Wear Your Mood

Big Data, The Self and Social Platforms

Prepared by Xingyu Pu
(23024979)

Table of Conten

GitHub：https://github.com/XingyuPu925/wear-your-mood-XingyuPu.git

Introduction

With the rapid growth of Internet technology and rising living standards, personalized fashion recommendation systems have become a popular research topic. Color, as a key element of fashion, can directly reflect and influence emotions. However, traditional color recommendation systems often focus on objective factors like season and skin tone, while overlooking users' emotional states.

This project presents an emotion-based fashion color recommendation system. It analyzes user-input emotional keywords, crawls matching color schemes from professional color websites, and uses deep learning to generate personalized outfit suggestions.

The system addresses three main challenges:

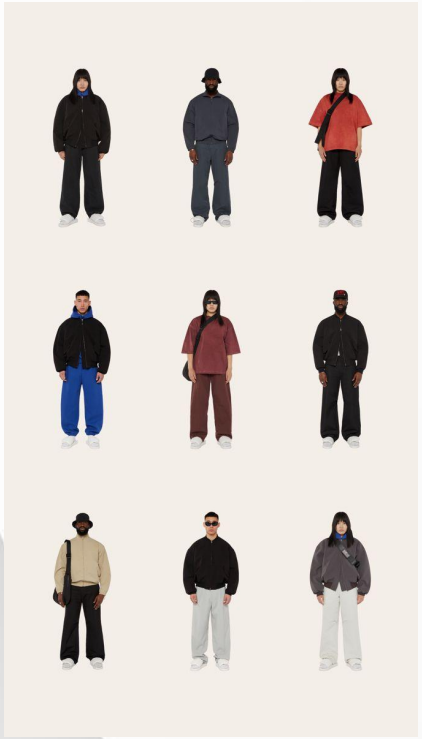
- How to find emotion-related color combinations online
- How to present these color palettes with interactive features
- How to turn color palettes into practical fashion advice

Built with Python Flask, the system combines web crawling (from sites like Coolors and ColorHunt) with DeepSeek API for fashion suggestion generation. A responsive front-end displays the results, making the system useful both for everyday users and fashion designers seeking color inspiration.

Introduction

User Group

• Emotion–sensitive youth: who are aware of their mental states and use clothing as self expression.
• Fashion–forward users: interested in trying color–based outfit suggestions tied to moods.
• Creative technologists: seeking interactive digital artworks merging data with aesthetics.
Especially relevant for Gen–Z, this platform combines fashion psychology, color theory, and interaction.



The primary users for “Wear Your Mood” are young, digital–native individuals—primarily GenZ—who are highly self–aware and accustomed to expressing their emotions through visual mediums. These users are luent in digital aesthetics, fashion culture, and emotional language.

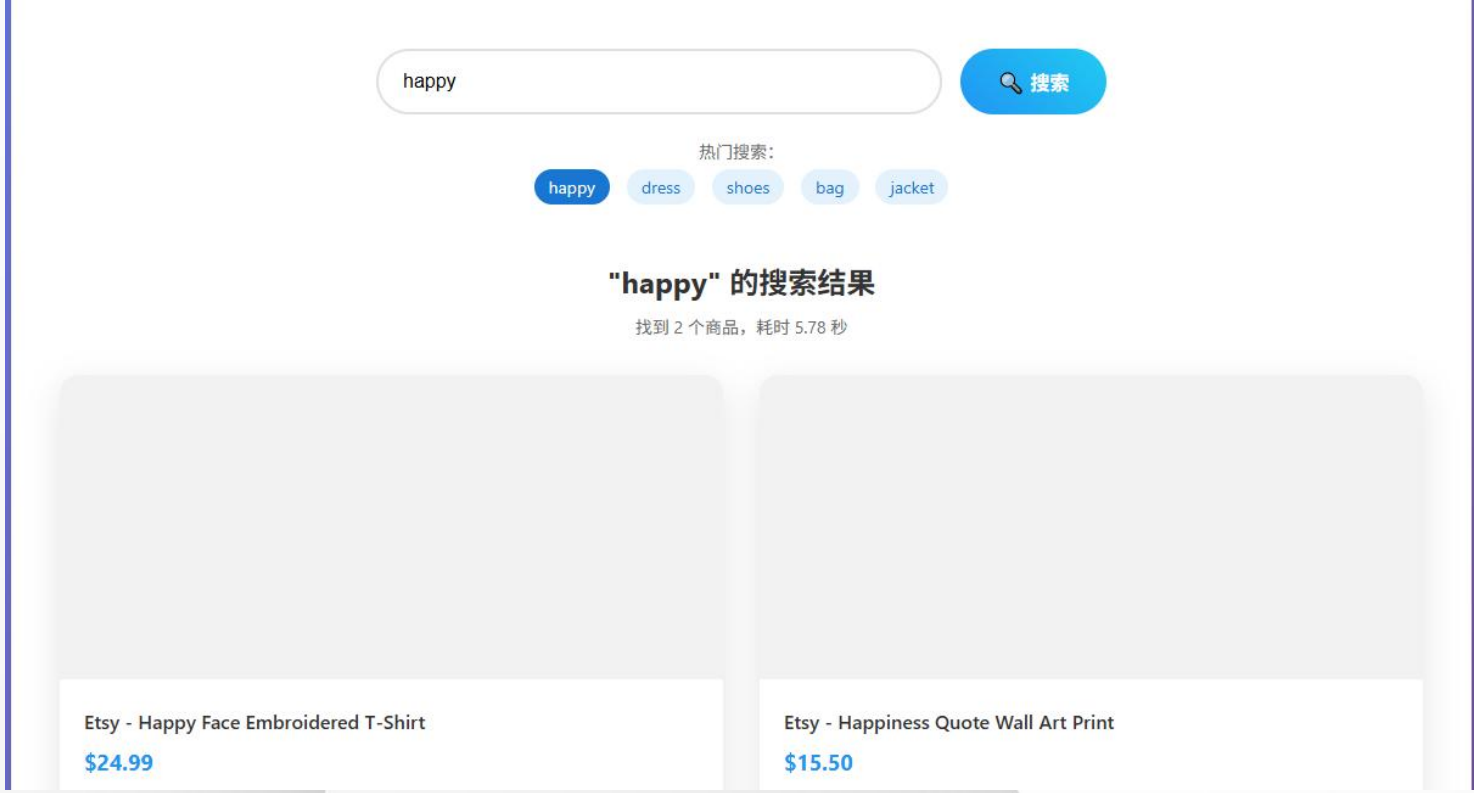
In an era of mood trackers, social filters, and Instagram aesthetics, fashion has become a key medium of emotional storytelling. Many users find comfort and empowerment by dressing in a way that reflects their mental state. However, there’s currently a gap between emotional data and fashion recommendations in an interactive, visual form.

This project bridges that gap—offering an intuitive, playful way for users to translate their inner feelings into wearable color combinations and artistic visuals. It enhances emotional awareness, supports mental wellness, and inspires creative dressing based on mood

Problem & Research

Initially, my idea was to collect product images from fashion websites based on users' emotional input. I targeted platforms such as ASOS.com and Pinterest, hoping to directly crawl fashion-related visuals linked to specific emotions. However, during implementation, I discovered that most of these websites actively block bots, preventing automated data collection.

For example, when scraping ASOS or Etsy, the responses only included product names and prices, but no valid image URLs. I tested many similar platforms, and all presented the same issue. On the other hand, websites without anti-scraping measures often lacked emotional keyword coverage, making the available images too limited or irrelevant.



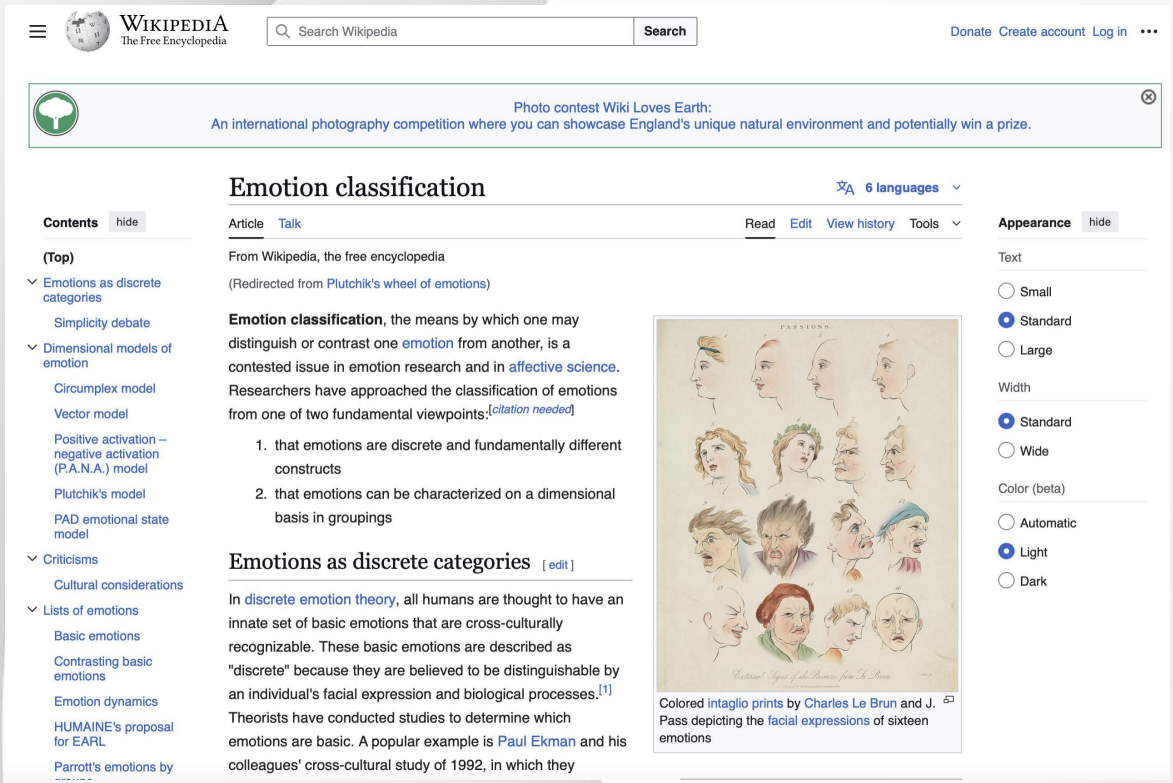
So I decided to get some color matching website color, because most of the color can get text, numbers, symbols (such as #334455 or rgb (255,255,255), etc.) can be generated in the front-end, compared to the picture crawling will be a little simpler, but in practice found that some of the site also has an anti-crawl mechanism, I decided to use a mixed collection strategy, combined with direct access to the target website and Google search in two ways, has enhanced the scope of crawling, and finally finally let most of the emotional keywords are able to get the corresponding color matching suggestions

Color Psychology

https://www.verywellmind.com/color-psychology-2795824

Color psychology suggests that colors have a significant impact on human emotions and behavior. Warm colors such as red, orange, and yellow are often associated with feelings of energy, excitement, and urgency, while cool colors like blue, green, and purple tend to evoke calmness, relaxation, or sometimes sadness. Although cultural background and personal experiences can influence individual responses to color, many of these associations are widely recognized. The theory is frequently applied in real-world settings—from hospital walls painted blue to reduce anxiety, to retail environments using red to stimulate a ction. While the scientific basis for color-emotion links is still evolving, color psychology remains a practical and intuitive framework for designing mood-based systems. This research directly informs the foundation o f ‘Wear Your Mood’, where users are recommended outfits and color palettes based on emotional input

https://en.wikipedia.org/wiki/Plutchik%27s_wheel_of_emotions



Wikipedia
The Free Encyclopedia

Search Wikipedia | Search

Donate | Create account | Log in

Photo contest Wiki Loves Earth:
An international photography competition where you can showcase England's unique natural environment and potentially win a prize.

Emotion classification

6 languages

Contents hide

(Top)

- Emotions as discrete categories
  - Simplicity debate
- Dimensional models of emotion
  - Circumplex model
  - Vector model
  - Positive activation – negative activation (P.A.N.A.) model
  - Plutchik's model
  - PAD emotional state model
- Criticisms
  - Cultural considerations
- Lists of emotions
  - Basic emotions
  - Contrasting basic emotions
  - Emotion dynamics
  - HUMAINE's proposal for EARL
  - Parrott's emotions by

Article | Talk | Read | Edit | View history | Tools

From Wikipedia, the free encyclopedia

(Redirected from Plutchik's wheel of emotions)

Emotion classification, the means by which one may distinguish or contrast one emotion from another, is a contested issue in emotion research and in affective science. Researchers have approached the classification of emotions from one of two fundamental viewpoints:[citation needed]

1. that emotions are discrete and fundamentally different constructs
2. that emotions can be characterized on a dimensional basis in groupings

Emotions as discrete categories [edit]

In discrete emotion theory, all humans are thought to have an innate set of basic emotions that are cross-culturally recognizable. These basic emotions are described as "discrete" because they are believed to be distinguishable by an individual's facial expression and biological processes.[1] Theorists have conducted studies to determine which emotions are basic. A popular example is Paul Ekman and his colleagues' cross-cultural study of 1992, in which they

Colored intaglio prints by Charles Le Brun and J. Pass depicting the facial expressions of sixteen emotions

Appearance hide

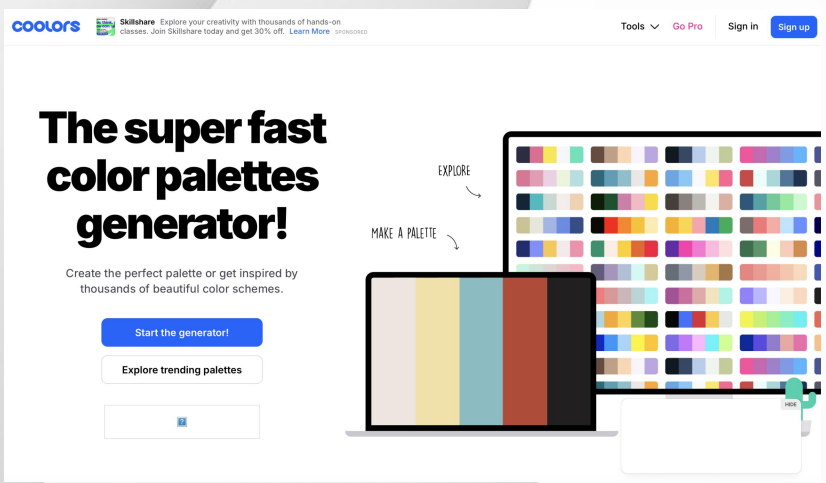
Text
- Small
- Standard
- Large

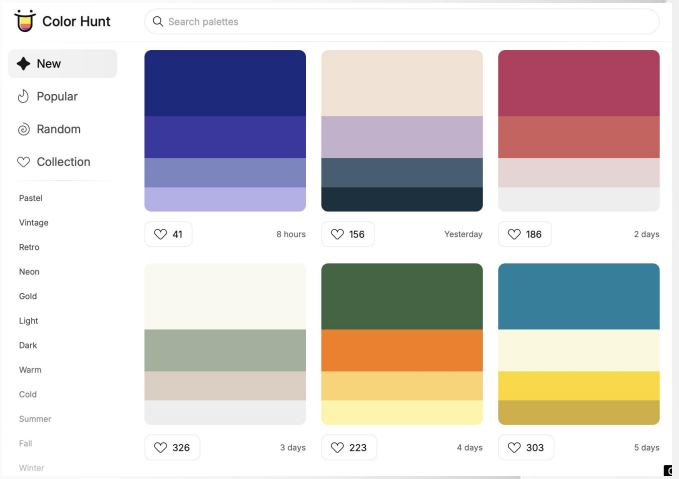
Width
- Standard
- Wide

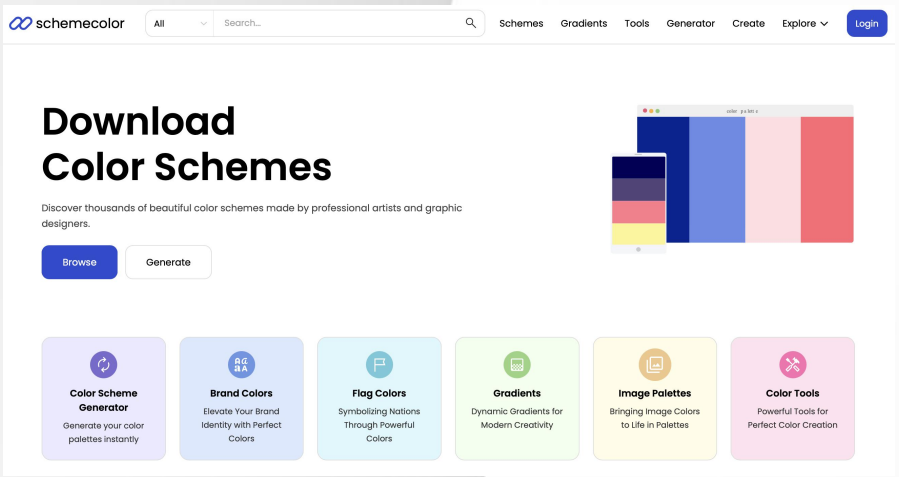
Color (beta)
- Automatic
- Light
- Dark

Psychological models like Plutchik's Wheel link emotions to colors—yellow for joy, red for anger, blue for sadness, and green for fear. Warm colors often represent high-energy emotions, while cool tones suggest calm or introspection. These associations help visualize emotional states and are widely used in mood-based design.

Data Source







The system primarily sources color palette data from the following professional color websites:

- Coolors.co — Offers a wide range of designer-created color palettes and supports keyword-based search.
- ColorHunt.co — A popular color scheme platform featuring daily curated palettes.
- SchemeColor.com — A professional site with a broad variety of categorized color schemes.
- ColorHexa.com — Provides detailed color information along with related palette suggestions.
- Design-Seeds.com — Extracts color inspiration from nature and everyday life scenes.
- ColorCombos.com — Focuses on commonly used color combinations in commercial design.

In addition, the system uses Google search to retrieve more relevant resources, ensuring a diverse and comprehensive data source.

Scraping Methods

Direct Crawling from Target Websites:

- Construct specific URLs based on emotional keywords (e.g., coolors.co/palettes/search/happy)
- Use random User-Agent headers to simulate browser access
- Parse HTML pages to extract color palette data

Google Search Enhancement:

- Build search queries including the target website domain (e.g., happy color palette site:coolors.co)
- Parse search results to obtain relevant links
- Perform secondary crawling on the retrieved URLs

This hybrid strategy leverages structured data from professional color websites while using search engines to expand data coverage, enhancing both the scope and accuracy of the system.

1. Set User-Agent

Code lines: 13—18
Screenshot suggestion: Show the USER_AGENTS list and get_random_user_agent() function
Description:
Simulates requests from real browsers to avoid being blocked by target websites.

```
12  # User Agent List
13  USER_AGENTS = [
14      "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
15      "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
16      "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0",
17      "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.1 Safari/605.1.15"
18  ]
```

2. Extract Colors from Text

Code lines: 20—31
Screenshot suggestion: Show the entire extract_colors_from_text() function
Description:
Uses regular expressions to extract hex codes, rgba values, and common color names from text.

```
20  def get_random_user_agent():
21      return random.choice(USER_AGENTS)
22
23  def extract_colors_from_text(text):
24      """Extract color code from text"""
25      # Match hexadecimal color code
26      hex_colors = re.findall(r'#(?:[0-9a-fA-F]{3}){1,2}\b', text)
27      # Match rgb/rgba color
28      rgb_colors = re.findall(r'rgba?\(\s*\d+\s*,\s*\d+\s*,\s*\d+\s*(?:,\s*[\d.]+\s*)?\)', text)
29      # Match color names
30      color_names = re.findall(r'\b(?:red|green|blue|yellow|orange|purple|pink|brown|black|white|gray|grey|cyan|magenta|violet|gold|silver|lavender|lime|teal|in
31
```

3. Extract Color Values from HTML Structure

Code lines: 33–74
Screenshot suggestion: Show the entire extract_color_palettes() function
Description:
Parses the HTML structure to locate elements that contain color data using multiple methods like style attributes, data-color, or inner text.

```
33
34  def extract_color_palettes(soup):
35      """Extract a color scheme from a page"""
36      palettes = []
37
38      # Find possible color scheme containers
39      palette_containers = soup.find_all(class_=re.compile(r'palette|color-group|swatches|colors|scheme|combination'))
40
41      for container in palette_containers:
42          # Extract color elements
43          color_elements = container.find_all(class_=re.compile(r'color|swatch|chip|sample'))
44
45          # If there is no specific class, try to find elements with background color
46          if not color_elements:
47              color_elements = container.find_all(style=re.compile(r'background-color|color'))
48
49          colors = []
50          for element in color_elements:
51              # Get color from style attribute
52              if 'style' in element.attrs:
53                  bg_match = re.search(r'background-color:\s*(.*?)[;]', element['style'])
54                  if bg_match:
55                      colors.append(bg_match.group(1).strip())
56                  else:
57                      color_match = re.search(r'color:\s*(.*?)[;]', element['style'])
58                      if color_match:
59                          colors.append(color_match.group(1).strip())
60
61              # Get color from data attribute
62              elif 'data-color' in element.attrs:
63                  colors.append(element['data-color'])
64
65              # Get color from text
66              else:
67                  text_colors = extract_colors_from_text(element.get_text())
68                  if text_colors:
69                      colors.extend(text_colors)
70
71          # If 3-8 colors are found, it is considered as an effective color scheme
72          if 3 <= len(colors) <= 8:
73              palettes.append(colors)
74
```

4. Search for Palette URLs Using Google

Code lines: 76–101
Screenshot suggestion: Show the entire search_google_for_palettes() function
Description:
Builds Google search queries using emotional keywords to find relevant color palette pages from known websites.

```
76
77  def search_google_for_palettes(keyword):
78      """Using Google to search for color schemes"""
79      try:
80          query = f"{keyword} color palette site:coolors.co OR site:colorhunt.co OR site:colordesigner.io OR site:schecolor.com"
81          url = f"https://www.google.com/search?q={quote(query)}"
82
83          headers = {
84              'User-Agent': get_random_user_agent(),
85              'Accept-Language': 'en-US,en;q=0.9'
86          }
87
88          response = requests.get(url, headers=headers, timeout=15)
89          response.raise_for_status()
90
91          soup = BeautifulSoup(response.text, 'html.parser')
92          links = []
93
94          # Extract the first 5 related links
95          for result in soup.select('.tF2Cxc')[:5]:
96              link = result.a['href']
97              if link.startswith('http'):
98                  links.append(link)
99
100         return links
101     except Exception as e:
```

5A. Build Target Website List

Code lines: 106—133
Screenshot suggestion: Show the sites list and the loop appending google_links in crawl_palette_sites()
Description:
Defines target URLs for direct crawling and dynamically adds Google search results for wider coverage.

```
105  def crawl_palette_sites(keyword):
106      """Crawling color scheme website"""
107      sites = [
108          {
109              'url': f'https://coolors.co/palettes/search/{keyword}',
110              'type': 'coolors'
111          },
112          {
113              'url': f'https://colorhunt.co/palettes/{keyword}',
114              'type': 'colorhunt'
115          },
116          {
117              'url': f'https://www.colorhexa.com/color-{keyword}',
118              'type': 'colorhexa'
119          },
120          {
121              'url': f'https://www.color-name.com/{keyword}-color',
122              'type': 'colorname'
123          },
124          {
125              'url': f'https://www.design-seeds.com/search/{keyword}/',
126              'type': 'designseeds'
127          },
128          {
129              'url': f'https://www.schemecolor.com/s/{keyword}',
130              'type': 'schemecolor'
131          },
132          {
133              'url': f'https://www.colorcombos.com/color-schemes.html?search={keyword}',
134              'type': 'colorcombos'
```

5B. Fetch and Parse Web Pages

Code lines: 148–161
Screenshot suggestion: Show the for site in sites: block with requests.get() and BeautifulSoup
Description:
Sends HTTP requests to each site and parses the HTML using BeautifulSoup to extract palette containers.

```
148     for site in sites:
149         try:
150             headers = {
151                 'User-Agent': get_random_user_agent(),
152                 'Accept-Language': 'en-US,en;q=0.9',
153                 'Referer': 'https://www.google.com/'
154             }
155 
156             print(f"Fetching: {site['url']}")
157             response = requests.get(site['url'], headers=headers, timeout=15)
158             response.raise_for_status()
159 
160             soup = BeautifulSoup(response.text, 'html.parser')
161 
```

6. Extract Palettes by Site Type

Code lines: 166—212
Screenshot suggestion: Show the different if site['type'] blocks (e.g., for coolors, colorhunt, colorhexa, etc.)
Description:
Applies different parsing rules depending on the site structure to extract hex color codes accurately.

```
162         # Different extraction strategies are adopted according to different website types
163         if site['type'] == 'coolors':
164             palette_divs = soup.select('.palette_container')
165             for div in palette_divs:
166                 colors = []
167                 color_divs = div.select('.palette_color')
168                 for color_div in color_divs:
169                     hex_code = color_div.get('data-hex')
170                     if hex_code:
171                         colors.append(f'#{hex_code}')
172                 if colors:
173                     all_palettes.append(colors)
174 
175         elif site['type'] == 'colorhunt':
176             palette_divs = soup.select('.palette')
177             for div in palette_divs:
178                 colors = []
179                 color_divs = div.select('.color')
180                 for color_div in color_divs:
181                     if 'style' in color_div.attrs:
182                         bg_color = re.search(r'background-color:\s*(.*?)[;]', color_div['style'])
183                         if bg_color:
184                             colors.append(bg_color.group(1).strip())
185                 if colors:
186                     all_palettes.append(colors)
187 
188         elif site['type'] == 'colorhexa':
189             color_table = soup.select('.color-table tbody tr')
190             if color_table:
191                 palette = []
192                 for row in color_table[:8]:  # Take up to 8 colors
193                     hex_code = row.select_one('td:nth-child(2)').get_text(strip=True)
194                     if hex_code:
195                         palette.append(f'#{hex_code}')
196                 if palette:
197                     all_palettes.append(palette)
198 
199         elif site['type'] == 'schemecolor':
200             palette_divs = soup.select('.palette-container')
201             for div in palette_divs:
202                 colors = []
203                 color_divs = div.select('.palette-color')
204                 for color_div in color_divs:
205                     hex_code = color_div.find(class_='hexcode').get_text(strip=True)
206                     if hex_code:
```

```
206                     if hex_code:
207                         colors.append(f'#{hex_code}')
208                 if colors:
209                     all_palettes.append(colors)
210 
211         # General extraction method
212         extracted = extract_color_palettes(soup)
213         all_palettes.extend(extracted)
214 
215         time.sleep(random.uniform(1, 3))  # random latency
216 
217     except Exception as e:
218         print(f"Error crawling {site['url']}: {str(e)}")
219         continue
```

Data Processing

Color Data Extraction Process

The system adopts a multi-layered approach to color extraction:

Page Structure Analysis:

- Identify common color palette containers (e.g., elements with classes like "palette" or "color-group")
- Extract HTML elements that contain background color or inline color styles

Color Value Extraction:

- Extract background-color or color values from the style attribute
- Retrieve color codes (e.g., data-hex) from data attributes
- Parse color names or hexadecimal codes from the element's text content

```
# Match hexadecimal color code
hex_colors = re.findall(r'#(?:[0-9a-fA-F]{3}){1,2}\b', text)
# Match rgb/rgba color
rgb_colors = re.findall(r'rgba?\(\s*\d+\s*,\s*\d+\s*,\s*\d+\s*(?:,\s*[\d.]+\s*)?\)', text)
# Match color names
color_names = re.findall(r'\b(?:red|green|blue|yellow|orange|purple|pink|brown|black|white|gray|grey|cyan|magenta|violet|gold|silver|lavender|lime|teal|in
```

Data Deduplication and Validation

To ensure data quality, the system applies the following processing steps to the collected color palettes:

Deduplication:

- Convert each color list into a tuple
- Use a set to remove duplicate entries

```
unique_palettes = []
seen = set()
for palette in palettes:
    palette_tuple = tuple(palette)
    if palette_tuple not in seen:
        seen.add(palette_tuple)
        unique_palettes.append(palette)
```

System Architecture

Backend Implementation

The system's backend is built using the Python Flask framework, with the following core modules:

1. Route Controller:
   - / — Returns the front-end HTML page
   - /get_palettes — Handles requests for color palette generation

2. Crawler Scheduler:
   - Manages crawling tasks across multiple websites
   - Controls request frequency and timeout settings
   - Handles exceptions during the crawling process

3. Data Processor:
   - Cleans raw data
   - Validates color formats
   - Removes duplicate palettes

4. API interface:
Receive front-end requests
Return JSON format data

```python
@app.route('/get_palettes', methods=['POST'])
def get_palettes():
    keyword = request.form.get('keyword', '').strip().lower()
    if not keyword:
        return jsonify({'error': 'Please enter a keyword'})
    try:
        palettes = crawl_palette_sites(keyword)
        if not palettes:
            return jsonify({'error': f'No color palettes found for "{keyword}"'})
        return jsonify({
            'keyword': keyword,
            'palettes': unique_palettes[:5]
        })
    except Exception as e:
        return jsonify({'error': str(e)})
```

API Integration

Integration with DeepSeek API

Considering the need for logical, expressive, and stylistically coherent fashion suggestions, I initially explored both ChatGPT and DeepSeek. After comparing factors such as performance and pricing, I ultimately chose the DeepSeek API to generate personalized fashion recommendations for users.

With the API integrated, the system can still provide outfit suggestions even when the input color values cannot be retrieved through web scraping.

1. Request construction

```
const response = await fetch(DEEPSEEK_API_URL, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${DEEPSEEK_API_KEY}`
    },
    body: JSON.stringify({
        model: 'deepseek-chat',
        messages: [{role: 'user', content: prompt}],
        temperature: 0.7,
        max_tokens: 1000
    })
});
```

2.Cue word design

```
let prompt;
if (colors.length > 0) {
    prompt = `Provide a detailed fashion advice based on the emotion "${emotion}"
    and these color palette: ${colorList}. Suggest what color to use for tops,
    what colors would work well for bottoms, and what accessories would complement
    this look. Also suggest suitable styles and materials...`;
} else {
    prompt = `Provide a detailed fashion advice based on the emotion "${emotion}".
    Suggest color combinations, styles, and materials that would best represent...`;
}
```

Output Design

Front-end technical implementation

1. Responsive design:
Using CSS Flexbox and Grid layouts
Adapting to different screen sizes

```css
@media (max-width: 768px) {
    .palettes {
        grid-template-columns: 1fr;
    }
}
```

2. Interactive function, color click to copy

```javascript
colorElement.addEventListener('click', () => {
    navigator.clipboard.writeText(color).then(() => {
        colorElement.textContent = 'Copied!';
        setTimeout(() => {
            colorElement.textContent = color;
        }, 1000);
    });
});
```

Technology Stack and Dependencies

The system is developed using Python 3.x as the programming language. The main dependencies are as follows:

- **beautifulsoup4==4.12.3**
- **requests==2.31.0**
- **flask==3.0.2**

All dependencies are managed through a requirements.txt file.

Key Technical Implementations

Web Parsing Technology

The system uses the BeautifulSoup4 library to perform multi-mode HTML parsing.

```
def extract_colors_palettes(soup):
    palettes = []
    for container in soup.select('.palette_container'):
        colors = [f"#{color['data-hex']}" for color in container.select('.palette_color')]
        if len(colors) >= 3:  # A valid palette should contain at least 3 colors
            palettes.append(colors)
    return palettes
```

Web Request Optimization
Anti-crawl strategy via Requests library:

```python
# List of possible User-Agent headers
USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)...",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)..."
]

def get_with_retry(url, max_retries=3):
    for attempt in range(max_retries):
        try:
            response = requests.get(
                url,
                headers={'User-Agent': random.choice(USER_AGENTS)},
                timeout=10
            )
            response.raise_for_status()
            return response
        except Exception as e:
            if attempt == max_retries - 1:
                raise
            time.sleep(2 ** attempt)
```

Final Display

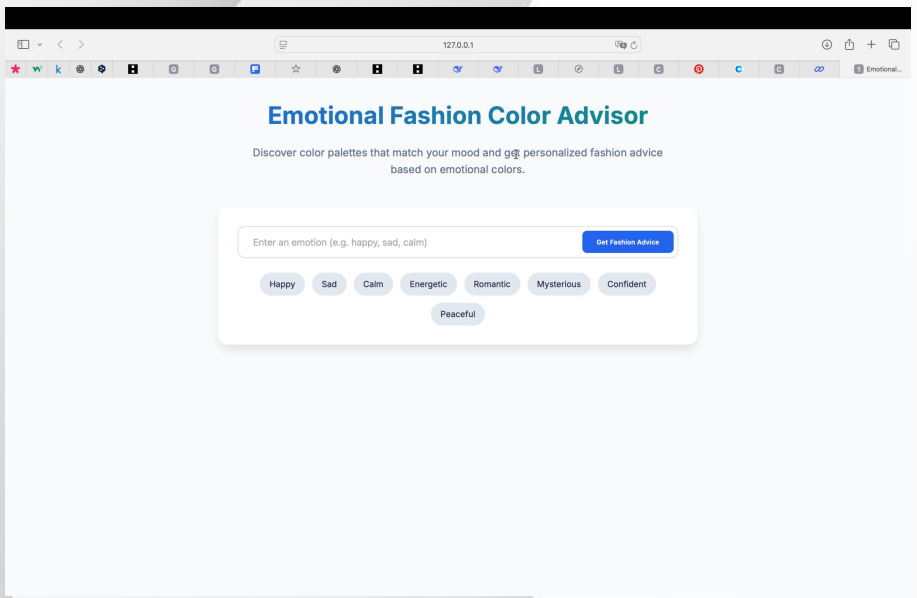


Figure 1. Emotion input interface with keyword buttons and text field.

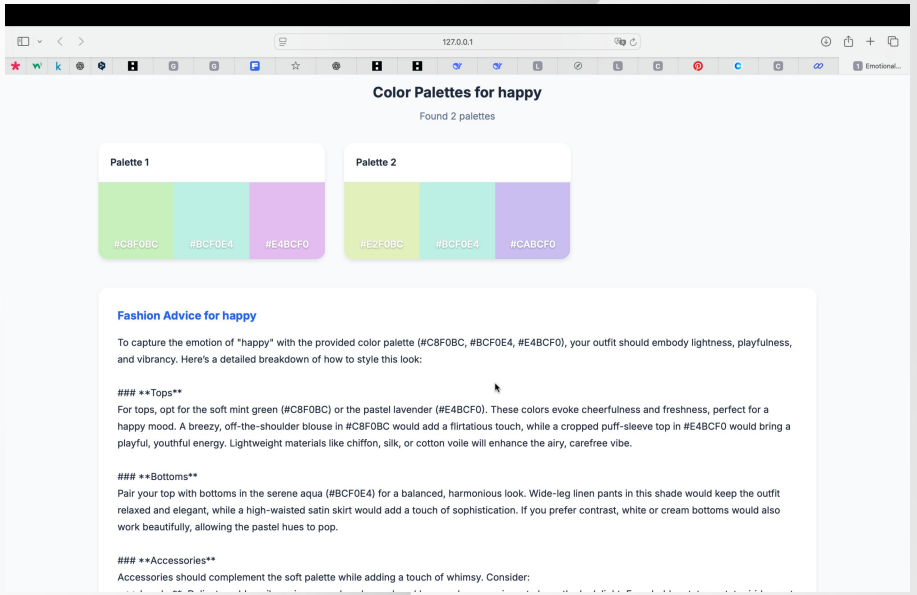


Figure 2. Display of matching color palettes and generated fashion advice.

The final interface of the system is clean and intuitive, guiding users from emotional input to outfit suggestions with ease.

On the homepage, users can either type in an emotion (e.g., happy, sad, calm) or select from preset keywords. Once submitted, the system retrieves matching color palettes from various online sources and displays them as color swatches.

Each palette includes its corresponding hex codes for clear visual reference. Below the palettes, personalized fashion advice is generated via the DeepSeek API, offering styling suggestions for tops, bottoms, and accessories based on the selected emotion and color mood.

Since the system fetches data in real-time through web scraping, the results may take a few seconds to load. This ensures the color palettes and recommendations are accurate and up-to-date.

This final display not only provides practical fashion guidance, but also effectively translates abstract emotions into visual and color-driven outfit solutions.

Reflection

This project involved several technical and strategic challenges that led to valuable learning outcomes.

At first, I planned to crawl product images from fashion websites like ASOS and Pinterest based on emotional keywords. However, most sites had anti-scraping measures, preventing access to valid image data. To overcome this, I shifted to collecting color palettes from professional color websites and enhanced the process with Google search queries. This hybrid approach improved both coverage and accuracy.

The system was built using Flask for backend logic and DeepSeek API for generating outfit suggestions. The API ensured that even without color data, the system could still provide results.

Overall, this project improved my skills in data collection, system design, and API integration. It also laid a solid foundation for future development, such as adding image generation or emotion recognition features.

References

1. Coolors. Color palette generator & inspiration. Retrieved from https://coolors.co
2. Color Hunt. Curated color palettes for designers and artists. Retrieved from https://colorhunt.co
3. SchemeColor. Color schemes and combinations. Retrieved from https://www.schemecolor.com
4. ColorHexa. Color encyclopedia and palette generator. Retrieved from https://www.colorhexa.com
5. Design Seeds. Color inspiration from nature and lifestyle. Retrieved from https://www.design-seeds.com
6. ColorCombos. Commercial color combinations and design tools. Retrieved from https://www.colorcombos.com
7. Google Search. Used to expand data coverage through keyword-based queries. Retrieved from https://www.google.com
8. DeepSeek API. AI-powered text generation for fashion recommendations. Retrieved from https://deepseek.com
9. BeautifulSoup4. Python library for parsing HTML and XML. Retrieved from https://www.crummy.com/software/BeautifulSoup/
10. Requests. HTTP library for Python. Retrieved from https://requests.readthedocs.io/
11. Flask. Web development framework for Python. Retrieved from https://flask.palletsprojects.com/
12. https://www.verywellmind.com/color-psychology-2795824
13. https://en.wikipedia.org/wiki/Plutchik%27s_wheel_of_emotions