Webスクレイピングの活用によるデータ収集と利活用

Webページから情報やデータを自動＋大量に取得する技術

Webページ　　Webスクレイピング　　データ抽出・出力

Source: https://www.keywalker.co.jp/web-crawler/web-scraping.html

Webスクレイピングを用いたライブラリ

BeautifulSoup
requests
time
(Random)
⇒HTMLやXMLからデータを取得

pandas_datareader　Web上の様々なソースにアクセスできるPythonライブラリ
⇒経済データや金融商品のデータを取得

※注意点：規約違反になっていないか、確認の上実行してください。
Twitter, Instagram, Amazon, メルカリは、規約で禁止

Webスクレイピングを使って検索結果ページの画像を自動保存

```
#●画像ファイルをダウンロードするための準備
# ①-①.ライブラリをインポート
import time
import re
import requests
from pathlib import Path
from bs4 import BeautifulSoup
# ①-②.出力フォルダを作成
output_folder = Path('いらすとや')
output_folder.mkdir(exist_ok=True)
# ①-③.スクレイピングしたいURLを設定
url = 'https://www.irasutoya.com/search?q=%E7%8C%AB'
# ①-④.画像ページのURLを格納するリストを用意
linklist = []

#●検索結果ページから画像のリンクを取り出す
# ②-①.検索結果ページのhtmlを取得
html = requests.get(url).text
# ②-②.検索結果ページのオブジェクトを作成
soup = BeautifulSoup(html, 'lxml')
# ②-③.画像リンクのタグをすべて取得
a_list =soup.select('div.boxmeta.clearfix > h2 > a')
# ②-④.画像リンクを1つずつ取り出す
for a in a_list:
# ②-⑤.画像ページのURLを抽出
    link_url = a.attrs['href']
# ②-⑥.画像ページのURLをリストに追加
    linklist.append(link_url)
    time.sleep(1.0)
```

import time
import re 画像を探す時、文字列の検索に使います
import requests Webページの読み込みに使います
from pathlib import Path 画像を保存する時、ファイルパスの操作に使います
from bs4 import BeautifulSoup 画像を探す時、htmlの検索に使います

「いらすとや」というフォルダ名のPathオブジェクトを作成して、変数output_folderとします

変数output_folderから「いらすとや」フォルダを作成します
同じ名前のフォルダがあってもエラーが起きないように指定します

画像をダウンロードしたいページのURLを変数urlに設定します

変数linklistとして空のリストを作成します

requests.get(url)で、検索結果ページ(変数url)からデータ(Responseオブジェクト)を取得します。オブジェクトのtext属性を使って、htmlの文字列で取り出します。取得した検索結果ページのhtmlを、変数htmlとします

BeautifulSoup(html, 'lxml')で、検索結果ページのhtml(変数html)からBeautifulSoupオブジェクトを作成して、変数soupとします

BeautifuSoupオブジェクト(変数soup)から、指定したタグの要素(tagオブジェクト)を全て抽出します

取得したaタグをリストにしたa_listの中から一つずつ取り出してfor文以下の処理をしていきます

```
# ●各画像ページから画像ファイルのURLを特定
# ③-①.画像ページのURLを1つずつ取り出す
for page_url in linklist:
# ③-②.画像ページのhtmlを取得
    page_html = requests.get(page_url).text
# ③-③.画像ページのオブジェクトを作成
    page_soup = BeautifulSoup(page_html, "lxml")
# ③-④.画像ファイルのタグをすべて取得
    img_list = page_soup.select('div.entry > div > a > img')
# ③-⑤.imgタグを1つずつ取り出す
    for img in img_list:
# ③-⑥.画像ファイルのURLを抽出
        img_url = (img.attrs['src'])
# ③-⑦.画像ファイルの名前を抽出
        filename = re.search(".*¥/(.*png|.*jpg)$",img_url)
# ③-⑧.保存先のファイルパスを生成
        save_path = output_folder.joinpath(filename.group(1))
        time.sleep(1.0)
# ●画像ファイルのURLからデータをダウンロード
        try:
# ④-①.画像ファイルのURLからデータを取得
            image = requests.get(img_url)
# ④-②.保存先のファイルパスにデータを保存
            open(save_path, 'wb').write(image.content)
# ④-③.保存したファイル名を表示
            print(save_path)
            time.sleep(1.0)
        except ValueError:
# ④-④.失敗した場合はエラー表示
            print("ValueError!")
```

for文で、リスト(linklist)から画像ページのURLを1つずつ取り出します。取り出したURLを変換page_urlとして、③-②から④-④までの処理をリストにある全てのURLにループ処理(プログラムに「繰り返し同じ処理をさせる」仕組み)していきます

requests.get(page_url)で画像ページ(変数page_url)からResponseオブジェクトを取得します。オブジェクトのtext属性を使いhtmlの文字列を取り出して変数page_htmlとします

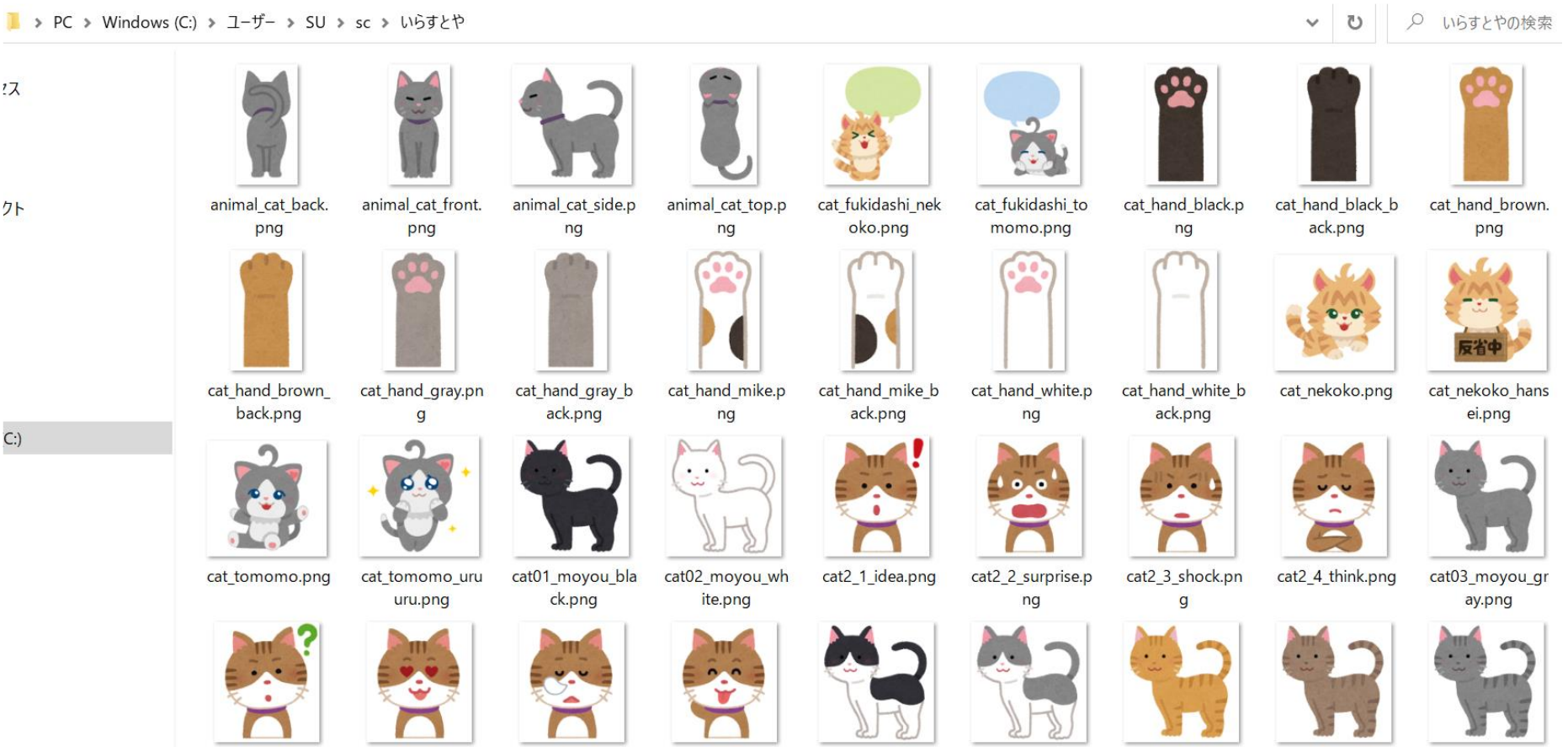
BeautifulSoup(page_html, “lxml”)で、画像ページのhtml(変数page_html)からBeautifulSoupオブジェクトを作成します。こうすることで、タグや属性を使って要素を探すことができる状態になります。作成した画像ページのBeautifulSoupオブジェクトは変数page_soupとします

BeautifuSoupオブジェクト(変数page_soup)から指定したタグの要素(tagオブジェクト)をリストで全て抽出します

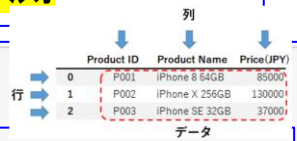
第一引数の(".*¥/(.*png|.*jpg)$",img_url)では正規表現で画像ファイルを指定しています

open(save_path, 'wb')で出力先のファイルパス(変数save_path)を書き込みモード(wb)で開いてから、write(image.content)で画像ファイル(変数image)のバイナリデータ(content属性)を書き込んで保存します

スクレイピングが成功した画像はフォルダ内に保存される



WebスクレイピングとPythonの機械学習ライブラリ「Scikit-learn」を用いてデータ収集＋可視化＋予測 前日から株価が上がったかどうかを予測



入力 [1]:

```
import pandas_datareader.data as web
import pandas as pd
import datetime
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

#Get Nikkei stock average from web
nikkei = web.DataReader("NIKKEI225", "fred", "1950/1/1")
plt.plot(nikkei)

#Transform to Logarithmic rate of return
nikkei['log_change']=np.log(nikkei['NIKKEI225'])-np.log(nikkei['NIKKEI225'].shift(1))
nikkei2=nikkei['log_change'].values

#Reshape for matrix with 7days data + 8th-day data as one line
term = 7
pricedata = []

length=len(nikkei2)
for i in range(0,length-term-1):
    pricedata.append(nikkei2[i:i+term+1])

df=pd.DataFrame(np.array(pricedata).reshape(-1,8))
df.columns = ['1st_day', '2nd_day', '3rd_day','4th_day', '5th_day', '6th_day','7th_day','8th_day_pred']
```

pandas-datareaderを用いてWebへアクセス＋pandasのDataFrameの形でデータの取得が可能

from モジュール名 import クラス名（関数名や変数名）

fred, google や yahoo など

DataReader()の引数にnameにデータセット名（NIKKEI225）、data_sourceにデータソース名（fred）、start変数にデータ取得開始年月日、end変数にデータ取得終了年月日が入ります。

1950年1月1日の日経平均から2021年8月12日までのデータを取得

対数を取得

過去7日間のデータを基に8日目の株価が前日より上がるか下がるかを学習

list型のリスト（配列）に要素を追加したり別のリストを結合したりするには、リストのメソッドappend()を使用

NumPy配列ndarrayの形状を変換

```
#Delete N/A data
df=df.dropna()
df=df.reset_index(drop=True)

#Transfrom to boolian based on the increase/decrease from 1 day before
for i in range(1,len(df)):
    if df['8th_day_pred'][i]>0:
        df['8th_day_pred'][i]=1
    elif df['8th_day_pred'][i]<0:
        df['8th_day_pred'][i]=0

#Separate conditions and answer
x=np.array(df.drop(['8th_day_pred'],axis=1))
y=np.array(df['8th_day_pred'], dtype=np.int16)

#Separate train data and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0,shuffle=False)

#Set grid search condition
grid_param = [{"n_estimators": [50,100],
               "max_depth"   : [None,5],
              }]

#Set cross validation condition
kfold_cv = KFold(n_splits=5, shuffle=True)

#Set algorizm
clf = GridSearchCV(RandomForestClassifier(), grid_param, cv=kfold_cv)

#Machine learning
clf.fit(x_train, y_train)
print("Best parameter = ", clf.best_estimator_)
```

参考:
https://newtechnologylifestyle.net/rain_test_split%E9%96%A2%E6%95%B0%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%A6%E3%83%87%E3%83%BC%E3%82%BF%E3%82%92%E5%88%86%E5%89%B2%E3%81%99%E3%82%8B/

train_test_split関数の引数(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)

scikit-learnのtrain_test_split関数を使用してデータを分割

sklearnで交差検証をする時に使うKfold
n_split:データの分割数。つまりk。検定はここで指定した数値の回数行われます。
shuffle:Trueなら連続する数字でグループ分けせず、ランダムにデータを選択

GridSearchCVのモデルインスタンス作成
使用するモデル、最適化したいパラメータセット、交差検定の回数

学習実行をするには、clf.fit()を使用します。

```
#Prediction
y_pred = clf.predict(x_test)

#output
print(classification_report(y_pred, y_test))
print("Accuracy [%] = " , accuracy_score(y_test, y_pred))
print(datetime.datetime.now().strftime('%Y.%m.%d. %H:%M:%S'), "   Analysis have done")
```

学習がclf.fit()だったのに対し、テストはclf.predict()を使用します。

```
Best parameter =  RandomForestClassifier(max_depth=5)
              precision    recall  f1-score   support

           0       0.29      0.48      0.36      1161
           1       0.71      0.51      0.59      2828

    accuracy                           0.50      3989
   macro avg       0.50      0.50      0.48      3989
weighted avg       0.58      0.50      0.53      3989

Accuracy [%] =  0.5026322386563048
2021.08.12. 17:23:43    Analysis have done
```

評価精度

正解率(Accuracy):正解率とは、予測結果全体がどれくらい真の値と一致しているかを表す指標