



# Text Extraction and Clean-up in NLP



Abdallah Ashraf · Following

8 min read · Jul 31



97



## CLEAN-TEXT FOR NLP



Text extraction and cleanup refer to extracting the raw textual content from input data by removing irrelevant information like formatting, metadata and other non-text elements. The goal is to convert the text into the required encoding for natural language processing. The exact process depends on the formats of the available data within an organization, such as static data from

PDFs, HTML pages or plain text files, or a continuous stream of data. As shown in the figure below, text extraction and cleanup involves:

- Identifying the textual content within the input data, which can be in a variety of formats
- Removing any markup tags, metadata fields, or other code that is not part of the actual text
- Converting the extracted text into the proper encoding for analysis, such as UTF-8 for most NLP tasks today

**A**

**B**

```
<p style="text-align: justify;">
The book will be around 350 pages. It will be accompanied by a code
repository containing several Jupyter notebooks for all the chapters to
give a walk-through and explain the code in detail. The code base is in
Python and various machine learning and natural language processing
libraries. The book assumes that the readers have a good grasp of
programming but no theoretical and practical knowledge of NLP.
</p>

<p><br>
</p>

<a
href="https://www.oreilly.com/library/view/practical-natural-language/97
81492054047/">

</a>

<h1 style="color: #e74c3c;">Commonly Asked Questions</h1>
<ul>
<li> Can I contribute to the book?
<p>The book is accompanied by open source Jupyter notebooks and demo
applications. If you are a great ML or front-end engineer looking to
build something meaningful you can apply by filling <a
href="https://goo.gl/forms/dofNcw251ix26ajk1">this form</a>. Also refer
to the next question.
</p>
```

**C**

(a) PDF invoice, (b) HTML texts, and © text embedded in an image

The initial stage of text extraction and cleanup plays a crucial role in preparing unstructured data for natural language processing. It lays the foundation for further preprocessing steps like tokenization, lemmatization and part-of-speech tagging..

While text extraction itself does not require natural language processing techniques, it is an important step that affects the entire NLP pipeline. In fact, text extraction can be the most time-consuming part of an NLP project. This section will provide some examples to illustrate the various issues involved in text extraction and cleanup. We'll examine aspects of extracting text from different sources and preparing that text for downstream NLP tasks. we'll look at a few examples to illustrate different issues involved in this step in this section. We'll also touch on some of the important aspects of text extraction from various sources as well as cleanup to make them consumable in downstream pipelines.

## HTML Parsing and Cleanup

Say we're working on a project where we're building a forum search engine for programming questions. We've identified Stack Overflow as a source and decided to extract question and best-answer pairs from the website. How can we go through the text-extraction step in this case? If we observe the HTML markup of a typical Stack Overflow question page, we notice that questions and answers have special tags associated with them. We can utilize this information while extracting text from the HTML page. While it may seem like writing our own HTML parser is the way to go, for most cases we encounter, it's more feasible to utilize existing libraries such as Beautiful Soup and Scrapy, which provide a range of utilities to parse web pages. The following code snippet shows how to use Beautiful Soup to address the problem described here, extracting a question and its best-answer pair from a Stack Overflow web page:

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
myurl = "https://stackoverflow.com/questions/415511/ \
how-to-get-the-current-time-in-python"
html = urlopen(myurl).read()
```

```
soupified = BeautifulSoup(html, "html.parser")
question = soupified.find("div", {"class": "question"})
questiontext = question.find("div", {"class": "post-text"})
print("Question: \n", questiontext.get_text().strip())
answer = soupified.find("div", {"class": "answer"})
answertext = answer.find("div", {"class": "post-text"})
print("Best answer: \n", answertext.get_text().strip())
```

Here, we're relying on our knowledge of the structure of an HTML document to extract what we want from it. This code shows the output as follows:

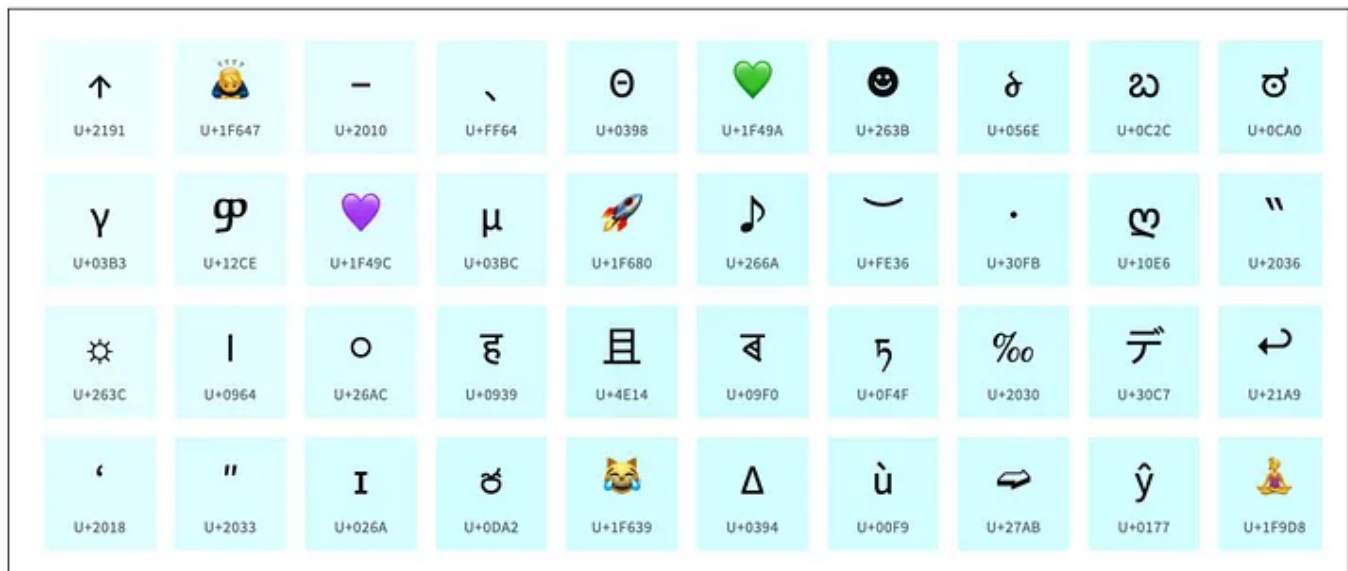
```
Question:
What is the module/method used to get the current time?
Best answer:
Use:
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2009, 1, 6, 15, 8, 24, 78915)
>>> print(datetime.datetime.now())
2009-01-06 15:08:24.789150
And just the time:
>>> datetime.datetime.now().time()
datetime.time(15, 8, 24, 78915)
>>> print(datetime.datetime.now().time())
15:08:24.789150
See the documentation for more information.
To save typing, you can import the datetime object from the datetime module:
>>> from datetime import datetime
Then remove the leading datetime. from all of the above.
```

In this example, we had a specific need: extracting a question and its answer. In some scenarios — for example, extracting postal addresses from web pages — we would get all the text (instead of only parts of it) from the web page first, before doing anything else. Typically, all HTML libraries have some function that can strip off all HTML tags and return only the content

between the tags. But this often results in noisy output, and you may end up seeing a lot of JavaScript in the extracted content as well. In such cases, we should look to extract content between only those tags that typically contain text in web pages.

## Unicode Normalization

As we develop code for cleaning up HTML tags, we may also encounter various Unicode characters, including symbols, emojis, and other graphic characters. A handful of Unicode characters are shown in the Figure.



Unicode characters

To parse such non-textual symbols and special characters, we use Unicode normalization. This means that the text we see should be converted into some form of binary representation to store in a computer. This process is known as text encoding. Ignoring encoding issues can result in processing errors further in the pipeline. There are several encoding schemes, and the default encoding can be different for different operating systems. Sometimes (more commonly than you think), especially when dealing with text in multiple languages, social media data, etc., we may have to convert

between these encoding schemes during the text-extraction process. You can refer to this book (Dickinson, Markus, Chris Brew, and Detmar Meurers. Language and Computers) for an introduction to how language is represented on computers and what difference an encoding scheme makes. Here is an example of Unicode handling:

```
text = 'I love 🍕! Shall we book a 🚗 to gizza?'
Text = text.encode("utf-8")
print(Text)
```

which outputs:

```
b'I love Pizza \xf0\x9f\x8d\x95! Shall we book a cab \xf0\x9f\x9a\x95
to get pizza?'
```

## Spelling Correction

In the world of fast typing and fat-finger typing, incoming text data often has spelling errors. This can be prevalent in search engines, text-based chatbots deployed on mobile devices, social media, and many other sources. While we remove HTML tags and handle Unicode characters, this remains a unique problem that may hurt the linguistic understanding of the data, and shorthand text messages in social micro-blogs often hinder language processing and context understanding. Two such examples follow:

**Shorthand typing:** Hllo world! I am back!

**Fat finger problem [20]:** I promise that I will not bresk the silence again!

While shorthand typing is prevalent in chat interfaces, fat-finger problems are common in search engines and are mostly unintentional. Despite our understanding of the problem, we don't have a robust method to fix this, but we still can make good attempts to mitigate the issue. Microsoft released a REST API that can be used in Python for potential spell checking:

```
import requests
import json
api_key = "<ENTER-KEY-HERE>"
example_text = "Hollo, wrld" # the text to be spell-checked
data = {'text': example_text}
params = {
    'mkt': 'en-us',
    'mode': 'proof'
}
headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Ocp-Apim-Subscription-Key': api_key,
}
response = requests.post(endpoint, headers=headers, params=params, data=data)
json_response = response.json()
print(json.dumps(json_response, indent=4))
```

Output (partially shown here):

```
"suggestions": [
  {
    "suggestion": "Hello",
    "score": 0.9115257530801
  },
  {
    "suggestion": "Hollow",
    "score": 0.858039839213461
  },
  {
    "suggestion": "Hallo",
    "score": 0.597385084464481
  }
]
```

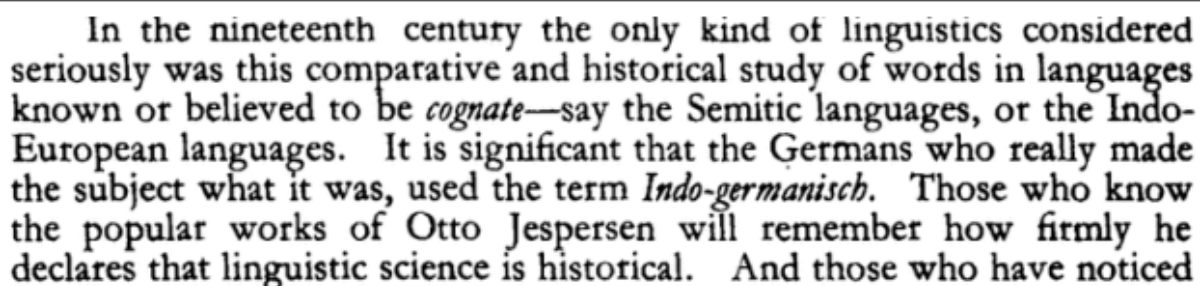
Going beyond APIs, we can build our own spell checker using a huge dictionary of words from a specific language. A naive solution would be to look for all words that can be composed with minimal alteration (addition,



deletion, substitution) to its constituent letters. For example, if “Hello” is a valid word that is already present in the dictionary, then the addition of “o” (minimal) to “Hllo” would make the correction.

## System-Specific Error Correction

HTML or raw text scraped from the web are just a couple of sources for textual data. Consider another scenario where our dataset is in the form of a collection of PDF documents. The pipeline in this case starts with extraction of plain text from PDF documents. However, different PDF documents are encoded differently, and sometimes, we may not be able to extract the full text, or the structure of the text may get messed up. If we need full text or our text has to be grammatical or in full sentences (e.g., when we want to extract relations between various people in the news based on newspaper text), this can impact our application. While there are several libraries, such as PyPDF, PDFMiner, to extract text from PDF documents, they are far from perfect, and it's not uncommon to encounter PDF documents that can't be processed by such libraries. Another common source of textual data is scanned documents. Text extraction from scanned documents is typically done through optical character recognition (OCR), using libraries such as Tesseract. Consider the example image — a snippet from a 1950 article in a journal .



In the nineteenth century the only kind of linguistics considered seriously was this comparative and historical study of words in languages known or believed to be *cognate*—say the Semitic languages, or the Indo-European languages. It is significant that the Germans who really made the subject what it was, used the term *Indo-germanisch*. Those who know the popular works of Otto Jespersen will remember how firmly he declares that linguistic science is historical. And those who have noticed

An example of scanned text

The code snippet below shows how the Python library pytesseract can be used to extract text from this image:

```
from PIL import Image
from pytesseract import image_to_string
filename = "somefile.png"
text = image_to_string(Image.open(filename))
print(text)
```

This code will print the output as follows, where “\n” indicates a newline character:

```
'in the nineteenth century the only Kind of linguistics considered\nseriously
was this comparative and historical study of words in languages\nknown or
believed to be cognate—say the Semitic languages, or the Indo-\nEuropean
languages. It is significant that the Germans who really made\nthe subject what
it was, used the term Indo-germanisch. Those who know\nthe popular works of
Otto Jespersen will remember how fitmly he\ndeclares that linguistic
science is historical. And those who have noticed'
```

We notice that there are two errors in the output of the OCR system in this case. Depending on the quality of the original scan, OCR output can potentially have larger amounts of errors. How do we clean up this text before feeding it into the next stage of the pipeline? One approach is to run the text through a spell checker such as pyenchant, which will identify misspellings and suggest some alternatives. More recent approaches use neural network architectures to train word/character-based language models, which are in turn used for correcting OCR text output based on the context .

If we take an example of a voice-based assistant. In such case, the source of text extraction is the output of an automatic speech recognition (ASR) system. Like OCR, it's common to see some errors in ASR, owing to various factors, such as dialectical variations, slang, non-native English, new or domain-specific vocabulary, etc. The above-mentioned approach of spell checkers or neural language models can be followed here as well to clean up the extracted text.

What we've seen so far are just some examples of potential issues that may come up during the text-extraction and cleaning process. Though NLP plays a very small role in this process, I hope these examples illustrate how text extraction and cleanup could pose challenges in a typical NLP pipeline.

Data Science

NLP

Data Cleaning

Natural language processing

Data Wrangling

### More from the list: "NLP"

Curated by Himanshu Birla



Jon Gi... in Towards Data ...

#### Characteristics of Word Embeddings

★ · 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

#### The Word2vec Hyperparameters

★ · 6 min read · Sep 3, 2021

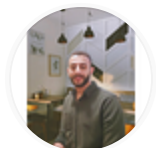


Jon Gi... in

#### The Word2vec

★ · 15 min read



[View list](#)

## Written by Abdallah Ashraf

126 Followers

Following

Data Analytics | Data Science | Tech enthusiast | Sharing knowledge

### More from Abdallah Ashraf

# reprocessing ext Data



Abdallah Ashraf

## Text Pre-Processing for NLP :

Let's start with a simple question: why do we still have to pre-process text?

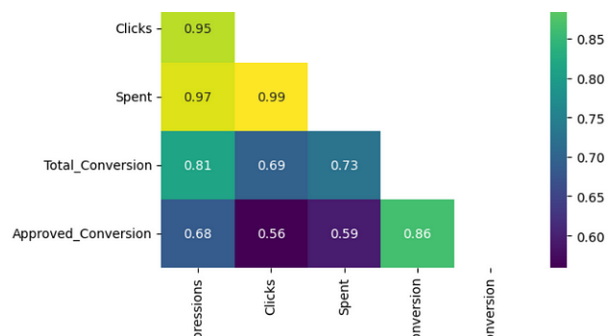
8 min read · Aug 31



86



14



Abdallah Ashraf

## Correlation in machine learning— All you need to know

What is correlation?

7 min read · Sep 22



Abdallah Ashraf

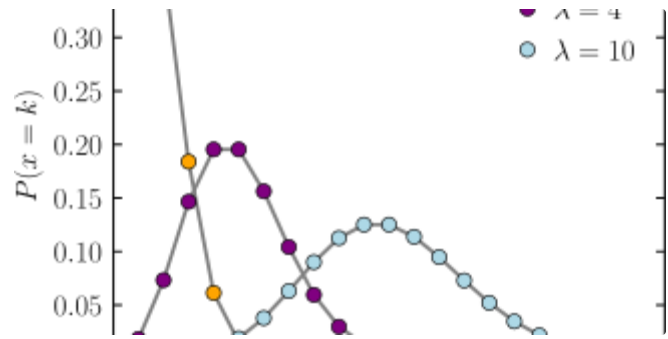
## How to get insights from a visualization

The process of gaining insights can be understood as an effort to increase your...

6 min read · Jul 20

233 2

...



Abdallah Ashraf

## Probability Distributions—Statistics for machine learning

Understanding Probability distributions

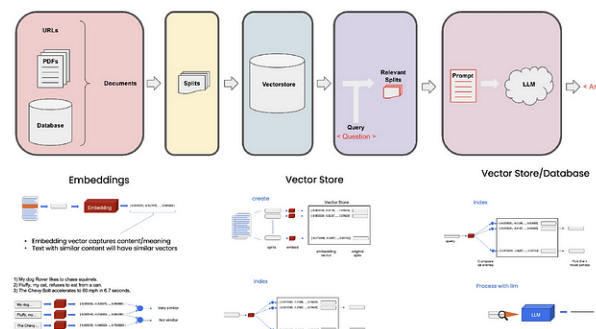
9 min read · 4 days ago

...

...

See all from Abdallah Ashraf

## Recommended from Medium





Alex Reed

## Writing Your First NLP Python Script: From Text Preprocessing t...

Welcome to the world of Natural Language Processing (NLP)! NLP is a fascinating field a...

5 min read · Sep 22



56



## TeeTracker

## Chat with your PDF (Streamlit Demo)

## Conversation with specific files

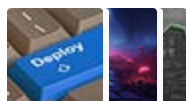
4 min read · Sep 15



56



## Lists



## Predictive Modeling w/ Python

20 stories · 452 saves



## New\_Reading\_List

174 stories · 133 saves



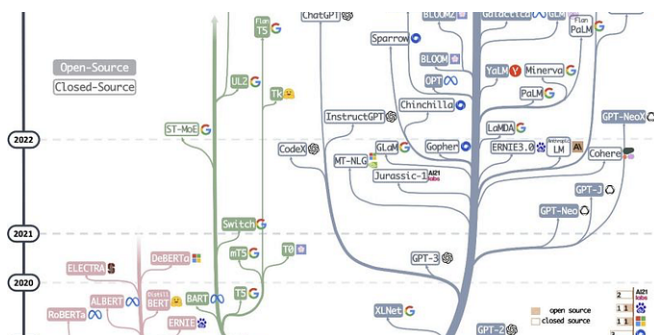
# Natural Language Processing

669 stories · 283 saves



## Practical Guides to Machine Learning

10 stories · 519 saves



```

Jokovic, Entity Label: PER, Confidence score: 0.9974638223648071
Open, Entity Label: MISC, Confidence score: 0.9965554475784302
Entity Label: LOC, Confidence score: 0.9993627664727063
ntity Label: MISC, Confidence score: 0.9981368780136108
Nadal, Entity Label: PER, Confidence score: 0.9987477660179138
Entity Label: MISC, Confidence score: 0.9151148796081543

```



Haifeng Li

# A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14



Seffa B

## Named Entity Recognition with Transformers: Extracting Metadata

3 min read · Jun 12

 372







 7









 Nimrita Koul

## Natural Language Processing with Python Part 5: Text Classification

This article is the fifth in the series of my articles covering the sessions I delivered for...

8 min read · May 4


 1









 Lukas Niederhäuser

## Exploratory Text Analysis of Swiss and German companies from...

The objective of this article is to gather and analyse text from Wikipedia pages that...

8 min read · Jul 5

 39









See more recommendations