GETTING STARTED WITH WOLFRAM LANGUAGE: SERVICECONNECT, SENTIMENT ANALYSIS, and social networkS

Swede White, Wolfram Research

Outline of Presentation

SideNotes cell

SideNotes and SideCode cells will not show during presentation.

- A brief overview of Wolfram Notebooks and Documentation
- Example 1: ServiceConnect[] with Reddit
- Example 2: ServiceConnect[] with Twitter
- Example 3: Trade Show Influencer Networks and Information Diffusion

Wolfram Notebook Interface and Documentation

- Code is inputted with immediate output.

- Cells contain code, text, or other content that can be expanded, collapsed, grouped.
- The Wolfram Documentation Center contains information on all functions contained here and elsewhere.
  - To quickly access documentation, highlight the function, and press shift+Command+f or use the Help dropdown menu.

```
ServiceConnect
```

---

example 1: reddit and stephen wolfram's *idea makers*

Stephen Wolfram wrote a book called *Idea Makers* in 2016, and we are interested in examining a corresponding "Ask Me Anything" (AMA) he conducted on Reddit about the book and its contents. Reddit is an online platform known as "the front page of the internet." In the popular r/IAmA sub-reddit, individuals and organizations of note answer questions from anyone participating.

Grab comments and build a text network using Nearest

First, we use ServiceConnect[] to interface with Reddit's API. You will need a Reddit account for the Wolfram Connector to properly access the API. A web browser will open asking for permission for the Wolfram Connector to access Reddit. This has the advantage of not having to keep track of API tokens. We define our service connection below. We will put a semicolon after our line of code to suppress the output.

create service connection

```
reddit = ServiceConnect["Reddit"];
```

We then use our defined service connection to grab post comment data from Stephen Wolfram's *Idea Makers* AMA using the URL. Note, the Reddit API has a rate limit of 500 comments.

grab comments

```
amaComments = reddit[
   "PostCommentsData",
   "Post" ->
    "https://www.reddit.com/r/IAmA/comments/4tspts/im_stephen_wolframask_me_anything/"
   MaxItems -> 500];
```

We now parse out comments and the data of relevance to us, including comment author, the body of the comment, its score, creation date, whether or not it was edited, and its GlobalID.

```
comments = amaComments["Comments"];
data =
  comments[All, {"Author", "Body", "Score", "CreationDate", "Edited", "GlobalID"}];
```

use machine learning to find similarities

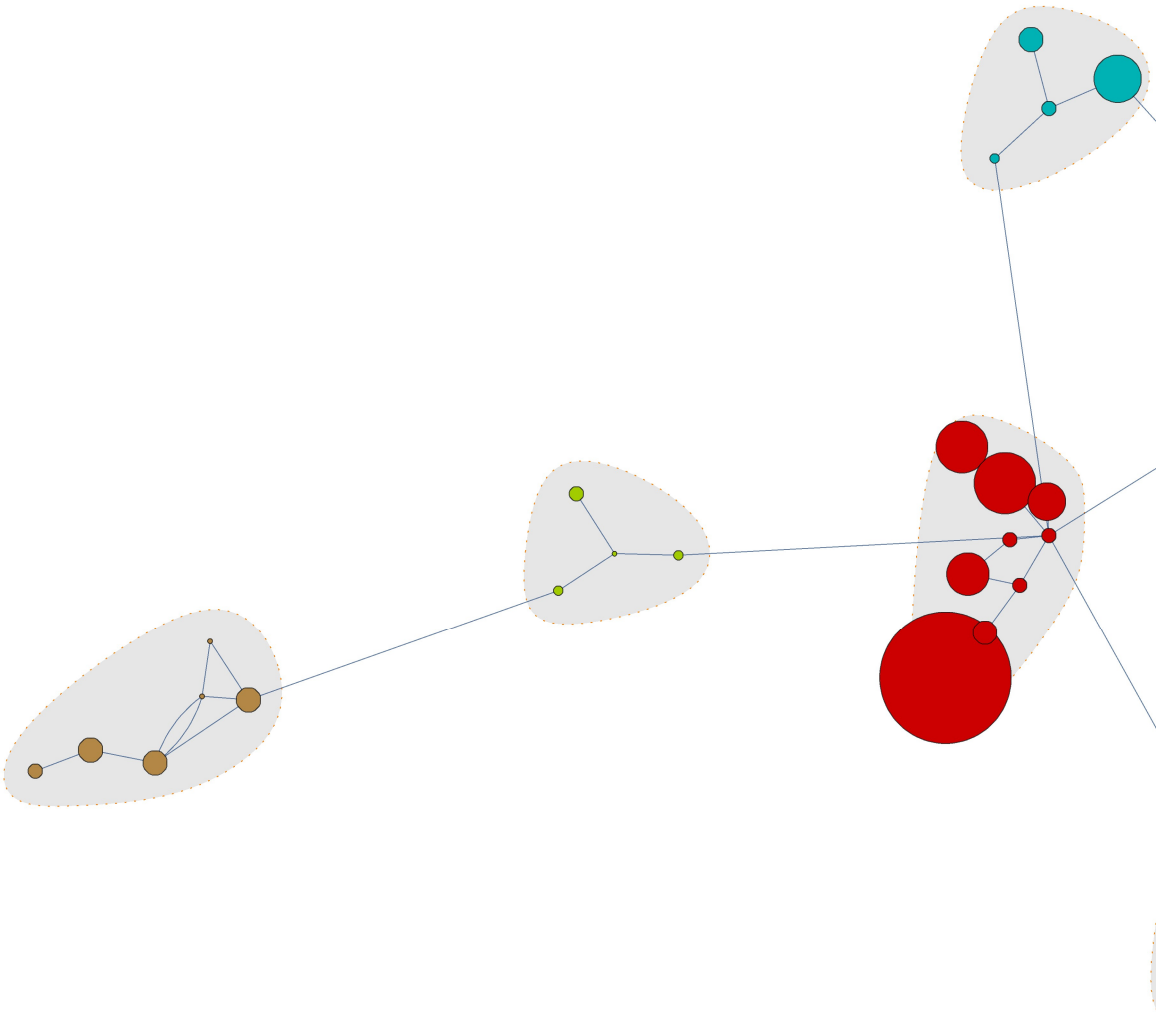
Next, we use the machine learning function Nearest[] to find similarities between the text of comments in the AMA.

```
similarities = Flatten[
   Outer[Rule, {#}, Nearest[DeleteCases[Normal[data[All, "Body"]], #], #]] & /@
    Normal[data[All, "Body"]]];
```

build a community graph

Now that we have our similar comments defined, we can build a graph to visualize these similarities. Each vertex (or node) of the graph is connected by edges (or lines) that tell us how all these comments relate to one another in feature space. We begin by constructing a graph then using CommunityGraphPlot[] to cluster the most similar comments together. We also scale each each vertex by its score, or up-vote, in the Reddit AMA so we have an idea of comment popularity.

```
With[{g = Graph[similarities,
    VertexSize -> Normal[data[All, #["Body"] -> {"Scaled", #["Score"] / 500} &]],
  VertexLabels -> Placed["Name", Tooltip], DirectedEdges -> False, ImageSize -> Large]},
 CommunityGraphPlot[#, FindGraphCommunities[#],
    CommunityRegionStyle -> Directive[Opacity[.2], Gray],
    CommunityBoundaryStyle -> Directive[Orange, Dotted],
    ImageSize -> Large, DirectedEdges -> False] &@g]
```



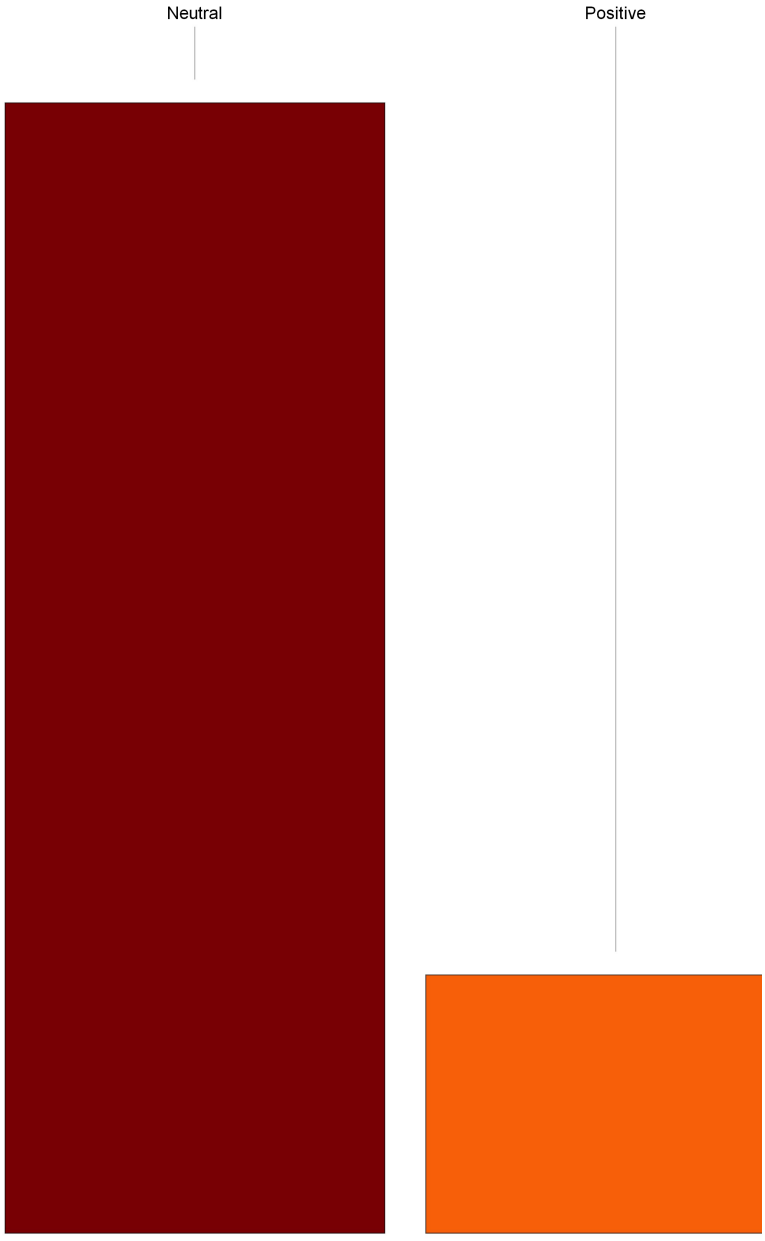
Since we used Tooltip[] in our VertexLabels[] function, we can use the mouse to hover over each vertex to view the text of the AMA comment.

Use machine learning for sentiment and topic analysis

Next, we can use a built-in Wolfram Language sentiment classifier to get an understanding of the emotional content of the comments overall. We first run our classifier then tally each feature of positive, negative, and neutral to build our chart. We'll then use ColorFunction[] to style our chart.

built-in sentiment classifier

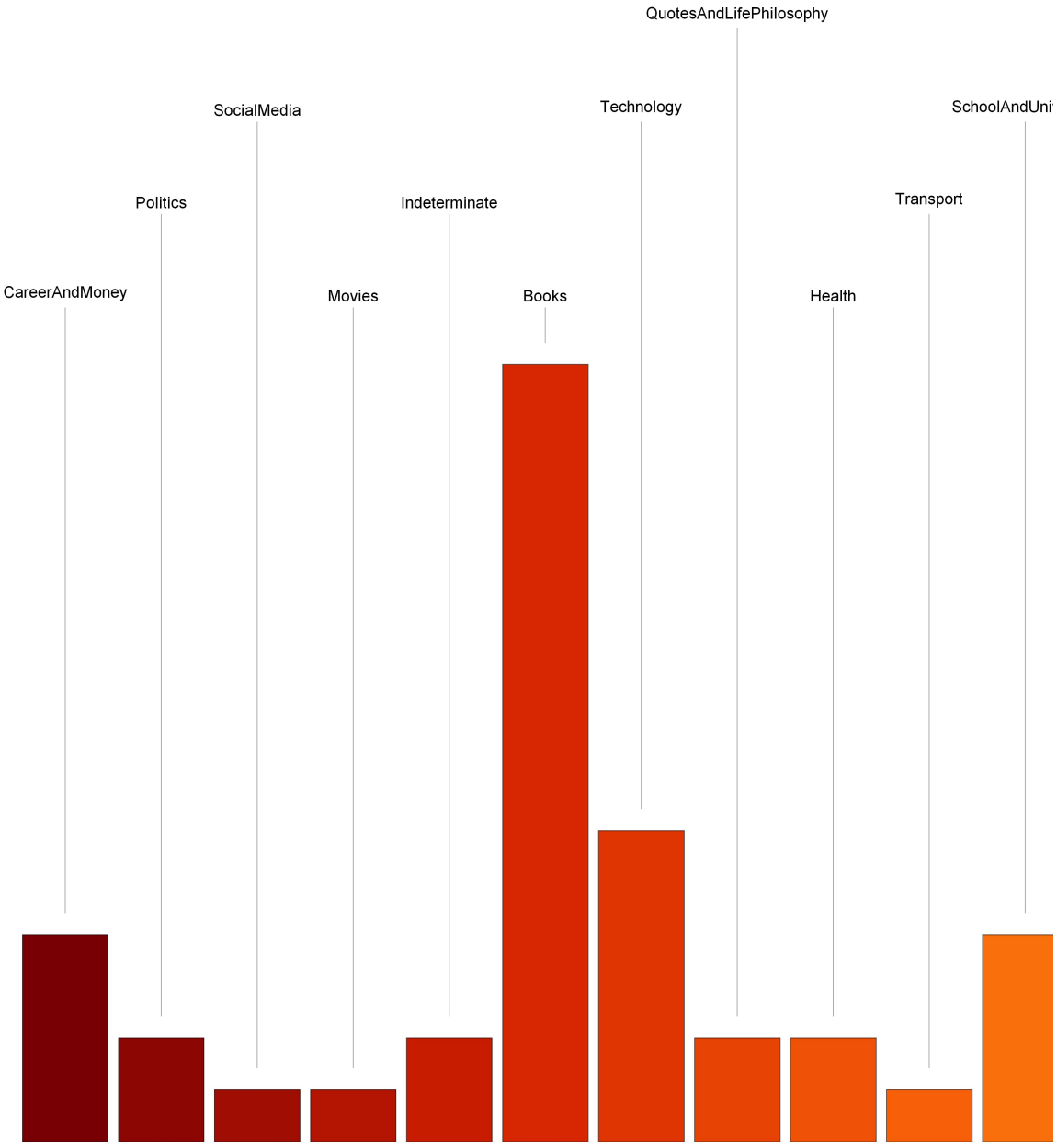
```
cSent = Tally@Classify["Sentiment", data[All, "Body"]];
BarChart[Apply[Callout, Reverse[cSent, 2], {1}],
 Axes -> False, ChartStyle → "SolarColors", ImageSize → Large]
```



As we can see, most comments are neutral with a slightly higher number being binned as negative. Next, we can use another built-in Wolfram Language classifier to examine what topics each comment fall into using machine learning. This particular classifier “FacebookTopic” is trained on millions of Facebook statuses through a Wolfram Research data donor program.

built-in topic classifier

```
categories = Tally@Classify["FacebookTopic", data[All, "Body"]];
BarChart[Apply[Callout, Reverse[categories, 2], {1}],
 Axes -> False, ChartStyle → "SolarColors", ImageSize → Large]
```



example 2: examining twitter feeds

service connection and exploratory analysis

We now use another ServiceConnect[] API for Twitter following the same procedure as Reddit. We first define our new service connection for Twitter and a web browser will ask for permission to

access the Twitter API through your account.

create service connection

```
twitter = ServiceConnect["Twitter"];
```

get basic account information

We now get basic information about Wolfram Research's Twitter feed as an example. You can access any account on Twitter using this function.

```
twitter["UserData", "Username" → "wolframresearch"]
```

```
{ ‹| ID → 22 659 609, Name → Wolfram, ScreenName → WolframResearch, Location → Champaign, IL,
   FollowersCount → 31 128, FriendsCount → 376, FavouritesCount → 2291 |› }
```

```
twitter["UserData", "Username" → "elonmusk"]
```

```
{ ‹| ID → 44 196 397, Name → Elon Musk, ScreenName → elonmusk, Location → Boring,
   FollowersCount → 14 343 253, FriendsCount → 47, FavouritesCount → 646 |› }
```

grab tweet strings and delete stop words

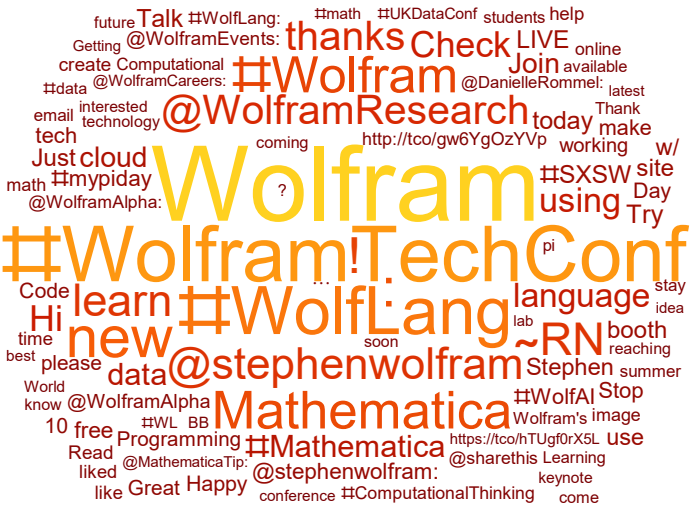
Next, we build a function to to grab strings of tweets. Once we have this function built, we can input any user name to save time in future queries. We use DeleteStopWords[] to get rid of punctuation and the common Twitter parlance for re-tweet, RT.

```
getTweetStrings[username_, number_: 4000] := Block[{text, tweetsWRI},
  tweetsWRI = twitter["TweetList", "Username" → username, MaxItems → number];
  text = Flatten[StringSplit[Normal[tweetsWRI[All, "Text"]]], 1];
  StringDelete[DeleteStopwords[text],
   {"&amp", ";", ".", ",", "-", "_", "\"", "(", ")", "RT"}]]
```

create a word cloud

We can now easily visualize the most common words in Wolfram Research's tweets using WordCloud. We style our word cloud using ColorFunction.

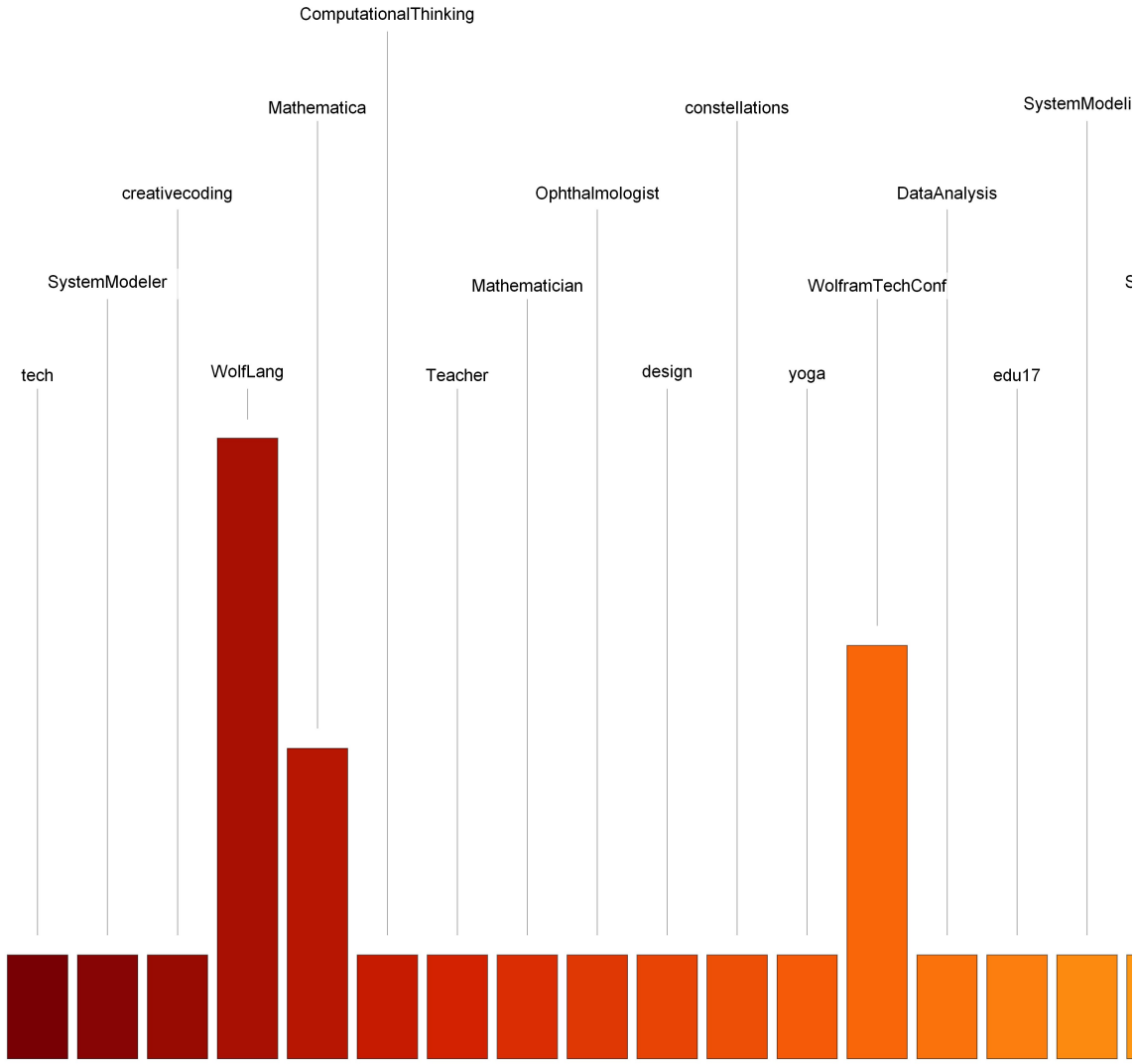
```
WordCloud[getTweetStrings["wolframresearch"],
 ColorFunction → ColorData["SolarColors"]]
```



examine hashtags

We can also examine hashtags, a popular way of tagging and/or expressing tone, topic, or sentiment in tweets. We first grab hashtags used by Wolfram Research on Twitter then tally them to create a chart.

```
wriHash = twitter["UserHashtags", "Username" → "wolframresearch", MaxItems → 35];
talliedHash = Tally[wriHash];
BarChart[Apply[Callout, Reverse[talliedHash, 2], {1}],
 ChartStyle → "SolarColors", Axes → False, ImageSize → Large]
```

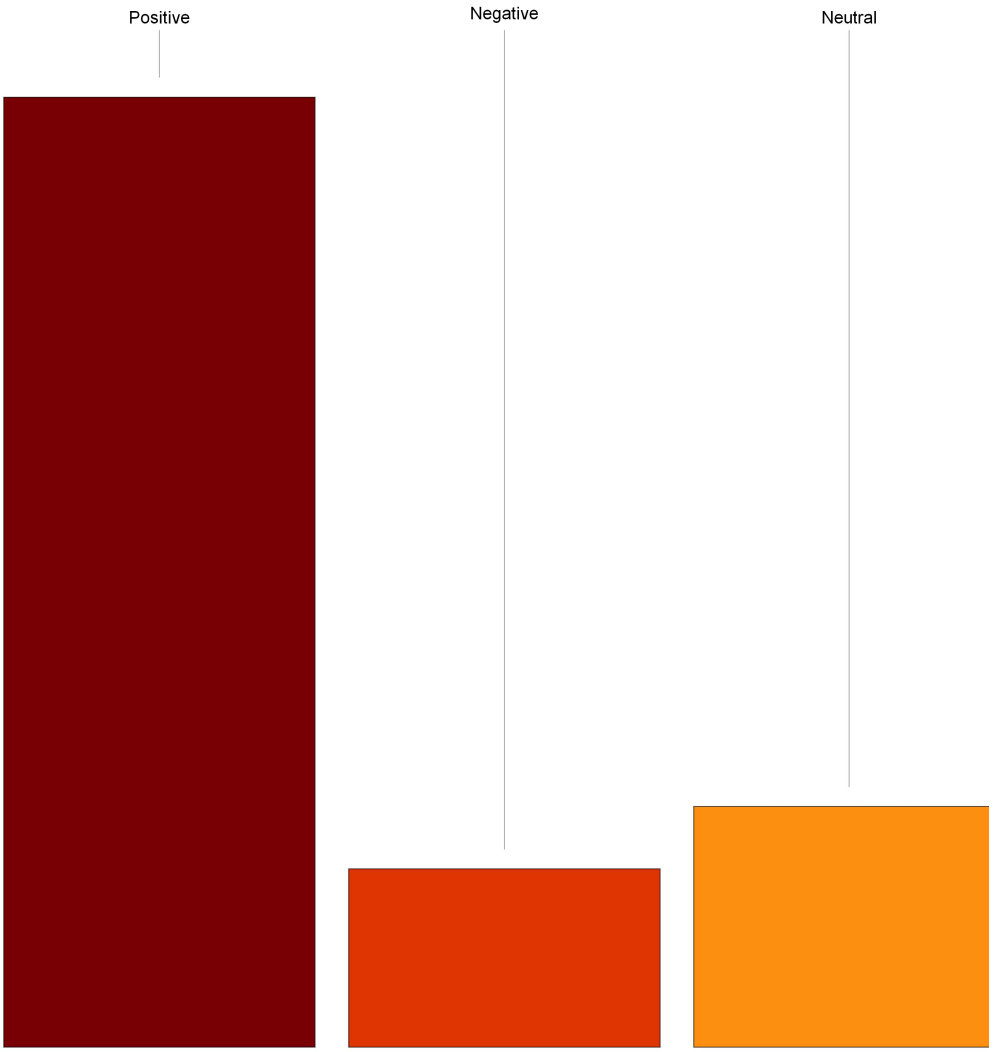


Use machine learning for sentiment and topic analysis

We can again use a simple and easy to use sentiment classifier built-in to the Wolfram Language.

Use classify for sentiment analysis

```
wriTweetsStrings = getTweetStrings["wolframresearch"];
cSentTwit = Tally@Classify["Sentiment", wriTweetsStrings];
BarChart[Apply[Callout, Reverse[cSentTwit, 2], {1}],
 Axes -> False, ChartStyle → "SolarColors", ImageSize → Large]
```



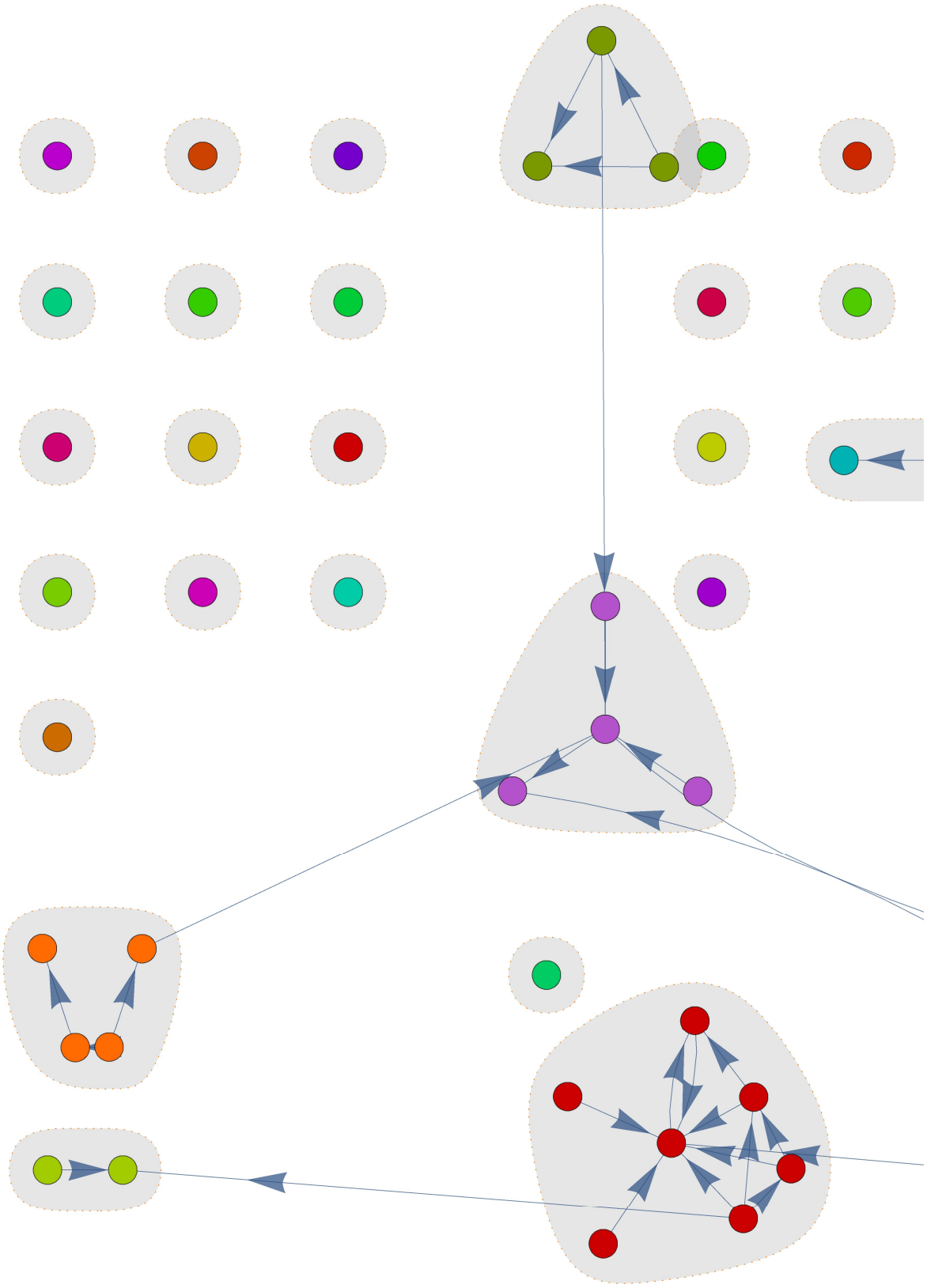
examine follower mentions and twitter activity

Wolfram Language has many built-in functions to collect an analyze tweets.

examine a follower mention network

We can use the Twitter API to look at how many of Wolfram Research's followers mention one another on the platform. We will use these data to then build a graph. Since we again used ToolTip, we can hover over each vertex to view the username of the Twitter user. Arrows indicate what account each user is mentioning in the network.

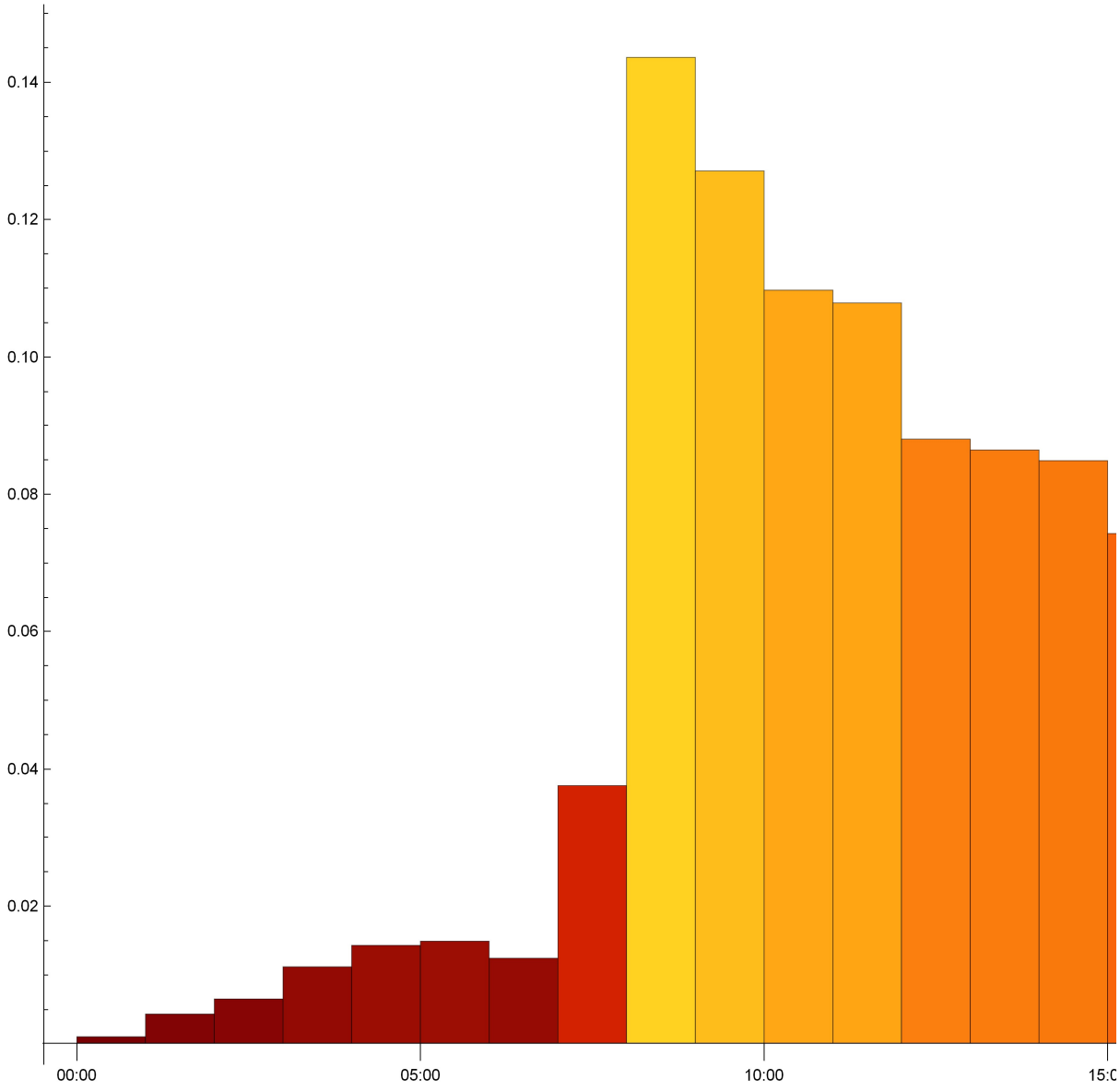
```
With[{mentionGraph = twitter["FriendMentionNetwork", "Username" → "wolframresearch"]},
 CommunityGraphPlot[#, FindGraphCommunities[#], Method → "Hierarchical",
    CommunityRegionStyle → Directive[Opacity[.2], Gray], CommunityBoundaryStyle →
     Directive[Orange, Dotted], ImageSize → Large] &@mentionGraph]
```



examining twitter activity

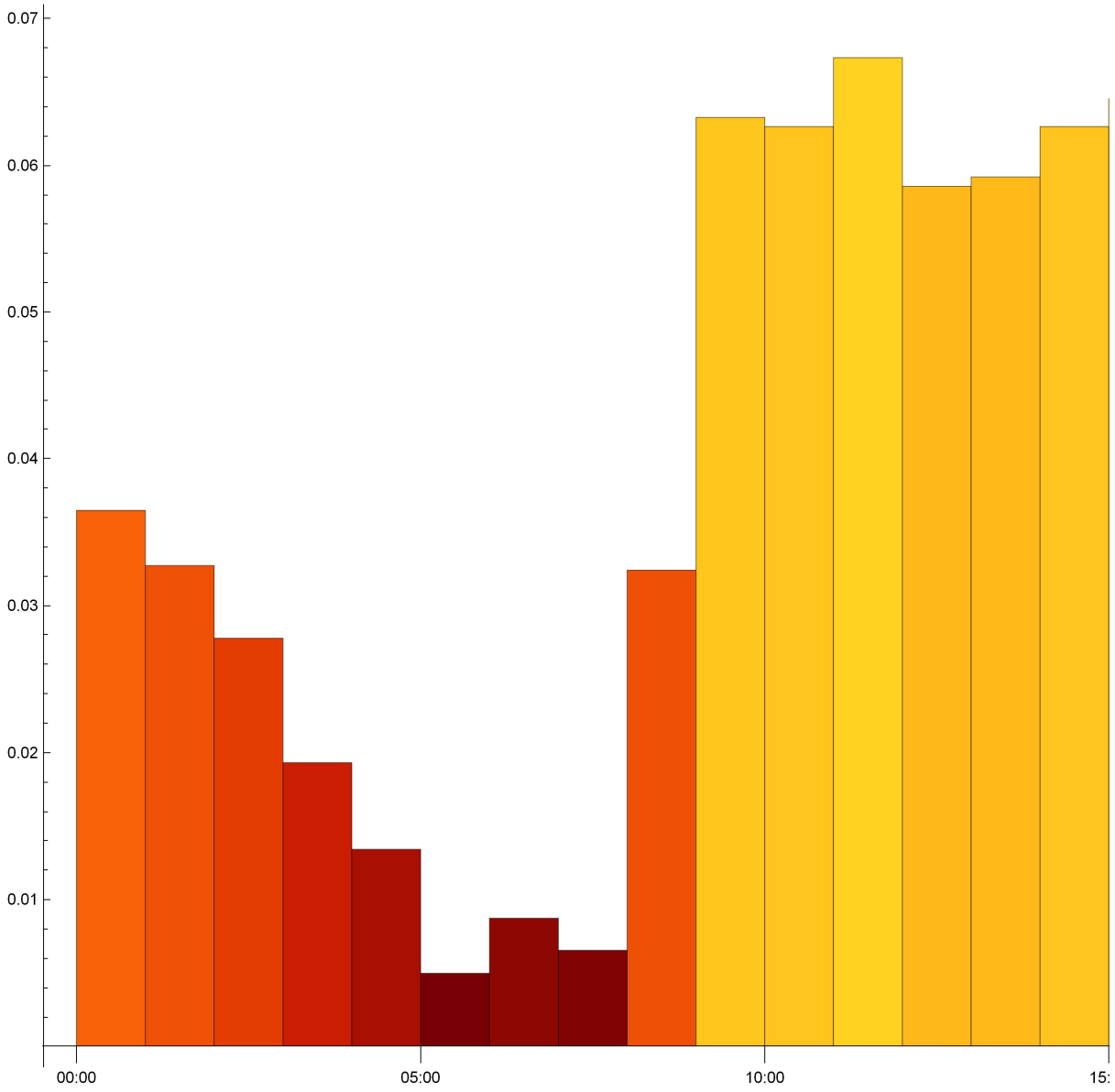
We now grab tweets are a series of events to visualize the frequencies of tweets.

```
wriEventseries =
  twitter["TweetEventSeries", "Username" → "wolframresearch", MaxItems → 5000];
hourlypercents = DateHistogram[wriEventseries["Dates"], "Hour", "Probability",
  DateReduction → "Day", ColorFunction → "SolarColors", ImageSize → Large]
```



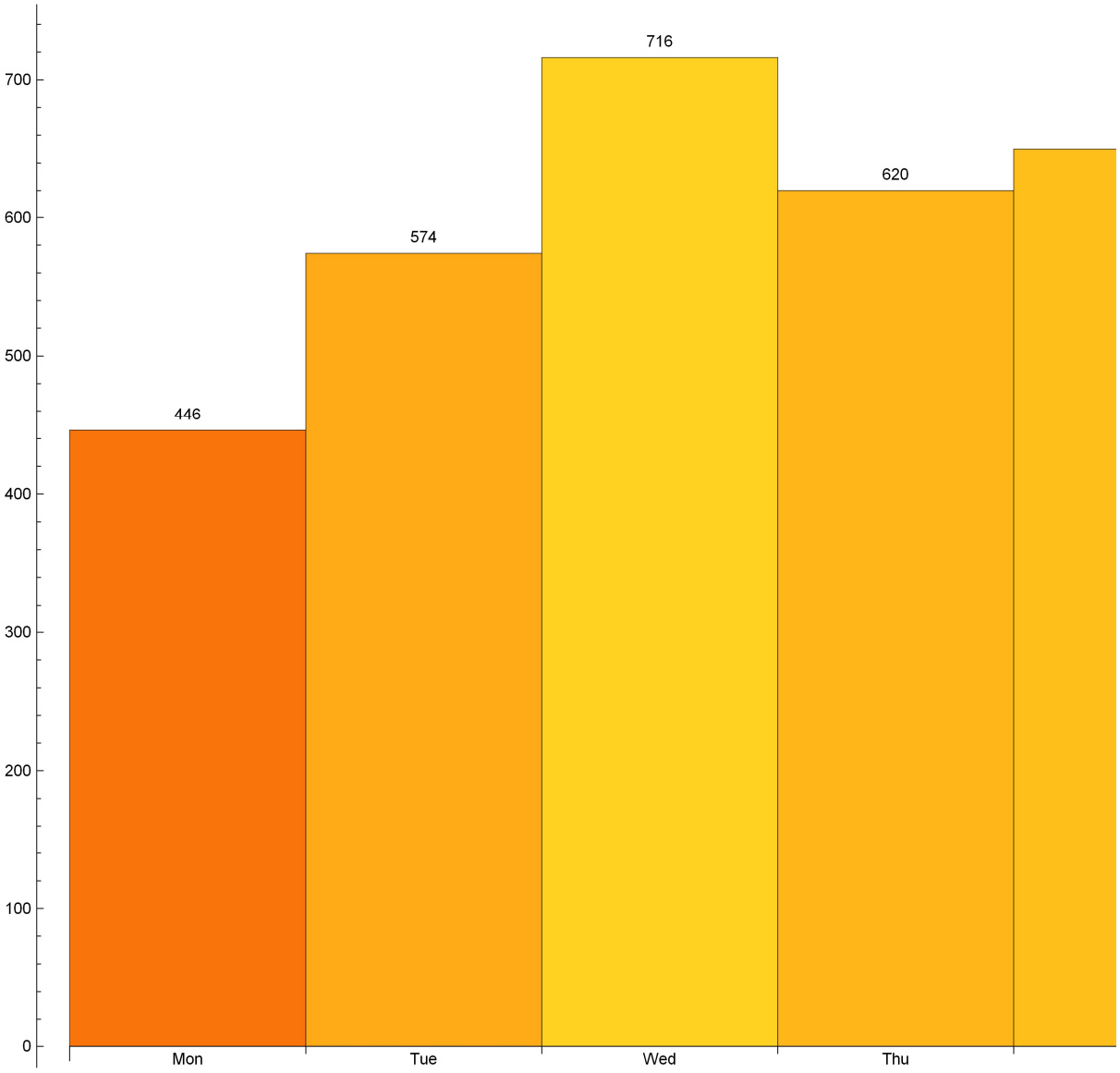
We can see the most common time of tweets for Wolfram Research. We can also check other accounts. We'll look at Elon Musk as an example.

```
muskEventseries = twitter["TweetEventSeries", "Username" → "elonmusk", MaxItems → 5000];
hourlypercents = DateHistogram[muskEventseries["Dates"], "Hour", "Probability",
  DateReduction → "Day", ColorFunction → "SolarColors", ImageSize → Large]
```



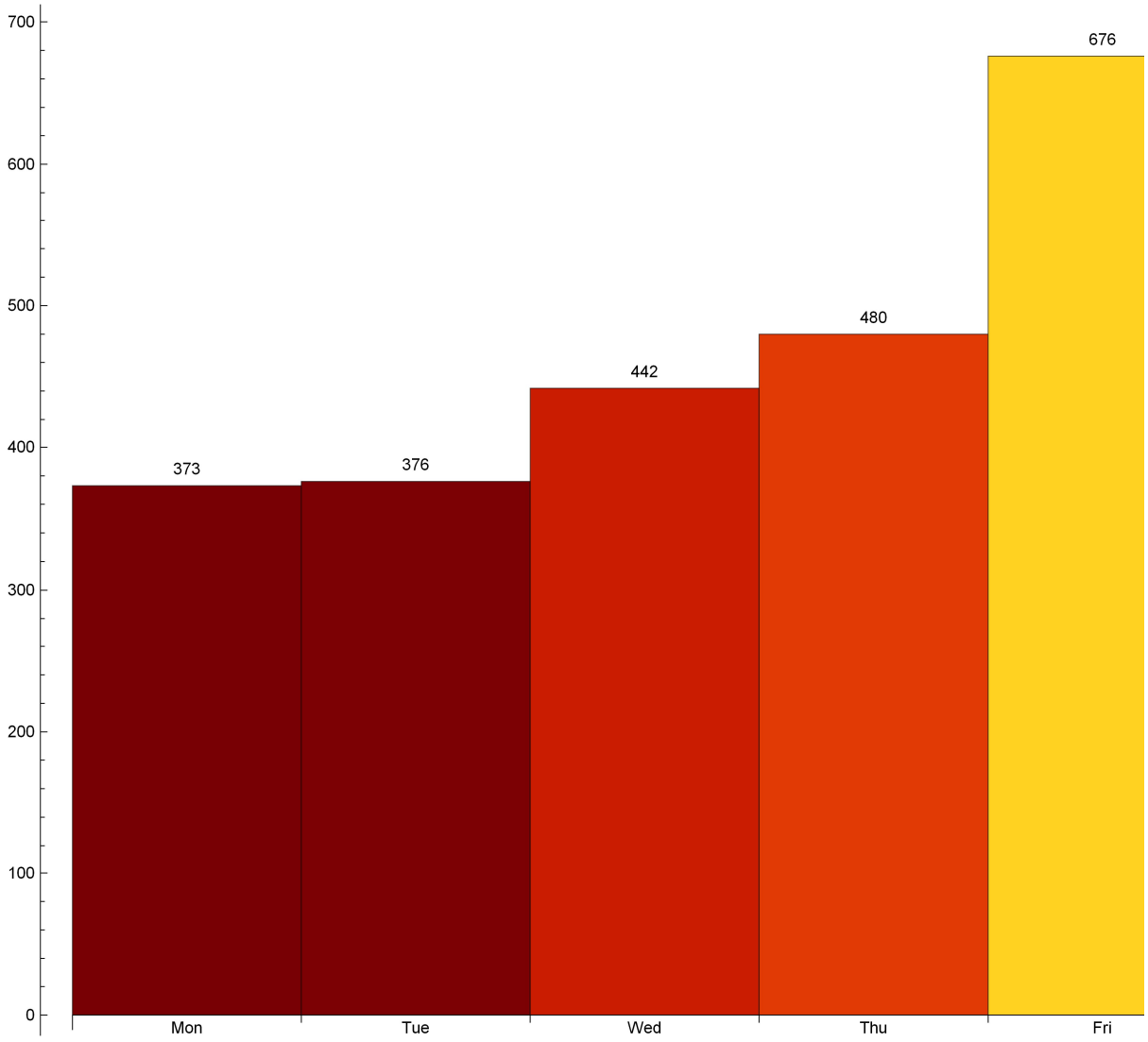
We can then look at the frequency of tweets by days of the week for Wolfram Research.

```
WRIweekcounts = DateHistogram[wriEventseries["Dates"], "Day", DateReduction → "Week",
   ColorFunction → "SolarColors", LabelingFunction → Above, ImageSize → Large]
```



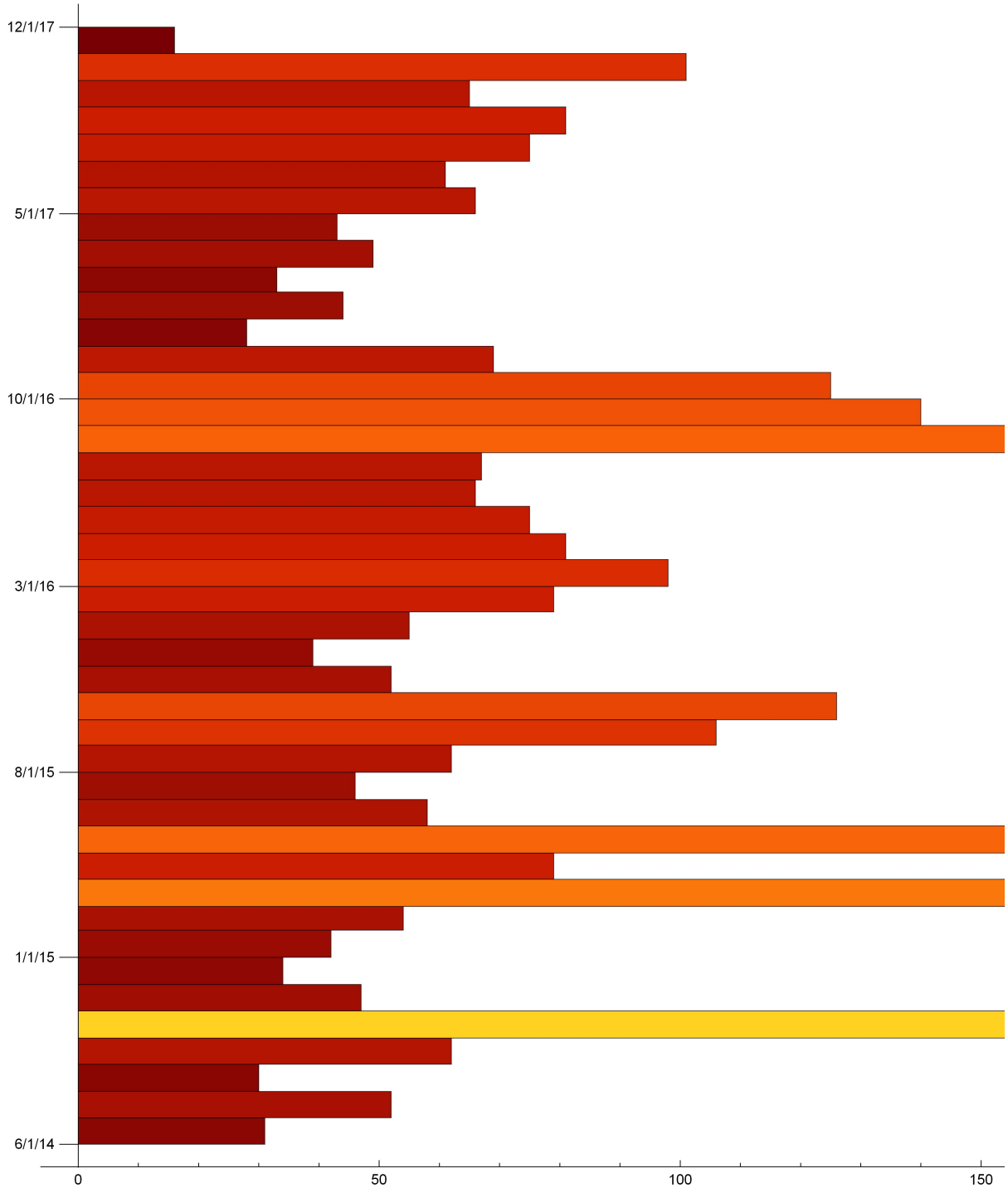
And now the same for Elon Musk.

```
muskTweetcounts = DateHistogram[muskEventseries["Dates"], "Day", DateReduction → "Week",
  ColorFunction → "SolarColors", LabelingFunction → Above, ImageSize → Large]
```



Interesting, Elon Musk is much more likely to tweet on Friday than other days of the week. Next, we can break down tweets by all time, going back to the earliest tweets we previously grabbed, first for Wolfram Research.

```
WRIalltimecounts = DateHistogram[wriEventseries["Dates"], "Month",
  ColorFunction → "SolarColors", BarOrigin → Left, ImageSize → Large]
```



examining twitter topics

As a further example, we can examine #machinelearning on Twitter. We'll first search for his Twitter handle.

```
search = twitter["TweetSearch", "Query" → "#machinelearning",
   MaxItems → 5000, "Language" → Entity["Language", "English"]];
```

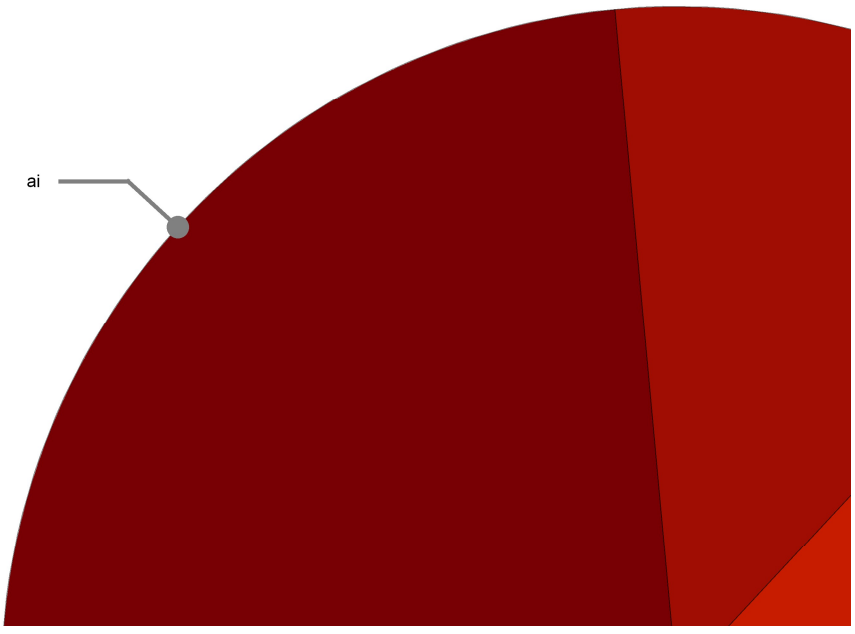
Once we have these 5,000 tweets, we can then create a function to look at the most significant

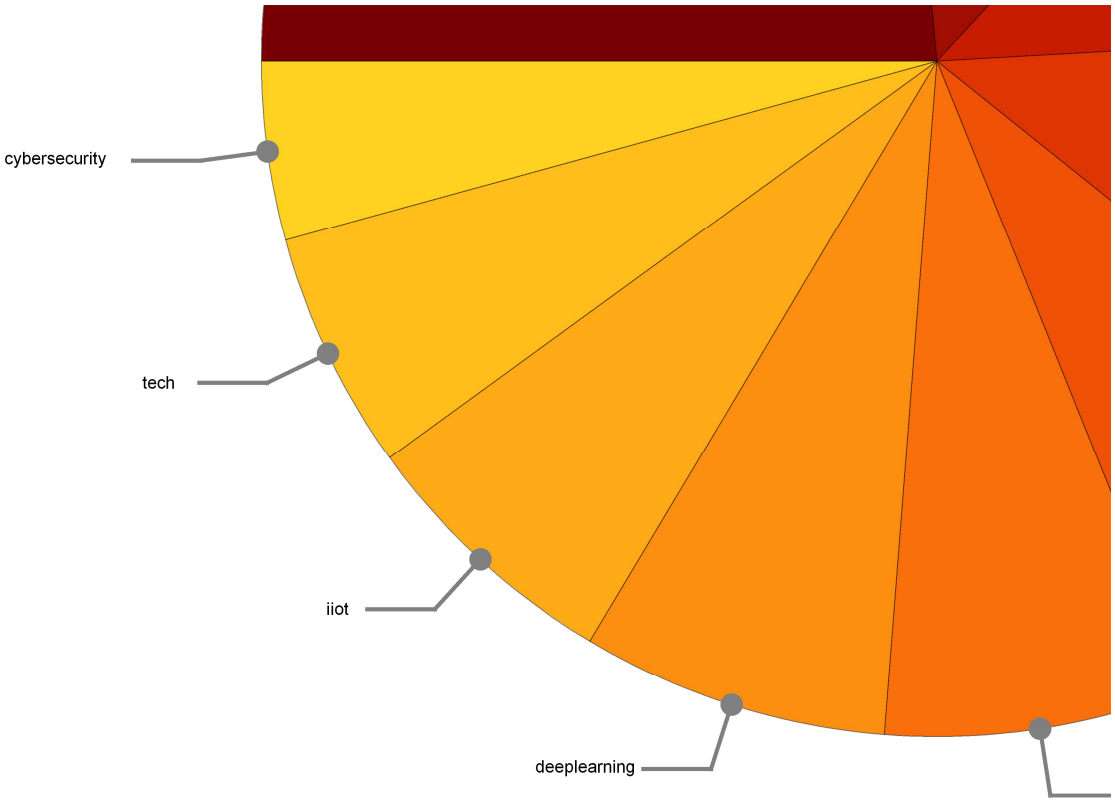
words people have used in relations to Stephen Wolfram's Twitter handle.

```
FilterTweets[tweets_] := StringReplace[ToLowerCase@tweets,
  {RegularExpression["[http]*[s]*[:][\\/\\/][a-z0-9.\\/]+[ ]*"] → " ",
   RegularExpression["[rt: ]*[rt ]*[\\@][a-z0-9]*[:]*"] → " ",
   RegularExpression["[^a-z\\s:]"] | ":" → " "}];
tweets = FilterTweets[#["Text"]] & /@ (Normal@search);
processedTweets = DeleteCases[#, ""] & /@ (Function[{wordList},
     (If[Classify["Profanity", wordList], "", wordList]), Listable][
    DeleteCases[DeleteDuplicates@(DeleteStopwords@ToLowerCase@TextWords[#]),
       "h" | "ht" | "htt" | "http" | "https" | "machine" |
        "learning" | "machinelearning"] & /@ tweets]);
significantWords = Take[(Reverse@(SortBy[Tally@(# & /@ (Flatten[(Tally@# & /@
                processedTweets), 1])) /. {{w_, _}, c_} → {w, c}, Last])), 10];
topWordsQuantities = N[#[[2]] / 1000] & /@ significantWords;
topWordsNames = First@# & /@ significantWords;
```

Now that we have our text processed to get rid of URL links and profanity, we can create a pie chart with these words.

```
pieChart = PieChart[# & /@ topWordsQuantities,
  ChartLabels → Placed[topWordsNames, "RadialCallout"],
  ChartStyle → "SolarColors", LabelingFunction →
   (Row[{NumberForm[100 N[# / Total[topWordsQuantities]],
        {3, 2}], "%"}] &), ImageSize → Large]
```





example 3: social networks and information diffusion

Next, we examine social network analysis with a brief simulation of information diffusion using an epidemiological model. We begin with data on people who attend tech conferences. These data are an association matrix in CSV format indicating if each individual in the matrix has attended nine different conferences.

wrangling data

We begin by importing the data using Import to grab a CSV file of trade show attendance.

import data to check structure and formatting

```
csv = Import["/Users/swedew/Dropbox (Wolfram)/WTC/Talk/ConferenceData.csv"];
```

We can then check the dimensions of our data to check its structure.

```
Dimensions@%
```

```
{264, 10}
```

We can see that we have 10 columns and 264 rows. Let's not take a quick look at how the data are organized by looking at the first four entries.

```
csv[[ ;; 4]]
```

```
{{, Strata, SIGGRAPH, JSM, Disrupt, CES, Techweek, JMM, Collision, SXSW},
 {Applegate.Sean, 0, 0, 1, 1, 0, 0, 0, 0, 0},
 {Adams.Jeffrey, 0, 0, 1, 1, 0, 1, 1, 1, 0}, {Allen.Darien, 0, 0, 1, 0, 0, 0, 0, 0, 0}}
```

We can see that we have nine conferences and the first four individuals in our dataset. Let's change our 1 values to True and 0 to False and give column one a header name.

```
csv1 = csv /. {0 → False, 1 → True};
assoc0 = With[{header = (csv1[[1]] /. "" → "Name")},
   AssociationThread[header → #] & /@ Rest@csv1];
```

And now let's check to see what our data now look like.

```
assoc0[[ ;; 3]]
```

```
{⟨|Name → Applegate.Sean, Strata → False, SIGGRAPH → False, JSM → True, Disrupt → True,
  CES → False, Techweek → False, JMM → False, Collision → False, SXSW → False|⟩,
 ⟨|Name → Adams.Jeffrey, Strata → False, SIGGRAPH → False, JSM → True, Disrupt → True,
  CES → False, Techweek → True, JMM → True, Collision → True, SXSW → False|⟩,
 ⟨|Name → Allen.Darien, Strata → False, SIGGRAPH → False, JSM → True, Disrupt → False,
  CES → False, Techweek → False, JMM → False, Collision → False, SXSW → False|⟩}
```

We now manipulate our data so that the names of the individuals are in First, Last format and remove the period. We'll also create a data object out of it.

```
assoc1 = Module[{rec = #},
     rec["Name"] = StringRiffle[Reverse[StringSplit[rec["Name"], "."]]];
     rec] & /@ assoc0;
ds = Dataset[assoc1]
```

Name	Strata	SIGGRAPH	JSM	Disrupt	CES	Techweek
Sean Applegate	False	False	True	True	False	False
Jeffrey Adams	False	False	True	True	False	True
Darien Allen	False	False	True	False	False	False
Dale Ashbury	False	False	True	False	False	True
Steve Allerton	True	False	False	False	False	False
Jody Ackerman	False	False	False	False	False	False
Chris Ambrose	False	False	False	False	False	False
Jane Abrams	False	True	False	False	False	False
Connor Baines	False	False	False	False	False	False
Henry Ballard	False	False	True	False	False	False
John Barry	False	False	True	False	True	True
Jacques Bernard	False	False	False	False	True	False
James Batchelor	True	False	False	False	False	False
Jed Bates	False	True	True	False	True	False
Cole Bryce	True	False	False	False	False	False
James Blake	True	False	False	False	False	False
Jonathan Broome	False	False	True	False	False	False
Lester Bolton	False	False	False	False	True	False
Katherine Bratton	False	False	False	False	False	False
Bryce Bricknell	False	False	False	False	False	True

showing 1–20 of 263

Now let's create a variable for our conference attendee names and look at one of the individuals in our data object.

```
names = Normal@ds[All, "Name"];
ds@SelectFirst[#["Name"] == "Sean Applegate" &]
```

Name	Sean Applegate
Strata	False
SIGGRAPH	False
JSM	True
Disrupt	True
CES	False
Techweek	False
JMM	False
Collision	False
SXSW	False

So we can see after some very brief data manipulation that Sean Applegate attended JSM and Disrupt.

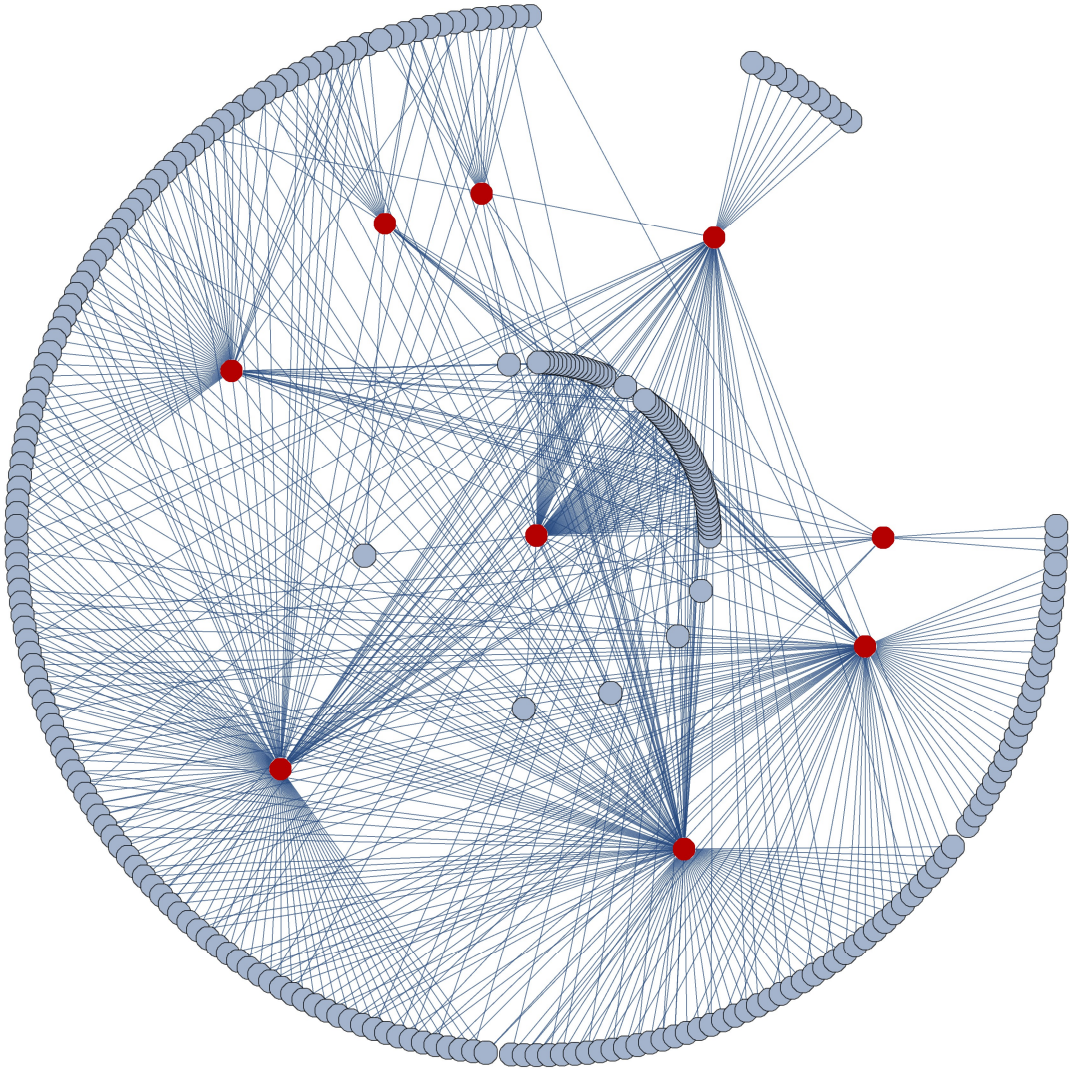
Social network analysis at the macro and micro levels

Let’s start at the conference level with visualizations before drilling down to the individual level by first defining conferences to build network graphs. We

a look at the conference level

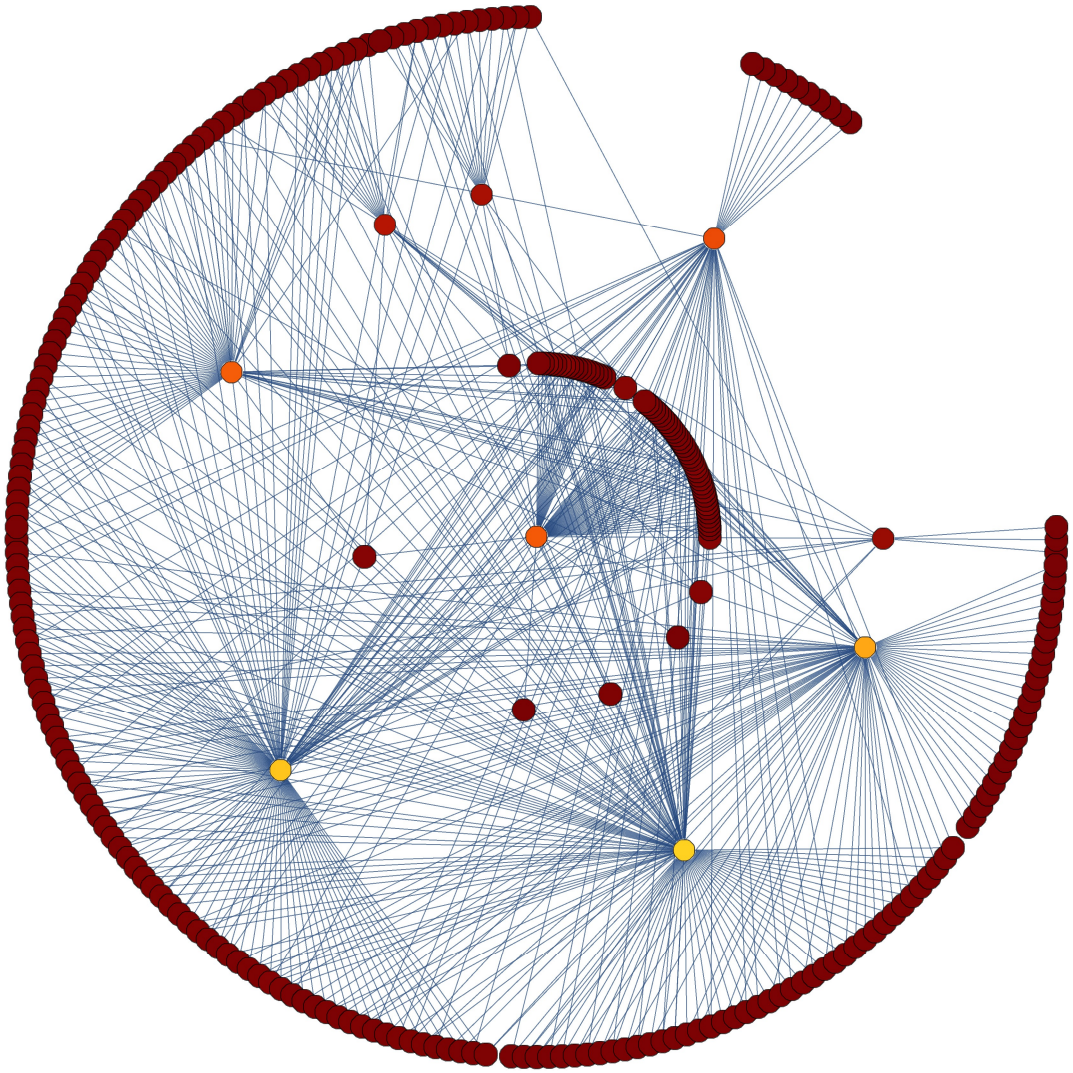
We’ll also use Tooltip so we can hover over our conference vertices that we are also styled through the HighlightGraph function.

```
conferences = Normal@Rest[Keys[First[ds]]];
cg = HighlightGraph[Flatten[Normal[With[{row = #, name = #Name},
        If[row[#], name ⟞ #, Nothing] & /@ conferences] & /@ ds]],
  conferences, VertexLabels → Placed["Name", Tooltip], ImageSize → Large,
  VertexLabelStyle → Background → White,  VertexSize → 5, GraphLayout → "RadialDrawing"]
```



We can now use more styling functions to show us which conferences have the most attendees. To do so, we will create a function based on vertex degrees, or the number of connections the conferences have. We then use ColorData[] to visualize which vertices have the highest degree count.

```
HighlighVertexDegree[g_, vd_] := HighlightGraph[g, Table[Style[VertexList[g][[i]],
      ColorData["SolarColors"][vd[[i]] / Max[vd]]], {i, VertexCount[g]}]];
vd = VertexDegree[cg];
HighlighVertexDegree[cg, vd]
```



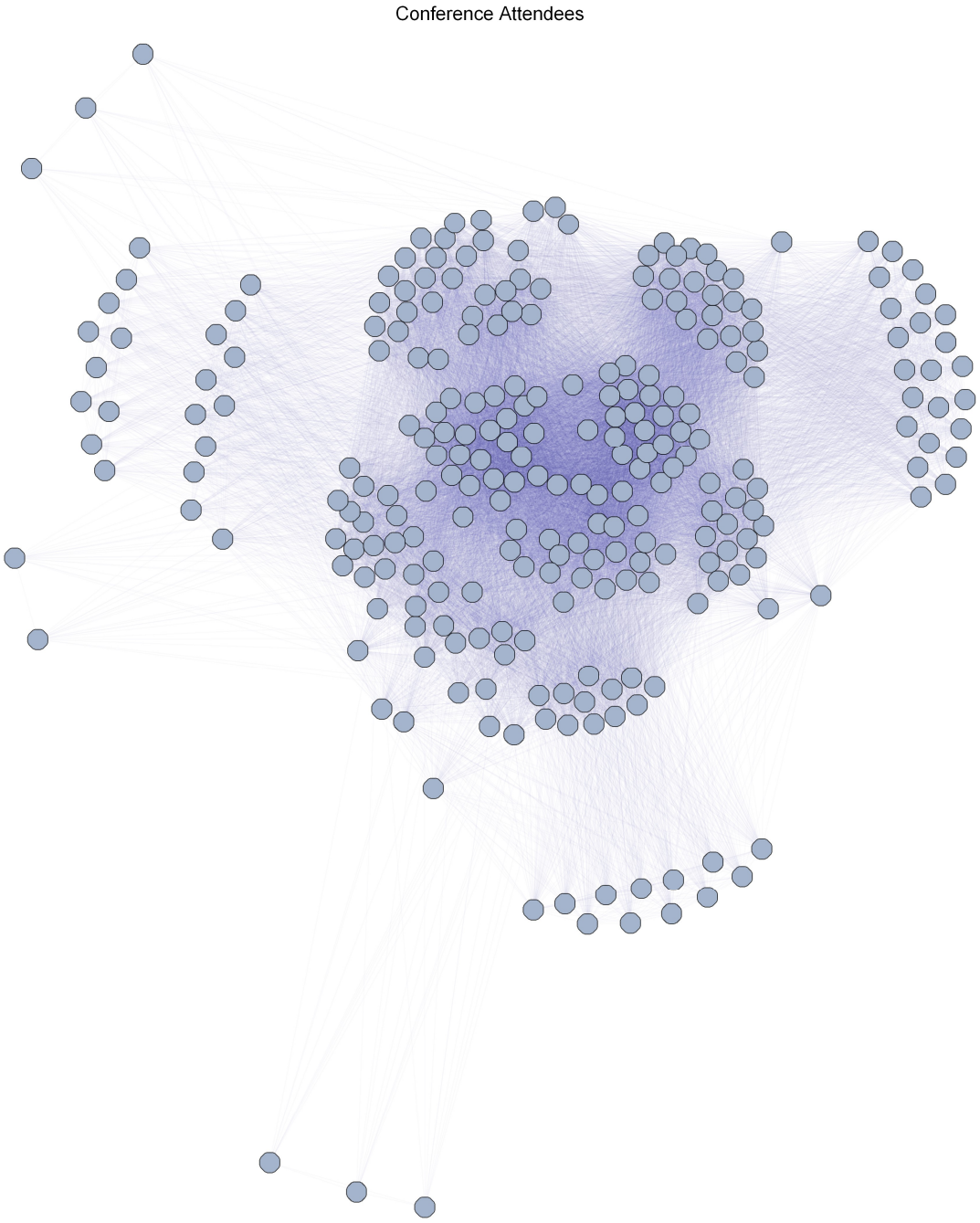
a look at the individual level

Now that we have an idea that our most attended conferences are Collision and SXSW, let's drill down to which individuals we might want to target with some sort of information with the hopes that it may spread throughout some of these conferences.

create a bipartite graph

We first organize our data into a bipartite adjacency matrix at the individual level. With this, we are looking at the potential of individuals having met at conferences.

```
bipartiteAdjacencyMatrix = Boole@Normal[ds[Values, Rest]];
edges = ReplacePart[
    bipartiteAdjacencyMatrix.Transpose[bipartiteAdjacencyMatrix], {i_, i_} → 0];
personLevelGraph = AdjacencyGraph[names, edges,
  EdgeStyle → {Opacity[.02]}, ImageSize → Large, VertexSize → Automatic,
  VertexLabels → Placed["Name", Tooltip], PlotLabel → "Conference Attendees"]
```



We have used Tooltip once again so we can hover over each vertex to get the conference attendees name. While we can see some clear clusters and outliers here, we can use some built-in Wolfram Language algorithms and graph measurements to highlight central actors in out network.

centrality measurements

We begin by creating a function that will highlight a graph with a color function by each centrality measure that we are going to use, Betweenness Centrality, Closeness Centrality, and Eigenvector Centrality.

```
HighlightCentrality[g_, cc_] := HighlightGraph[g, Table[Style[VertexList[g][[i]],
    ColorData["SolarColors"][cc[[i]] / Max[cc]]], {i, VertexCount[g]}]]
```

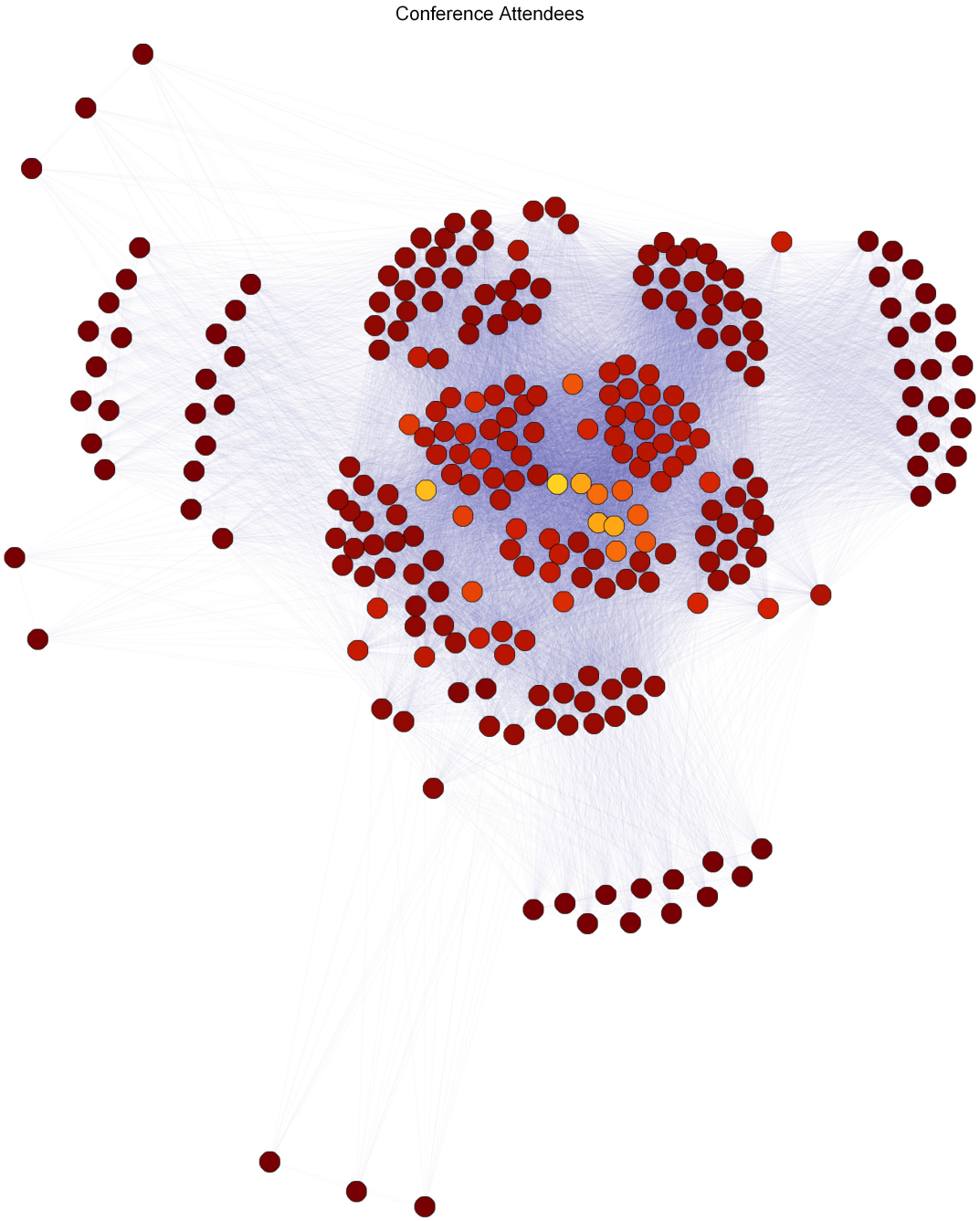
We'll also go ahead a design a function to make tables of centrality measurements that we will use. This will give us the top five central actors in the network by each measurement we use.

```
TopFive[measure_, heading_] :=
 TableForm[TakeLargestBy[Transpose[{names, measure}], #[[2]] &, 5],
  TableHeadings → {None, {"Conference Attendee", heading}}]
```

betweenness centrality

Since we have previously defined our function, we simply plug in BetweennessCentrality. We have also previously defined our Tooltip function, so we can mouse over each vertex to see the name of the conference attendee.

```
HighlightCentrality[personLevelGraph, BetweennessCentrality[personLevelGraph]]
```



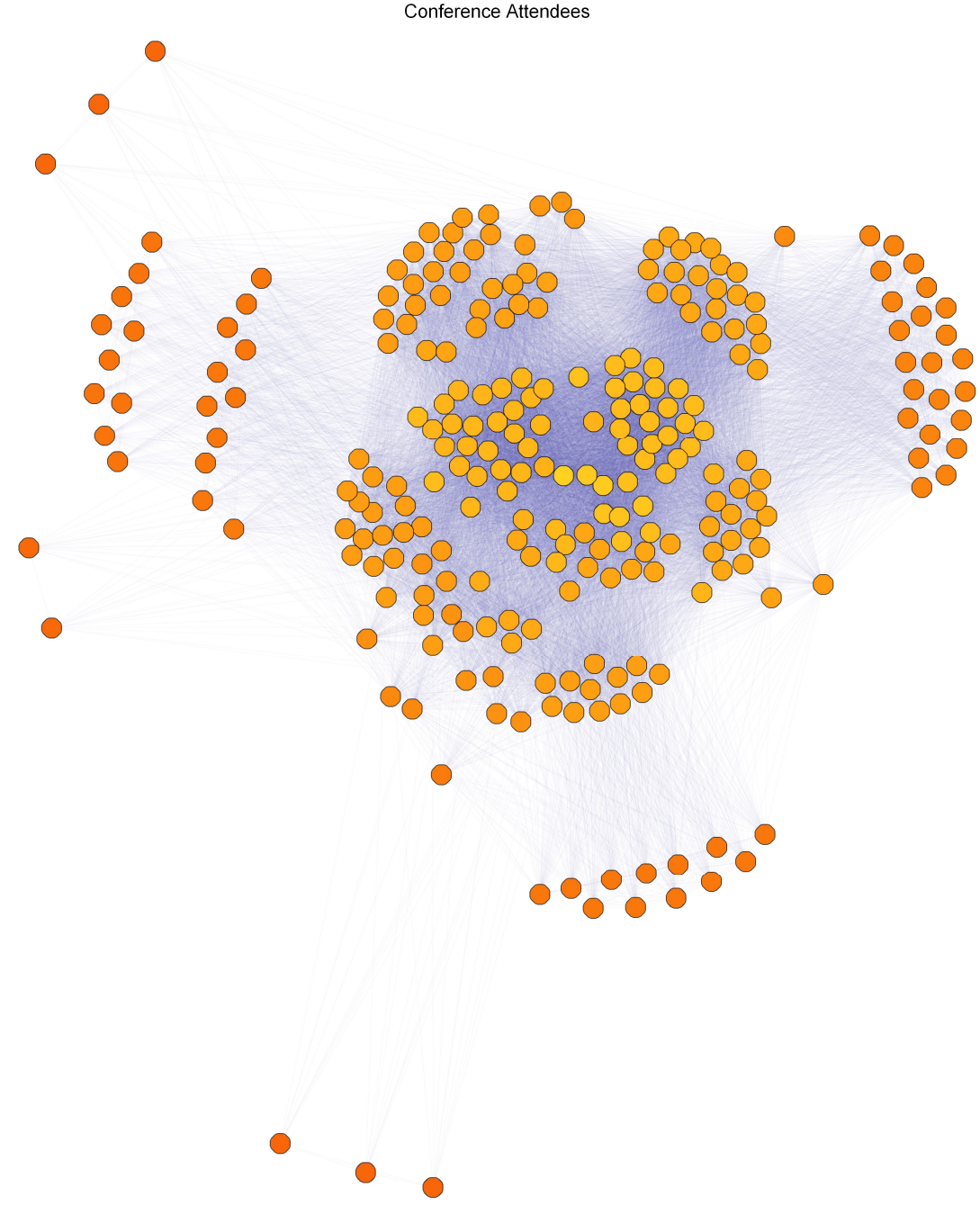
And now we can make our table.

```
betweenness = BetweennessCentrality[personLevelGraph];
TopFive[betweenness, "Betweenness Centrality"]
```

Conference Attendee	Betweenness Centrality
John Whitney	479.701
Christopher Willis	422.175
Jed Bates	368.088
Ryan Chase	368.088
Zachary Griffity	361.846

closeness centrality

```
HighlightCentrality[personLevelGraph, ClosenessCentrality[personLevelGraph]]
```



And we then create another chart.

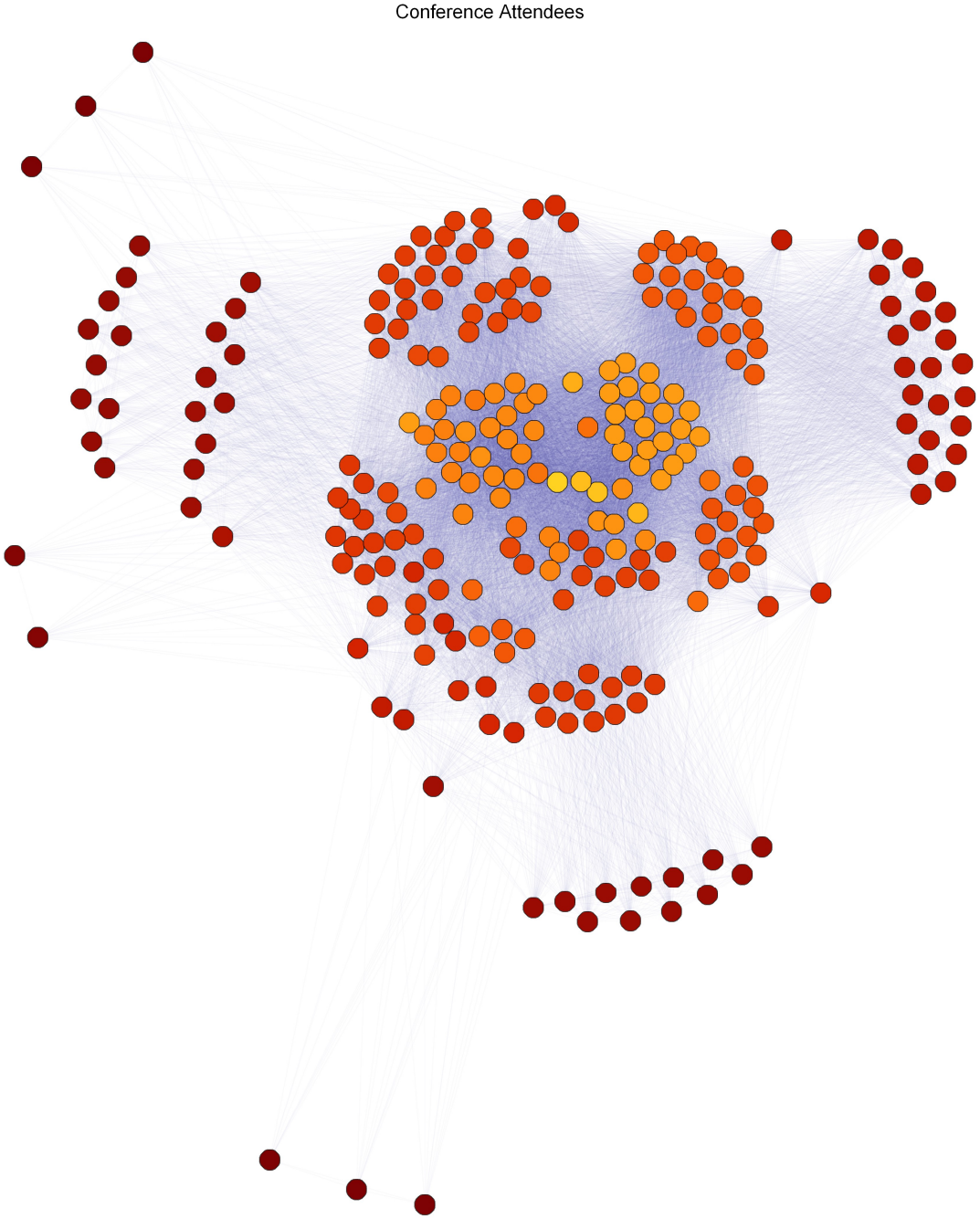
```
closeness = ClosenessCentrality[personLevelGraph];
TopFive[closeness, "ClosenessCentrality"]
```

Conference Attendee	ClosenessCentrality
John Whitney	0.97037
Richard Smith	0.935714
Zachary Griffity	0.932384
Chad Youngworth	0.909722
Ryan Chase	0.894198

eigenvector centrality

And lastly, we look at eigenvector centrality.

```
HighlightCentrality[personLevelGraph, EigenvectorCentrality[personLevelGraph]]
```



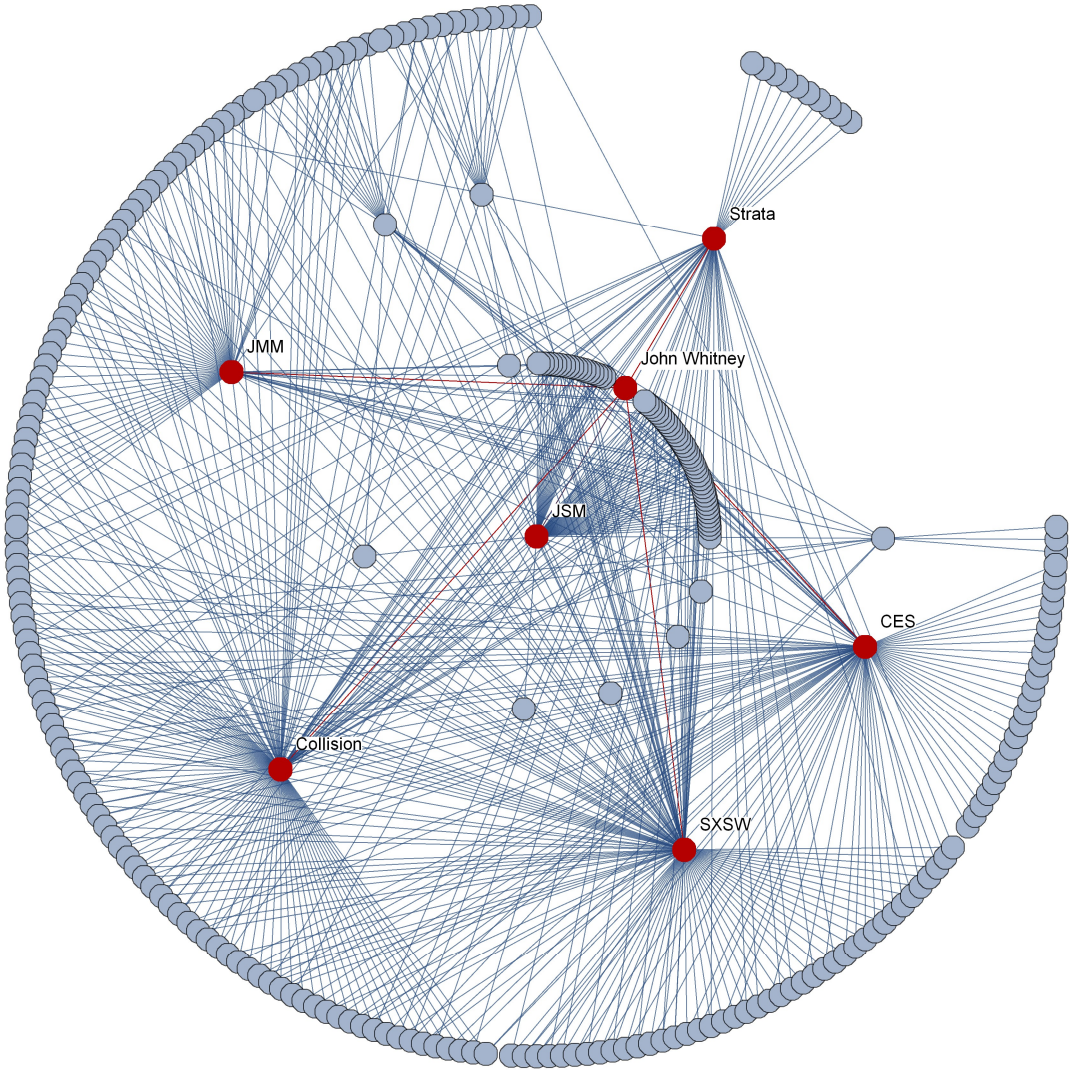

```
eigenvectorCentrality = EigenvectorCentrality[personLevelGraph];
TopFive[eigenvectorCentrality, "Eigenvector Centrality"]
```

Conference Attendee	Eigenvector Centrality
John Whitney	0.00964548
Richard Smith	0.00888356
Zachary Griffity	0.00860165
Chad Youngworth	0.00816193
Jessica Potsworth	0.00771823

We see that John Whitney is repeatedly at the top of our centrality measures. Next, let's create a graph to see how he relates to each conference. We'll use the radial drawing graph layout function

for a more intuitive visualization.

```
With[{g = Flatten[Normal[With[{row = #, name = #Name},
         If[row[#], name ⟼ #, Nothing] & /@ conferences] & /@ ds]]},
  HighlightGraph[g,
   NeighborhoodGraph[g, "John Whitney", 1, VertexLabels → Automatic],
   GraphLayout → "RadialDrawing", VertexLabels → Automatic,
   VertexLabelStyle → Background -> White, ImageSize → Large]]
```



simulating information diffusion

We can now simulate how information might spread through a network using a probabilistic model. We can think of this as an epidemiological model where an “infected” node has a coin toss probability of sharing that information with others. We can, of course, adjust our probabilities and apply this to our actual social network.

infection function

```
infect[neighbors_List, p_] :=
  Pick[neighbors, RandomChoice[{p, 1 - p} → {1, 0}, Length[neighbors]], 1];
step[graph_, {recentlyInfected_, infectedLongAgo_}, p_] :=
  {Complement[
    infect[AdjacencyList[graph, recentlyInfected], p], recentlyInfected,
    infectedLongAgo], Join[infectedLongAgo, recentlyInfected]};
```

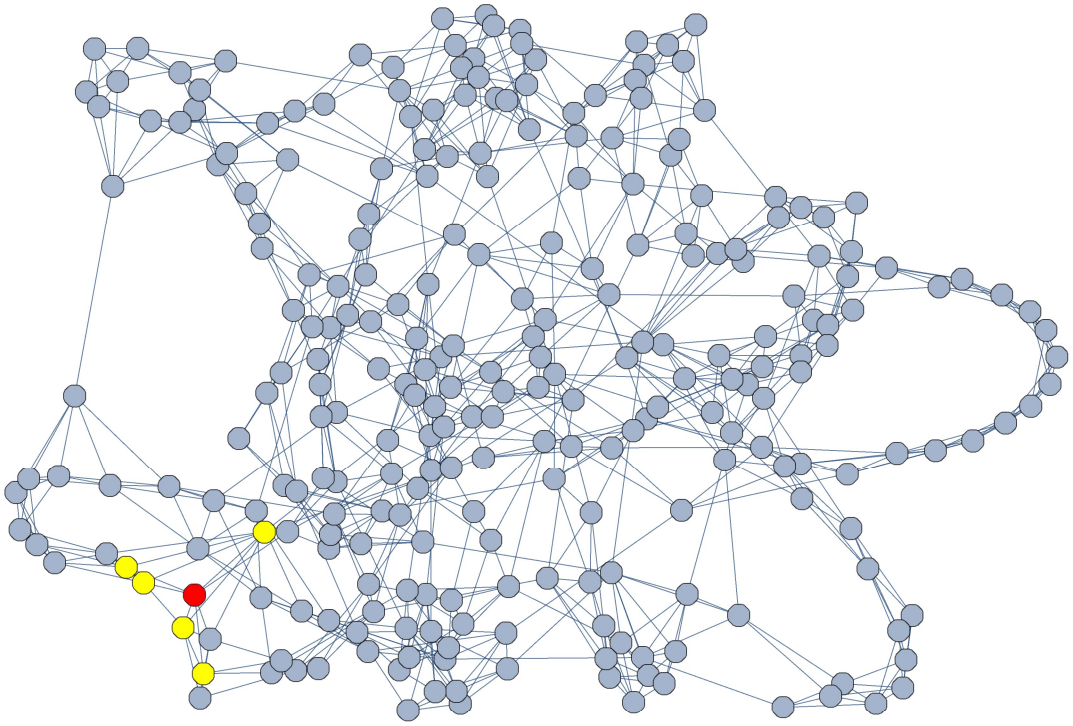
define a small world network and visualize steps of infection

```
network = RandomGraph[WattsStrogatzGraphDistribution[264, 0.1, 3], ImageSize → Large];
```

```
step[network, {{1}, {}}, .5]
```

```
{{2, 3, 4, 262, 264}, {1}}
```

```
HighlightGraph[Graph[network, VertexSize → 2],
 {Style[1, Red]} ~ Join ~ (Style[#, Yellow] & /@ {2, 3, 4, 262, 264})]
```

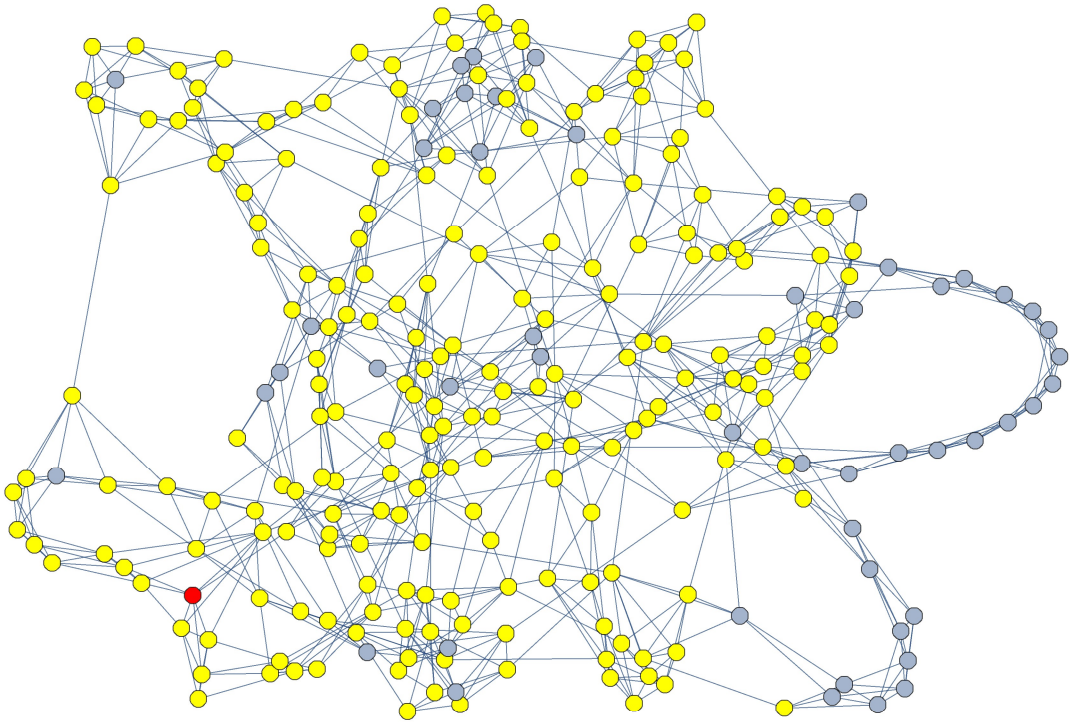


We can then visualize this further through building a list of people who become infected through interaction in our network.

```
infected[g_, i_, p_] := Last[FixedPoint[step[g, #, p] &, {i, {}}]]
```

```
infectedList = infected[network, {1} , 0.4] ;
```

```
HighlightGraph[Graph[network, VertexSize → 1.5],
 {Style[1, Red]} ~Join~ (Style[#, Yellow]
     & /@ Complement[infectedList, {1}])]
```



And finally, we can visualize this interactively using Manipulate.

```
epidemicHistory = FixedPointList[step[network, #, .3] &, {{1}, {}}];
Manipulate[
 Column[{HighlightGraph[
       Graph[network, VertexSize → 3], Join[Style[#, Red] & /@ #[[1]],
        Style[#, Yellow] & /@ #[[2]],
        Style[#, {Red, Thick}] & /@ EdgeList@Subgraph[network, x_ ⟷ y_ /;
             MemberQ[#[[1]], x] && MemberQ[#[[2]], y]]]]
     &@@epidemicHistory[[index]],
   ListPlot[Length[Join @@ #] & /@ Take[epidemicHistory, index],
    AspectRatio → 1 / 3, Filling → Axis, AxesLabel → {"Time", "Spread"}, AxesOrigin →
     {0, 0}, Ticks → {Range[index], Range[0, Length[Join @@ epidemicHistory[[-1, -1]]],
       Max[1, IntegerPart[Length[Join @@ epidemicHistory[[-1, -1]]] / 5]]]},
    ImageSize → Large]}], {{index, 1, "Iteration Number"}, 1,
  Length[epidemicHistory], 1}]
```

Iteration Number

```
HighlightGraph[Graph[network, VertexSize → 3], Join[epidemicHistory, 1,
  EdgeList[Subgraph[network, x_ ⟷ y_ /; MemberQ[epidemicHistory[[1]][[1]], x] &&
      MemberQ[epidemicHistory[[1]][[2]], y]]]]
ListPlot[Take[0, 0], AspectRatio → 1/3, Filling → Axis, AxesLabel → {Time, Spread},
 AxesOrigin → {0, 0}, Ticks → {{1}, {0, 1, 2, 3}}, ImageSize → Large]
```

Updating from Wolfram Research server (100%)

Part: Part specification epidemicHistory[[1]] is longer than depth of object.

HighlightGraph: A graph object is expected at position 1 in HighlightGraph[Graph[network, VertexSize → 3], Join[epidemicHistory, 1, EdgeList[Subgraph[network, Pattern[≪2≫] ⟷ Pattern[≪2≫] /; MemberQ[≪2≫] && MemberQ[≪2≫]]]]].

Take: Nonatomic expression expected at position 1 in Take[epidemicHistory, 1].

Take: Nonatomic expression expected at position 1 in Take[0, 0].

Part: Part specification epidemicHistory[[-1, -1]] is longer than depth of object.

Part: Part specification epidemicHistory[[-1, -1]] is longer than depth of object.

General: Further output of Part::partd will be suppressed during this calculation.

Take: Nonatomic expression expected at position 1 in Take[0., 0].

General: Further output of Take::normal will be suppressed during this calculation.

ListPlot: Take[0., 0] is not a list of numbers or pairs of numbers.

ListPlot: Take[0., 0] is not a list of numbers or pairs of numbers.

ListPlot: Take[0., 0] is not a list of numbers or pairs of numbers.

General: Further output of ListPlot::lpn will be suppressed during this calculation.

Part: Part specification epidemicHistory[[1]] is longer than depth of object.

HighlightGraph: A graph object is expected at position 1 in HighlightGraph[Graph[network, VertexSize → 3], Join[epidemicHistory, 1, EdgeList[Subgraph[network, Pattern[≪2≫] ⟷ Pattern[≪2≫] /; MemberQ[≪2≫] && MemberQ[≪2≫]]]]].

Take: Nonatomic expression expected at position 1 in Take[epidemicHistory, 1].

Take: Nonatomic expression expected at position 1 in Take[0, 0].

Part: Part specification epidemicHistory[[-1, -1]] is longer than depth of object.

Part: Part specification epidemicHistory[[-1, -1]] is longer than depth of object.

General: Further output of Part::partd will be suppressed during this calculation.

Take: Nonatomic expression expected at position 1 in Take[0., 0].

General: Further output of Take::normal will be suppressed during this calculation.