Demo: Networks from Real Data

Networks from real data

First off, what directory are we in?

In[•]:=
```
Directory[]
```

Out[•]=
```
/Users/gigglepuss
```

I have a data file!

I really really like League of Legends. It's an online competitive video game. The game developers have their own "restful" API (RESTful is just a standardized API) for collecting any amount of data on the video game... for free! But you do need to make an account to get the data. Check it out here: https://developer.riotgames.com/
Anyways, I downloaded data from the API through Python and then transformed the data into a network. I saved the network into a JSON file.

Here's how I structured the network: Character A ---> Character B Means that A won a game against B. The weight means the number of games where A beat B. This network has parallel edges that go in opposite directions, meaning edges A-->B and B-->A can both exist, each with their own edge weight. There are 140+ nodes in this network and one full month worth of games for 200 players in North America (only the best 200 players on the server!).

I only collected the data and made a network, I know nothing about it!

Let's see if we can load the file here:

In[2]:=
```
json = Import["PycharmProjects/League/network_demo.json"]
```

Out[•]=
{directed → True, multigraph → False, graph → ⋯ 1 ⋯, nodes → {⋯ 1 ⋯}, links →
{{weight → 42, source → 81, target → 53}, {weight → 92, source → 81, target → 43},
{weight → 148, source → 81, target → 164}, {weight → 203,
source → 81, target → 145}, {weight → 28, source → 81, target → 240},
{weight → 38, source → 81, target → 99}, {weight → 48, source → 81, target → 62},
{weight → 491, source → 81, target → 236}, {weight → 52, source → 81,
target → 432}, {weight → 207, source → 81, target → 555},
{weight → 67, source → 81, target → 238}, {weight → 56, source → 81, target → 28},
{weight → 112, source → 81, target → 202}, ⋯ 18 216 ⋯,
{weight → 1, source → 75, target → 5}, {weight → 1, source → 75, target → 163},
{weight → 1, source → 75, target → 89}, {weight → 2, source → 75, target → 53},
{weight → 1, source → 75, target → 238}, {weight → 1, source → 75, target → 55},
{weight → 1, source → 75, target → 427}, {weight → 1, source → 75, target → 114},
{weight → 1, source → 75, target → 30}, {weight → 1, source → 75, target → 58},
{weight → 2, source → 75, target → 497}, {weight → 1, source → 75, target → 51},
{weight → 1, source → 75, target → 105}}}

large output | show less | show more | show all | set size limit...

Hm... We need to get this into a format Mathematica will like!

If you're a data engineer, this is what you do! ;)

Let's look at the Graph function to see what kinds of formats it accepts.

Graph

It would be easiest to made an edge list like with the CAs above. We can put the weights in there as well, as long as we preserve the order of the edges.
It looks like we can get all of this information from the "links ->" part of the file.

In[3]:=
```
links = Lookup[json, "links"]
```

Out[3]=
{{weight → 42, source → 81, target → 53},
{weight → 92, source → 81, target → 43}, {weight → 148, source → 81, target → 164},
{weight → 203, source → 81, target → 145},
{weight → 28, source → 81, target → 240}, {weight → 38, source → 81, target → 99},
{weight → 48, source → 81, target → 62}, ⋯ 18 229 ⋯,
{weight → 1, source → 75, target → 114}, {weight → 1, source → 75, target → 30},
{weight → 1, source → 75, target → 58}, {weight → 2, source → 75, target → 497},
{weight → 1, source → 75, target → 51}, {weight → 1, source → 75, target → 105}}

large output | show less | show more | show all | set size limit...

Let's get rid of those arrows.

In[4]:=
```
links = Values /@ links
```

Out[4]=
```
{{42, 81, 53}, {92, 81, 43}, {148, 81, 164}, {203, 81, 145}, {28, 81, 240},
 {38, 81, 99}, {48, 81, 62}, {491, 81, 236}, {52, 81, 432}, {207, 81, 555},
 {67, 81, 238}, {56, 81, 28}, {112, 81, 202}, {76, 81, 516}, {129, 81, 8},
 {139, 81, 104}, {46, 81, 31}, {66, 81, 114}, {66, 81, 41}, {82, 81, 64},
 {76, 81, 201}, {41, 81, 51}, {32, 81, 1}, {43, 81, 96}, … 18 194 …,
 {2, 75, 25}, {2, 75, 142}, {2, 75, 80}, {3, 75, 145}, {1, 75, 54}, {1, 75, 20},
 {2, 75, 98}, {1, 75, 1}, {1, 75, 91}, {1, 75, 85}, {1, 75, 21}, {1, 75, 5},
 {1, 75, 163}, {1, 75, 89}, {2, 75, 53}, {1, 75, 238}, {1, 75, 55}, {1, 75, 427},
 {1, 75, 114}, {1, 75, 30}, {1, 75, 58}, {2, 75, 497}, {1, 75, 51}, {1, 75, 105}}
```

large output | show less | show more | show all | set size limit...

That is a nice list! It is a list of lists. For each of the lists in the list, we can make an edge list by taking the second and third element. How can we easily take them? By making a special function! But this time we are going to do it in shorthand. In this format, the # is the variable, the & activates it, and /@ means "apply it to all the elements in this list".

In[5]:=
```
edges = #[[2]] → #[[3]] & /@ links
```

Out[5]=
```
{81 → 53, 81 → 43, 81 → 164, 81 → 145, 81 → 240, 81 → 99, 81 → 62, 81 → 236, 81 → 432,
 81 → 555, 81 → 238, 81 → 28, 81 → 202, 81 → 516, 81 → 8, 81 → 104, 81 → 31,
 81 → 114, 81 → 41, 81 → 64, 81 → 201, 81 → 51, 81 → 1, 81 → 96, 81 → 79, 81 → 83,
 81 → 98, 81 → 13, 81 → 40, 81 → 107, 81 → 85, 81 → 203, 81 → 58, 81 → 245,
 81 → 5, 81 → 35, 81 → 26, 81 → 136, 81 → 17, 81 → 110, 81 → 101, 81 → 127,
 … 18 159 …, 75 → 96, 75 → 23, 75 → 3, 75 → 117, 75 → 164, 75 → 48, 75 → 267,
 75 → 61, 75 → 81, 75 → 35, 75 → 41, 75 → 16, 75 → 50, 75 → 19, 75 → 27, 75 → 104,
 75 → 31, 75 → 25, 75 → 142, 75 → 80, 75 → 145, 75 → 54, 75 → 20, 75 → 98,
 75 → 1, 75 → 91, 75 → 85, 75 → 21, 75 → 5, 75 → 163, 75 → 89, 75 → 53, 75 → 238,
 75 → 55, 75 → 427, 75 → 114, 75 → 30, 75 → 58, 75 → 497, 75 → 51, 75 → 105}
```

large output | show less | show more | show all | set size limit...

To get the weights, we can just take the first element. Easy!

In[6]:=
```
weights = First /@ links
```

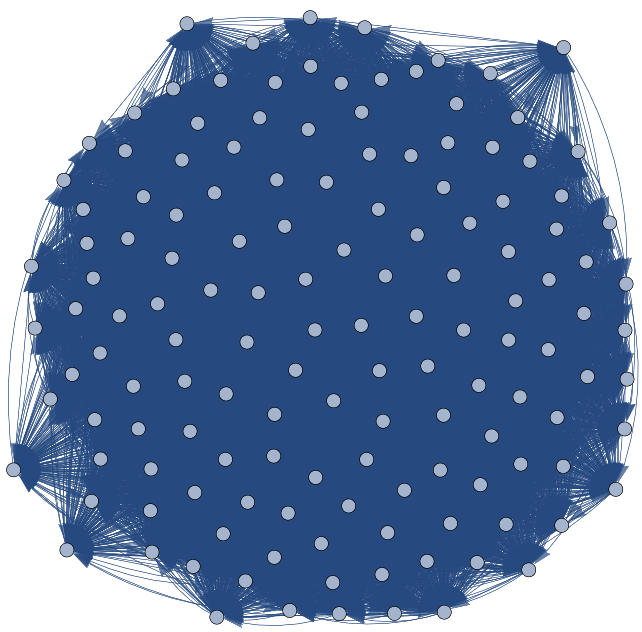
Out[6]=
```
{42, 92, 148, 203, 28, 38, 48, 491, 52, 207, 67, 56, 112, 76, 129, 139, 46, 66, 66, 82,
 76, 41, 32, 43, 35, 7, 32, 43, 93, 95, 61, 47, 105, 42, 119, 28, 16, 37, 23, 54,
 23, 27, 78, 79, 114, 75, 112, 156, 12, 115, 119, 47, 26, 60, 33, 22, 35, 87, 16,
 63, 115, 34, 111, 44, 57, 56, 37, 47, 111, 17, 85, 104, 22, 46, 70, 47, 36, 34,
 18, 23, 82, 20, 24, 47, 25, 57, 17, 41, 20, 15, 21, 50, 90, 15, 97, 41, 16, 125,
 30, 42, 34, 15, 32, … 18 036 …, 1, 2, 1, 2, 4, 1, 1, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 4, 2, 1, 1, 3, 3, 2, 2, 2, 1,
 1, 2, 1, 1, 4, 1, 1, 2, 1, 1, 2, 1, 4, 3, 8, 2, 2, 1, 1, 2, 3, 2, 1, 1, 2, 1, 2, 1, 1,
 1, 3, 2, 1, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1}
```

large output | show less | show more | show all | set size limit...

Excellent! We are now ready to plot our graph..... or.... are we? See how there are more than 18K edges. Ouch. I hope this doesn’t break my laptop.

In[7]:=
```
biggraph = Graph[edges, EdgeWeight → weights]
```

Out[7]=



Gross. Well, it doesn’t really look like anything so it’s difficult to get any information out of looking at it. Instead, it would be much easier to apply some measures to gain a sense of what is happening!

Let’s blast it with measures!

In the documentation, you can search “graph properties and measurements” and a whole bunch of things will pop up.

In[8]:=
```
VertexCount[biggraph]
EdgeCount[biggraph]
```

Out[8]= 141

Out[9]= 18 242

In[22]:=
```
VertexDegree[biggraph]
VertexList[biggraph]
SortBy[Transpose[{VertexList[biggraph], VertexDegree[biggraph]}], Last]
```

Out[22]=
```
{280, 274, 279, 280, 280, 266, 265, 273, 280, 273, 280, 276, 274, 279, 276, 279,
 280, 267, 275, 277, 278, 276, 263, 268, 266, 260, 219, 274, 267, 280, 277, 279,
 267, 278, 274, 278, 263, 230, 266, 259, 265, 254, 246, 280, 273, 279, 276, 279,
 280, 221, 279, 280, 274, 242, 270, 254, 248, 261, 278, 247, 275, 278, 253, 280,
 278, 275, 278, 267, 273, 278, 246, 280, 280, 250, 279, 275, 274, 264, 263, 260,
 249, 278, 268, 258, 272, 260, 272, 230, 276, 268, 262, 249, 274, 280, 219, 279,
 267, 239, 279, 263, 276, 258, 245, 260, 249, 250, 274, 276, 275, 268, 277,
 216, 203, 271, 247, 237, 266, 256, 233, 241, 223, 223, 243, 273, 243, 266,
 259, 258, 211, 231, 167, 246, 202, 256, 166, 224, 158, 220, 240, 144, 184}
```

Out[23]=
```
{81, 53, 43, 164, 145, 240, 99, 62, 236, 432, 555, 238, 28, 202, 516, 8, 104,
 31, 114, 41, 64, 201, 51, 1, 96, 79, 83, 98, 13, 40, 107, 85, 203, 58, 245,
 5, 35, 26, 136, 17, 110, 101, 127, 7, 80, 119, 16, 25, 157, 222, 27, 39, 50,
 38, 126, 19, 111, 29, 4, 2, 56, 117, 131, 267, 77, 23, 91, 92, 141, 497, 6,
 142, 24, 161, 9, 121, 54, 34, 421, 3, 45, 63, 69, 59, 143, 90, 21, 113, 55,
 82, 44, 103, 498, 12, 60, 163, 112, 254, 48, 67, 105, 89, 84, 20, 22, 133,
 266, 76, 412, 11, 122, 154, 429, 78, 72, 74, 37, 134, 18, 268, 42, 150, 115,
 36, 30, 68, 61, 15, 420, 14, 32, 86, 57, 10, 33, 427, 102, 120, 223, 75, 106}
```

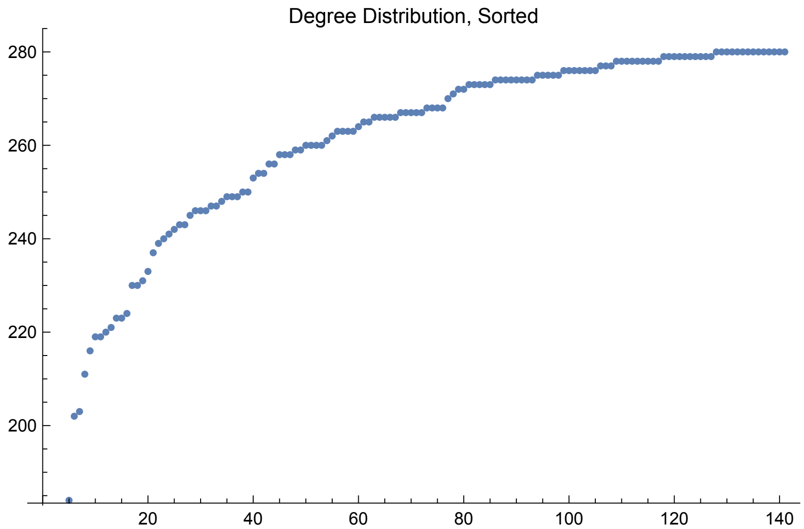
Out[24]=
```
{{75, 144}, {102, 158}, {33, 166}, {32, 167}, {106, 184}, {57, 202}, {429, 203},
 {420, 211}, {154, 216}, {60, 219}, {83, 219}, {120, 220}, {222, 221}, {42, 223},
 {150, 223}, {427, 224}, {26, 230}, {113, 230}, {14, 231}, {18, 233}, {74, 237},
 {254, 239}, {223, 240}, {268, 241}, {38, 242}, {30, 243}, {115, 243}, {84, 245},
 {6, 246}, {86, 246}, {127, 246}, {2, 247}, {72, 247}, {111, 248}, {22, 249},
 {45, 249}, {103, 249}, {133, 250}, {161, 250}, {131, 253}, {19, 254}, {101, 254},
 {10, 256}, {134, 256}, {15, 258}, {59, 258}, {89, 258}, {17, 259}, {61, 259},
 {3, 260}, {20, 260}, {79, 260}, {90, 260}, {29, 261}, {44, 262}, {35, 263}, {51, 263},
 {67, 263}, {421, 263}, {34, 264}, {99, 265}, {110, 265}, {37, 266}, {68, 266},
 {96, 266}, {136, 266}, {240, 266}, {13, 267}, {31, 267}, {92, 267}, {112, 267},
 {203, 267}, {1, 268}, {11, 268}, {69, 268}, {82, 268}, {126, 270}, {78, 271},
 {21, 272}, {143, 272}, {36, 273}, {62, 273}, {80, 273}, {141, 273}, {432, 273},
 {28, 274}, {50, 274}, {53, 274}, {54, 274}, {98, 274}, {245, 274}, {266, 274},
 {498, 274}, {23, 275}, {56, 275}, {114, 275}, {121, 275}, {412, 275}, {16, 276},
 {55, 276}, {76, 276}, {105, 276}, {201, 276}, {238, 276}, {516, 276}, {41, 277},
 {107, 277}, {122, 277}, {4, 278}, {5, 278}, {58, 278}, {63, 278}, {64, 278},
 {77, 278}, {91, 278}, {117, 278}, {497, 278}, {8, 279}, {9, 279}, {25, 279},
 {27, 279}, {43, 279}, {48, 279}, {85, 279}, {119, 279}, {163, 279}, {202, 279},
 {7, 280}, {12, 280}, {24, 280}, {39, 280}, {40, 280}, {81, 280}, {104, 280},
 {142, 280}, {145, 280}, {157, 280}, {164, 280}, {236, 280}, {267, 280}, {555, 280}}
```

So many things we could explore, so let’s focus on a game-centric aspect.
Since A-->B means that character A beat character B, character A would be really a really good charac-

ter to play if it a ton of out-degrees and very little in-degrees! Let’s explore this idea a little bit. First, what is the distribution of all degrees? In degree and out degrees together? Separate?

In[43]:=
```
ListPlot[VertexDegree[biggraph] // Sort, PlotLabel → "Degree Distribution, Sorted"]
```

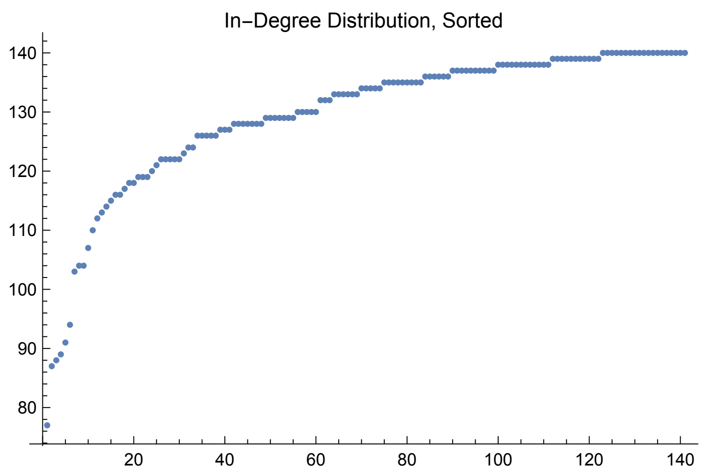
Out[43]=



The x-axis doesn’t matter here since the numbers are just labels for characters.

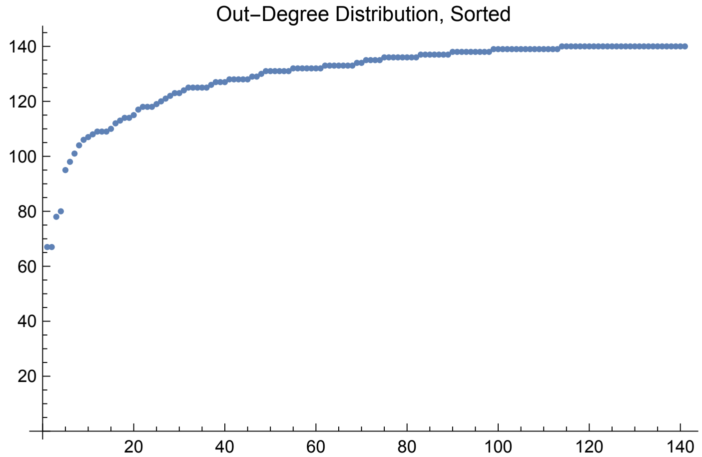
In[30]:=
```
ListPlot[VertexInDegree[biggraph] // Sort,
 PlotLabel → "In-Degree Distribution, Sorted"]
```

Out[30]=



In[31]:=
```
ListPlot[VertexOutDegree[biggraph] // Sort,
 PlotLabel → "Out-Degree Distribution, Sorted"]
```

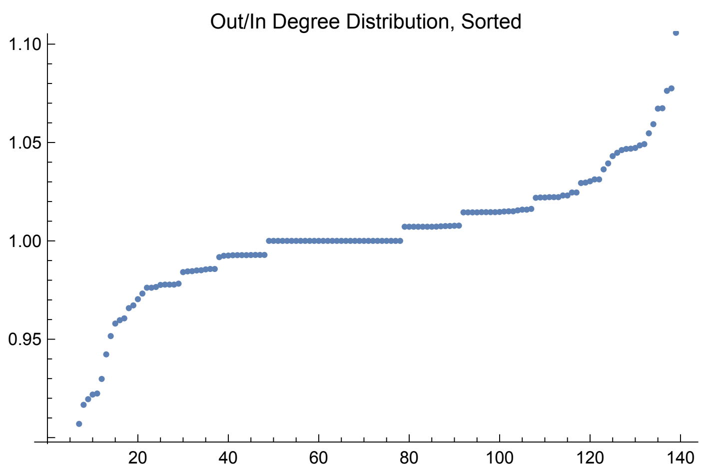
Out[31]=



It looks like, for the most part, the majority of characters have similar in-degree and out-degree, except for a small handful of characters. Perhaps these characters are unpopular or something. Let's see the ratio of out-degree and in-degree. A high ratio indicates a character beats more characters than loses against more characters.

In[44]:=
```
ListPlot[(VertexOutDegree[biggraph] / VertexInDegree[biggraph]) // Sort,
 PlotLabel → "Out/In Degree Distribution, Sorted"]
```

Out[44]=

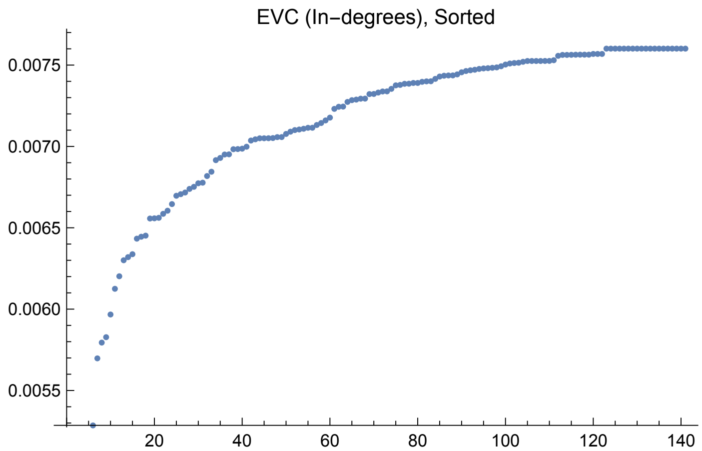


Whoa! That looks really cool! Those characters on the right must be really good, and the ones on the left must be really bad...
However, it may not be actually be reflective of our intuition since there are edge weights. Some edges are really heavy, meaning there may A-->B for a TON of matches and B-->A for only one or two matches. The edges are not equal! Let's see if there are measures that take edge weight into account. Also A-->B edges and A-->C edges can have different weights too.

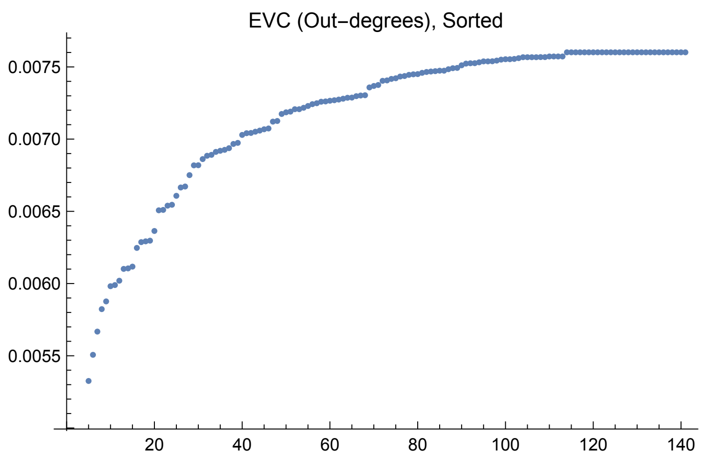
Back to the guide!

EigenvectorCentrality is a really good one to use. It's a lot like Page Rank. Basically, it means that if someone is connected a LOT to a bunch of other people, it has a higher value. But it has an even HIGHER value if all the nodes are connected to have a high value! Intuitively, it means you're more popular if you have lots of popular friends.

In[38]:=
```
ListPlot[EigenvectorCentrality[biggraph, "In"] // Sort,
 PlotLabel → "EVC (In-degrees), Sorted"]
```

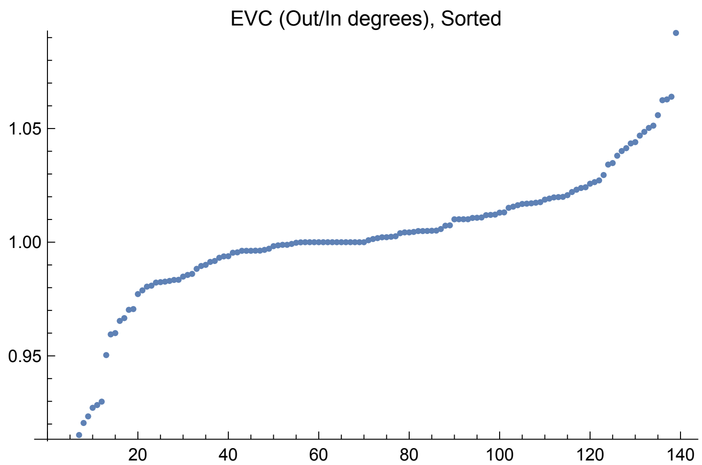
Out[38]=


In[39]:=
```
ListPlot[EigenvectorCentrality[biggraph, "Out"] // Sort,
 PlotLabel → "EVC (Out-degrees), Sorted"]
```

Out[39]=


In[41]:=
```
ListPlot[
  (EigenvectorCentrality[biggraph, "Out"] / EigenvectorCentrality[biggraph, "In"]) //
   Sort, PlotLabel → "EVC (Out/In degrees), Sorted"]
```

Out[41]=



Most excellent! Now we can, with a higher degree of confidence, say that these make intuitive sense.