```
# !pip install contractions


import joblib
import pandas as pd
import numpy as np
import re
import nltk
import contractions
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# check data
data_path = '/content/drive/MyDrive/Colab Notebooks/reddit_comments_with_subreddits.csv'

df = pd.read_csv(data_path)
df
```

	subreddit	comment
0	Canada	All skirt, no knickers. As they say.
1	Canada	The key word is in your own comment "Either wa...
2	Canada	Gasoline is a minor part of oil companies reve...
3	Canada	>DEI establishes quotas which make sure that r...
4	Canada	We all stole the best continent in the world t...
...	...	...
219846	Ottawa	Bhahahaha. God keep our land.. glorious and fr...
219847	Ottawa	It's classless. I wouldn't expect anything els...
219848	Ottawa	This is bad... Orange man will use this as an ...
219849	Ottawa	No, it's petty. Insane and rude even if you ha...
219850	Ottawa	I don't agree with it. Their anthem had existe...

219851 rows × 2 columns

```
df.columns
```

```
Index(['subreddit', 'comment'], dtype='object')
```

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import contractions

nltk.download('stopwords')
nltk.download('wordnet')

stop_words = set(stopwords.words('english'))
custom_stopwords = {
    'im', 'dont', 'like', 'people', 'thats', 'think', 'want', 'make', 'get', 'one', 'right', 'life','anyone', 'made',
    'also', 'still', 'could', 'said', 'much', 'go', 'anyone', 'better', 'love', 'hope', 'every', 'lot', 'someone', 'nigger',
```

```
    'know', 'even', 'pay', 'going', 'need', 'year', 'years', 'lol', 'guy', 'say', 'time', 'got', 'always', 'care',
    'way', 'long', 'thing', 'actually', 'mean', 'would', 'day', 'man', 'let', 'see', 'really', 'good', 'take', 'put'
}
all_stopwords = stop_words.union(custom_stopwords)

lemmatizer = WordNetLemmatizer()

def clean_comment(text):
    if pd.isnull(text):
        return ""

    # Expand contractions
    text = contractions.fix(text)

    # Lowercase and remove URLs and special chars
    text = text.lower()
    text = re.sub(r"http\S+|www\S+|https\S+", '', text)
    text = re.sub(r'[^a-z\s]', '', text)

    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    # Tokenize, remove stopwords, lemmatize
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in all_stopwords]

    return ' '.join(tokens)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
df['clean_comment'] = df['comment'].astype(str).apply(clean_comment)
df[['comment', 'clean_comment']].head()
```

	comment	clean_comment
0	All skirt, no knickers. As they say.	skirt knickers
1	The key word is in your own comment "Either wa...	key word comment either sugarcoated saying tie...
2	Gasoline is a minor part of oil companies reve...	gasoline minor part oil company revenue differ...
3	>DEI establishes quotas which make sure that r...	dei establishes quota sure race play decisive ...
4	We all stole the best continent in the world t...	stole best continent world together count some...

```
# Load model and vectorizer
model = joblib.load("/content/drive/MyDrive/Colab Notebooks/toxicity_model.pkl")
tfidf = joblib.load("/content/drive/MyDrive/Colab Notebooks/Preprocessing.pkl")
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/base.py:380: InconsistentVersionWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.4.2 when using version 1.6.1. This might lead to breaking
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/base.py:380: InconsistentVersionWarning: Trying to unpickle estimator RandomForestClassifier from version 1.4.2 when using version 1.6.1. This might lead to breaking
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/base.py:380: InconsistentVersionWarning: Trying to unpickle estimator TfidfTransformer from version 1.4.2 when using version 1.6.1. This might lead to breaking code o
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/base.py:380: InconsistentVersionWarning: Trying to unpickle estimator TfidfVectorizer from version 1.4.2 when using version 1.6.1. This might lead to breaking code or
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
```

```
# Apply TF-IDF preprocessing
comments = df['clean_comment'].astype(str).tolist()
X_tfidf = tfidf.transform(comments)

# Predict toxicity
predictions = model.predict(X_tfidf)
probs = model.predict_proba(X_tfidf)[:, 1]

# Add prediction results to the DataFrame
df['Toxic/NonToxic'] = np.where(predictions == 1, 'Toxic', 'NonToxic')
df['Toxicity_Probability'] = probs

# Optional: Display toxic comment counts
toxic_count = (df['Toxic/NonToxic'] == 'Toxic').sum()
print(f"\nTotal toxic comments found: {toxic_count} out of {len(df)}")

# Show final DataFrame with predictions
df.head()
```

```
Total toxic comments found: 24002 out of 219851
```

	subreddit	comment	clean_comment	Toxic/NonToxic	Toxicity_Probability
0	Canada	All skirt, no knickers. As they say.	skirt knickers	NonToxic	0.113935
1	Canada	The key word is in your own comment "Either wa...	key word comment either sugarcoated saying tie...	NonToxic	0.020000
2	Canada	Gasoline is a minor part of oil companies reve...	gasoline minor part oil company revenue differ...	NonToxic	0.010000
3	Canada	>DEI establishes quotas which make sure that r...	dei establishes quota sure race play decisive ...	NonToxic	0.020000
4	Canada	We all stole the best continent in the world t...	stole best continent world together count some...	NonToxic	0.040000

```
df['subreddit'].unique()
```

```
array(['Canada', 'Conservative', 'Politics', 'Worldnews', 'Democrats',
       'CanadianPolitics', 'Ontario', 'vancouver', 'Alberta',
       'PoliticalDiscussion', 'Ask_Politics', 'ModeratePolitics',
       'NeutralPolitics', 'Republican', 'AskTrumpSupporters',
       'AskConservatives', 'PoliticalHumor', 'ConservativeMemes',
       'EnoughLibertarianSpam', 'PoliticalCompassMemes', 'BuyCanadian',
       'Toronto', 'Ottawa'], dtype=object)
```

Word Cloud of Toxic Comments

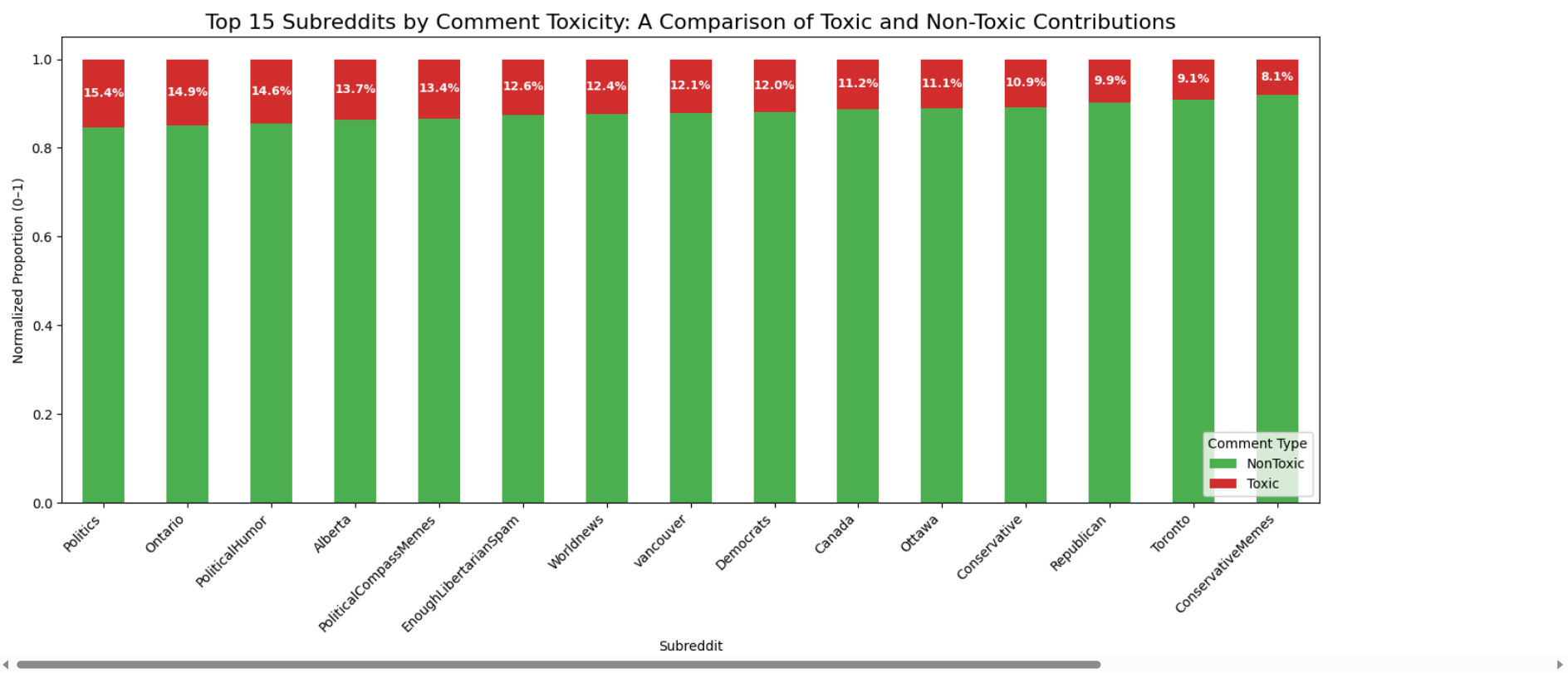
```
# Filter only toxic clean comments
toxic_comments = df[df['Toxic/NonToxic'] == 'Toxic']['clean_comment']

# Join all toxic comments into a single string
toxic_text = " ".join(toxic_comments.tolist())

# Create the word cloud
wordcloud = WordCloud(
    width=800,
    height=400,
    background_color='white',
    stopwords=stop_words,  # optional, in case your cleaning didn't remove all stopwords
    max_words=200
).generate(toxic_text)

# Plot the word cloud
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Toxic Comments', fontsize=16)
```

```
plt.show()
```

Top 15 Subreddits by Comment Toxicity: A Comparison of Toxic and Non-Toxic Contributions

```
# Count toxic and non-toxic
toxicity_counts = df.groupby(['subreddit', 'Toxic/NonToxic']).size().unstack(fill_value=0)

# Normalize (each row adds up to 1)
toxicity_normalized = toxicity_counts.div(toxicity_counts.sum(axis=1), axis=0)

# Sort by toxic proportion
toxicity_normalized = toxicity_normalized.sort_values(by='Toxic', ascending=False).head(15)

# Plot with better visual contrast and annotations
fig, ax = plt.subplots(figsize=(14, 7))

bars = toxicity_normalized.plot(
    kind='bar',
    stacked=True,
    color=['#4CAF50', '#D32F2F'],  # green for NonToxic, red for Toxic
    ax=ax
)

plt.title('Top 15 Subreddits by Comment Toxicity: A Comparison of Toxic and Non-Toxic Contributions', fontsize=16)
plt.ylabel('Normalized Proportion (0–1)')
plt.xlabel('Subreddit')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Comment Type', loc='lower right')

# Add percentage labels to Toxic part
for idx, row in enumerate(toxicity_normalized.itertuples()):
    toxic_val = getattr(row, 'Toxic')
    if toxic_val > 0.01:  # only label if it's not tiny
        ax.text(idx, 1 - toxic_val / 2, f'{toxic_val:.1%}', ha='center', va='center', color='white', fontsize=9, fontweight='bold')
```

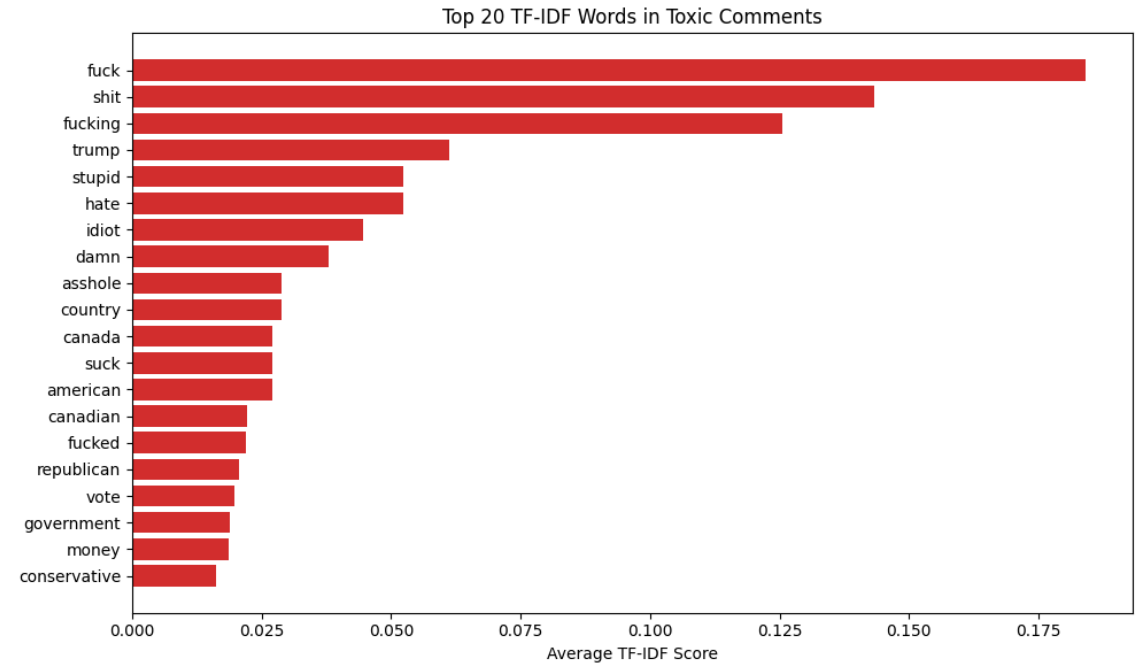
```
plt.tight_layout()
plt.show()
```



Top 20 TF-IDF Words in Toxic Comments

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Separate toxic comments
toxic_texts = df[df['Toxic/NonToxic'] == 'Toxic']['clean_comment'].dropna()

# Step 2: Vectorize
vectorizer = TfidfVectorizer(max_features=20, stop_words='english')
tox_vec = vectorizer.fit_transform(toxic_texts)
tox_words = vectorizer.get_feature_names_out()

# Step 3: Compute mean TF-IDF scores
mean_tfidf = tox_vec.mean(axis=0).A1  # convert to flat array

# Step 4: Sort and plot
sorted_indices = np.argsort(mean_tfidf)[::-1]
sorted_words = tox_words[sorted_indices]
sorted_scores = mean_tfidf[sorted_indices]
```

```
# Plot
plt.figure(figsize=(10, 6))
plt.barh(sorted_words[::-1], sorted_scores[::-1], color='#D32F2F')  # red for toxic
plt.xlabel('Average TF-IDF Score')
plt.title('Top 20 TF-IDF Words in Toxic Comments')
plt.tight_layout()
plt.show()
```



Topic Modeling

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt

# Step 1: Vectorize toxic comments using bigrams and refined stopwords
count_vectorizer = CountVectorizer(
    max_df=0.9,
    min_df=5,
    stop_words=list(all_stopwords),
    ngram_range=(1, 2)  # include unigrams + bigrams
)
X = count_vectorizer.fit_transform(toxic_texts)

# Step 2: Fit LDA with updated params
lda = LatentDirichletAllocation(n_components=3, learning_method='batch', random_state=42)
lda.fit(X)

# Step 3: Show top words per topic
def display_topics(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print(f"\nTopic #{topic_idx + 1}:")
        top_indices = topic.argsort()[::-1][:n_top_words]
```

```
        top_words = [feature_names[i] for i in top_indices]
        print(" | ".join(top_words))  # use | for easier visual separation

# Run display
display_topics(lda, count_vectorizer.get_feature_names_out(), n_top_words=10)
```

```
    Topic #1:
    fuck | fucking | stupid | damn | dumb | mask | suck | yeah | hate | trump

    Topic #2:
    shit | trump | hate | vote | republican | fucking | fuck | idiot | conservative | give

    Topic #3:
    fuck | fucking | shit | american | country | canada | cannot | tax | flag | canadian
```

Top Named Entities in Toxic vs Non-Toxic Comments

```
# !pip install -U spacy
# !python -m spacy download en_core_web_sm



import spacy
from collections import Counter
from tqdm import tqdm
tqdm.pandas()

# Load spaCy English model
nlp = spacy.load("en_core_web_sm")



# Optional: Sample a manageable number of toxic comments for speed (or remove this line for full dataset)
toxic_sample = df[df['Toxic/NonToxic'] == 'Toxic'].sample(10000, random_state=42)

# Function to extract named entities
def extract_entities(text):
    doc = nlp(text)
    return [ent.text for ent in doc.ents if ent.label_ in ["PERSON", "ORG", "GPE", "LOC"]]

