From Code to Career: Assessing Competitive Programmers for Industry Placement

by

Examination Roll:

Fathima Binthe Muhammed (201756)
Umit Saha (201782)
Md Imranur Rahman Akib (201785)

A Researech Project Report submitted to the
Institute of Information Technology
in partial fulfillment of the requirements for the degree of
Bachelor of Science in
Information and Communication Technology

Supervisor: Md Fazlul Karim Patwary, Professor
Special Mention: Md. Biplob Hosen, Assistant Professor

Institute of Information Technology
Jahangirnagar University
Savar, Dhaka-1342

February 2025

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

Fathima Binthe Muhammed	Umit Saha	Md Imranur Rahman Akib
Roll:201756	Roll:201782	Roll:201785

CERTIFICATE

This is to certify that the thesis entitled **From Code to Career: Assessing Competitive Programmers for Industry Placement** has been prepared and submitted by **Fathima Binthe Muhammed**, **Umit Saha** and **Md Imranur Rahman Akib** in partial fulfilment of the requirement for the degree of Bachelor of Science (honors) in Information Technology on February 24, 2025.

Md Fazlul Karim Patwary
Supervisor

Accepted and approved in partial fulfilment of the requirement for the degree Bachelor of Science (honors) in Information Technology.

Dr. Jesmin Akhter
Chairman

Prof. Dr. M. Shamim Kaiser
Member

---

prof. Md. Fazlul Karim Patwary

Member

Prof. Dr. Kazi Muheymin-Us-Sakib
External Member

ACKNOWLEDGEMENTS

We feel pleased to have the opportunity of expressing our heartfelt thanks and gratitude to those who all rendered their cooperation in making this report.

This thesis is performed under the supervision of Md Fazlul Karim Patwary, Professor, Institute of Information Technology (IIT), Jahangirnagar University, Savar, Dhaka. During the work, he has supplied us a number of books, journals, and materials related to the present investigation. Without his help, kind support and generous time spans he has given, we could not perform the project work successfully in due time. First and foremost, we wish to acknowledge our profound and sincere gratitude to him for his guidance, valuable suggestions, encouragement and cordial cooperation.

We express our utmost gratitude to Dr. Risala Tasin Khan, Director, IIT, Jahangirnagar University, Savar, Dhaka, for his valuable advice that have encouraged us to complete the work within the time frame. Moreover, we would also like to thank the other faculty members of IIT who have helped us directly or indirectly by providing their valuable support in completing this work.

We express our gratitude to all other sources from where we have found help. We are indebted to those who have helped us directly or indirectly in completing this work.

Last but not least, we would like to thank all the staff of IIT, Jahangirnagar University and our friends who have helped us by giving their encouragement and cooperation throughout the work.

ABSTRACT

In today's fast-paced tech industry, there is a growing need for tools that can evaluate a programmer's job readiness based on their coding performance. This project focuses on predicting a Codeforces user's potential to secure various levels of software engineering jobs. The primary objective is to analyze how a user's competitive programming activity reflects their chances of landing jobs ranging from entry-level roles to positions at major tech companies. To achieve this, we gathered user data through the Codeforces API, processed key performance metrics, and built a prediction model using a Random Forest Classifier. The model categorizes users into four levels of employability—from those needing further practice to those ready for top-tier tech jobs. The system was implemented using Flask and deployed on Render for real-time predictions. Our evaluation of the model's accuracy, precision, recall, and F1-score demonstrated that the approach effectively distinguishes between different skill levels based on coding proficiency and engagement. This work provides a foundation for using machine learning in career assessments and could be extended to predict job readiness across broader technical fields. Future enhancements may involve incorporating additional datasets, improving model accuracy, and expanding the system's applicability to international job markets.

LIST OF ABBREVIATIONS

API	Application Programming Interface
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
CSV	Comma-Separated Values
ML	Machine Learning
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
RF	Random Forest
AI	Artificial Intelligence
NLP	Natural Language Processing
UI	User Interface
AWS	Amazon Web Services
LSTMs	Long Short-Term Memory Networks
SMOTE	Synthetic Minority Over-sampling Technique

LIST OF NOTATIONS

α	Alpha (a constant or parameter)
max	Maximum value
$\cos\theta$	Cosine of angle θ
X'	Transformed feature value after normalization
X	Original feature value
$X_{\min}$	Minimum feature value in dataset
$X_{\max}$	Maximum feature value in dataset
μ	Mean of a dataset
σ	Standard deviation of a dataset
$\log(X+1)$	Log transformation applied to skewed data
$\{0, 1, 2, 3\}$	Encoded job status categories (No Job, Entry-Level Job, High-Paying Job, Tech Giants)

LIST OF FIGURES

Figure

LIST OF TABLES

Table

TABLE OF CONTENTS

CHAPTER I

Introduction

1.1 Background and Motivation

The demand for skilled software engineers has grown significantly in recent years, driven by rapid advancements in artificial intelligence, data science, and software development. To identify top talent, many tech companies rely on competitive programming platforms like Codeforces, LeetCode, and CodeChef to assess candidates' problem-solving abilities. Among these, Codeforces stands out as one of the most widely used platforms, attracting programmers from all over the world.

Competitive programming helps individuals develop essential skills such as logical thinking, algorithmic expertise, and coding efficiency—qualities that are highly valued in the software industry. While many successful programmers leverage these skills to secure jobs at top companies like Google, Microsoft, and Meta, others struggle to make a smooth transition into the job market. This raises an important question: *Can a machine learning model predict a programmer's career trajectory based on their competitive programming performance?*

This research seeks to explore that question by analyzing historical data from Codeforces users, identifying key factors that contribute to career success, and developing a predictive model that classifies programmers based on their career status. By applying machine learning techniques, this study aims to provide insights into the relationship between competitive programming performance and career growth, offering an AI-driven approach to career prediction.

1.2 Problem Statement

While competitive programming is widely acknowledged as a valuable tool for assessing technical skills, there is no standardized method to determine how a pro-

grammer's performance on platforms like Codeforces translates into career success. Many recruiters use competitive programming achievements as a screening criterion, yet there remains uncertainty about which specific metrics—such as problem-solving history, contest participation, or rating trends—are the most reliable indicators of employability. Likewise, job seekers often invest significant time and effort into competitive programming without a clear understanding of how it impacts their career prospects.

This research aims to bridge this gap by developing a machine learning-powered system capable of predicting a programmer's career trajectory based on their Codeforces activity. Specifically, the study seeks to determine how accurately a machine learning model can forecast career outcomes, identify the key competitive programming factors that influence employability, and evaluate whether an automated system can provide meaningful career insights based on competitive programming profiles. By leveraging historical data and advanced classification models, this research not only enhances our understanding of the relationship between competitive programming and career success but also provides a data-driven tool to assist programmers in making informed career choices while helping recruiters refine their hiring strategies.

1.3 Objectives of the Study

The main objectives of this study are:

- To collect and preprocess Codeforces user data, extracting meaningful features relevant to career prediction.
- To design and evaluate machine learning models that can predict a programmer's career trajectory based on their competitive programming history.
- To identify key performance indicators—such as problem-solving consistency, rating trends, and contest participation—that have the greatest impact on career outcomes.
- To develop a user-friendly web application that allows individuals to enter their Codeforces handle and receive instant career predictions.
- To deploy the system on a cloud platform, ensuring accessibility and scalability for users.

1.4 Scope

This study focuses on analyzing the competitive programming data of Codeforces users to predict career outcomes using machine learning techniques. It involves the application of classification models to assess how competitive programming performance correlates with career trajectories. Additionally, the study includes the development of a Flask-based web application that provides real-time career predictions, enabling users to receive instant insights based on their Codeforces activity. To ensure accessibility and scalability, the predictive model is deployed on Render.com, making it publicly available for users seeking career guidance based on their programming performance.

1.5 Contribution of the Project

This project offers several key contributions:

Development of a Career Prediction Model – A machine learning-based system capable of predicting a competitive programmer's career trajectory based on their Codeforces activity.

Feature Engineering for Career Prediction – Identification of crucial factors (such as rating growth, contest participation frequency, and problem-solving diversity) that influence career outcomes.

Deployment of a Real-Time Career Prediction System – A Flask-based web application that allows users to input their Codeforces handle and receive AI-generated career insights.

Practical Implications for Job Seekers and Recruiters – A data-driven tool that helps programmers assess their job market readiness and provides recruiters with a new metric for talent evaluation.

Open-Source Implementation – The project is available on GitHub, allowing developers and researchers to contribute improvements and expand its capabilities.

1.6 Report Organization

The remainder of this report is structured as follows. **Chapter 2 (Literature Review)** examines existing research on competitive programming, machine learning in career prediction, and hiring trends in the software industry. **Chapter 3 (Methodology)** details the processes involved in data collection, feature extraction, model development, and system deployment. **Chapter 4 (Results and Analysis)**

presents the performance of the predictive model, case studies, and insights based on real Codeforces users. Finally, **Chapter 5 (Conclusion and Future Work)** summarizes key findings, highlights limitations, and proposes potential improvements for future iterations of the project.

CHAPTER II

Literature Review

This chapter delves into prior research on predicting student performance, the role of academic competitions, and the application of machine learning techniques in career forecasting. By analyzing existing studies, this review identifies key insights and highlights gaps that our research, *From Code to Career: Assessing Competitive Programmers for Industry Placement*, aims to address. Additionally, it explores the challenges and opportunities associated with using competitive programming as a tool for career prediction.

2.1 Introduction to Job Market Trends for Software Engineers

The demand for skilled software engineers has grown significantly in recent years, with competitive programming emerging as a key measure of technical expertise.[13] Many companies now incorporate data-driven recruitment strategies, prioritizing a candidate's problem-solving abilities and coding efficiency over traditional academic credentials.[11]

Platforms such as Codeforces, LeetCode, and AtCoder have become widely recognized benchmarks for assessing coding proficiency. Studies suggest that participation in these contests correlates with better job placement rates, as it fosters algorithmic thinking, debugging skills, and the ability to write optimized code under time constraints.[2] Given this trend, there is a growing need for an automated system that can analyze a programmer's competitive programming history and predict career prospects based on performance metrics.[3]

2.2 Role of Competitive Programming in Job Placement

Competitive programming serves as more than just an extracurricular activity—it has become a recognized pathway to employment in top-tier technology firms like Google, Microsoft, and Meta. Many of these companies actively recruit candidates based on their competitive programming achievements and Codeforces rankings.[4]

Research has shown that frequent participation in contests strengthens logical reasoning, enhances problem-solving skills, and improves adaptability to real-world coding challenges.[9] Additionally, a programmer's contest performance and rating progression can serve as strong indicators of job success. University training programs that integrate competitive programming into their curriculum have reported higher employment rates among graduates, further emphasizing its importance as a career accelerator.[5] These findings support the idea that an AI-driven system could analyze competitive programming data and provide meaningful career predictions.[8]

2.3 Machine Learning in Career Prediction

Advancements in machine learning (ML) have unlocked new possibilities in talent evaluation, enabling AI-driven models to analyze competitive programming performance and forecast career outcomes.[12][10]

Several supervised learning models, including Random Forest, Support Vector Machines (SVM), and Deep Learning, have been applied to historical coding contest data with promising results.[15] These models effectively classify programmers based on contest participation, rating trends, and problem-solving consistency,[16] offering valuable insights into the likelihood of securing software engineering roles.[14]

Further developments in deep learning and feature engineering techniques have refined career prediction models, making AI-based assessments increasingly accurate and practical for real-world applications.[7][1]

2.4 Related Works in Predictive Modeling for Hiring

Numerous research studies have explored the use of predictive modeling to assess a candidate's job readiness, with a focus on competitive programming experience.[19] Machine learning techniques such as classification algorithms, feature extraction, and ensemble learning have been extensively applied to Codeforces data to evaluate a programmer's employability.[18][6]

Educational data mining (EDM) has also been used to study students' programming performance, revealing a strong connection between long-term engagement in competitive programming and career success.[17] . Table 2.1 summarizes key findings from recent studies on the role of competitive programming in career prediction.

Study	**Objective**	**Methodology**	**Key Findings**
Liu et al. (2021)	Impact of CP on career success	Codeforces performance analysis	Strong correlation with job success
Alnahhas et al. (2020)	Predictive hiring model	ML models (SVM, RF) on Codeforces data	ML predicts job placement effectively
Rahman et al. (2023)	AI-based career prediction	Deep learning on CP metrics	High accuracy in forecasting careers
Avlonitis et al. (2023)	Career planning with RL	Q-Learning, Sarsa	5% increase in career success
Li et al. (2022)	AI for CP performance analysis	AlphaCode, transformer model	Ranked in top 54.3% of Codeforces
Decorte et al. (2023)	NLP for career path prediction	CareerBERT on resume data	43.01% recall@10 achieved
Faruque et al. (2024)	AI-driven career guidance	NLP, ML on student data	Improved job matching outcomes
VidyaShreeram & Muthukumaravel (2021)	Student career prediction	Decision Tree, RF	RF achieved 93% accuracy

Table 2.1: Summary of key findings in career prediction research

2.5 Research Gap

Despite significant progress in predictive modeling for hiring, several challenges remain unresolved:

Limited Real-World Career Data – Many studies focus exclusively on contest performance without incorporating actual job placement data, making it difficult to accurately assess employability.

Lack of Personalized Career Insights – Most existing models classify programmers into broad job readiness categories but do not offer personalized career guidance based on individual performance trends.

Imbalance in Training Data – Research indicates that high-rated programmers have more data available, potentially biasing models and resulting in inaccurate predictions for lower-rated users.

Need for More Robust Evaluation Metrics – Many ML studies prioritize classification accuracy, but other critical metrics such as precision, recall, and long-term career growth tracking are often overlooked.

Deployment in Real-World Applications – While numerous academic studies have successfully developed predictive models, very few have been deployed for public use or integrated into real-world hiring processes.

By addressing these gaps, our research aims to develop a more comprehensive and practical system that leverages competitive programming data to provide accurate and personalized career predictions.

CHAPTER III

Methodology

This chapter details the methodology followed in developing the Codeforces Job Prediction System, covering dataset acquisition, preprocessing, model development, and system implementation.

3.1 Data Collection and Preprocessing

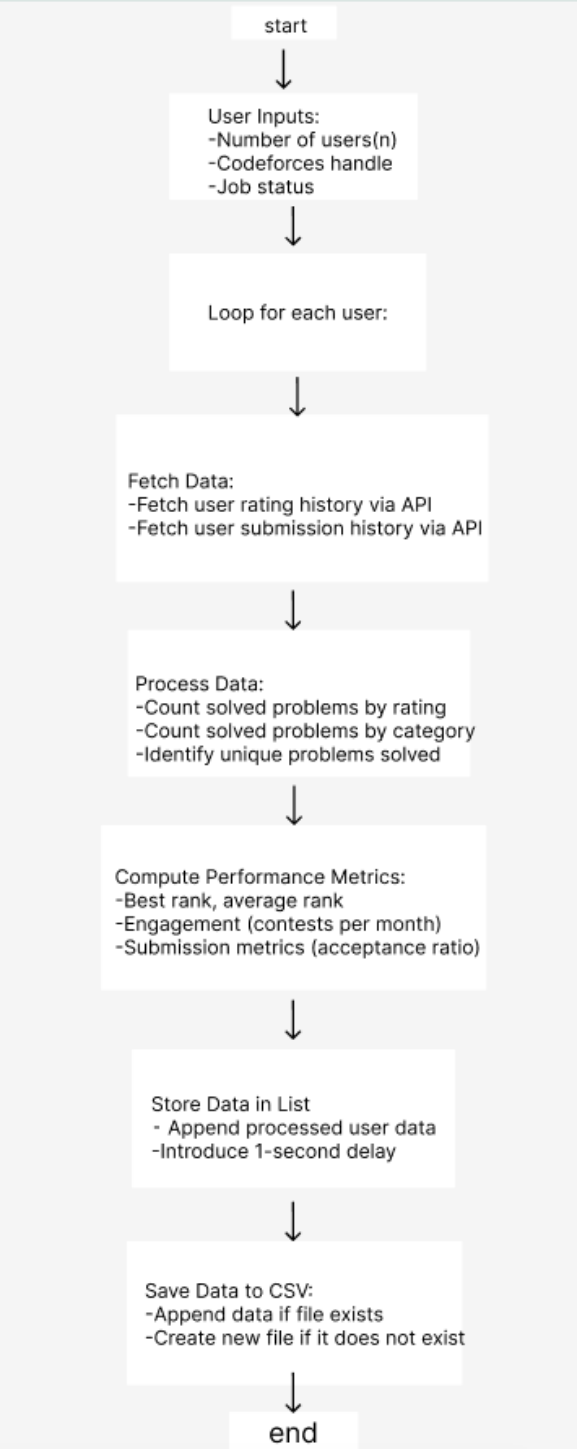

Figure 3.1: Flowchart of data collection and preprocessing

3.1.1 Codeforces API Data Extraction

The Codeforces API is an essential tool for extracting data related to competitive programming activities on the Codeforces platform. It offers developers the ability to access a variety of data, including user statistics, contest results, problem details, and more. In this section, we'll provide an overview of the Codeforces API, explain its available endpoints, and describe how we used it to gather the data required for our job prediction model.

3.1.1.1 Overview of Codeforces API

Codeforces provides a public-facing API that facilitates access to structured data related to contests, problems, users, and performance history. This API follows the RESTful architecture, and it utilizes a simple request-response model through HTTP GET requests. The API delivers responses in JSON format, making it easy to parse and work with in programming languages like Python.

For more details, the official API documentation can be found here:
`https://codeforces.com/apiHelp`

The API can be used in several different ways, including:

- Analyzing user performance for research or development purposes
- Automating the tracking of contests and rankings
- Extracting problem sets to build practice material
- Observing trends and patterns in competitive programming

The format for each API request is simple and consistent, following the pattern:
`https://codeforces.com/api/{methodName}?parameters`

For instance, to access a user's rating history, you would use:
`https://codeforces.com/api/user.rating?handle=tourist`

3.1.1.2 Key API Endpoints

The Codeforces API offers several endpoints, each designed to provide specific functionality. Below are the key endpoints, some of which were used in our study, along with other useful options available in the API:

User-Related Endpoints These endpoints provide information about users, including their performance and activity.

user.info: This endpoint returns information about one or more users.
Example: `https://codeforces.com/api/user.info?handles=tourist,Um_nik`
It includes details such as username, rank, rating, and title.

user.rating (Used in our project): This endpoint provides the rating history of a user.
Example: `https://codeforces.com/api/user.rating?handle=tourist`
It is especially useful for tracking performance and improvement over time.

user.status (Used in our project): This endpoint provides submission history for a user.
Example: `https://codeforces.com/api/user.status?handle=tourist`
It gives details such as problems solved, verdicts (e.g., Accepted, Wrong Answer), and programming languages used.

Contest-Related Endpoints These endpoints provide information related to Codeforces contests.

contest.list: This endpoint returns a list of all past and upcoming contests.
Example: `https://codeforces.com/api/contest.list`
It includes details such as contest names, start times, durations, and participant statistics.

contest.standings: This endpoint provides standings for a specific contest.
Example: `https://codeforces.com/api/contest.standings?contestId=5666&from=1&count=5`
It includes rankings, scores, and submission counts for top contestants.

contest.ratingChanges: This endpoint shows the rating changes for users in a particular contest.
Example: `https://codeforces.com/api/contest.ratingChanges?contestId=566`
This helps to analyze how a contest affects user ratings.

problemset.recentStatus: This endpoint returns recent submissions made by users.
Example: `https://codeforces.com/api/problemset.recentStatus?count=10`
It is useful for tracking real-time problem-solving activities on the platform.

Problem-Related Endpoints These endpoints provide details about the problems available on Codeforces.

problemset.problems: This endpoint returns a list of problems from the Codeforces problem set.
Example: `https://codeforces.com/api/problemset.problems`
It includes problem names, difficulty ratings, and associated tags like "dynamic programming" or "graphs."

problemset.recentStatus: This endpoint provides recent submissions from all users.
Example: `https://codeforces.com/api/problemset.recentStatus?count=10`
It is useful for monitoring problem-solving activity in real-time.

3.1.1.3 Data Extraction Process

The data extraction process involved pulling competitive programming statistics from the Codeforces API and organizing them into a dataset that could be used for predictive modeling. The primary objective was to collect historical performance data for Codeforces users, including their rating changes, problem-solving habits, and submission activity.

Steps for Data Extraction

1. **Collecting User Handles**
   The system accepts multiple Codeforces usernames (handles) as input. Each user is asked to provide their handle as well as their current job status, which is crucial for supervised learning.

Figure 3.2: User handle 1

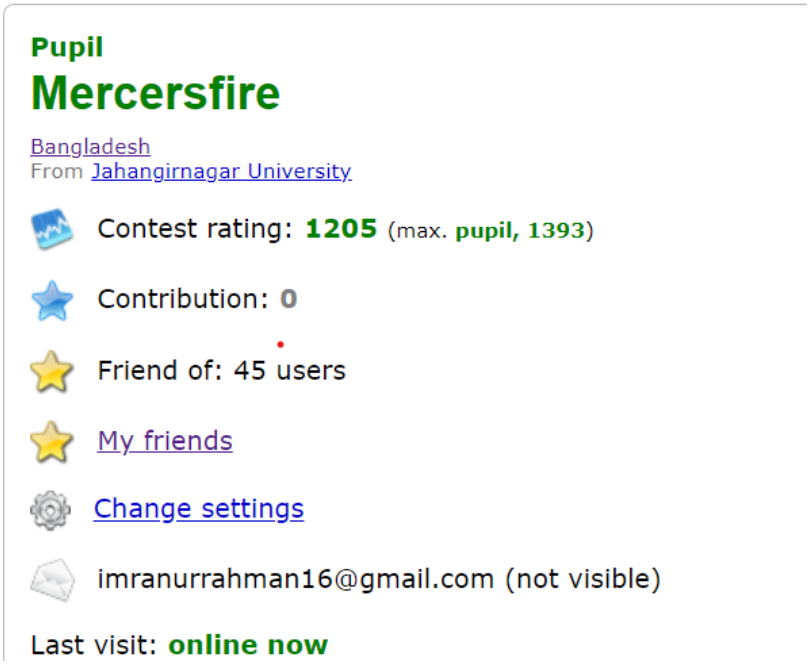

Figure 3.3: User handle 2

2. **Fetching Rating History (user.rating Endpoint)**

API calls are made to `user.rating?handle={handle}` for each user to retrieve their rating history. The key data points extracted include:

- New Rating: The user's updated rating after a contest.
- Rank: The user's rank in various contests.
- Contest Timestamps: To calculate engagement metrics based on the times of contests.

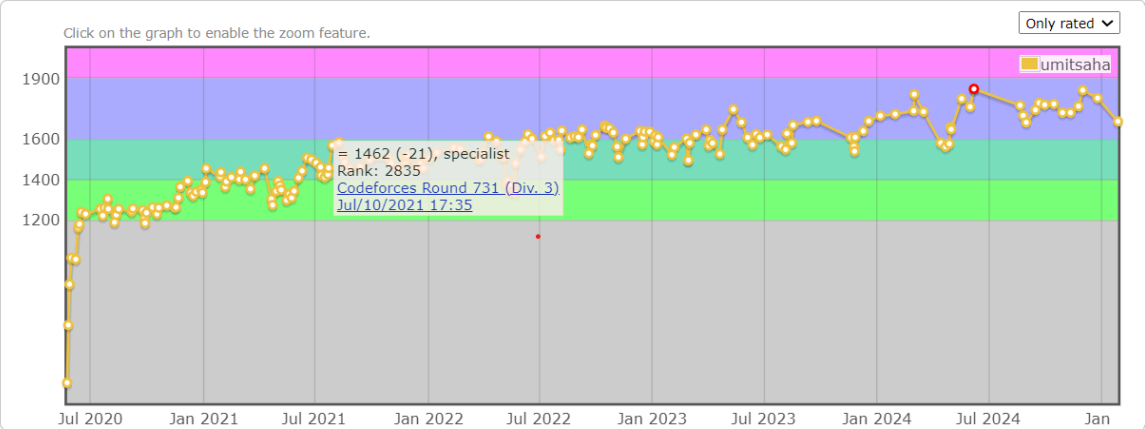

Figure 3.4: Rating history of a user

3. **Fetching Submission History (user.status Endpoint)**
   For each user, an API request is made to `user.status?handle={handle}` to gather submission data. Key fields that are extracted include:

   - Verdict: Whether the submission was successful (e.g., Accepted, Wrong Answer, etc.).
   - Problem Rating: The difficulty of the problems solved.
   - Problem Tags: Categories associated with problems like "dynamic programming" and "graphs."
   - Submission Timestamp: To analyze activity frequency over time.

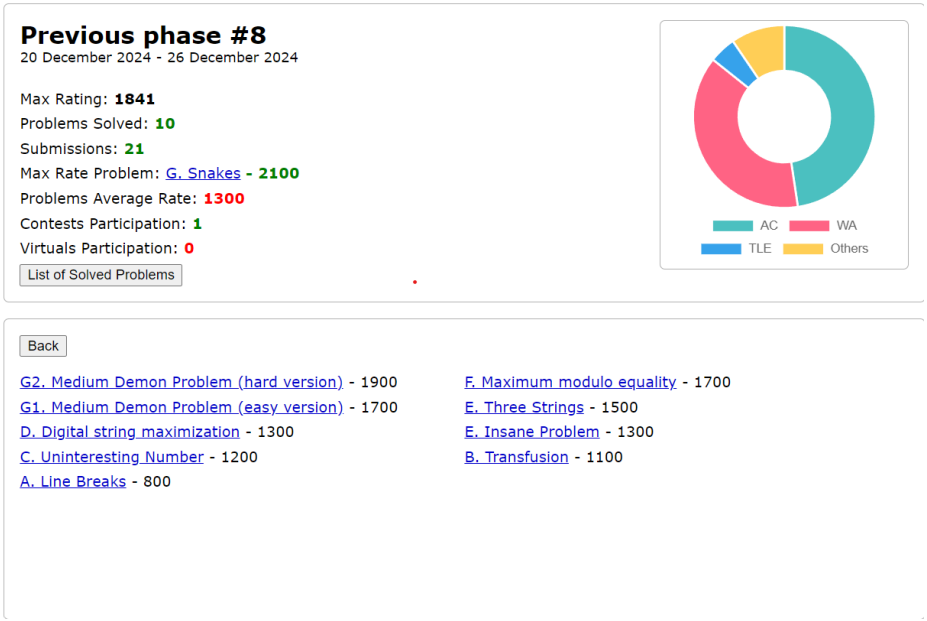

Figure 3.5: Users submission history of a week

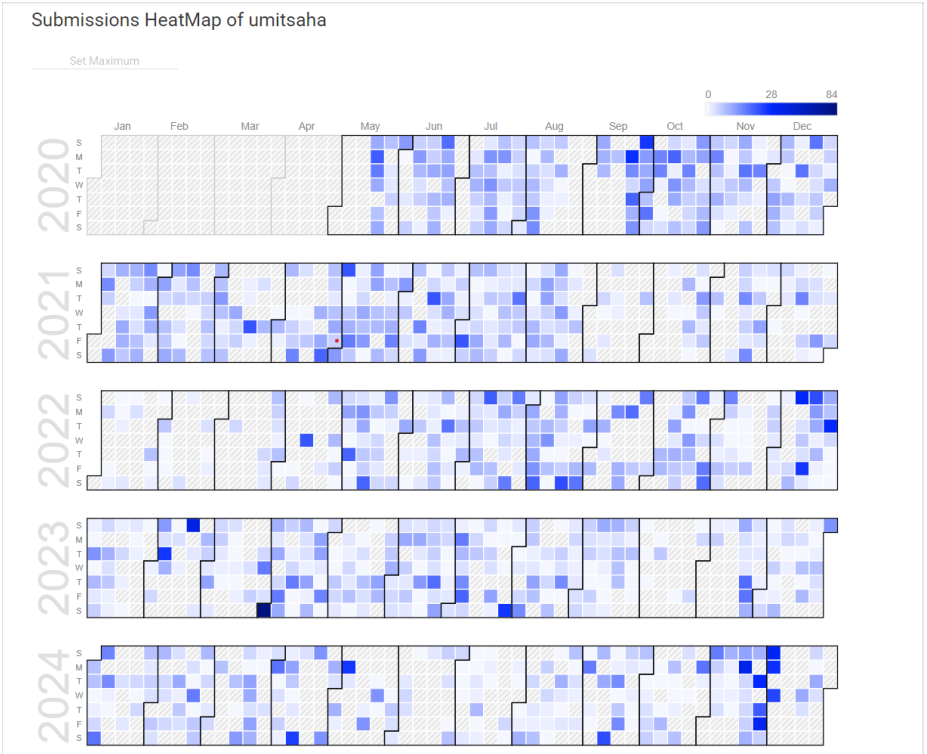

Figure 3.6: User submission hitmap

Processing Data

Once the data is extracted, several statistics are computed:

Problem-Solving Statistics: The number of problems solved within different difficulty ranges (e.g., 800-1100, 1200-1400, etc.).

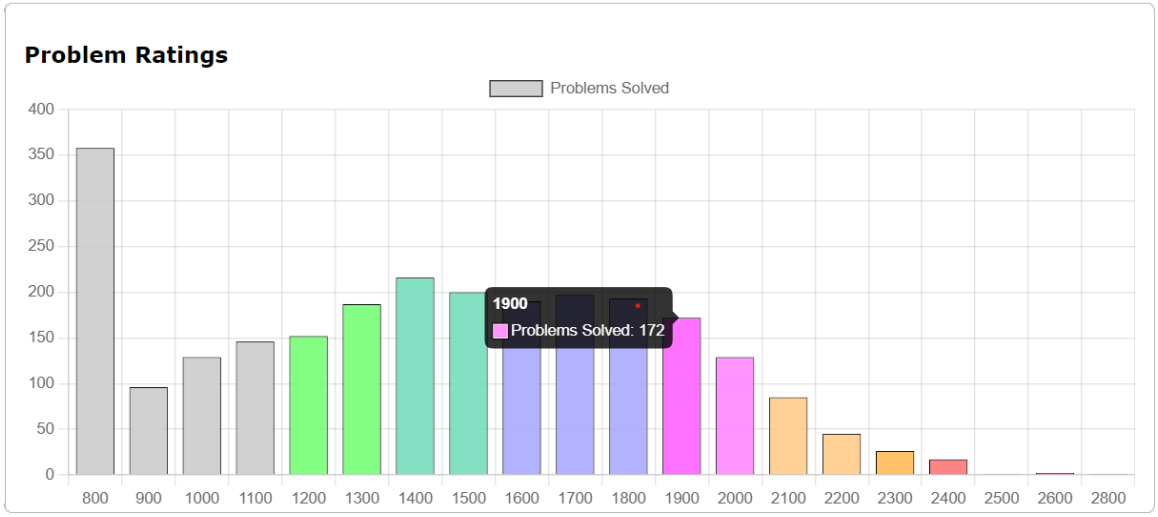

Figure 3.7: Users problem ratings

Categorization by Problem Tags: Counting the number of problems solved in various categories.

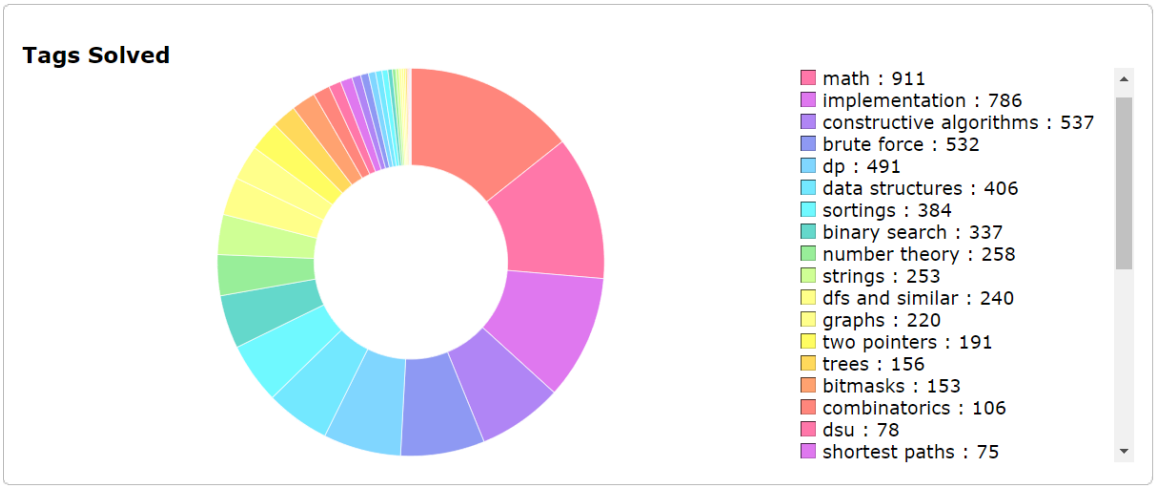

Figure 3.8: Users problem tags

Engagement Metrics: Calculating the number of contests participated in per month using contest timestamps.

Contests of	umitsaha
Number of contests	200
Best rank	130 (1822)
Worst rank	11007 (1365)
Max up	400 (1360)
Max down	-154 (1956)

Figure 3.9: Contest participation

Submission Frequency: Determining the average number of submissions per day and the overall acceptance ratio.

Some numbers about	umitsaha
Tried	2676
Solved	2617
Average attempts	2.67
Max attempts	64 (1812-B)
Solved with one submission	1417 (54.15%)
Max AC(s)	5 (220-B)

Figure 3.10: Submission frequency

Performance Metrics: Identifying the best rank, average rank across all

contests, and the average rank for the top 50 performances.

4. **Saving Extracted Data**

   The processed data is saved in a structured CSV file (`codeforces_user_data_with_category`).
   The script checks if this file already exists and appends new data if needed.

3.1.1.4 Challenges in API Data Extraction

During the process of extracting data from the Codeforces API, several challenges were encountered:

- **Rate Limiting & API Downtime**
  The Codeforces API imposes rate limits that can cause delays in data retrieval.
  Solution: A time delay of one second between API calls was implemented using `time.sleep(1)` to prevent hitting rate limits.
- **Handling Incomplete or Missing Data**
  Some users lacked contest histories or submission records, which led to missing data for certain fields.
  Solution: Default values (e.g., 0 or NaN) were used for missing fields to ensure the dataset remained properly structured.
- **Error Handling for Invalid Users**
  If a user entered an invalid Codeforces handle, the API would return an error.
  Solution: The script checks the response status before attempting to parse the data and provides error messages for invalid handles.
- **Data Processing Complexity**
  Certain features, such as categorizing problems and tracking ranking histories, required multiple iterative calculations.
  Solution: Python dictionaries and sets were used to efficiently count problem tags and eliminate redundant data.
- **CSV File Handling**
  The dataset had to be updated without overwriting any existing data.
  Solution: The `os.path.exists()` function was used to check if the CSV file already existed, and new data was appended accordingly.

3.1.2 Features Extracted for Prediction

The effectiveness of any machine-learning model relies significantly on the quality and relevance of the features derived from raw data. In this study, data extracted from the Codeforces API was processed to generate features reflecting a user's problem-solving abilities, consistency, and engagement. These features act as measurable indicators of a user's likelihood of securing a job as a software engineer. By carefully selecting and engineering these features, the model gains a comprehensive understanding of how a participant's competitive programming performance correlates with job placement outcomes.

3.1.2.1 Feature Categories

To facilitate a structured analysis, the extracted features were grouped into four major categories. Each category represents a distinct aspect of a user's competitive programming profile:

User Performance Metrics – This category evaluates a user's overall success and performance in competitive programming through measures such as ratings, ranks, and problem-solving records.

Engagement & Activity Metrics – These features capture the frequency and consistency of user participation by analyzing contest engagement and submission patterns over time.

Problem-Solving Skills – This dimension examines the user's ability to solve problems across different levels of difficulty and tracks their expertise in specific algorithmic categories.

Ranking & Rating Trends – This category monitors the user's growth trajectory by analyzing rating improvements and significant changes over time.

3.1.2.2 Extracted Features

A diverse range of features was extracted to represent the users' performance comprehensively. These features were selected to capture both quantitative and qualitative aspects of competitive programming proficiency.

User Performance Metrics: Key indicators of a participant's competitive programming ability were derived, including the best rating, which reflects the highest rating a user has ever achieved on Codeforces. Another important feature is the total number of contests participated in, which reveals the user's experience level and sustained engagement in competitive programming. The problems solved feature quantifies the total number of unique problems a user has successfully completed with an accepted verdict, providing insight into their problem-solving efficiency.

Additionally, the average problem rating represents the mean difficulty of the problems the user has solved, indicating whether the user is consistently tackling more advanced problems. The acceptance ratio measures the proportion of successful submissions relative to the total attempts, highlighting the user's accuracy and efficiency. To further capture performance in contests, best rank records the lowest rank (i.e., the user's best performance in a contest), while the average rank gives an overall measure of a user's typical contest standing. For a more focused analysis of top performances, the average rank of the best 50 performances calculates the mean rank across the user's most successful contests, if available.

Engagement & Activity Metrics: Engagement and consistency are essential indicators of a user's dedication to improving their skills. To measure this, the contests per month feature calculates how regularly the user participates in rated contests. Another critical engagement metric is the average submissions per day, which reflects the user's daily activity level based on the timestamps of their first and last recorded submissions. Days active on Codeforces measures the total span of time a user has been active on the platform, providing a timeline of their sustained engagement.

Problem-Solving Skills: To capture the depth and breadth of a user's problem-solving capabilities, the dataset includes information on the number of problems solved within different difficulty ranges. These difficulty categories were divided into five tiers:

- 800-1100: Beginner-level problems.
- 1200-1400: Intermediate-level problems.
- 1500-1800: Advanced problems requiring more complex algorithms.

- 1900-2100: Expert-level problems, often reflecting deep algorithmic understanding.
- 2100+: Elite-level problems, indicative of exceptional problem-solving capabilities.

In addition to problem difficulty, the dataset tracks problems solved by category, which reflects expertise in various algorithmic and computational domains. This feature identifies the number of problems a user has solved in specialized areas such as "graphs," "dynamic programming (dp)," "brute force," and more advanced categories like "string suffix structures." This categorization is particularly useful for assessing a candidate's preparedness for real-world software engineering tasks, as many technical interviews emphasize similar algorithmic topics.

Ranking & Rating Trends: Understanding how a user's performance evolves over time provides valuable insights into their growth and improvement potential. Rating progression records the user's rating after each contest, enabling an analysis of how consistently a participant improves over time. Another important feature is the improvement rate, which measures the difference between the user's initial and most recent rating, offering a straightforward metric of long-term progress. To capture rapid advances, the number of rapid rating jumps tracks instances where a user experienced significant rating increases within a short time frame, reflecting sudden spikes in skill and performance.

3.1.2.3 Significance of Feature Selection

The features selected for this study provide a comprehensive profile of a Codeforces participant, capturing their skill, dedication, and long-term progress. Each category of features contributes unique insights that collectively enhance the predictive power of the machine-learning model. By incorporating diverse performance indicators—ranging from contest outcomes to problem-solving patterns—this feature set allows the model to effectively distinguish between users who are more likely to secure technology-related jobs and those who are not.

Through the extraction and careful structuring of these features, the model is equipped to analyze how specific aspects of competitive programming correlate with real-world job placement outcomes. This approach not only strengthens

the model's predictive accuracy but also offers practical insights for programmers aiming to enhance their employability and for recruiters seeking to identify promising technical talent.

3.1.2.4 Feature Importance for Prediction

Understanding the relative importance of each feature is crucial for improving the accuracy and interpretability of the predictive model. After extracting the relevant features from the Codeforces dataset, we conducted an analysis to identify which variables had the most significant impact on predicting job placement outcomes. This analysis not only helps in optimizing the model's performance but also provides insights into the attributes that are most indicative of a candidate's potential employability.

The most influential features in our model included the best rating, contests participated, acceptance ratio, and average problem rating. These features strongly reflect a user's technical proficiency, consistency, and ability to solve problems accurately—factors that align closely with the competencies sought by technology employers. Users with higher ratings and consistent contest participation demonstrate sustained engagement and advanced problem-solving capabilities, making these features highly predictive of successful job outcomes.

Moderately important features included solved problems by category and engagement metrics. Analyzing the diversity of problem categories solved provides a nuanced understanding of a candidate's breadth of knowledge, while engagement metrics, such as participation frequency, highlight the user's commitment and practice habits. Although these features are less predictive on their own, they add meaningful context when combined with other performance measures.

Less impactful were features such as rapid rating jumps and unique problems solved by difficulty range. While rapid improvements in rating suggest bursts of progress, they do not necessarily correlate with long-term performance or job readiness. Similarly, the distribution of problems solved across different difficulty levels, though reflective of problem-solving ability, showed weaker predictive power when compared to other metrics.

By identifying and weighting these features appropriately, our model becomes better equipped to assess and predict a participant's likelihood of securing a software engineering role. This feature importance analysis also provides valuable

feedback for competitive programmers by highlighting the aspects of performance that most strongly correlate with career success.

3.1.3 Handling Missing or Incomplete Data

Dealing with missing or incomplete data is a common challenge when working with real-world datasets, and it plays a crucial role in maintaining the accuracy and reliability of machine-learning models. Since our dataset relies on information extracted from the Codeforces API, gaps in data can arise due to several reasons, including user inactivity, system errors, or API limitations. Without proper handling, these inconsistencies could undermine the model's effectiveness and lead to biased predictions.

To address these challenges, we adopted a systematic approach to manage missing data and ensure the integrity of our dataset. When user records were incomplete—such as users with no contest history or missing submission details—we assigned default values (e.g., zero or NaN) to maintain a consistent dataset structure. This approach allowed us to standardize the dataset without discarding valuable information, ensuring that the machine-learning model could still process and learn from partial records.

Additionally, error handling was implemented to manage cases where invalid or nonexistent Codeforces handles were provided. Whenever an API request returned an error, we verified the response before including the data in our dataset. This validation step prevented erroneous entries and maintained the accuracy of the extracted information. Furthermore, we accounted for API rate limitations by introducing time delays between requests to avoid exceeding the permitted call frequency, ensuring the completeness of our data extraction process.

By adopting these strategies, we minimized the impact of missing or incomplete data on our model. This careful data handling improved the model's robustness and ensured that the insights derived from the study reflected real-world scenarios. As a result, our predictions remain reliable and generalizable to a wider range of competitive programmers and employment outcomes.

3.1.3.1 Sources of Missing Data

Missing or incomplete data in the dataset can originate from several sources, each affecting the quality and reliability of the model. One significant source is inactive users, where some Codeforces users create accounts but never participate in contests or submit solutions, resulting in missing values for key metrics like "Best Rating" and "Contests Participated." Another common issue arises from incomplete rating history—users who have participated in only a few contests lack sufficient data to compute advanced metrics such as "Best Rank" or "Average Rank in Best 50 Contests." Additionally, unrecorded submissions contribute to missing data when users attempt problems but do not achieve an "Accepted" verdict, leading to zero problems being counted as solved. There are also cases of missing problem ratings, as not all problems on Codeforces come with a predefined difficulty rating, affecting calculations related to "Average Problem Rating." Furthermore, API call failures due to network issues, rate limits imposed by the Codeforces API, or temporary server downtimes may result in incomplete or missing records during data extraction.

3.1.3.2 Strategies for Handling Missing Data

To maintain the integrity and usability of the dataset, several strategies were implemented to address missing or incomplete data effectively:

(a) **Default Value Substitution**
Missing values for critical features were substituted with default values to maintain dataset consistency. For instance, if a user had no contest participation history, fields such as "Best Rating" and "Contests Participated" were assigned a value of zero. Similarly, when a solved problem lacked a predefined rating, it was replaced with the average rating of comparable problems solved by other users.

(b) **Mean/Median Imputation**
For features where missing data could significantly impact model performance, mean or median imputation was applied. In cases where "Average Problem Rating" was missing, the median rating of all successfully solved problems was used as a substitute. For features like "Best Rank" and "Average Rank," if users participated in fewer than five contests, the median rank from users with similar participation levels was used to fill the gaps.

(c) **Dropping Unusable Records**
Records with no meaningful information for prediction were excluded from the dataset. Specifically, users who lacked both contest participation and any recorded submissions were removed, as their data did not contribute to model learning.

(d) **Data Interpolation for Rating Progression**
For users with gaps in their rating history due to skipped contests, missing values were estimated using linear interpolation. This approach relied on known values before and after the missing entries, allowing us to maintain a smooth rating progression without introducing artificial bias.

3.1.3.3 Impact of Missing Data Handling on Model Performance

Implementing robust strategies for handling missing data played a crucial role in improving the model's effectiveness and accuracy. By preserving as much user information as possible, the model maintained a richer dataset for training without discarding valuable insights. Addressing missing values also reduced bias by preventing the overrepresentation of highly active users while still including data from less active participants. Furthermore, these measures improved prediction accuracy by ensuring that incomplete records did not distort the model's learning process, leading to a more balanced and reliable prediction output.

3.1.4 Data Normalization and Transformation

Normalization and transformation are critical preprocessing techniques that ensure all features contribute equally to model training. Since the dataset derived from the Codeforces API includes diverse feature types—such as numerical metrics, rankings, and categorical data—applying these methods enhances model performance by improving convergence and reducing the influence of features with large magnitudes.

3.1.4.1 Importance of Data Normalization and Transformation

Raw datasets often consist of features with varying scales and distributions, which can negatively affect model performance if left unprocessed. For instance, features like "Best Rating," which ranges from 0 to 3500, can dominate

smaller-scale features such as "Acceptance Ratio," which falls between 0 and 1. This disparity can bias the model toward features with larger numerical values. Additionally, some machine learning algorithms assume normally distributed data, meaning that unnormalized inputs may slow down the training process and hinder the model's ability to generalize effectively. Models like K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) are particularly sensitive to differences in magnitude, leading to inaccurate predictions if the data is not standardized. By applying appropriate normalization and transformation techniques, these issues were mitigated, resulting in better model accuracy and faster convergence.

3.1.4.2 Techniques Used for Normalization and Transformation

(a) **Min-Max Normalization (Feature Scaling)**
This technique scales all numerical features to a uniform range between 0 and 1, ensuring that each feature has equal weight during model training. It is calculated using the formula:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Where X' represents the scaled feature value, X is the original value, and $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of the feature, respectively.
Applied to:

- Best Rating
- Contests Participated
- Problems Solved
- Average Problem Rating
- Best Rank
- Engagement (Contests per Month)
- Average Submissions per Day

(b) **Standardization (Z-Score Normalization)**
For features with high variability, Z-score normalization was applied to ensure a mean of 0 and a standard deviation of 1. This transformation is

particularly useful for metrics with different scales and is calculated as:

$$X' = \frac{X - \mu}{\sigma}$$

Where μ is the mean and σ is the standard deviation of the feature.

Applied to:

- Average Rank (Best 50 Contests)
- Average Rank
- Rating Progression

(c) **Log Transformation**

Log transformation was used to manage skewed data distributions and reduce the impact of extreme values. This transformation is expressed by the formula:

$$X' = \log(X + 1)$$

This approach was particularly useful for features where certain users had abnormally high values.

Applied to:

- Problems Solved (to account for users solving thousands of problems)
- Submissions per Day (to normalize users with extreme activity levels)

(d) **Encoding Categorical Variables**

Categorical features were transformed into a format usable by the model through encoding techniques. One-Hot Encoding was applied to represent problem categories (such as "dynamic programming" or "greedy") as binary features. If a user solved problems in a given category, the corresponding value was set to 1; otherwise, it was set to 0. For target labels like "Job Status," Label Encoding was used to convert career outcomes into numeric values:

- 0 = No Job
- 1 = Entry-Level Job
- 2 = High-Paying Job
- 3 = Tech Giants

3.2 Model Development and Training

This chapter describes the methodology used to develop the Codeforces Job Prediction System. It explains the selection of machine learning models, the data pipeline for training and testing, feature engineering techniques, handling imbalanced data, model training, performance evaluation, and deployment.

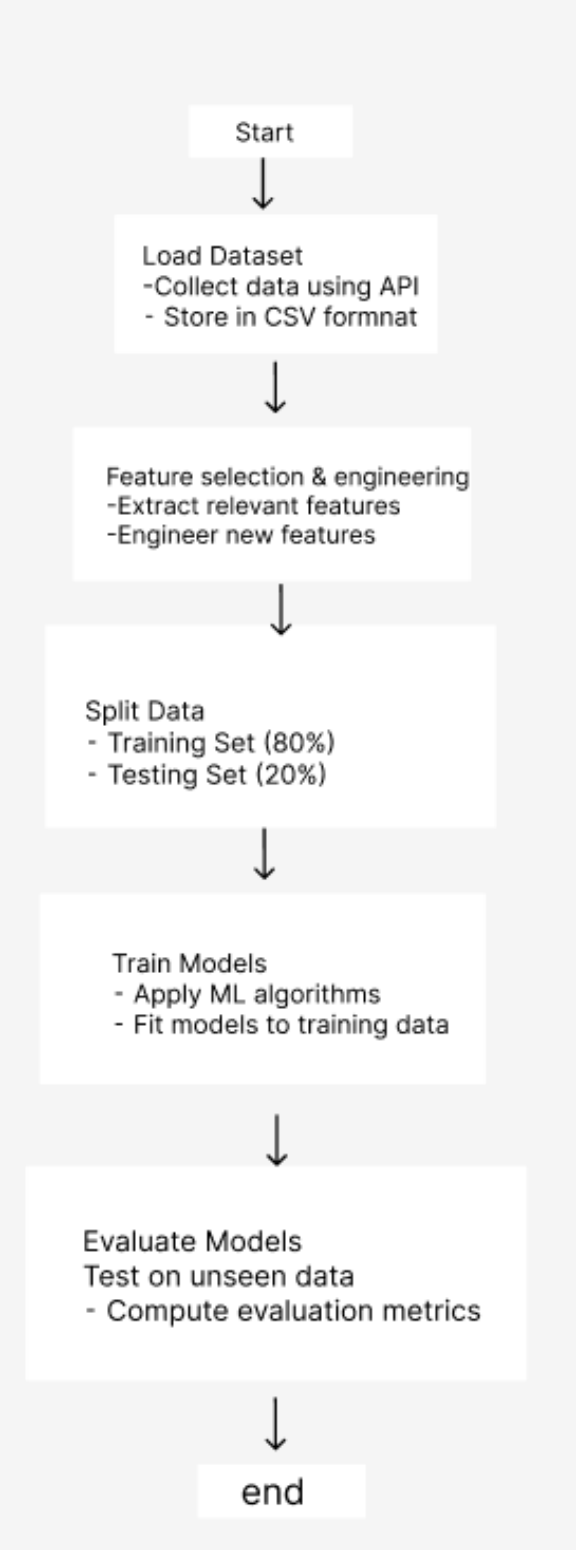

Figure 3.11: Model development flowchart

3.2.1 Model Selection and Justification

The goal of this project is to predict the job status of Codeforces users based on their competitive programming activity. To achieve this, three machine learning models were evaluated:

- **Random Forest Classifier**
- **Support Vector Machine (SVM)**
- **K-Nearest Neighbors (KNN)**

3.2.1.1 Random Forest Classifier

Random Forest is an *ensemble learning method* that operates by constructing multiple *decision trees* during training and outputs the class that is the *mode of the predictions* from all trees. It reduces *overfitting*, improves *accuracy*, and handles *high-dimensional data* effectively. It is particularly useful when dealing with *non-linear relationships* in data and *missing values*.

3.2.1.2 Support Vector Machine (SVM)

SVM is a *supervised learning algorithm* that finds the optimal *hyperplane* to separate different classes in a high-dimensional space. It works well for *both linear and non-linear classification problems* using *kernel functions*. SVM is effective in *high-dimensional spaces* and robust against *overfitting*, especially in cases where the number of features is greater than the number of samples.

3.2.1.3 K-Nearest Neighbors (KNN)

KNN is a *non-parametric, instance-based learning algorithm* that classifies new data points based on the *majority vote of k-nearest neighbors*. It calculates distances (e.g., *Euclidean distance*) to determine the closest training samples. KNN is simple and effective but can be computationally expensive for *large datasets*.

Among these, the Random Forest Classifier was selected as the final model due to its high accuracy, robustness, and ability to handle complex feature relationships.

3.2.1.4 Comparison of Models

Model	Advantages	Disadvantages	Accuracy
Random Forest	Handles missing data, prevents overfitting, interpretable	Can be slow for large datasets	Best Performing
SVM	Effective in high-dimensional spaces	Computationally expensive for large data	Moderate
KNN	Simple, easy to understand	Poor performance for large datasets	Low

Table 3.1: Comparison of Machine Learning Models

3.2.2 Data Pipeline for Training and Testing

The machine learning workflow follows a structured data pipeline consisting of:

Loading the Dataset: Data was collected using the Codeforces API and stored in CSV format.

Feature Selection and Engineering: Various features were extracted and engineered to improve model performance.

Data Splitting: The dataset was divided into training (80%) and testing (20%) sets.

Model Training: Different machine learning models were trained using the training dataset.

Model Evaluation: The trained models were tested on unseen data using various evaluation metrics.

3.2.3 Feature Engineering Techniques

Feature engineering is crucial for improving model accuracy. The following techniques were applied:

Derived Metrics: Additional features like contest participation, average problem rating, and acceptance ratio were created.

Feature Scaling: Data normalization was applied to bring all features to a comparable scale.

Categorical Encoding: Textual problem tags were converted into numerical representations.

Handling Sparse Data: Missing values were imputed with default values or the mean of available data.

3.2.4 Handling Imbalanced Data

Since some job statuses were underrepresented in the dataset, various techniques were employed to balance the data. **Oversampling** the number of samples in minority classes was increased to ensure a more balanced distribution. **Class Weights** higher importance was assigned to less frequent job categories to prevent bias in model training. **Synthetic Data Generation** the SMOTE (Synthetic Minority Over-sampling Technique) method was utilized to generate artificial samples, further improving class balance and enhancing the model's ability to generalize.

3.2.5 Training the Machine Learning Model

Random Forest Classifier

Support Vector Machine (SVM)

K-Nearest Neighbors (KNN)

3.2.6 Performance Metrics Used

The trained models were evaluated using the following metrics:

Accuracy: Measures the overall accuracy of the predictions.

Precision: Evaluates the proportion of correctly classified positive cases.

Recall: Measures how well the model identifies positive cases.

F1 Score: A harmonic mean of precision and recall.

Confusion Matrix: Displays misclassifications for each class.

3.2.7 Flask API Deployment Plan

To make the model accessible via a web application, a Flask-based API was developed.

3.2.7.1 Flask Application Code

The Flask application is designed to serve the machine learning model efficiently.

Developing a Flask API: The system includes a Flask-based API that facilitates interaction between users and the predictive model.

Implementing Endpoints: The application provides dedicated endpoints for handling predictions and model interactions, ensuring seamless integration and accessibility for users.

3.2.7.2 Deployment Strategy

The deployment process involves several key steps to ensure seamless integration and accessibility. First, the project is uploaded to GitHub, providing version control and collaboration capabilities. Next, a new web service is created on Render, a cloud platform that facilitates easy deployment. The dependencies required for the application are then configured using the `requirements.txt` file, ensuring that all necessary packages are installed. Finally, the API is deployed and thoroughly tested to verify its functionality and reliability before making it publicly accessible.

3.2.7.3 User Interface

A simple HTML frontend allows users to enter a Codeforces username and receive job status prediction.

Figure 3.12: User Interface

3.3 System Implementation and Deployment

The implementation and deployment of the system were carried out using a combination of modern software tools, technologies, and cloud services. Below is a detailed overview of the steps involved in implementing and deploying the system.

3.3.1 Software Tools and Technologies Used

The system was developed using a combination of open-source and widely used technologies. The core components of the system include:

Programming Languages: Python was used for backend development, while HTML, CSS, and JavaScript were used for the frontend.

Machine Learning Frameworks: Scikit-learn was used to develop machine learning models, including Random Forest, SVM, and KNN.

Flask: A lightweight and flexible Python web framework used to handle API requests and serve model predictions.

Frontend Libraries: Material UI components and Google Fonts were used for a modern UI design.

Version Control: Git was used for version control, with hosting on GitHub for collaboration.

Deployment Platform: The application was deployed on Render, enabling smooth web-based access.

3.3.2 Backend Implementation (Flask API)

The backend of the system is implemented using the Flask framework, a lightweight Python-based micro-framework ideal for small to medium-scale web applications. It serves as the primary API for handling requests from the frontend. The backend processes user input, fetches data from the Codeforces API, extracts relevant features (e.g., user ratings, solved problems, participation history), and passes them to the trained machine learning model for job status prediction. The Flask API handles various routes and endpoints, ensuring smooth interaction between the frontend and model. Additionally, it incorporates error handling for invalid usernames and potential failures in the Codeforces API.

3.3.3 Frontend Development

The frontend provides a user-friendly interface that seamlessly interacts with the backend. Built using HTML, CSS, and JavaScript, it features a retro-modern aesthetic with custom styling. **Input Form:** Users enter their Codeforces username, which is sent to the backend for prediction. **Material UI Components:** The UI utilizes Material UI for a clean and modern design, enhancing buttons and icons. **Dynamic Result Display:** Predictions are shown with color-coded status messages like "Keep pushing!" or "You're on your way!" **Responsive Design:** The interface adapts to different screen sizes, ensuring a smooth experience across devices.

3.3.4 API Integration for Model Inference

Integrating the machine learning model with the backend API enables real-time predictions. Upon receiving a Codeforces username, the backend fetches user

data via the Codeforces API, processes it, and applies the trained model to predict career status. The Flask API loads the pre-trained model (Random Forest, SVM, or KNN), extracts relevant features, and generates a prediction, which is then sent to the frontend for display. This seamless integration ensures fast and accurate responses, providing users with meaningful career insights in real time.

3.3.5 Hosting and Deployment on Render

The system was deployed using Render, following these steps:

GitHub Integration: The repository was linked to Render for automatic deployment.

Backend Deployment: The Flask API was deployed as a Web Service.

Frontend Hosting: HTML, CSS, and JavaScript files were deployed as a Static Site.

Continuous Deployment: Any changes to the GitHub repository are automatically deployed.

The final deployed web application is accessible at: *Codeforces Career Prediction.*

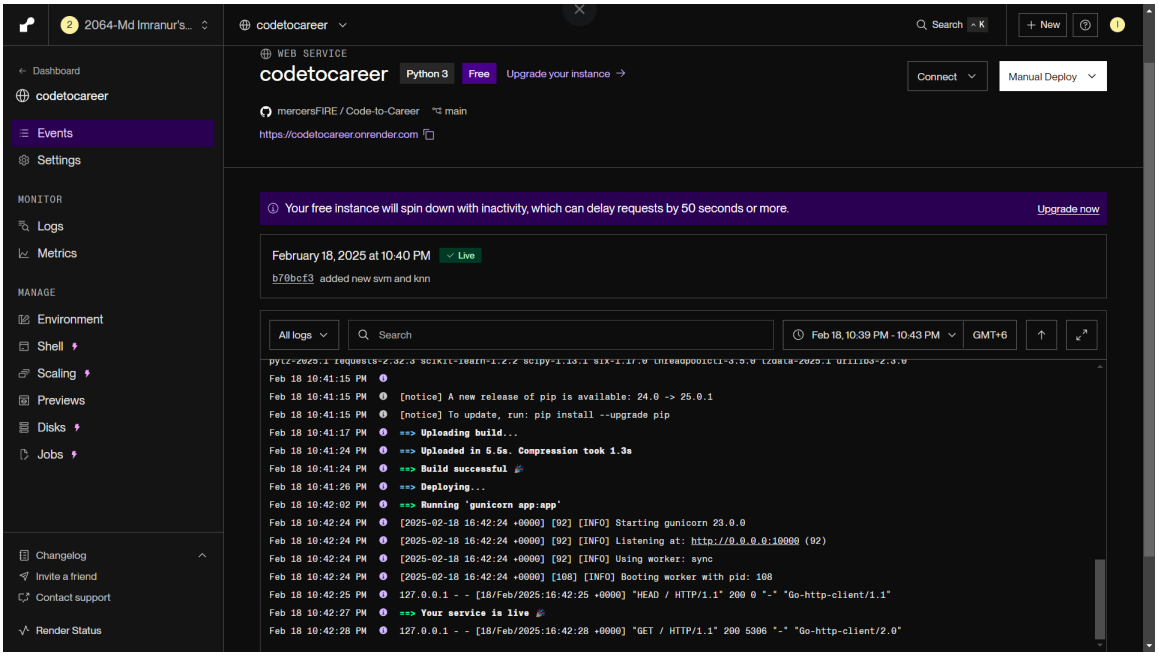

Figure 3.13: Deployed on Render

CHAPTER IV

Results and Analysis

This chapter presents an in-depth analysis of the model's performance, including its evaluation using multiple metrics, a comparison with traditional career prediction methods, real-world case studies from Codeforces users, and a discussion of the challenges encountered during the project.

4.1 Model Performance on Test Data

To evaluate the effectiveness of the machine learning models, we tested them on a separate dataset that was not used during training. This ensured an unbiased assessment of their predictive capabilities. Three models—Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN)—were compared to determine which performed best in predicting the career outcomes of Codeforces users.

The dataset used for testing included key features such as user ratings, the number of problems solved, contest participation, and problem-solving patterns over time. Once the models were trained and validated, they were applied to the test dataset to classify users into four career status categories:

- **0:** Needs improvement
- **1:** On the right track
- **2:** Strong candidate
- **3:** Highly competitive

Among the three models, Random Forest consistently delivered the most accurate and stable predictions, demonstrating its effectiveness in career outcome prediction based on competitive programming data.

4.2 Evaluation Metrics: Accuracy, Precision, Recall, and F1-score

4.2.1 Random forest

Splitting Data: 80% training, 20% testing. these performance are base on 20% testing

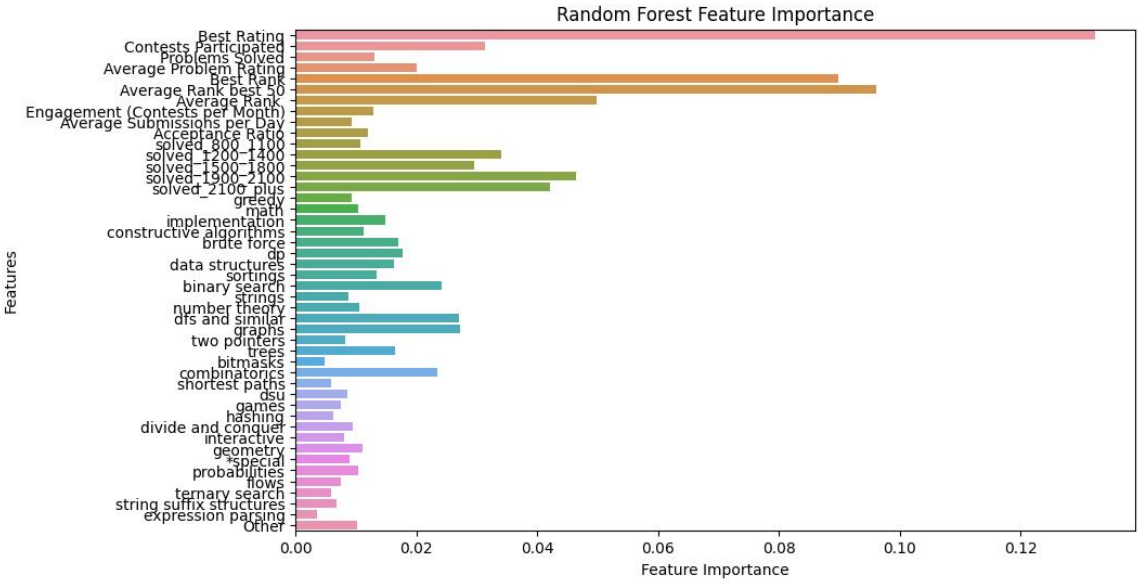

Figure 4.1: Features and their importance

rank	feature	importance
1	Best Rating	0.1207
2	Average Rank Best 50	0.1154
3	Best Rank	0.0759
4	Average Rank	0.0543
5	Contests Participated	0.0438
6	Solved_1900_2100	0.0343
7	Solved_1200_1400	0.0321
8	Solved_2100_plus	0.0264
9	Solved_1500_1800	0.0248
10	Dfs & similar	0.0243

Figure 4.2: Top Important features

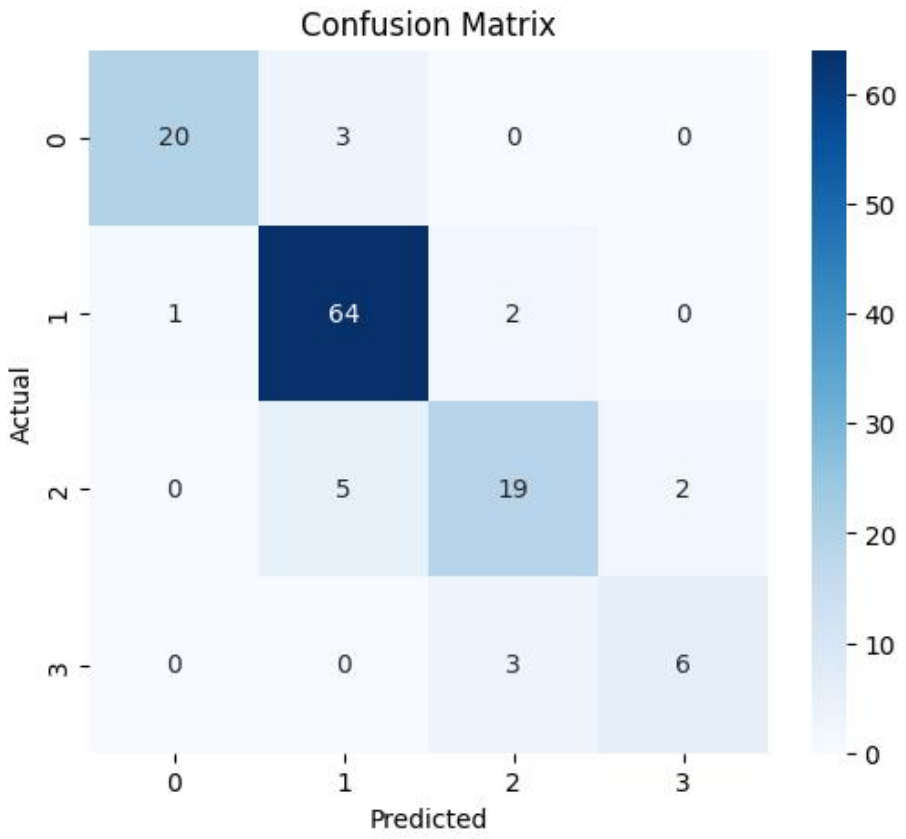

Figure 4.3: Confusion matrix

class	precison	recall	F1-Score	Support
0	0,92	1.00	0.96	23
1	0.94	0.93	0.93	67
2	0.76	0.73	0.75	26
3	0.67	0.67	0.67	9
overall accuracy	88%			

Figure 4.4: Classification report

4.2.2 SVM

Splitting Data: 80% training, 20% testing. these performance are base on 20% testing

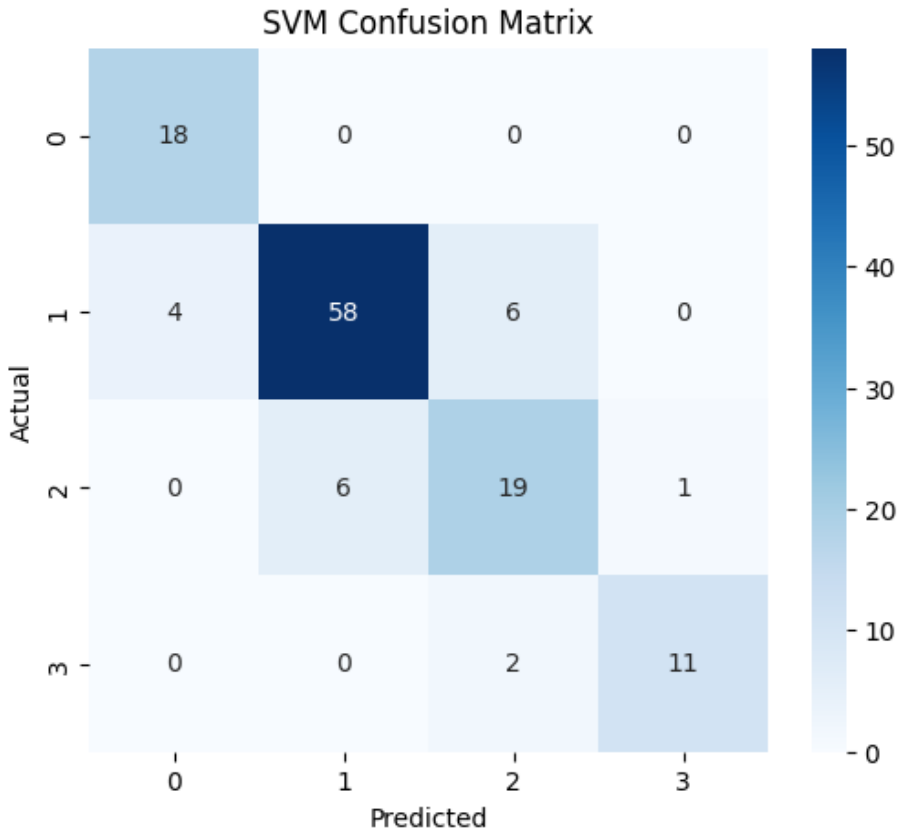

Figure 4.5: Confusion matrix

class	precison	recall	F1-Score	Support
0	0.82	1.00	0.90	18
1	0.91	0.85	0.88	68
2	0.70	0.73	0.72	26
3	0.92	0.85	0.88	13
overall accuracy	84.80%			
Macro average	0.84	0.86	0.84	
Weighted Average	0.85	0.85	0.85	

Figure 4.6: Classification report

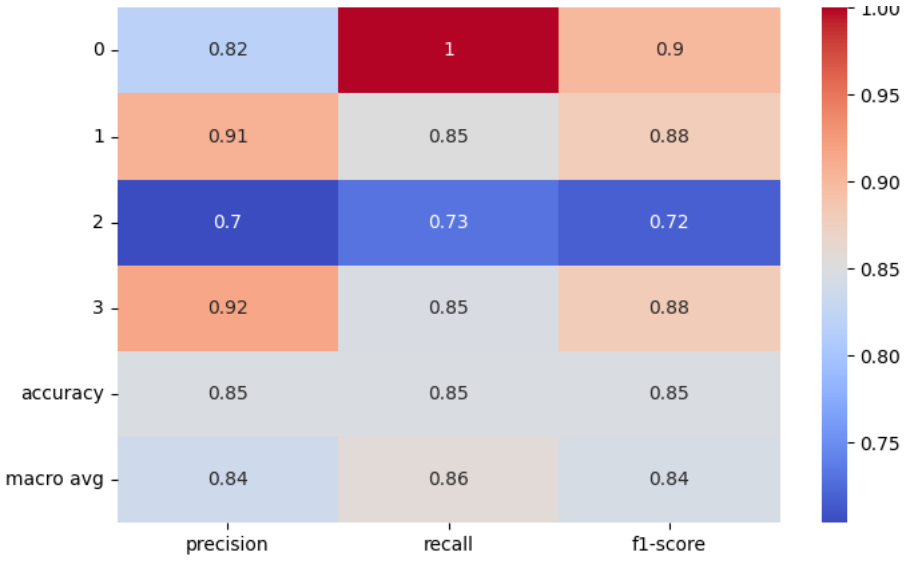

Figure 4.7: Classification report hitmap

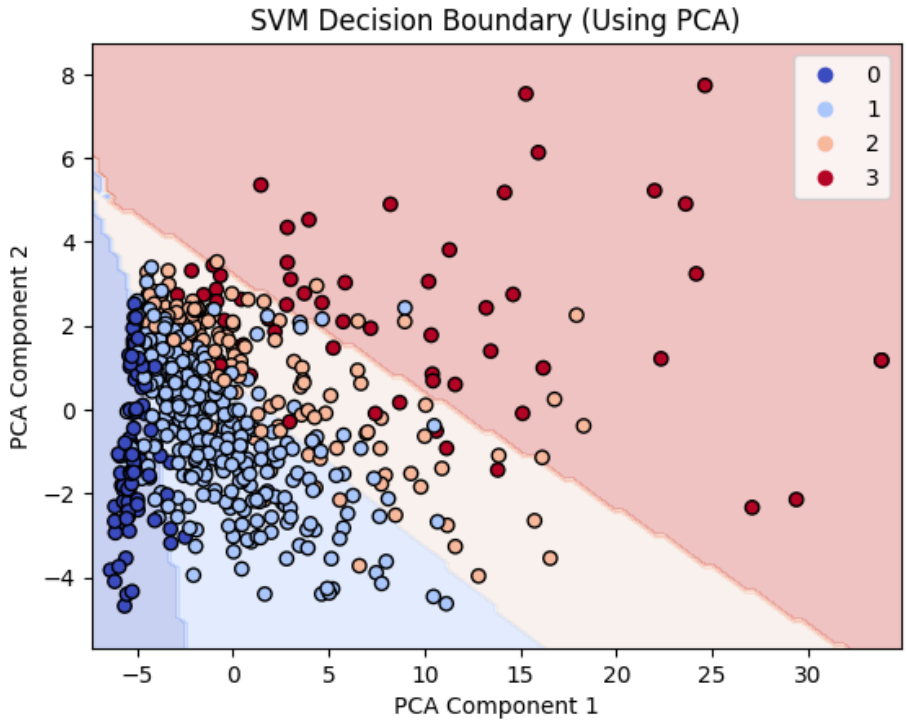

Figure 4.8: SVM decision boundary

4.2.3 KNN

Splitting Data: 80% training, 20% testing. these performance are base on 20% testing

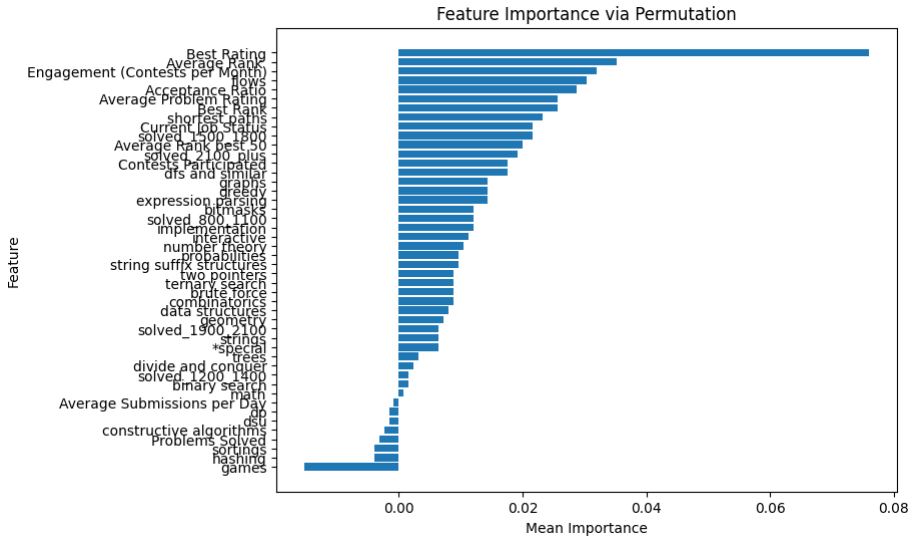

Figure 4.9: Feature importance

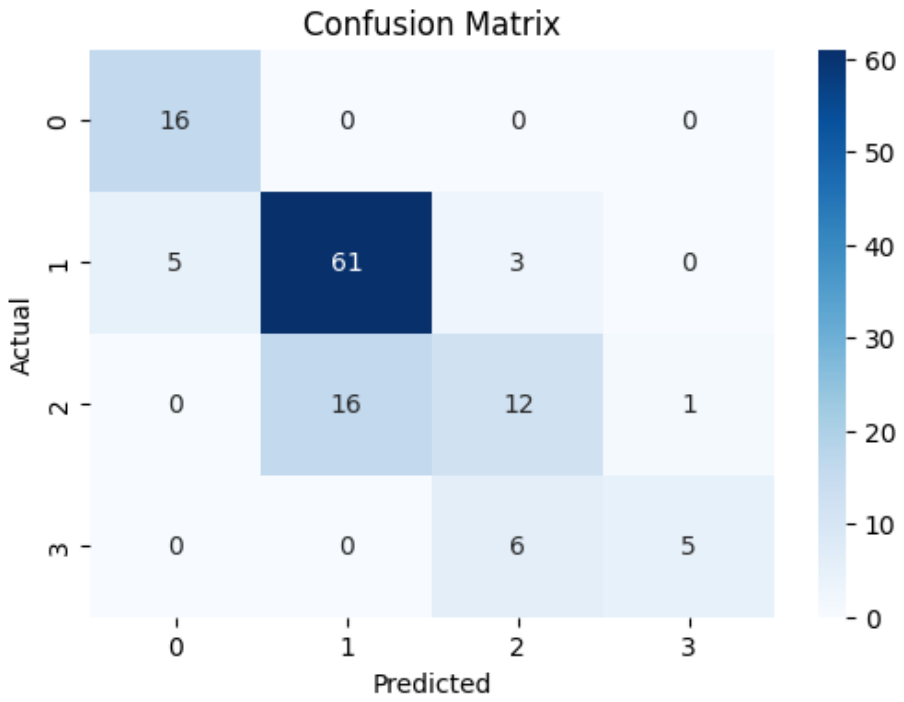

Figure 4.10: Confusion matrix

Overall, the Random Forest model outperformed SVM,KNN and other ML models across all metrics, making it the most reliable choice for predicting career potential based on competitive programming data.

4.3 Comparison with Traditional Career Prediction Methods

To put the performance of our machine learning models into perspective, we compared them with conventional approaches to career prediction and hiring. Traditional methods typically rely on factors like resumes, academic qualifications, and interviews, which do not always capture a candidate's real-time problem-solving abilities or technical skills.

In contrast, our machine learning model evaluates quantifiable data—such as user ratings, contest participation, and historical problem-solving trends—to make objective and data-driven predictions. This approach offers several advantages over traditional hiring methods:

Identifying Hidden Talent: The model can recognize promising candidates

who may not have prestigious degrees or strong resumes but excel in problem-solving and algorithmic thinking.

Reducing Bias: Unlike human evaluations, which can be influenced by subjective factors, machine learning models offer a standardized and consistent assessment.

Speed and Scalability: The model can assess large volumes of candidates quickly and efficiently, automating the evaluation process for technical hiring.

By leveraging these benefits, our machine learning-based approach presents a more effective and scalable way to identify top programming talent, particularly in fields that prioritize problem-solving skills.

4.4 Case Studies on Real Codeforces Users

To demonstrate how the model works in real-world scenarios, we analyzed the performance of actual Codeforces users. These case studies highlight how the model predicts career status based on different competitive programming metrics.

4.4.1 User 1: "MARUF_1903"

- **Highest Rating:** 1082
- **Problems Solved:** 28
- **Contests Participated:** 10
- **Predicted Career Status:** 0 (Needs Improvement)

Figure 4.11: Level 0

4.4.2 User 2: "mercersfire"

- **Highest Rating:** 1393
- **Problems Solved:** 720
- **Contests Participated:** 159
- **Predicted Career Status:** 1 (On the Right Track)

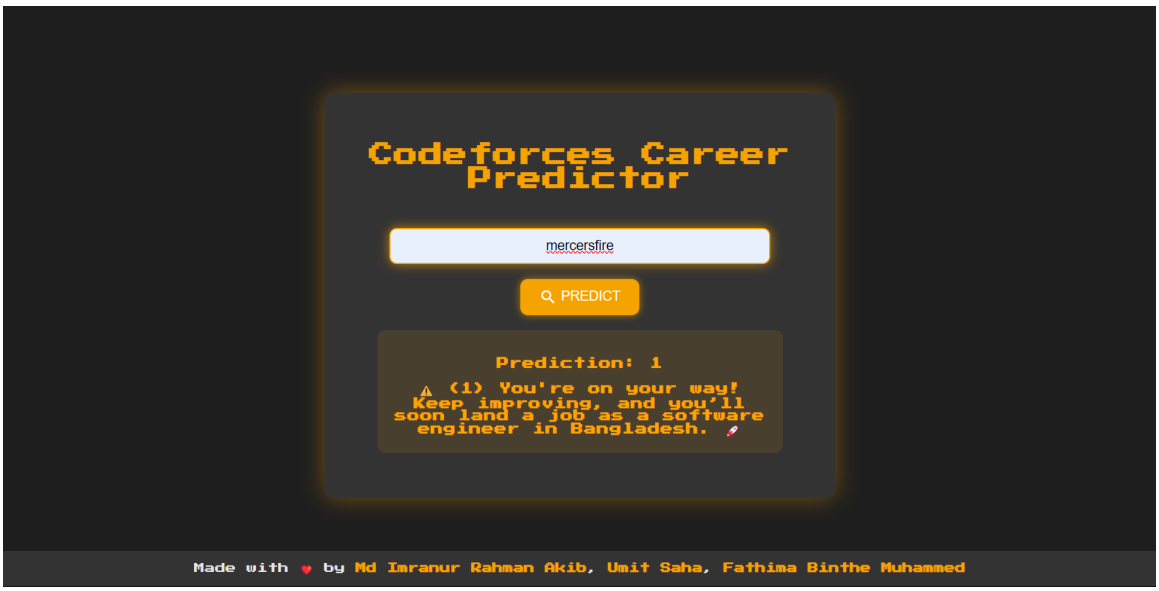

Figure 4.12: Level 1

Analysis: Although this user has a lower rating compared to others, their regular contest participation reflects a commitment to improving their skills. The model identifies them as someone on the right track, indicating that with continued effort and practice, they could further enhance their career prospects.

4.4.3 User 3: "umitsaha"

- **Highest Rating:** 1847
- **Problems Solved:** 2616
- **Contests Participated:** 200
- **Predicted Career Status:** 2 (Strong Candidate)

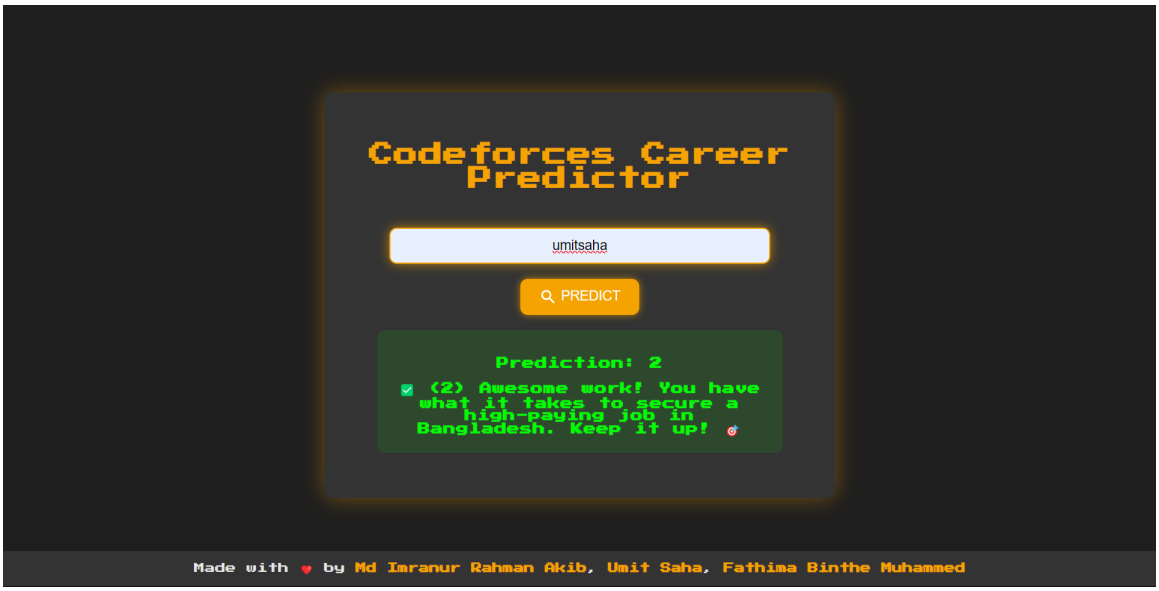

Figure 4.13: Level 2

Analysis: Despite having a moderate rating, this user has shown strong consistency in problem-solving and active participation in contests. Their persistence and engagement suggest significant potential, leading the model to classify them as a strong candidate for competitive job roles.

4.4.4 User 4: "monna4335"

- **Highest Rating:** 2010
- **Problems Solved:** 4081

- **Contests Participated:** 314
- **Predicted Career Status:** 3 (Highly Competitive)

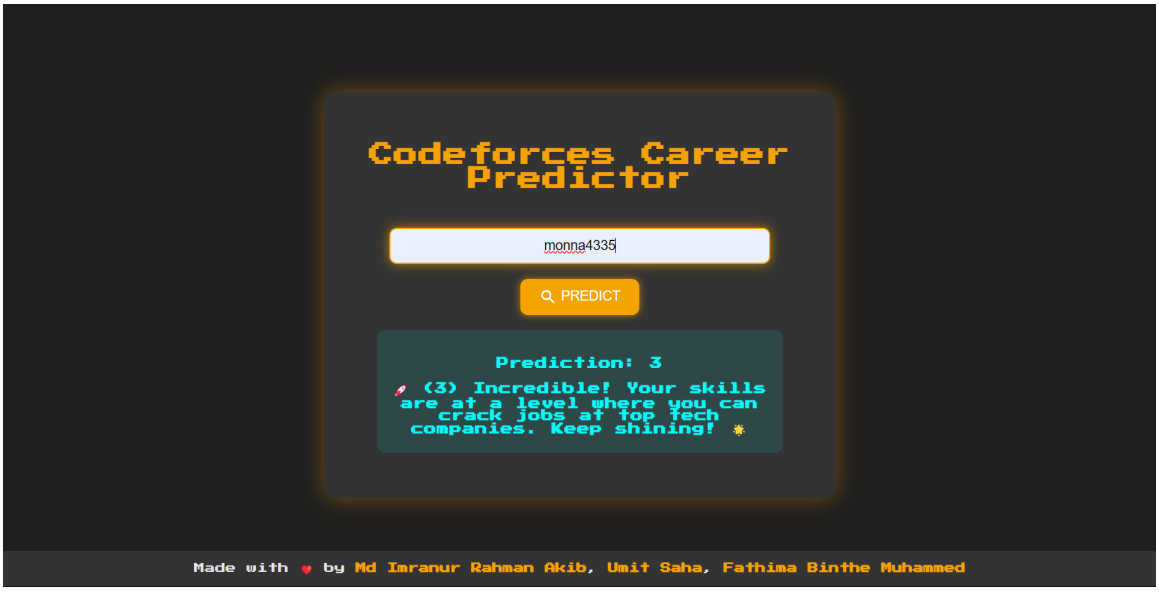

Figure 4.14: Level 3

Analysis: This user has an impressive competitive programming profile, with a high rating and extensive problem-solving experience. Given their strong track record, the model predicts that they have the potential to secure positions in top-tier tech companies.

These case studies illustrate how the model evaluates various performance indicators to assess career potential. By analyzing user data, it offers valuable insights into a programmer's growth trajectory and possible career opportunities in the tech industry.

4.5 Limitations and Challenges

While the model has shown promising results, there are several challenges and limitations that need to be considered:

Data Quality and Availability: The accuracy of predictions depends on the quality of data retrieved from the Codeforces API. However, this data is not always complete or fully reliable. Instances of missing or incomplete information can affect the model's ability to provide precise career assessments.

Feature Selection Constraints: The model primarily relies on competitive programming metrics such as ratings, problem-solving history, and contest participation. However, these factors alone do not fully capture an individual's professional capabilities. Other aspects like academic background, work experience, and personal projects could offer a more well-rounded prediction if included.

Potential Overfitting: While the model performs well on the test data, there is a risk of overfitting, particularly with complex algorithms like Random Forest. This means the model may adapt too closely to the training data and struggle to generalize effectively when analyzing new users.

Bias in Data Interpretation: Success in competitive programming does not always translate directly into workplace performance. Some individuals may excel in coding contests but lack key professional skills such as teamwork, communication, or real-world problem-solving experience. The model does not currently account for these factors, which are crucial for career success.

Scalability Challenges: Since the model is trained specifically on Codeforces data, its applicability to other competitive programming platforms or real-world hiring processes may be limited. Expanding its scope to include data from platforms like LeetCode, AtCoder, or even professional networks such as LinkedIn could improve its versatility and effectiveness.

Despite these limitations, the model provides a data-driven approach to assessing the career potential of competitive programmers. It offers valuable insights that can help individuals refine their skills and assist companies in identifying strong technical talent more efficiently.

CHAPTER V

Conclusion and Future Work

5.1 Key Takeaways

This study set out to explore how competitive programming activity on Codeforces can be used to predict career outcomes using machine learning. By gathering user data through the Codeforces API and extracting meaningful features, multiple classification models—including Random Forest, SVM, and KNN—were tested. Among these, the Random Forest classifier performed the best and was ultimately chosen for deployment. The results highlighted that factors such as contest participation frequency, problem-solving diversity, and rating trends play a significant role in predicting career trajectories.

To make these insights accessible, a Flask-based web application was developed, allowing users to input their Codeforces handle and receive real-time career predictions. The application was deployed via Render.com and designed with a user-friendly interface to ensure easy navigation.

These findings reinforce the idea that engagement in competitive programming can be a strong indicator of career potential, demonstrating the increasing relevance of AI-powered tools in career guidance and hiring decisions.

5.2 Future Improvements

To enhance the accuracy and usability of the system, several improvements are proposed:

Expanding Data Sources – Integrating data from platforms such as

LeetCode, AtCoder, and CodeChef would create a more comprehensive dataset, leading to improved prediction accuracy.

Incorporating Additional Career Indicators – Future iterations could include LinkedIn profiles, GitHub activity, and internship experiences to provide a more well-rounded career assessment.

Exploring Advanced Deep Learning Models – Using neural networks like LSTMs and Transformers could help identify more complex patterns in user performance over time.

Implementing a User Feedback System – Allowing users to verify or adjust their predicted career status would help refine the model's accuracy.

Enhancing Mobile Accessibility – Optimizing the interface for mobile devices would improve user engagement and accessibility.

Scalability Through Cloud Services – Transitioning from Render.com to cloud-based services like AWS Lambda, Google Cloud Functions, or Azure would ensure better performance and scalability as user traffic increases.

5.3 Final Thoughts

This project demonstrates the potential of machine learning in bridging the gap between competitive programming and career outcomes. By leveraging AI, it provides an insightful tool for both aspiring software engineers and recruiters, aiding in data-driven career planning and hiring decisions.

While the current system delivers meaningful predictions, there is ample opportunity for further development. Expanding the scope of career factors, improving model scalability, and refining the user experience will make this tool even more valuable in guiding programmers toward successful careers.

References

[1] Kamran Abid, Naeem Aslam, Muhammad Fuzail, Muhammad Sajid Maqbool, Kainat Sajid, et al. An efficient deep learning approach for prediction of student performance using neural network. *VFAST Transactions on Software Engineering*, 11(4):67–79, 2023.

[2] Ammar Alnahhas and Nour Mourtada. Predicting the performance of contestants in competitive programming using machine learning techniques. *Olympiads in Informatics*, 14:3–20, 2020.

[3] Jauhar Arifin and Riza Satria Perdana. Ugrade: Autograder for competitive programming using contestant pc as worker. In *2019 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6. IEEE, 2019.

[4] Piroska Biró and Tamás Kádek. Automatic evaluation of programming tasks at the university of debrecen. In *INTED2020 Proceedings*, pages 3522–3527. IATED, 2020.

[5] Catarina Félix and Sónia Rolland Sobral. Predicting students' performance using survey data. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1017–1023. IEEE, 2020.

[6] Chowdhury Md Intisar, Yutaka Watanobe, Manoj Poudel, and Subhash Bhalla. Classification of programming problems based on topic modeling. In *Proceedings of the 2019 7th International Conference on Information and Education Technology*, pages 275–283, 2019.

[7] Seonghoon Jang and Seung-Jung Shin. Machine learning-based programming analysis model proposal: Based on user behavioral analysis. *International journal of advanced smart convergence*, 9(4):179–183, 2020.

[8] Toshiyasu Kato, Yasushi Kambayashi, Yuki Terawaki, and Yasushi Kodama. Estimating grades from students' behaviors in programming exercises using deep learning. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1115–1119. IEEE, 2017.

[9] Toshiyasu Kato, Yasushi Kambayashi, Yuki Terawaki, and Yasushi Kodama. Analysis of students' behaviors in programming exercises using deep learning. In *Smart Education and e-Learning 2017 4*, pages 38–47. Springer, 2018.

[10] WooJeong Kim, Soyoung Rhim, John YJ Choi, and Kyungsik Han. Modeling learners' programming skills and question levels through machine learning. In *HCI International 2020–Late Breaking Posters: 22nd International Conference, HCII 2020, Copenhagen, Denmark, July 19–24, 2020, Proceedings, Part II 22*, pages 281–288. Springer, 2020.

[11] Yong Liu, Kai Tian, Haifeng Wang, Hengyuan Liu, Yonghao Wu, and Xiang Chen. Data-driven based student programming competition award prediction via machine learning models. In *2021 16th International Conference on Computer Science & Education (ICCSE)*, pages 463–468. IEEE, 2021.

[12] Wenli Looi. Analysis of code submissions in competitive programming contests.

[13] Md Mahbubur Rahman, Badhan Chandra Das, Al Amin Biswas, and Md Musfique Anwar. Predicting participants' performance in programming contests using deep learning techniques. In *International Conference on Hybrid Intelligent Systems*, pages 166–176. Springer, 2022.

[14] Rahul and Rahul Katarya. Deep auto encoder based on a transient search capsule network for student performance prediction. *Multimedia Tools and Applications*, 82(15):23427–23451, 2023.

[15] Miguel A Rubio. Automated prediction of novice programmer performance using programming trajectories. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II 21*, pages 268–272. Springer, 2020.

[16] Guohua Shen, Sien Yang, Zhiqiu Huang, Yaoshen Yu, and Xin Li. The prediction of programming performance using student profiles. *Education and Information Technologies*, 28(1):725–740, 2023.

[17] Bin Xu, Sheng Yan, Xin Jiang, and Shaoge Feng. Scfh: A student analysis model to identify students' programming levels in online judge systems. *Symmetry*, 12(4):601, 2020.

[18] Kevin KF Yuen, Dennis YW Liu, and Hong Va Leong. Competitive programming in computational thinking and problem solving education. *Computer Applications in Engineering Education*, 31(4):850–866, 2023.

[19] Fanghui Zha and Yong Wang. Correlation analysis of subject competition and programming ability for novice programmers. In *2021 7th International Symposium on System and Software Reliability (ISSSR)*, pages 25–31. IEEE, 2021.