**Annexure-I**

**Term Paper**

**STUDENT JOB PROFILE SUGGESTER**

**A Term Paper Report**

**Submitted in partial fulfilment of the requirements for the**

**award of degree of**

**Bachelor of Technology**

**(Computer Science Engineering)**

A logo with text overlay

Description automatically generated**Submitted to**

**LOVELY PROFESSIONAL UNIVERSITY**

**PHAGWARA, PUNJAB**

**From 1st Aug 2024 to 25th Oct 2024**

**SUBMITTED BY**

**Name: Abhishek Kumar**

**Registration Number: 12213692**

**Faculty: Sajjad Manzoor Mir**

**Annexure-II: Student Declaration**

**To Whom It May Concern:**

I, **Abhisek Kumar, 12213692**, hereby declare that the work done on **“Student Job Profile Suggester”** from Aug 2024 to October 2024, is a record of original work for the partial fulfilment of the requirements for the award of the degree, Bachelor of Technology.

**Name of the student:** Abhishek Kumar

**Registration Number:** 12213692

**Dated: 18TH JANUARY 2025**

# **ACKNOWLEDGMENT**

Primarily I would like to thank God for being able to learn a new technology. Then I would like to express my special thanks of gratitude to the teacher and instructor of the course Machine Learning who provided me with the golden opportunity to learn a new technology.

I would also like to thank my college, Lovely Professional University, for offering such a course, which not only improved my programming skills but also taught me other new technologies.

Then I would like to thank my parents and friends who helped me with valuable suggestions and guidance for choosing this course.

Finally, I would like to thank everyone who greatly helped me.

**Dated: 18TH JANUARY 2025**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Contents** | **Page** |
| **1** | **Title** | **1** |
| **2** | **Student Declaration** | **2** |
| **3** | **Acknowledgment** | **2** |
| **4** | **Table of Contents** | **3** |
| **5** | **Abstract** | **4** |
| **6** | **Objective** | **4** |
| **7** | **Introduction** | **5-7** |
| **8** | **Theoretical Background** | **7-10** |
| **9** | **Hardware & Software** | **10-13** |
| **10** | **Methodology** | **13-34** |
| **11** | **Flowchart** | **34-36** |
| **12** | **Results** | **36-38** |
| **13** | **Summary** | **38-39** |
| **14** | **Conclusion** | **39** |
| **15** | **Bibliography** | **39-40** |
| **16** | **Annexure** | **40** |

|  |
| --- |
| App Link: <https://student-job-profile-suggester.streamlit.app/>  Dataset link: [Jobs Dataset (Kaggle)](https://www.kaggle.com/datasets/yuvjeetarora/student-job-profile)  Github link: [student-job-profile-suggester](https://github.com/akupadhyay8/Student-Job-Profile-Suggester) |

1. **ABSTRACT**

In today's competitive job market, students often struggle to identify the right career paths that align with their academic strengths and technical skills. This project aims to predict suitable job profiles for students using machine learning algorithms. The model is trained on a comprehensive dataset containing students' scores in various academic subjects such as Data Structures and Algorithms (DSA), Database Management Systems (DBMS), and Mathematics, along with information about their proficiency in different skills such as Python, JavaScript, and Machine Learning. The dataset comprises both numerical and categorical features, and the target variable is the student's preferred job profile.

Multiple machine learning models, including Decision Trees, Random Forests, Gradient Boosting, and XGBoost, were implemented and evaluated. Through rigorous hyperparameter tuning and performance comparison, the XGBoost model emerged as the most accurate with an overall accuracy of 97%. To facilitate user interaction, a Streamlit-based web application was developed. This app allows users to input their scores and skills through an intuitive interface, after which the trained model predicts their job profile in real-time. The integration of machine learning with a user-friendly web interface ensures that students can access personalized career guidance based on their academic and skill-related data. This project demonstrates the potential of data-driven methods to assist students in making informed career choices.

1. **OBJECTIVE**

The primary objective of this project is to develop an intelligent system that predicts a student’s most suitable job profile based on their academic performance and technical skills. With the vast number of career options available in the technology and engineering domains, students often find it challenging to identify roles that best match their strengths and capabilities. This project aims to address this gap by building a machine learning model capable of analysing students' subject scores and skillsets, thereby providing tailored job profile suggestions.

The system is designed to offer accurate predictions by utilizing both numerical data (e.g., scores in subjects like DSA, DBMS, and Mathematics) and categorical data (e.g., proficiency in skills such as Python, Java, and Machine Learning). Key objectives include:

1. Designing and implementing machine learning models that can classify and predict the student’s most suitable job profile.
2. Optimizing the models through hyperparameter tuning to enhance prediction accuracy.
3. Developing a user-friendly web application using Streamlit that allows students to input their academic and skill data and receive job profile predictions in real-time.
4. Ensuring that the model provides insights based on a combination of both academic and technical skills, making the predictions relevant to the student’s overall competence.

Ultimately, this project seeks to create a tool that empowers students by offering data-driven career guidance, helping them make informed decisions about their future roles in the industry.

1. **INTRODUCTION**

In the ever-evolving world of technology and business, career choices for students have become increasingly diversified. The rapid growth in fields such as Data Science, Software Engineering, Artificial Intelligence, and Cybersecurity has made it crucial for students to align their academic performance and skill development with specific job roles. In particular, engineering students in India face immense pressure to secure placements in a competitive job market, where multiple factors—including academic performance, extracurricular involvement, and technical skills—play a pivotal role in determining their career trajectory.

This project aims to provide an automated system to predict a student’s ideal job profile based on a combination of academic performance in key subjects and proficiency in specific technical skills. By analysing students' scores in subjects like Data Structures and Algorithms (DSA), Database Management Systems (DBMS), Operating Systems (OS), and Computer Networks (CN), along with their expertise in programming languages and tools such as Python, Java, and Machine Learning, the system can recommend suitable job profiles, such as Software Developer, Data Scientist, or Systems Engineer. This predictive system offers a data-driven approach to assist students in aligning their capabilities with relevant career paths.

* 1. **Role of Skills and Academic Subjects in Job Placement**

In the modern job market, recruiters place substantial emphasis on both **academic performance** and **technical skills** when evaluating candidates for various roles. Subjects like DSA, DBMS, OS, and CN are considered core to computer science and engineering curricula, and strong performance in these areas indicates a solid foundation in problem-solving, system design, and software development.

* **Data Structures and Algorithms (DSA):** This subject is crucial for almost all technical roles, particularly software development and system design. It tests a student's ability to optimize solutions, manage resources effectively, and solve complex problems under time constraints. Companies like Google, Amazon, and Microsoft place significant weight on DSA skills during technical interviews.
* **Database Management Systems (DBMS):** DBMS is important for roles such as Database Administrator, Data Engineer, and Backend Developer. It covers the organization, storage, and retrieval of data, which is fundamental to building scalable applications and managing large datasets in industries like banking, e-commerce, and healthcare.
* **Operating Systems (OS):** A strong understanding of OS concepts is crucial for roles like Systems Engineer, Cloud Architect, and DevOps Engineer. This subject covers the management of hardware and software resources, system security, and multitasking, which are vital in environments involving large-scale infrastructures or cloud platforms.
* **Computer Networks (CN):** CN is fundamental for careers in network engineering, cybersecurity, and telecommunications. It equips students with the knowledge to design, implement, and maintain secure communication networks, making it a high-demand skill in today's digital world.

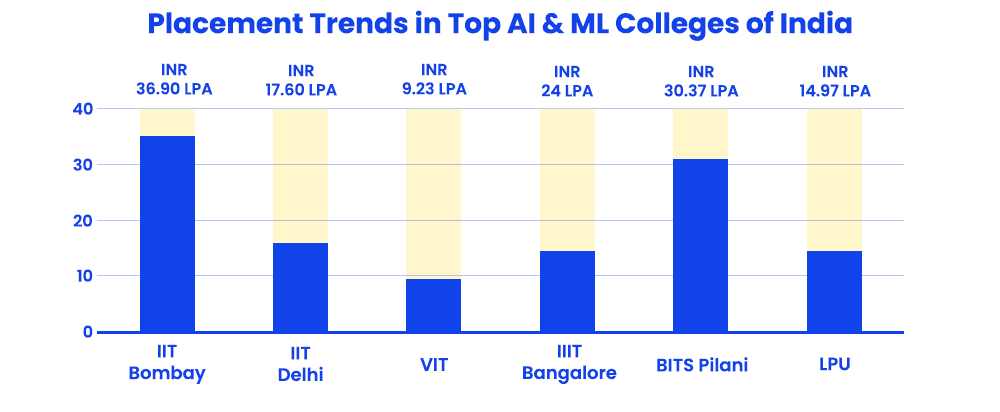
Beyond academic subjects, **technical skills** like proficiency in programming languages and tools—such as Python, Java, and Machine Learning—are critical in shaping career prospects. Skills in **Python** and **Machine Learning** open doors to high-growth areas like Data Science and AI, while expertise in **Java** is still highly valued in enterprise application development. Other key skills such as proficiency in **problem-solving**, **creativity**, and participation in **hackathons** can significantly boost employability by showcasing a student’s ability to innovate and work under pressure.

* 1. **Job Profiles in the Technology Sector**

The technology sector offers a wide array of job profiles, each requiring different combinations of academic knowledge and skills. Some of the most sought-after roles include:

* **Software Developer:** This role involves designing, building, and maintaining software applications. It requires strong programming skills, knowledge of algorithms, and the ability to work with multiple technologies and frameworks. Developers can specialize in areas like web development, mobile app development, or enterprise software solutions.
* **Data Scientist/Analyst:** Data Scientists analyse and interpret complex datasets to help businesses make data-driven decisions. This role demands skills in statistics, data mining, and machine learning, along with proficiency in programming languages such as Python and R.
* **Systems Engineer/Administrator:** Systems Engineers work on the design and management of complex systems in both hardware and software environments. This role requires deep knowledge of operating systems, networking, and systems integration.
* **DevOps Engineer:** DevOps Engineers bridge the gap between development and operations, ensuring efficient deployment and integration of systems. They are skilled in automation tools, cloud services, and continuous integration/continuous deployment (CI/CD) pipelines.
* **Cybersecurity Analyst:** Cybersecurity professionals ensure the protection of data and systems from digital attacks. They need expertise in encryption, network security, and vulnerability assessment.
  1. **Placement Trends in Indian Universities**

Placements in Indian engineering colleges, especially for computer science and IT students, have become a key indicator of success. According to placement reports from top institutes like the Indian Institutes of Technology (IITs) and National Institutes of Technology (NITs), the demand for roles in **Software Development**, **Data Science**, and **AI** has surged in recent years. For example, IIT Bombay’s 2023 placement report shows that **50%** of students were recruited into software development roles, while **20%** entered data science positions. The average package for software developers ranged from ₹10 to ₹15 lakhs per annum, with top companies offering even higher packages.



**Figure 1: Placement Trends in Top AI and ML Colleges of India**

Placement trends also show an increased demand for specialized roles in **AI**, **Cloud Computing**, and **Cybersecurity**, which are among the fastest-growing fields. With the rise of digital transformation initiatives across industries, there is a clear shift towards job profiles that require advanced technical skills in automation, big data, and cloud technologies. The emergence of start-ups and tech unicorns in India has further broadened the range of opportunities available to students, especially those with niche skills like machine learning, data engineering, and full-stack development.

Interestingly, recruiters are not solely focused on technical proficiency. Increasingly, employers are looking for **well-rounded candidates** who demonstrate creativity, leadership, and adaptability. Participation in **extracurricular activities**, such as hackathons, coding competitions, and internships, is often seen as a differentiator in hiring decisions. This aligns with the observation that students with high problem-solving skills and experience in **real-world projects** are more likely to secure top-tier job offers, as they bring practical knowledge to the table.

* 1. **Need for Data-Driven Job Profile Prediction**

Given the variability in student performance, skills, and career preferences, there is a need for a systematic and data-driven approach to help students identify the most suitable job profiles. Current methods of job placement guidance often rely on manual career counselling, which can be subjective and limited in scope. By building a machine learning model that considers both academic performance and technical skills, this project aims to automate the process of predicting the most appropriate career paths for students.

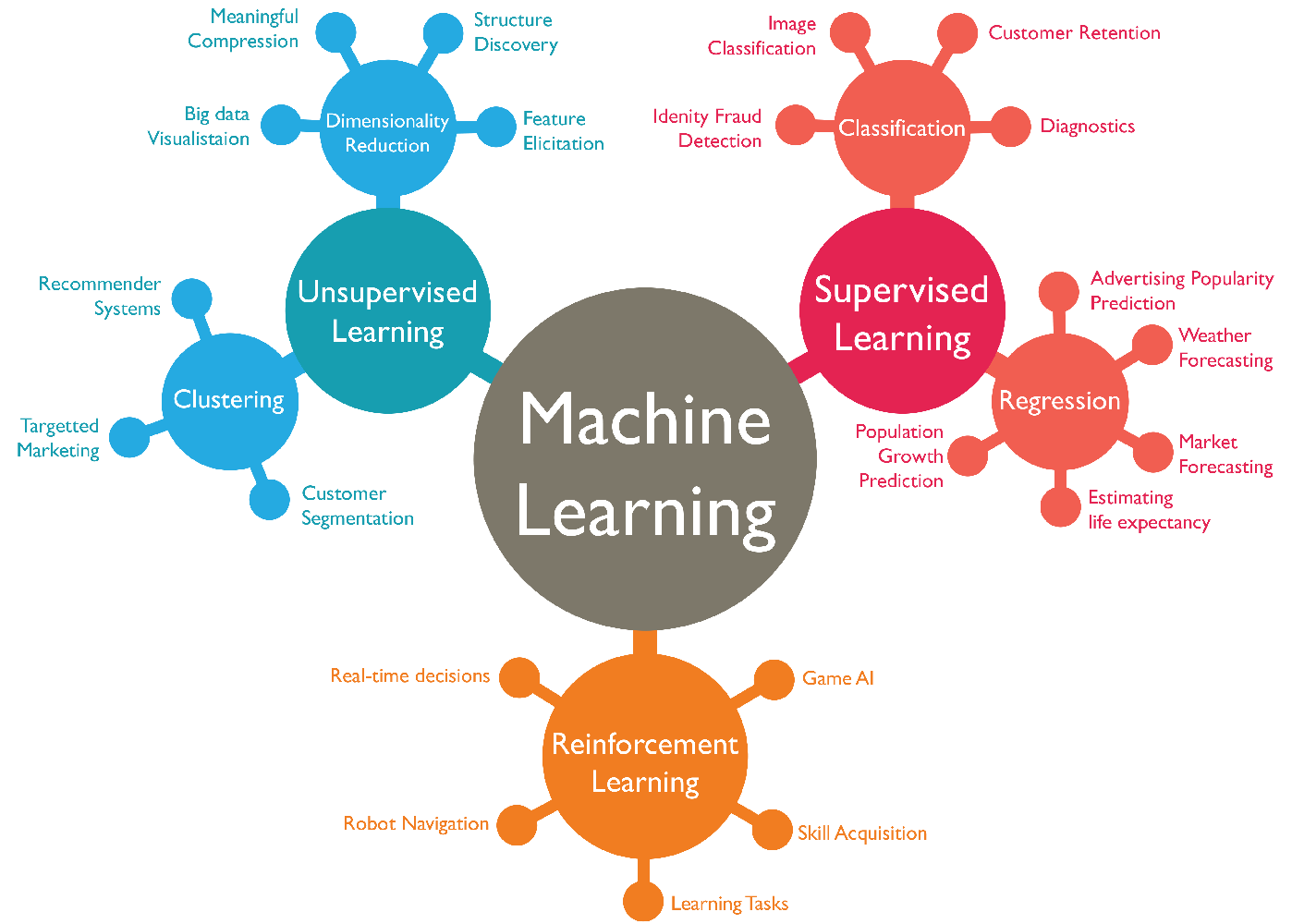
Through the analysis of academic scores, participation in technical activities, and skills in demand, this project hopes to contribute towards a more **personalized and accurate career guidance system**. Such a system can benefit students across universities by providing them with data-driven insights into their career possibilities, thereby helping them make more informed decisions about their future.

1. **THEORETICAL BACKGROUND**

The field of machine learning has revolutionized the way data is analysed, allowing computers to learn from data and make decisions without being explicitly programmed. In this project, we use various machine learning techniques to predict job profiles for students based on their academic performance and skills. To understand the foundation of this project, we need to delve into several key areas, including the types of machine learning, classification models, data preprocessing techniques, and model evaluation methods.

* 1. **Machine Learning Overview**

Machine learning is a subset of artificial intelligence that involves the development of algorithms that can learn from data and improve their performance over time. There are three primary types of machine learning:

* **Supervised Learning**: In supervised learning, the model is trained on a labelled dataset, meaning that the input features (such as academic scores and skills) are paired with the correct output labels (job profiles). The objective of the model is to learn the mapping from inputs to outputs and make accurate predictions for unseen data. This project is a classic example of supervised learning, as the dataset contains both input features and the corresponding job profiles.
* **Unsupervised Learning**: In unsupervised learning, the model is provided with input data but without any corresponding output labels. The goal is to identify patterns or structures within the data, such as clustering similar data points together. While unsupervised learning is not directly applicable to this project, it is commonly used in areas like data exploration and feature extraction.
* **Reinforcement Learning**: Reinforcement learning involves training models to make decisions in a sequential manner by receiving feedback in the form of rewards or penalties. It is mainly used in tasks such as robotics and game theory, and is not a focus of this project.

**Figure 2: Machine Learning Types**

In this project, we use **supervised learning** to predict a student’s job profile based on a combination of numerical and categorical features.

* 1. **Classification in Machine Learning**

The primary task in this project is **classification**, where the goal is to assign a class label (job profile) to a given set of input features (student scores and skills). The classification models are trained on a dataset containing labelled examples of students with known job profiles, and the models learn to predict the job profile for new, unseen students.

Several classification algorithms are used in this project, each with its strengths and limitations:

* **Decision Tree Classifier**: A decision tree is a flowchart-like model where each internal node represents a decision based on the value of a feature, and each leaf node represents a predicted class (job profile). Decision trees are easy to interpret and can handle both numerical and categorical data. However, they are prone to overfitting, especially when the tree becomes too deep.
* **Random Forest Classifier**: A random forest is an ensemble method that combines multiple decision trees to create a more robust and accurate model. Each tree is trained on a random subset of the data and the features, and the final prediction is made by aggregating the predictions of all the trees. Random forests reduce overfitting and improve generalization but can be computationally intensive.
* **Gradient Boosting Classifier (GBM)**: Gradient boosting is another ensemble technique that builds models sequentially, with each new model trying to correct the errors of the previous ones. This approach leads to high accuracy, but it can be slow to train and is sensitive to hyperparameters.
* **XGBoost**: XGBoost, or Extreme Gradient Boosting, is an advanced implementation of the gradient boosting algorithm. It incorporates techniques like regularization to prevent overfitting and parallel processing to speed up training. XGBoost is highly efficient and typically achieves better performance than other ensemble methods. It is the final model chosen for this project due to its accuracy and scalability.
  1. **Data Preprocessing Techniques**

Before training any machine learning model, it is essential to preprocess the data to ensure that it is in a suitable format for analysis. Data preprocessing is a crucial step because raw data often contains inconsistencies, missing values, or features that need to be transformed. The preprocessing steps in this project include:

* **Handling Missing Values**: Missing values can occur in any dataset due to incomplete records. In this project, the dataset is first checked for missing values, which, if present, would be handled by either imputing them or removing the incomplete rows, depending on the situation.
* **Feature Scaling**: Feature scaling is the process of normalizing or standardizing numerical features so that they fall within a similar range. In this project, student scores (e.g., in DSA, DBMS, etc.) are scaled using **StandardScaler** to ensure that no single feature dominates the others during model training. Scaling is particularly important when using algorithms like gradient boosting or XGBoost, which are sensitive to feature magnitude.
* **One-Hot Encoding**: The dataset includes categorical features, such as the technical skills (e.g., Python, Java, etc.). These categorical features must be converted into numerical form so that the machine learning models can process them. One-hot encoding is a method used to transform categorical variables into binary vectors. For example, the skill "Python" might be encoded as [0, 1], and "Java" as [1, 0].
* **Label Encoding**: The target variable in this project is the job profile, which is a categorical feature. To train the model, this feature is label encoded, meaning each job profile is assigned a unique numerical value. For example, "Data Scientist" could be encoded as 0, "Software Developer" as 1, and so on.
  1. **Model Evaluation Metrics**

Once the models are trained, it is important to evaluate their performance using appropriate metrics. The evaluation metrics used in this project include:

* **Accuracy**: Accuracy is the proportion of correct predictions made by the model. While it is a simple metric, it may not always be the best indicator of performance, especially when the classes are imbalanced.
* **Precision and Recall**: Precision is the proportion of true positive predictions among all positive predictions, while recall is the proportion of true positive predictions among all actual positive instances. Precision and recall are important when dealing with imbalanced datasets, as they provide insights into how well the model distinguishes between different job profiles.
* **F1 Score**: The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall, making it useful when there is a trade-off between the two.
* **Confusion Matrix**: A confusion matrix is a table that shows the number of true positive, true negative, false positive, and false negative predictions. It provides a detailed breakdown of the model’s performance across different classes and helps identify where the model is making mistakes.
* **ROC Curve and AUC**: The Receiver Operating Characteristic (ROC) curve plots the true positive rate (recall) against the false positive rate. The Area Under the Curve (AUC) is a measure of the model’s ability to distinguish between different classes. A higher AUC indicates a better-performing model.
  1. **Streamlit for Deployment**

The deployment of the machine learning model in this project is done using **Streamlit**, an open-source framework for creating interactive web applications for machine learning and data science projects. Streamlit simplifies the process of building a user interface and allows users to interact with the model by inputting their academic scores and technical skills. The following components of the Streamlit application are essential:

* **User Input Forms**: The application includes input fields such as sliders and dropdowns, where users can enter their academic scores and select their technical skills.
* **Prediction Output**: Once the user inputs their data, the machine learning model predicts the job profile, which is displayed on the web interface in real-time.
* **Model Integration**: The trained machine learning model (XGBoost) is integrated into the Streamlit application using the joblib library, allowing the app to make predictions based on the user’s input.

The theoretical foundation of this project lies in the concepts of supervised learning, specifically classification, as well as data preprocessing techniques like feature scaling and encoding. The models chosen—Decision Trees, Random Forests, Gradient Boosting, and XGBoost—are powerful classification tools that have been fine-tuned to achieve accurate job profile predictions. The use of Streamlit for deployment further enhances the project by providing a user-friendly interface for real-time interaction with the model.

By combining these advanced machine learning techniques with effective data preprocessing and model evaluation, this project aims to offer a reliable and data-driven solution for predicting job profiles based on student performance and skillsets.

1. **HARDWARE & SOFTWARE REQUIREMENTS**

In order to successfully implement the "Job Profile Prediction Based on Skills and Academic Performance" project, the system requires both hardware and software configurations that support machine learning model development, data processing, and web application deployment. Below are the detailed requirements:

* 1. **Hardware Requirements**

To ensure efficient execution of the machine learning algorithms and a smooth experience with the data-intensive tasks, the following hardware configuration is recommended:

1. **Processor:**
   * **Intel Core i5 (or higher)** or **AMD Ryzen 5 (or higher)**.
   * A multi-core processor with a clock speed of 2.5 GHz or above is essential for faster data processing and model training.
2. **RAM:**
   * **4 GB RAM (minimum)**, with **8 GB RAM (recommended)**.
   * Machine learning operations, such as training models on large datasets, and running web applications like Streamlit, require significant memory resources to avoid performance bottlenecks.
3. **Storage:**
   * **256 GB SSD (recommended)** or **1 TB HDD**.
   * Solid-state drives (SSD) are preferred for faster read/write speeds, especially during data processing and saving/loading machine learning models.
4. **Graphics Processing Unit (GPU) (Optional):**
   * For handling large datasets and more complex machine learning models like deep learning, a **dedicated GPU** such as **NVIDIA GTX 1060 or higher** may be beneficial. However, for this specific project, the GPU is optional since the models used (Random Forest, Decision Tree, etc.) do not necessarily require GPU acceleration.
5. **Operating System:**
   * **Windows 10/11**, **macOS** (version 10.13 or later), or **Linux (Ubuntu 20.04 LTS or later)**.
   1. **Software Requirements**

The software components required for this project include tools for data preprocessing, machine learning model development, and deployment. Here’s a breakdown of the necessary software:

1. **Programming Language:**
   * **Python 3.7 or later**.
   * Python is the core programming language used for data manipulation, machine learning model development, and web application deployment in this project. Libraries like pandas, scikit-learn, XGBoost, and Streamlit are utilized for various tasks.
2. **Libraries and Frameworks:**
   * **Pandas:**
     + For data manipulation, cleaning, and analysis of the student dataset.
   * **NumPy:**
     + Used for numerical computations, handling arrays, and performing operations on datasets.
   * **Scikit-learn:**
     + Provides implementations of machine learning algorithms such as Decision Trees, Random Forests, and data preprocessing techniques like scaling and encoding.
   * **XGBoost:**
     + For implementing the XGBoost algorithm, which is known for high accuracy and efficiency in classification tasks.
   * **Matplotlib/Seaborn:**
     + Used for data visualization, such as plotting distribution graphs, confusion matrices, and learning curves.
   * **Joblib:**
     + To save and load trained machine learning models, making it easier to use the model in production.
   * **Streamlit:**
     + An open-source framework used to develop the web interface for user interaction. It allows users to input their academic scores and skills and receive job profile predictions in real-time.
3. **Integrated Development Environment (IDE):**
   * **Jupyter Notebook:**
     + For developing and experimenting with the machine learning models, performing data preprocessing, and visualizing results.
   * **VS Code / PyCharm / Spyder:**
     + For writing and debugging Python code, especially during the development of the app.py for the Streamlit web application.
4. **Version Control:**
   * **Git / GitHub:**
     + For tracking code changes and version control. GitHub is also used for collaboration and code repository hosting, allowing easier deployment and sharing of code with other contributors.
5. **Database Management (Optional):**
   * **SQLite / MySQL:**
     + If storing user data or making the project more dynamic, a database such as SQLite or MySQL can be used for storing past predictions or user information.
6. **Model Deployment:**
   * **Heroku / AWS / GCP (Optional):**
     + For deploying the Streamlit web application to a cloud platform. Heroku offers an easy-to-use platform for hosting Python web apps, while AWS and GCP provide scalable infrastructure for larger applications.
7. **Other Dependencies:**
   * **LabelEncoder/OneHotEncoder:**
     + Used to convert categorical data (such as skills) into numerical form for model training.
   * **StandardScaler:**
     + Required for normalizing numerical data to ensure that all features contribute equally during model training.
   1. **Optional Hardware/Software Add-ons:**
8. **GPU-Accelerated Development:**
   * If you are working with larger datasets or plan to extend the project with more computationally demanding models like deep learning, using a GPU for accelerated computations (e.g., via **CUDA** or **TensorFlow** on NVIDIA GPUs) can significantly reduce training time.
9. **Anaconda Distribution:**
   * Anaconda simplifies the management of Python packages and environments, especially when handling various dependencies for machine learning and data science projects.
10. **Docker:**
    * If deploying the project in a production environment, Docker can be used to containerize the application, ensuring consistent behaviour across different environments by packaging the app along with its dependencies.

**6. METHODOLOGY**

**6.1. Methodology (Part 1): Suggester Notebook**

**6.1.1. Importing Required Libraries**

import warnings

warnings.filterwarnings('ignore')

**Purpose:** Suppresses any warning messages that could clutter the output. Warnings are often related to deprecated features or non-critical issues that don't affect code execution.

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**Purpose:** Imports essential libraries for data handling and visualization:

* **os**: For interacting with the operating system (e.g., file paths).
* **numpy**: Handles numerical operations, especially arrays and matrices.
* **pandas**: Manages and manipulates structured data in DataFrames.
* **matplotlib and seaborn**: For data visualization (e.g., plots, histograms, and heatmaps).

**6.1.2. Machine Learning Libraries**

from sklearn.model\_selection import train\_test\_split, GridSearchCV, learning\_curve

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, label\_binarize

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_curve, auc

from xgboost import XGBClassifier # XGBoost

from sklearn.multiclass import OneVsRestClassifier

from lazypredict.Supervised import LazyClassifier

**Purpose:** Imports machine learning functions and metrics from scikit-learn and xgboost:

* train\_test\_split: For splitting the dataset into training and test sets.
* GridSearchCV: Used to perform hyperparameter tuning for ML models.
* learning\_curve: Helps visualize the learning progress of a model.
* LabelEncoder, OneHotEncoder, StandardScaler: Preprocessing tools for encoding categorical data and scaling numerical features.
* DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, XGBClassifier: Classifiers used to predict the target variable.
* accuracy\_score, confusion\_matrix, etc.: Metrics for evaluating model performance.
* OneVsRestClassifier: For handling multi-class classification using a strategy that treats the classification as multiple binary problems.

import joblib

**Purpose:** joblib is used to save and load trained models for future use without retraining them.

**6.1.3. Loading the Dataset**

# loading the dataset

data = pd.read\_csv('../Dataset/StudentPlacement.csv')

data.tail()

**Purpose:** Loads the student placement dataset from a CSV file using pandas. The tail() function shows the last five rows of the dataset for a quick look at its structure.

**6.1.4. Understanding the Dataset**

# shape of the dataset

data.shape

**Purpose:** Displays the number of rows and columns in the dataset.

# basic info of the dataset

data.info()

**Purpose:** Provides details like data types of columns, non-null values, and memory usage of the dataset.

# checking for any null values

data.isnull().sum()

**Purpose:** Checks each column for missing (null) values.

# checking for any duplicate values in the dataset

data[data.duplicated(data.columns[:-1])]

**Purpose:** Identifies any duplicate records in the dataset, excluding the target column ('Profile').

**6.1.5. Exploratory Data Analysis (EDA)**

# viewing the unique profiles that this dataset have

data["Profile"].unique()

**Purpose:** Lists all unique values in the target column ('Profile'), which represents different job profiles.

# Extracting the numerical and categorical features

numerical\_data = data.select\_dtypes(include=['number'])

categorical\_data = data.select\_dtypes(include=['object', 'category'])

**Purpose:** Separates numerical and categorical columns. numerical\_data includes features like scores, while categorical\_data includes features like 'Skill 1' and 'Skill 2'.

# no of unique values does each numberical feature haves

numerical\_data.nunique()

**Purpose:** Shows the number of unique values in each numerical column.

**6.1.6. Correlation Heatmap**

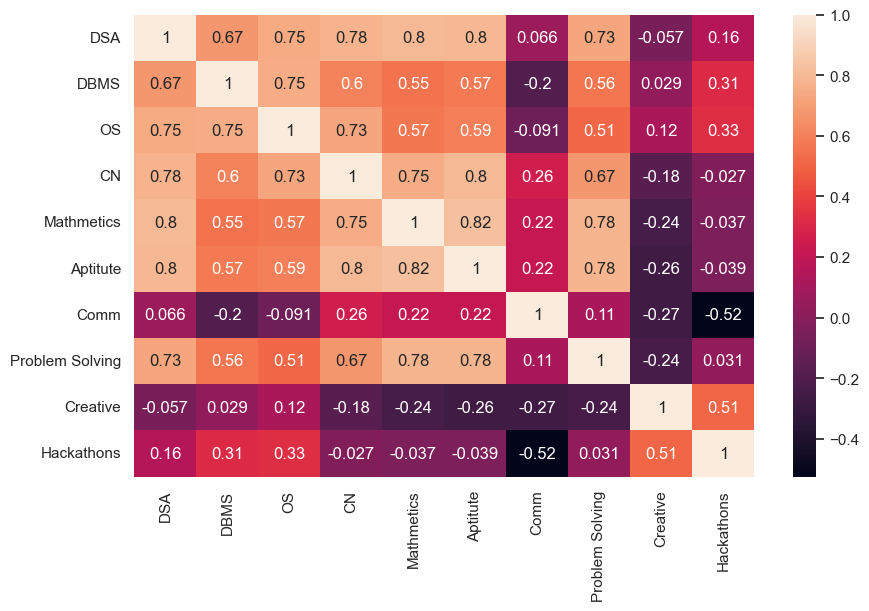
# Set the figure size

sns.set(rc={'figure.figsize':(10, 6)}) # width=10, height=6

sns.heatmap(numerical\_data.corr(), annot = True)

plt.show()

**Purpose:** Creates a correlation heatmap to visualize the relationships between numerical features (e.g., DSA, DBMS, etc.).

**Figure 3: Correlation Heatmap between the Columns**

**6.1.7. Distribution Plots**

# Set plot size

plt.figure(figsize=(16, 12))

# Loop through each numerical column and plot the distribution

for i, col in enumerate(numerical\_cols, 1):

plt.subplot(5, 2, i)

sns.histplot(data[col], kde=True, bins=20) # Histogram with kernel density estimate (KDE)

plt.title(f'Distribution of {col}')

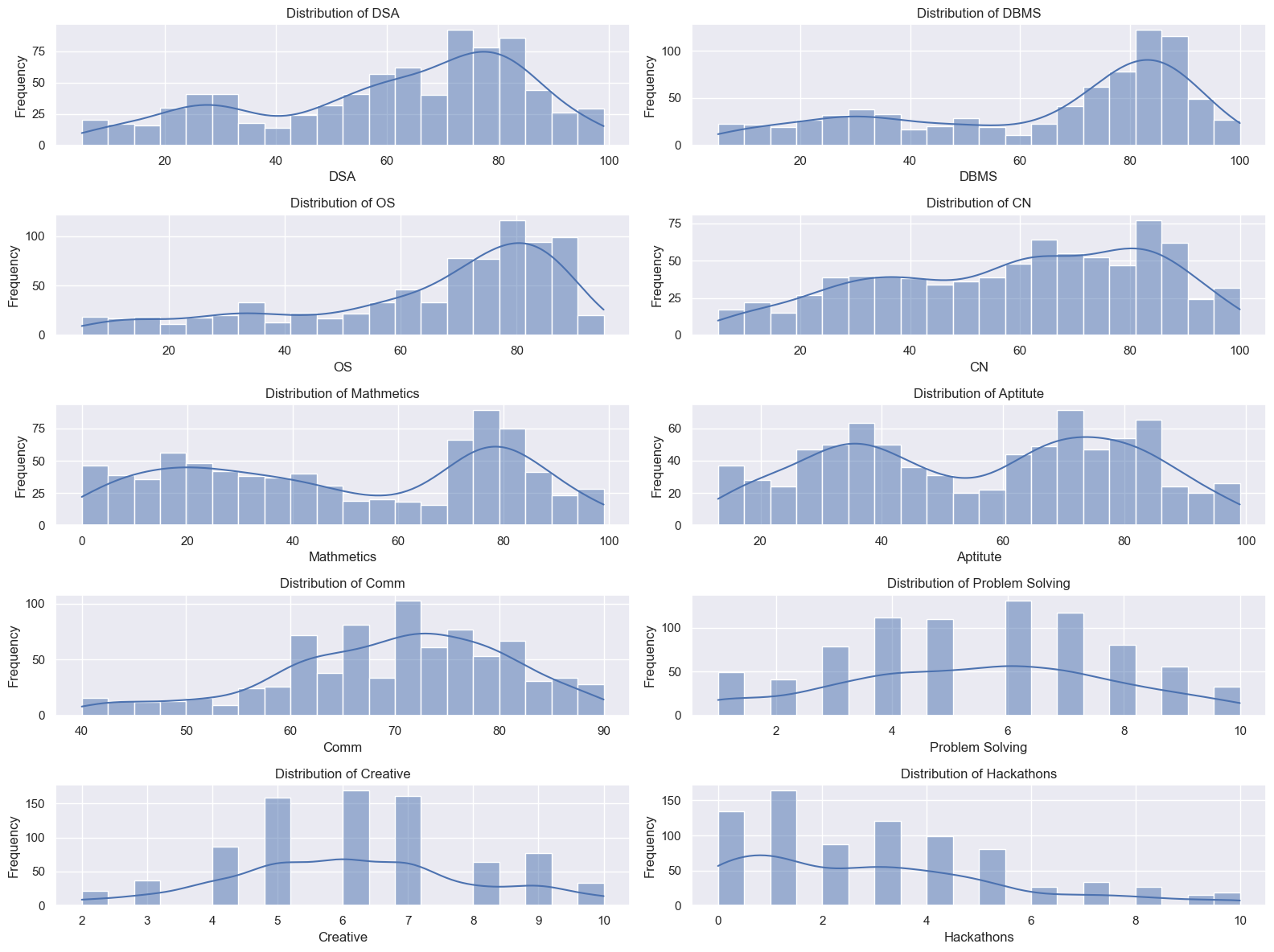
plt.xlabel(col)

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

**Purpose:** Plots histograms for each numerical column to analyze the distribution of values.



**Figure 4: Distribution plots of each column**

**6.1.8. Box Plots for Outlier Detection**

# Set plot size

plt.figure(figsize=(10, 10))

# Loop through each numerical column and plot a box plot

for i, col in enumerate(numerical\_cols, 1):

plt.subplot(3, 6, i)

sns.boxplot(y=data[col])

plt.title(f'Box Plot of {col}')

plt.ylabel(col)

plt.tight\_layout()

plt.show()

**Purpose:** Plots box plots for each numerical column to identify outliers in the dataset.

A group of blue boxes with white text

Description automatically generated

**Figure 5: Box plots for each numerical column**

**6.1.9. Outlier Removal Using Interquartile Range (IQR)**

# Function to remove outliers based on IQR

def remove\_outliers(df, columns):

for col in columns:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Remove outliers

df = df[(df[col] >= lower\_bound) & (df[col] <= upper\_bound)]

return df

# Remove outliers from the numerical columns

cleaned\_data = remove\_outliers(data.copy(), numerical\_cols)

**Purpose:** Removes outliers from the dataset by calculating the interquartile range (IQR) and filtering values that fall outside a reasonable range (1.5 \* IQR).

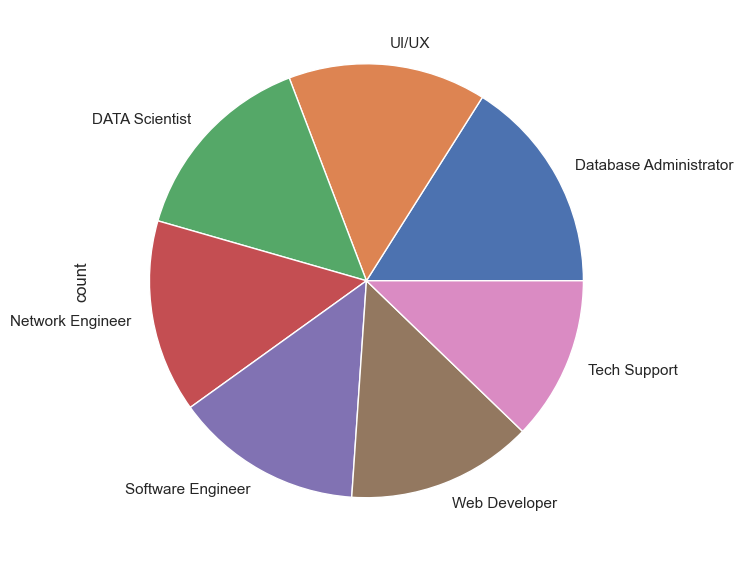
**6.1.10. Pie Chart of Profile Distribution**

# Set plot size

plt.figure(figsize=(7, 20))

cleaned\_data['Profile'].value\_counts().plot.pie()

plt.show()

**Purpose:** Plots a pie chart to show the distribution of the target variable (Profile).

**Figure 6: Pie Chart of target variable**

**6.1.11. One-Hot Encoding for Categorical Features**

# Step 1: One-hot encode 'Skill 1' and 'Skill 2'

skill\_encoder = OneHotEncoder( handle\_unknown='ignore')

# Step 2: Fit the encoder on 'Skill 1' and 'Skill 2' columns

skills\_encoded = skill\_encoder.fit\_transform(cleaned\_data[['Skill 1', 'Skill 2']]).toarray() \* 2.0

# Create a DataFrame from the dense array

skills\_encoded\_df = pd.DataFrame(skills\_encoded, columns=skill\_encoder.get\_feature\_names\_out(['Skill 1', 'Skill 2']))

**Purpose:** Encodes the categorical columns ('Skill 1' and 'Skill 2') using one-hot encoding and gives them more weight (multiplies by 2.0). Creates a DataFrame skills\_encoded\_df to store the encoded features.

**6.1.12. Combining Numerical and Encoded Data**

# Step 2: Combine original numerical features with encoded skill columns

X = pd.concat([numerical\_features.reset\_index(drop=True), skills\_encoded\_df.reset\_index(drop=True)], axis=1)

**Purpose:** Combines the numerical features and the one-hot encoded categorical features into a single feature matrix X.

**6.1.13. Feature Scaling**

# Step 3: Adjust feature importance by scaling and assigning higher weight to skills

scaler = StandardScaler()

X[X.columns[:10]] = scaler.fit\_transform(X[X.columns[:10]])

**Purpose:** Scales the first 10 numerical columns to ensure they are standardized (mean of 0 and standard deviation of 1), which is important for distance-based algorithms.

**6.1.14. Encoding the Target Variable**

profile\_encoder = LabelEncoder()

cleaned\_data['Profile'] = profile\_encoder.fit\_transform(cleaned\_data['Profile'])

# Target column (Profile)

y = cleaned\_data['Profile']

**Purpose:** Converts the categorical target column ('Profile') into numerical labels using LabelEncoder.

**6.1.15. Splitting the Dataset**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, shuffle=True, test\_size=0.3, stratify=y)

**Purpose:** Splits the dataset into training (70%) and test (30%) sets. The stratify parameter ensures that the proportion of different profiles in the training and test sets is the same.

**6.1.16. Model Optimization Function**

To enhance the performance of the machine learning models used in this project, a hyperparameter optimization strategy was employed. The models were tuned using a combination of decision trees, random forests, gradient boosting, and XGBoost classifiers. The following steps were undertaken:

def optimize\_tree\_and\_boosting\_models(X\_train, y\_train):

# Define hyperparameter space for each model

param\_spaces = {

'DT': {

'max\_depth': [5, 10, 15],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 3, 5],

},

'RF': {

'n\_estimators': [200, 250, 300],

'max\_depth': [10, 15, 20],

'min\_samples\_split': [1, 2, 5],

'min\_samples\_leaf': [1, 2, 5],

'bootstrap': [True, False],

},

'GB': { # Gradient Boosting

'n\_estimators': [10, 50, 100],

'learning\_rate': [0.01, 0.1, 0.5],

'max\_depth': [10, 15, 20],

'subsample': [0.1, 0.3, 0.8],

},

'XGB': { # XGBoost

'n\_estimators': [300, 400, 500],

'learning\_rate': [0.001, 0.01, 0.1],

'max\_depth': [3, 5, 10],

'subsample': [0.5, 0.8, 1.0],

'colsample\_bytree': [0.01, 0.1, 0.5],

}

}

# Initialize the models

models = {

'DT': DecisionTreeClassifier(random\_state=42),

'RF': RandomForestClassifier(random\_state=42),

'GB': GradientBoostingClassifier(random\_state=42),

'XGB': XGBClassifier(random\_state=42, eval\_metric='mlogloss'), # Removed use\_label\_encoder

}

best\_models = {}

# Loop through models and optimize each using GridSearchCV

for model\_name in models:

print(f"Optimizing {model\_name}...")

# Initialize GridSearchCV

opt = GridSearchCV(

models[model\_name],

param\_spaces[model\_name],

cv=5, # 5-fold cross-validation

n\_jobs=-1, # Use all available cores

verbose=0

)

# Fit the GridSearchCV on the training data

opt.fit(X\_train, y\_train)

# Store the best model found

best\_models[model\_name] = opt.best\_estimator\_

print(f"Best parameters for {model\_name}: {opt.best\_params\_}")

print(f"Best score for {model\_name}: {opt.best\_score\_}\n")

return best\_models

# Optimize and get the best models

best\_models = optimize\_tree\_and\_boosting\_models(X\_train, y\_train)

**6.1.16.1 Hyperparameter Tuning**

The **GridSearchCV** function was used to search across multiple hyperparameter configurations for the decision tree, random forest, and gradient boosting models. The hyperparameters were selected based on an initial understanding of the dataset and adjusted through trial and error. The grid search was performed with 3-fold cross-validation and the models were scored using accuracy metrics.

**Decision Tree** hyperparameters included:

* max\_depth: The maximum depth of the tree.
* min\_samples\_split: The minimum number of samples required to split an internal node.
* min\_samples\_leaf: The minimum number of samples that a leaf node must have.

**Random Forest** hyperparameters included:

* n\_estimators: The number of trees in the forest.
* max\_depth: The maximum depth of each tree.
* min\_samples\_split: The minimum number of samples required to split a node.
* min\_samples\_leaf: The minimum number of samples in a leaf node.
* bootstrap: Whether bootstrap samples are used when building trees.

**Gradient Boosting** hyperparameters included:

* n\_estimators: The number of boosting stages to be run.
* learning\_rate: Shrinks the contribution of each tree by this factor.
* max\_depth: Maximum depth of the individual estimators.
* subsample: Fraction of samples used for fitting the individual trees.

**6.1.16.2 Model Training and Optimization**

The models were trained on the preprocessed training data (X\_train, y\_train). For each model, the best set of hyperparameters was selected based on the highest accuracy achieved during the grid search cross-validation. The optimized models were then evaluated on the test set.

**6.1.17. Model Evaluation**

Once the models were trained and optimized, their performance was evaluated using the test data. Several evaluation metrics were considered to assess the classification performance of each model, including accuracy, precision, recall, F1-score, and AUC-ROC. These metrics provided insight into how well the models generalized to unseen data and handled imbalanced classes.

# Function to evaluate models and save them with their test accuracy

def evaluate\_models(best\_models, X\_test, y\_test):

for model\_name, model in best\_models.items():

y\_pred = model.predict(X\_test) # Predict on test data

accuracy = accuracy\_score(y\_test, y\_pred) # Calculate accuracy

# Convert accuracy to an integer percentage (e.g., 0.93 becomes 93)

accuracy\_percentage = int(accuracy \* 100)

print(f"{model\_name} Test Accuracy: {accuracy\_percentage}%")

# Evaluate models

evaluate\_models(best\_models, X\_test, y\_test)

**6.1.17.1 Accuracy**

The **accuracy** score represents the ratio of correctly predicted instances to the total instances. This is the most intuitive performance measure, but it can be misleading if the classes are imbalanced.

**6.1.17.2 Precision, Recall, F1-Score**

The **precision** score measures the ratio of correctly predicted positive observations to the total predicted positives. **Recall** (also known as sensitivity) measures the ability of the model to find all the positive instances. The **F1-Score** is the harmonic mean of precision and recall, providing a balance between the two.

# Model evaluations

def evaluate\_models\_on\_test(best\_models, X\_test, y\_test):

"""Evaluates each model in the best\_models dictionary on the test dataset and prints the test score."""

test\_scores = {}

# Loop through each model and compute its test score

for model\_name, model in best\_models.items():

# Get predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate the accuracy score (or you can use another metric like f1\_score, precision, etc.)

score = accuracy\_score(y\_test, y\_pred)

# Store the score in the dictionary

test\_scores[model\_name] = score

print(f"Classification Report of {model\_name}: \n\n{classification\_report(y\_test, y\_pred)}\n")

# Print the test score for the model

print(f"{model\_name} Test Accuracy: {score:.2f}\n\n")

return test\_scores

# testing the data on best trained models

test\_scores = evaluate\_models\_on\_test(best\_models, X\_test, y\_test)

**Output**:

Classification Report of DT:

precision recall f1-score support

0 0.89 0.94 0.92 35

1 0.91 0.84 0.88 38

2 0.89 0.94 0.91 34

3 0.94 0.91 0.92 33

4 0.97 0.97 0.97 29

5 0.97 1.00 0.99 35

6 0.97 0.94 0.95 32

accuracy 0.93 236

macro avg 0.93 0.93 0.93 236

weighted avg 0.93 0.93 0.93 236

DT Test Accuracy: 0.93

Classification Report of RF:

precision recall f1-score support

0 0.92 0.97 0.94 35

1 0.95 0.95 0.95 38

2 1.00 1.00 1.00 34

3 1.00 0.94 0.97 33

4 0.97 1.00 0.98 29

5 1.00 1.00 1.00 35

6 1.00 0.97 0.98 32

accuracy 0.97 236

macro avg 0.98 0.98 0.98 236

weighted avg 0.98 0.97 0.97 236

RF Test Accuracy: 0.97

Classification Report of GB:

precision recall f1-score support

0 0.92 0.97 0.94 35

1 0.92 0.95 0.94 38

2 1.00 0.94 0.97 34

3 1.00 0.94 0.97 33

4 0.94 1.00 0.97 29

5 1.00 1.00 1.00 35

6 1.00 0.97 0.98 32

accuracy 0.97 236

macro avg 0.97 0.97 0.97 236

weighted avg 0.97 0.97 0.97 236

GB Test Accuracy: 0.97

Classification Report of XGB:

precision recall f1-score support

0 1.00 0.94 0.97 35

1 0.90 0.97 0.94 38

2 0.91 0.94 0.93 34

3 1.00 0.94 0.97 33

4 1.00 1.00 1.00 29

5 0.97 0.97 0.97 35

6 0.97 0.97 0.97 32

accuracy 0.96 236

macro avg 0.97 0.96 0.96 236

weighted avg 0.96 0.96 0.96 236

XGB Test Accuracy: 0.96

**Table 1: Classification Reports of Multiple Models**

**6.1.17.3 Confusion Matrix**

A **confusion matrix** provides a summary of the model’s performance on the classification task by comparing actual and predicted class labels. It shows the true positives, true negatives, false positives, and false negatives.

"""Plots confusion matrices for all models in a 2x2 grid."""

def plot\_confusion\_matrices(best\_models, X\_test, y\_test):

num\_models = len(best\_models)

nrows = (num\_models + 1) // 2 # Ensures proper row count for even/odd models

ncols = 2 # 2 columns for the grid layout

fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(12, 6 \* nrows))

axes = axes.flatten() # Flatten in case there's only one row or column

for idx, (model\_name, model) in enumerate(best\_models.items()):

# Get predictions on the test set

y\_pred = model.predict(X\_test)

# Compute confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Plot the confusion matrix

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx])

axes[idx].set\_title(f'Confusion Matrix for {model\_name}')

axes[idx].set\_xlabel('Predicted')

axes[idx].set\_ylabel('Actual')

# If there are empty subplots (when number of models < total subplots), turn off those axes

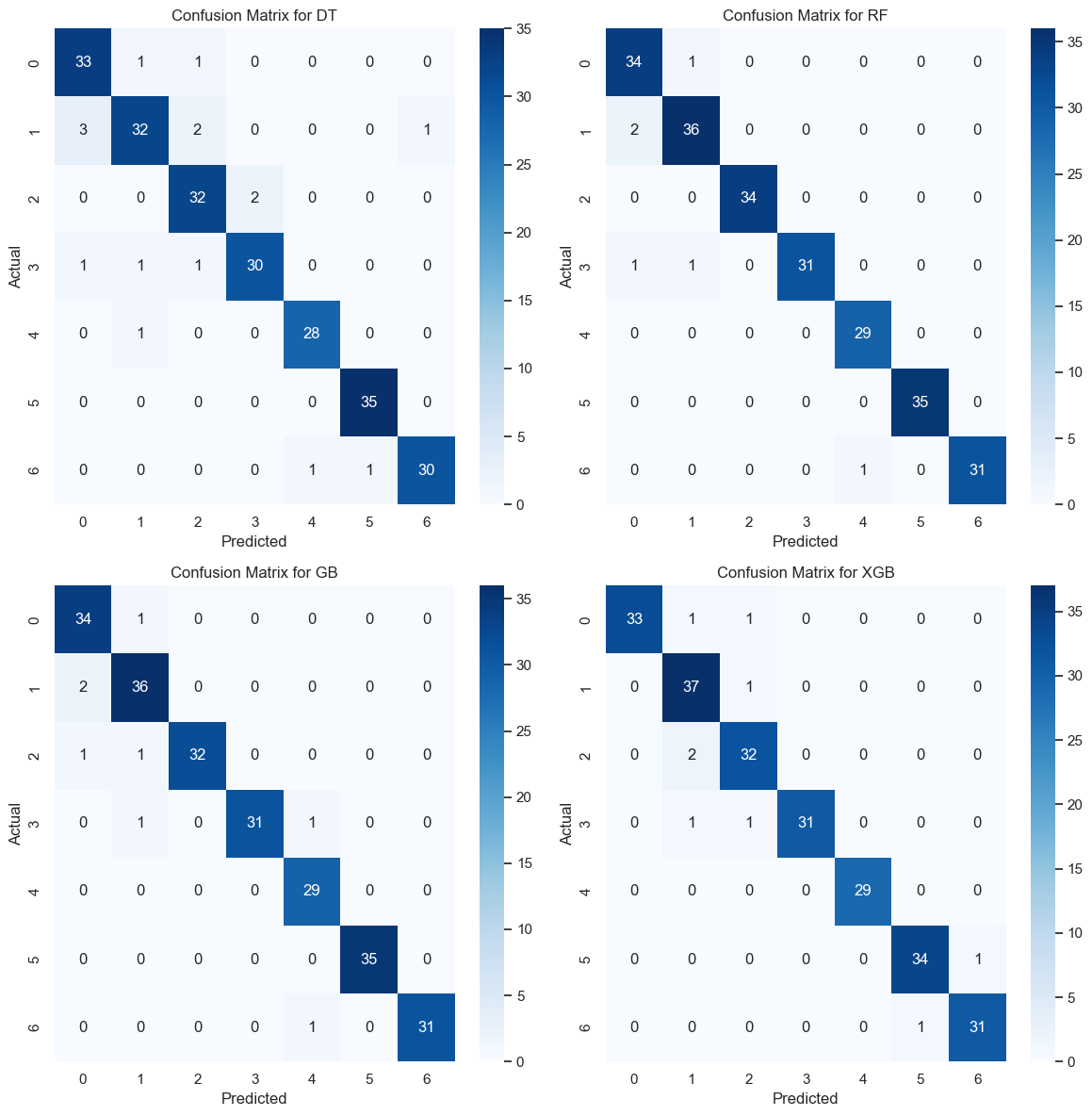
for i in range(len(best\_models), len(axes)):

axes[i].axis('off')

plt.tight\_layout()

plt.show()

# Call the function to plot confusion matrices

plot\_confusion\_matrices(best\_models, X\_test, y\_test)

**Figure 7: Confusion Matrix of multiple models**

**6.1.17.4 ROC**

The **Receiver Operating Characteristic (ROC)** curve plots the true positive rate (recall) against the false positive rate. The **Area Under the Curve (AUC)** score indicates the overall ability of the model to discriminate between classes. A higher AUC indicates better performance.

def plot\_multiclass\_roc\_curves(best\_models, X\_test, y\_test, n\_classes):

"""Plots ROC curves for multiclass classification using the One-vs-Rest strategy."""

# Binarize the labels for multiclass classification

y\_test\_binarized = label\_binarize(y\_test, classes=np.arange(n\_classes))

# Create a figure for the ROC curves

plt.figure(figsize=(12, 8))

for model\_name, model in best\_models.items():

if hasattr(model, 'predict\_proba'):

# For models that support predict\_proba

y\_score = model.predict\_proba(X\_test)

else:

# For models that support decision\_function

y\_score = model.decision\_function(X\_test)

# Compute ROC curve and AUC for each class

for i in range(n\_classes):

fpr, tpr, \_ = roc\_curve(y\_test\_binarized[:, i], y\_score[:, i])

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'{model\_name} (Class {i} AUC = {roc\_auc:.2f})')

# Plot the diagonal line (random chance)

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Multiclass ROC Curves for All Models')

plt.legend(loc='lower right')

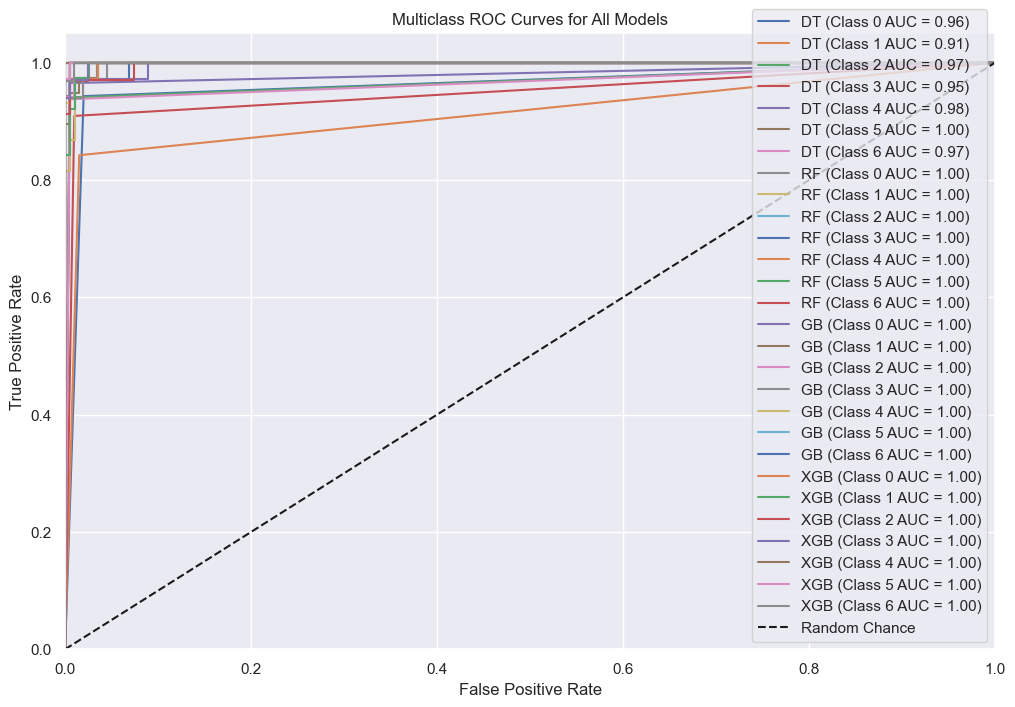
plt.show()

# Assuming n\_classes is the number of unique profiles in the target variable

n\_classes = len(np.unique(y\_test))

# Call the function to plot ROC curves

plot\_multiclass\_roc\_curves(best\_models, X\_test, y\_test, n\_classes)



**Figure 8: Multiclass ROC Curves of All Models**

**6.1.17.5 Learning Curve**

"""Plots learning curves for all models in a 2x2 grid."""

def plot\_learning\_curves(best\_models, X\_train, y\_train, cv=5, scoring='accuracy'):

num\_models = len(best\_models)

nrows = (num\_models + 1) // 2 # Dynamically calculate rows

ncols = 2 # Set columns to 2 for the grid layout

fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(12, 6 \* nrows))

axes = axes.flatten() # Flatten the grid to easily iterate over subplots

for idx, (model\_name, model) in enumerate(best\_models.items()):

# Generate learning curve data

train\_sizes, train\_scores, test\_scores = learning\_curve(

model, X\_train, y\_train, cv=cv, n\_jobs=-1, scoring=scoring, train\_sizes=np.linspace(0.1, 1.0, 10)

)

# Calculate mean and standard deviation for training and validation scores

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

# Plot learning curve

ax = axes[idx] # Select subplot

ax.set\_title(f"Learning Curve for {model\_name}")

ax.set\_xlabel("Training examples")

ax.set\_ylabel(scoring.capitalize())

# Plot the training score curve

ax.plot(train\_sizes, train\_scores\_mean, label="Training score", color="r")

ax.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, color="r", alpha=0.1)

# Plot the validation score curve

ax.plot(train\_sizes, test\_scores\_mean, label="Cross-validation score", color="g")

ax.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, color="g", alpha=0.1)

ax.legend(loc="best")

ax.grid()

# If there are fewer models than subplots, turn off extra axes

for i in range(len(best\_models), len(axes)):

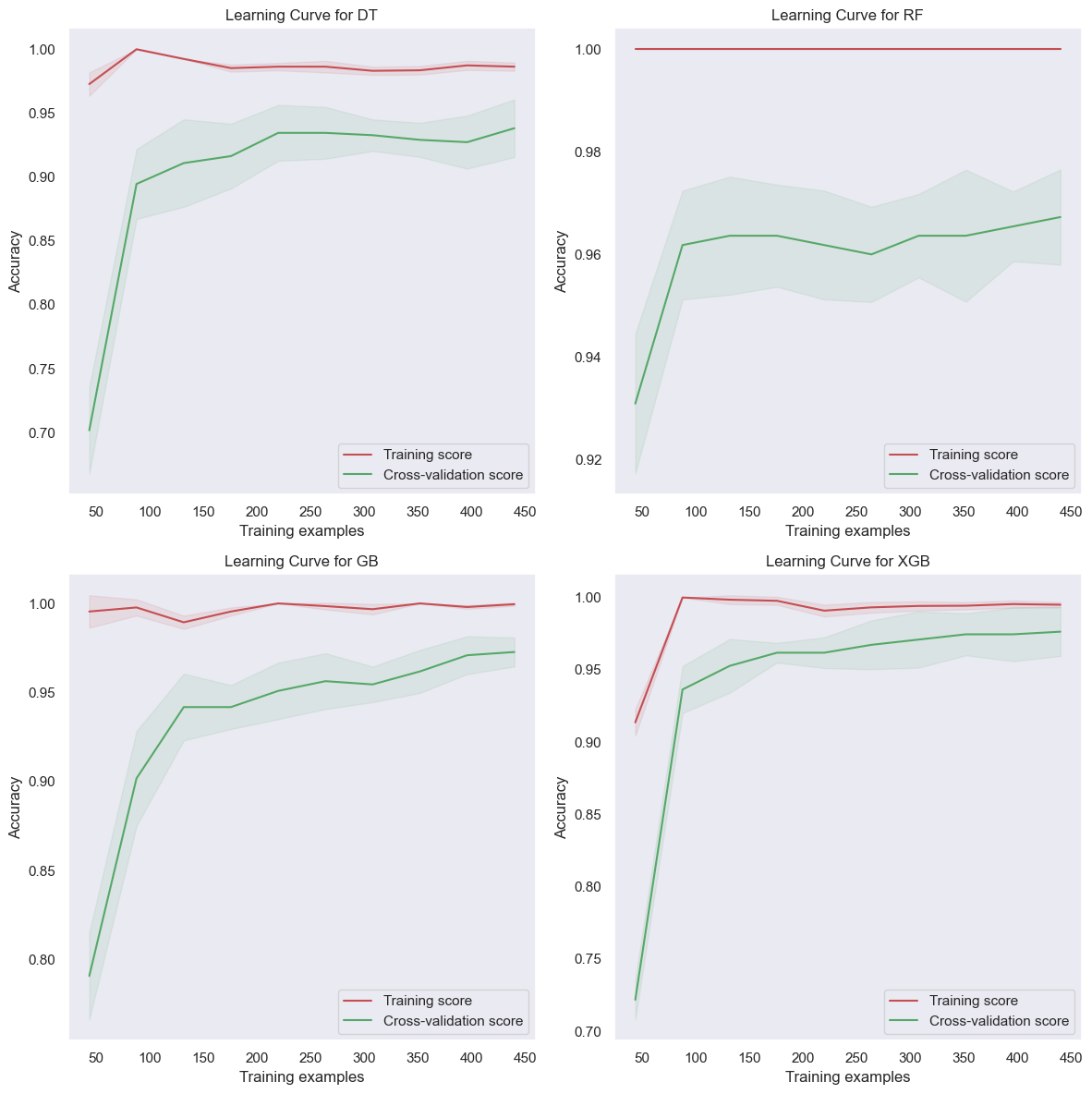
axes[i].axis('off')

plt.tight\_layout()

plt.show()

# Plot learning curves for all models

plot\_learning\_curves(best\_models, X\_train, y\_train)



**Figure 9: Learning Curve of Multiple Models**

**6.1.18. Model Saving**

To enable future use of the trained models without the need for retraining, the final models were serialized using **joblib**. This allowed for saving the models in binary format and loading them later for prediction or further analysis.

def save\_models(best\_models, scaler, skill\_encoder, profile\_encoder):

# Ensure the save directory exists

save\_dir = "../Models/"

os.makedirs(save\_dir, exist\_ok=True)

# Save the model with the formatted accuracy in the filename

model\_filename = os.path.join(save\_dir, "best\_modelsV2.joblib")

joblib.dump((best\_models, scaler, skill\_encoder, profile\_encoder ), model\_filename)

save\_models(best\_models, scaler, skill\_encoder, profile\_encoder)

**6.1.18.1. Purpose of Model Saving**

By saving the models, we ensure that we can deploy them for real-time prediction tasks or integrate them into a web or mobile application without requiring the entire model training process each time. This improves scalability and efficiency in production environments.

**6.1.19. Conclusion**

The **model optimization** process played a crucial role in improving the performance of the machine learning models. By carefully selecting hyperparameters and using cross-validation, the models were able to achieve a higher degree of accuracy, generalizing well to unseen data. The use of multiple classification algorithms allowed for a comparative analysis, with each model exhibiting different strengths depending on the feature set and target variable.

**6.2. Methodology (Part 2): Streamlit Application Development**

This section outlines the methodology for creating an interactive web application using Streamlit to make the model accessible and user-friendly for predicting job profiles based on users' skills and academic scores. The Streamlit app, app.py, consists of three main components: model initialization, user input acquisition, and prediction.

**6.2.1. Model Initialization**

The code starts by importing necessary libraries and initializing model components.

import streamlit as st

import numpy as np

import joblib # Using joblib for loading the saved model

streamlit is imported to build the web interface, numpy for data manipulation, and joblib for loading the saved model.

# Assuming the model file is in the same directory as your script (app.py)

model\_path = "Models/best\_modelsV2.joblib"

# Initialize model components

best\_models, scaler, skill\_encoder, profile\_encoder = None, None, None, None

The path to the model (best\_modelsV2.joblib) is specified.

Variables best\_models, scaler, skill\_encoder, and profile\_encoder are set to None initially; these will later hold the loaded model, scaler, skill encoder, and profile encoder, respectively.

# Try to load the model and handle errors

try:

best\_models, scaler, skill\_encoder, profile\_encoder = joblib.load(model\_path)

except FileNotFoundError:

st.error("Error: Model V2 file not found!")

st.stop() # Stop further execution if the model is not loaded

The model and its components are loaded from the specified file path using joblib.load. If the file is not found, a descriptive error is displayed, and the app stops execution to avoid further errors.

**6.2.2. Collecting User Input**

The get\_user\_input function is defined to gather user information on skills and scores, which are then preprocessed before prediction.

# Function to get user input

def get\_user\_input():

skillsList = ('Angular', 'Ansible', 'BASH/SHELL', 'C/C++', 'Cisco Packet Tracer', 'Deep Learning', 'Figma', 'GitHub', 'HTML/CSS', 'Java', 'JavaScript', 'Linux', 'Machine Learning', 'MySQL','Node.js', 'Oracle', 'Photoshop', 'PyTorch', 'Python', 'R', 'React', 'TensorFlow', 'Wireshark')

# Drop-down for skill selections

skill\_1 = st.selectbox('Skill 1', skillsList, index=21)

skill\_2 = st.selectbox('Skill 2', skillsList, index=19)

# Collecting user inputs for each feature using sliders

dsa = st.slider('DSA score (0-100)', min\_value=0, max\_value=100, value=72)

dbms = st.slider('DBMS score (0-100)', min\_value=0, max\_value=100, value=74)

os = st.slider('Operating Systems score (0-100)', min\_value=0, max\_value=100, value=73)

cn = st.slider('Computer Networks score (0-100)', min\_value=0, max\_value=100, value=60)

mathematics = st.slider('Mathematics score (0-100)', min\_value=0, max\_value=100, value=87)

aptitude = st.slider('Aptitude score (0-100)', min\_value=0, max\_value=100, value=82)

communication = st.slider('Communication score (0-100)', min\_value=0, max\_value=100, value=64)

problem\_solving = st.slider('Problem Solving score (0-10)', min\_value=0, max\_value=10, value=7)

creativity = st.slider('Creativity score (0-10)', min\_value=0, max\_value=10, value=6)

hackathons = st.slider('Number of Hackathons', min\_value=0, max\_value=10, value=4)

# Ensure input is transformed appropriately for skills

user\_skills = skill\_encoder.transform([[skill\_1, skill\_2]]).toarray()

# Create a list of numerical features i.e. user\_input

user\_input = [dsa, dbms, os, cn, mathematics, aptitude, communication, problem\_solving, creativity, hackathons]

# Combine numerical features and encoded skills

numerical\_features = scaler.transform(np.array(user\_input).reshape(1, -1))

user\_input\_transformed = np.hstack((numerical\_features, user\_skills)) # Combine arrays

return user\_input\_transformed

* Two dropdowns allow users to select their top two skills from a predefined skillsList. st.selectbox displays the dropdowns and allows users to pick a skill.
* The user provides scores for various subjects using st.slider, setting a default value within the range of possible values for each subject (e.g., 0–100 for DSA).
* The selected skills are transformed using skill\_encoder. This encoder maps categorical skills into numerical values, ensuring they can be processed by the model. toarray() is used to convert the output into a NumPy array.
* The scores and skills are combined into a single feature set:
  + user\_input: A list of numerical feature scores provided by the user.
  + scaler.transform(...): Standardizes the numerical input using a pre-fitted scaler.
  + np.hstack(...): Merges standardized scores with encoded skills for the final input format (user\_input\_transformed) required by the model.

**6.2.3. Building the Streamlit Application**

The main main() function defines the structure of the Streamlit application, including the title, instructions, and prediction logic.

# Main function to build the Streamlit app

def main():

# Project Title and Description

st.title("Job Profile Prediction")

st.write("### About This Project")

st.write(

"""

This project aims to predict suitable job profiles for students based on their scores in various subjects and skills.

The predictions are made using machine learning models.

This web application allows users to input their scores and receive job suggestions.

"""

)

st.write("### How to Use This Application")

st.write(

"""

1. Select your top two skills from the dropdown lists provided.

2. Use the sliders to enter your scores in various subjects including DSA, DBMS, Operating Systems, Computer Networks, Mathematics, Aptitude, Communication, Problem Solving, Creativity, and Hackathons.

3. Click the \*\*Predict Job Profile\*\* button to see your predicted job profile based on the entered information.

"""

)

st.warning("Note: The model doesn't solely depends on yours skills, but also on the scores you given. Model suggests according your scores and skills combined.")

# Get user input

user\_input\_original = get\_user\_input()

# When the user clicks the "Predict" button

if st.button('Predict Job Profile'):

# Make a prediction based on the user input

user\_input = user\_input\_original.copy()

# Predict using the model

predicted\_profile = best\_models['XGB'].predict(user\_input)

predicted\_job = profile\_encoder.inverse\_transform(predicted\_profile)

# Display the prediction result

st.success(f'Predicted Job Profile: {predicted\_job[0]}')

* The app title and description are displayed using st.title and st.write for introductory text.
* Additional usage instructions guide the user through interacting with the app, improving the user experience.
* The function get\_user\_input() is called, storing the processed user input in user\_input\_original.
* When the user clicks the Predict Job Profile button:
  + A copy of user\_input\_original is created for prediction.
  + The XGB model (part of best\_models) makes a prediction based on user\_input.
  + The encoded prediction is decoded using profile\_encoder.inverse\_transform, converting it into a readable job profile.
  + Finally, st.success displays the predicted job profile as a success message on the app.

**6.2.4. Running the App**

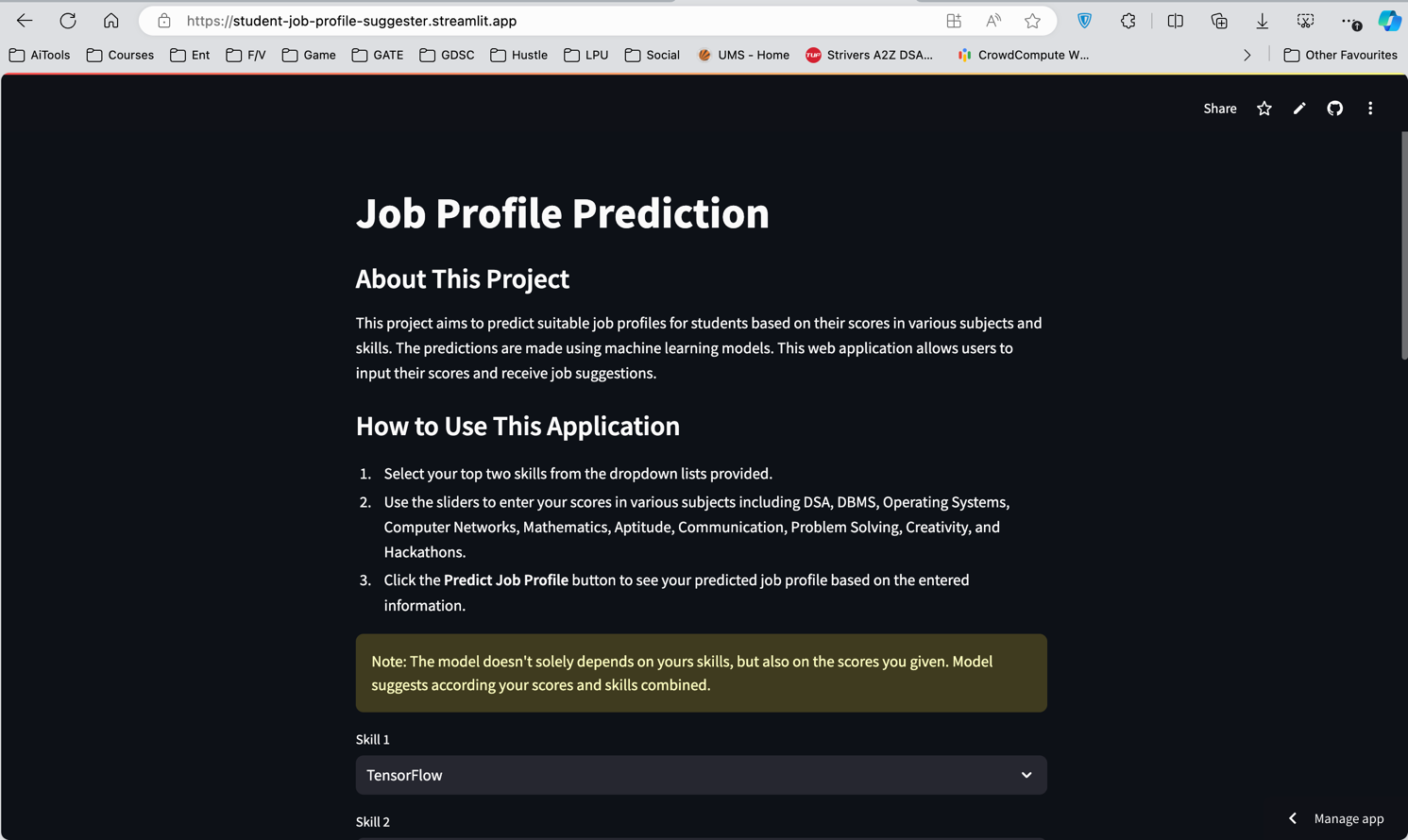
# Run the app

if \_\_name\_\_ == '\_\_main\_\_':

main()

* The app runs by calling the main() function when executed. This standard structure ensures that the app runs only when explicitly invoked, maintaining modularity and readability.

**Streamlit Application Link**: [student-job-profile-suggester.streamlit.app](https://student-job-profile-suggester.streamlit.app/)

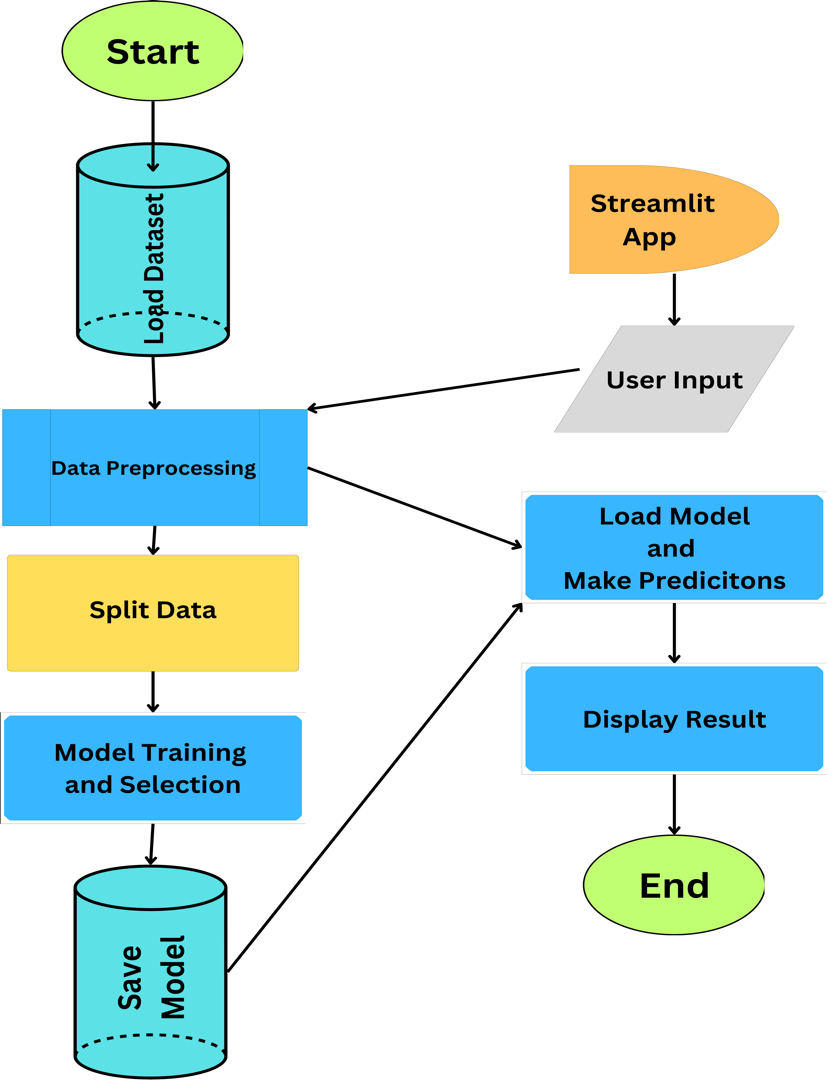


**Figure 10: Streamlit Web Application**

**7. FLOWCHARTS**

The flow below shows the detailed view of the project in simple way and one can understand about it.

1. **Start**  
   The process begins with loading the dataset and initializing necessary libraries and components.
2. **Load Dataset**  
   The dataset is loaded into a DataFrame, displaying information about students’ scores across various subjects, skills, and the job profiles for which they are suitable. This serves as the foundation for model training.
3. **Data Preprocessing**  
   The dataset undergoes essential preprocessing steps:
   * **Checking for Missing and Duplicate Values**: Ensuring data completeness and accuracy by handling any missing or duplicate values.
   * **Encoding Categorical Variables**: Converting categorical data such as Skill 1, Skill 2, and Profile to numerical form for compatibility with machine learning models.
   * **Scaling Numerical Features**: Standardizing numerical features, like subject scores, to ensure consistent scaling, aiding in model performance and stability.
4. **Split Data**  
   The preprocessed data is split into training and testing sets. The training set is used to build and train the model, while the testing set evaluates its performance.
5. **Model Training and Selection**
   * Several models are trained, including Decision Tree, Random Forest, and XGBoost classifiers.
   * Hyperparameter tuning is conducted with GridSearchCV to identify the best model configurations.
   * Models are evaluated based on metrics like accuracy, confusion matrix, and classification report.
   * The best-performing model is selected based on evaluation metrics and saved for use in the application.
6. **Save Model Components**  
   The best-trained model, along with the encoders for skills and the scaler for numerical data, are saved using joblib. This enables efficient loading and use within the Streamlit application.
7. **Streamlit App (User Interface)**  
   The Streamlit app is the user-facing interface. The application initializes by loading the saved model components (best model, scaler, and encoders) for immediate use.
8. **User Input Collection**  
   Users input their information:
   * They select their top two skills (Skill 1 and Skill 2) from dropdowns.
   * Scores in subjects and skills like DSA, DBMS, Communication, Creativity, and others are provided through sliders, allowing users to input data easily.
9. **Preprocess User Input**  
   User inputs are processed in real-time:
   * Skills are encoded using the saved skill encoder.
   * Numerical data is scaled with the saved scaler.
   * The transformed inputs are combined, creating a feature set ready for prediction.
10. **Make Prediction**  
    The app uses the selected best model to predict the most suitable job profile based on the user's input data. The model’s output is then decoded into a readable format.
11. **Display Result**  
    The app displays the predicted job profile to the user, providing insights into suitable career paths based on their academic and skill profile.
12. **End**  
    The process completes with the output display, ready for the next user input.



**Figure 11: Flow chart of complete Model training and Streamlit App**

1. **RESULTS**

The evaluation of the trained models was performed on the test dataset using a variety of performance metrics such as accuracy, precision, recall, F1-score, and the classification report for each model. The goal was to compare the performance of each model and determine which one provides the best predictions for student job profiles.

**1. Decision Tree Classifier (DT)**

The Decision Tree classifier achieved an accuracy of **93%** on the test set. Below is the detailed classification report for the model:

* **Precision**: Precision across all job profiles ranged between 0.89 and 0.97, with the highest precision for profile 5 (0.97).
* **Recall**: The recall score showed that the model was able to correctly identify most classes, with a range of 0.84 to 1.00.
* **F1-Score**: F1-scores were generally high, with the weighted average F1-score being **0.93**.

The **confusion matrix** indicated that most of the predictions were accurate, with minimal misclassifications across the profiles.

*DT Test Accuracy: 0.93*

**2. Random Forest Classifier (RF)**

The Random Forest classifier outperformed the Decision Tree classifier, achieving a higher test accuracy of **97%**. The detailed classification report is as follows:

* **Precision**: All profiles had precision scores above 0.92, with profiles 2, 3, 5, and 6 achieving perfect precision (1.00).
* **Recall**: The model demonstrated excellent recall for all profiles, with values ranging between 0.94 and 1.00.
* **F1-Score**: F1-scores were similarly impressive, with a weighted average F1-score of **0.97**.

The confusion matrix for Random Forest showed near-perfect classification results, with very few misclassifications.

*RF Test Accuracy: 0.97*

**3. Gradient Boosting Classifier (GB)**

The Gradient Boosting classifier also achieved a test accuracy of **97%**, matching the performance of the Random Forest model. The classification report shows strong results:

* **Precision**: Precision ranged from 0.92 to 1.00 across all job profiles, with a perfect precision score for profiles 2, 3, 5, and 6.
* **Recall**: The recall values were similarly high, with perfect recall for multiple profiles.
* **F1-Score**: The weighted average F1-score for Gradient Boosting was **0.97**.

The confusion matrix reflected minimal errors, and the model was highly accurate in its predictions across most profiles.

*GB Test Accuracy: 0.97*

**4. XGBoost Classifier (XGB)**

The XGBoost classifier achieved a slightly lower accuracy of **96%** compared to Random Forest and Gradient Boosting, but still demonstrated strong performance. Here are the detailed results:

* **Precision**: Precision scores ranged from 0.90 to 1.00, with profiles 0, 3, and 4 achieving perfect precision.
* **Recall**: The recall values showed that most of the job profiles were correctly classified, with profile 1 having a recall score of 0.97.
* **F1-Score**: The F1-scores were similarly strong, with a weighted average F1-score of **0.96**.

Although the model slightly underperformed compared to Random Forest and Gradient Boosting, it still demonstrated robust results, making it a viable model for classification.

*XGB Test Accuracy: 0.96*

**5. Summary of Model Performance**

The table below provides a summary of the test accuracy for each model:

|  |  |
| --- | --- |
| Model | Test Accuracy |
| *Decision Tree (DT)* | *0.93* |
| *Random Forest (RF)* | *0.97* |
| *Gradient Boosting (GB)* | *0.97* |
| *XGBoost (XGB)* | *0.96* |

**Table 2: Model Accuracies of Models**

The **Random Forest** and **Gradient Boosting** classifiers achieved the highest accuracy on the test dataset, both with **97% accuracy**, followed closely by the **XGBoost** classifier at **96%** and the **Decision Tree** classifier at **93%**. Given the minimal differences in accuracy, Random Forest and Gradient Boosting are considered the most suitable models for predicting student job profiles based on skills and academic performance.

**6. Interpretation of Results**

All models demonstrated strong performance on the test dataset, with Random Forest and Gradient Boosting leading the pack. These models excelled due to their ability to handle the complexity of the features and their robustness against overfitting, which was controlled via hyperparameter tuning.

Although the Decision Tree model provided decent performance, its accuracy was lower compared to the ensemble methods. On the other hand, XGBoost performed competitively but slightly underperformed relative to the Random Forest and Gradient Boosting models, likely due to the specific nature of the dataset and feature distribution.

The results suggest that ensemble methods like Random Forest and Gradient Boosting are well-suited for this problem due to their ability to capture feature interactions and complex patterns in the data.

1. **SUMMARY**

The primary objective of this project was to predict a student’s job profile based on their academic performance, skills, and participation in various activities. By analysing features such as subject scores, specific skills, and involvement in extracurricular activities, this model provides valuable insights into how different factors impact career paths in technical fields.

The dataset used for this project consisted of 800 entries with 18 features, including both numerical and categorical attributes. Preprocessing steps included handling categorical data through one-hot encoding, scaling numerical features, and assigning varying levels of importance to specific features. Model training was performed using Decision Trees, Random Forest, Gradient Boosting, and XGBoost classifiers. The performance of each model was evaluated based on metrics like accuracy, precision, recall, and F1-score.

The Random Forest and Gradient Boosting models emerged as the most accurate, each achieving a test accuracy of 97%, followed closely by XGBoost with 96%. These models demonstrated robustness and high performance, making them suitable for predicting job profiles based on the given dataset. Their effectiveness likely stems from their ability to handle feature interactions and reduce overfitting through ensemble learning techniques.

In conclusion, this project highlights the potential for machine learning models to predict job profiles with high accuracy based on academic performance and skills. These findings can help educational institutions and students alike to understand how different competencies and academic achievements correlate with career outcomes, thus aiding in more targeted skill development and career guidance. Further enhancements, such as expanding the dataset with additional features or exploring neural networks, could potentially improve model accuracy and broaden its applicability in different educational settings.

1. **CONCLUSION**

This project successfully demonstrates the application of machine learning techniques to predict students' job profiles based on academic scores, skills, and extracurricular activities. By utilizing data-driven models, we identified key features influencing career paths, providing insight into how specific skills and subject expertise contribute to a student's potential job role. The Random Forest and Gradient Boosting models achieved the highest performance, highlighting the strength of ensemble learning for this classification task.

The project underscores the value of advanced algorithms in predicting career outcomes, which can guide students in focusing on specific skills and subjects that align with their desired career trajectories. Institutions can leverage similar models to provide personalized career counselling, helping students make informed decisions on skills acquisition and academic pursuits.

Future work could focus on refining the model by incorporating a wider array of features, such as internship experiences or project work, to capture additional dimensions of a student’s profile. Overall, this project illustrates the meaningful role of predictive analytics in educational and career guidance, promoting data-driven insights for both students and academic institutions.

1. **BIBLIOGRAPHY**
2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.
3. Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.
4. Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5-32.
5. Quinlan, J. R. (1986). *Induction of Decision Trees*. Machine Learning, 1(1), 81–106.
6. Brownlee, J. (2020). *Machine Learning Mastery with Python*. Machine Learning Mastery.
7. He, H., & Garcia, E. A. (2009). *Learning from Imbalanced Data*. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284.
8. McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference, 56-61.
9. Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.
10. Zhao, D., & You, Q. (2018). *Interactive Data Visualization Using Python and Streamlit*. Towards Data Science.
11. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
12. **ANNEXURE**

**12.1. Figures**

|  |  |
| --- | --- |
| Figure 1: Placement Trends in Top AI and ML Colleges of India | Page: 6 |
| Figure 2: Machine Learning Types | Page: 8 |
| Figure 3: Correlation Heatmap between the Columns | Page: 16 |
| Figure 4: Distribution plots of each column | Page: 17 |
| Figure 5:Box plots for each numerical columns | Page: 18 |
| Figure 6: Pie chart of targe variable | Page: 19 |
| Figure 7: Confusion Matrix of Multiple models | Page: 25 |
| Figure 8: Multiclass ROC curves of All Models | Page: 27 |
| Figure 9: Learning Curve of Multiple Models | Page: 29 |
| Figure 10: Streamlit Web Application | Page: 34 |
| Figure 11: Flow chart of complete Model training and Streamlit App | Page: 36 |

**12.2. Tables**

|  |  |
| --- | --- |
| Table 1: Classification Reports of Multiple Models | Page: 24 |
| Table 2: Accuracies of Models | Page: 38 |