# Apply entity extraction
toxic_sample['entities'] = toxic_sample['comment'].progress_apply(extract_entities)

# Flatten and count
toxic_entities = Counter([ent for sublist in toxic_sample['entities'] for ent in sublist])
```

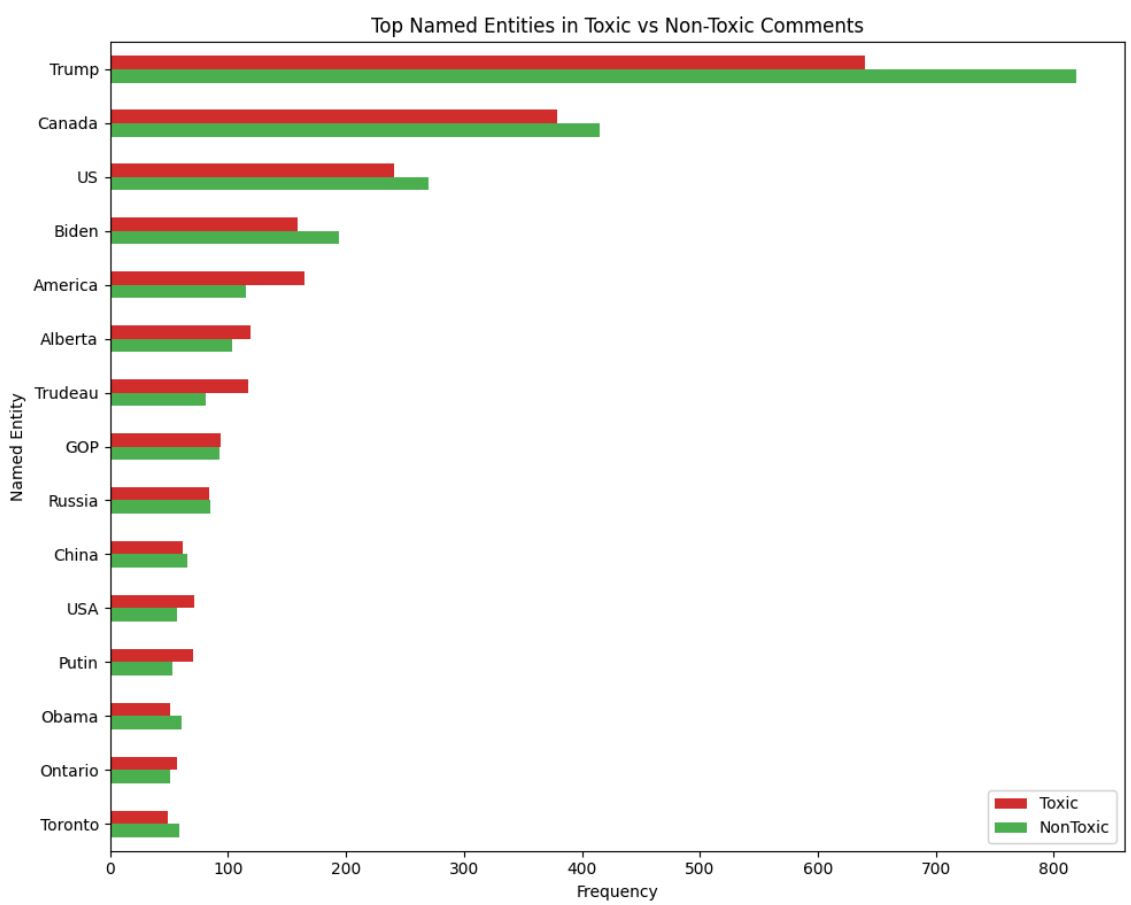
```
100%|██████████| 10000/10000 [02:14<00:00, 74.26it/s]
```

```
non_toxic_sample = df[df['Toxic/NonToxic'] == 'NonToxic'].sample(10000, random_state=42)
non_toxic_sample['entities'] = non_toxic_sample['comment'].progress_apply(extract_entities)
non_toxic_entities = Counter([ent for sublist in non_toxic_sample['entities'] for ent in sublist])
```

```
100%|██████████| 10000/10000 [02:22<00:00, 70.13it/s]
```

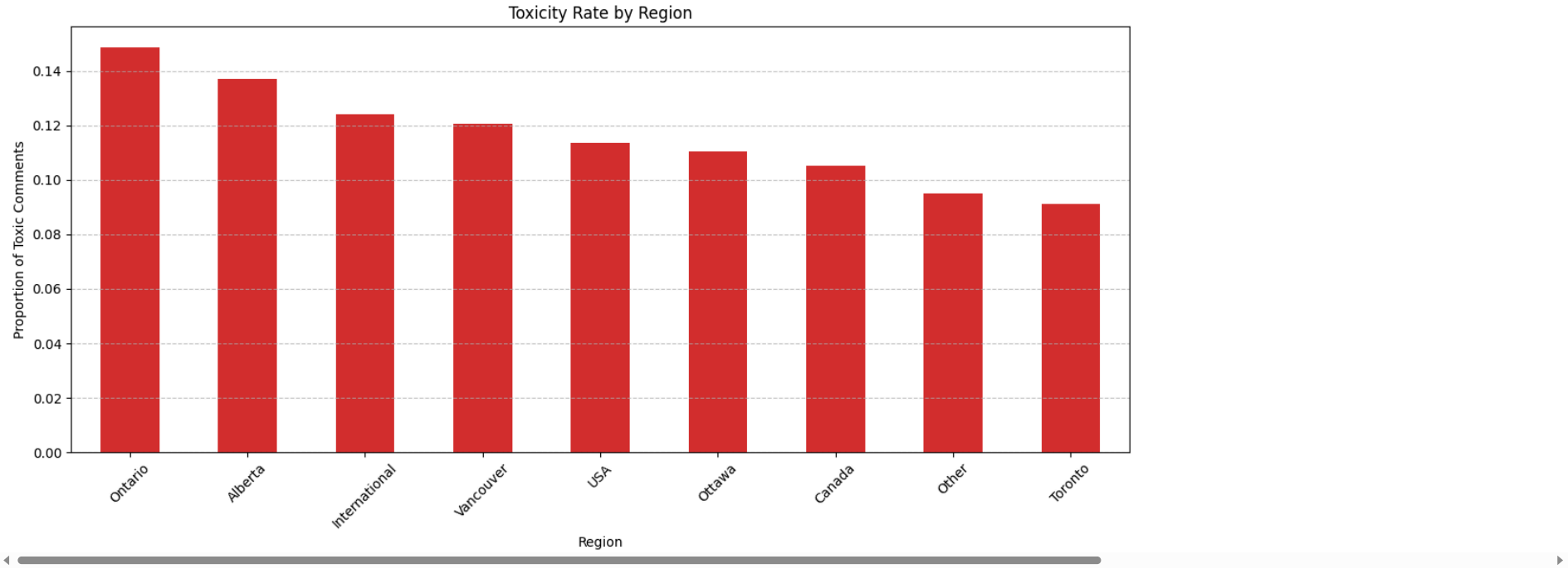
```
# Get top entities
toxic_top = dict(toxic_entities.most_common(20))
non_toxic_top = dict(non_toxic_entities.most_common(20))

# Combine into a DataFrame
entity_df = pd.DataFrame([toxic_top, non_toxic_top]).T.fillna(0)
entity_df.columns = ['Toxic', 'NonToxic']
```

```
entity_df['Total'] = entity_df['Toxic'] + entity_df['NonToxic']
entity_df = entity_df.sort_values(by='Total', ascending=False)

import matplotlib.pyplot as plt

# Plot
entity_df[['Toxic', 'NonToxic']].head(15).plot(
    kind='barh',
    figsize=(10, 8),
    color=['#D32F2F', '#4CAF50']
)
plt.title('Top Named Entities in Toxic vs Non-Toxic Comments')
plt.xlabel('Frequency')
plt.ylabel('Named Entity')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



Toxicity Rate by Region

```
subreddit_region_map = {
    'Toronto': 'Toronto',
    'Ottawa': 'Ottawa',
    'vancouver': 'Vancouver',
    'Ontario': 'Ontario',
    'Alberta': 'Alberta',
    'Canada': 'Canada',
    'CanadianPolitics': 'Canada',
    'Conservative': 'Canada',
    'Democrats': 'USA',
    'Republican': 'USA',
    'Worldnews': 'International',
    # Add other mappings as needed
}



df['region'] = df['subreddit'].map(subreddit_region_map).fillna('Other')



region_toxicity = df.groupby(['region', 'Toxic/NonToxic']).size().unstack(fill_value=0)
region_toxicity['Total'] = region_toxicity.sum(axis=1)
region_toxicity['Toxicity_Rate'] = region_toxicity['Toxic'] / region_toxicity['Total']
region_toxicity = region_toxicity.sort_values(by='Toxicity_Rate', ascending=False)



import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
region_toxicity['Toxicity_Rate'].plot(kind='bar', color='#D32F2F')
plt.title('Toxicity Rate by Region')
plt.ylabel('Proportion of Toxic Comments')
plt.xlabel('Region')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```




```
region_texts = df.groupby('region')['clean_comment'].apply(lambda x: ' '.join(x))


from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=100, stop_words='english')
tfidf_matrix = vectorizer.fit_transform(region_texts)
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), index=region_texts.index, columns=vectorizer.get_feature_names_out())


top_terms_per_region = {}

for region in tfidf_df.index:
    top_words = tfidf_df.loc[region].sort_values(ascending=False).head(5)
    top_terms_per_region[region] = top_words
    print(f"\n🔥 {region} Top Topics:\n", top_words)
```

```
🔥 Alberta Top Topics:
 canada          0.387970
conservative    0.248851
canadian        0.237527
trump           0.223238
government      0.193311
Name: Alberta, dtype: float64

🔥 Canada Top Topics:
 canada        0.361521
trump         0.352641
canadian      0.270401
country       0.213515
government    0.181726
Name: Canada, dtype: float64
```

```
🔥 International Top Topics:
 trump         0.324394
country       0.320985
world         0.273832
government    0.197704
fuck          0.196000
Name: International, dtype: float64

🔥 Ontario Top Topics:
 canada        0.276046
work          0.245531
fuck          0.234230
canadian      0.233947
government    0.227731
Name: Ontario, dtype: float64

🔥 Other Top Topics:
 trump         0.550857
american      0.193784
question      0.178586
state         0.175900
republican    0.166466
Name: Other, dtype: float64

🔥 Ottawa Top Topics:
 police      0.272558
city        0.254807
canada      0.221596
canadian    0.199838
work        0.199265
Name: Ottawa, dtype: float64

🔥 Toronto Top Topics:
 city        0.454785
work        0.240866
look        0.206221
canada      0.199072
canadian    0.187523
Name: Toronto, dtype: float64
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

toronto_comments = df[df['region'] == 'Toronto']['clean_comment']
vec = CountVectorizer(stop_words='english', max_df=0.9, min_df=5)
X = vec.fit_transform(toronto_comments)

lda = LatentDirichletAllocation(n_components=3, random_state=42)
lda.fit(X)

# Show top words per topic
for idx, topic in enumerate(lda.components_):
    top_words = [vec.get_feature_names_out()[i] for i in topic.argsort()[-10:]]
    print(f"🧠 Toronto Topic #{idx + 1}: {', '.join(top_words[::-1])}")
```

```
🧠 Toronto Topic #1: city, dog, toronto, removed, vote, canada, amazing, mayor, best, shit
🧠 Toronto Topic #2: toronto, cop, canadian, look, remember, post, wow, american, new, saw
🧠 Toronto Topic #3: bike, lane, car, work, city, ford, traffic, road, job, street
```

```
# !pip install pyLDAvis --upgrade
```

```
import pyLDAvis
import pyLDAvis.lda_model

pyLDAvis.lda_model.prepare
```

```
pyLDAvis.lda_model.prepare
def prepare(lda_model, dtm, vectorizer, **kwargs)

Create Prepared Data from sklearn's LatentDirichletAllocation and CountVectorizer.
```

```
from gensim import corpora, models
from gensim.utils import simple_preprocess
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis
import pandas as pd

# Step 1: Add region name as prefix to every comment (optional but helps with interpretability)
df['lda_text'] = df.apply(lambda row: f"{row['region']} {row['clean_comment']}", axis=1)

# Step 2: Preprocess all comments (tokenization)
all_comments = df['lda_text'].dropna().tolist()
texts = [simple_preprocess(comment) for comment in all_comments]

# Step 3: Create dictionary and corpus
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

# Step 4: Train LDA model (you can increase num_topics for more depth)
lda_model = models.LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=10,  # Increase if needed
    passes=10,
    random_state=42
)

# Step 5: Create and save the interactive visualization
panel = gensimvis.prepare(lda_model, corpus, dictionary)
pyLDAvis.save_html(panel, '/content/drive/MyDrive/Colab Notebooks/ALL_regions_LDA.html')
print("✅ All-region LDA visualization saved to: ALL_regions_LDA.html")
```

```
✅ All-region LDA visualization saved to: ALL_regions_LDA.html
```