EDA and Prediction on Titanic DataSet

Import Libraries

Let's import some libraries to get started!

```
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.

```
train = pd.read_csv('titanic_train.csv')
```

```
train.head()
```

	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

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!

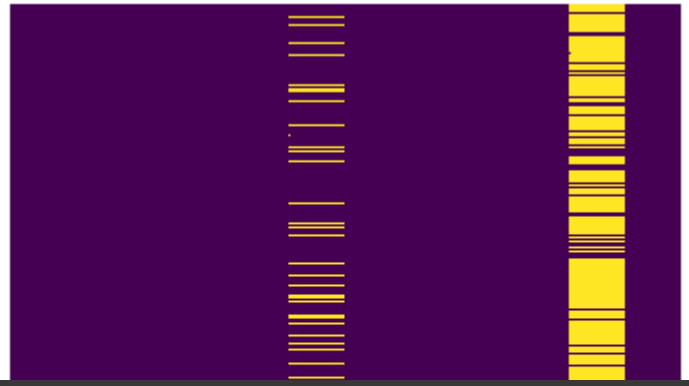
```
1 train.isnull()
```

	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

```
1 sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f9af68fd0>
```

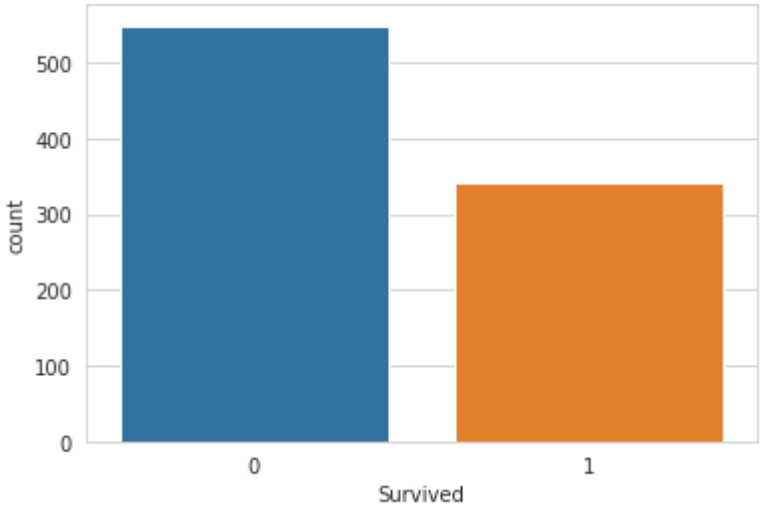


Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably 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! Check out the video for full explanations over these plots, this code is just to serve as reference.

```
1 sns.set_style('whitegrid')
2 sns.countplot(x='Survived',data=train)
```

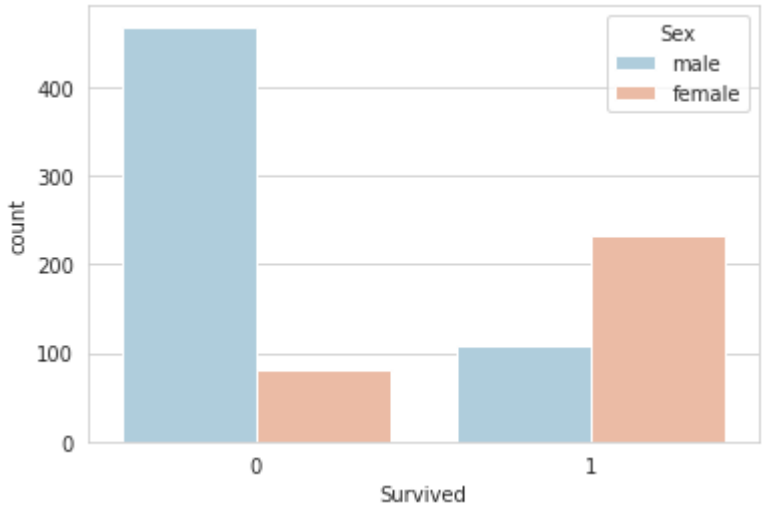
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f9868c6d0>
```




```
1 sns.set_style('whitegrid')
```

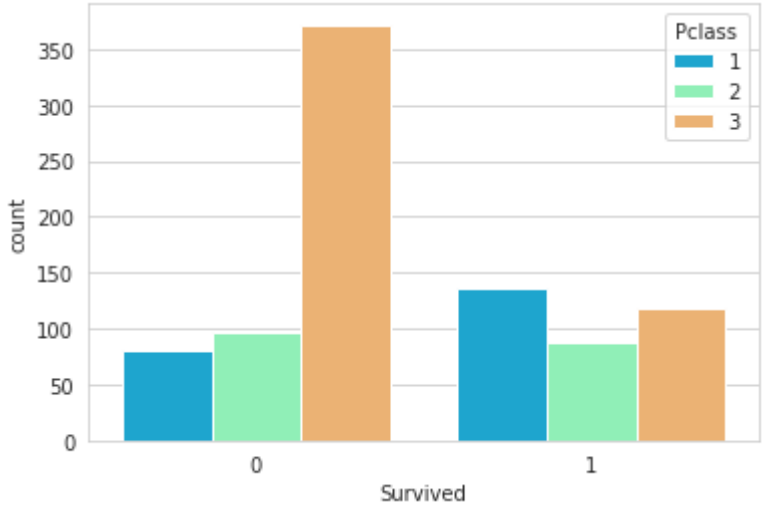
```
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f981ba150>
```



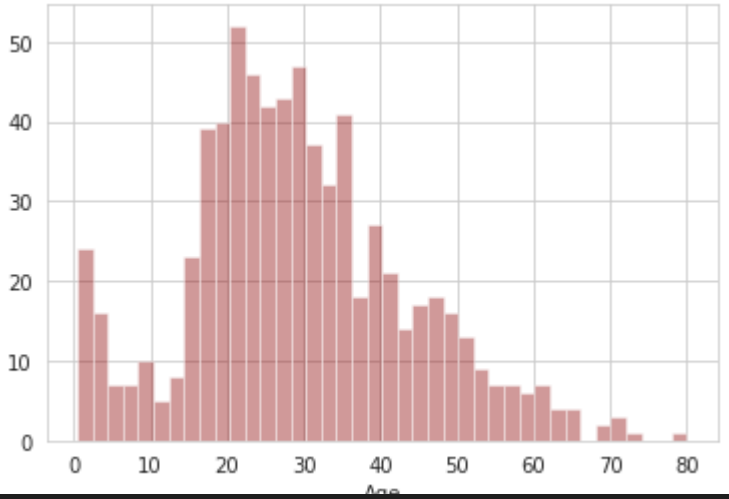

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f9812a210>
```



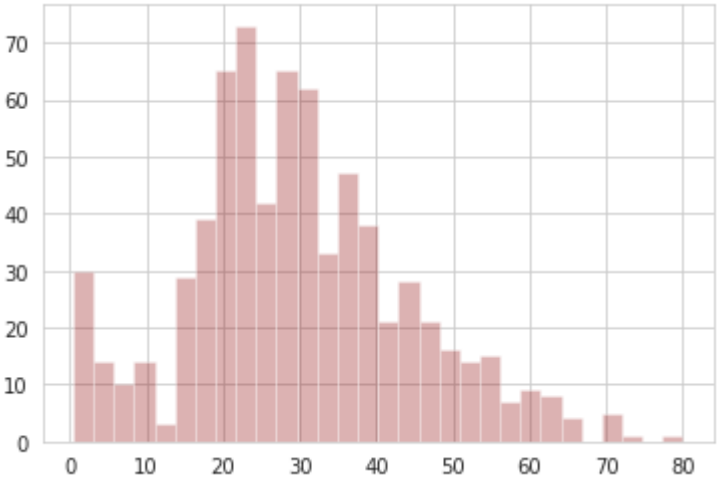

```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=40)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be remo
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f98075b90>
```



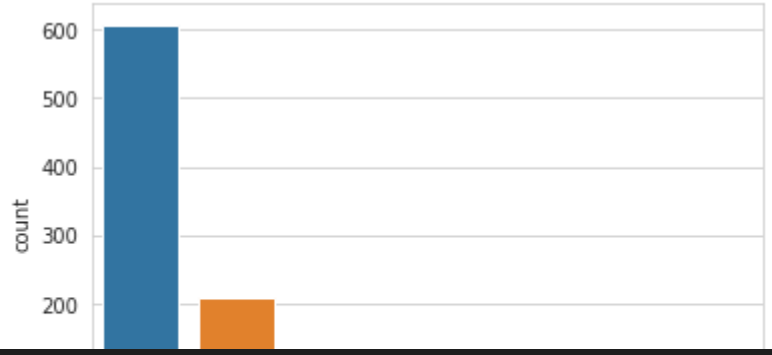

```
1 train['Age'].hist(bins=30,color='darkred',alpha=0.3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f97f2c310>
```



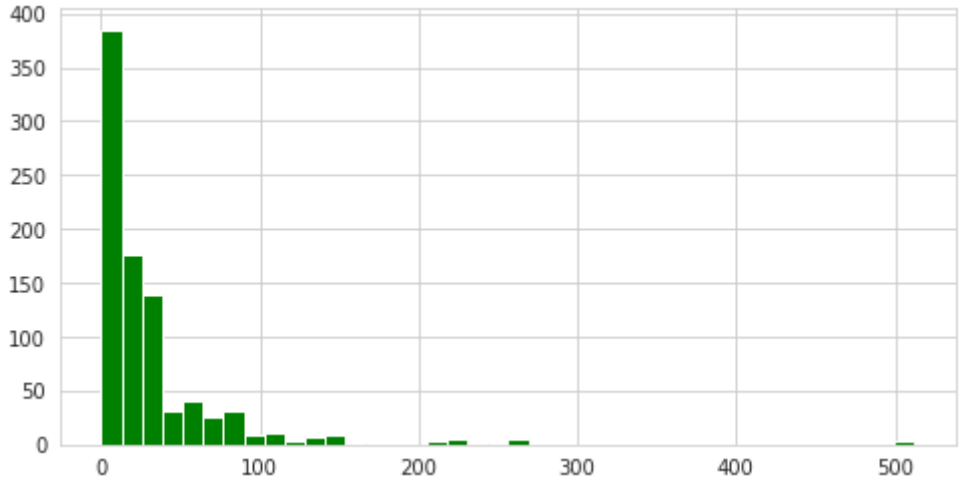

```
1 sns.countplot(x='SibSp',data=train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f97e7cdd0>
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f97df50d0>
```

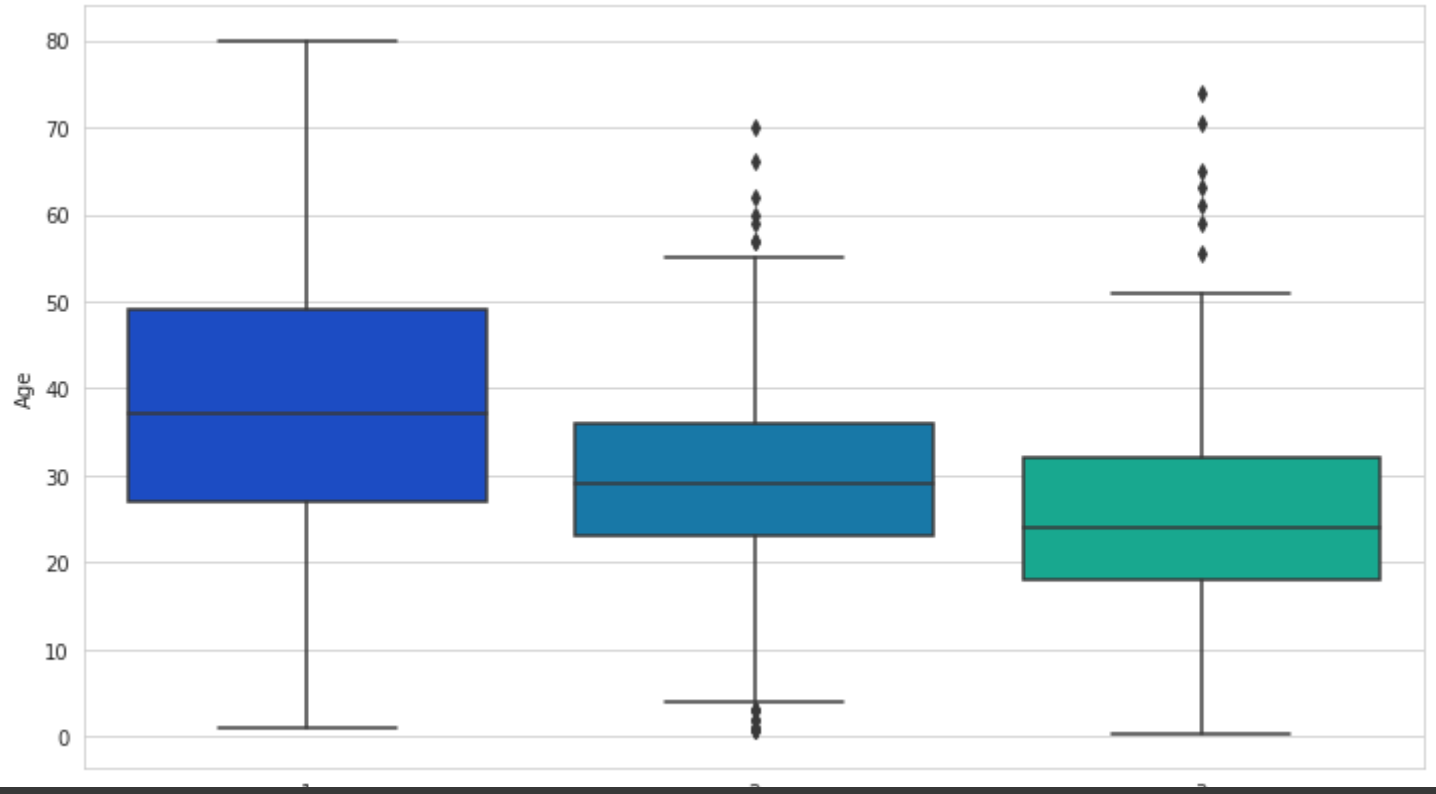


Data Cleaning

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:

```
1 plt.figure(figsize=(12, 7))
2 sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f97ced810>
```



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]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24
```

```
15
16     else:
17         return Age
```

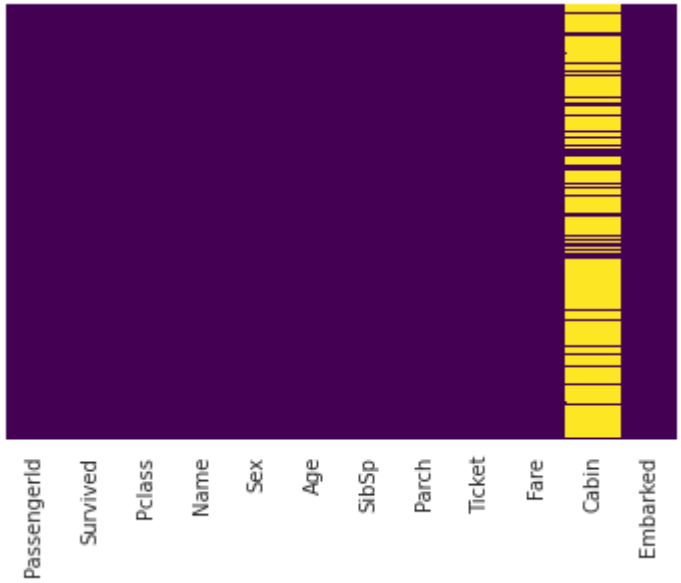
Now apply that function!

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

Now let's check that heat map again!

```
1 sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f97c8d450>
```



Filling embarked with mode

```
1 def fill_Embarked(df):
2     df.Embarked.fillna('S',inplace=True)
3     return df
```

```
train = fill_Embarked(train)

```

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

```
train.drop('Cabin', axis=1, inplace=True)

```

```
train.head()
```

	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

```
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.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 11 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          891 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  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.5+ KB
```

```
pd.get_dummies(train['Embarked'],drop_first=True).head()
```

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

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

```
train.head()
```

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

```
1 train.head()
```

	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	7.2500	S	1	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	0	0	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	0	0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	0	0	1

```
1 train.drop(['PassengerId','Sex', 'Embarked','Name', 'Ticket'], axis=1, inplace=True)
2
```

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

```
1 train.drop('Survived',axis=1).head()
```

	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	3	22.0	1	0	7.2500	1	0	1
1	1	38.0	1	0	71.2833	0	0	0

```
train['Survived'].head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
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

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: 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
    extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
```

Let's move on to evaluate our model!

Evaluation

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

```
def evaluationOnTest(model):
    predictions = model.predict(X_test)
    confusionMat = confusion_matrix(y_test, predictions)
    accuracy = accuracy_score(y_test, predictions)
    print('confusion matrix')
    print(confusionMat)

    print('Accuracy')
    print(accuracy)

    print('Classification Report')
    print(classification_report(y_test, predictions))


```

logistic regression evaluation

```
evaluationOnTest(logmodel)

```

```
confusion matrix
[[135  19]
 [ 37  77]]
Accuracy
0.7910447761194029
Classification Report
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       154
           1       0.80      0.68      0.73       114

    accuracy                           0.79       268
   macro avg       0.79      0.78      0.78       268
weighted avg       0.79      0.79      0.79       268
```

let's try some other classifier which may increase accuracy

```
from sklearn.ensemble import RandomForestClassifier

```

```
regmodel = RandomForestClassifier(max_depth=40, random_state=10, criterion='entropy', n_estimators=200)
regmodel.fit(X_train, y_train)

```

```
RandomForestClassifier(criterion='entropy', max_depth=40, n_estimators=200,
                       random_state=10)
```

Random forest evaluation

```
evaluationOnTest(regmodel)

```

```
confusion matrix
[[134  20]
 [ 32  82]]
Accuracy
0.8059701492537313
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       154
           1       0.80      0.72      0.76       114

    accuracy                           0.81       268
```

```
       macro avg       0.81      0.79      0.80       268
    weighted avg       0.81      0.81      0.80       268
```

```
1
```