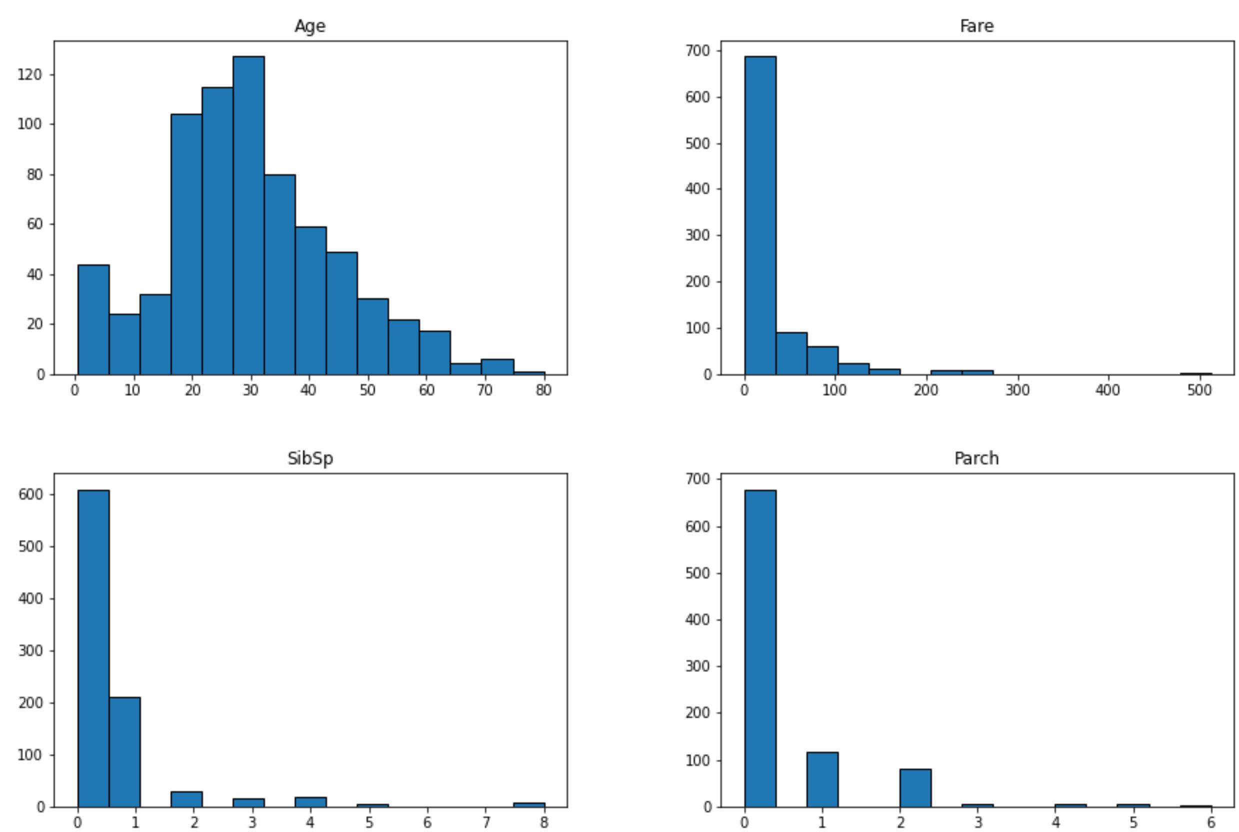
```
<Axes: title={'center': 'Embarked'}>
```

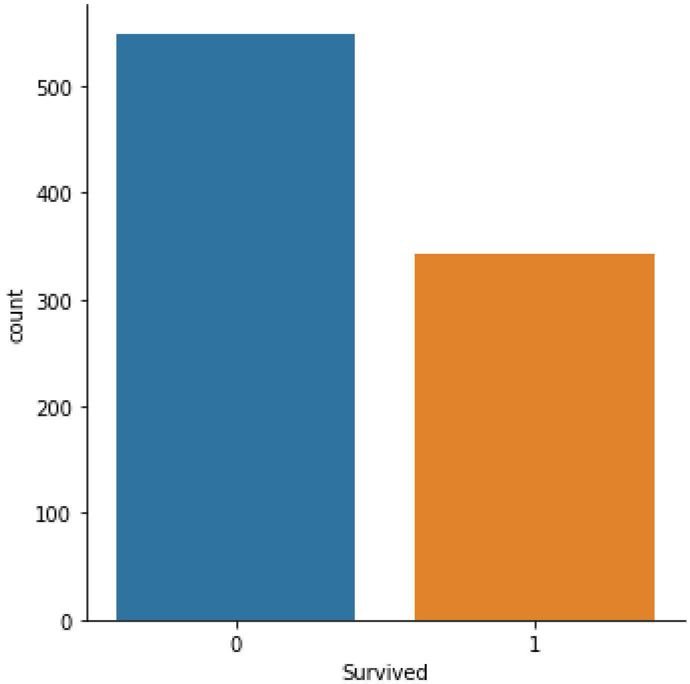
Out[3]:

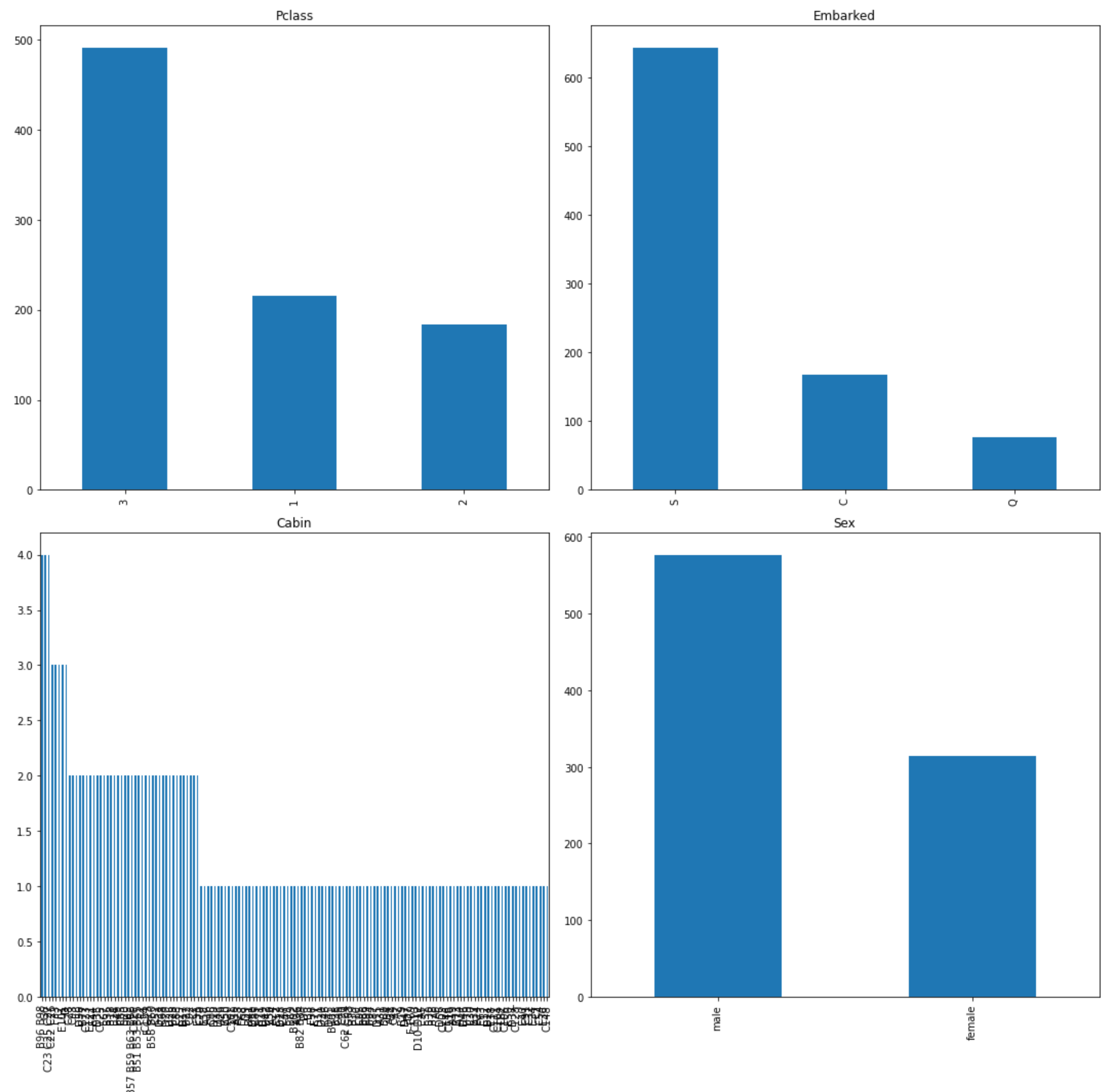
```
<Axes: title={'center': 'Cabin'}>
```

Out[3]:

```
<Axes: title={'center': 'Sex'}>
```




count
500
400
300
200
100
0
0
1
Survived


Pclass
3
1
2
Embarked
S
C
Q
Cabin
Sex
male
female

Investigation of Missing Data and Outliers

In [4]:
```
# find null counts, percentage of null values, and column type
null_count = titanic_training_data.isnull().sum()
null_percentage = titanic_training_data.isnull().sum() * 100 / len(titanic_training_da
column_type = titanic_training_data.dtypes

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[4]:

	Missing Count	Percentage Missing	Column Type
Cabin	687	77.104377	object
Age	177	19.865320	float64
Embarked	2	0.224467	object

Let's address the missing data appropriately in a new dataframe that we'll name titanic_training_data_cleaned.

In [5]:
```
# Create a new dataframe called titanic_training_data_cleaned so we don't modify the c
titanic_training_data_cleaned = titanic_training_data.copy(deep=True)

# change Null values to the most common value (S) for Embarked
titanic_training_data_cleaned['Embarked'].fillna('S', inplace=True)

# fill Nulls for Age with median value
titanic_training_data_cleaned['Age'].fillna(titanic_training_data_cleaned['Age'].media

# Create new cabin-related variables that will be more useful and cleaner than the ori
titanic_training_data_cleaned['Cabin_Data_Indicator'] = titanic_training_data_cleaned[

titanic_training_data_cleaned['First_Cabin_Deck'] = np.where(titanic_training_data_cle
                                                             titanic_training_data_cle
                                                             'None')

# Create a new variable indicating whether a passenger is a child
titanic_training_data_cleaned['Child_Indicator'] = titanic_training_data_cleaned['Age'
titanic_training_data_cleaned['Child_Indicator'] = titanic_training_data_cleaned['Chil


# Theoretically we could create a new variable for cabin number here if we're interest
# Time permitting, maybe I will circle back to work on that some more, but for now I'n
# important pieces of this assignment - especially since intuitively I don't think cab
# difference for survival rates

# There also could be value in creating a variable about marriage status based on whet
# But we can circle back to that if we have time I think

# Drop the original Cabin variable since it has so many null values and since some pas
# making the original variable difficult to work with
titanic_training_data_cleaned.drop(['Cabin'],axis=1,inplace=True)
```

In [6]:
```
titanic_training_data_cleaned['Survived'].value_counts()
```

Out[6]:
```
0    549
1    342
Name: Survived, dtype: int64
```

Check the distributions of the variables in the newly cleaned dataframe. Also, check for missing values in this new dataframe.

In [7]:
```
# show first five rows of the data
titanic_training_data_cleaned.head(20)
# show number of columns and rows
titanic_training_data_cleaned.shape
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	
5	6	0	3	Moran, Mr. James	male	28.0	0	0	330877	8.4583	
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	
12	13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.8542	
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.1250	
17	18	1	2	Williams, Mr. Charles Eugene	male	28.0	0	0	244373	13.0000	
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.0000	
19	20	1	3	Masselmani, Mrs. Fatima	female	28.0	0	0	2649	7.2250	

(891, 14)

Variable Key Guide:

Variable name	Variable label	Variable value and value label	Variable type
PassengerID	Passenger ID		Numerical
Survived	Did the passenger survive?	0 = No, 1 = Yes	Indicator, dichotomous
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd	Categorical
Name	Passenger name		String
Sex	Passenger's sex	Male, Female	Categorical
Age	Passenger's age		Numerical
Sibsp	# of siblings / spouses aboard the Titanic		Numerical
Parch	# of parents / children aboard the Titanic		Nmerical
Ticket	Ticket number		String
Fare	Passenger fare		Numerical, continuous
Embarked	Port of embarkation	C = Cherbourg, Q = Queenstown, S = Southampton	Categorical
Cabin Data Indicator	Cabin number		Indicator, dichotomuous
First cabin deck	In first cabin class		Categorical

Variable name	Variable label	Variable value and value label	Variable type
Child_indicator	Was a child, under 18 years old		Indicator, dichotomuous

In [8]:
```
# find null counts, percentage of null values, and column type
null_count = titanic_training_data_cleaned.isnull().sum()
null_percentage = titanic_training_data_cleaned.isnull().sum() * 100 / len(titanic_tra
column_type = titanic_training_data_cleaned.dtypes

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[8]:

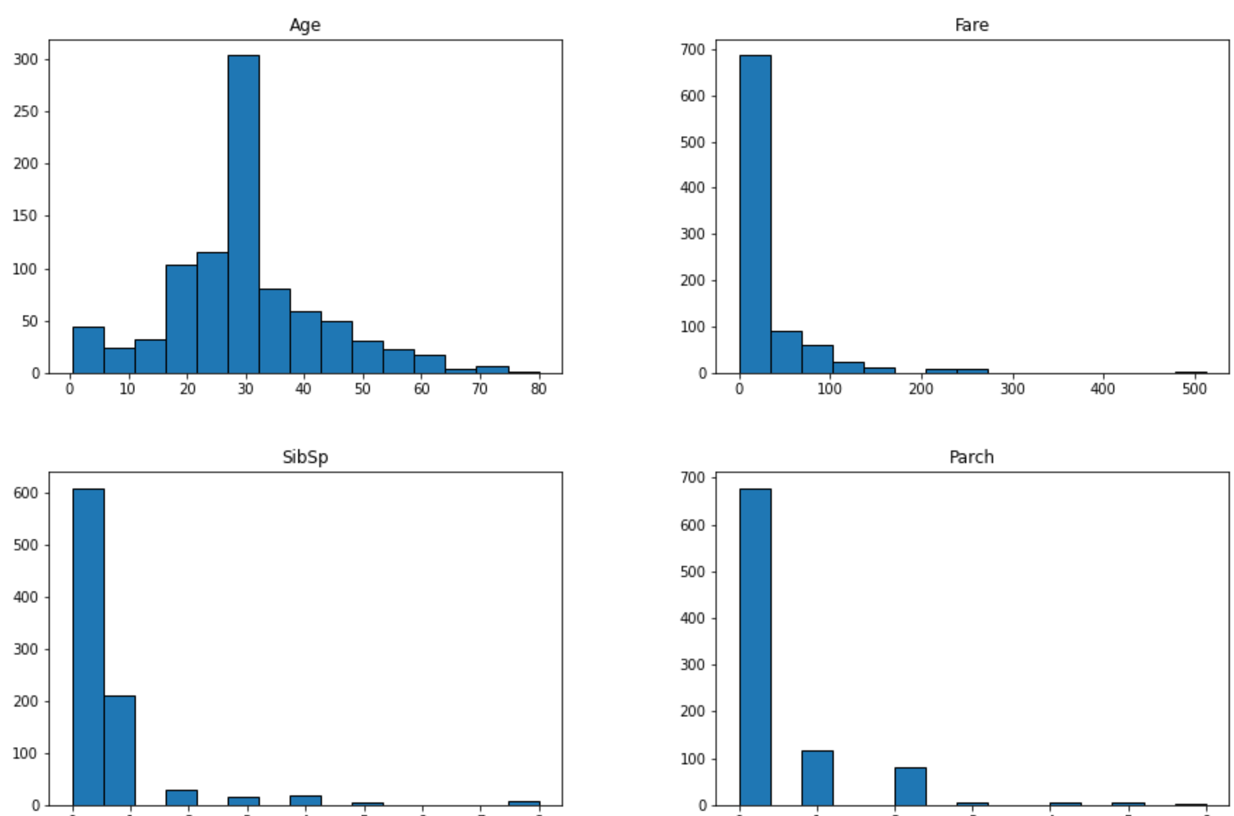
Missing Count	Percentage Missing	Column Type

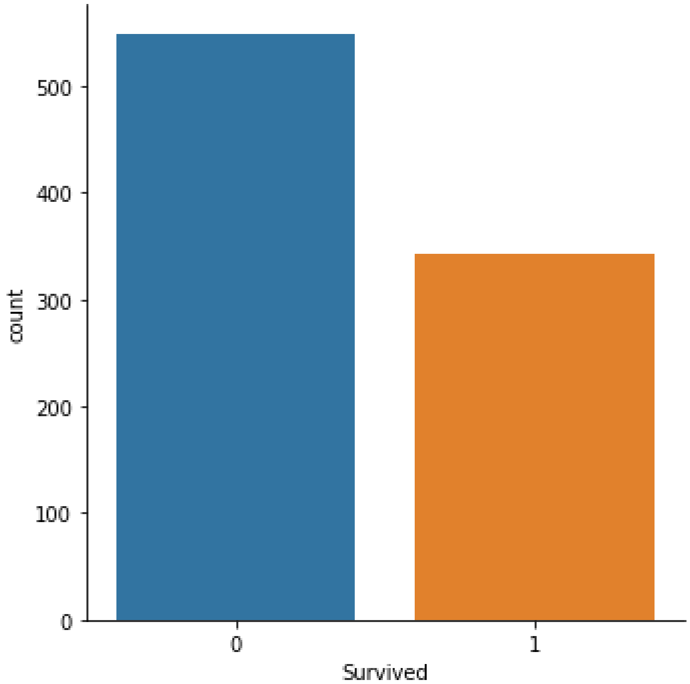
In [9]:
```
# Update our definitions of the indicator, numeric, and categorical variables to refle
numeric_variables = ['Age', 'Fare', 'SibSp', 'Parch']

indicator_variables = ['Survived', 'Cabin_Data_Indicator', 'Child_Indicator']

categorical_variables = ['Pclass', 'Embarked', 'First_Cabin_Deck', 'Sex']

# Numeric Variable Visualizations
titanic_training_data_cleaned[numeric_variables].describe()

titanic_training_data_cleaned[numeric_variables].hist(edgecolor = 'black',
                                             bins = 15, figsize = (15, 10),
                                             layout = (2, 2), grid = False)

# Indicator Variable Visualizations

for var, subplot in zip(indicator_variables, ax.flatten()):
    sns.catplot(x = var, kind = 'count', data = titanic_training_data_cleaned)

fig.tight_layout()


# Categorical Variable Visualizations
fig, ax = plt.subplots(2, 2, figsize = (15, 15))
for var, subplot in zip(categorical_variables, ax.flatten()):
    titanic_training_data_cleaned[var].value_counts().plot(kind = 'bar', ax = subplot,

fig.tight_layout()
```

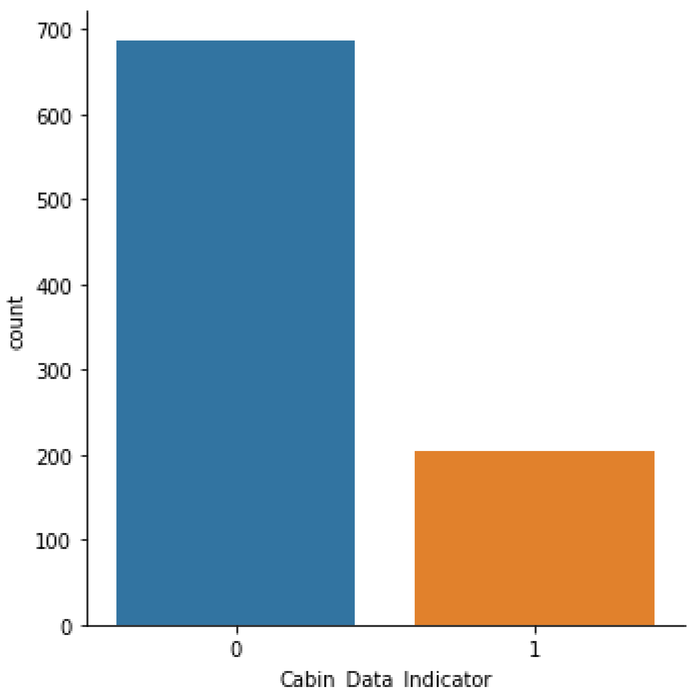
Out[9]:

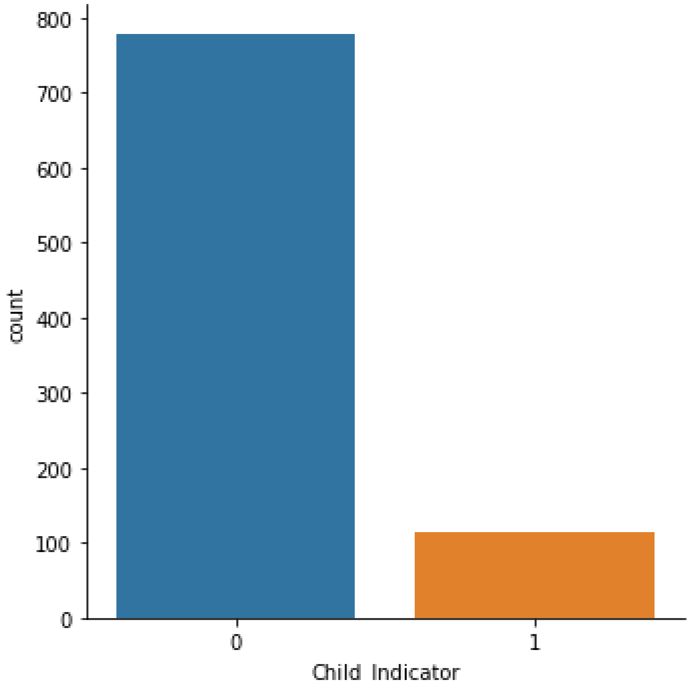
	Age	Fare	SibSp	Parch
count	891.000000	891.000000	891.000000	891.000000
mean	29.361582	32.204208	0.523008	0.381594
std	13.019697	49.693429	1.102743	0.806057
min	0.420000	0.000000	0.000000	0.000000
25%	22.000000	7.910400	0.000000	0.000000
50%	28.000000	14.454200	0.000000	0.000000
75%	35.000000	31.000000	1.000000	0.000000
max	80.000000	512.329200	8.000000	6.000000

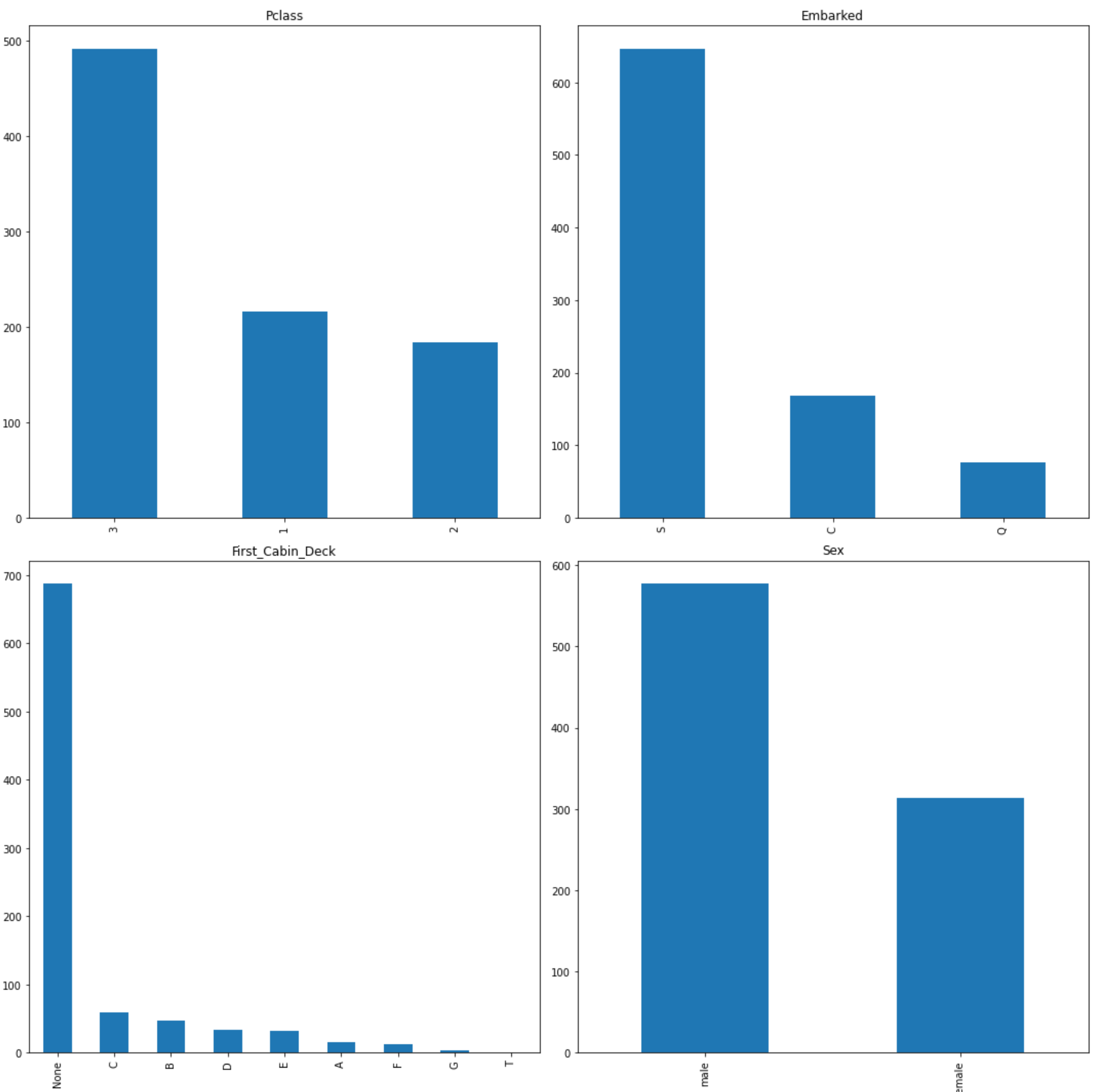
```
Out[9]: array([[<Axes: title={'center': 'Age'}>,
                <Axes: title={'center': 'Fare'}>],
               [<Axes: title={'center': 'SibSp'}>,
                <Axes: title={'center': 'Parch'}>]], dtype=object)
Out[9]: <seaborn.axisgrid.FacetGrid at 0x1a6130712e0>
Out[9]: <seaborn.axisgrid.FacetGrid at 0x1a656fccb50>
Out[9]: <seaborn.axisgrid.FacetGrid at 0x1a657457cd0>
Out[9]: <Axes: title={'center': 'Pclass'}>
Out[9]: <Axes: title={'center': 'Embarked'}>
Out[9]: <Axes: title={'center': 'First_Cabin_Deck'}>
Out[9]: <Axes: title={'center': 'Sex'}>
```




count
0
100
200
300
400
500
0
1
Survived


count
0
100
200
300
400
500
600
700
0
1
Cabin_Data_Indicator


count
Child_Indicator
0
1


Pclass
3
1
2
Embarked
S
C
Q
First_Cabin_Deck
None
C
B
D
E
A
F
G
T
Sex
male
female

Examination of the Relationship between the Dependent Variable and Potential Predictors

Let's create some visualizations to examine the relationship between potential predictors and our dependent variable.

In [10]:
```
# Numeric Variable Visualizations

fig, ax = plt.subplots(2, 2, figsize=(11, 11))

for var, subplot in zip(numeric_variables, ax.flatten()):
         sns.boxplot(x = 'Survived', y = var, data=titanic_training_data_cleaned, ax=s

fig.tight_layout()

# Indicator Variable Visualizations

for var, subplot in zip(indicator_variables, ax.flatten()):
    g = sns.catplot(
        data = titanic_training_data_cleaned,
        x = var,
        y = "Survived",
        kind = "bar",
        height = 6,
        aspect = 0.6,
        ci = None)

    g.set_axis_labels(var, "Proportion That Survived")
    g.set(ylim = (0,1))

fig.tight_layout()



# Categorical Variable Visualizations
for var, subplot in zip(categorical_variables, ax.flatten()):
    g = sns.catplot(
        data = titanic_training_data_cleaned,
        x = var,
        y = "Survived",
        kind = "bar",
        height = 6,
        aspect = 0.6,
        ci = None)

    g.set_axis_labels(var, "Proportion That Survived")
    g.set(ylim = (0,1))

fig.tight_layout()
```

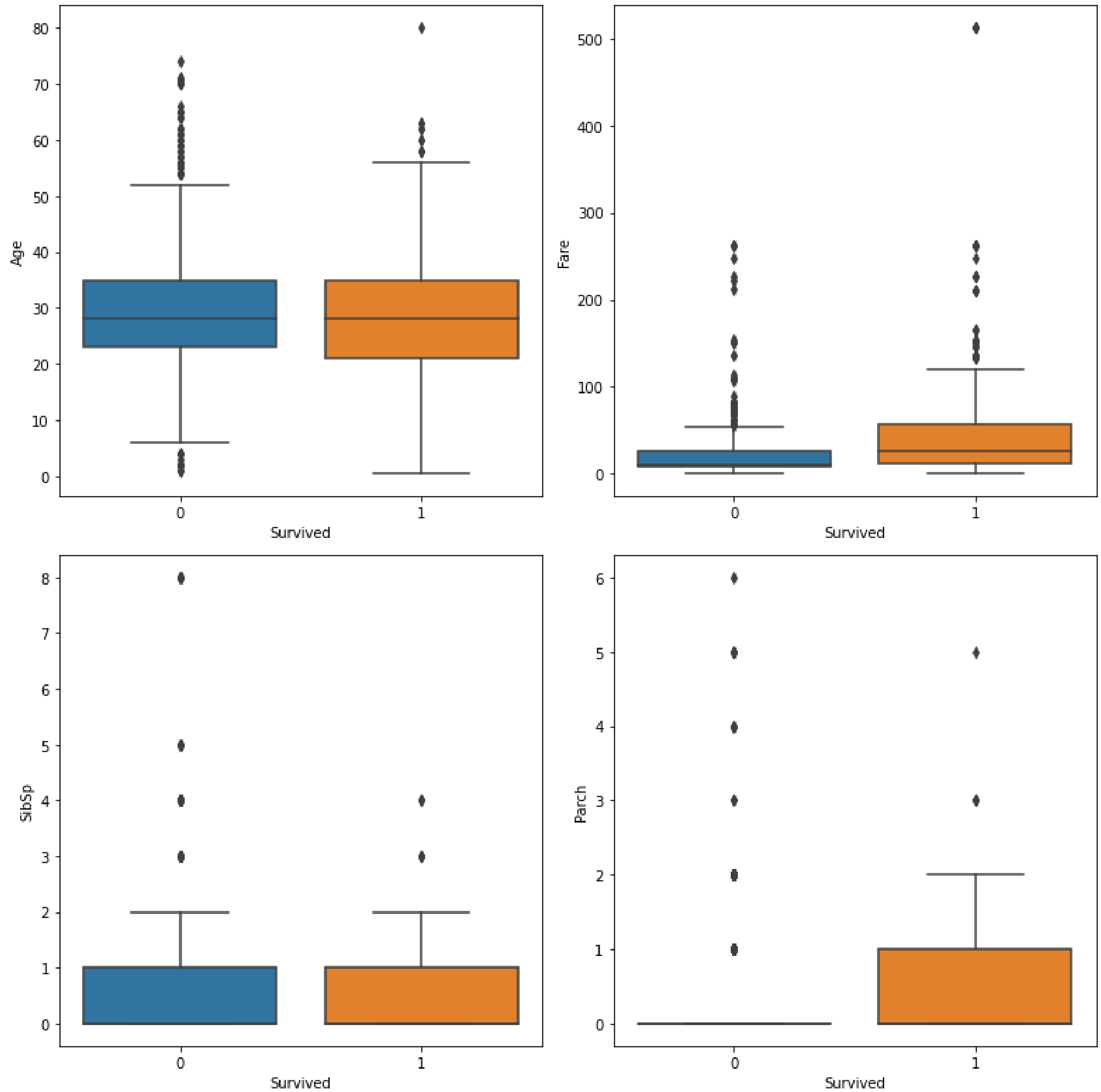
Out[10]:
```
<Axes: xlabel='Survived', ylabel='Age'>
```

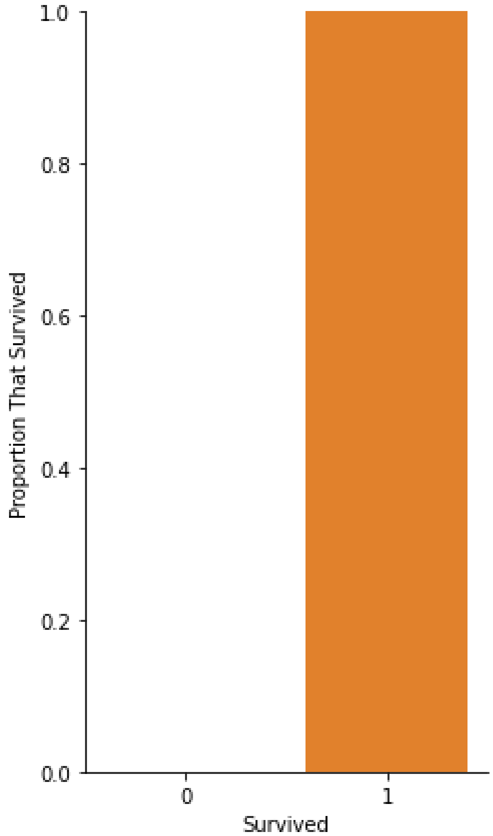
Out[10]:
```
<Axes: xlabel='Survived', ylabel='Fare'>
```

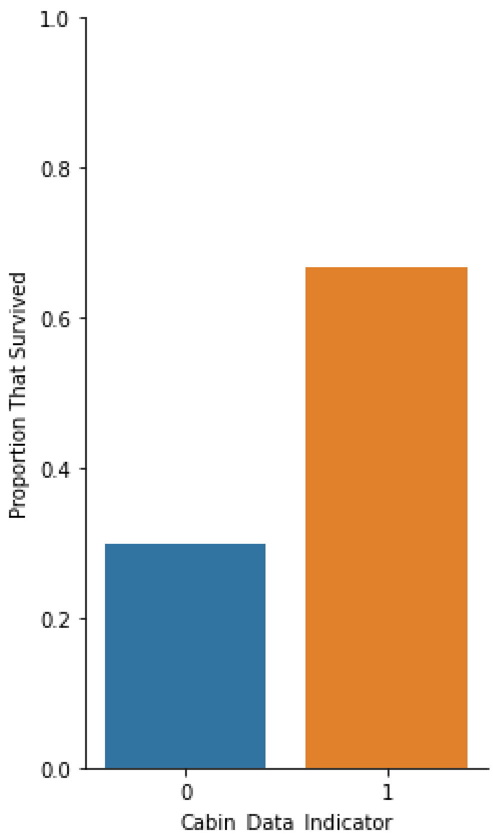
Out[10]:
```
<Axes: xlabel='Survived', ylabel='SibSp'>
```

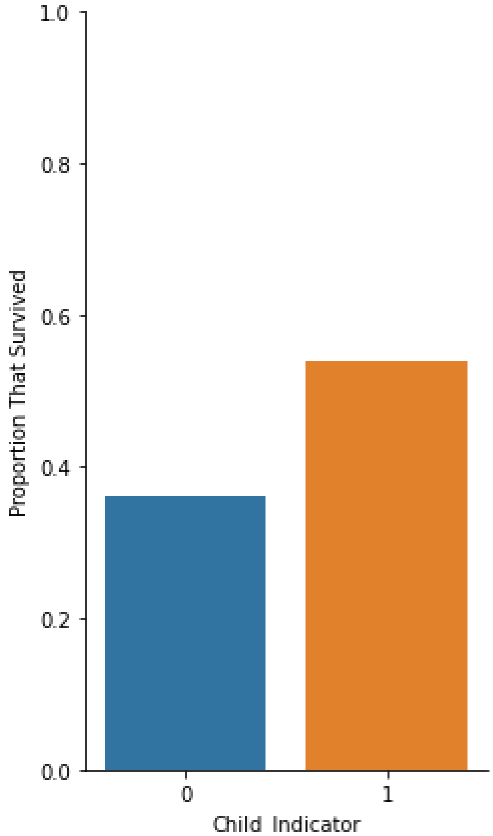
Out[10]:
```
<Axes: xlabel='Survived', ylabel='Parch'>
```

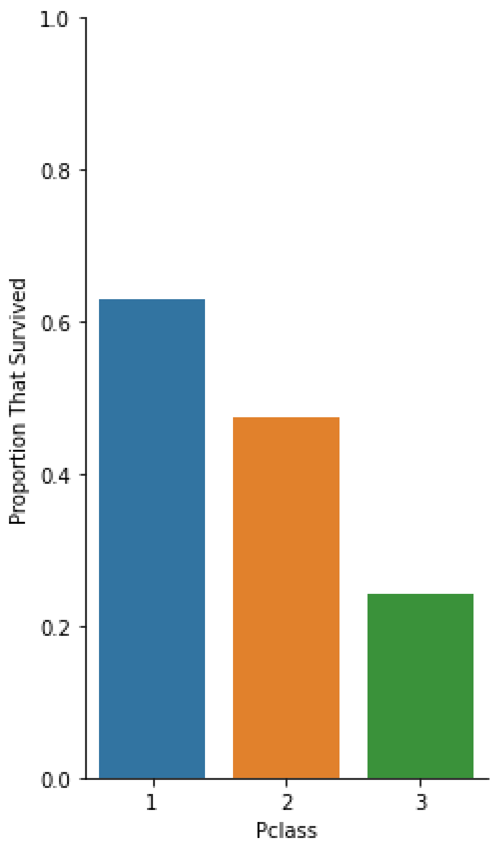
```
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a654098df0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a654098df0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a6566658e0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a6566658e0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656a6bc70>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656a6bc70>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a654098df0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a654098df0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656b1e940>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656b1e940>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656e0fbe0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656e0fbe0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656a1f5b0>
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1a656a1f5b0>
```

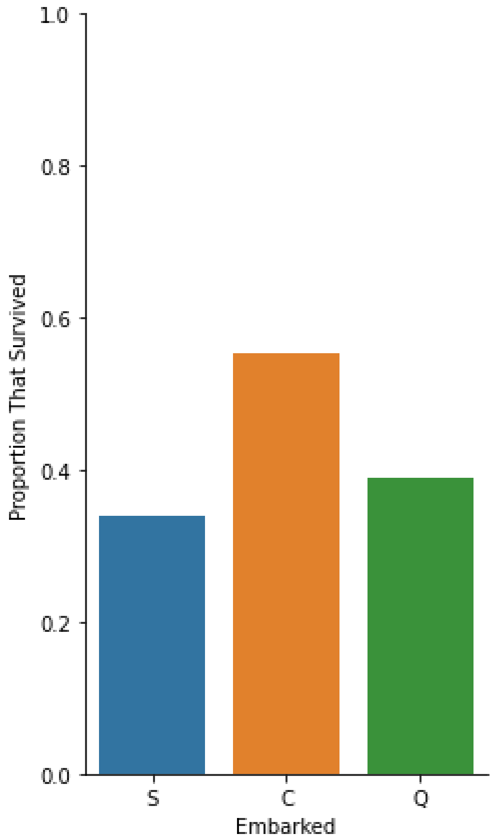

Age
0
10
20
30
40
50
60
70
80
Survived
0
1
Fare
0
100
200
300
400
500
Survived
0
1
SibSp
0
1
2
3
4
5
6
7
8
Survived
0
1
Parch
0
1
2
3
4
5
6
Survived
0
1

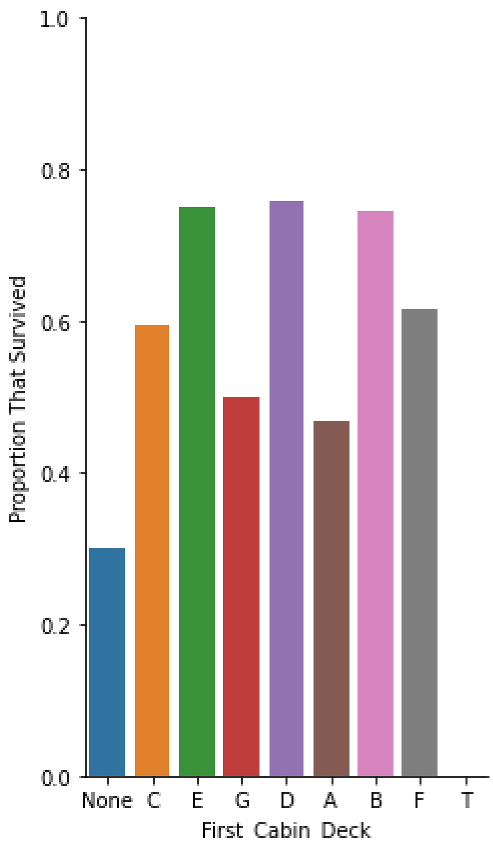

1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
0
1
Survived

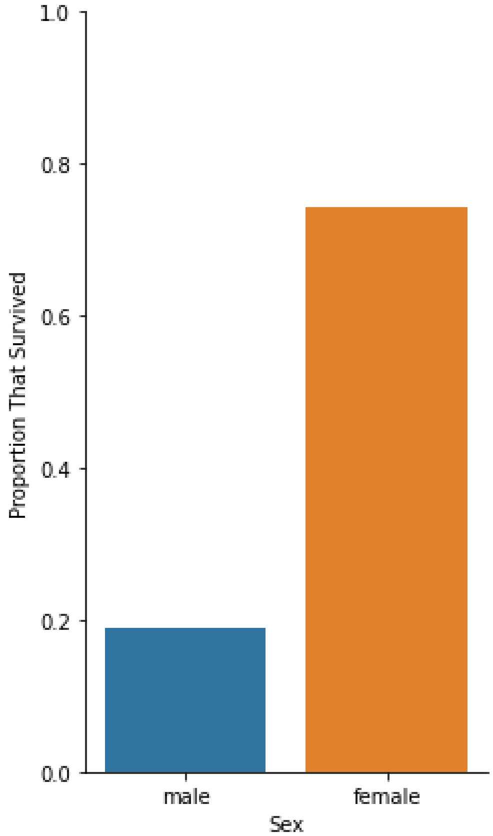

1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
0
1
Cabin_Data_Indicator


1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
0
1
Child_Indicator


1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
1
2
3
Pclass


1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
S
C
Q
Embarked


1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
None
C
E
G
D
A
B
F
T
First_Cabin_Deck

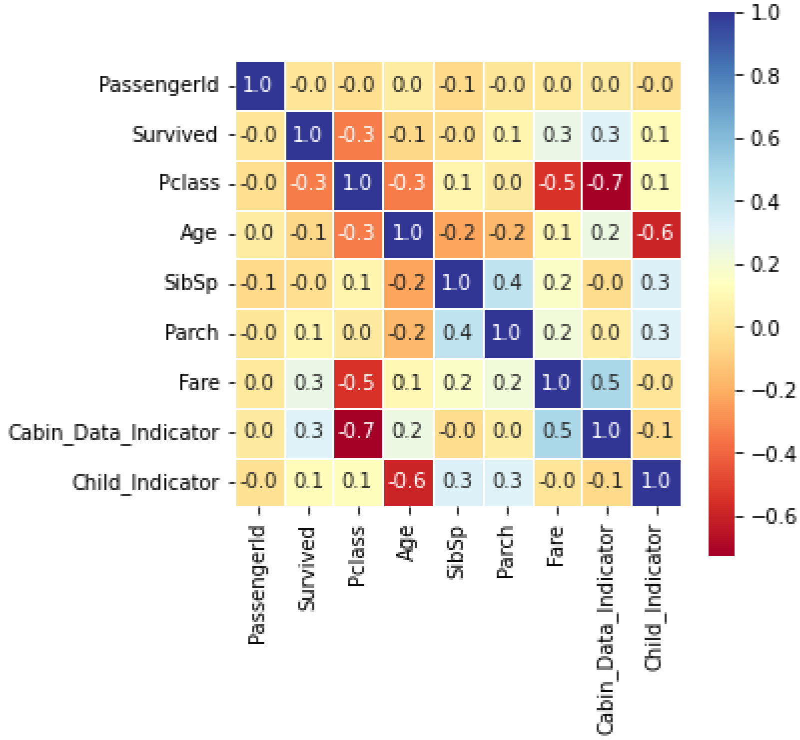


Logistic Regression

First let's examine the relationship between variables

In [11]:
```
titanic_correlations = titanic_training_data_cleaned
correlations_training = titanic_correlations.corr()

f, ax = plt.subplots(figsize = (5,5))
sns.heatmap(correlations_training, vmax = 1, square = True, annot = True,
            cmap = 'RdYlBu', linewidths = 0.5, fmt=".1f")
```

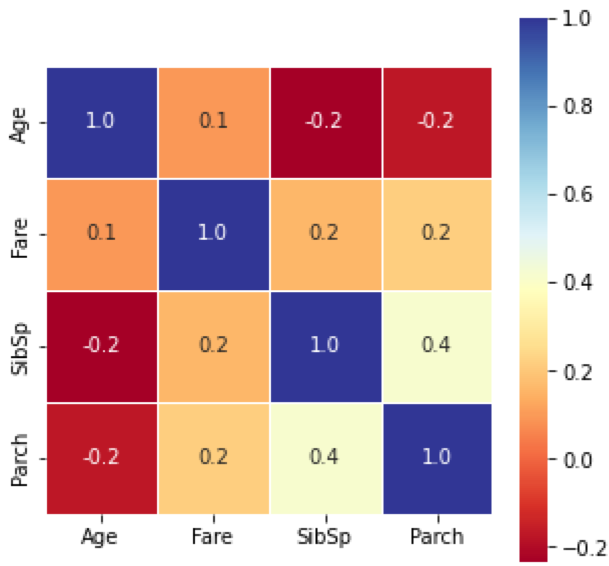
Out[11]: <Axes: >



In [12]:

```
titanic_correlations = titanic_training_data_cleaned[numeric_variables]
correlations_training = titanic_correlations.corr()

f, ax = plt.subplots(figsize = (5,5))
sns.heatmap(correlations_training, vmax = 1, square = True, annot = True,
            cmap = 'RdYlBu', linewidths = 0.5, fmt=".1f")
```

Out[12]: <Axes: >



The Pearson correlations suggest that Pclass is strongly associated with Cabin_data_indicator. The following were moderately associated: 1) age and child_indicator, 2) fare and Pclass, and 3) fare and cabin_data_indicator.

Note: Moderately associated: .5 - .7, strongly associated: >.7

No variables were strongly or moderately associated with Survived. In fact, the Pearson correlation analysis suggested there were weak associations, and there may be no issues with multicollinearity among these variables.

Point Biserial correlations between Survived and continuous data

In [13]:
```
from scipy import stats

#Point biserial correlation coefficient and p value for survived and age:
corr1 = stats.pointbiserialr(titanic_training_data_cleaned.Survived, titanic_training_
print("Point biserial correlation coefficient and p value for sale price and age:")
corr1
#Point biserial correlation coefficient and p value for survived and fare:
corr2 = stats.pointbiserialr(titanic_training_data_cleaned.Survived, titanic_training_
print("Point biserial correlation coefficient and p value for sale price and fare:")
corr2
#Point biserial correlation coefficient and p value for survived and Sibsp:
corr3 = stats.pointbiserialr(titanic_training_data_cleaned.Survived, titanic_training_
print("Point biserial correlation coefficient and p value for sale price and Sibsp:")
corr3
#Point biserial correlation coefficient and p value for survived and Parch:
corr4 = stats.pointbiserialr(titanic_training_data_cleaned.Survived, titanic_training_
print("Point biserial correlation coefficient and p value for sale price and Parch:")
corr4
```

```
Point biserial correlation coefficient and p value for sale price and age:
```
Out[13]:
```
SignificanceResult(statistic=-0.06491041993052588, pvalue=0.05276068847585567)
```
```
Point biserial correlation coefficient and p value for sale price and fare:
```
Out[13]:
```
SignificanceResult(statistic=0.25730652238496243, pvalue=6.1201893419246185e-15)
```
```
Point biserial correlation coefficient and p value for sale price and Sibsp:
```
Out[13]:
```
SignificanceResult(statistic=-0.03532249888573563, pvalue=0.29224392869829086)
```
```
Point biserial correlation coefficient and p value for sale price and Parch:
```
Out[13]:
```
SignificanceResult(statistic=0.08162940708348347, pvalue=0.014799245374727947)
```

Given that the outcome of interest (Survived) is dichotomous, a series Point Biserial analyses were employed to examine associations between continuous IVs and the DV. As suggested by the series of Pearson correlations, Survived exhibited weak assocations with all continuous variables.

In [14]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# the independent variables set
x = titanic_training_data_cleaned[['Age', 'Fare', 'SibSp', 'Parch']]

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = x.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(x.values, i)
                          for i in range(len(x.columns))]
```

```
print(vif_data)
x.corr()
```

```
  feature       VIF
0     Age  1.436234
1    Fare  1.532322
2   SibSp  1.437732
3   Parch  1.493068
```

Out[14]:

	Age	Fare	SibSp	Parch
Age	1.000000	0.096688	-0.233296	-0.172482
Fare	0.096688	1.000000	0.159651	0.216225
SibSp	-0.233296	0.159651	1.000000	0.414838
Parch	-0.172482	0.216225	0.414838	1.000000

An inspection of VIF values suggested that there was no evidence of multicollinearity present among continuous variables.

Further examination of the data grouping by Survived vs Died

In [15]:
```
titanic_training_data_cleaned.groupby('Survived').mean()
```

Out[15]:

Survived	PassengerId	Pclass	Age	SibSp	Parch	Fare	Cabin_Data_Indicator	Child
0	447.016393	2.531876	30.028233	0.553734	0.329690	22.117887	0.123862	
1	444.368421	1.950292	28.291433	0.473684	0.464912	48.395408	0.397661	

Encode Embarked, Sex, and First Cabin Deck

In [16]:
```
# Encode Embarked, Sex, and First Cabin Deck
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

# Embarked
le.fit(np.array(titanic_training_data_cleaned['Embarked']).reshape(-1,1))
titanic_training_data_cleaned['encoded_Embarked'] = le.transform(titanic_training_data

# Cabin Deck
le.fit(np.array(titanic_training_data_cleaned['First_Cabin_Deck']).reshape(-1,1))
titanic_training_data_cleaned['encoded_FirstCabinDeck'] = le.transform(titanic_trainin

# Sex
le.fit(np.array(titanic_training_data_cleaned['Sex']).reshape(-1,1))
titanic_training_data_cleaned['encoded_Sex'] = le.transform(titanic_training_data_clea

Survived = titanic_training_data_cleaned['Survived'].to_list()
classification_model_data = titanic_training_data_cleaned.drop(
                                                              columns=['Sex','Passeng
                                                                        'Survived','Na
```

```
                                                    'Embarked','Fi
                                                    'Age', 'SibSp'
                                                    'Parch', 'Cabi
                                                   ])
```

```
C:\Users\cmark\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

Out[16]:
```
LabelEncoder()
```

Out[16]:
```
LabelEncoder()
```

Out[16]:
```
LabelEncoder()
```

In [17]:
```
classification_model_data.head()
```

Out[17]:

	Pclass	Child_Indicator	encoded_Embarked	encoded_Sex
0	3	0	2	1
1	1	0	0	0
2	3	0	2	0
3	1	0	2	0
4	3	0	2	1

Dummy coding categorical variables

In [18]:
```
cat_vars=['encoded_Embarked']

for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(classification_model_data[var], prefix=var)
    classification_model_data=classification_model_data.join(cat_list)

data_vars=classification_model_data.columns.values.tolist()
to_keep=[i for i in data_vars if i not in cat_vars]

data_final=classification_model_data[to_keep]
data_final.columns.values


classification_model_data = classification_model_data.drop(
                                                 columns=['encoded_Embar
                                                    ])
```

Out[18]:
```
array(['Pclass', 'Child_Indicator', 'encoded_Sex', 'encoded_Embarked_0',
       'encoded_Embarked_1', 'encoded_Embarked_2'], dtype=object)
```

In [19]:
```
np.array(classification_model_data)
titanic_training_data_cleaned['Survived']
```

Out[19]:
```
array([[3, 0, 1, 0, 1],
       [1, 0, 0, 0, 0],
       [3, 0, 0, 0, 1],
       ...,
       [3, 0, 0, 0, 1],
       [1, 0, 1, 0, 0],
       [3, 0, 1, 1, 0]], dtype=int64)
```

Out[19]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Logistic Regression

In [20]:
```
from scipy.signal._signaltools import _centered
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# predictors
X = np.array(classification_model_data)

# add constant
X = sm.add_constant(X)

# response
y = titanic_training_data_cleaned['Survived']

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary())
probabilities = result.predict(X)

# Convert probabilities to binary predictions using a 0.5 threshold
y_pred = np.where(probabilities > 0.5, 1, 0)

# Calculate the accuracy score
acc_score = accuracy_score(y, y_pred)
print("Accuracy Score:", acc_score)

# Confusion matrix

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve

cm = confusion_matrix(y, y_pred)
cm_logistic = cm
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Did not survive','
disp.plot()

# Curves
```

```
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import precision_recall_curve, auc
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import roc_auc_score


fpr, tpr, _ = roc_curve(np.array(y), probabilities)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('Logistic ROC Curve')
# roc auc score
auc1 = roc_auc_score(y, probabilities)
print("The roc auc score is:", auc1)

prec, recall, _ = precision_recall_curve(y, probabilities)
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
plt.title('Logistic Precision-Recall Curve')
# precision-recall auc score
auc2 = auc(recall, prec)
print("The prec-recall auc score is:", auc2)
```

```
Optimization terminated successfully.
         Current function value: 0.452127
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               Survived   No. Observations:                  891
Model:                          Logit   Df Residuals:                      885
Method:                           MLE   Df Model:                            5
Date:                Sun, 23 Apr 2023   Pseudo R-squ.:                  0.3210
Time:                        21:10:34   Log-Likelihood:                -402.85
converged:                       True   LL-Null:                       -593.33
Covariance Type:            nonrobust   LLR p-value:                 3.751e-80
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          3.6276      0.335     10.830      0.000       2.971       4.284
x1            -1.0279      0.116     -8.898      0.000      -1.254      -0.801
x2             1.0096      0.264      3.818      0.000       0.491       1.528
x3            -2.5875      0.187    -13.851      0.000      -2.954      -2.221
x4             0.0098      0.368      0.027      0.979      -0.712       0.732
x5            -0.4979      0.229     -2.172      0.030      -0.947      -0.049
==============================================================================
Accuracy Score: 0.7800224466891134
```

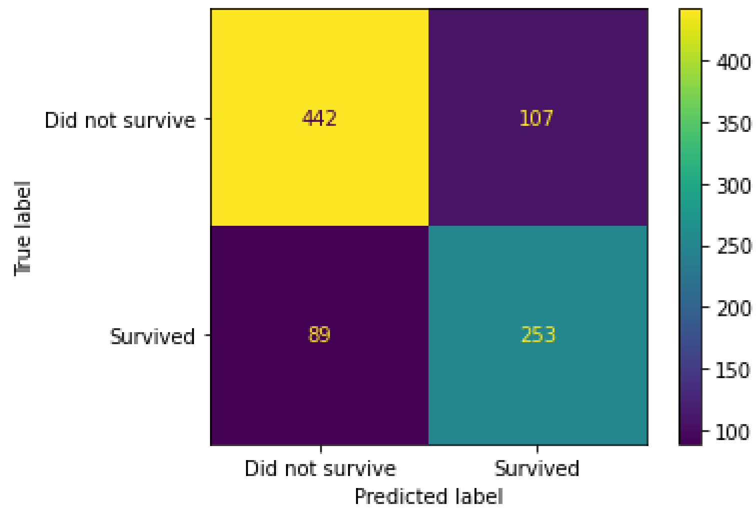
Out[20]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a65ad29be0>
```

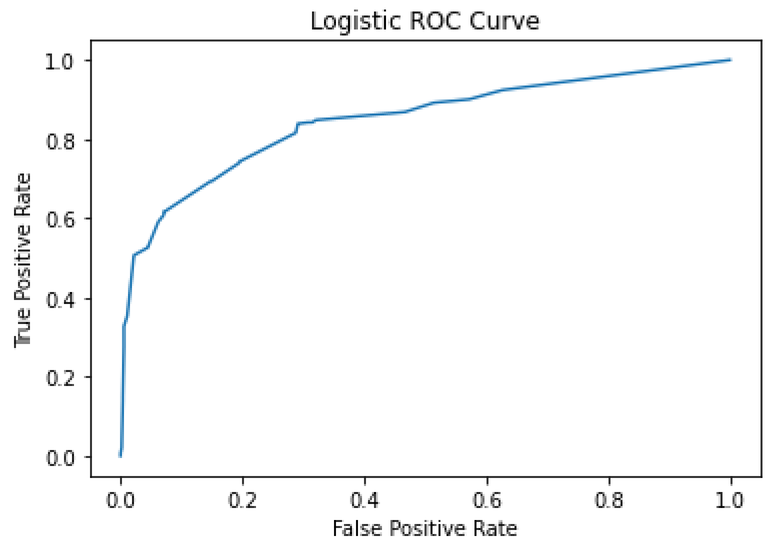
Out[20]:
```
Text(0.5, 1.0, 'Logistic ROC Curve')
```

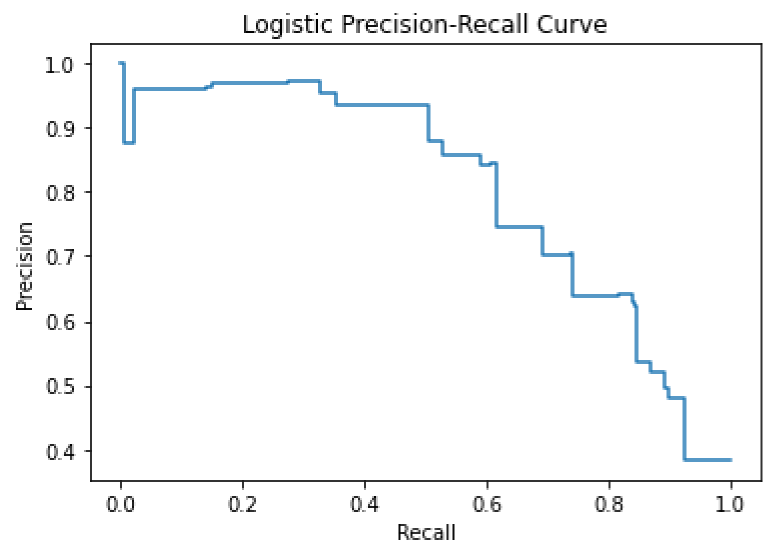
```
The roc auc score is: 0.8458707485167076
```

Out[20]:
```
Text(0.5, 1.0, 'Logistic Precision-Recall Curve')
```

```
The prec-recall auc score is: 0.8153271635412416
```







Logistic regression: Odds ratios

In [21]:
```
model_odds = pd.DataFrame(np.exp(result.params), columns= ['OR'])
model_odds['z-value']= result.pvalues
model_odds[['2.5%', '97.5%']] = np.exp(result.conf_int())

model_odds
```

Out[21]:

	OR	z-value	2.5%	97.5%
const	37.622838	2.473094e-27	19.513796	72.537295
x1	0.357756	5.682098e-19	0.285270	0.448660
x2	2.744373	1.345671e-04	1.634429	4.608084
x3	0.075209	1.257526e-43	0.052151	0.108464
x4	1.009827	9.788171e-01	0.490630	2.078449
x5	0.607779	2.984026e-02	0.387811	0.952512

- Coef. 1 Pclass, Passenger class: Passenger class was significantly, negatively associated with survival. Of note: interpretation of passenger class given the directionality and operationalization of Pclass suggests that those lower class status were less likely to survive. Those with a lower class status were about 63% less likely to survive.
- Coef. 2 Child_Indicator, binary variable indicating if a child: Being a child was significantly, positively associated with survival. Children on board were more likely to have survived. Children were 2.7 times more likely to have survived compared to adults (anyone 18 years old or older).
- Coef. 3 encoded_Sex, dichotomous variable where 1 = male, ref = female: Relative to women, being a man was significantly, negatively associated with survival. Men were less likely to have survived. Men were 93% less likely to have survived compared to women.
- Coef. 4 encoded_Embarked_1, binary, embarked at S = Southampton and ref Q = Queenstown: Compared to those that embarked in Queenstown, there doesn't seem to be a statistically significant differences in embarking from Southampton.
- Coef. 5 encoded_Embarked_2, binary, embarked at C = Cherbourg and ref Q = Queenstown: Similarly, embarking in Cherbourg was negatively associated with survival, and this association was significant. There was a lower likelihood of survival for those that embarked at Cherbourg. Those that embarked at Cherbourg were 39% less likely to have survived.

Logistic Assumption check: multicollinearity

In [22]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# the independent variables set
x = classification_model_data[['Pclass', 'Child_Indicator',
                               'encoded_Embarked_1', 'encoded_Embarked_2', 'encoded_Se

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = x.columns

# calculating VIF for each feature
```

```
vif_data["VIF"] = [variance_inflation_factor(x.values, i)
                          for i in range(len(x.columns))]

print(vif_data)
x.corr()
```

```
            feature       VIF
0            Pclass  5.975058
1   Child_Indicator  1.187256
2 encoded_Embarked_1  1.571048
3 encoded_Embarked_2  4.224112
4       encoded_Sex  2.779324
```

Out[22]:

	Pclass	Child_Indicator	encoded_Embarked_1	encoded_Embarked_2	encoded_
Pclass	1.000000	0.125620	0.221009	0.074053	0.131
Child_Indicator	0.125620	1.000000	-0.033194	0.000543	-0.107
encoded_Embarked_1	0.221009	-0.033194	1.000000	-0.499421	-0.074
encoded_Embarked_2	0.074053	0.000543	-0.499421	1.000000	0.119
encoded_Sex	0.131900	-0.107150	-0.074115	0.119224	1.000

Using a VIF cutoff value of 5.0, there is some evidence of multicollinearity within this multiple logistic regression model with this particular set of predictors.

In [23]:

```
classification_model_data.describe()
```

Out[23]:

	Pclass	Child_Indicator	encoded_Sex	encoded_Embarked_1	encoded_Embarked_2
count	891.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	0.126824	0.647587	0.086420	0.725028
std	0.836071	0.332962	0.477990	0.281141	0.446751
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	0.000000	0.000000
50%	3.000000	0.000000	1.000000	0.000000	1.000000
75%	3.000000	0.000000	1.000000	0.000000	1.000000
max	3.000000	1.000000	1.000000	1.000000	1.000000

Logistic regression assumption check: linearity between continuous var (Pclass) and DV (Survived)

In [24]:

```
## import statsmodels.api as sm
from statsmodels.genmod.generalized_linear_model import GLM
from statsmodels.genmod import families

df_titanic_lt = titanic_training_data_cleaned.copy()  # lt = logit transform

# Define continuous variables
continuous_var = ['Pclass']
```

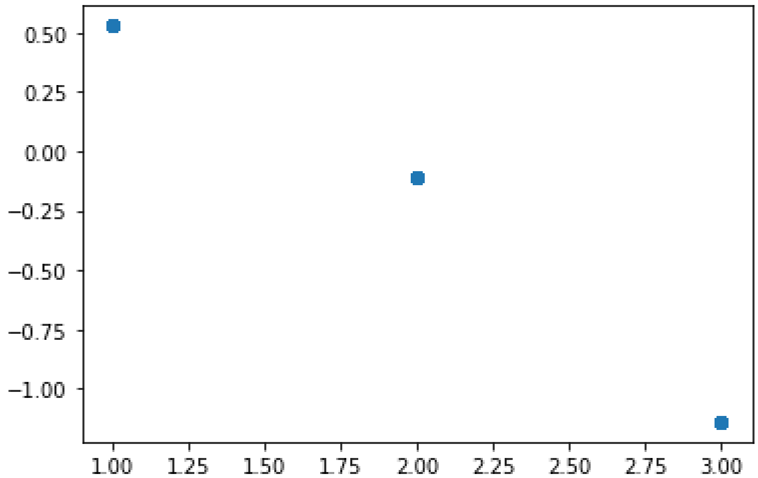
```
# Add logit transform interaction terms (natural log) for continuous variables e.g. Ag
for var in continuous_var:
    df_titanic_lt[f'{var}:Log_{var}'] = df_titanic_lt[var].apply(lambda x: x * np.log(

# Keep columns related to continuous variables
cols_to_keep = continuous_var + df_titanic_lt.columns.tolist()[-len(continuous_var):]
cols_to_keep

# Redefine independent variables to include interaction terms
X_lt = df_titanic_lt[cols_to_keep]
y_lt = df_titanic_lt['Survived']

# Add constant
X_lt_constant = sm.add_constant(X_lt, prepend=False)

# Build model and fit the data (using statsmodel's Logit)
logit_results = GLM(y_lt, X_lt_constant, family=families.Binomial()).fit()

# Display summary results
print(logit_results.summary())

predicted = logit_results.predict(X_lt_constant)

# Get log odds values
log_odds = np.log(predicted / (1 - predicted))

# Visualize predictor continuous variable vs logit values (Age)
plt.scatter(x=X_lt_constant['Pclass'].values, y=log_odds);
plt.show()
```

Out[24]:

```
['Pclass', 'Pclass:Log_Pclass']

                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               Survived   No. Observations:                  891
Model:                            GLM   Df Residuals:                      888
Model Family:                Binomial   Df Model:                            2
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -541.55
Date:                Sun, 23 Apr 2023   Deviance:                       1083.1
Time:                        21:10:36   Pearson chi2:                     891.
No. Iterations:                     4   Pseudo R-squ. (CS):             0.1097
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
Pclass                0.3979      1.099      0.362      0.717      -1.756       2.551
Pclass:Log_Pclass    -0.7483      0.657     -1.139      0.255      -2.036       0.539
const                 0.1327      1.171      0.113      0.910      -2.163       2.429
=====================================================================================
```

Out[24]:

```
<matplotlib.collections.PathCollection at 0x1a65cf4bdf0>
```



A visual inspection of this assumption:

- We are interested in the p-values for the logit transformed interaction terms of Pclass:Log_Pclass
- From the summary table above, the p value for Pclass:Log_Pclass is > .05, which is non-significant
- This means that there is linearity in the PClass feature, and the assumption has been not been violated.
- *Note: Pclass is treated as continuous/numerical in this model*

LDA Model

In [25]:
```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(classification_model_data)
y = Survived
lda = LinearDiscriminantAnalysis(solver = 'svd')
lda.fit(X, y)
predictions = lda.predict(X)

# Basic stats for LDA result

print('Coefficients:',lda.coef_)
print('Means:',lda.means_)
print('Score:',lda.score(X,y))

# Confusion matrix

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve

cm = confusion_matrix(y, predictions)
cm_lda = cm
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Did not survive','
disp.plot()

# Curves
```

```
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay

probabilities = lda.predict_proba(X)[:,1]

fpr, tpr, _ = roc_curve(np.array(Survived), probabilities)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('LDA ROC Curve')
# roc auc score
auc1 = roc_auc_score(y, probabilities)
print("The roc auc score is:", auc1)

prec, recall, _ = precision_recall_curve(Survived, probabilities)
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
plt.title('LDA Precision-Recall Curve')
# precision-recall auc score
auc2 = auc(recall, prec)
print("The prec-recall auc score is:", auc2)
```

Out[25]:
```
LinearDiscriminantAnalysis()
```

```
Coefficients: [[-0.93688562  0.35078045 -1.61862964 -0.00919148 -0.24042946]]
Means: [[ 0.26715372 -0.09647984  0.42885221 -0.00288115  0.11814043]
 [-0.42885202  0.15487553 -0.68842066  0.004625   -0.18964648]]
Score: 0.7912457912457912
```

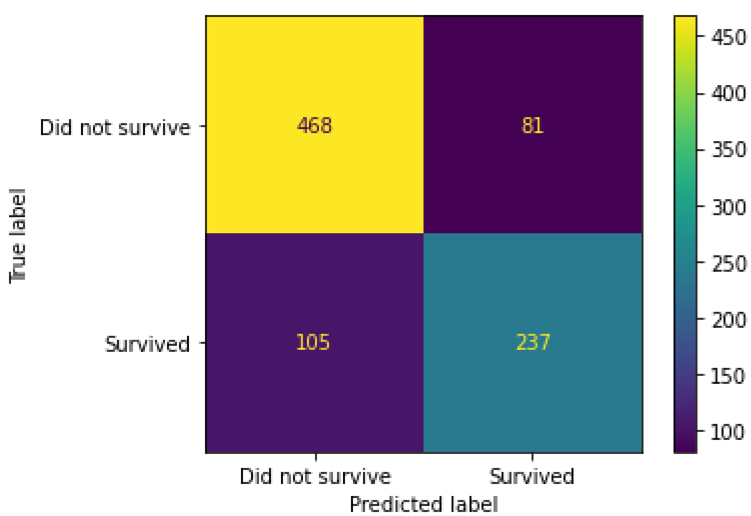
Out[25]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a65cf0f3d0>
```

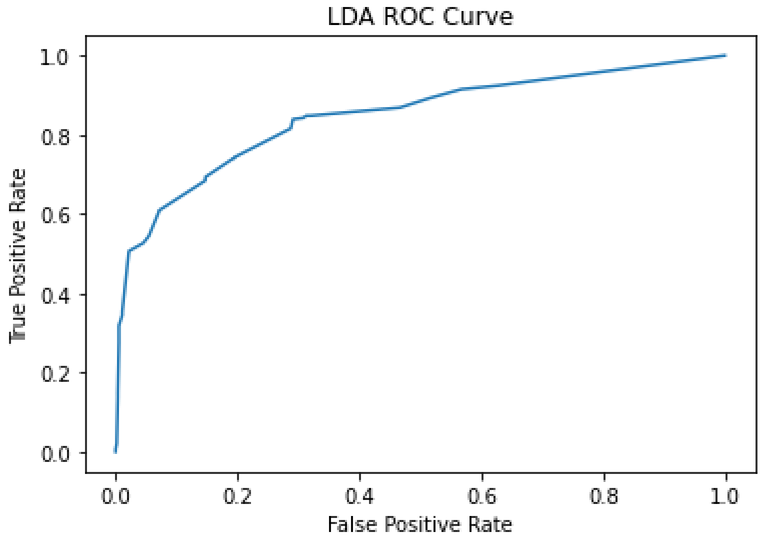
Out[25]:
```
Text(0.5, 1.0, 'LDA ROC Curve')
```

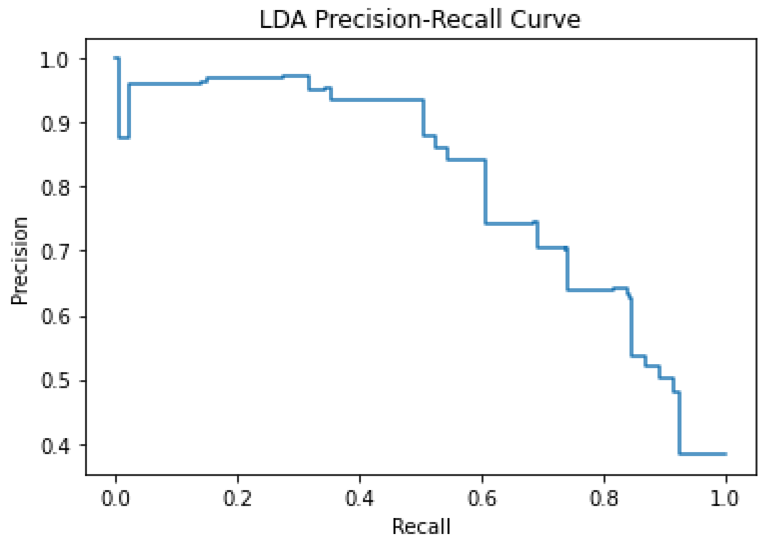
```
The roc auc score is: 0.8458787375238339
```

Out[25]:
```
Text(0.5, 1.0, 'LDA Precision-Recall Curve')
```

```
The prec-recall auc score is: 0.8133706940021923
```







QDA Model

In [26]:
```
# Try with QDA (Quadratic Discriminant Analysis)

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis()
qda.fit(X, y)

# Basic stats for QDA result

print('Means:',qda.means_)
print('Score:',qda.score(X,y))

predictions = qda.predict(X)
probabilities = qda.predict_proba(X)[:,1]

# Confusion
cm = confusion_matrix(y, predictions)
cm_qda = cm
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Did not survive','
disp.plot()
```

```python
# Curves
fpr, tpr, _ = roc_curve(y, probabilities)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('QDA ROC Curve')
# roc auc score
auc1 = roc_auc_score(y, probabilities)
print("The roc auc score is:", auc1)

prec, recall, _ = precision_recall_curve(y, probabilities)
pr_display = PrecisionRecallDisplay(precision=prec, recall=recall).plot()
plt.title('QDA Precision-Recall Curve')# precision-recall auc score
auc2 = auc(recall, prec)
print("The prec-recall auc score is:", auc2)
```

Out[26]:
```
QuadraticDiscriminantAnalysis()
```

```
Means: [[ 0.26715372 -0.09647984  0.42885221 -0.00288115  0.11814043]
 [-0.42885202  0.15487553 -0.68842066  0.004625   -0.18964648]]
Score: 0.7676767676767676
```

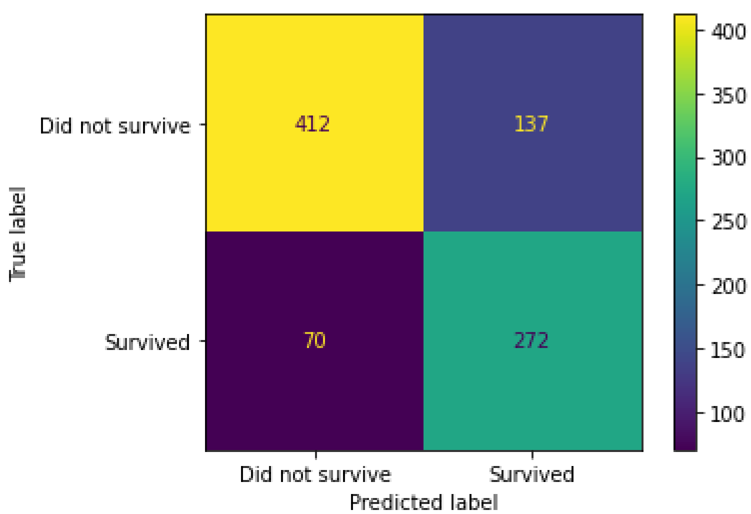
Out[26]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a65d31b460>
```

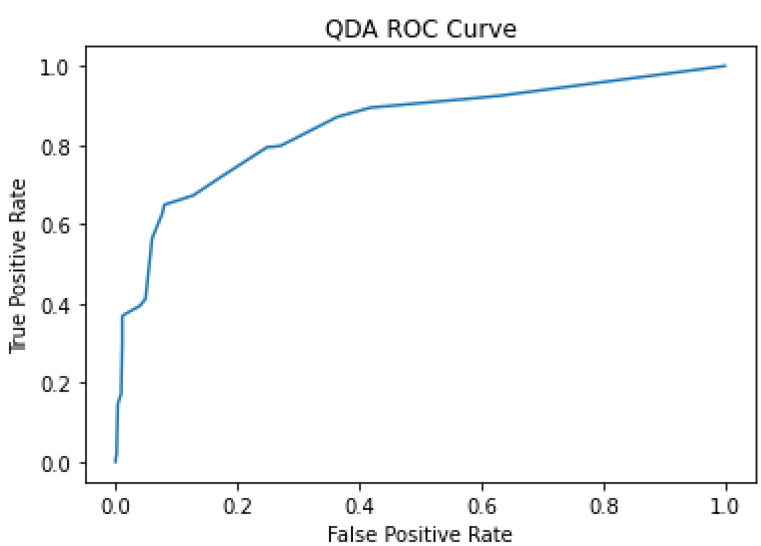
Out[26]:
```
Text(0.5, 1.0, 'QDA ROC Curve')
```

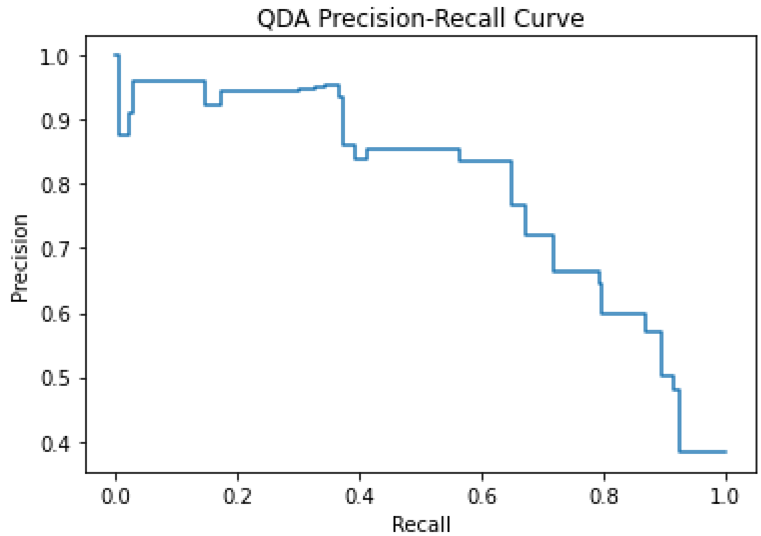
```
The roc auc score is: 0.8458707485167077
```

Out[26]:
```
Text(0.5, 1.0, 'QDA Precision-Recall Curve')
```

```
The prec-recall auc score is: 0.7998892536727021
```







In [27]:
```
# More detialed performance statistics:

#Sensitivity = TP / (TP + FN) = recall  = TPR
#Specificity = TN / (TN + FP) = TNR
#Precision = TP / (TP + FP) = PPV
#PPV = TP / ( TP + FP )  (sensitivity * prevalence) / ((sensitivity * prevalence) + (1
#NPV = TN / (TN + FN) = ((specificity * (1-prevalence)) / ( (sensitivity * (1-preveler

#Recall =
#Precision = (sensitivity * prevalence) / ((sensitivity * prevalence) + (1-specificity

### QDA ###

tn = cm_logistic[0,0]
fp = cm_logistic[0,1]
fn = cm_logistic[1,0]
tp = cm_logistic[1,1]

print('')
print('Logistic Regression')
print('sensitivity', tp / (tp + fn))
print('specificity', tn / (tn + fp))
print('recall', tp / (tp + fn))
print('precision', tp / (tp + fp))
print('accuracy', (tp + tn)/(tp+tn+fp+fn))
#print('training logistic sklearn score',logit_model.score(X,y)) # using 'score' metho




### LDA ###

tn = cm_lda[0,0]
fp = cm_lda[0,1]
fn = cm_lda[1,0]
tp = cm_lda[1,1]

print('')
print('LINEAR DISCRIMINANT ANALYSIS')
print('sensitivity', tp / (tp + fn))
print('specificity', tn / (tn + fp))
```

```
print('recall', tp / (tp + fn))
print('precision', tp / (tp + fp))
print('accuracy', (tp + tn)/(tp+tn+fp+fn))
print('training LDA sklearn score',lda.score(X,y)) # using 'score' method for scikitle
### QDA ###

tn = cm_qda[0,0]
fp = cm_qda[0,1]
fn = cm_qda[1,0]
tp = cm_qda[1,1]

print('')
print('QUADRATIC DISCRIMINANT ANALYSIS')
print('sensitivity', tp / (tp + fn))
print('specificity', tn / (tn + fp))
print('recall', tp / (tp + fn))
print('precision', tp / (tp + fp))
print('accuracy', (tp + tn)/(tp+tn+fp+fn))
print('training QDA sklearn score',qda.score(X,y)) # using 'score' method for scikitle
```

```
Logistic Regression
sensitivity 0.7397660818713451
specificity 0.8051001821493625
recall 0.7397660818713451
precision 0.7027777777777777
accuracy 0.7800224466891134

LINEAR DISCRIMINANT ANALYSIS
sensitivity 0.6929824561403509
specificity 0.8524590163934426
recall 0.6929824561403509
precision 0.7452830188679245
accuracy 0.7912457912457912
training LDA sklearn score 0.7912457912457912

QUADRATIC DISCRIMINANT ANALYSIS
sensitivity 0.7953216374269005
specificity 0.7504553734061931
recall 0.7953216374269005
precision 0.6650366748166259
accuracy 0.7676767676767676
training QDA sklearn score 0.7676767676767676
```

QDA appears to have somewhat better metrics for both recall and precision, but slightly worse performance with respect to specificity. As a result, the overall 'accuracy' metric of LDA appears slightly preferablet to QDA. We also consider the 'score' method for the sklearn implementation of each model. This attribute is the

Create K-Nearest Neighbors Model

Make modifications to the training dataset that are specifically needed for a K-Nearest Neighbors model

In [28]:
```
# Create a new training dataframe specifically for the KNN model so that we don't inte
# used for other models
```

```
knn_training_validation_df = titanic_training_data_cleaned.copy(deep=True)

# dummy encode the Sex and PClass variables
knn_training_validation_df = pd.get_dummies(knn_training_validation_df, columns=['Sex'

# Apply Min-Max Scaling to the Fare variable
import os
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler()

knn_training_validation_df[['min_max_scaled_fare']] = min_max_scaler.fit_transform(knn
knn_training_validation_df[['min_max_scaled_Pclass']] = min_max_scaler.fit_transform(k

# show first five rows of the data
knn_training_validation_df.head(10)

# Check that min-max scaling applied to the fare variable correctly
knn_training_validation_df['min_max_scaled_fare'].describe()
knn_training_validation_df['min_max_scaled_Pclass'].describe()
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Embarked	Cabin
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	S	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	C	
2	3	1	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	S	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803	53.1000	S	
4	5	0	3	Allen, Mr. William Henry	35.0	0	0	373450	8.0500	S	
5	6	0	3	Moran, Mr. James	28.0	0	0	330877	8.4583	Q	
6	7	0	1	McCarthy, Mr. Timothy J	54.0	0	0	17463	51.8625	S	
7	8	0	3	Palsson, Master. Gosta Leonard	2.0	3	1	349909	21.0750	S	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	27.0	0	2	347742	11.1333	S	
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	14.0	1	0	237736	30.0708	C	

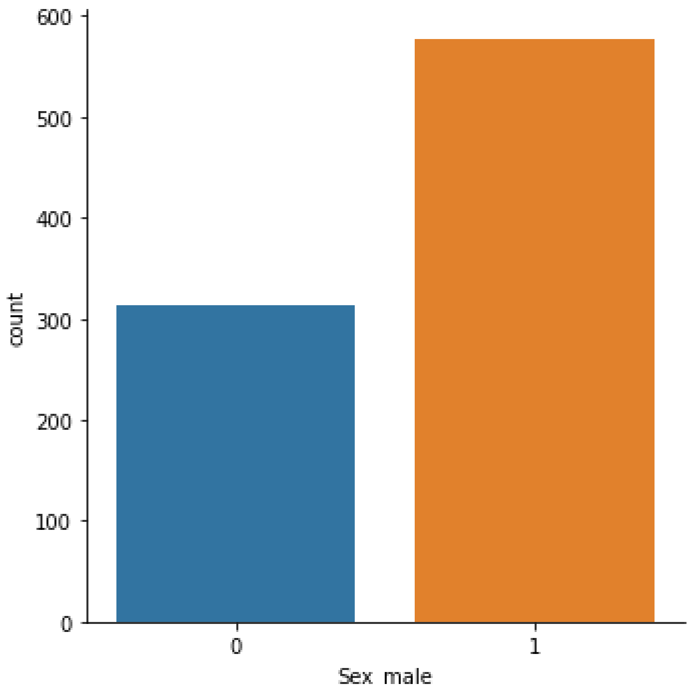
Out[28]:

```
count    891.000000
mean       0.062858
std        0.096995
min        0.000000
25%        0.015440
50%        0.028213
75%        0.060508
max        1.000000
Name: min_max_scaled_fare, dtype: float64
```

Out[28]:
```
count    891.000000
mean       0.654321
std        0.418036
min        0.000000
25%        0.500000
50%        1.000000
75%        1.000000
max        1.000000
```

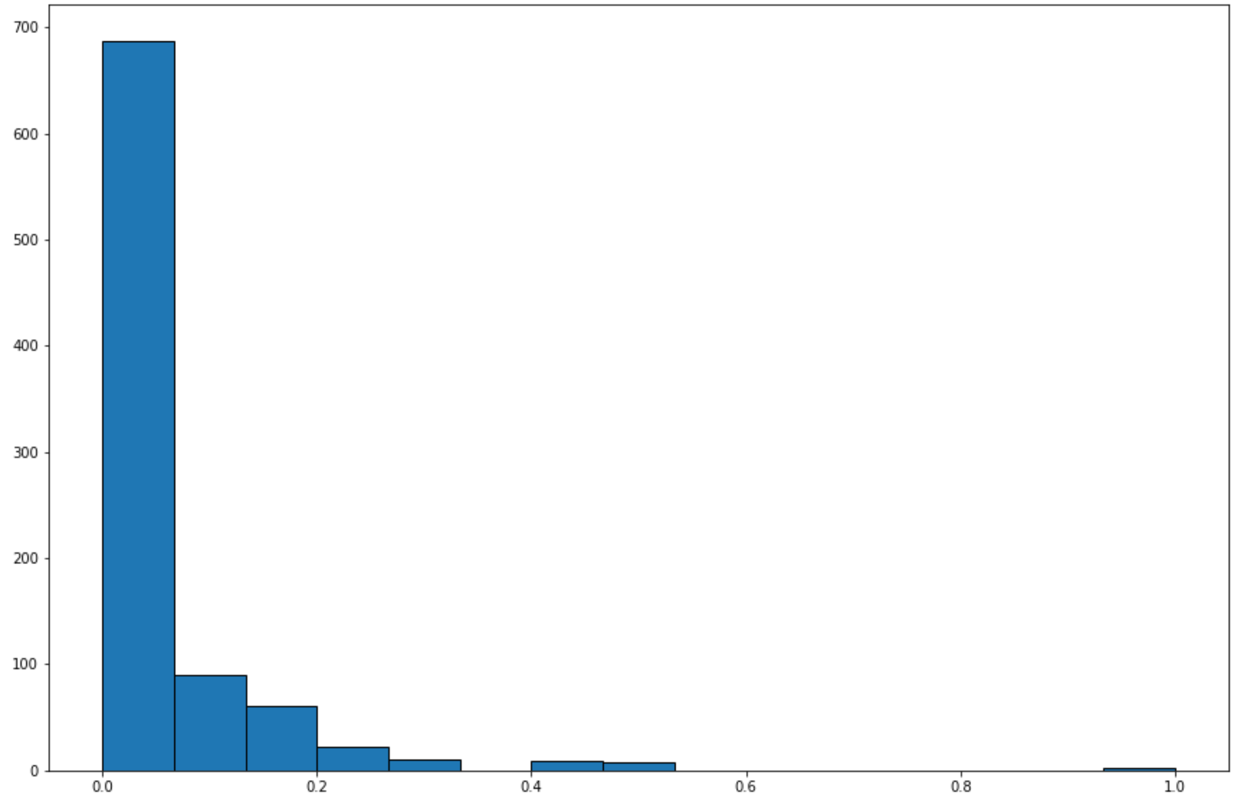
In [29]:
```python
# Create visualizations for the distributions for each of our new variables

# New Indicator Variable Visualizations - Sex_male
sns.catplot(x = 'Sex_male', kind = 'count', data = knn_training_validation_df)
```

Out[29]:
```
<seaborn.axisgrid.FacetGrid at 0x1a65d3a7f40>
```



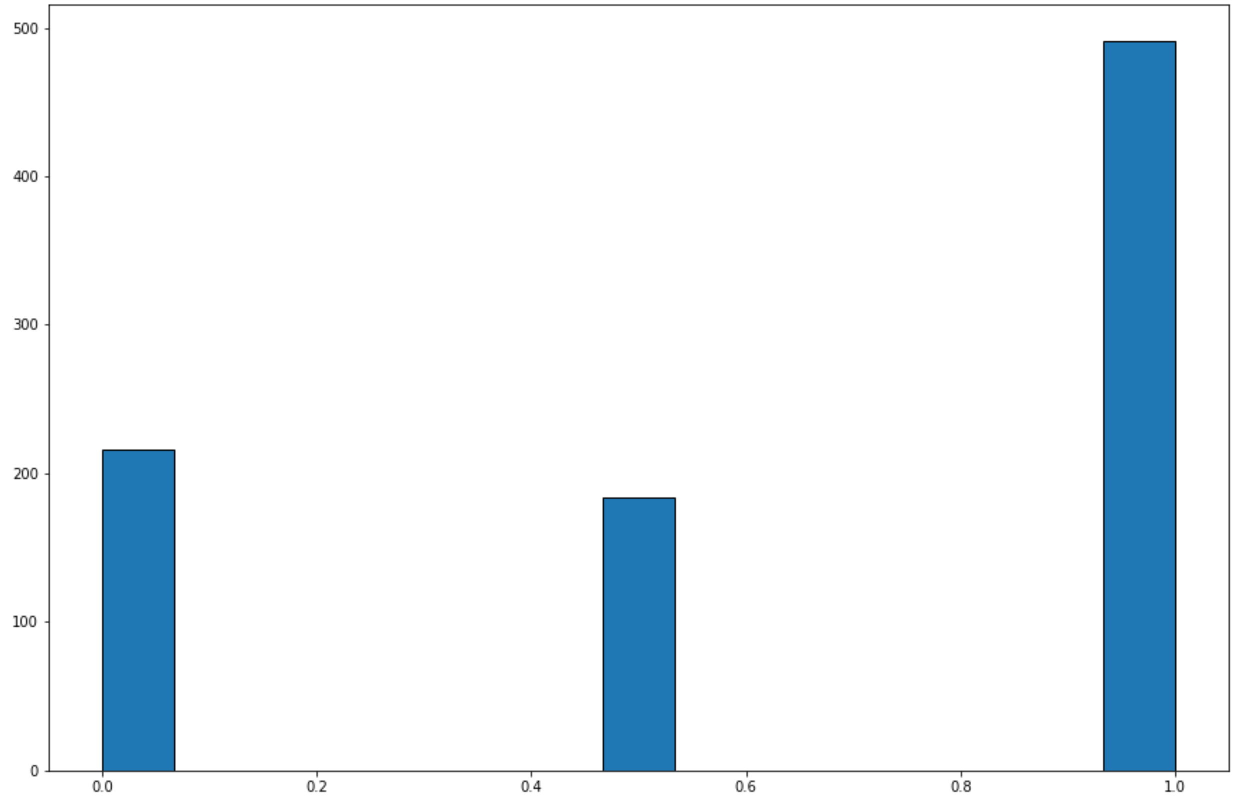
In [30]:
```python
# New Numeric Variable Visualizations - Min_Max_Scaled_Fare
knn_training_validation_df['min_max_scaled_fare'].hist(edgecolor = 'black',
                                                       bins = 15, figsize = (15, 10),
                                                       grid = False)
```

Out[30]:
```
<Axes: >
```



In [31]:
```
# New Numeric Variable Visualizations - Min_Max_Scaled_Pclass
knn_training_validation_df['min_max_scaled_Pclass'].hist(edgecolor = 'black',
                                            bins = 15, figsize = (15, 10),
                                            grid = False)
```

Out[31]: <Axes: >



Split the KNN training/validation dataframe into two training and validation dataframes (which will better enable us to tune hyperparameters later).

In [32]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor

# Split Kaggle's Training dataframe into training and validation dataframes
knn_training_validation_x = knn_training_validation_df.drop(columns=['PassengerId',
                                                                      'Survived',
                                                                     'Pclass',
                                                                     'Name',
                                                                     'Age',
                                                                     'SibSp',
                                                                     'Parch',
                                                                     'Ticket',
                                                                     'Fare',
                                                                     'Embarked',
                                                                     'First_Cabin_Deck'
knn_training_validation_y = knn_training_validation_df['Survived']

X_train_knn, X_validation_knn, y_train_knn, y_validation_knn = train_test_split(knn_tr
                                                                                knn_tr
                                                                                test_s
                                                                                random

# Fit a K-Nearest Neighbors Model to the Training Dataframe
# Use the validation dataframe to tune the hyperparameters in such a way that we find
# model weights.

parameters = {
     "n_neighbors": range(1, 50),
     "weights": ["uniform", "distance"],
 }

gridsearch = GridSearchCV(KNeighborsRegressor(), parameters)
gridsearch.fit(X_train_knn, y_train_knn)
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': range(1, 50),
                         'weights': ['uniform', 'distance']})

knn_optimal_weights_method = gridsearch.best_params_["weights"]

knn_optimal_k = gridsearch.best_params_["n_neighbors"]

print(f"The optimal value of k is {knn_optimal_k}.")

print(f"The optimal way of assigning weights in the knn model is via the {knn_optimal_
```

Out[32]:
```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': range(1, 50),
                         'weights': ['uniform', 'distance']})
```

Out[32]:
```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': range(1, 50),
                         'weights': ['uniform', 'distance']})
```

```
The optimal value of k is 12.
The optimal way of assigning weights in the knn model is via the uniform method.
```

In [33]:

```python
{'n_neighbors': knn_optimal_k, 'weights': knn_optimal_weights_method}
validation_preds_grid = gridsearch.predict(X_validation_knn)
validation_mse = mean_squared_error(y_validation_knn, validation_preds_grid)
validation_rmse = sqrt(validation_mse)

print(f"After applying the optimal KNN model to the validation dataset, the root mean
```

Out[33]:

```
{'n_neighbors': 12, 'weights': 'uniform'}
```

```
After applying the optimal KNN model to the validation dataset, the root mean squared
error of the predictions is 0.4012.
```

Create a Receiving Operating Characteristic (ROC) Curve using the data from the validation dataset

In [34]:

```python
from matplotlib import rcParams

validation_df_knn = X_validation_knn.join(y_validation_knn)
validation_df_knn['knn_prediction'] = validation_preds_grid.tolist()

rcParams['figure.figsize'] = 8, 8

Survived = validation_df_knn[validation_df_knn["Survived"].isin([1])]
Died = validation_df_knn[validation_df_knn["Survived"].isin([0])]

Survived_Count = Survived.shape[0]
Died_Count = Died.shape[0]

Prediction_Cutoff = np.arange(0, 1, 0.01).tolist()
Cutoff_df = pd.DataFrame(Prediction_Cutoff, columns=['Prediction_Cutoff'])

Sensitivity_List = []
False_Positive_List = []
Precision_List = []


for pc in Cutoff_df['Prediction_Cutoff']:
    true_positive_count = (Survived['knn_prediction'] >= pc).sum()
    Sensitivity_List.append(true_positive_count / Survived_Count)
    false_positive_count = (Died['knn_prediction'] >= pc).sum()
    False_Positive_List.append(false_positive_count / Died_Count)
    Precision_List.append(true_positive_count / (true_positive_count + false_positive_

Cutoff_df['Sensitivity'] = Sensitivity_List
Cutoff_df['False Positive Rate'] = False_Positive_List
Cutoff_df['Precision'] = Precision_List

fig, ax = plt.subplots()
ax.plot(Cutoff_df['False Positive Rate'], Cutoff_df['Sensitivity'], 'red')
ax.set_title('ROC Curve', fontsize = 15)
ax.set_xlabel('False Positive Rate', fontsize = 15)
ax.set_ylabel('Sensitivity', fontsize = 15)

plt.tick_params(axis='both', which='major', labelsize=12)
```

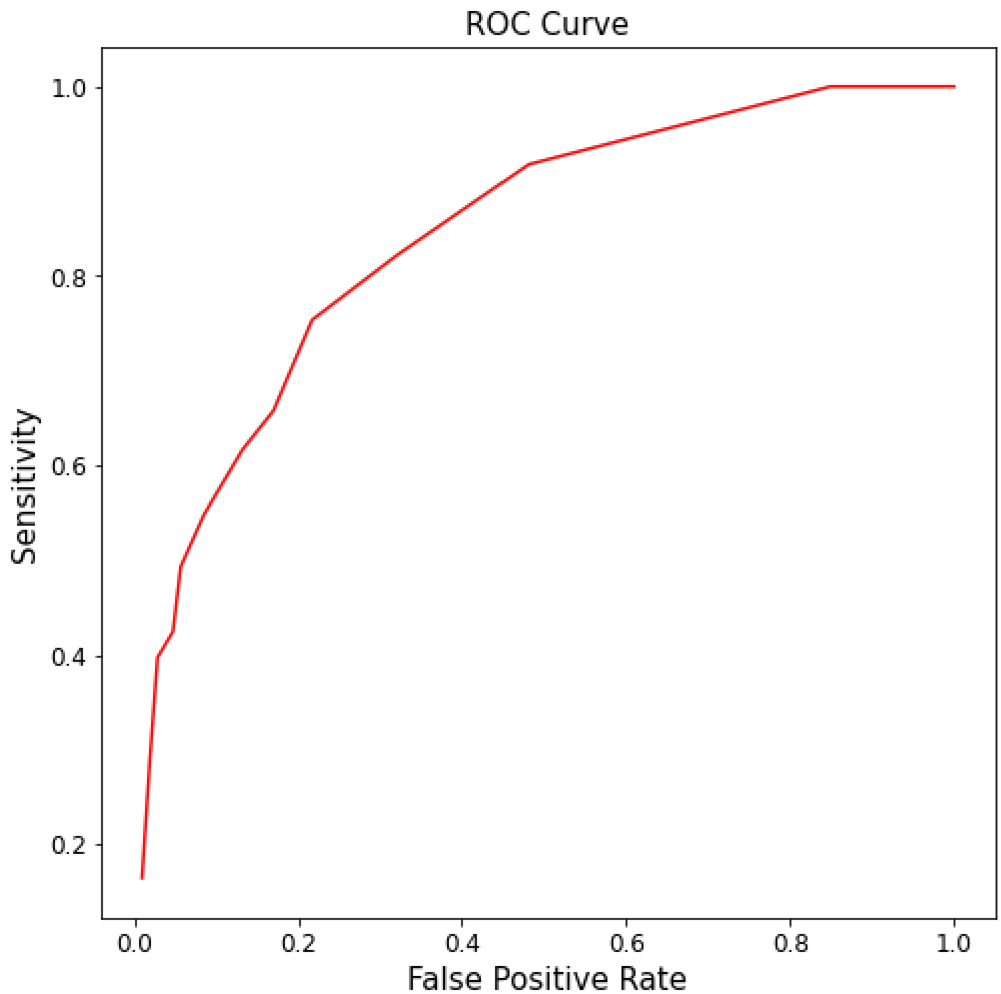
Out[34]:

```
[<matplotlib.lines.Line2D at 0x1a65da7da30>]
```

Out[34]: Text(0.5, 1.0, 'ROC Curve')

Out[34]: Text(0.5, 0, 'False Positive Rate')

Out[34]: Text(0, 0.5, 'Sensitivity')



Create a Precision - Recall Curve using the data from the validation dataset

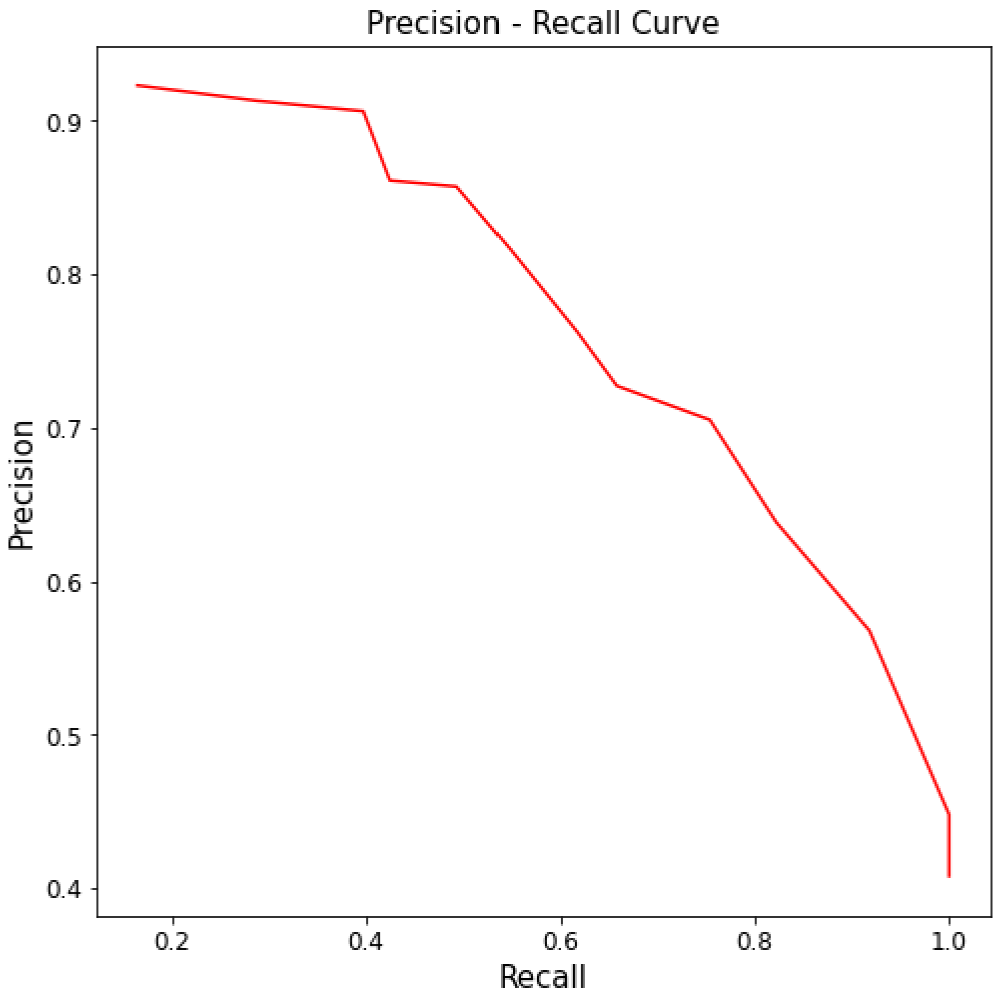
In [35]:
```
fig, ax = plt.subplots()
ax.plot(Cutoff_df['Sensitivity'], Cutoff_df['Precision'], 'red')
ax.set_title('Precision - Recall Curve', fontsize = 15)
ax.set_xlabel('Recall', fontsize = 15)
ax.set_ylabel('Precision', fontsize = 15)

plt.tick_params(axis='both', which='major', labelsize=12)
```

Out[35]: [<matplotlib.lines.Line2D at 0x1a65dadca60>]

Out[35]: Text(0.5, 1.0, 'Precision - Recall Curve')

Out[35]: Text(0.5, 0, 'Recall')

Out[35]: Text(0, 0.5, 'Precision')



Find the cutoff value that maximizes the percent of the validation dataset assigned accurate predictions

In [36]:
```
Cutoff_df['Specificity'] = 1 - Cutoff_df['False Positive Rate']
Cutoff_df['Percent_Correctly_Predicted_In_Validation_DF'] = ( ( Survived.shape[0] * Cu

KNN_Cutoff_Shortlist_df = Cutoff_df[Cutoff_df.Percent_Correctly_Predicted_In_Validatic

Optimal_KNN_Cutoff = KNN_Cutoff_Shortlist_df['Prediction_Cutoff'].median()

print(f"The optimal cutoff boundary for predicting survivals for this model is {Optima
```

```
The optimal cutoff boundary for predicting survivals for this model is 0.2950.
```

Import and Clean Testing Dataset

Import the Titanic Testing Dataset

In [37]:
```
import pandas as pd
titanic_testing_data = pd.read_csv('test.csv')

# show first five rows of the data
titanic_testing_data.head(100)
# show number of columns and rows
titanic_testing_data.shape
```

Out[37]:

	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
...	...	...	...	...	...	...	...	...	...	...	...
95	987	3	Tenglin, Mr. Gunnar Isidor	male	25.0	0	0	350033	7.7958	NaN	S
96	988	1	Cavendish, Mrs. Tyrell William (Julia Florence...	female	76.0	1	0	19877	78.8500	C46	S
97	989	3	Makinen, Mr. Kalle Edvard	male	29.0	0	0	STON/O 2. 3101268	7.9250	NaN	S
98	990	3	Braf, Miss. Elin Ester Maria	female	20.0	0	0	347471	7.8542	NaN	S
99	991	3	Nancarrow, Mr. William Henry	male	33.0	0	0	A./5. 3338	8.0500	NaN	S

100 rows × 11 columns

Out[37]:

```
(418, 11)
```

Check the testing dataset for missing values

In [38]:

```
# find null counts, percentage of null values, and column type
null_count = titanic_testing_data.isnull().sum()
null_percentage = titanic_testing_data.isnull().sum() * 100 / len(titanic_testing_data
column_type = titanic_testing_data.dtypes

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
```

```
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[38]:

	Missing Count	Percentage Missing	Column Type
Cabin	327	78.229665	object
Age	86	20.574163	float64
Fare	1	0.239234	float64

Appropriately address the missing values in the testing dataframe. Add the newly created variables as well to the testing dataframe.

In [39]:
```
# Create a new dataframe called titanic_training_data_cleaned so we don't modify the c
titanic_testing_data_cleaned = titanic_testing_data.copy(deep=True)

# change Null for Fare with the median value from the Training dataset
titanic_testing_data_cleaned['Fare'].fillna(titanic_training_data_cleaned['Fare'].medi

# fill Nulls for Age with median value from the Training dataset
titanic_testing_data_cleaned['Age'].fillna(titanic_training_data_cleaned['Age'].median

# Create new cabin-related variables that will be more useful and cleaner than the ori
titanic_testing_data_cleaned['Cabin_Data_Indicator'] = titanic_testing_data_cleaned['C

titanic_testing_data_cleaned['First_Cabin_Deck'] = np.where(titanic_testing_data_clean
                                                            titanic_testing_data_clea
                                                            'None')

# Create a new variable indicating whether a passenger is a child
titanic_testing_data_cleaned['Child_Indicator'] = titanic_testing_data_cleaned['Age']
titanic_testing_data_cleaned['Child_Indicator'] = titanic_testing_data_cleaned['Child_

# Drop the original Cabin variable since it has so many null values and since some pas
# making the original variable difficult to work with
titanic_testing_data_cleaned.drop(['Cabin'],axis=1,inplace=True)
```

Examine whether the desired modifications to the testing dataframe applied correctly.

In [40]:
```
# show first five rows of the data
titanic_training_data_cleaned.head(20)
# show number of columns and rows
titanic_training_data_cleaned.shape
```

Out[40]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	
5	6	0	3	Moran, Mr. James	male	28.0	0	0	330877	8.4583	
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	
12	13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.8542	
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.1250	
17	18	1	2	Williams, Mr. Charles Eugene	male	28.0	0	0	244373	13.0000	
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.0000	
19	20	1	3	Masselmani, Mrs. Fatima	female	28.0	0	0	2649	7.2250	

(891, 17)

In [41]:
```
# find null counts, percentage of null values, and column type
null_count = titanic_training_data_cleaned.isnull().sum()
null_percentage = titanic_training_data_cleaned.isnull().sum() * 100 / len(titanic_tra
column_type = titanic_training_data_cleaned.dtypes

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[41]:

	Missing Count	Percentage Missing	Column Type

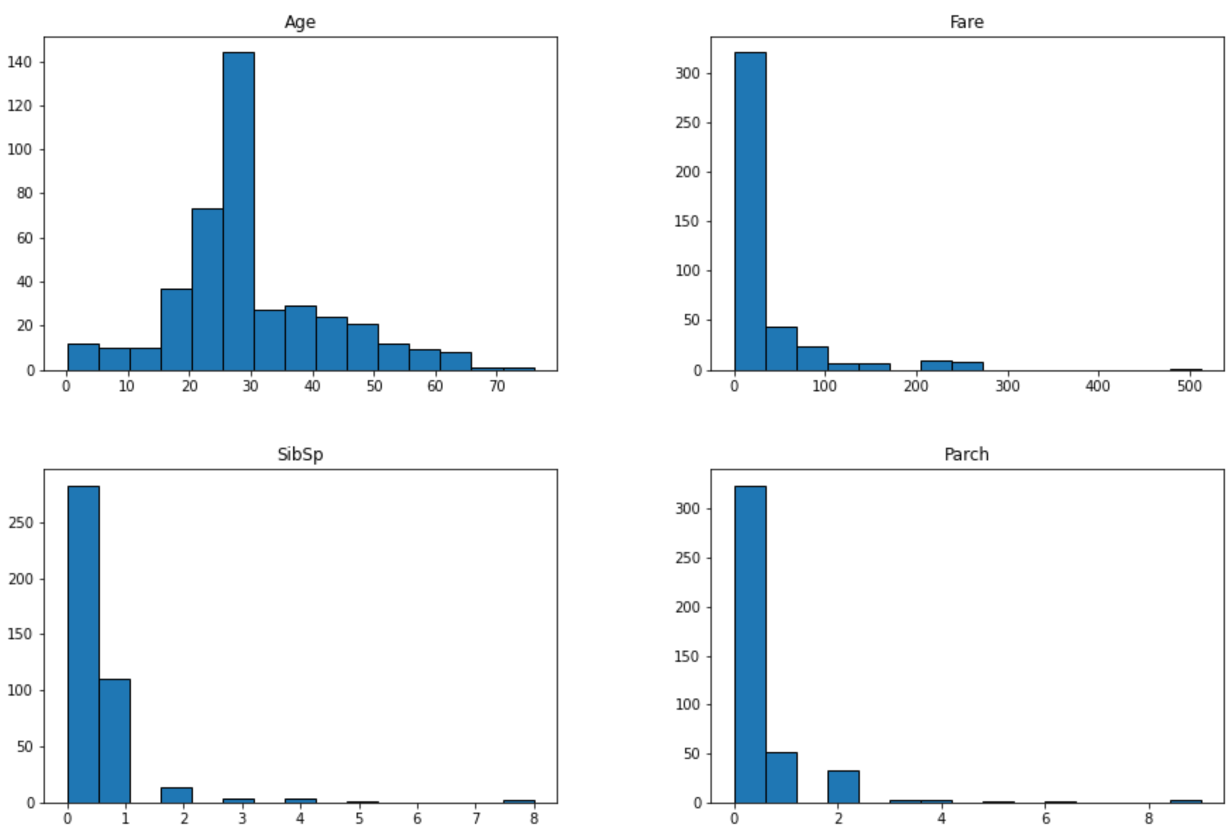
Conduct exploratory data analysis on the variables in the testing dataframe to confirm that all the values appear to be reasonable (to proactively address data value errors if needed).

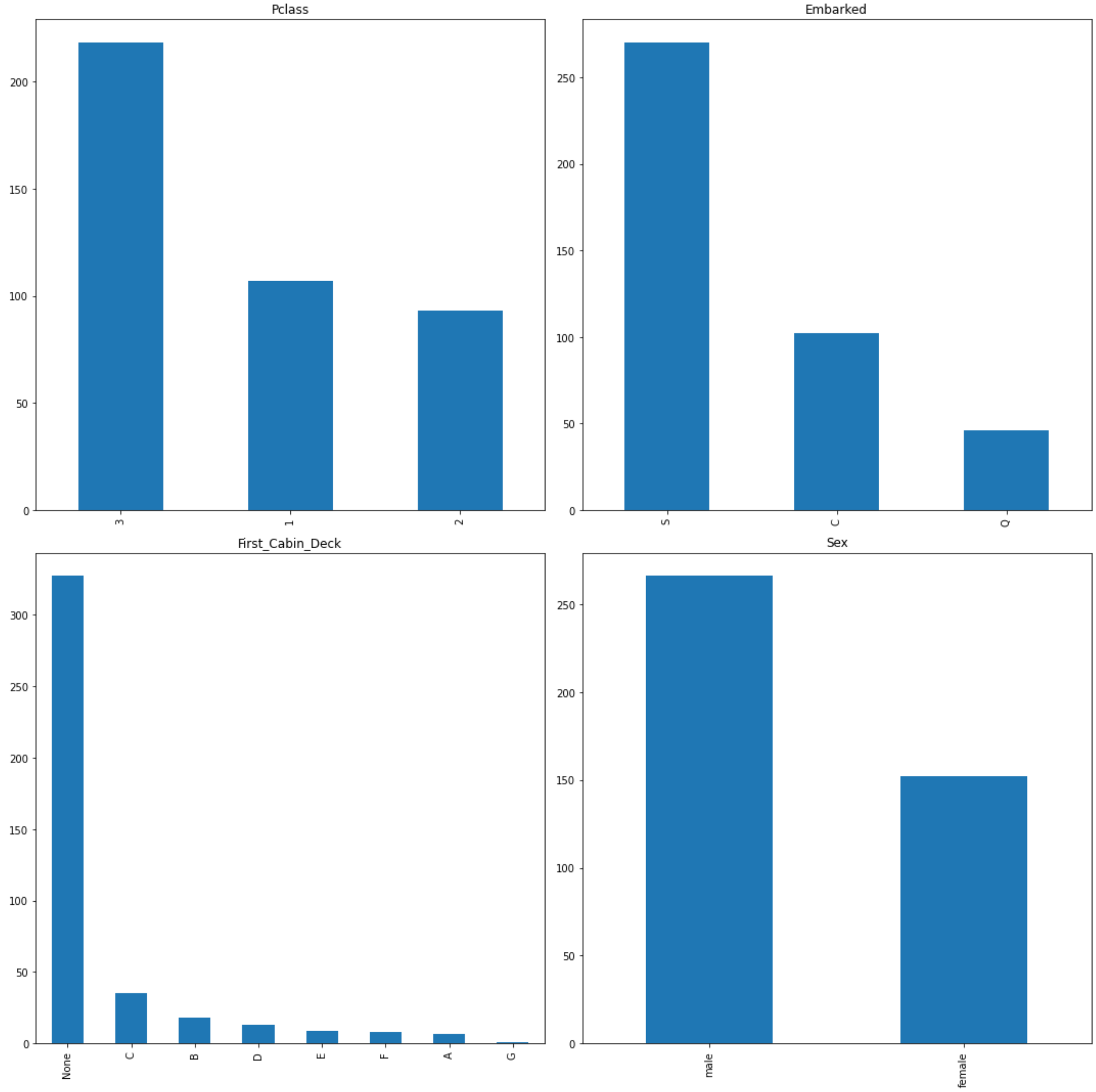
In [42]:
```
# Numeric Variable Visualizations
titanic_testing_data_cleaned[numeric_variables].describe()

titanic_testing_data_cleaned[numeric_variables].hist(edgecolor = 'black',
                                                      bins = 15, figsize = (15, 10),
                                                      layout = (2, 2), grid = False)



# Categorical Variable Visualizations
fig, ax = plt.subplots(2, 2, figsize = (15, 15))
for var, subplot in zip(categorical_variables, ax.flatten()):
    titanic_testing_data_cleaned[var].value_counts().plot(kind = 'bar', ax = subplot,

fig.tight_layout()
```

Out[42]:

	Age	Fare	SibSp	Parch
count	418.000000	418.000000	418.000000	418.000000
mean	29.805024	35.576535	0.447368	0.392344
std	12.667969	55.850103	0.896760	0.981429
min	0.170000	0.000000	0.000000	0.000000
25%	23.000000	7.895800	0.000000	0.000000
50%	28.000000	14.454200	0.000000	0.000000
75%	35.750000	31.471875	1.000000	0.000000
max	76.000000	512.329200	8.000000	9.000000

Out[42]:
```
array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Fare'}>],
       [<Axes: title={'center': 'SibSp'}>,
        <Axes: title={'center': 'Parch'}>]], dtype=object)
```
Out[42]:
```
<Axes: title={'center': 'Pclass'}>
```
Out[42]:
```
<Axes: title={'center': 'Embarked'}>
```
Out[42]:
```
<Axes: title={'center': 'First_Cabin_Deck'}>
```
Out[42]:
```
<Axes: title={'center': 'Sex'}>
```



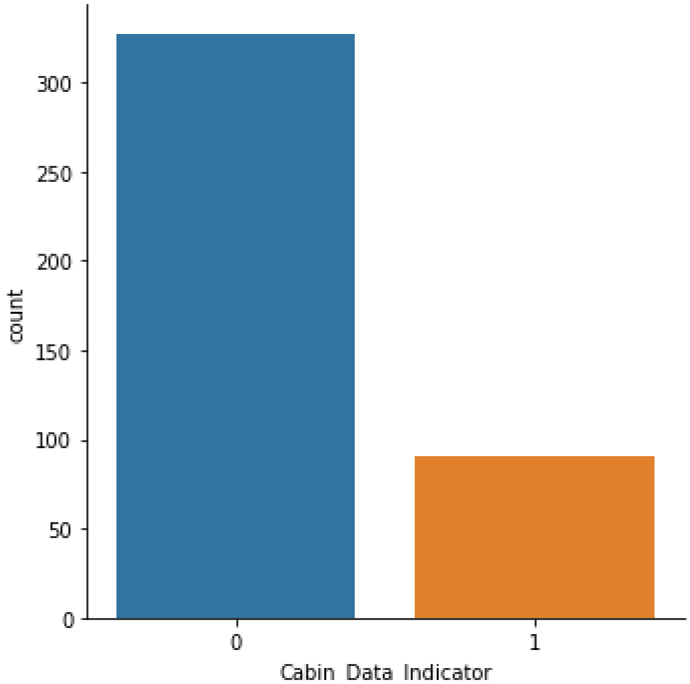


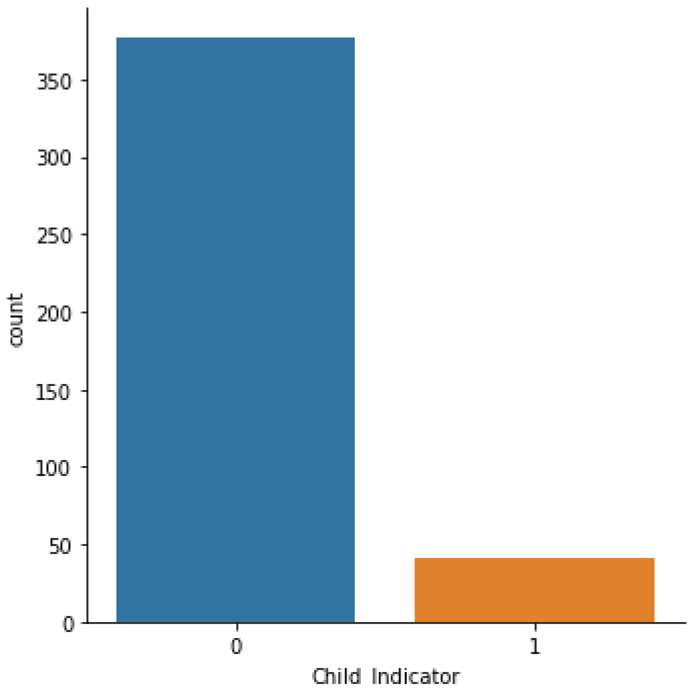
In [43]:

```
# Indicator Variable Visualizations
indicator_predictors = ['Cabin_Data_Indicator', 'Child_Indicator']


for var, subplot in zip(indicator_predictors, ax.flatten()):
    sns.catplot(x = var, kind = 'count', data = titanic_testing_data_cleaned)

fig.tight_layout()
```

Out[43]: <seaborn.axisgrid.FacetGrid at 0x1a65de03130>

Out[43]: <seaborn.axisgrid.FacetGrid at 0x1a65e72ae50>





Encode Embarked, Sex, and First Cabin Deck in Test Dataset

In [44]:

```
# Encode Embarked, Sex, and First Cabin Deck
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

# Embarked
le.fit(np.array(titanic_testing_data_cleaned['Embarked']).reshape(-1,1))
titanic_testing_data_cleaned['encoded_Embarked'] = le.transform(titanic_testing_data_c

# Cabin Deck
le.fit(np.array(titanic_testing_data_cleaned['First_Cabin_Deck']).reshape(-1,1))
titanic_testing_data_cleaned['encoded_FirstCabinDeck'] = le.transform(titanic_testing_
```

```
# Sex
le.fit(np.array(titanic_testing_data_cleaned['Sex']).reshape(-1,1))
titanic_testing_data_cleaned['encoded_Sex'] = le.transform(titanic_testing_data_cleane


test_classification_model_data = titanic_testing_data_cleaned.drop(
                                                              columns=['Sex','Passeng
                                                                       'Embarked','Fi
                                                                       'Age', 'SibSp'
                                                                       'Parch', 'Cabi
                                                                      ])
test_classification_model_data.head(10)
```

```
C:\Users\cmark\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

Out[44]: LabelEncoder()

Out[44]: LabelEncoder()

Out[44]: LabelEncoder()

Out[44]:

	Pclass	Child_Indicator	encoded_Embarked	encoded_Sex
0	3	0	1	1
1	3	0	2	0
2	2	0	1	1
3	3	0	2	1
4	3	0	2	0
5	3	1	2	1
6	3	0	1	0
7	2	0	2	1
8	3	0	0	0
9	3	0	2	1

Encode Categorical Variables (Embarked)

In [45]:

```
cat_vars=['encoded_Embarked']

for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(test_classification_model_data[var], prefix=var)
    test_classification_model_data=test_classification_model_data.join(cat_list)

data_vars=test_classification_model_data.columns.values.tolist()
to_keep=[i for i in data_vars if i not in cat_vars]

test_data_final=test_classification_model_data[to_keep]
test_data_final.columns.values
```

```
test_classification_model_data = test_classification_model_data.drop(
                                                               columns=['encoded_Embar
                                                                       ])

# show dataframe with encoded categorical variables
test_classification_model_data.head(10)
```

Out[45]:

```
array(['Pclass', 'Child_Indicator', 'encoded_Sex', 'encoded_Embarked_0',
       'encoded_Embarked_1', 'encoded_Embarked_2'], dtype=object)
```

Out[45]:

	Pclass	Child_Indicator	encoded_Sex	encoded_Embarked_1	encoded_Embarked_2
0	3	0	1	1	0
1	3	0	0	0	1
2	2	0	1	1	0
3	3	0	1	0	1
4	3	0	0	0	1
5	3	1	1	0	1
6	3	0	0	1	0
7	2	0	1	0	1
8	3	0	0	0	0
9	3	0	1	0	1

Apply Logistic Regression Model to the Testing Dataset

In [46]:

```
# Generate predictions on test dataset

# predictors
X_test = np.array(test_classification_model_data)

# add constant
X_test = sm.add_constant(X_test)

# use logistic model constructed using the training dataset
probabilities = result.predict(X_test)

# Convert probabilities to binary predictions using a 0.5 threshold
y_test_pred = np.where(probabilities > 0.5, 1, 0)
```

In [47]:

```
# Put the logistic predictions into a Pandas dataframe
prediction_df_logistic = pd.DataFrame(y_test_pred, columns=['Survived'])


# Add the PassengerId column to the front of the logistic predictions dataframe
prediction_df_logistic.insert(0, 'PassengerId', titanic_testing_data_cleaned['Passenge

# Show first 5 rows of the df
prediction_df_logistic.head()

# Output predictions to csv
prediction_df_logistic.to_csv('test_predictions_logistic_v1.csv', index=False)
```

Out[47]:

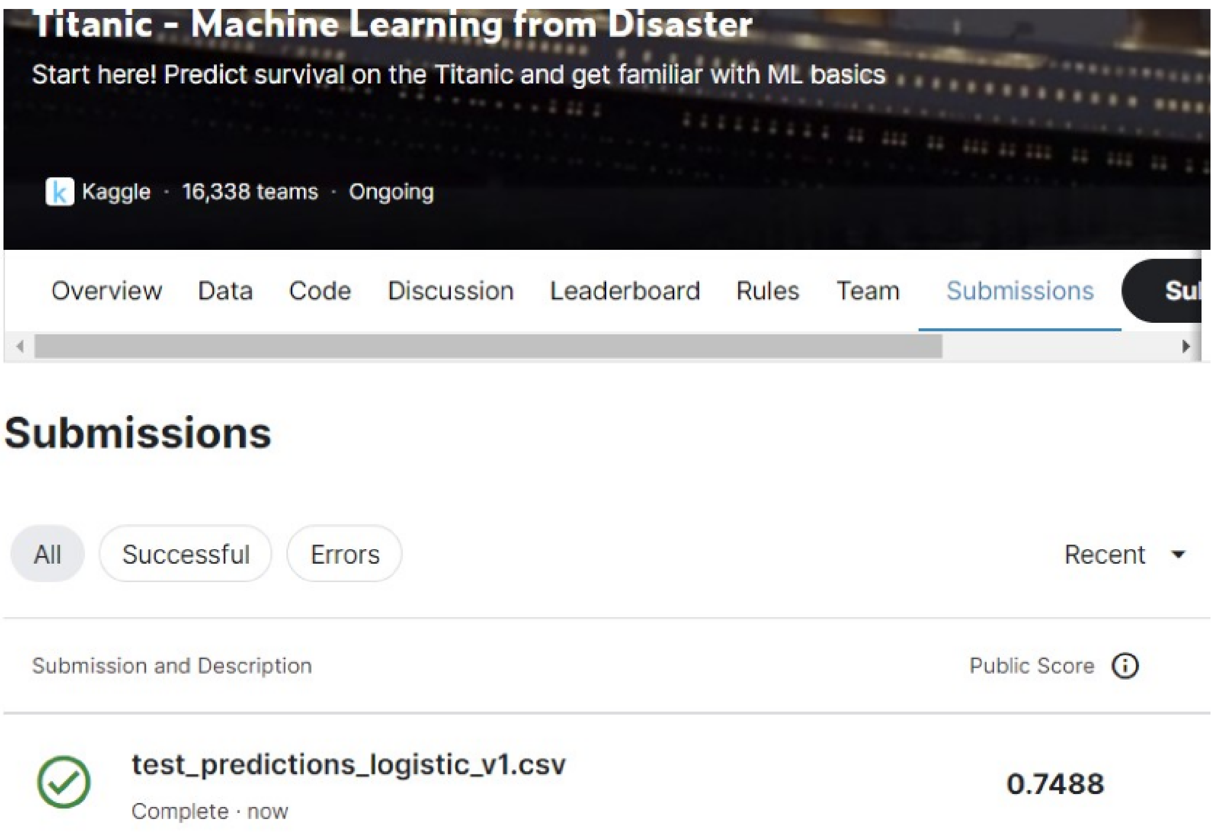
	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

In [48]:
```
# Display the kaggle results associated with the Logistic Regression
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_Logistic_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[48]: <Figure size 1080x1080 with 0 Axes>

Out[48]: <matplotlib.image.AxesImage at 0x1a65df63640>

Out[48]: (-0.5, 852.5, 609.5, -0.5)



Apply LDA Model to Testing Dataset

In [49]:
```
# Generate predictions on test dataset

# scale predictors since that is what we did in the training data
scaler = StandardScaler()
```

```
X_test_scale = scaler.fit_transform(test_classification_model_data)

# predict survival on test dataset using lda model
lda_probabilities = lda.predict(X_test_scale)

# Convert probabilities to binary predictions using a 0.5 threshold
y_test_lda_pred = np.where(lda_probabilities > 0.5, 1, 0)
```

In [50]:
```
# Put the lda predictions into a Pandas dataframe
prediction_df_lda = pd.DataFrame(y_test_lda_pred, columns=['Survived'])

# Add the PassengerId column to the front of the lda predictions dataframe
prediction_df_lda.insert(0, 'PassengerId', titanic_testing_data_cleaned['PassengerId']

# Show first 5 rows of the df
prediction_df_lda.head()

# Output predictions to csv
prediction_df_lda.to_csv('test_predictions_lda_v1.csv', index=False)
```

Out[50]:

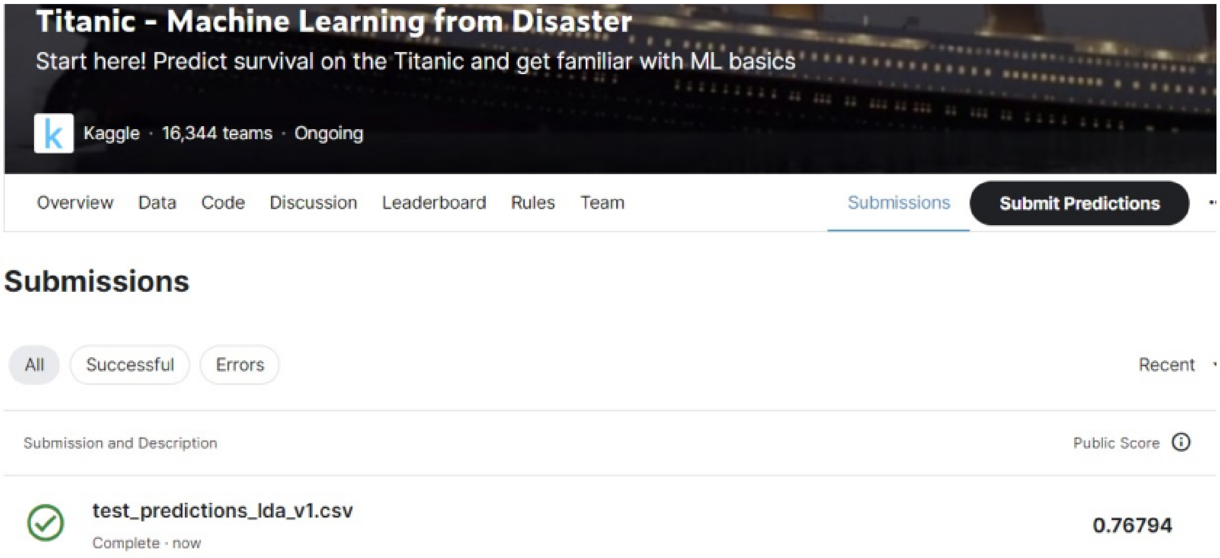
	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

In [51]:
```
# Display the kaggle results associated with the LDA
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_LDA_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[51]:
```
<Figure size 1080x1080 with 0 Axes>
```

Out[51]:
```
<matplotlib.image.AxesImage at 0x1a65e37d5e0>
```

Out[51]:
```
(-0.5, 1231.5, 579.5, -0.5)
```



Apply QDA Model to Testing Dataset

In [52]:
```
# Generate predictions on test dataset

# scale predictors since that is what we did in the training data
scaler = StandardScaler()
X_test_scale = scaler.fit_transform(test_classification_model_data)

# predict survival on test dataset using qda model
qda_probabilities = qda.predict(X_test_scale)

# Convert probabilities to binary predictions using a 0.5 threshold
y_test_qda_pred = np.where(qda_probabilities > 0.5, 1, 0)
```

In [53]:
```
# Put the qda predictions into a Pandas dataframe
prediction_df_qda = pd.DataFrame(y_test_qda_pred, columns=['Survived'])

# Add the PassengerId column to the front of the qda predictions dataframe
prediction_df_qda.insert(0, 'PassengerId', titanic_testing_data_cleaned['PassengerId']

# Show first 5 rows of the df
prediction_df_qda.head()

# Output predictions to csv
prediction_df_qda.to_csv('test_predictions_qda_v1.csv', index=False)
```

Out[53]:

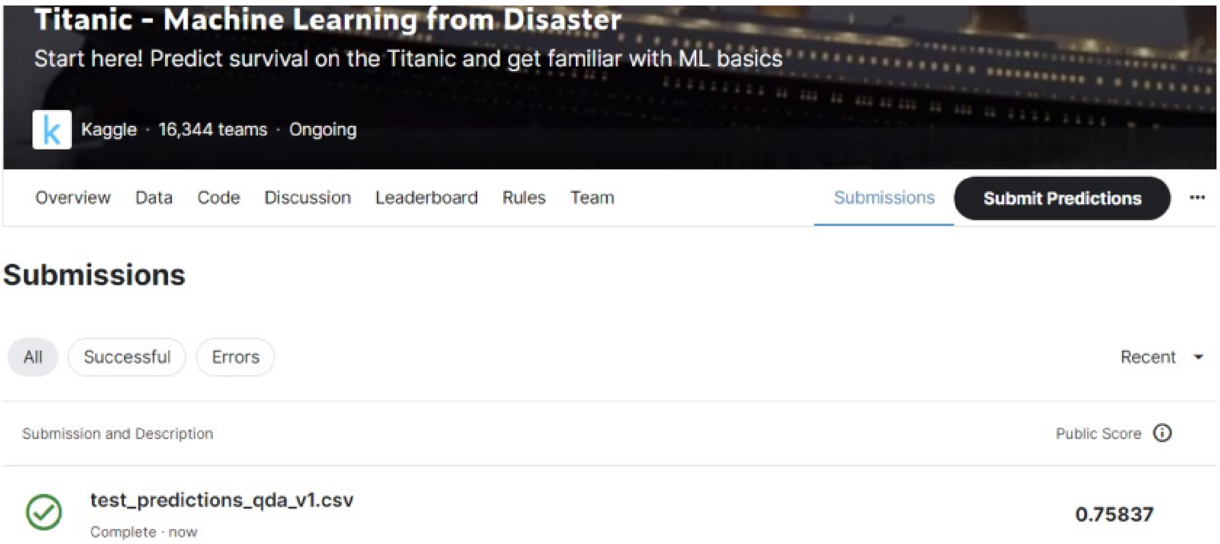
	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

In [54]:
```
# Display the kaggle results associated with the QDA
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_QDA_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[54]:
```
<Figure size 1080x1080 with 0 Axes>
```

Out[54]:
```
<matplotlib.image.AxesImage at 0x1a65dfbcf40>
```

Out[54]:
```
(-0.5, 1250.5, 570.5, -0.5)
```



Apply K-Nearest Neighbors Model to the Testing Dataset

Adjust the testing dataset to reflect the modifications made to the KNN training dataset

In [55]:
```
# Create a new training dataframe specifically for the KNN model so that we don't inte
# used for other models
knn_testing_df = titanic_testing_data_cleaned.copy(deep=True)

# dummy encode the Sex and PClass variables
knn_testing_df = pd.get_dummies(knn_testing_df, columns=['Sex'], drop_first=True)

# Apply Min-Max Scaling to the Fare variable

min_max_scaler = MinMaxScaler()

knn_testing_df[['min_max_scaled_fare']] = min_max_scaler.fit_transform(knn_testing_df[
knn_testing_df[['min_max_scaled_Pclass']] = min_max_scaler.fit_transform(knn_testing_d


# show first five rows of the data
knn_testing_df.head(10)

# Check that min-max scaling applied to the fare variable correctly
knn_testing_df['min_max_scaled_fare'].describe()
knn_testing_df['min_max_scaled_Pclass'].describe()
```

Out[55]:

	PassengerId	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Embarked	Cabin_Data_Indica
0	892	3	Kelly, Mr. James	34.5	0	0	330911	7.8292	Q	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	47.0	1	0	363272	7.0000	S	
2	894	2	Myles, Mr. Thomas Francis	62.0	0	0	240276	9.6875	Q	
3	895	3	Wirz, Mr. Albert	27.0	0	0	315154	8.6625	S	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	22.0	1	1	3101298	12.2875	S	
5	897	3	Svensson, Mr. Johan Cervin	14.0	0	0	7538	9.2250	S	
6	898	3	Connolly, Miss. Kate	30.0	0	0	330972	7.6292	Q	
7	899	2	Caldwell, Mr. Albert Francis	26.0	1	1	248738	29.0000	S	
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	18.0	0	0	2657	7.2292	C	
9	901	3	Davies, Mr. John Samuel	21.0	2	0	A/4 48871	24.1500	S	

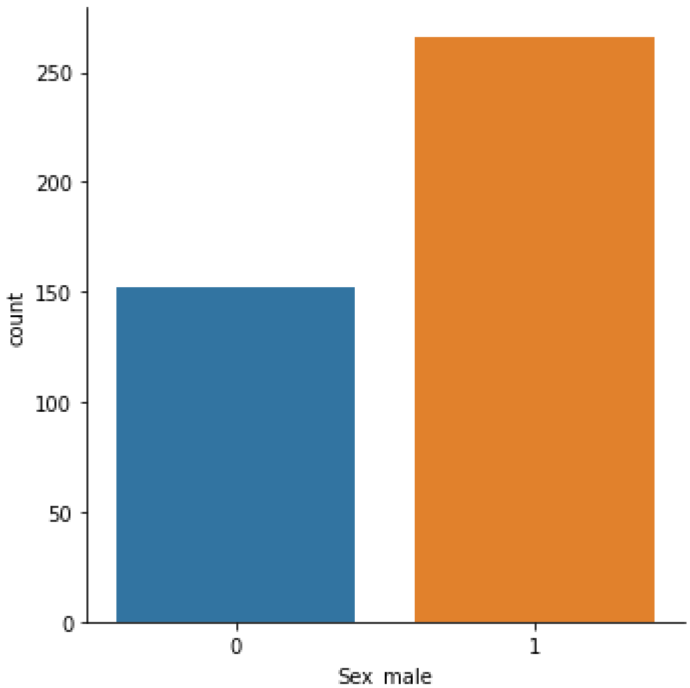
Out[55]:

```
count    418.000000
mean       0.069441
std        0.109012
min        0.000000
25%        0.015412
50%        0.028213
75%        0.061429
max        1.000000
Name: min_max_scaled_fare, dtype: float64
```

Out[55]:
```
count    418.000000
mean       0.632775
std        0.420919
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
```

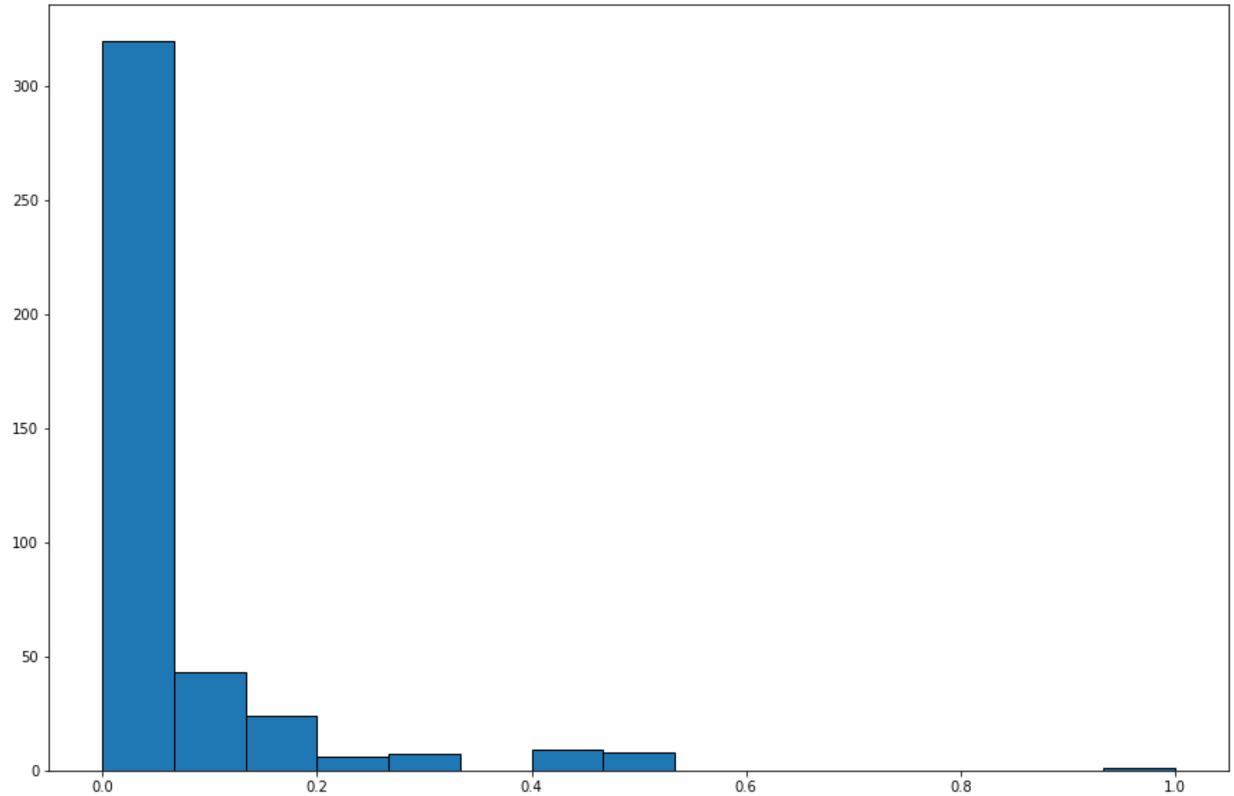
In [56]:
```
# Create visualizations for the distributions for each of our new variables

# New Indicator Variable Visualizations
sns.catplot(x = 'Sex_male', kind = 'count', data = knn_testing_df)
```

Out[56]:
```
<seaborn.axisgrid.FacetGrid at 0x1a65e352a30>
```



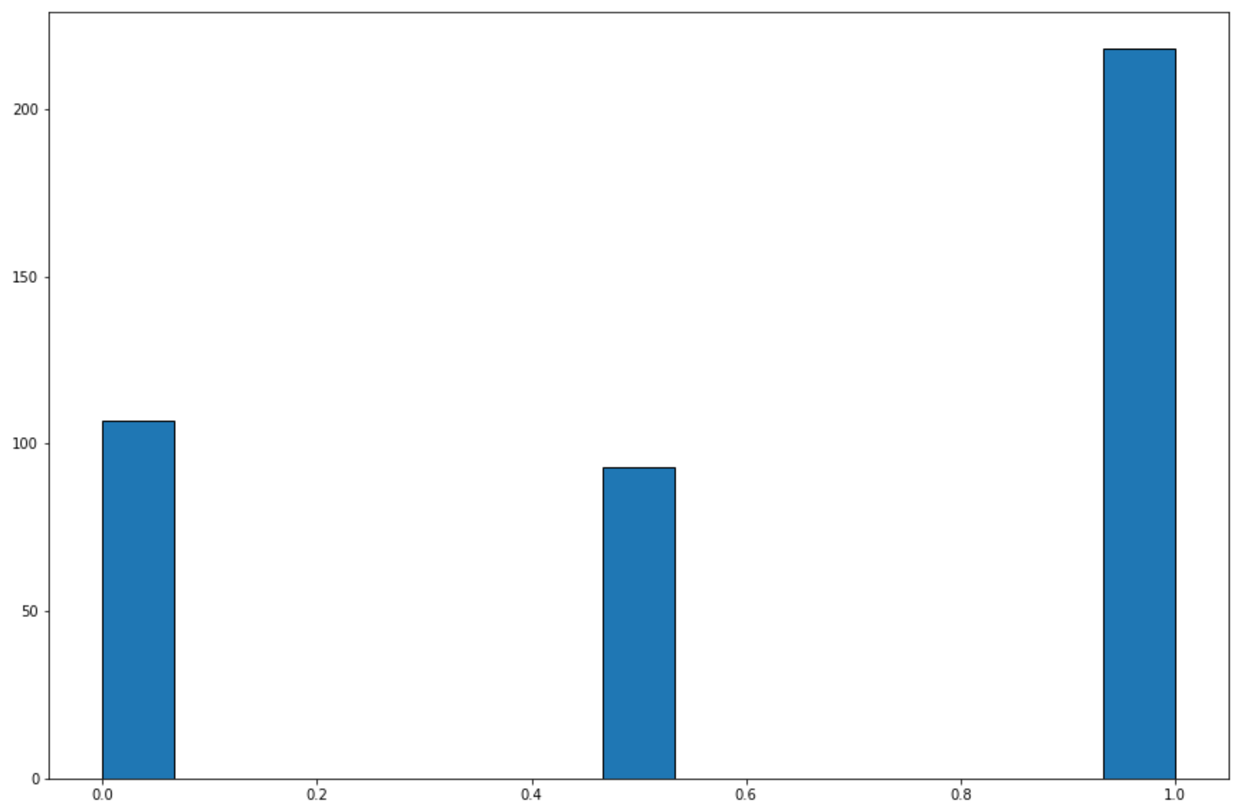
In [57]:
```
# New Numeric Variable Visualizations - min_max_scaled_fare
knn_testing_df['min_max_scaled_fare'].hist(edgecolor = 'black',
                                           bins = 15, figsize = (15, 10),
                                           grid = False)
```

Out[57]:
```
<Axes: >
```



In [58]:
```
# New Numeric Variable Visualizations - min_max_scaled_pclass
knn_testing_df['min_max_scaled_Pclass'].hist(edgecolor = 'black',
                                             bins = 15, figsize = (15, 10),
                                             grid = False)
```

Out[58]: <Axes: >



Apply the K Nearest Neighbors Model to the Testing Dataset

In [59]:
```python
# Apply the K - Nearest Model to Elicit Predictions For Testing Dataset
knn_testing_x = knn_testing_df.drop(columns=['PassengerId',
                                             'Pclass',
                                             'Name',
                                             'Age',
                                             'SibSp',
                                             'Parch',
                                             'Ticket',
                                             'Fare',
                                             'Embarked',
                                             'First_Cabin_Deck'])

knn_testing_preds_grid = gridsearch.predict(knn_testing_x)

# Put the KNN predictions into a Pandas dataframe
prediction_df_knn = pd.DataFrame(knn_testing_preds_grid, columns=['KNN_Prediction'])


# Add the PassengerId column to the front of the KNN predictions dataframe
prediction_df_knn.insert(0, 'PassengerId', knn_testing_df['PassengerId'])

# Use the Optimal Cutoff Boundary Determined Earlier to Turn Predictions into zeros ar
prediction_df_knn['Survived'] = np.where(prediction_df_knn['KNN_Prediction'] >= Optima

# Drop the KNN_Prediction Column
prediction_df_knn.drop(['KNN_Prediction'],axis=1,inplace=True)

#output predictions to csv
prediction_df_knn.to_csv('test_predictions_knn_v1.csv', index=False)
```

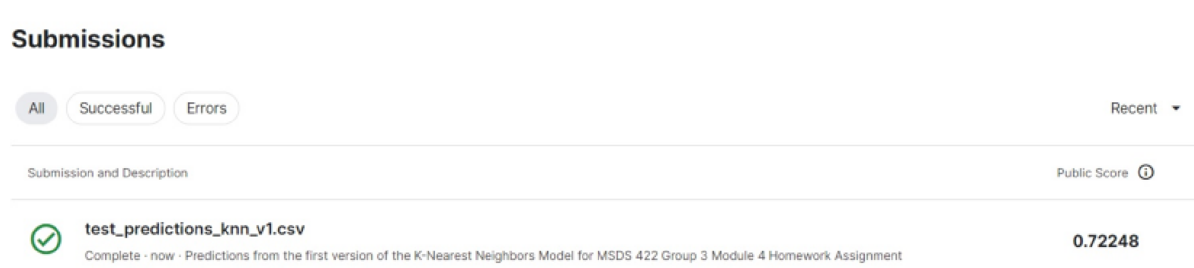
Display the kaggle results associated with the K-Nearest Neighbors Model

In [60]:
```python
# Display the kaggle results associated with the KNN Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_KNN_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[60]:
```
<Figure size 1080x1080 with 0 Axes>
```

Out[60]:
```
<matplotlib.image.AxesImage at 0x1a65dbfda60>
```

Out[60]:
```
(-0.5, 1494.5, 337.5, -0.5)
```



In [ ]:

```

```