Technical Documentation - YouTube Sentiment Analysis Project

📖 Code Structure & Explanation

This document provides detailed technical explanations of the code implementation.

---

1. Data Collection Module (`youtube_scraper.py`)

Overview

This module interfaces with the YouTube Data API v3 to collect comments from specified videos.

Key Components

1.1 API Authentication

```python
from dotenv import load_dotenv
import os

load_dotenv()
API_KEY = os.getenv('YOUTUBE_API_KEY')
youtube = build('youtube', 'v3', developerKey=API_KEY)
```

Explanation:

- Uses `python-dotenv` to securely load API credentials
- Initializes the YouTube API client with authentication
- `developerKey` parameter authenticates all API requests

1.2 Video ID Extraction

```python
def extract_video_id(url):
    if 'youtube.com/watch?v=' in url:
        return url.split('watch?v=')[1].split('&')[0]
    elif 'youtu.be/' in url:
        return url.split('youtu.be/')[1].split('?')[0]
    else:
        return url
```

Explanation:

- Handles multiple YouTube URL formats
- Extracts the 11-character video ID
- Returns ID directly if already in correct format

1.3 Comment Collection

```python
def get_video_comments(video_id, max_comments=500):
    request = youtube.commentThreads().list(
        part='snippet,replies',
        videoId=video_id,
        maxResults=100,
        order='relevance',
        textFormat='plainText'
    )
```

API Parameters:

- `part='snippet,replies'`: Fetches comment data and replies
- `maxResults=100`: Maximum per request (API limit)
- `order='relevance'`: Sorts by most relevant first
- `textFormat='plainText'`: Returns text without HTML tags

Pagination Logic:

```python
```

```
while response and len(comments_data) < max_comments:
    # Process current page
    for item in response['items']:
        # Extract comment data

    # Get next page
    if 'nextPageToken' in response:
        request = youtube.commentThreads().list(
            part='snippet,replies',
            videoId=video_id,
            pageToken=response['nextPageToken'],
            maxResults=100
        )
        response = request.execute()
```

Explanation:

- API returns max 100 comments per request
- Uses pagination tokens to fetch additional pages
- Continues until reaching `max_comments` limit

1.4 Data Structure

```
python

comments_data.append({
    'comment_id': item['id'],
    'video_id': video_id,
    'author': comment['authorDisplayName'],
    'text': text,
    'like_count': comment['likeCount'],
    'published_at': comment['publishedAt'],
    'reply_count': item['snippet']['totalReplyCount'],
    'is_reply': False
})
```

Fields Collected:

- `comment_id`: Unique identifier
- `video_id`: Links comment to video
- `author`: Commenter's display name
- `text`: Comment content

- `like_count`: Number of likes
- `published_at`: Timestamp (ISO 8601 format)
- `reply_count`: Number of replies
- `is_reply`: Boolean flag for nested comments

---

2. Data Cleaning Module (`data_cleaning.py`)

Overview

Preprocesses raw comment data to prepare for sentiment analysis.

Key Components

2.1 Text Cleaning Function

```python
def clean_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove emails
    text = re.sub(r'\S+@\S+', '', text)

    # Remove mentions
    text = re.sub(r'@\w+', '', text)

    # Remove special characters
    text = re.sub(r'[^a-zA-Z\s.,!?]', '', text)

    # Remove extra whitespace
    text = ' '.join(text.split())

    return text.strip()
```

Regular Expression Patterns:

- `r'http\S+'`: Matches URLs starting with http/https
- `r'\S+@\S+'`: Matches email addresses

- `r'@\w+'`: Matches @ mentions
- `r'[^a-zA-Z\s.,!?]'`: Keeps only letters and basic punctuation

2.2 Duplicate Removal

```
python

df = df.drop_duplicates(subset=['text'], keep='first')
```

Explanation:

- Removes exact duplicate comments
- Keeps first occurrence
- Prevents bias in sentiment analysis

2.3 Empty Comment Filtering

```
python

df = df[df['text'].notna()]
df = df[df['text'].str.strip() != '']
```

Explanation:

- Removes null values
- Removes whitespace-only comments
- Ensures all data is analyzable

---

3. Sentiment Analysis Module (`sentiment_analysis.py`)

Overview

Analyzes emotional tone of comments using NLP techniques.

3.1 VADER Sentiment Analysis

How VADER Works

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analyzer specifically attuned to social media text.

Key Features:

- Pre-trained on social media text
- Handles emoticons, slang, and abbreviations
- Returns compound score from -1 (negative) to +1 (positive)

Implementation

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def analyze_sentiment(text, analyzer):
    scores = analyzer.polarity_scores(text)
    compound = scores['compound']

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

    return compound, sentiment
```

VADER Output:

```python
{
    'neg': 0.0,     # Negative score
    'neu': 0.508,   # Neutral score
    'pos': 0.492,   # Positive score
    'compound': 0.128 # Aggregate score
}
```

Classification Thresholds:

- `compound >= 0.05`: Positive
- `compound <= -0.05`: Negative
- `else`: Neutral

3.2 Alternative: TextBlob

```python
from textblob import TextBlob

def analyze_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity  # -1 to 1

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

    return polarity, sentiment
```

TextBlob vs VADER:

Feature	VADER	TextBlob
Training Data	Social media	General text
Speed	Faster	Slower
Emoticon Support	Yes	Limited
Accuracy (Social)	Higher	Lower
Easy Setup	Moderate	Easier

---

4. Data Structures & Flow

4.1 Data Flow Diagram

```
Input: YouTube URL
      ↓
YouTube API Request
      ↓
Raw JSON Response
      ↓
Pandas DataFrame (Raw)
```

```
    ↓
Text Cleaning (Regex)
    ↓
Pandas DataFrame (Cleaned)
    ↓
Sentiment Analysis (VADER/TextBlob)
    ↓
Pandas DataFrame (With Scores)
    ↓
Visualization (Matplotlib)
    ↓
Output: CSV + Charts
```

4.2 DataFrame Schema

Raw Comments DataFrame

```
Columns: [comment_id, video_id, author, text, like_count,
        published_at, reply_count, is_reply]
Types:   [str, str, str, str, int, str, int, bool]
```

Cleaned Comments DataFrame

```
Columns: [comment_id, video_id, author, text, like_count,
        published_at, reply_count, is_reply, cleaned_text]
Types:   [str, str, str, str, int, str, int, bool, str]
```

Final Results DataFrame

```
Columns: [comment_id, video_id, author, text, like_count,
        published_at, reply_count, is_reply, cleaned_text,
        sentiment_score, sentiment]
Types:   [str, str, str, str, int, str, int, bool, str,
        float, str]
```

---

5. Visualization Implementation

5.1 Pie Chart - Sentiment Distribution

```
python
```

```
sentiment_counts = df['sentiment'].value_counts()
colors = ['#2ecc71', '#95a5a6', '#e74c3c']  # green, gray, red

axes[0].pie(sentiment_counts.values,
        labels=sentiment_counts.index,
        autopct='%1.1f%%',
        colors=colors,
        startangle=90)
```

Explanation:

- `value_counts()`: Counts occurrences of each sentiment
- `autopct='%1.1f%%'`: Shows percentages with 1 decimal
- `startangle=90`: Starts from top of circle
- Color scheme: Green (positive), Gray (neutral), Red (negative)

5.2 Histogram - Score Distribution

```
python

axes[1].hist(df['sentiment_score'],
         bins=30,
         color='#3498db',
         edgecolor='black',
         alpha=0.7)
axes[1].axvline(x=0, color='red', linestyle='--', linewidth=2)
```

Explanation:

- `bins=30`: Divides range into 30 intervals
- `alpha=0.7`: Sets transparency
- `axvline(x=0)`: Vertical line at neutral point
- Shows distribution curve of sentiment scores

---

6. Error Handling

6.1 API Errors

```
python
```

```
try:
    request = youtube.commentThreads().list(...)
    response = request.execute()
except Exception as e:
    if 'commentsDisabled' in str(e):
        print("Comments are disabled")
    elif 'quotaExceeded' in str(e):
        print("API quota exceeded")
    else:
        print(f"Error: {e}")
```

Common API Errors:

- `commentsDisabled`: Video has comments turned off
- `quotaExceeded`: Daily API limit reached
- `videoNotFound`: Invalid video ID
- `forbidden`: API key restrictions

6.2 Data Processing Errors

```
python

if not text or text.strip() == '':
    return 0, 'neutral'
```

Handles:

- Empty strings
- Null values
- Whitespace-only text

---

7. Performance Optimization

7.1 Batch Processing

```
python

# Process 100 comments per API call
maxResults=100
```

7.2 Rate Limiting

```python
time.sleep(0.5)  # Small delay between requests
```

Prevents:

- API rate limit errors
- Server overload
- IP blocking

7.3 Memory Management

```python
# Use iterator for large datasets
for chunk in pd.read_csv('file.csv', chunksize=1000):
    process(chunk)
```

---

8. Data Quality Metrics

8.1 Cleaning Effectiveness

```python
original_count = len(df)
df = df.drop_duplicates()
cleaned_count = len(df)
print(f"Removed {original_count - cleaned_count} duplicates")
```

8.2 Sentiment Distribution Balance

```python
sentiment_counts = df['sentiment'].value_counts(normalize=True)
print(f"Positive: {sentiment_counts['positive']*100:.1f}%")
```

---

9. API Quota Management

9.1 Quota Costs

```
Operation            | Cost (Units)
-------------------- | ------------
search.list          | 100
videos.list          | 1
commentThreads.list  | 1
```

9.2 Daily Limit Strategy

```python
# Calculate estimated quota usage
num_videos = 5
comments_per_video = 500
pages_per_video = math.ceil(comments_per_video / 100)

total_quota = (
    100 +                       # Initial search
    (num_videos * 1) +          # Video details
    (num_videos * pages_per_video * 1) # Comment pages
)

print(f"Estimated quota usage: {total_quota} units")
```

---

10. Testing & Validation

10.1 Unit Tests Example

```python
def test_clean_text():
    assert clean_text("Check out http://example.com") == "check out"
    assert clean_text("Email me@example.com") == "email"
    assert clean_text("@user hello!") == "hello"
```

10.2 Data Validation

```python
```

```
# Validate sentiment scores are in range
assert df['sentiment_score'].between(-1, 1).all()

# Validate no empty cleaned text
assert (df['cleaned_text'].str.len() > 0).all()
```

---

11. Security Considerations

11.1 API Key Protection

```
python

# ✅ Good
load_dotenv()
API_KEY = os.getenv('YOUTUBE_API_KEY')

# ❌ Bad
API_KEY = "AIzaSy..."  # Never hardcode!
```

11.2 Input Validation

```
python

def extract_video_id(url):
    # Validate URL format before processing
    if not url:
        raise ValueError("URL cannot be empty")
    # ... rest of function
```

---

12. Scalability Considerations

For Large-Scale Analysis:

1. **Database Storage**: Use PostgreSQL/MongoDB instead of CSV
2. **Async Processing**: Use `asyncio` for parallel API calls
3. **Caching**: Store processed results to avoid re-analysis
4. **Distributed Processing**: Use Apache Spark for big data
5. **Cloud Deployment**: AWS Lambda or Google Cloud Functions

13. Dependencies Version Info

```python
# requirements.txt
google-api-python-client==2.80.0
pandas==2.0.0
python-dotenv==1.0.0
vaderSentiment==3.3.2
matplotlib==3.7.1
textblob==0.17.1  # Alternative to VADER
```

---

14. Algorithmic Complexity

Time Complexity

- Data Collection: O(n) where n = number of comments
- Text Cleaning: O(n × m) where m = average comment length
- Sentiment Analysis: O(n)
- Visualization: O(n)

Space Complexity

- O(n) for storing comments in memory
- Can be optimized with streaming for very large datasets

This technical documentation provides the foundation for understanding, extending, and maintaining the YouTube Sentiment Analysis project.