Logistic Regression Theory Notes

EXPLORATION AND VISUALIZATION OF OUR DATA - Step 1

In [1]:
```
import pandas as pd
import numpy as py
```

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

In [4]:
```
%matplotlib inline
```

In [5]:
```
#First, we want to bring our train file to a panda dataframe

train = pd.read_csv('titanic_train.csv')
```

In [7]:
```
train.head()
```

Out[7]:

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

In [8]:
```
#So just to check where we are missing data, if it's the case

train.isnull()
```

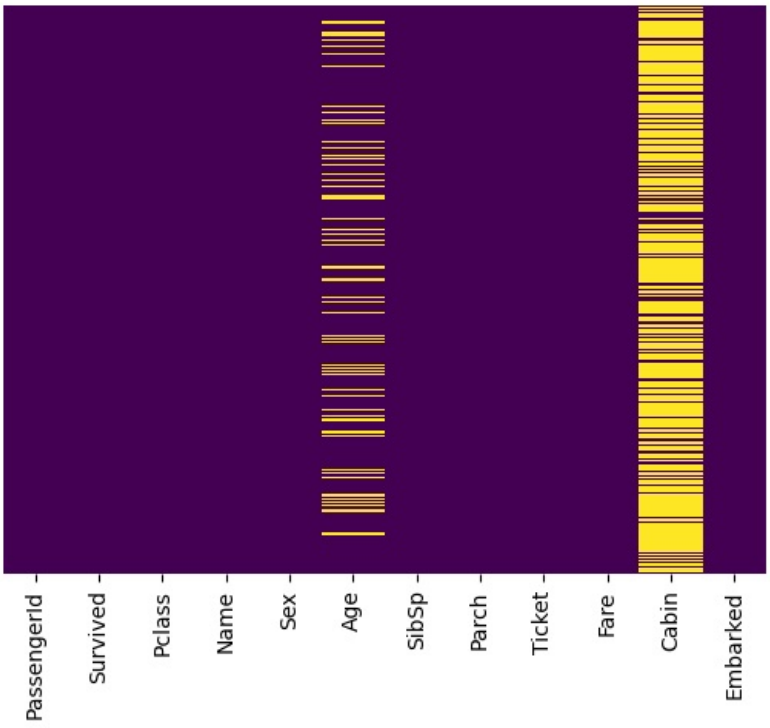
Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

In [9]:
```
# We are going to use Seaborn heatmap to check it out more in depth

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

Out[9]:
```
<AxesSubplot:>
```

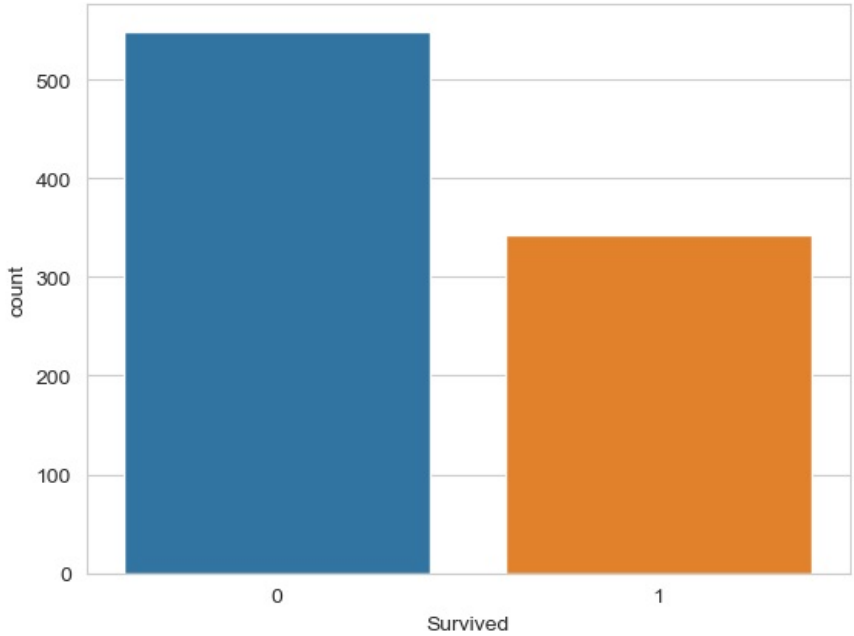


In [10]:
```
#We will comeback to the missing data later on. But we wil find a solution later
```

In [11]:
```
sns.set_style('whitegrid')
```

In [12]:
```
#We want to have a ratio of surviors and non survivors
# So countplot

