O - Obtaining Data

S - Scrubbing / Cleaning our data

E - Exploring / Visualizing our data will allow us to find patterns and trends

M - Modeling our data will give us our predictive power as a wizard

N - INterpreting our data

Labels in both KNN and NB of where each stage takes place

KNN

CSV Handling (O,S)

Take the 3 different CSV files and store them into variables and format them

In [2]:
```
from mlxtend.plotting import plot_decision_regions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

genderData = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classification 
testData = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classification La
trainData = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classification L
```

In [3]:
```
trainData = trainData.set_index("PassengerId")
trainData = trainData.drop(["Name","Cabin","Embarked","Ticket"], axis = 1)
trainData[['Sex']] = trainData[['Sex']].replace('male',1)
trainData[['Sex']] = trainData[['Sex']].replace('female',0)
trainData = trainData.fillna(-1)

genderData = genderData.set_index("PassengerId")

testData = testData.set_index("PassengerId")
testData = testData.drop(["Name","Cabin","Embarked","Ticket"], axis = 1)
testData[['Sex']] = testData[['Sex']].replace('male',1)
testData[['Sex']] = testData[['Sex']].replace('female',0)
testData = testData.fillna(-1)
```

In [4]:
```
survivedData = trainData.Survived
newTrainData = trainData.drop("Survived", axis = 1)
```

Scaling and Cross Validaiton (Library and Diabetes Way) (E, M)

Scaling and Splitting my data

In [5]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
newTrainData = scaler.fit_transform(newTrainData)
testData = scaler.transform(testData)
```

In [6]:
```
#the purpose of train test split is to be testing on random data points rath
# We stratify by the survived data so that the data that we are working with
# If the overall survived data is 40% death and 60% survived, then every sec
# It is good to do this do prevent overfitting/underfitting
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(newTrainData, survivedDa
print(X_train.shape, y_train.shape)
print(testData.shape, genderData.shape)
```

```
(712, 6) (712,)
(418, 6) (418, 1)
```

Cross validation the Source's Way, and then the KNeighborsClassifier way

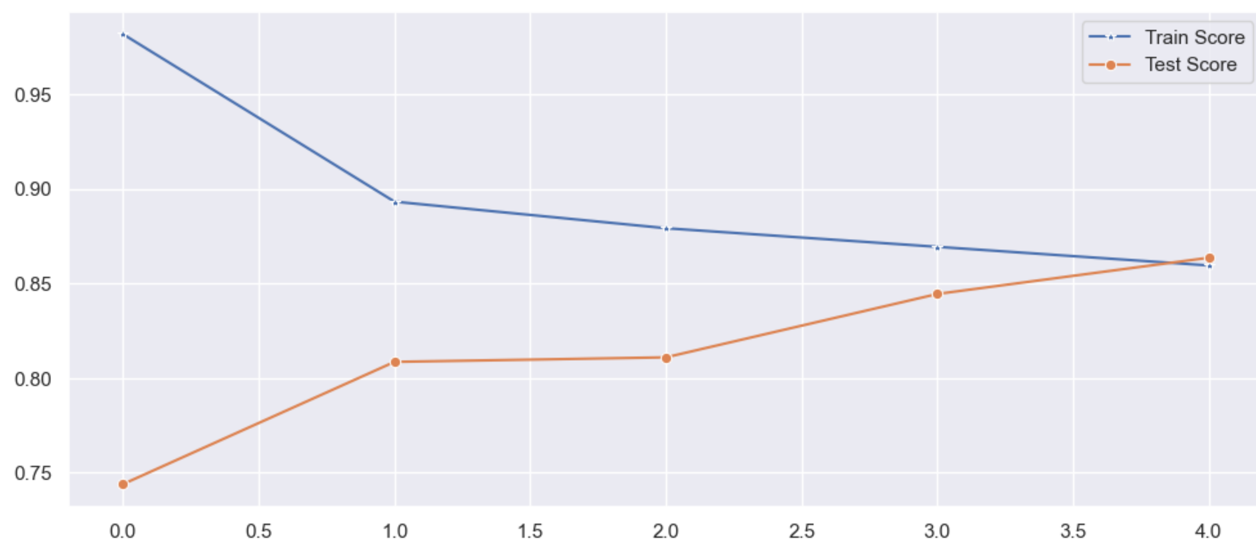
In [7]:
```
#SOURCE: https://www.statology.org/k-fold-cross-validation-in-python/#:~:tex

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt

#define cross-validation method to use
cv = KFold(n_splits=5, random_state=1, shuffle=True)

#builds multiple linear regression model
model = LinearRegression()

#uses k-fold CV to evaluate model
scores = cross_val_score(model, newTrainData, survivedData, scoring='neg_mea
                         cv=cv, n_jobs=-1)

#view RMSE
sqrt(mean(absolute(scores)))

#The lower the number (RMSE), the close it is to being accurate
```

Out[7]:
```
0.5457076265957654
```

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier


test_scores = []
train_scores = []

for i in range(1,6):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(testData,genderData))

max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_
print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(

max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_sco
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(la

# Makes i different classifiers and scores the train and test data each time
# It creates a new knn variables with i number of neighbors, 1 through 5
# When making a new knn, the number of neighbors is how many groups the data
# Breaking the data into different sets of data is how we can handle overfit
# Once it has walked through the range (cross validated a total of 5 times,
```

```
Max train score 98.17415730337079 % and k = [1]
Max test score 86.36363636363636 % and k = [5]
```

In [9]:
```
plt.figure(figsize=(12,5))
p = sns.lineplot(train_scores,marker='*',label='Train Score')
p = sns.lineplot(test_scores,marker='o',label='Test Score')
```



Finding the best split and scoring the data to that split

In [10]:
```
knn = KNeighborsClassifier(5)

knn.fit(newTrainData,survivedData)
knn.score(testData,genderData)

#Xtest = testData
#ytest = genderData
```

Out[10]:
```
0.8325358851674641
```

In [11]:
```
# Only suppposed to be run once

genderData = genderData.reset_index(drop=True).to_numpy()
genderData = np.resize(genderData, [418,])
```

Results (N)

Going to analyze my each graph here

Confusion Matrix: my confusion matrix shows that my trained data prediction was accuarate because the bulk of the numbers are in the PredictedNo X ActualNO position (true negative TN) and PredictedYes X ActualYes (true positive TP) having the data predominantly in these two positions shows that my model was good

Precision Score: The precision score futher analyzes the data from the confusion matrix The definitin of precision is the accuracy of positive predictions The formula is Precision = TP/(TP + FP) Recall Score: The fraction of positives that were correctly identified A recall greater than 0.5 is good Recall = TP/(TP + FN) F1 Score: Created by finding the weighted average of the precision and recall F1 = 2 * (Precision * Recall) / (Precision + Recall)

Receiver Operating Characteristic (ROC) tells about how good the model can distinguish whether or not someone lived or died Better models do better dinstinguishing, bad models don't A model is seen to be good the faster it gets to a high tpr from a low fpr The closer the classifier line is to the dashed line, the poorer the classifier Our ROC rises fast and isn't closed to the dashed line, so it is understood to be good this claim is supported by AOC being 0.9 (close to 1 is good, close to 0.5 is bad

Making the confusion matrix

In [13]:
```
#import confusion_matrix
from sklearn.metrics import confusion_matrix
#let us get the predictions using the classifier we had fit above
y_pred = knn.predict(testData)
confusion_matrix(genderData,y_pred)
pd.crosstab(genderData, y_pred, rownames=['True'], colnames=['Predicted'], m
```

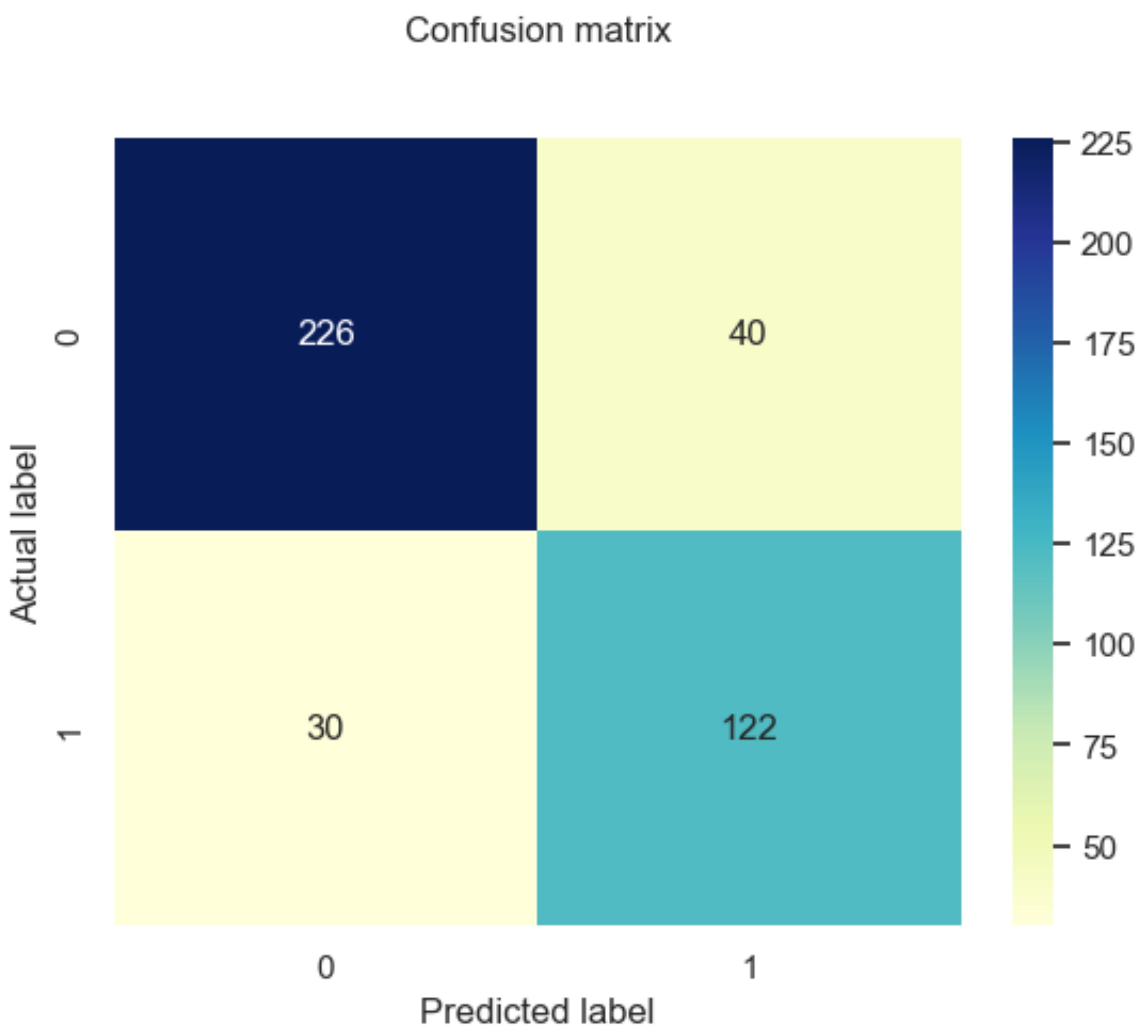
Out[13]:

Predicted	0	1	All
True			
0	226	40	266
1	30	122	152
All	256	162	418

Making a more detailed confusion matrix

In [14]:
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(genderData, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g'
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[14]:
```
Text(0.5, 20.049999999999997, 'Predicted label')
```



Precision Recall F1-Score

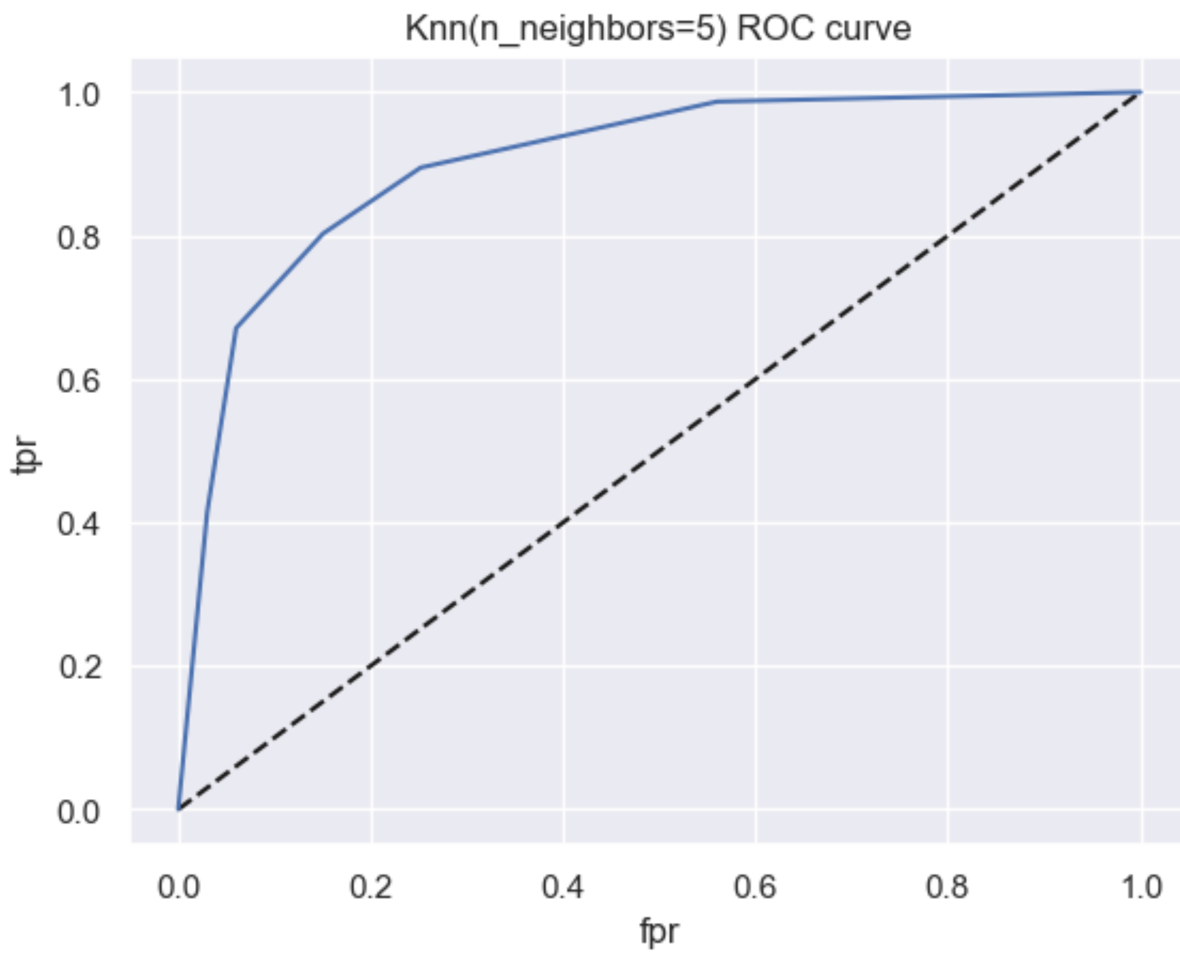
In [15]:
```
#import classification_report
from sklearn.metrics import classification_report
print(classification_report(genderData,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.85      0.87       266
           1       0.75      0.80      0.78       152

    accuracy                           0.83       418
   macro avg       0.82      0.83      0.82       418
weighted avg       0.84      0.83      0.83       418
```

Receiver Operating Characteristic curve and Area under the curve

In [16]:
```
from sklearn.metrics import roc_curve
y_pred_proba = knn.predict_proba(testData)[:,1]
fpr, tpr, thresholds = roc_curve(genderData, y_pred_proba)
```

In [17]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=5) ROC curve')
plt.show()
```



In [18]:
```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(genderData,y_pred_proba)
```

Out[18]: 0.9021567075583696

NB

CSV Handling (O,S)

Take the 3 different CSV files and store them into variables and format them

In [19]:
```
from mlxtend.plotting import plot_decision_regions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

genderDataNB = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classificatio
testDataNB = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classification
trainDataNB = pd.read_csv('/Users/lukehenry/Documents/Jupyter/Classification

trainDataNB = trainDataNB.set_index("PassengerId")
trainDataNB = trainDataNB.drop(["Name","Cabin","Ticket"], axis = 1)
trainDataNB[['Sex']] = trainDataNB[['Sex']].replace('male',1)
trainDataNB[['Sex']] = trainDataNB[['Sex']].replace('female',0)
trainDataNB['Embarked'] = trainDataNB['Embarked'].replace(['S', 'C', 'Q'], [
#trainDataNB = trainDataNB.dropna()
trainDataNB = trainDataNB.fillna(0)

testDataNB = testDataNB.set_index("PassengerId")
testDataNB = testDataNB.drop(["Name","Cabin","Ticket"], axis = 1)
testDataNB[['Sex']] = testDataNB[['Sex']].replace('male',1)
testDataNB[['Sex']] = testDataNB[['Sex']].replace('female',0)
testDataNB['Embarked'] = testDataNB['Embarked'].replace(['S', 'C', 'Q'], [1,
#testDataNB = testDataNB.dropna()
testDataNB = testDataNB.fillna(0)


genderDataNB = genderDataNB.dropna()
genderDataNB = genderDataNB.set_index("PassengerId")

# survivedDataNB = trainDataNB[["Survived"]]
survivedDataNB = trainDataNB.Survived
trainDataNB = trainDataNB.drop("Survived", axis = 1)
```

Splitting the data (E)

Splitting the data into train and validaiton for X and Y

In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_validation, Y_train, Y_validation = train_test_split(trainDataNB,
print(X_train.shape, X_validation.shape)
print(Y_train.shape, Y_validation.shape)
```

```
(712, 7) (179, 7)
(712,) (179,)
```

Breaking down the train data into 2 parts

In [21]:
```
from sklearn.model_selection import train_test_split
X_train1, X_train2, Y_train1, Y_train2 = train_test_split(X_train, Y_train,
print(X_train1.shape, X_train2.shape)
print(Y_train1.shape, Y_train2.shape)
```

```
(498, 7) (214, 7)
(498,) (214,)
```

Classifier and Cross Validation (M)

Creating my Gaussian classifier and fitting it

In [22]:
```
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()
classifier.fit(X_train2, Y_train2)
```

Out[22]:
```
▾ GaussianNB
GaussianNB()
```

Cross validating 5 times with 30% of the train data

In [23]:
```
from sklearn.model_selection import cross_validate

print('Metrics with only 30% of train data')
cross_validate(classifier, trainDataNB, survivedDataNB, scoring='neg_mean_ab
                          cv=5, n_jobs=1)
# cross_validate(classifier, (X_train, Y_train), (X_validation, Y_validatior
```

```
Metrics with only 30% of train data
```

Out[23]:
```
{'fit_time': array([0.00294709, 0.00312114, 0.00294018, 0.00079632, 0.00069
666]),
 'score_time': array([0.0051918 , 0.0015471 , 0.00063705, 0.00043488, 0.000
41127]),
 'test_score': array([-0.24022346, -0.21910112, -0.20224719, -0.21348315, -
0.19662921])}
```

Refitting my classifier to a partial fit of the subsetted data

In [24]:
```
classifier = classifier.partial_fit(X_train1, Y_train1)
```

Cross validating 5 times with 70% of the train data

In [25]:
```
from sklearn.model_selection import cross_validate
from numpy import mean
from numpy import absolute
from numpy import sqrt
print('Metrics with the remaining 70% of train data')
# cross_validate(classifier, (X_train, Y_train), (X_validation, Y_validation

cross_validate(classifier, trainDataNB, survivedDataNB, scoring='neg_mean_ab
                         cv=5, n_jobs=1)
```

```
Metrics with the remaining 70% of train data
```

Out[25]:
```
{'fit_time': array([0.00337911, 0.00273466, 0.00227118, 0.00086188, 0.00070
906]),
 'score_time': array([0.00256777, 0.00479698, 0.00057697, 0.00044608, 0.000
42701]),
 'test_score': array([-0.24022346, -0.21910112, -0.20224719, -0.21348315, -
0.19662921])}
```

Results (N)

Going to analyze my each graph here Our data for the NB classifier is showing to be more accurate than the k-NN

Confusion Matrix: my confusion matrix shows that my trained data prediction was VERY accuarate because the bulk of the numbers are in the PredictedNo X ActualNO position (true negative TN) and PredictedYes X ActualYes (true positive TP) having the data predominantly in these two positions shows that my model was good

Precision Score: The precision score futher analyzes the data from the confusion matrix The definitin of precision is the accuracy of positive predictions The formula is Precision = TP/(TP + FP) Recall Score: The fraction of positives that were correctly identified A recall greater than 0.5 is good Recall = TP/(TP + FN) F1 Score: Created by finding the weighted average of the precision and recall F1 = 2 * (Precision * Recall) / (Precision + Recall)

Receiver Operating Characteristic (ROC) tells about how good the model can distinguish whether or not someone lived or died Better models do better dinstinguishing, bad models don't A model is seen to be good the faster it gets to a high tpr from a low fpr The closer the classifier line is to the dashed line, the poorer the classifier Our ROC rises fast and isn't closed to the dashed line, so it is understood to be good this claim is supported by AOC being 0.91 (close to 1 is good, close to 0.5 is bad

Handling genderDataNB so that it will work properly with the graphs

In [27]:
```
genderDataNB = genderDataNB.reset_index(drop=True).to_numpy()
genderDataNB = np.resize(genderDataNB, [418,])
```

Making the confusion matrix

In [28]:
```
#import confusion_matrix
from sklearn.metrics import confusion_matrix
#let us get the predictions using the classifier we had fit above
y_pred = classifier.predict(testDataNB)
confusion_matrix(genderDataNB,y_pred)
pd.crosstab(genderDataNB, y_pred, rownames=['True'], colnames=['Predicted'],
```

Out[28]:

Predicted	0	1	All
True			
0	231	35	266
1	8	144	152
All	239	179	418

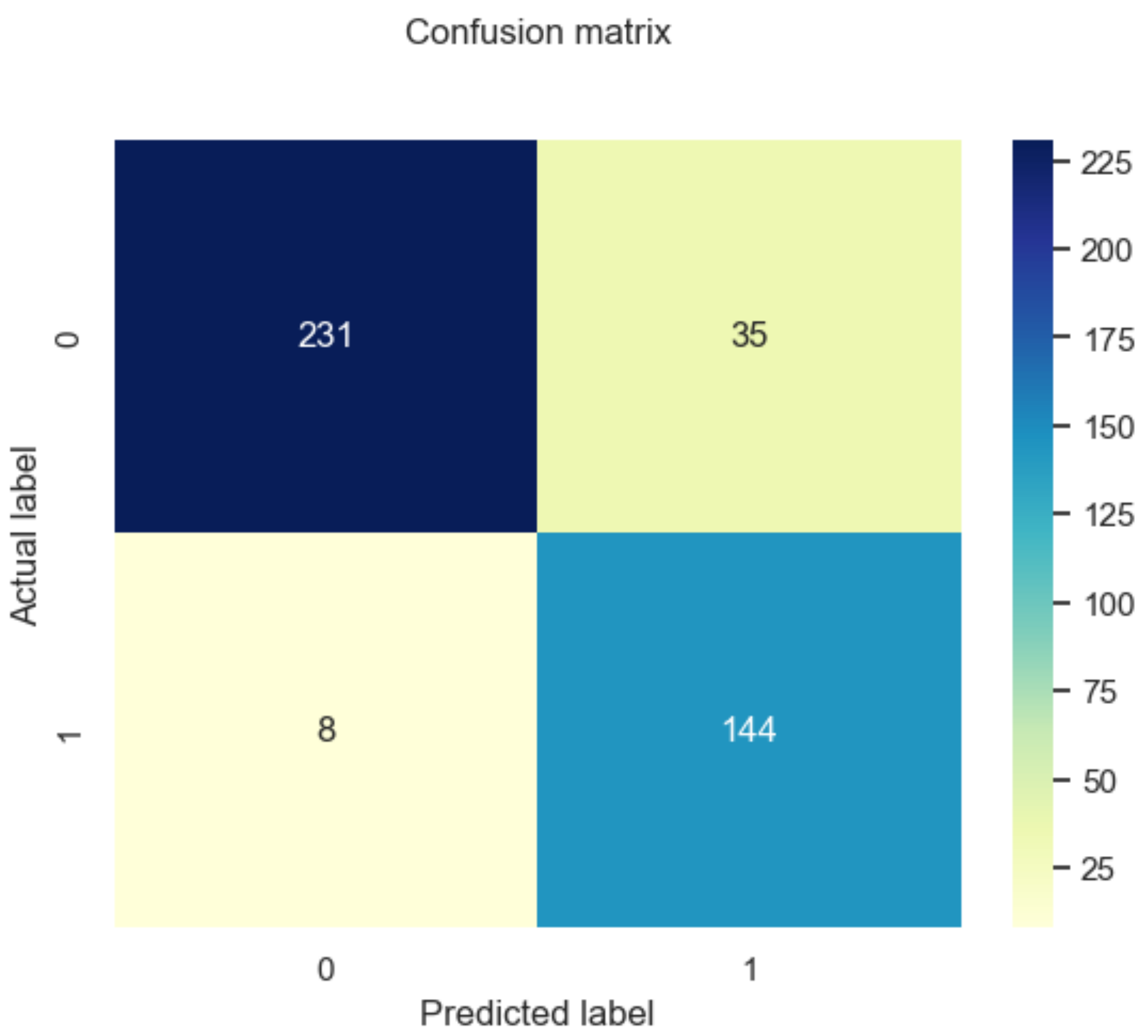
Making a more detailed confusion matrix

In [29]:

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(genderDataNB, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g'
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[29]:

```
Text(0.5, 20.049999999999997, 'Predicted label')
```



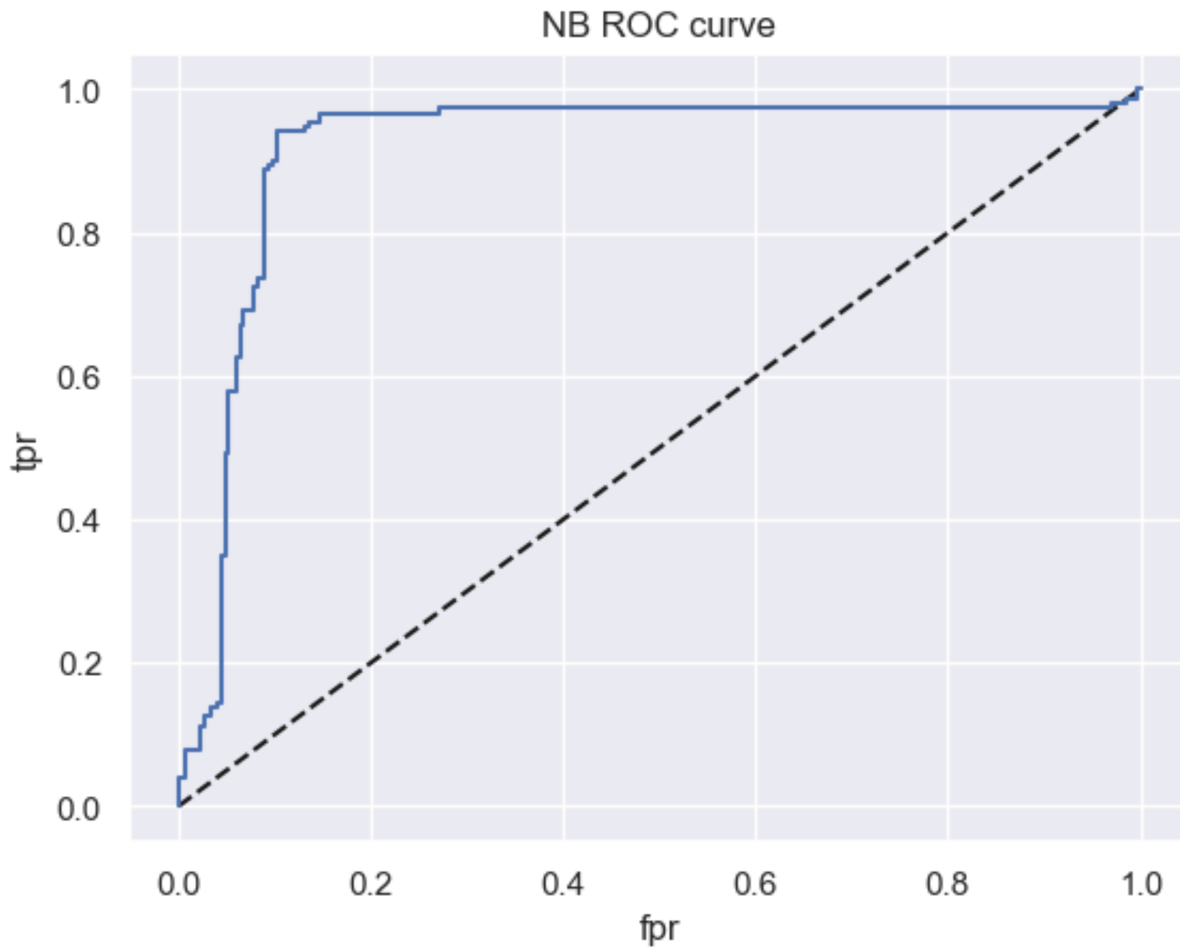
Precision Recall F1-Score

In [30]:
```
#import classification_report
from sklearn.metrics import classification_report
print(classification_report(genderDataNB,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.87      0.91       266
           1       0.80      0.95      0.87       152

    accuracy                           0.90       418
   macro avg       0.89      0.91      0.89       418
weighted avg       0.91      0.90      0.90       418
```

Rate of change curve and Area under the curve

In [31]:
```
from sklearn.metrics import roc_curve
y_pred_proba = classifier.predict_proba(testDataNB)[:,1]
fpr, tpr, thresholds = roc_curve(genderDataNB, y_pred_proba)

plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('NB ROC curve')
plt.show()
```



In [32]:
```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(genderDataNB,y_pred_proba)
```

Out[32]: 0.9160318559556787

In [ ]: