Logistic Regression with Python

Titanic Data Set from Kaggle.

We'll be trying to predict a classification- survival or deceased. Let's begin our understanding of implementing Logistic Regression in Python for classification.

We'll use a "semi-cleaned" version of the titanic data set, if you use the data set hosted directly on Kaggle, you may need to do some additional cleaning not shown in this lecture notebook.

In [26]:
```
import pandas as pd
import numpy as np
```

In [27]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib in line
```

```
UsageError: unrecognized arguments: line
```

In [28]:
```
train = pd.read_csv('titanic_train.csv')
```

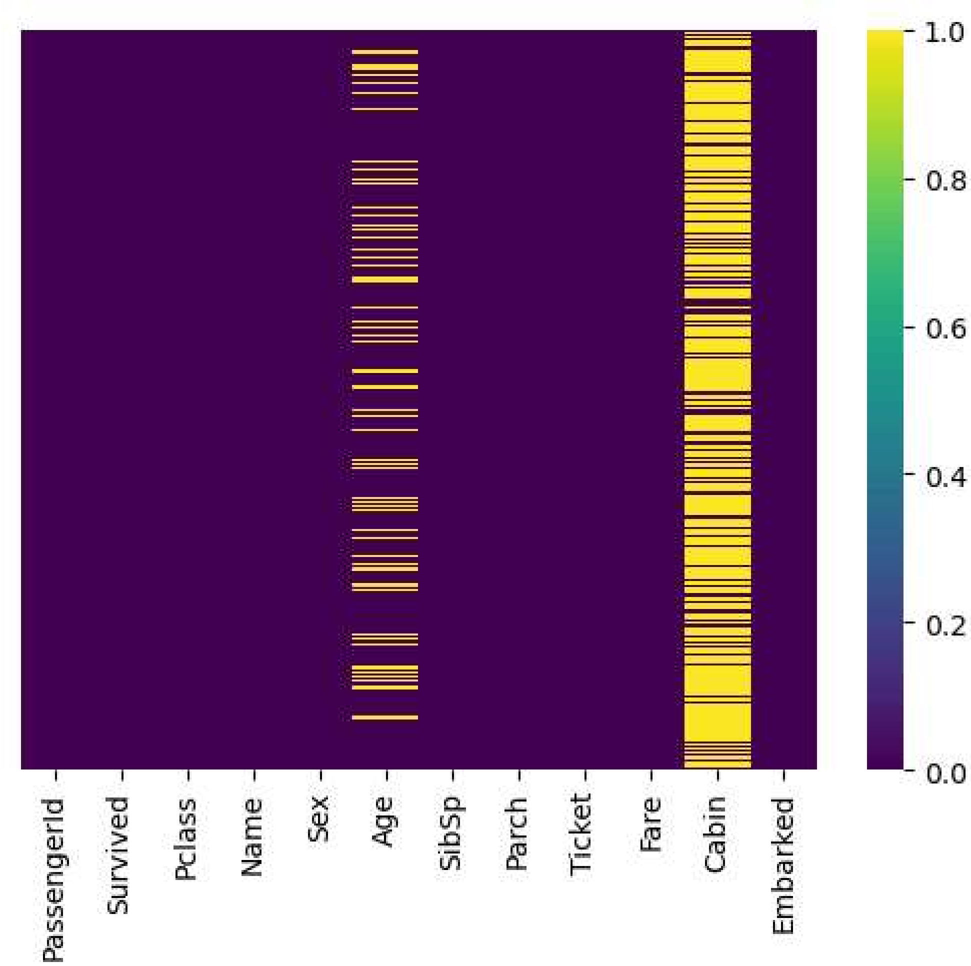
In [29]:
```
train.head()
```

Out[29]:

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

In [30]:
```
sns.heatmap(train.isnull(),yticklabels=False,cmap='viridis')
```

Out[30]: <Axes: >

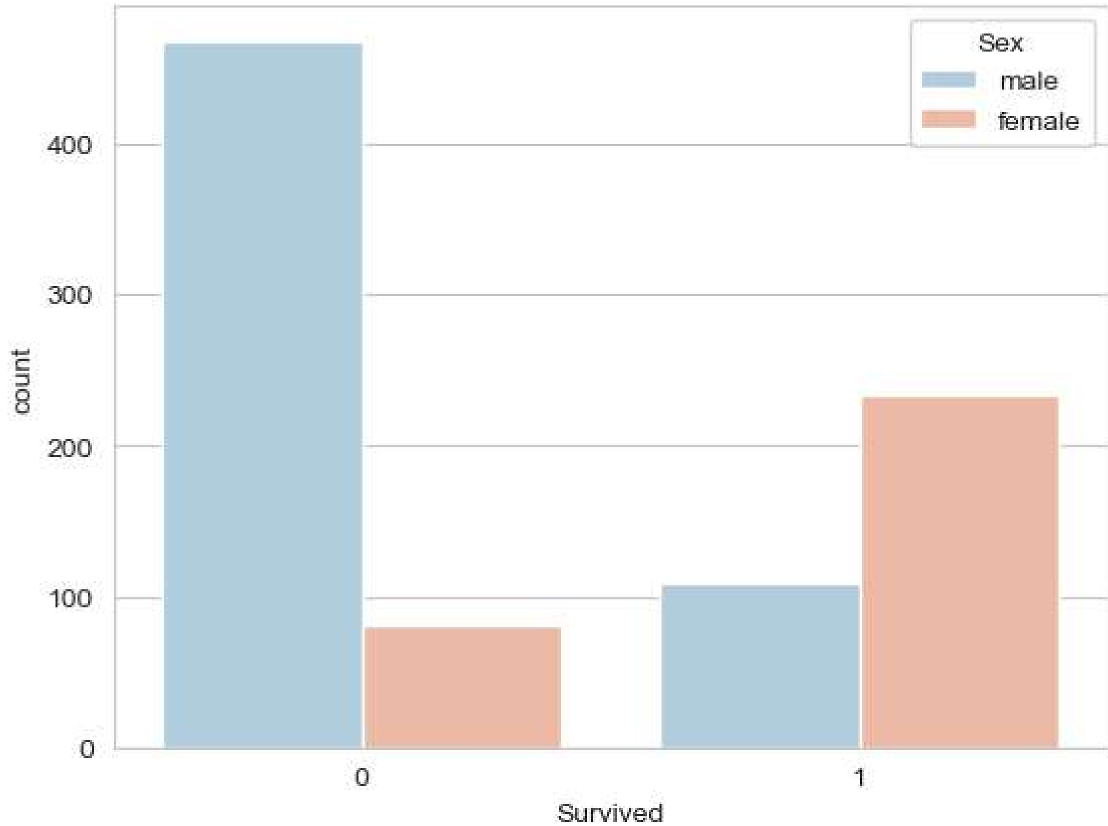


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

In [31]:
```
sns.set_style('whitegrid')
```

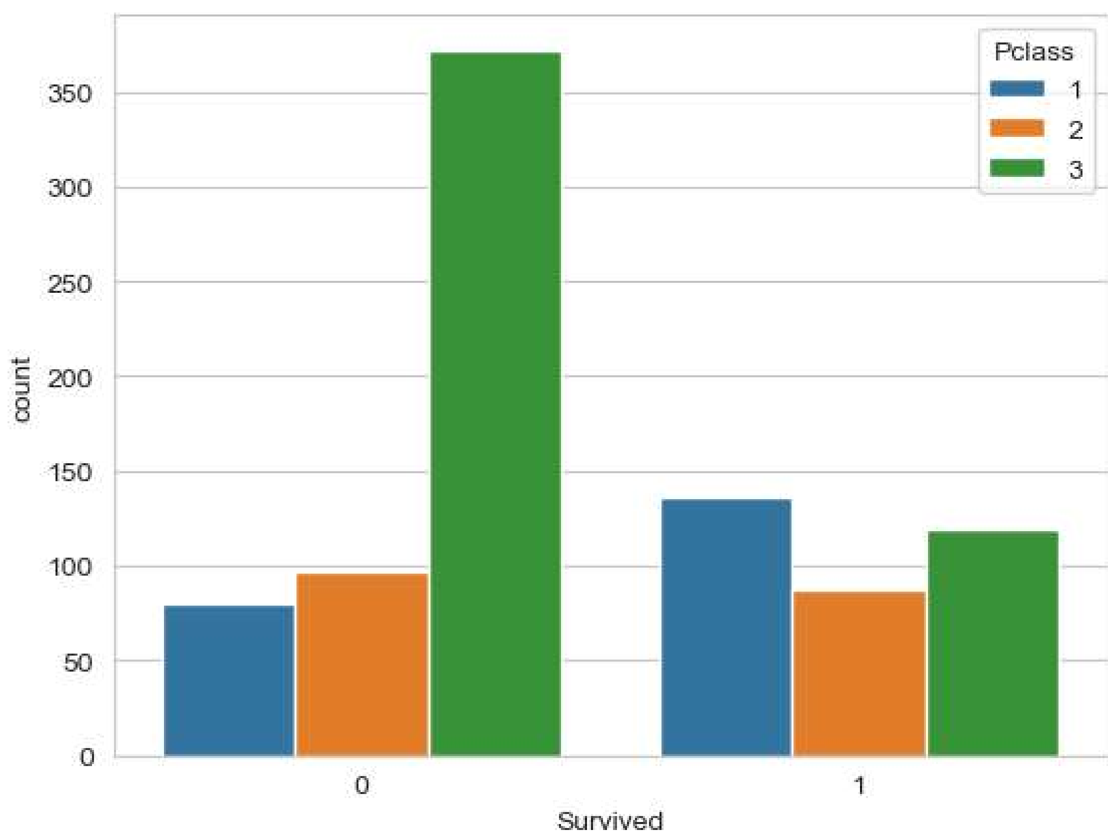
In [32]:
```
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

