Human Resource Management: Predicting Employee Promotions using Machine Learning

Hand-Out:
1. Tarun Yadav
2. Kushagra Sarin

Smartbridge
www.smartbridge.com

Human Resource Management: Predicting Employee Promotions using Machine Learning

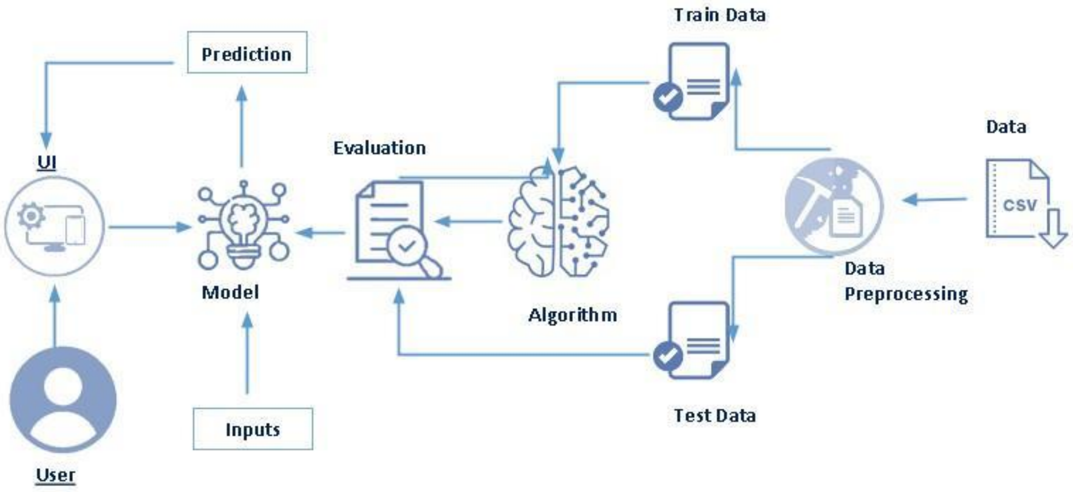
Employee Promotion Prediction Using Machine Learning involves developing a model to forecast the likelihood of employees being promoted within an organization based on various factors such as performance metrics, tenure, skills, and feedback. This project aims to enhance workforce management strategies 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:



Project Flow:

- The user interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Drop unwanted features
  - Checking for null values
  - Remove negative data
  - Handling outlier
  - Handling categorical data
  - Handling Imbalanced data
  - Splitting data into train and test
- Model building
  - Import the model building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating performance of the model
  - Save the model

- Application building
  - Create an HTML file
  - Build python code
  - Run the Application

Prior Knowledge:

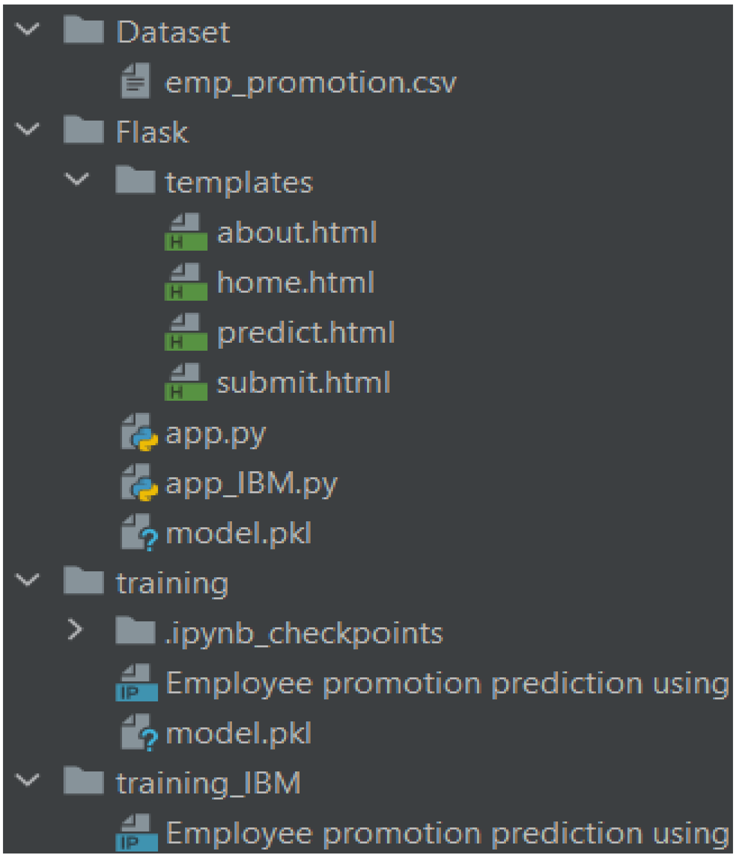
One should have knowledge of the following Concepts

Please refer to the videos below to gain sufficient required knowledge to complete the project.

- **Supervised and unsupervised learning:** https://youtu.be/kE5QZ8G_78c
- **Regression Classification and Clustering :** https://youtu.be/6za9_mh3uTE
- **Random Forest Classifier:** https://youtu.be/nxFG5xdpDto
- **Ensemble Technique:** https://youtu.be/KIOeZ5cFZ50
- **Decision Tree Classifier:** https://youtu.be/qDcl-FRnwSU
  - **KNN**: Refer the https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - **Xgboost**: Refer the https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
  - **Evaluation metrics:** Refer the https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask:** https://youtu.be/lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.

- For IBM deployment app_IBM.py file is used.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and training_ibm folder contains IBM model training files.

Milestone 1: Data collection

Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used drug200.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

https://drive.google.com/file/d/14eQR1VWHwuomPaXdKuIkZEpSstvav8Db/view?usp=sharing

Milestone 2: Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 2.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as fivethirtyeight.

To know about the packages refer the link given on pre requisites.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import pickle
from sklearn.metrics import classification_report,confusion_matrix
plt.style.use('fivethirtyeight')
pd.set_option('display.max_rows',None)
```

Activity 2.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

```
# 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	

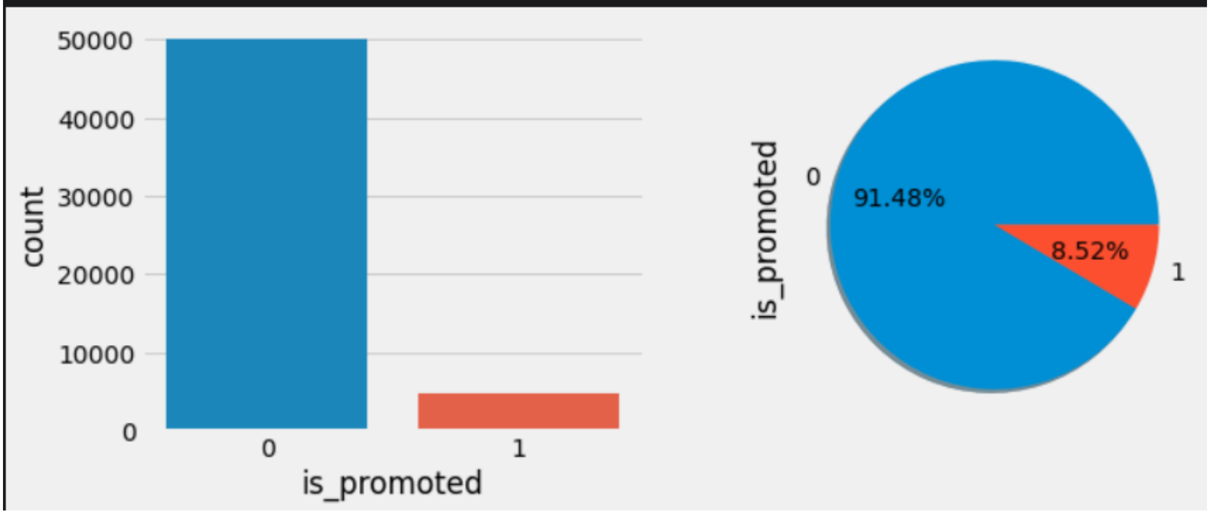
Activity 2.3: Univariate analysis

n simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as pie plot, box plot and count plot.

- Count plot and pie plot are used on the target variable. From the below image, we identified our data is imbalanced. 91% of the employees are not promoted. To get better model performance, imbalanced data should be converted to balanced data. Handling imbalanced data will be discussed on data pre processing.

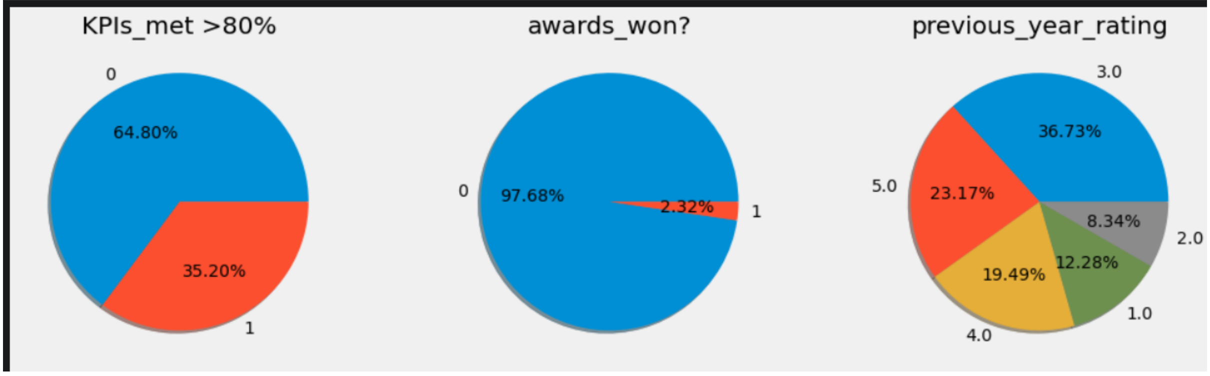
```
# Data is imbalanced

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()
```



- A pie plot is used on value counts() of the required features. From the below graph, we get a clear understanding that 97.68% of employees have not won any awards. Around 65% of employees have KPIs > 80%. More than 75% of employees have a previous year rating > 3.0. Instead of pie plot count plot can also be used.

```
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()
```

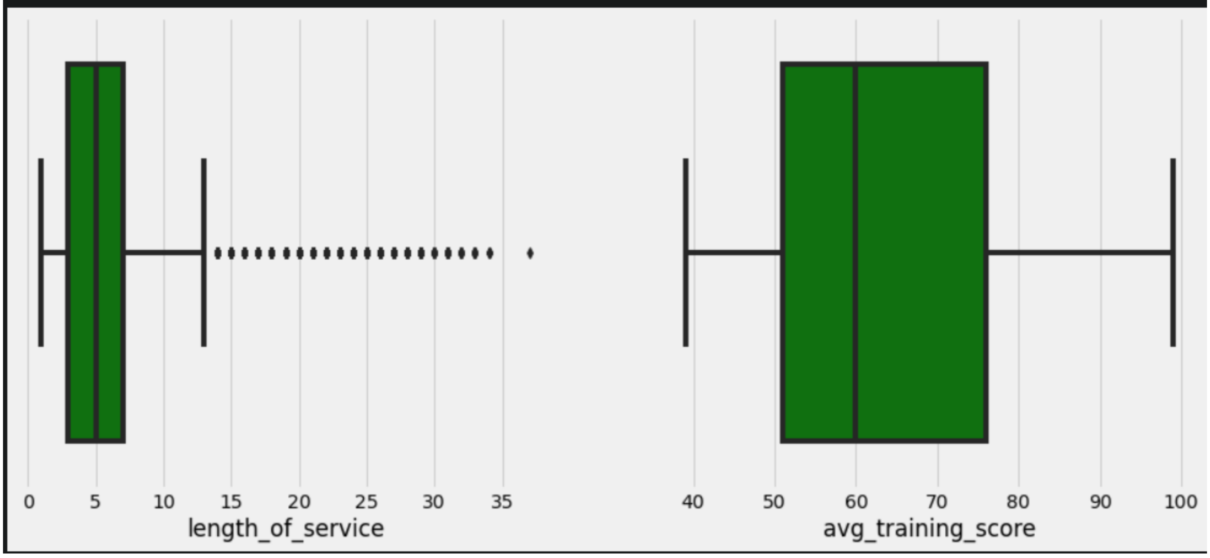


- Box plot is used on the length of service and average training score feature. Length of services feature has more outliers. The model should not be built without handling the outliers. Here, outliers are handled by the capping method. Capping will be discussed on data pre-processing.

```
# 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'>
```



