Gorges_Analysis_INFO_3350_Final

December 9, 2023

Analyzing Ithaca's Gorge Stewarding Program

Ming DeMers
12/9/2023
Hours worked: 30

1 Abstract

This project aims to analyze the textual aspect of the logbook. We identify two types of entries: interactions and observation. The former is what we mainly focus on, as its where stewards talk and interact with those in the park, including those who are violating the rules. Due to the nature of the logbook, the analysis can only confidently account for post-2018. We first look at the sentiment of interactions and find that they are overall neutral, though partially negative. Over time, the sentiment has also lowered. We then observe that interactions and observations are similar to Euclidean observation; we choose to only analyze interactions therein. We then use unsupervised learning to try to discover clusters. We find there are three primary clusters: no violation, violation, and violation in Cascadilla Falls (one of the three gorges). We are interested in if there are different clusters or topics of violations. Finally, we try to predict if an interaction is considered a violation.

2 Background

The gorges at Cornell University are one of the university's most iconic and scenic features. Nestled in the heart of the campus, the gorges are a series of deep and narrow ravines carved by the flow of water over thousands of years. These beautiful natural formations offer a tranquil escape from the hustle and bustle of campus life, providing students and visitors with picturesque spots for relaxation and exploration.

While the gorges are captivating, they can be deceptively dangerous. Safety measures are enforced and visitors are urged to follow the guidelines.

The Gorge Stewards Program was established in 2014 to improve safety and enjoyment of the gorges, following the death of a student earlier that year. Gorge stewards walk the paths from May to September to provide information about trails, safety rules, natural history, activities, and swimming alternatives. Stewards also track visitor use, including unauthorized or illegal uses.

The stewards contribute to a logbook that tabulates the amount of visitors in the gorges, amount of violations, as well as tracking the temperature and issues/hazards in the gorge. The stewards describe each interaction they have and observations they make during their shift.

3 The Data

The data is sourced from the Gorge Stewards Program, courtesy of the Cornell Botanic Gardens and Cornell Outdoor Education (COE). It comprises of a table for each year recorded, with a number of variables detailing gorge usage, violations, weather and more.

Violations are when visitors do not abide by posted rules of the Gorge. These include swimming, diving, or being in certain parts of the stream, trespassing, illegal substance possessions, and more. The majority of violations involve individuals trying to swim - a deceptively dangerous activity, often following to the death of a person every few years. Since the establishment of the program, no deaths or rescues have been reported; and violations have steadily decreased as more people have visited the beautiful gorges.

Because much of the log book are qualitative entries, we utilize textual analysis to attempt to extract quantitative meaning and measures.

4 Experiment and Hypothesis

With this data, we hope to first glean some interesting insight from the log book. We further hope to see how we might be able to determine what an "interaction" is, how it compares to "observations," and see if there are different types of interactions. We finally look to see if we can predict within these types, based on entry texts, dates, and the steward writing it.

It is hypothesis that there are different types of interactions: violation, direction giving, and other. As for observation, we predict there are observations of trash, of broken/vandalised parts of the gorges, and weather. We think we can predict these states, based on the entry message.

5 Import and Setup

We import and install necessary packages. We also clean and split the data appropiately.

5.1 Imports

[ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

[3]:
```
# !pip install hdbscan
# !pip install umap-learn
```

```
Collecting hdbscan
  Downloading hdbscan-0.8.33.tar.gz (5.2 MB)
                             5.2/5.2 MB
14.8 MB/s eta 0:00:00
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Collecting cython<3,>=0.27 (from hdbscan)
  Using cached Cython-0.29.36-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (1.9 MB)
```

```
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.23.5)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.10/dist-packages (from hdbscan) (1.2.2)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20->hdbscan)
(3.2.0)
Building wheels for collected packages: hdbscan
  Building wheel for hdbscan (pyproject.toml) … done
  Created wheel for hdbscan:
filename=hdbscan-0.8.33-cp310-cp310-linux_x86_64.whl size=3039180
sha256=28c1d22f1879530bf440798303935e95d79f5d71f1b844b186b424e7506956b0
  Stored in directory: /root/.cache/pip/wheels/75/0b/3b/dc4f60b7cc455efaefb62883
a7483e76f09d06ca81cf87d610
Successfully built hdbscan
Installing collected packages: cython, hdbscan
  Attempting uninstall: cython
    Found existing installation: Cython 3.0.6
    Uninstalling Cython-3.0.6:
      Successfully uninstalled Cython-3.0.6
Successfully installed cython-0.29.36 hdbscan-0.8.33
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
                             90.9/90.9 kB
2.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (1.2.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (0.58.1)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
                             55.8/55.8 kB
8.2 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (4.66.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn)
(0.41.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
```

```
packages (from pynndescent>=0.5->umap-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->umap-learn)
(3.2.0)
Building wheels for collected packages: umap-learn
  Building wheel for umap-learn (setup.py) … done
  Created wheel for umap-learn: filename=umap_learn-0.5.5-py3-none-any.whl
size=86832
sha256=5376a2985d6686d775c38c0e9c92917f6ccacf58f3464c444c4a1e0b6d378930
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a3b431db59b
806a10da736612ebbc66c1bcc5
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
```

[4]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
from collections import Counter
from nltk.corpus import stopwords
import string
from nltk import sent_tokenize, word_tokenize
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances, cosine_distances,␣
 ↪cosine_similarity
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
import seaborn as sns
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB, GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score,␣
 ↪recall_score, f1_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.cluster import DBSCAN
import re
from sklearn.svm import SVC
from umap import UMAP
from sklearn.model_selection import train_test_split
from sklearn.decomposition import LatentDirichletAllocation
from   sklearn.preprocessing import StandardScaler
import spacy
from   transformers import DistilBertTokenizerFast, DistilBertModel,␣
 ↪DistilBertForSequenceClassification
import requests
import torch
from transformers import Trainer, TrainingArguments
import random
from collections import defaultdict
from transformers import AutoTokenizer, AutoModelForSequenceClassification,␣
 ↪Trainer, TrainingArguments
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data…
```

[5]:
```
np.random.seed(42) # set to keep each run consistent
```

5.2 Data Cleaning

We import the the gorges dataset and clean some fields that are erroneous.

[6]:
```
# read in data
gorges = pd.read_csv("/content/gorges.csv")
gorges = gorges.iloc[:, :-12]

# fixing some broken dates
gorges.loc[146, 'Date'] = "6/30/2022"
gorges.loc[215, 'Date'] = "5/28/2021"
gorges.loc[509, 'Date'] = "6/5/2019"
gorges.loc[529, 'Date'] = "6/19/2019"
gorges.loc[558, 'Date'] = "7/3/2019"
gorges.loc[589, 'Date'] = "7/17/2019"
gorges.loc[599, 'Date'] = "7/22/2019"
gorges.loc[620, 'Date'] = "7/31/2019"
gorges.loc[649, 'Date'] = "8/14/2019"
gorges.loc[669, 'Date'] = "8/28/2019"
gorges.loc[667, 'Date'] = "8/28/2019"

# convert date into a date object
```

```
gorges['Date'] = pd.to_datetime(gorges['Date'])

print(len(gorges))
gorges.head()
```

```
1107
```

[6]:

	Date	Patrol Time	Steward name	High Temperature	Weather	Number of observed gorge users	total Number of people obseved violating a rule	Number of person interactions: Alternatives	Number of person interactions: Warnings	Number of person interactions: Directions
0	2023-05-25	11AM-7PM	Brenner	61	sunny, windy	168	0	0.0	0.0	1.0
1	2023-05-26	11AM-7PM	Brenner	68	sunny, clear	264	0	0.0	0.0	9.0
2	2023-05-27	11AM-7PM	Brenner	77	sunny	251	2	0.0	2.0	5.0
3	2023-05-28	11-7PM	Phillip	81	Sunny	184	5	0.0	5.0	9.0
4	2023-05-29	11-7PM	Phillip	85	Sunny	102	0	0.0	0.0	

```
4                                        6.0

   Total number of person interactions (Daily) Number of contacts to CUPD  \
0                                         1.0                           0
1                                         9.0                           0
2                                         7.0                           0
3                                        14.0                           0
4                                         0.0                           0

  Number of People above Ithaca Falls?  \
0                                    0
1                                    0
2                                    0
3                                    0
4                                    0

                        Description of Interactions  \
0                   Someone asked about north campus
1           lots of commencement direction questions
2        few people suntanning on rocks in the water
3  Lots of people looking for arboretum, three do…
4      Big day for tourists, small day for violators

                       Description of notable events
0                              saw some geese babies!
1                                numerous grad photos
2            someone asked me where the slope was lol
3                               I saw a cool butterfly
4  Found boathouse key on Bebee lake trail. Is no…
```

5.3 Interactions/Observation Splitting

For interactions and observations, we split the data. That is, becuase many of the interaction entry actually contain more than one interaction, we create additional rows. For example, the interaction entry:

> "saw three people in gorge, chainlink fence broken, two dogs off leash"

becomes

> "saw three people in the gorge"
> "chainlink fence broken"
> "two dogs off leash"

[24]:
```
# extract out the observations column that aren't nan
interactions_df = gorges[gorges['Description of Interactions'].notna()]
interactions_df = interactions_df[interactions_df['Description of␣
 ↪Interactions'] != '0']
interactions_df = interactions_df.iloc[:285, :]
```

```
# keep only non-na observations
observations_df = gorges[gorges['Description of notable events'].notna()]

#place all the interactions and observations into a dictionary (this is later␣
 ↪uncased, but may be helpful)
log = {'interaction': [],'observation': []}

for i in range(len(interactions_df)):
    msg = {'index': i,
           'date': interactions_df.iloc[i, 0],
           "steward": interactions_df.iloc[i, 2],
           "msg": interactions_df.iloc[i, 13]}
    log['interaction'].append(msg)

for i in range(len(observations_df)):
    msg = {'index': i,
           'date': observations_df.iloc[i, 0],
           "steward": observations_df.iloc[i, 2],
           "msg": observations_df.iloc[i, 14]}
    log['observation'].append(msg)
```

[25]:
```
# reassign dict to respective lists
interactions = log['interaction']

print("Interactions before split:", len(interactions))

interactions_split = []

for interaction in interactions:
    parts = re.split(r'[.,;]', interaction['msg'])
    for part in parts:
        part = part.strip()
        if part:
            new_inter = interaction.copy()
            new_inter['msg'] = part
            interactions_split.append(new_inter)


for interaction in interactions:
    parts = re.split(r'[.,;]', interaction['msg'])
    for part in parts:
        part = part.strip()
        if part:
            new_inter = interaction.copy()
            new_inter['msg'] = part
            interactions_split.append(new_inter)
```

```
interactions_unsplit = []
interactions_unsplit = interactions
interactions = interactions_split

print("Total Interactions after split:", len(interactions))
print("First three interactions:")
for i in range(3):
  print(interactions[i])
print("")
```

```
Interactions before split: 285
Total Interactions after split: 1158
First three interactions:
{'index': 0, 'date': Timestamp('2023-05-25 00:00:00'), 'steward': 'Brenner',
'msg': 'Someone asked about north campus'}
{'index': 1, 'date': Timestamp('2023-05-26 00:00:00'), 'steward': 'Brenner',
'msg': 'lots of commencement direction questions'}
{'index': 2, 'date': Timestamp('2023-05-27 00:00:00'), 'steward': 'Brenner',
'msg': 'few people suntanning on rocks in the water'}
```

[16]:
```
# observations cleaning
observations = log['observation']

observations_split = []
observations_unsplit = []

for entry in observations:
    # Split the 'msg' by commas, periods, and semicolons
    parts = re.split(r'[.,;]', entry['msg'])
    for part in parts:
        # Remove leading and trailing whitespaces from each part
        part = part.strip()
        if part:
            new_entry = entry.copy()  # Create a copy of the original entry
            new_entry['msg'] = part  # Replace 'msg' with the current part
            observations_split.append(new_entry)

observations_unsplit = observations
observations = observations_split

print("Total before split observations:", len(observations_unsplit))
print("Total after split observations:", len(observations_split))
print('First three observation texts:')
for i in range(220, 223):
  print(observations[i]['msg'])
```

```
Total before split observations: 223
Total after split observations: 403
First three observation texts:
Soooooo many people out today
I think it might be a record for me
People everywhere!
```

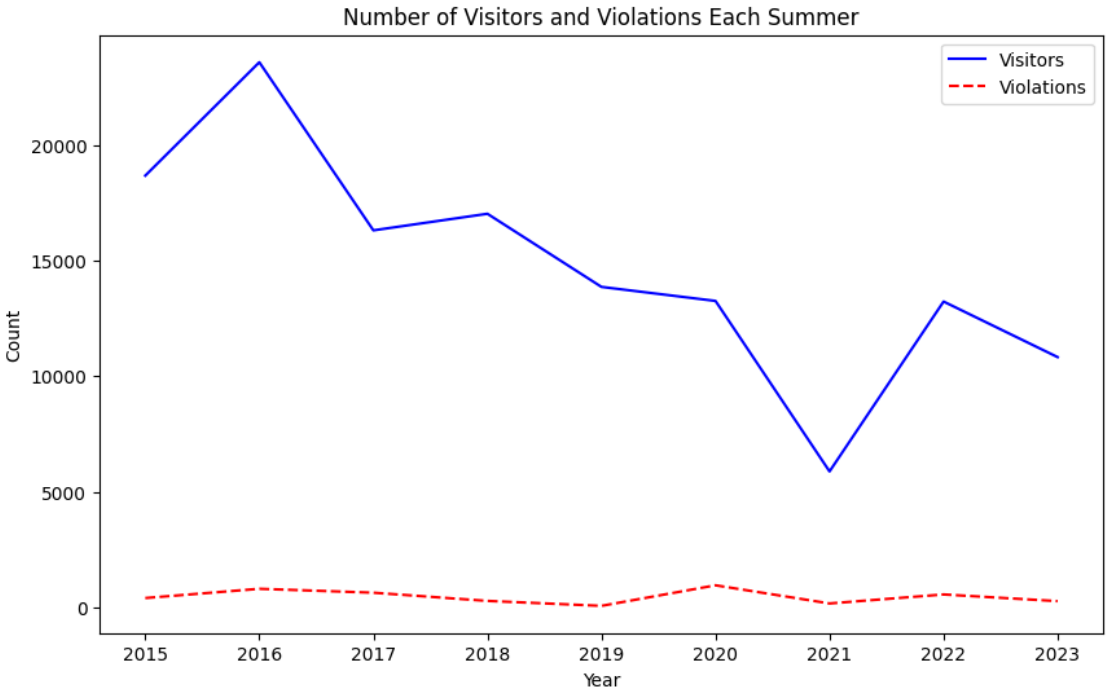
What would be only 285 interactions now becomes over 1,100, and we almost double our observations. While some interactions and observations are now empty characters or one word, they are handled in the future, and overall provide a more nuanced dataset to work on.

6 Exploratory Data Analysis

We analyze some overall trends and get a better idea of the data.

6.1 Visitorship Over Time

[17]:
```
gorges_tbl = pd.read_csv('/content/gorges_table.csv') # this dataset was␣
 ↪generated from the r datareport

plt.figure(figsize=(10, 6))

plt.plot(gorges_tbl['year(date)'], gorges_tbl['visitors'], label='Visitors',␣
 ↪color='blue', linestyle='solid')
plt.plot(gorges_tbl['year(date)'], gorges_tbl['violations'],␣
 ↪label='Violations', color='red', linestyle='dashed')

plt.title('Number of Visitors and Violations Each Summer')
plt.xlabel('Year')
plt.ylabel('Count')
plt.legend()

plt.show()
```

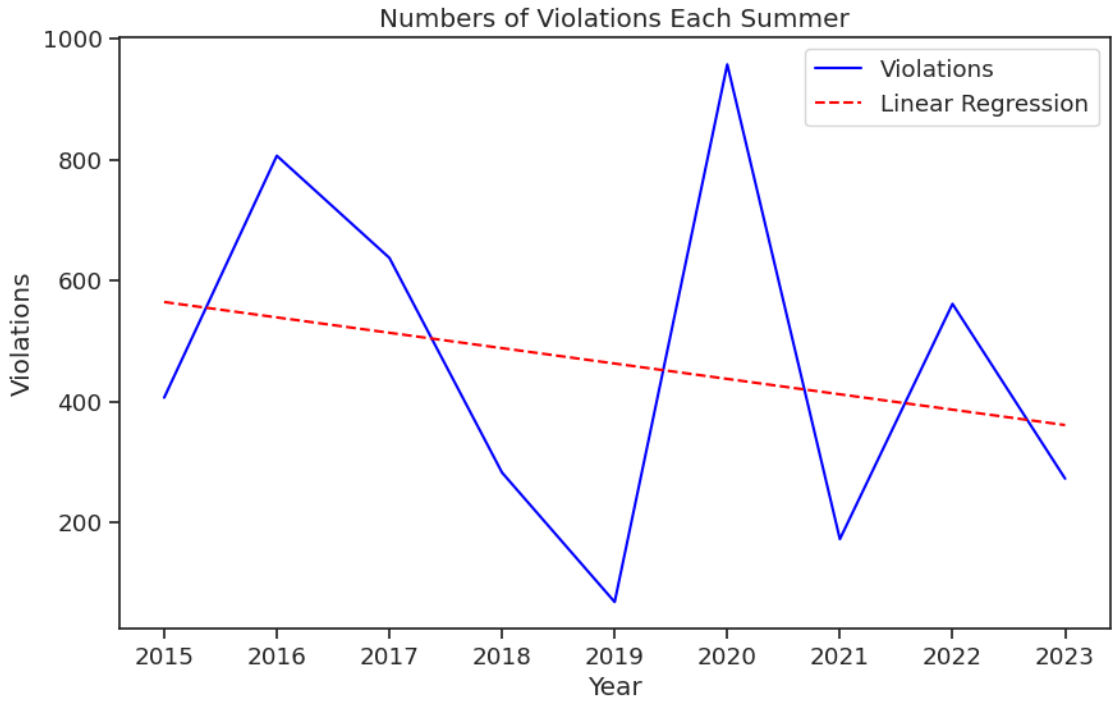


Interestingly, we see the amount of visitors spikes in 2016, and falls to it lowest in 2021. 2020 shows only slightly lesser numbers than previous years, and 2022 shows membership returned near fully. It's hard to determine why this drop occured - the COVID-19 Pandemic would be a likely explanation, but that doesn't seem to account for 2020's regular number.

6.2 Violations Across the Summers

[ ]:
```
gorges_tbl = pd.read_csv('/content/gorges_table.csv') # this dataset was␣
 ↪generated from the r datareport

plt.figure(figsize=(10, 6))

plt.plot(gorges_tbl['year(date)'], gorges_tbl['violations'],␣
 ↪label='Violations', color='blue', linestyle='solid')
plt.title('Numbers of Violations Each Summer')
plt.xlabel('Year')
plt.ylabel('Violations')
plt.legend()

# Adding a linear regression line
coefficients = np.polyfit(gorges_tbl['year'], gorges_tbl['violations'], 1)
polynomial = np.poly1d(coefficients)
plt.plot(gorges_tbl['year'], polynomial(gorges_tbl['year']), color='red',␣
 ↪linestyle='dashed', label='Linear Regression')
```

```
plt.legend()
plt.show()
```

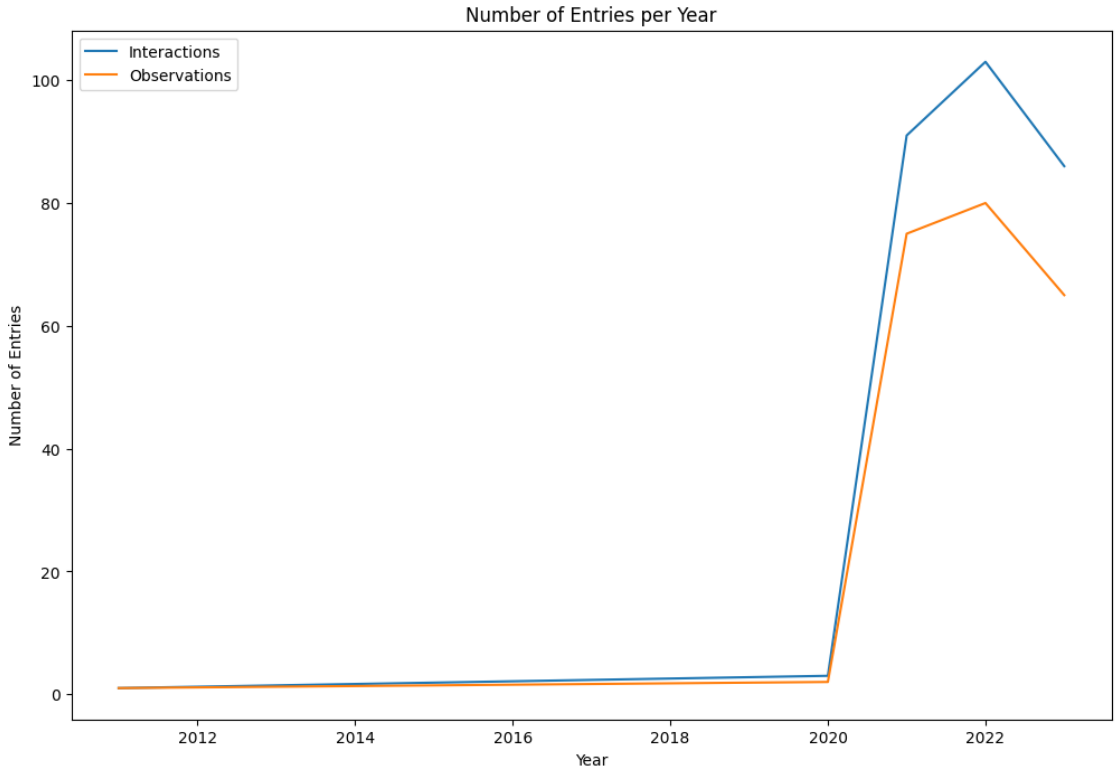


Over the 8 years of the program, violations overall have a downward trend, which is a good outcome of the Stewards Program. However, the decline is not steady, and there's a noticeable spike in 2020 and 2022. Again, this could be explained by the COVID pandemic: one of the few activities during the pandemic was going outside, and perhaps individuals did not follow rules as they were unaware of them or more careless during that year.

6.3 Log Book Entries Over Time

[ ]:
```
# amount of rows per year

num_inter = interactions_df.groupby('Date').size().groupby(lambda x: x.year).
 ↪sum()
num_obsv = observations_df.groupby('Date').size().groupby(lambda x: x.year).
 ↪sum()

# graph this out
plt.figure(figsize=(12, 8))
plt.plot(num_inter, label='Interactions')
plt.plot(num_obsv, label='Observations')
plt.xlabel('Year')
```

```
plt.ylabel('Number of Entries')
plt.title('Number of Entries per Year')
plt.legend()
plt.show()
```



We see that entries in the log book were very low until 2019, were they saw a spike and remained around 90 entries each summer. This is because the logbook changed its format in 2019. Previously there were numerous columns that asked for granular data; post-2018, there were only fields for violations and observations. This does affect the data, and the report can only realistically apply to post-2018 gorges.

6.4 Word Frequency

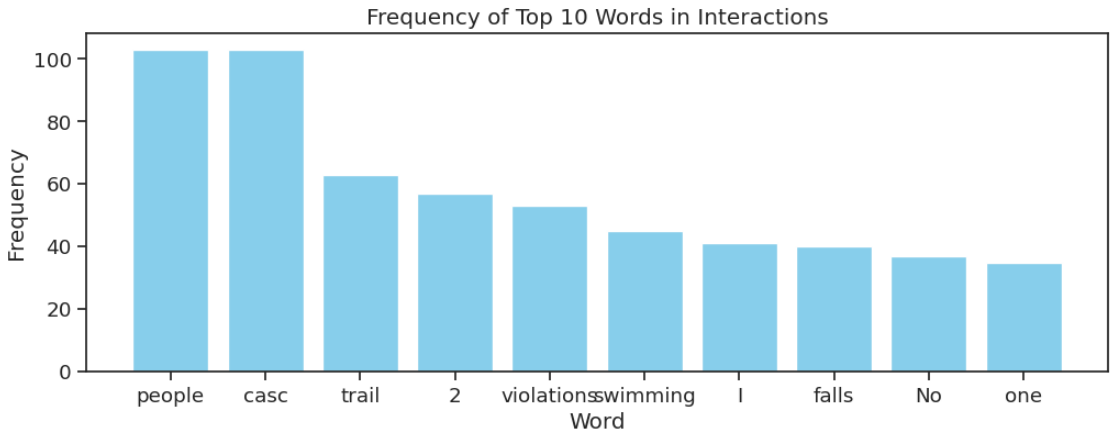
[ ]:
```
# return 20 most frequent words, not including stopwords

eng_stopwords = stopwords.words('english')
eng_stopwords.extend(string.punctuation)
eng_stopwords.extend(['lower'])

def word_stats(data, stopwords=eng_stopwords, n=20):
    if (stopwords != None):
        for stopword in stopwords:
```

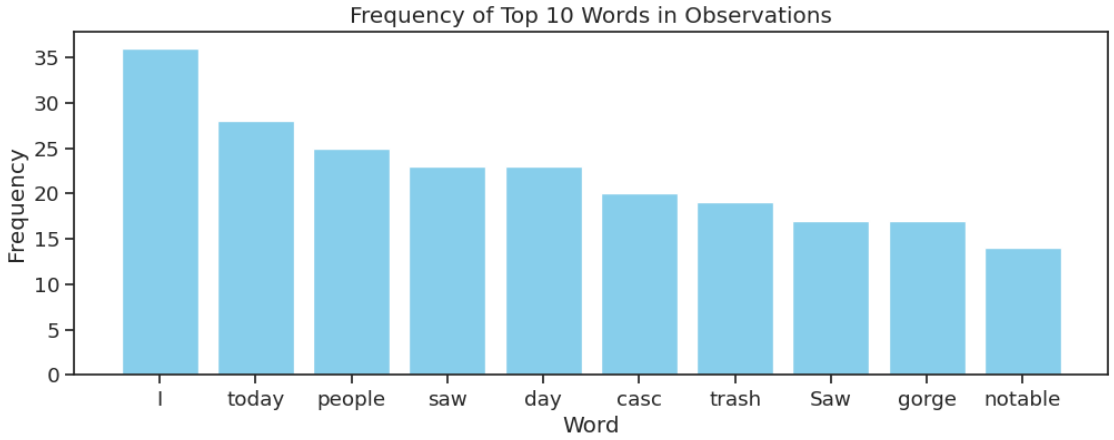
```
            del data[stopword]

    print("Number of total tokens in text:", sum(data.values()))
    print("Number of unique words in interactions:", len(data.values()))
    print("Top", n, "most frequent words in text:")
    print(data.most_common(n))

words = []
for description in interactions_df['Description of Interactions']:
    words += nltk.word_tokenize(description)
inter_counter = Counter(words)

words = []
for description in observations_df['Description of notable events']:
    words += nltk.word_tokenize(description)
obsv_counter = Counter(words)

print(word_stats(inter_counter))
print(word_stats(obsv_counter))
```

```
Number of total tokens in text: 2808
Number of unique words in interactions: 823
Top 20 most frequent words in text:
[('people', 103), ('casc', 103), ('trail', 63), ('2', 57), ('violations', 53),
('swimming', 45), ('I', 41), ('falls', 40), ('No', 37), ('one', 35), ('gorge',
35), ('asked', 33), ('leash', 33), ('today', 33), ('1', 27), ('dog', 27),
('bridge', 27), ('cascadilla', 27), ('water', 24), ('3', 23)]
None
Number of total tokens in text: 1834
Number of unique words in interactions: 923
Top 20 most frequent words in text:
[('I', 36), ('today', 28), ('people', 25), ('saw', 23), ('day', 23), ('casc',
20), ('trash', 19), ('Saw', 17), ('gorge', 17), ('notable', 14), ('trail', 13),
("n't", 13), ('got', 12), ('Beebe', 12), ('water', 12), ('nice', 12), ('lots',
11), ('many', 11), ('rain', 11), ('Nothing', 11)]
None
```

[ ]:
```
# visualization of the frequency of words in interactions

data = inter_counter.most_common(10)
words, values = zip(*data)
plt.figure(figsize=(12, 4))
plt.bar(words, values, color='skyblue')
plt.xlabel('Word')
plt.ylabel('Frequency')
plt.title('Frequency of Top 10 Words in Interactions')
plt.show()
```

```
data = obsv_counter.most_common(10)
words, values = zip(*data)
plt.figure(figsize=(12, 4))
plt.bar(words, values, color='skyblue')
plt.xlabel('Word')
plt.ylabel('Frequency')
plt.title('Frequency of Top 10 Words in Observations')
plt.show()
```





We find that among the most common words in both are, unsurprisingly, *people* and *casc* People is obvious, as it pertains the visitors, and casc refers to Cascidilla gorge. For violations, *violation* is amongst them, and so is *swimming*, which might elude to the most common type of violation. In observations, we see *trash* and *saw*, this makes sense as one sees with their eyes, and people tend to litter.

7 Sentiment Analysis

7.1 Methods

[26]:
```
# vader sentiment analyzer
vader = SentimentIntensityAnalyzer()

# method that tokenizes an input text
def tokenize_text(text) :
    return [word_tokenize(sent.lower()) for sent in sent_tokenize(text)]

# method to return sentiment scores of a sentence
def text_sentiment(analyzer, tokens, vader_lexicon):
    sentiment_dict = analyzer.polarity_scores(' '.join(tokens))
    compound_score = sentiment_dict['compound']

    if compound_score >= 0.05:
        overall_sentiment = 'pos'
    elif compound_score <= -0.05:
        overall_sentiment = 'neg'
    else:
        overall_sentiment = 'neu'

    matched_terms = {}
    for token in tokens:
        token_lower = token.lower()
        if token_lower in vader_lexicon:
            matched_terms[token_lower] = vader_lexicon[token_lower]

    sentiment_dict['overall sentiment'] = overall_sentiment
    sentiment_dict['matched terms'] = matched_terms

    return sentiment_dict

# method that pretty prints the sentiment scores of a sentiment list
def print_sentiment(sentiments_list):
  print("Total sentences:", len(sentiments_list))
  print("Total positive sentences:", sum([sent['overall sentiment'] == 'neg'␣
 ↪for sent in sentiments_list]))
  print("Total neutral sentences:", sum([sent['overall sentiment'] == 'neu' for␣
 ↪sent in sentiments_list]))
  print("Total negative sentences:", sum([sent['overall sentiment'] == 'pos'␣
 ↪for sent in sentiments_list]))
  print("Average sentiment:", sum([sent['compound'] for sent in␣
 ↪inter_sentiment]) / len(sentiments_list))
```

7.2 Interactions Sentiment

[30]:
```
# create a list of tokenized interactions
inter_tokens = []

for interaction in interactions:
  inter_tokens.extend(tokenize_text(interaction['msg']))

# print(inter_tokens)
# we remove "lower" as it has a negative sentiment score and biased the overall␣
 ↪sentiment
# inter_tokens = [(filter(lambda x: x != "lower", sentence)) for sentence in␣
 ↪inter_tokens]
# We also remove empty messages
# inter_tokens = [list(filter(lambda x: x != "", sentence)) for sentence in␣
 ↪inter_tokens]

inter_sentiment = [text_sentiment(vader, tokens, vader.lexicon) for tokens in␣
 ↪inter_tokens]

print_sentiment(inter_sentiment)

print("\nSentences with three or more matched terms:")
# print the most matched terms
iter_count = 0
for sent in inter_sentiment:
    if len(sent['matched terms']) > 1:
      if iter_count < 10:
        print(sent['overall sentiment'])
        print(sent['matched terms'])
        print('---')
        iter_count += 1
        continue
      else:
        break
```

```
Total sentences: 1172
Total positive sentences: 232
Total neutral sentences: 824
Total negative sentences: 116
Average sentiment: -0.05337491467576785

Sentences with three or more matched terms:
neu
{'want': 0.3, 'leave': -0.2}
---
pos
{'ignored': -1.3, 'thank': 1.5}
```

```
---
pos
{'super': 2.9, 'nice': 1.8}
---
pos
{'easiest': 1.8, 'worried': -1.2}
---
neg
{'barrier': -0.5, 'lower': -1.2}
---
pos
{'lower': -1.2, 'easy': 1.9}
---
neg
{'chilling': -0.1, 'lower': -1.2}
---
neg
{'no': -1.2, 'warning': -1.4}
---
neg
{'crying': -2.1, 'alone': -1.0}
---
pos
{'parties': 1.7, 'lower': -1.2}
---
```

Overall, we see that the overall sentiment is marginally negative. There are around double negative than there are positive, but far more neutral. This is somewhat expected - most violations likely contain negative words, such as *ignored*, *trash*, *crude*, *violation*, which we've seen in the log. However, many entries in interactions are positive; they have visitors cooperating and being kind. Overall, however, there are simply many more neutral entries, and the positive and negative weigh each other out.

7.3 Observations Sentiment

[23]:
```
# create a list of tokenized observations
obsv_tokens = []
for observation in observations:
  obsv_tokens.extend(tokenize_text(observation['msg']))

obsv_sentiment = [text_sentiment(vader, token, vader.lexicon) for token in␣
 ↪obsv_tokens]

print_sentiment(obsv_sentiment)

print("\nSentences with three or more matched terms:")
# print the most matched terms
for sent in obsv_sentiment:
```

```
    if len(sent['matched terms']) > 2:
        print(sent['overall sentiment'])
        print(sent['matched terms'])
        print('---')
```

```
Total sentences: 425
Total positive sentences: 47
Total neutral sentences: 298
Total negative sentences: 80
Average sentiment: -0.14718917647058805

