Titanic Classification :

Build a predictive model to determine the likelihood of survival for passengers on the Titanic using data science techniques in Python.

In [1]:
```
import numpy as np                          # linear algebra
import pandas as pd                         # data processing, CSV file I/O (e.g. pd.read_
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

The Data

reading in the titanic_train.csv file into a pandas dataframe.

In [2]:
```
train = pd.read_csv('C:\Users\swain\Downloads\titanic_train.csv')
```

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

Out[3]:

	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	

Missing Data

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

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

Out[4]:

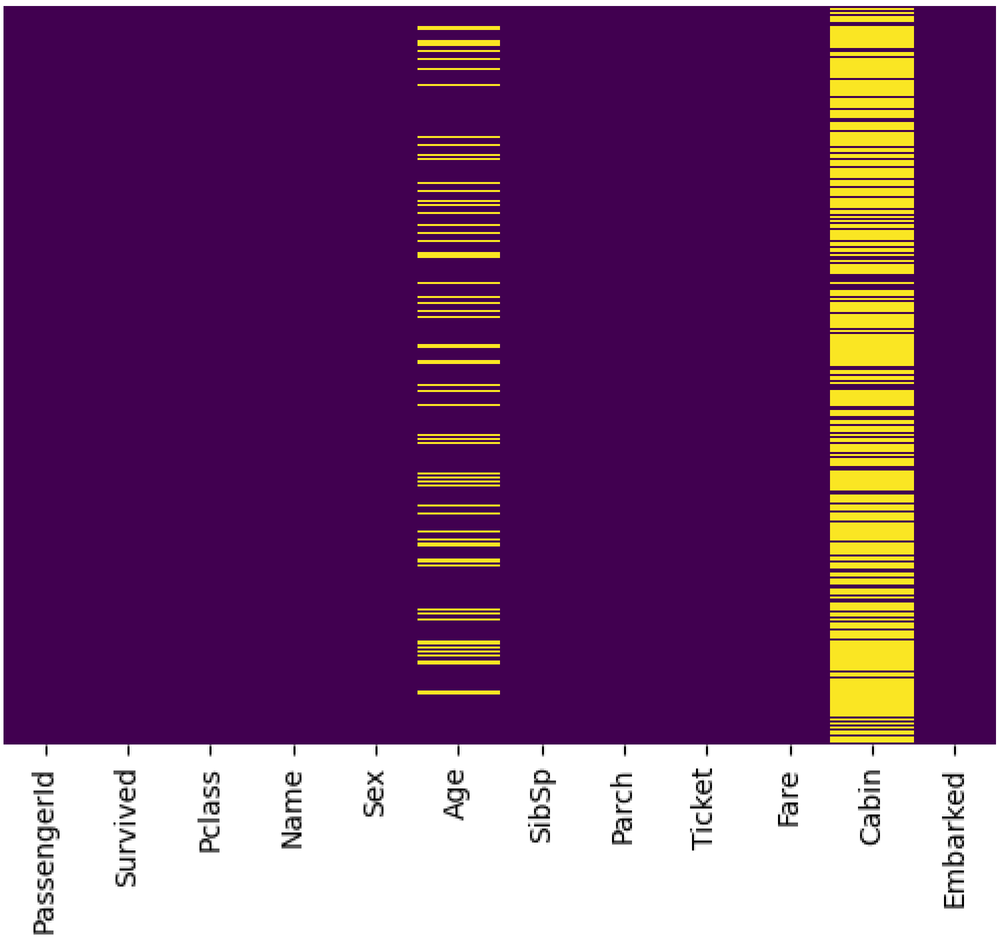
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	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 [5]:

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

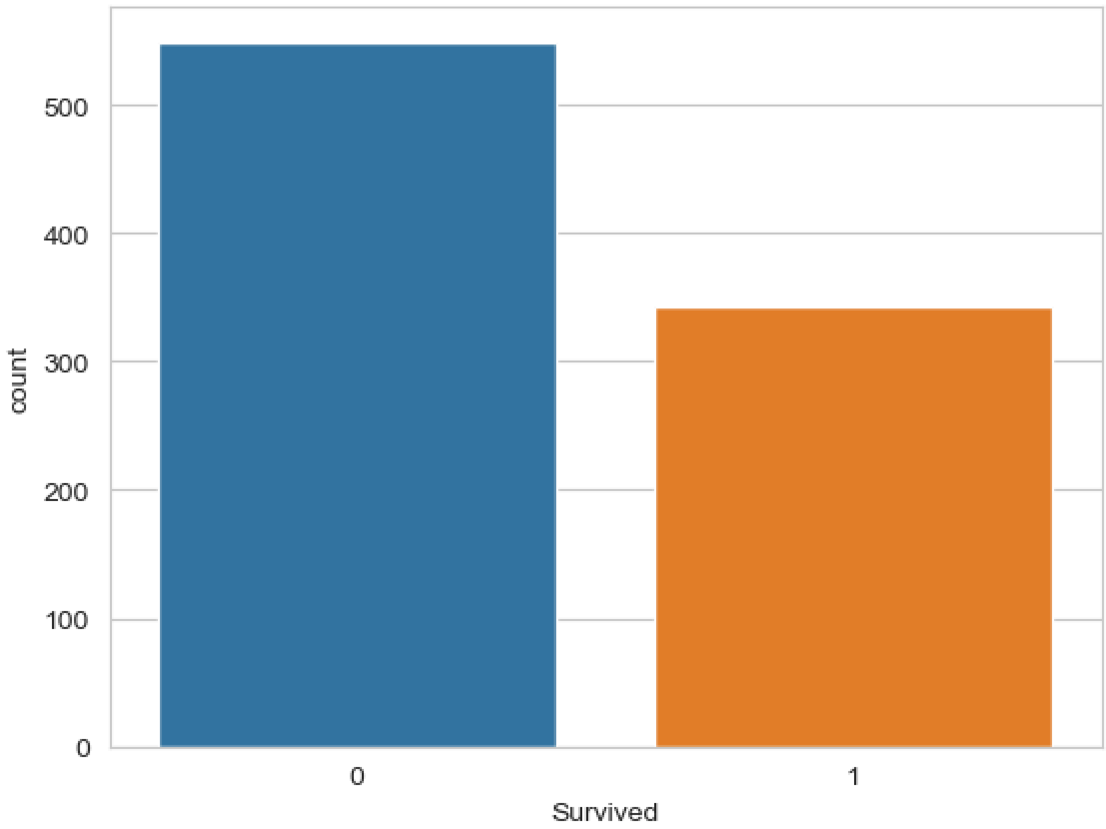
Out[5]: <Axes: >



In [6]:

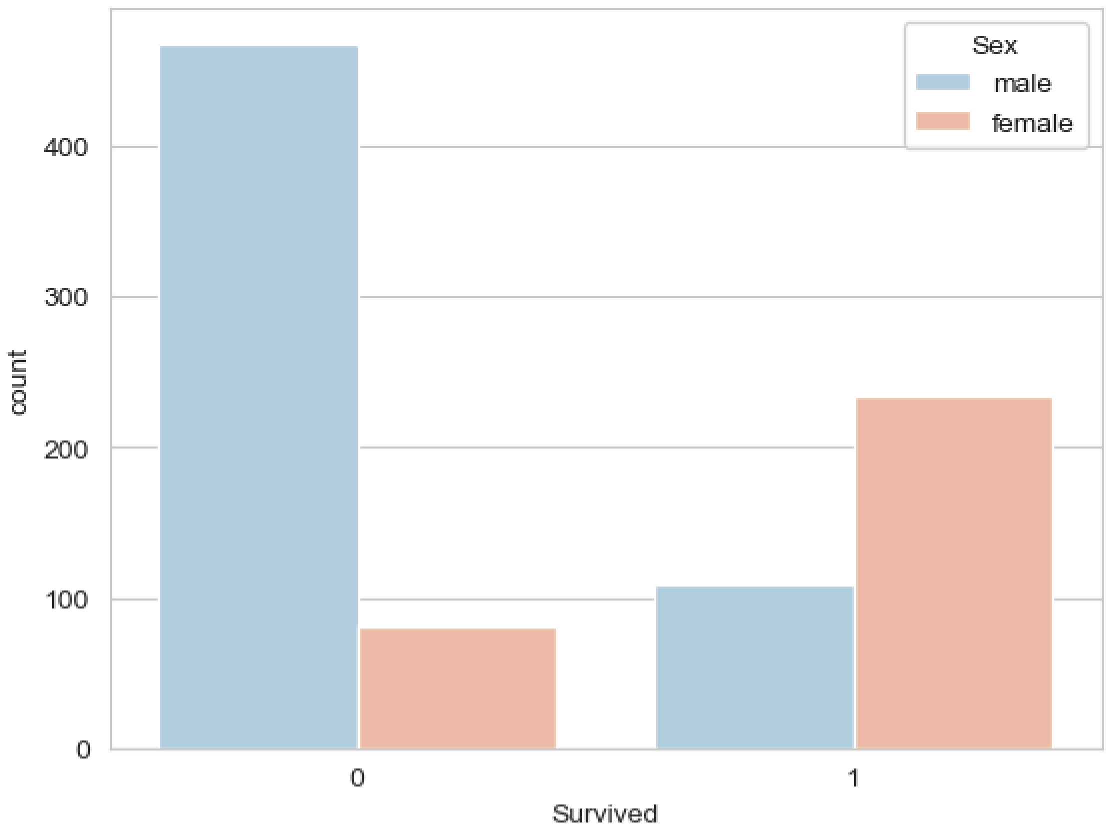
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train)
```

