1. Name & CCIDs

- Yonael Bekele - yonael
- Michael Lin - zichun3

---

2. A list of all the resources used

- https://www.regextester.com/ - To test regex expressions. Also had a couple useful regex expressions premade i.e \d+(\%|\s\bpercent\b)(.*?) to get percentages
- https://regexr.com/ - To test regex
- https://dev.to/catherinecodes/a-regex-cheatsheet-for-all-those-regex-haters-and-lovers--2cj1
- https://stackoverflow.com/a/23794010

---

3. Execution Instructions

Setup

```
 # Setup python virtual environment
$ virtualenv venv --python=python3
$ source venv/bin/activate

# Install python dependencies
$ pip install -r requirements.txt
```

Run

Simply run `python main.py`, output `TSV` files are saved into `output/` folder.

By default, the program assume wiki files are under the `data` directory, and write output file to `output` directory. However, you may change it if you want, see below for advance usage.

```
usage: main.py [-h] [--input INPUT] [--output OUTPUT]

Extract relations from wiki files.

optional arguments:
  -h, --help       show this help message and exit
  --input INPUT    Provide path to directory of input wiki files
  --output OUTPUT  Provide path to save output TSV files
```

More

Run the command below to check how many facts we extracted are missing compare to the sample data set.

The script added a bunch of special handling for things like `musicComposer -> music`, `producer -> producers`, but still requires some manual work to double check.

> `$ python check.py > coverage_report.txt`

Notes

- For evidence, we try to keep it as short as possible. e.g, `plainlist` only show the first line which contains the `predicate`, but not the objects. We could've gotten the whole line including all the plainlist items but DENILSON BARBOSA SAID IT'S OKAY.

---

4. Design Decisions

- We have an Extract class that goes through the file and extracts information while it cleans the text removing comments
- We first breakdown the text by matching open parantheses/brackets to closed brackets and processing as a token. (*preprocess* & *balanced*)
- Most of the information we need to cover the sample cases are provided in the Infobox. So, we search for Infobox materials in our tokens and have two cases of processing: Plainlist or not. (*get_relation* handles these two cases)
- We then look at the Categories and have 3 cases: Winner, Type or neither which results in Category. We found this an efficient way to get winner awards that is consistent throughout most cases. If Film is present we tokenize the string and find the type film as well as the other string (strictly alphabetical) which is a type that describes the movie. As well as finding other relations using the categories. (*category_relation*)
- We then look at tokens with "Rotten Tomatoes", and find the approval rating. (*approval_relation*)
- We then do another tokenize method using NLTK's sent_tokenize to find new patterns of relations
- We search for more unique patterns using the regex *magic list (_get_relations_from_text)*
- We verified our coverage using a script to do a diff between our output and the data provided