Password Strength Analysis Tool

In today's digital world, **passwords are the first line of defense** for protecting personal accounts, corporate systems, and sensitive information. Yet, weak and reused passwords remain one of the **most exploited vulnerabilities** in cybersecurity. Data breaches, identity theft, and unauthorized access often stem from poor password hygiene, such as using short passwords, predictable patterns, or personal information.

Despite the availability of security tools, most users—and even some organizations—still underestimate the **importance of evaluating password strength** accurately. Basic checks like "at least 8 characters and one symbol" are not enough in the face of modern password-cracking techniques, which leverage powerful GPUs, leaked password databases, and pattern recognition algorithms. What we need is **a smarter, context-aware way** to analyze and understand how secure a password truly is.

➢ Why Do We Need This Tool?

- **To uncover weaknesses beyond basic criteria**: Many passwords technically meet "complexity" requirements but are still easy to guess (e.g., Password123!). This tool goes deeper by using **entropy-based models** to assess actual unpredictability.
- **To simulate how attackers think**: Hackers look for patterns, common words, repeated characters, and personal details like your name or birthdate. This tool replicates those tactics to **identify vulnerabilities before an attacker does**.
- **To reduce reliance on guesswork**: Users often don't know why a password is weak. This tool provides **clear, actionable feedback**, helping individuals and organizations make informed decisions.
- **To raise awareness**: By showing entropy scores and estimated crack times, the tool **educates users about the real strength of their passwords**, turning abstract concepts into tangible insights.
- **To scale security checks**: In environments where administrators manage hundreds of users, this tool allows **batch scanning of password lists**, identifying weak or reused credentials efficiently.

➢ What Makes This Tool Important?

- **It empowers users** to take control of their own security by learning how to build stronger passwords.
- **It aids IT teams** in enforcing better password policies and auditing existing ones.
- **It supports developers and system architects** by enabling password strength enforcement during onboarding, registration, or authentication flows.

- **It contributes to risk mitigation** by detecting flaws that could be exploited by brute force, dictionary attacks, or social engineering.

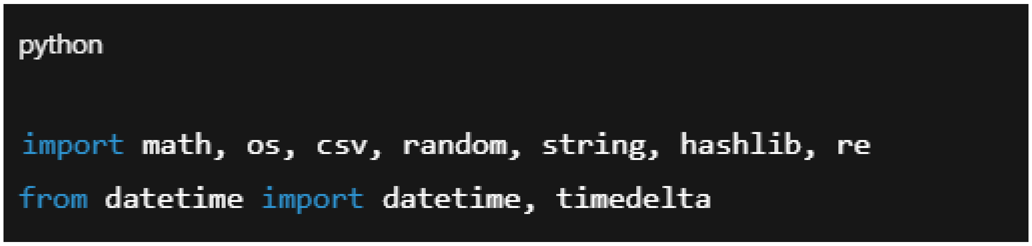
Most importantly, this tool **bridges the gap between users and security principles** by delivering insights that are technical in accuracy, yet understandable in plain language.

➢ How Does the Tool Achieve This?

This Python-based command-line utility implements a multi-layered approach:

- **Entropy Calculation**: Quantifies how unpredictable a password is, based on character variety and length. This models the actual difficulty an attacker would face when trying to crack the password using brute force.
- **Pattern Recognition**: Detects simple sequences like 1234, repeated characters like aaa, and keyboard patterns like qwerty—all of which are frequently used and easily guessed.
- **Contextual Awareness**: By optionally allowing users to input names, birthdays, or email addresses, the tool flags passwords that contain personal data—common targets for social engineering attacks.
- **Detailed Feedback**: Rather than simply labeling passwords as "weak", the tool explains **why** a password is weak, what specific improvements can be made, and offers **tailored suggestions** for stronger alternatives.
- **File Scanning & Report Generation**: It supports bulk analysis of password files, generating full reports that identify duplicates, common passwords, weak entries, and possible security risks—all without exposing sensitive information, thanks to redaction and hashing options.
- **Strong Password Generator**: When a password fails the strength test, the tool can generate a complex and secure alternative using a mix of uppercase, lowercase, numbers, and symbols—randomly selected to maximize entropy.

1. Imports and Dependencies



```
python

import math, os, csv, random, string, hashlib, re
from datetime import datetime, timedelta
```

Explanation of Imports:

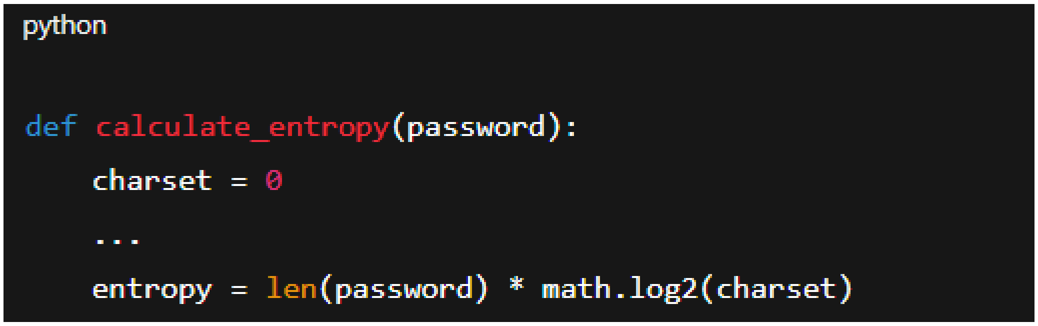
Module	Purpose
math	Used for entropy calculation via log2.
os	Handles file paths and I/O validation.
csv	(Imported for future extension; not used yet).
random & string	Create secure random passwords.
hashlib	Secure hashing of passwords (SHA-256) for duplicate detection.
re	Pattern matching (repeated chars, sequences, personal info).
datetime & timedelta	Log time and estimate password crack durations.

2. Entropy-Based Password Strength Model

➢ What is Entropy?

Entropy in cybersecurity refers to the **amount of unpredictability or randomness** in a password. The higher the entropy, the **more time and effort** an attacker needs to crack the password via brute force.

calculate_entropy(password)



```python
def calculate_entropy(password):
    charset = 0
    ...
    entropy = len(password) * math.log2(charset)
```

➢ How it works:

- Dynamically determines character diversity used in the password:
    - Lowercase (26)
    - Uppercase (26)
    - Numbers (10)
    - Special characters from a defined secure set (!@#$%^&*)
- Calculates entropy using:
  Entropy = Length × log2(Character Set Size)

➢ Why it's secure:

Entropy reflects the total number of possible combinations for a given password structure. It’s a realistic model of how difficult a password would be to guess programmatically.

---

classify_entropy(entropy, password)

- Converts the entropy into a score from **0 to 100**.
- Categorizes strength:
    - **Weak**: entropy < 40
    - **Moderate**: 40 ≤ entropy < 60
    - **Strong**: entropy ≥ 60
- Calls explain_strength() for human-readable analysis.

---

explain_strength(strength, pw)

Evaluates why a password has been classified the way it is by checking:

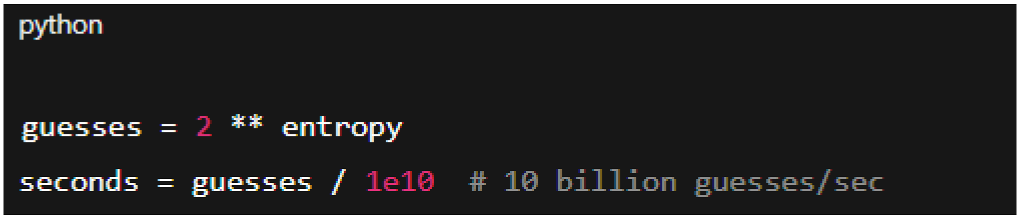
- **Length**: Short passwords are penalized.

- **Content-only types**: Only letters, only numbers = weak.
- **Missing elements**: Uppercase, special characters, etc.

The function outputs **personalized advice**, enhancing user understanding and promoting better habits.

---

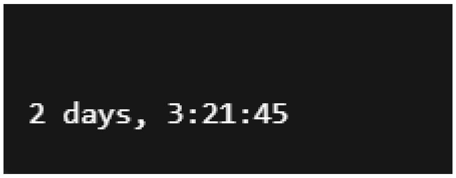
estimate_crack_time(entropy)



python

```python
guesses = 2 ** entropy
seconds = guesses / 1e10  # 10 billion guesses/sec
```

Estimates how long a password would take to crack using modern GPUs or password cracking rigs. Returns output in a timedelta format like:



```
2 days, 3:21:45
```

Useful for explaining risk to non-technical users or auditing systems.

---

3. Pattern and Behavior-Based Checks

has_patterns(password)

Detects **common human patterns**, such as:

- Sequential digits or characters (1234, abcd, qwerty)
- Repeated characters (e.g., aaaa, 111)

contains_personal_info(password, context_data)

Checks whether the password contains any user-supplied context (e.g., name, email, birthday). This detects the #1 most common mistake: **using personal data in passwords**.

---

4. Strong Password Generator

generate_strong_password(length=14)

Generates a cryptographically secure password using:

- **Uppercase and lowercase letters**
- **Numbers**
- **Allowed special characters**

Why it matters:

- Enforces a **minimum security threshold**.
- Adds randomness through Python's random.choice().

Used:

- For suggestions when a weak password is detected.
- On demand by the user.

---

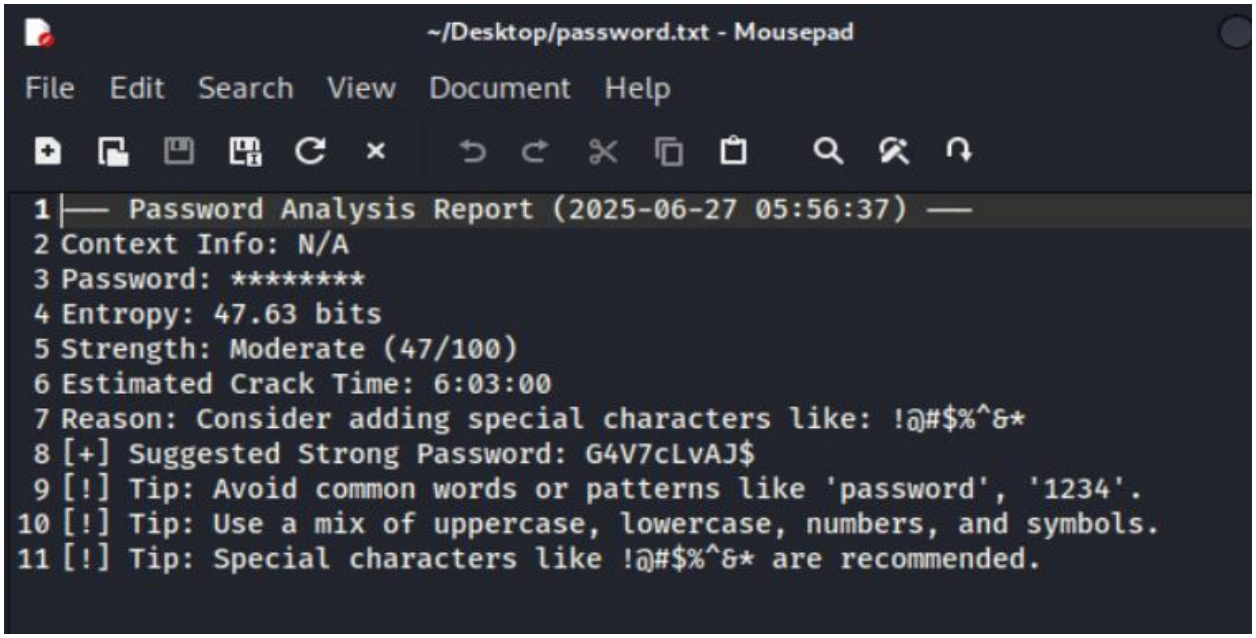
5. Individual Password Analysis & Reporting

save_password_analysis(pw, context_data, output_file="password.txt")

A full-featured function that:

- Calculates entropy and strength.
- Estimates crack time.
- Checks for patterns, common words, personal info.
- Suggests a better password if needed.
- Writes all findings into a well-structured text report.

➢ **Sample Output File:**



```
├── Password Analysis Report (2025-06-27 05:56:37) ──
Context Info: N/A
Password: ********
Entropy: 47.63 bits
Strength: Moderate (47/100)
Estimated Crack Time: 6:03:00
Reason: Consider adding special characters like: !@#$%^&*
[+] Suggested Strong Password: G4V7cLvAJ$
[!] Tip: Avoid common words or patterns like 'password', '1234'.
[!] Tip: Use a mix of uppercase, lowercase, numbers, and symbols.
[!] Tip: Special characters like !@#$%^&* are recommended.
```

Helps users **understand the consequences** of their choices and improve them.

---

6. Bulk File-Based Scanning

scan_password_file(filepath, context_data, redact=False, encrypt=False)

Reads a text file (e.g., passwords.txt) with one password per line and:

- Runs each password through the same analysis pipeline.
- Marks **duplicates** using SHA-256 hashes.
- Flags risky patterns and common passwords.
- Stores results in a report like passwords_report.txt.

➢ Options:

- **Redact** passwords (********)
- **Encrypt** with SHA-256 hashes

➢ Useful for:

- IT teams performing audits
- Checking employee password reuse
- Privacy-respecting security reviews

---

7. User Interaction: Command-Line Interface (CLI)

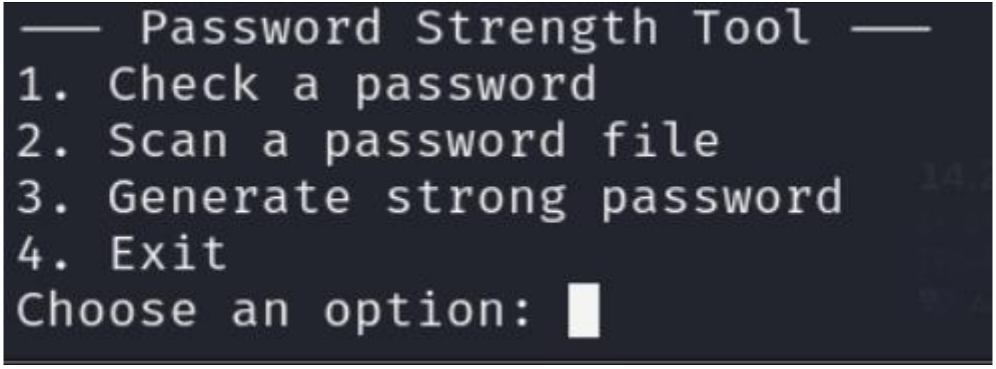
main()

Starts a menu-driven interface that lets users:

- Analyze a single password
- Scan a file of passwords
- Generate a new strong password
- Exit the program

It optionally prompts the user for **context data** (name, email, etc.), which enhances the analysis.

- Sample Menu:



```
—— Password Strength Tool ——
1. Check a password
2. Scan a password file
3. Generate strong password
4. Exit
Choose an option: █
```

---

8. Summary of Features

Feature	Description
Entropy-based analysis	Measures randomness & strength mathematically
Personalized feedback	Explains password weaknesses in plain language
Pattern & context detection	Flags predictable or risky user behaviors
File scanning	Scans large lists of passwords and reports findings
SHA-256 hashing support	Encrypts data in reports to ensure confidentiality
Human behavior modeling	Detects real-world risks like birthdays, names, etc.
Offline and portable	No internet needed, data remains local
Report generation	Provides formatted output for each password or scan
Strong password generator	Creates secure, complex passwords automatically

9. Suggested Enhancements

Idea	Why It Matters
GUI Interface	Use Tkinter or PyQt to make it user-friendly for non-tech users
Real-time analyzer	Provide feedback while typing a password
Graphical reports	Use matplotlib or plotly to visualize entropy vs. strength
Online API integration	Check against real data breaches (e.g., HaveIBeenPwned)
CSV support	Parse large CSVs with usernames and passwords
Integration ready	Make it part of CI/CD pipelines or internal tools

10.Final Thoughts

This tool is much more than a typical "strong password" checker. It is designed with a cybersecurity mindset—one that takes into account **human error**, **reused credentials**, and **cracking models** based on entropy and probability.

Unlike simple validators that only check for the presence of characters, this tool evaluates **how secure a password would be under realistic attack conditions**. By educating the user, flagging mistakes, and offering alternatives, it contributes to stronger digital hygiene.

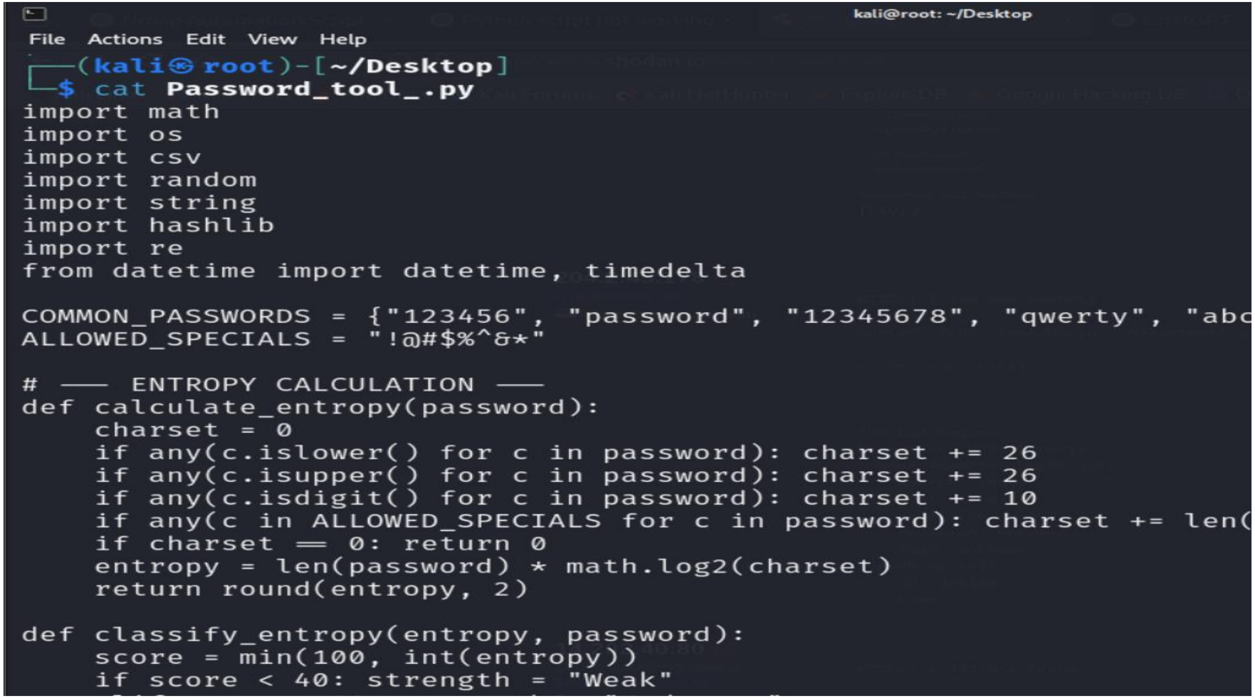
It is suitable for:

- **Cybersecurity learners** building foundational skills.
- **Penetration testers** analyzing password dumps.
- **Organizations** doing internal audits.
- **Developers** enforcing password standards in apps.

By combining **math, human psychology, and automation**, this tool makes password security tangible and actionable.

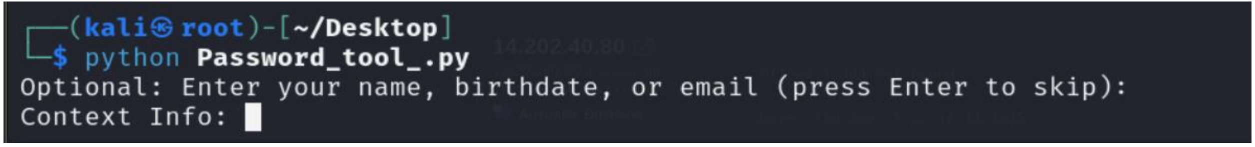
11.Results/Output

- First we will create a python file using commad “cat >Password_tool_.py” and command like “cat Password_tool_.py” can be used to view the file.



```
┌──(kali㉿root)-[~/Desktop]
└─$ cat Password_tool_.py
import math
import os
import csv
import random
import string
import hashlib
import re
from datetime import datetime, timedelta

COMMON_PASSWORDS = {"123456", "password", "12345678", "qwerty", "abc
ALLOWED_SPECIALS = "!@#$%^&*"

# —— ENTROPY CALCULATION ——
def calculate_entropy(password):
    charset = 0
    if any(c.islower() for c in password): charset += 26
    if any(c.isupper() for c in password): charset += 26
    if any(c.isdigit() for c in password): charset += 10
    if any(c in ALLOWED_SPECIALS for c in password): charset += len(
    if charset == 0: return 0
    entropy = len(password) * math.log2(charset)
    return round(entropy, 2)

def classify_entropy(entropy, password):
    score = min(100, int(entropy))
    if score < 40: strength = "Weak"
```

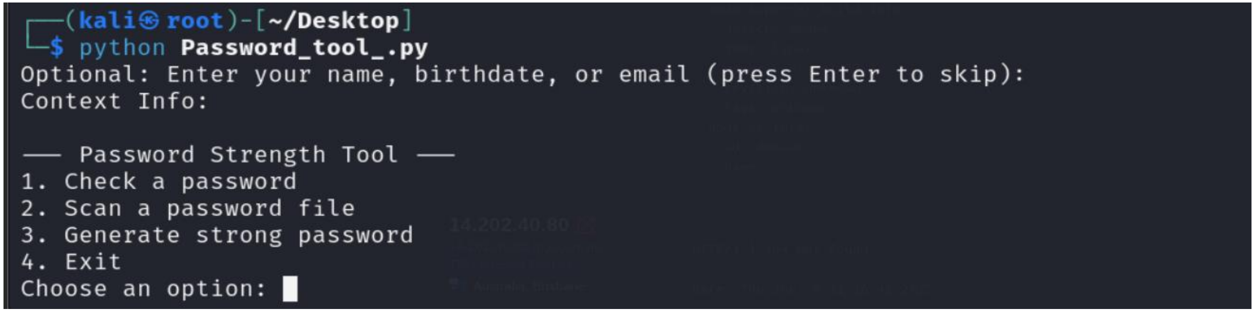
- Now for running the python script/tool , we will use the command python “Password_tool_.py”



```
┌──(kali㉿root)-[~/Desktop]
└─$ python Password_tool_.py
Optional: Enter your name, birthdate, or email (press Enter to skip):
Context Info: █
```

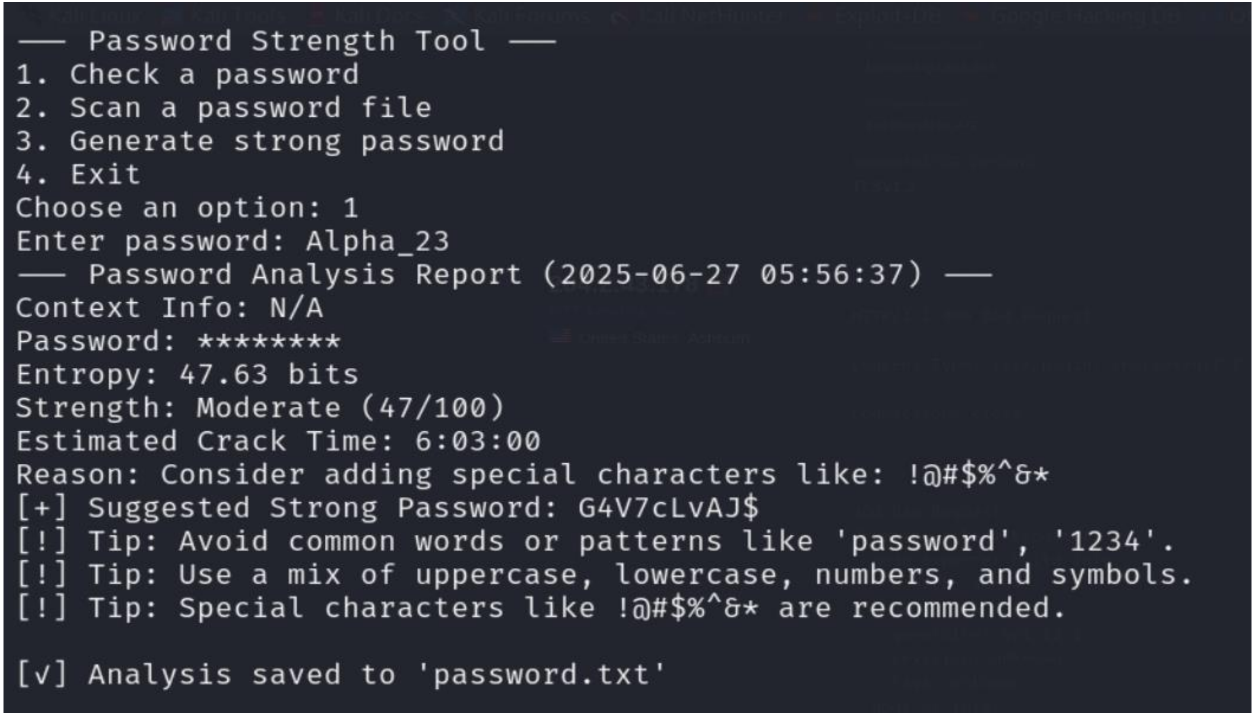
- After running the tool u will get choice of giving optional info (it’s add so that when file is cerated we will know who gave the input), it’s optional so no need for giving the data

- At bottom u will see the options like check password, scan a password file etc . Select the function u want to perform and follow the instructions.



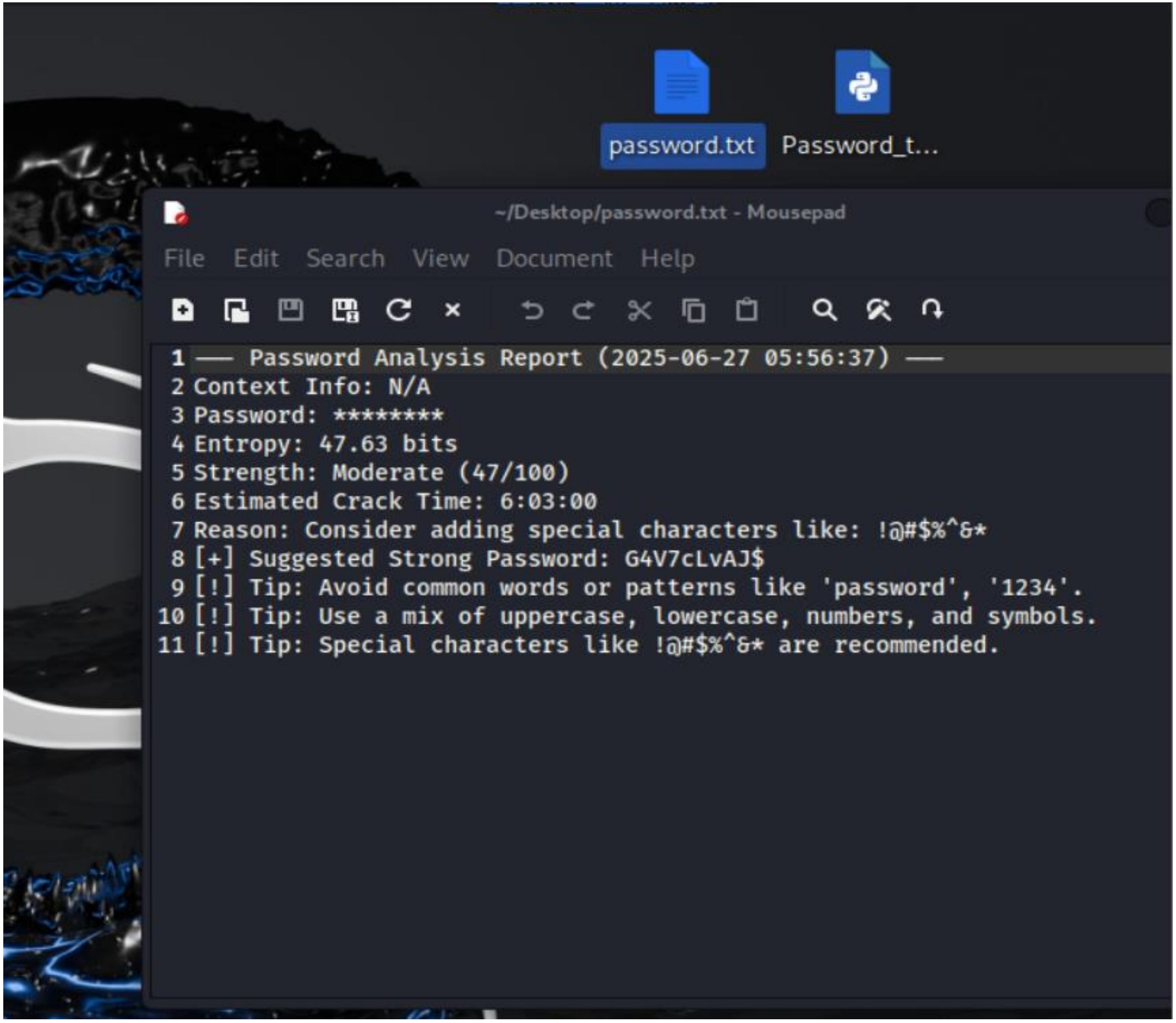
```
┌──(kali㉿root)-[~/Desktop]
└─$ python Password_tool_.py
Optional: Enter your name, birthdate, or email (press Enter to skip):
Context Info:

—— Password Strength Tool ——
1. Check a password
2. Scan a password file
3. Generate strong password
4. Exit
Choose an option: █
```

- W'll start with option 1 , when we enter the input '1' it will ask for Password(of which u want to check the strength) , then it will give the detailed output about the strength of the given password and also saves the output in file Password.txt .



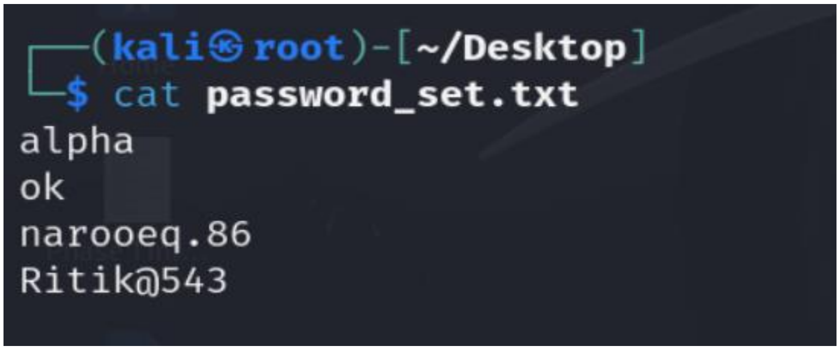
```
—— Password Strength Tool ——
1. Check a password
2. Scan a password file
3. Generate strong password
4. Exit
Choose an option: 1
Enter password: Alpha_23
—— Password Analysis Report (2025-06-27 05:56:37) ——
Context Info: N/A
Password: ********
Entropy: 47.63 bits
Strength: Moderate (47/100)
Estimated Crack Time: 6:03:00
Reason: Consider adding special characters like: !@#$%^&*
[+] Suggested Strong Password: G4V7cLvAJ$
[!] Tip: Avoid common words or patterns like 'password', '1234'.
[!] Tip: Use a mix of uppercase, lowercase, numbers, and symbols.
[!] Tip: Special characters like !@#$%^&* are recommended.

[√] Analysis saved to 'password.txt'
```

- As we can see that a txt file is created with the output.



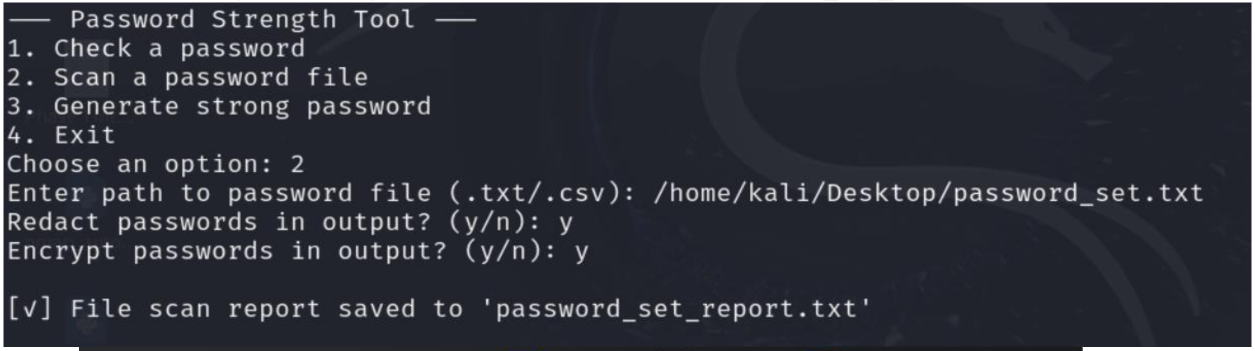
```
—— Password Analysis Report (2025-06-27 05:56:37) ——
Context Info: N/A
Password: ********
Entropy: 47.63 bits
Strength: Moderate (47/100)
Estimated Crack Time: 6:03:00
Reason: Consider adding special characters like: !@#$%^&*
[+] Suggested Strong Password: G4V7cLvAJ$
[!] Tip: Avoid common words or patterns like 'password', '1234'.
[!] Tip: Use a mix of uppercase, lowercase, numbers, and symbols.
[!] Tip: Special characters like !@#$%^&* are recommended.
```

- Now for checking the file containing password , first we have to create a file(can also use other files if u have) using command “cat>password_set.txt” and fill some random passwords.

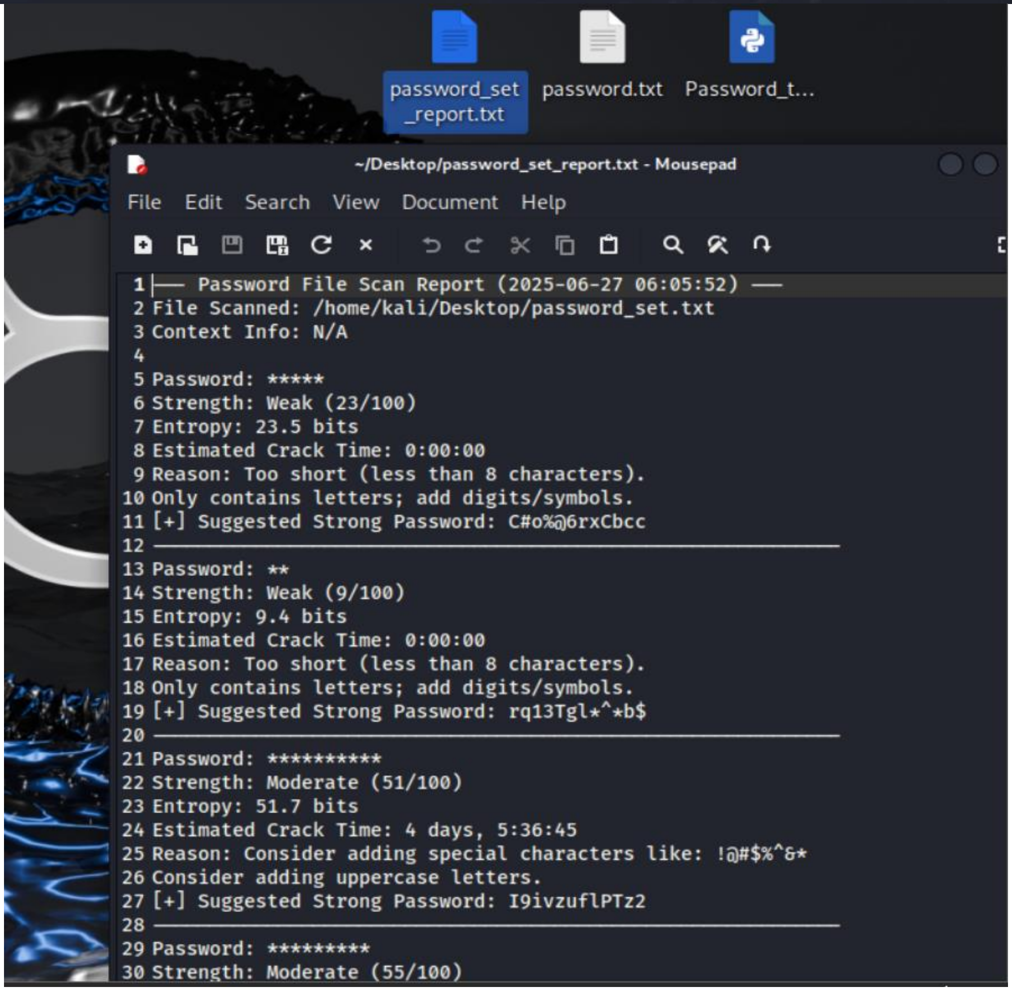


```
┌──(kali㉿root)-[~/Desktop]
└─$ cat password_set.txt
alpha
ok
narooeq.86
Ritik@543
```

- Now we will go for 2, when we enter the input ‘2’ it will ask for Password file or path of file (of which u want to check the strength) , then it will give the detailed output about the strength of the given password list in file Password_set_report.txt .

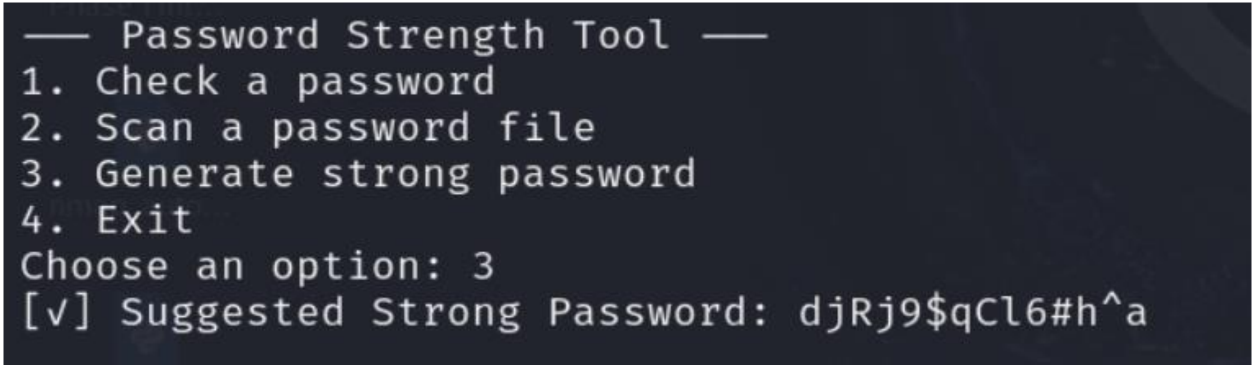


```
── Password Strength Tool ──
1. Check a password
2. Scan a password file
3. Generate strong password
4. Exit
Choose an option: 2
Enter path to password file (.txt/.csv): /home/kali/Desktop/password_set.txt
Redact passwords in output? (y/n): y
Encrypt passwords in output? (y/n): y

[√] File scan report saved to 'password_set_report.txt'
```



```
── Password File Scan Report (2025-06-27 06:05:52) ──
File Scanned: /home/kali/Desktop/password_set.txt
Context Info: N/A

Password: *****
Strength: Weak (23/100)
Entropy: 23.5 bits
Estimated Crack Time: 0:00:00
Reason: Too short (less than 8 characters).
Only contains letters; add digits/symbols.
[+] Suggested Strong Password: C#o%@6rxCbcc
────────────────────────────────────────────────────────────
Password: **
Strength: Weak (9/100)
Entropy: 9.4 bits
Estimated Crack Time: 0:00:00
Reason: Too short (less than 8 characters).
Only contains letters; add digits/symbols.
[+] Suggested Strong Password: rq13Tgl*^*b$
────────────────────────────────────────────────────────────
Password: **********
Strength: Moderate (51/100)
Entropy: 51.7 bits
Estimated Crack Time: 4 days, 5:36:45
Reason: Consider adding special characters like: !@#$%^&*
Consider adding uppercase letters.
[+] Suggested Strong Password: I9ivzuflPTz2
────────────────────────────────────────────────────────────
Password: *********
Strength: Moderate (55/100)
```

- Now for option 3 and 4 , When we choose the option ‘3’ it will suggest strong password and for final option ‘4’ , it will simply close the tool/terminate python script



```
—— Password Strength Tool ——
1. Check a password
2. Scan a password file
3. Generate strong password
4. Exit
Choose an option: 3
[√] Suggested Strong Password: djRj9$qCl6#h^a
```

12.Source Code

```
import math
import os
import csv
import random
import string
import hashlib
import re
from datetime import datetime, timedelta

COMMON_PASSWORDS = {"123456", "password", "12345678", "qwerty", "abc123", "password1"}
ALLOWED_SPECIALS = "!@#$%^&*"

# --- ENTROPY CALCULATION ---
def calculate_entropy(password):
    charset = 0
    if any(c.islower() for c in password): charset += 26
    if any(c.isupper() for c in password): charset += 26
    if any(c.isdigit() for c in password): charset += 10
    if any(c in ALLOWED_SPECIALS for c in password): charset += len(ALLOWED_SPECIALS)
    if charset == 0: return 0
    entropy = len(password) * math.log2(charset)
    return round(entropy, 2)

def classify_entropy(entropy, password):
    score = min(100, int(entropy))
    if score < 40: strength = "Weak"
    elif score < 60: strength = "Moderate"
    else: strength = "Strong"
    reason = explain_strength(strength, password)
    return strength, score, reason

def explain_strength(strength, pw):
    reasons = {
        "Weak": [],
        "Moderate": [],
        "Strong": ["Password is long enough with good character diversity."]
    }
    if len(pw) < 8:
        reasons["Weak"].append("Too short (less than 8 characters).")
    if pw.isalpha():
        reasons["Weak"].append("Only contains letters; add digits/symbols.")
    if pw.isdigit():
        reasons["Weak"].append("Only digits; add letters and symbols.")
    if not any(c in ALLOWED_SPECIALS for c in pw):
        reasons["Moderate"].append(f"Consider adding special characters like: {ALLOWED_SPECIALS}")
    if not any(c.isupper() for c in pw):
        reasons["Moderate"].append("Consider adding uppercase letters.")
    return "\n".join(reasons[strength]) if strength in reasons else "Well-balanced password."

def estimate_crack_time(entropy):
    guesses = 2 ** entropy
    guesses_per_second = 1e10
    seconds = guesses / guesses_per_second
    return str(timedelta(seconds=int(seconds)))

def has_patterns(password):
    patterns = ["1234", "abcd", "qwerty", "1111"]
    for p in patterns:
        if p in password.lower():
            return True
    if re.search(r'(.)\1{2,}', password):
        return True
    return False
```

```
def contains_personal_info(password, context_data):
    for item in context_data:
        if item.lower() in password.lower():
            return True
    return False

def generate_strong_password(length=14):
    if length < 8:
        length = 12
    chars = string.ascii_letters + string.digits + ALLOWED_SPECIALS
    return ''.join(random.choice(chars) for _ in range(length))

def show_tips():
    return (
        "[!] Tip: Avoid common words or patterns like 'password', '1234'.\n"
        "[!] Tip: Use a mix of uppercase, lowercase, numbers, and symbols.\n"
        f"[!] Tip: Special characters like {ALLOWED_SPECIALS} are recommended."
    )

# --- INDIVIDUAL PASSWORD ANALYSIS ---
def save_password_analysis(pw, context_data, output_file="password.txt"):
    entropy = calculate_entropy(pw)
    strength, score, reason = classify_entropy(entropy, pw)
    crack_time = estimate_crack_time(entropy)
    suggested = generate_strong_password(len(pw) + 2)
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    result = [
        f"--- Password Analysis Report ({now}) ---",
        f"Context Info: {' | '.join(context_data) if context_data else 'N/A'}",
        f"Password: {'*' * len(pw)}",
        f"Entropy: {entropy} bits",
        f"Strength: {strength} ({score}/100)",
        f"Estimated Crack Time: {crack_time}",
        f"Reason: {reason}"
    ]

    if pw in COMMON_PASSWORDS:
        result.append("[!] Warning: This password is commonly used and easily cracked!")
    if has_patterns(pw):
        result.append("[!] Warning: Predictable pattern detected.")
    if contains_personal_info(pw, context_data):
        result.append("[!] Warning: Personal/context info found in password.")
    if strength != "Strong":
        result.append(f"[+] Suggested Strong Password: {suggested}")

    result.append(show_tips())

    with open(output_file, "w", encoding="utf-8") as f:
        f.write("\n".join(result))

    print("\n".join(result))
    print(f"\n[✓] Analysis saved to '{output_file}'")

# --- PASSWORD FILE SCAN ---
def scan_password_file(filepath, context_data, redact=False, encrypt=False):
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            lines = f.readlines()

        seen = set()
        report = []
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        filename = os.path.basename(filepath).split('.')[0]
        report_file = f"{filename}_report.txt"
        header = [
            f"--- Password File Scan Report ({now}) ---",
            f"File Scanned: {filepath}",
            f"Context Info: {' | '.join(context_data) if context_data else 'N/A'}",
            ""
        ]

        for line in lines:
            pw = line.strip()
            if not pw:
                continue

            hash_pw = hashlib.sha256(pw.encode()).hexdigest()
            duplicate = hash_pw in seen
            seen.add(hash_pw)

            entropy = calculate_entropy(pw)
            strength, score, reason = classify_entropy(entropy, pw)
            crack_time = estimate_crack_time(entropy)

            pw_display = "*" * len(pw) if redact else (hash_pw if encrypt else pw)

            report.append(f"Password: {pw_display}")
            report.append(f"Strength: {strength} ({score}/100)")
            report.append(f"Entropy: {entropy} bits")
            report.append(f"Estimated Crack Time: {crack_time}")
            report.append(f"Reason: {reason}")
            if pw in COMMON_PASSWORDS:
                report.append("[!] Common password warning!")
            if has_patterns(pw):
                report.append("[!] Pattern detected!")
            if contains_personal_info(pw, context_data):
                report.append("[!] Personal info found!")
            if duplicate:
                report.append("[!] Duplicate password detected!")
            if strength != "Strong":
                report.append(f"[+] Suggested Strong Password: {generate_strong_password(len(pw) + 2)}")
            report.append("-" * 60)

        with open(report_file, "w", encoding='utf-8') as rf:
            rf.write("\n".join(header + report))

        print(f"\n[✓] File scan report saved to '{report_file}'")

    except Exception as e:
        print(f"Error: {e}")

# --- MAIN MENU ---
def main():
    context_data = []
    print("Optional: Enter your name, birthdate, or email (press Enter to skip):")
    user_input = input("Context Info: ").strip()
    if user_input:
        context_data = re.split(r'\W+', user_input)

    while True:
        print("\n--- Password Strength Tool ---")
        print("1. Check a password")
        print("2. Scan a password file")
        print("3. Generate strong password")
        print("4. Exit")
        choice = input("Choose an option: ").strip()

        if choice == '1':
            pw = input("Enter password: ").strip()
            save_password_analysis(pw, context_data)
        elif choice == '2':
            path = input("Enter path to password file (.txt/.csv): ").strip()
            redact = input("Redact passwords in output? (y/n): ").lower().startswith('y')
            encrypt = input("Encrypt passwords in output? (y/n): ").lower().startswith('y')
            scan_password_file(path, context_data, redact=redact, encrypt=encrypt)
        elif choice == '3':
            print(f"[✓] Suggested Strong Password: {generate_strong_password()}")
        elif choice == '4':
            break
        else:
            print("Invalid option.")

if __name__ == '__main__':
    main()
```