

Tutorial: How to Parse Figure Skating Score PDFs

Kyle Wang

The goal of this tutorial is to document the steps I took to download and parse 2022 Olympic Figure Skating scoresheets from the following website: <https://results.isu.org/results/season2122/owg2022/>. We will be using Python. The important packages needed for this are pandas, numpy, os, requests, re, and pdfplumber.

Taking a brief look at this page, one will notice that score sheets are referenced to by hyperlinks and fortunately, scoresheets are the only pdf file references that have the word "judge" in their name.

The general steps for pulling this data will be as follows then:

1. Pull the entire HTML content of the webpage and parse it for all the scoresheet file names, i.e. pdf files with the word "judge" in their name.
2. Send GET requests to download the pdf files to a newly created data directory. Use a Python package like pdfplumber to parse the pdfs for their entire raw text.
3. Use regex expressions to find important data features that we are interested in saving.
4. Merge all the data into one dataframe and save to a csv.

Step 1

We are given the website link: <https://results.isu.org/results/season2122/owg2022/>. Use requests to pull the html content of the page and then use the following regex pattern to find all the .pdf files linked to on the page:

```
regex_pattern = r'href=( [^>]+\.pdf)'
```

The above is finding all instances of strings that begin with "href=" and end with ".pdf" where any character inbetween is not whitespace and not an ending HTML bracket. Then, given list of all such strings, filter for all those that have the substring "judge".

Step 2

With all the important data links, we may now use requests again to download all the desired scoring sheet files. In my analysis, I am only interested in the singles competitions so one may also filter all the pdf links above to only those that include the substring "singles".

After downloading the files, use pdfplumber to grab the raw text strings, line by line.

```

pages_text = []
with pdfplumber.open(PDF_PATH) as pdf:
    for page in pdf.pages:
        text = page.extract_text()
        if text:
            pages_text.append(text)

full_text = "\n".join(pages_text)

```

Step 3

The general idea here is as follows. Now that we have the raw text of the pdfs, we can do anything we want. Looking at these pdfs, we notice that each block of a skater's scores and info begin with the same header.

```

# regex pattern for finding each skater
skater_header_pattern = re.compile(
    r"""
    ^(\d+)\s+                      # 1 rank
    (.+)\s+                         # 2 name
    ([A-Z]{3})\s+                   # 3 NOC code
    (\d+)\s+                         # 4 starting num
    (\d+\.\d{2})\s+                  # 5 total segment score
    (\d+\.\d{2})\s+                  # 6 total element score
    (\d+\.\d{2})\s+                  # 7 total program score
    (-?\d+\.\d{2})\$                # 8 total deductions
    """,
    re.VERBOSE | re.MULTILINE
)

```

Thus, our algorithm will be to find all instances of these headers in the text file and then store the entire substring between each header. This substring should contain all the text related to previous header's scores so we will store each (header, substring) pair as a tuple.

```

# find all skater header matches
matches = list(skater_header_pattern.finditer(full_text))

# for each skater header match, get the complete text between next match
skater_blocks = []
for i, m in enumerate(matches):
    start = m.start()
    end = matches[i + 1].start() if i + 1 < len(matches) else
len(full_text)
    skater_blocks.append((m, full_text[start:end]))

```

Looping through these tuples, we then use regex patterns on this substring to find the specific element and program component scores for a particular skater.

```
# regex pattern for finding each skater's element
element_pattern = re.compile(
    r"""
    ^\s*(\d+)\s+                      # 1 element number
    ([A-Za-z0-9!*<>q]+)\s+          # 2 element code
    (?:(\S+)\s+)?                     # 3 optional info column (x, q, !, etc.)
    ([\d.]+)\s+                       # 4 base value
    (?:\b(x)\b\s+)?                  # 5 optional extra points column (x)
    ([\-\d.]+)\s+                      # 6 GOE
    ((?:(?:-?\d+)|-)(?:\s+(?:-?\d+)|-)){8}\s+  # 7 judges scores
    ([\d.]+)$                         # 8 final score
    """,
    re.VERBOSE | re.MULTILINE
)

# regex pattern for finding each skater's program component
program_components_pattern = re.compile(
    r"""

    ^(Skating\S+Skills|Transitions|Performance|Composition|Interpretation\S+of
    \s+the\S+Music)\s+  # 1 component
    (\d+\.\d{2})\s+          # 2 factor
    ((?:\d+\.\d{2}\s+){9})   # 3 judge scores
    (\d+\.\d{2})$            # 4 final score
    """,
    re.VERBOSE | re.MULTILINE
)

# loop through all pairs of (header, substring)
for header, block in skater_blocks:
    ...

```

Step 4

You may choose many different ways of parsing the regex matches that you get above and putting that data into a dataframe. The way I went about it was to store two dataframes, an `element_df` and a `program_df`, to hold the different elements and program components respectively. Now, we can simply concat these two dataframes together and allow the element specific columns, such as `base value`, to identify which row came from which dataframe.

Step 5

Now, loop through all scoring pdfs and do steps 2-4 and then, simply concat them all together. You will likely want to create a helper column such as `isMens` or `isShort` to denote which specific event a scoring

sheet came from.