SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

# Human Resource Management: Predicting Employee Promotions Using Machine Learning

## Project Flow:

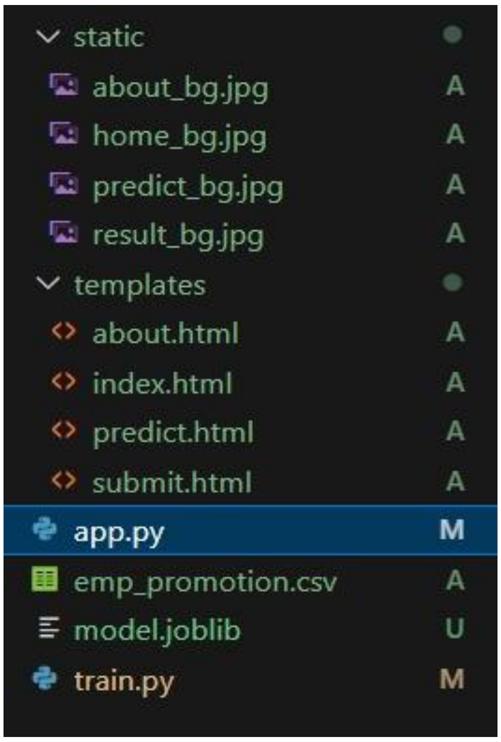
1. User interacts with the UI to enter the input.
2. Entered input is analysed by the model which is integrated.
3. Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities listed below,
4. Define Problem / Problem Understanding
   a. Specify the business problem
   b. Business requirements
   c. Literature Survey
   d. Social or Business Impact.
5. Data Collection &Preparation
   a. Collect the dataset
   b. Data Preparation
6. Exploratory Data Analysis
   a. Descriptive statistical
   b. Visual Analysis
7. Model Building
   a. Training the model in multiple algorithms
   b. Testing the model
8. Performance Testing & Hyperparameter Tuning
   a. Testing model with multiple evaluation metrics
   b. Comparing model accuracy before & after applying hyperparameter tuning
9. Model Deployment
   a. Save the best model
   b. Integrate with Web Framework
10. Project Demonstration & Documentation
    a. Record explanation Video for project end to end solution
    b. Project Documentation-Step by step project development procedure

## Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

1. ML Concepts
2. Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
3. Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
4. KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
5. Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
6. Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model- evaluation-error-metrics/
7. Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

## Project Structure:



```
∨ static                 ●
    about_bg.jpg         A
    home_bg.jpg          A
    predict_bg.jpg       A
    result_bg.jpg        A
∨ templates              ●
    about.html           A
    index.html           A
    predict.html         A
    submit.html          A
  app.py                 M
  emp_promotion.csv      A
  model.joblib           U
  train.py               M
```

1. We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
2. Dtc_model.pkl is our saved model. Further we will use this model for flask integration.
3. Data Folder contains the Dataset used
4. The Notebook file contains procedure for building the model.

# Human Resource Management: Predicting Employee Promotions Using Machine Learning

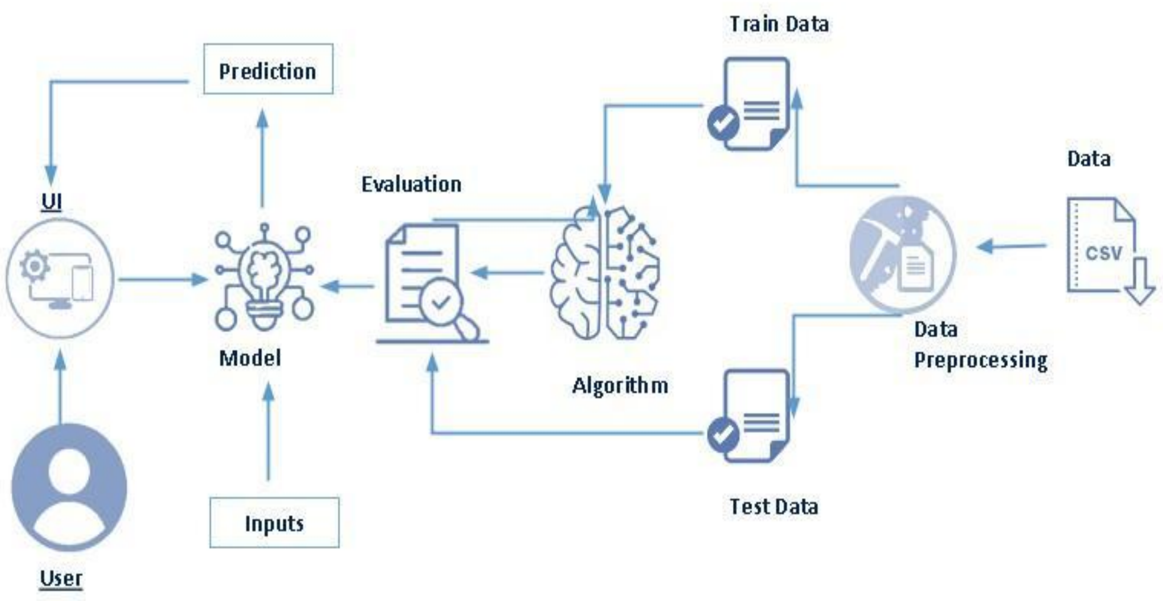
Employee Promotion Prediction Using Machine Learning involves developing a model to forecast the likelihood of employees being promoted within an organization based on various factors such as performance metrics, tenure, skills, and feedback. This project aims to enhance workforce management strategies by identifying high-potential employees deserving of advancement opportunities, thereby fostering employee engagement, retention, and organizational growth.

**Scenario 1**: In a large corporation, HR faces challenges in identifying top performers suitable for promotion due to the sheer volume of employees. By implementing a machine learning model, HR can efficiently analyze employee data to pinpoint individuals demonstrating exceptional capabilities and potential for advancement, streamlining the promotion process and ensuring deserving employees are recognized.

**Scenario 2**: A rapidly expanding startup wants to establish a fair and transparent promotion process to retain talent and incentivize growth. By leveraging machine learning algorithms, the company can assess various criteria, including project contributions, skill development, and leadership qualities, to predict which employees are most likely to thrive in higher roles, fostering a culture of meritocracy and career progression.

**Scenario 3:** In a competitive industry where talent retention is critical, a company seeks to proactively identify and nurture high-performing employees to prevent attrition. By deploying a machine learning solution, the organization can identify individuals with the potential for promotion, providing them with targeted development opportunities and career paths tailored to their strengths, fostering loyalty and commitment among top talent.

## Technical Architecture:



## Abstract

Human Resource Management (HRM) is a critical component of organizational strategy, focusing on talent acquisition, development, and retention. The objective of this project is to create a robust, data-driven system for predicting employee promotions, thereby enhancing the precision, fairness, and transparency of career progression decisions. Traditional promotion methods often rely on subjective judgments, leading to inherent biases and inconsistencies. This project utilizes historical HR data, encompassing performance metrics, demographic attributes, and professional experience, to develop a predictive machine learning model.

The methodology includes a comprehensive data preprocessing pipeline (handling missing values, outlier capping, categorical encoding, and SMOTE for class imbalance) and the comparative evaluation of multiple classification algorithms: Decision Tree, Random Forest, K-Nearest Neighbors (KNN), and XGBoost. The Random Forest Classifier was selected as the optimal model based on rigorous cross-validation and evaluation metrics, achieving an accuracy of **95%**. The best-performing model is seamlessly integrated into an interactive web application built using the **Flask** framework, providing HR managers with a real-time decision support tool. This project not only solves a critical business problem by streamlining the promotion cycle but also provides a complete, end-to-end demonstration of practical machine learning deployment.

## Milestone 1: Project Objectives

The primary objective of this project is to develop a machine learning model capable of predicting whether an employee is likely to receive a promotion based on historical HR data. In achieving this, the project aims to provide both technical and conceptual understanding of the end-to-end machine learning workflow and its real-world application in Human Resource Management.

### 1.1. Problem Identification and Classification

The first objective is to clearly define the problem and classify it accurately. The target variable, **is_promoted**, is binary (1 = Promoted, 0 = Not Promoted). Therefore, this project is fundamentally a **Binary Classification** problem.

### 1.2. Data Preprocessing and Cleaning

- To learn and apply advanced preprocessing techniques (imputation, encoding, scaling) to prepare the raw dataset for effective model training.
- To implement techniques for detecting and managing data anomalies, specifically handling missing values, identifying outliers, and filtering illogical "negative data."

### 1.3. Data Analysis and Visualization

- To gain actionable insights from the dataset by performing comprehensive **Exploratory Data Analysis (EDA)**.
- To use data visualization (plots, charts, heatmaps) to identify patterns, correlations, and relationships between employee features and promotion status.

### 1.4. Model Selection and Algorithm Application

- To implement and comparatively evaluate multiple robust machine learning algorithms, including **Decision Tree, Random Forest, KNN, and XGBoost**, to determine which model yields the highest predictive performance.
- To evaluate model performance using critical metrics such as Accuracy, Precision, Recall, and F1-Score, specifically focusing on the performance of the minority class (Promoted).

### 1.5. Handling Outliers and Data Transformation

- To apply the **Interquartile Range (IQR) method** and the **capping technique** to mitigate the disruptive influence of extreme values (outliers) in numerical features.

### 1.6. Web Application Development using Flask

- To integrate the final, optimized machine learning model into a **Flask-based web application** that allows HR managers to input employee data and receive instant, real-time promotion predictions through an interactive user interface.

## Milestone 2. Project Flow

The project follows a systematic end-to-end process, ensuring that the development is modular, reproducible, and aligns with standard MLOps principles. The flow is divided into five main stages, commencing from data acquisition and culminating in a deployed web application.

### 2.1. User Interaction and System Response

The system operates as a decision support tool with the following interaction cycle:

1. **Input Submission:** The user (HR manager/analyst) accesses the web interface and inputs an employee's detailed attributes (e.g., training score, performance rating).
2. **Model Analysis:** The integrated machine learning model receives the preprocessed input and analyzes it based on the patterns learned from the historical dataset.
3. **Prediction Output:** The model generates a binary prediction (Promoted or Not Promoted), which is then immediately showcased on the UI in a clear and visual manner.

### 2.2. Key Project Stages

| Stage | Activities | Key Techniques |
|---|---|---|
| **Data Collection** | Collect the raw dataset (e.g., from Kaggle). | CSV/Excel Loading, Initial Data Inspection |
| **Visualizing & Analysis** | Univariate, Multivariate, and Descriptive Analysis. | Box Plots, Histograms, Correlation Heatmaps |
| **Data Pre-processing** | Drop redundant features, handle null values, remove negative data, cap outliers, encode categorical data, balance imbalanced data (SMOTE), and split data. | Mode Imputation, Capping (IQR), Label Encoding, SMOTE, train_test_split |
| **Model Building** | Initialize, train, test, and compare multiple classification models. | Decision Tree, Random Forest, KNN, XGBoost, Confusion Matrix, Classification Report |
| **Application Building** | Design HTML pages, write the Python Flask backend, integrate the saved model, and run the final application. | HTML, Flask Framework, pickle (or joblib) |

## Milestone 3. Pre-requisites

To successfully develop this project, a foundational development environment and specific Python packages must be installed and configured.

### 3.1. Development Environment Setup

**Anaconda Navigator:** A free and open-source distribution of Python specifically tailored for data science and machine learning. It simplifies package and environment management using **Conda**.

- **Tools Used:** The project utilizes **Jupyter Notebook** for interactive EDA and model prototyping, and **Spyder** (or Visual Studio Code) for developing the structured Python application code (app.py).

### 3.2. Essential Python Packages

The following packages are mandatory for data manipulation, modeling, and deployment: | Package | Purpose in Project | | :--- | :--- | | **Pandas** | Data analysis and manipulation (reading CSV, handling DataFrames). | | **NumPy** | Fundamental package for numerical operations and array processing. | | **Matplotlib / Seaborn** | Data visualization for EDA (creating plots, graphs, and statistical charts). | | **Scikit-learn (sklearn)** | Core machine learning library for model selection, preprocessing, and evaluation (Decision Tree, KNN, Metrics). | | **XGBoost** | High-performance implementation of Gradient Boosting algorithm. | | **imbalanced-learn (imblearn)** | Library required for handling imbalanced datasets (specifically the **SMOTE** algorithm). | | **Flask** | Lightweight Python Web Framework for deploying the model as a web application. | | **pickle** | Serialization module used to save and load the trained machine learning model (model.pkl). |

## Milestone 4. Prior Knowledge

A strong theoretical background in the following concepts is essential for navigating the complexities of the project:

### 4.1. Core Machine Learning Concepts

- **Supervised Learning:** The framework used in this project, where the model learns from labeled data (input features target outcome).
- **Classification:** The specific task type, involving the prediction of a categorical outcome (e.g., promoted/not promoted).
- **Ensemble Techniques:** Methods that combine multiple machine learning models to achieve better predictive performance than a single model (e.g., Random Forest and XGBoost).

### 4.2. Algorithmic Understanding

- **Decision Tree Classifier:** Understanding the process of recursive partitioning based on impurity measures.
- **Random Forest Classifier:** Knowledge of the **Bagging** (Bootstrap Aggregating) technique and how feature randomness contributes to variance reduction.
- **KNN (K-Nearest Neighbors):** Understanding distance metrics (Euclidean, Manhattan) and the concept of a "lazy learner."
- **XGBoost:** Knowledge of **Gradient Boosting** and how new models correct the errors (residuals) of previous sequential models.

### 4.3. Evaluation and Deployment Fundamentals

- **Evaluation Metrics:** Understanding the critical distinction between Accuracy (misleading on imbalanced data), Precision (minimizing False Positives), and Recall (minimizing False Negatives).

- **Flask Web Framework:** Basic knowledge of setting up application routes, handling HTTP methods (GET/POST), and rendering dynamic HTML templates.

## Milestone 5. Data Collection

The success of any predictive model hinges on the quality and relevance of the data. This section details the source and structure of the dataset used.

### Datasets

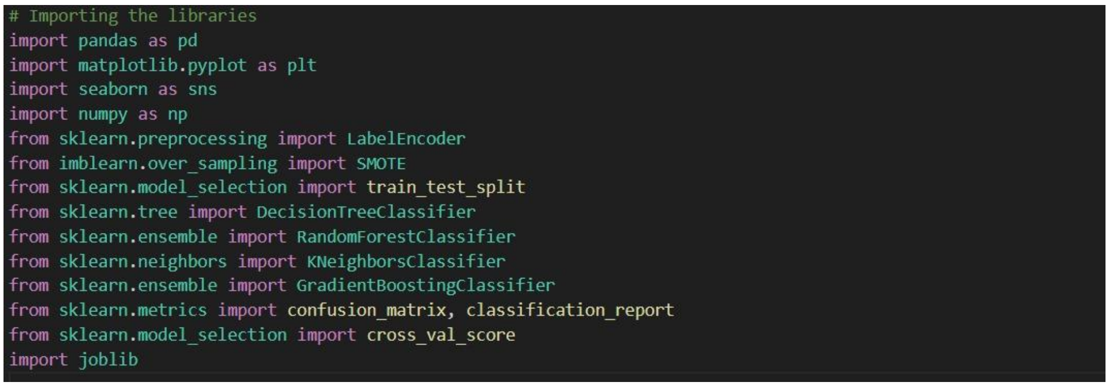
The dataset is obtained from popular open-source repositories for data science projects.

- **Source: Kaggle** (www.kaggle.com) or **UCI Repository** are commonly used platforms.
- **Dataset Format:** The data is typically provided in a **CSV** (Comma Separated Values) or **Excel** format.
- **Features:** The dataset contains historical records of employee attributes, including performance ratings, training scores, tenure, and demographics (as detailed in the Data Description section of the Appendix).
- **Link :** https://drive.google.com/file/d/14eQR1VWHwuomPaXdKuIkZEpSstvav8Db/view?usp=sharing

## Milestone 6. Visualising and Analysis the Data

Before any transformation, the raw data must be understood through visualization and statistical analysis. The visualization style **'fivethirtyeight'** is adopted for clarity and aesthetic appeal.

### 6.1. Import the Libraries



```
# Importing the libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score
import joblib
```

The necessary Python libraries are imported to set up the environment for data handling, visualization, and modeling.

Python

# Data Handling and Numerics

import pandas as pd

import numpy as np

# Visualization Libraries (using fivethirtyeight style)

import matplotlib.pyplot as plt

import seaborn as sns

plt.style.use('fivethirtyeight')

# Machine Learning Utilities

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

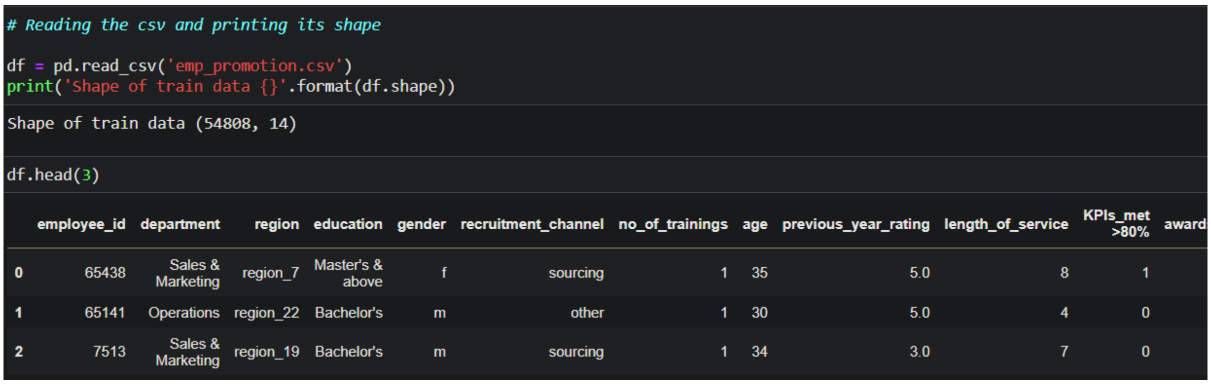
from imblearn.over_sampling import SMOTE

# Model Serialization

import pickle

## 6.2. Read the Dataset

The dataset, assumed to be in CSV format, is loaded into a Pandas DataFrame for efficient manipulation and analysis.



```
# Reading the csv and printing its shape

df = pd.read_csv('emp_promotion.csv')
print('Shape of train data {}'.format(df.shape))
```

```
Shape of train data (54808, 14)
```

```
df.head(3)
```

| | employee_id | department | region | education | gender | recruitment_channel | no_of_trainings | age | previous_year_rating | length_of_service | KPIs_met >80% | award |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 65438 | Sales & Marketing | region_7 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | |
| 1 | 65141 | Operations | region_22 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | |
| 2 | 7513 | Sales & Marketing | region_19 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | |

Python

# Reading the dataset

data = pd.read_csv('path_to_dataset/HR_Promotion_Data.csv')

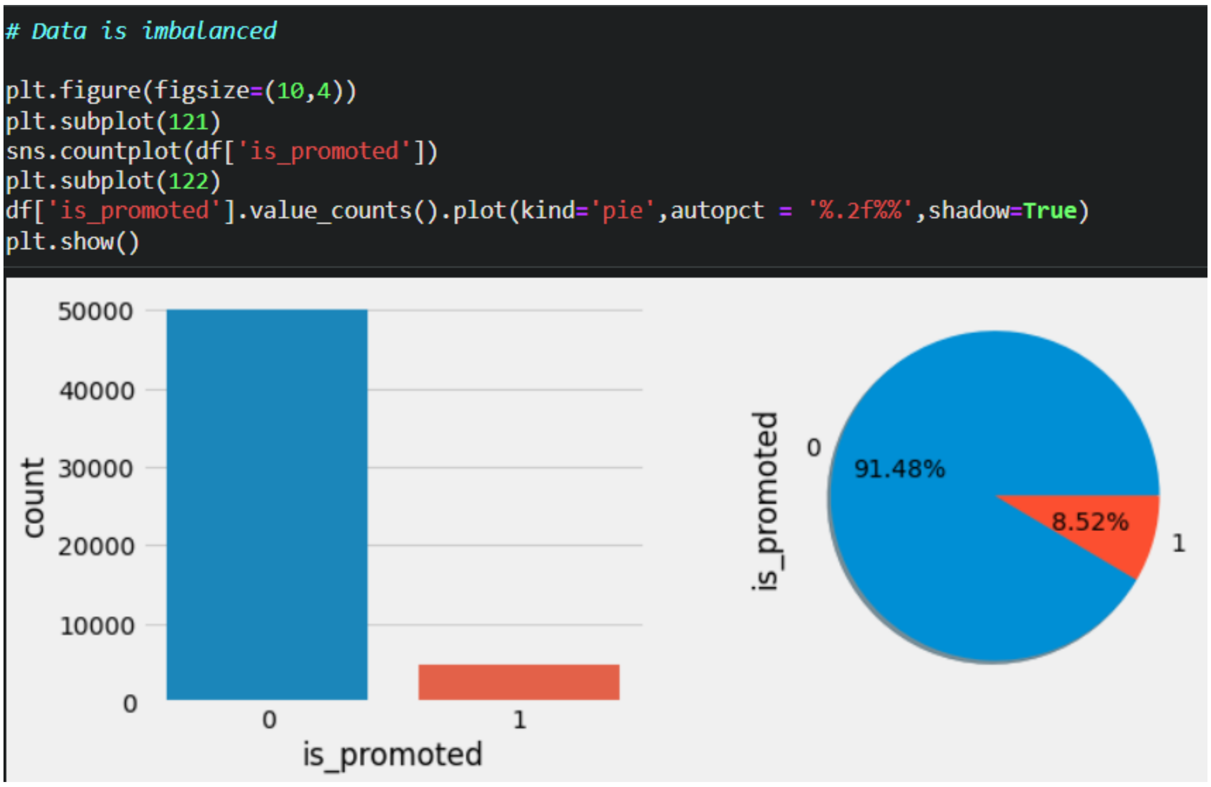
# Displaying the first few records to confirm successful load

print(data.head())

**6.3. Univariate Analysis**

Univariate analysis focuses on examining the distribution and characteristics of each single feature.

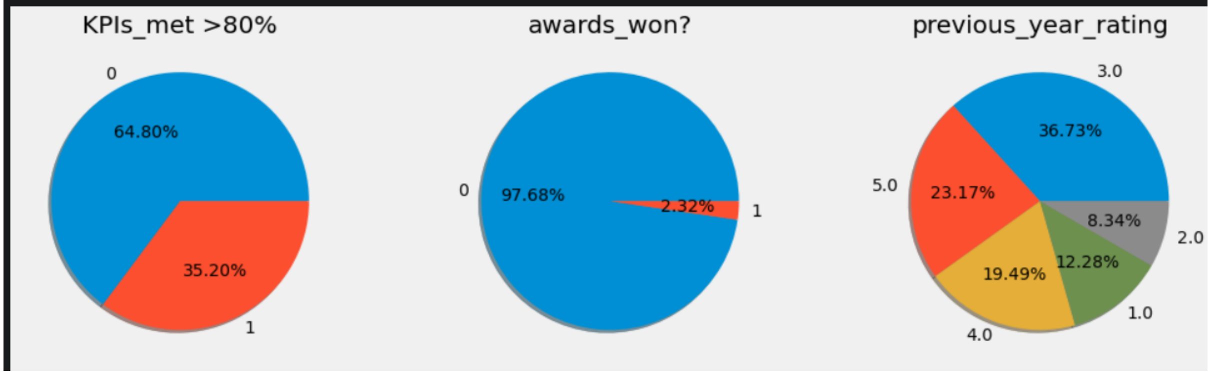
- **Target Variable Imbalance: Count plots** or **pie plots** on the target variable, is_promoted, revealed a severe class imbalance, with approximately **91% of employees not promoted**. This is a critical finding that necessitates using the SMOTE technique later in preprocessing.



```
# Data is imbalanced

plt.figure(figsize=(10,4))
plt.subplot(121)
sns.countplot(df['is_promoted'])
plt.subplot(122)
df['is_promoted'].value_counts().plot(kind='pie',autopct = '%.2f%%',shadow=True)
plt.show()
```

- **Categorical Features:** Analysis of features like awards_won (97.68% zero) and previous_year_rating (over 75% rated ) indicates highly skewed distributions.

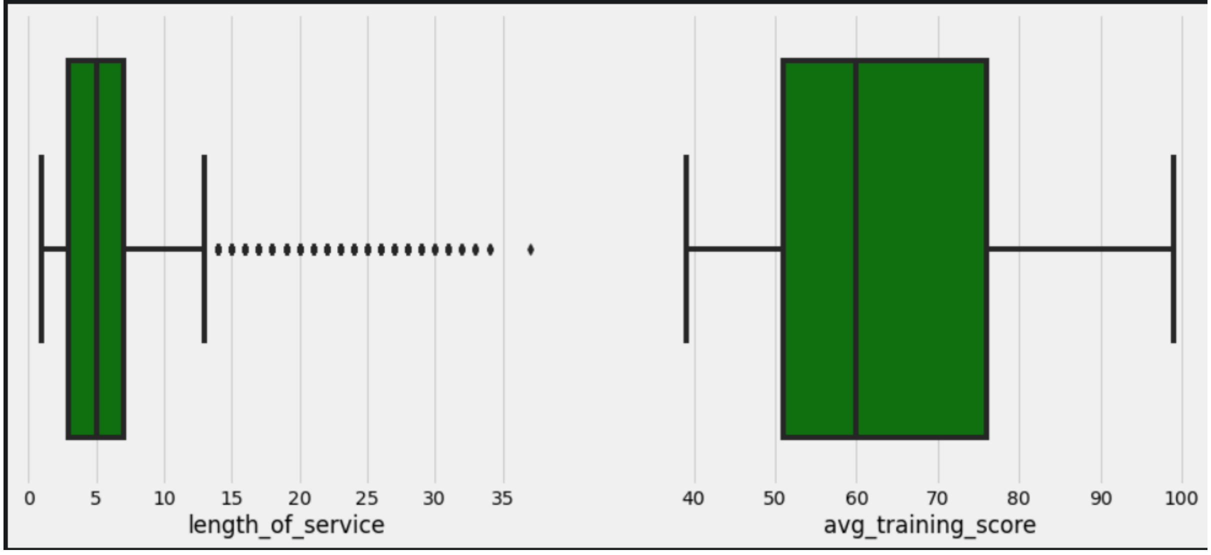
```
plt.figure(figsize=(16,10))
plt.subplot(231)
plt.axis('off')
plt.title('KPIs_met >80%')
df['KPIs_met >80%'].value_counts().plot(kind='pie',shadow=True,autopct = '%.2f%%')
plt.subplot(232)
plt.axis('off')
plt.title('awards_won?')
df['awards_won?'].value_counts().plot(kind='pie',shadow=True,autopct = '%.2f%%')
plt.subplot(233)
plt.axis('off')
plt.title('previous_year_rating')
df['previous_year_rating'].value_counts().plot(kind='pie',shadow=True,autopct = '%.2f%%')
plt.show()
```



- **Numerical Features (Box Plots): Box plots** applied to features like length_of_service clearly show the presence of numerous **outliers**, confirming the need for anomaly treatment via capping.

```
# Length of services column has outliers

plt.figure(figsize=(14,6))
plt.subplot(121)
sns.boxplot(df['length_of_service'],color='g')
plt.subplot(122)
sns.boxplot(df['avg_training_score'],color='g')
```

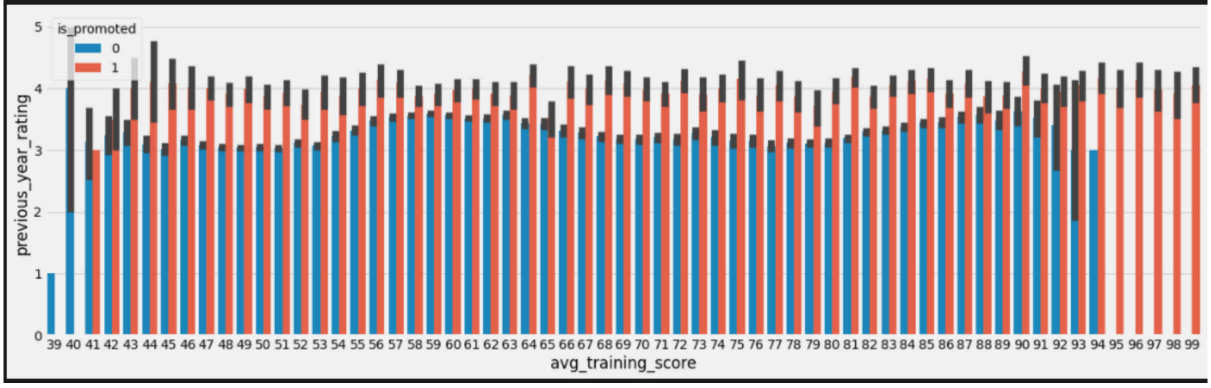
```
<AxesSubplot:xlabel='avg_training_score'>
```



## 6.4. Multivariate Analysis

```
""" From the below bar plot, we came to know that employee with training score > 95 & previous year rating > 3 got promoted. """

plt.figure(figsize=(20,6))
sns.barplot(df['avg_training_score'],df['previous_year_rating'],df['is_promoted'])
```

```
<AxesSubplot:xlabel='avg_training_score', ylabel='previous_year_rating'>
```

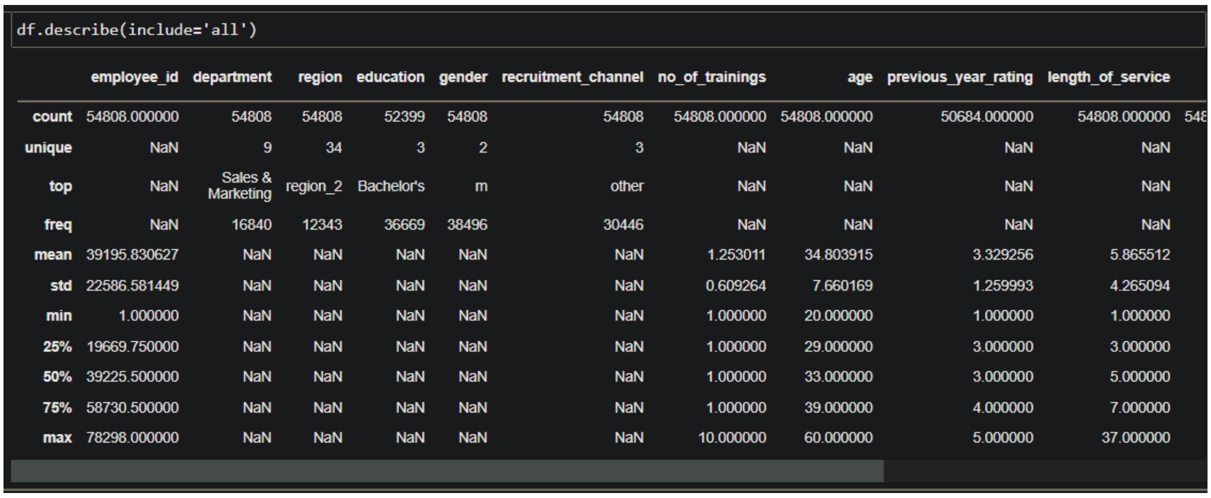


Multivariate analysis explores the relationship between two or more variables, which is vital for feature selection.

- **Correlation Matrix (Heatmap):** A heatmap visualizing the correlation matrix identifies the strength and direction of linear relationships. High positive correlation is often observed between performance features (previous_year_rating, average_training_score) and the target variable (is_promoted).
- **Bivariate Plots:** Plots showing the relationship between a numerical feature (e.g., average_training_score) and the binary target variable often reveal clear separability, justifying the feature's predictive power.

## 6.5. Descriptive Analysis

The Pandas describe() function provides a statistical summary of the dataset.



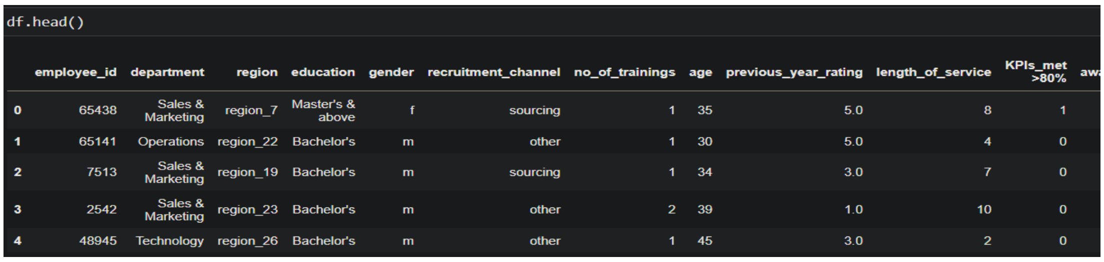
```
df.describe(include='all')
```

| | employee_id | department | region | education | gender | recruitment_channel | no_of_trainings | age | previous_year_rating | length_of_service | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 54808.000000 | 54808 | 54808 | 52399 | 54808 | 54808 | 54808.000000 | 54808.000000 | 50684.000000 | 54808.000000 | 548 |
| **unique** | NaN | 9 | 34 | 3 | 2 | 3 | NaN | NaN | NaN | NaN | |
| **top** | NaN | Sales & Marketing | region_2 | Bachelor's | m | other | NaN | NaN | NaN | NaN | |
| **freq** | NaN | 16840 | 12343 | 36669 | 38496 | 30446 | NaN | NaN | NaN | NaN | |
| **mean** | 39195.830627 | NaN | NaN | NaN | NaN | NaN | 1.253011 | 34.803915 | 3.329256 | 5.865512 | |
| **std** | 22586.581449 | NaN | NaN | NaN | NaN | NaN | 0.609264 | 7.660169 | 1.259993 | 4.265094 | |
| **min** | 1.000000 | NaN | NaN | NaN | NaN | NaN | 1.000000 | 20.000000 | 1.000000 | 1.000000 | |
| **25%** | 19669.750000 | NaN | NaN | NaN | NaN | NaN | 1.000000 | 29.000000 | 3.000000 | 3.000000 | |
| **50%** | 39225.500000 | NaN | NaN | NaN | NaN | NaN | 1.000000 | 33.000000 | 3.000000 | 5.000000 | |
| **75%** | 58730.500000 | NaN | NaN | NaN | NaN | NaN | 1.000000 | 39.000000 | 4.000000 | 7.000000 | |
| **max** | 78298.000000 | NaN | NaN | NaN | NaN | NaN | 10.000000 | 60.000000 | 5.000000 | 37.000000 | |

- **Numerical Summary:** Provides count, mean, standard deviation, minimum, maximum, and quartile values (, , ) for numerical columns. This helps in understanding the central tendency, spread, and the presence of extreme values.
- **Categorical Summary:** Provides unique counts, the most frequent value (top), and its frequency, crucial for identifying categories that dominate the dataset.

## Milestone 7. Data Pre-processing

Data preprocessing is performed to clean the dataset, ensure consistency, and transform features into a format suitable for machine learning algorithms.

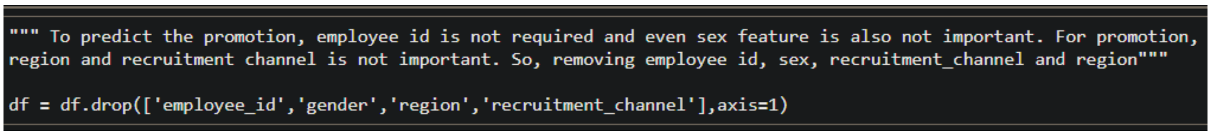


```
df.head()
```

| | employee_id | department | region | education | gender | recruitment_channel | no_of_trainings | age | previous_year_rating | length_of_service | KPIs_met >80% | aw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 65438 | Sales & Marketing | region_7 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | |
| 1 | 65141 | Operations | region_22 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | |
| 2 | 7513 | Sales & Marketing | region_19 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | |
| 3 | 2542 | Sales & Marketing | region_23 | Bachelor's | m | other | 2 | 39 | 1.0 | 10 | 0 | |
| 4 | 48945 | Technology | region_26 | Bachelor's | m | other | 1 | 45 | 3.0 | 2 | 0 | |

### 7.1. Drop Unwanted Features

Features that are irrelevant or potentially introduce bias are removed to simplify the model and improve generalization.



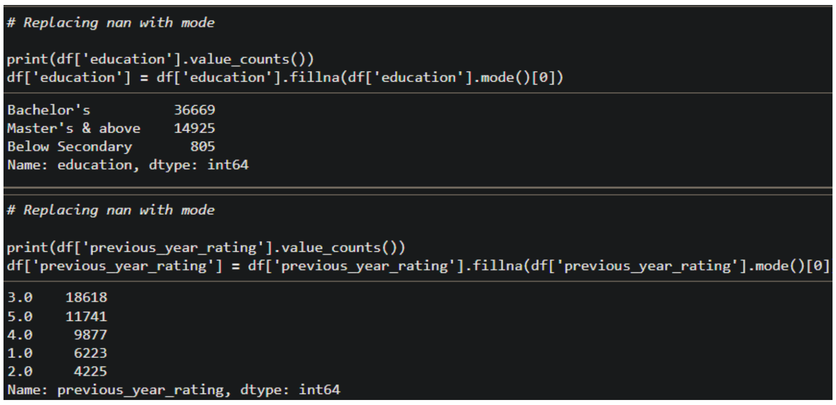
```
""" To predict the promotion, employee id is not required and even sex feature is also not important. For promotion,
region and recruitment channel is not important. So, removing employee id, sex, recruitment_channel and region"""

df = df.drop(['employee_id','gender','region','recruitment_channel'],axis=1)
```

- **Rationale:** Features like employee_id (unique identifier) offer no predictive value. Features like gender, region, and recruitment_channel are removed to prevent the model from learning based on subjective or unethical biases, focusing the prediction strictly on performance, experience, and training.

### 7.2. Checking for Null Values

Null values were found primarily in the education and previous_year_rating columns.

- **Method :** Since both are categorical/ordinal in nature, **Mode Imputation** is the preferred method, as it fills the missing spots with the most frequent value, preserving the overall distribution.



```
# Replacing nan with mode

print(df['education'].value_counts())
df['education'] = df['education'].fillna(df['education'].mode()[0])
```

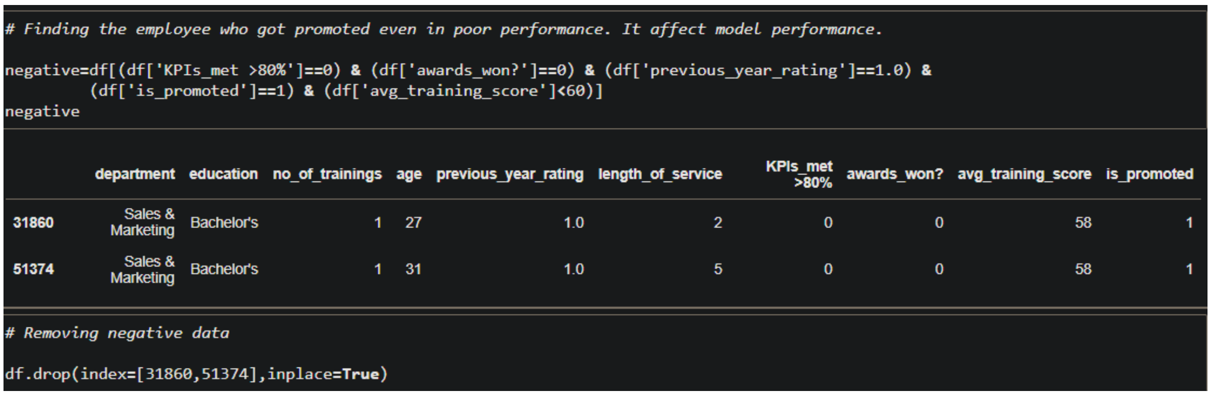
```
Bachelor's          36669
Master's & above    14925
Below Secondary       805
Name: education, dtype: int64
```

```
# Replacing nan with mode

print(df['previous_year_rating'].value_counts())
df['previous_year_rating'] = df['previous_year_rating'].fillna(df['previous_year_rating'].mode()[0]
```

```
3.0    18618
5.0    11741
4.0     9877
1.0     6223
2.0     4225
Name: previous_year_rating, dtype: int64
```

## 7.3. Remove Negative Data

Illogical or contradictory data points, referred to as "negative data" in the context of this project, are removed.



```
# Finding the employee who got promoted even in poor performance. It affect model performance.

negative=df[(df['KPIs_met >80%']==0) & (df['awards_won?']==0) & (df['previous_year_rating']==1.0) &
        (df['is_promoted']==1) & (df['avg_training_score']<60)]
negative
```

| | department | education | no_of_trainings | age | previous_year_rating | length_of_service | KPIs_met >80% | awards_won? | avg_training_score | is_promoted |
|---|---|---|---|---|---|---|---|---|---|---|
| **31860** | Sales & Marketing | Bachelor's | 1 | 27 | 1.0 | 2 | 0 | 0 | 58 | 1 |
| **51374** | Sales & Marketing | Bachelor's | 1 | 31 | 1.0 | 5 | 0 | 0 | 58 | 1 |

```
# Removing negative data

df.drop(index=[31860,51374],inplace=True)
```

- **Definition of Negative Data:** Records where an employee was **promoted** despite simultaneously having *no awards*, a *previous year rating of 1.0*, *KPIs less than 80%*, and an *average training score below 60*.
- **Rationale:** Such outliers in the performance-to-promotion mapping introduce noise that severely degrades model accuracy. Removing them ensures the model learns based on meritocratic patterns.

## 7.4. Handling Outliers

Outliers are handled using the statistical **Interquartile Range (IQR) method** and **Capping**.

- **IQR Calculation:** The fences for outlier detection are calculated as:
- **Capping Technique:** Extreme values are not removed (to avoid data loss) but are instead replaced with the respective Upper or Lower Bound. This **Winsorization** approach mitigates the influence of outliers while preserving the data size.

## 7.5. Handling Categorical Values

Categorical features are converted to numerical format using appropriate encoding techniques.

- **Feature Mapping (Ordinal Data):** Used for the **education** feature, where categories have a clear inherent order (e.g., Bachelor's 1, Master's 2, PhD 3). This preserves the magnitude of the ordered feature.
- **Label Encoding (Nominal Data):** Used for the **department** feature. The LabelEncoder assigns unique integers (0 to ) based on alphabetical order.

## 7.6. Handling Imbalanced Data

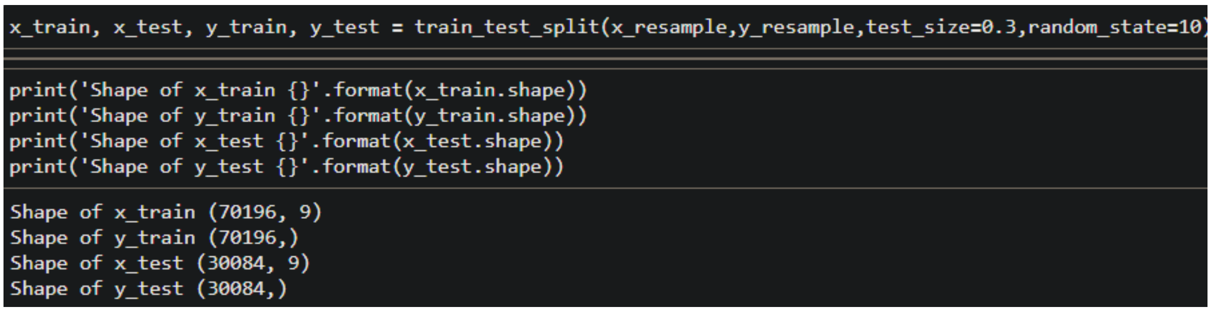
The severe class imbalance observed in the target variable (91% non-promoted) is addressed using the **SMOTE** technique.

- **SMOTE (Synthetic Minority Over-sampling Technique):** SMOTE works by synthesizing new, artificial data points for the minority class (Promoted) based on the feature space distance of existing minority samples. This results in a balanced dataset where the ratio of Promoted to Not Promoted classes is close to 1:1.
- **Impact:** Balancing the data prevents the model from being biased towards the majority class, which is crucial for accurately predicting the rare (but important) event of a promotion.

## 7.7. Splitting Data into Train and Test

The preprocessed and balanced dataset is split into two parts:

- **Ratio:** Typically, an training and testing split is used for model development.
- **Function:** The train_test_split() function from Scikit-learn is used, along with a specified random_state to ensure the split is reproducible.



```
x_train, x_test, y_train, y_test = train_test_split(x_resample,y_resample,test_size=0.3,random_state=10
```

```
print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_test.shape))
print('Shape of y_test {}'.format(y_test.shape))
```

```
Shape of x_train (70196, 9)
Shape of y_train (70196,)
Shape of x_test (30084, 9)
Shape of y_test (30084,)
```

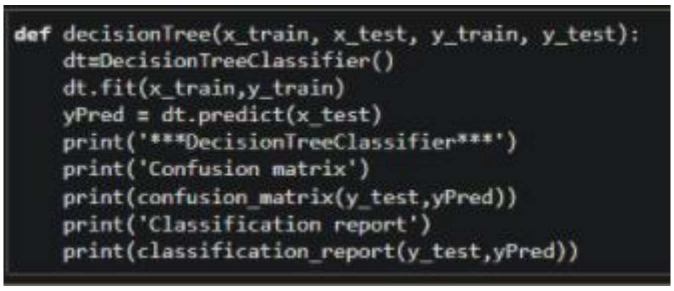
## Milestone 8. Model Building

Four distinct classification models are built and trained on the pre-processed data to identify the optimal algorithm for the task.

### 8.1. Decision Tree Model

**Core Principle:** The Decision Tree builds a set of hierarchical, non-linear rules to partition the data.

- **Implementation:** The DecisionTreeClassifier is initialized and fitted to the training data. The model is then evaluated using a **Confusion Matrix** and **Classification Report**.

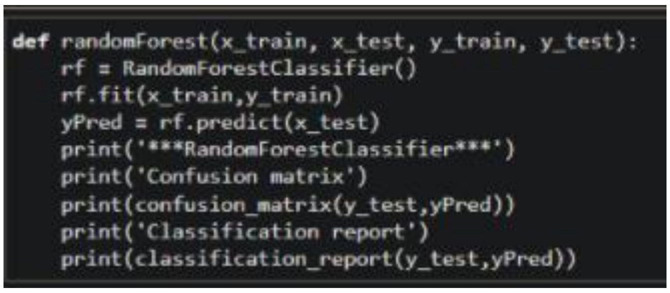


```
def decisionTree(x_train, x_test, y_train, y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train)
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

### 8.2. Random Forest Model

**Core Principle:** Random Forest uses the **Bagging** ensemble method. It trains hundreds of independent Decision Trees on random subsets of the data (bootstrapped samples) and features, combining their predictions via majority voting.

- **Advantage:** This parallel approach significantly reduces the **variance** (overfitting) inherent in single Decision Trees while maintaining high predictive power.
- **Implementation:** The RandomForestClassifier is utilized, which often provides a strong baseline performance.



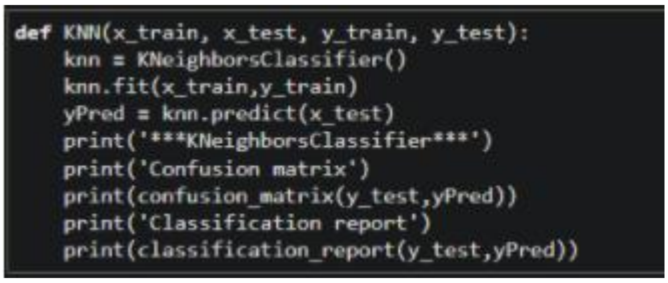
```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

### 8.3. KNN Model

**Core Principle:** K-Nearest Neighbors classifies a new data point by measuring its distance to all training samples and assigning the class of the closest neighbors.

- **Distance Metric:** The **Euclidean Distance** is commonly used.

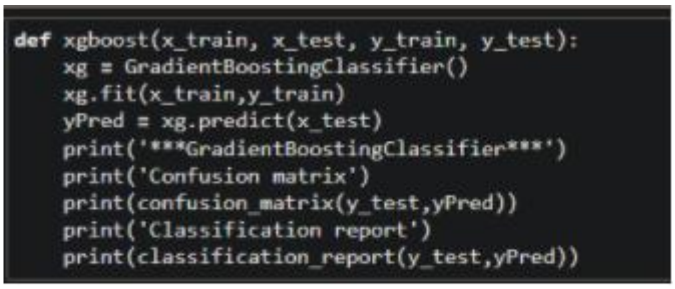
- **Implementation:** The KNeighborsClassifier is initialized. The selection of the optimal value is critical for balancing noise sensitivity and over-smoothing.



```
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

### 8.4. XGBoost Model

**Core Principle:** XGBoost (Extreme Gradient Boosting) is an optimized, sequential ensemble method. New trees are built to correct the **residuals** (errors) of the aggregate prediction made by all previous trees.



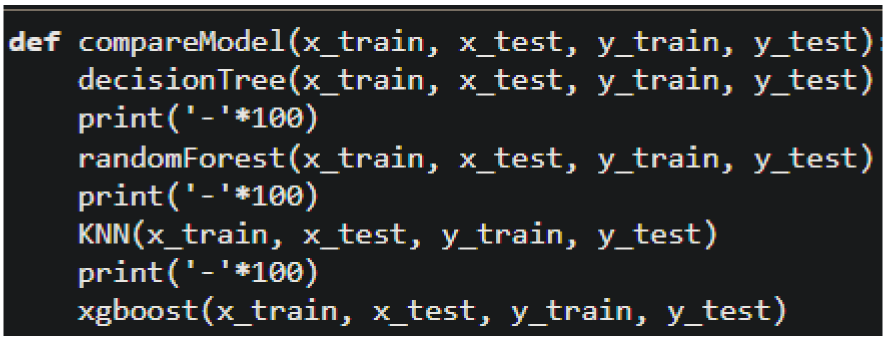
```
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

- **Objective Function:** The model minimizes a regularized objective function, balancing the training loss and the complexity of the model ().
- **Advantage:** The inclusion of regularization () helps prevent overfitting, making it highly robust for complex datasets.

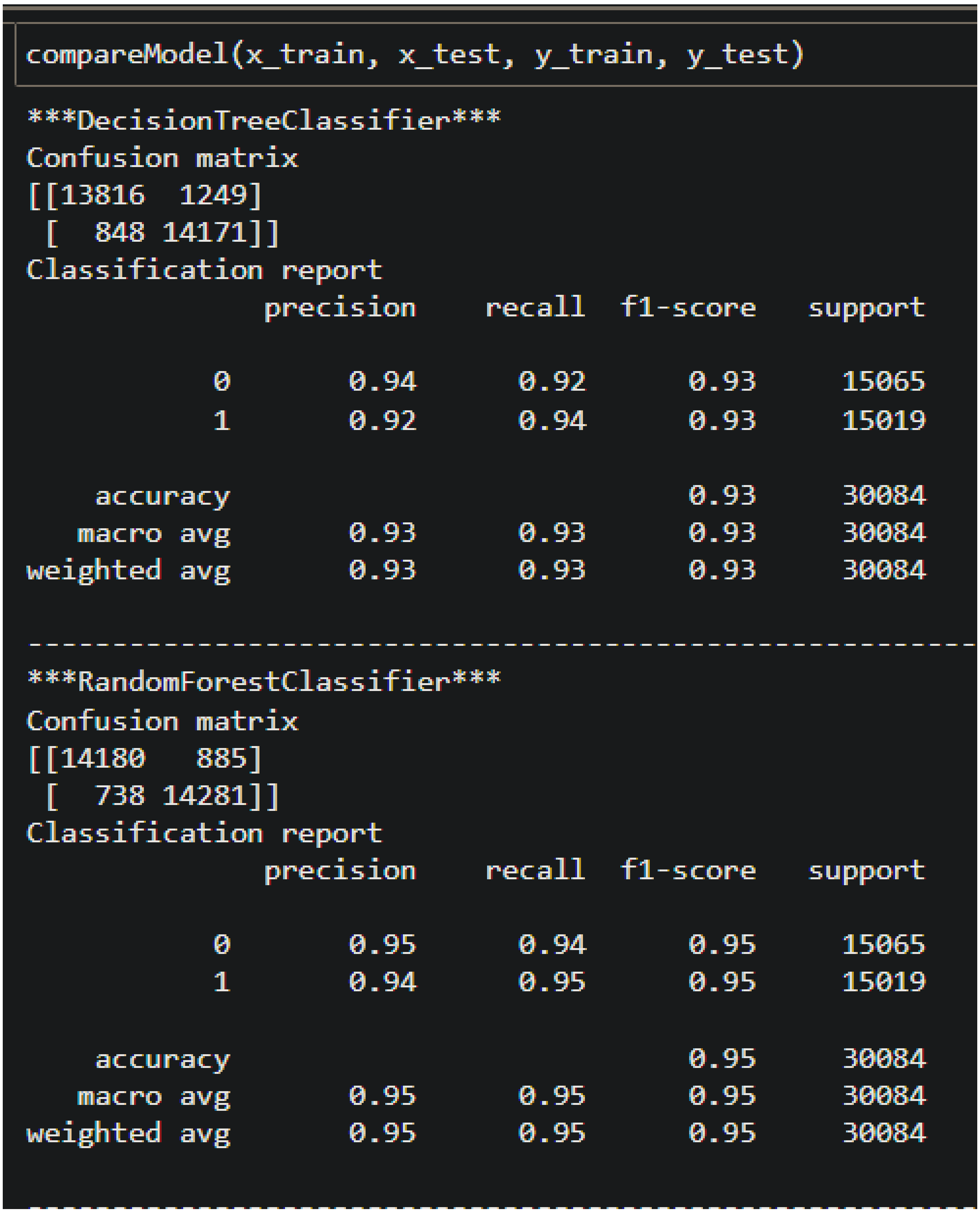
### 8.5. Compare the Model

A dedicated comparison function is run to assess the performance of all four models.

- **Selection Criterion:** The **Random Forest model** is selected as the best performer, achieving accuracy and showing the highest balance of **True Positives** and **True Negatives** in the Confusion Matrix.

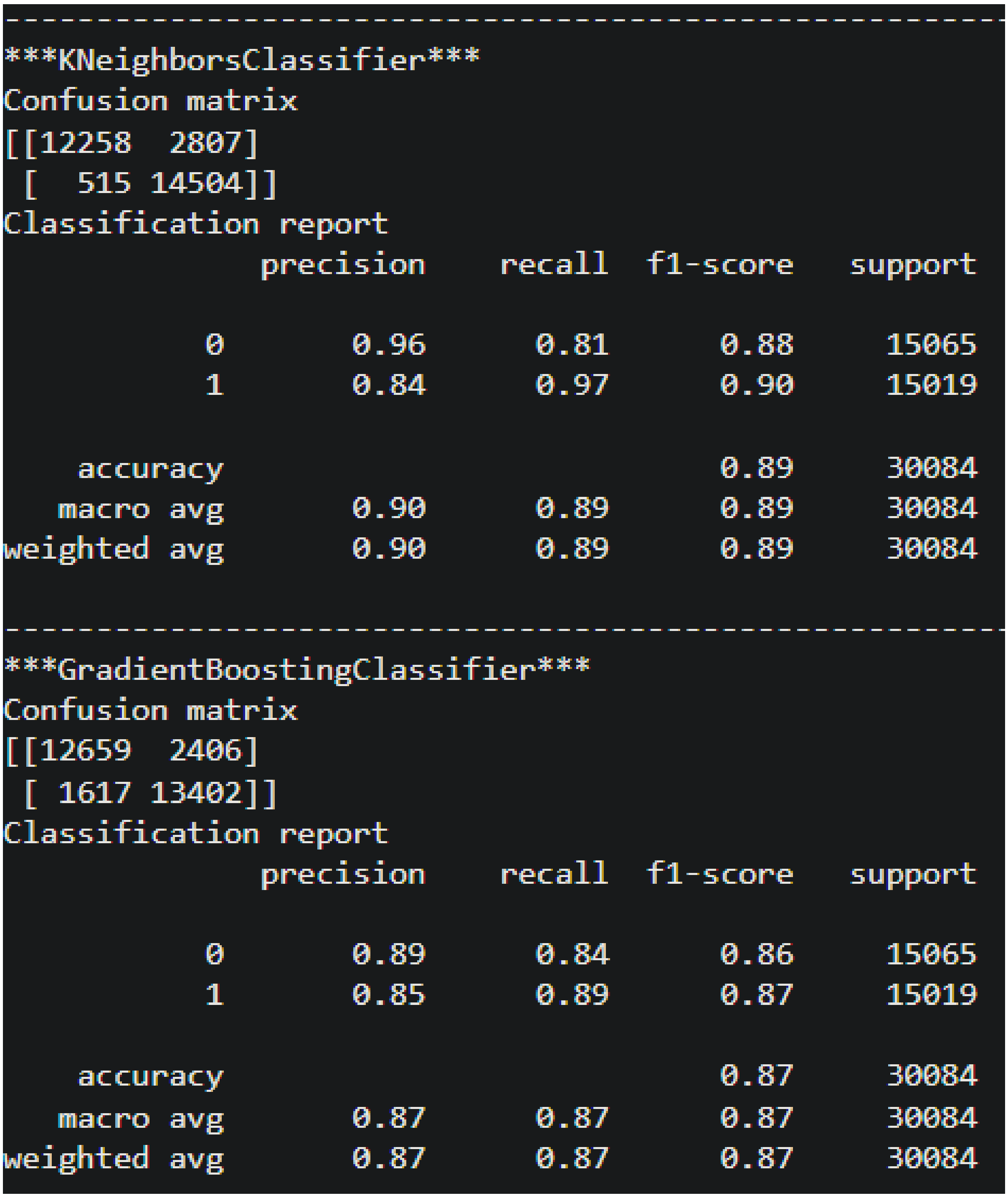


```
def compareModel(x_train, x_test, y_train, y_test)
    decisionTree(x_train, x_test, y_train, y_test)
    print('-'*100)
    randomForest(x_train, x_test, y_train, y_test)
    print('-'*100)
    KNN(x_train, x_test, y_train, y_test)
    print('-'*100)
    xgboost(x_train, x_test, y_train, y_test)
```



```
compareModel(x_train, x_test, y_train, y_test)
```

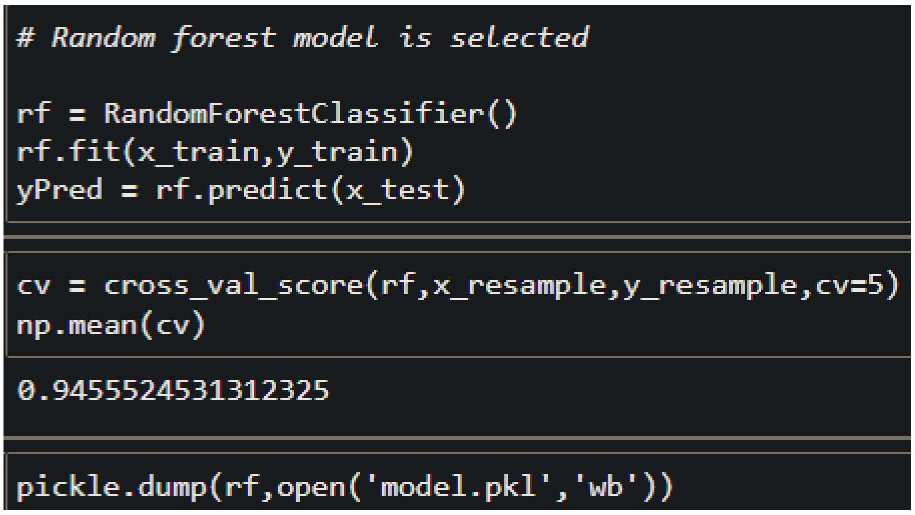
```
***DecisionTreeClassifier***
Confusion matrix
[[13816  1249]
 [  848 14171]]
Classification report
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     15065
           1       0.92      0.94      0.93     15019

    accuracy                           0.93     30084
   macro avg       0.93      0.93      0.93     30084
weighted avg       0.93      0.93      0.93     30084

--------------------------------------------------------
***RandomForestClassifier***
Confusion matrix
[[14180   885]
 [  738 14281]]
Classification report
              precision    recall  f1-score   support

           0       0.95      0.94      0.95     15065
           1       0.94      0.95      0.95     15019

    accuracy                           0.95     30084
   macro avg       0.95      0.95      0.95     30084
weighted avg       0.95      0.95      0.95     30084

--------------------------------------------------------
```



```
--------------------------------------------------------------
***KNeighborsClassifier***
Confusion matrix
[[12258  2807]
 [  515 14504]]
Classification report
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     15065
           1       0.84      0.97      0.90     15019

    accuracy                           0.89     30084
   macro avg       0.90      0.89      0.89     30084
weighted avg       0.90      0.89      0.89     30084

--------------------------------------------------------------
***GradientBoostingClassifier***
Confusion matrix
[[12659  2406]
 [ 1617 13402]]
Classification report
              precision    recall  f1-score   support

           0       0.89      0.84      0.86     15065
           1       0.85      0.89      0.87     15019

    accuracy                           0.87     30084
   macro avg       0.87      0.87      0.87     30084
weighted avg       0.87      0.87      0.87     30084
```

### 8.6. Evaluating Performance of the Model and Saving the Model

The best model's performance is finalized through rigorous evaluation, and the model is persisted for deployment.



```
# Random forest model is selected

rf = RandomForestClassifier()
rf.fit(x_train,y_train)
yPred = rf.predict(x_test)
```

```
cv = cross_val_score(rf,x_resample,y_resample,cv=5)
np.mean(cv)
```

```
0.9455524531312325
```

```
pickle.dump(rf,open('model.pkl','wb'))
```

- **Cross-Validation:** The cross_val_score function (with folds) is used on the **Random Forest model** to ensure its average performance is stable and reliable across different data subsets, confirming its generalization capability.
- **Model Saving:** The finalized **Random Forest model** is serialized and saved using the pickle.dump() function, creating the file **model.pkl**. This file will be loaded directly into the Flask application's memory.

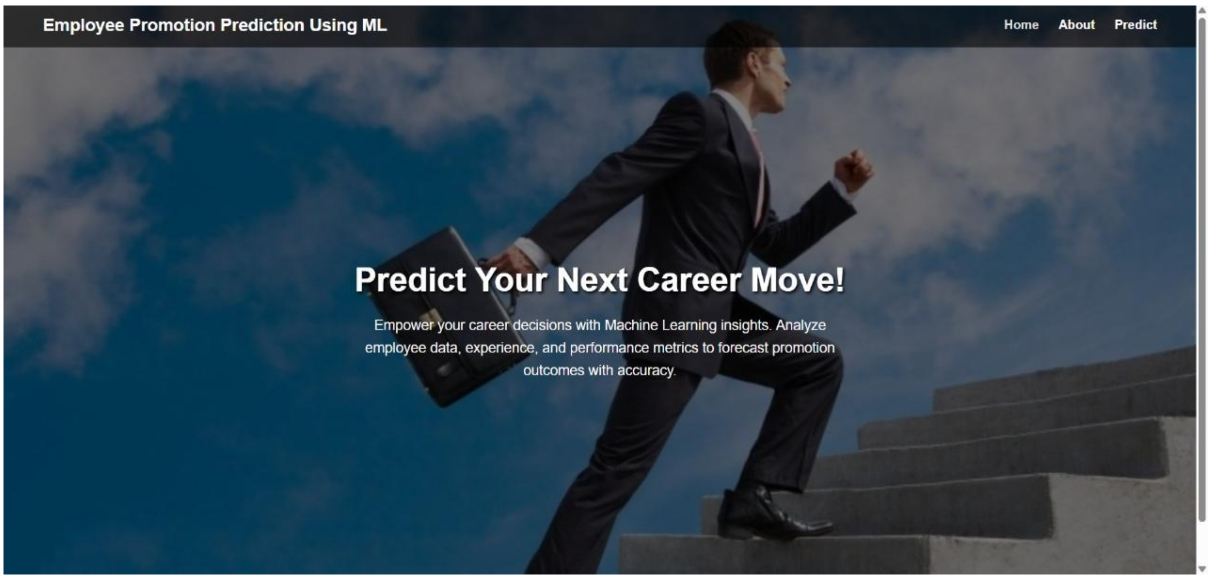
## Milestone 9. Application Building

The final stage involves deploying the model into an interactive web application using Python's Flask framework.

### 9.1. Building HTML Pages

The web interface is composed of four HTML pages stored in the templates folder, ensuring a structured user journey:

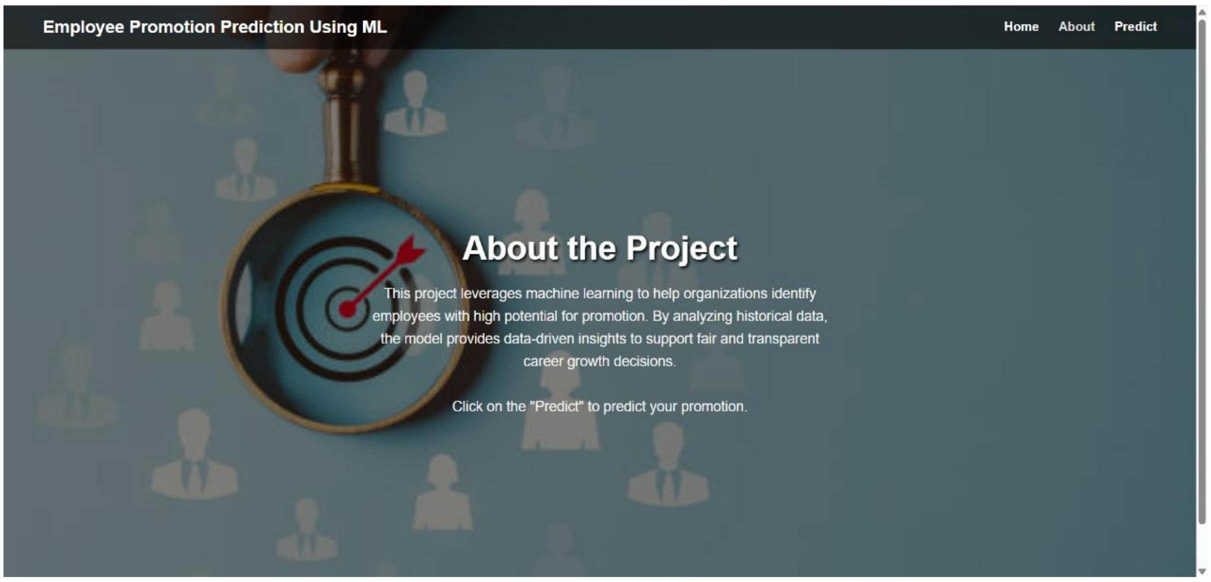
- **home.html:** The landing page with a brief introduction and navigation links.



Predict Your Next Career Move!

Empower your career decisions with Machine Learning insights. Analyze employee data, experience, and performance metrics to forecast promotion outcomes with accuracy.

- **about.html:** Details the project's background, objectives, and technology stack.

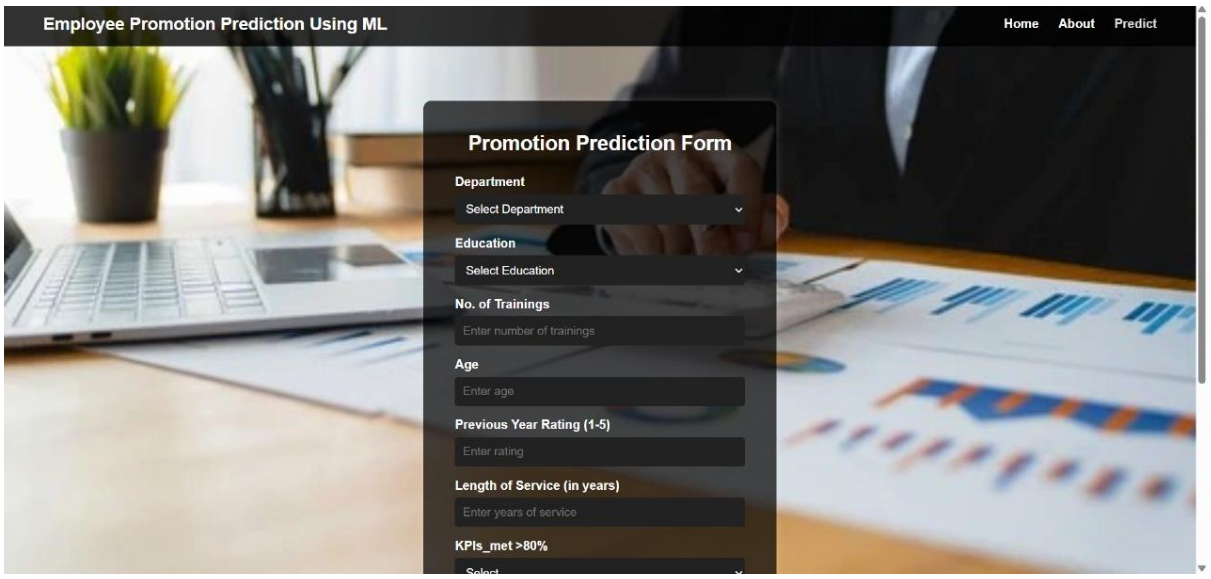


About the Project

This project leverages machine learning to help organizations identify employees with high potential for promotion. By analyzing historical data, the model provides data-driven insights to support fair and transparent career growth decisions.

Click on the "Predict" to predict your promotion.

- **predict.html:** Contains the **HTML Form** for users to input the required employee parameters (KPIs, scores, tenure). This form uses the **POST** method to send data to the Flask backend.


Employee Promotion Prediction Using ML

Home About Predict

Promotion Prediction Form

Department

Select Department

Education

Select Education

No. of Trainings

Enter number of trainings

Age

Enter age

Previous Year Rating (1-5)

Enter rating

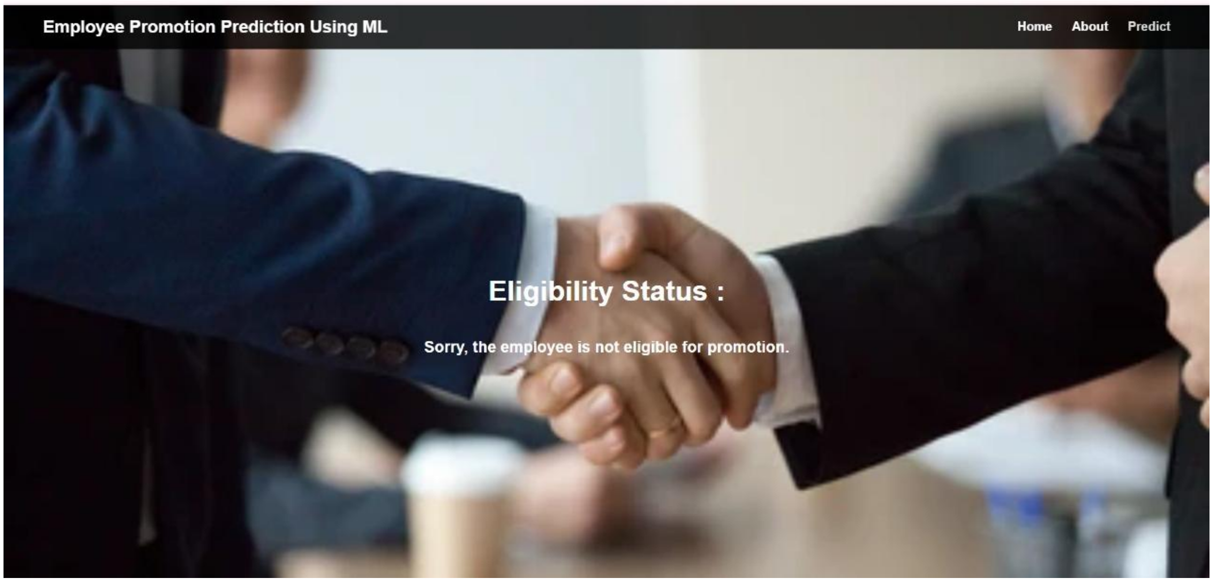
Length of Service (in years)

Enter years of service

KPIs_met >80%

Select

- **submit.html:** The results page used by the Flask backend to dynamically display the prediction output.


Employee Promotion Prediction Using ML

Home About Predict

Eligibility Status :

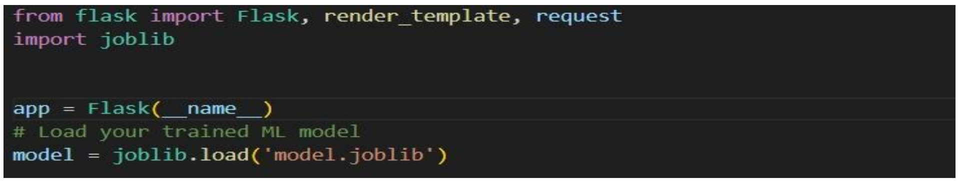
Sorry, the employee is not eligible for promotion.

## 9.2. Build Python Code

The primary script, app.py, handles the backend logic:

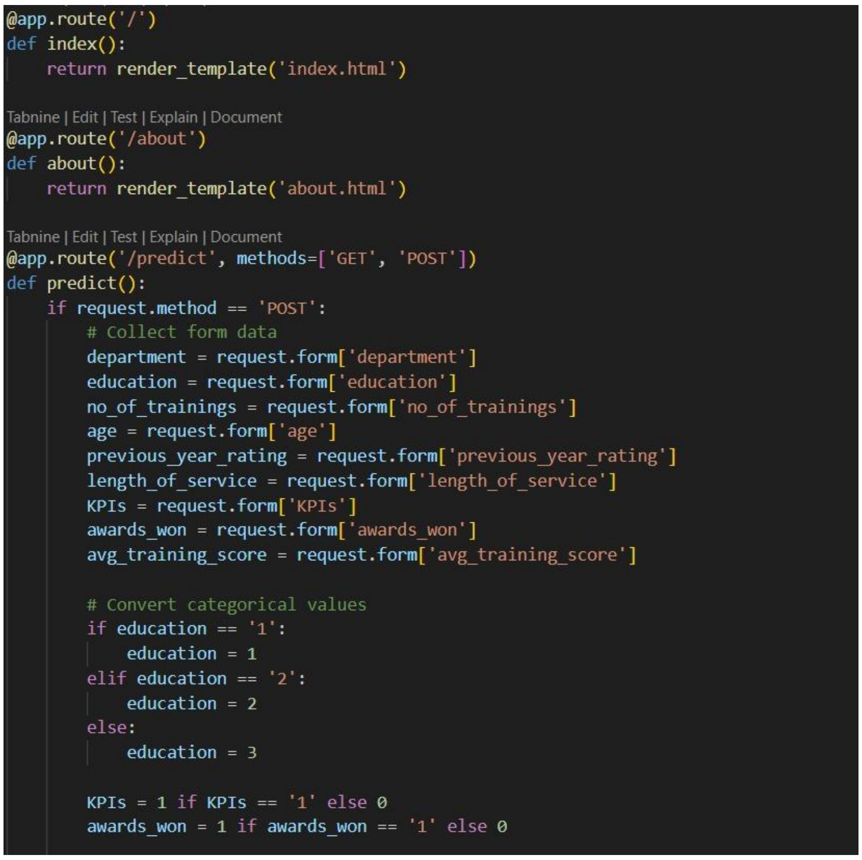
```
from flask import Flask, render_template, request
import joblib
```

- **Model Loading:** The script loads model.pkl using the pickle.load() function.



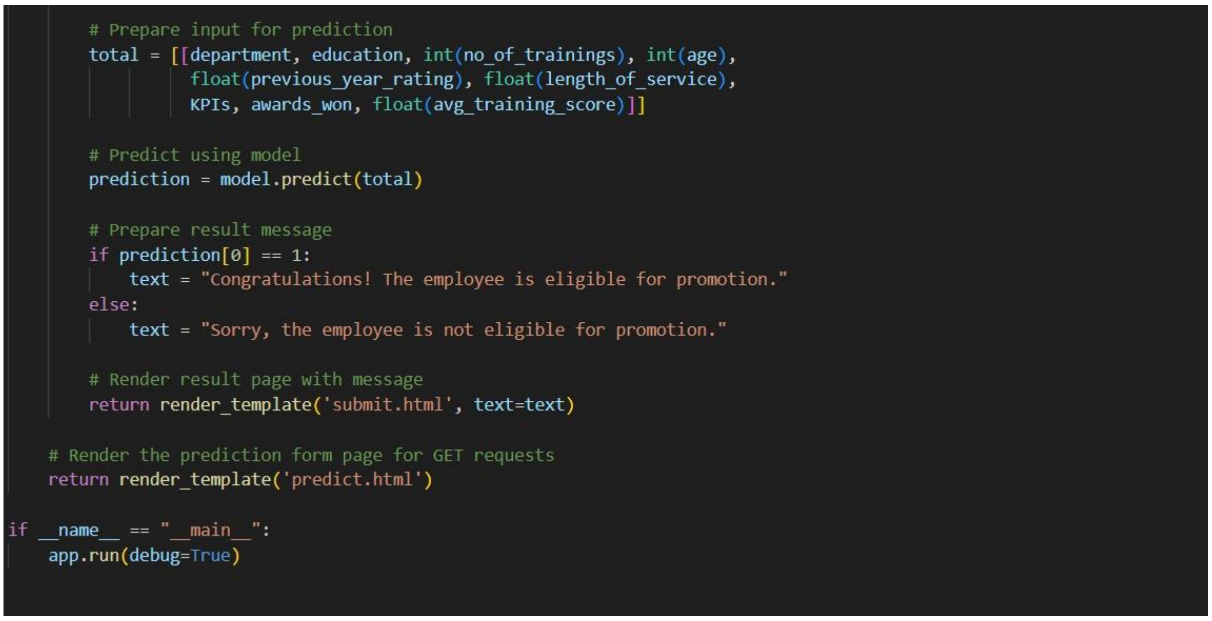
```python
from flask import Flask, render_template, request
import joblib


app = Flask(__name__)
# Load your trained ML model
model = joblib.load('model.joblib')
```

- **Routing:** The @app.route() decorator binds functions to URLs (e.g., / to home(), /predict to predict()).



```python
@app.route('/')
def index():
    return render_template('index.html')

Tabnine | Edit | Test | Explain | Document
@app.route('/about')
def about():
    return render_template('about.html')

Tabnine | Edit | Test | Explain | Document
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        # Collect form data
        department = request.form['department']
        education = request.form['education']
        no_of_trainings = request.form['no_of_trainings']
        age = request.form['age']
        previous_year_rating = request.form['previous_year_rating']
        length_of_service = request.form['length_of_service']
        KPIs = request.form['KPIs']
        awards_won = request.form['awards_won']
        avg_training_score = request.form['avg_training_score']

        # Convert categorical values
        if education == '1':
            education = 1
        elif education == '2':
            education = 2
        else:
            education = 3

        KPIs = 1 if KPIs == '1' else 0
        awards_won = 1 if awards_won == '1' else 0
```

- **Input Retrieval:** The predict() function retrieves employee data from the HTML form using request.form (via POST request).
- **Prediction and Output:** The retrieved data is converted to a numerical array, passed to model.predict(), and the binary result (0 or 1) is translated into a human-readable message, which is then rendered on submit.html.



```python
        # Prepare input for prediction
        total = [[department, education, int(no_of_trainings), int(age),
                  float(previous_year_rating), float(length_of_service),
                  KPIs, awards_won, float(avg_training_score)]]

        # Predict using model
        prediction = model.predict(total)

        # Prepare result message
        if prediction[0] == 1:
            text = "Congratulations! The employee is eligible for promotion."
        else:
            text = "Sorry, the employee is not eligible for promotion."

        # Render result page with message
        return render_template('submit.html', text=text)

    # Render the prediction form page for GET requests
    return render_template('predict.html')

if __name__ == "__main__":
    app.run(debug=True)
```

## 9.3. Run the Application

The Flask application is executed from the command line:

Bash

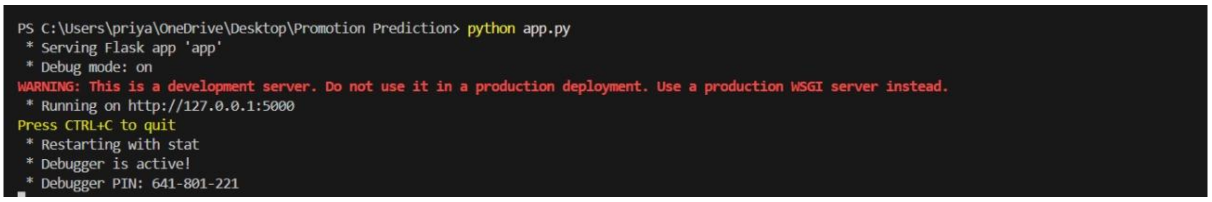
# Navigate to the project directory

$ cd path/to/project

# Execute the Flask application

$ python app.py

The application runs on a local server (e.g., http://127.0.0.1:5000/). Users access the URL, navigate to the **Predict** page, input data, and receive instant feedback upon hitting **Submit**.



```
PS C:\Users\priya\OneDrive\Desktop\Promotion Prediction> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 641-801-221
```

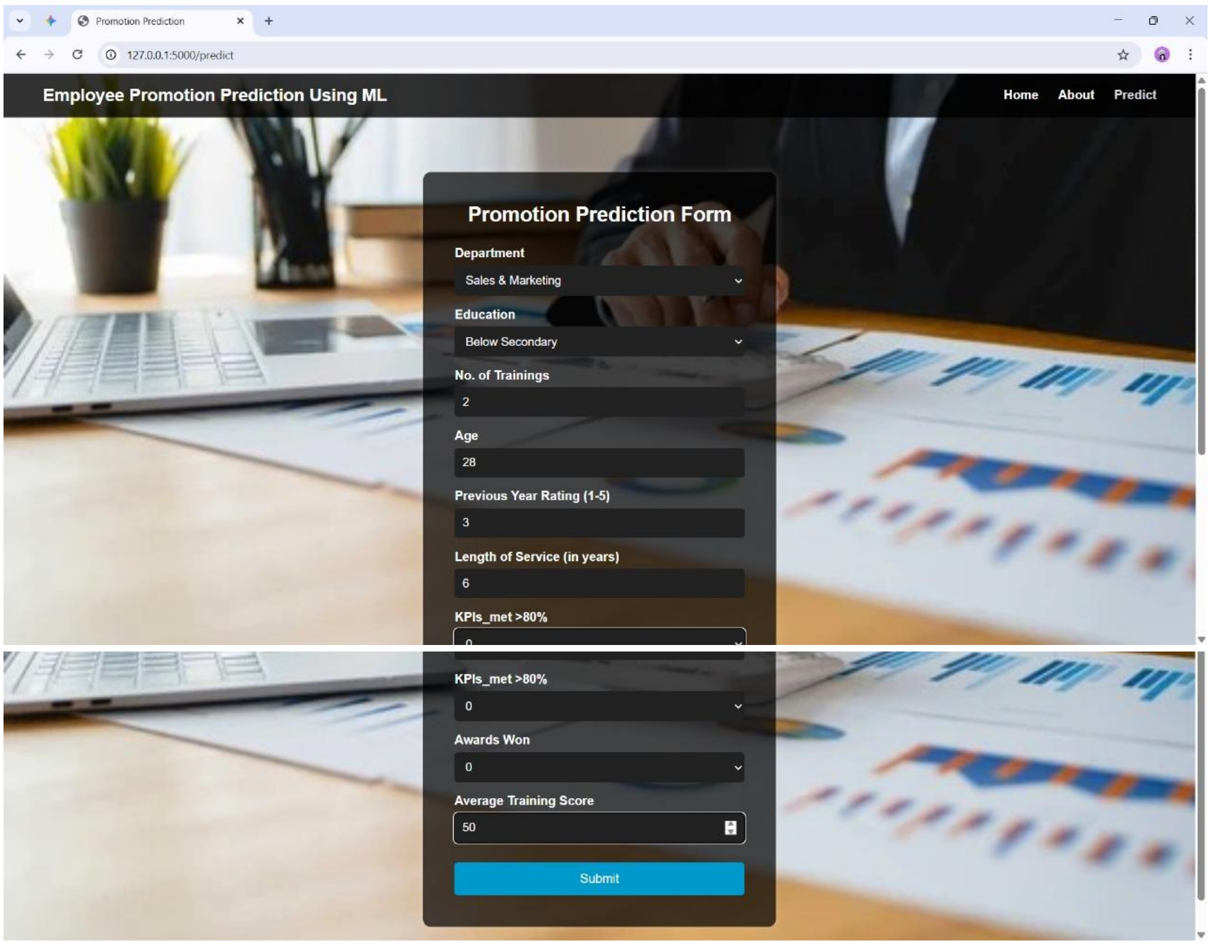
## 9.4. Output

The output demonstrates the real-time functionality of the deployed system.

- **Home/About Navigation:** The initial pages confirm the functional structure and navigation.

- **Input Demonstration:** A demonstration of input values is provided (e.g., high KPI and training score, leading to **"Employee is likely to be promoted."**).
- **Output Display:** The predicted outcome is dynamically displayed on the submit.html page, validating the end-to-end integration of the machine learning model and the web interface.

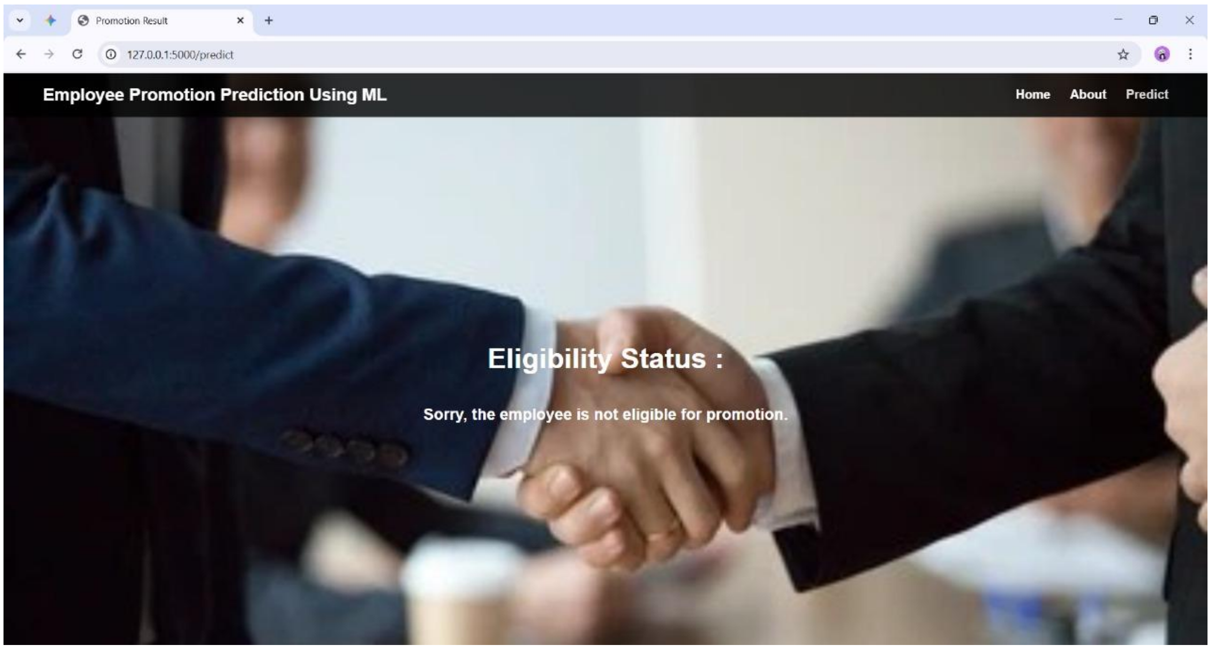
**Input-1:**



Promotion Prediction Form

| | |
|---|---|
| Department | Sales & Marketing |
| Education | Below Secondary |
| No. of Trainings | 2 |
| Age | 28 |
| Previous Year Rating (1-5) | 3 |
| Length of Service (in years) | 6 |
| KPIs_met >80% | 0 |
| Awards Won | 0 |
| Average Training Score | 50 |

## Output-1:


Promotion Result

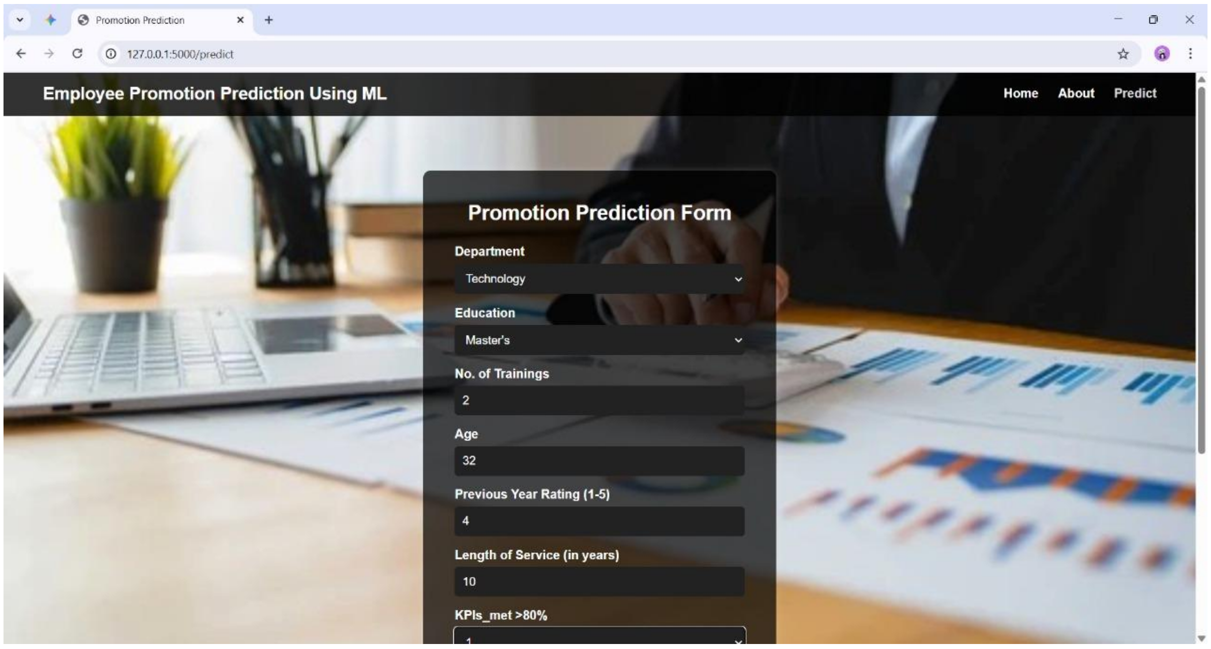
127.0.0.1:5000/predict

Employee Promotion Prediction Using ML

Eligibility Status :

Sorry, the employee is not eligible for promotion.

## Input-2:


Promotion Prediction

127.0.0.1:5000/predict

Employee Promotion Prediction Using ML

Promotion Prediction Form

Department: Technology

Education: Master's

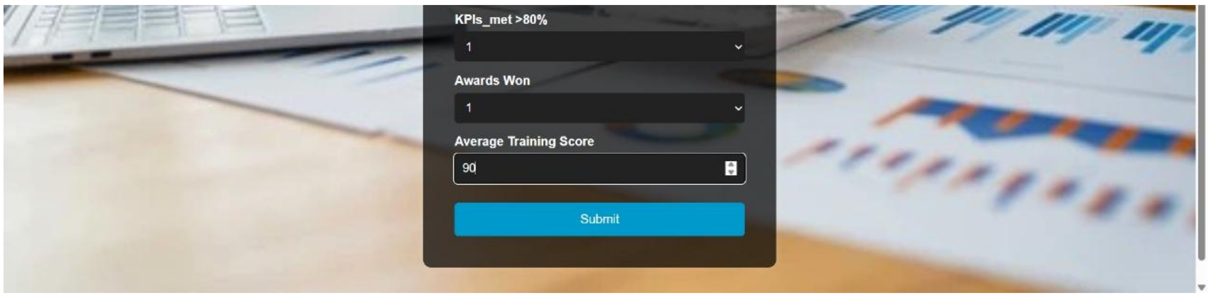
No. of Trainings: 2

Age: 32

Previous Year Rating (1-5): 4

Length of Service (in years): 10

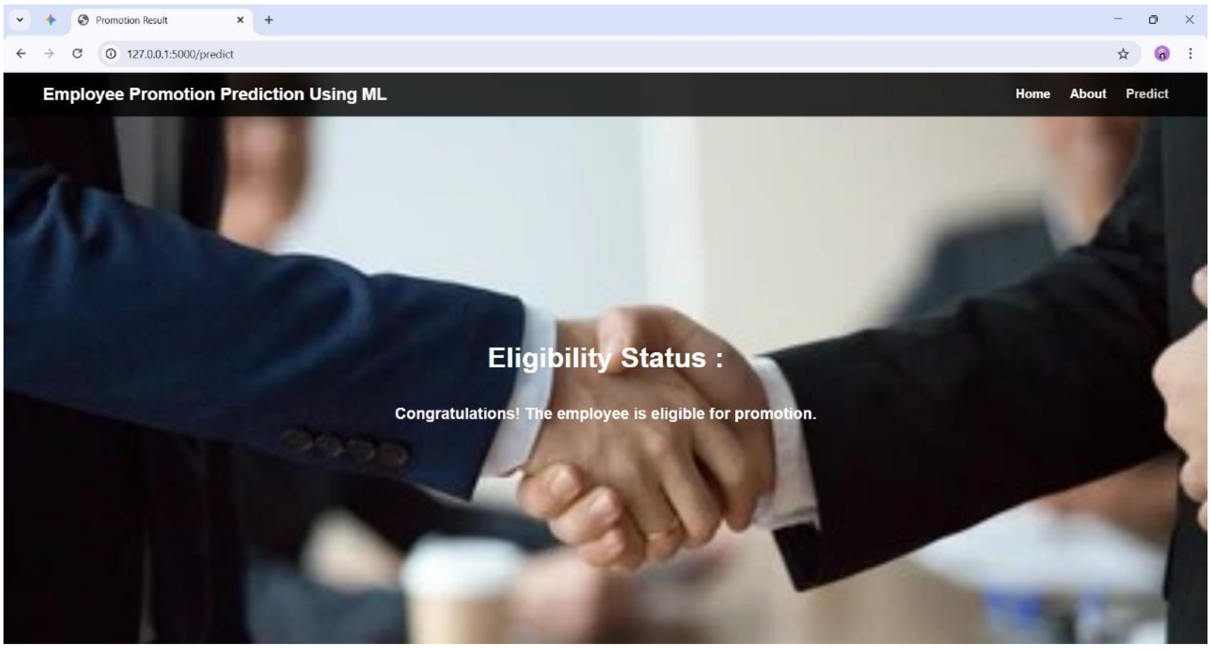
KPIs_met >80%


KPIs_met >80%: 1

Awards Won: 1

Average Training Score: 90

**Output-2:**


Promotion Result

127.0.0.1:5000/predict

Eligibility Status :

Congratulations! The employee is eligible for promotion.