Import Libraries

first Import Required libraries !

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt;
import seaborn as sns
%matplotlib inline
```

Reading The Data

In [2]:
```
train = pd.read_csv('train.csv')
```

train.info()

In [3]:
```
train.head()
```

Out[3]:

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

In [4]:
```
train.tail()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN

Exploratory Data Analysis

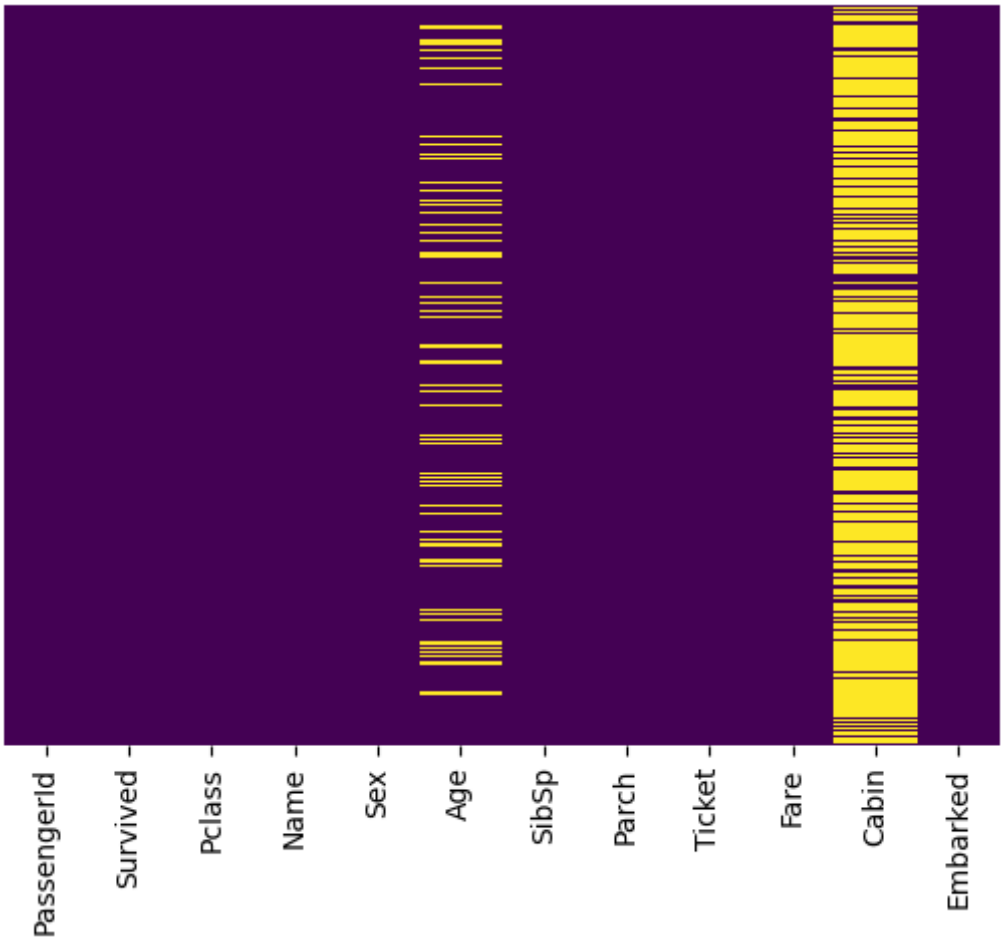
To Start Exploratory Data Analysis By checking out missing data!

Missing Data

I can use seaborn to create a simple heatmap to see where we are missing data!

In [5]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[5]:
```
<AxesSubplot:>
```

