Enron Compiled

January 26, 2021

1 Python Programming Team Project by Max Needle, Sammie Kim, and Toby Du

This project focuses on the analysis of the Enron Emails corpus. We approached it by using sentiment analysis, organizational network analysis (ONA), topic clustering, and a number of visualizations. In this project, we first extracted data from each email (i.e., date, message, recipient, sender) and incorporated additional information about the job level of the email senders ("Titles.xlsx") and Enron's monthly stock price ("Enron_Monthly.xlsx"). We then conducted sentiment analysis and topic clustering (5 topics) on the messages. We used ONA and sentiment analysis to investigate differences in network centrality (degree and betweenness centrality) and email sentiment between emails sent by management and those sent by other employees. We then used email topic probability (for each topic), email sentiment, and Enron's stock price to investigate correlations between these variables over time. Finally, we used scatterplots, word clouds, network diagrams and time series visualization to make inferences.

[87]:
```
# import packages
import numpy as np
import pandas as pd
import seaborn as sns
import glob
import os
from email.parser import Parser
from afinn import Afinn
from wordcloud import WordCloud
from scipy import stats
from datetime import datetime
import networkx as nx
from collections import Counter
import re
import pingouin as pg
import researchpy as rp
import plotly.express as px
from textblob import TextBlob
import statistics
import wget
import itertools
```

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline

from statsmodels.stats.anova import anova_lm
from statsmodels.formula.api import ols
from statsmodels.graphics.factorplots import interaction_plot
import statsmodels.stats.multicomp
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import statsmodels.api as sm
```

[73]:
```
# set max view
pd.set_option('display.max_rows', 100000)
pd.set_option('display.max_columns', 100000)
pd.set_option('display.max_colwidth',1000)
```

2 Construct full df

2.0.1 Make Series of all full emails

[10]:
```
# Input: path to directory
# Processing: iterates through path to get to every file
# Output: list of filepaths in a directory

def getListOfFiles(dirName):
    listOfFile = os.listdir(dirName)
    allFiles = list()
    for entry in listOfFile:
        fullPath = os.path.join(dirName, entry)
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath) # recursion
        else:
            allFiles.append(fullPath)

    return allFiles
```

[8]:
```
# downloading and unzipping the Enron Emails corpus takes several minutes
wget.download('https://www.cs.cmu.edu/~enron/enron_mail_20150507.tar.
↪gz','enron_mail_20150507.tar.gz')
```

```
!tar xzf enron_mail_20150507.tar.gz
```

[11]:
```
# get all filepaths in main directory
files= getListOfFiles('maildir/')
```

[74]:
```
# create a DataFrame with all of the full emails
emails= [open(f, mode='r', encoding="utf8", errors='ignore').read() for f in␣
 ↪files]
df= pd.Series(emails)
df= df.to_frame('Email')
```

[75]:
```
# convert the email entries to Parser datatypes to easily extract info on␣
 ↪sender, recipients, subject, message, date
df['Email']= df['Email'].apply(Parser().parsestr)
```

2.0.2 Extract message info

[76]:
```
# make a new column for the subject and message of the email (to be used later␣
 ↪for topic modeling and wordclouds)
df['Message']= [e['subject']+" "+e.get_payload().replace('\n',' ') for e in␣
 ↪df['Email']]
```

2.0.3 Add column for sentiment of message

[84]:
```
# label sentiment analyzer
af = Afinn()
```

[85]:
```
# analyze sentiment (this also takes a while to run)
df['Sentiment']= df['Message'].apply(af.score)
```

2.0.4 Add columns for the probabilities of the top 5 topics in each email

[281]:
```
# pre-processing

# create list of 150 most frequently-occuring words
words= (" ".join(df['Message'])).lower()
counter = Counter(words.split())
common = counter.most_common()
tk= list(pd.DataFrame(common[0:150])[0])

# remove frequently-occuring words that are stopwords (will be removed by␣
 ↪CountVectorizer)
non_stopword = [t for t in tk if t not in stopwords.words('english')]
#print(non_stopword) # review common words
to_remove =␣
 ↪['to','please','subject','pm','would','cc','re','may','from','said','get','know','one','nee
```

```
↪'sent','could','image','think','also','information','message','original','like','let','us',
↪'attached','meeting','day','make','two','email','first','corp','want','thanks','see','next'
            'use','contact','take']

# drop these common words
def drop_email_words(message):
    dropped = []
    for word in message.split():
        if word not in to_remove:
            dropped += [word]
    return " ".join(dropped)

# create new preprocessed column with the lowercased message and without
↪punctuation, numbers, new line, tab, and extra white spaces
df['Preprocess']=df['Message'].str.replace(r'[^\w\s]','')
df['Preprocess']=df['Preprocess'].str.replace('\d+', '')
df['Preprocess']=df['Preprocess'].str.replace("\n","")
df['Preprocess']=df['Preprocess'].str.replace("\t","")
df['Preprocess']=df['Preprocess'].str.replace(' +',' ')
df['Preprocess']=df['Preprocess'].str.lower()
df['Preprocess']=df['Preprocess'].apply(drop_email_words)
```

[282]:
```
# create document-term matrix for preprocessed messages (documents)
count_vect = CountVectorizer(ngram_range= (1,2), max_df=0.6, min_df=2,
↪stop_words='english')
doc_term_matrix = count_vect.fit_transform(df['Preprocess'])
```

[283]:
```
# find top 5 topic clusters
LDA = LatentDirichletAllocation(n_components=5, random_state=42) # 5 topics
LDA.fit(doc_term_matrix)
topic_values = LDA.transform(doc_term_matrix)
```

[284]:
```
# print the 15 words with highest probabilities for each of the 5 topics
for i, topic in enumerate(LDA.components_):
    print(f'Top 15 words for topic #{i+1}: '+", ".join([count_vect.
↪get_feature_names()[i] for i in topic.argsort()[-15:]]))
    print('')

# based on top 15 words, these topics are interpreted as:
# 1. Reporting
# 2. Revenue
# 3. Regulation
# 4. Management
# 5. Energy Market
```

```
Top 15 words for topic #1: good, im, work, final, schedule, scheduled, houston,
date, gas, deal, new, just, vince, time, enron

Top 15 words for topic #2: prices, business, year, billion, million, market,
electricity, california, gas, state, new, company, energy, power, enron

Top 15 words for topic #3: legal, received, list, trading, company, business,
john, questions, report, error, energy, credit, agreement, new, enron

Top 15 words for topic #4: updated, david, way, th, game, gas, houston, mike,
john, agreement, week, new, jeff, time, enron

Top 15 words for topic #5: ferc, width, business, million, price, tr, company,
td, gas, california, enron, new, market, energy, power
```

[285]:
```
# adds columns with the probabilities of each topic in each message
topics_df= pd.DataFrame(topic_values, columns= ["Topic1","Topic2",␣
↪"Topic3","Topic4", "Topic5"])
df= pd.concat([df,topics_df], axis=1)
```

2.0.5 Extract Recipient info

[287]:
```
# make new column for the list of recipients of each email and delete all email␣
↪entries where there are no recipients
df['Recipient_list']= [e['To'] for e in df['Email']]
df= df[df['Recipient_list'].notnull()]
```

[288]:
```
# clean recipient data and transform it into lists of string email addresses
df['Recipient_list']= [r.replace("\n\t", "").split(', ') for r in␣
↪df['Recipient_list']];
```

[289]:
```
# create a new Series of the recipient lists expanded so that, for each email,␣
↪each recipient has its own row but
# they all have the index of the original email
recipient= (df.Recipient_list.apply(pd.Series)
              .stack()
              .reset_index(level=1, drop=True)
              .to_frame('Recipient'))
```

[290]:
```
# merge the expanded recipient list onto the original DataFrame on the indexes␣
↪(this will copy the email
# to all recipient rows) and drop the old column of recipient lists
df= df.join(recipient)
df.drop('Recipient_list',axis=1, inplace=True)
```

2.0.6 Extract Sender info

[291]:
```
# make a new column for the senders of every email and delete all email entries␣
↪where there is no sender
df['Sender']= [e['from'] for e in df['Email']]
df= df[df['Sender'].notnull()]
```

2.0.7 Extract date info

[292]:
```
# make a new column for the date of the email
df['Date']= [datetime.strptime("/".join(e['date'].split()[1:4]), '%d/%b/%Y')␣
↪for e in df['Email']]
```

[293]:
```
# reset the index so that each row has its own unique index
df= df.reset_index().drop('index',axis=1)
```

[296]:
```
# drop column with full message (no longer valueable)
df= df.drop('Email', axis=1)
```

2.0.8 Add columns for sender and recipient job level groups (only available for a subset of emails)

Job Level groups (annotated in "Titles.xlsx")

Group A: management - CEO - COO - Director - General Counsel - Managing Director - President - Vice President

Group B: other - Administrative Assistant - Analyst - Government Relations Executive - In-House Lawyer - Manager - Senior Analyst - Senior Specialist - Specialist - Trader

[297]:
```
# read in titles and groups to df
url_titles = 'https://raw.github.com/mneedle/
↪ONA-Sentiment-Analysis-and-Topic-Clustering-in-Python/master/Titles.xlsx'
titles_groups = pd.read_excel(url_titles)

title_subset = titles_groups[["Email","Title"]]
titles = dict(zip(title_subset.Email, title_subset.Title))
df["SenderTitle"]= df["Sender"].map(titles)
df["RecipientTitle"]= df["Recipient"].map(titles)

group_subset= titles_groups[["Email","Group"]]
groups= dict(zip(group_subset.Email, group_subset.Group))
df["SenderGroup"]= df["Sender"].map(groups)
df["RecipientGroup"]= df["Recipient"].map(groups)
```

2.0.9 Add column for monthly stock price for month of email

[298]:
```
# read in stock prices to df
url_stock = 'https://raw.github.com/mneedle/
 ↪ONA-Sentiment-Analysis-and-Topic-Clustering-in-Python/master/Enron_Monthly.
 ↪xlsx'
stock = pd.read_excel(url_stock)
prices= stock[["Date","Last Price"]]
prices['Month_Year_String']= [d.strftime('%Y-%m') for d in prices['Date']]
monthly_prices= dict(zip(prices['Month_Year_String'], prices['Last Price']))
```

```
/Users/mneedle/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[299]:
```
# make column with years and months, map monthly stock prices, and delete␣
 ↪column
df['Date_String']= [d.strftime('%Y-%m') for d in df['Date']]
df['Monthly Stock Price']= df['Date_String'].map(monthly_prices)
df= df.drop('Date_String',axis=1)
```

3 Save full df

[300]:
```
df.to_excel('df_full.xlsx')
```

4 Reload full df

[301]:
```
df= pd.read_excel("df_full.xlsx").drop("Unnamed: 0", axis=1)
```

5 Create scatterplot of emails sent over time

[302]:
```
#plot email sent over time
fig = px.scatter(df, x="Date", y="Sender",
                 title="Sent Email Over Time",
                )
fig.update_yaxes(nticks=30)
fig.update_xaxes(nticks=50)
```

```
fig.show()
```

```
<Figure size 432x288 with 0 Axes>
```

5.1 Analye categorical sentiment and produce wordclouds for emails sent by each job level group

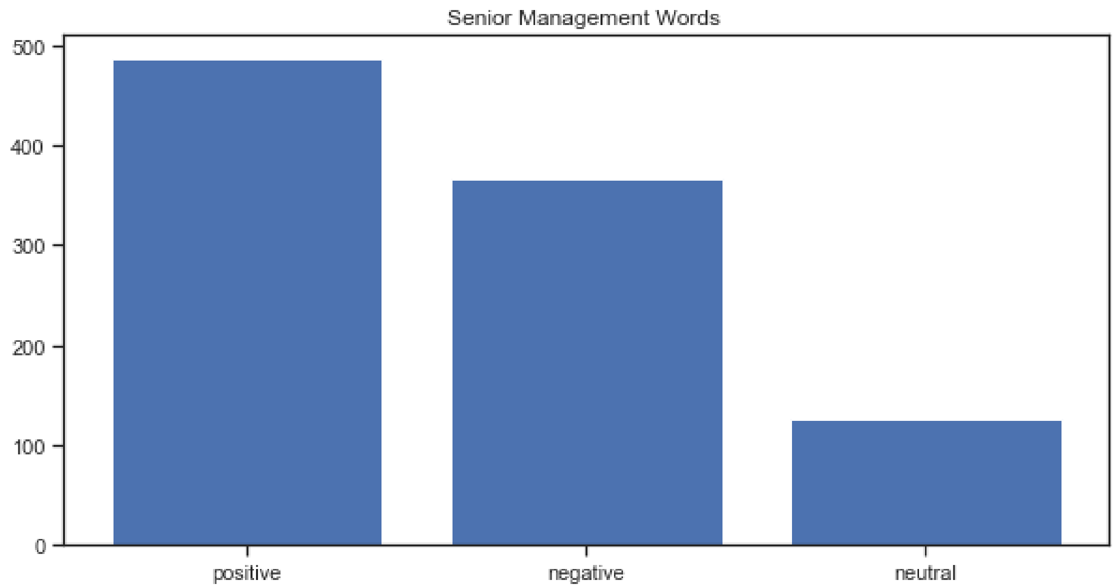
GroupA: Senior Management

[353]:
```
# get and clean all the words in emails sent by Senior Management
common_words_GroupA = (" ".join([str(message) for message in␣
↪df["Preprocess"][df['SenderGroup']=='GroupA']]))
common_GroupA_no_stop= [w for w in common_words_GroupA.split()]
```

[356]:
```
#categorizing the words into positive, negative and neutral
positive = []
negative = []
neutral = []
scores_of_words = []
for text in common_GroupA_no_stop:
    blob = TextBlob(text)
    scores_of_words.append(blob.sentiment.polarity)
    if(blob.sentiment.subjectivity>0.1):
        if(blob.sentiment.polarity==0.0):
            neutral.append(text)
        if(blob.sentiment.polarity>0.0):
            positive.append(text)
        if(blob.sentiment.polarity<0.0):
            negative.append(text)

# removing duplicates words from positive,negative and neutral words list.
uniqueWords_neutral = []
uniqueWords_positive = []
uniqueWords_negative = []

for i in positive:
      if not i in uniqueWords_positive:
            uniqueWords_positive.append(i);
for j in negative:
      if not j in uniqueWords_negative:
            uniqueWords_negative.append(j);
for k in neutral:
      if not k in uniqueWords_neutral:
            uniqueWords_neutral.append(k);
```

[357]:
```
# creating the bar graph
names = ["positive","negative","neutral"]
```

```
values =␣
↪[len(uniqueWords_positive),len(uniqueWords_negative),len(uniqueWords_neutral)]

fig, axs = plt.subplots( figsize=(10, 5), sharey=True)
axs.bar(names, values)
axs.set_title('Senior Management Words ')
plt.show()
```



[358]:

```
# create word cloud
all_words = uniqueWords_positive + uniqueWords_negative + uniqueWords_neutral
wordcloud = WordCloud(width = 1000, height = 500, background_color ='white').
↪generate(' '.join(all_words))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

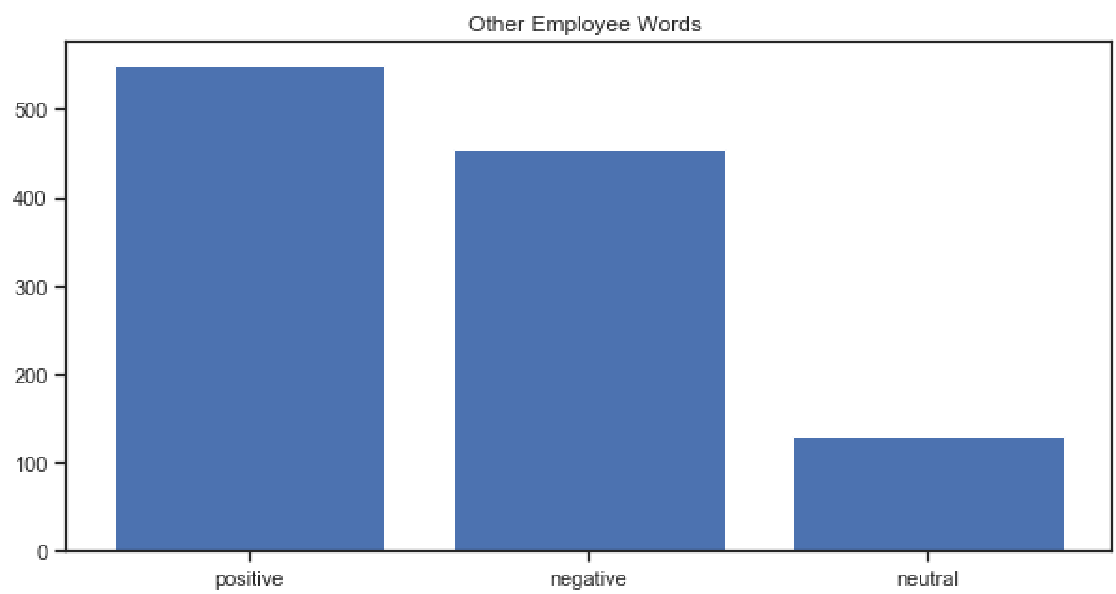
GroupB: Other Employees

[359]:
```
# get and clean all the words in emails sent by other employees
common_words_GroupB = (" ".join([str(message) for message in␣
↪df["Preprocess"][df['SenderGroup']=='GroupB']]))
common_GroupB_no_stop= [w for w in common_words_GroupB.split()]
```

[360]:
```
#categorizing the words into positive, negative and neutral
positive = []
negative = []
neutral = []
scores_of_words = []
for text in common_GroupB_no_stop:
    blob = TextBlob(text)
    scores_of_words.append(blob.sentiment.polarity)
    if(blob.sentiment.subjectivity>0.1):
        if(blob.sentiment.polarity==0.0):
            neutral.append(text)
        if(blob.sentiment.polarity>0.0):
            positive.append(text)
        if(blob.sentiment.polarity<0.0):
            negative.append(text)

# removing duplicates words from positive,negative and neutral words list.
uniqueWords_neutral = []
uniqueWords_positive = []
uniqueWords_negative = []

for i in positive:
      if not i in uniqueWords_positive:
```

```
            uniqueWords_positive.append(i);
for j in negative:
      if not j in uniqueWords_negative:
            uniqueWords_negative.append(j);
for k in neutral:
      if not k in uniqueWords_neutral:
            uniqueWords_neutral.append(k);
```

[361]:
```
# creating the bar graph
names = ["positive","negative","neutral"]
values =␣
 ↪[len(uniqueWords_positive),len(uniqueWords_negative),len(uniqueWords_neutral)]

fig, axs = plt.subplots( figsize=(10, 5), sharey=True)
axs.bar(names, values)
axs.set_title('Other Employee Words ')
plt.show()
```



[362]:
```
# create word cloud
all_words = uniqueWords_positive + uniqueWords_negative + uniqueWords_neutral
wordcloud = WordCloud(width = 1000, height = 500, background_color ='white').
 ↪generate(' '.join(all_words))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

```
plt.show()
```

6 Data analysis

6.1 Are job levels significantly different in terms of network centrality (degree and betweenness centrality)? 2 t-tests

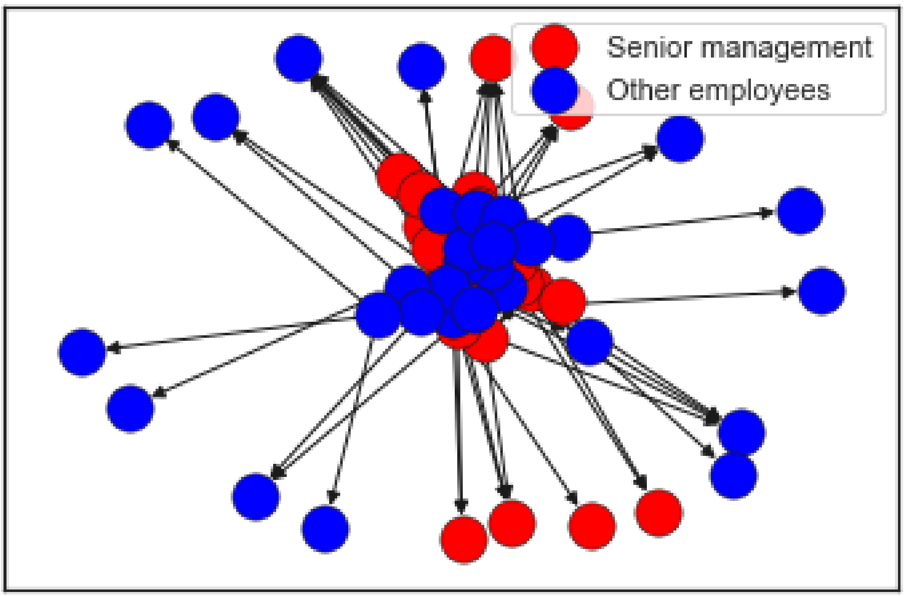
[363]:
```
# get subset of emails between people who both belong to job level groups
# then subest the data by sender-recipient email total
df_network = df[df['SenderGroup'].notnull() & df['RecipientGroup'].notnull()]
df_weighted= df_network.groupby(['Sender','Recipient']).count().reset_index()
```

[364]:
```
# create the network
G=nx.from_pandas_edgelist(df_weighted, "Sender", "Recipient", ['Message'], nx.
↪DiGraph())
```

[365]:
```
# create one full dictionary of the emails included in df_network
sender_groups= dict(zip(df_network['Sender'], df_network['SenderGroup']))
senders= list(sender_groups.keys())
full_groups= dict(zip(df_network['Recipient'], df_network['RecipientGroup']))
full= list(full_groups.keys())
overlap= [s for s in senders if s not in full]
for i in overlap:
    full_groups[i]= sender_groups[i]
```

[366]:
```
# divide full dictionary into groups based on job levels
GroupA= [e for e in full_groups if full_groups[e]=='GroupA']
GroupB= [e for e in full_groups if full_groups[e]=='GroupB']
```

[367]:
```
# plot network
G=nx.from_pandas_edgelist(df_weighted, "Sender", "Recipient", ['Message'], nx.
↪DiGraph())

rednodes = GroupA
bluenodes = GroupB
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos=pos,nodelist=rednodes, node_color='red',␣
↪label='Senior management', linewidths= 0.5, edgecolors='k')
nx.draw_networkx_nodes(G,pos=pos,nodelist=bluenodes, node_color='blue',␣
↪label='Other employees', linewidths= 0.5, edgecolors='k')
nx.draw_networkx_edges(G,pos=pos)
plt.legend(loc= 'upper right', numpoints = 1)
plt.show()
```



[368]:
```
# create df of only employees in groups
groups_df= pd.DataFrame.from_dict(full_groups, orient='index')
```

[370]:
```
# add columns for degree centrality
degCent= nx.degree_centrality(G)
groups_df['degCent']=groups_df.index.map(degCent)
```

[371]:
```
# add column for betweenness centrality
betCent = nx.betweenness_centrality(G)
groups_df['betCent']=groups_df.index.map(betCent)
```

[373]:
```
# t-test for degree centrality between groups
stats.ttest_ind(groups_df[groups_df[0]=='GroupA'].degCent,␣
↪groups_df[groups_df[0]=='GroupB'].degCent, equal_var = True)
```

```
# The p-value < 0.05, so the two groups are significantly different in degree␣
↪centrality
```

[373]:
```
Ttest_indResult(statistic=3.4774080802148677, pvalue=0.0009008249555561087)
```

[374]:
```
# t-test for betweenness centrality between groups
stats.ttest_ind(groups_df[groups_df[0]=='GroupA'].betCent,␣
↪groups_df[groups_df[0]=='GroupB'].betCent, equal_var = True)
# The p-value > 0.05, so the two groups are not significantly different in␣
↪betweenness centrality
```

[374]:
```
Ttest_indResult(statistic=1.6635817677597107, pvalue=0.10093774277045783)
```

6.2 Is sentiment significantly different by job levels? 1 t-test

[376]:
```
# make subset of master df for this analysis
sent_from_groups= df[df['SenderGroup'].notnull()]
```

[377]:
```
# t-test
stats.ttest_ind(sent_from_groups[sent_from_groups['SenderGroup']=='GroupA'].
↪Sentiment,sent_from_groups[sent_from_groups['SenderGroup']=='GroupB'].
↪Sentiment, equal_var = False)
# The p-value < 0.05, so the two groups are significantly different
```

[377]:
```
Ttest_indResult(statistic=66.35487048106465, pvalue=0.0)
```

6.3 Are stock price, topics, and sentiment associated? Correlations

[379]:
```
# get correlation coefficients between continuous measures
corr_df= df[['Monthly Stock␣
↪Price','Sentiment','Topic1','Topic2','Topic3','Topic4','Topic5']]
corr_df.corr()
```

[379]:
```
                     Monthly Stock Price  Sentiment    Topic1    Topic2  \
Monthly Stock Price             1.000000   0.014015  0.059655 -0.007544
Sentiment                       0.014015   1.000000  0.044759 -0.186374
Topic1                          0.059655   0.044759  1.000000 -0.318383
Topic2                         -0.007544  -0.186374 -0.318383  1.000000
Topic3                         -0.050042   0.089898 -0.269603 -0.179785
Topic4                          0.007887   0.048272 -0.398856 -0.303930
Topic5                         -0.030996   0.006776 -0.173800 -0.178514

                       Topic3    Topic4    Topic5
Monthly Stock Price -0.050042  0.007887 -0.030996
Sentiment            0.089898  0.048272  0.006776
Topic1              -0.269603 -0.398856 -0.173800
Topic2              -0.179785 -0.303930 -0.178514
Topic3               1.000000 -0.159571 -0.227253
Topic4              -0.159571  1.000000 -0.248342
```

```
Topic5              -0.227253 -0.248342  1.000000
```

[381]:
```python
# all correlations are significant
for i in corr_df.corr():
    for j in corr_df.corr():
        temp = df[df[i].notnull() & df[j].notnull()]
        h= stats.spearmanr(temp[i], temp[j])
        print(i,j,h)
```

```
Monthly Stock Price Monthly Stock Price
SpearmanrResult(correlation=0.9999999999999998, pvalue=0.0)
Monthly Stock Price Sentiment SpearmanrResult(correlation=0.016764182699370238,
pvalue=4.919690127903901e-66)
Monthly Stock Price Topic1 SpearmanrResult(correlation=0.0785651757568891,
pvalue=0.0)
Monthly Stock Price Topic2 SpearmanrResult(correlation=-0.017468254310081047,
pvalue=1.537721718849242e-71)
Monthly Stock Price Topic3 SpearmanrResult(correlation=-0.014899271095993715,
pvalue=1.5345623472256598e-52)
Monthly Stock Price Topic4 SpearmanrResult(correlation=0.05853158052587101,
pvalue=0.0)
Monthly Stock Price Topic5 SpearmanrResult(correlation=0.0067561207116674645,
pvalue=4.615160550357742e-12)
Sentiment Monthly Stock Price SpearmanrResult(correlation=0.016764182699370238,
pvalue=4.919690127903901e-66)
Sentiment Sentiment SpearmanrResult(correlation=1.0, pvalue=0.0)
Sentiment Topic1 SpearmanrResult(correlation=-0.060124932103978186, pvalue=0.0)
Sentiment Topic2 SpearmanrResult(correlation=-0.08862194940960513, pvalue=0.0)
Sentiment Topic3 SpearmanrResult(correlation=0.10063475503762762, pvalue=0.0)
Sentiment Topic4 SpearmanrResult(correlation=-0.018129517231679,
pvalue=6.041359068393551e-77)
Sentiment Topic5 SpearmanrResult(correlation=-0.13617724004743328, pvalue=0.0)
Topic1 Monthly Stock Price SpearmanrResult(correlation=0.0785651757568891,
pvalue=0.0)
Topic1 Sentiment SpearmanrResult(correlation=-0.06012493210397818, pvalue=0.0)
Topic1 Topic1 SpearmanrResult(correlation=1.0, pvalue=0.0)
Topic1 Topic2 SpearmanrResult(correlation=-0.2755166447371565, pvalue=0.0)
Topic1 Topic3 SpearmanrResult(correlation=-0.1213883702959755, pvalue=0.0)
Topic1 Topic4 SpearmanrResult(correlation=-0.19526447425284624, pvalue=0.0)
Topic1 Topic5 SpearmanrResult(correlation=-0.026111272205414107,
pvalue=1.5112121815118546e-157)
Topic2 Monthly Stock Price SpearmanrResult(correlation=-0.017468254310081047,
pvalue=1.537721718849242e-71)
Topic2 Sentiment SpearmanrResult(correlation=-0.08862194940960513, pvalue=0.0)
Topic2 Topic1 SpearmanrResult(correlation=-0.2755166447371565, pvalue=0.0)
Topic2 Topic2 SpearmanrResult(correlation=1.0, pvalue=0.0)
Topic2 Topic3 SpearmanrResult(correlation=-0.1350875604112256, pvalue=0.0)
Topic2 Topic4 SpearmanrResult(correlation=-0.25815313856261174, pvalue=0.0)
```

```
Topic2 Topic5 SpearmanrResult(correlation=-0.1273268767238512, pvalue=0.0)
Topic3 Monthly Stock Price SpearmanrResult(correlation=-0.014899271095993715,
pvalue=1.5345623472256598e-52)
Topic3 Sentiment SpearmanrResult(correlation=0.1006347550376276, pvalue=0.0)
Topic3 Topic1 SpearmanrResult(correlation=-0.1213883702959755, pvalue=0.0)
Topic3 Topic2 SpearmanrResult(correlation=-0.1350875604112256, pvalue=0.0)
Topic3 Topic3 SpearmanrResult(correlation=1.0, pvalue=0.0)
Topic3 Topic4 SpearmanrResult(correlation=0.11949167866755406, pvalue=0.0)
Topic3 Topic5 SpearmanrResult(correlation=-0.1849894553168171, pvalue=0.0)
Topic4 Monthly Stock Price SpearmanrResult(correlation=0.058531580525871006,
pvalue=0.0)
Topic4 Sentiment SpearmanrResult(correlation=-0.018129517231679004,
pvalue=6.041359068392861e-77)
Topic4 Topic1 SpearmanrResult(correlation=-0.19526447425284624, pvalue=0.0)
Topic4 Topic2 SpearmanrResult(correlation=-0.2581531385626117, pvalue=0.0)
Topic4 Topic3 SpearmanrResult(correlation=0.11949167866755406, pvalue=0.0)
Topic4 Topic4 SpearmanrResult(correlation=1.0, pvalue=0.0)
Topic4 Topic5 SpearmanrResult(correlation=-0.1181604083091389, pvalue=0.0)
Topic5 Monthly Stock Price SpearmanrResult(correlation=0.0067561207116674645,
pvalue=4.615160550357742e-12)
Topic5 Sentiment SpearmanrResult(correlation=-0.13617724004743328, pvalue=0.0)
Topic5 Topic1 SpearmanrResult(correlation=-0.026111272205414107,
pvalue=1.5112121815118546e-157)
Topic5 Topic2 SpearmanrResult(correlation=-0.1273268767238512, pvalue=0.0)
Topic5 Topic3 SpearmanrResult(correlation=-0.1849894553168171, pvalue=0.0)
Topic5 Topic4 SpearmanrResult(correlation=-0.11816040830913889, pvalue=0.0)
Topic5 Topic5 SpearmanrResult(correlation=1.0, pvalue=0.0)
```

6.3.1 Plot of sentiment, sentiment by group level, topic probabilities, and monthly stock price

[382]:
```
plot_df= df.copy()
```

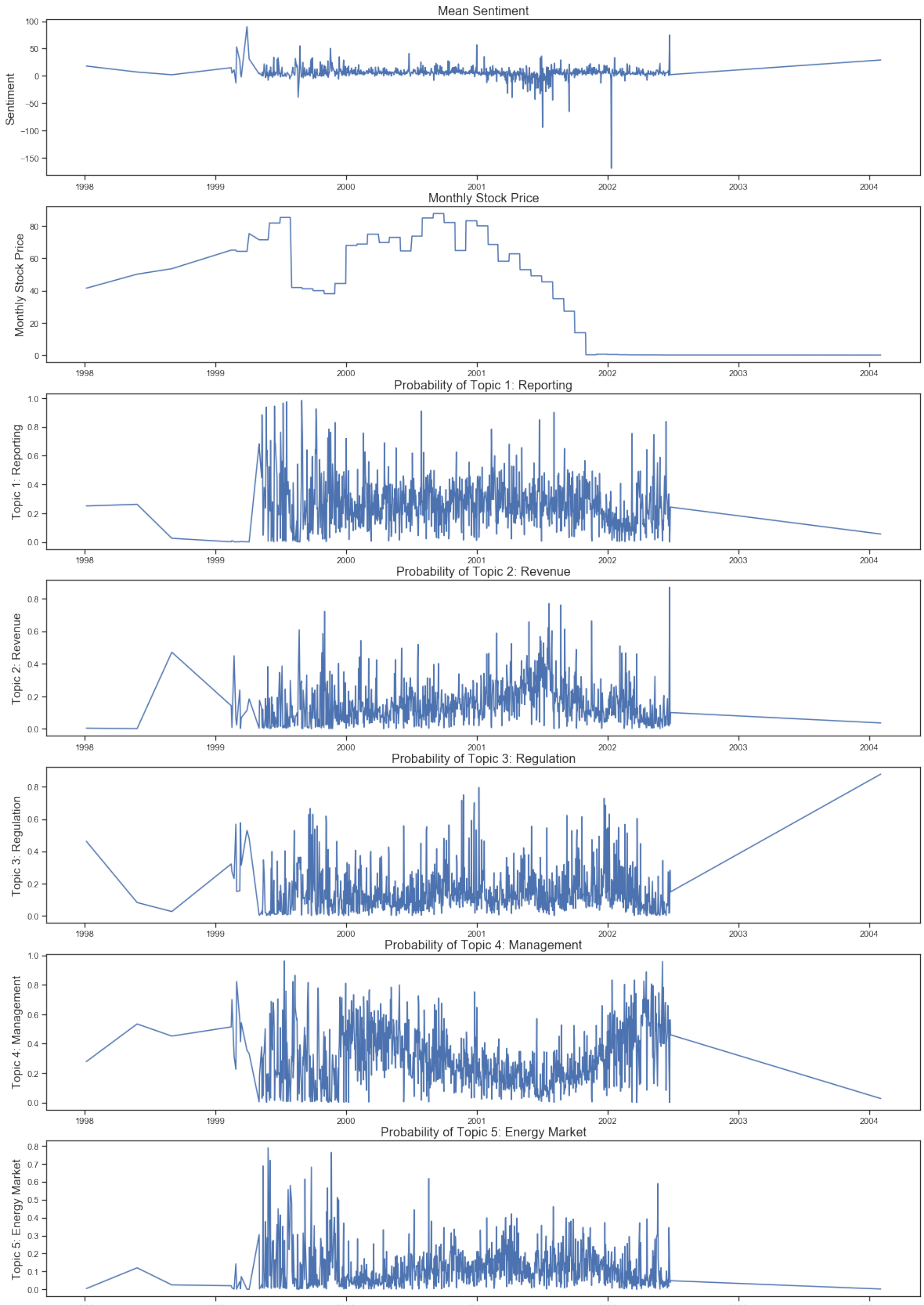
[383]:
```
plot_df['Date_byMonth']=df['Date']
```

[384]:
```
new_pivot= plot_df.pivot_table(index='Date_byMonth',␣
↪values=['Sentiment','Topic1','Topic2','Topic3','Topic4','Topic5','Monthly␣
↪Stock Price'], aggfunc='mean')\
    .dropna(how="any")
```

[387]:
```
fig, ax = plt.subplots(7,1, figsize= [20,30])

ax[0].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[0].plot(new_pivot.index,new_pivot.Sentiment)
ax[0].set_ylabel('Sentiment', fontsize= 15)
ax[0].set_title('Mean Sentiment', fontsize=16)

ax[1].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[1].plot(new_pivot.index,new_pivot['Monthly Stock Price'])
```

```
ax[1].set_ylabel('Monthly Stock Price', fontsize= 15)
ax[1].set_title('Monthly Stock Price', fontsize=16)

ax[2].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[2].plot(new_pivot.index,new_pivot['Topic1'])
ax[2].set_ylabel('Topic 1: Reporting', fontsize= 15)
ax[2].set_title('Probability of Topic 1: Reporting', fontsize=16)

ax[3].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[3].plot(new_pivot.index,new_pivot['Topic2'])
ax[3].set_ylabel('Topic 2: Revenue', fontsize= 15)
ax[3].set_title('Probability of Topic 2: Revenue', fontsize=16)

ax[4].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[4].plot(new_pivot.index,new_pivot['Topic3'])
ax[4].set_ylabel('Topic 3: Regulation', fontsize= 15)
ax[4].set_title('Probability of Topic 3: Regulation', fontsize=16)

ax[5].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[5].plot(new_pivot.index,new_pivot['Topic4'])
ax[5].set_ylabel('Topic 4: Management', fontsize= 15)
ax[5].set_title('Probability of Topic 4: Management', fontsize=16)

ax[6].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[6].plot(new_pivot.index,new_pivot['Topic5'])
ax[6].set_ylabel('Topic 5: Energy Market', fontsize= 15)
ax[6].set_title('Probability of Topic 5: Energy Market', fontsize=16)
```

[387]:
```
Text(0.5, 1.0, 'Probability of Topic 5: Energy Market')
```


Mean Sentiment
Sentiment
100
50
0
−50
−100
−150
1998
1999
2000
2001
2002
2003
2004
Monthly Stock Price
Monthly Stock Price
80
60
40
20
0
Probability of Topic 1: Reporting
Topic 1: Reporting
Probability of Topic 2: Revenue
Topic 2: Revenue
Probability of Topic 3: Regulation
Topic 3: Regulation
Probability of Topic 4: Management
Topic 4: Management
Probability of Topic 5: Energy Market
Topic 5: Energy Market