Bookworm

Social networks from novels

PyData NYC 2017

Harrison Pim

What’s in this talk?

- No neural networks
- No useful stuff for work

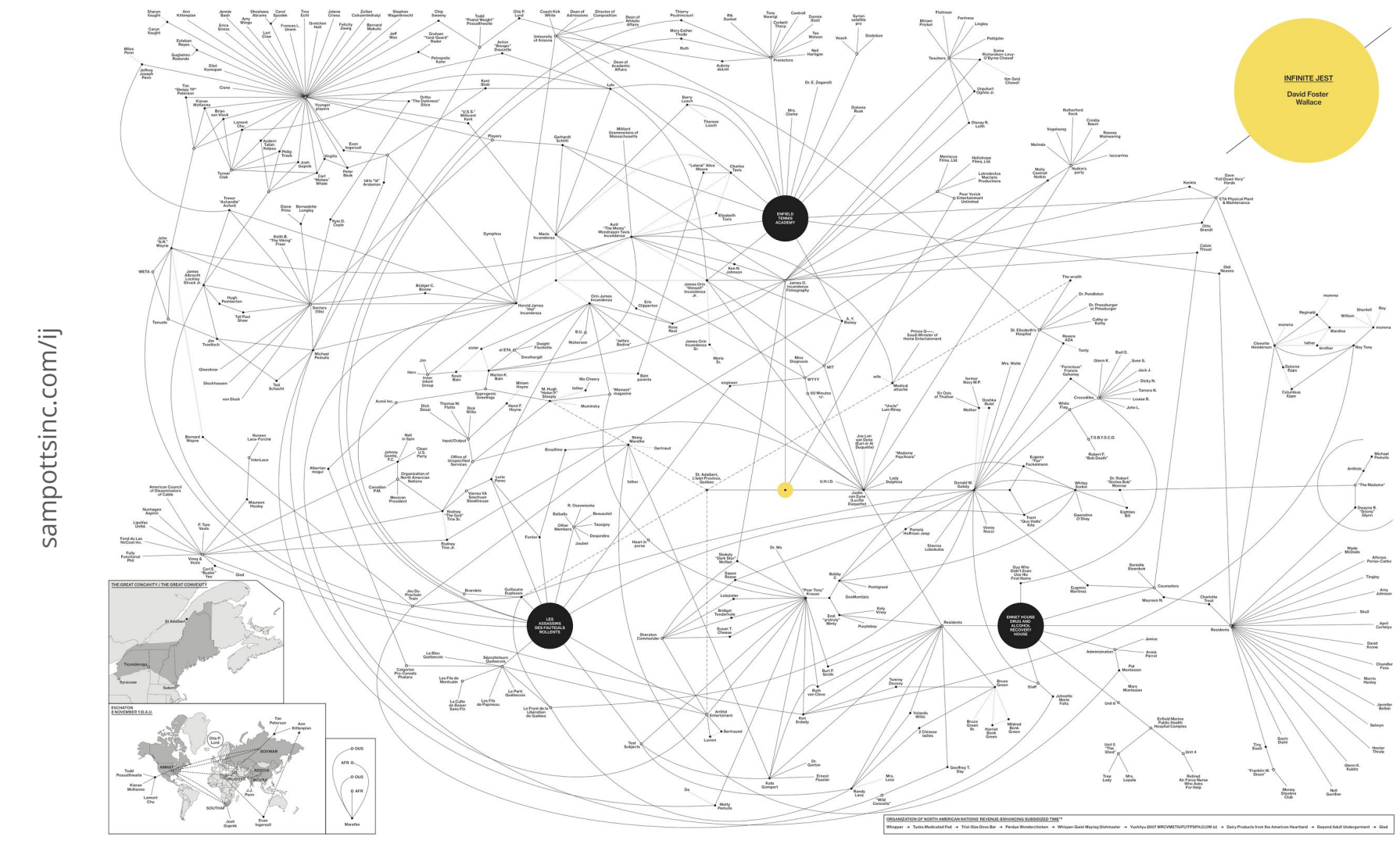
Who am I?

Harrison Pim

Twitter @hmpim
Github harrisonpim (.github.io)

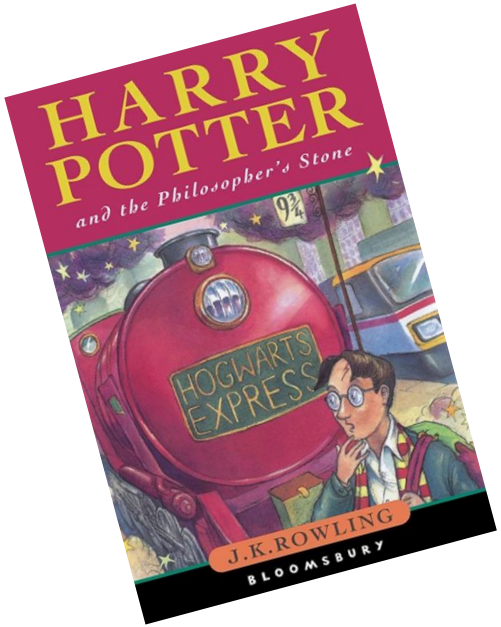
Did physics, doing data science

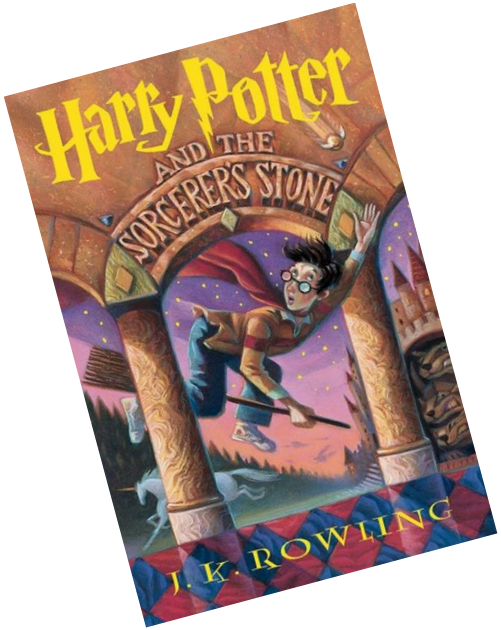
Infinite Jest

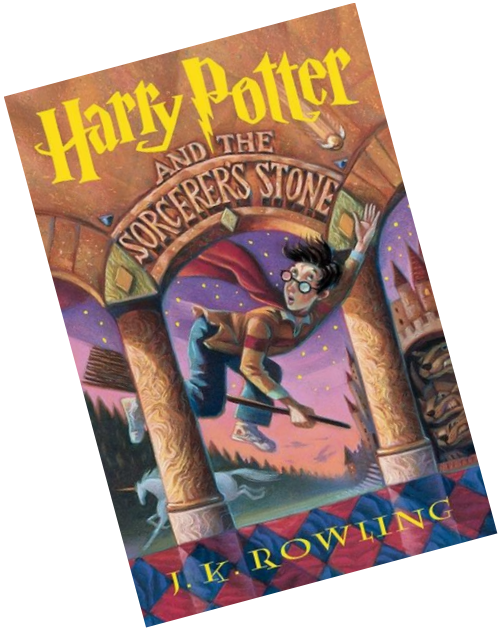


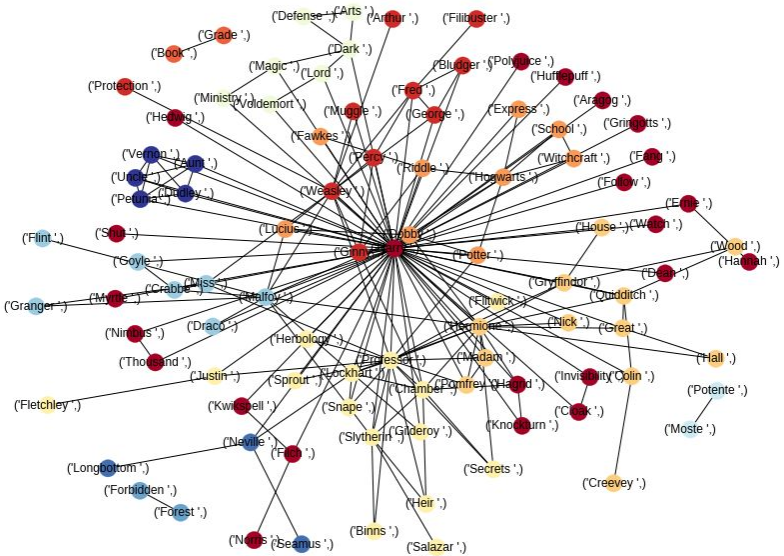
sampottsinc.com/ij

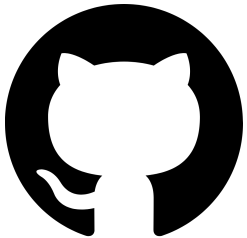
Can we duplicate this thing programmatically?


HARRY
POTTER
and the Philosopher's Stone
9¾
HOGWARTS
EXPRESS
J.K.ROWLING
BLOOMSBURY


Harry Potter
AND THE
SORCERER'S STONE
J. K. ROWLING


Harry Potter
AND THE
SORCERER'S STONE
J. K. ROWLING


('Arts ',)
('Defense ',)
('Arthur ',)
('Filibuster ',)
('Grade ',)
('Book ',)
('Dark ',)
('Magic ',)
('Lord ',)
('Bludger ',)
('Polyjuice ',)
('Hufflepuff ',)
('Protection ',)
('Ministry ',)
('Voldemort ',)
('Fred ',)
('Arabog ',)
('Hedwig ',)
('Muggle ',)
('George ',)
('Express ',)
('School ',)
('Gringotts ',)
('Fawkes ',)
('Vernon ',)
('Aunt ',)
('Uncle ',)
('Petunia ',)
('Dursley ',)
('Percy ',)
('Riddle ',)
('Witchcraft ',)
('Ford ',)
('Hogwarts ',)
('Follow ',)
('Weasley ',)
('Eight ',)
('Flint ',)
('Snape ',)
('Lucius ',)
('House ',)
('Watch ',)
('Wood ',)
('Goyle ',)
('Potter ',)
('Hannah ',)
('Miss ',)
('Gryffindor ',)
('Dean ',)
('Crabbe ',)
('Granger ',)
('Malfoy ',)
('Quidditch ',)
('Flitwick ',)
('Nick ',)
('Great ',)
('Draco ',)
('Hermione ',)
('Nimbus ',)
('Herbology ',)
('Hall ',)
('Thorstand ',)
('Professor ',)
('Madam ',)
('Justin ',)
('Sprout ',)
('Lockhart ',)
('Invisibility ',)
('Colin ',)
('Pomfrey ',)
('Hagrid ',)
('Potente ',)
('Fletchley ',)
('Kwikspell ',)
('Snape ',)
('Chamber ',)
('Cloak ',)
('Knockturn ',)
('Gilderoy ',)
('Moste ',)
('Neville ',)
('Slytherin ',)
('Fleur ',)
('Secrets ',)
('Longbottom ',)
('Creevey ',)
('Forbidden ',)
('Heir ',)
('Forest ',)
('Binns ',)
('Nott ',)
('Seamus ',)
('Salazar ',)



harrisonpim/bookworm

Finding connections between character names

Scraping text and taking names

Getting Book Text

Super simple

```
def load_book(book_path, lower=True):
    '''
    Reads in a novel from a .txt file, and returns it in (optionally
    lowercased) string form

    Parameters
    ----------
    book_path : string (required)
        path to txt file containing full text of book to be analysed
    lower : bool (optional)
        If True, the returned string will be lowercased;
        If False, the returned string will retain its original case formatting.

    Returns
    -------
    book : string
        book in string form
    '''
    with open(book_path) as f:
        book = f.read()
    if lower:
        book = book.lower()
    return book
```

Explicit Names

Easiest approach is to begin with an explicit list of names. `load_characters()` pulls in names from `.csv` file

```
def load_characters(charaters_path):
    '''
    Reads in a .csv file of character names

    Parameters
    ----------
    charaters_path : string (required)
        path to csv file containing full list of characters to be examined.
        Each character should take up one line of the file. If the character is
        referred to by multiple names, nicknames or sub-names within their
        full name, these should be split by commas, eg:
        Harry, Potter
        Lord, Voldemort, You-Know-Who
        Giant Squid

    Returns
    -------
    characters : list
        list of tuples naming characters in text
    '''
    with open(charaters_path) as f:
        reader = csv.reader(f)
        characters = [tuple(name.lower()+' ' for name in row) for row in reader]
    return characters
```

Explicit Names

Easiest approach is to begin with an explicit list of names. `load_characters()` pulls in names from `.csv` file

However, we can't call this *real* automation...

```
def load_characters(charaters_path):
    '''
    Reads in a .csv file of character names

    Parameters
    ----------
    charaters_path : string (required)
        path to csv file containing full list of characters to be examined.
        Each character should take up one line of the file. If the character is
        referred to by multiple names, nicknames or sub-names within their
        full name, these should be split by commas, eg:
        Harry, Potter
        Lord, Voldemort, You-Know-Who
        Giant Squid

    Returns
    -------
    characters : list
        list of tuples naming characters in text
    '''
    with open(charaters_path) as f:
        reader = csv.reader(f)
        characters = [tuple(name.lower()+' ' for name in row) for row in reader]
    return characters
```

Implicit Names

We can also generate an implicit list of plausible character names from the text of the book itself

```
remove_punctuation = lambda s: s.translate(str.maketrans('', '', string.punctuation+'’'))


def extract_character_names(book):
    '''
    Automatically extracts lists of plausible character names from a book

    Parameters
    ----------
    book : string (required)
        book in string form (with original upper/lowercasing intact)

    Returns
    -------
    characters : list
        list of plausible character names
    '''

    nlp = spacy.load('en')
    stopwords = nltk.corpus.stopwords.words('english')

    words = [remove_punctuation(p) for p in book.split()]
    unique_words = list(set(words))

    characters = [word.text for word in nlp(' '.join(unique_words)) if word.pos_ == 'PROPN']
    characters = [c for c in characters if len(c) > 3]
    characters = [c for c in characters if c.istitle()]
    characters = [c for c in characters if not (c[-1] == 's' and c[:-1] in characters)]
    characters = list(set([c.title() for c in [c.lower() for c in characters]]) - set(stopwords))

    return [tuple([c + ' ']) for c in set(characters)]
```

Book Splitting

```
def get_sentence_sequences(book):
    '''
    Splits a book into its constituent sentences

    Parameters
    ----------
    book : string (required)
        book in string form

    Returns
    -------
    sentences : list
        list of strings, where each string is a sentence in
        the novel as interpreted by NLTK's tokenize() function
    '''
    det = nltk.data.load('tokenizers/punkt/english.pickle')
    sentences = det.tokenize(book)
    return sentences
```

```
def get_word_sequences(book, n=50):
    '''
    Takes a book and splits it into its constituent words,
    returning a list of substrings which comprise the book,
    whose lengths are determined by a set number of words
    (default = 50).

    Parameters
    ----------
    book : string (required)
        book in string form
    n : int (optional)
        number of words to be contained in each returned
        sequence (default = 50)

    Returns
    -------
    sequences : list
        list of strings, where each string is a list of n
        words as interpreted by NLTK's word_tokenize()
        function.
    '''
    book_words = word_tokenize(book)
    sequences = [' '.join(book_words[i: i+n])
                 for i in range(0, len(book_words), n)]
    return sequences
```

```
def get_character_sequences(book, n=200):
    '''
    Takes a book and splits it into a list of substrings of
    length n (default = 200).

    Parameters
    ----------
    book : string (required)
        book in string form
    n : int (optional)
        number of characters to be contained in each returned
        Sequence (default = 200)

    Returns
    -------
    sequences : list
        list of strings comprising the book, where each string
        is of length n.
    '''
    return [''.join(book[i: i+n]) for i in range(0, len(book),
n)]
```

Book Splitting

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a haircut.

```
def get_word_sequences(book, n=50):
    '''
    Takes a book and splits it into its constituent words,
    returning a list of substrings which comprise the book,
    whose lengths are determined by a set number of words
    (default = 50).

    Parameters
    ----------
    book : string (required)
        book in string form
    n : int (optional)
        number of words to be contained in each returned
        sequence (default = 50)

    Returns
    -------
    sequences : list
        list of strings, where each string is a list of n
        words as interpreted by NLTK's word_tokenize()
        function.
    '''
    book_words = word_tokenize(book)
    sequences = [' '.join(book_words[i: i+n])
                 for i in range(0, len(book_words), n)]
    return sequences
```

```
def get_character_sequences(book, n=200):
    '''
    Takes a book and splits it into a list of substrings of
    length n (default = 200).

    Parameters
    ----------
    book : string (required)
        book in string form
    n : int (optional)
        number of characters to be contained in each returned
        Sequence (default = 200)

    Returns
    -------
    sequences : list
        list of strings comprising the book, where each string
        is of length n.
    '''
    return [''.join(book[i: i+n]) for i in range(0, len(book),
n)]
```

Book Splitting

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a haircut.

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a

```
def get_character_sequences(book, n=200):
    '''
    Takes a book and splits it into a list of substrings of
    length n (default = 200).

    Parameters
    ----------
    book : string (required)
        book in string form
    n : int (optional)
        number of characters to be contained in each returned
        Sequence (default = 200)

    Returns
    -------
    sequences : list
        list of strings comprising the book, where each string
        is of length n.
    '''
    return [''.join(book[i: i+n]) for i in range(0, len(book),
n)]
```

Book Splitting

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a haircut.

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a hai

Finding Connections

```
def find_connections(sequences, characters):
    '''
    Takes a novel and its character list and counts instances of each character
    in each sequence.

    Parameters
    ----------
    sequences : list (required)
        list of substrings representing the novel to be analysed
    characters : list (required)
        list of character names (as tuples)

    Returns
    -------
    df : pandas.DataFrame
        columns = character names
        indexes = sequences
        values  = counts of instances of character name in sequence
    '''
    df = pd.DataFrame({str(c): {s: 0 for s in sequences} for c in characters})

    for sequence in sequences:
        for character in characters:
            if any(name in sequence for name in character):
                df[str(character)][sequence] += 1
    return df
```

Vernon
Petunia
Dudley
Lily
James
Dumbledore
Harry
Voldemort
...

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a haircut.

Finding Connections

```
def find_connections(sequences, characters):
    '''
    Takes a novel and its character list and counts instances of each character
    in each sequence.

    Parameters
    ----------
    sequences : list (required)
        list of substrings representing the novel to be analysed
    characters : list (required)
        list of character names (as tuples)

    Returns
    -------
    df : pandas.DataFrame
        columns = character names
        indexes = sequences
        values  = counts of instances of character name in sequence
    '''
    df = pd.DataFrame({str(c): {s: 0 for s in sequences} for c in characters})

    for sequence in sequences:
        for character in characters:
            if any(name in sequence for name in character):
                df[str(character)][sequence] += 1
    return df
```

Vernon
Petunia
Dudley
Lily
James
Dumbledore
Harry
Voldemort
...

About once a week, Uncle Vernon looked over the top of his newspaper and shouted that Harry needed a haircut.

Finding Connections

	('bloody baron ',)	('tufty ',)	('voldemort ', ' lord ', ' you-know-who ')	('diggle ',)	('mrs evans ',)	('narcissa ', ' mrs. malfoy ')	('nearly headless nick ',)	('filch ',)	('baruffio ',)	('pansy parkinson ',)
"You'll understand later, Neville," said Ron as they stepped over him and pulled on the invisibility cloak.	0	0	0	0	0	0	0	0	0	0
On top of the stool she put a pointed wizard's hat.	0	0	0	0	0	0	0	0	0	0
There was somethin' goin' on that night he hadn't counted on -- I dunno what it was, no one does -- but somethin' about you stumped him, all right."	0	0	0	0	0	0	0	0	0	0
Send me an answer as soon as possible.	0	0	0	0	0	0	0	0	0	0
"Potter, did she say?"	0	0	0	0	0	0	0	0	0	0
Hagrid could be seen from the upstairs windows defrosting broomsticks on the Quidditch field, bundled up in a long moleskin overcoat, rabbit fur gloves, and enormous beaverskin boots.	0	0	0	0	0	0	0	0	0	0
The rest of the team landed next to George to complain, too.	0	0	0	0	0	0	0	0	0	0
shouted Hagrid, who had a boat to himself.	0	0	0	0	0	0	0	0	0	0
"Professor Dumbledore left ten minutes ago," she said coldly.	0	0	0	0	0	0	0	0	0	0
Fred and George have lost loads of points in all the time they've been here, and people still like them."	0	0	0	0	0	0	0	0	0	0

Cooccurrence / Adjacency

```
def calculate_cooccurence(df):
    '''
    Uses the dot product to calculate the number of times two characters appear
    in the same sequences. This is the core of the bookworm graph.

    Parameters
    ----------
    df : pandas.DataFrame (required)
        columns = character names
        indexes = sequences
        values  = counts of instances of character name in sequence

    Returns
    -------
    cooccurence : pandas.DataFrame
        columns = character names
        indexes = character names
        values  = counts of character name cooccurences in all sequences
    '''
    characters = df.columns.values
    cooccurence = df.values.T.dot(df.values)
    np.fill_diagonal(cooccurence, 0)
    cooccurence = pd.DataFrame(cooccurence, columns=characters, index=characters)
    return cooccurence
```

$$C = D^{\dagger} \square D$$

$$C_{i=j} = 0$$

Cooccurrence / Adjacency

```
def calculate_cooccurence(df):
    '''
    Uses the dot product to calculate the number of times two characters appear
    in the same sequences. This is the core of the bookworm graph.

    Parameters
    ----------
    df : pandas.DataFrame (required)
        columns = character names
        indexes = sequences
        values  = counts of instances of character name in sequence

    Returns
    -------
    cooccurence : pandas.DataFrame
        columns = character names
        indexes = character names
        values  = counts of character name cooccurences in all sequences
    '''
    characters = df.columns.values
    cooccurence = df.values.T.dot(df.values)
    np.fill_diagonal(cooccurence, 0)
    cooccurence = pd.DataFrame(cooccurence, columns=characters, index=characters)
    return cooccurence
```

	('emeric the evil ',)	('hagrid ', ' rubeus ')	('hooch ',)	('mandy brocklehurst ',)	('goyle ',)	('tufty ',)	('marge ', ' aunt ')	('bertie bott ',)	('perenelle flamel ',)	('narcissa ', ' mrs. malfoy ')
('elfric the eager ',)	0	0	0	0	0	0	0	0	0	0
('bloody baron ',)	0	0	0	0	0	0	0	0	0	0
('theodore nott ',)	0	0	0	0	0	0	0	0	0	0
('ollivander ',)	0	1	0	0	0	0	0	0	0	0
('ron ', ' weasley ')	0	6	0	0	1	0	0	0	0	0
('mcgonagall ', ' minerva ')	0	3	0	0	0	0	0	0	0	0
('madam pince ',)	0	0	0	0	0	0	0	0	0	0
('charlie ',)	0	1	0	0	0	0	0	0	0	0
('seamus finnigan ',)	0	0	0	0	0	0	0	0	0	0
('vindictus viridian ',)	0	0	0	0	0	0	0	0	0	0

NetworkX

```
def get_interaction_df(cooccurence, threshold=0):
    '''
    Produces an dataframe of interactions between characters using the
    cooccurence matrix of those characters. The return format is directly
    analysable by networkx in the construction of a graph of characters.

    Parameters
    ----------
    cooccurence : pandas.DataFrame (required)
        columns = character names
        indexes = character names
        values  = counts of character name cooccurences in all sequences
    threshold : int (optional)
        The minimum character interaction strength needed to be included in the
        returned interaction_df

    Returns
    -------
    interaction_df : pandas.DataFrame
        DataFrame enumerating the strength of interactions between charcters.
        source = character one
        target = character two
        value = strength of interaction between character one and character two
    '''
    rows, columns = np.where(np.triu(cooccurence.values, 1) > threshold)

    return pd.DataFrame(np.column_stack([cooccurence.index[rows],
                                         cooccurence.columns[columns],
                                         cooccurence.values[rows, columns]]),
                        columns=['source', 'target', 'value'])
```

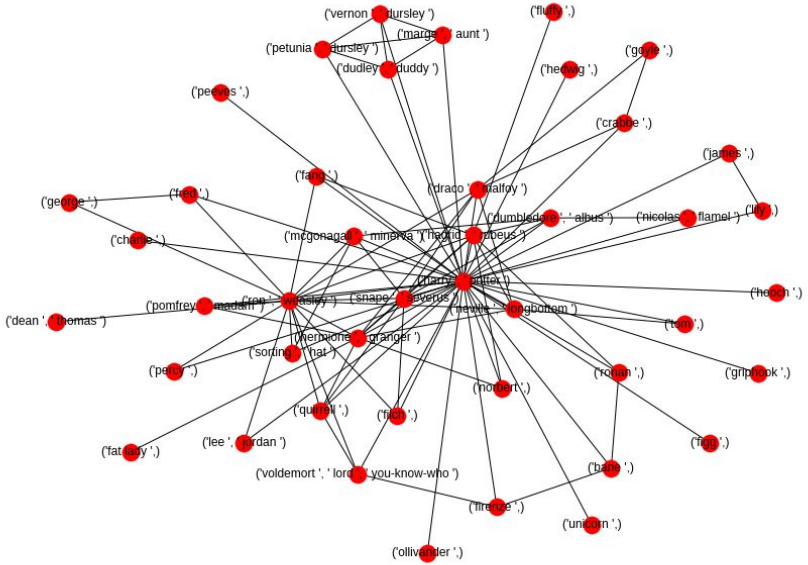
	source	target	value
7430	('ron ', ' weasley ')	('harry ', ' potter ')	84
730	('harry ', ' potter ')	('ron ', ' weasley ')	84
1	('vernon ', ' dursley ')	('petunia ', ' dursley ')	44
749	('harry ', ' potter ')	('hermione ', ' granger ')	44
135	('petunia ', ' dursley ')	('vernon ', ' dursley ')	44
9995	('hermione ', ' granger ')	('harry ', ' potter ')	44
1760	('hagrid ', ' rubeus ')	('harry ', ' potter ')	41
688	('harry ', ' potter ')	('hagrid ', ' rubeus ')	41
7499	('ron ', ' weasley ')	('hermione ', ' granger ')	40
10045	('hermione ', ' granger ')	('ron ', ' weasley ')	40
150	('petunia ', ' dursley ')	('marge ', ' aunt ')	32
2026	('marge ', ' aunt ')	('petunia ', ' dursley ')	32
778	('harry ', ' potter ')	('snape ', ' severus ')	20
13910	('snape ', ' severus ')	('harry ', ' potter ')	20
275	('dudley ', ' duddy ')	('harry ', ' potter ')	17
677	('harry ', ' potter ')	('dudley ', ' duddy ')	17

Visualising the explicit-name network

```
G = nx.from_pandas_dataframe(interaction_df,
                             source='source',
                             target='target')

nx.draw_spring(G, with_labels=True)
```

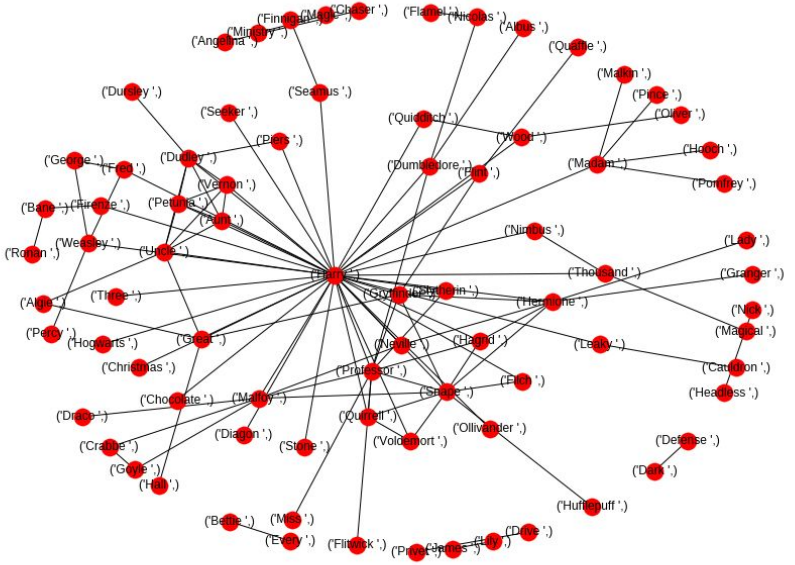
Visualising the explicit-name network

```
G = nx.from_pandas_dataframe(interaction_df,
                             source='source',
                             target='target')

nx.draw_spring(G, with_labels=True)
```



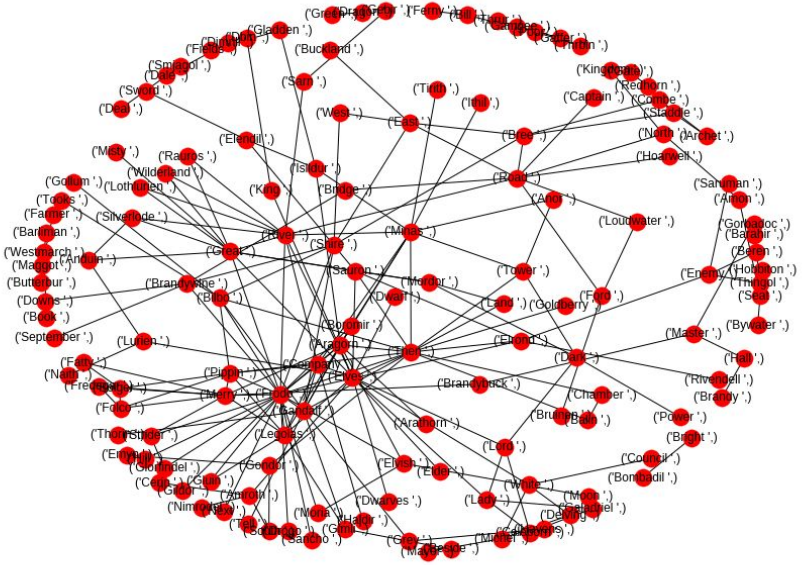
Visualising the implicit-name network

```
G = nx.from_pandas_dataframe(interaction_df,
                             source='source',
                             target='target')

nx.draw_spring(G, with_labels=True)
```



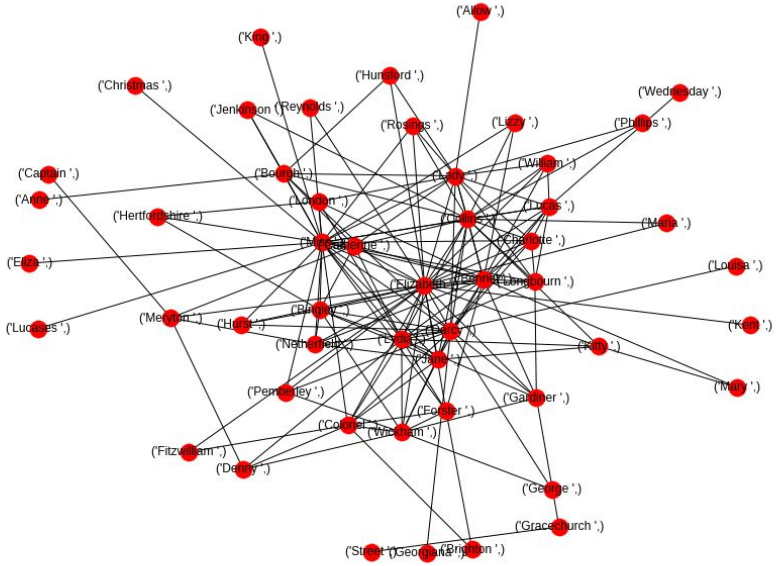
Visualising the implicit-name network

```
G = nx.from_pandas_dataframe(interaction_df,
                             source='source',
                             target='target')

nx.draw_spring(G, with_labels=True)
```



Visualising the implicit-name network

```
G = nx.from_pandas_dataframe(interaction_df,
                             source='source',
                             target='target')

nx.draw_spring(G, with_labels=True)
```



Character Relationships

```
print_five_closest(('harry ', ' potter '))
```

```
('harry ', ' potter ')
----------------------
('ron ', ' weasley ')
('hermione ', ' granger ')
('hagrid ', ' rubeus ')
('snape ', ' severus ')
('dudley ', ' duddy ')
```

Character Importance

```
pd.Series(nx.pagerank(G)).sort_values(ascending=False)
```

```
('harry ', ' potter ')                  0.162006
('ron ', ' weasley ')                   0.077277
('snape ', ' severus ')                 0.053273
('draco ', ' malfoy ')                  0.043361
('hermione ', ' granger ')              0.042616
('mcgonagall ', ' minerva ')            0.039662
('hagrid ', ' rubeus ')                 0.034685
```

Character Importance

```
pd.Series(nx.hits(G)[0]).sort_values(ascending=False)
```

```
('harry ', ' potter ')                  1.203997e-01
('ron ', ' weasley ')                   7.506373e-02
('snape ', ' severus ')                 6.839702e-02
('hermione ', ' granger ')              5.501891e-02
('hagrid ', ' rubeus ')                 5.256913e-02
('draco ', ' malfoy ')                  4.849397e-02
('neville ', ' longbottom ')            4.455587e-02
```

Character Pathfinding (?)

```
nx.dijkstra_path(G,
                 source="('Hedwig ',)",
                 target="('Flamel ',)")
```

```
["('Hedwig ',)",
 "('Harry ',)",
 "('Dumbledore ',)",
 "('Nicolas ',)",
 "('Flamel ',)"]
```

Time / chronology

Novels Develop

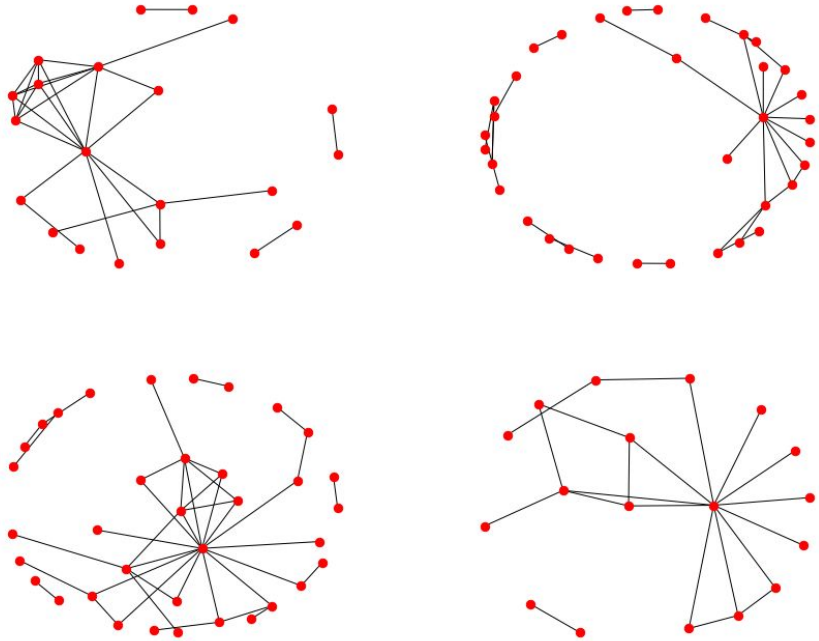
We can watch the progression of character relationships

Temporal Networks

```
chronological_network('data/raw/hp_philosophers_stone.txt',
                      n_sections=4,
                      cumulative=False)
```

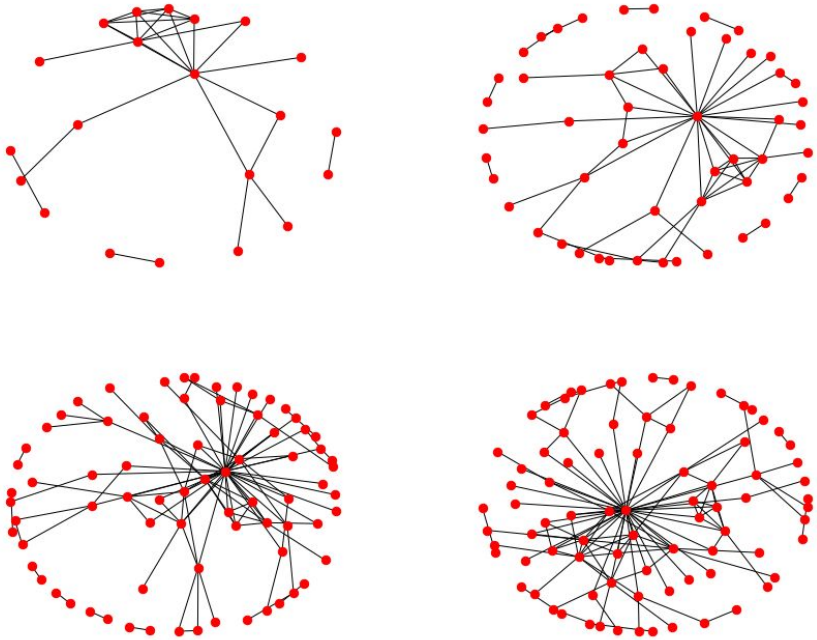
Temporal Networks

```
chronological_network('data/raw/hp_philosophers_stone.txt',
                      n_sections=4,
                      cumulative=False)
```



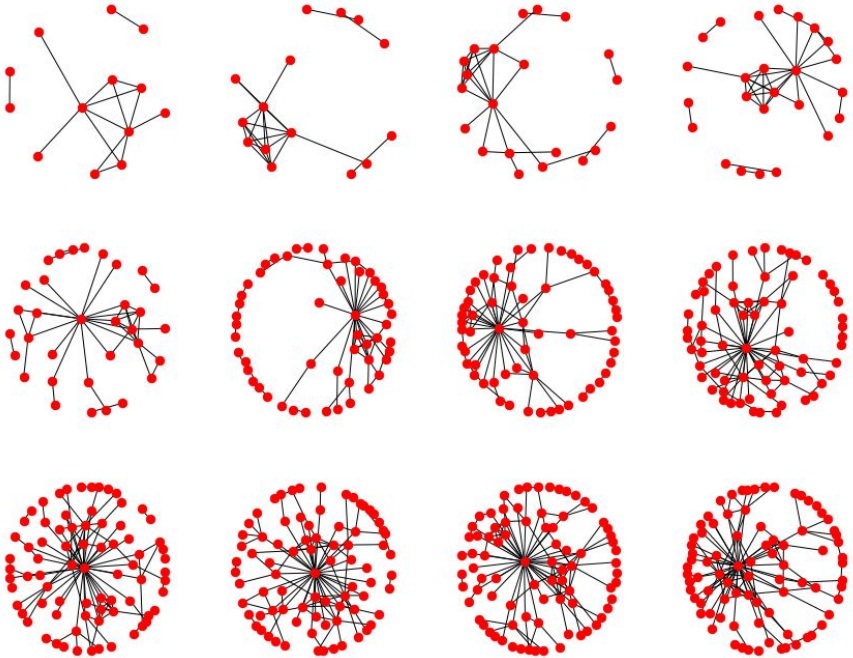
Temporal Networks

```
chronological_network('data/raw/hp_philosophers_stone.txt',
                      n_sections=4,
                      cumulative=True)
```



Temporal Networks

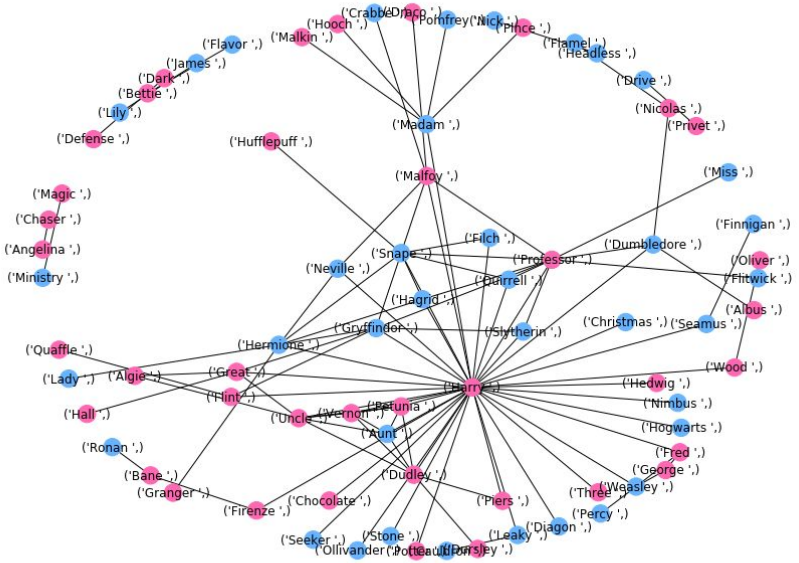
```
chronological_network('data/raw/hp_philosophers_stone.txt',
                      n_sections=12,
                      cumulative=True)
```



Gender and Stable Marriages

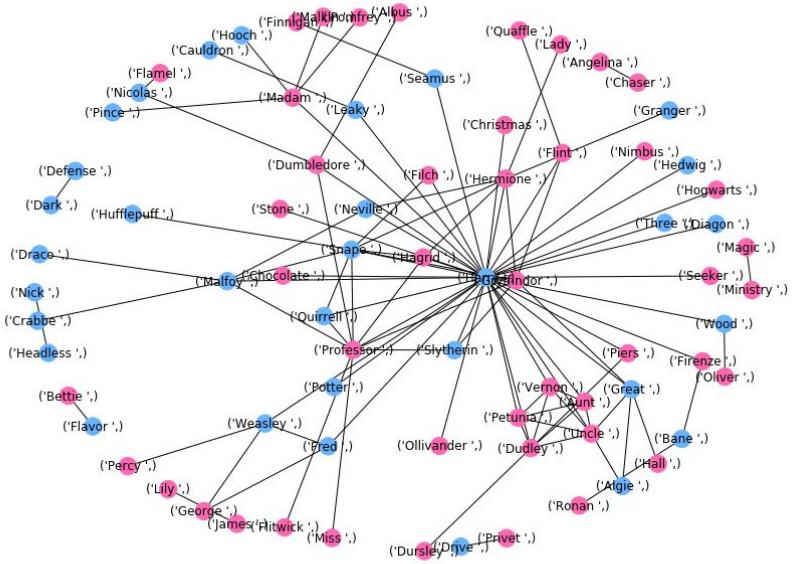
Gender

```
genders = {character: random.choice(['M', 'F']) for character in characters}

proposers, acceptors = get_gendered_preferences(interaction_df)
```



Gale-Shapley

```
genders = {character: random.choice(['M', 'F']) for character in characters}

proposers, acceptors = get_gendered_preferences(interaction_df)

marriages = gale_shapley(female_preferences, male_preferences)
```



Gale-Shapley

```
genders = {character: random.choice(['M', 'F']) for character in characters}

proposers, acceptors = get_gendered_preferences(interaction_df)

marriages = gale_shapley(female_preferences, male_preferences)

marriages["('Harry ',)"]
```

Gale-Shapley

```
genders = {character: random.choice(['M', 'F']) for character in characters}

proposers, acceptors = get_gendered_preferences(interaction_df)

marriages = gale_shapley(female_preferences, male_preferences)

marriages["('Harry ',)"]
```

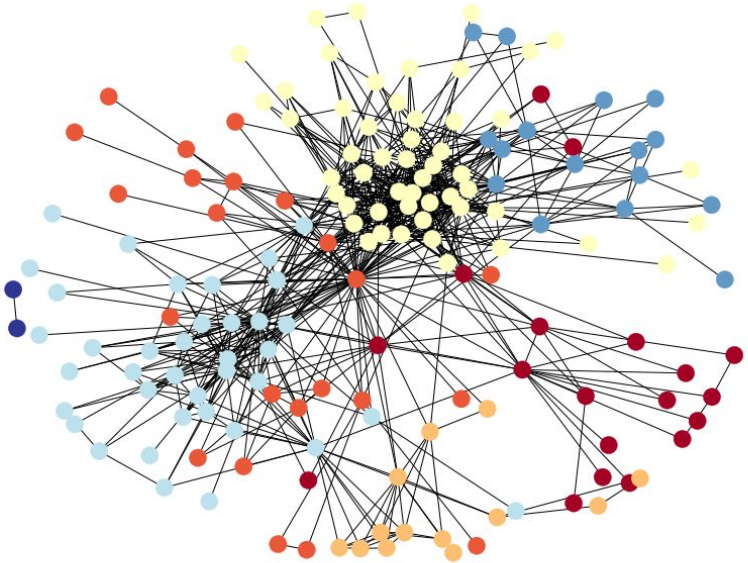
```
"('Hermione ',)"
```

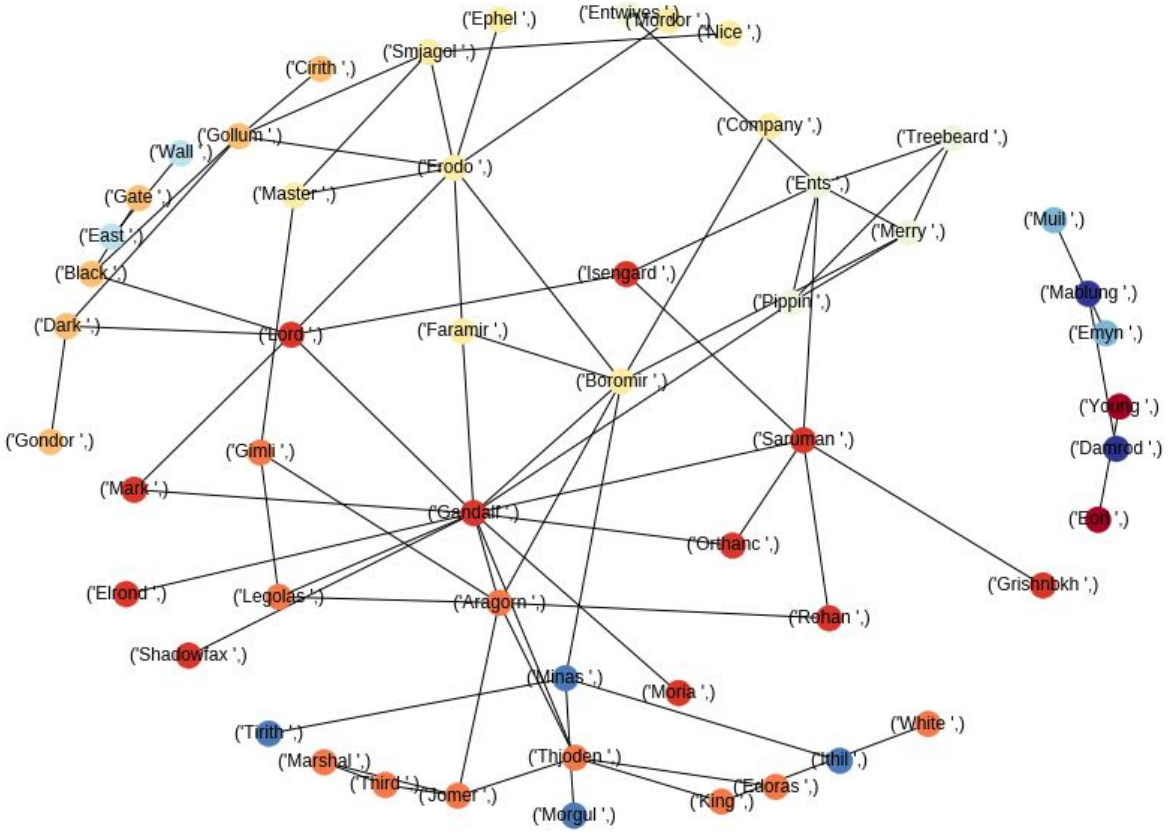
Community Detection

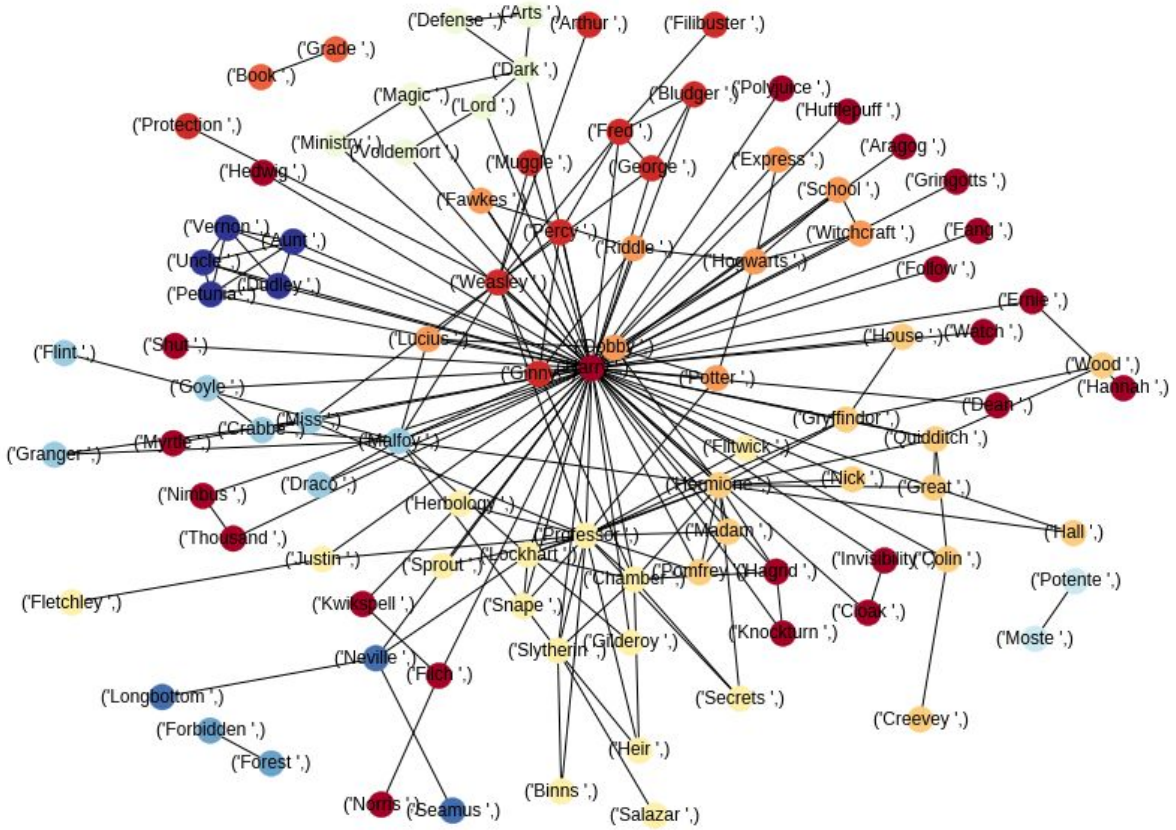
Character Communities/Cliques

Not going into detail here, but it's worth looking it up if you're interested!

Uses `python-louvain`, based on Blondel et al's implementation of Louvain Modularity [2008]



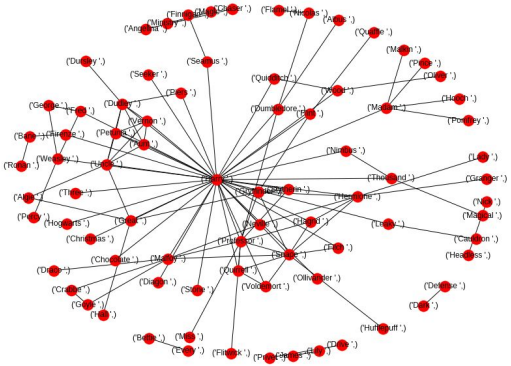

('Ephel ',)
('Entwives ',)
('Mordor ',)
('Nice ',)
('Smjagol ',)
('Cirith ',)
('Company ',)
('Treebeard ',)
('Gollum ',)
('Wall ',)
('Erodo ',)
('Ents ',)
('Gate ',)
('Master ',)
('Muil ',)
('East ',)
('Merry ',)
('Black ',)
('Isengard ',)
('Pippin ',)
('Mablung ',)
('Dark ',)
('Lord ',)
('Faramir ',)
('Emyn ',)
('Boromir ',)
('Young ',)
('Gondor ',)
('Gimli ',)
('Saruman ',)
('Damrod ',)
('Mark ',)
('Gandalf ',)
('Eon ',)
('Orthanc ',)
('Elrond ',)
('Legolas ',)
('Aragorn ',)
('Grishnokh ',)
('Rohan ',)
('Shadowfax ',)
('Minas ',)
('Moria ',)
('White ',)
('Turith ',)
('Marshal ',)
('Thjoden ',)
('Ithil ',)
('Third ',)
('Jomer ',)
('Edoras ',)
('King ',)
('Morgul ',)


('Defense ',)
('Arts ',)
('Arthur ',)
('Filibuster ',)
('Grade ',)
('Book ',)
('Dark ',)
('Magic ',)
('Lord ',)
('Bludger ',)
('Polyjuice ',)
('Hufflepuff ',)
('Protection ',)
('Ministry ',)
('Voldemort ',)
('Muggle ',)
('Fred ',)
('George ',)
('Express ',)
('Aragog ',)
('Hedwig ',)
('Fawkes ',)
('School ',)
('Gringotts ',)
('Vernon ',)
('Aunt ',)
('Uncle ',)
('Dudley ',)
('Petunia ',)
('Percy ',)
('Riddle ',)
('Hogwarts ',)
('Witchcraft ',)
('Fang ',)
('Follow ',)
('Weasley ',)
('Ernie ',)
('Lucius ',)
('Bobby ',)
('House ',)
('Watch ',)
('Flint ',)
('Shut ',)
('Ginny ',)
('Potter ',)
('Wood ',)
('Hannah ',)
('Goyle ',)
('Dean ',)
('Miss ',)
('Gryffindor ',)
('Crabbe ',)
('Myrtle ',)
('Malfoy ',)
('Quidditch ',)
('Granger ',)
('Flitwick ',)
('Nick ',)
('Great ',)
('Nimbus ',)
('Draco ',)
('Hermione ',)
('Herbology ',)
('Thousand ',)
('Professor ',)
('Madam ',)
('Hall ',)
('Justin ',)
('Sprout ',)
('Lockhart ',)
('Invisibility ',)
('Colin ',)
('Chamber ',)
('Pomfrey ',)
('Hagrid ',)
('Potente ',)
('Fletchley ',)
('Kwikspell ',)
('Snape ',)
('Cloak ',)
('Knockturn ',)
('Moste ',)
('Gilderoy ',)
('Neville ',)
('Slytherin ',)
('Filch ',)
('Secrets ',)
('Longbottom ',)
('Creevey ',)
('Forbidden ',)
('Heir ',)
('Forest ',)
('Binns ',)
('Norris ',)
('Seamus ',)
('Salazar ',)

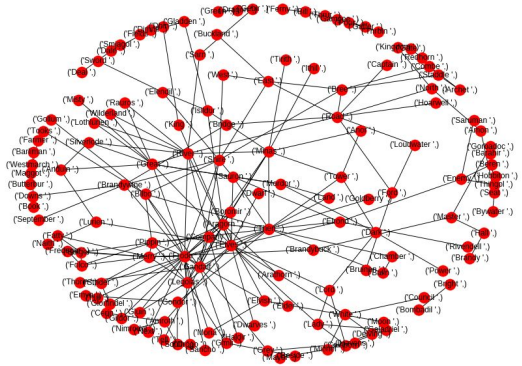
Graph Matching

Novel Similarity

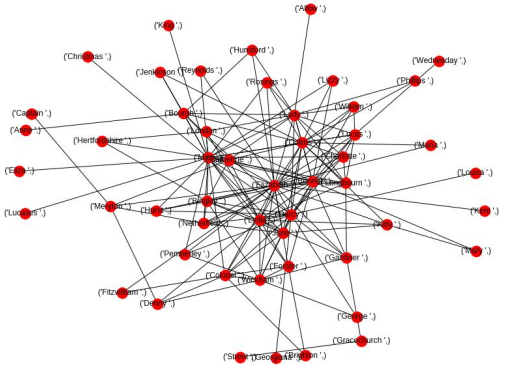
Harry Potter



Lord of The Rings



Pride and Prejudice



Novel Similarity

Given that we can enumerate our characters' similarity to one another, can we also compute the similarity of two arbitrary novels?

Again, not going into details here, but the answer is **yes**!

Uses Laplacian spectrum

Novel Similarity

	Harry Potter	Lord of The Rings	Pride and Prejudice
Harry Potter	0.000000	862.261406	1718.486502
Lord of The Rings	862.261406	0.000000	1745.118988
Pride and Prejudice	1718.486502	1745.118988	0.000000

Novel Similarity

	Harry Potter	Lord of The Rings	Pride and Prejudice
Harry Potter	0.000000	862.261406	1718.486502
Lord of The Rings	862.261406	0.000000	1745.118988
Pride and Prejudice	1718.486502	1745.118988	0.000000

We can build a network of novels!

We can build a network of novels!

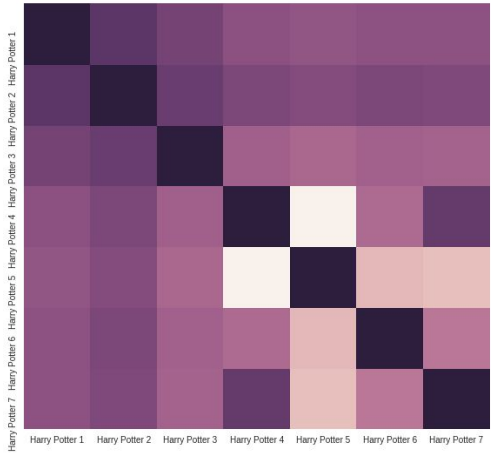
maybe...

We can build a network of novels!

still developing...

Novel typicality

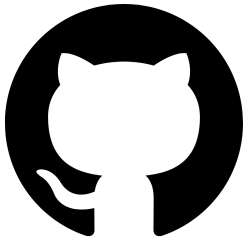
```
books = {'Harry Potter 1': hp1,
         'Harry Potter 2': hp2,
         'Harry Potter 3': hp3,
         'Harry Potter 4': hp4,
         'Harry Potter 5': hp5,
         'Harry Potter 6': hp6,
         'Harry Potter 7': hp7}

comp = comparison_df(books)

sns.heatmap(30000 - comp)

comp.sum().sort_values()
```




```
Harry Potter 2    31476.788631
Harry Potter 1    36947.458866
Harry Potter 3    45052.378293
Harry Potter 7    53917.374836
Harry Potter 4    56299.526770
Harry Potter 6    58468.196001
Harry Potter 5    77860.986712
dtype: float64
```

Future Questions, Plans, Applications

- Can we apply bookworm to massive corpora?
- Can bookworm be optimised for HPC?
- Do genre texts have common features, and if so, can we use that information to classify new ones?
- How does bookworm deal with historical texts or biographies?
- Can we adapt it to interpret screenplays?



harrisonpim/bookworm

Thanks

Harrison Pim

Twitter @hmpim
Github harrisonpim (.github.io)