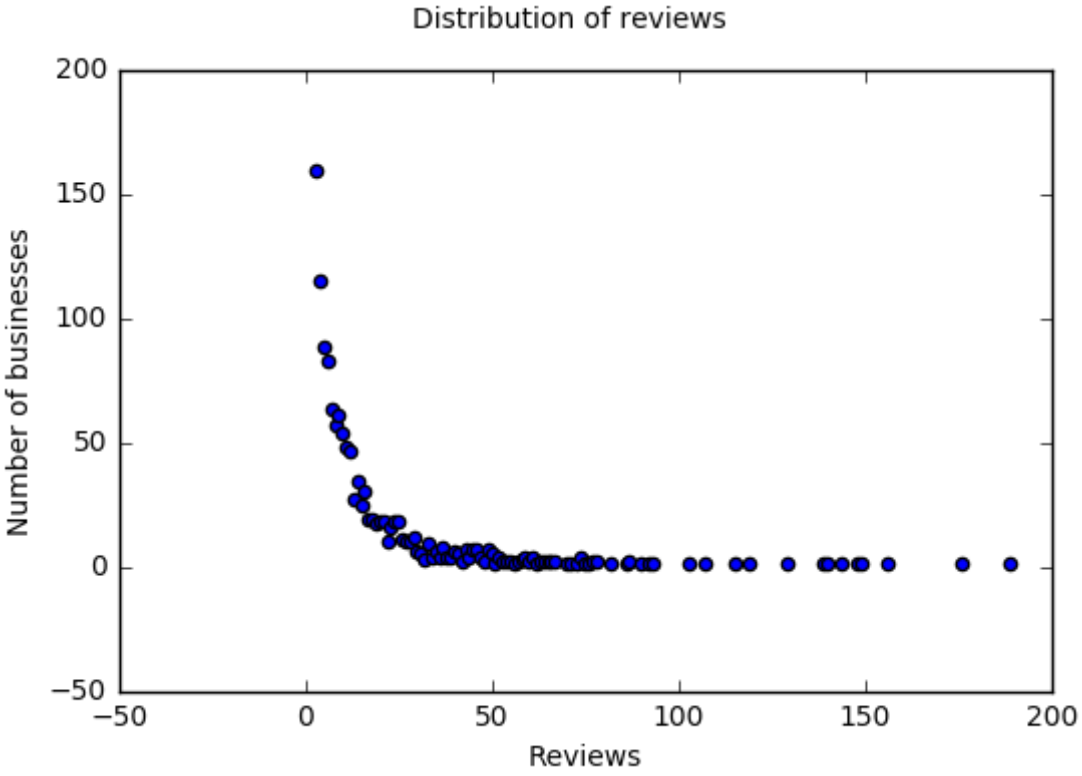
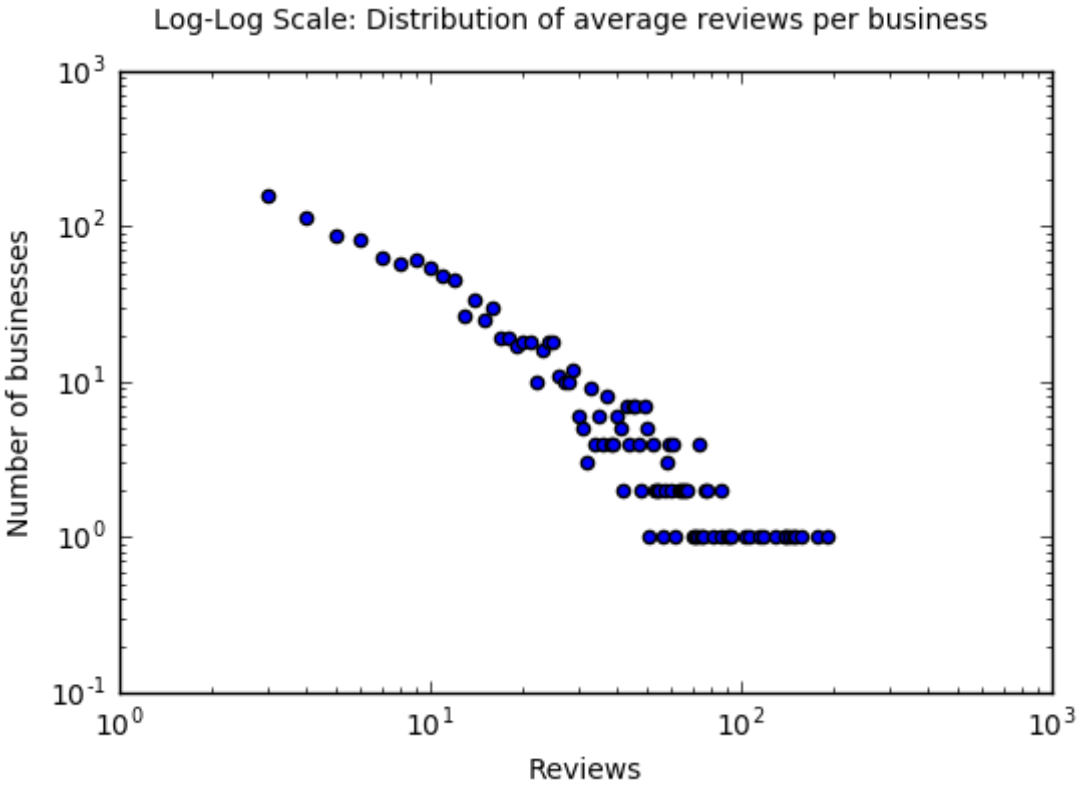
In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [2]:
```
import parser
```

In [3]:
```
businesses = parser.getBusinesses()
review_counts = []
for business in businesses:
    review_counts.append(business.getReviewCount())

nonzero_review_counts = [n for n in review_counts if n > 0]
```

In [4]:
```
from collections import Counter

frequency = dict(Counter(nonzero_review_counts))

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Distribution of reviews')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Reviews")
ply.ylabel("Number of businesses")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Log-Log Scale: Distribution of average reviews per busi
ness')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Reviews")
ply.ylabel("Number of businesses")
ax.set_yscale('log')
ax.set_xscale('log')
```


Distribution of reviews
Number of businesses
Reviews
200
150
100
50
0
−50
−50
0
50
100
150
200


Log-Log Scale: Distribution of average reviews per business
Number of businesses
Reviews
10^3
10^2
10^1
10^0
10^{-1}
10^0
10^1
10^2
10^3

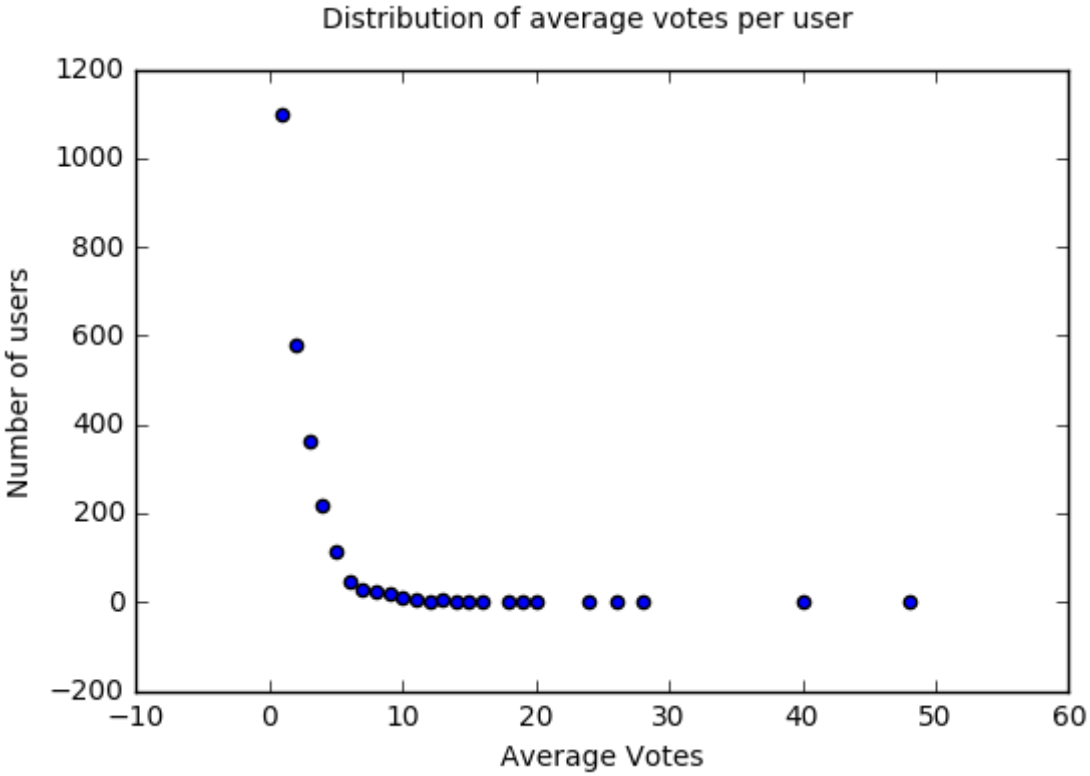
In [ ]:

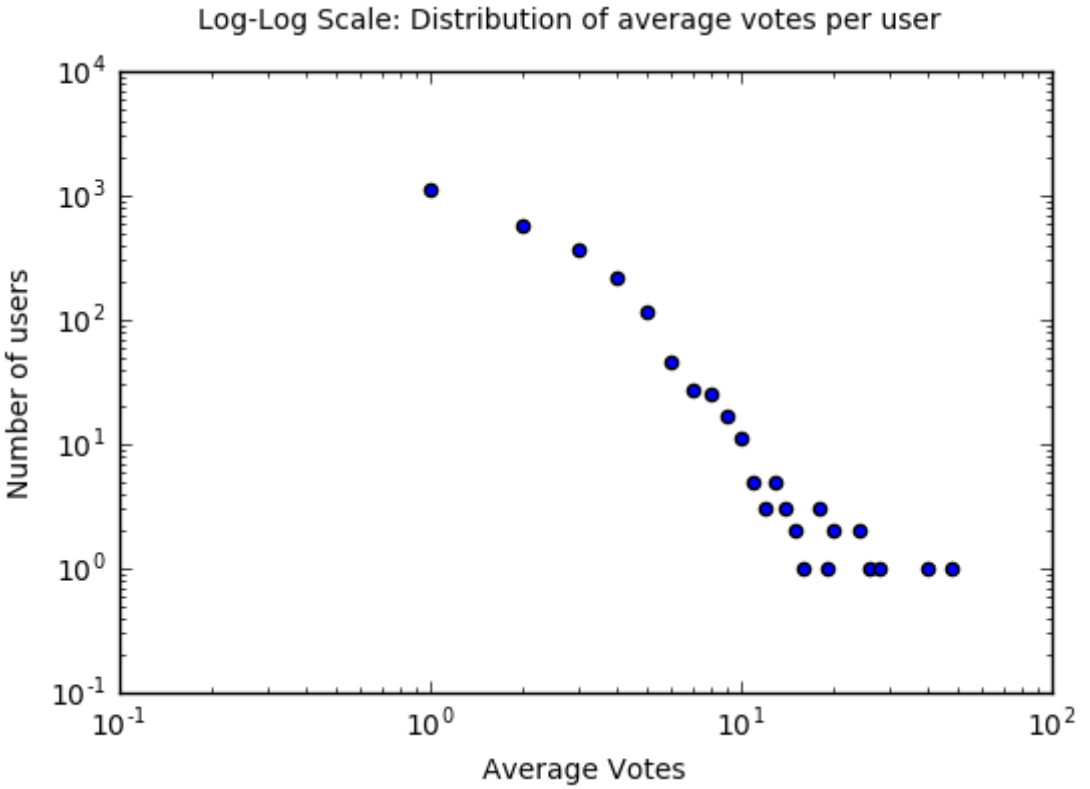
In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [2]:
```
import parser
```

In [3]:
```
users = parser.getUsers()
avg_vote_counts = []
for user in users:
        avg_vote_counts.append(user.getAverageVotes())

nonzero_avg_vote_counts = [n for n in avg_vote_counts if n > 0]
```

In [4]:
```
from collections import Counter

frequency = dict(Counter(nonzero_avg_vote_counts))

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Distribution of average votes per user')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Average Votes")
ply.ylabel("Number of users")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Log-Log Scale: Distribution of average votes per user')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Average Votes")
ply.ylabel("Number of users")
ax.set_yscale('log')
ax.set_xscale('log')
```


Distribution of average votes per user
Number of users
Average Votes
1200
1000
800
600
400
200
0
−200
−10
0
10
20
30
40
50
60


Log-Log Scale: Distribution of average votes per user
Number of users
Average Votes
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-1}
10^0
10^1
10^2

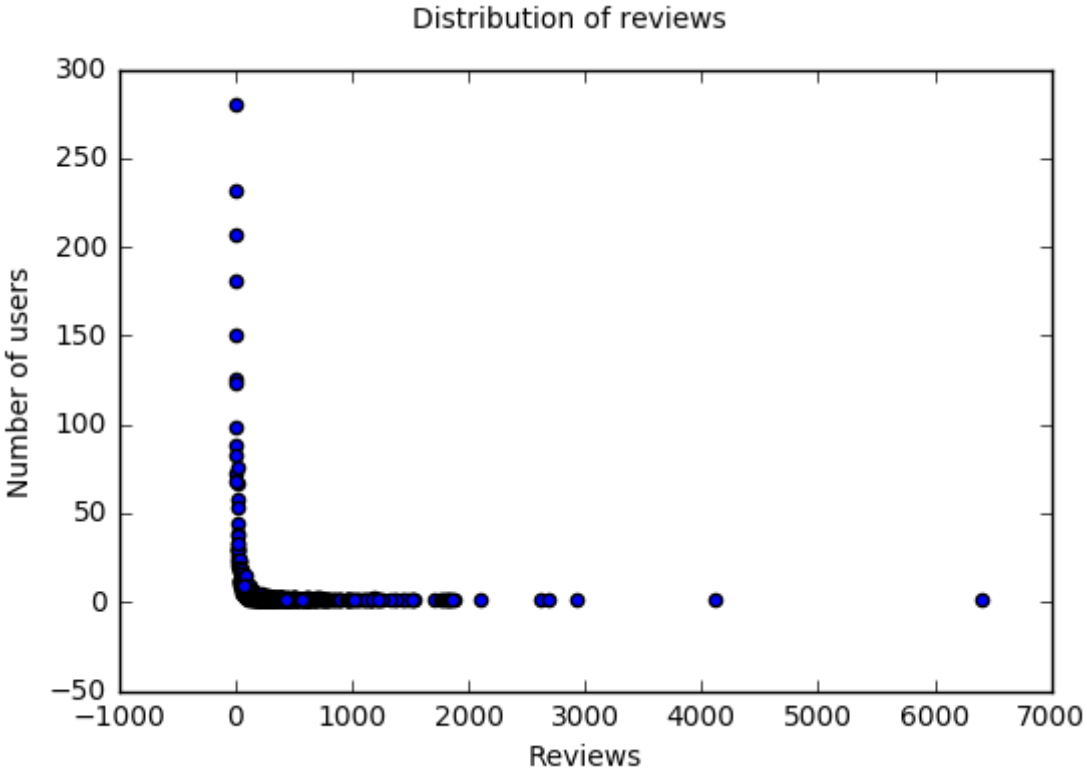
In [ ]:

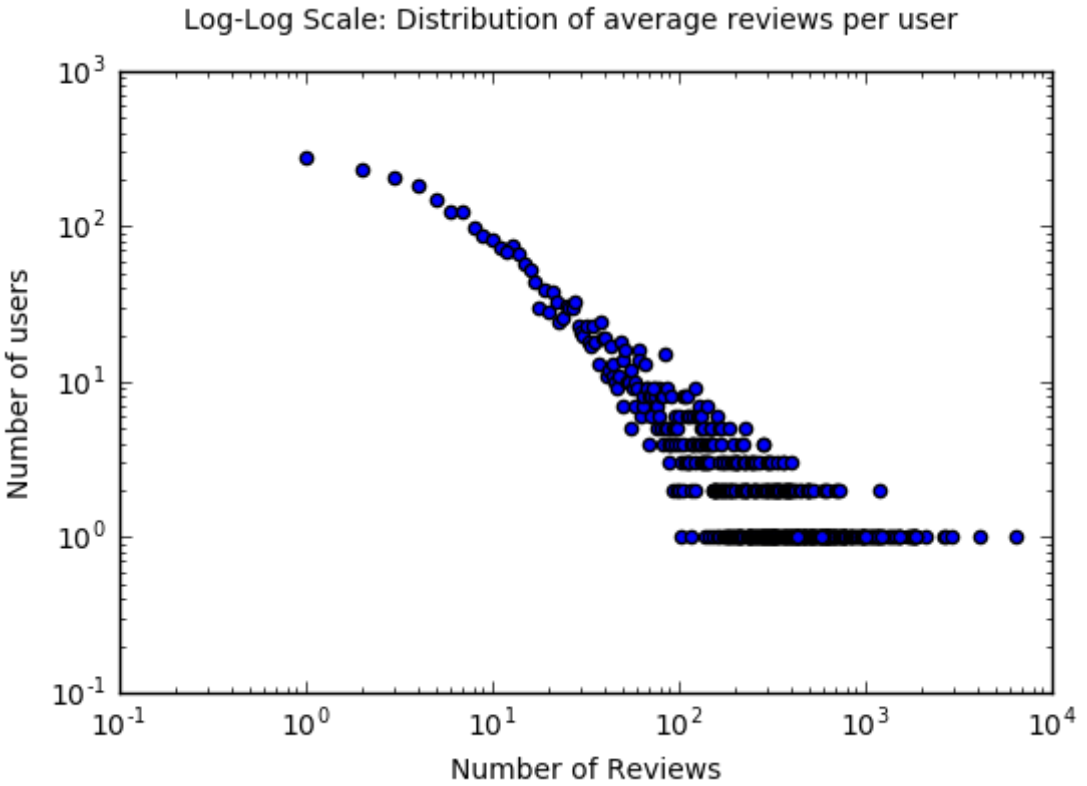
In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [2]:
```
import parser
```

In [3]:
```
users = parser.getUsers()
review_counts = []
for user in users:
    review_counts.append(user.getReviewCount())

nonzero_review_counts = [n for n in review_counts if n > 0]
```

In [4]:
```
from collections import Counter

frequency = dict(Counter(nonzero_review_counts))

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Distribution of reviews')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Reviews")
ply.ylabel("Number of users")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Log-Log Scale: Distribution of average reviews per use
r')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Number of Reviews")
ply.ylabel("Number of users")
ax.set_yscale('log')
ax.set_xscale('log')
```


Distribution of reviews
Number of users
Reviews
300
250
200
150
100
50
0
−50
−1000
0
1000
2000
3000
4000
5000
6000
7000


Log-Log Scale: Distribution of average reviews per user
Number of users
Number of Reviews
10^3
10^2
10^1
10^0
10^{-1}
10^{-1}
10^0
10^1
10^2
10^3
10^4

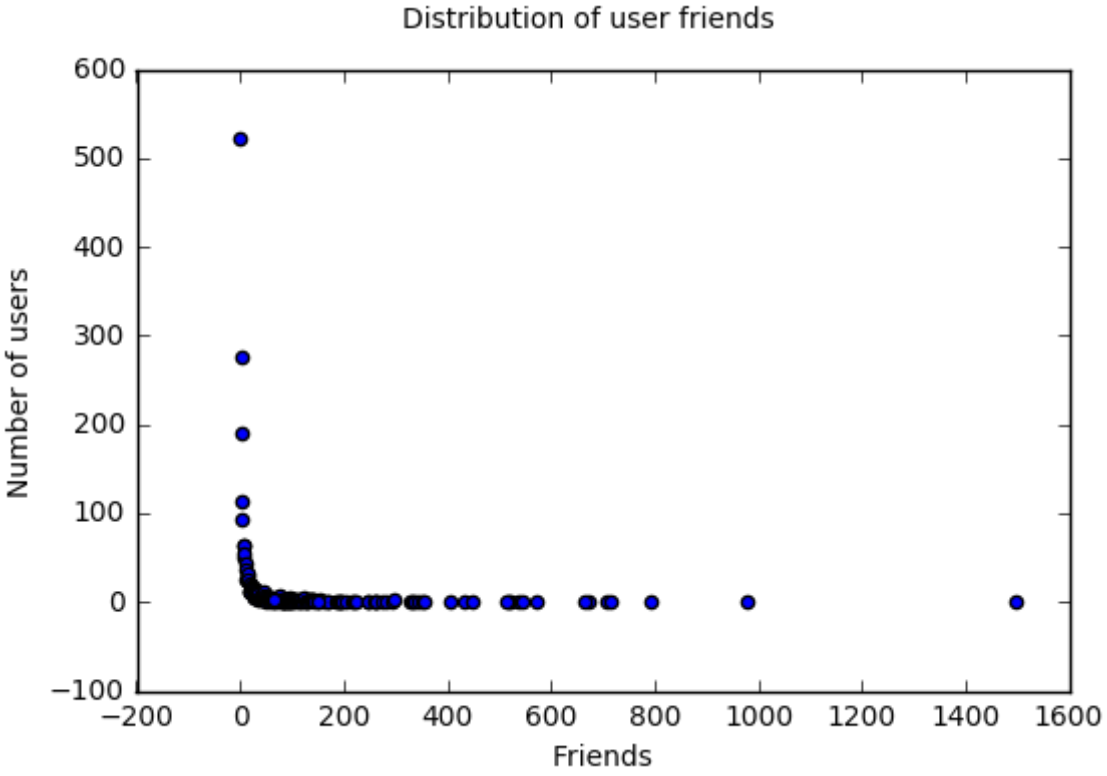
In [ ]:

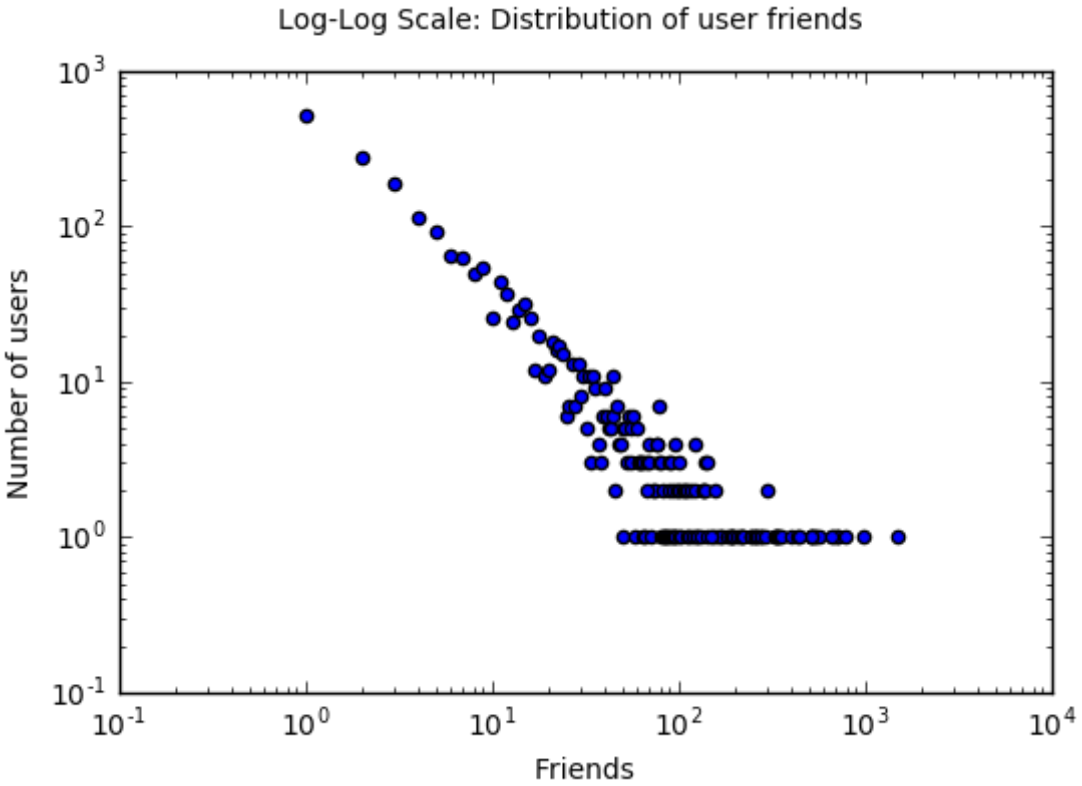
In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [2]:
```
import parser
```

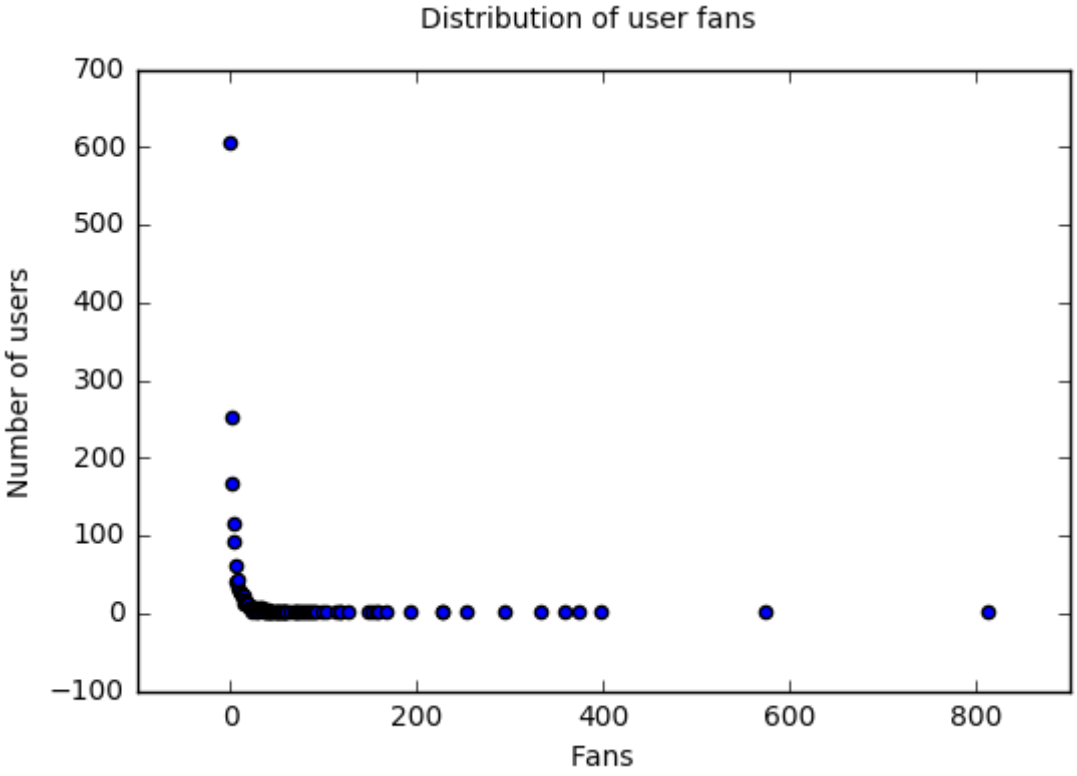
In [3]:
```
users = parser.getUsers()
friend_counts = []
for user in users:
        friend_counts.append(user.getFriendCount())

nonzero_friend_counts = [n for n in friend_counts if n > 0]

fan_counts = []
yelping_since_fans = {}
elite_for_fans = {}
for user in users:
        fan_counts.append(user.getFanCount())
        yelping_since = user.getYelpingSince()
        elite_for = user.getEliteNum()
        if yelping_since in yelping_since_fans.keys():

yelping_since_fans[yelping_since].append(user.getFanCount())
        else:
            yelping_since_fans[yelping_since] = [user.getFanCount()]

        if elite_for in elite_for_fans.keys():
            elite_for_fans[elite_for].append(user.getFanCount())
        else:
            elite_for_fans[elite_for] = [user.getFanCount()]

nonzero_fan_counts = [n for n in fan_counts if n > 0]
```

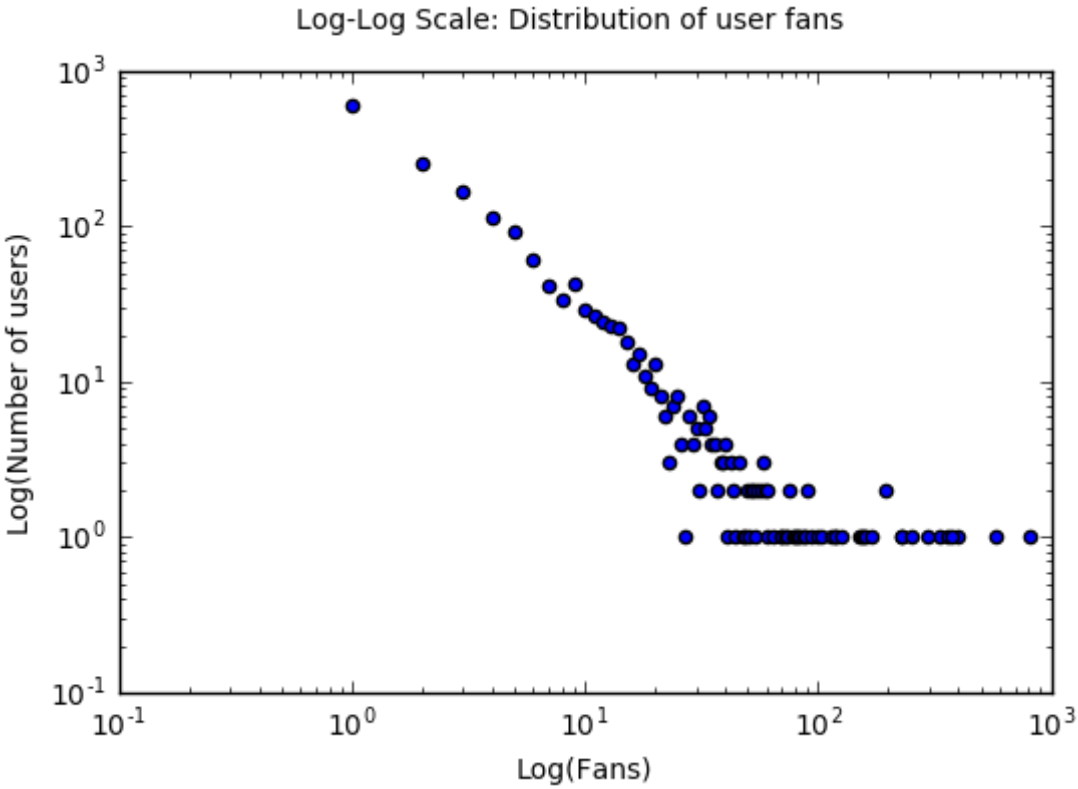
In [4]:
```
from collections import Counter

frequency = dict(Counter(nonzero_friend_counts))

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Distribution of user friends')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Friends")
ply.ylabel("Number of users")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Log-Log Scale: Distribution of user friends')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Friends")
ply.ylabel("Number of users")
ax.set_yscale('log')
ax.set_xscale('log')
```


Distribution of user friends
Number of users
Friends
600
500
400
300
200
100
0
−100
−200
0
200
400
600
800
1000
1200
1400
1600


Log-Log Scale: Distribution of user friends
Number of users
Friends
10^3
10^2
10^1
10^0
10^{-1}
10^{-1}
10^0
10^1
10^2
10^3
10^4

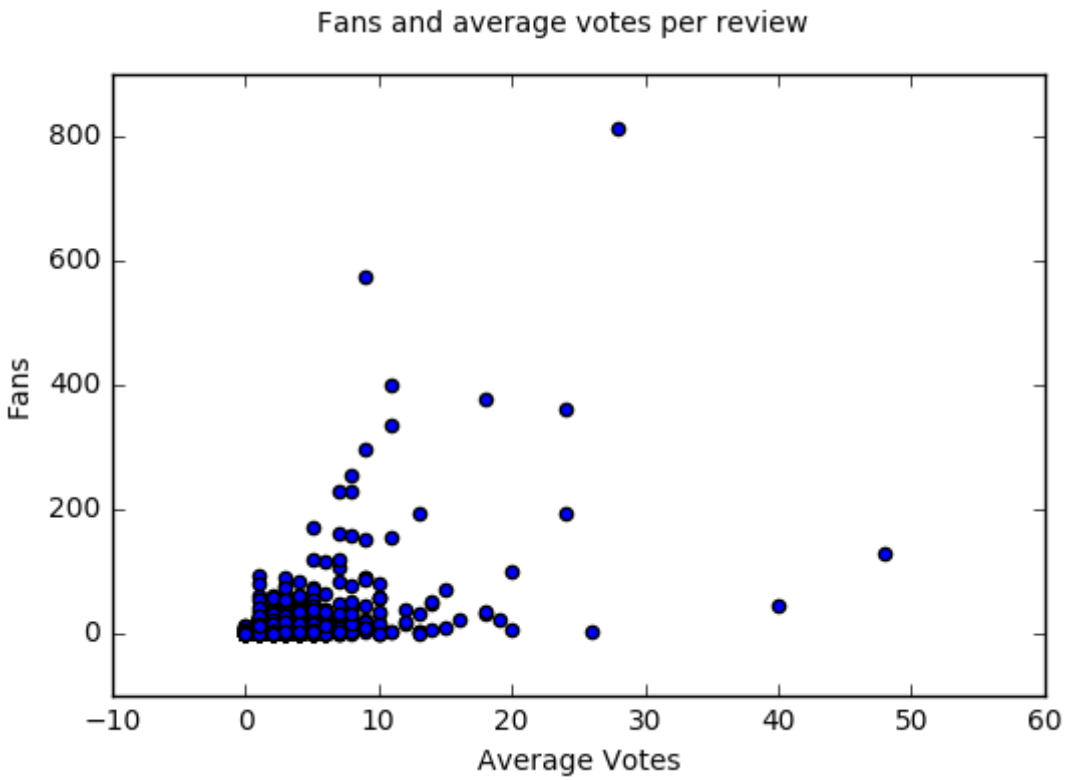
In [5]:
```
from collections import Counter

frequency = dict(Counter(nonzero_fan_counts))

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Distribution of user fans')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Fans")
ply.ylabel("Number of users")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Log-Log Scale: Distribution of user fans')
ax.scatter(frequency.keys(), [frequency[n] for n in
frequency.keys()])
ply.xlabel("Log(Fans)")
ply.ylabel("Log(Number of users)")
ax.set_yscale('log')
ax.set_xscale('log')
```

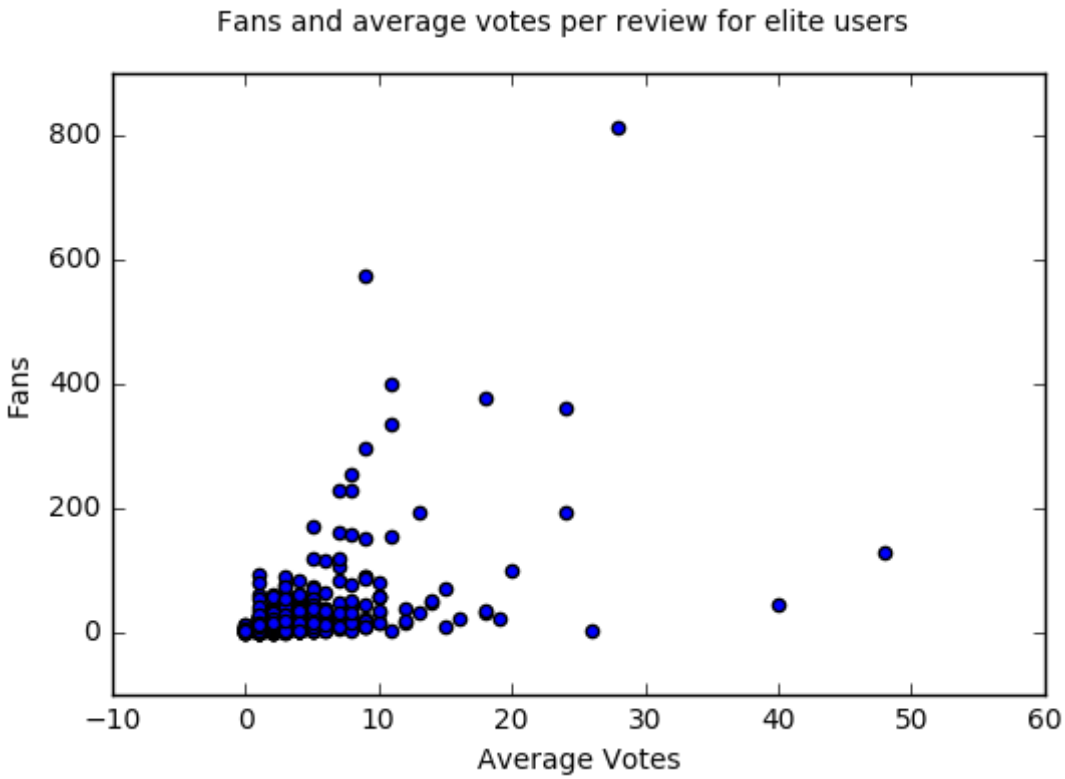

Distribution of user fans
Number of users
Fans
700
600
500
400
300
200
100
0
−100
0
200
400
600
800


Log-Log Scale: Distribution of user fans
Log(Number of users)
Log(Fans)
10^3
10^2
10^1
10^0
10^{-1}
10^{-1}
10^0
10^1
10^2
10^3

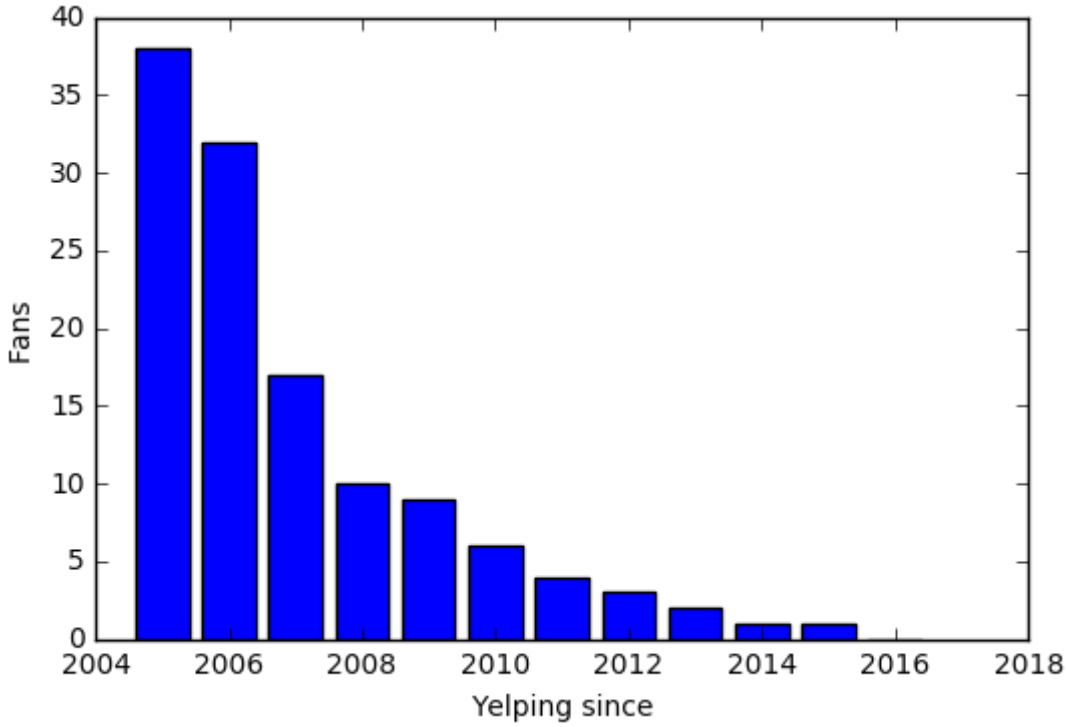
In [24]:
```
fan_counts = []
vote_counts = []

elite_fan_counts = []
elite_vote_counts = []

for user in users:
    if user.getEliteNum() > 0:
        elite_fan_counts.append(user.getFanCount())
        elite_vote_counts.append(user.getAverageVotes())

for user in users:
        fan_counts.append(user.getFanCount())
        vote_counts.append(user.getAverageVotes())
```

In [25]:
```
fig = ply.figure()
ax = ply.gca()
fig.suptitle('Fans and average votes per review')
ax.scatter(vote_counts, fan_counts)
ply.xlabel("Average Votes")
ply.ylabel("Fans")
ply.show()

fig = ply.figure()
ax = ply.gca()
fig.suptitle('Fans and average votes per review for elite users')
ax.scatter(elite_vote_counts, elite_fan_counts)
ply.xlabel("Average Votes")
ply.ylabel("Fans")
ply.show()
```

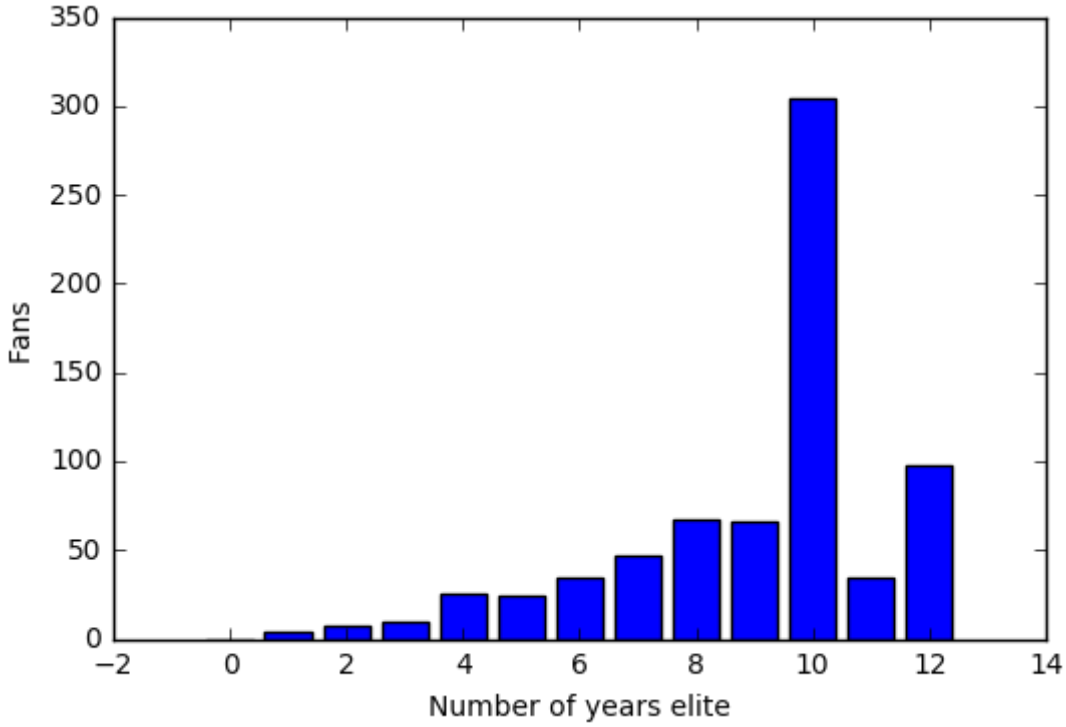

Fans and average votes per review
Fans
Average Votes
800
600
400
200
0
−10
0
10
20
30
40
50
60


Fans and average votes per review for elite users
Fans
Average Votes
800
600
400
200
0
−10
0
10
20
30
40
50
60

In [14]:
```
yelping_since = []
fan_counts = []

for year in yelping_since_fans:
    yelping_since.append(year)
    fan_counts.append(sum(yelping_since_fans[year])/len(yelping_since
_fans[year]))

ply.bar(yelping_since, fan_counts, align='center')
ply.xlabel("Yelping since")
ply.ylabel("Fans")
ply.show()
```

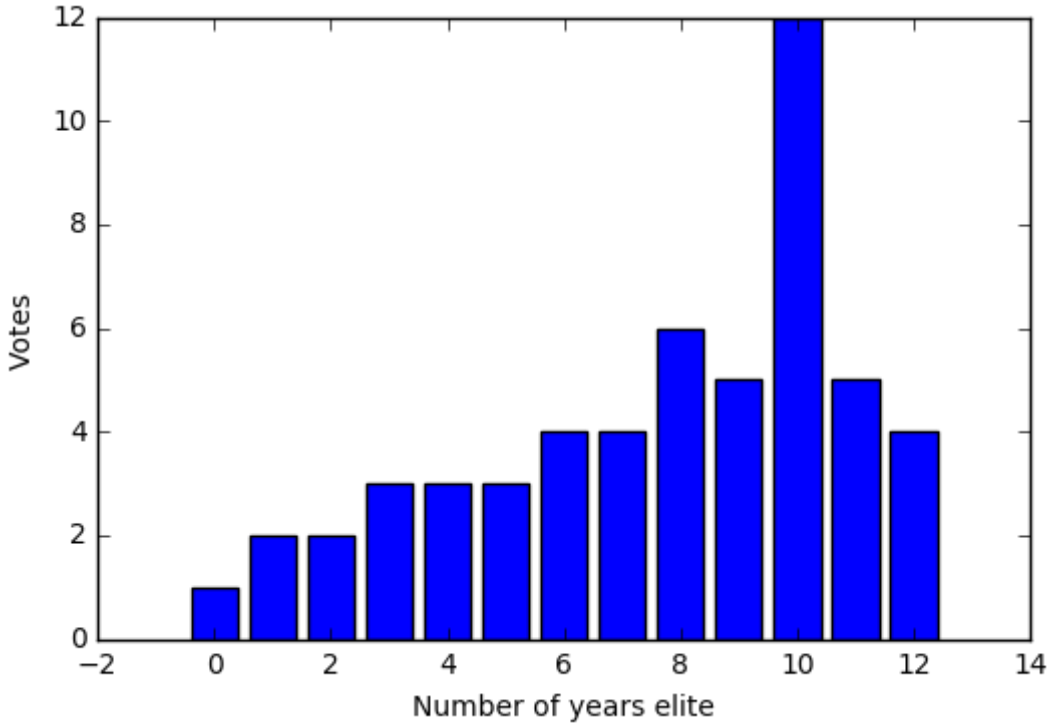


In [15]:
```
elite_for = []
fan_counts = []

for years in elite_for_fans:
    elite_for.append(years)
    fan_counts.append(sum(elite_for_fans[years])/len(elite_for_fans[y
ears]))

ply.bar(elite_for, fan_counts, align='center')
ply.xlabel("Number of years elite")
ply.ylabel("Fans")
ply.show()
```



In [16]:
```
elite_for_votes = {}
for user in users:
        if user.getEliteNum() in elite_for_votes.keys():
            elite_for_votes[user.getEliteNum()].append(user.getAverag
eVotes())
        else:
            elite_for_votes[user.getEliteNum()] = [user.getAverageVot
es()]
```

In [12]:
```
elite_for = []
vote_counts = []

for years in elite_for_fans:
    elite_for.append(years)
    vote_counts.append(sum(elite_for_votes[years])/len(elite_for_vote
s[years]))

ply.bar(elite_for, vote_counts, align='center')
ply.xlabel("Number of years elite")
ply.ylabel("Votes")
ply.show()
```



In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

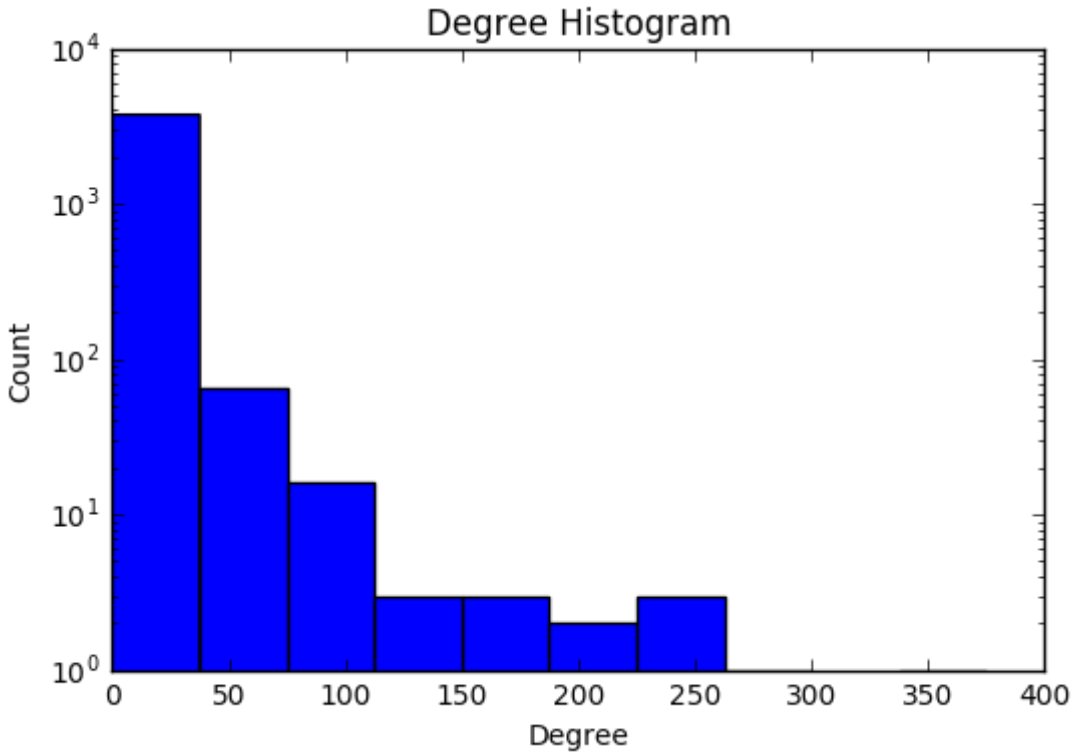
In [2]:
```
import parser
```

In [3]:
```
users = parser.getUsers()
```

In [4]:
```
g = nx.Graph()
nodes = []

for user in users:
    g.add_node(user)

nodes1 = g.nodes()
nodes2 = g.nodes()

for node1 in nodes1:
    for node2 in nodes2:
        if node1.getID() != node2.getID():
            if node1.isFriend(node2.getID()):
                g.add_edge(node1, node2)
    nodes2.remove(node1)

p = float(2*len(g.edges()))/(len(g.nodes()) * (len(g.nodes()) - 1))
```

In [5]:
```
nx.write_gexf(g, "friendship.gexf")
degree_sequence = []

for degree in g.degree().values():
    degree_sequence.append(degree)
```

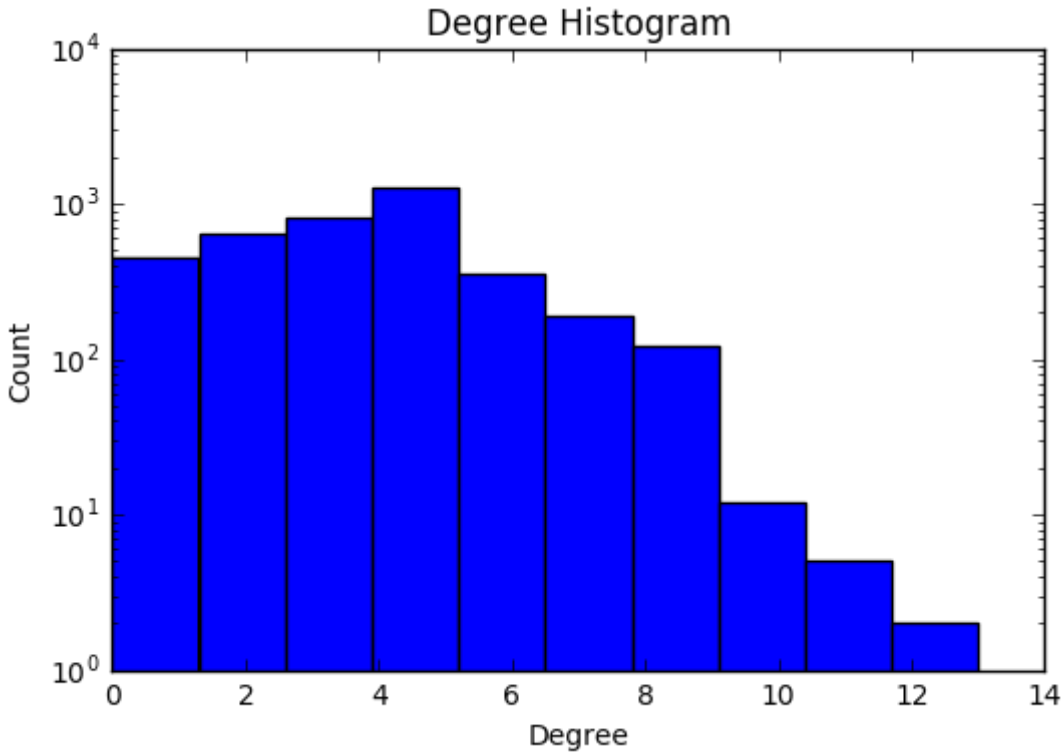
In [6]:
```
ply.hist(degree_sequence, log=True)
ply.title("Degree Histogram")
ply.ylabel("Count")
ply.xlabel("Degree")
ply.show()
```



In [7]:
```
g = nx.fast_gnp_random_graph(len(users), p)
```

In [8]:
```
nx.write_gexf(g, "friendship-random.gexf")
degree_sequence = []

for degree in g.degree().values():
    degree_sequence.append(degree)
```

In [9]:
```
ply.hist(degree_sequence, log=True)
ply.title("Degree Histogram")
ply.ylabel("Count")
ply.xlabel("Degree")
ply.show()
```



In [ ]:

In [2]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [3]:
```
from extract_user_reviews import *
```

In [4]:
```
import parser
```

In [5]:
```
business_users = {}
for user_id in user_review_business.keys():
    if user_review_business[user_id] in business_users.keys():
        business_users[user_review_business[user_id]].append(user_id)
    else:
        business_users[user_review_business[user_id]] = [user_id]
```

In [12]:
```
from textblob import TextBlob

g = nx.Graph()
users = parser.getUsers()
for user in users:
    g.add_node(user,
               friendCount = user.getFriendCount(),
               yelpingSince = user.getYelpingSince())

same_polarity = 0
opposite_polarity = 0
nodes1 = g.nodes()
nodes2 = g.nodes()
for node1 in nodes1:
    for node2 in nodes2:
        if node1.getID() != node2.getID():
            if node1.isFriend(node2.getID()):
                for business_id in business_users.keys():
                    correview_users = business_users[business_id]
                    if node1.getID() in correview_users:
                        if node2.getID() in correview_users:
                            g.add_edge(node1, node2)
                            polarity = []
                            for review1 in
user_reviews[node1.getID()]:
                                if review1["business_id"] == business
_id:
                                    b = TextBlob(review1["text"])
                                    polarity.append(b.sentiment.polar
ity)
                                    break
                            for review2 in
user_reviews[node2.getID()]:
                                if review2["business_id"] == business
_id:
                                    b = TextBlob(review2["text"])
                                    polarity.append(b.sentiment.polar
ity)
                                    break
                            if (polarity[0] >= 0 and polarity[1] >=
0) or (polarity[1] < 0 and polarity[1] < 0):
                                same_polarity += 1
                            else:
                                opposite_polarity += 1
    nodes2.remove(node1)

print same_polarity
print opposite_polarity
```

-0.0764814814815
0.151666666667
We are on Holiday from the United States, and Hellers Full Breakfast has been the our first truly regrettable breakfast. We found this place a short walk from our lodgings on Yelp, under Gluten-free Restaurant, it is not a remotely Gluten-free location. After asking about GF options we were greated with the standard song and dance, we deal with this all the time. Only one person in our group is Gluten-intolerant but everything we ordered was terible.
Fried and poached eggs so overcooked they would bounce, burned toast, underdone potatoes and bacon, tasteless sausage, cold beans, oily mushrooms, and the black pudding tasted as if it was made with sawdust. My tea was weak enough to be indistinguishable from dirty water.
I know that some will discount this because we are American, but we have loved breakfast here just this place was a let down.

Went here based on a Yelp review that they were very good at accommodating allergies. Not the case. Besides that, my bacon and potato were undercooked, I ate one bite and did not think it prudent to eat any more. My poached egg was so over cooked that I did not recognize it as an egg at first. My first thought was why is there a rubber ball on my plate? I'm from America, the home of disappointing breakfasts, I come in with the lowest of expectations, and this was worse than the cafe at the bus station.
-0.0138888888889
0.268422619048
HAGGIS HOTDOG! Decided on The Huxley after the place we attempted to initially find was closed on Sundays. Sad times and we were hungry!

But The Huxley was open Hurrah! As I've capitalised: HAGGIS HOTDOG! My first taste of haggis, it's not that bad, it actually tasted better than the BBQ dog, which was too sweet in my opinion. Also, I think they might have taken on the comments from other reviewers, because the HH came in a baguette and that was lovely.

I'm thinking I should've got something beef though, because of other reviews and also there's a cow model next to the entrance and one coming out of the balcony above it?

This place has changed again, seriously? Being a frequent(ish) visitor to Edinburgh I find that this place is different every time I'm in town. I believe it has changed hands more times a parcel during a game of pass the parcel played by a group of hyperactive kids.

The Huxley is a fantastic looking place however, it's obvious that the owners have splashed out on some very tasteful furniture and decor. The staff were also very friendly, the bar well stocked and there was a nice selections of draught beers on offer. I also enjoyed the music, not too loud and added nicely to the vibe of the place.

Prices however are pretty steep, something which I'm sure the owners realise. You can tell they are after a certain type of clientèle here, as you are greated at the door by a YAT (young attractive female).

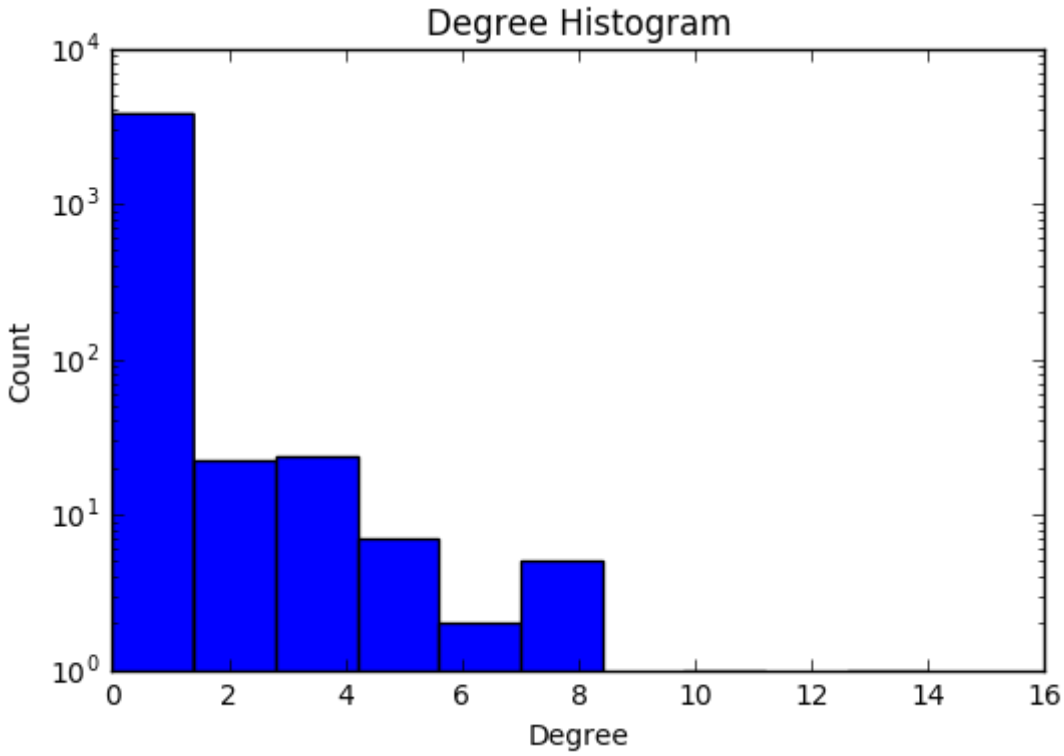
Had a wee swatch at the food as it passed and it looked pretty tasty. Nice place for a wee upmarket drink if your in this neck of the woods.

```
167
2
```

In [13]:
```
nx.write_gexf(g, "friendship.gexf")

print len(g.edges())
print len(g.nodes())
degree_sequence = []

for degree in g.degree().values():
    degree_sequence.append(degree)
```

```
169
3828
```

In [14]:
```
ply.hist(degree_sequence, log=True)
ply.title("Degree Histogram")
ply.ylabel("Count")
ply.xlabel("Degree")
ply.show()
```



In [21]:
```
from textblob import TextBlob

g = nx.Graph()
users = parser.getUsers()
for user in users:
    if user.getEliteNum() > 0:
        g.add_node(user)

nodes1 = g.nodes()
nodes2 = g.nodes()
for node1 in nodes1:
    for node2 in nodes2:
        if node1.getID() != node2.getID():
            for business_id in business_users.keys():
                correview_users = business_users[business_id]
                if node1.getID() in correview_users:
                    if node2.getID() in correview_users:

                        polarity = []
                        for review in user_reviews[node1.getID()]:
                            if review["business_id"] == business_id:
                                b = TextBlob(review["text"])
                                polarity.append(b.sentiment.polarity)
                                break
                        for review in user_reviews[node2.getID()]:
                            if review["business_id"] == business_id:
                                b = TextBlob(review["text"])
                                polarity.append(b.sentiment.polarity)
                                break
                        if (polarity[0] >=0 and polarity[1] < 0) or
(polarity[0] < 0 and polarity[1] >= 0):
                            g.add_edge(node1, node2)
    nodes2.remove(node1)
```

In [16]:
```
outdeg = g.degree()
to_remove = [n for n in outdeg if outdeg[n] == 0]
g.remove_nodes_from(to_remove)
```

In [18]:
```
nx.write_gexf(g, "elite-users-different.gexf")


print (len(g.edges()))
print len(g.nodes())
degree_sequence = []

for degree in g.degree().values():
    degree_sequence.append(degree)
```

```
104
123
```

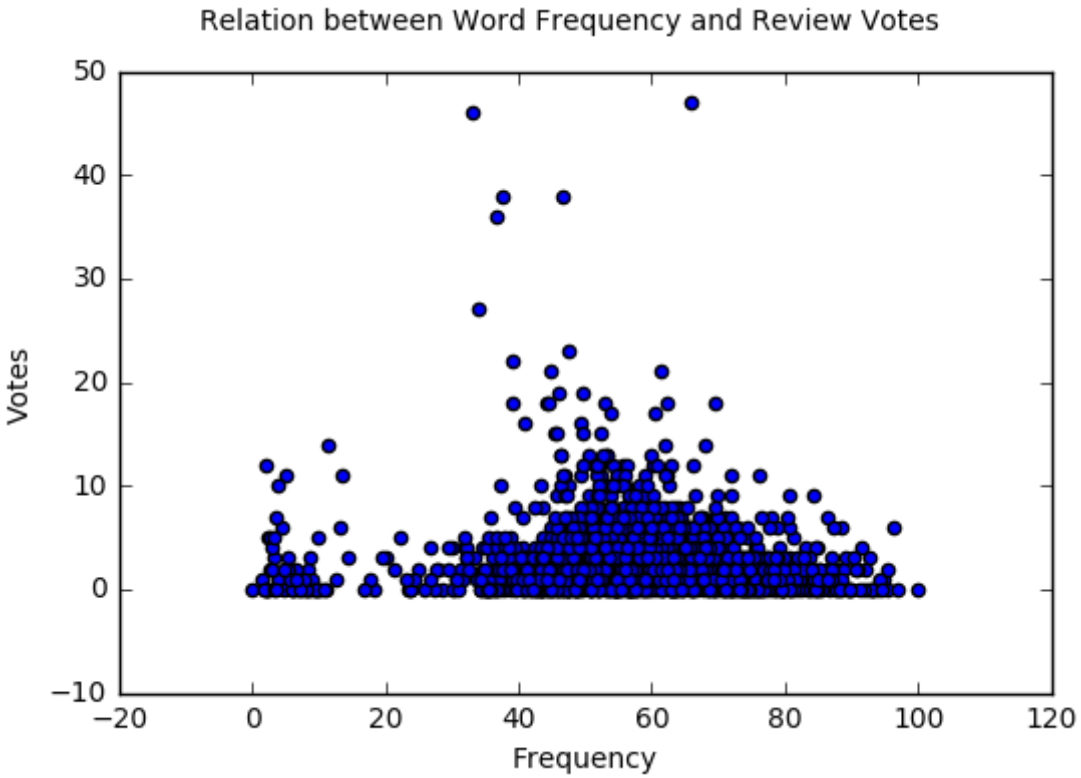
In [ ]:
```
ply.hist(degree_sequence, log=True)
ply.title("Degree Histogram")
ply.ylabel("Count")
ply.xlabel("Degree")
ply.show()
```

In [ ]:
```

```

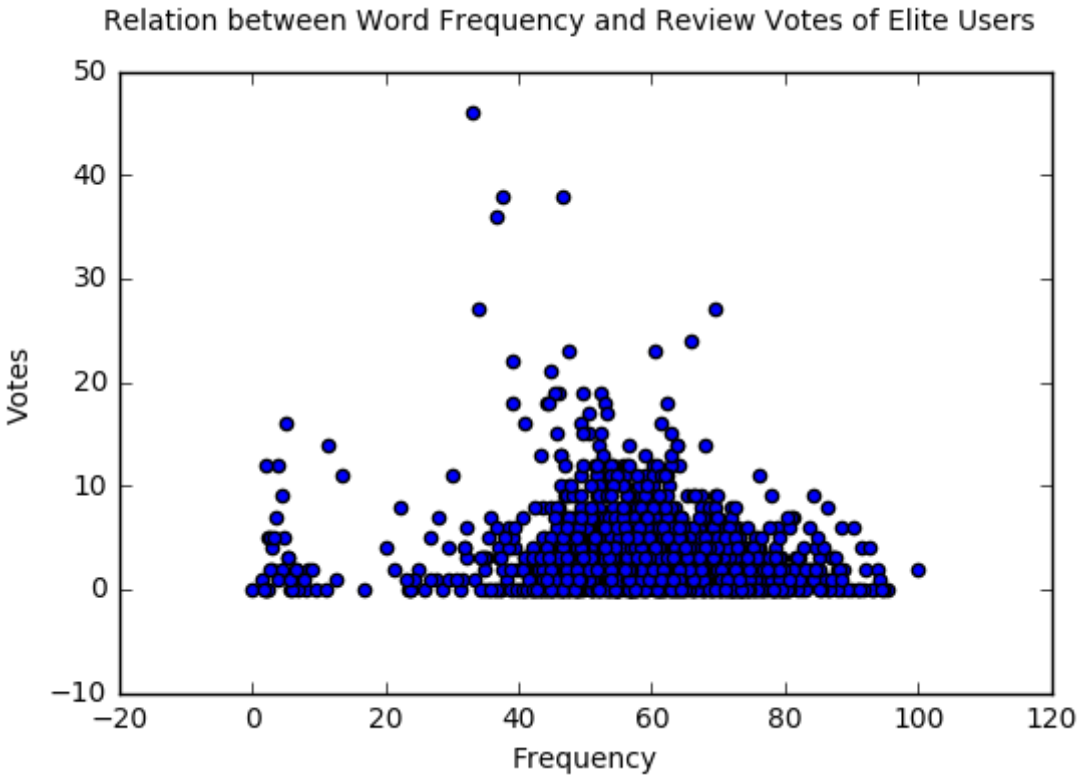
In [1]:
```
import networkx as nx
import matplotlib.pyplot as ply

%matplotlib inline
```

In [2]:
```
from extract_restaurant_reviews import *
```

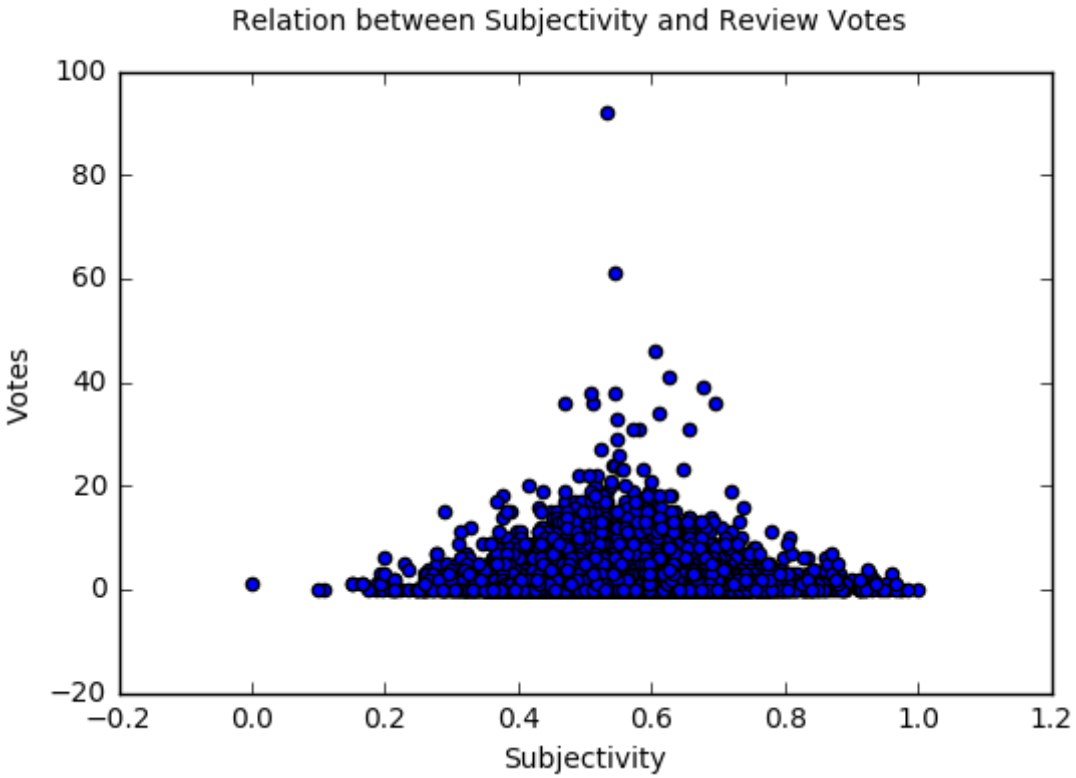
In [3]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Word Frequency and Review Votes')
ax.scatter(frequency_votes.keys(), [frequency_votes[n] for n in frequ
ency_votes.keys()])
ply.xlabel("Frequency")
ply.ylabel("Votes")
ply.show()
```



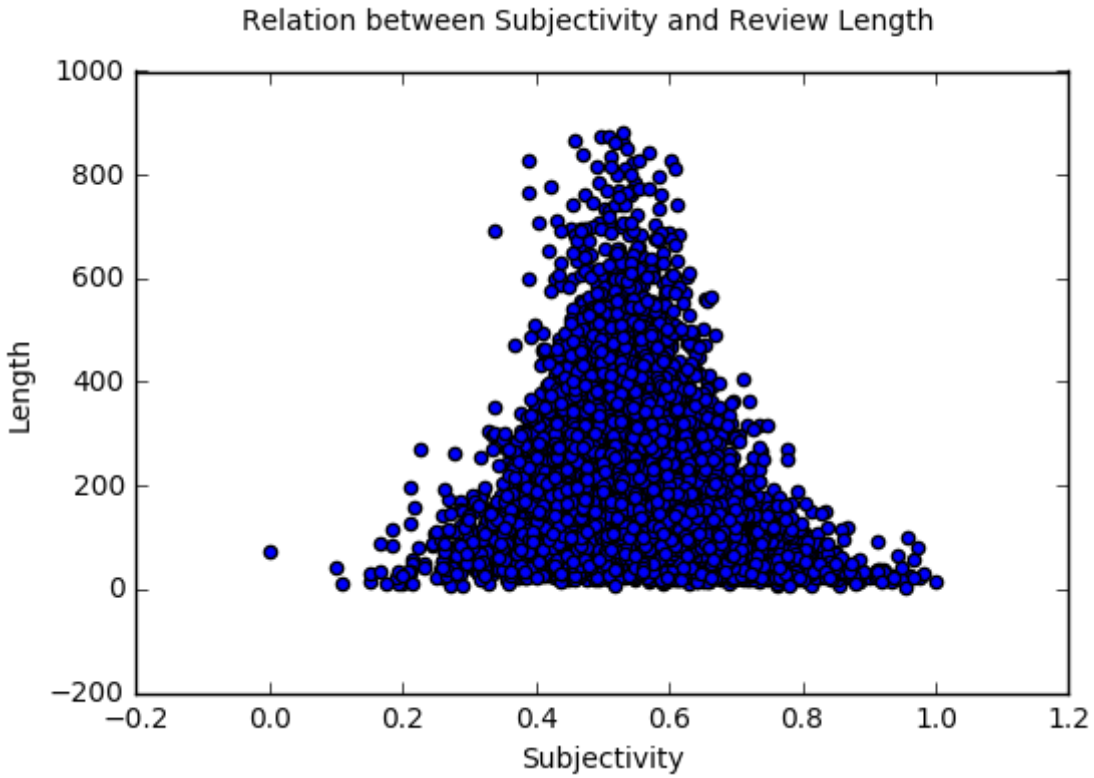
In [10]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Word Frequency and Review Votes of Elite Users')
ax.scatter(elite_frequency_votes.keys(), [elite_frequency_votes[n] for n in elite_frequency_votes.keys()])
ply.xlabel("Frequency")
ply.ylabel("Votes")
ply.show()
```



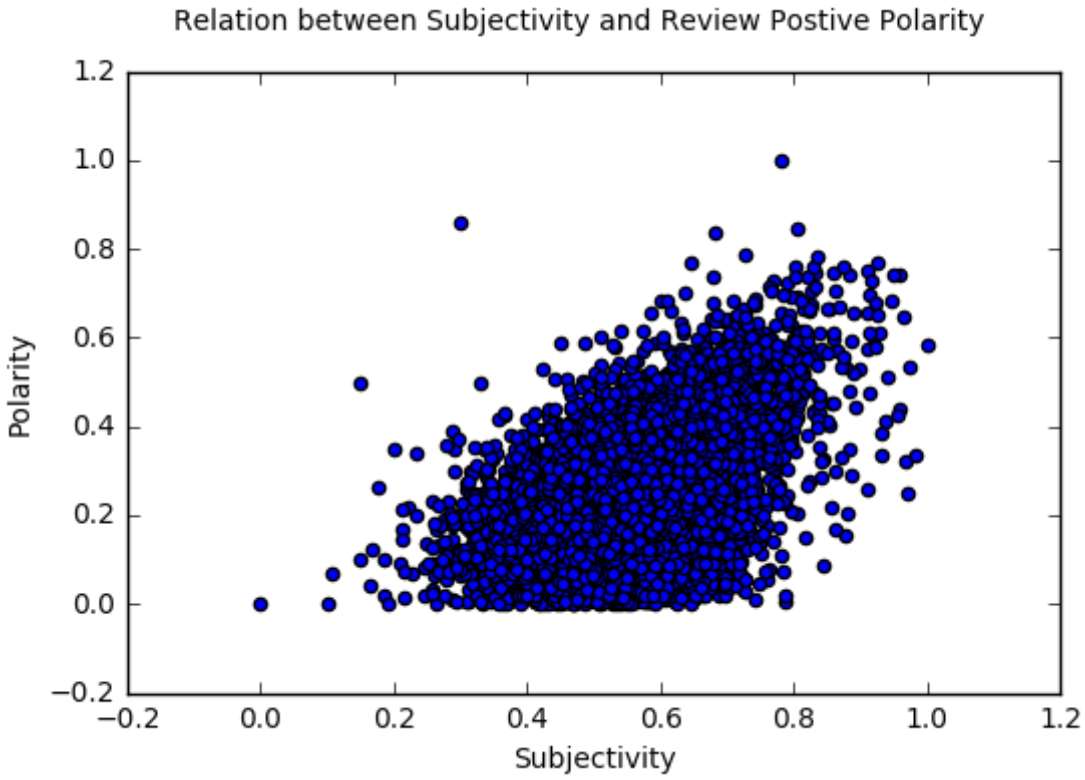
In [5]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Subjectivity and Review Votes')
ax.scatter(subjectivity_votes.keys(), [subjectivity_votes[n] for n in
 subjectivity_votes.keys()])
ply.xlabel("Subjectivity")
ply.ylabel("Votes")
ply.show()
```



In [6]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Subjectivity and Review Length')
ax.scatter(subjectivity_review_length.keys(), [subjectivity_review_le
ngth[n] for n in subjectivity_review_length.keys()])
ply.xlabel("Subjectivity")
ply.ylabel("Length")
ply.show()
```



In [7]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Subjectivity and Review Postive Polari
ty')
ax.scatter(subjectivity_pos_polarity.keys(), [subjectivity_pos_polari
ty[n] for n in subjectivity_pos_polarity.keys()])
ply.xlabel("Subjectivity")
ply.ylabel("Polarity")
ply.show()
```



In [8]:
```
fig = ply.figure()
ax = ply.gca()

fig.suptitle('Relation between Subjectivity and Review Negative Polar
ity')
ax.scatter(subjectivity_neg_polarity.keys(), [subjectivity_neg_polari
ty[n] for n in subjectivity_neg_polarity.keys()])
ply.xlabel("Subjectivity")
ply.ylabel("Polarity")
ply.show()
```