# Logistic Regression on titanic survival problem

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.

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.

## Reading training data

# Reading training data  
train\_data = pd.read\_csv("C:/Users/G01212601/Downloads/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/13-Logistic-Regression/titanic\_train.csv")

## Heatmap to check null data

Sometimes you can have null data in your data set, which can be checked by isnull() method and through plot using heatmap

Wherever False is there, means there is undefined or null value present in the data set.

print(train\_data.isnull())

PassengerId Survived Pclass Name ... Ticket Fare Cabin Embarked

0 False False False False ... False False True False

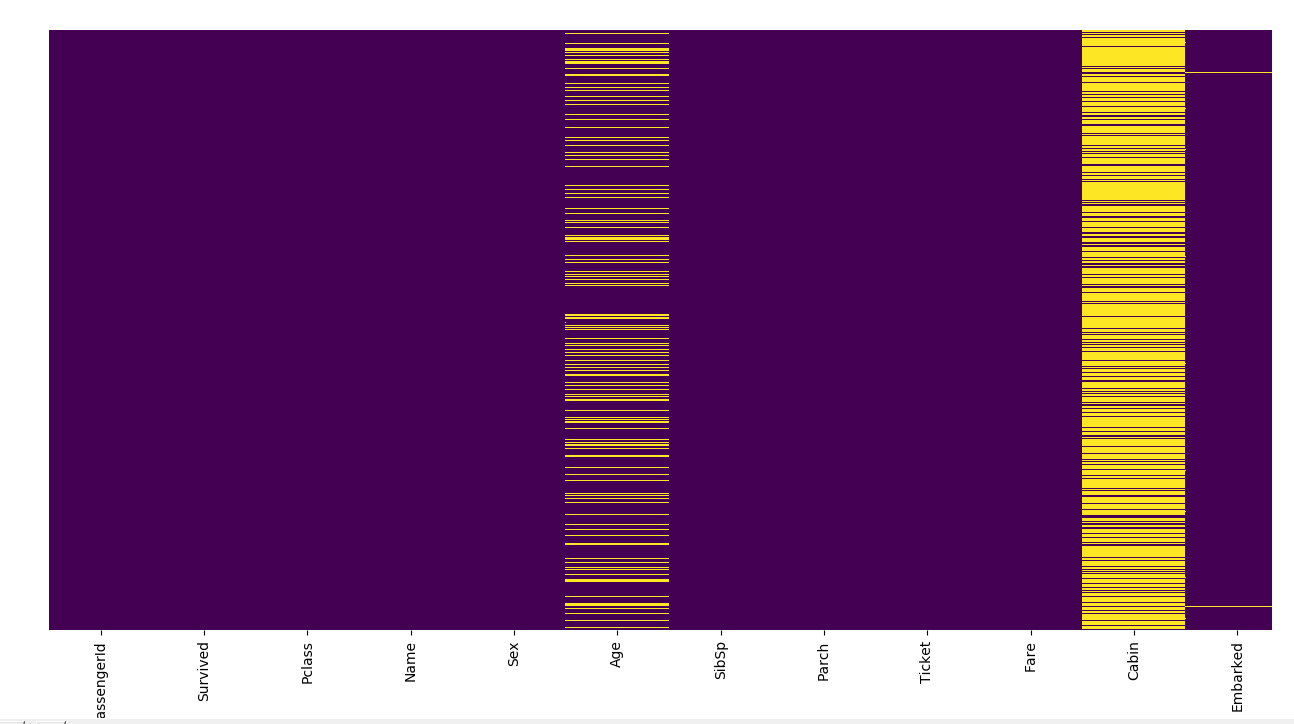
1 False False False False ... False False False False

2 False False False False ... False False True False

Using heatmap to visualize columns having null data. The yellow lines in the plot denote the False points means null data, so we can observe that some data in age column is null whereas, lots of data in cabin column is null.

We will see further how to modify this null data.

sns.heatmap(train\_data.isnull(),yticklabels=False,cbar=False,cmap="viridis")

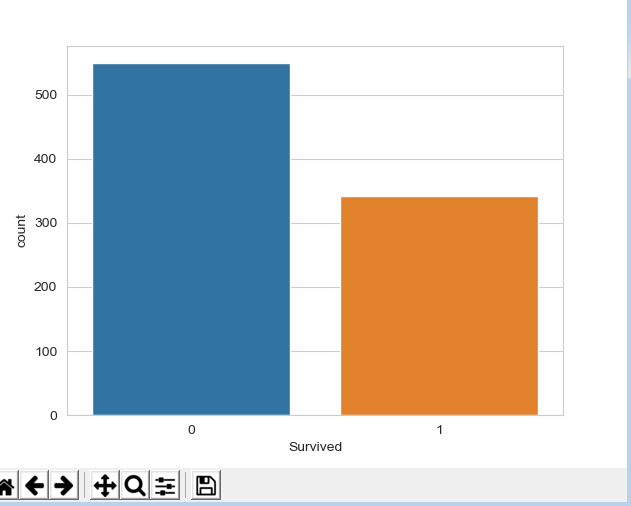


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably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

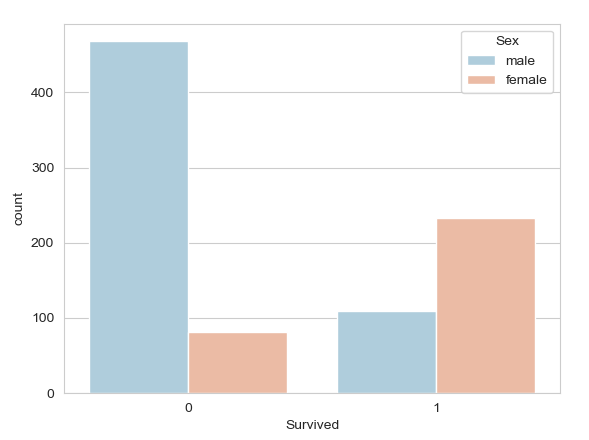
## Visualizing data

First we will visualize count of persons survived and deceased which can be done using count plot

# Analysing count of persons survived vs deceased  
plt.figure(2)  
sns.set\_style("whitegrid")  
sns.countplot(x="Survived",data=train\_data)

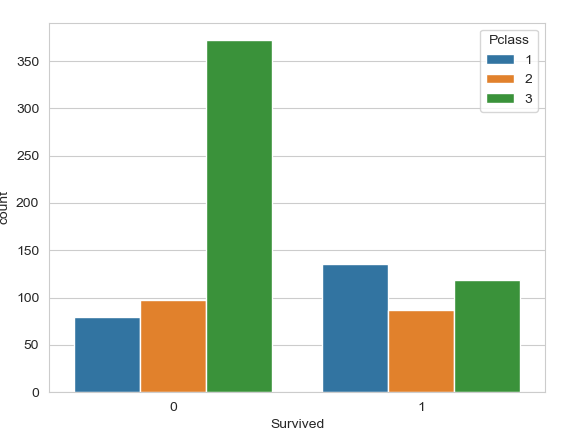


sns.countplot(x="Survived",data=train\_data,hue="Sex",palette="RdBu\_r")



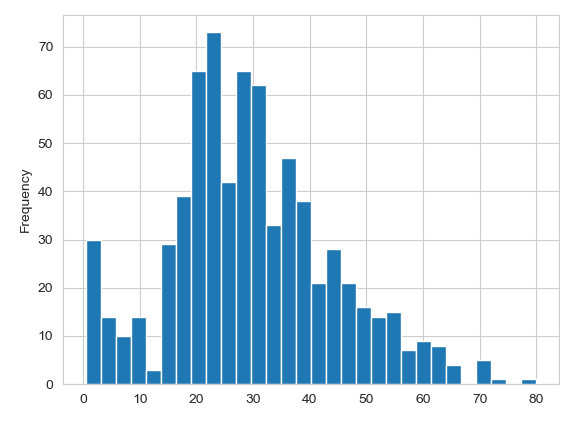
We will now get an idea about people survived and their passenger class. The trend which we observed here will help us later in understanding the model and the weights of coefficients.

# Analysing count of persons survived vs deceased on basis of passenger class  
plt.figure(3)  
sns.set\_style("whitegrid")  
sns.countplot(x="Survived",data=train\_data,hue="Pclass")



We will now analyze the age of passengers on titanic using distribution plots, this has been done using pandas internal visualizations but can also be done using seaborn.

# Analysing age of passengers present on titanic  
plt.figure(4)  
train\_data["Age"].dropna().plot.hist(bins=30)



## 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:

# Data Cleaning  
# Plotting a box plot to check relationship between passenger class and age  
plt.figure(num=7)  
sns.boxplot(x="Pclass",y="Age",data=train\_data)

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.

def impute\_age(cols):  
 age = cols[0]  
 pClass = cols[1]  
 temp\_df = train\_data.groupby("Pclass").mean().round(0)  
 if pd.isnull(age):  
 if pClass == 1:  
 return temp\_df.loc[1, "Age"]  
 elif pClass == 2:  
 return temp\_df.loc[2, "Age"]  
 else:  
 return temp\_df.loc[3, "Age"]  
 else:  
 return age

# Filling empty age data with mean of age with relation to passenger class and again analysing the heatmap to  
# verify null values in age  
train\_data["Age"] = train\_data[["Age","Pclass"]].apply(impute\_age,axis=1)  
plt.figure(num=8)  
sns.heatmap(train\_data.isnull(),yticklabels=False,cbar=False,cmap="viridis")

Great! Let's go ahead and drop the Cabin column since it has lots of empty data and that can’t be take care of in a good manner and also the row in Embarked that is NaN.

## 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 our titanic data set, we have sex and embarked columns which have categorical data and we need to convert them.

# Converting categorical columns into useful features  
sex = pd.get\_dummies(train\_data["Sex"],drop\_first=True) # we have dropped first since it gives output two columns male and  
 # female which will be perfect fit for each other and these columns  
 # can cause our ML algorithm to perform badly  
embark = pd.get\_dummies(train\_data["Embarked"],drop\_first=True) # Since this column has three categories, so dropping only  
 # the first column will be sufficient  
train\_data = pd.concat([train\_data,sex,embark],axis=1)  
print("\*\*\*\*\*\*Checking head of new data frame with converted categorical columns\*\*\*\*\*\*")  
print(train\_data.head(5))

We will also drop passengerId since it will not have any impact on our algorithm since it’s just normal ID of a passenger

# Dropping passengerId column since its redundant  
train\_data.drop("PassengerId",inplace=True,axis=1)  
print("\*\*\*\*\*\*Checking head of final data frame\*\*\*\*\*\*")  
print(train\_data.head(5))  
print()

## Training model and predicting values

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).

# Specifing features and y  
X = train\_data.drop("Survived",axis=1)  
y = train\_data["Survived"]  
  
# Splitting data into train and test data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)  
  
# Instantiating model, training it and predicting values  
logicModel = LogisticRegression()  
logicModel.fit(X=X\_train,y=y\_train)  
predictions = logicModel.predict(X\_test)

## Evaluating model

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

# Evaluating model using scikit's classification report  
print("\*\*\*\*\*Classification report\*\*\*\*\*")  
print(classification\_report(y\_true=y\_test,y\_pred=predictions))  
print()  
  
# Evaluating model using confusion matrix  
print("\*\*\*\*\*Confusion matrix\*\*\*\*\*")  
print(confusion\_matrix(y\_true=y\_test,y\_pred=predictions))  
print()

# Complete Code-

def impute\_age(cols):  
 age = cols[0]  
 pClass = cols[1]  
 temp\_df = train\_data.groupby("Pclass").mean().round(0)  
 if pd.isnull(age):  
 if pClass == 1:  
 return temp\_df.loc[1, "Age"]  
 elif pClass == 2:  
 return temp\_df.loc[2, "Age"]  
 else:  
 return temp\_df.loc[3, "Age"]  
 else:  
 return age  
  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report,confusion\_matrix  
import warnings  
warnings.filterwarnings("ignore", category=FutureWarning)  
  
# Reading training data  
train\_data = pd.read\_csv("C:/Users/G01212601/Downloads/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/13-Logistic-Regression/titanic\_train.csv")  
  
# Analysing null data in our data set  
plt.figure(1)  
sns.heatmap(train\_data.isnull(),yticklabels=False,cbar=False,cmap="viridis")  
  
# Analysing count of persons survived vs deceased on basis of sex  
plt.figure(2)  
sns.set\_style("whitegrid")  
sns.countplot(x="Survived",data=train\_data,hue="Sex",palette="RdBu\_r")  
  
# Analysing count of persons survived vs deceased on basis of passenger class  
plt.figure(3)  
sns.set\_style("whitegrid")  
sns.countplot(x="Survived",data=train\_data,hue="Pclass")  
  
# Analysing age of passengers present on titanic  
plt.figure(4)  
train\_data["Age"].dropna().plot.hist(bins=30)  
  
# Analysing sibling and spouses  
plt.figure(5)  
sns.set\_style("whitegrid")  
sns.countplot(x="SibSp",data=train\_data)  
  
# Analysing fares on the titanic  
plt.figure(6)  
train\_data["Fare"].dropna().plot.hist(bins=50)  
  
# # Can use cufflinks for more interactive plots  
# import cufflinks as cf  
# cf.go\_offline()  
# train\_data["Fare"].dropna().iplot(kind="hist",bins=50)  
  
# Data Cleaning  
# Plotting a box plot to check relationship between passenger class and age  
plt.figure(num=7)  
sns.boxplot(x="Pclass",y="Age",data=train\_data)  
  
# Filling empty age data with mean of age with relation to passenger class and again analysing the heatmap to  
# verify null values in age  
train\_data["Age"] = train\_data[["Age","Pclass"]].apply(impute\_age,axis=1)  
plt.figure(num=8)  
sns.heatmap(train\_data.isnull(),yticklabels=False,cbar=False,cmap="viridis")  
  
# Dropping whole Cabin column and empty row in Embarked column  
train\_data.drop("Cabin",inplace=True,axis=1)  
train\_data.dropna(inplace=True)  
plt.figure(num=9)  
sns.heatmap(train\_data.isnull(),yticklabels=False,cbar=False,cmap="viridis")  
  
# Converting categorical columns into useful features  
sex = pd.get\_dummies(train\_data["Sex"],drop\_first=True) # we have dropped first since it gives output two columns male and  
 # female which will be perfect fit for each other and these columns  
 # can cause our ML algorithm to perform badly  
embark = pd.get\_dummies(train\_data["Embarked"],drop\_first=True) # Since this column has three categories, so dropping only  
 # the first column will be sufficient  
train\_data = pd.concat([train\_data,sex,embark],axis=1)  
print("\*\*\*\*\*\*Checking head of data frame with converted categorical columns\*\*\*\*\*\*")  
print(train\_data.head(5))  
print()  
  
# Dropping columns which may not be required in algorithm  
train\_data.drop(["Name","Sex","Ticket","Embarked"],inplace=True,axis=1)  
  
# Dropping passengerId column since its redundant  
train\_data.drop("PassengerId",inplace=True,axis=1)  
print("\*\*\*\*\*\*Checking head of final data frame\*\*\*\*\*\*")  
print(train\_data.head(5))  
print()  
  
# Specifing features and y  
X = train\_data.drop("Survived",axis=1)  
y = train\_data["Survived"]  
  
# Splitting data into train and test data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)  
  
# Instantiating model, training it and predicting values  
logicModel = LogisticRegression()  
logicModel.fit(X\_train,y\_train)  
predictions = logicModel.predict(X\_test)  
  
# Printing predictions and y\_test head  
print("\*\*\*\*\*Predictions head\*\*\*\*\*")  
print(pd.DataFrame(predictions).head())  
print()  
print("\*\*\*\*\*Y\_test head\*\*\*\*\*")  
print(pd.DataFrame(y\_test).head())  
print()  
  
# Evaluating model using scikit's classification report  
print("\*\*\*\*\*Classification report\*\*\*\*\*")  
print(classification\_report(y\_true=y\_test,y\_pred=predictions))  
print()  
  
# Evaluating model using confusion matrix  
print("\*\*\*\*\*Confusion matrix\*\*\*\*\*")  
print(confusion\_matrix(y\_true=y\_test,y\_pred=predictions))  
print()  
  
plt.show()