Homework 2: Classification Methods

Luying Jiang

In [1]:
```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import seaborn
from tabulate import tabulate
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
as QDA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
```

The Bayes Classifier

Question 1

a.

In [2]:
```
np.random.seed(1234)
```

b.

In [3]:
```
x1 = np.random.uniform(-1, 1, 200)
x2 = np.random.uniform(-1, 1, 200)
```

c.

In [4]:
```
error = np.random.normal(0, 0.5, 200)
y = x1 + x1**2 + x2 + x2**2 + error
```

d.

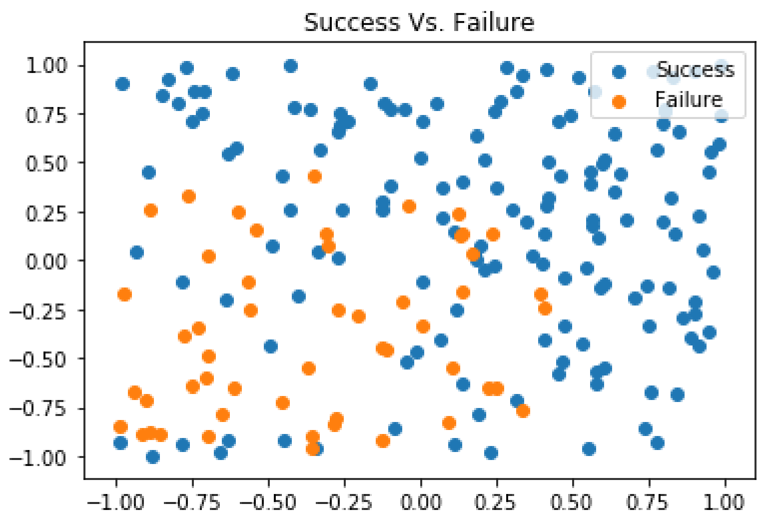
Since $log(\frac{P(Success)}{1-P(Success)}) = Y$**,**

$$P(Success) = \frac{e^Y}{1+e^Y}$$

In [5]:
```
p_success = (math.e ** y)/(1 + math.e ** y)
```

e.

In [6]:
```
success = p_success > 0.5
failure = p_success <= 0.5
plt.scatter(x1[success], x2[success])
plt.scatter(x1[failure], x2[failure])
plt.title('Success Vs. Failure')
plt.legend(['Success', 'Failure'], loc=1);
```



f & g & h.

In [7]:
```
x = np.column_stack((x1,x2))
df = pd.DataFrame(x)
gnb = GaussianNB()
gnb.fit(df, success)
```

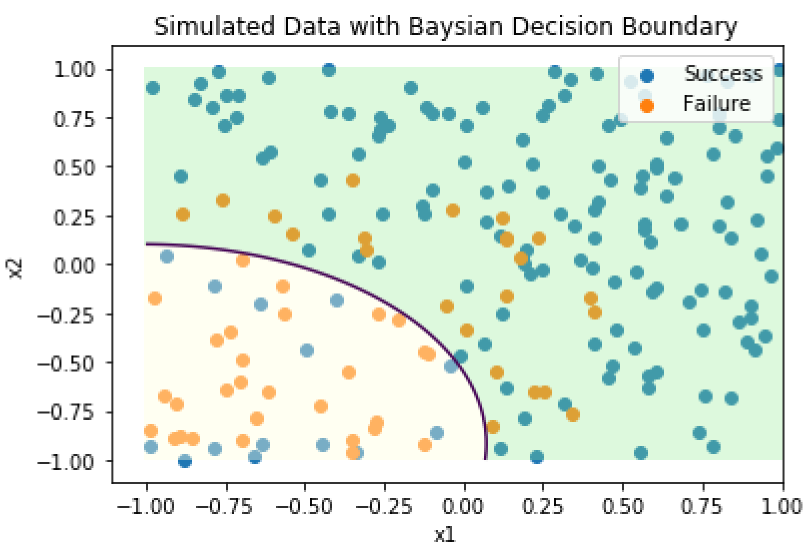
Out[7]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
In [8]: p = np.linspace(-1, 1, 100)
        q = np.linspace(-1, 1, 100)
        xv, yv = np.meshgrid(p, q)
        z = gnb.predict_proba(np.c_[xv.ravel(), yv.ravel()])
        z
```

```
Out[8]: array([[8.22971292e-01, 1.77028708e-01],
               [8.22773717e-01, 1.77226283e-01],
               [8.22419949e-01, 1.77580051e-01],
               ...,
               [1.18056694e-04, 9.99881943e-01],
               [1.06288784e-04, 9.99893711e-01],
               [9.55915874e-05, 9.99904408e-01]])
```

```
In [9]: z = z[:, 1].reshape(xv.shape)
```

```
In [10]: plt.scatter(x1[success], x2[success])
         plt.scatter(x1[failure], x2[failure])
         plt.contour(xv, yv, z, [0.5])
         plt.contourf(xv, yv, z, [0,0.5], colors = 'lightyellow', alpha=.4)
         plt.contourf(xv, yv, z, [0.5,1], colors='lightgreen', alpha=.3)
         plt.xlabel('x1')
         plt.ylabel('x2')
         plt.title('Simulated Data with Baysian Decision Boundary')
         plt.legend(['Success','Failure'], loc=1);
```



Exploring Simulated Differences between LDA and QDA

Question 2

If the Bayes decision boundary is linear, we expect QDA to perform better on the training set because its higher flexiblity may yield a closer fit. On the test set, we expect LDA to perform better than QDA, because QDA could overfit the linearity on the Bayes decision boundary.

In [11]:
```
def simulate(n, nonlinear = 0):

    error_lst = []
    for i in range(1000):
        x1 = np.random.uniform(-1, 1, n)
        x2 = np.random.uniform(-1, 1, n)
        y_simu = x1 + x2 + (x1 ** 2) * nonlinear + (x2 ** 2) * nonlinear
+ np.random.normal(0, 1, n)
        y_simu_bi = y_simu >= 0
        X = np.column_stack((x1, x2))
        X_train, X_test, y_train, y_test = train_test_split(X, y_simu_bi
, test_size=0.3, shuffle=True)
        lda = LDA()
        lda.fit(X_train, y_train)
        lda_train_err = 1 - lda.score(X_train, y_train)
        lda_test_err = 1- lda.score(X_test, y_test)
        qda = QDA()
        qda.fit(X_train, y_train)
        qda_train_err = 1 - qda.score(X_train,y_train)
        qda_test_err = 1- qda.score(X_test, y_test)
        error_lst.append([lda_train_err, lda_test_err, qda_train_err, qd
a_test_err])

    return error_lst
```

In [12]:
```
error_lst = simulate(1000, 0)
```

In [13]:
```
df = pd.DataFrame(error_lst, columns=['lda_train_error', 'lda_test_erro
r', 'qda_train_error', 'qda_test_error'])
df.head(10)
```

Out[13]:

	lda_train_error	lda_test_error	qda_train_error	qda_test_error
0	0.265714	0.306667	0.265714	0.303333
1	0.278571	0.273333	0.282857	0.276667
2	0.287143	0.270000	0.288571	0.256667
3	0.284286	0.280000	0.280000	0.283333
4	0.264286	0.226667	0.261429	0.216667
5	0.278571	0.296667	0.278571	0.300000
6	0.261429	0.226667	0.262857	0.223333
7	0.285714	0.270000	0.285714	0.273333
8	0.264286	0.313333	0.264286	0.306667
9	0.255714	0.300000	0.257143	0.310000

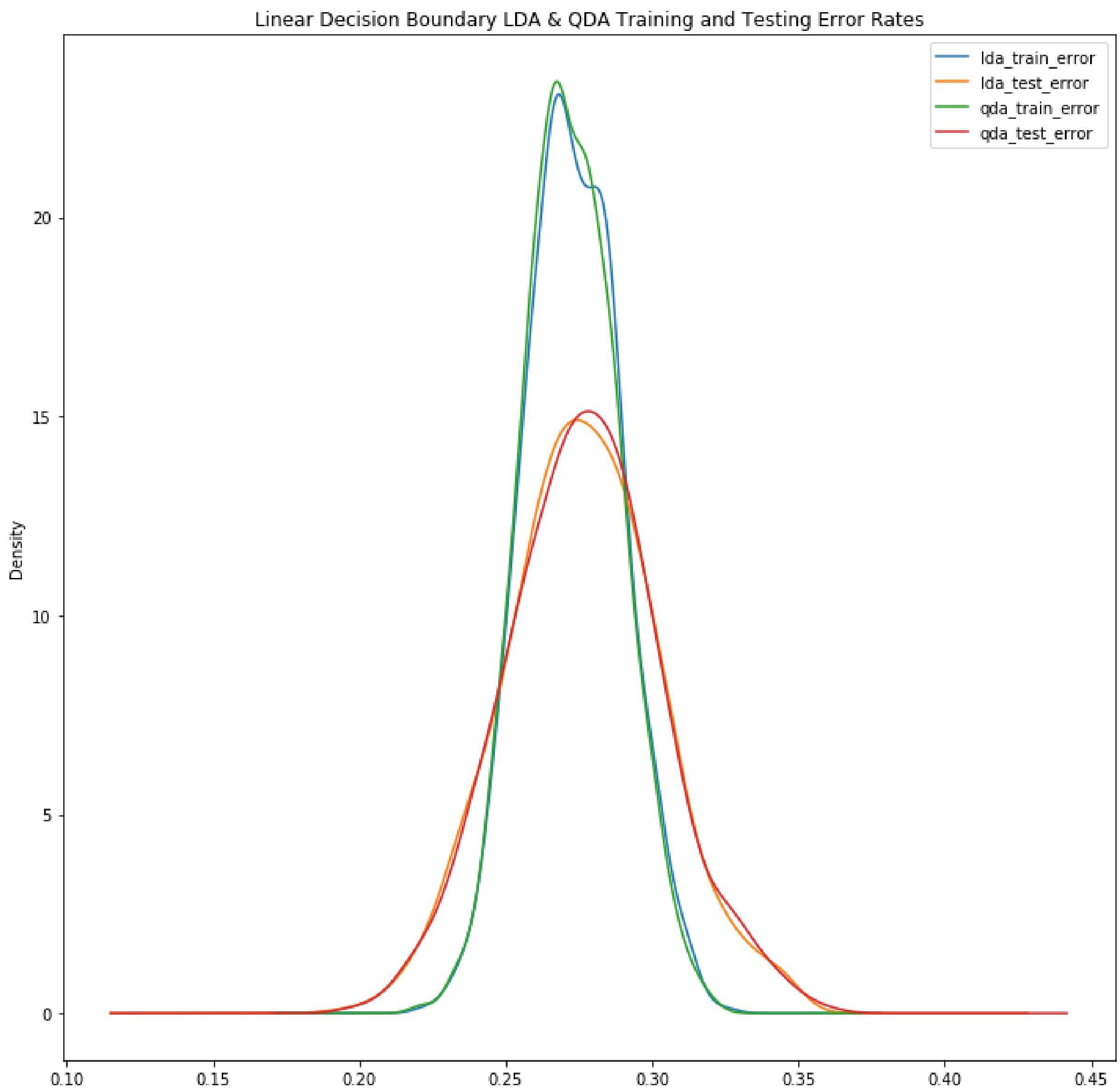
In [14]:
```
df.describe()
```

Out[14]:

	lda_train_error	lda_test_error	qda_train_error	qda_test_error
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.273219	0.276280	0.272479	0.276820
std	0.016298	0.026244	0.016121	0.026368
min	0.221429	0.193333	0.221429	0.196667
25%	0.261429	0.260000	0.261429	0.260000
50%	0.272857	0.276667	0.271429	0.276667
75%	0.284286	0.293333	0.284286	0.293333
max	0.325714	0.350000	0.321429	0.360000

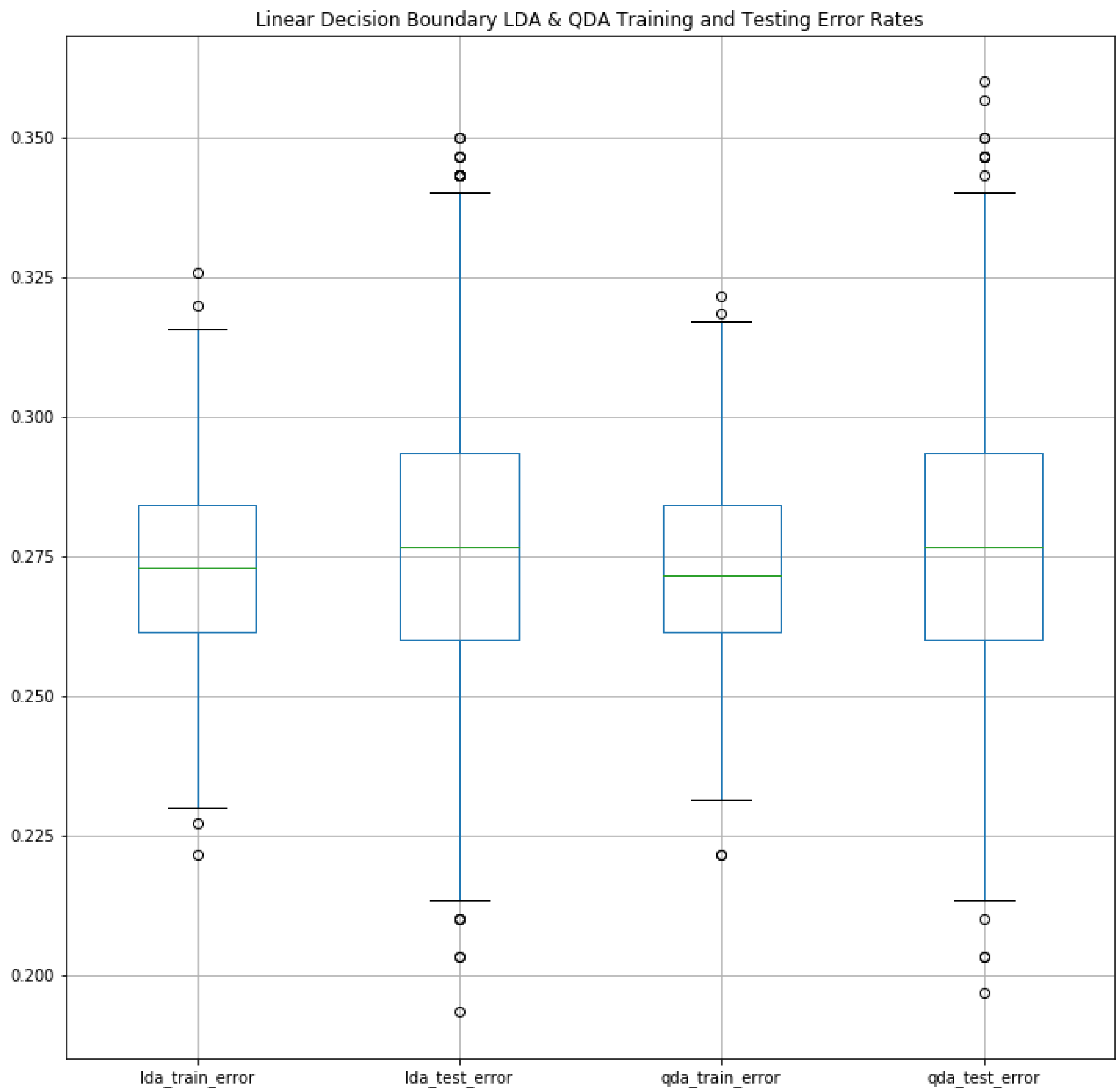
In [15]:
```
df.plot.density(figsize=(12,12))
plt.title('Linear Decision Boundary LDA & QDA Training and Testing Error Rates')
```

Out[15]:
```
Text(0.5, 1.0, 'Linear Decision Boundary LDA & QDA Training and Testing Error Rates')
```



In [16]:
```
df.boxplot(figsize=(12,12))
plt.title('Linear Decision Boundary LDA & QDA Training and Testing Error Rates')
```

Out[16]:
```
Text(0.5, 1.0, 'Linear Decision Boundary LDA & QDA Training and Testing Error Rates')
```



As shown above, QDA performs better on the trainning set and LDA performs better on test set.

Question 3

If the Bayes decision bounary is non-linear, we expect QDA to perform better both on the training and test sets.

In [17]:
```
error_lst2 = simulate(1000, 1)
```

In [18]:

```
df = pd.DataFrame(error_lst2, columns=['lda_train_error', 'lda_test_erro
r', 'qda_train_error', 'qda_test_error'])
df.describe()
```

Out[18]:

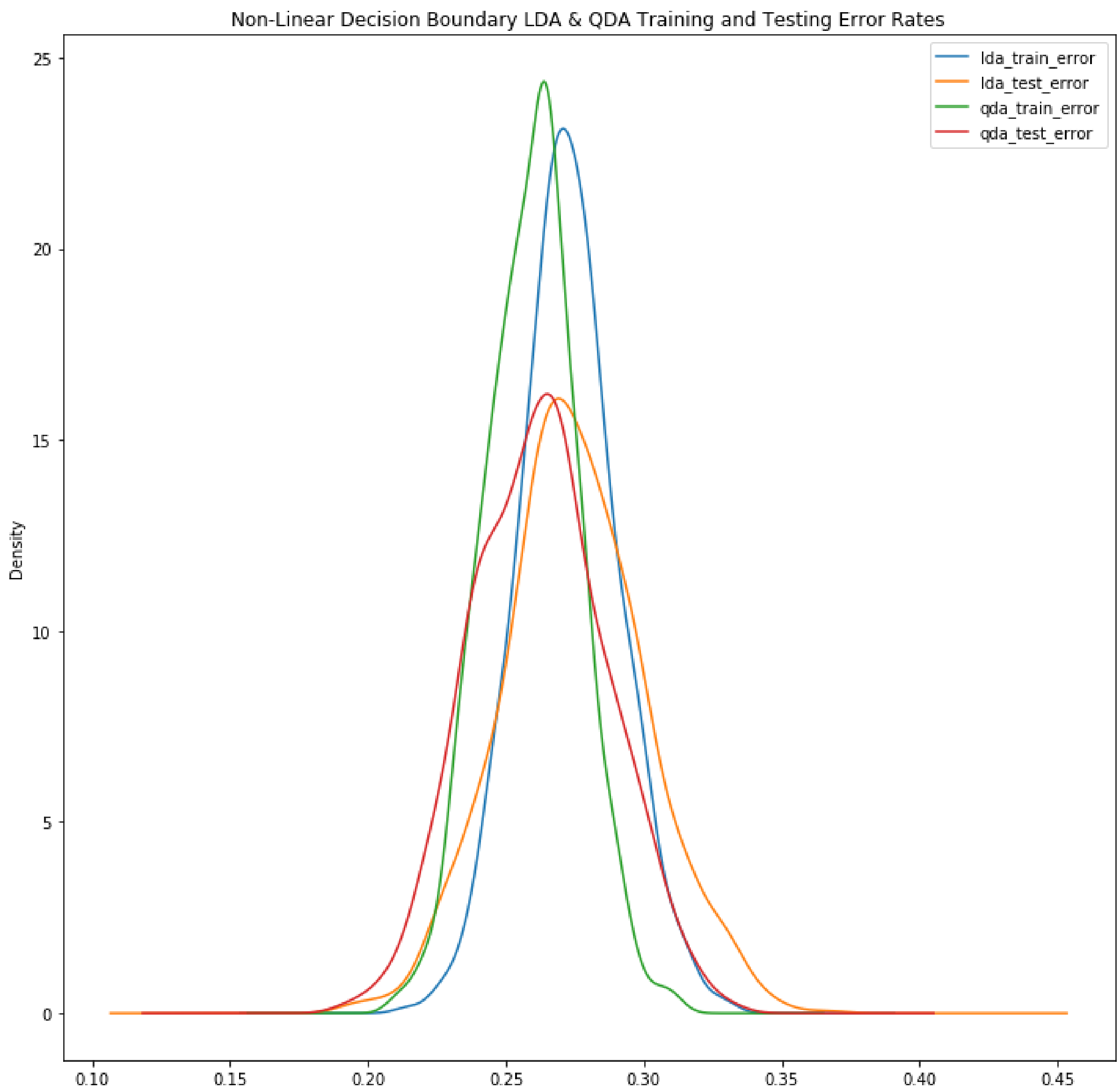
	lda_train_error	lda_test_error	qda_train_error	qda_test_error
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.272799	0.274627	0.259430	0.262140
std	0.017898	0.025932	0.016799	0.024327
min	0.212857	0.193333	0.208571	0.190000
25%	0.261429	0.256667	0.247143	0.243333
50%	0.271429	0.273333	0.260000	0.263333
75%	0.284286	0.290000	0.270000	0.276667
max	0.331429	0.366667	0.312857	0.333333

In [19]:

```
df.plot.density(figsize=(12,12))
plt.title('Non-Linear Decision Boundary LDA & QDA Training and Testing E
rror Rates')
```

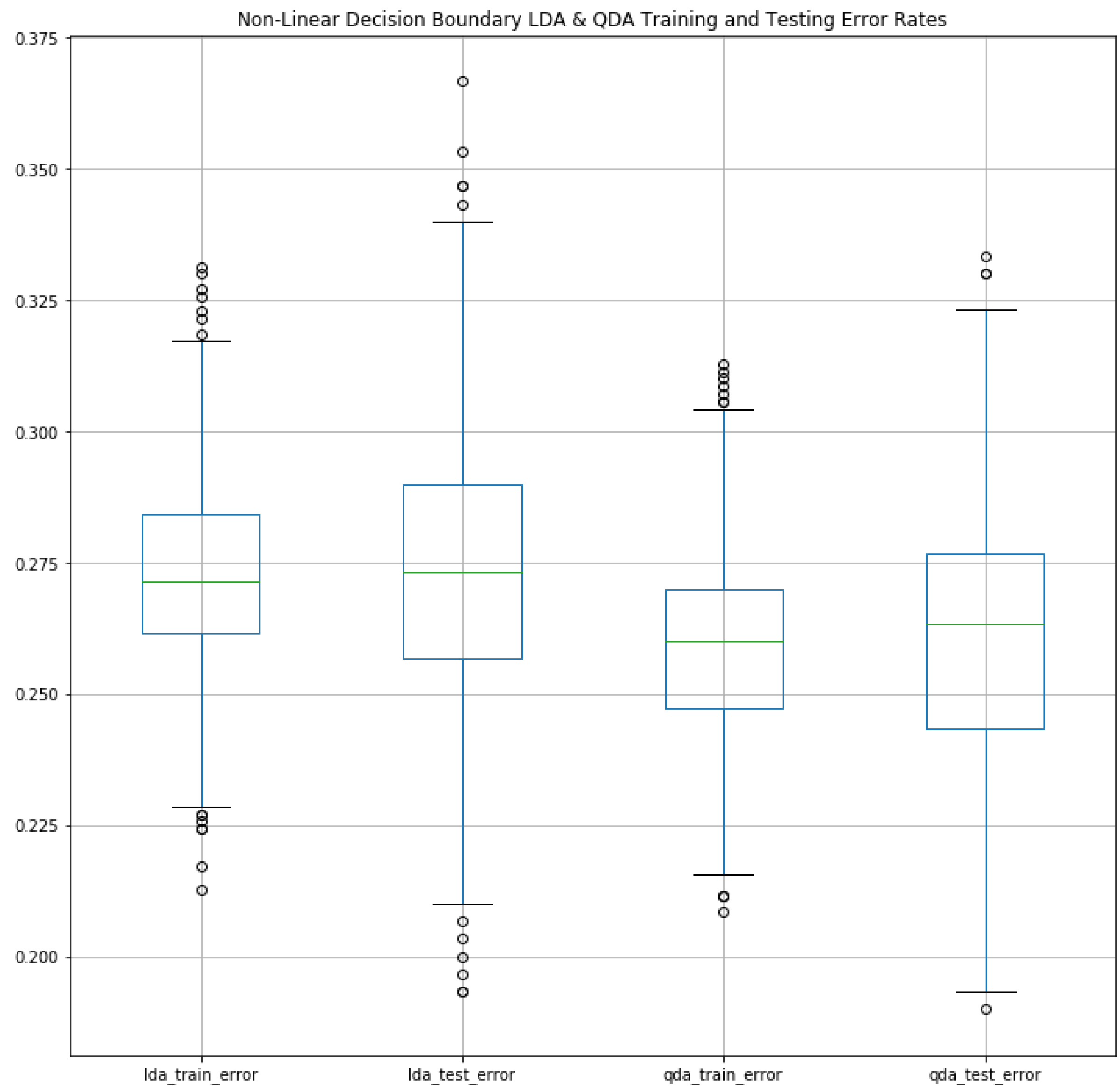
Out[19]:

```
Text(0.5, 1.0, 'Non-Linear Decision Boundary LDA & QDA Training and Tes
ting Error Rates')
```



In [20]:
```
df.boxplot(figsize=(12,12))
plt.title('Non-Linear Decision Boundary LDA & QDA Training and Testing E
rror Rates')
```

Out[20]:
```
Text(0.5, 1.0, 'Non-Linear Decision Boundary LDA & QDA Training and Tes
ting Error Rates')
```



As shown above, QDA performs better in both testing and training data.

Question 4

In general, QDA, which is more flexible than LDA and so has higher variance, performs better than LDA if the training set is very large. The error rate of QDA relative to LDA decreases

In [21]:
```python
error_lst_e02 = simulate(100, 1)
df_100 = pd.DataFrame(error_lst_e02, columns=['lda_train_error_100', 'ld
a_test_error_100', 'qda_train_error_100', 'qda_test_error_100'])
```

In [22]:
```python
df_100.describe()
```

Out[22]:

	lda_train_error_100	lda_test_error_100	qda_train_error_100	qda_test_error_100
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.265357	0.286500	0.243329	0.275933
std	0.052539	0.081847	0.049239	0.083286
min	0.085714	0.033333	0.100000	0.033333
25%	0.228571	0.233333	0.214286	0.225000
50%	0.257143	0.300000	0.242857	0.266667
75%	0.300000	0.333333	0.271429	0.333333
max	0.428571	0.533333	0.385714	0.566667

In [23]:
```python
error_lst_e03 = simulate(1000, 1)
df_1000 = pd.DataFrame(error_lst_e03, columns=['lda_train_error_1000',
'lda_test_error_1000', 'qda_train_error_1000', 'qda_test_error_1000'])
```

In [24]:
```python
df_1000.describe()
```

Out[24]:

	lda_train_error_1000	lda_test_error_1000	qda_train_error_1000	qda_test_error_1000
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.272271	0.274613	0.258596	0.261163
std	0.017076	0.025663	0.016545	0.025658
min	0.211429	0.190000	0.201429	0.180000
25%	0.261429	0.259167	0.247143	0.243333
50%	0.272857	0.273333	0.258571	0.260000
75%	0.284286	0.290000	0.270000	0.276667
max	0.337143	0.360000	0.320000	0.340000

In [25]:
```python
error_lst_e04 = simulate(10000, 1)
df_10000 = pd.DataFrame(error_lst_e04, columns=['lda_train_error_10000',
'lda_test_error_10000', 'qda_train_error_10000', 'qda_test_error_10000'
])
```

In [26]:
```python
df_10000.describe()
```

Out[26]:

	lda_train_error_10000	lda_test_error_10000	qda_train_error_10000	qda_test_error_10000
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.273486	0.273920	0.260487	0.261210
std	0.005467	0.007931	0.005087	0.008048
min	0.258000	0.249667	0.245143	0.236000
25%	0.270000	0.268667	0.257143	0.255667
50%	0.273714	0.273667	0.260571	0.261333
75%	0.277143	0.279333	0.263857	0.266667
max	0.291429	0.300000	0.276286	0.287000

In [27]:
```python
error_lst_e05 = simulate(100000, 1)
df_100000 = pd.DataFrame(error_lst_e05, columns=['lda_train_error_100000', 'lda_test_error_100000', 'qda_train_error_100000', 'qda_test_error_100000'])
```

In [28]:
```python
df_10000.describe()
```

Out[28]:

	lda_train_error_10000	lda_test_error_10000	qda_train_error_10000	qda_test_error_10000
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.273486	0.273920	0.260487	0.261210
std	0.005467	0.007931	0.005087	0.008048
min	0.258000	0.249667	0.245143	0.236000
25%	0.270000	0.268667	0.257143	0.255667
50%	0.273714	0.273667	0.260571	0.261333
75%	0.277143	0.279333	0.263857	0.266667
max	0.291429	0.300000	0.276286	0.287000

In [29]:
```python
df = pd.concat([df_100, df_1000, df_10000, df_100000], axis=1)
```

In [30]:
```
df.head(10)
```

Out[30]:

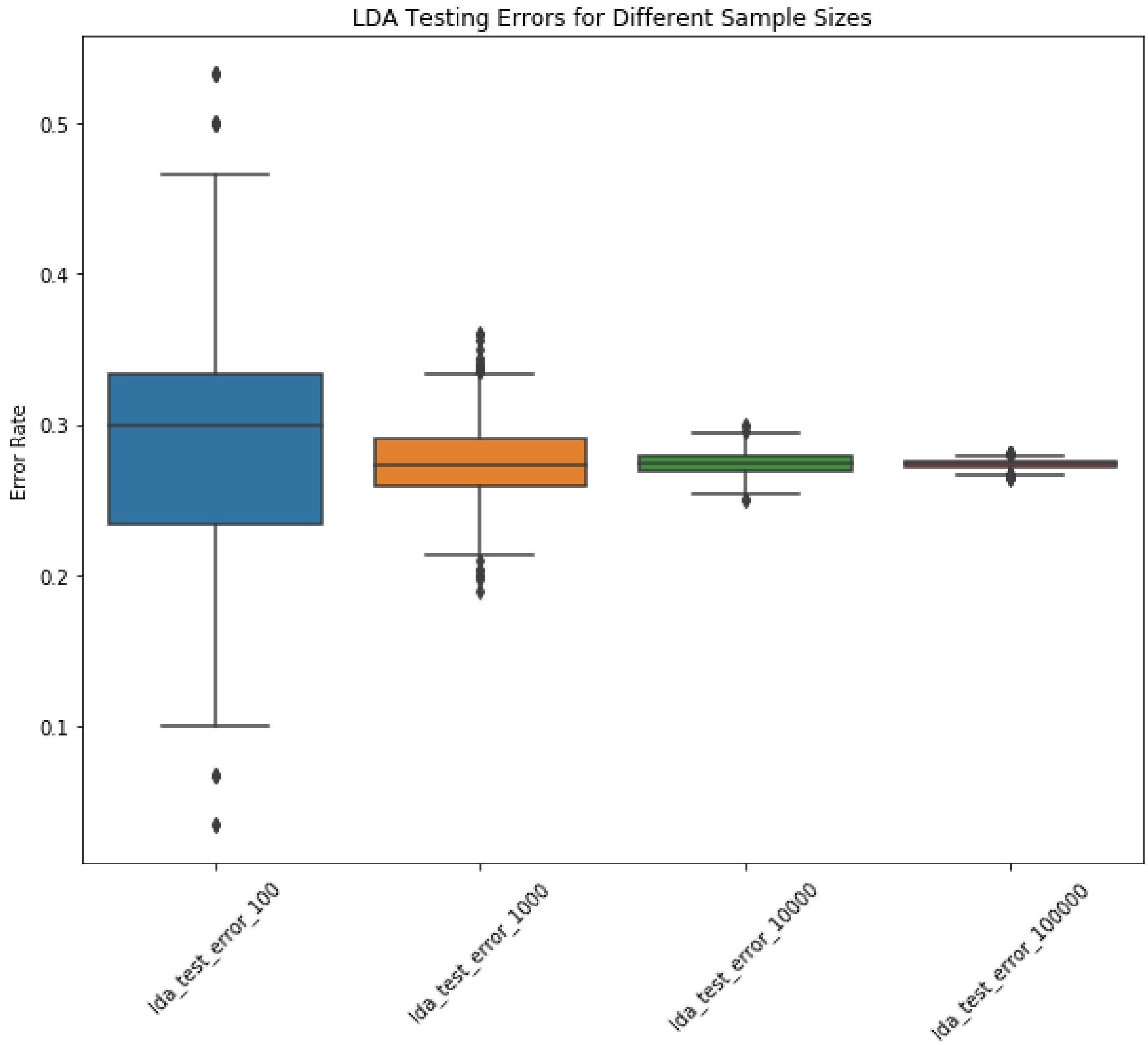
	lda_train_error_100	lda_test_error_100	qda_train_error_100	qda_test_error_100	lda_train_error_1
0	0.214286	0.266667	0.157143	0.266667	0.271
1	0.328571	0.433333	0.314286	0.400000	0.268
2	0.300000	0.266667	0.300000	0.266667	0.277
3	0.300000	0.233333	0.342857	0.266667	0.278
4	0.228571	0.233333	0.200000	0.233333	0.275
5	0.371429	0.166667	0.314286	0.166667	0.291
6	0.300000	0.433333	0.271429	0.333333	0.294
7	0.285714	0.166667	0.242857	0.200000	0.287
8	0.171429	0.400000	0.200000	0.300000	0.278
9	0.228571	0.266667	0.214286	0.266667	0.264

In [31]:
```
df.columns
```

Out[31]:
```
Index(['lda_train_error_100', 'lda_test_error_100', 'qda_train_error_10
0',
       'qda_test_error_100', 'lda_train_error_1000', 'lda_test_error_10
00',
       'qda_train_error_1000', 'qda_test_error_1000', 'lda_train_error_
10000',
       'lda_test_error_10000', 'qda_train_error_10000', 'qda_test_error
_10000',
       'lda_train_error_100000', 'lda_test_error_100000',
       'qda_train_error_100000', 'qda_test_error_100000'],
      dtype='object')
```

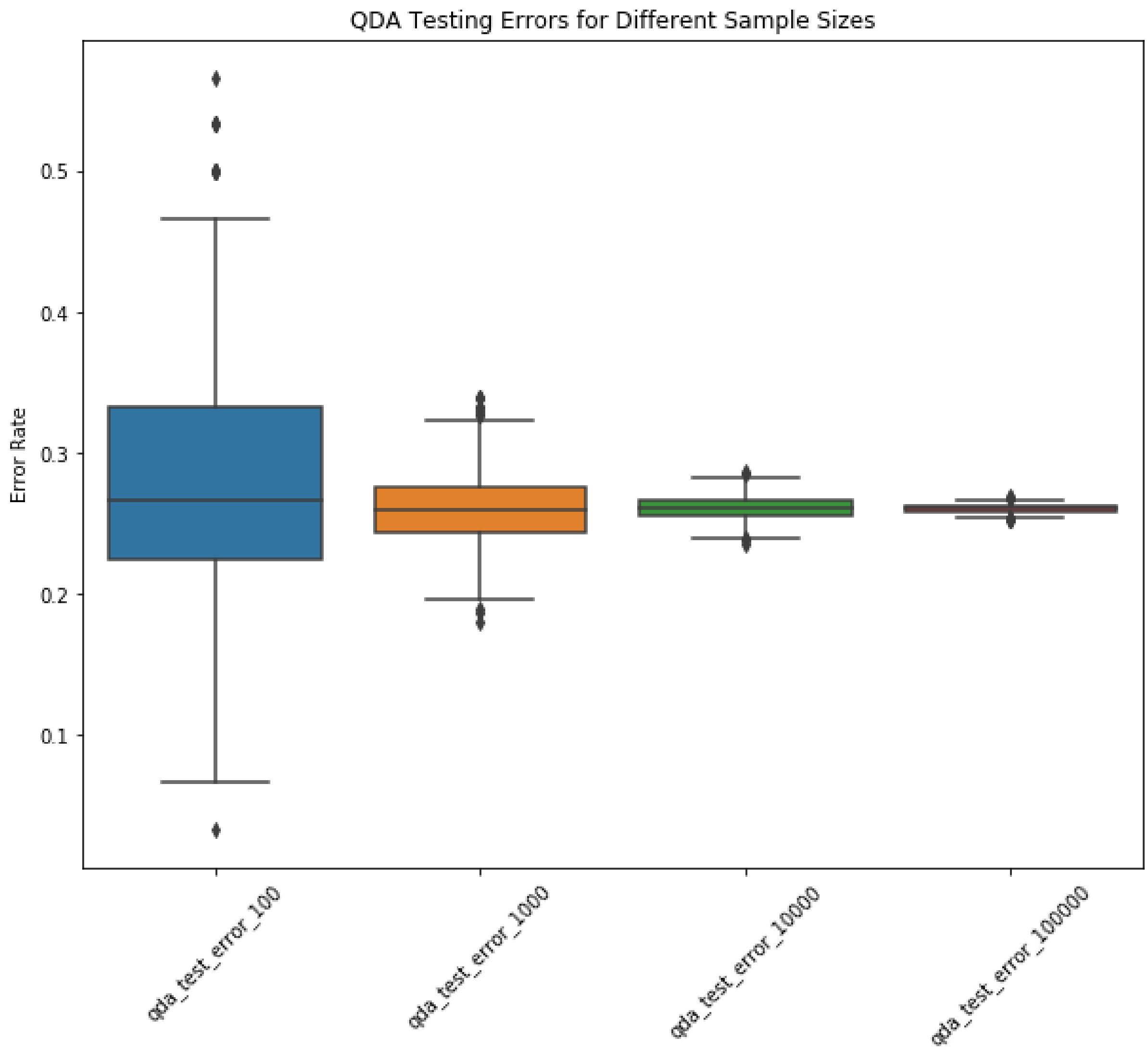
In [32]:

```
plt.figure(figsize=(10,8))
seaborn.boxplot(data=df[['lda_test_error_100','lda_test_error_1000',
                         'lda_test_error_10000','lda_test_error_100000'
]])
plt.xticks(rotation=45)
plt.ylabel('Error Rate')
plt.title('LDA Testing Errors for Different Sample Sizes');
```



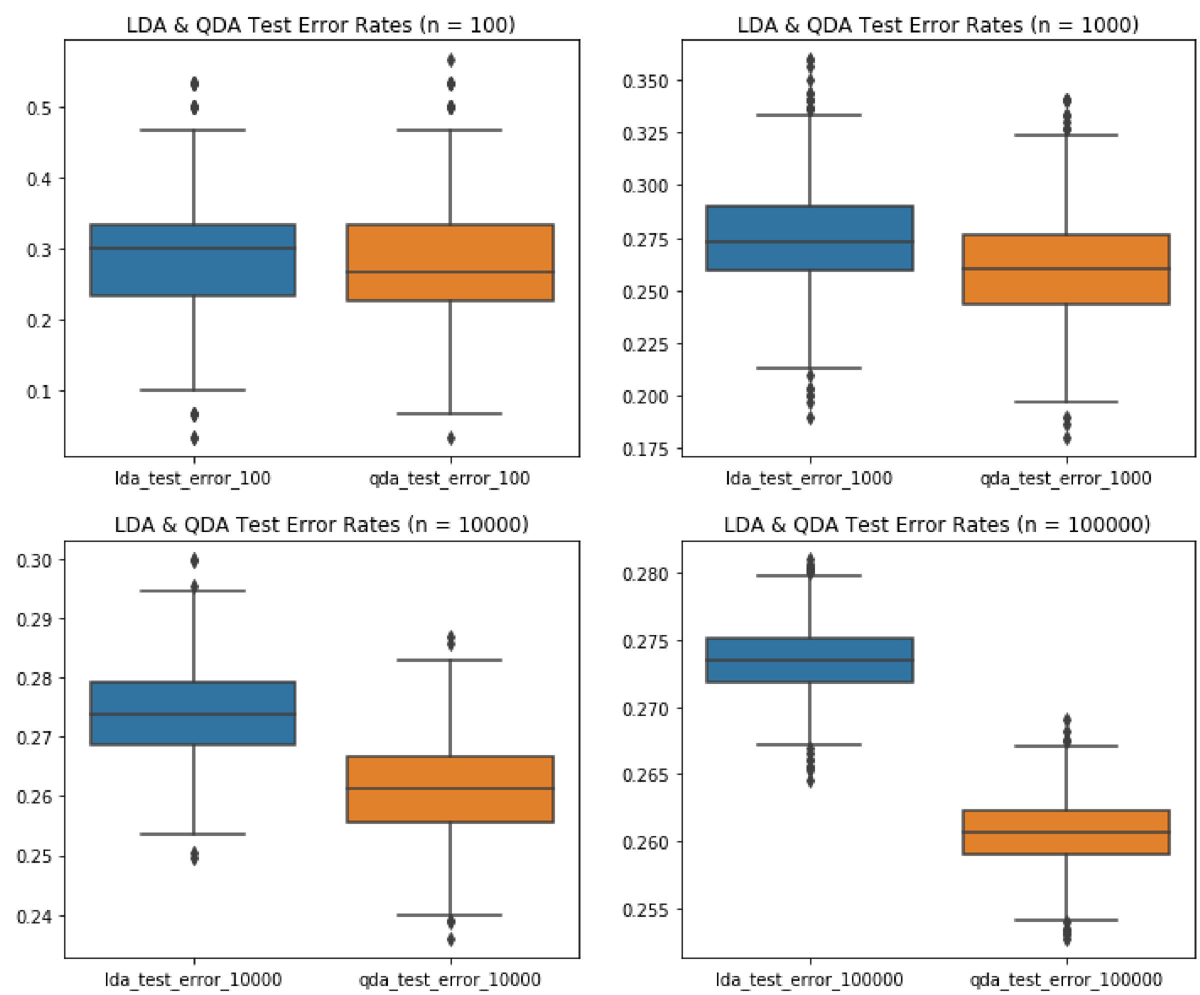
In [33]:
```
plt.figure(figsize=(10,8))
seaborn.boxplot(data=df[['qda_test_error_100','qda_test_error_1000',
                         'qda_test_error_10000','qda_test_error_100000'
]])
plt.ylabel('Error Rate')
plt.xticks(rotation=45)
plt.title('QDA Testing Errors for Different Sample Sizes')
```

Out[33]: Text(0.5, 1.0, 'QDA Testing Errors for Different Sample Sizes')



```
In [34]: fig= plt.figure(figsize=(12,10))
         ax1 = plt.subplot(221)
         seaborn.boxplot(data=df[['lda_test_error_100', 'qda_test_error_100']])
         plt.title('LDA & QDA Test Error Rates (n = 100)')
         ax2 = plt.subplot(222)
         seaborn.boxplot(data=df[['lda_test_error_1000', 'qda_test_error_1000']])
         plt.title('LDA & QDA Test Error Rates (n = 1000)')
         ax2 = plt.subplot(223)
         seaborn.boxplot(data=df[['lda_test_error_10000', 'qda_test_error_10000'
         ]])
         plt.title('LDA & QDA Test Error Rates (n = 10000)')
         ax2 = plt.subplot(224)
         seaborn.boxplot(data=df[['lda_test_error_100000', 'qda_test_error_10000
         0']])
         plt.title('LDA & QDA Test Error Rates (n = 100000)')
```

```
Out[34]: Text(0.5, 1.0, 'LDA & QDA Test Error Rates (n = 100000)')
```



As we can see above, for both the LDA and QDA average test error rates decrease as the sample sizes increase. However, the average test error rate for QDA decreases at a higher rate compared with that for LDA, which matches our expectation.

Modeling voter turnout

Question 5

In [35]:
```
df = pd.read_csv('mental_health.csv')
```

In [36]:
```
df.head()
```

Out[36]:

	vote96	mhealth_sum	age	educ	black	female	married	inc10
0	1.0	0.0	60.0	12.0	0	0	0.0	4.8149
1	1.0	NaN	27.0	17.0	0	1	0.0	1.7387
2	1.0	1.0	36.0	12.0	0	0	1.0	8.8273
3	0.0	7.0	21.0	13.0	0	0	0.0	1.7387
4	0.0	NaN	35.0	16.0	0	1	0.0	4.8149

In [84]:
```
df.dropna(inplace=True)
```

a.

In [78]:
```
y = df['vote96']
X = df[['mhealth_sum','age','educ','black','female','married','inc10']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

b.

In [79]:

```
# logistic regression model
log = LogisticRegression()
log.fit(X_train,y_train)
log_err = 1 - log.score(X_test, y_test)

# Linear discriminant model
lda = LDA()
lda.fit(X_train,y_train)
lda_err = 1 - lda.score(X_test, y_test)

# Quadratic discriminant model
qda = QDA()
qda.fit(X_train,y_train)
qda_err = 1 - qda.score(X_test, y_test)

# Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb_err = 1 - gnb.score(X_test, y_test)

# K-nearest neighbors with K = 1, 2, . . . , 10
def KNN(n):
    knn = KNeighborsClassifier(n_neighbors = n, metric = 'euclidean')
    knn.fit(X_train, y_train)
    knn_err = 1 - knn.score(X_test, y_test)
    return knn, knn_err
```

```
/Users/luyingjiang/anaconda3/lib/python3.7/site-packages/sklearn/linear
_model/logistic.py:432: FutureWarning: Default solver will be changed t
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [80]:

```
n1, n1_err = KNN(1)
n2, n2_err = KNN(2)
n3, n3_err = KNN(3)
n4, n4_err = KNN(4)
n5, n5_err = KNN(5)
n6, n6_err = KNN(6)
n7, n7_err = KNN(7)
n8, n8_err = KNN(8)
n9, n9_err = KNN(9)
n10, n10_err = KNN(10)
```

C.

i. Error Rate

In [81]:
```
print(tabulate([['Logistic Regression', log_err],
['LDA', lda_err],
['QDA', qda_err],
['Naive Bayes test error', gnb_err],
['KNN(n=1)', n1_err],
['KNN(n=2)', n2_err],
['KNN(n=3)', n3_err],
['KNN(n=4)', n4_err],
['KNN(n=5)', n5_err],
['KNN(n=6)', n6_err],
['KNN(n=7)', n7_err],
['KNN(n=8)', n8_err],
['KNN(n=9)', n9_err],
['KNN(n=10)', n10_err]],
headers = ['Type', 'Error Rate']))
```

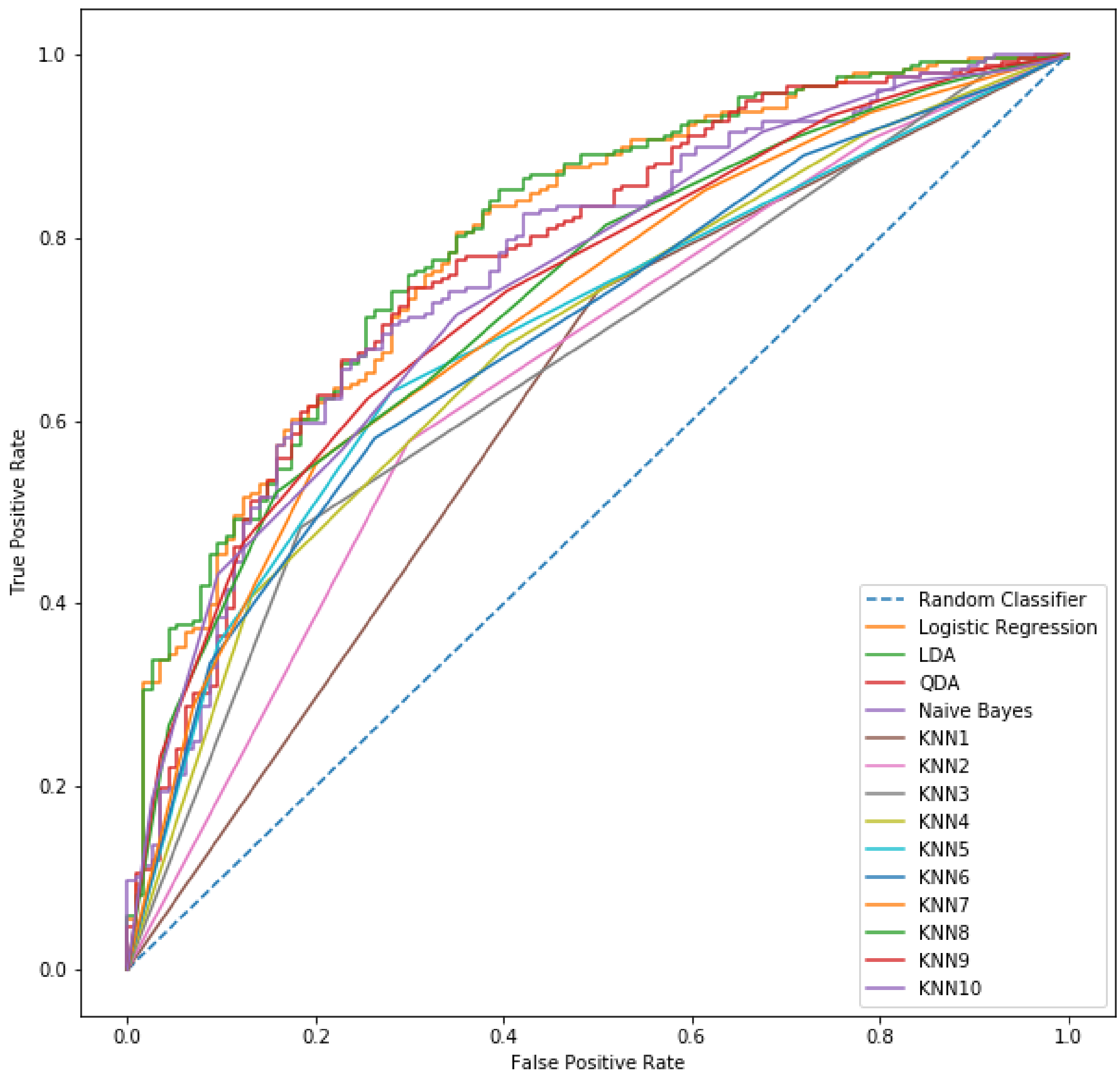
```
Type                      Error Rate
----------------------  ------------
Logistic Regression         0.251429
LDA                         0.245714
QDA                         0.277143
Naive Bayes test error      0.288571
KNN(n=1)                    0.337143
KNN(n=2)                    0.382857
KNN(n=3)                    0.354286
KNN(n=4)                    0.345714
KNN(n=5)                    0.331429
KNN(n=6)                    0.34
KNN(n=7)                    0.3
KNN(n=8)                    0.291429
KNN(n=9)                    0.297143
KNN(n=10)                   0.294286
```

ii. ROC curve(s) / Area under the curve (AUC)

In [82]:
```
def roc_auc(model, name):
    probs = model.predict_proba(X_test)[:, 1]
    auc = roc_auc_score(y_test, probs)
    print(name + ': AUC = %.2f' % (auc))
    fpr, tpr, _ = roc_curve(y_test, probs)
    plt.plot(fpr, tpr, label = name)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc = 'lower right')
```

In [85]:
```
models = [log, lda, qda, gnb, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10]
names = ['Logistic Regression', 'LDA', 'QDA', "Naive Bayes" ,
'KNN1','KNN2','KNN3','KNN4','KNN5','KNN6','KNN7','KNN8','KNN9','KNN10']
plt.figure(figsize=(12, 12))
rand_probs = [0] * len(y_test)
rand_fpr, rand_tpr, _ = roc_curve(y_test, rand_probs)
plt.plot(rand_fpr, rand_tpr, linestyle='--', label='Random Classifier')
for i, model in enumerate(models):
    roc_auc(model, names[i])
```

```
Logistic Regression: AUC = 0.79
LDA: AUC = 0.80
QDA: AUC = 0.77
Naive Bayes: AUC = 0.76
KNN1: AUC = 0.62
KNN2: AUC = 0.65
KNN3: AUC = 0.66
KNN4: AUC = 0.69
KNN5: AUC = 0.70
KNN6: AUC = 0.69
KNN7: AUC = 0.72
KNN8: AUC = 0.73
KNN9: AUC = 0.74
KNN10: AUC = 0.75
```



d. Which model performs the best? Be sure to define what you mean by “best” and identify supporting evidence to support your conclusion(s).

As shown above, in terms of the error rate, the best model is LDA. It has the lowest error rate. In terms of the ROC/AUC, LDA performs the best with its high AUC. Therefore, LDA is the best for this data set.