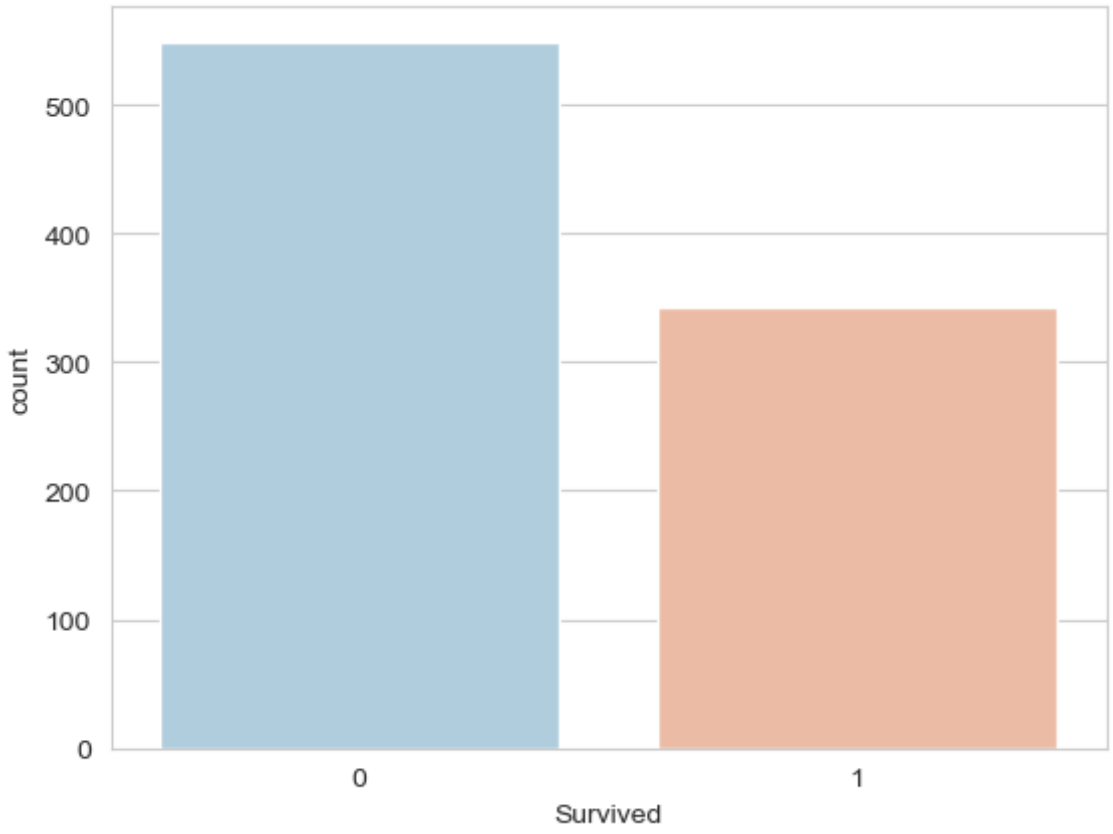


In [6]:
```
train.isnull().sum().sort_values(ascending=False)
```

Out[6]:
```
Cabin          687
Age            177
Embarked         2
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
SibSp            0
Parch            0
Ticket           0
Fare             0
dtype: int64
```

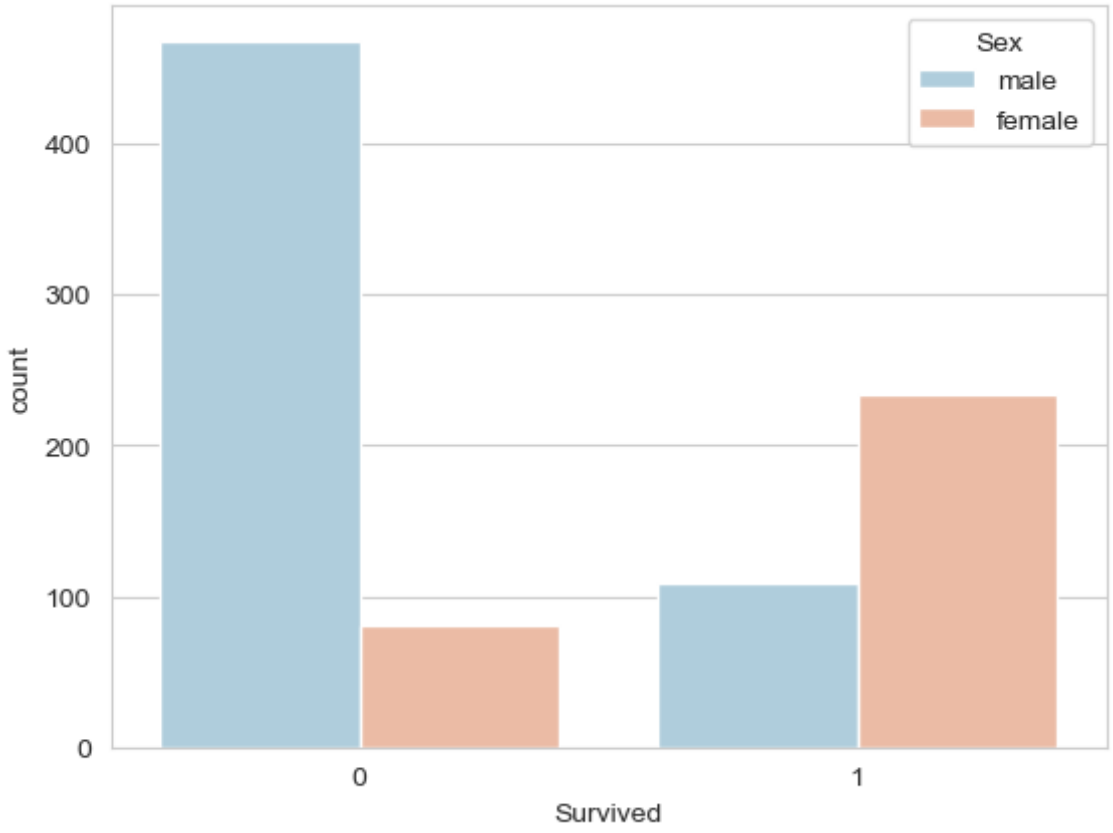
In [7]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```

Out[7]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



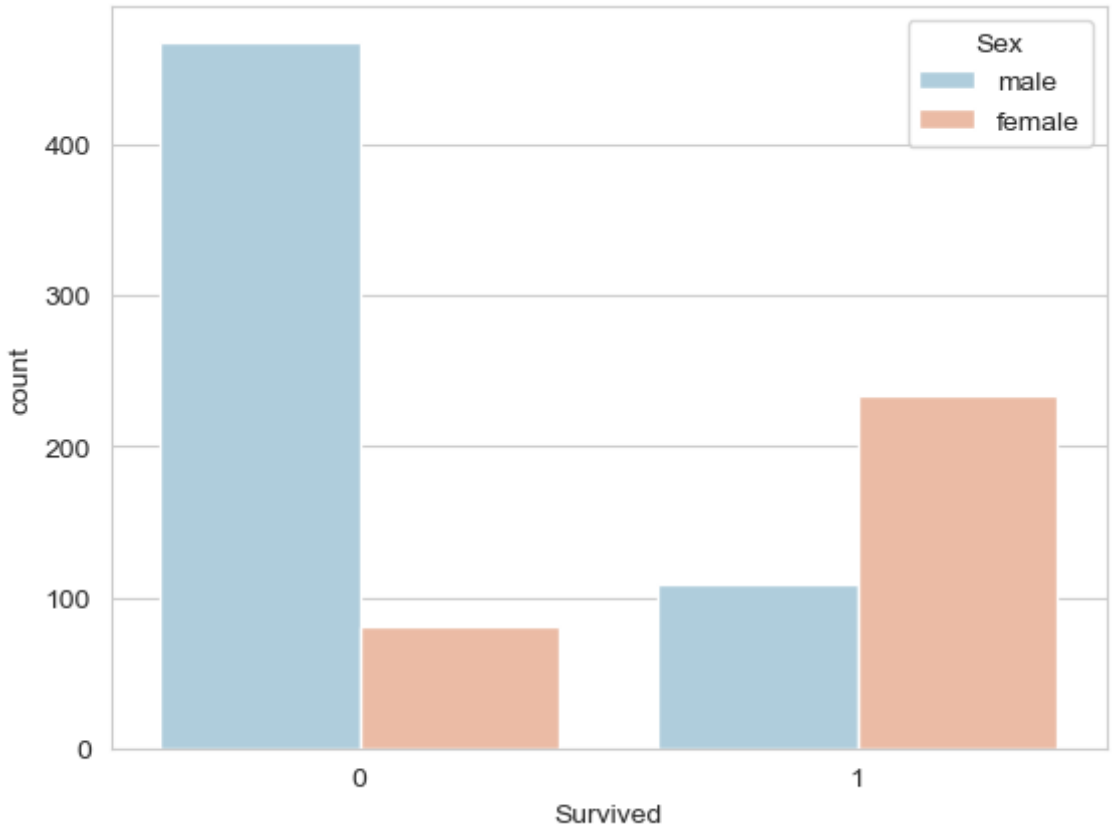
In [8]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

Out[8]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



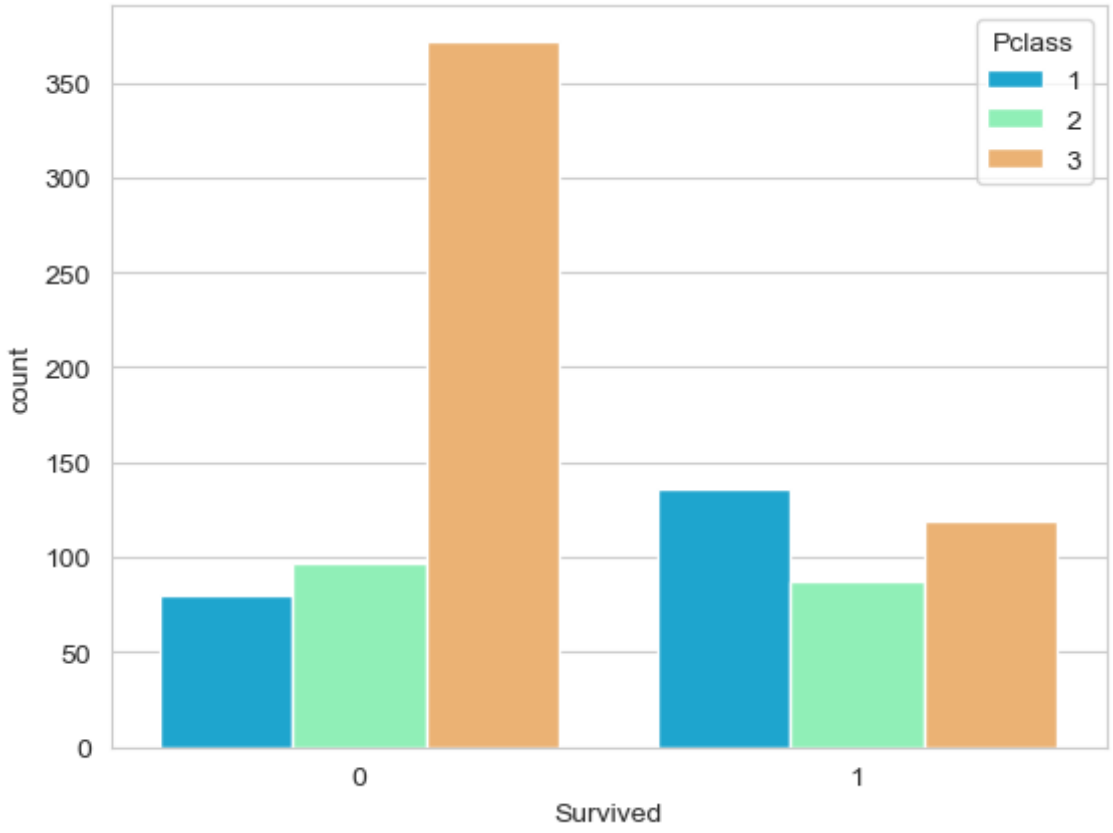
In [9]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

Out[9]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



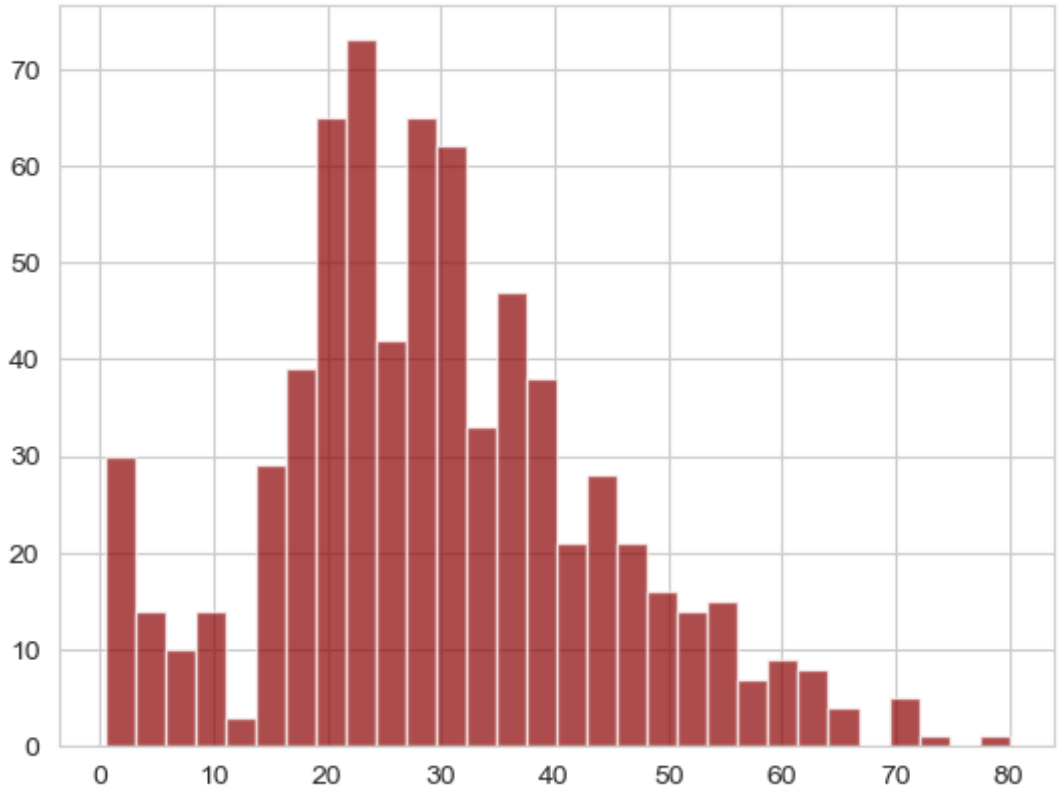
In [10]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

Out[10]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



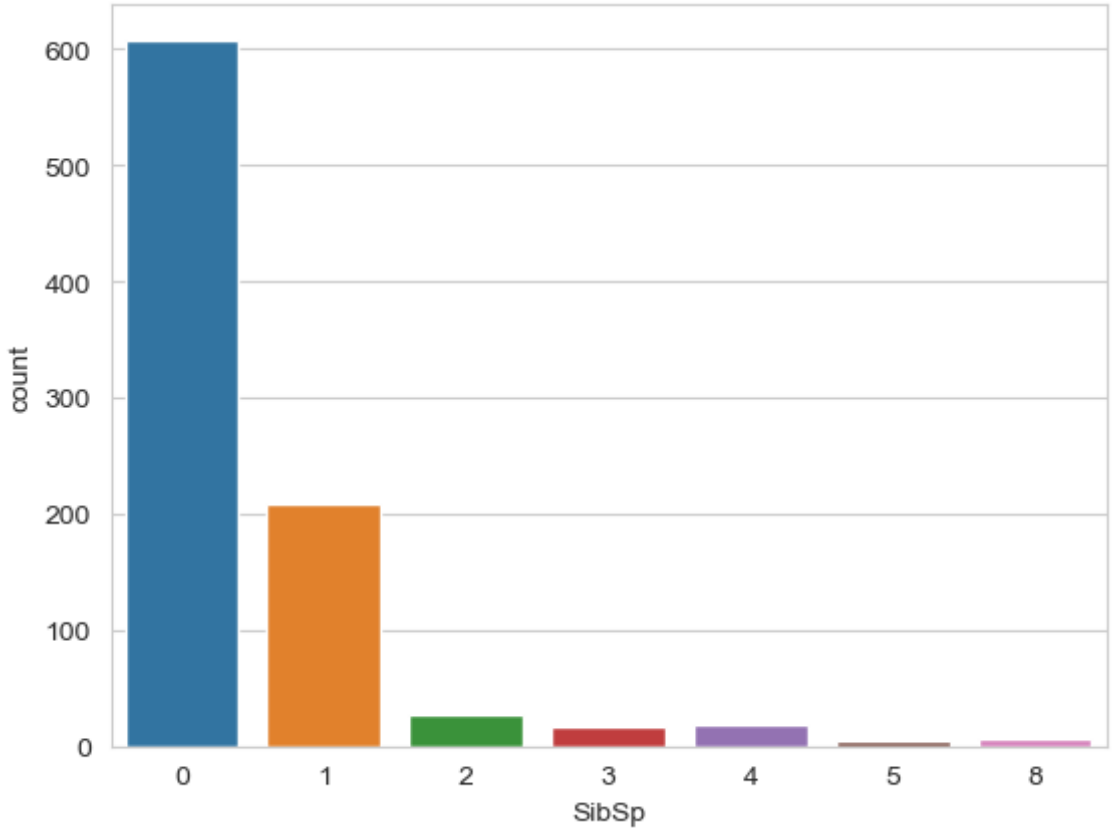
In [11]:
```
train['Age'].hist(bins=30,color='darkred',alpha=0.7)
```

Out[11]:
```
<AxesSubplot:>
```



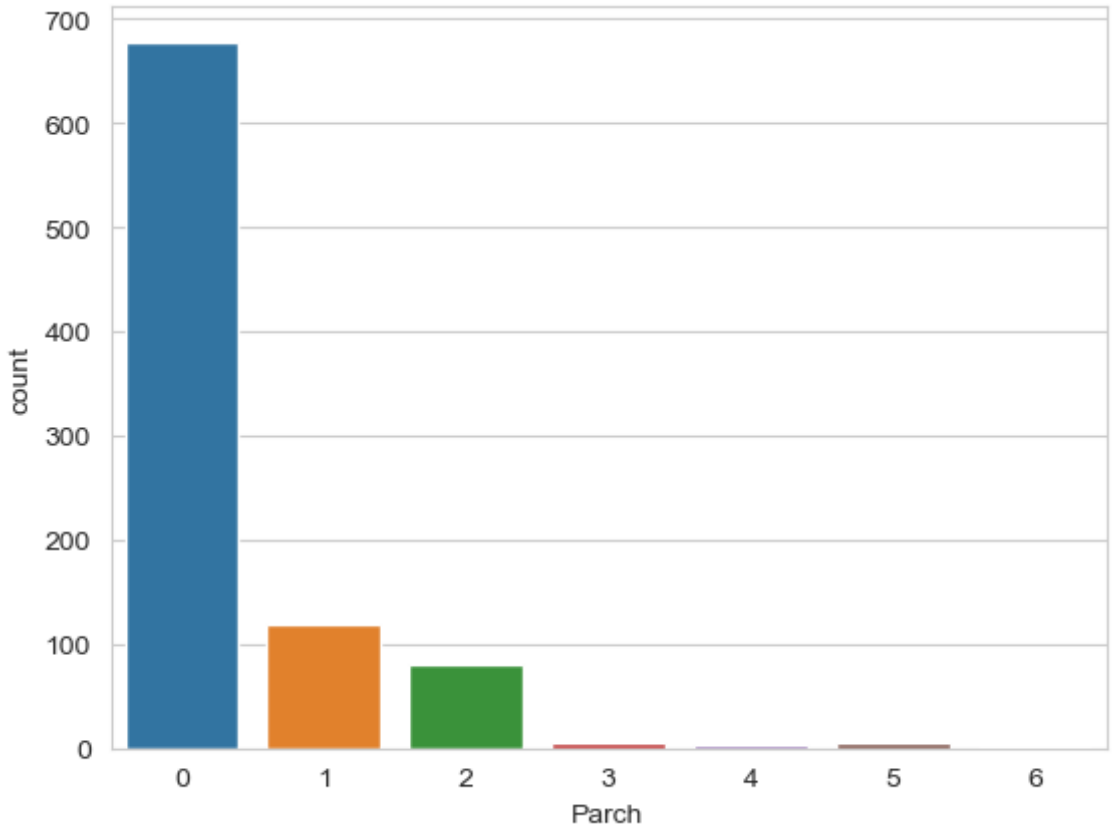
In [12]:
```
sns.countplot(x='SibSp',data=train)
```

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



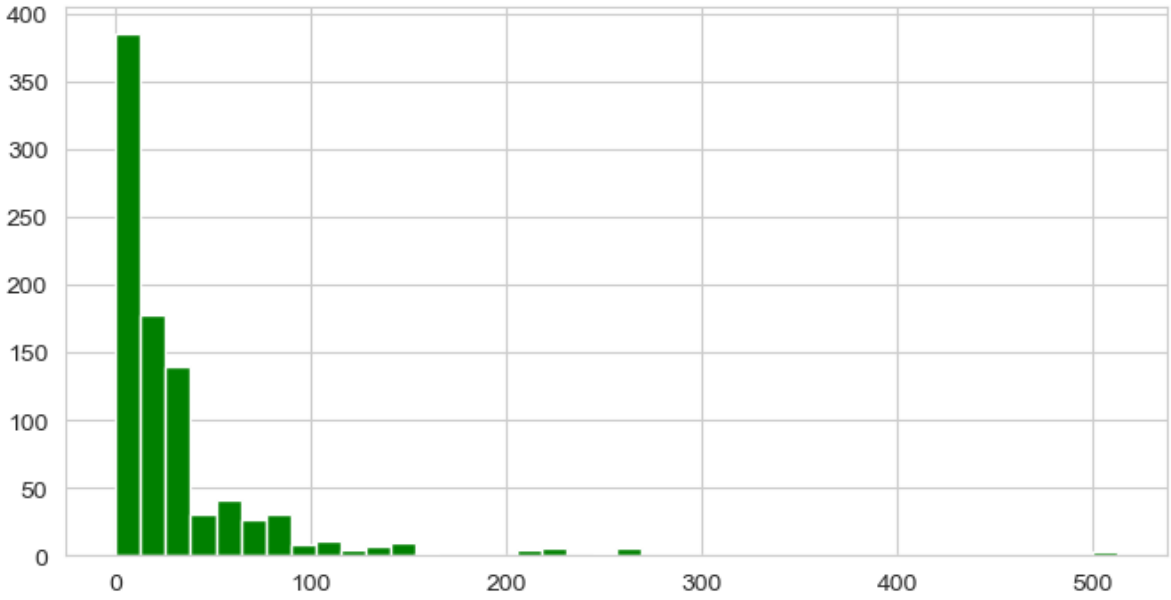
In [13]:
```
sns.countplot(x='Parch',data=train)
```

Out[13]:
```
<AxesSubplot:xlabel='Parch', ylabel='count'>
```



In [14]:
```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

Out[14]:
```
<AxesSubplot:>
```

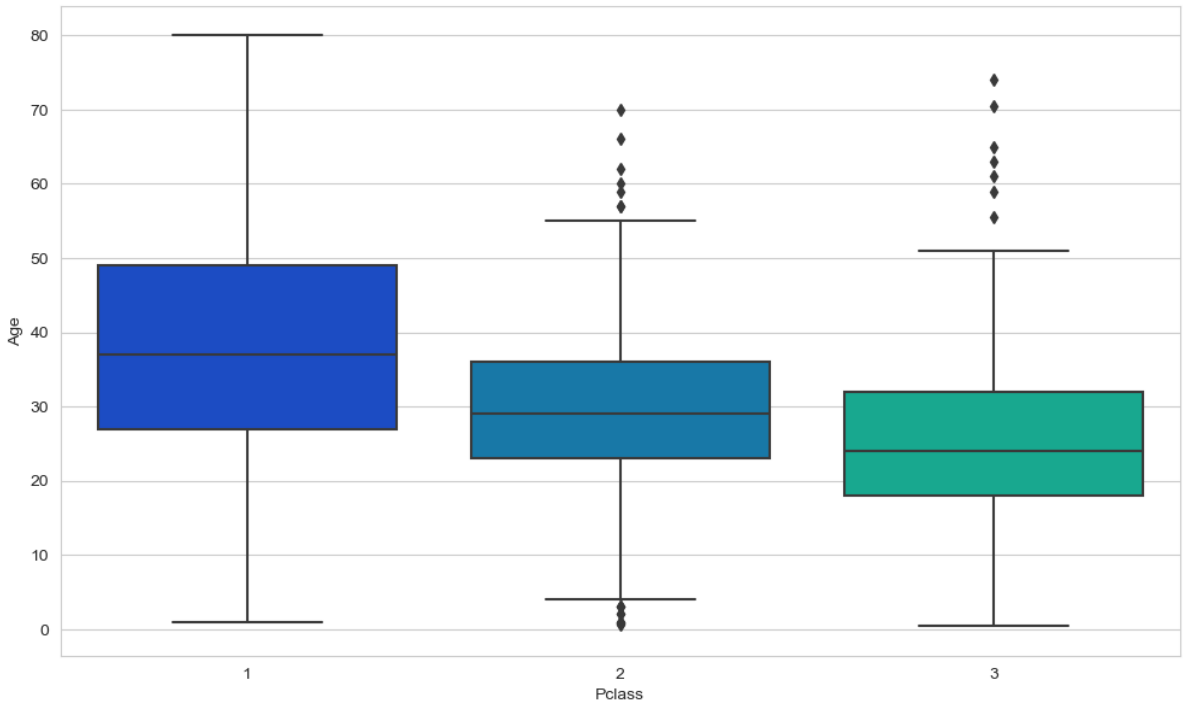


Data Cleaning

I 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 [15]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

Out[15]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



We can see the Rich 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 [1]:
```
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
```

Now apply that function!

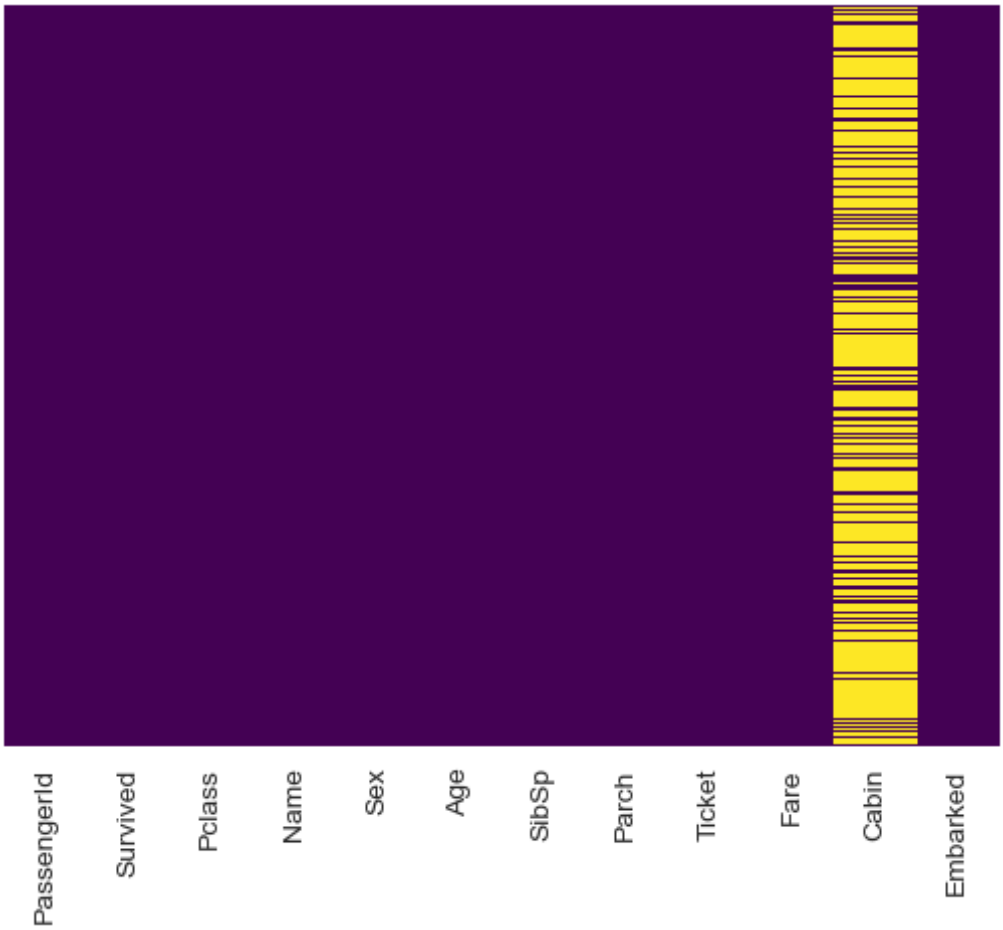
In [17]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

In [18]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

In [19]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[19]:
```
<AxesSubplot:>
```



Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.

We will sum of family member

In [20]:
```
train.drop('Cabin',axis=1,inplace=True)
```

In [21]:
```
train.head()
```

Out[21]:

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

In [22]:
```
train.dropna(inplace=True)
```

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 [23]:
```
train.info()
```

