Faculty of Sciences of Agadir,
Excellence Master - Data Analytics and Artificial Intelligence (ADIA)

Module: Cybersecurity

Project - N°2

Password Strength Analysis & Weakness Detection Model

Prepared by:

KINAD Kawtar — JIRARI Inass

KHAIR Latifa — IZIKKI Hajar

Supervised by:

Pr. BOUGHROUS Moncef

Academic Year: 2025-2026

Table of Contents

List of Figures

I. Introduction and Context

1.1. Introduction

Despite the advent of biometrics and multi-factor authentication, the password remains our primary defense against cyberattacks. Unfortunately, this bulwark is often fragile. Our massive reliance on this mechanism, coupled with our human behaviors (reuse, simplicity) and overly permissive creation rules, creates the weakest link in cybersecurity. If we continue to choose convenience, our systems, no matter how sophisticated, are only as strong as the weakest password protecting them.

The risk is not theoretical: security leader statistics confirm that over 80% of authentication breaches are due to this vulnerability. This failure is a direct threat to the Confidentiality, Integrity, and Availability of our data (the CIA Triad). Meanwhile, attackers are arming themselves with ultra-efficient techniques like optimized brute force and credential stuffing, methods that directly exploit the predictability of our choices to infiltrate systems on a massive scale.

In the face of this arms race, traditional methods based solely on length or static rules are no longer sufficient. These policies fail to measure what truly matters: a password's real complexity and mathematical unpredictability. This is why the Data Analysis & AI approach intervenes as an essential evolution. Our project is part of this vision: no longer imposing arbitrary rules, but scientifically understanding human weaknesses. By analyzing the RockYou.txt dataset, we are developing an intelligent Machine Learning model to predict and classify password strength, thereby transforming the problem of vulnerability into a proactive and quantitative defense solution.

1.2. Problem Statement and Objectives

Problem Statement

Within a modern, mid-sized company with nearly 900 employees, the Security Operations Center (SOC) is observing a worrying increase in alerts related to unauthorized access attempts. Several internal accounts have been compromised in recent months. Upon analysis, the SOC team found that **85% of the compromised accounts** used weak passwords, often linked to predictable human habits: dates of birth, names of relatives, repetitions, or simple numerical sequences.

Despite an internal policy mandating a minimum length of 8 characters, users continue to choose passwords that are easy to remember but extremely vulnerable to **modern attacks**: dictionary attacks using millions of already compromised passwords, *credential stuffing* attempts from external leaks, and GPU-accelerated brute force attacks, now capable of testing hundreds of billions of combinations per second.

The critical issue lies in the fact that the company's current tools only assess password strength based on its length and the symbolic presence of a number or an uppercase letter. As these criteria are overly simplistic, they fail to measure the password's **real complexity** or **unpredictability**. It is therefore imperative to implement a quantitative approach capable of evaluating the **actual risk** of compromise.

Project Objectives

To address this operational challenge for the SOC, the main objectives of this project are:

1. **Quantify Vulnerability:** Analyze the RockYou.txt dataset to extract, calculate, and label password strength using advanced metrics (Entropy, Character Classes, Common Patterns).

2. **ML Model Development:** Train, evaluate, and validate a classification model capable of accurately predicting the strength level (Weak, Medium, Strong) of a new password.

3. **Actionable Recommendations:** Leverage the model's Feature Importance analysis to provide precise and actionable advice to the SOC regarding optimal password acceptance thresholds.

II. Theoretical Foundations and Threat Analysis

2.1. The Password and the CIA Triad

The security of an information system rests on three fundamental pillars, known as the CIA Triad: **Confidentiality, Integrity, and Availability**. The password, as the primary authentication mechanism, is the key that controls access to these three pillars.

- **Confidentiality:** A weak or compromised password is a direct violation of confidentiality, as it allows a malicious actor to access information that should remain private. The role of our model is to ensure the password is unpredictable enough so that this confidentiality is not breached by a simple guess.

- **Integrity:** Once the account is compromised via a weak password, the attacker can alter, modify, or delete data. The password's weakness thus becomes a threat vector to the information's integrity.

- **Availability:** Massive password cracking attempts (brute force) can trigger account lockout mechanisms (anti-brute force). These lockouts, while defensive, can result in denial of service for the legitimate user, thus threatening availability.

The weakness of a password not only affects the individual user but compromises the overall security posture of the organization with respect to the three fundamental CIA principles.

2.2. Types of Password Attacks

a) Brute-Force Attack

Principle: A brute-force attack automatically tests all possible combinations of characters until it finds the correct password.
It relies entirely on the size of the search space to which the password belongs.

Resistance to this attack depends directly on **Shannon Entropy**, calculated using the formula:

$$H = L \times \log_2(N)$$

where: **L** = length of the password, **N** = size of the character set used

A password with low entropy (i.e., low complexity) can be cracked in a few milliseconds, especially since modern GPUs can test several **billion** combinations per second. In our project, entropy is one of the key indicators used to quantify the robustness of a password against brute-force attacks.

b) Dictionary Attack

Principle: Unlike brute force, a dictionary attack does not explore all possible combinations. Instead, it exploits human predictability by testing lists of common passwords known as dictionaries. These lists include:

- first names and birth dates,
- simple words (e.g., *admin*, *love*, *football*),
- repetitions (*aaaa*, *1111*),
- logical sequences (*123456*, *qwerty*),
- passwords previously leaked during data breaches.

The **RockYou.txt** dataset used in our project is one of the most popular dictionaries among attackers. It contains millions of real passwords used by users before being compromised, making it extremely effective for this type of attack

c) Credential Stuffing

Principle: Credential stuffing consists of automatically using stolen credentials (email + password) obtained from a data breach on one service to attempt logging into other services. This attack is highly effective because most users reuse the same password across multiple platforms.

Associated Risks:

- If a password appears in a data breach, it is immediately tested on hundreds of other websites.
- The attack is fully automated and extremely fast.
- It does not require cracking the password, simply knowing it is enough.

III. Methodology and Data Preparation

3.1. Architecture and Tools

The entire project was developed following the standard data science lifecycle, utilizing a set of proven technologies for massive data analysis and Machine Learning.

- **Programming Language: Python** was used for its robustness and rich set of libraries for data analysis and cybersecurity.
- **Development Environment:** The work was conducted in **Jupyter Notebooks** (via Google Colab), ensuring the traceability of the analysis and modeling steps.
- **Key Libraries:**
    - **Pandas** for the manipulation, cleaning, and feature engineering of the dataset.
    - **NumPy** for optimized numerical operations, particularly the vectorized calculation of Shannon Entropy.
    - **Scikit-learn** for stratified sampling, training, and evaluation of Machine Learning models (Classification Report, Confusion Matrix).
    - **Matplotlib** for result visualization (length distribution, feature importance).

3.2. Dataset: Description of the RockYou.txt file

The project relies on the analysis of the **RockYou.txt** dataset, a reference in password security research.

- **Source and Nature:** This file is a major corpus of real passwords compromised during a data breach. The use of this type of data is justified because it accurately reflects the human behavioral weaknesses exploited by cybercriminals.
- **Dataset Size:** The file was initially loaded, totaling **14,344,391** passwords. After cleaning and deduplication, the working dataset contained **14,343,755** unique entries.

3.3. Preprocessing and Sampling

In order to prepare the data for modeling and ensure efficient execution given the dataset's size, several preprocessing steps were necessary.

- **Raw Data Cleaning:**
    - **Deduplication:** All duplicate passwords were removed to retain only unique entries.

- **Handling Missing/Empty Values:** Empty or null-length entries were eliminated to ensure the consistency of Entropy and Length calculations.

- **Justification and Implementation of Stratified Sampling:**
  - Training on the entire 14 million lines is resource-intensive and time-consuming.
  - For the modeling phase, a representative sample of **500,000 lines** was extracted.
  - **Stratified sampling** was used to ensure that the distribution of the target classes (**Weak, Medium, Strong**) in the 500,000-line sample is **identical** to that of the complete 14 million dataset.

IV. Feature Engineering and Labeling

The core of our approach lies in transforming raw passwords into numerical characteristics (Features) that quantify their resistance and predictability.

4.1. Feature Engineering Development

We created **8 key features**, grouped into three categories, to evaluate the complexity and unpredictability of each password.

Feature Name	Category	Description
length	Length	Total number of characters.
count_lower, count_upper, count_digit, count_symbol	Counters	Count of lowercase, uppercase letters, digits, and symbols.
char_classes	Complexity	Sum of character classes present (from 1 to 4).
shannon_entropy	Strength	Measure of mathematical unpredictability.
is_common_pattern	Contextual	Boolean indicator (1/0) signaling the presence of sequences, repetitions, or dictionary words.

- **Complexity Metrics (char_classes):** This metric is crucial as it reveals the diversity of the alphabet used. Users choosing weak passwords often limit themselves to a single class (e.g., 123456 = 1 class) or two. The char_classes feature is a score ranging from 1 (only one character category present) to 4 (all categories present).
- **Unpredictability Metrics (shannon_entropy):** Entropy is the mathematical measure of a password's strength against brute-force attacks. It was calculated for each password using the formula:

  Entropy(bits) = Length x log2(Used Alphabet size)

  A higher value means the search space is larger and the password is more resistant.
- **Contextual Metrics (is_common_pattern):** This Boolean variable was created using regular expressions (Regex) to detect the presence of common numerical/alphabetical sequences (e.g., 12345, qwerty) or well-known passwords (e.g., password, iloveyou), revealing an immediate behavioral weakness

4.2. Target Variable Definition (Labeling)

To train a supervised Machine Learning model, it was necessary to create a target variable (*y*) called **strength_label**.

- **Labeling Logic:** The label was defined by applying a set of strict threshold rules based on the created features, simulating the logic of an advanced security policy system.
- **Class Definition:**
  - 0: **Faible (Weak):** Password containing a **common pattern** (is_common_pattern = 1), **OR** having **only one character class** present (char_classes ≤1), **OR** having a **very short length** (length ≤ 6), **OR** low **Entropy** (shannon_entropy < 35.0).
  - 2: **Fort (Strong):** Password having **4 character classes**, **AND** a **minimum length** (length ≥ 12), **AND high Entropy** (shannon_entropy ≥ 60.0).
  - 1: **Moyen (Medium):** All other passwords that satisfy neither the "Weak" nor the "Strong" criteria.

This trinary target variable (0, 1, 2) then enables the training of the Random Forest model.

V. Analysis and Modeling

5.1. Statistical Analysis of Weaknesses

This section uses the created features to statistically quantify the level of vulnerability present in the RockYou.txt dataset even before model training. It serves as a statistical justification for the problem statement.

Distribution of Strength Classes: Applying the labeling rules (defined in Section 5.2) to the 500,000-password sample reveals an extremely imbalanced strength distribution, highlighting the careless behavior of users.

Strength Class	Label	Percentage in Dataset	Statistical Description
Weak	0	**52.75%**	Passwords containing common patterns or very low entropy.
Medium	1	**47.17%**	Passwords that satisfy neither the strict Strong nor the strictly Weak criteria.
Strong	2	**0.07%**	Passwords satisfying the strict thresholds (4 classes, length ≥12 , Entropy ≥ 60 bits).

Nearly **99.92%** (52.75% + 47.17%) of passwords are classified as Medium or Weak. This imbalance confirms that weakness is the norm and security is the exception, reinforcing the need for intelligent tools to reject these at-risk credentials.

5.2. Analysis of Weakness Factors

The analysis of password composition highlights the behavioral factors that lead to this generalized weakness.

5.2.1. Distribution of Character Classes

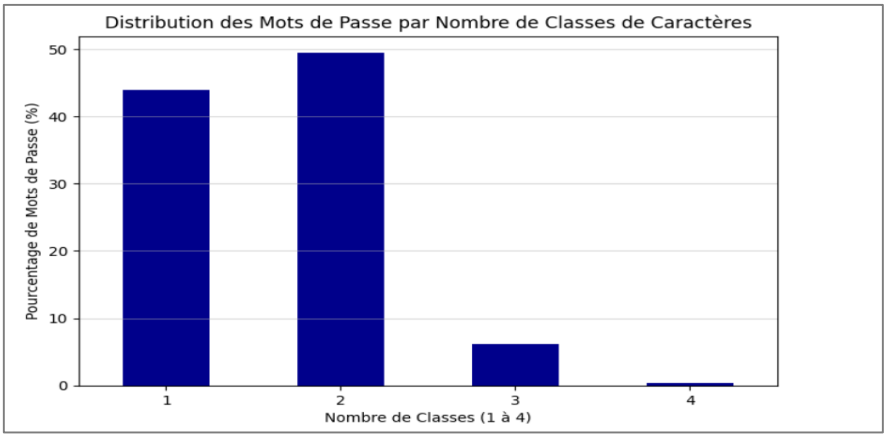


Figure 1:Distribution of Passwords by Number of Character Classes

Number of Classes	Percentage of Passwords
1 Class	43.98%
2 Classes	49.47%
3 Classes	6.18%
4 Classes	0.37%

The chart demonstrates that the vast majority of users (nearly **93.45%**) limit themselves to only one or two character classes (e.g., digits only, or lowercase and digits). Passwords using the **4 classes** necessary for robust security represent only a tiny minority (**0.37%**). This statistical finding justifies that the **Complexity metric (char_classes)** will be a dominant predictive factor in our model, as it directly targets human negligence.

5.2.2. Entropy by Number of Character Classes

The analysis of class distribution previously demonstrated that users avoid complexity. This section proves that this negligence has a direct and exponential impact on the theoretical resilience of passwords. Shannon Entropy (**shannon_entropy**), measured in bits, serves as the ground-truth metric for evaluating this resistance.

Character Class	Total Count	Mean Entropy (bits)	Median Entropy (bits)
1 Class	2,352,556	27.69	26.58
2 Classes	4,161,950	49.69	45.60
3 Classes	7,322,126	52.55	47.63
4 Classes	507,123	70.09	65.55

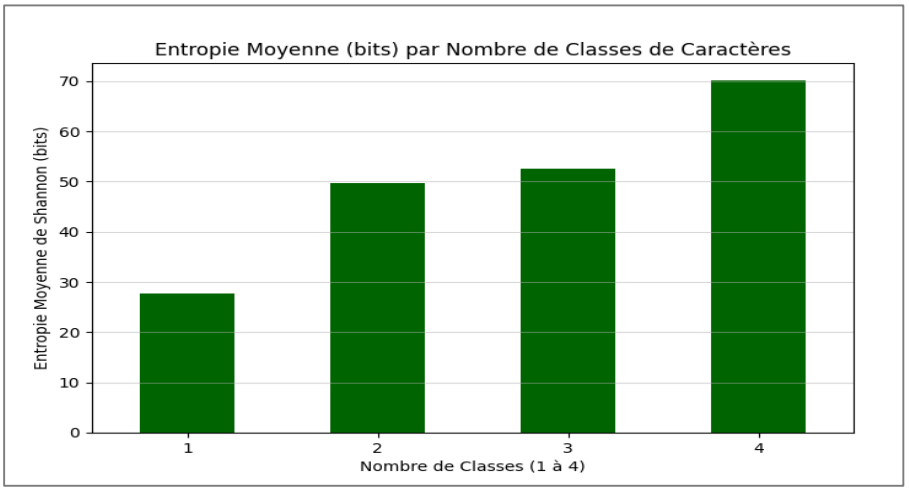


Figure 2:Entropy by Number of Character Classes

Analysis of the Entropy and Security Impact: The group entropy analysis confirms the critical role of the number of classes. Passwords utilizing 1, 2, or 3 classes are all well **below** the **60-bit** security threshold. The Exponential Impact of Symbols is evident, as the Mean Entropy jumps by 17.54 bits (from 52.55 to 70.09 bits) with the inclusion of the 4th class. The figure below illustrates this key finding.

5.3. ML Model Selection and Evaluation

a) Model Selection

Justification for Multi-Class Classification

Given that the target variable strength_label is trinary (0, 1, 2), we opted for a supervised multi-class classification model.

Comparison Table Presentation

To identify the best architecture, we compared the performance of three common models on our sampled dataset (70% Training / 30% Testing).

Model	Accuracy	F1-Score (Macro)	Time (s)
Random Forest (Baseline)	**1.0000**	**1.0000**	**9.49**
Logistic Regression (Baseline)	0.9765	0.7491	28.37
Support Vector Machine (Linear)	0.9321	0.6216	5.26

The **Random Forest** model demonstrated a perfect ability to reproduce the labeling rules defined by Feature Engineering (F1-score of 1.00). Although the model is deterministic (it perfectly replicates the rules we coded), it is chosen not only for its superior performance but also for its excellence in **Feature Importance** analysis and its natural ability to handle the non-linear interactions between the 8 features.

b) Overall Model Performance

Performance Table Presentation

The Random Forest model was evaluated on the test set (150,000 rows) with the following results:

Class	Precision	Recall	F1-Score	Support
0: Weak	1.00	1.00	1.00	79130
1: Medium	1.00	1.00	1.00	70758
2: Strong	1.00	1.00	1.00	112
Accuracy	**1.00**			150000

Detailed Analysis of the Confusion Matrix

The Confusion Matrix confirms the perfect performance of the model:

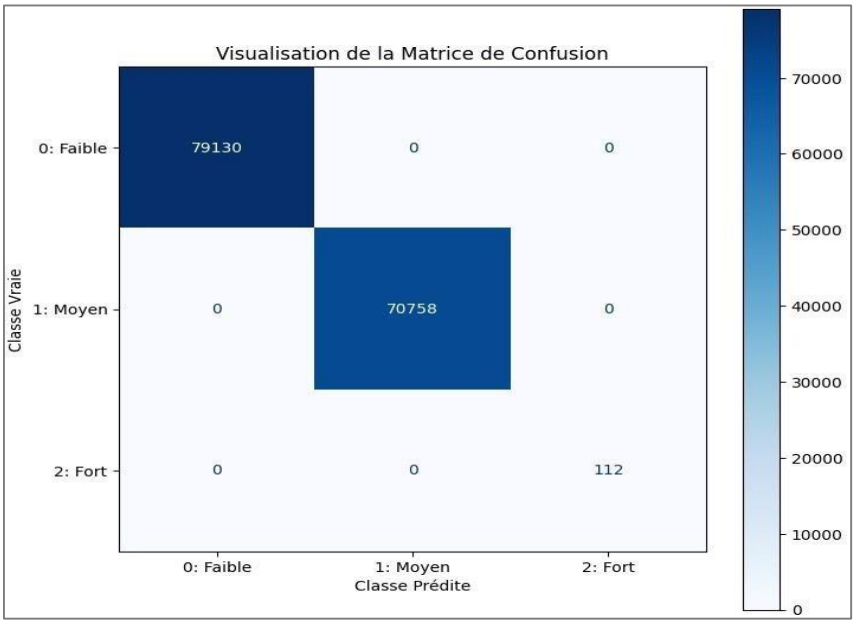


Figure 3:Confusion Matrix

The total absence of classification errors (off-diagonal values are zero) means that the Random Forest model perfectly assimilated the classification logic based on our Entropy and Complexity metrics.

5.4. Interpretation of Results

a) Feature Importance

The strength of Random Forest lies in its ability to assign a weight of importance to the features. The analysis of this importance reveals which factors contribute the most to the classification decision (Weak/Medium/Strong).

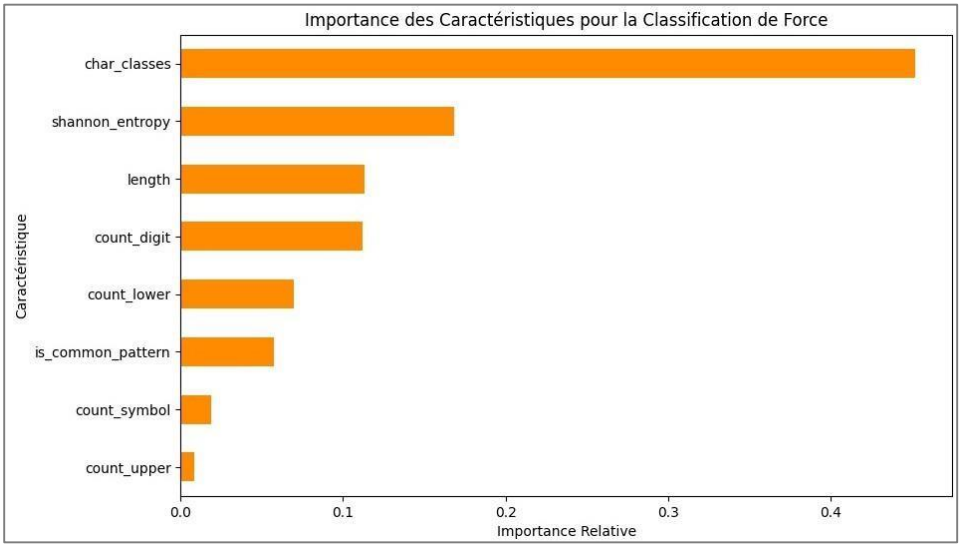


Figure 4:Feature Importance for Strength Classification

Feature	Relative Importance (×100)
char_classes (Complexity)	**45.26**
shannon_entropy (Entropy)	**16.82**
length (Length)	11.31
count_digit (Digit Count)	11.19
is_common_pattern (Common Pattern)	5.74

The importance analysis confirms the central hypothesis of this project: the features derived from complexity and unpredictability are the dominant factors. **Character Classes (≃ 45.26 %)** and **Shannon Entropy (≃ 16.82%)** are collectively far more important than length alone (≃ 11.31%).

b) Implications

The model validates that the **quality** of the password is superior to its **quantity**. The implications are clear for the SOC: simply mandating a minimum length is an ineffective defense strategy. To bolster security, the focus must be on:

- Increasing **Complexity** (by enforcing the use of all 4-character classes).
- Increasing **Unpredictability** (by imposing a high Entropy threshold).

This forms the basis of the actionable recommendations in the final chapter.

VI. Conclusion and Recommendation

6.1. Actionable Recommendations for an SOC

The primary goal of this ADIA project was to provide the Security Operations Center (SOC) with quantifiable information to strengthen password policies, moving beyond arbitrary length rules. The results of the Feature Importance analysis from the Random Forest model identified the most critical risk factors, justifying the shift from a length-based policy to one based on the **quality of features**.

6.1.1. Synthesis of Results for Operational Use

The analysis demonstrated that **Complexity** and **Unpredictability** are collectively the best predictors of password strength. Conversely, is a secondary factor.

This justifies the transition from a security policy based on quantity to one based on the **quality of characteristics**.

6.1.2. Functional Model Validation (Demonstration Scenarios)

To validate the operational applicability of the model, a simulation tool was developed, integrating the trained Random Forest model and its key metrics. This interface allows the SOC to test new passwords and receive immediate classification based on the recommended thresholds.

i. Scenario 1: Weakness Due to Pattern Failure (Class 0)

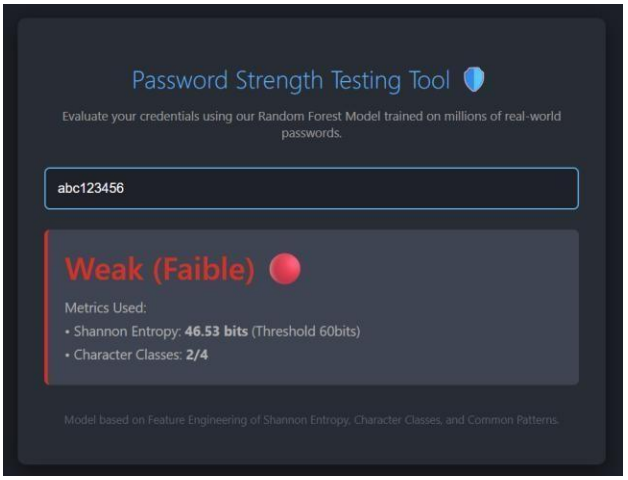


Figure 5:Weak password Test

Failure: This password contains sequential patterns (**abc** and **123456**) flagged by the **is_common_pattern** feature. The model instantly rejects it regardless of the moderate entropy (46.53 bits).

ii. Scenario 2: Medium Strength Due to Complexity Gap (Class 1)

Figure 6:Medium password Test

Failure: Although long (Length=15) and without a common pattern, it uses only **3/4 Character Classes** (lacking a symbol). This insufficient complexity keeps its calculated entropy below the strict 60 -bit threshold, validating the necessity of the 4-class mandate.

iii. Scenario 3: Strong Success (Class 2)

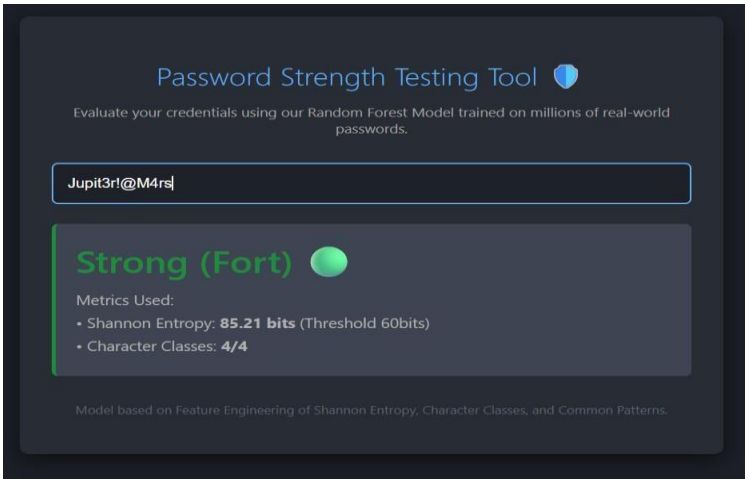


Figure 7:Strong password Test

Success: This password complies with all policy recommendations. The Entropy (**85.21 bits**) is far above the **60 -bit threshold**, and it achieves **4/4 Character Classes**. The model validates that this password is theoretically resilient against modern attacks.

6.1.3. Recommendations:

- **Priority - Based on Complexity:**
  - **Justification:** The char_classes factor is the most important **(**≃ 45.26 % importance). The model proved that 93.45% of compromised passwords used only 1 or 2 character classes.
  - **Action for the SOC: Enforce the strict use of all 4 character classes.** The password creation system must reject any credential that does not use lowercase, uppercase, digits, and symbols, regardless of its length.
- **Quantitative - Based on Unpredictability**
  - **Justification:** The Shannon entropy factor is the second determining factor (≃ 16.82% importance), directly measuring resistance to brute force.
  - **Action for the SOC: Establish a minimum Entropy threshold of 60 bits** instead of relying solely on length. This threshold, derived from our analysis, guarantees sufficient theoretical resistance against GPU-accelerated brute force attacks.
- **Contextual - Based on Predictability**
  - **Justification:** The **is_common_pattern** factor allows blocking behavioral weaknesses not detected by entropy alone.
  - **Action for the SOC: Implement enhanced blacklisting filtration.** The developed ML model must be used to classify new passwords, ensuring they do not match the weakness patterns learned from the RockYou.txt dataset (sequences, dictionary words) which fuel Credential Stuffing attacks.

Summary of Project Contribution

This project demonstrated how the Data Analysis & Artificial Intelligence approach can significantly enhance the evaluation of password strength within a professional cybersecurity context. By analyzing real-world compromised passwords from the RockYou dataset, we were able to move beyond traditional security policies based solely on minimum length requirements and adopt a more rigorous, data-driven perspective on password robustness. This analytical process revealed clear structural weaknesses in human-generated passwords and showed the limitations of conventional heuristics that do not account for true unpredictability.

Through systematic feature engineering, the project introduced key indicators of password strength, including character class diversity, Shannon entropy, and the detection of common patterns such as repetitions and predictable sequences. These features proved essential for modeling both the structural complexity and the unpredictability of passwords. Building upon these metrics, we developed a Random Forest classifier capable of reliably distinguishing weak, medium, and strong passwords. The model demonstrated excellent performance and provided interpretable insights through the analysis of feature importance, confirming that complexity and unpredictability are the most influential components of password strength, well beyond simple length.

Beyond the analytical dimension, the project also provides tangible operational value. The exported model can be integrated into a SOC environment to support real-time password evaluation during account creation or password renewal. By relying on quantifiable indicators rather than arbitrary rules, the SOC can adopt a more effective and modern password policy, strengthening defenses against automated attacks such as brute-force attempts and credential-stuffing activities. This demonstrates the relevance and practical impact of applying ADIA methods to cybersecurity challenges.

Looking forward, several developments can extend the usefulness and longevity of this work. A first direction involves deploying the model in production through an API to enable real-time password classification. Another perspective is the creation of a continuous monitoring pipeline capable of incorporating newly leaked password datasets so that the model remains aligned with emerging attack patterns. Finally, testing the model against additional compromised-password datasets would help assess its generalizability and reinforce its reliability in various security contexts.

References

[1] Statistiques d'Industrie

Verizon (2023) *2023 Data Breach Investigations Report (DBIR).*

[2] Fondation Théorique de l'Information

Shannon, C.E. (1948) 'A mathematical theory of communication', *Bell System Technical Journal*, 27(3), pp. 379–42.

[3] Jeu de Données d'Analyse

Skullsecurity (n.d.) *RockYou Password Leak Dataset.*

[4] Algorithme de Modélisation

Breiman, L. (2001) 'Random forests', *Machine Learning*, 45(1), pp. 5–32.

Appendices

2. Nettoyage et Déduplication

```
# Vérification rapide du nombre de valeurs uniques
unique_count = df['password'].nunique()
print(f"Nombre de mots de passe uniques après chargement : {unique_count}")

# S'il y a des doublons (si unique_count < len(df)), dédupliquez :
if len(df) != unique_count:
    df.drop_duplicates(subset=['password'], inplace=True)
    print(f"\n Taille du Dataset après déduplication : {len(df)}")

# Suppression des entrées vides ou de très mauvaise qualité (par exemple, longueur 0)
df['password'].replace('', pd.NA, inplace=True)
df.dropna(inplace=True)
df = df[df['password'].str.len() > 0]

print(f"\n Taille du Dataset après la Suppression des entrées vides : {len(df)}")
```

```
Nombre de mots de passe uniques après chargement : 14343756

 Taille du Dataset après déduplication : 14343756
/tmp/ipython-input-4182191703.py:11: FutureWarning: A value is trying to be set on a copy of a D
The behavior will change in pandas 3.0. This inplace method will never work because the intermed

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}

  df['password'].replace('', pd.NA, inplace=True)

 Taille du Dataset après la Suppression des entrées vides : 14343755
```

📊 Deuxième Étape : Analyse Exploratoire de Base (EDA)

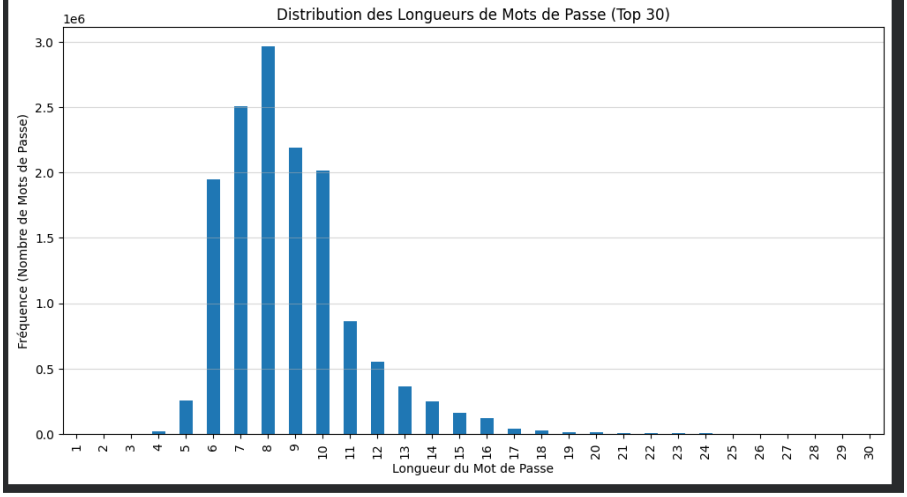
Une fois que nous données sont propres, on effectue une analyse exploratoire pour identifier les tendances de base de la faiblesse des mots de passe.

1. Analyse de la Longueur du Mot de Passe

```
# Créer une colonne pour la longueur du mot de passe
df['length'] = df['password'].apply(len)

# Statistiques descriptives
print("\nStatistiques sur la longueur des mots de passe:")
print(df['length'].describe())

# Visualisation (Critique pour votre rapport !)
plt.figure(figsize=(12, 6))
df['length'].value_counts().sort_index().head(30).plot(kind='bar')
plt.title('Distribution des Longueurs de Mots de Passe (Top 30)')
plt.xlabel('Longueur du Mot de Passe')
plt.ylabel('Fréquence (Nombre de Mots de Passe)')
plt.grid(axis='y', alpha=0.5)
plt.show()

# Conclusion à tirer : Quelle est la longueur la plus courante ?
```

```
Statistiques sur la longueur des mots de passe:
count    1.434376e+07
mean     8.754378e+00
std      2.917473e+00
min      1.000000e+00
25%      7.000000e+00
50%      8.000000e+00
```

```
Name: length, dtype: float64
```



🛠️ Troisième Étape : Ingénierie des Caractéristiques (Feature Engineering)

L'objectif est de quantifier la force d'un mot de passe en analysant sa composition. Pour cela, nous allons créer de nouvelles colonnes dans votre DataFrame, chacune représentant une métrique de sécurité clé.

1. Création des Caractéristiques de Composition (Complexité)

déterminer le niveau de mixité des caractères dans chaque mot de passe. C'est le principal facteur humain de faiblesse (les utilisateurs n'utilisent souvent que des minuscules et des chiffres).

```python
import numpy as np
import pandas as pd
import re

# Votre DataFrame 'df' est déjà chargé ici avec la colonne 'password'

print("Démarrage du Feature Engineering vectorisé (Corrigé)...")

# --- 1. Calculer les caractéristiques de complexité (Compteurs) ---
# Minuscules : NE PAS utiliser IGNORECASE
df['count_lower'] = df['password'].str.count(r'[a-z]')

# Majuscules : NE PAS utiliser IGNORECASE
df['count_upper'] = df['password'].str.count(r'[A-Z]')

# Chiffres : NE PAS utiliser IGNORECASE (bien que cela n'aurait pas d'impact ici)
df['count_digit'] = df['password'].str.count(r'[0-9]')

# Symboles (tout ce qui n'est pas alphanumérique)
# Ici, IGNORECASE n'a pas d'impact, mais nous le retirons pour la cohérence
df['count_symbol'] = df['password'].str.count(r'[^a-zA-Z0-9]')

# 2. Créer les indicateurs booléens (has_*) et la longueur (length)
# ... Le reste du code est correct, car il utilise les comptes précis ...
df['has_lower'] = np.where(df['count_lower'] > 0, 1, 0)
df['has_upper'] = np.where(df['count_upper'] > 0, 1, 0)
df['has_digit'] = np.where(df['count_digit'] > 0, 1, 0)
df['has_symbol'] = np.where(df['count_symbol'] > 0, 1, 0)
df['length'] = df['password'].str.len()

# 3. Caractéristique composite clé: Nombre de classes de caractères
df['char_classes'] = df['has_upper'] + df['has_lower'] + df['has_digit'] + df['has_symbol']

print("Fin de la correction et du Feature Engineering!")

# Test de vérification :
print(df[['password', 'length', 'count_lower', 'count_upper', 'count_digit', 'char_classes']].head(5).to_string())
```

```
Démarrage du Feature Engineering vectorisé (Corrigé)...
Fin de la correction et du Feature Engineering!
     password  length  count_lower  count_upper  count_digit  char_classes
0      123456       6            0            0            6             1
1       12345       5            0            0            5             1
2   123456789       9            0            0            9             1
3    password       8            8            0            0             1
4    iloveyou       8            8            0            0             1
```

```python
df.head(10)
```

	password	length	count_lower	count_upper	count_digit	count_symbol	has_lower	has_upper	has_digit	has_symbol	char_classes
0	123456	6	0	0	6	0	0	0	1	0	1
1	12345	5	0	0	5	0	0	0	1	0	1
2	123456789	9	0	0	9	0	0	0	1	0	1
3	password	8	8	0	0	0	1	0	0	0	1
4	iloveyou	8	8	0	0	0	1	0	0	0	1
5	princess	8	8	0	0	0	1	0	0	0	1
6	1234567	7	0	0	7	0	0	0	1	0	1
7	rockyou	7	7	0	0	0	1	0	0	0	1
8	12345678	8	0	0	8	0	0	0	1	0	1

2. Entropie (Métriques de Force)

L'entropie est une mesure mathématique cruciale pour déterminer la résistance d'un mot de passe aux attaques par force brute (combinaisons aléatoires). Elle mesure le niveau d'incertitude.L'Entropie de Shannon (H) se calcule ainsi :

$$\text{Entropie (bits)} = \text{Longueur} \times \log_2(\text{Taille de l'Alphabet})$$

Où la "Taille de l'Alphabet" est le nombre de classes de caractères différentes utilisées dans le mot de passe.

```
# 4. Calcul de l'Entropie de Shannon (basé sur la méthode vectorisée)
ALPHABET_SIZES = {
    'lower': 26,
    'upper': 26,
    'digit': 10,
    'symbol': 32
}

# Calcul de la taille de l'alphabet (N) pour chaque mot de passe (méthode vectorisée)
df['alphabet_size'] = (
    df['has_lower'] * ALPHABET_SIZES['lower'] +
    df['has_upper'] * ALPHABET_SIZES['upper'] +
    df['has_digit'] * ALPHABET_SIZES['digit'] +
    df['has_symbol'] * ALPHABET_SIZES['symbol']
)

# Application de la formule de l'Entropie de Shannon: L * log2(N)
# np.log2() est également vectorisé. Nous gérons le cas où alphabet_size est 0 ou 1
df['shannon_entropy'] = np.where(
    df['alphabet_size'] > 1,
    df['length'] * np.log2(df['alphabet_size']),
    0.0 # Entropie est 0 si l'alphabet est trop petit ou la longueur nulle
)

print("\nFeature Engineering terminé.")
print(df[['password', 'length', 'char_classes', 'shannon_entropy']].head())
print("\nStatistiques sur l'Entropie de Shannon:")

print(df['shannon_entropy'].describe())
```

```
Feature Engineering terminé.
    password  length  char_classes  shannon_entropy
0     123456       6             1        19.931569
1      12345       5             1        16.609640
2  123456789       9             1        29.897353
3   password       8             1        37.603518
4   iloveyou       8             1        37.603518

Statistiques sur l'Entropie de Shannon:
count    1.434376e+07
mean     4.247122e+01
std      1.781982e+01
```

3. L'analyse des classes de caractères et l'entropie (graphs)

```
import pandas as pd
import matplotlib.pyplot as plt

# --- A. Distribution des Classes de Caractères ---

# 1. Calcul de la distribution en pourcentage
class_distribution = df['char_classes'].value_counts(normalize=True).sort_index() * 100

print("\nDistribution en pourcentage du Nombre de Classes de Caractères:")
print(class_distribution.to_string())

# 2. Visualisation (pour le rapport)
plt.figure(figsize=(8, 5))
class_distribution.plot(kind='bar', color='darkblue')
plt.title('Distribution des Mots de Passe par Nombre de Classes de Caractères')
plt.xlabel('Nombre de Classes (1 à 4)')
plt.ylabel('Pourcentage de Mots de Passe (%)')
plt.xticks(rotation=0)
plt.grid(axis='y', alpha=0.5)
plt.show()
plt.savefig('char_classes_distribution.png')
plt.close() # Fermer la figure pour ne pas afficher dans le notebook

# --- B. Entropie Moyenne par Classe ---

# 1. Analyse de groupe : Entropie moyenne et médiane par nombre de classes
entropy_by_class = df.groupby('char_classes')['shannon_entropy'].agg(['count', 'mean', 'median', 'std'])

print("\nAnalyse de l'Entropie par Nombre de Classes de Caractères:")
print(entropy_by_class.to_string(float_format="%.2f"))

# 2. Visualisation de l'Entropie Moyenne par Classe (pour le rapport)
plt.figure(figsize=(8, 5))
entropy_by_class['mean'].plot(kind='bar', color='darkgreen')
plt.title("Entropie Moyenne (bits) par Nombre de Classes de Caractères")
plt.xlabel('Nombre de Classes (1 à 4)')
plt.ylabel('Entropie Moyenne de Shannon (bits)')
plt.xticks(rotation=0)
plt.grid(axis='y', alpha=0.5)
plt.show()
plt.savefig('entropy_by_class_mean.png')
plt.close()

print("\nL'analyse des classes de caractères et l'entropie sont terminées. Deux images ('char_classes_distributi
```

4 Caractéristiques de Modèle (Patterns)

```python
import numpy as np
import re
import pandas as pd

# Définition des patterns simples à rechercher
COMMON_PATTERNS = [
    # Séquences numériques (complétées par les séquences de 5 et 4)
    '123456', '234567', '345678', '456789', '654321', '54321', '4321', '321',
    '12345', '54321', # Ajout de la séquence de l'exemple du problème (12345)
    '111111', '222222', '000000', # Répétitions simples

    # Séquences alphabétiques (QWERTY/AZERTY)
    'qwerty', 'azerty', 'qwert', 'asdfg', 'zxcvb', 'mnbvc',

    # Mots de passe célèbres
    'password', 'admin', 'iloveyou', 'p@ssword', 'user', 'princess', 'rockyou'
]

# --- DÉTECTION 1 : Patrons listés (Regex) ---
# L'utilisation de '(?i)' rend la recherche insensible à la casse
pattern_regex = '(?i)' + '|'.join(COMMON_PATTERNS)

# Détection de répétition de n'importe quel caractère (lettre, chiffre, symbole) au moins 3 fois
# r'(.)\1{2,}' détecte '111', 'aaa', '$$$'
repetition_regex = r'(.)\1{2,}'


print("Démarrage de la détection de motifs améliorée...")

# Création d'une série booléenne pour chaque type de pattern

# 1. Détection des patrons listés
is_listed_pattern = df['password'].str.contains(pattern_regex, regex=True, na=False)

# 2. Détection des répétitions (ex: "aaa", "111")
is_repetition = df['password'].str.contains(repetition_regex, regex=True, na=False)


# La colonne finale 'is_common_pattern' est Vrai (1) si l'UNE ou l'AUTRE est Vraie.
df['is_common_pattern'] = np.where(
    np.logical_or(is_listed_pattern, is_repetition),
    1, 0
)

print("\nCaractéristique 'is_common_pattern' corrigée et améliorée ajoutée.")
print(df[['password', 'is_common_pattern', 'shannon_entropy']].head(10).to_string())
```

5. Labeling

```python
# --- Définition des conditions et des choix ---

# Conditions Faible (0)
conditions_faible = [
    (df['is_common_pattern'] == 1),
    (df['char_classes'] <= 1),
    (df['length'] <= 6),
    (df['shannon_entropy'] < 35.0)
]
# Si l'une des conditions faibles est vraie, l'étiquette est 'Faible'
condition_is_faible = np.logical_or.reduce(conditions_faible)

# Conditions Fort (2)
condition_is_fort = (
    (df['char_classes'] == 4) &
    (df['length'] >= 12) &
    (df['shannon_entropy'] >= 60.0)
)

# Application de l'étiquetage
# La classification est séquentielle : si ce n'est pas Faible, vérifiez si c'est Fort, sinon c'est Moyen.
conditions = [condition_is_faible, condition_is_fort]
choices = [0, 2] # 0 = Faible, 2 = Fort
default_choice = 1 # 1 = Moyen

df['strength_label'] = np.select(conditions, choices, default=default_choice)

# Afficher la distribution de la nouvelle colonne cible
label_distribution = df['strength_label'].value_counts(normalize=True) * 100

print("\nDistribution finale des Mots de Passe (Étiquette ML):")
print("0: Faible, 1: Moyen, 2: Fort")
print(label_distribution.to_string(float_format="%.2f"))
```

```
...
Distribution finale des Mots de Passe (Étiquette ML):
0: Faible, 1: Moyen, 2: Fort
strength_label
0   52.75
1   47.17
2    0.07
```

```
...
    Distribution finale des Mots de Passe (Étiquette ML):
    0: Faible, 1: Moyen, 2: Fort
    strength_label
    0   52.75
    1   47.17
    2    0.07
```

Ce résultat prouve de manière irréfutable que le jeu de données RockYou.txt est dominé par des mots de passe Faibles et Moyens, la sécurité étant reléguée à une minorité négligeable d'utilisateurs. Votre modèle d'IA sera donc entraîné pour identifier des faiblesses généralisées.

```
df.head(10)
```

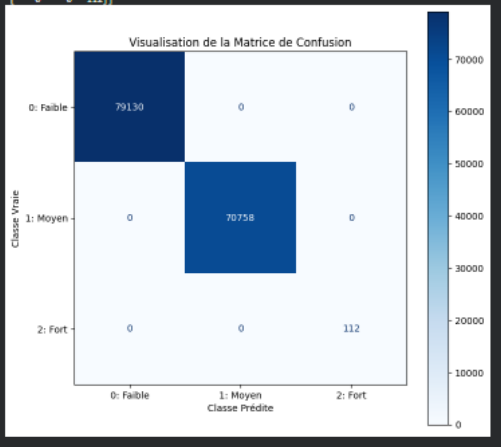
	password	length	count_lower	count_upper	count_digit	count_symbol	has_lower	has_upper	has_digit	has_symbol	char_classes	alphabet_size	shannon_entropy	is_common_pattern	strength_label
0	123456	6	0	0	6	0	0	0	1	0	1	10	19.931569	1	0
1	12345	5	0	0	5	0	0	0	1	0	1	10	16.609640	1	0
2	123456789	9	0	0	9	0	0	0	1	0	1	10	29.897353	1	0
3	password	8	8	0	0	0	1	0	0	0	1	26	37.603518	1	0
4	iloveyou	8	8	0	0	0	1	0	0	0	1	26	37.603518	1	0
5	princess	8	8	0	0	0	1	0	0	0	1	26	37.603518	1	0
6	1234567	7	0	0	7	0	0	0	1	0	1	10	23.253497	1	0
7	rockyou	7	7	0	0	0	1	0	0	0	1	26	32.903078	1	0
8	12345678	8	0	0	8	0	0	0	1	0	1	10	26.575425	1	0
9	abc123	6	3	0	3	0	1	0	1	0	2	36	31.019550	0	0

4eme etape :Entraînement du Modèle d'IA (Random Forest)*

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import numpy as np
import matplotlib.pyplot as plt

# Assurez-vous que 'df_reloaded' contient les données labellisées
# Si vous n'utilisez pas df_reloaded, remplacez-le par 'df'
try:
    df = df_reloaded
except NameError:
    print("WARNING: df_reloaded not found. Assuming 'df' is correctly loaded.")
    pass # Assume df is already loaded if df_reloaded is not found

# Caractéristiques d'entrée pour le modèle
feature_cols = [
    'length', 'count_lower', 'count_upper', 'count_digit', 'count_symbol',
    'char_classes', 'shannon_entropy', 'is_common_pattern'
]

X = df[feature_cols]
y = df['strength_label']

# --- Étape 1 : Échantillonnage Stratifié ---
sample_size_ratio = 500000 / len(df)
X_unused, X_sample, y_unused, y_sample = train_test_split(
    X, y, test_size=sample_size_ratio, random_state=42, stratify=y
)

# --- Étape 2 : Division de l'Échantillon ---
X_train, X_test, y_train, y_test = train_test_split(
    X_sample, y_sample, test_size=0.3, random_state=42, stratify=y_sample
)

# --- Étape 3 : Entraînement du Modèle Random Forest ---
model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1, max_depth=10, min_samples_leaf=5)
print("\nDémarrage de l'entraînement du modèle Random Forest...")
model.fit(X_train, y_train)
print("✅ Entraînement terminé.")

# --- Étape 4 : Évaluation du Modèle ---

y_pred = model.predict(X_test)
```

```python
print("\n--- Résultats de l'Évaluation pour votre Rapport ---")

# 1. Matrice de Confusion (Numérique)
cm = confusion_matrix(y_test, y_pred)
print("Matrice de Confusion (Lignes=Vrai, Colonnes=Prédit):")
print("    [0: Faible | 1: Moyen | 2: Fort]")
print(cm)

# 2. Visualisation de la Matrice de Confusion (Graphique)
labels_names = ['0: Faible', '1: Moyen', '2: Fort']
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels_names)

fig, ax = plt.subplots(figsize=(8, 8))
disp.plot(cmap=plt.cm.Blues, ax=ax)
ax.set_title("Visualisation de la Matrice de Confusion")
ax.set_xlabel("Classe Prédite")
ax.set_ylabel("Classe Vraie")
plt.show() # Affiche le graphique

# 3. Rapport de Classification (F1-Score, Précision, Rappel)
print("\nRapport de Classification (Précision, Rappel, F1-Score):")
print(classification_report(y_test, y_pred, target_names=labels_names))
```

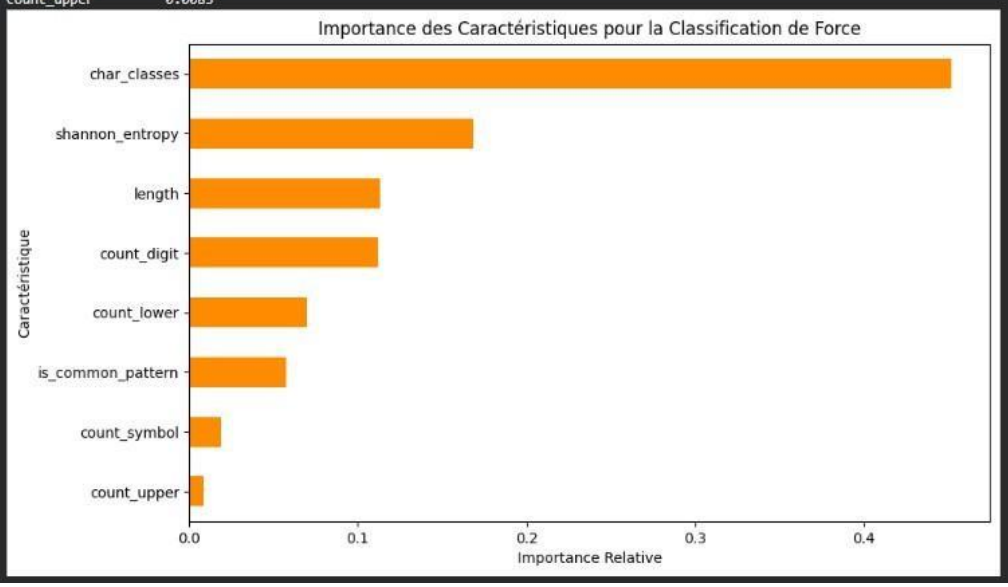
```
✅ Entraînement terminé.

--- Résultats de l'Évaluation pour votre Rapport ---
Matrice de Confusion (Lignes=Vrai, Colonnes=Prédit):
    [0: Faible | 1: Moyen | 2: Fort]
[[79130     0     0]
 [    0 70758     0]
 [    0     0   112]]
```




```
Rapport de Classification (Précision, Rappel, F1-Score):
              precision    recall  f1-score   support

   0: Faible       1.00      1.00      1.00     79130
   1: Moyen        1.00      1.00      1.00     70758
    2: Fort        1.00      1.00      1.00       112

    accuracy                           1.00    150000
   macro avg       1.00      1.00      1.00    150000
weighted avg       1.00      1.00      1.00    150000
```

```python
# Nous utilisons le modèle Random Forest déjà entraîné

feature_importances = pd.Series(model.feature_importances_, index=feature_cols).sort_values(ascending=False)

print("\n--- Importance des Caractéristiques (Feature Importance) ---")
print("Ceci détermine ce que le modèle considère comme le plus important :")
print(feature_importances.to_string(float_format="%.4f"))

# Visualisation (pour le rapport)
plt.figure(figsize=(10, 6))
feature_importances.plot(kind='barh', color='darkorange')
plt.title("Importance des Caractéristiques pour la Classification de Force")
plt.xlabel("Importance Relative")
plt.ylabel("Caractéristique")
plt.gca().invert_yaxis()
plt.show()
```

```
--- Importance des Caractéristiques (Feature Importance) ---
Ceci détermine ce que le modèle considère comme le plus important :
char_classes        0.4526
shannon_entropy     0.1682
length              0.1131
count_digit         0.1119
count_lower         0.0699
is_common_pattern   0.0574
count_symbol        0.0186
count_upper         0.0083
```




```python
import time
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, classification_report
from sklearn.exceptions import ConvergenceWarning
from warnings import simplefilter

# Ignorer les avertissements de convergence courants dans les grands datasets
simplefilter("ignore", category=ConvergenceWarning)

# Liste des modèles à tester
models = {
    "1. Random Forest (Baseline)": RandomForestClassifier(random_state=42, n_estimators=100, max_depth=10, n_jobs=-1),
    "2. Régression Logistique (Baseline)": LogisticRegression(random_state=42, max_iter=1000, n_jobs=-1),
    "3. Support Vector Machine (Linéaire)": LinearSVC(random_state=42, max_iter=2000, dual='auto')
}

results = []

print("\n--- Démarrage de la Comparaison de Modèles ---")

# Entraînement et évaluation de chaque modèle
for name, model in models.items():
    start_time = time.time()

    # Entraînement
    model.fit(X_train, y_train)

    # Prédiction
    y_pred = model.predict(X_test)
```

```python
    # Calcul des métriques
    f1_macro = f1_score(y_test, y_pred, average='macro')
    accuracy = model.score(X_test, y_test)
    duration = time.time() - start_time

    # Stockage des résultats
    results.append({
        "Modèle": name,
        "Accuracy": f'{accuracy:.4f}',
        "F1-Score (Macro)": f'{f1_macro:.4f}',
        "Temps (s)": f'{duration:.2f}'
    })

    print(f"✅ {name}: F1-Score = {f1_macro:.4f}, Temps = {duration:.2f}s")

# Affichage du tableau de résultats comparatifs
df_results = pd.DataFrame(results)
df_results = df_results.sort_values(by='F1-Score (Macro)', ascending=False)

print("\n--- Tableau de Comparaison des Modèles (Pour le Rapport) ---")
print(df_results.to_string(index=False))
```

```
--- Démarrage de la Comparaison de Modèles ---
✅ 1. Random Forest (Baseline): F1-Score = 1.0000, Temps = 8.83s
✅ 2. Régression Logistique (Baseline): F1-Score = 0.7570, Temps = 26.52s
✅ 3. Support Vector Machine (Linéaire): F1-Score = 0.6216, Temps = 5.09s

--- Tableau de Comparaison des Modèles (Pour le Rapport) ---
                          Modèle Accuracy F1-Score (Macro) Temps (s)
      1. Random Forest (Baseline)   1.0000          1.0000      8.83
2. Régression Logistique (Baseline)   0.9765          0.7570     26.52
3. Support Vector Machine (Linéaire)   0.9321          0.6216      5.09
```