

Portfolio Assignment 3 - Scraping

```
In [ ]: # Install packages
import praw
import pandas as pd
from praw.models import MoreComments
import json
import os
import datetime
```

API

```
In [ ]: # The weird letters are the credentials from my API

reddit = praw.Reddit(client_id='hQafbgnPnl6Jl_wabPofuA',
                      client_secret='OZ0EsYse7J-JiYE7kPk3mddQgMZ2nA',
                      user_agent='WebScraper/Own-Biscotti6249')

# to verify whether the instance is authorized instance or not
print(reddit.read_only)
```

Scraping

```
In [ ]: # convert data
def date_to_timestamp(date_str):
    return int(datetime.datetime.strptime(date_str, "%Y-%m-%d").timestamp())

# gets posts within a subreddit and date range
def fetch_submission_ids(subreddit_name, start_timestamp, end_timestamp):
    submission_ids = []
    subreddit = reddit.subreddit(subreddit_name)
    print(f"Fetching posts from r/{subreddit_name} between {start_timestamp} and {end_timestamp}")

    for submission in subreddit.new(limit=None): # posts in chronological order
        # dont get posts out of the desired time range
        if submission.created_utc < start_timestamp:
            break
        if start_timestamp <= submission.created_utc <= end_timestamp:
            print(f"Found post: {submission.title} (ID: {submission.id})")
            submission_ids.append(submission.id)

    if not submission_ids:
        print("No posts found in the specified date range.")
    return submission_ids

# fetch comments and their replies
def fetch_comments_and_replies(comment, submission_id, submission_title, start_timestamp, end_timestamp):
    if start_timestamp <= comment.created_utc <= end_timestamp:
        comment_body = comment.body if comment.body != '[deleted]' else None
        comment_author = comment.author.name if comment.author else None

    comment_data = {
        "submission_id": submission_id,
        "submission_title": submission_title,
        "comment_id": comment.id,
        "comment": comment_body,
```

```

        "author": comment.author,
        "created": datetime.datetime.fromtimestamp(comment.created_utc).strftime(
        "upvotes": comment.score,
        "depth": depth,
        "replies": []
    }

    if comment.replies:
        for reply in comment.replies:
            if isinstance(reply, praw.models.Comment) and start_timestamp <= repl
                comment_data['replies'].append(fetch_comments_and_replies(reply,

    return comment_data
return None

# Fetch comments from specific submissions
def fetch_comments_from_submissions(submission_ids, start_timestamp, end_timestamp):
    all_comments = []

    for submission_id in submission_ids:
        try:
            submission = reddit.submission(id=submission_id)
            print(f"Fetching comments from: {submission.title} (ID: {submission.id})"

            submission.comments.replace_more(limit=None)
            for comment in submission.comments.list():
                comment_data = fetch_comments_and_replies(comment, submission.id, sub
                if comment_data:
                    all_comments.append(comment_data)

        except Exception as e:
            print(f"An error occurred while fetching comments from {submission_id}: {

    return all_comments

# save to JSON
def save_comments_to_json(comments, json_file_name):
    folder_path = 'reddit_data_new'
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

    json_file_path = os.path.join(folder_path, json_file_name)

    with open(json_file_path, 'w', encoding='utf-8') as json_file:
        json.dump(comments, json_file, ensure_ascii=False, indent=4)

    print(f"Comments saved to {json_file_path}")

# putting it all together
def main(subreddit_name, start_date, end_date, json_file_name):
    start_timestamp = date_to_timestamp(start_date)
    end_timestamp = date_to_timestamp(end_date)

    # posts from the subreddit
    submission_ids = fetch_submission_ids(subreddit_name, start_timestamp, end_timest

    # comments from the sposts
    comments = fetch_comments_from_submissions(submission_ids, start_timestamp, end_t

    # Save
    save_comments_to_json(comments, json_file_name)

```

```
In [ ]: # Run the script for democrats
if __name__ == "__main__":
    subreddit_name = "democrats"
    start_date = "2024-11-06"
    end_date = "2024-11-18"
    json_file_name = "democrat_comments.json"

    main(subreddit_name, start_date, end_date, json_file_name)
```

```
In [ ]: # Run the script for republicans
if __name__ == "__main__":
    subreddit_name = "republican"
    start_date = "2024-11-05" #
    end_date = "2024-11-19"
    json_file_name = "republican_comments.json"

    main(subreddit_name, start_date, end_date, json_file_name)
```

preprocessing_analysis_final

December 1, 2024

1 Portfolio Assignment 3 - Preprocessing and Sentiment Analysis

```
[ ]: # import packages
import json
import string
import nltk
from nltk.tokenize import word_tokenize
import re
from nltk.stem import WordNetLemmatizer
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import numpy as np
import pandas as pd
from datetime import datetime
import seaborn as sns
import matplotlib.pyplot as plt
import urllib.request
import csv
from scipy.special import softmax
```

2 Loading in data

```
[ ]: # loading the data in for liberals
with open('reddit_data_new/democrat_comments.json') as f:
    data_lib = json.load(f)

# loading the data in for conservative
with open('reddit_data_new/republican_comments.json') as f:
    data_con = json.load(f)
```

3 Preparing the twitter-roberta-base model for sentiment analysis

```
[ ]: # load the model in and specify its task as sentiment
task='sentiment'
MODEL = f"cardiffnlp/twitter-roberta-base-{task}"
```

```

# load huggingface's tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

# Preprocess text (username and links)
def preprocess(text):
    new_text = []

    for t in text.split(" "):
        t = '@user' if t.startswith('@') and len(t) > 1 else t
        t = 'http' if t.startswith('http') else t
        new_text.append(t)
    return " ".join(new_text)

# map the numeric labels onto negative, neutral and positive from a document
labels=[]
mapping_link = f"https://raw.githubusercontent.com/cardiffnlp/tweeteval/main/
↳datasets/{task}/mapping.txt"
with urllib.request.urlopen(mapping_link) as f:
    html = f.read().decode('utf-8').split("\n")
    csvreader = csv.reader(html, delimiter='\t')
labels = [row[1] for row in csvreader if len(row) > 1]

# look at labels
print("Labels:", labels)

```

4 Extract comments from the json-file, which contains comments from r/conservative

```

[ ]: # Get comments from the "comment" column, including submission_id
def extract_comments(data_con):
    comments_with_timestamps = []
    for entry in data_con: # Loop through each object in the JSON
        if "comment" in entry and "created" in entry and "submission_id" in_
↳entry: # Check if all keys exist
            comments_with_timestamps.append({
                "comment": entry["comment"],
                "created": entry["created"], # Extract the created timestamp
                "submission_id": entry["submission_id"] # Extract the_
↳submission ID
            })
    return comments_with_timestamps

# Use the function
text = extract_comments(data_con)

```

```
# Let's see what it looks like
for comment in text:
    print(comment)
```

5 Sentiment Analysis for conservatives

```
[ ]: # Empty list to store the results
results = []

# Loop through each comment
for entry in text: # Assuming text contains dictionaries with "comment",
    ↪ "created", and "submission_id"
    comment = entry.get("comment") # Extract the comment
    created = entry.get("created") # Extract the created timestamp
    submission_id = entry.get("submission_id") # Extract the submission ID

    if comment and isinstance(comment, str) and comment.strip():
        preprocessed_comment = preprocess(comment)

        # Tokenize and prepare input
        encoded_input = tokenizer(preprocessed_comment, return_tensors='pt',
    ↪ truncation=True, max_length=512)
        output = model(**encoded_input)

        # Get the logits and apply softmax
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)

        # Find the top sentiment
        ranking = np.argsort(scores)[::-1]
        top_sentiment = labels[ranking[0]]
        top_score = np.round(float(scores[ranking[0]]), 4)

        # Save the result
        results.append({
            "Comment": comment,
            "Created": created, # Add the original timestamp from the JSON
            "Submission ID": submission_id, # Add the submission ID
            "Sentiment": top_sentiment,
            "Score": top_score,
            "Negative Probability": np.round(float(scores[0]), 4),
            "Neutral Probability": np.round(float(scores[1]), 4),
            "Positive Probability": np.round(float(scores[2]), 4),
        })

# Add Subreddit column
```

```

for item in results:
    item["Subreddit"] = "r/Republican"

# Convert to DataFrame
df = pd.DataFrame(results)

# Check the DataFrame
print(df)

# Save to CSV
df.to_csv("republican_sentiment_check.csv", index=False)
print("Results saved to 'republican_sentiment.csv'")

```

6 Extract comments from the json-file, which contains comments from r/liberal

```

[ ]: # use the function
text_lib = extract_comments(data_lib)

# lets see what it looks like
for comment in text_lib:
    print(comment)

```

7 Sentiment Analysis for liberals

```

[ ]: # Loop through each comment
for entry in text_lib: # Assuming text_lib contains dictionaries with
    ↪ "comment", "created", and "submission_id"
    comment = entry.get("comment") # Extract the comment
    created = entry.get("created") # Extract the created timestamp
    submission_id = entry.get("submission_id") # Extract the submission ID

    if comment and isinstance(comment, str) and comment.strip():
        preprocessed_comment = preprocess(comment)

        # Tokenize and prepare input
        encoded_input = tokenizer(preprocessed_comment, return_tensors='pt',
    ↪ truncation=True, max_length=512)
        output = model(**encoded_input)

        # Get the logits and apply softmax
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)

        # Find the top sentiment

```

```

ranking = np.argsort(scores)[::-1]
top_sentiment = labels[ranking[0]]
top_score = np.round(float(scores[ranking[0]]), 4)

# Save the result
results.append({
    "Comment": comment,
    "Created": created, # Add the original timestamp from the JSON
    "Submission ID": submission_id, # Add the submission ID
    "Sentiment": top_sentiment, # Properly assigned from labels
    "Score": top_score,
    "Negative Probability": np.round(float(scores[0]), 4),
    "Neutral Probability": np.round(float(scores[1]), 4),
    "Positive Probability": np.round(float(scores[2]), 4),
})

# Add Subreddit column
for item in results:
    item["Subreddit"] = "r/Democrats"

# Convert to DataFrame
df = pd.DataFrame(results)

# Check the DataFrame
print(df)

# Save to CSV
df.to_csv("democrat_sentiment_checkk.csv", index=False)
print("Results saved to 'democrat_sentiment.csv'")

```

8 load the data in again

```
[ ]: dfc = pd.read_csv('republican_sentiment_checkk.csv')
     dfl = pd.read_csv('democrat_sentiment_checkk.csv')
```

9 Removing bot comments

```
[ ]: # Remove rows where the 'Comment' contains "I am a bot"
     dfc = dfc[~dfc['Comment'].str.contains("I am a bot", na=False)]
     dfl = dfl[~dfl['Comment'].str.contains("I am a bot", na=False)]
```


10 Proportion of negative and positive comments before aggregating

```
[ ]: colours = ['red', 'gray', 'green']

# Calculate sentiment proportions
sentiment_counts = dfc.groupby(["Subreddit", "Sentiment"]).size().
    ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",
    ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Republican")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1) # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

```
[ ]: # Calculate sentiment proportions
sentiment_counts = dfl.groupby(["Subreddit", "Sentiment"]).size().
    ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",
    ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Democrats")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1) # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

11 Conservative line chart

```
[ ]: colors = ['green', 'red', 'blue']

# datetime format
dfc["Created"] = pd.to_datetime(dfc["Created"]) # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date # Extract date for grouping
```

```

# filter data for the specific date range
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot it
plt.figure(figsize=(12, 6))
sns.lineplot(data=filtered_dfl, x="Date", y="Score", hue="Sentiment",
             style="Subreddit", markers=True, palette = colors)
plt.title("Republican Sentiment between 6/11-18/11")
plt.xlabel("Date")
plt.ylabel("Sentiment Score")

# Set x-axis ticks
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

plt.tight_layout()
plt.legend(title="Sentiment")
plt.show()

```

```

[ ]: from matplotlib import rcParams

# change font
rcParams['font.family'] = 'Georgia'

# Define colors for sentiment categories
colors = {"positive": "green", "negative": "red", "neutral": "blue"}

# datetime
dfc["Created"] = pd.to_datetime(dfc["Created"]) # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date # Extract date for grouping

# Filter
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot sentiment scores

```

```

plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors # Use the specified color palette
)

# change titles
plt.title("Republican Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)

# Set x-axis ticks to every date in the range
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

# Remove unnecessary stuff
handles, labels = plt.gca().get_legend_handles_labels()
filtered_legend = [(h, l) for h, l in zip(handles, labels) if l in colors.
    ↪keys()]
plt.legend(*zip(*filtered_legend), title="Sentiment")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

```

```

[ ]: from matplotlib import rcParams

# change font
rcParams['font.family'] = 'Georgia'

# Define colors
colors = {"positive": "green", "negative": "red", "neutral": "blue"}

# datetime
dfc["Created"] = pd.to_datetime(dfc["Created"]) # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date # Extract date for grouping

# Filter

```

```

start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot the daily sentiment scores
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors # Use the specified color palette
)

# change titles
plt.title("Republican Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)

# Set x-axis ticks to every date in the range
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

# Remove unnecessary legend stuff
handles, labels = plt.gca().get_legend_handles_labels()
filtered_legend = [(h, l) for h, l in zip(handles, labels) if l in colors.
    ↪keys()]
plt.legend(*zip(*filtered_legend), title="Sentiment")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

```

12 Liberal line chart

```
[ ]: # Set font family
rcParams['font.family'] = 'Georgia'

# Define colors
colors = ['blue', 'green', 'red']

# datetime
dfl["Created"] = pd.to_datetime(dfl["Created"]) # Use 'Created' from JSON
dfl["Date"] = dfl["Created"].dt.date # Extract date for grouping

# filter data for range
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfl[(dfl["Date"] >= start_date) & (dfl["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot the filtered data
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors
)

# title and labels
plt.title("Democratic Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)

plt.xticks(date_range, rotation=45)

# Set explicit x-axis limits
plt.xlim(start_date, end_date)

# Adjust layout
plt.tight_layout()
```

```
plt.legend(title="Sentiment")

# Lets check the plot
plt.show()
```

13 lets aggregate the data

```
[ ]: # Aggregating data by 'Submission_ID'
def most_common_sentiment(series):
    # Get the most common sentiment
    if len(series) > 1:
        return series.value_counts().idxmax()
    else:
        return series.iloc[0]

# aggregatong
dfc = dfc.groupby('Submission ID').agg(
    Score=('Score', 'mean'), # mean for each Submission_ID
    Subreddit=('Subreddit', 'first'), # Take the first Subreddit
    Sentiment=('Sentiment', most_common_sentiment), # Take the first encoded_
    ↳value
    Created=('Created', 'first') # Take the first created timestamp
).reset_index()

# View the aggregated data
print(dfc)
```

```
[ ]: # Perform the aggregation
df1 = df1.groupby('Submission ID').agg(
    Score=('Score', 'mean'), # mean for each Submission_ID
    Subreddit=('Subreddit', 'first'), # Take the first Subreddit
    Sentiment=('Sentiment', most_common_sentiment), # Take the first encoded_
    ↳value
    Created=('Created', 'first') # Take the first created timestamp
).reset_index()
# View the aggregated data
print(df1)
```

14 Proportion of negative, netural and positive comments for republicans

```
[ ]: colours = ['red', 'gray', 'green']

# Calculate sentiment proportions
sentiment_counts = dfc.groupby(["Subreddit", "Sentiment"]).size().
    ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",
    ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Republican")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1) # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

15 Proportion of negative, netural and positive comments for democrats

```
[ ]: # Calculate sentiment proportions
sentiment_counts = dfl.groupby(["Subreddit", "Sentiment"]).size().
    ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",
    ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/democrats")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1) # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

16 Save the data

```
[ ]: # get the necessary columns
data_rep = dfc[['Score', 'Subreddit', 'Sentiment', 'Submission ID']]
data_dem = df1[['Score', 'Subreddit', 'Sentiment', 'Submission ID']]

# lets make them into 0 & 1
data_rep['Subreddit_encoded'] = 0 # Republican encoded as 0
data_dem['Subreddit_encoded'] = 1 # Democrat encoded as 1

[ ]: # save
combined_df.to_csv("combined_df.csv", index=False)

combined_df
```


Portfolio Assignment 3 - Chi square test

Nadia Hajighassem

2024-01-12

```
#install packages
pacman::p_load('car',
               'ggplot2',
               'tidyverse',
               'dplyr',
               'rcompanion',
               'hrbrthemes')
```

```
# load in data
df <- read.csv("combined_df.csv")
```

Make Contingency table

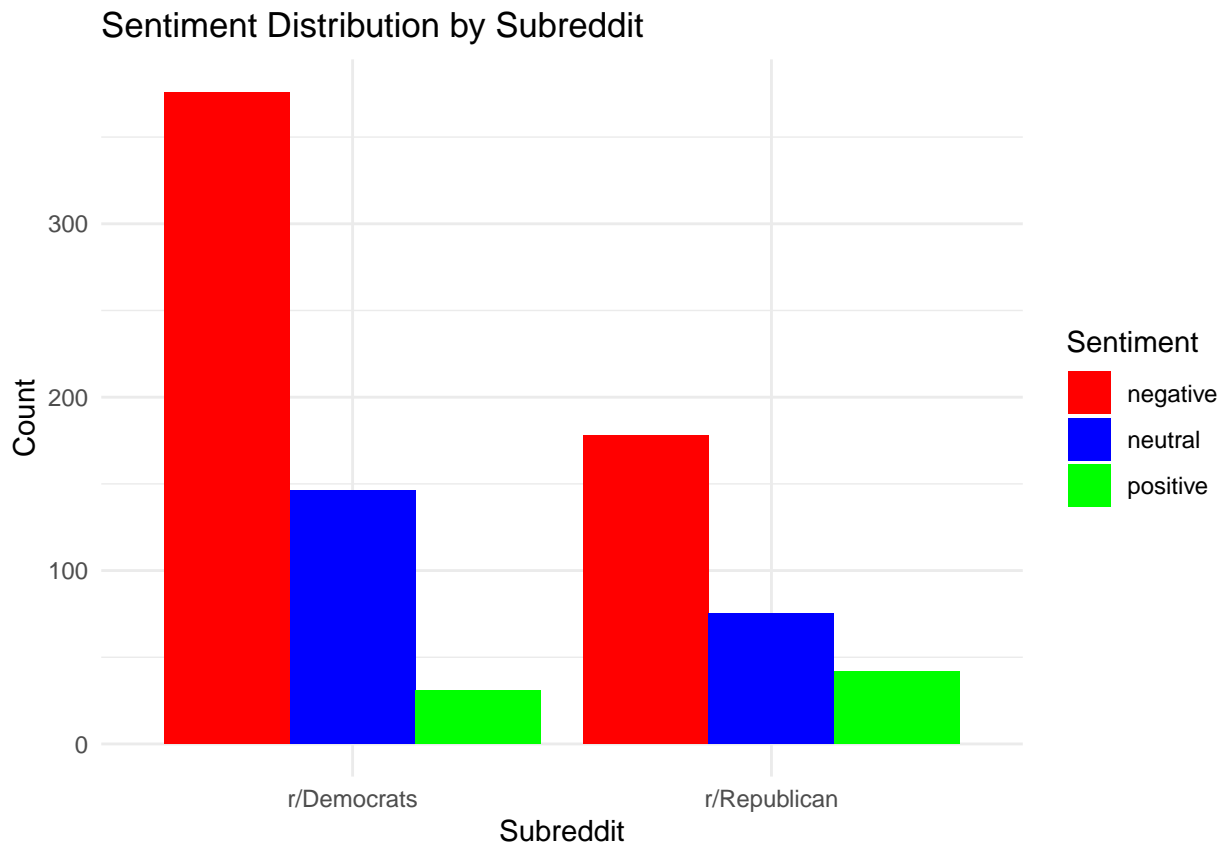
```
# prepare data for chi-square test
sentiment_table <- table(df$Subreddit, df$Sentiment)

# View the table
print(sentiment_table)
```

```
##
##           negative neutral positive
## r/Democrats      376     146       31
## r/Republican     178       75       42
```

```
color <- c("red", "blue", "green")

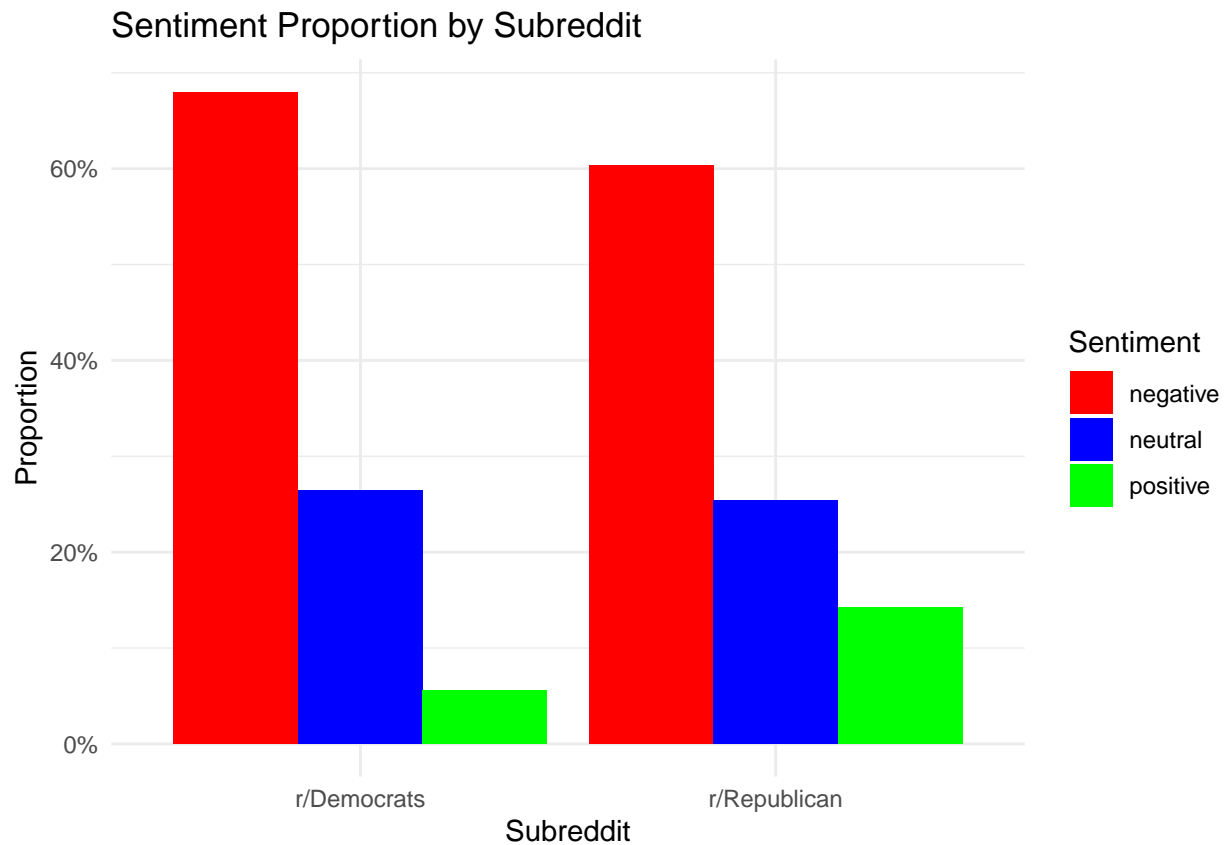
ggplot(df, aes(x = Subreddit, fill = Sentiment)) +
  geom_bar(position = "dodge") +
  scale_fill_manual(values = color) +
  labs(title = "Sentiment Distribution by Subreddit",
       x = "Subreddit",
       y = "Count") +
  theme_minimal()
```



```
# Calculate the proportion of each sentiment within each Subreddit
df_prop <- df %>%
  group_by(Subreddit, Sentiment) %>%
  summarise(Count = n()) %>%
  mutate(Proportion = Count / sum(Count))
```

```
## `summarise()` has grouped output by 'Subreddit'. You can override using the
## `.groups` argument.
```

```
# Plot the data as proportions
ggplot(df_prop, aes(x = Subreddit, y = Proportion, fill = Sentiment)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = color) +
  labs(
    title = "Sentiment Proportion by Subreddit",
    x = "Subreddit",
    y = "Proportion",
    fill = "Sentiment"
  ) +
  theme_minimal() +
  scale_y_continuous(labels = scales::percent)
```



```
# lets check the numbers
df_prop
```

```
## # A tibble: 6 x 4
## # Groups:   Subreddit [2]
##   Subreddit   Sentiment Count Proportion
##   <chr>       <chr>    <int>      <dbl>
## 1 r/Democrats negative    376      0.680
## 2 r/Democrats neutral     146      0.264
## 3 r/Democrats positive     31      0.0561
## 4 r/Republican negative    178      0.603
## 5 r/Republican neutral      75      0.254
## 6 r/Republican positive     42      0.142
```

```
# chi-square test of independence
chi_sq_test <- chisq.test(sentiment_table)
print(chi_sq_test)
```

```
##
## Pearson's Chi-squared test
##
## data:  sentiment_table
## X-squared = 18.445, df = 2, p-value = 9.88e-05
```

```

# lets check the residuals
residuals <- chi_sq_test$residuals
residuals

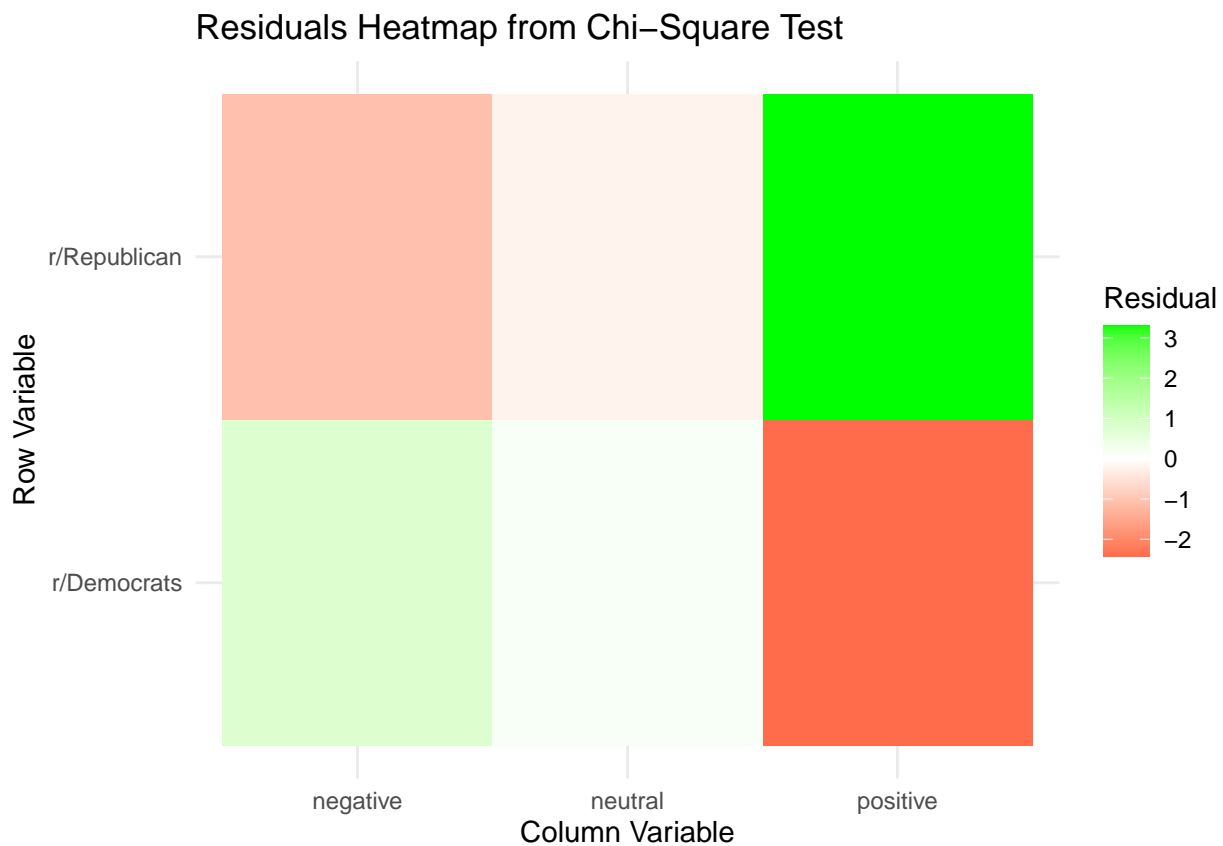
##
##           negative    neutral    positive
##  r/Democrats  0.7746543  0.1566766 -2.4066425
##  r/Republican -1.0606193 -0.2145140  3.2950585

library(ggplot2)

# convert to df
residuals_df <- as.data.frame(as.table(residuals))
colnames(residuals_df) <- c("Row", "Column", "Residual")

# heatmap
ggplot(residuals_df, aes(x = Column, y = Row, fill = Residual)) +
  geom_tile() +
  scale_fill_gradient2(low = "red", mid = "white", high = "green") + # green is positive, red is negative
  labs(title = "Residuals Heatmap from Chi-Square Test",
       x = "Column Variable",
       y = "Row Variable",
       fill = "Residual") +
  theme_minimal()

```



```
# Calculate Cramer's V
cramers_v <- cramerV(sentiment_table)

# Print the result
print(cramers_v)
```

```
## Cramer V
## 0.1475
```