EMPLOYEE PERFORMANCE PREDICTION
Machine Learning Project Report

Executive Summary:

The Employee Performance Prediction project uses machine learning to analyze employee data such as experience, training, feedback, and productivity to predict future performance. By applying supervised learning algorithms, the system identifies high and low performers with accuracy. It assists HR and management in making data-driven decisions for promotions, training, and workforce optimization, ultimately improving organizational efficiency.

Project Scenarios :

Scenario 1: Talent Retantion:

HR departments can use the machine learning predictions to identify high-performing employees at risk of attrition. By analyzing factors contributing to employee turnover and predicting performance trends, HR can implement targeted retention strategies, such as personalized career development plans or incentive programs, to retain top talent.

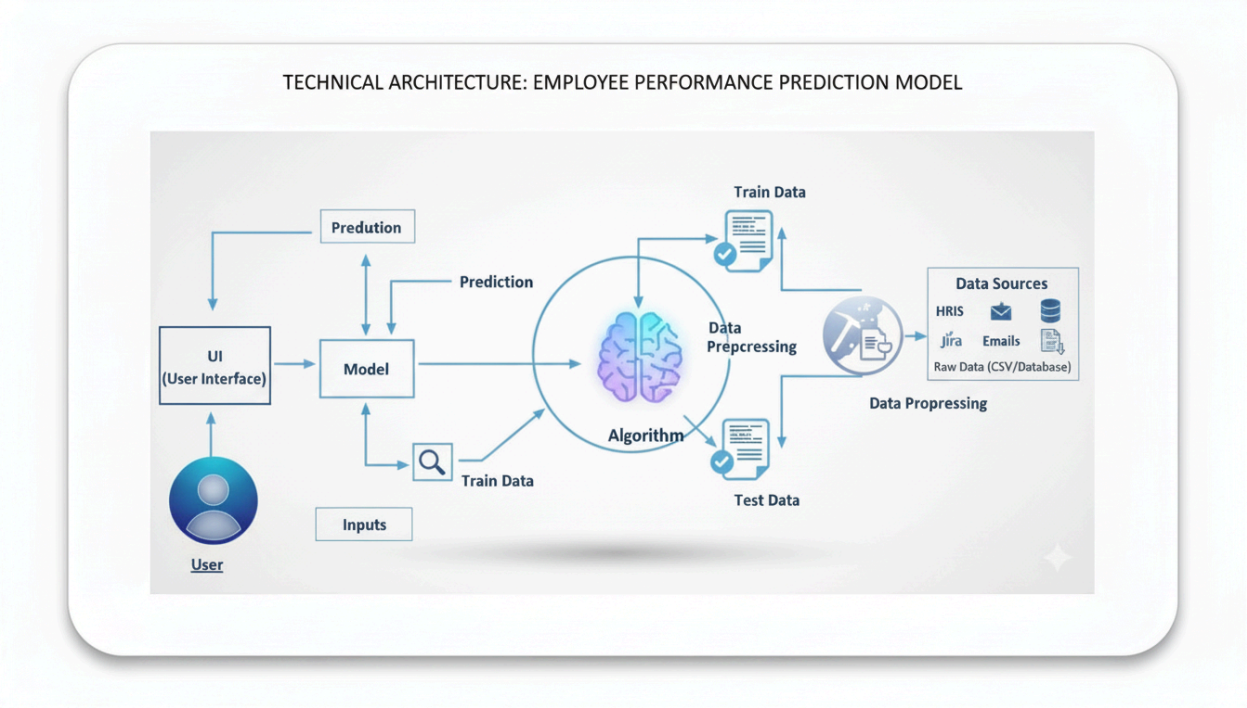
Scenario 2: Performance Improvement:

Managers and team leaders can leverage the predictions to identify areas where employees may need additional support or training. By understanding performance patterns and potential challenges, managers can provide timely coaching, resources, or skill development opportunities to enhance employee performance and productivity.

Scenario 3: Resource Allocation:

Organizations can optimize resource allocation by using machine learning predictions to match employees with projects or tasks that align with their strengths and capabilities. This ensures efficient utilization of talent, improves project outcomes, and enhances overall organizational performance.

Technical Architecture :



Project Flow :

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

Project Activities :

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**
  - Correlation analysis
  - Descriptive analysis

- **Data pre-processing**
    - Checking for null values
    - Handling Date & department column
    - Handling categorical 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 model
    - Save the model

- **Application Building**
    - Create an HTML file
    - Build python code

Prerequisites :

Software Requirements:

- **Anaconda Navigator and Visual Studio**
- **Python packages:**
    - numpy
    - pandas
    - scikit-learn
    - matplotlib
    - scipy
    - pickle-mixin
    - seaborn
    - Flask

Prior Knowledge Required:

- **ML Concepts**
    - Supervised learning
    - Unsupervised learning
    - Logistic Regression
    - Decision tree
    - Random forest
    - Evaluation metrics
    - Naive Bayes
    - KNN
    - Xgboost

Flask Basics

Milestone 1: Data Collection

Data collection is fundamental to machine learning, providing the raw material for training algorithms and making predictions. For the Employee Performance Prediction project, we utilized a comprehensive worker productivity dataset containing operational and resource metrics to forecast an employee's or team's actual performance.

Activity : Dataset Collection

The dataset contains 1197 records from a garment manufacturing setting, detailing daily team performance, time metrics (over_time, idle_time), and resource allocation (no_of_workers, incentive) to analyze their impact on actual_productivity.

The dataset was obtained from Kaggle.com

Link: https://www.kaggle.com/datasets/utkarshsarbahi/productivity-prediction-of-garment-employees

Column Details :

The columns included in the dataset are:

- date
- quarter
- department
- day
- team
- targeted_productivity
- smv (Standard Minute Value)
- wip (Work in Progress)
- over_time
- incentive
- idle_time
- idle_men
- no_of_style_change
- no_of_workers
- actual_productivity

Milestone 2 : Visualizing and analyzing data

Activity 1 : Importing the libraries

Import the necessary libraries as per our project requirement-

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# import MultiColumnLabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import pickle
```

✓ [3] < 10 ms

Activity 2 : Read the Dataset

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.

```
data =  pd.read_csv('../Dataset/garments_worker_productivity.csv')
data.head()
```

✓ [4] 72ms

5 rows 5 rows × 15 cols

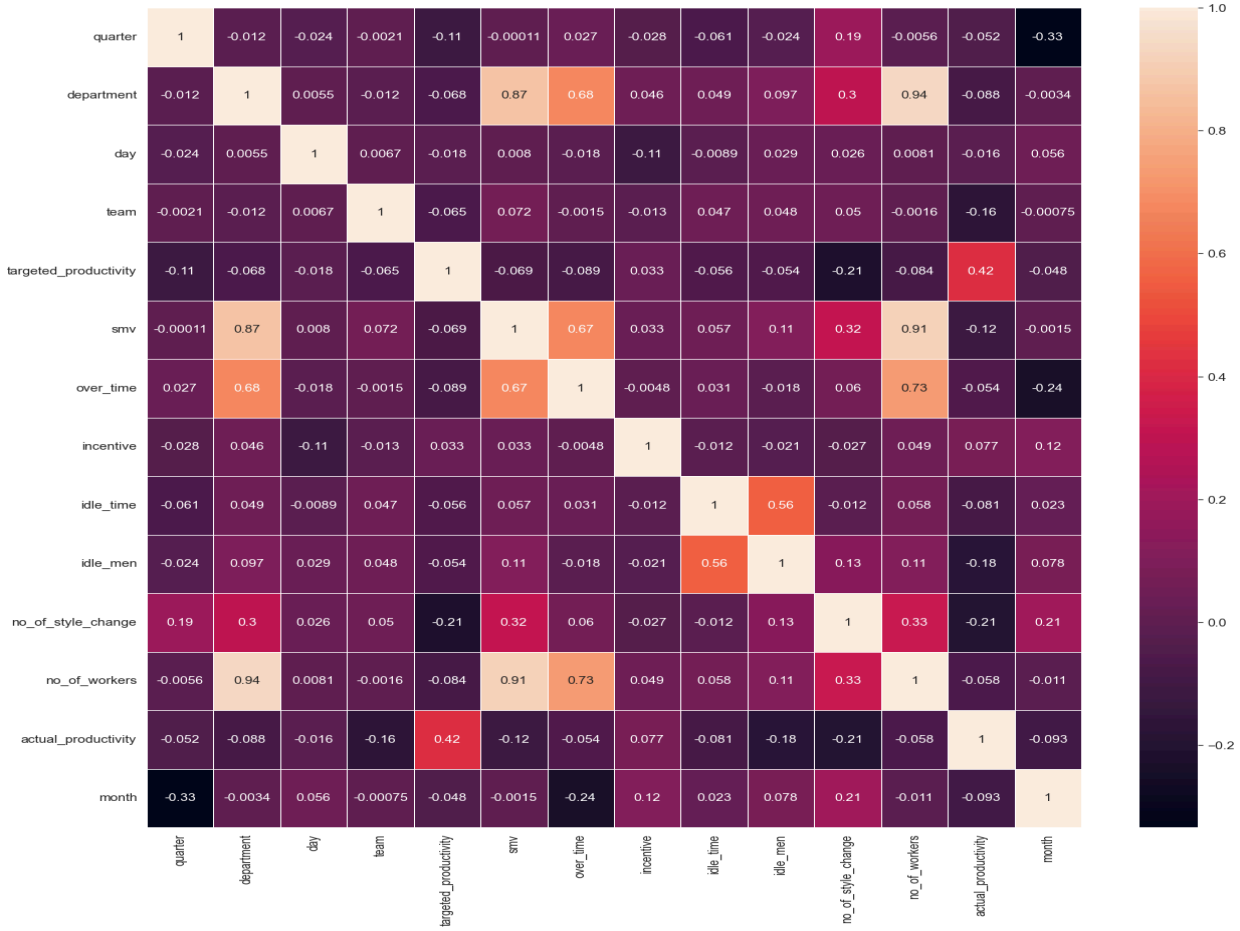
	date	quarter	department	day	team	targeted_productivity
0	1/1/2015	Quarter1	sweing	Thursday	8	0.8
1	1/1/2015	Quarter1	finishing	Thursday	1	0.7
2	1/1/2015	Quarter1	sweing	Thursday	11	0.8
3	1/1/2015	Quarter1	sweing	Thursday	12	0.8
4	1/1/2015	Quarter1	sweing	Thursday	6	0.8

Activity 3 : Correlation Analysis

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.

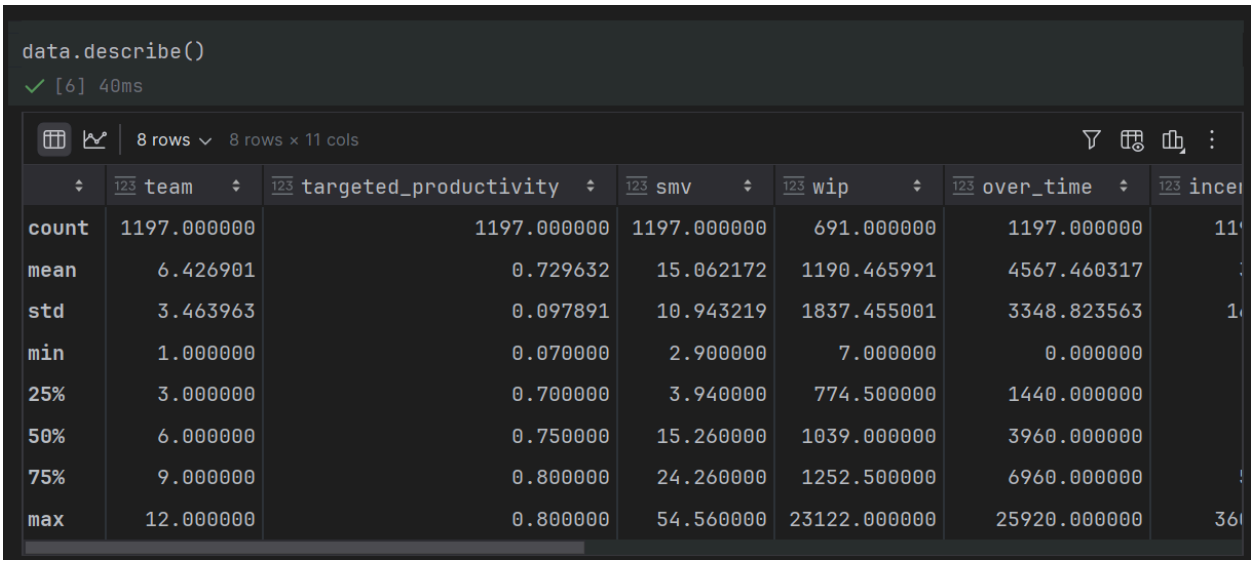
```
import seaborn as sns
import matplotlib.pyplot as plt

corrMatrix = data_ev.corr()
fig, ax = plt.subplots(figsize=(15,15)) # Sample figsize in inches
sns.heatmap(corrMatrix, annot=True, linewidths =0.5, ax=ax)
plt.show()
✓ [19] 698ms
```

Output :



Activity 4 : Descriptive analysis

Descriptive analysis is a statistical method used to summarize the core characteristics of a dataset. For this purpose, the pandas library in Python offers the powerful .describe() function. This function generates key statistics tailored to the data type. For numerical columns, it calculates measures of central tendency and dispersion, including the count, mean, standard deviation, minimum/maximum, and percentiles. For categorical columns, it provides the count of unique values, the most frequent value (top), and its corresponding frequency.



```
data.describe()
✓ [6] 40ms
```

8 rows × 11 cols

	team	targeted_productivity	smv	wip	over_time	incer
count	1197.000000	1197.000000	1197.000000	691.000000	1197.000000	11
mean	6.426901	0.729632	15.062172	1190.465991	4567.460317	
std	3.463963	0.097891	10.943219	1837.455001	3348.823563	1
min	1.000000	0.070000	2.900000	7.000000	0.000000	
25%	3.000000	0.700000	3.940000	774.500000	1440.000000	
50%	6.000000	0.750000	15.260000	1039.000000	3960.000000	
75%	9.000000	0.800000	24.260000	1252.500000	6960.000000	
max	12.000000	0.800000	54.560000	23122.000000	25920.000000	36

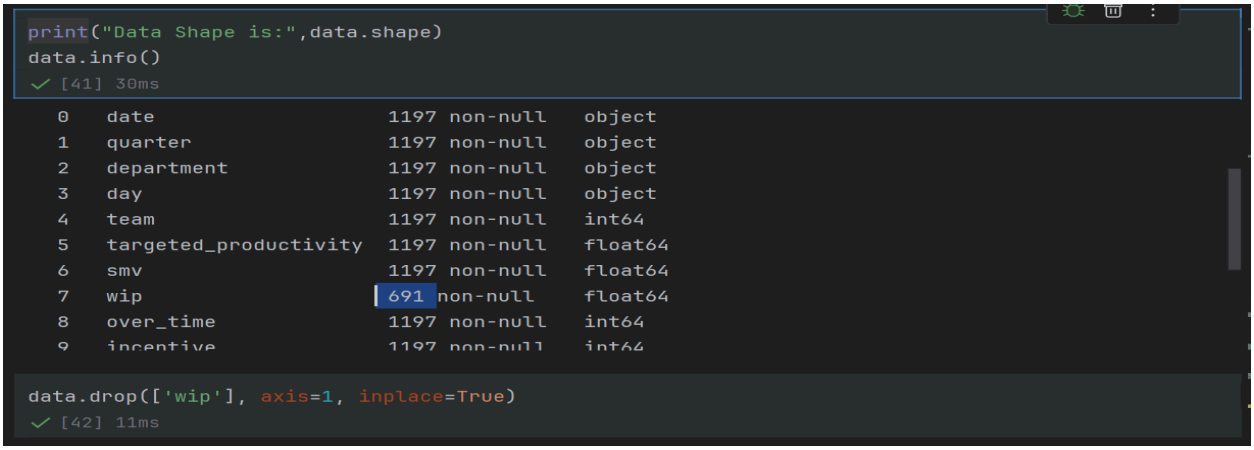
Milestone 3 : Data Pre-Processing

Activity 1 : Checking for Null Values

The initial data preprocessing stage focused on assessing data completeness. We quantified missing values across the dataset by executing the data.isnull().sum() command, which calculates the total null entries for each feature.

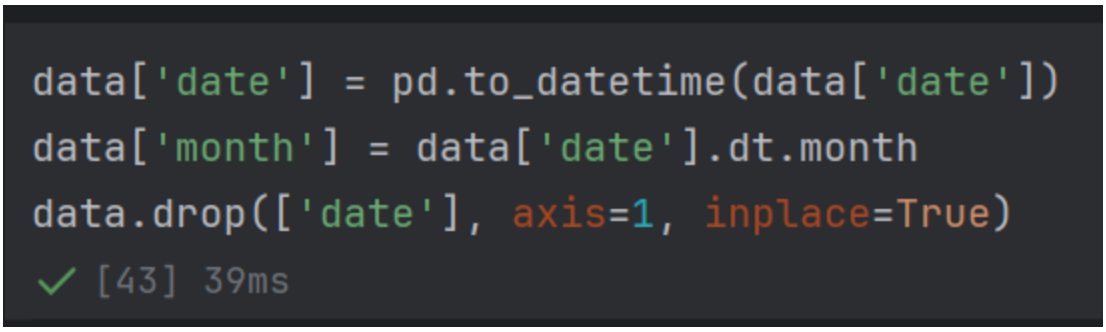
This analysis revealed that the **wip (Work in Progress)** feature suffered from a substantial amount of missing data. To ensure the integrity and reliability of subsequent modeling, the decision was made to remove this feature entirely. Following this modification, the final structure of the dataset was confirmed by inspecting its dimensions via the .shape attribute and reviewing the updated summary of data types

and non-null counts using the .info() method.

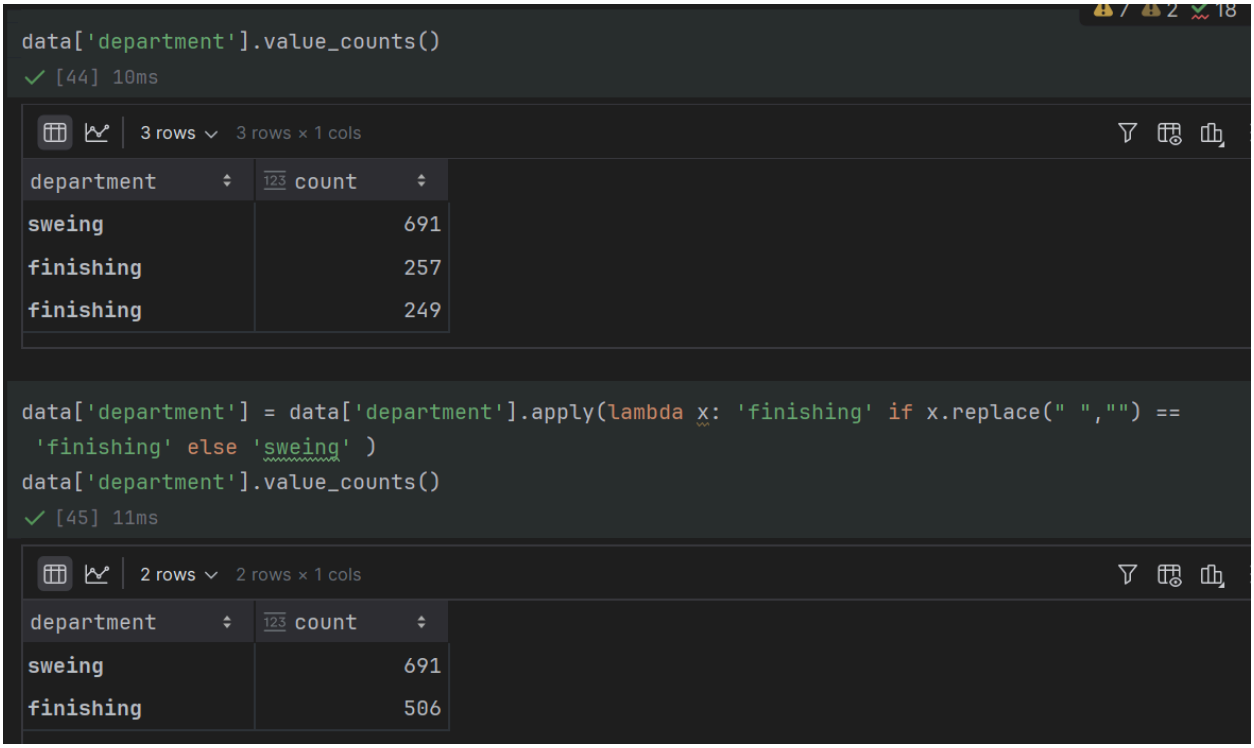


Activity 2 : Handling Date & Department Column

Here First we are converting the date column into datetime format.
Then converting date column to month (month index) & transferring the values into a new column called month. As we have the month column now we don't need date, so we will drop it.



Below image we can see that in department column the values are split into 3 categories Sweing, finishing, finishing. Finishing class is repeating twice, so we will merge them into 1.



Activity 3 : Handling Categorical Values

A key step in data preprocessing involves transforming categorical data into a numerical format, as most machine learning algorithms require numerical inputs. In this dataset, the features quarter, department, and day were identified as categorical. To handle these, we employed a label encoding technique, which assigns a unique integer to each category within a feature. This transformation was efficiently applied to all three columns using a MultiColumnLabelEncoder.

To prepare the dataset for analysis, it was necessary to convert its categorical features into a numerical representation. While several encoding techniques exist, we chose to apply label encoding to the **quarter**, **department**, and **day** columns. This process was implemented using the MultiColumnLabelEncoder, which systematically assigns a distinct integer value to each unique category present in these features, making them suitable for the modeling phase.

```
from multi_column_label_encoder import MultiColumnLabelEncoder
encoder = MultiColumnLabelEncoder(columns=['quarter','department','team','day',
 'no_of_style_change','month'])
data_ev = encoder.fit_transform(data)
data_ev.sample(5)
```

✓ [52] 34ms

5 rows × 14 cols

	quarter	department	day	team	targeted_productivity	smv
988	0	1	2	10	0.75	29.4
1104	0	1	1	8	0.50	23.4
53	0	0	2	1	0.70	3.9
965	3	0	1	7	0.80	4.6
1025	0	0	4	9	0.80	4.6

Activity 4 : Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set. After that x is converted into array format then passed into a new variable called X.

Here X and y variables are created. On X variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing X, y, test_size, random_state.

```
x=data_ev.drop(['actual_productivity'],axis=1)
y=data_ev['actual_productivity']
X=x.to_numpy()
```

✓ [35] 10ms

```
# Splitting the data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,train_size=0.8,random_state=0)
```

✓ [36] 17ms

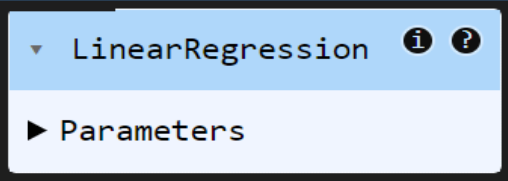
Milestone 4 : Model Building

Activity 1 : Linear Regression Model

Linear Regression has been initialized with the name model_lr. Then predictions are taken from x_test given to a variable named pred_test. After that Mean absolute error, mean squared error & r2_scrores are obtained.

```
from sklearn.linear_model import LinearRegression
model_lr=LinearRegression()
✓ [23] < 10 ms
```

```
model_lr.fit(x_train,y_train)
✓ [24] 94ms
```




```
pred_test = model_lr.predict(x_test)
print("test_MSE:",mean_squared_error(y_test, pred_test))
print("test_MAE:",mean_absolute_error(y_test, pred_test))
print("R2_score:{}".format(r2_score(y_test, pred_test)))
✓ [25] 15ms
```

```
test_MSE: 0.021507203121475953
test_MAE: 0.10727112243966751
R2_score:0.27256608178023445
```

Activity 2 : Random Forest Model

Random Forest has been initialized with the name model_rf. Then predictions are taken from x_test given to a variable named pred. After that Mean absolute error, mean squared error & r2_scrores are obtained.

```
from sklearn.ensemble import RandomForestRegressor
model_rf = RandomForestRegressor(n_estimators=200,max_depth=5)
✓ [26] < 10 ms

model_rf.fit(x_train,y_train)
✓ [27] 641ms

▾ RandomForestRegressor
▶ Parameters

pred = model_rf.predict(x_test)
print("test_MSE:",mean_squared_error(y_test, pred))
print("test_MAE:",mean_absolute_error(y_test, pred))
print("R2_score: {}". format(r2_score(y_test, pred)))
✓ [28] 27ms

test_MSE: 0.016158328115793305
test_MAE: 0.08691001458343507
R2_score: 0.4534800333282253
```

Activity 3 : Xgboost Model

XGBoost has been initialized with the name model_xgb. Then predictions are taken from x_test given to a variable named pred3. After that Mean absolute error, mean squared error & r2_scrores are obtained.

```
model_xgb = xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1)
model_xgb.fit(x_train,y_train)
✓ [29] 269ms

▾ XGBRegressor
▶ Parameters

pred3= model_xgb.predict(x_test)

print("test_MSE:",mean_squared_error(y_test, pred3))
print("test_MAE:",mean_absolute_error(y_test, pred3))
print("R2_score:{}".format(r2_score(y_test, pred3)))
✓ [30] 26ms

test_MSE: 0.013852956010708758
test_MAE: 0.07574575366973699
R2_score:0.5314541824486023
```

Activity 4 : Compare The Model

For comparing the above three models MSE, MAE & r2_scroes are used.

Linear Regression Model:

```
pred_test = model_lr.predict(x_test)
print("test_MSE:",mean_squared_error(y_test, pred_test))
print("test_MAE:",mean_absolute_error(y_test, pred_test))
print("R2_score:{}".format(r2_score(y_test, pred_test)))
✓ [25] 15ms
```

```
test_MSE: 0.021507203121475953
test_MAE: 0.10727112243966751
R2_score:0.27256608178023445
```

Random Forest Model:

```
pred = model_rf.predict(x_test)
print("test_MSE:",mean_squared_error(y_test, pred))
print("test_MAE:",mean_absolute_error(y_test, pred))
print("R2_score: {}". format(r2_score(y_test, pred)))
✓ [28] 27ms
```

```
test_MSE: 0.016158328115793305
test_MAE: 0.08691001458343507
R2_score: 0.4534800333282253
```

Xgboost Model:

```
pred3= model_xgb.predict(x_test)

print("test_MSE:",mean_squared_error(y_test, pred3))
print("test_MAE:",mean_absolute_error(y_test, pred3))
print("R2_score:{}".format(r2_score(y_test, pred3)))
✓ [30] 26ms
```

```
test_MSE: 0.013852956010708758
test_MAE: 0.07574575366973699
R2_score:0.5314541824486023
```

After calling the function, the results of models are displayed as output. From the three model **xgboost is performing well** because R2_score is 0.5314.

Activity 5 : Evaluating performance of the model and saving the model

From sklearn, metrics r2_score is used to evaluate the score of the model. On the parameters, we have given y_test & pred3. Our model is performing well. So, we are saving the model by pickle.dump().

Evaluating Model:

```
model_xgb = pickle.load(open('model.pkl', 'rb'))
Mcle = pickle.load(open('mcle.pkl', 'rb'))

data = pd.read_csv('../Dataset/garments_worker_productivity.csv')
data.drop(['wip'], axis=1, inplace=True)
data['date'] = pd.to_datetime(data['date'])
data['month'] = data['date'].dt.month
data.drop(['date'], axis=1, inplace=True)
data.drop(['actual_productivity'], axis=1, inplace=True)
data['department'] = data['department'].apply(lambda x: 'finishing' if x.replace(" ","") ==
 'finishing' else 'sweing' )

data[0:1]


✓ [38] 71ms
```

1 row 1 rows × 13 cols

	quarter	department	day	team	targeted_productivity	smv
0	Quarter1	sweing	Thursday	8	0.8	26.16

Saving the best model

