Python scripting for network analysis

Lab session on February 20th

Python environments: Spyder, Jupyter

console

editor

jupyter notebook

Some useful features

tab-completion

"?" lookup

Jupyter notebook: mix of editor & console

multiline editing:

1. Ctrl-Enter: run cell
2. Shift-Enter: run & select next
3. Alt-Enter: run & insert cell

in-line output

Jupyter notebook: different cell types

python code (the default)

markdown: headings, lists, etc. including math notation: $e = A_{ij}$ or as centered:

$$E = mc^2$$

possibly other code (R, Julia, etc. — 40 overall)

Jupyter notebook caveats

out-of-order cells

File → Download As — .ipynb, .py, .html

This presentation is also a jupyter notebook

embedded cells:

In [1]:
```
1+1
```

Out[1]: 2

In [ ]:
```

```

will upload in .ipynb format

need "pip install RISE"

NetworkX

create network, nodes, edges

In [2]:
```
import networkx
```

In [ ]:
```

```

In [3]:
```
g = networkx.Graph()
```

In [4]:
```
g
```

Out[4]:
```
<networkx.classes.graph.Graph at 0x7fb4441a7550>
```

In [5]:
```
g.add_node(1)
```

In [6]:
```
g.nodes()
```

Out[6]:
```
NodeView((1,))
```

In [7]:
```
g.add_edge(2,3)
```

In [11]:
```
networkx.MultiGraph?
```

In [12]:
```
g.nodes()
```

Out[12]:
```
NodeView((1, 2, 3))
```

In [13]:
```
g.edges()
```

Out[13]:
```
EdgeView([(2, 3)])
```

In [14]:
```
g.add_node("asdf")
```

In [15]:
```
g.add_node((1,2,3))
```

In [16]:
```
g.nodes()
```

Out[16]:
```
NodeView((1, 2, 3, 'asdf', (1, 2, 3)))
```

In [17]:
```
g.add_node([1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-f4477ba54a58> in <module>
----> 1 g.add_node([1,2,3])

~/anaconda3/lib/python3.7/site-packages/networkx/classes/graph.py in add_node(sel
f, node_for_adding, **attr)
    497         doesn't change on mutables.
    498         """
--> 499         if node_for_adding not in self._node:
    500             self._adj[node_for_adding] = self.adjlist_inner_dict_factory
()
    501             self._node[node_for_adding] = attr

TypeError: unhashable type: 'list'
```

In [18]:
```
hash((1,2,3))
```

Out[18]:
```
2528502973977326415
```

In [19]:
```
hash([1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-35e31e935e9e> in <module>
----> 1 hash([1,2,3])

TypeError: unhashable type: 'list'
```

In [ ]:

create network, nodes, edges – caveats

different network types

adding edge silently adds missing nodes

not everything can be used as nodes

Node, edge attributes

In [20]:
```
g.add_node(42, some_attribute='value')
g.add_edge(42, 137, some_other_attribute=[1,2,3])
```

In [25]:
```
g.node[42]
```

Out[25]:
```
{'some_attribute': 'value', 'new_attribute': 'new_value'}
```

In [ ]:

In [24]:
```
g[42][137]
```

Out[24]:
```
{'some_other_attribute': [1, 2, 3], 'other_new_attribute': [4, 5, 6]}
```

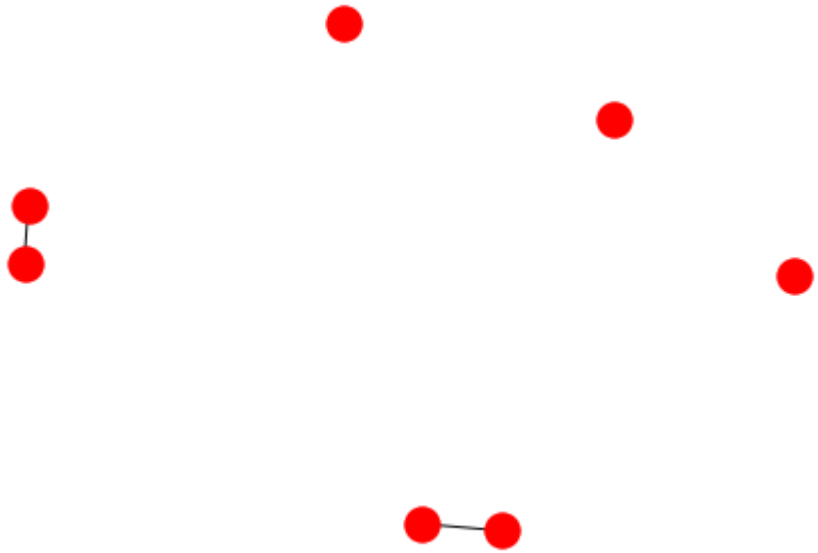
In [ ]:

In [23]:
```
g.node[42]['new_attribute'] = 'new_value'
g[42][137]['other_new_attribute'] = [4,5,6]
```

In [ ]:

Drawing the network

In [27]:
```
networkx.draw(g)
```



In [28]:
```
import matplotlib.pyplot as plt
```

In [29]:
```
plt.show()
```

In [ ]:
```

```

(might need to call matplotlib's show() function)

comparison with cytoscape

Iterating over nodes, edges

In [30]:
```
g.nodes()
g.nodes(data=True)
```

Out[30]:
```
NodeDataView({1: {}, 2: {}, 3: {}, 'asdf': {}, (1, 2, 3): {}, 42: {'some_attribut
e': 'value', 'new_attribute': 'new_value'}, 137: {}})
```

In [ ]:
```
```

In [31]:
```
for node, attrs in g.nodes(data=True):
    print(node, attrs)
```

```
1 {}
2 {}
3 {}
asdf {}
(1, 2, 3) {}
42 {'some_attribute': 'value', 'new_attribute': 'new_value'}
137 {}
```

In [32]:
```
for source, target, attrs in g.edges(data=True):
    print(source, target, attrs)
```

```
2 3 {}
42 137 {'some_other_attribute': [1, 2, 3], 'other_new_attribute': [4, 5, 6]}
```

In [ ]:
```
```

Loading data

In [33]:
```
graph = networkx.read_edgelist('word_association_graph_DSF.txt',
                               create_using=networkx.DiGraph(),
                               nodetype=str, data=[('weight', float),])
```

In [34]:
```
graph.number_of_nodes()
```

Out[34]: 10616

In [35]:
```
graph.number_of_edges()
```

Out[35]: 72171

Calculating degrees

In [37]:
```
len(graph.neighbors(n))
```

In [41]:
```
degrees = {}
for source, target, attrs in graph.edges(data=True):
    if source not in degrees:
        degrees[source] = 1 #.. .create it and set to 1...
    else:
        degrees[source] += 1 #.. .increment it....
```

In [42]:
```
degrees['APPLE']
```

Out[42]:
```
17
```

In [44]:
```
graph.has_node('ALIEN')
```

Out[44]:
```
True
```

In [45]:
```
degrees['ALIEN'],
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-45-0c86eeee7299> in <module>
----> 1 degrees['ALIEN']

KeyError: 'ALIEN'
```

In [ ]:
```
degrees = {}
for source, target, attrs in graph.edges(data=True):
    if source not in degrees:
        degrees[source] = 1 #.. .create it and set to 1...
    else:
        degrees[source] += 1 #.. .increment it....
```

In [38]:
```
d = {}
```

In [39]:
```
d[1] += 1
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-39-b8f44857a8fd> in <module>
----> 1 d[1] += 1

KeyError: 1
```

In [57]:
```
out_degrees = {}
for node in graph.nodes():
    out_degrees[node] = 0
for source, target, attrs in graph.edges(data=True):
    out_degrees[source] += 1 #.. .increment it....
```

In [ ]:
```

```

In [47]:
```
degrees['ALIEN']
```

Out[47]:
```
0
```

In [48]:
```
import collections
```

In [50]:
```
dd = collections.defaultdict(int)
```

In [51]:
```
dd[1]= 42
```

In [52]:
```
dd[2] += 1
```

In [53]:
```
dd
```

Out[53]:
```
defaultdict(int, {1: 42, 2: 1})
```

In [54]:
```
dd['ALIEN']
```

Out[54]:
```
0
```

In [56]:
```
in_degrees = {}
for node in graph.nodes():
    in_degrees[node] = 0
for source, target, attrs in graph.edges(data=True):
    in_degrees[target] += 1 #.. .increment it....
```

In [ ]:

In [ ]:

Looking at the results

In [58]:
```
in_degrees['ALIEN']
```

Out[58]:
```
7
```

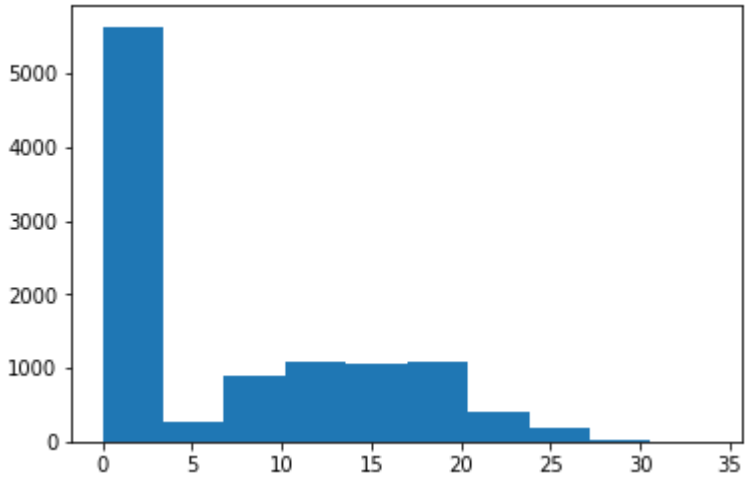
In [60]:
```
in_degrees.values()
```

Out[60]:
```
dict_values([7, 6, 1, 1, 47, 2, 35, 50, 47, 63, 9, 5, 51, 1, 1, 156, 11, 2, 10, 1
7, 18, 132, 49, 1, 1, 21, 70, 13, 1, 6, 83, 3, 1, 2, 34, 15, 158, 110, 1, 1, 22,
1, 1, 7, 10, 101, 2, 4, 1, 8, 70, 36, 63, 18, 14, 16, 36, 4, 8, 1, 5, 48, 1, 51,
1, 51, 13, 4, 53, 13, 50, 32, 12, 17, 6, 196, 1, 1, 3, 2, 50, 3, 11, 92, 28, 7, 1
0, 40, 1, 41, 3, 103, 19, 17, 30, 3, 25, 4, 18, 32, 14, 20, 12, 5, 27, 97, 38, 4
8, 65, 120, 2, 42, 1, 57, 95, 36, 15, 255, 24, 23, 24, 26, 2, 4, 33, 183, 90, 5,
6, 1, 5, 8, 43, 45, 3, 1, 14, 4, 52, 129, 10, 15, 21, 20, 46, 80, 16, 2, 70, 46,
11, 7, 35, 1, 53, 36, 6, 2, 6, 5, 2, 13, 73, 7, 1, 6, 4, 3, 324, 28, 4, 54, 46, 1
6, 46, 302, 26, 7, 1, 1, 154, 14, 1, 2, 17, 5, 276, 12, 44, 229, 25, 46, 90, 75,
137, 1, 5, 2, 11, 19, 17, 3, 259, 1, 85, 44, 70, 10, 4, 30, 49, 32, 94, 17, 9, 1
7, 19, 16, 18, 1, 33, 67, 40, 18, 17, 1, 40, 33, 23, 38, 4, 21, 61, 80, 19, 1, 1
9, 2, 71, 1, 9, 19, 7, 1, 10, 13, 8, 13, 13, 32, 29, 37, 10, 59, 2, 11, 1, 1, 4,
23, 1, 3, 1, 145, 1, 2, 18, 91, 1, 2, 14, 7, 1, 11, 37, 149, 14, 3, 1, 6, 19, 14,
66, 3, 63, 1, 7, 28, 6, 1, 35, 4, 9, 18, 10, 25, 19, 163, 7, 29, 10, 80, 25, 3, 1
0, 62, 2, 25, 1, 11, 28, 1, 13, 8, 47, 37, 22, 87, 7, 10, 4, 49, 61, 47, 10, 18,
26, 38, 34, 71, 60, 67, 2, 42, 14, 30, 5, 6, 17, 1, 7, 2, 53, 10, 46, 11, 1, 25,
19, 15, 47, 77, 27, 17, 12, 11, 24, 28, 8, 6, 19, 16, 41, 14, 12, 2, 45, 20, 71,
13, 8, 6, 10, 47, 21, 43, 15, 30, 3, 61, 16, 49, 6, 9, 7, 7, 10, 12, 38, 4, 12,
4, 4, 16, 4, 60, 72, 31, 3, 2, 17, 10, 10, 8, 106, 25, 34, 17, 2, 6, 17, 30, 3, 3
5, 38, 126, 53, 38, 7, 8, 19, 35, 2, 61, 12, 9, 6, 7, 64, 11, 1, 10, 1, 2, 34, 3,
12, 30, 1, 1, 1, 28, 25, 10, 84, 4, 95, 1, 8, 18, 3, 33, 20, 20, 31, 10, 20, 19,
99, 13, 58, 12, 103, 1, 11, 87, 2, 6, 10, 21, 8, 54, 12, 29, 40, 17, 22, 10, 1,
1, 28, 2, 6, 5, 9, 4, 21, 39, 9, 33, 22, 1, 151, 8, 74, 56, 5, 3, 6, 2, 20, 171,
134, 3, 19, 51, 70, 2, 31, 8, 127, 19, 71, 71, 8, 2, 4, 37, 8, 2, 8, 5, 29, 12, 3
2, 37, 25, 22, 9, 19, 8, 9, 9, 8, 11, 4, 34, 6, 1, 1, 32, 7, 24, 11, 1, 2, 12, 2,
4, 41, 97, 185, 21, 20, 27, 37, 15, 30, 2, 40, 86, 1, 5, 4, 61, 7, 43, 39, 10, 6,
5, 2, 1, 2, 77, 73, 18, 7, 9, 3, 25, 1, 3, 4, 14, 10, 52, 1, 6, 1, 48, 7, 22, 9,
3, 4, 61, 8, 37, 17, 3, 3, 4, 1, 181, 15, 10, 3, 11, 6, 21, 40, 23, 20, 55, 2, 2
7, 24, 52, 9, 17, 2, 34, 3, 11, 11, 1, 89, 2, 15, 1, 12, 7, 26, 64, 1, 31, 40, 9,
3, 55, 5, 11, 24, 4, 2, 62, 2, 19, 24, 3, 30, 3, 13, 48, 112, 32, 5, 1, 4, 26, 1,
37, 2, 11, 14, 4, 29, 120, 78, 41, 3, 1, 4, 3, 21, 2, 15, 1, 3, 11, 13, 19, 5, 3,
15, 3, 19, 6, 2, 2, 13, 2, 8, 1, 30, 8, 4, 83, 4, 2, 59, 17, 2, 8, 4, 20, 3, 1, 1
1, 10, 54, 50, 7, 6, 3, 8, 1, 2, 2, 1, 3, 3, 24, 4, 6, 22, 5, 114, 4, 15, 8, 3,
8, 2, 54, 13, 1, 43, 10, 20, 9, 5, 3, 50, 105, 29, 29, 8, 3, 11, 1, 20, 22, 46, 1
6, 12, 3, 6, 6, 2, 3, 9, 8, 13, 1, 35, 6, 19, 64, 9, 62, 5, 21, 1, 12, 36, 3, 9,
10, 12, 56, 3, 8, 1, 9, 84, 14, 17, 4, 17, 18, 3, 5, 10, 16, 43, 11, 2, 3, 12, 8,
9, 36, 3, 13, 23, 19, 82, 57, 14, 31, 58, 22, 15, 13, 2, 6, 6, 65, 24, 19, 4, 30,
```

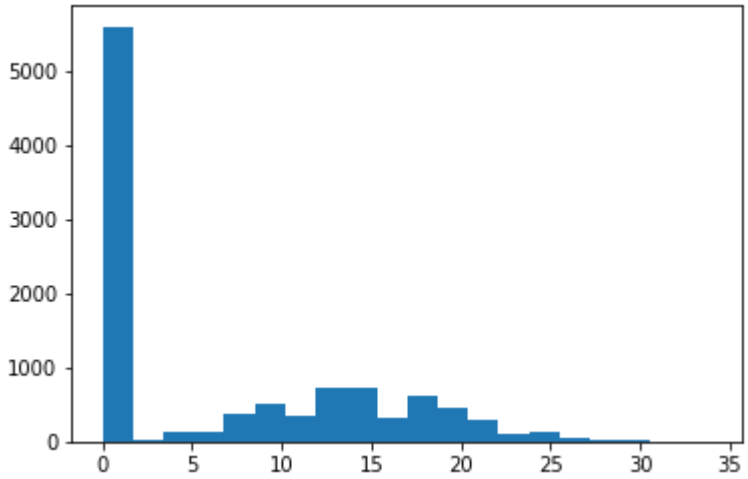
In [61]:
```
import matplotlib.pyplot as plt
```

In [63]:
```
plt.hist(out_degrees.values())
```

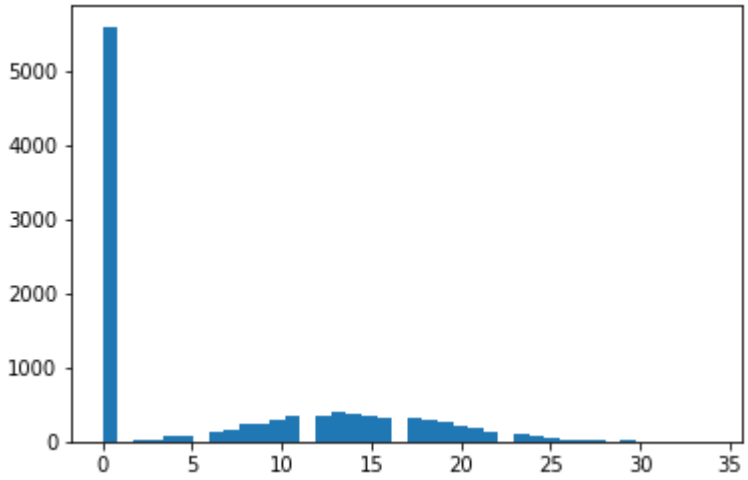
Out[63]:
```
(array([5633.,  256.,  882., 1090., 1049., 1086.,  404.,  182.,   27.,
           7.]),
 array([ 0. ,  3.4,  6.8, 10.2, 13.6, 17. , 20.4, 23.8, 27.2, 30.6, 34. ]),
 <a list of 10 Patch objects>)
```



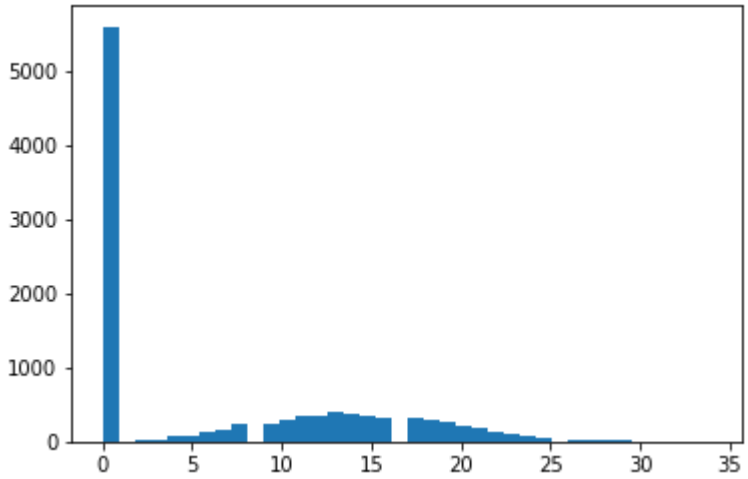
In [65]:
```
_ = plt.hist(out_degrees.values(), bins=20)
```



In [66]:
```
_ = plt.hist(out_degrees.values(), bins=40)
```



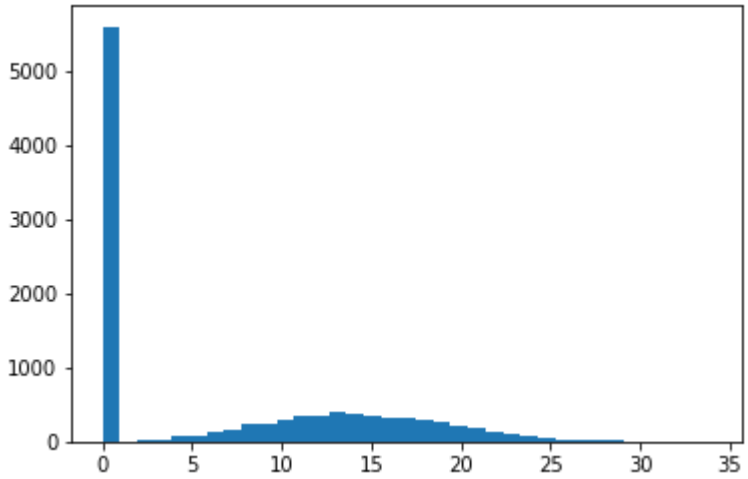
In [67]:
```
_ = plt.hist(out_degrees.values(), bins=38)
```



In [68]:
```
max(out_degrees.values())
```

Out[68]: 34

In [69]:
```
_ = plt.hist(out_degrees.values(), bins=35)
```



In [70]:
```
len([v for v in out_degrees.values() if v == 1])
```

Out[70]: 1

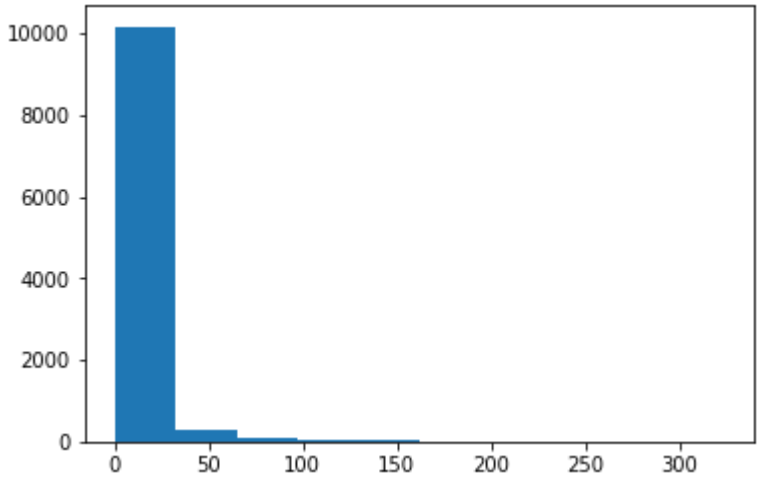
In [71]:
```
len([v for v in out_degrees.values() if v == 2])
```

Out[71]: 13

In [72]:
```
len([v for v in out_degrees.values() if v == 0])
```

Out[72]: 5598

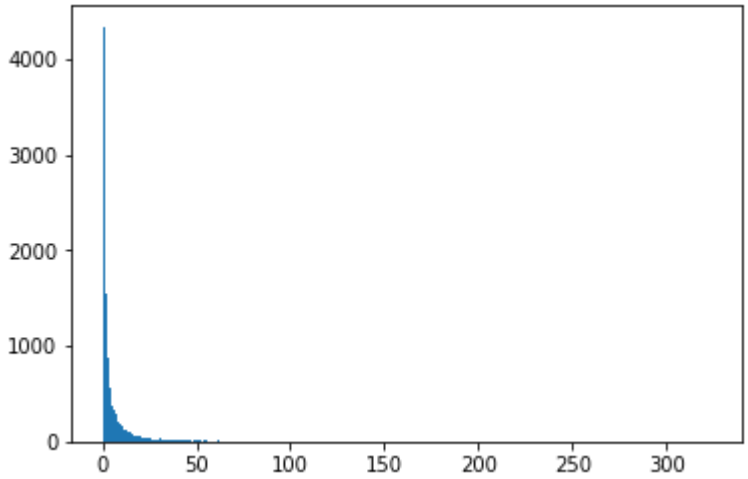
In [73]:
```
_ = plt.hist(in_degrees.values())
```



In [74]:
```
max(in_degrees.values())
```

Out[74]: 324

In [75]:
```
_ = plt.hist(in_degrees.values(), bins=325)
```



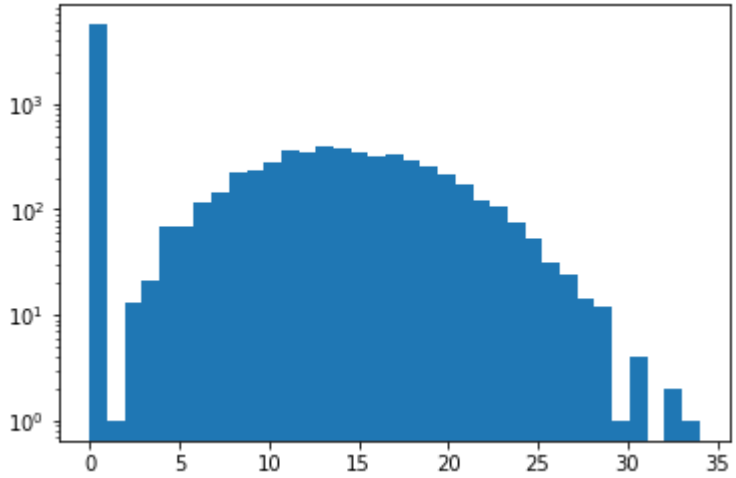
In [76]:
```
%matplotlib
```

```
Using matplotlib backend: Qt5Agg
```

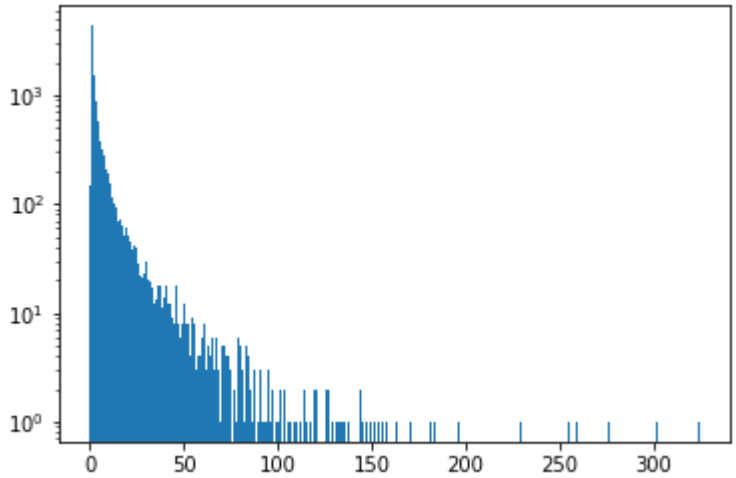
In [79]:
```
_ = plt.hist(in_degrees.values(), bins=325)
```

In [80]:
```
_ = plt.hist(in_degrees.values(), bins=325, log=True)
```

In [81]:
```
%matplotlib inline
_ = plt.hist(out_degrees.values(), bins=35, log=True)
```



In [82]:
```
_ = plt.hist(in_degrees.values(), bins=325, log=True)
```



Homework assignment

Calculate, plot and describe the in- and out-strength (weighted degree) distribution of the word association dataset

Write a script to create a network of a ring of N nodes, with first and second neighbors connected. An example: