Case Study

Random Password Auditor

Ashutosh Pawar

Sam Altman

Roll Number: 150096725130

BTech CSE 25-29

Semester I

Sprint II

Subject: Python

Table of contents

1. Problem Statement

Security audit tool generating random passwords, testing strength, and conducting batch security assessments.

2. Background / Scenario

Modern systems rely heavily on strong passwords to protect user accounts and sensitive data, yet manually auditing large numbers of passwords is inefficient and error-prone. This project addresses that challenge by simulating a real-world security audit scenario in which an automated tool generates passwords in bulk, evaluates their strength against defined security policies, and produces structured reports and visual insights. The Random Password Auditor demonstrates how automation, data analysis, and systematic reporting can be used to efficiently assess password security and support the enforcement of stronger authentication practices.

3. Objective / Learning Goals

The primary objective of this project is to design and implement a Random Password Auditor that automates the generation, evaluation, and reporting of password security assessments.

The core goals are:

- To generate random passwords in batches with configurable parameters such as length and character composition.
- To evaluate password strength against defined security criteria and assign quantitative security scores.
- To identify weak passwords, document vulnerabilities, and automatically generate remediation recommendations.
- To produce comprehensive audit reports, logs, and visualizations using structured data analysis techniques.

Through this project, the key learning goal was to gain practical experience in Python programming with an emphasis on object-oriented design, functional programming concepts (decorators and lambda expressions), data analysis using Pandas, visualization using Matplotlib, and the implementation of automated security audit workflows aligned with real-world cybersecurity practices.

4. Tools Used

This project was developed on a Unix-based operating system (macOS) and designed for a Linux-compatible environment. The implementation primarily used **Python** for core logic and automation, along with supporting tools for development, data analysis, and visualization.

- **Programming Language:** Python 3 was used to implement password generation, auditing logic, reporting, and automation features.
- **Development Environment: Neovim (nvim)** was used as the primary IDE due to its efficiency, extensibility, and strong support for Python development.
- **Libraries and Frameworks:**
    - **Pandas** – for organizing audit data, performing analysis, and managing tabular results.
    - **Matplotlib** – for generating visualizations such as password strength distributions and audit metrics.
    - **time** – for measuring audit execution duration.
- **Programming Concepts and Utilities:**
    - **Object-Oriented Programming (OOP)** – for structuring the system into modular classes such as `PasswordAuditor`, `AuditReport`, `SecurityAnalyzer`, and `AuditLogger`.
    - **Decorators** – to automate audit start and end logging without modifying core logic.
    - **Lambda Expressions** – for concise scoring calculations within the auditing process.
- **File Handling and Output Formats:**
    - **CSV files** – to store structured audit results for further analysis.
    - **Text files (.txt)** – to generate human-readable audit reports and certifications.
    - **Image files** – for exporting visualization charts generated during the audit.

These tools collectively enabled the development of a modular, automated, and scalable password auditing system aligned with real-world cybersecurity assessment practices.

5. Implementation (Code)

[This section contains the full, clean source code for the three python scripts: `password_class.py, audit_report.py, auditor_main.py`]

1] **`Password_class.py`**

```
import random
import string
import re

class SecurityAnalyzer:
    """Evaluates password strength and identifies vulnerabilities."""

    @staticmethod
    def calculate_score(password):
        """Calculates a security score (0-100) using lambda logic."""
        score = 0

        # Criteria checks using lambdas
        length_score = (lambda p: min(len(p) * 4, 40))(password)  # Up to 40 pts for 
length
        upper_score = (lambda p: 10 if any(c.isupper() for c in p) else 0)(password)
        lower_score = (lambda p: 10 if any(c.islower() for c in p) else 0)(password)
        digit_score = (lambda p: 20 if any(c.isdigit() for c in p) else 0)(password)
        special_score = (lambda p: 20 if any(c in string.punctuation for c in p) else 
0)(password)

        score = length_score + upper_score + lower_score + digit_score + special_score
        return min(score, 100)

    @staticmethod
    def identify_vulnerabilities(password):
        """Identifies specific weaknesses in the password."""
        issues = []
        if len(password) < 8:
            issues.append("Too short (< 8 chars)")
        if not any(c.isdigit() for c in password):
            issues.append("Missing numbers")
        if not any(c.isupper() for c in password):
            issues.append("Missing uppercase")
        if not any(c in string.punctuation for c in password):
            issues.append("Missing special characters")

        # Simple pattern checks
        common_patterns = ["123", "abc", "password", "admin", "test"]
        if any(pat in password.lower() for pat in common_patterns):
            issues.append("Predictable pattern found")

        return issues

    @staticmethod
    def evaluate_strength(score):
        """Categorizes score into Weak, Medium, Strong."""
        if score < 50:
            return "Weak"
        elif score < 80:
            return "Medium"
        else:
            return "Strong"


class PasswordAuditor:
    """Generates and tests passwords."""

    def __init__(self):
        self.passwords = []
        self.audit_results = []

    def generate_random_passwords(self, count=50):
        """Generates a batch of random passwords with varying complexity."""
        self.passwords = []
        chars = string.ascii_letters + string.digits + string.punctuation

        for _ in range(count):
            # Randomize length between 6 and 16 to ensure a mix of weak/strong
            length = random.randint(6, 16)

            # Occasionally create intentionally weak passwords for demonstration
            if random.random() < 0.2:
                p = "".join(random.choices(string.ascii_lowercase + string.digits,
k=random.randint(4, 8)))
            else:
                p = "".join(random.choices(chars, k=length))

            self.passwords.append(p)
        return self.passwords

    def run_audit(self):
        """Audits the generated passwords using SecurityAnalyzer."""
        self.audit_results = []
        for pwd in self.passwords:
            score = SecurityAnalyzer.calculate_score(pwd)
            strength = SecurityAnalyzer.evaluate_strength(score)
            vulns = SecurityAnalyzer.identify_vulnerabilities(pwd)

            self.audit_results.append({
                "Password": pwd,
                "Score": score,
                "Strength": strength,
                "Vulnerabilities": vulns
            })
        return self.audit_results
```

2] **audit_report.sh**

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

class AuditLogger:
    """Handles logging of audit events."""

    @staticmethod
    def log(message):
        """Simple logger that prints to console with timestamp."""
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"[{timestamp}] {message}")


class AuditReport:
    """Generates reports and visualizations."""

    def __init__(self, audit_data):
        self.audit_data = audit_data
        self.df = pd.DataFrame(audit_data)

    def save_to_csv(self, filename="audit_results.csv"):
        """Saves audit results to CSV."""
        try:
            self.df.to_csv(filename, index=False)
            AuditLogger.log(f"Results saved to {filename}")
        except Exception as e:
            AuditLogger.log(f"Error saving CSV: {e}")

    def generate_txt_report(self, filename="audit_report.txt"):
        """Generates a comprehensive text report."""
        total = len(self.df)
        strength_counts = self.df['Strength'].value_counts()
        avg_score = self.df['Score'].mean()

        strong_count = strength_counts.get("Strong", 0)
        medium_count = strength_counts.get("Medium", 0)
        weak_count = strength_counts.get("Weak", 0)

        weak_passwords = self.df[self.df['Strength'] == 'Weak'].head(5)

        report = f"""RANDOM PASSWORD AUDITOR
AUDIT SUMMARY:
Audit Date: {datetime.date.today()}
Total Passwords Tested: {total}

STRENGTH DISTRIBUTION:
```

```
Strong: {strong_count} ({strong_count/total*100:.1f}%)
Medium: {medium_count} ({medium_count/total*100:.1f}%)
Weak: {weak_count} ({weak_count/total*100:.1f}%)

WEAK PASSWORDS IDENTIFIED (Top 5):
"""
        for i, row in weak_passwords.iterrows():
            number = i + 1
            password = row["Password"]
            vulnerabilities = ", ".join(row["Vulnerabilities"])

            report += f'{number}. "{password}" - {vulnerabilities}\n'

        report += f"""
SECURITY SCORE: {int(avg_score)}/100

RECOMMENDATIONS:
1. All weak passwords must be changed.
2. Enforce minimum 12 character length.
3. Require special character usage.
4. Implement password expiry policy.

AUDIT CERTIFICATION:
Status: {'PASS' if weak_count == 0 else 'CONDITIONAL PASS' if weak_count < 5 else
'FAIL'}
Weak Passwords: {weak_count} (to be remediated)
"""
        try:
            with open(filename, 'w') as f:
                f.write(report)
            AuditLogger.log(f"Report saved to {filename}")
        except Exception as e:
            AuditLogger.log(f"Error saving TXT report: {e}")

    def create_visualizations(self):
        """Creates strength distribution chart."""
        if self.df.empty:
            return

        try:
            plt.figure(figsize=(8, 6))
            strength_counts = self.df['Strength'].value_counts()
            colors = {'Strong': 'green', 'Medium': 'orange', 'Weak': 'red'}

            # Map colors to the indices presented in strength_counts
            bar_colors = [colors.get(x, 'blue') for x in strength_counts.index]

            strength_counts.plot(kind='bar', color=bar_colors)
            plt.title('Password Strength Distribution')
            plt.xlabel('Strength Category')
            plt.ylabel('Count')
            plt.tight_layout()
            plt.savefig('strength_distribution.png')
            AuditLogger.log("Visualization saved to strength_distribution.png")
            plt.close()
        except Exception as e:
            AuditLogger.log(f"Error creating visualization: {e}")
```

3] **auditor_main.py**

```python
import time
from password_class import PasswordAuditor
from audit_report import AuditReport, AuditLogger

def audit_decorator(func):
    """Decorator to log the start and end of the audit process."""
    def wrapper(*args, **kwargs):
        AuditLogger.log("Starting Security Audit...")
        start_time = time.time()
        result = func(*args, **kwargs)
        duration = time.time() - start_time
        AuditLogger.log(f"Audit completed in {duration:.2f} seconds.")
        return result
    return wrapper

@audit_decorator
def main():
    try:
        # Initialize Auditor
        auditor = PasswordAuditor()

        # 1. Batch Password Generation
        count = 50
        AuditLogger.log(f"Generating {count} random passwords...")
        auditor.generate_random_passwords(count)

        # 2. Password Auditing
        AuditLogger.log("Analyzing password strength...")
        results = auditor.run_audit()

        # 3. Audit Reporting & Visualization
        AuditLogger.log("Generating reports...")
        reporter = AuditReport(results)

        reporter.save_to_csv("audit_results.csv")
        reporter.generate_txt_report("audit_report.txt")
        reporter.create_visualizations()

        AuditLogger.log("Audit process finished successfully.")

        # Print a snippet of the output to console for verification
        print("\n--- Snippet of Audit Results ---")
        print(reporter.df.head())
        print("--------------------------------")

    except Exception as e:
        AuditLogger.log(f"An error occurred during execution: {e}")
```

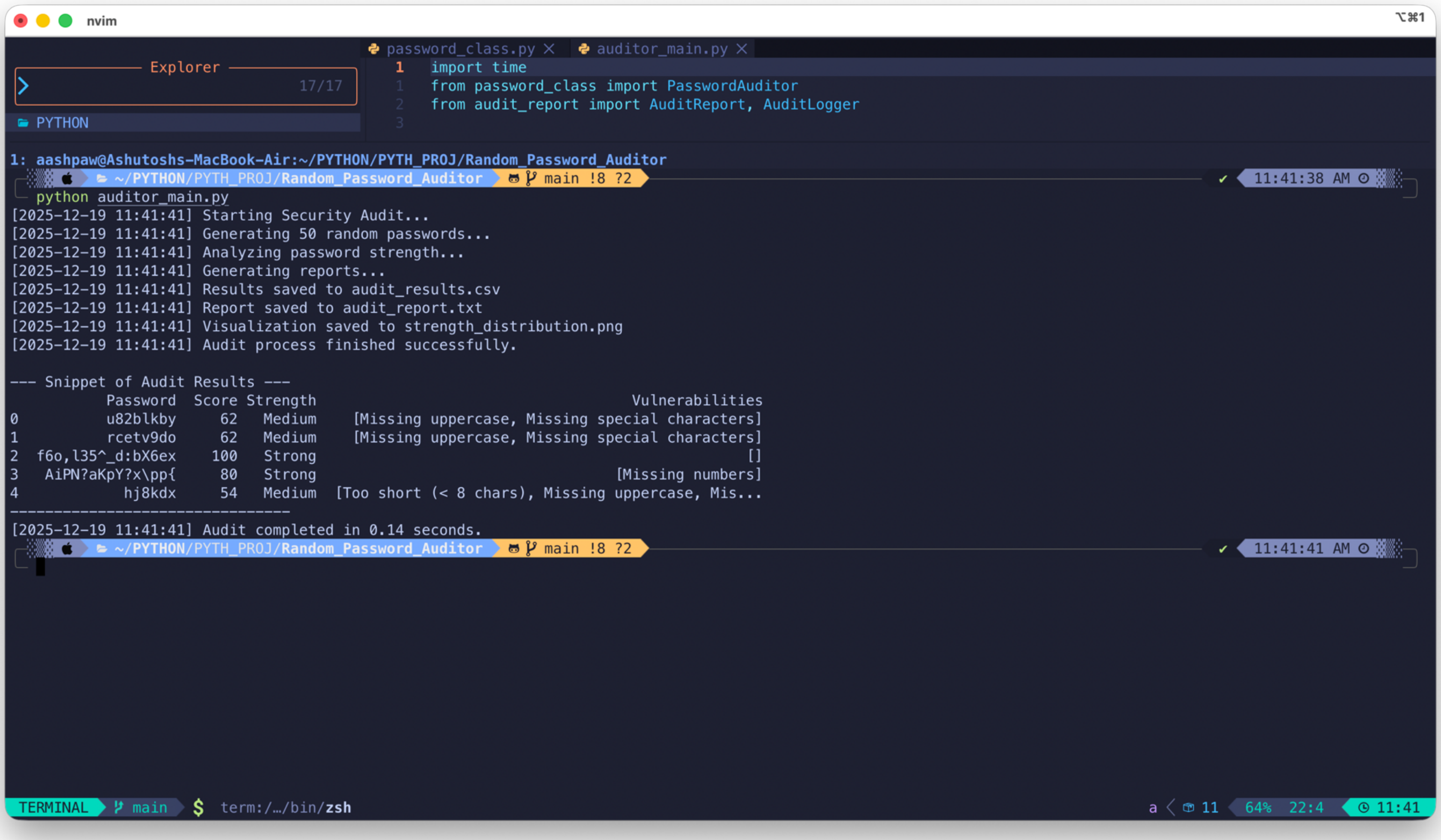
```python
if __name__ == "__main__":
    main()
```

6. Output:

This section presents the observable results produced by the Random Password Auditor during execution. The following outputs were generated and are demonstrated through screenshots and saved files:

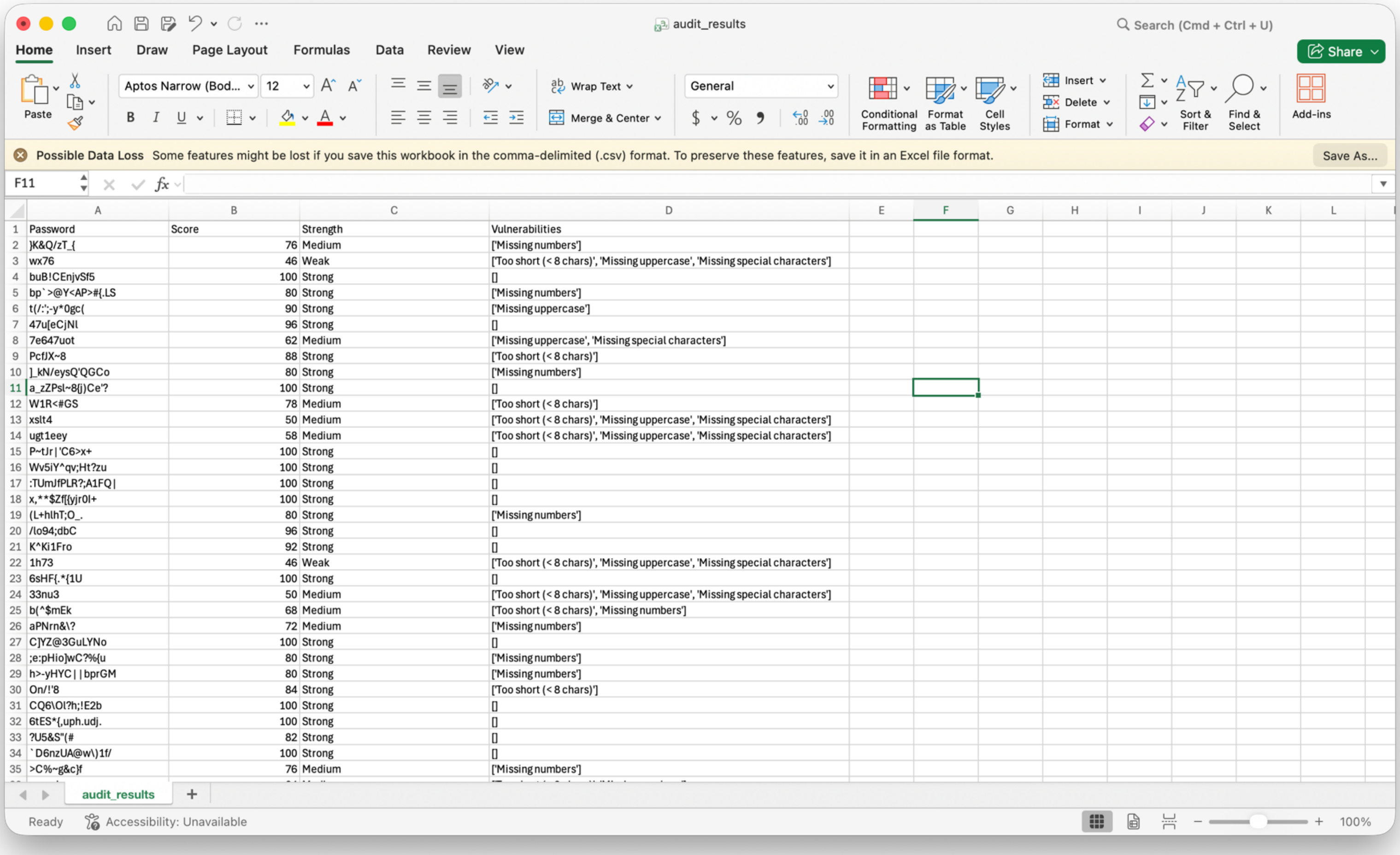
- Execution of the main audit script (`auditor_main.py`), showing automated password generation, audit progress logs, and successful completion of the security audit.
- Sample entries from the generated **audit_results.csv** file, displaying password scores, classifications, and identified vulnerabilities.
- Excerpts from the **audit_report.txt** file, highlighting the list of weak passwords, overall security score, recommendations, and audit certification status.
- Visual outputs generated using Matplotlib, including password strength distribution charts and vulnerability metrics.

These outputs collectively verify the correct functioning of the system and demonstrate the effectiveness of the automated password auditing and reporting process.



```python
import time
from password_class import PasswordAuditor
from audit_report import AuditReport, AuditLogger
```

```
1: aashpaw@Ashutoshs-MacBook-Air:~/PYTHON/PYTH_PROJ/Random_Password_Auditor
 ~/PYTHON/PYTH_PROJ/Random_Password_Auditor  main !8 ?2                    ✓  11:41:38 AM
python auditor_main.py
[2025-12-19 11:41:41] Starting Security Audit...
[2025-12-19 11:41:41] Generating 50 random passwords...
[2025-12-19 11:41:41] Analyzing password strength...
[2025-12-19 11:41:41] Generating reports...
[2025-12-19 11:41:41] Results saved to audit_results.csv
[2025-12-19 11:41:41] Report saved to audit_report.txt
[2025-12-19 11:41:41] Visualization saved to strength_distribution.png
[2025-12-19 11:41:41] Audit process finished successfully.

--- Snippet of Audit Results ---
          Password  Score Strength                                     Vulnerabilities
0         u82blkby     62   Medium    [Missing uppercase, Missing special characters]
1         rcetv9do     62   Medium    [Missing uppercase, Missing special characters]
2  f6o,l35^_d:bX6ex    100   Strong                                                []
3   AiPN?aKpY?x\pp{     80   Strong                                 [Missing numbers]
4           hj8kdx     54   Medium  [Too short (< 8 chars), Missing uppercase, Mis...
--------------------------------
[2025-12-19 11:41:41] Audit completed in 0.14 seconds.
 ~/PYTHON/PYTH_PROJ/Random_Password_Auditor  main !8 ?2                    ✓  11:41:41 AM
```

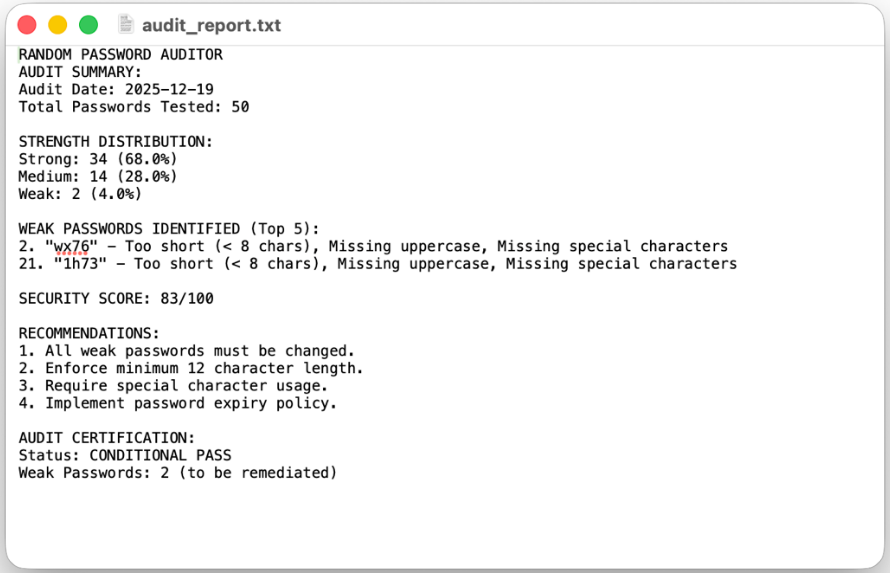

audit_results

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

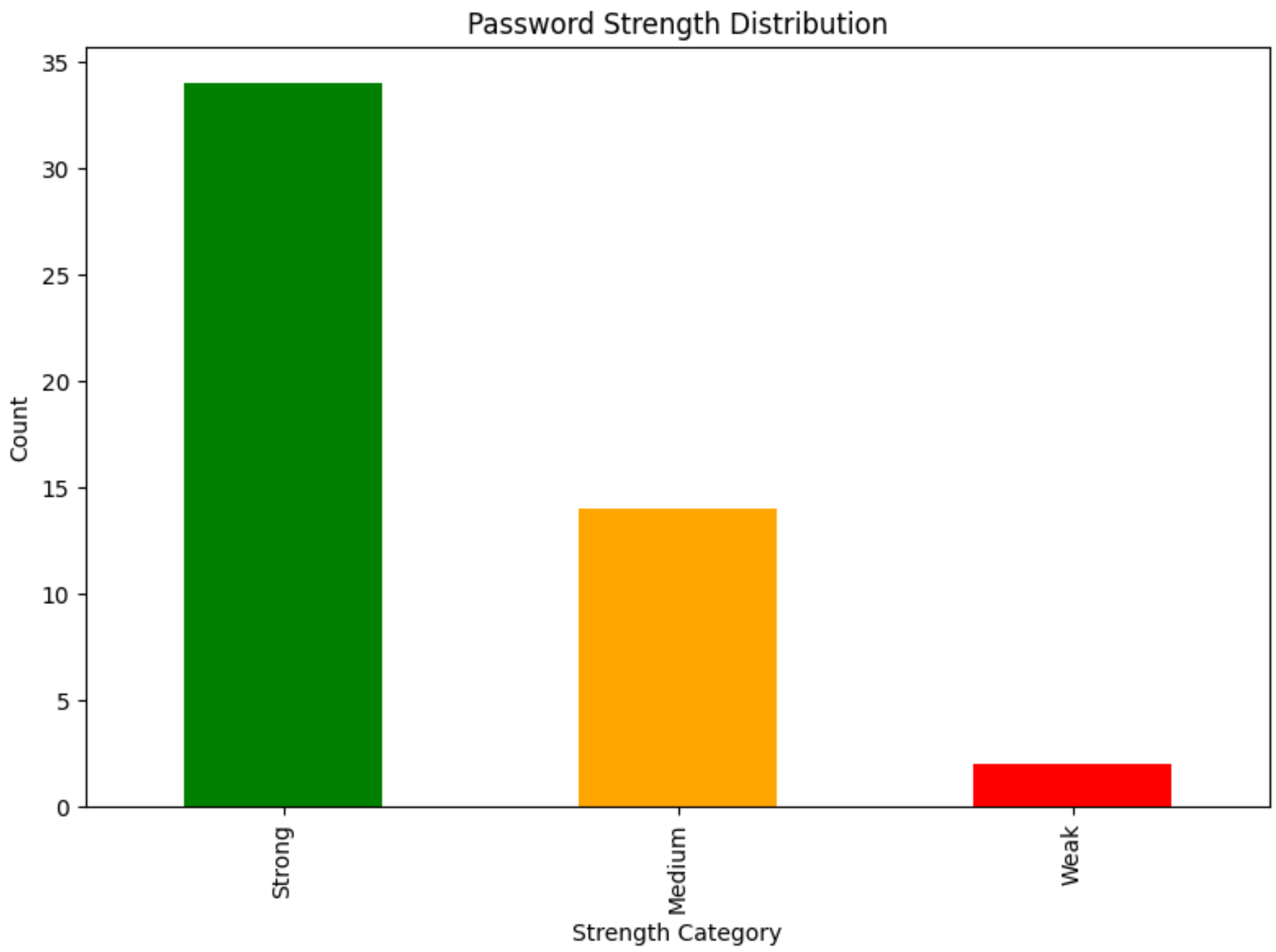
	A	B	C	D
1	Password	Score	Strength	Vulnerabilities
2	}K&Q/zT_{	76	Medium	['Missing numbers']
3	wx76	46	Weak	['Too short (< 8 chars)', 'Missing uppercase', 'Missing special characters']
4	buB!CEnjvSf5	100	Strong	[]
5	bp`>@Y<AP>#{.LS	80	Strong	['Missing numbers']
6	t(/:';-y*0gc(	90	Strong	['Missing uppercase']
7	47u[eCjNl	96	Strong	[]
8	7e647uot	62	Medium	['Missing uppercase', 'Missing special characters']
9	PcfJX~8	88	Strong	['Too short (< 8 chars)']
10	]_kN/eysQ'QGCo	80	Strong	['Missing numbers']
11	a_zZPsl~8{j)Ce'?	100	Strong	[]
12	W1R<#GS	78	Medium	['Too short (< 8 chars)']
13	xslt4	50	Medium	['Too short (< 8 chars)', 'Missing uppercase', 'Missing special characters']
14	ugt1eey	58	Medium	['Too short (< 8 chars)', 'Missing uppercase', 'Missing special characters']
15	P~tJr|'C6>x+	100	Strong	[]
16	Wv5iY^qv;Ht?zu	100	Strong	[]
17	:TUmJfPLR?;A1FQ|	100	Strong	[]
18	x,**$Zf[{yjr0I+	100	Strong	[]
19	(L+hlhT;O_.	80	Strong	['Missing numbers']
20	/lo94;dbC	96	Strong	[]
21	K^Ki1Fro	92	Strong	[]
22	1h73	46	Weak	['Too short (< 8 chars)', 'Missing uppercase', 'Missing special characters']
23	6sHF{.*{1U	100	Strong	[]
24	33nu3	50	Medium	['Too short (< 8 chars)', 'Missing uppercase', 'Missing special characters']
25	b(^$mEk	68	Medium	['Too short (< 8 chars)', 'Missing numbers']
26	aPNrn&\?	72	Medium	['Missing numbers']
27	C]YZ@3GuLYNo	100	Strong	[]
28	;e:pHio]wC?%{u	80	Strong	['Missing numbers']
29	h>-yHYC||bprGM	80	Strong	['Missing numbers']
30	On/!'8	84	Strong	['Too short (< 8 chars)']
31	CQ6\Ol?h;!E2b	100	Strong	[]
32	6tES*{,uph.udj.	100	Strong	[]
33	?U5&S"(#	82	Strong	[]
34	`D6nzUA@w\)1f/	100	Strong	[]
35	>C%~g&c}f	76	Medium	['Missing numbers']

audit_results

Ready Accessibility: Unavailable


audit_report.txt

```
RANDOM PASSWORD AUDITOR
AUDIT SUMMARY:
Audit Date: 2025-12-19
Total Passwords Tested: 50

STRENGTH DISTRIBUTION:
Strong: 34 (68.0%)
Medium: 14 (28.0%)
Weak: 2 (4.0%)

WEAK PASSWORDS IDENTIFIED (Top 5):
2. "wx76" - Too short (< 8 chars), Missing uppercase, Missing special characters
21. "1h73" - Too short (< 8 chars), Missing uppercase, Missing special characters

SECURITY SCORE: 83/100

RECOMMENDATIONS:
1. All weak passwords must be changed.
2. Enforce minimum 12 character length.
3. Require special character usage.
4. Implement password expiry policy.

AUDIT CERTIFICATION:
Status: CONDITIONAL PASS
Weak Passwords: 2 (to be remediated)
```


Password Strength Distribution
Count
35
30
25
20
15
10
5
0
Strong
Medium
Weak
Strength Category

7. References:

1. **Harvard University – CS50P: Introduction to Programming with Python**
   Harvard CS50P course materials and lectures were referenced for Python fundamentals, functional programming concepts, decorators, and best practices in writing clean, readable, and modular Python code.
2. **Python Software Foundation.** ***Python Documentation***
   Official Python documentation was used to understand core language features such as decorators, lambda expressions, exception handling, and file I/O.
   https://docs.python.org/3/ , https://docs.python.org/3/tutorial/classes.html , https://docs.python.org/3/library/functions.html#any
3. **Pandas Development Team.** ***Pandas Documentation***
   Referenced for DataFrame creation, data manipulation, iteration methods, and CSV file handling used in audit reporting.
   https://pandas.pydata.org/docs/
4. **Matplotlib Developers.** ***Matplotlib Documentation***
   Used for learning data visualization techniques, including plotting distributions and generating audit charts.
   https://matplotlib.org/stable/
5. **OWASP Foundation.** ***Password Security Guidelines***
   Referenced for common password weaknesses, strength evaluation criteria, and best practices in password security auditing.
   https://owasp.org/