

DNA02

August 1, 2016

1 Table of Contents

- 1 Import Libraries
- 2 Helper Functions
- 3 Data Import
 - 3.1 Split into years
 - 3.2 Create yearly graphs
 - 3.3 Split into months and create monthly graphs
 - 3.4 Visualise Networks
- 4 Centrality and Assortativity Analysis
 - 4.1 Centrality Analysis: Year
 - 4.2 Calculate average of attributes for yearly networks
 - 4.3 Observations
 - 4.4 Monthly Analysis
 - 4.5 Comparison of Yearly and Monthly trends in Centrality
- 5 Attributes Analysis on Graphs
 - 5.1 Instantaneous Amplitude
 - 5.2 Instantaneous Frequency
 - 5.3 Instantaneous Phase
 - 5.4 Derivative of IA
 - 5.5 2nd Derivative of IA
 - 5.6 Instantaneous Acceleration
 - 5.7 Cosine of IP
 - 5.8 Amplitude Weighted IF
 - 5.9 Amplitude weighted Instantaneous Phase
 - 5.10 Mean Curvature
 - 5.11 Kernel PCA 3 Components Ratio
- 6 ICA Ratio
 - 6.1 Resistance Distance
 - 6.2 Cosine of Attributes
 - 6.3 Stationarity Ratio
- 7 Frequency Wavenumber (fk) and Radon Plots
- 8 Towards Global Measures
 - 8.1 Subgraph Stationarity, $\zeta\zeta$
 - 8.2 Persistence & Emergence
 - 8.3 NRMS of Emergence
 - 8.4 NRMS of Network Attributes

8.5 Sort by freq and wavenumber indices
9 Attribute Correlations
9.1 Correlation Matrix
9.2 Correlation Dendogram
9.3 Correlation Matrix > 0.7
9.4 Correlation Network of Attributes
10 Feature Ranking with Gradient Boosting Regressor
10.1 Feature Ranking for predicting Emergence
10.2 Feature Ranking for predicting Average Degree
11 Signal to Noise ratio

2 Import Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import networkx as nx
        import seaborn as sns
        import matplotlib.pyplot as plt
        import scipy as sc
        import random
        from scipy.signal import *
        from numpy.linalg import *
        #plotting parameters
%matplotlib inline
sns.set(style="whitegrid", color_codes=True, context='paper')
plt.rc('axes', grid=False, titlesize='large', labelsize='large', labelweight='normal')
plt.rc('lines', linewidth=4)
plt.rc('font', family='serif', size=16, serif='Georgia Pro')
plt.rc('figure', figsize = (15,6), titlesize='large', titleweight='heavy')
plt.rc('grid', linewidth=5)
```

Set color palette

```
In [628]: sns.palplot(sns.husl_palette(20, h=0.8, l=0.5))
```



```
In [629]: sns.set_palette(sns.husl_palette(20, h=0.8, l=0.5))
```

3 Helper Functions

```
In [188]: def get_val(val):
            return sorted(set(val.values()))
```

```

In [189]: def avg_cent (cent):
    avg = sum(set(cent.values()))/len(cent)
    return avg

In [190]: def get_cen (net):
    degC = nx.degree_centrality(net)
    cloC = nx.closeness_centrality(net)
    betC = nx.betweenness_centrality(net)
    eigC = nx.eigenvector_centrality_numpy(net)
    katzC = nx.katz_centrality_numpy(net)
    loadC = nx.load_centrality(net)

    #cen_df = pd.DataFrame

    return [degC,cloC,betC,eigC,katzC, loadC]

In [191]: def stationarity_ratio(G):
    #stationarity ratio with laplian
    L = nx.normalized_laplacian_matrix(G).todense()
    U = eigvals(L)
    C = np.cov(L)
    CF = np.dot(L,np.dot(np.dot(U.T,C),U))
    r = np.linalg.norm(np.diag(CF))/np.linalg.norm(CF)

    return [r]

In [192]: #cite:`klein1993resistance`
def resistance_distance(net):
    M = nx.normalized_laplacian_matrix(net).todense()
    pseudo = pinv(M)
    N = M.shape[0]
    d = np.diag(pseudo)
    rd = np.kron(d,np.ones((N,1))).T+np.kron(d,np.ones((N,1))).T - pseudo

    return [rd, rd.mean()]

In [193]: def calc_seisatt(net):
    M = nx.normalized_laplacian_matrix(net).todense()
    Ht = hilbert(M)

    #Basic attributes IA, IP, IF
    IA = np.real(np.nan_to_num(np.sqrt(np.dot(M,M)+np.dot(Ht,Ht))))
    IP = np.real(np.nan_to_num(np.arctan(Ht/M)))
    IF,_ = np.real(np.nan_to_num(np.asarray(np.gradient(IP)))))

    #Derivatives
    dIA,_ = np.nan_to_num(np.asarray(np.gradient(IA)))
    d2IA,_ = np.nan_to_num(np.asarray(np.gradient(dIA)))
    IAcc,_ = np.nan_to_num(np.asarray(np.gradient(IF)))

```

```

#Derived Attributes
cosIP = np.cos(IP)
IA_weit_IF = IA * IF
IA_weit_IP = IA*IP

att_globalval = pd.DataFrame([IA.mean(), IP.mean(), IF.mean(), dIA.mean(),
                               IAcc.mean(), cosIP.mean(), IA_weit_IF.mean()])
att_globalval.columns = ['InstAmp', 'InstPhase', 'InstFreq', 'dInstAmp',
                        'IAwIP']

return [IA, IP, IF, dIA, d2IA, IAcc, cosIP, IA_weit_IF, IA_weit_IP, att_globalval]

```

In [194]: **def** curvature(net):

```

from skimage.feature import hessian_matrix, hessian_matrix_det, hessian
M = nx.normalized_laplacian_matrix(net).todense()
M = np.float64(M)
fx, fy = np.gradient(M)
Hxx, Hxy, Hyy = hessian_matrix(M)
K = np.divide((np.dot(Hxx, Hxy)-np.dot(Hxy, Hxy)), \
              (1+np.dot(fx, fx)+np.dot(fy, fy)))

```

```

He1,_ = hessian_matrix_eigvals(Hxx,Hxy,Hyy)
mean_curv = np.trace(He1)
s, a = np.linalg.slogdet(He1)
conc = s * np.exp(a)
Pmax = np.max(He1)
Pmin = np.min(He1)

return [K,mean_curv]

```

In [195]: **def** cal_avgstat(net):

```

#calculate all attributes from previously defined functions here
degC,cloC,betC,eigC,katzC, loadC = get_cent(net)
_,meanK = curvature(net)
_,att_globalval = calc_seisatt(net)
_,norm_rd = resistance_distance(net)
r = stationarity_ratio(net)
den = nx.density(net)
clustcof = nx.average_clustering(net)
algcon = nx.algebraic_connectivity(net)

#create attribute volume here
stat_df = pd.DataFrame([avg_cent(degC), avg_cent(cloC), avg_cent(betC),
                        avg_cent(katzC), avg_cent(loadC), den, clustcof, algcon, meanK])
stat_df.columns= ['AvgDeg', 'AvgCloseness', 'AvgBet', 'AvgEig', 'AvgKatz',
                  'AlgebraicConnect', 'MeanKurv',]
stat_df = stat_df.join(att_globalval)

```

```

stat_df['MeanResistanceDist']= norm_rd
stat_df['StatRat']=r

return stat_df

In [196]: def get_top_keys(dictionary, top):
    items = dictionary.items()
    items = sorted(items, reverse=True, key=lambda x: x[1])
    obj = map(lambda x: x[0], items[:top])
    for i in obj:
        print(i)
#return map(lambda x: x[0], items[:top])

In [197]: def create_graph(df):
    tmp = df.values[:,1:3]
    G= nx.Graph()
    G = nx.from_edgelist(tmp)

    return G

In [198]: def rms(a, axis=None):
    from numpy import mean, sqrt, square
    rms = sqrt(mean(square(a), axis=axis))
    return rms

def nrms(a,b):
    nrms = 200*rms(a-b) / (rms(a)+ rms(b))
    return nrms

In [590]: def get_centhistval(cent_dict, title=None):
    from collections import Counter
    vals = list(Counter(cent_dict.values()))
    freq = list(Counter(cent_dict.values()).values())
    plt.figure()
    pd.DataFrame.from_dict(cent_dict, orient='index').plot(kind='hist',
#sns.distplot(vals,hist=True,kde=False, color='#6F4E37')
    plt.title("Distribution of " + str(title), fontsize=20)
    plt.xlabel("Centrality Value", fontsize=18)
    plt.ylabel("No. of Nodes", fontsize=18)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)
    plt.tight_layout()

    plt.show()

In [200]: def pad_shape(x, ref, offset=0):
    result = np.zeros_like(ref)
    result[0:x.shape[0]+0,0:x.shape[1]+0] = x

```

```
    return result
```

```
In [201]: def subgraph_stat(net1,net2):
    net1_int_net2 = net1.copy()
    net1_int_net2.remove_nodes_from(n for n in net1 if n not in net2)
    net1_u_net2 = nx.disjoint_union(net1, net2)
    int_adjmat = nx.adjacency_matrix(net1_int_net2).todense()
    uni_adjmat = nx.adjacency_matrix(net1_u_net2).todense()
    int_adjmat_pad = pad_shape(int_adjmat,uni_adjmat)
```

```
Ct = np.divide(norm(int_adjmat_pad),norm(uni_adjmat))
```

```
return Ct
```

```
In [202]: def fk_plot(f,name):
    freq = sc.fft(f)
    wavnum = 1/freq
```

```
plt.scatter(freq, wavnum, s=60, marker='^', c='k')
plt.title("F-K Plot of "+str(name), fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel("Wavenumber,k", fontsize=18)
plt.ylabel("Frequency,f", fontsize=18)
```

```
plt.show()
```

```
return [freq,wavnum]
```

```
In [203]: def plot_radon(m, name):
    from skimage.transform import radon
    theta = np.linspace(0., 180., max(m.shape), endpoint=False)
    sinogram = radon(m, theta=theta, circle=True)

    fig, ax = plt.subplots(1,2)

    ax[0].scatter(sinogram[0], sinogram[1], s=60, marker='^', c='k')
    ax[0].set_title("Radon Transform Plot of "+str(name), fontsize=14)

    ax[1].scatter(1/sinogram[0], sinogram[1], s=60, marker='^', c='r')
    ax[1].set_title("1/Radon[0] vs Radon[1] Plot of "+str(name), fontsize=14)

    plt.show()

return sinogram
```

4 Data Import

```
In [204]: data = pd.read_excel("../Data/data_03.2.xlsx")
In [205]: data.head()

Out[205]:      timestamp    to    from  year  month
0 1979-12-31 21:00:00    24    153  1979     12
1 1979-12-31 21:00:00    24    153  1979     12
2 1979-12-31 21:00:00    29     29  1979     12
3 1979-12-31 21:00:00    29     29  1979     12
4 1979-12-31 21:00:00    29     29  1979     12

In [206]: years = sorted(set(data.year))
years

Out[206]: [1979, 1998, 1999, 2000, 2001, 2002]

In [207]: #total % of mislabelled 1979 entries
          (data[data.year==1979].count()/data.shape[0]) * 100

Out[207]: timestamp      0.138746
          to            0.138746
          from           0.138746
          year           0.138746
          month          0.138746
          dtype: float64
```

4.1 Split into years

```
In [208]: df_98 = data[data.year==years[1]]
df_99 = data[data.year==years[2]]
df_2k = data[data.year==years[3]]
df_2k1 = data[data.year==years[4]]
df_2k2 = data[data.year==years[5]]

In [209]: df_2k.head()

Out[209]:      timestamp    to    from  year  month
3971 2000-01-03 06:47:00    82     51  2000     1
3972 2000-01-03 06:47:00    82     51  2000     1
3973 2000-01-03 06:47:00    82     51  2000     1
3974 2000-01-03 06:47:00    82     51  2000     1
3975 2000-01-03 06:47:00    82     51  2000     1
```

4.2 Create yearly graphs

```
In [210]: Gt0 = create_graph(df_98)
Gt1 = create_graph(df_99)
Gt2 = create_graph(df_2k)
Gt3 = create_graph(df_2k1)
Gt4 = create_graph(df_2k2)
```

4.3 Split into months and create monthly graphs

```
In [211]: nov_98 = df_98[df_98.month==11]
dec_98= df_98[df_98.month==12]

G_nov98 = create_graph(nov_98)
G_dec98 = create_graph(dec_98)

In [212]: jan_99=df_99[df_99.month==1]
feb_99=df_99[df_99.month==2]
mar_99=df_99[df_99.month==3]
apr_99=df_99[df_99.month==4]
may_99=df_99[df_99.month==5]
jun_99=df_99[df_99.month==6]
jul_99=df_99[df_99.month==7]
aug_99=df_99[df_99.month==8]
sep_99=df_99[df_99.month==9]
oct_99=df_99[df_99.month==10]
nov_99=df_99[df_99.month==11]
dec_99=df_99[df_99.month==12]

G_jan_99=create_graph(jan_99)
G_feb_99=create_graph(feb_99)
G_mar_99=create_graph(mar_99)
G_apr_99=create_graph(apr_99)
G_may_99=create_graph(may_99)
G_jun_99=create_graph(jun_99)
G_jul_99=create_graph(jul_99)
G_aug_99=create_graph(aug_99)
G_sep_99=create_graph(sep_99)
G_oct_99=create_graph(oct_99)
G_nov_99=create_graph(nov_99)
G_dec_99=create_graph(dec_99)

In [213]: jan_2k=df_2k[df_2k.month==1]
feb_2k=df_2k[df_2k.month==2]
mar_2k=df_2k[df_2k.month==3]
apr_2k=df_2k[df_2k.month==4]
may_2k=df_2k[df_2k.month==5]
jun_2k=df_2k[df_2k.month==6]
jul_2k=df_2k[df_2k.month==7]
aug_2k=df_2k[df_2k.month==8]
sep_2k=df_2k[df_2k.month==9]
oct_2k=df_2k[df_2k.month==10]
nov_2k=df_2k[df_2k.month==11]
dec_2k=df_2k[df_2k.month==12]
```

```
G_jan_2k=create_graph(jan_2k)
G_feb_2k=create_graph(feb_2k)
G_mar_2k=create_graph(mar_2k)
G_apr_2k=create_graph(apr_2k)
G_may_2k=create_graph(may_2k)
G_jun_2k=create_graph(jun_2k)
G_jul_2k=create_graph(jul_2k)
G_aug_2k=create_graph(aug_2k)
G_sep_2k=create_graph(sep_2k)
G_oct_2k=create_graph(oct_2k)
G_nov_2k=create_graph(nov_2k)
G_dec_2k=create_graph(dec_2k)
```

```
In [214]: jan_2k1=df_2k1[df_2k1.month==1]
feb_2k1=df_2k1[df_2k1.month==2]
mar_2k1=df_2k1[df_2k1.month==3]
apr_2k1=df_2k1[df_2k1.month==4]
may_2k1=df_2k1[df_2k1.month==5]
jun_2k1=df_2k1[df_2k1.month==6]
jul_2k1=df_2k1[df_2k1.month==7]
aug_2k1=df_2k1[df_2k1.month==8]
sep_2k1=df_2k1[df_2k1.month==9]
oct_2k1=df_2k1[df_2k1.month==10]
nov_2k1=df_2k1[df_2k1.month==11]
dec_2k1=df_2k1[df_2k1.month==12]
```

```
G_jan_2k1=create_graph(jan_2k1)
G_feb_2k1=create_graph(feb_2k1)
G_mar_2k1=create_graph(mar_2k1)
G_apr_2k1=create_graph(apr_2k1)
G_may_2k1=create_graph(may_2k1)
G_jun_2k1=create_graph(jun_2k1)
G_jul_2k1=create_graph(jul_2k1)
G_aug_2k1=create_graph(aug_2k1)
G_sep_2k1=create_graph(sep_2k1)
G_oct_2k1=create_graph(oct_2k1)
G_nov_2k1=create_graph(nov_2k1)
G_dec_2k1=create_graph(dec_2k1)
```

```
In [215]: jan_2k2=df_2k2[df_2k2.month==1]
feb_2k2=df_2k2[df_2k2.month==2]
mar_2k2=df_2k2[df_2k2.month==3]
apr_2k2=df_2k2[df_2k2.month==4]
may_2k2=df_2k2[df_2k2.month==5]
jun_2k2=df_2k2[df_2k2.month==6]
jul_2k2=df_2k2[df_2k2.month==7]
aug_2k2=df_2k2[df_2k2.month==8]
```

```
sep_2k2=df_2k2[df_2k2.month==9]
oct_2k2=df_2k2[df_2k2.month==10]
nov_2k2=df_2k2[df_2k2.month==11]
dec_2k2=df_2k2[df_2k2.month==12]
```

```
G_jan_2k2=create_graph(jan_2k2)
G_feb_2k2=create_graph(feb_2k2)
G_mar_2k2=create_graph(mar_2k2)
G_apr_2k2=create_graph(apr_2k2)
G_may_2k2=create_graph(may_2k2)
G_jun_2k2=create_graph(jun_2k2)
G_jul_2k2=create_graph(jul_2k2)
G_aug_2k2=create_graph(aug_2k2)
G_sep_2k2=create_graph(sep_2k2)
G_oct_2k2=create_graph(oct_2k2)
G_nov_2k2=create_graph(nov_2k2)
G_dec_2k2=create_graph(dec_2k2)
```

4.4 Visualise Networks

```
In [216]: plt.figure(figsize=(18,18))
plt.suptitle('Enron Email Dynamic Network Year level', fontsize=24)
plt.subplot(321)
nx.draw_spring(Gt0, node_color='#810541', node_shape='^', edge_color ='#C00000')
plt.title("Graph in 1998", fontsize=18)

plt.subplot(322)
nx.draw_spring(Gt1, node_color='#810541', node_shape='^', edge_color ='#C00000')
plt.title("Graph in 1999", fontsize=18)

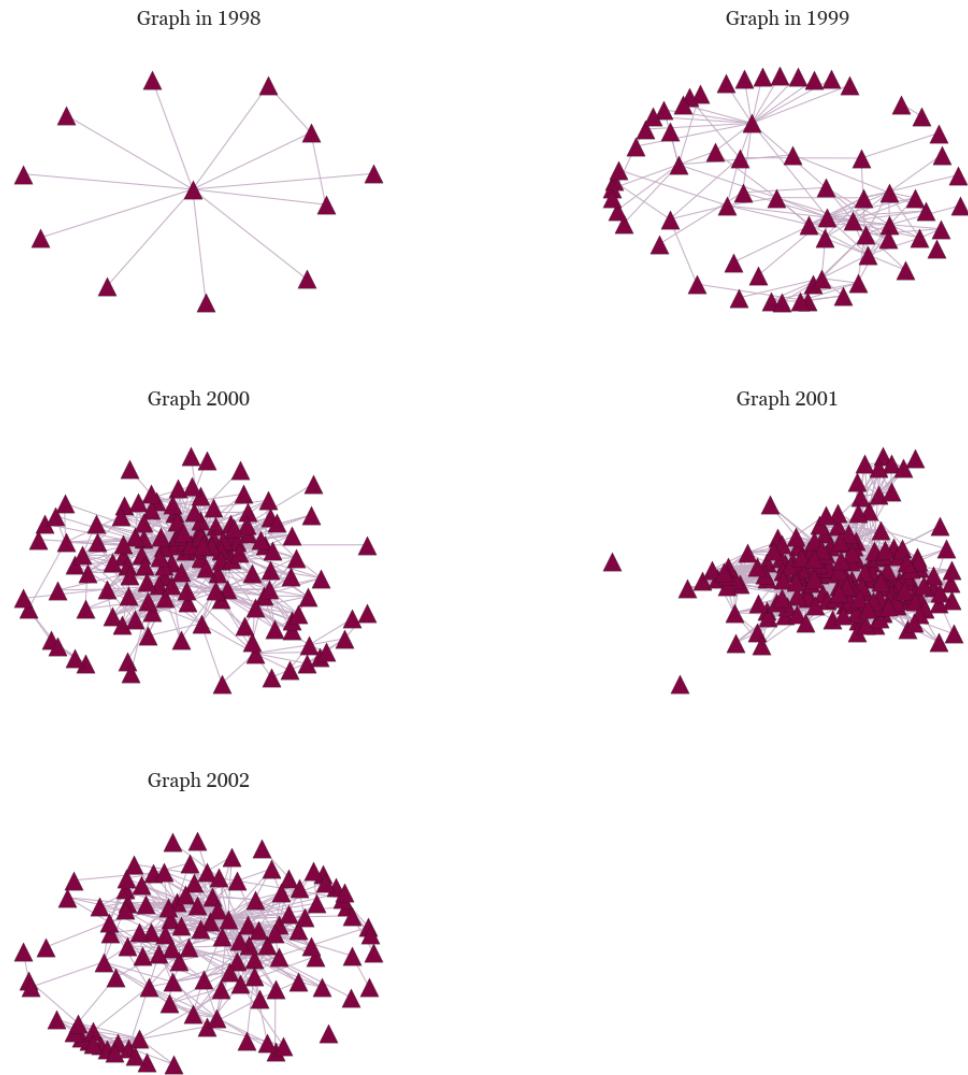
plt.subplot(323)
nx.draw_spring(Gt2, node_color='#810541', node_shape='^', edge_color ='#C00000')
plt.title("Graph 2000", fontsize=18)

plt.subplot(324)
nx.draw_spring(Gt3, node_color='#810541', node_shape='^', edge_color ='#C00000')
plt.title("Graph 2001", fontsize=18)

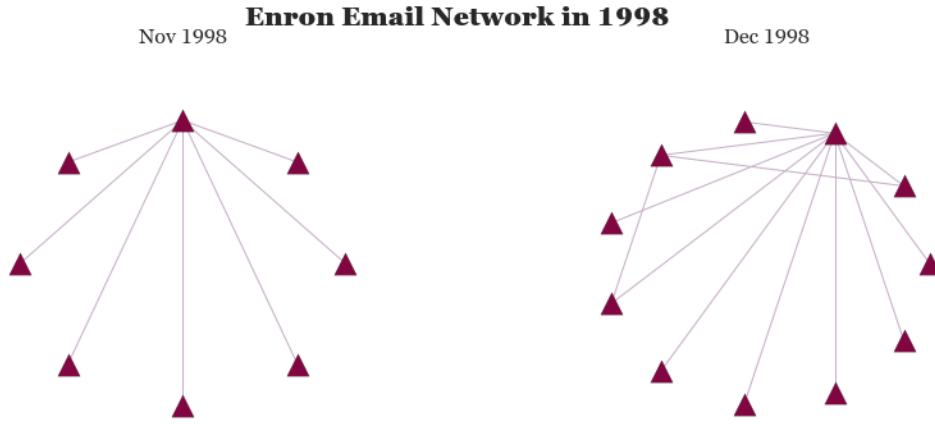
plt.subplot(325)
nx.draw_spring(Gt4, node_color='#810541', node_shape='^', edge_color ='#C00000')
plt.title("Graph 2002", fontsize=18)

plt.show()
```

Enron Email Dynamic Network Year level



```
In [217]: plt.figure()
plt.suptitle("Enron Email Network in 1998", fontsize=20)
plt.subplot(121)
plt.title("Nov 1998", fontsize=16)
nx.draw_circular(G_nov98, node_color='#810541', node_shape='^', edge_color
plt.subplot(122)
plt.title("Dec 1998", fontsize=16)
nx.draw_circular(G_dec98, node_color='#810541', node_shape='^', edge_color
```



```
In [218]: plt.figure(figsize=(32,18))
plt.suptitle("Enron Email Network in 99", fontsize=40)

plt.subplot(331)
plt.title("Jan 99", fontsize=25)
nx.draw_circular(G_jan_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(332)
plt.title("Feb 99", fontsize=25)
nx.draw_circular(G_feb_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(333)
plt.title("Mar 99", fontsize=25)
nx.draw_circular(G_mar_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(334)
plt.title("Apr 99", fontsize=25)
nx.draw_circular(G_apr_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(335)
plt.title("May 99", fontsize=25)
nx.draw_circular(G_may_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(336)
plt.title("Jun 99", fontsize=25)
nx.draw_circular(G_jun_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(337)
plt.title("Jul 99", fontsize=25)
nx.draw_circular(G_jul_99, node_color='#810541', node_shape='^', edge_color
```

```

plt.subplot(338)
plt.title("Aug 99", fontsize=25)
nx.draw_circular(G_aug_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(339)
plt.title("Sep 99", fontsize=25)
nx.draw_circular(G_sep_99, node_color='#810541', node_shape='^', edge_color

plt.figure(figsize=(32, 5))
# plt.suptitle("Enron Email Network in 99", fontsize=40)

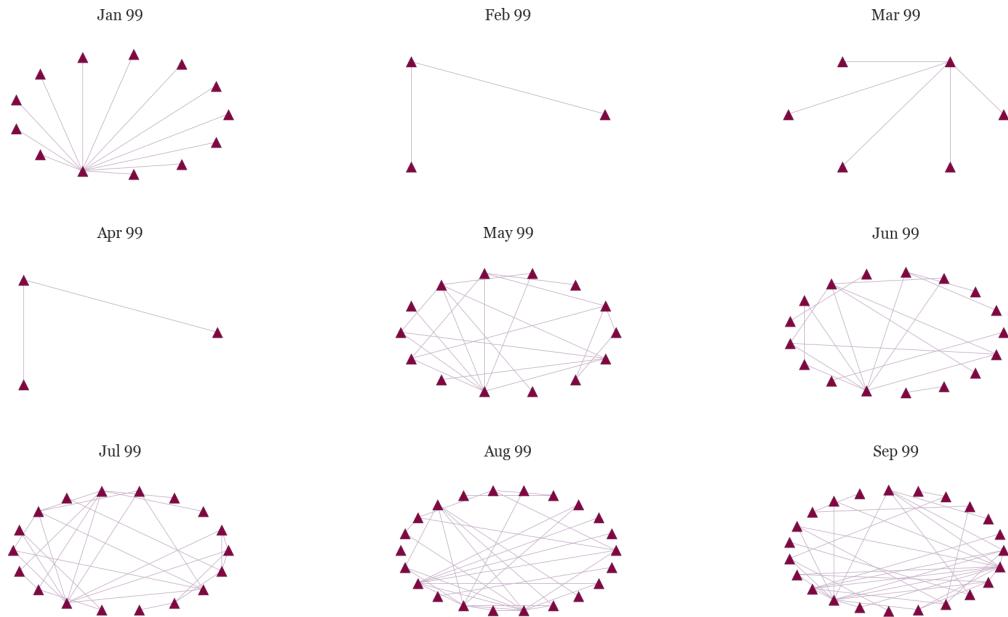
plt.subplot(131)
plt.title("Oct 99", fontsize=25)
nx.draw_circular(G_oct_99, node_color='#810541', node_shape='^', edge_color

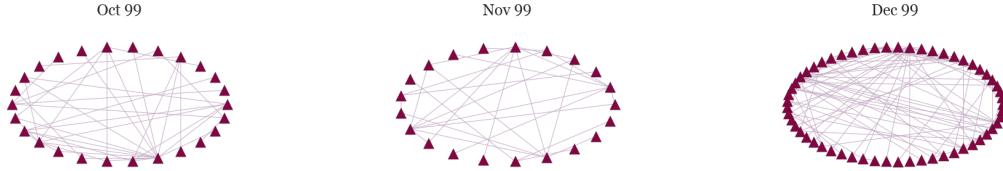
plt.subplot(132)
plt.title("Nov 99", fontsize=25)
nx.draw_circular(G_nov_99, node_color='#810541', node_shape='^', edge_color

plt.subplot(133)
plt.title("Dec 99", fontsize=25)
nx.draw_circular(G_dec_99, node_color='#810541', node_shape='^', edge_color

```

Enron Email Network in 99





```
In [219]: plt.figure(figsize=(32,18))
plt.suptitle("Enron Email Network in 2k", fontsize=40)

plt.subplot(331)
plt.title("Jan 2k", fontsize=25)
nx.draw_circular(G_jan_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(332)
plt.title("Feb 2k", fontsize=25)
nx.draw_circular(G_feb_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(333)
plt.title("Mar 2k", fontsize=25)
nx.draw_circular(G_mar_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(334)
plt.title("Apr 2k", fontsize=25)
nx.draw_circular(G_apr_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(335)
plt.title("May 2k", fontsize=25)
nx.draw_circular(G_may_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(336)
plt.title("Jun 2k", fontsize=25)
nx.draw_circular(G_jun_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(337)
plt.title("Jul 2k", fontsize=25)
nx.draw_circular(G_jul_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(338)
plt.title("Aug 2k", fontsize=25)
nx.draw_circular(G_aug_2k, node_color='#810541', node_shape='^', edge_color

plt.subplot(339)
plt.title("Sep 2k", fontsize=25)
nx.draw_circular(G_sep_2k, node_color='#810541', node_shape='^', edge_color
```

```

plt.figure(figsize=(32, 5))
#plt.suptitle("Enron Email Network in 2k", fontsize=40)

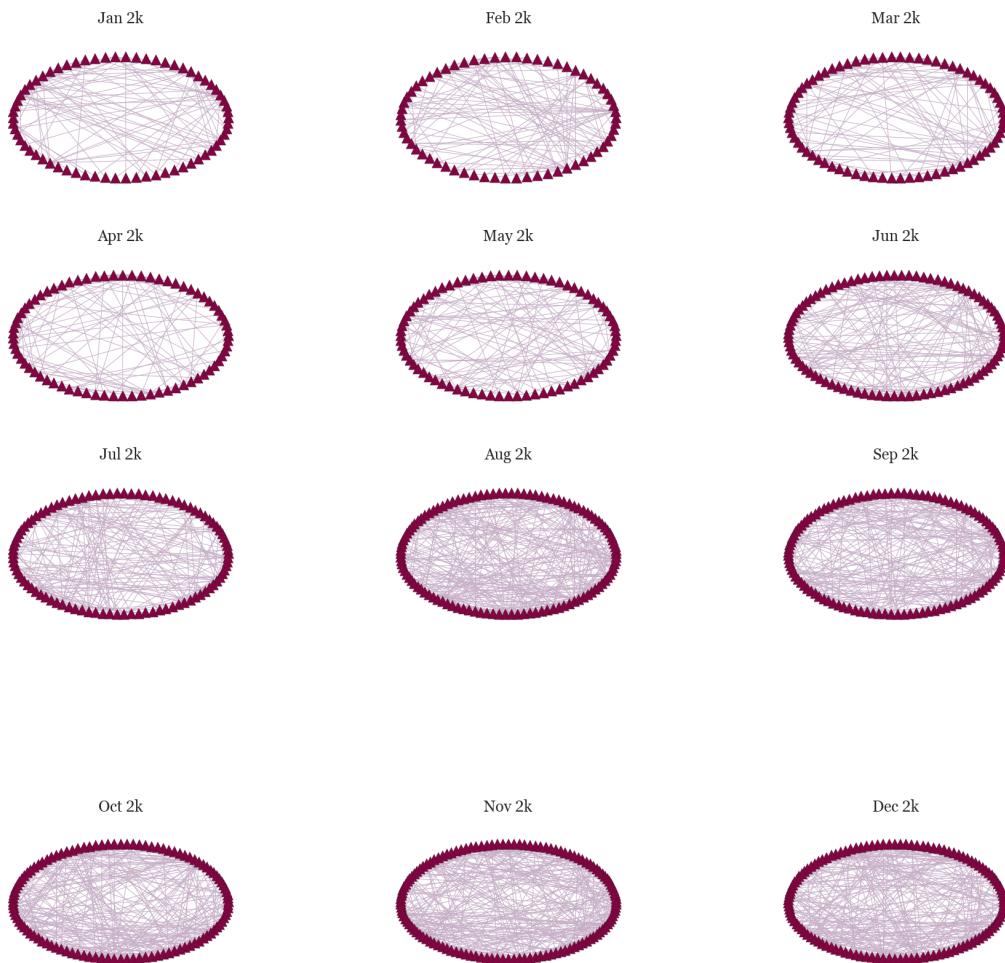
plt.subplot(131)
plt.title("Oct 2k", fontsize=25)
nx.draw_circular(G_oct_2k, node_color='#810541', node_shape='^', edge_color='purple')

plt.subplot(132)
plt.title("Nov 2k", fontsize=25)
nx.draw_circular(G_nov_2k, node_color='#810541', node_shape='^', edge_color='purple')

plt.subplot(133)
plt.title("Dec 2k", fontsize=25)
nx.draw_circular(G_dec_2k, node_color='#810541', node_shape='^', edge_color='purple')

```

Enron Email Network in 2k



```
In [220]: plt.figure(figsize=(32,18))
plt.suptitle("Enron Email Network in 2k1", fontsize=40)

plt.subplot(331)
plt.title("Jan 2k1", fontsize=25)
nx.draw_circular(G_jan_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(332)
plt.title("Feb 2k1", fontsize=25)
nx.draw_circular(G_feb_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(333)
plt.title("Mar 2k1", fontsize=25)
nx.draw_circular(G_mar_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(334)
plt.title("Apr 2k1", fontsize=25)
nx.draw_circular(G_apr_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(335)
plt.title("May 2k1", fontsize=25)
nx.draw_circular(G_may_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(336)
plt.title("Jun 2k1", fontsize=25)
nx.draw_circular(G_jun_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(337)
plt.title("Jul 2k1", fontsize=25)
nx.draw_circular(G_jul_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(338)
plt.title("Aug 2k1", fontsize=25)
nx.draw_circular(G_aug_2k1, node_color='#810541', node_shape='^', edge_col

plt.subplot(339)
plt.title("Sep 2k1", fontsize=25)
nx.draw_circular(G_sep_2k1, node_color='#810541', node_shape='^', edge_col

plt.figure(figsize=(32,5))
# plt.suptitle("Enron Email Network in 2k1", fontsize=40)

plt.subplot(131)
plt.title("Oct 2k1", fontsize=25)
nx.draw_circular(G_oct_2k1, node_color='#810541', node_shape='^', edge_col

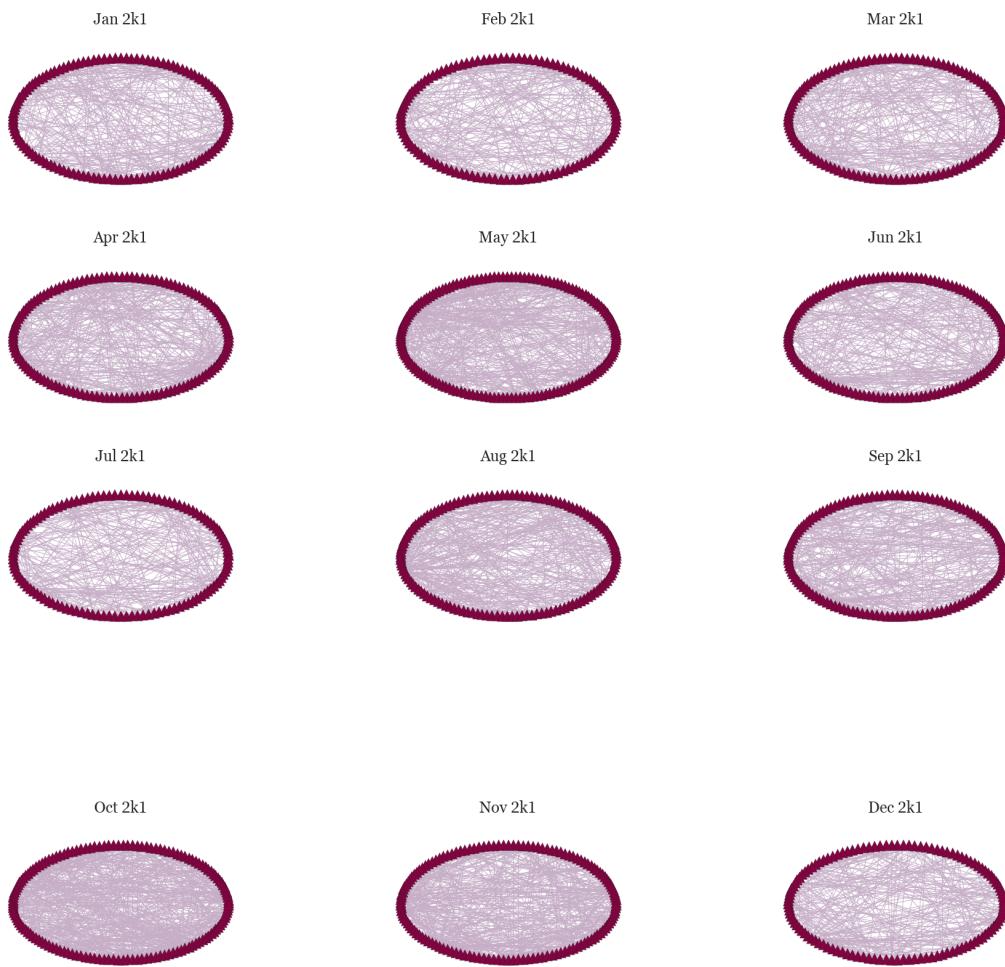
plt.subplot(132)
plt.title("Nov 2k1", fontsize=25)
```

```

nx.draw_circular(G_nov_2k1, node_color='#810541', node_shape='^', edge_color='black', edge_width=0.5)
plt.subplot(133)
plt.title("Dec 2k1", fontsize=25)
nx.draw_circular(G_dec_2k1, node_color='#810541', node_shape='^', edge_color='black', edge_width=0.5)

```

Enron Email Network in 2k1



```

In [221]: plt.figure(figsize=(32,18))
plt.suptitle("Enron Email Network in 2k2", fontsize=40)

plt.subplot(331)
plt.title("Jan 2k2", fontsize=25)
nx.draw_circular(G_jan_2k2, node_color='#810541', node_shape='^', edge_color='black', edge_width=0.5)

plt.subplot(332)
plt.title("Feb 2k2", fontsize=25)

```

```

nx.draw_circular(G_feb_2k2, node_color='#810541', node_shape='^', edge_color='purple')

plt.subplot(333)
plt.title("Mar 2k2", fontsize=25)
nx.draw_circular(G_mar_2k2, node_color='#810541', node_shape='^', edge_color='purple')

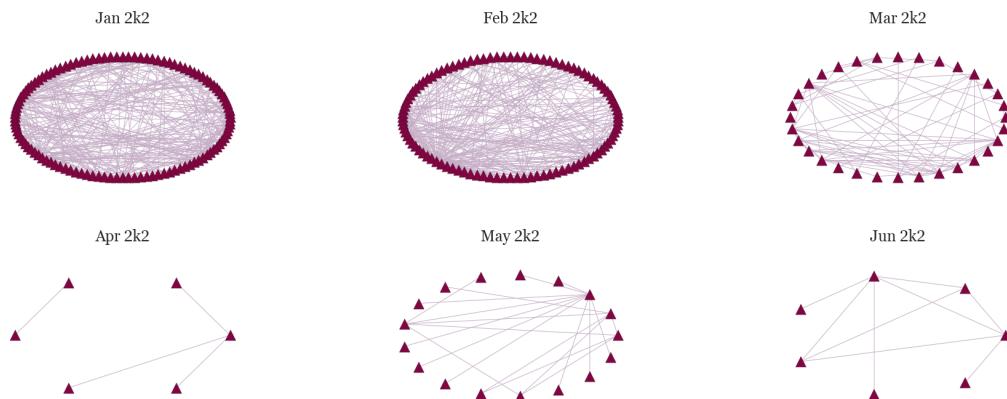
plt.subplot(334)
plt.title("Apr 2k2", fontsize=25)
nx.draw_circular(G_apr_2k2, node_color='#810541', node_shape='^', edge_color='purple')

plt.subplot(335)
plt.title("May 2k2", fontsize=25)
nx.draw_circular(G_may_2k2, node_color='#810541', node_shape='^', edge_color='purple')

plt.subplot(336)
plt.title("Jun 2k2", fontsize=25)
nx.draw_circular(G_jun_2k2, node_color='#810541', node_shape='^', edge_color='purple')

```

Enron Email Network in 2k2



5 Centrality and Assortativity Analysis

We look at some traditional network measures in this section for the networks both at the year and monthly level.

Analysis Methodology

1. Analyse network at the yearly level using centrality measures and other measures
 - Find top 5 nodes for each centrality measure across the years
2. Analyse network across years using averaged measures
3. Analyse network in monthly intervals using average centrality and other measures
4. Compare yearly and monthly trends

The specific measures used are:

Degree Centrality

Degree Centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information).

Closeness Centrality

The farness of a node x is defined as the sum of its distances from all other nodes, and its closeness was defined by the reciprocal of the farness.

Betweenness Centrality

Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes

Eigenvector Centrality

Eigenvector centrality (also called eigencentrality) is a measure of the influence of a node in a network. It assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.

Katz Centrality

Katz centrality measures the number of all nodes that can be connected through a path, while the contributions of distant nodes are penalized.

Load Centrality

The load centrality of a node is the fraction of all shortest paths that pass through that node.

Density

The ratio of the number of edges and the number of possible edges.

Clustering Coefficient

The local clustering of each node in a graph is the fraction of triangles that actually exist over all possible triangles in its neighborhood. The average clustering coefficient of a graph is the mean of local clusterings.

Algebraic Connectivity

By the definition of algebraic connectivity, the second smallest eigenvalue of the graph Laplacian is a lower bound on the node connectivity. Increases in algebraic connectivity actually correspond to a decrease in node connectivity. This means that the networks are actually less robust with respect to node-connectivity as the algebraic connectivity increases.

References:

1. [Centrality](#)
2. [Algebraic Connectivity](#)

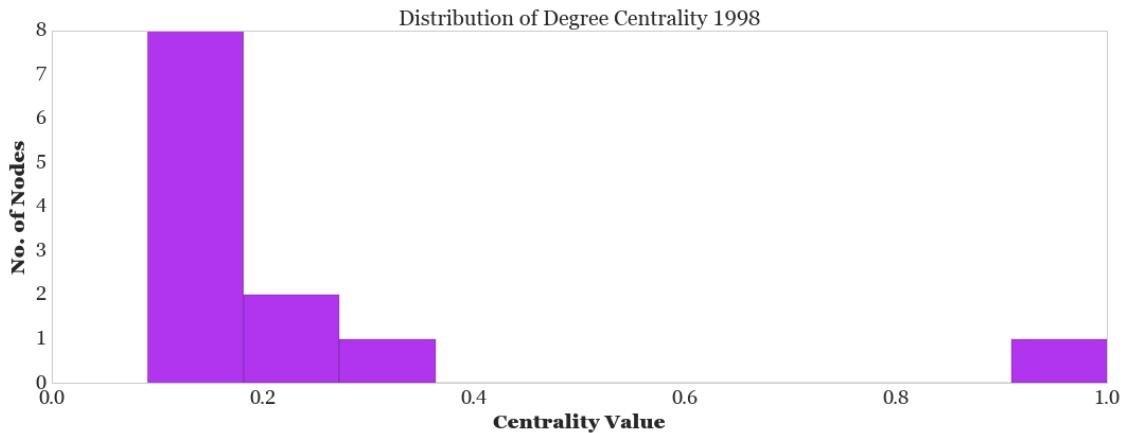
5.1 Centrality Analysis: Year

```
In [630]: degC0, cloC0, betC0, eigC0, katzC0, loadC0 = get_cent(Gt0)
          degC1, cloC1, betC1, eigC1, katzC1, loadC1 = get_cent(Gt1)
          degC2, cloC2, betC2, eigC2, katzC2, loadC2 = get_cent(Gt2)
          degC3, cloC3, betC3, eigC3, katzC3, loadC3 = get_cent(Gt3)
          degC4, cloC4, betC4, eigC4, katzC4, loadC4 = get_cent(Gt4)
```

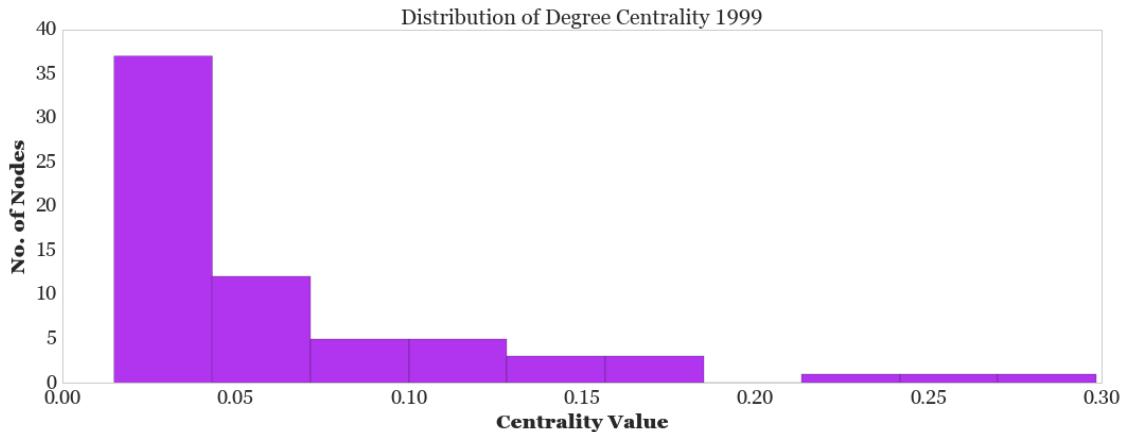
```
In [631]: get_centhistval(degC0, 'Degree Centrality 1998')
          get_centhistval(degC1, 'Degree Centrality 1999')
          get_centhistval(degC2, 'Degree Centrality 2000')
```

```
get_centhistval(degC3, 'Degree Centrality 2001')
get_centhistval(degC4, 'Degree Centrality 2002')
```

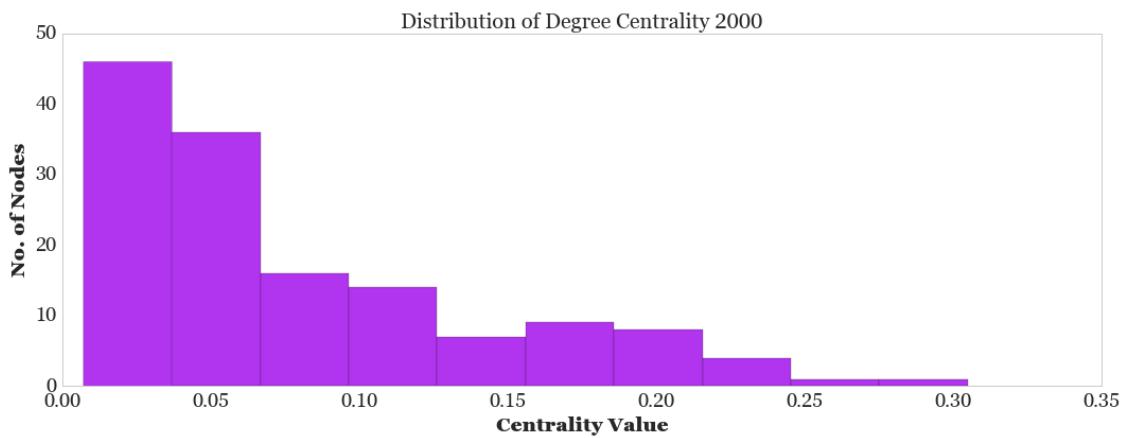
<matplotlib.figure.Figure at 0x1632f2f8b38>



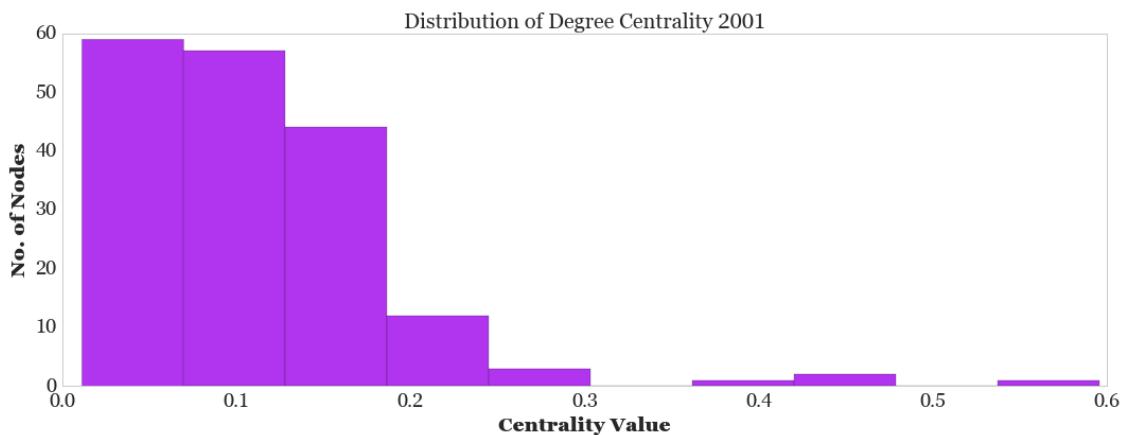
<matplotlib.figure.Figure at 0x1632f2f8be0>



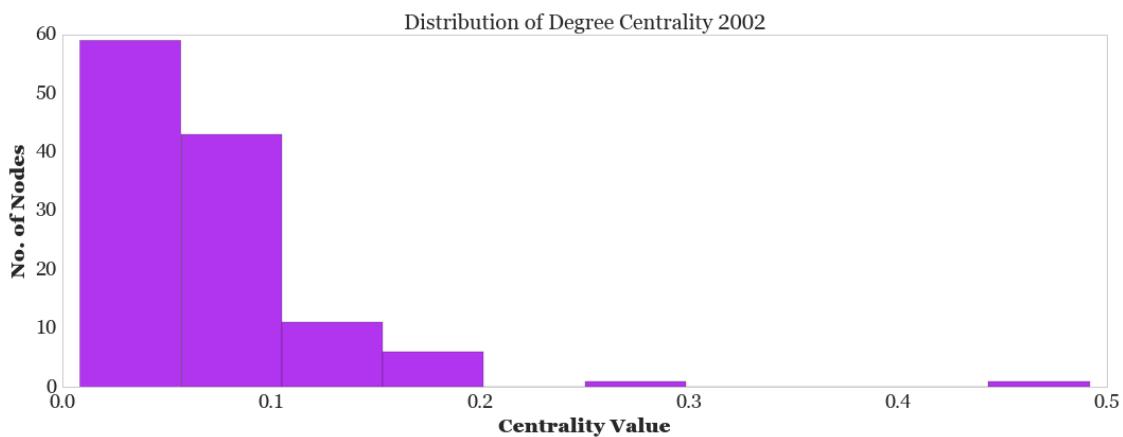
<matplotlib.figure.Figure at 0x1632c606c18>



<matplotlib.figure.Figure at 0x16330183b70>



<matplotlib.figure.Figure at 0x16331ac7978>



```
In [632]: print("Top 5 Degree Centrality Nodes year 98")
    get_top_keys(degC0,5)
    print("Top 5 Degree Centrality Nodes year 99")
    get_top_keys(degC1,5)
    print("Top 5 Degree Centrality Nodes year 2k")
    get_top_keys(degC2,5)
    print("Top 5 Degree Centrality Nodes year 2k1")
    get_top_keys(degC3,5)
    print("Top 5 Degree Centrality Nodes year 2k2")
    get_top_keys(degC4,5)
```

```
Top 5 Degree Centrality Nodes year 98
```

```
114  
112  
65  
145  
160
```

```
Top 5 Degree Centrality Nodes year 99
```

```
153  
169  
114  
155  
38
```

```
Top 5 Degree Centrality Nodes year 2k
```

```
82  
27  
140  
157  
78
```

```
Top 5 Degree Centrality Nodes year 2k1
```

```
82  
153  
107  
94  
126
```

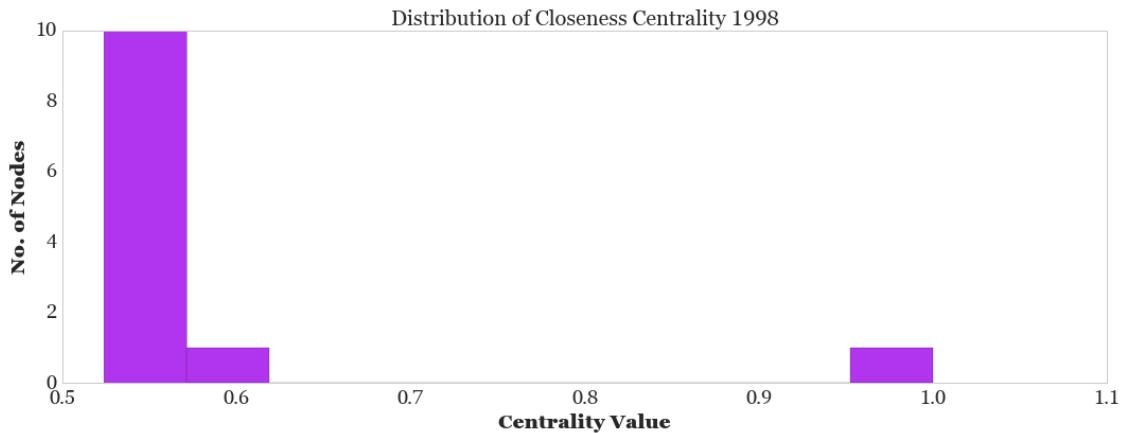
```
Top 5 Degree Centrality Nodes year 2k2
```

```
105  
107  
82  
90  
159
```

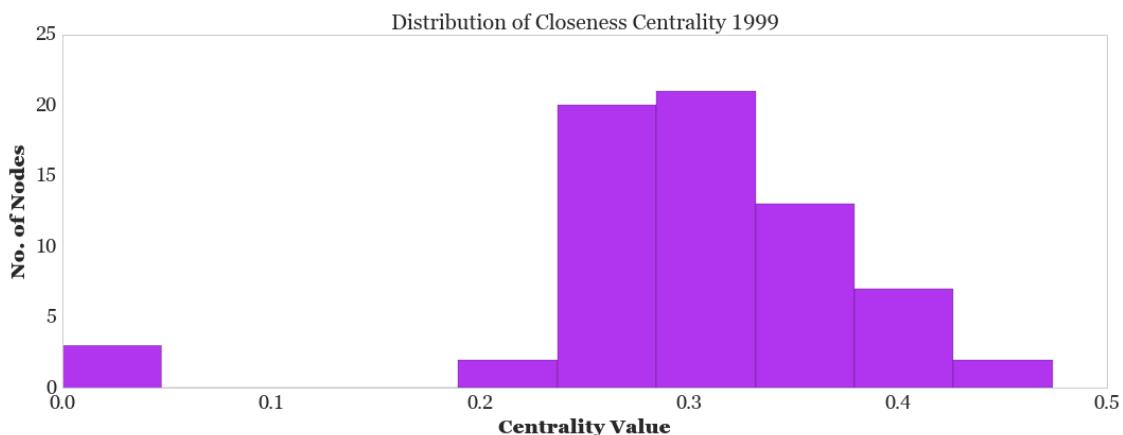
```
In [633]: get_centhistval(cloC0,'Closeness Centrality 1998')
    get_centhistval(cloC1,'Closeness Centrality 1999')
```

```
get_centhistval(cloC2, 'Closeness Centrality 2000')
get_centhistval(cloC3, 'Closeness Centrality 2001')
get_centhistval(cloC4, 'Closeness Centrality 2002')
```

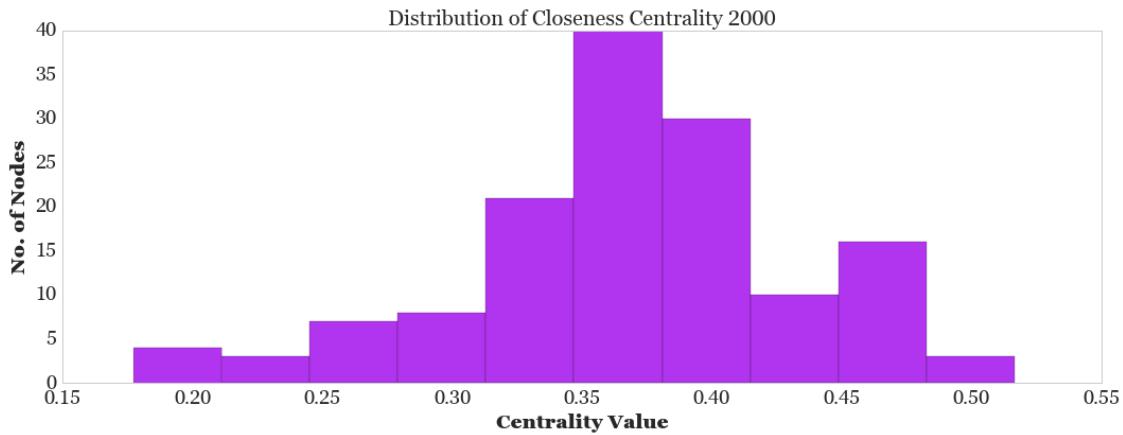
<matplotlib.figure.Figure at 0x163323189b0>



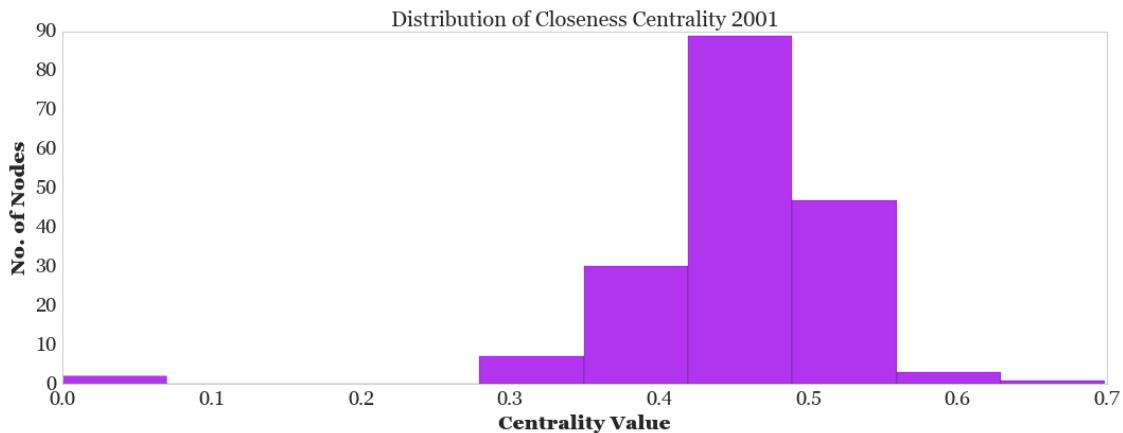
<matplotlib.figure.Figure at 0x1632f2f8ac8>



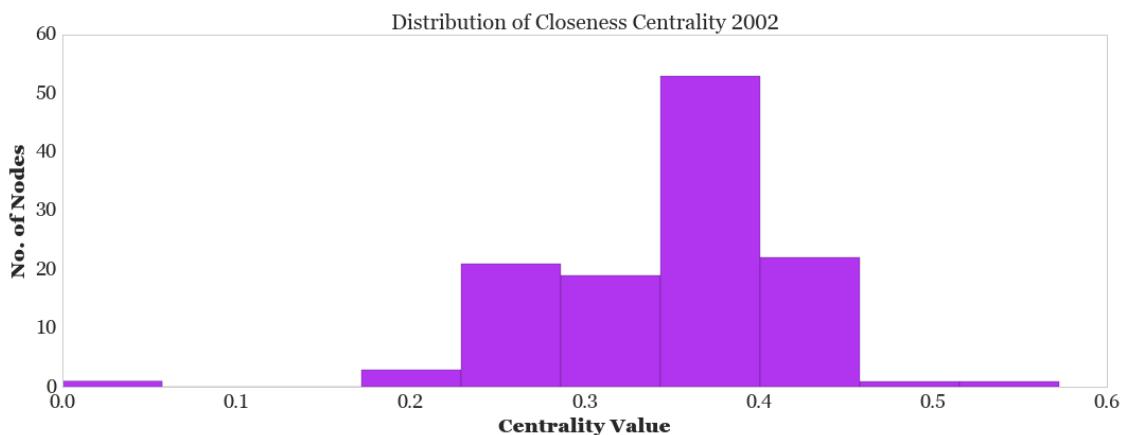
<matplotlib.figure.Figure at 0x163367a91d0>



<matplotlib.figure.Figure at 0x163374fd278>



<matplotlib.figure.Figure at 0x163367b1940>



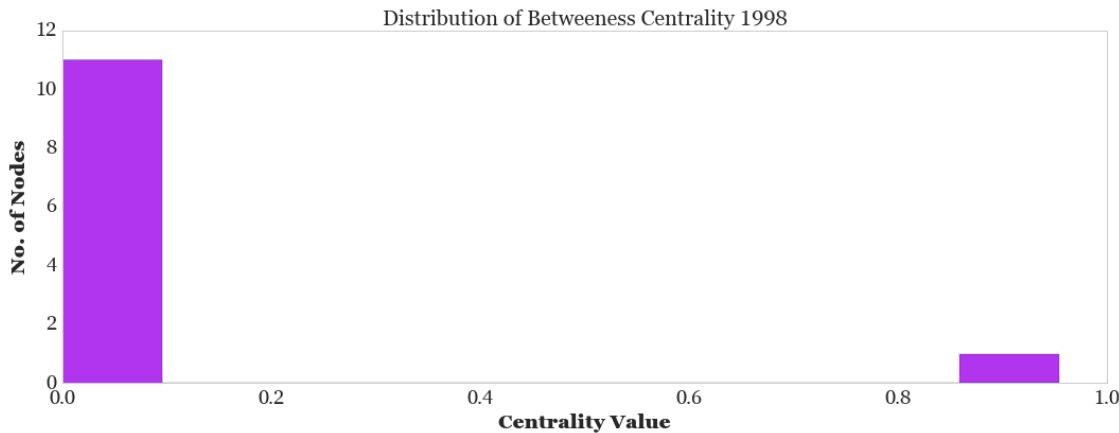
```
In [634]: print("Top 5 Closeness Centrality Nodes year 98")
get_top_keys(cloC0,5)
print("Top 5 Closeness Centrality Nodes year 99")
get_top_keys(cloC1,5)
print("Top 5 Closeness Centrality Nodes year 2k")
get_top_keys(cloC2,5)
print("Top 5 Closeness Centrality Nodes year 2k1")
get_top_keys(cloC3,5)
print("Top 5 Closeness Centrality Nodes year 2k2")
get_top_keys(cloC4,5)

Top 5 Closeness Centrality Nodes year 98
114
112
65
145
160
Top 5 Closeness Centrality Nodes year 99
169
153
114
22
145
Top 5 Closeness Centrality Nodes year 2k
82
27
140
145
163
Top 5 Closeness Centrality Nodes year 2k1
82
153
107
94
126
Top 5 Closeness Centrality Nodes year 2k2
105
107
34
82
133
```

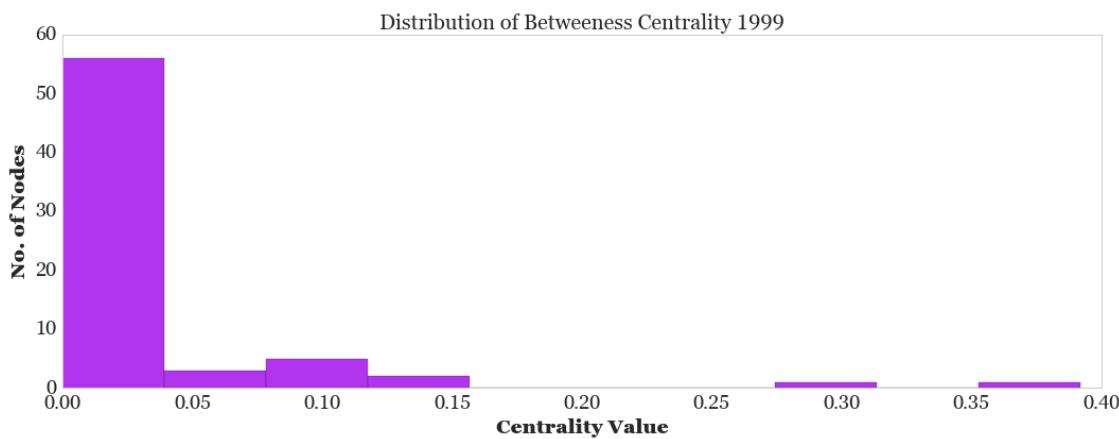
```
In [635]: get_centhistval(betC0,'Betweeness Centrality 1998')
get_centhistval(betC1,'Betweeness Centrality 1999')
```

```
get_centhistval(betc2, 'Betweeness Centrality 2000')
get_centhistval(betc3, 'Betweeness Centrality 2001')
get_centhistval(betc4, 'Betweeness Centrality 2002')
```

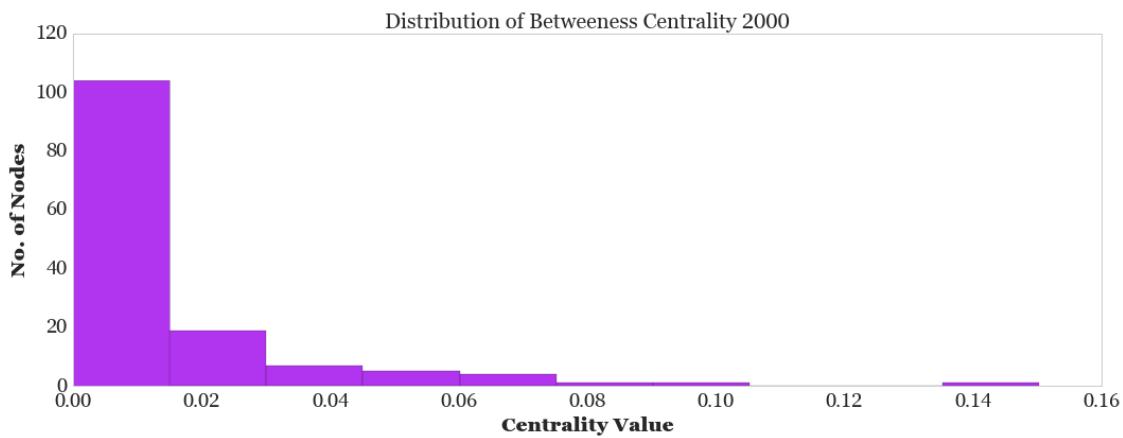
<matplotlib.figure.Figure at 0x16331ad74a8>



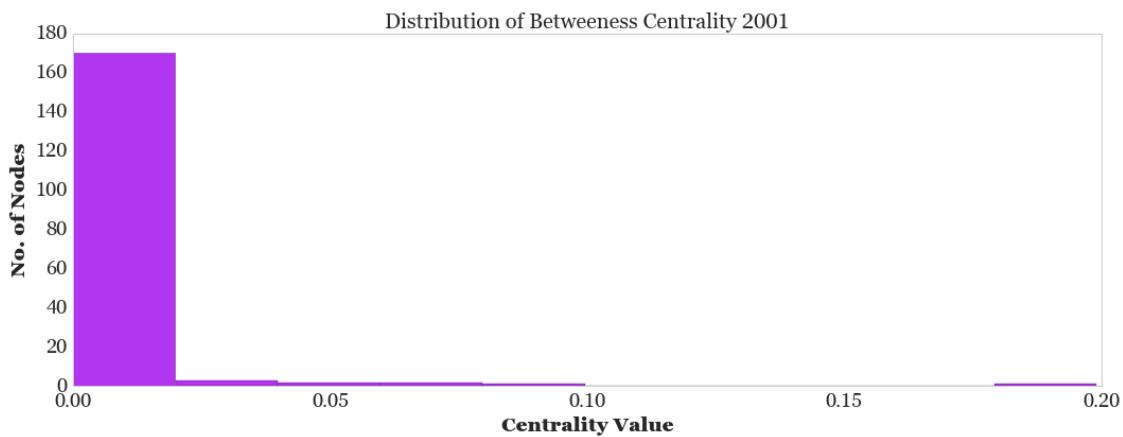
<matplotlib.figure.Figure at 0x1632f44f550>



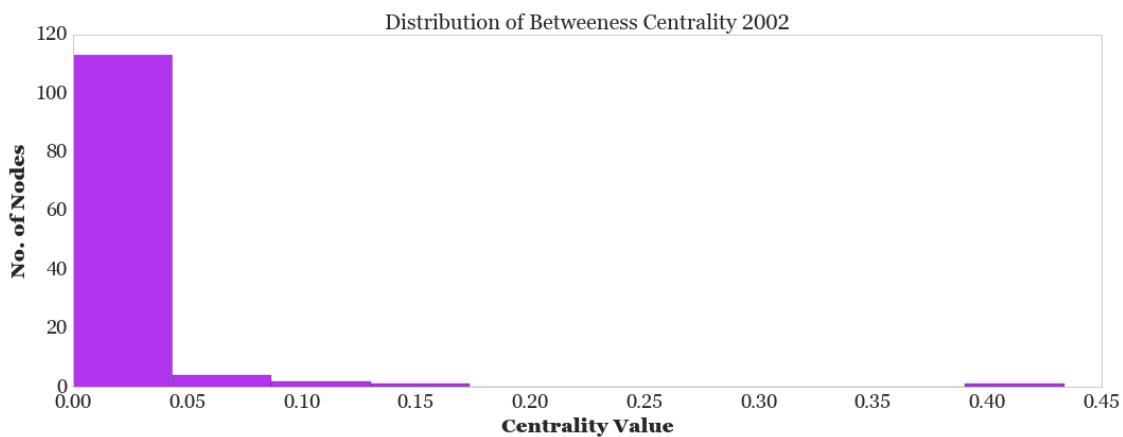
<matplotlib.figure.Figure at 0x16332321b38>



<matplotlib.figure.Figure at 0x1632c1a3160>



<matplotlib.figure.Figure at 0x163228ec2b0>



```
In [636]: print("Top 5 Betweeness Centrality Nodes year 98")
get_top_keys(betC0,5)
print("Top 5 Betweeness Centrality Nodes year 99")
get_top_keys(betC1,5)
print("Top 5 Betweeness Centrality Nodes year 2k")
get_top_keys(betC2,5)
print("Top 5 Betweeness Centrality Nodes year 2k1")
get_top_keys(betC3,5)
print("Top 5 Betweeness Centrality Nodes year 2k2")
get_top_keys(betC4,5)
```

```
Top 5 Betweeness Centrality Nodes year 98
```

```
114  
112  
160  
65  
155
```

```
Top 5 Betweeness Centrality Nodes year 99
```

```
153  
169  
17  
163  
22
```

```
Top 5 Betweeness Centrality Nodes year 2k
```

```
82  
63  
140  
157  
79
```

```
Top 5 Betweeness Centrality Nodes year 2k1
```

```
82  
153  
94  
107  
9
```

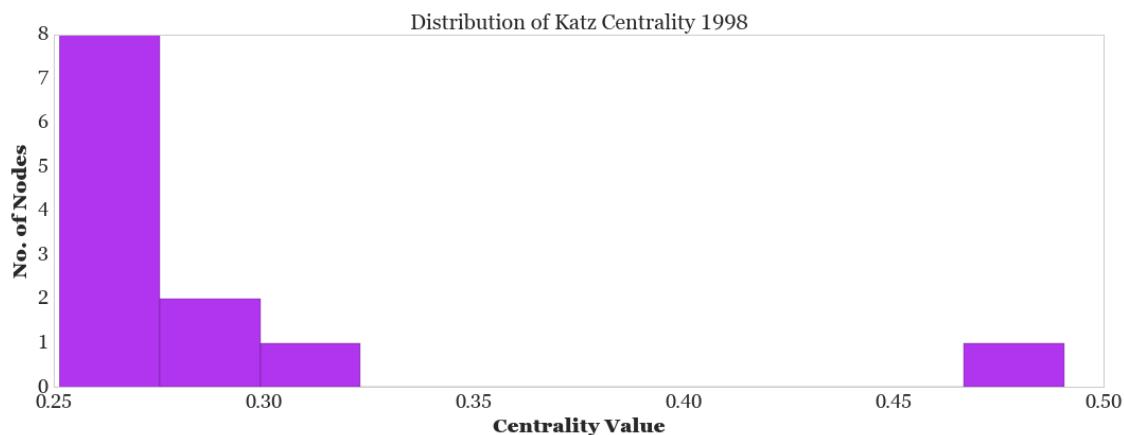
```
Top 5 Betweeness Centrality Nodes year 2k2
```

```
105  
34  
107  
158  
17
```

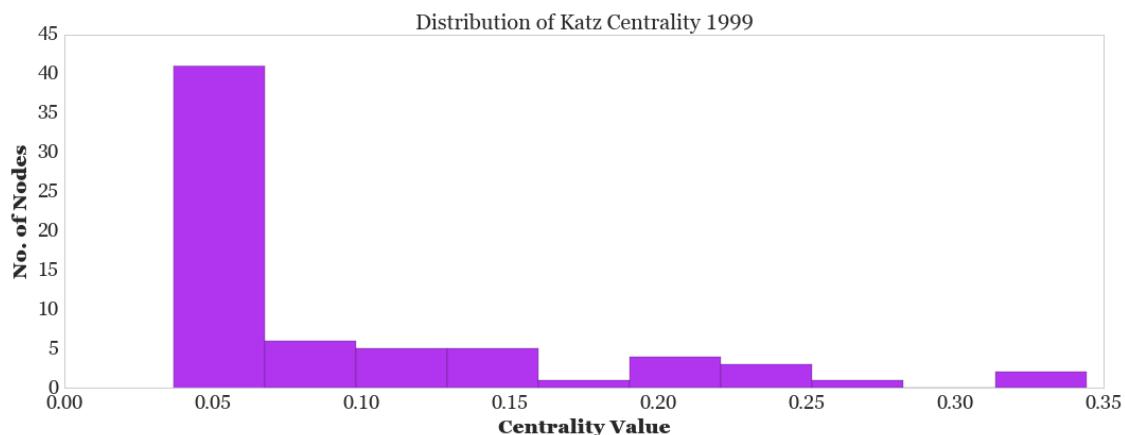
```
In [637]: get_centhistval(katzC0,'Katz Centrality 1998')
get_centhistval(katzC1,'Katz Centrality 1999')
```

```
get_centhistval(katzC2, 'Katz Centrality 2000')
get_centhistval(katzC3, 'Katz Centrality 2001')
get_centhistval(katzC4, 'Katz Centrality 2002')
```

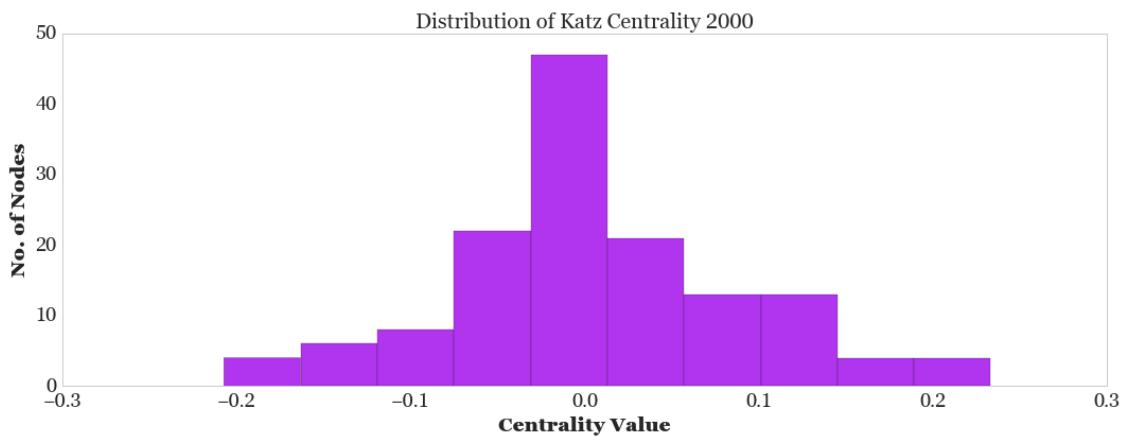
<matplotlib.figure.Figure at 0x163367bca58>



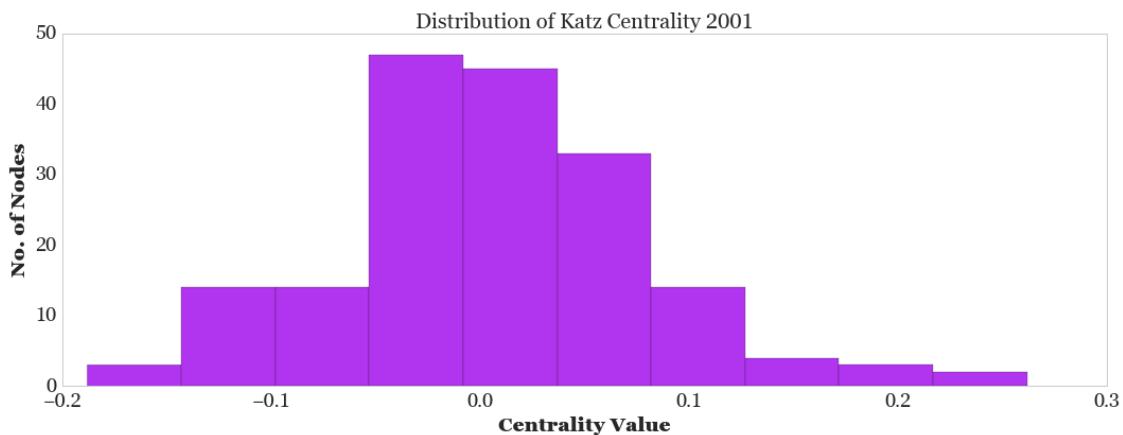
<matplotlib.figure.Figure at 0x1631187e160>



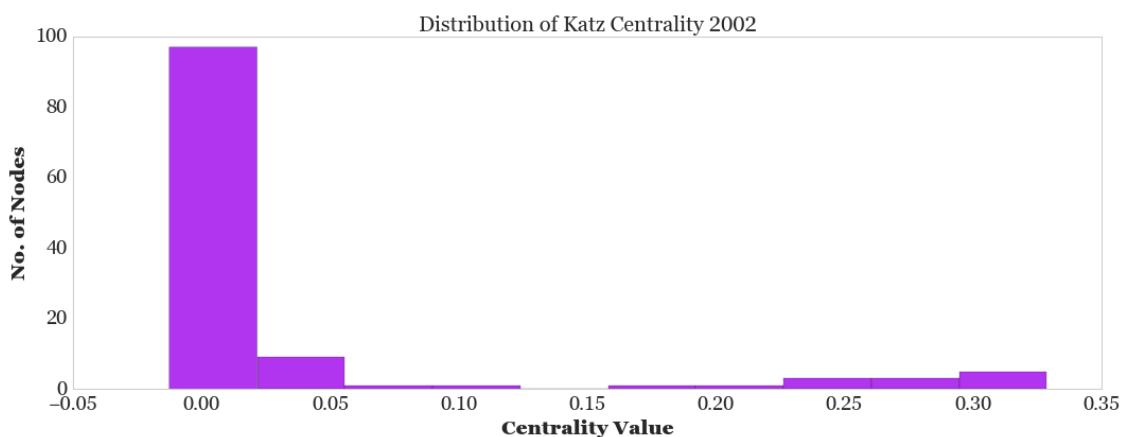
<matplotlib.figure.Figure at 0x16321e7af60>



<matplotlib.figure.Figure at 0x1632cc0d668>



<matplotlib.figure.Figure at 0x1632a874cc0>



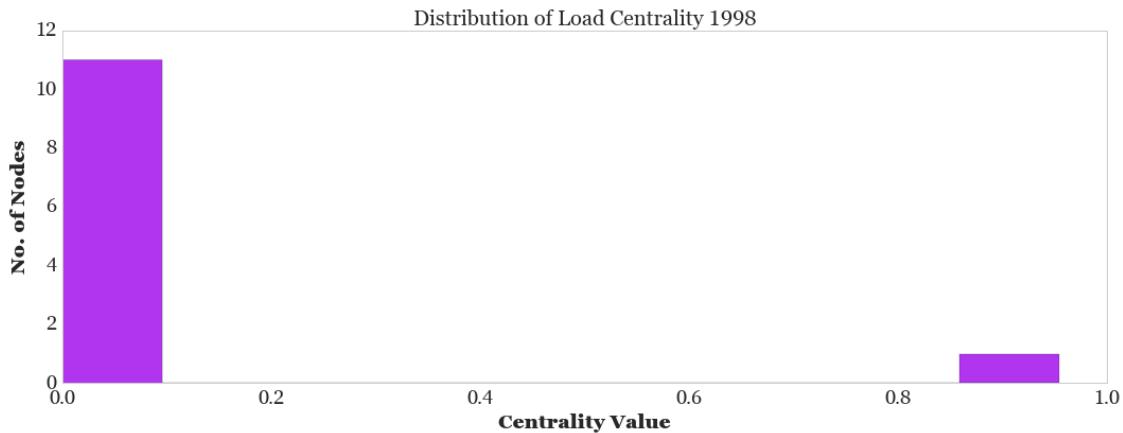
```
In [638]: print("Top 5 Katz Centrality Nodes year 98")
get_top_keys(katzC0,5)
print("Top 5 Katz Centrality Nodes year 99")
get_top_keys(katzC1,5)
print("Top 5 Katz Centrality Nodes year 2k")
get_top_keys(katzC2,5)
print("Top 5 Katz Centrality Nodes year 2k1")
get_top_keys(katzC3,5)
print("Top 5 Katz Centrality Nodes year 2k2")
get_top_keys(katzC4,5)

Top 5 Katz Centrality Nodes year 98
114
112
65
145
155
Top 5 Katz Centrality Nodes year 99
114
169
38
155
112
Top 5 Katz Centrality Nodes year 2k
114
169
33
158
155
Top 5 Katz Centrality Nodes year 2k1
167
133
32
153
21
Top 5 Katz Centrality Nodes year 2k2
98
103
124
158
108
```

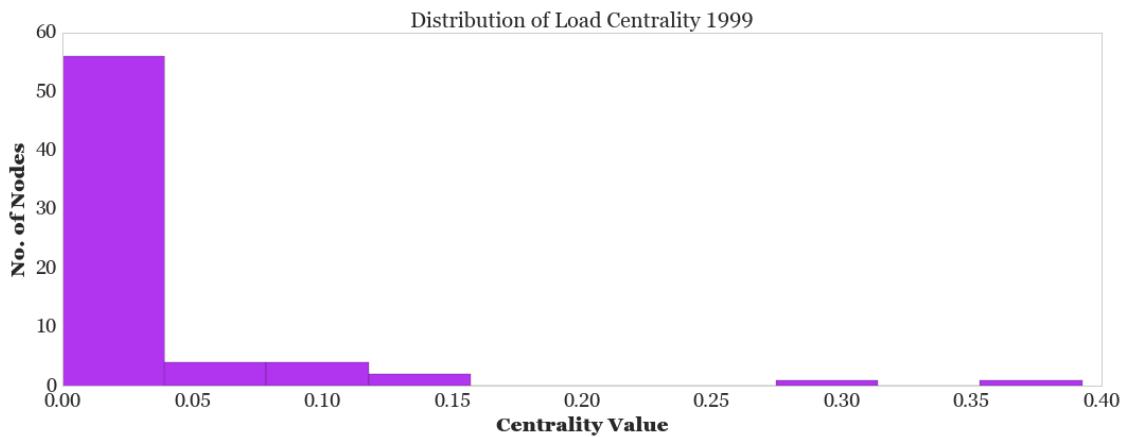
```
In [639]: get_centhistval(loadC0,'Load Centrality 1998')
get_centhistval(loadC1,'Load Centrality 1999')
```

```
get_centhistval(loadC2, 'Load Centrality 2000')
get_centhistval(loadC3, 'Load Centrality 2001')
get_centhistval(loadC4, 'Load Centrality 2002')
```

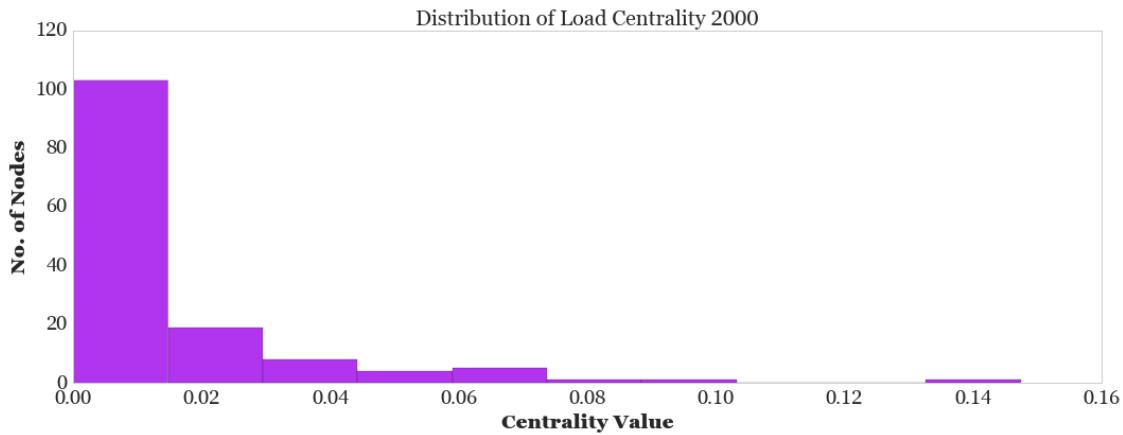
<matplotlib.figure.Figure at 0x163367bc390>



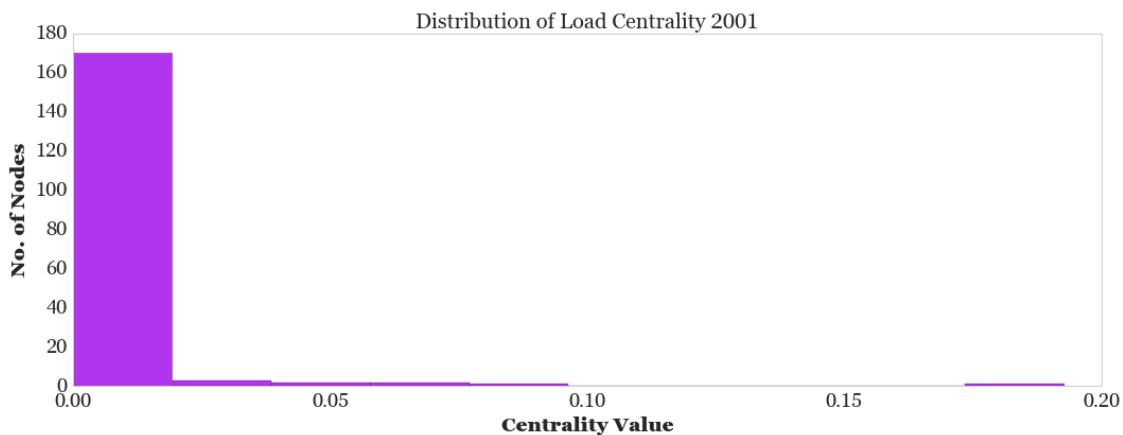
<matplotlib.figure.Figure at 0x1632a6bbf28>



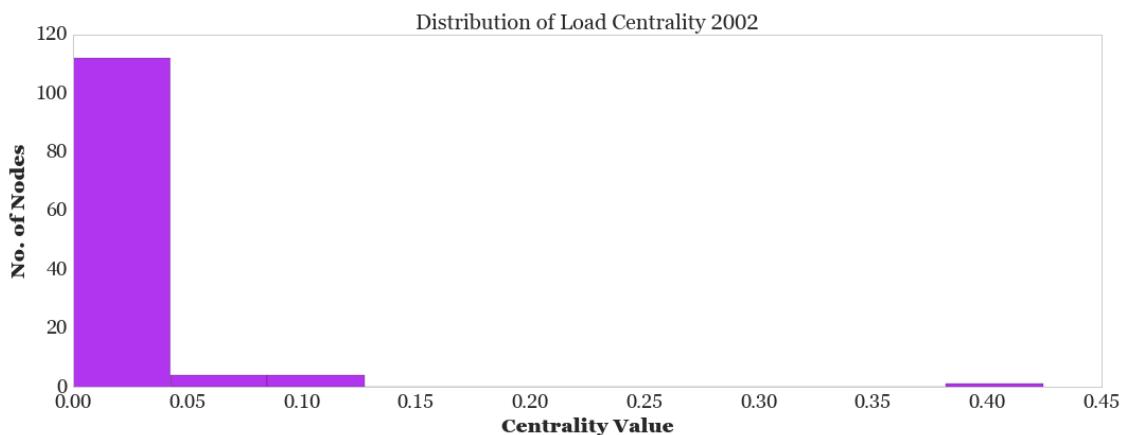
<matplotlib.figure.Figure at 0x16315a36780>



<matplotlib.figure.Figure at 0x1632aaf3a58>



<matplotlib.figure.Figure at 0x1632ad58ac8>



```
In [640]: print("Top 5 Load Centrality Nodes year 98")
get_top_keys(loadC0,5)
print("Top 5 Load Centrality Nodes year 99")
get_top_keys(loadC1,5)
print("Top 5 Load Centrality Nodes year 2k")
get_top_keys(loadC2,5)
print("Top 5 Load Centrality Nodes year 2k1")
get_top_keys(loadC3,5)
print("Top 5 Load Centrality Nodes year 2k2")
get_top_keys(loadC4,5)

Top 5 Load Centrality Nodes year 98
114
112
160
65
155
Top 5 Load Centrality Nodes year 99
153
169
17
163
22
Top 5 Load Centrality Nodes year 2k
82
63
140
79
125
Top 5 Load Centrality Nodes year 2k1
82
153
94
107
9
Top 5 Load Centrality Nodes year 2k2
105
34
107
158
17
```

```
In [641]: all_year_G =tuple([Gt0,Gt1,Gt2,Gt3,Gt4])
all_year_G[:5]
```

```
Out[641]: (<networkx.classes.graph.Graph at 0x1630f76acc0>,
             <networkx.classes.graph.Graph at 0x1630f767198>,
             <networkx.classes.graph.Graph at 0x1630f767208>,
             <networkx.classes.graph.Graph at 0x1630f7672e8>,
             <networkx.classes.graph.Graph at 0x1630f767d30>)
```

```
In [642]: density_year = []
avclustcoff_year = []
algebconn_year = []

for i in all_year_G:
    density_year.append(nx.density(i))
    avclustcoff_year.append(nx.average_clustering(i))
    algebconn_year.append(nx.algebraic_connectivity(i))
```

```
In [643]: density_year
```

```
Out[643]: [0.19696969696969696,
           0.058384547848990345,
           0.07981220657276995,
           0.11179461427405687,
           0.06666666666666667]
```

```
In [644]: avclustcoff_year
```

```
Out[644]: [0.22525252525252523,
           0.21497313000340684,
           0.4493480384358036,
           0.5037725950614431,
           0.4292464564453092]
```

```
In [645]: algebconn_year
```

```
Out[645]: [1.0000000000000002, 0.0, 0.12190804467954658, 0.0, 0.0]
```

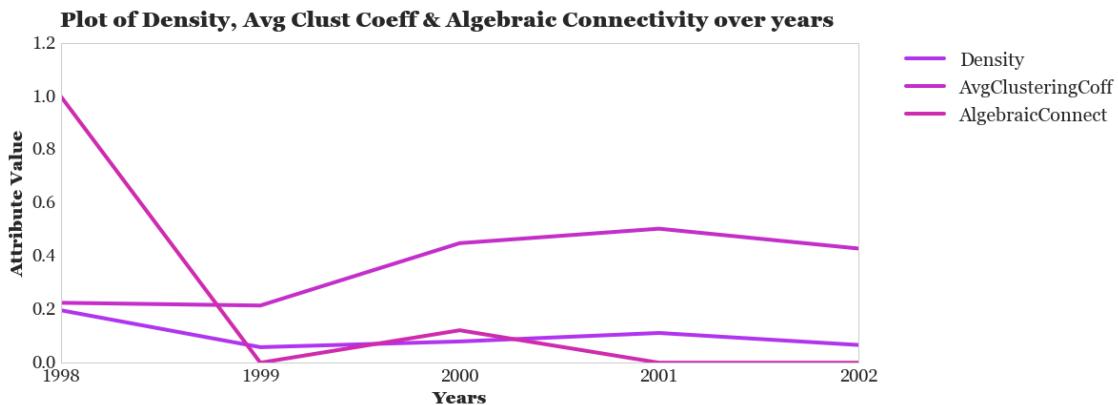
```
In [646]: assorstat_year = pd.DataFrame([density_year, avclustcoff_year, algebconn_year])
assorstat_year.columns = ['Density', 'AvgClusteringCoff', 'AlgebraicConnect'
assorstat_year = assorstat_year.T
assorstat_year.columns = years[1:]
assorstat_year = assorstat_year.T
assorstat_year
```

```
Out[646]:      Density  AvgClusteringCoff  AlgebraicConnect
1998    0.196970          0.225253        1.000000
1999    0.058385          0.214973        0.000000
2000    0.079812          0.449348        0.121908
2001    0.111795          0.503773        0.000000
2002    0.066667          0.429246        0.000000
```

```
In [647]: assorstat_year.plot(fontsize=18, use_index=True)
plt.suptitle("Plot of Density, Avg Clust Coeff & Algebraic Connectivity over years", fontsize=18)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.5)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=18)
plt.yticks(fontsize=18)

Out[647]: (array([ 0.,  0.2,  0.4,  0.6,  0.8,  1.,  1.2,  1.4]),  

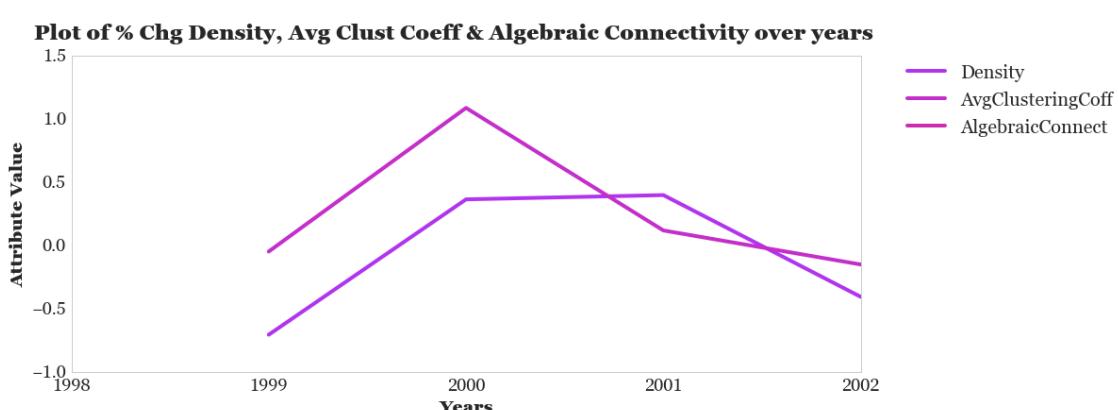
 <a list of 8 Text yticklabel objects>)
```



```
In [648]: assorstat_year.pct_change().plot(fontsize=18, use_index=True)
plt.suptitle("% Chg Density, Avg Clust Coeff & Algebraic Connectivity over years", fontsize=18)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.5)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=18)
plt.yticks(fontsize=18)

Out[648]: (array([-1., -0.5,  0.,  0.5,  1.,  1.5]),  

 <a list of 6 Text yticklabel objects>)
```



```
In [649]: print("No. of edges 2000 and 2001: \n", nx.number_of_edges(Gt2),nx.number_of_edges(Gt3))
print("No. of nodes 2000 and 2001: \n", nx.number_of_nodes(Gt2),nx.number_of_nodes(Gt3))
```

No. of edges 2000 and 2001:

799 1781

No. of nodes 2000 and 2001:

142 179

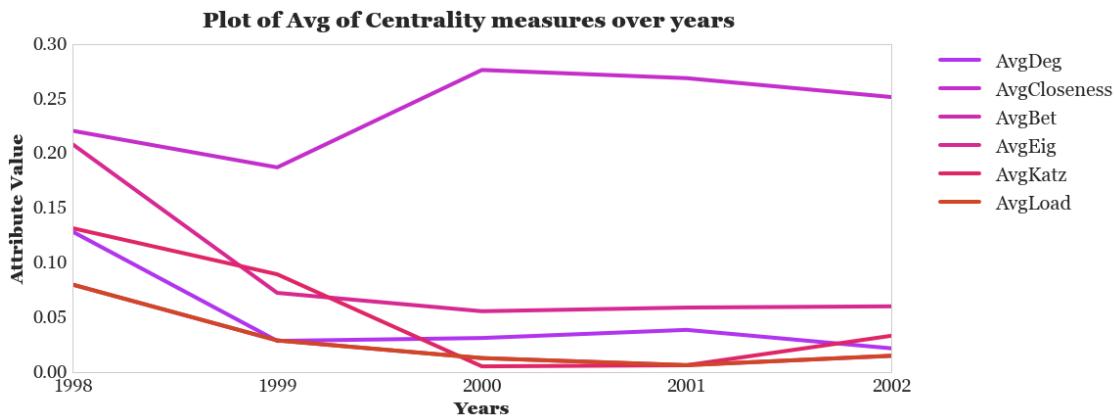
5.2 Calculate average of attributes for yearly networks

```
In [650]: y_avgst0 = cal_avgstat(Gt0)
y_avgst1 = cal_avgstat(Gt1)
y_avgst2 = cal_avgstat(Gt2)
y_avgst3 = cal_avgstat(Gt3)
y_avgst4 = cal_avgstat(Gt4)
```

```
y_avgst_all = y_avgst0.append(y_avgst1).append(y_avgst2).append(y_avgst3)
y_avgst_all = y_avgst_all.T
y_avgst_all.columns = years[1:]
y_avgst_all = y_avgst_all.T
```

```
In [651]: y_avgst_all.iloc[:, :6].plot(fontsize=18)
plt.suptitle("Plot of Avg of Centrality measures over years", fontsize=22)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=18)
plt.yticks(fontsize=18)
```

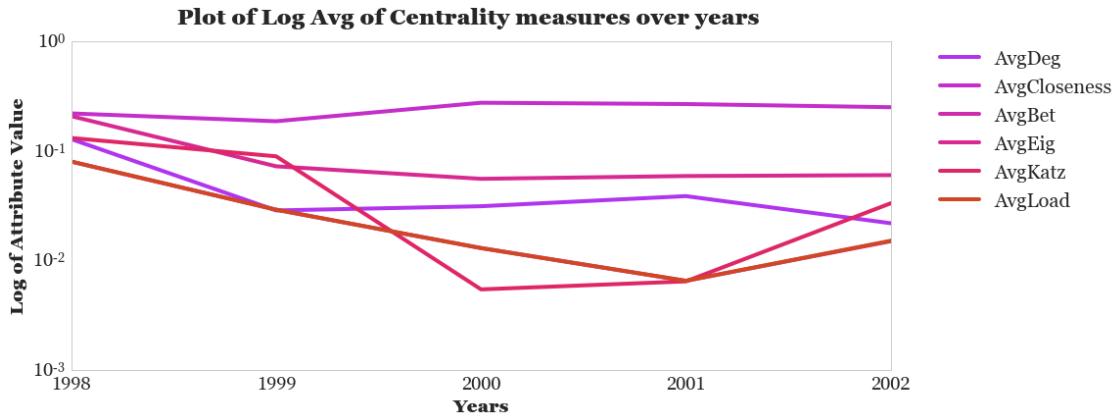
```
Out[651]: (array([ 0. ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,  0.35]),  
<a list of 8 Text yticklabel objects>)
```



```
In [652]: y_avgst_all.iloc[:, :6].plot(fontsize=18, logy=True)
plt.suptitle("Plot of Log Avg of Centrality measures over years", fontsize=18)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Log of Attribute Value", fontsize=18)
plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=18)
plt.yticks(fontsize=18)

Out[652]: (array([ 1.00000000e-04,   1.00000000e-03,   1.00000000e-02,
       1.00000000e-01,   1.00000000e+00,   1.00000000e+01]),  

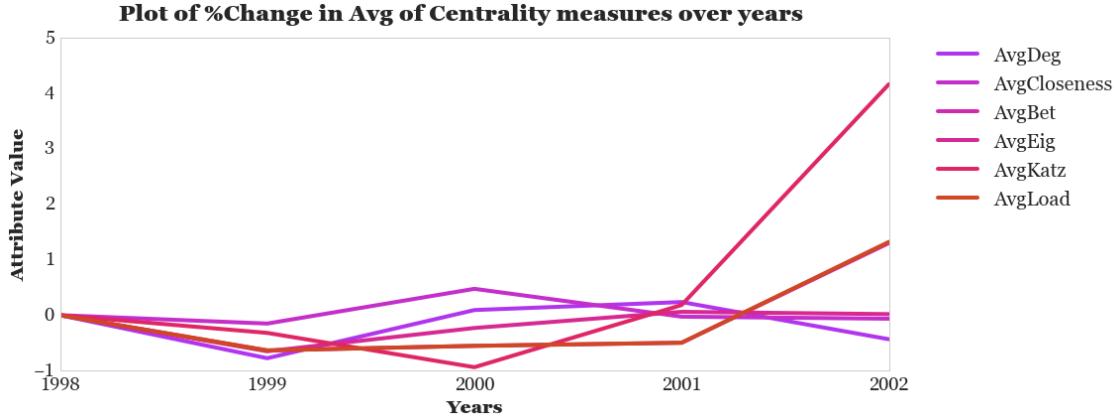
<a list of 6 Text yticklabel objects>)
```



```
In [653]: y_avgst_all.iloc[:, :6].pct_change().fillna(0).plot(fontsize=18)
plt.suptitle("Plot of %Change in Avg of Centrality measures over years", fontsize=18)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=18)
plt.yticks(fontsize=18)

Out[653]: (array([-1.,  0.,  1.,  2.,  3.,  4.,  5.]),  

<a list of 7 Text yticklabel objects>)
```



5.3 Observations

The algebraic connectivity > 0 indicates that the network is connected so excluding the first network the only other time that this measure indicates that the network is connected is in 2000. This is the first signal that emerges from these set of measures. From the percentage change plot we see that the year 2000 experiences the largest increase in density and clustering coefficient. The density implies that more edges that could be formed are formed in the network in 2000 as opposed to the other years. While the density is largely similar between 2000 and 2001 the local average clustering coefficient decreases between 2000 and 2001 this could largely be down to the larger network size. This means that although the network has grown the number of triangles has decreased due to much higher number of edges between nodes.

From the average of the centrality measures we see a similar signal across the years. We see that 2000 again experiences a big change compared to the previous years. The Closeness Centrality dominates due to its scale but in the log plot we see that the Closeness Centrality changes the most for this year before almost flat lining for the remaining years. The Degree centrality confirms the observations regarding the density because we see only marginal change in the Degree Centrality between 2000 and 2001. The Eigenvector Centrality tells us that the power structure or the influence structure in the network did not change much within the network over this time period. However, the Katz centrality and Load centrality are much more interesting. The Katz centrality is an indicator of node importance and from 2000 it seems that there was an increase in the number average number of influential nodes and the path between them decreased as indicated by the falling Load centrality. They coincide in 2001 just at the end of the Golden age of Enron. The rising Katz and Load Centrality can be interpreted as the involvement of outside actors as the accounting scandal unfolded leading to more influential players in the network. Hence Load and Katz Centrality experience a largest percentage increase during this time.

5.4 Monthly Analysis

Calculate average statistics for monthly networks

```
In [654]: stat_nov98 = cal_avgstat(G_nov98)
stat_dec98 = cal_avgstat(G_dec98)
```

```

stat98= stat_nov98.append(stat_dec98).T
stat98.columns = ['Nov98', 'Dec98']
stat98.head()

```

Out [654] :

| | Nov98 | Dec98 |
|--------------|----------|----------|
| AvgDeg | 0.142857 | 0.145455 |
| AvgCloseness | 0.192308 | 0.242737 |
| AvgBet | 0.125000 | 0.086869 |
| AvgEig | 0.188611 | 0.198999 |
| AvgKatz | 0.185435 | 0.175111 |

In [655] :

```

stat_jan_99=cal_avgstat(G_jan_99)
stat_feb_99=cal_avgstat(G_feb_99)
stat_mar_99=cal_avgstat(G_mar_99)
stat_apr_99=cal_avgstat(G_apr_99)
stat_may_99=cal_avgstat(G_may_99)
stat_jun_99=cal_avgstat(G_jun_99)
stat_jul_99=cal_avgstat(G_jul_99)
stat_aug_99=cal_avgstat(G_aug_99)
stat_sep_99=cal_avgstat(G_sep_99)
stat_oct_99=cal_avgstat(G_oct_99)
stat_nov_99=cal_avgstat(G_nov_99)
stat_dec_99=cal_avgstat(G_dec_99)

```

```

stat_99 = stat_jan_99.append(stat_feb_99).append(stat_mar_99).append(stat_apr_99)
stat_99 = stat_99.append(stat_jun_99).append(stat_jul_99).append(stat_aug_99)
stat_99 = stat_99.append(stat_sep_99).append(stat_nov_99).append(stat_dec_99)
stat_99.columns = ['jan_99', 'feb_99', 'mar_99', 'apr_99', 'may_99', 'jun_99',
stat_99.head()

```

Out [655] :

| | jan_99 | feb_99 | mar_99 | apr_99 | may_99 | jun_99 |
|--------------|----------|----------|----------|----------|----------|----------|
| AvgDeg | 0.083333 | 0.500000 | 0.200000 | 0.500000 | 0.120879 | 0.066176 |
| AvgCloseness | 0.117057 | 0.555556 | 0.259259 | 0.555556 | 0.372644 | 0.157004 |
| AvgBet | 0.076923 | 0.333333 | 0.166667 | 0.333333 | 0.110806 | 0.028922 |
| AvgEig | 0.132902 | 0.569036 | 0.275965 | 0.569036 | 0.224884 | 0.166311 |
| AvgKatz | 0.057692 | 0.390223 | 0.277684 | 0.390223 | 0.263068 | 0.191024 |
| | jul_99 | aug_99 | sep_99 | oct_99 | nov_99 | dec_99 |
| AvgDeg | 0.094771 | 0.088745 | 0.071146 | 0.070769 | 0.085714 | 0.025689 |
| AvgCloseness | 0.295189 | 0.223778 | 0.275119 | 0.169032 | 0.179564 | 0.147367 |
| AvgBet | 0.100490 | 0.057143 | 0.077734 | 0.028803 | 0.051128 | 0.038836 |
| AvgEig | 0.189606 | 0.140039 | 0.152945 | 0.136185 | 0.147848 | 0.069981 |
| AvgKatz | 0.221168 | 0.183912 | 0.189669 | 0.172447 | 0.194931 | 0.106213 |

In [656] :

```

stat_jan_2k=cal_avgstat(G_jan_2k)
stat_feb_2k=cal_avgstat(G_feb_2k)
stat_mar_2k=cal_avgstat(G_mar_2k)
stat_apr_2k=cal_avgstat(G_apr_2k)
stat_may_2k=cal_avgstat(G_may_2k)

```

```
stat_jun_2k=cal_avgstat(G_jun_2k)
stat_jul_2k=cal_avgstat(G_jul_2k)
stat_aug_2k=cal_avgstat(G_aug_2k)
stat_sep_2k=cal_avgstat(G_sep_2k)
stat_oct_2k=cal_avgstat(G_oct_2k)
stat_nov_2k=cal_avgstat(G_nov_2k)
stat_dec_2k=cal_avgstat(G_dec_2k)
```

```
In [657]: stat_2k = stat_jan_2k.append(stat_feb_2k).append(stat_mar_2k).append(stat_apr_2k)
stat_2k = stat_2k.append(stat_jun_2k).append(stat_jul_2k).append(stat_aug_2k)
stat_2k = stat_2k.append(stat_oct_2k).append(stat_nov_2k).append(stat_dec_2k)
stat_2k.columns = ['jan_2k', 'feb_2k', 'mar_2k', 'apr_2k', 'may_2k', 'jun_2k', 'jul_2k', 'aug_2k', 'sep_2k', 'oct_2k', 'nov_2k', 'dec_2k']
stat_2k.head()
```

```
Out[657]:
```

| | jan_2k | feb_2k | mar_2k | apr_2k | may_2k | jun_2k |
|--------------|----------|----------|----------|----------|----------|----------|
| AvgDeg | 0.022378 | 0.026626 | 0.014809 | 0.010464 | 0.014035 | 0.010867 |
| AvgCloseness | 0.165628 | 0.208593 | 0.157985 | 0.113773 | 0.164460 | 0.192022 |
| AvgBet | 0.034948 | 0.035326 | 0.046531 | 0.042264 | 0.033091 | 0.027877 |
| AvgEig | 0.068894 | 0.083170 | 0.065712 | 0.053698 | 0.072435 | 0.068502 |
| AvgKatz | 0.099556 | 0.104652 | 0.097354 | 0.103359 | 0.104188 | 0.091210 |

| | jul_2k | aug_2k | sep_2k | oct_2k | nov_2k | dec_2k |
|--------------|----------|----------|----------|----------|----------|----------|
| AvgDeg | 0.012697 | 0.016514 | 0.012147 | 0.018086 | 0.010884 | 0.013887 |
| AvgCloseness | 0.206035 | 0.223131 | 0.192691 | 0.222388 | 0.177589 | 0.174585 |
| AvgBet | 0.026787 | 0.017815 | 0.019417 | 0.019777 | 0.017789 | 0.019199 |
| AvgEig | 0.068268 | 0.062787 | 0.061723 | 0.062449 | 0.054661 | 0.051451 |
| AvgKatz | 0.086094 | 0.068437 | 0.073747 | 0.064164 | 0.058620 | 0.038943 |

```
In [658]: stat_jan_2k1=cal_avgstat(G_jan_2k1)
stat_feb_2k1=cal_avgstat(G_feb_2k1)
stat_mar_2k1=cal_avgstat(G_mar_2k1)
stat_apr_2k1=cal_avgstat(G_apr_2k1)
stat_may_2k1=cal_avgstat(G_may_2k1)
stat_jun_2k1=cal_avgstat(G_jun_2k1)
stat_jul_2k1=cal_avgstat(G_jul_2k1)
stat_aug_2k1=cal_avgstat(G_aug_2k1)
stat_sep_2k1=cal_avgstat(G_sep_2k1)
stat_oct_2k1=cal_avgstat(G_oct_2k1)
stat_nov_2k1=cal_avgstat(G_nov_2k1)
stat_dec_2k1=cal_avgstat(G_dec_2k1)

stat_2k1 = stat_jan_2k1.append(stat_feb_2k1).append(stat_mar_2k1).append(stat_apr_2k1).append(stat_may_2k1).append(stat_jun_2k1).append(stat_jul_2k1).append(stat_aug_2k1).append(stat_sep_2k1).append(stat_oct_2k1).append(stat_nov_2k1).append(stat_dec_2k1)
stat_2k1.columns = ['jan_2k1', 'feb_2k1', 'mar_2k1', 'apr_2k1', 'may_2k1', 'jun_2k1', 'jul_2k1', 'aug_2k1', 'sep_2k1', 'oct_2k1', 'nov_2k1', 'dec_2k1']
stat_2k1.head()
```

```
Out[658]:          jan_2k1    feb_2k1    mar_2k1    apr_2k1    may_2k1    jun_2k1
AvgDeg      0.013179  0.012052  0.012982  0.015607  0.012401  0.008815
```

| | | | | | | |
|--------------|----------|----------|----------|----------|----------|----------|
| AvgCloseness | 0.191559 | 0.208446 | 0.186307 | 0.214056 | 0.221824 | 0.209019 |
| AvgBet | 0.017710 | 0.021867 | 0.017525 | 0.017928 | 0.011548 | 0.019936 |
| AvgEig | 0.055170 | 0.064934 | 0.058781 | 0.051721 | 0.054499 | 0.049033 |
| AvgKatz | 0.048244 | 0.074620 | 0.057339 | 0.002995 | 0.022108 | 0.059015 |
| | jul_2k1 | aug_2k1 | sep_2k1 | oct_2k1 | nov_2k1 | dec_2k1 |
| AvgDeg | 0.016036 | 0.016639 | 0.012930 | 0.019379 | 0.016612 | 0.011789 |
| AvgCloseness | 0.199306 | 0.247621 | 0.195912 | 0.232163 | 0.241737 | 0.202481 |
| AvgBet | 0.020539 | 0.015186 | 0.021407 | 0.012743 | 0.017048 | 0.022688 |
| AvgEig | 0.053400 | 0.051352 | 0.051611 | 0.058342 | 0.056969 | 0.065293 |
| AvgKatz | 0.046579 | 0.000057 | 0.021531 | 0.003949 | 0.002392 | 0.074214 |

```
In [659]: stat_jan_2k2=cal_avgstat(G_jan_2k2)
stat_feb_2k2=cal_avgstat(G_feb_2k2)
stat_mar_2k2=cal_avgstat(G_mar_2k2)
stat_apr_2k2=cal_avgstat(G_apr_2k2)
stat_may_2k2=cal_avgstat(G_may_2k2)
stat_jun_2k2=cal_avgstat(G_jun_2k2)

stat_2k2 = stat_jan_2k2.append(stat_feb_2k2).append(stat_mar_2k2).append(stat_apr_2k2).append(stat_may_2k2).append(stat_jun_2k2)
stat_2k2.columns = ['jan_2k2', 'feb_2k2', 'mar_2k2', 'apr_2k2', 'may_2k2', 'jun_2k2']
stat_2k2.head()
```

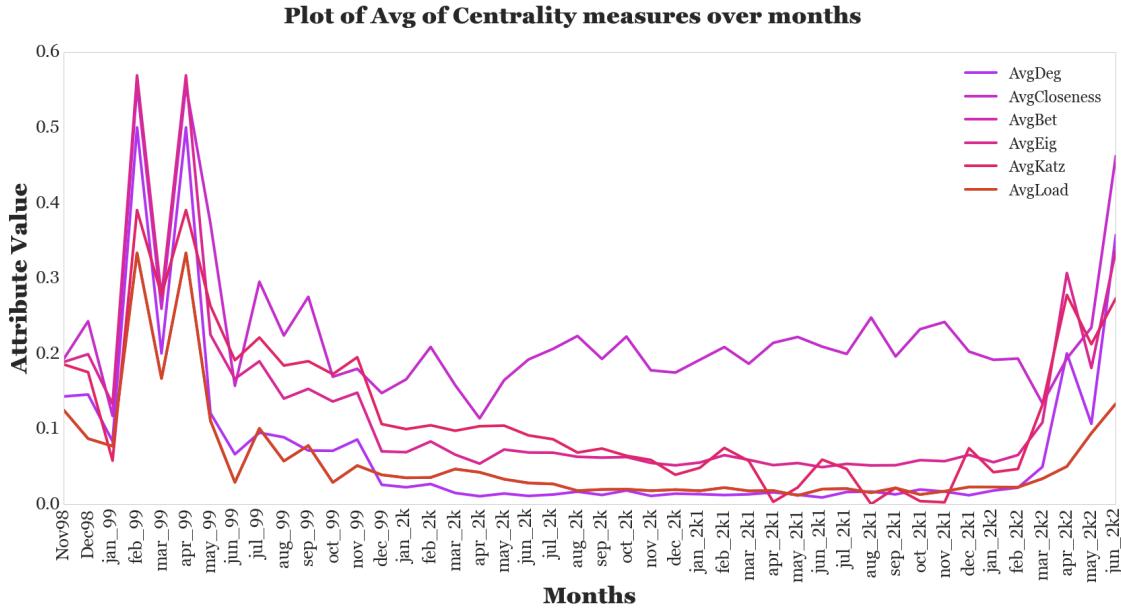
```
Out[659]:      jan_2k2    feb_2k2    mar_2k2    apr_2k2    may_2k2    jun_2k2
AvgDeg      0.018018  0.021818  0.049395  0.200000  0.106618  0.357143
AvgCloseness 0.191514  0.193146  0.133727  0.193333  0.234243  0.462193
AvgBet       0.022664  0.022488  0.033737  0.050000  0.094118  0.133333
AvgEig        0.055595  0.065151  0.108396  0.306719  0.180674  0.336275
AvgKatz       0.042359  0.046586  0.131887  0.277666  0.212198  0.273203
```

```
In [660]: stat_all = stat98.join(stat_99).join(stat_2k).join(stat_2k1).join(stat_2k2)
```

```
In [661]: months = list(stat_all.index)
```

```
In [662]: stat_all.iloc[:, :6].plot(fontsize=22, rot=90, figsize=(28,12))
plt.suptitle("Plot of Avg of Centrality measures over months", fontsize=30)
plt.xlabel("Months", fontsize=33)
plt.ylabel("Attribute Value", fontsize=33)
plt.legend(fontsize=23, loc=1)
# plt.legend(fontsize=20, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[662]: (array([ 0.,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7]),<a list of 8 Text yticklabel objects>)
```

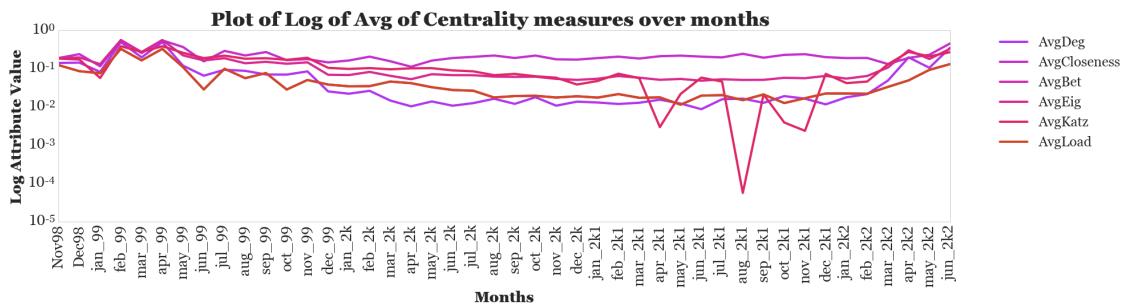
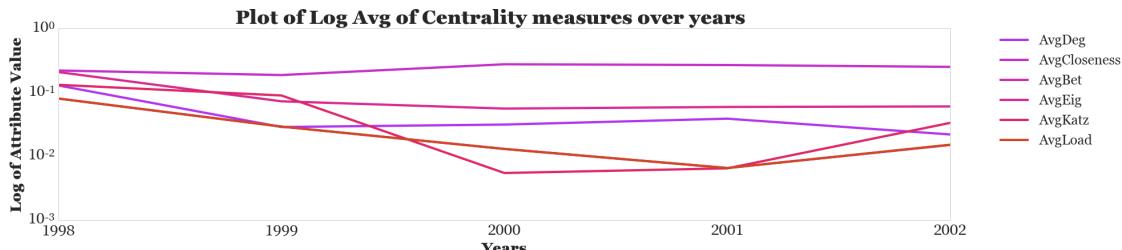


5.5 Comparison of Yearly and Monthly trends in Centrality

```
In [663]: y_avgst_all.iloc[:, :6].plot(fontsize=22, figsize=(28, 6), logy=True)
plt.suptitle("Plot of Log Avg of Centrality measures over years", fontsize=24)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Log of Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

stat_all.iloc[:, :6].plot(fontsize=22, rot=90, figsize=(28, 6), logy=True)
plt.suptitle("Plot of Log of Avg of Centrality measures over months", fontsize=24)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Log Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

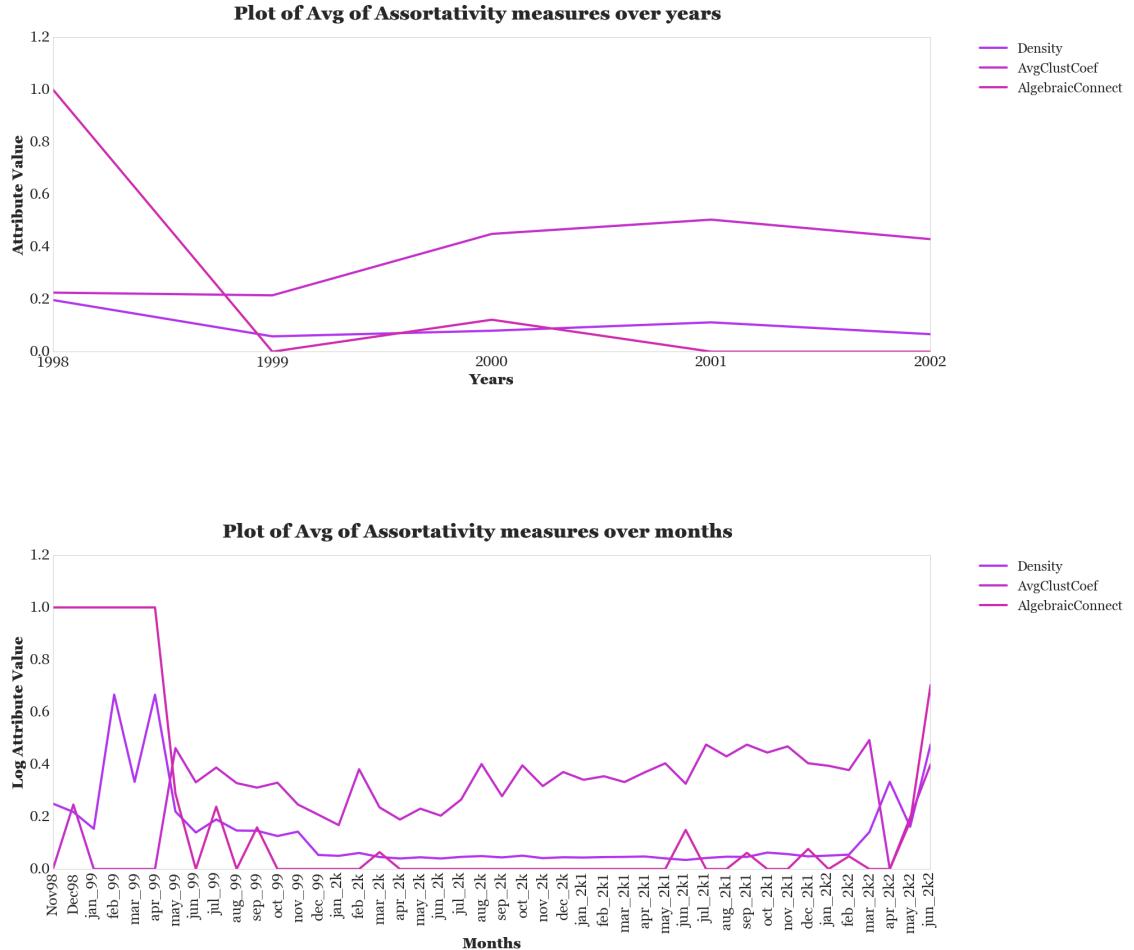
Out[663]: (array([
  1.00000000e-06,   1.00000000e-05,   1.00000000e-04,
  1.00000000e-03,   1.00000000e-02,   1.00000000e-01,
  1.00000000e+00,   1.00000000e+01]), <a list of 8 Text yticklabel objects>)
```



```
In [664]: y_avgst_all.iloc[:,6:9].plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Avg of Assortativity measures over years", fontsize=24)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

stat_all.iloc[:,6:9].plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Avg of Assortativity measures over months", fontsize=24)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Log Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[664]: (array([ 0.,  0.2,  0.4,  0.6,  0.8,  1.,  1.2,  1.4]),  
<a list of 8 Text yticklabel objects>)
```



6 Attributes Analysis on Graphs

Here I implement novel attributes taken from seismic data analysis.

The complex trace is defined as

$$C(t) = S(t) + iH(t)$$

Where:

$C(t)$ = Complex Trace

$S(t)$ = Real Data

$H(t)$ = Hilbert Transform of data

Complex Trace Attributes:

1. Instantaneous Amplitude, IA

$$\sqrt{S(t)^2 + H(t)^2}$$

2. Instantaneous Phase, IP

$$\arctan\left(\frac{H(t)}{S(t)}\right)$$

3. Instantatneous Frequency, IF

$$\frac{d}{dt}(IP))$$

4. Derivative of IA

$$\frac{d}{dt}(IA)$$

5. 2nd Derivative of IA

$$\frac{d^2}{dt^2}(IA)$$

6. Instantaneous Acceleration, IAcc

$$\frac{d}{dt}(IF)$$

7. Amplitude Weighted Instantaneous Phase

$$IA * IP$$

8. Amplitude Weighted Instantaneous Frequency

$$IA * IF$$

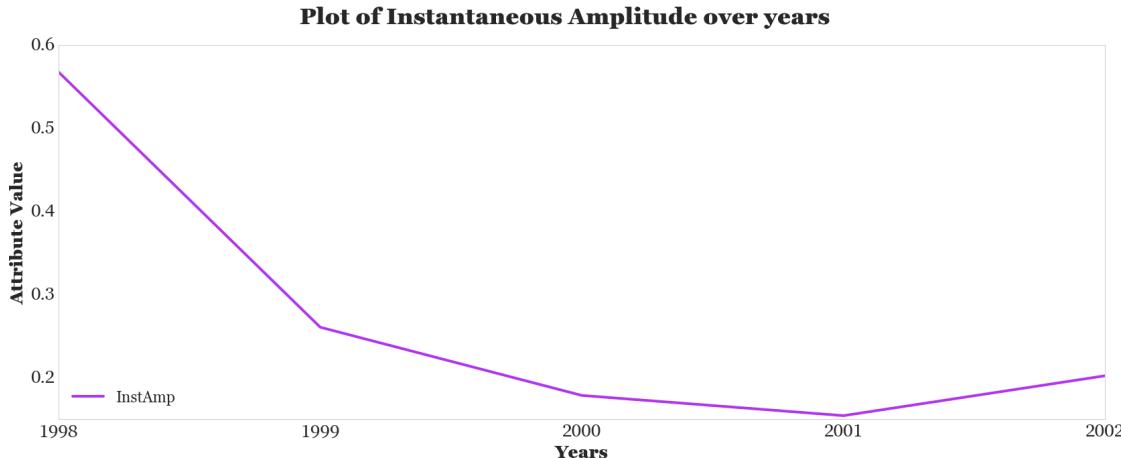
Attributes from the normalised Graph Laplacian:

- Gaussian Curvature
- Kernel PCA 3 Component Ratio Change
- Resistane Distance
- Stationarity Ratio

6.1 Instantaneous Amplitude

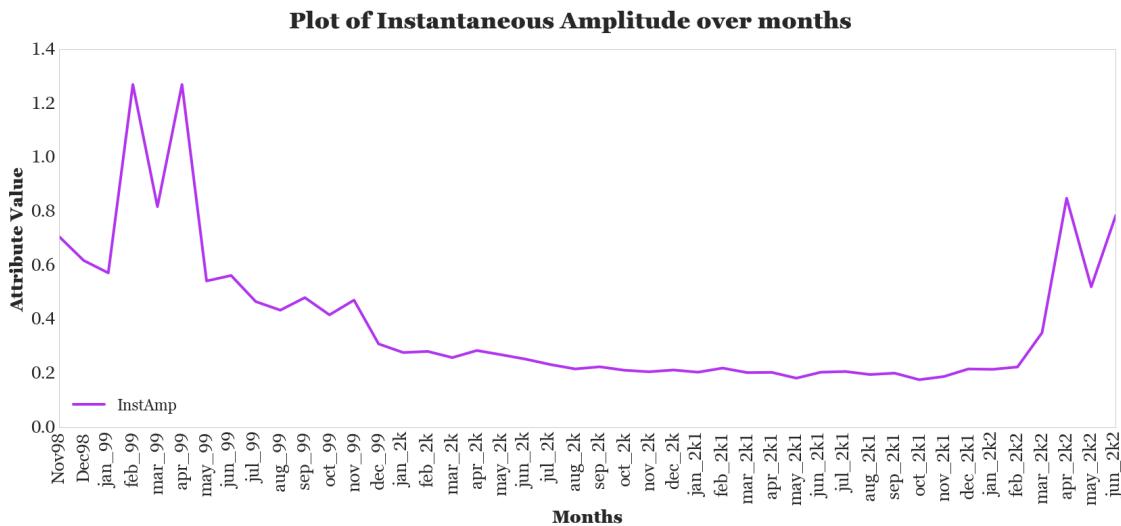
```
In [665]: y_avgst_all.InstAmp.plot(fontsize=22, figsize=(28,10))
          plt.suptitle("Plot of Instantaneous Amplitude over years", fontsize=33)
          plt.xlabel("Years", fontsize=25)
          plt.ylabel("Attribute Value", fontsize=25)
          plt.legend(fontsize=23, loc=3)
          plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
          plt.yticks(fontsize=26)

Out[665]: (array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7]),  
 <a list of 7 Text yticklabel objects>)
```



```
In [666]: stat_all.InstAmp.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Amplitude over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

Out[666]: (array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  1.2,  1.4,  1.6]),<a list of 9 Text yticklabel objects>)
```



6.2 Instantaneous Frequency

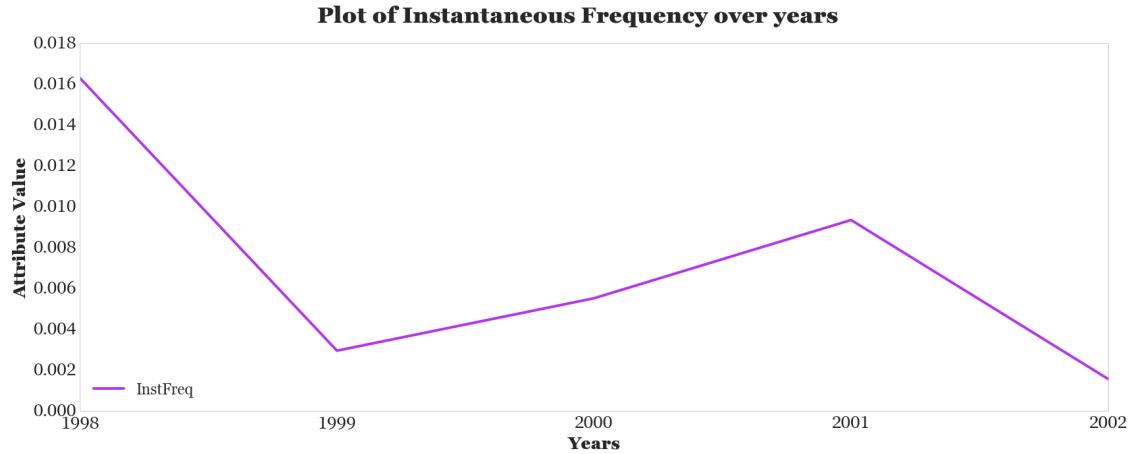
```
In [667]: y_avgst_all.InstFreq.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Frequency over years", fontsize=33)
```

```

plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[667]: (array([ 0. , 0.002, 0.004, 0.006, 0.008, 0.01 , 0.012, 0.014,
       0.016, 0.018]), <a list of 10 Text yticklabel objects>

```

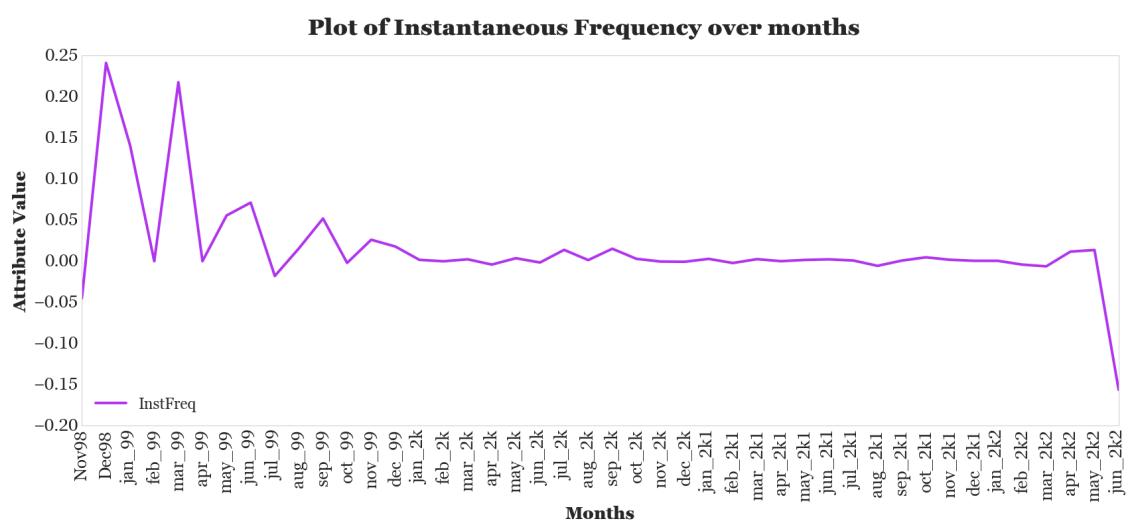


```

In [668]: stat_all.InstFreq.plot(fontsize=22, rot=90, figsize=(28,10))
          plt.suptitle("Plot of Instantaneous Frequency over months", fontsize=33)
          plt.xlabel("Months", fontsize=25)
          plt.ylabel("Attribute Value", fontsize=25)
          plt.legend(fontsize=23, loc=3)
          plt.xticks(np.arange(len(months)), months, fontsize=26)
          plt.yticks(fontsize=26)

Out[668]: (array([-0.2 , -0.15, -0.1 , -0.05, 0. , 0.05, 0.1 , 0.15, 0.2 ,
       0.25]), <a list of 10 Text yticklabel objects>

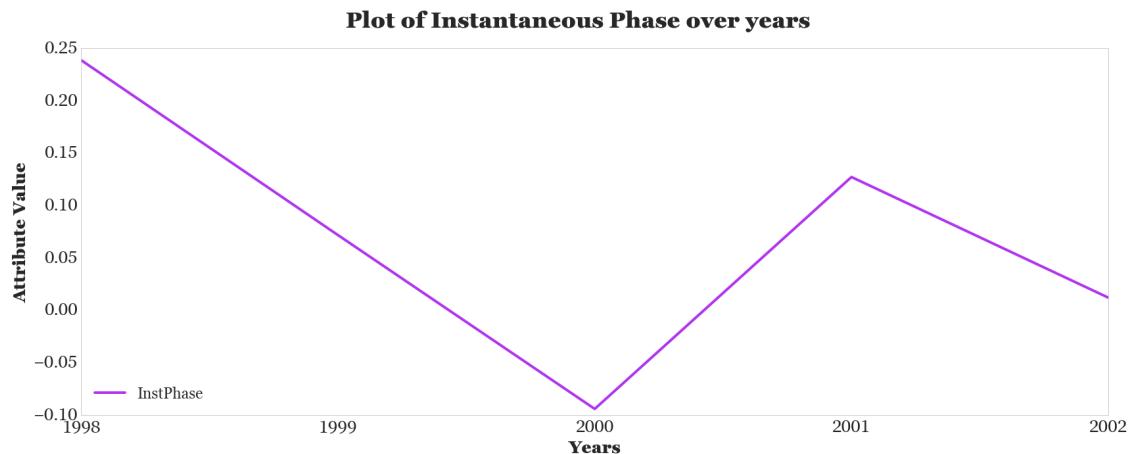
```



6.3 Instantaneous Phase

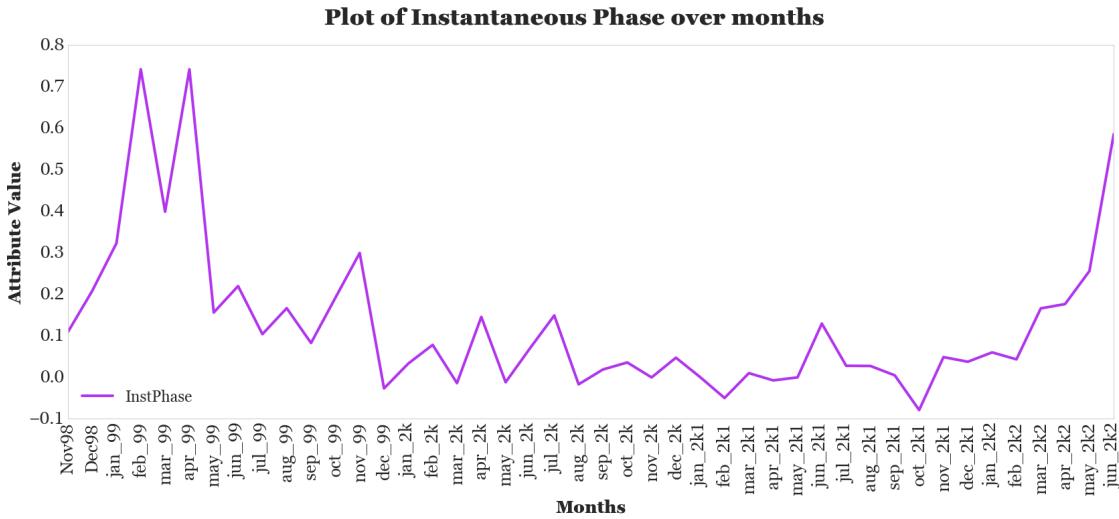
```
In [669]: y_avgst_all.InstPhase.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Phase over years", fontsize=33)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[669]: (array([-0.1 , -0.05,  0. ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ]),
<a list of 9 Text yticklabel objects>)
```



```
In [670]: stat_all.InstPhase.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Phase over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

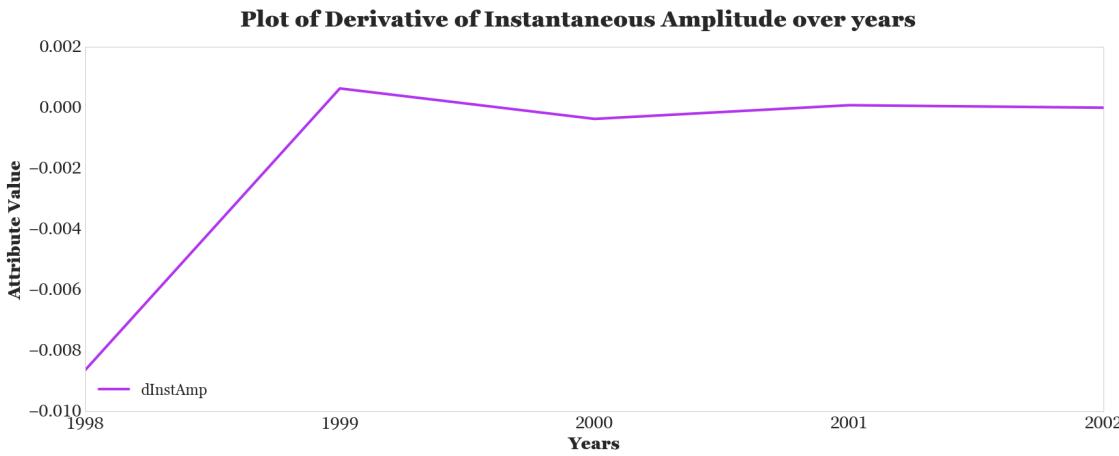
Out[670]: (array([-0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8]),
<a list of 10 Text yticklabel objects>)
```



6.4 Derivative of IA

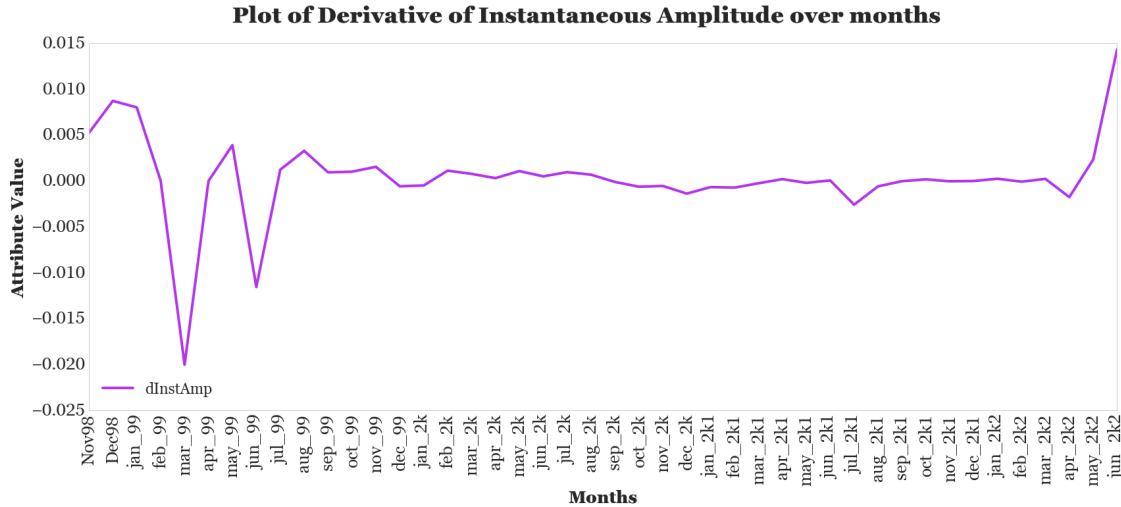
```
In [671]: y_avgst_all.dInstAmp.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Derivative of Instantaneous Amplitude over years",
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[671]: (array([-0.01 , -0.008, -0.006, -0.004, -0.002,  0.   ,  0.002]),  
<a list of 7 Text yticklabel objects>)
```



```
In [672]: stat_all.dInstAmp.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Derivative of Instantaneous Amplitude over months",
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

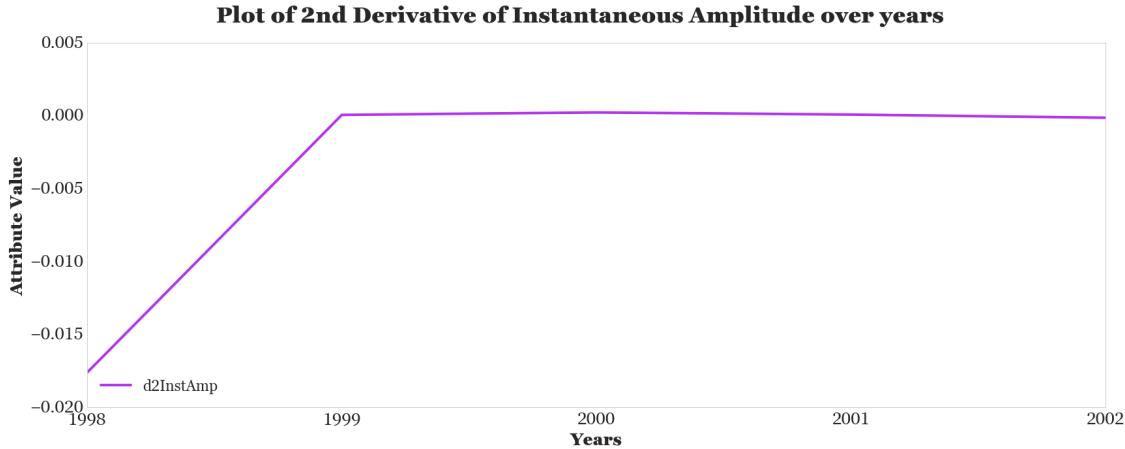
Out[672]: (array([-0.03 , -0.025, -0.02 , -0.015, -0.01 , -0.005,  0.    ,  0.005,
       0.01 ,  0.015]), <a list of 10 Text yticklabel objects>)
```



6.5 2nd Derivative of IA

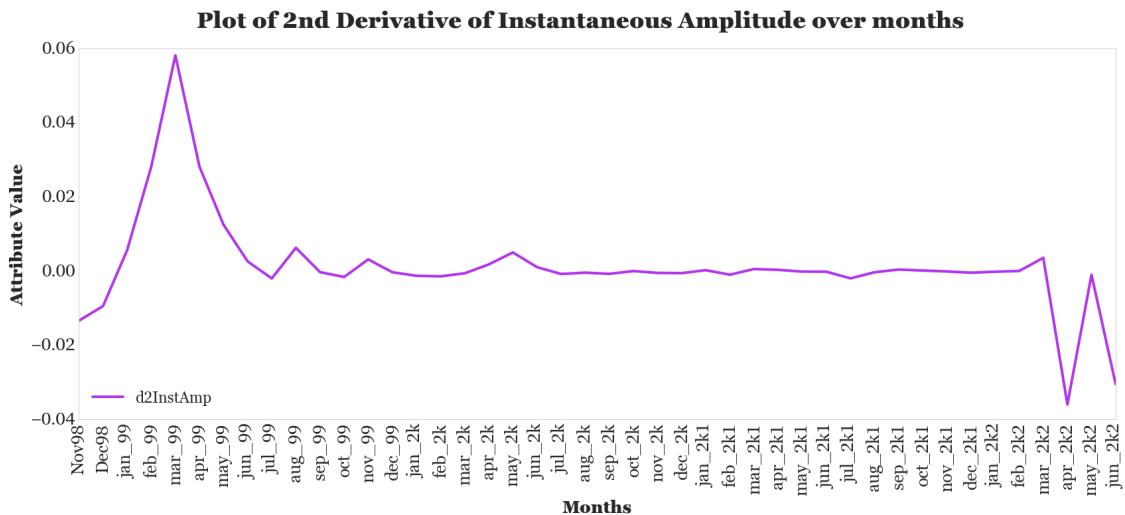
```
In [673]: y_avgst_all.d2InstAmp.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of 2nd Derivative of Instantaneous Amplitude over years",
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[673]: (array([-0.02 , -0.015, -0.01 , -0.005,  0.    ,  0.005,  0.01 ]),
<a list of 7 Text yticklabel objects>)
```



```
In [674]: stat_all.d2InstAmp.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of 2nd Derivative of Instantaneous Amplitude over months", fontsize=25)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

Out[674]: (array([-0.04, -0.02,  0. ,  0.02,  0.04,  0.06,  0.08]),<a list of 7 Text yticklabel objects>)
```



6.6 Instantaneous Acceleration

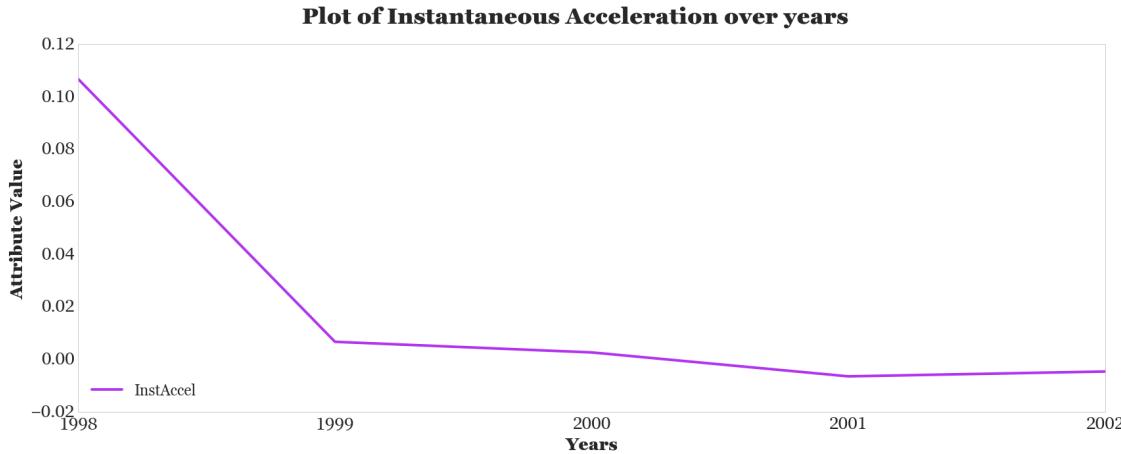
```
In [675]: y_avgst_all.InstAccel.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Acceleration over years", fontsize=30)
```

```

plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

```

Out [675]: (array([-0.02, 0., 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14]),
<a list of 9 Text yticklabel objects>)

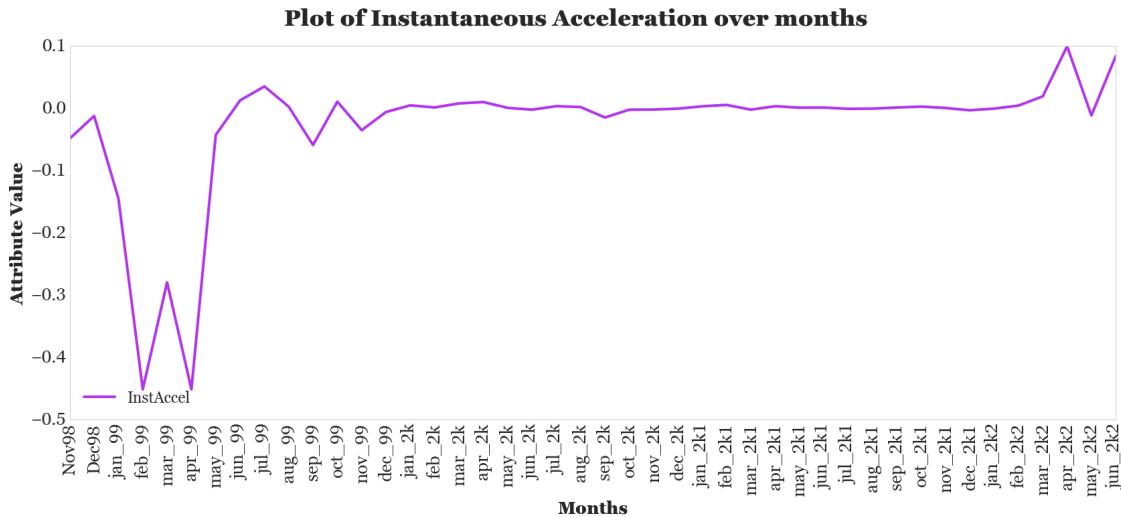


```

In [676]: stat_all.InstAccel.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Instantaneous Acceleration over months", fontsize=30)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

```

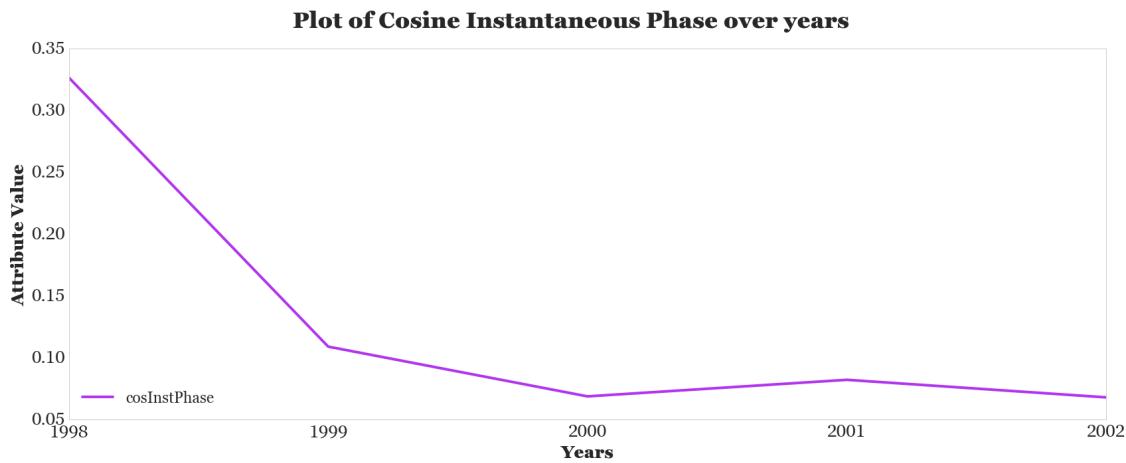
Out [676]: (array([-0.5, -0.4, -0.3, -0.2, -0.1, 0., 0.1, 0.2]),
<a list of 8 Text yticklabel objects>)



6.7 Cosine of IP

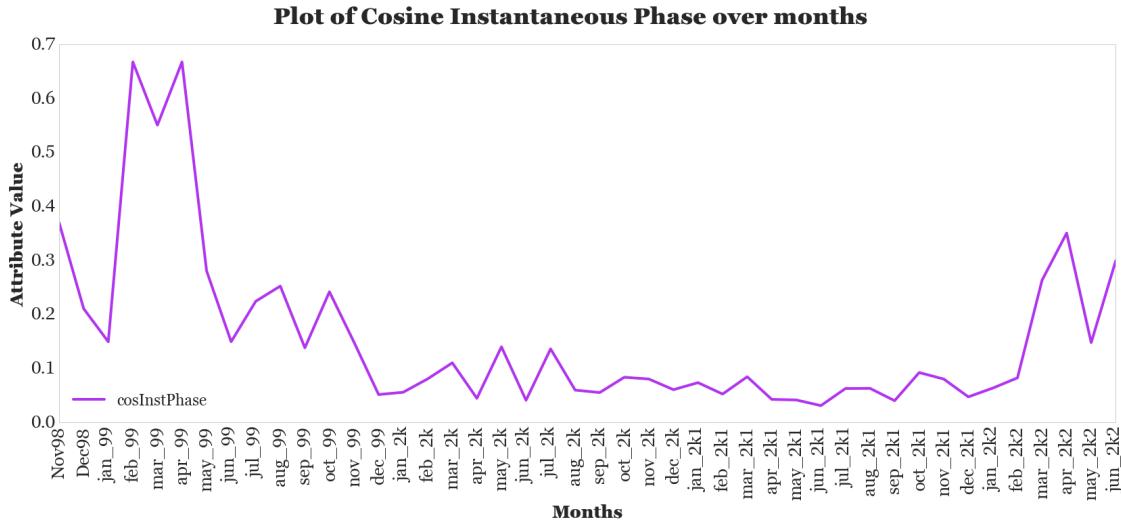
```
In [677]: y_avgst_all.cosInstPhase.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Cosine Instantaneous Phase over years", fontsize=33)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[677]: (array([ 0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,  0.35,  0.4 ]),
<a list of 8 Text yticklabel objects>)
```



```
In [678]: stat_all.cosInstPhase.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Cosine Instantaneous Phase over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

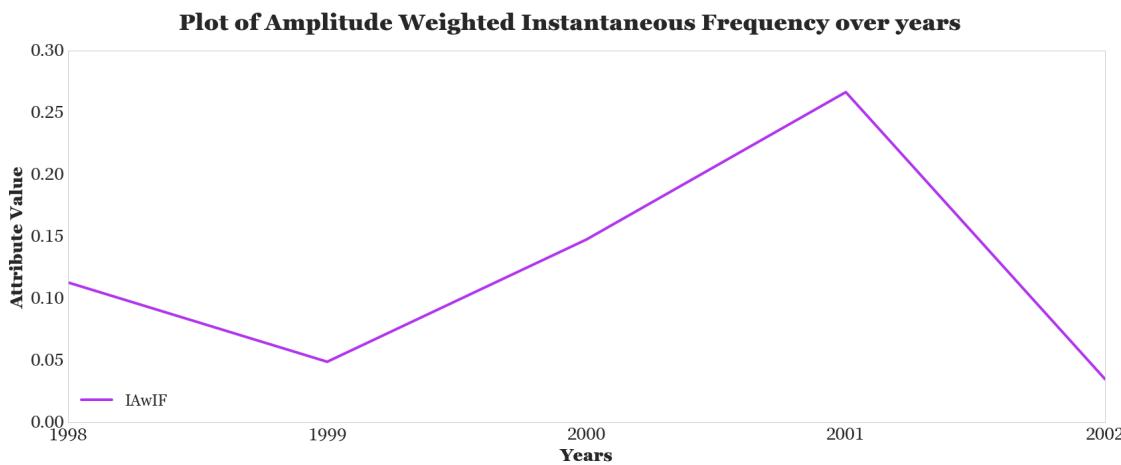
Out[678]: (array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8]),
<a list of 9 Text yticklabel objects>)
```



6.8 Amplitude Weighted IF

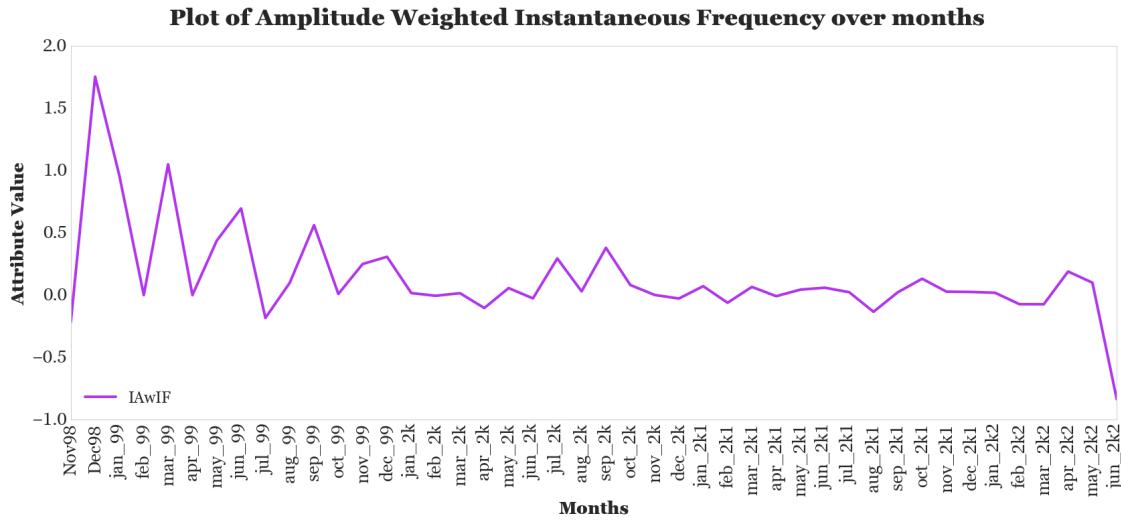
```
In [679]: y_avgst_all.IAwIF.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Amplitude Weighted Instantaneous Frequency over years", color='red', fontsize=25)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)
```

```
Out [679]: (array([ 0. , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35]),  
<a list of 8 Text yticklabel objects>)
```



```
In [680]: stat_all.IAwIF.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Amplitude Weighted Instantaneous Frequency over months")
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

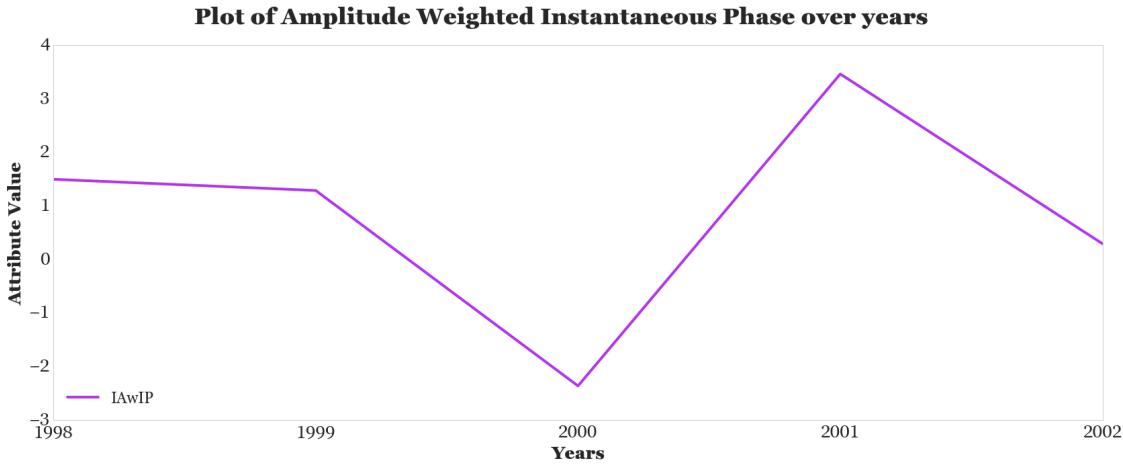
Out[680]: (array([-1. , -0.5,  0. ,  0.5,  1. ,  1.5,  2. ]),
<a list of 7 Text yticklabel objects>)
```



6.9 Amplitude weighted Instantaneous Phase

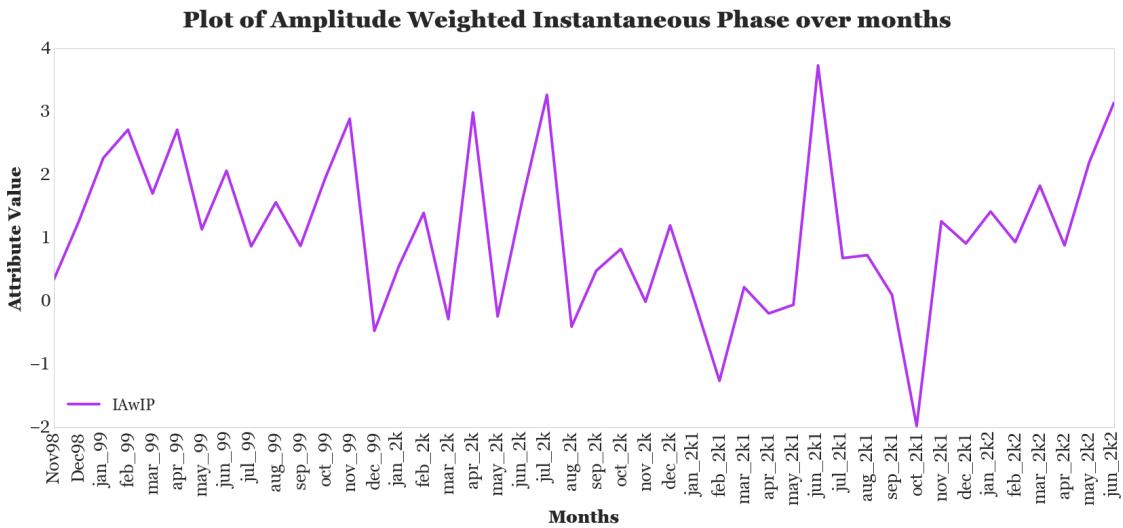
```
In [681]: y_avgst_all.IAwIP.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Amplitude Weighted Instantaneous Phase over years",
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[681]: (array([-3., -2., -1.,  0.,  1.,  2.,  3.,  4.]),
<a list of 8 Text yticklabel objects>)
```



```
In [682]: stat_all.IAwIP.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Amplitude Weighted Instantaneous Phase over months",
             fontsize=25)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

Out[682]: (array([-2., -1.,  0.,  1.,  2.,  3.,  4.]),
            <a list of 7 Text yticklabel objects>)
```



6.10 Mean Curvature

There are many ways to define curvature I use the Gaussian Curvature which is derived from the Hessian Matrix. A surface might be curved upward in places, curve downward in places,

or even be flat in places. Also, at some given point, the surface may be curved upward in some directions and downward in others. The curvature measure helps us detect such change. Gaussian curvature can be positive, negative, or zero. A useful property of the curvature attribute is that it is independent of orientation of the surface. If we place vectors on this surface the would indicates where the curve bends i.e., the vectors are either diverging, converging, or parallel. This can correspond to the negative, positive or zero value of curvature.

$$K = \frac{f_{xx}f_{yy} + f_{xy}f_{yx}}{(1 + f_x^2 + f_y^2)^2}$$

and the mean curvature is

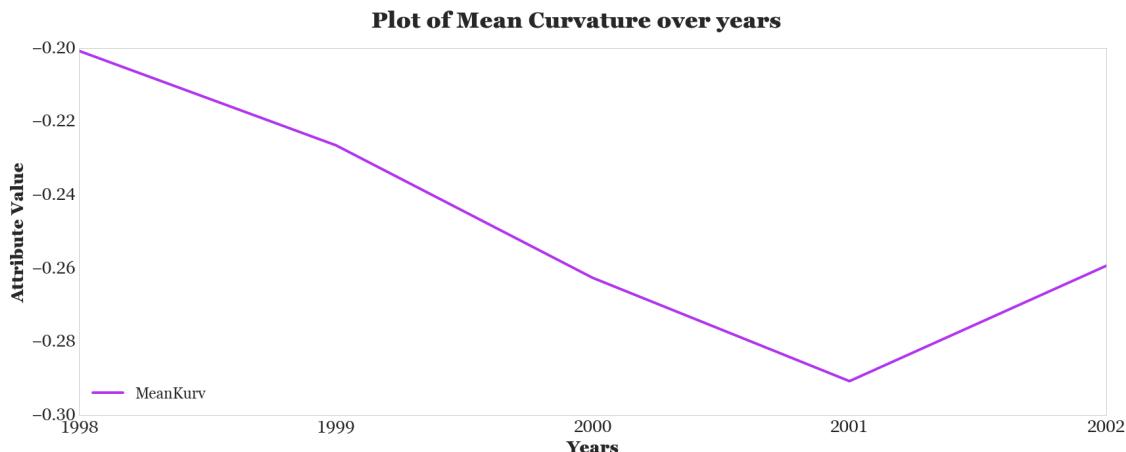
$$K_{mean} = \text{tr}(\lambda_{hessian})$$

the trace of the eigenvalues of the Hessian Matrix.

The Mean Curvature for the network seems to be negative throughout this range which could mean that this network is diverging. This rate of divergence is captured on a yearly and monthly scale in this case by this attribute in its normal form. To get a better sense and interpret this attribute the percent change plot is more useful here we see that the two peaks are well highlighted and the remaining profile is smoothed indicating that the network is not changing very fast more most of the months but the divergence is greater in 1999 during its golden period and in 2002 when the firm went bankrupt. This ties well with what we know from reality and what the other attributes highlight. In contrast to the KPCA we have to note that this highlights the big features as opposed to more subtle features. Curvature is typically used to delineate structures such as faults or other structures such as anticlines (+ curvature) or synclines (- curvature). In the network context it also seems to have a preference for large features and recovers the peaks of the signal very well.

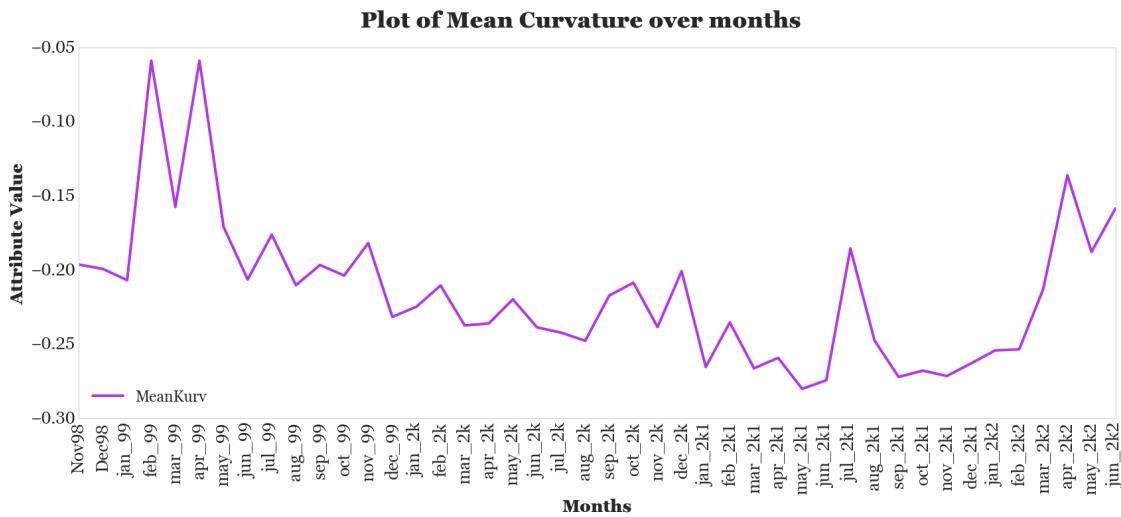
```
In [683]: y_avgst_all.MeanKurv.plot(fontsize=22, figsize=(28,10))
          plt.suptitle("Plot of Mean Curvature over years", fontsize=33)
          plt.xlabel("Years", fontsize=25)
          plt.ylabel("Attribute Value", fontsize=25)
          plt.legend(fontsize=23, loc=3)
          plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
          plt.yticks(fontsize=26)

Out[683]: (array([-0.3 , -0.28, -0.26, -0.24, -0.22, -0.2 , -0.18]), 
           <a list of 7 Text yticklabel objects>)
```



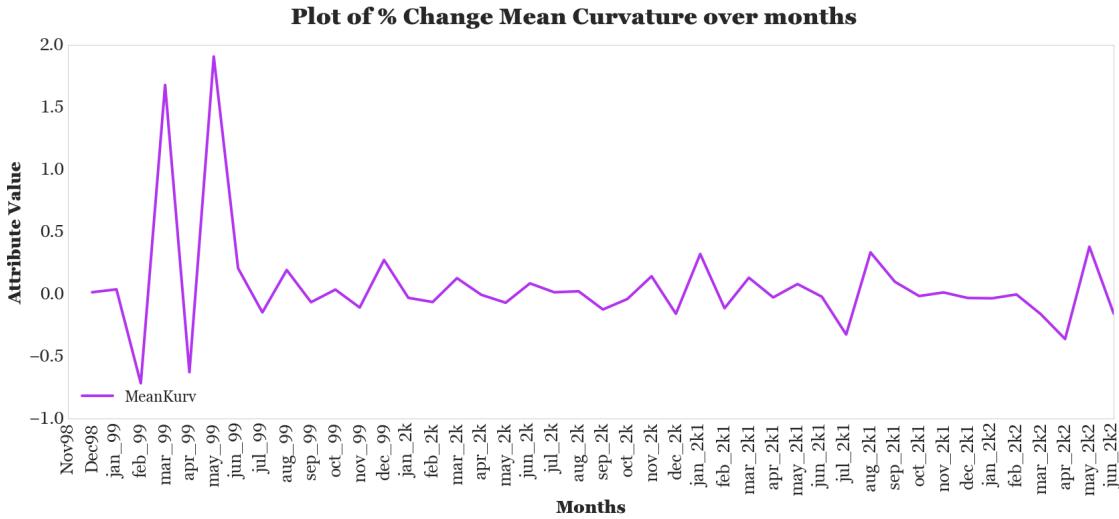
```
In [684]: stat_all.MeanKurv.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Mean Curvature over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[684]: (array([-0.35, -0.3 , -0.25, -0.2 , -0.15, -0.1 , -0.05]),  
<a list of 7 Text yticklabel objects>)
```



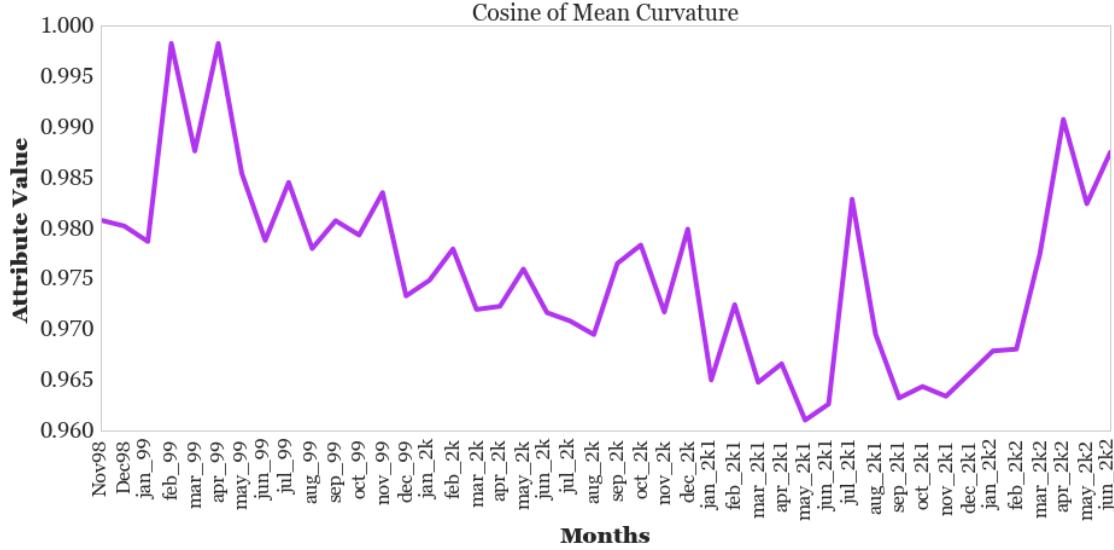
```
In [685]: stat_all.MeanKurv.pct_change().plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of % Change Mean Curvature over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=3)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[685]: (array([-1. , -0.5,  0. ,  0.5,  1. ,  1.5,  2. ]),
<a list of 7 Text yticklabel objects>)
```



```
In [686]: stat_all.MeanKurv.apply(lambda x: np.cos(x)).plot(title="Cosine of Mean Curvature")
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
```

Out[686]: <matplotlib.text.Text at 0x16337da5ba8>



6.11 Kernel PCA 3 Components Ratio

Here I calculate the Kernel PCA 3 Component ratio from the Normalised Graph Laplacian. This is the ratio of the main element difference between two networks. The reason a Kernel PCA is used

is because of its ability to handle non-linearity through kernels. Here the RBF kernel is used to fit the Normalised Graph Laplacian and then the ratio is calculated as follows:

Step 1: Fit and transform the Normalised Graph Laplacian keeping only the first 3 components: PC1, PC2 and PC3

Step 2: Calculate the Kernel PCA Ratio,

$$KPCA_r = \frac{PC1 - PC3}{PC1 - PC2}$$

Step 3: For all the networks calculate the change in the KPCA Ratio as

$$\delta K_r = \frac{K_{rt0}}{K_{rt1}}$$

recursively to derive a rolling measure of the change in KLPCA Ratio over time.

The KLPCA is one of the main element analysis methods in seismic attribute analysis where it is used to calculate the correlation in a multitrace window. A small value represents a degree of intermittent or no correlation of geological phenomena. It is also used to detect discontinuities, such as faults and unconformities. Here the purpose is to locate big discontinuities in our networks over time. This represents a scalable and easy way to locate the biggest changes even when we are dealing with a large number of dynamic networks. Most of the values encountered are relatively small. But from the monthly plot we see that there is a large discontinuity in June and July 1999 whose scale dominates the plot. Hence the log of this attribute is used and we see other smaller signals emerge as a result.

It is encouraging to see that the attribute picks up the Feb-Mar 99 peak like the other attributes but the discontinuity it finds is June/July 99 which are relatively minor peaks according to the other attributes. This could be due to the fact that I use 3 components of the KLPCA to calculate the change as opposed to just the first component. The second and third KPCA components are typically used to find geological unconformities which is a discontinuity in rock sequence indicating interruption of sedimentation, commonly accompanied by erosion of rocks below the break i.e it represents an abrupt change. Hence it can be reasonably assumed that by subtracting these components from the first component we are more likely to find these more abrupt breaks in the data that are not as clear looking at other attributes.

```
In [687]: all_month_G = tuple([G_nov98, G_dec98, G_jan_99, G_feb_99, G_mar_99, G_apr_99,
                           G_nov_99, G_dec_99, G_jan_2k, G_feb_2k, G_mar_2k, G_apr_2k, G_may_2k,
                           G_oct_2k, G_nov_2k, G_dec_2k, G_jan_2k1, G_feb_2k1, G_mar_2k1, G_apr_2k1,
                           G_aug_2k1, G_sep_2k1, G_oct_2k1, G_nov_2k1, G_dec_2k1, G_jan_2k1])
```

```
In [688]: def kPCA_ratio(net):
    from sklearn.decomposition import KernelPCA
    kPCA = KernelPCA(n_components=3, kernel='rbf')
    m = nx.normalized_laplacian_matrix(net).todense()
    X_kPCA = kPCA.fit_transform(m)
    PC1 = X_kPCA[:, 0]
    PC2 = X_kPCA[:, 1]
    PC3 = X_kPCA[:, 2]
    kPCA_rat = norm(PC1 - PC3 / (PC1 - PC2))
    return kPCA_rat
```

```
In [689]: def kpca_att(net):
    kpca_chg = []
    for i in range(len(net)-1):
        x = int(i)
        y = 1+x
        kpcar1= kpca_ratio(net[x])
        kpcar2= kpca_ratio(net[y])
        kpca_chg.append(np.divide(kpcar2,kpcar1))
    kpca_chgpad = np.zeros(len(kpca_chg)+1);
    kpca_chgpad[1:] = kpca_chg

    return kpca_chgpad
```

```
In [690]: kpca_chg_y= kpca_att(all_year_G)
          kpca_chg_m = kpca_att(all_month_G)
```

```
In [691]: kpca_chg_m
```

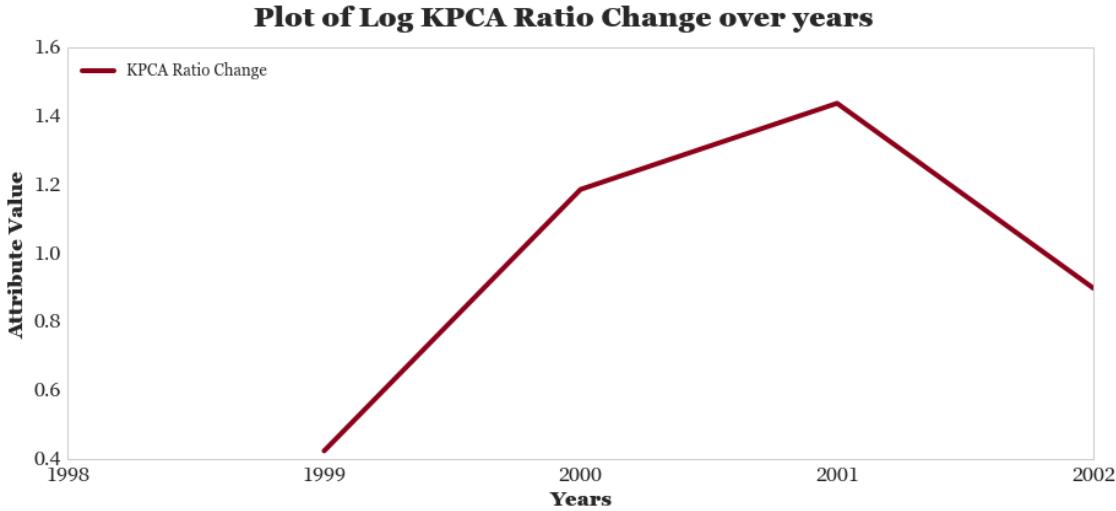
```
Out[691]: array([ 0.0000000e+00, 4.88883814e-01, 3.45460170e+00,
                   8.87338190e-02, 4.98218569e+01, 2.00715120e-02,
                   1.10992264e+01, 1.17738962e+14, 2.32254840e-14,
                   6.41831923e-01, 1.08075430e+01, 2.31947056e-02,
                   1.67524790e+02, 5.02193494e+00, 1.46856694e-03,
                   1.48814197e+01, 5.66322852e-01, 4.25853308e-01,
                   1.12397367e+01, 1.39875121e+01, 3.09181415e-02,
                   1.33596264e+01, 2.87383472e-01, 3.97053182e-01,
                   1.16145289e+00, 7.21630828e-01, 3.22575646e-01,
                   5.95254751e-01, 1.68793140e+00, 2.01282069e+01,
                   3.26933404e-01, 1.86023362e-01, 2.57792631e+01,
                   1.49918679e-01, 3.29429718e+01, 5.12458586e+00,
                   8.46003304e-03, 3.97316868e-01, 1.62273256e+01,
                   1.63227862e-01, 3.55140567e-01, 1.86324914e-02,
                   1.95969629e+00, 2.01185382e+00])
```

```
In [692]: max(kpca_chg_m), min(kpca_chg_m)
```

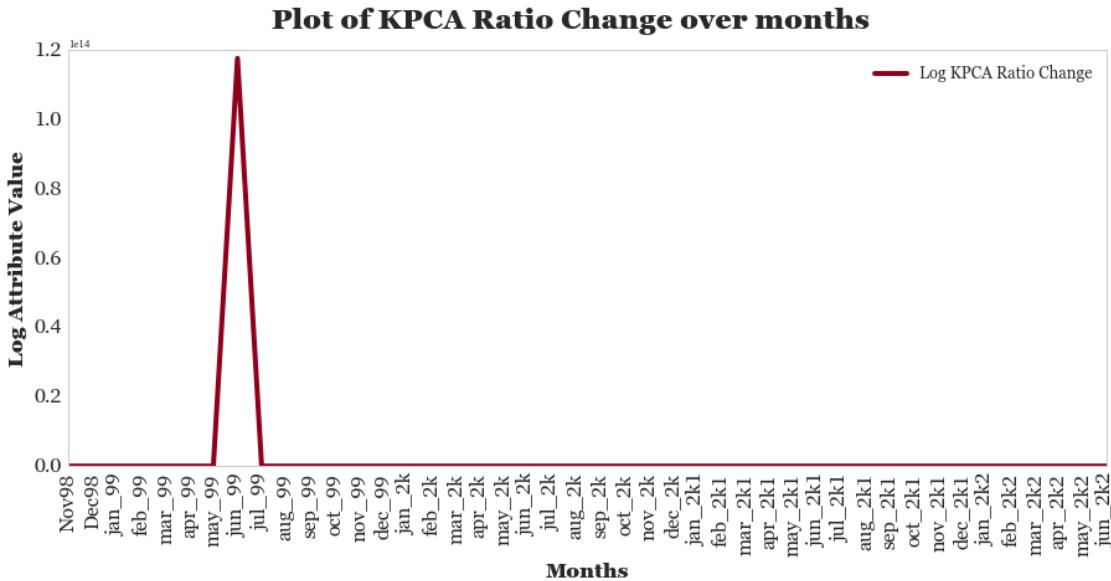
```
Out[692]: (117738962094924.84, 0.0)
```

```
In [693]: plt.plot(np.log(kpca_chg_y), '#8C001A',label='KPCA Ratio Change')
plt.suptitle("Plot of Log KPCA Ratio Change over years", fontsize=22)
plt.xlabel("Years", fontsize=16)
plt.ylabel("Attribute Value", fontsize=16)
plt.legend(fontsize=13, loc=2)
plt.xticks(np.arange(len(years[1:])), years[1:], fontsize=16)
plt.yticks(fontsize=16)
```

```
Out[693]: (array([ 0.4,  0.6,  0.8,  1. ,  1.2,  1.4,  1.6]),  
<a list of 7 Text yticklabel objects>)
```



```
In [694]: plt.plot(kpca_chg_m, '#8C001A', label='Log KPCA Ratio Change')
plt.suptitle("Plot of KPCA Ratio Change over months", fontsize=22)
plt.xlabel("Months", fontsize=16)
plt.ylabel("Log Attribute Value", fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.xticks(np.arange(len(months)), months, fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.autoscale()
```

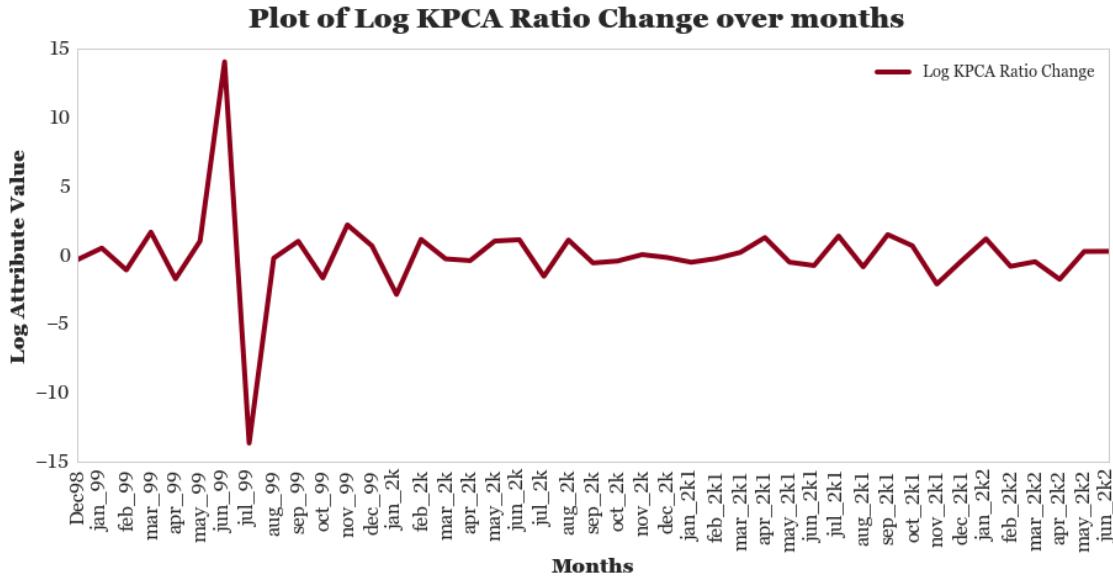


```
In [695]: plt.plot(np.log10(kpca_chg_m), '#8C001A', label='Log KPCA Ratio Change')
plt.suptitle("Plot of Log KPCA Ratio Change over months", fontsize=22)
```

```

plt.xlabel("Months", fontsize=16)
plt.ylabel("Log Attribute Value", fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.xticks(np.arange(len(months)), months, fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.autoscale()

```



```
In [696]: y_avgst_all['LogKPCARatio'] = np.log10(kpca_chg_y)
stat_all['LogKPCARatio'] = np.log10(kpca_chg_m)
```

```
In [697]: stat_all.LogKPCARatio.iloc[0]=0
y_avgst_all.LogKPCARatio.iloc[0]=0
```

7 ICA Ratio

Similarly to the KPCA ratio above I derive a ICA ratio for comparison. Since this another decomposition technique but assumes non-Gaussian structure it should find more local features in contrast to PCA which find smore global features. However, since we used a Kernel PCA with a RBF Kernel this should handle non-linearity much better and I expect the ICA ratio in this case to behave similarly to the KPCA ratio but should highlight more local changes.

$$ICA_r = \frac{IC1 - IC3}{IC1 - IC2}$$

Then the change ratio is defined as:

$$\delta ICA_r = \frac{ICA_{rt0}}{ICA_{rt1}}$$

This is calculated the same way as the KPCA ratio above. As with the KPCA Ratio

The trends that the ICA ratio highlights are similar to what we have seen so far. But it also highlights other points of interest. Similarly to the KPCA ratio the values are large so there is a need to scale it to keep the values in a reasonable range. So the log of the ICA Ration is used.

```
In [785]: def fica_ratio(net):
    from sklearn.decomposition import FastICA
    fica = FastICA(n_components=3, max_iter=3000, tol=0.001, algorithm='p'
    m = nx.normalized_laplacian_matrix(net).todense()
    X_fica = fica.fit_transform(m)
    fica_ratio = norm(X_fica[:,0]-X_fica[:,2]/(X_fica[:,0]-X_fica[:,1]))

    return fica_ratio

In [786]: def fica_att(net):
    fica_chg = []
    for i in range(len(net)-1):
        x = int(i)
        y = 1+x
        fical= fica_ratio(net[x])
        fica2= fica_ratio(net[y])
        fica_chg.append(np.divide(fica2,fical))
    fica_chgpad = np.zeros(len(fica_chg)+1);
    fica_chgpad[1:] = fica_chg

    return fica_chgpad

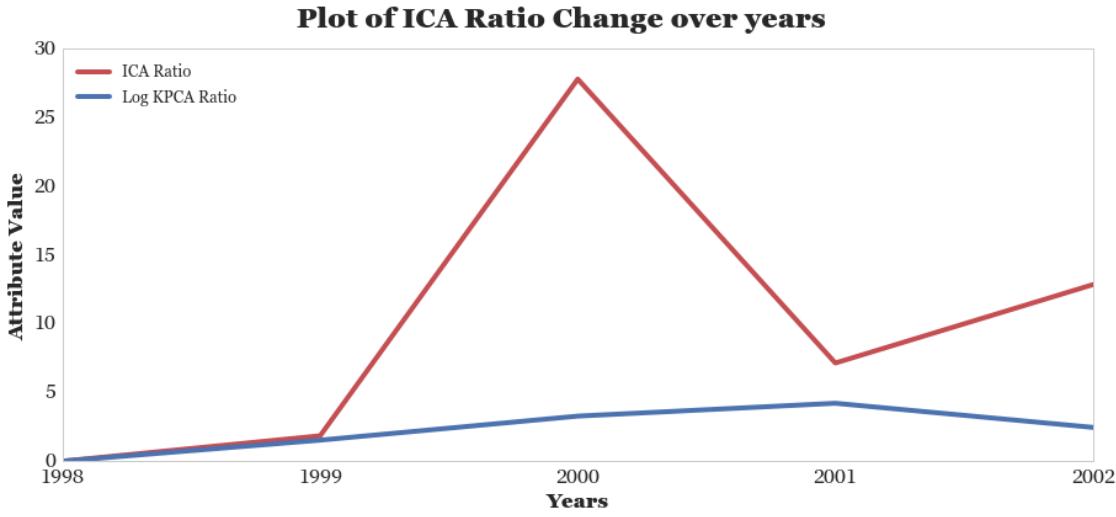
In [787]: fica_ratio_y = fica_att(all_year_G)
          fica_ratio_m = fica_att(all_month_G)

C:\Users\arsha_000\Anaconda3\lib\site-packages\sklearn\decomposition\fastica_.py:11
warnings.warn('FastICA did not converge. Consider increasing '


In [788]: plt.plot(fica_ratio_y,'r',label='ICA Ratio')
          plt.plot(kpca_chg_y, 'b', label='Log KPCA Ratio')

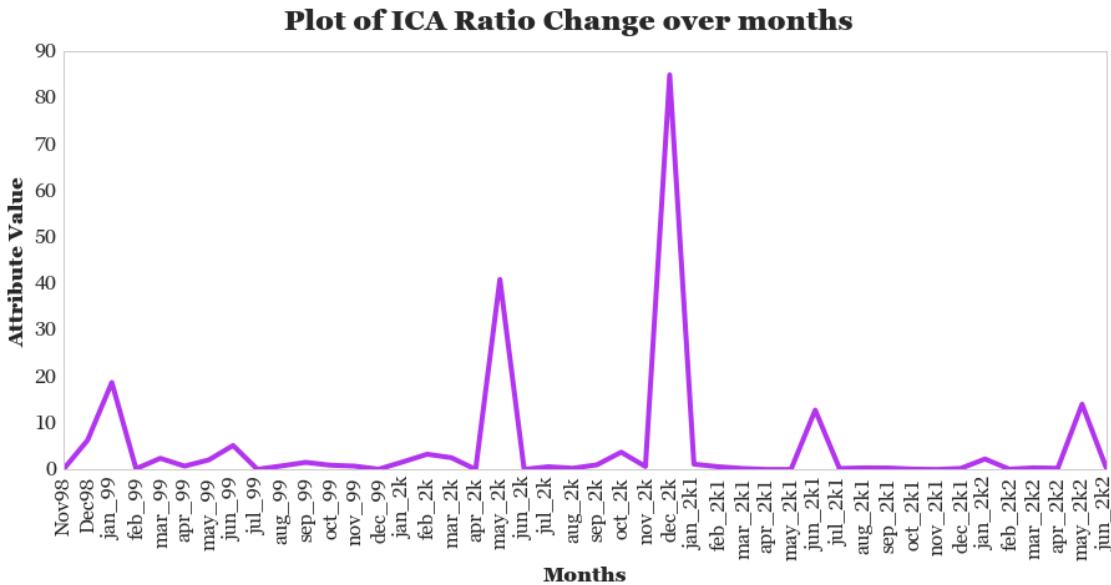
          plt.suptitle("Plot of ICA Ratio Change over years", fontsize=22)
          plt.xlabel("Years", fontsize=16)
          plt.ylabel("Attribute Value", fontsize=16)
          plt.legend(fontsize=13, loc=2)
          plt.xticks(np.arange(len(years[1:])), years[1:], fontsize=16)
          plt.yticks(fontsize=16)

Out[788]: (array([ 0.,  5., 10., 15., 20., 25., 30.]),
 <a list of 7 Text yticklabel objects>)
```

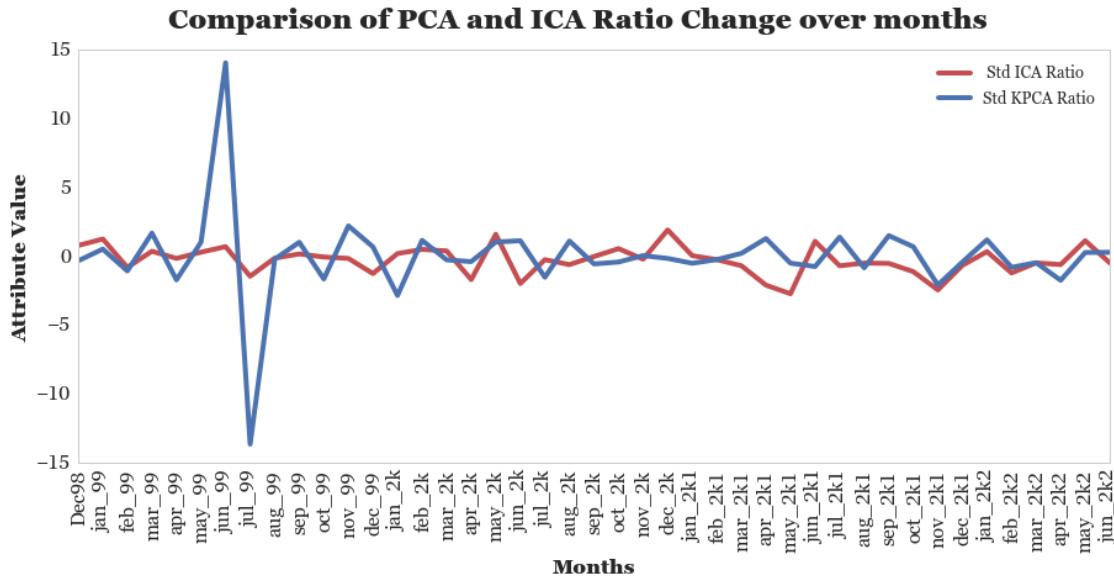


```
In [789]: plt.plot(fica_ratio_m)
plt.suptitle("Plot of ICA Ratio Change over months", fontsize=22)
plt.xlabel("Months", fontsize=16)
plt.ylabel("Attribute Value", fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.xticks(np.arange(len(months)), months, fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.autoscale()
```

C:\Users\arsha_000\Anaconda3\lib\site-packages\matplotlib\axes_axes.py:519: UserWarning
warnings.warn("No labelled objects found. "



```
In [807]: plt.plot(np.log10(fica_ratio_m), 'r', label=' Std ICA Ratio')
plt.plot(np.log10(kpca_chg_m), 'b', label='Std KPCA Ratio')
plt.suptitle("Comparison of PCA and ICA Ratio Change over months", fontsize=16)
plt.xlabel("Months", fontsize=16)
plt.ylabel("Attribute Value", fontsize=16)
plt.legend(fontsize=13, loc=1)
plt.xticks(np.arange(len(months)), months, fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.autoscale()
```



```
In [808]: np.log10(fica_ratio_y)
```

```
Out[808]: array([-inf,  0.2662831 ,  1.4441966 ,  0.85391823,  1.10888409])
```

```
In [809]: np.log10(fica_ratio_m)
```

```
Out[809]: array([-inf,  7.98451032e-01,  1.27270056e+00,
       -7.93122031e-01,  3.80167937e-01, -1.37848567e-01,
       3.13873827e-01,  7.13256561e-01, -1.43569167e+00,
      -1.17858427e-01,  1.84319057e-01, -3.69306897e-02,
      -1.40603795e-01, -1.24049507e+00,  2.12114877e-01,
       5.16442976e-01,  4.03708514e-01, -1.67823244e+00,
       1.61210571e+00, -1.96649834e+00, -2.31105429e-01,
      -5.89048951e-01, -6.14025599e-04,  5.70823867e-01,
     -1.97714809e-01,  1.92946261e+00,  5.85820886e-02,
     -2.28416013e-01, -6.71834810e-01, -2.08249853e+00,
```

```

-2.70919209e+00,    1.10696080e+00,  -6.77068671e-01,
-4.76886507e-01,   -5.02199110e-01,  -1.10056726e+00,
-2.43173415e+00,   -6.41763365e-01,  3.53229549e-01,
-1.19562963e+00,   -4.63277133e-01,  -5.88206213e-01,
1.14788864e+00,   -4.66096464e-01])

```

```
In [792]: y_avgst_all['LogICARatio'] = np.log10(fica_ratio_y)
stat_all['LogICARatio'] = np.nan_to_num(np.log10(fica_ratio_m))
```

```
In [810]: stat_all.LogICARatio.iloc[0]=0
y_avgst_all.LogICARatio.iloc[0]=0
```

7.1 Resistance Distance

The resistance distance between vertices i and j of a graph G is defined as the effective resistance between the two vertices (as when a battery is attached across them) when each graph edge is replaced by a unit resistor. This resistance distance is a metric on graphs.

I calculate the resistance distance as:

M = Normalised Graph Laplacian Matrix

N = Length of M

P = Moore-Penrose Pseudo Inverse of M

D = The diagonal of $Pinv$

$$R_d = (D \otimes (N, 1))^T + (D \otimes (N, 1))^T - P - P^T$$

Here \otimes denotes the Kronecker or outer product

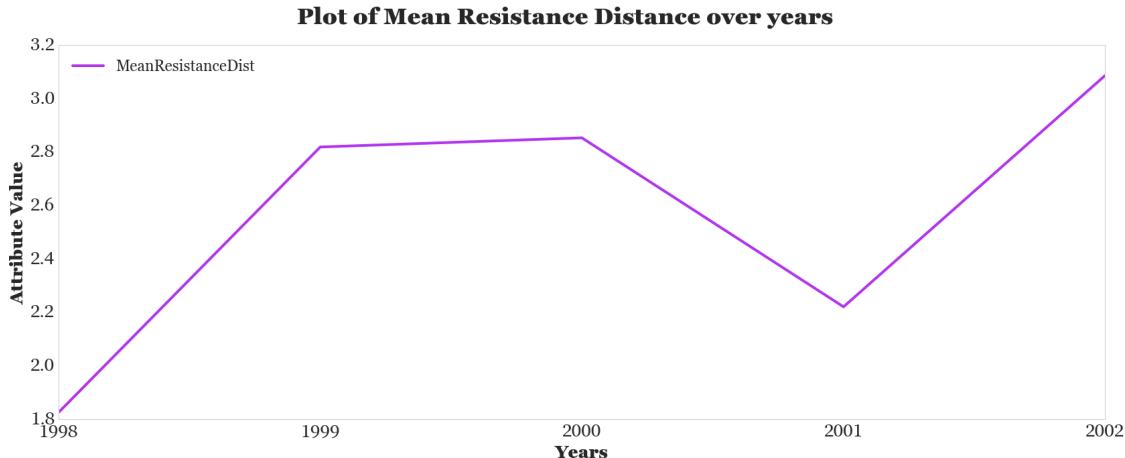
Ref:

1. Weisstein, Eric W. "Resistance Distance." From MathWorld—A Wolfram Web Resource.
<http://mathworld.wolfram.com/ResistanceDistance.html>
2. Function implemented from [here](#)

```
In [811]: y_avgst_all.MeanResistanceDist.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Mean Resistance Distance over years", fontsize=33)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=2)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

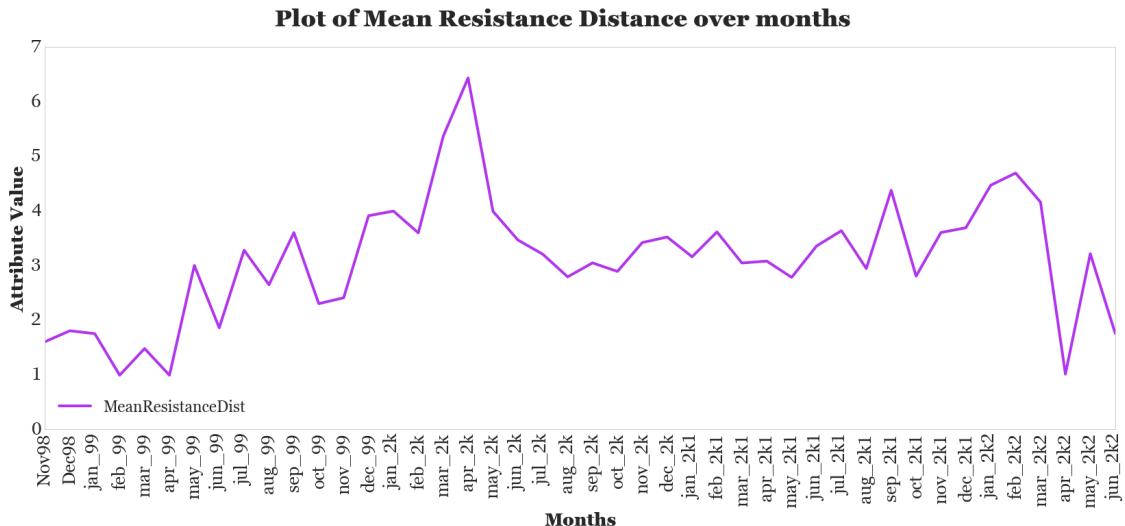
Out[811]: (array([ 1.6,  1.8,  2. ,  2.2,  2.4,  2.6,  2.8,  3. ,  3.2]),
```

<a list of 9 Text yticklabel objects>



```
In [812]: stat_all.MeanResistanceDist.plot(fontsize=22, rot=90, figsize=(28,10))
          plt.suptitle("Plot of Mean Resistance Distance over months", fontsize=33)
          plt.xlabel("Months", fontsize=25)
          plt.ylabel("Attribute Value", fontsize=25)
          plt.legend(fontsize=23, loc=3)
          plt.xticks(np.arange(len(months)), months, fontsize=26)
          plt.yticks(fontsize=26)

Out[812]: (array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.]),
            <a list of 8 Text yticklabel objects>)
```



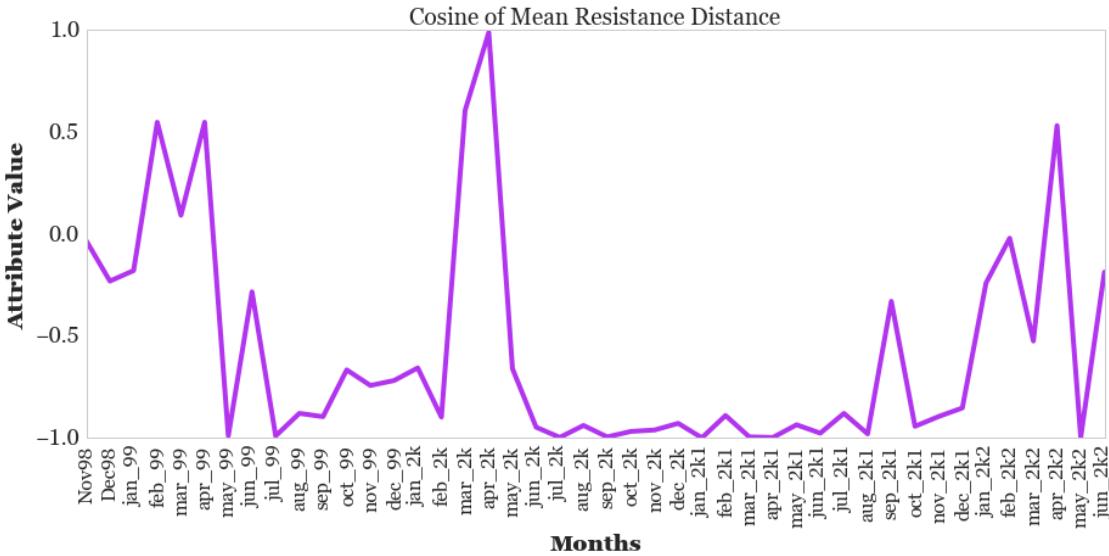
```
In [813]: stat_all.MeanResistanceDist.apply(lambda x: np.cos(x)).plot(title="Cosine transformation of MeanResistanceDist")
          plt.xticks(np.arange(len(months)), months, fontsize=16);
```

```

plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

```

Out [813]: <matplotlib.text.Text at 0x1632f9f5e80>



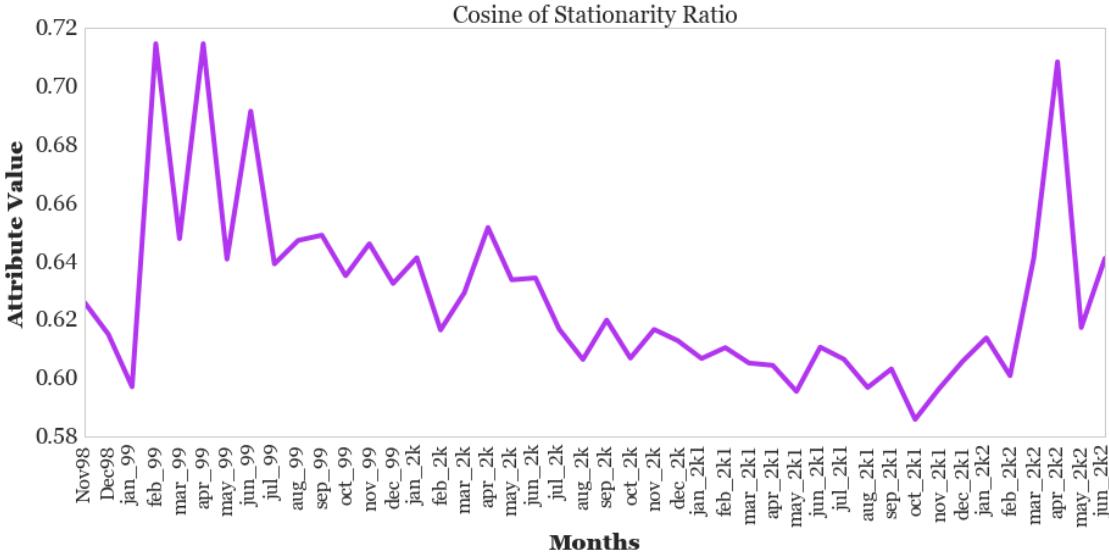
7.2 Cosine of Attributes

```

In [814]: stat_all.StatRat.apply(lambda x: np.cos(x)).plot(title="Cosine of Stationarity Ratio")
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

```

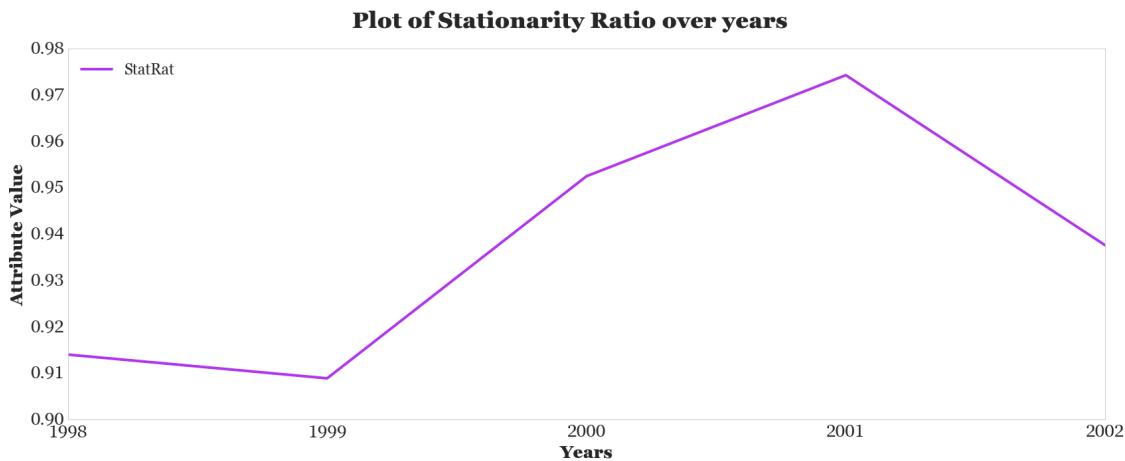
Out [814]: <matplotlib.text.Text at 0x16331ff0cc0>



7.3 Stationarity Ratio

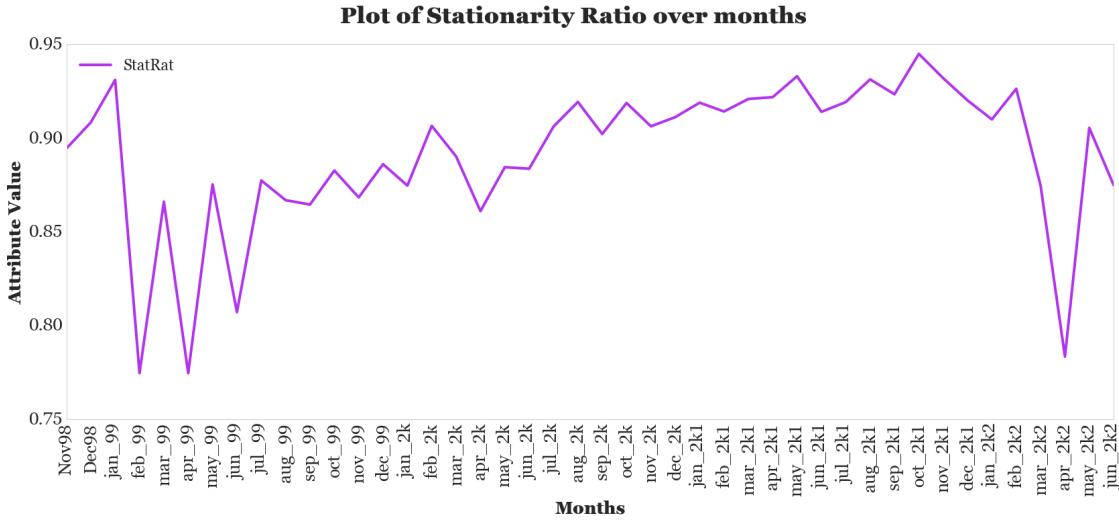
```
In [815]: y_avgst_all.StatRat.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Stationarity Ratio over years", fontsize=33)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=2)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

Out[815]: (array([ 0.9 , 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98]),  
 <a list of 9 Text yticklabel objects>)
```



```
In [816]: stat_all.StatRat.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Stationarity Ratio over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=23, loc=2)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

Out[816]: (array([ 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95]),  
 <a list of 6 Text yticklabel objects>)
```



```
In [817]: stat_all.iloc[:,9:17].plot(fontsize=18, rot =90)
plt.suptitle("Plot of Attributes over months", fontsize=18)
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

stat_all.iloc[:,18: ].plot(fontsize=18, rot =90)
plt.suptitle("Plot of Attributes over months", fontsize=18)
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

stat_all.iloc[:,9:17].apply(lambda x: np.cos(x)*-1).plot(fontsize=18, rot =90)
plt.suptitle("Plot of Cosine of Attributes over months", fontsize=18)
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

stat_all.iloc[:,18: ].apply(lambda x: np.cos(x)*-1).plot(fontsize=18, rot =90)
plt.suptitle("Plot of Cosine of Attributes over months", fontsize=18)
plt.xticks(np.arange(len(months)), months, fontsize=16);
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

stat_all.iloc[:,6].plot(fontsize=18, rot=90)
plt.suptitle("Plot of Avg of Centrality measures over months", fontsize=18)
```

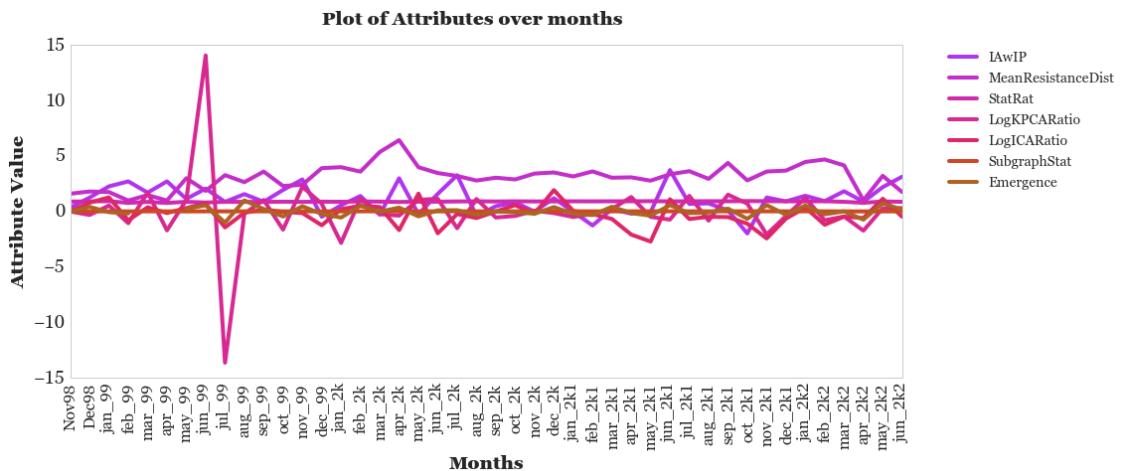
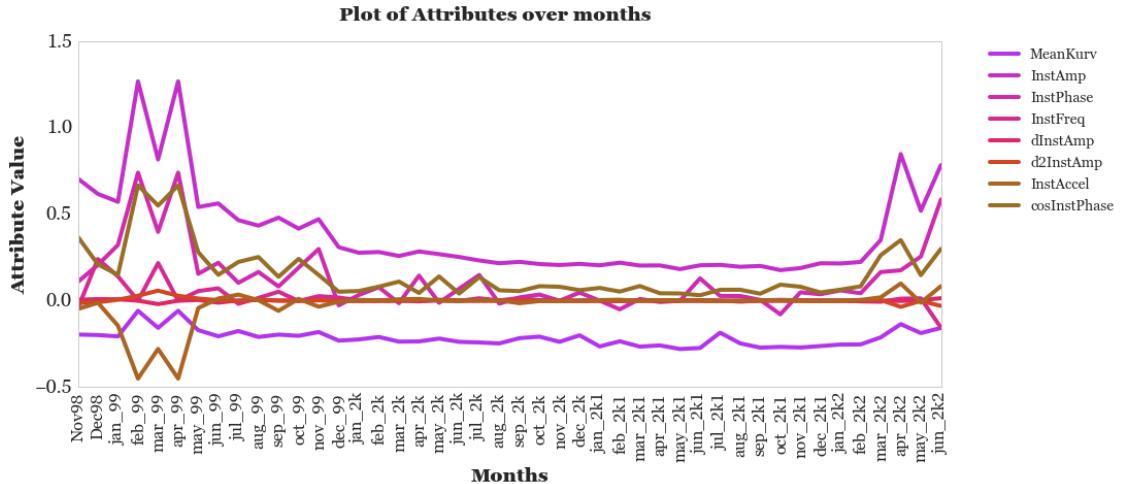
```

plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=14, loc=1)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=16)
plt.yticks(fontsize=16)

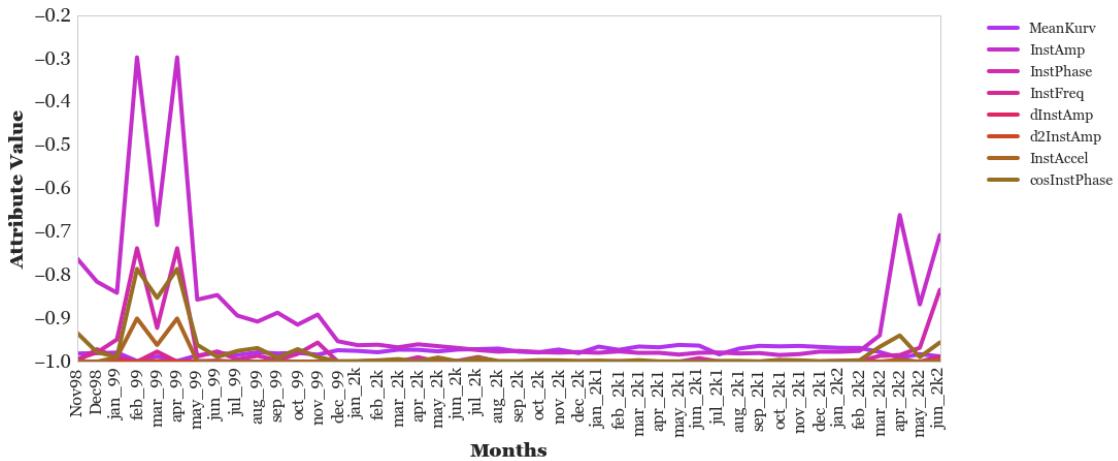
stat_all.iloc[:, :6].apply(lambda x: np.cos(x) * -1).plot(fontsize=18, rot=90)
plt.suptitle("Plot of Cosine of Avg of Centrality measures over months",
plt.xlabel("Months", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=14, loc=1)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=16)
plt.yticks(fontsize=16)

```

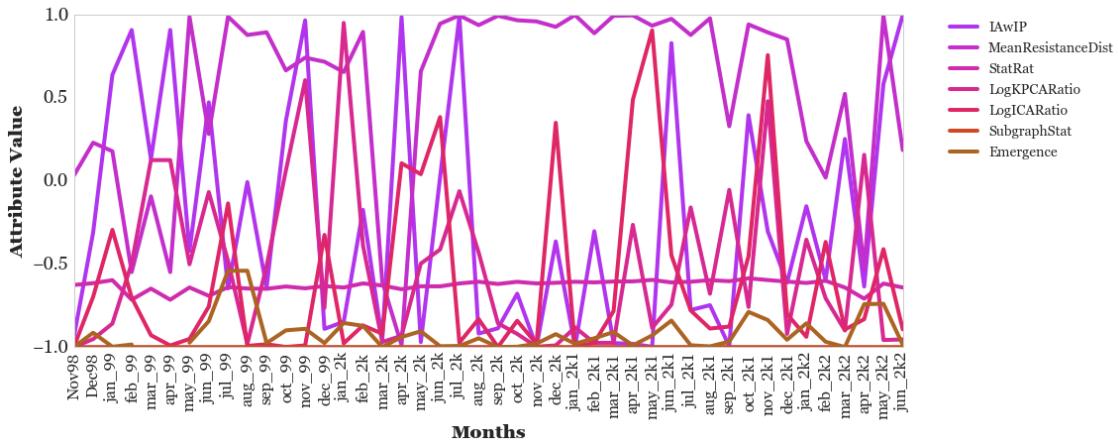
Out[817]: (array([-1. , -0.98, -0.96, -0.94, -0.92, -0.9 , -0.88, -0.86, -0.84, -0.82]),
<a list of 10 Text yticklabel objects>)

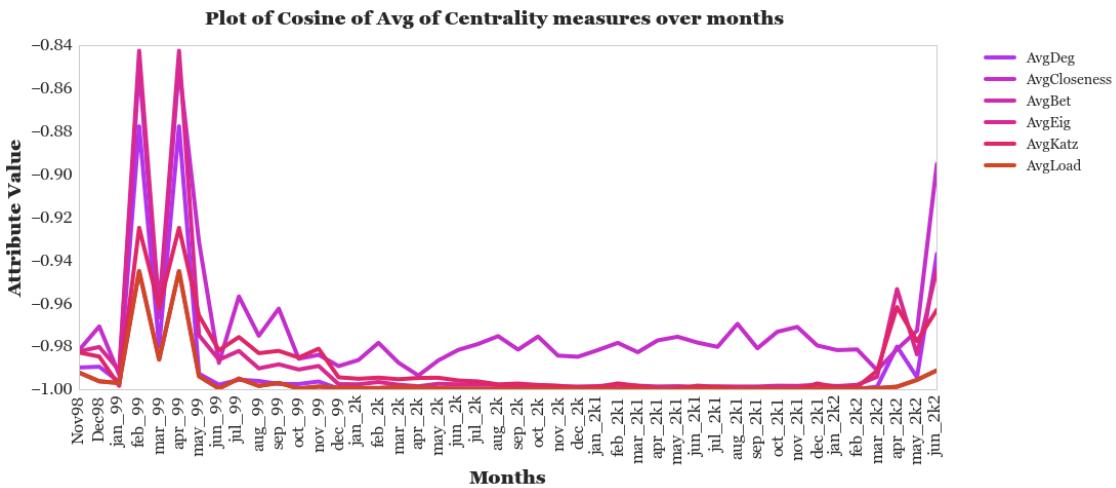
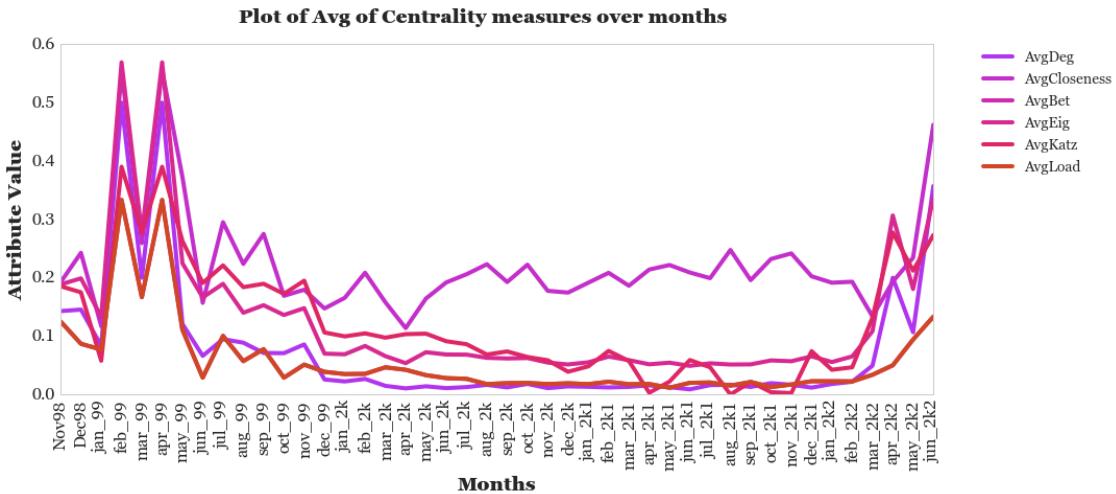


Plot of Cosine of Attributes over months



Plot of Cosine of Attributes over months





```
In [818]: y_avgst_all.iloc[:,9:].plot(fontsize=18)
plt.suptitle("Plot of Attributes over years", fontsize=18)
plt.xticks(years[1:],[i for i in years[1:]],fontsize=26)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)

y_avgst_all.iloc[:,9:].apply(lambda x: np.cos(x)**-1).plot(fontsize=18)
plt.suptitle("Plot of Cosine of Attributes over years", fontsize=18)
plt.xticks(years[1:],[i for i in years[1:]],fontsize=26)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
```

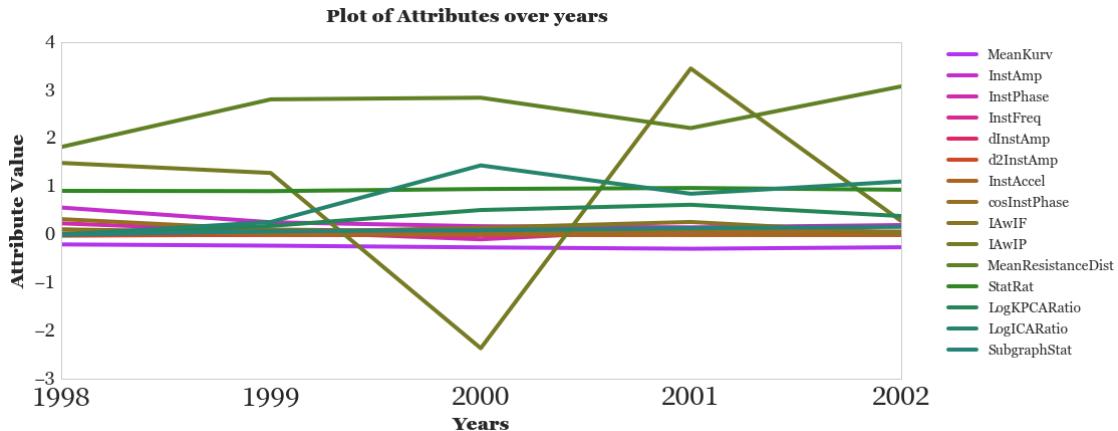
```

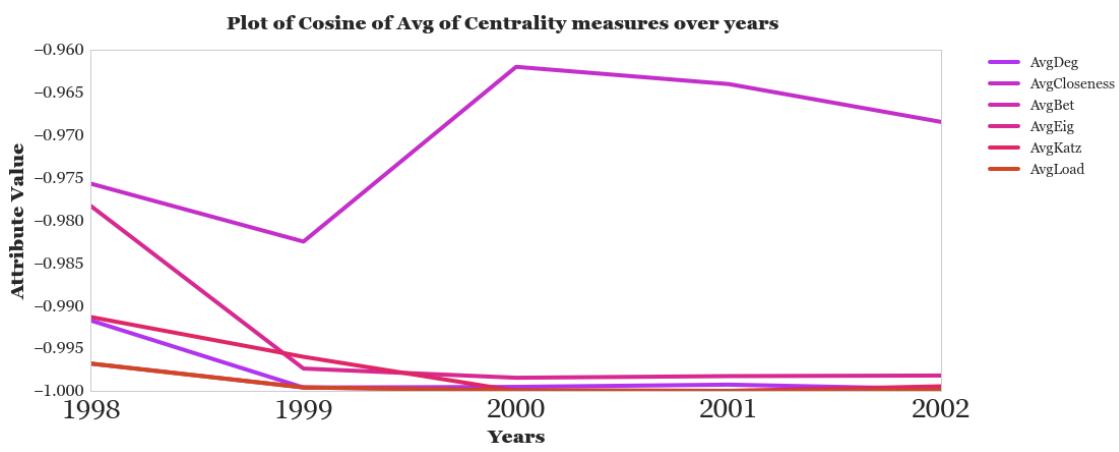
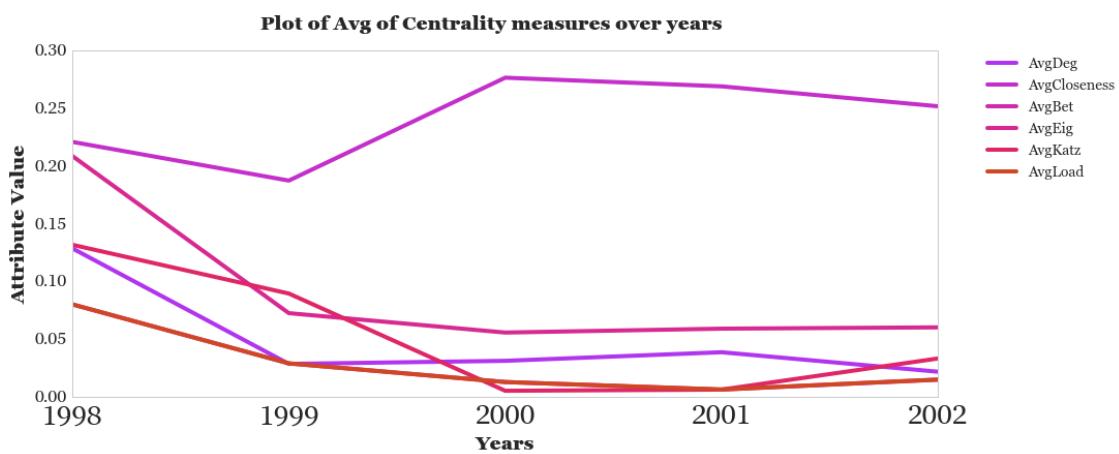
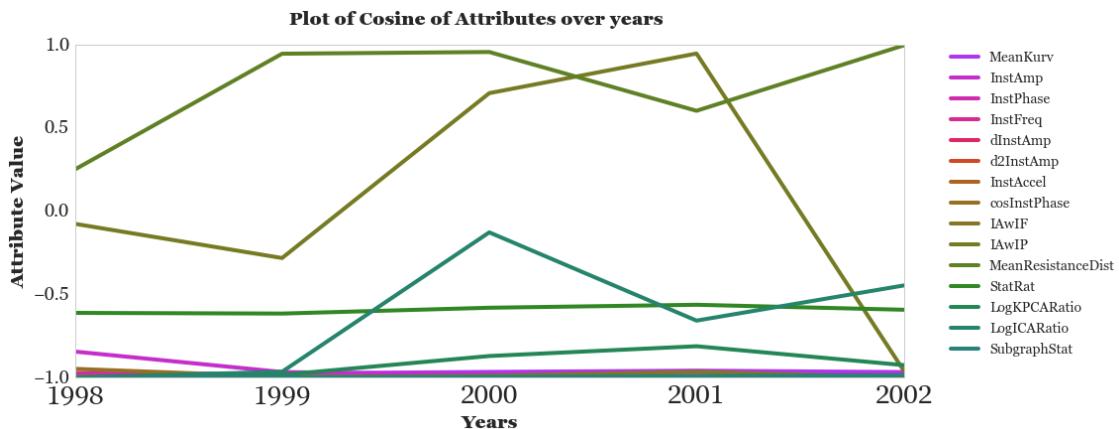
y_avgst_all.iloc[:, :6].plot(fontsize=18)
plt.suptitle("Plot of Avg of Centrality measures over years", fontsize=18)
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=14, loc=1)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=16)

y_avgst_all.iloc[:, :6].apply(lambda x: np.cos(x) * -1).plot(fontsize=18)
plt.suptitle("Plot of Cosine of Avg of Centrality measures over years", f
plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=14, loc=1)
plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=16)

```

Out[818]: (array([-1. , -0.995, -0.99 , -0.985, -0.98 , -0.975, -0.97 , -0.965, -0.96]), <a list of 9 Text yticklabel objects>)

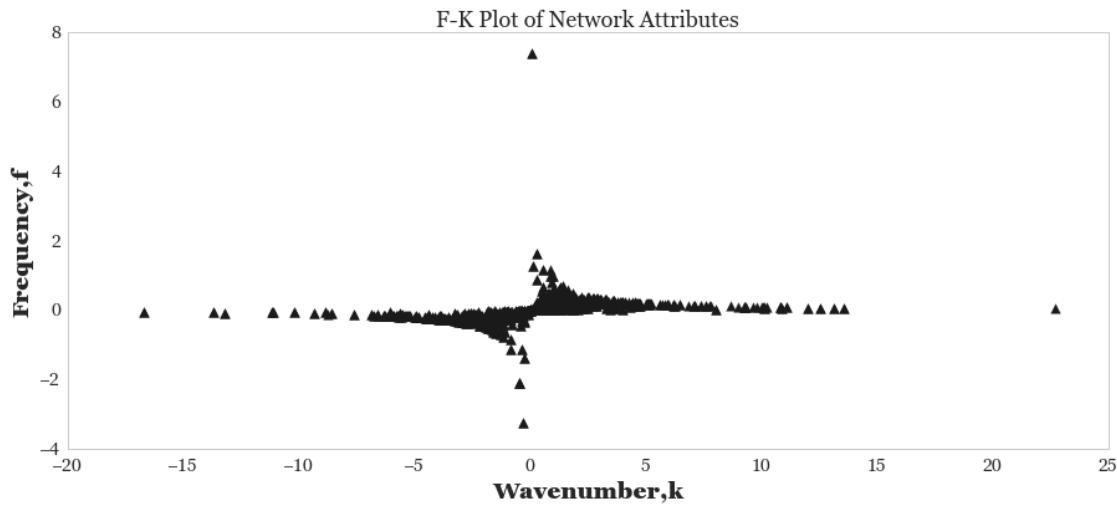




8 Frequency Wavenumber (fk) and Radon Plots

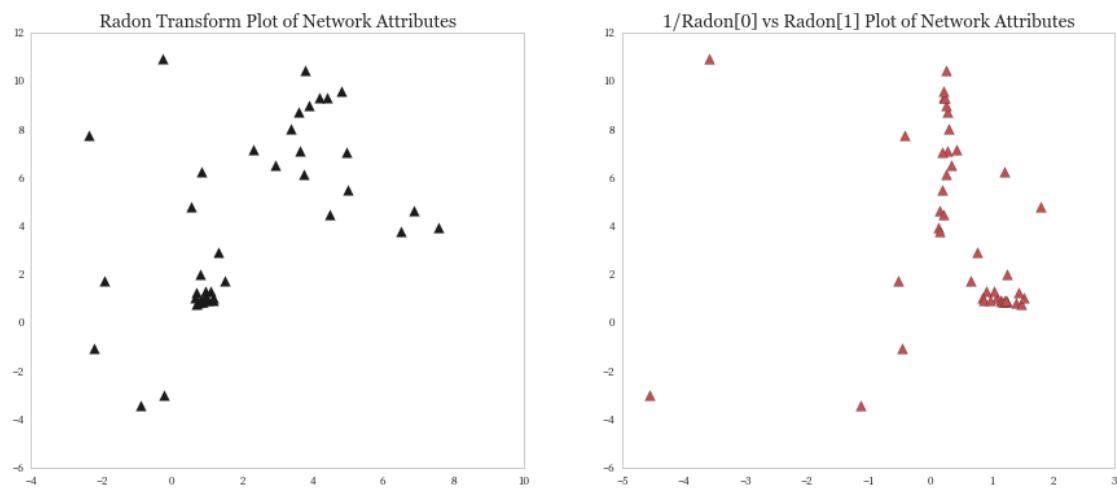
```
In [819]: freq,k = fk_plot(stat_all.values, 'Network Attributes')
```

```
C:\Users\arsha_000\Anaconda3\lib\site-packages\numpy\core\numeric.py:533: ComplexWav
```



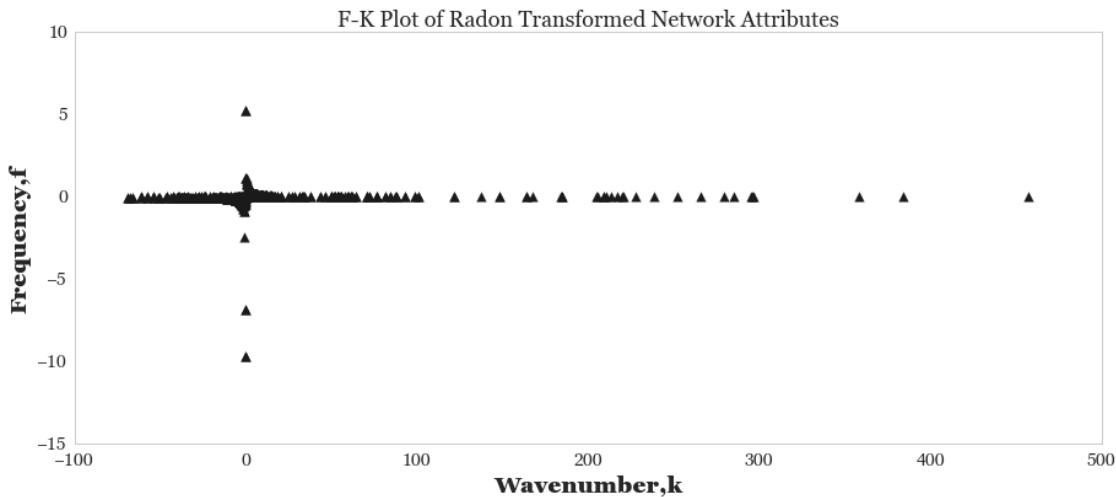
```
In [820]: radon_att = plot_radon(stat_all.values, 'Network Attributes')
```

```
C:\Users\arsha_000\Anaconda3\lib\site-packages\skimage\transform\radon_transform.py
```



```
In [821]: freq,k = fk_plot(radon_att,'Radon Transformed Network Attributes')

C:\Users\arsha_000\Anaconda3\lib\site-packages\numpy\core\numeric.py:533: ComplexWarning
  return array(a, dtype, copy=False, order=order, subok=True)
```



9 Towards Global Measures

9.1 Subgraph Stationarity, ζ

This is computed from the adjacency matrix of graphs and is done in two steps. Essentially this is comparing the common subgraphs between two networks and deriving a correlation score. This is done in two steps.

Step 1: Calculate Correlation, C_t between the graphs at time, t and time, $t+1$.

$$C(t) = \frac{A(t) \cap A(t+1)}{A(t) \cup A(t+1)}$$

Step 2: Calculate the Subgraph Stationarity, ζ

$$\zeta = \frac{\sum_{t=0}^{t_{max}-1} C(t, t+1)}{t_{max} - t_0 - 1}$$

From the Subgraph Stationarity ζ we can calculate the amount of members that change at each time step as

$$1 - \zeta$$

```
In [822]: Ct_y = []
for i in range(0, len(all_year_G)-1):
    x = int(i)
    y = x + 1
    Ct_y.append(subgraph_stat(all_year_G[x], all_year_G[y]))
```

```

In [823]: Ct_m = []
    for i in range(0, len(all_month_G)-1):
        x = int(i)
        y = x +1
        Ct_m.append(subgraph_stat(all_month_G[x], all_month_G[y]))

In [824]: Ct_y

Out[824]: [0.30310524825452068,
            0.37838536898931635,
            0.55196098792073822,
            0.66442754991336916]

In [825]: tmp3 = np.zeros(len(Ct_y)+1)
    tmp3[1:] = Ct_y
    tmp3

Out[825]: array([ 0.           ,  0.30310525,  0.37838537,  0.55196099,  0.66442755])

In [826]: Ct_m[:5]

Out[826]: [0.56195148694901642,
            0.67700320038633011,
            0.3779644730092272,
            0.53452248382484879,
            0.53452248382484879]

In [827]: tmp4 = np.zeros(len(Ct_m)+1)
    tmp4[1:] = Ct_m
    tmp4

Out[827]: array([ 0.           ,  0.56195149,  0.6770032 ,  0.37796447,  0.53452248,
            0.53452248,  0.30151134,  0.62418778,  0.56493268,  0.64655455,
            0.60572805,  0.66449864,  0.69156407,  0.48989795,  0.62609903,
            0.63748937,  0.66022529,  0.68268832,  0.61033232,  0.62400363,
            0.62754405,  0.61876404,  0.69455555,  0.66183399,  0.67545887,
            0.66466807,  0.6858916 ,  0.70393524,  0.67087945,  0.67054686,
            0.67039676,  0.70622784,  0.62254302,  0.59803739,  0.68533074,
            0.62489999,  0.7118685 ,  0.748884 ,  0.62864935,  0.67852807,
            0.40610041,  0.2763854 ,  0.37047929,  0.57259833])

In [828]: np.cumsum(tmp3) / tmp3.shape[0]

Out[828]: array([ 0.           ,  0.06062105,  0.13629812,  0.24669032,  0.37957583])

In [829]: zeta_y= pd.DataFrame(tmp3)
    zeta_y = zeta_y.cumsum(axis=1)/(zeta_y.shape[0]-1)
    zeta_y = zeta_y.T
    zeta_y.columns=years[1:]
    zeta_y = pd.DataFrame(zeta_y).T
    zeta_y.columns=['ZetaYear']
    zeta_y.head()

```

```
Out[829]:      ZetaYear
1998    0.000000
1999    0.075776
2000    0.094596
2001    0.137990
2002    0.166107
```

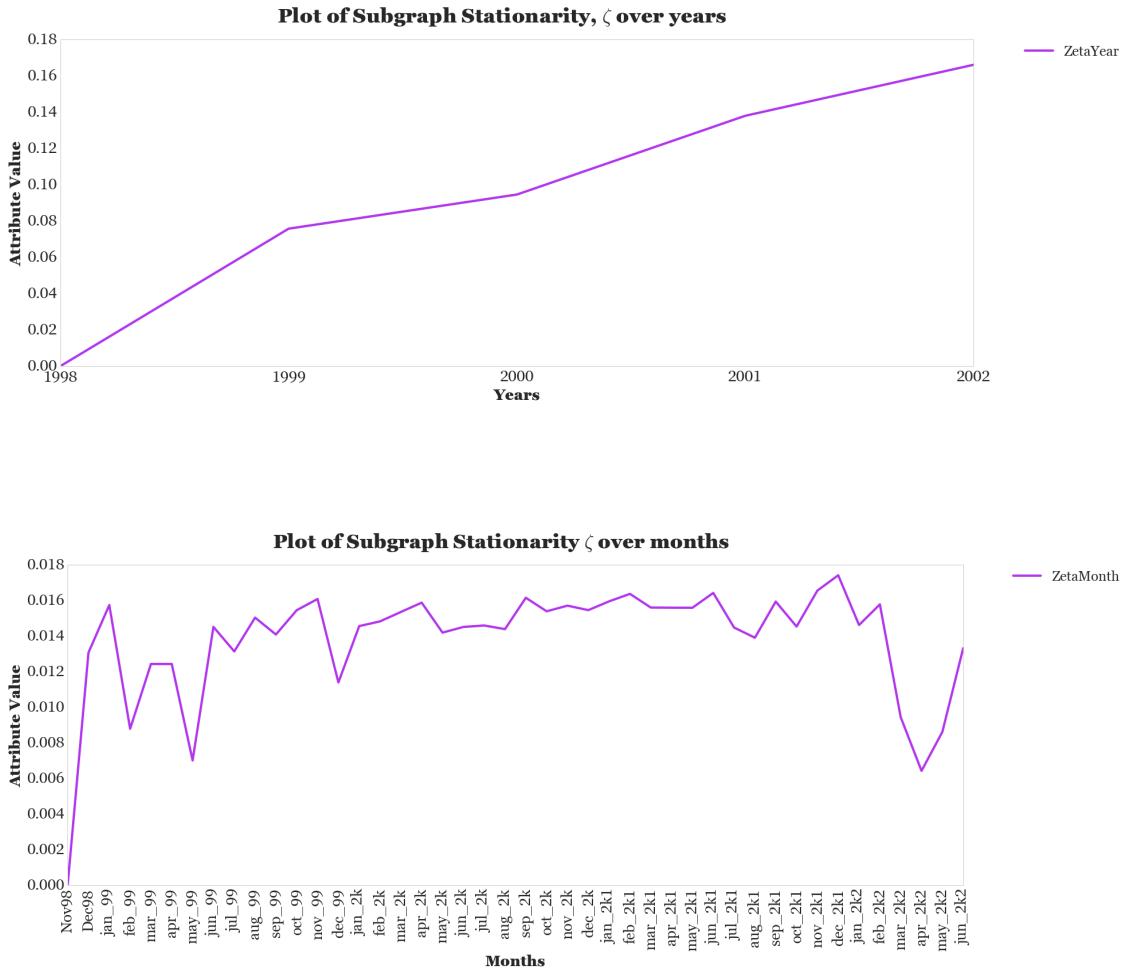
```
In [830]: zeta_m = pd.DataFrame(tmp4)
zeta_m = zeta_m.cumsum(axis=1)/(zeta_m.shape[0]-1)
zeta_m = zeta_m.T
zeta_m.columns=months
zeta_m = pd.DataFrame(zeta_m).T
zeta_m.columns=['ZetaMonth']
zeta_m.head()
```

```
Out[830]:      ZetaMonth
Nov98    0.000000
Dec98    0.013069
jan_99   0.015744
feb_99   0.008790
mar_99   0.012431
```

```
In [831]: zeta_y.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Subgraph Stationarity, $\zeta$ over years", fontsize=24)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

zeta_m.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Subgraph Stationarity $\zeta$ over months", fontsize=24)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

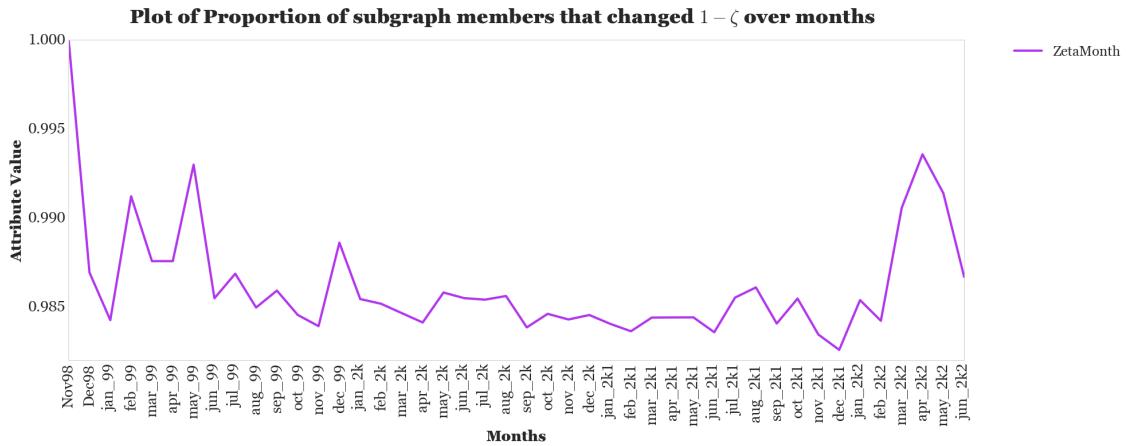
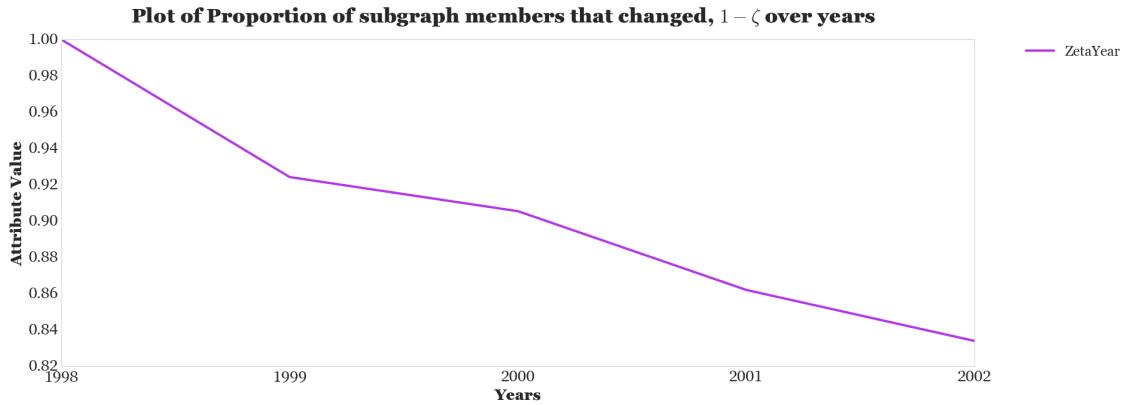
```
Out[831]: (array([ 0.      ,  0.002,  0.004,  0.006,  0.008,  0.01  ,  0.012,  0.014,
   0.016,  0.018]), <a list of 10 Text yticklabel objects>)
```



```
In [832]: (1-zeta_y).plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Proportion of subgraph members that changed, $1-\zeta$ over years")
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

(1-zeta_m).plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Proportion of subgraph members that changed $1-\zeta$ over months")
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)
```

```
Out[832]: (array([ 0.98 , 0.985, 0.99 , 0.995, 1. , , 1.005]),  
<a list of 6 Text yticklabel objects>)
```



```
In [833]: y_avgst_all['SubgraphStat']=zeta_y
stat_all['SubgraphStat']=zeta_m
```

9.2 Persistence & Emergence

Persistence and Emergence are discussed in the paper below.

At the very beginning timestep where $t = 1$ or at any timestep when both $P_{i,t}$ and $P_{i,t-1}$ are zero, Emergence is equal to 0, which indicates a neutral change. When t is larger than 1 and $\max\{P_{i,t}, P_{i,t-1}\} \neq 0$, it is defined to be the normalized change of Persistence at the most recent timesteps. Depending on whether the Persistence contains negative values, normalization terms are defined differently.

Ref:

- [1] W. Wei and K. M. Carley, "Measuring Temporal Patterns in Dynamic Social Networks," ACM Trans. Knowl. Discov. Data, vol. 10, no. 1, pp. 1–27, 2015.

```
In [834]: stat_all.head()
```

```

Out[834]:
```

| | AvgDeg | AvgCloseness | AvgBet | AvgEig | AvgKatz | AvgLoad |
|--------|----------|--------------|----------|----------|----------|----------|
| Nov98 | 0.142857 | 0.192308 | 0.125000 | 0.188611 | 0.185435 | 0.125000 |
| Dec98 | 0.145455 | 0.242737 | 0.086869 | 0.198999 | 0.175111 | 0.086869 |
| jan_99 | 0.083333 | 0.117057 | 0.076923 | 0.132902 | 0.057692 | 0.076923 |
| feb_99 | 0.500000 | 0.555556 | 0.333333 | 0.569036 | 0.390223 | 0.333333 |
| mar_99 | 0.200000 | 0.259259 | 0.166667 | 0.275965 | 0.277684 | 0.166667 |

| | Density | AvgClustCoef | AlgebraicConnect | MeanKurv | ... |
|--------|----------|--------------|------------------|---------------|-----|
| Nov98 | 0.250000 | 0.000000 | | 1.0 -0.196247 | ... |
| Dec98 | 0.218182 | 0.246465 | | 1.0 -0.199212 | ... |
| jan_99 | 0.153846 | 0.000000 | | 1.0 -0.206795 | ... |
| feb_99 | 0.666667 | 0.000000 | | 1.0 -0.058750 | ... |
| mar_99 | 0.333333 | 0.000000 | | 1.0 -0.157405 | ... |

| | InstAccel | cosInstPhase | IAwIF | IAwIP | MeanResistanceDi |
|--------|-----------|--------------|---------------|----------|------------------|
| Nov98 | -0.048538 | 0.369891 | -2.164452e-01 | 0.345301 | 1.6038 |
| Dec98 | -0.012273 | 0.210647 | 1.753912e+00 | 1.255140 | 1.8040 |
| jan_99 | -0.144891 | 0.149447 | 9.562196e-01 | 2.265634 | 1.7512 |
| feb_99 | -0.451282 | 0.667721 | -1.233581e-17 | 2.714095 | 0.9904 |
| mar_99 | -0.279494 | 0.550674 | 1.049833e+00 | 1.703481 | 1.4787 |

| | StatRat | LogKPCARatio | LogICARatio | SubgraphStat | Emergence |
|--------|----------|--------------|-------------|--------------|-----------|
| Nov98 | 0.894427 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| Dec98 | 0.908295 | -0.310794 | 0.798451 | 0.013069 | 0.417931 |
| jan_99 | 0.930949 | 0.538398 | 1.272701 | 0.015744 | -0.062591 |
| feb_99 | 0.774597 | -1.051911 | -0.793122 | 0.008790 | -0.174946 |
| mar_99 | 0.866025 | 1.697420 | 0.380168 | 0.012431 | NaN |

[5 rows x 25 columns]

```

In [835]: persistence_y = y_avgst_all.mean(axis=1)/y_avgst_all.shape[0]
persistence_m = stat_all.mean(axis=1)/stat_all.shape[0]

```

```

In [836]: persistence_y.plot(fontsize=22, figsize=(28,10))
plt.suptitle("Plot of Persistence over years", fontsize=33)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
# plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=26)
plt.yticks(fontsize=26)

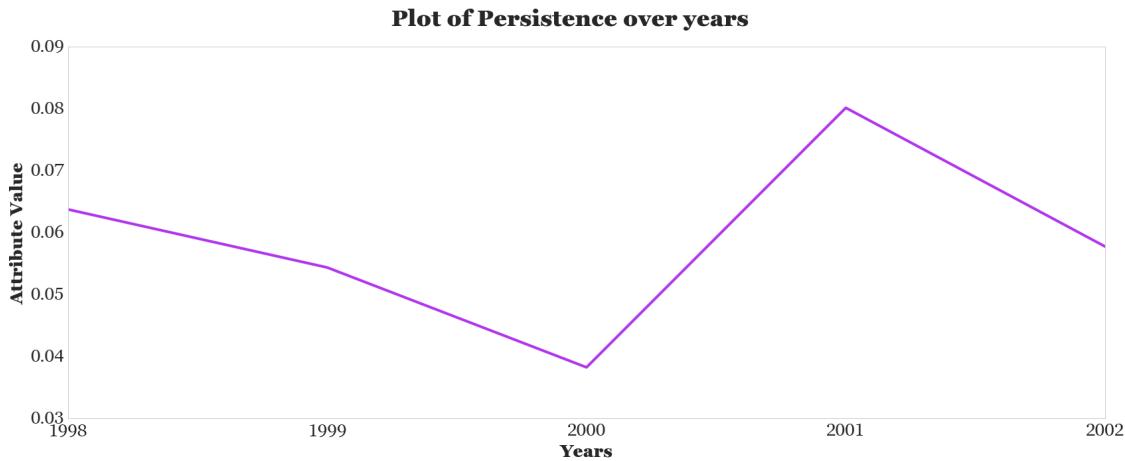
```

```

Out[836]: (array([ 0.02,  0.03,  0.04,  0.05,  0.06,  0.07,  0.08,  0.09]),  

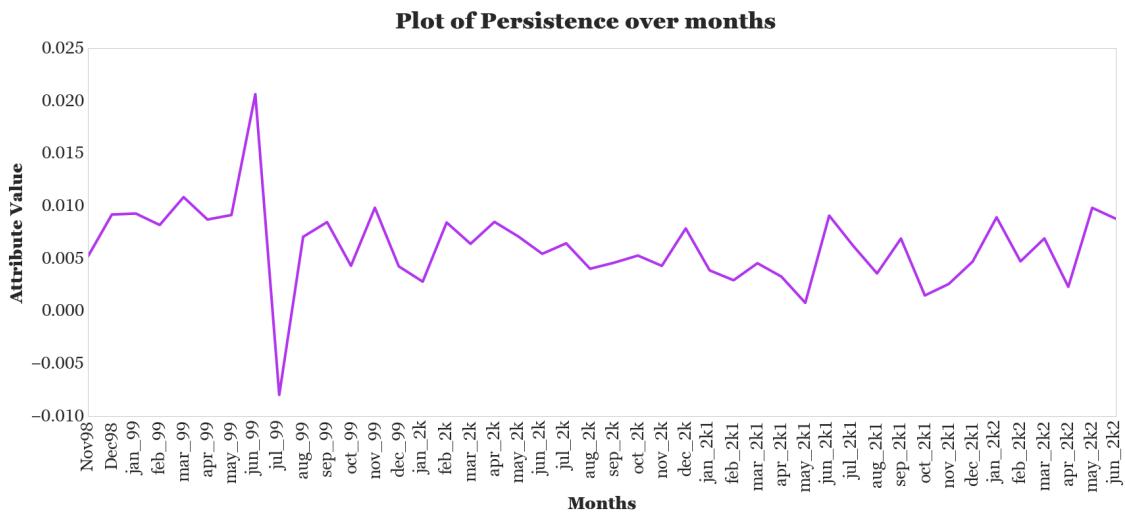
 <a list of 8 Text yticklabel objects>

```



```
In [837]: persistence_m.plot(fontsize=22, rot=90, figsize=(28,10))
plt.suptitle("Plot of Persistence over months", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Attribute Value", fontsize=25)
# plt.legend(fontsize=24, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

Out[837]: (array([-0.01 , -0.005,  0. ,  0.005,  0.01 ,  0.015,  0.02 ,  0.025,
       0.03 ]), <a list of 9 Text yticklabel objects>)
```



```
In [838]: def emergence(per):
    tmp = np.asarray(per)
    emerg = []
```

```

for i in range(len(tmp)-1):
    x= int(i)
    y = x +1
    #print(tmp[y], tmp[x])
    if tmp[y]==tmp[x]:
        emerg.append(0)
    elif tmp[y] < (0) or tmp[x] < 0:
        res = (tmp[y]-tmp[x]) / (abs(tmp[y])+abs(tmp[x]))
        emerg.append(res)
    else:
        res = (tmp[y]-tmp[x]) / max([tmp[y],tmp[x]])
        emerg.append(res)
tmp2 = np.zeros(len(emerg)+1)
tmp2[1:] = emerg

return tmp2

```

In [839]: emerg_y = emergence(persistence_y)
emerg_m = emergence(persistence_m)

In [840]: emerg_y_df = pd.DataFrame(emerg_y).T
emerg_y_df.columns=years[1:]
emerg_y_df = emerg_y_df.T
emerg_y_df.columns = ['EmergenceYear']
emerg_y_df.head()

Out[840]: EmergenceYear
1998 0.000000
1999 -0.147138
2000 -0.296412
2001 0.522657
2002 -0.279397

In [841]: emerg_m_df = pd.DataFrame(emerg_m).T
emerg_m_df.columns=months
emerg_m_df = emerg_m_df.T
emerg_m_df.columns = ['EmergenceMonth']
emerg_m_df.head()

Out[841]: EmergenceMonth
Nov98 0.000000
Dec98 0.433631
jan_99 0.010569
feb_99 -0.117516
mar_99 0.244502

In [842]: emerg_y_df.plot(fontsize=22, rot=90)
plt.suptitle("Plot of Emergence over years", fontsize=22)

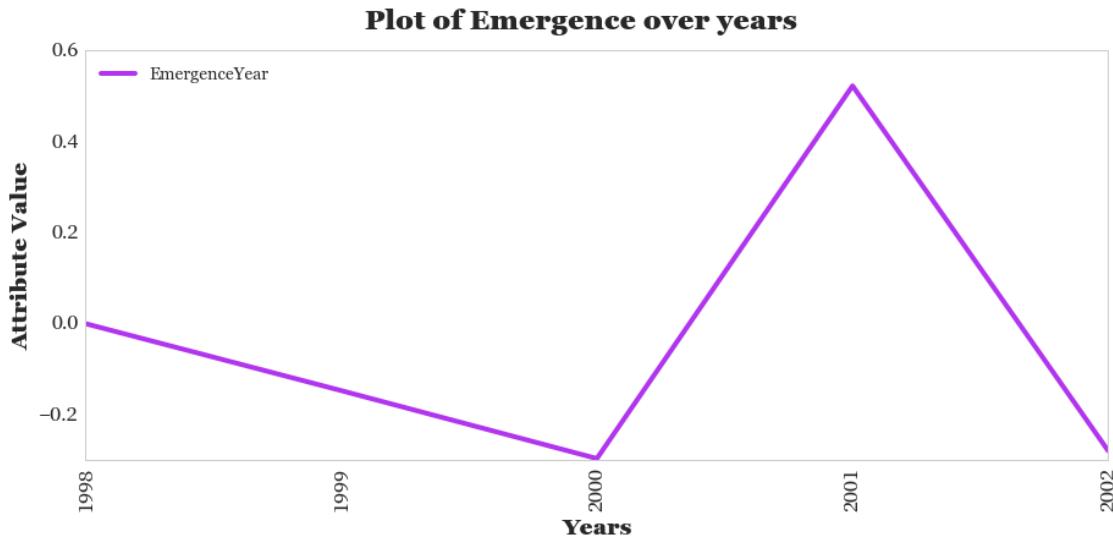
```

plt.xlabel("Years", fontsize=18)
plt.ylabel("Attribute Value", fontsize=18)
plt.legend(fontsize=14, loc=2)
plt.xticks(years[1:], [i for i in years[1:]], fontsize=16)
plt.yticks(fontsize=16)

Out[842]: (array([-0.4, -0.2,  0.,  0.2,  0.4,  0.6,  0.8]),  

           <a list of 7 Text yticklabel objects>)

```

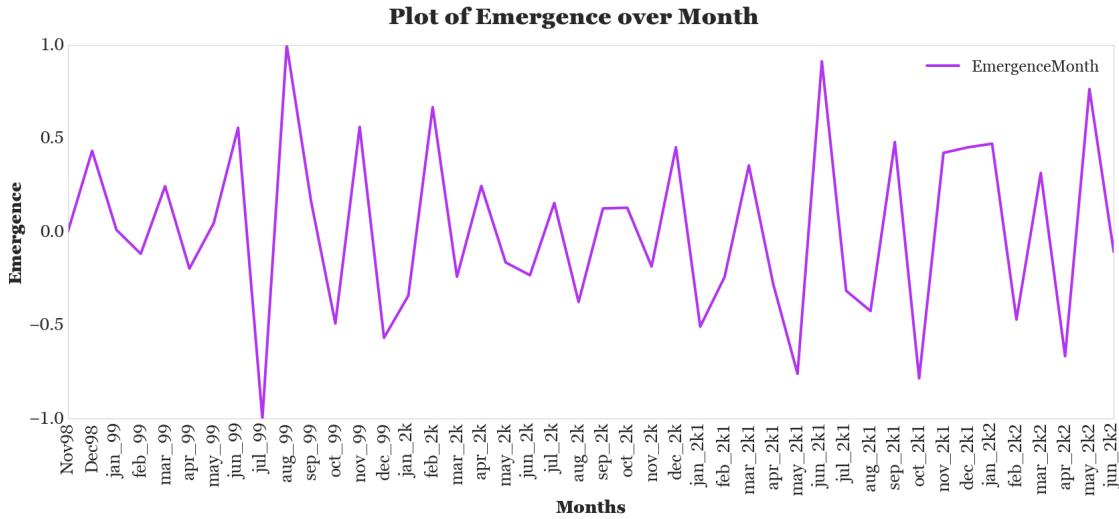


```

In [843]: emerg_m_df.plot(fontsize=22, figsize=(28,10), rot=90)
plt.suptitle("Plot of Emergence over Month", fontsize=33)
plt.xlabel("Months", fontsize=25)
plt.ylabel("Emergence", fontsize=25)
plt.legend(fontsize=24, loc=1)
plt.xticks(np.arange(len(months)), months, fontsize=26)
plt.yticks(fontsize=26)

```

```
Out[843]: (array([-1., -0.5,  0.,  0.5,  1.]), <a list of 5 Text yticklabel obje
```



9.3 NRMS of Emergence

The NRMS is a repeatability metric used in 4D seismic to quantify the likeness of two traces.

It is defined as:

$$NRMS = \frac{200RMS(a - b)}{RMS(a) + RMS(b)}$$

where the RMS operator is

$$RMS = \sqrt{\frac{\sum(x)^2}{N}}$$

```
In [844]: nrms_emerg = []
for i in range(len(emerg_m)-1):
    x = int(i)
    y = x + 1
    nrms_emerg.append(nrms(emerg_m[x], emerg_m[y]))
```

```
In [845]: nrms_emerg[:2]
```

```
Out[845]: [200.0, 190.48300800472288]
```

9.4 NRMS of Network Attributes

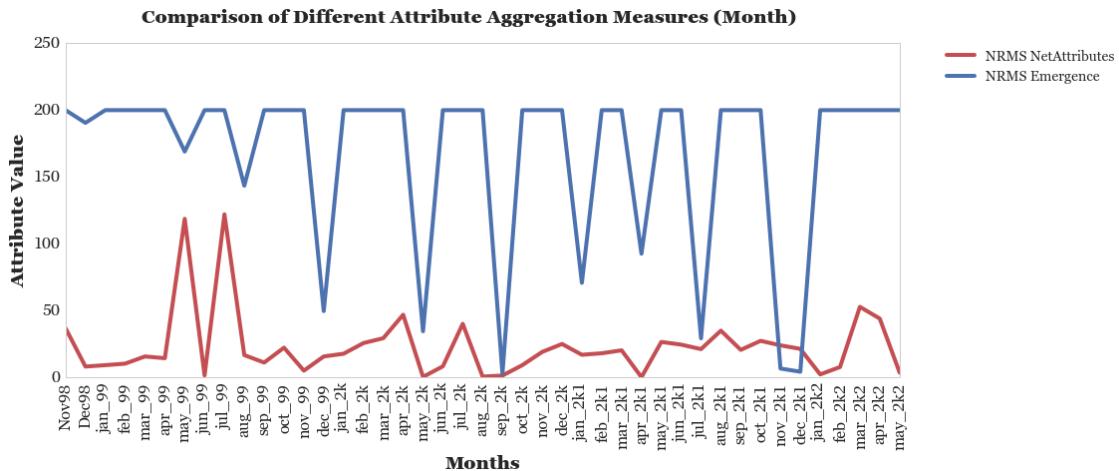
```
In [846]: rms_netatt = stat_all.apply(rms, axis=1).values
```

```
In [847]: nrms_netatt = []
for i in range(rms_netatt.shape[0]-1):
    x = int(i)
    y = x + 1
    nrms_netatt.append(nrms(rms_netatt[x], rms_netatt[y]))
```

```
In [848]: nrms_netatt[:5]
```

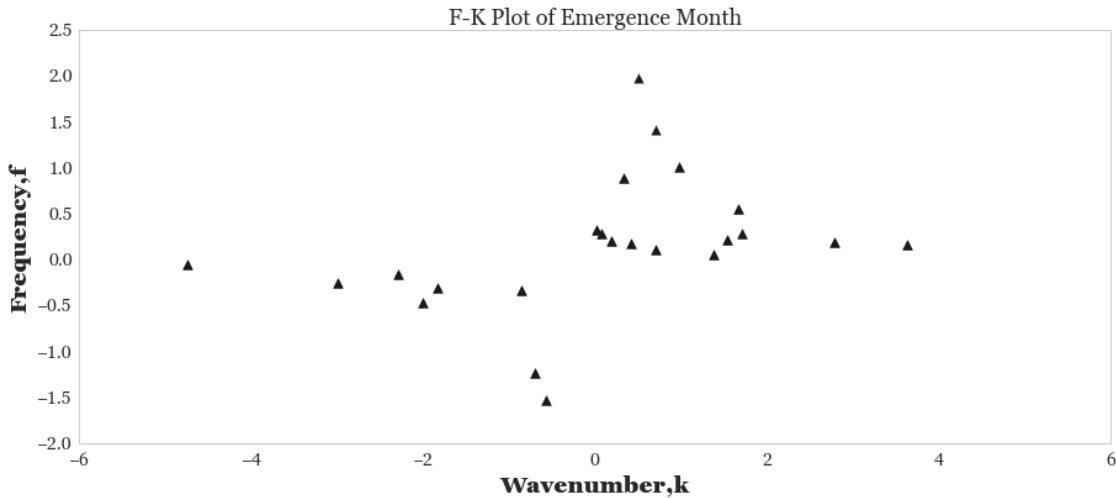
```
Out[848]: [36.822294687642831,  
          8.1275610050666458,  
          9.1723709176001194,  
          10.30256724996277,  
          15.697349414258385]
```

```
In [849]: plt.suptitle("Comparison of Different Attribute Aggregation Measures (Month)"  
                     #plt.plot(emerg_m, 'k', label='EmergenceMonth')  
                     plt.plot(nrms_netatt, 'r', label='NRMS NetAttributes')  
                     plt.plot(nrms_emerg, 'b', label='NRMS Emergence')  
                     plt.xlabel("Months", fontsize=18)  
                     plt.ylabel("Attribute Value", fontsize=18)  
                     plt.legend(fontsize=14, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)  
                     plt.xticks(np.arange(len(months)), months, fontsize=16, rotation=90)  
                     plt.yticks(fontsize=16)  
                     plt.autoscale()
```



```
In [850]: f,k = fk_plot(emerg_m, 'Emergence Month')
```

```
C:\Users\arsha_000\Anaconda3\lib\site-packages\numpy\core\numeric.py:533: ComplexWarning:  
return array(a, dtype, copy=False, order=order, subok=True)
```



```
In [851]: print("Min/Max Freq Index:", np.argmax(f), np.argmin(f))
print("Min/Max wavenumber Index:", np.argmax(k), np.argmin(k))
```

```
Min/Max Freq Index: 14 16
Min/Max wavenumber Index: 0 9
```

```
In [852]: print(emerg_m_df.iloc[np.argmax(f)])
print(emerg_m_df.iloc[np.argmin(f)])
print(emerg_m_df.iloc[np.argmax(k)])
print(emerg_m_df.iloc[np.argmin(k)])
```

```
EmergenceMonth    -0.341605
Name: jan_2k, dtype: float64
EmergenceMonth    -0.240677
Name: mar_2k, dtype: float64
EmergenceMonth     0.0
Name: Nov98, dtype: float64
EmergenceMonth     1.0
Name: aug_99, dtype: float64
```

9.5 Sort by freq and wavenumber indices

```
In [853]: emerg_m_df.iloc[np.argsort(f)].head(n=10)
```

```
Out[853]:          EmergenceMonth
mar_2k           -0.240677
mar_2k1          0.355305
feb_2k           0.667449
apr_2k1          -0.278886
```

```

feb_2k1      -0.242035
apr_2k       0.245745
nov_99        0.561742
jul_2k1      -0.315396
sep_99        0.164630
sep_2k1      0.480412

```

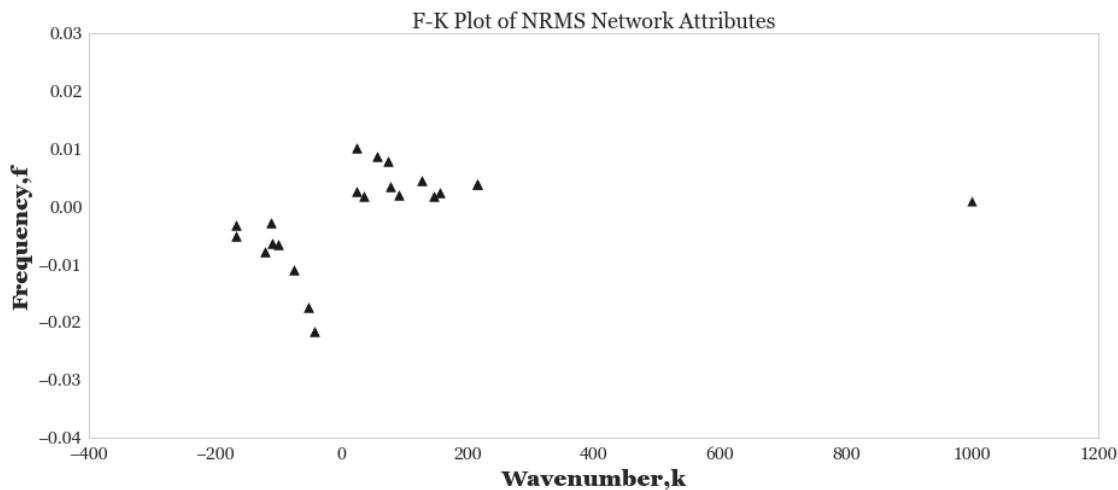
In [854]: emerg_m_df.iloc[np.argsort(k)].head(n=10)

Out [854]:

| | EmergenceMonth |
|---------|----------------|
| aug_99 | 1.000000 |
| oct_2k1 | -0.783987 |
| feb_99 | -0.117516 |
| apr_2k2 | -0.665712 |
| jul_2k1 | -0.315396 |
| nov_99 | 0.561742 |
| jun_2k | -0.232054 |
| dec_2k | 0.452563 |
| sep_99 | 0.164630 |
| sep_2k1 | 0.480412 |

In [855]: f0, k0 = fk_plot(nrms_netatt, 'NRMS Network Attributes')

C:\Users\arsha_000\Anaconda3\lib\site-packages\numpy\core\numeric.py:533: ComplexWarning: return array(a, dtype, copy=False, order=order, subok=True)



In [856]: nrms_netatt_df = pd.DataFrame(nrms_netatt).T
nrms_netatt_df.columns=months[1:]
nrms_netatt_df = nrms_netatt_df.T
nrms_netatt_df.columns = ['EmergenceMonth']
nrms_netatt_df.head()

```

Out[856]:          EmergenceMonth
Dec98           36.822295
jan_99          8.127561
feb_99          9.172371
mar_99          10.302567
apr_99          15.697349

In [857]: nrms_netatt_df.iloc[np.argsort(f0)].head(n=10)

Out[857]:          EmergenceMonth
jun_2k           0.385283
jan_2k1          25.002485
dec_2k           19.067249
jul_2k           8.280636
mar_99           10.302567
apr_2k2          52.801606
mar_2k2          7.766187
apr_99           15.697349
feb_99           9.172371
may_2k2          44.021470

In [858]: nrms_netatt_df.iloc[np.argsort(k0)].head(n=10)

Out[858]:          EmergenceMonth
aug_99           122.043866
nov_2k1          27.351382
oct_99           11.122599
sep_2k1          34.995816
jul_2k1          24.530456
dec_99           5.061962
apr_2k2          52.801606
mar_99           10.302567
oct_2k1          20.617890
sep_99           16.698520

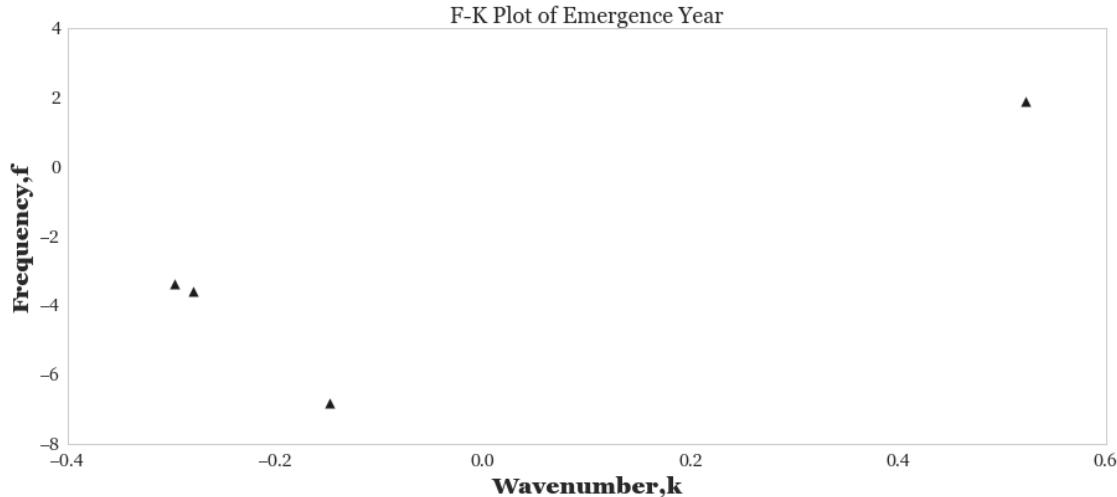
In [859]: print(nrms_netatt_df.iloc[np.argmax(f0)])
print(nrms_netatt_df.iloc[np.argmin(f0)])
print(nrms_netatt_df.iloc[np.argmax(k0)])
print(nrms_netatt_df.iloc[np.argmin(k0)])

EmergenceMonth    36.822295
Name: Dec98, dtype: float64
EmergenceMonth    0.385283
Name: jun_2k, dtype: float64
EmergenceMonth    1.540422
Name: jul_99, dtype: float64
EmergenceMonth    122.043866
Name: aug_99, dtype: float64

```

```
In [860]: f1,k1 = fk_plot(emerg_y_df.values, 'Emergence Year')
```

```
C:\Users\arsha_000\Anaconda3\lib\site-packages\numpy\core\numeric.py:533: ComplexWarning
  return array(a, dtype, copy=False, order=order, subok=True)
```



```
In [861]: print(emerg_y_df.iloc[np.argmax(f1)])
print(emerg_y_df.iloc[np.argmin(f1)])
print(emerg_y_df.iloc[np.argmax(k1)])
print(emerg_y_df.iloc[np.argmin(k1)])
```

```
EmergenceYear      0.522657
Name: 2001, dtype: float64
EmergenceYear     -0.296412
Name: 2000, dtype: float64
EmergenceYear      0.0
Name: 1998, dtype: float64
EmergenceYear      0.0
Name: 1998, dtype: float64
```

10 Attribute Correlations

```
In [862]: stat_all['Emergence']=emerg_m_df
```

```
In [863]: corr_m = stat_all.corr()
```

10.1 Correlation Matrix

```
In [864]: plt.figure(figsize=(18,9))
sns.heatmap(corr_m, cmap='PiYG', robust=True, fmt='d', linewidths=1, square=True)
```

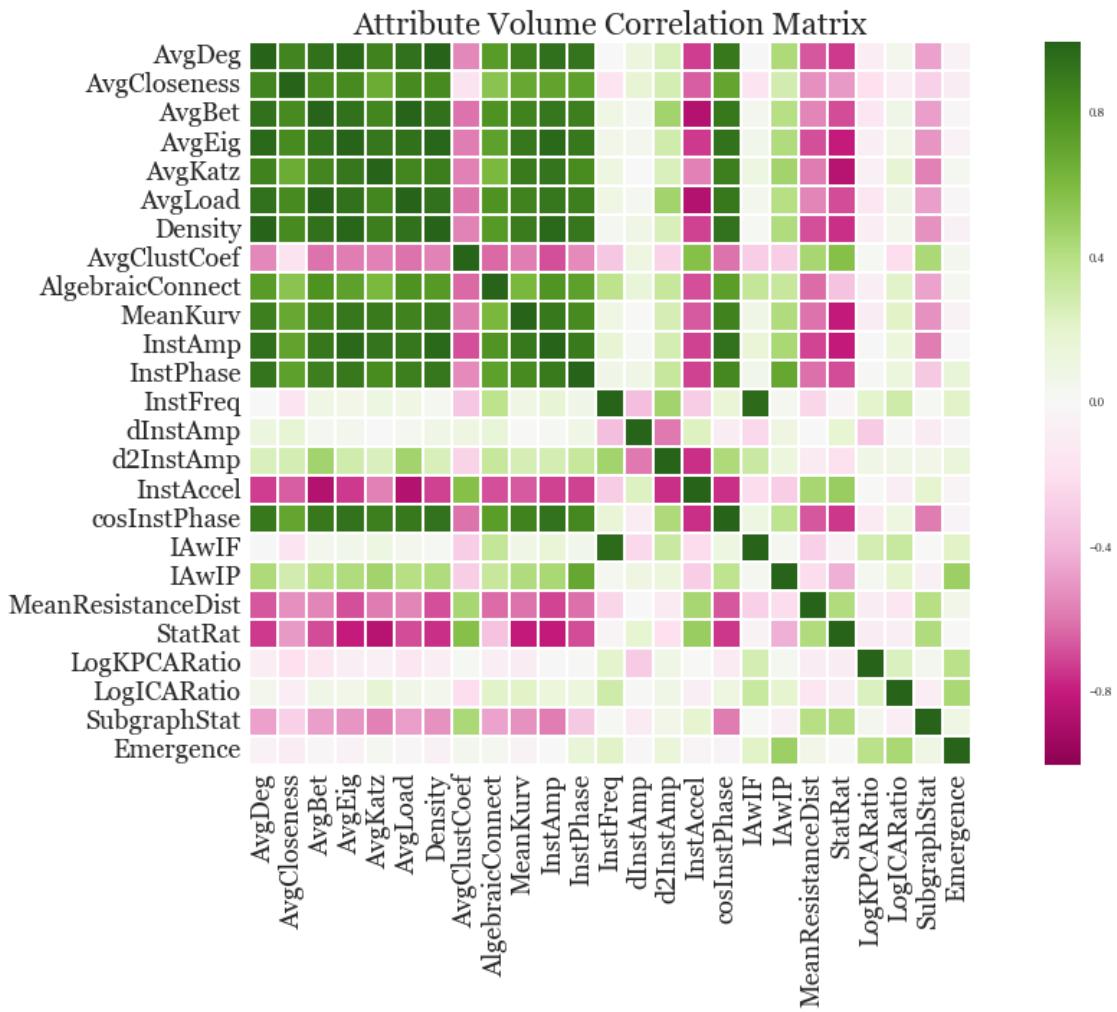
```

plt.title("Attribute Volume Correlation Matrix", fontsize=22)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

Out[864]: (array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,
       9.5, 10.5, 11.5, 12.5, 13.5, 14.5, 15.5, 16.5, 17.5,
      18.5, 19.5, 20.5, 21.5, 22.5, 23.5, 24.5]),

<a list of 25 Text yticklabel objects>

```



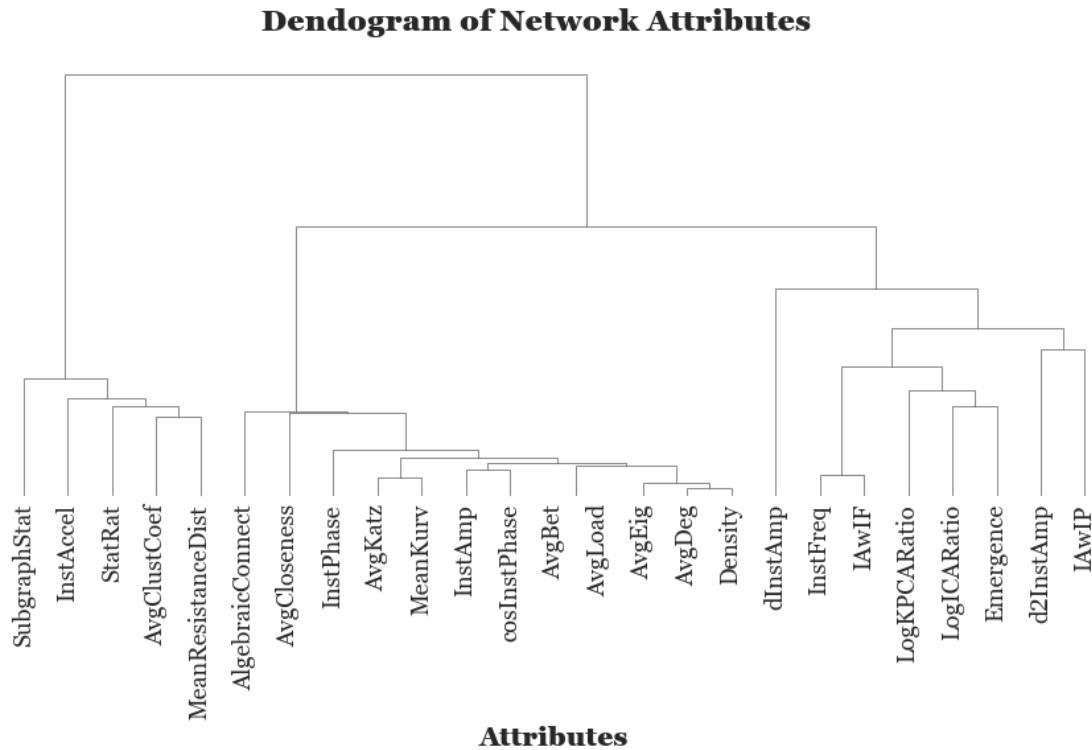
10.2 Correlation Dendrogram

```

In [865]: sns.dendrogram(corr_m)
plt.suptitle("Dendrogram of Network Attributes", fontsize=22)
plt.xlabel("Attributes", fontsize=20)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

```

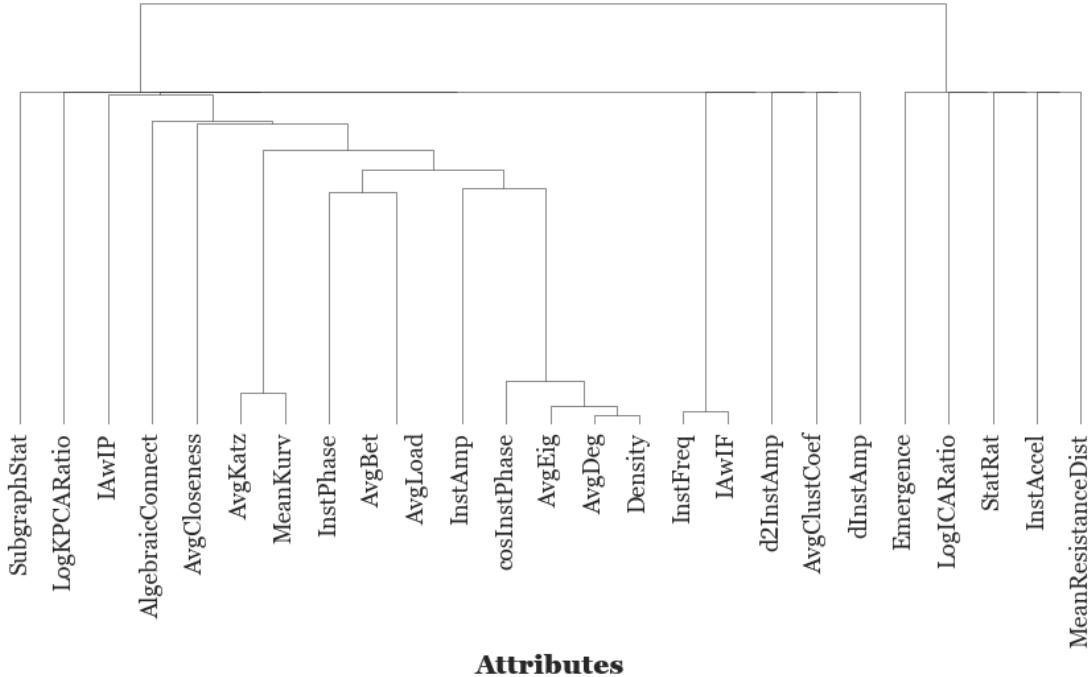
```
Out[865]: (array([], dtype=float64), <a list of 0 Text yticklabel objects>)
```



```
In [960]: g = sns.dendrogram(corr_m, method='average', metric='chebyshev')
plt.suptitle("Dendrogram of Correlation Matrix Network Attributes", fontsize=18)
plt.xlabel("Attributes", fontsize=20)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
```

```
Out[960]: (array([], dtype=float64), <a list of 0 Text yticklabel objects>)
```

Dendrogram of Correlation Matrix Network Attributes



```
In [867]: g_reordered_ind[:5]
```

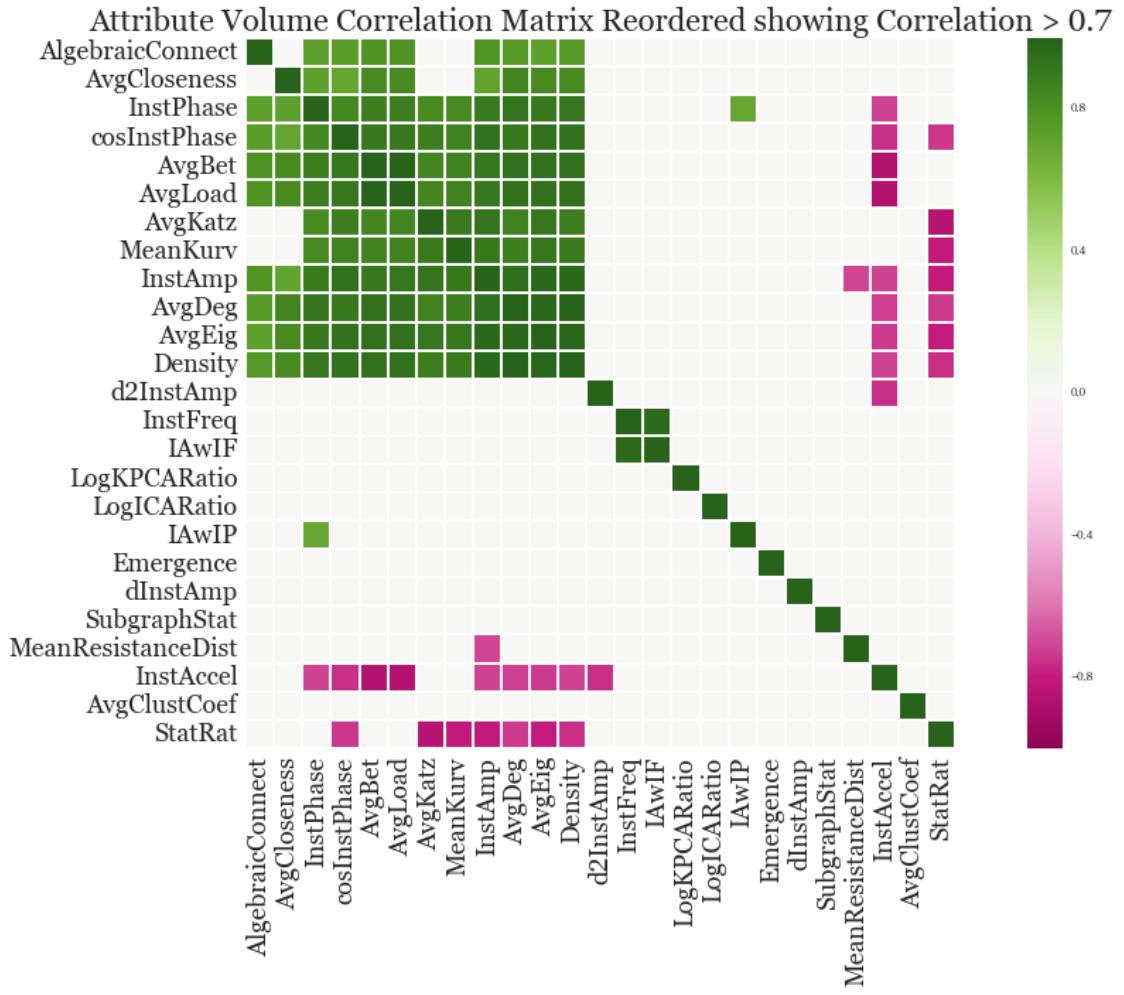
```
Out[867]: [8, 1, 11, 16, 2]
```

10.3 Correlation Matrix > 0.7

```
In [868]: threshold = 0.7
corr_m.values[np.where(abs(corr_m.values) < threshold)] = 0
```

```
In [869]: plt.figure(figsize=(18, 9))
sns.heatmap(corr_m.iloc[g_reordered_ind, g_reordered_ind], cmap='PiYG', ro
plt.title("Attribute Volume Correlation Matrix Reordered showing Correlat
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
```

```
Out[869]: (array([ 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,
9.5, 10.5, 11.5, 12.5, 13.5, 14.5, 15.5, 16.5, 17.5,
18.5, 19.5, 20.5, 21.5, 22.5, 23.5, 24.5]),
<a list of 25 Text yticklabel objects>)
```



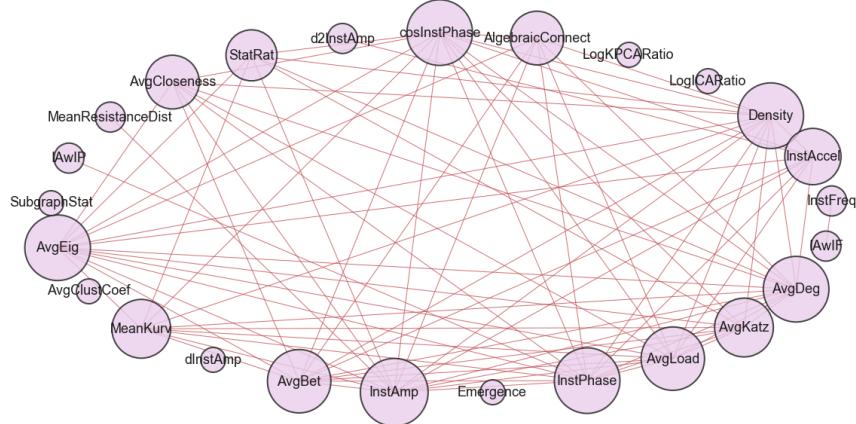
10.4 Correlation Network of Attributes

```
In [870]: names = corr_m.index.values

In [1000]: G_corr = nx.Graph(corr_m.values)
            pos=nx.fruchterman_reingold_layout(G_corr, iterations=1000, k=200)

In [1023]: #ref: https://github.com/traims/correlation-based-networks/blob/master/
            components = nx.connected_components(G_corr)
            plt.figure(figsize=(26,12))
            plt.suptitle("Correlation Network of Attributes", fontsize=33)
            plt.axis('off')
            for i in components:
                component = G_corr.subgraph(i)
                nx.draw_networkx(component, with_labels = True, node_size = [component
                    labels = dict([(x, names[x]) for x in component.nodes()]), pos=
                    edge_color='r', node_color='#E9CFEC', linewidths=2,
```

Correlation Network of Attributes



```
In [1027]: nx.write_gexf(G_corr, 'corr_mat_att.gexf')
```

11 Feature Ranking with Gradient Boosting Regressor

11.1 Feature Ranking for predicting Emergence

Here I use gradient boosting regression to predict Emergence based on all the other network attributes. Since Emergence is a global measure that indicates changes in activity over time. Since the attribute volume is small I use a 50/50 test/train split for this analysis. Also this should prevent overfitting from this sample size.

```
In [930]: stat_all.drop(['Emergence'], axis=1).apply(sc.stats.zscore, axis=1).head()
```

```
Out[930]:          AvgDeg  AvgCloseness  AvgBet  AvgEig  AvgKatz  AvgLoad  ...
Nov98   -0.231771    -0.111925 -0.275049 -0.120884 -0.128583 -0.275049
Dec98   -0.465140    -0.289901 -0.570673 -0.368687 -0.411719 -0.570673
jan_99  -0.559882    -0.505132 -0.570289 -0.479407 -0.601511 -0.570289
feb_99   0.161135     0.237556 -0.068129  0.256099  0.010127 -0.068129
mar_99  -0.505708    -0.397612 -0.566513 -0.367139 -0.364003 -0.566513

          Density  AvgClustCoef  AlgebraicConnect  MeanKurv  ...
Nov98    0.027894    -0.577992      1.845554 -1.053605 ...
Dec98   -0.334133    -0.283186      1.074190 -1.086004 ...
jan_99  -0.445404    -0.695175      0.928336 -1.030909 ...
feb_99   0.390399    -0.526657      0.848927 -0.607472 ...
mar_99  -0.262491    -0.870534      0.953594 -1.157661 ...

d2InstAmp  InstAccel  cosInstPhase  IAWIF  IAWIP \
```

```

Nov98    -0.610365   -0.695625      0.318457  -1.102557  0.258862
Dec98    -0.744127   -0.749261      -0.347707   2.432245  1.533786
jan_99   -0.685971   -0.930407      -0.452545   0.857258  2.983107
feb_99   -0.487903   -1.147434      0.391850  -0.526657  3.206809
mar_99   -0.764361   -1.380367      0.133966   1.044495  2.236832

          MeanResistanceDist  StatRat  LogKPCARatio  LogICARatio  Subgraph
Nov98            3.308987  1.589693     -0.577992  -0.577992  -0.577992
Dec98            2.522560  0.908998     -1.287001   0.711131  -0.706131
jan_99           2.148026  0.816231      0.178920  1.371068  -0.661068
feb_99           0.835815  0.538866     -1.973649  -1.617663  -0.517663
mar_99           1.826949  0.709207      2.225776  -0.177059  -0.847059

[5 rows x 24 columns]

```

```

In [931]: X= stat_all.drop(['Emergence'],axis=1).apply(sc.stats.zscore, axis=1)
y = stat_all.Emergence

In [932]: X = X.astype(np.float32)
offset = int(X.shape[0] * 0.5)
X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]

In [933]: from sklearn.metrics import mean_squared_error
from sklearn import ensemble

In [934]: params = {'n_estimators': 10, 'max_depth': 10, 'min_samples_split': 10,
               'learning_rate': 0.1, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)
clf.fit(X_train, y_train)

Out[934]: GradientBoostingRegressor(alpha=0.9, init=None, learning_rate=0.1, loss='ls',
                                     max_depth=10, max_features=None, max_leaf_nodes=None,
                                     min_samples_leaf=1, min_samples_split=10,
                                     min_weight_fraction_leaf=0.0, n_estimators=10, presort='auto',
                                     random_state=None, subsample=1.0, verbose=0, warm_start=False)

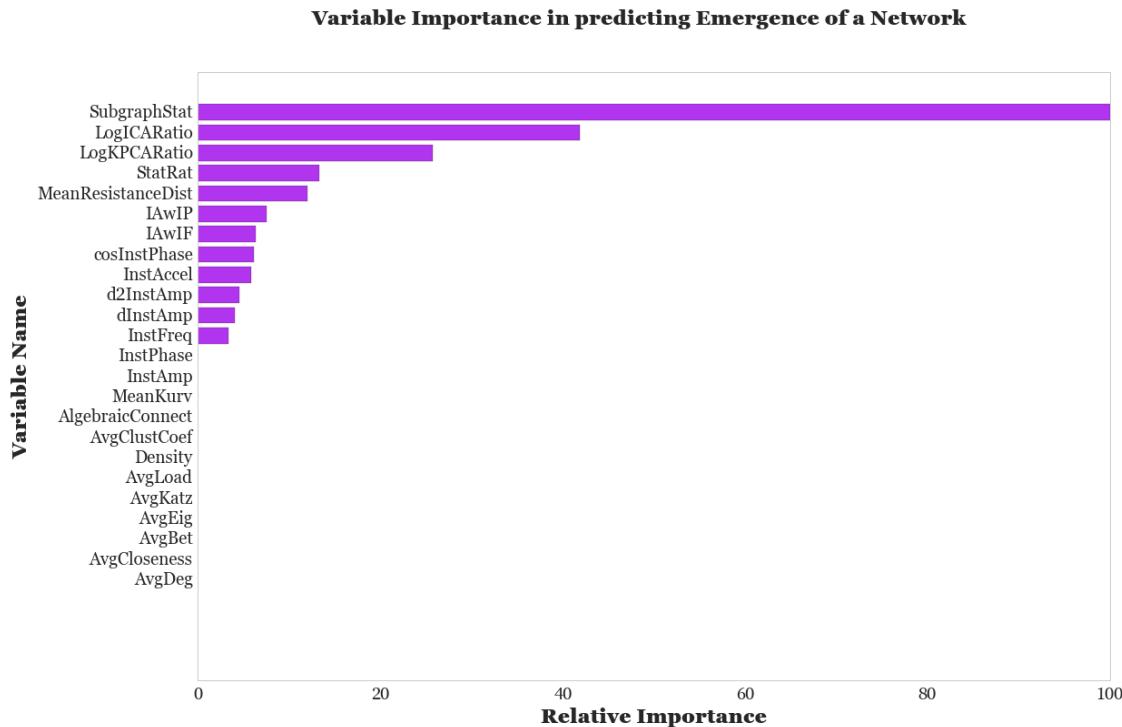
In [935]: feature_importance = clf.feature_importances_

# make importances relative to max importance
plt.figure(figsize=(18,12))
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
feature_names = list(X)
pos = np.arange(sorted_idx.shape[0])
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, feature_names, fontsize=19)
plt.ylabel("Variable Name", fontsize=22)
plt.xlabel('Relative Importance', fontsize=22)

```

```
plt.suptitle('Variable Importance in predicting Emergence of a Network',
             plt.xticks(fontsize=18)
```

```
Out[935]: (array([ 0., 20., 40., 60., 80., 100.]),
            <a list of 6 Text xticklabel objects>)
```

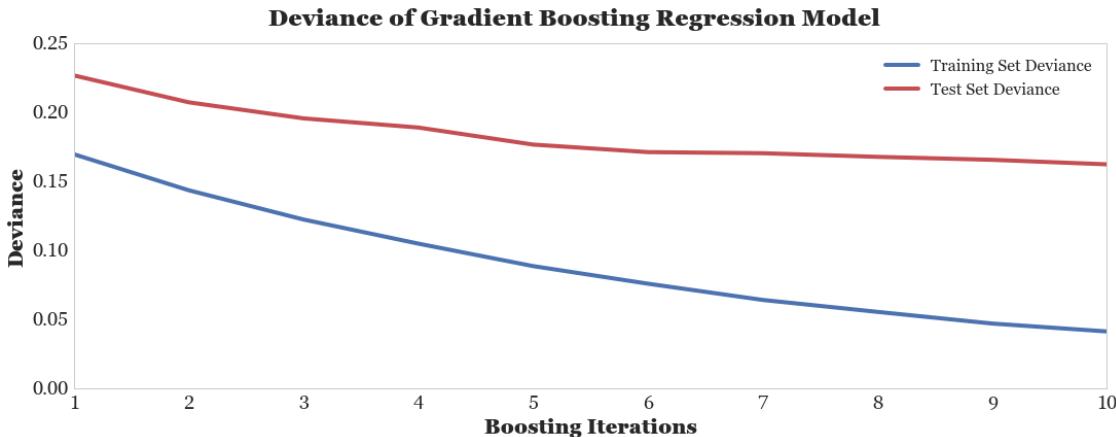


```
In [936]: test_score = np.zeros((params['n_estimators']),), dtype=np.float64)
```

```
for i, y_pred in enumerate(clf.staged_predict(X_test)):
    test_score[i] = clf.loss_(y_test, y_pred)

plt.figure(figsize=(18, 6))
plt.suptitle('Deviance of Gradient Boosting Regression Model', fontsize=22)
plt.plot(np.arange(params['n_estimators']) + 1, clf.train_score_, 'b-',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc=1, fontsize=16)
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
```

```
Out[936]: (array([ 0. , 0.05, 0.1 , 0.15, 0.2 , 0.25]),
            <a list of 6 Text yticklabel objects>)
```



```
In [937]: mse = mean_squared_error(y_test, clf.predict(X_test))
print("Gradient Boosting MSE: ", mse)

Gradient Boosting MSE:  0.162520869208
```

11.2 Feature Ranking for predicting Average Degree

```
In [938]: X= stat_all.drop(['AvgDeg'],axis=1).apply(sc.stats.zscore, axis=1)
y = stat_all.AvgDeg

In [939]: X = X.astype(np.float32)
offset = int(X.shape[0] * 0.5)
X_train, y_train = X[:offset], y[:offset]
X_test, y_test = X[offset:], y[offset:]

In [940]: from sklearn.metrics import mean_squared_error
from sklearn import ensemble

In [941]: params = {'n_estimators': 10, 'max_depth': 10, 'min_samples_split': 10,
               'learning_rate': 0.1, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)
clf.fit(X_train, y_train)

Out[941]: GradientBoostingRegressor(alpha=0.9, init=None, learning_rate=0.1, loss='ls',
                                     max_depth=10, max_features=None, max_leaf_nodes=None,
                                     min_samples_leaf=1, min_samples_split=10,
                                     min_weight_fraction_leaf=0.0, n_estimators=10, presort='auto',
                                     random_state=None, subsample=1.0, verbose=0, warm_start=False)

In [942]: feature_importance = clf.feature_importances_

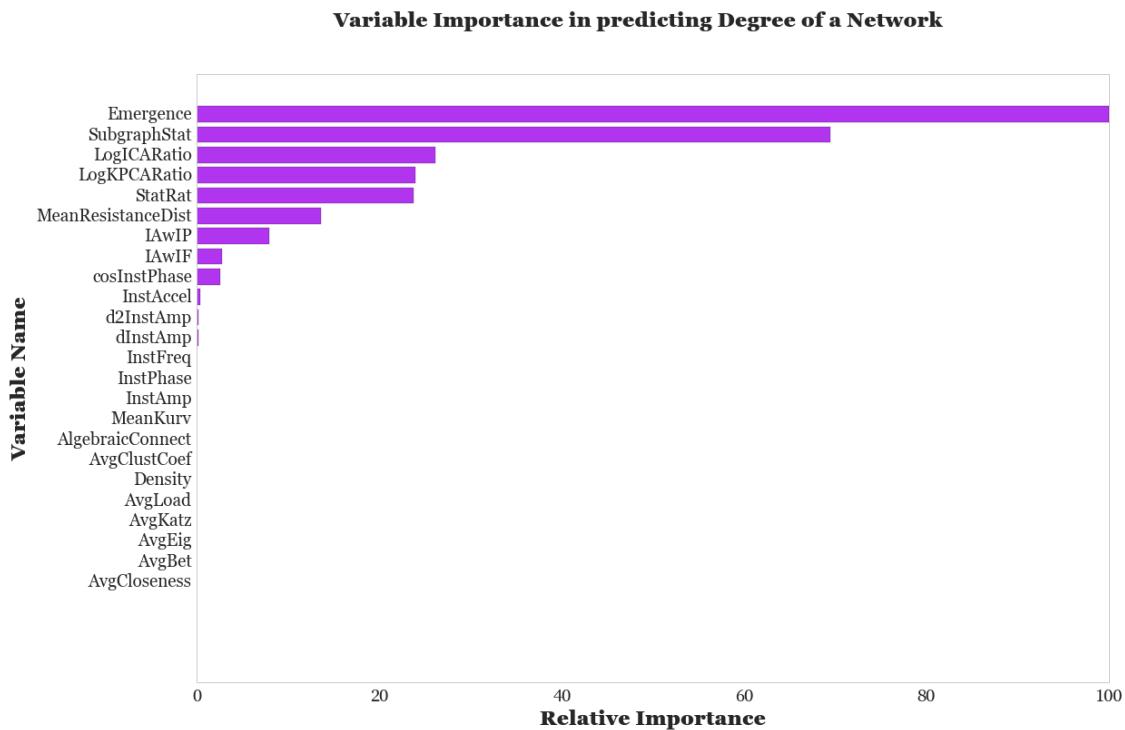
# make importances relative to max importance
```

```

plt.figure(figsize=(18,12))
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
feature_names = list(X)
pos = np.arange(sorted_idx.shape[0])
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, feature_names, fontsize=19)
plt.ylabel("Variable Name", fontsize=22)
plt.xlabel('Relative Importance', fontsize=22)
plt.suptitle('Variable Importance in predicting Degree of a Network', fontweight='bold', fontsize=22)
plt.xticks(fontsize=18)

Out[942]: (array([ 0., 20., 40., 60., 80., 100.]),
<a list of 6 Text xticklabel objects>)

```



```

In [943]: test_score = np.zeros((params['n_estimators']),), dtype=np.float64)

for i, y_pred in enumerate(clf.staged_predict(X_test)):
    test_score[i] = clf.loss_(y_test, y_pred)

plt.figure(figsize=(18, 6))
plt.suptitle('Deviance of Gradient Boosting Regression Model', fontsize=22)
plt.plot(np.arange(params['n_estimators']) + 1, clf.train_score_, 'b-',
         label='Training Set Deviance')

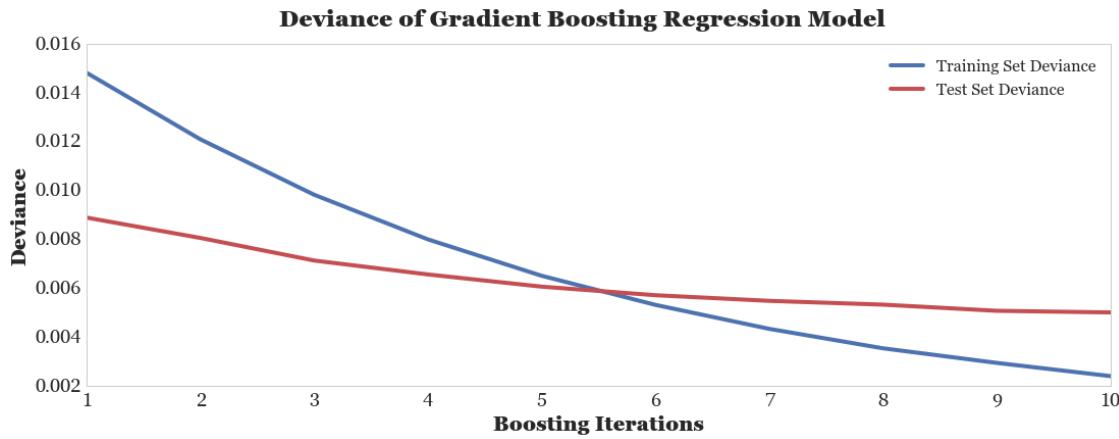
```

```

plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc=1, fontsize=16)
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

Out[943]: (array([ 0.002,  0.004,  0.006,  0.008,  0.01 ,  0.012,  0.014,  0.016]),<a list of 8 Text yticklabel objects>)

```



```

In [944]: mse = mean_squared_error(y_test, clf.predict(X_test))
          print("Gradient Boosting MSE: " ,mse)

```

Gradient Boosting MSE: 0.00501546942411

12 Signal to Noise ratio

Here I calculate the signal to noise ratio of the attribute volume. This is done for each variable by:

$$SNR = \frac{Mean_{att}}{StdDev_{att}}$$

```

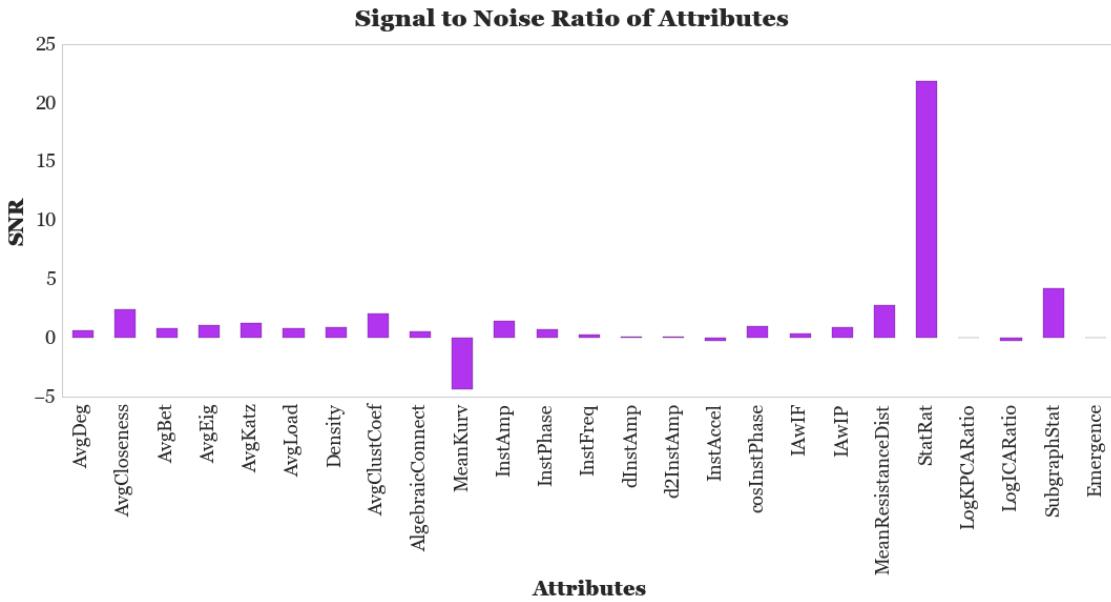
In [949]: (stat_all.mean()/stat_all.std()).plot(kind='bar', fontsize=18, figsize=(18, 10))
          plt.suptitle('Signal to Noise Ratio of Attributes', fontsize=22)
          plt.xlabel('Attributes')
          plt.ylabel('SNR')
          plt.xticks(fontsize=18)
          plt.yticks(fontsize=18)

```

```

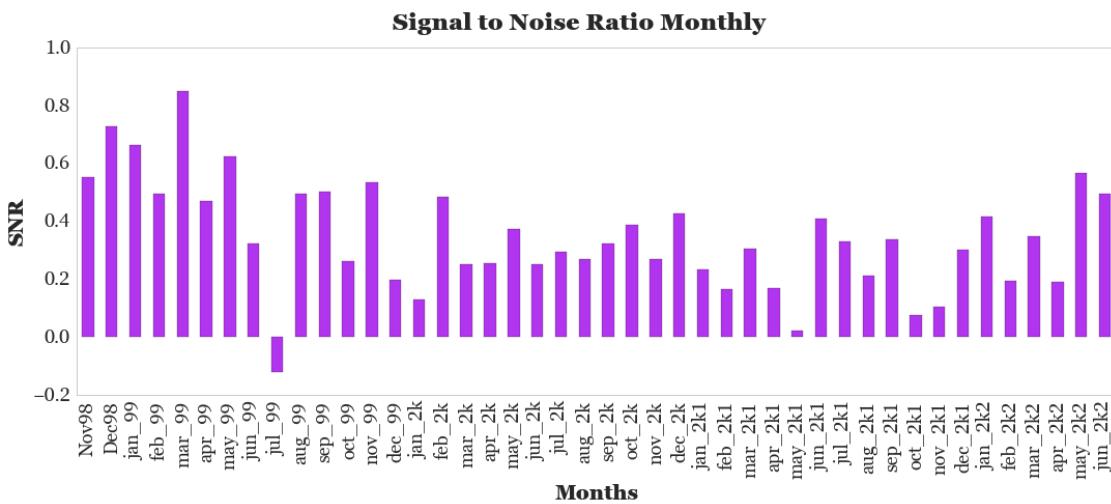
Out[949]: (array([-5.,  0.,  5., 10., 15., 20., 25.]),<a list of 7 Text yticklabel objects>)

```



```
In [950]: (stat_all.mean(axis=1)/stat_all.std(axis=1)).plot(kind='bar', fontsize=18)
plt.suptitle('Signal to Noise Ratio Monthly', fontsize=22)
plt.xlabel('Months')
plt.ylabel('SNR')
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
```

```
Out[950]: (array([-0.2,  0.,  0.2,  0.4,  0.6,  0.8,  1.,  1.2]), 
<a list of 8 Text yticklabel objects>)
```



```
In [996]: stat_all.to_excel('enron_dynet_attvol.xlsx')
```

```
In [ ]:
```