Name: Krit Gupta
UIN: 927001565

WEEK 12 ASSIGNMENT

PYTHON:

```
import pandas as pd
import numpy as np
import string
# from Class_replace_impute_encode import ReplaceImputeEncode
from nltk import pos_tag
from time import time
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from PIL import Image
import random

# Increase column width to let pandy read large text columns
pd.set_option('max_colwidth', 32000)
# Read N=13,575 Hotel Reviews
df = pd.read_excel("hotels.xlsx")


sentiment_file = pd.read_excel("afinn_sentiment_words.xlsx")


def my_analyzer(s):
    # Synonym List
    syns = {"n't": 'not', 'to30': 'to 30', \
            'wont': 'would not', 'cant': 'can not', 'cannot': 'can not', \
            'couldnt': 'could not', 'shouldnt': 'should not', \
            'wouldnt': 'would not', 'straightforward': 'straight forward'}

    # Preprocess String s
    s = s.lower()
    # Replace special characters with spaces
    s = s.replace('-', ' ')
    s = s.replace('_', ' ')
    s = s.replace(',', '. ')
    # Replace not contraction with not
    s = s.replace("'nt", " not")
    s = s.replace("n't", " not")
    # Tokenize
    tokens = word_tokenize(s)
    # tokens = [word.replace(',','') for word in tokens ]
    tokens = [word for word in tokens if ('*' not in word) and \
              ("'" != word) and ("``" != word) and \
              (word != 'description') and (word != 'dtype') \
              and (word != 'object') and (word != "'s")]

    # Map synonyms
    for i in range(len(tokens)):
        if tokens[i] in syns:
            tokens[i] = syns[tokens[i]]

    # Remove stop words
    punctuation = list(string.punctuation) + ['..', '...']
```

Name: Krit Gupta
UIN: 927001565

```
    pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
    others = ["'d", "co", "ed", "put", "say", "get", "can", "become", \
              "los", "sta", "la", "use", "iii", "else"]
    stop = stopwords.words('english') + punctuation + pronouns + others
    filtered_terms = [word for word in tokens if (word not in stop) and \
                      (len(word) > 1) and (not word.replace('.', '', 1).isdigit())
\
                      and (not word.replace("'", '', 2).isdigit())]

    # Lemmatization & Stemming - Stemming with WordNet POS
    # Since lemmatization requires POS need to set POS
    tag_wrds = pos_tag(filtered_terms, lang='eng')
    # Stemming with for terms without WordNet POS
    stemmer = SnowballStemmer("english")
    wn_tags = {'N': wn.NOUN, 'J': wn.ADJ, 'V': wn.VERB, 'R': wn.ADV}
    wnl = WordNetLemmatizer()
    stemmed_tokens = []
    for tagged_token in tag_wtag:
        term = tagged_token[0]
        pos = tagged_token[1]
        pos = pos[0]
        try:
            pos = wn_tags[pos]
            stemmed_tokens.append(wnl.lemmatize(term, pos=pos))
        except:
            stemmed_tokens.append(stemmer.stem(term))
    return stemmed_tokens


def display_topics(lda, terms, n_terms=15):
    for topic_idx, topic in enumerate(lda):
        if topic_idx > 8:
            break
        message = "Topic #%d: " % (topic_idx + 1)
        print(message)
        abs_topic = abs(topic)
        topic_terms_sorted = \
            [[terms[i], topic[i]] \
             for i in abs_topic.argsort()[:-n_terms - 1:-1]]
        k = 5
        n = int(n_terms / k)
        m = n_terms - k * n
        for j in range(n):
            l = k * j
            message = ''
            for i in range(k):
                if topic_terms_sorted[i + l][1] > 0:
                    word = "+" + topic_terms_sorted[i + l][0]
                else:
                    word = "-" + topic_terms_sorted[i + l][0]
                message += '{:<15s}'.format(word)
            print(message)
        if m > 0:
            l = k * n
            message = ''
            for i in range(m):
                if topic_terms_sorted[i + l][1] > 0:
                    word = "+" + topic_terms_sorted[i + l][0]
                else:
                    word = "-" + topic_terms_sorted[i + l][0]
                message += '{:<15s}'.format(word)
            print(message)
        print("")
    return
```

Name: Krit Gupta
UIN: 927001565

```
attribute_map = {
    'doc': [3, (0, 1e+12), [0, 0]],
    'hotel': [2, ('Bally's', 'Bellagio', 'Circus Circus', 'Encore', 'Excalibur'),
[0, 0]],
    'Review': [3, (''), [0, 0]],
    'topic': [2, (0, 1, 2, 3, 4, 5, 6), [0, 0]],
    'T1': [0, (-1e+8, 1e+8), [0, 0]],
    'T2': [0, (-1e+8, 1e+8), [0, 0]],
    'T3': [0, (-1e+8, 1e+8), [0, 0]],
    'T4': [0, (-1e+8, 1e+8), [0, 0]],
    'T5': [0, (-1e+8, 1e+8), [0, 0]],
    'T6': [0, (-1e+8, 1e+8), [0, 0]],
    'T7': [0, (-1e+8, 1e+8), [0, 0]]}
# Setup program constants
n_comments = len(df['Review'])  # Number of hotel reviews
m_features = None  # Number of SVD Vectors
stop_words = 'english'  # Stop Word Dictionary
comments = df['Review']  # place all text reviews in reviews
num_topics = 7  # number of topic clusters to extract
num_iterations = 10  # maximum number of itertions
max_df = 0.7  # learning offset for LDAmax proportion of docs/reviews allowed for a
term
# Create Word Frequency by Review Matrix using Custom Analyzer
cv = CountVectorizer(max_df=0.7, min_df=4, max_features=m_features, \
                     analyzer=my_analyzer, ngram_range=(1, 2))
tf = cv.fit_transform(comments)
terms = cv.get_feature_names()
term_sums = tf.sum(axis=0)
term_counts = []
for i in range(len(terms)):
    term_counts.append([terms[i], term_sums[0, i]])


def sortSecond(e):
    return e[1]


term_counts.sort(key=sortSecond, reverse=True)
print("\nTerms with Highest Frequency:")
for i in range(10):
    print('{:<15s}{:>5d}'.format(term_counts[i][0], term_counts[i][1]))
print("")
# Modify tf, term frequencies, to TF/IDF matrix from the data
print("Conducting Term/Frequency Matrix using TF-IDF")
tfidf_vect = TfidfTransformer(norm=None, use_idf=True)  # set norm=None
tf = tfidf_vect.fit_transform(tf)

term_idf_sums = tf.sum(axis=0)
term_idf_scores = []
for i in range(len(terms)):
    term_idf_scores.append([terms[i], term_idf_sums[0, i]])
print("The Term/Frequency matrix has", tf.shape[0], " rows, and", \
      tf.shape[1], " columns.")
print("The Term list has", len(terms), " terms.")
term_idf_scores.sort(key=sortSecond, reverse=True)
print("\nTerms with Highest TF-IDF Scores:")
for i in range(10):
    print('{:<15s}{:>8.2f}'.format(term_idf_scores[i][0], \
                                   term_idf_scores[i][1]))
num_reviews = len(df['Review'])  # Number of wine reviews
m_features = 100  # Number of SVD Vectors
stop_words = 'english'  # Stop Word Dictionary
5
ngram = (1, 2)  # n-gram POS modeling
reviews = df['Review']  # place all text reviews in reviews
num_topics = 9  # number of topic clusters to extract
```

Name: Krit Gupta
UIN: 927001565

```
num_iterations = 10  # maximum number of itertions for LDA
learning_offset = 10.  # learning offset for LDA
learning_method = 'online'  # learning method for LDA
tfidf = True
lda = LatentDirichletAllocation(n_components=num_topics,
num_iterations=num_iterations, \
                                learning_method=learning_method, \
                                learning_offset=learning_offset, \
                                random_state=12345)
U = lda.fit_transform(tf)
print('{:.<22s}{:>6d}'.format("Number of Reviews", len(reviews)))
print('{:.<22s}{:>6d}'.format("Number of Terms", len(terms)))
print("\nTopics Identified using LDA with TF_IDF")
display_topics(lda.components_, terms, n_terms=15)
# Store topic group for each doc in topics[]
topics = [0] * n_comments
topic_counts = [0] * (num_topics + 1)
for i in range(n_comments):
    max = abs(U[i][0])
    topics[i] = 0
    for j in range(num_topics):
        x = abs(U[i][j])
        if x > max:
            max = x
            topics[i] = j
    topic_counts[topics[i]] += 1

print('{:<6s}{:>8s}{:>8s}'.format("TOPIC", "COMMENTS", "PERCENT"))
for i in range(num_topics):
    print('{:>3d}{:>10d}{:>8.1%}'.format((i + 1), topic_counts[i], \
                                         topic_counts[i] / n_comments))

# Create comment_scores[] and assign the topic groups
comment_scores = []
for i in range(n_comments):
    u = [0] * (num_topics + 1)
    u[0] = topics[i]
    for j in range(num_topics):
        u[j + 1] = U[i][j]
    comment_scores.append(u)

# Augment Dataframe with topic group information
cols = ["topic"]
for i in range(num_topics):
    s = "T" + str(i + 1)
    cols.append(s)
df_topics = pd.DataFrame.from_records(comment_scores, columns=cols)
df = df.join(df_topics)
# Setup Sentiment dictionary
sentiment_dic = {}
for i in range(len(sentiment_file)):
    sentiment_dic[sentiment_file.iloc[i][0]] = sentiment_file.iloc[i][1]


def my_preprocessor(s):
    # Preprocess String s
    s = s.lower()
    # Replace special characters with spaces
    s = s.replace('-', ' ')
    s = s.replace('_', ' ')
    s = s.replace(',', '. ')
    # Replace not contraction with not
    s = s.replace("'nt", " not")
    s = s.replace("n't", " not")
    return s
```

Name: Krit Gupta
UIN: 927001565

```
cv = CountVectorizer(max_df=0.7, min_df=4, max_features=None, \
                     preprocessor=my_preprocessor, ngram_range=(1, 2))
tf = cv.fit_transform(df['Review'])
terms = cv.get_feature_names()
num_reviews = tf.shape[0]
n_terms = tf.shape[1]
print('{:.<22s}{:>6d}'.format("Number of Reviews", num_reviews))
print('{:.<22s}{:>6d}'.format("Number of Terms", n_terms))

min_sentiment = +5
max_sentiment = -5
avg_sentiment, min, max = 0, 0, 0
min_list, max_list = [], []
sentiment_score = [0] * num_reviews
for i in range(num_reviews):
    # Iterate over the terms with nonzero scores
    n_sentiment_file = 0
    term_list = tf[i].nonzero()[1]
    if len(term_list) > 0:
        for t in np.nditer(term_list):
            score = sentiment_dic.get(terms[t])
            if score != None:
                sentiment_score[i] += score * tf[i, t]
                n_sentiment_file += tf[i, t]
    if n_sentiment_file > 0:
        sentiment_score[i] = sentiment_score[i] / n_sentiment_file
    if sentiment_score[i] == max_sentiment and n_sentiment_file > 3:
        max_list.append(i)
    if sentiment_score[i] > max_sentiment and n_sentiment_file > 3:
        max_sentiment = sentiment_score[i]
        max = i
        max_list = [i]
    if sentiment_score[i] == min_sentiment and n_sentiment_file > 3:
        min_list.append(i)
    if sentiment_score[i] < min_sentiment and n_sentiment_file > 3:
        min_sentiment = sentiment_score[i]
        min = i
        min_list = [i]
    avg_sentiment += sentiment_score[i]
avg_sentiment = avg_sentiment / num_reviews
print("\nCorpus Average Sentiment: ", avg_sentiment)
print("\nMost Negative Reviews with 4 or more Sentiment Words:")
for i in range(len(min_list)):
    print("{:<s}{:<d}{:<s}{:<5.2f}".format(" Review ", min_list[i], \
                                           " Sentiment is ", min_sentiment))
print("\nMost Positive Reviews with 4 or more Sentiment Words:")
for i in range(len(max_list)):
    print("{:<s}{:<d}{:<s}{:<5.2f}".format(" Review ", max_list[i], \
                                           " Sentiment is ", max_sentiment))

stopw = set(STOPWORDS)
stopw.add("stay")
stopw.add("hotel")
stopw.add("room")
stopw.add("consider")


def shades_of_grey(word, font_size, position, orientation, random_state=None,
**kwargs):
    return "hsl(0, 0%%, %d%%)" % random.randint(60, 1000)


# circle_mask = np.array(Image.open("CircleMask.png"))
cloud = WordCloud(background_color="maroon", max_words=200, stopwords=stopw, \
                  random_state=341)
```

Name: Krit Gupta
UIN: 927001565

```
cloud.generate(df['Review'].iloc[min_list[0]])
print("\nMost Negative Review Sentiment=", min_sentiment)
plt.imshow(cloud.recolor(color_func=shades_of_grey, random_state=3), \
           interpolation="bilinear")
plt.axis("off")
plt.figure()
s = ""
i = min_list[0]
term_list = tf[i].nonzero()[1]
if len(term_list) > 0:
    for t in np.nditer(term_list):
        score = sentiment_dic.get(terms[t])
        if score != None:
            s += terms[t] + " "
cloud.generate(s)
plt.imshow(cloud.recolor(color_func=shades_of_grey, random_state=3), \
           interpolation="bilinear")
plt.axis("off")
plt.figure()
plt.show()

cloud.generate(df['Review'].iloc[max_list[0]])
print("\nMost Positive Review Sentiment=", max_sentiment)
plt.imshow(cloud.recolor(color_func=shades_of_grey, random_state=3), \
           interpolation="bilinear")
plt.axis("off")
plt.figure()
s = ""
i = max_list[0]
term_list = tf[i].nonzero()[1]
if len(term_list) > 0:
    for t in np.nditer(term_list):
        score = sentiment_dic.get(terms[t])
        if score != None:
            s += terms[t] + " "
cloud.generate(s)
plt.imshow(cloud.recolor(color_func=shades_of_grey, random_state=3),
interpolation="bilinear")
plt.axis("off")
plt.figure()
plt.show()

corpus_sentiment = {}
n_sentiment_file = 0
for i in range(num_reviews):
    # Iterate over the terms with nonzero scores
    term_list = tf[i].nonzero()[1]
    if len(term_list) > 0:
        for t in np.nditer(term_list):
            score = sentiment_dic.get(terms[t])
            if score != None:
                n_sentiment_file += tf[i, t]
                current_count = corpus_sentiment.get(terms[t])
                if current_count == None:
                    corpus_sentiment[terms[t]] = tf[i, t]
                else:
                    corpus_sentiment[terms[t]] += tf[i, t]
print("The Corpus contains a total of ", len(corpus_sentiment), " unique sentiment
words")
print("The total number of sentiment words in the Corpus is", n_sentiment_file)
cloud = WordCloud(background_color="maroon", max_words=200, \
                  random_state=341)
cloud.generate_from_frequencies(corpus_sentiment)
plt.imshow(cloud.recolor(color_func=shades_of_grey, random_state=3),
interpolation="bilinear")
plt.axis("off")
```

Name: Krit Gupta
UIN: 927001565

```
plt.figure()
plt.show()
```

OUTPUT:

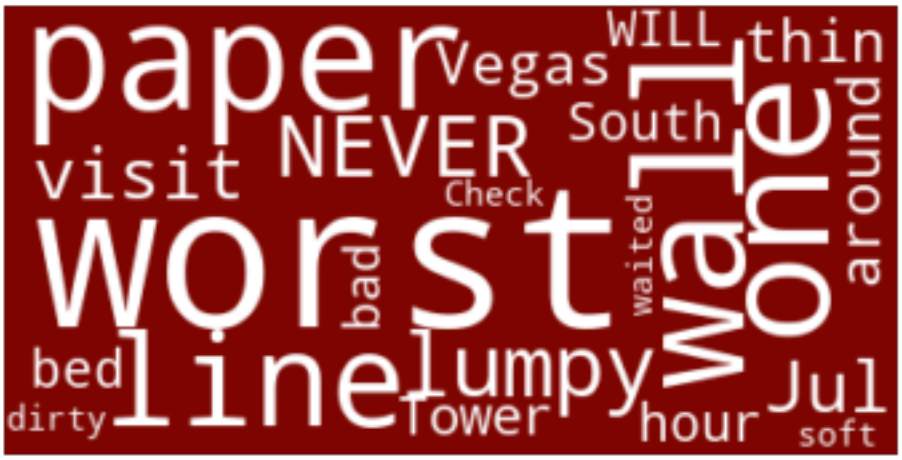
```
Terms with Highest Frequency:
great           1603
go              1494
good            1432
would           1406
strip           1243
time            1202
check           1191
night           1179
one             1162
quot            1162

Conducting Term/Frequency Matrix using TF-IDF
('The Term/Frequency matrix has', 1671, ' rows, and', 3152, ' columns.'
)
('The Term list has', 3152, ' terms.')

Terms with Highest TF-IDF Scores:
quot            2962.70
great           2721.80
go              2602.81
good            2564.84
would           2458.36
check           2281.95
strip           2247.83
pool            2234.49
time            2223.09
one             2212.57
Number of Reviews.....  1671
Number of Terms.......  3152

Topics Identified using LDA with TF_IDF
Topic #1:
+circus         +kid            +quot           +go             +bad
+get            +one            +like           +family         +place
+take           +excalibur      +people         +nice           +pay

Topic #2:
+quot           +luggage        +guard          +husband        +daughter
+help           +security       +bag            +order          +make
+cart           +wheelchair     +wait           +shuttle        +dinner

Topic #3:
+smoke          +non            +call           +check          +would
+smoking        +day            +tell           +circus         +charge
+book           +even           +per            +go             +quot

Topic #4:
+bellagio       +great          +pool           +view           +best
+love           +strip          +good           +fountain       +night
+excellent      +vega           +location       +would          +beautiful

Topic #5:
+pool           +hot            +york           +new            +day
```

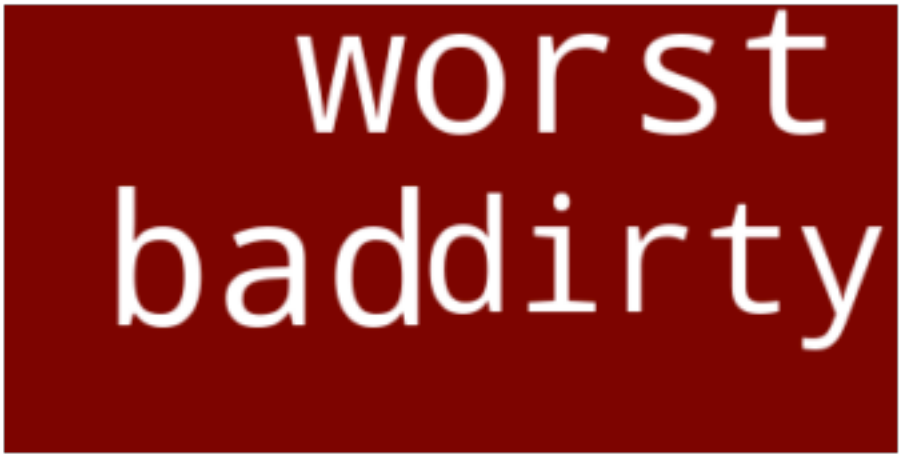
Name: Krit Gupta
UIN: 927001565

```
+water          +cold           +great          +inside         +fresh
+couple         +need           +look           +place          +food

Topic #6:
+excalibur      +value          +amp            +good           +vega
+time           +tower          +strip          +money          +bally
+clean          +great          +circus         +spend          +location

Topic #7:
+encore         +wynn           +strip          +great          +mandalay
+area           +walk           +restaurant     +bay            +casino
+bellagio       +pool           +cafe           +nice           +shop

Topic #8:
+encore         +wynn           +suite          +w/             +service
+check          +guest          +bed            +floor          +area
+quot           +spa            +light          +give           +beautiful

Topic #9:
+check          +would          +quot           +ask            +go
+night          +kid            +back           +bed            +sheet
+get            +take           +desk           +could          +upgrade

TOPIC COMMENTS PERCENT
  1       290    0.0%
  2        34    0.0%
  3       165    0.0%
  4       386    0.0%
  5       110    0.0%
  6       217    0.0%
  7       150    0.0%
  8       152    0.0%
  9       167    0.0%
Number of Reviews.....  1671
Number of Terms....... 17527
('\nCorpus Average Sentiment: ', 1.3219487240725591)

Most Negative Reviews with 4 or more Sentiment Words:
 Review 57 Sentiment is -2.75

Most Positive Reviews with 4 or more Sentiment Words:
 Review 628 Sentiment is 3.75
 Review 1236 Sentiment is 3.75
('\nMost Negative Review Sentiment=', -2.75)
```

WORDCLOUDS:

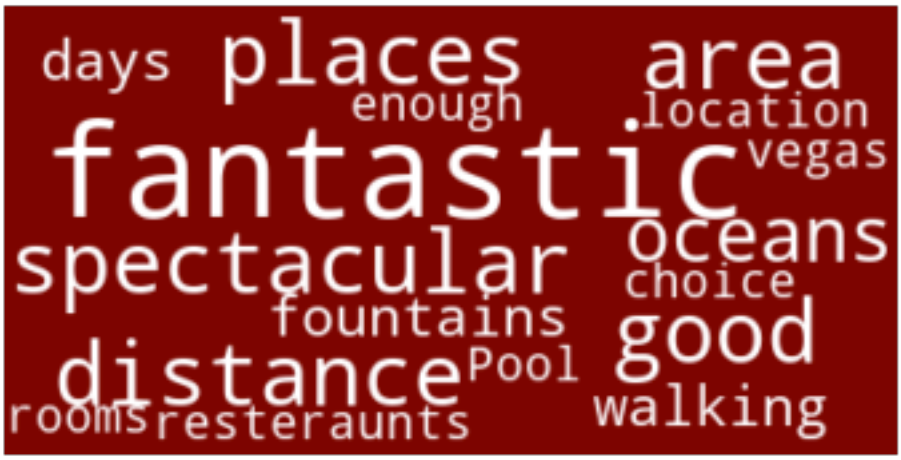
Name: Krit Gupta
UIN: 927001565





('\nMost Positive Review Sentiment=', 3.75)

Name: Krit Gupta
UIN: 927001565




```
('The Corpus contains a total of ', 485, ' unique sentiment words')
('The total number of sentiment words in the Corpus is', 22524)
```