HW3

1) A function is convex if its second derivative ~~~~ should be positive. ~~~~ ~~~~

The log likelihood is given by.

$$J(\theta) = -\sum_{i=1}^{N} \left[y_i \log g(\theta^T x_i) + (1-y_i) \log (1 - g(\theta^T x_i)) \right]$$

Here $g(\theta^T x_i)$ is given by $\frac{1}{1+e^{-\theta^T x_i}}$

Using chain rule, we get.

$$g(z) = \frac{1}{1+e^{-z}} \quad ; \quad g'(z) = \frac{1}{(1+e^{-z})^2} \cdot e^{-z}$$

$$= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}} \right)$$

$$= g(z)\, g(1-z)$$

$$\frac{\partial}{\partial \theta} (-\log g(z)) = \frac{-g'(z)}{g(z)} \cdot \frac{\partial}{\partial \theta} \cdot z = (g(z)-1)\, x.$$

$$\frac{\partial^2}{\partial \theta^2} = (-\log g(\theta^T x)) = \frac{\partial}{\partial a} (g(z)-1)\, x.$$

$$= \frac{\partial}{\partial \theta} (g(z) . x) = g'(z) . x.$$

$$= g(z)(1-g(z)) \frac{\partial z}{\partial \theta} . x = g(z)(1-g(z)$$

$$\cdot x x^T \geq 0$$

$$\frac{\partial}{\partial \theta}\left(-\log\left(1-g(\theta^T x)\right)\right) = \frac{-1}{1-g(z)}\frac{\partial}{\partial \theta}\left(1-g(z)\right)$$

$$= \frac{-\left(-g(z)\left(1-g(z)\right)x\right.}{1-g(z)} = g(z)x.$$

$$\frac{\partial^2}{\partial \theta^2}\left(-\log\left(1-g(\theta^T x)\right)\right) = \frac{\partial\left(g(z)x\right)}{\partial \theta} = x\,g(z)\left(1-g(z)\right)x^T \geq 0$$

The xx^T term in both are squared term implying it can never be negative.

Since both the terms second derivative have xx^T and are therefore greater than 0, ~~thus~~ the function as a whole is convex.

2

a). X_1 and X_2 are continous.

$$\pi_J\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right) = \frac{e^{\alpha_J + B'_J\begin{bmatrix}x_1\\x_2\end{bmatrix}}}{\sum_{j=1}^{J} e^{(\alpha + B)'_j\begin{bmatrix}x_1\\x_2\end{bmatrix}}}.$$

For three classes and $J=3$, each class will have.

$3\times(J-1)$ parameters making that $\underline{\underline{6}}$

b). Assuming X_1 is catagorical. with 5 categories and X_2 is continous

Softmax would be. $$\pi_J(x) = \frac{e^{\alpha_J + B^T x}}{\sum_{j=1}^{\hat{}} e^{\alpha_j + B^T x}}.$$

There are 5 binary one hot variables. needed to represent the categories.

Besides that X_2 which is continous making a total of 6.

$$6\times(J-1) = 6\times 2 = \underline{\underline{12}}$$

4

a) $P(Y=1 \mid x_1=1,\ x_2=0,\ x_3=1)$.

using bayes rule $\quad P(A \mid B) = \dfrac{P(B \mid A) \cdot P(A)}{P(B)}$

$P(Y=1) = 4/7$

$P(x_1=1) = 4/7 \quad P(x_1=1 \mid Y=1) = 3/4$.

$P(x_2=0) = 3/7 \quad P(x_2=0 \mid Y=1) = 1/4$.

$P(x_3=1) = 4/7 \quad P(x=1 \mid Y=1) = 1/4$.

$$P(Y=1 \mid X_i=1, x_2=0, x_3=1) = \frac{4/7 \cdot 3/4 \cdot 1/4 \cdot 1/4}{3/7 \cdot 4/7 \cdot 4/7}$$

$$= \frac{49}{256} = 0.19.$$

$P(Y=1 \mid x_1=1,\ x_2=1, x_3=1)$.

$P(Y=1) = 4/7$.

$P(x_1=1) = 4/7, \quad P(x_1=1 \mid Y=1) = 3/4$.

$P(x_2=1) = 4/7, \quad P(x_2=1 \mid Y=1) = 3/7$.

$P(x_3=1) = 4/7, \quad P(x_3=1 \mid Y=1) = 1/4$

$$P = \frac{3/7 \cdot 3/4 \cdot 1/4}{4/7 \cdot 4/7 \cdot 4/7} = \frac{441}{1024} = 0.43$$

b). $P(Y=1 \mid x_1=1,\ x_2=0,\ x_3=1)$.

$P(Y=1) = 4/7$

$P(x_1 \mid x_2=0,\ x_3=1,\ Y=1) = 0$

$P(x_2 \mid x_3=1,\ Y=1) = 0$

$P(x_3 \mid Y=1) = 1/4$.

$P(Y=1 \mid x_1=1,\ x_2=0,\ x_3=1) = 0 \cdot 4/7 \cdot 1/4 = \underline{\underline{0}}$

$P(Y=1 \mid x_1=1,\ x_2=1,\ x_3=1)$.

$P(Y=1) = 4/7$.

$P(x_1=1 \mid x_2=1,\ x_3=1,\ Y=1) = 0$

$P(x_2=1 \mid x_3=1,\ Y=1) = 1$

$P(x_3=1,\ Y=1) = 1/4$.

$P(Y=1 \mid x_1=1,\ x_2=1,\ x_3=1) = 0 \cdot 1/4 \cdot 4/7 = \underline{0}$

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import random
from sklearn import linear_model
import scipy.stats
from sklearn.metrics import accuracy_score
np.set_printoptions(suppress=True)
```

In [2]:

```
def genData(numPoints):
    x = np.zeros(shape=(numPoints, 3))
    y = np.zeros(shape=numPoints)

    x[:,0] = 1
    #x[:,1] = [i for i in range(numPoints)]
    x[:,1] = np.random.uniform(0, 3, size = numPoints)
    x[:,2] = np.random.uniform(0, 3, size = numPoints)



    for i in range(0, numPoints): y[i] =  1/(1 + np.exp( - (-3 + (x[i][1]) + (x[
i][2]))))
    #for i in range(0, numPoints): y[i] =  1/(1 + np.exp( - (-3 + (x[i][1]))))


    y = np.array([1 if i > 0.5 else 0 for i in y])
    return x, y
```

In [3]:

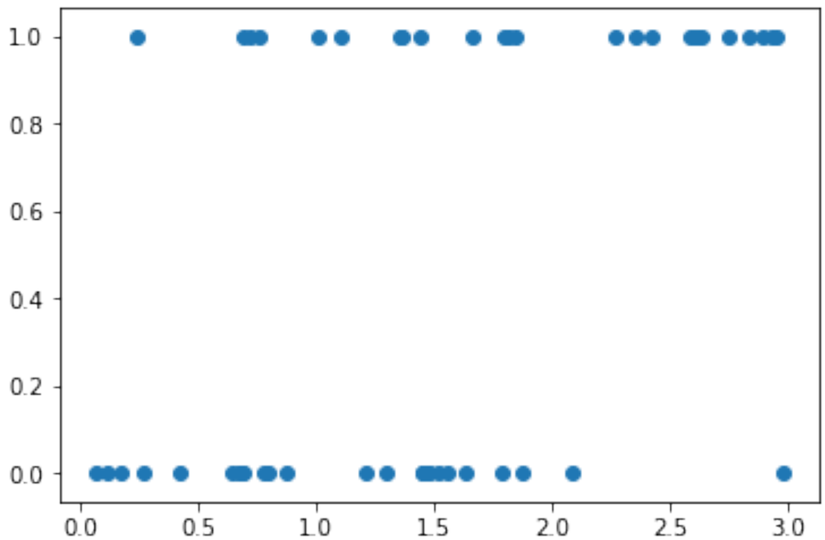
```
x,y = (genData(50))
#y_temp = np.array([1 if i > 0.5 else 0 for i in y])
```

In [4]:

```
plt.scatter(x[:,1], y)
```

Out[4]:

<matplotlib.collections.PathCollection at 0x7ff2ba97cd30>



In [5]:

```
def gradientDescent(x, y, theta, alpha, m, numIterations):

    xTrans = x.transpose()

    for i in range(0, numIterations):

        loss = (1 / (1 + np.exp(-(1 + (np.dot(x, theta)))))) - y

        #print (np.mean(loss))

        #gradient = np.dot(xTrans,  np.dot(x, theta) - y) / m #Partial derivativ
e

        gradient = np.dot(xTrans, loss) / m
        theta = theta - alpha * gradient


    return theta
```

In [6]:

```
m, n = np.shape(x)

theta = gradientDescent(x,y, np.ones(n), 1, m, 70000)
```

In [7]:

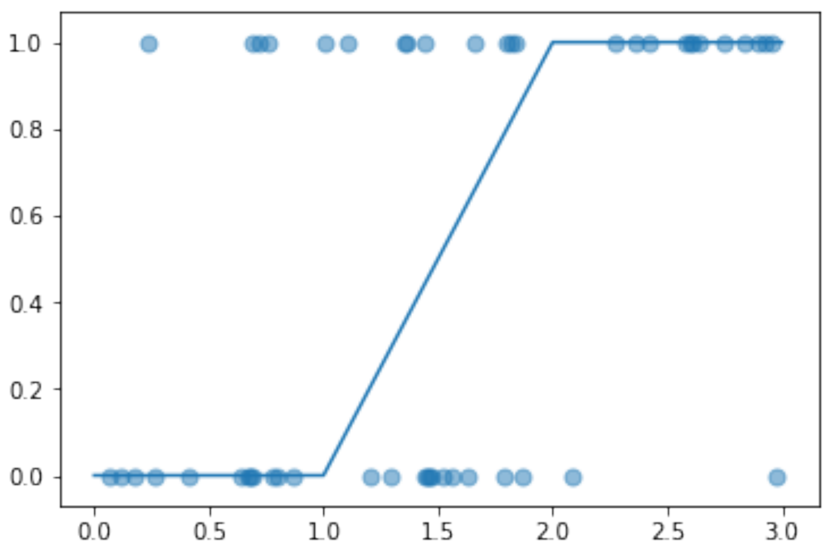
```
theta
```

Out[7]:

```
array([-66.12729013,  20.52518735,  21.32420216])
```

In [8]:

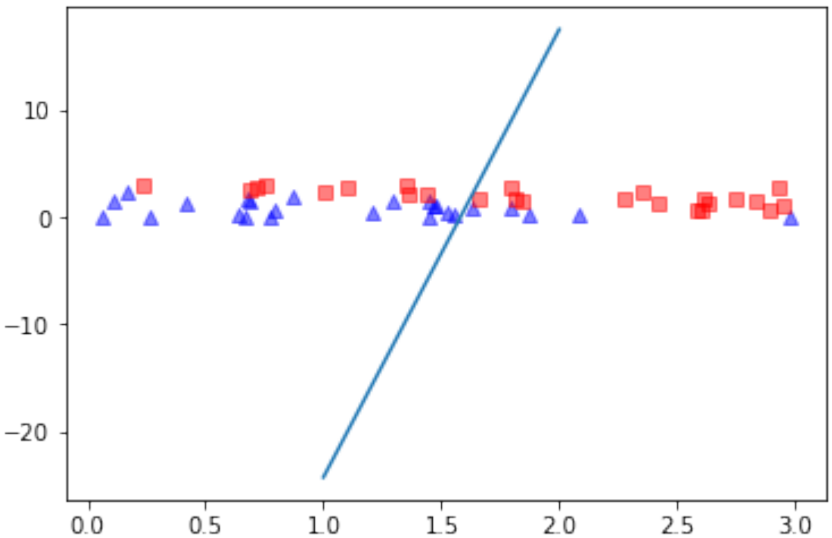
```
hypothesis = lambda x: 1 / (1 + (np.exp(-(x * theta[1] + x * theta[2]+ theta[0])
)))

plt.plot([i for i in range(0,4)],[hypothesis(i) for i in range(0,4)])

plt.scatter([i[1] for i in x], y, s=50, alpha = 0.5)
plt.show()
```



Decision boundaries for X[1] vs X[2] and X[1] vs X[0] respectively for Batch GD

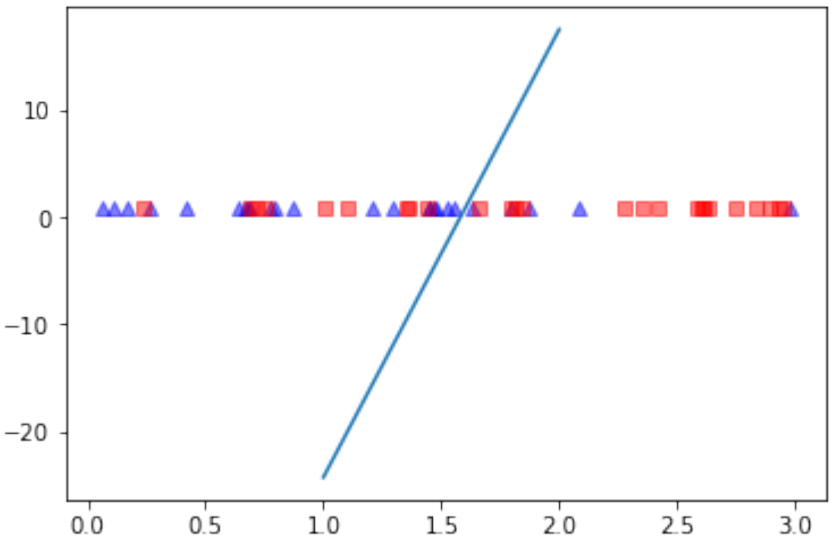
In [9]:

```
d_boundary_plot = lambda x: x*theta[1] + x*theta[2] + theta[0]
plt.plot([i for i in range(1,3)],[d_boundary_plot(i) for i in range(1,3)])

#plt.scatter([i[1] for i in x], [i[2] for i in x], s=50, alpha = 0.5)

for label,marker,color in zip(range(0,2),('^', 's'),('blue', 'red')):

        plt.scatter(x=x[:,1].real[y == label],
                y=x[:,2].real[y == label],
                marker=marker,
                color=color,
                alpha=0.5
                )
```



In [10]:

```
d_boundary_plot = lambda x: x*theta[1] + x*theta[2] + theta[0]
plt.plot([i for i in range(1,3)],[d_boundary_plot(i) for i in range(1,3)])

#plt.scatter([i[1] for i in x], [i[2] for i in x], s=50, alpha = 0.5)

for label,marker,color in zip(range(0,2),('^', 's'),('blue', 'red')):

        plt.scatter(x=x[:,1].real[y == label],
                y=x[:,0].real[y == label],
                marker=marker,
                color=color,
                alpha=0.5
                )
```



In [11]:

```
def gradientDescentSto(x, y, theta, alpha, m, numIterations, delta=2, conv=0.000
00001):

    xTrans = x.transpose()
    count = 0
    theta_prev = theta + delta + 1

    while (count < numIterations):

            count += 1

            for i in range(m):

                hypothesis = x[i][0]*theta[0] + x[i][1]*theta[1] + x[i][2]*theta
[2]

                loss = 1/(1 + np.exp(- hypothesis)) - y[i]

                #loss = hypothesis - y[i]

                gradient1 = x[i][0] * loss
                gradient0 = x[i][1] * loss

                theta_prev = theta
                theta[1] = theta[1] - alpha * gradient0
                theta[0] = theta[0] - alpha * gradient1


    return theta
```

In [12]:

```
theta = gradientDescentSto(x, y, theta, 0.08, m, 10000)
```

In [13]:

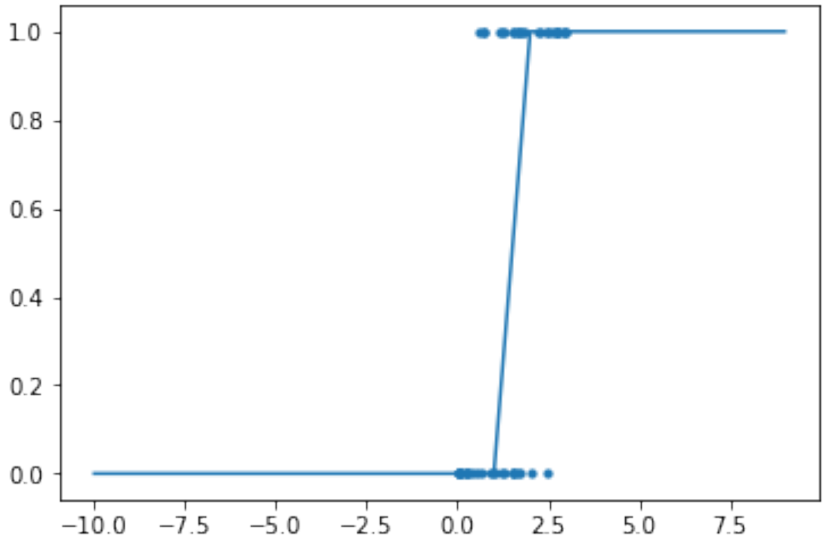
```
theta
```

Out[13]:

```
array([-65.34984189,  20.63300455,  21.32420216])
```

In [14]:

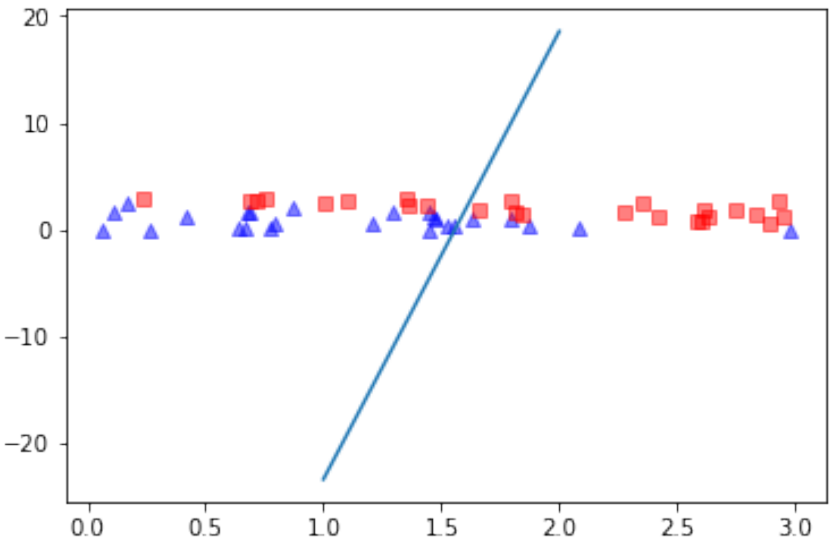
```
hypothesis = lambda x: 1 / (1 + (np.exp(-(x * theta[1] + x * theta[2] + theta[0]
))))

plt.plot([i for i in range(-10,10)],[hypothesis(i) for i in range(-10,10)])
plt.scatter([i[2] for i in x], y, s=10)
plt.show()
```



Decision boundaries for X[1] vs X[2] and X[1] vs X[0] respectively for Stochastic GD

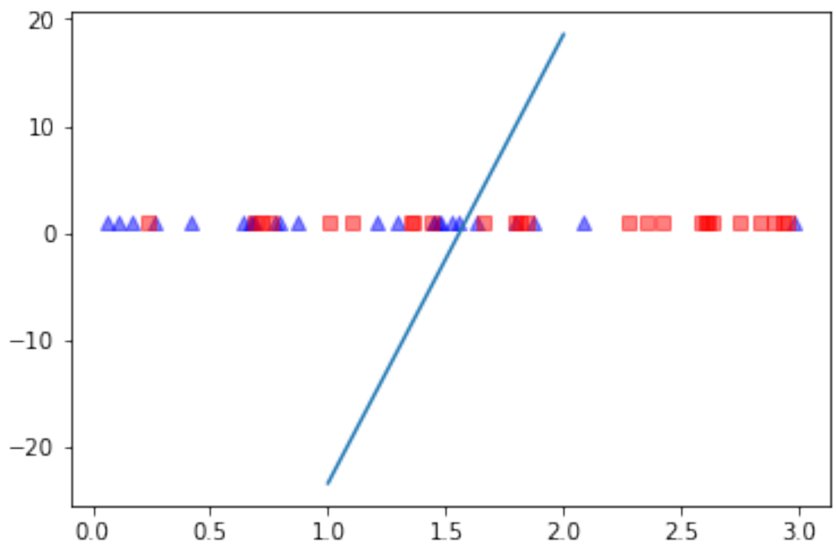
In [15]:

```
d_boundary_plot = lambda x: x*theta[1] + x*theta[2] + theta[0]
plt.plot([i for i in range(1,3)],[d_boundary_plot(i) for i in range(1,3)])

#plt.scatter([i[1] for i in x], [i[2] for i in x], s=50, alpha = 0.5)

for label,marker,color in zip(range(0,2),('^', 's'),('blue', 'red')):

        plt.scatter(x=x[:,1].real[y == label],
                y=x[:,2].real[y == label],
                marker=marker,
                color=color,
                alpha=0.5
                )
```



In [16]:

```
d_boundary_plot = lambda x: x*theta[1] + x*theta[2] + theta[0]
plt.plot([i for i in range(1,3)],[d_boundary_plot(i) for i in range(1,3)])

#plt.scatter([i[1] for i in x], [i[2] for i in x], s=50, alpha = 0.5)

for label,marker,color in zip(range(0,2),('^', 's'),('blue', 'red')):

        plt.scatter(x=x[:,1].real[y == label],
                y=x[:,0].real[y == label],
                marker=marker,
                color=color,
                alpha=0.5
                )
```



In [17]:

```
d_boundary = lambda x: x[1]*theta[1] + x[2]*theta[2] + theta[0] * x[0]
y_hat = [1 if i > 0 else 0 for i in [d_boundary(i) for i in x]]
```

In [18]:

```
accuracy_score(y, y_hat)
```

Out[18]:

```
1.0
```

Linear discriminant analysis

In [19]:

```
x_old = x
x = x[:, 1:]
```

In [20]:

```
mean_vs = []
y_temp = np.array([1 if i > 0.5 else 0 for i in y])
for i in range(0,2):mean_vs.append(np.mean(x[y_temp==i], axis = 0))
print(mean_vs)
```

```
[array([1.11687638, 0.83202265]), array([1.90450382, 1.97216123])]
```

In [21]:

```
within_class = np.zeros((2,2))

for cl,mv in zip(range(0,2), mean_vs):

    class_scatter_matrix = np.zeros((2,2))
    for row in x[y_temp == cl]:
        row, mv = row.reshape(2,1), mv.reshape(2,1)
        class_scatter_matrix += (row-mv).dot((row-mv).T)
    within_class += class_scatter_matrix

within_class
```

Out[21]:

```
array([[ 28.46191249, -14.49320429],
       [-14.49320429,  26.81805413]])
```

In [22]:

```
mean = np.mean(x, axis=0)
between_class = np.zeros((2,2))

for i,mean_vec in enumerate(mean_vs):
    n = x[y_temp==i+1,:].shape[0]
    mean_vec = mean_vec.reshape(2,1)
    overall_mean = mean.reshape(2,1)
    between_class += n * (mean_vec - mean).dot((mean_vec - mean).T)

between_class
```

Out[22]:

```
array([[ 5.91092904, 10.74652962],
       [10.74652962, 19.63921255]])
```

In [23]:

```
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(within_class).dot(between_class
))
for i in range(len(eig_vals)): eigvec_sc = eig_vecs[:,i].reshape(2,1)
print(eig_vals, eig_vecs)
```

```
[0.00058142 1.85935676] [[-0.87693265 -0.6255713 ]
 [ 0.48061328 -0.780167  ]]
```

In [29]:

```
eig_pairs = sorted([(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_
vals))]
                   , key=lambda k: k[0], reverse=True)
eig_pairs
```

Out[29]:

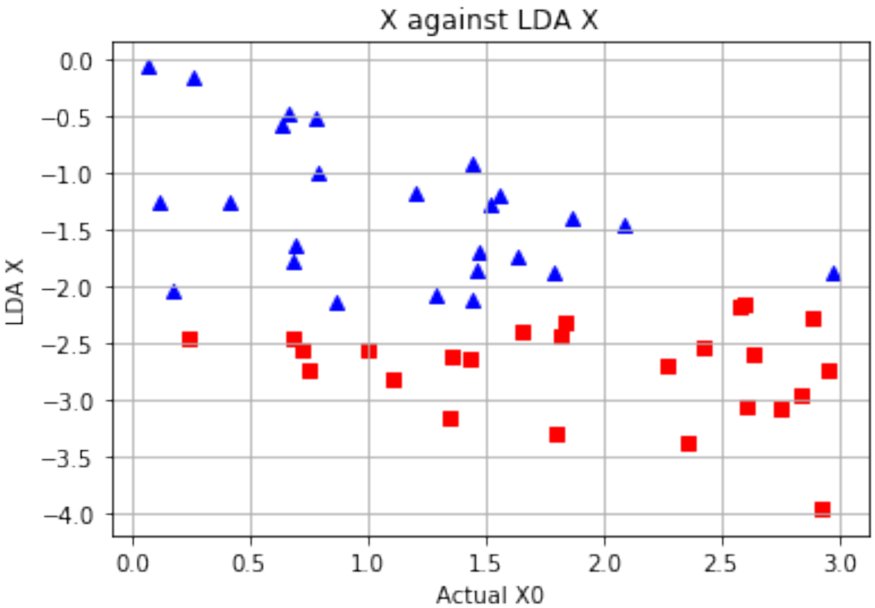
```
[(1.859356764932039, array([-0.6255713, -0.780167 ])),
 (0.0005814232087433258, array([-0.87693265,  0.48061328]))]
```

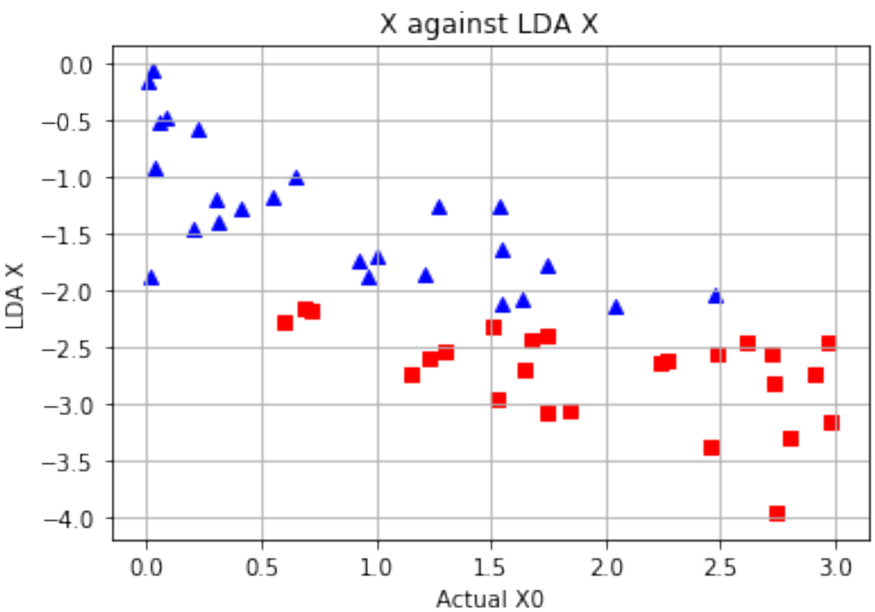
In [33]:

```
W = eig_pairs[0][1].reshape(2,1)
x_lda = x.dot(W)
```

In [41]:

```
from matplotlib import pyplot as plt

def plot_step_lda(feature):

    ax = plt.subplot(111)
    for label,marker,color in zip(range(0,2),('^', 's', 'o'),('blue', 'red', 'gr
een')):

        plt.scatter(x=x[:,feature].real[y == label],
                y=x_lda[:,0].real[y == label],
                marker=marker, color=color)

    plt.xlabel('Actual X0')
    plt.ylabel('LDA X')
    plt.title('X against LDA X')


    plt.grid()
    plt.tight_layout
    plt.show()

plot_step_lda(0)
plot_step_lda(1)
```

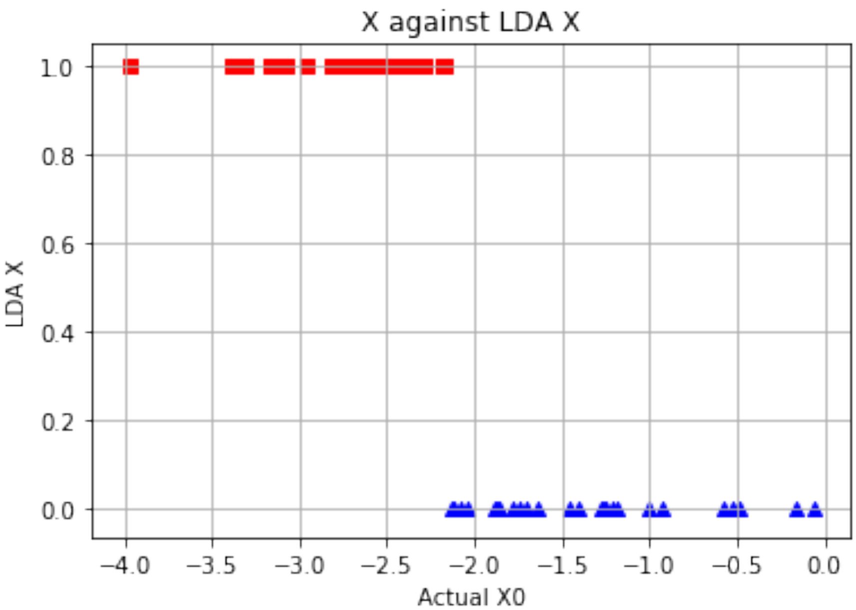




The above plot shows the relationship between actual X and LDA transformed X. Both X0 and X1 are shown to be linearly seperated at X ~ 2.0

In [42]:

```
ax = plt.subplot(111)
for label,marker,color in zip(range(0,2),('^', 's'),('blue', 'red')): plt.scatte
r(x=x_lda[:,0].real[y == label],y=y.real[y == label],marker=marker,color=color)

plt.xlabel('Actual X0')
plt.ylabel('LDA X')
plt.title('X against LDA X')


plt.grid()
plt.tight_layout
plt.show()
```



There is a clean seperation once LDA is applied to X

In [584]:

```
y_hat = [1 if i>=2 else 0 for i in x_lda]
accuracy_score(y, y_hat)
```

Out[584]:

```
0.96
```

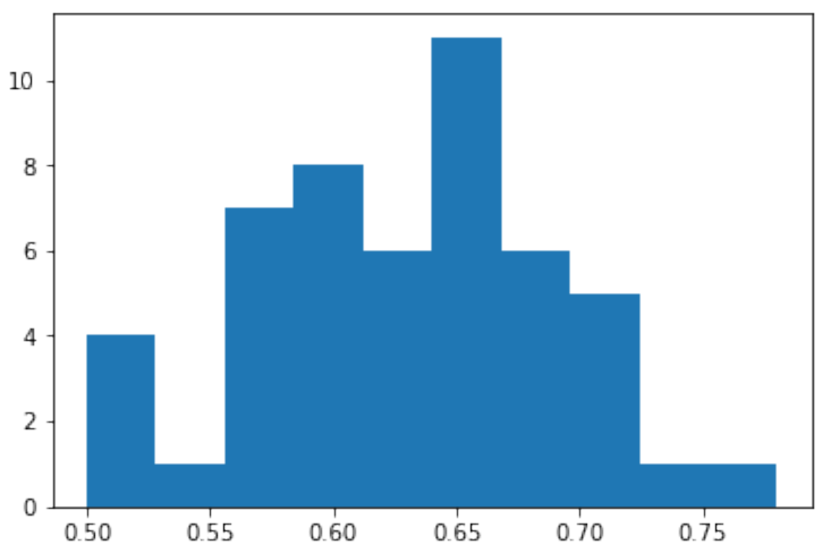
Histograms of the three methods

In [609]:

```
accuracies_gd = []

for run in range(50):

    x,y = (genData(50))
    m, n = np.shape(x)
    theta = gradientDescent(x,y, np.ones(n), 1, m, 70000)

    d_boundary = lambda x: x[0]*theta[1] + x[1]*theta[2] + x[2]*theta[0]


    y_hat = [0 if i > 0 else 1 for i in [d_boundary(i) for i in x]]
    accuracies_gd.append(accuracy_score(y, y_hat))

plt.hist(np.array(accuracies_gd), label = 'GD')
```

Out[609]:

```
(array([ 4.,  1.,  7.,  8.,  6., 11.,  6.,  5.,  1.,  1.]),
 array([0.5  , 0.528, 0.556, 0.584, 0.612, 0.64 , 0.668, 0.696, 0.72
4,
        0.752, 0.78 ]),
 <a list of 10 Patch objects>)
```



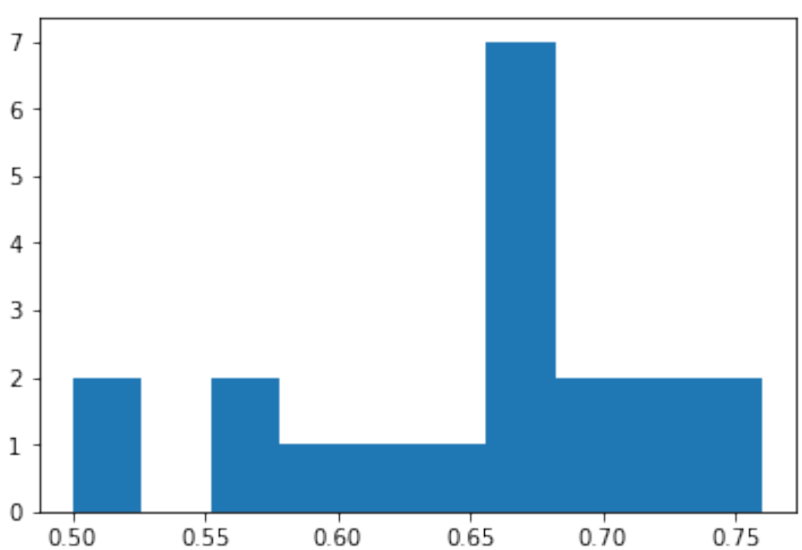
In [620]:

```
accuracies_sto = []

for run in range(20):

    print (run)
    x,y = (genData(50))
    m, n = np.shape(x)
    theta = gradientDescentSto(x,y, np.ones(n), 1, m, 10000)

    d_boundary = lambda x: x[0]*theta[1] + x[1]*theta[2] + x[2]*theta[0]


    y_hat = [0 if i > 0 else 1 for i in [d_boundary(i) for i in x]]
    accuracies_sto.append(accuracy_score(y, y_hat))

plt.hist(np.array(accuracies_sto), label = 'GD')
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Out[620]:

```
(array([2., 0., 2., 1., 1., 1., 7., 2., 2., 2.]),
 array([0.5  , 0.526, 0.552, 0.578, 0.604, 0.63 , 0.656, 0.682, 0.70
8,
        0.734, 0.76 ]),
 <a list of 10 Patch objects>)
```



In [45]:

```
accuracies_lda = []

for run in range(200):

    x,y = (genData(50))
    x = x[:, 1:]

    mean_vs = []
    y_temp = np.array([1 if i > 0.5 else 0 for i in y])
    for i in range(0,2):mean_vs.append(np.mean(x[y_temp==i], axis = 0))
    print(mean_vs)

    within_class = np.zeros((2,2))

    for cl,mv in zip(range(0,2), mean_vs):

        class_scatter_matrix = np.zeros((2,2))
        for row in x[y_temp == cl]:
            row, mv = row.reshape(2,1), mv.reshape(2,1)
            class_scatter_matrix += (row-mv).dot((row-mv).T)
        within_class += class_scatter_matrix


    overall_mean = np.mean(x, axis=0)

    mean = np.mean(x, axis=0)
    between_class = np.zeros((2,2))

    for i,mean_vec in enumerate(mean_vs):
        n = x[y_temp==i+1,:].shape[0]
        mean_vec = mean_vec.reshape(2,1)
        overall_mean = mean.reshape(2,1)
        between_class += n * (mean_vec - mean).dot((mean_vec - mean).T)

    eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(within_class).dot(between_c
lass))
    for i in range(len(eig_vals)): eigvec_sc = eig_vecs[:,i].reshape(2,1)

    eig_pairs = sorted([(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(
eig_vals))]
                      , key=lambda k: k[0], reverse=True)


    W = eig_pairs[0][1].reshape(2,1)
    x_lda = x.dot(W)

    y_hat = [1 if i>=2 else 0 for i in x_lda]
    accuracies_lda.append(accuracy_score(y, y_hat))

plt.hist(np.array(accuracies_lda), label = 'GD')
```

```
[array([0.83929704, 1.00614434]), array([2.26066898, 1.98306645])]
[array([1.06232932, 1.05398051]), array([1.69221586, 2.20116009])]
[array([0.8595049 , 1.12125005]), array([1.90860151, 2.04333948])]
[array([0.66273389, 1.12640724]), array([2.0469637 , 1.79945824])]
[array([1.39749888, 0.72031956]), array([1.97557825, 1.95054578])]
[array([1.16347381, 1.01042746]), array([2.07469372, 1.86183587])]
[array([0.98200943, 0.92036327]), array([1.80268477, 2.2628307 ])]
[array([1.13439717, 1.09243617]), array([2.17336398, 2.28375346])]
[array([1.25256252, 0.97733217]), array([1.93141848, 2.28168542])]
[array([0.84419823, 0.91675769]), array([2.26498206, 1.89756332])]
[array([0.73760576, 1.41559017]), array([2.01185166, 1.8043372 ])]
[array([1.19567031, 0.6815785 ]), array([2.25383607, 1.77951421])]
[array([1.13134132, 1.07579967]), array([1.98603291, 1.97414516])]
[array([1.08014185, 0.9773905 ]), array([2.1687166 , 2.05693904])]
[array([0.92437414, 1.27888806]), array([1.77244258, 2.20451583])]
[array([0.94577157, 0.82670936]), array([2.10106905, 1.89203975])]
[array([1.04056687, 0.86694862]), array([1.84393481, 2.1233452 ])]
[array([1.1376197 , 1.01674285]), array([1.95185591, 1.80288235])]
[array([0.79833992, 1.04388146]), array([1.99347243, 2.13186886])]
[array([1.11348563, 0.92840595]), array([2.16059205, 1.77703197])]
[array([1.17397058, 0.7901681 ]), array([2.05593733, 1.98202127])]
[array([1.02757846, 1.06142999]), array([2.05920861, 1.95480949])]
[array([1.09639654, 0.80886487]), array([1.92884076, 1.97198275])]
[array([1.03332626, 0.88547872]), array([2.0162863, 2.060518 ])]
[array([1.01642599, 0.94183089]), array([1.96869499, 2.05641464])]
[array([0.8334427 , 0.79823615]), array([2.21331414, 1.83948053])]
[array([1.0974606 , 0.92377971]), array([2.01187447, 2.04512918])]
[array([0.90334048, 1.13076672]), array([2.11443585, 1.97782092])]
[array([0.94844166, 0.98771563]), array([2.23343228, 1.69019859])]
[array([1.02636027, 1.28305872]), array([1.97338957, 2.02634463])]
[array([1.00490103, 0.99833925]), array([1.96769705, 1.75880355])]
[array([1.04004681, 0.98802631]), array([1.90451316, 2.00244706])]
[array([0.8371501 , 1.10444209]), array([1.84189095, 2.16167476])]
[array([1.14778684, 0.95436929]), array([2.13162302, 1.83861561])]
[array([0.76344194, 1.33282895]), array([1.96857004, 1.95690778])]
[array([0.93710826, 1.15980332]), array([1.88983318, 2.02440109])]
[array([1.07358488, 1.07455568]), array([1.95049053, 2.13105952])]
[array([0.99597244, 1.20381687]), array([1.91977011, 2.21975561])]
[array([0.98242316, 0.89770331]), array([2.11831111, 2.1170304 ])]
[array([0.81203613, 1.10178435]), array([1.94710593, 1.90929428])]
[array([0.89721168, 1.0803754 ]), array([1.9731049, 1.9711887])]
[array([0.95514714, 0.96270942]), array([2.05381239, 1.7455297 ])]
[array([1.01984562, 0.9033315 ]), array([1.91527404, 2.12371993])]
[array([0.94577072, 1.02794188]), array([1.72712178, 2.23386794])]
[array([1.16062009, 0.79901388]), array([1.8363361 , 2.17504166])]
[array([0.93329937, 0.97759638]), array([2.11606276, 1.78148375])]
[array([0.9986055 , 1.07649602]), array([2.24285238, 1.70709711])]
[array([0.86272636, 0.99789699]), array([2.16081022, 1.87144328])]
[array([0.98013833, 1.04247989]), array([2.15355015, 1.82210936])]
[array([1.04280776, 1.00771868]), array([2.02068867, 1.91002787])]
[array([1.02507636, 0.87423156]), array([2.01317073, 2.0864678 ])]
[array([0.80640918, 1.04962882]), array([2.12106105, 1.8754633 ])]
[array([1.05425878, 0.90262161]), array([1.96071607, 2.10267842])]
[array([0.78469714, 1.1424283 ]), array([1.9980087 , 2.19465726])]
```

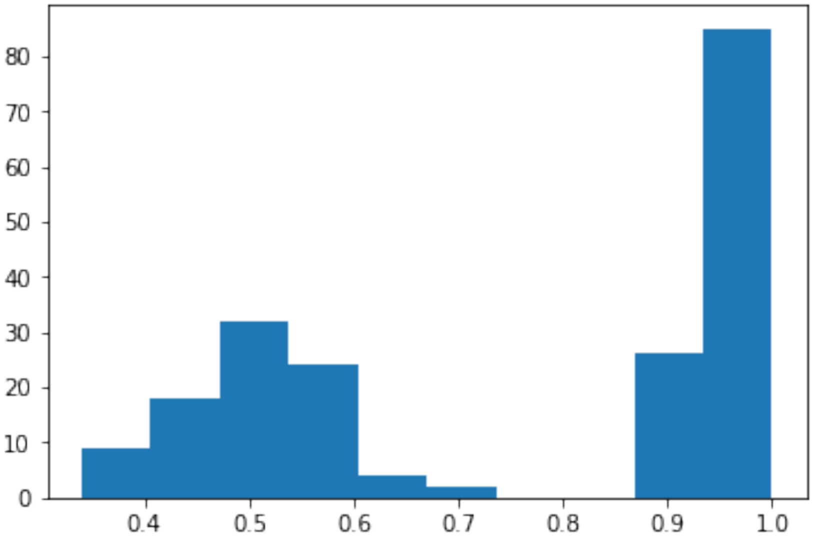
```
[array([1.0187896 , 0.95930482]), array([2.03487846, 2.00838616])]
[array([0.94920769, 1.09558346]), array([2.04181521, 1.96686198])]
[array([0.9850755 , 1.05782142]), array([2.05664234, 1.96258126])]
[array([0.97467426, 0.92362596]), array([1.95959361, 2.22828744])]
[array([1.01354359, 0.88067864]), array([1.7399228 , 2.15377644])]
[array([1.03096888, 0.95656048]), array([2.01971526, 2.27358889])]
[array([0.83884659, 1.00104054]), array([2.2737725 , 1.81754569])]
[array([0.93150606, 1.01448678]), array([1.87034135, 1.88970036])]
[array([0.98982547, 1.12657339]), array([1.75187528, 2.14249861])]
[array([1.08722262, 1.12176508]), array([1.76054122, 2.22835194])]
[array([0.89317817, 1.32315068]), array([2.14885953, 1.81801011])]
[array([0.96495706, 1.05547138]), array([2.06152634, 1.98093605])]
[array([1.06401855, 0.91694693]), array([1.94244596, 1.99419318])]
[array([1.1576754 , 0.77499116]), array([1.74686025, 2.20172313])]
[array([0.8644793 , 0.87279762]), array([1.89238983, 1.98834183])]
[array([0.77768114, 1.20253381]), array([1.99136713, 1.95884892])]
[array([1.13023102, 0.91269429]), array([2.17933597, 1.7709569 ])]
[array([0.98295646, 1.03465817]), array([2.04238763, 2.01288952])]
[array([0.94075902, 1.11511166]), array([2.01883025, 2.02861991])]
[array([0.8374107, 0.8967099]), array([2.22148945, 1.72178718])]
[array([0.74149054, 1.23793015]), array([2.29515958, 2.05398902])]
[array([0.75098503, 1.15422086]), array([1.96091443, 2.05211366])]
[array([0.94744598, 1.21940906]), array([1.87685268, 2.03723423])]
[array([1.31144042, 0.74843565]), array([2.17798531, 1.80851183])]
[array([0.98119353, 1.04024666]), array([1.98242932, 1.88409487])]
[array([1.02698106, 1.09791466]), array([1.78598718, 2.14682561])]
[array([1.28609271, 1.06179867]), array([1.86549669, 2.03757217])]
[array([0.9776009, 0.9505665]), array([2.04611714, 1.90512287])]
[array([1.13917281, 0.79774233]), array([1.9931527 , 1.94452213])]
[array([1.08768423, 0.84190773]), array([1.91524911, 1.94332417])]
[array([0.95661743, 1.11908085]), array([2.19042843, 1.72685068])]
[array([0.97447827, 1.14358185]), array([1.72282137, 2.26676241])]
[array([0.97544354, 1.15446694]), array([2.2433486 , 1.79430862])]
[array([0.69744925, 1.09948165]), array([2.10338083, 1.98459342])]
[array([0.88042985, 1.27256599]), array([2.18749136, 2.01143858])]
[array([1.14251272, 1.06024899]), array([2.00683961, 2.02427038])]
[array([1.00316244, 1.17710026]), array([2.09866005, 2.12738189])]
[array([0.87965253, 0.99474752]), array([2.07822984, 1.93459654])]
[array([1.08692177, 0.76609084]), array([1.95387412, 2.39359852])]
[array([0.94488541, 1.13389773]), array([2.00203541, 2.04066981])]
[array([1.00345647, 0.82361179]), array([1.96617699, 1.88962846])]
[array([0.97290481, 0.9284181 ]), array([2.08552645, 1.93277709])]
[array([1.06809917, 0.7981405 ]), array([2.01615291, 2.05246596])]
[array([0.91259064, 1.00628098]), array([1.913272 , 2.0471595])]
[array([1.0124558 , 0.76692204]), array([1.88043352, 2.06031986])]
[array([1.17310866, 0.88368542]), array([1.94794595, 2.30651371])]
[array([0.97487933, 0.97795637]), array([1.92609561, 1.81397593])]
[array([0.9804472 , 0.88594997]), array([2.04196922, 2.12466676])]
[array([1.11902342, 1.08154952]), array([2.1135933 , 2.15013116])]
[array([1.17049173, 1.11846485]), array([1.99499374, 1.89668078])]
[array([0.87767549, 0.84794846]), array([1.98862296, 1.93827141])]
[array([0.84501108, 0.9970872 ]), array([2.24067081, 1.72593652])]
[array([0.98223638, 0.98321692]), array([1.98446481, 2.06428485])]
```

```
[array([1.14555919, 0.8320302 ]), array([1.94353021, 2.00167603])]
[array([0.84081072, 1.31219657]), array([2.04582929, 2.06110463])]
[array([1.31290758, 0.85092762]), array([1.88530945, 2.05148955])]
[array([0.98200676, 0.97215319]), array([1.9541728 , 2.17860249])]
[array([0.87985607, 1.033894  ]), array([2.04762923, 2.36775863])]
[array([0.99889289, 0.87143169]), array([2.09420053, 2.07468455])]
[array([0.96729998, 1.11220999]), array([2.13481332, 1.87460162])]
[array([1.14626346, 0.95527101]), array([2.10715627, 1.74832571])]
[array([0.92476209, 1.08626494]), array([2.07345521, 2.02551452])]
[array([1.02310744, 1.10140118]), array([2.02371936, 1.9177891 ])]
[array([0.97173812, 0.94507005]), array([1.97841746, 2.09880763])]
[array([0.96617337, 1.01694151]), array([1.91933126, 2.23545936])]
[array([0.95414165, 0.83551079]), array([2.05350838, 2.15319374])]
[array([0.74665858, 1.27294339]), array([1.93638029, 2.1038933 ])]
[array([0.90545211, 1.17648914]), array([2.03631846, 1.96281025])]
[array([1.00136926, 1.14853625]), array([2.18522938, 2.00388689])]
[array([1.00269073, 1.12438708]), array([2.1793768 , 1.66953849])]
[array([0.98175687, 0.810882  ]), array([2.00232881, 1.91801269])]
[array([1.15727498, 0.93427923]), array([1.98375837, 1.95206165])]
[array([0.96870911, 0.97495181]), array([2.07525249, 1.99435207])]
[array([0.92732918, 1.04948086]), array([2.24667283, 1.94216583])]
[array([0.93137288, 1.06835065]), array([2.24216227, 2.09710479])]
[array([0.93112614, 0.76068832]), array([1.97727979, 2.00118108])]
[array([0.93343355, 1.21840232]), array([1.69113418, 2.10148343])]
[array([0.65468454, 1.21563852]), array([2.0777007 , 1.93493437])]
[array([1.12117932, 0.99929409]), array([2.06457838, 2.06849632])]
[array([1.38777588, 1.0008914 ]), array([1.93048985, 2.02047269])]
[array([1.01993161, 1.02709989]), array([2.00398303, 2.0894369 ])]
[array([1.07734997, 1.13108666]), array([1.8285302 , 2.20426822])]
[array([0.73730781, 1.19700454]), array([2.34242768, 1.87738941])]
[array([0.93623307, 0.90999097]), array([2.04486664, 1.96434157])]
[array([0.91888943, 1.15891638]), array([1.8779909 , 2.15465844])]
[array([1.19598307, 0.84575188]), array([2.04495351, 2.17163982])]
[array([1.18937505, 1.08761405]), array([2.13758892, 1.84511849])]
[array([1.0329363 , 1.07166973]), array([1.98311587, 1.92803744])]
[array([1.03709324, 0.65945672]), array([1.89081081, 2.00973508])]
[array([1.08673129, 1.09558406]), array([2.04485624, 2.00693289])]
[array([1.03468633, 1.10090318]), array([2.02658483, 1.89917981])]
[array([0.97808783, 1.12981822]), array([1.80163298, 2.27655403])]
[array([0.83766086, 1.03165072]), array([2.12491001, 1.99920902])]
[array([1.07241857, 0.95415171]), array([2.02683286, 1.99590785])]
[array([0.52635082, 1.34788645]), array([2.01940613, 2.1382894 ])]
[array([0.88914555, 1.25650872]), array([2.02197152, 1.90801394])]
[array([1.2220379 , 0.82657678]), array([2.04228772, 1.88503024])]
[array([0.91144989, 0.90469011]), array([1.76109402, 2.26219286])]
[array([0.91178439, 1.06394946]), array([1.96671787, 1.90666275])]
[array([1.19841597, 0.97079335]), array([2.09023118, 2.09783802])]
[array([0.95382489, 1.09497329]), array([2.12109312, 1.74640991])]
[array([0.91509607, 0.88518847]), array([1.75669562, 2.33939869])]
[array([0.89624202, 1.00852165]), array([1.95228528, 2.1508613 ])]
[array([0.80198664, 1.15569117]), array([1.99000312, 2.13380851])]
[array([0.94519642, 1.11146312]), array([1.939032  , 2.09111697])]
[array([0.98043973, 1.0061151 ]), array([2.06326862, 1.87411696])]
```

```
[array([0.88359008, 0.97133768]), array([2.01407735, 1.96052391])]
[array([0.80662218, 1.18524509]), array([2.06433012, 2.02786743])]
[array([1.09179892, 1.10040275]), array([1.92206043, 2.15744785])]
[array([1.08621267, 0.70876204]), array([2.09707448, 1.91992236])]
[array([1.02783816, 1.13655578]), array([2.04664229, 1.97074779])]
[array([1.2168597 , 0.87940899]), array([1.83085801, 2.09985794])]
[array([0.81768489, 1.09043142]), array([2.13917817, 1.97122945])]
[array([0.96727134, 1.08150998]), array([2.04388369, 1.80561552])]
[array([1.04107215, 0.75498104]), array([1.80170031, 2.18048071])]
[array([0.83773481, 1.24351639]), array([1.95958932, 1.85957116])]
[array([0.99471653, 1.09877008]), array([2.00273053, 2.02224125])]
[array([1.00480178, 0.93719494]), array([1.87729271, 2.01327963])]
[array([0.97688047, 1.24141854]), array([1.89181528, 2.12724071])]
[array([1.09518935, 0.94820823]), array([2.02054174, 1.98822704])]
[array([0.69345593, 0.9383353 ]), array([2.00894008, 1.90233502])]
[array([1.16732116, 0.84910072]), array([2.03915319, 1.88568495])]
[array([0.98814649, 1.01577103]), array([2.07326289, 1.77370833])]
[array([0.94834767, 1.05855506]), array([2.03944708, 1.96461227])]
[array([1.02083314, 0.87102883]), array([2.20668645, 1.94243538])]
[array([0.99560413, 1.22051229]), array([1.90301718, 2.01683234])]
[array([1.05402853, 0.87218802]), array([1.8870957 , 1.90045551])]
[array([1.11059615, 0.75216445]), array([1.91259898, 1.92599931])]
[array([0.96922394, 1.04968873]), array([1.97059337, 1.97019341])]
[array([1.20578487, 0.74406278]), array([2.04086721, 1.98200078])]
[array([0.8381803, 0.78331  ]), array([2.29564214, 1.87614481])]
[array([0.99451224, 0.95653145]), array([2.03035393, 1.74496909])]
[array([1.15371399, 0.80932965]), array([2.00920047, 2.11132994])]
[array([0.99022561, 0.97838969]), array([2.04766283, 2.02273148])]
[array([1.13103245, 0.90147812]), array([2.11791168, 1.77349256])]
[array([0.89622563, 0.93380054]), array([2.08828548, 1.76894305])]
[array([1.04015909, 1.14587093]), array([2.09137075, 1.88276824])]
[array([0.61485077, 1.4053672 ]), array([2.08450327, 2.0320207 ])]
[array([0.95788838, 1.10641587]), array([2.00883901, 2.0681884 ])]
[array([1.09311754, 0.93554341]), array([2.05146156, 2.1121795 ])]
[array([1.0058557 , 1.21992646]), array([1.58025075, 2.09007811])]
[array([0.7863716 , 1.01655771]), array([2.09302648, 1.86646201])]
[array([0.80052404, 1.22684253]), array([1.99549724, 2.04560022])]
[array([0.83376376, 1.07891748]), array([2.07251111, 2.00163969])]
[array([0.88152574, 1.03048357]), array([1.85380017, 2.13225989])]
[array([0.83457479, 1.0717429 ]), array([2.03430186, 1.99407116])]
```

Out[45]:

```
(array([ 9., 18., 32., 24.,  4.,  2.,  0.,  0., 26., 85.]),
 array([0.34 , 0.406, 0.472, 0.538, 0.604, 0.67 , 0.736, 0.802, 0.86
8,
        0.934, 1.   ]),
 <a list of 10 Patch objects>)
```



0
10
20
30
40
50
60
70
80
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Problem 6

In [626]:

```
hw3 = pd.read_csv("hw3_dataset.txt", delim_whitespace=True, names = ['id', 'fund
ing', 'fv', 'shares', 'l_buyout'])
```

6 A

Trying out EDA

In [630]:

```
hw3.describe()
```

Out[630]:

	funding	fv	shares	l_buyout
count	482.000000	4.820000e+02	4.820000e+02	482.000000
mean	0.439834	2.651672e+07	2.227942e+06	0.093361
std	0.496883	2.632174e+07	1.413872e+06	0.291240
min	0.000000	1.200000e+06	3.000000e+05	0.000000
25%	0.000000	1.025000e+07	1.300000e+06	0.000000
50%	0.000000	1.950000e+07	2.000000e+06	0.000000
75%	1.000000	3.250000e+07	2.700000e+06	0.000000
max	1.000000	2.346000e+08	1.101862e+07	1.000000

The fv and shares features have really large values compared to l_buyout and funding columns.

In [627]:

```
hw3.head()
```

Out[627]:

	id	funding	fv	shares	l_buyout
0	1	0	1200000	3000000	0
1	2	0	1454000	1454000	1
2	3	0	1500000	300000	0
3	4	0	1530000	510000	0
4	5	0	2000000	800000	0

In [628]:

```
"""
we know that id is not giving any useful information
also, we'll check for missing values here
"""

hw3 = hw3.drop('id', axis=1)
nas = hw3.isnull().sum()
print(nas)
```

```
funding     0
fv          0
shares      0
l_buyout    0
dtype: int64
```

There are no missing values in this dataset

In [632]:

```
features = hw3.drop('funding', axis=1)
target = hw3['funding']
```

In [633]:

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=
0.33, random_state=42)
```

In [634]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 322 entries, 157 to 102
Data columns (total 3 columns):
fv           322 non-null int64
shares       322 non-null int64
l_buyout     322 non-null int64
dtypes: int64(3)
memory usage: 10.1 KB
```

In [635]:

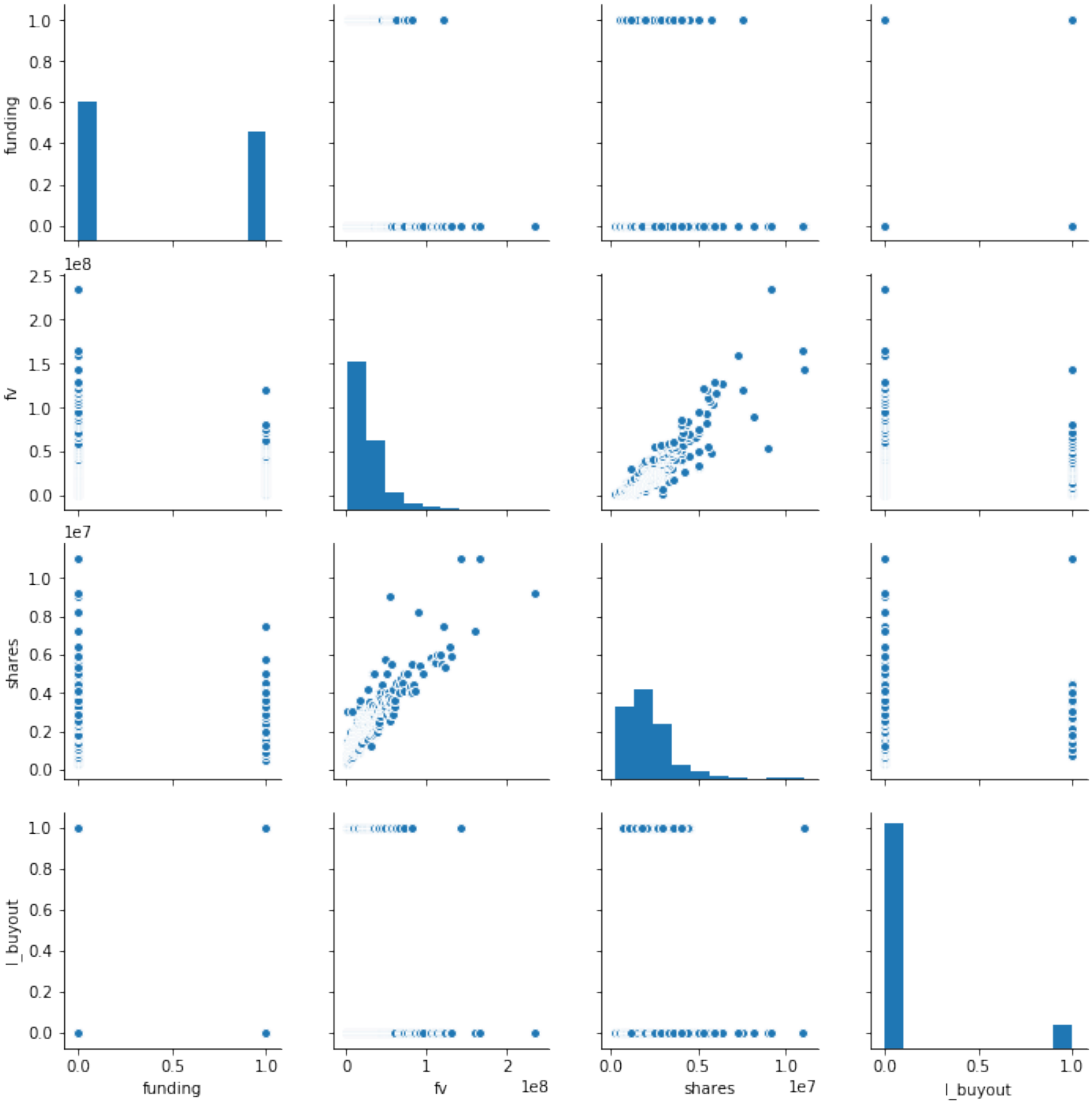
```
X_train.head()
```

Out[635]:

	fv	shares	l_buyout
157	13000000	1300000	0
449	63000000	4500000	1
118	10103125	1325000	0
114	9625000	1375000	0
439	55000000	2500000	0

In [653]:

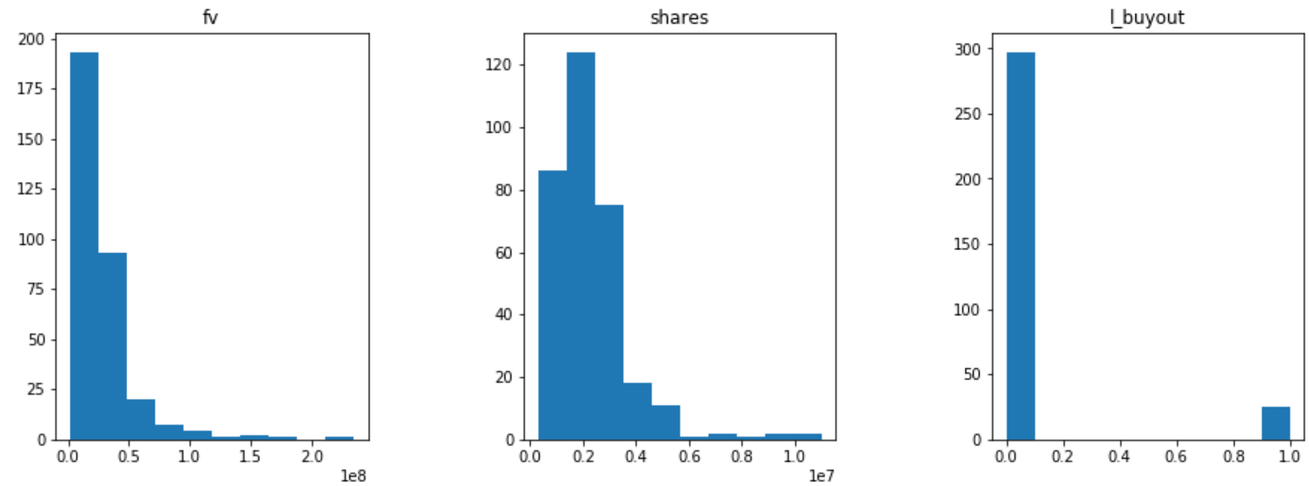
```
sns.pairplot(hw3, vars=hw3.columns)
plt.show()
```



funding
fv
shares
l_buyout
1e8
1e7

In [644]:

```
fig, axs = plt.subplots(1,3, figsize=(15, 5), facecolor='w', edgecolor='k')
fig.subplots_adjust(hspace = .5, wspace=.5)

for i,j in enumerate(X_train.columns):
    axs[i].hist(X_train[j])
    axs[i].set_title(j)
```

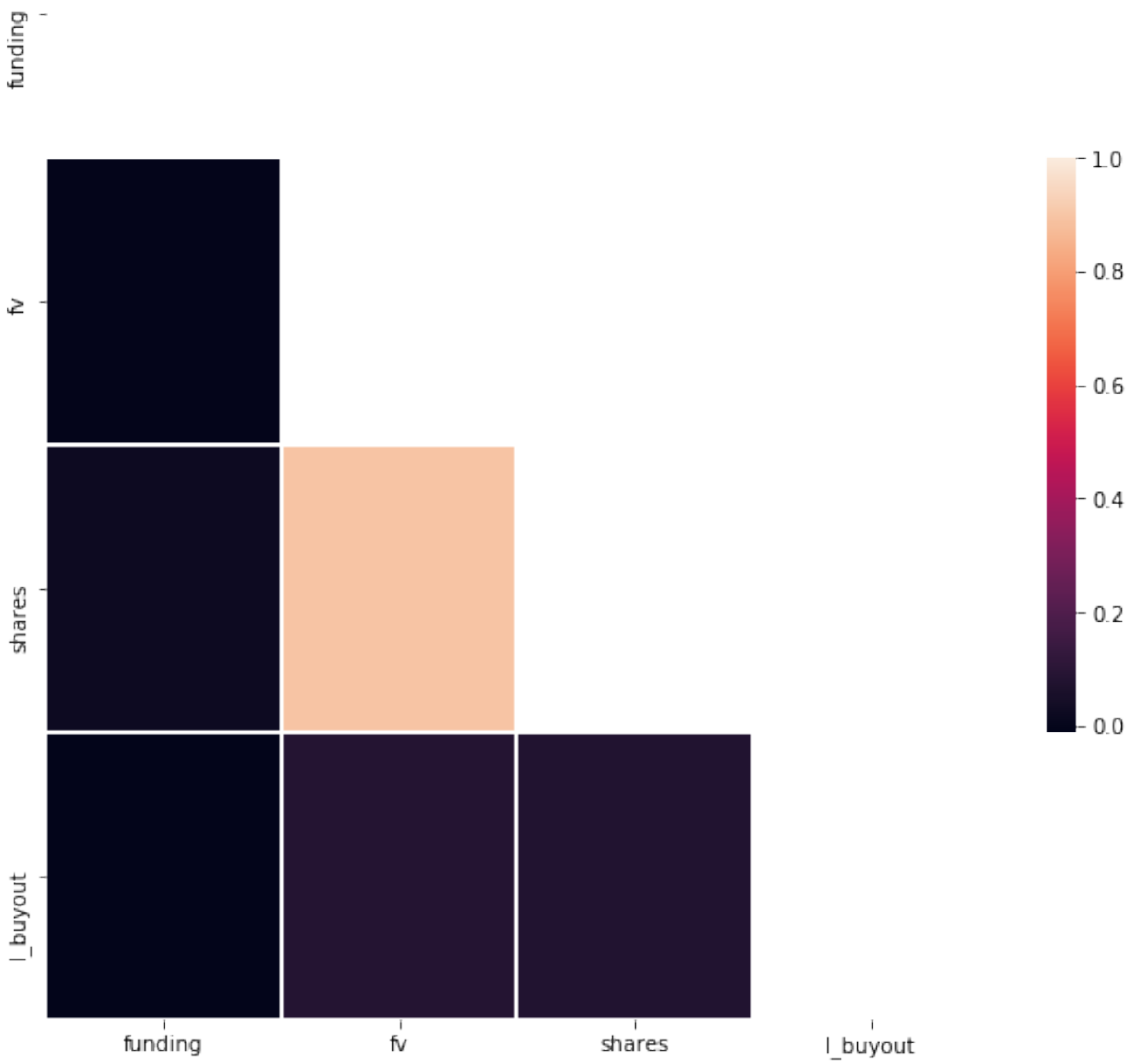


In [649]:

```
corr = hw3.corr()
mask = np.zeros_like(corr, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True
plt.subplots(figsize = (10, 10))
sns.heatmap(corr, mask = mask, xticklabels = 1,
            yticklabels = 1, linewidths = 1, cbar_kws = {"shrink": .5})
```

Out[649]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f01443329b0>


funding
fv
shares
l_buyout
funding
fv
shares
l_buyout
1.0
0.8
0.6
0.4
0.2
0.0

Inference

In [654]:

```
X_train.head()
```

Out[654]:

	fv	shares	l_buyout
157	13000000	1300000	0
449	63000000	4500000	1
118	10103125	1325000	0
114	9625000	1375000	0
439	55000000	2500000	0

6 B

In [ ]:

```
X_train['capital'] = np.log(X_train['face_value'] * X_train['n_shares'])
X_validate['capital'] = np.log(X_validate['face_value'] * X_validate['n_shares']
)
X_test['capital'] = np.log(X_test['face_value'] * X_test['n_shares'])

X_train['fvl'] = np.log(X_train.face_value)
X_train['nsl'] = np.log(X_train.n_shares)

X_validate['fvl'] = np.log(X_validate.face_value)
X_validate['nsl'] = np.log(X_validate.n_shares)

X_test['fvl'] = np.log(X_test.face_value)
X_test['nsl'] = np.log(X_test.n_shares)
```

The first thing to try is the log value of the two features which have a positively skewed distr.

In [660]:

```
X_train['fv_log'] = np.log(X_train['fv'])
X_test['fv_log'] = np.log(X_test['fv'])
X_train['shares_log'] = np.log(X_train['shares'])
X_test['shares_log'] = np.log(X_test['shares'])
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  """
```

In [710]:

```
# method to print basic metrics of a classifier
def print_metrics(y_test, y_hat):
    print("Accuracy of model is ", metrics.accuracy_score(y_test, y_hat))
    print("Precision of model is ", metrics.precision_score(y_test, y_hat))
    print("Recall of model is ", metrics.recall_score(y_test, y_hat))
    print("Confusion matrix\n", metrics.confusion_matrix(y_test, y_hat))

# method to plot AUC
def plot_roc_auc(model, X_test):
    y_hat_probabilities = model.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_hat_probabilities)
    auc = metrics.roc_auc_score(y_test, y_hat_probabilities)
    plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
    plt.legend(loc=4)
    plt.show()
```

In [711]:

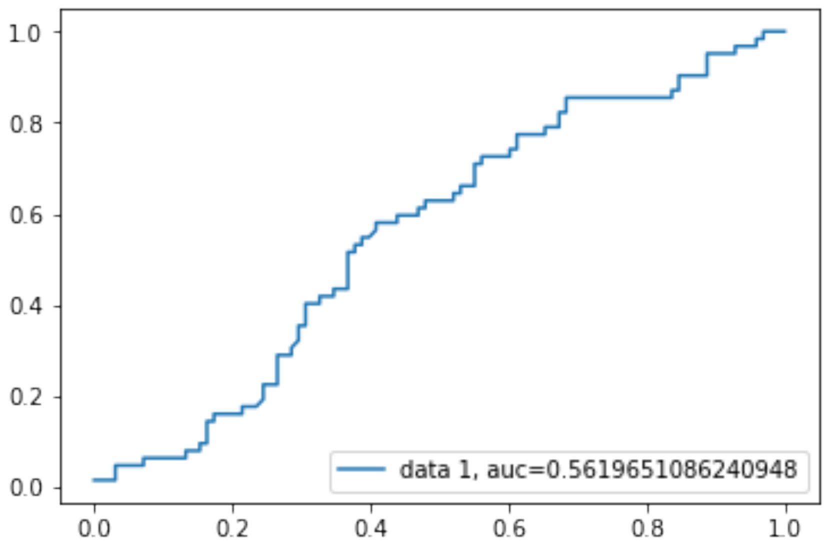
```
lr = LogisticRegression()
lr.fit(X_train[['fv_log', 'shares_log', 'l_buyout']], y_train)

y_hat = lr.predict(X_test[['fv_log', 'shares_log', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.5625
Precision of model is  0.43333333333333335
Recall of model is  0.41935483870967744
Confusion matrix
 [[64 34]
 [36 26]]
```

So, that's an okayish accuracy. We just tried all variables we thought would work. Let's try plotting the Area under curve.

In [712]:

```
plot_roc_auc(lr, X_test[['fv_log', 'shares_log', 'l_buyout']])
```



Let's try more feature transformation. Since face_value and number of shares are highly correlated in stock market, it's worth trying. Moreover, there's no point in using two highly correlated features in building a model. the correlation between them is greater than 0.8. I will multiply them and then take a log over the multiplied quantity.

In [713]:

```
# All the values are greater than 0 so no need to do log(p+1) as shown in class.

X_train['stock_value'] = np.log((X_train['fv'] * X_train['shares']))
X_test['stock_value'] = np.log((X_test['fv'] * X_test['shares']))
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
```

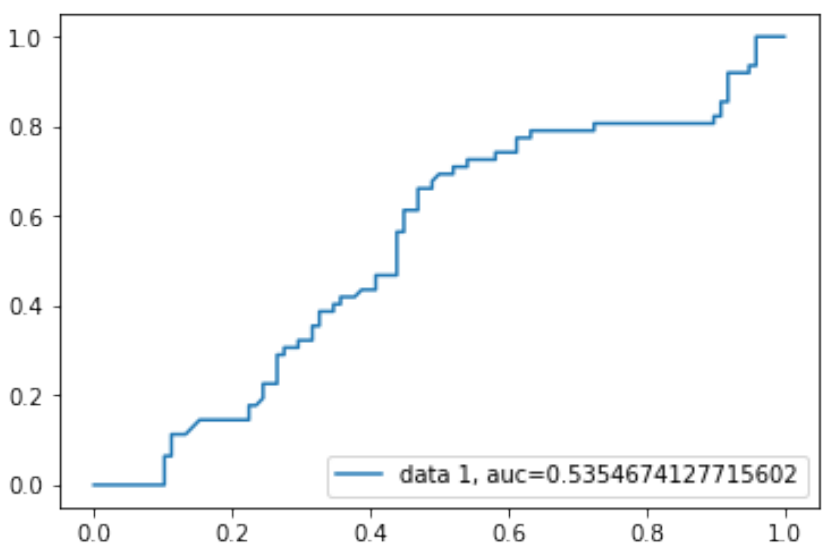
In [714]:

```
lr = LogisticRegression()
lr.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat = lr.predict(X_test[['stock_value', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.55625
Precision of model is  0.0
Recall of model is  0.0
Confusion matrix
 [[89  9]
 [62  0]]
```

In [715]:

```
plot_roc_auc(lr, X_test[['stock_value', 'l_buyout']])
```



Didn't give a significant improvement. Logistic regression has different solver so will try a different solver.

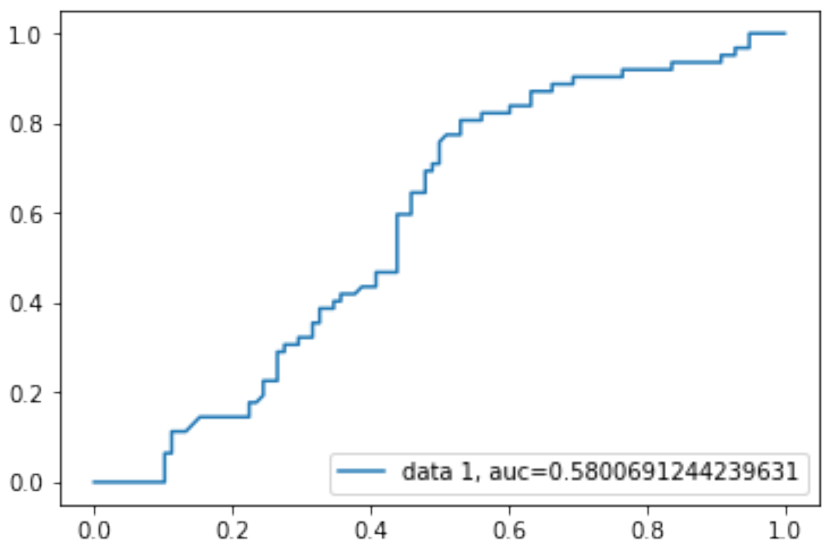
In [736]:

```
lr = LogisticRegression(solver='lbfgs')
lr.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat = lr.predict(X_test[['stock_value', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.55
Precision of model is  0.4074074074074074
Recall of model is  0.3548387096774194
Confusion matrix
 [[66 32]
 [40 22]]
```

In [717]:

```
plot_roc_auc(lr, X_test[['stock_value', 'l_buyout']])
```



Though it didn't improve significantly, it is definitely better than the previous model.

6 C

Now, we try another kind of classifier- LDA. Again following a similar approach, let's start with all features and slowly narrow down to the ones which we found useful from our logistic regression model.

In [718]:

```
lda = sklearn_lda()
lda.fit(X_train[['fv_log', 'shares_log', 'l_buyout']], y_train)

y_hat = lda.predict(X_test[['fv_log', 'shares_log', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.575
Precision of model is  0.45
Recall of model is  0.43548387096774194
Confusion matrix
 [[65 33]
 [35 27]]
```

Now, trying the enhanced version:

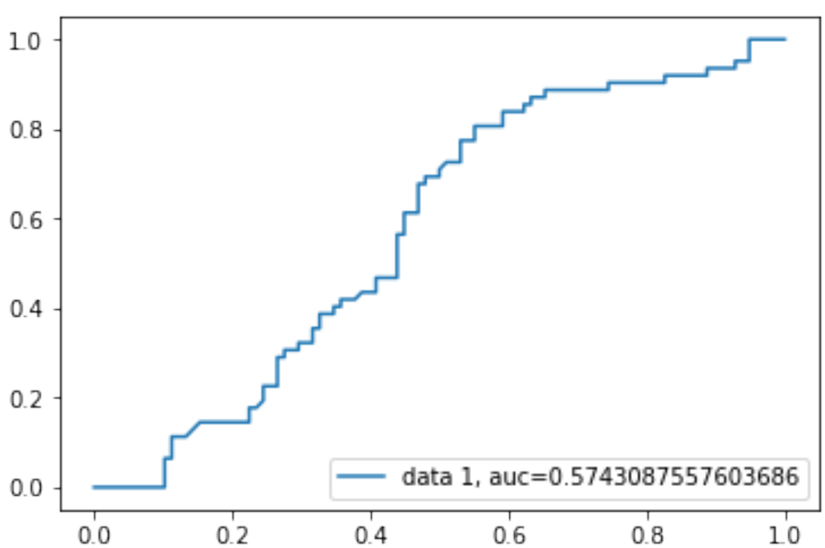
In [719]:

```
lda = sklearn_lda()
lda.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat = lda.predict(X_test[['stock_value', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.55
Precision of model is  0.4074074074074074
Recall of model is  0.3548387096774194
Confusion matrix
 [[66 32]
 [40 22]]
```

In [720]:

```
plot_roc_auc(lda, X_test[['stock_value', 'l_buyout']])
```



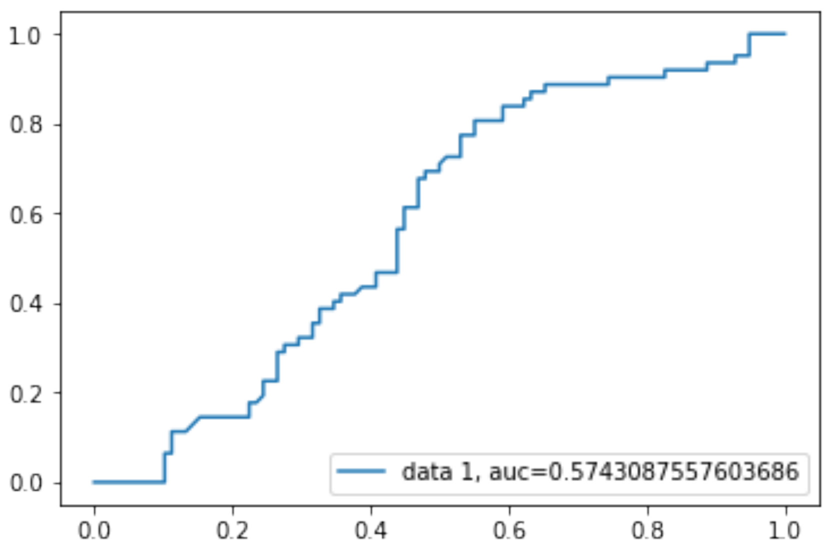
In [731]:

```
lda = sklearn_lda(solver='svd')
lda.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat = lda.predict(X_test[['stock_value', 'l_buyout']])
print_metrics(y_test, y_hat)
```

```
Accuracy of model is  0.55
Precision of model is  0.4074074074074074
Recall of model is  0.3548387096774194
Confusion matrix
 [[66 32]
 [40 22]]
```

In [732]:

```
plot_roc_auc(lda, X_test[['stock_value', 'l_buyout']])
```



Let's even test the training accuracy of our model

In [723]:

```
y_hat_train = lda.predict(X_train[['stock_value', 'l_buyout']])
print_metrics(y_train, y_hat_train)
```

```
Accuracy of model is  0.5807453416149069
Precision of model is  0.5619834710743802
Recall of model is  0.4533333333333333
Confusion matrix
 [[119  53]
 [ 82  68]]
```

Here, we observe again that feature transformation provides better AUC compared to fitting the model on the original features. However, changing solver here didn't make any difference.

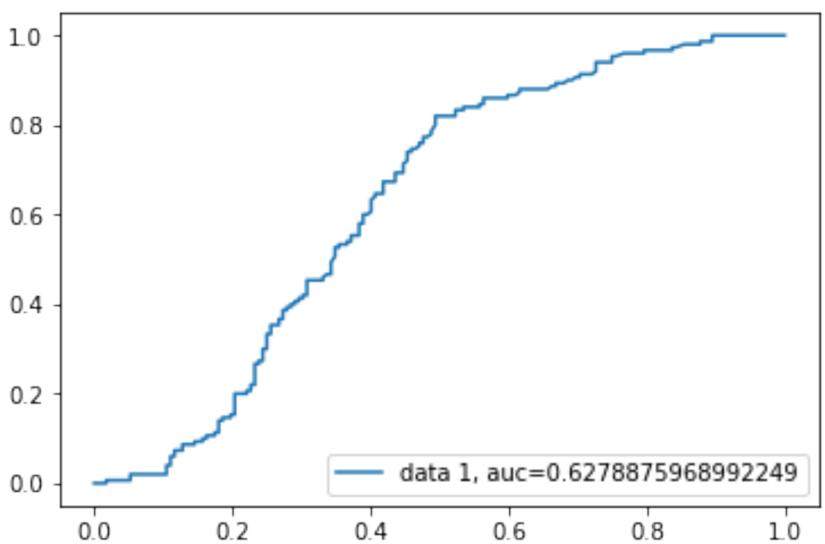
6 D

Let's try finding training accuracies:
a) Logistic regression

In [725]:

```
lr = LogisticRegression(solver='lbfgs')
lr.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat_train = lr.predict(X_train[['stock_value', 'l_buyout']])
print_metrics(y_train, y_hat_train)

y_hat_probabilities = lr.predict_proba(X_train[['stock_value', 'l_buyout']])[::,
1]
fpr, tpr, _ = metrics.roc_curve(y_train,  y_hat_probabilities)
auc = metrics.roc_auc_score(y_train, y_hat_probabilities)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

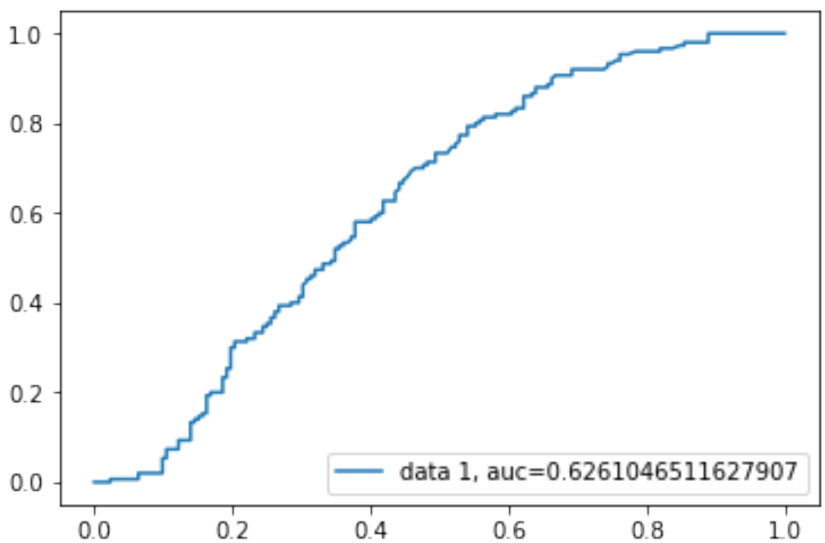
```
Accuracy of model is  0.5745341614906833
Precision of model is  0.5546218487394958
Recall of model is  0.44
Confusion matrix
 [[119  53]
 [ 84  66]]
```



In [727]:

```
lr = LogisticRegression(solver='lbfgs')
lr.fit(X_train[['fv_log', 'l_buyout', 'shares_log']], y_train)

y_hat_train = lr.predict(X_train[['fv_log', 'l_buyout', 'shares_log']])
print_metrics(y_train, y_hat_train)

y_hat_probabilities = lr.predict_proba(X_train[['fv_log', 'l_buyout', 'shares_lo
g']])[::,1]
fpr, tpr, _ = metrics.roc_curve(y_train,  y_hat_probabilities)
auc = metrics.roc_auc_score(y_train, y_hat_probabilities)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

```
Accuracy of model is  0.5838509316770186
Precision of model is  0.5634920634920635
Recall of model is  0.47333333333333333
Confusion matrix
 [[117  55]
 [ 79  71]]
```

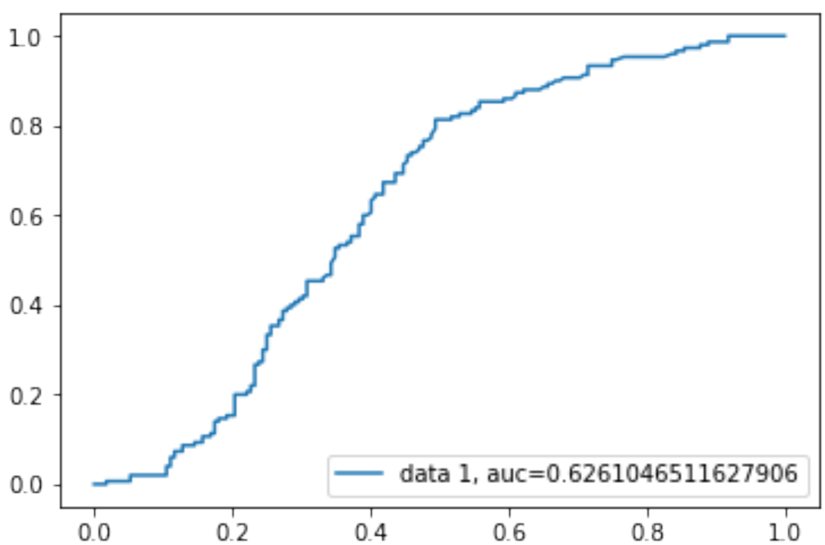


b) Linear Discriminant Analysis

In [737]:

```
lda = sklearn_lda(solver='svd')
lda.fit(X_train[['stock_value', 'l_buyout']], y_train)

y_hat_train = lda.predict(X_train[['stock_value', 'l_buyout']])
print_metrics(y_train, y_hat_train)

y_hat_probabilities = lda.predict_proba(X_train[['stock_value', 'l_buyout']])[::
,1]
fpr, tpr, _ = metrics.roc_curve(y_train,  y_hat_probabilities)
auc = metrics.roc_auc_score(y_train, y_hat_probabilities)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

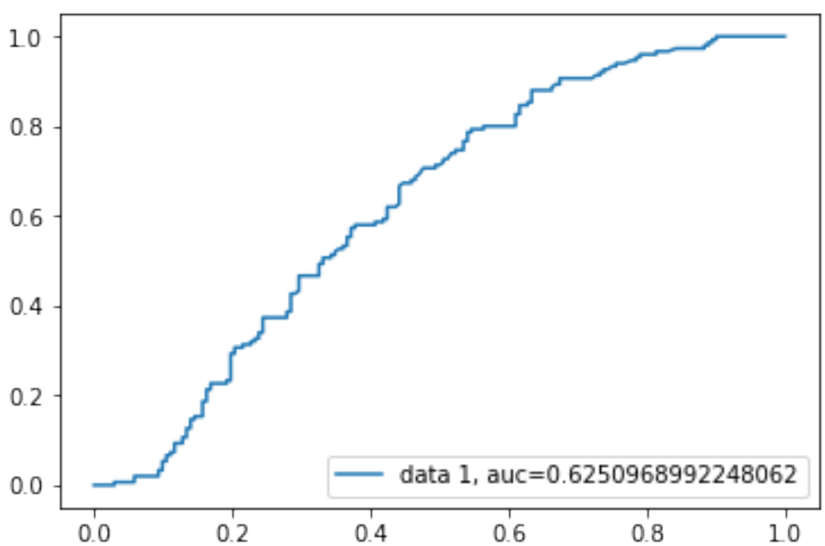
```
Accuracy of model is  0.5807453416149069
Precision of model is  0.5619834710743802
Recall of model is  0.4533333333333333
Confusion matrix
 [[119  53]
 [ 82  68]]
```



In [738]:

```
lda = sklearn_lda(solver='svd')
lda.fit(X_train[['fv_log', 'l_buyout', 'shares_log']], y_train)

y_hat_train = lda.predict(X_train[['fv_log', 'l_buyout', 'shares_log']])
print_metrics(y_train, y_hat_train)

y_hat_probabilities = lda.predict_proba(X_train[['fv_log', 'l_buyout', 'shares_l
og']])[::,1]
fpr, tpr, _ = metrics.roc_curve(y_train,  y_hat_probabilities)
auc = metrics.roc_auc_score(y_train, y_hat_probabilities)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

```
Accuracy of model is  0.5900621118012422
Precision of model is  0.5671641791044776
Recall of model is  0.5066666666666667
Confusion matrix
 [[114  58]
 [ 74  76]]
```



Here I tested out traning accuracies in my model.