Titanic

We're going to be working with the Titanic - Machine Learning from Disaster competition available on Kaggle.com [https://www.kaggle.com/competitions/titanic/overview]. We will predict the classification: Survived or Deceased, for passengers who were on the Titanic. The train and test data files were downloaded from the website and are attached with the submission.

Importing Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

Importing the Data

We will 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	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

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

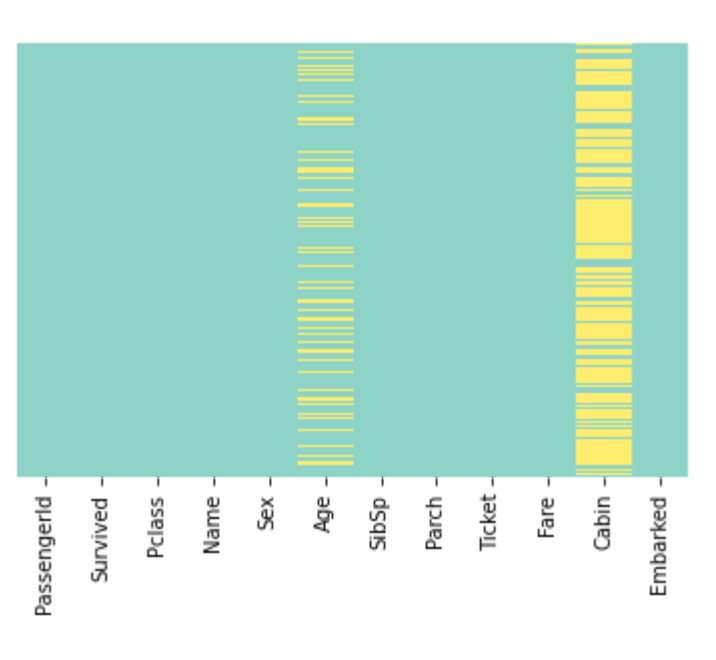
Out[4]:

	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

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

```
In [5]: sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='Set3')
Out[5]: <AxesSubplot:>
```



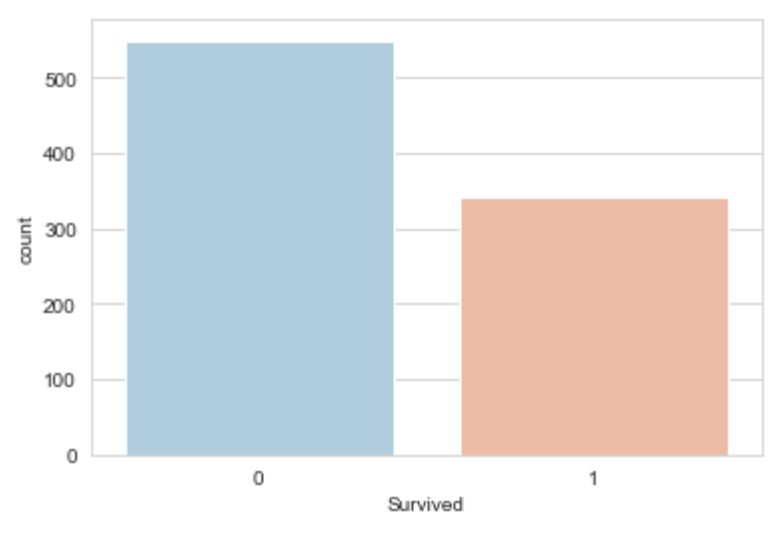
Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably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

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

We will first set style to 'whitegrid' just because I think it looks nice. First thing I want to do is just get a count of who survived who didn't survive. For classification problem, it's always a good idea just to see the ratio of the actual target labels.

Now we can do this for simple counterplot. I'm going to go ahead and add a palette argument to make this a little easier to see.

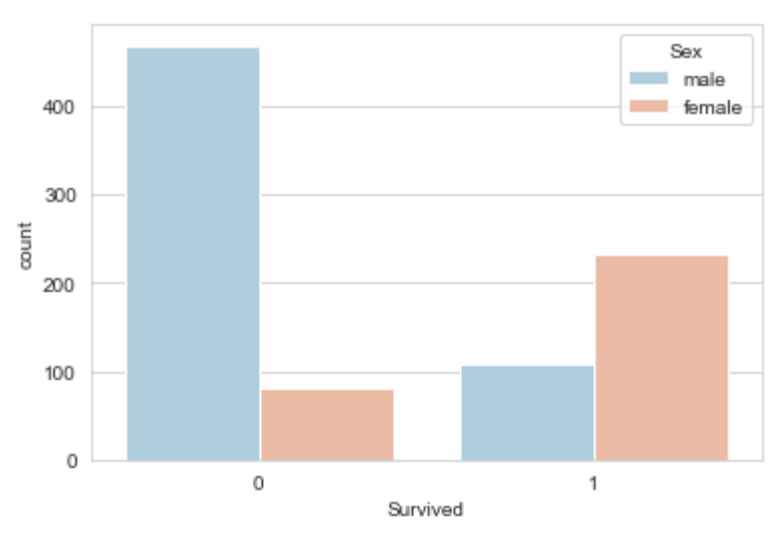
```
In [6]: sns.set_style('whitegrid')
        sns.countplot(x='Survived',data=train,palette='RdBu_r')
Out[6]: <AxesSubplot:xlabel='Survived', ylabel='count'>
```



And here we can see that I have quite a few more people that did not survive(0) versus people who survived(1). It looks like around 550 of these passengers in this particular training set did not survive and we have about 350ish, maybe a little less that actually survived.

We're going to go ahead and create another count plot of seeing survival with the hue of sex. According to the palette argument, male is blue and female is red or a light pink color.

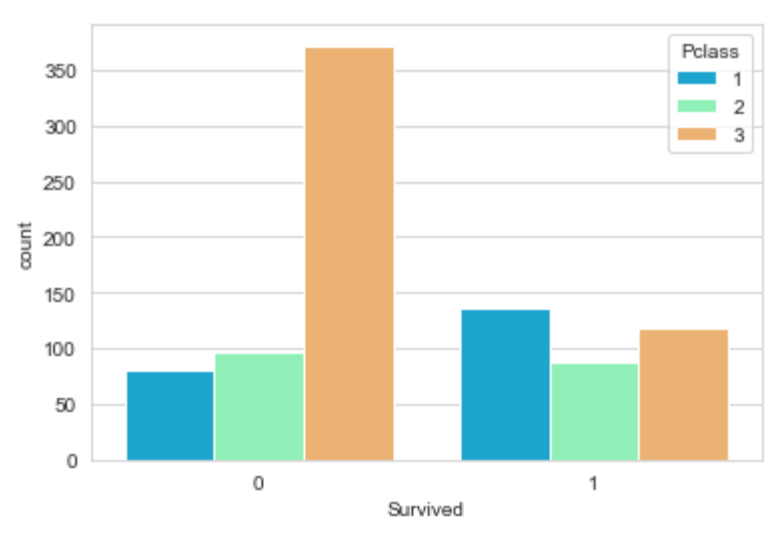
```
In [7]: sns.set_style('whitegrid')
        sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
Out[7]: <AxesSubplot:xlabel='Survived', ylabel='count'>
```



Looking at this plot, I can really tell there's a trend here. It looks like people that did not survive were much more likely to be male and people that did survive were almost about twice as likely to be female. And we're going to see that trend come up later when we're looking at the factors and coefficients of what led to someone's survival.

Let's go ahead and continue exploring the data by changing this hue to the passenger class. We will change the palette argument to rainbow.

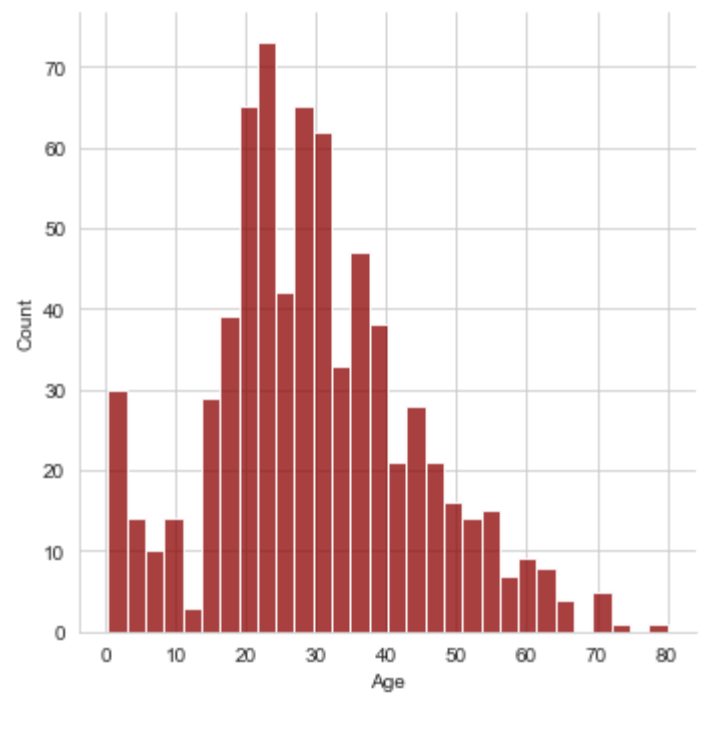
```
In [8]: sns.set_style('whitegrid')
        sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
Out[8]: <AxesSubplot:xlabel='Survived', ylabel='count'>
```



Looking at this we can kind of tell the trends here on survived versus not survived. It looks like the people who did not survive were overwhelmingly part of the third class or the lowest class that was the cheapest to get onto. And it looks like more or less the people that did survive were leaning a little more towards the higher classes: One and Two. We will also have to see how many people are actually in the passenger class or per passenger class.

Let's go ahead and just get an idea of the age of people on the Titanic. To do that, I can go ahead and do a Distribution type plot.

```
In [9]: sns.displot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
Out[9]: <seaborn.axisgrid.FacetGrid at 0x1b0877dca90>
```



We have what appears to be almost bimodel distribution where we have quite a few young passengers between ages 0 and 10. So there's quite a few children in that zone. But there after, we can see it starts to get an average age towards somewhere around 20 and 30.

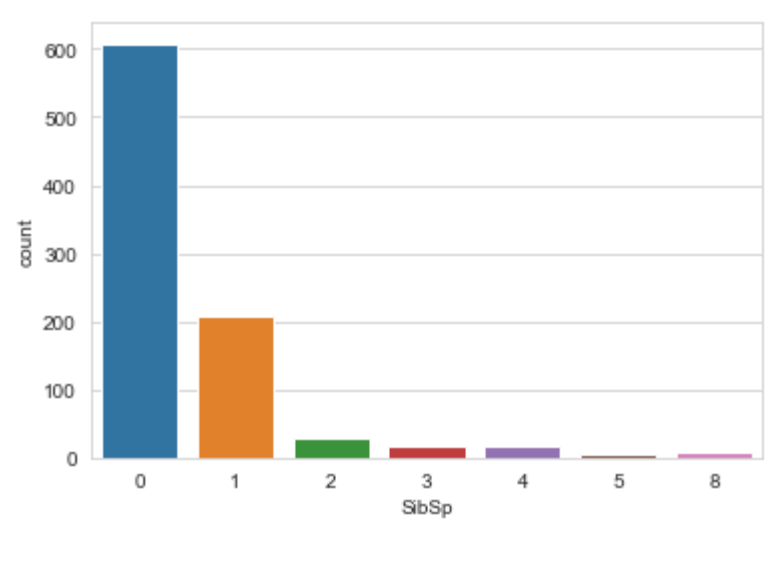
So it is quite maybe skewed towards younger passengers. And as the older you get the less representation you have on board.

Let's go ahead and explore some of the other coulumns just to get a reminder of what those coulumns are.

```
In [10]: train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Let's go ahead and explore SibSp which is the number of siblings or spouses on board.

```
In [11]: sns.countplot(x='SibSp',data=train)
Out[11]: <AxesSubplot:xlabel='SibSp', ylabel='count'>
```

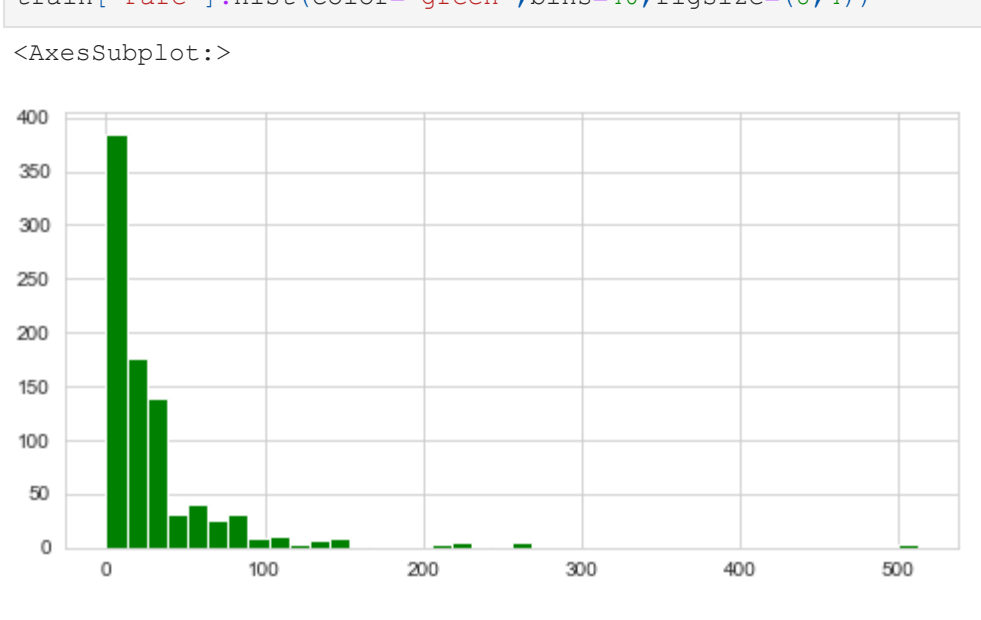


Now looking at this plot I can basically immediately tell that most people onboard did not have neither children or a spouse on board. And if we look at the second most popular option it's just one which is actually most likely to be probably a spouse versus just having a single parent when children onboard. So those numbers pretty much make sense, we have a lot of single people on board. They don't have a spouse and how many children are onboard. Probably, those are the men in the third class.

And after that you have the second most common type of passenger in our data set is some of just one, either sibling or spouse. That's probably going to be their spouse. Couples on board.

Another column we haven't actually explored yet is the Fare column. Let's go ahead and take a look at how much people paid as a distribution.

```
In [12]: train['Fare'].hist(color='green',bins=40,figsize=(8,4))
Out[12]: <AxesSubplot:>
```

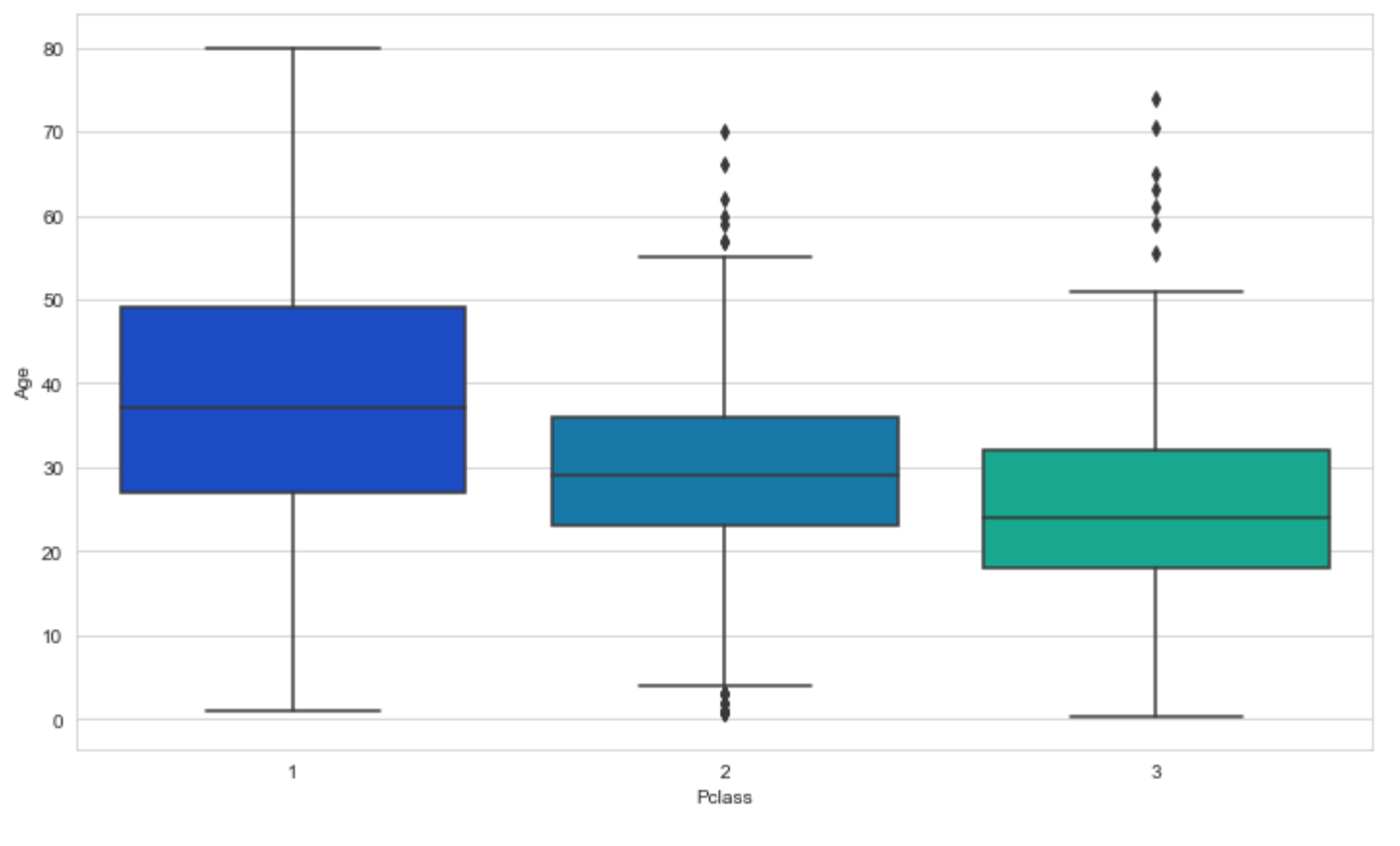


It looks like most of the purchase prices are between 0 on a 50. So it makes sense that things are distributed towards the cheaper fare tickets because as you have already seen most passengers are actually in the cheaper third class. And keep in mind these are fares back when the Titanic was around and that was in 1912. So these prices haven't been adjusted to inflation.

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.

```
In [13]: plt.figure(figsize=(12, 7))
         sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
Out[13]: <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.

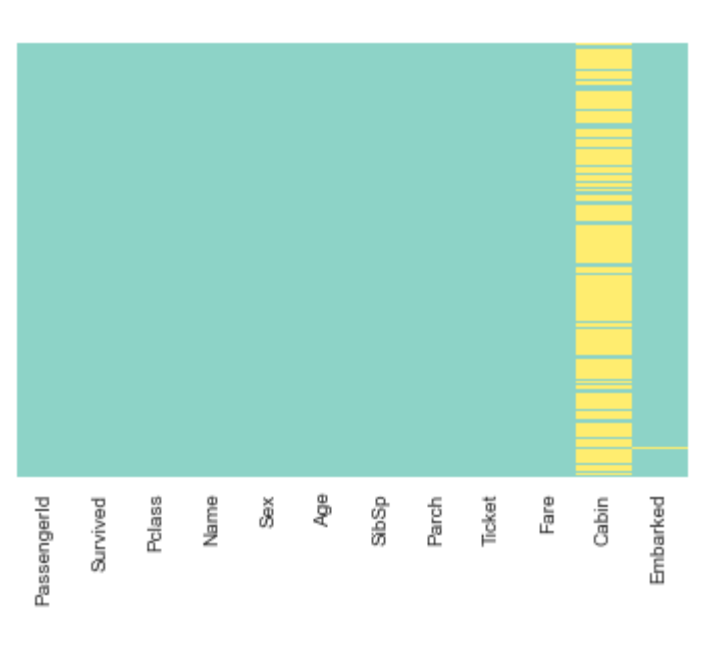
```
In [14]: def impute_age(cols):
             Age = cols[0]
             Pclass = cols[1]
             
             if pd.isnull(Age):

                 if Pclass == 1:
                     return 37

                 elif Pclass == 2:
                     return 29

                 else:
                     return 24

             else:
                 return Age
```

Now let's apply that function!

```
In [15]: train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

```
In [16]: sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='Set3')
Out[16]: <AxesSubplot:>
```



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

```
In [17]: train.drop('Cabin',axis=1,inplace=True)
```

```
In [18]: train.head()
```

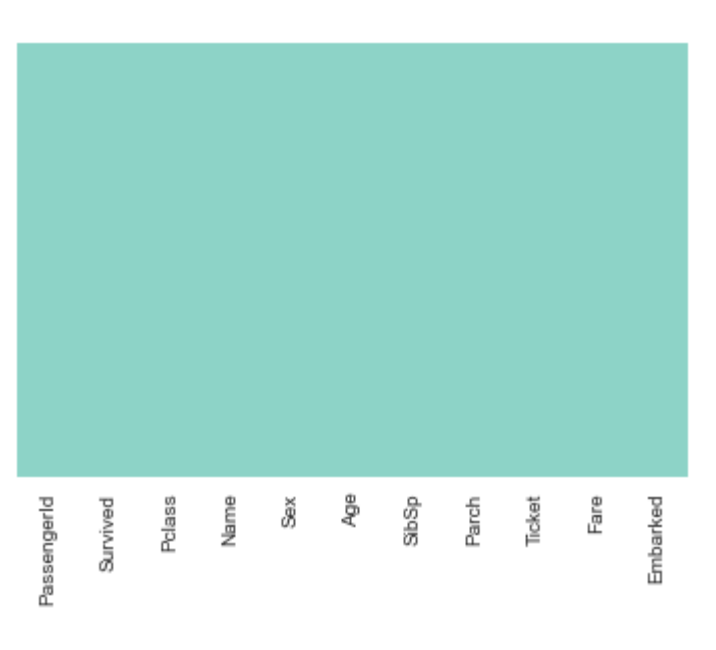
Out[18]:

	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 [19]: train.dropna(inplace=True)
```

Let's check the heat map once more.

```
In [20]: sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='Set3')
Out[20]: <AxesSubplot:>
```



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 [21]: 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
```

If you see the sex column, we have a categorical feature of male or female. A machine learning algorithm isn't going to take in just a string of male or female. We will have to create a new column of a zero or one value for if someone is male or not, in order to encode that information in a way that a machine learning algorithm can understand it. We'll do a similar thing for the embarked column representing cities.

```
In [22]: sex = pd.get_dummies(train['Sex'],drop_first=True)
         embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

Let's drop the columns we don't want to feed to our algorithm.

```
In [23]: train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

Let's check what our new coulumns look like.

```
In [24]: sex.head()
```

Out[24]:

	male
0	1
1	0
2	0
3	0
4	1

```
In [25]: embark.head()
```

Out[25]:

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

We're going use concatenation to add these new columns to our data frame.

```
In [26]: train = pd.concat([train,sex,embark],axis=1)
```

Let's ckeck out the head of our data frame.

```
In [27]: train.head()
```

Out[27]:

	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

Notice we have these new columns Male, Q and S. These are going to be a replacement columns for our machine learning algorithm.

The last thing we want to look at is this first numerical column was this Passenger Id. Notice this Passenger Id is essentially just an index that starts at 1.

We go ahead and check the tail of this.

```
In [28]: train.tail()
```

Out[28]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
886	887	0	2	27.0	0	0	13.00	1	0	1
887	888	1	1	19.0	0	0	30.00	0	0	1
888	889	0	3	24.0	1	2	23.45	0	0	1
889	890	1	1	26.0	0	0	30.00	1	0	0
890	891	0	3	32.0	0	0	7.75	1	1	0

Even though it's numerical, it's not really useful to us essentially as an index or to predict whether passengers survive just based off their position index.

```
In [29]: train.drop(['PassengerId'],axis=1,inplace=True)
```

Let's take a final look at the columns one last time to make sure everything is okay.

```
In [30]: train.head()
```

Out[30]:

	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 [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=0.30, 
                                                             random_state=101)
```

Training and Predicting

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

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

D:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to conver
ge (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()
```

```
In [35]: 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 [36]: from sklearn.metrics import classification_report
```

```
In [37]: 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
```