sns.countplot(x='Survived',data = train)
```

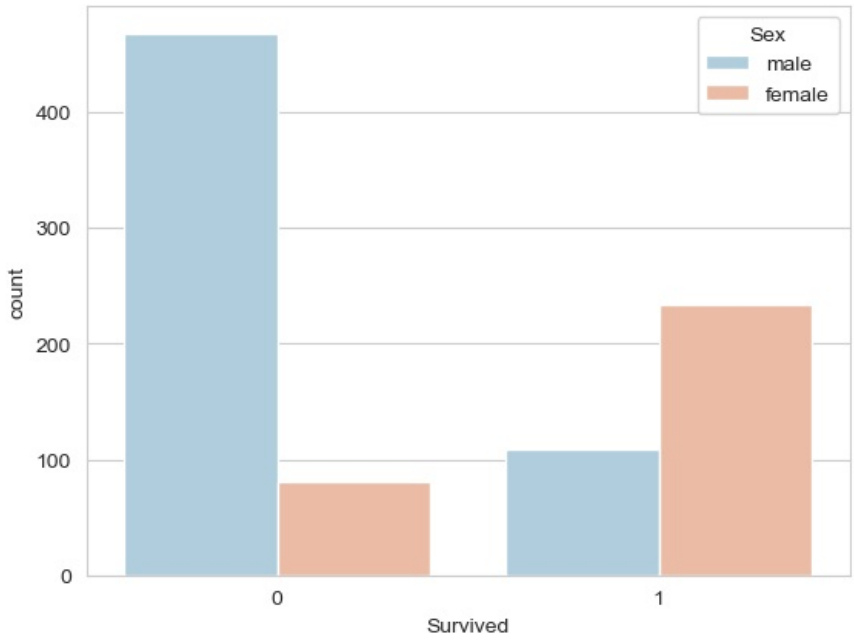
Out[12]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [14]:
```
#So less survivors, let's compare it by sex. Color is just a preference
```

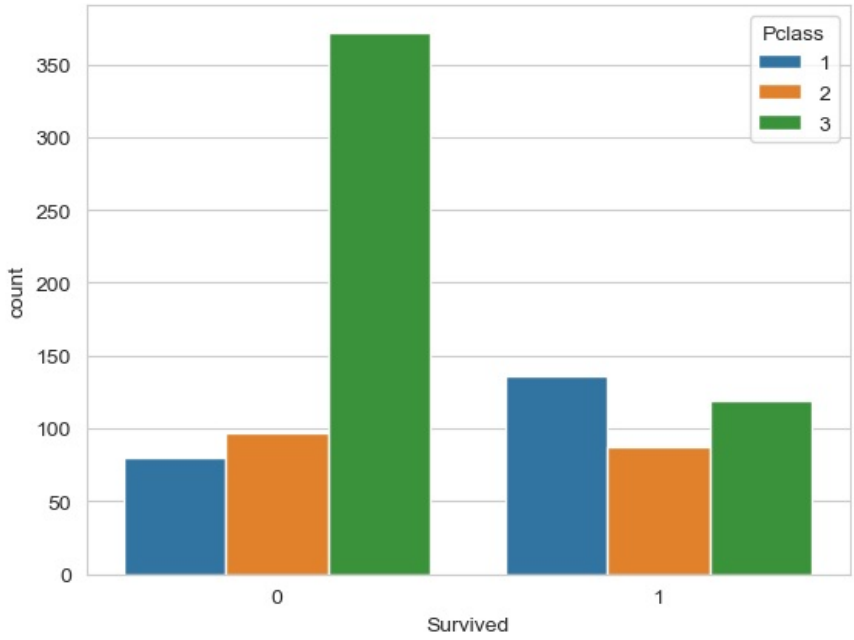
```
sns.countplot(x='Survived',hue = 'Sex', data = train,palette='RdBu_r')
```

Out[14]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



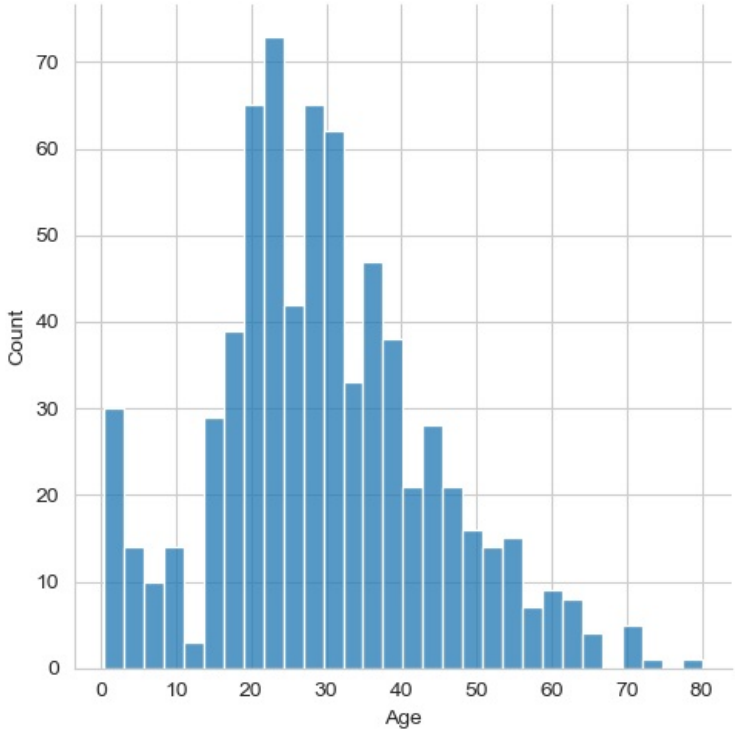
In [15]:
```
#Compared to passenger class

sns.countplot(x='Survived',hue = 'Pclass', data = train)
```

Out[15]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [17]:
```
#Let's have an idea on their age. *DROP NULL VALUES IT WILL MESS UP THE DIST

sns.displot(train['Age'].dropna(),bins=30)
```

Out[17]:
```
<seaborn.axisgrid.FacetGrid at 0x23617d6a4c0>
```

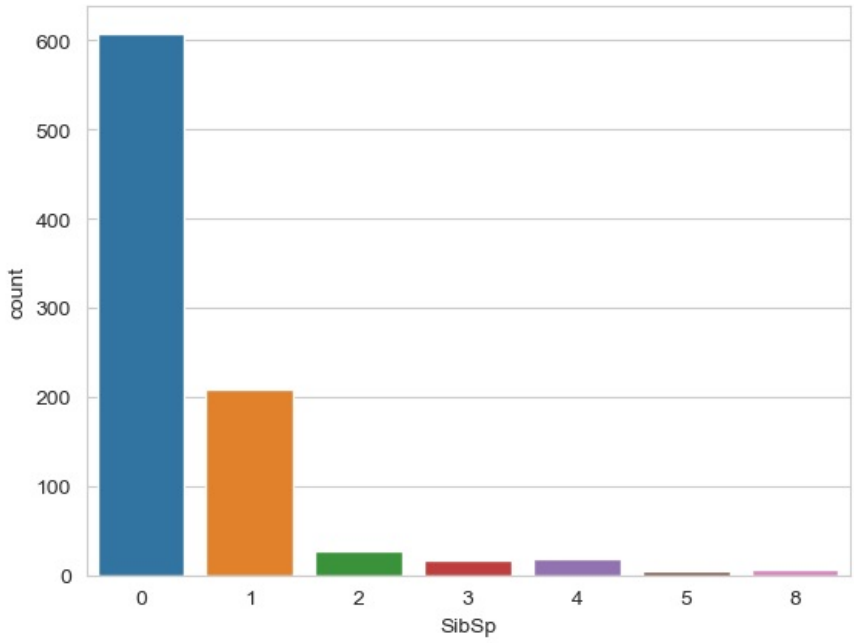


In [18]:
```
#Check the info rq

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
```

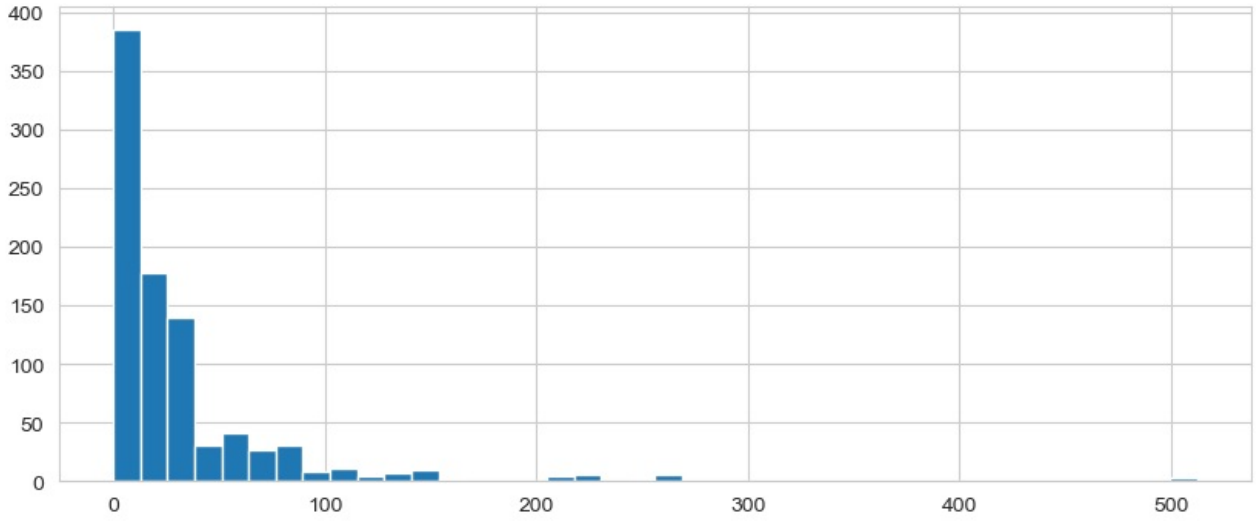
In [19]:
```
#Let's check sibling info

sns.countplot(x='SibSp',data=train) #Most people didn't have a spouse or relative on board
```

Out[19]:
```
<AxesSubplot:xlabel='SibSp', ylabel='count'>
```



In [24]:
```
#To check how much people paid

train['Fare'].hist(bins=40,figsize=(10,4))
```

Out[24]: <AxesSubplot:>

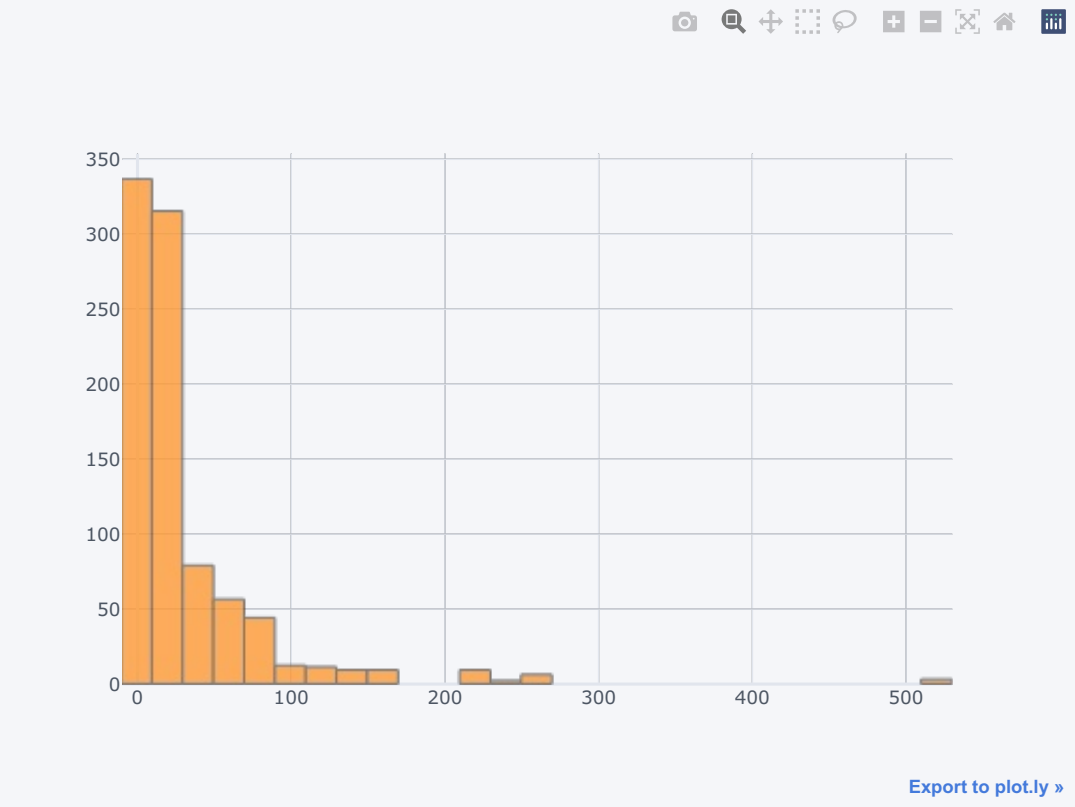


In [25]:
```
#If you wanted to make these plots interactive with cufflinks,
#here's the solution;
```

In [26]:
```
import cufflinks as cf
```

In [27]:
```
cf.go_offline()
```

In [33]:
```
train['Fare'].iplot(kind='hist',bins=50)
```



In [ ]:

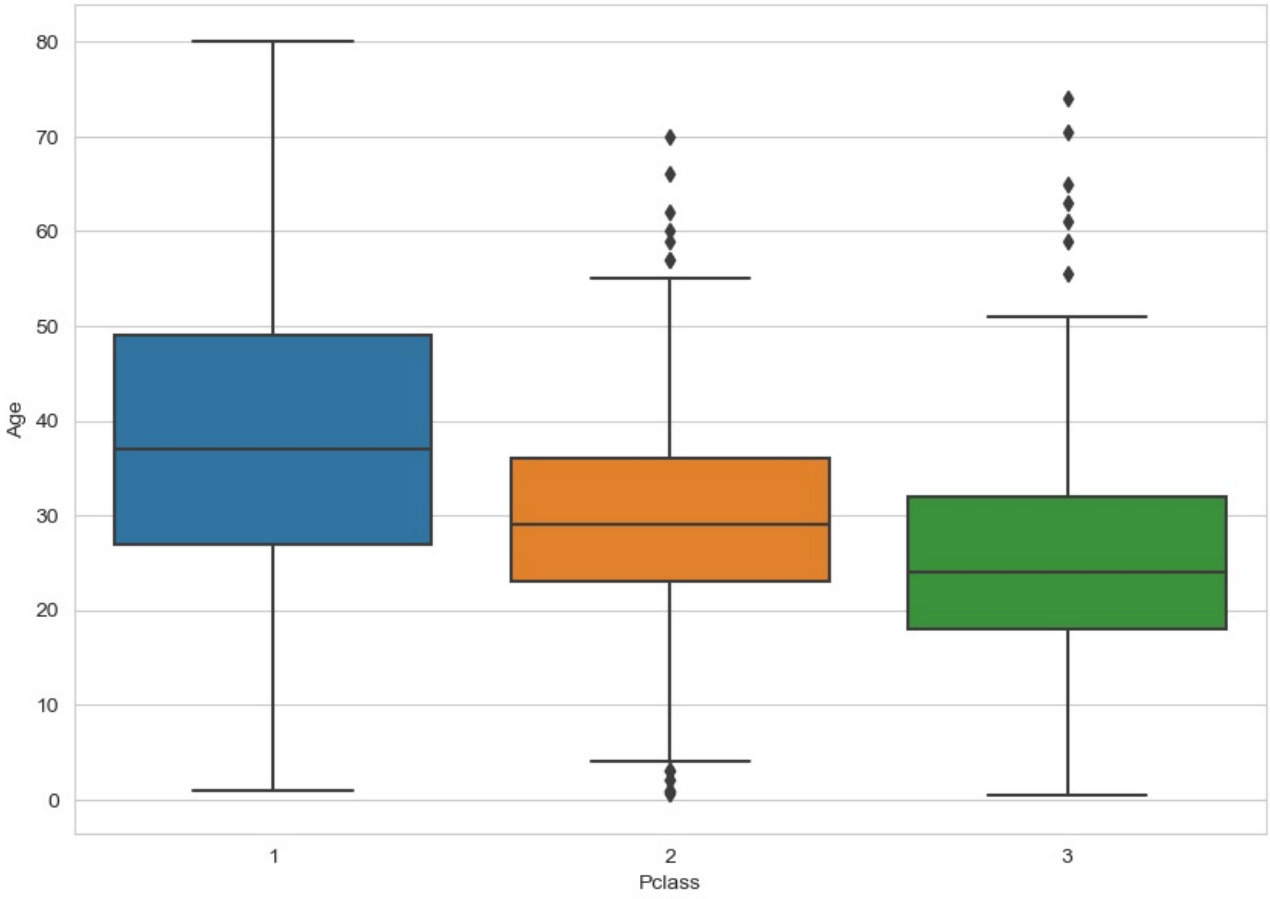
CLEANING OUR DATA - Step 2

In [34]:
```
#This will allow us to transform our data into an acceptable
# form for our machine learning algorithm
```

In [35]:
```
#Remember the missing age column? We want to find a way to fill in
# instead of dropping the column
```

In [37]:
```
#One way to do this is amputation. Which is taking the mean age of all the passengers.
# So take the average age and fill in the missing values with that.
#  What would be even better is the average age by passenger class

# Let's see

plt.figure(figsize=(10,7))
sns.boxplot(x='Pclass',y='Age',data=train)
```

Out[37]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```

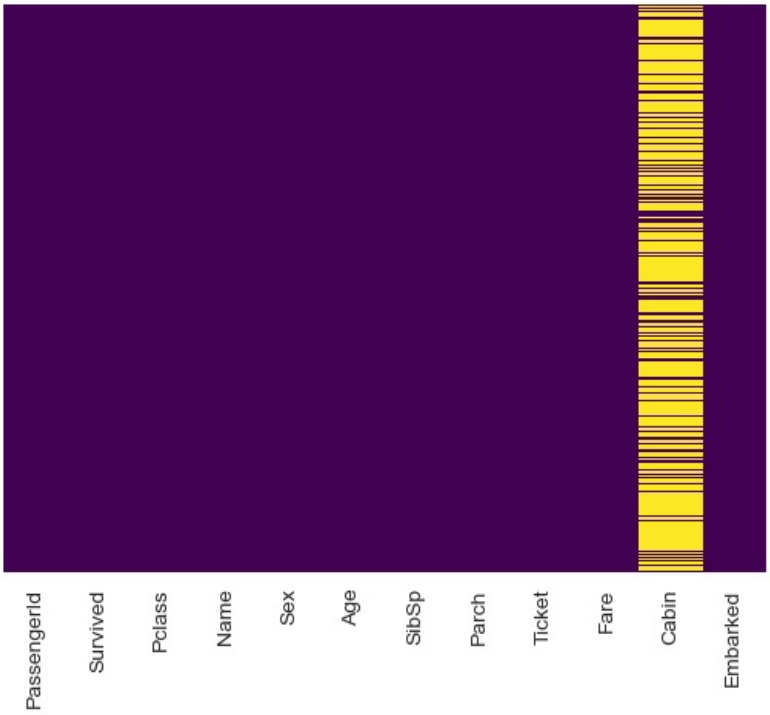


In [39]:
```
#So let's add the average age of each class to where it is missing
# in their respective classes

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 [41]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

In [44]:
```
# Now let's check our previous heatmap for further confirmation

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

#Now we have our age filled in with quite logical values
# What about the cabin data? Well there is too much data missing to form
#  an estimation with the data we already have.
```

Out[44]:
```
<AxesSubplot:>
```



In [45]:
```
#So let's just drop the whole cabin column

train.drop('Cabin',axis=1,inplace=True)
```

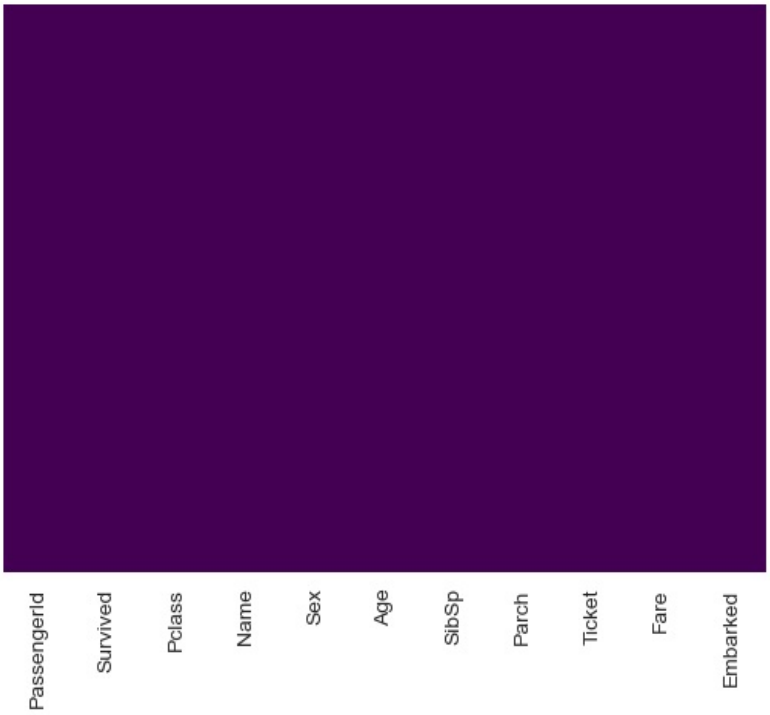
In [46]:
```
#Let's check if it is still there
train.head()
# Now it's gone
```

Out[46]:

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

In [47]:
```
#Let's rerun our heatmap to check

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

#Perfect!! No missing data
```

Out[47]:
```
<AxesSubplot:>
```



In [48]:
```
#Now what we have to do is deal with categorical features
# So convert them to a 'dummy variable' using pandas. Because
#  Categories like Sex will not be able to be taken by our machine
#   So we have to translate it to a code of 0 or 1.
```

In [49]:
```
#So we will use this panda function to convert our categorical variables
# to indicator variables

pd.get_dummies(train['Sex'])
```

Out[49]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

In [50]:
```
#However, there is a multicolinearity problem as there is two columns.
# We can and have to drop one.
#  As if female at row 2 is 'zero' it means male can only be 'one'
#So we add

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

Out[50]:

	male
0	1
1	0
2	0
3	0
4	1
...	...
886	1
887	0
888	0
889	1
890	1

891 rows × 1 columns

In [52]:
```
#Let's set it to

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

In [53]:
```
#Let's check our df
```

In [54]:
```
sex   #Looks good
```

Out[54]:

	male
0	1
1	0
2	0
3	0
4	1
...	...
886	1
887	0
888	0
889	1
890	1

891 rows × 1 columns

In [55]:
```
#Let's do it for embark

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

In [56]:
```
#Check embark
embark.head() #So it was 3 originally meaning if it isn't one of the 2 then it is the 3rd
```

Out[56]:

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

In [57]:
```
#So what we can do is let's add our new columns back to the original dataframe
train = pd.concat([train,sex,embark],axis=1)
```

In [58]:
```
#Let's get the head of train
train.head()
```

Out[58]:

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

In [59]:
```
#So we do not need the sex column and embarked column anymore
# Let's drop it and drop the columns we can't use including names and tickets
#  As our model will only work with categorical and numerical values
```

In [60]:
```
#Dropping values
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [61]:
```
train.head() #This is perfect for our machine learning algorithm
```

Out[61]:

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

In [69]:
```
#We will drop passenger id too as it is not useful for our calculations

train.drop('PassengerId', axis=1, inplace=True)
```

In [70]:
```
train.head()
```

Out[70]:

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1

In [ ]:
```

```

Building a model - Step 3

In [71]:
```
#Let's divide our data then

X = train.drop('Survived',axis=1)
y = train ['Survived']
```

In [73]:
```
from sklearn.model_selection import train_test_split
```

In [74]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [75]:
```
from sklearn.linear_model import LogisticRegression
```

In [76]:
```
logmodel = LogisticRegression()
```

In [77]:
```
logmodel.fit(X_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py: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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Out[77]:
```
LogisticRegression()
```

In [78]:
```
predictions = logmodel.predict(X_test)
```

In [78]:
```
predictions = logmodel.predict(X_test)
```

In [79]:
```
from sklearn.metrics import classification_report
```

In [80]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       154
           1       0.78      0.67      0.72       114

    accuracy                           0.78       268
   macro avg       0.78      0.77      0.77       268
weighted avg       0.78      0.78      0.78       268
```

In [81]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,predictions)
```

Out[81]:
```
array([[133,  21],
       [ 38,  76]], dtype=int64)
```

In [ ]: