Password Strength Analyzer with Entropy Calculation

Step 1: Set Up Python Environment

1. **Ensure Python3 and Pip are Installed**
   - Kali Linux typically comes with Python3 pre-installed, but let's verify it:

```
python3 --version
pip3 --version
```

   - If `pip3` is not installed, install it using:

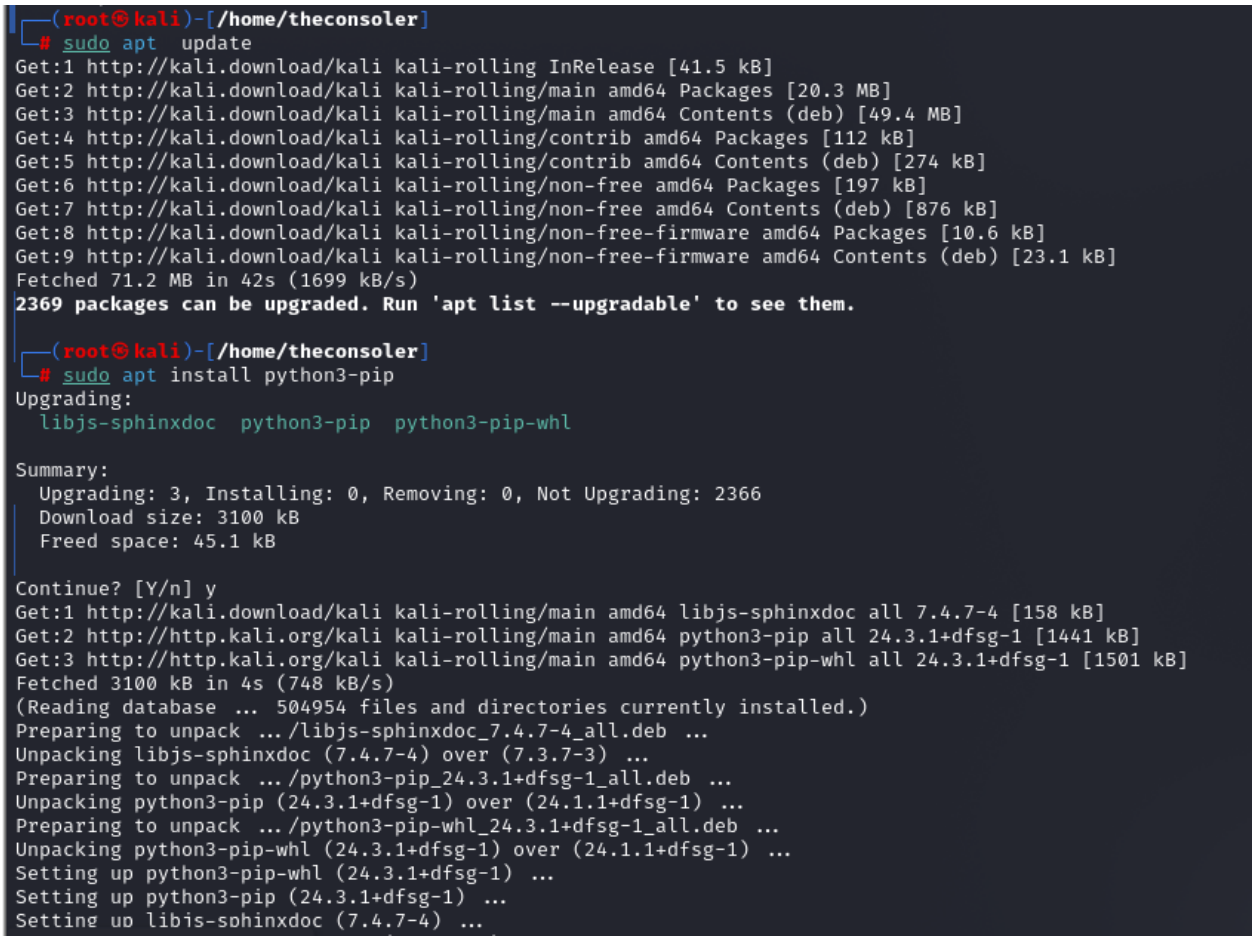
```
sudo apt update
```

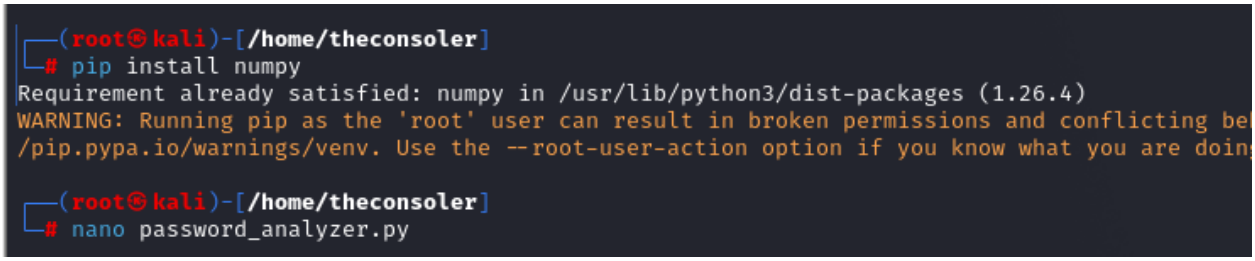
```
sudo apt install python3-pip
```

2. **Install Required Libraries**

   - For this project, we'll use `math` and `re` (regular expressions), which are built-in Python libraries, so you don't need to install them separately.
   - However, if you plan to expand your project with advanced statistical functions, install `numpy` as well:

```
pip3 install numpy
```





NUMPY IS ALREADY INSTALLED.

Step 2: Create the Python Script

1. **Create a New Python File**
   - Use a text editor like `nano` to create a file called `password_analyzer.py` in your project folder:

     ```
     nano password_analyzer.py
     ```

2. **Structure Your Script**

   In the file, we'll add different functions to analyze the password's strength by calculating entropy, detecting patterns, checking against common words, and generating recommendations.

Step 3: Write the Entropy Calculation Function

1. **Add Imports at the Top of Your Script**

   ```
   import math
   import re
   ```

2. **Define the Entropy Calculation Function**
   - Entropy is a measure of randomness and helps estimate the strength of a password. We'll calculate entropy based on the character set and password length.

```
def calculate_entropy(password):
    charset_size = 0

    # Determine charset size based on characters used in t
he password
    if re.search(r'[a-z]', password):
        charset_size += 26  # Lowercase letters
    if re.search(r'[A-Z]', password):
        charset_size += 26  # Uppercase letters
    if re.search(r'[0-9]', password):
        charset_size += 10  # Digits
    if re.search(r'[^a-zA-Z0-9]', password):
        charset_size += 32  # Special characters

    # Entropy calculation
    entropy = math.log2(charset_size ** len(password)) if
charset_size else 0
    return entropy
```

3. **Save and Test the Entropy Function**
   - Save the file with `Ctrl + O`, press `Enter`, then exit with `Ctrl + X`.
   - Test it by running Python interactively:

     ```
     python3
     ```

   - Import and test the function:

     ```
     from password_analyzer import calculate_entropy
     print(calculate_entropy("TestPassword123!"))  # Replace
     ```

```
with any sample password
```

- You should see an entropy value.

Step 4: Write Pattern Detection Function

In the `password_analyzer.py` file, add a function to check for common patterns like repeated or sequential characters.

```
def check_patterns(password):
    patterns = []

    if re.search(r'(.)\1\1', password):  # Check for three co
nsecutive identical characters
        patterns.append("Repeated characters")

    if re.search(r'(012|123|234|345|456|567|678|789)', passwo
rd):  # Simple sequence
        patterns.append("Sequential numbers")

    if re.search(r'(abc|bcd|cde|def)', password, re.IGNORECAS
E):  # Simple alpha sequence
        patterns.append("Sequential letters")

    return patterns
```

Step 5: Create Dictionary Check Function

We'll add a function that flags dictionary words or commonly used passwords. You can create a set of common passwords in the script or use a wordlist file.

```
def dictionary_check(password, common_passwords):
```

```
    lower_password = password.lower()
    if lower_password in common_passwords:
        return "Password is too common"
    return None
```

Step 6: Add Recommendations Function

This function will provide feedback based on entropy, patterns, and dictionary checks.

```
def generate_recommendations(entropy, patterns, dictionary_fl
ag):
    recommendations = []

    if entropy < 40:
        recommendations.append("Increase the password length
or add more character types.")

    if patterns:
        recommendations.append("Avoid common patterns: " + ",
".join(patterns))

    if dictionary_flag:
        recommendations.append("Avoid common or dictionary wo
rds.")

    if not recommendations:
        recommendations.append("Your password is strong!")

    return recommendations
```

Step 7: Combine All Functions in a Main Analysis Function

Now we'll write the main function that ties everything together and outputs the results.

```
def analyze_password(password):
    common_passwords = {"password", "123456", "qwerty", "admi
n", "letmein"}  # Add more as needed

    # Entropy calculation
    entropy = calculate_entropy(password)
    print(f"Entropy: {entropy:.2f} bits")

    # Pattern checks
    patterns = check_patterns(password)
    print(f"Detected Patterns: {patterns if patterns else 'No
ne'}")

    # Dictionary check
    dictionary_flag = dictionary_check(password, common_passw
ords)
    if dictionary_flag:
        print(dictionary_flag)

    # Recommendations
    recommendations = generate_recommendations(entropy, patte
rns, dictionary_flag)
    print("Recommendations:")
    for rec in recommendations:
        print(f" - {rec}")
```

Step 8: Finalize and Test the Password Analyzer

1. **Add the Main Execution Code**

At the bottom of `password_analyzer.py`, add code to take input from the user and call the `analyze_password` function.

```
if __name__ == "__main__":
    password = input("Enter a password to analyze: ")
    analyze_password(password)
```

2. **Run the Script**
   - Save and exit `password_analyzer.py` (using `Ctrl + O`, `Enter`, and `Ctrl + X`).
   - Run the script from the terminal:

     ```
     python3 password_analyzer.py
     ```

   - Enter various passwords to see the output. The script should display entropy, detected patterns, and recommendations.

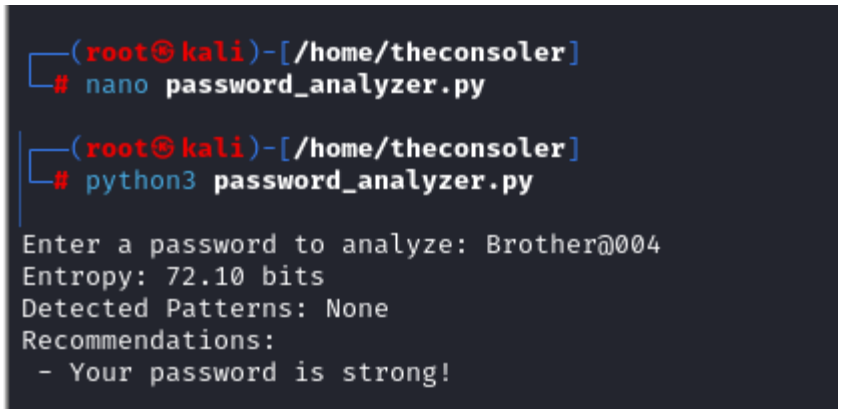
Step 9: Expand and Improve (Optional)

Here are a few ideas for enhancing the project:

- **Use External Dictionary**: Use a wordlist like `rockyou.txt` located at `/usr/share/wordlists/rockyou.txt` for a more comprehensive dictionary check.

  ```
  with open("/usr/share/wordlists/rockyou.txt") as f:
      common_passwords = set(word.strip().lower() for word i
  n f)
  ```

- **Estimate Time to Crack**: Based on entropy, estimate the time to crack the password using various attack methods.
- **Build a GUI**: If you're interested, use Tkinter to create a graphical interface.



PASSWORD - Brother@004

RESULT - Your Password is strong !

BUILDING GUI FOR PASSWORD ANALYZER WITH TKINTER

Building a GUI with Tkinter for your Password Strength Analyzer is a great idea! This will make the tool more user-friendly and accessible. I'll guide you step-by-step, following a similar process to what we did before. Tkinter comes pre-installed with Python, so you won't need additional installations for this part.

Step 1: Install Tkinter (if needed)

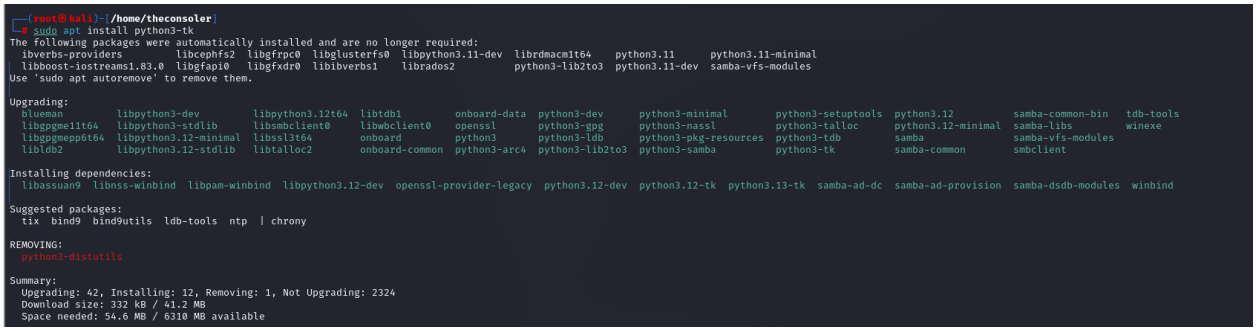
On Kali Linux, Tkinter may already be installed. If it's not, install it using the following command:

```
sudo apt update
sudo apt install python3-tk
```

Verify that it's installed by running:

```
python3 -m tkinter
```

If the Tkinter GUI window opens, you're good to go.



Step 2: Update the `password_analyzer.py` Script for GUI Integration

Let's modify your existing script to add Tkinter functionality. This will involve creating a GUI layout with input fields, buttons, and areas to display results.

1. **Open Your Python Script**

   Open the `password_analyzer.py` file in a text editor:

   ```
   nano password_analyzer.py
   ```

2. **Import Tkinter and Update Imports**

   At the top of your script, add the `tkinter` import and rename any previous imports if necessary:

   ```
   import math
   import re
   import tkinter as tk
   ```

```
from tkinter import messagebox
```

Step 3: Design the GUI Layout

Here, we'll define a basic layout that includes:

- An entry field for the password
- A button to analyze the password
- A text area to display the entropy, patterns, and recommendations

Add the following code after the import statements:

```
def create_gui():
    # Create the main window
    window = tk.Tk()
    window.title("Password Strength Analyzer")
    window.geometry("400x400")  # Width x Height

    # Label for Password Entry
    password_label = tk.Label(window, text="Enter Password:")
    password_label.pack(pady=10)

    # Password Entry Field
    password_entry = tk.Entry(window, show="*", width=30)
    password_entry.pack(pady=5)

    # Result Area
    result_text = tk.Text(window, height=15, width=45, wrap
="word", state="disabled")
    result_text.pack(pady=10)

    # Analyze Button
    analyze_button = tk.Button(window, text="Analyze Passwor
d", command=lambda: analyze_password_gui(password_entry, resu
```

```
lt_text))
    analyze_button.pack(pady=10)

    # Run the GUI main loop
    window.mainloop()
```

Step 4: Modify `analyze_password` Function for GUI Output

Now, let's adapt the `analyze_password` function to output results to the `result_text` widget in the GUI.

Add this new function below `create_gui`:

```
def analyze_password_gui(password_entry, result_text):
    # Clear previous results
    result_text.config(state="normal")
    result_text.delete(1.0, tk.END)

    # Get password from entry field
    password = password_entry.get()

    # Sample set of common passwords (expand as needed)
    common_passwords = {"password", "123456", "qwerty", "admi
n", "letmein"}

    # Calculate Entropy
    entropy = calculate_entropy(password)
    result_text.insert(tk.END, f"Entropy: {entropy:.2f} bits
\n\n")

    # Pattern Check
    patterns = check_patterns(password)
    result_text.insert(tk.END, f"Detected Patterns: {patterns
if patterns else 'None'}\n\n")
```

```
    # Dictionary Check
    dictionary_flag = dictionary_check(password, common_passw
ords)
    if dictionary_flag:
        result_text.insert(tk.END, f"{dictionary_flag}\n\n")

    # Recommendations
    recommendations = generate_recommendations(entropy, patte
rns, dictionary_flag)
    result_text.insert(tk.END, "Recommendations:\n")
    for rec in recommendations:
        result_text.insert(tk.END, f" - {rec}\n")

    # Disable editing in result text box
    result_text.config(state="disabled")
```

Step 5: Connect GUI Functions to Main Functionality

At the bottom of your script, replace the command-line interface section with a call to `create_gui()`. This will open the GUI when the script is run.

```
if __name__ == "__main__":
    create_gui()
```

```
  GNU nano 8.1
import math
import re
import tkinter as tk
from tkinter import messagebox


def calculate_entropy(password):
    charset_size = 0

    # Determine charset size based on characters used in the password
    if re.search(r'[a-z]', password):
        charset_size += 26  # Lowercase letters
    if re.search(r'[A-Z]', password):
        charset_size += 26  # Uppercase letters
    if re.search(r'[0-9]', password):
        charset_size += 10  # Digits
    if re.search(r'[^a-zA-Z0-9]', password):
        charset_size += 32  # Special characters

    # Entropy calculation
    entropy = math.log2(charset_size ** len(password)) if charset_size else 0
    return entropy

def check_patterns(password):
    patterns = []
    if re.search(r'(.)\1\1', password):  # Check for three consecutive identical characters
        patterns.append("Repeated characters")
    if re.search(r'(012|123|234|345|456|567|678|789)', password):  # Simple sequence
        patterns.append("Sequential numbers")
    if re.search(r'(abc|bcd|cde|def)', password, re.IGNORECASE):  # Simple alpha sequence
        patterns.append("Sequential letters")
    return patterns

def dictionary_check(password, common_passwords):
    lower_password = password.lower()
    if lower_password in common_passwords:
        return "Password is too common"
    return None

def generate_recommendations(entropy, patterns, dictionary_flag):
    recommendations = []
    if entropy < 40:
        recommendations.append("Increase the password length or add more character types.")
    if patterns:
        recommendations.append("Avoid common patterns: " + ", ".join(patterns))
    if dictionary_flag:
        recommendations.append("Avoid common or dictionary words.")
    if not recommendations:
        recommendations.append("Your password is strong!")
    return recommendations


def analyze_password(password):
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"}  # Add more as needed
    # Entropy calculation
    entropy = calculate_entropy(password)
    print(f"Entropy: {entropy:.2f} bits")

    # Pattern checks
    patterns = check_patterns(password)
    print(f"Detected Patterns: {patterns if patterns else 'None'}")
```

```
    # Dictionary check
    dictionary_flag = dictionary_check(password, common_passwords)
    if dictionary_flag:
        print(dictionary_flag)
    # Recommendations
    recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
    print("Recommendations:")
    for rec in recommendations:
        print(f" - {rec}")

if __name__ == "__main__":
    password = input("Enter a password to analyze: ")
    analyze_password(password)

def create_gui():
    # Create the main window
    window = tk.Tk()
    window.title("Password Strength Analyzer")
    window.geometry("400x400")  # Width x Height
    # Label for Password Entry
    password_label = tk.Label(window, text="Enter Password:")
    password_label.pack(pady=10)
    # Password Entry Field
    password_entry = tk.Entry(window, show="*", width=30)
    password_entry.pack(pady=5)
    # Result Area
    result_text = tk.Text(window, height=15, width=45, wrap="word", state="disabled")
    result_text.pack(pady=10)
    # Analyze Button
    analyze_button = tk.Button(window, text="Analyze Password", command=lambda: analyze_password_gui(password_entry, result_text))
    analyze_button.pack(pady=10)
    # Run the GUI main loop
    window.mainloop()

def analyze_password_gui(password_entry, result_text):
    # Clear previous results
    result_text.config(state="normal")
    result_text.delete(1.0, tk.END)
    # Get password from entry field
    password = password_entry.get()
    # Sample set of common passwords (expand as needed)
    common_passwords = {"password", "123456", "qwerty", "admin", "letmein"}
    # Calculate Entropy
    entropy = calculate_entropy(password)
    result_text.insert(tk.END, f"Entropy: {entropy:.2f} bits\n\n")
    # Pattern Check
    patterns = check_patterns(password)
    result_text.insert(tk.END, f"Detected Patterns: {patterns if patterns else 'None'}\n\n")
    # Dictionary Check
    dictionary_flag = dictionary_check(password, common_passwords)
    if dictionary_flag:
        result_text.insert(tk.END, f"{dictionary_flag}\n\n")
    # Recommendations
    recommendations = generate_recommendations(entropy, patterns, dictionary_flag)
    result_text.insert(tk.END, "Recommendations:\n")
    for rec in recommendations:
        result_text.insert(tk.END, f" - {rec}\n")
    # Disable editing in result text box
    result_text.config(state="disabled")
```

Step 6: Run and Test the GUI

1. Save and exit the file in the terminal (with `Ctrl + O`, `Enter`, then `Ctrl + X`).
2. Run your script:

```
python3 password_analyzer.py
```

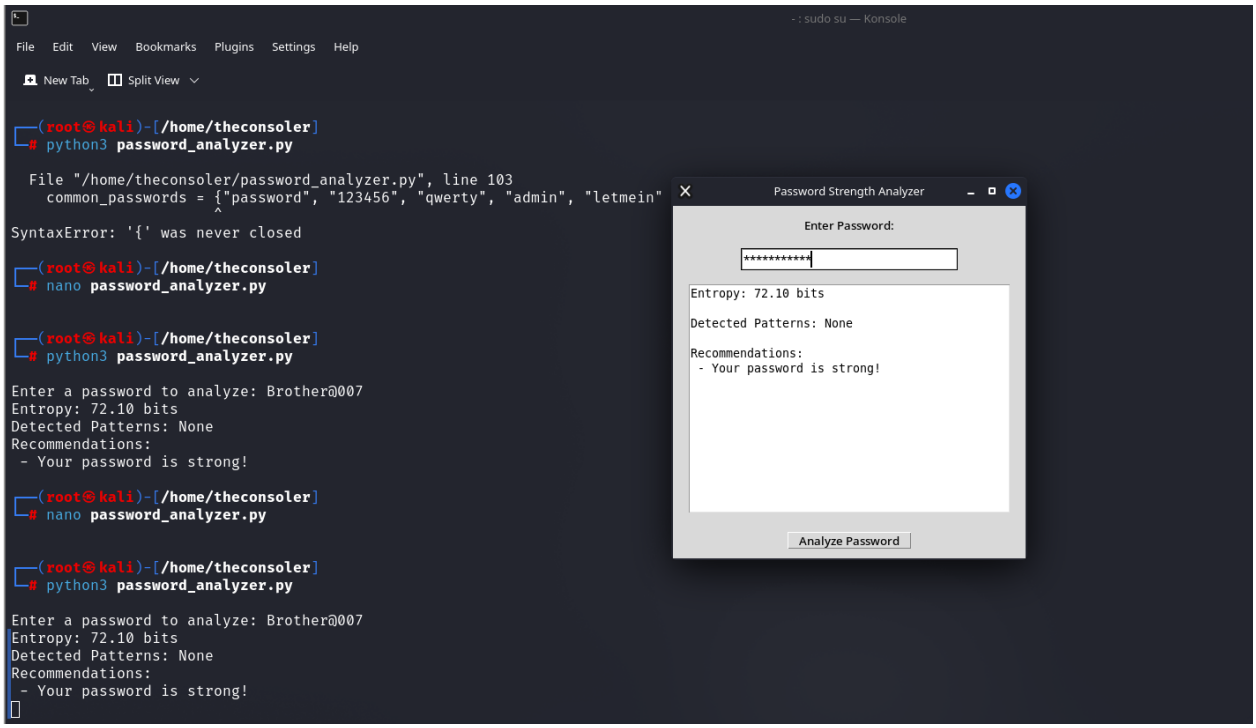
You should see a GUI window open with the following elements:

- A field where you can enter a password.
- A button to analyze the password.
- A text area displaying the entropy, patterns, and recommendations.

Step 7: Test Your GUI Application

Try entering different passwords to see how the results update in the text area. The GUI should display:

- **Entropy**: The calculated entropy value in bits.
- **Detected Patterns**: Any patterns like repeated or sequential characters.
- **Recommendations**: Tips on how to make the password stronger.



This completes your password analyzer with a GUI.