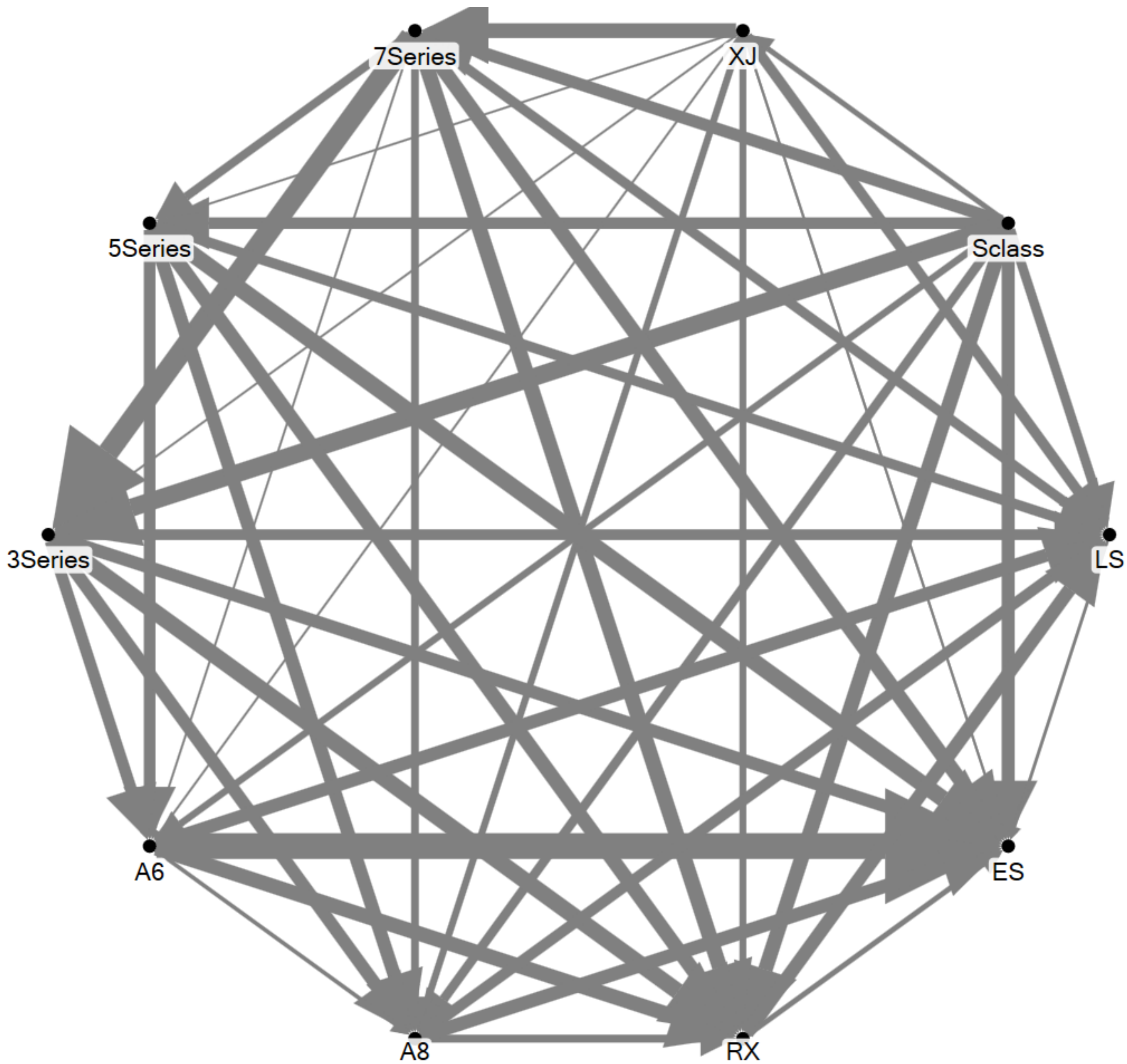
Text Analytics Assignment #3

Due: Monday 29th September by 11:59 p.m. on Canvas

Task A. The **sentiment scores** worksheet in the data file “Assignment 3 Sentiment Scores.csv” (on Canvas) provides sentiment scores (+5 to -5) of forum users on 10 car models. Each row represents a post (not shown) that can mention multiple models. Only positive and negative sentiments are noted.

From these sentiment scores, create a directed product comparison network (and use NodeXL or any other network analysis tool to display the network suitably). Use the principles laid out in the article “Product comparison networks” to answer this question.



Part A

Car 1	Car 2	Car2 Preference	Car1 Preference
LS	ES	1.17	2.00
RX	ES	1.67	2.00
A8	ES	3.40	3.00
A6	ES	6.00	1.67
3series	ES	3.25	6.00
5series	ES	4.00	5.00
7series	ES	3.60	2.80
XJ	ES	1.00	1.00
Sclass	ES	3.29	2.62
RX	LS	3.40	1.14
A8	LS	2.86	2.84
A6	LS	3.33	1.29
3series	LS	2.80	4.00
5series	LS	2.70	3.10
7series	LS	2.64	2.75
XJ	LS	2.75	2.89
Sclass	LS	2.55	2.68

A8	RX	2.20	1.00
A6	RX	3.67	0.00
3series	RX	4.00	4.00
5series	RX	3.60	0.00
7series	RX	3.75	0.00
XJ	RX	2.00	0.00
Sclass	RX	3.64	2.50
A6	A8	1.38	3.17
3series	A8	3.25	5.00
5series	A8	3.33	5.00
7series	A8	2.18	3.17
XJ	A8	2.00	2.14
Sclass	A8	2.50	2.06
3series	A6	3.00	5.00
5series	A6	3.00	5.00
7series	A6	0.00	5.00
XJ	A6	1.00	0.00
Sclass	A6	2.00	2.33
5series	3series	0.00	0.00
7series	3series	5.00	2.00

XJ	3series	0.00	3.00
Sclass	3series	4.33	3.33
7series	5series	2.33	1.00
XJ	5series	0.00	2.00
Sclass	5series	3.00	2.00
XJ	7series	3.67	3.17
Sclass	7series	3.11	2.44
Sclass	XJ	1.67	2.29

Task B. Calculate both unweighted and weighted PageRank scores for each car. What are the correlations between these metrics and sales figures shown below? What additional information do weighted PageRanks capture? Use a python script to calculate weighted PageRanks. Unweighted PageRanks can be calculated in NodeXL, or you can write a python script for that task as well.

Model	Approximate # sold in the U.S.A. (2012+2013)
Audi A6	20k
Audi A8	12k
BMW 3-series	220k
BMW 5-series	60k
BMW 7-series	14k
Jaguar XJ	6.6k
Lexus ES	135k
Lexus LS	30k
Lexus RX	120k
Mercedes S-class	25k

Unweighted pagerank:

To calculate unweighted pagerank, we can also use networkx package in python script.

	A	B	C	D	
1			Visual Properties		
2	Vertex 1	Vertex 2	Color	Width	St
3	LS	ES		1.1666666	
4	RX	ES		1.6666666	
5	A8	ES		3.4	
6	A6	ES		6	
7	3series	ES		3.25	
8	5series	ES		4	
9	7series	ES		3.6	
10	XJ	ES		1	
11	Sclass	ES		3.2857142	
12	RX	LS		3.4	
13	A8	LS		2.8571428	
14	A6	LS		3.3333333	
15	3series	LS		2.8	
16	5series	LS		2.7	
17	7series	LS		2.6428571	
18	XJ	LS		2.75	
19	Sclass	LS		2.5540540	
20	A8	RX		2.2	
21	A6	RX		3.6666666	
22	3series	RX		4	
23	5series	RX		3.6	
24	7series	RX		3.75	
25	XJ	RX		2	
26	Sclass	RX		3.6363636	
27	A6	A8		1.375	

First we need to create a graph by creating nodes with names of all types of cars. Then we use the results from task A to create edges. The results is shown above. After the graph is created, we use pagerank function to calculate pagerank.

Code:

```
import networkx as nx
import csv
#create graph
G = nx.Graph()
#create list of all car models
with open('C:\car.csv','rU')  as f:
        fr = csv.reader(f)
        for i in fr:
                G.add_edge(i[0],i[i])
pr = nx.pagerank(G,alpha=0.85)
print pr
```

Result:

```
{'3series': 0.09205607467959792,
 '5series': 0.09205607467959792,
 '7series': 0.10198598133010053,
 'A6': 0.10198598133010053,
 'A8': 0.10198598133010053,
 'ES': 0.10198598133010055,
 'LS': 0.10198598133010053,
 'RX': 0.10198598133010053,
 'Sclass': 0.10198598133010053,
 'XJ': 0.10198598133010053}
```

To find the correlation between sales and pagerank, we build correlation in SPSS using the above results. The p value of the correlation is 0.092, is between [0.01,0.1]. There's correlation between sales and unweighted page rank, but not very strong.
Result:

Correlations

```
[DataSet1]
```

Correlations

		Sales	UnweightedPageRank
Sales	Pearson Correlation	1	-.561
	Sig. (2-tailed)		.092
	N	10	10
UnweightedPageRank	Pearson Correlation	-.561	1
	Sig. (2-tailed)	.092	
	N	10	10

Weighted:
We use the same method to calculate weighted pagerank.

Code:

```
import networkx as nx
import csv

import os
os.getcwd()
'/Users/sumi'
os.chdir('/Users/sumi/documents/Study')
os.chdir('/Users/sumi/documents/Study/Fall/Text_analysis/Assignment/3')

G = nx.Graph()
G.add_node(1,Brand="ES")
```

```
G.add_node(2,Brand="LS")
G.add_node(3,Brand="RX")
G.add_node(4,Brand="A8")
G.add_node(5,Brand="A6")
G.add_node(6,Brand="3series")
G.add_node(7,Brand="5series")
G.add_node(8,Brand="7series")
G.add_node(9,Brand="XJ")
G.add_node(10,Brand="Sclass")
c=csv.reader(open("input.csv","rU"))
for i in c:
	G.add_edge(int(i[0]), int(i[1]), weight=float(i[2]))

pr= nx.pagerank(G,alpha=0.85)
pr
```

Audi A6	20k	0.108738431
Audi A8	12k	0.106264474
BMW 3-series	220k	0.123201655
BMW 5-series	60k	0.104234268
BMW 7-series	14k	0.102515484
Jaguar XJ	6.6k	0.07997261
Lexus ES	135k	0.101723286
Lexus LS	30k	0.090725962
Lexus RX	120k	0.093595426
Mercedes S-class	25k	0.089028405

Correlations

Correlations

		price	wpr
price	Pearson Correlation	1	.567
	Sig. (2-tailed)		.088
	N	10	10
wpr	Pearson Correlation	.567	1
	Sig. (2-tailed)	.088	
	N	10	10

We see that the Pearson Correlation for weighted page rank has improved to 0.567 from 0.561 for unweighted page rank, but this is not a very significant improvement.

Task C. The above sentiment scores above were obtained by manually reading each post. The file "Assignment 3 Edmunds Posts.xlsx" provide a bunch of actual messages (combine the worksheets). Your task is to automate the sentiment extraction from each post. As in tasks A and B, focus on the same 10 models (note that other models may also be mentioned, but that they should be ignored). Write one or more python script(s) to generate sentiment scores for the 10 models just as in the **sentiment scores** worksheet. This will be an unsupervised approach. One possibility (but not the only one) is to take the dictionary of SentiStrength (along with the default sentiment scores) and use it as inputs in your script(s). Your script should consider lemmatization (e.g., liking and liked must be treated as the same).

Generate sentiment scores with your script(s), find weighted PageRank of each of the 10 cars and correlate with the sales figures above. How does the correlation of this automated approach compare with that of manual scoring in task B?

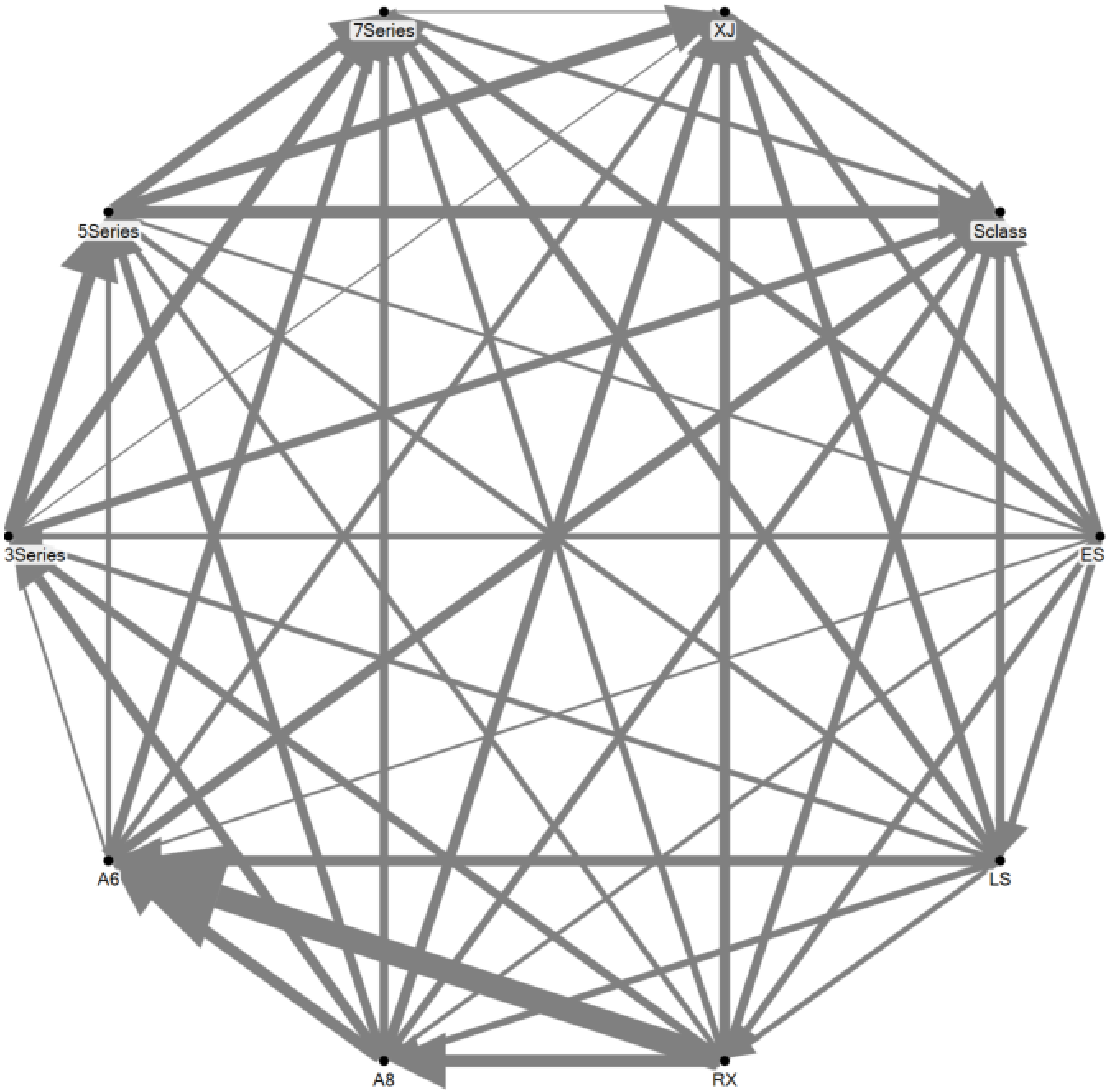
```
import nltk
import os
import csv
import pandas as pd
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import math
"""
   First of all, we will read Edmund Posts xlsx file (all tabs) and collate them.
   Then we will do few treatments like removing NaNs and duplicates.
"""

# Reading Data
path = os.getcwd()
files = os.listdir(path)
files
```

```
files_xlsx = [f for f in files if f[-4:] == 'xlsx']
files_xlsx

df = pd.DataFrame()

data1 = pd.read_excel(files_xlsx[0], 'Posts 1st set')
data2 = pd.read_excel(files_xlsx[0], 'Posts 2nd set')
data3 = pd.read_excel(files_xlsx[0], 'Posts 3rd set')
data4 = pd.read_excel(files_xlsx[0], 'Posts 4th set')
data5 = pd.read_excel(files_xlsx[0], 'Posts 5th set')
data6 = pd.read_excel(files_xlsx[0], 'Posts 6th set')

pieces = [df, data1, data2, data3, data4, data5, data6]
df_collated = pd.concat(pieces)

# Removing NaN's and duplicates from the collated file
df_new = df_collated[pd.notnull(df_collated['Posts'])]
df_final = df_new.drop_duplicates(cols='Posts')

"""
    Once the data is properly read, next step is to treat and remove punctuations, tokenize the sentence and remove stopwords.
    After that we will lemmatize the remaining tokens and consider only those words which have a length of greater than 1.
    Beacuse any word of length = 1 would mostly be gibberish. For instance, "" would have become "P" by now.
"""

# Removing punctuations, tokenizing and removing stopwords

reviewlines = pd.Series.tolist(df_final['Posts'])

reviewlinesfinal = []
for rl in reviewlines:
    rl = rl.encode('ascii','ignore')
    rl = rl.replace("'", " '")
    rl = rl.replace(".", " .")
    rl = rl.replace("-", " -")
    rl = rl.replace(",", " ,")
    rl = rl.replace("!", " !")
    rl = rl.replace("?", " ?")
    rl = rl.strip().lower().translate(None, "!#$%&()*+,-./:;<=>?@[\]^_'{|}~")
    rl = nltk.word_tokenize(rl)
    temprl = []
```

```
    stop = stopwords.words('english')
    for word in rl:
        if word not in stop:
            temprl.append(word)
    reviewlinesfinal.append(temprl)

# Lemmatizing
wnl = WordNetLemmatizer()
reviewlinesfinallemmatized = []
for rlf in reviewlinesfinal:
    templmt = []
    for word in rlf:
        if len(word) > 1:
            templmt.append(wnl.lemmatize(word))
        else:
            continue
    reviewlinesfinallemmatized.append(templmt)
"""
    Next step is to get the index for each occurence of all the 10 models in a particular review.
    Once we have those indexes, we will use the to create chunks by traversing right and left of that index.
    There are few considerations that we need to take care of. For example, chunk of a particular model should
    not consider other models in it because it might include sentiment for either of the 2 cars.
"""

# Getting indexes for each occurence of car model


model = ['lexuses','lexusls','rx','a8','a6','3series','5series','7series','xj','sclass']

indexlist = []

for index, rlfl in enumerate(reviewlinesfinallemmatized):
    ind = {}
    for i, tk in enumerate(rlfl):
        if tk in model:
            if tk not in ind.keys():
                ind[tk] = ind.get(tk, [i])
            else:
                ind[tk].append(i)
    indexlist.append(ind)
```

```
# Creating chunks
chunklist = []
nwords = 10

for index, il in enumerate(indexlist):
    chunkdictionary = {}
    for key, value in il.iteritems():
        clist = []
        for v in value:
            chunk = []
            flag1 = 0
            flag2 = 0
            for i in range(1, nwords + 1):
                try:
                    if reviewlinesfinallemmatized[index][(v-i)] != key and reviewlinesfinallemmatized[index][(v-i)] in
model:
                        flag1 = 1
                    if  flag1 == 0:
                        chunk.append(reviewlinesfinallemmatized[index][(v-i)])
                except:
                    pass
                try:
                    if reviewlinesfinallemmatized[index][(v+i)] != key and reviewlinesfinallemmatized[index][(v+i)] in
model:
                        flag2 = 1
                    if  flag2 == 0:
                        chunk.append(reviewlinesfinallemmatized[index][(v+i)])
                except:
                    pass
            clist.extend(chunk)
        chunkdictionary[key] = clist
    chunklist.append(chunkdictionary)

"""
    Then we will use sentistrength dictionary to get the sentiment scores for each car model.
    We will normalize the result as the length of the chunk might vary for each model in a particular review.
    We will store the final output in the form of list of dictionaries.
    Finally, we will convert this list of dictionaries to a csv file.
"""
```

```
# Getting sentiments

dictionary = pd.read_csv("C:/Users/Neerav Basant/Desktop/Fall Semester/Text Mining and Decision Analysis/GH 3/Sentistrength_Dictionary.csv")
y = dictionary.set_index('Word').to_dict()

sentimentscore = []


for index, il in enumerate(chunklist):
    postscore = {}
    count = {}
    postscorefinal = {}
    for key, value in il.iteritems():
            for ch in value:
                count[key] = count.get(key,0) + 1
                for k, v in y['Score'].iteritems():
                    if ch == k:
                        postscore[key] = postscore.get(key,0) + v
            try:
                postscorefinal[key] = float((postscore[key])/math.sqrt(count[key]))
            except:
                pass
    sentimentscore.append(postscorefinal)

# Get the final output in csv

f = open('Sentiment_Score_Final.csv', 'wb')
dict_writer = csv.DictWriter(f, model)
dict_writer.writer.writerow(model)
dict_writer.writerows(sentimentscore)
```



Weighted pagerank:

```
>>> G = nx.Graph()
>>> G.add_node(1,Brand="ES")
>>> G.add_node(2,Brand="LS")
>>> G.add_node(3,Brand="RX")
>>> G.add_node(4,Brand="A8")
>>> G.add_node(5,Brand="A6")
>>> G.add_node(6,Brand="3series")
>>> G.add_node(7,Brand="5series")
>>> G.add_node(8,Brand="7series")
>>> G.add_node(9,Brand="XJ")
>>> G.add_node(10,Brand="Sclass")
```

```
>>> c=csv.reader(open("senti.csv","rU"))
>>> for i in c:
...     G.add_edge(int(i[0]), int(i[1]))
>>> pr= nx.pagerank(G,alpha=0.85)
>>> pr
```

ES	0.099903818
LS	0.10369195
RX	0.101680606
A8	0.082694676
A6	0.104443273
3series	0.086427541
5series	0.112736214
7series	0.099887393
XJ	0.104895607
Sclass	0.103638924

Correlations

		Pagerank	price
Pagerank	Pearson Correlation	1	-.365
	Sig. (2-tailed)		.299
	N	10	10
price	Pearson Correlation	-.365	1
	Sig. (2-tailed)	.299	
	N	10	10

The resulte seems to be really confusing for us, the correlation is negative for unsupervised one while for supervised one the score is 0.56. And the absolute value decrease as well which means there's less correlation between unsupervised sentiment and price.