Analysis of Reddit Comment Sentiments

Authors: Cameron Mandley, Jasmine Wu

Provide your credentials to the runtime

```
from google.colab import auth
auth.authenticate_user()
print('Authenticated')
```

```
Authenticated
```

Use Classic DataFrame Display

```
%unload_ext google.colab.data_table
```

```
The google.colab.data_table extension is not loaded.
```

Declare the Cloud project ID which will be used throughout this notebook

```
project_id = 'mindful-marking-297202'
```

Sample approximately 2000 random rows

```
%load_ext google.cloud.bigquery
```

```
from google.cloud import bigquery
client = bigquery.Client(project=project_id)

subreddit_list = client.query('''
  SELECT subreddit, count(*)
    FROM `fh-bigquery.reddit_comments.2010`
    GROUP BY subreddit
    '''
  ).to_dataframe()
```

```
WARNING:google.auth._default:No project ID could be determined. Consider running
```

```
from google.cloud import bigquery
client = bigquery.Client(project=project_id)

df = client.query('''
SELECT *
FROM `fh-bigquery.reddit_comments.2010` AS x
WHERE subreddit IN(
    SELECT subreddit
    FROM `fh-bigquery.reddit_comments.2010`
    GROUP BY subreddit
    HAVING COUNT(*) > 10000
    ORDER BY COUNT(*)
    LIMIT 50
)
    '''
  ).to_dataframe()
```

```
WARNING:google.auth._default:No project ID could be determined. Consider running
```

```
df.head()
```

	body	score_hidden	archived	name	author	author_flair_text	c
1	Was the idea of the policy good? Yes Was	False	True	t1_c0yip0s	bradmurray	None	

```
def grab_Month(date):
  return str(date)[5:7]
```

offense

```
import datetime
df['Date'] = df['created_utc'].apply(datetime.datetime.fromtimestamp)
df['Month'] = df['Date'].apply(grab_Month)
```

...

Remove Deleted Comments

can!

```
df = df[df['body'] != '[deleted]']
```

bad.

```
from textblob import TextBlob
import pandas as pd
import numpy as np
from numpy import random
import nltk
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
get_ipython().run_line_magic('matplotlib', 'inline')
from nltk import word_tokenize, download
import string
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

Remove Stopwords and Tokenize Data (takes a long time to run this next cell)

```
from tqdm import tqdm
nltk.download('stopwords')
nltk.download('punkt')

comments = df['body']
comments_processed = []
for sentence in tqdm(comments):
    comments_processed.append(' '.join(token.lower() for token in nltk.word_tokenize(s
```

```
  0%|          | 0/566132 [00:00<?, ?it/s][nltk_data] Downloading package stopwo
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
100%|██████████| 566132/566132 [57:59<00:00, 162.69it/s]
```

Format Comments

```
from collections import Counter

# split() returns list of all the words in the string
total_comments = [0] * len(comments_processed)
i = 0
for comment in comments_processed:
  split_comment = comment.split()
  total_comments[i] = split_comment
  i = i+1


punc = '''!()-[]{};:'"\, <>./?@#$%^&*_~`'''
# Removing punctuations in string
# Using loop + punctuation string
for i in range(len(total_comments)):
  for j in range(len(total_comments[i])):
    for ele in total_comments[i][j]:
      if ele in punc:
        total_comments[i][j] = 'None'
  total_comments[i] = np.array(total_comments[i])[np.array(total_comments[i]) != 'None
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: FutureWarning: e
  if __name__ == '__main__':
```

```
string_comments = [str(comment).replace('[', '').replace(']', '').replace('\'', '') fo
```

Determine Comment Polarities

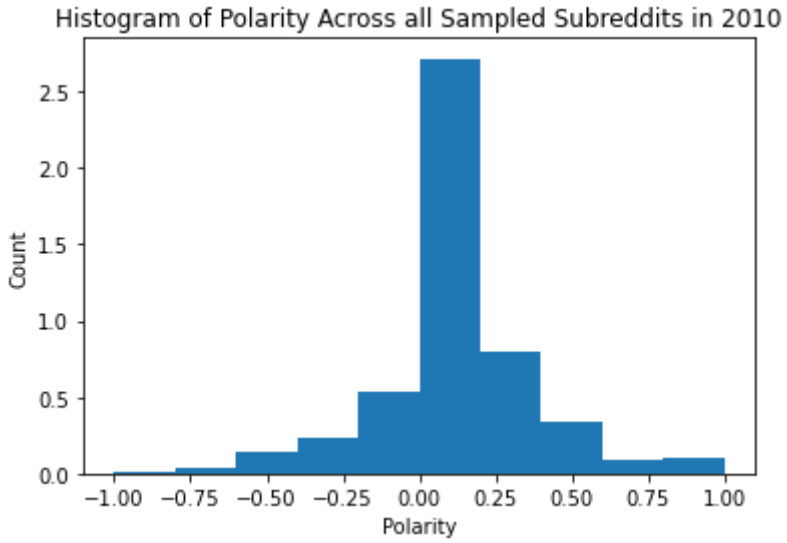
```
def detect_polarity(text):
  return TextBlob(text).sentiment.polarity


polarity = [detect_polarity(comment) for comment in string_comments]


comments_processed_df = pd.DataFrame({'body': comments_processed, 'polarity': polarity


plt.hist(comments_processed_df['polarity'], density = True)
plt.xlabel('Polarity')
plt.ylabel('Count')
```

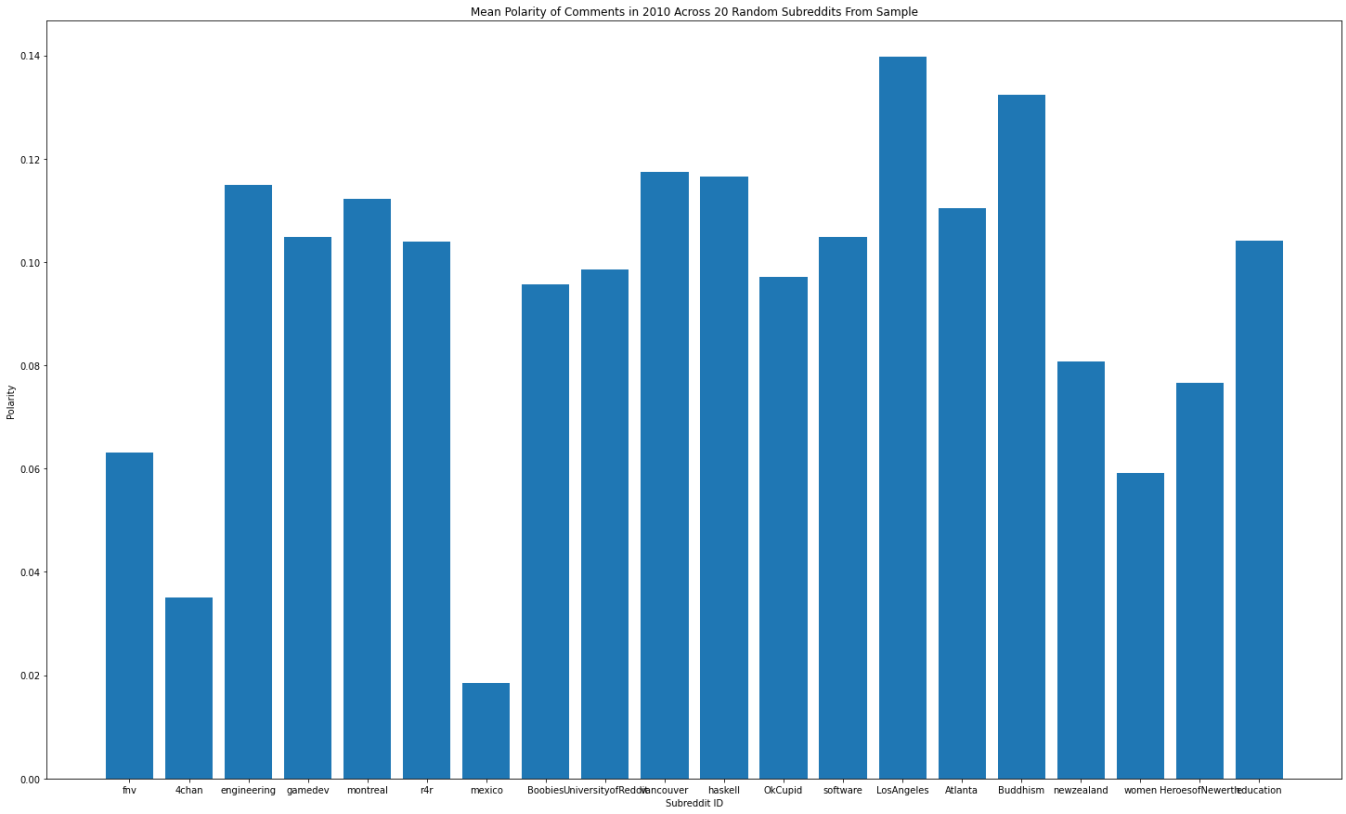
```
plt.ylabel('Count')
plt.title('Histogram of Polarity Across all Sampled Subreddits in 2010');
```




```
import seaborn as sns


aggregated = comments_processed_df.groupby('subreddit_id').agg(np.mean).sample(20)


plt.figure(figsize=(25,15))
plt.bar(x=aggregated.index, height = aggregated['polarity'])
plt.xlabel('Subreddit ID')
plt.ylabel('Polarity')
plt.title('Mean Polarity of Comments in 2010 Across 20 Random Subreddits From Sample')
plt.show();
```



Initialize and Fill Retention Dictionary

```
retention_rate = {}
for i in df['subreddit'].unique():
  retention_rate[i] = {}
  for j in sorted(df['Month'].unique()):
    retention_rate[i][j] = 0


months = sorted(df['Month'].unique())

for subreddit in df['subreddit'].unique():
  for j in range(11):
    i = 0
    first_month_authors = df[df['Month'] == months[j]][df['subreddit'] == subreddit][

    if len(first_month_authors) == 0:
      retention_rate[subreddit][months[j]] = 0

    else:
      second_month_authors = df[df['Month'] == months[j+1]][df['subreddit'] == subredd
      for author in first_month_authors:
        if author in second_month_authors:
          i += 1

      retention_rate[subreddit][months[j]] = i/len(first_month_authors)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning: Boo
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: UserWarning: Bo
  if sys.path[0] == '':
```

```
r = pd.DataFrame(columns = months)
```

Plot Retention Rate Over the Year for Each Subreddit

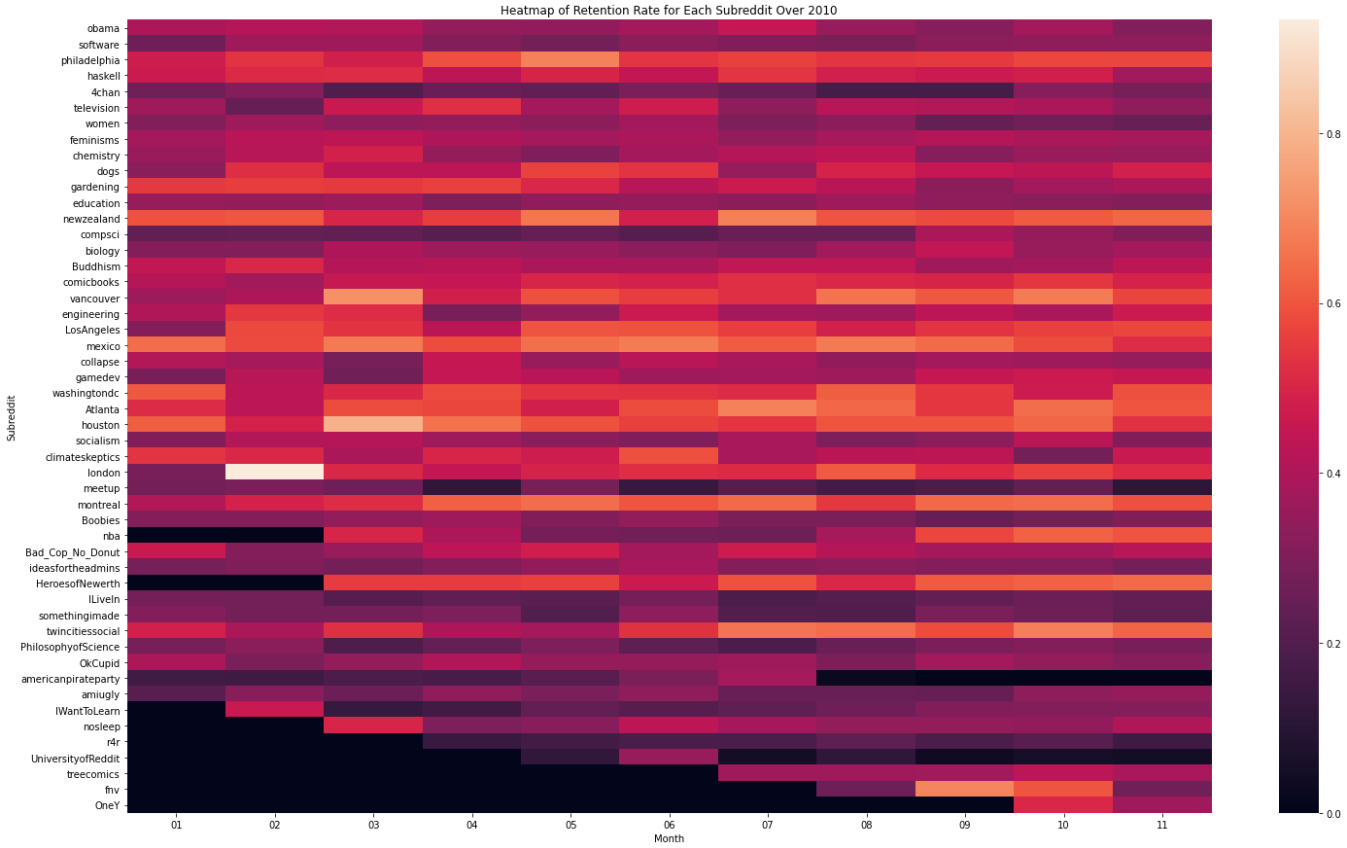
```
subreddits = df['subreddit'].unique()
mean_retention_rates = np.array([])
retention_df = pd.DataFrame({})

for subreddit in subreddits:
  monthly_retention_rates = []
  for month in months:
    monthly_retention_rates = np.append(monthly_retention_rates, retention_rate[subre
  r.loc[subreddit] = monthly_retention_rates
  #sns.heatmap(month, subreddit, monthly_retention_rates[-1])
  #plt.plot(months[:11], monthly_retention_rates[:11])

  mean_retention_rates = np.append(mean_retention_rates, np.mean(monthly_retention_ra
```

```
plt.figure(figsize=(25,15))
sns.heatmap(r[months[:11]])
plt.ylabel('Subreddit')
plt.xlabel('Month')
plt.title('Heatmap of Retention Rate for Each Subreddit Over 2010')
```

```
Text(0.5, 1.0, 'Heatmap of Retention Rate for Each Subreddit Over 2010')
```

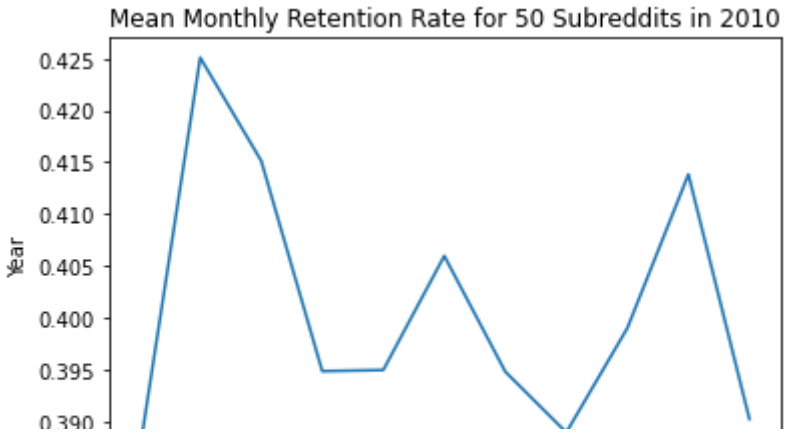



```
months
```

```
['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12']
```

```
subreddits = df['subreddit'].unique()
means = np.array([])

for month in months:
  subreddit_retention_rates = []
  for subreddit in subreddits:
    if retention_rate[subreddit][month] == np.inf:
      break
    subreddit_retention_rates = np.append(subreddit_retention_rates, retention_rate[su
  means = np.append(means, np.mean(subreddit_retention_rates))
plt.plot(months[:11], means[:11])
plt.title('Mean Monthly Retention Rate for 50 Subreddits in 2010')
plt.xlabel('Month')
plt.ylabel('Year');
```



Plot Yearly Mean Retention Rate for Each Subredit

Month

```
plt.figure(figsize=(25,15))
plt.barh(y = subreddits, width = mean_retention_rates)
```

```
<BarContainer object of 50 artists>
```