Python Learning Analytics & Sentiment Analysis Project

Erich Assuncao

November, 2025

A graduate-level portfolio case study using Python, Tableau, and statistical analysis to examine participation patterns, reply dynamics, timing effects, word count, and sentiment in online discussion data. Originally developed as part of graduate coursework in Learning and Knowledge Analytics.

Abstract

This report analyzes my participation in COMP 683 over Weeks 1–6 using a combination of custom Python scripts and Tableau visualizations. Discussion forum PDFs were exported from Brightspace and converted to structured data using a Python pipeline (pdfplumber for extraction, NLTK/VADER for sentiment, and pandas for processing). I computed post-level features (word count, sentiment score and label, day-in-study-week) and derived a reply-count measure that excludes self-replies. Four research questions guided the analysis: (RQ1) my posting frequency versus peers; (RQ2) factors associated with reply counts (timing, sentiment, word count); (RQ3) my total replies received versus peers; and (RQ4) comparisons of sentiment between my posts and peers'. Findings show my posting frequency was consistently at or above the peer average; earlier posts received more replies (moderate negative correlation between day and replies), while post length and sentiment showed no relationship to replies; my original posts received more replies than the cohort average and median; and sentiment distributions did not differ significantly between me and peers (both strongly positive overall). These results suggest that early and steady participation matters more for engagement than post length or tone. Limitations include small sample sizes and positively skewed sentiment that constrains variability.

Introduction

Motivation

Participation data provides tangible evidence of engagement and can guide reflective improvement. By quantifying posting patterns, timing, and tone, we can separate assumptions (e.g., "longer posts get more replies") from patterns supported by data. In a collaborative graduate course, such evidence supports fair self-assessment, constructive feedback, and targeted strategies for future discussions.

Goals and Research Questions

This project addresses the following research questions:

- RQ1: How does my weekly posting activity compare to that of my peers?
- RQ2: What factors are associated with the number of replies an original post receives?
- • RQ2a: Is timing within the study week associated with reply count?
- • RQ2b: Is sentiment associated with reply count?
- • RQ2c: Is word count associated with reply count?
- RQ3: How does the total number of replies my posts received compare to peers?
- RQ4: What is the overall sentiment of my posts compared to that of my peers?

Report Structure

The report proceeds as follows. The Research Design section describes data sources, tools, analytical processes, and methodology. The Results section presents findings for each research question with narrative descriptions of figures. The Discussion interprets the

results and limitations. The Conclusions summarize contributions and potential future directions.

Research Design

Project Overview

I exported Brightspace discussion forums for Weeks 1–6 as PDFs, then used a custom Python pipeline to convert unstructured text into a structured dataset. A second script derived reply counts for original posts while excluding self-replies. Tableau dashboards and statistical tests (in Python) were then used to explore relationships and compare my participation to peers'.

Data Sources

- Brightspace discussion forums (Weeks 1–6), exported as PDF files.
- Structured dataset produced from the PDFs containing: week, thread_title, author, posted_at, day_in_study_week, post_type, word_count, sentiment (VADER compound), sentiment_label, and post_text.

Note: Even though the assignment required the inclusion of data from both Brightspace and the Landing (Weeks 1–6), the Landing platform only contained personal introductions rather than course-related discussion content. For this reason, the analysis focuses exclusively on the Brightspace data for Weeks 1–6, which provided substantive posts suitable for quantitative and qualitative analysis.

Tools and How They Were Used

- Python (pdfplumber) to extract and normalize text from PDF exports.
- Python (NLTK/VADER) to compute sentiment scores and labels.
- Python (pandas) for data processing and derivations (e.g., day-in-study-week, word count, reply counting).
- Tableau for visual exploration and charts (bar charts with CIs, scatter plots with trend lines, distributions).

Analytics Processes/Procedures

The workflow follows a standard analytics process resembling CRISP–DM: (1) business understanding (define RQs focused on participation), (2) data understanding (inspect PDF exports and forum structure), (3) data preparation (PDF parsing, cleaning, variable engineering), (4) modeling/analysis (visualizations, correlations, confidence intervals, chi-square), and (5) evaluation and reporting (interpretation and write-up).

Research Methodology

- Extraction: Parse Brightspace PDFs using robust header patterns and normalization to separate thread titles, authors, dates, and body text.

- Feature Engineering: Compute word_count, VADER sentiment (compound), sentiment_label thresholds; compute day_in_study_week anchored to course calendar.
- Classification: Label first author entry per thread as original; subsequent entries as replies.
- Reply Counting: Aggregate replies to each original post, excluding any replies by the original author.
- Statistical Analysis: Pearson correlations with p-values; 95% confidence intervals for means; chi-square for sentiment-category distributions.
- Visualization: Tableau charts for weekly activity, scatter plots with trend lines, and distribution comparisons.

Results

RQ1: Weekly Posting Activity vs. Peers

My weekly posting counts over Weeks 1–6 were: 3, 3, 2, 2, 2, and 3. Peer averages were: 2.083 (Week 1), 1.833 (Week 2), 1.5 (Week 3), 1.333 (Week 4), 2.25 (Week 5), and 1.667 (Week 6). A bar chart with 95% confidence intervals (CIs) showed my activity was consistently at or above the peer average in all weeks. Week 5 saw the only peer spike above my count. Overall, my participation was sustained and generally higher than peers'.

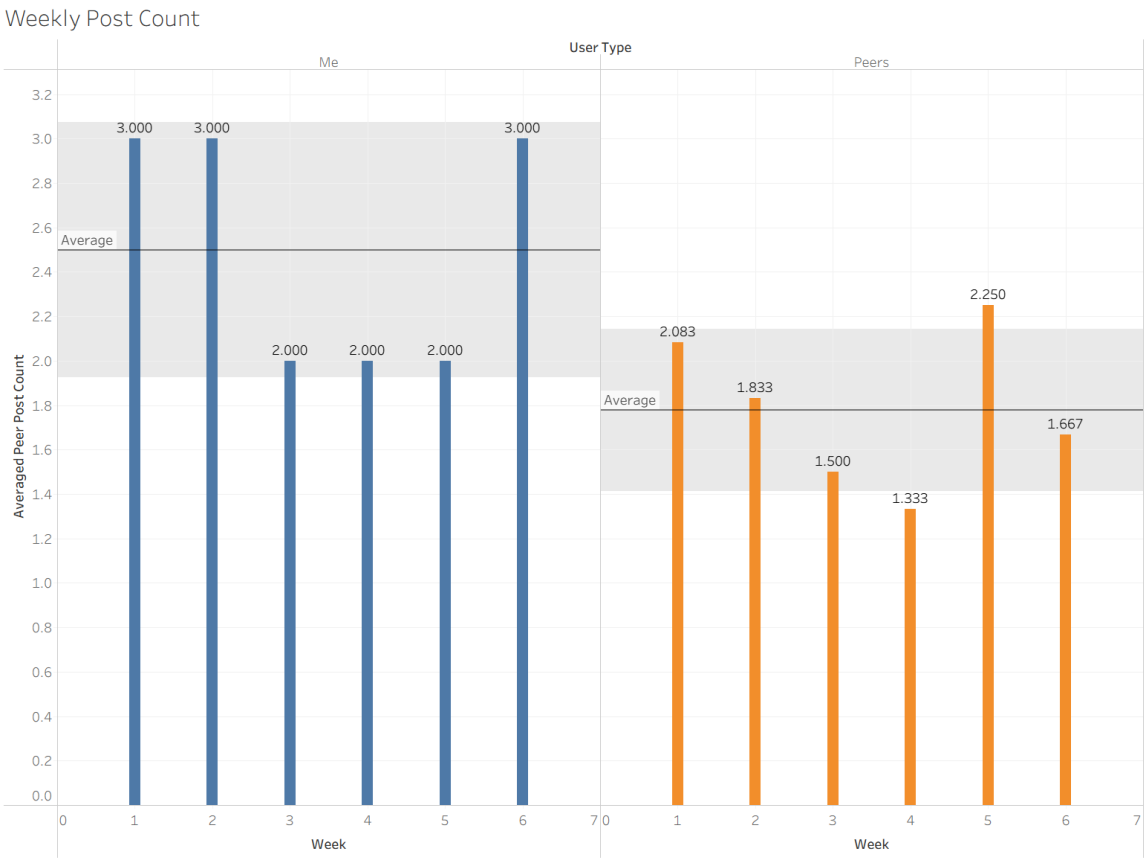


Figure 1. Weekly posts: me vs. peer average (with 95% CIs).

RQ2: Factors Associated with Replies to Original Posts

RQ2a: Timing (Day in Study Week)

There was a moderate negative correlation between day-in-study-week and number of replies (r = –0.35, p = 0.008), indicating earlier posts received more replies. The pattern was visually apparent in a downward-sloping trend line.

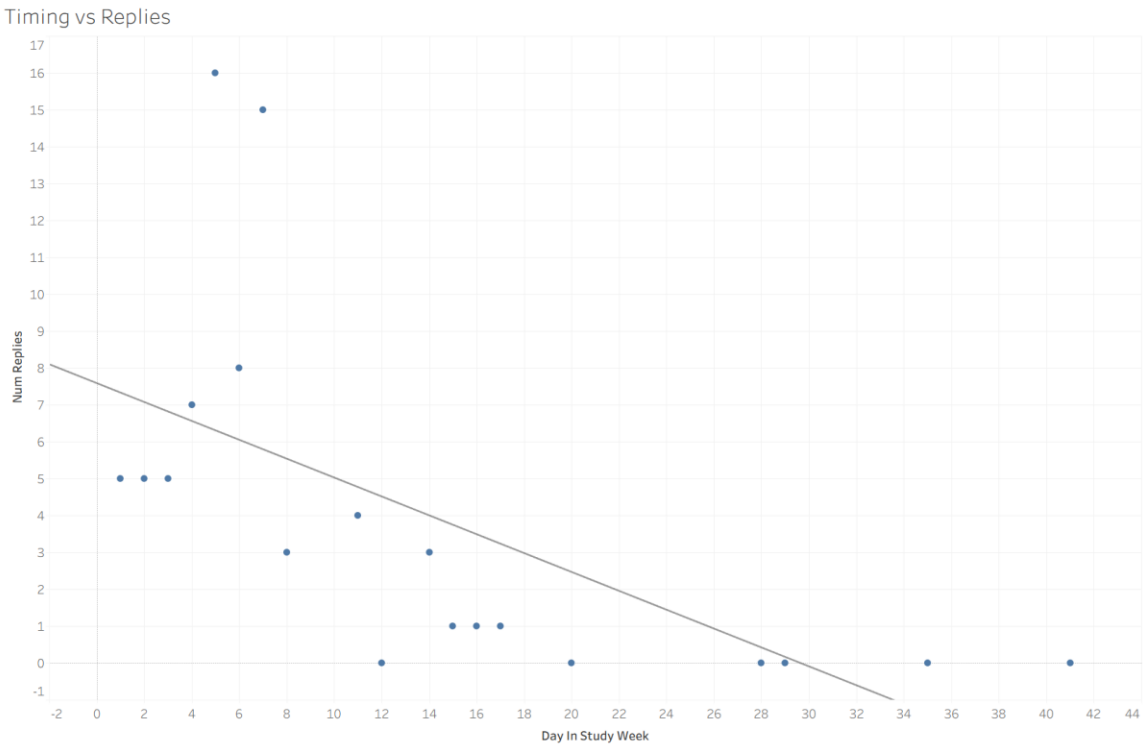


Figure 2. Scatter of day-in-study-week vs. replies (with trend line).

RQ2b: Sentiment

No meaningful relationship was found between sentiment and number of replies (r = –0.028, p = 0.838). Most original posts had high positive sentiment (> 0.9), limiting variability.

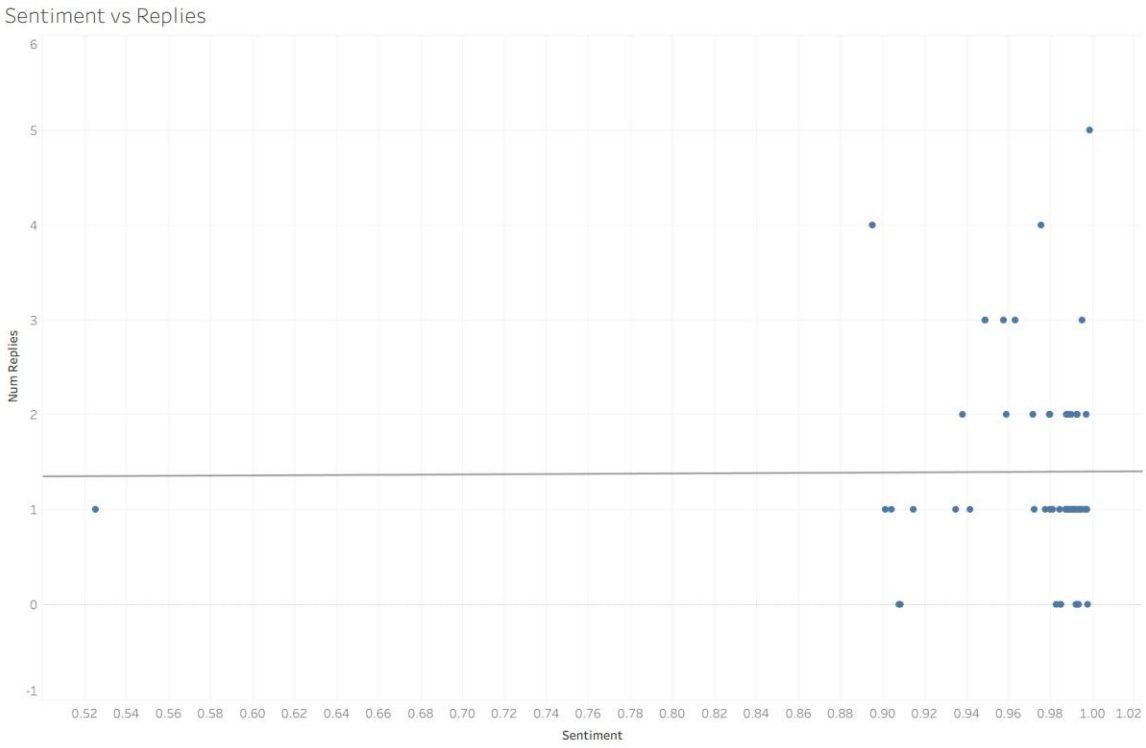


Figure 3. Scatter of sentiment vs. replies (with trend line).

RQ2c: Word Count

No relationship was observed between word count and number of replies ($r = -0.018$, $p = 0.895$). Data points were widely scattered without a discernible trend.

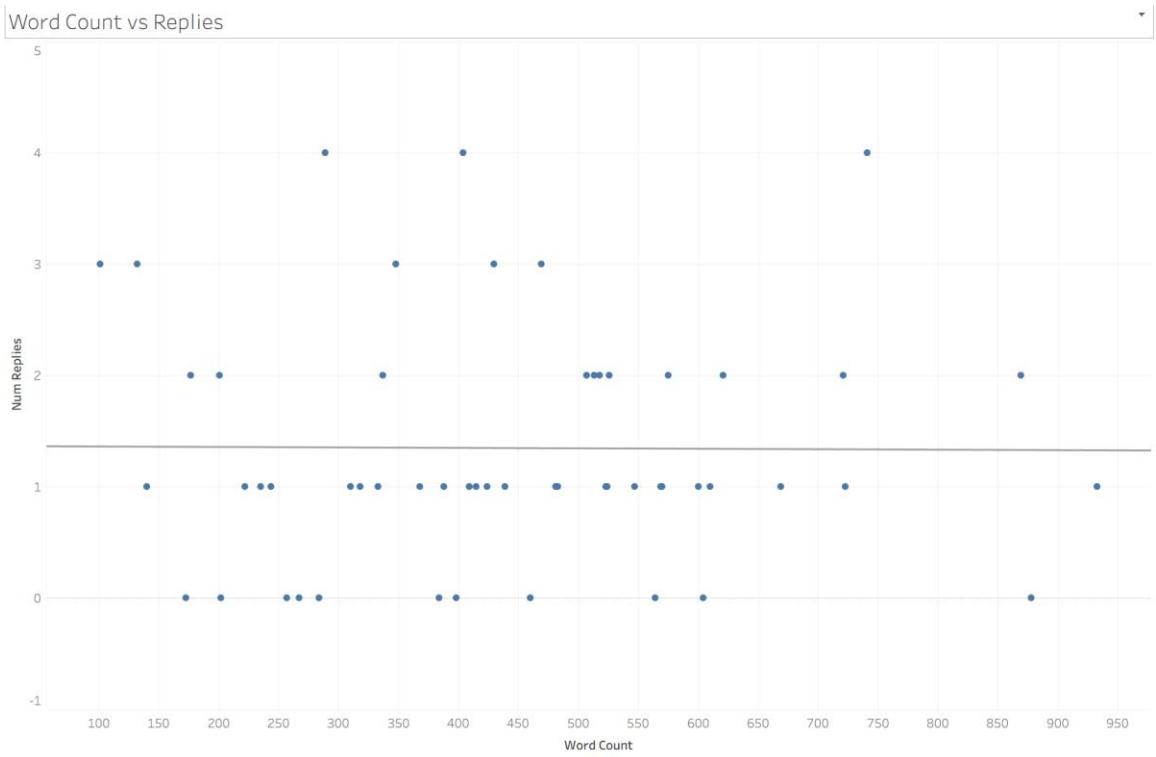


Figure 4. Scatter of word count vs. replies (with trend line).

RQ3: Replies Received—Me vs. Peers

Total replies received (sum across my original posts): 9. Class median: 7. Class average: 6.17. The 95% CI for mean replies across authors was [4.36, 7.97]. My result exceeded both the median and the upper CI bound, suggesting an above-average engagement outcome. A likely contributor is volume: I authored more original posts than average, providing more opportunities to attract replies.

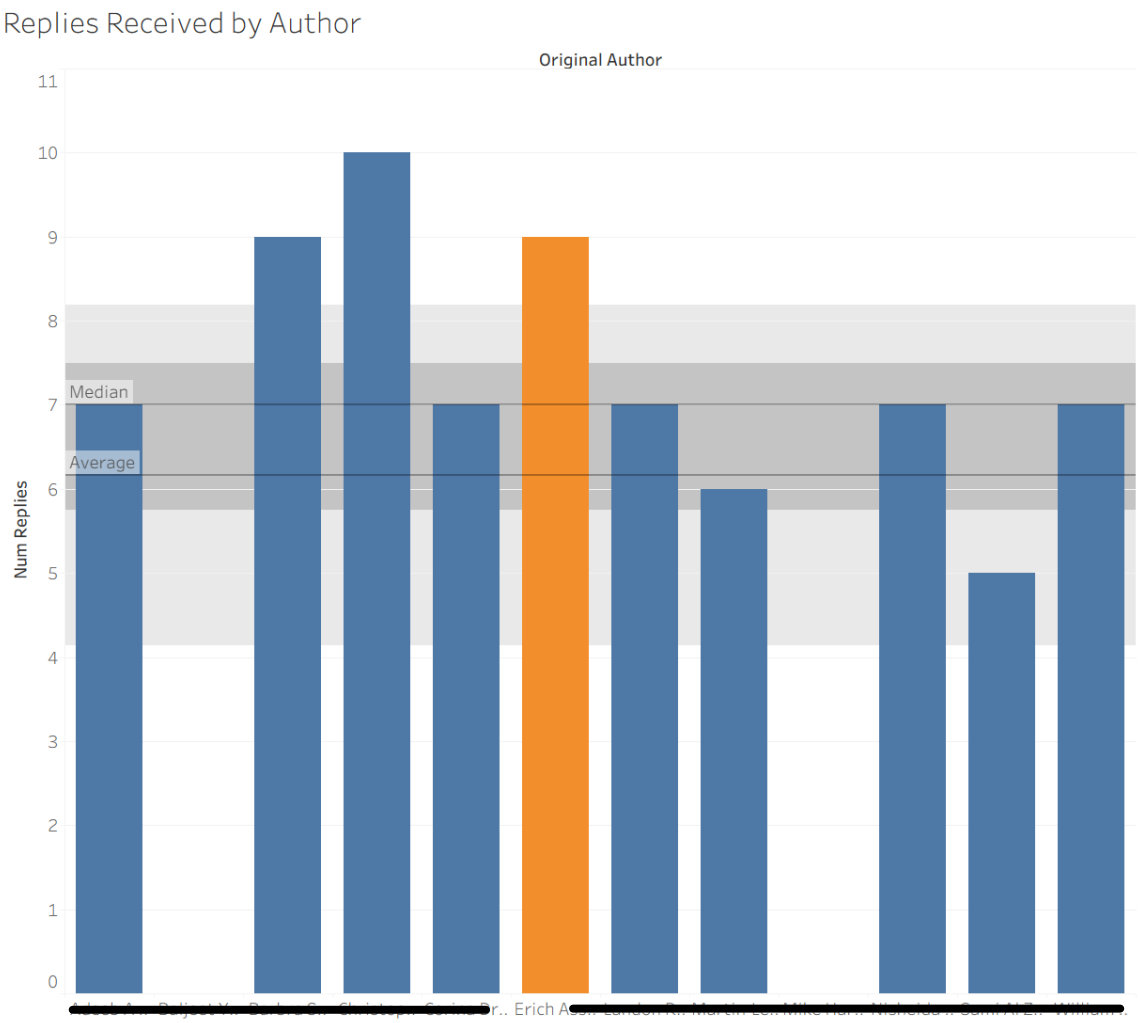
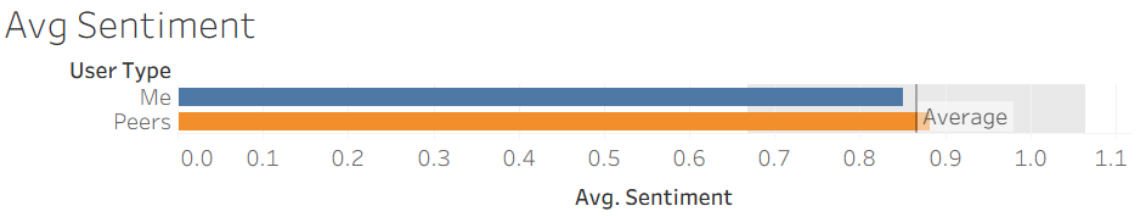


Figure 5. Total replies per author (95% CI and median; my bar highlighted).

RQ4: Sentiment—Me vs. Peers

Mean sentiment (me): 0.850 (95% CI [0.748, 0.952], n = 15). Mean sentiment (peers): 0.881 (95% CI [0.837, 0.926], n = 128). The difference is not statistically significant; both are firmly in the strongly positive range. A chi-square test on sentiment categories (strongly negative, negative, neutral, positive, strongly positive) was not significant ($\chi^2 = 3.21$, $p = 0.523$). My distribution included slightly more 'positive' (vs. 'strongly positive') posts, but the overall tone did not differ meaningfully.



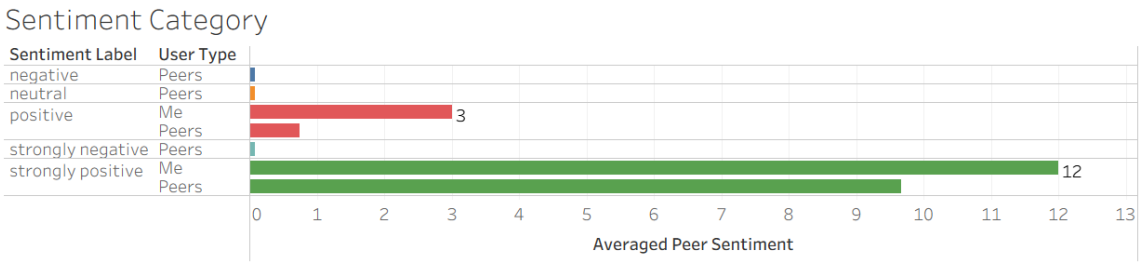


Figure 6. Sentiment distribution comparison and CIs by group.

Discussion

The dominant factor associated with replies was timing: earlier posts garnered more responses, likely due to increased visibility and the temporal dynamics of asynchronous forums. Contrary to intuition, neither post length nor sentiment predicted replies; verbosity and cheerful tone did not translate into more engagement in this dataset. My higher total replies likely reflect consistent, timely participation and higher original-post volume, which created more reply opportunities.

Limitations include sample size, a compressed sentiment range (ceiling effect), and reliance on automated sentiment (VADER), which can misinterpret academic tone. The analysis focuses on Brightspace, as the Landing platform contained only personal introductions rather than weekly course discussions. Consequently, integrating Landing data would not have added substantive analytical value or improved coverage. Future work could model thread-level context, topic effects, and network position (e.g., social network analysis) to explain reply dynamics beyond timing.

Conclusions

This participation analysis shows that steady, early engagement is the best predictor of replies in our course forums. My participation frequency was consistently at or above peer averages, and my original posts attracted more replies than typical in the cohort. Length and sentiment of posts did not matter for reply counts, and overall tone was uniformly positive for all participants. The pipeline (PDF extraction → structured dataset → visualization/statistics) is reusable and can be extended to include Landing data and additional analytics (e.g., topic modeling, network analysis) for deeper insight.

References

Hutto, C. J., & Gilbert, E. (2014). VADER: A parsimonious rule-based model for sentiment analysis of social media text. Proceedings of the 8th International AAAI Conference on Weblogs and Social Media (ICWSM), 216–225.

Johnson, J. (n.d.). pdfplumber (Version x.y.z) [Computer software]. Retrieved from https://github.com/jsvine/pdfplumber

Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

The pandas development team. (2020). pandas-dev/pandas: Pandas (Version x.y.z) [Computer software]. https://doi.org/10.5281/zenodo.3509134

Tableau Software. (n.d.). Tableau Desktop [Computer software]. https://www.tableau.com

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining.

Appendix A

Python Script 1 – Data Extraction and Sentiment Analysis

```
import re

import os

import glob

from datetime import datetime, date, timedelta


import pdfplumber

import pandas as pd


# --------------------------------

# Sentiment analysis setup (VADER)

# --------------------------------

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer


# Download once (quiet) and init analyzer

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

```
sia = SentimentIntensityAnalyzer()


def simple_sentiment(text: str) -> float:
    """Compute sentiment using VADER (compound score: -1..+1)."""
    if not text or not text.strip():
        return 0.0
    return sia.polarity_scores(text)['compound']


# ---------------------------------
# COURSE CALENDAR ANCHOR
# ---------------------------------
COURSE_WEEK1_MONDAY = date(2025, 9, 8)


def study_week_start(week_num):
    """Return Monday date for the given study week number (1-based)."""
    if week_num is None or COURSE_WEEK1_MONDAY is None:
        return None
    return COURSE_WEEK1_MONDAY + timedelta(days=7 * (int(week_num) - 1))


# ---------------------------------
# Header extraction regex (robust)
# ---------------------------------
# More flexible name pattern that handles initials and various name formats
# Matches: "John Doe", "Adeeb A Haque", "Mary-Jane O'Brien", etc.
NAME_PATTERN = r"[A-Z][A-Za-z\u00C0-\u024F\u2019'\-]*(?:\s+[A-Z][A-Za-z\u00C0-\u024F\u2019'\-]*){0,4}"
```

```
# Primary pattern: Expects the format to be strictly:

# [Thread Title Line - MUST start with capital letter]

# [Author Name] - [Date]

# The title should start with a capital letter and be a reasonable discussion title

HEADER_REGEX = re.compile(

   r"^(?P<title>[A-Z][^\n]{4,100}?)\s*\n"

   r"(?P<author>" + NAME_PATTERN + r")\s*[-–—]\s*"

   r"(?P<datestr>(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|
March|April|May|June|July|August|September|October|November|December)\s+\d{1,2},\s
+\d{4}\s+\d{1,2}:\d{2}\s*[AP]M)",

   re.MULTILINE

)


# Alternative: Just look for Author - Date (no title line before)

# This is the most reliable pattern since we know: Author Name - Date is always the header

ALT_HEADER_REGEX = re.compile(

   r"^(?P<author>" + NAME_PATTERN + r")\s*[-–—]\s*"

   r"(?P<datestr>(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|
March|April|May|June|July|August|September|October|November|December)\s+\d{1,2},\s
+\d{4}\s+\d{1,2}:\d{2}\s*[AP]M)",

   re.MULTILINE

)


# If a header-like line sneaks into a body, this helps split it off.

# Only match if it's at start of line to avoid matching names mentioned in text

HEADER_LIKE_IN_BODY = re.compile(
```

```
    r"^(?:(?:Unit|Week)\s*\d+\s*Discussion\s*[-–—]?\s*)?" + NAME_PATTERN + r"\s*[-–—]\s*(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|May|June|July|August|September|October|November|December)\s+\d{1,2},\s+\d{4}\s+\d{1,2}:\d{2}\s*[AP]M",
    re.MULTILINE
)

# ---------------------------------
# Extraction & parsing
# ---------------------------------
def extract_text(path: str) -> str:
    """Extract and pre-clean text from PDF pages."""
    chunks = []
    with pdfplumber.open(path) as pdf:
        for page in pdf.pages:
            t = page.extract_text() or ""
            chunks.append(t)
    text = "\n".join(chunks)

    # --- aggressive normalization ---
    text = text.replace("\r", "\n")
    text = re.sub(r"[ \t]+", " ", text)
    text = re.sub(r"\n{3,}", "\n\n", text)
    text = text.replace("\u00A0", " ")
    text = text.replace("\u2013", "–").replace("\u2014", "—")

    # Key fixes for header separation
```

```
    # Ensure 'unread!' is always followed by newlines for better separation
    text = re.sub(r"(?i)\bunread!\s*", "unread!\n\n", text)
    # Ensure Week/Unit labels are on new lines
    text = re.sub(r"(?i)(?<!\n)(Week\s*\d+\s*(?:Discussion|:))", r"\n\n\1", text)
    text = re.sub(r"(?i)(?<!\n)(Unit\s*\d+\s*Discussion)", r"\n\n\1", text)
    # Clean up excessive newlines
    text = re.sub(r"\n{3,}", "\n\n", text)

    return text


def recover_title(lines, line_offsets, header_start):
    """Helper to find a likely thread title by looking at lines before the header."""
    title = ""
    line_idx = 0
    for idx, (off, ln) in enumerate(line_offsets):
        if off <= header_start < off + len(ln) + 1:
            line_idx = idx
            break

    # Look back up to 6 lines for a plausible title
    for j in range(max(0, line_idx - 6), line_idx):
        cand = lines[j].strip()
        # A good title candidate is:
        # - Starts with a capital letter (thread titles always do)
        # - Reasonably long (not just one or two words)
        # - Contains "Discussion", "Week", "Unit", or ends with "?"
```

```
        # - Is not just a name-date pattern (author line)
        if (len(cand) > 10 and
            cand[0].isupper() and
            not re.match(NAME_PATTERN + r'\s*[-––—
]\s*(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)', cand) and
            ("Discussion" in cand or "Week" in cand or "Unit" in cand or cand.endswith("?"))
        ):
            title = cand
            break

    if not title:
        # Look for the line immediately before the header as last resort
        if line_idx > 0:
            cand = lines[line_idx - 1].strip()
            # Must start with capital and not be a date or author line
            if (len(cand) > 5 and
                cand[0].isupper() and
                not re.match(r'^(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)', cand) and
                not re.match(NAME_PATTERN + r'\s*[-––—]', cand)
            ):
                title = cand

    return title if title else "Unknown Thread"


def parse_posts(text: str):
    """Parse discussion posts from extracted text."""
```

```
posts = []
# Ensure 'unread!' is on its own line and followed by a newline
text = re.sub(r"(?i)\bunread!\s*", "unread!\n", text)
text = re.sub(r"\n{3,}", "\n\n", text)  # normalize multiple newlines

matches = list(HEADER_REGEX.finditer(text))
mode = "FULL" if matches else "ALT"
if mode == "ALT":
    matches = list(ALT_HEADER_REGEX.finditer(text))

if not matches:
    print("WARNING: No post headers found in the text!")
    return posts

# Build line offsets to recover titles in ALT mode
lines = text.splitlines()
line_offsets, offset = [], 0
for ln in lines:
    line_offsets.append((offset, ln))
    offset += len(ln) + 1  # +1 for newline

for i, m in enumerate(matches):
    start = m.end()
    end = matches[i + 1].start() if i + 1 < len(matches) else len(text)
    body = text[start:end].strip()
```

```
    # Remove all 'unread!' markers from body
    body = re.sub(r"(?i)\bunread!\s*", "", body).strip()

    # CRITICAL: Look for embedded headers and truncate before them
    # Search for any pattern that looks like: Name - Date
    embedded_header = HEADER_LIKE_IN_BODY.search(body)
    if embedded_header:
        # Take only the content BEFORE the embedded header
        body = body[:embedded_header.start()].strip()
        print(f"DEBUG: Found embedded header at position {embedded_header.start()}, truncating post")

    # Additional safety: if we see "Week N Discussion -" pattern in body, truncate there
    week_pattern = re.search(r'\n\s*Week\s*\d+\s*Discussion\s*[-–—]', body, re.IGNORECASE)
    if week_pattern:
        body = body[:week_pattern.start()].strip()
        print(f"DEBUG: Found 'Week N Discussion' pattern, truncating post")

    if mode == "FULL":
        title = m.group("title").strip()
        # Clean up title - remove common issues
        title = re.sub(r'\n', ' ', title)  # Remove embedded newlines
        title = re.sub(r'\s+', ' ', title)  # Normalize whitespace
        # If title looks wrong (contains author name or is just fragments), look for better one
        if len(title) < 5 or len(title) > 150:
            title = recover_title(lines, line_offsets, m.start())
```

```
        else:
            # Reconstruct a title by scanning up for a likely thread name
            title = recover_title(lines, line_offsets, m.start())

        author = m.group("author").strip()
        # Clean up author - remove embedded newlines and extra whitespace
        author = re.sub(r'\n', ' ', author)
        author = re.sub(r'\s+', ' ', author)
        # Safety: if author line still contains 'Discussion', peel it off
        if "discussion" in author.lower():
            author = re.sub(r"(?i)\b(?:unit|week)\s*\d+\s*discussion\s*[-–—]?\s*", "",
author).strip()
        # Remove leading/trailing dashes or other artifacts
        author = re.sub(r'^[-–—\s]+|[-–—\s]+$', '', author).strip()

        datestr = m.group("datestr").strip()

        # Parse datetime (short or long month name)
        posted_at = None
        for fmt in ("%b %d, %Y %I:%M %p", "%B %d, %Y %I:%M %p"):
            try:
                posted_at = datetime.strptime(datestr, fmt)
                break
            except Exception:
                continue
```

```
        posts.append({
            "thread_title": title,
            "author": author,
            "posted_at": posted_at,
            "post_text": body
        })

    return posts


def classify(posts):
    """Label posts as 'original' or 'reply' within each thread."""
    first_by_thread = {}
    for p in posts:
        t = p["thread_title"]
        if t not in first_by_thread:
            p["post_type"] = "original"
            first_by_thread[t] = True
        else:
            p["post_type"] = "reply"
    return posts


def to_rows(week_num, posts):
    """Convert parsed posts into rows suitable for CSV/Excel export."""
    start = study_week_start(week_num)  # Monday of that study week
    rows = []
    for p in posts:
```

```
wc = len(re.findall(r"\w+", p["post_text"] or ""))
sent = simple_sentiment(p["post_text"] or "")

# Granular sentiment classification (your bins)
if sent <= -0.6:
    sent_label = "strongly negative"
elif sent > -0.6 and sent <= -0.15:
    sent_label = "negative"
elif sent > -0.15 and sent < 0.15:
    sent_label = "neutral"
elif sent >= 0.15 and sent <= 0.6:
    sent_label = "positive"
else:  # sent > 0.6
    sent_label = "strongly positive"

# Day-in-study-week: Monday=1 … Sunday=7; late posts become 8, 9, ...
if p["posted_at"] and start:
    delta_days = (p["posted_at"].date() - start).days
    day_in = delta_days + 1  # Monday=1
else:
    # Fallback (rare): just weekday Mon=1..Sun=7
    day_in = p["posted_at"].isoweekday() if p["posted_at"] else None

rows.append({
    "week": week_num,
    "thread_title": p["thread_title"],
```

```
            "author": p["author"],
            "posted_at": p["posted_at"].date().isoformat() if p["posted_at"] else None,
            "day_in_study_week": day_in,          # may be >7 for late posts
            "post_type": p["post_type"],
            "word_count": wc,
            "sentiment": round(sent, 4),
            "sentiment_label": sent_label,
            "post_text": p["post_text"]
        })
    return rows


def process(input_dir="InputData", out_dir="OutputData", pattern="W*.pdf"):
    """Process all PDFs matching pattern and output per-week + combined CSV/XLSX."""
    os.makedirs(out_dir, exist_ok=True)
    full_pattern = os.path.join(input_dir, pattern)

    all_rows = []
    for pdf_path in sorted(glob.glob(full_pattern)):
        base = os.path.basename(pdf_path)
        m = re.search(r"W(\d+)\.pdf$", base, re.IGNORECASE)
        week_num = int(m.group(1)) if m else None

        text = extract_text(pdf_path)
        posts = parse_posts(text)
        posts = classify(posts)
        rows = to_rows(week_num, posts)
```

```
        # Per-week CSV (Excel-friendly UTF-8 with BOM)
        df = pd.DataFrame(rows)
        out_csv = os.path.join(out_dir, f"Week{week_num or 'X'}_discussion.csv")
        df.to_csv(out_csv, index=False, encoding="utf-8-sig")

        # Per-week Excel file (requires openpyxl installed)
        out_excel = os.path.join(out_dir, f"Week{week_num or 'X'}_discussion.xlsx")
        df.to_excel(out_excel, index=False, engine='openpyxl')

        all_rows.extend(rows)

    if all_rows:
        combined = pd.DataFrame(all_rows)
        combined.to_csv(os.path.join(out_dir, "discussion_allweeks.csv"),
                        index=False, encoding="utf-8-sig")
        combined.to_excel(os.path.join(out_dir, "discussion_allweeks.xlsx"),
                          index=False, engine='openpyxl')
    return True

if __name__ == "__main__":
    process()
```

Appendix B

Python Script 2 – Reply Counting and Aggregation

```
import pandas as pd
import os

# Ensure output directory exists
os.makedirs("OutputData", exist_ok=True)

# Load the dataset
file_path = "InputData/discussion_allweeks.csv"
df = pd.read_csv(file_path)

# Prepare output list
summary_df_with_wc = []

# Track current original post
current_original = None

# Iterate through the rows in order
for idx, row in df.iterrows():
    if row['post_type'] == 'original':
        # Track the original post with word count, sentiment, and sentiment label included
        current_original = {
            'original_author': row['author'],
            'posted_at': row['posted_at'],
            'day_in_study_week': row['day_in_study_week'],
            'word_count': row['word_count'],
            'sentiment': row['sentiment'],
```

```
            'sentiment_label': row['sentiment_label'],
            'num_replies': 0
        }
        summary_df_with_wc.append(current_original)
    elif row['post_type'] == 'reply' and current_original:
        # Count reply only if it's from a different author
        if row['author'] != current_original['original_author']:
            current_original['num_replies'] += 1

# Convert to DataFrame
summary_df_wc = pd.DataFrame(summary_df_with_wc)

# Save to CSV in OutputData folder
summary_df_wc.to_csv("OutputData/original_posts_reply_counts.csv", index=False)
```