Out[6]: <Axes: xlabel='Survived', ylabel='count'>



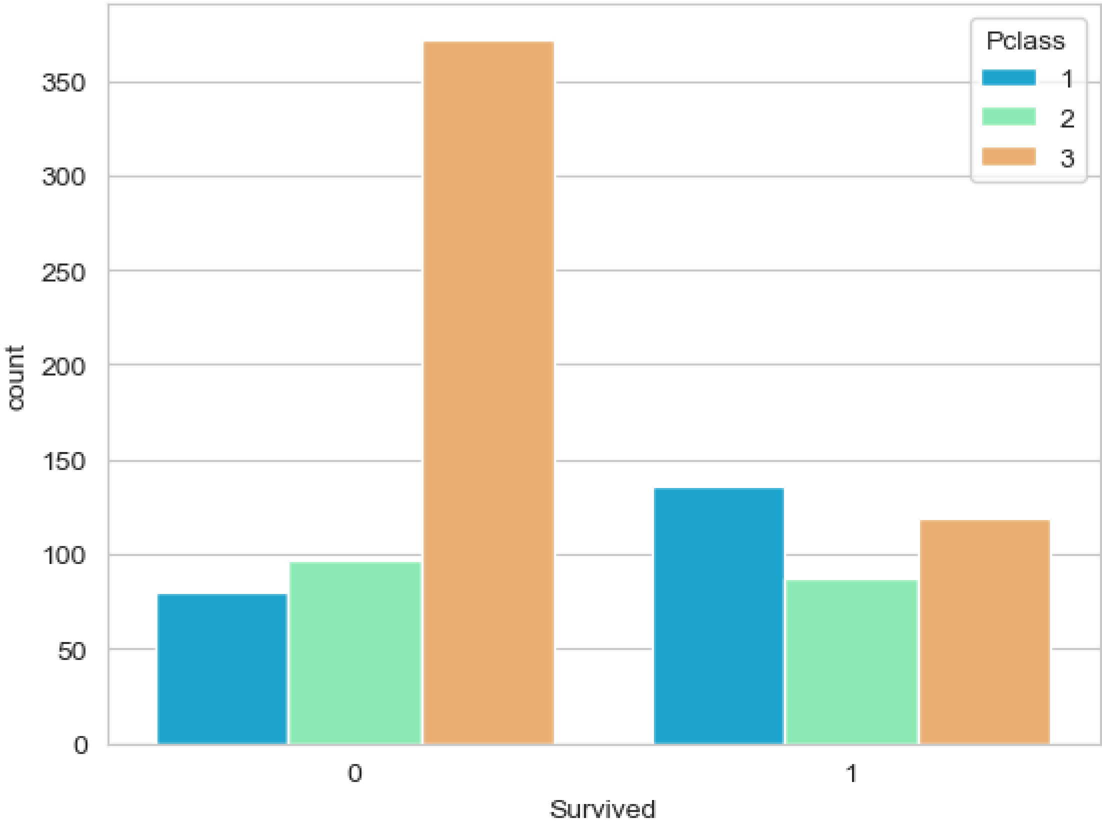
In [7]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

