Miniproject 2

In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

In [27]:
```
def clean_data(movies_file, data_file):
    unique_title_id_map = {}  # to keep track of titles that already have an id
    needed_updates = {}  # this array will map ids that need to be changed to the id
    # they should be changed to
    with open(movies_file, 'r') as f:
        for line in f:
            line_data = line.strip('\n').split('\t')
            movie_id, title = line_data[0], line_data[1]
            if str(title) in unique_title_id_map:
                needed_updates[movie_id] = unique_title_id_map[str(title)]
            else:
                unique_title_id_map[str(title)] = str(movie_id)
    # print(needed_updates)

    data_arr = np.loadtxt(data_file, dtype=np.int)
    for i, row in enumerate(data_arr):
        if str(row[1]) in needed_updates:
            data_arr[i, 1] = needed_updates[str(row[1])]
    return (data_arr)
```

In [28]:
```
Y_train = np.loadtxt('data/train.txt').astype(int)
Y_test = np.loadtxt('data/test.txt').astype(int)

#movie_cols = ['Movie ID','Movie Title', 'Unknown', 'Action', 'Adventure', 'Animatio
#'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi

data_arr = clean_data('data/movies.txt','data/data.txt')

#rat_cols = ['User ID', 'Movie ID', 'Rating']
#ratings = np.loadtxt('data/data.txt', names=rat_cols)
```

In [29]:
```
print(data_arr)
```

```
[[ 196  242    3]
 [ 186  302    3]
 [  22  377    1]
 ...
 [ 276 1090    1]
 [  13  225    2]
 [  12  203    3]]
```

HW 5 Code:

In [30]:
```
def grad_U(Ui, Yij, Vj, reg, eta):
    """
    Takes as input Ui (the ith row of U), a training point Yij, the column
    vector Vj (jth column of V^T), reg (the regularization parameter lambda),
    and eta (the learning rate).

    Returns the gradient of the regularized loss function with
    respect to Ui multiplied by eta.
    """
    return eta * np.subtract(reg * Ui, (Yij - np.dot(Ui, Vj))* Vj)

def grad_V(Vj, Yij, Ui, reg, eta):
    """
    Takes as input the column vector Vj (jth column of V^T), a training point Yij,
    Ui (the ith row of U), reg (the regularization parameter lambda),
    and eta (the learning rate).

    Returns the gradient of the regularized loss function with
    respect to Vj multiplied by eta.
    """
    return eta * np.subtract(reg * Vj, (Yij - np.dot(Ui, Vj))* Ui)

def get_err(U, V, Y, reg=0.0):
    """
    Takes as input a matrix Y of triples (i, j, Y_ij) where i is the index of a user
    j is the index of a movie, and Y_ij is user i's rating of movie j and
    user/movie matrices U and V.

    Returns the mean regularized squared-error of predictions made by
    estimating Y_{ij} as the dot product of the ith row of U and the jth column of V

    sum = 0.0
    for x in range(len(Y)):
        i = Y[x, 0] - 1
        j = Y[x, 1] - 1
        Yij = Y[x, 2]
        sum += (Yij - np.dot(U[i], V[j]))**2
    return reg / 2 * (np.linalg.norm(U)**2 + np.linalg.norm(V)**2) + 0.5 * sum"""
    N,D = Y.shape
    err = 0

    for n in range(N):
        i = Y[n,0] - 1
        j = Y[n,1] - 1
        yij = Y[n,2]
        err += (yij - np.dot(U[i], V[j]))**2

    U_norm = np.linalg.norm(U)
    V_norm = np.linalg.norm(V)

    return (reg/2 *(U_norm**2 + V_norm**2) + err/2) / N


def train_model(M, N, K, eta, reg, Y, eps=0.0001, max_epochs=300):
    """
    Given a training data matrix Y containing rows (i, j, Y_ij)
    where Y_ij is user i's rating on movie j, learns an
    M x K matrix U and N x K matrix V such that rating Y_ij is approximated
    by (UV^T)_ij.

    Uses a learning rate of <eta> and regularization of <reg>. Stops after
    <max_epochs> epochs, or once the magnitude of the decrease in regularized
    MSE between epochs is smaller than a fraction <eps> of the decrease in
    MSE after the first epoch.
```

In [31]:
```
M = max(max(Y_train[:,0]), max(Y_test[:,0])).astype(int) # users
N = max(max(Y_train[:,1]), max(Y_test[:,1])).astype(int) # movies

K = 20

reg = 0 #10**-3
eta = 0.03 # learning rate
E_in = 0
E_out = 0

# Use to compute Ein and Eout
U,V, err = train_model(M, N, K, eta, reg, Y_train)
E_in = err
E_out = get_err(U, V, Y_test)
```

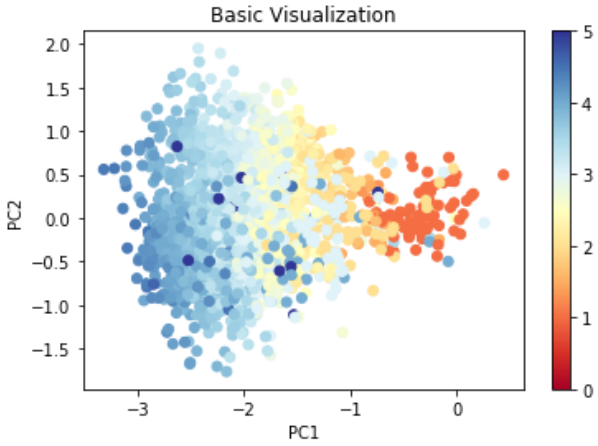
In [32]:
```
print(E_in)
print(E_out)
```

```
0.259049940313177
0.6425242815339629
```

In [8]:
```
movie_rating = np.zeros((1682, 1))
movie_num_user_rating = np.zeros((1682, 1))
for row in Y_train:
    # 0 is user id, 1 is movie id, 2 is rating
    movie_rating[row[1]-1] += row[2]
    movie_num_user_rating[row[1]-1] += 1
for row in Y_test:
    # 0 is user id, 1 is movie id, 2 is rating
    movie_rating[row[1]-1] += row[2]
    movie_num_user_rating[row[1]-1] += 1
movie_avg_rating = np.divide(np.array(movie_rating), np.array(movie_num_user_rating)
```

In [9]:
```
def visualize_2d(M, title):
    """Project a matrix into 2 dimensions and visualize it. """
    A, sigma, B = np.linalg.svd(M)
    M_proj = np.matmul(A[:,:2].transpose(), M)

    cm = plt.cm.get_cmap('RdYlBu')
    #plt.close('all')
    #ax = plt.figure().gca()
    index = range(M.shape[1])
    sc = plt.scatter(M_proj[0,index], M_proj[1,index], vmin=0,vmax=5, c=movie_avg_ra
    plt.colorbar(sc)
    plt.title(title)
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    #plt.savefig('basic.png')
    plt.show()

    return M_proj
```

In [10]:
```
title = 'Basic Visualization'
visualize_2d(V.T, title)
```



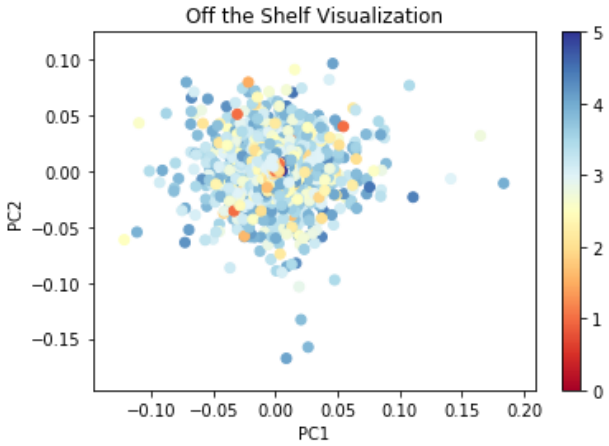
Out[10]:
```
array([[-2.771799  , -2.06359735, -2.19230287, ..., -0.68910227,
        -1.10195534, -1.41527178],
       [ 0.98868754,  0.79238156, -0.08720798, ...,  0.71179272,
         0.20558979, -0.55082704]])
```

Off the Shelf Code:

In [37]:
```
import numpy as np
from scipy.sparse.linalg import svds

def off_train(M, N, Y):
    train_m = np.zeros((M,N))
    arr = np.arange(len(Y))

    for index in arr:
        i = Y[index, 0] - 1
        j = Y[index, 1] - 1
        Yij = Y[index,2]
        train_m[i][j] = Yij

    #U, s, V = svds(train_m, k = 20)
    U, s, V = np.linalg.svd(train_m)

    return U, s, V
```

In [38]:
```
U_off, Sigma, V_off = off_train(M, N, Y_train)
#U = np.matmul(U, np.diag(np.sqrt(Sigma)))
#V = np.matmul(np.diag(np.sqrt(Sigma)), V)
```

In [39]:
```
title = 'Off the Shelf Visualization'
visualize_2d(V_off.T, title)
```



Out[39]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

In [35]:
```
#print(get_err(U, V, Y_train))
print(get_err(U, V, Y_test))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: