In [29]:

```
import os
import csv
import pandas as pd
import numpy as np
import sklearn
import string
import statsmodels.api as sm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.decomposition import NMF
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from nltk import tokenize
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

In [2]:

```
def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
    print()
```

In [3]:

```
def display_topics(model, feature_names, num_topics, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        if topic_idx < num_topics:
            print("{:11}".format("Topic %d:" %(topic_idx)), end='')
            print(", ".join(['{:04.3f}*'.format(topic[i])+feature_names[i] \
                             for i in topic.argsort()[:-no_top_words-1:-1]]))
```

Read in Data

In [4]:

```
# Read in Data
data = pd.read_csv('hash_house.csv')
data['userid'] = data['Unnamed: 0']
data.head()
```

Out[4]:

	Unnamed: 0	name	stars_y	text	userid
0	0	Hash House A Go Go	5	Firstly, this restaurant is in The Linq Hotel,...	0
1	1	Hash House A Go Go	4	This place had monsterous proportions OMG! One...	1
2	2	Hash House A Go Go	5	This place freaking rocks. Must go to when in ...	2
3	3	Hash House A Go Go	3	Visited HHAGG ago go for the first time on 5/5...	3
4	4	Hash House A Go Go	3	Big portions. Sharing is highly recommended. H...	4

Number of Topics

In [5]:

```
# Split reviews into individual sentences
df = pd.DataFrame(columns=['userid','sentence','stars'])
for i in range(0,len(data),1):
    sentences = tokenize.sent_tokenize(data.text[i])
    for j in sentences:
        df = df.append({'userid':data.userid[i],'sentence':j,'stars':data.stars_y[i]},ignore_index=Tru
e)
```

In [6]:
```
df.head()
```

Out[6]:

	userid	sentence	stars
0	0	Firstly, this restaurant is in The Linq Hotel,...	5
1	0	Expect a line.	5
2	0	Waited only about 15 minutes to be seated, tho...	5
3	0	Greeted by Tony our waiter who was really warm...	5
4	0	Ordered the Sage Fried Chicken and Waffles.	5

In [7]:
```
# Create Corpus for TFIDF
corpus = []
for i in df.sentence:
        corpus.append(i)
```

7 Topics

In [8]:
```
n_components = 7
n_top_words = 15

# TFIDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(corpus)

# NMF reduction
nmf = NMF(n_components=n_components).fit(tfidf)
W_pos = nmf.fit_transform(tfidf)

# Output Topics
print("\nTopics in NMF model (generalized Kullback-Leibler divergence):")
tfidf_feature_names = tfidf_vectorizer.get_feature_names()
print_top_words(nmf, tfidf_feature_names, n_top_words)
```

```
Topics in NMF model (generalized Kullback-Leibler divergence):
Topic #0: great service friendly excellent experience staff customer slow server fast atmosphere atte
ntive waiter quick bad
Topic #1: chicken waffles fried sage benedict ordered bacon got eggs delicious andy waffle potatoes c
rispy hash
Topic #2: huge portions large big share portion delicious people prices plate massive enormous hungry
meal tasty
Topic #3: good really pretty service overall just potatoes biscuits bloody thing mary taste coffee bi
scuit wasn
Topic #4: place vegas breakfast definitely hash love house try time come eat best recommend just las
Topic #5: food amazing delicious man vs awesome just came lot price excellent took quality tasty larg
e
Topic #6: wait worth long time minutes hour seated 30 table minute 45 20 come definitely 10
```

- Topic #0: Service
- Topic #1: Food
- Topic #2: "Worth it"
- Topic #3: Food / Service
- Topic #4:
- Topic #5: Food
- Topic #6: Wait

Label Sentences

In [9]:

```python
# Append Topic with highest score
array = []
# For all NMF array
for i in range(0,len(W_pos),1):
    # Create dictionary with Topics and its NMF scores for each sentence
    topic_dict = {}
    # Drop sentences that have length less than 10 by setting topic to -1
    if len(corpus[i])>=10:
        for ind, w in enumerate(W_pos[i]):
            topic_dict[ind] = w
        # Classify sentence to the topic with highest score
        array.append(max(topic_dict, key=topic_dict.get))
    else:
        array.append(-1)
# Create new column in df for topic
df['Topic'] = array
```

In [10]:

```python
df.head()
```

Out[10]:

	userid	sentence	stars	Topic
0	0	Firstly, this restaurant is in The Linq Hotel,...	5	4
1	0	Expect a line.	5	6
2	0	Waited only about 15 minutes to be seated, tho...	5	6
3	0	Greeted by Tony our waiter who was really warm...	5	3
4	0	Ordered the Sage Fried Chicken and Waffles.	5	1

Vader Sentiment Analysis

In [11]:

```python
# Initialize Sentiment Intensity Analyzer
analyser = SentimentIntensityAnalyzer()
```

In [12]:

```python
# Append Sentiment Intensity Scores for each sentence
array = []
for i in df.sentence:
    # Generate Sentiment Intensity Scores and store in array
    score = analyser.polarity_scores(i)
    array.append(score['compound'])
# Create new column in df for sentiment intensity score
df['sentiment'] = array
```

In [13]:

```python
df.head()
```

Out[13]:

	userid	sentence	stars	Topic	sentiment
0	0	Firstly, this restaurant is in The Linq Hotel,...	5	4	0.0000
1	0	Expect a line.	5	6	0.0000
2	0	Waited only about 15 minutes to be seated, tho...	5	6	0.0000
3	0	Greeted by Tony our waiter who was really warm...	5	3	0.8669
4	0	Ordered the Sage Fried Chicken and Waffles.	5	1	0.0000

In [14]:

```python
# Initialize Final df of intensity scores
df_scores = pd.DataFrame(columns=['userid','0','1','2','3','4','5','6','stars'])
# For every user aggregate the sentiment scores by topic
for i in range(0,len(data),1):
    # Create df of scores from same user
    temp_df = df[df.userid==i].reset_index(drop=True)
    # For every topic
    topic_score = []
    for j in range(0,7,1):
        score = 0
        count = 0
        for k in range(0,len(temp_df),1):
            # If topic equal to current topic
            if temp_df.Topic[k] == j:
                # Add sentiment score
                score = score + temp_df.sentiment[k]
                # Increase count
                count = count + 1
        # If count = 0 then no score for topic
        if count==0:
            topic_score.append(0)
        # Else append average score for topic
        else:
            topic_score.append(score/count)
    # Insert UserId and Star Rating
    topic_score.insert(0,temp_df.userid[0])
    topic_score.insert(len(topic_score),temp_df.stars[0])
    # Transform and Append into main df
    temp = pd.DataFrame(pd.Series(topic_score))
    temp = temp.transpose()
    temp.columns = df_scores.columns
    df_scores = df_scores.append(temp,ignore_index=True)
```

In [15]:

```python
df_scores.head()
```

Out[15]:

	userid	0	1	2	3	4	5	6	stars
0	0.0	0.7436	0.193525	0.0000	0.13765	0.04970	0.113150	0.11880	5.0
1	1.0	0.0000	0.000000	0.0000	0.00000	0.00000	0.000000	-0.20015	4.0
2	2.0	0.0000	0.000000	0.0000	0.00000	-0.21075	0.000000	0.00000	5.0
3	3.0	0.0000	-0.011300	0.3182	0.43720	0.31845	0.286075	0.00000	3.0
4	4.0	0.0000	0.000000	0.0000	-0.41580	0.59840	0.000000	0.00000	3.0

In [16]:

```python
df_scores.mean()
```

Out[16]:

```
userid    2923.000000
0            0.145195
1            0.123677
2            0.165368
3            0.173203
4            0.176851
5            0.190683
6            0.086749
stars        3.919446
dtype: float64
```

Linear Regression

In [53]:
```
# Split into predictors and target
X = df_scores.drop(['userid','stars'],axis=1)
y = df_scores.stars
# Split Train vs Test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,stratify=y,random_state=1)
# Split Test set into validation & test set
X_test2, X_val, y_test2, y_val = train_test_split(X_test,y_test,test_size=0.5,stratify=y_test,random_s
tate=1)
```

Simple Linear Regression

In [95]:
```
model = sm.OLS(y_train,sm.add_constant(X_train)).fit()
print(model.params)
print()
print('Mean Squared Error: ',mean_squared_error(y_val,model.predict(sm.add_constant(X_val))))
print('AIC: ',model.aic)
```

```
const    3.174016
0        0.680940
1        0.503598
2        0.530514
3        0.282414
4        1.121439
5        0.966572
6        0.692612
dtype: float64

Mean Squared Error:  1.0871000026560464
AIC:  13435.641863966863
```

$$yhat = 3.17 + 0.68 * Service + 0.50 * Food1 + 0.53 * Worth + 0.28 * Food/Service + 1.12 * Topic4 + 0.97 * Food2 + 0.69$$

Removed Intercept and Non-Topics

In [105]:
```
# Split into predictors and target
X = df_scores.drop(['userid','stars','4'],axis=1)
y = df_scores.stars
# Split Train vs Test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,stratify=y,random_state=1)
# Split Test set into validation & test set
X_test2, X_val, y_test2, y_val = train_test_split(X_test,y_test,test_size=0.5,stratify=y_test,random_s
tate=1)

model = sm.OLS(y_train,X_train).fit()
print(model.params)
print()
print('Mean Squared Error: ',mean_squared_error(y_val,model.predict(X_val)))
print('AIC: ',model.aic)
```

```
0    2.413325
1    2.605733
2    3.110401
3    2.821018
5    3.400820
6    2.471009
dtype: float64

Mean Squared Error:  6.117380574771179
AIC:  21352.176868844
```

$$yhat = 2.41 * Service + 2.61 * Food1 + 3.11 * Worth + 2.82 * Food/Service + 3.40 * Food2 + 2.47 * Wait$$