MODULE 5: TITANIC: MACHINE LEARNING THROUGH DISASTER

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

MSDS 422: Practical Machine Learning

April 30th, 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). Tree-based methods (Random Forest, Gradient Boosted Trees, Extra Trees) were employed to examine differences in classification (prediction). Model performance metrics were 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 Cabin, Age, Embarked contained missingness. Embarked contained only two missing values, and the proportion of Embarked 'S' represents almost 75% of the data; so, we filled the two missing instances with 'S'. With regard to Cabin, more than 75% of the data contain missingness. Descriptive statistics revealed that most of the fields are comparable, with the exception of Fare. The mean Fare value across the whole dataset was about 32 (including missing cabin rows) with a top decile of over 75% survival rate, while the mean fare for just rows with missing cabin data is only 19 with top 5% and top decile survival rates of only 51% and 37%, respectively.

To explore this problem, we trained a random forest classifier on the 204 instances where cabin data is present and used our model to predict the cabin level for 600+ instances. A limitation of this approach is the stark difference in characteristics among the training and predicted instances may mean our model is an imperfect predictor. In addition, if Age is left out of the cabin data model since it contains 177 missing values, we lose training data. Two new features were created to address missingness: a dichotomous 'cabin' indicator where missing values were computed as 0, and a numeric variable to reflect each passenger's first cabin's deck. A dichotomous variable (child indicator) was also created to classify passengers under 18 as children (set to '1'). We used forward filling; however, Age had a weaker relationship to survival rate relative to our child indicator, giving credence to our approach. The proportion of survival rate for our imputed variables is a near exact match for the original dataset.

We first predicted passenger survival via a Random Forest model. For this exercise, we applied 5-fold cross-validation (CV) when fitting the random forest model to the training dataset in order to determine the optimal values for the following hyperparameters: 1) number of trees, 2) maximum features considered when splitting node, 3) maximum depth, and 4) splitting criteria. Application of the best model from the hyperparameter tuning process yielded an accuracy, precision, and recall of 0.79, 0.81, and 0.64, respectively.

Application of this model to the test dataframe yielded similar results with Kaggle reporting the random forest model's test predictions accuracy as 0.7655.

We then predicted passenger survival via gradient boosting and extreme gradient boosting (XGBoost) methods. Starting with the gradient boosting method, we used grid search with 10-fold CV to determine the optimal values for the hyperparameters. The model with the best classification rates from the hyperparameter tuning process yielded an accuracy, precision, and recall of 0.87, 0.89, and 0.74, respectively. The female indicator variable appears to be the most important predictor of survival in this model, followed by fare, passenger ticket class, and age. Kaggle reported the model's test prediction accuracy as 0.7847. XGBoost (a more regularized form of Gradient Boosting) was explored next; we utilized grid search with 10-fold CV to determine the optimal values for the hyperparameters. Overall, the tuned parameters balance the complexity of the data and prevent overfitting by using a combination of subsampling, L1 and L2 regularization techniques, and a high learning rate. Application of the hyperparameter tuning process yielded an accuracy, precision, and recall of 0.90, 0.91, and 0.84, respectively. This analysis suggested the female indicator variable was the most important predictor of survival, followed by passenger ticket class, and cabin indicator. Kaggle reported the model's survival prediction accuracy as 0.7655.

The Extra Trees methodology has been shown to be a powerful ensemble supervised machine learning method, and may perform better than Random Forests. Extra Trees can produce low variance and gives feature's importance. To first explore applications of Extra Trees, an Extra Trees regression model was employed. Data was split into predictor and outcome components as well as into training and testing sets. An Extra Trees regression model was then fit to the testing data and used to make predictions. The Extra Trees Regression application explained 21.44% of the variance ($R^2 = .2144$). Next, an Extra Trees classification model was explored. Data was split into predictor and outcome components as well as into training and testing sets, and then 10-fold CV was applied when determining the optimal values for: 1) n_estimators, 2) max_samples, and 3) min_samples_leaf. After implementing the optimal, identified values, the classification model demonstrated an accuracy, precision, and recall values of .892, .931, and .781 respectively. This model also identified 'female_indicator' as the most important feature. Application to the test data yielded similar results, with Kaggle reporting the model's test prediction accuracy as .7775.

This analysis provided additional insights about factors that may have contributed to passenger survival on the Titanic. The tree-based classification models suggest that the female indicator variable, passenger ticket class, age, and passengers with cabin data records were important predictors of survival. This suggests that males and potentially, passengers of lower social class were less likely to survive on the Titanic. All of the tree models that were constructed had the female indicator variable as the most important predictor of survival. Results from the Kaggle submissions reveal that accuracy rates for all the tree models were fairly similar, with the Gradient Boosting model performing slightly better with a classification accuracy of 78%. Some basic parameter tuning helped to promote a balance between capturing complexity of the data and preventing overfitting, but some additional feature engineering and more computationally expensive tuning could prove useful in improving classification accuracy for survival of passengers on the Titanic.

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 [73]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Import Data

In [74]:
```
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[74]:

	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[74]:
```
(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 [75]:
```
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[75]:

	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[75]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Fare'}>],
       [<AxesSubplot:title={'center':'SibSp'}>,
        <AxesSubplot:title={'center':'Parch'}>]], dtype=object)
```

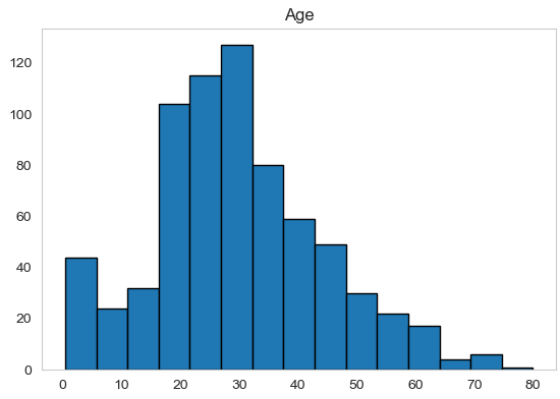
Out[75]:
```
<seaborn.axisgrid.FacetGrid at 0x162da782610>
```

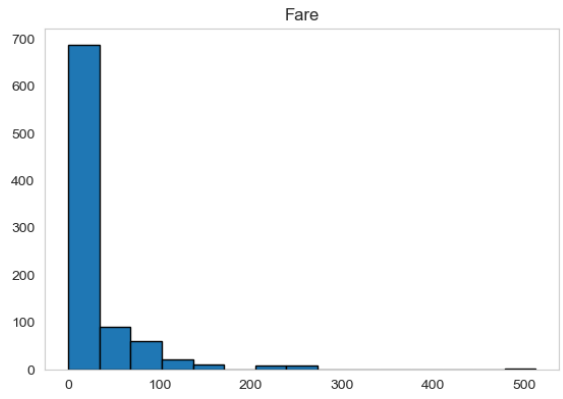
Out[75]:
```
<AxesSubplot:title={'center':'Pclass'}>
```

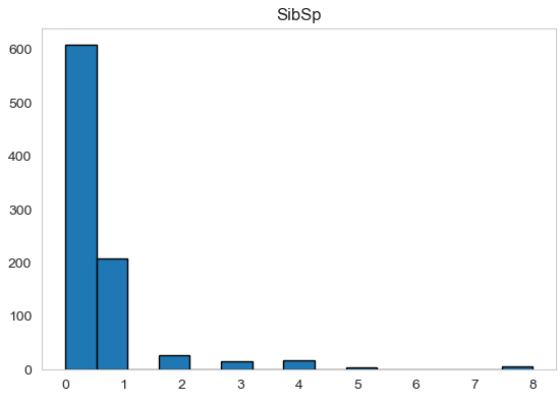
Out[75]:
```
<AxesSubplot:title={'center':'Embarked'}>
```

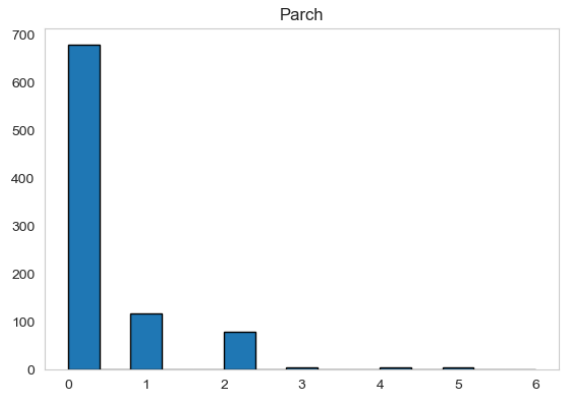
Out[75]:
```
<AxesSubplot:title={'center':'Cabin'}>
```

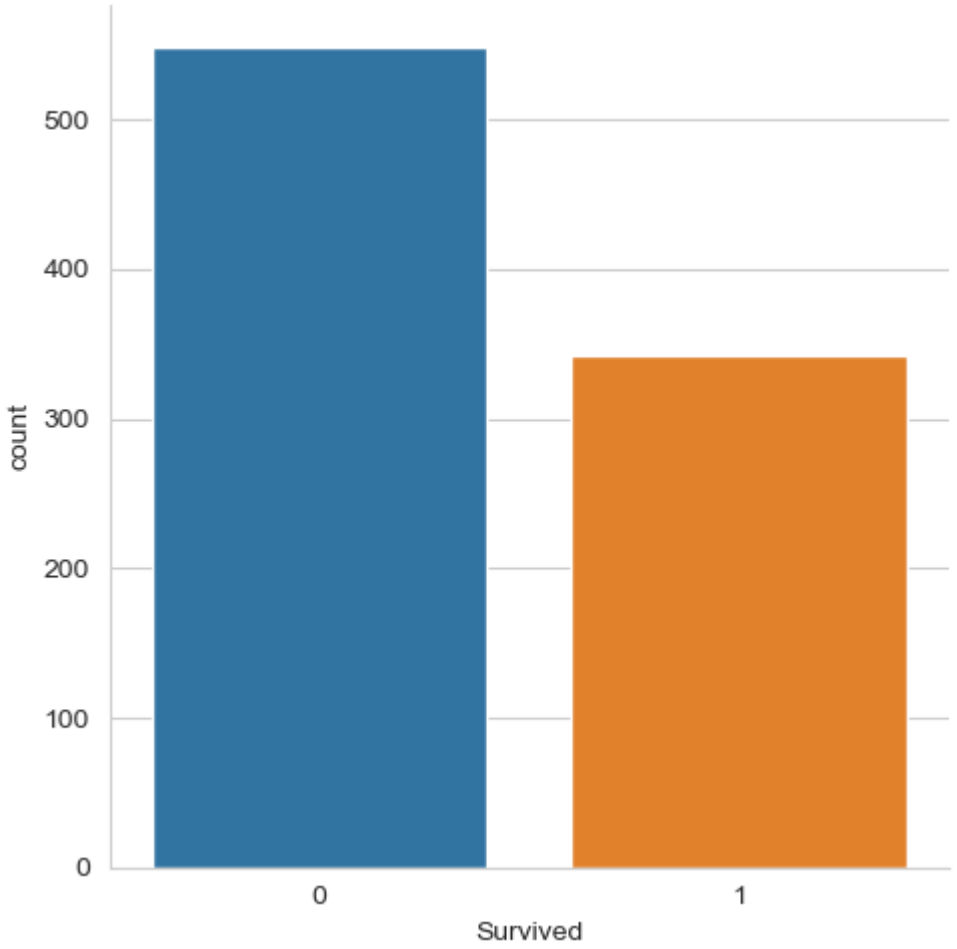
Out[75]:
```
<AxesSubplot:title={'center':'Sex'}>
```

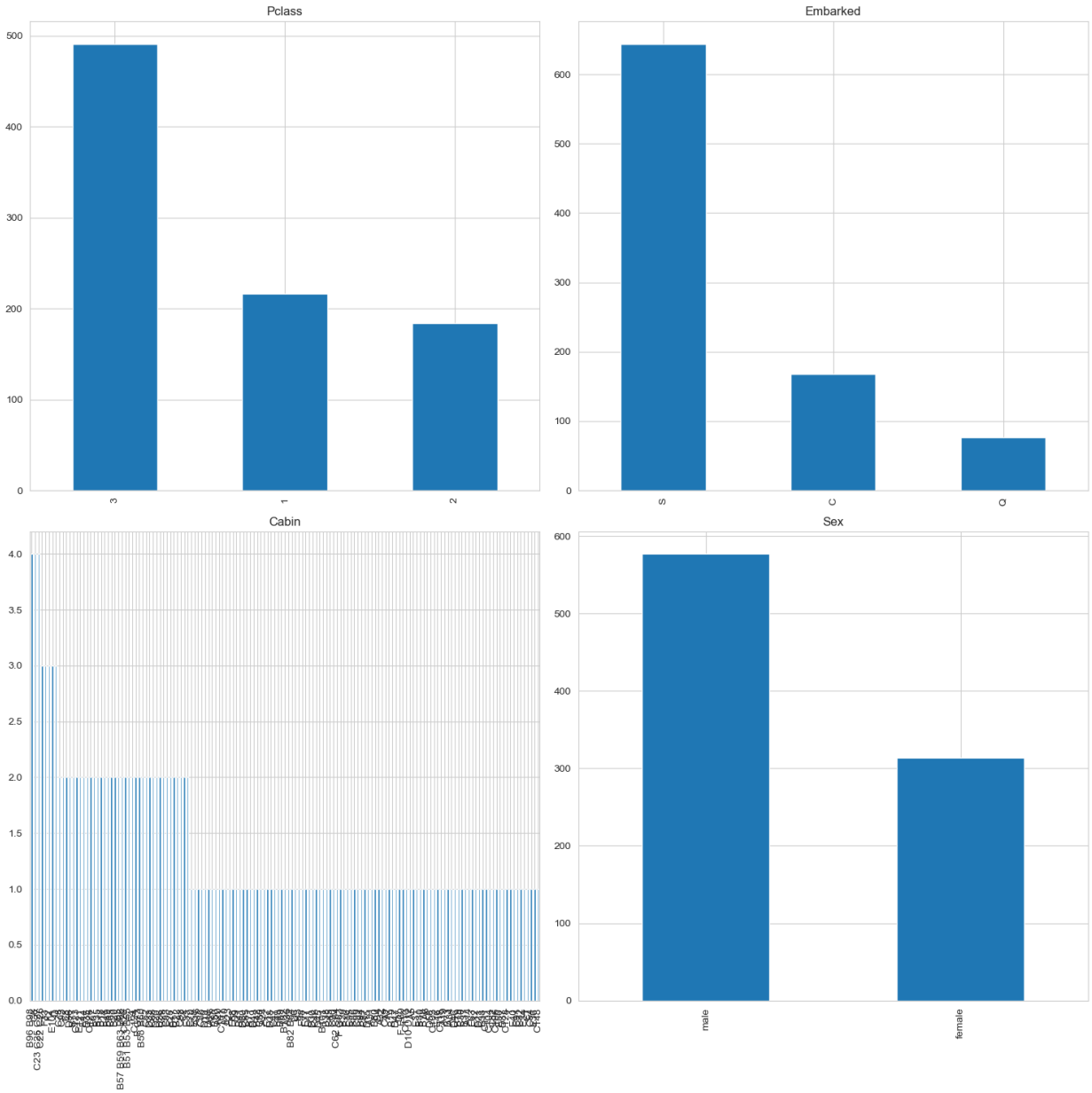









500
400
300
200
100
0
count
0
1
Survived



Investigation of Missing Data and Outliers

In [76]:
```
# 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[76]:

	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 [77]:
```
# 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)

# 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[

# Create a variable for deck associated with the passenger's first listed cabin
titanic_training_data_cleaned['First_Cabin_Deck'] = titanic_training_data_cleaned['Cab

def conditions(s):
    if (s['First_Cabin_Deck'] == "A"):
        return 1
    elif (s['First_Cabin_Deck'] == "B"):
        return 2
    elif (s['First_Cabin_Deck'] == "C"):
        return 3
    elif (s['First_Cabin_Deck'] == "D"):
        return 4
    elif (s['First_Cabin_Deck'] == "E"):
        return 5
    elif (s['First_Cabin_Deck'] == "F"):
        return 6
    elif (s['First_Cabin_Deck'] == "G"):
        return 7
    elif (s['First_Cabin_Deck'] == "T"):
        return 10
    elif (s['First_Cabin_Deck'] == "n"):
        return np.nan
    else:
        return np.nan

titanic_training_data_cleaned['Deck'] = titanic_training_data_cleaned.apply(conditions

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


# Apply Imputation to Fill In Null Values for Missing Data Values
titanic_training_data_cleaned.fillna(method='ffill', inplace=True)
titanic_training_data_cleaned.fillna(method='bfill', inplace=True)

# 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
```

In [78]:
```
titanic_training_data_cleaned['Survived'].value_counts()
```

Out[78]:
```
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 [79]:
```
# 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[79]:

	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	35.0	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	2.0	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	31.0	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 [80]:
```
# 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[80]:

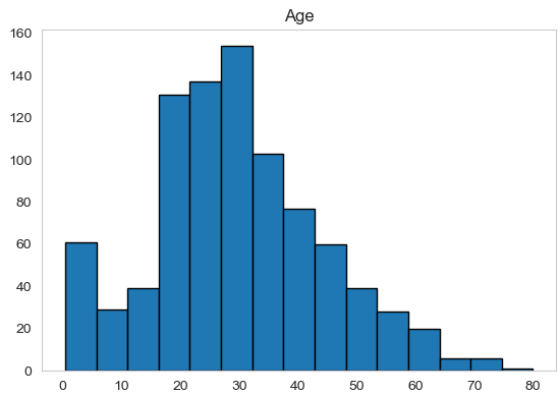
	Missing Count	Percentage Missing	Column Type

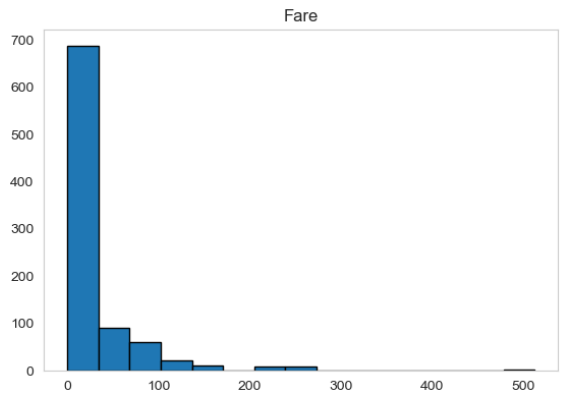
In [81]:
```
# Update our definitions of the indicator, numeric, and categorical variables to refle
numeric_variables = ['Age', 'Fare', 'SibSp', 'Parch']

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

categorical_variables = ['Pclass', 'Embarked', '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()
```

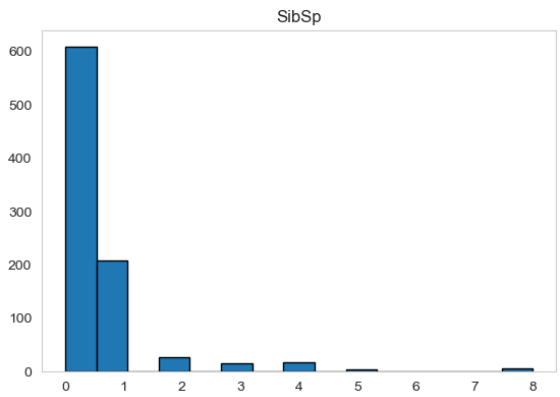
Out[81]:

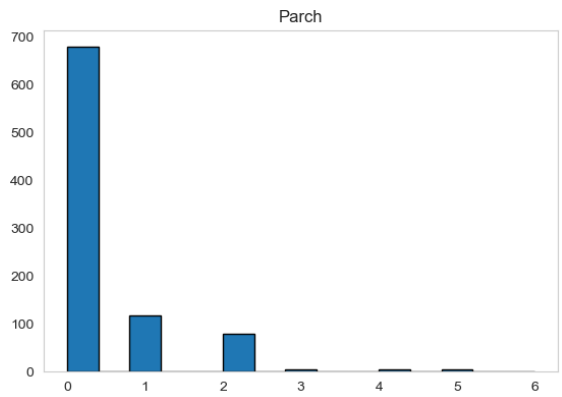
	Age	Fare	SibSp	Parch
count	891.00000	891.000000	891.000000	891.000000
mean	29.58156	32.204208	0.523008	0.381594
std	14.55459	49.693429	1.102743	0.806057
min	0.42000	0.000000	0.000000	0.000000
25%	20.00000	7.910400	0.000000	0.000000
50%	28.00000	14.454200	0.000000	0.000000
75%	38.00000	31.000000	1.000000	0.000000
max	80.00000	512.329200	8.000000	6.000000

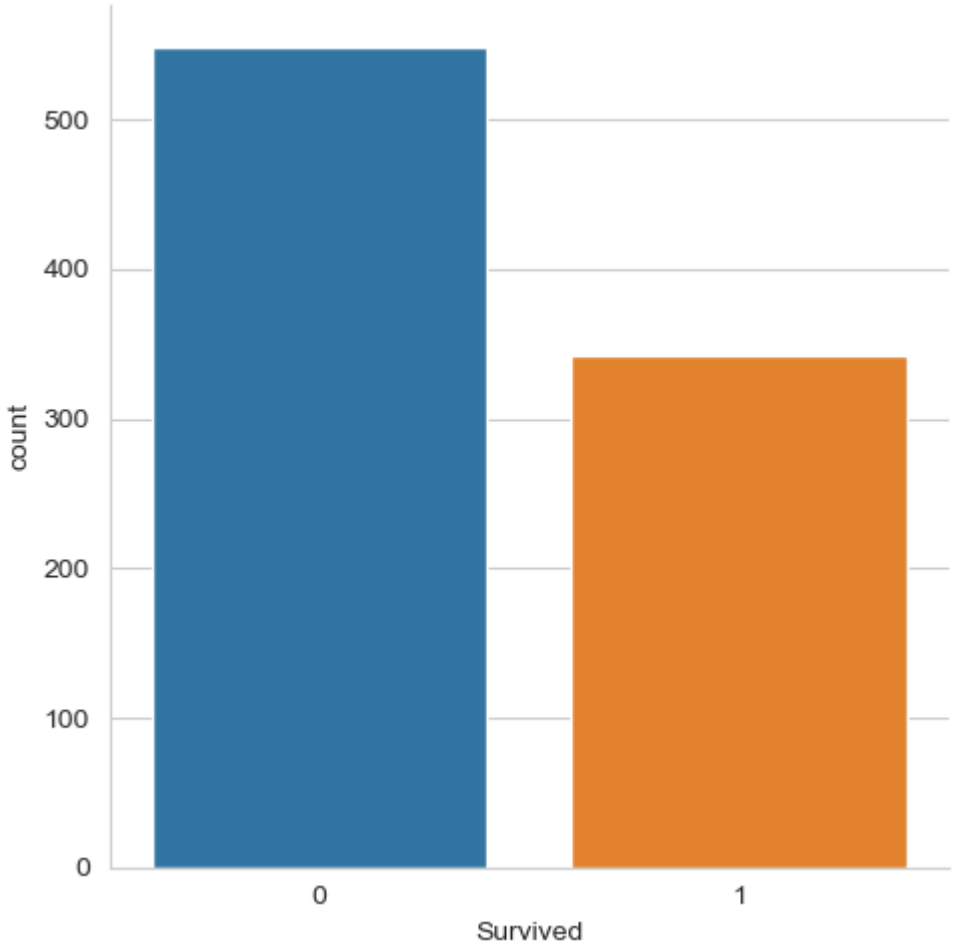
Out[81]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Fare'}>],
       [<AxesSubplot:title={'center':'SibSp'}>,
        <AxesSubplot:title={'center':'Parch'}>]], dtype=object)
```
Out[81]:
```
<seaborn.axisgrid.FacetGrid at 0x162e28ba850>
```
Out[81]:
```
<seaborn.axisgrid.FacetGrid at 0x162e072ebb0>
```
Out[81]:
```
<seaborn.axisgrid.FacetGrid at 0x162e0bc2550>
```
Out[81]:
```
<AxesSubplot:title={'center':'Pclass'}>
```
Out[81]:
```
<AxesSubplot:title={'center':'Embarked'}>
```
Out[81]:
```
<AxesSubplot:title={'center':'Deck'}>
```
Out[81]:
```
<AxesSubplot:title={'center':'Sex'}>
```

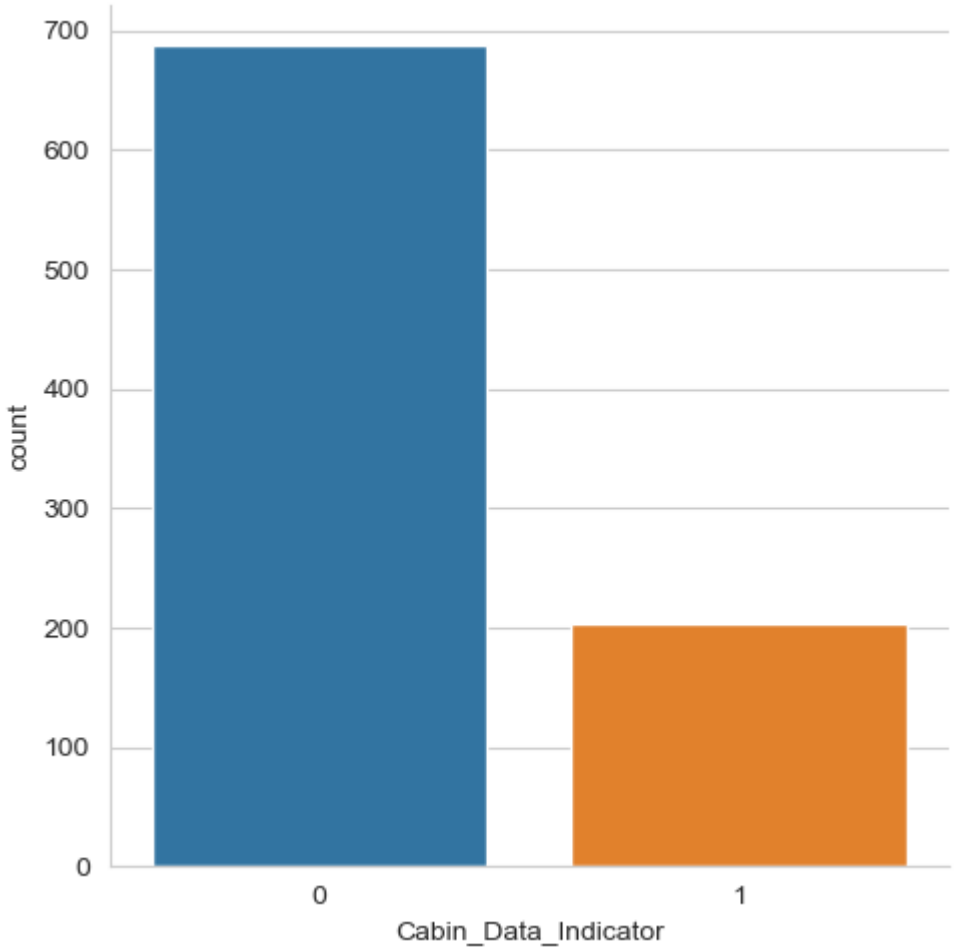


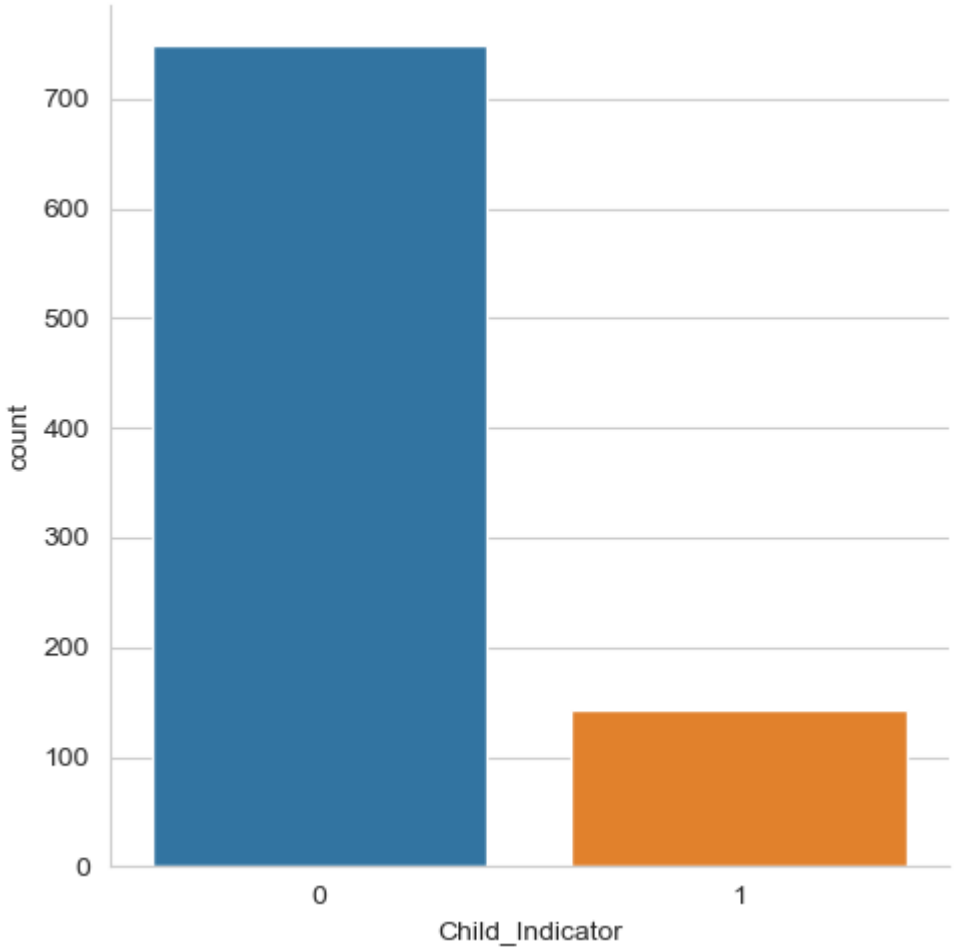


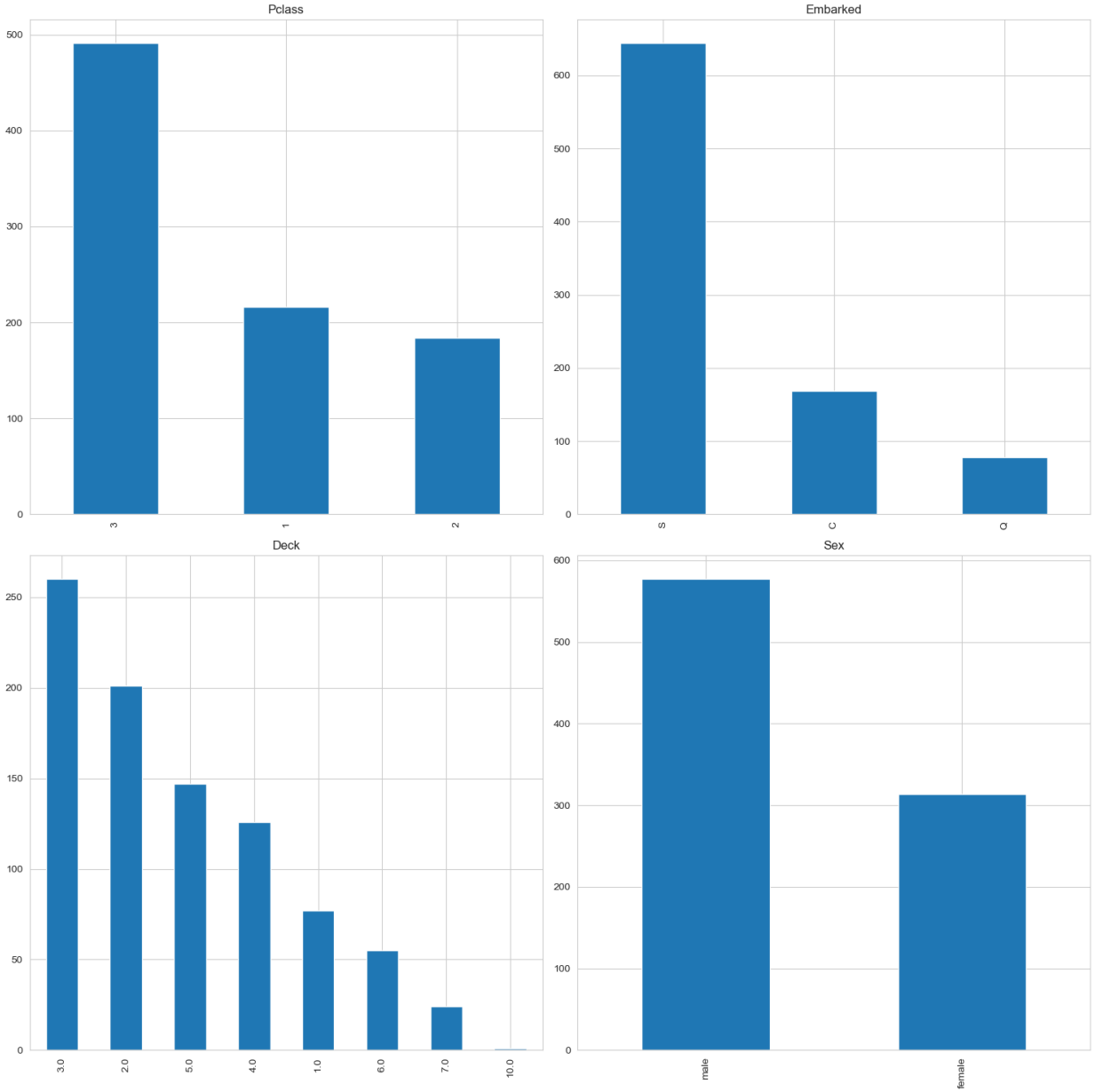





500
400
300
200
100
0
count
0
1
Survived


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


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



In [82]:
```
# Explore cabin values

cabins = titanic_training_data['Cabin'].dropna().tolist()
letters = []
for cabin in cabins:
    letters.append(cabin[0])
np.unique(letters)

# Explore cabin missing values

cabin_eda = titanic_training_data[titanic_training_data['Cabin'].isnull()].reset_index
cabin_eda.describe()
```

Out[82]:
```
array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'T'], dtype='<U1')
```

Out[82]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	687.000000	687.000000	687.000000	529.000000	687.000000	687.000000	687.000000
mean	443.208151	0.299854	2.639010	27.555293	0.547307	0.365357	19.157325
std	259.215905	0.458528	0.589602	13.472634	1.207492	0.827106	28.663343
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	214.500000	0.000000	2.000000	19.000000	0.000000	0.000000	7.877100
50%	441.000000	0.000000	3.000000	26.000000	0.000000	0.000000	10.500000
75%	664.500000	1.000000	3.000000	35.000000	1.000000	0.000000	23.000000
max	891.000000	1.000000	3.000000	74.000000	8.000000	6.000000	512.329200

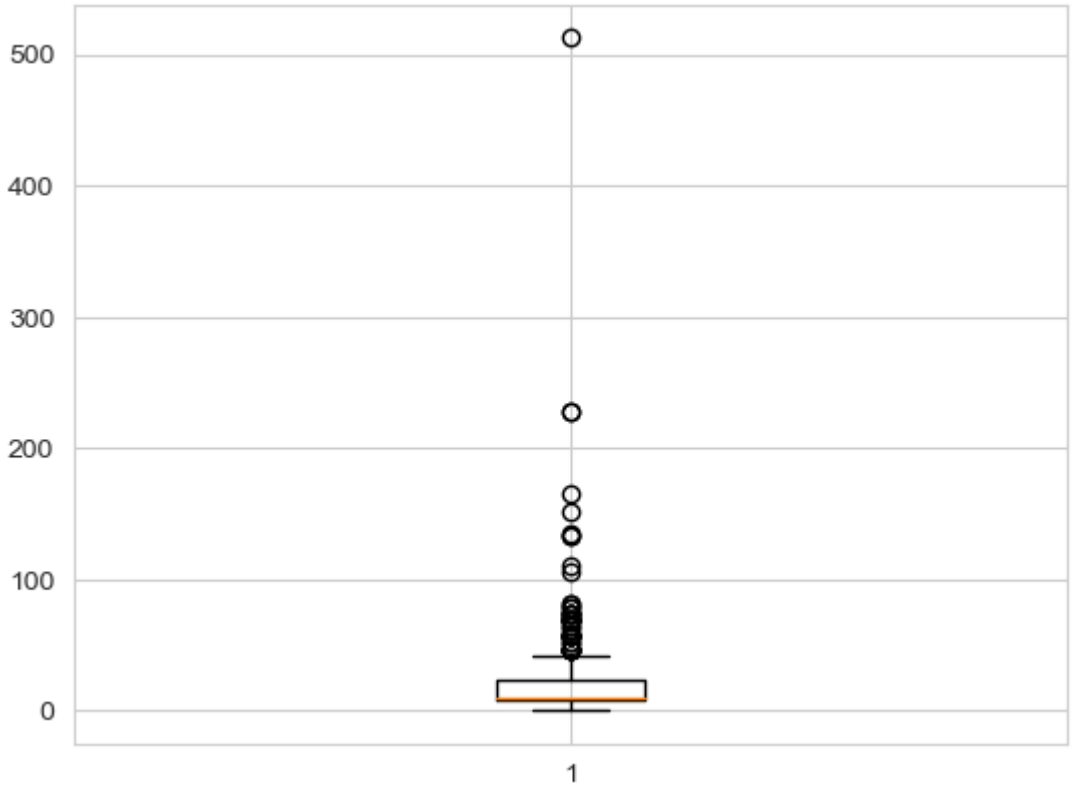
In [83]:
```
# Compare stats versus whole dataset
titanic_training_data.describe()
```

Out[83]:

	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 [84]:
```
# do the fare outliers survive?
plt.boxplot(cabin_eda['Fare'])
```

Out[84]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x162e2879070>,
  <matplotlib.lines.Line2D at 0x162e2879250>],
 'caps': [<matplotlib.lines.Line2D at 0x162e2871160>,
  <matplotlib.lines.Line2D at 0x162e2871310>],
 'boxes': [<matplotlib.lines.Line2D at 0x162e28798e0>],
 'medians': [<matplotlib.lines.Line2D at 0x162e2871d00>],
 'fliers': [<matplotlib.lines.Line2D at 0x162e2871ee0>],
 'means': []}
```



In [85]:
```
# Fare percentiles (5%): whole data set
np.percentile(titanic_training_data['Fare'], np.arange(0, 100, 5))

# Fare percentiles (5%): missing cabin data
np.percentile(cabin_eda['Fare'], np.arange(0, 100, 5))
```

Out[85]:
```
array([  0.     ,   7.225  ,   7.55   ,   7.75   ,   7.8542 ,   7.9104 ,
         8.05   ,   9.     ,  10.5    ,  13.     ,  14.4542 ,  16.1    ,
        21.6792 ,  26.     ,  27.     ,  31.     ,  39.6875 ,  56.4958 ,
        77.9583 , 112.07915])
```

Out[85]:
```
array([ 0.     ,  7.125  ,  7.25   ,  7.75   ,  7.775  ,  7.8771 ,
        7.8958 ,  8.05   ,  8.05   ,  9.4375 , 10.5    , 13.     ,
       13.945  , 15.5    , 18.88166, 23.     , 26.     , 27.765  ,
       33.     , 56.4958 ])
```

In [86]:
```
# check IQR for original data
q3, q1 = np.percentile(titanic_training_data['Fare'], [75 ,25])
iqr = q3 - q1
outlier_threshold = q3 + 1.5*iqr
outlier_threshold

# check IQR for missing cabin data
q3, q1 = np.percentile(cabin_eda['Fare'], [75 ,25])
iqr = q3 - q1
outlier_threshold = q3 + 1.5*iqr
outlier_threshold
```

Out[86]:
```
65.6344
```

Out[86]:
```
45.684349999999995
```

In [87]:
```
## full training set
print('SURVIVAL RATE FOR DIFFERENT FARE LEVELS')
```

```
print('survival rate mean:',np.mean(titanic_training_data['Fare']))
print('survival rate top quartile:',np.mean(titanic_training_data[titanic_training_dat
print('survival rate top 10%:',np.mean(titanic_training_data[titanic_training_data['Fa
print('survival rate top 5%:',np.mean(titanic_training_data[titanic_training_data['Far
print('survival rate all ourliers (1.5iqr):',np.mean(titanic_training_data[titanic_tra
print('')
## missing cabin values
print('COMPARE WITH VALUES MISSING CABIN DATA')
print('survival rate mean:',np.mean(cabin_eda['Fare']))
print('survival rate top quartile:',np.mean(cabin_eda[cabin_eda['Fare'] > 23]['Survive
print('survival rate top 10%:',np.mean(cabin_eda[cabin_eda['Fare'] > 33]['Survived']))
print('survival rate top 5%:',np.mean(cabin_eda[cabin_eda['Fare'] > 56]['Survived']))
print('survival rate ourliers:',np.mean(cabin_eda[cabin_eda['Fare'] > 46]['Survived']))
```

```
SURVIVAL RATE FOR DIFFERENT FARE LEVELS
survival rate mean: 32.2042079685746
survival rate top quartile: 0.581081081081081
survival rate top 10%: 0.7586206896551724
survival rate top 5%: 0.7555555555555555
survival rate all ourliers (1.5iqr): 0.6810344827586207

COMPARE WITH VALUES MISSING CABIN DATA
survival rate mean: 19.1573253275109
survival rate top quartile: 0.38011695906432746
survival rate top 10%: 0.36764705882352944
survival rate top 5%: 0.5135135135135135
survival rate ourliers: 0.425531914893617
```

In [88]:

```
plt.subplot(1,2,1)
plt.hist(cabin_eda['Fare'], bins = 100)
plt.title('missing cabin data')

plt.subplot(1,2,2)
plt.hist(titanic_training_data['Fare'], bins = 100)
plt.title('complete data set')
```

Out[88]:

```
<AxesSubplot:>
```

Out[88]:
```
(array([ 13., 315., 114.,  51.,  41.,  68.,  21.,  13.,   4.,   8.,   2.,
          9.,   2.,   7.,   5.,   2.,   2.,   0.,   0.,   0.,   1.,   1.,
          0.,   0.,   0.,   0.,   3.,   0.,   0.,   1.,   0.,   0.,   1.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          2.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          1.]),
 array([  0.      ,   5.123292,  10.246584,  15.369876,  20.493168,
         25.61646 ,  30.739752,  35.863044,  40.986336,  46.109628,
         51.23292 ,  56.356212,  61.479504,  66.602796,  71.726088,
         76.84938 ,  81.972672,  87.095964,  92.219256,  97.342548,
        102.46584 , 107.589132, 112.712424, 117.835716, 122.959008,
        128.0823  , 133.205592, 138.328884, 143.452176, 148.575468,
        153.69876 , 158.822052, 163.945344, 169.068636, 174.191928,
        179.31522 , 184.438512, 189.561804, 194.685096, 199.808388,
        204.93168 , 210.054972, 215.178264, 220.301556, 225.424848,
        230.54814 , 235.671432, 240.794724, 245.918016, 251.041308,
        256.1646  , 261.287892, 266.411184, 271.534476, 276.657768,
        281.78106 , 286.904352, 292.027644, 297.150936, 302.274228,
        307.39752 , 312.520812, 317.644104, 322.767396, 327.890688,
        333.01398 , 338.137272, 343.260564, 348.383856, 353.507148,
        358.63044 , 363.753732, 368.877024, 374.000316, 379.123608,
        384.2469  , 389.370192, 394.493484, 399.616776, 404.740068,
        409.86336 , 414.986652, 420.109944, 425.233236, 430.356528,
        435.47982 , 440.603112, 445.726404, 450.849696, 455.972988,
        461.09628 , 466.219572, 471.342864, 476.466156, 481.589448,
        486.71274 , 491.836032, 496.959324, 502.082616, 507.205908,
        512.3292  ]),
 <BarContainer object of 100 artists>)
```

Out[88]:
```
Text(0.5, 1.0, 'missing cabin data')
```

Out[88]:
```
<AxesSubplot:>
```

Out[88]:
```
(array([ 17., 320., 129.,  53.,  43., 104.,  31.,  19.,   4.,  12.,  23.,
         16.,   6.,  12.,  10.,  19.,  10.,   8.,   2.,   0.,   2.,   6.,
          3.,   4.,   0.,   0.,   7.,   0.,   2.,   7.,   0.,   0.,   2.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   4.,   0.,   1.,
          4.,   0.,   0.,   0.,   2.,   0.,   0.,   6.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          3.]),
 array([  0.      ,   5.123292,  10.246584,  15.369876,  20.493168,
         25.61646 ,  30.739752,  35.863044,  40.986336,  46.109628,
         51.23292 ,  56.356212,  61.479504,  66.602796,  71.726088,
         76.84938 ,  81.972672,  87.095964,  92.219256,  97.342548,
        102.46584 , 107.589132, 112.712424, 117.835716, 122.959008,
        128.0823  , 133.205592, 138.328884, 143.452176, 148.575468,
        153.69876 , 158.822052, 163.945344, 169.068636, 174.191928,
        179.31522 , 184.438512, 189.561804, 194.685096, 199.808388,
        204.93168 , 210.054972, 215.178264, 220.301556, 225.424848,
        230.54814 , 235.671432, 240.794724, 245.918016, 251.041308,
        256.1646  , 261.287892, 266.411184, 271.534476, 276.657768,
        281.78106 , 286.904352, 292.027644, 297.150936, 302.274228,
        307.39752 , 312.520812, 317.644104, 322.767396, 327.890688,
        333.01398 , 338.137272, 343.260564, 348.383856, 353.507148,
        358.63044 , 363.753732, 368.877024, 374.000316, 379.123608,
        384.2469  , 389.370192, 394.493484, 399.616776, 404.740068,
        409.86336 , 414.986652, 420.109944, 425.233236, 430.356528,
        435.47982 , 440.603112, 445.726404, 450.849696, 455.972988,
        461.09628 , 466.219572, 471.342864, 476.466156, 481.589448,
        486.71274 , 491.836032, 496.959324, 502.082616, 507.205908,
        512.3292  ]),
 <BarContainer object of 100 artists>)
```

Out[88]:
```
Text(0.5, 1.0, 'complete data set')
```

In [89]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

# Embarked
len(titanic_training_data[titanic_training_data['Embarked']=='S']) / len(titanic_trair
len(titanic_training_data[titanic_training_data['Embarked']=='C']) / len(titanic_trair
len(titanic_training_data[titanic_training_data['Embarked']=='Q']) / len(titanic_trair

len(titanic_training_data[titanic_training_data['Embarked'].isnull()]) # only two inst

df = titanic_training_data.drop(columns = ['Name','Age','Cabin','PassengerId'])
df.fillna(value = 'S',inplace = True)

le.fit(df['Embarked'])#.reshape(-1,1)
df['Embarked'] = le.transform(df['Embarked'])

le.fit(df['Sex'])#.reshape(-1,1)
df['Sex'] = le.transform(df['Sex'])
```

Out[89]:
```
0.7227833894500562
```

Out[89]:
```
0.18855218855218855
```

Out[89]:
```
0.08641975308641975
```

Out[89]:
```
2
```

Out[89]:
```
LabelEncoder()
```

Out[89]:
```
LabelEncoder()
```

In [90]:
```
y = titanic_training_data['Cabin']
X = df
```

```
df['Cabin'] = y
df_cabin = df['Cabin'].to_list()
```

In [91]:
```
letters = []
for i in range(0,len(df)):
    if df_cabin[i] is not None:
        letters.append(str(df_cabin[i])[0])
    else:
        letters.append(np.nan)

df['cabin_letter'] = letters
le.fit(df['cabin_letter'])#.reshape(-1,1)
df['cabin_code'] = le.transform(df['cabin_letter'])

df.head()
len(df)
len(df.dropna())
```

Out[91]:
```
LabelEncoder()
```

Out[91]:

	Survived	Pclass	Sex	SibSp	Parch	Ticket	Fare	Embarked	Cabin	cabin_letter	cabin_code
0	0	3	1	1	0	A/5 21171	7.2500	2	NaN	n	8
1	1	1	0	1	0	PC 17599	71.2833	0	C85	C	2
2	1	3	0	0	0	STON/O2. 3101282	7.9250	2	NaN	n	8
3	1	1	0	1	0	113803	53.1000	2	C123	C	2
4	0	3	1	0	0	373450	8.0500	2	NaN	n	8

Out[91]:
```
891
```

Out[91]:
```
204
```

In [92]:
```
#df['Cabin'] = y
null_count = df.isnull().sum()
df2 = df.dropna().reset_index().drop(columns = 'index')
null_count
```

Out[92]:
```
Survived          0
Pclass            0
Sex               0
SibSp             0
Parch             0
Ticket            0
Fare              0
Embarked          0
Cabin           687
cabin_letter      0
cabin_code        0
dtype: int64
```

In [93]:
```
df3 = df2.drop(columns = ['Cabin'])

import numpy as np
```

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(df3[['Survived', 'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare

from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(df3[['Survived', 'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare', 'Embarked']],df3['
#reg.predict(np.array([[3, 5]]))
```

Out[93]:
```
RandomForestClassifier(max_depth=2, random_state=0)
```

In [94]:
```
le.fit(cabin_eda['Embarked'])#.reshape(-1,1)
cabin_eda['Embarked'] = le.transform(cabin_eda['Embarked'])

le.fit(cabin_eda['Sex'])#.reshape(-1,1)
cabin_eda['Sex'] = le.transform(cabin_eda['Sex'])

#cabin_eda.head()
#df3.head()

predicted_cabin_linear = reg.predict(cabin_eda[['Survived', 'Pclass', 'Sex', 'SibSp',
cabin_eda['predicted_cabin_linear'] = predicted_cabin_linear

predicted_cabin_rf = clf.predict(cabin_eda[['Survived', 'Pclass', 'Sex', 'SibSp', 'Par
cabin_eda['predicted_cabin_rf'] = predicted_cabin_rf
```

Out[94]:
```
LabelEncoder()
```

Out[94]:
```
LabelEncoder()
```

In [95]:
```
df2.head()
cabin_eda.head()
df
```

Out[95]:

	Survived	Pclass	Sex	SibSp	Parch	Ticket	Fare	Embarked	Cabin	cabin_letter	cabin_code
0	1	1	0	1	0	PC 17599	71.2833	0	C85	C	2
1	1	1	0	1	0	113803	53.1000	2	C123	C	2
2	0	1	1	0	0	17463	51.8625	2	E46	E	4
3	1	3	0	1	1	PP 9549	16.7000	2	G6	G	6
4	1	1	0	0	0	113783	26.5500	2	C103	C	2

Out[95]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	
1	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
2	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	
3	6	0	3	Moran, Mr. James	1	NaN	0	0	330877	8.4583	NaN	
4	8	0	3	Palsson, Master. Gosta Leonard	1	2.0	3	1	349909	21.0750	NaN	

Out[95]:

	Survived	Pclass	Sex	SibSp	Parch	Ticket	Fare	Embarked	Cabin	cabin_letter	cabin_cc
0	0	3	1	1	0	A/5 21171	7.2500	2	NaN	n	
1	1	1	0	1	0	PC 17599	71.2833	0	C85	C	
2	1	3	0	0	0	STON/O2. 3101282	7.9250	2	NaN	n	
3	1	1	0	1	0	113803	53.1000	2	C123	C	
4	0	3	1	0	0	373450	8.0500	2	NaN	n	
...	...	...	...	...	...	...	...	...	...	...	
886	0	2	1	0	0	211536	13.0000	2	NaN	n	
887	1	1	0	0	0	112053	30.0000	2	B42	B	
888	0	3	0	1	2	W./C. 6607	23.4500	2	NaN	n	
889	1	1	1	0	0	111369	30.0000	0	C148	C	
890	0	3	1	0	0	370376	7.7500	1	NaN	n	

891 rows × 11 columns

In [96]:

```
predictions = []
for i in range(0,len(df)):
    temp = df[['Survived', 'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare', 'Embarked']]
    pred = clf.predict(temp)
    predictions.append(pred)
df['predicted_cabin'] = predictions

new_cabin_series = []
for i in range(0,len(df)):
    if df.loc[i,'cabin_letter'] == 'n':
        new_cabin_series.append(df.loc[i,'predicted_cabin'])
```

```
    else:
        new_cabin_series.append(df.loc[i,'cabin_letter'])
```

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 [97]:
```
# 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[97]:
```
<AxesSubplot:xlabel='Survived', ylabel='Age'>
```

Out[97]:
```
<AxesSubplot:xlabel='Survived', ylabel='Fare'>
```

Out[97]: <AxesSubplot:xlabel='Survived', ylabel='SibSp'>

Out[97]: <AxesSubplot:xlabel='Survived', ylabel='Parch'>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e28993a0>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e28993a0>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e1258520>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e1258520>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e142f880>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e142f880>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e12381c0>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e12381c0>

Out[97]: <seaborn.axisgrid.FacetGrid at 0x162db1588e0>

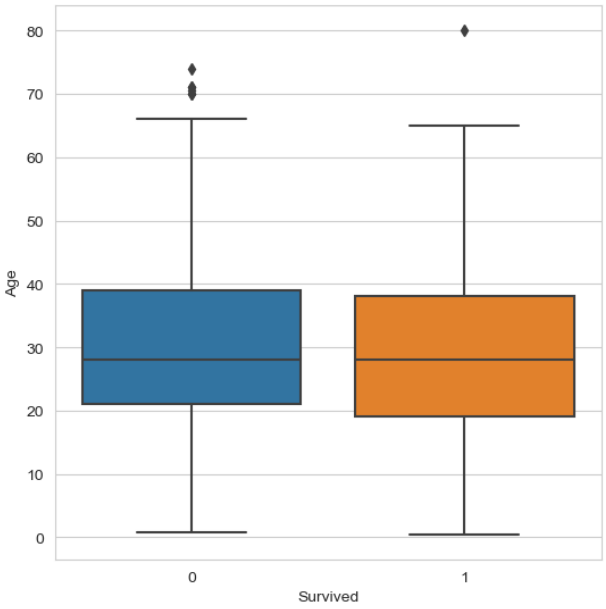
Out[97]: <seaborn.axisgrid.FacetGrid at 0x162db1588e0>

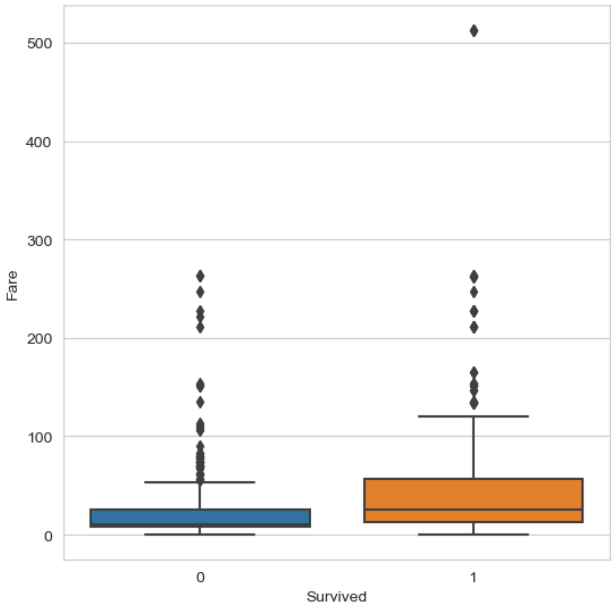
Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e142f880>

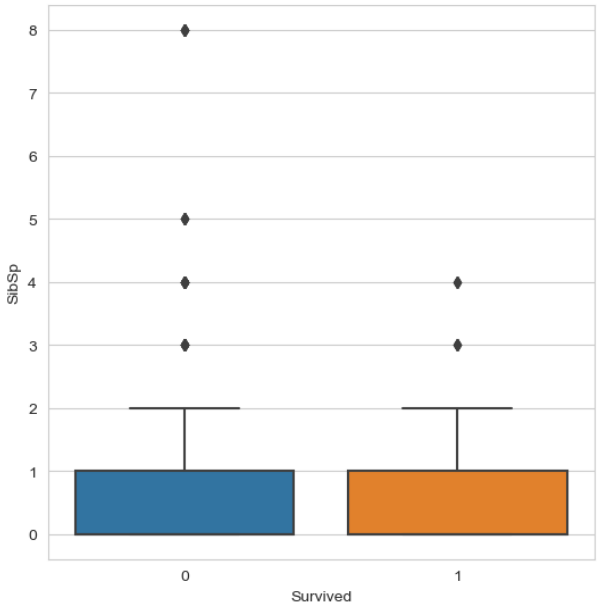
Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e142f880>

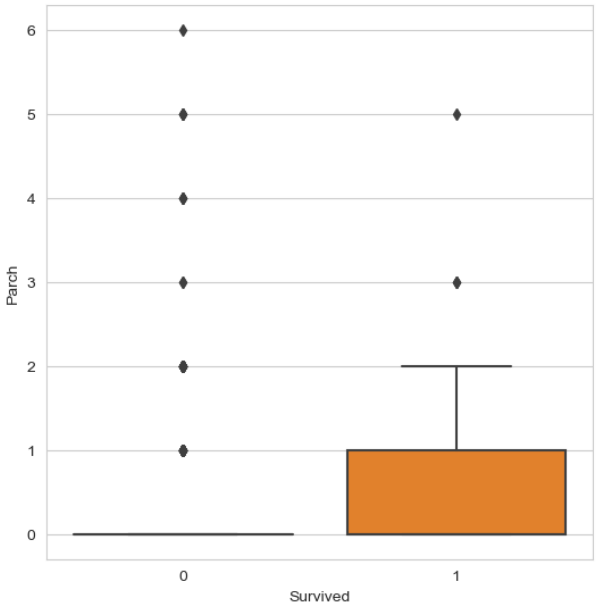
Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e0809280>

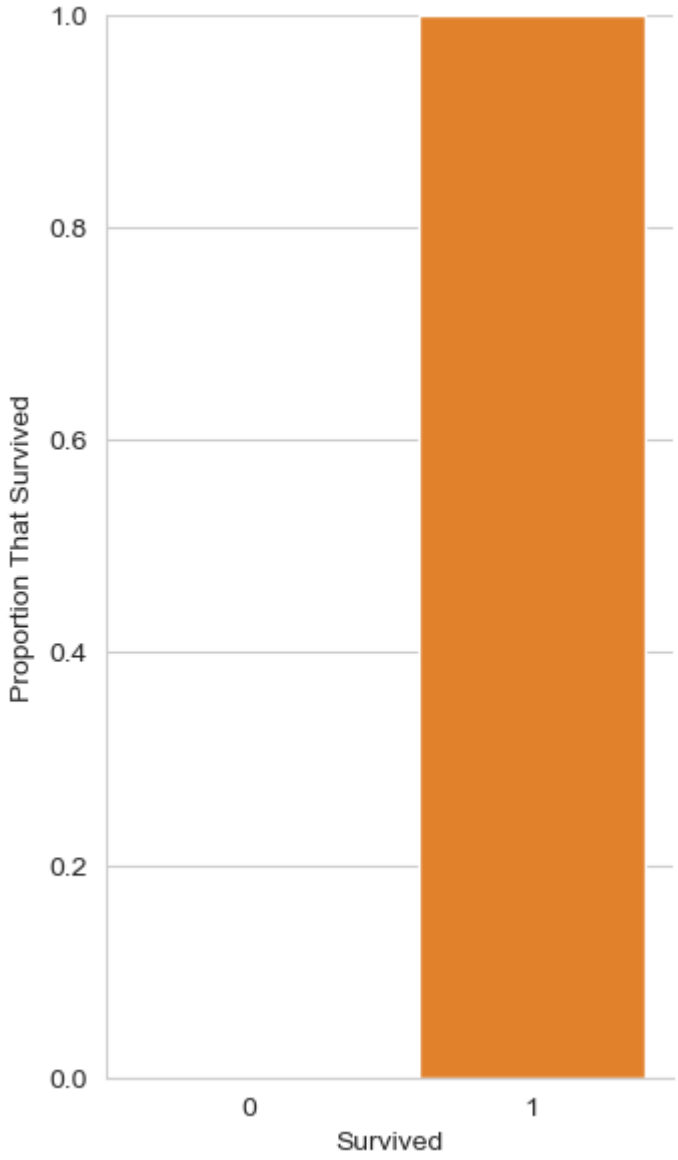
Out[97]: <seaborn.axisgrid.FacetGrid at 0x162e0809280>

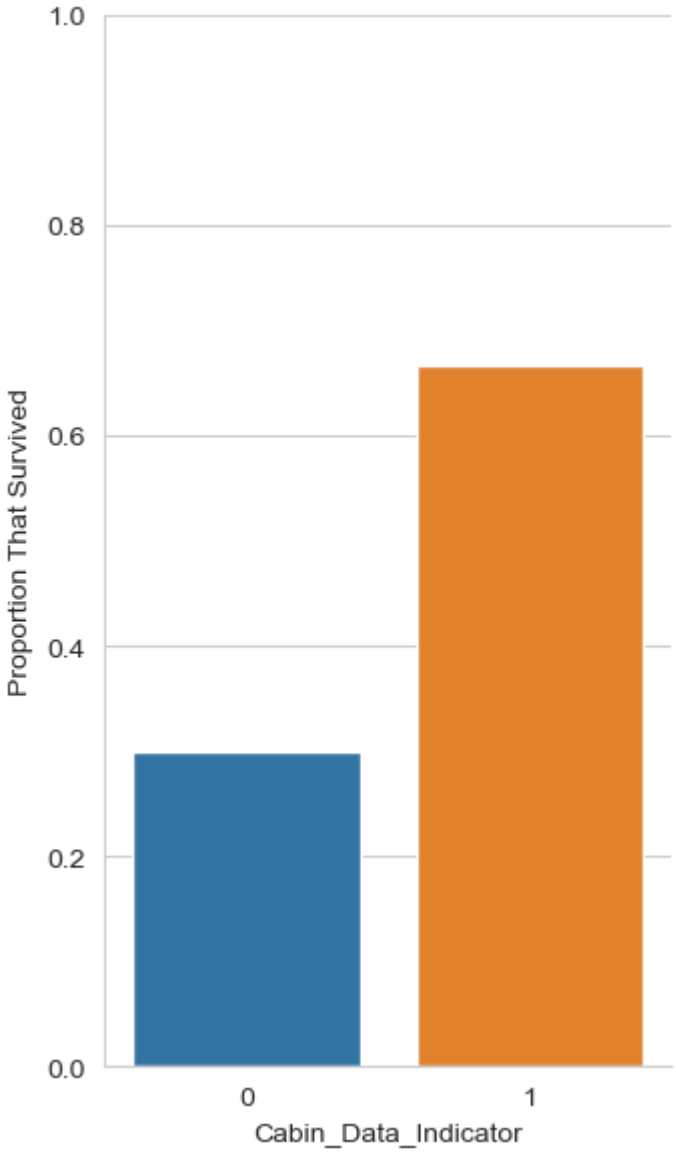

Age
Survived
0
1

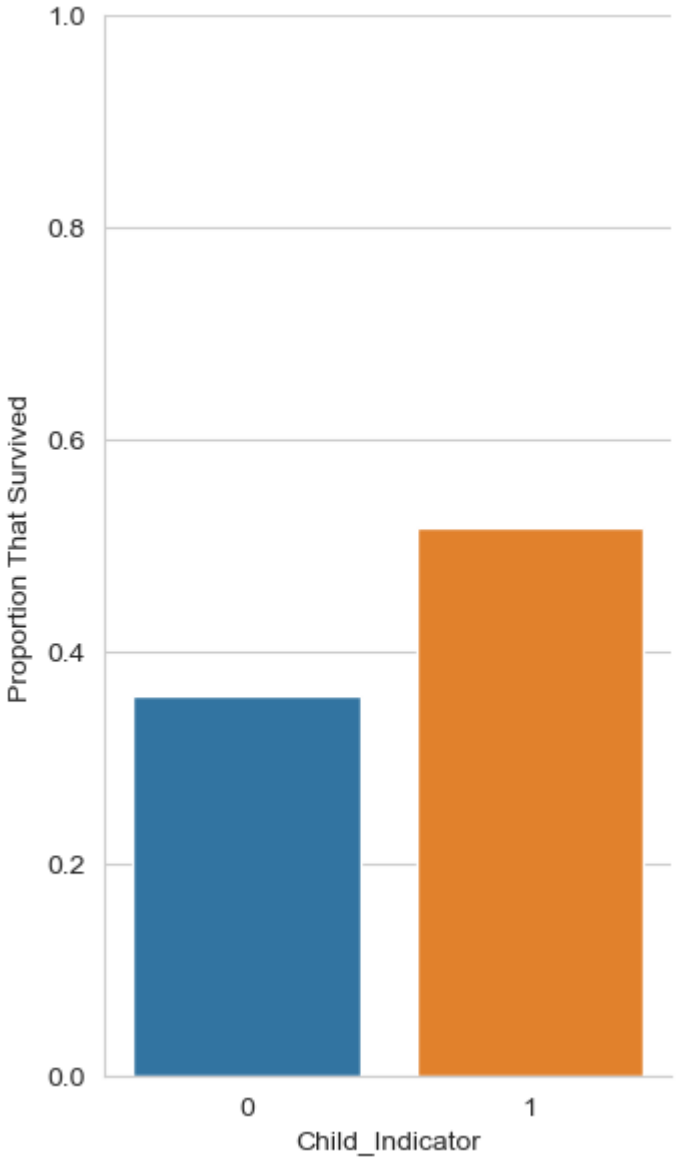

Fare
Survived
0
1

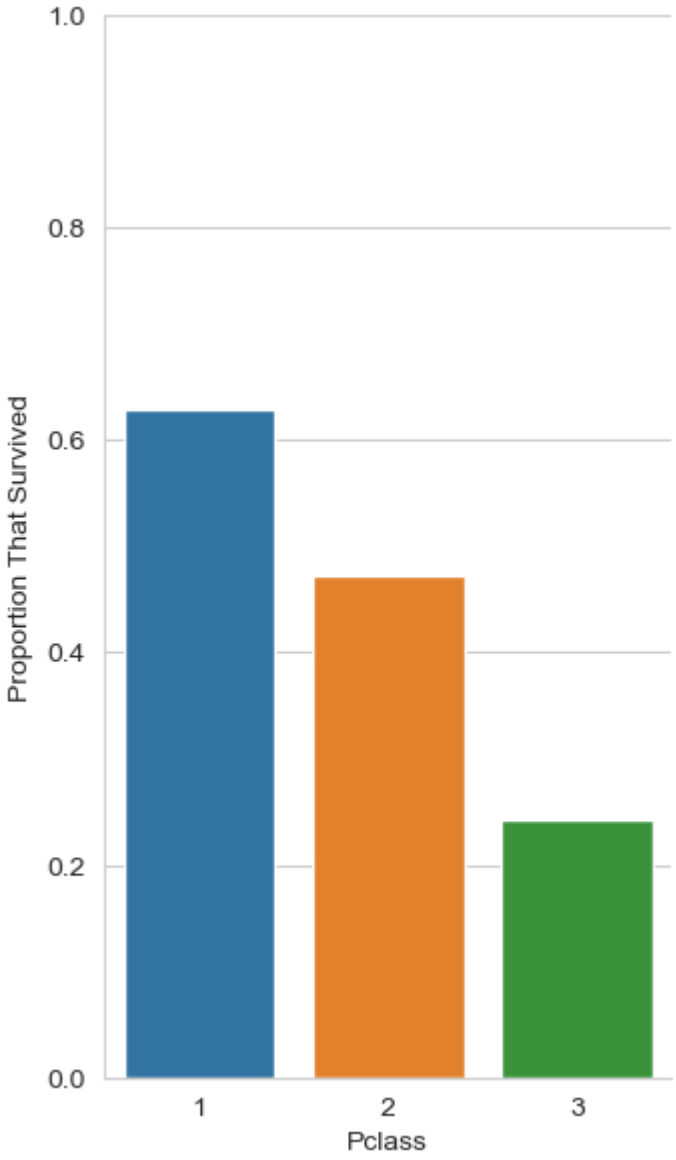

SibSp
Survived
0
1

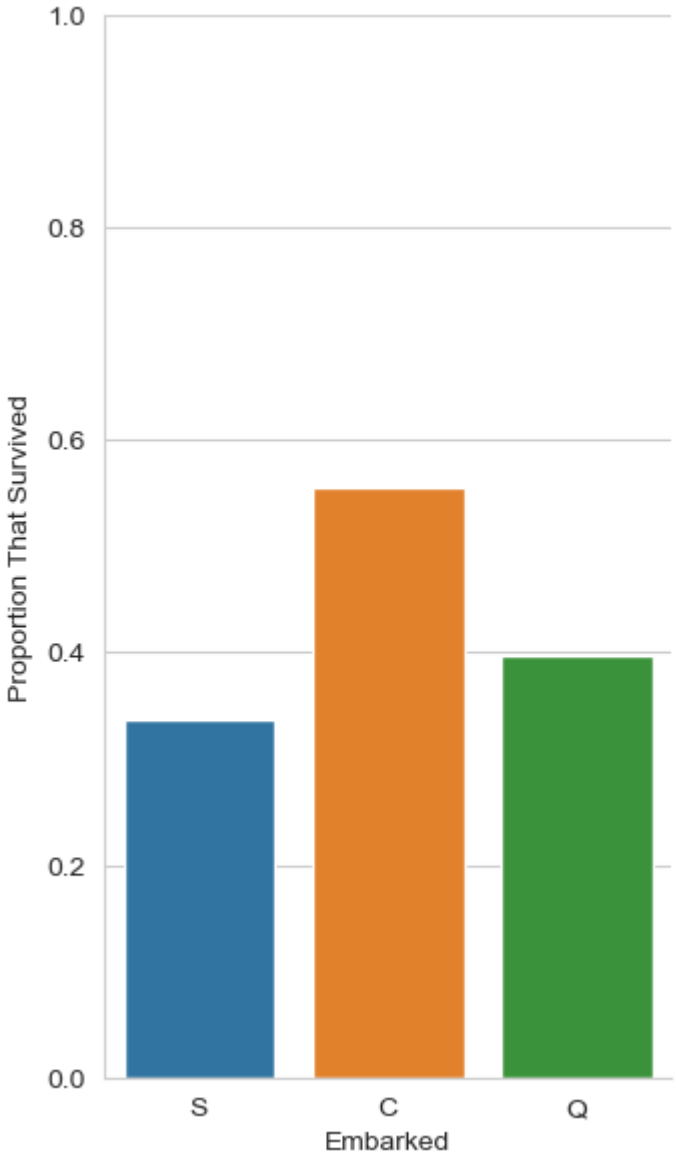

Parch
Survived
0
1

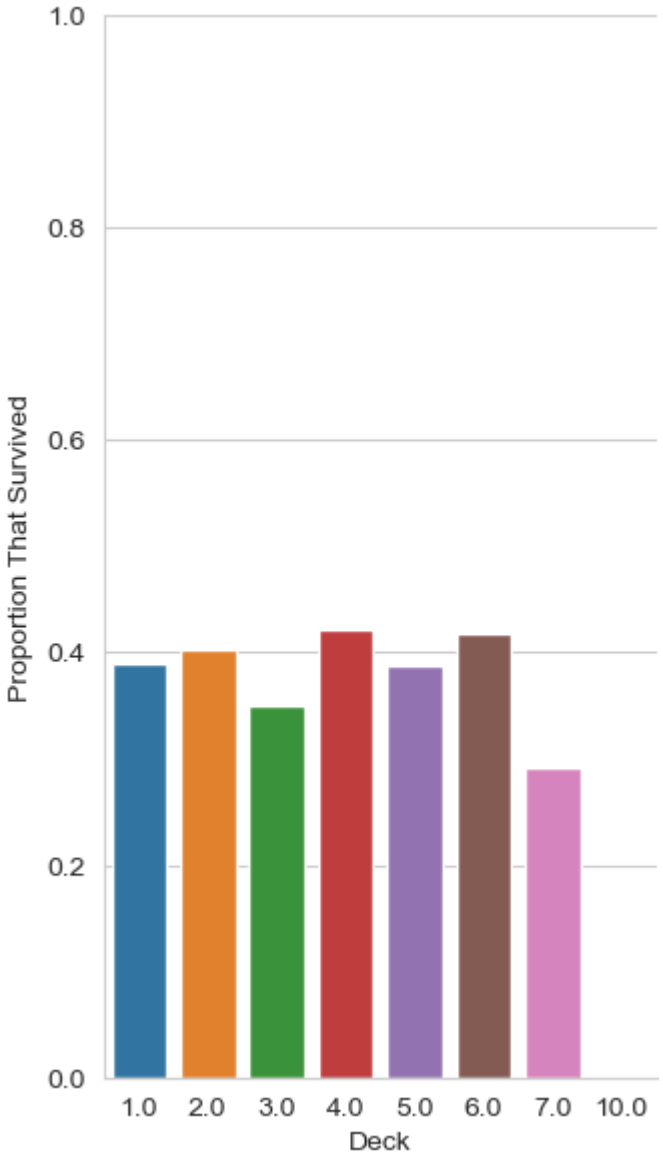

1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
0
1
Survived

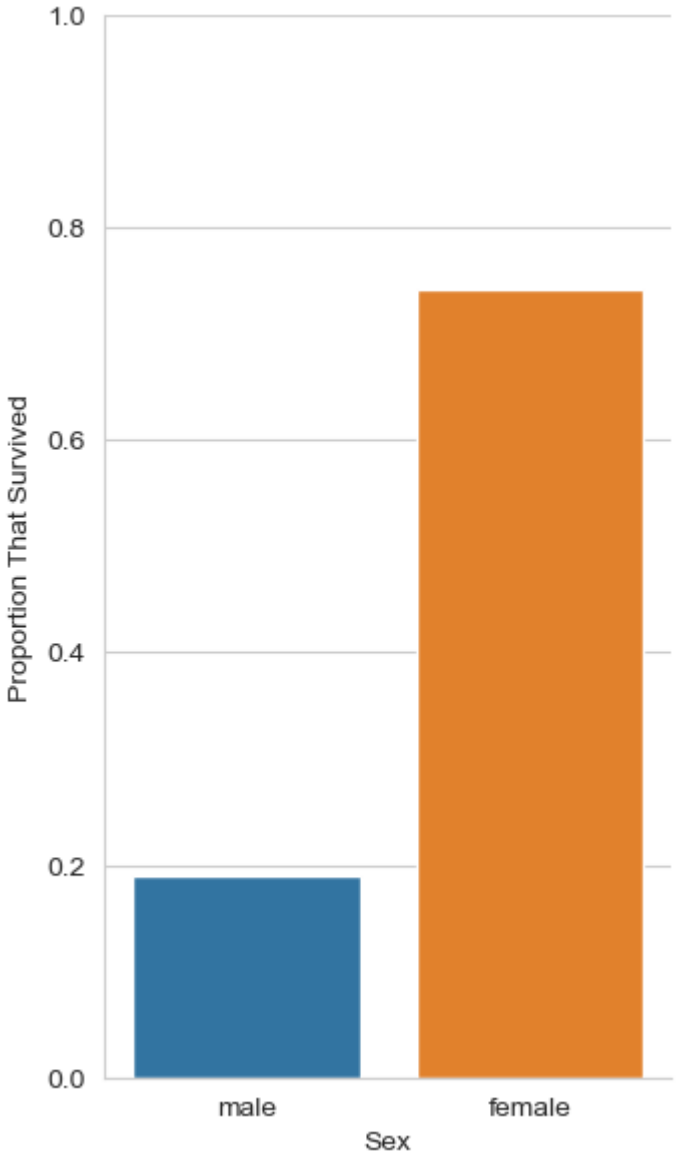

1.0
0.8
0.6
0.4
0.2
0.0
Proportion That Survived
0
1
Cabin_Data_Indicator


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
1.0
2.0
3.0
4.0
5.0
6.0
7.0
10.0
Deck



Random Forest

First, we need to conduct data cleaning exercises that will be needed only for the Random Forest data model. We will complete these exercises in a copy of the training dataset so that we don't impact other subsequent data anlayses.

In [98]:
```
# Create a copy of the training dataset
rf_training_validation_df = titanic_training_data_cleaned.copy(deep=True)

# Drop variables from the training dataset that we do not want to be included in the R
rf_training_validation_df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True

# Encode the sex and embarked variables
rf_training_validation_df['Female_Indicator'] = np.where(rf_training_validation_df['Se
rf_training_validation_df.drop(['Sex'], axis=1, inplace=True)

rf_training_validation_df['Embarked_C_Indicator'] = np.where(rf_training_validation_df
rf_training_validation_df['Embarked_Q_Indicator'] = np.where(rf_training_validation_df
rf_training_validation_df.drop(['Embarked'], axis=1, inplace=True)
```

In [99]:
```
len(rf_training_validation_df['Age'].to_list())
```

Out[99]:
```
891
```

Next, we will use our training and validation datasets to conduct hyperparameter tuning to find the best hyperparameters for random forest modeling.

In [28]:
```
# Import Required Modules
#pip install graphviz
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz

# Split the training dataset into predictor and outcome components
rf_train_validation_x = rf_training_validation_df.drop('Survived', axis=1)
rf_train_validation_y = rf_training_validation_df['Survived']

# Split the Kaggle Titanic training data into training and validation components
rf_x_train, rf_x_validation, rf_y_train, rf_y_validation = train_test_split(rf_train_v
                                                                            rf_train_validat
                                                                                  test_size=
                                                                                random_stat


# Conduct hyperparameter tuning for random forest models
param_dist = {'n_estimators': randint(10,500),
              'max_depth': randint(1,20),
             'max_features': randint(1,11),
             'criterion': ("gini", "entropy")}

rf = RandomForestClassifier()

rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_dist,
                                 n_iter=5,
                                 cv=5)

rand_search.fit(rf_x_train, rf_y_train)

# Create a variable for the best model
best_rf = rand_search.best_estimator_

# Print the best hyperparameters
print('Best hyperparameters:',  rand_search.best_params_)
```

Out[28]:
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

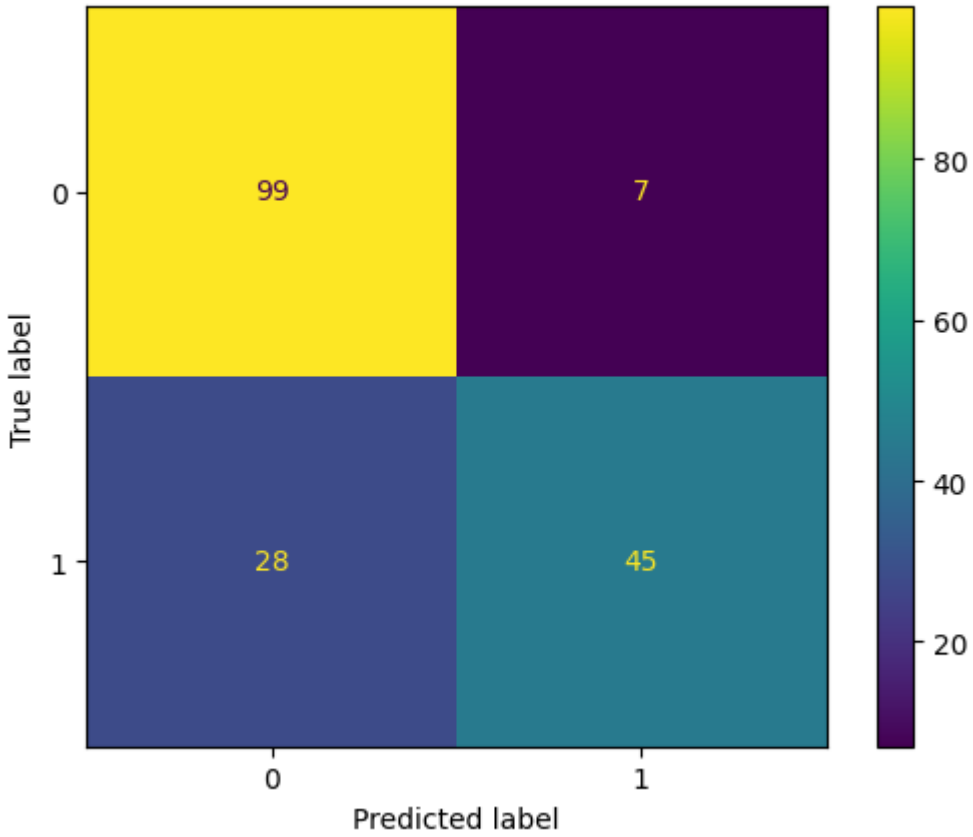
```
Best hyperparameters: {'criterion': 'entropy', 'max_depth': 9, 'max_features': 8, 'n_
estimators': 279}
```

Next, we will assess the strength of the random forest model associated with the optimal hyperparameters by applying the model to the validation dataset and observing the resulting confusion matrix, accuracy, precision, and recall.

In [29]:
```
# Generate predictions with the best model
y_validation_predictions_rf = best_rf.predict(rf_x_validation)

# Create the confusion matrix associated with the best random forest model
cm = confusion_matrix(rf_y_validation, y_validation_predictions_rf)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();

# Calculate the accuracy, precision, and recall associated with the predictions of the

accuracy_rf_validation = accuracy_score(rf_y_validation, y_validation_predictions_rf)
precision_rf_validation = precision_score(rf_y_validation, y_validation_predictions_rf
recall_rf_validation = recall_score(rf_y_validation, y_validation_predictions_rf)

print("Accuracy:", accuracy_rf_validation)
print("Precision:", precision_rf_validation)
print("Recall:", recall_rf_validation)
```

Out[29]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23a73145b20>
```

```
Accuracy: 0.8044692737430168
Precision: 0.8653846153846154
Recall: 0.6164383561643836
```



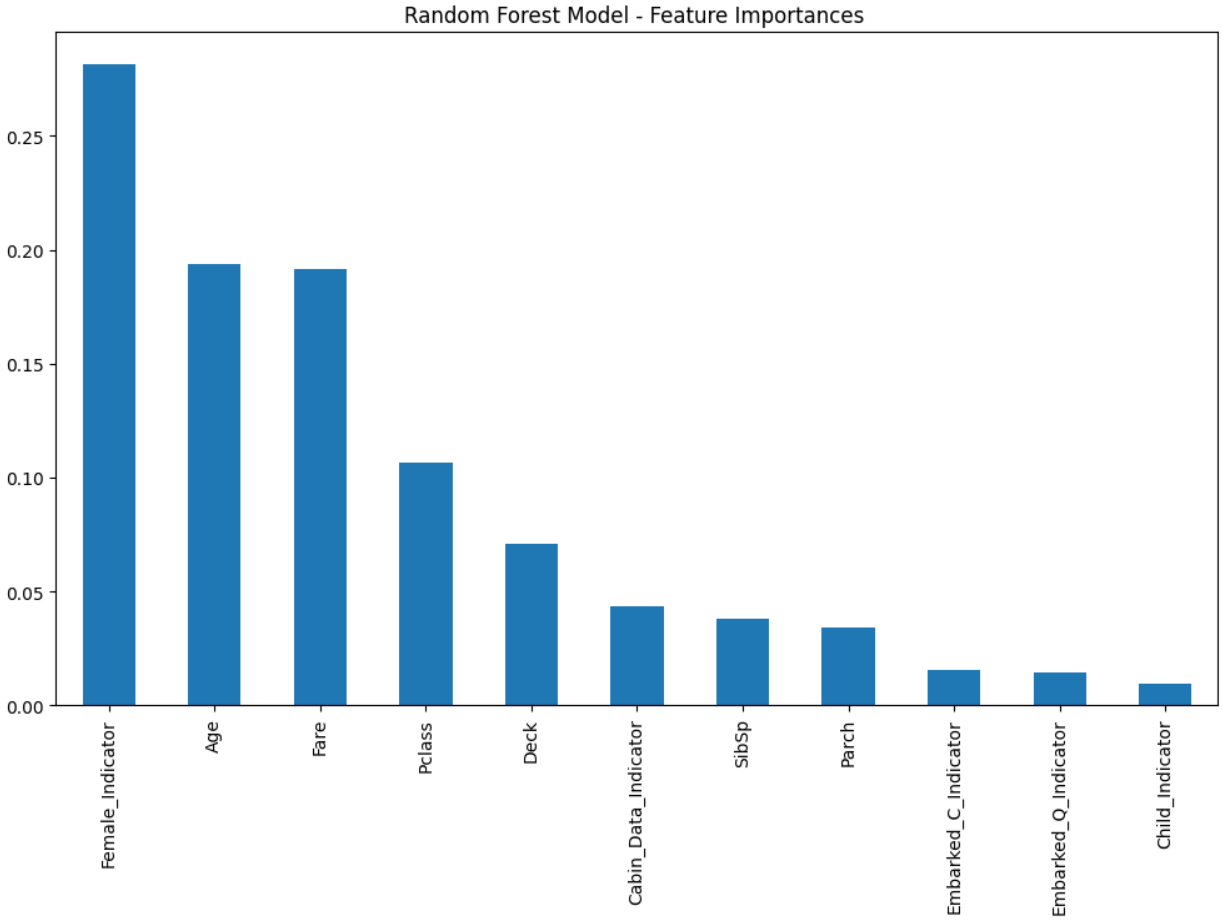
Next, we will visually display the variable importance metrics elicited from the random forest models associated with the optimal hyperparameter values.

In [30]:
```
# Plot a bar chart to display variable importances in the random forest models associa
feature_importances_rf_train = pd.Series(best_rf.feature_importances_, index=rf_train_

plt.figure(figsize=(12,7))

feature_importances_rf_train.plot.bar()

plt.title("Random Forest Model - Feature Importances ")
```

Out[30]:
```
<Figure size 1200x700 with 0 Axes>
```

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

Out[30]:
```
Text(0.5, 1.0, 'Random Forest Model - Feature Importances ')
```



Gradient Boosted Trees

First, set up the dataset using the rf_training_validation_df that was used to fit the random forest model

In [31]:
```
#pip install xgboost
```

In [32]:
```
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import GridSearchCV

# Seperate the predictor variables from the the outcome variable
gbtrees_X = rf_training_validation_df.drop('Survived', axis=1)
gbtress_y = rf_training_validation_df['Survived']
```

First, let's go through a simple classification example, using decision trees as the base predictors - this is the basis of an ensemble classification method so we can compare results from this simple example, manually building an ensemble to using GradientBoostingClassifier and XGBClassifier to train the ensembles more easily.

Manual Ensemble Example

In [33]:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

# fit a DecisionTreeRegressor, using default untuned parameters and max_depth=2
tree_model = DecisionTreeClassifier(max_depth=2, random_state=1)
tree_model.fit(gbtrees_X, gbtress_y)
y_pred1 = tree_model.predict(gbtrees_X)

# Accuracy Score of tree model 1
accuracy = accuracy_score(gbtress_y,y_pred1)
print('\naccuracy_score of tree model 1: ', accuracy)

# train a second DecisionTreeRegressor on the residual errors made by the first predic
gbtress_y2 = gbtress_y - y_pred1
tree_model2 = DecisionTreeClassifier(max_depth=2, random_state=1)
tree_model2.fit(gbtrees_X, gbtress_y2)
y_pred2 = tree_model2.predict(gbtrees_X)

# Accuracy Score of tree model 2
accuracy = accuracy_score(gbtress_y,y_pred2)
print('\naccuracy_score of tree model 2: ', accuracy)

# And then we'll train a third regressor on the residual errors made by the second pre
gbtress_y3 = gbtress_y2 - y_pred2
tree_model3 = DecisionTreeClassifier(max_depth=2, random_state=1)
tree_model3.fit(gbtrees_X, gbtress_y3)
y_pred3 = tree_model3.predict(gbtrees_X)

# Accuracy Score of tree model 3
accuracy = accuracy_score(gbtress_y,y_pred3)
print('\naccuracy_score of tree model 3: ', accuracy)



# ensemble containing three trees. It can make predictions on a new instance simply by
y_pred = sum(tree.predict(gbtrees_X) for tree in (tree_model, tree_model2, tree_model3


# Accuracy Score of model
accuracy = accuracy_score(gbtress_y,y_pred)
print('\naccuracy_score of ensemble: ', accuracy)
```

Out[33]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=1)
```

```
accuracy_score of tree model 1:  0.7867564534231201
```

Out[33]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=1)
```

```
accuracy_score of tree model 2:  0.6475869809203143
```

Out[33]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=1)
```

```
accuracy_score of tree model 3:  0.6105499438832772

accuracy_score of ensemble:  0.8092031425364759
```

We see an accuracy score of 0.809 in this manual ensemble classification example with 3 trees and no hyperparameter tuning.

Gradient Boosting Classifier and XGBoost Classifier

Let's use now use a Gradient Boosting Classifier and XGBoost Classifier with hyperparameter tuning to build new classification models.

Gradient Boosting

reference: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/ boosting specific parameters to tune:

1. n_estimators
2. Learning rate
3. subsample

tree specific parameters to tune:

1. max_depth
2. min_samples_split
3. min_samples_leaf
4. max_features

In [40]:
```
# Gradient boosting tunning
gb_model = GradientBoostingClassifier(random_state = 1)

# Set up 10-fold cross-validation to find the best hyperparameters from the grid below
cv = KFold(n_splits=10, random_state=1, shuffle=True)
gb_param_grid = {
              # Boosting Parameters
              'n_estimators' : [80,100,200], ### number of sequential trees to be mode
              'learning_rate': [0.01, 0.1, 0.5], ### impact of each tree on the final
              'subsample': [0.8,1], # fraction of observations to be selected for each
```

```
            # Tree-Specific Parameters
            'max_depth': [4, 6, 8, 10], ### maximum depth of a tree
            'min_samples_split': [2, 10, 20], ### minimum samples (or observations)
            'min_samples_leaf': [2, 10, 20], ### minimum samples (or observations) r
            #'max_features': [1,2,3] ### number of features to consider while search
            }

# set GridSearchCV parameters TAKES LONG TIME TO RUN, COMMENT OUT WHEN NOT TUNING
#gb = GridSearchCV(gb_model,param_grid = gb_param_grid, cv=cv, scoring="accuracy", n_j


# fit the model using parms from GridSearch
#gb.fit(gbtrees_X, gbtress_y)

# Create a variable for the best model
#gb_tuned_model = gb.best_estimator_

#print('Gradient Boosting...')
#print('\nGradient Boost accuracy score: ', gb.best_score_)
#print('\nGradient Boost parameters: ', gb.best_params_)
```

```
Fitting 3 folds for each of 648 candidates, totalling 1944 fits
```

Out[40]:

GridSearchCV

estimator: GradientBoostingClassifier

GradientBoostingClassifier

```
Gradient Boosting...

Gradient Boost accuracy score:  0.8294051627384961

Gradient Boost parameters:  {'learning_rate': 0.01, 'max_depth': 8, 'min_samples_lea
f': 20, 'min_samples_split': 2, 'n_estimators': 200, 'subsample': 0.8}
```

In [34]:

```
# set model with best parameters from tuning
gb_tuned_model =  GradientBoostingClassifier(n_estimators = 200,
                                  max_depth = 8,
                                  learning_rate = 0.01,
                                  max_features = 0.3,
                                  min_samples_leaf = 20,
                                  min_samples_split = 2,
                                  random_state = 1,
                                  subsample = 0.8)

# fit the model using parms from GridSearch
gb_tuned_model.fit(gbtrees_X, gbtress_y)
```

Out[34]:

GradientBoostingClassifier

```
GradientBoostingClassifier(learning_rate=0.01, max_depth=8, max_features=0.
3,
                           min_samples_leaf=20, n_estimators=200,
                           random_state=1, subsample=0.8)
```

In [35]:

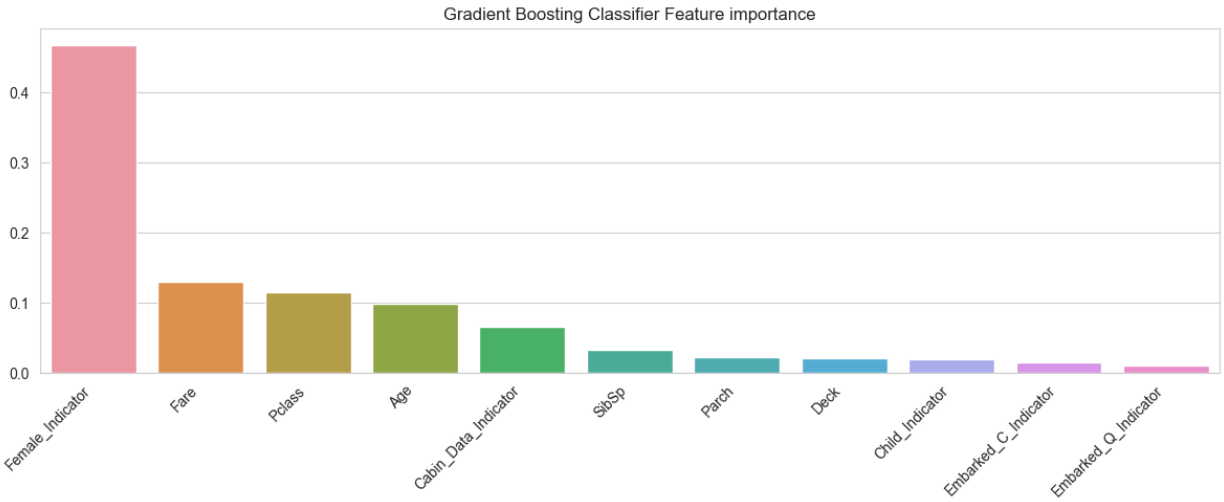
```
# Calculate feature importances
importances = gb_tuned_model.feature_importances_
```

```
# Visualize Feature Importance
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [gbtrees_X.columns[i] for i in indices]

plt.figure(figsize = (12, 5))
sns.set_style("whitegrid")
chart = sns.barplot(x = names, y=importances[indices])
plt.xticks(rotation=45, horizontalalignment='right', fontweight='light')
plt.title('Gradient Boosting Classifier Feature importance')
plt.tight_layout()
```

Out[35]:
```
<Figure size 1200x500 with 0 Axes>
```

Out[35]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'Female_Indicator'),
  Text(1, 0, 'Fare'),
  Text(2, 0, 'Pclass'),
  Text(3, 0, 'Age'),
  Text(4, 0, 'Cabin_Data_Indicator'),
  Text(5, 0, 'SibSp'),
  Text(6, 0, 'Parch'),
  Text(7, 0, 'Deck'),
  Text(8, 0, 'Child_Indicator'),
  Text(9, 0, 'Embarked_C_Indicator'),
  Text(10, 0, 'Embarked_Q_Indicator')])
```

Out[35]:
```
Text(0.5, 1.0, 'Gradient Boosting Classifier Feature importance')
```



We can see that female indicator, fare, age, and pclass appear to be the most important predictors of survival in this model

Calculate ROC and precision-recall curves for the model

In [36]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import precision_recall_curve, auc
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve
```

```
predictions = gb_tuned_model.predict(gbtrees_X)
probabilities = gb_tuned_model.predict_proba(gbtrees_X)[:,1]


# Calculate the accuracy, precision, and recall associated with the predictions of the

accuracy_gb = accuracy_score(gbtress_y, predictions)
precision_gb = precision_score(gbtress_y, predictions)
recall_gb = recall_score(gbtress_y, predictions)

print("Accuracy:", accuracy_gb)
print("Precision:", precision_gb)
print("Recall:", recall_gb)


# Confusion

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


# Curves

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

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

```
Accuracy: 0.8653198653198653
Precision: 0.8908450704225352
Recall: 0.7397660818713451
```

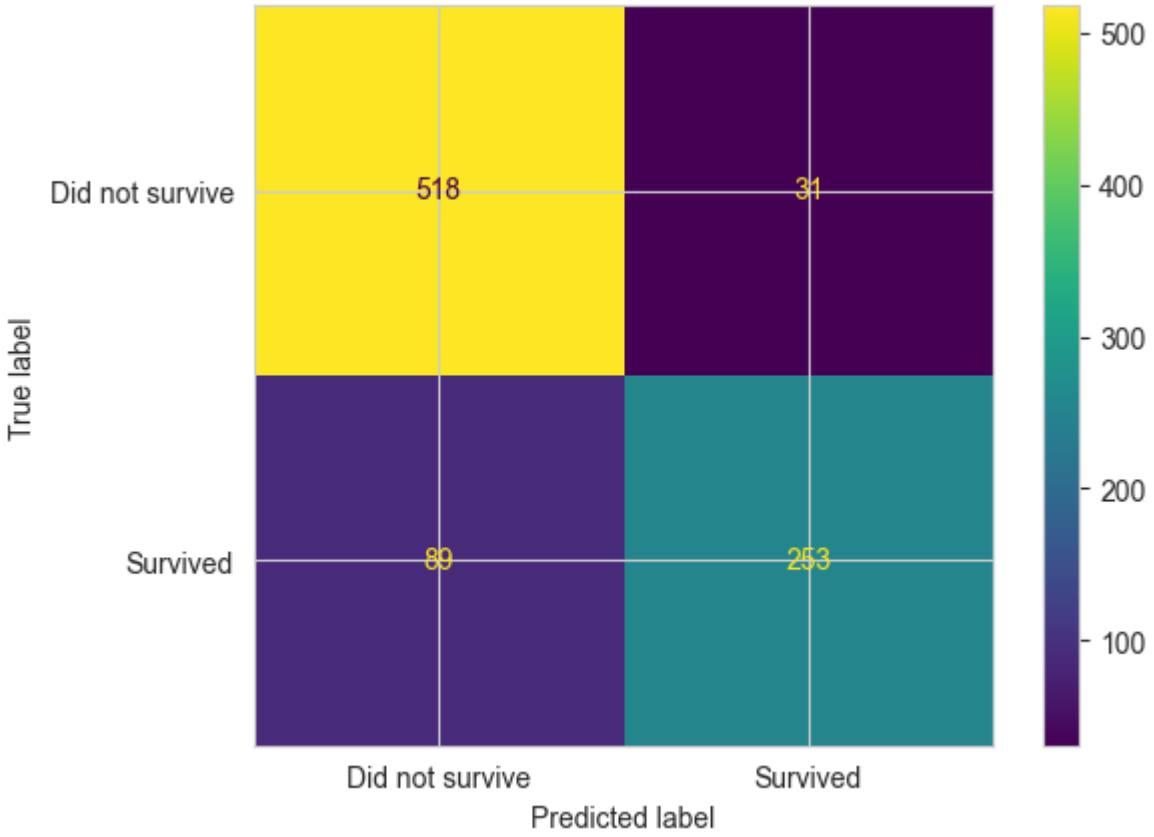
Out[36]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23a76d51340>
```

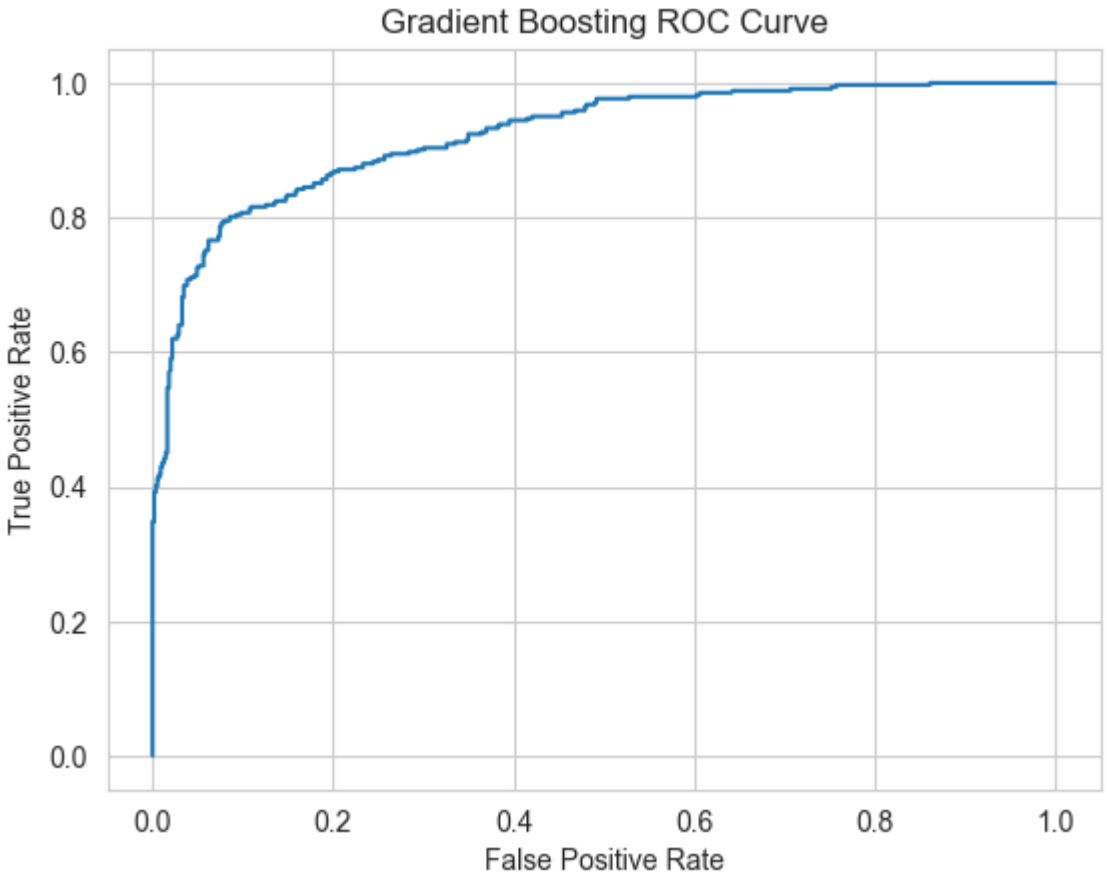
Out[36]:
```
Text(0.5, 1.0, 'Gradient Boosting ROC Curve')
```

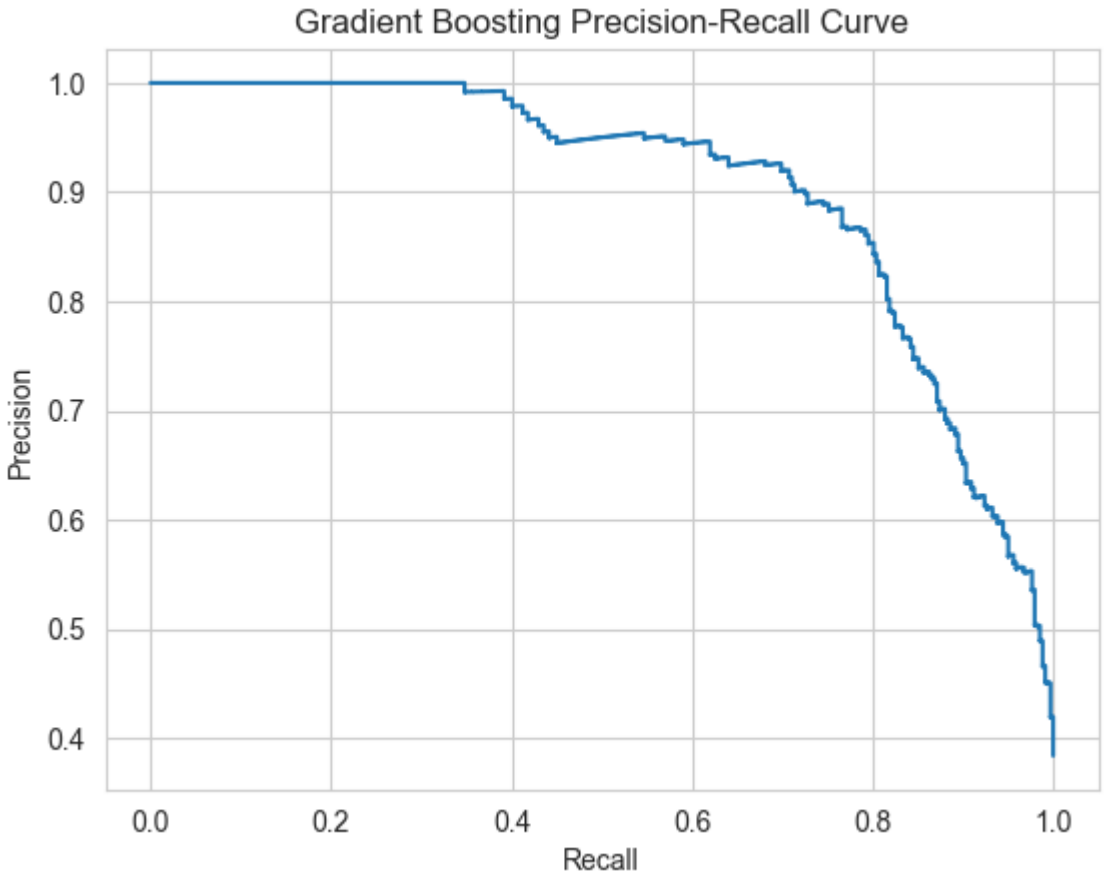
```
The roc auc score is: 0.922682389032691
```

Out[36]:
```
Text(0.5, 1.0, 'Gradient Boosting Precision-Recall Curve')
```

```
The prec-recall auc score is: 0.9034855538287773
```


Did not survive
Survived
True label
518
31
80
253
Did not survive
Survived
Predicted label
500
400
300
200
100


Gradient Boosting ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



XGBoost

reference: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/#XGBoost_Parameters

Some XGBoost Parameters General Parameters: Guide the overall functioning

1. booster [default=gbtree]
   - gbtree: tree-based models
   - gblinear: linear models

Booster Parameters: Guide the individual booster (tree/regression) at each step

1. learning_rate [default=0.3]
   - Analogous to the learning rate in GBM
2. max_depth
   - The maximum depth of a tree is the same as GBM
3. gamma [default=0]
   - Gamma specifies the minimum loss reduction required to make a split
4. reg_lambda [default=1]
   - L2 regularization term on weights
5. reg_alpha [default=0]
   - L1 regularization term on weight
6. subsample [default=1]

- Same as the subsample of GBM. Denotes the fraction of observations to be random samples for each tree

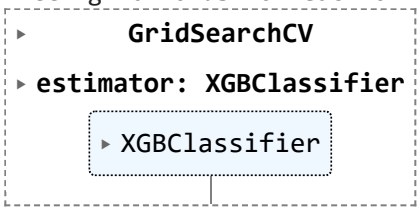
Learning Task Parameters: Guide the optimization performed

1. objective [default=reg:linear]
   - binary: logistic
2. eval_metric [default according to objective]

In [80]:
```
# XGBoost boosting tunning
xgb_model = XGBClassifier(objective = 'binary:logistic', seed = 1)


# Set up 10-fold cross-validation to find the best hyperparameters from the grid below
cv = KFold(n_splits=10, random_state=1, shuffle=True)

xgb_param_grid = {
                 'learning_rate' : [0.01, 0.1, 0.5],
                 'gamma'         : [0, 0.1, 0.2],
                 'min_child_weight':[8, 10, 12],
                 'max_depth'     : [4, 5, 6],
                 'reg_lambda'    : [0, 0.1, 0.5, 1],
                 'reg_alpha'     : [0, 0.1, 0.5, 1],
                 'subsample'     : [0.8, 1]
               }


# set GridSearchCV parameters WILL TAKE A WHILE TO RUN
#xgb = GridSearchCV(xgb_model,param_grid = xgb_param_grid, cv=cv, scoring="accuracy",
#xgb.fit(gbtrees_X, gbtress_y)

# Create a variable for the best model
#xgb_tuned_model = xgb.best_estimator_
```

```
Fitting 10 folds for each of 2592 candidates, totalling 25920 fits
```

Out[80]:



In [37]:
```
#print('Gradient Boosting...')
#print('\nGradient Boost accuracy score: ', xgb.best_score_)
#print('\nGradient Boost parameters: ', xgb.best_params_)
```

In [38]:
```
# build model with tuned parameters from grid search
xgb_tuned_model =  XGBClassifier(objective = 'binary:logistic',
                                 seed = 1,
                                 gamma = 0,
                                 max_depth = 4,
                                 learning_rate = 0.5,
                                 min_child_weight =  12,
                                 reg_alpha = 0.1,
                                 reg_lambda = 1,
                                 subsample = 0.8)
```

```
# fit the model using parms from GridSearch
xgb_tuned_model.fit(gbtrees_X, gbtress_y)
```

Out[38]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=N
one,
              gamma=0, gpu_id=None, grow_policy=None, importance_type=Non
e,
              interaction_constraints=None, learning_rate=0.5, max_bin=No
ne,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
```

In [39]:

```
# Calculate feature importances
importances = xgb_tuned_model.feature_importances_

# Visualize Feature Importance
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [gbtrees_X.columns[i] for i in indices]

plt.figure(figsize = (12, 5))
sns.set_style("whitegrid")
chart = sns.barplot(x = names, y=importances[indices])
plt.xticks(rotation=45, horizontalalignment='right', fontweight='light')
plt.title('XGBoost. Feature importance')
plt.tight_layout()
```

Out[39]:

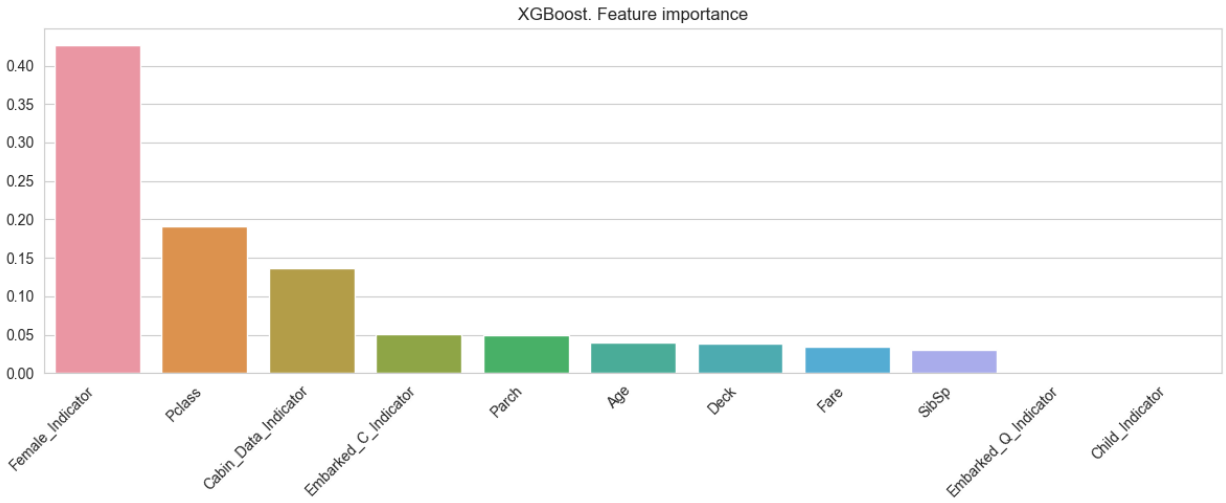
```
<Figure size 1200x500 with 0 Axes>
```

Out[39]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'Female_Indicator'),
  Text(1, 0, 'Pclass'),
  Text(2, 0, 'Cabin_Data_Indicator'),
  Text(3, 0, 'Embarked_C_Indicator'),
  Text(4, 0, 'Parch'),
  Text(5, 0, 'Age'),
  Text(6, 0, 'Deck'),
  Text(7, 0, 'Fare'),
  Text(8, 0, 'SibSp'),
  Text(9, 0, 'Embarked_Q_Indicator'),
  Text(10, 0, 'Child_Indicator')])
```

Out[39]:

```
Text(0.5, 1.0, 'XGBoost. Feature importance')
```



We can see the female indicator, passenger ticket class, and cabin data indicator appear to be the most important predictors of survival in this model.

Calculate ROC and precision-recall curves for the model

In [40]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import precision_recall_curve, auc
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve


predictions = xgb_tuned_model.predict(gbtrees_X)
probabilities = xgb_tuned_model.predict_proba(gbtrees_X)[:,1]



# Calculate the accuracy, precision, and recall associated with the predictions of the

accuracy_xgb = accuracy_score(gbtress_y, predictions)
precision_xgb = precision_score(gbtress_y, predictions)
recall_xgb = recall_score(gbtress_y, predictions)

print("Accuracy:", accuracy_xgb)
print("Precision:", precision_xgb)
print("Recall:", recall_xgb)



# Confusion

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


# Curves

fpr, tpr, _ = roc_curve(gbtress_y, probabilities)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('Gradient Boosting ROC Curve')
# roc auc score
```

```
auc1 = roc_auc_score(gbtress_y, probabilities)
print("The roc auc score is:", auc1)

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

```
Accuracy: 0.9034792368125701
Precision: 0.9050632911392406
Recall: 0.8362573099415205
```

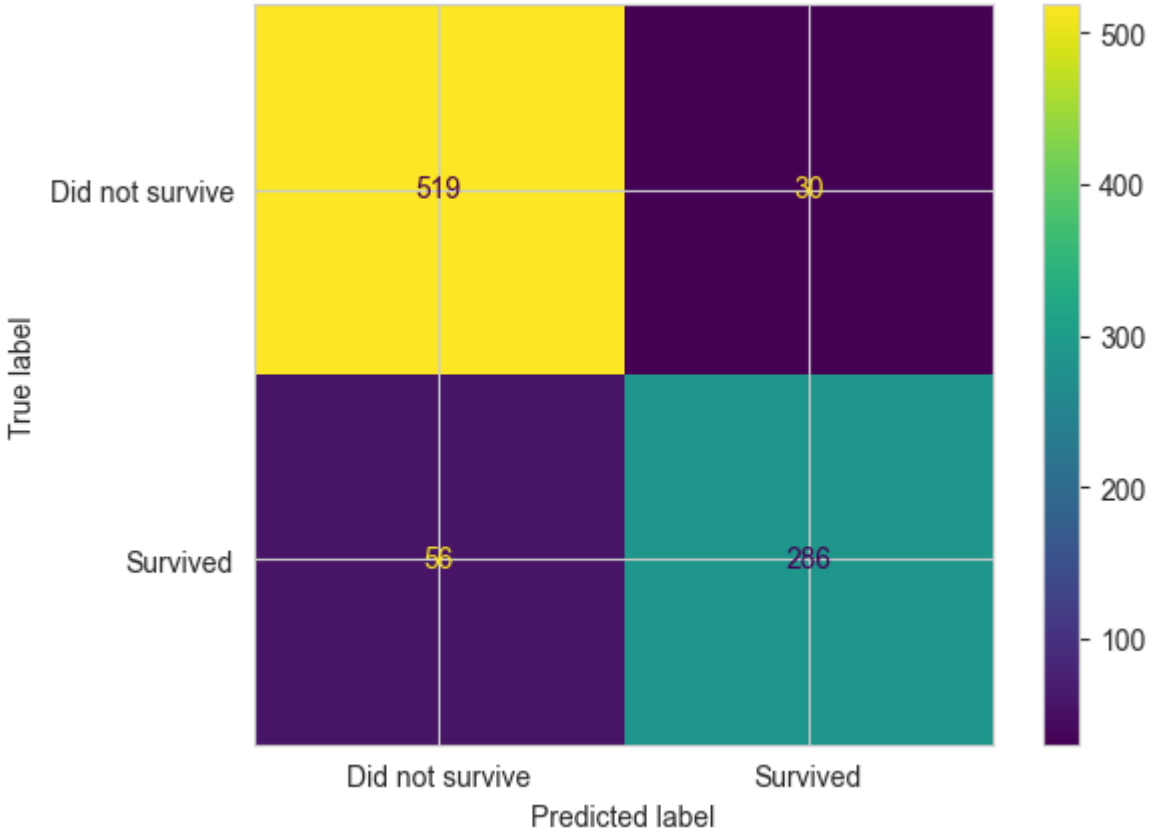
Out[40]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23a78a89ca0>
```

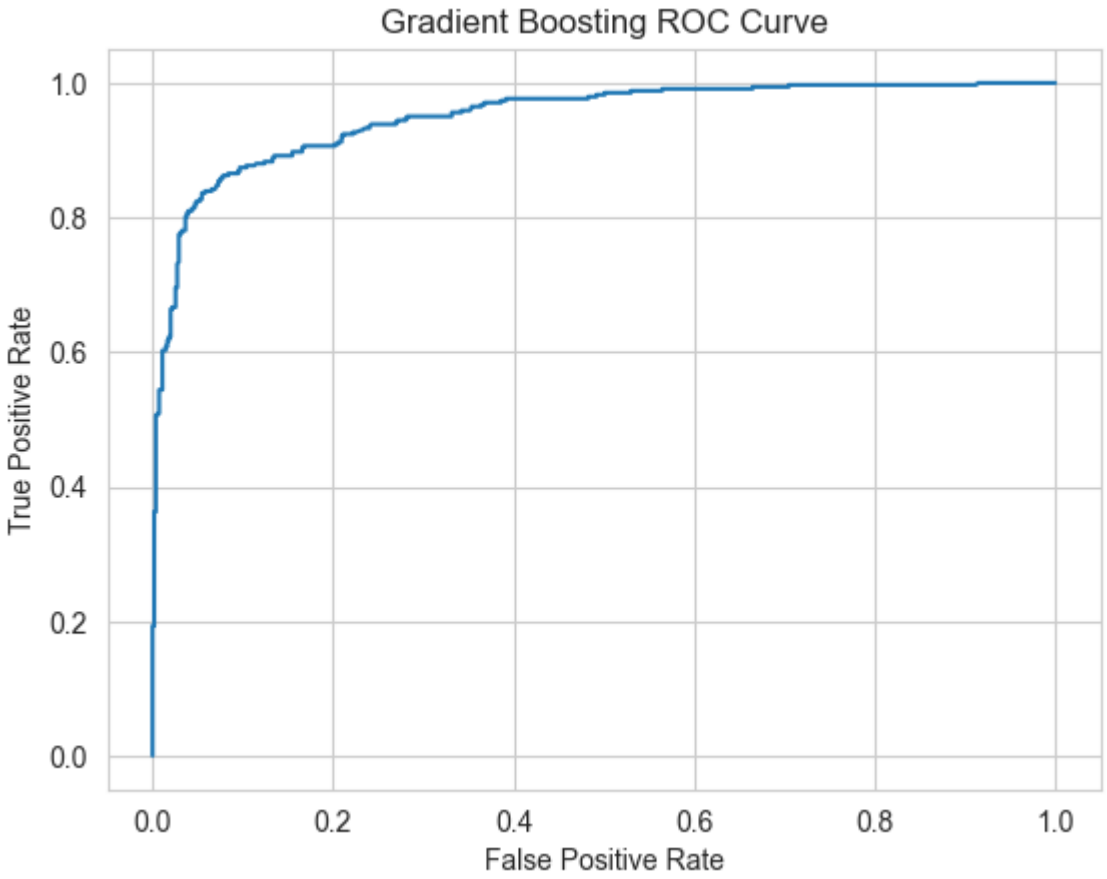
Out[40]:
```
Text(0.5, 1.0, 'Gradient Boosting ROC Curve')
```

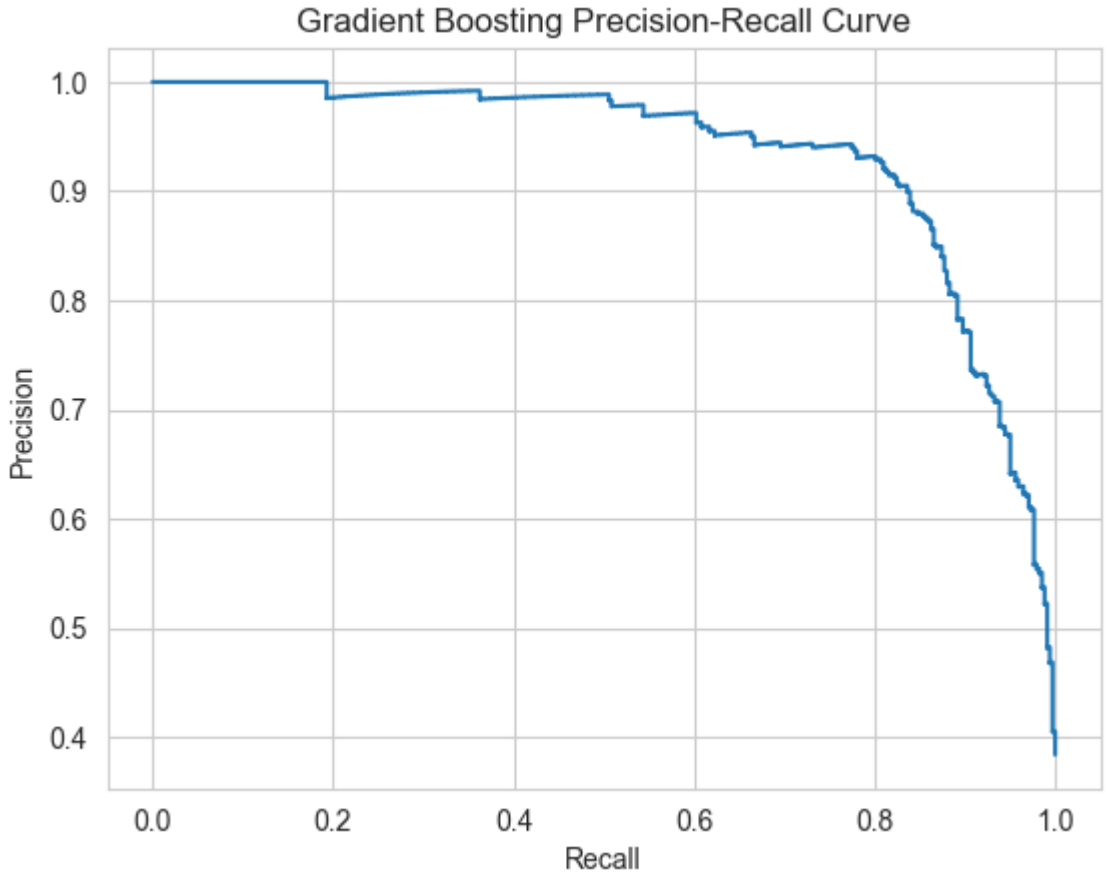
```
The roc auc score is: 0.9500820204731621
```

Out[40]:
```
Text(0.5, 1.0, 'Gradient Boosting Precision-Recall Curve')
```

```
The prec-recall auc score is: 0.9345517500326131
```




Gradient Boosting ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Gradient Boosting Precision-Recall Curve
Precision
Recall
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.2
0.4
0.6
0.8
1.0

Extra Trees

Extra Trees Regressor model

First we'll setup the needed packages, and then set X and Y

In [100…
```
#Load packages
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, cross_val_score
from numpy import mean
import seaborn as sns


# Split the training dataset into predictor and outcome components
extratrees_x = rf_training_validation_df.drop('Survived', axis=1)
extratrees_y = rf_training_validation_df['Survived']
```

Next, we split the data:

In [101…
```
# importing the module
from sklearn.model_selection import train_test_split
# splitting the dataset
X_train, X_test, y_train, y_test=train_test_split(extratrees_x, extratrees_y, test_siz
```

Now we will initialize an Extra Trees Regression model

In [102…
```
# importing the module
from sklearn.ensemble import ExtraTreesRegressor
# initializing the model
ET_regressor = ExtraTreesRegressor(n_estimators=100, random_state=1)
# printing the parameters
print(ET_regressor.get_params())
```

```
{'bootstrap': False, 'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth': Non
e, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity
_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_
leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state':
1, 'verbose': 0, 'warm_start': False}
```

We'll train the model first:

In [103…
```
# Training the model
ET_regressor.fit(X_train, y_train)
```

Out[103]:
```
ExtraTreesRegressor(random_state=1)
```

As well as make predictions using this model:

In [104…
```
# Making predictions
Regressor_pred = ET_regressor.predict(X_test)
```

We can then visualize the actual values and predicted values:

In [105…
```
# fitting the size of the plot
plt.figure(figsize=(15, 8))
```

```
# plotting the graphs
plt.plot([i for i in range(len(y_test))],y_test, color = 'green',label="actual values'
plt.plot([i for i in range(len(y_test))],Regressor_pred, color='red', label="Predicted
# showing the plotting
plt.legend()
plt.show()
```

Out[105]:
```
<Figure size 1500x800 with 0 Axes>
```

Out[105]:
```
[<matplotlib.lines.Line2D at 0x162e1d2bc40>]
```

Out[105]:
```
[<matplotlib.lines.Line2D at 0x162e1d2bfd0>]
```

Out[105]:
```
<matplotlib.legend.Legend at 0x162e1d2bdf0>
```

To determine how well the model performs, we can calculate R-squared, which helps determine goodness of fit:

In [107…
```
# Importing the required module
from sklearn.metrics import  r2_score
# Evaluating model performance
print('R-square score is :', r2_score(y_test, Regressor_pred))
```

```
R-square score is : 0.21442671287046267
```

Extra Trees classifier model

Next, we can conduct an Extra Trees classifier model to explore differences between an Extra Trees regressin model compared to a Extra trees classifier model

Let's initialize the model first and fit it to the split data. Then let's evaluate the model using 10-fold cross-validation as the evaluation metric:

In [108…
```
# importing the module
from sklearn.ensemble import ExtraTreesClassifier
```

```
# initializing the model
ET_classifier = ExtraTreesClassifier(n_estimators=100, random_state=1)

# evaluate using cross-validation
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(ET_classifier, extratrees_x, extratrees_y, scoring='accuracy'
print('Accuracy: ', mean(scores))

# Training the model
ET_classifier.fit(X_train, y_train)
```

```
Accuracy:  0.7946566791510612
```

Out[108]:
```
ExtraTreesClassifier(random_state=1)
```

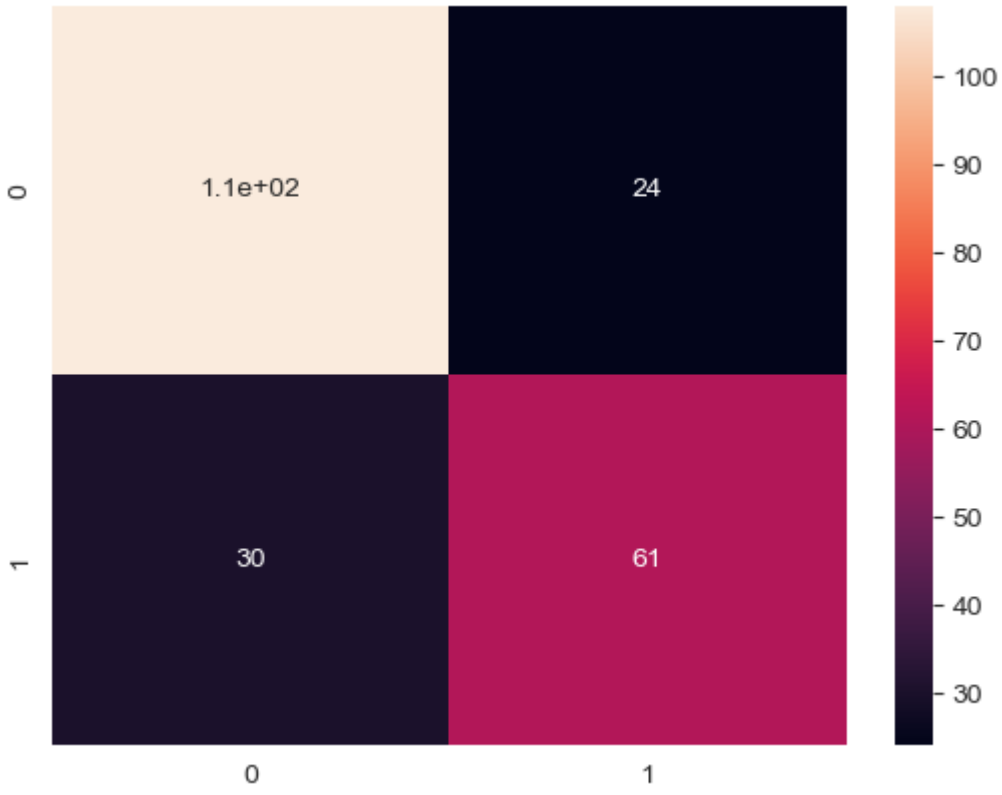
Next, this can be used to predict

In [109…
```
# making predictions
classifier_pred = ET_classifier.predict(X_test)
```

We can visualize performance through a confusion matrix

In [110…
```
# importing seaborn
import seaborn as sns
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
# providing actual and predicted values
cm = confusion_matrix(y_test, classifier_pred)
# If True, write the data value in each cell
sns.heatmap(cm,annot=True)
```

Out[110]:
```
<AxesSubplot:>
```



Next let's determine the optimal parameters for an Extra Trees classifier model

In [111…
```
# printing the parameters
print(ET_classifier.get_params())
```

```
{'bootstrap': False, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'ma
x_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_wei
ght_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'ra
ndom_state': 1, 'verbose': 0, 'warm_start': False}
```

In [112…
```
from sklearn.model_selection import GridSearchCV
cv = KFold(n_splits=10, random_state=1, shuffle=True)
parameters = {
    'n_estimators' : [100, 200, 500],
    'min_samples_leaf': [5,10,20],
    'max_features': [2,3,4],
    'criterion': ['gini','entropy'],
    'max_depth': [5,10,20]


}
clf = GridSearchCV(ExtraTreesClassifier(), param_grid=parameters, cv=cv)
clf.fit(X_train, y_train)
print(clf.best_estimator_)
```

Out[112]:
```
GridSearchCV(cv=KFold(n_splits=10, random_state=1, shuffle=True),
             estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [5, 10, 20], 'max_features': [2, 3, 4],
                         'min_samples_leaf': [5, 10, 20],
                         'n_estimators': [100, 200, 500]})
ExtraTreesClassifier(max_depth=20, max_features=4, min_samples_leaf=5)
```

In [113…
```
from sklearn.metrics import classification_report
```

Next, let's predict using this updated Extra Trees classifier model:

In [114…
```
clf_predictions = clf.predict(X_test)

# print classification report
print(classification_report(y_test, clf_predictions))
```

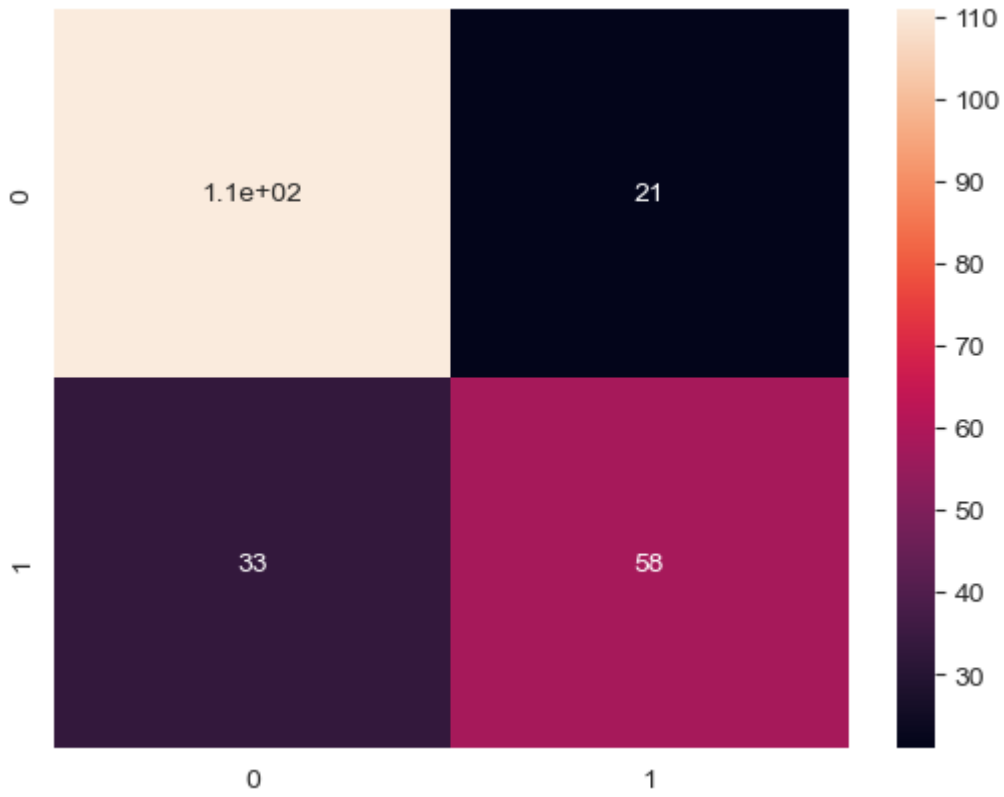
```
              precision    recall  f1-score   support

           0       0.77      0.84      0.80       132
           1       0.73      0.64      0.68        91

    accuracy                           0.76       223
   macro avg       0.75      0.74      0.74       223
weighted avg       0.76      0.76      0.75       223
```

We can visualize performance through a confusion matrix

In [115…
```
# importing seaborn
import seaborn as sns
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix

# providing actual and predicted values
cm = confusion_matrix(y_test, clf_predictions)

# If True, write the data value in each cell
sns.heatmap(cm,annot=True)
```

Out[115]: <AxesSubplot:>



In [116…

```
# Calculate the accuracy, precision, and recall associated with the predictions of the

accuracy_ExtraTrees = accuracy_score(y_test, clf_predictions)
precision_ExtraTrees = precision_score(y_test, clf_predictions)
recall_ExtraTrees = recall_score(y_test, clf_predictions)

print("Accuracy:", accuracy_ExtraTrees)
print("Precision:", precision_ExtraTrees)
print("Recall:", recall_ExtraTrees)
```

```
Accuracy: 0.757847533632287
Precision: 0.7341772151898734
Recall: 0.6373626373626373
```

In [117…

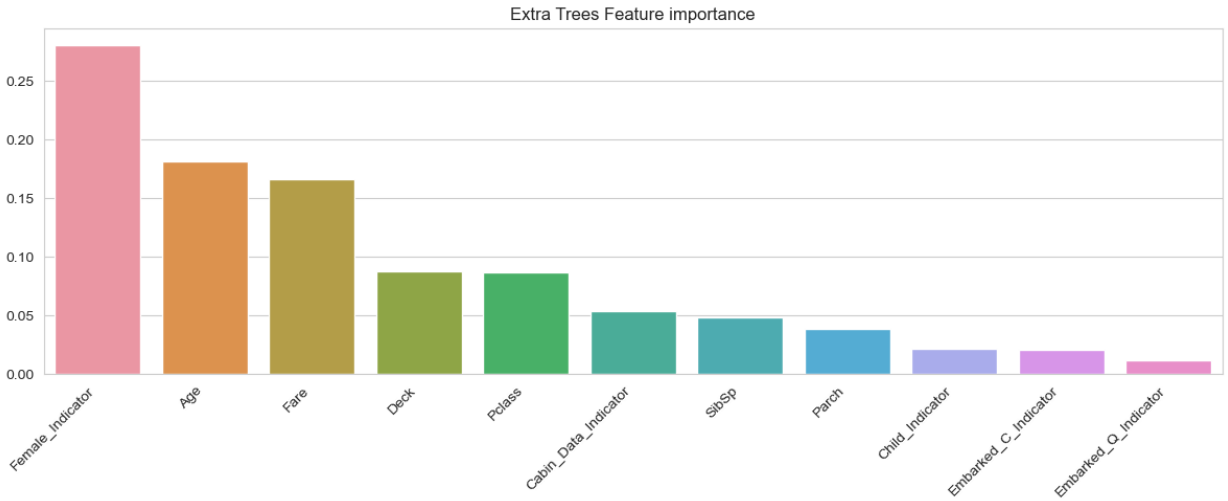
```
# Calculate feature importances
importances = ET_classifier.feature_importances_

# Visualize Feature Importance
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]
```

```
plt.figure(figsize = (12, 5))
sns.set_style("whitegrid")
chart = sns.barplot(x = names, y=importances[indices])
plt.xticks(rotation=45, horizontalalignment='right', fontweight='light')
plt.title('Extra Trees Feature importance')
plt.tight_layout()
```

Out[117]:
```
<Figure size 1200x500 with 0 Axes>
```

Out[117]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'Female_Indicator'),
  Text(1, 0, 'Age'),
  Text(2, 0, 'Fare'),
  Text(3, 0, 'Deck'),
  Text(4, 0, 'Pclass'),
  Text(5, 0, 'Cabin_Data_Indicator'),
  Text(6, 0, 'SibSp'),
  Text(7, 0, 'Parch'),
  Text(8, 0, 'Child_Indicator'),
  Text(9, 0, 'Embarked_C_Indicator'),
  Text(10, 0, 'Embarked_Q_Indicator')])
```

Out[117]:
```
Text(0.5, 1.0, 'Extra Trees Feature importance')
```



Import and Clean Testing Dataset

Import the Titanic Testing Dataset

In [58]:
```
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[58]:

	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[58]:
```
(418, 11)
```

Check the testing dataset for missing values

In [59]:
```
# 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[59]:

	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 [60]:

```
# 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)

# 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'] = titanic_testing_data_cleaned['Cabir

def conditions(s):
    if (s['First_Cabin_Deck'] == "A"):
        return 1
    elif (s['First_Cabin_Deck'] == "B"):
        return 2
    elif (s['First_Cabin_Deck'] == "C"):
        return 3
    elif (s['First_Cabin_Deck'] == "D"):
        return 4
    elif (s['First_Cabin_Deck'] == "E"):
        return 5
    elif (s['First_Cabin_Deck'] == "F"):
        return 6
    elif (s['First_Cabin_Deck'] == "G"):
        return 7
    elif (s['First_Cabin_Deck'] == "T"):
        return 10
    elif (s['First_Cabin_Deck'] == "n"):
        return np.nan
    else:
        return np.nan

titanic_testing_data_cleaned['Deck'] = titanic_testing_data_cleaned.apply(conditions,

# 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', 'First_Cabin_Deck'],axis=1,inplace=True)

# Apply Imputation to Fill In Null Values for Missing Data Values
titanic_testing_data_cleaned.fillna(method='ffill', inplace=True)
titanic_testing_data_cleaned.fillna(method='bfill', inplace=True)

# 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_
```

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

In [61]:
```
# 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[61]:

	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	35.0	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	2.0	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	31.0	0	0	2649	7.2250	

(891, 14)

In [62]:
```
# 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[62]:

	Missing Count	Percentage Missing	Column Type

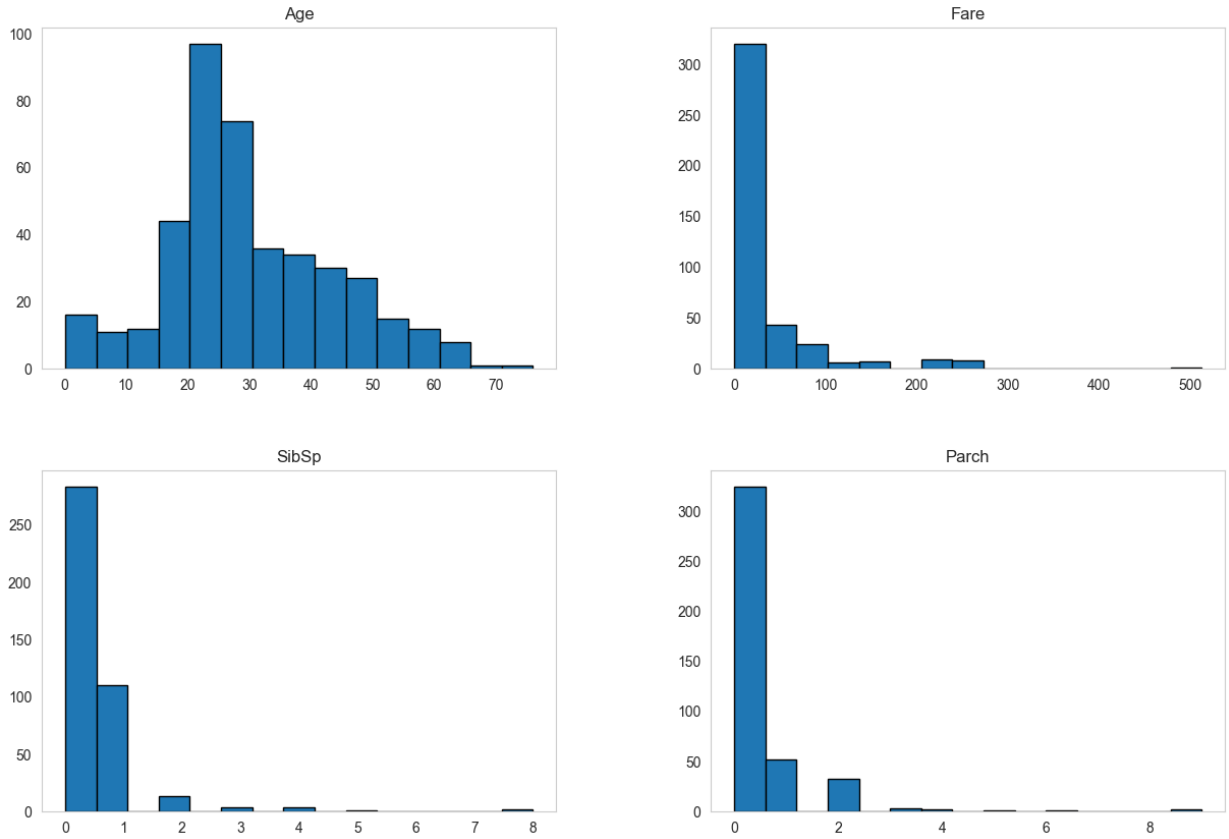
Conduct exploratory data analysis on the variables in the testing dataframe to confirm that all the values appear to be reasonable (to proactively address data value errors if needed).

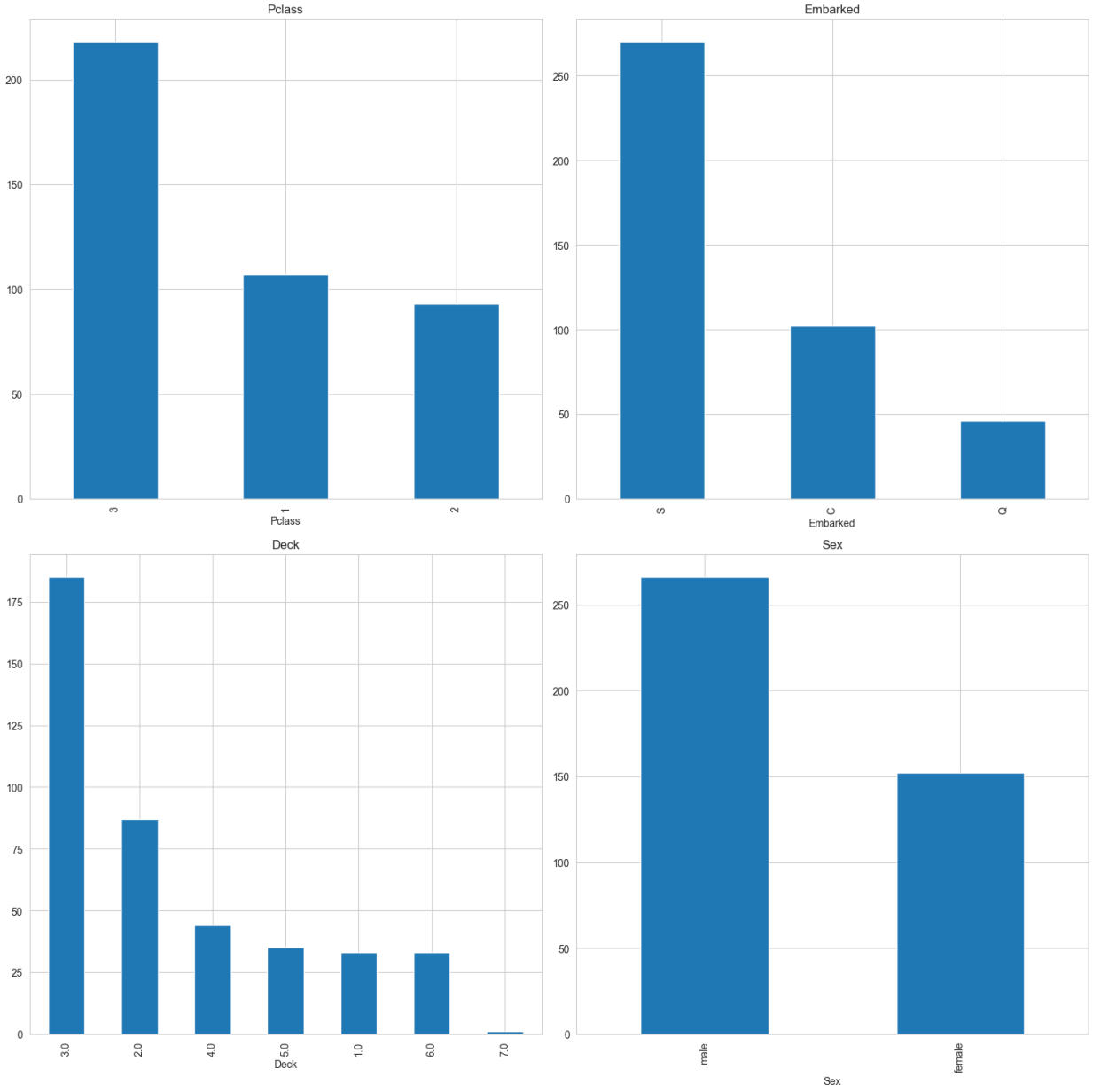
In [63]:
```
# Numeric Variable Visualizations
titanic_testing_data_cleaned[numeric_variables].describe()

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[63]:

	Age	Fare	SibSp	Parch
count	418.000000	418.000000	418.000000	418.000000
mean	30.125000	35.560845	0.447368	0.392344
std	13.905601	55.856972	0.896760	0.981429
min	0.170000	0.000000	0.000000	0.000000
25%	22.000000	7.895800	0.000000	0.000000
50%	27.000000	14.454200	0.000000	0.000000
75%	39.000000	31.471875	1.000000	0.000000
max	76.000000	512.329200	8.000000	9.000000

Out[63]:
```
array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Fare'}>],
       [<Axes: title={'center': 'SibSp'}>,
        <Axes: title={'center': 'Parch'}>]], dtype=object)
```
Out[63]:
```
<Axes: title={'center': 'Pclass'}, xlabel='Pclass'>
```
Out[63]:
```
<Axes: title={'center': 'Embarked'}, xlabel='Embarked'>
```
Out[63]:
```
<Axes: title={'center': 'Deck'}, xlabel='Deck'>
```
Out[63]:
```
<Axes: title={'center': 'Sex'}, xlabel='Sex'>
```

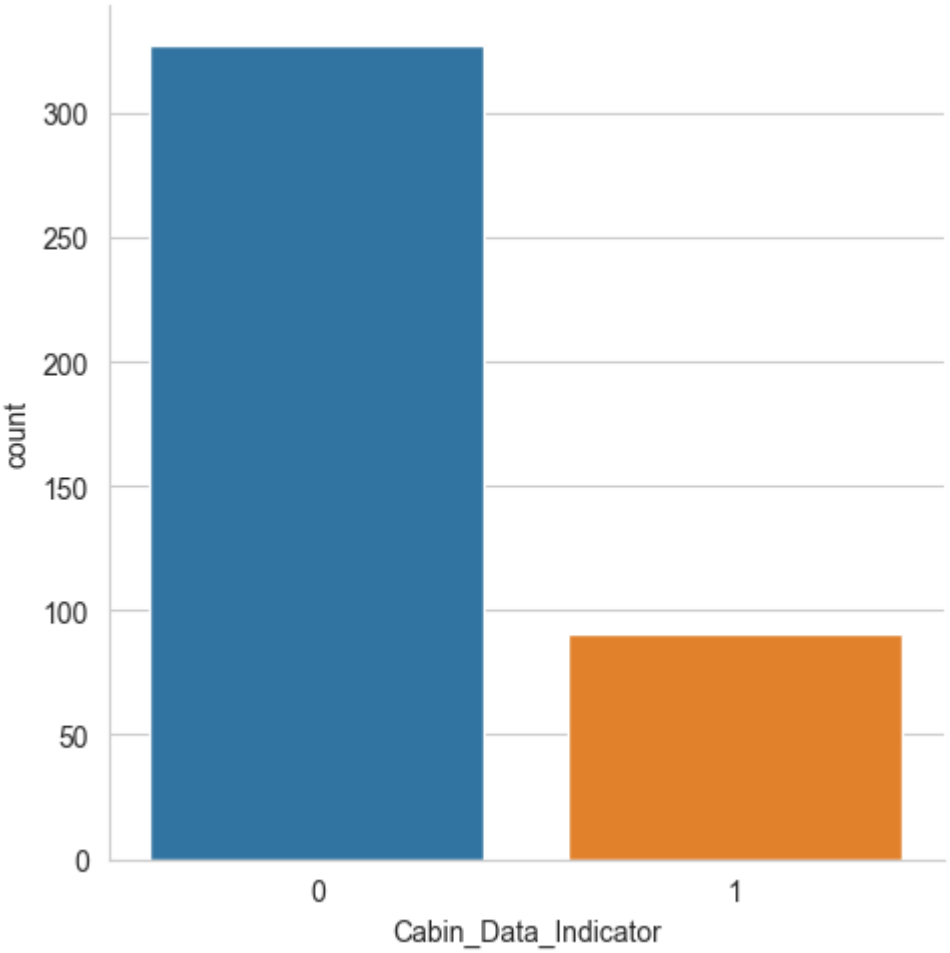


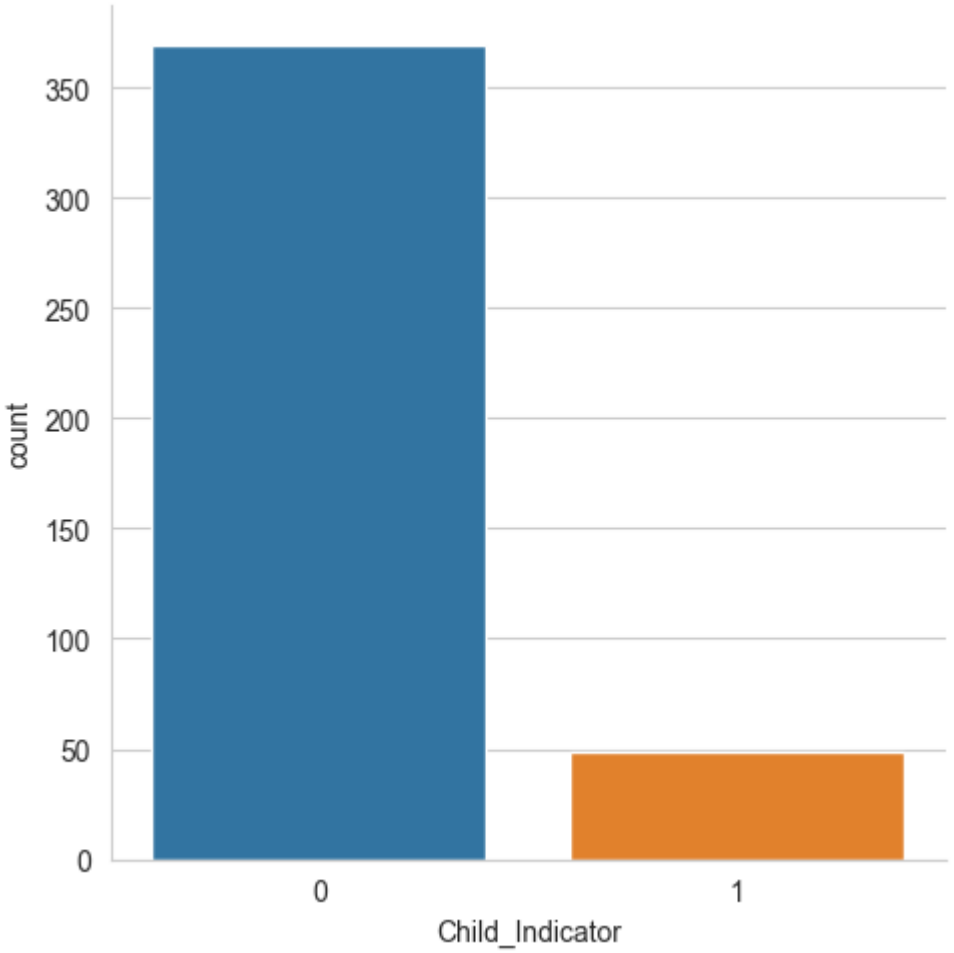


In [64]:
```
# Indicator Variable Visualizations
indicator_predictors = ['Cabin_Data_Indicator', 'Child_Indicator']


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

fig.tight_layout()
```

Out[64]: <seaborn.axisgrid.FacetGrid at 0x23a7d7b1940>

Out[64]: <seaborn.axisgrid.FacetGrid at 0x23a7ad2e190>


300
250
200
150
100
50
0
count
0
1
Cabin_Data_Indicator



Apply Random Forest Model to Test Data

Conduct data cleaning exercises that are specific to the Random Forest Model

In [65]:
```
# Create a copy of the testing dataset
rf_testing_df = titanic_testing_data_cleaned.copy(deep=True)

# Drop variables from the training dataset that we do not want to be included in the R
rf_testing_df.drop(['Name', 'Ticket'], axis=1, inplace=True)

# Encode the sex and embarked variables
rf_testing_df['Female_Indicator'] = np.where(rf_testing_df['Sex'] == 'female', 1, 0)
rf_testing_df.drop(['Sex'], axis=1, inplace=True)

rf_testing_df['Embarked_C_Indicator'] = np.where(rf_testing_df['Embarked'] == 'C', 1,
rf_testing_df['Embarked_Q_Indicator'] = np.where(rf_testing_df['Embarked'] == 'Q', 1,
rf_testing_df.drop(['Embarked'], axis=1, inplace=True)
```

Apply the Random Forest Model to the Test Dataframe

In [66]:
```
# Create a dataframe for predictor variables in the test dataframe for random forest m
rf_testing_x = rf_testing_df.drop(columns=['PassengerId'])

# Apply the Random Forest model to the test dataset
y_test_predictions_rf = best_rf.predict(rf_testing_x)
```

```
# Put the random forest predictions into a Pandas dataframe
prediction_df_rf = pd.DataFrame(y_test_predictions_rf, columns=['Survived'])

# Add the PassengerId column to the front of the random forest predictions dataframe
prediction_df_rf.insert(0, 'PassengerId', rf_testing_df['PassengerId'])

#output predictions to csv
#prediction_df_rf.to_csv('test_predictions_random_forest_v1.csv', index=False)
```

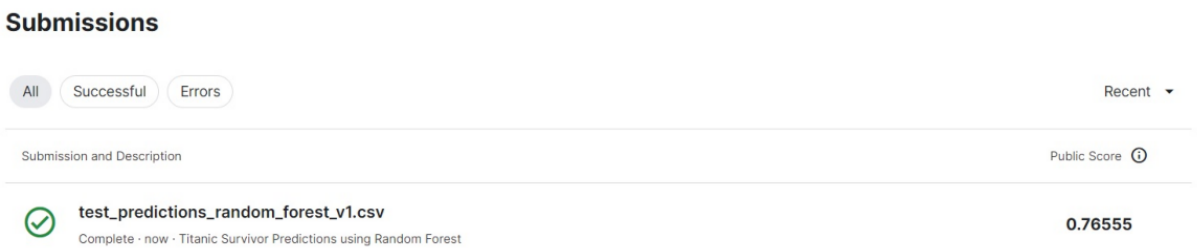
Display the Kaggle Results from the Random Forest Model

In [67]:
```
# Display the kaggle results associated with the Random Forest Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_Random_Forest_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[67]:
```
<Figure size 1500x1500 with 0 Axes>
```

Out[67]:
```
<matplotlib.image.AxesImage at 0x23a7c9607f0>
```

Out[67]:
```
(-0.5, 1496.5, 330.5, -0.5)
```



Apply Boosting Models to Test Data

In [68]:
```
# Create a dataframe for predictor variables in the test dataframe for random forest m
boosting_testing_x = rf_testing_df.drop(columns=['PassengerId'])

# Apply the Random Forest model to the test dataset
y_test_predictions_gb = gb_tuned_model.predict(boosting_testing_x)

# Put the random forest predictions into a Pandas dataframe
prediction_df_gb = pd.DataFrame(y_test_predictions_gb, columns=['Survived'])

# Add the PassengerId column to the front of the random forest predictions dataframe
prediction_df_gb.insert(0, 'PassengerId', rf_testing_df['PassengerId'])

#output predictions to csv
#prediction_df_gb.to_csv('test_predictions_gb_v1.csv', index=False)
```

In [69]:
```
# Display the kaggle results associated with the Gradient Boosting Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_GB_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[69]: <Figure size 1500x1500 with 0 Axes>

Out[69]: <matplotlib.image.AxesImage at 0x23a7c9ba9a0>

Out[69]: (-0.5, 860.5, 104.5, -0.5)

In [70]:
```
# Apply the Random Forest model to the test dataset
y_test_predictions_xgb = xgb_tuned_model.predict(boosting_testing_x)

# Put the random forest predictions into a Pandas dataframe
prediction_df_xgb = pd.DataFrame(y_test_predictions_xgb, columns=['Survived'])

# Add the PassengerId column to the front of the random forest predictions dataframe
prediction_df_xgb.insert(0, 'PassengerId', rf_testing_df['PassengerId'])

#output predictions to csv
#prediction_df_xgb.to_csv('test_predictions_xgb_v1.csv', index=False)
```

In [71]:
```
# Display the kaggle results associated with the Gradient Boosting Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Titanic_XGB_Kaggle_Results_v1.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[71]: <Figure size 1500x1500 with 0 Axes>

Out[71]: <matplotlib.image.AxesImage at 0x23a7cb713a0>

Out[71]: (-0.5, 881.5, 105.5, -0.5)

Apply Extra Trees to Test Data

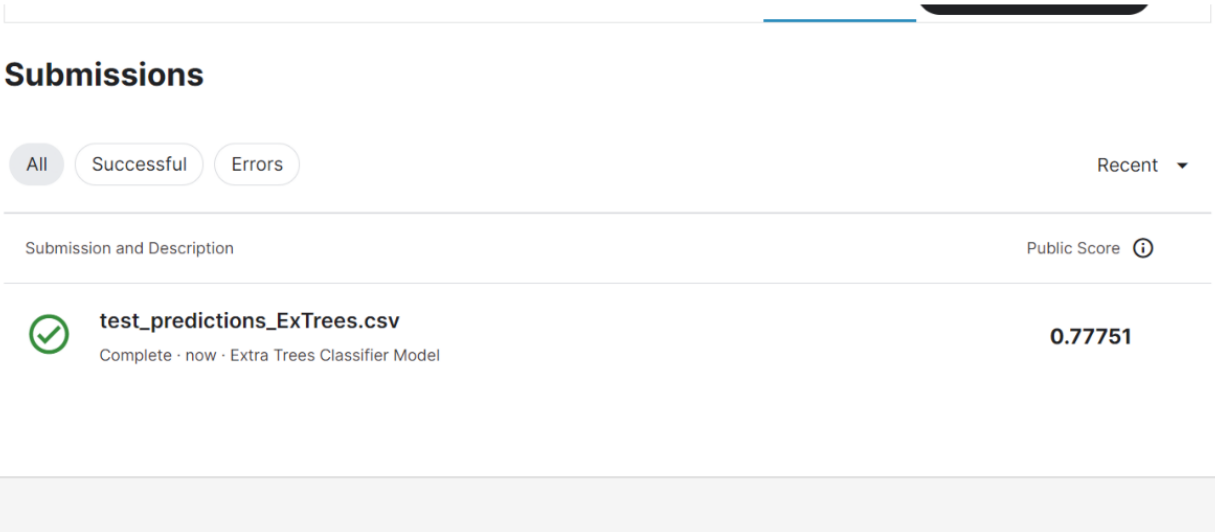
In [72]:
```
# Create a dataframe for predictor variables in the test dataframe for Extra Trees mod
ExTrees_testing = rf_testing_df.drop(columns=['PassengerId'])

# Apply the ExtraTrees model to the test dataset
y_test_predictions_ExTrees = ET_classifier2.predict(ExTrees_testing)

# Put the Extra Trees predictions into a Pandas dataframe
prediction_df_ExTrees = pd.DataFrame(y_test_predictions_ExTrees, columns=['Survived'])

# Add the PassengerId column to the front of the ExtraTrees predictions dataframe
prediction_df_ExTrees.insert(0, 'PassengerId', rf_testing_df['PassengerId'])

#output predictions to csv
#prediction_df_ExTrees.to_csv('test_predictions_ExTrees.csv', index=False)
```

In [73]:
```
# Display the kaggle results associated with the Extra Trees Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('ExtraTrees_Classifier_KaggleSub.png')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[73]:
```
<Figure size 1500x1500 with 0 Axes>
```

Out[73]:
```
<matplotlib.image.AxesImage at 0x23a7cb96df0>
```

Out[73]:
```
(-0.5, 1377.5, 491.5, -0.5)
```



In [ ]: