Coursework 2

Group 3: Carwyn Cowell, Yang Zhou, Nishal Dave, Ran Xu

In [12]:
```
import sklearn.linear_model as skl_lm
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import pandas as pd
import numpy as np

from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn import model_selection
from sklearn import preprocessing
from sklearn import neighbors


import statsmodels.formula.api as smf
import statsmodels.api as sm
%matplotlib inline
```

(1)

(a)

In [13]:
```
df = pd.read_csv('Heart.csv').iloc[:,1:]
df = df.dropna()
df.head()
```

Out[13]:

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
1	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

In [14]:

```
#https://archive.ics.uci.edu/ml/datasets/heart+disease - information source
#Create dummy variables for the categorical variables as new columns

#Male=1
df.rename(columns={"Sex": "Male"},inplace=True)

#Heart Disease (Dependent)
df.loc[:,"Heart Disease"] = [1 if x=="Yes" else 0 for x in df["AHD"]]
df.drop(columns="AHD", inplace=True)

#Chest Pain: Default = Typical Chest Pain
df.loc[:,"Asymptomatic Chest Pain"] = [1 if x=="asymptomatic" else 0 for x in df["ChestPain"]]
df.loc[:,"Nonanginal Chest Pain"]   = [1 if x=="nonanginal" else 0 for x in df["ChestPain"]]
df.loc[:,"Nontypical Chest Pain"]   = [1 if x=="nontypical" else 0 for x in df["ChestPain"]]
df.drop(columns="ChestPain",inplace=True)

#Thal defect: Default = Normal
df.loc[:,"Thal:Fixed"]      = [1 if x=="fixed" else 0 for x in df["Thal"]]
df.loc[:,"Thal:Reversable"] = [1 if x=="reversable" else 0 for x in df["Thal"]]
df.drop(columns="Thal",inplace=True)


#Resting ECG: Default = Normal
df.loc[:,"RECG:Abnormal"] = [1 if x==1 else 0 for x in df["RestECG"]]
df.loc[:,"RECG:LVH"]      = [1 if x==2 else 0 for x in df["RestECG"]]
df.drop(columns="RestECG",inplace=True)


#Slope of ST Segment: Default = Flat
df.loc[:,"Slope:Upsloping"]   = [1 if x==1 else 0 for x in df["Slope"]]
df.loc[:,"Slope:Downsloping"] = [1 if x==3 else 0 for x in df["Slope"]]
df.drop(columns="Slope",inplace=True)

#Number of Major Vessels: Default = 0
df.loc[:,"Ca: 1"] = [1 if x==1.0 else 0 for x in df["Ca"]]
df.loc[:,"Ca: 2"] = [1 if x==2.0 else 0 for x in df["Ca"]]
df.loc[:,"Ca: 3"] = [1 if x==3.0 else 0 for x in df["Ca"]]
df.drop(columns="Ca",inplace=True)


#Standardizing continuous variables
VartoStdze = ["Age","RestBP","Chol","MaxHR","Oldpeak"]
```

```
scale = StandardScaler()
for i in VartoStdze:
    df[i]=scale.fit_transform(df[i].values.reshape(-1,1))

df.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
Age	297.0	-1.237319e-16	1.001688	-2.827176	-0.724124	0.161372	0.714807	2.485798
Male	297.0	6.767677e-01	0.468500	0.000000	0.000000	1.000000	1.000000	1.000000
RestBP	297.0	4.810966e-16	1.001688	-2.125634	-0.659431	-0.095506	0.468418	3.851964
Chol	297.0	-1.911116e-16	1.001688	-2.337704	-0.700254	-0.083802	0.551914	6.099981
Fbs	297.0	1.447811e-01	0.352474	0.000000	0.000000	0.000000	0.000000	1.000000
MaxHR	297.0	5.143660e-16	1.001688	-3.431849	-0.724769	0.148482	0.716096	2.287949
ExAng	297.0	3.265993e-01	0.469761	0.000000	0.000000	0.000000	1.000000	1.000000
Oldpeak	297.0	-1.334511e-16	1.001688	-0.906712	-0.906712	-0.219520	0.467672	4.419026
Heart Disease	297.0	4.612795e-01	0.499340	0.000000	0.000000	0.000000	1.000000	1.000000
Asymptomatic Chest Pain	297.0	4.781145e-01	0.500364	0.000000	0.000000	0.000000	1.000000	1.000000
Nonanginal Chest Pain	297.0	2.794613e-01	0.449492	0.000000	0.000000	0.000000	1.000000	1.000000
Nontypical Chest Pain	297.0	1.649832e-01	0.371792	0.000000	0.000000	0.000000	0.000000	1.000000
Thal:Fixed	297.0	6.060606e-02	0.239009	0.000000	0.000000	0.000000	0.000000	1.000000
Thal:Reversable	297.0	3.872054e-01	0.487933	0.000000	0.000000	0.000000	1.000000	1.000000
RECG:Abnormal	297.0	1.346801e-02	0.115462	0.000000	0.000000	0.000000	0.000000	1.000000
RECG:LVH	297.0	4.915825e-01	0.500773	0.000000	0.000000	0.000000	1.000000	1.000000
Slope:Upsloping	297.0	4.680135e-01	0.499818	0.000000	0.000000	0.000000	1.000000	1.000000
Slope:Downsloping	297.0	7.070707e-02	0.256768	0.000000	0.000000	0.000000	0.000000	1.000000
Ca: 1	297.0	2.188552e-01	0.414168	0.000000	0.000000	0.000000	0.000000	1.000000
Ca: 2	297.0	1.279461e-01	0.334594	0.000000	0.000000	0.000000	0.000000	1.000000
Ca: 3	297.0	6.734007e-02	0.251033	0.000000	0.000000	0.000000	0.000000	1.000000

In [10]:
```
df['Male'].corr(df['Heart Disease'])
```

Out[10]:
```
0.2784666966537961
```

(b)

In [ ]:
```
X = df.drop(columns="Heart Disease")
y = df["Heart Disease"]
model = sm.Logit(y, sm.add_constant(X)).fit()
model.summary2().tables[1]
```

```
Optimization terminated successfully.
         Current function value: 0.308251
         Iterations 8
```

Out[ ]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-4.017949	0.850195	-4.725913	0.000002	-5.684302	-2.351597
Age	-0.212378	0.226961	-0.935747	0.349403	-0.657215	0.232458
Male	1.670152	0.552489	3.022957	0.002503	0.587293	2.753011
RestBP	0.491548	0.208329	2.359475	0.018301	0.083230	0.899866
Chol	0.230723	0.212353	1.086507	0.277255	-0.185481	0.646926
Fbs	-0.574079	0.592542	-0.968842	0.332624	-1.735439	0.587281
MaxHR	-0.451075	0.268354	-1.680898	0.092783	-0.977039	0.074888
ExAng	0.653306	0.447446	1.460077	0.144269	-0.223673	1.530285
Oldpeak	0.454812	0.278436	1.633451	0.102374	-0.090913	1.000537
Asymptomatic Chest Pain	2.373287	0.709097	3.346914	0.000817	0.983482	3.763092
Nonanginal Chest Pain	0.393353	0.700340	0.561660	0.574348	-0.979288	1.765994
Nontypical Chest Pain	1.448396	0.809140	1.790045	0.073447	-0.137488	3.034281
Thal:Fixed	-0.168439	0.810312	-0.207869	0.835331	-1.756622	1.419744
Thal:Reversable	1.433319	0.440569	3.253339	0.001141	0.569821	2.296818
RECG:Abnormal	1.000887	2.638430	0.379350	0.704428	-4.170340	6.172115
RECG:LVH	0.486408	0.396328	1.227287	0.219715	-0.290381	1.263197
Slope:Upsloping	-1.302289	0.486200	-2.678506	0.007395	-2.255222	-0.349355
Slope:Downsloping	-0.695528	0.872625	-0.797053	0.425420	-2.405843	1.014786
Ca: 1	2.237444	0.514773	4.346468	0.000014	1.228508	3.246380
Ca: 2	3.271852	0.785128	4.167286	0.000031	1.733030	4.810674
Ca: 3	2.188715	0.928648	2.356884	0.018429	0.368599	4.008832

The constant value is the largest in magnitude and the most significant. This indicates that individuals with all of the default characteristics, as defined above, have a relatively low

probability of heart disease.

Interestingly, age does not seem to be a statistically significant parameter in our model as it has a p-value of approximately 0.35. Thus, we are unable to conclude that age is an important indicator of heart disease.

Conversely, being male is highly significant, with a p-value of 0.002, and associated with a large increase in the likelihood of having a heart condition. This result is reassuring as it reflects the correlation calculated above.

Of similar statistical significance are the parameters for RestBP, MaxHR, Oldpeak, Asymptomatic Chest Pain, Nontypical Chest Pain, Thal: Reversable, Slope: Upsloping and all Ca indicator variables. All have p-values at or below 0.1. MaxHR and Slope: Upsloping seem to be associated with a decrease in the likelihood of having heart disease. Whereas all of the above variables increase the probability that the individual has a heart condition, this likely reflects our baseline being a relatively healthy individual thus deviations from this are typically bad.

However, there are also a number of other variables that are not of statistical significance as their p-values are larger than 0.1. Chol, Fbs, ExAng, NonAnginal Chest Pain, RECG: Abnormal, RECG: LVH and Slope: Downsloping are all of statistical insignificance to the prediction of heart disease.

(c)

As this is a medical analysis it is likely that we would want to choose a classification rule that focuses on reducing the missclassification of those that do have a heart condition as individuals that do not have a heart condition ie the false-negative rate as missing an underlying heart condition may prove fatal. To reduce the class specific rate we can reduce the threshold for which we classify someone as having a heart condition. In doing so we would make more false positives and tell people they have a heart condition when they do not but this is likely to be an acceptable sacrifice for a hospital especially if combined with further tests. We will use a classificaiton rule that states that if the probability of someone having a heart condition is greater than 20% we will classify them as having a heart condition.

(d)

In [ ]:
```
Xofs=pd.DataFrame(
    {"Male":0,
     "Age":55,
     "RestBP":130,
     "Chol":246,
     "Fbs":0,
     "MaxHR":150,
     "ExAng":1,
     "Oldpeak":1,
     "Asymptomatic Chest Pain":0,
     "Nonanginal Chest Pain":0,
     "Nontypical Chest Pain":0,
     "Thal:Fixed":0,
     "Thal:Reversable":0,
     "RECG:Abnormal":0,
     "RECG:LVH":1,
     "Slope:Upsloping":0,
     "Slope:Downsloping":0,
```

```
     "Ca: 1":0,
     "Ca: 2":0,
     "Ca: 3":0},
      index=[0]
                )
VartoStdze = ["Age","RestBP","Chol","MaxHR","Oldpeak"]
scale = StandardScaler()
for i in VartoStdze:
    Xofs[i]=scale.fit_transform(Xofs[i].values.reshape(-1,1))
```

In [ ]:
```
clf = skl_lm.LogisticRegression(solver='newton-cg')
clf.fit(X,y)
prob = clf.fit(X,y).predict_proba(Xofs)
print(prob)
```

```
[[0.85624912 0.14375088]]
```

Based on this result and the threshold determined above, we would classify this patient to not have heart disease as:

$pr(y = 1) = 0.14 < 0.2$

Therefore they do not cross the classification boundary.

(e)

Testing model accuracy requires evaluating the test error rate, in order to acquire this we need a testing sample, in this case, we have used all the observations to train the model. Although question (d) offers an observation, we do not know this patients true state to determine whether model classified her correctly or not.

(f)

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,train_size=207, random_state=10)
```

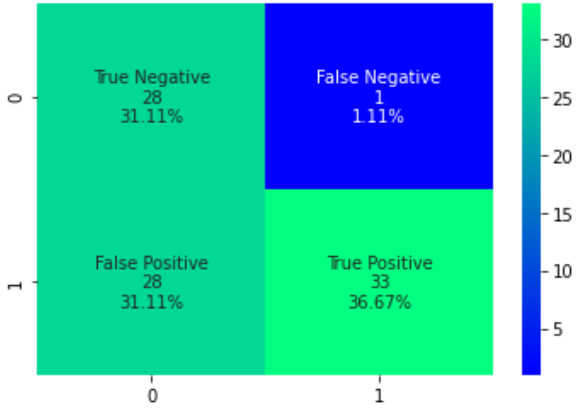
(g)

In [ ]:
```
#ESTIMATION
logit = skl_lm.LogisticRegression(solver='newton-cg')
logit.fit(X_train,y_train)
logitprob = logit.predict_proba(X_test)

#OUTPUT
testlogit=[]
for i in range(len(logitprob)):
    if (logitprob[i][1]>0.2)==True:
        testlogit.append(1)
    else:
        testlogit.append(0)
```

```
error_rate_logit1 = 1-accuracy_score(y_test, testlogit)
print("Test error rate:",round(error_rate_logit1,3))
print(classification_report(y_test, testlogit, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testlogit).T
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='winter')
```

```
Test error rate: 0.322
              precision    recall  f1-score   support

           0       0.97      0.50      0.66        56
           1       0.54      0.97      0.69        34

    accuracy                           0.68        90
   macro avg       0.75      0.74      0.68        90
weighted avg       0.81      0.68      0.67        90
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f66cb9ee390>



The test error rate for this test sample is 0.322, implying that around 32% of its classifications were incorrect.

(h)

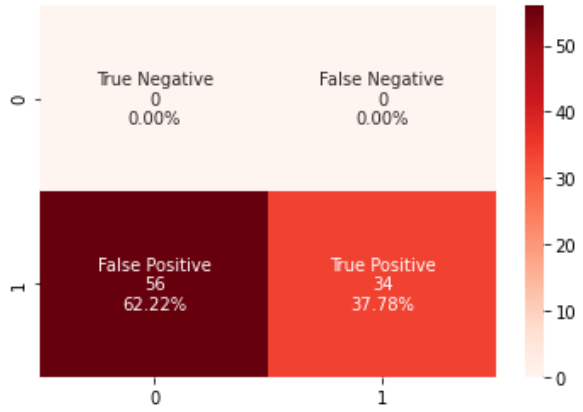
In [ ]:
```
#RANDOM SAMPLING
import random
```

```
random.seed(10)
randomclf = []
for i in range(207):
    n = random.choice([0,1])
    randomclf.append(n)

#ESTIMATION
logit.fit(X_train,randomclf)
randprob = logit.predict_proba(X_test)

#OUTPUT
testclf=[]
for i in range(len(randprob)):
    if (randprob[i][1]>0.2)==True:
        testclf.append(1)
    else:
        testclf.append(0)

error_rate_logit2 = 1-accuracy_score(y_test, testclf)
print("Test error rate:",round(error_rate_logit2,3))
print(classification_report(y_test, testclf, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testclf).T
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Reds')
```

```
Test error rate: 0.622
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.38      1.00      0.55        34

    accuracy                           0.38        90
   macro avg       0.19      0.50      0.27        90
weighted avg       0.14      0.38      0.21        90
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f66cb467f50>



In [ ]:

(i)

Using the random generator function we achieve a test error rate of 0.622 meaning that 62% of observations were misclassified in comparison with 32% when using the logistic regression results. The increased test error rate seems to be driven by an increase in the number of false positives which has increased significantly when classifying randomly.

(2)

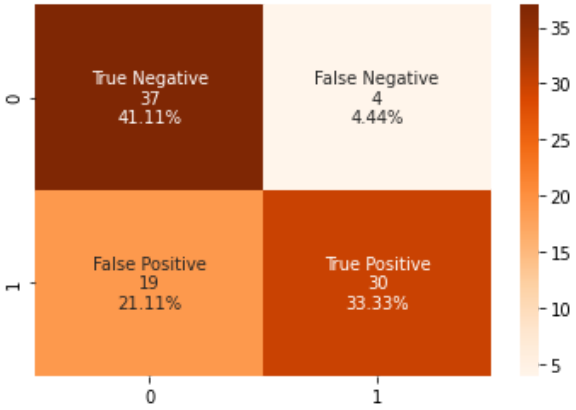
(a)

