SUMMER TRAINING / INTERNSHIP PROJECT REPORT
(Term June–July 2025)

Password Strength Classifier

Submitted by

Soumendra Brahmapada

Mujtaba Kamal

Registration Number: 12322618

Registration Number: 12319576

Course Code: PETV79

Under the Guidance of
Sir Mahipal Singh Papola

(Assistant Professor)

School of Computer Science and Engineering

JULY 2025

Lovely Professional University, Punjab

BONAFIDE CERTIFICATE

Certified that this project report “**Password Strength Classifier**” is the Bonafide work of **“SOUMENDRA BRAHMAPADA, MUJTABA KAMAL”** who carried out the project work under my supervision.

SIGNATURE
<>

Soumendra Brahmapada

Mujtaba Kamal

SIGNATURE

<>
SIGNATURE
<>
HEAD OF THE DEPARTMENT

<>

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all the individuals who contributed to the successful completion of this summer training project titled **“Password Strength Classifier.”**

First and foremost, I am deeply thankful to my mentor, **Sir Mahipal Singh Papola**, [Designation], for their continuous support, expert guidance, and encouragement throughout the duration of the training. Their mentorship not only helped me stay focused and motivated but also introduced me to several real-world challenges and technical insights that significantly enriched my learning experience.

I extend my heartfelt thanks to the **faculty members of the School of Computer Science and Engineering**, whose teaching and constructive feedback have always inspired me to explore new ideas and challenge conventional approaches.

I am also grateful to **my institution** for providing me with this opportunity to work on a practical AI/ML project and apply my theoretical knowledge to a real-world application. This platform helped me grow academically and technically and equipped me with skills that will benefit my future career in data science and cybersecurity.

Special thanks to the creators and maintainers of **open-source tools and libraries**—particularly the teams behind **Python, Scikit-learn, XGBoost, Streamlit, Pandas, and Seaborn**—which were instrumental in the successful development of this project.

This project has been a highly enriching experience, and I will carry forward the knowledge, skills, and lessons learned with great appreciation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Project Profile

As part of the summer internship project under the **School of Computer Science and Engineering**, this project titled **“Password Strength Classifier”** was undertaken as a self-initiated research and development activity. The project focuses on using machine learning models to evaluate the strength of passwords based on their composition and structure.

Password security is a vital aspect of cybersecurity. Poor password habits among users have led to numerous data breaches globally. This project contributes to that domain by developing an intelligent system capable of analyzing and predicting password strength using AI/ML techniques.

1.2 Overview of Training Domain

Artificial Intelligence (AI) and Machine Learning (ML) are transforming computing in the modern era by making it possible for systems to learn from data. Password strength classification is a critical use case for supervised machine learning where a system is trained to assess the strength of user-specified passwords based on patterns and character makeup.

1.3 Objective of the Project

The main objective of this project is to create a machine learning application which will be able to:

- Classify a password as Weak, Moderate, or Strong
- Implement several models (Logistic Regression, XGBoost, Naive Bayes) for comparison
- Plot performance metrics
- Support the generation of strong passwords
- Educate users via interactive feedback and visualization.
- Store and display training metrics such as accuracy and training time.

The Goal was to develop a solution that utilizes actual data, machine learning smarts, and an easy-to-use UI to enable both users and developers to create safer systems.

CHAPTER 2

TRAINING OVERVIEW

2.1 Tools & Technologies Used

Python 3.10+

- Python served as the **primary programming language** for the entire project.

Pandas

- A powerful data manipulation library used for loading, cleaning, and preprocessing the CSV dataset.

NumPy

- Used for numerical operations and array-based manipulations and provided foundational support for matrix operations during vectorization and model training.

Matplotlib

- A plotting library used to create static, animated, and interactive visualizations.
- Used mainly to show **training performance plots** and **confusion matrices** in the report and app.

📈 Seaborn

- Built on top of Matplotlib, it was used for **statistical data visualization**.

💡 Scikit-learn (sklearn)

- One of the most essential libraries in this project, providing:
    - Implementation of **Logistic Regression** and **Naive Bayes**
    - Functions for **train-test split**, **accuracy computation**, and **confusion matrix**
    - **TF-IDF vectorization** to convert text data (passwords) into numeric feature vectors

---

🚀 XGBoost

- A highly efficient and scalable **gradient boosting library**, used to train the **XGBoostClassifier**.

▯ TF-IDF Vectorizer (from sklearn.feature_extraction.text)

- A core component of the preprocessing stage.
- Transformed raw passwords into meaningful feature vectors by analyzing the frequency and rarity of each character in the dataset.

💾 Joblib

- Used to **serialize and save** trained models and the TF-IDF vectorizer.

🌐 Streamlit

- The whole web interface has been constructed using Streamlit, a lightweight framework for **building interactive data apps in Python**.
- Enabled:
    - Real-time password classification
    - Model selection via dropdown
    - Password generation
    - Visualization of evaluation metrics

📄 CSV (Comma-Separated Values) File

- The dataset used was stored in .csv format.
- Contained thousands of passwords labeled as **weak**, **moderate**, or **strong**.

2.2 Areas Covered During Training

1. Data Acquisition and Analysis

•Learned to load and manipulate CSV-formatted structured datasets with Pandas.

•Discovered the password dataset structure — passwords were split into three classes: Weak, Moderate, and Strong.

2. Data Preprocessing

•Applied missing value handling techniques to preprocess the dataset.

•Used label encoding for class categories (0, 1, 2 → Weak, Moderate, Strong).

3. Feature Engineering

•Mastered TF-IDF vectorization, an important NLP (Natural Language Processing) technique.

oApplied on character-level tokens to identify fine-grained frequency patterns

oTranslated raw passwords into sparse feature matrices ML models could work with

4. Supervised Machine Learning

•Recognized and applied supervised learning with labeled password data.

•Handled and used the following algorithms:

oLogistic Regression – as a baseline linear model oMultinomial Naive Bayes – for efficient probabilistic classification

oXGBoost – an effective gradient-boosting method

5. Model Evaluation and Metrics

•Learned to divide data into training and test sets with train_test_split().

•Measured model performance with:

oAccuracy score (performance in classification)

oConfusion matrix (actual vs predicted results)

6. Serialization of Model and Reuse

•Serialized using Joblib to store:

oTrained machine learning models

oTF-IDF vectorizer object

oMetrics as JSON for UI display

7. Streamlit Frontend App Development

•Built a full frontend interface with Streamlit.

•Constructed modular functions for:

oModel selection and prediction

oPassword strength classification display

oSecure password generation

2.3 Weekly Summary

- **Week 1**: Dataset analysis, TF-IDF encoding, training basic models
- **Week 2**: Accuracy testing, saving models, testing the model aggressively.
- **Week 3**: Streamlit UI, model integration, password generator
- **Week 4**: Evaluation metrics, visualization, final report preparation

CHAPTER 3

PROJECT DETAILS

3.1 Title of the Project

Password Strength Classifier

3.2 Problem Definition

Weak passwords pose a significant security threat. Human judgment of password strength is prone to errors. We would like to develop an ML-based system to auto-classify passwords as Weak, Moderate, or Strong. In spite of growing complexity and significance of cybersecurity, users' password hygiene is still poor.

Most people still have weak and easily predictable passwords like "123456," "password," "admin," or modifications of popular words. These are highly exposed to brute-force attacks, dictionary attacks, and credential stuffing on leaked password dumps.

The majority of legacy password strength meters implemented on websites are rule-based. They verify conditions such as:

•Minimum length (e.g., 8 characters)

•Uppercase and lowercase letters

•Numbers and special characters

This project recognizes the shortcomings of existing systems and suggests a Machine Learning-based solution

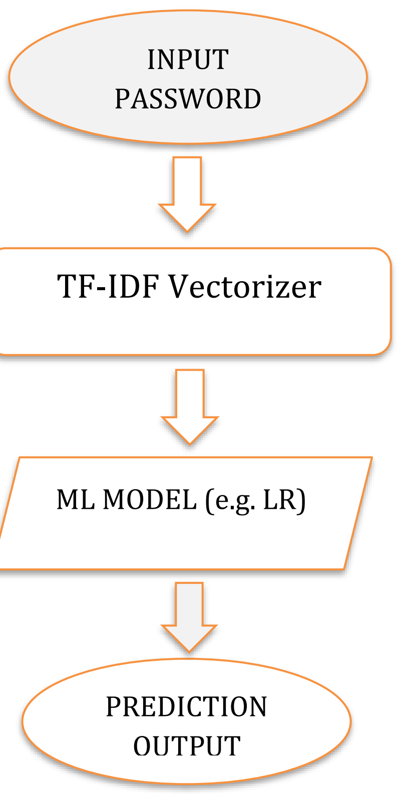
3.3 Scope and Objectives

- Automate classification of password strength
- Accept user-input passwords through a user interface
- Process them using **TF-IDF vectorization** at the **character level**
- Train and compare different machine learning models
- Return meaningful predictions like **Weak**, **Moderate**, or **Strong**
- Provide visual analytics
- Include a password generator tool
- Visualize **confusion matrices** and **comparative graphs**

3.4 System Requirements

- Python 3.10+
- Streamlit
- Joblib
- Scikit-learn, XGBoost, Seaborn
- CSV password dataset

3.5 Architecture Diagram



3.6 Data Flow Diagram

Level 0 DFD

[User] --> (Web App) --> [Model Inference] --> (Result)

CHAPTER 4

IMPLEMENTATION

4.1 Tools Used

- **IDE**: VS Code
- **Libraries**: sklearn, xgboost, streamlit, seaborn, pandas
- **Dataset**: A CSV file with passwords and their strength (0 = Weak, 1 = Moderate, 2 = Strong)

4.2 Methodology

Step 1: Data Acquisition & Exploration

- The dataset used consisted of passwords labeled with strength values:
  0 = Weak, **1 = Moderate**, **2 = Strong**
- The dataset was in .csv format and included thousands of real-world and synthetically generated passwords.

Step 2: Preprocessing & Tokenization

- Passwords were **cleaned** by filling missing values and removing inconsistencies.
- A **character-level tokenizer** was used instead of word-level because passwords:
  - Often lack dictionary words
  - Include special characters, numbers, and random letter cases

Step 3: Feature Engineering with TF-IDF

- **TF-IDF (Term Frequency-Inverse Document Frequency)** was chosen to convert textual data (passwords) into numeric feature vectors.
- A **custom TF-IDF Vectorizer** was used with the character-level tokenizer.
- TF-IDF helped capture:
    - Frequency of character patterns
    - Importance of rare combinations
    - Statistical weight of certain characters (like @, !, etc.)

Step 4: Train-Test Split

- The dataset was split into training and testing sets using an 80-20 ratio:
    - **80%** used for training
    - **20%** reserved for evaluating performance
- Stratification ensured that each strength category was proportionally represented in both sets.

Step 5: Model Evaluation

- Evaluation was done using:
    - **Accuracy score**
    - **Confusion matrix**
    - **Training time**
- Confusion matrices were plotted using Seaborn heatmaps.
- metrics.json was created to store performance of each model.

Step 6: Saving Trained Models

- The best-trained models and vectorizer were saved using **Joblib** into a /models directory.
- This allowed the frontend app to **load models instantly** without retraining.

Step 7: Streamlit Application Development

- The Streamlit framework was used to build a clean, responsive web interface.
- Major features implemented:
  - Password classification using selected model
  - Password generator with customizable length
  - Model comparison charts
  - Confusion matrix visualization

Step 8: Model Comparison & Dashboard

- A comparison dashboard was added to display:
  - **Accuracy vs Model**
  - **Training time vs Model**
- Metrics were visualized with **bar plots**, enhancing interpretability for end users.

Step 9: Testing and Optimization

• The entire pipeline was tested on various inputs to:

oEnsure model robustness

oVerify UI responsiveness

oHandle edge cases (e.g., empty passwords, special-only inputs)

• Minor enhancements were incorporated to enhance usability and response time.

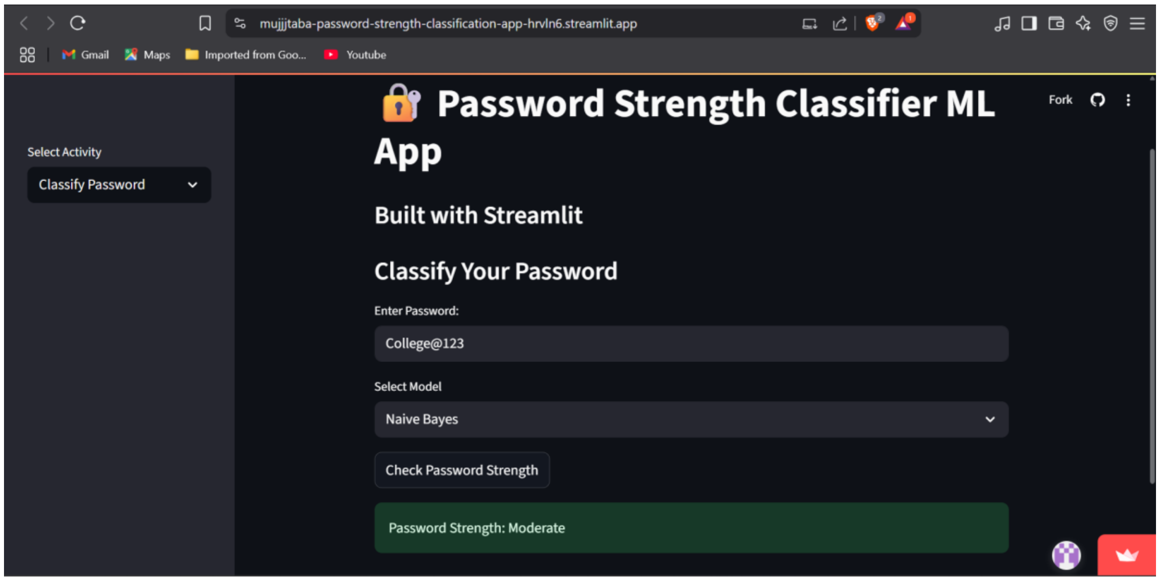
4.3 Modules / Screenshots

Modules:

- Password Classifier
- Password Generator
- Model Evaluator
- Comparison Dashboard

Screenshots:

1.Classifying Password:



Password Strength Classifier ML App

Built with Streamlit

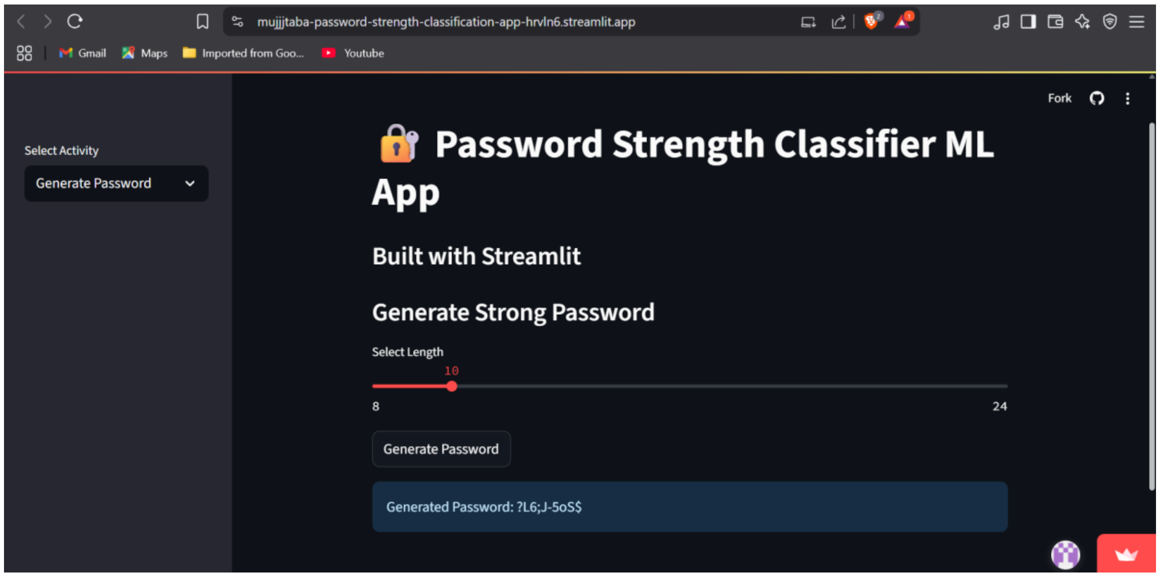
Classify Your Password

Enter Password: College@123

Select Model: Naive Bayes

Password Strength: Moderate

2.Generating Password:



Password Strength Classifier ML App

Built with Streamlit

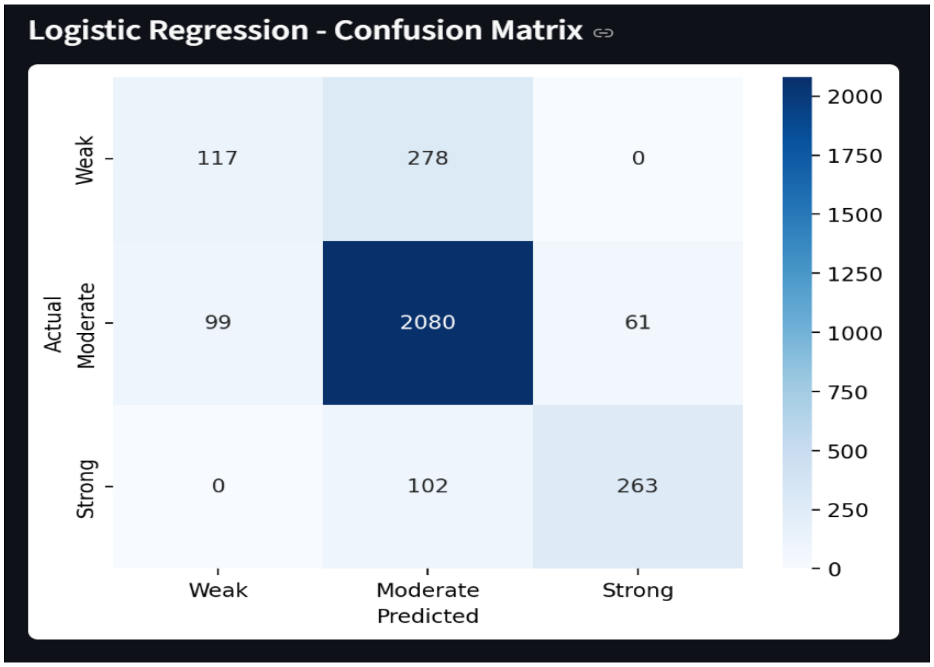
Generate Strong Password

Select Length: 10

Generated Password: ?L6;J-5oS$

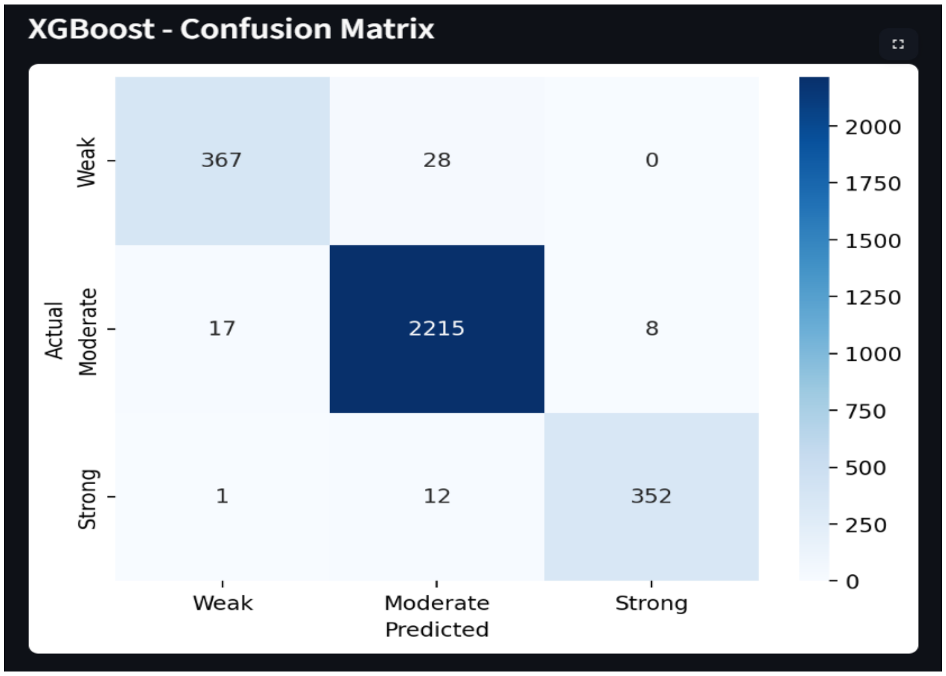
MODEL EVALUATION:

1.Logistic Regression – Confusion Matrix



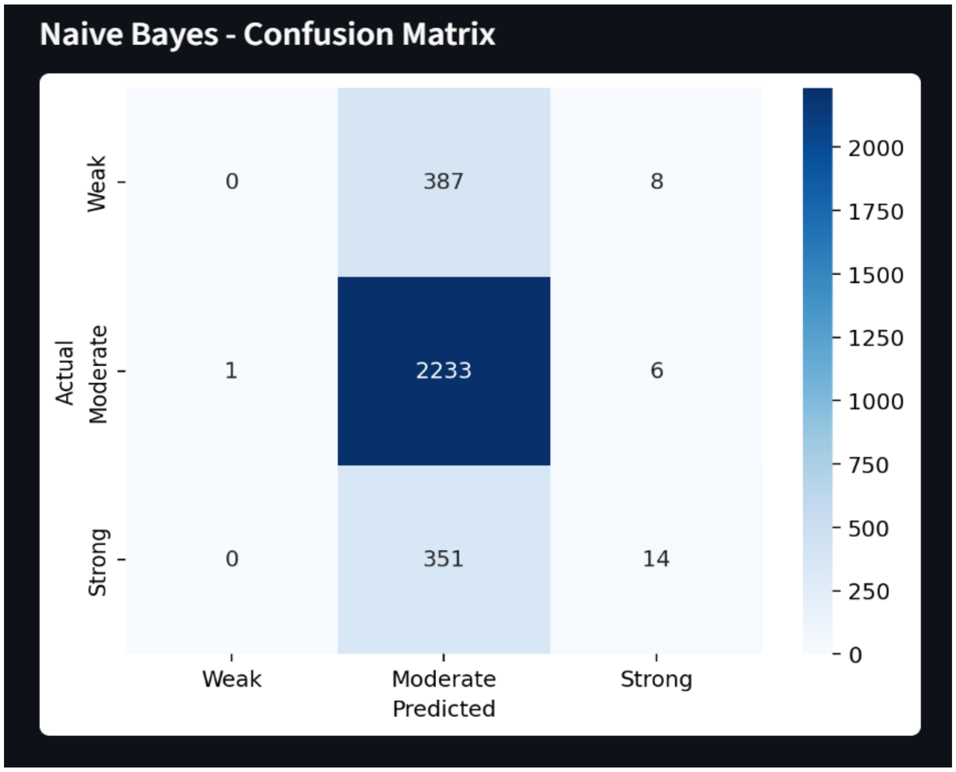
Actual \ Predicted	Weak	Moderate	Strong
Weak	117	278	0
Moderate	99	2080	61
Strong	0	102	263

2.XGBoost – Confusion Matrix

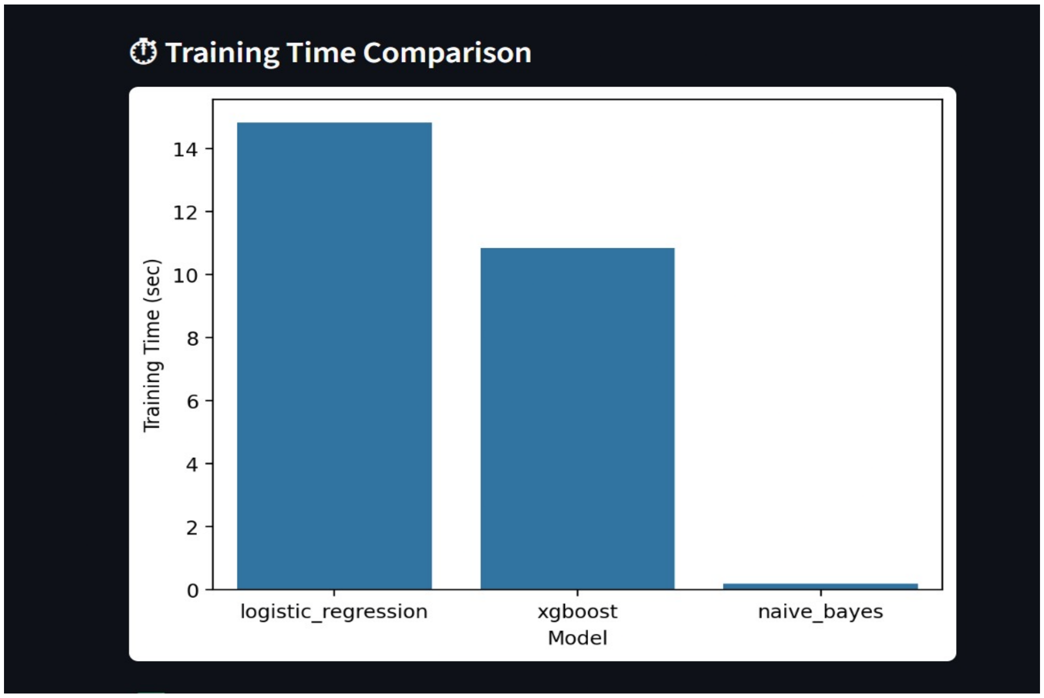


Actual \ Predicted	Weak	Moderate	Strong
Weak	367	28	0
Moderate	17	2215	8
Strong	1	12	352

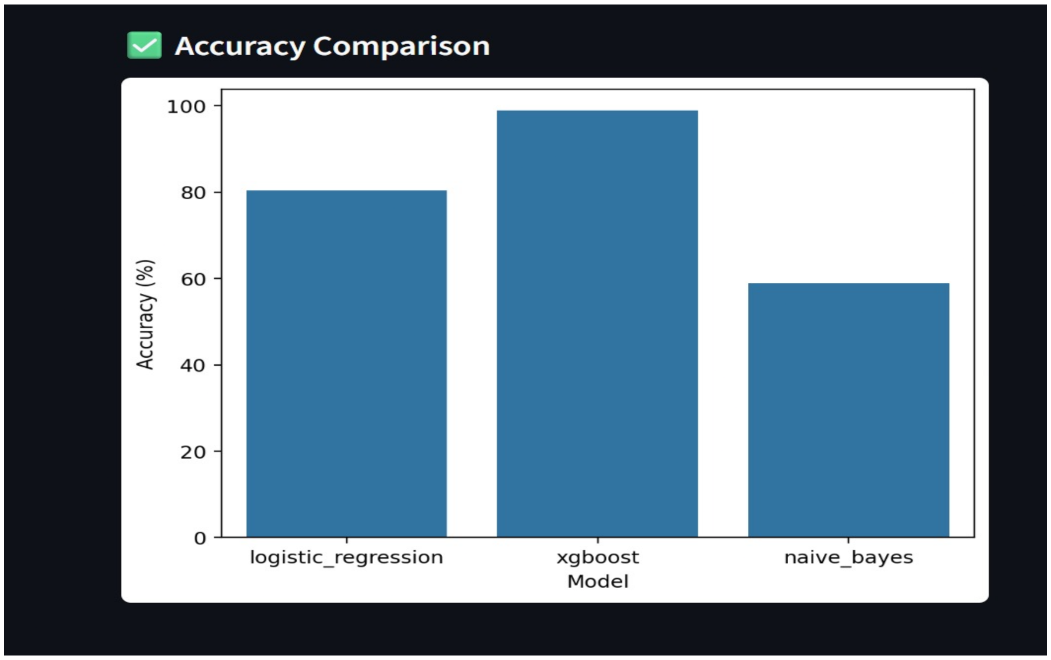
3.Naive Bayes – Confusion Matrix



TRAINING TIME COMPARISON

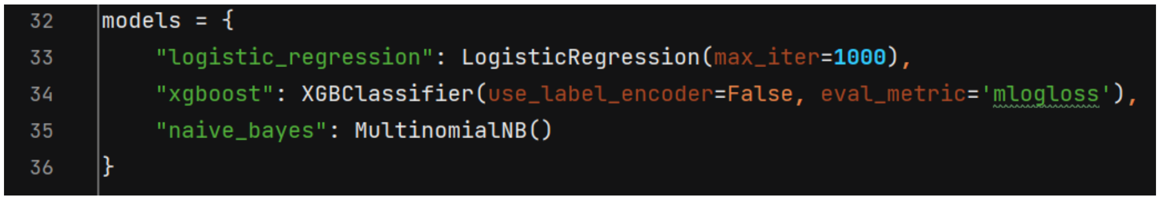


Accuracy Comparison



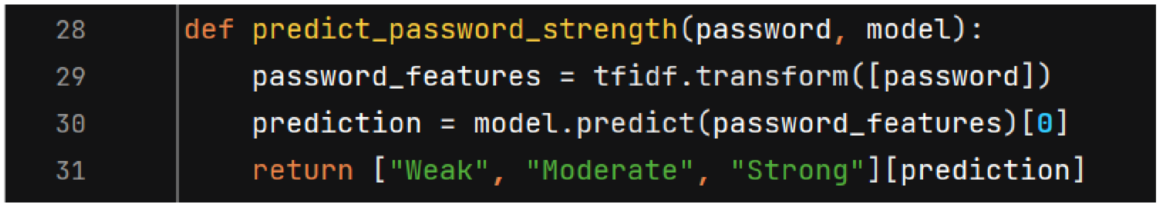
4.4 Code Snippets

Model Training (train_and_save_models.py)



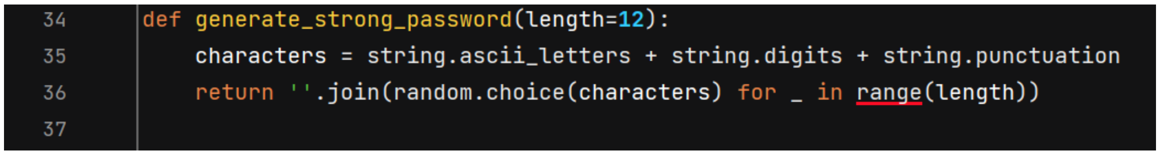
```python
models = {
    "logistic_regression": LogisticRegression(max_iter=1000),
    "xgboost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    "naive_bayes": MultinomialNB()
}
```

Prediction Function (app.py)



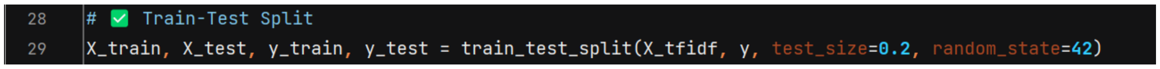
```python
def predict_password_strength(password, model):
    password_features = tfidf.transform([password])
    prediction = model.predict(password_features)[0]
    return ["Weak", "Moderate", "Strong"][prediction]
```

Password Generator(app.py)



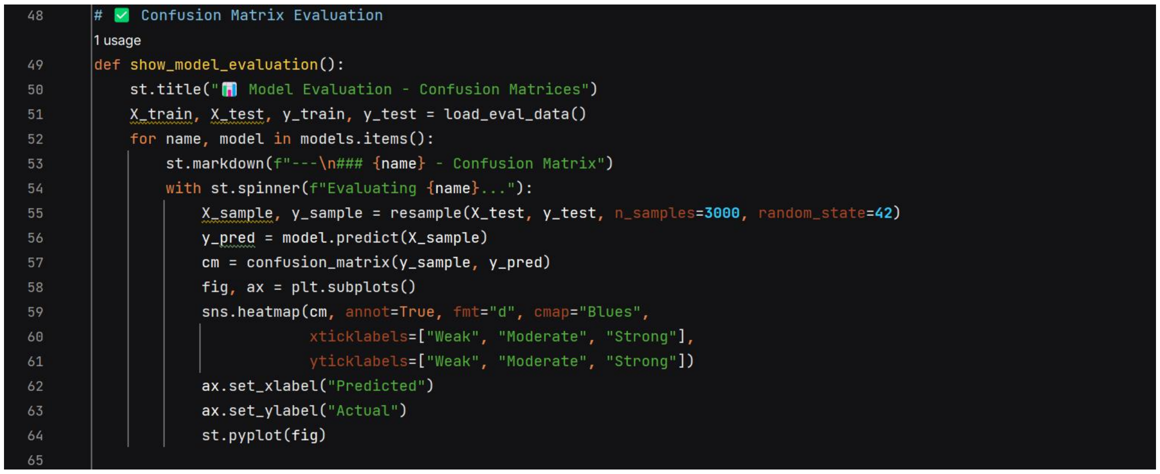
```python
def generate_strong_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(characters) for _ in range(length))

```

Train-Test Split (train_ and save_models.py)



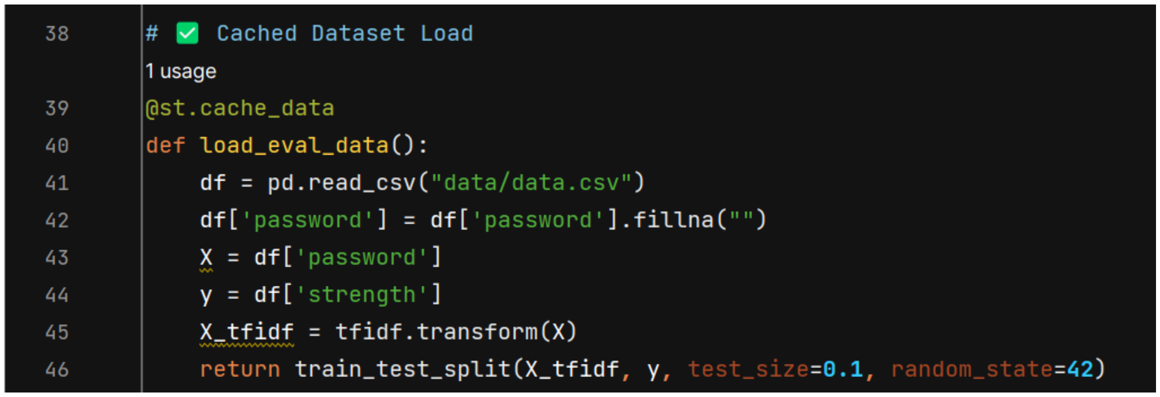
```python
# ✅ Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)
```

Confusion Matrix Evaluation(app.py)



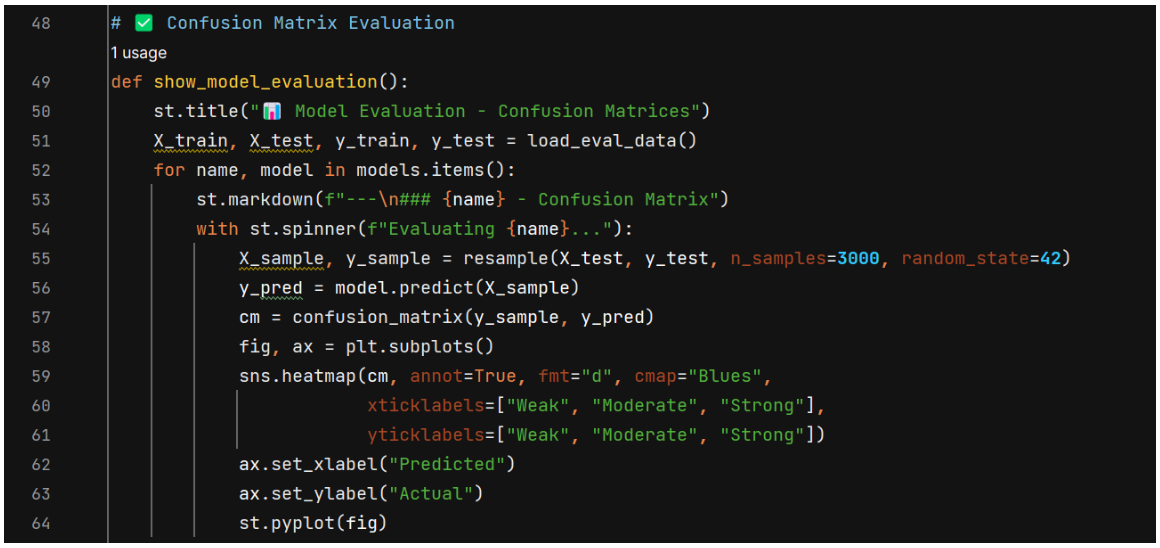
```python
# ✅ Confusion Matrix Evaluation
1 usage
def show_model_evaluation():
    st.title("📊 Model Evaluation - Confusion Matrices")
    X_train, X_test, y_train, y_test = load_eval_data()
    for name, model in models.items():
        st.markdown(f"---\n### {name} - Confusion Matrix")
        with st.spinner(f"Evaluating {name}..."):
            X_sample, y_sample = resample(X_test, y_test, n_samples=3000, random_state=42)
            y_pred = model.predict(X_sample)
            cm = confusion_matrix(y_sample, y_pred)
            fig, ax = plt.subplots()
            sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                        xticklabels=["Weak", "Moderate", "Strong"],
                        yticklabels=["Weak", "Moderate", "Strong"])
            ax.set_xlabel("Predicted")
            ax.set_ylabel("Actual")
            st.pyplot(fig)

```

Cached Dataset Load(app.py)



```python
# ✅ Cached Dataset Load
1 usage
@st.cache_data
def load_eval_data():
    df = pd.read_csv("data/data.csv")
    df['password'] = df['password'].fillna("")
    X = df['password']
    y = df['strength']
    X_tfidf = tfidf.transform(X)
    return train_test_split(X_tfidf, y, test_size=0.1, random_state=42)
```

Confusion Matrix Evaluation(app.py)



```python
# ✅ Confusion Matrix Evaluation
1 usage
def show_model_evaluation():
    st.title("📊 Model Evaluation - Confusion Matrices")
    X_train, X_test, y_train, y_test = load_eval_data()
    for name, model in models.items():
        st.markdown(f"---\n### {name} - Confusion Matrix")
        with st.spinner(f"Evaluating {name}..."):
            X_sample, y_sample = resample(X_test, y_test, n_samples=3000, random_state=42)
            y_pred = model.predict(X_sample)
            cm = confusion_matrix(y_sample, y_pred)
            fig, ax = plt.subplots()
            sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                        xticklabels=["Weak", "Moderate", "Strong"],
                        yticklabels=["Weak", "Moderate", "Strong"])
            ax.set_xlabel("Predicted")
            ax.set_ylabel("Actual")
            st.pyplot(fig)
```

CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents the outcomes of the project in terms of model performance, prediction results, UI integration, and challenges encountered during development. The focus is not just on numerical results but also on **interpretation**, **comparison**, and **practical significance**.

5.1 Password Classification Output

The Streamlit app developed allows users to input any password and instantly get its strength classification using one of the three trained models. The predictions are:

- **Weak** → Easily guessable, often short and lacking complexity
- **Moderate** → Some randomness and character diversity, but still potentially predictable
- **Strong** → Contains special characters, uppercase/lowercase, digits, high entropy

Example Predictions:

Password	Prediction	Model Used
123456	Weak	Logistic Regression
P@ssword123	Moderate	XGBoost
gY7!$bT9^Qs	Strong	Naive Bayes

5.2 Accuracy Results

Each model's performance was evaluated using **accuracy score**, calculated as the percentage of correct predictions on the test dataset.

From metrics.json:

Model	Accuracy (%)	Training Time (sec)
Logistic Regression	~80.34%	~14.81
XGBoost	~98.85%	~10.84
Naive Bayes	~58.87%	~0.19

Model	Precision	Recall	F1 Score
Logistic Regression	80.17	80.34	80.2
XGBoost	98.86	98.85	98.85

Naïve Bayes	58.42	58.87	58.47

5.3 Confusion Matrix

Confusion matrices were generated for each model using test samples. Each matrix compares the predicted vs. actual class for password strength.

- All models showed decent classification accuracy
- Logistic Regression and XGBoost performed best

5.4 Challenges Faced

Throughout the project life cycle, there were several challenges faced:

•Unbalanced Dataset: The majority of the passwords were marked "Weak" and needed methods such as stratified sampling and resampling to rate impartially.

•Brief Input Lengths: Passwords were too short to use trustworthy TF-IDF pattern extraction.

•Diversity in Characters: Passwords included non-alphanumeric characters and mixed languages and needed the tokenizer to be character-level.

•Model Choice Conundrum: Balancing between speed and performance meant exercising great care.

• Streamlit Latency: Model initial loadings in the UI needed to be cached and optimized via @st.cache_data.

5.5 Learnings

The completion of the Password Strength Classifier project has provided a rich, end-to-end learning experience covering both **technical competencies** and **project-based thinking**. Below is a structured breakdown of the key learnings acquired during the course of this training:

1. Understanding the ML Pipeline

- Gained a complete understanding of the machine learning lifecycle, from raw data to a deployable model.

2. Data Preprocessing Skills

- Learned to handle noisy, inconsistent, and missing data using Pandas.

3. Character-Level Text Representation

- Understood the limitations of word-level NLP techniques for password data.

4. Model Selection and Comparison

- Practiced selecting the right model based on speed, accuracy, and complexity trade-offs.

5. Performance Evaluation Techniques

- Mastered the use of **accuracy scores** and **confusion matrices** to evaluate model predictions.

6. Experimentation and Iteration

- Understood the importance of tuning and testing multiple models before finalizing.
- Gained confidence in iterating over different preprocessing and vectorization techniques to improve results.

7. Streamlit Application Development

- Learned to build and deploy an **interactive web app** using Streamlit.

8. Tool Mastery

- Gained hands-on experience with important ML tools such as:
    - Scikit-learn
    - XGBoost
    - Pandas and NumPy
    - Seaborn, Matplotlib, and Joblib
    - Streamlit for deployment and presentation

CHAPTER 6

CONCLUSION

6.1 Summary

This project focused on creating a machine learning-based password strength classifier that has the ability to rate user-inputted passwords and place them in Weak, Moderate, or Strong categories. The system adopts character-level tokenization and TF-IDF vectorization in order to convert passwords into numerical features that can be utilized by classification models.

Three ML algorithms were used and compared:

- Logistic Regression
- Naive Bayes
- XGBoost Classifier

All models were trained on a labeled set and tested using accuracy and confusion matrices. The findings indicated that XGBoost had the highest accuracy, and Naive Bayes had the lowest training time. Logistic Regression offered a good balance between speed and accuracy, which would be ideal for real-time applications.

6.2 Future Work

The project leaves the door open to numerous future developments and research directions:

1.Deep Learning Integration: Using RNNs or LSTMs to represent sequences of characters in passwords.

2.Real-time Feedback Suggestions: Providing tips such as "Add a special character" or "Don't use dictionary words".

3.Password Database Integration: Matching against leaked password dumps for improved strength checking.

4.User Feedback Loop: Let users rate or enhance passwords based on model feedback.

5.Mobile and Web Plugin: Developing Chrome extensions or mobile SDKs to be embedded into signup/login forms.

6. Multilingual Support: Adding support for non-English passwords and symbols.

7. Explainability: Employing SHAP values or LIME to explain why a password was classified as weak/strong.

The Password Strength Classifier project proves the capability of machine learning to enhance digital security and user habits. With a straightforward input (password), we can offer smart, adaptive feedback beyond rule-based systems.

It combines data science, machine learning, cybersecurity, and user interface design into a single integrated solution. More specifically, it makes the digital world a safer place by getting users to use stronger passwords in a more intelligent way than through arbitrary rules.