Sentences with three or more matched terms:
pos
{'violations': -2.4, 'good': 1.9, 'nice': 1.8}
---
neg
{'surprised': 0.9, 'no': -1.2, 'violations': -2.4, 'number': 0.3}
---
pos
{'like': 1.5, 'great': 3.1, 'heron': 0.1}
---
pos
{'love': 3.2, 'easily': 1.4, 'violators': -1.9}
---
pos
{'rainy': -0.3, 'nice': 1.8, 'cool': 1.3}
---
pos
{'dead': -3.3, 'heroes': 2.3, 'save': 2.2}
---
```

There's a overall negative sentiment for observations entries. Among observations are not only violations, but also observations of weather, animals, and trash. We see words such as *trash*, *rainy*, *violators*, which certianly contribute to the negative rating. Again, there are many more neutral than either positive or negative.

7.4 Sentiment of Interactions Over The Years

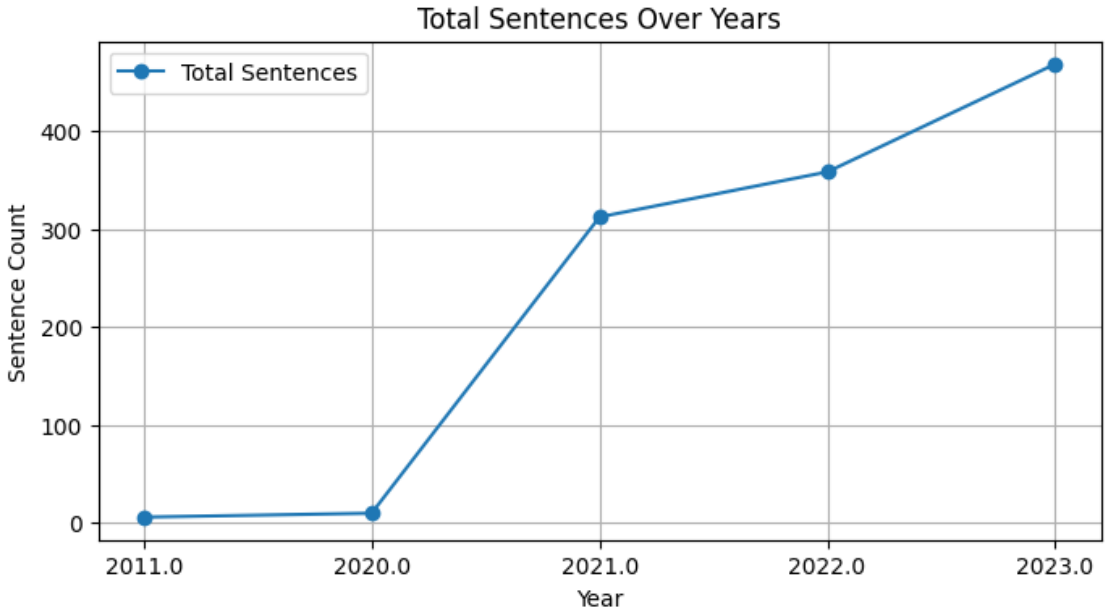
[31]:
```
inter_sent_date = []
for sent in interactions:
  token_sent = tokenize_text(sent['msg'])
  token_sent = (map(str, token_sent))
  token_sent = ' '.join(token_sent)

  sentiment = text_sentiment(vader, token_sent, vader.lexicon)
  content = {
      'date': sent['date'],
      'sentiment': sentiment
```

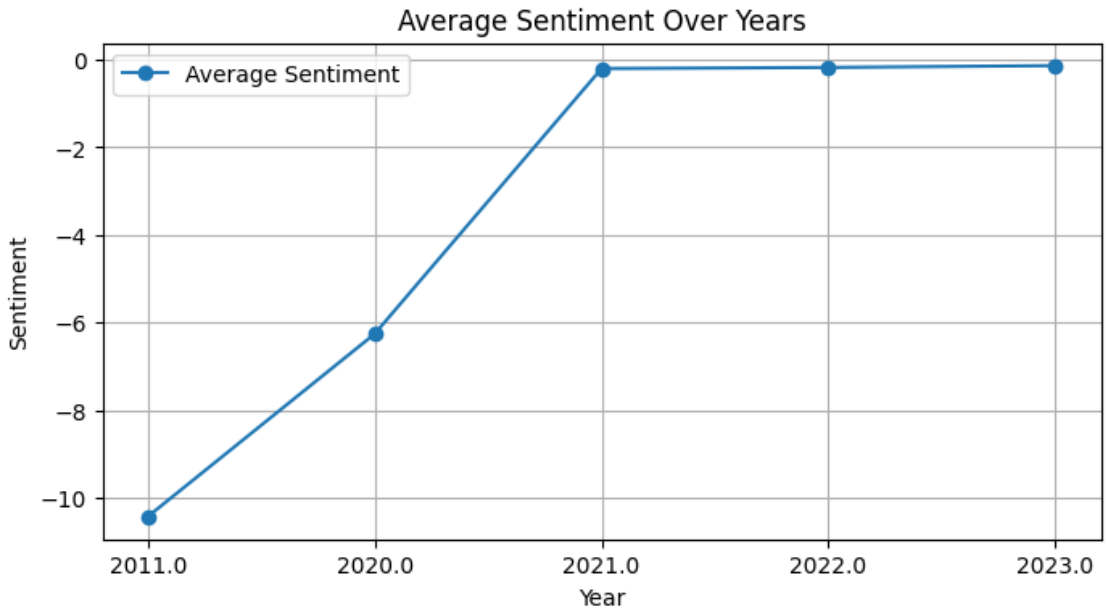
```
  }
  inter_sent_date.append(content)

inter_sent_date = pd.DataFrame(inter_sent_date)
inter_sent_date.date = pd.to_datetime(inter_sent_date.date).dt.year

# Calculate the average sentiment of interactions for each year
for key, group in inter_sent_date.groupby(inter_sent_date.date):
  print("Year", key)
  print_sentiment(group['sentiment'])
  print("---")
```

```
Year 2011.0
Total sentences: 6
Total positive sentences: 0
Total neutral sentences: 6
Total negative sentences: 0
Average sentiment: -10.425899999999986
---
Year 2020.0
Total sentences: 10
Total positive sentences: 0
Total neutral sentences: 10
Total negative sentences: 0
Average sentiment: -6.255539999999991
---
Year 2021.0
Total sentences: 312
Total positive sentences: 0
Total neutral sentences: 312
Total negative sentences: 0
Average sentiment: -0.20049807692307664
---
Year 2022.0
Total sentences: 358
Total positive sentences: 0
Total neutral sentences: 358
Total negative sentences: 0
Average sentiment: -0.1747357541899439
---
Year 2023.0
Total sentences: 468
Total positive sentences: 0
Total neutral sentences: 468
Total negative sentences: 0
Average sentiment: -0.13366538461538444
---
```

[32]:
```
# hard coded output of previous cell
sent_year_output = """Year 2011.0
Total sentences: 6
Total positive sentences: 0
Total neutral sentences: 6
Total negative sentences: 0
Average sentiment: -10.425899999999986
---
Year 2020.0
Total sentences: 10
Total positive sentences: 0
Total neutral sentences: 10
Total negative sentences: 0
Average sentiment: -6.255539999999991
---
Year 2021.0
Total sentences: 312
Total positive sentences: 0
Total neutral sentences: 312
Total negative sentences: 0
Average sentiment: -0.20049807692307664
---
Year 2022.0
Total sentences: 358
Total positive sentences: 0
Total neutral sentences: 358
Total negative sentences: 0
Average sentiment: -0.1747357541899439
---
Year 2023.0
Total sentences: 468
Total positive sentences: 0
Total neutral sentences: 468
Total negative sentences: 0
Average sentiment: -0.13366538461538444
---
"""

# Extracting data
years = re.findall(r'Year (\d+\.\d+)', sent_year_output)
total_sentences = list(map(int, re.findall(r'Total sentences: (\d+)',␣
 ↪sent_year_output)))
average_sentiments = list(map(float, re.findall(r'Average sentiment: (-?\d+\.
 ↪\d+)', sent_year_output)))

# Plotting
plt.figure(figsize=(8, 4))
```

```
plt.plot(years, total_sentences, label='Total Sentences', marker='o')
plt.xlabel('Year')
plt.ylabel('Sentence Count')
plt.title('Total Sentences Over Years')
plt.legend()
plt.grid(True)
plt.show()

# Plotting
plt.figure(figsize=(8, 4))
plt.plot(years, average_sentiments, label='Average Sentiment', marker='o')
plt.xlabel('Year')
plt.ylabel('Sentiment')
plt.title('Average Sentiment Over Years')
plt.legend()
plt.grid(True)
plt.show()
```





Over the years, we see a sharp increase of sentiment, from -10 to practically neutral in 2021 and therein. This matches with the boost in entires in 2021; it is likely that the extra entries are netural, thus lowering the sentiment score. The sentiment stays negative, the enture duration, however.

7.5 Sentiment Discussion

Based off the overall neutral sentiment in observations and interactions, it's clear that the entires are seemingly matter of fact and few interactions are radical. While there are particular entries that stand out as interesting (i.e. mentions of rude gestures, random acts of kindness), and the negatives tend to outweigh the positive twicefold, the overwhelming amount of entries are basic descriptions of events.

8 Euclidean Differences

[42]:
```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', norm='l2',␣
 ↪use_idf=False)

# extract out only the messages of log
inter_obsv_tokens = [inter_tokens, obsv_tokens]

# features = vectorizer.fit_transform(inter_obsv_tokens)
features = tfidf_vectorizer.fit_transform([str(x) for x in inter_obsv_tokens])


print('Matrix shape:', features.shape)
# print('Feature labels:', vectorizer.get_feature_names_out())
```

```
# print('Stopwords used:', vectorizer.stop_words_, '\n')
# print('First line vector:', features[0].toarray(), '\n')

# print('Unique words:', list(set(log[0].split())))

euclidean_dist = euclidean_distances(features)
cosine_dist = cosine_distances(features)

print("\nEuclidean Distances (Normalized, TF-IDF):")
print(euclidean_dist)
print("\nCosine Distances (Normalized, TF-IDF):")
print(cosine_dist)
```

```
Matrix shape: (2, 1157)

Euclidean Distances (Normalized, TF-IDF):
[[0.         0.65734335]
 [0.65734335 0.        ]]

Cosine Distances (Normalized, TF-IDF):
[[0.         0.21605014]
 [0.21605014 0.        ]]
```

8.1 Discussion

The TF-IDF matrix has a shape of (2, 1157), indicating that there are 2 types, onbservations and interactions, and 1157 features, which are tokens representing the entries. After normalization, the euclidean distances matrix shows that difference between the two types is 0.756. The cosine difference between the two is low, at 0.293. These two numbers indicate that interactions and observations are considerable similar in textual content and high degree of similiarity based on their TF-IDF representations.

Due to their similarity and for simplicity of this project, the work will mainly focus on violations herein; violations are more interesting and have more potential for recommendations for change in the future.

9 Unsupervised Clustering Labelling Types of Interactions

We now conduct unsupervised clustering on interactions, violations, and observations, with a variety of methods.

9.1 Methods

[40]:
```
def plot_compare(X, labels, title, alpha=0.2):
    '''
    Takes an array of object data, a list of cluster labels, a title string,␣
 ↪and an optional alpha value.
```

```
    Reduces dimensions to 2 if necessary and plots the clustering with and␣
 ↪without coloring by label.
    Returns nothing.
    '''
    if X.shape[1] > 2:
        svd = TruncatedSVD(n_components=2)
        X_reduced = svd.fit_transform(X)
    else:
        X_reduced = X

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    ax1.scatter(X_reduced[:, 0], X_reduced[:, 1], alpha=alpha)
    ax1.set_title('Unclustered Data')

    ax2.scatter(X_reduced[:, 0], X_reduced[:, 1], c=labels, cmap='viridis',␣
 ↪alpha=alpha)
    ax2.set_title('Clustered Data')

    fig.suptitle(title)

    plt.show()

# Pull sample texts from each label set
def pull_samples(texts, labels, n=3):
    unique_labels = np.unique(labels)

    for label in unique_labels:
        label_indices = np.where(labels == label)[0]

        print("Label:", label)
        print("Number of texts in this cluster:", len(label_indices))

        if len(label_indices) == 0:
            print("No texts in this cluster.")
        else:
            sampled_indices = np.random.choice(label_indices, size=min(n,␣
 ↪len(label_indices)), replace=False)

            for i, index in enumerate(sampled_indices):
                print('\033[1m' + "Sample text", i + 1, "(Index:", index, "):"␣
 ↪+ '\033[0m', texts[index])

        print("----------------------------")
```

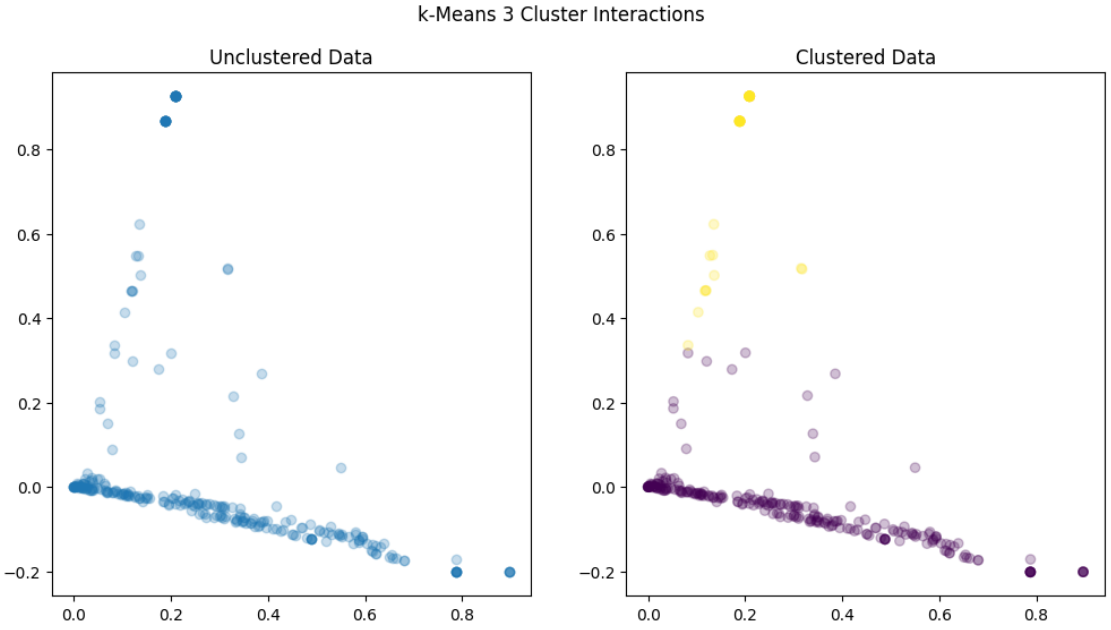
9.2 Interactions

9.2.1 K-Means

K-means clustering partitions the dataset into a set of distinct, non-overlapping groups or clusters.

[43]:
```
# k-Means clustering with n_clusters = 2
inter = []
for interaction in interactions_unsplit:
  inter.append(interaction['msg'])

inter_tokens = tfidf_vectorizer.fit_transform(inter)


inter_pred_2 = KMeans(n_clusters=2, n_init='auto').fit_predict(inter_tokens)
print("Shape of clustered interactions:", inter_pred_2.shape)
plot_compare(inter_tokens, inter_pred_2, "k-Means 3 Cluster Interactions",␣
 ↪alpha = 0.25)

pull_samples(inter, inter_pred_2, 10)
```

```
Shape of clustered interactions: (285,)
```




```
Label: 0
Number of texts in this cluster: 240
Sample text 1 (Index: 3 ): Lots of people looking for arboretum, three
dogs off leash on Bebee Lake trail and Cascadilla Trail. 2 people past trail in
Hemlock Gorge
```

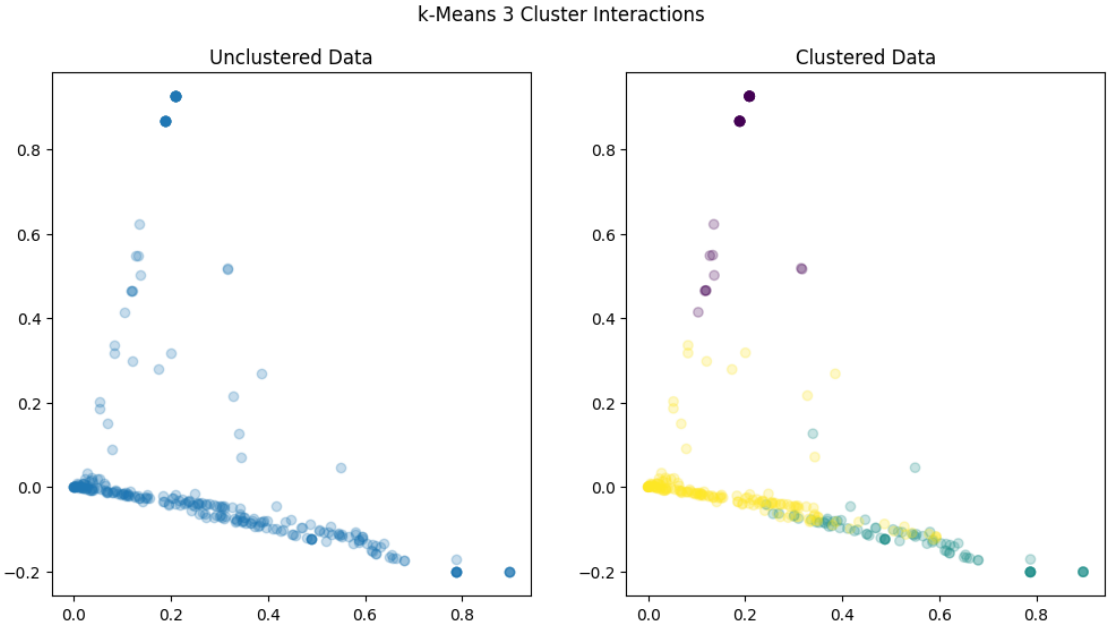
```
Sample text 2 (Index: 154 ): all people off trail in casc :)
Sample text 3 (Index: 61 ): everyone loves that rock in casc
Sample text 4 (Index: 221 ): Talked to some women with dogs in the gorge
behind Risley; lots of studentsswiming in hemlock gorge area; family of 4 trying
to hop over fence to get to lower cascadilla
Sample text 5 (Index: 204 ): 2 people below suspension bridge above
horseshoe falls, 3 people on Cascadilla Gorge trail closed section
Sample text 6 (Index: 201 ): 2 people above ithaca falls, 4 under the
suspension bridge on fall creek
Sample text 7 (Index: 40 ): Four people off trail is cascadilla, two at
the bottom, 2 at the top
Sample text 8 (Index: 2 ): few people suntanning on rocks in the water
Sample text 9 (Index: 33 ): person taking pick below suspension bridge.
person fishing above bebe lake goarge.
Sample text 10 (Index: 210 ): Family of five past the signs in upper
Hemlock Gorge
----------------------------
Label: 1
Number of texts in this cluster: 45
Sample text 1 (Index: 234 ): No violations today; lots of people walking
around Bebe
Sample text 2 (Index: 208 ): No violations today.
Sample text 3 (Index: 235 ): No violations today, very quiet and once
the rain came it emptied out.
Sample text 4 (Index: 281 ): no violations
Sample text 5 (Index: 238 ): No violations today.
Sample text 6 (Index: 241 ): no violations
Sample text 7 (Index: 262 ): No violations today
Sample text 8 (Index: 209 ): No violations today.
Sample text 9 (Index: 243 ): No violations; no violations
Sample text 10 (Index: 215 ): No violations today.
----------------------------
```

With a k-means clustering algorith, we get two labels. Looking at the sample texts, it seems to show that cluster 1 are violations, and cluster 2 are no violations.

[48]:
```
# k-Means clustering with n_clusters = 3
inter = []
for interaction in interactions_unsplit:
  inter.append(interaction['msg'])

inter_vectors = tfidf_vectorizer.fit_transform(inter)


inter_pred_3 = KMeans(n_clusters=3, random_state=123).fit_predict(inter_vectors)
print("Shape of clustered interactions:", inter_pred_3.shape)
plot_compare(inter_tokens, inter_pred_3, "k-Means 3 Cluster Interactions",␣
 ↪alpha = 0.25)
```

```
pull_samples(inter, inter_pred_3, 15)
```

```
Shape of clustered interactions: (285,)
```




```
Label: 0
Number of texts in this cluster: 44
Sample text 1 (Index: 211 ): No violations today.
Sample text 2 (Index: 238 ): No violations today.
Sample text 3 (Index: 234 ): No violations today; lots of people walking
around Bebe
Sample text 4 (Index: 225 ): No violations today.
Sample text 5 (Index: 215 ): No violations today.
Sample text 6 (Index: 280 ): no violations
Sample text 7 (Index: 212 ): No violations today.
Sample text 8 (Index: 235 ): No violations today, very quiet and once
the rain came it emptied out.
Sample text 9 (Index: 266 ): No violations spotted
Sample text 10 (Index: 245 ): No violations. Gave several families
directions.; Weather was a bit too cold for the violators today
Sample text 11 (Index: 262 ): No violations today
Sample text 12 (Index: 270 ): no violations
Sample text 13 (Index: 274 ): no violations
Sample text 14 (Index: 251 ): No violations today
Sample text 15 (Index: 222 ): No violations today.
----------------------------
Label: 1
```

Number of texts in this cluster: 79
Sample text 1 (Index: 154): all people off trail in casc :)
Sample text 2 (Index: 172): one person of trail in casc
Sample text 3 (Index: 134): all off trails in casc
Sample text 4 (Index: 31): 2 people outside of casc fenced area warned, 1 family swimming in casc given alternative swim locations, gave directions to the same 2 people multiple times throughout my shift and gave my opinion on the best trails/views
Sample text 5 (Index: 113): mostly children off trail in casc
Sample text 6 (Index: 30): family swimming in casc, told them where else they could go. walked up casc with a man asking about what we do and where to hike
Sample text 7 (Index: 174): lots off different groups of trail in casc
Sample text 8 (Index: 46): All people in restricted casc areas
Sample text 9 (Index: 38): asked guy reading on rocks past fence in casc to move
Sample text 10 (Index: 15): 2 swimmers above beebe, 2 people picnicking in casc amphitheater area, various picture takers
Sample text 11 (Index: 44): people beyond fence in casc
Sample text 12 (Index: 47): People in the big bend ~3/4 down casc, dog off leash
Sample text 13 (Index: 60): someone looked like they were about to go beyond railing in casc gorge
Sample text 14 (Index: 168): all off trail in casc
Sample text 15 (Index: 159): just people off trail in casc

Label: 2
Number of texts in this cluster: 162
Sample text 1 (Index: 28): no interactions today
Sample text 2 (Index: 126): 2 people wading at Fall creek gorge
Sample text 3 (Index: 1): lots of commencement direction questions
Sample text 4 (Index: 227): Two dogs off leash. One at upper cascadilla and one on Beebe Lake Trail
Sample text 5 (Index: 188): 3 dogs off leash, 1 person in the water of Hemlock Gorge, 2 people off trail
Sample text 6 (Index: 111): Some lingering reunion people
Sample text 7 (Index: 62): water was really flowing which inspired attempts to climb waterfalls for some reason
Sample text 8 (Index: 85): Saw 2 guys in the water from the top of Thurston Ave bridge, whistled but they didn't look. Walked down there and met them walking back up. I told them they couldn't be in the water but had some language barrier. They explained that they were collecting tennis balls that had rolled down there and had slipped and fallen in. I said that was why they couldn't be in the water but I don't think they understood. Gave 1 guy directions
Sample text 9 (Index: 40): Four people off trail is cascadilla, two at the bottom, 2 at the top
Sample text 10 (Index: 26): one person on rocks essentially directly

```
below suspension bridge
Sample text 11 (Index: 185 ): 1 dog off leash and 4 people wandering off
trail
Sample text 12 (Index: 232 ): First time I've seen someone climbing over
the lower gate in the clsoed section of Cascadilla.
Sample text 13 (Index: 112 ): many prospective students and visitors;
told story of how girl fell into cascadilla but landed on racoon and both she
and racoon lived sometime in the 80s
Sample text 14 (Index: 95 ): none
Sample text 15 (Index: 4 ): Big day for tourists, small day for
violators
----------------------------
```

With an additional third cluster, we now have a category that seems to be violations in cascadilla. Therefore:
Cluster 1: No Violation
CLuster 2: Cascadilla-related violation
Cluster 2: Violation

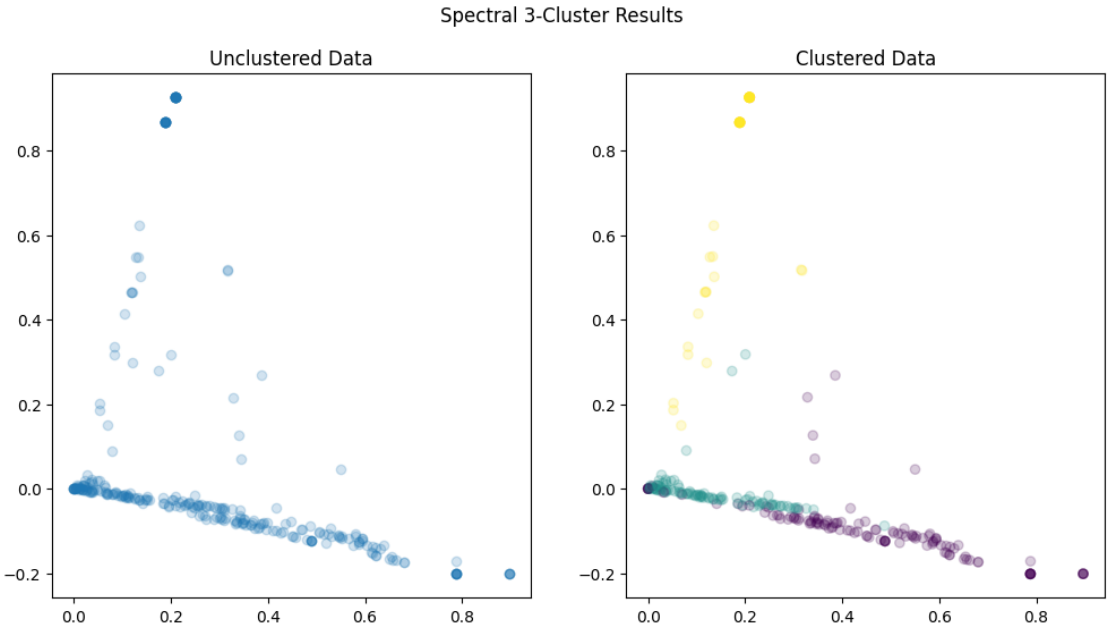
9.2.2 Spectral Clustering

Spectral clustering bases its decisions on the spectral (eigenvalue) properties of a similarity matrix. It makes less-linear decisions than k-means and may be more nuanced.

[49]:
```
spectral_clustering_3 = SpectralClustering(n_clusters=3, affinity='precomputed')

cosine_similarity_matrix = cosine_similarity(inter_vectors)

inter_spectral_3 = spectral_clustering_3.fit_predict(cosine_similarity_matrix)

print("Spectral Clustering Labels Shape:", inter_spectral_3.shape)

plot_compare(inter_vectors, inter_spectral_3, "Spectral 3-Cluster Results")

pull_samples(inter, inter_spectral_3, n=10)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

```
Spectral Clustering Labels Shape: (285,)
```



Label: 0
Number of texts in this cluster: 128
Sample text 1 (Index: 151): people off trail in casc
Sample text 2 (Index: 97): 3 people swimming in casc, one kid running around and swimming in casc, 4 people swimming in casc, one guy asked me if swimming was allowed in casc and I told him alternative options, 10 people drinking and swimming at the swimming hole upstream of the tennis courts, one dude fishing at hemlock gorge
Sample text 3 (Index: 100): All violations were around 12-1 when it was still hot before the storms. 2 people wading in casc, 2 more people wading in casc, 2 people in spot upstream of tennis courts. I had some laughs with people while we were all drenched during the storms
Sample text 4 (Index: 138): all off trail, swimming, etc. in Cascadilla, plus one above the tennis courts
Sample text 5 (Index: 184): not really any violations every though there were tons of people, I think most people were just taking photos and stuff
Sample text 6 (Index: 76): 1 guy fishing on top of falls by Beebe lake, said he wasn't from the area and didn't know. 2 Saw 2 girls on top of falls in Casc, attempting to walk down the side, blew whistle and gestured them out. They were gone by the time I got to the top of the stairs
Sample text 7 (Index: 13): fly fisher in Fall Creek (by Risley); dipping toes in prohibited part of casc; 2 swimmers in beebe; 3 people under suspension bridge
Sample text 8 (Index: 140): fishing in cascadilla, kids on falls, couple people off trail
Sample text 9 (Index: 159): just people off trail in casc
Sample text 10 (Index: 23): had to ask guy twice to move in casc- he

```
was rude, church group tried to climb falls in casc- they were nice
----------------------------
Label: 1
Number of texts in this cluster: 107
Sample text 1 (Index: 85 ): Saw 2 guys in the water from the top of
Thurston Ave bridge, whistled but they didn't look. Walked down there and met
them walking back up. I told them they couldn't be in the water but had some
language barrier. They explained that they were collecting tennis balls that had
rolled down there and had slipped and fallen in. I said that was why they
couldn't be in the water but I don't think they understood. Gave 1 guy
directions
Sample text 2 (Index: 277 ): A group of 4 swimming in Risley gorge
Sample text 3 (Index: 127 ): guy fishing at hemlock gorge
Sample text 4 (Index: 87 ): None, really.
Sample text 5 (Index: 232 ): First time I've seen someone climbing over
the lower gate in the clsoed section of Cascadilla.
Sample text 6 (Index: 0 ): Someone asked about north campus
Sample text 7 (Index: 119 ): 5 people swimming in Cascadilla, left
without complaint
Sample text 8 (Index: 200 ): Under suspension bridge - came across two
people past the sign taking off their shoes!
Sample text 9 (Index: 230 ): Took a photo for a couple who graduated in
1999; One person treading water in Hemlock Gorge, swimming into the strong
current for exercise? Ignored requests to get out, called CUPD
Sample text 10 (Index: 210 ): Family of five past the signs in upper
Hemlock Gorge
----------------------------
Label: 2
Number of texts in this cluster: 50
Sample text 1 (Index: 234 ): No violations today; lots of people walking
around Bebe
Sample text 2 (Index: 279 ): no violations
Sample text 3 (Index: 272 ): no violations
Sample text 4 (Index: 207 ): No violations today.
Sample text 5 (Index: 257 ): no violations. Helped a few families with
directions
Sample text 6 (Index: 208 ): No violations today.
Sample text 7 (Index: 251 ): No violations today
Sample text 8 (Index: 206 ): No violations today.
Sample text 9 (Index: 215 ): No violations today.
Sample text 10 (Index: 212 ): No violations today.
----------------------------
```

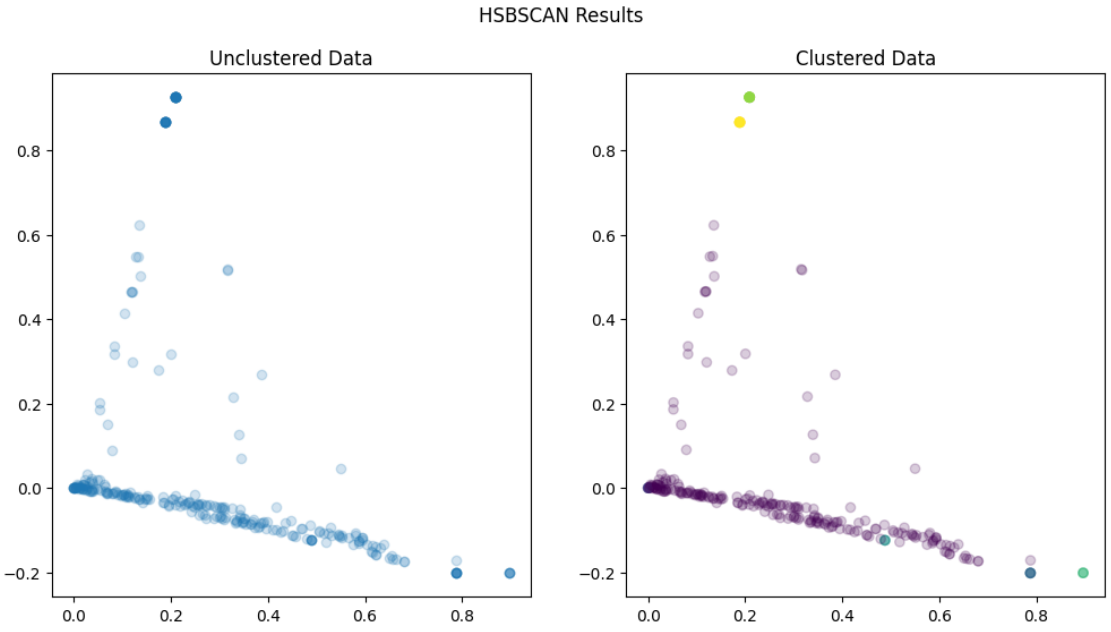
We see that spectral clustering just about produces the same result.

9.2.3 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN), does not need a predefined cluster number and instead handles varied shapes best, making its own decisions.

```
[ ]: inter_hdv_pred = DBSCAN().fit_predict(inter_vectors.toarray())

print("HDBSCAN Clustering Labels Shape:", inter_hdv_pred.shape)

plot_compare(inter_vectors, inter_hdv_pred, "HSBSCAN Results")

pull_samples(inter, inter_hdv_pred, n=5)
```

```
HDBSCAN Clustering Labels Shape: (285,)
```




```
Label: -1
Number of texts in this cluster: 226
Sample text 1 (Index: 82 ): Directions near bebe again
Sample text 2 (Index: 99 ): 9 wading or dog off leash in casc, 2 in
process of hopping fence in fall creek picnic area as I went by. All complied
amicably. Snake swimming upriver in casc
Sample text 3 (Index: 210 ): Family of five past the signs in upper
Hemlock Gorge
Sample text 4 (Index: 65 ): gave someone directions in between
suspension and stewart
Sample text 5 (Index: 147 ): 2 people with off leash dog under bridge by
tenis court, rest off trail in casc
----------------------------
```

```
Label: 0
Number of texts in this cluster: 7
Sample text 1 (Index: 155 ): none
Sample text 2 (Index: 96 ): none
Sample text 3 (Index: 93 ): none
Sample text 4 (Index: 71 ): none
Sample text 5 (Index: 95 ): none
----------------------------
Label: 1
Number of texts in this cluster: 8
Sample text 1 (Index: 142 ): off trail in casc
Sample text 2 (Index: 150 ): all off trail in casc
Sample text 3 (Index: 139 ): off trail in casc
Sample text 4 (Index: 161 ): all off trail in casc
Sample text 5 (Index: 156 ): off trail in casc
----------------------------
Label: 2
Number of texts in this cluster: 5
Sample text 1 (Index: 137 ): all off trails in casc
Sample text 2 (Index: 135 ): all off trails in casc
Sample text 3 (Index: 136 ): all off trails in casc
Sample text 4 (Index: 134 ): all off trails in casc
Sample text 5 (Index: 132 ): all off trails in casc
----------------------------
Label: 3
Number of texts in this cluster: 5
Sample text 1 (Index: 154 ): all people off trail in casc :)
Sample text 2 (Index: 179 ): People of trail in casc
Sample text 3 (Index: 164 ): some people of trail in casc
Sample text 4 (Index: 165 ): people off trail in casc
Sample text 5 (Index: 151 ): people off trail in casc
----------------------------
Label: 4
Number of texts in this cluster: 19
Sample text 1 (Index: 259 ): No violations today
Sample text 2 (Index: 206 ): No violations today.
Sample text 3 (Index: 219 ): No violations today.
Sample text 4 (Index: 222 ): No violations today.
Sample text 5 (Index: 263 ): No violations today
----------------------------
Label: 5
Number of texts in this cluster: 15
Sample text 1 (Index: 268 ): no violations
Sample text 2 (Index: 278 ): no violations
Sample text 3 (Index: 240 ): No violations.
Sample text 4 (Index: 280 ): no violations
Sample text 5 (Index: 270 ): no violations
----------------------------
```

9.2.4 Findings

Overall, we see that k-means with three clusters is probably best. Whereas 2 clusters only differentiates between violation/non-violation, the third cluster can also represent interactions held in cascadilla. The spectral clustering provides just about the same results, and DBSCAN seems to overfit, where many labels have only a few entries in them.

Overall, the three-cluster k-means has sizes of 44 non-violations, 79 violations, 162 violations in cascadilla.

9.3 Violations

Based of the clustering from k-means-3, we affix a label that indicates if we think the interaction is a violation or not.

We are ultimately curious to see if we can discern if there are different types of violations within the label.

9.3.1 Create Violations Label

[50]:
```
# add the cluster number as a dict key pair in interactions
for i, cluster in enumerate(inter_pred_3):
  if cluster == 0:
    interactions_unsplit[i]['violation_label'] = False
  else:
    interactions_unsplit[i]['violation_label'] = True
  if cluster == 1:
    interactions_unsplit[i]['casc_label'] = True
  else:
    interactions_unsplit[i]['casc_label'] = False

violation_count = 0
print("Violations sample:")
for interaction in interactions_unsplit:
  if interaction['violation_label'] == True:
    violation_count += 1
    if violation_count <= 5:
      print(interaction['msg'])

print('---')
no_vio_count = 0
print("No violations sample:")
for interaction in interactions_unsplit:
  if interaction['violation_label'] == False:
    no_vio_count += 1
    if no_vio_count <= 5:
     print(interaction['msg'])

print('---')
```

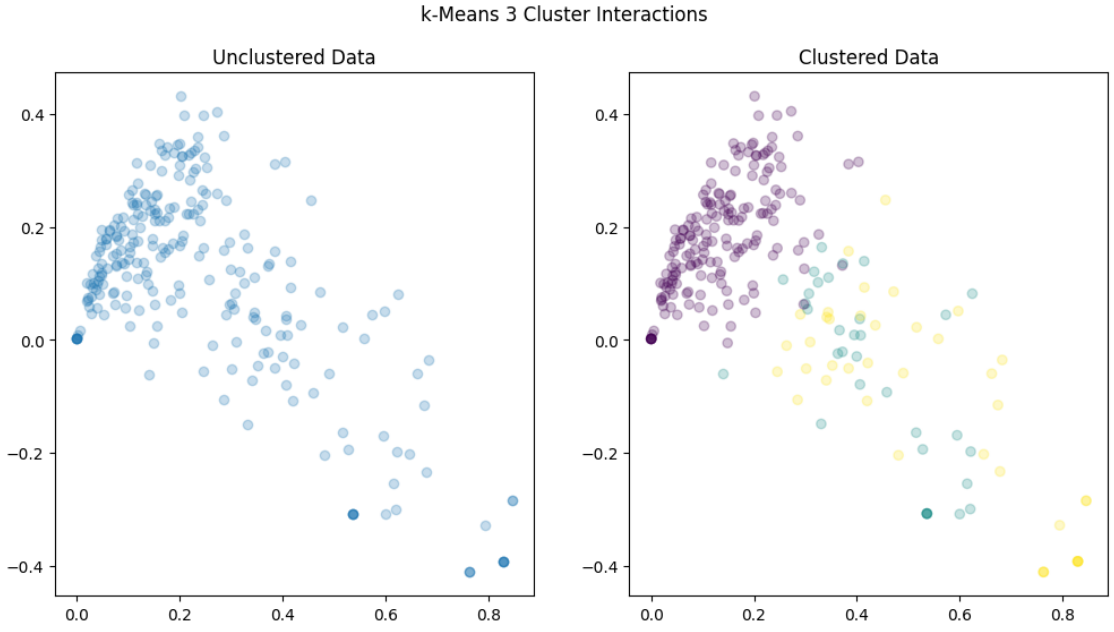
```
casc_count = 0
print("Casc sample:")
for interaction in interactions_unsplit:
  if interaction['casc_label'] == True:
    casc_count += 1
    if casc_count <= 5:
      print(interaction['msg'])

print('---')
print("Number of violations:", violation_count)
print("Number of no violations:", no_vio_count)
print("Number of casc violations:", casc_count)
```

```
Violations sample:
Someone asked about north campus
lots of commencement direction questions
few people suntanning on rocks in the water
Lots of people looking for arboretum, three dogs off leash on Bebee Lake trail
and Cascadilla Trail. 2 people past trail in Hemlock Gorge
Big day for tourists, small day for violators
---
No violations sample:
No violations today.
No violations today.
No violations today.
No violations today.
No violations today.
---
Casc sample:
1 dog off leash on upper casc
Four guys taking photos in lower Casc
mentally disabled woman did not want to leave spot in casc, took two trys, lady
with dog left when asked also in casc
2 swimmers above beebe, 2 people picnicking in casc amphitheater area, various
picture takers
3 seperate groups taking pictures standing in the water falls in casc, two boys
taking picutures on planet b signs- had to treaten to call the police, couple of
people needed less steep alternatives to casc
---
Number of violations: 241
Number of no violations: 44
Number of casc violations: 79
```

9.3.2 K-Means

[ ]:
```
violations = []

for interaction in interactions_unsplit:
  if interaction['violation_label'] == True:
    violations.append(interaction)

print(len(violations))

vio = []
for violation in violations:
  vio.append(violation['msg'])

vio_tokens = tfidf_vectorizer.fit_transform(vio)

# k-means 3 cluster
vio_pred_3 = KMeans(n_clusters=3, n_init='auto').fit_predict(vio_tokens)
print("Shape of clustered interactions:", vio_pred_3.shape)
plot_compare(vio_tokens, vio_pred_3, "k-Means 3 Cluster Interactions", alpha =␣
↪0.25)

pull_samples(vio, vio_pred_3, 10)
```

```
249
Shape of clustered interactions: (249,)
```




```
Label: 0
```

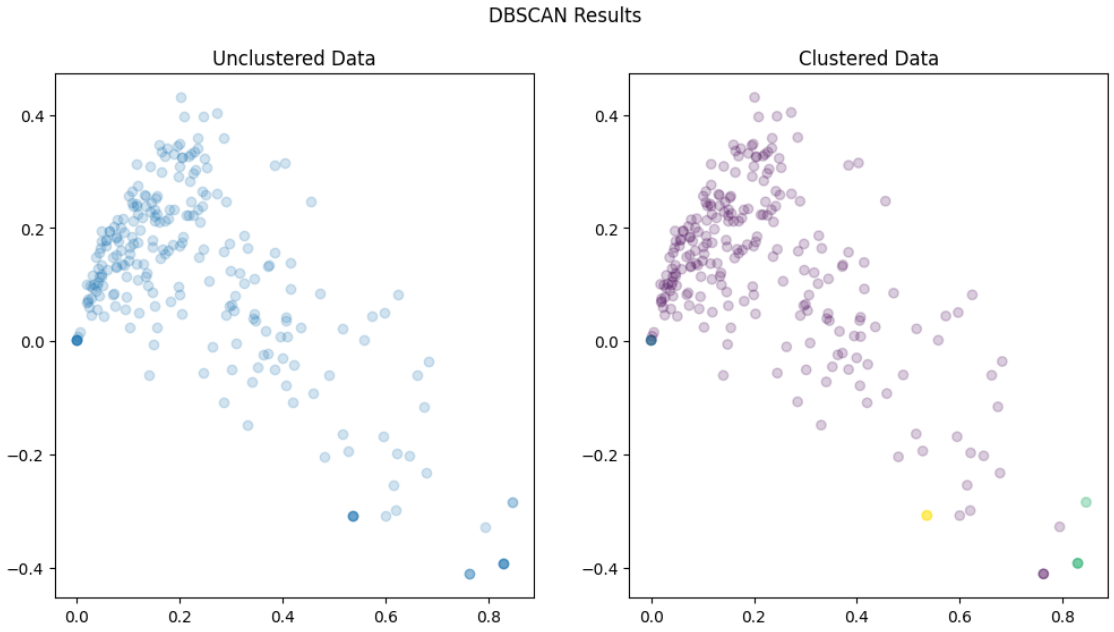
Number of texts in this cluster: 173
Sample text 1 (Index: 9): People trying to swim in the base of ithaca falls area, child leaning over the edge of that pool part, guy fishing at end of hemlock from atop waterfall. Lots of tourists and families. Lot of people who wanted to swim in weird places.
Sample text 2 (Index: 104): Literally no one in the gorges, kinda strange For reference counted under 15 cascadilla all day
Sample text 3 (Index: 114): got to see my parents for a little bit at lunch! No problems today at ithaca falls
Sample text 4 (Index: 26): one person on rocks essentially directly below suspension bridge
Sample text 5 (Index: 203): 2 people below suspension bridge above horseshoe falls, 3 people on Cascadilla Gorge trail closed section
Sample text 6 (Index: 209): 3 students swimming in the hemlock gorge area
Sample text 7 (Index: 205): Family of five past the signs in upper Hemlock Gorge
Sample text 8 (Index: 73): Person under the suspension bridge, people in restricted parts of casc
Sample text 9 (Index: 159): no major interactions, only about an hour of no rain
Sample text 10 (Index: 80): More and more directions, lots of peopel around bb lake looking for dorms and what not
---
Label: 1
Number of texts in this cluster: 35
Sample text 1 (Index: 176): dogs of leash in casc
Sample text 2 (Index: 133): all off trails in casc
Sample text 3 (Index: 179): 4 dogs off leash
Sample text 4 (Index: 131): all off trails in casc
Sample text 5 (Index: 140): one dog off leash, rest off trail in casc
Sample text 6 (Index: 51): Three guys trying to get in the water at lower casc. One dog off leash at Beebe
Sample text 7 (Index: 136): all off trails in casc
Sample text 8 (Index: 135): all off trails in casc
Sample text 9 (Index: 152): Dogs off leashes around beebee
Sample text 10 (Index: 146): 2 people with off leash dog under bridge by tenis court, rest off trail in casc
---
Label: 2
Number of texts in this cluster: 41
Sample text 1 (Index: 122): people off trail in cascadilla, 4 people swimming by ezra's tunnel
Sample text 2 (Index: 121): people off trail in cascadilla
Sample text 3 (Index: 148): all off campus in casc gorge - mostly kids and photographers
Sample text 4 (Index: 178): People of trail in casc
Sample text 5 (Index: 72): mostly people past fence in casc, also 2

trail bike riders
Sample text 6 (Index: 117): off trail in casc
Sample text 7 (Index: 145): 1 person off trail in casc
Sample text 8 (Index: 3): Lots of people looking for arboretum, three dogs off leash on Bebee Lake trail and Cascadilla Trail. 2 people past trail in Hemlock Gorge
Sample text 9 (Index: 171): one person of trail in casc
Sample text 10 (Index: 101): people off trail in casc, low number of interactions but tons of people in a schol group. Hungout with some ducklings!

9.3.3 DBSCAN

[ ]:
```
vio_tokens = tfidf_vectorizer.fit_transform(vio)

vio_hdv_pred = DBSCAN().fit_predict(vio_tokens.toarray())

print("DBSCAN Clustering Labels Shape:", vio_hdv_pred.shape)

plot_compare(vio_tokens, vio_hdv_pred, "DBSCAN Results")

pull_samples(vio, vio_hdv_pred, n=5)
```

DBSCAN Clustering Labels Shape: (249,)



Label: -1
Number of texts in this cluster: 230
Sample text 1 (Index: 55): Big, big day for violations. Lots of people

bringing their very young kids (<10yo) swimming in the gorges
1. [LOWER CASC] French tourists dipping toes in water too close to smaller falls (behind a railing); told them where they could do that safely a little further down where there was no gate
2. [LOWER CASC] Two ppl climbed behind a fence in casc; too close to fall; asked them to stay on the right side of the railing
3. [LOWER CASC] Fam swimming next to the second big fall; provided alternate swimming locations
4. [LOWER CASC] Fam swimming by first big fall (like right under it), let them know it was dangerous + provided alternate swimming locations
5. [LOWER CASC] Someone sitting right at the top of one of the bigger falls drinking coffee; asked them to sit elsewhere
6. [LOWER CASC] dad and two toddlers swimming under the first big waterfall, provided alt swim locations (dad was kind of dismissive; one of his kids went I TOLD YOUUUU after I let dad know it wasn't safe/allowed; child and I bonded over our love of frozen)
7. [LOWER CASC] 2 ppl sitting behind a railing/fence too close to fall, asked them to move to an unrailinged place
8. [LOWER CASC] chatted with some bikers doing a waterfall tour, recommended buttermilk
9. [FALL CREEK] couple ppl asked for directions
10. [FALL CREEK] 12 people trespassing under risley bridge; 5 (one fam) were just sitting on rocks, very nice; two fam (alumni) + their kids swimming under (including directly under) horseshoe falls, had to whistle at them, asked them to leave, alt swim locations
Sample text 2 (Index: 52): Many adults roaming the gorges in search of directons, one dog off leash on Beebe trail
Sample text 3 (Index: 76): people standing at the top of or under waterfalls in casc; one person fishing in beebe next to a no fishing sign, provided alternatives
Sample text 4 (Index: 169): photographers in casc
Sample text 5 (Index: 210): Talked to some people who wanted to go down into Cascadilla Gorge about repairs
---
Label: 0
Number of texts in this cluster: 7
Sample text 1 (Index: 70): none
Sample text 2 (Index: 95): none
Sample text 3 (Index: 92): none
Sample text 4 (Index: 154): none
Sample text 5 (Index: 93): none

Label: 1
Number of texts in this cluster: 7
Sample text 1 (Index: 165): off trail in casc
Sample text 2 (Index: 164): people off trail in casc
Sample text 3 (Index: 155): off trail in casc
Sample text 4 (Index: 141): off trail in casc

```
Sample text 5 (Index: 150 ): people off trail in casc
----------------------------
Label: 2
Number of texts in this cluster: 5
Sample text 1 (Index: 135 ): all off trails in casc
Sample text 2 (Index: 136 ): all off trails in casc
Sample text 3 (Index: 131 ): all off trails in casc
Sample text 4 (Index: 134 ): all off trails in casc
Sample text 5 (Index: 133 ): all off trails in casc
----------------------------
```

9.3.4 Findings

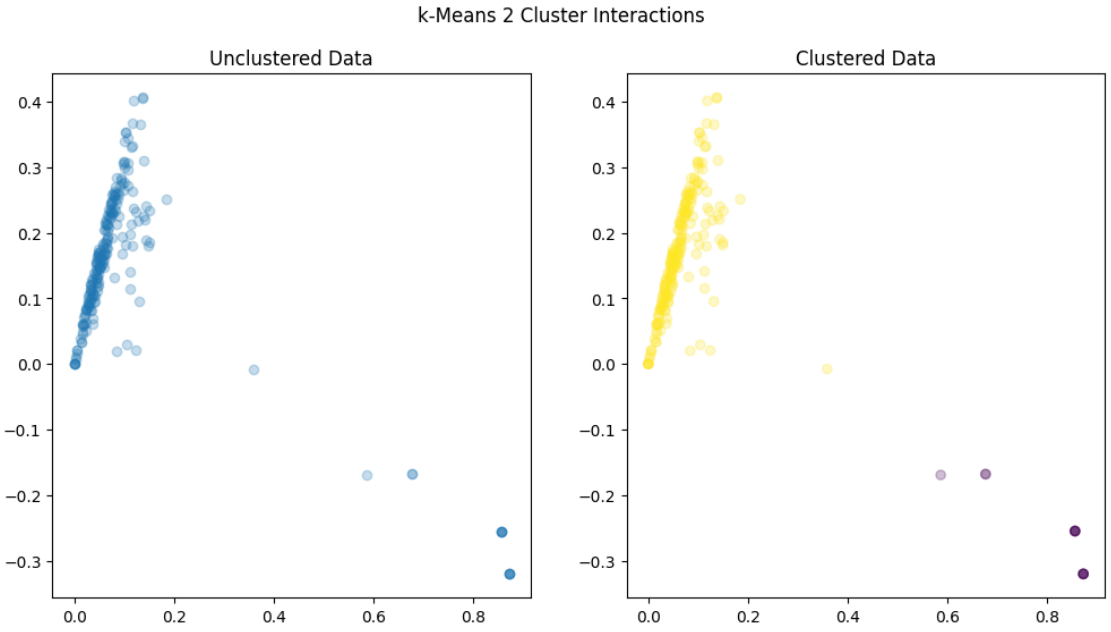
Overall, it's difficult to tell. Our best outcome is k-means, and it seems to largely create two types: one that represents most violations accounter for teh first cluster, and a group about being off the trail and having pets off leash, which would account for the second and third cluster.

9.4 Observations

9.4.1 K-Means

[ ]:
```
# k-Means clustering with n_clusters = 2
obsv = []
for observation in observations:
  obsv.append(observation['msg'])

obsv_vectors = tfidf_vectorizer.fit_transform(obsv)

obsv_pred_2 = KMeans(n_clusters=2, n_init='auto').fit_predict(obsv_vectors)
print("Shape of clustered interactions:", obsv_pred_2.shape)
plot_compare(obsv_vectors, obsv_pred_2, "k-Means 2 Cluster Interactions", alpha␣
 ↪= 0.25)

pull_samples(obsv, obsv_pred_2, 5)
```

```
Shape of clustered interactions: (223,)
```




```
Label: 0
Number of texts in this cluster: 13
Sample text 1 (Index: 196 ): Nothing notable, gorgeous day
Sample text 2 (Index: 174 ): Nothing notable to report.
Sample text 3 (Index: 218 ): nothing notable happened today.
Sample text 4 (Index: 194 ): Nothing notable.
Sample text 5 (Index: 178 ): Nothing notable.
----------------------------
Label: 1
Number of texts in this cluster: 210
Sample text 1 (Index: 69 ): took out some trash
Sample text 2 (Index: 97 ): surprised there were no violations with the
large number of people counted, maybe because of the cool temperatures or the
fact that its tuesday
Sample text 3 (Index: 222 ): lots of poeple all over! great last day.
Hope to continue again next summer!
Sample text 4 (Index: 48 ): whistles are really, really loud. picked up
some trash. saw some cool plants
Sample text 5 (Index: 122 ): got ice cream, saw snake
----------------------------
```

9.4.2 DBSCAN

[ ]:
```
obsv_hdv_pred = DBSCAN().fit_predict(obsv_vectors.toarray())

print("DBSCAN Clustering Labels Shape:", obsv_hdv_pred.shape)
```

```
plot_compare(obsv_vectors, obsv_hdv_pred, "DBSCAN Results")

pull_samples(obsv, obsv_hdv_pred, n=5)
```

```
DBSCAN Clustering Labels Shape: (223,)
```


```
Label: -1
Number of texts in this cluster: 213
Sample text 1 (Index: 33 ): air got worse throughout the day
Sample text 2 (Index: 2 ): someone asked me where the slope was lol
Sample text 3 (Index: 66 ): Anabela dumped her bike quite a few times,
but didn't get hurt…much!
Sample text 4 (Index: 159 ): I retreived the infamous red plate! I also
met Todd Bittner from the Botanic Gardens, he says to tell Mark hello.
Sample text 5 (Index: 114 ): Rainy day for about two hours, kinda scared
everyone off
----------------------------
Label: 0
Number of texts in this cluster: 5
Sample text 1 (Index: 194 ): Nothing notable.
Sample text 2 (Index: 197 ): Nothing notable.
Sample text 3 (Index: 203 ): Nothing notable.
Sample text 4 (Index: 178 ): Nothing notable.
Sample text 5 (Index: 175 ): Nothing notable.
----------------------------
Label: 1
Number of texts in this cluster: 5
```

```
Sample text 1 (Index: 218 ): nothing notable happened today.
Sample text 2 (Index: 216 ): Nothing notable happened today.
Sample text 3 (Index: 219 ): nothing notable happened today.
Sample text 4 (Index: 217 ): Nothing notable happened today.
Sample text 5 (Index: 221 ): nothing notable happened today.
----------------------------
```

9.4.3 Findings

Observations proved less interesting. The two clusters that both k-means and DBSCAN discovered were essentially an observation or non-observation.

9.5 Discussion

It's interesting to see how we might be able to group our interactions and observations into clusters. Overall, it seems that each respective group had a binary label: either violation or non-violation, observation or non-observation. This makes sense overall.

Within the violations, however, we were able to eek out a little insight into the types of violations. There was a cluster that represented cascadilla, one about animals off leash or people off the path, and one that covered the rest.

We use these clustering of violation/non-violation for future classification.

10 Classification

We now attempt to see if we can classify violations and non-violations based on features in the dataset and models.

10.1 Labelling

We tokenize the interactions and hand label based on the k-means clustering, as well as hand labelling some false-positives.

[ ]:
```
inter_tokenized = []
for inter in interactions_unsplit:
  inter_tokenized.append(nltk.word_tokenize(inter['msg']))

violation_labels = ['no violation', 'casc', 'other']

steward_labels = []
for interaction in interactions_unsplit:
  steward = interaction['steward']
  if steward not in steward_labels:
    steward_labels.append(steward)

year_labels = []
for interaction in interactions_unsplit:
  year = str(interaction['date']).split('-')[0]
```

```
  if year not in year_labels:
    year_labels.append(year)
```

[ ]:
```
false_positives = [0, 1, 4, 6, 8, 10, 11, 19, 22, 24, 27, 32, 39, 42, 49, 50,␣
↪57, 61, 62, 65, 69, 71, 78, 79, 81, 87, 93, 94, 95, 96, 104, 105, 107, 109,␣
↪115, 120, 144, 148, 155, 157, 171, 184, 198, 220, 226, 236, 242, 256, 299,␣
↪301,296, 295,294, 311, 314, 316, 317, 318, 319, 320, 322, 323, 324, 325, 326]
for interaction in interactions_unsplit:
  if interaction['index'] in false_positives:
    interaction['violation_label'] = False
```

10.2 Pre-Processing and Vectorizing

We again split the data, so we have more interactions, as was done in the import and setup phase.

[ ]:
```
interactions_resplit = []
for interaction in interactions_unsplit:
    parts = re.split(r'[.,;\n]', interaction['msg'])
    for part in parts:
        part = part.strip()
        if part:
            new_inter = interaction.copy()
            new_inter['msg'] = part
            interactions_resplit.append(new_inter)

interactions_df = pd.DataFrame(interactions_resplit)

# for loop through interactions_df
for i, interaction in interactions_df.iterrows():
  interaction['index'] = i

interactions_df.set_index('index', inplace=True)
labels = interactions_df['violation_label']

print('Total interactions:',len(interactions_df))
print('number of violations:',␣
↪len(interactions_df[interactions_df['violation_label'] == True]))
print('number of non-violations:',␣
↪len(interactions_df[interactions_df['violation_label'] == False]))
print('interactions dataframe:')
interactions_df.head()
```

```
Total interactions: 591
number of violations: 490
number of non-violations: 101
interactions dataframe:
```

[ ]:
```
            date  steward                                                msg  \
index
0     2023-05-25  Brenner                    Someone asked about north campus
1     2023-05-26  Brenner            lots of commencement direction questions
2     2023-05-27  Brenner         few people suntanning on rocks in the water
3     2023-05-28  Phillip                Lots of people looking for arboretum
3     2023-05-28  Phillip  three dogs off leash on Bebee Lake trail and C…

       violation_label  casc_label
index
0                False       False
1                False       False
2                 True       False
3                 True        True
3                 True        True
```

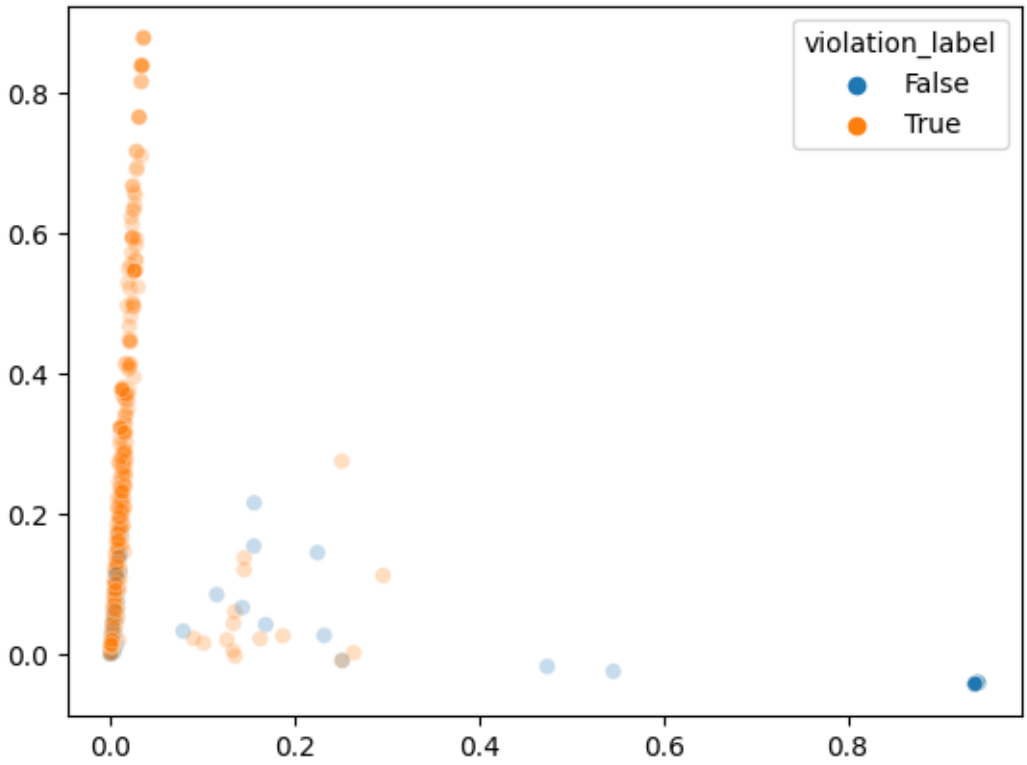
[ ]:
```
# vectorize
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(
    strip_accents='unicode',
    lowercase=True,
    use_idf=True)

inter_feats = tfidf_vectorizer.fit_transform(interactions_df['msg'])

print("Feature matrix shape:", inter_feats.shape)

tfidf_vectorizer.get_feature_names_out()[:25]
```

```
Feature matrix shape: (591, 801)
```

[ ]:
```
array(['10', '10yo', '11', '12', '15', '1999', '1st', '2nd', '30', '50',
       '80s', 'able', 'about', 'above', 'accumulating', 'across',
       'actually', 'adults', 'advised', 'after', 'afternoon', 'again',
       'air', 'all', 'allowed'], dtype=object)
```

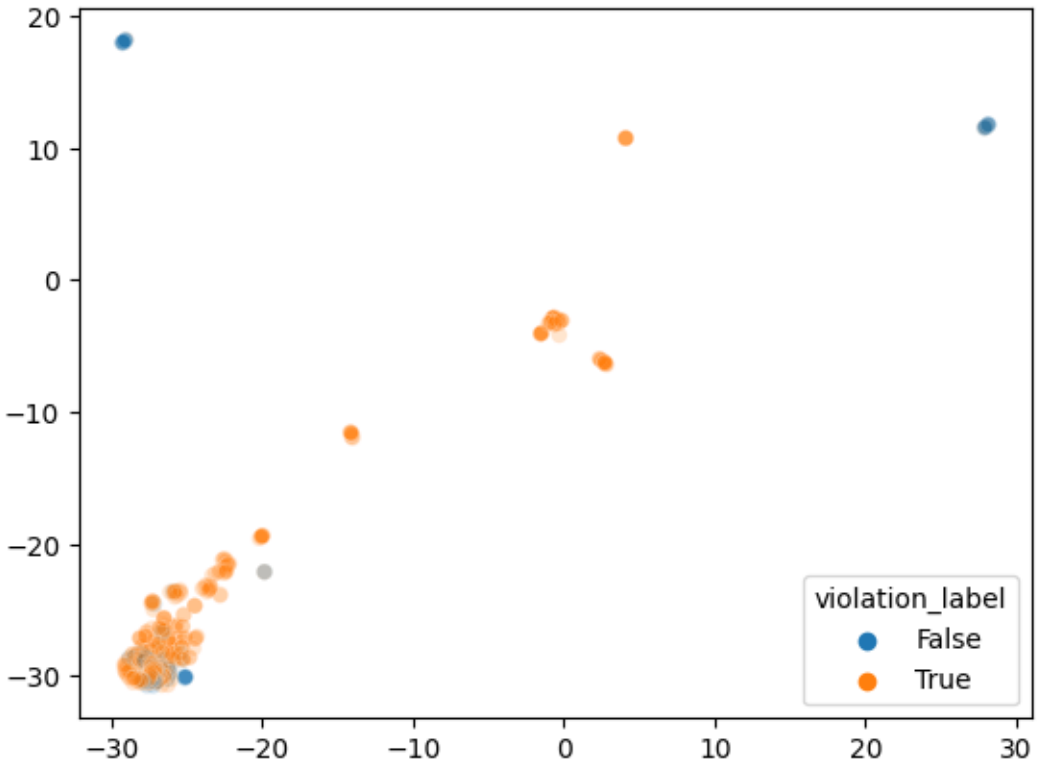
10.3 Truncated SVD Visualization

[ ]:
```
X_reduced = TruncatedSVD().fit_transform(inter_feats)
sns.scatterplot(x=X_reduced[:,0], y=X_reduced[:,1],␣
 ↪hue=interactions_df['violation_label'], alpha=0.25)
plt.show()
```



10.4 UMAP Visualization

[ ]:
```
X_umap = UMAP().fit_transform(inter_feats)
sns.scatterplot(x=X_umap[:,0], y=X_umap[:,1],␣
  ↪hue=interactions_df['violation_label'], alpha=0.2)
plt.show()
```



[ ]:
```
# Calculate baseline performance

# most_common_label = labels.mode().iloc[0]

# base_preds = [most_common_label] * len(labels)

# base_acc = accuracy_score(labels, base_preds)
# print("Baseline Accuracy:", base_acc)

# base_f1_score = f1_score(labels, base_preds, average='binary',␣
↪pos_label=most_common_label)
# print("Baseline F1 Score:", base_f1_score)
```

10.5 Select K-Best Features

[ ]:
```
# select kbest features
%%capture

from sklearn.feature_selection import SelectKBest, mutual_info_classif

k_best = SelectKBest(score_func=mutual_info_classif, k=8)
selected_features = k_best.fit_transform(inter_feats, labels)
```

```
selected_feature_indices = k_best.get_support(indices=True)
selected_feature_names = tfidf_vectorizer.
 ↪get_feature_names_out()[selected_feature_indices]
```

[ ]:
```
print(selected_feature_names)
```

```
['casc' 'in' 'no' 'people' 'the' 'to' 'today' 'violations']
```

10.5.1 Findings

Unsurprisingly, we see that among the most important features are *casc* and *violations*. These obvoiusly indicate an interaction that's most likely a violation. The other words likely are included because they are so common.

10.6 Classifiers Performance

[ ]:
```
# Classifers to test
classifiers = {
    'kNN': KNeighborsClassifier(),
    'Logit':LogisticRegression(),
    'decision_tree': DecisionTreeClassifier(),
    'rand_forest': RandomForestClassifier(),
    'naive_bayes': GaussianNB(),
    'multiNB': MultinomialNB(),
    'svc': SVC()
}

def compare_scores(scores_dict):
    '''
    Takes a dictionary of cross_validate scores.
    Returns a color-coded Pandas dataframe that summarizes those scores.
    '''
    df = pd.DataFrame(scores_dict).T.applymap(np.mean).style.
 ↪background_gradient(cmap='RdYlGn')
    return df
```

[ ]:
```
scores_dict = {}

for clf_model, clf in classifiers.items():
        scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro',␣
 ↪'f1_micro']
        scores = cross_validate(clf, inter_feats.toarray(), labels, cv=5,␣
 ↪scoring=scoring)
        scores_dict[clf_model] = {
            'Accuracy': np.mean(scores['test_accuracy']),
            'Precision (Macro)': np.mean(scores['test_precision_macro']),
            'Recall (Macro)': np.mean(scores['test_recall_macro']),
            'F1 (Macro)': np.mean(scores['test_f1_macro']),
```

```
            'F1 (Micro)': np.mean(scores['test_f1_micro'])
        }

scores_df = compare_scores(scores_dict)
display(scores_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
<pandas.io.formats.style.Styler at 0x7e26967774c0>
```

10.6.1 Findings

We see that the best performing are random forst, multiNB, and SVC. Their accuracy is around 0.88 each, meaning they are overall 88% correct. Their 0.89 percision means they were 90% correct in its predictions of violations. However, their lower recall meant they only could find 70% of true positives.

10.7 Model Tuning

10.7.1 Stratified K-Folds

[ ]:
```
max_depth_values = [None, 100000, 1000, 100, 20, 10, 5]

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracy_scores = {}

def get_accuracy_scores(max_depth):
    classifier = DecisionTreeClassifier(max_depth=max_depth)
    scores = cross_validate(classifier, inter_feats.toarray(), labels, cv=cv,␣
 ↪scoring='accuracy')
    return np.mean(scores['test_score'])

for depth in max_depth_values:
    accuracy_scores[depth] = get_accuracy_scores(depth)

for depth, accuracy in accuracy_scores.items():
    print(f"Max Depth: {depth if depth is not None else 'Unlimited'}: Accuracy:␣
 ↪{accuracy:.4f}")
```

```
Max Depth: Unlimited: Accuracy: 0.8511
Max Depth: 100000: Accuracy: 0.8579
Max Depth: 1000: Accuracy: 0.8511
Max Depth: 100: Accuracy: 0.8460
Max Depth: 20: Accuracy: 0.8545
```

```
Max Depth: 10: Accuracy: 0.8613
Max Depth: 5: Accuracy: 0.8714
```

10.7.2 Naive Bayes

[ ]:
```
%%capture

naive_bayes_classifier = MultinomialNB()

f1_scores_naive_bayes = []

# we iterate over different feature counts (1-5) and find the best F1 score␣
 ↪based on cross-validation
for k in range(1, 10):
    selected_features = k_best.fit_transform(inter_feats, labels)

    f1_scores = cross_validate(naive_bayes_classifier, selected_features,␣
 ↪labels, cv=5,
                               scoring=make_scorer(f1_score,␣
 ↪average='weighted'))['test_score']

    f1_scores_naive_bayes.append((k, f1_scores.mean()))

best_k_naive_bayes, best_f1_naive_bayes = max(f1_scores_naive_bayes, key=lambda␣
 ↪x: x[1])
```

[ ]:
```
print(f"Best Number of Features for Naive Bayes: {best_k_naive_bayes}")
print(f"Best F1 Score for Naive Bayes: {best_f1_naive_bayes:.4f}")
```

```
Best Number of Features for Naive Bayes: 1
Best F1 Score for Naive Bayes: 0.8458
```

10.7.3 KNN

[ ]:
```
knn_classifier = KNeighborsClassifier()

f1_scores_knn = []

# Iterate over different k values and find the best k value
for k in range(1, 10):
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(inter_feats.toarray())

    f1_scores = cross_validate(knn_classifier, scaled_data, labels, cv=5,
                               scoring=make_scorer(f1_score,␣
 ↪average='weighted'))['test_score']

    f1_scores_knn.append((k, f1_scores.mean()))
```

```
best_k_knn, best_f1_knn = max(f1_scores_knn, key=lambda x: x[1])

print(f"Best Number of Neighbors (k) for kNN: {best_k_knn}")
print(f"Best F1 Score for kNN: {best_f1_knn:.4f}")
```

```
Best Number of Neighbors (k) for kNN: 1
Best F1 Score for kNN: 0.8389
```

10.7.4 Findings

We discover that with tuning, most of the models performed best with the least dimensionality. With their tuning, they were able to score margianlly better, with around 0.85 for each. This is a strong score that indicates that the interactions are able to be predicted well by these statistical models.

10.8 Discussion

Basic statistical models seems to be very good at predicting whether an interaction is a violation or not. A simple KNN model or naive bayes model of just one fold or one feature can lead to a f1 score of 0.85, meaning the model is not only accurate but also percise.

This largely makes sense as the difference between violation and non-violation is can be easily picked out based on patterns. For non-violations, the tokens `no` and `violations` tend to be helpful, and for any other interaction with `violation` probably indicates a violation. These models, then, are quite good at picking up this discrepency and predicting as such.

11 Topic Models

We are briefly interested extracting possible topics from our interactions.

11.1 Pre-Processing

[52]:
```
def display_topics(model, feature_names, k_top_words, title, n_topics=10):
    fig, axes = plt.subplots(round(n_topics/5), 5, figsize=(30, 15),␣
 ↪sharex=True)
    axes = axes.flatten()
    for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[-k_top_words:]
        top_features = feature_names[top_features_ind]
        weights = topic[top_features_ind]

        ax = axes[topic_idx]
        ax.barh(top_features, weights, height=0.7)
        ax.set_title(f"Topic {topic_idx +1}", fontdict={"fontsize": 30})
        ax.tick_params(axis="both", which="major", labelsize=20)
        for i in "top right left".split():
            ax.spines[i].set_visible(False)
```

```
        fig.suptitle(title, fontsize=40)

    plt.subplots_adjust(top=0.90, bottom=0.05, wspace=0.90, hspace=0.3)
    plt.tight_layout()
    plt.show()
```

[ ]:
```
count_vectorizer = CountVectorizer(
    input = 'content',
    encoding = 'utf-8',
    strip_accents = 'unicode',
    stop_words='english',
    lowercase = True,
    min_df = 0.001,
    max_df = 0.25
)

inter_count_feats = count_vectorizer.fit_transform(interactions_df['msg'])
```

11.2 LDA

[ ]:
```
# topic model code
lda = LatentDirichletAllocation(
    n_components=20,
    n_jobs=-1,
    verbose=1,
    max_iter=10,
    evaluate_every=0
)

lda.fit(inter_count_feats)
```

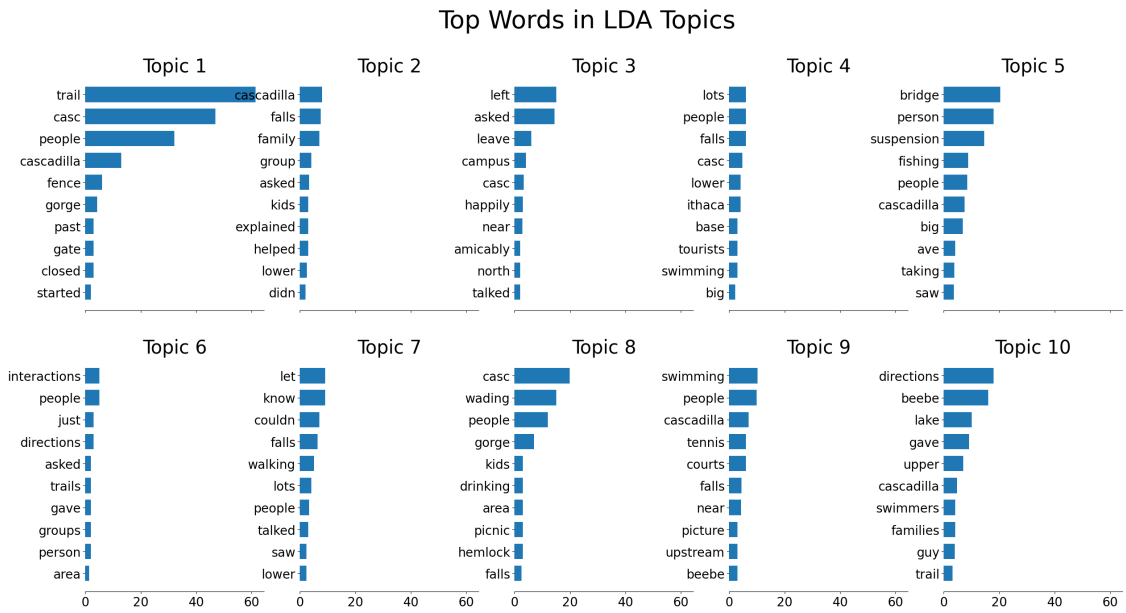
```
iteration: 1 of max_iter: 10
iteration: 2 of max_iter: 10
iteration: 3 of max_iter: 10
iteration: 4 of max_iter: 10
iteration: 5 of max_iter: 10
iteration: 6 of max_iter: 10
iteration: 7 of max_iter: 10
iteration: 8 of max_iter: 10
iteration: 9 of max_iter: 10
iteration: 10 of max_iter: 10
```

[ ]:
```
LatentDirichletAllocation(evaluate_every=0, n_components=20, n_jobs=-1,
                          verbose=1)
```

11.3 Top Words Visualization

[ ]:
```
# plot topic output


k_top_words = 10
feature_names = count_vectorizer.get_feature_names_out()
title = "Top Words in LDA Topics"

display_topics(lda,
               feature_names,
               k_top_words,
               title)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-131-20609c8376f0> in <cell line: 8>()
      6 title = "Top Words in LDA Topics"
      7
----> 8 display_topics(lda,
      9                feature_names,
     10                k_top_words,

<ipython-input-128-c90f05300098> in display_topics(model, feature_names,␣
 ↪k_top_words, title, n_topics)
      7         weights = topic[top_features_ind]
      8
----> 9         ax = axes[topic_idx]
     10         ax.barh(top_features, weights, height=0.7)
     11         ax.set_title(f"Topic {topic_idx +1}", fontdict={"fontsize": 30}

IndexError: index 10 is out of bounds for axis 0 with size 10
```



11.3.1 Findings

It's difficult to disvern between the 10 topics suggested by our LDA model.

11.4 Dot Topic Matrix

[ ]:
```
# doc-topic matrix
doc_topic_matrix = lda.transform(inter_count_feats)
print("Doc-topic matrix shape:", doc_topic_matrix.shape)

X_topics = doc_topic_matrix
y = interactions_df['violation_label']

classifier = LogisticRegression(max_iter=1000)

cross_val_scores = cross_val_score(classifier,
                                   X_topics,
                                   y,
                                   cv=5,
                                   scoring='f1_weighted')

average_weighted_f1 = cross_val_scores.mean()

print("Average Cross-Validated Weighted F1 Score:", average_weighted_f1)
```

```
Doc-topic matrix shape: (591, 20)
Average Cross-Validated Weighted F1 Score: 0.8111780433870104
```

11.5 Modeling

[ ]:
```
# classify using informative tokens
k_best = SelectKBest(score_func=mutual_info_classif, k=25)
x_best_features = k_best.fit_transform(inter_count_feats,␣
 ↪interactions_df['violation_label'])

cross_val_scores = cross_val_score(classifier,
                                   x_best_features,
                                   interactions_df['violation_label'],
                                   cv=5,
                                   scoring='f1_weighted')

average_f1_score = cross_val_scores.mean()

print("Shape of the Input Matrix:", x_best_features.shape)
print("Average Cross-Validated Weighted F1 Score:", average_f1_score)
```

```
Shape of the Input Matrix: (591, 25)
Average Cross-Validated Weighted F1 Score: 0.848481221695363
```

[ ]:
```
# classify using SVD
svd = TruncatedSVD(n_components=20)

x_svd = svd.fit_transform(inter_count_feats)

classifier = LogisticRegression(max_iter=1000)

weighted_f1_scorer = make_scorer(f1_score, average='weighted')

cross_val_scores = cross_val_score(classifier,
                                   x_svd,
                                   labels,
                                   cv=5,
                                   scoring=weighted_f1_scorer)

average_f1_score = cross_val_scores.mean()

print("Shape of the Input Matrix (TruncatedSVD):", x_svd.shape)
print("Average Cross-Validated Weighted F1 Score (TruncatedSVD):",␣
 ↪average_f1_score)
```

```
Shape of the Input Matrix (TruncatedSVD): (591, 20)
Average Cross-Validated Weighted F1 Score (TruncatedSVD): 0.8370096079009898
```

12 BERT-Based Classifier

We are now interested in seeing if we predict whether an interaction is a violation or non-violation with the pre-trained BERT NLP model.

12.1 Train Test Split

We split using a standard 80/20 train-test.

[ ]:
```
inter_texts = {'violation': [], 'non-violation': []}

for index, interaction in interactions_df.iterrows():
  if interaction['violation_label'] == True:
    label = 'violation'
  else:
    label = 'non-violation'
  msg = interaction['msg']
  inter_texts[label].append(msg)

print("Total number of texts:", len(inter_texts['violation']) +␣
 ↪len(inter_texts['non-violation']))
print("Number of violation texts:", len(inter_texts['violation']))
print("Number of non-violation texts:", len(inter_texts['non-violation']))
```

```
Total number of texts: 591
Number of violation texts: 490
Number of non-violation texts: 101
```

[ ]:
```
train_texts = []
train_labels = []
test_texts = []
test_labels = []

for label, text in inter_texts.items():
  test_split = int(len(text) * 0.8)
  validation_split = int(test_split * 0.2)
  text = random.sample(text, len(text))

  for inter in text[test_split:]:
    test_texts.append(inter)
    test_labels.append(label)

  for inter in text[:test_split]:
    train_texts.append(inter)
    train_labels.append(label)

print("80/20 Train-test split:")
```

```
print("Training set:", len(train_texts), "interactions and", len(train_labels),␣
  ↪"labels")
print("Testing set:", len(test_texts), "interactions and", len(test_labels),␣
  ↪"labels")
```

```
80/20 Train-test split:
Training set: 472 interactions and 472 labels
Testing set: 119 interactions and 119 labels
```

12.2 Tokenizing

We load `DistilBertTokenizerFast` from the HuggingFace library and use it to encode all of the text. This tokenizer breaks words into pieces, trunkates them into tokens, and adds padding and tokens for the BERT model.

[ ]:
```
# encode and tokenize data for lil BERT
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
```

[ ]:
```
unique_labels = set(label for label in train_labels)
label2id = {label: id for id, label in enumerate(unique_labels)}
id2label = {id: label for label, id in label2id.items()}

print(label2id.keys())
```

```
dict_keys(['violation', 'non-violation'])
```

[ ]:
```
# encode and tokenize data for lil BERT
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-cased')

train_encodings = tokenizer(train_texts, truncation=True, padding=True)
test_encodings  = tokenizer(test_texts, truncation=True, padding=True)

' '.join(train_encodings[1].tokens[0:100])
```

[ ]:
```
'[CLS] no violations [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD]'
```

12.3 Mapping Labels

We set up a dictionary to map each unique label in the training data to a unique integer. There are only two labels, “violation” and “non-violation,” so they are assigned to 0 and 1, respectively. This allows us to map lables to IDs and vice versa.

[ ]:
```
# set labels
unique_labels = set(label for label in train_labels)
label2id = {label: id for id, label in enumerate(unique_labels)}
id2label = {id: label for label, id in label2id.items()}
```

```
print("label2id and id2label keys:")
print(label2id.keys())
print(id2label.keys())
```

```
label2id and id2label keys:
dict_keys(['violation', 'non-violation'])
dict_keys([0, 1])
```

12.4 Encoding

We now encode the texts and labels.

[ ]:
```
train_encodings = tokenizer(train_texts, truncation=True, padding=True)
test_encodings  = tokenizer(test_texts, truncation=True, padding=True)

train_labels_encoded = [label2id[y] for y in train_labels]
test_labels_encoded  = [label2id[y] for y in test_labels]

type(train_encodings)
```

[ ]:
```
transformers.tokenization_utils_base.BatchEncoding
```

12.5 Custom Torch Dataset

We now create two datasets for training and testing, useing a class `MyDataset` that we define.

[ ]:
```
# mydataset class
class MyDataset(torch.utils.data.Dataset):
  def __init__(self, encodings, labels):
    self.encodings = encodings
    self.labels = labels

  def __getitem__(self, idx):
    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
    item['labels'] = torch.tensor(self.labels[idx])
    return item

  def __len__(self):
    return len(self.labels)
```

[ ]:
```
# create datasets
train_dataset = MyDataset(train_encodings, train_labels_encoded)
test_dataset = MyDataset(test_encodings, test_labels_encoded)

print(f"train_dataset: {train_dataset}")
print(f"test_dataset: {test_dataset}")
print(f"train_dataset length: {len(train_dataset)}")
```

```
print(f"test_dataset length: {len(test_dataset)}")
```

```
train_dataset: <__main__.MyDataset object at 0x7e2696b77880>
test_dataset: <__main__.MyDataset object at 0x7e26969e2d40>
train_dataset length: 472
test_dataset length: 119
```

12.6 Model Training

[ ]:
```
# !pip install transformers[torch]
# !pip install accelerate
```

[ ]:
```
model = DistilBertForSequenceClassification.from_pretrained(
  'distilbert-base-cased',
  num_labels=len(id2label)
).to('cuda')
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-cased and are newly initialized:
['pre_classifier.bias', 'pre_classifier.weight', 'classifier.weight',
'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:
```
# # save model
# cached_model_directory_name = 'distilbert-finetuned-violation-classification'
# trainer.save_model(cached_model_directory_name)
```

[ ]:
```
training_args = TrainingArguments(
  num_train_epochs=10,
  per_device_train_batch_size=14,
  per_device_eval_batch_size=18,
  learning_rate=5e-5,
  warmup_steps=100,
  weight_decay=0.15,
  output_dir='./results',
  logging_dir='./logs',
  logging_steps=50,
  evaluation_strategy='steps',
)

def compute_metrics(pred):
  labels = pred.label_ids
  preds = pred.predictions.argmax(-1)
  score = f1_score(labels, preds, average='weighted')
  return {
      'f1': score,
  }
```

```
trainer = Trainer(
  model=model,
  args=training_args,
  train_dataset=train_dataset,
  eval_dataset=test_dataset,
  compute_metrics=compute_metrics
)
```

12.6.1 Argument Values

The values we set are important to ensure performance and speed. Due to the smaler size of this dataset, we choose 10 training epochs and 0.15 weight_decay, as that was found to be the ideal number, where the model would be able to have room to adjust, and not take up too much compute time. Batch size for training and evaluation is 14 and 18, respectively. The smaller size allowed for a more methodical but slower speed in training. Due to the small corpus, it still trained in reasonable time and was overall advantageous. The warmup steps were set to 100, as the dataset was comparably smaller than most other sets, where the default 50 would've been set. The weight decay was set to a much larger number, so the model would avoid overfitting, but have lattitude to improve its performance on each epoch.

[ ]:
```
trainer.train()
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
TrainOutput(global_step=340, training_loss=0.16126482381540186,
metrics={'train_runtime': 24.3441, 'train_samples_per_second': 193.887,
'train_steps_per_second': 13.966, 'total_flos': 50068537085760.0, 'train_loss':
0.16126482381540186, 'epoch': 10.0})
```

12.7 Model Evaluation

[ ]:
```
trainer.evaluate()
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
{'eval_loss': 0.3361034095287323,
 'eval_f1': 0.9198916408668731,
 'eval_runtime': 0.2342,
 'eval_samples_per_second': 508.131,
 'eval_steps_per_second': 25.62,
 'epoch': 10.0}
```

[ ]:
```
predicted_results = trainer.predict(test_dataset)
# print("predicted_results:")
# print(predicted_results)

print("---")
print("predicted_results.predictions:")
```

```
print(predicted_results.predictions[1])

print("---")
print("predicted_results.predictions.shape:")
print(predicted_results.predictions.shape)

predicted_labels = predicted_results.predictions.argmax(-1)
predicted_labels = predicted_labels.flatten().tolist()
predicted_labels = [id2label[l] for l in predicted_labels]

print(classification_report(test_labels,
                            predicted_labels))
```

```
<IPython.core.display.HTML object>

---
predicted_results.predictions:
[ 3.7721546 -4.510902 ]
---
predicted_results.predictions.shape:
(119, 2)
              precision    recall  f1-score   support

non-violation       0.70      0.90      0.79        21
    violation       0.98      0.92      0.95        98

     accuracy                           0.92       119
    macro avg       0.84      0.91      0.87       119
 weighted avg       0.93      0.92      0.92       119
```

[ ]:
```
for _true_label, _predicted_label, _text in random.sample(list(zip(test_labels,␣
 ↪predicted_labels, test_texts)), 10):
  if _true_label == _predicted_label:
    print('LABEL:', _true_label)
    print(' TEXT:', _text[:])
    print('---')
```

```
LABEL: violation
 TEXT: dad leading daughter onto outcropping in the middle of biggest casc falls
---
LABEL: violation
 TEXT: They were super nice
---
LABEL: violation
 TEXT: People trying to swim in the base of ithaca falls area
---
LABEL: violation
 TEXT: above or below the falls
```

```
---
LABEL: non-violation
 TEXT: small day for violators
---
LABEL: violation
 TEXT: some mom asked me to explain to their kid why they couldn't swim in casc
---
LABEL: violation
 TEXT: only about an hour of no rain
---
LABEL: violation
 TEXT: they'd been there all day
---
LABEL: violation
 TEXT: off trail in casc
---
LABEL: non-violation
 TEXT: No violations today
---
```

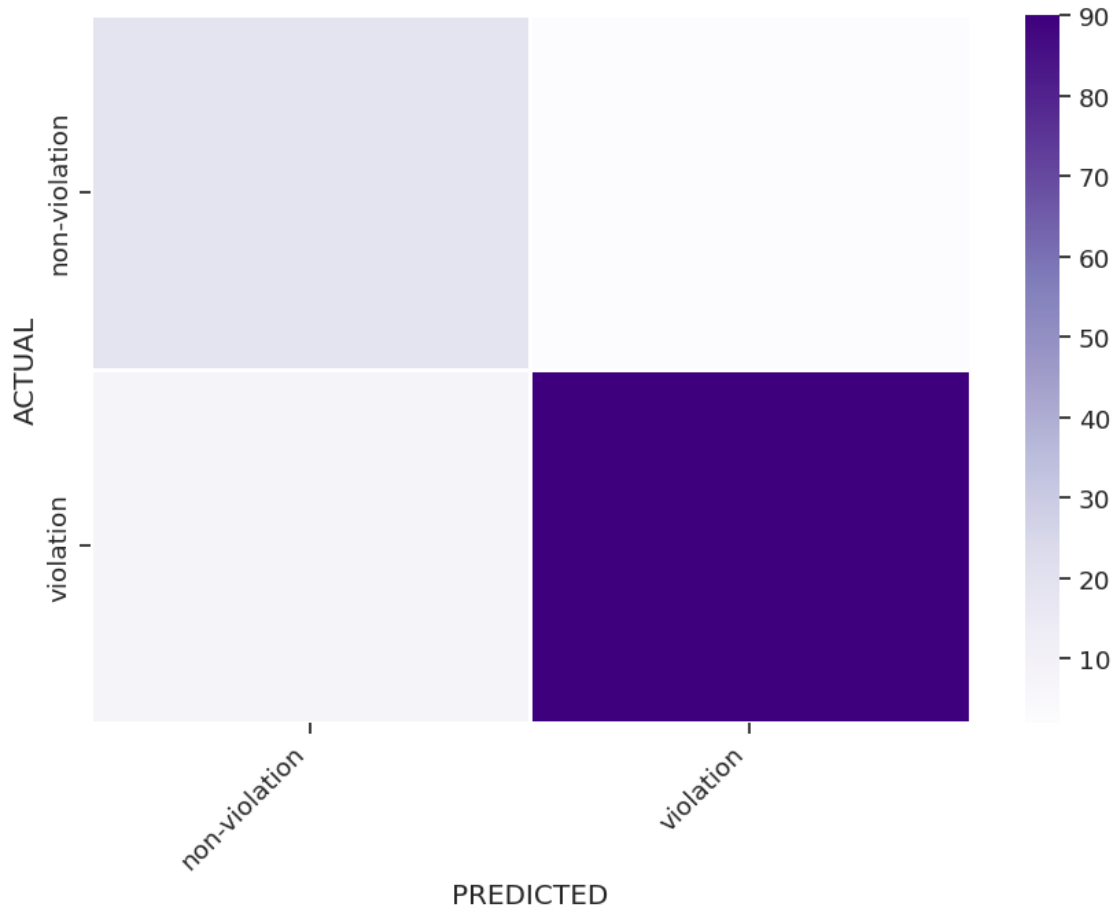
[ ]:
```
for _true_label, _predicted_label, _text in random.sample(list(zip(test_labels,␣
 ↪predicted_labels, test_texts)), 50):
  if _true_label != _predicted_label:
    print('ACTUAL LABEL:', _true_label)
    print('PREDICTED LABEL:', _predicted_label)
    print('TEXT\n', _text[:])
    print('---')
```

```
ACTUAL LABEL: violation
PREDICTED LABEL: non-violation
TEXT
 People in the water in Casc
---
ACTUAL LABEL: violation
PREDICTED LABEL: non-violation
TEXT
 Explained to a family that Cascadilla was still closed
---
ACTUAL LABEL: violation
PREDICTED LABEL: non-violation
TEXT
 6 swimmers in upper casdadilla
---
ACTUAL LABEL: violation
PREDICTED LABEL: non-violation
TEXT
 people sitting in water near falls
---
```

```
ACTUAL LABEL: violation
PREDICTED LABEL: non-violation
TEXT
 Gave directions to a few families
---
```

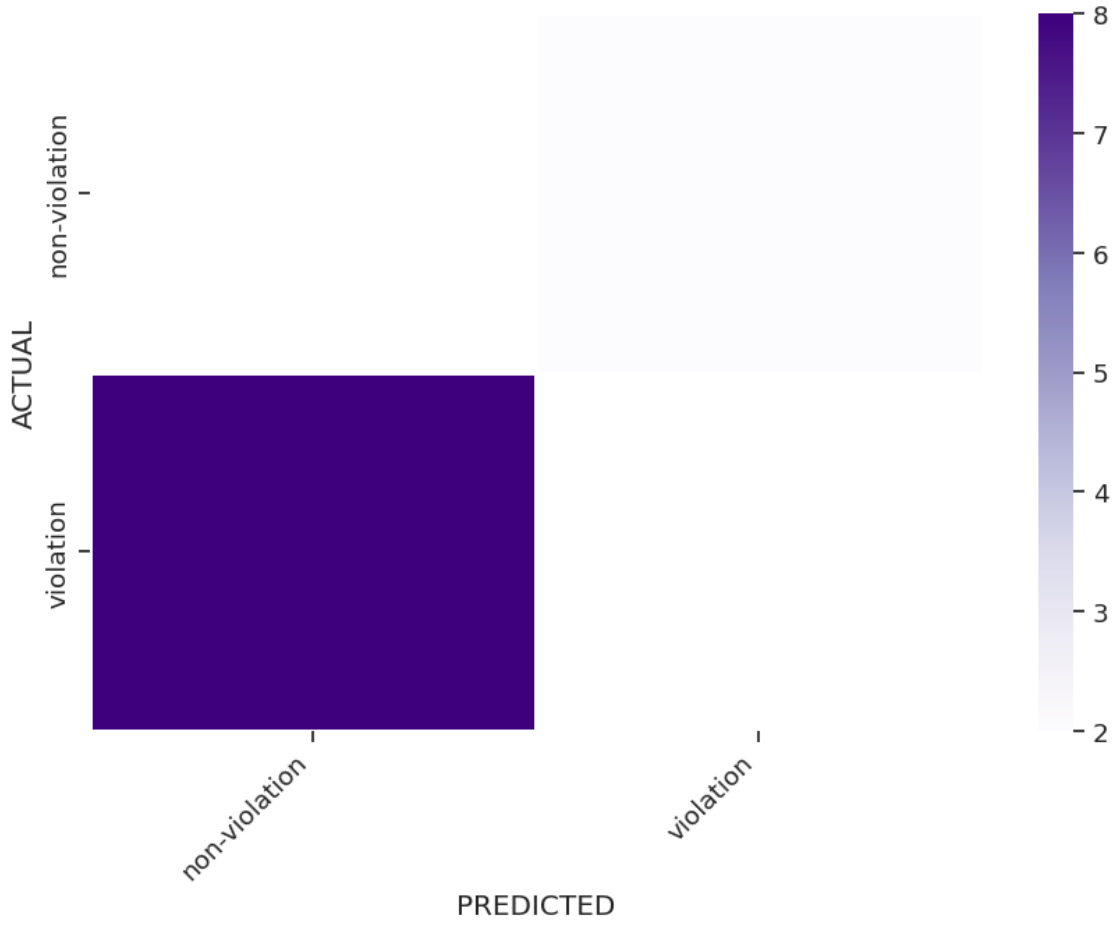
[ ]:
```
inter_classifications_dict = defaultdict(int)
for _true_label, _predicted_label in zip(test_labels, predicted_labels):
  inter_classifications_dict[(_true_label, _predicted_label)] += 1

dicts_to_plot = []
for (_true_genre, _predicted_genre), _count in inter_classifications_dict.
 ↪items():
  dicts_to_plot.append({'ACTUAL': _true_genre,
                        'PREDICTED': _predicted_genre,
                        'Number of Classifications': _count})

df_to_plot = pd.DataFrame(dicts_to_plot)
df_wide = df_to_plot.pivot_table(index='ACTUAL',
                                 columns='PREDICTED',
                                 values='Number of Classifications')
```

[ ]:
```
plt.figure(figsize=(9,7))
sns.set(style='ticks', font_scale=1.2)
sns.heatmap(df_wide, linewidths=1, cmap='Purples')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[ ]:
```
inter_classifications_dict = defaultdict(int)
for _true_label, _predicted_label in zip(test_labels, predicted_labels):
  if _true_label != _predicted_label: # Remove the diagonal to highlight␣
 ↪misclassifications
    inter_classifications_dict[(_true_label, _predicted_label)] += 1

dicts_to_plot = []
for (_true_genre, _predicted_genre), _count in inter_classifications_dict.
 ↪items():
  dicts_to_plot.append({'ACTUAL': _true_genre,
                        'PREDICTED': _predicted_genre,
                        'Number of Classifications': _count})

df_to_plot = pd.DataFrame(dicts_to_plot)
df_wide = df_to_plot.pivot_table(index='ACTUAL',
                                 columns='PREDICTED',
                                 values='Number of Classifications')
```

```
plt.figure(figsize=(9,7))
sns.set(style='ticks', font_scale=1.2)
sns.heatmap(df_wide, linewidths=1, cmap='Purples')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



12.8 Analysis

We find that the model performs much better than any of our previous classifiers. Our BERT-based model performs best when predicting violations, has a weakness of predicting non-violation when it's actually a violation. This is indicated in the heatmaps

Overall, however, the model is exceptional. The accuracy of 0.92 means that it accurately predicts in both classes at a success rate of 92% out of 119 cases. It does even cases of true positives. That is, it's 98% accurate at predicting violation and captures 92% overall of the violations. The 0.95 f1-score means its very percise and accurate.

When we look at what the model's samples of correct and incorrect, we see that it tends to label

descriptions as violation. Also, we find that some interactions may be mislabelled, which could lead to discrepency in performance.

All in all, the metrics indicate a very strong model. However, overfitting ought to be considered, as the log book is small in interactions not widely varied. Further analysis is done.

13 RoBERTa

RoBERTa is a LLM that builds on BERT, modifying key parameters, removes next sentence pre-training, and is trained on a larger more diverse dataset. We also try to employ RoBERTa and see how it performs.

[ ]:
```
# tokenize and encode the dataset with roberta
rob_tokenizer = AutoTokenizer.from_pretrained("roberta-base")

rob_train_encodings = tokenizer(train_texts, truncation=True, padding=True)
rob_test_encodings  = tokenizer(test_texts, truncation=True, padding=True)

rob_train_labels_encoded = [label2id[y] for y in train_labels]
rob_test_labels_encoded  = [label2id[y] for y in test_labels]

type(rob_train_encodings)
```

```
config.json:   0%|          | 0.00/481 [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

[ ]:
```
transformers.tokenization_utils_base.BatchEncoding
```

13.1 Tokenizing

We tokenize with an autoTokenizer an encode the tokens with the roberta model.

[ ]:
```
rob_train_dataset = MyDataset(rob_train_encodings, rob_train_labels_encoded)
rob_test_dataset = MyDataset(rob_test_encodings, rob_test_labels_encoded)

print(f"train_dataset: {rob_train_dataset}")
print(f"test_dataset: {rob_test_dataset}")
print(f"train_dataset length: {len(rob_train_dataset)}")
print(f"test_dataset length: {len(rob_test_dataset)}")
```

```
train_dataset: <__main__.MyDataset object at 0x7e2696e99a20>
test_dataset: <__main__.MyDataset object at 0x7e2696e99420>
train_dataset length: 472
test_dataset length: 119
```

13.2 RoBERTa Model Training

```
[ ]: rob_model = AutoModelForSequenceClassification.from_pretrained("roberta-base",␣
      ↪num_labels=2)

rob_training_args = TrainingArguments(
    output_dir="roberta-base-classification",
    num_train_epochs=15,
    per_device_train_batch_size=14,
    per_device_eval_batch_size=20,
    learning_rate=5e-5,
    warmup_steps=100,
    weight_decay=0.15,
    logging_steps=50,
    evaluation_strategy="epoch"
)

rob_trainer = Trainer(
  model=rob_model,
  args=rob_training_args,
  train_dataset=rob_train_dataset,
  eval_dataset=rob_test_dataset,
  compute_metrics=compute_metrics
)
```

```
Some weights of RobertaForSequenceClassification were not initialized from the
model checkpoint at roberta-base and are newly initialized:
['classifier.out_proj.weight', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.dense.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
[ ]: rob_trainer.train()
```

```
<IPython.core.display.HTML object>
```

```
[ ]: TrainOutput(global_step=510, training_loss=0.19638148686465096,
     metrics={'train_runtime': 85.5226, 'train_samples_per_second': 82.785,
     'train_steps_per_second': 5.963, 'total_flos': 149171635058400.0, 'train_loss':
     0.19638148686465096, 'epoch': 15.0})
```

13.2.1 Argument Values

We set values to similar as the BERT model. However, there are fewer warm up steps and the weight_decay is lowered. We increase the epochs and slightly increase the batch size, to allow for more training overall.

13.3 Model Evaluation

[ ]:
```
rob_predicted_results = rob_trainer.predict(test_dataset)
rob_predicted_results.predictions.shape

rob_predicted_labels = rob_predicted_results.predictions.argmax(-1)
rob_predicted_labels = rob_predicted_labels.flatten().tolist()
rob_predicted_labels = [id2label[l] for l in rob_predicted_labels]

len(predicted_labels)

print(classification_report(test_labels,
                            rob_predicted_labels))
```

```
<IPython.core.display.HTML object>
```

	precision	recall	f1-score	support
non-violation	0.64	0.67	0.65	21
violation	0.93	0.92	0.92	98
accuracy			0.87	119
macro avg	0.78	0.79	0.79	119
weighted avg	0.88	0.87	0.88	119

13.4 Analysis

Overall, the RoBERTa model performs slightly worse than the BERT model. The f1 accuracy score 0.87 is stil very good, and the violation percision is excellent. However, the model does not perform as well on predicting for non-violation. It could only capture 2/3 of these types of interactions.

14 Conclusion

The gorge steward's logbook provided a plethora of data, both quantitative and qualitative. The textual section, in particular, was the focus of this project. We found that, over time, violations in the Gorges decreased, as visitorship stayed roughly the same. That shows the success of the Gorge Steward Program.

However, on the Steward side, the jobs remain challenging at times. Many visitors knowingly break the rules and don't listen to a steward when asked to leave. Rarely, the police have to get involved. Many stewards often observe trash in the gorges, too. As a result, overall sentiment in the steward's log is negative, though the vast majority of entries are simply neutral descriptions. There are, however, a few instances of visitors being disrespectful or crude to the steward.

Although the log book had sections for interactions with visitors and observations the steward makes, the two had a lot in common. Often, one would read an interaction and think it belonged in the observations section, and vice versa. Euclidean differences proved that they were similar to us, showing that the two were indeed the same. Of course, there was plenty of overlap, as both

observations and interactions often took place in similar or identical locations, thus leading to many shared tokens.

Amongst the entries, it was interesting to see if there were any kinds of groupings. We found that in interactions, there were two clusters: violation and non-violation, as well as a label that considered if it was in Cascadilla or not. For observations, it was a similar result of observation/non-observation. Within the violations, we further divided them into off-trail/off-leash, to other violations, and no-violation.

Given the violation/non-violation type of interactions, we tried to train classification models to predict whether an interaction was a violation or not based on the entry. Many entries said "no violations," so in theory our model should perform well. The task of labeling was difficult, and in part, based on the clustering done previously.

Among statistical models, the random forest, multiNB, and SVC performed well. Even a simple logistic regression was adequate. Ultimately, KNN and Naive Bayes, with minimal tuning, were able to achieve 0.85 f1 scores.

We tried to build on that score with BERT-based models. These NLP models are pre-trained and can be further trained on our dataset. After encoding, tokenizing, and creating our dataset, we trained the BERT model on our dataset for quite a few generations. The result, however, was a very strong model that boasted a 0.92 f1 score. Not only was it accurate, but it was very precise, capturing 91% of true positives and true negatives.

Training on RoBERTa, a similar model that's said to have had more equitable training on a wider dataset, performed almost as well. A respectable 0.87 f1-score only indicated that recall was lower for non-violation interactions.

Overall, it's very fascinating to see how one might break down and analyze a log book such as this one. In the future, more work can be done to answer more complex questions. In particular, better labeling of the interactions is recommended, and further analysis of topics among interactions - beyond what we discussed.

If we could extract labels that identified the type of violation, such as dog-off-leash, smoking, swimming, etc., then we could go even further with classification. How might temperature be used to predict? How about the day of the week or even the year? How about all of the indicators combined?

It is with hope that this project is not complete but instead put on pause for further inspiration and time to work. Years in the future, when the logbook grows, even more analysis can be done. And perhaps other parks have similar log books – one can imagine a dataset that chronicles the national parks over the last 50 years. That would be an undoubtedly fascinating project.