Get spreadsheet names

In [22]:
```
!ls main_product/ > lights.txt
```

Import Libraries

In [45]:
```
import nltk
nltk.download('vader_lexicon')
nltk.download('punkt')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/kushthaker/nltk_data...
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/kushthaker/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[45]: True

In [46]:
```
import pandas as pd
import numpy as np
import time


from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
```

Requirement file

In [31]:
```
# !pip install -r requirements.txt
# !pip freeze > requirements.txt
```

In [32]:
```
with open('lights.txt','r') as l:
    csvs = [f.strip() for f in l.read().split('\n') if f != '']

df = pd.read_csv('main_product/' + csvs[0])

for csv in csvs[1:]:
    df = pd.concat([df, pd.read_csv('main_product/' + csv)])
```

Read a spreadsheet

In [33]:
```
df.columns
```

Out[33]:
```
Index(['Unnamed: 0', 'author_id', 'verified_purchase', 'review_title',
       'reviews', 'country', 'date', 'reviewer_name', 'ratings',
       'people_find_helpful', 'start_time', 'end_time', 'product_name',
       'average_rating', 'price', 'total_reviews', 'product_category',
```

```
       'product_id', 'meta_data', 'rank', 'product_image_url'],
      dtype='object')
```

In [36]:

```
# Keep column reviews only
df = df[['product_id','reviews']].dropna()
print(f"New shape: {df.shape}")
df.head(3)
```

```
New shape: (11434, 2)
```

Out[36]:

	product_id	reviews
0	B001NZO85O	Happy with the quality & make. Surpassed my ex...
1	B001NZO85O	Used it with 2 Fenix Flashlights E12 130-Lumen...
2	B001NZO85O	You won't be disappointed. This thing is aweso...

Partition reviews into sentences

In [ ]:

```
reviews_corpus = list(df['reviews'])
# Partition into sentences
reviews_in_sentences = [sent_tokenize(review.lower()) for review in reviews_corp
reviews_length = [len(review) for review in reviews_in_sentences]
print(f"Number of reviews: {len(reviews_in_sentences)}")
print(f"Number of sentences in each review: {reviews_length}")
```

!!! Need extra work on this!!!

Use Fliashlight corpus for matching sentences into aspects

https://docs.google.com/document/d/1ZiQVK4czqH0UGWZM1XaEhribQUT0xkI71Xh_aCGtw1E/edit

In [38]:

```
size_keywords_string = "Size, small, tiny, petite, slim, compact, large, big, gi
huge, enormous, gigantic, bulky, colossal, massive, sizable, weight, heavy, ligh

quality_keywords_string = "Build, built, quality,durability, sturdy, sturdiness,
coating, solid, cheaply, aluminum, steel, titanium, brass, copper,\
material, metal, rubber, plastic, nylon, bent, broke, faulty, shatter,\
waterproof, dustproof, corrosion, ingress, drop, shock, impact, resistance,\
screws, threads, knurling, anodized, flicker, housing, indestructible,\
wet, temperature, hot, heat, overheat, cold,  well"

battery_keywords_string = "Battery, batteries, rechargeable, charge, charging,\
recharge, USB, solar, runtime, hours, lifetime, dies, died, dead"

design_keywords_string = "Features, design, setting, settings, mode, modes,\
interface, programmable, memory, dim, roll, design, roll, upright, stand, tailst
strobe, sos, float, warranty, grip, rotate, rotating, head, hang, lantern, eco,
zoom, clip, lanyard, holster, indicator, easy to use, versatile, switch, twist,\
activation, clicky, click, magnetize, accessories, bezel"

beam_keywords_string = "Power, project, projects, far, illuminate, shine,\
focus, distance, range, feet, meters, beam, distance, visibility, throw, \
flood, lumens, bright, brightness, lens, optics, frosted, reflector, mule, LED,
colour, color, hotspot, spill, corona, lux, candelas, intensity, lights"
```

```
price_keywords_string = "Price, cost, costly, pricey, pricy, expensive, overpric
unreasonable, value, affordable, cheap, $, bargain, budget, cash, discount, mone


size_keywords = size_keywords_string.lower().replace(" ","").split(",")
quality_keywords = quality_keywords_string.lower().replace(" ","").split(",")
battery_keywords = battery_keywords_string.lower().replace(" ","").split(",")
design_keywords = design_keywords_string.lower().replace(" ","").split(",")
beam_keywords = beam_keywords_string.lower().replace(" ","").split(",")
price_keywords = price_keywords_string.lower().replace(" ","").split(",")
```

Helper functions

In [39]:

```
# checkPresence takes in:
# a sentence: represented by a string
# keywords: a list of keywords
# returns True if the sentence contains any of the keywords
def checkPresence(sentence, keywords):
    for keyword in keywords:
        if keyword in word_tokenize(sentence):
            return True
    return False

# checkPresence takes in:
# reviews_in_sentences: a list of list of sentences
#  (A review is represented by a list of sentences)
# keywords: a list of keywords
# returns: a list of filtered review which contains the keywords.
#         an empty string for a review that contains no keyword.
def filteredReview(reviews_in_sentences, keywords):
    ret = []
    for sentences in reviews_in_sentences:
        filtered = ''
        for sentence in sentences:
            if checkPresence(sentence, keywords):
                filtered += sentence
        ret.append(filtered)
    return ret
```

Match sentences into aspects

Took 4 seconds to process 80 reviews. (Need 250 seconds for 5000 reviews)

In [40]:

```
start = time.time()
df['size'] = filteredReview(reviews_in_sentences, size_keywords)
df['quality'] = filteredReview(reviews_in_sentences, quality_keywords)
df['battery'] = filteredReview(reviews_in_sentences, battery_keywords)
df['design'] = filteredReview(reviews_in_sentences, design_keywords)
df['beam'] = filteredReview(reviews_in_sentences, beam_keywords)
df['price'] = filteredReview(reviews_in_sentences, price_keywords)
end = time.time()
print(f"Took {end - start} seconds to match sentences into aspects.")
```

```
Took 962.9222347736359 seconds to match sentences into aspects.
```

In [41]:
```
df.head()
```

Out[41]:

	product_id	reviews	size	quality	battery	design	beam	price
0	B001NZO85O	Happy with the quality & make. Surpassed my ex...		happy with the quality & make.both fit well an...		both fit well and even accommodated the clip o...	excellent adjustability if you need a close-up...	
1	B001NZO85O	Used it with 2 Fenix Flashlights E12 130-Lumen...					used it with 2 fenix flashlights e12 130-lumen...	
2	B001NZO85O	You won't be disappointed. This thing is aweso...						
3	B001NZO85O	Works well with my Fenix LD22 flashlight as de...		works well with my fenix ld22 flashlight as de...				
4	B001NZO85O	old one was worn and stretched very good						

In [78]:
```
df.to_csv('sentence_partition.csv')
```

Sentiment Part

In [79]:
```
df_sent = pd.read_csv('sentence_partition.csv', encoding='utf8')
```

In [80]:
```
def find_sentiment(sentence):
    return sid.polarity_scores(sentence)

df.iloc[:,3:] = df.iloc[:,3:].applymap(find_sentiment)
```

In [83]:
```
df.to_csv('sentence_sentiment.csv')
```

In [136…
```
def get_size_scores(row):
    if type(row['size']) == dict:
        if row['size'].get('neg',0) and row['size'].get('pos',0):
            row['size'] = np.max((row['size'].get('neg',0), row['size'].get('pos
        elif row['size'].get('pos',0):
```

```
            row['size'] = row['size'].get('pos',0)
        elif row['size'].get('neg',0):
            row['size'] = row['size'].get('neg',0)
        else:
            row['size'] = 0

        return row
    return row
```

In [137…

```
def get_quality_scores(row):
    if type(row['quality']) == dict:
        if row['quality'].get('neg',0) and row['quality'].get('pos',0):
            row['quality'] = np.max((row['quality'].get('neg',0), row['quality']
        elif row['quality'].get('pos',0):
            row['quality'] = row['quality'].get('pos',0)
        elif row['quality'].get('neg',0):
            row['quality'] = row['quality'].get('neg',0)
        else:
            row['quality'] = 0

        return row
    return row
```

In [138…

```
def get_battery_scores(row):
    if type(row['battery']) == dict:
        if row['battery'].get('neg',0) and row['battery'].get('pos',0):
            row['battery'] = np.max((row['battery'].get('neg',0), row['battery']
        elif row['battery'].get('pos',0):
            row['battery'] = row['battery'].get('pos',0)
        elif row['battery'].get('neg',0):
            row['battery'] = row['battery'].get('neg',0)
        else:
            row['battery'] = 0

        return row
    return row
```

In [139…

```
def get_design_scores(row):
    if type(row['design']) == dict:
        if row['design'].get('neg',0) and row['design'].get('pos',0):
            row['design'] = np.max((row['design'].get('neg',0), row['design'].ge
        elif row['design'].get('pos',0):
            row['design'] = row['design'].get('pos',0)
        elif row['design'].get('neg',0):
            row['design'] = row['design'].get('neg',0)
        else:
            row['design'] = 0

        return row
    return row
```

In [140…

```
def get_beam_scores(row):
```

```
    if type(row['beam']) == dict:
        if row['beam'].get('neg',0) and row['beam'].get('pos',0):
            row['beam'] = np.max((row['beam'].get('neg',0), row['beam'].get('pos
        elif row['beam'].get('pos',0):
            row['beam'] = row['beam'].get('pos',0)
        elif row['beam'].get('neg',0):
            row['beam'] = row['beam'].get('neg',0)
        else:
            row['beam'] = 0

        return row
    return row
```

In [141…

```
def get_price_scores(row):
    if type(row['price']) == dict:
        if row['price'].get('neg',0) and row['price'].get('pos',0):
            row['price'] = np.max((row['price'].get('neg',0), row['price'].get('
        elif row['price'].get('pos',0):
            row['price'] = row['price'].get('pos',0)
        elif row['price'].get('neg',0):
            row['price'] = row['price'].get('neg',0)
        else:
            row['price'] = 0

        return row
    return row
```

In [146…

```
df = df.apply(get_size_scores,axis=1)
df = df.apply(get_quality_scores,axis=1)
df = df.apply(get_battery_scores,axis=1)
df = df.apply(get_design_scores,axis=1)
df = df.apply(get_beam_scores,axis=1)
df = df.apply(get_price_scores,axis=1)
```

In [147…

```
df.head()
```

Out[147…

	product_id	reviews	size	quality	battery	design	beam	price
0	B001NZO85O	Happy with the quality & make. Surpassed my ex...	0.0	0.409	0.0	0.338	0.316	0.0
1	B001NZO85O	Used it with 2 Fenix Flashlights E12 130-Lumen...	0.0	0.000	0.0	0.000	0.000	0.0
2	B001NZO85O	You won't be disappointed. This thing is aweso...	0.0	0.000	0.0	0.000	0.000	0.0
3	B001NZO85O	Works well with my Fenix LD22 flashlight as de...	0.0	0.208	0.0	0.000	0.000	0.0
4	B001NZO85O	old one was worn and stretched very good	0.0	0.000	0.0	0.000	0.000	0.0

In [149…

```
products = df.product_id.unique()
```

```
sent_df_final = pd.DataFrame({'product':products,
                              'size':np.zeros(len(products)),
                              'quality':np.zeros(len(products)),
                              'battery':np.zeros(len(products)),
                              'design':np.zeros(len(products)),
                              'beam':np.zeros(len(products)),
                              'price':np.zeros(len(products))
                             })
```

Out[149…

	product	size	quality	battery	design	beam	price
0	B001NZO85O	0.0	0.0	0.0	0.0	0.0	0.0
1	B005CWRB44	0.0	0.0	0.0	0.0	0.0	0.0
2	B0062PVSGW	0.0	0.0	0.0	0.0	0.0	0.0
3	B0091TRPVI	0.0	0.0	0.0	0.0	0.0	0.0
4	B00937X7G0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
85	B0841RSDCR	0.0	0.0	0.0	0.0	0.0	0.0
86	B086PW9TTP	0.0	0.0	0.0	0.0	0.0	0.0
87	B087CG1YW6	0.0	0.0	0.0	0.0	0.0	0.0
88	B08BTQ2T4C	0.0	0.0	0.0	0.0	0.0	0.0
89	B08DCSF6ZX	0.0	0.0	0.0	0.0	0.0	0.0

90 rows × 7 columns

In [195…

```
# sent_df_final.loc[sent_df_final['product'] == 'B08DCSF6ZX']['size']
```

Out[195…

	product	size	quality	battery	design	beam	price
0	B001NZO85O	1.0	0.0	0.0	0.0	0.0	0.0
1	B005CWRB44	0.0	0.0	0.0	0.0	0.0	0.0
2	B0062PVSGW	0.0	0.0	0.0	0.0	0.0	0.0
3	B0091TRPVI	0.0	0.0	0.0	0.0	0.0	0.0
4	B00937X7G0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
85	B0841RSDCR	0.0	0.0	0.0	0.0	0.0	0.0
86	B086PW9TTP	0.0	0.0	0.0	0.0	0.0	0.0
87	B087CG1YW6	0.0	0.0	0.0	0.0	0.0	0.0
88	B08BTQ2T4C	0.0	0.0	0.0	0.0	0.0	0.0
89	B08DCSF6ZX	0.0	0.0	0.0	0.0	0.0	0.0

90 rows × 7 columns

In [198…

```
size_df = df.loc[df['size'] > 0]
quality_df = df.loc[df['quality'] > 0]
battery_df = df.loc[df['battery'] > 0]
design_df = df.loc[df['design'] > 0]
beam_df = df.loc[df['beam'] > 0]
price_df = df.loc[df['price'] > 0]

for i,product in enumerate(products):
    sent_df_final.loc[i,'size'] = np.mean(size_df.loc[size_df.product_id == prod
    sent_df_final.loc[i,'quality'] = np.mean(quality_df.loc[quality_df.product_i
    sent_df_final.loc[i,'battery'] = np.mean(battery_df.loc[battery_df.product_i
    sent_df_final.loc[i,'design'] = np.mean(design_df.loc[design_df.product_id =
    sent_df_final.loc[i,'beam'] = np.mean(beam_df.loc[beam_df.product_id == prod
    sent_df_final.loc[i,'price'] = np.mean(price_df.loc[price_df.product_id == p
```

In [199…

```
sent_df_final
```

Out[199…

	product	size	quality	battery	design	beam	price
0	B001NZO85O	0.162880	0.256529	0.146292	0.210179	0.215273	0.243250
1	B005CWRB44	0.276473	0.228552	0.178218	0.166431	0.295173	0.277621
2	B0062PVSGW	0.271035	0.246449	0.175576	0.208615	0.275401	0.241623
3	B0091TRPVI	0.288658	0.290800	0.194815	0.181698	0.259889	0.295250
4	B00937X7G0	0.334429	0.264286	0.202900	0.226259	0.326632	0.373400
…	…	…	…	…	…	…	…
85	B0841RSDCR	0.238538	0.236000	0.167667	0.136273	0.297864	0.142000
86	B086PW9TTP	0.368500	0.105500	0.229500	0.179000	0.252500	0.104000
87	B087CG1YW6	0.359333	0.272818	0.210200	0.184333	0.373156	0.217778
88	B08BTQ2T4C	0.304944	0.193600	0.227417	0.203429	0.289783	0.204667
89	B08DCSF6ZX	0.175200	0.217000	0.201778	0.233700	0.252063	0.178750

90 rows × 7 columns

In [200…

```
sent_df_final.to_csv('some_sentiment.csv')
```

In [ ]: