Human Resource Management: Predicting Employee Promotions Using Machine Learning

Final Project Report

1. **Introduction**

1.1. Project overviews

This project, Human Resource Management: Predicting Employee Promotions Using Machine Learning, is dedicated to developing an advanced machine learning model designed to forecast the promotion probabilities of employees within an organization. By meticulously analyzing critical factors such as performance metrics, tenure, skills, and feedback, this model empowers HR departments to accurately identify and nurture high-potential employees for career advancement. The initiative aims to optimize workforce management strategies, enhance employee engagement, and bolster retention rates, thereby driving organizational growth and success.

1.2. Objectives

- Enhance workforce management strategies.
- Develop a predictive model for employee promotions.
- Establish a fair and transparent promotion process in startups.
- Proactively identify & nurture high-performing employees to improve retention rates.

2. **Project Initialization and Planning Phase**

2.1. Define Problem Statement

The client faces challenges in identifying suitable candidates for promotion, necessitating a system to streamline this process. 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:

To address this, we propose a machine learning solution that automates employee promotions. By leveraging employee datasets, we will build and evaluate various

classification models, including Decision Tree, Random Forest, KNN, and XGBoost. 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 major 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.
- (EDA) Exploratory Data Analysis: 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

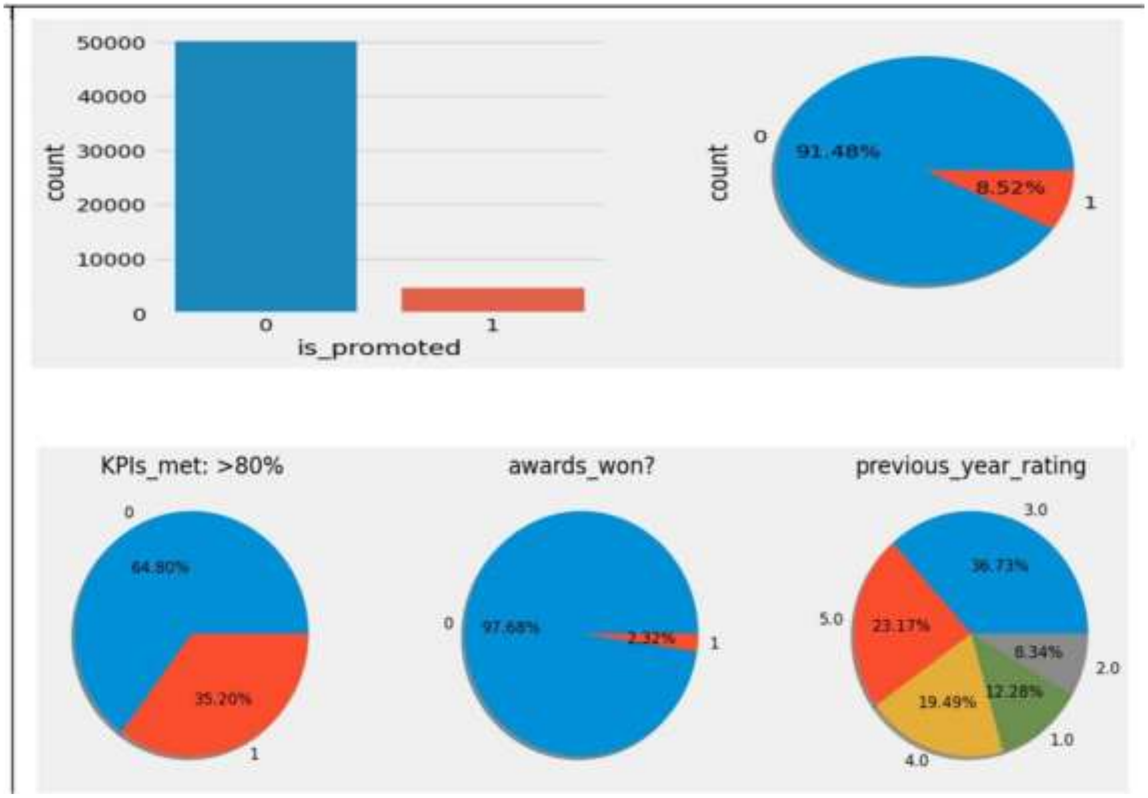
Overview:

Dimensions: 54808 rows × 14 columns Descriptive

statistics:

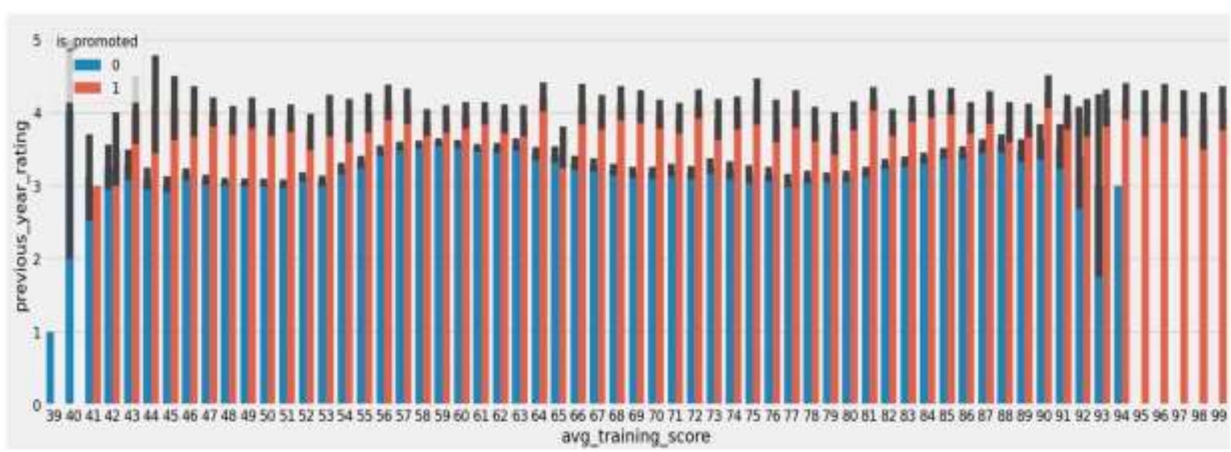
	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_scor
count	54808.000000	54808	54808	52399	54808	54808	54808.000000	54808.000000	50684.000000	54808.000000	54808.000000	54808.000000	54808.00000
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 :

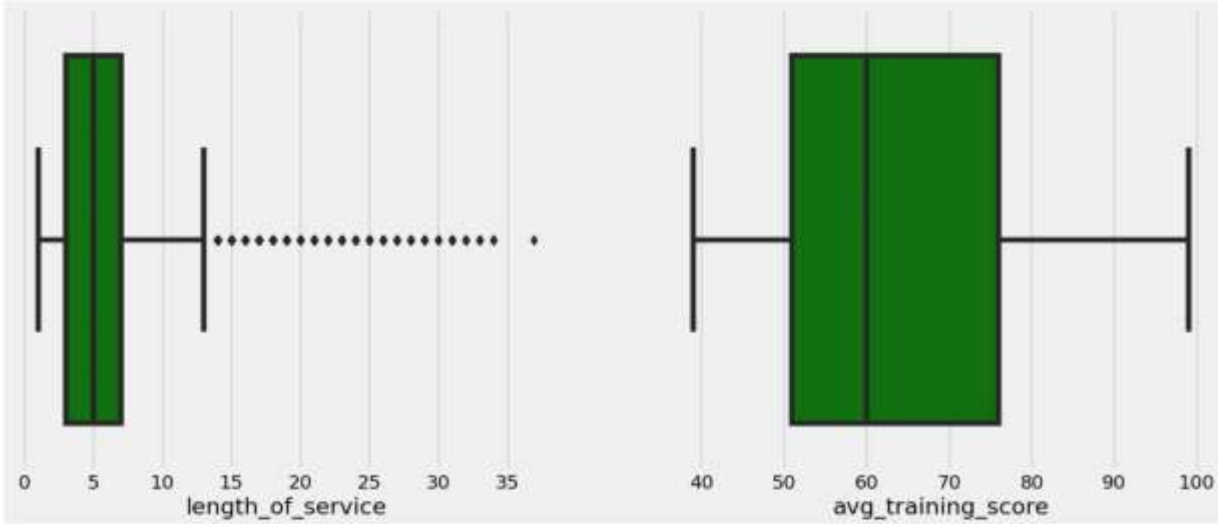


Multivariate Analysis:

<Axes: xlabel='avg_training_score', ylabel='previous_year_rating'>



Outliers and Anomalies:



Loading Data :

```python
# Reading the csv and printing its shape

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

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

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

	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:

```python
# Replacing nan with mode

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

```python
# 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 :

```python
# 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.

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

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

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 :

Decision Tree

```python
def decisionTree(X_train, X_test, y_train, y_test):
    #  parameter grid
    param_grid = {
        'max_depth': [None, 10, 20, 30, 40, 50],
        'min_samples_split': [2, 10, 20],
        'min_samples_leaf': [1, 5, 10],
        'criterion': ['gini', 'entropy']
    }

    model = DecisionTreeClassifier(random_state=42)   # Initialize the DecisionTreeClassifier

    # Initialize the GridSearchCV
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

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

    best_model = grid_search.best_estimator_   # Get the best estimator

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

    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("Best Parameters found by GridSearchCV:")
    print(grid_search.best_params_)
    print("\nConfusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return best_model

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

RANDOM FOREST MODEL

```python
def randomForest(X_train, X_test, y_train, y_test):
    # Define the parameter grid
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'bootstrap': [True, False]
    }

    model = RandomForestClassifier(random_state=42)

    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_

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

    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("Best Parameters found by GridSearchCV:")
    print(grid_search.best_params_)
    print("\nConfusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return best_model

randomForest(X_train, X_test, y_train, y_test)
```

KNN Model

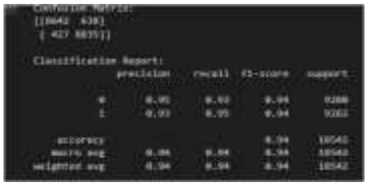
```
def KNN(X_train, X_test, y_train, y_test):

    param_grid = {
        'n_neighbors': [3, 5, 7, 9, 11],
        'weights': ['uniform', 'distance'],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
        'p': [1, 2]
    }

    model = KNeighborsClassifier(n_neighbors=5)

    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_

    y_pred = best_model.predict(X_test)

    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("Best Parameters found by GridSearchCV:")
    print(grid_search.best_params_)
    print("\nConfusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return best_model

KNN(X_train, X_test, y_train, y_test)
```

Xgboost Model

```
def xgboost(X_train, X_test, y_train, y_test):

    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 4, 5],
        'subsample': [0.8, 0.9, 1.0],
        'min_samples_split': [2, 5, 10]
    }

    model = GradientBoostingClassifier(random_state=42)

    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_

    y_pred = best_model.predict(X_test)

    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("Best Parameters found by GridSearchCV:")
    print(grid_search.best_params_)
    print("\nConfusion Matrix:")
    print(cm)
    print("\nClassification Report:")
    print(cr)
    print(f"Accuracy: {accuracy:.2f}")

    return best_model

xgboost(X_train, X_test, y_train, y_test)
```

Model Validation and Evaluation Report:

Decision Tree : -



```
Accuracy: 0.94
```

```
Confusion Matrix:
[[8642  638]
 [ 427 8835]]
```

Random Forest : -

```
Accuracy: 0.96
```

```
Confusion Matrix:
[[8892  388]
 [ 403 8859]]
```

K- Nearest Neighbour (KNN) : -

```
Accuracy: 0.93
```

```
Confusion Matrix:
[[8294  986]
 [ 222 9040]]
```

XGboost : -

```
Accuracy: 0.93
```

```
Confusion Matrix:
[[8678  602]
 [ 695 8567]]
```

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

```
param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy']
}
```

Optimal Values:

```
Best Parameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': 40, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
Accuracy: 0.94
```

Random Forest:

Tuned Hyperparameters

```python
# Function to train and evaluate a Random Forest model
def randomForest(X_train, X_test, y_train, y_test):

    # Define the parameter grid
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'bootstrap': [True, False]
    }
```

Optimal Values:

```
Best Parameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': 40, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
Accuracy: 0.96
```

KNN:

Tuned Hyperparameters

```python
# Function to train and evaluate a KNN model with hyperparameter tuning
def KNN(X_train, X_test, y_train, y_test):
    # Define the parameter grid
    param_grid = {
        'n_neighbors': [3, 5, 7, 9, 11],
        'weights': ['uniform', 'distance'],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
        'p': [1, 2]
    }
```

Optimal Values:

```
Best Parameters found by GridSearchCV:
{'algorithm': 'ball_tree', 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
```

```
Accuracy: 0.93
```

XGboost:

Tuned Hyperparameters

```python
# Function to train and evaluate a Gradient Boosting model with hyperparameter tuning
def xgboost(X_train, X_test, y_train, y_test):
    # Define the parameter grid
    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 4, 5],
        'subsample': [0.8, 0.9, 1.0],
        'min_samples_split': [2, 5, 10]
    }
```

Optimal Values:

```
Best Parameters found by GridSearchCV:
{'learning_rate': 0.2, 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 300, 'subsample': 0.8}
```

```
Accuracy: 0.93
```

5.2. Performance Metrics Comparison Report

Decision Tree:

Baseline Metric :

```
Confusion Matrix:
[[8642  638]
 [ 427 8835]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      9280
           1       0.93      0.95      0.94      9262

    accuracy                           0.94     18542
   macro avg       0.94      0.94      0.94     18542
weighted avg       0.94      0.94      0.94     18542
```

Optimized Metric:

```
Confusion Matrix:
[[8668  612]
 [ 453 8809]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      9280
           1       0.94      0.95      0.94      9262

    accuracy                           0.94     18542
   macro avg       0.94      0.94      0.94     18542
weighted avg       0.94      0.94      0.94     18542

Accuracy: 0.94
```

Random Forest:

Baseline Metric :

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

           0       0.96      0.96      0.96      9280
           1       0.96      0.96      0.96      9262

    accuracy                           0.96     18542
   macro avg       0.96      0.96      0.96     18542
weighted avg       0.96      0.96      0.96     18542

Accuracy: 0.96
```

```
Confusion Matrix:
[[8892  388]
 [ 403 8859]]
```

Optimized Metric:

```
Confusion Matrix:
[[8959  321]
 [ 421 8841]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      9280
           1       0.96      0.95      0.96      9262

    accuracy                           0.96     18542
   macro avg       0.96      0.96      0.96     18542
weighted avg       0.96      0.96      0.96     18542

Accuracy: 0.96
```

KNN:

Baseline Metric :

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

           0       0.98      0.84      0.90      9280
           1       0.86      0.98      0.92      9262

    accuracy                           0.91     18542
   macro avg       0.92      0.91      0.91     18542
weighted avg       0.92      0.91      0.91     18542

Accuracy: 0.91
```

```
Confusion Matrix:
[[7796 1484]
 [ 153 9109]]
```

Optimized Metric:

```
Confusion Matrix:
[[8294  986]
 [ 222 9040]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.89      0.93      9280
           1       0.90      0.98      0.94      9262

    accuracy                           0.93     18542
   macro avg       0.94      0.93      0.93     18542
weighted avg       0.94      0.93      0.93     18542

Accuracy: 0.93
```

XGboost:

Baseline Metric :

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

           0       0.89      0.84      0.86      9280
           1       0.85      0.90      0.87      9262

    accuracy                           0.87     18542
   macro avg       0.87      0.87      0.87     18542
weighted avg       0.87      0.87      0.87     18542

Accuracy: 0.87
```

```
Confusion Matrix:
[[7755 1525]
 [ 924 8338]]
```

Optimized Metric:

```
Confusion Matrix:
[[8678  602]
 [ 695 8567]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93      9280
           1       0.93      0.92      0.93      9262

    accuracy                           0.93     18542
   macro avg       0.93      0.93      0.93     18542
weighted avg       0.93      0.93      0.93     18542

Accuracy: 0.93
```

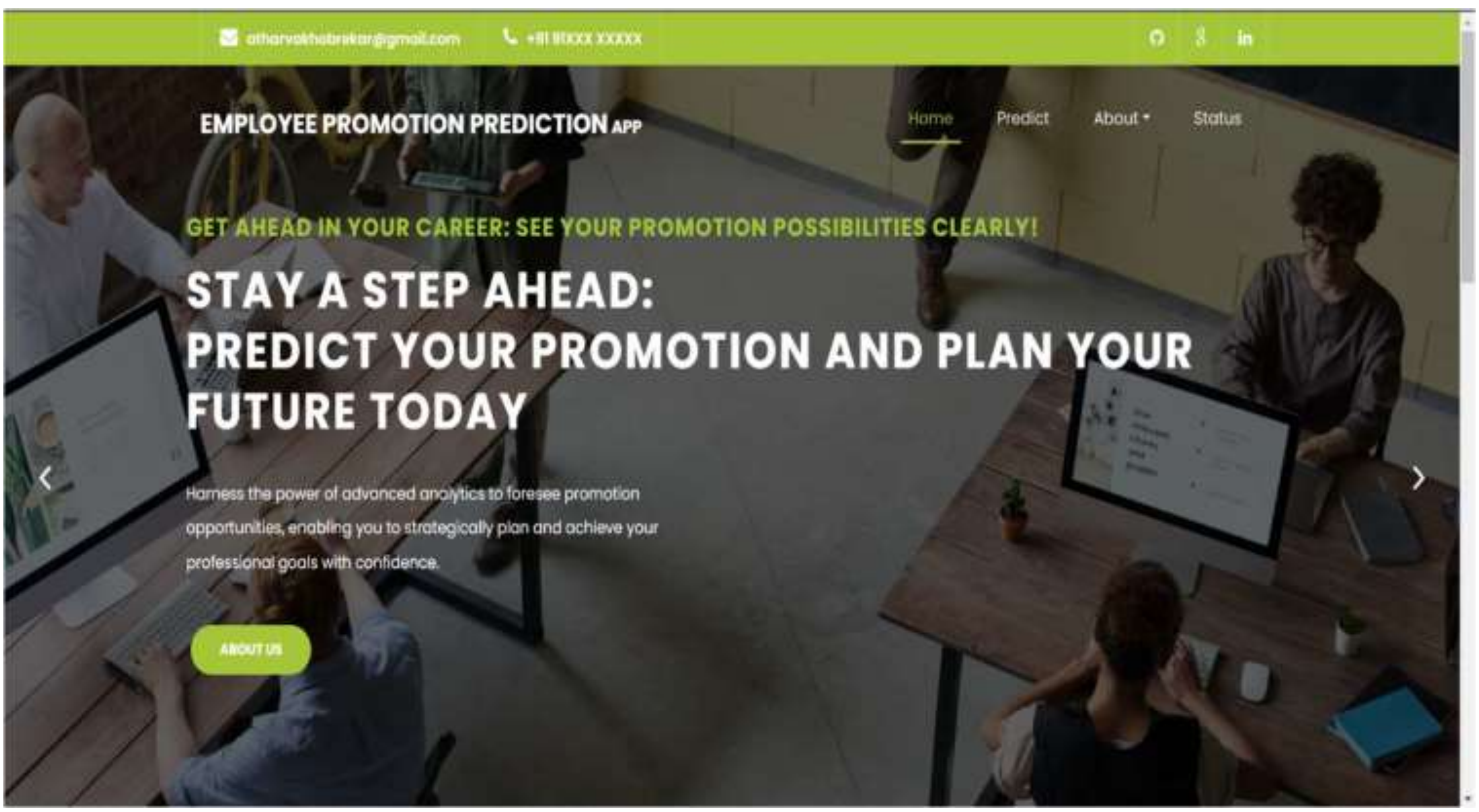
5.3. Final Model Selection Justification

Model : Random Forest

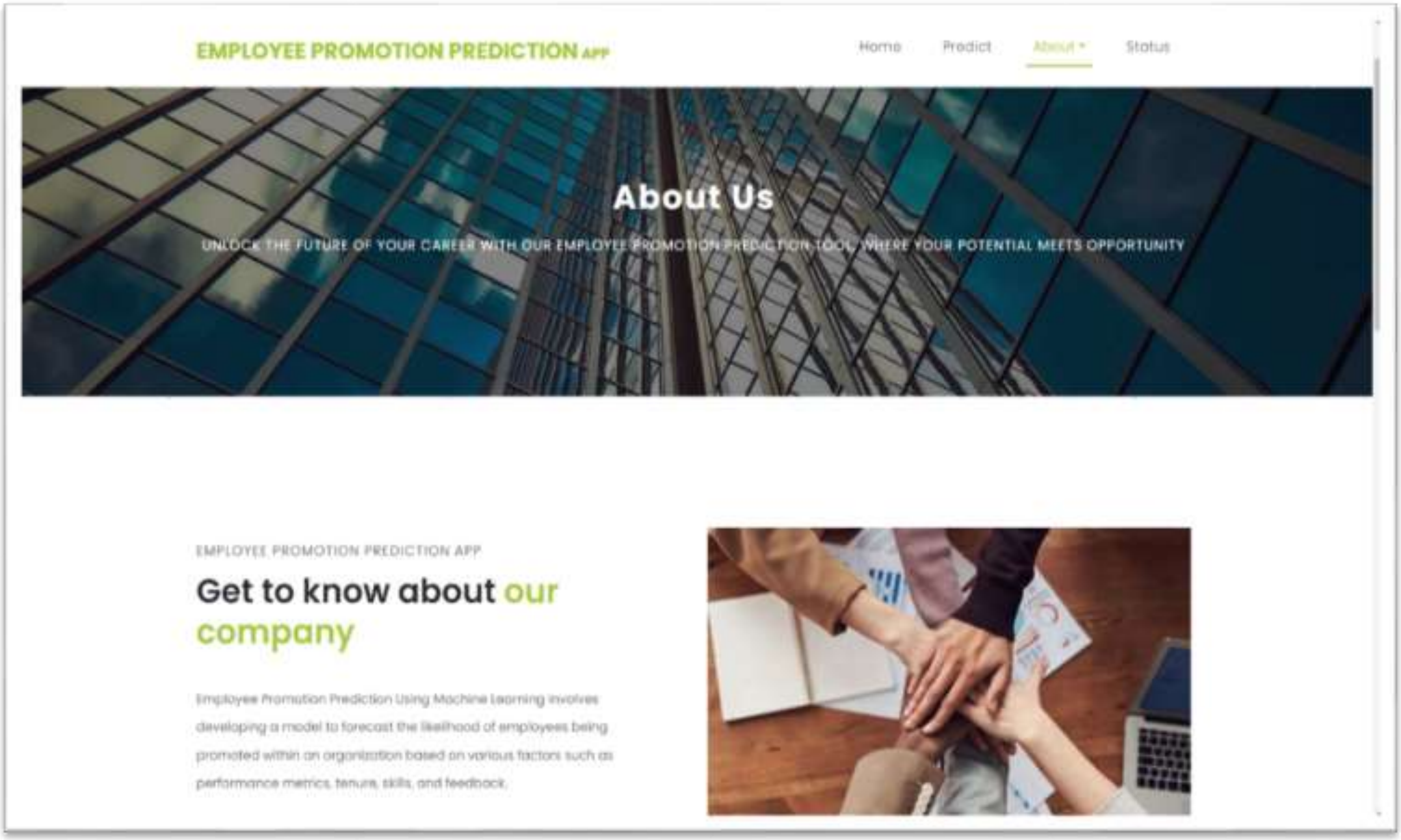
Reasoning : I chose the Random Forest model for predicting employee promotions due to its highest accuracy of 95%, outpacing Decision Tree, KNN, and Gradient Boosting. Its robustness, ability to handle overfitting, and insights into feature importance, combined with its capability to manage complex, 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

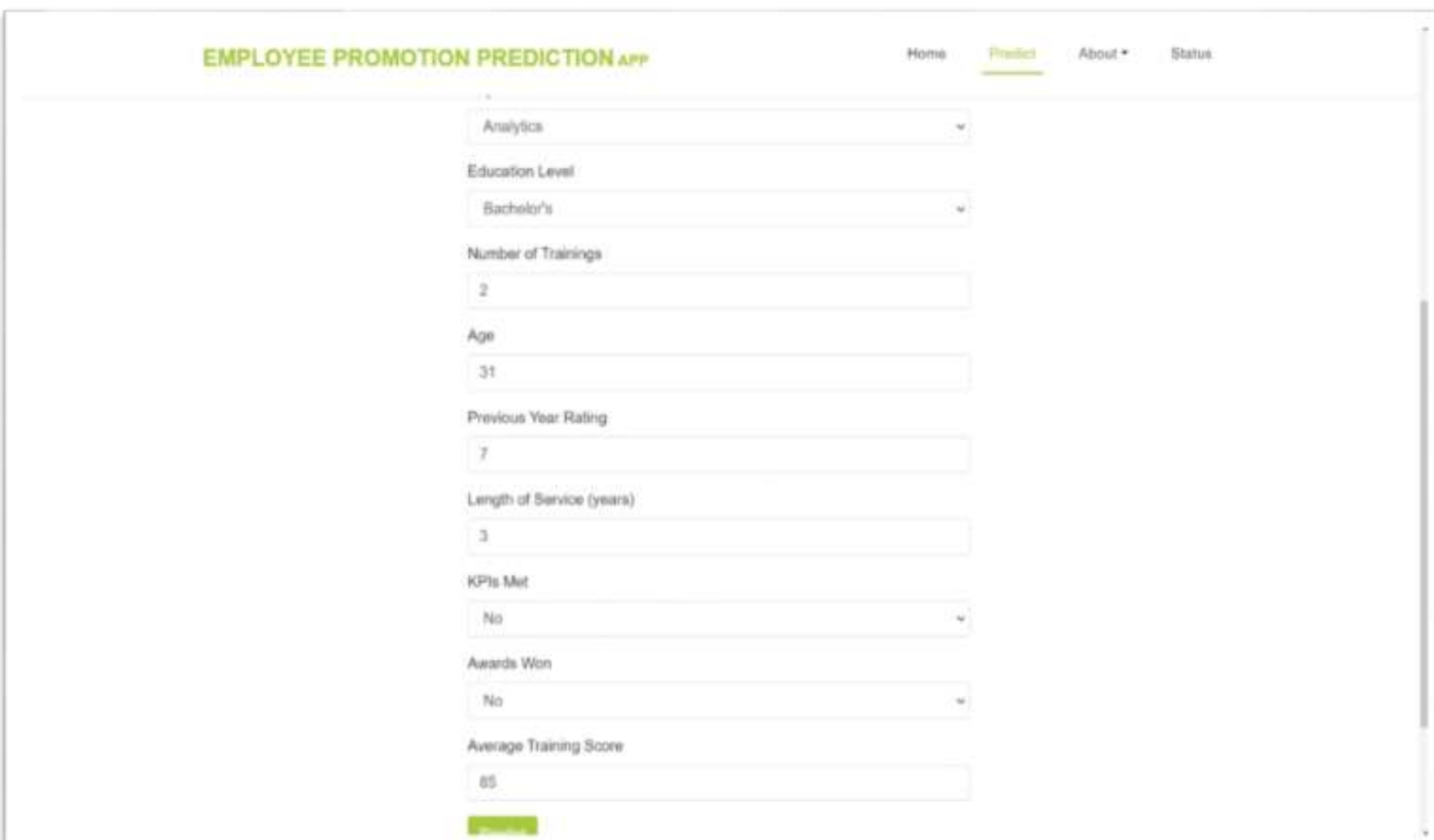
6.1. Output Screenshots **Home Page :**



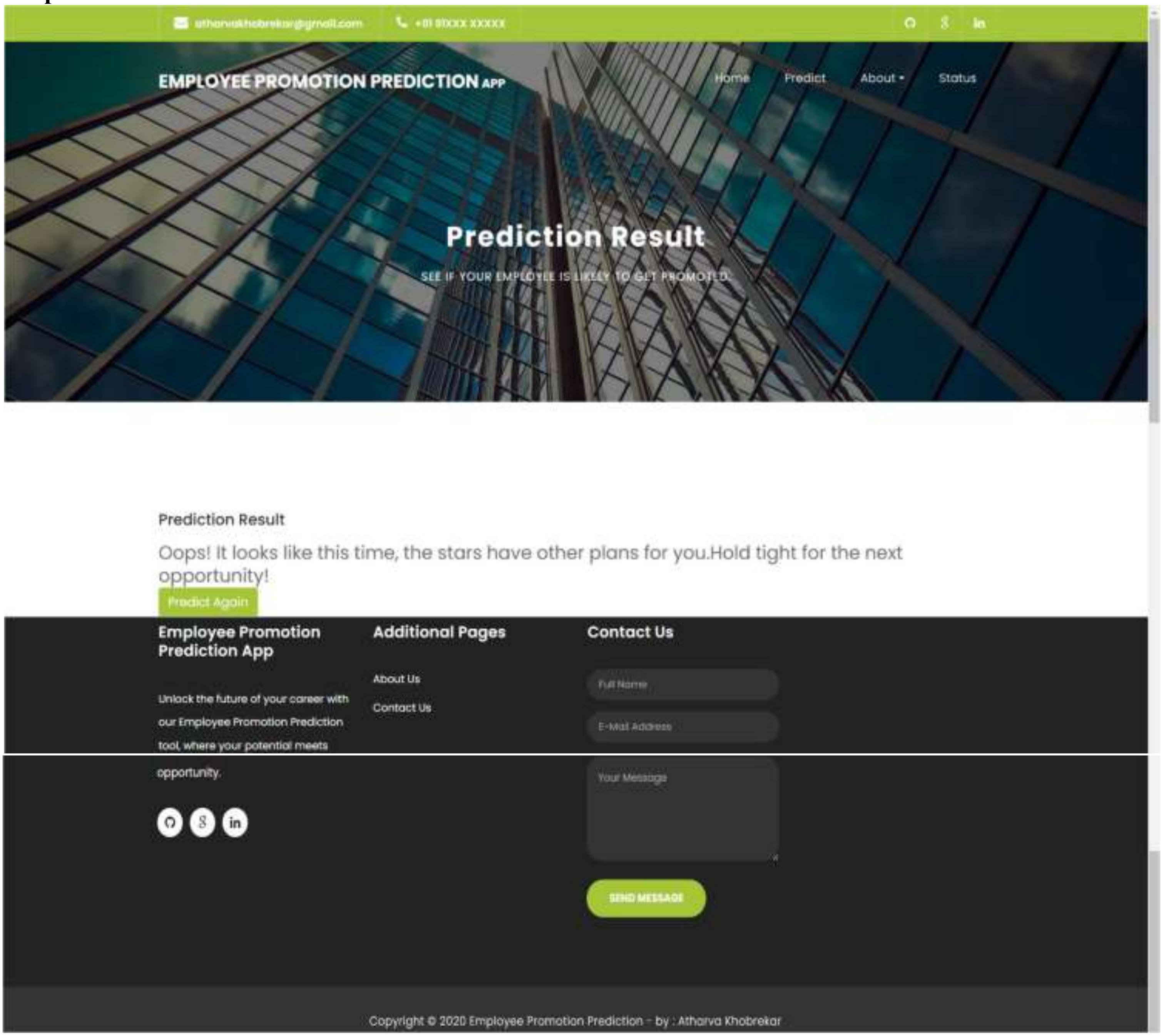
About Page :



Input 1 :

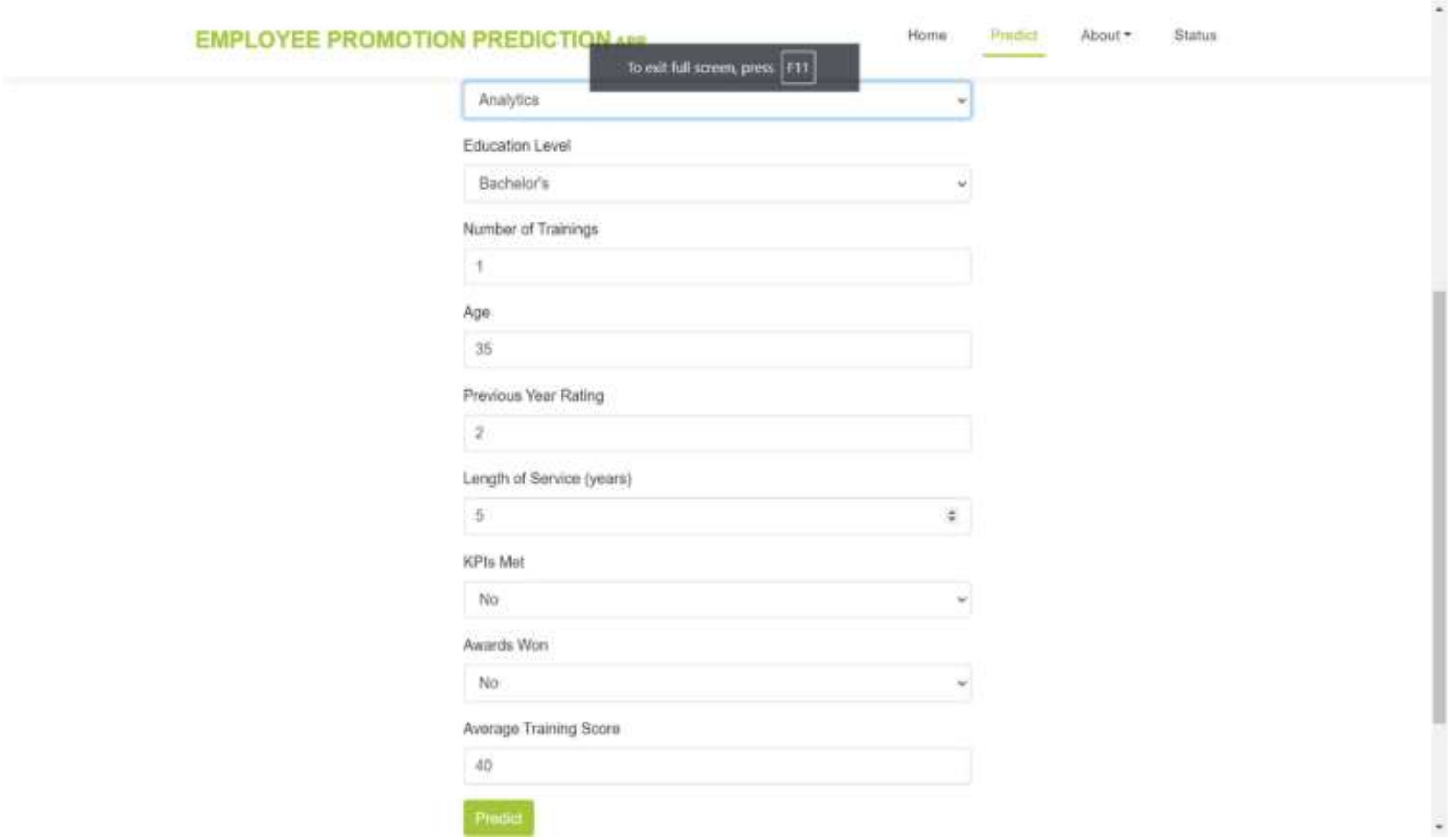


Output1:- 1:

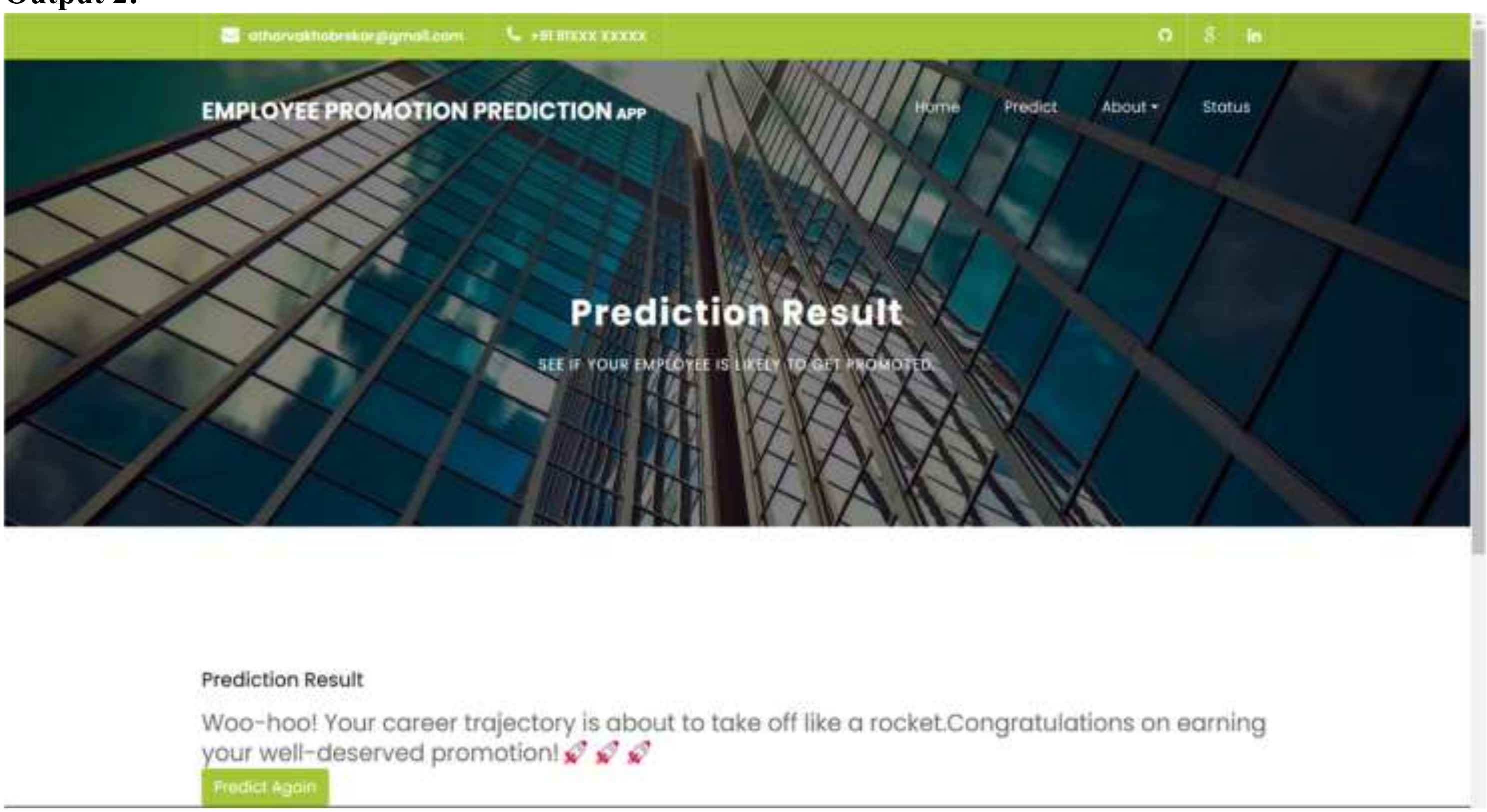


EMPLOYEE PROMOTION PREDICTION APP
Home
Predict
About
Status
Prediction Result
SEE IF YOUR EMPLOYEE IS LIKELY TO GET PROMOTED.
Prediction Result
Oops! It looks like this time, the stars have other plans for you.Hold tight for the next opportunity!
Predict Again
Employee Promotion Prediction App
Unlock the future of your career with our Employee Promotion Prediction tool, where your potential meets opportunity.
Additional Pages
About Us
Contact Us
Contact Us
Full Name
E-Mail Address
Your Message
SEND MESSAGE
Copyright © 2020 Employee Promotion Prediction - by : Atharva Khobrekar

Input2:-



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

https://github.com/FS22AI014/Human-Resource-Management-Predicting-EmployeePromotions-using-Machine-Learning