Exploratory Data Analysis

In [1]:
```
# importing necessary libraries
import pandas as pd
import numpy as np
import json
import csv
import os
import cPickle as pickle
from collections import Counter

from sklearn.cluster import KMeans, SpectralClustering

import networkx as nx

from my_utilities import *

import matplotlib.pyplot as plt
%matplotlib inline
```

Reading the data (stored as dataframes in pickled files)

In [2]:
```
# Loading the pickled data frames from users, businesses, and revie
# Assuming these data frames were obtained already using 'make_data
dataframe_pickle_filename = '../data/dataframes.pkl'
with open(dataframe_pickle_filename, 'r') as f:
    (user_df, business_df, review_df) = pickle.load(f)
```

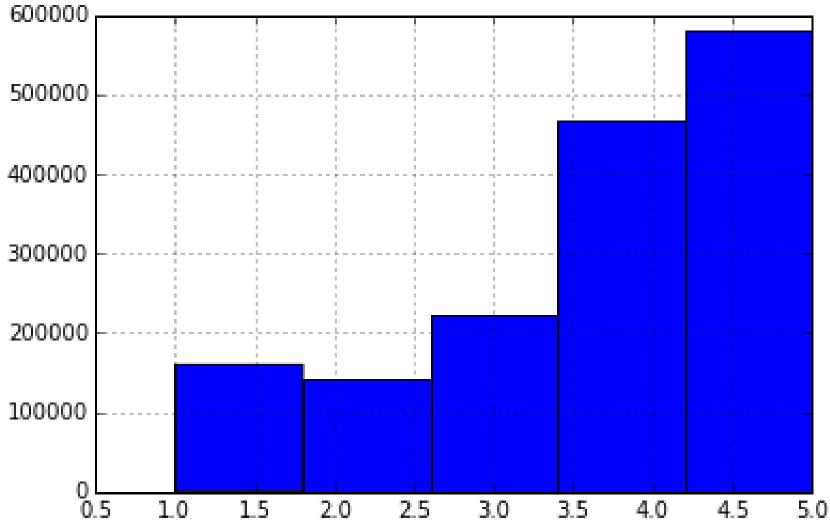
Trying to understand the data

Looking at some overall stats

In [3]:
```
# average rating
print review_df.review_stars.mean()
review_df.review_stars.hist(bins=5)
```

```
3.74265579278
```

Out[3]: `<matplotlib.axes._subplots.AxesSubplot at 0x10a021750>`



Ratings are skewed toward 5.

In [4]:
```
# Standard deviation of the all reviews. (This will be the baseline
review_df.review_stars.std()
```

Out[4]: `1.3114679041155461`

Find cities for businesses using k-means.

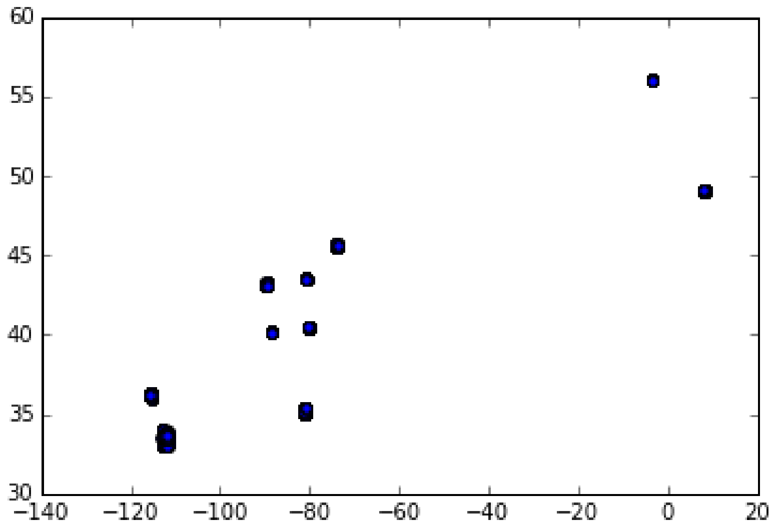
In [5]:
```
X = business_df[['business_latitude', 'business_longitude']].values
```

In [6]:
```
km = KMeans(10, n_jobs=8)
```

In [7]:
```
y = km.fit_predict(X)
```

In [8]:
```
plt.scatter(X[:,1], X[:,0])
```

Out[8]: <matplotlib.collections.PathCollection at 0x10c37c150>



In [9]:
```
city_locations = km.cluster_centers_
```

In [10]:
```
n_businesses_per_city = Counter(y).most_common()
```

In [11]:
```
sorted_y = [c[0] for c in n_businesses_per_city]
sorted_y_map = {}
for i in range(10):
      sorted_y_map[sorted_y[i]] = i
```

In [12]:
```
business_df['business_city_int'] = map(lambda i: sorted_y_map[i], y
```

In [13]:
```
city_names = ['Phoenix', 'Las Vegas', 'Charlotte', 'Montreal', 'Edi
              'Urbana-Champaign', 'Waterloo']
```

Find reviews for businesses in each city

In [14]:
```
# Dataframe with the reviews (with business city).
review_city_df = business_df[['business_id_int', 'business_city_int
```

In [15]:
```
# Reviews for three specific cities.
review_phoenix = review_city_df[review_city_df.business_city_int ==
review_lasvegas = review_city_df[review_city_df.business_city_int =
review_charlotte = review_city_df[review_city_df.business_city_int
review_montreal = review_city_df[review_city_df.business_city_int =
```

In [16]:
```
len(review_phoenix.user_id_int.unique()), len(review_lasvegas.user_
```

Out[16]: (128330, 181712, 24649, 13861)

In [17]:
```
len(review_phoenix.business_id_int.unique()), len(review_lasvegas.b
```

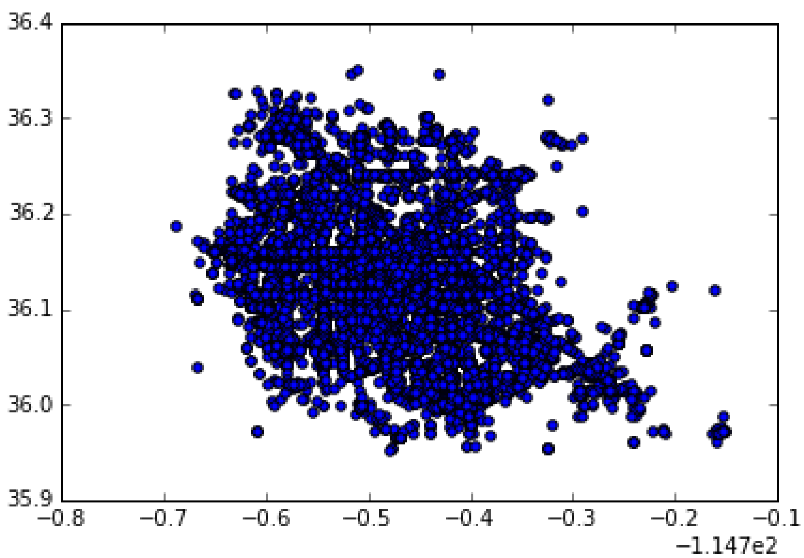
Out[17]: (25225, 16469, 5149, 3918)

In [18]:
```
review_phoenix.shape[0], review_lasvegas.shape[0], review_charlotte
```

Out[18]: (591192, 679460, 98034, 49484)

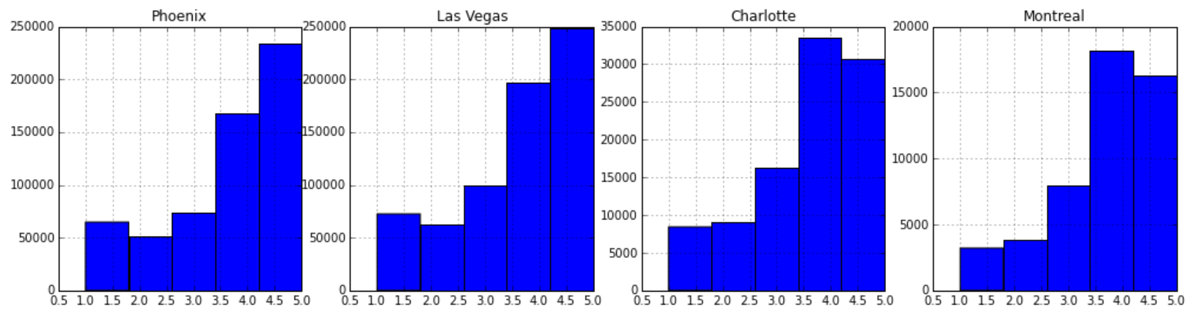
In [19]:
```
# Check the clustering was done correctly.
X = business_df[business_df.business_city_int==1][['business_latitu
plt.scatter(X[:,1], X[:,0])
```

Out[19]: <matplotlib.collections.PathCollection at 0x10e00f5d0>



In [20]:
```
# Looking at review distributions in each city (Phoenix, Las Vegas,
plt.figure(figsize = (17, 4))
plt.subplot(1,4,1)
plt.title("Phoenix")
review_phoenix.review_stars.hist(bins=5)
plt.subplot(1,4,2)
plt.title("Las Vegas")
review_lasvegas.review_stars.hist(bins=5)
plt.subplot(1,4,3)
plt.title("Charlotte")
review_charlotte.review_stars.hist(bins=5)
plt.subplot(1,4,4)
plt.title("Montreal")
review_montreal.review_stars.hist(bins=5)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x10f80fc10>



It is quite interesting that Montreal has quite different review distribution compared to other cities.

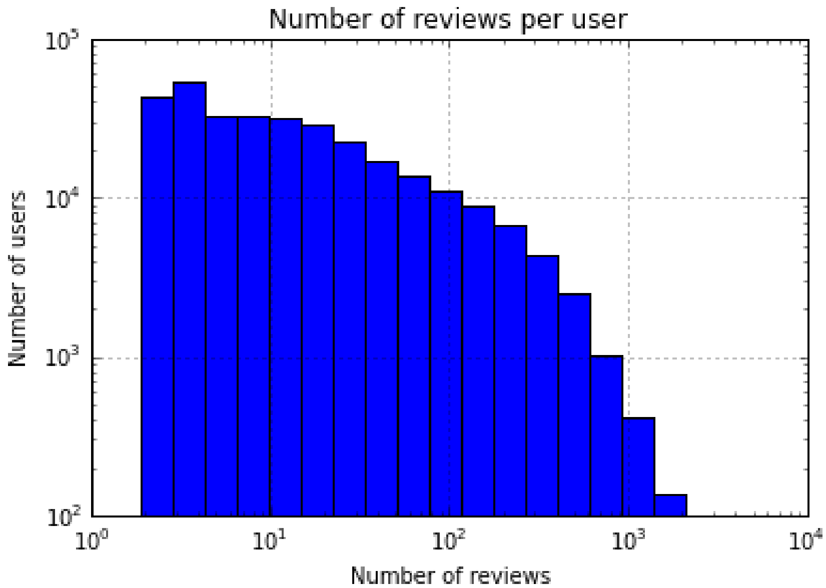
In [21]:
```
print review_phoenix.review_stars.std(), review_lasvegas.review_sta
```

1.34162771089 1.32606119856 1.16331249085

Number of reviews per user.

In [22]:
```
plt.ylim([100,100000])
plt.gca().set_xscale("log")
plt.yscale('log')
plt.title('Number of reviews per user')
plt.xlabel('Number of reviews')
plt.ylabel('Number of users')
user_df.user_review_count.hist(bins=np.logspace(0.1, 3.5, 20))
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x10e028690>

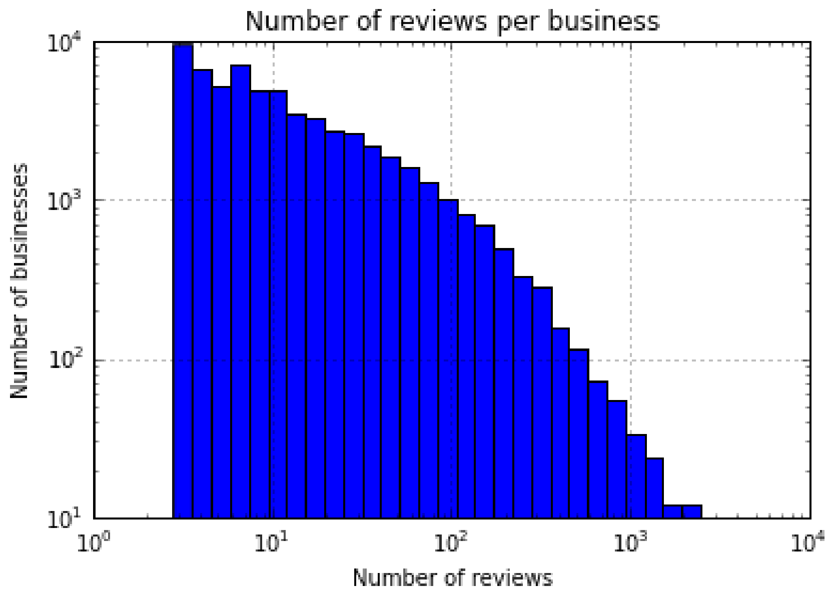


Users have at least 1 review.

Number of reviews per business.

In [23]:
```
plt.ylim([10, 10000])
plt.gca().set_xscale("log")
plt.yscale('log')
plt.title('Number of reviews per business')
plt.xlabel('Number of reviews')
plt.ylabel('Number of businesses')
business_df.business_review_count.hist(bins=np.logspace(0.45, 3.5,
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x10c70c690>



In [24]:
```
business_df.business_review_count.min()
```

Out[24]: 3

Businesses have at least 3 reviews.

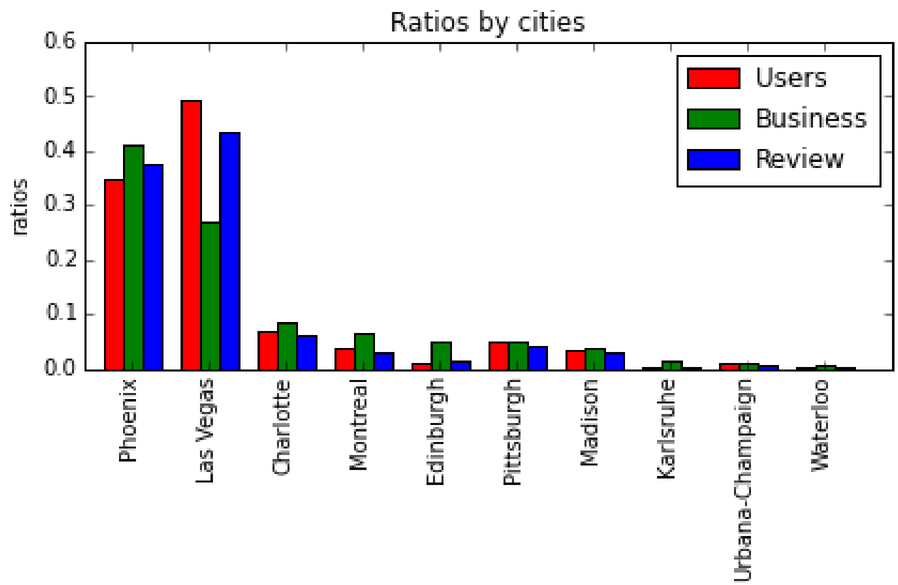
Some counts per city

In [25]:
```
n_users = float(user_df.shape[0])
n_businesses = float(business_df.shape[0])
n_reviews = float(review_df.shape[0])
users_by_city = np.array(review_city_df.groupby('business_city_int'
businesses_by_city = np.array(review_city_df.groupby('business_city
reviews_by_city = np.array(review_city_df.groupby('business_city_in
users_by_city_ratio = users_by_city / n_users
businesses_by_city_ratio = businesses_by_city / n_businesses
reviews_by_city_ratio = reviews_by_city / n_reviews
```

In [26]:
```
def plot_counts(users_by_city, businesses_by_city, reviews_by_city,
    fig = plt.figure()
    ax = fig.add_subplot(111)

    ind = np.arange(10) # the x locations for the groups
    width = 0.25        # the width of the bars

    ## the bars
    rects0 = ax.bar(ind, users_by_city, width, color='r')
    rects1 = ax.bar(ind + width, businesses_by_city, width, color='
    rects2 = ax.bar(ind + 2 * width, reviews_by_city, width, color=

    # axes and labels
    ax.set_xlim(-width, len(ind) + width)
    ax.set_ylim(0, ylim)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.set_xticks(ind + 1.5*width)
    xtickNames = ax.set_xticklabels(city_names)
    plt.setp(xtickNames, rotation=90, fontsize=10)

    ## add a legend
    ax.legend((rects0[0], rects1[0], rects2[0]), ('Users', 'Busines
```

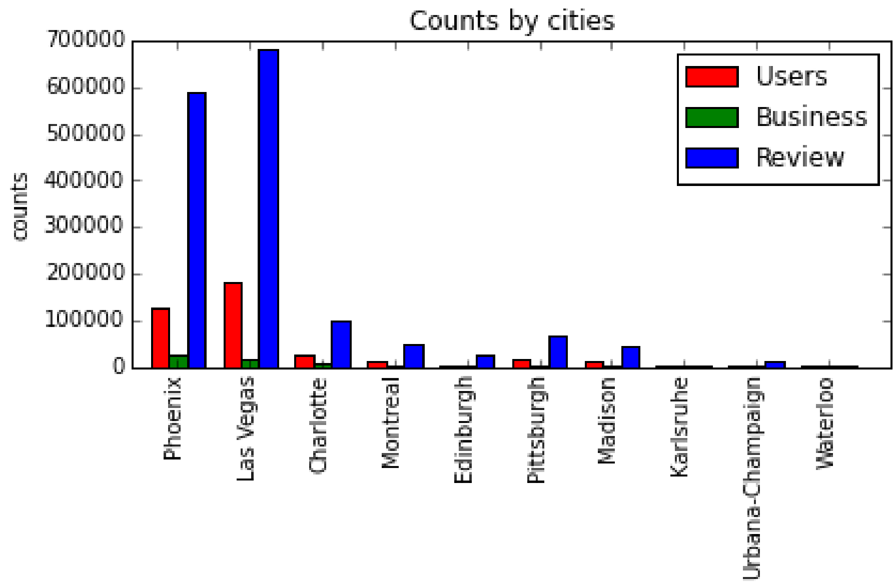
In [27]:
```
plt.figure(num=None, figsize=(8, 5), dpi=200)
plot_counts(users_by_city_ratio, businesses_by_city_ratio, reviews_
            'ratios', ylim=0.6)
plt.tight_layout()
plt.savefig('../fig/ratios_by_city.png', dpi=200)
```

```
<matplotlib.figure.Figure at 0x10c398310>
```



In [28]:
```
plt.figure(num=None, figsize=(8, 5), dpi=200)
plot_counts(users_by_city, businesses_by_city, reviews_by_city, 'Co
            'counts', ylim=700000)
plt.tight_layout()
plt.savefig('../fig/counts_by_city.png', dpi=160)
```

```
<matplotlib.figure.Figure at 0x10e0972d0>
```



Phoenix and Las Vegas are two biggest cities here. A unique characteristic we can see is that in Las Vegas the number of users is much greater than the number of businesses; on the other hand, in Phoenix, the number of businesses is greater than the number of users. At first, I assumed there might be many tourists in Las Vegas, but based on counts below, there are not many users who leave reviews in multiple cities (only 5%). Then two possible explanations are: (1) there are just many more users than businesses, (2) users in Las Vegas are much more active.

Number of cities per user

Here trying to find how many cities each user left reviews.

In [30]:
```
cities_by_user = np.array(review_city_df.groupby('user_id_int').bus
```

In [31]:
```
sum(cities_by_user == 1) / float(n_users)
```

Out[31]:
```
0.95011112171577383
```

In [32]:
```
sum(cities_by_user == 2) / float(n_users)
```

Out[32]:
```
0.045563993837176006
```

In [33]:
```
sum(cities_by_user > 2) / float(n_users)
```

Out[33]: 0.0043248844470501618

Most of users left reviews in only one city (95 %), and 4.56 % of users left reviews in one city. There are only 0.43 % of users who left reviews in more than two cities. Also, the earliest date and the latest date for reviews are 2004-10-12 and 2015-01-08.

In [34]:
```
print review_df.review_date.min(), review_df.review_date.max()
```

```
2004-10-12 2015-01-08
```

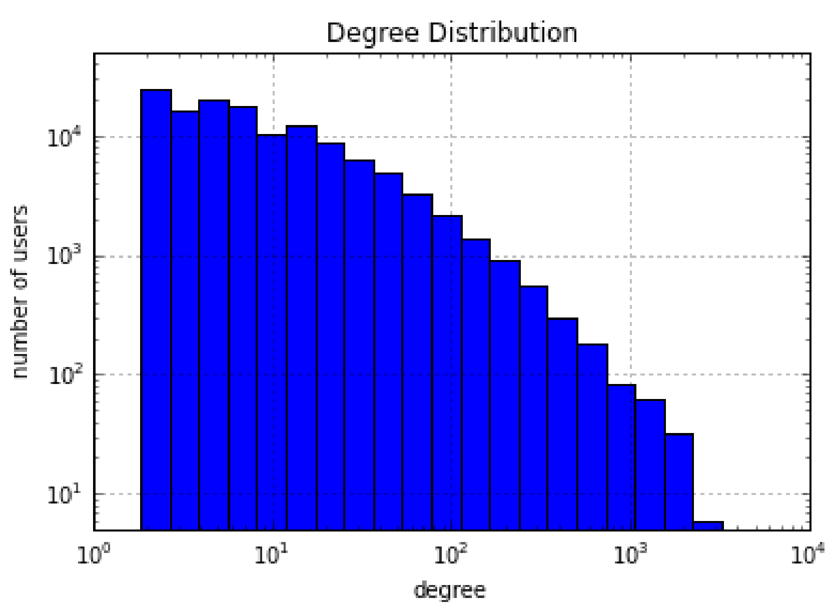
Degree distribution

Now we can look at the network structure. The easiest stat we can look is the degree distribution.

In [35]:
```
degrees = pd.read_csv("../data/degrees", names=['user_id', 'degree'
```

In [36]:
```
def plot_degrees(degree_col, y_min, y_max):
    plt.ylim([y_min, y_max])
    plt.gca().set_xscale("log")
    plt.gca().set_yscale('log')
    plt.title('Degree Distribution')
    plt.xlabel('degree')
    plt.ylabel('number of users')
    degree_col.hist(bins=np.logspace(0.1, 4, 25))
```

In [37]:
```
plot_degrees(degrees.degree, 5, 50000)
```



Finding the number of users with at least one friend using degree info.

In [38]:
```
# A set containing users with at least one friend.
users2_set = set(degrees[degrees.degree > 0].user_id.values)
```

Create new dataframes that are reduced using only above users.

In [39]:
```
# Create all dataframes that only have data with users with at leas
# First, find users first.
user_df2 = user_df[user_df.apply(lambda x: x['user_id_int'] in user
# Second, find reviews done by these users.
review_df2 = review_df[review_df.apply(lambda x: x['user_id_int'] i
# Lastly, find businesses that only these reviews are done for.
businesses2_set = set(review_df2.business_id_int.unique())
business_df2 = business_df[business_df.apply(lambda x: x['business_
```

In [40]:
```
# Dataframe with the reviews (with business city).
review_city_df2 = business_df2[['business_id_int', 'business_city_i
    .merge(review_df2, on=['business_id_int'], how='inner')
```

In [41]:
```
print user_df2.shape[0], business_df2.shape[0], review_df2.shape[0]
```

```
174094 59106 1143736
```

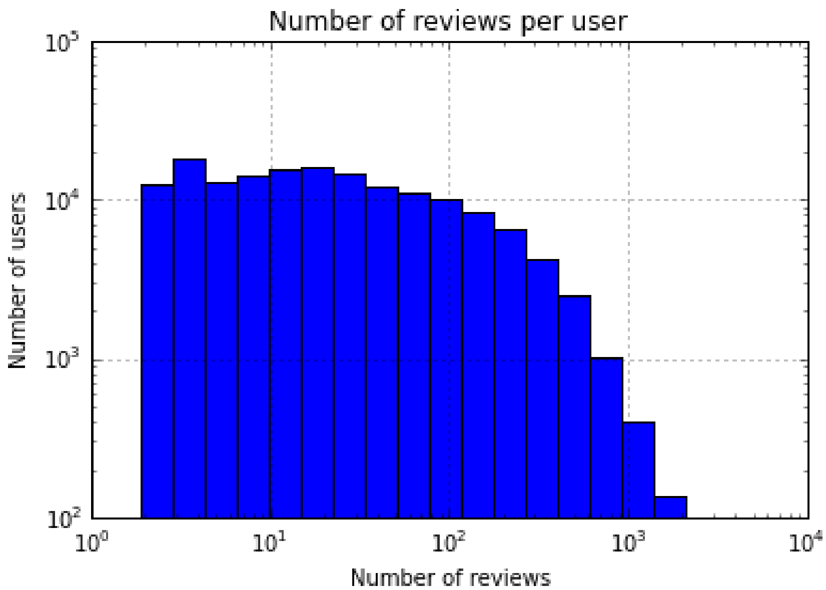
Now doing above analyses with the reduced set.

In [42]:
```
# Reviews for three specific cities.
review_phoenix2 = review_city_df2[review_city_df2.business_city_int
review_lasvegas2 = review_city_df2[review_city_df2.business_city_in
review_montreal2 = review_city_df2[review_city_df2.business_city_in
```

Number of reviews per user

In [43]:
```
plt.ylim([100,100000])
plt.gca().set_xscale("log")
plt.yscale('log')
plt.title('Number of reviews per user')
plt.xlabel('Number of reviews')
plt.ylabel('Number of users')
user_df2.user_review_count.hist(bins=np.logspace(0.1, 3.5, 20))
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x10ca20810>

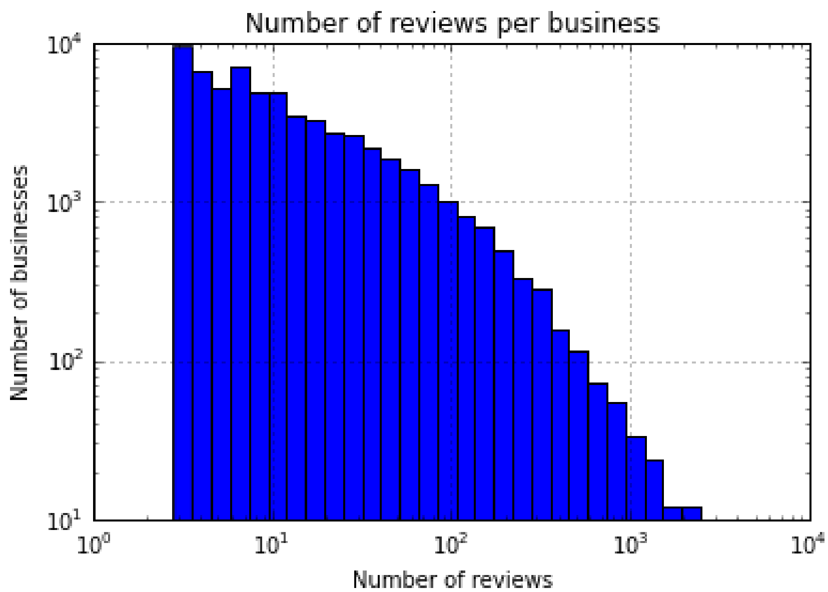


Looks different when number of reviews are small. Seems like users are more active overall.

Number of reviews per business

In [44]:
```
plt.ylim([10, 10000])
plt.gca().set_xscale("log")
plt.yscale('log')
plt.title('Number of reviews per business')
plt.xlabel('Number of reviews')
plt.ylabel('Number of businesses')
business_df.business_review_count.hist(bins=np.logspace(0.45, 3.5,
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x10f1b7310>

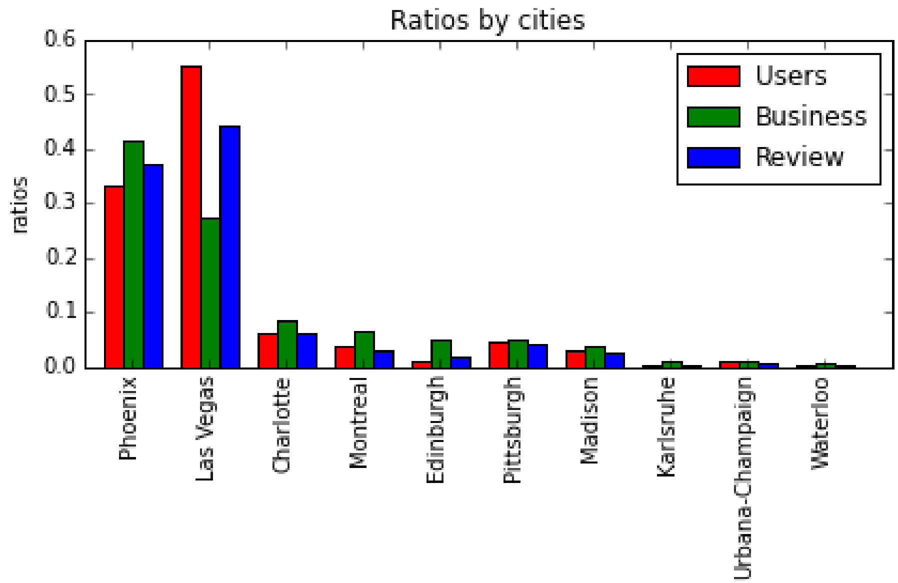


Looks almost the same as before.

Some counts per city

In [45]:
```
n_users2 = float(user_df2.shape[0])
n_businesses2 = float(business_df2.shape[0])
n_reviews2 = float(review_df2.shape[0])
users_by_city_ratio2 = np.array(review_city_df2.groupby('business_c
businesses_by_city_ratio2 = np.array(review_city_df2.groupby('busin
reviews_by_city_ratio2 = np.array(review_city_df2.groupby('business
```

In [46]:
```
plot_counts(users_by_city_ratio2, businesses_by_city_ratio2, review
            'ratios', ylim=0.6)
plt.tight_layout()
plt.savefig('../fig/ratios_by_city.png', dpi=200)
```



Overall trends are the same here.

Degree distribution should be the same as before, except users with degree zero.

In [47]:
```
cities_by_user2 = np.array(review_city_df2.groupby('user_id_int').b
```

In [48]:
```
print sum(cities_by_user2 == 1) / float(n_users2),\
    sum(cities_by_user2 == 2) / float(n_users2),\
    sum(cities_by_user2 > 2) / float(n_users2)
```

```
0.920278700013 0.0714211862557 0.00830011373166
```

User by city computed from 'find_users_by_city.py'.

Here it uses the network friends to determine the city if a user has reviews in multiple cities. If threre are ties between friends, it chooses the city randomly. As shown above, 92.1% of users have reviews in only one city, and 0.8% of users are chosen randomly, which means out of users with multiple cities, about 10% of them are chosen randomly.

In [49]:
```
user_by_city = pd.read_csv('../data/user_by_city', names=['user_id'
user_by_city.groupby('city').count()
```

Out[49]:

	user_id
city	
0	52104
1	92761
2	9257
3	5270
4	1450
5	6928
6	4305
7	135
8	1507
9	377

Looking at subnetworks for each city.

In [50]:
```
# Dataframes for each city.
degrees_subnet = []
for i in range(10):
    degrees_subnet.append(pd.read_csv('../data/degrees%s' % i, name
```

In [51]:
```
# Number of edges for all subgraphs.
nedges_subnet = []
for i in range(10):
    nedges_subnet.append(degrees_subnet[i].values.sum(axis=0)[1])
    print nedges_subnet[i]
print sum(nedges_subnet)
```

```
389936
1516460
43082
18657
9048
26391
13900
228
2652
522
2020876
```

In [52]:
```
# Number of edges for the total graph.
nedges = degrees.values.sum(axis=0)[1]
print nedges
```

```
2576179
```

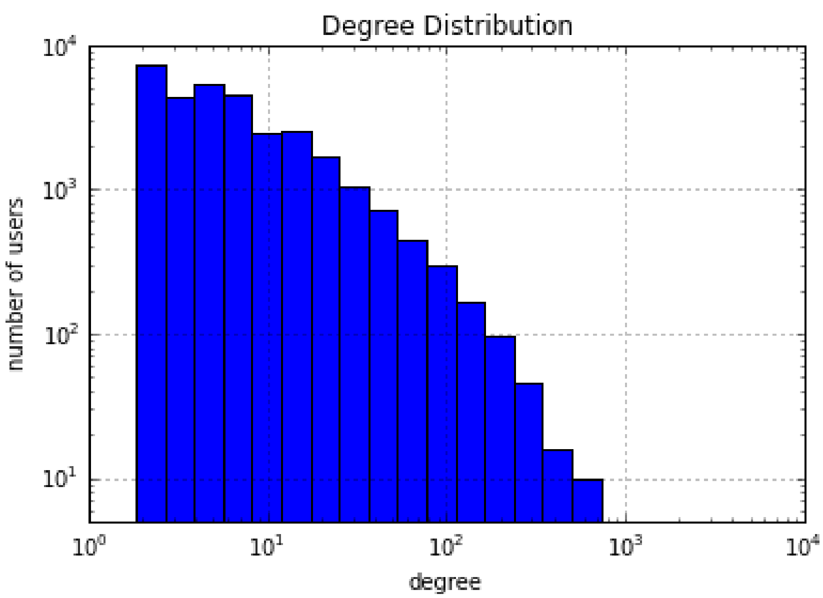
In [53]:
```
# Percentage of edges that are connecting inside communities.
sum(nedges_subnet)/float(nedges)
```

Out[53]:
```
0.78444704347019367
```

Degree distributions for subnets for three cities.

Phoenix

In [54]:
```
plot_degrees(degrees_subnet[0].degree, 5, 10000)
plt.savefig('../fig/degree_phoenix.png')
```

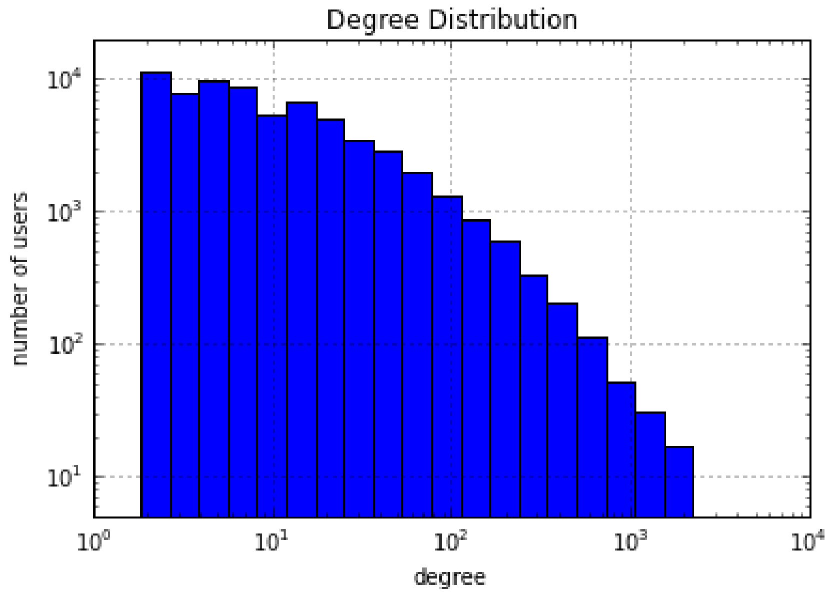


In [55]:
```
print degrees_subnet[0].degree.mean(), degrees_subnet[0].degree.med
```

```
8.5645632454 3.0
```

Las Vegas

In [56]:
```
plot_degrees(degrees_subnet[1].degree, 5, 20000)
plt.savefig('../fig/degree_lasvegas.png')
```

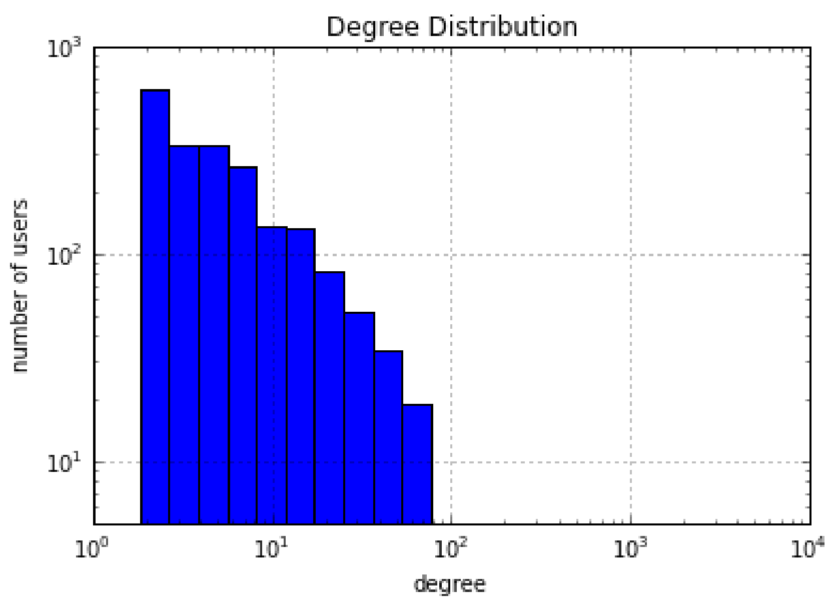


In [57]:
```
print degrees_subnet[1].degree.mean(), degrees_subnet[1].degree.mec
```

17.2295631426 4.0

Montreal

In [58]:
```
plot_degrees(degrees_subnet[3].degree, 5, 1000)
plt.savefig('../fig/degree_lasvegas.png')
```



In [59]:
```
print degrees_subnet[3].degree.mean(), degrees_subnet[3].degree.mec
```

5.41567489115 2.0

If we look at above results, three cities have quite different social networks. The avereage degrees are quite different. It is hard to find reasons why they are so different.

Correlation between values.

reviews per user and degrees per user

In [60]:
```
rev_count = user_df.user_review_count
deg_count = degrees.degree
```

In [61]:
```
from scipy.stats.stats import pearsonr
```

In [62]:
```
pearsonr(rev_count, deg_count)[0]
```

Out[62]:
```
0.48288144800008354
```

As expected, the more active users are in terms of writing reviews, the more friends those users have. We can clearly see that when we look at those values together.

In [63]:
```
zip(rev_count, deg_count)[:100]
```

Out[63]:
```
[(108, 206),
 (1233, 1904),
 (442, 354),
 (11, 4),
 (66, 4),
 (1589, 1094),
 (170, 11),
 (54, 39),
 (59, 2),
 (96, 52),
 (20, 6),
 (979, 123),
 (42, 13),
 (192, 2),
 (129, 24),
 (350, 12),
 (6, 0),
 (10, 5),
 (7, 1),
 (19, 2),
 (68, 7),
 (33, 7),
 (6, 1),
 (19, 8),
 (9, 6),
 (9, 1),
```

```
(4, 0),
(10, 0),
(11, 0),
(2, 0),
(12, 3),
(36, 1),
(3, 3),
(4, 0),
(21, 0),
(31, 2),
(90, 3),
(18, 2),
(14, 0),
(7, 3),
(33, 0),
(58, 13),
(12, 0),
(23, 1),
(2, 0),
(13, 4),
(2, 1),
(1, 0),
(56, 2),
(2, 0),
(2, 0),
(16, 11),
(1, 15),
(59, 0),
(5, 2),
(1, 4),
(2, 9),
(14, 0),
(8, 0),
(4, 0),
(3, 0),
(2, 0),
(9, 1),
(1, 1),
(12, 1),
(2, 0),
(4, 0),
(47, 0),
(13, 0),
(14, 0),
(6, 0),
(3, 0),
(1, 0),
(7, 1),
(1, 0),
(41, 1),
(4, 0),
(7, 1),
(2, 0),
```

```
 (7, 0),
 (2, 0),
 (10, 0),
 (25, 0),
 (3, 0),
 (1, 0),
 (3, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (23, 10),
 (16, 1),
 (50, 33),
 (155, 24),
 (430, 161),
 (35, 1),
 (6, 0),
 (97, 20),
 (13, 0),
 (48, 17)]
```

Finding friends of friends for a network of a given city

In [64]:
```
city = 0
my_graph = read_dictlist_from_file('../data/network' + str(city) +
```

In [65]:
```
friends_of_friends = {}
for node in my_graph:
    friends_of_friends[node] = set(my_graph[node])
    for friend1 in my_graph[node]:
        for friend2 in my_graph[friend1]:
            friends_of_friends[node].add(friend2)
```

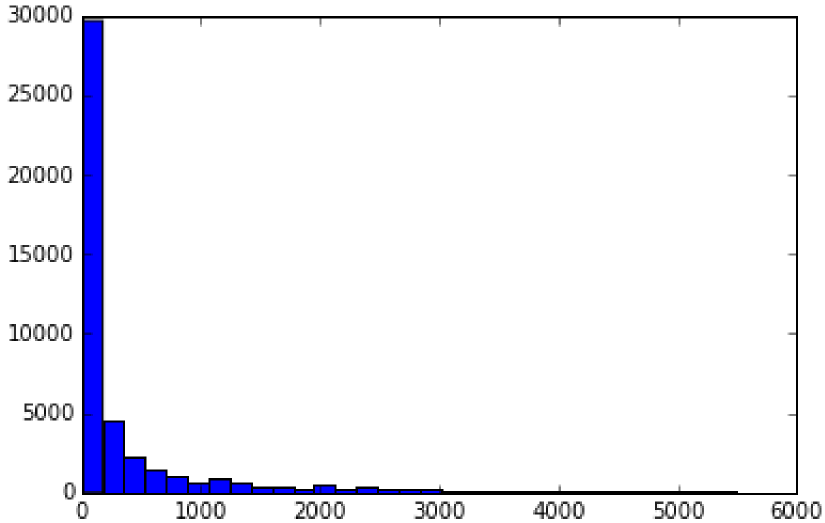
In [66]:
```
nff = []
for user in friends_of_friends:
    nff.append(len(friends_of_friends[user]))
```

In [67]:
```
plt.xlim(0,6000)
plt.hist(nff, bins=100)
```

Out[67]:
```
(array([  2.96650000e+04,   4.47800000e+03,   2.30600000e+03,
          1.49800000e+03,   9.99000000e+02,   6.46000000e+02,
          8.67000000e+02,   6.67000000e+02,   4.42000000e+02,
          3.46000000e+02,   2.69000000e+02,   4.71000000e+02,
          2.65000000e+02,   3.31000000e+02,   2.41000000e+02,
          2.27000000e+02,   2.04000000e+02,   1.62000000e+02,
          1.50000000e+02,   1.29000000e+02,   1.21000000e+02,
          1.07000000e+02,   7.80000000e+01,   9.30000000e+01,
          8.20000000e+01,   6.30000000e+01,   5.10000000e+01,
```

```
                4.10000000e+01,   5.40000000e+01,   4.40000000e+01,
                4.10000000e+01,   3.00000000e+01,   2.60000000e+01,
                3.70000000e+01,   2.20000000e+01,   2.10000000e+01,
                2.10000000e+01,   2.50000000e+01,   2.00000000e+01,
                1.60000000e+01,   1.70000000e+01,   1.10000000e+01,
                1.40000000e+01,   9.00000000e+00,   1.10000000e+01,
                1.30000000e+01,   1.10000000e+01,   6.00000000e+00,
                5.00000000e+00,   4.00000000e+00,   7.00000000e+00,
                1.00000000e+01,   4.00000000e+00,   6.00000000e+00,
                1.00000000e+00,   2.00000000e+00,   5.00000000e+00,
                5.00000000e+00,   5.00000000e+00,   4.00000000e+00,
                3.00000000e+00,   2.00000000e+00,   0.00000000e+00,
                4.00000000e+00,   2.00000000e+00,   1.00000000e+00,
                0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
                1.00000000e+00,   1.00000000e+00,   0.00000000e+00,
                1.00000000e+00,   1.00000000e+00,   1.00000000e+00,
                0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                1.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
                1.00000000e+00]),
         array([  2.00000000e+00,   1.79100000e+02,   3.56200000e+02,
                5.33300000e+02,   7.10400000e+02,   8.87500000e+02,
                1.06460000e+03,   1.24170000e+03,   1.41880000e+03,
                1.59590000e+03,   1.77300000e+03,   1.95010000e+03,
                2.12720000e+03,   2.30430000e+03,   2.48140000e+03,
                2.65850000e+03,   2.83560000e+03,   3.01270000e+03,
                3.18980000e+03,   3.36690000e+03,   3.54400000e+03,
                3.72110000e+03,   3.89820000e+03,   4.07530000e+03,
                4.25240000e+03,   4.42950000e+03,   4.60660000e+03,
                4.78370000e+03,   4.96080000e+03,   5.13790000e+03,
                5.31500000e+03,   5.49210000e+03,   5.66920000e+03,
                5.84630000e+03,   6.02340000e+03,   6.20050000e+03,
                6.37760000e+03,   6.55470000e+03,   6.73180000e+03,
                6.90890000e+03,   7.08600000e+03,   7.26310000e+03,
                7.44020000e+03,   7.61730000e+03,   7.79440000e+03,
                7.97150000e+03,   8.14860000e+03,   8.32570000e+03,
                8.50280000e+03,   8.67990000e+03,   8.85700000e+03,
                9.03410000e+03,   9.21120000e+03,   9.38830000e+03,
                9.56540000e+03,   9.74250000e+03,   9.91960000e+03,
                1.00967000e+04,   1.02738000e+04,   1.04509000e+04,
                1.06280000e+04,   1.08051000e+04,   1.09822000e+04,
                1.11593000e+04,   1.13364000e+04,   1.15135000e+04,
                1.16906000e+04,   1.18677000e+04,   1.20448000e+04,
                1.22219000e+04,   1.23990000e+04,   1.25761000e+04,
                1.27532000e+04,   1.29303000e+04,   1.31074000e+04,
                1.32845000e+04,   1.34616000e+04,   1.36387000e+04,
                1.38158000e+04,   1.39929000e+04,   1.41700000e+04,
                1.43471000e+04,   1.45242000e+04,   1.47013000e+04,
```

```
          1.48784000e+04,   1.50555000e+04,   1.52326000e+04,
          1.54097000e+04,   1.55868000e+04,   1.57639000e+04,
          1.59410000e+04,   1.61181000e+04,   1.62952000e+04,
          1.64723000e+04,   1.66494000e+04,   1.68265000e+04,
          1.70036000e+04,   1.71807000e+04,   1.73578000e+04,
          1.75349000e+04,   1.77120000e+04]),
 <a list of 100 Patch objects>)
```



Looking at all small networks (biggest components).

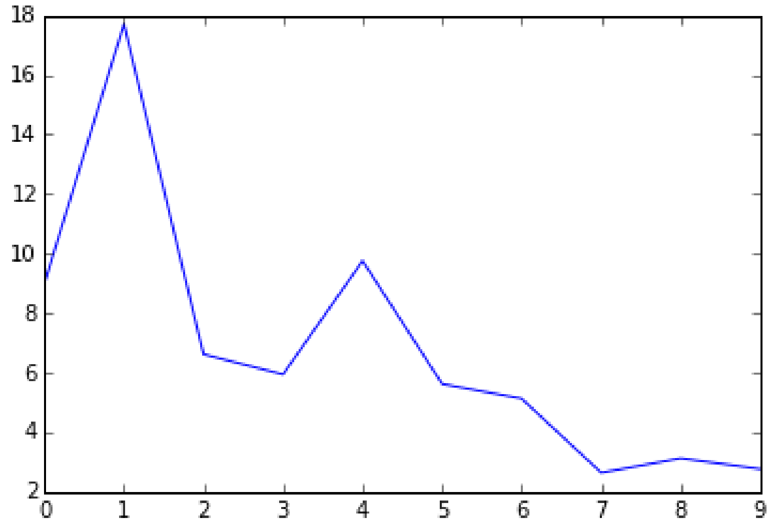
In [68]:

```
# Reading the biggest component of each graph for each city.
my_nets = []  # dict of lists
my_nets_nx = []  # networkx objects
for i in range(10):
    my_nets.append(reindex_graph(read_dictlist_from_file('../data/r
    my_nets_nx.append(convert_to_nx(my_nets[-1]))
```

Number of average degrees.

In [69]:
```
average_degrees = []
for i in range(10):
    average_degrees.append(np.mean(my_nets_nx[i].degree().values())
plt.plot(average_degrees)
```

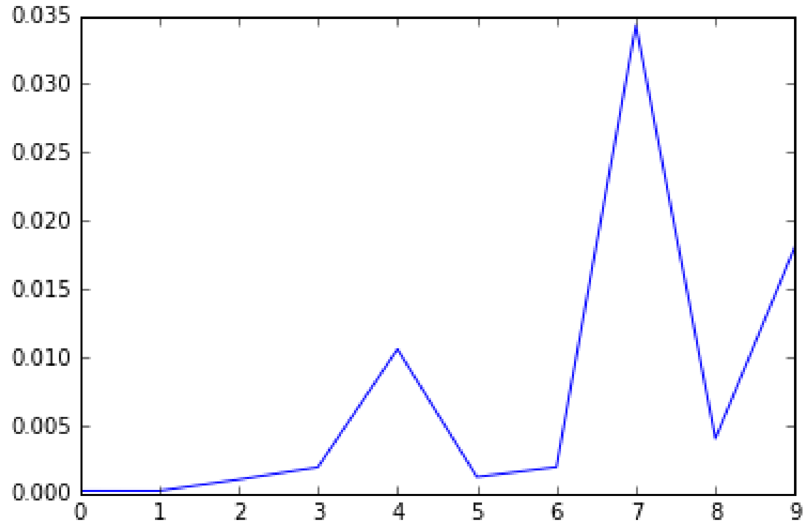
Out[69]: [<matplotlib.lines.Line2D at 0x10da93250>]



Densities

In [70]:
```
densities = []
for i in range(10):
    densities.append(nx.density(my_nets_nx[i]))
plt.plot(densities)
```

Out[70]: [<matplotlib.lines.Line2D at 0x1aa24f310>]



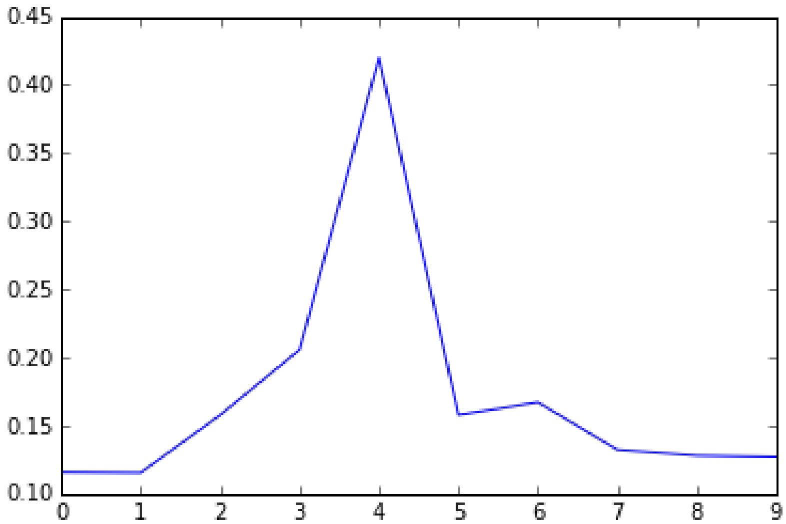
Assortativities.

In [ ]:
```
assortativities = []
for i in range(10):
    assortativities.append(nx.degree_assortativity_coefficient(my_r
plt.plot(assortativities)
```

Average clustering coefficients

In [71]:
```
average_clusterings = []
for i in range(10):
    average_clusterings.append(nx.average_clustering(my_nets_nx[i])
plt.plot(average_clusterings)
```

Out[71]:
```
[<matplotlib.lines.Line2D at 0x1a9e4a2d0>]
```



Spectral clustering for cities.

In [ ]:
```
# Adjacency matrix for small cities (city number >= 2 only)
adjacency = []
for i in range(2, 10):
    adjacency.append(adjacency_matrix(my_nets[i]))
```

In [ ]:
```
spectral = []
for i in range(8):
    spectral.append(SpectralClustering(n_clusters=2, eigen_solver='
                                       assign_labels='kmeans'))
```

In [ ]:
```
for i in range(8):
    spectral[i].fit(adjacency[i])
```

In [ ]:
```
n_communities = []
for i in range(5):
    n_communities.append((spectral[0].labels_ == i).sum())
plt.plot(n_communities)
```

In [ ]:
```
# Error distribution for matrix factorization case
errors = pd.read_csv('tt', header=None, names=['e'])
errors.e.hist(bins=30)
```

Differences between real rating and predictions.

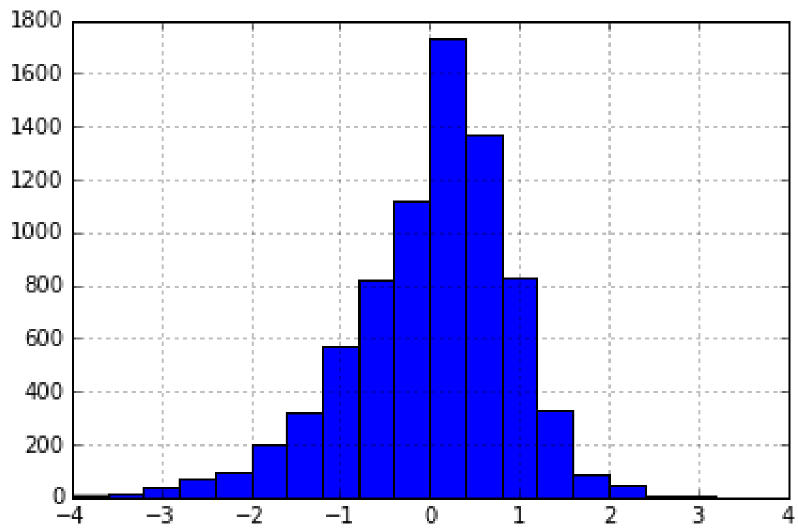
In [72]:
```
# Read the data for one run.
rating_diff_df = pd.read_csv('../data/ratings_by_item2', delimiter=
```

In [73]:
```
ratings_diff_nonull = rating_diff_df[['real', 'average', 'friend',
```

In [74]:
```
diff_ave = ratings_diff_nonull.real - ratings_diff_nonull.average
```

In [75]:
```
diff_ave.hist(bins=20)
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9e61f50>



In [76]:
```
diff_ave.std()
```

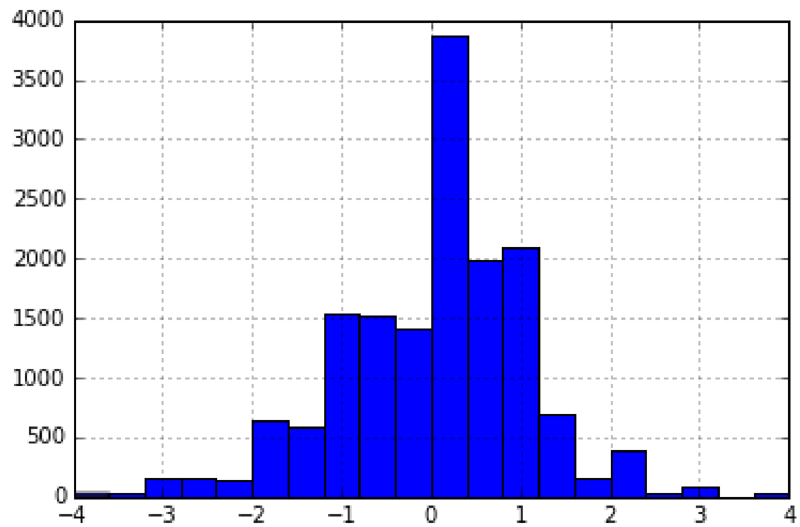
Out[76]: 0.91056367396673188

In [77]:
```
stds = []
for n in range(1,100):
    diff_fr = ratings_diff_nonull[ratings_diff_nonull['nfr'] == n].
              ratings_diff_nonull[ratings_diff_nonull['nfr'] == n]
    stds.append(np.sqrt(np.mean(np.square(diff_fr))))
```

In [79]:
```
diff_fr2 = rating_diff_df.real - rating_diff_df.friend2
```

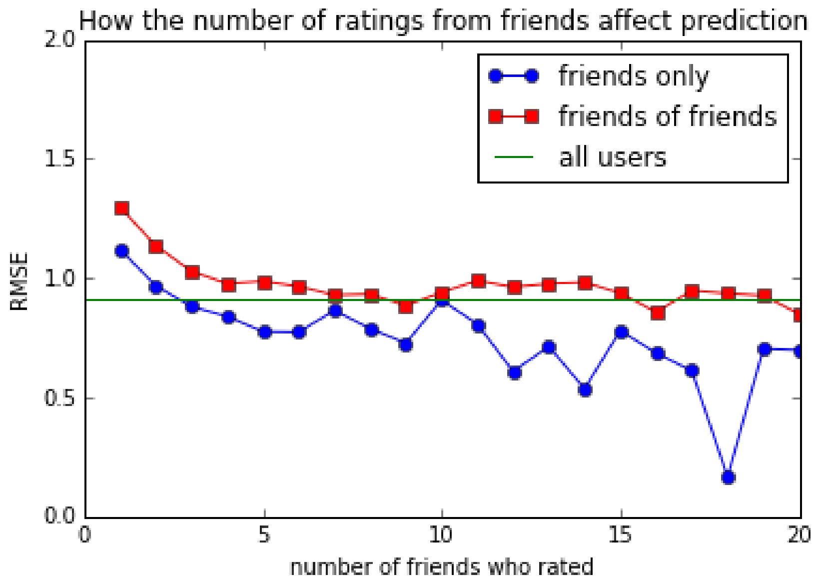
In [80]:
```
diff_fr2.hist(bins=20)
```

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9e19a90>



In [81]:
```
stds2 = []
for n in range(1,100):
    diff_fr = rating_diff_df[rating_diff_df['nfr2'] == n].real - \
              rating_diff_df[rating_diff_df['nfr2'] == n].friend2
    stds2.append(np.sqrt(np.mean(np.square(diff_fr))))
```

In [150]:
```
plt.xlabel('number of friends who rated')
plt.ylabel('RMSE')
plt.xlim(0,20)
plt.ylim(0,2)
plt.title('How the number of ratings from friends affect predictior
fig1 = plt.plot(range(1,100), stds, marker='o')
fig2 = plt.plot(range(1,100), stds2, marker='s', color='r')
b = np.sqrt(np.mean(np.square(diff_ave)))
fig3 = plt.plot([0, 100], [b, b], color='g')
plt.legend((fig1[0], fig2[0], fig3[0]), ('friends only', 'friends c
plt.savefig('../fig/friends.png', dpi=200)
```



In [129]:
```
# Baseline is already obtained for the whole ratings of the dataset
# of the all ratings.
b = 1.3114679041155461
```

Baseline numbers for each city was obtained using Validator.get_baseline method. Here it computes the standard deviation for the given ratings of the training set.

In [131]:
```
ba_df = np.loadtxt('../data/baseline_each_city')
```

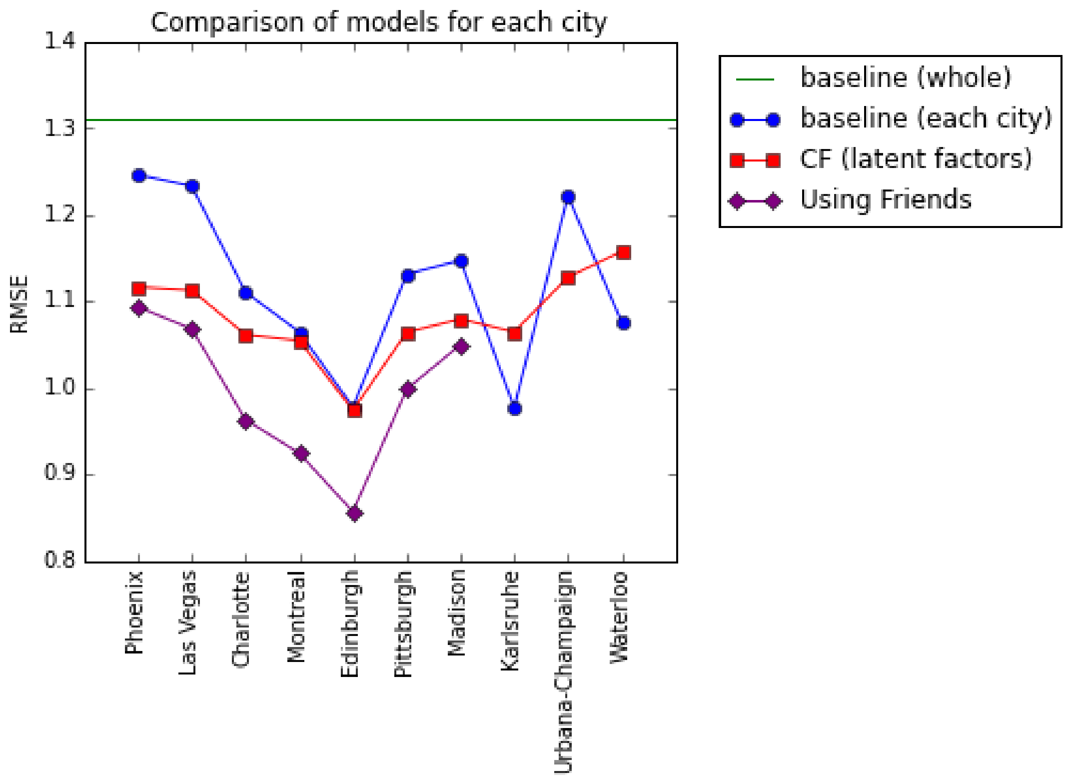
CF was run using (n_feasures=2, learning_rate=0.009, regularization_parameter=0.07) and K=10 with item bias considered (not user bias).

In [132]:
```
cf = np.loadtxt('../data/cf_rmse')
```

Using_Friends has been used to find the RMSE for the network-based model.

In [100]:
```
uf_df = np.loadtxt('../data/prel_results')#, delimiter=' ', names=[
```

In [148]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
plt.title('Comparison of models for each city')
plt.ylabel('RMSE')
plt.xlim(-1,10)
plt.ylim(0.8, 1.4)
ax.set_xticks(range(10))
xtickNames = ax.set_xticklabels(city_names)
plt.setp(xtickNames, rotation=90, fontsize=10)
ax.set_position([0.1,0.1,1.2,0.9])
fig1 = plt.plot([-2, 100], [b, b], color='g', label='baseline (whol
fig2 = plt.plot(ba_df[:,0], ba_df[:,1], marker='o', color='b', labe
fig3 = plt.plot(cf, marker='s', color='r', label='CF (latent factor
fig4 = plt.plot(uf_df[:,0], uf_df[:,1], marker='D', color='purple',
plt.legend(bbox_to_anchor=(1.05, 1), loc=2 )
plt.tight_layout()
plt.savefig('../fig/rmse.png', dpi=200)
```



In [151]:
```
cf
```

Out[151]:
```
array([ 1.1164795 ,  1.11277415,  1.06140494,  1.05461539,  0.9738
4244,
        1.06415573,  1.07947529,  1.06458209,  1.12783704,  1.1577
5142])
```

In [ ]: