

# Python による 「スクレイピング & 自然言語処理」入門

---

Twitter: @salinger001101

戸嶋 龍哉

参考資料(サンプルコード):

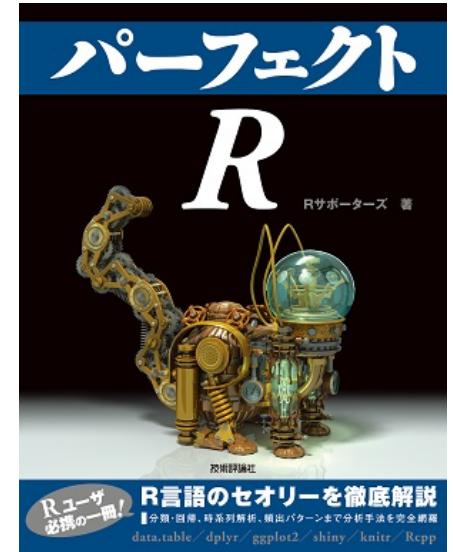
[https://github.com/Salinger/found\\_it\\_project\\_06\\_crwl/blob/master/src/python\\_crawler\\_nlp.ipynb](https://github.com/Salinger/found_it_project_06_crwl/blob/master/src/python_crawler_nlp.ipynb)

# 自己紹介

- ・氏名：戸嶋 龍哉（とじま たつや）
- ・DATUM STUDIO 株式会社にてデータエンジニアとして、さまざまな業種の企業におけるデータ分析活用基盤の構築、テキストマイニングによる分析、機械学習アルゴリズムの整備に従事。データ分析を活用し1円でも多くの収益を上げるべく、がんばっている。



## 著書



# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# セミナー概要

---

- ・ クローラーとはWeb上のデータを自動的に収集するための道具です。クローラーを活用することで、担当者が手動で行っていたWeb情報収集の効率化、また自社だけでは入手できないさまざまデータを取得し自社データと結合することで新たな示唆を得ることが可能になります。
- ・ 今回のセミナーでは初心者を対象にクローラーを作成し対象サイトのデータを収集、テキスト解析を行い、分析結果を得るまでの一連の流れについて、Python で使用するライブラリ、解析手法を交えて解説いたします。

※本発表は所属する組織とは一切関係がありません

# 今回やること

---

- 今回は対象ページとして 日本酒物語 日本酒ランキング(人数) とそれに紐づく各銘柄の詳細を収集し、解析を行います。
1. 解析のための下準備
  2. クローラーによるデータの収集
  3. テキスト解析
    - TFIDF によるレビュー中の特徴的な形容詞の抽出
    - 単語ベースのクラスタリング

# 進め方

---

- 基本的にはスライドの記載内容に従って進めていきますが、サンプルコードを見ながら進めるとより理解しやすいです。
- 手元に端末のある方は別途コードを記載したページを開いておいてください。

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# 質問 1

---

「クローラー」って単語  
セミナー前に聞いたことあった方  
手を上げていただけますか？

## 質問 2

---

「クローラー」  
実際に作ったことがある方  
手を上げていただけますか？

# そもそもクローラーって何？

- クローラー (crawler)
  - インターネット上にあるWebサイト、テキスト、画像、動画など、さまざまなデータを収集するプログラムのこと。
- スクレイピング, クローリング (crawling)
  - クローラーを動かして、データを収集すること。

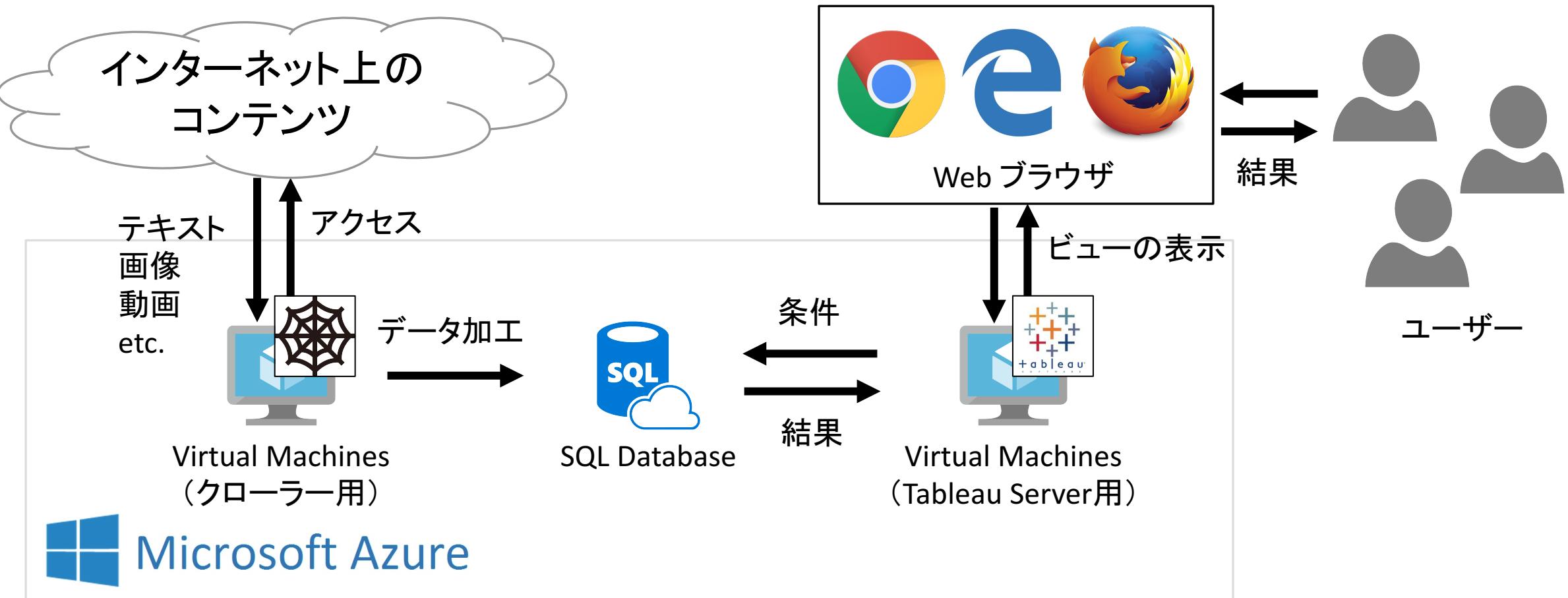


# クローラーのメリット

---

- 担当者が手動で行っていた Web からの情報収集の自動化。
  - 情報収集にかかっていた時間の大幅削減が可能に。
  - 人間には不可能な量の情報を収集することが可能に。
- データの自動的な蓄積が可能。
  - 一定期間で消えてしまうデータの長期間に渡る分析が可能に。

# データ分析で用いられる クローラーシステムの例 (Azure 環境)



# クローラー作成に関する技術 (クローラー本体)

---

- プログラミング言語:
  - 主に [Python](#) で作成。後の分析作業が楽。
- ライブラリ
  - Requests : Python の HTTP ライブラリ。
  - Scrapy : Web スクレイピングフレームワーク。
  - PhantomJS : コマンドラインで動作する仮想ブラウザフレームワーク。JavaScript を使用したサイトの取扱も可能になる。
  - BeautifulSoup : HTML のパーサ。HTML の構造を効率よく扱える。

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# 質問 1

---

「自然言語処理」って単語  
セミナー前に聞いたことあった方  
手を上げていただけますか？

## 質問 2

---

「自然言語処理」で  
何らかの解析を行ったことがある方  
手を上げていただけますか？

# 自然言語処理の分野（概要）

---

- 人間が日常的に使っている言葉（自然言語）をコンピュータに処理させるための技術。
- 基礎技術
  - 形態素解析： 文を形態素（言語で意味を持つ最小単位）に分解し、品詞などを判別する。
  - 構文解析： 文の構造の解析。修飾-被修飾の関係推定など。
  - 語義曖昧性解消： 複数意味を持つ語句の意味の特定。
  - 照応解析： 代名詞の推定、省略された語句の補完。

# 自然言語処理の応用範囲

---

- 自動要約文生成
- 情報抽出
- 情報検索
- 機械翻訳
- 質問応答システム  
(QAシステム)
- 光学文字認識 (OCR)
- 音声認識・合成
- 自然言語生成
- かな漢字変換
- 文書校正
- 固有表現抽出

etc.

# 自然言語処理の応用範囲

---

- 自動要約文生成
- 情報抽出
- 情報検索
- 機械翻訳
- 質問応答システム  
(QAシステム)
- 光学文字認識 (OCR)
- 音声認識・合成
- 自然言語生成
- かな漢字変換
- 文書校正
- 固有表現抽出

今回は  
このあたり！

etc.

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# Python の導入

---

- Python 初心者の方は Python 3 の最新版を導入すればよい。
  - 現時点の最新版は Python 3.6
- Anaconda を利用しての導入が楽なのでおすすめ。
  - インストーラーをダウンロードして実行するだけ。  
<https://www.continuum.io/downloads>
- 開発環境は Jupyter Notebook がおすすめ。
  - 今回見てもらってる HTML のような感じでコードを書ける。

# ライブラリの導入

---

- 例えば Mac なら Jupyter Notebook 上から次のコマンドを実行すれば導入できる。

## 1. MeCab の導入

```
!brew install mecab mecab-ipadic  
!pip install mecab-python3
```

## 2. クローラー関係のライブラリの導入

```
!conda install -y html5lib  
!conda install -y requests  
!conda install -y BeautifulSoup4
```

# 使用するライブラリの読み込み

```
In [3]: # ファイル操作
import glob
import csv

# データ処理・視覚化
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# クローラー
import time
from datetime import datetime
from bs4 import BeautifulSoup
import requests

# テキスト解析
import MeCab
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
```

# 定数の設定

```
In [4]: # 日本酒物語 日本酒ランキング（人数）の URL  
FOLLOWRANK_URL = "http://www.sakeno.com/followrank/"  
  
# クロール時の待ち時間  
WAIT_TIME = 5  
  
# 銘柄マスターの出力先  
MEIGARA_MASTER_PATH = "../data/meigara_maseter.csv"  
# 銘柄評価スコアの出力先ディレクトリ  
MEIGARA_SCORES_DIR = "../data/meigara_scores/"  
# 銘柄コメントの出力先ディレクトリ  
MEIGARA_COMMENTS_DIR = "../data/meigara_comments/"  
  
# TFIDF スコア算出後の結果出力先  
TFIDF_PATH = "../data/tfidf.csv"  
# クラスタリング結果の結果出力先  
CLUSTER_PATH = "../data/tfidf.csv"
```

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# クローラーによるデータ取得の流れ

---

1. 対象ページのHTMLの取得。
2. 取得対象情報が含まれている部分のタグの特定。
3. 取得対象情報のパース。
4. 結果の保存。

# 対象ページ

The screenshot shows the homepage of the Nihonshu Monogatari website. The main content area displays the '日本酒ランキング (人数)' (Sake Ranking by Number of People) table. The table has columns for '順位 (人數)' (Ranking (Number of People)), '銘柄名' (Brand Name), '蔵元名' (Brewery Name), and '商品' (Product). The top five entries are highlighted with a blue border. A blue box highlights the first entry: '1位 (85人) 獺祭 旭酒造 (山口県) 山口県 岩国市'. Below each entry is a link to search for products on Amazon, Rakuten, and Yahoo!.

日本酒ランキング (人数)			
順位 (人數)	銘柄名	蔵元名	商品
1位 (85人)	獺祭	旭酒造 (山口県) 山口県 岩国市	▼獺祭の商品を探す Amazon 楽天市場 Yahoo!
2位 (75人)	醸し人九平次	萬乘醸造 愛知県 名古屋市緑区	▼醸し人九平次の商品を探す Amazon 楽天市場 Yahoo!
3位 (61人)	出羽桜	出羽桜酒造 山形県 天童市	▼出羽桜の商品を探す Amazon 楽天市場 Yahoo!
4位 (60人)	田酒	西田酒造店 青森県 青森市	▼田酒の商品を探す Amazon 楽天市場 Yahoo!
5位 (58人)	黒龍	黒龍酒造 福井県 吉田郡	▼黒龍の商品を探す Amazon 楽天市場 Yahoo!

このページの  
□で囲まれた  
部分から順位、  
銘柄、蔵元名、  
銘柄の詳細ページ  
のURLを取得する。

# Request を使用してページ取得

---

```
In [5]: # ページの HTML を取得
response = requests.get(FOLLOWRANK_URL)

# 正しく取得できたかどうか HTTP ステータスコードで確認
if not response.status_code == 200:
    raise ValueError("Invalid response")
else:
    print("OK.")
```

OK.

# 中身の確認

---

```
In [7]: response.encoding = 'euc_jp'  
print(response.text[:1000])
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://ww  
w.w3.org/TR/html4/loose.dtd">  
<html lang="ja">  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=euc-jp">  
<meta http-equiv="Content-Style-Type" content="text/css">  
<meta http-equiv="Content-Script-Type" content="text/javascript">  
<meta name="viewport" content="width=device-width, initial-scale=1, maxim  
um-scale=4">  
<meta name="keywords" content="日本酒, ランキング, 口コミ, 評価">  
<title>日本酒ランキング（人数） - 日本酒物語</title>  
<link rel="stylesheet" href="http://www.sakeno.com/incfiles/sakeno.css">
```

# BeautifulSoup を利用して テーブル要素の取得

---

- <table> ~ </table> の部分を抜き出す。

```
In [8]: soup = BeautifulSoup(response.text, "lxml")
table = soup.body.find("table")
```

- 各行要素 <tr> ~ </tr> の部分を抽出

```
In [9]: trs = table.find_all("tr")[2:] # 先頭のゴミをカット
```

# 行要素のパース

```
def parse_tr(tr):
    # 順位
    tds = tr.find_all("td")
    rank = int(tds[0].get_text().split("位")[0])

    # 銘柄
    a = tds[1].find("a")
    meigara = a.get_text()
    detail_url = a.get("href")
    yomi = tds[1].find("div").string

    # 酒造
    location = tds[2].find_all("a")
    kuramoto = location[0].string
    prefecture = location[1].string
    city = location[2].string

    tr_1 = [
        rank, meigara, yomi,
        kuramoto, prefecture, city,
        detail_url
    ]
    return tr_1

ranking_list = [parse_tr(tr) for tr in trs]
ranking_list[:5]
```

```
[[1,
  '獺祭',
  'だっさい',
  '旭酒造（山口県）',
  '山口県',
  '岩国市',
  'http://www.sakeno.com/meigara/931'],
 [2,
  '釀し人九平次',
  'かもしづとくへいじ',
  '萬乘醸造',
  '愛知県',
  '名古屋市',
  'http://www.sakeno.com/meigara/735'],
 [3,
  '出羽桜',
  'でわざくら',
  '出羽桜酒造',
  '山形県',
  '天童市',
  'http://www.sakeno.com/meigara/219']]
```

# データフレームへの変換

```
In [11]: meigara_master_df = pd.DataFrame(  
    ranking_list,  
    columns=[ "rank", "meigara", "yomi",  
              "kuramoto", "prefecture", "city",  
              "detail_url" ]  
)  
  
# ユニークな連番 ID を追加  
meigara_master_df[ "meigara_id" ] = meigara_master_df.index.to_series() + 1  
meigara_master_df = meigara_master_df[  
    [ "meigara_id",  
      "rank", "meigara", "yomi",  
      "kuramoto", "prefecture", "city",  
      "detail_url" ]  
]
```

# CSVとして保存

```
# 銘柄マスタデータの出力
meigara_master_df.to_csv(
    MEIGARA_MASTER_PATH,
    encoding="utf-8",
    sep=",",
    index=False
)

meigara_master_df.head()
```

	meigara_id	rank	meigara	yomi	kuramoto	prefecture	city	detail_url
0	1	1	獺祭	だっさい	旭酒造 (山口県)	山口県	岩国市	<a href="http://www.sakeno.com/meigara/1">http://www.sakeno.com/meigara/1</a>
1	2	2	醸し人 九平次	かもしげ とくへい じ	萬乘醸造	愛知県	名古屋市	<a href="http://www.sakeno.com/meigara/2">http://www.sakeno.com/meigara/2</a>
2	3	3	出羽桜	でわざくら	出羽桜酒 造	山形県	天童市	<a href="http://www.sakeno.com/meigara/3">http://www.sakeno.com/meigara/3</a>
3	4	4	田酒	でんしゅ	西田酒造 店	青森県	青森市	<a href="http://www.sakeno.com/meigara/4">http://www.sakeno.com/meigara/4</a>
4	5	5	黒龍	こくりゅう	黒龍酒造	福井県	吉田郡	<a href="http://www.sakeno.com/meigara/5">http://www.sakeno.com/meigara/5</a>

# 詳細ページ（レビュー部分）

日本酒口コミNo.6193  
[すっきりして飲みやすい](#)

おいしいです  
あいうそん (2016年10月20日 11時57分54秒)

日本酒口コミNo.6126  
[獺祭 等外23](#)

獺祭 等外23 山田錦23 生酒 27BY  
ライチ様な立ち香、含み香。抜ける香りはやや甘く。  
ソフトなタッチ、ウェットな質感、甘みはやや強め。  
味は序盤が強く、中盤以降は終息していくやや苦み。

通常版の二割三分のようなさらりとした淡さがなく、やや濃い目の甘みが特徴的でした。  
等外ってことで敢えて造り分けてるのかと思いましたが、コレ書いてて気付きました、これは生なんですね。  
¥23k（税抜）かあ、うーん。  
富生谷欠 (2016年08月08日 22時20分28秒)

日本酒口コミNo.5992  
[獺祭50](#)

獺祭と言えば高精米、磨きが強調され、50%はその最低ランクである  
しかし全国の銘酒蔵もこの50%クラスで、ずいぶんと美味しい酒を醸す。  
とわ言えこの獺祭50、美味しい。この酒質、全国銘酒蔵も意識せざるおえない  
のでは。  
岩国のお土産で2割3部を見かけた時はビックリしたが、50はとにかくなかなか手に入らない。  
イオンでは一升瓶10000円越えするが、銘酒店で2850円？の定価販売  
した。  
4合瓶で1500円。今日は久しぶりにありつけた。

- コメント一覧から次の要素を取得する。

1. 投稿ID
2. タイトル
3. 投稿日時
4. ユーザ名
5. テキスト

※流れとしては同じなので省略。  
詳細はサンプルコードを参照。

# ここまでまとめ

---

- クローラーを活用することにより…
  - 銘柄のリスト情報（銘柄マスタ）を取得できた。
  - 銘柄マスタに紐づく各詳細ページ（200ページ文）のコメント一覧を取得できた。

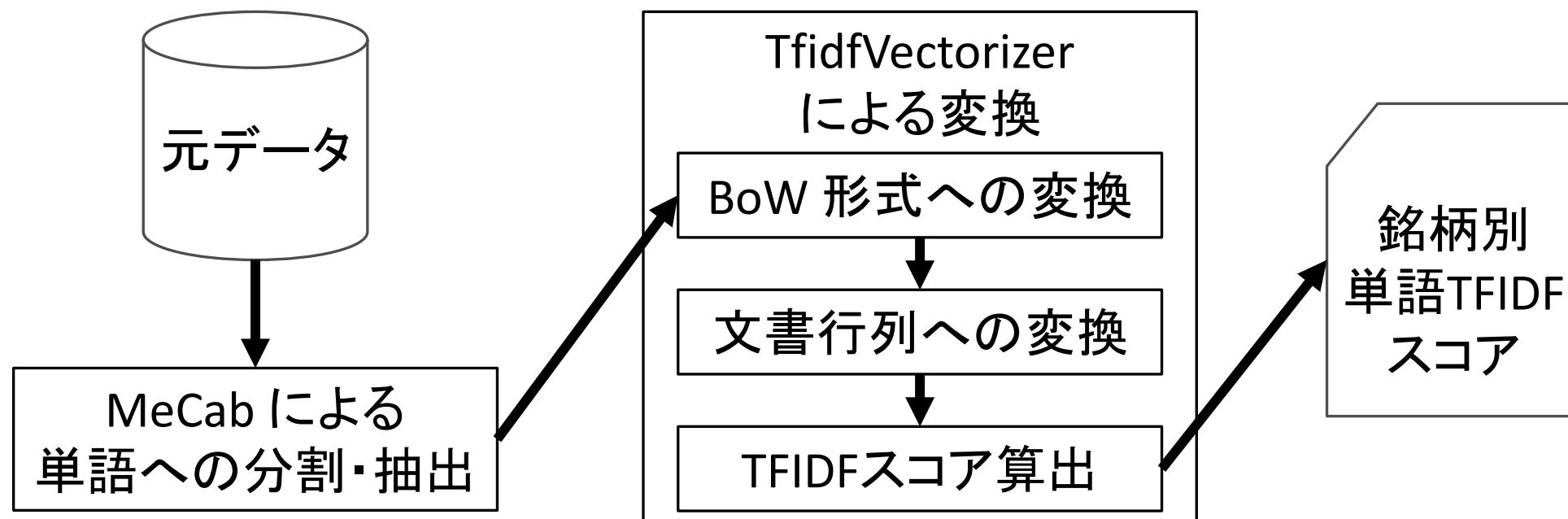
# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# TFIDFによるレビュー中の特徴的な形容詞の抽出

- この解析では、あるドキュメント中における特徴的な単語（特徴語）の抽出を行う。



# Bag-of-Words (BoW) 形式への変換

- ・集めたデータを各種統計処理で扱うためには、行列形式への変換が必要。
- ・文章に単語が含まれているかどうかのみを考え、単語の並び方などは考慮しないモデル。一番シンプルなモデルは単語があれば 1、なければ 0 となる。単語の出現回数を (TF: Term Frequency) 使う場合もある。
- ・今回は Term Frequency を採用。

すももももももものうち (1)

↓ MeCab による変換

[すもも, も, もも, も, もも, の, うち] (2)

↓ CountVectorizer による変換

{すもも: 1, も:2, もも: 2, の: 1, うち:1} (3)

# 文書行列への変換

---

- 各文書に含まれる単語を列に、文書を行、単語の出現回数を要素とした行列形式に変換する。
- 例：以下のテキストの文書行列への変換
  - (a) 「すもももももももものうち」
  - (b) 「料理も景色もすばらしい」
  - (c) 「私の趣味は写真撮影です」

列のラベルは単語の出現の早い順に

[すもも, も, もも, の, うち, 料理, 景色, 素晴らしい, 私, 趣味, は, 写真撮影, です]

とすると、文書行列は下記のようになる。

```
[[1,2,2,1,1,0,0,0,0,0,0,0], # (a)
 [0,2,0,0,0,1,1,1,0,0,0,0], # (b)
 [0,0,0,1,0,0,0,0,1,1,1,1]] # (c)
```

# TFIDFの計算

---

- $\text{TFIDF} = \text{TF} \times \text{IDF}$
- TF : 文書内における単語の出現頻度。
  - 「ある文書中である単語が何回出現したか」
  - 1つの文書に多く出現する単語ほど重要度が高くなる。
- IDF : (ある単語が含まれている文書数 / 全ての文書数) の逆数
  - 多数の文書に出現する単語ほど重要度が低くなる。

$$\text{TFIDF}(w, d) = \text{TF}_{w,d} \times \log\left(\frac{N_d}{DF_{w,d}}\right)$$

TF: ある文書  $d$  に出現した単語  $w$  の回数  
DF: 単語  $w$  が含まれている文書の数  
N: 全ての文書数

# TFIDFの意味

---

- TFIDF が大きな値になる。  
→ 文書内である特定の単語が多く出現し、かつその単語は他の文書ではほとんど出現しない。
- 例
  - 「私」 : 各文書内における出現回数が多いが、多くの文書に出現するので重要度は低い。
  - 「特許」 : 「特許」を話題の中心にしている特定の文書内には多く現れ、一般的な文書には現れないで重要度が高い。

# TfidfVectorizer を用いた変換

---

- CountVectorizer (from scikit-learn)
  - BoW 形式への変換、文書行列への変換、単語と列番号の対応付けなどの作業をまとめて行うことができる。
- TfidfVectorizer (from scikit-learn)
  - 前述の CountVectorizer による行列化と TFIDF の計算を同時にを行うことが出来る。
  - 今回はこれを利用する。

# 対象品詞の単語群を返す関数を定義

```
In [16]: def split_text(text, target_pos=["形容詞"]):
    tagger = MeCab.Tagger()
    text_str = text
    tagger.parse('')
    node = tagger.parseToNode(text_str)

    words = []
    while node:
        l = node.feature.split(',')
        pos = l[0]
        if pos in target_pos:
            # unicode 型に戻す
            if l[6] == "*":
                word = node.surface # 変化しない語は表層形をそのまま使う
            else:
                word = l[6]           # 動詞や形容詞は原形を使う
            words.append(word)
        node = node.next
    return " ".join(words)      # スペース区切りで単語を結合し返す
```

# 全銘柄のコメントを読み込んで 単語単位に分割・対象品詞の単語のみ抽出

```
In [17]: # 全ファイル読み込み
comment_files = glob.glob("../data/meigara_comments/*.csv")

# 縦方向に単純結合
comment_df = pd.concat([pd.read_csv(f) for f in comment_files])
comment_df = comment_df.reset_index()

# 全ての text に対して形容詞の抽出を行う
comment_df[ "split" ] = comment_df[ "text" ].map(split_text)
comment_df.head()
```

	index	meigara_id	toukou_id	title	created_at	user_name	text	split
0	0	1	6193	すっきりし て飲みやす い	2016-10- 20 11:57:54	あいうそん	おいしいです	おいし い
1	1	1	6126	獺祭 等外2 3	2016-08- 08 22:20:28	富牟谷欠	獺祭 等外2 3 山田錦2 3 生酒 2 7 B Y ライチ様な立ち香、含み香。抜ける 香りはやや甘く。...	甘い 強 い 淡い ない 濃 い
2	2	1	5992	獺祭50	2016-04- 26 19:24:02	季がらし	獺祭と言えば高精米、磨きが強調さ れ、50%はその最低ランクであるし かし全国の銘酒蔵もこの5...	美味い 美味しい

# 銘柄別に単語をまとめる

```
In [18]: meigara_comments_df = comment_df\  
    .groupby("meigara_id")["split"]\  
    .apply(lambda x: "%s" % ' '.join(x))\  
    .reset_index()  
meigara_comments_df.head()
```

Out[18]:

	meigara_id	split
0	1	おいしい 甘い 強い 淡い ない 濃い 美味い 美味い 美味しい 悪い 若々しい 青い 鋭い...
1	2	ない ない 美味い 美味い ない 早い 早い 若い 美味い 高い たまらない いい 甘い ...
2	3	甘い 柔らかい 無い 美味い イイ やすい 力強い 鋭い 美味い 甘い くどい 旨い 甘い ...
3	4	いい 若い 深い 無い 新しい 甘い 濃い 旨い 美味い 高い 物足りない 甘酸っぱい 良...
4	5	軽い 良い ほしい 強い うまい 不味い 苦い 良い 良い 広い 美味しい ない 美味しい...

# TFIDFの計算

---

```
In [19]: vectorizer = TfidfVectorizer()
tfidfs = vectorizer.fit_transform(meigara_comments_df[ "split" ])
tfidfs
```

```
Out[19]: <200x292 sparse matrix of type '<class 'numpy.float64'>'  
with 4099 stored elements in Compressed Sparse Row format>
```

# 各銘柄 TFIDF スコア上位5件の 単語を抽出

```
In [20]: ## TFIDF の結果からi 番目のドキュメントの特徴的な上位 n 語を取り出す
def extract_feature_words(terms, tfidfs, i, n):
    tfidf_array = tfidfs[i]
    top_n_idx = tfidf_array.argsort()[-n:][::-1]
    words = [terms[idx] for idx in top_n_idx]
    return words
```

```
In [21]: # index 順の単語のリスト
terms = vectorizer.get_feature_names()
```

```
In [22]: top_n = 5 # 上位5件
highscore_words = [
    extract_feature_words(terms, tfidfs.toarray(), i, top_n)
    for i
    in range(len(meigara_comments_df.index))
]
highscore_words_str = " ".join(l) for l in highscore_words]
meigara_comments_df["highscore_words"] = highscore_words_str
```

# 銘柄マスタと結合して結果の確認

```
In [23]: # マスタデータの JOIN  
master_df = pd.read_csv(MEIGARA_MASTER_PATH)  
result_df = master_df.merge(meigara_comments_df, on="meigara_id", how="inner")  
result_df.head()
```

Out[23]:

	meigara_id	rank	meigara	yomi	kuramoto	prefecture	city	detail_url	split	highscore_words
0	1	1	獺祭	だっさい	旭酒造 (山口 県)	山口県	岩 国 市	<a href="http://www.sakeno.com/meigara/931">http://www.sakeno.com/meigara/931</a>	おいしい 甘い 強い 淡い ない 濃い 美味い 美味い 美味しい 悪い 若々しい 青い 錐い...	良い 美味しい 悪 い ない うまい
1	2	2	醸し人 九平次	かもしひ とくへい じ	萬乘醸造	愛知県	名 古 屋 市	<a href="http://www.sakeno.com/meigara/735">http://www.sakeno.com/meigara/735</a>	ないない 美味い 美味い ない 早い 早い 若い 美味 い 高い いたまらない いい 甘い ...	美味い 不味い な い 甘い 旨い
2	3	3	出羽桜	でわざく ら	出羽桜酒 造	山形県	天 童 市	<a href="http://www.sakeno.com/meigara/219">http://www.sakeno.com/meigara/219</a>	甘い 柔らかい 無い 美味 い イイ やすい 力強い 錐 い 美味い 甘い くどい 旨 い 甘い ...	素晴らしい 甘い やすい イイ 美味 しい

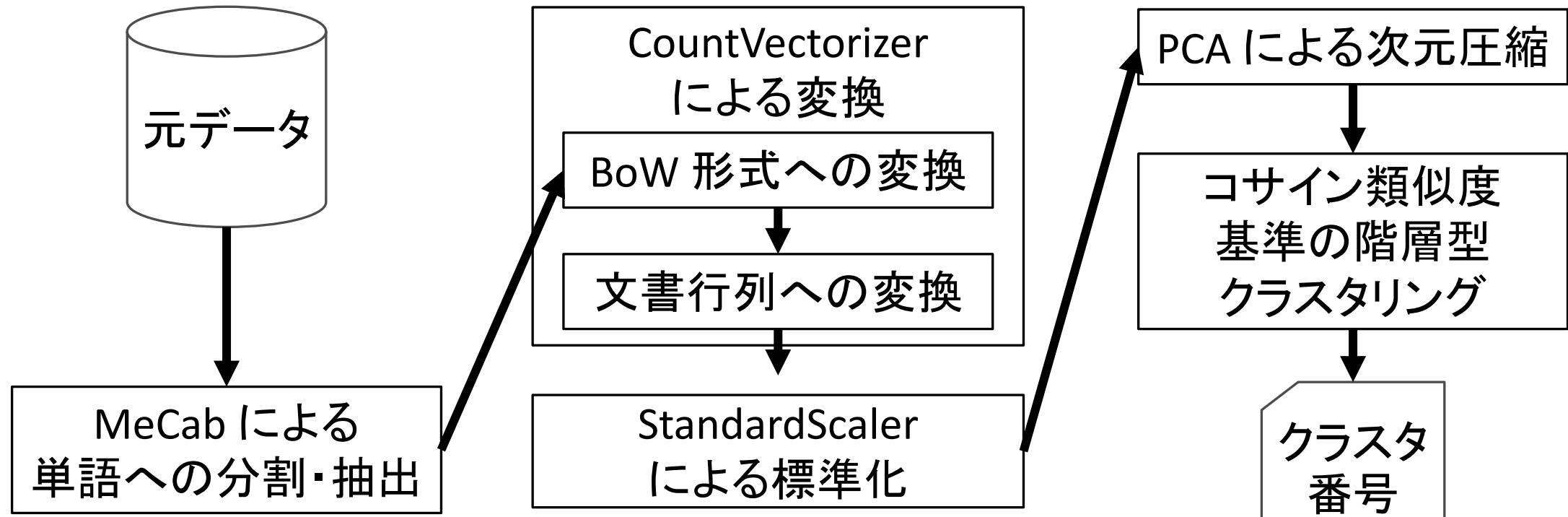
# TFIDF 計算結果の保存

---

```
In [24]: target_cols = ["rank", "meigara", "kuramoto", "detail_url", "highscore_words"]
result_df[target_cols].to_csv(TFIDF_PATH, index=False)
```

# 単語ベースのクラスタリング

- レビュー内に含まれている単語の傾向が似ている銘柄  
= 特徴が似ている銘柄と仮定して解析してみる。



# 単語の抽出

- 今回は 名詞 / 動詞 / 形容詞 を使用

```
In [25]: noun_verb_adj_words = comment_df[ "text" ]\n    .apply(lambda x: split_text(x, target_pos=[ "名詞", "動詞", "形容詞" ]))\ncomment_df[ "split_noun_verb_adj" ] = noun_verb_adj_words\nmeigara_comments_df = comment_df\\n    .groupby("meigara_id")[ "split_noun_verb_adj" ]\\n    .apply(lambda x: "%s" % ' '.join(x))\\n    .reset_index()\n\nmeigara_comments_df.head()
```

Out[25]:

	meigara_id	split_noun_verb_adj
0	1	おいしい 獺祭 等外 2 3 山田 錦 2 3 生酒 2 7 BY ライチ 様 立ち 香 ...
1	2	Wow very oishii Sake ! 米吟釀 aka 酿す 人 九平次 彼 地 ...
2	3	上立つ仄か甘い香り含み柔らかい入る派手さ無い純大吟旨み特徴個 ...
3	4	好き酒一つするコクあるいい感じ田酒特純生飲む開栓直後含むす...
4	5	甘み感じるする辛口。後味軽い酸味ある。ラベル飲む方「冷やす」...

# CountVectorizer による文書行列化

```
In [26]: vectorizer = CountVectorizer()
word_counts = vectorizer.fit_transform(meigara_comments_df.split_noun_verb_adj)
wca = word_counts.toarray()
wca
```

```
Out[26]: array([[0, 0, 0, ..., 0, 0, 1],
                 [0, 0, 0, ..., 0, 1, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

# StandardScaler による標準化

---

- ・今回のようにレビューの件数が大きく異なる場合、数値を標準化すると良いクラスタリング結果が得やすい。
- ・各要素から平均を引いて、標準偏差で割る（Zスコア）。
- ・この変換を行うと、平均が 0 で標準偏差・分散が 1 になる。

```
In [27]: sds = StandardScaler()  
X = sds.fit_transform(wca)
```

# PCA による次元圧縮

- 単語行列は疎な状態。 次元圧縮を行うことで、より効率が良く、直感に近いクラスタリング結果を得られる。
- 今回は 7252 次元（単語数）から 30 次元に圧縮。

```
In [28]: model = PCA(n_components=30)
X_decomp = model.fit_transform(X)
X_decomp
```

```
Out[28]: array([[ 2.28167681e+02, -1.16255034e+02, -4.89383813e+01, ...,
   9.14023707e-02, -8.60942515e-01, -6.03637929e-01],
   [ 1.12847840e+02,  1.94123781e+02, -6.26221348e+01, ...,
   3.46364954e-01, -1.24565097e+00, -6.24311590e-01],
   [ 2.25193159e+01,  6.46982151e+00,  1.69020459e+01, ...,
   1.00232518e+00,  6.03836210e-02, -5.87677278e-02],
   ...,
```

# クラスタリング実行

---

```
In [29]: model = AgglomerativeClustering(n_clusters=6, linkage="average", affinity="cosine")
y = model.fit_predict(X_decomp)
y
```

```
Out[29]: array([2, 2, 2, 2, 2, 5, 0, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 3, 0, 0,
 1, 2, 3, 3, 3, 3, 0, 0, 0, 3, 0, 2, 0, 2, 1, 3, 3, 2, 1, 3, 3, 0,
 0, 3, 3, 3, 3, 3, 3, 1, 2, 0, 3, 3, 3, 3, 0, 0, 3, 2, 3, 3, 3, 3,
 3, 3, 0, 0, 3, 2, 3, 3, 0, 3, 2, 3, 3, 3, 3, 3, 0, 4, 3, 2, 3, 3, 3,
 3, 2, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 0, 3, 3,
 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 0, 3, 3, 3, 3, 3,
 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3,
 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

# 銘柄マスタと結合し結果の確認

```
In [30]: # マスタデータの JOIN
master_df = pd.read_csv(MEIGARA_MASTER_PATH)
result_df = master_df.merge(meigara_comments_df, on="meigara_id", how="inner")

result_df["cluster"] = y
result_df.sort_values(by=[ "cluster", "rank" ])
```

Out[30]:

	meigara_id	rank	meigara	yomi	kuramoto	prefecture	city	detail_url	split_noun_verb_adj	cluster
6	7	7	新政	あらまさ	新政酒造	秋田県	秋田市	<a href="http://www.sakeno.com/meigara/55">http://www.sakeno.com/meigara/55</a>	冷蔵する いる 気つける 開栓する ブシャー 吹き出す しまう 半年 寝かせる いる...	0
8	9	7	鳳凰美田	ほうおう びでん	小林酒造 (栃木県)	栃木県	小山市	<a href="http://www.sakeno.com/meigara/95">http://www.sakeno.com/meigara/95</a>	鳳凰 美田 濾過 本生 20 17 / 2 頂く アル添フル ーティー 香り ラベル...	0
9	10	10	風の森	かぜのも り	油長酒造	奈良県	御所市	<a href="http://www.sakeno.com/meigara/898">http://www.sakeno.com/meigara/898</a>	風森 ALPHA TYPE 3 米吟 醸八錦 5027 BY 軽 い吟醸 香含...	0

# 結果の出力

---

```
In [31]: target_cols = ["rank", "meigara", "kuramoto", "detail_url", "cluster"]
result_df[target_cols].to_csv(CLUSTER_PATH, index=False)
```

# ここまでまとめ

---

- ・クロールしてきた文書から特徴語が抽出できた。
- ・クロールしてきた文書間で似ている者同士をまとめることができた。

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# 今回カバーできなかった点

---

- 形態素解析用辞書の改善
  - デフォルトのものだと「山田錦」が「山田」 + 「錦」に分割されてしまう。  
また連続する名詞の扱い方を改善するためにも辞書を工夫する必要がある。
- 否定語の扱い
  - 美味しくない → 美味しい + ない と分割される。  
このままでは「美味しい」としてカウントされてしまう。
- 数値を含んだ単語の取扱
  - 例えばアルコール度数を表すような数値、精米歩合の数値などが  
うまく扱えていない。
- ノイズとなるような単語のカット
  - 例えば「Wow very oishii Sake！」などのテキストが含まれているが、  
このようなものが含まれていてもあまり有益な結果を得られない。

# まとめ

---

- Python を用い、クローラーを作成することで対象サイトのデータを自動的に収集できた。
- テキスト解析を行い、特徴語の抽出、類似文書のクラスタリングの結果を得ることができた。

# 目次

---

1. はじめに
2. クローラーとは？
3. 自然言語処理とは？
4. 解析のための下準備
5. クローラーによるデータの取得
6. テキスト解析
7. おわりに
8. 付録

# 参考文献

---

- 日本酒物語 日本酒ランキング（人数）： <http://www.sakeno.com/followrank/>