Out[32]: <Axes: xlabel='Survived', ylabel='count'>



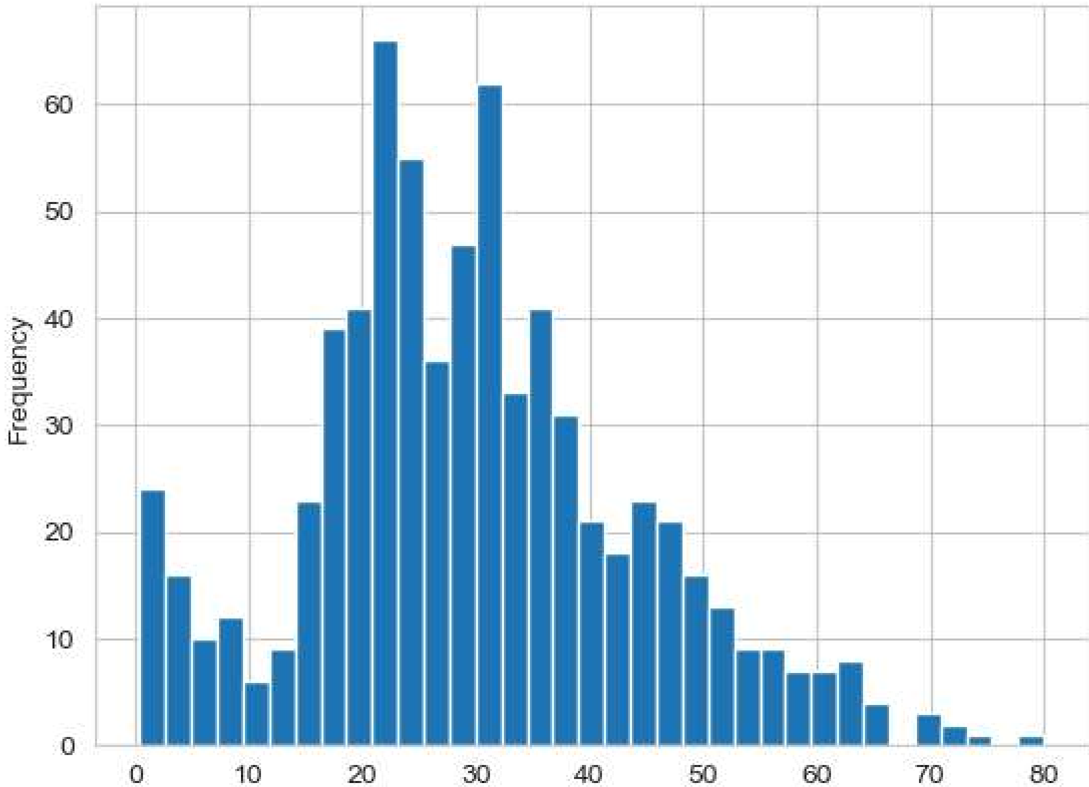
In [33]:
```
sns.countplot(x='Survived',hue='Pclass',data=train)
```

Out[33]: <Axes: xlabel='Survived', ylabel='count'>



In [34]:
```
train['Age'].plot.hist(bins=35)
```

Out[34]:
```
<Axes: ylabel='Frequency'>
```

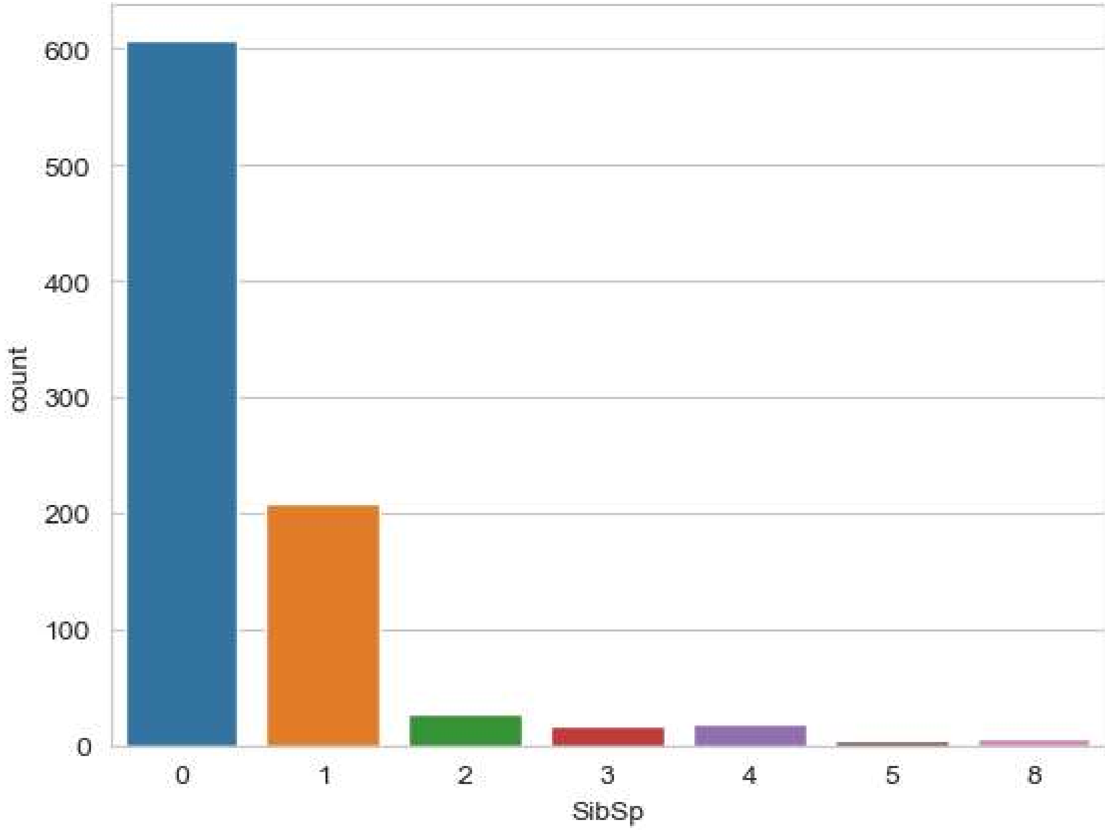


In [35]:
```
train.info()
```

```
<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
```

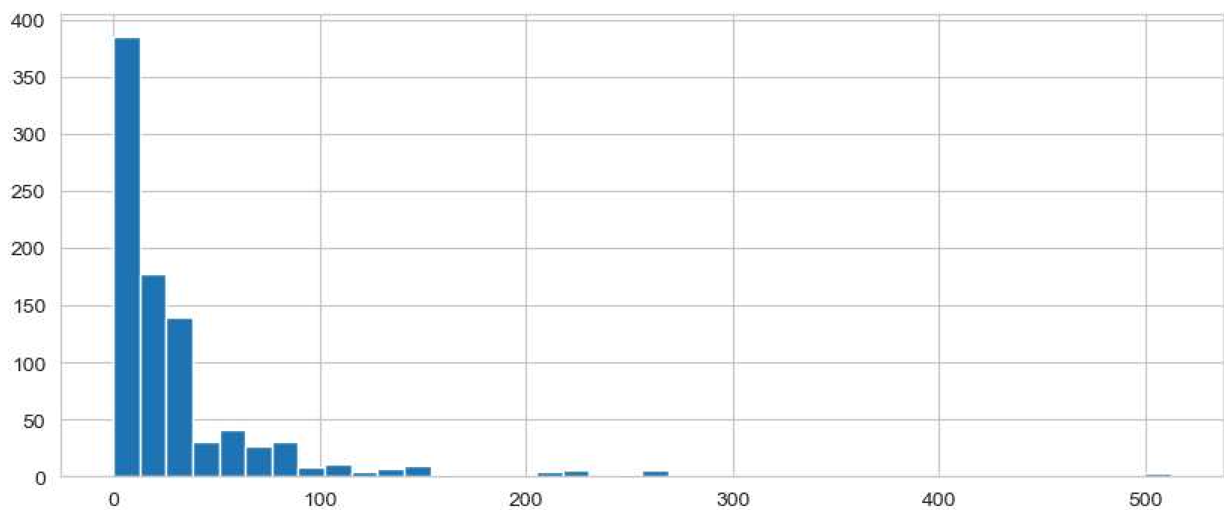
In [36]:
```
sns.countplot(x='SibSp',data=train)
```

