# Stream Mining

## One-Hot Encoding and DGIM

Zeno Adrian Weil

Data Science 1
Goethe University Frankfurt

7th June 2022

# One-Hot Encoding

# One-Hot Encoding

- **categorical** features common

$$x \in \{\text{red}, \text{green}, \text{blue}\}$$

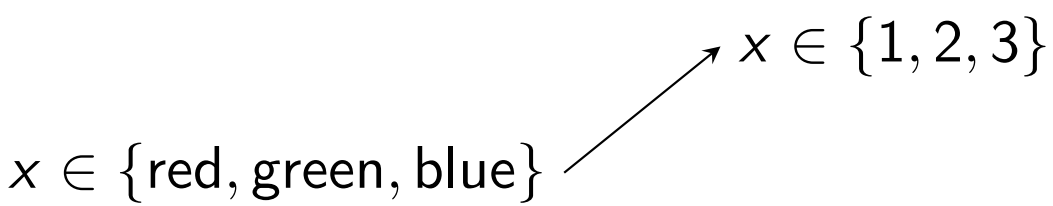
# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms

$$x \in \{\text{red}, \text{green}, \text{blue}\}$$

# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms
- naïve approach: number serially



# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms
- naïve approach: number serially

$x \in \{\text{red}, \text{green}, \text{blue}\}$ → $x \in \{1, 2, 3\}$ ✗

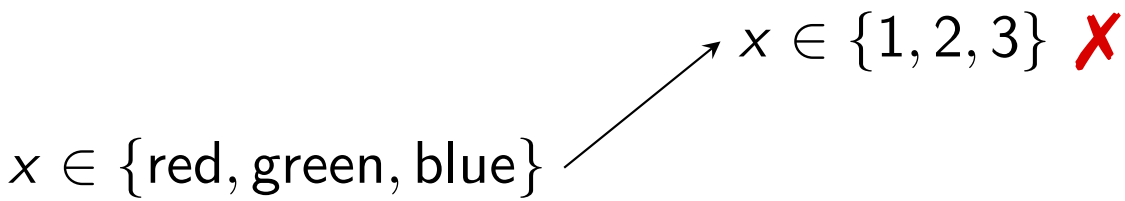
# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms
- naïve approach: number serially
  - meaningless numerical calculations

$x \in \{\text{red}, \text{green}, \text{blue}\}$ → $x \in \{1, 2, 3\}$ ✗

# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms
- naïve approach: number serially
  - meaningless numerical calculations
- **one-hot encoding**



# One-Hot Encoding

- **categorical** features common
- need for numbers in algorithms
- naïve approach: number serially
  - meaningless numerical calculations
- **one-hot encoding**
  - one binary feature for each possible value

$$x \in \{\text{red}, \text{green}, \text{blue}\} \rightarrow x \in \{1, 2, 3\}$$ ✗

$$x \in \{\text{red}, \text{green}, \text{blue}\} \rightarrow (x_{\text{red}}, x_{\text{green}}, x_{\text{blue}}) \in \{0, 1\}^3$$ ✓

# The Datar-Gionis-Indyk-Motwani Algorithm

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

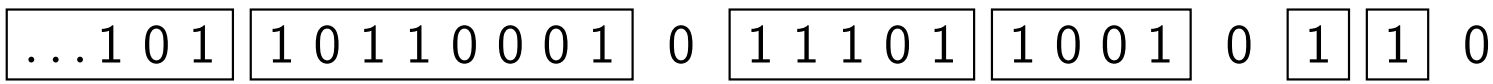
- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**

# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**

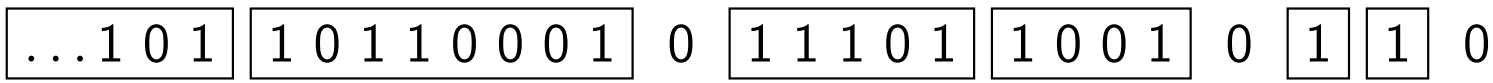


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**

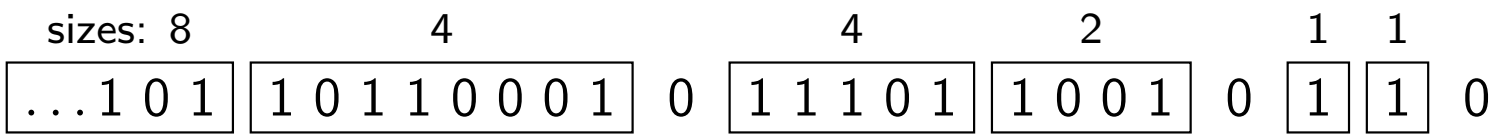


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones

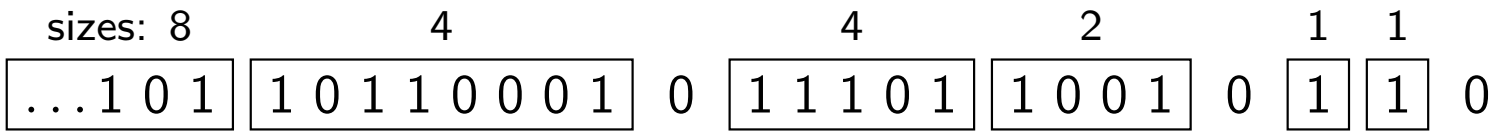


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two

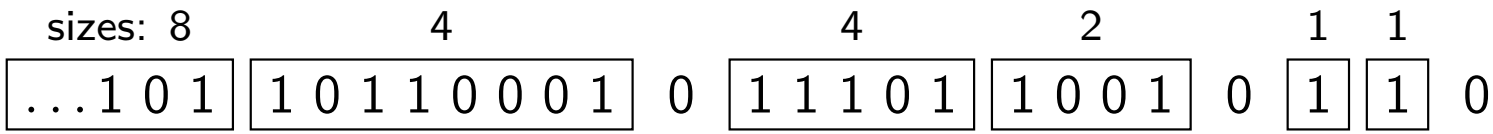


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones

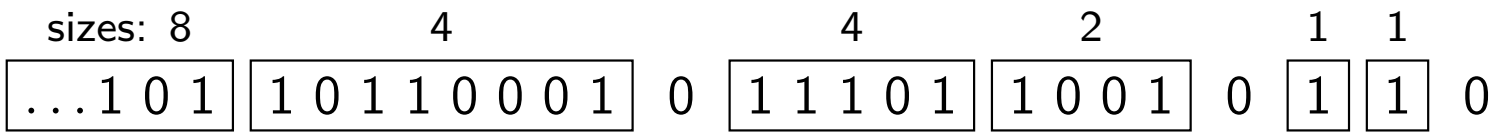


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones
- **estimation**: half the size of the oldest bucket + sum of sizes of all other buckets

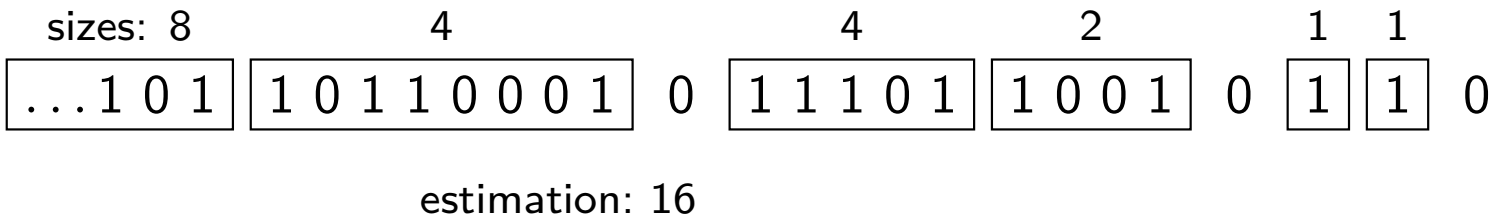


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones
- **estimation**: half the size of the oldest bucket + sum of sizes of all other buckets

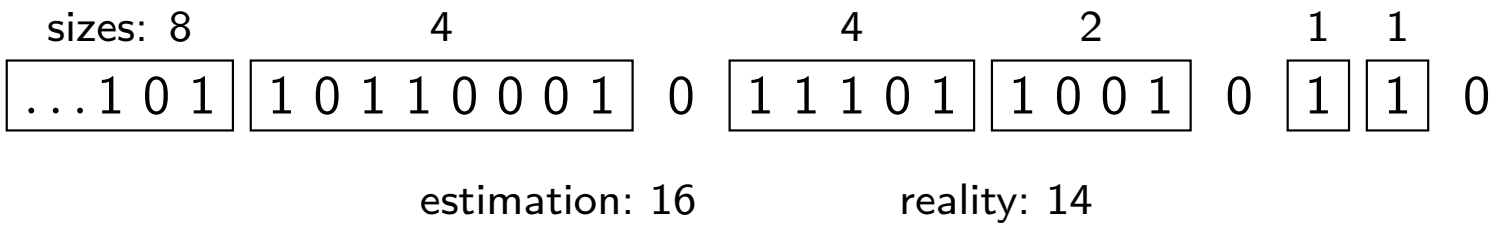


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones
- **estimation**: half the size of the oldest bucket + sum of sizes of all other buckets

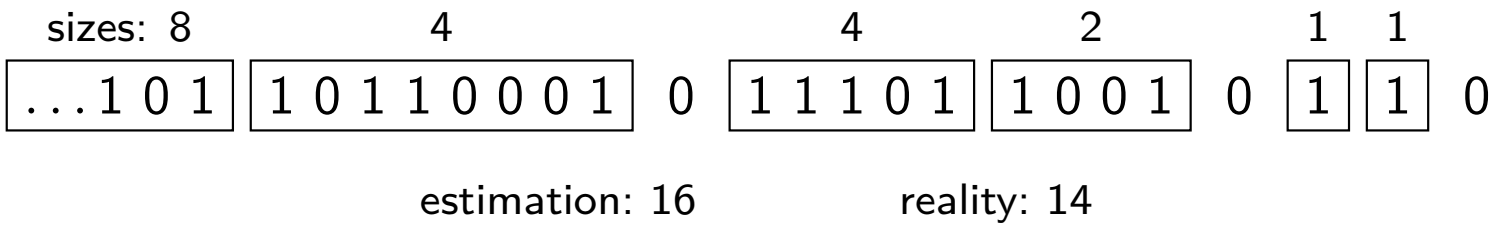


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones
- **estimation**: half the size of the oldest bucket + sum of sizes of all other buckets
  - error rate: 50%

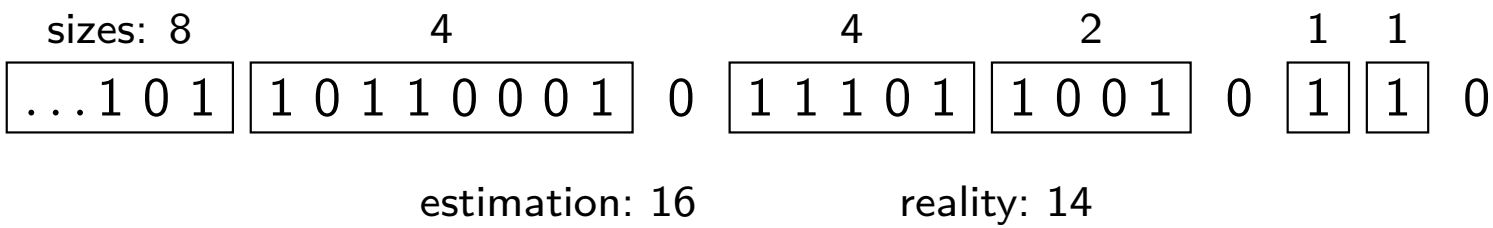


# The Datar-Gionis-Indyk-Motwani Algorithm

## Objectives

- **Estimate** the number of **ones** in a bit stream!
- Be **space-efficient**!

- window size $N$
- $\mathcal{O}(\log_2 N)$ **buckets**
  - **timestamp**
  - **size** = number of ones
    - powers of two
  - include all ones; end with ones
- **estimation**: half the size of the oldest bucket + sum of sizes of all other buckets
  - error rate: 50%
- needs only $\mathcal{O}((\log_2 N)^2)$ **bits**



# The Mushroom Data Set (J.S. Schlimmer, 1987)

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species
  - 4208 edible
  - 3916 poisonous

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species
  - 4208 edible
  - 3916 poisonous
- **22 attributes** with 128 possible values

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species
  - 4208 edible
  - 3916 poisonous
- **22 attributes** with 128 possible values
- saved as CSV

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
                        ...
```

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species
  - 4208 edible
  - 3916 poisonous
- **22 attributes** with 128 possible values
- saved as CSV

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
                        ...
```

Are there simple rules to determine edibility?

# The Mushroom Data Set (J.S. Schlimmer, 1987)

- **8124 samples** of 23 mushroom species
  - 4208 edible
  - 3916 poisonous
- **22 attributes** with 128 possible values
- saved as CSV

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
                        ...
```

Are there simple rules to determine edibility? **Yes!** (e.g. odour)

# Implementation

# Implementation

- load CSV with Python

# Implementation

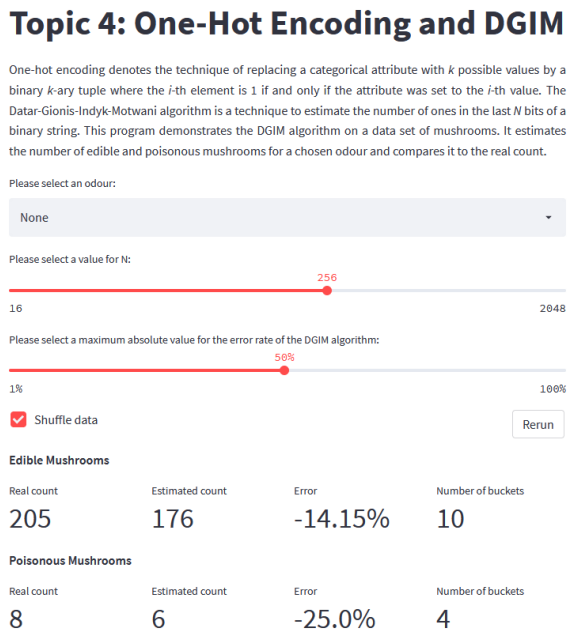
- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour

# Implementation

- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**

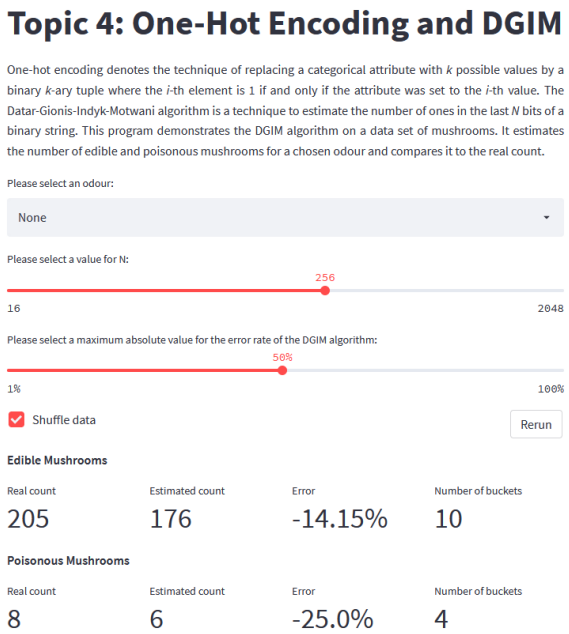
# Implementation

- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**
- **Streamlit** for the **interface**



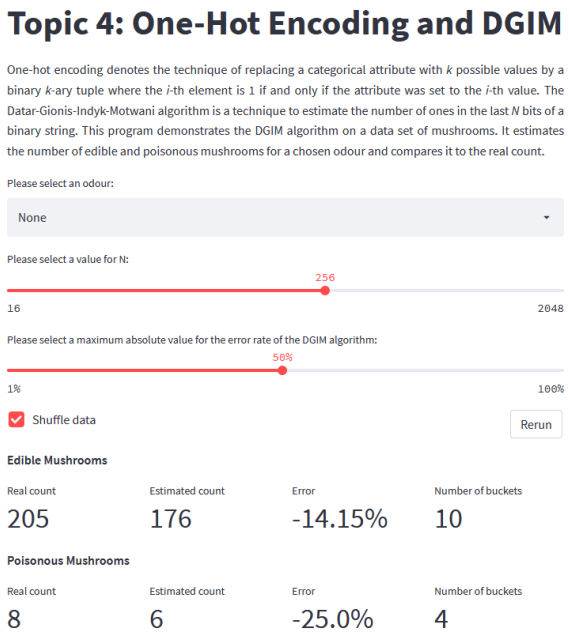
# Implementation

- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**
- **Streamlit** for the **interface**
- options



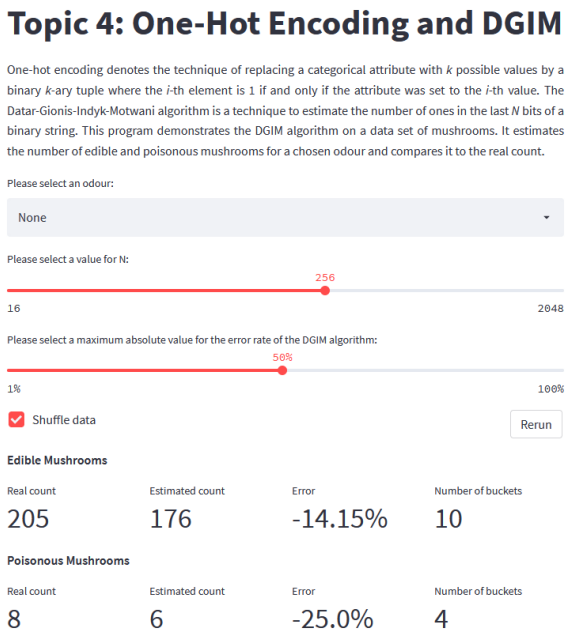
# Implementation

- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**
- **Streamlit** for the **interface**
- options
  - odour type



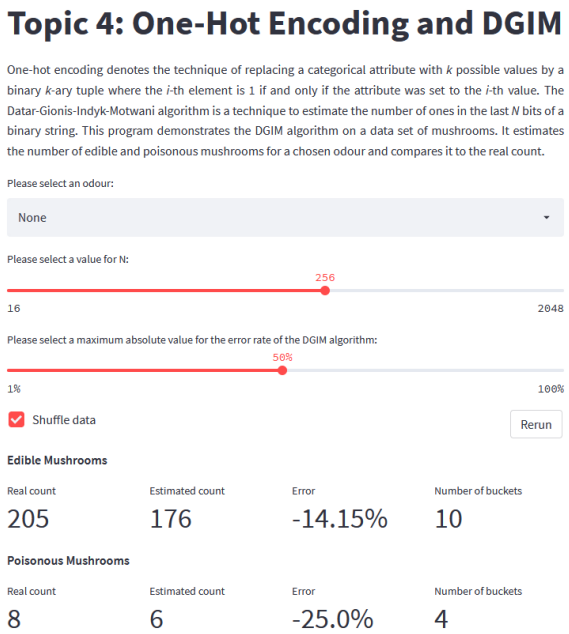
# Implementation

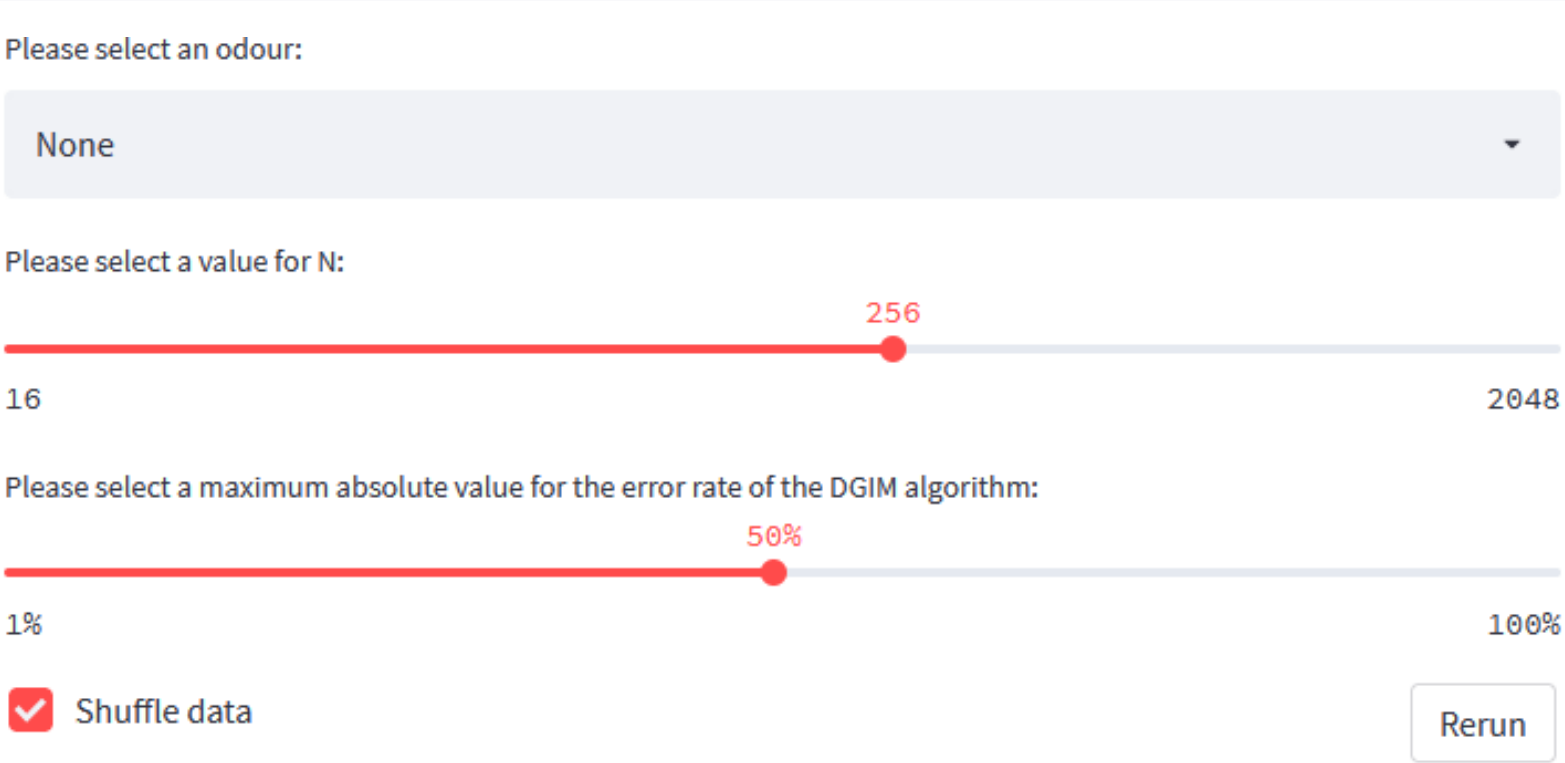
- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**
- **Streamlit** for the **interface**
- options
  - odour type
  - window size $N$

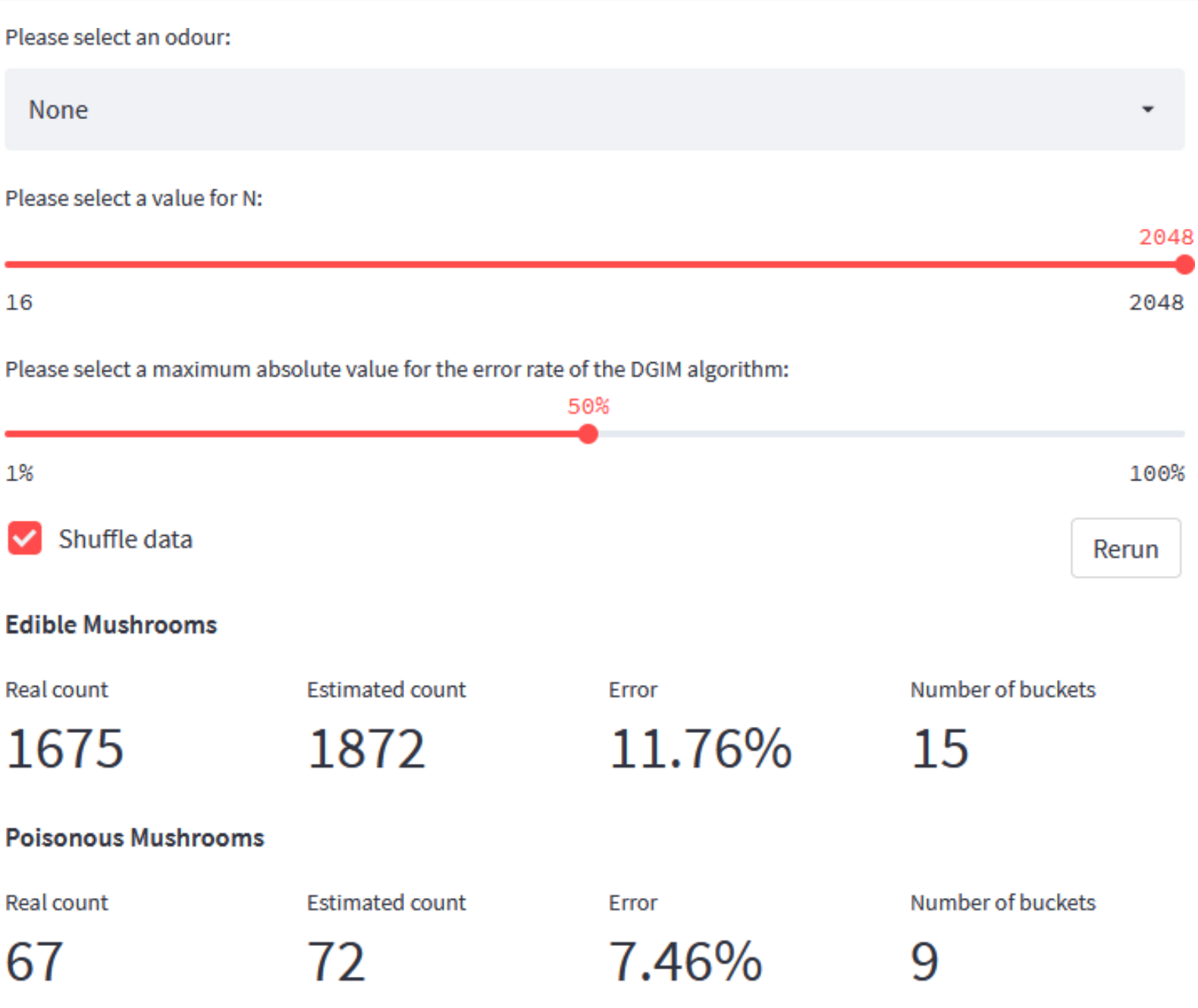


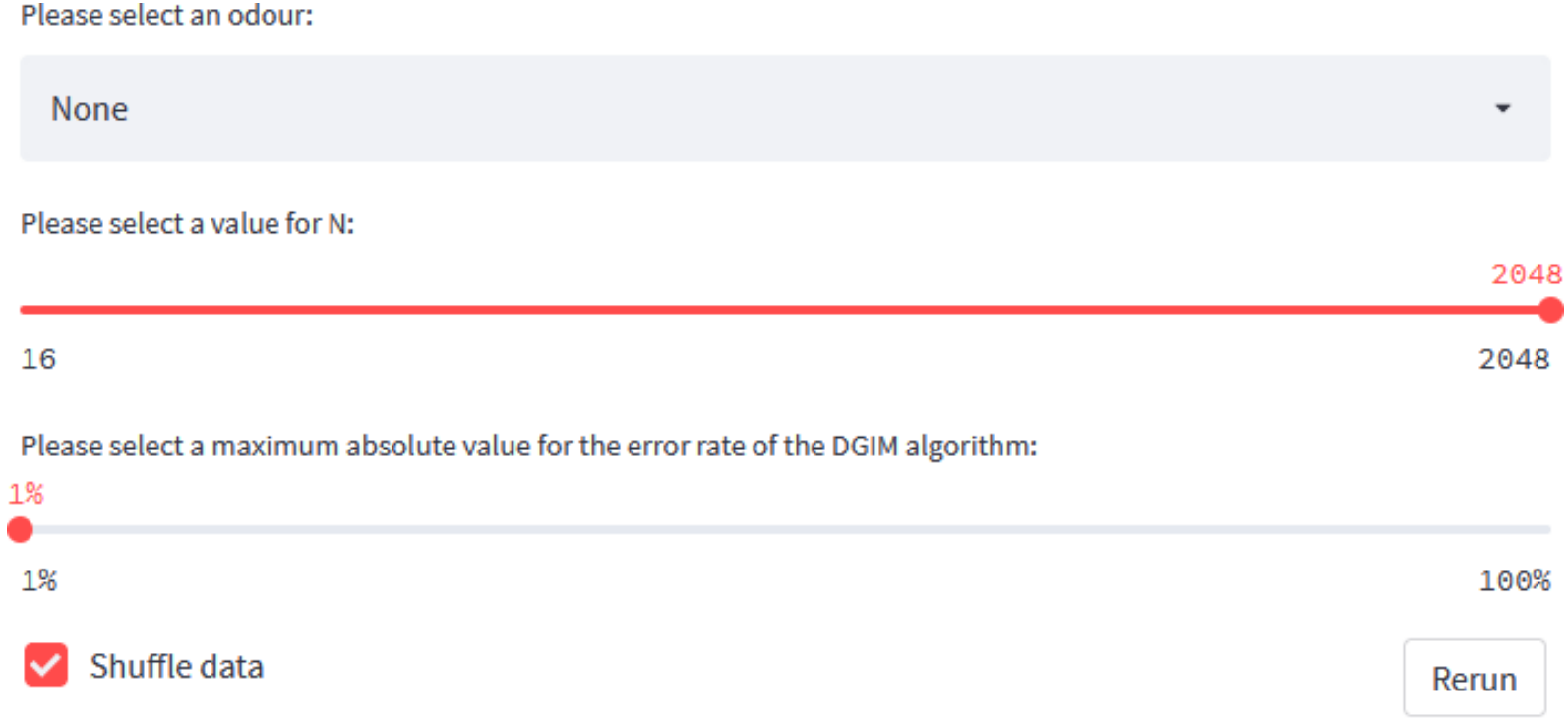
# Implementation

- load CSV with Python
- **2D array** for the **one-hot encoding** of the odour
- Python package **dgim** for the **algorithm**
- **Streamlit** for the **interface**
- options
  - odour type
  - window size $N$
  - error rate




Please select an odour:
None
Please select a value for N:
256
16
2048
Please select a maximum absolute value for the error rate of the DGIM algorithm:
50%
1%
100%
Shuffle data
Rerun
Edible Mushrooms
Real count
214
Estimated count
208
Error
-2.8%
Number of buckets
10
Poisonous Mushrooms
Real count
11
Estimated count
12
Error
9.09%
Number of buckets
6


Please select an odour:
None
Please select a value for N:
2048
16
2048
Please select a maximum absolute value for the error rate of the DGIM algorithm:
50%
1%
100%
Shuffle data
Rerun
Edible Mushrooms
Real count
1675
Estimated count
1872
Error
11.76%
Number of buckets
15
Poisonous Mushrooms
Real count
67
Estimated count
72
Error
7.46%
Number of buckets
9


Please select an odour:
None
Please select a value for N:
2048
16
2048
Please select a maximum absolute value for the error rate of the DGIM algorithm:
1%
1%
100%
Shuffle data
Rerun
Edible Mushrooms
Real count
1678
Estimated count
1672
Error
-0.36%
Number of buckets
410
Poisonous Mushrooms
Real count
68
Estimated count
68
Error
0.0%
Number of buckets
68

# References


- Project code: `https://github.com/s9770652/DS1-DGIM`
- Mushroom data set: `https://archive-beta.ics.uci.edu/ml/datasets/mushroom`
- *Streamlit*: `https://streamlit.io/`
- Python package *dgim*: `https://pypi.org/project/dgim/`
- Description of one-hot encoding: `https://sherbold.github.io/intro-to-data-science/04_Data-Analysis-Overview.html#Features`
- Description of the DGIM algorithm (Section 4.6): `http://infolab.stanford.edu/~ullman/mmds/ch4.pdf`