In [8]:

```
#1a
import numpy as np
csv = 'https://www.dropbox.com/s/oqoyy9p849ewzt2/linear.csv?dl=1'
data = np.genfromtxt(csv,delimiter=',')
X = data[:,1:]
Y = data[:,0]
vX = X[0:10,:]
vY = Y[0:10]
tX = X[10:,:]
tY = Y[10:]

print('vX')
print(vX.shape)
print('tX')
print(tX.shape)
print('vY')
print(vY.shape)
print('tY')
print(tY.shape)
```

```
vX
(10L, 4L)
tX
(40L, 4L)
vY
(10L,)
tY
(40L,)
```

In [9]:

```
#1b
import theano
import theano.tensor as T
n = X.shape[0]
d = X.shape[1]
learn_rate = 0.5



x = T.matrix(name='x')  # feature matrix
y = T.vector(name='y')  # response vector
w = theano.shared(np.zeros((d,1)),name='w')




reg_penalty = 0.15
reg_loss = T.sum((T.dot(x, w).T - y)**2)/2/n  +reg_penalty*(w[0,0]**2+w[1,0]**2+w[2,0]**2)/
grad_reg_loss = T.grad(reg_loss, wrt=w)

train_model = theano.function(inputs=[],
                       outputs=reg_loss,
                       updates=[(w, w - learn_rate * grad_reg_loss)],
                       givens={x:X, y:Y})

# Execute the gradient descent algorithm.
n_steps = 100
for i in range(n_steps):
    train_model()

print(w.get_value())
```

```
[[-0.51575135]
 [ 1.18644932]
 [ 0.03302971]
 [-1.86038231]]
```

In [10]:

```
#1c
from scipy.optimize import fmin_l_bfgs_b as minimize
csv = 'https://www.dropbox.com/s/oqoyy9p849ewzt2/linear.csv?dl=1'
data = np.genfromtxt(csv,delimiter=',')

def costgrad(w,*args):

    reg_penalty = 0.15

    x = args[0]
    y = args[1]
    n = x.shape[0]
    cost = np.sum((np.dot(x, w).T - y)**2)/2/n  + reg_penalty*(w[0]**2+w[1]**2+w[2]**2)/2

    a = np.asarray([w[0], w[1], w[2],0])
    grad = reg_penalty*(a) + np.dot(np.dot(x.T,x),w)/n - np.dot(x.T,y)/n

    return cost, grad

x = data[:,1:]
d = x.shape[1]
w = np.zeros((d,1))
y = data[:,0]
optw,cost,messages = minimize(costgrad,w,args=(x,y),factr=10,pgtol=1e-10)

print(optw)
print(cost)
```

```
[-0.51575135  1.18644932  0.03302971 -1.86038231]
0.14782238561
```

In [11]:

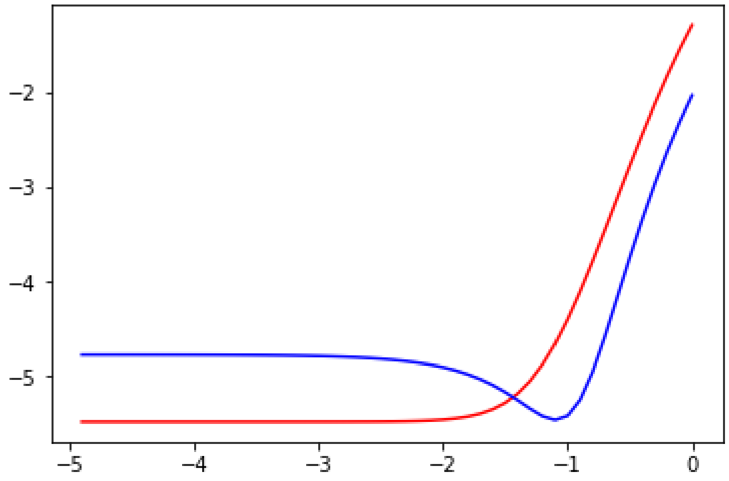
```
#1d
def ridge_regression(tX,tY,l):
    n = tX.shape[0]
    A=np.eye(4)
    A[3,3]=0
    b=np.dot(np.dot(np.linalg.inv(n*l*A+np.dot(tX.T,tX)),tX.T),tY)
    return b
w = ridge_regression(X,Y,0.15)
print(w)
```

```
[-0.51575135  1.18644932  0.03302971 -1.86038231]
```

In [12]:

```
#1e
import matplotlib.pyplot as plt
tn = tX.shape[0]
vn = vX.shape[0]
tloss = []
vloss = []
index = -np.arange(0,5,0.1)

for i in index:
    w = ridge_regression(tX,tY,10**i)
    tloss = tloss+[np.sum((np.dot(tX,w)-tY)**2)/tn/2]
    vloss = vloss+[np.sum((np.dot(vX,w)-vY)**2)/vn/2]
print(w)
plt.plot(index,np.log(tloss),'r')
plt.plot(index,np.log(vloss),'b')
plt.show()

#lambda that minimizes validation loss is 0.1
```

```
[-0.57006903  1.3753366   0.02830073 -1.88456156]
```



In [ ]:

```
```

In [2]:

```
#2a
import numpy as np
import numpy.random as rng
from sklearn.cluster import KMeans
import matplotlib.image as mpimg

n_colors = 32
img = mpimg.imread('https://www.dropbox.com/s/bmwwfct2qxjfje4/sutd.png?dl=1')
img = img[:,:,:3]
img.shape
w, h, d = tuple(img.shape)
image_array = np.reshape(img, (w * h, d))
print(image_array.shape)

print("Fitting model on a small sub-sample of the data")
image_array_sample = image_array[rng.randint(w * h, size=1000)]
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array_sample)
kmeans_palette = kmeans.cluster_centers_
kmeans_labels = kmeans.predict(image_array)
print(kmeans_labels)
```

```
(706580L, 3L)
Fitting model on a small sub-sample of the data
[ 7  6  6 ..., 18 18 18]
```

In [3]:

```
#2b
from sklearn.metrics import pairwise_distances_argmin
random_palette = image_array[rng.randint(w*h,size=n_colors)]
print("Predicting color indices on the full image (random)")
random_labels = pairwise_distances_argmin(random_palette,image_array,axis=0)
print(random_labels)
```

```
Predicting color indices on the full image (random)
[25 23 23 ..., 27 27 27]
```

#2c) To find z that minimizes centroid:

$$\frac{\partial}{\partial z}\sum_{i=1}^{m} ||x^{(i)} - z||^2 = 0$$

$$-2\sum_{i=1}^{m} ||x^{(i)} - z|| = 0$$

$$\sum_{i=1}^{m} x^{(i)} - \sum_{i=1}^{m} z = 0$$

z is not dependent on index i:

$$\sum_{i=1}^{m} x^{(i)} - mz = 0$$

$$z = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

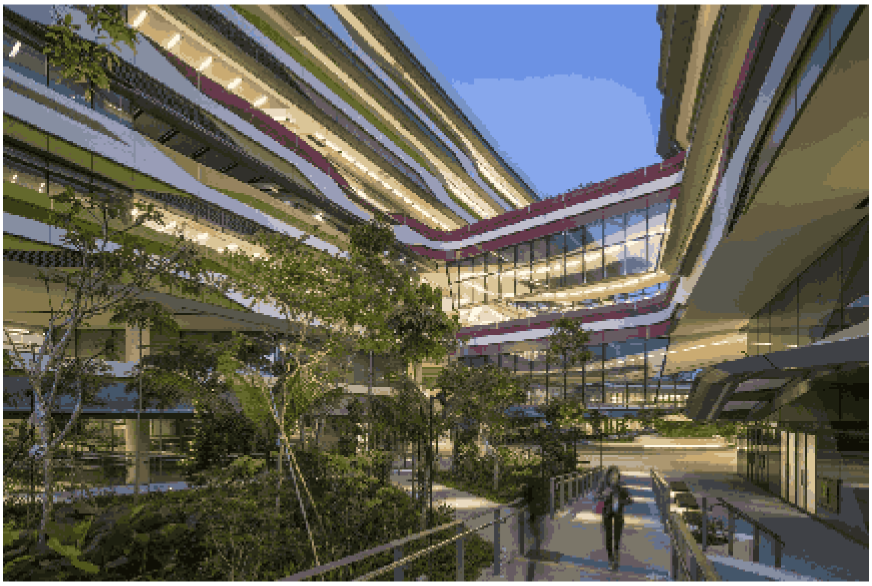
Proven

In [7]:

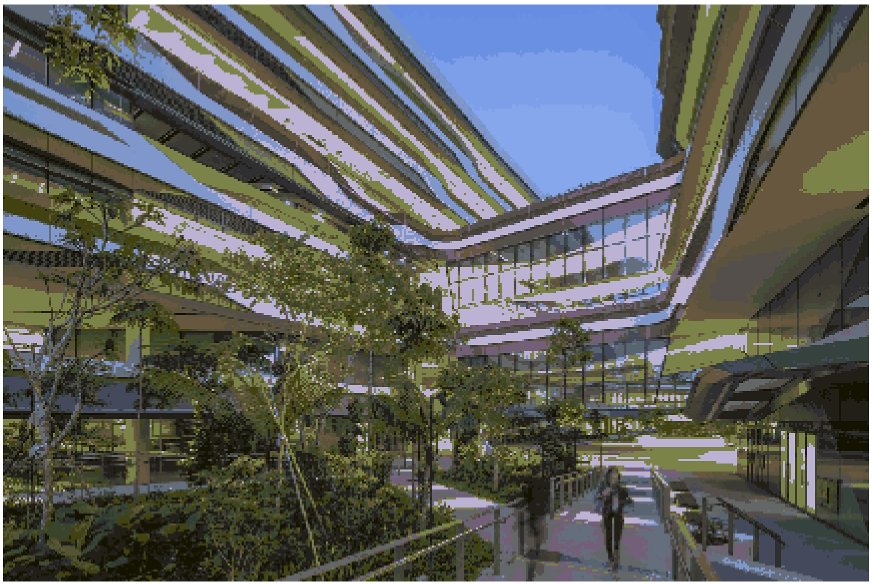
```
# Display all results, alongside original image
import matplotlib.pyplot as plt

def recreate_image(palette, labels, w, h):
    """Recreate the (compressed) image from the palette & labels"""
    d = palette.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = palette[labels[label_idx]]
            label_idx += 1
    return image

plt.figure(1)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Original image (16.8 million colors)')
plt.imshow(img)

plt.figure(2)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Compressed image (K-Means)')
plt.imshow(recreate_image(kmeans_palette, kmeans_labels, w, h))

plt.figure(3)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Compressed image (Random)')
plt.imshow(recreate_image(random_palette, random_labels, w, h))
plt.show()
```

Original image (16.8 million colors)

Compressed image (K-Means)



Compressed image (Random)



In [ ]:

In [1]:

```
#3a
import numpy as np
import pandas as pd
from IPython.display import display

X_data  = pd.read_csv("https://www.dropbox.com/s/klcmymaki2bxuey/train.csv?dl=1")
X_test  = pd.read_csv("https://www.dropbox.com/s/srbr7zmdbxzh9b4/test.csv?dl=1")
X_valid = X_data.sample(frac=0.2,random_state=200)
X_train = X_data.drop(X_valid.index)
Y_data  = X_data["Survived"]
Y_valid = X_valid["Survived"]
Y_train = X_train["Survived"]
ID_test = X_test["PassengerId"]

display(X_train.head())
display(X_train.describe())
display(X_test.head())
display(X_test.describe())

def preprocess(df):
    df = df.copy()
    df.drop(["PassengerId","Survived"],axis=1,inplace=True,errors="ignore")
    df.drop(["Name","Ticket","Cabin"],axis=1,inplace=True)
    df["Embarked"].fillna(df["Embarked"].mode()[0],inplace=True)
    df["Fare"].fillna(df["Fare"].median(),inplace=True)
    df["Age"].fillna(df["Age"].mean(),inplace=True)
    df = df.join(pd.get_dummies(df["Embarked"]))
    df.drop(["Embarked"],axis=1,inplace=True)
    df = df.join(pd.get_dummies(df["Sex"]))
    df.drop(["Sex"],axis=1,inplace=True)
    df = df.join(pd.get_dummies(df["Pclass"]))
    df.drop(["Pclass"],axis=1,inplace=True)
    df.loc[:,"Family"] = (df["Parch"]+df["SibSp"]>0)*1
    df.loc[:,"Child"] = (df["Age"]<16)*1
    return df


X_train = preprocess(X_train)
X_valid = preprocess(X_valid)
X_data = preprocess(X_data)
X_test = preprocess(X_test)
display(X_train.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	(
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William	male	35.0	0	0	373450	8.0500	

Henry

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	713.000000	713.000000	713.000000	568.000000	713.000000	713.000000	713.000000
mean	444.734923	0.374474	2.333801	29.449243	0.496494	0.381487	31.154686
std	259.331603	0.484327	0.830050	14.866483	1.029236	0.809213	45.471961
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	215.000000	0.000000	2.000000	20.000000	0.000000	0.000000	7.895800
50%	440.000000	0.000000	3.000000	28.000000	0.000000	0.000000	13.791700
75%	669.000000	1.000000	3.000000	39.000000	1.000000	0.000000	31.000000
max	889.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

	Age	SibSp	Parch	Fare	C	Q	S	female	male	1	2	3	Family	Child

	Age	SibSp	Parch	Fare	C	Q	S	female	male	1	2	3	Family	Child
0	22.000000	1	0	7.2500	0	0	1	0	1	0	0	1	1	0
2	26.000000	0	0	7.9250	0	0	1	1	0	0	0	1	0	0
3	35.000000	1	0	53.1000	0	0	1	1	0	1	0	0	1	0
4	35.000000	0	0	8.0500	0	0	1	0	1	0	0	1	0	0
5	29.449243	0	0	8.4583	0	1	0	0	1	0	0	1	0	0

In [2]:

```
#3b
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
print(logreg.score(X_valid, Y_valid))
```

```
0.792134831461
```

In [4]:

```
#3c
logreg = LogisticRegression()
logreg.fit(X_data, Y_data)
Y_test = logreg.predict(X_test)

coeff_df = pd.DataFrame(X_data.columns.delete(0))
coeff_df.columns = ['Features']
coeff_df["Coefficient Estimate"] = pd.Series(logreg.coef_[0])
display(coeff_df)
```

	Features	Coefficient Estimate
0	SibSp	-0.022540
1	Parch	-0.599687
2	Fare	-0.318363
3	C	0.003106
4	Q	0.180251
5	S	0.298824
6	female	-0.112214
7	male	1.483798
8	1	-1.116937
9	2	1.008920
10	3	0.216046
11	Family	-0.858106
12	Child	0.692171

In [6]:

```
#3d
ans = pd.DataFrame({"PassengerId":ID_test,"Survived":Y_test})
ans = pd.DataFrame({"PassengerId":ID_test,"Survived":Y_test})
ans.to_csv("submit.csv", index=False)
print(ans)
#score 0.77990
```

```
     PassengerId  Survived
0            892         0
1            893         0
2            894         0
3            895         0
4            896         1
5            897         0
6            898         1
7            899         0
8            900         1
9            901         0
10           902         0
11           903         0
12           904         1
13           905         0
14           906         1
15           907         1
16           908         0
17           909         0
18           910         1
19           911         1
20           912         0
21           913         0
22           914         1
23           915         1
24           916         1
25           917         0
26           918         1
27           919         0
28           920         0
29           921         0
..           ...       ...
388         1280         0
389         1281         0
390         1282         0
391         1283         1
392         1284         0
393         1285         0
394         1286         0
395         1287         1
396         1288         0
397         1289         1
398         1290         0
399         1291         0
400         1292         1
401         1293         0
402         1294         1
403         1295         0
404         1296         0
405         1297         0
406         1298         0
407         1299         0
```

```
408         1300         1
409         1301         1
410         1302         1
411         1303         1
412         1304         1
413         1305         0
414         1306         1
415         1307         0
416         1308         0
417         1309         0

[418 rows x 2 columns]
```