POLITECNICO
MILANO 1863

PROGETTO DI INGEGNERIA INFORMATICA

Simulation Of Electoral Systems In Python

Supervisor: Professor Giovanni Agosta

Team Members

Ole Evjen-Caspersen, Henrik Bang-Olsen, Ask Kvarven

Simulating Electoral Systems in Python

For access to the complete codebase and a detailed tutorial on how to run the code, please visit the following GitHub repository: https://github.com/bangkorkor/electoral-systems-OHA.

The folders for this project, which are discussed in this report, can be found in LeggiElettorali/Norwegian and LeggiElettoriali/Dutch.

1 - Introduction

The application has been developed for the course "Progetto di Ingegneria Informatica" at Politecnico di Milano. The course aims to allow students to practically experiment with some of the knowledge learned during their studies, through the creation of a practical project, under the guidance of a responsible teacher. The team chose the project: *Simulating Electoral Systems in Python* by professor Giovanni Agosta. The project aims to create an electoral system simulator using as input the open data of the 2019 European Parliament Election.

1.1 - Motivation

The team made the decision to simulate the Norwegian and Dutch electoral systems on the European Parliament Election voting data from 2019. The project aims at programming and understanding how two different electoral systems would affect an election. There are a lot of different electoral systems used today, and with different success. Therefore the project could provide useful information by analyzing the outcome of different systems used in the same country. This could give some clues on why one system is more preferable than others in a given country. The group decided to look into the Norwegian and Dutch electoral systems as they are quite different and could give some interesting results.

2 - The Norwegian System

The Norwegian system is quite intricate and is based on the Sainte-Laguë method, which is used in multiple parts of the system. The system works by assigning a set of mandates between the different regions. These mandates, called district mandates (or region mandates), are allocated by looking at votes in each region to decide who wins these mandates. In addition, we also have something called adjustment mandates. There is one adjustment for each region. The adjustment mandates are assigned to ensure that the parties in the election receive a more proportional representation based on the nationwide vote count than what the allocation of district mandates would suggest.

1. The distribution of mandates to the regions

1.1 Calculating region score

In the Norwegian system, the region divides the region mandates between them by giving each region a score that is calculated as such:

$$region\ score\ =\ 1,4\ *\ land\ size\ [km^2]\ +\ region\ population\ .$$

1.2 Sainte-Laguë method

The region scores are then put through the Sainte-Laguë method, which is best explained with a basic example:
Let there be 3 parties (A, B and C) and they are competing for 5 mandates in a parlament.
The total number of votes received is:

Party A: 100 000 votes
Party B: 80 000 votes
Party C: 60 000 votes

Sainte-Laguë method works by allocating the first mandate to the party with the highest number of votes, which is Party A, then the votes for Party A gets divided by 3. The next mandate will be allocated to Party B, thus we need to divide Party B’s vote with 3. We do this until all the mandates are distributed. We divide the votes each time a party is allocated a mandate with the next odd number in the series (1, 3, 5, 7, 9, …). We can represent this in a table:

Party	Votes	Votes/1	Votes/3	Votes/5	Votes/7	Votes/9
A	100 000	100 000	33 333	20 000	14 286	11 111
B	80 000	80 000	26 667	16 000	11 429	8 889
C	60 000	60 000	20 000	12 000	8 571	6 667

Thus the 5 mandates would be allocated as such:

First mandate: Part A (100 000 votes)
Second mandate: Party B (80 000 votes)
Third mandate: Party C (60 000 votes)
Fourth mandate: Party A (33 333 votes)
Fifth mandate: Party B (26 667 votes)

This is the logic of the Sainte-Laguë method which is used to distribute the 53 (excluding the adjustment mandates) mandate between the 20 regions in Italy. The only difference from the example is that instead of parties we have regions and instead of votes we have region scores as described in 1.1.

2. The distribution of district mandates (region mandates)

Now that we have distributed the district mandates to the regions, which in the case of Italy amounts to 53 mandates, we must divide them between the parties. This is achieved by calculating the number of votes each party received in each region to determine the number of mandates each party secures in those regions. Again we used the Sainte-Laguë method, but this time a modified version. The only difference is that the first divisor is 1,4 instead of 1. This is done to make it harder for the smaller parties to get mandates, and is often referred to in Norway as the *4% threshold*.

3. The distribution of adjustment mandates

This part is quite complicated; therefore, we will break it down further:

1. The 4% threshold

For a party to be qualified for the adjustment mandates, the party must have received 4% or more of the vote on a nationwide scale. This is simply calculated by taking the total number of votes each party received and dividing it by the total number of votes in the election.

2. “National” distribution

Now that we have established which parties are entitled to the adjustment mandates, we must decide how many each party should receive. This is done by doing a new “national” distribution for the parties over the threshold. In the national distribution, we combine the total district mandates for parties above the threshold and the adjustment mandates, then redistribute them using the modified Sainte-Laguë method.

3. Checking the difference

Now that each party over the threshold has received a number of mandates from the “national” distribution, we check this number against the actual number of mandates they have received. For example, if Party A received 50 mandates in the national distribution and had 48 district mandates in the actual election, they would receive 50 - 48 = 2 adjustment mandates. However, if Party B received 30 mandates in the national distribution and had 31 district mandates in the actual election, the difference would amount to 30 - 31 = -1. This means that Party B is overrepresented, and should not receive any adjustment mandates. Therefore, in any case a party gets a negative difference between national distribution and actual distribution, we must do point 2. again, but without Party B and their mandates and votes. This includes the one mandate party B was “overrepresented” by. Meaning if we originally were to distribute 20 adjustment mandates, it would now only be 19 because of Party B.

4. Calculate a weighted quotient

For each party allocated adjustment mandates, it should be calculated a weighted quotient for them in each region. This quotient is calculated as such:

- The party's number of votes in the region should be divided by a number that is 1 bigger than the doubled number of region mandates won in the region:

$$Quotient \ = \ \frac{\#\, party's \ \ votes\ in\ the\ region}{2{*}region\ mandates\ {+}1}$$

- Thereafter, we must weight this quotient by dividing the quotient on the average number of votes per region mandate in the region:

$$Weighted\ quotient \ = \ \frac{Quotient}{\frac{\#\ votes\ in\ the\ region}{\#\ \ region\ mandates\ in\ the\ region}}$$

5. Distributing the adjustment mandates

Now that each party allocated adjustment mandates have a weighted quotient in each region, we can start distributing the parties adjustment mandates from the different regions. The first adjustment mandate should go to the party and the region with the largest weighted quotient. The second adjustment mandate is allocated to the party and region with the second largest weighted quotient, and so on. This continues until all the parties have gotten their calculated number of adjustment mandates and each region has received one adjustment mandate.

3 - The Dutch System

The Netherlands uses a system of *proportional representation*. In simple terms, this means that the number of mandates a party gets allocated is proportional to the number of votes that party receives nationwide. It doesn't matter where the votes are cast from, as mandates in the parliament are not assigned to specific regions or parts of the country. The only thing that counts is the total number of votes each party receives. This system sounds really simple, and compared to the one used in Norway, it is. Nonetheless, we will see that things can get a little complicated when it comes to rounding off the number of votes, and how this will result in "excess" mandates that will have to be allocated using other principles. Understanding exactly how this works was essential when implementing the system in our python program, and we will therefore quickly explain the mathematics behind it in three steps:

1. Calculating the Hare quota:

In order to assign the first mandates, the so-called "Hare quota" has to be calculated. This value is calculated by simply dividing the total number of valid votes nationwide by the number of mandates that are to be allocated. We can write this as:

$$Quota \ = \ \frac{Total\,number\ of\ votes}{Total\,number\ of\ mandates}$$

2. Using the quota to allocate mandates:

The next step is to divide the total number of votes received by each party by the quota we just calculated, making sure to always round down to the nearest integer. This will give us the number of mandates allocated to each party:

$$Mandates\ for\ party\ X \ = \lfloor \frac{Total\,number\ of\ votes\ received\ by\ party\ X}{Quota} \rfloor$$

3. Applying the D'Hondt method to allocate the remaining mandates:

Because we're always rounding down, there will be instances where one or more mandates remain not distributed. When this happens, the distribution of these mandates will be calculated using another method called the D'Hondt method. In this method a new quota is calculated, but this time, one for each party. The total number of votes received by a party is divided by the number of mandates assigned to this party so far plus one.

$$D'Hondt\ quota\ for\ party\ X \ = \ \frac{Total\,number\ of\ votes\ received\ by\ party\ X}{Mandates\ \ already\ assigned\ to\ party\ X + 1}$$

The party with the highest D'Hondt quota receives the first remaining mandate. For the allocation of the next mandate the same rules hold, but because party X's just got another mandate, their quota will have to be recalculated, with + 2 in the denominator this time instead of + 1. This keeps going on until all the mandates are allocated.

4. Requirements

The software is designed to meet both functional and non-functional requirements, ensuring a robust, user-friendly, and efficient tool for simulating election processes. These requirements are essential for accurate simulations, maintaining high performance, and ensuring the system's long-term viability.

4.1 Functional Requirements

To meet the functional requirements, the software must accurately simulate the election processes for both the Norwegian and Dutch election systems. This involves several key capabilities:

1. **Simulation of Election System**: The system must accurately replicate the election processes of both the Norwegian and Dutch system. This includes implementing the specific rules and procedures of each election system to ensure realistic and reliable simulations.
2. **Result Visualization**: The system must provide visualizations of the election results, including charts and graphs. This helps in analyzing and presenting the simulation outcomes in an accessible and understandable manner.
3. **Data Handling**: The system must be capable of reading and processing input data files. This includes files containing votes, party lists, and region information. The data is stored in a well-organized manner to facilitate efficient access and integration into the simulation process.
4. **Ease of Execution:** The system must be designed to be easy to run, including clear documentation and scripts for setting up the environment, installing dependencies, and executing the simulations. This ensures that users can easily set up and run the simulations without extensive technical knowledge.

4.2 Non-Functional Requirements

In addition to the functional requirements, the system must also meet non-functional requirements to ensure usability, maintainability and scalability:

1. **Usability**: The system should be user-friendly when configuring and running simulations. This includes clear documentation and intuitive controls, enabling users to set up and execute simulations with minimal effort.
2. **Maintainability**: The codebase should be easy to understand and modify, allowing for straightforward updates and debugging. This is achieved through a modular

design, where each component is self-contained and handles specific aspects of the simulation.

3. **Scalability**: The architecture must follow a clear template for each electoral system, as described further in chapter 5. This ensures that new systems can be added following the same architecture, making it easier maintaining the codebase.

By meeting these functional and non-functional requirements, the election simulation system provides a comprehensive and reliable tool for simulating and analyzing election processes.

5 - Software Architecture

The codebase is meticulously organized into several distinct components, ensuring clarity and maintainability. This modular design is essential as it allows the team to navigate and update the codebase with ease, maintaining each component separately without affecting the whole system. Each module is self-contained, handling specific aspects of the election simulation, which aids in both debugging and extending the complexity of the system.

The architecture supports simulations for both the Norwegian and Dutch election systems, which, despite their differences, share a common structural framework. The systems are structured in the same way but are separated into different folders and run independently.

The code is structured in three primary directories:

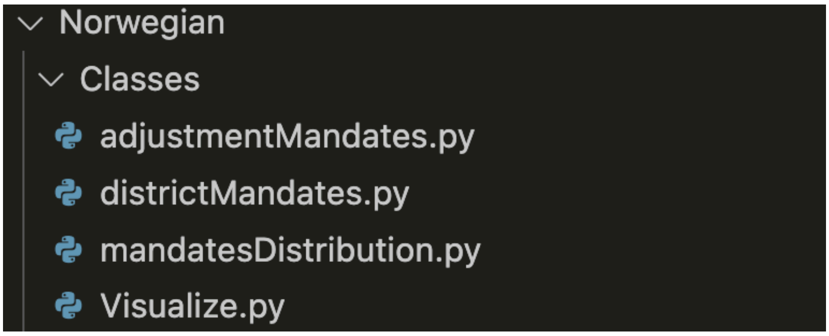
- Classes
- Data
- Instances

5.1 Classes

The **Classes** directory contains all the class definitions used throughout the system. These classes encapsulate the core functionalities and behaviors required for the election simulation. Here the actual calculations for simulating the mandate-distribution will be made.

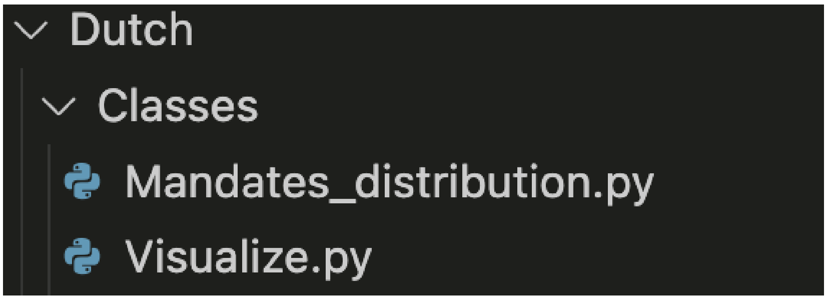
The Norwegian system, being more complex than the Dutch, uses three classes to calculate the mandates-distribution (the final result). **districtMandates.py** distributes the right amount of mandates to every region, **adjustmentMandates.py** calculates the *adjustment mandates* for every region, and **mandateDistribution.py** calculates the standard *region mandates*. Together, these three classes form the rule for simulating the Norwegian system. For the Dutch system, the calculation is being done all within the same class; **Mandates_distribution.py.**

In the **Classes** directory, the **Visualize.py** class is also placed. In this class the different plots are being made, using the matplotlib and poli_sci_kit library.



```
Norwegian
  Classes
    adjustmentMandates.py
    districtMandates.py
    mandatesDistribution.py
    Visualize.py
```

Figure 1: Classes for the Norwegian system



```
Dutch
  Classes
    Mandates_distribution.py
    Visualize.py
```

Figure 2: Classes for the Dutch system

5.2 Data

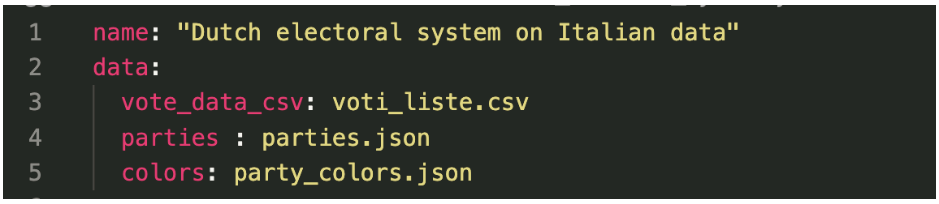
The **Data** directory is dedicated to storing all the necessary datasets and configuration files required by the simulations. This directory includes: the actual votes stored in the **voti_liste.csv** file, a list of all the different parties that can be voted for.

The Norwegian system also includes a list (**parties.json**) detailing the size and population of every region, which is used to calculate the number of mandates each region is entitled to.

5.3 Instances

The **Instances** directory includes an instance configuration file that determines which specific files will be used in a given simulation. This configuration file lists the names of various input files required for the simulation. In the __main__.py script, the instance configuration file is read to direct the input files to the appropriate classes within the simulation and visualization modules.

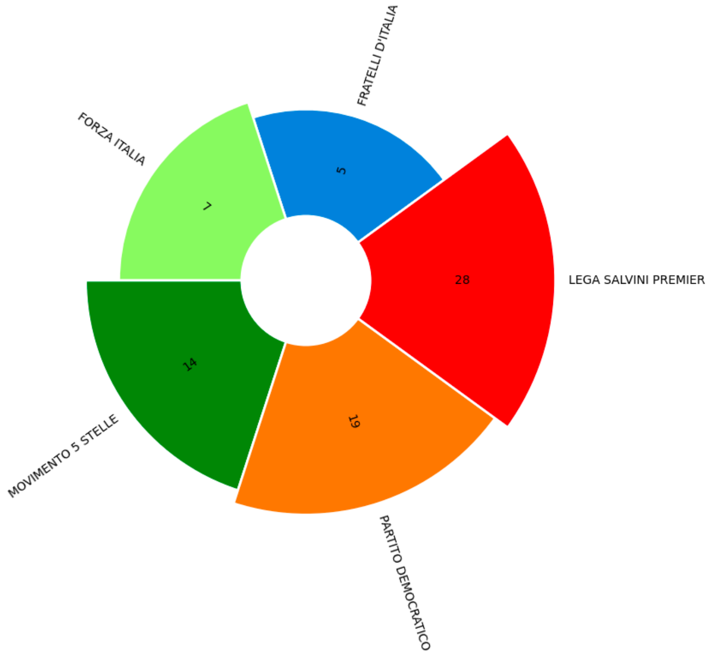
The primary purpose of the instance configuration file is to enhance code readability and manageability. By centralizing the file references, it allows for straightforward modifications and easy identification of the input files associated with each simulation run. This structure not only simplifies the setup process but also ensures that each simulation is well-organized and easy to navigate.



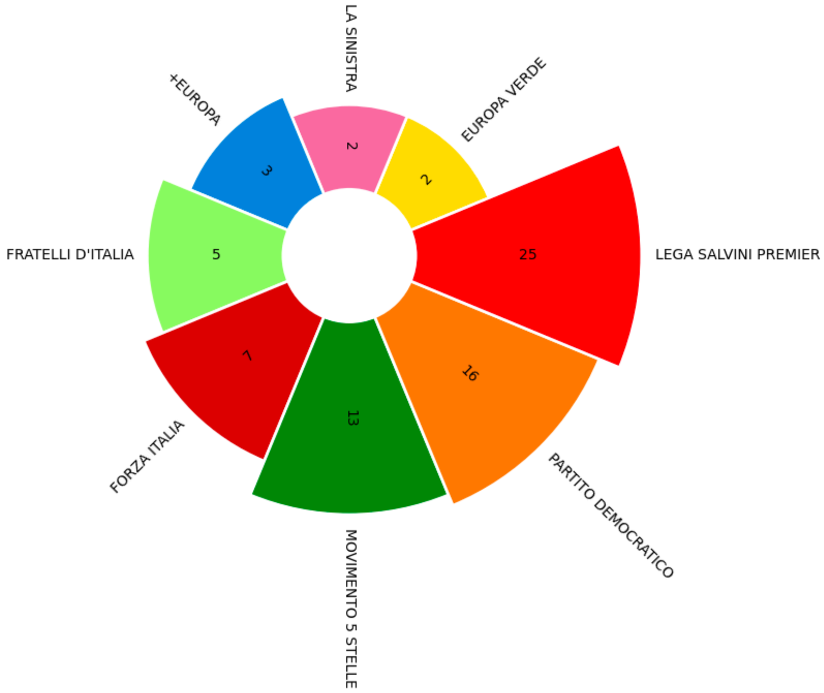
```yaml
name: "Dutch electoral system on Italian data"
data:
  vote_data_csv: voti_liste.csv
  parties : parties.json
  colors: party_colors.json
```

Figure 3: **Dutch_electoral_system.yalm**. This is the instance-file for the Dutch System

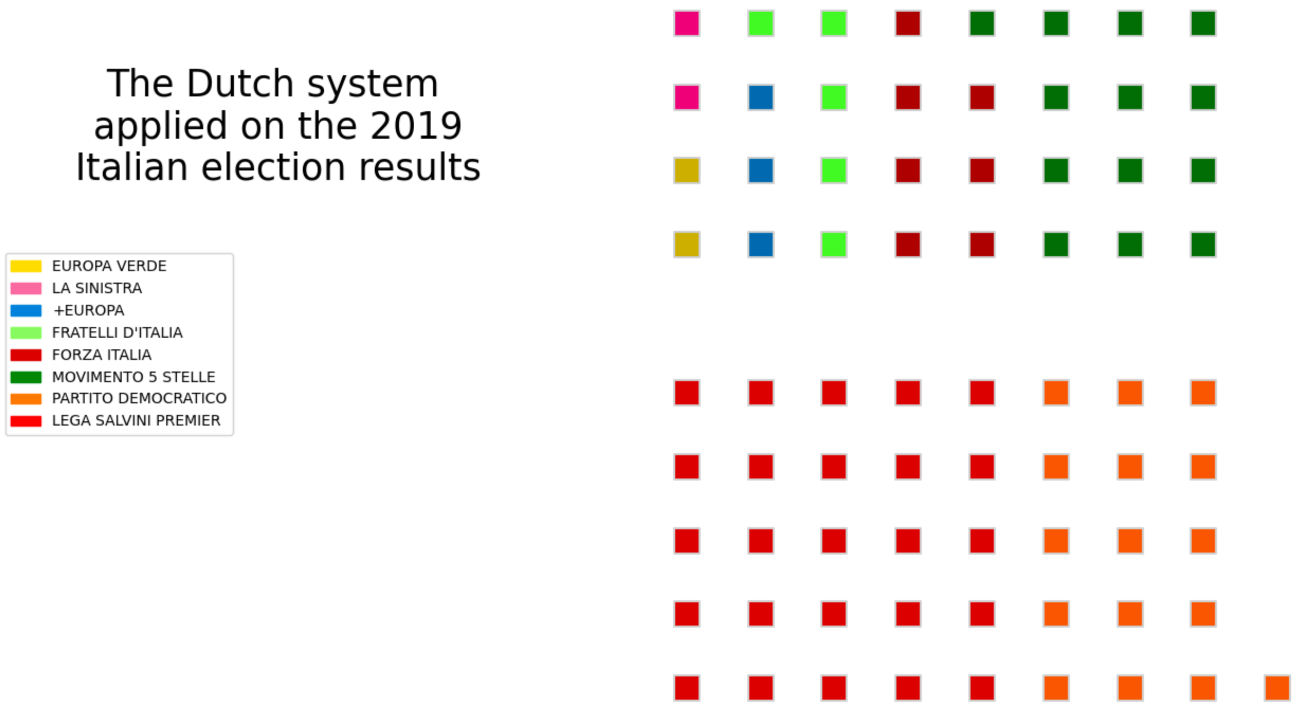
6 - Results



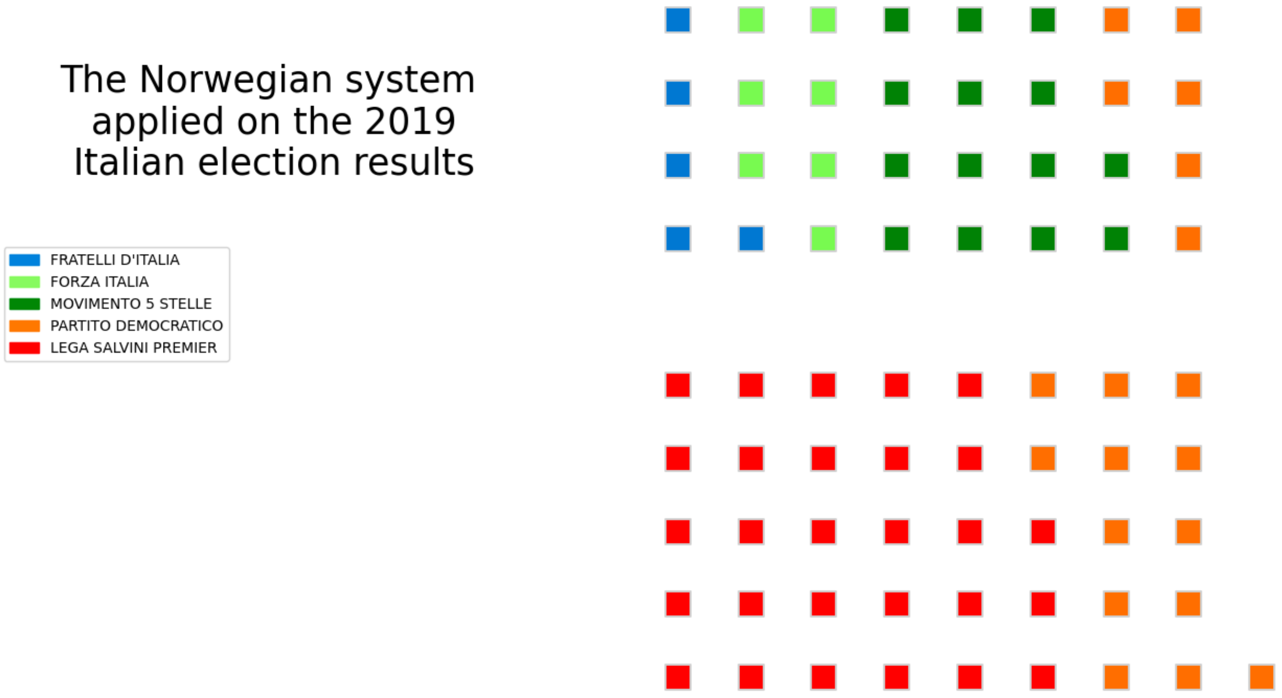
The Norwegian system applied on the 2019 Italian election results



The Dutch system applied on the 2019 Italian election results



The Dutch system applied on the 2019 Italian election results



The Norwegian system applied on the 2019 Italian election results

7 - Analysis

The main goal of this project was to use different electoral systems on the Italian election and see the outcome. However, it's important to understand what worked and what didn't, because this can lead to an interesting understanding. Therefore, in this analysis, we'll see why some aspects of both the Norwegian and Dutch electoral systems fit the Italian well, and also look at some flaws.

The Norwegian and Dutch elections differ in one important way and that is in the emphasis on regions. For Norway, which is a very long country, it's important to get representatives for every part of the country, not only the most populated. This doesn't account for the Netherlands, which is a smaller country and doesn't have big regional differences in the country. Therefore, we would assume that the Norwegian system would fit the Italian system better, since Italy, similar to Norway, is a long country with differences in the regions. However, there are some big flaws in applying the Norwegian system directly to Italy without modifications.

In Norway, the distribution of region mandates, as described in section 2, is based on population and area size of the region. This works well in Norway, because the less populated regions of Norway are very large in area. The area size is 1,4 times more important than the population size, meaning that it evens out some of the disadvantages the regions with big areas and small populations have. The story is not the same for Italy, since the population and area size is somewhat correlated. Meaning that the smaller region Autso Valley is distributed 0 regional mandates. One can argue that it would be unfair to give them more mandates when they represent such a little part of Italy. However, it makes little sense that their votes are without any meaning, if we don't consider the adjustment mandate. Either way, it is clear that this is a big flaw in the application of the Norwegian system in the Italian election.

The Norwegian election also makes it difficult for smaller parties to get mandates by implementing the 4% threshold. This is not used in the Dutch system, allowing for a broader variation of representation in the parliament. This is also seen in the different outcome of the Norwegian and Dutch model used on the Italian election. Whereas the Dutch gave 8 different parties mandates, and the Norwegian only 5. This may be due to the fact that Norway doesn't want a small party in one region to gain regional power, when it only represents one limited part of Norway, and it would make more sense for them to gain control through the district election.

It's apparent that each country's electoral system is a product of its unique political culture and history. Norway’s system is designed to balance regional representation with stability, reflecting Norway's geography and demography. The Netherland’s system, emphasis on proportional representation without regional weighting, and represents a more homogeneous population distribution. Therefore, when using these systems in the Italian election, it’s obvious that we need to modify these systems based on the political culture and history of Italy.

8 - Conclusion

We have seen the usage of both the Norwegian and Dutch system on the Italian European parliament election from 2019. While the Norwegian and Dutch systems both have their strengths, applying them directly to the Italian context without modifications lead to some significant challenges. The Norwegian system’s emphasis on regional representation and high electoral threshold did marginalize smaller regions and parties when applied to the Italian election. Conversely, the Dutch system with its absolute proportional representation might lead to fragmentation and instability. Thus, a tailored approach, where we might take in parts of both systems would be most effective. There is no one-fits-all electoral system, and shows the importance of understanding the political landscape in the country when using an electoral system.