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



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

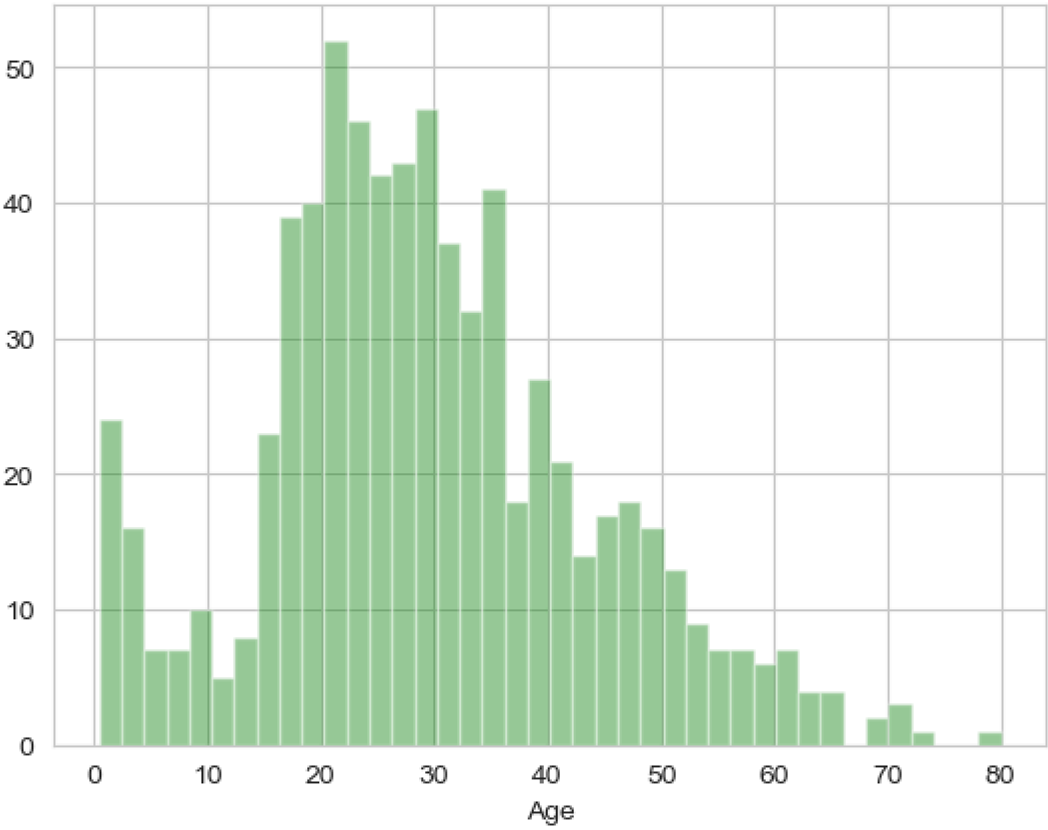
Out[8]: <Axes: xlabel='Survived', ylabel='count'>



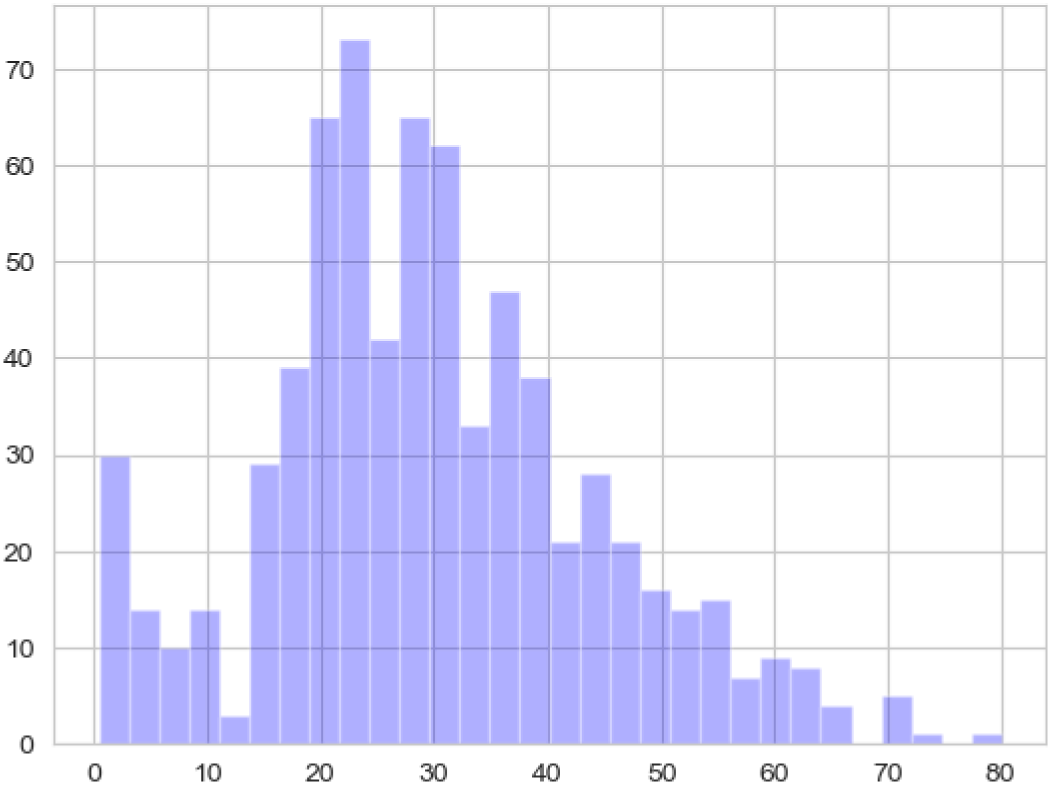
In [10]:
```
sns.distplot(train['Age'].dropna(),kde=False,color='green',bins=40)
```

```
/var/folders/2k/46dv5zj97cb0nhjw799hgpc40000gn/T/ipykernel_53741/3673825748.py:1: Use
rWarning:

`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(train['Age'].dropna(),kde=False,color='green',bins=40)
```

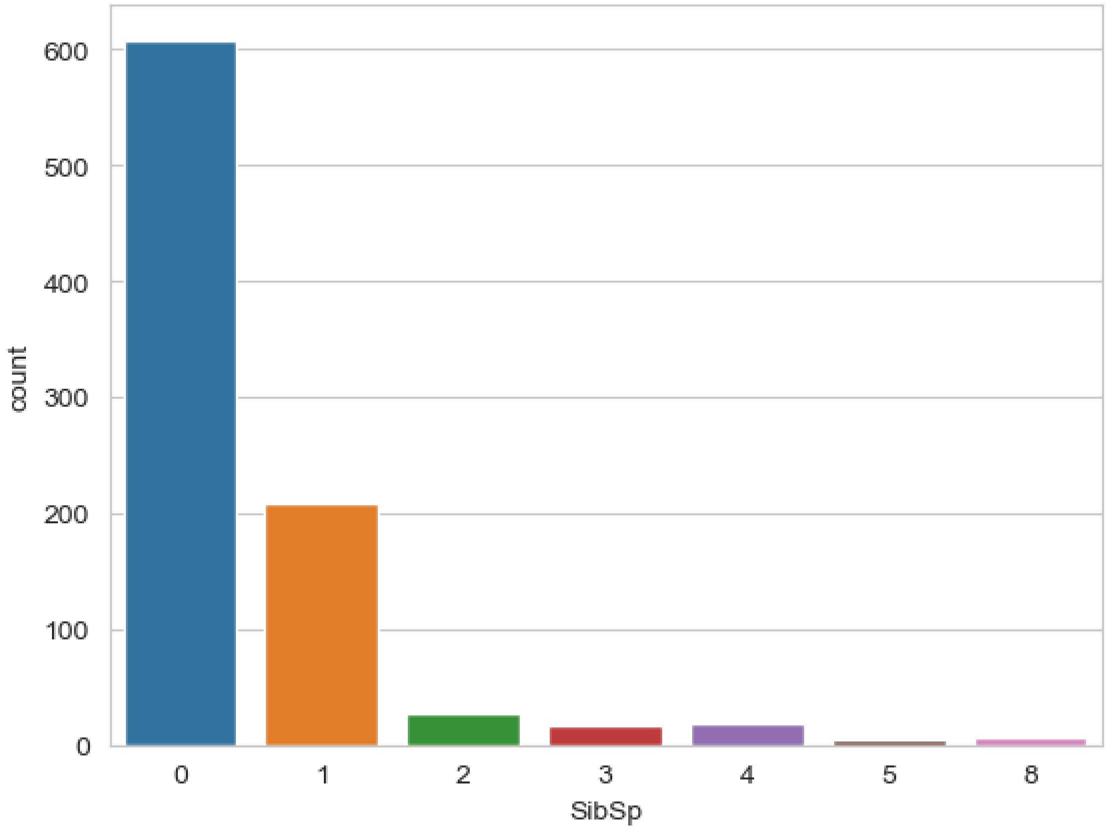
Out[10]: <Axes: xlabel='Age'>



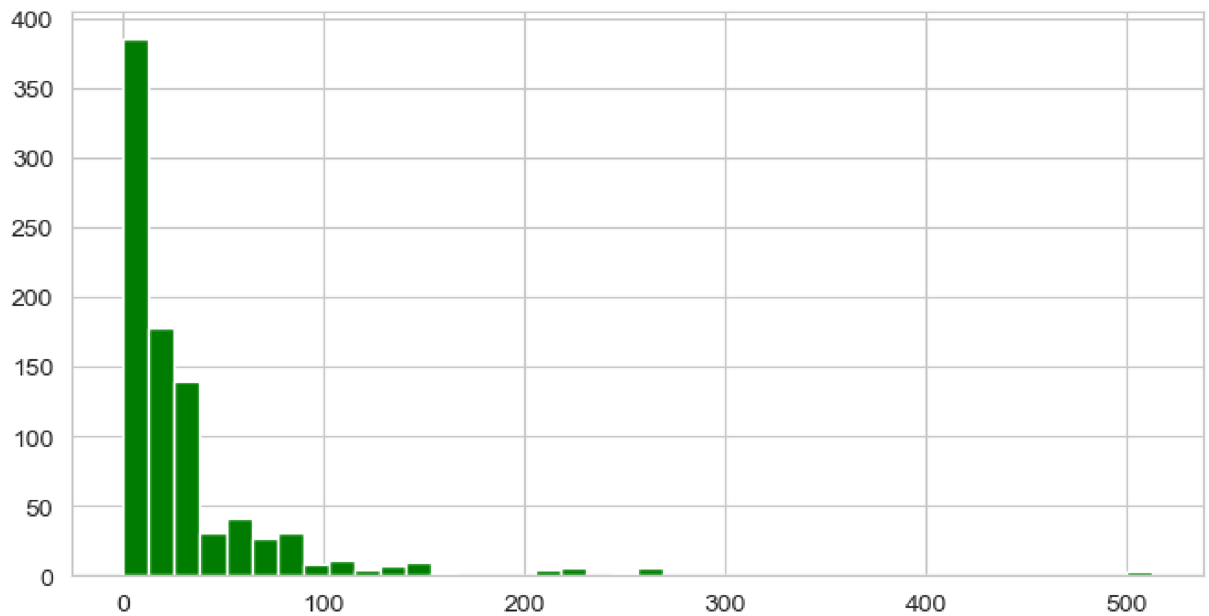
In [12]:
```
train['Age'].hist(bins=30,color='blue',alpha=0.3)
```
Out[12]: <Axes: >



In [13]:
```
sns.countplot(x='SibSp',data=train)
```
Out[13]: <Axes: xlabel='SibSp', ylabel='count'>



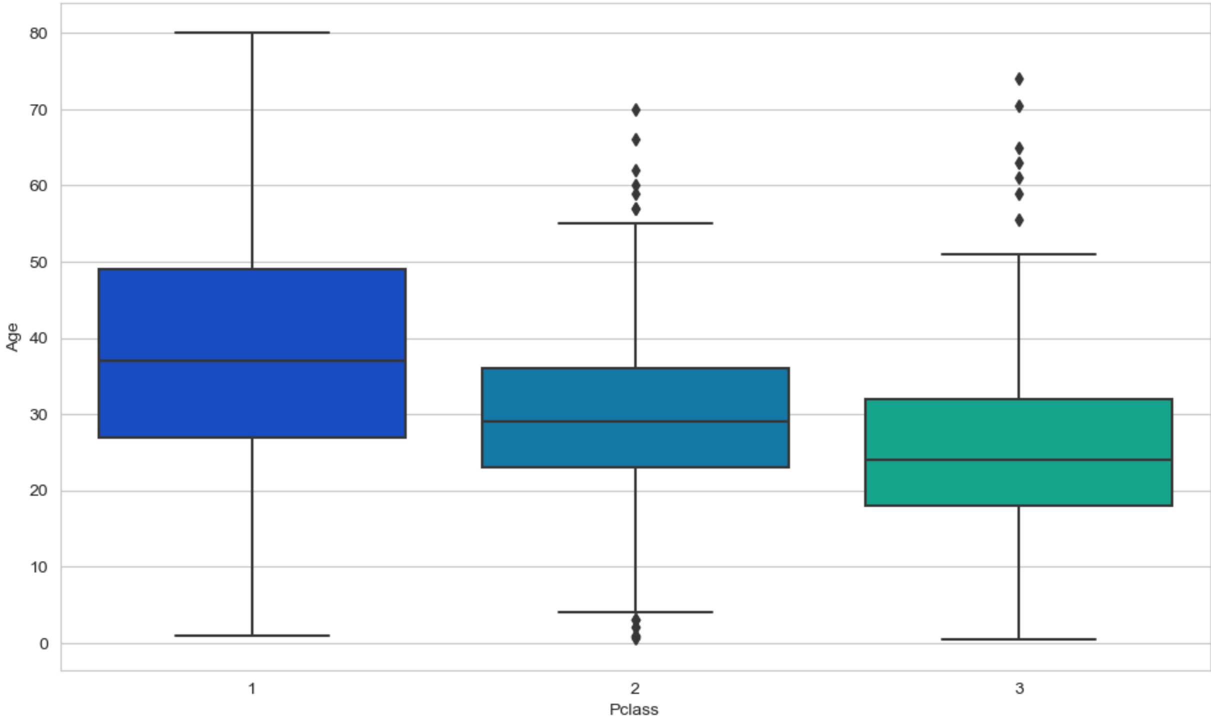
In [14]:
```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```
Out[14]: <Axes: >



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 [15]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```
Out[15]: <Axes: xlabel='Pclass', ylabel='Age'>

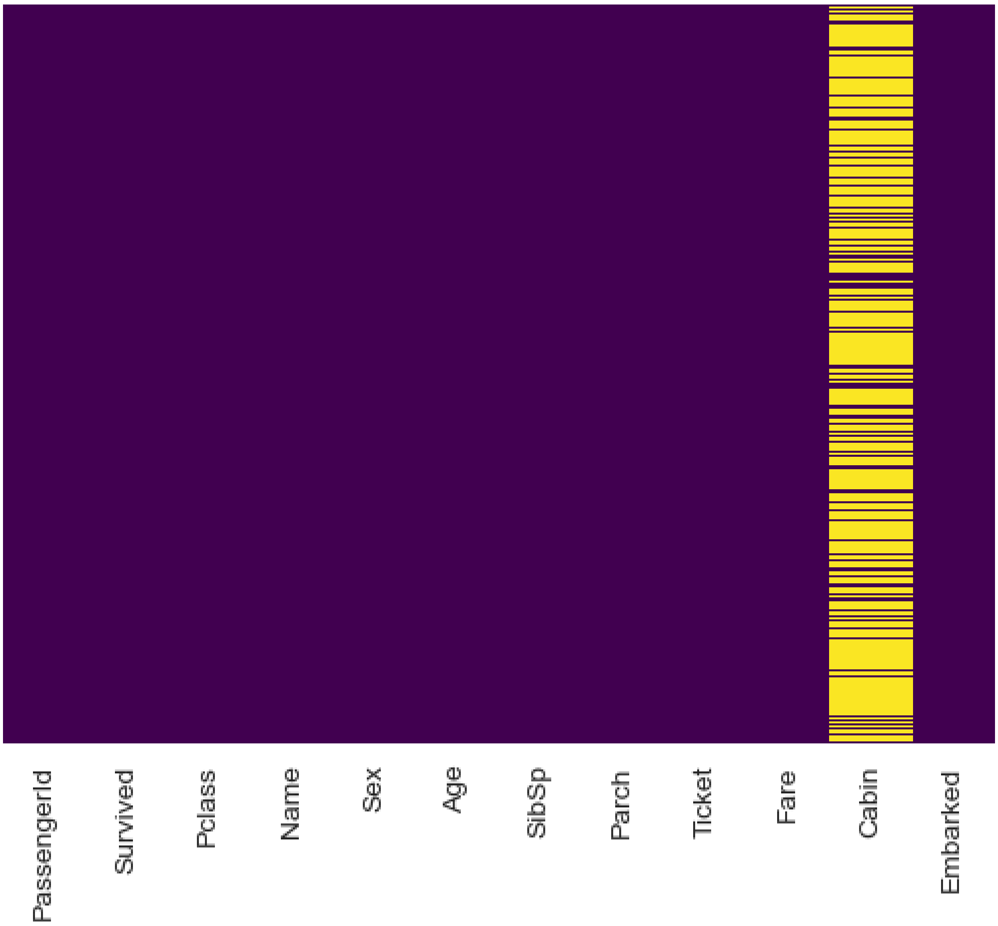


In [16]:
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

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

In [18]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[18]: <Axes: >



In [19]:
```
train.drop('Cabin',axis=1,inplace=True)
```

In [20]:
```
train.head()
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarke
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	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarke
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

In [21]:
```
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 [22]:
```
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 [23]:
```
pd.get_dummies(train['Embarked'],drop_first=True).head()
```

Out[23]:

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

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.head()
```

Out[26]:

	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

In [27]:

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

In [28]:

```
train.head()
```

Out[28]:

	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

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 [29]:

```
train.drop('Survived',axis=1).head()
```

Out[29]:

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

In [30]:

```
train['Survived'].head()
```

Out[30]:

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [31]:

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

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.3
                                                    random_state=101)
```

Training and Predicting

In [33]:
```
from sklearn.linear_model import LogisticRegression
```

In [34]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/Users/sandipkumarsahoo/anaconda3/lib/python3.11/site-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(
```

Out[34]:
```
▾ LogisticRegression
LogisticRegression()
```

In [35]:
```
predictions = logmodel.predict(X_test)
```

In [36]:
```
from sklearn.metrics import confusion_matrix
```

In [37]:
```
accuracy=confusion_matrix(y_test,predictions)
```

In [38]:
```
accuracy
```

Out[38]:
```
array([[148,  15],
       [ 39,  65]])
```

In [39]:
```
from sklearn.metrics import accuracy_score
```

In [40]:
```
accuracy=accuracy_score(y_test,predictions)
accuracy
```

Out[40]:
```
0.797752808988764
```

In [41]:
```
predictions
```

Out[41]:
```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 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, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
```

```
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1])
```

Evaluation

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

In [42]:

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

In [43]:

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

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       163
           1       0.81      0.62      0.71       104

    accuracy                           0.80       267
   macro avg       0.80      0.77      0.78       267
weighted avg       0.80      0.80      0.79       267
```

In [ ]: