Logistic Regression with Python

For this lecture we will be working with the Titanic Data Set from Kaggle. 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 [1]:
```
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.

In [2]:
```
train = pd.read_csv('13 Logistic Regression (Titanic) Train.csv')
```

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

Out[3]:

	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

Exploratory Data Analysis

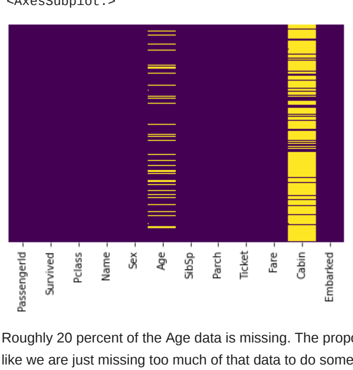
Let's begin some exploratory data analysis! We'll start by checking out missing data!

Missing Data

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

In [4]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis') # yticklabels = y values, cbar = sidebar
```

Out[4]: <AxesSubplot:>

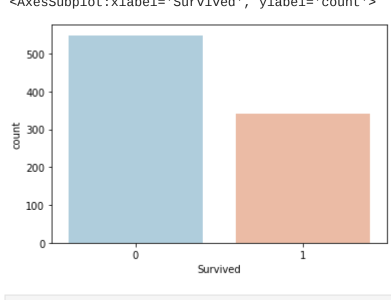


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably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

Let's continue on by visualizing

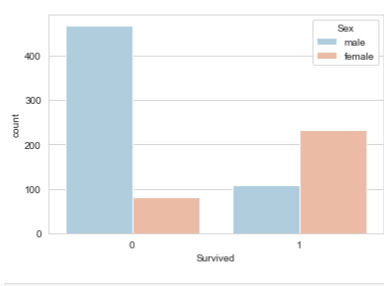
In [5]:
```
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```

Out[5]: <AxesSubplot:xlabel='Survived', ylabel='count'>



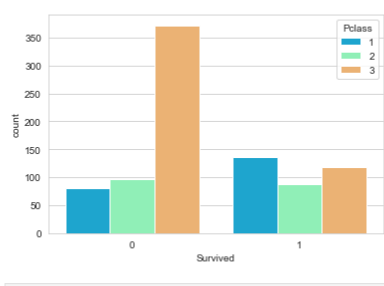
In [6]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

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



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

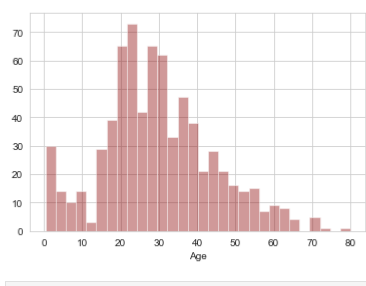
Out[7]: <AxesSubplot:xlabel='Survived', ylabel='count'>



In [8]:
```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
```

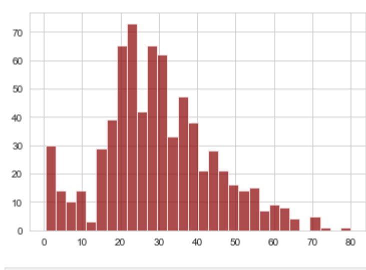
```
C:\Users\Yasin\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]: <AxesSubplot:xlabel='Age'>



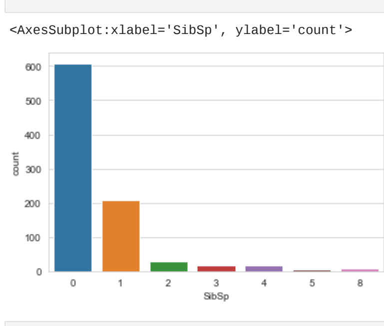
In [9]:
```
train['Age'].hist(bins=30,color='darkred',alpha=0.7)
```

Out[9]: <AxesSubplot:>



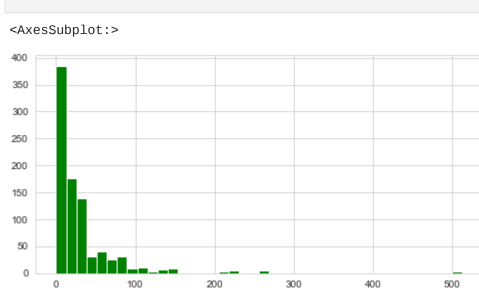
In [10]:
```
sns.countplot(x='SibSp',data=train)
```

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



In [11]:
```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

Out[11]: <AxesSubplot:>



Cufflinks for plots

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

In [12]:
```
import cufflinks as cf
cf.go_offline()
```

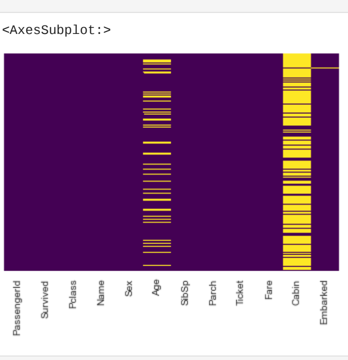
In [13]:
```
train['Fare'].iplot(kind='hist',bins=30,color='green')
```

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:

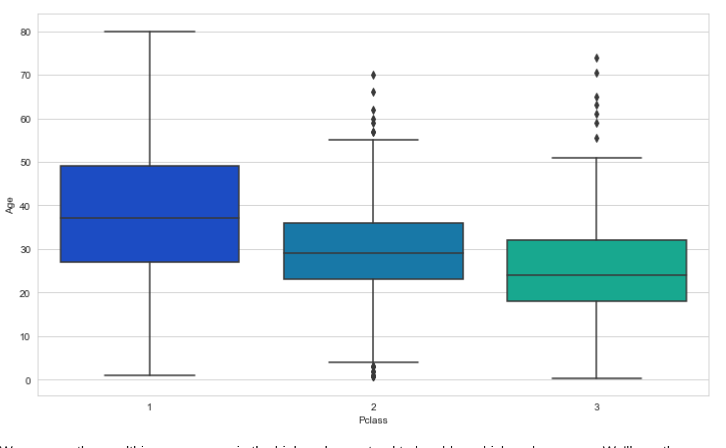
In [14]:
```
sns.heatmap(data=train.isnull(), yticklabels=False, cbar=False,cmap='viridis')
```

Out[14]: <AxesSubplot:>



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 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 [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
```

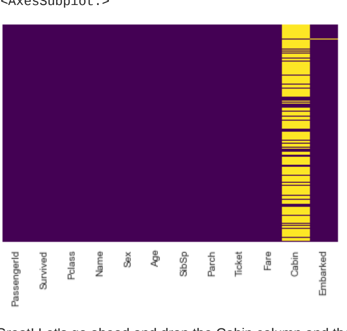
Now apply that function!

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

Now let's check that heat map again!

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

Out[18]: <AxesSubplot:>



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

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

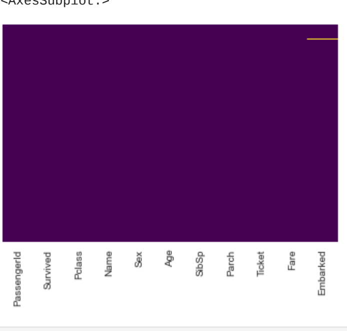
In [20]:
```
train.head()
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	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 [21]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[21]: <AxesSubplot:>



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

Out[24]:

	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 [25]:
```
pd.get_dummies(train['Sex'], drop_first=True)
```

Out[25]:

	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 [26]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
```

In [27]:
```
pd.get_dummies(train['Embarked'])
```

Out[27]:

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

889 rows × 3 columns

In [28]:
```
pd.get_dummies(train['Embarked'],drop_first=True)
```

Out[28]:

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1
...	...	...
886	0	1
887	0	1
888	0	1
889	0	0
890	1	0

889 rows × 2 columns

In [29]:
```
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

In [30]:
```
train = pd.concat([train,sex,embark],axis=1)
train.head(1)
```

Out[30]:

	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	7.25	S	1	0	1

In [31]:
```
train.drop(['Sex','Embarked','Name','Ticket','PassengerId'],axis=1,inplace=True)
```

In [32]:
```
train.head()
```

Out[32]:

	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

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 [33]:
```
from sklearn.model_selection import train_test_split
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

In [35]:
```
from sklearn.linear_model import LogisticRegression
```

In [36]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
C:\Users\Yasin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: 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[36]: LogisticRegression()

In [37]:
```
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 [38]:
```
from sklearn.metrics import classification_report
```

In [39]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       163
           1       0.82      0.71      0.76       104

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

In [40]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, predictions))
```

```
[[147  16]
 [ 30  74]]
```

In [ ]: