Therapeutic web scraper

Presentation

Project Overview

An advanced web application that scrapes, analyzes, and visualizes mental health discussions from Reddit, providing researchers and clinicians with AI-enhanced insights into public mental health discourse.

KEY FEATURES

1. Reddit Content Scraping

- Targets mental health subreddits (r/mentalhealth, r/depression, r/anxiety, etc.)
- Filter by keywords and post limits
- Smart filtering of moderator posts

2.Multi-level Content Analysis

- Sentiment analysis of posts
- AI-powered therapeutic insights
- Keyword extraction and trend identification

3.Interactive Dashboard

- Sentiment distribution visualization
- Top therapeutic keywords visualization
- Filterable content table with direct analysis access

4.Caching System

- Reduces API calls
- Improves application performance
- Implements intelligent cache filtering

REDDIT API INTEGRATION (PRAW)

```
# reddit_scraper.py
class RedditScraper:
    def __init__(self, client_id, client_secret, user_agent):
        self.reddit = praw.Reddit(
            client_id=client_id,
            client_secret=client_secret,
            user_agent=user_agent
        )

    def scrape_mentalhealth(self, keywords, limit=10):
        # Scrape mental health related posts
        subreddits = ['mentalhealth', 'depression', 'anxiety', 'therapy', 'CPTSD']
        # Implementation details...
```

- Authenticates with Reddit API
- Scrapes multiple mental health subreddits
- Filters by keywords and metadata

NLP: SENTIMENT ANALYSIS

```
# sentiment_analyzer.py
class SentimentAnalyzer:
    def __init__(self, model_name="distilbert-base-uncased-finetuned-sst-2-english"):
        # Initialize HuggingFace pipeline
        self.nlp = pipeline("sentiment-analysis", model=model_name)

    def analyze_text(self, text):
        # Process text (limit to 512 chars for model constraints)
        processed_text = text[:512]
        result = self.nlp(processed_text)[0]
        return {
            'label': result['label'],
            'score': result['score']
        }
```

- Uses Hugging Face Transformers
- Leverages pre-trained DistilBERT model
- Provides sentiment label and confidence score

GOOGLE GEMINI AI INTEGRATION

```
# gemini_analyzer.py
class GeminiAnalyzer:
    def __init__(self):
        self.api_key = os.getenv('GEMINI_API_KEY')
        self.base_url = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"

    def generate_insight(self, text):
        # Constructs prompt for mental health professional perspective
        prompt = (
            "Analyze this Reddit post from a mental health professional perspective. "
            "Identify key emotional themes, potential concerns, and provide "
            "supportive, clinically-informed insights. Keep response concise (3-4 sentences)."
            # Implementation details...
        )
```

- Integrates Google's Gemini 2.0 Flash model
- Generates therapeutic insights from mental health professional perspective
- Customizes prompt for clinically-informed analysis

SMART MOD POST FILTERING

```
def filter_mod_posts(posts: List[Dict]) -> List[Dict]:
    """Filter out any posts that might be from moderators with comprehensive checks"""
    mod_keywords = [
        'moderator', 'mod ', 'mods ', 'modding', 'moderation',
        'rules', 'rule', 'announcement', 'official', 'meta',
        'welcome', 'introduction', 'guideline', 'reminder', 'update'
    ]
```

- Multiple filtering layers
- Improves data quality by removing non-therapeutic content
- Applied during scraping and when loading from cache

INTELLIGENT CACHING SYSTEM

```
def get_cache_key(keywords, limit):
    """Generate a cache key based on search parameters"""
    key_string = f"{','.join(sorted(keywords))}-{limit}"
    return hashlib.md5(key_string.encode()).hexdigest()

def get_cached_results(cache_key):
    """Try to get results from cache with mod post filtering"""
    cache_file = f"data/cache_{cache_key}.json"
    if os.path.exists(cache_file):
        with open(cache_file, 'r', encoding='utf-8') as f:
            cached_data = json.load(f)
            return filter_mod_posts(cached_data)
    return None
```

- MD5 hashing for deterministic cache keys
- Automatic mod post filtering when loading from cache
- Reduces API usage and improves response time

DATA VISUALIZATION

Chart.jsfor interactive visualizations:

- Pie chart for sentiment distribution
- Bar chart for keyword frequency

Bootstrap 5 Components:

- Dynamic tables with filtering
- Progress bars for sentiment scores
- Modal dialogs for detailed analysis

TECHNOLOGIES & LIBRARIES

Backend (Python)

- Flask: Lightweight Python web framework
- Jinja2: Server-side templating with inheritance (base.html pattern)
- dotenv: Environment configuration management
- Typing: Type hints for improved code quality

Data Collection & Processing

- PRAW (Python Reddit API Wrapper): Sophisticated Reddit integration
- Requests: HTTP library for API interactions
- JSON: Data interchange format
- Logging: Comprehensive error and event tracking
- hashlib: MD5 hashing for cache management
- datetime: Timestamp processing

Natural Language Processing

- HuggingFace Transformers: Specifically uses "distilbert-base-uncased-finetuned-sst-2-english"
- Google Gemini API: Implements custom prompt engineering for mental health contexts
- Collections.Counter: Efficient keyword frequency tracking

Frontend Technologies

- Bootstrap 5: Modern responsive UI framework
- Chart.js: Interactive and responsive charts
- Bootstrap Icons: Visual iconography system
- Custom CSS: Specialized styling with responsiveness
- Fetch API: Asynchronous requests for smooth user experience

Error Handling Best Practices Implemented

1. **Granularity:** Errors are caught at appropriate boundaries
2. **Specificity:** Error messages describe what went wrong
3. **Recoverability:** Non-critical errors don't crash the system
4. **Transparency:** Errors are logged and communicated to users
5. **Graceful Degradation:** System provides partial functionality when components fail
6. **Defense in Depth:** Multiple validation layers prevent cascading failures

TECHNICAL CHALLENGES & SOLUTIONS

1.API Rate Limiting
Challenge: Reddit API has strict rate limits
Solution: Intelligent caching system
2.AI Response Quality
Challenge: Ensuring clinically appropriate insights
Solution: Carefully crafted prompts for Gemini API
3.Data Quality
Challenge: Filtering mod posts and irrelevant content
Solution: Multi-faceted filtering algorithm

Thank **You**

You can acess the application via:

Presented by
Rima Chaieb
BA/IT