Titanic Write-up
Dr. Fulton: MSDS 422
Nicholas Bergeland

This week for my assignment, I reapproached the Titanic dataset from Kaggle, and entered into the competition once more. As opposed to our week 1 discussion post where we got our first look at the dataset, (and last week) this time we were able to build and test a few different models. The best performing model I built this week was the "random forest classifier" a type of regression, decision model. This model scored at 97.98% correct survival classification. If we assume that a random "yes or no" guess yields in the neighborhood of 50% success this is an astounding improvement. It is also higher than both other models (logistic regression, and vector learning methods), which tested in the 80% ranges respectively.

Over the course of training and deploying the models I was able to successfully get all five saved to CSV, and therefore submitted to Kaggle for scoring again this week. Over my submissions I was able to get my submission testing up to .7703 up from .75358 last week.

Before being able to build a useful model, it is useful to learn more about the data. I did this with some simple visualization methods within matplotlib and seaborn packages. The visualization that I found the most insightful was the passenger age by fare class. Unsurprisingly, passenger class moved up with age. This was demonstrated by third class having a median age of twenty five years old. Each class moved up roughly five years respectively with median, all the way up to roughly forty for first class.

Another visualization I found insightful was the number of passengers an individual was accompanied by. The bar chart illustrates, more passengers actually traveled by themselves than with either group or two or three, but less than the two combined. After this we see large drop offs in group size, apart from what appears to be a single group of eight passengers!

After learning about the datasets cleaning and preprocessing are needed to tune them. I went about identifying elements to drop by viewing which columns contained nulls on a seaborn heatmap. The “Cabin” variable contained the most nulls and was likely to impact the models, so I dropped it. I also drop “Sex”, “Embarked”, “Name”, and “Ticket” labels, which were used in the creation of dummy variables during model building.

After tuning the datasets I was able to build out the models. For this assignment five types of models were used (logistic regression, random forest classifier, gradient boosting trees, support vector learning, and extra trees). Of the five, random forest classifier greatly outperformed the other two methods of modeling (by over 10%). Beyond that, if a random guess were to yield a 50% success rate all of the models produced blow that out of the water, further demonstrating the benefits provided by machine learning models. I was pleased with the addition of the extra models this week as it bumped my Kaggle score up (P. 24).

Appendix:

In [3]:
```
Import
os
# Any results you write to the current directory are saved as output.

#Importing all the needed libraries import
numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt import seaborn as sns
```

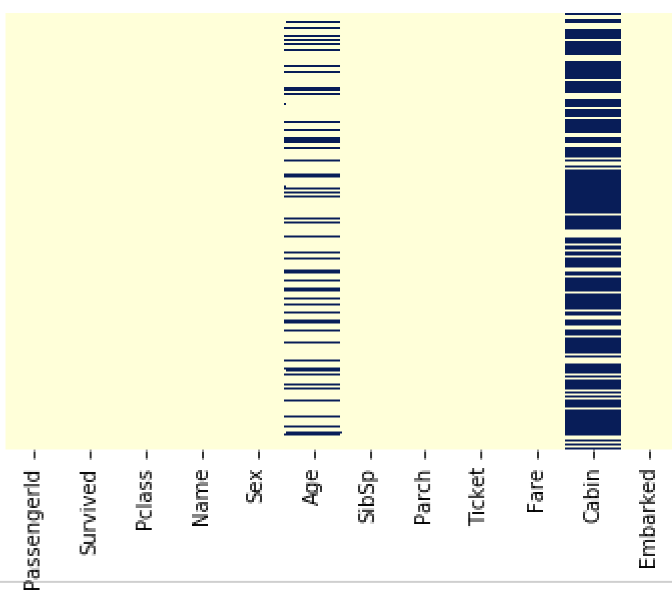
In [8]:
```
#Importing the Training & Test sets
train=pd.read_csv('train.csv')

test = pd.read_csv('test.csv')
```

In [9]:
```
#Visualizing the null values using HeatMaps
sns.heatmap(train.isnull(),yticklabels=False, cbar=False, cmap='YlGnBu')
```

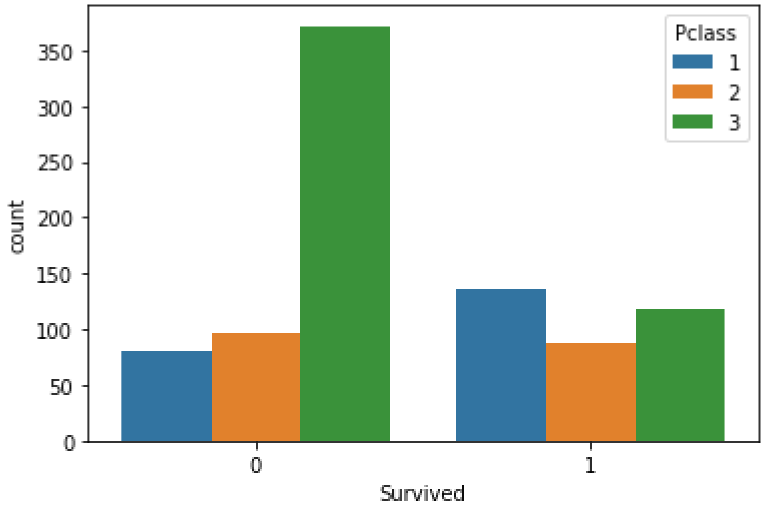
Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2028cf10>
```



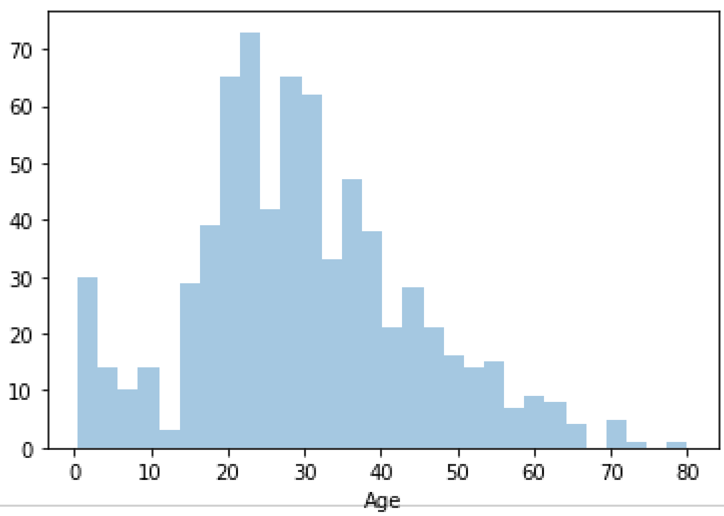
In [10]:
```
#Visualizing Survivors based on their Passenger Classes
sns.countplot(x='Survived',hue='Pclass',data=train)
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2102c110>
```



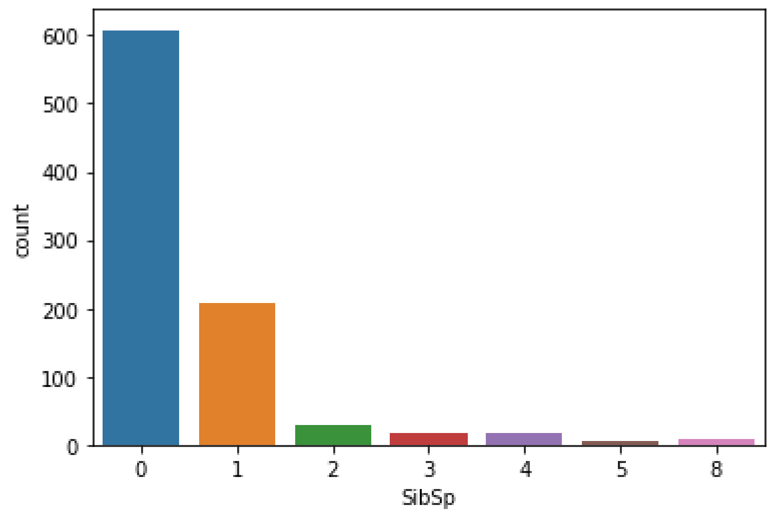
In [11]:
```
#Plotting on the basis of the Age of the Passengers
sns.distplot(train['Age'].dropna(), kde=False, bins=30)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d210d4250>



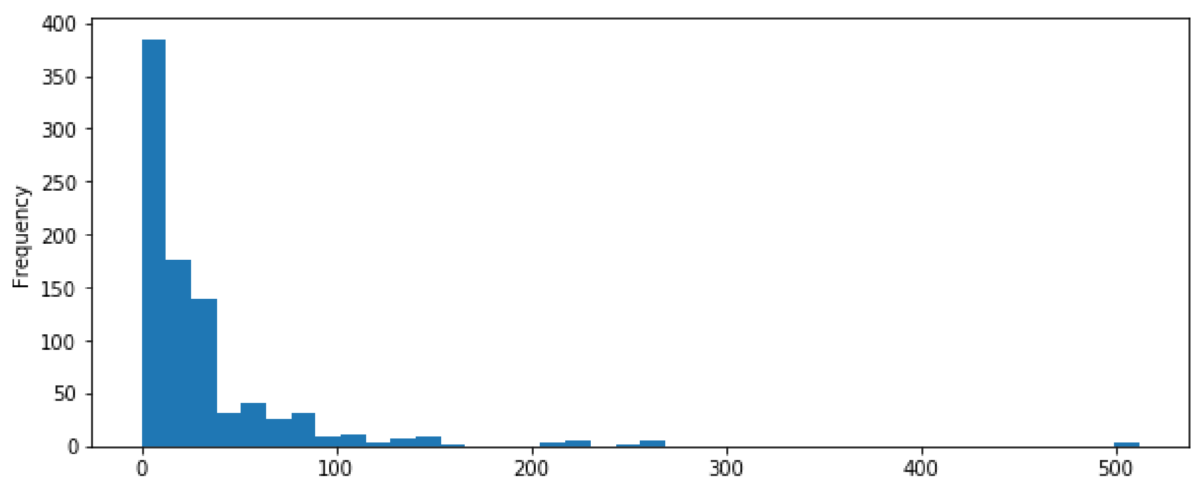
In [12]:
```
#Counting the number of Passengers who had boarded with their siblings a
nd/or their spouses sns.countplot(x='SibSp', data=train)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d2108aad0>



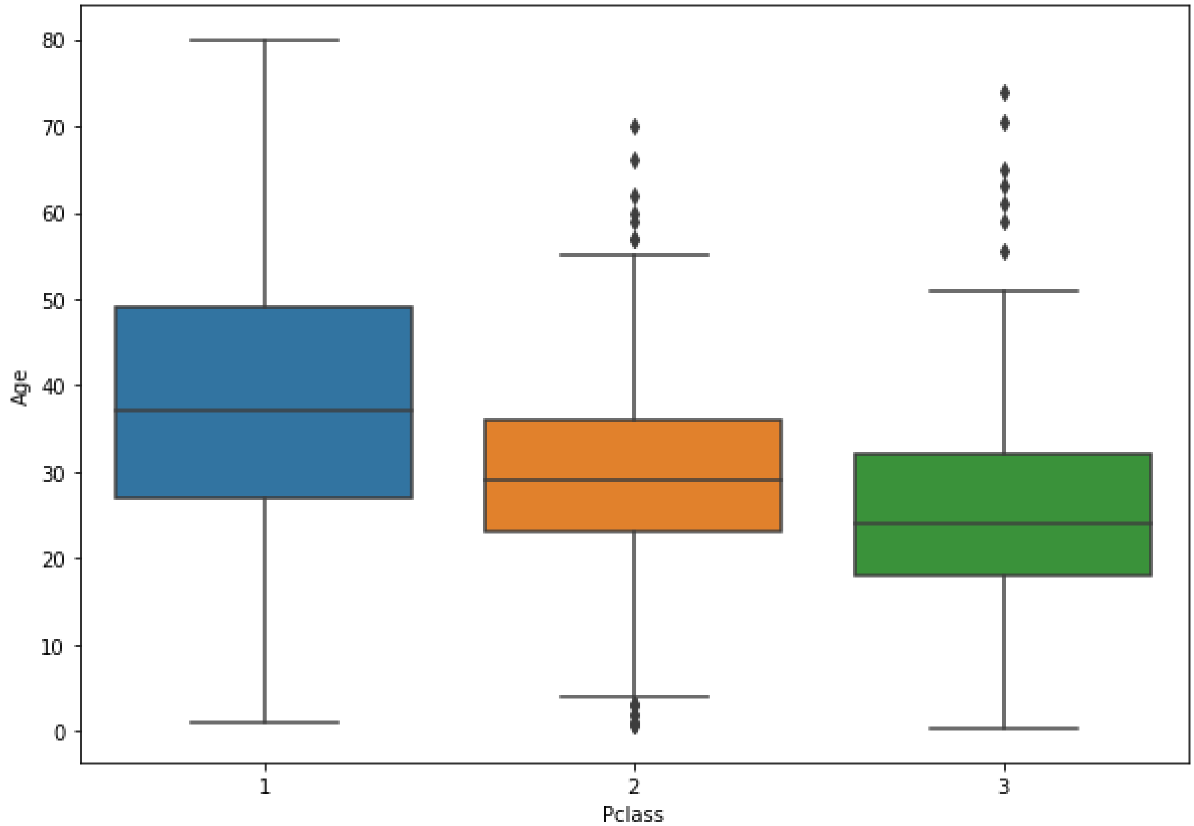
In [13]:
```
#Plotting Fare against the number of Passengers
train['Fare'].plot.hist(bins=40, figsize=(10,4))
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d2127b4d0>



In [14]:
```
#Getting the Age of the Passengers based on their Class, also the
average Age per class plt.figure(figsize=(10,7)) sns.boxplot(x='Pclass',
y='Age', data=train)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d2125b610>



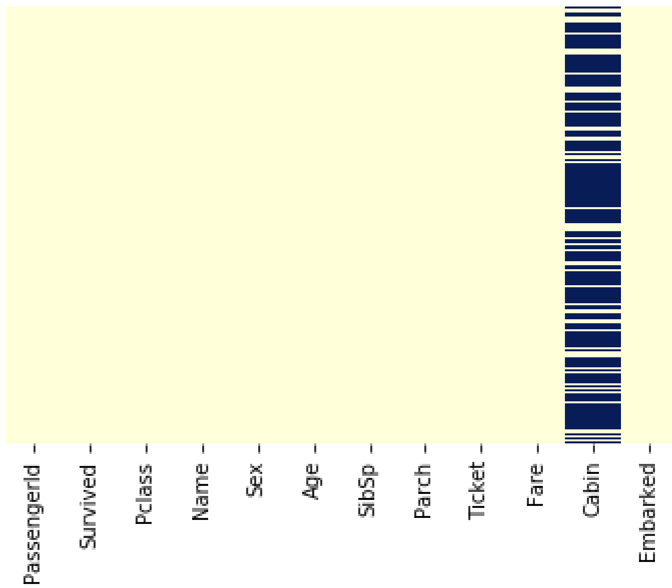
In [15]:
```
#Defing a  function that will impute the Age columns on the basis of the
Pclass
def impute_age(cols):
    Age=cols[0]
    Pclass=cols[1]

    if pd.isnull(Age):
        if Pclass==1:
            return 37
        elif Pclass==2:
            return 29
        else:
            return 24
    else:
        return Age
```

In [16]:
```
#Calling the above defined function to impute the Age column
train['Age']=train[['Age', 'Pclass']].apply(impute_age, axis=1)
```

In [17]:
```python
#Visualizing after the Imputation of the Age column
sns.heatmap(train.isnull(), yticklabels=False, cbar=False, cmap='YlGnBu'
)
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d1e0acb90>



In [18]:
```python
#Dropping the Cabin column since it doesn't have any direct effect on th
e prediction
train.drop('Cabin', axis=1, inplace=True)
```

In [19]:
```python
#Checking the dataset
train.head()
```

Out[19]:

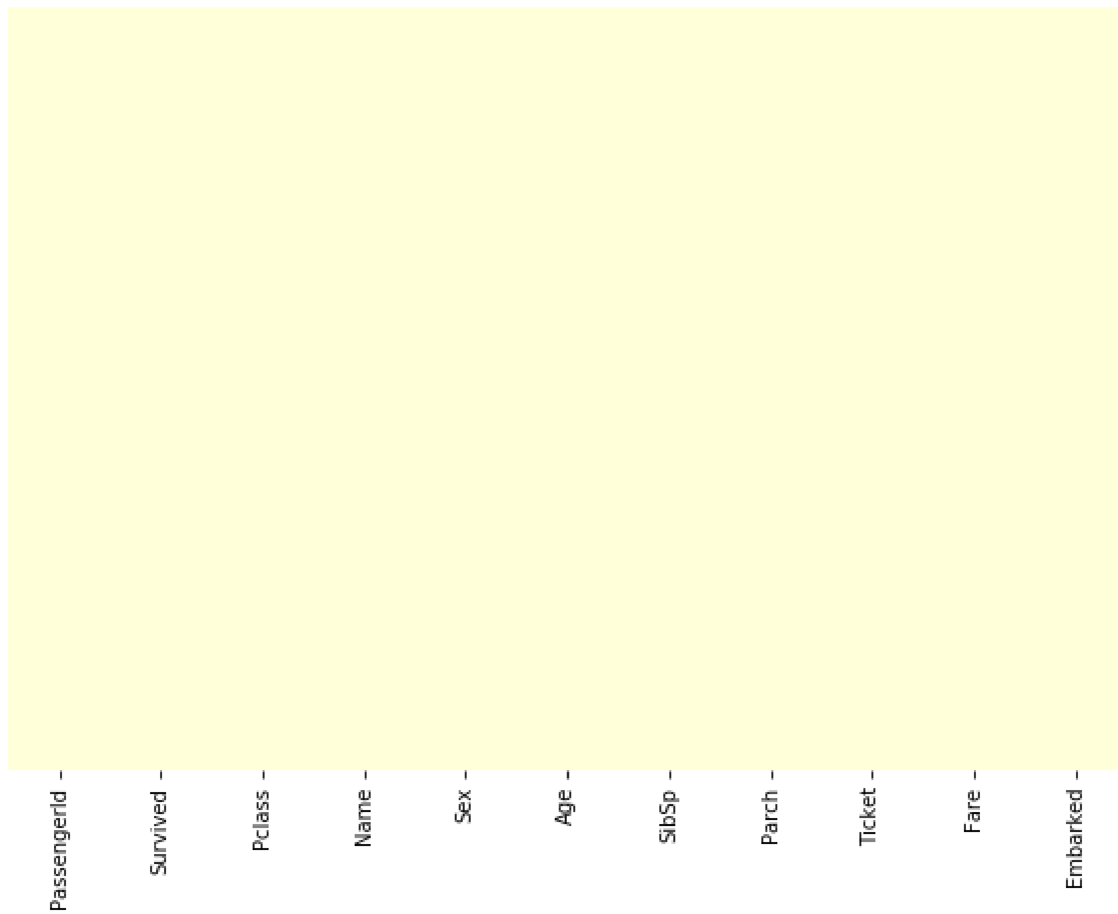
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	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 [20]:
```
#Using HeatMap again to visualize the resultant set
plt.figure(figsize=(10,7))
sns.heatmap(train.isnull(), yticklabels=False, cbar=False,
cmap='YlGnBu' )
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d2171b590>



In [21]:
```
#Dropping the remaining rows with null values in them
train.dropna(inplace=True)
```

In [22]:
```
#Dealing with categorical column Sex, Embarked, making dummmies
sex=pd.get_dummies(train['Sex'], drop_first=True)
embark=pd.get_dummies(train['Embarked'], drop_first=True)
classes=pd.get_dummies(train['Pclass'])
```

In [23]:
```
#Concatinating the newly created dummy columns with the existing datafra
me
train=pd.concat([train,sex,embark], axis=1)
train=pd.concat([train,classes], axis=1)
```

In [24]:
```
#Dropping columns which will not be used during training
train.drop(['Sex', 'Embarked', 'Name', 'Ticket'], axis=1, inplace=True)
train.drop('PassengerId', axis=1, inplace=True)
train.drop('Pclass', axis=1, inplace=True)
```

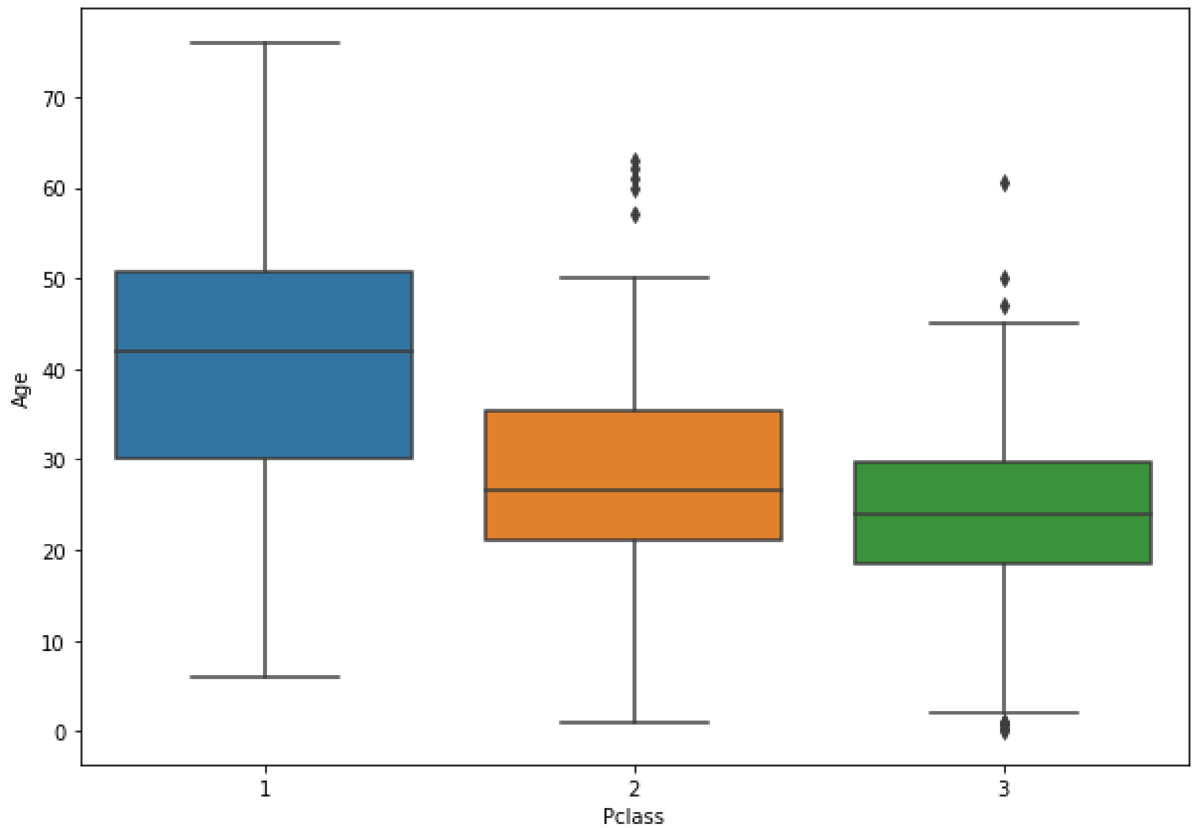
In [25]:
```
# We are done with cleaning of the training set. We now need to
do the s ame to the Test set
```

```
In [26]: #Visulazing missing values using HeatMap
plt.figure(figsize=(18,12))
sns.heatmap(test.isnull(),yticklabels=False, cbar=False,
cmap='YlGnBu')
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d21996b10>
```


```
In [27]: #Using BoxPlot to see which age group belongs to which
Pclass plt.figure(figsize=(10,7))
sns.boxplot(x='Pclass', y='Age', data=test)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d2198c7d0>
```



In [28]:
```
#Defining a function which will be used to impute Age columns of the Tes
t set
def impute_age2(cols):
    Age=cols[0]
    Pclass=cols[1]

    if pd.isnull(Age):
        if Pclass==1:
            return 42
        elif Pclass==2:
            return 26
        else:
            return 24
    else:
        return Age
```

In [29]:
```
#Imputing the Age column by calling the above function
test['Age']=test[['Age', 'Pclass']].apply(impute_age2, axis=1)
```

In [30]:
```
# As we saw in the heatmap that some of the values of 'Fare'
column are  missing
#Imputing Fare column for missing
values from sklearn.preprocessing
import Imputer
```

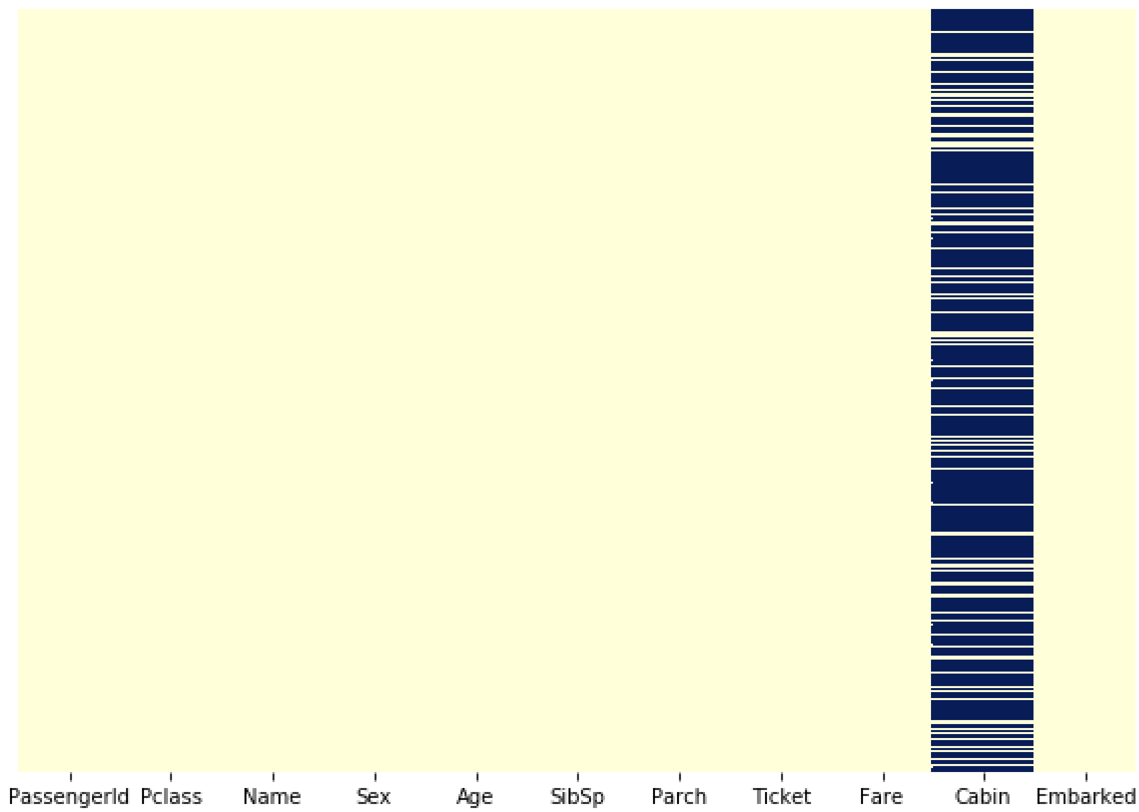
```
imputer = Imputer(missing_values = 'NaN', strategy = 'mean',
axis = 0) imputer = imputer.fit(test.iloc[:, 8:9])
test.iloc[:, 8:9] = imputer.transform(test.iloc[:, 8:9])
```

```
  /Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/skle arn/utils/deprecation.py:66: DeprecationWarning:
Class Imputer is depre cated; Imputer was deprecated in version
                 0.20 and will be removed in 0.
22. Import impute.SimpleImputer from sklearn instead.
warnings.warn(msg, category=DeprecationWarning)
```

In [31]:

```
#Using HeatMap again to visualize the remaining missing values
from the  set
plt.figure(figsize=(10,7)) sns.heatmap(test.isnull(),
yticklabels=False, cbar=False, cmap='YlGnBu')
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d22285d10>



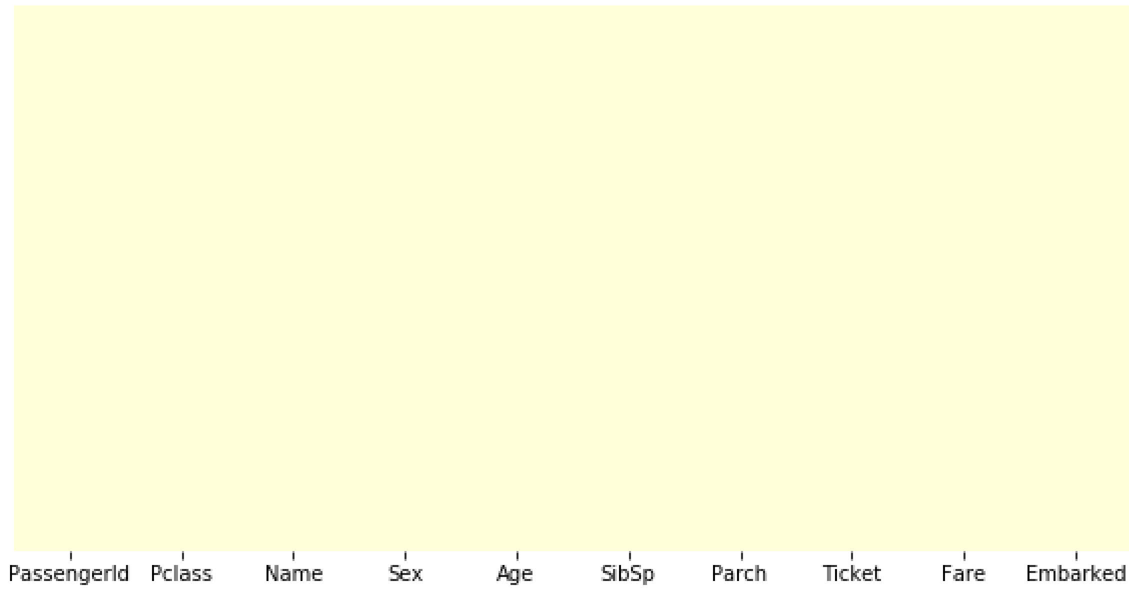
In [32]:

```
# The 'Cabin' column has too many values missing, so cannot perform impu
tation here.
#Dropping the Cabin column as done in the Training set
test.drop('Cabin', axis=1, inplace=True)
```

In [33]:

```
#Visualizing again to see the effect after Imputation
plt.figure(figsize=(10,5)) sns.heatmap(test.isnull(),
yticklabels=False, cbar=False, cmap='YlGnBu')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d22298550>



In [34]:
```
#Creating Dummy variables for categorical feaatures of the set and conca
tinating them to the Test set
sex2=pd.get_dummies(test['Sex'], drop_first=True)
embark2=pd.get_dummies(test['Embarked'], drop_first=True)
test=pd.concat([test,sex2,embark2], axis=1)
pclasses=pd.get_dummies(test['Pclass'])
test=pd.concat([test,pclasses], axis=1)
```

In [35]:
```
#Saving PassengerId before dropping it so that we can add it to the resu
ltant csv file later
result=test.iloc[:,0]
```

In [36]:
```
#Dropping the redundant features which were converted to dummies earlier
test.drop([ 'Sex', 'Embarked', 'Name', 'Ticket'], axis=1, inplace=True)
test.drop('PassengerId', axis=1, inplace=True)
test.drop('Pclass', axis=1, inplace=True)
```

In [37]:
```
#Building a logistic regression model
#Train Test Split
x_train=train.iloc[:,1:]
y_train=train.iloc[:,0:1]
x_test=test.iloc[:,:]
```

In [38]:

```
#Training and predicting
from sklearn.linear_model import LogisticRegression
logisticReg=LogisticRegression()
logisticReg.fit(x_train,y_train)

y_pred= logisticReg.predict(x_test)
```

```
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/skle arn/linear_model/logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/skle arn/utils/validation.py:724:
DataConversionWarning: A column-vector y w as passed when a 1d
array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [39]:

```
#calculating our models accuracy
accuracy = round(logisticReg.score(x_train, y_train) * 100, 2)
```

In [40]:

```
#Check accuracy
print(accuracy)
```

```
81.21
```

In [41]:

```
#predictions to csv
df=pd.DataFrame(dict(PassengerId = result, Survived = y_pred)).reset_ind
ex()
df.drop('index', axis=1, inplace=True)

df.to_csv('logresult.csv', index=False)
```

In [42]:

```
#using random forest classifier
from sklearn.ensemble import RandomForestClassifier
ranFor = RandomForestClassifier(n_estimators = 70)
```

In [43]:

```
#training and predicting
ranFor.fit(x_train,y_train)
y_pred2= ranFor.predict(x_test)
```

```
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/ipyk ernel_launcher.py:2: DataConversionWarning: A
column-vector y was passe d when a 1d array was expected.
Please change the shape of y to (n_samp les,), for example
using ravel().
```

In [44]:
```
#calculate accuracy of model 2
accuracy2 =round(ranFor.score(x_train, y_train)*100,2)
```

In [45]:
```
print(accuracy2)
```

```
97.98
```

In [46]:
```
#write these results to CSV
df=pd.DataFrame(dict(PassengerId = result, Survived = y_pred2)).reset_in
dex()
df.drop('index', axis=1, inplace=True)

df.to_csv('rfcresult.csv', index=False)
```

In [47]:
```
#using support vector machine
from sklearn.svm import SVC
svc=SVC()
```

In [48]:
```
#training and predicting
svc.fit(x_train, y_train)

y_pred3=svc.predict(x_test)
```

```
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/skle arn/utils/validation.py:724:
DataConversionWarning: A column-vector y w as passed when a 1d
array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-
packages/skle arn/svm/base.py:193: FutureWarning: The default
value of gamma will cha nge from 'auto' to 'scale' in version
0.22 to account better for unscal ed features. Set gamma
explicitly to 'auto' or 'scale' to avoid this wa rning.
  "avoid this warning.", FutureWarning)
```

In [49]:
```
#calculating model accuracy
accuracy3=round(svc.score(x_train, y_train)*100,2)
```

In [50]:
```
print(accuracy3)
```

```
87.74
```

In [51]:
```
#Writing the predictions to a csv file
df=pd.DataFrame(dict(PassengerId = result, Survived = y_pred2)).reset_in
dex()
df.drop('index', axis=1, inplace=True)

df.to_csv('svcresult.csv', index=False)
```

In [ ]:

GBT

```
# Titanic GBT
# This Python 3 environment comes with many helpful analytics libraries insta
lled
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [7]:

```
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will li
st the files in the input directory

import os

# Any results you write to the current directory are saved as output.
data=pd.read_csv('train.csv')
data_test = pd.read_csv('test.csv')
rich_features = pd.concat([data[['Fare', 'Pclass', 'Age']],
                           pd.get_dummies(data['Sex'], prefix='Sex'),
                           pd.get_dummies(data['Embarked'], prefix='Embarked'
)],
                          axis=1)
```

In [8]:

```
rich_features_no_male = rich_features.drop('Sex_male', 1)
rich_features_final = rich_features_no_male.fillna(rich_features_no_male.drop
na().median())
survived_column = data['Survived']
target = survived_column.values



rich_features_test = pd.concat([data_test[['Fare', 'Pclass', 'Age']],
                           pd.get_dummies(data_test['Sex'], prefix='Sex'),
```

```
                            pd.get_dummies(data_test['Embarked'], prefix='Emba
rked')],
                           axis=1)

rich_features_no_male_test = rich_features_test.drop('Sex_male', 1)
rich_features_final_test = rich_features_no_male_test.fillna(rich_features_no
_male_test.dropna().median())
#Non Linear Model Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,subsample
=.8, max_features=.5)
```

In [9]:

```
#scores = cross_val_score(gb, rich_features_final, target, cv=5, n_jobs=4,sco
ring='accuracy')
#print("Gradient Boosted Trees CV scores:")
#print("min: {:.3f}, mean: {:.3f}, max: {:.3f}".format(scores.min(), scores.m
ean(), scores.max()))

gb.fit(rich_features_final, survived_column)
```

Out[9]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                            learning_rate=0.1, loss='deviance', max_depth=3,
                            max_features=0.5, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None
,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_iter_no_change=None, presort='auto',
                            random_state=None, subsample=0.8, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False)
```

In [11]:

```
predsGB = gb.predict(rich_features_final_test)
predsGB
```

Out[11]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
```

```
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0])
```

In [12]:

```
my_submissionGB = pd.DataFrame({'PassengerId': data_test['PassengerId'], 'Sur
vived': predsGB})
my_submissionGB.to_csv(f'submissionGB.csv', index=False)
```

Extra Tree Classifier

```
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
```

In [2]:

```
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will li
st the files in the input directory

import os

# Any results you write to the current directory are saved as output.
train=pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
combine = [train, test]
```

In [3]:

```
#Method for finding substrings
def substrings_in_string(big_string, substrings):
    for substring in substrings:
        if substring in big_string:
            return substring
    return np.nan
```

In [4]:

```
#Mappings
title_list=['Mrs', 'Mr', 'Master', 'Miss', 'Major', 'Rev',
                    'Dr', 'Ms', 'Mlle','Col', 'Capt', 'Mme', 'Countess',
                    'Don', 'Jonkheer']
```

```python
cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'T', 'G', 'Unknown']

title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for df in combine:
    # Convert the male and female groups to integer form
    df["Sex"][df["Sex"] == "male"] = 0
    df["Sex"][df["Sex"] == "female"] = 1

    #Map and Create Title Feature
    df['Title'] = df['Name'].astype(str).map(lambda x: substrings_in_string(x
, title_list))
    df['Title'] = df['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
        'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    df['Title'] = df['Title'].replace('Mlle', 'Miss')
    df['Title'] = df['Title'].replace('Ms', 'Miss')
    df['Title'] = df['Title'].replace('Mme', 'Mrs')
    df['Title'] = df['Title'].map(title_mapping)
    df['Title'] = df['Title'].fillna(0)
    #Map and Create Deck feature
    df['Deck'] = df['Cabin'].astype(str).map(lambda x: substrings_in_string(x
, cabin_list))
    df["Deck"][df["Deck"] == "A"] = 1
    df["Deck"][df["Deck"] == "B"] = 2
    df["Deck"][df["Deck"] == "C"] = 3
    df["Deck"][df["Deck"] == "D"] = 4
    df["Deck"][df["Deck"] == "E"] = 5
    df["Deck"][df["Deck"] == "F"] = 6
    df["Deck"][df["Deck"] == "G"] = 7
    df["Deck"][df["Deck"] == "T"] = 8
    df["Deck"] = df["Deck"].fillna(0)

    #Create Family size, Fare per person, and isAlone features
    df['Family_size'] = df['SibSp']+df['Parch']+1

    df['Fare_Per_Person']=df['Fare']/(df['Family_size'])

    df['isAlone']=0
    df.loc[df['Family_size']==1, 'isAlone'] = 1

    # Impute the Embarked variable to the mode
    df["Embarked"] = df["Embarked"].fillna("S")
    # Convert the Embarked classes to integer form
    df["Embarked"][df["Embarked"] == "S"] = 0
```

```
    df["Embarked"][df["Embarked"] == "C"] = 1
    df["Embarked"][df["Embarked"] == "Q"] = 2
```

```
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [5]:

```
#Impute ages based off sex and class
guess_ages = np.zeros((2,3))
for df in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = df[(df['Sex'] == i) & \
                                  (df['Pclass'] == j+1)]['Age'].dropna()
            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
        for j in range(0, 3):
            df.loc[ (df.Age.isnull()) & (df.Sex == i) & (df.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]

    df['Age'] = df['Age'].astype(int)

for df in combine:
    #set child feature
    df["child"] = float('NaN')
    df["child"][df["Age"] < 18] = 1
    df["child"][df["Age"] >=18] = 0
```

```
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
/Users/nicholasbergeland/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

In [6]:

```
#Set single null fare in test data
test['Fare'] = test['Fare'].fillna(0)
test['Fare_Per_Person'] = test['Fare_Per_Person'].fillna(0)
```

In [7]:

```
#Create target feature set
excl = ['PassengerId', 'Survived', 'Ticket', 'Cabin', 'Name']
cols = [c for c in train.columns if c not in excl]


target = train["Survived"].values
features = train[cols].values
```

In [8]:

```
#Extra Trees Classifier
etc = ExtraTreesClassifier(n_estimators=1000, max_depth=9, min_samples_split=6, min_samples_leaf=4, n_jobs=-1, random_state=10, verbose=0)
etcmod = etc.fit(features, target)
```

In [9]:

```
#Show feature importances
fi = etcmod.feature_importances_
importances = pd.DataFrame(fi, columns = ['importance'])
importances['feature'] = cols
print(importances.sort_values(by='importance', ascending=False))
```

```
    importance          feature
1     0.410548              Sex
7     0.169475            Title
0     0.143338           Pclass
8     0.049355             Deck
9     0.035726      Family_size
10    0.028768  Fare_Per_Person
5     0.028147             Fare
11    0.027211          isAlone
3     0.027077            SibSp
12    0.023895            child
2     0.022275              Age
6     0.021317         Embarked
4     0.012869            Parch
```

In [10]:

```
#Predict test
test_features = test[cols].values
pred = etcmod.predict(test_features)
```

```python
PassengerId = np.array(test["PassengerId"]).astype(int)
my_solution = pd.DataFrame(pred, PassengerId, columns = ["Survived"])

my_solution.to_csv("extraTrees.csv", index_label = ["PassengerId"])
```

In [11]:

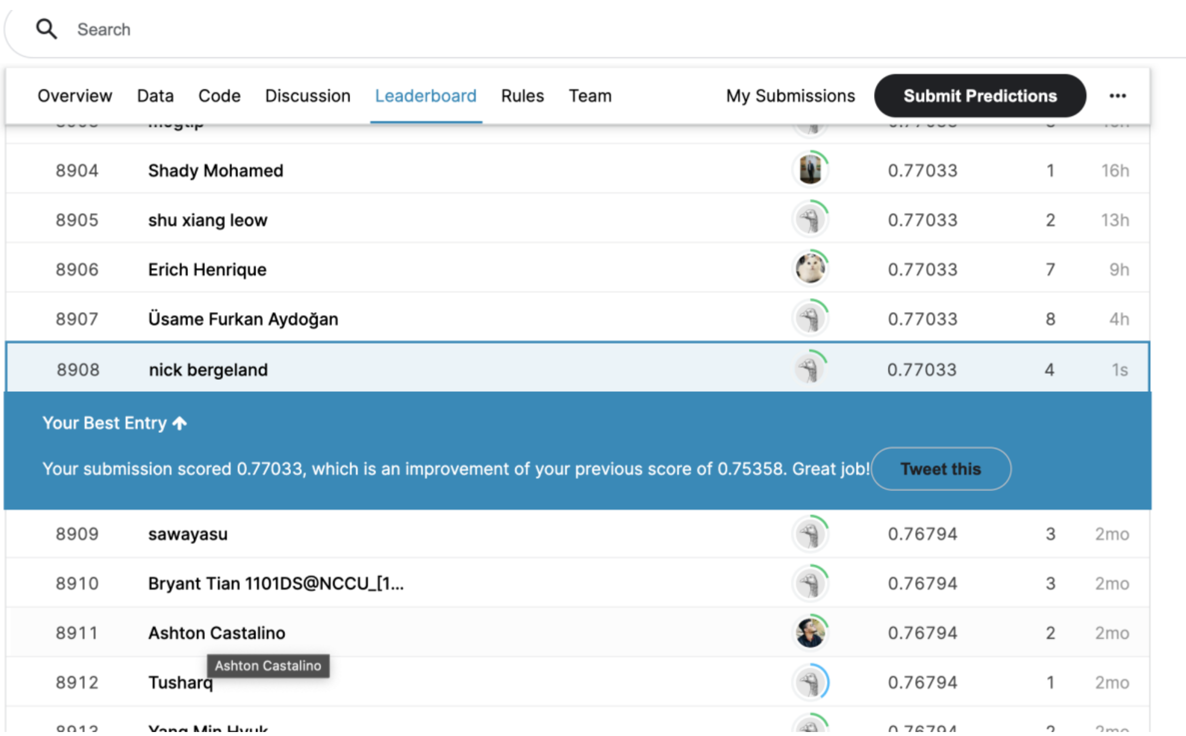
```python
my_solution
```

Out[11]:

	Survived
892	0
893	0
894	0
895	0
896	1
...	...
1305	0
1306	1
1307	0
1308	0
1309	1

418 rows × 1 columns

Kaggle Score



Works Cited:

1. Arshid. “Titanic: Machine Learning from Disaster Solution.” Kaggle. Kaggle, March 22, 2018. https://www.kaggle.com/arshid/titanic-machine-learning-from-disaster-solution/notebook.
2. Ladduci. “Titanic Gradient Booster.” Kaggle. Kaggle, May 20, 2019. https://www.kaggle.com/ladduci/titanic-gradient-booster.
3. Casselas, R. “Titanic Extra Trees.” Accessed February 6, 2022. https://www.kaggle.com/rcasellas/titanic-extra-trees-classifierß.