# Data Analytics Lab: Assignment - 2
# A mathematical essay on logistic regression

Nagappan N
*Department of Metallurgical and Materials Engineering*
*IIT Madras*
Chennai, India
mm19b040@smail.iitm.ac.in

***Abstract*—This work focuses on the tragic incident of sinking of the Titanic. We build a logistic regression model that predicts whether or not a person survived. We built a model that predicts this with an accuracy of about 83%.**

## I. Introduction

The sinking of Titanic is one of the most infamous shipwrecks in history. The unfortunate incident has been studied by various researchers around the world and today we know the reasons for the failure of the Titanic. Many people had lost their lives due to insufficient life boats. It is seen that some groups of people had more chances of survival than some others.

Logistic regression is a statistical model that is used mainly for classification tasks. It is a supervised machine learning technique. It involves the logistic function about which we will see in the upcoming section.

Here, we try to use logistic regression to predict whether or not a person survived using inputs from the titanic dataset. This includes data related to age, gender, travel class and family.

In the next section, we give a brief overview of logistic regression and show how it works. Then we explore our dataset through various visualizations and perform data cleaning. Following these, we implement logistic regression on our dataset to make the predictions. Finally, we present the conclusions that we derive from our analysis and predictions.

## II. Logistic Regression

Logistic regression estimates the probability of an event using a set of independent variables. [1] As the prediction is a probability, the dependent variables varies in the range of 0 to 1. For binary classification, a probability less than .5 will predict 0 while a probability greater than 0.5 will predict 1. The independent variables involved maybe categorical or continuous.

There are 3 major types of logistic regression:

- Binary logistic regression: In this case, the dependent variable has only 2 possible outcomes.
- Multinomial logistic regression: In this case, the dependent variable has only 3 or more possible outcomes.
- Ordinal logistic regression: Here as well the dependent variable has only 3 or more possible outcomes but also these output values have a defined order.

Lets see how logistic regression works. First, we look at the logistic function:

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

Fig. 1. Count plot showing the survival statistics based on gender.

This is also called as the sigmoid function. In logistic regression, our aim is to maximize the likelihood function. [2] The likelihood function is nothing but a joint pdf of our sample observations and joint distribution is the multiplication of the conditional probability for observing each example given the distribution parameters.

## III. Datasets

We have been given 2 datasets (train and test data). The train data has all columns in the test data plus the additional column Survived which tells us whether or not the person survived. This is our target column. We first use the train dataset to build a logistic regression model and then apply it on the test dataset to make our predictions.

The train dataset has the following columns:

1) PassengerId
2) Survived: Tells whether or not the person survived. 1 means survived and 0 means did not survive.
3) Pclass: The class in which the passenger travelled in the titanic. There are 3 classes: 1st, 2nd and 3rd.
4) Name: Name of the passenger.
5) Sex: Gender of the passenger.
6) Age: Age of the passenger.
7) SibSp: Number of of siblings / spouses aboard the Titanic.
8) Parch: Number of parents / children aboard the Titanic.
9) Ticket: Ticket number of the passenger.

10) Fare: Price of the ticket for the passenger.
11) Cabin: Cabin number of the passenger.
12) Embarked: Tells us the port of embarkation. C = Cherbourg, Q = Queenstown and S = Southampton.

### A. *Data Visualization*

Firstly, we analyze how the various features in our dataset are distributed and related.

From figure 3, we see that females were more likely to survive that the males. Even though the number of females was less in comparison to the number of males, the number of females who survived is much larger than the male counterpart.

From figure 4, we see that the proportion of passengers who survive class 1 is higher than that of class 2 which in turn is much higher than that of class 3.

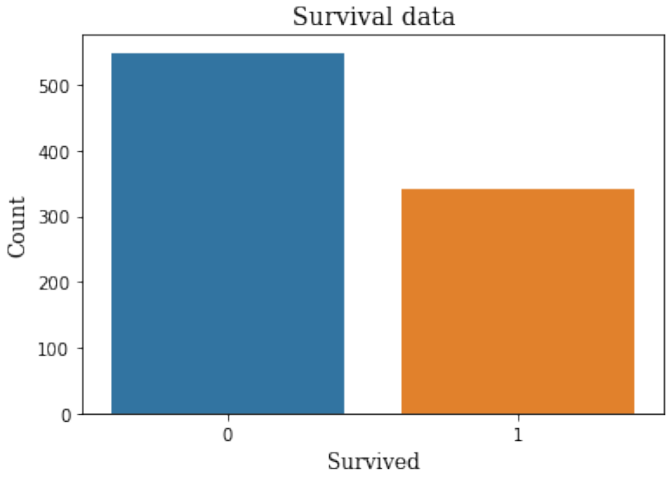



Fig. 2. Countplot showing the number of people who survived and who did not.

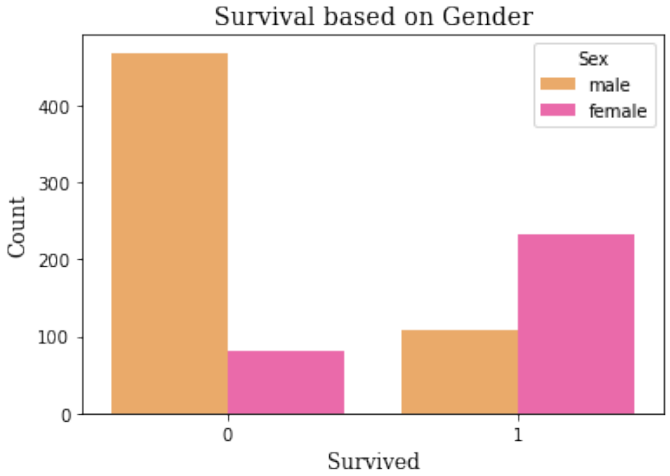



Fig. 3. Count plot showing the survival statistics based on gender.

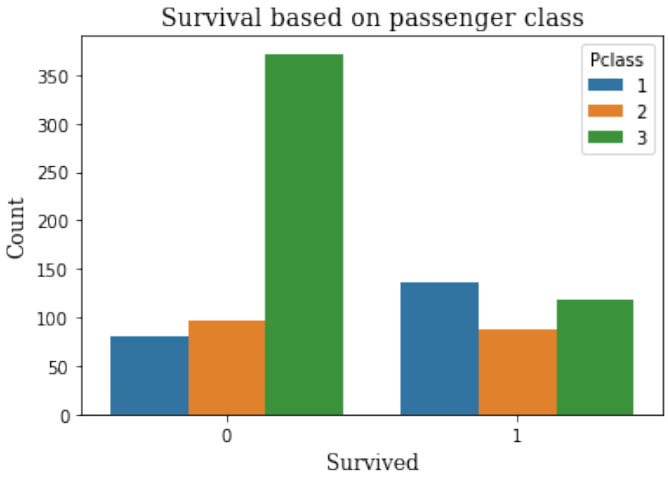



Fig. 4. Count plot showing the survival statistics based on the travel class of passenger.

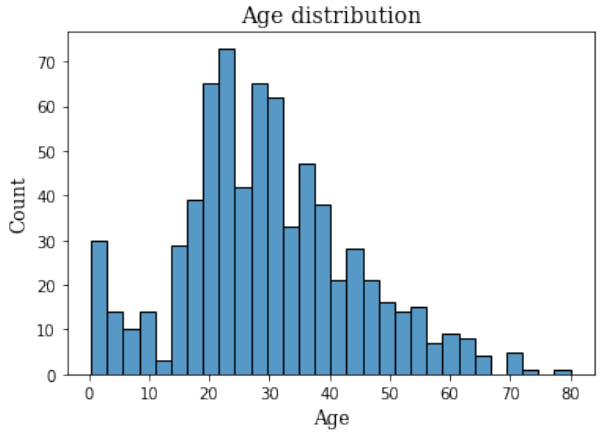



Fig. 5. Histogram of the age distribution of people who travelled.

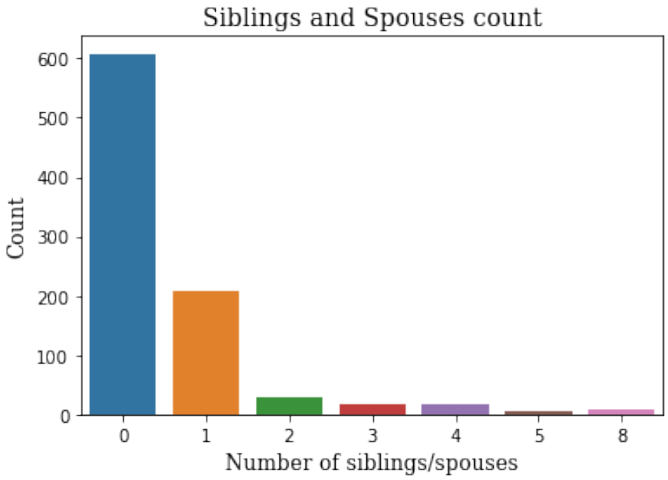



Fig. 6. Count plot showing the number of siblings and spouses who were travelling with the passenger.

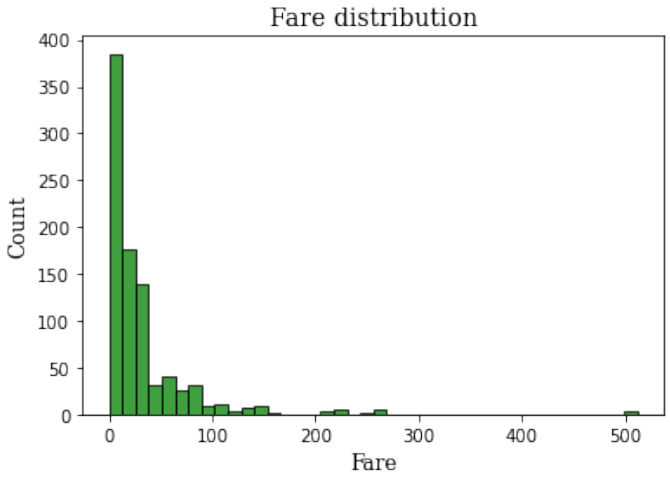



Fig. 7. Histogram of the fare distribution of passengers.

### B. Data Preprocessing

We process the train dataset initially and follow the same process for the test data as well. In the age column, around 20% of the data are missing. The average age of the passengers in the passenger class, to which the datapoint with a missing age belongs, might be used to fill in these missing data. The Cabin column is almost fully filled with null values, which makes us to drop the whole column. There is one missing fare value in the test data which we will with the average fare value of that particular class.

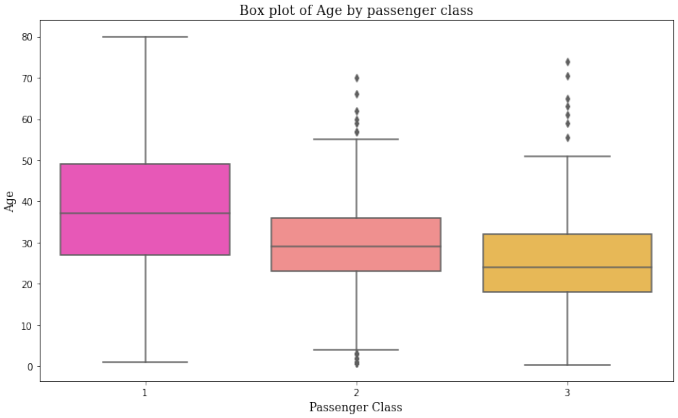



Fig. 8. Box plot showing the age distribution of passengers in each class.

Some columns don't actually provide any insightful information. However, they would be beneficial if the necessary data was extracted from them or if they were combined with other columns.

We make an effort to identify passengers' titles, such as "Master," "Mr," "Mrs," and "Miss," and the rest is collectively referred to as "Rare." We could infer from the survival rate that women and children fared better than men in terms of survival. As a result, the name columns are converted into labels and assigned to one of the five title categories.

The age is divided into 5 age categories with labels for each, including 0–16, 16–32, 32–48, 48–64, and 64–80. Later, the ageband column is taken out. When the survival rate is compared, it is once more evident that youngsters have a higher rate of survival than adults.

The number of siblings, parents, and children is added to the SibSp and Parch columns, together with 1 (the passenger himself/herself), to create a new column called Family size. This family size field is eventually changed to a new column called "IsAlone," which indicates whether the traveller was travelling alone or with family. When the survival rate is examined, it can be seen that passengers travelling alone fared better than those travelling with family members, which makes sense.

The fare column is also split into 4 fare bands, namely, 0-8, 8-14, 14-31 and 31-512 and each are given a label. The remaining insignificant columns like passenger id and ticket are removed. The sex and embarked columns are given labels by using the get dummies function from the pandas. The original sex and embarked columns are then dropped.

## IV. Modelling

The given train dataset is further divided into 30% for testing and 70% for training the model. Finally, Pclass, age, male (Sex), Q, S (Embarked), fare, title, and IsAlone are taken into account as input features. The 'Survived' column of the output, which contains a binary value of 0 or 1, must be predicted. Using the Sklearn library, a logistic regression model is constructed. The model is then made to predict the testing data after being fitted with the training data. Using the aid of the metrics library's classification report function from Sklearn, we discover that our model has an 83% accuracy rate in predicting the "Survived" attribute.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.83 | 0.91 | 0.87 | 163 |
| 1 | 0.83 | 0.70 | 0.76 | 104 |
| accuracy | | | 0.83 | 267 |
| macro avg | 0.83 | 0.80 | 0.81 | 267 |
| weighted avg | 0.83 | 0.83 | 0.82 | 267 |

Fig. 9. Classification report for our logistic regression model

Using this model, we make predictions on the test dataset.

## V. Conclusions

We analyzed the titanic dataset and visualized to understand the correlation between the various features and chances of survival. Then we built a logistic regression model that predicts whether or not the passenger survived.

## REFERENCES


[1] J. Tolles and W. J. Meurer, "Logistic Regression: Relating Patient Characteristics to Outcomes," JAMA, vol. 316, no. 5, pp. 533–534, Aug. 2016, doi: 10.1001/JAMA.2016.7653.

[2] C. Gourieroux and A. Monfort, "Asymptotic properties of the maximum likelihood estimator in dichotomous logit models," J Econom, vol. 17, no. 1, pp. 83–97, Sep. 1981, doi: 10.1016/0304-4076(81)90060-9.

# EE4708: Data Analytics Lab

## Assignment 2

**Name: Nagappan N**

**Roll number: MM19B040**

First, we import necessary libraries.

In [556]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

# The Data

Let's start by reading in the titanic_train.csv file into a pandas dataframe.

In [557]:

```
train = pd.read_excel('train.xlsx')
```

In [558]:

```
train.head()            # display the first 5 rows of the train dataset
```

Out[558]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |

In [559]:

```
test = pd.read_excel('test.xlsx')
```

In [560]:

```
test.head()                    # display the first 5 rows of the test dataset
```

Out[560]:

| | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Emba |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 892 | 3 | Kelly, Mr. James | male | 34.5 | 0 | 0 | 330911 | 7.8292 | NaN | |
| **1** | 893 | 3 | Wilkes, Mrs. James (Ellen Needs) | female | 47.0 | 1 | 0 | 363272 | 7.0000 | NaN | |
| **2** | 894 | 2 | Myles, Mr. Thomas Francis | male | 62.0 | 0 | 0 | 240276 | 9.6875 | NaN | |
| **3** | 895 | 3 | Wirz, Mr. Albert | male | 27.0 | 0 | 0 | 315154 | 8.6625 | NaN | |
| **4** | 896 | 3 | Hirvonen, Mrs. Alexander (Helga E Lindqvist) | female | 22.0 | 1 | 1 | 3101298 | 12.2875 | NaN | |

In [561]:

```
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
```

# Exploratory Data Analysis

Let's begin some exploratory data analysis! We'll start by checking out missing data!

In [562]:

```
train.info()
```

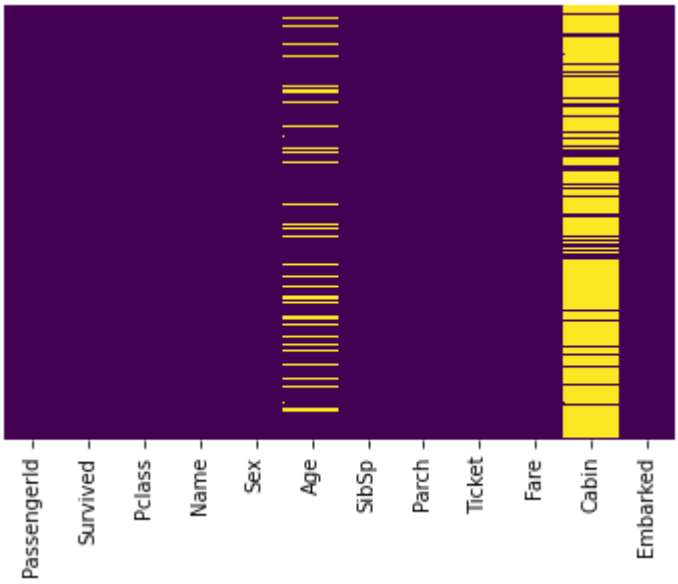
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null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

In [563]:

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[563]:

```
<AxesSubplot:>
```

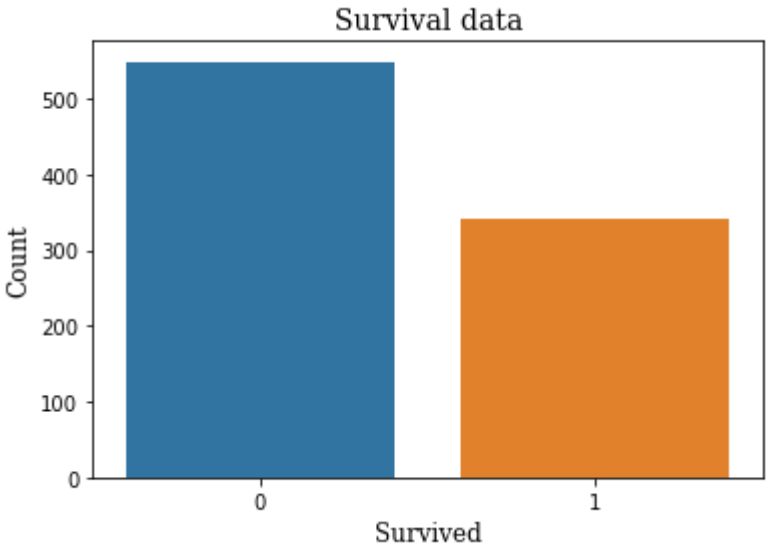



Two columns have missing values. One of them is Age. It has 177 missing values which we can try to fill using some method. The other column is Cabin which has 687 missing values and hence it is not possible to fill in the values.

# Data Visualization:

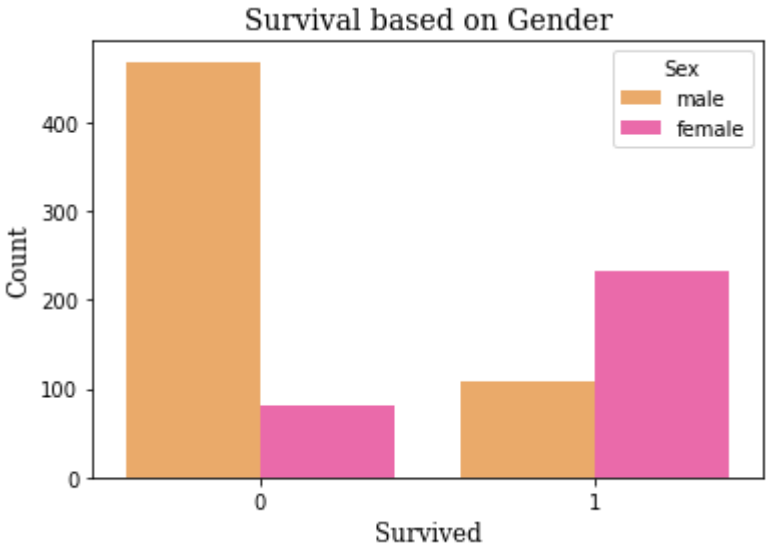
In [564]:

```
#sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train).set(title="Survival data")
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Survival data',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Survived',fontdict=font2)
plt.savefig('Survival.png',bbox_inches='tight')
```



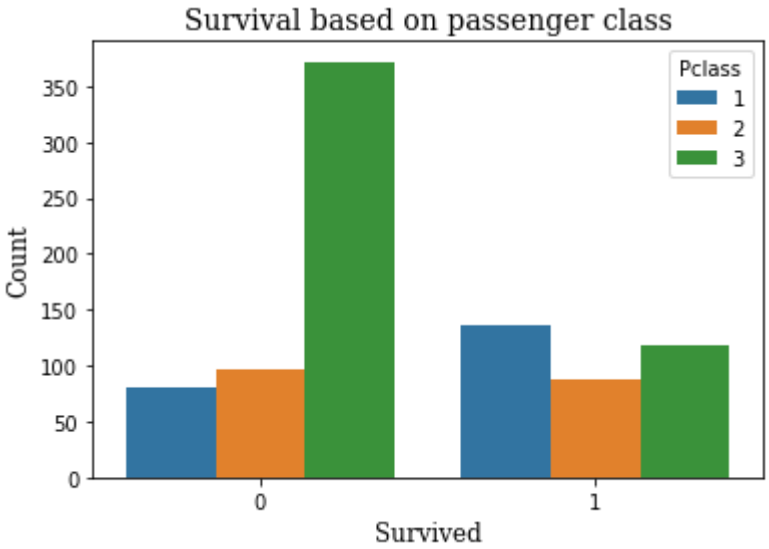

In [565]:

```
sns.countplot(x='Survived',hue='Sex',data=train,palette='spring_r').set(title='Survival bas
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Survival based on Gender',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Survived',fontdict=font2)
plt.savefig('Survival by gender.png',bbox_inches='tight')
```



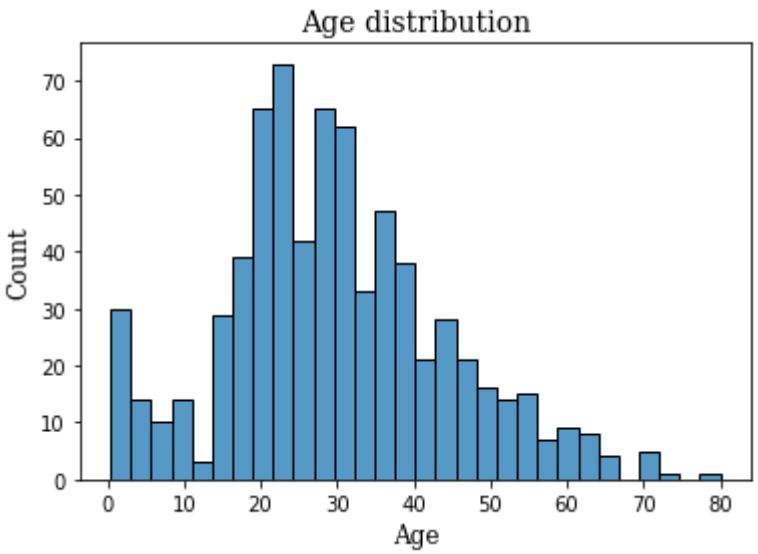
In [566]:

```
sns.countplot(x='Survived',hue='Pclass',data=train)
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Survival based on passenger class',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Survived',fontdict=font2)
plt.savefig('Survival by class.png',bbox_inches='tight')
```



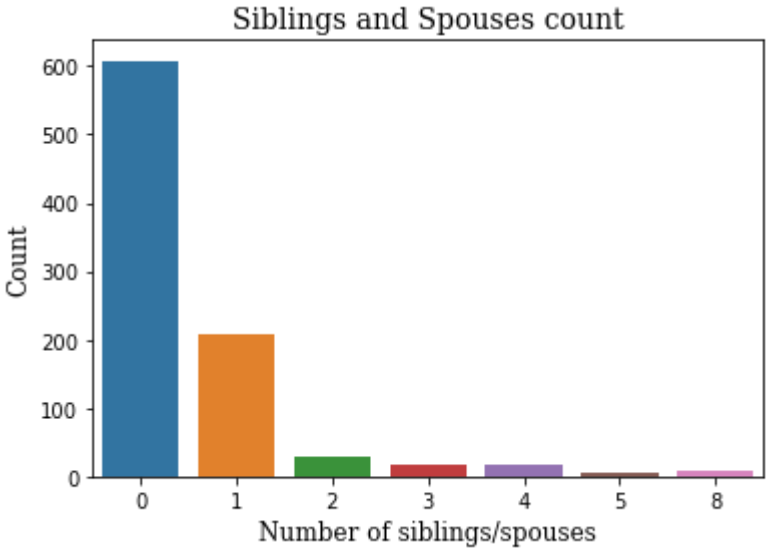

In [567]:

```
#train['Age'].hist(bins=30,color='darkred',alpha=0.7)
sns.histplot(train['Age'],bins=30)
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Age distribution',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Age',fontdict=font2)
plt.savefig('Age distribution.png',bbox_inches='tight')
```



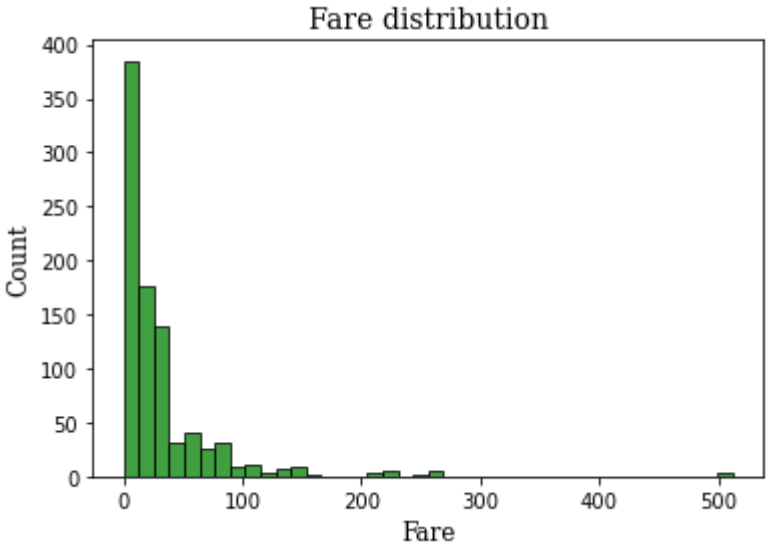
In [568]:

```
sns.countplot(x='SibSp',data=train)
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Siblings and Spouses count',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Number of siblings/spouses',fontdict=font2)
plt.savefig('sibsp.png',bbox_inches='tight')
```




In [569]:

```
#train['Fare'].hist(color='green',bins=40,figsize=(8,4))
sns.histplot(train['Fare'],bins=40,color='green')
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Fare distribution',fontdict=font1)
plt.ylabel('Count',fontdict=font2)
plt.xlabel('Fare',fontdict=font2)
plt.savefig('fare.png',bbox_inches='tight')
```

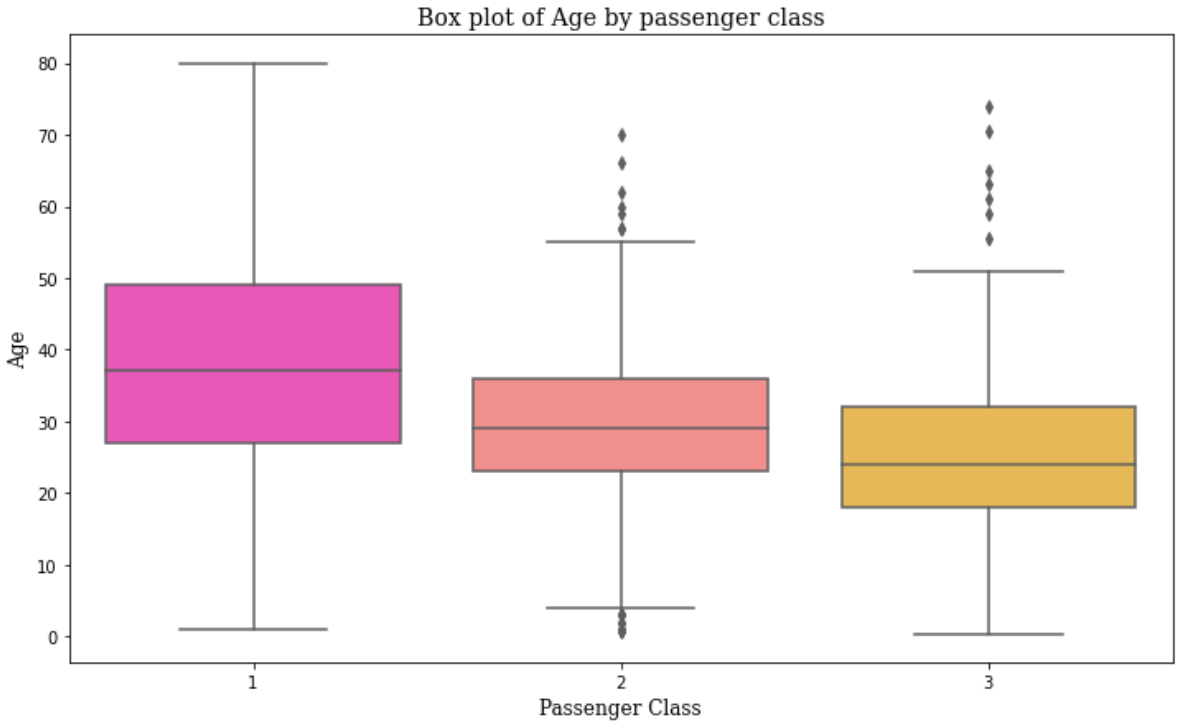



# Data Cleaning

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). To fill in a more appropriate fashion, we can check the average age by passenger class.

In [570]:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='spring')
font2 = {'family':'serif','color':'black','size':12}
font1 = {'family':'serif','color':'black','size':14}
plt.title('Box plot of Age by passenger class',fontdict=font1)
plt.ylabel('Age',fontdict=font2)
plt.xlabel('Passenger Class',fontdict=font2)
plt.savefig('age boxplot.png',bbox_inches='tight')
```




We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

In [571]:

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

In [572]:

```
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
```

In [573]:

```
test[test['Fare'].isnull()]
```

Out[573]:

| | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **152** | 1044 | 3 | Storey, Mr. Thomas | male | 60.5 | 0 | 0 | 3701 | NaN | NaN | S |

Lets fill the missing Fare of this passenger with the average fare of class 3 passengers.

In [574]:

```
test['Fare']=test['Fare'].fillna(test['Fare'].groupby(test['Pclass']).mean()[3])
```

In [575]:

```
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         418 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
```

In [576]:

```
combine = [train, test]
```

In [577]:

```
for dataset in combine:
    dataset['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
    dataset.drop('Cabin',axis=1,inplace=True)
    dataset.info()
    #dataset.dropna(inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 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   Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(4)
memory usage: 32.8+ KB
```

In [578]:

```
test['Age'].value_counts()
```

Out[578]:

```
24.0    80
37.0    19
29.0    17
22.0    17
19.0    15
        ..
55.0     1
11.0     1
28.5     1
49.0     1
60.0     1
Name: Age, Length: 72, dtype: int64
```

In [579]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 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   Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(4)
memory usage: 32.8+ KB
```

In [582]:

```
train.dropna(inplace=True)
train.info()          # We see that now there are no missing values.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Name         889 non-null    object 
 4   Sex          889 non-null    object 
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64  
 7   Parch        889 non-null    int64  
 8   Ticket       889 non-null    object 
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [583]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 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   Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(4)
memory usage: 32.8+ KB
```

# Feature Engineering

**Name column:**

In [584]:

```
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train['Title'], train['Sex'])
```

Out[584]:

| Sex | female | male |
|---|---|---|
| Title | | |
| Capt | 0 | 1 |
| Col | 0 | 2 |
| Countess | 1 | 0 |
| Don | 0 | 1 |
| Dr | 1 | 6 |
| Jonkheer | 0 | 1 |
| Lady | 1 | 0 |
| Major | 0 | 2 |
| Master | 0 | 40 |
| Miss | 181 | 0 |
| Mlle | 2 | 0 |
| Mme | 1 | 0 |
| Mr | 0 | 517 |
| Mrs | 124 | 0 |
| Ms | 1 | 0 |
| Rev | 0 | 6 |
| Sir | 0 | 1 |

In [585]:

```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'D

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

Out[585]:

| | Title | Survived |
|---|---|---|
| **0** | Master | 0.575000 |
| **1** | Miss | 0.701087 |
| **2** | Mr | 0.156673 |
| **3** | Mrs | 0.792000 |
| **4** | Rare | 0.347826 |

In [586]:

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train.head()
```

Out[586]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | I |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |

In [587]:

```
train = train.drop(['Name', 'PassengerId'], axis=1)
test = test.drop(['Name'], axis=1)
combine = [train, test]
```

**Age column:**

In [588]:

```
train['AgeBand'] = pd.cut(train['Age'], 5)
train[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_values(by='
```

Out[588]:

| | AgeBand | Survived |
|---|---|---|
| 0 | (0.34, 16.336] | 0.550000 |
| 1 | (16.336, 32.252] | 0.336714 |
| 2 | (32.252, 48.168] | 0.410138 |
| 3 | (48.168, 64.084] | 0.426471 |
| 4 | (64.084, 80.0] | 0.090909 |

In [589]:

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age']
train.head()
```

Out[589]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Ticket | Fare | Embarked | Title | AgeBanc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 1.0 | 1 | 0 | A/5 21171 | 7.2500 | S | 1 | (16.336 32.252 |
| 1 | 1 | 1 | female | 2.0 | 1 | 0 | PC 17599 | 71.2833 | C | 3 | (32.252 48.168 |
| 2 | 1 | 3 | female | 1.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | S | 2 | (16.336 32.252 |
| 3 | 1 | 1 | female | 2.0 | 1 | 0 | 113803 | 53.1000 | S | 3 | (32.252 48.168 |
| 4 | 0 | 3 | male | 2.0 | 0 | 0 | 373450 | 8.0500 | S | 1 | (32.252 48.168 |

In [590]:

```
train = train.drop(['AgeBand'], axis=1)
combine = [train, test]
train.head()
```

Out[590]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Ticket | Fare | Embarked | Title |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 1.0 | 1 | 0 | A/5 21171 | 7.2500 | S | 1 |
| **1** | 1 | 1 | female | 2.0 | 1 | 0 | PC 17599 | 71.2833 | C | 3 |
| **2** | 1 | 3 | female | 1.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | S | 2 |
| **3** | 1 | 1 | female | 2.0 | 1 | 0 | 113803 | 53.1000 | S | 3 |
| **4** | 0 | 3 | male | 2.0 | 0 | 0 | 373450 | 8.0500 | S | 1 |

### SibSp and Parch columns:

In [591]:

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort_value
```

Out[591]:

| | FamilySize | Survived |
|---|---|---|
| **3** | 4 | 0.724138 |
| **2** | 3 | 0.578431 |
| **1** | 2 | 0.552795 |
| **6** | 7 | 0.333333 |
| **0** | 1 | 0.300935 |
| **4** | 5 | 0.200000 |
| **5** | 6 | 0.136364 |
| **7** | 8 | 0.000000 |
| **8** | 11 | 0.000000 |

In [592]:

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

Out[592]:

| | IsAlone | Survived |
|---|---|---|
| **0** | 0 | 0.505650 |
| **1** | 1 | 0.300935 |

In [593]:

```
train = train.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
test = test.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)
combine = [train, test]

train.head()
```

Out[593]:

| | Survived | Pclass | Sex | Age | Ticket | Fare | Embarked | Title | IsAlone |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 1.0 | A/5 21171 | 7.2500 | S | 1 | 0 |
| **1** | 1 | 1 | female | 2.0 | PC 17599 | 71.2833 | C | 3 | 0 |
| **2** | 1 | 3 | female | 1.0 | STON/O2. 3101282 | 7.9250 | S | 2 | 1 |
| **3** | 1 | 1 | female | 2.0 | 113803 | 53.1000 | S | 3 | 0 |
| **4** | 0 | 3 | male | 2.0 | 373450 | 8.0500 | S | 1 | 1 |

## Converting categorical features:

In [594]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  889 non-null    int64  
 1   Pclass    889 non-null    int64  
 2   Sex       889 non-null    object 
 3   Age       889 non-null    float64
 4   Ticket    889 non-null    object 
 5   Fare      889 non-null    float64
 6   Embarked  889 non-null    object 
 7   Title     889 non-null    int64  
 8   IsAlone   889 non-null    int64  
dtypes: float64(2), int64(4), object(3)
memory usage: 69.5+ KB
```

In [595]:

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

In [596]:

```
train.drop(['Sex','Embarked','Ticket'],axis=1,inplace=True)
```

In [597]:

```
train = pd.concat([train,sex,embark],axis=1)
```

In [598]:

```
train.head()
```

Out[598]:

| | Survived | Pclass | Age | Fare | Title | IsAlone | male | Q | S |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | 1.0 | 7.2500 | 1 | 0 | 1 | 0 | 1 |
| **1** | 1 | 1 | 2.0 | 71.2833 | 3 | 0 | 0 | 0 | 0 |
| **2** | 1 | 3 | 1.0 | 7.9250 | 2 | 1 | 0 | 0 | 1 |
| **3** | 1 | 1 | 2.0 | 53.1000 | 3 | 0 | 0 | 0 | 1 |
| **4** | 0 | 3 | 2.0 | 8.0500 | 1 | 1 | 1 | 0 | 1 |

# Train Test Split

First, we divide our given training data into train and test.

In [599]:

```
from sklearn.model_selection import train_test_split
```

In [600]:

```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

# Training and Predicting

In [601]:

```
from sklearn.linear_model import LogisticRegression
```

In [602]:

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

```
C:\Users\91944\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:814: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[602]:

```
LogisticRegression()
```

In [603]:

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

# Evaluating our model

We can check precision,recall,f1-score using classification report!

In [604]:

```
from sklearn.metrics import classification_report
```

In [605]:

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87       163
           1       0.83      0.70      0.76       104

    accuracy                           0.83       267
   macro avg       0.83      0.80      0.81       267
weighted avg       0.83      0.83      0.82       267
```

# Predictions on test dataset

In [606]:

```
test
```

Out[606]:

| | PassengerId | Pclass | Sex | Age | Ticket | Fare | Embarked | Title | IsAlone |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 892 | 3 | male | 1.0 | 330911 | 7.8292 | Q | 1 | 1 |
| **1** | 893 | 3 | female | 2.0 | 363272 | 7.0000 | S | 3 | 0 |
| **2** | 894 | 2 | male | 1.0 | 240276 | 9.6875 | Q | 1 | 1 |
| **3** | 895 | 3 | male | 2.0 | 315154 | 8.6625 | S | 1 | 1 |
| **4** | 896 | 3 | female | 2.0 | 3101298 | 12.2875 | S | 3 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **413** | 1305 | 3 | male | 1.0 | A.5. 3236 | 8.0500 | S | 1 | 1 |
| **414** | 1306 | 1 | female | 2.0 | PC 17758 | 108.9000 | C | 5 | 1 |
| **415** | 1307 | 3 | male | 1.0 | SOTON/O.Q. 3101262 | 7.2500 | S | 1 | 1 |
| **416** | 1308 | 3 | male | 2.0 | 359309 | 8.0500 | S | 1 | 1 |
| **417** | 1309 | 3 | male | 1.0 | 2668 | 22.3583 | C | 4 | 0 |

418 rows × 9 columns

In [607]:

```
t= test.copy(deep=False)
```

In [608]:

```
sext = pd.get_dummies(test['Sex'],drop_first=True)
embarkt = pd.get_dummies(test['Embarked'],drop_first=True)
```

In [609]:

```
test.drop(['Sex','Embarked','PassengerId','Ticket'],inplace=True,axis=1)
```

In [610]:

```
test = pd.concat([test,sext,embarkt],axis=1)
```

In [611]:

```
test.head()
```

Out[611]:

| | Pclass | Age | Fare | Title | IsAlone | male | Q | S |
|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 1.0 | 7.8292 | 1 | 1 | 1 | 1 | 0 |
| **1** | 3 | 2.0 | 7.0000 | 3 | 0 | 0 | 0 | 1 |
| **2** | 2 | 1.0 | 9.6875 | 1 | 1 | 1 | 1 | 0 |
| **3** | 3 | 2.0 | 8.6625 | 1 | 1 | 1 | 0 | 1 |
| **4** | 3 | 2.0 | 12.2875 | 3 | 0 | 0 | 0 | 1 |

In [612]:

```
pp=logmodel.predict(test)
```

In [613]:

```
pp = pd.DataFrame(pp)
pp
```

Out[613]:

| | 0 |
|---|---|
| **0** | 0 |
| **1** | 0 |
| **2** | 0 |
| **3** | 0 |
| **4** | 0 |
| **...** | ... |
| **413** | 0 |
| **414** | 1 |
| **415** | 0 |
| **416** | 0 |
| **417** | 0 |

418 rows × 1 columns

In [614]:

```
t=t.reset_index()
t=t.drop('index',axis=1)
```

In [615]:

```
t = pd.concat([t,pp],axis=1)
t
```

Out[615]:

| | PassengerId | Pclass | Sex | Age | Ticket | Fare | Embarked | Title | IsAlone | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 892 | 3 | male | 1.0 | 330911 | 7.8292 | Q | 1 | 1 | 0 |
| **1** | 893 | 3 | female | 2.0 | 363272 | 7.0000 | S | 3 | 0 | 0 |
| **2** | 894 | 2 | male | 1.0 | 240276 | 9.6875 | Q | 1 | 1 | 0 |
| **3** | 895 | 3 | male | 2.0 | 315154 | 8.6625 | S | 1 | 1 | 0 |
| **4** | 896 | 3 | female | 2.0 | 3101298 | 12.2875 | S | 3 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **413** | 1305 | 3 | male | 1.0 | A.5. 3236 | 8.0500 | S | 1 | 1 | 0 |
| **414** | 1306 | 1 | female | 2.0 | PC 17758 | 108.9000 | C | 5 | 1 | 1 |
| **415** | 1307 | 3 | male | 1.0 | SOTON/O.Q. 3101262 | 7.2500 | S | 1 | 1 | 0 |
| **416** | 1308 | 3 | male | 2.0 | 359309 | 8.0500 | S | 1 | 1 | 0 |
| **417** | 1309 | 3 | male | 1.0 | 2668 | 22.3583 | C | 4 | 0 | 0 |

418 rows × 10 columns

In [616]:

```
pt = t[['PassengerId',0]]
```

In [617]:

```
pt.to_csv('pt.csv')
```