In [ ]:
```
#ESTIMATION
knn5 = neighbors.KNeighborsClassifier(n_neighbors=5)
knnpred5 = knn5.fit(X_train, y_train)
knn5prob = knnpred5.predict_proba(X_test)

#OUTPUT
testknn5=[]
for i in range(len(knn5prob)):
    if (knn5prob[i][1]>0.2)==True:
        testknn5.append(1)
    else:
        testknn5.append(0)

error_rate_knn5 = 1-accuracy_score(y_test, testknn5)
print("Test error rate:",round(error_rate_knn5,3))
print(classification_report(y_test, testknn5, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testknn5).T
```

```
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Oranges')
```

```
Test error rate: 0.256
              precision    recall  f1-score   support

           0       0.90      0.66      0.76        56
           1       0.61      0.88      0.72        34

    accuracy                           0.74        90
   macro avg       0.76      0.77      0.74        90
weighted avg       0.79      0.74      0.75        90
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f66cb928cd0>
```

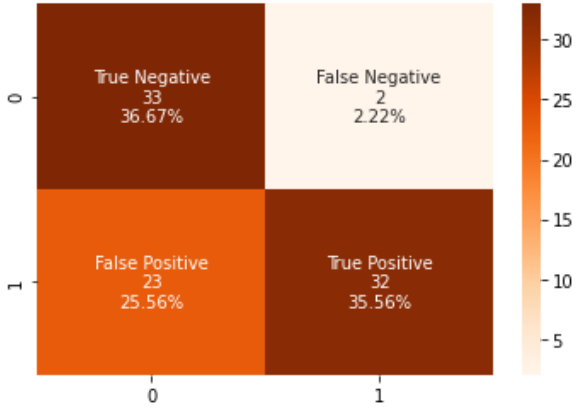


In [ ]:
```
#ESTIMATION
knn10 = neighbors.KNeighborsClassifier(n_neighbors=10)
knnpred10 = knn10.fit(X_train, y_train)
knn10prob = knnpred10.predict_proba(X_test)

#OUTPUT
testknn10=[]
for i in range(len(knn10prob)):
    if (knn10prob[i][1]>0.2)==True:
        testknn10.append(1)
    else:
        testknn10.append(0)

error_rate_knn10 = 1-accuracy_score(y_test, testknn10)
print("Test error rate:",round(error_rate_knn10,3))
```

```
print(classification_report(y_test, testknn10, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testknn10).T
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Oranges')
```

```
Test error rate: 0.278
              precision    recall  f1-score   support

           0       0.94      0.59      0.73        56
           1       0.58      0.94      0.72        34

    accuracy                           0.72        90
   macro avg       0.76      0.77      0.72        90
weighted avg       0.81      0.72      0.72        90
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f66cb3e7190>



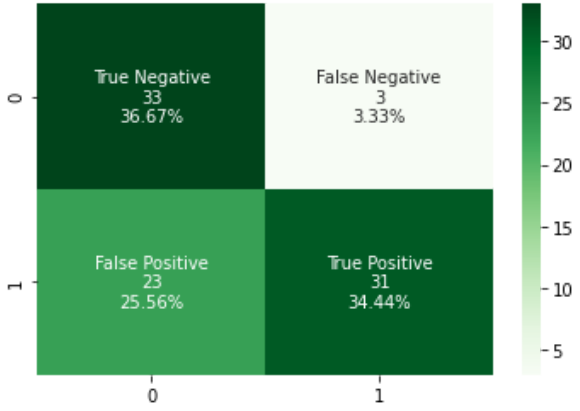
(b)

In [ ]:
```
#ESTIMATION
lda = LinearDiscriminantAnalysis()
ldapred = lda.fit(X_train, y_train)
ldaprob = ldapred.predict_proba(X_test)

#OUTPUT
testlda=[]
```

```
for i in range(len(ldaprob)):
    if (ldaprob[i][1]>0.2)==True:
        testlda.append(1)
    else:
        testlda.append(0)

error_rate_lda = 1-accuracy_score(y_test, testlda)
print("Test error rate:",round(error_rate_lda,3))
print(classification_report(y_test, testlda, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testlda).T
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Greens')
```

```
Test error rate: 0.289
              precision    recall  f1-score   support

           0       0.92      0.59      0.72        56
           1       0.57      0.91      0.70        34

    accuracy                           0.71        90
   macro avg       0.75      0.75      0.71        90
weighted avg       0.79      0.71      0.71        90
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f66cb26ded0>
```



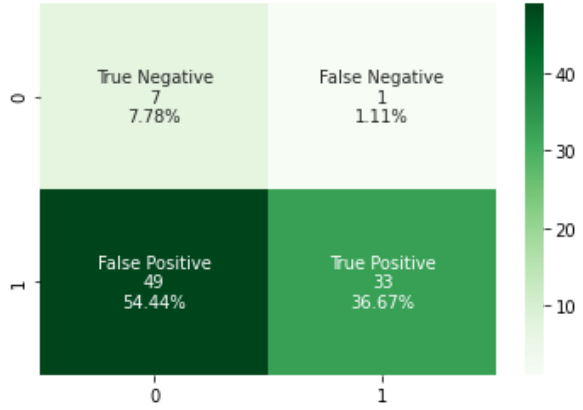
In [ ]:
```
#ESTIMATION
qda = QuadraticDiscriminantAnalysis()
qdapred = qda.fit(X_train, y_train)
```

```
qdaprob = qdapred.predict_proba(X_test)

#OUTPUT
testqda=[]
for i in range(len(qdaprob)):
    if (qdaprob[i][1]>0.2)==True:
        testqda.append(1)
    else:
        testqda.append(0)

error_rate_qda = 1-accuracy_score(y_test, testqda)
print("Test error rate:",round(error_rate_qda,3))
print(classification_report(y_test, testqda, digits=2))

#CONFUSION MATRIX
cf_matrix=confusion_matrix(y_test, testqda).T
group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Greens')
```

```
Test error rate: 0.556
              precision    recall  f1-score   support

           0       0.88      0.12      0.22        56
           1       0.40      0.97      0.57        34

    accuracy                           0.44        90
   macro avg       0.64      0.55      0.39        90
weighted avg       0.70      0.44      0.35        90
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/discriminant_analysis.py:691: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f66cb201990>



(c)

In [ ]:
```
TER=pd.DataFrame([error_rate_logit1,error_rate_knn5,error_rate_knn10,error_rate_lda,error_rate_qda])
TER.rename(columns={0:"Test Error Rate"},index={0:"Logit",1:"KNN5",2:"KNN10",3:"LDA",4:"QDA"},inplace=True)
TER.round({"Test Error Rate":3})
```

Out[ ]:

	Test Error Rate
Logit	0.322
KNN5	0.256
KNN10	0.278
LDA	0.289
QDA	0.556

From the above, we can see that the 5-Nearest-Neighbours approach is best with a test error rate of 0.256. This is followed by 10-Nearest-Neighbours then LDA. The fourth best is logistic regression with a test error rate of 0.322. The worst, by some distance, is QDA which has a test error of 0.556. The very poor performance of QDA suggests the underlying distributions may not be normal, potentially due to the small sample size.

(d)

Using train test split changes the outcome again and again depending on how the function determines the 4 samples - use K-fold instead.

In [ ]:
```
models = [logit,knn5,knn10,lda,qda]
te=[]
for i in models:
    te=te+[(1-model_selection.cross_val_score(i,X,y,cv=model_selection.KFold(n_splits=10)).mean())]
te=pd.DataFrame(te)
```

```
te.rename(columns={0:"Test Error Rate"}, index={0:"Logit",1:"KNN5",2:"KNN10",3:"LDA",4:"QDA"},inplace=True)
te.round({"Test Error Rate":3})
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/discriminant_analysis.py:691: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

Out[ ]:

	Test Error Rate
Logit	0.139
KNN5	0.226
KNN10	0.199
LDA	0.135
QDA	0.182

After cross-validation, the order of performance changes. Subsequently, LDA now performs best followed by logistic regression, QDA, 10-Nerest-Neighbours then 5-Nearest-Neighbours. QDA displays a large improvement in test error rate, from 0.556 to 0.182, with all other approaches also decreasing their test error rate but to a lesser extent.

Whereas above we were looking at the test error rate from one split of the data, by cross validating the data we are potentially presenting a better representation of the true quality of the approaches as we are taking different samples. This argument is seemingly supported by the fact that some approaches improved but others worsened when using cross-validation, suggesting the Nearest-Neighbour approaches worked well for the individual data split but are not as good once multiple different splits are carried out.

(e)

The model with the lowest test error rate using 10-fold CV is linear discriminant analysis - using patient Xofs from before we can reclassify to see how the model performs.

In [ ]:
```
model_selection.cross_val_predict(lda,X,y,cv=model_selection.KFold(n_splits=10))
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train).predict_proba(Xofs)
```

Out[ ]:
```
array([[0.8849061, 0.1150939]])
```

Based on the result above, we see that:

$$pr(y = 1) = 0.12 < 0.2$$

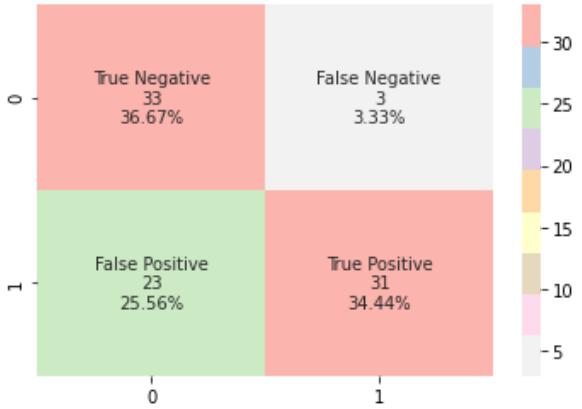
Therefore the patient is again, classified as not having heart disease despite using a more accurate model.

(f)

In [ ]:
```
cf_matrix=confusion_matrix(y_test, testlda).T

group_names = ["True Negative","False Negative","False Positive","True Positive"]
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
```

```
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt="", cmap='Pastel1_r')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f66cb0d3d90>



The confusion matrix presents a breakdown of the in-class performance for LDA. Of particular interest are the false-positive and false-negative rates that are shown. The false-positive rate is 3.33% which reflects 3 classifications where we predicted that the person had heart disease when in reality they did not. Moreover, the false-negative rate is 25.56% reflecting 23 predictions where we estimated that they did not have heart disease when in reality they did.

The false-negative rate being relatively higher than the false-positive result in this case may be of concern as we are missing a lot of people that may have a life-threatening heart condition. However, the confusion matrix shows that, overall, we correctly predict the majority of observations with a true-negative rate of 36.67% and a true-positive rate of 34.44%

(g)

In [ ]:
```
ldaprob = ldapred.predict_proba(X_test)
testlda0=[]
testlda025=[]
testlda05=[]
testlda075=[]
testlda1=[]
testldalist = [testlda0,testlda025,testlda05,testlda075,testlda1]

for i in range(len(ldaprob)):
    if (ldaprob[i][1]>0)==True:
        testlda0.append(1)
    else:
        testlda0.append(0)
```

```
for i in range(len(ldaprob)):
    if (ldaprob[i][1]>0.25)==True:
        testlda025.append(1)
    else:
        testlda025.append(0)

for i in range(len(ldaprob)):
    if (ldaprob[i][1]>0.5)==True:
        testlda05.append(1)
    else:
        testlda05.append(0)

for i in range(len(ldaprob)):
    if (ldaprob[i][1]>0.75)==True:
        testlda075.append(1)
    else:
        testlda075.append(0)

for i in range(len(ldaprob)):
    if (ldaprob[i][1]>1)==True:
        testlda1.append(1)
    else:
        testlda1.append(0)

for i in testldalist:
  print("Test error rate:",round(1-accuracy_score(y_test, i),3))
```

```
Test error rate: 0.622
Test error rate: 0.244
Test error rate: 0.222
Test error rate: 0.167
Test error rate: 0.378
```

In [ ]:

```
cf_matrix0=confusion_matrix(y_test, testlda0).T
cf_matrix025=confusion_matrix(y_test, testlda025).T
cf_matrix05=confusion_matrix(y_test, testlda05).T
cf_matrix075=confusion_matrix(y_test, testlda075).T
cf_matrix1=confusion_matrix(y_test, testlda1).T
#TPR = TP/(TP+FN)
#FPR = FP/(FP+TN)
cfmatrix=[cf_matrix0,cf_matrix025,cf_matrix05,cf_matrix075,cf_matrix1]
TP=[]
for i in cfmatrix:
    TP=TP+[i[1][1]/(i[1][1]+i[0][1])]
TP=np.nan_to_num(np.asarray(TP))
TP=pd.DataFrame(TP).rename(columns={0:"True Positive Rate"})

FP=[]
for i in cfmatrix:
    FP=FP+[i[1][0]/(i[1][0]+i[0][0])]
FP=np.nan_to_num(np.asarray(FP))
FP=pd.DataFrame(FP).rename(columns={0:"False Positive Rate"})
```

```
TPFP=pd.concat([TP,FP],axis=1)
TPFP.rename(index={0: "0", 1: "0.25", 2: "0.5", 3: "0.75", 4: "1"},inplace=True)
TPFP
```

Out[ ]:

	True Positive Rate	False Positive Rate
0	1.000000	1.000000
0.25	0.911765	0.339286
0.5	0.823529	0.250000
0.75	0.735294	0.107143
1	0.000000	0.000000

When the threshold is 0, it means classifying the individual as having heart disease if the estimated AHD is greater than 0. In this case, we take all estimations as having heart disease, so no one who actually have heart disease was missed to be identified, so the true positive rate is 1. But all individuals who do not have heart disease are classified as having the disease, so the false positive rate also equals to one.

When the threshold is 0.25, it means classifying the individual as having heart disease if the estimated AHD is greater than 0.25, so the majority of individuals have heart disease can be identified, and also the majority of individuals who do not have heart disease are unclassified, thus we have a high true positive rate and a high false positive rate.

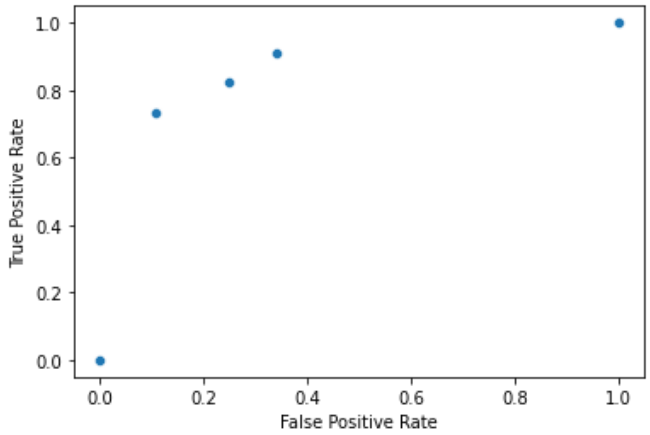
As the threshold increases, the proportion of estimations that are classified as having heart disease decreases, which means less proportion of individuals who actually have heart disease are classified as positive, hence the true positive rate is decreasing. At the same time, higher proportion of individuals are classified as not having heart disease, so individuals who actually not having the disease are correctly classified, hence the false positive rate also declines.

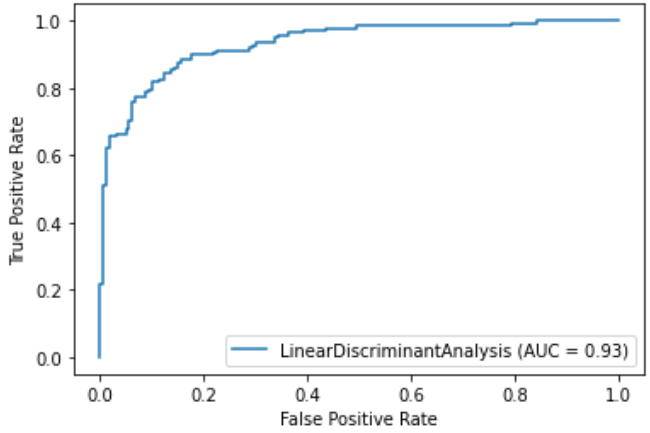
At the threshold of 1, individuals who have estimated AHD smaller than 1 are all classified as not having heart disease, so all individuals are classified as negative, therefore the true positive rate is 0 and the false positive rate is also 0.

(h)

In [ ]:
```
sns.scatterplot(x=TPFP["False Positive Rate"],y=TPFP["True Positive Rate"])
plot_roc_curve(lda,X,y)
```

Out[ ]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f66cb0870d0>
```





By plotting the results of the false positive rate against the true positive rate obtained from the confusion matrices at each threshold and comparing it to the ROC curve generated by the estimator, it is clear to see how the first figure characterises the ROC curve which supports the results obtained in question (g).