# **Content Style Grading Feature Documentation**

## **1. Introduction**

The Content Style Grading feature analyzes and evaluates the consistency, typography, and font styles used across web pages. By extracting information from URLs, it provides valuable insights to enhance content quality. Here’s why it matters:

* **Consistency**: Ensures uniformity in design elements, enhancing user experience.
* **Typography**: Influences readability, aesthetics, and emotional impact.
* **Font Styles**: Conveys brand personality and professionalism.

## **2. How It Works**

1. **URL Extraction**: Collects content from input URL.
2. **Consistency Check**:
   * Compares design elements (colors, spacing) across pages.
   * Flags inconsistencies for further analysis.
3. **Typography Assessment**:
   * Identifies font families, sizes, and styles.
   * Consider readability and visual appeal.
4. **Font Style Grading**:
   * Scores font choices based on established guidelines available at different research papers and blogs.

## **3. Usage Guide**

1. Assessing a Website:
   * Input URL or upload HTML/CSS files.
   * Receive content style score and detailed report.
2. Interpreting Results:
   * Understand content style and recommendations.
   * Iterate based on feedback.

## **4. Work Distribution**

Timeline: 2 Weeks

* Backend & Algorithm : Animesh
* Frontend : Vidya
* Detailing & Designing : Ankita

## **5. Technical Elements**

## **6. References and Resources**

* [Classifying Typography Styles: Definitive Guide - 2024 (inkbotdesign.com)](https://inkbotdesign.com/typography-styles/)
  + Importance of typography
* [Font Psychology: New research & practical insights - Cognition Today](https://cognitiontoday.com/font-psychology-research-and-application/)
  + Different typography research insights.
* [Design Consistency Guide with 9 Best Practices (uxpin.com)](https://www.uxpin.com/studio/blog/guide-design-consistency-best-practices-ui-ux-designers/)
  + About design consistency. Can be included if possible to write the algo.
* [The Role Of Typography In Web Design: An In-Depth Guide - Clio Websites](https://cliowebsites.com/the-role-of-typography-in-web-design/)
  + Different typography types and their roles.
* [Principles of Typography in UI Design | by Bryson M. | Medium | UX Planet](https://uxplanet.org/principles-of-typography-in-ui-design-bc28f1f9666d)
  + Rules for proper and consistent typography and content.
* [Consistency and standards in UX/UI design: The key to success | by Akansha Tandon | Bootcamp (uxdesign.cc)](https://bootcamp.uxdesign.cc/consistency-and-standards-in-ux-ui-design-the-key-to-success-8392d5d56336)
  + Role and significance of consistency in web pages.

To verify alignment properties such as text-align, float, and flex, you can follow these steps:

### **1. Extract CSS Properties**

* **Use JavaScript** to extract and verify CSS properties of elements on a webpage.

### **2. Check text-align Property**

**JavaScript Example**:  
const element = document.querySelector('.your-element');

const textAlign = window.getComputedStyle(element).textAlign;

console.log(`Text Align: ${textAlign}`);

### **3. Check float Property**

**JavaScript Example**:  
const float = window.getComputedStyle(element).float;

console.log(`Float: ${float}`);

### **4. Check Flexbox Properties**

**JavaScript Example**:  
const display = window.getComputedStyle(element).display;

const justifyContent = window.getComputedStyle(element).justifyContent;

const alignItems = window.getComputedStyle(element).alignItems;

console.log(`Display: ${display}`);

console.log(`Justify Content: ${justifyContent}`);

console.log(`Align Items: ${alignItems}`);

### **5. Automate the Process**

**Create a Function** to automate the verification of alignment properties for multiple elements.  
function checkAlignment(selector) {

const element = document.querySelector(selector);

const styles = window.getComputedStyle(element);

return {

textAlign: styles.textAlign,

float: styles.float,

display: styles.display,

justifyContent: styles.justifyContent,

alignItems: styles.alignItems

};

}

const alignment = checkAlignment('.your-element');

console.log(alignment);

### **6. Integrate with Your Analysis Tool**

* **Incorporate these checks** into your content style grading feature to automatically verify alignment properties across different elements and pages.

By following these steps, you can effectively verify and analyze the alignment properties of elements on a webpage.

To decide which elements should be checked for alignment properties, you can follow these steps:

### **1. Identify Key Elements**

* **Focus on Common Elements**:
  + **Headers**: <h1>, <h2>, etc.
  + **Paragraphs**: <p>
  + **Images**: <img>
  + **Containers**: <div>, <section>, <article>
  + **Lists**: <ul>, <ol>, <li>
  + **Navigation**: <nav>, <ul>, <li>

### **2. Define Criteria for Selection**

* **Visual Importance**: Elements that are visually significant and contribute to the overall layout.
* **Frequency of Use**: Elements that appear frequently across different pages.
* **Design Guidelines**: Elements specified in design guidelines or style guides.

### **3. Automate Element Selection**

**Use JavaScript** to automatically select and analyze these elements.  
const elementsToCheck = [

'h1', 'h2', 'h3', 'p', 'img', 'div', 'section', 'article', 'ul', 'ol', 'li', 'nav'

];

elementsToCheck.forEach(selector => {

document.querySelectorAll(selector).forEach(element => {

const styles = window.getComputedStyle(element);

console.log(`Element: ${selector}, Text Align: ${styles.textAlign}, Float: ${styles.float}, Display: ${styles.display}`);

});

});

### **4. Customizable Selection**

**Allow Customization**: Provide options for users to specify additional elements or exclude certain elements.  
function checkAlignment(customSelectors = []) {

const defaultSelectors = [

'h1', 'h2', 'h3', 'p', 'img', 'div', 'section', 'article', 'ul', 'ol', 'li', 'nav'

];

const selectors = [...new Set([...defaultSelectors, ...customSelectors])];

selectors.forEach(selector => {

document.querySelectorAll(selector).forEach(element => {

const styles = window.getComputedStyle(element);

console.log(`Element: ${selector}, Text Align: ${styles.textAlign}, Float: ${styles.float}, Display: ${styles.display}`);

});

});

}

// Example usage with custom selectors

checkAlignment(['footer', 'header']);

### **5. Analyze and Report**

**Generate Reports**: Summarize the alignment properties of the selected elements and highlight any inconsistencies.  
function generateReport() {

const report = [];

elementsToCheck.forEach(selector => {

document.querySelectorAll(selector).forEach(element => {

const styles = window.getComputedStyle(element);

report.push({

element: selector,

textAlign: styles.textAlign,

float: styles.float,

display: styles.display

});

});

});

return report;

}

const alignmentReport = generateReport();

console.log(alignmentReport);

By following these steps, you can systematically decide which elements to check for alignment properties and automate the process to ensure consistency across your web pages.  
  
  
Based on the available blogs and research, here are some inconsistency rules for alignment properties in web design:

### **1. Consistency in Text Alignment**

* **Rule**: All similar elements (e.g., headings, paragraphs) should have the same text-align property.
* **Example**: If all <h1> elements are left-aligned, ensure that all other <h1> elements are also left-aligned.
* [**Source**: Consistent text alignment helps maintain a cohesive design and improves readability1](https://www.uxpin.com/studio/blog/alignment-in-design-making-text-and-visuals-more-appealing/).

### **2. Uniform Use of Float**

* **Rule**: Use the float property consistently for similar elements.
* **Example**: If images within a certain section are floated to the right, ensure all images in that section follow the same pattern.
* [**Source**: Inconsistent use of float can disrupt the visual flow and create a confusing user experience1](https://www.uxpin.com/studio/blog/alignment-in-design-making-text-and-visuals-more-appealing/).

### **3. Consistent Flexbox Properties**

* **Rule**: Apply display: flex, justify-content, and align-items uniformly across similar containers.
* **Example**: If a container uses justify-content: center and align-items: center, ensure all similar containers use the same properties.
* [**Source**: Consistent use of flexbox properties ensures a harmonious and balanced layout1](https://www.uxpin.com/studio/blog/alignment-in-design-making-text-and-visuals-more-appealing/).

### **4. Alignment in Grid Systems**

* **Rule**: Maintain consistent alignment within grid systems.
* **Example**: Ensure that all grid items align properly within their respective columns and rows.
* [**Source**: Proper alignment in grid systems enhances the overall structure and readability of the design2](https://balsamiq.com/learn/articles/visual-hierarchy-and-alignment/).

### **5. Visual Hierarchy and Alignment**

* **Rule**: Use alignment to reinforce the visual hierarchy.
* **Example**: Headings should be aligned consistently to guide the user’s eye through the content.
* [**Source**: Proper alignment helps create a clear visual hierarchy, making the interface easier to scan and understand2](https://balsamiq.com/learn/articles/visual-hierarchy-and-alignment/).

### **6. Avoiding Mismatched Alignments**

* **Rule**: Avoid using different alignment properties for similar elements within the same context.
* **Example**: Do not mix left-aligned and center-aligned headings within the same section.
* [**Source**: Mismatched alignments can create a disjointed and confusing user experience1](https://www.uxpin.com/studio/blog/alignment-in-design-making-text-and-visuals-more-appealing/).

### **Implementation Steps with Code**

#### **1. Extract and Compare Properties**

**JavaScript Example**:  
function checkUniformAlignment(selector) {

const elements = document.querySelectorAll(selector);

const alignmentValues = new Set();

elements.forEach(element => {

const styles = window.getComputedStyle(element);

alignmentValues.add(styles.textAlign);

});

return alignmentValues.size === 1; // Returns true if all alignments are the same

}

const selectors = ['h1', 'h2', 'h3', 'p'];

const consistencyReport = {};

selectors.forEach(selector => {

consistencyReport[selector] = checkUniformAlignment(selector);

});

console.log(consistencyReport);

#### **2. Generate Inconsistency Report**

**JavaScript Example**:  
function findInconsistencies(selector) {

const elements = document.querySelectorAll(selector);

const alignmentValues = {};

elements.forEach(element => {

const styles = window.getComputedStyle(element);

const alignment = styles.textAlign;

if (!alignmentValues[alignment]) {

alignmentValues[alignment] = [];

}

alignmentValues[alignment].push(element);

});

return alignmentValues;

}

const inconsistencyDetails = {};

selectors.forEach(selector => {

const inconsistencies = findInconsistencies(selector);

if (Object.keys(inconsistencies).length > 1) {

inconsistencyDetails[selector] = inconsistencies;

}

});

console.log(inconsistencyDetails);

By following these rules and implementing the provided code, you can ensure alignment consistency across your web design, enhancing the overall user experience.