Logistic Regression with Python

For this lecture we will be working with the Titanic Data Set from Kaggle (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 [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('titanic_train.csv')
```

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

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

In [5]:
```
train.describe()
```

Out[5]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.2
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.6
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.00
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.91
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.4
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.0
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.

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!

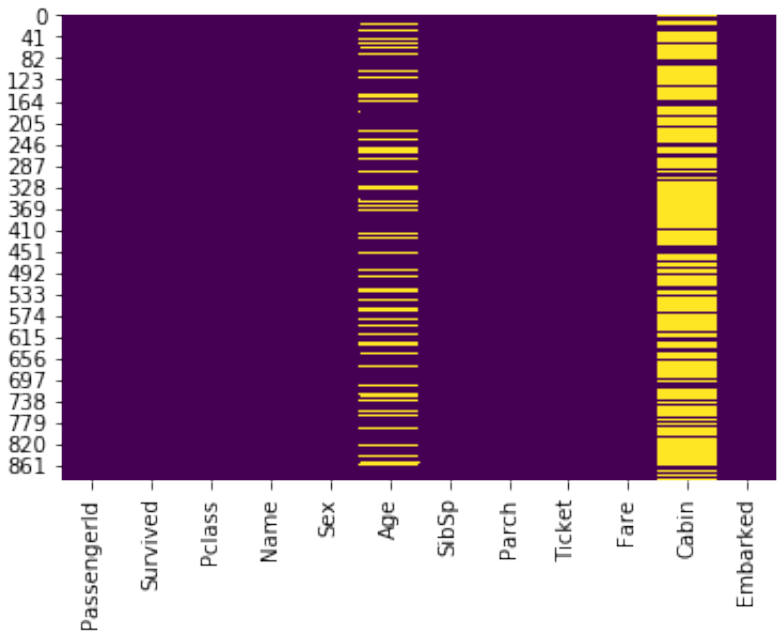
In [4]:
```
train.isnull().head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	False	False	False	False	False	False	False	False	False	False	True
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True

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

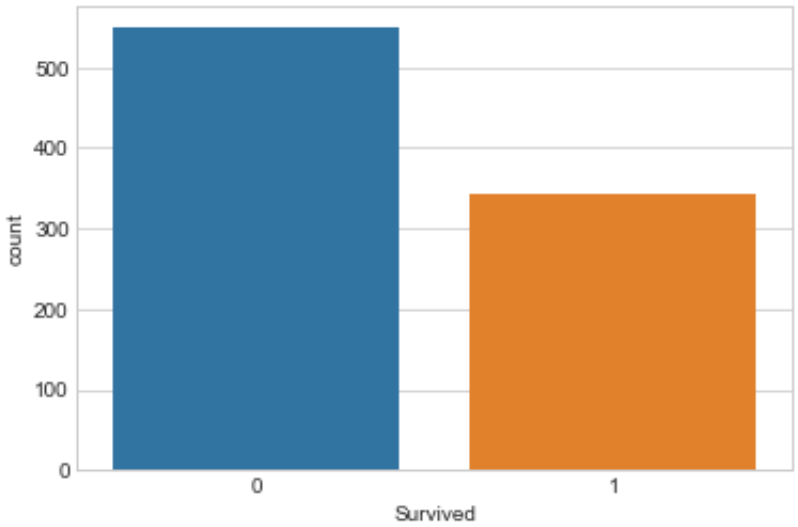
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x10ea9f518>



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"

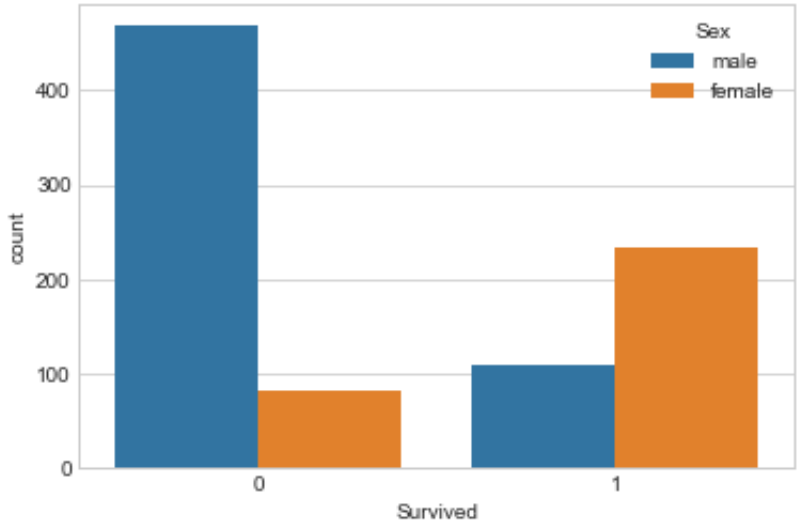
In [6]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x10eba0128>



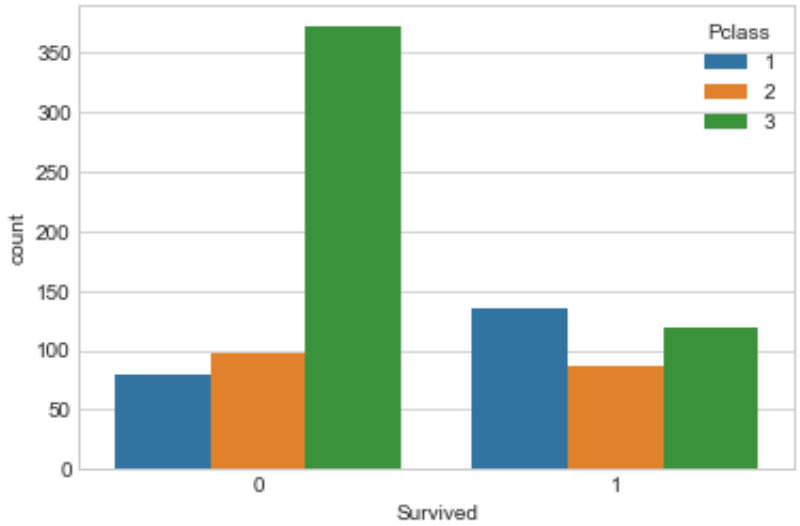
In [7]:
```
#let us see survival by gender
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x10eb506a0>



In [8]:
```
#survival by passenger class
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x10eb67908>

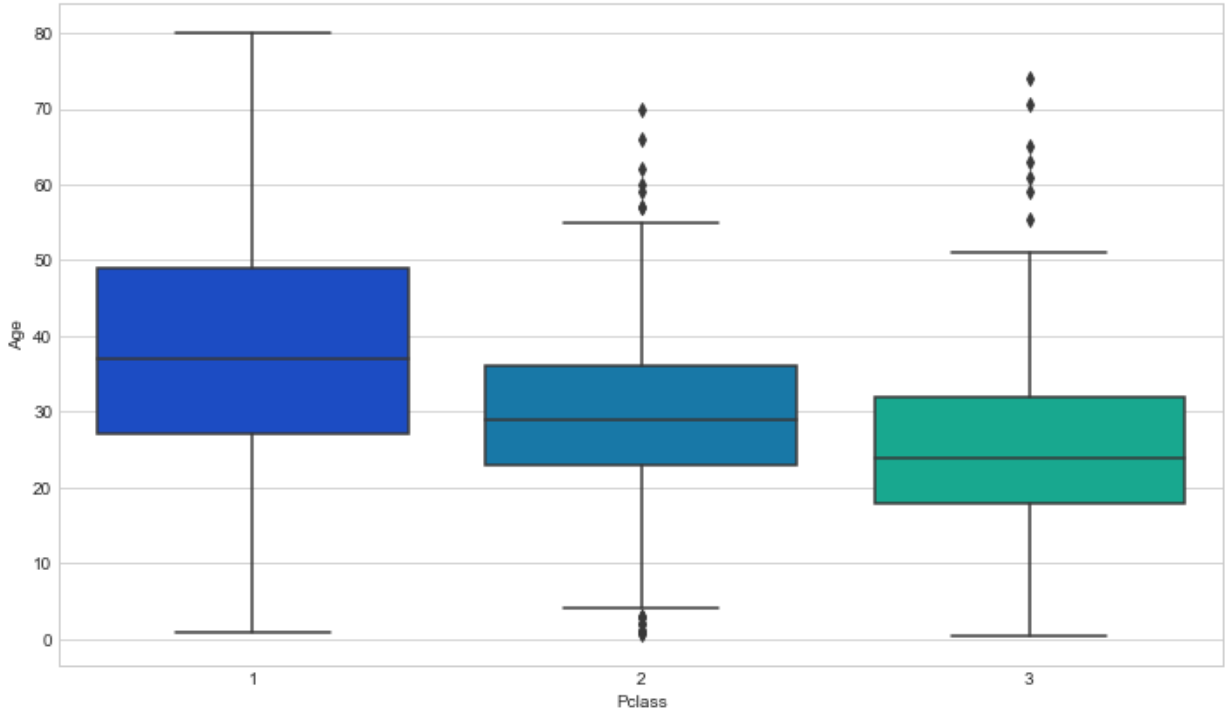


Data Cleaning

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 [9]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e62f5c0>



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 [10]:
```
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
```

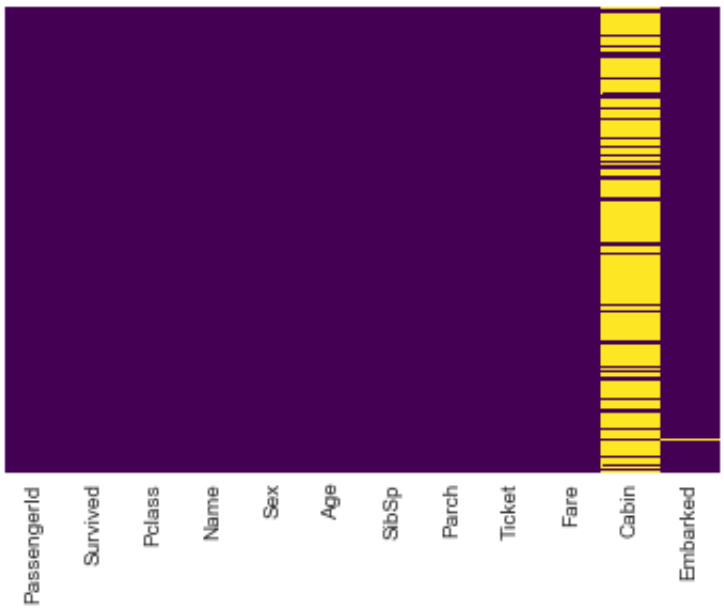
Now apply that function!

In [11]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

In [12]:
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis'
)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e6d4ac8>



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

In [13]:
```
train.drop('Cabin',axis=1,inplace=True)
```

In [14]:
```
train.head()
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0

In [15]:
```
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 [16]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    889 non-null int64
Survived       889 non-null int64
Pclass         889 non-null int64
Name           889 non-null object
Sex            889 non-null object
Age            889 non-null float64
SibSp          889 non-null int64
Parch          889 non-null int64
Ticket         889 non-null object
Fare           889 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [17]:
```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

In [18]:
```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [19]:
```
train = pd.concat([train,sex,embark],axis=1)
```

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

Out[20]:

	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 [43]:
```
from sklearn.model_selection import train_test_split
```

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Surviv
ed',axis=1),
                                                    train['Survived'],
test_size=0.30,
                                                    random_state=101)
```

Training and Predicting

In [45]:
```
from sklearn.linear_model import LogisticRegression
```

In [46]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Out[46]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jo
bs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0
.0001,
          verbose=0, warm_start=False)
```

In [47]:
```
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 [48]:
```
from sklearn.metrics import classification_report
```

In [49]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.81      0.93      0.86       163
          1       0.85      0.65      0.74       104

avg / total       0.82      0.82      0.81       267
```

Great Job!