Introduction to Data Science

BSCS *[6-A]*

Assignment # 04

Submitted By:

Natasha Fatima
Enroll: 03-134231-055

Submitted to: Dr. Khawaja Qasim

Case Study – Classifying Reddit Posts with Python & Visualizing in Power BI

DEPARTMENT OF COMPUTER SCIENCES
BAHRIA UNIVERSITY, LAHORE CAMPUS

Assignment Overview
Deliverables and Instructions

In this case study, you will build a **real-world data science pipeline** that performs the following:

1. Connect to **Reddit API** using Python
2. Collect Reddit posts from specific subreddits
3. Classify posts into categories (e.g., sentiment, topic) using **NLP**
4. Store processed data in a structured format (CSV/Excel)
5. Load into **Power BI** for dashboard creation and insights

You will complete an **end-to-end solution**: from API to ML classification to BI dashboarding.

Assignment Breakdown

Part 1: Reddit API Integration (4 Marks)

1. Use **PRAW (Python Reddit API Wrapper)** or `requests` with Reddit's API.
2. Fetch data from at least **2 different subreddits**, e.g.:
    - `r/technology`, `r/mentalhealth`, `r/AskReddit`, etc.
3. Collect:
    - Title, body, author, number of comments, upvotes, timestamp

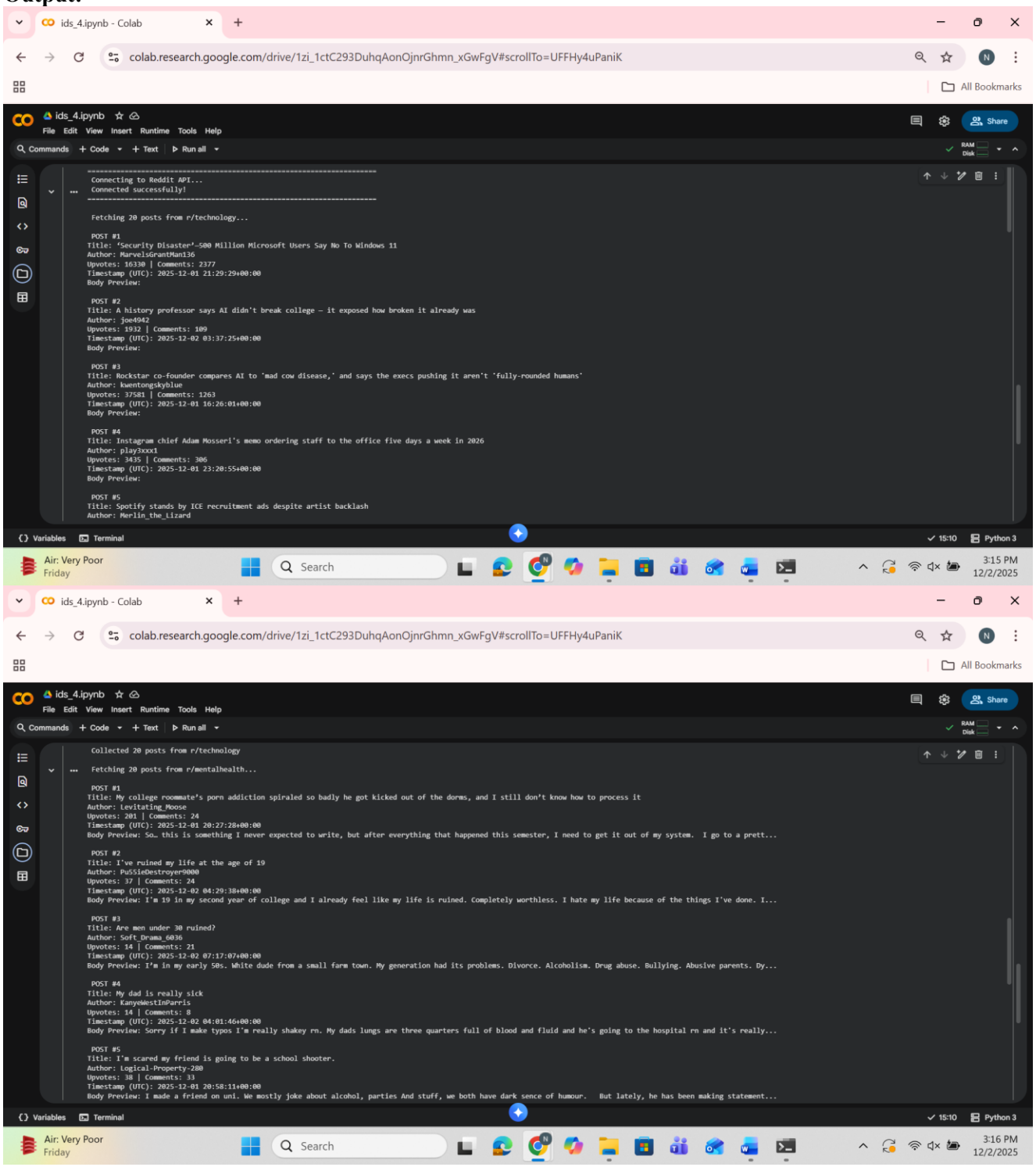
Code:

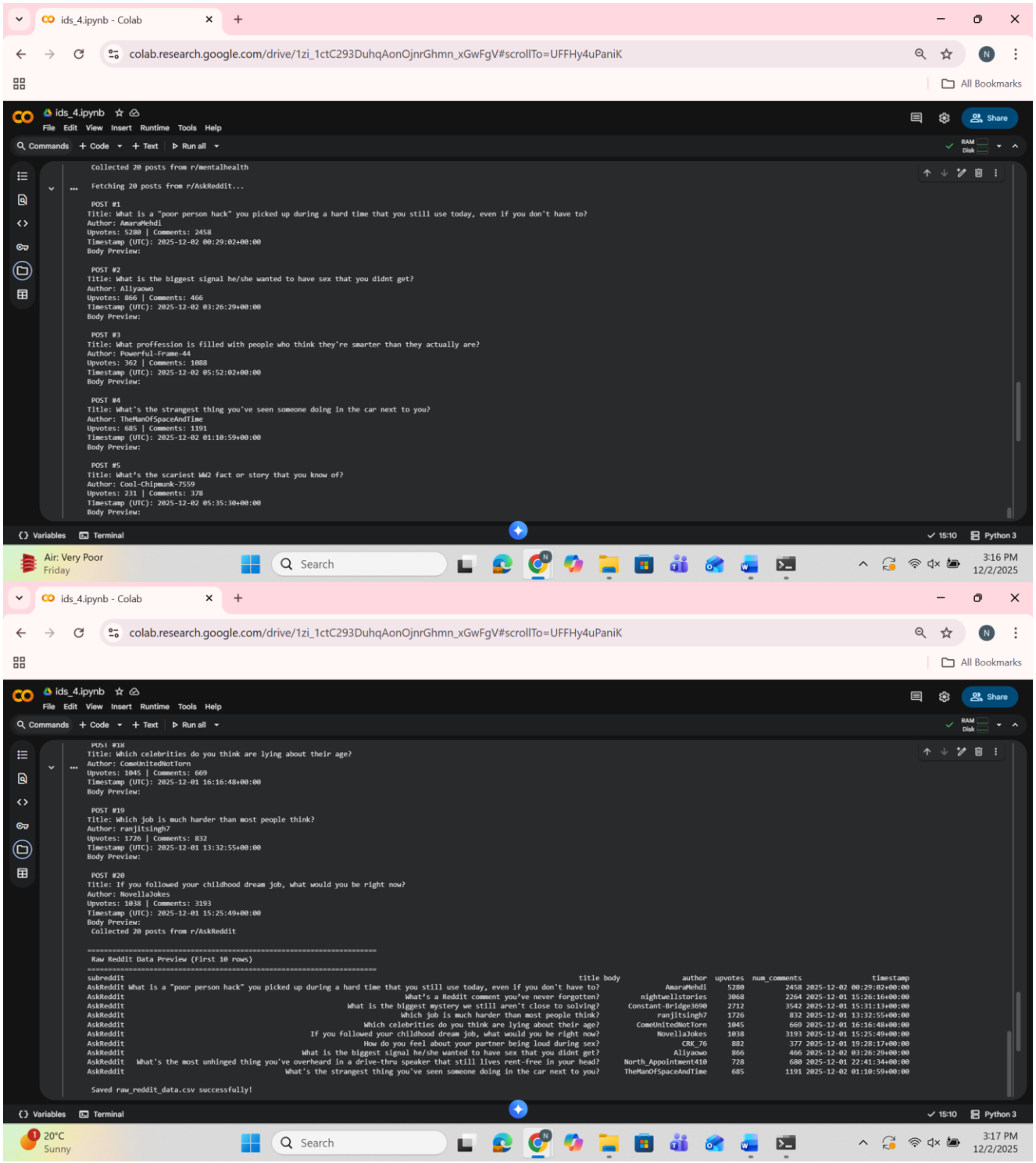
```
import logging
logging.getLogger("praw").setLevel(logging.ERROR)
import praw
from dotenv import load_dotenv
import os
import pandas as pd
from datetime import datetime, timezone
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Load environment variables
load_dotenv()
CLIENT_ID = os.getenv("REDDIT_CLIENT_ID")
CLIENT_SECRET = os.getenv("REDDIT_CLIENT_SECRET")
USER_AGENT = os.getenv("REDDIT_USER_AGENT")

# Basic check for credentials
if not CLIENT_ID or not CLIENT_SECRET or not USER_AGENT:
    raise ValueError("Reddit API credentials are missing in the environment variables.")

print("="*70)
print(" Connecting to Reddit API...")
reddit = praw.Reddit(client_id=CLIENT_ID,
                     client_secret=CLIENT_SECRET,
                     user_agent=USER_AGENT)
reddit.read_only = True
print(" Connected successfully!")
print("="*70)
```

```
# Subreddits & posts config
subreddits = ["technology", "mentalhealth", "AskReddit"]
posts_per_subreddit = 20
all_posts = []

# Fetch posts
for sub in subreddits:
    print(f"\n Fetching {posts_per_subreddit} posts from r/{sub}...")
    subreddit_posts = []

    for post in reddit.subreddit(sub).hot(limit=posts_per_subreddit + 5):  # extra in case of stickied
        if post.stickied:
            continue

        author_name = str(post.author) if post.author else "[deleted]"
        body_text = post.selftext if post.selftext else ""

        # Collect post
        post_data = {
            "subreddit": sub,
            "title": post.title,
            "body": body_text,
            "author": author_name,
            "upvotes": post.score,
            "num_comments": post.num_comments,
            "timestamp": datetime.fromtimestamp(post.created_utc, tz=timezone.utc)
        }
        subreddit_posts.append(post_data)

        # Print all posts (or first 20)
        if len(subreddit_posts) <= posts_per_subreddit:
            print(f"\n POST #{len(subreddit_posts)}")
            print(f"Title: {post.title}")
            print(f"Author: {author_name}")
            print(f"Upvotes: {post.score} | Comments: {post.num_comments}")
            print(f"Timestamp (UTC): {post_data['timestamp']}")
            print(f"Body Preview: {body_text[:150].replace(chr(10),' ')} {'...' if len(body_text) > 150 else
"}")

        # Stop when we have enough posts
        if len(subreddit_posts) >= posts_per_subreddit:
            break

    all_posts.extend(subreddit_posts)
    print(f" Collected {len(subreddit_posts)} posts from r/{sub}")

# Convert to DataFrame
df = pd.DataFrame(all_posts, columns=[
    "subreddit","title","body","author","upvotes","num_comments","timestamp"
])

# sort by subreddit then upvotes descending for better preview
df.sort_values(by=["subreddit","upvotes"], ascending=[True, False], inplace=True)

print("\n" + "="*70)
```

```
        print(" Raw Reddit Data Preview (First 10 rows) ")
        print("="*70)
        print(df.head(10).to_string(index=False))

        # Save CSV
        csv_filename = "raw_reddit_data.csv"
        df.to_csv(csv_filename, index=False, encoding="utf-8")
print(f"\n Saved {csv_filename} successfully!")
```

Output:





Fetching Reddit Data via API

We used the **PRAW** library to fetch 20 posts from each selected subreddit (r/technology, r/mentalhealth, r/AskReddit), collecting the title, body, author, upvotes, comments, and timestamp. The data is stored in a **Pandas DataFrame** and exported to raw_reddit_data.csv.