```
<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 [24]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

In [25]:
```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [26]:
```
train = pd.concat([train,sex,embark],axis=1)
```

In [27]:
```
train.head()
```

Out[27]:

	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

Great! Our data is ready for our model!

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

Great! Our data is ready for our model!

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 [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop(['Survived'],axis=1
```

Training and Predicting

In [30]:
```
from sklearn.linear_model import LogisticRegression
```

In [31]:
```
logmodel = LogisticRegression()
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
  n_iter_i = _check_optimize_result(
```

Out[31]:
```
LogisticRegression()
```

In [32]:
```
predictions = logmodel.predict(X_test)
X_test.head()
```

Out[32]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	male	Q	S
511	512	3	24.0	0	0	8.05	1	0	1
613	614	3	24.0	0	0	7.75	1	1	0
615	616	2	24.0	1	2	65.00	0	0	1
337	338	1	41.0	0	0	134.50	0	0	0
718	719	3	24.0	0	0	15.50	1	1	0

In [33]:
```
predictions
```

Out[33]:
```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0], dtype=int64)
```

Evaluation

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

In [34]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [35]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[53  4]
 [ 9 23]]
```

In [36]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89        57
           1       0.85      0.72      0.78        32

    accuracy                           0.85        89
   macro avg       0.85      0.82      0.84        89
weighted avg       0.85      0.85      0.85        89
```

Decision Tree Classifiction

In [37]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [38]:
```
dt_model=DecisionTreeClassifier()
dt_model.fit(X_train,y_train)
```

Out[38]:
```
DecisionTreeClassifier()
```

In [39]:
```
dt_pred = dt_model.predict(X_test)
```

In [40]:
```
print(confusion_matrix(y_test,dt_pred))
```

```
[[48  9]
 [ 5 27]]
```

In [41]:
```
print(classification_report(y_test,dt_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.84      0.87        57
           1       0.75      0.84      0.79        32

    accuracy                           0.84        89
   macro avg       0.83      0.84      0.83        89
weighted avg       0.85      0.84      0.84        89
```

Random Forest Classification

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [43]:
```
rf= RandomForestClassifier(n_estimators=500)
rf.fit(X_train,y_train)
```

Out[43]:
```
RandomForestClassifier(n_estimators=500)
```

In [44]:
```
rf_pre=rf.predict(X_test)
```

In [45]:
```
print(confusion_matrix(y_test,rf_pre))
```

```
[[52  5]
 [ 8 24]]
```

In [46]:
```
print(classification_report(y_test,rf_pre))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89        57
           1       0.83      0.75      0.79        32

    accuracy                           0.85        89
   macro avg       0.85      0.83      0.84        89
weighted avg       0.85      0.85      0.85        89
```

XGBoosts Classifier

In [47]:
```
from xgboost import XGBClassifier
xgboost = XGBClassifier(n_estimators=1000)
xgboost.fit(X_train,y_train)
```

Out[47]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=1000, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [48]:
```
xg_pred = xgboost.predict(X_test)
```

In [49]:
```
print(confusion_matrix(y_test,xg_pred))
```

```
[[52  5]
 [ 9 23]]
```

In [50]:
```
print(classification_report(y_test,xg_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88        57
           1       0.82      0.72      0.77        32

    accuracy                           0.84        89
   macro avg       0.84      0.82      0.82        89
weighted avg       0.84      0.84      0.84        89
```