Data Analytics Lab: Assignment-2
A Mathematical Essay on Logistic Regression

Arjav Singh
Metallurgical and Materials Engineering
Indian Institute of Technology Madras
Chennai, India
mm20b007@smail.iitm.ac.in

***Abstract*—This study focuses on predicting what group of people were more likely to survive the most infamous shipwreck in history, the sinking of the RMS Titanic, than others based on Age, Socioeconomic data, and other factors. A logistic regression model is used to model the importance of these factors and predict the probability of individuals surviving.**

***Index Terms*—Introduction, Logistic Regression, Data, Problem, Conclusion**

I. Introduction

Classification is the process of mapping features to different data classes or categories based on the learning from the input data. Logistic Regression is on the Machine Learning algorithms under the Supervised Learning technique, used for predicting the categorical dependent variable using the set of independent variables. It is a significant Machine Learning algorithm because it can provide probabilities between 0 and 1 and classify new data using continuous or discrete datasets by choosing a cutoff value and classifying the outputs with probability greater than the cutoff as one class, below the cutoff as other.

In the given problem, Logistic Regression is utilized to classify people into groups based on age, socioeconomic data, and other factors. The objective is to determine which group is most likely to survive the RMS Titanic's tragic sinking.

II. Logistic Regression

In the field of statistics, the logistic or logit model serves as a statistical framework utilized for the characterization of event likelihood. It achieves this by expressing the logarithm of the odds for the event as a linear amalgamation of one or more independent variables. Within the scope of regression analysis, logistic regression, commonly called logit regression, finds application in estimating coefficients within the linear combination that defines a logistic model. In binary logistic regression, a singular binary dependent variable is present, depicted by an indicator variable, wherein the two possible values are denoted as "0" and "1." Meanwhile, the independent variables can be binary (comprising two classes represented by indicator variables) or continuous (entailing real numerical values).

A. Formulation

A standard logistic function is a sigmoid function, which takes real input, t, and outputs a value between 0 and 1. This is interpreted as taking input log odds and having output probability for the logit. The standard logistic function $\sigma : \mathbb{R} \to (0, 1)$ is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Let us assume that t is a linear function of a single explanatory variable x. We can then express t as follows:

$$t = \beta_0 + \beta_1 x$$

And the general logistic function $p : \mathbb{R} \to (0, 1)$ can now be written as:

$$p(x) = \sigma(t) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

In the logistic model, the probability denoted as $p(x)$ is interpreted as the likelihood of the dependent variable Y being a success or case rather than a failure or non-case. It becomes evident that the response variables Y_i exhibit variability in their distribution; that is, $P(Y_i = 1 \mid X)$ differs from one data point X_i to another while maintaining independence given the design matrix X and shared parameters β.

One can now introduce the logit function, represented as g, which serves as the inverse of the standard logistic function, σ. The logit function's properties are evident through the following relationships:

$$g(p(x)) = \sigma^{-1}(p(x)) = \operatorname{logit}(p(x)) = \ln\left(\frac{p(x)}{1 - p(x)}\right)$$
$$= \beta_0 + \beta_1 x$$

Equivalently, after applying the exponential function to both sides, we obtain the odds:

$$\frac{p(x)}{1 - p(x)} = e^{\beta_0 + \beta_1 x}$$

In the case where t is a linear combination of multiple explanatory variables, we can express t as:

$$t = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

When this is used in the equation relating the log odds of the success to the values of the predictors, the linear regression converts to multiple regression with m explanatory variables, the parameters β_j for all $j = 0, 1, 2, \ldots, m$ are all estimated. Again the more versatile equation is:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$$

and

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m)}}$$

where usually $b = e$.

B. Parameter Estimation

Consider a generalized linear model function parameterized by θ,

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} = \mathrm{P}(Y = 1 | X; \theta)$$

Therefore,

$$\mathrm{P}(Y = 0 | X; \theta) = 1 - h_\theta(X)$$

and since $Y \in \{0, 1\}$, we see that $\mathrm{P}(y|X;\theta)$ is given by

$$\mathrm{P}(y|X;\theta) = h_\theta(X)^y (1 - h_\theta(X))^{1-y}$$

We now calculate the likelihood function assuming that all the observations in the sample are independently Bernoulli distributed,

$$L(\theta|y;x) = \mathrm{P}(Y|X;\theta) = \prod_{i=1}^{N} \mathrm{P}(y_i|x_i;\theta)$$

$$L(\theta|y;x) = \prod_{i=1}^{N} h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{1-y_i}$$

Typically, the log-likelihood is maximized,

$$\log L(\theta|y;x) = \sum_{i=1}^{N} \log \mathrm{P}(y_i|x_i;\theta)$$

Which is maximized using optimization techniques such as gradient descent to get the parameters' values.

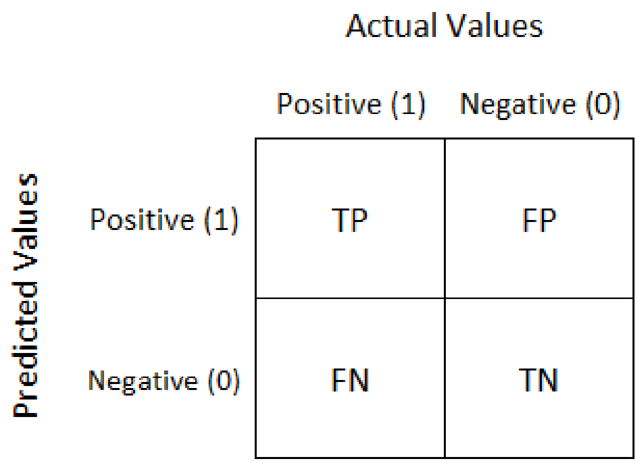


Fig. 1. Confusion Matrix.

C. Metrics for model evaluation

1) **Confusion Matrix**: It is used to summarize the performance of a classification algorithm on a set of test data for which the true values are previously known. Sometimes it is also called an error matrix. Terminologies of the Confusion matrix (Figure 1) are:
   - **True Positive**: TP means the model predicted yes, and the actual answer is also yes.
   - **True negative**: TN means the model predicted no, and the actual answer is also no.
   - **False positive**: FP means the model predicted yes, but the actual answer is no.
   - **False negative**: FN means the model predicted no, but the actual answer is yes.

   The rates calculated using the Confusion Matrix are:
   a) **Accuracy**: (TP+TN/Total) tells about overall how classifier Is correct.
   b) **True positive rate**: TP/(actual yes) it says about how much time yes is predicted correctly. It is also called "sensitivity" or "recall."
   c) **False positive rate**: FP/(actual number) says how much time yes is predicted when the actual answer is no.
   d) **True negative rate**: TN/(actual number) says how much time no is predicted correctly, and the actual answer is also no. It is also known as "specificity."
   e) **Misclassification rate**: (FP+FN)/(Total) It is also known as the error rate and tells about how often our model is wrong.
   f) **Precision**: (TP/ (predicted yes)) If it predicts yes, then how often is it correct.
   g) **Prevalence**: (actual yes /total) how often yes condition actually occurs.
2) **ROC curve (Receiver Operating Characteristic)**: The Receiver Operating Characteristic (ROC) curve is a

useful tool for assessing a model's performance by examining the trade-offs between its True Positive (TP) rate, also known as sensitivity, and its False Negative (FN) rate, which is the complement of specificity. This curve visually represents these two parameters.
The Area Under the Curve (AUC) metric to summarize the ROC curve concisely. The AUC quantifies the area under the ROC curve. In simpler terms, it measures how well the model can distinguish between positive and negative cases. A higher AUC indicates better classifier performance.
In essence, AUC categorizes model performance as follows:

- If AUC = 1, the classifier correctly distinguishes between all the Positive and Negative class points.
- If 0.5¡ AUC ¡ 1, the classifier will distinguish the positive class value from the negative one because it finds more TP and TN than FP and FN.
- If AUC = 0.5, the classifier cannot distinguish between positive and negative values.
- If AUC =0, the classifier predicts all positive as negative and negative as positive.

III. Problem

The problem at hand is centered around examining a hypothesis positing differential survival rates among distinct groups of individuals during the tragic sinking of the RMS Titanic. Logistic regression will be employed to investigate this hypothesis, incorporating various features such as age, socioeconomic indicators, and other relevant factors for analysis.

A. Exploratory Data Analysis and Feature Generation

The training dataset used in this study consists of 891 passengers and 12 features. Interpretation of the features is as follows:

- Survival: 0 if the passenger did not survive, 1 if the passenger survived.
- Pclass: Class of the ticket - 1st, 2nd, 3rd
- Sex: Gender of the passenger - male, female.
- Sibsp: Number of Siblings/Spouses.
- Parch: Number of parents/children.
- Ticket Number.
- Fare: Fare paid for the ticket.
- Cabin: Cabin Number.
- Embarked: Port of Embarkment - C, Q, S.

The initial step in the analysis involved the assessment of the survival rate following the shipwreck. As indicated in Figure 2, the data illustrates that a mere 38.4% of individuals managed to survive. Subsequently, an examination was conducted to ascertain the relationship between all features and the 'Survived' variable to categorize individuals into groups demonstrating a higher likelihood of survival.

The analysis commenced with an exploration of the influence of gender on survival rates. As depicted in Figure 3, a conspicuous pattern emerges, highlighting that females

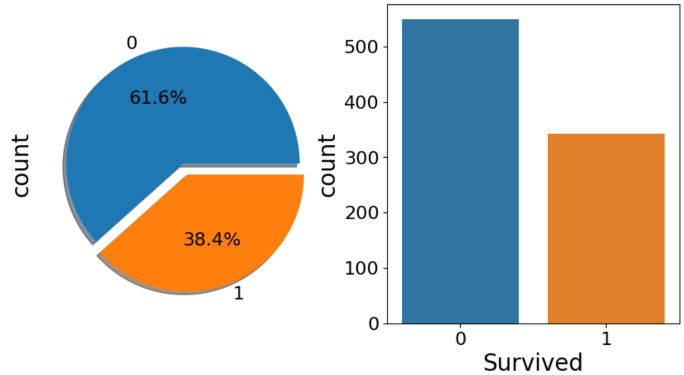


Fig. 2. Percentage of People Survived.

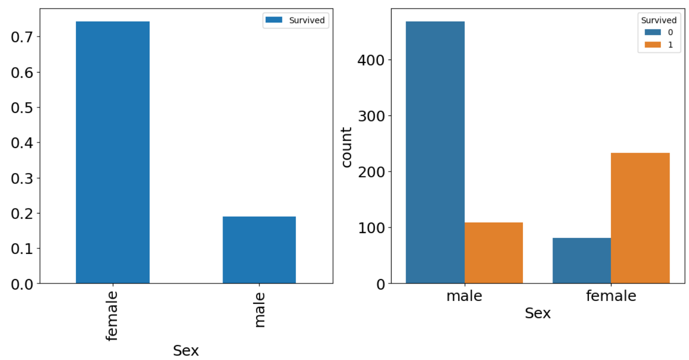


Fig. 3. Relation of Survival with Sex of the passengers.

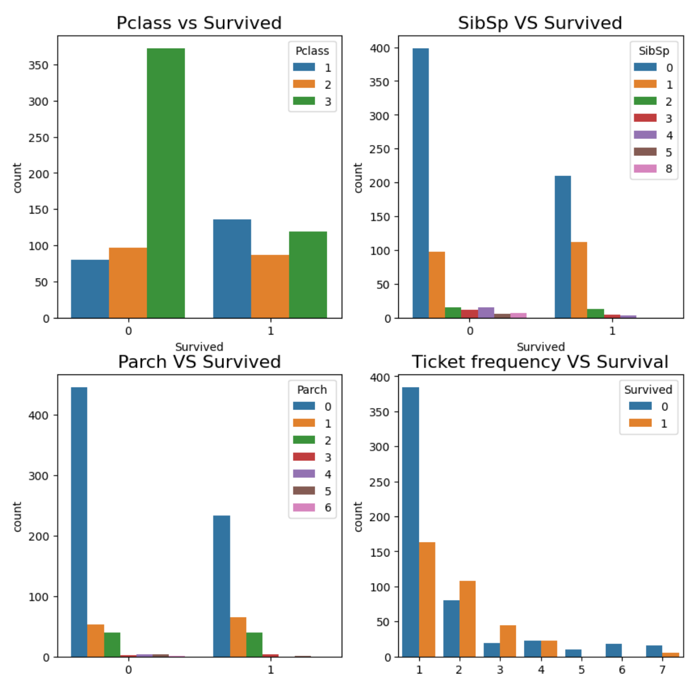


Fig. 4. Relation of Survival with different features.

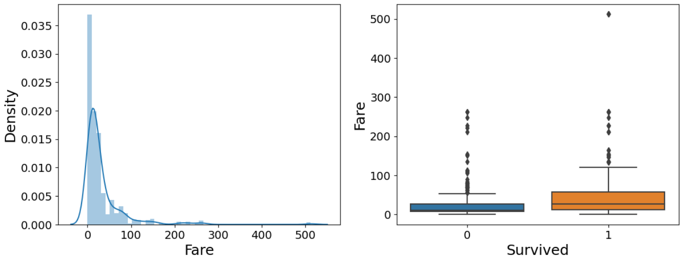


Fig. 5. Relation of Survival with Fares.

exhibited a notably higher likelihood of survival than males. Approximately 70% of females survived, whereas the survival rate among males was markedly lower, at only 20%.

Subsequently, the remaining features were explored, as depicted in Figure 4. The analysis revealed that individuals in passenger class 3 exhibited the lowest likelihood of survival, whereas those in passenger classes 1 and 2 had better survival rates. This observation finds support in the fare distribution, with passengers who paid higher fares being more inclined to survive, as illustrated in Figure 5.

Likewise, passengers who traveled with a solitary companion, forming groups of 2, were more likely to survive than other group sizes. A similar pattern was observed in ticket frequency and family size (created by merging the 'SibSp' and 'Parch' features), as depicted in Figure 6.

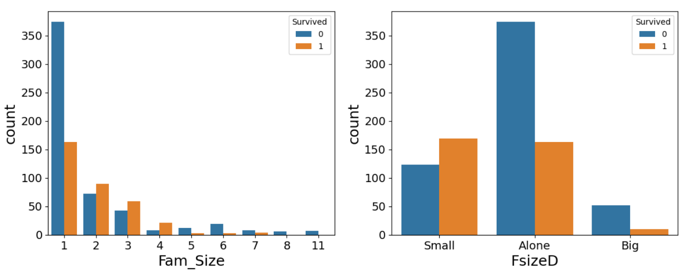


Fig. 6. Relation of Survival with Family Size and Family Type.

A noteworthy observation pertains to missing data within certain variables, as illustrated in Figure 7. Among the 12 features under consideration, it is observed that 'Age' exhibits a notable absence of data, accounting for approximately 19.8%. Similarly, the 'Cabin' feature presents a substantial proportion of missing data, encompassing approximately 77.1%. Moreover, the' Embarked' variable contains only two missing data points.

The Pearson Correlation coefficients between various features were examined to address missing data points. It was observed that the 'Age' feature exhibited the highest absolute correlation with 'Pclass' and 'Sex.' Consequently, a novel feature, denoted as 'title,' was created. Subsequently, the missing values within the 'Age' feature were imputed using the mean values derived from grouping the 'Pclass,' 'title,' and 'Sex' features. The transformation of the 'Age' feature through this engineering process is illustrated in Figure 8 and

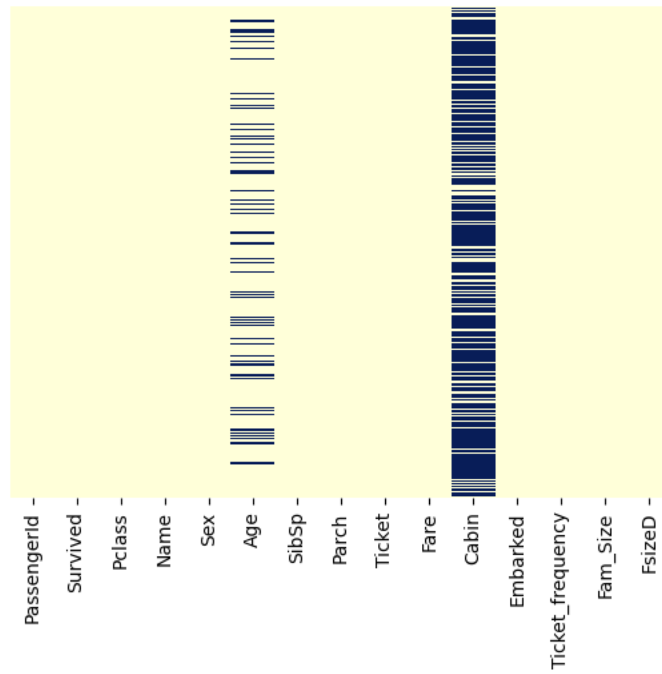


Fig. 7. Heatmap of Missing data.

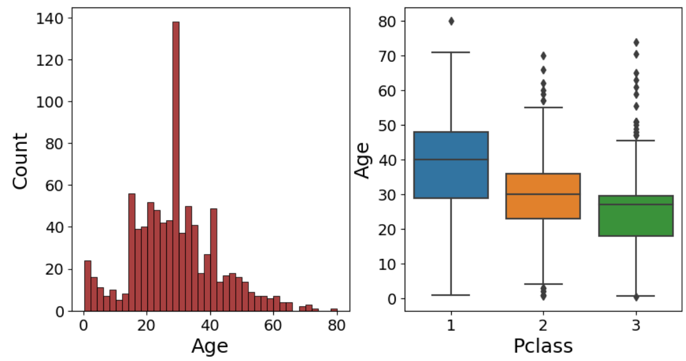


Fig. 8. Age data distribution and Age vs Pclass before data engineering.

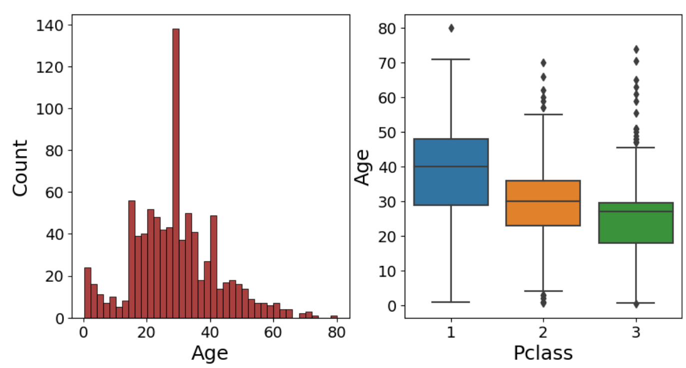


Fig. 9. Age data distribution and Age vs Pclass after data engineering.

Figure 9, depicting its distribution before and after the feature engineering steps.

In the context of the 'Cabin' feature, a new category labeled as 'Z' was introduced to address missing data points, while the outlier category 'T' was replaced with class 'A.' The resultant distribution of passenger class and survival likelihood concerning cabin data, following the feature engineering procedures, is illustrated in Figure 10.

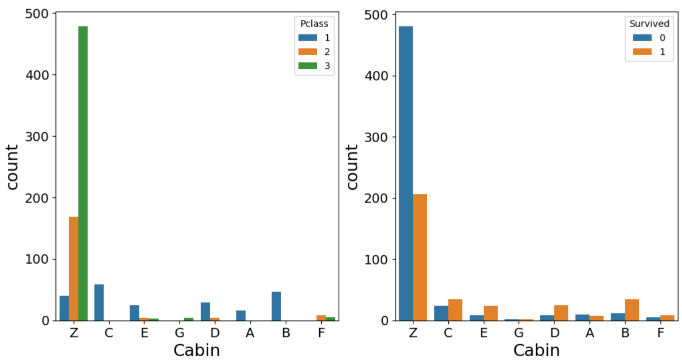


Fig. 10. Passenger class vs Cabin data and Survival chance vs Cabin data after data engineering.

Further analysis of the cabin data to the 'Pclass' and 'survived' features revealed notable insights. It was observed that a significant proportion of individuals assigned to the 'Z' cabin class predominantly belonged to Pclass 3 and exhibited lower chances of survival. Conversely, individuals from other cabin classes were primarily associated with Pclass 1 and demonstrated a higher likelihood of survival. Among the specific cabin classes 'A,' 'B,' 'C,' 'D,' 'E,' and 'F,' it was discerned that class 'A' exhibited the lowest survival rates, while the other cabin classes displayed comparatively better survival probabilities. This finding supports the previous intuition that 'Pclass' may contribute to increased chances of survival, as inferred from the analysis of the cabin data.

The 'Embarked' feature comprises three distinct categories, namely 'S,' 'C,' and 'Q,' as depicted in Figure 11. Notably, most passengers boarded from the 'S' class, with a significant portion belonging to 'Pclass 3.' In contrast, passengers from 'C' seem to have a higher survival rate. This observation may be attributed to the successful rescuing of all passengers from 'Pclass 1' and 'Pclass 2' who boarded from this port. The 'Embark S' location is the primary departure point for most affluent passengers. Nevertheless, the survival prospects for passengers embarking from 'S' are relatively low, primarily due to the unfortunate fate of a substantial portion of 'Pclass 3' passengers, with approximately 81% of them not surviving. Port 'Q' had nearly 95% of its passengers originating from 'Pclass 3.'

In the concluding Exploratory Data Analysis phase, the 'Age' variable is discretized into bins of width 5. Likewise, the 'Fare' variable is discretized into four bins. This binning process transforms these continuous variables into categorical representations, which the machine learning model can more

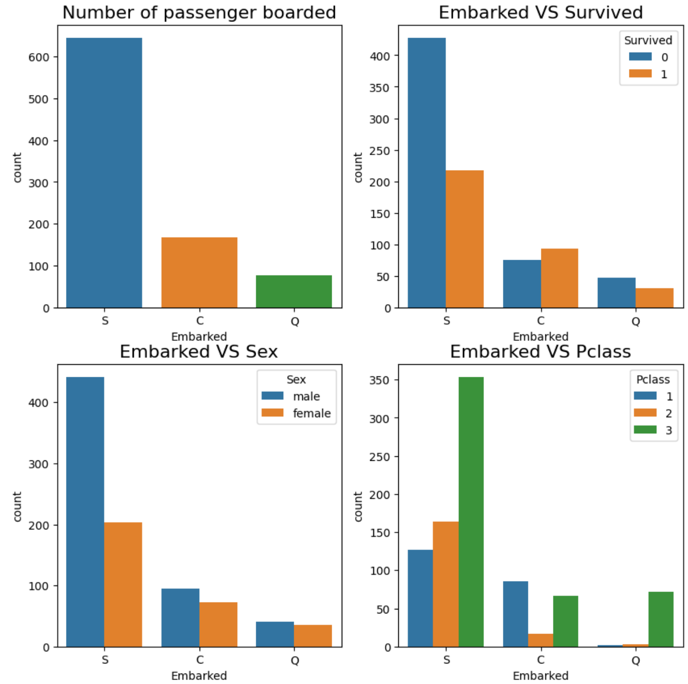


Fig. 11. Embarked feature analysis.

effectively process. Additionally, categorical variables are encoded into dummy variables, imparting meaningful numerical values to these categorical attributes.

B. Statistical Logistic Regression Modelling

The Sklearn library in Python is employed to implement Logistic Regression after preparing all independent variables. Subsequently, the data is split into train and validation sets with a ratio of 10:3.

After fitting the model, the analysis revealed that the model's accuracy is 80.22%. The rest of the values are available in the table below.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0 (Not Survived)	0.79	0.90	0.84	154
1 (Survived)	0.83	0.68	0.74	114
Accuracy	0.80 (Total: 268)			

IV. Implementing the Model to the Test Data

The provided test data have all the features of the train dataset. The methodology is followed for Exploratory Data Analysis, from data pre-processing to missing data handling for features including 'Age' and 'Cabin.' The transformation of the 'Age' feature through this engineering process is illustrated in Figure 12 and Figure 13, depicting its distribution before and after the feature engineering steps.

The model has effectively made predictions regarding the survival likelihood of all passengers, utilizing the parameters on which it was initially trained. This predictive performance is illustrated in Figure 14, which presents the percentage of

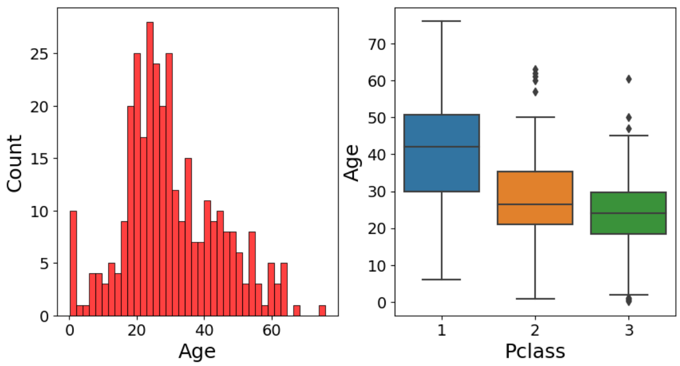


Fig. 12. Test Age data distribution and Age vs Pclass before data engineering.

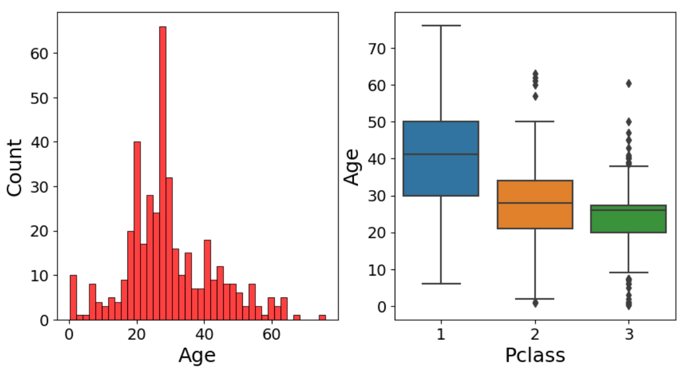


Fig. 13. Test Age data distribution and Age vs Pclass after data engineering.

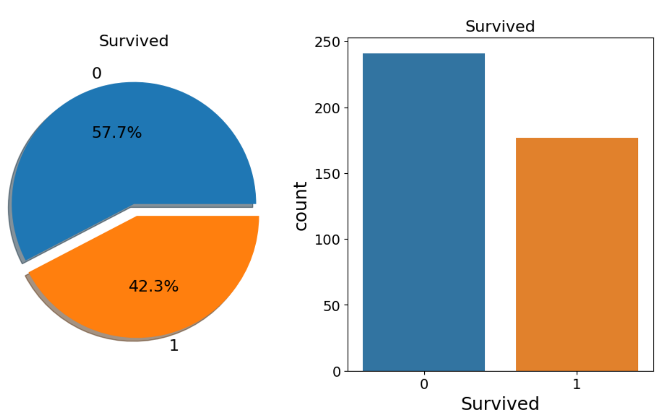


Fig. 14. Predicted percentage of survivors from the test data.

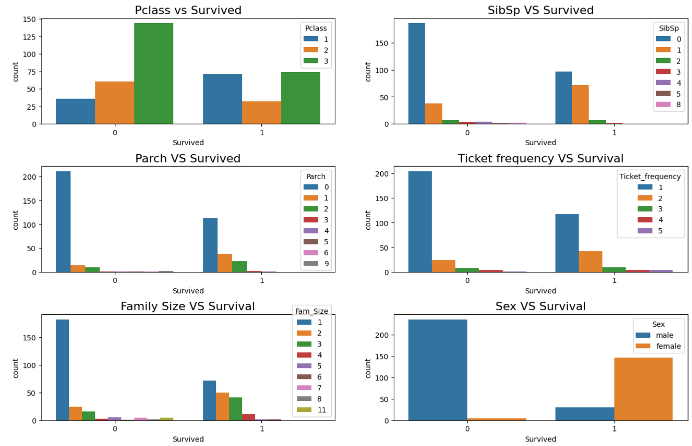


Fig. 15. Features vs Predicted survival chances.

survivors within the test data. Additionally, Figure 15 visually portrays the distribution of features as a barplot, highlighting their correlation with the predicted survival chances.

V. Conclusion

In this study, it is observed that among the passengers on the RMS Titanic, females exhibited a higher likelihood of survival when compared to males. Additionally, passengers occupying Pclass 1, which corresponds to a higher ticket price, demonstrated a greater survival probability than those in other passenger classes. Moreover, younger individuals with family sizes ranging from 2 to 4 members were found to have an increased likelihood of survival, likely attributed to assistance from family members and greater physical agility.

When applied to the test data set, the current model provided similar insights as mentioned above, with an accuracy of 80%, and we observed a similar behavior between features and the survival chance. The study suggests that in the future, the application of non-linear models may offer improved insights into modeling survival rates.

References

[1] JavaTpoint, "Logistic Regression in Machine Learning," JavaTpoint, [Online]. Available: https://www.javatpoint.com/logistic-regression-in-machine-learning.

[2] "Logistic Regression - Model Fitting," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression#Model_fitting.

[3] "Performance Measurement in Logistic Regression," Medium, [Online]. Available: https://meettank29067.medium.com/performance-measurement-in-logistic-regression-8c9109b25278.

[4] "Kaggle Titanic Competition: Missing Values." *Python in Plain English*, Available: https://python.plainenglish.io/kaggle-titanic-competition-missing-values-f3280267b361.

[5] "Kaggle Titanic Challenge: Create Them Features." *Python in Plain English*, Available: https://python.plainenglish.io/kaggle-titanic-challenge-create-them-features-a324ba577812. [Accessed: September 14, 2023].

MM20B007 DAL Assignment 2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

train = pd.read_excel('/content/drive/MyDrive/sem 7/EE5708/Assignment 2/train.xlsx')
test = pd.read_excel('/content/drive/MyDrive/sem 7/EE5708/Assignment 2/test.xlsx')
```

Exploratory Data Analysis and Feature Generation

The given data has 891 datapoints corresponding to each of the 12 features.

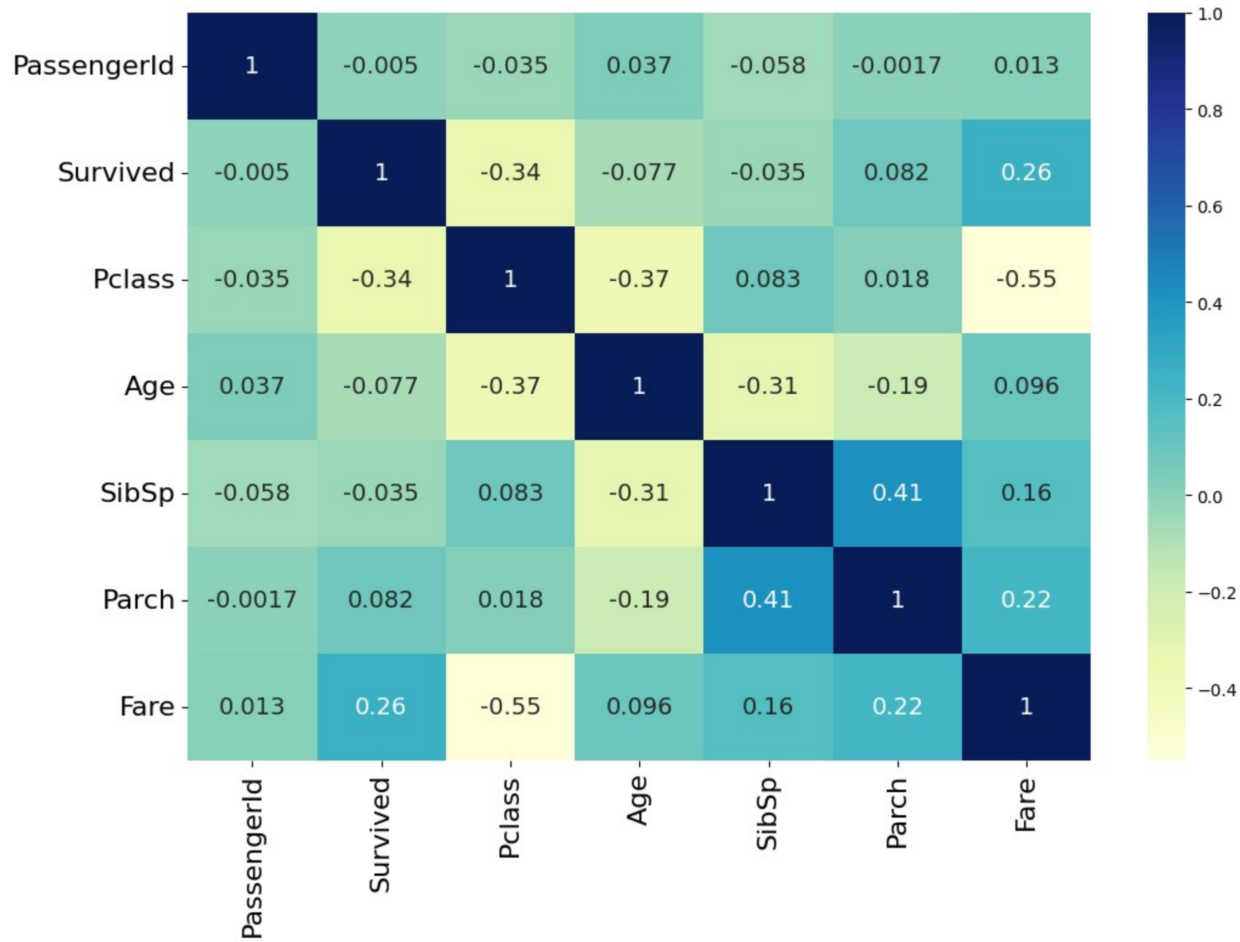
```
train.describe().transpose()
```

```
             count        mean         std   min       25%       50%
75%  \
PassengerId  891.0  446.000000  257.353842  1.00  223.5000  446.0000
668.5
Survived     891.0    0.383838    0.486592  0.00    0.0000    0.0000
1.0
Pclass       891.0    2.308642    0.836071  1.00    2.0000    3.0000
3.0
Age          714.0   29.699118   14.526497  0.42   20.1250   28.0000
38.0
SibSp        891.0    0.523008    1.102743  0.00    0.0000    0.0000
1.0
Parch        891.0    0.381594    0.806057  0.00    0.0000    0.0000
0.0
Fare         891.0   32.204208   49.693429  0.00    7.9104   14.4542
31.0

                  max
PassengerId  891.0000
Survived       1.0000
Pclass         3.0000
Age           80.0000
SibSp          8.0000
Parch          6.0000
Fare         512.3292
```

```
plt.figure(figsize = (12, 8))
sns.heatmap(train.corr(), annot = True, annot_kws={"size": 14}, cmap =
'YlGnBu')
plt.xticks(fontsize = 16, rotation = 90)
plt.yticks(fontsize = 16, rotation = 0)
```

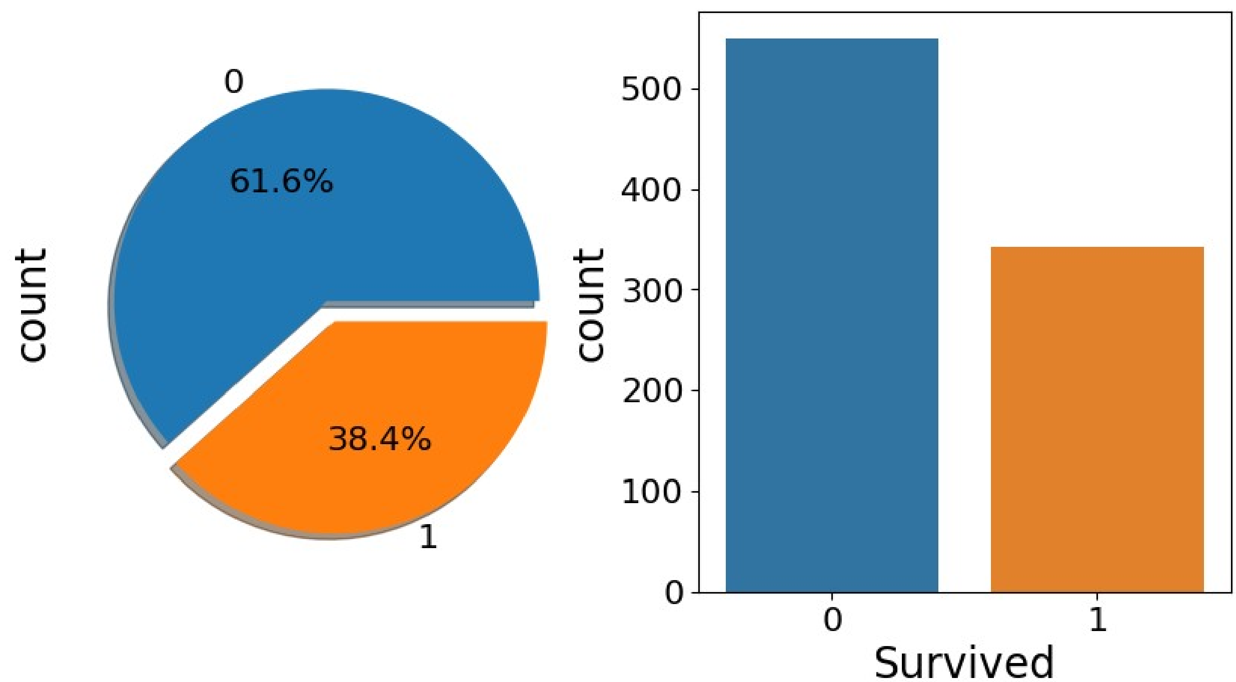
```
<ipython-input-4-3dc8cfd081d0>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(train.corr(), annot = True, annot_kws={"size": 14}, cmap
= 'YlGnBu')
```

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5]),
 [Text(0, 0.5, 'PassengerId'),
  Text(0, 1.5, 'Survived'),
  Text(0, 2.5, 'Pclass'),
  Text(0, 3.5, 'Age'),
  Text(0, 4.5, 'SibSp'),
  Text(0, 5.5, 'Parch'),
  Text(0, 6.5, 'Fare')])
```



Checking the percentage survivals

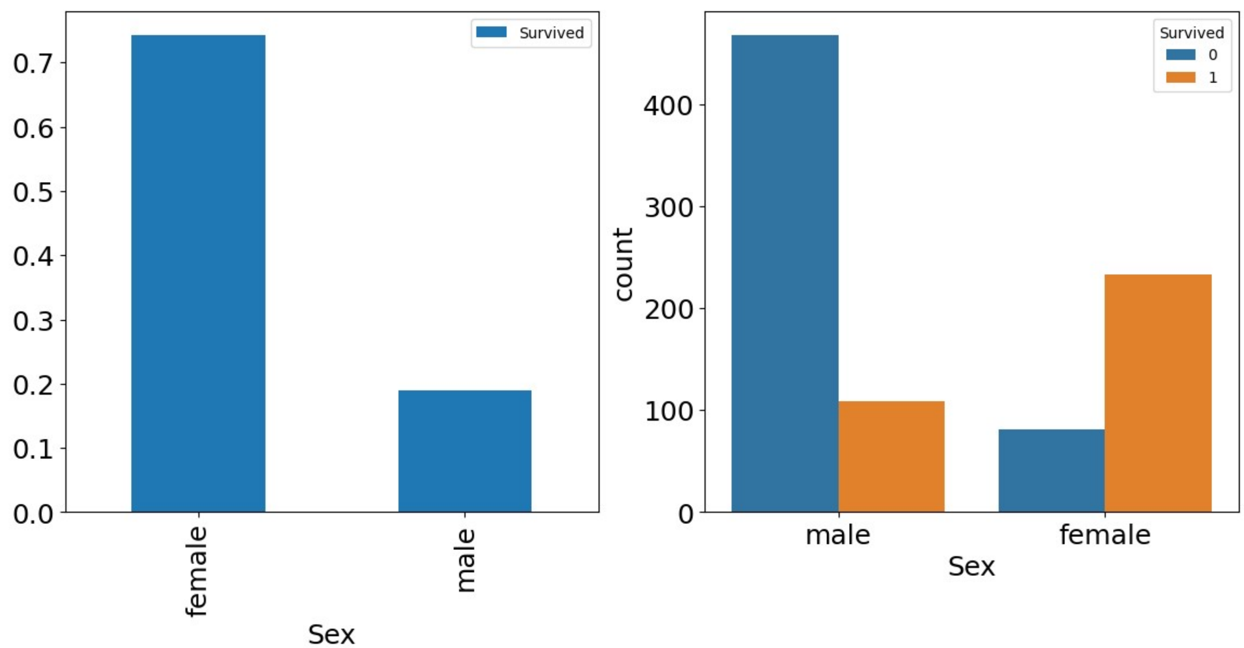
```
f,ax=plt.subplots(1,2,figsize=(10,5))
train['Survived'].value_counts().plot.pie(ax = ax[0],
explode=[0,0.1],autopct='%1.1f%%',shadow=True, textprops={'fontsize':
16})
ax[0].set_ylabel('count')

sns.countplot(x = 'Survived',data = train, ax=ax[1])

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=16)
    a.set_xlabel(a.get_xlabel(), fontsize=20)
    a.set_ylabel(a.get_ylabel(), fontsize=20)

plt.show()
```




```
df = train

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
```

```
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
f, ax = plt.subplots(1, 2, figsize = (14, 6))
data = train
data[['Sex', 'Survived']].groupby(['Sex']).mean().plot.bar(ax = ax[0])
sns.countplot(x = 'Sex', hue='Survived', data = data, ax=ax[1])

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=18)
    a.set_xlabel(a.get_xlabel(), fontsize=18, rotation = 0)
    a.set_ylabel(a.get_ylabel(), fontsize=18, rotation = 90)

plt.show()
```

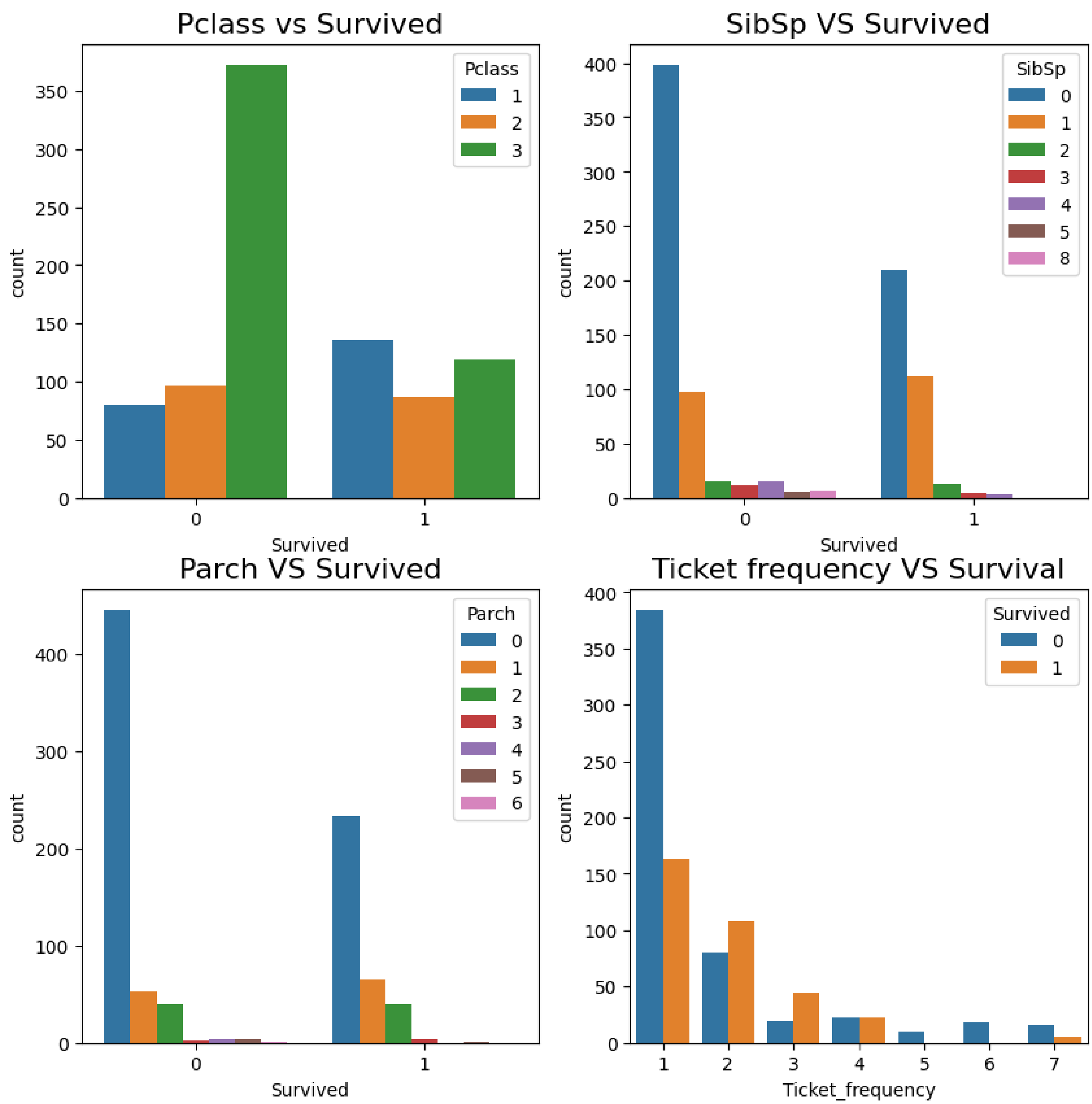


From the barplot male are much less likely to survive (~20%) than female (with a survival chance of ~70%).

```
# Defining the frequency of each ticket
df['Ticket_frequency'] = df.groupby('Ticket')
```

```
['Ticket'].transform('count')

f, ax = plt.subplots(2, 2, figsize = (10, 10))
sns.countplot(x = 'Survived', data = df,hue = 'Pclass', ax = ax[0, 0])
ax[0, 0].set_title('Pclass vs Survived', fontsize = 16)
sns.countplot(x = 'Survived', data = df, hue = 'SibSp', ax = ax[0, 1])
ax[0, 1].set_title('SibSp VS Survived', fontsize = 16)
sns.countplot(x = 'Survived', data = df, hue = 'Parch', ax = ax[1, 0])
ax[1, 0].set_title('Parch VS Survived', fontsize = 16)
sns.countplot(x = 'Ticket_frequency', data = df, hue = 'Survived', ax
= ax[1, 1])
ax[1, 1].set_title('Ticket frequency VS Survival', fontsize = 16)
```

```
Text(0.5, 1.0, 'Ticket frequency VS Survival')
```



1. Most of the people are from class 3 and have lowest survival chance, whereas passangers from class 1 and 2 have better chances.

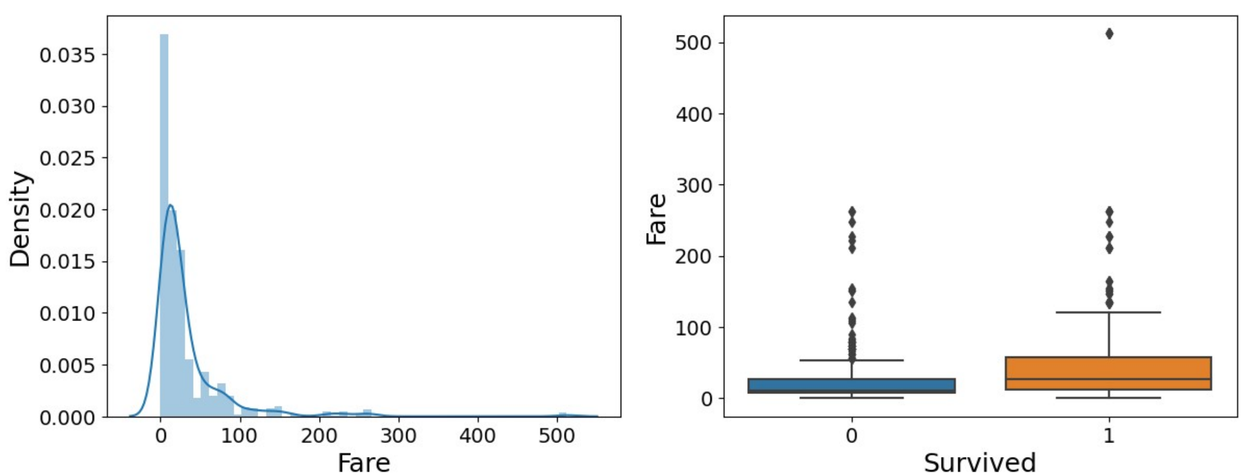
2. From SibSp vs Survived and Parch vs Survived it is clear that about ~70% passengers are alone and person with single partner has better chance of surviving (similar to rose and jack). But it need more refinement.

3. On careful observation we see that different ticket frequency has different rates of survival with the highest being for the group of two and the chance for groups of more than 4 being very less.

Role of Fare price

```
f, ax = plt.subplots(1, 2, figsize = (14, 5))

sns.distplot(df.Fare, ax = ax[0])
# ax[0].set_title('Density Distribution of Fare', fontsize = 18)

sns.boxplot(x = 'Survived', y = 'Fare', data = df, ax = ax[1])
# ax[1].set_title('Fare vs Survived', fontsize = 18)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18)
    a.set_ylabel(a.get_ylabel(), fontsize=18)
```

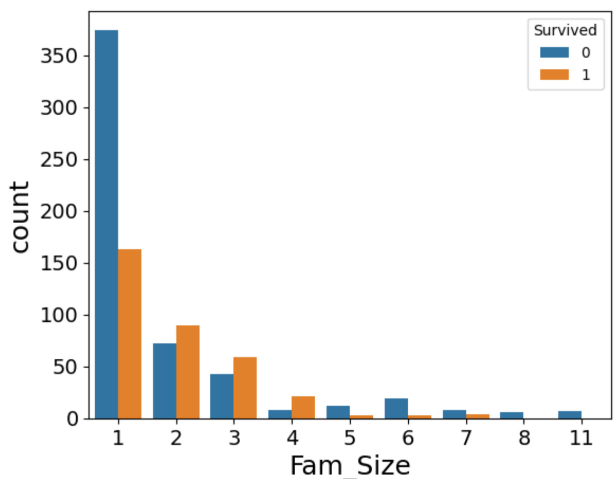
```
<ipython-input-10-724327ff15fe>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Fare, ax = ax[0])
```

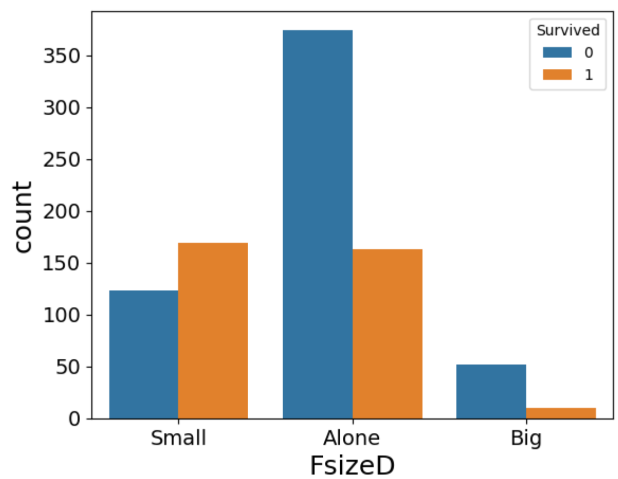


We see that there are people paying too much for their tickets (outliers) and people with higher fares are more likely to survive!

Checking family size because it plays a pivotal role in the survival.

```python
df['Fam_Size'] = df['Parch'] + df['SibSp'] + 1

f, ax = plt.subplots(1, 2, figsize = (14, 5))
sns.countplot(x = 'Fam_Size', data = df, hue = 'Survived', ax = ax[0])
# ax[0].set_title('Family Size VS Survival', fontsize = 18)

# Creating the classes in Fam_size
df.loc[:,'FsizeD']='Alone'
df.loc[(df['Fam_Size']>1),'FsizeD']='Small'
df.loc[(df['Fam_Size']>4),'FsizeD']='Big'

sns.countplot(x='FsizeD',data=df,hue='Survived', ax = ax[1])
# ax[1].set_title('Family Type VS Survival', fontsize = 18)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18)
    a.set_ylabel(a.get_ylabel(), fontsize=18)
```





We see that people with small family size have the highest chance of Surviving, and the ones with a big family, which the lease chance !

The above barplot suggest the same result inspite of the fact that most of the were single passengers their surivival rate is much lower than people with small family.

Handling the Missing Data

```python
train.info()
train.isna().sum()/train.shape[0]*100
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   PassengerId       891 non-null    int64
 1   Survived          891 non-null    int64
 2   Pclass            891 non-null    int64
 3   Name              891 non-null    object
 4   Sex               891 non-null    object
 5   Age               714 non-null    float64
 6   SibSp             891 non-null    int64
 7   Parch             891 non-null    int64
 8   Ticket            891 non-null    object
 9   Fare              891 non-null    float64
 10  Cabin             204 non-null    object
 11  Embarked          889 non-null    object
 12  Ticket_frequency  891 non-null    int64
 13  Fam_Size          891 non-null    int64
 14  FsizeD            891 non-null    object
dtypes: float64(2), int64(7), object(6)
memory usage: 104.5+ KB

PassengerId          0.000000
Survived             0.000000
Pclass               0.000000
Name                 0.000000
Sex                  0.000000
Age                 19.865320
SibSp                0.000000
Parch                0.000000
Ticket               0.000000
Fare                 0.000000
Cabin               77.104377
Embarked             0.224467
Ticket_frequency     0.000000
Fam_Size             0.000000
FsizeD               0.000000
dtype: float64
```

Out of the 12 features Age, Cabin, and Embarked have the missing datapoints.

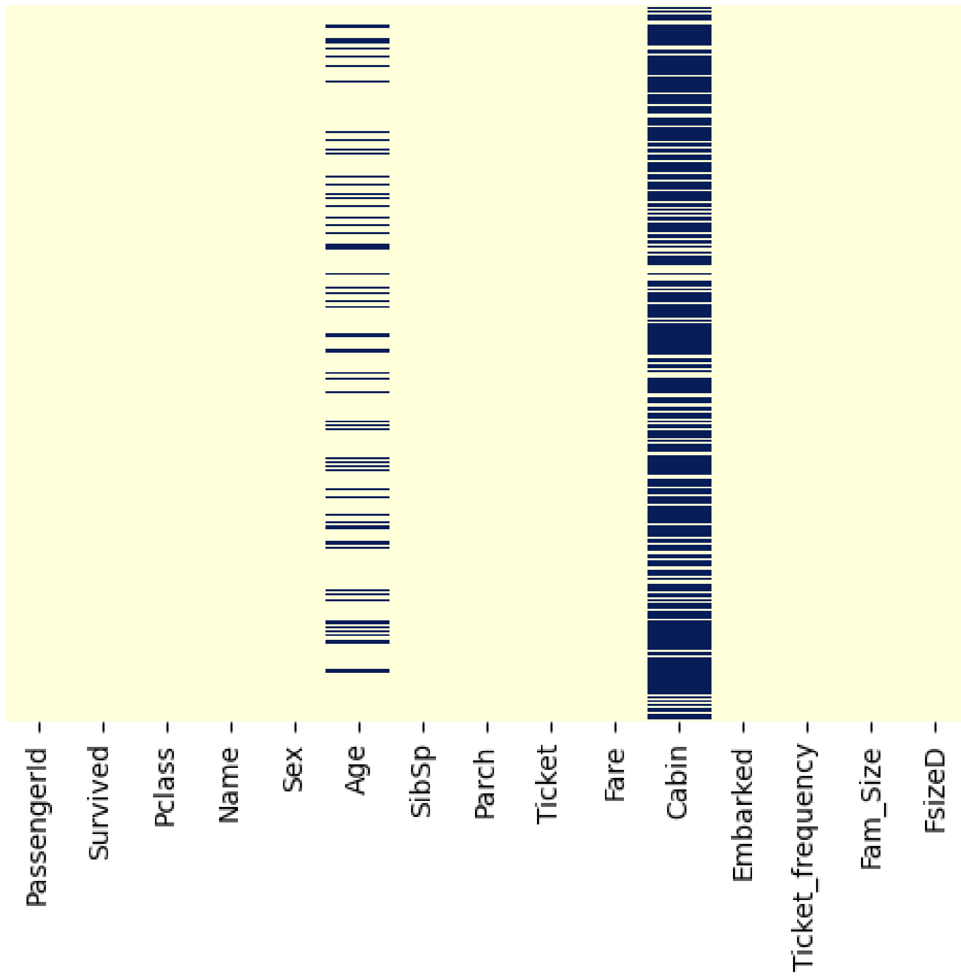
a. Age has 19.8 % data missing.

b. Cabin has 77.1 % data missing.

c. Embarked has 0.22 % data missing.

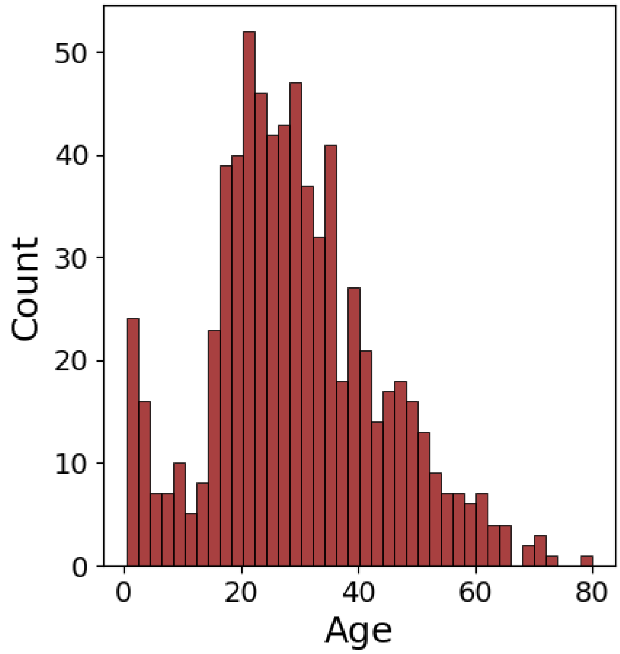
```
sns.heatmap(train.isnull(), yticklabels = False, cbar = False, cmap =
'YlGnBu')
# plt.title('Heatmap of Missing data', fontsize = 16)
```

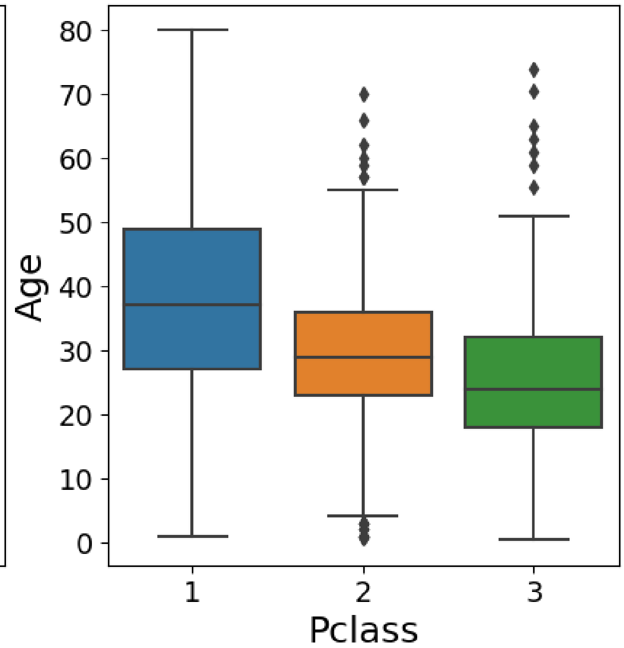
```
<Axes: >
```



Handling Age Data

```
f, ax = plt.subplots(1, 2, figsize = (10, 5))

sns.histplot(df.Age.dropna(), kde=False, color='darkred', bins=40, ax
= ax[0])
# ax[0].set_title('Distribution of Age data', fontsize = 16)

sns.boxplot(x = 'Pclass', y = 'Age', data = df, ax = ax[1])
# ax[1].set_title('Age vs Pclass', fontsize  = 16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18)
    a.set_ylabel(a.get_ylabel(), fontsize=18)
```





We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation).

From the correlation heatmap we can see that age has good absolute correlation with Pclass, SblSp, and Parch.

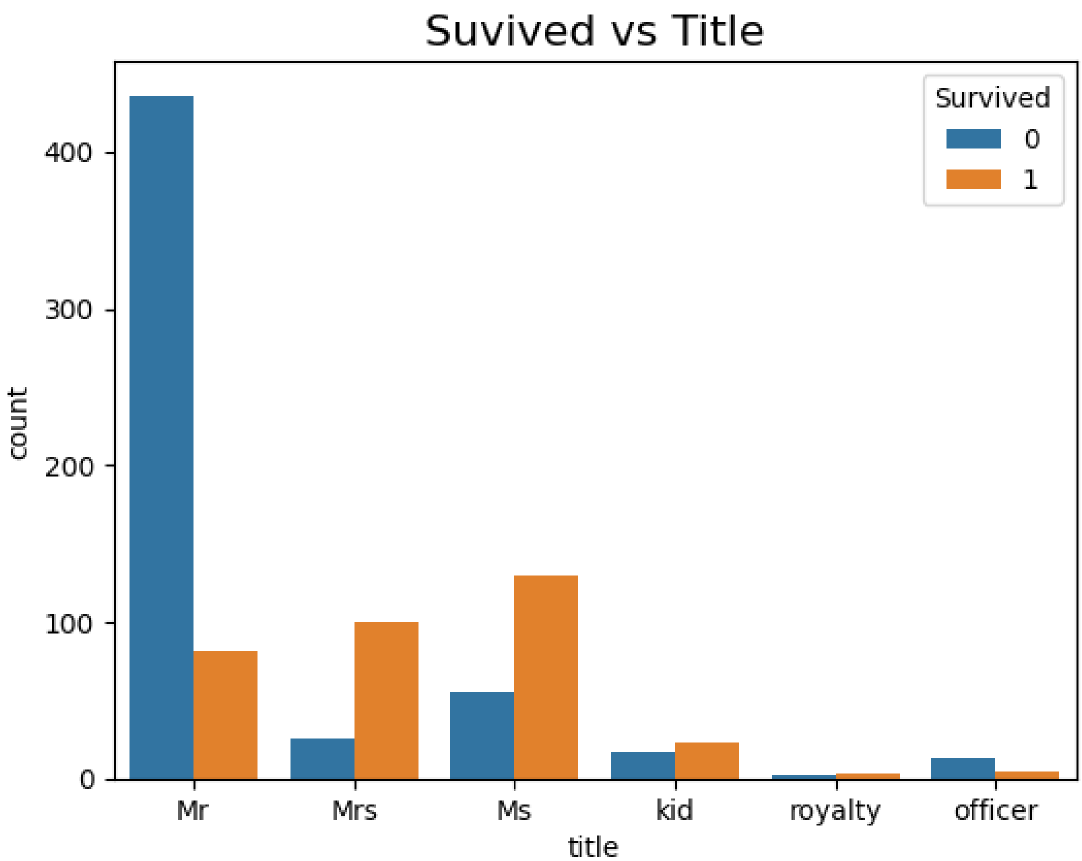
Hypothesis:

1. We will be looking to the name because if a family member survived than there are chances that the others will also survive.
2. We will be looking at the titles as well because survival rate changes with title.

```
# Grouping Title
new_title = {
    'Mr' : 'Mr','Ms' : 'Ms','Mrs' : 'Mrs','Rev' : 'officer','Sir' :
'royalty','theCountess' : 'royalty','Dona' : 'royalty','Capt' :
'officer','Col' : 'officer','Don' : 'royalty','Dr' :
'officer','Jonkheer' : 'royalty','Lady' : 'royalty','Major' :
'officer','Master' : 'kid','Miss' : 'Ms','Mlle' : 'Ms','Mme' : 'Mrs'
}

#Add Title
def add_title(df):
    df['title'] = df['Name'].apply(lambda x: x.split(",")[1])
    df['title'] = df['title'].apply(lambda x: x.split(".")[0])
    df.title = df.title.str.replace(' ', '')

add_title(df)

# Group Title
df['title'] = df['title'].apply(lambda x: new_title[x])
```

```python
# display(pd.DataFrame(df.groupby('title')['PassengerId'].nunique()))

sns.countplot(x = 'title', data = df, hue = 'Survived')
plt.title('Suvived vs Title', fontsize = 16)
```

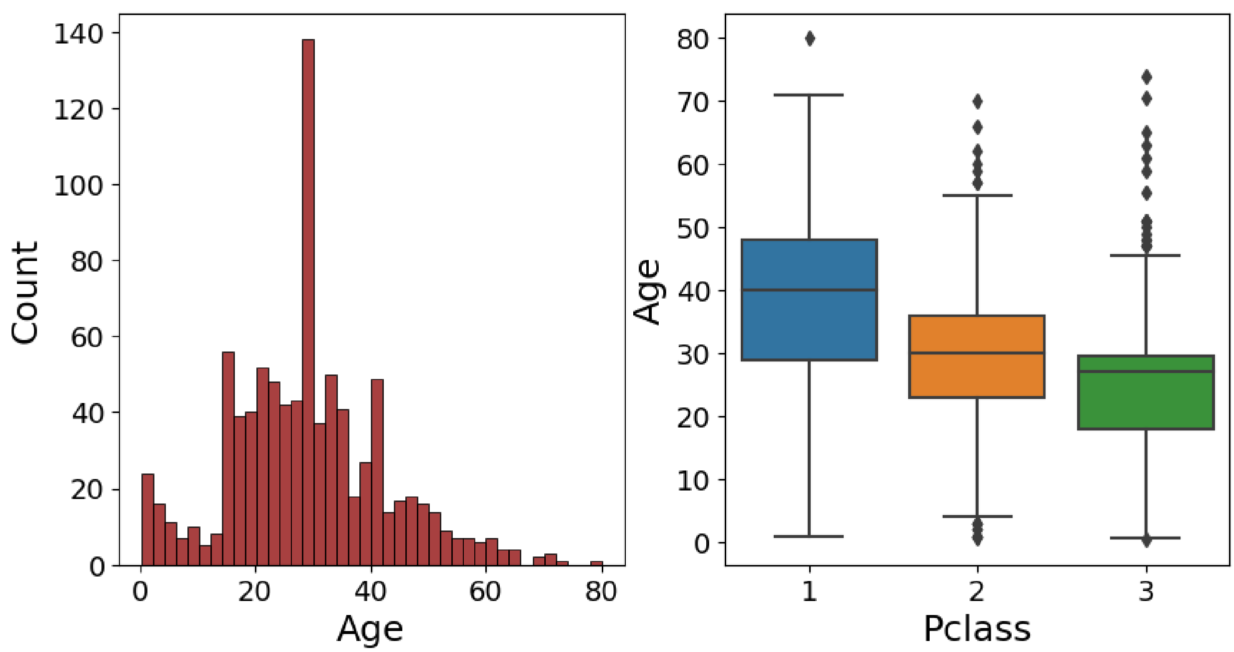
```
Text(0.5, 1.0, 'Suvived vs Title')
```




```python
# Function to Update missing age values
def update_age(params):
    pclass = params[0]
    title = params[1]
    sex = params[2]
    age = params[3]
    if pd.isnull(age):
        age = np.float(age_df[(age_df['title'] == title) &
(age_df["Sex"] == sex) & (age_df['Pclass'] == pclass)]["Age"])
    return age

# Dataframe to group age across Pclass, Title and Sex
age_df = df.groupby(['Pclass','title','Sex']).Age.mean().reset_index()

# Fill missing age
df['Age'] = df[['Pclass', 'title', 'Sex', 'Age']].apply(lambda x:
```

```
update_age(x), axis = 1)

f, ax = plt.subplots(1, 2, figsize = (10, 5))

sns.histplot(df.Age.dropna(), kde=False, color='darkred', bins=40, ax
= ax[0])
# ax[0].set_title('Distribution of Age data', fontsize = 16)

sns.boxplot(x = 'Pclass', y = 'Age', data = df, ax = ax[1])
# ax[1].set_title('Age vs Pclass', fontsize  = 16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18)
    a.set_ylabel(a.get_ylabel(), fontsize=18)
```

```
<ipython-input-16-cde44bc9ecc5>:8: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use
`np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  age = np.float(age_df[(age_df['title'] == title) & (age_df["Sex"] ==
sex) & (age_df['Pclass'] == pclass)]["Age"])
```



Cabin

Cabin has 77 % of it's data missing.

```python
df['Cabin'].unique()
```

```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60',
'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49',
'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91',
'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10',
'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63',
'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F
G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46',
'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17',
'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

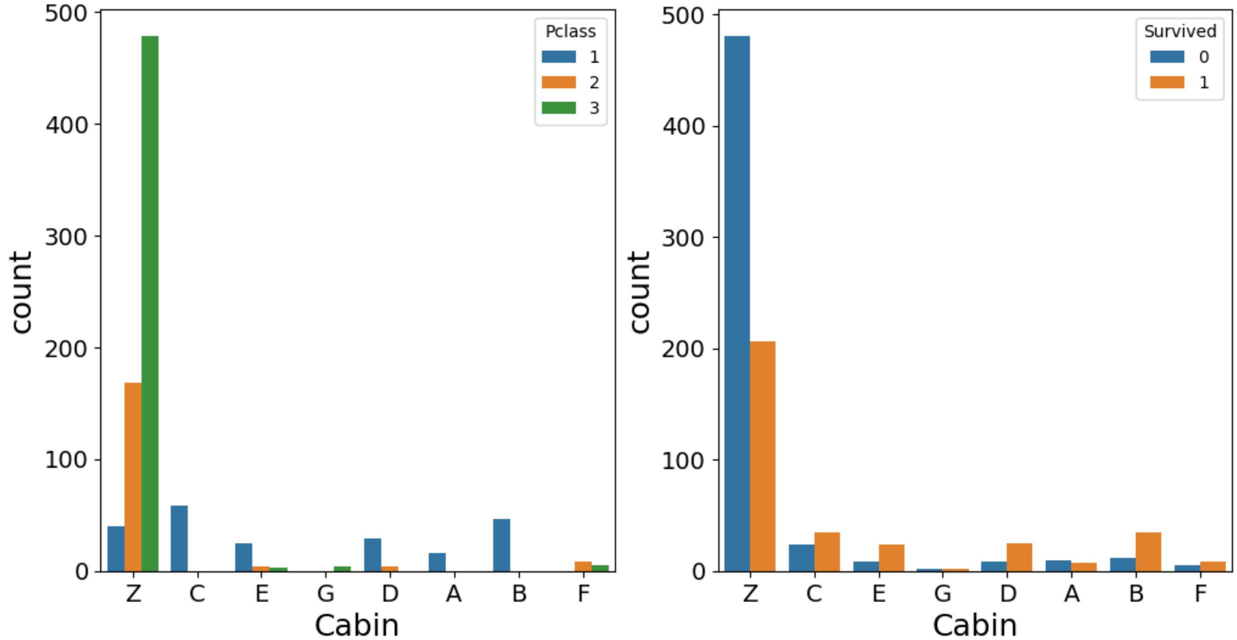
Creating a new cabin class = 'Z'. Replacing the nan value with it and removing the numbers from the given class just taking the alphabet.

```python
df['Cabin'] = df['Cabin'].fillna('Z')
df['Cabin'] = df['Cabin'].apply(lambda s: s[0])
df.loc[train[df['Cabin']=='T'].index,'Cabin']='A'
```

Visualizing the new Cabin data

```python
f, ax = plt.subplots(1, 2, figsize = (12, 6))
sns.countplot(x = 'Cabin', data = df, hue = 'Pclass', ax = ax[0])
# ax[0].set_title('New Cabin data VS Pclass', fontsize = 16)
sns.countplot(x = 'Cabin', data = df, hue = 'Survived', ax = ax[1])
# ax[1].set_title('New Cabin data VS Survived', fontsize = 16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18, rotation = 0)
    a.set_ylabel(a.get_ylabel(), fontsize=18, rotation = 90)
```

```
plt.show()
```



After visualizing the cabin data with respect to Pclass and survived features, we found out that most of the people with 'Z' cabin class belonged to the Pclass 3 and had less chances of survival. People from the other cabin class were majorly from Pclass 1 and had a better chance of survival. Among cabin class A, B, C, D, E, and F A had the lowest chance of survival while others were better off. This indicates that Pclass might be a factor resulting in higher chances of survival,

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   PassengerId       891 non-null    int64
 1   Survived          891 non-null    int64
 2   Pclass            891 non-null    int64
 3   Name              891 non-null    object
 4   Sex               891 non-null    object
 5   Age               891 non-null    float64
 6   SibSp             891 non-null    int64
 7   Parch             891 non-null    int64
 8   Ticket            891 non-null    object
 9   Fare              891 non-null    float64
 10  Cabin             891 non-null    object
 11  Embarked          889 non-null    object
 12  Ticket_frequency  891 non-null    int64
```

```
 13   Fam_Size           891 non-null    int64
 14   FsizeD             891 non-null    object
 15   title              891 non-null    object
dtypes: float64(2), int64(7), object(7)
memory usage: 111.5+ KB
```

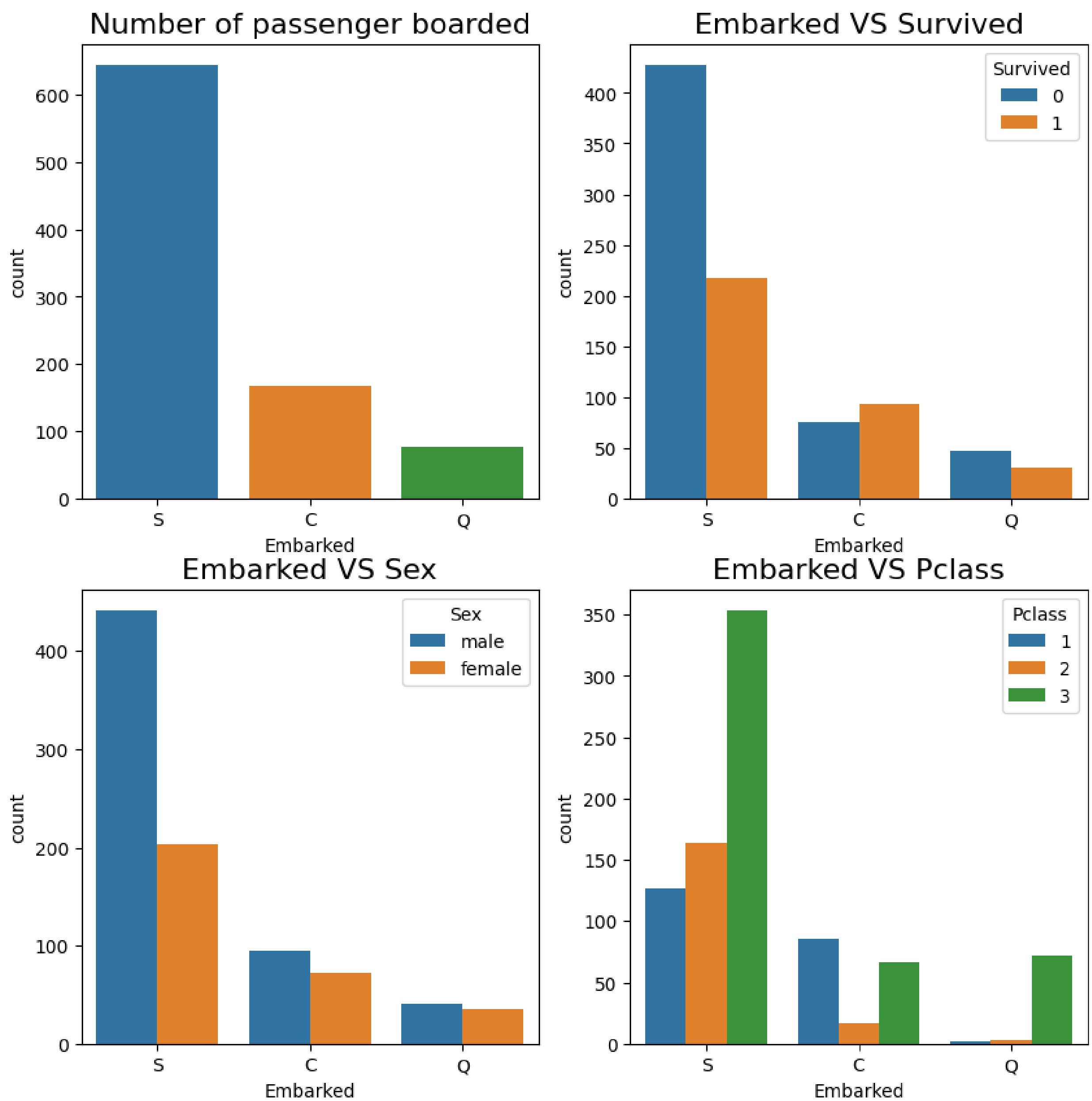
Analyzing the role 'Embarked'

```
df['Embarked'].unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

```
f, ax = plt.subplots(2, 2, figsize = (10, 10))
sns.countplot(x = 'Embarked', data = df, ax = ax[0, 0])
ax[0, 0].set_title('Number of passenger boarded', fontsize = 16)
sns.countplot(x = 'Embarked', data = df, hue = 'Survived', ax = ax[0,
1])
ax[0, 1].set_title('Embarked VS Survived', fontsize = 16)
sns.countplot(x = 'Embarked', data = df, hue = 'Sex', ax = ax[1, 0])
ax[1, 0].set_title('Embarked VS Sex', fontsize = 16)
sns.countplot(x = 'Embarked', data = df, hue = 'Pclass', ax = ax[1,
1])
ax[1, 1].set_title('Embarked VS Pclass', fontsize = 16)
```

```
Text(0.5, 1.0, 'Embarked VS Pclass')
```



1. From the above graph it is clear that maximum people boarded from port S and among those people around 75% were men and around same proportion of people could not survived. Also from plot between embarked and Pclass it is clear that most of the who boarded at S were in 3rd class.

2. People who boarded from port C have the max survival to death ration, also most of the poeple were in class 1st and 2nd.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   PassengerId        891 non-null    int64  
 1   Survived           891 non-null    int64  
 2   Pclass             891 non-null    int64  
 3   Name               891 non-null    object 
 4   Sex                891 non-null    object 
 5   Age                891 non-null    float64
 6   SibSp              891 non-null    int64  
 7   Parch              891 non-null    int64  
 8   Ticket             891 non-null    object 
 9   Fare               891 non-null    float64
 10  Cabin              891 non-null    object 
 11  Embarked           889 non-null    object 
 12  Ticket_frequency   891 non-null    int64  
 13  Fam_Size           891 non-null    int64  
 14  FsizeD             891 non-null    object 
 15  title              891 non-null    object 
dtypes: float64(2), int64(7), object(7)
memory usage: 111.5+ KB
```

Converting into Dummies

```
dummies = pd.get_dummies(df['Embarked'],drop_first=True)

df = pd.concat([df.drop('Embarked',axis=1),dummies],axis=1)

dummies = pd.get_dummies(df['Cabin'],drop_first=True)

df = pd.concat([df.drop('Cabin',axis=1),dummies],axis=1)

dummies = pd.get_dummies(df['Sex'],drop_first=True)

df = pd.concat([df.drop('Sex',axis=1),dummies],axis=1)

dummies = pd.get_dummies(df['title'],drop_first=True)

df = pd.concat([df.drop('title',axis=1),dummies],axis=1)

dummies = pd.get_dummies(df['FsizeD'],drop_first=True)

df = pd.concat([df.drop('FsizeD',axis=1),dummies],axis=1)

df.columns

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Age', 'SibSp', 
'Parch',
       'Ticket', 'Fare', 'Ticket_frequency', 'Fam_Size', 'Q', 'S', 
'B', 'C',
       'D', 'E', 'F', 'G', 'Z', 'male', 'Mrs', 'Ms', 'kid', 'officer',
```

```
       'royalty', 'Big', 'Small'],
      dtype='object')
```

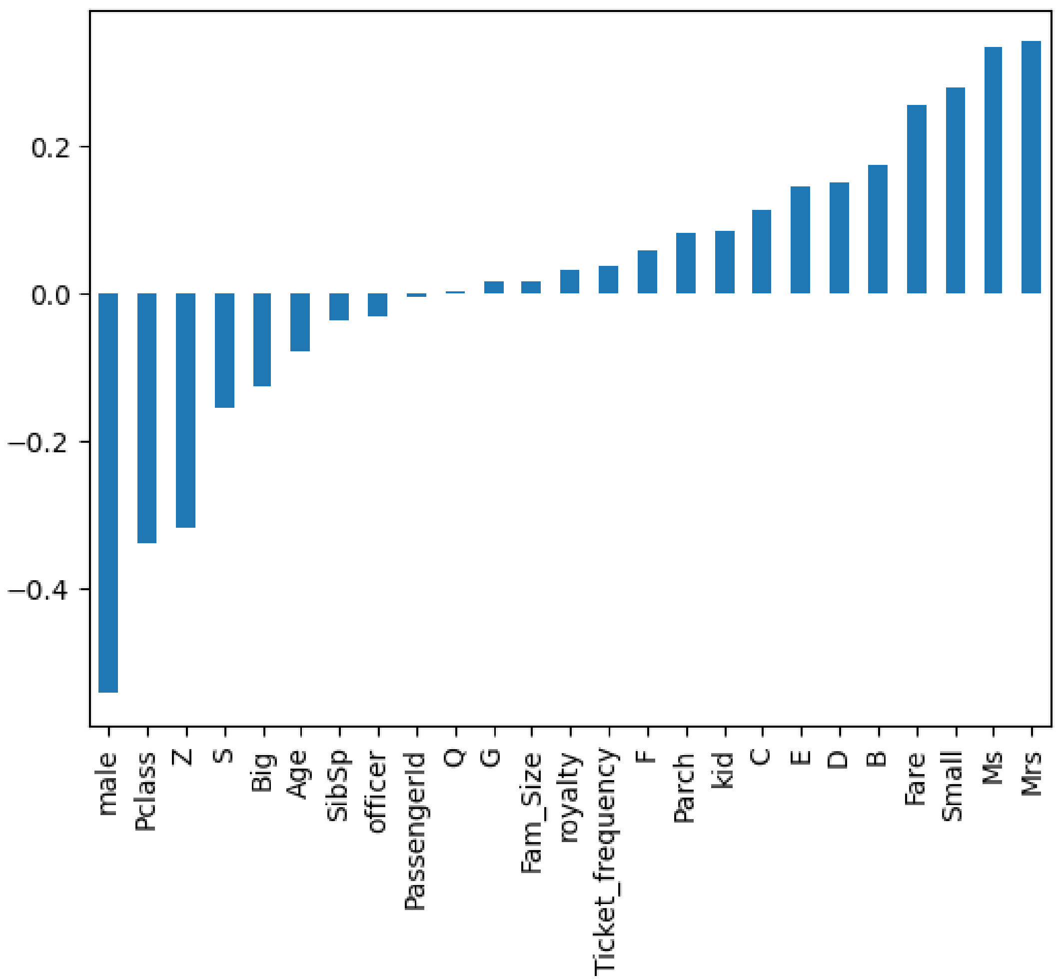
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   PassengerId       891 non-null    int64  
 1   Survived          891 non-null    int64  
 2   Pclass            891 non-null    int64  
 3   Name              891 non-null    object 
 4   Age               891 non-null    float64
 5   SibSp             891 non-null    int64  
 6   Parch             891 non-null    int64  
 7   Ticket            891 non-null    object 
 8   Fare              891 non-null    float64
 9   Ticket_frequency  891 non-null    int64  
 10  Fam_Size          891 non-null    int64  
 11  Q                 891 non-null    uint8  
 12  S                 891 non-null    uint8  
 13  B                 891 non-null    uint8  
 14  C                 891 non-null    uint8  
 15  D                 891 non-null    uint8  
 16  E                 891 non-null    uint8  
 17  F                 891 non-null    uint8  
 18  G                 891 non-null    uint8  
 19  Z                 891 non-null    uint8  
 20  male              891 non-null    uint8  
 21  Mrs               891 non-null    uint8  
 22  Ms                891 non-null    uint8  
 23  kid               891 non-null    uint8  
 24  officer           891 non-null    uint8  
 25  royalty           891 non-null    uint8  
 26  Big               891 non-null    uint8  
 27  Small             891 non-null    uint8  
dtypes: float64(2), int64(7), object(2), uint8(17)
memory usage: 91.5+ KB
```

```
df.drop(['Name', 'Ticket'], axis = 1, inplace = True)
```

```
df.corr()['Survived'].sort_values().drop('Survived').plot(kind='bar')
```

```
<Axes: >
```




```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   PassengerId       891 non-null    int64
 1   Survived          891 non-null    int64
 2   Pclass            891 non-null    int64
 3   Age               891 non-null    float64
 4   SibSp             891 non-null    int64
 5   Parch             891 non-null    int64
 6   Fare              891 non-null    float64
 7   Ticket_frequency  891 non-null    int64
 8   Fam_Size          891 non-null    int64
 9   Q                 891 non-null    uint8
 10  S                 891 non-null    uint8
 11  B                 891 non-null    uint8
```

```
 12  C                   891 non-null    uint8
 13  D                   891 non-null    uint8
 14  E                   891 non-null    uint8
 15  F                   891 non-null    uint8
 16  G                   891 non-null    uint8
 17  Z                   891 non-null    uint8
 18  male                891 non-null    uint8
 19  Mrs                 891 non-null    uint8
 20  Ms                  891 non-null    uint8
 21  kid                 891 non-null    uint8
 22  officer             891 non-null    uint8
 23  royalty             891 non-null    uint8
 24  Big                 891 non-null    uint8
 25  Small               891 non-null    uint8
dtypes: float64(2), int64(7), uint8(17)
memory usage: 77.6 KB
```

```
df.head()
```

```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  \
0            1         0       3  22.0      1      0   7.2500
1            2         1       1  38.0      1      0  71.2833
2            3         1       3  26.0      0      0   7.9250
3            4         1       1  35.0      1      0  53.1000
4            5         0       3  35.0      0      0   8.0500

   Ticket_frequency  Fam_Size  Q  ...  G  Z  male  Mrs  Ms  kid
officer  \
0                 1         2  0  ...  0  1     1    0   0    0
0
1                 1         2  0  ...  0  0     0    1   0    0
0
2                 1         1  0  ...  0  1     0    0   1    0
0
3                 2         2  0  ...  0  0     0    1   0    0
0
4                 1         1  0  ...  0  1     1    0   0    0
0

   royalty  Big  Small
0        0    0      1
1        0    0      1
2        0    0      0
3        0    0      1
4        0    0      0

[5 rows x 26 columns]
```

Building a Logistic Model (Training and Predicting)

```
X_train, X_val, y_train, y_val =
train_test_split(df.drop('Survived',axis=1), df['Survived'],
test_size=0.30, random_state=101)
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
predictions = logmodel.predict(X_val)
```

```
accuracy=confusion_matrix(y_val,predictions)
```

```
accuracy
```

```
array([[138,  16],
       [ 37,  77]])
```

Evaluation of the validation set

```
accuracy=accuracy_score(y_val,predictions)
accuracy
```

```
0.8022388059701493
```

```
print(classification_report(y_val,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       154
           1       0.83      0.68      0.74       114

    accuracy                           0.80       268
   macro avg       0.81      0.79      0.79       268
weighted avg       0.81      0.80      0.80       268
```

Using the Model in Test data to Predict

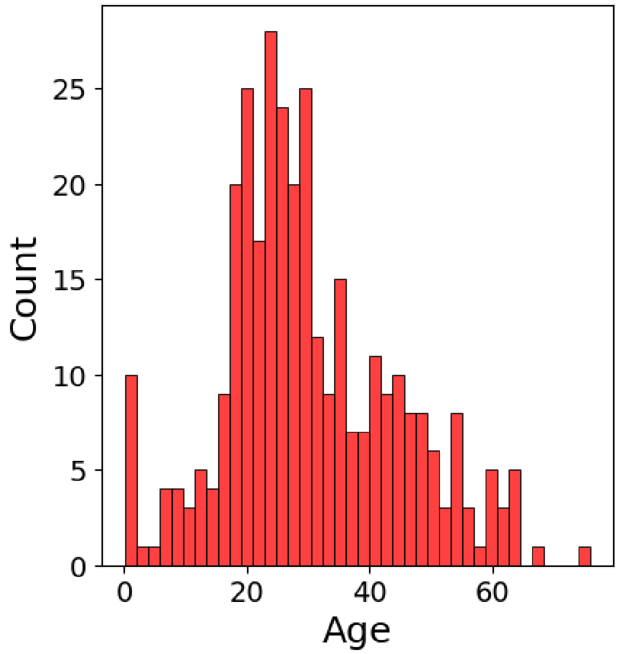
```
test.info()
```

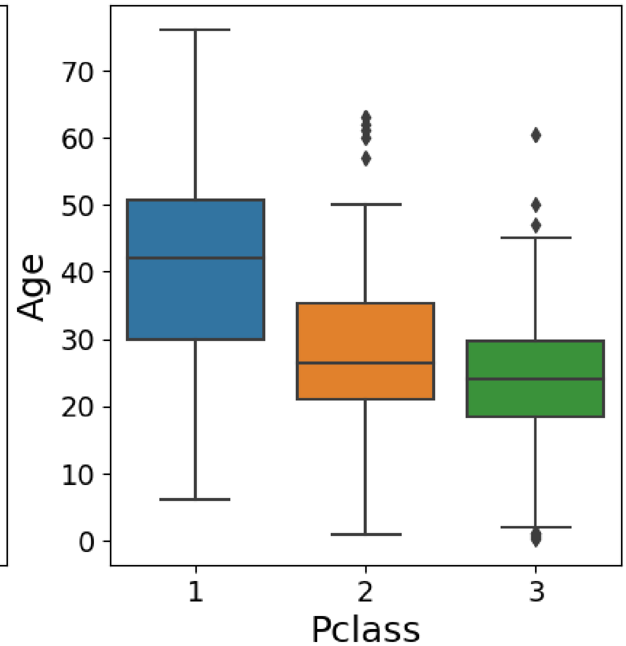
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```
dft = test
```

Handling Missing data

```
f, ax = plt.subplots(1, 2, figsize = (10, 5))

sns.histplot(dft.Age.dropna(), kde=False, color='red', bins=40, ax = 
ax[0])
# ax[0].set_title('Distribution of Age data', fontsize = 16)

sns.boxplot(x = 'Pclass', y = 'Age', data = dft, ax = ax[1])
# ax[1].set_title('Age vs Pclass', fontsize  = 16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18, rotation = 0)
    a.set_ylabel(a.get_ylabel(), fontsize=18, rotation = 90)
```

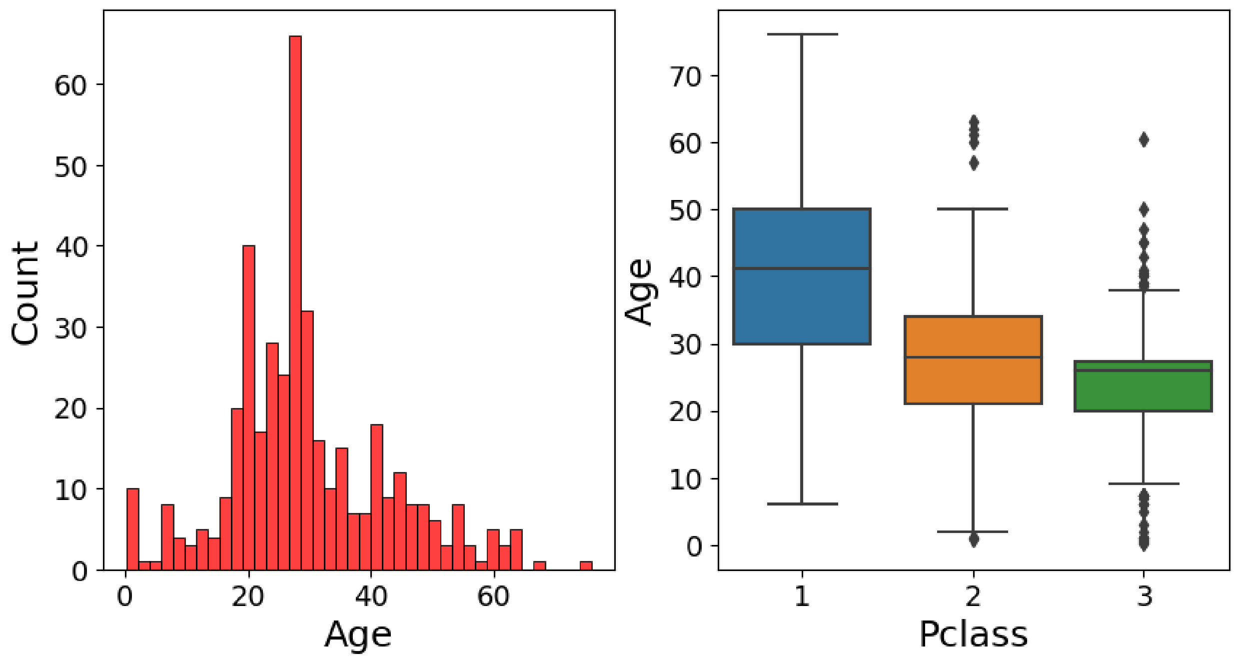





```
# Grouping Title
new_title = {
    'Mr' : 'Mr','Ms' : 'Ms','Mrs' : 'Mrs','Rev' : 'officer','Sir' :
'royalty','theCountess' : 'royalty','Dona' : 'royalty','Capt' :
'officer','Col' : 'officer','Don' : 'royalty','Dr' :
'officer','Jonkheer' : 'royalty','Lady' : 'royalty','Major' :
'officer','Master' : 'kid','Miss' : 'Ms','Mlle' : 'Ms','Mme' : 'Mrs'
}

#Add Title
def add_title(df):
    df['title'] = df['Name'].apply(lambda x: x.split(",")[1])
    df['title'] = df['title'].apply(lambda x: x.split(".")[0])
    df.title = df.title.str.replace(' ', '')

add_title(dft)

# Group Title
dft['title'] = dft['title'].apply(lambda x: new_title[x])

# Function to Update missing age values
def update_age(params):
    pclass = params[0]
    title = params[1]
    sex = params[2]
    age = params[3]
    if pd.isnull(age):
        age = np.float(age_df[(age_df['title'] == title) &
(age_df["Sex"] == sex) & (age_df['Pclass'] == pclass)]["Age"])
```

```
    return age

# Dataframe to group age across Pclass, Title and Sex
age_df =
dft.groupby(['Pclass','title','Sex']).Age.mean().reset_index()

# Fill missing age
dft['Age'] = dft[['Pclass', 'title', 'Sex', 'Age']].apply(lambda x:
update_age(x), axis = 1)

f, ax = plt.subplots(1, 2, figsize = (10, 5))

sns.histplot(dft.Age.dropna(), kde=False, color='red', bins=40, ax =
ax[0])
# ax[0].set_title('Distribution of Age data', fontsize = 16)

sns.boxplot(x = 'Pclass', y = 'Age', data = dft, ax = ax[1])
# ax[1].set_title('Age vs Pclass', fontsize  = 16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
    a.set_xlabel(a.get_xlabel(), fontsize=18, rotation = 0)
    a.set_ylabel(a.get_ylabel(), fontsize=18, rotation = 90)
```

```
<ipython-input-46-06e03c71d2e9>:8: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use
`float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use
`np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  age = np.float(age_df[(age_df['title'] == title) & (age_df["Sex"] ==
sex) & (age_df['Pclass'] == pclass)]["Age"])
```




```
dft.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          418 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
 11  title        418 non-null    object
dtypes: float64(2), int64(4), object(6)
memory usage: 39.3+ KB

dft['Cabin'].unique()

array([nan, 'B45', 'E31', 'B57 B59 B63 B66', 'B36', 'A21', 'C78',
'D34',
       'D19', 'A9', 'D15', 'C31', 'C23 C25 C27', 'F G63', 'B61',
'C53',
       'D43', 'C130', 'C132', 'C101', 'C55 C57', 'B71', 'C46', 'C116',
       'F', 'A29', 'G6', 'C6', 'C28', 'C51', 'E46', 'C54', 'C97',
```

```
'D22',
       'B10', 'F4', 'E45', 'E52', 'D30', 'B58 B60', 'E34', 'C62 C64',
       'A11', 'B11', 'C80', 'F33', 'C85', 'D37', 'C86', 'D21', 'C89',
       'F E46', 'A34', 'D', 'B26', 'C22 C26', 'B69', 'C32', 'B78',
       'F E57', 'F2', 'A18', 'C106', 'B51 B53 B55', 'D10 D12', 'E60',
       'E50', 'E39 E41', 'B52 B54 B56', 'C39', 'B24', 'D28', 'B41',
'C7',
       'D40', 'D38', 'C105'], dtype=object)
```

Creating a new cabin class = 'Z'. Replacing the nan value with it and removing the numbers from the given class just taking the alphabet.

```
dft['Cabin'] = dft['Cabin'].fillna('Z')
dft['Cabin'] = dft['Cabin'].apply(lambda s: s[0])
dft.loc[dft[dft['Cabin']=='T'].index,'Cabin']='A'

dft.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          418 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        418 non-null    object
 10  Embarked     418 non-null    object
 11  title        418 non-null    object
dtypes: float64(2), int64(4), object(6)
memory usage: 39.3+ KB

med = dft.Fare.mean()
dft.Fare.fillna(med, inplace = True)

dft.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
```

```
 2   Name          418 non-null    object
 3   Sex           418 non-null    object
 4   Age           418 non-null    float64
 5   SibSp         418 non-null    int64
 6   Parch         418 non-null    int64
 7   Ticket        418 non-null    object
 8   Fare          418 non-null    float64
 9   Cabin         418 non-null    object
 10  Embarked      418 non-null    object
 11  title         418 non-null    object
dtypes: float64(2), int64(4), object(6)
memory usage: 39.3+ KB
```

Making the required features

```
dft['Ticket_frequency'] = dft.groupby('Ticket')
['Ticket'].transform('count')

dft['Fam_Size'] = dft['Parch'] + dft['SibSp'] + 1
dft.loc[:,'FsizeD']='Alone'
dft.loc[(dft['Fam_Size']>1),'FsizeD']='Small'
dft.loc[(dft['Fam_Size']>4),'FsizeD']='Big'

dummies = pd.get_dummies(dft['Embarked'],drop_first=True)

dft = pd.concat([dft.drop('Embarked',axis=1),dummies],axis=1)

dummies = pd.get_dummies(dft['Cabin'],drop_first=True)

dft = pd.concat([dft.drop('Cabin',axis=1),dummies],axis=1)

dummies = pd.get_dummies(dft['Sex'],drop_first=True)

dft = pd.concat([dft.drop('Sex',axis=1),dummies],axis=1)

dummies = pd.get_dummies(dft['title'],drop_first=True)

dft = pd.concat([dft.drop('title',axis=1),dummies],axis=1)

dummies = pd.get_dummies(dft['FsizeD'],drop_first=True)

dft = pd.concat([dft.drop('FsizeD',axis=1),dummies],axis=1)

dft.drop(['Name', 'Ticket'], axis = 1, inplace = True)

dft.isnull().sum()

PassengerId         0
Pclass              0
Age                 0
SibSp               0
Parch               0
```

```
Fare                    0
Ticket_frequency        0
Fam_Size                0
Q                       0
S                       0
B                       0
C                       0
D                       0
E                       0
F                       0
G                       0
Z                       0
male                    0
Mrs                     0
Ms                      0
kid                     0
officer                 0
royalty                 0
Big                     0
Small                   0
dtype: int64
```

```
dft.columns
```

```
Index(['PassengerId', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'Ticket_frequency', 'Fam_Size', 'Q', 'S', 'B', 'C', 'D', 'E',
'F', 'G',
       'Z', 'male', 'Mrs', 'Ms', 'kid', 'officer', 'royalty', 'Big',
'Small'],
      dtype='object')
```

Building a Logistic Model & Evaluation

```
logmodel = LogisticRegression()
logmodel.fit(df.drop('Survived',axis=1), df['Survived'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()

predictions = logmodel.predict(dft)

predictions

array([0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
1,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
1,
       1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1,
0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
1])
```

Adding the new columns as Survived Column

```
dft['Survived'] = predictions

dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   PassengerId       418 non-null    int64  
 1   Pclass            418 non-null    int64  
 2   Age               418 non-null    float64
 3   SibSp             418 non-null    int64  
 4   Parch             418 non-null    int64  
 5   Fare              418 non-null    float64
 6   Ticket_frequency  418 non-null    int64  
 7   Fam_Size          418 non-null    int64  
 8   Q                 418 non-null    uint8  
 9   S                 418 non-null    uint8  
 10  B                 418 non-null    uint8  
 11  C                 418 non-null    uint8  
 12  D                 418 non-null    uint8  
 13  E                 418 non-null    uint8  
 14  F                 418 non-null    uint8  
 15  G                 418 non-null    uint8  
 16  Z                 418 non-null    uint8  
 17  male              418 non-null    uint8  
 18  Mrs               418 non-null    uint8  
 19  Ms                418 non-null    uint8  
 20  kid               418 non-null    uint8  
 21  officer           418 non-null    uint8  
 22  royalty           418 non-null    uint8  
 23  Big               418 non-null    uint8  
 24  Small             418 non-null    uint8  
 25  Survived          418 non-null    int64  
dtypes: float64(2), int64(7), uint8(17)
memory usage: 36.5 KB
```

Adding survived column in main data set

```
test['Survived'] = dft['Survived']

f,ax=plt.subplots(1,2,figsize=(12,6))
dft['Survived'].value_counts().plot.pie(ax = ax[0],
explode=[0,0.1],autopct='%1.1f%%',shadow=True, textprops={'fontsize':
16})
ax[0].set_title('Survived', fontsize=16)
ax[0].set_ylabel('')

sns.countplot(x = 'Survived',data = dft, ax=ax[1])
ax[1].set_title('Survived', fontsize=16)

for a in ax:
    a.tick_params(axis='both', which='major', labelsize=14)
```

```
    a.set_xlabel(a.get_xlabel(), fontsize=18)
    a.set_ylabel(a.get_ylabel(), fontsize=18)

plt.show()

f, ax = plt.subplots(3, 2, figsize = (16, 10))
plt.subplots_adjust(hspace=0.4)
sns.countplot(x = 'Survived', data = dft,hue = 'Pclass', ax = ax[0,
0])
ax[0, 0].set_title('Pclass vs Survived', fontsize = 16)
sns.countplot(x = 'Survived', data = dft, hue = 'SibSp', ax = ax[0,
1])
ax[0, 1].set_title('SibSp VS Survived', fontsize = 16)
sns.countplot(x = 'Survived', data = dft, hue = 'Parch', ax = ax[1,
0])
ax[1, 0].set_title('Parch VS Survived', fontsize = 16)
sns.countplot(x = 'Survived', data = dft, hue = 'Ticket_frequency', ax
= ax[1, 1])
ax[1, 1].set_title('Ticket frequency VS Survival', fontsize = 16)
sns.countplot(x = 'Survived', data = dft, hue = 'Fam_Size', ax = ax[2,
0])
ax[2, 0].set_title('Family Size VS Survival', fontsize = 16)
sns.countplot(x = 'Survived', data = test, hue = 'Sex', ax = ax[2,1])
ax[2, 1].set_title('Sex VS Survival', fontsize = 16)
```