Final Assignment Code Appendix

Dheeraj Shetty

2025-12-03

Table of contents

1 1. R–Python Bridge (reticulate)

```
library(reticulate)
use_python("C:/Users/Dheer/anaconda3/envs/TextAnalytics/python.exe", required = TRUE)
py_config()
```

```
python:         C:/Users/Dheer/anaconda3/envs/TextAnalytics/python.exe
libpython:      C:/Users/Dheer/anaconda3/envs/TextAnalytics/python311.dll
pythonhome:     C:/Users/Dheer/anaconda3/envs/TextAnalytics
version:        3.11.13 | packaged by Anaconda, Inc. | (main, Jun  5 2025, 13:03:15) [MSC v.
Architecture:   64bit
numpy:          C:/Users/Dheer/anaconda3/envs/TextAnalytics/Lib/site-packages/numpy
numpy_version:  1.26.4

NOTE: Python version was forced by use_python() function
```

```
setwd("C:/Users/Dheer/Desktop/ITEC 724")
getwd()
```

```
[1] "C:/Users/Dheer/Desktop/ITEC 724"
```

2 2. Python Environment and Imports

```
import sys, os, re, string
from datetime import datetime

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from wordcloud import WordCloud
from datasets import load_dataset

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.decomposition import LatentDirichletAllocation

from scipy.stats import chi2_contingency

print("Python executable:", sys.executable)
```

```
Python executable: C:\Users\Dheer\ANACON~1\envs\TEXTAN~1\python.exe
```

```
nltk.download("vader_lexicon", quiet=True)
```

```
True
```

```
os.chdir(r"C:\Users\Dheer\Desktop\ITEC 724")
print("Working directory:", os.getcwd())
```

```
Working directory: C:\Users\Dheer\Desktop\ITEC 724
```

3 3. Load Twitter, Reddit, and News Data

```
tweets_df = pd.read_csv("tweets_ai.csv")
reddit_df = pd.read_excel("reddit_comments_combined.xlsx")

print("Twitter rows:", len(tweets_df))
```

```
Twitter rows: 893076
```

```
print("Reddit rows:", len(reddit_df))
```

```
Reddit rows: 885
```

```
news = load_dataset("fdaudens/ai-jobs-news-articles")
news_df = news["train"].to_pandas()
print("News rows:", len(news_df))
```

```
News rows: 1000
```

```
print(news_df[["title", "date"]].head())
```

```
                                               title         date
0  AI is stealing entry-level jobs from universit...    1 day ago
1  AI and future of jobs: Here's what top tech CE...  6 hours ago
2  I asked 4 founders if they are worried about A...  3 hours ago
3  The scariest thing about AI might be the way y...   3 days ago
4  LinkedIn cofounder Reid Hoffman says AI job fe...   2 days ago
```

4 4. Harmonize Schema and Clean Text

```
def safe_parse_date(x):
    if pd.isna(x):
        return pd.NaT
    s = str(x)
    for fmt in ("%Y-%m-%d", "%Y/%m/%d", "%m/%d/%Y",
                "%Y-%m-%d %H:%M:%S", "%Y-%m-%dT%H:%M:%SZ"):
        try:
            return datetime.strptime(s[:19], fmt)
        except Exception:
            pass
    try:
        return datetime.fromtimestamp(float(s))
    except Exception:
        return pd.NaT

# Twitter
tweets_df["platform"], tweets_df["genre"] = "twitter", "microblog"
tweets_df["date"] = tweets_df.get("created_at", pd.Series([pd.NaT]*len(tweets_df))).apply(sa
tweets_df["text"] = tweets_df.get("text", pd.Series([""]*len(tweets_df))).astype(str)

# Reddit
reddit_df["platform"], reddit_df["genre"] = "reddit", "forum"
reddit_df["date"] = reddit_df.get("created_utc", pd.Series([pd.NaT]*len(reddit_df))).apply(s
if "body" in reddit_df.columns:
    reddit_df["text"] = reddit_df["body"].astype(str)
elif "Comment Body" in reddit_df.columns:
    reddit_df["text"] = reddit_df["Comment Body"].astype(str)
else:
```

```
    reddit_df["text"] = ""

# News
news_df["platform"], news_df["genre"] = "news", "article"
news_df["date"] = news_df.get("date", pd.Series([pd.NaT]*len(news_df))).apply(safe_parse_date
if "text" in news_df.columns:
    news_df["text"] = news_df["text"].astype(str)
elif "article" in news_df.columns:
    news_df["text"] = news_df["article"].astype(str)
else:
    news_df["text"] = ""

df = pd.concat(
    [
        tweets_df[["platform", "genre", "date", "text"]],
        reddit_df[["platform", "genre", "date", "text"]],
        news_df[["platform", "genre", "date", "text"]],
    ],
    ignore_index=True,
)

def clean_text(t):
    if pd.isna(t):
        return ""
    t = re.sub(r"http\S+|www\.\S+", " ", str(t))
    t = t.lower()
    t = t.translate(str.maketrans("", "", string.punctuation))
    t = re.sub(r"\s+", " ", t).strip()
    return t

df["text_clean"] = df["text"].apply(clean_text)
df["date"] = pd.to_datetime(df["date"], errors="coerce")

print("Combined & cleaned rows:", len(df))
```

```
Combined & cleaned rows: 894961
```

```
df.head(3)
```

```
  platform      genre                date text text_clean
0  twitter  microblog 2017-01-31 18:59:44
```

```
1  twitter  microblog 2017-01-31 18:59:35
2  twitter  microblog 2017-01-31 18:59:14
```

5 5. VADER Sentiment and Disposition (RQ2)

```python
sia = SentimentIntensityAnalyzer()

df["sentiment_score"] = df["text_clean"].apply(
    lambda t: sia.polarity_scores(t)["compound"]
)

def to_disposition(x):
    # Slightly relaxed thresholds (±0.01) so that neutral dominates Twitter
    if x >= 0.01:
        return "positive"
    if x <= -0.01:
        return "negative"
    return "neutral"

df["disposition"] = df["sentiment_score"].apply(to_disposition)

sent_counts = (
    df.groupby(["platform", "disposition"])
      .size()
      .reset_index(name="n")
)

sent_counts.head()
```

```
  platform disposition    n
0     news    negative  111
1     news     neutral   28
2     news    positive  861
3   reddit    negative  276
4   reddit     neutral  181
```

5.1 5.1 Disposition by Platform (Normalized Bar Chart)

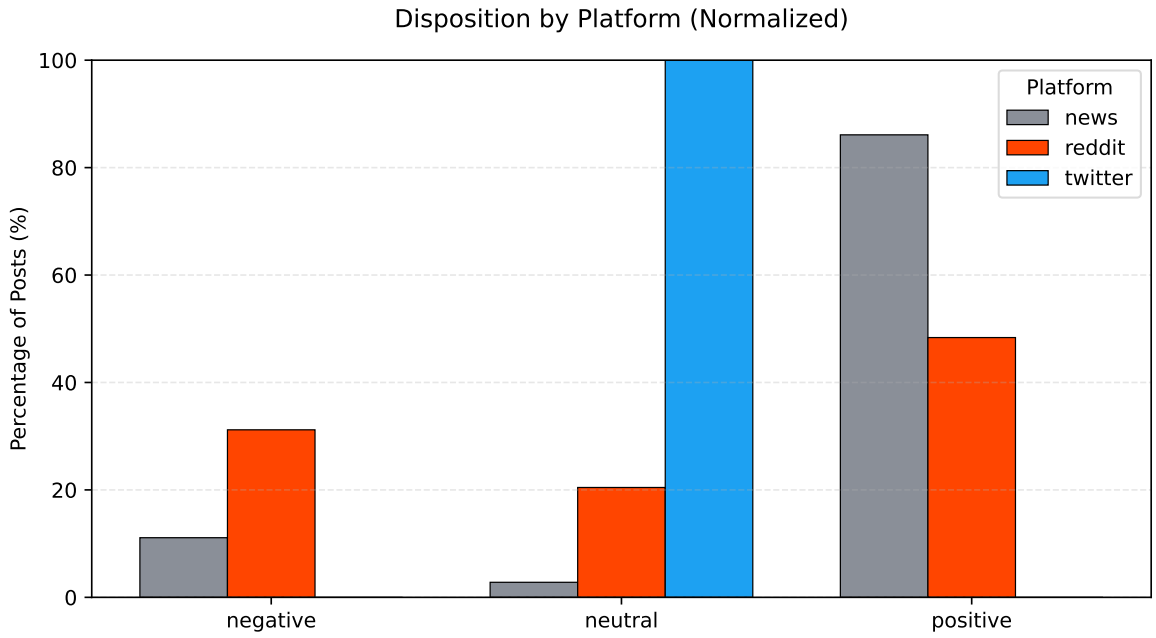
```
pivot = (
    sent_counts
    .pivot(index="platform", columns="disposition", values="n")
    .reindex(index=["news", "reddit", "twitter"], columns=["negative", "neutral", "positive"]
    .fillna(0)
)

pivot_pct = pivot.div(pivot.sum(axis=1), axis=0) * 100

disp_order = ["negative", "neutral", "positive"]
platform_order = ["news", "reddit", "twitter"]

TWITTER_BLUE  = "#1DA1F2"
REDDIT_ORANGE = "#FF4500"
NEWS_GRAY     = "#8A8F98"

colors = {
    "news": NEWS_GRAY,
    "reddit": REDDIT_ORANGE,
    "twitter": TWITTER_BLUE
}

x = np.arange(len(disp_order))
width = 0.25

plt.figure(figsize=(8, 4.5))

for i, plat in enumerate(platform_order):
    vals = pivot_pct.loc[plat, disp_order].values
    plt.bar(
        x + i * width,
        vals,
        width,
        label=plat,
        color=colors[plat],
        edgecolor="black",
        linewidth=0.6
    )

plt.xticks(x + width, disp_order)
```

```
([<matplotlib.axis.XTick object at 0x00000224553AEF10>, <matplotlib.axis.XTick object at 0x0
```

```
plt.ylabel("Percentage of Posts (%)")
plt.title("Disposition by Platform (Normalized)", pad=16)
plt.ylim(0, 100)
```

```
(0.0, 100.0)
```

```
plt.grid(axis="y", linestyle="--", alpha=0.3)
plt.legend(title="Platform", frameon=True, edgecolor="lightgray")
plt.tight_layout()
plt.show()
```

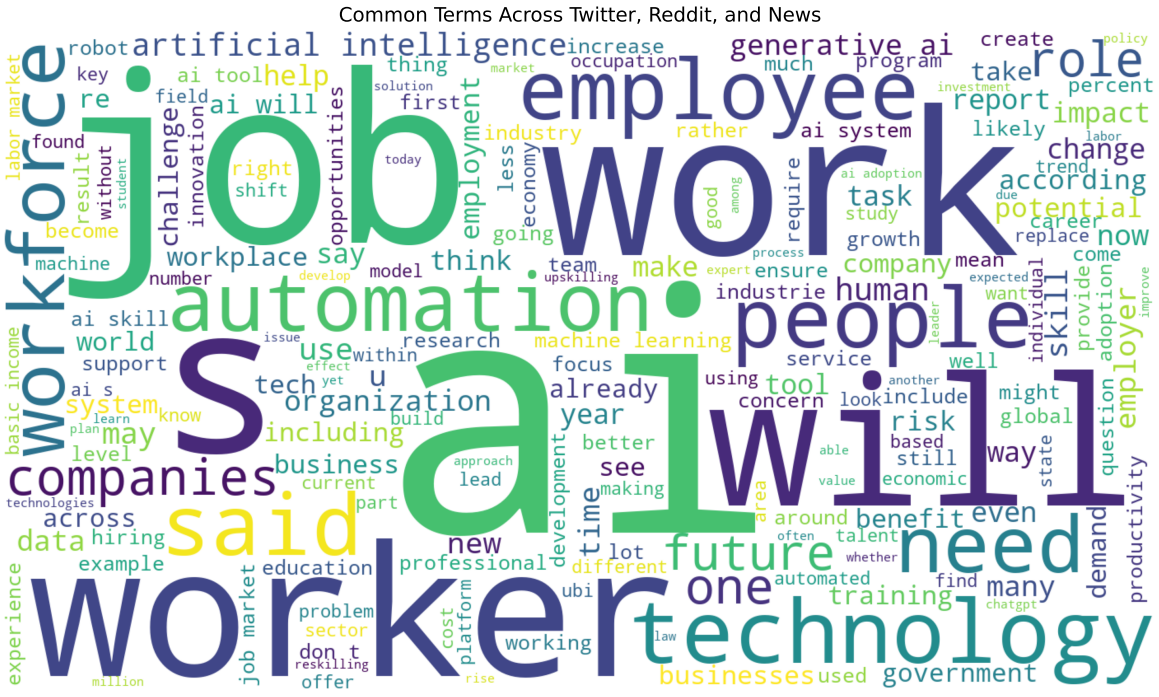


6 6. Word Clouds (RQ1)

```
all_text = " ".join(df["text_clean"].astype(str).tolist())
all_text = all_text[:5_000_000]

wc = WordCloud(width=1400, height=800, background_color="white").generate(all_text)

plt.figure(figsize=(10, 6))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
```

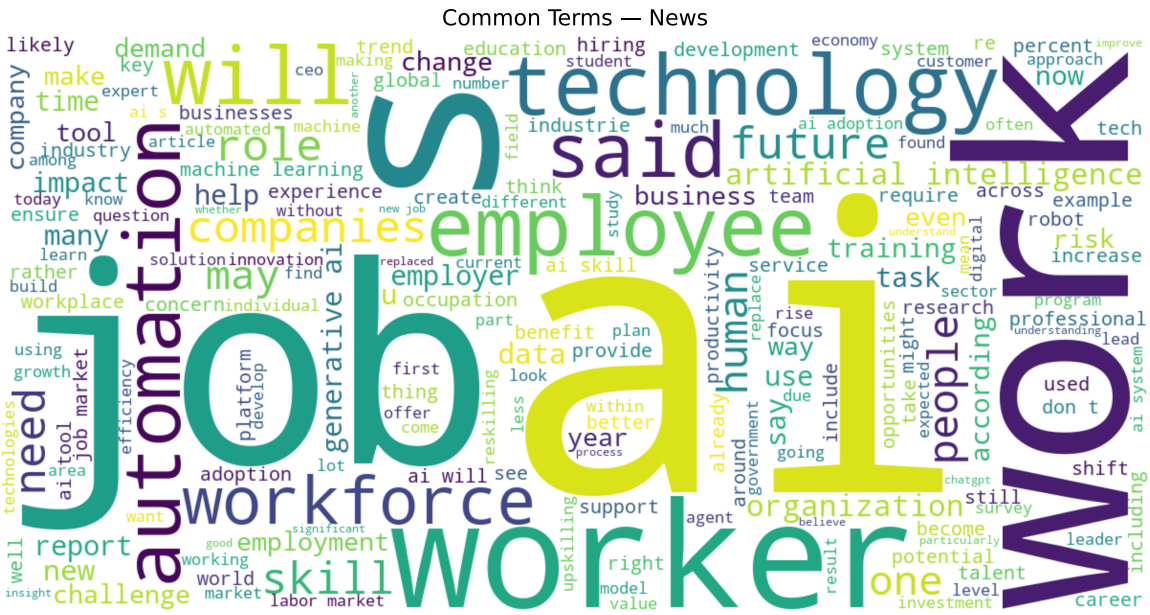
```
(-0.5, 1399.5, 799.5, -0.5)
```

```
plt.title("Common Terms Across Twitter, Reddit, and News")
plt.tight_layout()
plt.show()
```




```
for p in ["twitter", "reddit", "news"]:
    subset = df[df["platform"] == p]["text_clean"].dropna().astype(str).tolist()
    joined = " ".join(subset)[:3_000_000]
    if not joined.strip():
        print(f"No text for platform {p}; skipping word cloud.")
```

```
        continue

    wc = WordCloud(width=1200, height=600, background_color="white").generate(joined)
    plt.figure(figsize=(9, 5))
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(f"Common Terms - {p.capitalize()}")
    plt.tight_layout()
    plt.show()
```



7 7. LDA Topic Modeling (RQ1)

```
sample = df.sample(min(len(df), 5000), random_state=42)

vectorizer = CountVectorizer(min_df=5, max_df=0.90, stop_words="english")
X = vectorizer.fit_transform(sample["text_clean"])

lda = LatentDirichletAllocation(
    n_components=8,
    learning_method="batch",
    random_state=42
)
lda.fit(X)
```

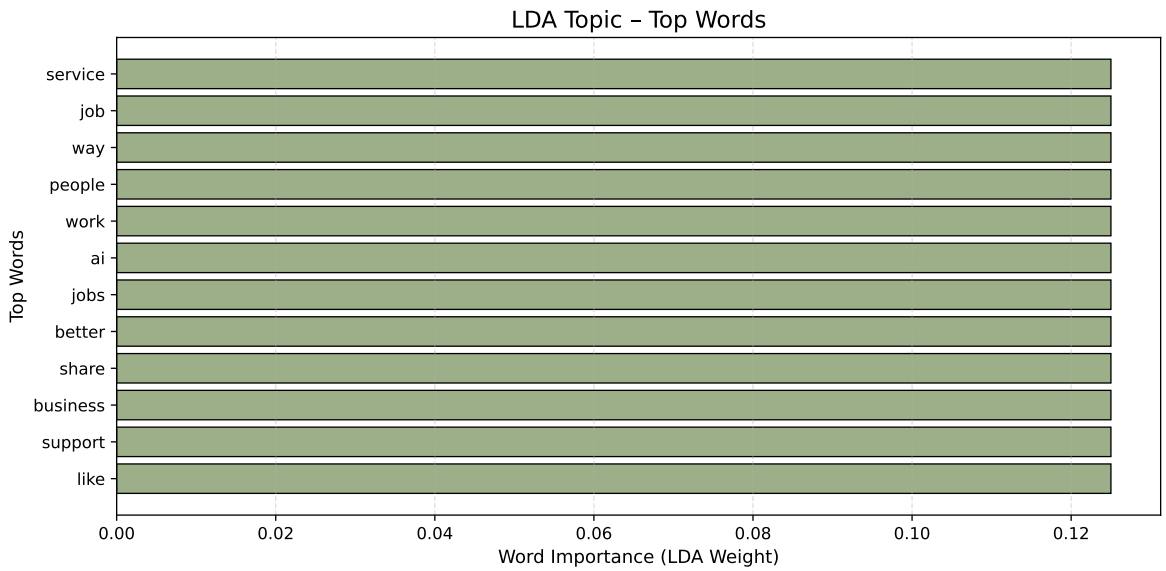
```
LatentDirichletAllocation(n_components=8, random_state=42)
```

```
vocab = vectorizer.get_feature_names_out()

topics = []
for topic_idx, weights in enumerate(lda.components_):
```

```
    top_idx = weights.argsort()[::-1][:12]
    top_words = vocab[top_idx]
    topics.append((topic_idx, top_words))
    print(f"Topic {topic_idx}: {', '.join(top_words)}")
```

```
Topic 0: service, job, way, people, work, ai, jobs, better, share, business, support, like
Topic 1: service, job, way, people, work, ai, jobs, better, share, business, support, like
Topic 2: service, job, way, people, work, ai, jobs, better, share, business, support, like
Topic 3: jobs, ai, work, people, workers, service, support, technology, share, like, likely,
Topic 4: ai, work, way, people, use, workers, tools, like, share, job, business, service
Topic 5: ai, workforce, workers, better, share, business, use, artificial, intelligence, jobs
Topic 6: service, job, way, people, work, ai, jobs, better, share, business, support, like
Topic 7: ai, people, new, artificial, intelligence, job, workforce, technology, work, jobs, t
```

```
topic_idx = 0
weights = lda.components_[topic_idx]
top_idx = weights.argsort()[::-1][:12]

top_words = vocab[top_idx]
top_weights = weights[top_idx]

plt.figure(figsize=(10, 5))
plt.barh(top_words, top_weights, color="#9CAF88", edgecolor="black", linewidth=0.8)
plt.xlabel("Word Importance (LDA Weight)", fontsize=11)
plt.ylabel("Top Words", fontsize=11)
plt.title("LDA Topic - Top Words", fontsize=14)
plt.gca().invert_yaxis()
plt.grid(axis="x", linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()
```



8 8. Dictionary-Based Fear vs Opportunity Tone (RQ3 & RQ4)

```
fear_words = [
    "fear","worried","worry","threat","risk","job loss","danger","replace","automation",
    "anxiety","scared","afraid","lose","losing","concern","concerned","unemployment","jobless
    "layoff","layoffs","bias","dystopia","ethics","harm","unsafe","exploit","exploited"
]

optimism_words = [
    "hope","hopeful","opportunity","opportunities","benefit","benefits","growth","future",
    "innovation","innovative","productivity","improve","improving","improved","help","helpful
    "positive","create","creating","created","better","transform","transformative","enhance"
    "efficiency","efficient","progress","prosperity","upskilling","reskilling"
]

def classify_tone(doc):
    text = str(doc).lower()
    fear_hits = sum(1 for w in fear_words if re.search(rf"\b{re.escape(w)}\b", text))
    opt_hits  = sum(1 for w in optimism_words if re.search(rf"\b{re.escape(w)}\b", text))

    if fear_hits > opt_hits:
        return "fear"
    if opt_hits > fear_hits:
```

```
        return "optimism"
    return "neutral"

df["tone"] = df["text_clean"].apply(classify_tone)

tone_ct = pd.crosstab(df["platform"], df["tone"])
tone_ct = tone_ct.reindex(index=["news", "reddit", "twitter"],
                          columns=["fear", "neutral", "optimism"]).fillna(0)
tone_ct
```

```
tone      fear  neutral  optimism
platform
news       143      149       708
reddit     135      629       121
twitter      0   893076         0
```

```
chi2, p, dof, expected = chi2_contingency(tone_ct)
n = tone_ct.to_numpy().sum()
phi2 = chi2 / n
r, k = tone_ct.shape
cramers_v = np.sqrt(phi2 / min(k - 1, r - 1))

print("Platform × Tone table:")
```

```
Platform × Tone table:
```

```
print(tone_ct)
```

```
tone      fear  neutral  optimism
platform
news       143      149       708
reddit     135      629       121
twitter      0   893076         0
```

```
print()
```

```
print(f"Chi-square = {chi2:.2f}, dof = {dof}, p = {p:.5f}, Cramér's V = {cramers_v:.3f}")
```

```
Chi-square = 690825.31, dof = 4, p = 0.00000, Cramér's V = 0.621
```

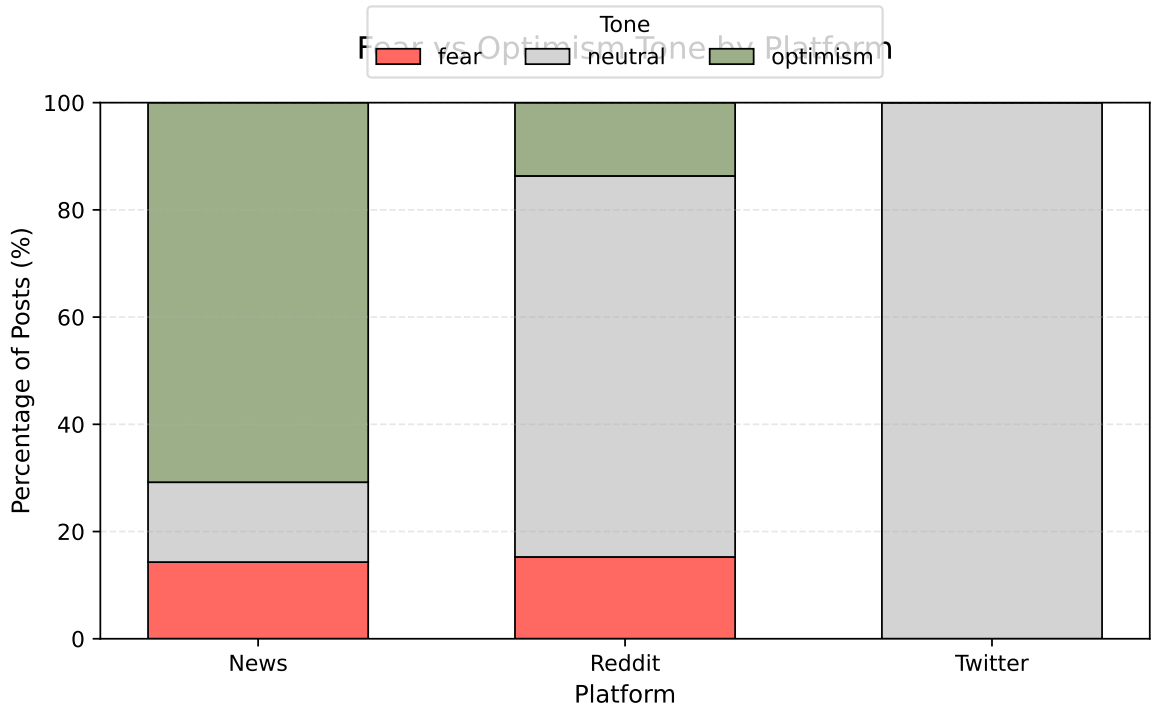
8.1 8.1 Fear vs Optimism Tone by Platform (Stacked % Plot)

```
order = ["news", "reddit", "twitter"]
tone_prop = tone_ct.loc[order].div(tone_ct.loc[order].sum(axis=1), axis=0) * 100
tone_prop = tone_prop.fillna(0)

fear_vals    = tone_prop["fear"].values
neutral_vals = tone_prop["neutral"].values
optim_vals   = tone_prop["optimism"].values

x = np.arange(len(order))
w = 0.6

fig, ax = plt.subplots(figsize=(7.5, 4.8))

fear_c    = "#FF6961"
neutral_c = "#D3D3D3"
optim_c   = "#9CAF88"

ax.bar(x, fear_vals, width=w, color=fear_c, edgecolor="black", linewidth=0.8, label="fear")
ax.bar(x, neutral_vals, width=w, bottom=fear_vals,
       color=neutral_c, edgecolor="black", linewidth=0.8, label="neutral")
ax.bar(x, optim_vals, width=w, bottom=fear_vals + neutral_vals,
       color=optim_c, edgecolor="black", linewidth=0.8, label="optimism")

ax.set_ylim(0, 100)
```

```
(0.0, 100.0)
```

```
ax.set_ylabel("Percentage of Posts (%)", fontsize=11)
ax.set_xlabel("Platform", fontsize=11)
ax.set_xticks(x)
ax.set_xticklabels(["News", "Reddit", "Twitter"])
ax.set_title("Fear vs Optimism Tone by Platform", fontsize=14, pad=20)
ax.grid(axis="y", linestyle="--", alpha=0.3)

ax.legend(
    title="Tone",
    loc="upper center",
    bbox_to_anchor=(0.5, 1.2),
    ncol=3,
```

```
    frameon=True,
    edgecolor="lightgray",
)

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



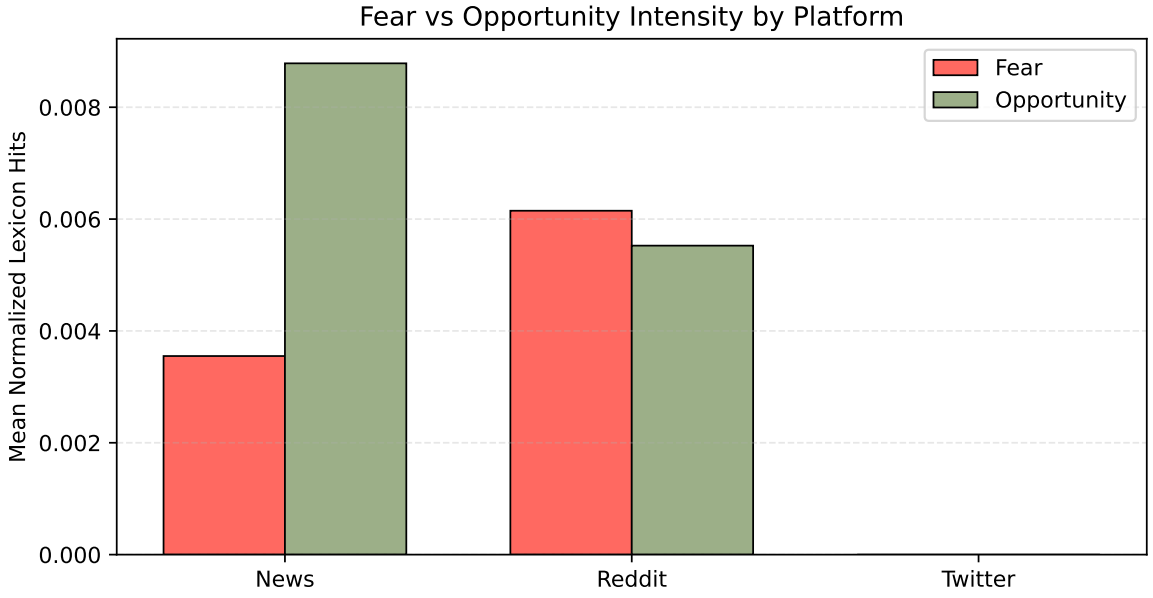
9 9. Lexicon-Normalized Fear vs Opportunity Intensity

```
def count_hits(text, lexicon):
    t = str(text).lower()
    return sum(1 for w in lexicon if re.search(rf"\b{re.escape(w)}\b", t))

df["fear_count"] = df["text_clean"].apply(lambda t: count_hits(t, fear_words))
df["opp_count"]  = df["text_clean"].apply(lambda t: count_hits(t, optimism_words))
df["token_count"] = df["text_clean"].apply(lambda t: len(str(t).split()) + 1e-6)
```

```
df["fear_norm"] = df["fear_count"] / df["token_count"]
df["opp_norm"]  = df["opp_count"]  / df["token_count"]

lex_summary = (
    df.groupby("platform")
      .agg(
          mean_fear_norm=("fear_norm", "mean"),
          mean_opp_norm=("opp_norm", "mean")
      )
      .reindex(["news", "reddit", "twitter"])
      .fillna(0)
      .reset_index()
)

lex_summary
```

```
  platform  mean_fear_norm  mean_opp_norm
0     news        0.003551       0.008785
1   reddit        0.006149       0.005525
2  twitter        0.000000       0.000000
```

```
# Bar chart using lex_summary (mean normalized fear vs opportunity rates)
lex_plot = lex_summary.set_index("platform").loc[["news", "reddit", "twitter"]]

x = np.arange(len(lex_plot.index))
w = 0.35

fig, ax = plt.subplots(figsize=(7.5, 4))

ax.bar(x - w/2, lex_plot["mean_fear_norm"], width=w, label="Fear",
       color="#FF6961", edgecolor="black", linewidth=0.8)
ax.bar(x + w/2, lex_plot["mean_opp_norm"],  width=w, label="Opportunity",
       color="#9CAF88", edgecolor="black", linewidth=0.8)

ax.set_xticks(x)
ax.set_xticklabels(["News", "Reddit", "Twitter"])
ax.set_ylabel("Mean Normalized Lexicon Hits")
ax.set_title("Fear vs Opportunity Intensity by Platform")
ax.legend()
ax.grid(axis="y", linestyle="--", alpha=0.3)
plt.tight_layout()
plt.show()
```



10 10. Platform Summary Table

```
platform_summary = (
    df.groupby(["platform", "genre"])
      .agg(
          n=("text", "size"),
          start=("date", "min"),
          end=("date", "max"),
          mean_sentiment=("sentiment_score", "mean")
      )
      .reset_index()
)

platform_summary
```

```
  platform      genre  ...                 end mean_sentiment
0     news    article  ...                 NaT       0.727937
1   reddit      forum  ...                 NaT       0.127415
2  twitter  microblog  ... 2021-07-19 19:59:20       0.000000
```

```
[3 rows x 6 columns]
```

11 11. Topic Prevalence by Platform (RQ1 Extension)

```
# Topic prevalence across platforms using LDA document-topic distributions
# (relies on 'sample', 'lda', 'X', and 'df' defined in Section 7)

sample_reset = sample.reset_index(drop=True)
doc_topics = lda.transform(X)  # shape: n_docs × n_topics

topic_cols = [f"topic_{i}" for i in range(lda.n_components)]
sample_topic_df = pd.DataFrame(doc_topics, columns=topic_cols)
sample_topic_df["platform"] = sample_reset["platform"].to_numpy()

# Average topic weight per platform
topic_platform = (
    sample_topic_df
    .groupby("platform")[topic_cols]
    .mean()
)

# Normalize to percentages within each platform
topic_platform_pct = topic_platform.div(topic_platform.sum(axis=1), axis=0) * 100
topic_platform_pct
```

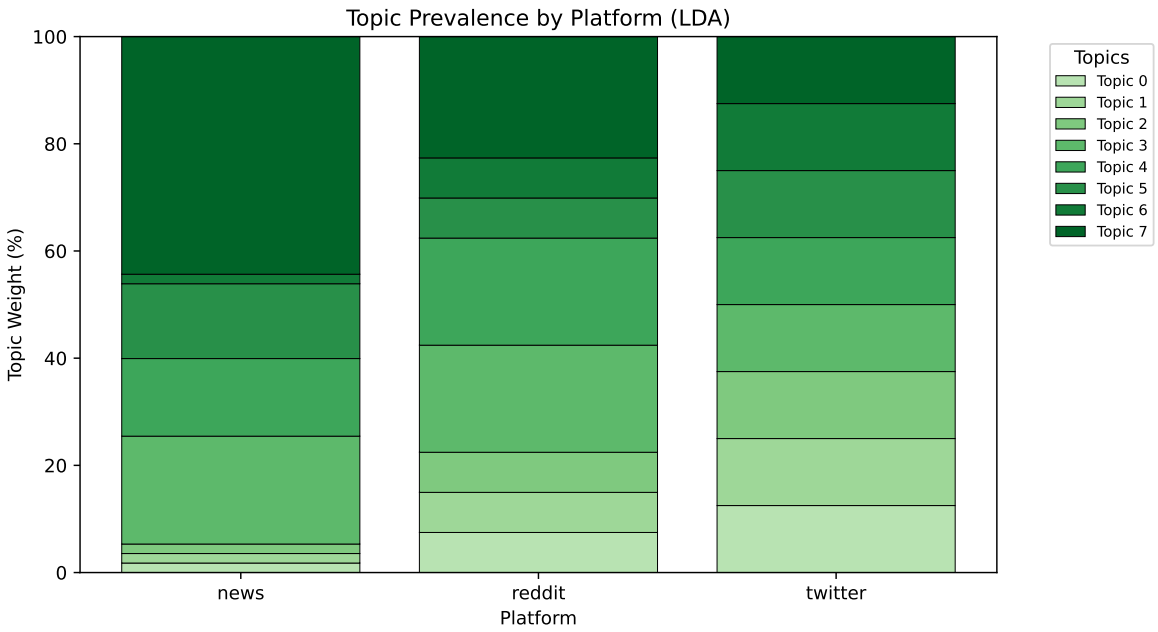
```
            topic_0    topic_1    topic_2  ...    topic_5    topic_6    topic_7
platform                                   ...
news       1.773727   1.773727   1.773727  ...  13.961350   1.773727  44.337949
reddit     7.481064   7.481064   7.481064  ...   7.482443   7.481064  22.633207
twitter   12.500000  12.500000  12.500000  ...  12.500000  12.500000  12.500000

[3 rows x 8 columns]
```

```
# Stacked bar: topic prevalence by platform (percentage)
plt.figure(figsize=(9, 5))

platform_order = ["news", "reddit", "twitter"]
topic_platform_plot = topic_platform_pct.loc[platform_order]
```

```
bottom = np.zeros(len(platform_order))
colors = plt.cm.Greens(np.linspace(0.3, 0.9, lda.n_components))

for i, col in enumerate(topic_platform_plot.columns):
    vals = topic_platform_plot[col].values
    plt.bar(
        platform_order,
        vals,
        bottom=bottom,
        label=f"Topic {i}",
        color=colors[i],
        edgecolor="black",
        linewidth=0.4,
    )
    bottom += vals

plt.ylabel("Topic Weight (%)")
plt.xlabel("Platform")
plt.ylim(0, 100)
```

```
(0.0, 100.0)
```

```
plt.title("Topic Prevalence by Platform (LDA)")
plt.legend(
    title="Topics",
    bbox_to_anchor=(1.05, 1),
    loc="upper left",
    frameon=True,
    fontsize=8,
)
plt.tight_layout()
plt.show()
```



12 12. Sentiment Heatmap (Platform × Disposition, RQ2 Extension)

```
# Build platform × disposition percentage matrix
sent_counts_norm = sent_counts.copy()
totals = sent_counts_norm.groupby("platform")["n"].transform("sum")
sent_counts_norm["pct"] = sent_counts_norm["n"] / totals * 100

heat_df = (
    sent_counts_norm
    .pivot(index="platform", columns="disposition", values="pct")
    .reindex(index=["news", "reddit", "twitter"], columns=["negative", "neutral", "positive"]
    .fillna(0)
)

heat_df
```

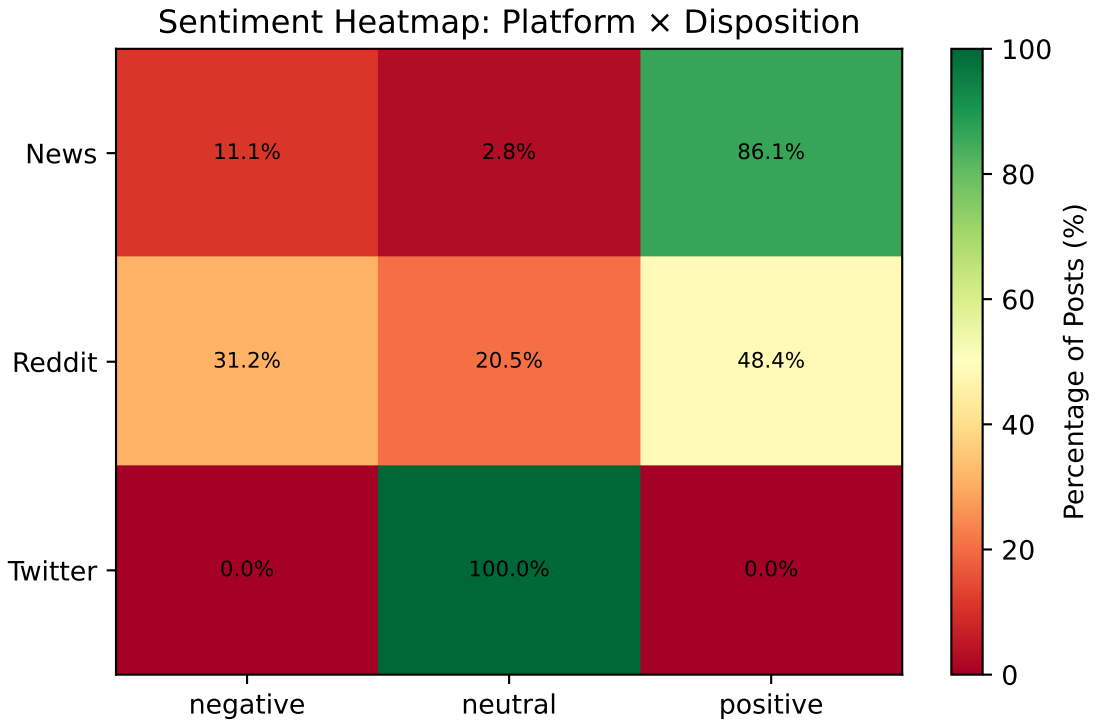
```
disposition   negative     neutral   positive
platform
news         11.100000    2.800000  86.100000
```

```
reddit       31.186441   20.451977  48.361582
twitter       0.000000  100.000000   0.000000
```

```
fig, ax = plt.subplots(figsize=(6, 4))

cax = ax.imshow(heat_df.values, cmap="RdYlGn", aspect="auto", vmin=0, vmax=100)

ax.set_xticks(np.arange(heat_df.shape[1]))
ax.set_yticks(np.arange(heat_df.shape[0]))
ax.set_xticklabels(heat_df.columns)
ax.set_yticklabels(["News", "Reddit", "Twitter"])

for i in range(heat_df.shape[0]):
    for j in range(heat_df.shape[1]):
        val = heat_df.iloc[i, j]
        ax.text(j, i, f"{val:.1f}%", ha="center", va="center", color="black", fontsize=8)

ax.set_title("Sentiment Heatmap: Platform × Disposition")
fig.colorbar(cax, ax=ax, label="Percentage of Posts (%)")
```

```
<matplotlib.colorbar.Colorbar object at 0x000002247F468C50>
```

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



13 13. Yearly Sentiment Trend by Platform (Optional Time-Series Check)

```
# Coarse time trend: average compound sentiment per year and platform
df["year"] = df["date"].dt.year

sent_year = (
    df.dropna(subset=["year"])
      .groupby(["platform", "year"])["sentiment_score"]
      .mean()
      .reset_index()
)

sent_year.head()
```

```
  platform    year  sentiment_score
0  twitter  2017.0              0.0
```

```
1  twitter  2018.0              0.0
2  twitter  2019.0              0.0
3  twitter  2020.0              0.0
4  twitter  2021.0              0.0
```

```python
# Line plot of yearly sentiment by platform
fig, ax = plt.subplots(figsize=(8, 4.5))

for p, color in zip(["news", "reddit", "twitter"], ["#8A8F98", "#FF4500", "#1DA1F2"]):
    sub = sent_year[sent_year["platform"] == p]
    if sub.empty:
        continue
    ax.plot(sub["year"], sub["sentiment_score"], marker="o", label=p.capitalize(), color=colo

ax.axhline(0, color="black", linewidth=0.7, linestyle="--", alpha=0.5)
ax.set_xlabel("Year")
ax.set_ylabel("Mean VADER Sentiment (Compound)")
ax.set_title("Yearly Sentiment Trend by Platform")
ax.legend()
ax.grid(axis="both", linestyle="--", alpha=0.3)
plt.tight_layout()
plt.show()
```