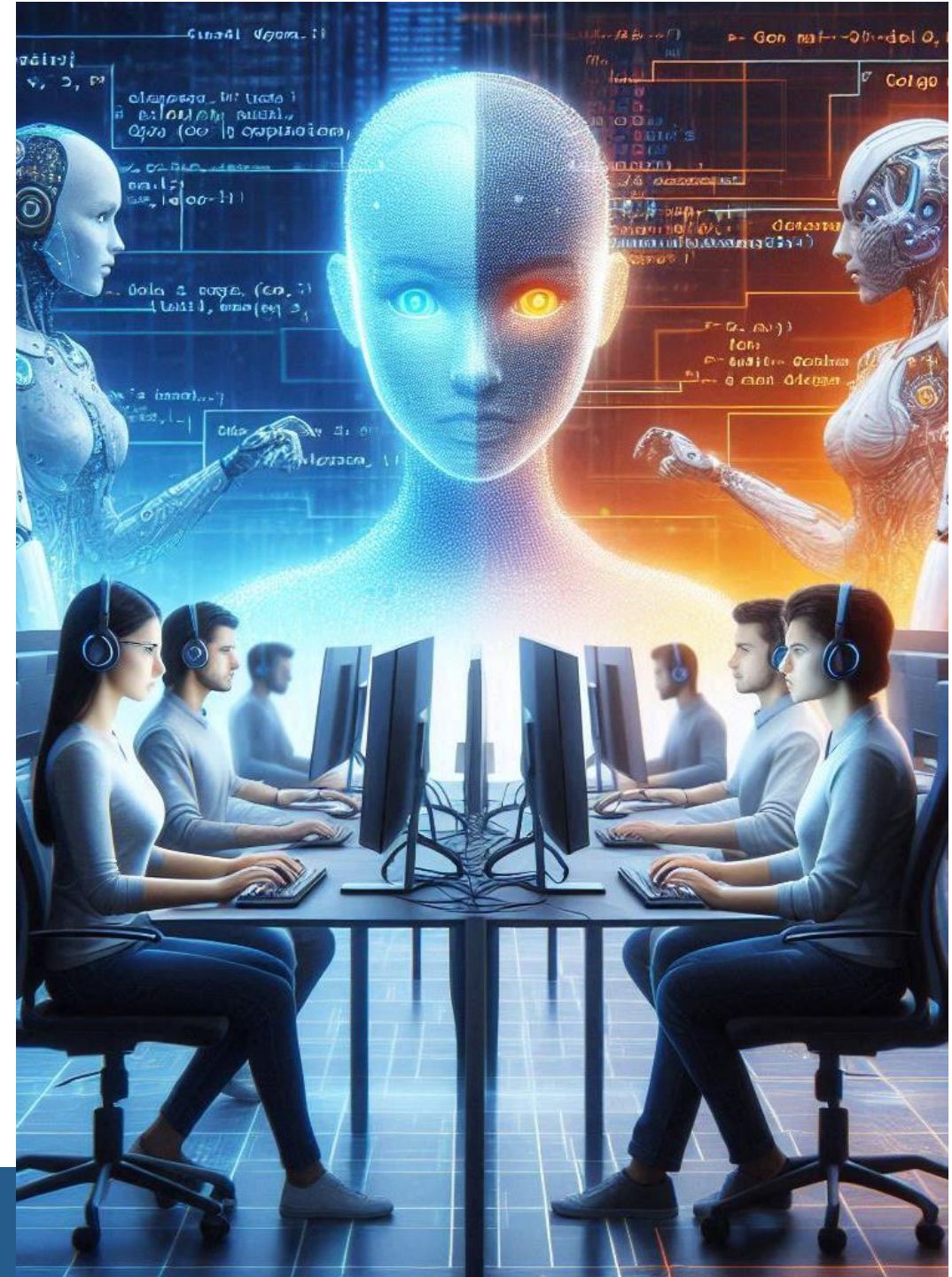


# Using AI for metadata and data cleaning

Alternative techniques



[info@contourcomponents.com](mailto:info@contourcomponents.com)



[contourcomponents.com](http://contourcomponents.com)

# Cleaning HTML documents

## Cleaning goals

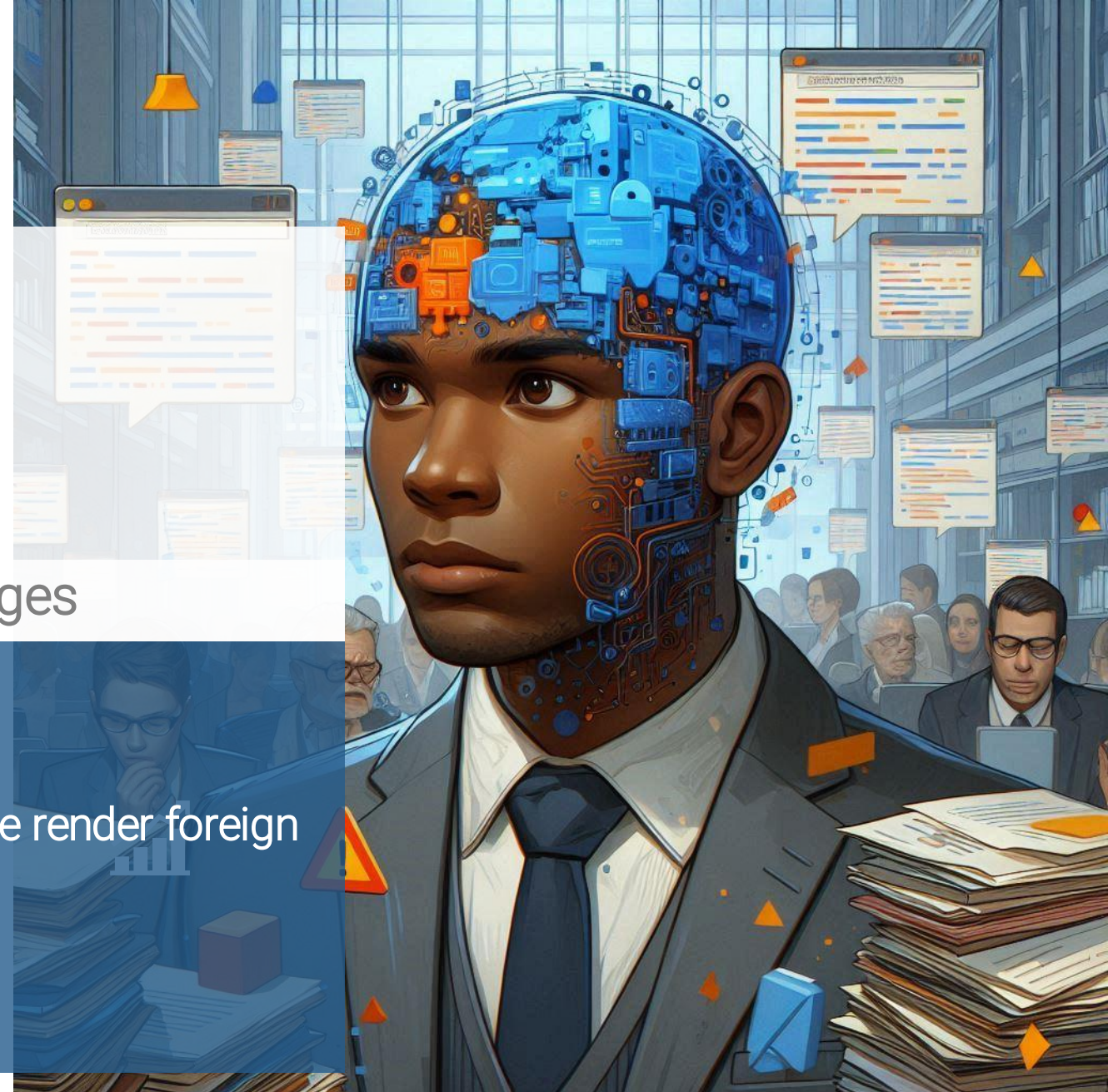
- ❑ Big Data preprocessing (i.e. millions of web pages) for upload to a database or other purposes
- ❑ Convert legacy archives of HTML documents to texts for upload to a database and uniform visualization



# Low quality of HTML documents

Issues of archives of legacy HTML pages

- ❑ Data volume has increased tenfold
- ❑ Pages of apps can be damaged while render foreign pages
- ❑ Hard to analyze, hard to localize



# An example of a low quality HTML document

```
<html><head><meta name="qrichtext" content="1" /><style
type="text/css">
p, li { white-space: pre-wrap; }
</style></head><body style=" font-family:"Roboto"; font-
size:8pt; font-weight:400; font-style:normal;">
<p align="center" style=" margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-
indent:0px;"><span style=" font-
size:18pt;">Prototype</span></p></body></html>
```



The document contains of many unnecessary tags and only one meaningful word:

“Prototype”



# Strait solution

Prompt:

“Please convert this HTML document in plain text”

```
<html><head><meta name="qrichtext" content="1" /><style type="text/css"> p, li { white-space: pre-wrap; } </style></head><body style=" font-family:'Roboto'; font-size:8pt; font-weight:400; font-style:normal;"> <p align="center" style=" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;"><span style=" font-size:18pt;">Prototype</span></p></body></html>
```



Plain text:  
“Prototype”



# Alternative solution

Prompt:

“Please create a function that converts this type of HTML in text”

```
<html><head><meta name="richtext"
content="1" /><style type="text/css">
p, li { white-space: pre-wrap; }
</style></head><body style=" font-
family: 'Roboto'; font-size: 8pt; font-
weight: 400; font-style: normal;">
<p align="center" style=" margin-top: 0px;
margin-bottom: 0px; margin-left: 0px; margin-
right: 0px; -qt-block-indent: 0; text-
indent: 0px;"><span style=" font-
size: 18pt;">Prototype</span></p></body></ht
ml>
```



Source code of the function



# Executing the AI created function

Parameter: HTML document

```
<html><head><meta name="richtext" content="1" /><style type="text/css"> p, li { white-space: pre-wrap; } </style></head><body style=" font-family:'Roboto'; font-size:8pt; font-weight:400; font-style:normal;"> <p align="center" style=" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;"><span style=" font-size:18pt;">Prototype</span></p></body></html>
```



Function



Result

Plain text:  
“Prototype”



# Source code of AI created function

```
DROP FUNCTION IF EXISTS tools.strip_html;
CREATE OR REPLACE FUNCTION tools.strip_html
(
  html_text TEXT,
  save_structure BOOLEAN DEFAULT true, -- save returns in Ansi text, ignored for HTML
  destination_format TEXT DEFAULT 'text' -- 'text' or 'html'
)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
DECLARE
  clean_text TEXT;
  destination_formats TEXT[] := {'text', 'html'};
BEGIN

  destination_format := TRIM(LOWER(destination_format));

  IF NOT destination_format = ANY (destination_formats) THEN
    RAISE EXCEPTION 'Unknown destination format: %', destination_format;
  END IF;

  IF destination_format = 'html' THEN
    save_structure := true;
  END IF;

  -- Remove conditional comments and their content
  clean_text := regexp_replace(html_text, '<!--\[if[^\>]*?>.*?!\[endif\]\-->', '', 'gi');

  -- Remove XML-like tags and their content
  clean_text := regexp_replace(clean_text, '<xml[^\>]*?>.*?</xml>', '', 'gi');

  -- Remove specific tags like <o:p> and their content
  clean_text := regexp_replace(clean_text, '<o:p[^\>]*?>.*?</o:p>', '', 'gi');

  -- Remove <style> and <script> tags and their content
  clean_text := regexp_replace(clean_text, '<style[^\>]*?>.*?</style>', '', 'gi');
  clean_text := regexp_replace(clean_text, '<script[^\>]*?>.*?</script>', '', 'gi');
```

Non-human programming style

```
-- Remove <head> and <meta> tags and their content
clean_text := regexp_replace(clean_text, '<head[^\>]*?>.*?</head>', '', 'gi');
clean_text := regexp_replace(clean_text, '<meta[^\>]*?>', '', 'gi');

-- If any { ... } block appears, it's most likely CSS, remove cautiously
clean_text := regexp_replace(clean_text, '{[^\}]*}', '');

-- Replace HTML entities (like &nbsp;)
clean_text := regexp_replace(clean_text, '&nbsp;', ' ');

-- Handle the rest based on the desired output format
IF destination_format = 'text' THEN
  IF save_structure THEN
    -- Add a newline after </o:p></font></p> and </div>
    clean_text := regexp_replace(clean_text, '</o:p></font></p>', '\n');

    -- Replace <p> and <br> tags with newline for structure preservation
    clean_text := regexp_replace(clean_text, '</p[^\>]*>', E'\n', 'gi');
    clean_text := regexp_replace(clean_text, '<br[^\>]*>', E'\n', 'gi');

    -- Replace <div> and </div> tags with newline for structure preservation
    clean_text := regexp_replace(clean_text, '</div[^\>]*>', E'\n', 'gi');

    -- Remove all other HTML tags while preserving the inner text content
    clean_text := regexp_replace(clean_text, '<[^\>]+>', '', 'g');

    -- Adjust the regex to collapse multiple newlines into one newline, not two
    clean_text := regexp_replace(clean_text, '\n\s*{2,}', E'\n', 'g');

    -- Trim leading and trailing whitespace and newlines
    clean_text := trim(both E'\n' FROM clean_text);
  ELSE
    -- For ANSI format without structure, replace </p> and </b> with a space
    clean_text := regexp_replace(clean_text, '</(p|b)>', ' ', 'gi'); -- Replace </p> and </b> with a space
    clean_text := regexp_replace(clean_text, '<[^\>]+>', '', 'g'); -- Remove all other HTML tags
    clean_text := regexp_replace(clean_text, '\s+', ' ', 'g'); -- Collapse multiple spaces
    clean_text := trim(both ' ' FROM clean_text); -- Trim leading and trailing spaces
  END IF;
ELSE
  -- Remove <pre> tags and their content
  clean_text := regexp_replace(clean_text, '<pre[^\>]*?>.*?</pre>', '', 'gi');
```

Comment for every step





# Executing of AI created function



Executing the function

```
99
100 SELECT * FROM tools.strip_html
101 (
102   '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
103   <html><head><meta name="qrichtext" content="1" /><style type="text/css">
104   p, li { white-space: pre-wrap; }
105   </style></head><body style=" font-family:'Roboto'; font-size:8pt; font-weight:400; font-style:normal;">
106   <p align="center" style=" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;"><
107   true,
108   'text'
109   );
110
111 SELECT * FROM
```

Data Output Messages Notifications

strip_html	text
1	Prototype

Result: plain text  
"Prototype"



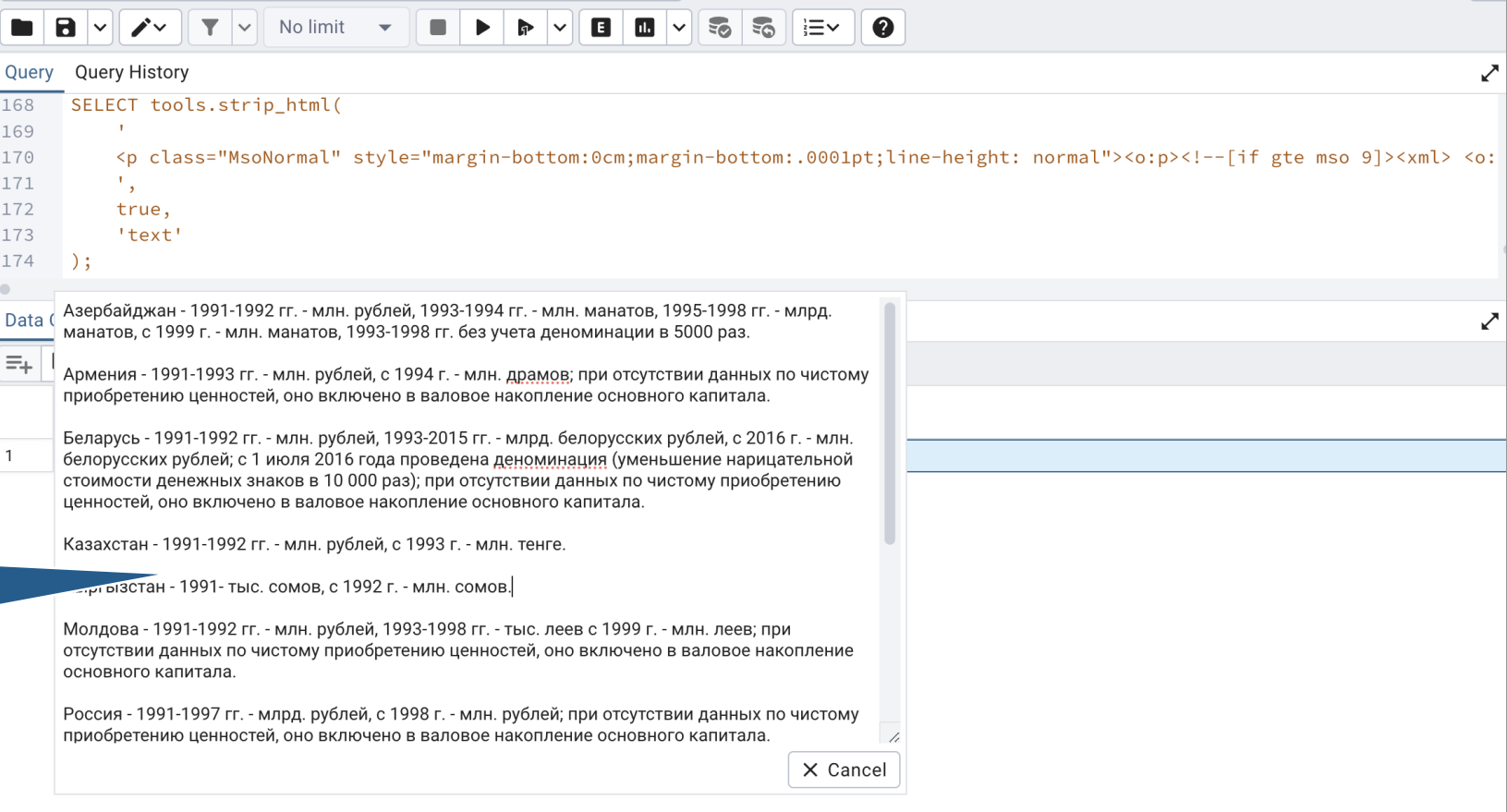
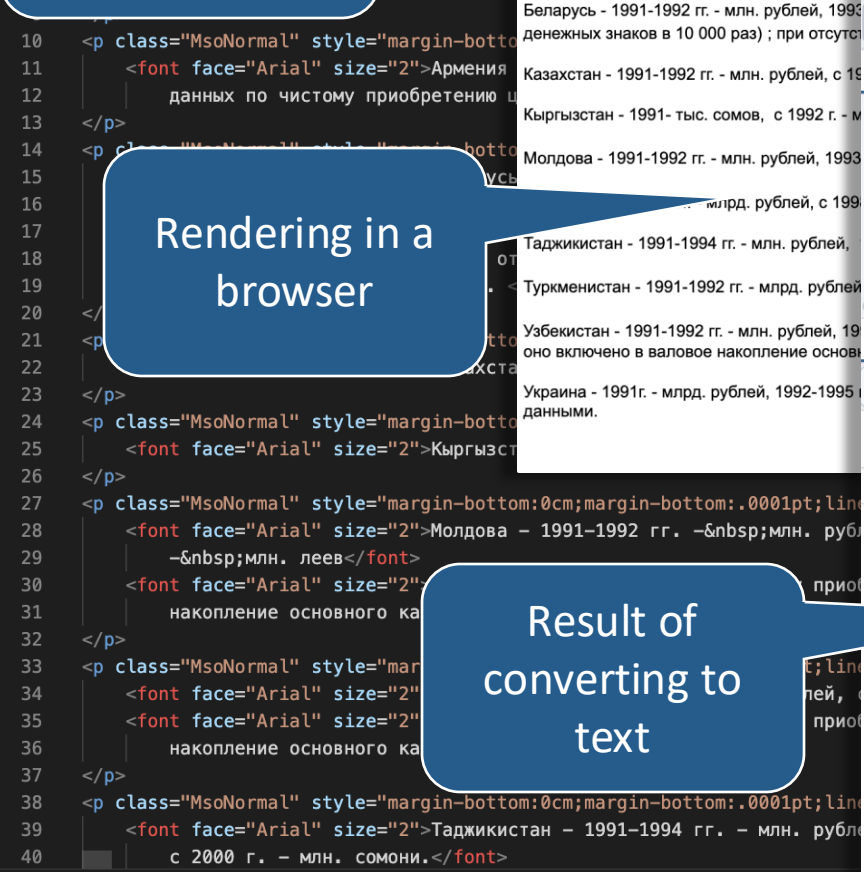
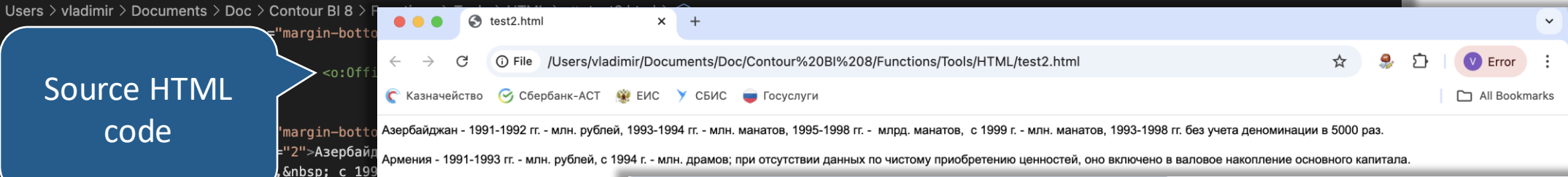


# Another example

Source HTML code

Rendering in a browser

Result of converting to text



# Iterative development

## Prompt

1. Create code for cleaning HTML
2. Mimic HTML structure – line breaks, bullets, and so on
3. Remove extra spaces
4. Add spaces after punctuation marks
5. Capitalize the first letters of sentences
6. ...

Artificial intelligence



Source code of the function



Execution of function

Result

- ❑ The intellectual capabilities of a person who sets the task are not unlimited
- ❑ Step-by-step improvement and development of algorithms is simpler
- ❑ The speed of generating a new version is crucial

Итерации



# AI vs a programmer

## Programmer

- 2 days per a version
- Increasing code complexity slows down development

3 iteration per week



## AI

- 2 seconds per a version
- Increasing code complexity does not affect development

30 iteration per hour



# Documenting a function

Prompt:  
“Please describe Please describe the algorithms for converting HTML to text”



Detailed  
documentation

The function `tools.strip_html` aims to convert HTML input into clean text or HTML, while optionally preserving certain structural elements like newlines. Here's a breakdown of the steps it takes and the algorithms used:

- 1. Input Validation and Preparation:**
  - The function accepts parameters HTML text (input HTML), save structure (whether to preserve formatting like line breaks), and destination format (either 'text' or 'html').
  - The destination format is checked for validity, and if it's not 'text' or 'html', an exception is raised.
  - If the destination format is 'html', save structure is automatically set to true.
- 2. Removing Conditional Comments:**
  - `regex_replace` is used to find and remove HTML conditional comments, such as `<!--[if ...]-->...<![endif]-->`. This is done using regular expressions that match the start and end of these blocks, ensuring they are deleted from the content.
- 3. Removing Specific XML-like Tags:**
  - Similar regular expressions remove `<xml>...</xml>` blocks and other custom HTML/XML tags like `<o:p>...</o:p>`, which are often remnants of rich text editors like Microsoft Word.
- 4. Removing Script, Style, Meta, and Head Tags:**
  - `<style>...</style>`, `<script>...</script>`, `<head>...</head>`, and `<meta>` tags (and their content, if applicable) are removed since they are not needed for plain text.
  - These elements are stripped because they usually contain non-visible content like scripts or metadata, not contributing to the document's main textual content.
- 5. Handling CSS Blocks:**
  - Any CSS blocks (indicated by curly braces `{...}`) are removed. This is a cautious approach to avoid leaving behind embedded CSS styling, which is irrelevant in text output.
- 6. Replacing HTML Entities:**
  - Some common HTML entities like `&nbsp;`, `&lt;`, `&gt;`, `&amp;`, `&quot;`, and `&apos;` are replaced with their corresponding characters (space, `<`, `>`, `&`, `"`, and `'`). This ensures the output is readable without HTML encoding artifacts.
- 7. Structural Preservation (Optional):**
  - If `save structure` is true, some key HTML structural elements like `<p>`, `<br>`, and `<div>` are replaced with newlines to maintain the content's formatting. This ensures that text broken across multiple lines in the HTML stays visually formatted in the plain text version.
  - Multiple newlines are collapsed into a single newline to avoid unnecessary whitespace.
  - If `save structure` is false, structural tags like `</p>` and `<br>` are replaced with a single space instead, resulting in a more compact representation without visible structure.

...



# Cleaning statistical metadata

## Errors in code lists

- Missed codes
- Missed values
- Wrong values
- Duplicates
- Wrong data types
- ...



# Straight solution

Prompt:

“Please find the errors in the code list and briefly describe them in the third column”

Code	Value
1	
2	Fish
	Meat
4	Male
5	Vegetables
01/01/2022	
7	\$123



Here's a table with the errors identified and described:

Code	Value	Error Description
1	(empty)	Missing value
2	Fish	No error
(empty)	Meat	Missing code
4	Male	No error
5	Juice	No error
01/01/2022	Bread	Invalid code format (date instead of a number)
7	\$123	Invalid format (contains special character "\$")

# Alternative solution

Prompt:

“Please find the errors in the code list and briefly describe them in the third column”

Code	Value	
1		
2	Fish	
	Meat	
4	Male	
5	Vegetables	
01/01/2022		
7	\$123	



To create a PL/pgSQL function that receives a code list, identifies errors, and returns a table with error descriptions, we can design the function to accept an input table or a recordset and perform validation checks based on conditions such as missing values, invalid formats, or special characters.

Here's an example of such a function:

Function Definition:

```
sql Copy code
CREATE OR REPLACE FUNCTION find_code_list_errors(code_list TEXT[])
RETURNS TABLE (code TEXT, value TEXT, error_description TEXT) AS
$$
DECLARE
    code_value RECORD;
BEGIN
    -- Loop through the provided code list (array of records)
    FOREACH code_value IN ARRAY code_list
    LOOP
        RETURN QUERY
        SELECT
            code_value.code,
            code_value.value,
            CASE
                -- Check if the code is NULL or empty
                WHEN code_value.code IS NULL OR code_value.code = '' THEN
                    'Invalid code'
                -- Check if the value is NULL or empty
                WHEN code_value.value IS NULL OR code_value.value = '' THEN
                    'Invalid value'
                -- Check if the code contains non-numeric characters
                WHEN code_value.code !~ '^[0-9]+$' THEN 'Invalid code'
                -- Check if the code contains special characters or invalid characters
                WHEN code_value.code ~ '[^0-9]' THEN 'Invalid character'
                -- No errors
                ELSE 'No error'
            END AS error_description;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```





# Executing of AI created function

Code	Value
1	
2	Fish
	Meat
4	Male
5	Vegetable
01/01/2022	
7	\$123



Code	Value	Error Description
1	(empty)	Missing value
2	Fish	No error
(empty)	Meat	Missing code
4	Male	No error
5	Juice	No error
01/01/2022	Bread	Invalid code format (date instead of a number)
7	\$123	Invalid format (contains special character "\$")

## Benefits

- Free: We do not use ChatGPT to process data
- Speed: We use our own servers and databases
- Convenience: The created function is built into the application



# Benefits of an alternative way to use AI for data cleaning

Generating program code instead of processing data

- Free
- Independent
- Embeddable
- Secure



# Советы ИИ

**Prompt:** How can we use AI for verification and cleaning statistical data?

**Replay:**



Artificial Intelligence (AI) can be leveraged to verify statistical data in several ways, primarily by automating the processes of data validation, identifying anomalies, ensuring consistency, and improving the overall accuracy of datasets.

- Automated Data Cleaning
- Anomaly Detection
- Natural Language Processing (NLP) for Documented Data
- Predictive Analytics
- Data Reconciliation
- Quality Control in Data Entry
- Bias Detection and Fairness Analysis
- AI-Driven Statistical Testing
- Real-Time Verification in Big Data



# Спасибо!



[info@contourcomponents.com](mailto:info@contourcomponents.com)



[contourcomponents.com](https://contourcomponents.com)