Human Resource Management: Predicting Employee Promotions Using Machine Learning

Final Project Report

1. Introduction

1.1. Project overviews

This project, titled Human Resource Management: Predicting Employee Promotions Using Machine Learning, aims to develop a machine learning model to forecast the likelihood of employees being promoted within an organization. By analyzing various factors such as performance metrics, tenure, skills, and feedback, the model assists HR departments in identifying high-potential employees for advancement. This initiative is designed to enhance workforce management strategies, foster employee engagement, and improve retention, ultimately contributing to the organization's growth and success.

1.2. Objectives

- Develop a predictive model for employee promotions.
- Streamline the promotion process in large corporations.
- Establish a fair and transparent promotion process in startups.
- Proactively identify and nurture high-performing employees to prevent attrition.

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

In today's competitive business environment, organizations face significant challenges in managing employee promotions efficiently and fairly due to the sheer volume of data, potential biases, and the need for transparent processes. Large corporations struggle to identify top performers, startups seek fair promotion systems to foster growth, and companies in competitive industries aim to retain high-performing employees. To address these issues, we propose developing a machine learning model to predict employee promotions based on factors such as performance metrics, tenure, skills, and feedback. This solution aims to streamline promotion processes, ensure fairness, enhance retention, and foster a culture of meritocracy and career progression, ultimately contributing to organizational growth and employee satisfaction.

2.2. Project Proposal (Proposed Solution)

Approach: The proposed solution involves using machine learning algorithms to analyze employee data and predict promotion eligibility based on various factors such as performance metrics, tenure, skills, and feedback. The model will be trained on historical data and validated to ensure accuracy and reliability.

Key Features:

- **Automated Data Analysis**: The model will automatically analyze large datasets, saving time and reducing manual errors.
- **Accurate Predictions**: Leveraging advanced machine learning techniques to provide precise predictions on promotion eligibility.
- **Transparency and Fairness**: The model will ensure a fair evaluation process by considering multiple performance factors objectively.
- **Scalability**: The solution can be scaled to accommodate organizations of different sizes and industries.

2.3. Initial Project Planning

The initial project planning phase outlines the key steps and milestones necessary for the successful execution of the project, "Human Resource Management: Predicting Employee Promotions Using Machine Learning." This phase is crucial in setting a clear roadmap and ensuring that all necessary preparations are in place. Below are the detailed plans for each step of the project:

Data Collection and Preprocessing

- Understanding & Loading Data: Gain a comprehensive understanding of the dataset structure and contents. Load the data into the working environment for analysis.
- Exploratory Data Analysis (EDA): Perform EDA to uncover patterns, trends, and relationships within the data. Visualize data distributions and correlations.
- Handling Null Values: Identify and address missing values in the dataset using appropriate imputation techniques or removing records if necessary.
- Handling Outliers: Detect and manage outliers that may skew the model by applying methods such as z-score, IQR, or transformation techniques.
- Handling Categorical Values: Encode categorical variables into numerical format using techniques like one-hot encoding or label encoding.

Model Building

- Training the Model: Train multiple machine learning models, including Decision Tree, Random Forest, XGBoost, and KNN, using the preprocessed data.
- Comparing Models: Compare the trained models based on performance metrics such as accuracy, precision, recall, and F1 score to identify the most effective model.
- Evaluating and Saving the Model: Evaluate the best-performing model using the test dataset and save the model for future use.
- Model Optimization: Fine-tune hyperparameters using techniques like Grid Search and Randomized Search to optimize model performance.

Web Integration and Deployment

- Building HTML Pages: Develop the front-end interface of the web application, including pages for home, about, prediction input, and results.
- Local Deployment: Deploy the web application locally to test its functionality and ensure smooth integration with the predictive model.

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

Data Collection Plan :

- Extract data from internal HR databases containing employee details, performance metrics, and promotion records.
- Prioritize datasets with comprehensive demographic information, including department, education level, and length of service.

Data Sources :

- Source Name: Kaggle Dataset
- Description: The dataset comprises various employee attributes such as department, education, training history, performance ratings, and promotion status.
- Location/URL: Kaggle HR Analytics Dataset
- Format: CSV
- Size: Approximately: 4 MB
- Access Permissions: Public

3.2. Data Quality Report

- Missing values in the ‘education’ and ‘previous year rating’ columns
- Categorical data in the dataset.
- Negative Data in the Dataset
- Imbalanced Data

3.3. Data Exploration and Preprocessing

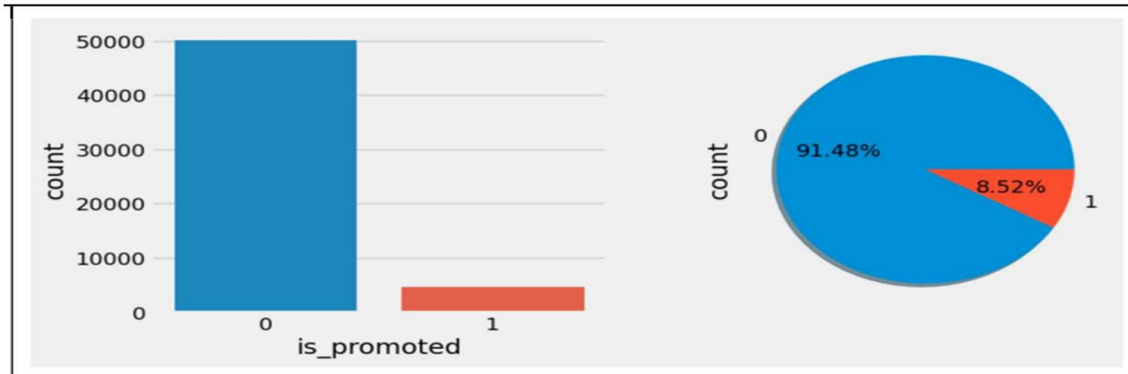
Data Overview:

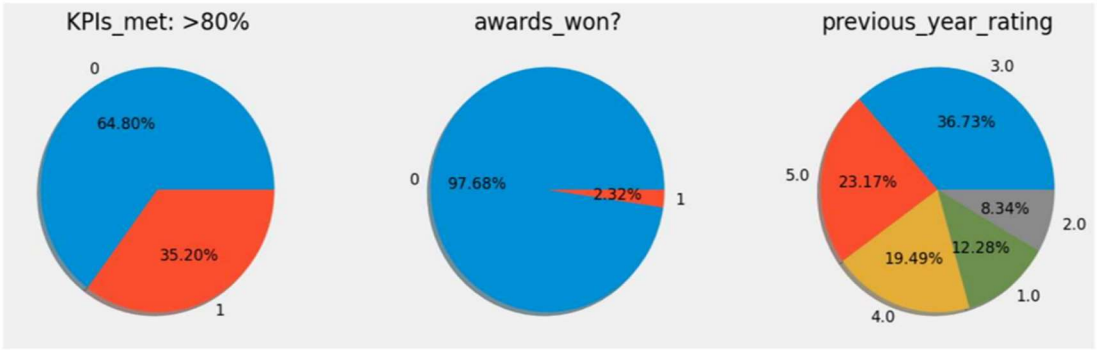
Dimensions: 54808 rows × 14 columns

Descriptive statistics:

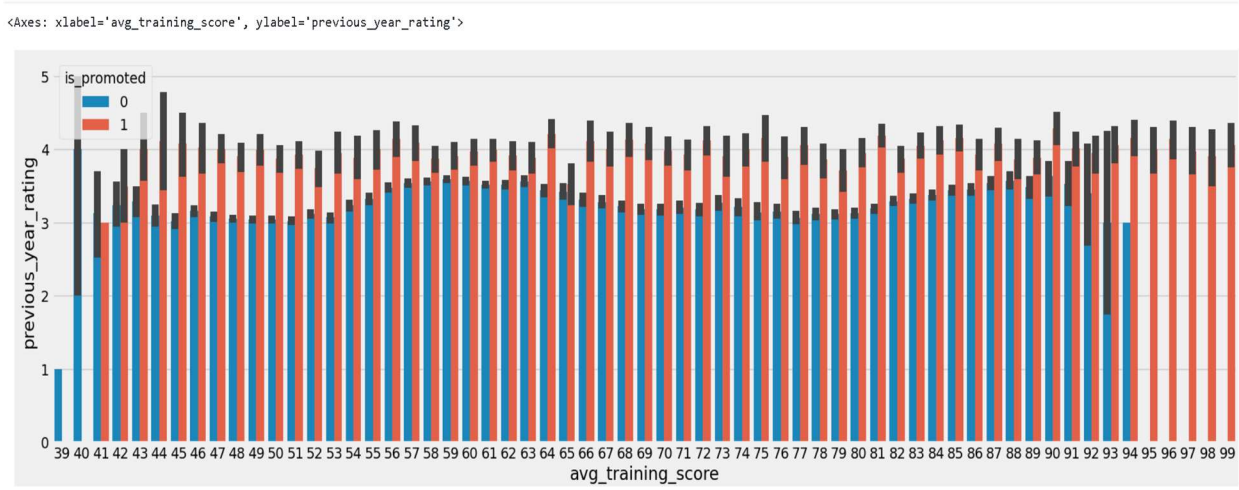
	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_scor
count	54808.000000	54808	54808	52399	54808	54808	54808.000000	54808.000000	50684.000000	54808.000000	54808.000000	54808.000000	54808.00000
unique	NaN	9	34	3	2	3	NaN	NaN	NaN	NaN	NaN	NaN	Nal
top	NaN	Sales & Marketing	region_2	Bachelor's	m	other	NaN	NaN	NaN	NaN	NaN	NaN	Nal
freq	NaN	16840	12343	36669	38496	30446	NaN	NaN	NaN	NaN	NaN	NaN	Nal
mean	39195.830627	NaN	NaN	NaN	NaN	NaN	1.253011	34.803915	3.329256	5.865512	0.351974	0.023172	63.38675
std	22586.581449	NaN	NaN	NaN	NaN	NaN	0.609264	7.660169	1.259993	4.265094	0.477590	0.150450	13.37155
min	1.000000	NaN	NaN	NaN	NaN	NaN	1.000000	20.000000	1.000000	1.000000	0.000000	0.000000	39.00000
25%	19669.750000	NaN	NaN	NaN	NaN	NaN	1.000000	29.000000	3.000000	3.000000	0.000000	0.000000	51.00000
50%	39225.500000	NaN	NaN	NaN	NaN	NaN	1.000000	33.000000	3.000000	5.000000	0.000000	0.000000	60.00000
75%	58730.500000	NaN	NaN	NaN	NaN	NaN	1.000000	39.000000	4.000000	7.000000	1.000000	0.000000	76.00000
max	78298.000000	NaN	NaN	NaN	NaN	NaN	10.000000	60.000000	5.000000	37.000000	1.000000	1.000000	99.00000

Univariate Analysis :

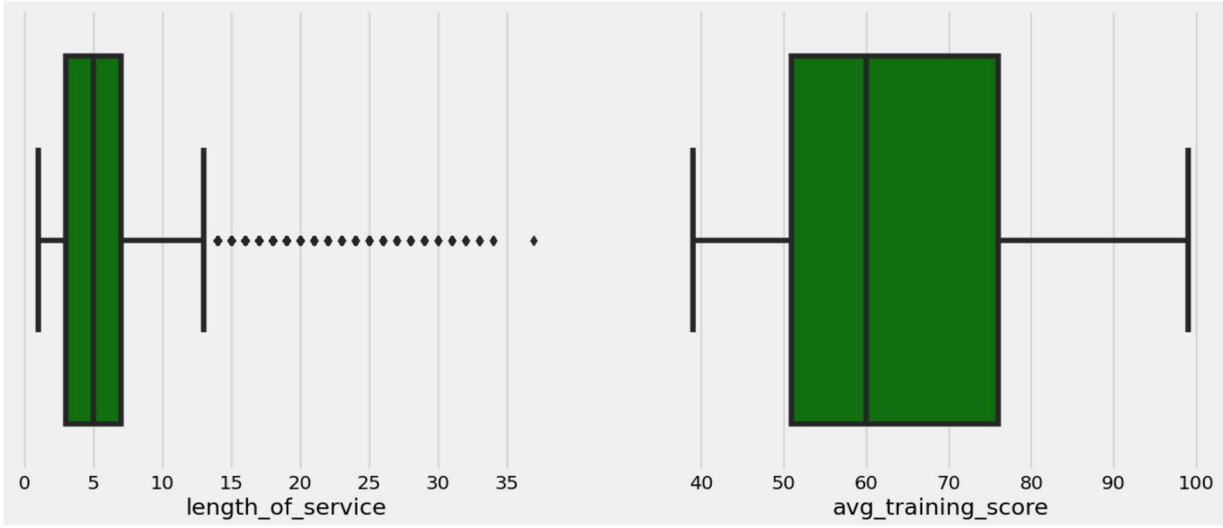




Multivariate Analysis:



Outliers and Anomalies:



Loading Data :

```
# Reading the csv and printing its shape

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

0.1s Python

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

```
df.head(5)
```

0.0s Python

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.0	8	1	0	49	0
1	65141	Operations	region_22	Bachelor's	m	other	1	30	5.0	4	0	0	60	0
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.0	7	0	0	50	0
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.0	10	0	0	50	0
4	48945	Technology	region_26	Bachelor's	m	other	1	45	3.0	2	0	0	73	0

Handling Missing Data:

```
# Replacing nan with mode

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

0.0s

```
# 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])
```

Data Transformation :

```
# Feature mapping is done on education column
import joblib
df['education']=df['education'].replace(("Below Secondary","Bachelor's","Master's & above"),(1,2,3))

lb = LabelEncoder()
df['department']=lb.fit_transform(df['department'])
```

4. Model Development Phase

4.1. Feature Selection Report:

Feature	Description	Selected (Yes/No)	Reasoning
employee_id	Unique identifier for each employee	No	Not required for predicting promotions as it doesn't provide predictive value
department	Department the employee belongs to	Yes	Relevant to determine promotion patterns across different departments
region	Region of the employee	**No**	Not important for predicting promotions in this context.
education	Employee's education level	Yes	Self-employed individuals may have different financial profiles.

gender	Employee's gender	No	Not important for predicting promotions in this context
Recruitment channel	Recruitment channel through which hired	No	Not important for predicting promotions in this context
No of trainings	Number of training sessions attended	Yes	Additional training sessions can improve promotion readiness
age	Age of the employee	Yes	Age can indicate experience and influence promotions
Previous year rating	Performance rating from the previous year	Yes	Direct indicator of past performance, crucial for promotion decisions
Length of service	Length of service in the company	Yes	Company loyalty and experience are important for promotions
KPIs met above 80	KPIs met above 80% (0/1)	Yes	KPI performance is critical for assessing employee performance
awards_won	Whether the employee has won any awards (0/1)	Yes	Awards indicate high performance and recognition, influencing promotion decisions

4.2. Model Selection Report

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance. To evaluate the performance confusion matrix and classification report is used

Model	Description	Hyperparameters	Performance Metric (e.g., Accuracy, F1 Score)
Decision Tree	Simple tree structure; interpretable, captures non-linear relationships, suitable for initial insights into promotion patterns	random_state=42	Accuracy score : 93%
Random Forest	An ensemble learning method for classification that operates by constructing multiple decision trees during training and outputting the mode of the classes as the prediction.	random_state=42	Accuracy score: 95%

K-Nearest Neighbors (KNN)	Classifies based on nearest neighbors; adapts well to data patterns, effective for local variations in promotion criteria	n_neighbors=5	Accuracy score: 89%
XGboost	Gradient boosting with trees; optimizes predictive performance, handles complex relationships, and is suitable for accurate promotion predictions	random_state=42	**Accuracy score: 86%**

4.3. Initial Model Training Code, Model Validation and Evaluation Report

Training code :

Descision Tree Model

```
def decisionTree(X_train, X_test, y_train, y_test):
    # Initialize the DecisionTreeClassifier
    model = DecisionTreeClassifier(random_state=42)

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("Confusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return model

# Call the function with training and testing data
decisionTree(x_train, x_test, y_train, y_test)
```

Random Forest Model

```
def randomForest(X_train, X_test, y_train, y_test):
    # Initialize the RandomForestClassifier
    model = RandomForestClassifier(random_state=42, n_estimators=100)
    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return model
# Call the function with training and testing data
randomForest(x_train, x_test, y_train, y_test)
```

KNN Model

```python
# Function to train and evaluate a KNN model
def KNN(X_train, X_test, y_train, y_test):
    # Initialize the KNeighborsClassifier
    model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors (k) as needed
    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")
    return model
# Call the function with training and testing data
KNN(x_train, x_test, y_train, y_test)
```

XGboost Model

```python
def xgboost(X_train, X_test, y_train, y_test):
    # Initialize the GradientBoostingClassifier
    model = GradientBoostingClassifier(random_state=42)
    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")
    return model

# Call the function with training and testing data
xgboost(x_train, x_test, y_train, y_test)
```

Model Validation and Evaluation Report:

Decision Tree:

```
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

Accuracy: 0.93
```

Accuracy: 0.93

```
Confusion Matrix:
[[13875  1190]
 [  902 14117]]
```

Random Forest:

```
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

Accuracy: 0.95
```

Accuracy: 0.95

```
Confusion Matrix:
[[14195   870]
 [  748 14271]]
```

KNN:

```
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

Accuracy: 0.89
```

Accuracy: 0.89

```
Confusion Matrix:
[[12266  2799]
 [  476 14543]]
```

XGboost:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86     15065
           1       0.85      0.89      0.87     15019

    accuracy                           0.86     30084
   macro avg       0.86      0.86      0.86     30084
weighted avg       0.86      0.86      0.86     30084

Accuracy: 0.86
```

Accuracy: 0.87

```
Confusion Matrix:
[[12631  2434]
 [ 1669 13350]]
```

5. Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

5.1. Hyperparameter Tuning Documentation

Decision Tree :

Tuned Hyperparameters

```
# Define the hyperparameters and their possible values
param_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5),
    'max_features': [None, 'auto', 'sqrt', 'log2']
}
```

Optimal Values:

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'criterion': 'gini', 'max_depth': 30, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 4}
```

```
Accuracy: 0.93
```

Random Forest:

Tuned Hyperparameters

```

# Define the hyperparameters and their possible values
param_dist = {
    'n_estimators': randint(100, 500),
    'max_features': ['auto', 'sqrt'],
    'max_depth': randint(10, 30),
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 3),
    'bootstrap': [True]
}
```

Optimal Values:

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Parameters: {'bootstrap': True, 'max_depth': 28, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 419}
Confusion Matrix:
```

```
Accuracy: 0.95
```

KNN:

Tuned Hyperparameters

```
# Define the hyperparameters and their possible values
param_dist = {
    'n_neighbors': randint(1, 30),  # Number of neighbors
    'weights': ['uniform', 'distance'],  # Weight function
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Distance metric
}
```

Optimal Values:

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 4, 'weights': 'distance'}
```

```
Accuracy: 0.91
```

XGboost:

Tuned Hyperparameters

```
    param_dist = {
        'n_estimators': randint(50, 500),  # N
        'learning_rate': uniform(0.01, 0.3),
        'max_depth': randint(3, 15),  # Maximu
        'min_child_weight': randint(1, 10),  #
        'subsample': uniform(0.5, 0.5),  # Fra
        'colsample_bytree': uniform(0.5, 0.5),
        'gamma': uniform(0, 5)  # Minimum loss
    }
```

Optimal Values:

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Parameters: {'colsample_bytree': 0.625125680257932, 'gamma': 0.1941736721471116, 'learning_rate': 0.10097965440196684, 'max_depth': 13, 'min_child_weight': 5, 'n_estimators': 3
```

```
Accuracy: 0.94
```

5.2. Performance Metrics Comparison Report

Decision Tree:

Baseline Metric :

```
Confusion Matrix:
[[13853  1212]
 [  878 14141]]

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

Accuracy: 0.93
```

Optimized Metric:

```
Confusion Matrix:
[[13966  1099]
 [  983 14036]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     15065
           1       0.93      0.93      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

Accuracy: 0.93
```

Random Forest:

Baseline Metric :

```
Confusion Matrix:
[[14187   878]
 [  758 14261]]

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

Accuracy: 0.95
```

Optimized Metric:

```
Confusion Matrix:
[[14248   817]
 [  814 14205]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     15065
           1       0.95      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

Accuracy: 0.95
```

KNN:

Baseline Metric :

```
Confusion Matrix:
[[12332  2733]
 [  534 14485]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.82      0.88     15065
           1       0.84      0.96      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

Accuracy: 0.89
```

Optimized Metric:

```
Confusion Matrix:
[[13156  1909]
 [  731 14288]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.87      0.91     15065
           1       0.88      0.95      0.92     15019

    accuracy                           0.91     30084
   macro avg       0.91      0.91      0.91     30084
weighted avg       0.91      0.91      0.91     30084

Accuracy: 0.91
```

XGboost:

Baseline Metric :

```
Confusion Matrix:
[[14233   832]
 [ 1067 13952]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     15065
           1       0.94      0.93      0.94     15019

    accuracy                           0.94     30084
   macro avg       0.94      0.94      0.94     30084
weighted avg       0.94      0.94      0.94     30084

Accuracy: 0.94
```

Optimized Metric:

```
Confusion Matrix:
[[12546  2519]
 [ 1454 13565]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.83      0.86     15065
           1       0.84      0.90      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

Accuracy: 0.87
```

5.3. Final Model Selection Justification

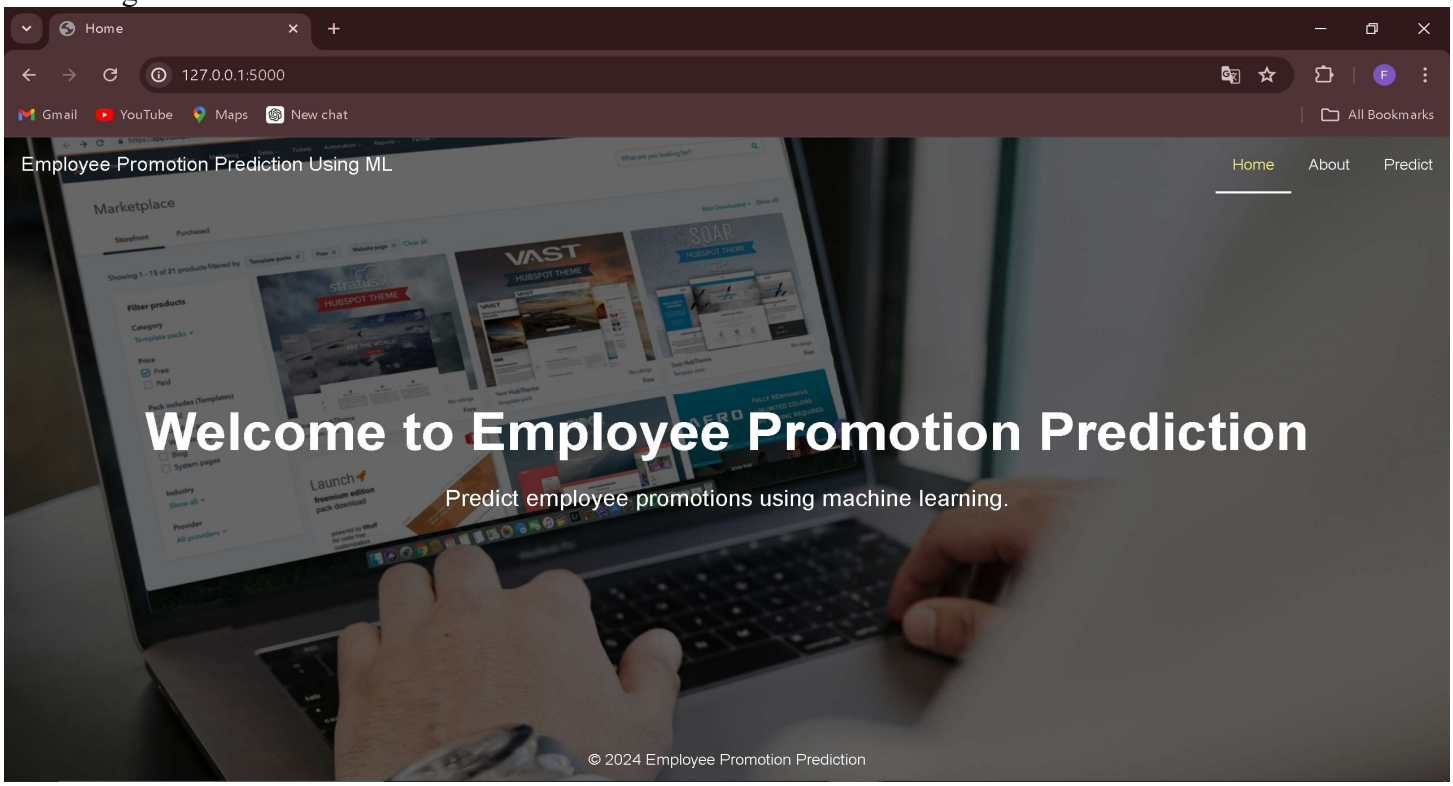
Model : Random Forest

Reasoning : I chose the Random Forest model for predicting employee promotions due to its highest accuracy of 95%, outpacing Decision Tree, KNN, and Gradient Boosting. Its robustness, ability to handle overfitting, and insights into feature importance, combined with its capability to manage complex, non-linear data and scale with large datasets, make it a reliable choice. Hyperparameter tuning further enhanced its performance, confirming its effectiveness for this task.

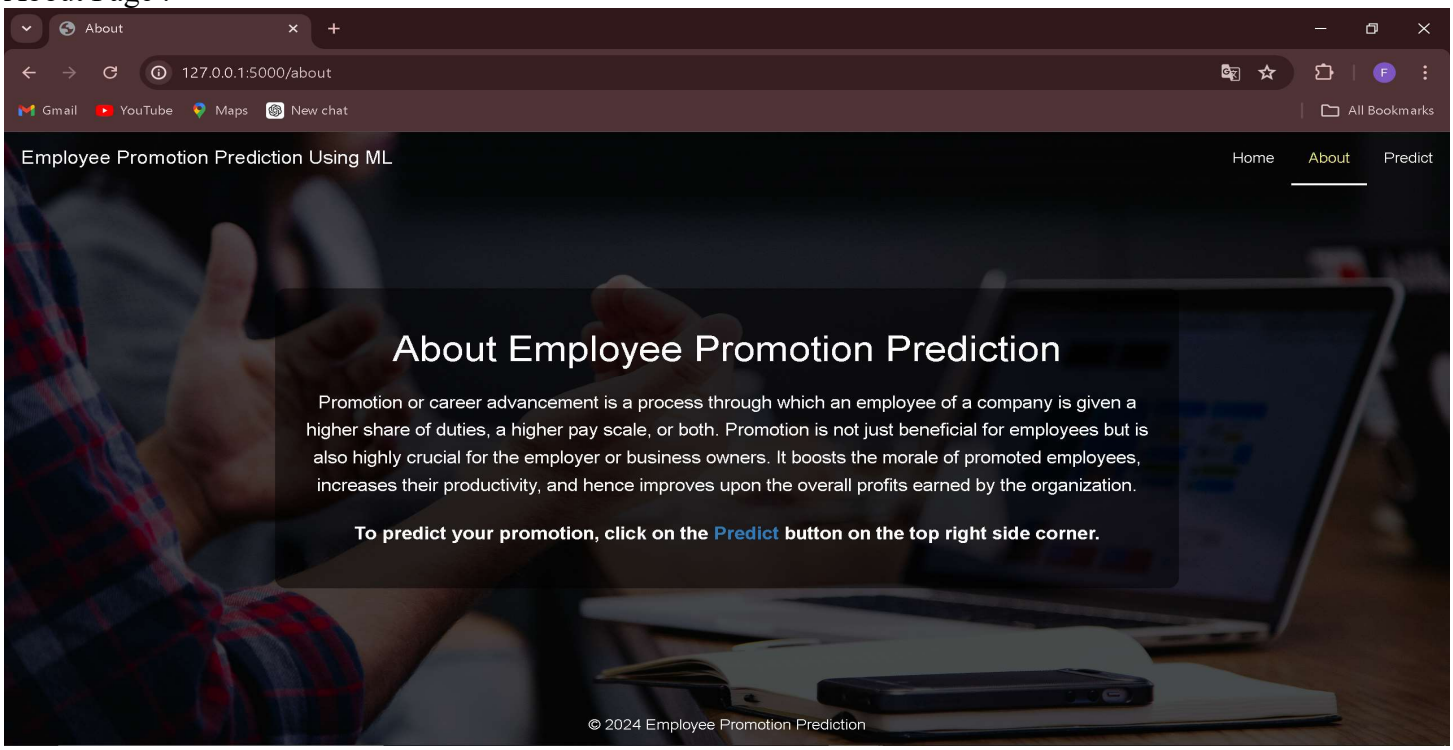
6. Results

6.1. Output Screenshots

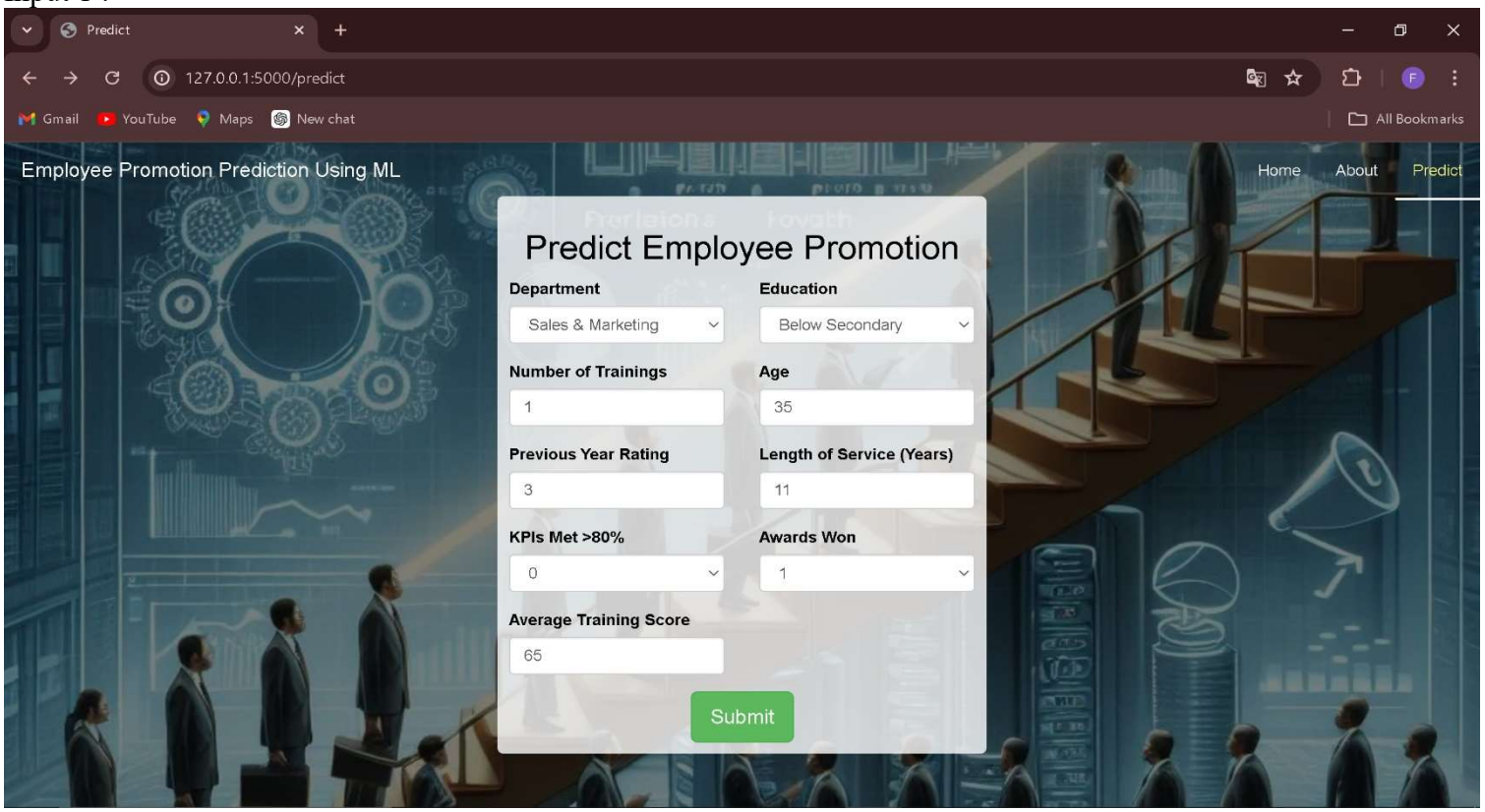
Home Page :



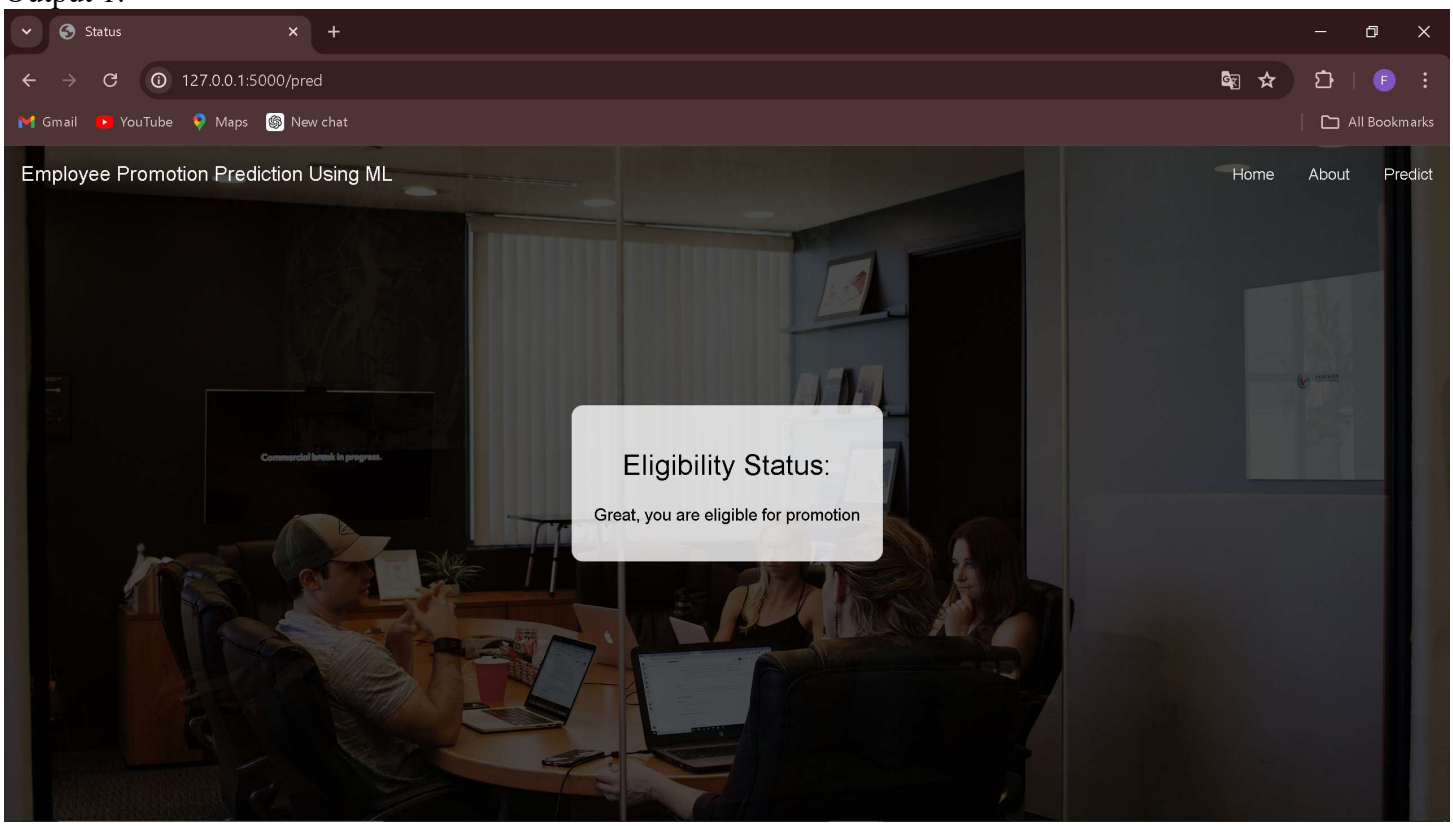
About Page :



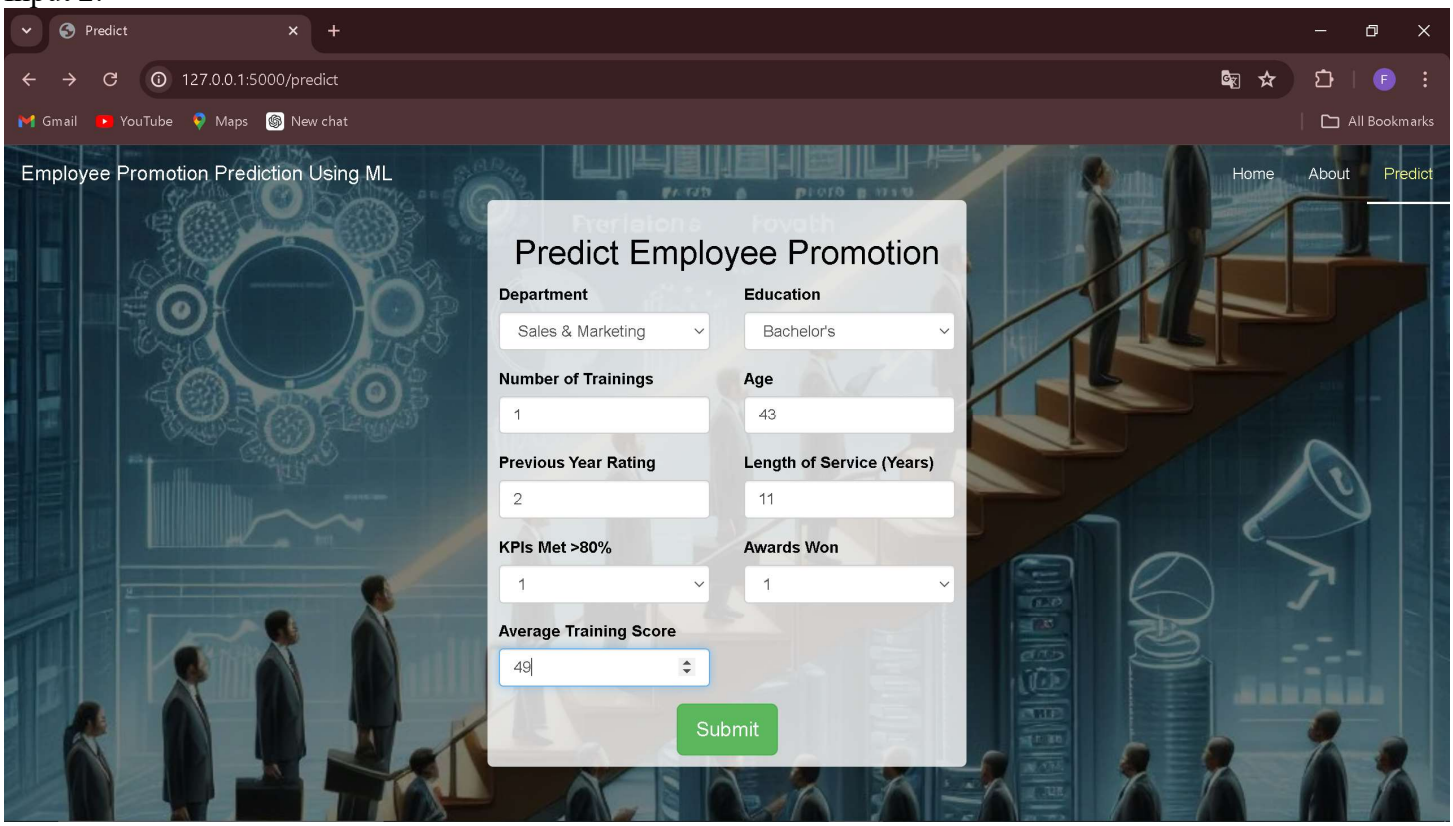
Input 1 :



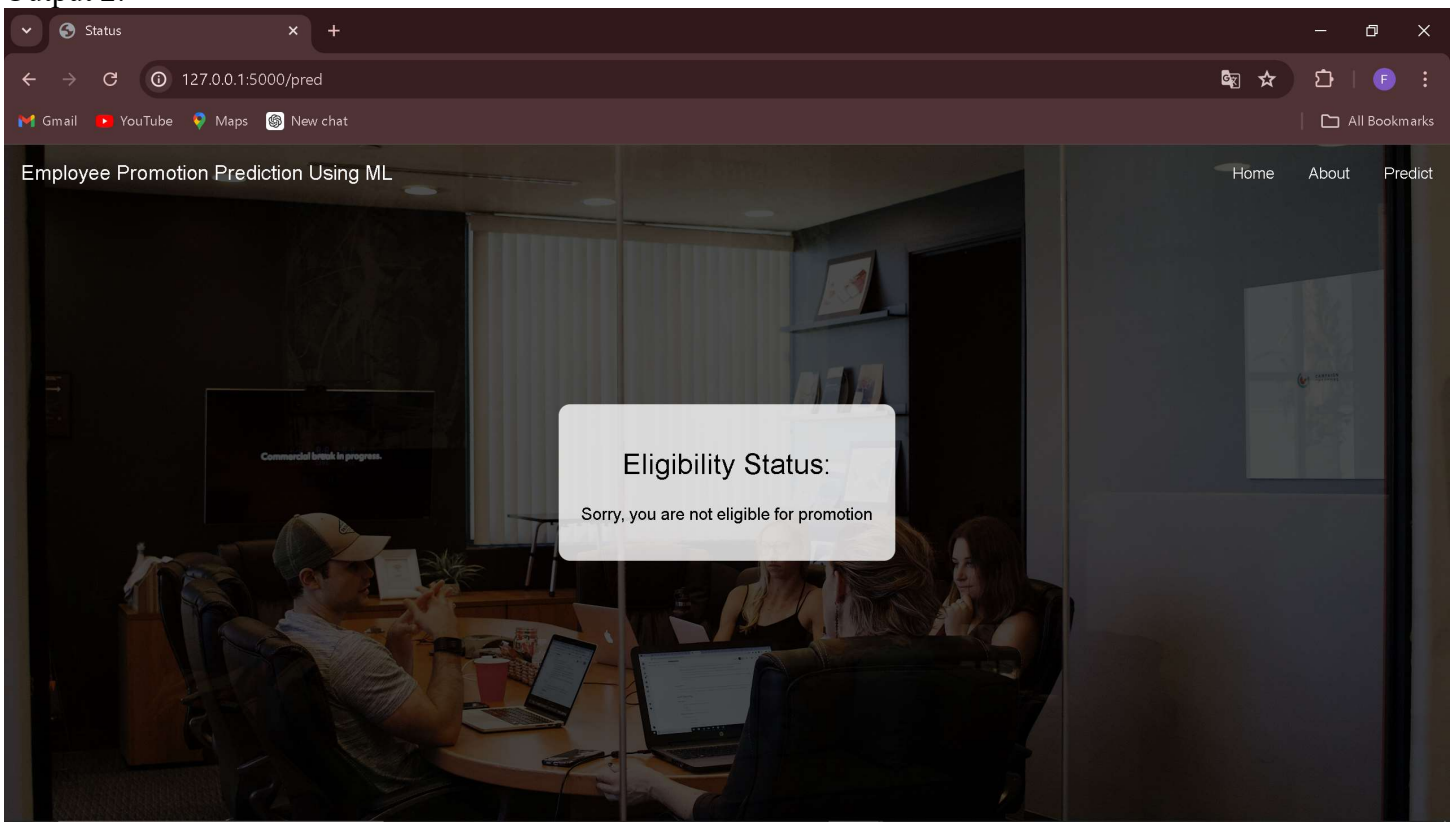
Output 1:



Input 2:



Output 2:



7. Advantages & Disadvantages:

Advantages:

- Efficient program for Employee promotion prediction
- Accurate output is produced
- Will predict Employee promotion with extreme accuracy
- Relatively inexpensive and fast

Disadvantages:

- It will work in all condition but some condition it may not give correct output

8. Conclusion

This project successfully developed a machine learning model to predict employee promotions, offering a data-driven solution to streamline HR processes. By accurately forecasting promotion eligibility, the model enhances fairness and transparency in the promotion process, improving employee satisfaction and retention.

9. Future Scope

- Expand Dataset: Incorporate additional datasets to further improve model accuracy and generalizability.
- Feature Expansion: Explore new features such as employee engagement scores and peer reviews.
- Model Deployment: Integrate the model into HR management systems for real-time promotion predictions.
- Continuous Learning: Implement a continuous learning system to update the model with new data periodically.

10. Appendix

10.1. Source Code

10.2. GitHub & Project Demo Link