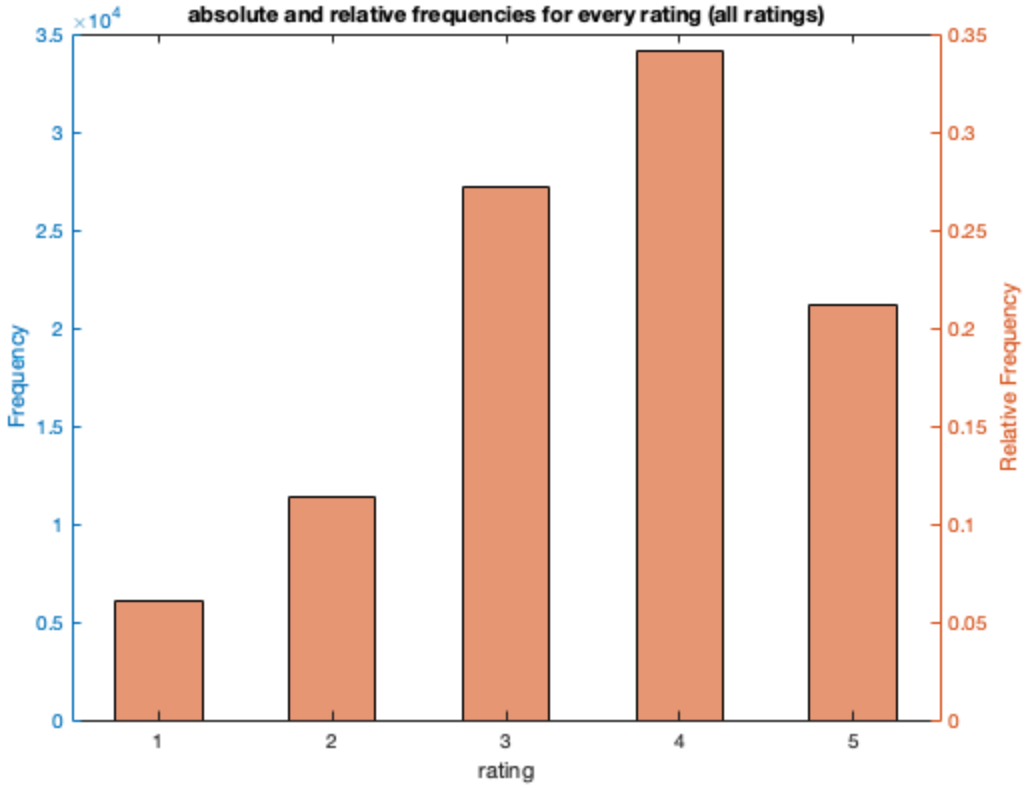
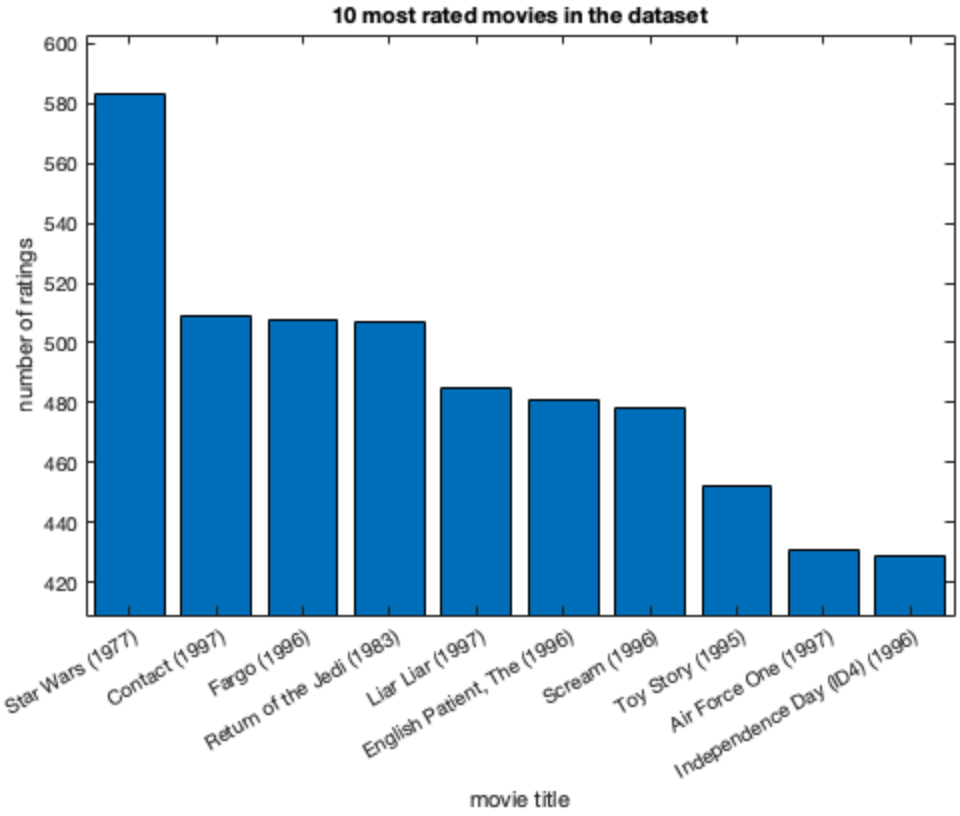
```
% CODE USED FOR 4.1

load('data.txt')
ratings = data(:,3); %column corresponding to tratings

%%histogram for all ratings
yyaxis left %want frequency on left y-axis
C = categorical(ratings,[1 2 3 4 5],{'1','2','3','4','5'}); %ratings
 are discrete so plot as categorical data
histogram(C,'BarWidth',0.5)
xlabel('rating')
ylabel('frequency')
yyaxis right %want relative frequency on right y-axis
histogram(C,'BarWidth',0.5,'Normalization','probability')
yyaxis left
title('absolute and relative frequencies for every rating (all
 ratings)')
ylabel('Frequency')
yyaxis right
ylabel('Relative Frequency')
```



Published with MATLAB® R2018b

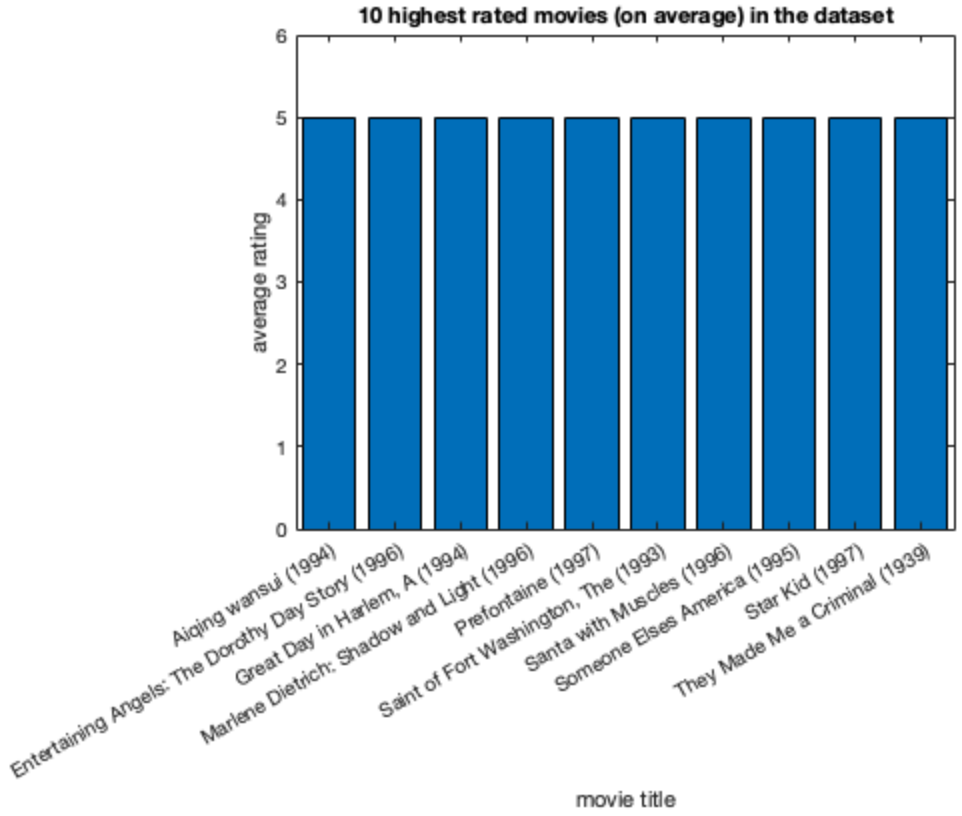
```
%CODE USED FOR 4.2

load('clean_ratings.txt');
movie_ids = clean_ratings(:,2);
unique_ids = unique(movie_ids);

freq_counts = [unique_ids,histc(movie_ids(:),unique_ids)]; %left
 column movie ids, right column number of ratings
disp(size(freq_counts))
[~,idx] = sort(freq_counts(:,2),'descend'); % sort by the 2nd column
sortedmat = freq_counts(idx,:);
top10 = sortedmat(1:10,:); %has top 10 in format (id,#ratings)
disp(top10)
X = categorical({'Star Wars (1977)','Contact (1997)','Fargo
 (1996)','Return of the Jedi (1983)','Liar Liar (1997)','English
 Patient, The (1996)','Scream (1996)','Toy Story (1995)','Air Force
 One (1997)','Independence Day (ID4) (1996)'});
X = reordercats(X,{'Star Wars (1977)','Contact (1997)','Fargo
 (1996)','Return of the Jedi (1983)','Liar Liar (1997)','English
 Patient, The (1996)','Scream (1996)','Toy Story (1995)','Air Force
 One (1997)','Independence Day (ID4) (1996)'});
%second one is to preserve order

bar(X,top10(:,2))
ylim([min(top10(:,2))-20,max(top10(:,2))+20])
title('10 most rated movies in the dataset')
ylabel('number of ratings')
xlabel('movie title')
```

```
        1664           2

    50   583
   258   509
   100   508
   181   507
   294   485
   286   481
   288   478
     1   452
   300   431
   121   429
```



Published with MATLAB® R2018b

```
%CODE USED FOR 4.3

load('clean_ratings.txt');
movie_ids = clean_ratings(:,2);
ratings = clean_ratings(:,3);
unique_ids = unique(movie_ids);
avg_ratings = zeros(length(unique_ids),1);
for i=1:length(unique_ids) %for each movie_id in the dataset,
 calculate the average rating
    movie_ratings =
 clean_ratings(clean_ratings(:,2)==unique_ids(i),3);
    number_ratings = length(movie_ratings);
    sum_ratings = sum(movie_ratings);
    avg_rating = sum_ratings/number_ratings;
    avg_ratings(i) = avg_rating;
end

[~,idx] = sort(avg_ratings,'descend');
top10_ratings = avg_ratings(idx);
top10_ratings = top10_ratings(1:10);
top10_ids = unique_ids(idx);
top10_ids = top10_ids(1:10);

X = categorical({'Great Day in Harlem, A (1994)','They Made Me a
 Criminal (1939)','Prefontaine (1997)','Marlene Dietrich: Shadow
 and Light (1996)','Star Kid (1997)','Saint of Fort Washington, The
 (1993)','Someone Elses America (1995)','Entertaining Angels: The
 Dorothy Day Story (1996)','Santa with Muscles (1996)','Aiqing wansui
 (1994)'});

bar(X,top10_ratings)
title('10 highest rated movies (on average) in the dataset')
ylabel('average rating')
xlabel('movie title')
ylim([0,6])
```



Published with MATLAB® R2018b

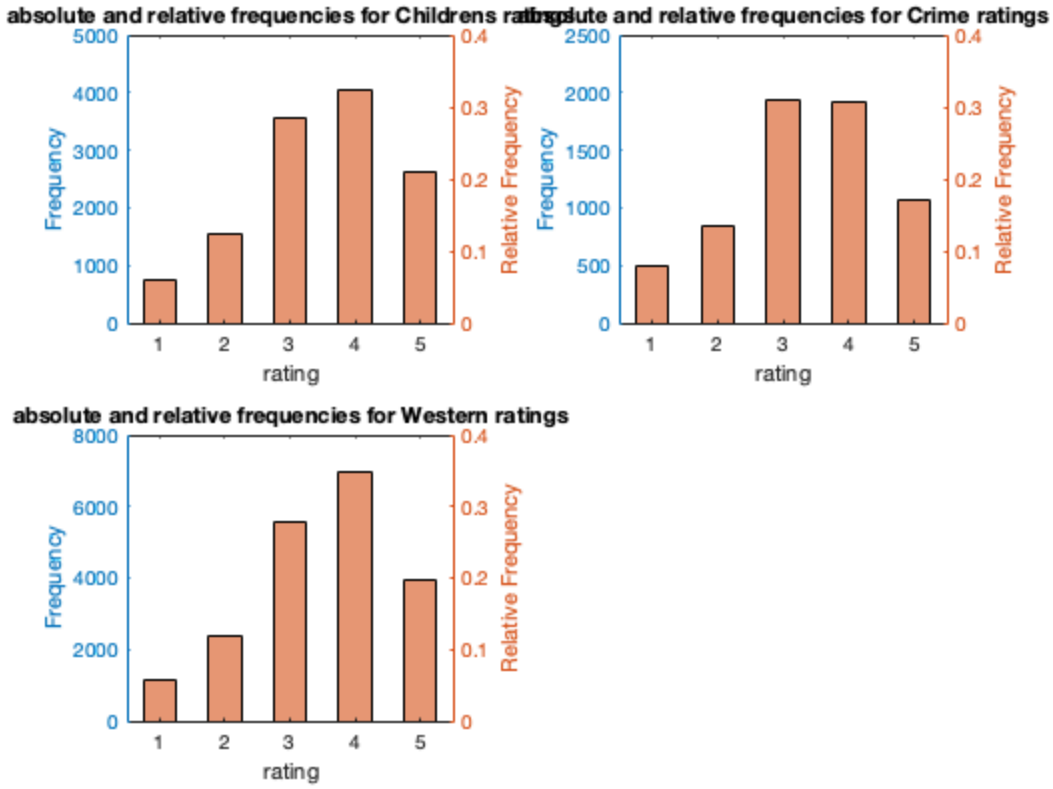
In [241]:
```
#FOR TASK 4.4
#this is to get lists of ratings for genres specified by their indices in 'genres'.
#Selected genres randomly but we ended up plotting for (crime,western,children)
# which have genre indices 4,6,18. This constructs the arrays which were used for
#plotting in MATLAB.
genres = list(np.random.randint(0,19,size=(3,1)))
genre_scores = [[],[],[]]
clean_ratings = np.loadtxt('clean_ratings.txt',dtype=int)
movies = pd.read_csv('movies.txt', sep="\t",header=None)

for row in clean_ratings:
    user_id,movie_id,rating = row[0],row[1],row[2]
    genre_row = movies.loc[movie_id-1][2:]
    for i,genre in enumerate(genres):
        if int(genre_row[genre]) == 1: #if movie is of a particular genre,append the rating
            genre_scores[i].append(rating)
```

In [243]:
```
np.savetxt('genre_array1.txt',np.array(genre_scores[0]),fmt='%d')
np.savetxt('genre_array2.txt',np.array(genre_scores[1]),fmt='%d')
np.savetxt('genre_array3.txt',np.array(genre_scores[2]),fmt='%d')
```

```
%CODE USED FOR 4.4

%each genre array simply contains all ratings for movies of that genre
load('genre_array1.txt')
load('genre_array2.txt')
load('genre_array3.txt')
```

FOR INDIVIDUAL PLOTS

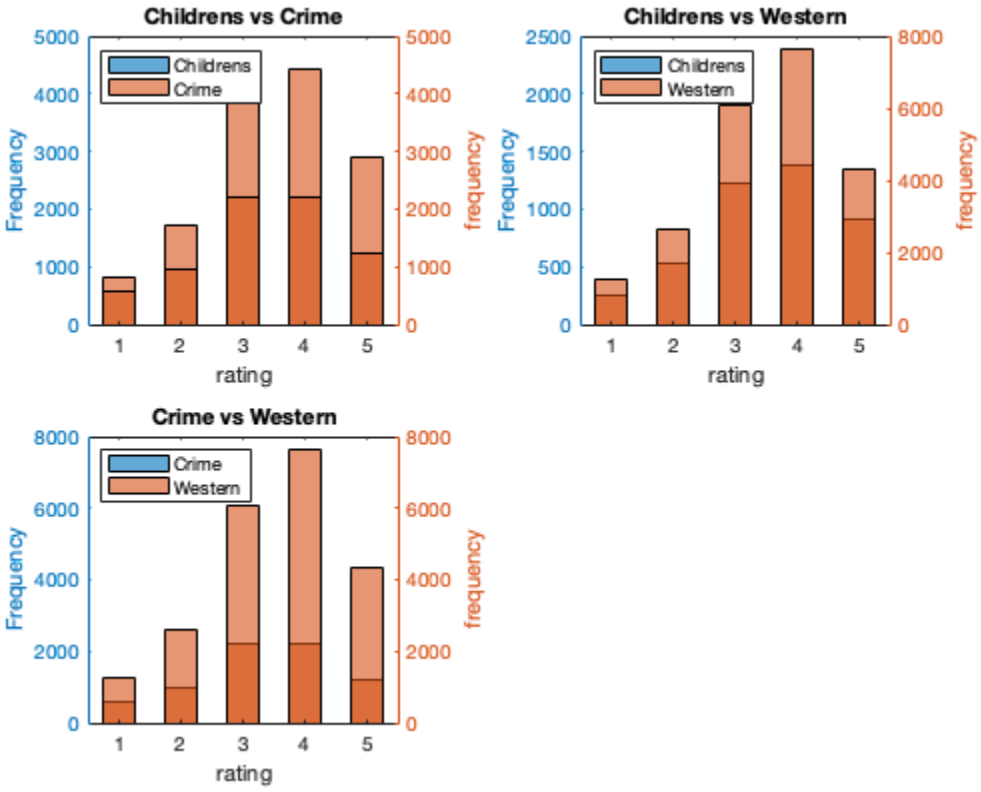
```
%%histogram for genre1
subplot(2,2,1)
yyaxis left
C = categorical(genre_array1,[1 2 3 4 5],{'1','2','3','4','5'});
histogram(C,'BarWidth',0.5)
xlabel('rating')
ylabel('frequency')
yyaxis right
histogram(C,'BarWidth',0.5,'Normalization','probability')
yyaxis left
title('absolute and relative frequencies for Childrens ratings')
ylabel('Frequency')
yyaxis right
ylabel('Relative Frequency')
%%histogram for genre2
subplot(2,2,2)
yyaxis left
C = categorical(genre_array2,[1 2 3 4 5],{'1','2','3','4','5'});
histogram(C,'BarWidth',0.5)
xlabel('rating')
ylabel('frequency')
yyaxis right
histogram(C,'BarWidth',0.5,'Normalization','probability')
yyaxis left
title('absolute and relative frequencies for Crime ratings')
ylabel('Frequency')
yyaxis right
ylabel('Relative Frequency')
%%histogram for genre3
subplot(2,2,3)
yyaxis left
C = categorical(genre_array3,[1 2 3 4 5],{'1','2','3','4','5'});
histogram(C,'BarWidth',0.5)
xlabel('rating')
ylabel('frequency')
yyaxis right
histogram(C,'BarWidth',0.5,'Normalization','probability')
yyaxis left
title('absolute and relative frequencies for Western ratings')
ylabel('Frequency')
yyaxis right
ylabel('Relative Frequency')
```



FOR PAIRWISE COMPARISON PLOTS

```
subplot(2,2,1)
C = categorical(genre_array1,[1 2 3 4 5],{'1','2','3','4','5'});
C2 = categorical(genre_array2,[1 2 3 4 5],{'1','2','3','4','5'});
h1 = histogram(C,'BarWidth',0.5); lbl = 'Childrens';
hold on;
h2 = histogram(C2,'BarWidth',0.5); lbl2 = 'Crime';
legend(lbl,lbl2,'Location','northwest')
title('Childrens vs Crime')
ylabel('frequency')
xlabel('rating')

subplot(2,2,2)
C = categorical(genre_array1,[1 2 3 4 5],{'1','2','3','4','5'});
C2 = categorical(genre_array3,[1 2 3 4 5],{'1','2','3','4','5'});
h1 = histogram(C,'BarWidth',0.5); lbl = 'Childrens';
hold on;
h2 = histogram(C2,'BarWidth',0.5); lbl2 = 'Western';
legend(lbl,lbl2,'Location','northwest')
title('Childrens vs Western')
ylabel('frequency')
xlabel('rating')

subplot(2,2,3)
```

```
C = categorical(genre_array2,[1 2 3 4 5],{'1','2','3','4','5'});
C2 = categorical(genre_array3,[1 2 3 4 5],{'1','2','3','4','5'});
h1 = histogram(C,'BarWidth',0.5); lbl = 'Crime';
hold on;
h2 = histogram(C2,'BarWidth',0.5); lbl2 = 'Western';
legend(lbl,lbl2,'Location','northwest')
title('Crime vs Western')
ylabel('frequency')
xlabel('rating')
```



Published with MATLAB® R2018b

In [262]:
```
#CODE FOR THE HEATMAP
movies = pd.read_csv('movies.txt', sep="\t",header=None)
#arrays to store sum of ratings for each decade,genre pair, as well as count for averaging later
years_genres = np.zeros((8,19)) #20s up to 90s are dim1. genres are dim2
years_genres_counts = np.zeros((8,19))
```

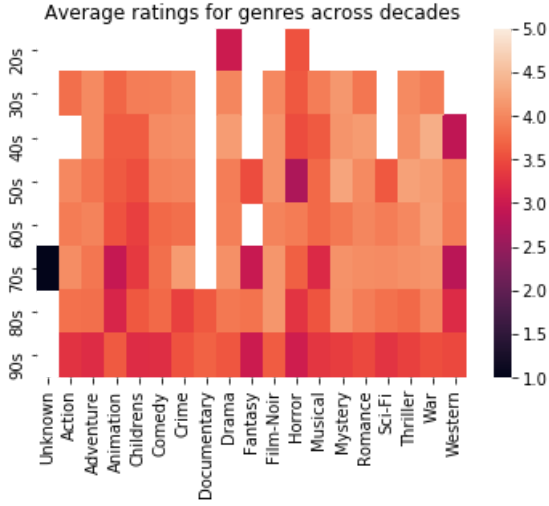
In [314]:
```
for row in clean_ratings:
    user_id,movie_id,rating = row[0],row[1],row[2]
    movie_info = movies.loc[movie_id-1]
    movie_title = movie_info[1]
    if movie_title == 'unknown': #ignore this movie
        continue
    year = re.findall('(\d{4})', movie_title)[-1] #regex to match 4digit sequencees.year always last.
    decade = int(year[2]) #3rd int in year indicates which decade
    genres = np.array(movie_info[2:])
    for i in range(19): #for each genre the movie is a member of, we add its rating and +1 to counts
        if int(genres[i]) == 1:
            years_genres[decade-2,i] += rating
            years_genres_counts[decade-2,i] += 1
```

In [272]:
```
avg_years_genres = np.divide(years_genres,years_genres_counts) #array holding average ratings
```

```
/Users/luiscosta/miniconda3/envs/myenv/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWa
rning: invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
```

In [306]:
```
sns.heatmap(avg_years_genres,yticklabels = ylabels,xticklabels = xlabels,vmin=1,vmax=5)
plt.title('Average ratings for genres across decades')
```

Out[306]:
```
Text(0.5, 1, 'Average ratings for genres across decades')
```



In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

HELPER FUNCTIONS

In [4]:
```
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 [5]:
```
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', 'Animat
ion', 'Childrens', 'Comedy', 'Crime', 'Documentary',
#'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-
Fi', 'Thriller', 'War', 'Western']

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

Basic Method

In [6]:
```
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 us
er,
    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^T

    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
```

In [ ]:
```
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.

    Returns a tuple (U, V, err) consisting of U, V, and the unregularized MSE
    of the model.
    """
    a, b = -0.5, 0.5
    U = (b - a) * np.random.random_sample((M, K)) + a
    V = (b - a) * np.random.random_sample((N, K)) + a

    # first iteration, get loss reduction for initial epoch
    err0 = get_err(U, V, Y)
    arr = np.arange(len(Y))
    np.random.shuffle(arr)
    for index in arr:
        i = Y[index, 0] - 1
        j = Y[index, 1] - 1
        Yij = Y[index, 2]
        U[i] -= grad_U(U[i], Yij, V[j], reg, eta)
        V[j] -= grad_V(V[j], Yij, U[i], reg, eta)
    err01 = err0 - get_err(U, V, Y)

    # second through last iterations
    for epoch in range(max_epochs - 1):
        last_err = get_err(U, V, Y)
        arr = np.arange(len(Y))
        np.random.shuffle(arr)
        for index in arr:
            i = Y[index, 0] - 1
            j = Y[index, 1] - 1
            Yij = Y[index, 2]
            U[i] -= grad_U(U[i], Yij, V[j], reg, eta)
            V[j] -= grad_V(V[j], Yij, U[i], reg, eta)
        curr_err = get_err(U, V, Y)
        if (last_err - curr_err) / err01 < eps:
            last_err = curr_err
            break
        last_err = curr_err
    return (U, V, last_err)
```

Bias Term Method

In [7]:
```
def bgrad_U(Yij, Ui, Vj, reg, eta, ai, bj, mu):
    """
    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), ai (the bias term for user), bj (bias
    term for movie), mu (the average of Y)

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

def bgrad_V(Yij, Ui, Vj, reg, eta, ai, bj, mu):
    """
    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), ai (the bias term for user), bj (bias
    term for movie), mu (the average of Y)

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

def bgrad_a(Yij, Ui, Vj, reg, eta, ai, bj, mu):
    """
    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), ai (the bias term for user), bj (bias
    term for movie), mu (the average of Y)

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

def bgrad_b(Yij, Ui, Vj, reg, eta, ai, bj, mu):
    """
    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), ai (the bias term for user), bj (bias
    term for movie), mu (the average of Y)

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

In [ ]:
```
def bget_err(Y, U, V, reg, a, b, mu):
    """
    Takes as input a matrix Y of triples (i, j, Y_ij) where i is the index of a us
er,
    j is the index of a movie, and Y_ij is user i's rating of movie j, the
    user/movie matrices U and V, the bias vectors a and b, and the average observe
d rating mu

    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^T
    """
    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 - mu - np.dot(U[i], V[j]) - a[i] - b[j])**2
    return reg / 2 * (np.linalg.norm(U)**2 + np.linalg.norm(V)**2 + np.linalg.norm
(a)**2 + np.linalg.norm(b)**2) + 0.5 * sum
```

In [ ]:
```
def btrain_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.

    Returns a tuple (U, V, a, b, err) consisting of U, V, the bias vectors, and th
e MSE
    of the model.
    """
    a, b = -0.5, 0.5
    U = (b - a) * np.random.random_sample((M, K)) + a
    V = (b - a) * np.random.random_sample((N, K)) + a
    A = (b - a) * np.random.random_sample((M, 1)) + a # bias for user
    B = (b - a) * np.random.random_sample((N, 1)) + a # bias for movie
    mu = np.mean(Y[:, 2]) # average of all observed rating

    # first iteration, get loss reduction for initial epoch
    err0 = bget_err(Y, U, V, reg, A, B, mu)
    arr = np.arange(len(Y))
    np.random.shuffle(arr)
    for index in arr:
        i = Y[index, 0] - 1
        j = Y[index, 1] - 1
        Yij = Y[index, 2]
        Ui, Vj, Ai, Bj = U[i], V[j], A[i], B[j]
        U[i] -= bgrad_U(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
        V[j] -= bgrad_V(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
        A[i] -= bgrad_a(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
        B[j] -= bgrad_b(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
    err01 = err0 - bget_err(Y, U, V, reg, A, B, mu)
    print(err01)

    # second through last iterations
    for epoch in range(max_epochs - 1):
        last_err = bget_err(Y, U, V, reg, A, B, mu)
        arr = np.arange(len(Y))
        np.random.shuffle(arr)
        for index in arr:
            i = Y[index, 0] - 1
            j = Y[index, 1] - 1
            Yij = Y[index, 2]
            Ui, Vj, Ai, Bj = U[i], V[j], A[i], B[j]
            U[i] -= bgrad_U(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
            V[j] -= bgrad_V(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
            A[i] -= bgrad_a(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
            B[j] -= bgrad_b(Yij, Ui, Vj, reg, eta, Ai, Bj, mu)
        curr_err = bget_err(Y, U, V, reg, A, B, mu)
        print('change in err / initial = ' + str((last_err - curr_err) / err01))
        if (last_err - curr_err) / err01 < eps:
            last_err = curr_err
            break
        last_err = curr_err
    return (U, V, A, B, last_err)
```

In [ ]:
```
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', encoding='utf-8') 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)
```

Basic Method Training

In [8]:
```
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)
```

Bias Term Method Training

In [9]:
```
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 = 0.1
eta = 0.03 # learning rate

print("Training model with M = %s, N = %s, k = %s, eta = %s, reg = %s"%(M, N, k, e
ta, reg))
bU, bV, A, B, e_in = btrain_model(M, N, k, eta, reg, Y_train)
```

```
Training model with M = 943, N = 1682, k = 20, eta = 0.03, reg = 0.1
[32786.5871227]
change in err / initial = [0.06823691]
change in err / initial = [0.04008331]
change in err / initial = [0.037765]
change in err / initial = [0.03760799]
change in err / initial = [0.0327737]
change in err / initial = [0.03067323]
change in err / initial = [0.02888244]
change in err / initial = [0.02354312]
change in err / initial = [0.02084751]
change in err / initial = [0.01869495]
change in err / initial = [0.01650587]
change in err / initial = [0.01298676]
change in err / initial = [0.01122367]
change in err / initial = [0.01479754]
change in err / initial = [0.00723683]
change in err / initial = [0.00894381]
change in err / initial = [0.0057497]
change in err / initial = [0.00738018]
change in err / initial = [0.00532938]
change in err / initial = [0.00775237]
change in err / initial = [0.00287223]
change in err / initial = [0.00553559]
change in err / initial = [0.00418568]
change in err / initial = [0.00322388]
change in err / initial = [0.00142367]
change in err / initial = [0.00394849]
change in err / initial = [0.00176765]
change in err / initial = [0.00527642]
change in err / initial = [-0.00027041]
```

In [ ]:
```
e_in /= len(Y_train)
e_out = get_err(Y_test, U, V, reg, A, B, np.mean(Y_test[:, 2]))/ len(Y_test)
```

In [ ]:
```
print('E_in is ' + str(e_in))
print('E_out is ' + str(e_out))
```

Off the Shelf

In [10]:
```
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

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_off, Sigma, V_off = off_train(M, N, Y_train)
```

Find the average rating for each movie

In [19]:
```
movie_rating = np.zeros((1682,))
movie_num_user_rating = np.zeros((1682,))
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_ratin
g))
print(movie_avg_rating)
```

```
[3.87831858 3.20610687 3.03333333 ... 2.         3.         3.        ]
```

Importing outside library AdjustText to make movie names not overlap

In [116]:
```
!pip install adjustText
```

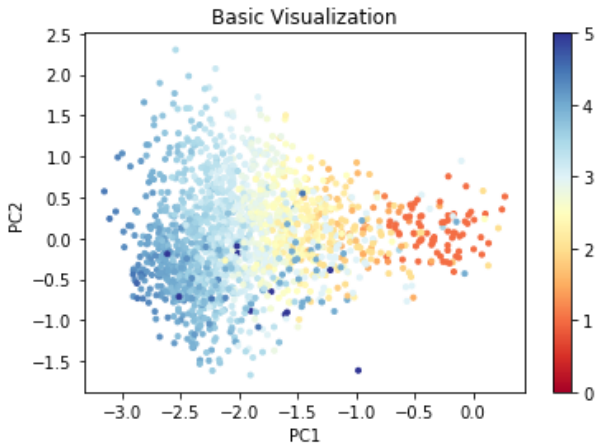
```
Collecting adjustText
  Downloading https://files.pythonhosted.org/packages/9e/15/4157718bf323fd5f5b81
c891c660d0f388e042d2689a558bf1389632dc44/adjustText-0.7.3.tar.gz
Requirement already satisfied: numpy in c:\users\serena\anaconda3\lib\site-packa
ges (from adjustText) (1.16.5)
Requirement already satisfied: matplotlib in c:\users\serena\anaconda3\lib\site-
packages (from adjustText) (3.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\serena\anaconda3\lib\sit
e-packages (from matplotlib->adjustText) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\serena\anaconda3\li
b\site-packages (from matplotlib->adjustText) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\us
ers\serena\anaconda3\lib\site-packages (from matplotlib->adjustText) (2.4.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\serena\anaconda
3\lib\site-packages (from matplotlib->adjustText) (2.8.0)
Requirement already satisfied: six in c:\users\serena\anaconda3\lib\site-package
s (from cycler>=0.10->matplotlib->adjustText) (1.12.0)
Requirement already satisfied: setuptools in c:\users\serena\anaconda3\lib\site-
packages (from kiwisolver>=1.0.1->matplotlib->adjustText) (41.4.0)
Building wheels for collected packages: adjustText
  Building wheel for adjustText (setup.py): started
  Building wheel for adjustText (setup.py): finished with status 'done'
  Created wheel for adjustText: filename=adjustText-0.7.3-cp37-none-any.whl size
=7104 sha256=c4263acf1a0d03153ae0fe6a71ff16a917ac07de66488c34eadd3235078fb502
  Stored in directory: C:\Users\serena\AppData\Local\pip\Cache\wheels\41\95\74\7
d347e136d672f8bc28e937032bc92baf4f80856763a7e7b72
Successfully built adjustText
Installing collected packages: adjustText
Successfully installed adjustText-0.7.3
```

Matrix Visualization with PC1 and PC2

In [20]:
```
from adjustText import adjust_text

def visualize_2d(M, title, index, marker_sz, **kwargs):
    """Project a matrix into 2 dimensions and visualize it.
    args:
    M - matrix to project (V matrix)
    index - indices of the movies to project
    names - names of the movies for labeling

    """
    names = kwargs.get('names', None)

    A, sigma, B = np.linalg.svd(M)
    M_proj = np.matmul(A[:,:2].transpose(), M)

    cm = plt.cm.get_cmap('RdYlBu')

    sc = plt.scatter(M_proj[0,index], M_proj[1,index], s=marker_sz**2, vmin=0,vmax
=5, c=movie_avg_rating[index], cmap=cm)
    if names != None:
        texts = []
        for i, name in zip(index, names):
            texts.append(plt.annotate(name, (M_proj[0, i], M_proj[1, i])))
        adjust_text(texts, autoalign='y')
    plt.colorbar(sc)
    plt.title(title)
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.show()

    return M_proj
```

Basic All Movies

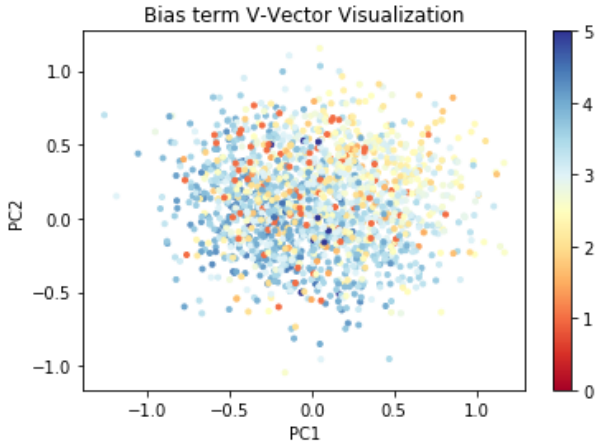
In [26]:
```
index = range((V.T).shape[1])
title = 'Basic Visualization'
visualize_2d(V.T,title, index, 3)
```



Out[26]:
```
array([[-2.77240821, -2.1366022 , -2.03999763, ..., -0.44030291,
        -0.91240738, -0.96931671],
       [ 1.14030668,  0.57141901,  0.87178507, ...,  0.07297196,
         0.30603668, -0.47763089]])
```

Bias Term All Movies

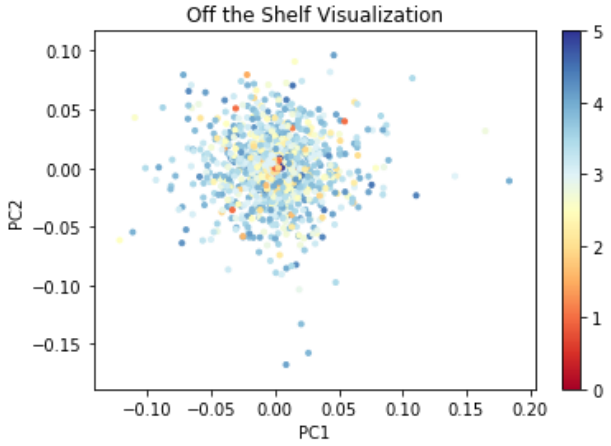
In [42]:
```
index = range((bV.T).shape[1])
visualize_2d(bV.T, 'Bias term V-Vector Visualization',index, 3)
```



Out[42]:
```
array([[ 0.01393449,  0.45263381,  0.21225774, ..., -0.13400777,
        -0.37313518,  0.16808487],
       [-0.13288461,  0.05326039,  0.36411833, ...,  0.0366133 ,
        -0.18325579, -0.0774266 ]])
```

Off the Shelf of All Movies

In [50]:
```
index = range((V_off.T).shape[1])
visualize_2d(V_off.T, 'Off the Shelf Visualization',index, 3)
```



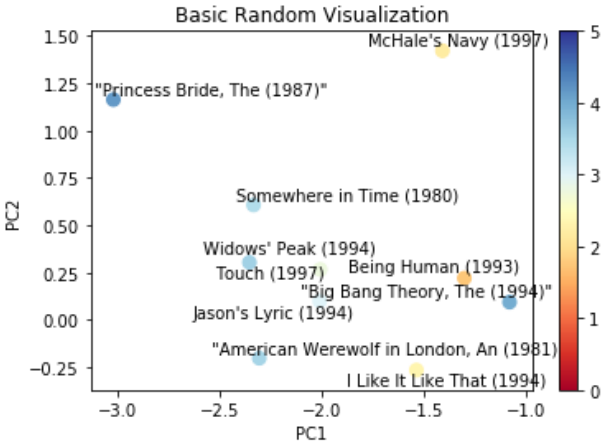
Out[50]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

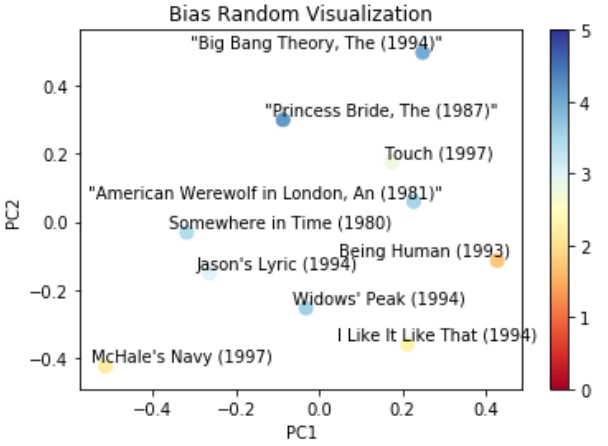
Get Movie Names

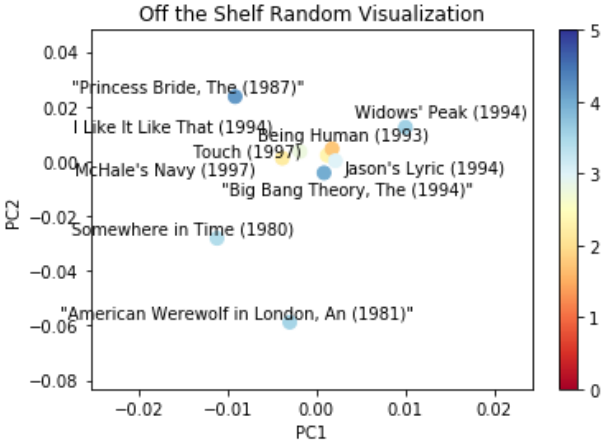
In [14]:
```
all_movies_names = []
with open('data/movies.txt', 'r', encoding='utf-8') as f:
    for line in f:
        line_data = line.strip('\n').split('\t')
        all_movies_names.append(line_data[1])

def get_movie_names(index):
    chosen = []
    for i in index:
        chosen.append(all_movies_names[i])
    return chosen
```

10 randomly selected movies

In [21]:
```
import random

num_movies = 1682
rand_index = np.random.choice(num_movies, 10, replace=False)
chosen_movie_names = get_movie_names(rand_index)
visualize_2d(V.T, 'Basic Random Visualization',rand_index, 8, names=chosen_movie_n
ames)
visualize_2d(bV.T, 'Bias Random Visualization',rand_index, 8, names=chosen_movie_n
ames)
visualize_2d(V_off.T, 'Off the Shelf Random Visualization',rand_index, 8, names=ch
osen_movie_names)
```





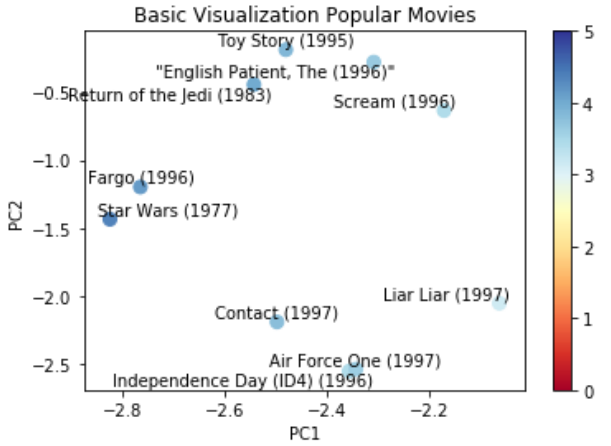


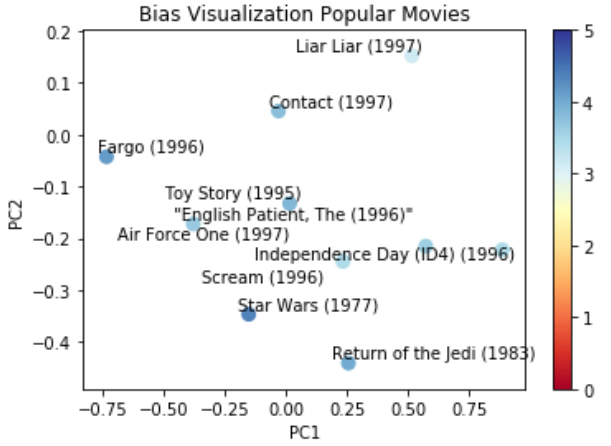
Out[21]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

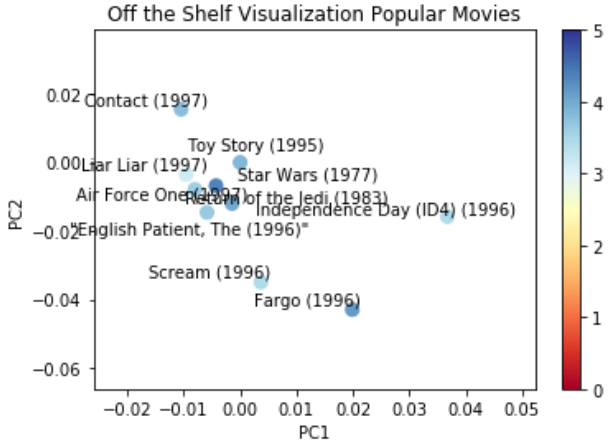
Top 10 Most Popular Movies

In [52]:
```
# index of top 10 by number of ratings
copy_num_rating = movie_num_user_rating.copy()
index_top10_popular = copy_num_rating.argsort()[-10:]
chosen_movie_names = get_movie_names(index_top10_popular)
for name in chosen_movie_names:
    print(name)
visualize_2d(V.T, 'Basic Visualization Popular Movies',index_top10_popular, 8, nam
es=chosen_movie_names)
visualize_2d(bV.T, 'Bias Visualization Popular Movies',index_top10_popular, 8, nam
es=chosen_movie_names)
visualize_2d(V_off.T, 'Off the Shelf Visualization Popular Movies',index_top10_pop
ular, 8, names=chosen_movie_names)

# maybe need to include genre??
```

```
Independence Day (ID4) (1996)
Air Force One (1997)
Toy Story (1995)
Scream (1996)
"English Patient, The (1996)"
Liar Liar (1997)
Return of the Jedi (1983)
Fargo (1996)
Contact (1997)
Star Wars (1977)
```





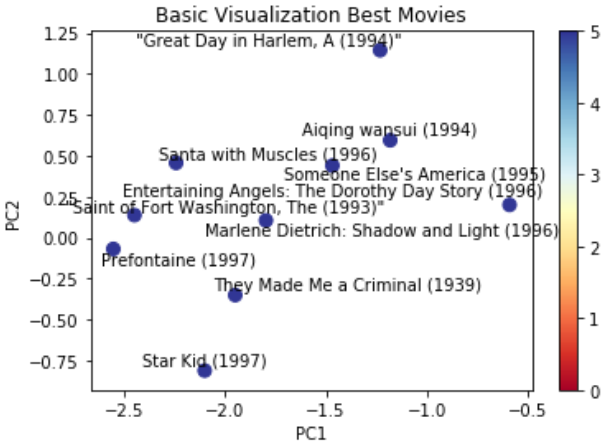


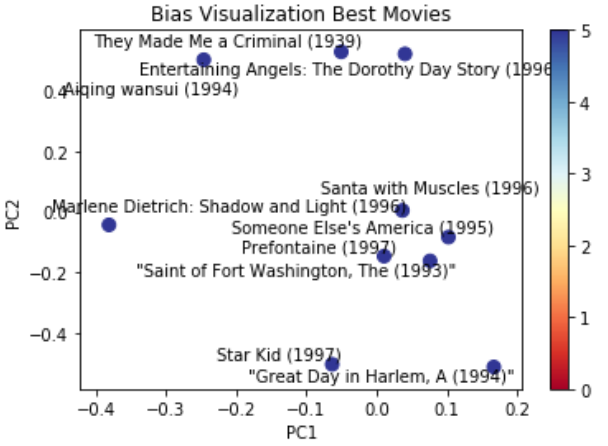
Out[52]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

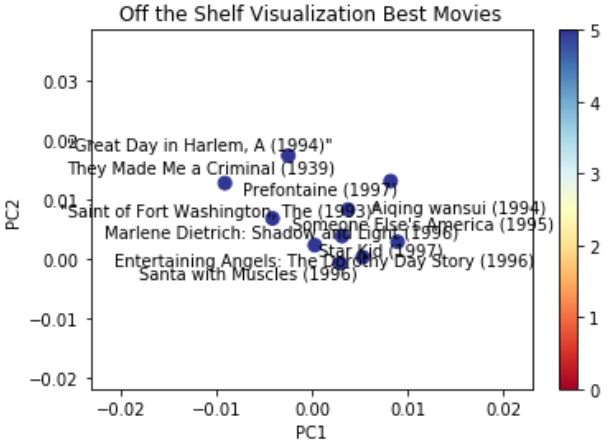
Top 10 Best Movies by Ratings

In [53]:
```
# index of top 10 by number of ratings
copy_avg_rating = movie_avg_rating.copy()
index_top10_rating = copy_avg_rating.argsort()[-10:]
chosen_movie_names = get_movie_names(index_top10_rating)
for name in chosen_movie_names:
    print(name)
visualize_2d(V.T,'Basic Visualization Best Movies', index_top10_rating, 8, names=c
hosen_movie_names)
visualize_2d(bV.T,'Bias Visualization Best Movies', index_top10_rating, 8, names=c
hosen_movie_names)
visualize_2d(V_off.T,'Off the Shelf Visualization Best Movies', index_top10_ratin
g, 8, names=chosen_movie_names)

# maybe need to include genre??
```

```
Aiqing wansui (1994)
Santa with Muscles (1996)
Prefontaine (1997)
Marlene Dietrich: Shadow and Light (1996)
Someone Else's America (1995)
They Made Me a Criminal (1939)
"Great Day in Harlem, A (1994)"
Entertaining Angels: The Dorothy Day Story (1996)
"Saint of Fort Washington, The (1993)"
Star Kid (1997)
```







Out[53]:
```
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]])
```

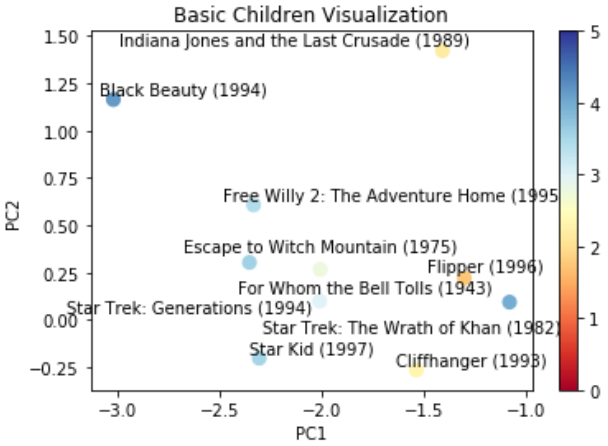
Selected Genre: Children, Crime, Western

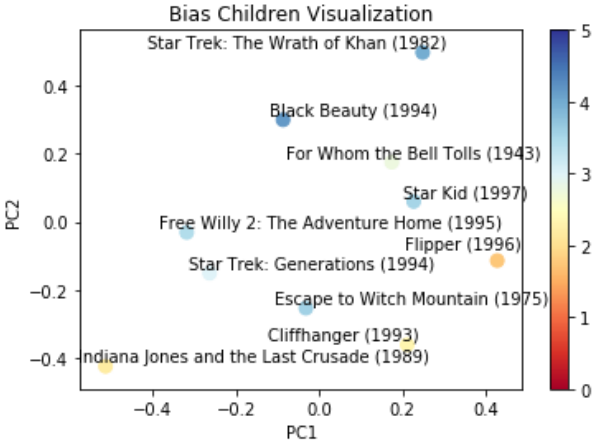
Getting the movie names and id from the 3 genres

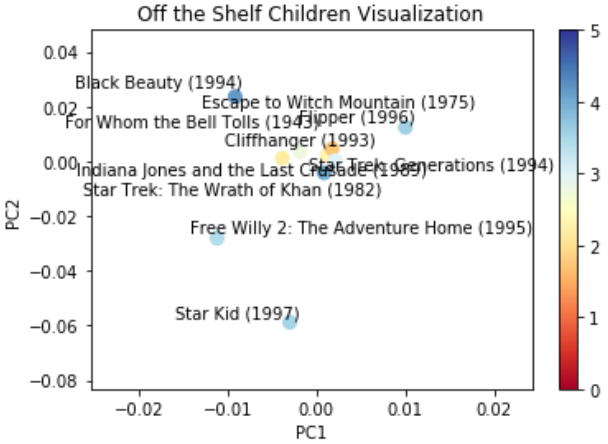
In [22]:
```
# (children, crime, western,) have indices 4,6,18
genres = [4, 6, 18]
movies_by_genre = [[], [], []] # children, crime, western
movies = pd.read_csv('data/movies.txt', sep="\t",header=None)
for row in range(len(movies)):
    genre_row = movies.loc[row][2:]
    movie_name = movies.loc[row][1]
    movie_id = movies.loc[row][0]
    for i,genre in enumerate(genres):
        if int(genre_row[genre]) == 1:
            movies_by_genre[i].append([movie_id, movie_name])

children, crime, western = np.array(movies_by_genre[0]).T, np.array(movies_by_genr
e[1]).T, np.array(movies_by_genre[2]).T
```

Children Visualization

In [23]:
```
# Children visualization
children_chosen = np.random.choice(children[0].astype(int), 10, replace=False) - 1
chosen_movie_names = get_movie_names(children_chosen)
visualize_2d(V.T, 'Basic Children Visualization',rand_index, 8, names=chosen_movie
_names)
visualize_2d(bV.T, 'Bias Children Visualization',rand_index, 8, names=chosen_movie
_names)
visualize_2d(V_off.T, 'Off the Shelf Children Visualization',rand_index, 8, names=
chosen_movie_names)
```



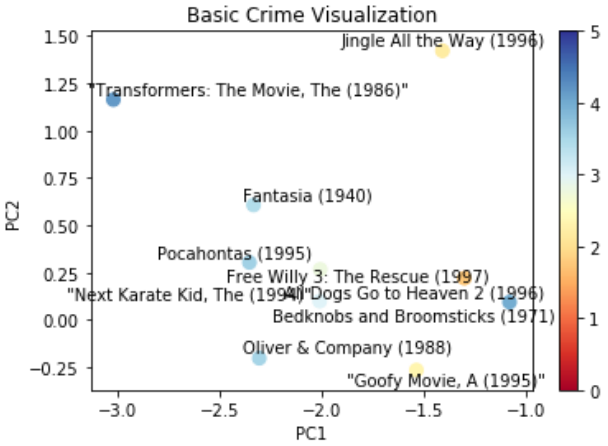


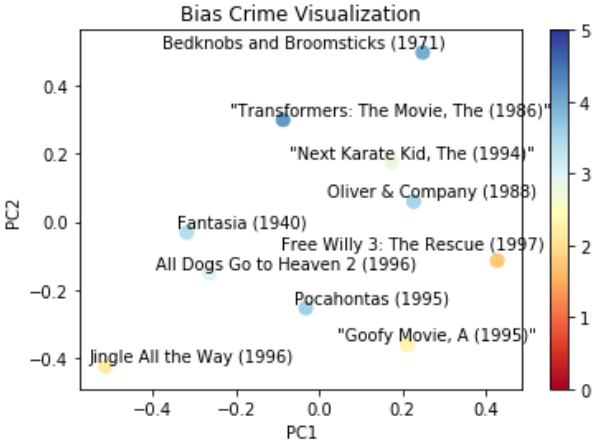


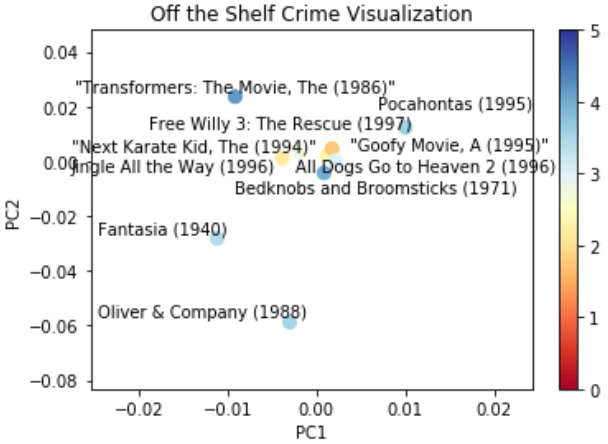
Out[23]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

Crime Visualization

In [24]:
```
# Crime visualization
crime_chosen = np.random.choice(crime[0].astype(int), 10, replace=False) - 1
chosen_movie_names = get_movie_names(crime_chosen)
visualize_2d(V.T, 'Basic Crime Visualization',rand_index, 8, names=chosen_movie_na
mes)
visualize_2d(bV.T, 'Bias Crime Visualization',rand_index, 8, names=chosen_movie_na
mes)
visualize_2d(V_off.T, 'Off the Shelf Crime Visualization',rand_index, 8, names=cho
sen_movie_names)
```



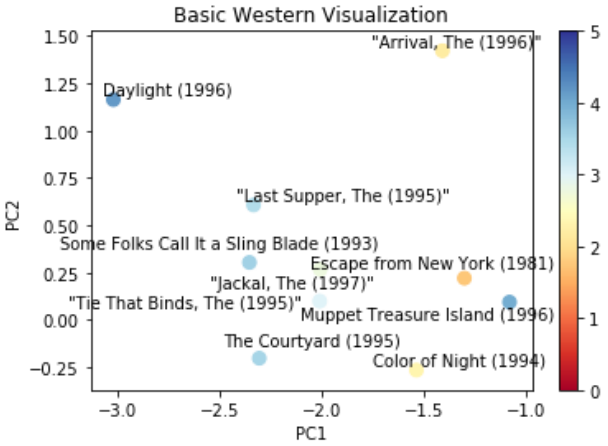


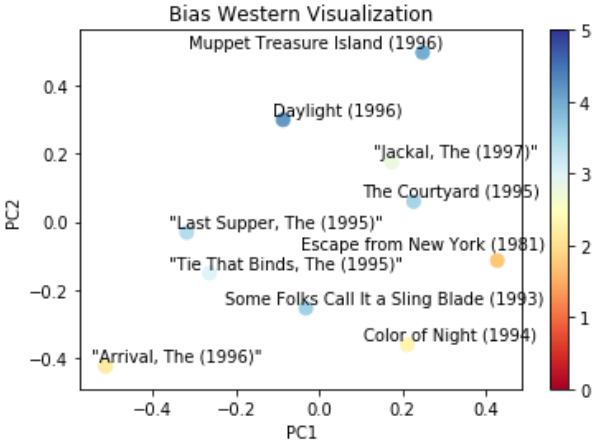


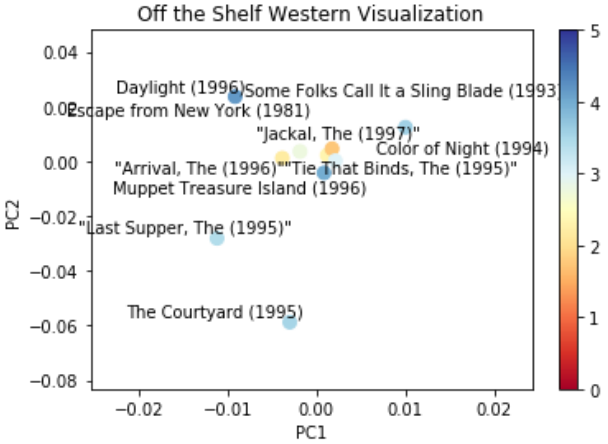
Out[24]:
```
array([[ 3.19975166e-17, -6.81846868e-02,  7.63564244e-02, ...,
         2.49092379e-03, -4.92054703e-03, -1.94694697e-03],
       [ 1.02762037e-17,  1.69763209e-02,  3.23234102e-02, ...,
         3.67341793e-04,  6.49603545e-03,  3.95882009e-03]])
```

Western Visualization

In [25]:
```
# Western visualization
western_chosen = np.random.choice(western[0].astype(int), 10, replace=False) - 1
chosen_movie_names = get_movie_names(western_chosen)
visualize_2d(V.T, 'Basic Western Visualization',rand_index, 8, names=chosen_movie_
names)
visualize_2d(bV.T, 'Bias Western Visualization',rand_index, 8, names=chosen_movie_
names)
visualize_2d(V_off.T, 'Off the Shelf Western Visualization',rand_index, 8, names=c
hosen_movie_names)
```







Out[25]:
```
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 [ ]:

In [ ]: