PERFUMES RATING PREDICTION

CONTENT

GOALS AND OBJECTIVES

Objective n° 1

Quality Estimation

Objective n° 2

Consumer Insights

Objective n° 3

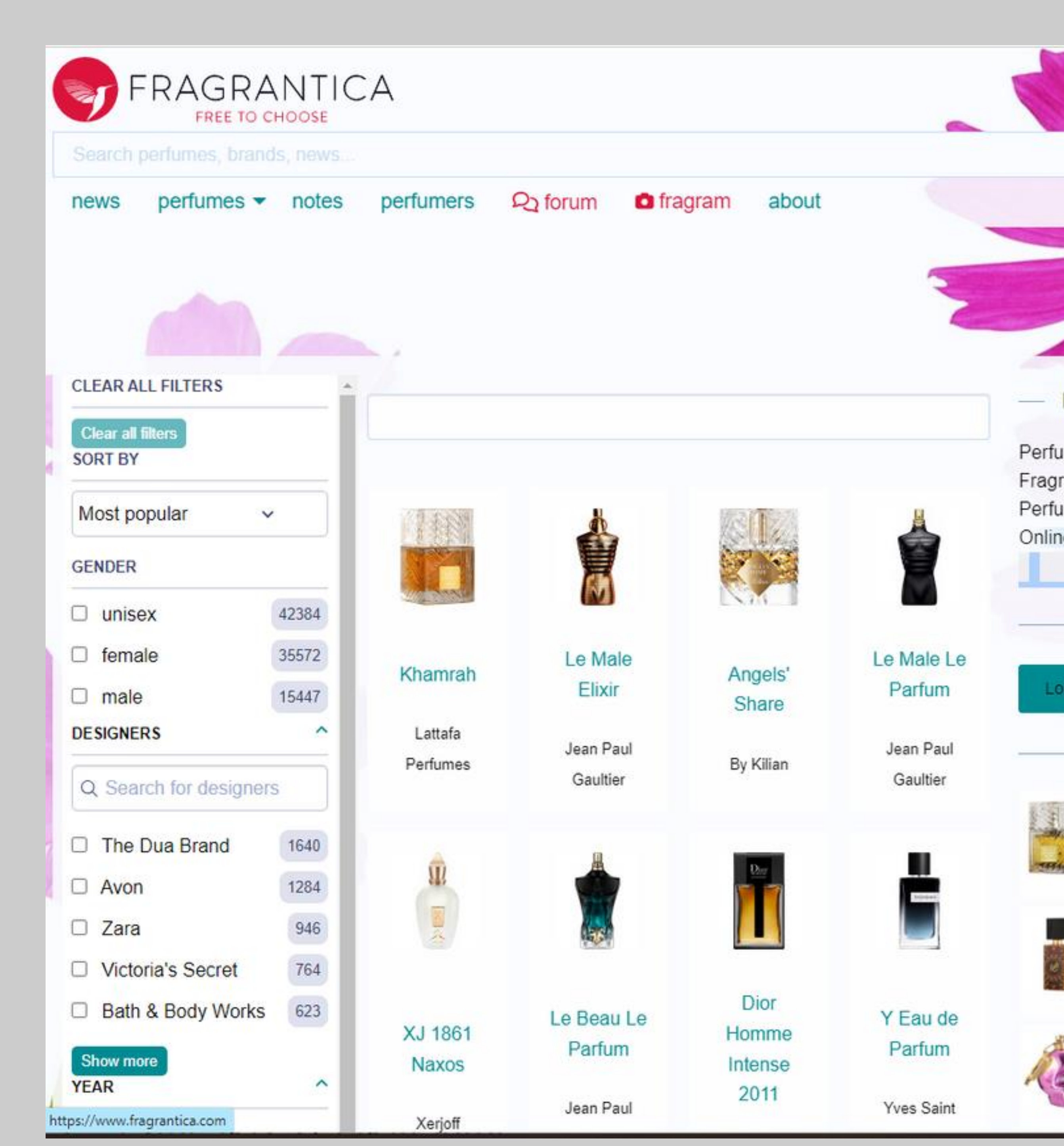
Companies Competitive Advantage

DATA COLLECTION

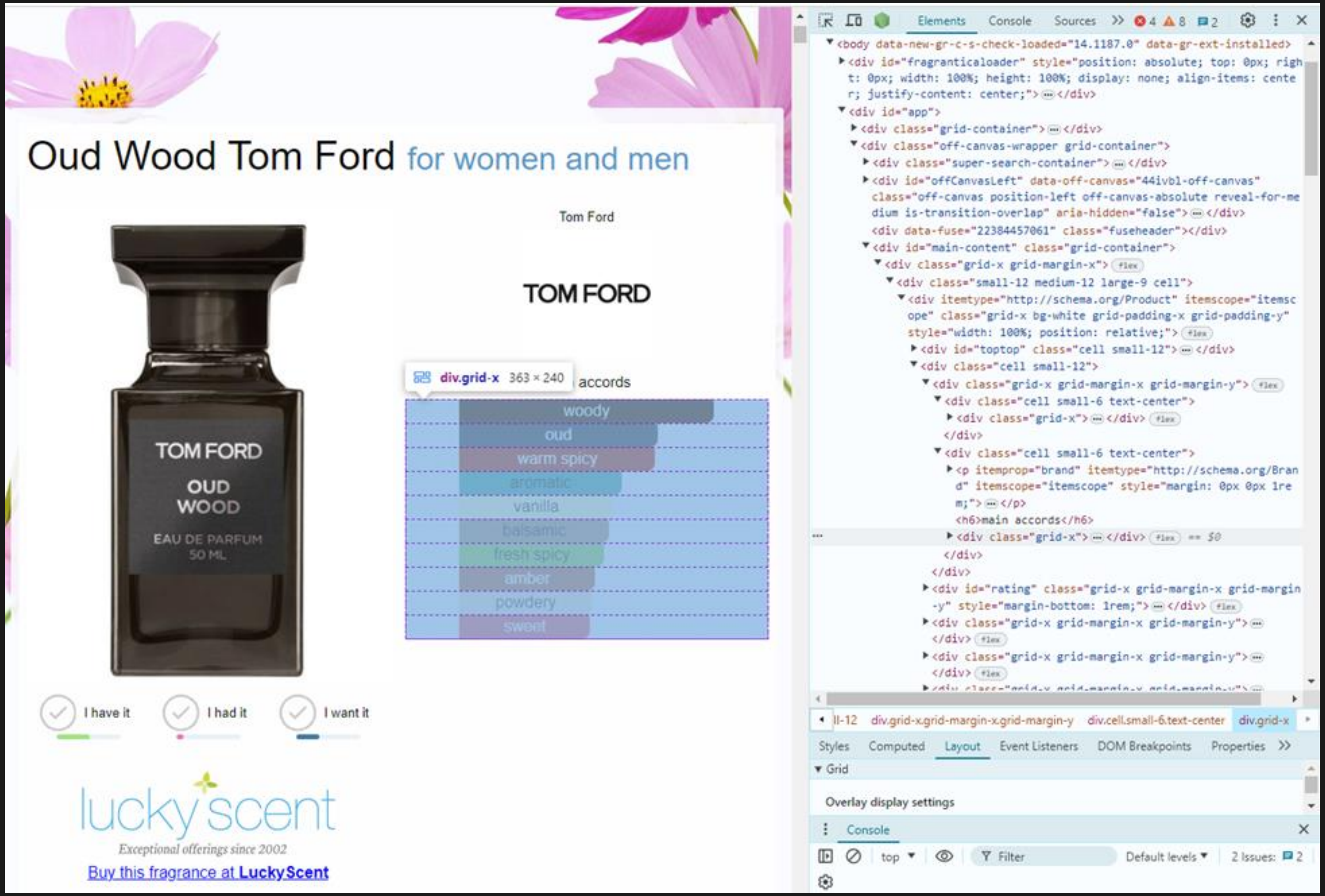
FRAGRANTICA.COM

Our target for obtaining Data – fragrantica.com

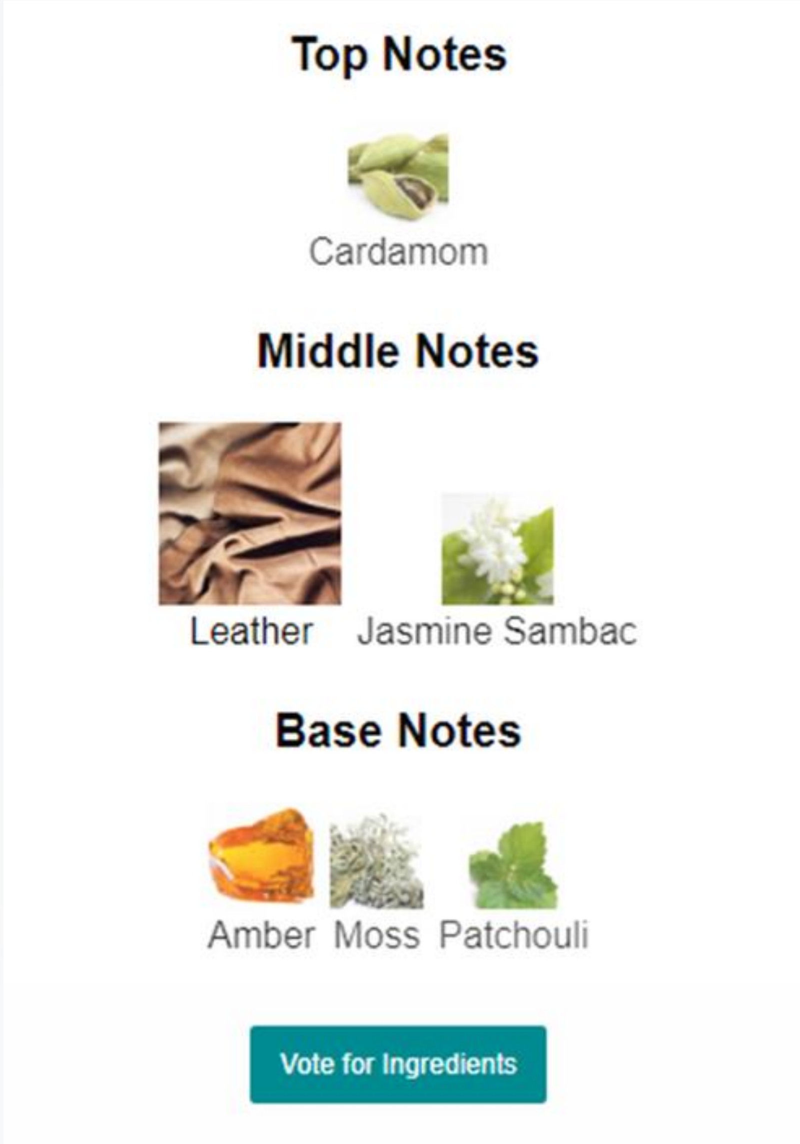
The website provides detailed information on thousands of perfumes based on user reviews and manufacturers data



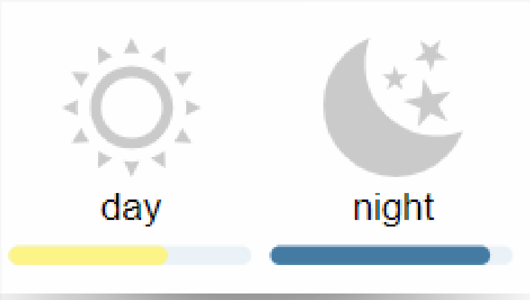
STARTING POINT



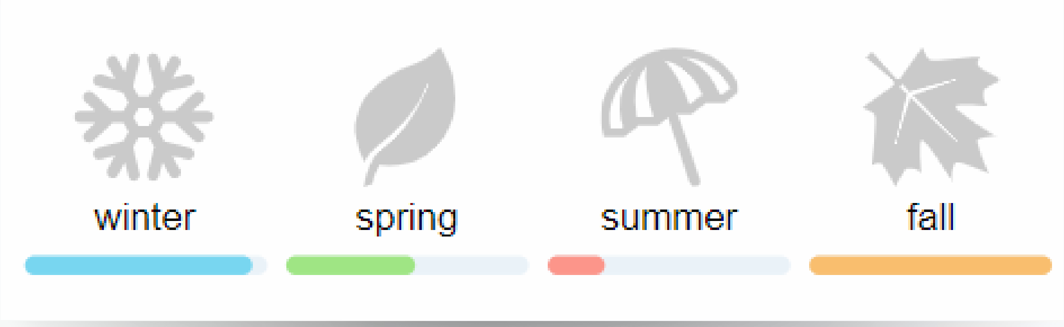
TARGET FEATURES



DAY/NIGHT SUITABILITY



SEASON FIT



RATING: MAIN TARGET

POPULARITY

Perfume rating 4.32 out of 5 with 12,662 votes

SCRAPING. PART 1

BeautifulSoup: Used for parsing HTML content and extracting relevant data using tags, ids, CSS selectors and Xpath

Requests: Initially used to fetch static content from webpages

FIRST CHALLENGE-

BYPASS 10 REQUESTS LIMITATION

429 Too Many Requests

Wow, you're quite the enthusiast! Unfortunately, our server can't handle the high volume of requests. Moreover, we need to protect our website from malicious activities.

It looks like you've opened more pages in a short time than one can possibly read. If you're a regular user, please take a break, explore our other interesting content, and come back later.

If you're crawling our website, please note that it's against our terms of service. Scraping and stealing proprietary database information is illegal, so kindly cease such actions. Fragrantica content is only allowed for private browsing purposes.

Remember, if you attempt to scrape our website, we'll have no choice but to call **John Wick**.

PART 2. INTRODUCING WEBCRAWLING

SELENIUM

An open-source tool used for automating web browsers to perform tasks such as web scraping, testing web applications, and automating repetitive web-based processes. (with a use of WebDriver - firefox, chrome,)

- Handle Consent Forms fill
- Handle Pages navigation
- Handle filter setups
(for parallel batch scrape

- Random Pauses between actions
- Simulated Human-like mouse movements using B-spline interpolation (from the current mouse location to the object; stay within the view area

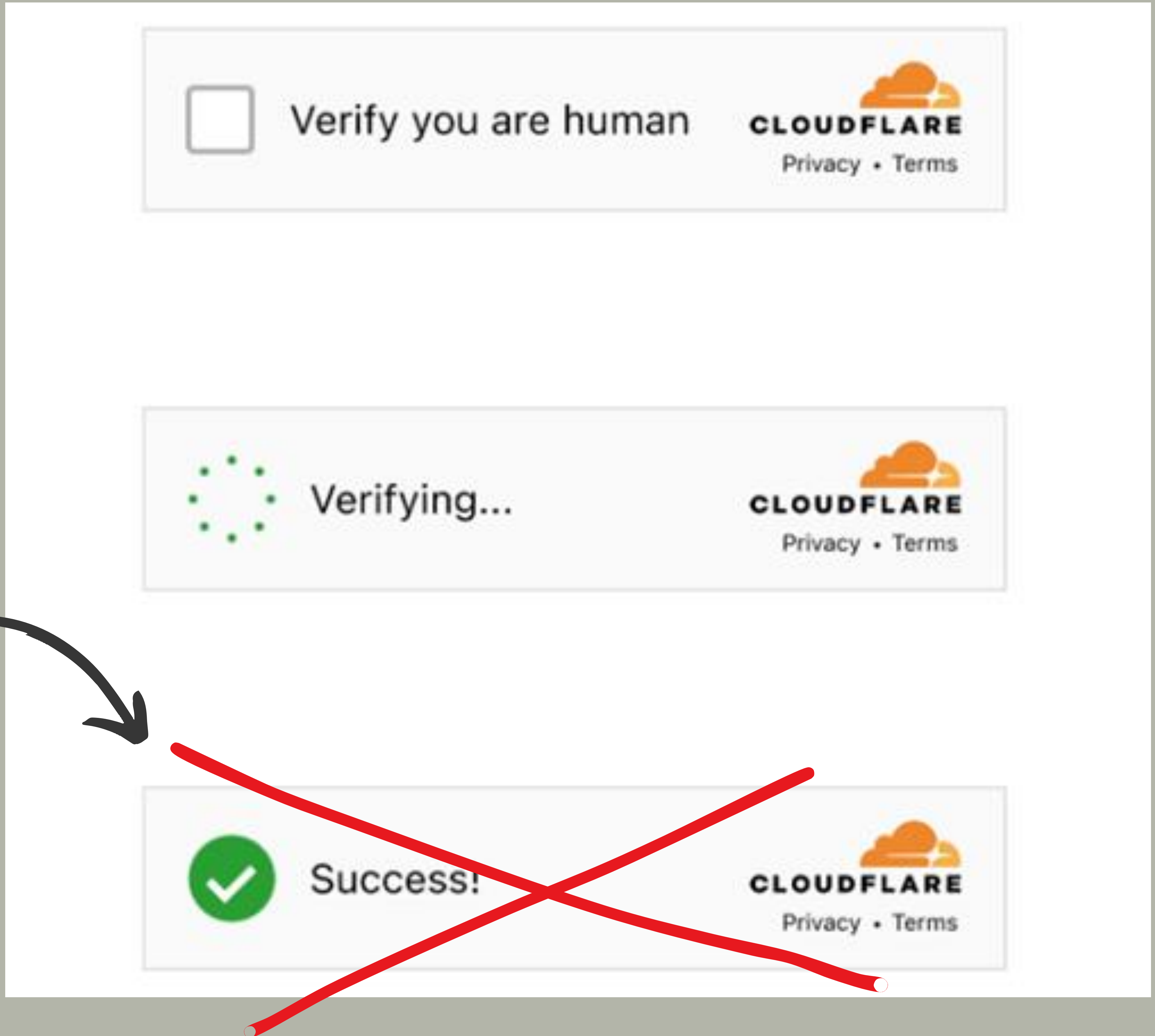
- Automatic subsequent clicks
- Random smooth scrolling along the pages (simulate page observation)
- + Some more random mouse movements

BYPASSING DETECTION - ACTIONCHANS, JS

We managed to bypass
the 10-request limit, however, we soon faced another issue...

SUCCESS! (PARTIAL)

CAPTCHA...



No success

PART 3. HANDLING CAPTCHA

Our attempts to resolve issues with free proxies, VPN use and settings updates were not successful

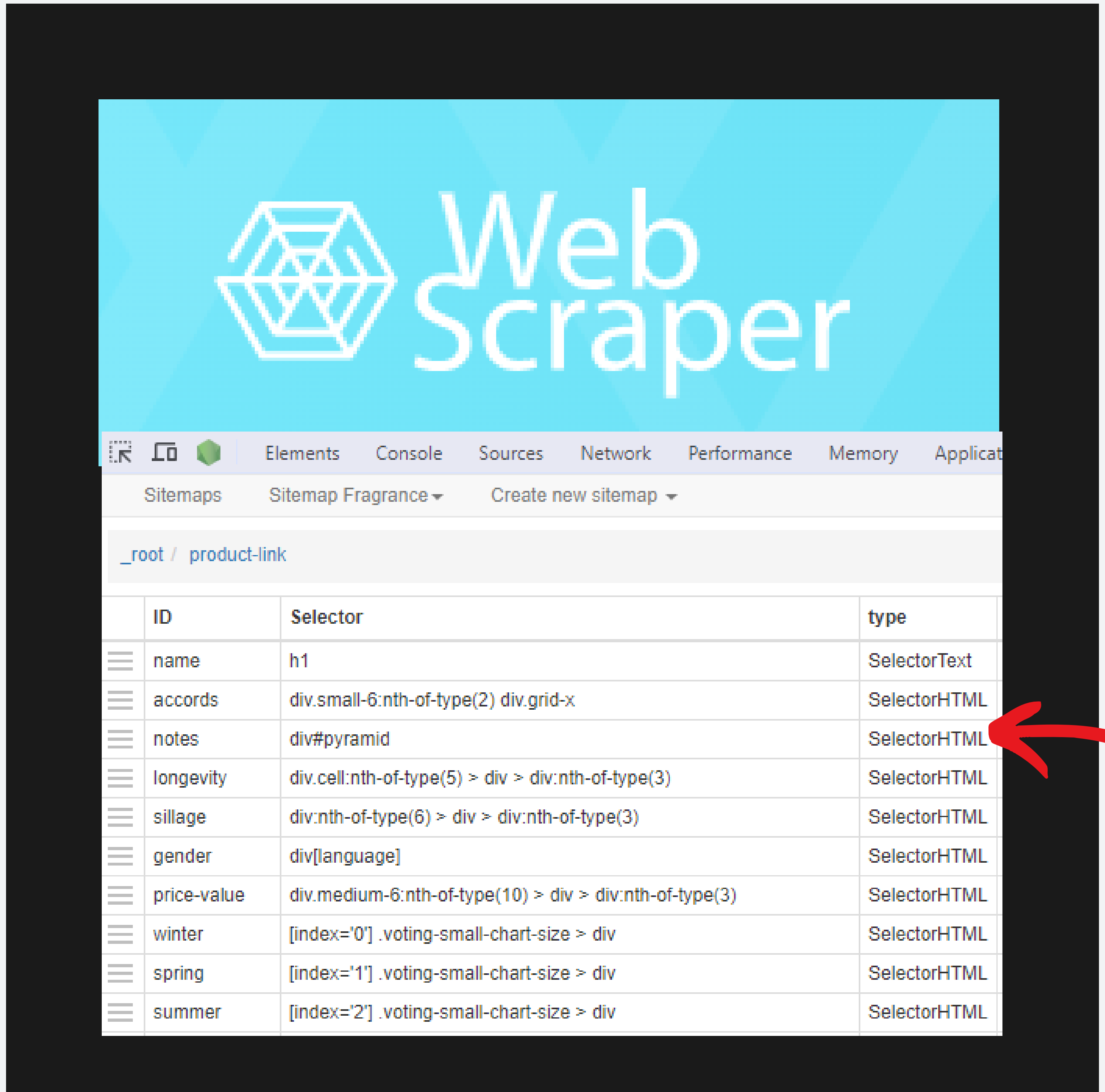
Undetected ChromeDriver: Used to avoid detection by anti-bot mechanisms on the website

IT worked!

After cleaning, as a result of web scraping, we succeeded in recovering almost 450 rows of data, which allowed us to start building ML models

PART 4. MORE DATA

We implemented a new approach to collecting data - use of 3-rd party applications with in-built rotating proxies



ID	Selector	type
name	h1	SelectorText
accords	div.small-6:nth-of-type(2) div.grid-x	SelectorHTML
notes	div#pyramid	SelectorHTML
longevity	div.cell:nth-of-type(5) > div > div:nth-of-type(3)	SelectorHTML
sillage	div:nth-of-type(6) > div > div:nth-of-type(3)	SelectorHTML
gender	div[language]	SelectorHTML
price-value	div.medium-6:nth-of-type(10) > div > div:nth-of-type(3)	SelectorHTML
winter	[index='0'] .voting-small-chart-size > div	SelectorHTML
spring	[index='1'] .voting-small-chart-size > div	SelectorHTML
summer	[index='2'] .voting-small-chart-size > div	SelectorHTML

process of creating sitemaps required for scraping (imported using json)

This method allowed us to increase the total number of collected data instances to 5139! (before preprocessing)

DATA PREPROCESSING

DATA EXTRACTION SCHEME

- *.csv*-file with html elements -> *.csv*-file with numbers, lists and strings

<div style="border-radius: 0.2rem; height: 0.3rem; background: rgb(120, 214, 240); width: 38.7283%; opacity: 1;"></div>

winter

38.7283

DATA EXTRACTION RESULTS

Dataframe with 17 columns with numbers, lists and dictionaries

winter	middle notes	longevity
74.0864	[Benzoin, Woodsy Notes, Virginia Cedar, Iris, ...	{'very weak': 7, 'weak': 16, 'moderate': 55, '...
72.1550	[Cinnamon, Spices, Ginger, Gingerbread]	{'very weak': 43, 'weak': 66, 'moderate': 307,...
57.7830	[Magnolia, Violet, Rose, Jasmine]	{'intimate': 125, 'moderate': 307, 'strong': 9...
53.4615	[Coconut]	{'intimate': 103, 'moderate': 222, 'strong': 8...
13.7500	[Osmanthus, Jasmine, Geranium]	{'very weak': 38, 'weak': 38, 'moderate': 58, ...

NAN VALUES & EMPTY COLUMNS

- Delete the columns with lots of NaN values .
- Delete the rows with NaN values
- Drop name column

```
nan_counts = perfumes.isna().sum()

print(nan_counts)
```

```
name               0
accords           36
longevity        822
sillage         2982
gender             0
price-value     5139
winter            33
spring            33
summer            33
fall              33
day               33
night             33
votes              6
rating             6
top notes       1785
middle notes      36
base notes      1815
dtype: int64
```

NOTES TRANSFORMATION

- The total number of notes was calculated (approx. 750)
- Only the most popular notes were leftt
- Initial lists with 750 notes -> Roughly 300 new columns of individual notes

```
perfumes_shortened.at[0, 'top notes']
```

```
"['Tea', 'Star Anise', 'Bergamot']"
```

top_Cranberry	top_Ivy	top_Italian_Mandarin	top_Sea_Notes	top_Cedar	top_Guava	top_Passionfruit	top_Narcissus	top_Fennel
0	0	0	0	0	0	0	0	0

LONGEVITY, SILLAGE, SEASON TRANSFORMATION

- Python dictionaries with number of votes as values -> new columns with keys as names
- Divide each entry by the total sum of the dictionary values -> range [0, 1]

```
perfumes_shortened.at[0, 'sillage']
```

```
'{'intimate': 156, 'moderate': 416, 'strong': 126, 'enormous': 119}'
```

sillage_intimate	sillage_moderate	sillage_strong	sillage_enormous
0.190942	0.509180	0.154223	0.145655

FINAL SHAPE

- 2598 samples
- 363 features
- 1 target

```
perfumes_shortened.shape
```

```
(2598, 364)
```

MODELS COMPARISON

PROBLEM DESCRIPTION

- Target value is a continuous variable [0, 5] -> **regression** should be used

- Dataset has more than *300 features* -> model should be able to predict the *target* using them.

LINEAR REGRESSION

Challenges

:

- Requires linearity of the data for good results
- Sensitive to outliers
- Requires *feature engineering* for non-linear relations (data contains >300 variables)

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 0.1294925260361281
RMSE: 0.1613567010542687
R-squared: 0.39429067608276547
```

KNN REGRESSION

Challenges

- “Black box” predictions
- Prone to overfitting with small number of neighbours
- Strugles with high dimensions

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 0.1256115384615385
RMSE: 0.16051855871614254
R-squared: 0.40056685943955417
```

RANDOM FOREST REGRESSION

Advantages:

- Captures non-linear relationships
- Robust to outliers
- Handles high dimensions well

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 0.04173846153846158
RMSE: 0.08424113740192044
R-squared: 0.8349031866853031
```

THANK'S FOR WATCHING