Out[36]:
```
<Axes: xlabel='SibSp', ylabel='count'>
```



In [37]:
```
train['Fare'].hist(bins=40,figsize=(10,4))
```

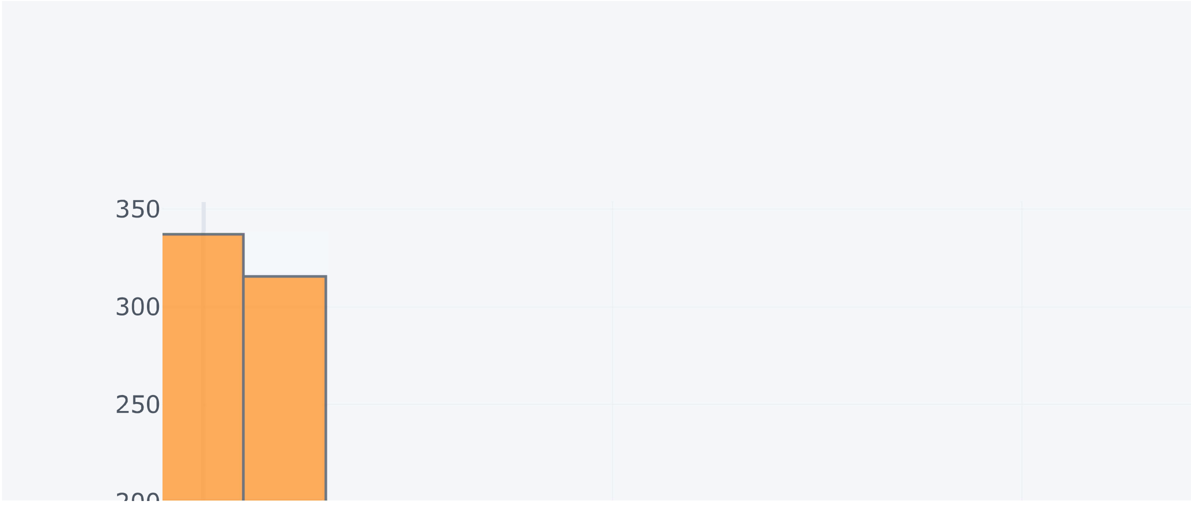
Out[37]: <Axes: >



In [38]:
```
import cufflinks as cf
```

In [39]:
```
cf.go_offline()
```

In [40]:
```
train['Fare'].iplot(kind='hist',bins=50)
```

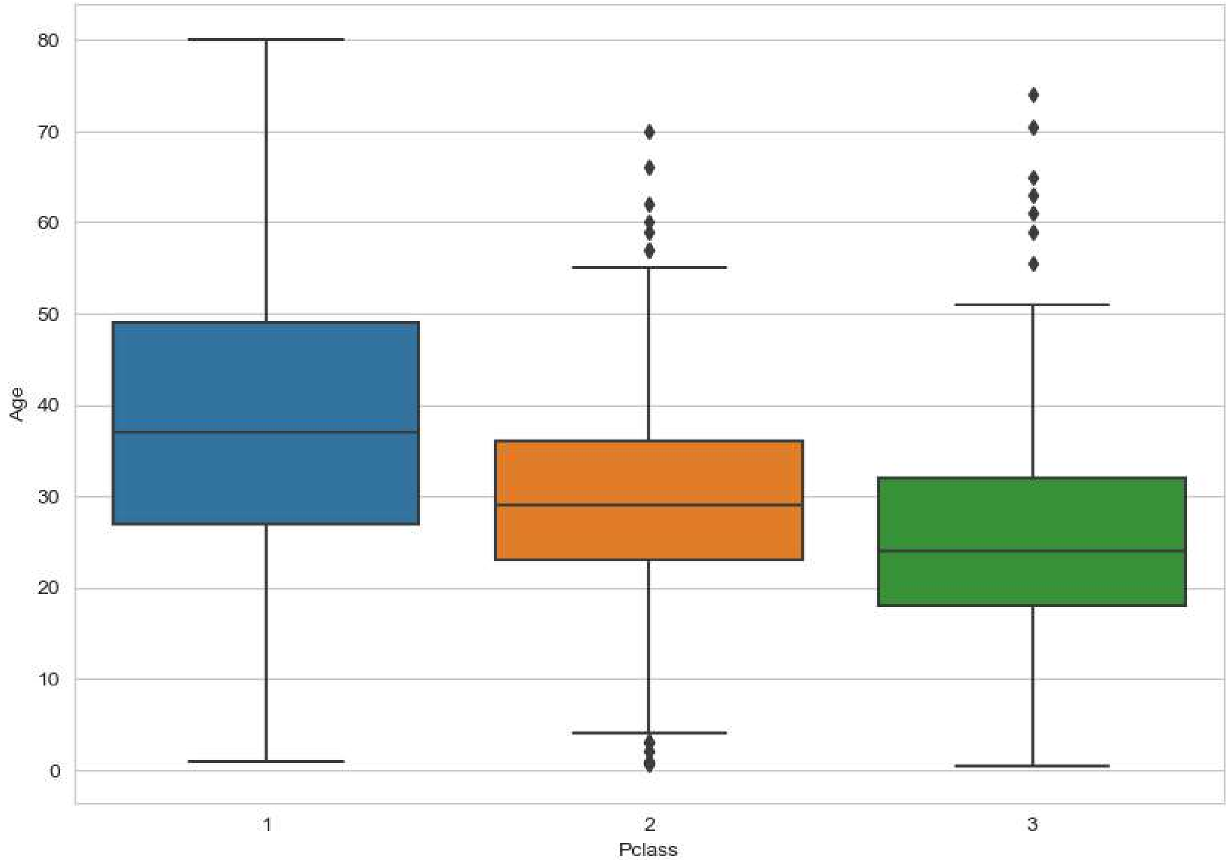


Data Cleaning

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). However we can be smarter about this and check the average age by passenger class. For example:

In [41]:
```
plt.figure(figsize=(10,7))
sns.boxplot(x='Pclass',y='Age',data=train)
```

Out[41]:
```
<Axes: xlabel='Pclass', ylabel='Age'>
```



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.

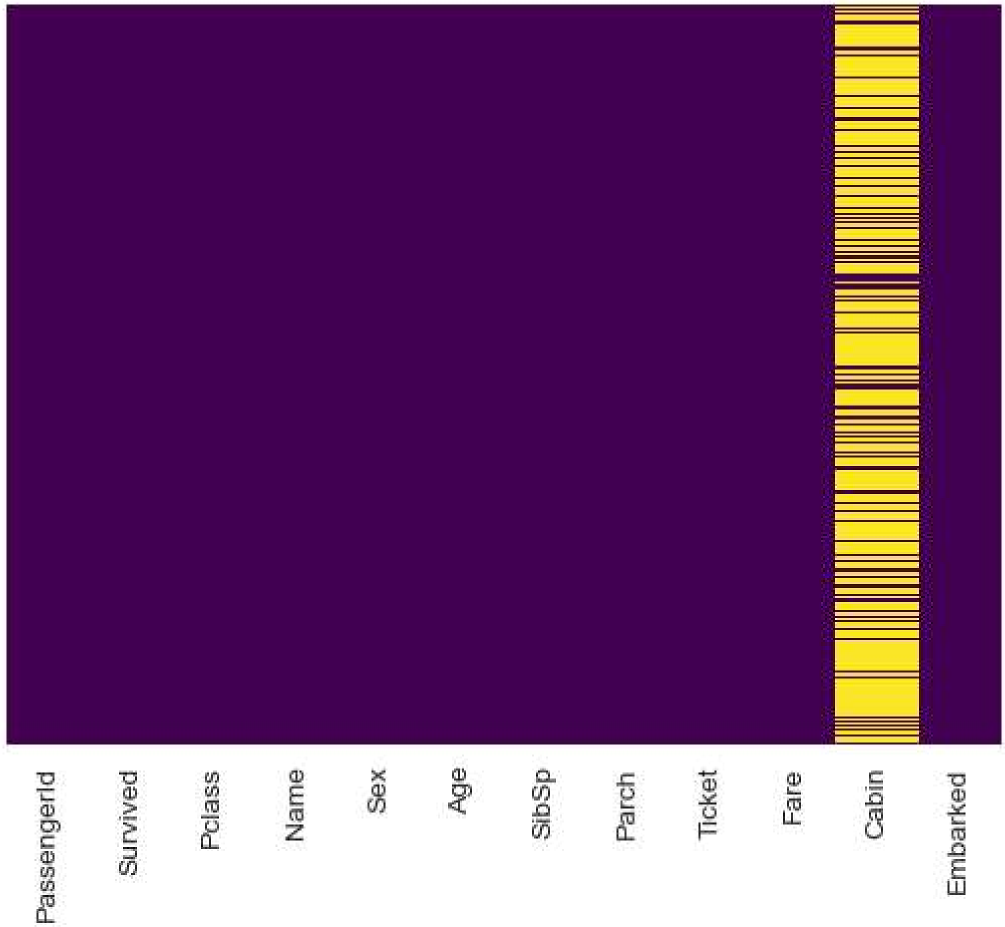
In [42]:
```
def impute_age(cols):
    Age=cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):
        if Pclass  == 1:
            return 37
        elif Pclass ==2:
            return 29
        else:
            return 24
    else:
        return Age
```

In [43]:
```
train['Age']=train[['Age','Pclass']].apply(impute_age,axis=1)
```

In [44]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

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



In [45]:
```
train.drop('Cabin',axis=1,inplace=True)
```

In [46]:
```
train.head()
```

Out[46]:

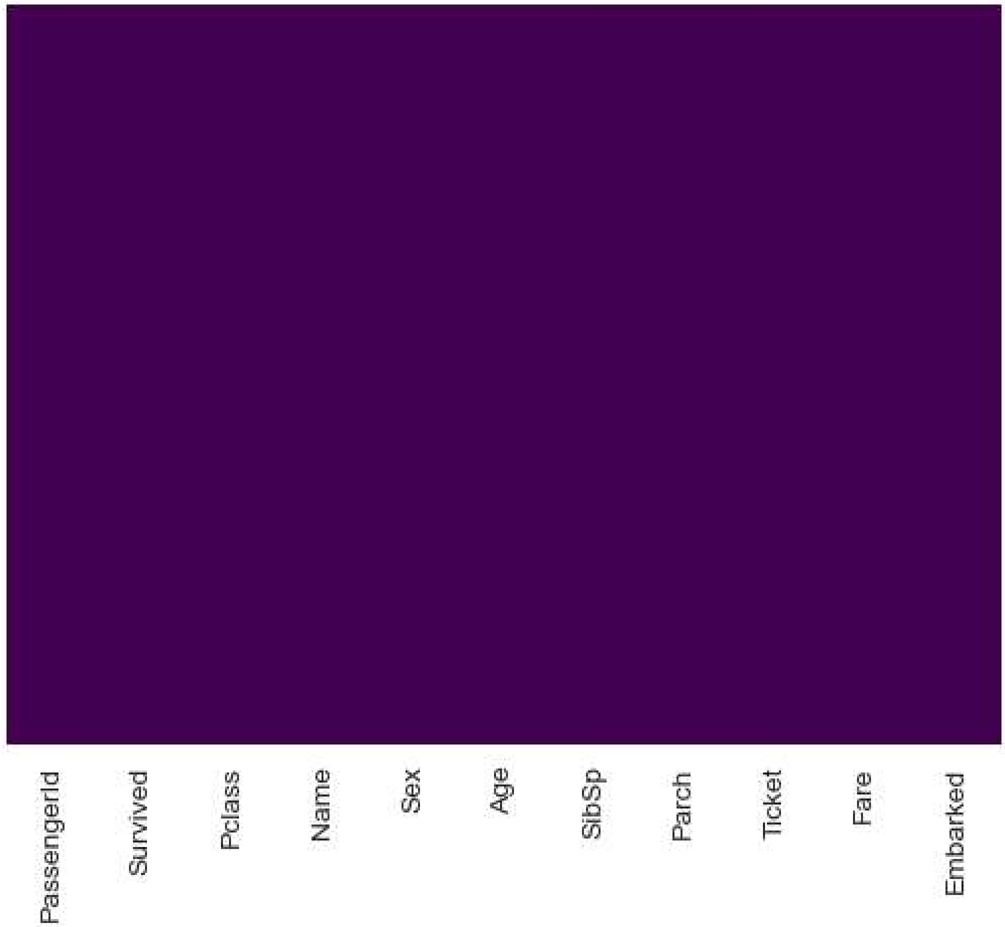
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarkec
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	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]:

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

Out[47]:

```
<Axes: >
```



Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

In [48]:
```
train.info()
```

```
<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
```

In [49]:
```
pd.get_dummies(train['Sex'],drop_first=True)
```

Out[49]:

	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 [50]:
```
sex=pd.get_dummies(train['Sex'],drop_first=True)
```

In [51]:
```
embark=pd.get_dummies(train['Embarked'],drop_first=True)
```

In [52]:
```
train=pd.concat([train,sex,embark],axis=1)
```

In [53]:
```
train.head(2)
```

Out[53]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	m
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	

In [54]:
```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [55]:
```
train.head()
```

Out[55]:

	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 [56]:
```
train.drop('PassengerId',axis=1,inplace=True)
```

In [57]:
```
train.head()
```

Out[57]:

	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

Building a Logistic Regression model

Let's start by splitting our data into a training set and test set (there is another test.csv file that you can play around with in case you want to use all this data for training).

Train Test Split

In [58]:
```
X=train.drop('Survived',axis=1)
y=train['Survived']
```

In [59]:
```
from sklearn.model_selection import train_test_split
```

In [60]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=101)
```

In [61]:
```
from sklearn.linear_model import LogisticRegression
```

In [62]:
```
logmodel=LogisticRegression()
```

In [63]:
```
logmodel.fit(X_train,y_train)
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:469: Conv
ergenceWarning:

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[63]:

```
▾ LogisticRegression
LogisticRegression()
```

In [64]:

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

Evaluation

In [65]:

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

In [66]:

```
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 [ ]: