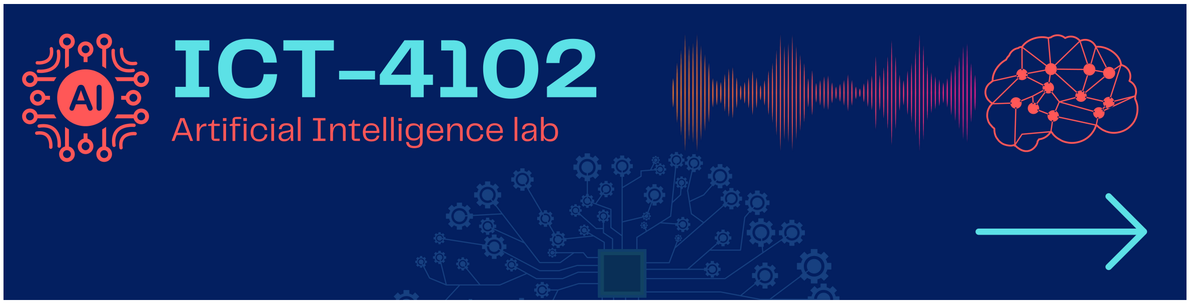


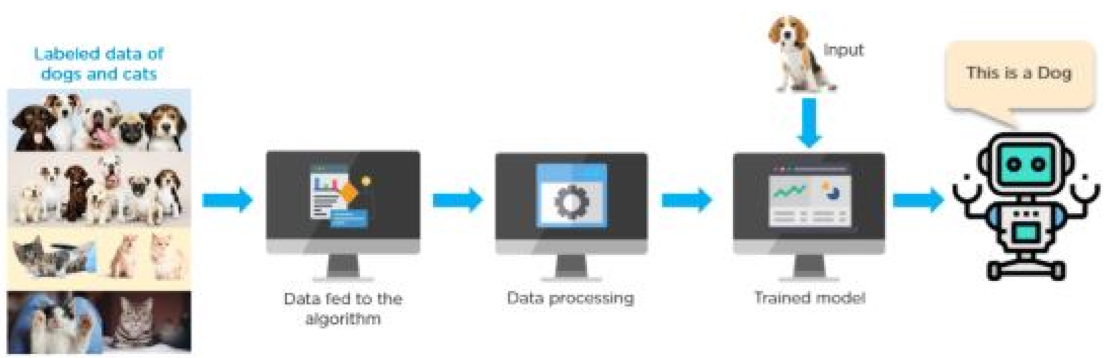
()

Supervised learning problems can be further classified into regression and classification problems.

*Classification: In a classification problem, the output variable is a category, such as “red” or “blue,” “disease” or “no disease,” “true” or “false,” etc.

*Regression: In a regression problem, the output variable is a real continuous value, such as “dollars” or “weight.”

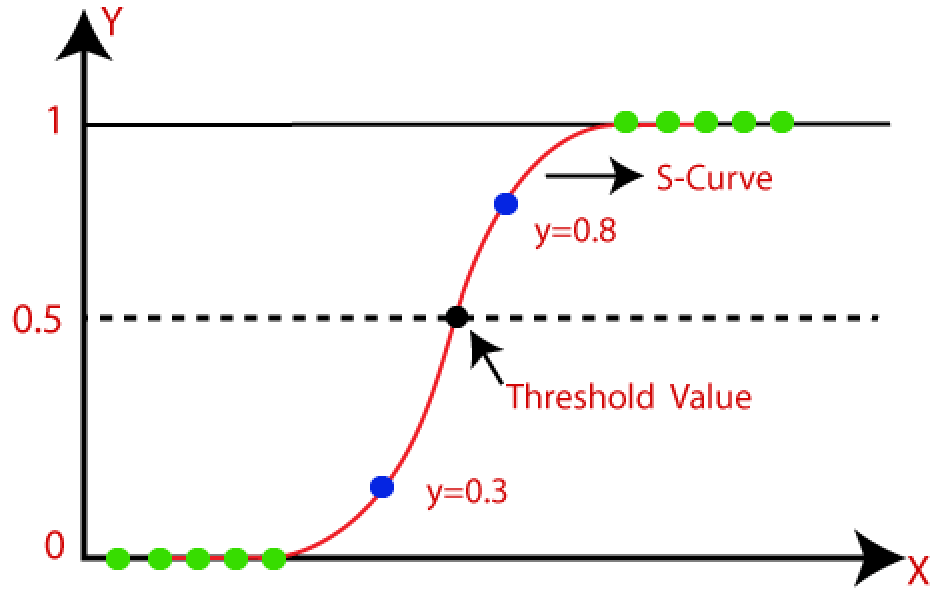
The following is an example of a supervised learning method where we have labeled data to identify dogs and cats. The algorithm learns from this data and trains a model to predict the new input.



Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Logistic Regression with Python

For this lab we will be working with the [Titanic Data Set from Kaggle (https://www.kaggle.com/c/titanic)](https://www.kaggle.com/c/titanic). This is a very famous data set and very often is a student's first step in machine learning!

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.

Import Libraries

Let's import some libraries to get started!

In [106]:
```
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.

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

In [107]:
```
train = pd.read_csv('titanic_train.csv')
```

In [108]:
```
train.head()
```

Out[108]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	I
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	I
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	I

Exploratory Data Analysis

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

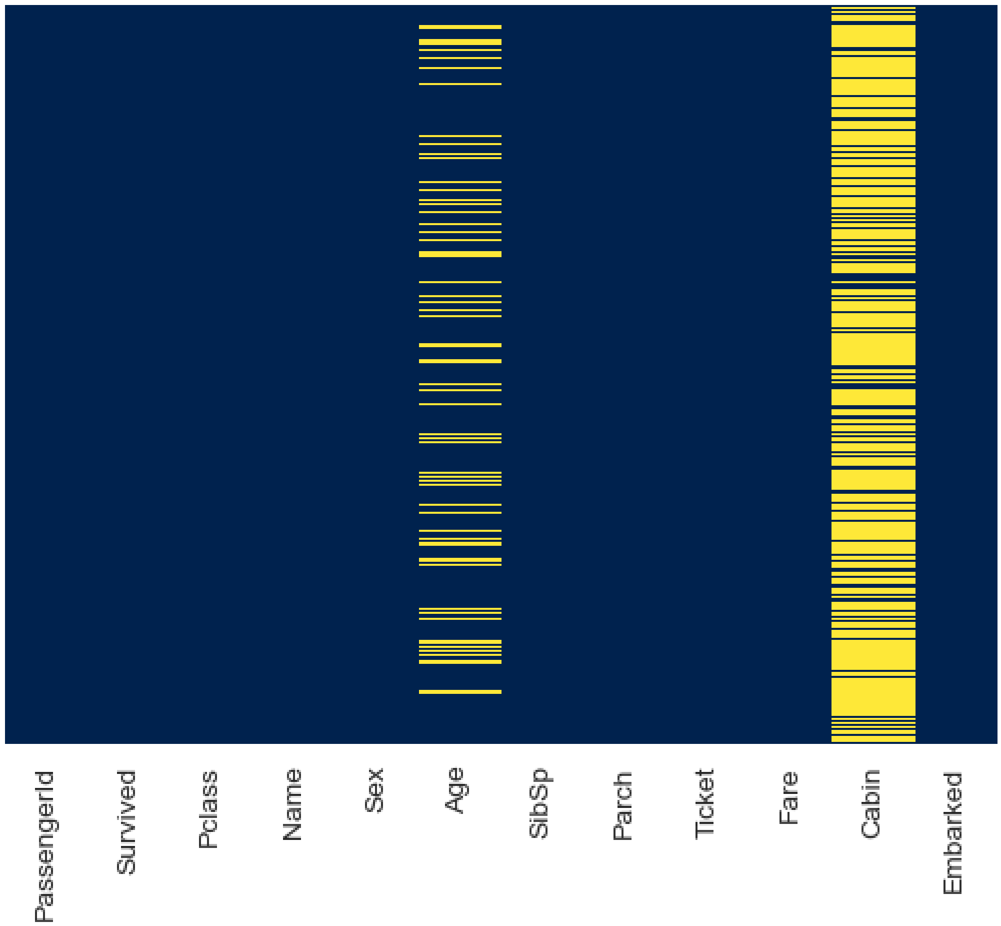
Missing Data

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

- cbar= False means Whether to draw a colorbar

In [109]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='cividis')
```

Out[109]: <Axes: >

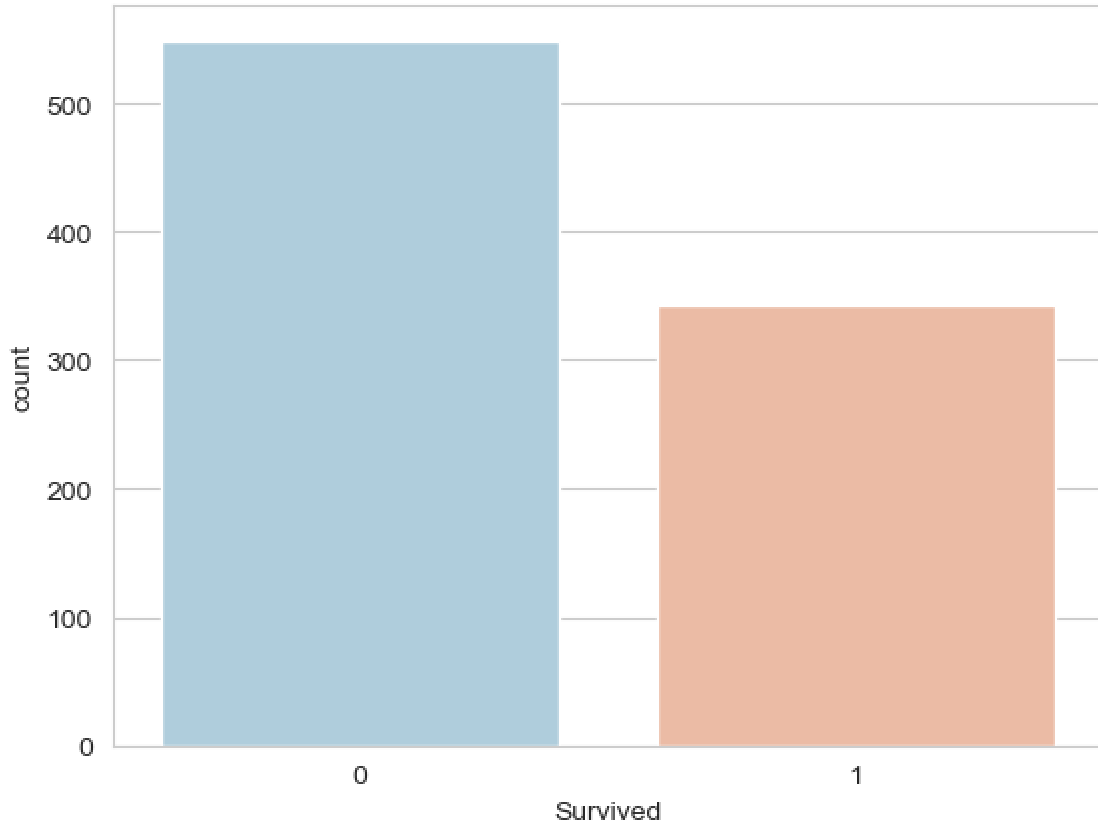


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably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

Let's continue on by visualizing some more of the data!

In [110]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```

Out[110]: <Axes: xlabel='Survived', ylabel='count'>

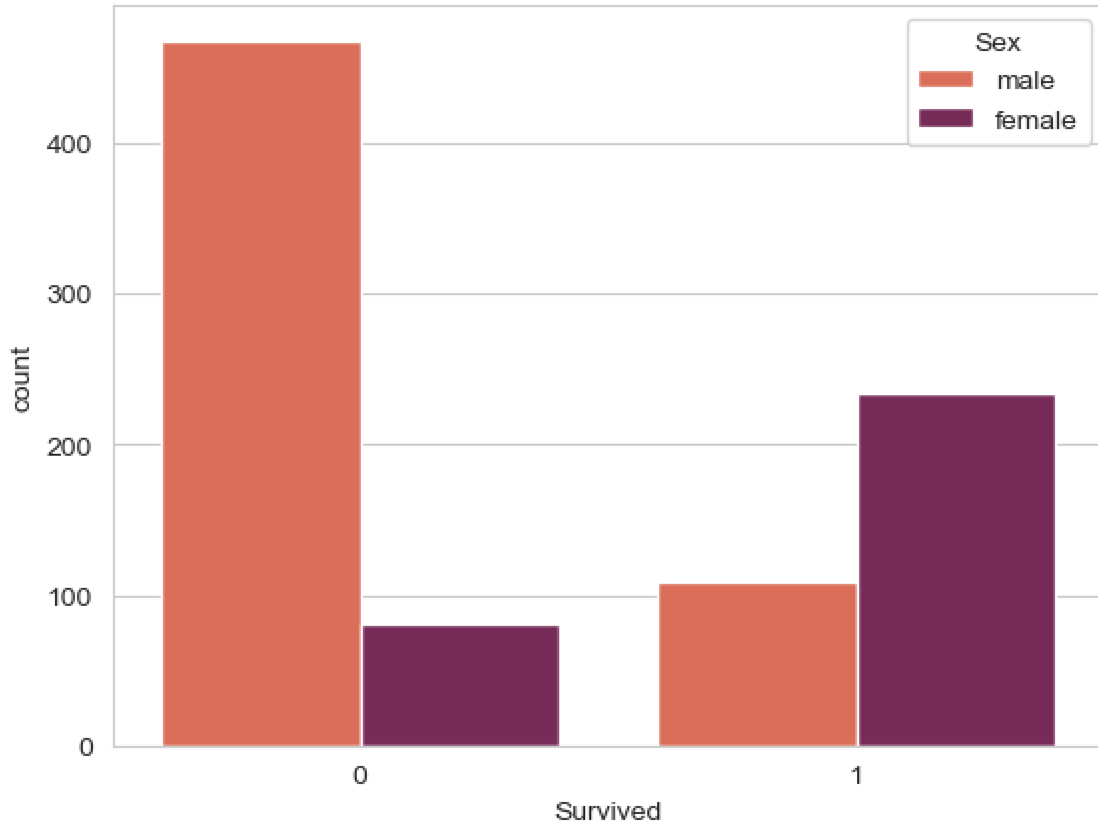


Grouped Count Plot in Seaborn : hue= parameter

By creating a grouped count plot, you can add an additional dimension of data into the visualization. This allows you to compare one category within another category. To do this in Seaborn, you can use the hue= parameter. The parameter accepts a string column label, adding a split for each subcategory in the dataset.

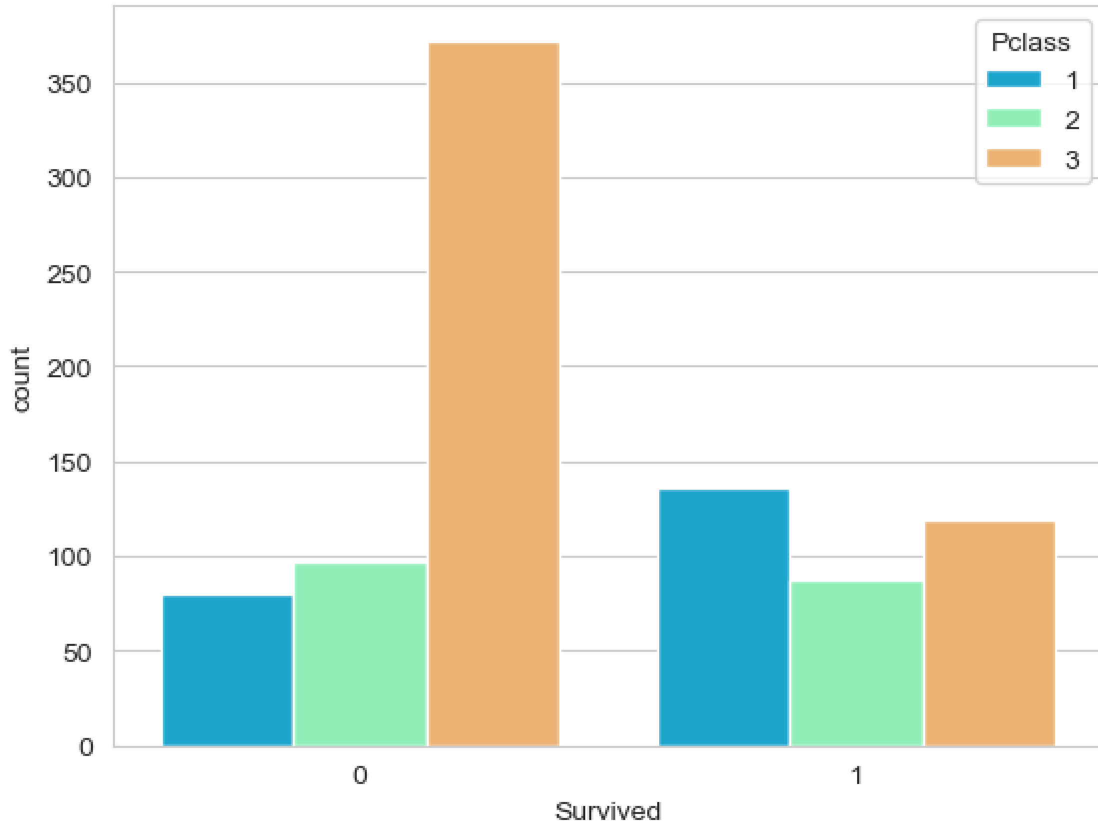
In [111]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='rocket_r')
```

Out[111]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



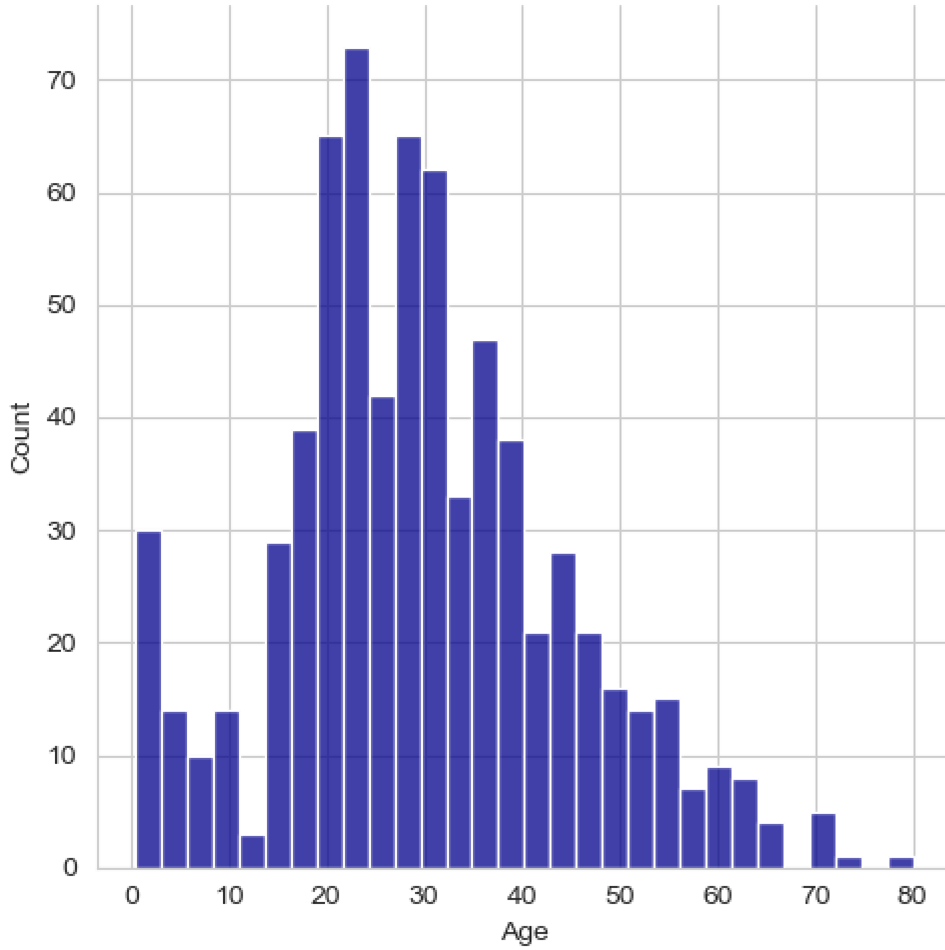
In [112]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

Out[112]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



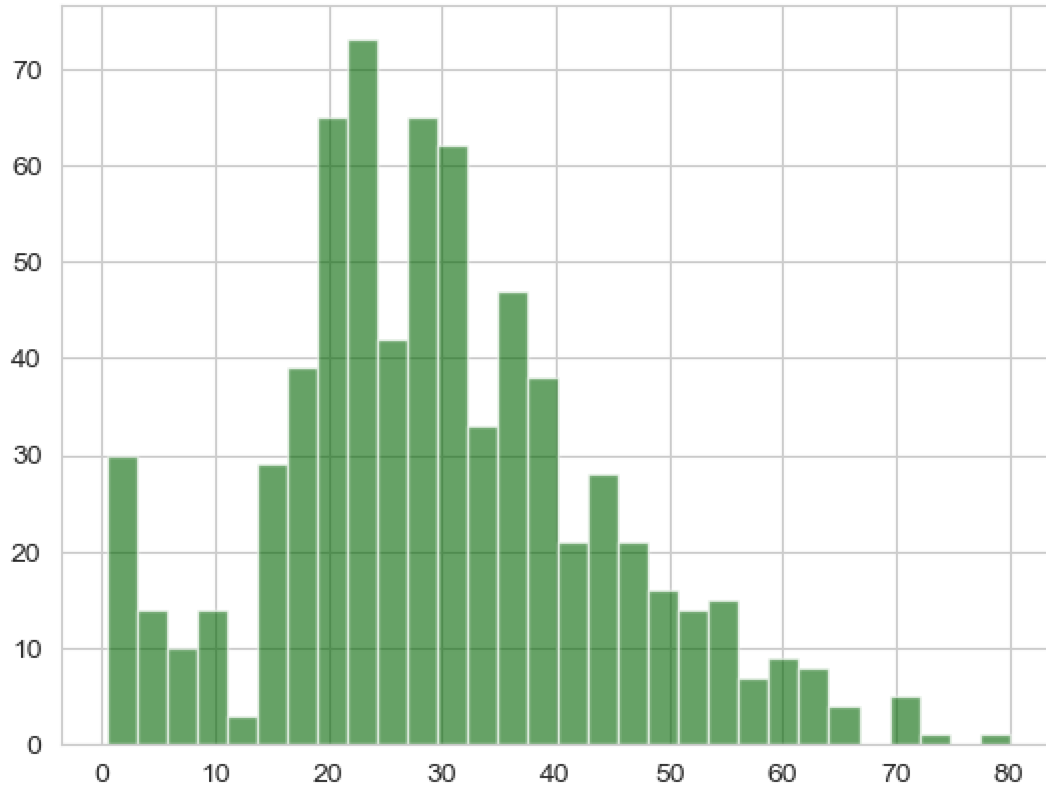
In [113]:
```
sns.displot(train['Age'].dropna(),kde=False,color='darkblue',bins=30)
```

Out[113]: <seaborn.axisgrid.FacetGrid at 0x13655332650>



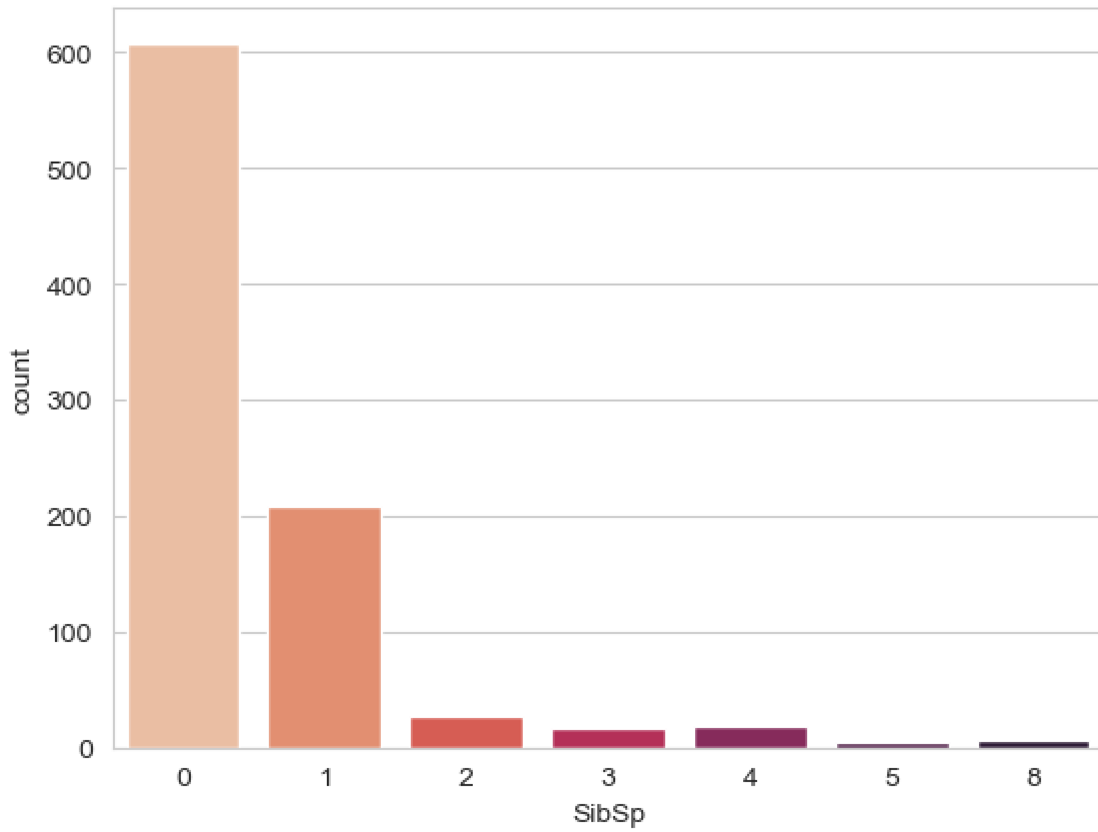
In [114]:
```
train['Age'].hist(bins=30,color='darkgreen',alpha=0.6)
```

Out[114]: <Axes: >



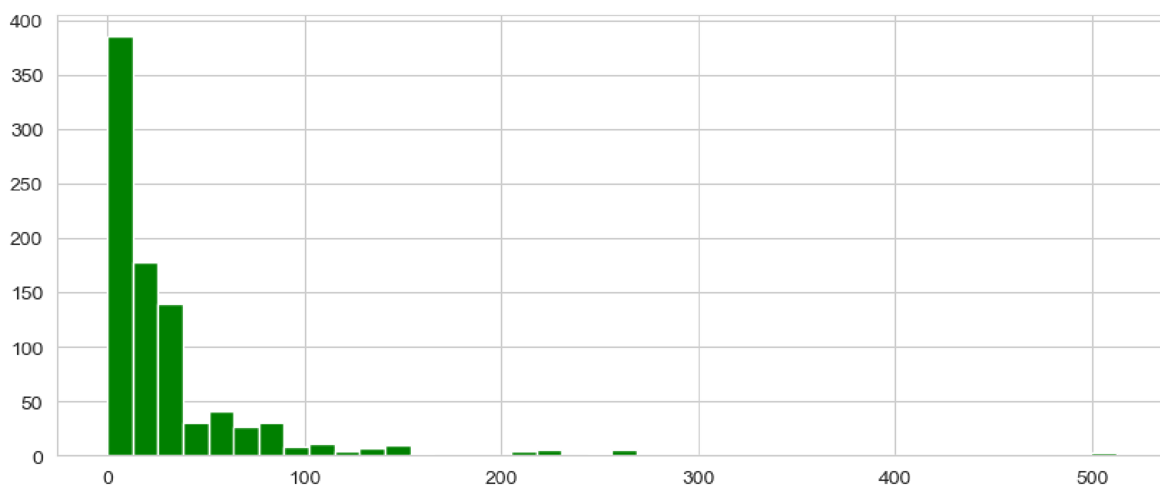
In [115]:
```
sns.countplot(x='SibSp',data=train,palette='rocket_r')
```

Out[115]: <Axes: xlabel='SibSp', ylabel='count'>



In [116]:
```
train['Fare'].hist(color='green',bins=40,figsize=(10,4))
```

Out[116]: <Axes: >



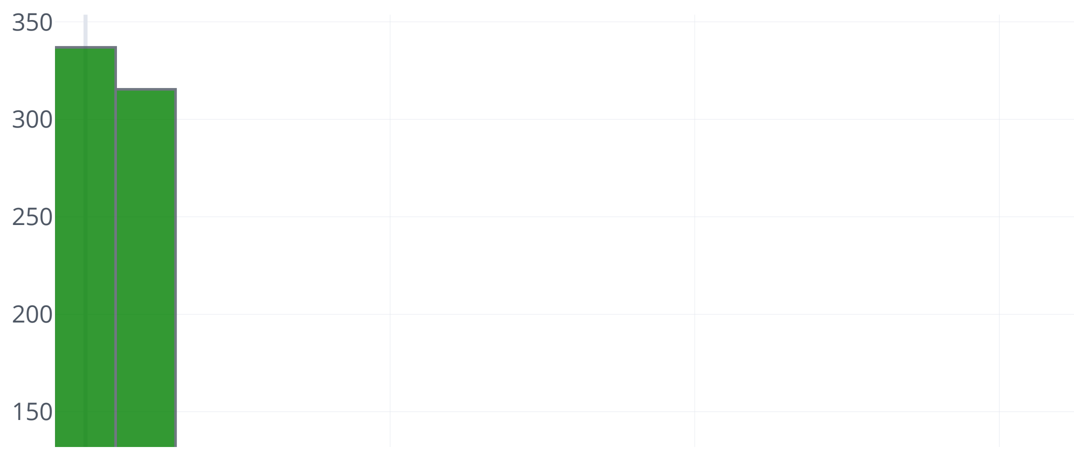
Cufflinks for plots

Let's take a quick moment to show an example of cufflinks!

- conda install -c conda-forge cufflinks-py

In [117]:
```
import cufflinks as cf
cf.go_offline()
```

In [118]:
```
train['Fare'].iplot(kind='hist',bins=30,color='green')
```

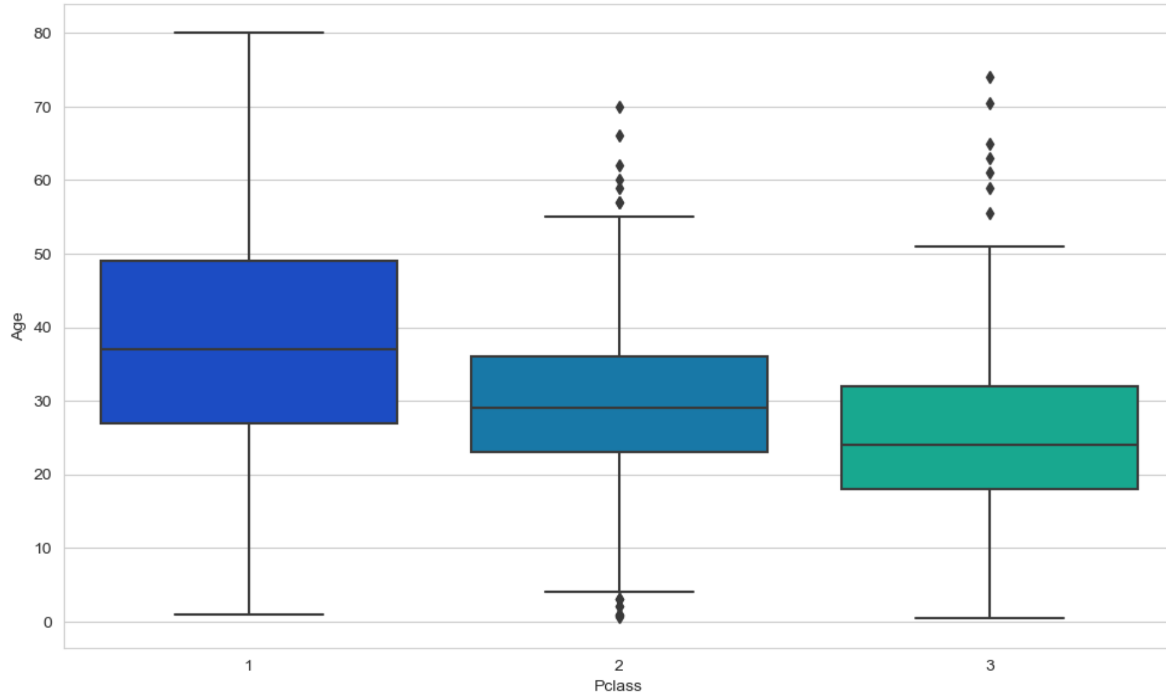


Data Cleaning

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 [119]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

Out[119]: <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 [120]:
```
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
```

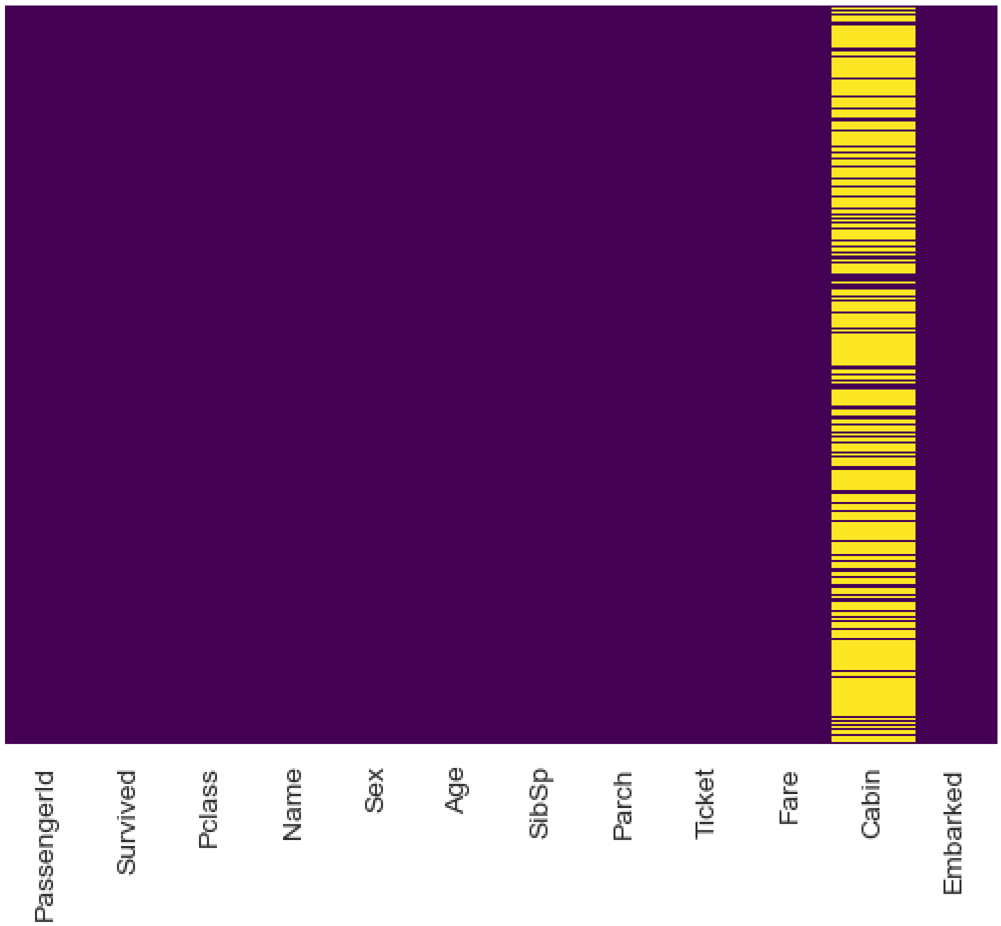
Now apply that function!

In [121]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

In [122]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[122]:
```
<Axes: >
```



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

In [123]:
```
train.drop('Cabin',axis=1,inplace=True)
```

In [124]:
```
train.head()
```

Out[124]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Er
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 [125]:
```
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 [126]:
```
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 [127]:
```
sex = pd.get_dummies(train['Sex'])
sex
```

Out[127]:

	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

889 rows × 2 columns

In [128]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
sex
```

Out[128]:

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

889 rows × 1 columns

In [129]:
```
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

In [130]:
```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [131]:
```
train = pd.concat([train,sex,embark],axis=1)
```

In [132]:
```
train.head()
```

Out[132]:

	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

In [133]:
```
from sklearn.model_selection import train_test_split
```

In [134]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis
                                                    train['Survived'], test_si
                                                    random_state=101)
```

Training and Predicting

In [135]:
```
from sklearn.linear_model import LogisticRegression
```

In [136]:
```
logmodel = LogisticRegression(solver='lbfgs', max_iter=1000)
logmodel.fit(X_train,y_train)
```

Out[136]:
```
▾        LogisticRegression
LogisticRegression(max_iter=1000)
```

In [137]:
```
predictions = logmodel.predict(X_test)
```

Let's move on to evaluate our model!

Evaluation

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

In [138]:
```
from sklearn.metrics import classification_report
```

In [139]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       163
           1       0.84      0.68      0.75       104

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

In [140]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
# Calculate the accuracy
accuracy = accuracy_score(y_test, predictions)

# Calculate the precision
precision = precision_score(y_test, predictions)

# Calculate the recall
recall = recall_score(y_test, predictions)

# Calculate the f1 score
f1 = f1_score(y_test, predictions)

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

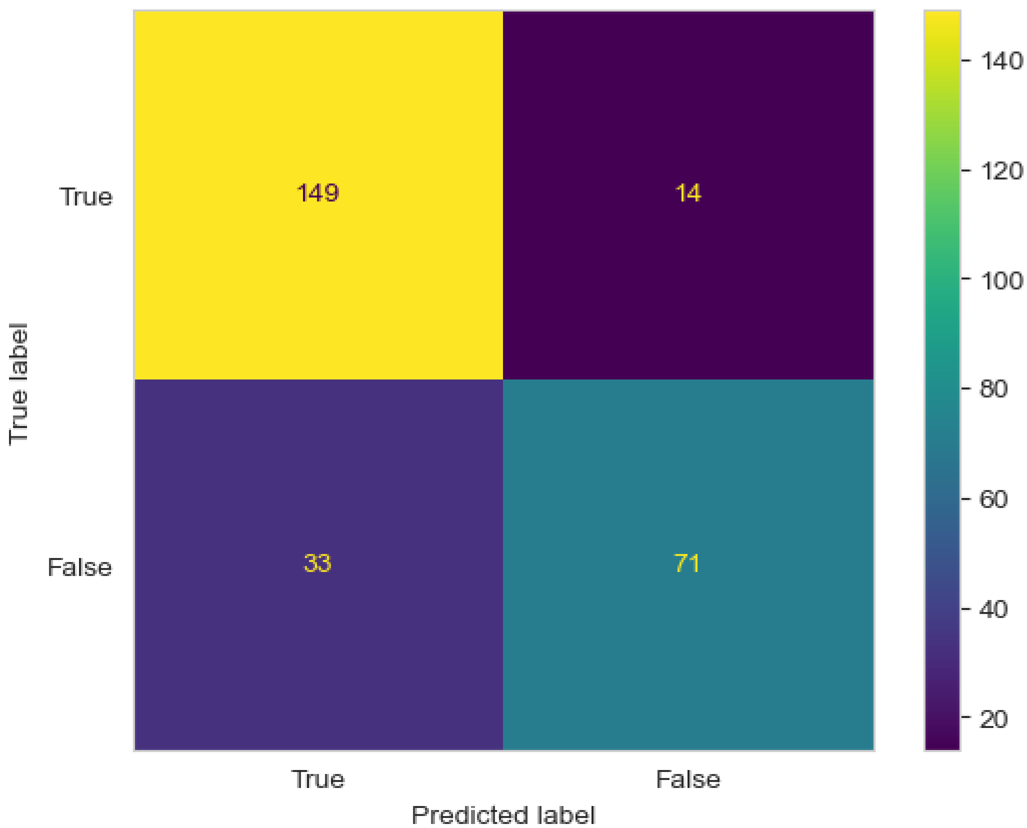
```
Accuracy: 0.8239700374531835
Precision: 0.8352941176470589
Recall: 0.6826923076923077
F1 Score: 0.7513227513227515
```

In [141]:
```
from sklearn.metrics import confusion_matrix
```

In [142]:
```
confusion_matrix(y_test,predictions)
```

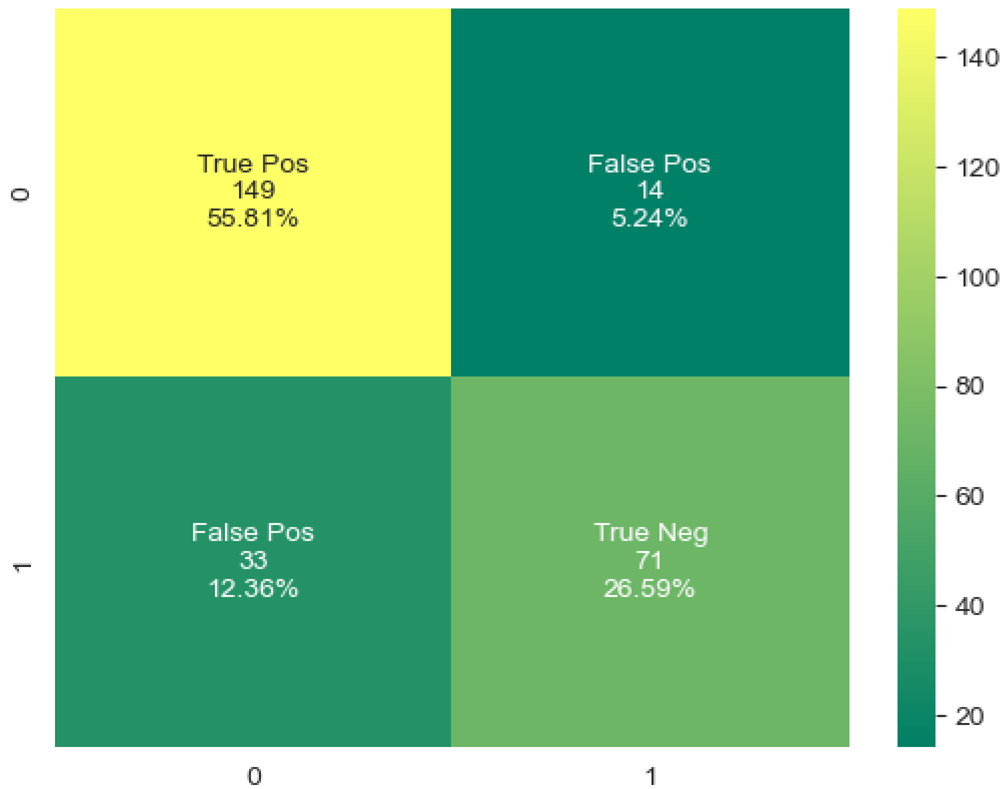
Out[142]:
```
array([[149,  14],
       [ 33,  71]], dtype=int64)
```

In [143]:
```
# Plotting a Confusion Matrix with Sklearn
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

conf_matrix = confusion_matrix(y_test, predictions)
vis = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels = [T

vis.plot()
plt.grid(False)
plt.show()
```



In [145]:
```
import seaborn as sns
%matplotlib inline
group_names = ['True Pos','False Pos','False Pos','True Neg']
group_counts = ["{0:0.0f}".format(value) for value in conf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in conf_matrix.flatten(
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(conf_matrix, annot=labels, fmt='', cmap='summer')
```

Out[145]: <Axes: >



In [ ]:

In [ ]: