

Enron email data set exploration

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import re
# for better looking pictures
%config InlineBackend.figure_format = 'retina'

In [2]: df = pd.read_feather('enron.feather')
df = df.sort_values(['Date','MailID'])
df.tail(5)

Out[2]: MailID Date From To Recipients Subject filename
603936 72679 2002-07-12 nancy.ehring ge_benefits 1 ABRA HRIS System m/discussion_threads/334...
604785 72797 2002-07-12 nancy.ehring ge_benefits 1 ABRA HRIS System fischer-m/notes/inbox/5...
603131 72402 2002-07-12 mark.fisher tom.nemila 1 WR613 Pitch System performance m/discussion_threads/336...
604568 72905 2002-07-12 kurt.anderson jeffduff 3 FW: RE: Revised Availability Numbers fischer-m/notes/inbox/4...
602333 72225 2002-07-12 angie.lentz ge_benefits 1 Essential IT information for Monday 7/15 Netwo... fischer-m/all_documents/427...
```

Email traffic over time

Group the data by Date and MailID, which will get you an index that collects all of the unique mail IDs per date. Then reset the index so that those date and mail identifiers become columns and then select for just those columns; we don't actually care about the counts created by the groupby (that was just to get the index). Create a histogram that shows the amount of traffic per day. Then specifically for email sent from richard.shapiro, and then john.lavorato. Because some dates are set improperly (to 1980), filter for dates greater than January 1, 1999.

```
In [3]: # condition
df_filter = df[df['Date'] > pd.Timestamp('1999,1,1')]

# deal with data
traffic = df_filter.groupby(['Date','MailID']).count().reset_index()

# Create a histogram
fig, ax = plt.subplots(figsize=(12,4))
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.hist(traffic['Date'],bins=100,color="#EDCA3B",edgecolor="black",lw=0.5)
plt.show()

D:\jupyter\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py:93: FutureWarning: In a future version these will be deprecated. Compare in descending order to match the standard library behavior. Use `ts.datestr()` or `ts.date()` instead.
result = libops.scalar_COMPARE(x.ravel(), y, op)
```

```
In [4]: # condition
df_filter = df[(df['Date'] > pd.Timestamp('1999,1,1')) & (df['From']=='richard.shapiro')]

# deal with data
traffic = df_filter.groupby(['Date','MailID']).count().reset_index()

# Create a histogram
fig, ax = plt.subplots(figsize=(12,4))
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.hist(traffic['Date'],bins=100,color="#EDCA3B",edgecolor="black",lw=0.5)
plt.show()
```

```
In [5]: # condition
df_filter = df[(df['Date'] > pd.Timestamp('1999,1,1')) & (df['From']=='john.lavorato')]

# deal with data
traffic = df_filter.groupby(['Date','MailID']).count().reset_index()

# Create a histogram
fig, ax = plt.subplots(figsize=(12,4))
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.hist(traffic['Date'],bins=100,color="#EDCA3B",edgecolor="black",lw=0.5)
plt.show()
```

Received emails

Count the number of messages received per user and then sort in reverse order. Make a bar chart showing the top 30 email recipients.

```
In [5]: # deal with data
df_recv = df.groupby(['To'],count)
df_recv = df_recv.sort_values('MailID',ascending=False).head(30)

# Create a bar chart
fig, ax = plt.subplots(figsize=(12,4))
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.bar(df_recv.index,df_recv['MailID'],rotation=90)
plt.show()

<ipython-input-5-0114a611f28>:10: UserWarning: FixedFormatter should only be used together with FixedLocator
ax.set_xticklabels(df_recv.index,rotation=90)
```

Sent emails

Make a bar chart indicating the top 30 mail senders. This is more complicated than the received emails because a single person can email multiple people in a single email. So, group by From and MailID, convert the index back to columns and then group again by From and get the count.

```
In [4]: # deal with data
df_send = df.groupby(['From','MailID']).count().reset_index()
columns = ['From','To']
df_send = df_send.groupby(['From']).count()
df_send = df_send.sort_values('To',ascending=False).head(30)

# Create a bar chart
fig, ax = plt.subplots(figsize=(12,4))
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.bar(df_send.index,df_send['To'])
ax.set_xticklabels(df_send.index,rotation=90)
plt.show()

<ipython-input-4-re802bf0d63>:12: UserWarning: FixedFormatter should only be used together with FixedLocator
ax.set_xticklabels(df_send.index,rotation=90)
```

Email heatmap

Given a list of Enron employees, compute a heat map that indicates how much email traffic went between each pair of employees. The heat map is not symmetric because Susan sending mail to Xue is not the same thing as Xue sending mail to Susan. The first step is to group the data frame by From and To columns in order to get the number of emails from person \$1 to person \$2. Then, create a 2D numpy matrix, \$C\\$S\$, of integers and set \$C[\\$i][\\$j]\$ to the count of person \$1\\$ to person \$2\$. Then, using matplotlib, ax.imshow(\$C\\$S\$, vmax=4000), show the heat map and add tick labels at 45 degrees for the X axis. Set the labels to the appropriate names. Draw the number of emails in the appropriate cells of the heat map, for all values greater than zero. Please note that when you draw text using ax.text(), the coordinates are X/Y whereas the coordinates in the \$C\\$S\$ matrix are row/column so you will have to flip the coordinates.

```
In [8]: people = ['jeff.skilling', 'kenneth.lay', 'louise.kitchen', 'tana.jones', 'sara.shackleton', 'vince.kaminski', 'sally.beck', 'john.lavorato', 'mark.taylor', 'greg.whalley', 'jeff.dasovich', 'steven.kean', 'eric.bass', 'david.foster', 'mark.mayer', 'david.deinney', 'susan.scott', 'gerald.mcmreec', 'carol.mind', 'daren.farmer', 'jeff.lavorato', 'mike.wernette', 'philip.love', 'michael.sanders', 'benjamin.schoen', 'alan.comes', 'sandra.maccann', 'ellen.kramer', 'elizabeth.sager', 'lisa.gomes', 'william.burford', 'suzanne.brown', 'matthew.green']

# deal with data
df_matrix = df.groupby(['From','To']).count()
C = np.zeros((len(people),len(people)))
for i, person_i in enumerate(people):
    for j, person_j in enumerate(people):
        try:
            C[i][j] = df_matrix.loc[from_s,to_s]['MailID']
        except:
            C[i][j] = 0

# heat map
fig, ax = plt.subplots(rows=1, nocols=1, figsize=(12,12))
im = ax.imshow(C, cmap='GnBu', vmax=4000)

for x in range(len(people)):
    for y in range(len(people)):
        if C[y,x]>0:
            ax.text(x,y,"%d" % C[y,x],horizontalalignment="center")

ax.set_xticks(range(len(people)))
ax.set_yticks(range(len(people)))
ax.set_xticklabels(people,rotation=45)
ax.set_yticklabels(people,rotation=45)
ax.set_title(" Heat Map : From v.s. To",weight="bold")

fig.colorbar(im,ax=ax, shrink=0.82)
plt.show()
```

Build graph and compute rankings

From the data frame, create a graph data structure using networkx. Create an edge from node A to node B if there is an email from A to B in the data frame. Although we do know the total number of emails between people, let's keep it simple and use simply a weight of 1 as the edge label. See networkx method add_edge() .

1. Using networkx, compute the pagerank between all nodes. Get the data into a data frame, sort in reverse order, and display the top 15 users from the data frame.
2. Compute the centrality for the nodes of the graph. The documentation says that centrality is "the fraction of nodes it is connected to."

I use DataFrame.from_dict to convert the dictionaries returned from the various networkx methods to data frames.

Node PageRank

```
In [10]: # create Digraph
G = nx.DiGraph()
edges = []
for t, r in zip(df['From'].values, df['To'].values):
    G.add_edge(t, r, weight=1)
PR = nx.pagerank(G)
df_pagerank = pd.DataFrame.from_dict(PR,orient='index')
df_pagerank.columns = ['PageRank']
PR_Top15 = df_pagerank.sort_values('PageRank',ascending=False).head(15)
PR_Top15
```

Centrality

```
In [11]: CG = nx.degree_centrality(G)
df_centrality = pd.DataFrame.from_dict(CG,orient='index')
df_centrality.columns = ['Centrality']
CG_Top15 = df_centrality.sort_values('Centrality',ascending=False).head(15)
CG_Top15
```

Plotting graph subsets

The email graph is way too large to display the whole thing and get any meaningful information out. However, we can look at subsets of the graph such as the neighbors of a specific node. To visualize we can use different strategies to layout the nodes. In this case, we will use two different layout strategies: spring and kamada-kawai. According to Wikipedia, these forced directed layout strategies have the characteristic: "the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart..."

Use networkx ego_graph() method to get a radius=1 neighborhood around jeff.skilling and draw the spring graph with a plot that is 20x20 inch so we can see details. Then, draw the same subgraph again using the kamada-kawai layout strategy. Finally, get the neighborhood around kenneth.lay and draw kamada-kawai.

```
In [12]: # spring layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'jeff.skilling', radius=1)
seed=np.random.RandomState(50)
plot = nx.spring_layout(hub_ego, seed=seed)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

```
# kamada-kawai layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'jeff.skilling', radius=1)
plot = nx.kamada_kawai.layout(hub_ego)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

Build graph and compute rankings

From the data frame, create a graph data structure using networkx. Create an edge from node A to node B if there is an email from A to B in the data frame. Although we do know the total number of emails between people, let's keep it simple and use simply a weight of 1 as the edge label. See networkx method add_edge() .

1. Using networkx, compute the pagerank between all nodes. Get the data into a data frame, sort in reverse order, and display the top 15 users from the data frame.
2. Compute the centrality for the nodes of the graph. The documentation says that centrality is "the fraction of nodes it is connected to."

I use DataFrame.from_dict to convert the dictionaries returned from the various networkx methods to data frames.

Node PageRank

```
In [10]: # create Digraph
G = nx.DiGraph()
edges = []
for t, r in zip(df['From'].values, df['To'].values):
    G.add_edge(t, r, weight=1)
PR = nx.pagerank(G)
df_pagerank = pd.DataFrame.from_dict(PR,orient='index')
df_pagerank.columns = ['PageRank']
PR_Top15 = df_pagerank.sort_values('PageRank',ascending=False).head(15)
PR_Top15
```

Centrality

```
In [11]: CG = nx.degree_centrality(G)
df_centrality = pd.DataFrame.from_dict(CG,orient='index')
df_centrality.columns = ['Centrality']
CG_Top15 = df_centrality.sort_values('Centrality',ascending=False).head(15)
CG_Top15
```

Plotting graph subsets

The email graph is way too large to display the whole thing and get any meaningful information out. However, we can look at subsets of the graph such as the neighbors of a specific node. To visualize we can use different strategies to layout the nodes. In this case, we will use two different layout strategies: spring and kamada-kawai. According to Wikipedia, these forced directed layout strategies have the characteristic: "the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart..."

Use networkx ego_graph() method to get a radius=1 neighborhood around kenneth.lay and draw the spring graph with a plot that is 20x20 inch so we can see details. Then, draw the same subgraph again using the kamada-kawai layout strategy. Finally, get the neighborhood around kenneth.lay and draw kamada-kawai.

```
In [12]: # spring layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'kenneth.lay', radius=1)
seed=np.random.RandomState(50)
plot = nx.spring_layout(hub_ego, seed=seed)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

```
# kamada-kawai layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'kenneth.lay', radius=1)
plot = nx.kamada_kawai.layout(hub_ego)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

Build graph and compute rankings

From the data frame, create a graph data structure using networkx. Create an edge from node A to node B if there is an email from A to B in the data frame. Although we do know the total number of emails between people, let's keep it simple and use simply a weight of 1 as the edge label. See networkx method add_edge() .

1. Using networkx, compute the pagerank between all nodes. Get the data into a data frame, sort in reverse order, and display the top 15 users from the data frame.
2. Compute the centrality for the nodes of the graph. The documentation says that centrality is "the fraction of nodes it is connected to."

I use DataFrame.from_dict to convert the dictionaries returned from the various networkx methods to data frames.

Node PageRank

```
In [10]: # create Digraph
G = nx.DiGraph()
edges = []
for t, r in zip(df['From'].values, df['To'].values):
    G.add_edge(t, r, weight=1)
PR = nx.pagerank(G)
df_pagerank = pd.DataFrame.from_dict(PR,orient='index')
df_pagerank.columns = ['PageRank']
PR_Top15 = df_pagerank.sort_values('PageRank',ascending=False).head(15)
PR_Top15
```

Centrality

```
In [11]: CG = nx.degree_centrality(G)
df_centrality = pd.DataFrame.from_dict(CG,orient='index')
df_centrality.columns = ['Centrality']
CG_Top15 = df_centrality.sort_values('Centrality',ascending=False).head(15)
CG_Top15
```

Plotting graph subsets

The email graph is way too large to display the whole thing and get any meaningful information out. However, we can look at subsets of the graph such as the neighbors of a specific node. To visualize we can use different strategies to layout the nodes. In this case, we will use two different layout strategies: spring and kamada-kawai. According to Wikipedia, these forced directed layout strategies have the characteristic: "the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart..."

Use networkx ego_graph() method to get a radius=1 neighborhood around kenneth.lay and draw the spring graph with a plot that is 20x20 inch so we can see details. Then, draw the same subgraph again using the kamada-kawai layout strategy. Finally, get the neighborhood around kenneth.lay and draw kamada-kawai.

```
In [12]: # spring layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'kenneth.lay', radius=1)
seed=np.random.RandomState(50)
plot = nx.spring_layout(hub_ego, seed=seed)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

```
# kamada-kawai layout
fig, ax = plt.subplots(figsize=(20,20))
hub_ego = nx.ego_graph(G, 'kenneth.lay', radius=1)
plot = nx.kamada_kawai.layout(hub_ego)
nx.draw(hub_ego, plot, node_size=0, with_labels=True, edge_color="#787878", width=.1, arrows=False, font_size=6)
```

Build graph and compute rankings

From the data frame, create a graph data structure using networkx. Create an edge from node A to node B if there is an email from A to B in the data frame. Although we do know the total number of emails between people, let's keep it simple and use simply a weight of 1 as the edge label. See networkx method add_edge() .

1. Using networkx, compute the pagerank between all nodes. Get the data into a data frame, sort in reverse order, and display the top 15 users from the data frame.
2. Compute the centrality for the nodes of the graph. The documentation says that centrality is "the fraction of nodes it is connected to."

I use DataFrame.from_dict to convert the dictionaries returned from the various networkx methods to data frames.

Node PageRank

```
In [10]: # create Digraph
G = nx.DiGraph()
edges = []
for t, r in zip(df['From'].values, df['To'].values):
    G.add_edge(t, r, weight=1)
PR = nx.pagerank(G)
df_pagerank = pd.DataFrame.from_dict(PR,orient='index')
df_pagerank.columns = ['PageRank']
PR_Top15 = df_pagerank.sort_values('PageRank',ascending=False).head(15)
PR_Top15
```

Centrality

```
In [11]: CG = nx.degree_centrality(G)
df_centrality = pd.DataFrame.from_dict(CG,orient='index')
df_centrality.columns = ['Centrality']
CG_Top15 = df_centrality.sort_values('Centrality',ascending=False).head(15)
CG_Top15
```

Plotting graph subsets

The email graph is way too large to display the whole thing and get any meaningful information out. However, we can look at subsets of the graph such as the neighbors of a specific node. To visualize we can use different strategies to layout the nodes. In this case, we will use two different layout strategies: spring and kamada-kawai. According to Wikipedia, these forced directed layout strategies have the characteristic: "the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart..."

Use networkx ego_graph() method to get a radius=1 neighborhood around kenneth.lay and draw the spring graph

