rcv Documentation

Release 0.1.1

Metric Geometry and Gerrymandering Group

Mar 01, 2019

Contents

rcv is a Python library for tabulating ballots from ranked-choice elections. The package is distributed under the BSD 3-Clause License.

CHAPTER 1

Example

```
from rcv import FractionalSTV, PreferenceSchedule

schedule = PreferenceSchedule.from_ballots([
    ("Kamala", "Amy", "Elizabeth"),
    ("Kamala", "Elizabeth", "Amy"),
    ("Kamala", "Elizabeth", "Amy"),
])

stv = FractionalSTV(schedule, seats=2)
winners = stv.elect()

assert winners == {"Kamala", "Elizabeth"}
```

CHAPTER 2

API Reference

class `rcv.`**`FractionalSTV`**(*schedule*, *seats*, *quota=<function droop_quota>*)

Tabulates ranked-choice ballots according to Fractional Single Transferable Vote rules.

```
>>> schedule = PreferenceSchedule.from_ballots([
...     ("Kamala", "Amy", "Elizabeth"),
...     ("Kamala", "Elizabeth", "Amy"),
...     ("Kamala", "Elizabeth", "Amy"),
... ])
>>> stv = FractionalSTV(schedule, seats=2)
>>> winners = stv.elect()
>>> winners == {"Kamala", "Elizabeth"}
True
```

Parameters

- **schedule** (`PreferenceSchedule`) – A *`PreferenceSchedule`* holding all the ranked-choice ballots cast in the election.
- **seats** (*`int`*) – the number of seats up for election
- **quota** (*`function or Number`*) – the quota that a candidate must meet to win a seat

elect()

Runs the Fractional Single Transferable Vote algorithm to determine the winners of the election.

Returns a set holding the names (as strings) of the elected candidates.

Return type Set[str]

`rcv.`**`droop_quota`**(*number_of_votes*, *number_of_seats*)

The Droop quota for Single Transferable Vote tabulation. A candidate whose vote total meets this quota wins a seat.

class `rcv.`**`PreferenceSchedule`**(*ballots*, *candidates=None*)

A reduced preference schedule.

The `from_ballots()` method can be useful for creating a preference schedule from raw preference orderings:

```
>>> PreferenceSchedule.from_ballots([
...     ("Amy", "Elizabeth", "Kirsten"),
...     ("Amy", "Elizabeth", "Kirsten"),
...     ("Amy", "Elizabeth", "Kirsten"),
...     ("Kirsten", "Amy"),
...     ("Elizabeth", "Kamala", "Kirsten"),
...     ("Kamala", "Elizabeth"),
...     ("Kamala", "Elizabeth"),
...     ("Kamala", "Amy"),
...     ("Kamala", "Amy"),
... ])
<PreferenceSchedule total_votes=9>
```

You can also create the `BallotSet` beforehand and pass it directly to the constructor:

```
>>> ballots = BallotSet({
...     (("Amy", "Elizabeth", "Kirsten"), 10),
...     (("Kirsten", "Amy"), 20),
...     (("Elizabeth", "Kamala", "Kirsten"), 15),
...     (("Kamala", "Elizabeth"), 16),
...     (("Kamala", "Amy"), 14),
... })
>>> PreferenceSchedule(ballots)
<PreferenceSchedule total_votes=75>
```

Parameters

- **ballots** – the `BallotSet` holding all of the valid ballots cast in the election
- **candidates** – optionally, the `Candidates` up for election. If not provided, the candidates will be inferred from the names on the ballots.

classmethod from_dataframe(*df*)

Create a preference schedule from a dataframe whose rows are ballots. That is, the first column is the first-ranked candidate for each ballot, the second is the second-ranked candidate, and so on.

The preference orders are cleaned using `normalize_preferences()`.

Python Module Index

r

Index