Activity 2.4: Multivariate analysis

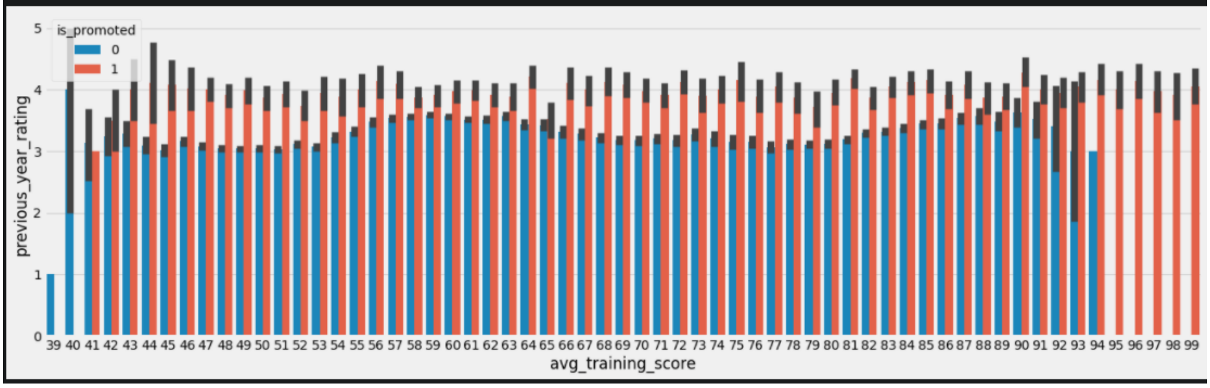
In simple words, multivariate analysis is to find the relation between multiple features. Here we have used barplot from seaborn package.

- Three features are passed as parameters for barplot(). A clear pattern is understandable from the below plot. Employees with an average training score greater than 95 and a previous year rating greater than 3 got promotions (100%).

```
""" 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'>
```



Activity 2.5: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
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	

Milestone 3: Data Pre-processing

As we have understood how the data is. Lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

In the data frame, head() function is used to display the first 5 data. Our dataset has employee id (unique values), department (totally 9 dept.), region (location), education, gender, recruitment channel, age, no. of trainings, previous year ratings, length of service, KPIs, award won, average training score and is_promoted (target variable) columns.

```
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	

Activity 3.1: Drop unwanted features

We are building the model to predict the promotion of employees. Employee id is not useful for predicting employee promotion. Generally, based on the performance promotion is given. No organizations will promote their employees by gender, region, and recruitment channel. So, these features are removed from the dataset

```
""" 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)
```

Activity 3.2: Checking for null values

For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that education column and previous year rating column has null values.

```
df.isnull().sum()

department                 0
education               2409
no_of_trainings            0
age                        0
previous_year_rating    4124
length_of_service          0
KPIs_met >80%              0
awards_won?                0
avg_training_score         0
is_promoted                0
dtype: int64
```

- Let’s handle the null values.
- For the education feature and previous year rating feature, null values are replaced with their respective mode[0] values. These two features don’t have continuous values. So, the mode value is replaced. The most frequent repeated value for education column is bachelor’s and for previous year rating is 3.

```
# 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
```

Activity 3.3: Remove negative data

Employees with poor performance got promoted. It affects model performance. So, negative value should be removed.

- Here list comprehension is used to find the negative data.
- Negative data: Employees with no awards, previous year rating was 1.0, KPIs less than 80% and average training score is less than 60.
- Now, negative data is 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)
```

Activity 3.4: Handling outliers

With the help of boxplot, outliers are visualized (refer activity 3 univariate analysis). And here we are going to find upper bound and lower bound of Na_to_K feature with some mathematical formula.

- To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.
- If outliers are removed, we lose more data. It will impact model performance.
- Here removing outliers is impossible. So, the capping technique is used on outliers.
- Capping: Replacing the outliers with upper bound values.

```
# Handling outliers

q1 = np.quantile(df['length_of_service'],0.25)
q3 = np.quantile(df['length_of_service'],0.75)

IQR = q3-q1

upperBound = (1.5*IQR)+q3
lowerBound = (1.5*IQR)-q1

print('q1 :',q1)
print('q3 :',q3)
print('IQR :',IQR)
print('Upper Bound :',upperBound)
print('Lower Bound :',lowerBound)
print('Skewed data :',len(df[df['length_of_service']>upperBound]))
```

```
q1 : 3.0
q3 : 7.0
IQR : 4.0
Upper Bound : 13.0
Lower Bound : 3.0
Skewed data : 3489
```

```
""" Here outliers can't be removrd. employee with higher length of services has higher promotion percentage.
    So, capping is done on this feature."""

pd.crosstab([df['length_of_service']>upperBound],df['is_promoted'])
```

is_promoted	0	1
length_of_service		
False	46885	4432
True	3255	234

```
# Capping

df['length_of_service']=[upperBound if x>upperBound else x for x in df['length_of_service']]
```

Activity 3.5: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using feature mapping and label encoding.

- In our project, categorical features are education and department feature. Feature mapping on education is done by replace() function.
- Label encoder is initialized and department feature is passed as parameter for fit_transform() function. Label encoding uses alphabetical ordering. In department feature we have 9 categories. Those categories are labelled in alphabetical order.

```
# Feature mapping is done on education column

df['education']=df['education'].replace(("Below Secondary","Bachelor's","Master's & above"),(1,2,3))
```

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

Activity 3.6: Handling Imbalanced data

From the activity - univariate analysis we found our data is imbalanced. Now let's split the dataset into x and y. Independent features are passed to x variable and dependent feature is passed to y variable. Then, to handle imbalanced data resampling are done with SMOTE.

- Import the SMOTE function from imblearn package.
- Create a variable and initialize smote() function. Now resampling is done with fit_resample() function
- SMOTE : Refer this link to lean more about SMOTE

```
# Splitting data and resampling it

x = df.drop('is_promoted',axis=1)
y = df['is_promoted']
print(x.shape)
print(y.shape)
```

```
(54806, 9)
(54806,)
```

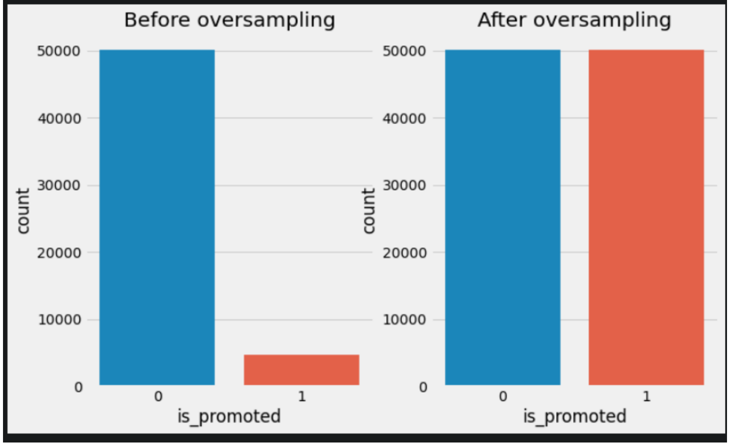
```
from imblearn.over_sampling import SMOTE
```

```
sm =SMOTE()
x_resample, y_resample = sm.fit_resample(x,y)
```

- Refer the below diagram to visualize the result of smote technique.

```
plt.figure(figsize=(10,6))
plt.subplot(121)
sns.countplot(y)
plt.title('Before oversampling')
plt.subplot(122)
sns.countplot(y_resample)
plt.title('After oversampling')
```

```
Text(0.5, 1.0, 'After oversampling')
```



Activity 3.7: Splitting data into train and test

Now let's split the Dataset into train and test sets. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x_resample, y_resample, test_size, random_state.

For deep understanding refer this link

```
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 4: Model Building

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.

Activity 4.1: Decision tree model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
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))
```

Activity 4.2: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
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))
```

Activity 4.3: KNN model

A function named KNN is created and train and test data are passed as the parameters. Inside the function, KNeighborsClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
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))
```

Activity 4.4: Xgboost model

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
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))
```

Activity 4.4: Compare the model

For comparing the above four models compareModel function is defined.

```
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)
```

After calling the function, the results of models are displayed as output. From the four model random forest and decision tree is performing well. From the below image, we can see the accuracy of the models. Both models have 95% and 93% accuracy. Random forest model accuracy is high. And from confusion matrix random forest has higher number of true positive and true negative. So, here random forest is selected and evaluated with cross validation. Additionally, we can tune the model with hyper parameter tuning techniques. But here we have not used it.

To get deep knowledge in confusion matrix and classification report refer the below links:

Link1

Link2

```
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
```

Activity 4.5: Evaluating performance of the model and saving the model

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x_resample, y_resample, cv (as 5 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Note: To understand cross validation, refer this link.

```
# 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'))
```

Milestone 5: Application building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script
- Run the application
- Output

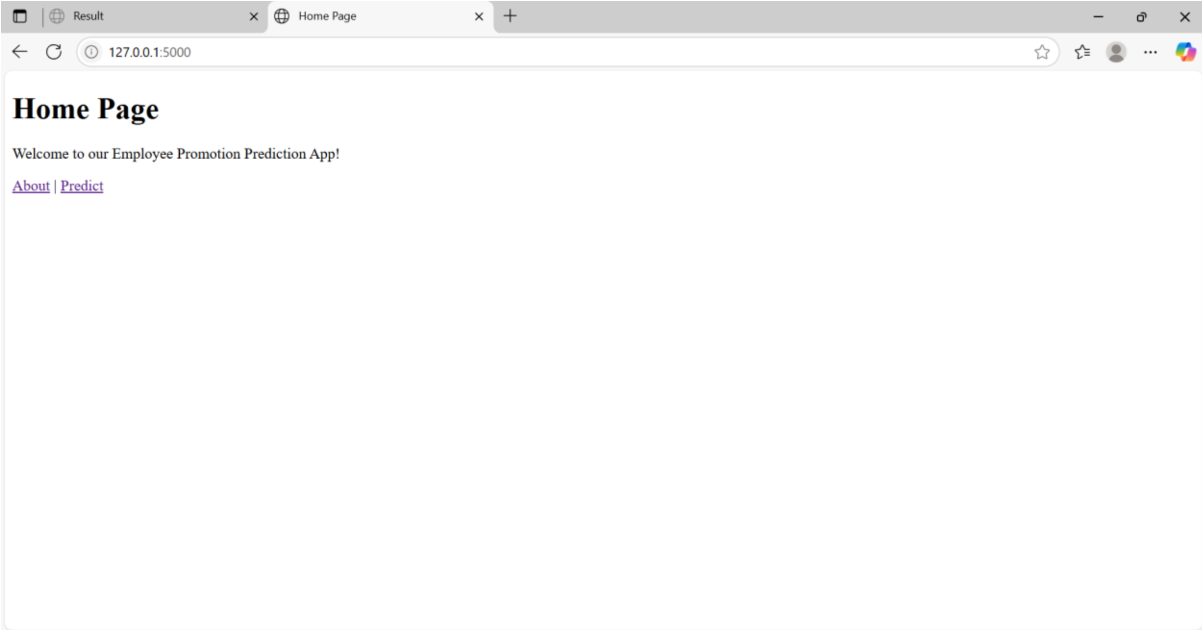
Activity 5.1: Building Html Pages

For this project, create three HTML files namely

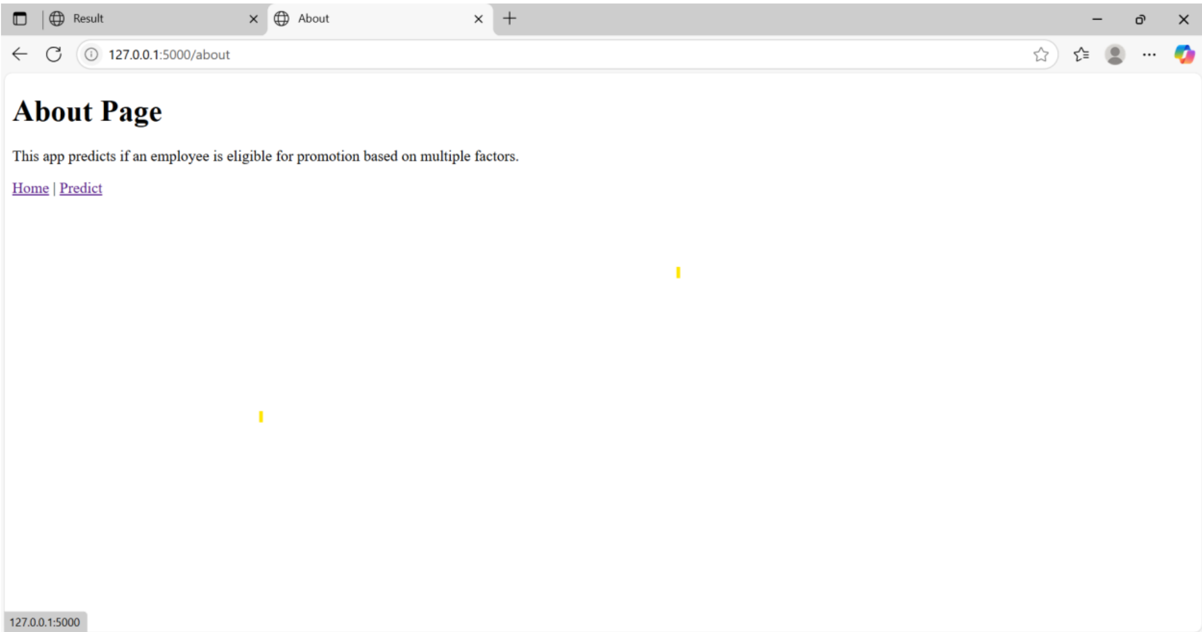
- home.html
- about.html
- predict.html
- submit.html

and save them in templates folder.

Let’s see how our home.html page looks like:

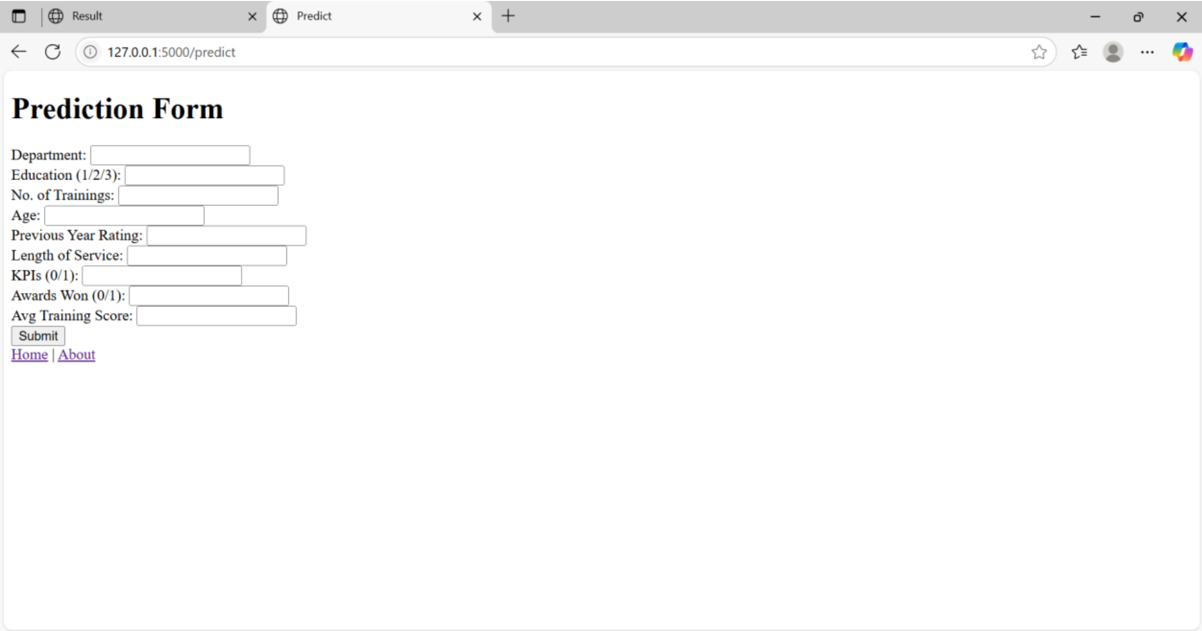


Let’s see how our about.html page looks like:



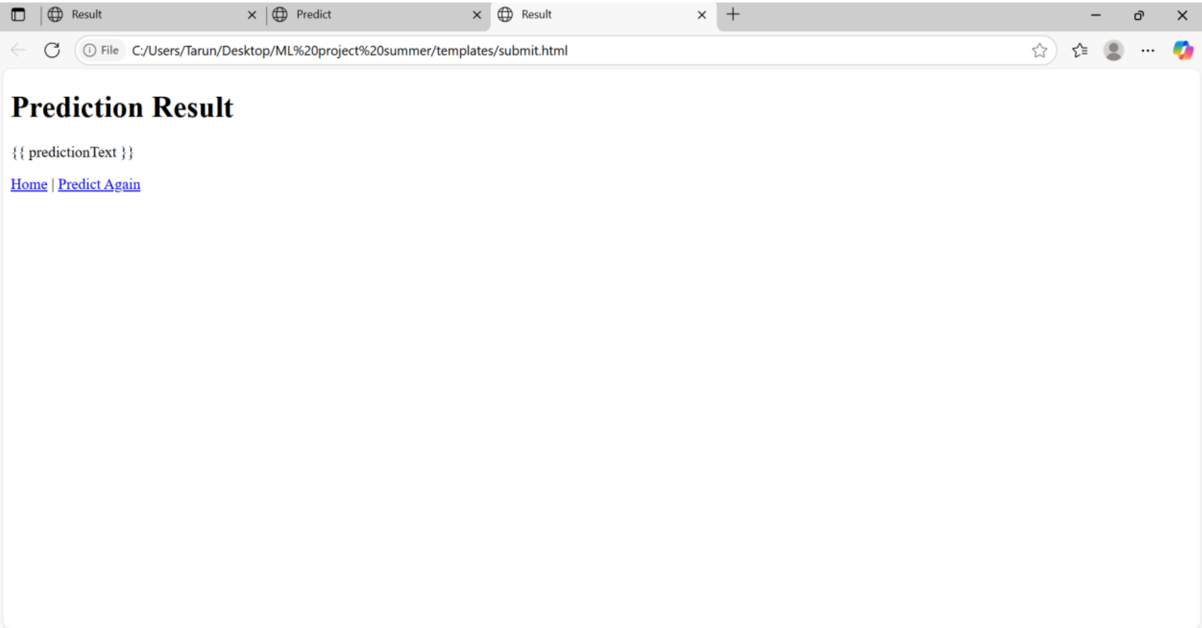
Now when you click on predict button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:



Now when you click on submit button from left bottom corner you will get redirected to submit.html

Lets look how our submit.html file looks like:



Activity 5.2: Build Python code

Import the libraries

Pickle: Pickle is a module in Python used for serializing and de-serializing Python objects.

Flask: Refer prior knowledge section mentioned above.

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

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```python
model = pickle.load(open('model.pkl', 'rb'))

app = Flask(__name__)
```

Render HTML page:

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


@app.route('/home')
def home1():
    return render_template('home.html')


@app.route('/about')
def about():
    return render_template('about.html')


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

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the predict html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```python
@app.route('/pred', methods=['POST'])
def pred():
    department = request.form['department']
    education = request.form['education']
    if education == '1':
        education = 1
    elif education == '2':
        education = 2
    else:
        education = 3
    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']
    if KPIs == '0':
        KPIs = 0
    else:
        KPIs = 1
    awards_won = request.form['awards_won']
    if awards_won == '0':
        awards_won = 0
    else:
        awards_won = 1
    avg_training_score = request.form['avg_training_score']
    total = [[department, education, no_of_trainings, age, float(previous_year_rating), float(length_of_service),
              KPIs, awards_won, avg_training_score]]
    prediction = model.predict(total)
    if prediction == 0:
        text = 'Sorry, you are not eligible for promotion'
    else:
```

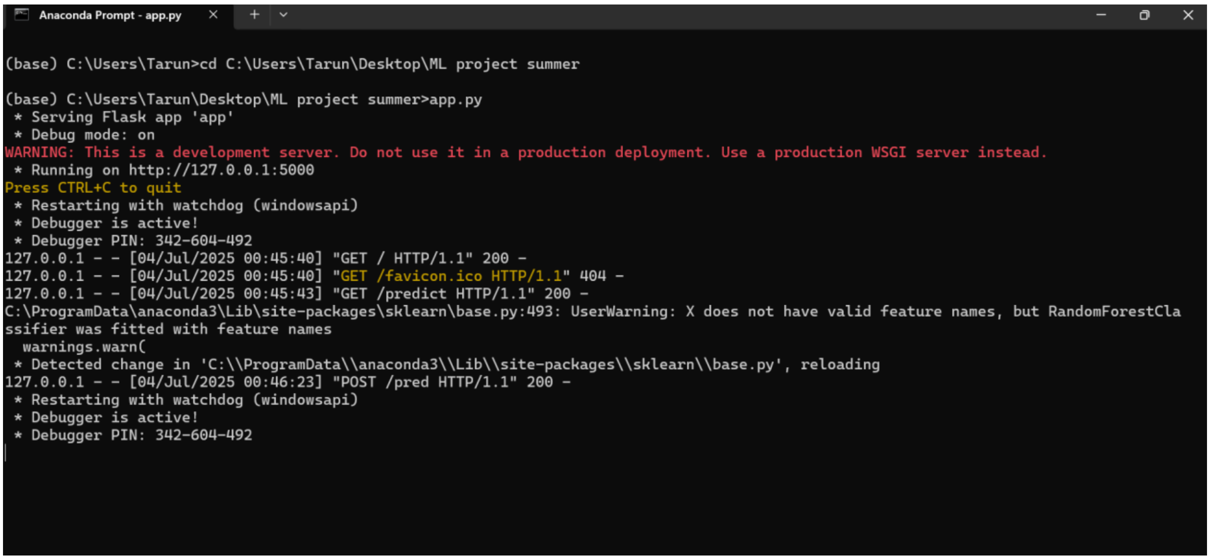
Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

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

Activity 5.3: Run the application

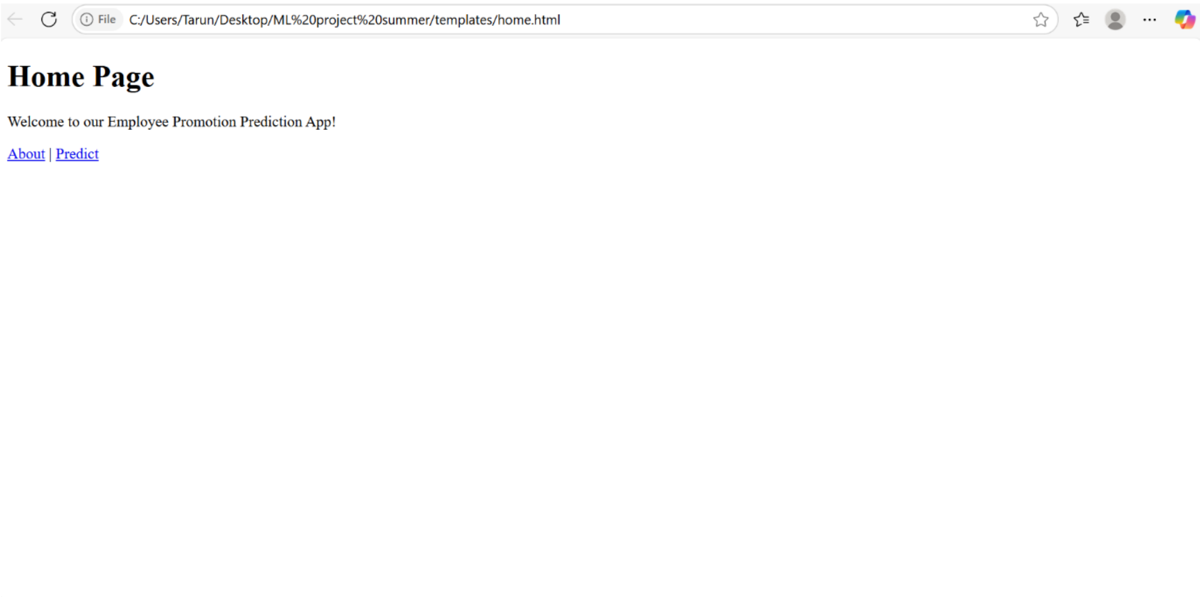
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



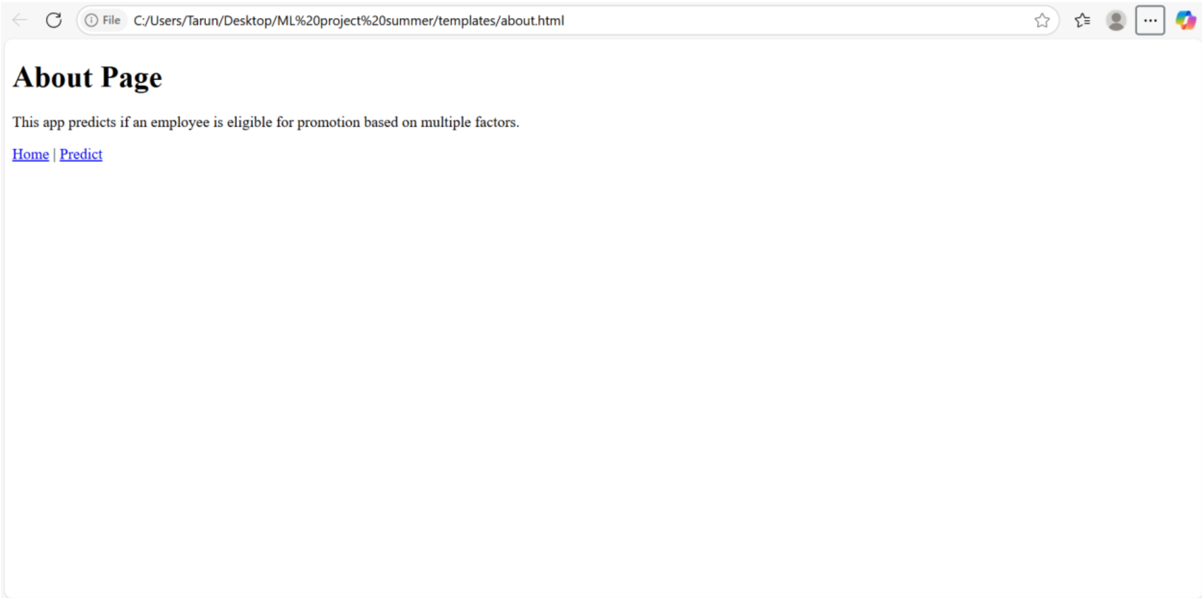
Now paste the URL on the browser, you will redirect to home.html page. Let's look our home page

Activity 5.4:Output

Home page

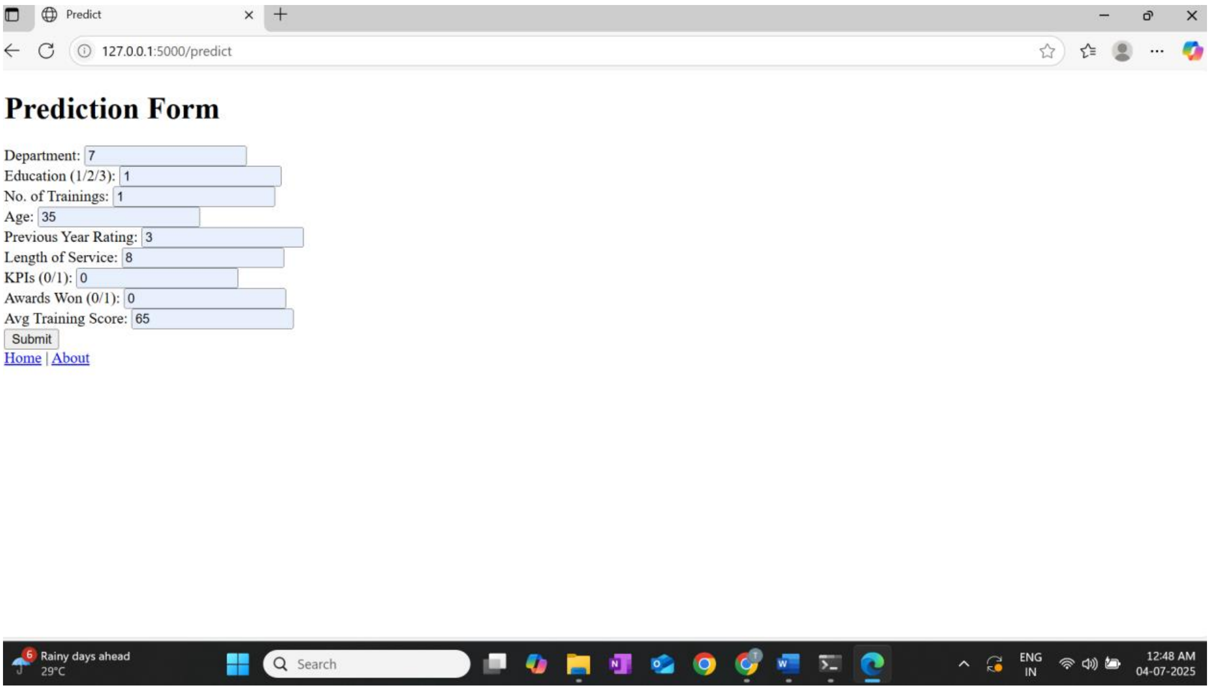


To know about the project click on About button on right top corner. Now it will redirect to about.html page

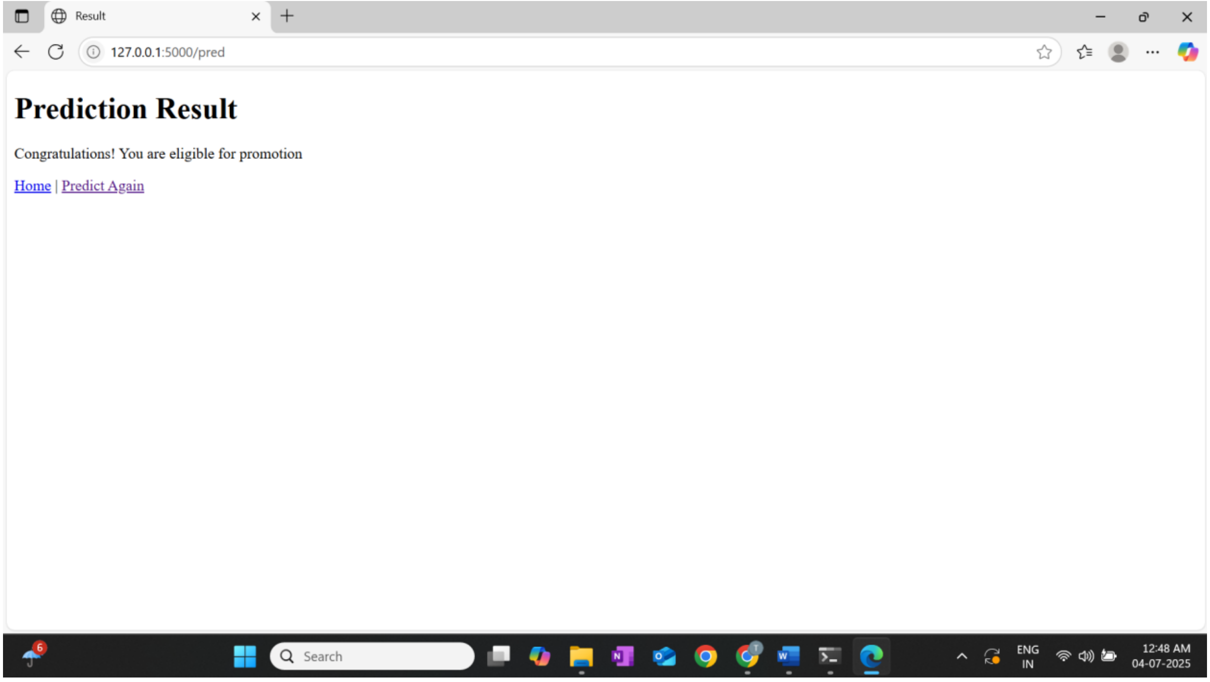


To predict your promotion click on predict button on right top corner. It will redirect to predict.html page. Now give your inputs and click on submit button. Output will be displayed in submit.html page.

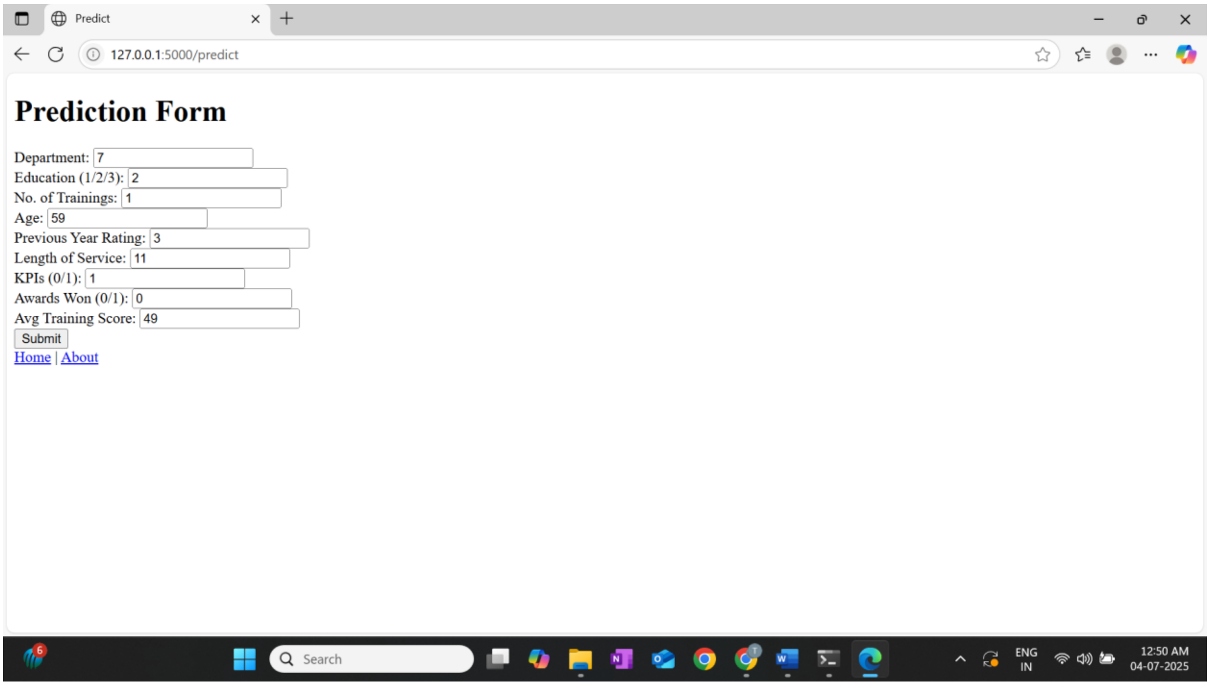
Input 1:



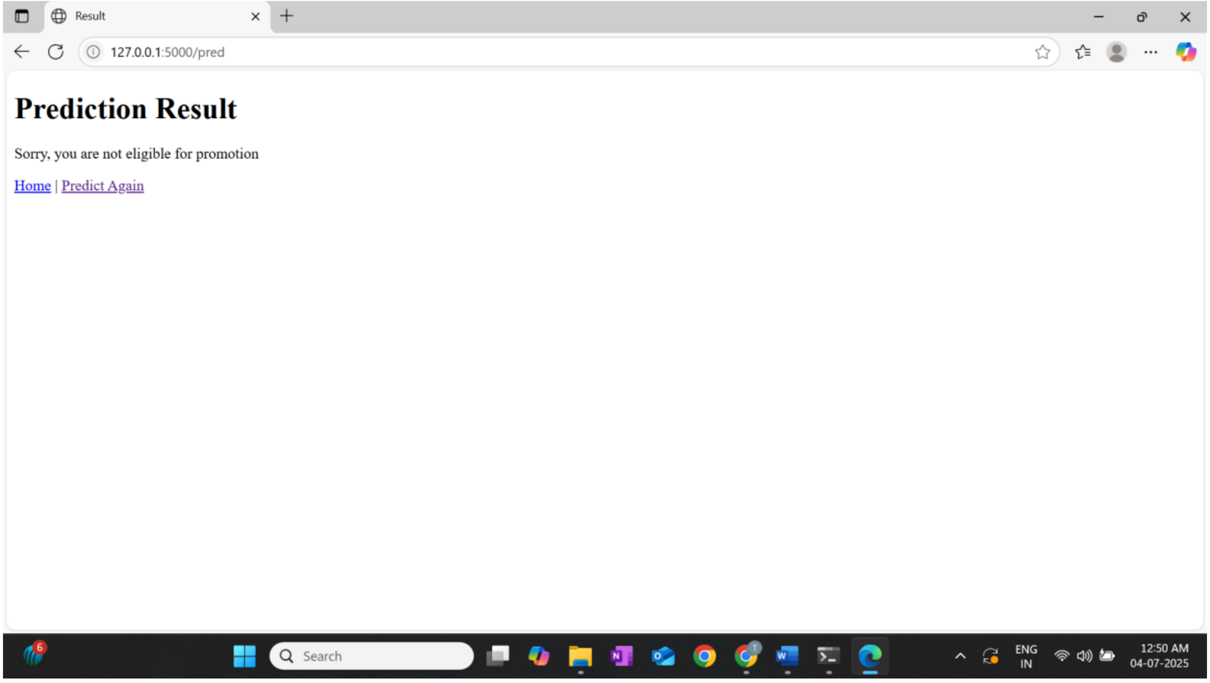
Output 1:



Input 2:



Output 2:



Conclusion:

The Employee Promotion Prediction project highlights the significant role of machine learning in optimizing human resource management processes. By leveraging employee data such as performance metrics, tenure, skill sets, and feedback, the model effectively identifies individuals with a high potential for promotion. This data-driven approach enables organizations to make fair, unbiased, and strategic promotion decisions, ultimately improving employee satisfaction and retention. Furthermore, the system supports leadership development and succession planning by recognizing talent early, contributing to long-term organizational success. Overall, this project offers a scalable solution that enhances workforce management, promotes merit-based advancement, and strengthens employee engagement across the organization.