Data Wrangling in Python

Sidharth Shah

Fafadia Tech

{Twitter: **@iamsidd**, Github: **sidharthshah**, Email: **sidharth@fafadiatech.com**}

Slides: https://github.com/fafadiatech/talks/blob/master/meetup/swift

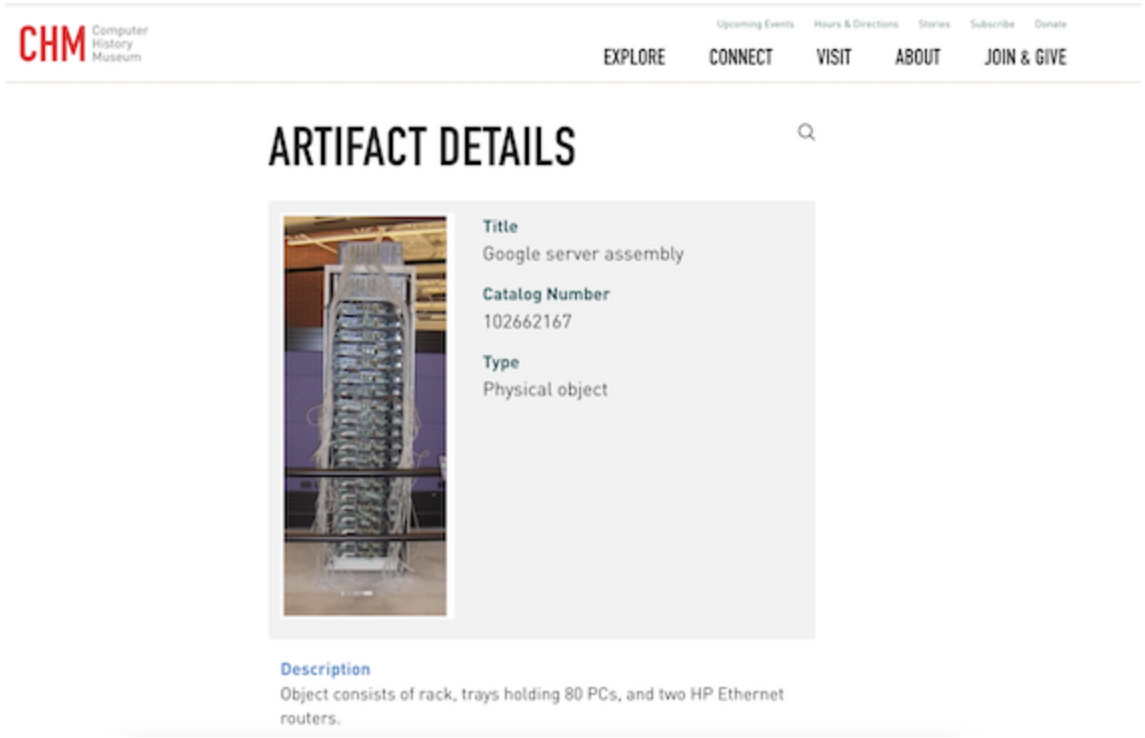
Introduction

What does it mean?

> Data wrangling is the process of removing errors and combining complex data sets to make them more accessible and easier to analyze.

Some motivating examples

Search Engines



Price Monitoring

Dynamic Pricing Engine

Creating Training Datasets

Search

Sign In

Register

ZALANDO RESEARCH · UPDATED 5 YEARS AGO

2424

New Notebook

Download (72 MB)

Fashion MNIST

An MNIST-like dataset of 70,000 28×28 labeled fashion images

Data Card | Code (1927) | Discussion (16)

About Dataset

Context

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

The original MNIST dataset contains a lot of handwritten digits. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. "If it doesn't work on MNIST, it won't work at all", they said. "Well, if it does work on MNIST, it may still fail on others."

Zalando seeks to replace the original MNIST dataset

Content

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

Usability

8.53

License

Other (specified in description)

Expected update frequency

Not specified

How does Google Work

Key components (at very high level)

1. Crawler: Fetches pages from internet
2. Indexer: Created inverted index term -> document
3. UI: Glue everything together

Ways to extract data

1. Crawler
2. API
3. Downloadable dataset

Toy Task

1. Input: List of all faculty pages
2. Output: CSV/TSV files tabulated results

Minimalist crawler

```
import requests

URL = "https://vcet.edu.in/"
response = requests.get(URL)
print(response.status_code)
print(response.text)
```

But wait.... this doesn't work

Lets try one more time

```
import requests

URL = "https://vcet.edu.in/"
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36'}
response = requests.get(URL, headers=headers)
print(response.status_code)
print(response.text)
```

What does a crawler look like?

```
frontier = []

while len(frontier) > 1:
    url = frontier.pop()
    html = download_page(url)
    all_urls = extract_urls(html)
    frontier.extend(all_urls)
```

Into RegEx rabbit hole

Places where regex are used extensively

1. Validation Rules {E.g. Email, Phone, Addresses}
2. Scraping {E.g. Extracting Prices, Text Snippets}
3. Translation {E.g. Replacing all upper cases to lower case}
4. Parsing Logs {E.g. Parsing Nginx and Apache logs}

Key components of RegEx

1. Expression
2. Quantifiers
3. Logic

Expression

1. `d` : Digits {0-9}
2. `w` : Word character this includes Alphabets {Case Insensitive}, Digits and Underscore
3. `s` : White spaces {including tabs, new line and character returns}

Note: Capital of above character match inverses. E.g. ‘D’ will match anything that is not a digit

Quantifiers

1. `.` : Matches occurance of any character
2. `?` : Optional Match
3. `+` : Match atleast one
4. `{x}` : Matches occurances of expression Exactly x number of times
5. `{x, y}` : Matches occurances of expression Range x, y number of times

Logic

1. `|` : Matches sub-expressions within the group
2. `(...)` : Encapsulate sub-expressions in a group
3. `^` : If used at starting of an expression it means “at the start”
4. `^` : If used in a group it negates

Class Characters

1. `[]` : Matches any of the character
2. `[a-z]` : Ranges of characters between a and z

Things to keep in mind while working with RegEx

1. RegEx are greedy by default {That means it tries to extract as much as possible until it conforms to a pattern even when a smaller part would have been syntactically sufficient.}
2. When writing regex make sure you test for positive and negative test cases

Examples

findall

```
import re
text_snippet = "there was a PEACH who PINCH, in return punch were flying around"
# re.compile compiles regex into an objects
# this makes it easier to work with regex
# re.IGNORECASE is a flag, you can have multiple such flags
pch_regex = re.compile(r"p.{1,3}ch", re.IGNORECASE)
for current_match in pch_regex.findall(text_snippet):
    print (current_match)
```

search

```
import re
def validate_email(current_email):
    """
    check if email is valid
    """
    email_re = re.compile(r"\w+\@\w+\.(com|co\.in)", re.IGNORECASE)
    # .search() method is used to TEST if regex matches at all
    return email_re.search(current_email) is not None
print validate_email("spammy@gmail.com")
print validate_email("spammy@.co")
```

finditer

```
import re
text_snippet = "there was a PEACH who PINCH, in return punch were flying around"
pch_regex = re.compile(r"p.{1,3}ch", re.IGNORECASE)
for current_match in pch_regex.finditer(text_snippet):
    print "Starts at:%d, Ends at:%d" % (current_match.start(), current_match.end())
```

BeautifulSoup4

Why use it?

1. RegEx have limitation
2. All web is HTML, everything is Tree
3. Make life easier

```
def extract_faculty(current):
    result = {}
    name = current.find('h2').text.strip()
    title = current.find('div', {'class': 'elementor-text-editor'})
    if not title:
        title = current.find("p")
    title = title.text.strip()
    email = "".join(extract_email(current.text))

    result['name'] = name
    result['title'] = title
    result['email'] = email
    return [result]
```

Real world challenges

1. How not to DDoS?
2. Distributed Crawling
3. Avoiding duplicates
4. Ensure sustained performance