Deliverables:

- Python code for API access
- Print/log at least 20 posts per subreddit
- Export raw data to CSV (`raw_reddit_data.csv`)

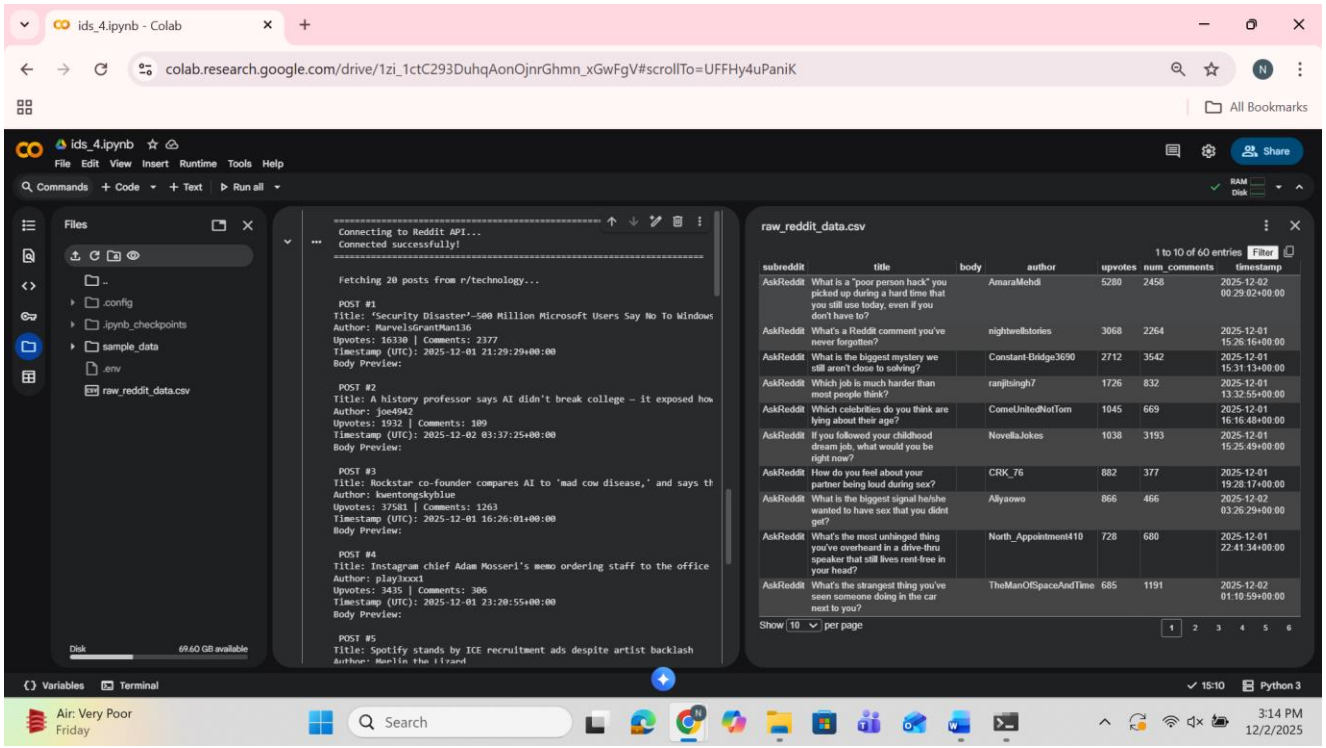


Figure 1: Raw Reddit data preview

Part 2: Text Cleaning & Preprocessing (3 Marks)

1. Apply basic preprocessing:
    - Remove stopwords, punctuation, URLs, etc.
    - Lowercasing, tokenization
2. Use libraries: `nltk`, `spaCy`, or `re`

Code:

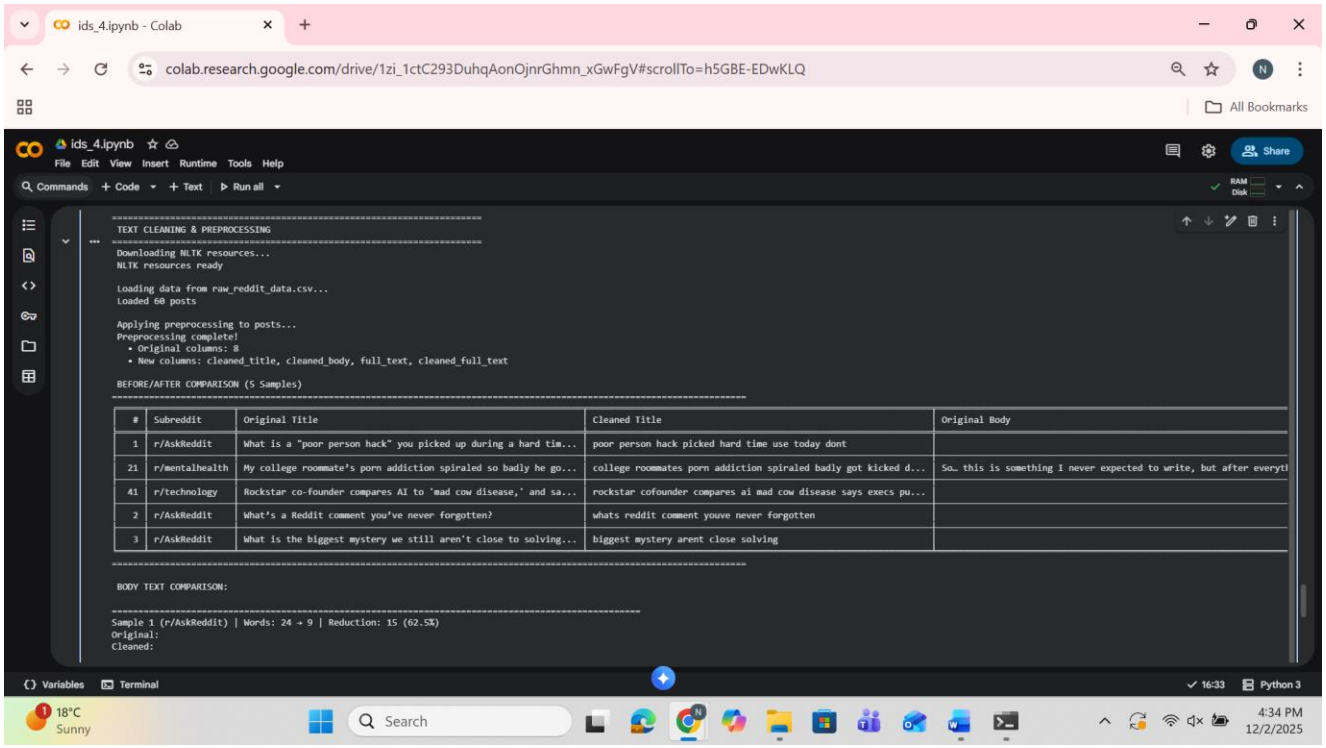
```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import pandas as pd
import warnings
from tabulate import tabulate

warnings.filterwarnings('ignore')

print("="*70)
print(" TEXT CLEANING & PREPROCESSING")
print("="*70)

print(" Downloading NLTK resources...")
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
print(" NLTK resources ready")

print("\n Loading data from raw_reddit_data.csv...")
df = pd.read_csv('raw_reddit_data.csv')
```

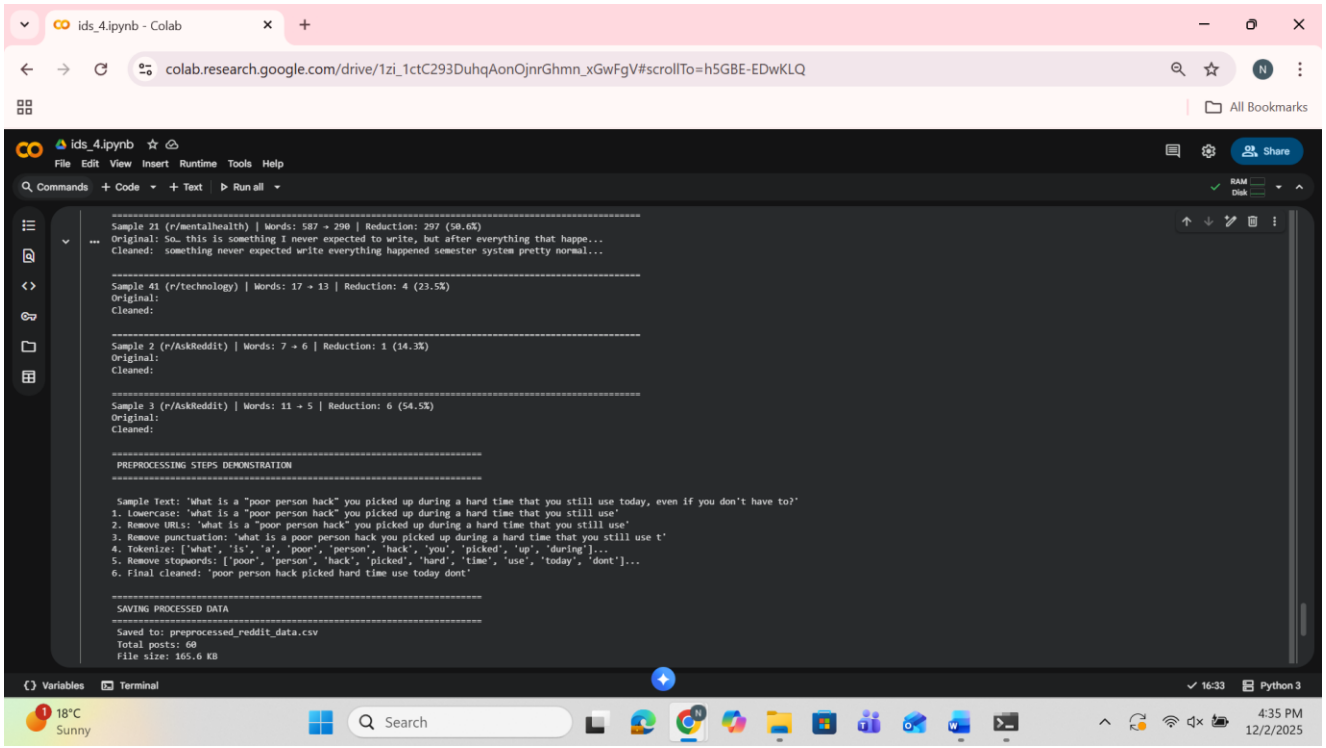
```
print(f" Loaded {len(df)} posts")

# Define stopwords
stop_words = set(stopwords.words('english'))
reddit_stopwords = [
    'like', 'get', 'would', 'could', 'also', 'really', 'one', 'much', 'many',
    'even', 'still', 'maybe', 'think', 'know', 'said', 'people', 'thing', 'things',
    'go', 'see', 'want', 'need', 'make', 'take', 'good'
]
stop_words.update(reddit_stopwords)

def preprocess_text(text, remove_stopwords=True, remove_numbers=True):
    if not isinstance(text, str) or pd.isna(text):
        return ""

    text = text.lower().strip()  # lowercase and trim
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)  # remove URLs
    text = re.sub(r'@\w+|#\w+', '', text)  # remove mentions & hashtags
    text = re.sub(r'[^a-zA-Z\s]' if remove_numbers else r'[^\w\s]', '', text)  # remove punctuation/numbers
    text = re.sub(r'\s+', ' ', text).strip()  # remove extra spaces
    tokens = word_tokenize(text)

    if remove_stopwords:
        tokens = [w for w in tokens if w not in stop_words]

    return ' '.join(tokens)

print("\n Applying preprocessing to posts...")
df['cleaned_title'] = df['title'].apply(preprocess_text)
df['cleaned_body'] = df['body'].apply(lambda x: preprocess_text(x) if pd.notna(x) else "")
df['full_text'] = df['title'] + ' ' + df['body'].fillna('')
df['cleaned_full_text'] = df['full_text'].apply(preprocess_text)

print(" Preprocessing complete!")
print(f"   • Original columns: {len([c for c in df.columns if 'cleaned' not in c])}")
print(f"   • New columns: cleaned_title, cleaned_body, full_text, cleaned_full_text")

sample_indices = []
for sub in df['subreddit'].unique():
    sub_df = df[df['subreddit'] == sub]
    if len(sub_df) > 0:
        sample_indices.append(sub_df.index[0])

# Ensure exactly 5 samples
if len(sample_indices) < 5:
    remaining = df.index.difference(sample_indices)[:5 - len(sample_indices)]
    sample_indices.extend(remaining)
sample_indices = sample_indices[:5]

truncate = lambda text, n: text[:n] + ("..." if len(text) > n else "")

# Prepare comparison data
comparison_data = []
for idx in sample_indices:
    original_title = df.loc[idx, 'title']
    cleaned_title = df.loc[idx, 'cleaned_title']
```

```
    body_text = df.loc[idx, 'body']
    original_body = "" if pd.isna(body_text) else str(body_text)
    cleaned_body = df.loc[idx, 'cleaned_body']
    subreddit = df.loc[idx, 'subreddit']

    orig_words = len(original_title.split()) + len(original_body.split())
    clean_words = len(cleaned_title.split()) + len(cleaned_body.split())
    reduction = orig_words - clean_words
    reduction_pct = (reduction / orig_words * 100) if orig_words > 0 else 0

    comparison_data.append({
        '#': idx + 1,
        'Subreddit': f"r/{subreddit}",
        'Original Title': truncate(original_title, 60),
        'Cleaned Title': truncate(cleaned_title, 60),
        'Original Body': truncate(original_body, 80),
        'Cleaned Body': truncate(cleaned_body, 80),
        'Word Count': f"{orig_words} → {clean_words}",
        'Reduction': f"{reduction} ({reduction_pct:.1f}%)"
    })

comparison_df = pd.DataFrame(comparison_data)

print("\n BEFORE/AFTER COMPARISON (5 Samples)")
print("="*120)
print(tabulate(comparison_df, headers='keys', tablefmt='fancy_grid', showindex=False, stralign='left'))
print("="*120)

print("\n BODY TEXT COMPARISON:")
for _, row in comparison_df.iterrows():
    print(f"\n{'='*100}")
    print(f'Sample {row['#']} ({row['Subreddit']}) | Words: {row['Word Count']} | Reduction:
{row['Reduction']}")
    print(f"Original: {row['Original Body']}")
    print(f"Cleaned:  {row['Cleaned Body']}")

print("\n" + "="*70)
print(" PREPROCESSING STEPS DEMONSTRATION")
print("="*70)

sample_idx = sample_indices[0]
sample_text = df.loc[sample_idx, 'title']
print(f"\n Sample Text: '{sample_text}'")

step_lower = sample_text.lower()
step_no_url = re.sub(r'http\S+|www\S+|https\S+', '', step_lower)
step_no_punct = re.sub(r'[^a-zA-Z\s]', '', step_no_url)
tokens = word_tokenize(step_no_punct)
tokens_no_stop = [w for w in tokens if w not in stop_words]

steps = [
    ("1. Lowercase", step_lower),
    ("2. Remove URLs", step_no_url),
    ("3. Remove punctuation", step_no_punct),
    ("4. Tokenize", tokens),
    ("5. Remove stopwords", tokens_no_stop),
```

```
    ("6. Final cleaned", preprocess_text(sample_text))
]

for name, result in steps:
    if isinstance(result, list):
        print(f"{name}: {result[:10]}...")
    else:
        print(f"{name}: '{result[:80]}'")

print("\n" + "="*70)
print(" SAVING PROCESSED DATA")
print("="*70)

output_file = "preprocessed_reddit_data.csv"
df.to_csv(output_file, index=False, encoding='utf-8')
print(f" Saved to: {output_file}")
print(f" Total posts: {len(df)}")
print(f" File size: {df.memory_usage(deep=True).sum() / 1024:.1f} KB")
```

Output:





Text Cleaning & Preprocessing

Reddit posts were cleaned by lowercasing, removing URLs, punctuation, numbers, and stopwords, and then tokenized. A before/after comparison of 5 samples shows how word counts were reduced, making the text cleaner and ready for analysis. The processed data was saved to preprocessed_reddit_data.csv.

Deliverables:

- Python function for preprocessing
- Before/after comparison table for at least 5 samples

Comparison Table:

BEFORE/AFTER COMPARISON (5 Samples)

#	Subreddit	Original Title	Cleaned Title	Original Body
1	r/AskReddit	What is a "poor person hack" you picked up during a hard tim...	poor person hack picked hard time use today dont	
21	r/mentalhealth	My college roommate's porn addiction spiraled so badly he go...	college roommates porn addiction spiraled badly got kicked d...	So… this is something I never expected to write, but after everyth
41	r/technology	Rockstar co-founder compares AI to 'mad cow disease,' and sa...	rockstar cofounder compares ai mad cow disease says execs pu...	
2	r/AskReddit	What's a Reddit comment you've never forgotten?	whats reddit comment youve never forgotten	
3	r/AskReddit	What is the biggest mystery we still aren't close to solving...	biggest mystery arent close solving	

Original Body	Cleaned Body	Word Count	Reduction
		24 → 9	15 (62.5%)
So… this is something I never expected to write, but after everything that happe...	something never expected write everything happened semester system pretty normal...	587 → 290	297 (50.6%)
		17 → 13	4 (23.5%)
		7 → 6	1 (14.3%)
		11 → 5	6 (54.5%)

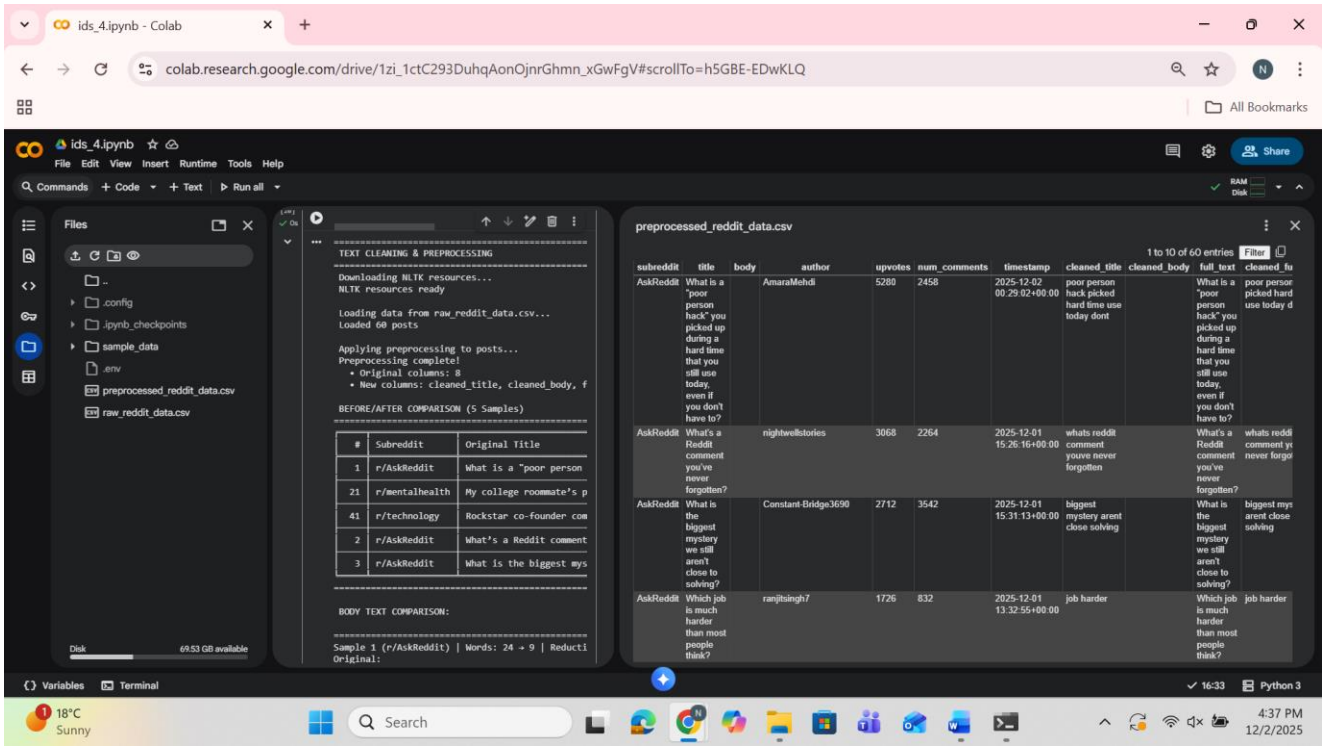


Figure 2: Preprocessed text samples

Part 3: Post Classification (5 Marks)

1. Choose **one classification task**:
   - **Sentiment analysis** (positive/neutral/negative) using `TextBlob`, `VADER`, or HuggingFace
   - OR **Topic classification** (e.g., tech, health, lifestyle) using fine-tuned model or keyword rules
2. Add a **classification column** to your dataset
3. Save the final output as `classified_reddit_data.csv`

Code :

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

from textblob import TextBlob
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon', quiet=True)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print("="*70)
print(" PART 3: POST CLASSIFICATION")
print("="*70)

# Load data
```

```
df = pd.read_csv('preprocessed_reddit_data.csv')
print(f" Loaded {len(df)} preprocessed posts")

# Initialize analyzer
vader = SentimentIntensityAnalyzer()

def analyze_sentiment_vader(text):
    """Analyze sentiment using VADER"""
    if not isinstance(text, str) or not text.strip():
        return {'sentiment': 'neutral', 'compound': 0}

    scores = vader.polarity_scores(text)
    compound = scores['compound']

    if compound >= 0.05:
        sentiment = 'positive'
    elif compound <= -0.05:
        sentiment = 'negative'
    else:
        sentiment = 'neutral'

    return {'sentiment': sentiment, 'compound': compound}

def analyze_sentiment_textblob(text):
    """Analyze sentiment using TextBlob"""
    if not isinstance(text, str) or not text.strip():
        return {'sentiment': 'neutral', 'polarity': 0}

    blob = TextBlob(text)
    polarity = blob.sentiment.polarity

    if polarity > 0:
        sentiment = 'positive'
    elif polarity < 0:
        sentiment = 'negative'
    else:
        sentiment = 'neutral'

    return {'sentiment': sentiment, 'polarity': polarity}

def get_final_sentiment(text):
    """Combine VADER and TextBlob results"""
    vader_result = analyze_sentiment_vader(text)
    textblob_result = analyze_sentiment_textblob(text)

    # If both agree, use that
    if vader_result['sentiment'] == textblob_result['sentiment']:
        return vader_result['sentiment']

    # If one is neutral, prefer the non-neutral
    if vader_result['sentiment'] == 'neutral':
```

```
        return textblob_result['sentiment']
    if textblob_result['sentiment'] == 'neutral':
        return vader_result['sentiment']

    # Otherwise use VADER (better for social media)
    return vader_result['sentiment']

print(" Performing sentiment analysis...")

# Get individual model results
df['vader_sentiment'] = df['cleaned_full_text'].apply(lambda x:
analyze_sentiment_vader(x)['sentiment'])
df['vader_compound'] = df['cleaned_full_text'].apply(lambda x:
analyze_sentiment_vader(x)['compound'])
df['textblob_sentiment'] = df['cleaned_full_text'].apply(lambda x:
analyze_sentiment_textblob(x)['sentiment'])
df['textblob_polarity'] = df['cleaned_full_text'].apply(lambda x:
analyze_sentiment_textblob(x)['polarity'])

# Get final combined sentiment
df['final_sentiment'] = df['cleaned_full_text'].apply(get_final_sentiment)

print(" Sentiment analysis complete!")

print(" Performing topic classification...")

topic_keywords = {
    'technology': ['tech', 'ai', 'software', 'computer', 'app', 'internet', 'digital',
                   'data', 'code', 'programming', 'machine', 'algorithm', 'cyber',
                   'security', 'cloud', 'crypto', 'phone', 'website', 'windows', 'android'],
    'mental_health': ['mental', 'therapy', 'anxiety', 'stress', 'depression', 'health',
                      'wellness', 'mindfulness', 'psychology', 'counseling', 'emotion',
                      'mind', 'happy', 'sad', 'angry', 'treatment', 'support', 'trauma',
                      'ptsd', 'disorder'],
    'education': ['school', 'college', 'university', 'study', 'learn', 'education',
                  'course', 'student', 'teacher', 'knowledge', 'skill', 'degree',
                  'exam', 'homework', 'class', 'professor', 'tuition', 'campus'],
    'entertainment': ['movie', 'music', 'game', 'tv', 'show', 'entertainment', 'fun',
                      'comedy', 'drama', 'film', 'song', 'artist', 'actor', 'celebrity',
                      'series', 'episode', 'season', 'netflix', 'youtube', 'spotify'],
    'politics': ['government', 'politic', 'election', 'vote', 'law', 'policy',
                 'democrat', 'republican', 'congress', 'senate', 'president',
                 'bill', 'senator', 'representative', 'supreme', 'court']
}

def classify_topic(text):
    """Classify text into topics based on keyword matching"""
    if not isinstance(text, str):
        return 'other'

    text_lower = text.lower()
```

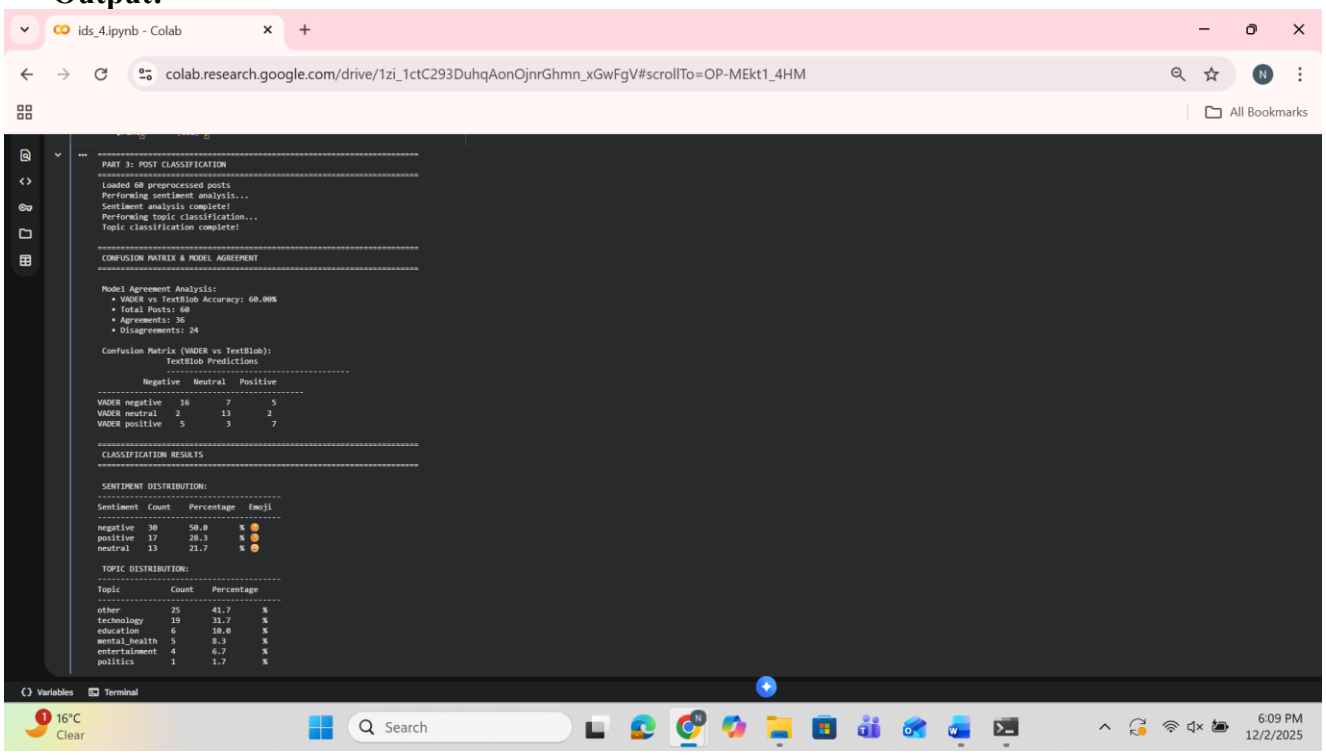
```
    scores = {}

    for topic, keywords in topic_keywords.items():
        score = sum(1 for keyword in keywords if keyword in text_lower)
        scores[topic] = score

    if max(scores.values()) > 0:
        return max(scores, key=scores.get)
    else:
        return 'other'

# Apply topic classification
df['primary_topic'] = df['cleaned_full_text'].apply(classify_topic)

print(" Topic classification complete!")

print("\n" + "="*70)
print(" CONFUSION MATRIX & MODEL AGREEMENT")
print("="*70)

# Calculate accuracy and confusion matrix
accuracy = accuracy_score(df['vader_sentiment'], df['textblob_sentiment'])
cm = confusion_matrix(df['vader_sentiment'], df['textblob_sentiment'],
                      labels=['negative', 'neutral', 'positive'])

print(f"\n Model Agreement Analysis:")
print(f"   • VADER vs TextBlob Accuracy: {accuracy:.2%}")
print(f"   • Total Posts: {len(df)}")
print(f"   • Agreements: {int(accuracy * len(df))}")
print(f"   • Disagreements: {int((1 - accuracy) * len(df))}")

print("\n Confusion Matrix (VADER vs TextBlob):")
print(" " * 15 + "TextBlob Predictions")
print(" " * 15 + "-" * 40)
print(" " * 10 + "Negative   Neutral   Positive")
print("-" * 45)

labels = ['negative', 'neutral', 'positive']
for i, true_label in enumerate(labels):
    row = f"VADER {true_label:<7}"
    for j in range(3):
        row += f"{cm[i][j]:^10}"
    print(row)

print("\n" + "="*70)
print(" CLASSIFICATION RESULTS")
print("="*70)

# Sentiment Distribution
print("\n SENTIMENT DISTRIBUTION:")
print("-" * 40)
```

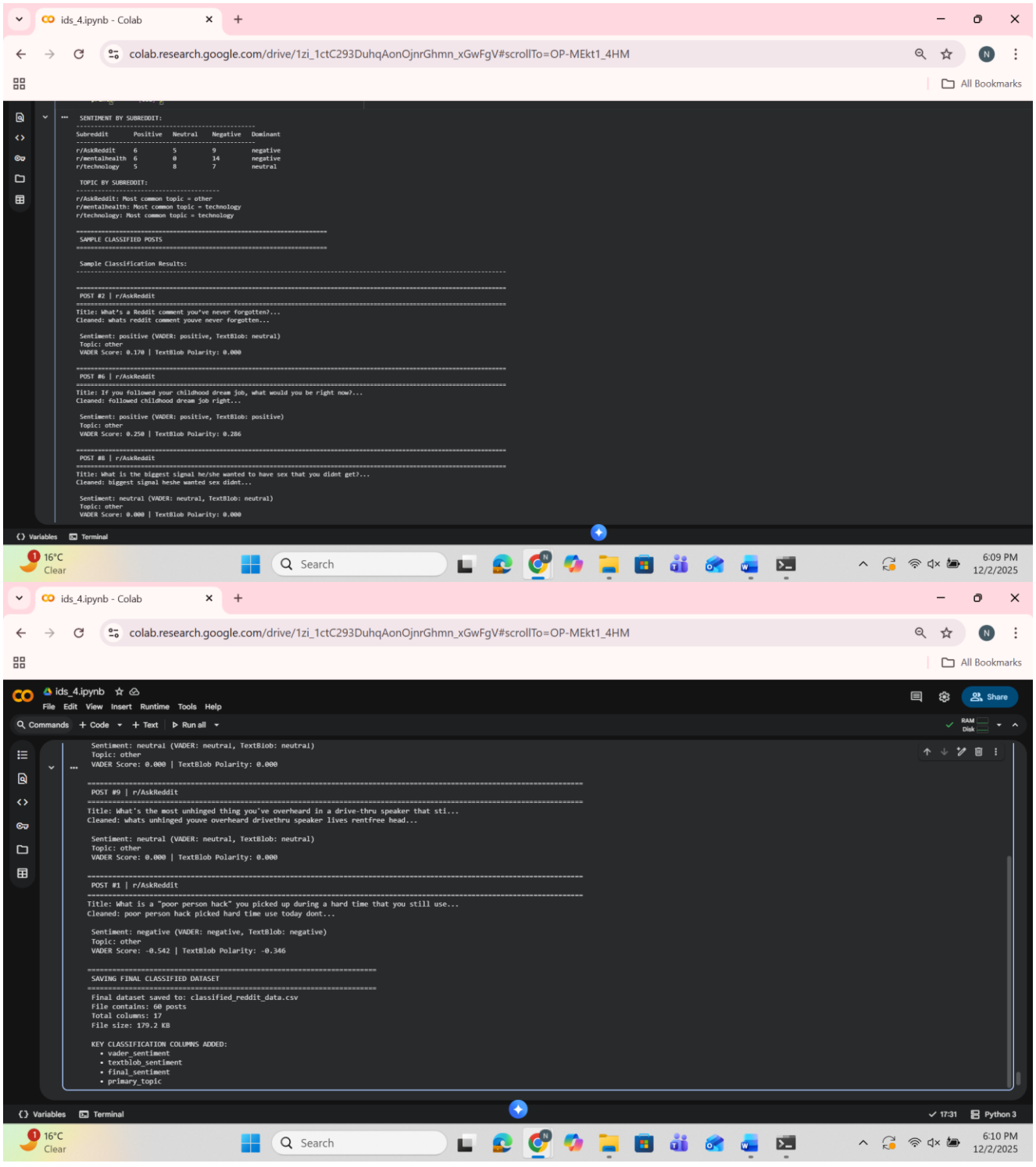
```
sentiment_counts = df['final_sentiment'].value_counts()
total = len(df)

print(f"{'Sentiment':<10} {'Count':<8} {'Percentage':<12} {'Emoji':<5}")
print("-" * 40)

emoji_map = {'positive': '😊', 'neutral': '😐', 'negative': '😔'}
for sentiment, count in sentiment_counts.items():
    percentage = (count / total) * 100
    emoji = emoji_map.get(sentiment, '❓')
    print(f"{sentiment:<10} {count:<8} {percentage:<11.1f}% {emoji:<5}")

# Topic Distribution
print("\n TOPIC DISTRIBUTION:")
print("-" * 40)

topic_counts = df['primary_topic'].value_counts()

print(f"{'Topic':<15} {'Count':<8} {'Percentage':<12}")
print("-" * 40)

for topic, count in topic_counts.items():
    percentage = (count / total) * 100
    print(f"{topic:<15} {count:<8} {percentage:<11.1f}%")

# Sentiment by Subreddit
print("\n SENTIMENT BY SUBREDDIT:")
print("-" * 50)

print(f"{'Subreddit':<15} {'Positive':<10} {'Neutral':<10} {'Negative':<10} {'Dominant':<10}")
print("-" * 50)

for subreddit in df['subreddit'].unique():
    sub_df = df[df['subreddit'] == subreddit]
    positive = len(sub_df[sub_df['final_sentiment'] == 'positive'])
    neutral = len(sub_df[sub_df['final_sentiment'] == 'neutral'])
    negative = len(sub_df[sub_df['final_sentiment'] == 'negative'])
    dominant = sub_df['final_sentiment'].value_counts().index[0]

    print(f"r/{subreddit:<13} {positive:<10} {neutral:<10} {negative:<10} {dominant:<10}")

# Topic by Subreddit
print("\n TOPIC BY SUBREDDIT:")
print("-" * 40)

for subreddit in df['subreddit'].unique():
    sub_df = df[df['subreddit'] == subreddit]
    dominant_topic = sub_df['primary_topic'].value_counts().index[0]
    print(f"r/{subreddit}: Most common topic = {dominant_topic}")
```

```
print("\n" + "="*70)
print(" SAMPLE CLASSIFIED POSTS ")
print("="*70)

# Select diverse samples
samples = []
for sentiment in ['positive', 'neutral', 'negative']:
    sentiment_samples = df[df['final_sentiment'] == sentiment].head(2).index.tolist()
    samples.extend(sentiment_samples)

samples = samples[:5]

print("\n Sample Classification Results:")
print("-" * 120)

for idx in samples:
    post = df.loc[idx]
    print(f"\n{'='*120}")
    print(f" POST #{idx + 1} | r/{post['subreddit']}")
    print(f"{'='*120}")
    print(f"Title: {post['title'][:80]}...")
    print(f"Cleaned: {post['cleaned_title'][:100]}...")
    print(f"\n Sentiment: {post['final_sentiment']} (VADER: {post['vader_sentiment']}, TextBlob:
{post['textblob_sentiment']})")
    print(f" Topic: {post['primary_topic']}")
    print(f" VADER Score: {post['vader_compound']:.3f} | TextBlob Polarity:
{post['textblob_polarity']:.3f}")

print("\n" + "="*70)
print(" SAVING FINAL CLASSIFIED DATASET")
print("="*70)

# Reorder columns for better readability
column_order = [
    'subreddit', 'title', 'cleaned_title', 'body', 'cleaned_body',
    'author', 'upvotes', 'num_comments', 'timestamp',
    'vader_sentiment', 'vader_compound',
    'textblob_sentiment', 'textblob_polarity',
    'final_sentiment', 'primary_topic',
    'full_text', 'cleaned_full_text'
]

# Select only existing columns
existing_columns = [col for col in column_order if col in df.columns]
remaining_columns = [col for col in df.columns if col not in existing_columns]
final_columns = existing_columns + remaining_columns

df_final = df[final_columns]

# Save to CSV
output_file = "classified_reddit_data.csv"
```

```
df_final.to_csv(output_file, index=False, encoding='utf-8')

print(f" Final dataset saved to: {output_file}")
print(f" File contains: {len(df_final)} posts")
print(f" Total columns: {len(df_final.columns)}")
print(f" File size: {df_final.memory_usage(deep=True).sum() / 1024:.1f} KB")

print(f"\n KEY CLASSIFICATION COLUMNS ADDED:")
classification_cols = ['vader_sentiment', 'textblob_sentiment', 'final_sentiment', 'primary_topic']
for col in classification_cols:
    print(f"   • {col}")
```

Output:





Explanation:

This code performs **sentiment and topic classification** on preprocessed Reddit posts. It uses **VADER and TextBlob** to analyze sentiment and combines their results for better accuracy. Posts are also classified into topics using keyword matching. Finally, it generates summary statistics, shows model agreement, and saves the fully classified dataset for further analysis.

Deliverables:

- Classification code
- Confusion matrix or accuracy score (if applicable)
- Final classified dataset (CSV)

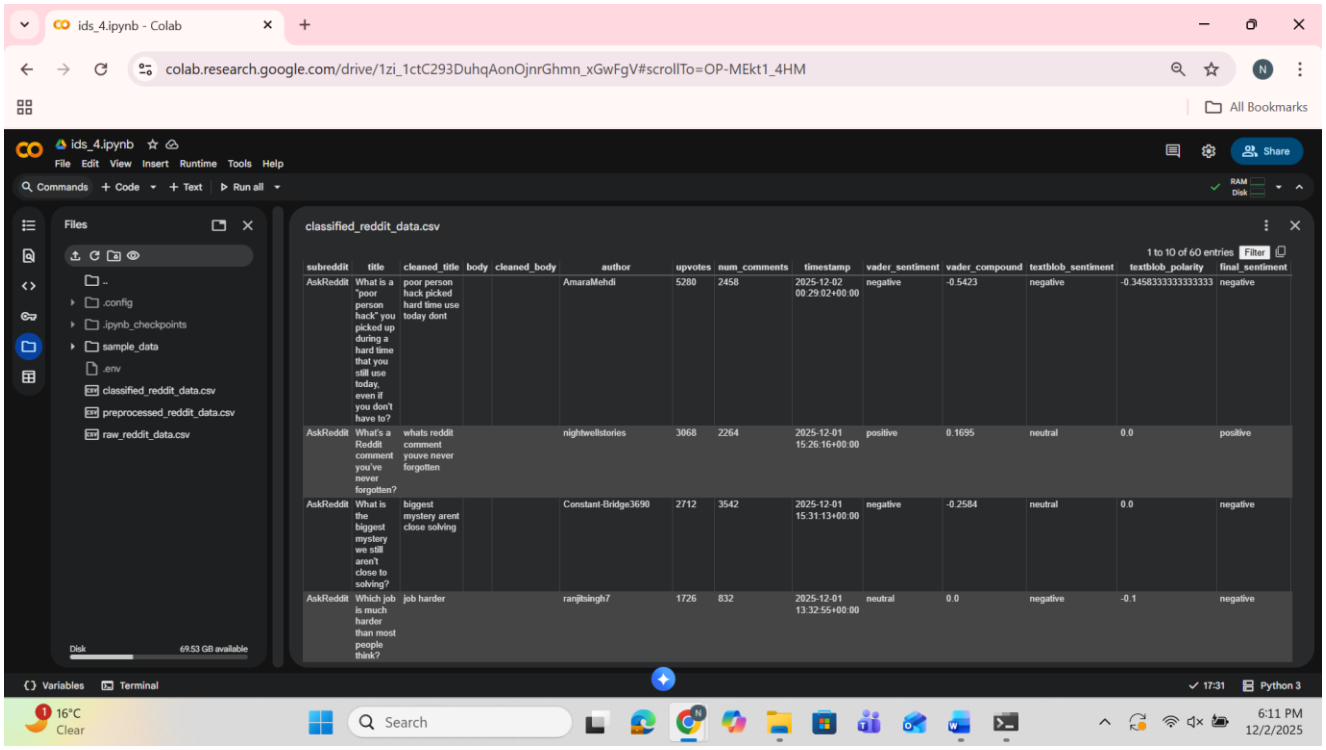
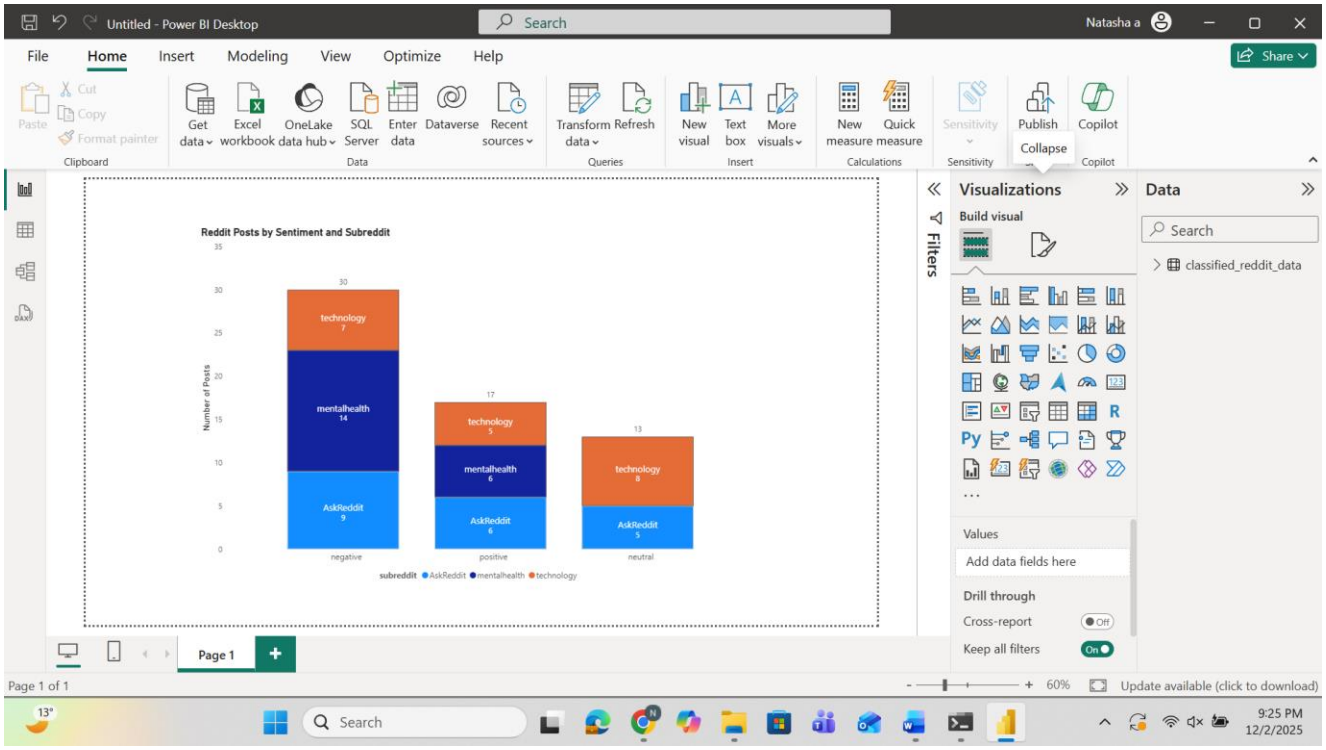


Figure 3: Classification results

Part 4: Power BI Dashboard Integration (6 Marks)

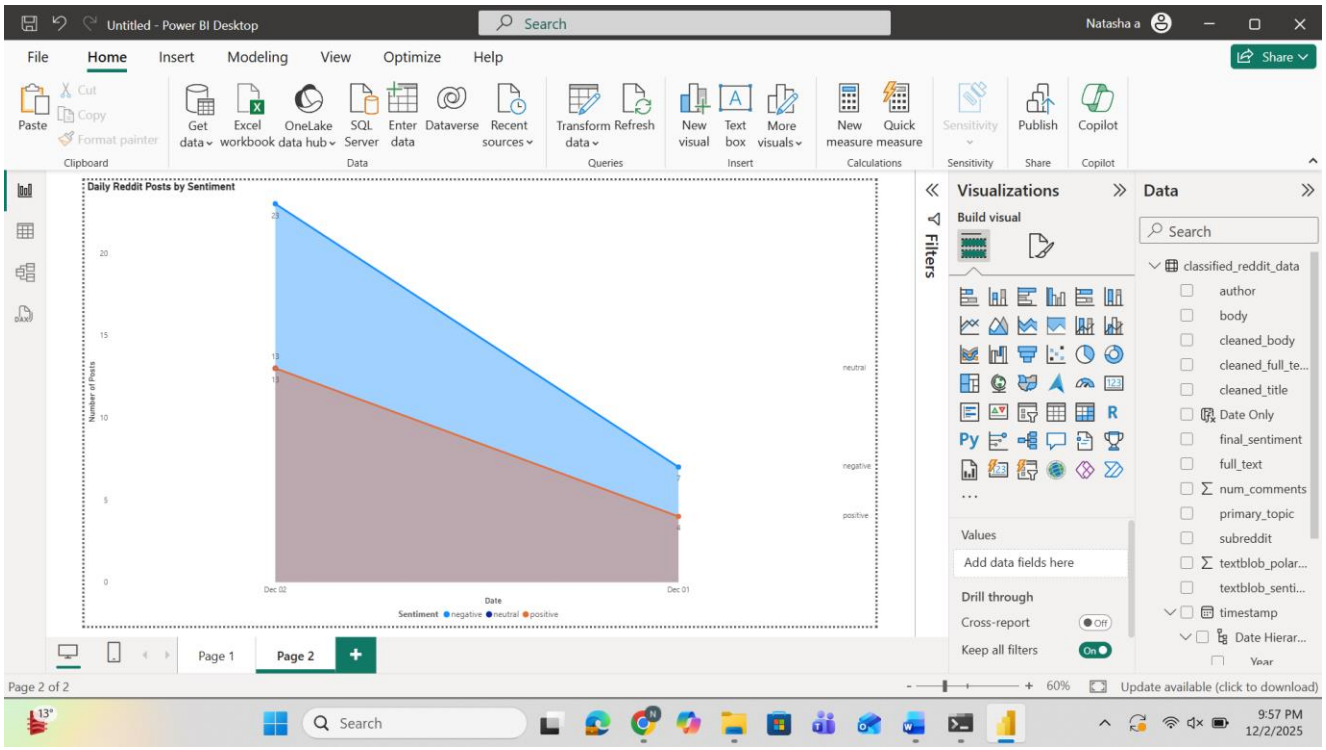
1. Import the `classified_reddit_data.csv` into **Power BI Desktop**
2. Design at least **3 visuals**, such as:
    - Bar chart of post counts by category/sentiment



Explanation:

This visualization displays post counts categorized by sentiment (negative/positive/neutral) with subreddit segmentation. Analysis reveals negative posts are most frequent overall, AskReddit dominates across all sentiment categories, technology discussions show more positive engagement, while mental health content appears primarily in negative and neutral topics

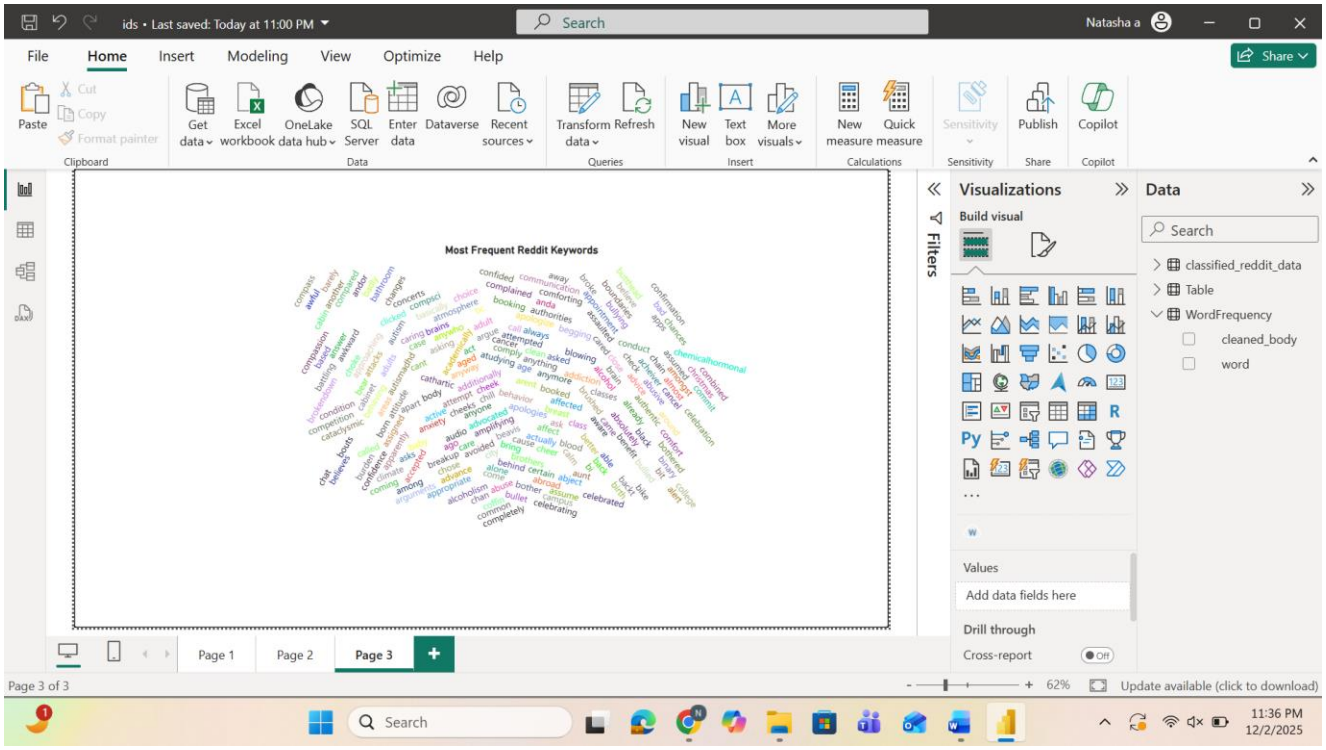
- Time series of posts over time



Explanation:

This time series shows sentiment volatility between December 1st and 2nd. Negative discussions dominated on December 1st but plummeted the following day. Meanwhile, neutral posts doubled to become most prevalent, while positive sentiment showed moderate growth. The chart reveals significant day-to-day shifts in community emotional tone.

- Word cloud or keyword frequency

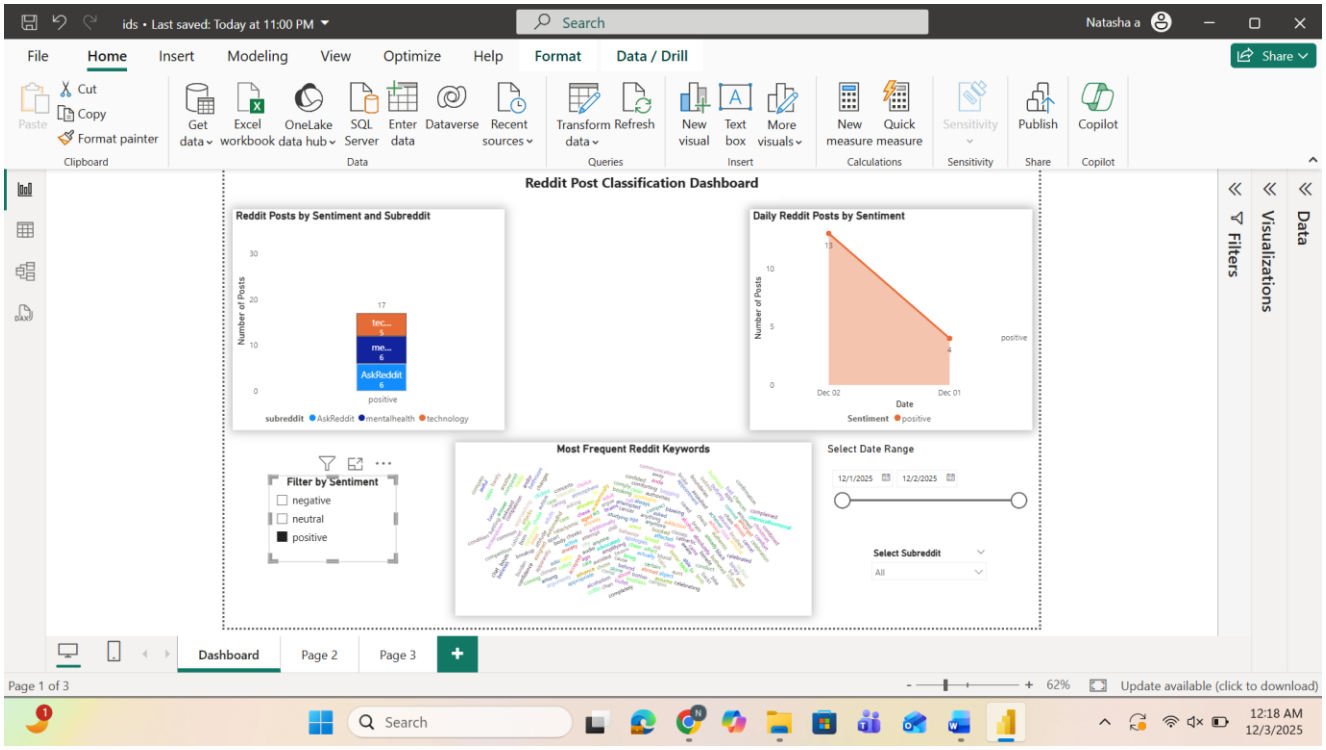


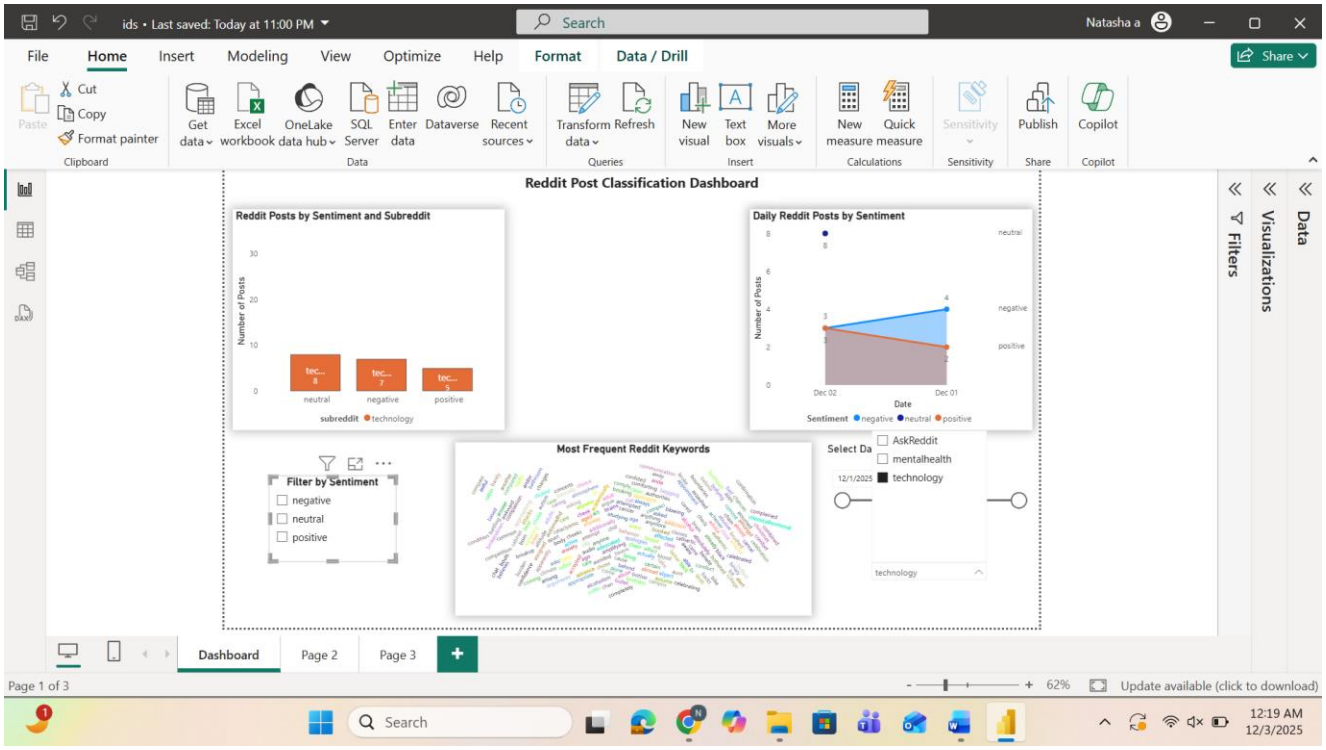
Explanation:

Keyword Frequency Analysis

This word cloud visualizes common terms in Reddit discussions, with word size indicating frequency. It reveals dominant topics like platform engagement, personal experiences, and recurring themes that drive community conversations.

3. Add filters (e.g., by subreddit, date, classification)





Explanation:

Interactive filters allow users to filter data by sentiment, date range, and subreddit community. These controls enable customized analysis of specific data segments, transforming the dashboard into an interactive exploration tool for targeted insights.

Deliverables:

- Power BI `.pbix` file
- Screenshots of the dashboard
- Explanation of each visual

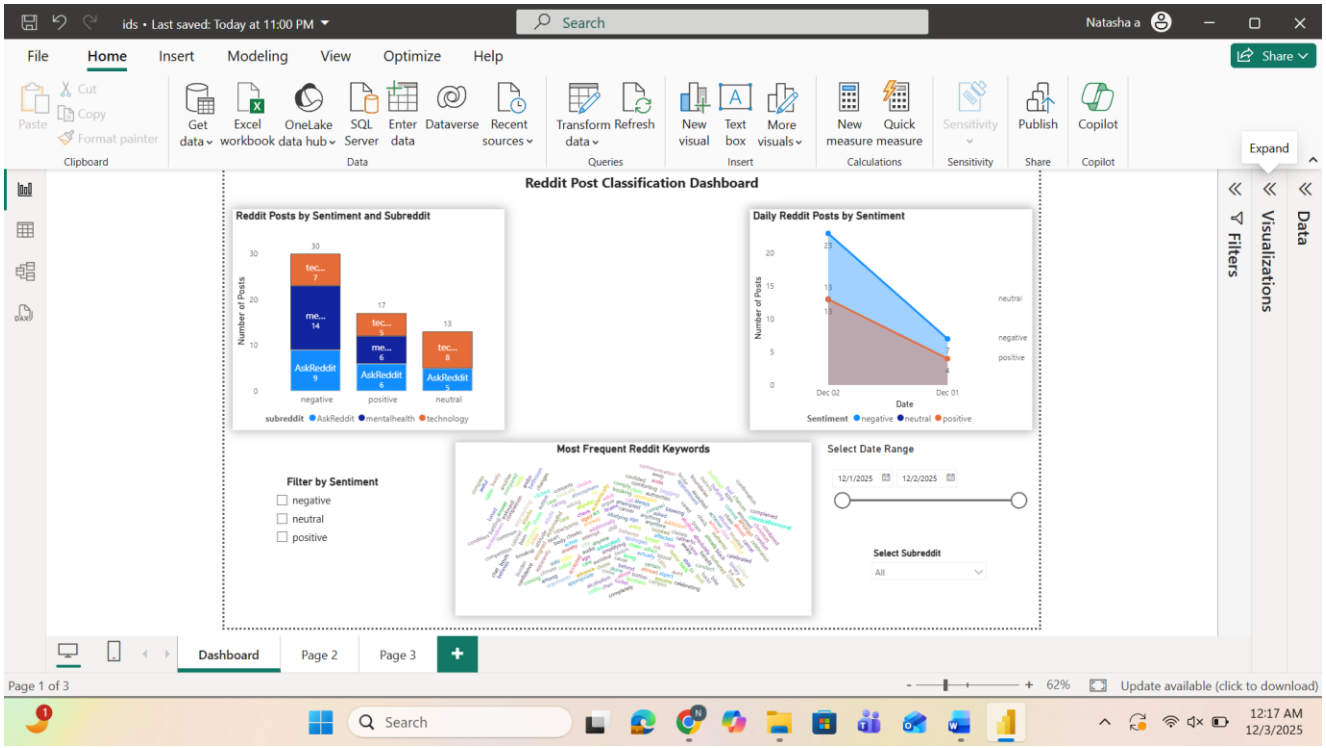


Figure 4: Reddit Post Classification Dashboard

Part 5: Report & Reflection (2 Marks)

Write a short report (PDF or in your notebook) covering:

- Steps taken from API → Cleaning → Classification → Visualization
- Tools and libraries used
- Challenges faced during integration
- Future improvements (e.g., real-time analysis, better models)

Include this in your final submission as a **PDF summary**.

Figure 5: Pipeline overview diagram (optional but encouraged)

Submission Checklist

Submit a ZIP file named:
`Assignment4_<YourName>_<RollNo>.zip`

Include:

- Python scripts or notebook
- `raw_reddit_data.csv` and `classified_reddit_data.csv`
- Power BI file (`.pbix`)
- PDF summary report (max 4 pages)
- Screenshots of:

- API output
- Preprocessed samples
- Classification code/results
- Power BI visuals

Grading Rubric

Part	Description	Marks
Part 1	Reddit API Integration	4
Part 2	Text Preprocessing	3
Part 3	NLP Classification	5
Part 4	Power BI Dashboard	6
Part 5	Report & Reflection	2
Total		**20**

Optional Extensions (Extra Credit)

- Real-time data refresh using scheduled scraping

CODE:

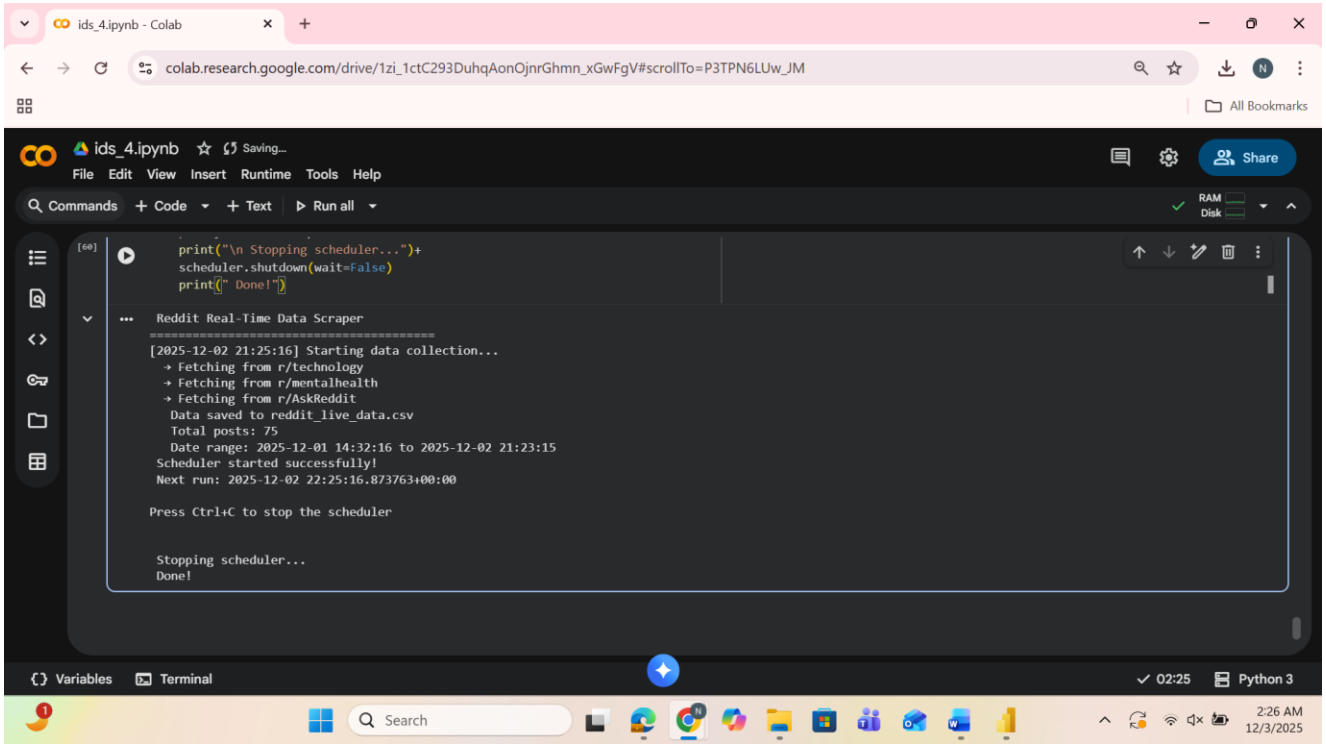
```
from apscheduler.schedulers.background import BackgroundScheduler
import praw
import pandas as pd
from datetime import datetime
import os
import time
from dotenv import load_dotenv

# Load Reddit credentials
load_dotenv()
CLIENT_ID = os.getenv("REDDIT_CLIENT_ID")
CLIENT_SECRET = os.getenv("REDDIT_SECRET")
USER_AGENT = os.getenv("REDDIT_USER_AGENT")

reddit = praw.Reddit(client_id=CLIENT_ID,
                     client_secret=CLIENT_SECRET,
                     user_agent=USER_AGENT)
reddit.read_only = True

def fetch_reddit_posts():
    """Fetch and save Reddit posts with error handling"""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print(f"[{timestamp}] Starting data collection...")

    try:
        subreddits = ["technology", "mentalhealth", "AskReddit"]
        posts_per_subreddit = 20
        all_posts = []

        for sub in subreddits:
```

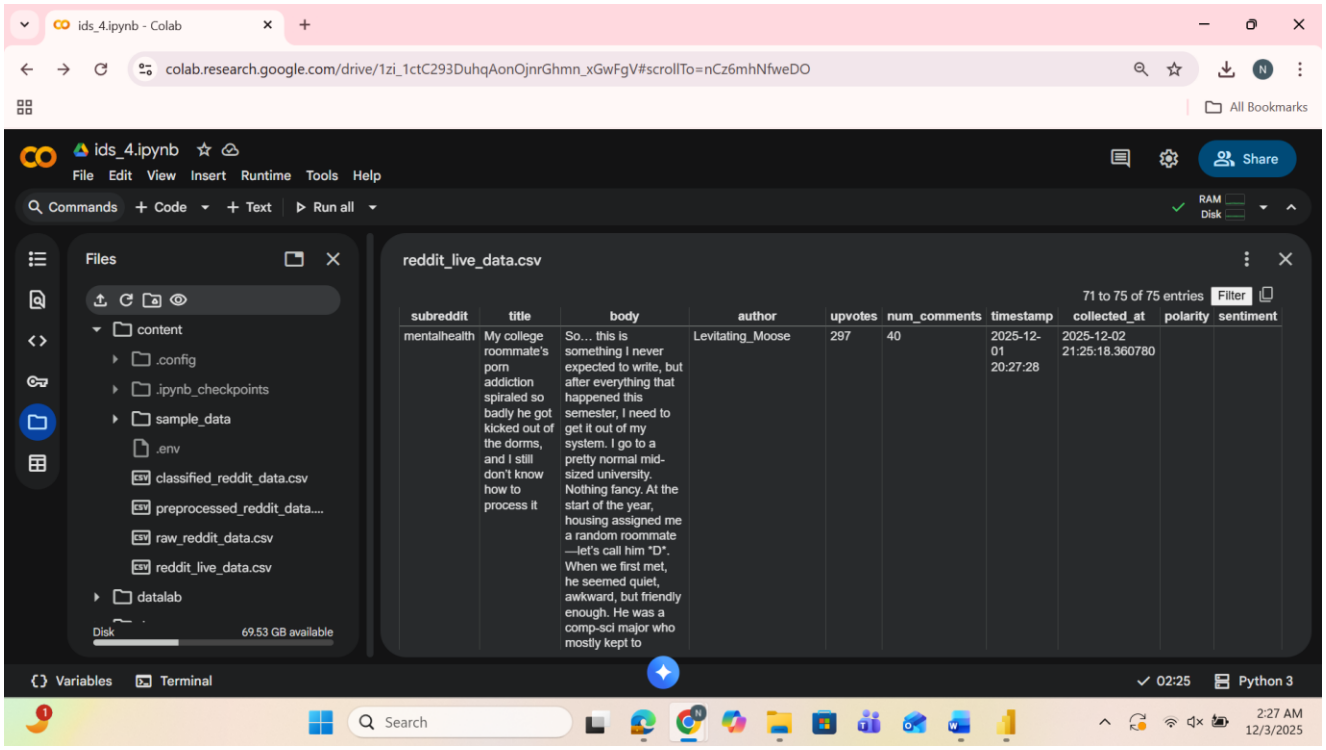
```
    try:
        print(f"  → Fetching from r/{sub}")
        for post in reddit.subreddit(sub).hot(limit=posts_per_subreddit + 5):
            if post.stickied:
                continue

            all_posts.append({
                "subreddit": sub,
                "title": post.title,
                "body": post.selftext if post.selftext else "",
                "author": str(post.author) if post.author else "[deleted]",
                "upvotes": post.score,
                "num_comments": post.num_comments,
                "timestamp": datetime.utcfromtimestamp(post.created_utc),
                "collected_at": datetime.now()
            })

    except Exception as sub_error:
        print(f"   Error fetching from r/{sub}: {str(sub_error)}")
        continue

# Create DataFrame from new posts
df_new = pd.DataFrame(all_posts)

output_file = "reddit_live_data.csv"

# Check if file exists and append data
if os.path.exists(output_file):
    try:
        existing_df = pd.read_csv(output_file)

        # Convert timestamp strings to datetime
        if 'timestamp' in existing_df.columns:
            existing_df['timestamp'] = pd.to_datetime(existing_df['timestamp'])

        if 'collected_at' in existing_df.columns:
            existing_df['collected_at'] = pd.to_datetime(existing_df['collected_at'])

        # Combine old and new data
        df_combined = pd.concat([existing_df, df_new], ignore_index=True)

        # Remove duplicates (based on title + timestamp)
        df_combined = df_combined.drop_duplicates(
            subset=['title', 'timestamp'],
            keep='last'
        )

        # Sort by timestamp (descending)
        df_combined = df_combined.sort_values('timestamp', ascending=False)

        # Keep only latest 1000 posts
```

```
            if len(df_combined) > 1000:
                df_combined = df_combined.head(1000)

            # Use combined data
            df_final = df_combined

        except Exception as e:
            print(f"   Error processing existing file, using new data only: {str(e)}")
            df_final = df_new
    else:
        # File doesn't exist, use only new data
        df_final = df_new

    # Save to CSV
    df_final.to_csv(output_file, index=False)

    print(f"   Data saved to {output_file}")
    print(f"   Total posts: {len(df_final)}")

    # Display date range only if we have data
    if len(df_final) > 0 and 'timestamp' in df_final.columns:
        try:
            min_date = df_final['timestamp'].min()
            max_date = df_final['timestamp'].max()
            print(f"   Date range: {min_date} to {max_date}")
        except:
            pass  # Skip if there's an error with date formatting

except Exception as e:
    print(f"[{datetime.now()}]  Error in fetch_reddit_posts: {str(e)}")

# Simple scheduler setup
scheduler = BackgroundScheduler()

# Schedule the job
scheduler.add_job(
    func=fetch_reddit_posts,
    trigger='interval',
    minutes=60,
    id='reddit_scraper',
    name='Reddit Data Scraper',
    replace_existing=True
)

# Run immediately and start scheduler
print(" Reddit Real-Time Data Scraper")
print("=" * 40)

# First run
fetch_reddit_posts()
```

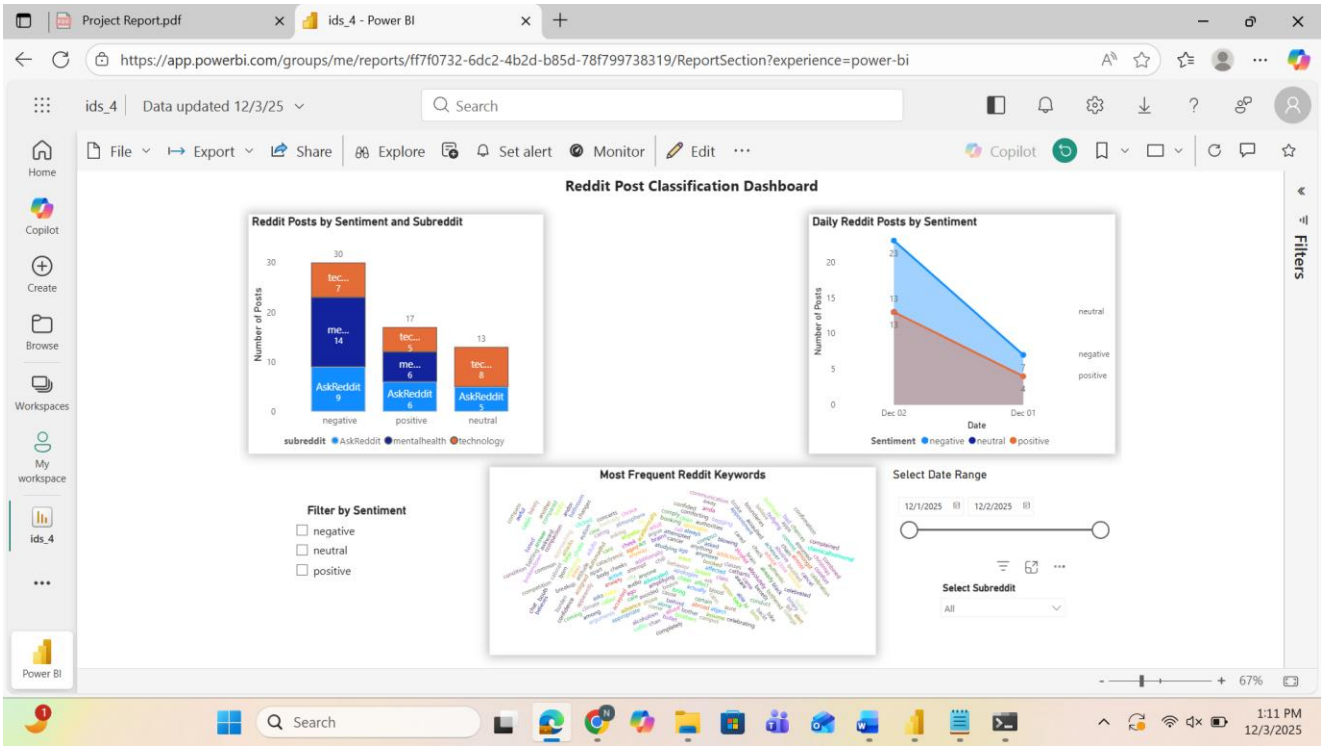
```
# Start scheduler
scheduler.start()
print(" Scheduler started successfully!")

if scheduler.get_jobs():
    print(f" Next run: {scheduler.get_jobs()[0].next_run_time}")

print("\nPress Ctrl+C to stop the scheduler\n")

# Keep main thread alive
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("\n Stopping scheduler...")
    scheduler.shutdown(wait=False)
    print(" Done!")
```

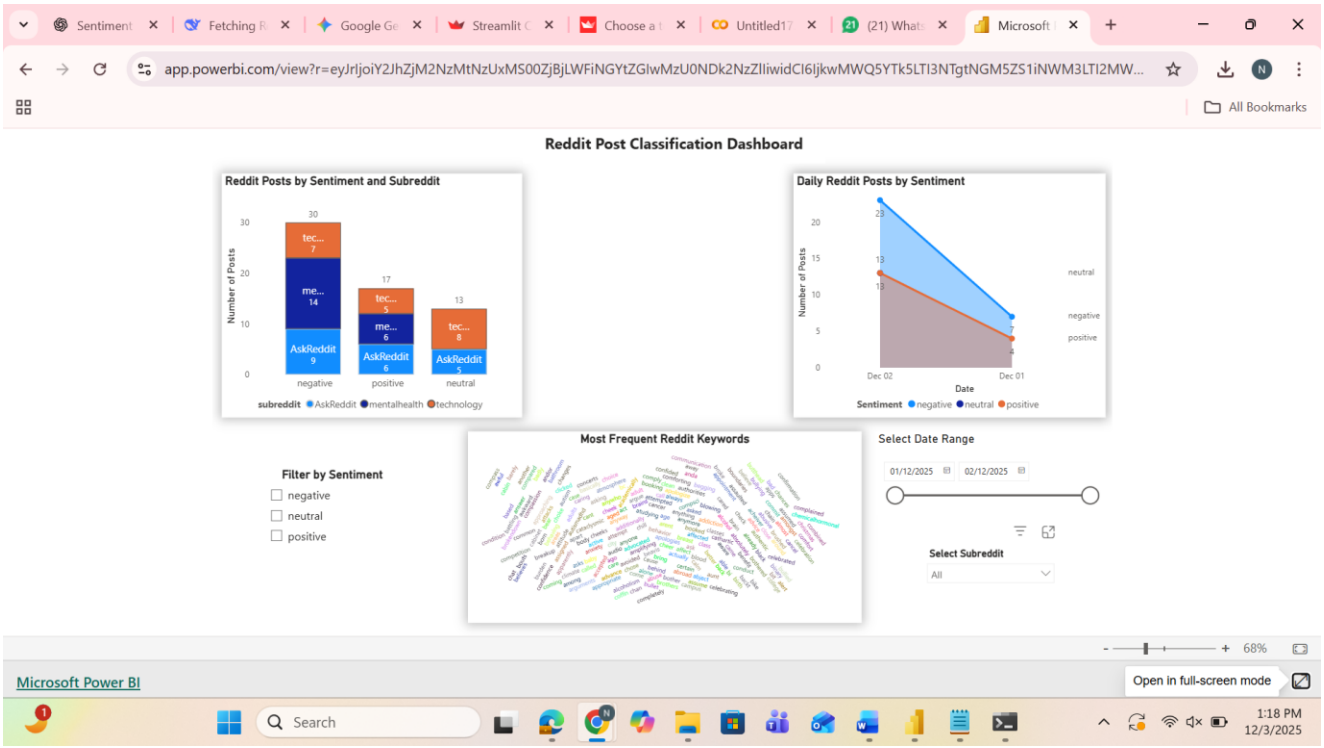
OUTPUT:





- Deploy Python app using Streamlit for frontend
- Publish Power BI dashboard to Power BI Web and share public link





Publish Power BI Dashboard to Power BI Web

Public Dashboard Link:
https://app.powerbi.com/view?r=eyJrIjoiY2JhZjM2NzMtNzUxMS00ZjBjLWFiNGYtZGIwMzU0NDk2NzZlIiwidCI6IjkwMWQ5YTk5LTI3NTgtNGM5ZS1iNWM3LTI2MWM2OTIwZmQzNyIsImMiOjl9

Status: Publicly accessible (No login required)

Implementation:

The Power BI dashboard containing sentiment analysis and topic classification visualizations was published to Microsoft's Power BI web service. Using the "Publish to web" feature, a public embed link was generated that allows anyone with internet access to view and interact with the dashboard.

Tools and Libraries

Area	Recommended Tools
API	`praw`, `requests`, `json`
Text NLP	`nltk`, `spaCy`, `TextBlob`, `transformers`, `sklearn`
Visualization	**Power BI Desktop**
Data Storage	`pandas`, `.csv`, `.xlsx`
Evaluation	`sklearn.metrics`, `matplotlib`, `seaborn`