```
pickle.dump(model_xgb, open('model.pkl','wb'))
pickle.dump(encoder, open('mcle.pkl','wb'))
✓ [31] 36ms
```

Milestone 5 : Application Building

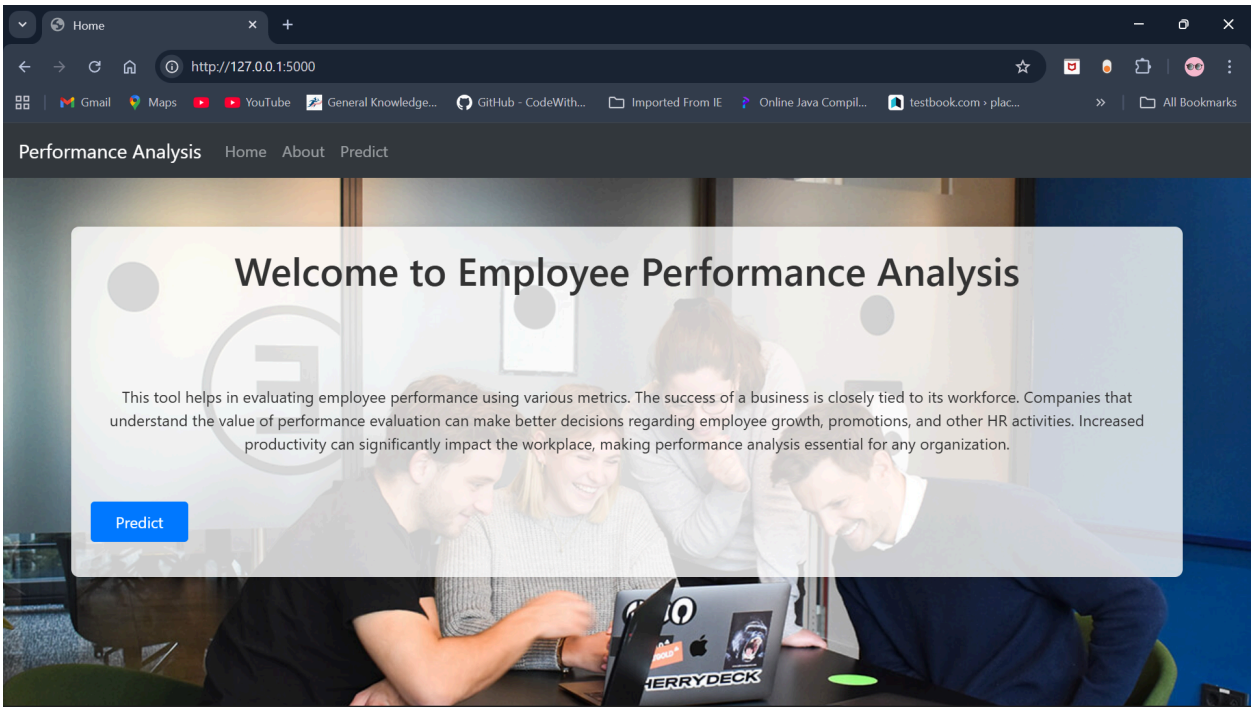
Activity 1: Building HTML pages

For this project create Four HTML files name

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

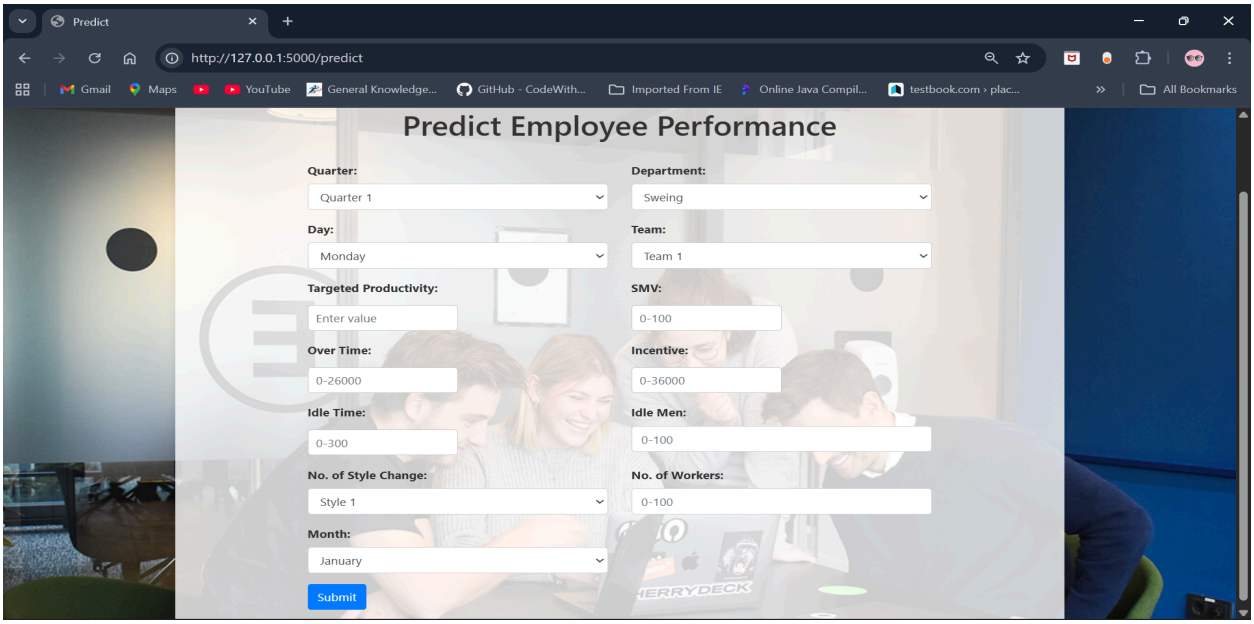
and save them in templates folder.

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



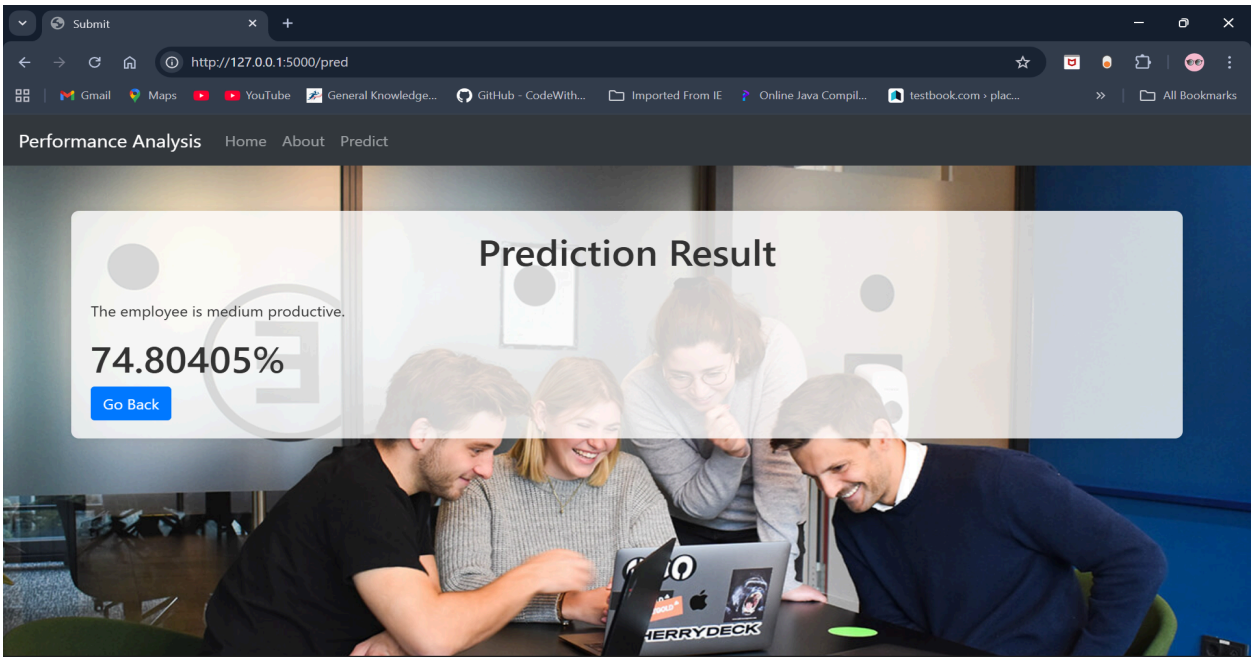
Now when you click on predict button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:

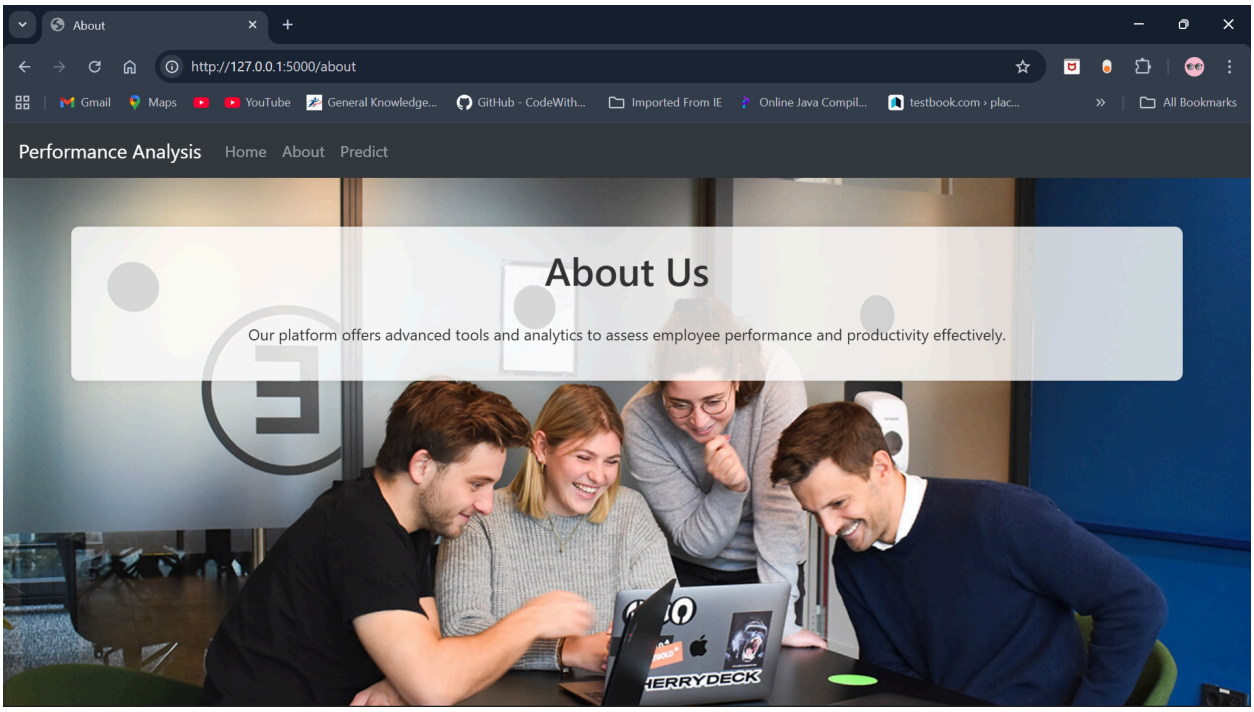


Now when you click on submit button from left bottom corner you will get redirected to submit.html

Lets look how our submit.html file looks like:



Lets look how our about.html file looks like:



Activity 2: Build Python code

Import the libraries

```
from flask import Flask, render_template, request
import pickle
import pandas as pd
```

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.

```
app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))
Mcle = pickle.load(open('mcle.pkl', 'rb'))
```

Render HTML page

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

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

/predict
@app.route("/predict")  2 usages
def home1():
    return render_template('predict.html')

/submit
@app.route("/submit")
def home2():
    return render_template('submit.html')

/pred
@app.route( rule: "/pred", methods=['POST'])  3 usages (2 dynamic)
def predict():
    quarter = request.form['quarter'] ## Querter 1, Quarter 2, Quarter 3, Quarter 4, Q
    department = request.form['department'] ## sweing , finishing
    day = request.form['day'] ## Wednesday,Sunday,Tuesday,Thursday,Monday,Saturday $Fr
    team = request.form['team'] ## 1,2,3,4,5,6,7,8,9,10,11,12
```

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 html page the values can be retrieved using POST Method.

Retrieves the value from UI

```python
@app.route("/pred", methods=['POST'])
def predict():
    quarter = request.form['quarter'] ## Querter 1, Quarter 2, Quarter 3, Quarter 4, Quarter 5
    department = request.form['department'] ## sweing , finishing
    day = request.form['day'] ## Wednesday,Sunday,Tuesday,Thursday,Monday,Saturday $Friday(Holiday)
    team = request.form['team'] ## 1,2,3,4,5,6,7,8,9,10,11,12
    targeted_productivity = request.form['targeted_productivity'] ## This is numerical Data
    smv = request.form['smv'] ## This is numerical Data | range 0-100
    over_time = request.form['over_time'] ## This is numerical Data  | range 0-26000
    incentive = request.form['incentive'] ## This is numerical Data  | range 0-36000
    idle_time = request.form['idle_time'] ## This is numerical Data  | range 0-300
    idle_men = request.form['idle_men'] ## This is numerical Data   | range 0-100
    no_of_style_change = request.form['no_of_style_change'] ## This is numerical Data | range 0-10
    no_of_workers = request.form['no_of_workers'] ## This is numerical Data | range 0-100
    month = request.form['month'] ## 1,2,3 (January, February, March) only dataset are available for these months

    # Prepare data for prediction
    data = {
        'quarter': [quarter.strip().lower()],
        'department': [department.strip().lower()],
        'day': [day.strip().lower()],
        'team': [team.strip()],
        'targeted_productivity': [float(targeted_productivity.strip())],
        'smv': [float(smv.strip())],
        'over_time': [int(over_time.strip())],
        'incentive': [int(incentive.strip())],
        'idle_time': [float(idle_time.strip())],
        'idle_men': [int(idle_men.strip())],
        'no_of_style_change': [int(no_of_style_change.strip())],
        'no_of_workers': [int(no_of_workers.strip())],
        'month': [int(month.strip())]
    }
    print(data)
    example_df = pd.DataFrame(data)
    # # Encode the data
    example_encoded = Mcle.transform(example_df)
    predicted_productivity_example = model.predict(example_encoded)
```

```python
# # Encode the data
example_encoded = Mcle.transform(example_df)
predicted_productivity_example = model.predict(example_encoded)
print(predicted_productivity_example)

# prediction = model.predict(total)
prediction = predicted_productivity_example[0]
print(prediction)
if prediction <= 0.3:
    text = 'The employee is averagely productive.'
elif 0.3 < prediction <= 0.8:
    text = 'The employee is medium productive'
else:
    text = 'The employee is highly productive'
return render_template('submit.html', prediction_text=text,score=prediction)
```

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 home.html page earlier.

Main Function

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

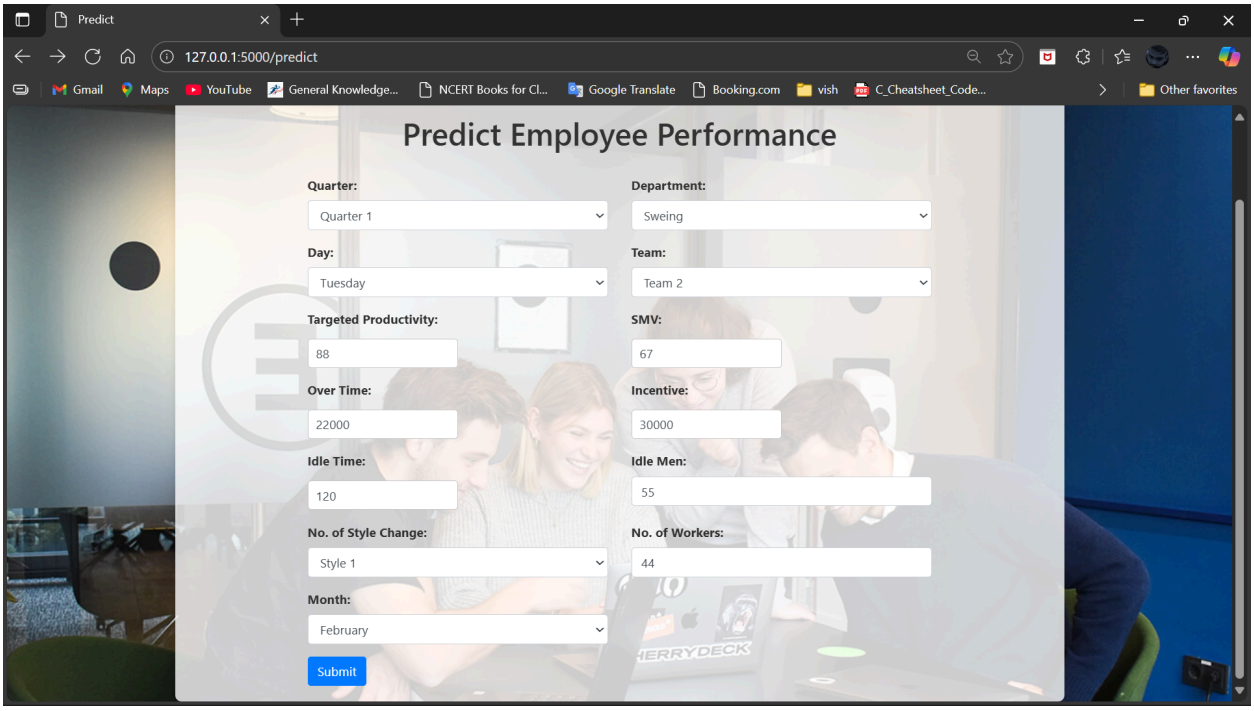
Activity 3: Run the application

To execute the application, launch the **Anaconda Prompt** and navigate to the project's root directory. Run the command python app.py to start the local web server. Access the application by opening a web browser and going to http://127.0.0.1:5000. To generate a prediction, navigate to the "**Predict**" page, enter the required input values into the form, and click "**Submit**" to view the resulting output from the model.

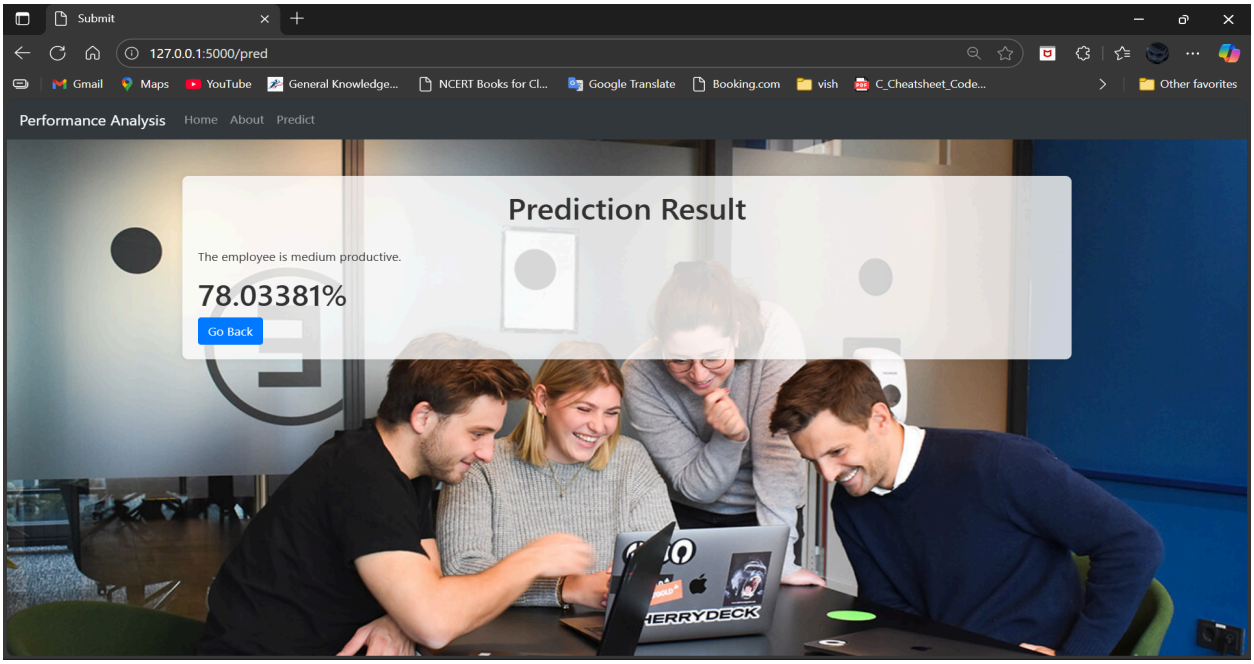
```
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Activity 4: Output

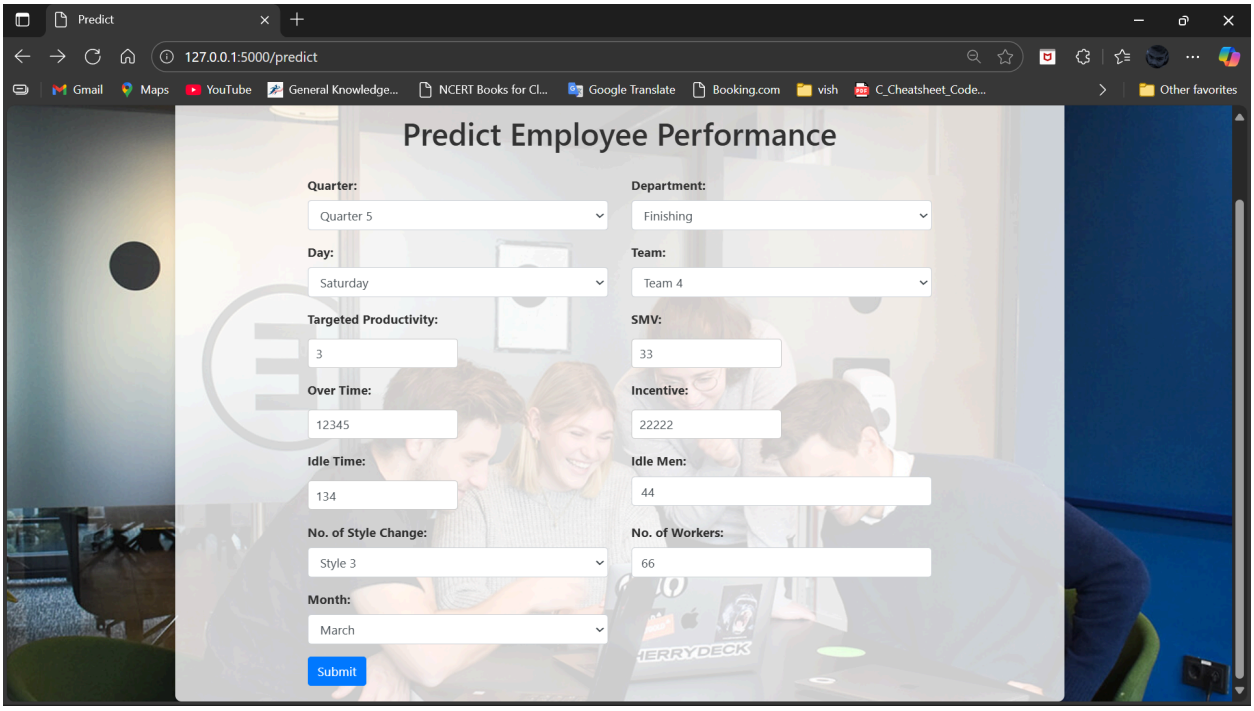
Input 1 :



Output 1:



Input 2 :



Output 2 :