EMPLOYEE PERFORMANCE PREDICTION

MACHINE LEARNING PROJECT REPORT

Employee Performance Prediction

Executive Summary

The Machine Learning Approach for Employee Performance Prediction with a comprehensive system designed to analyze various data points related to employees' work performance and use machine learning algorithms, leveraging ML technology stack, to predict and evaluate their future performance. By incorporating factors such as past performance metrics, training data, feedback, and external factors, the system aims to provide insights that can aid in talent management, resource allocation, and workforce optimization strategies.

Project Scenarios

Scenario 1: Talent Retention

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.

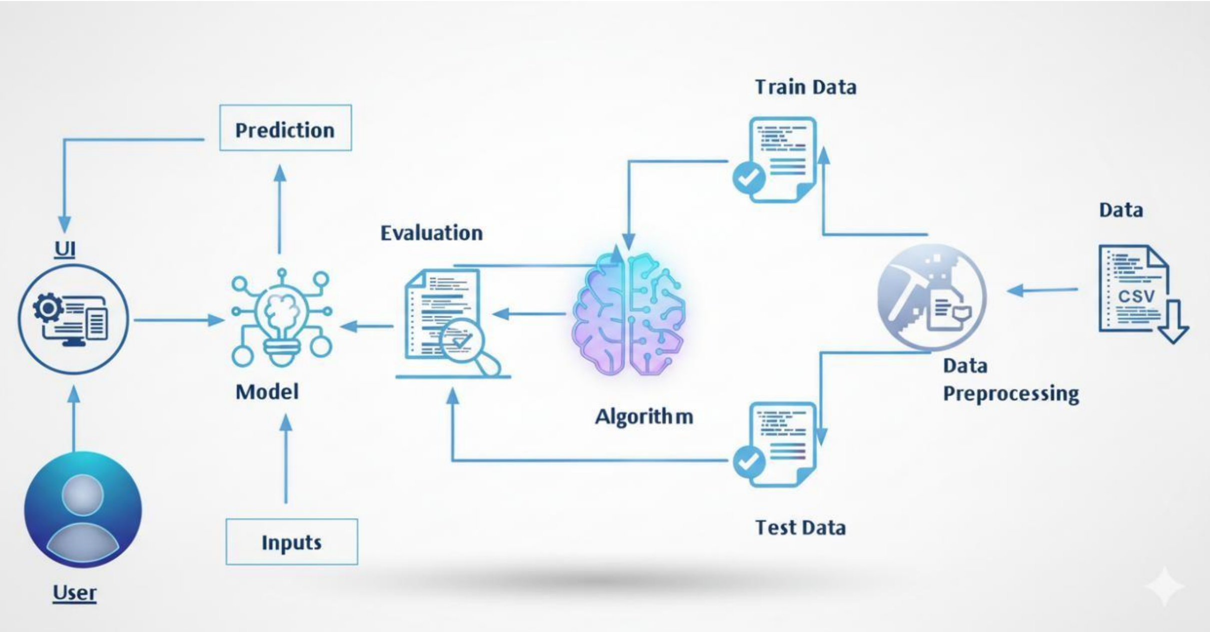
Scenario 2: Performance Improvement

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:

1.Data collection

- Collect the dataset or create the dataset

2.Visualizing and analyzing data

- Correlation analysis
- Descriptive analysis

3.Data pre-processing

- Checking for null values
- Handling Date & department column
- Handling categorical data
- Splitting data into train and test

4.Model building

- Import the model building libraries
- Initializing the model
- Training and testing the model
- Evaluating performance of model
- Save the model

5.Application Building

- Create an HTML file
- Build python code

Pre-requisites

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
  - Linear Regression
  - Decision tree
  - Random forest
  - Evaluation metrics
- **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 garments_worker_productivity.csv dataset from kaggle

Activity 1: Dataset Collection

- The dataset was obtained from Kaggle.com
- Link:"Productivity Prediction of Garment Employees"
- The dataset used in this project is garments_worker_productivity.csv.

Features:

Feature	Type	Description
date	datetime	Date of production
department	categorical	Sewing or Finishing
quarter	categorical	Quarter of the year (Q1–Q5)
day	categorical	Day of the week
team	numeric	Team number
targeted_productivity	numeric	Targeted productivity value
smv	numeric	Standard Minute Value
over_time	numeric	Extra hours worked
incentive	numeric	Incentive given to workers
idle_time	numeric	Total idle time in hours
idle_men	numeric	Number of idle workers
no_of_style_change	numeric	Number of style changes in the line
no_of_workers	numeric	Total workers in the team
actual_productivity	numeric	Target variable (actual productivity)

Dataset Overview:

- Total records: 2000+
- Missing values: 0–5% per feature (handled in preprocessing)
- Target: actual_productivity

Milestone 2:Visualizing and analyzing the data

Activity 1.Importing the libraries

Importing the necessary libraries

```
[1]: # Data manipulation and visualization
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Preprocessing & model selection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Machine Learning Models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

# Evaluation metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Saving the model
import pickle
```

Activity 2.Read the dataset : By using read_csv( )

Read the dataset

```
[2]: data = pd.read_csv("garments_worker_productivity.csv")
print(data.head())
```

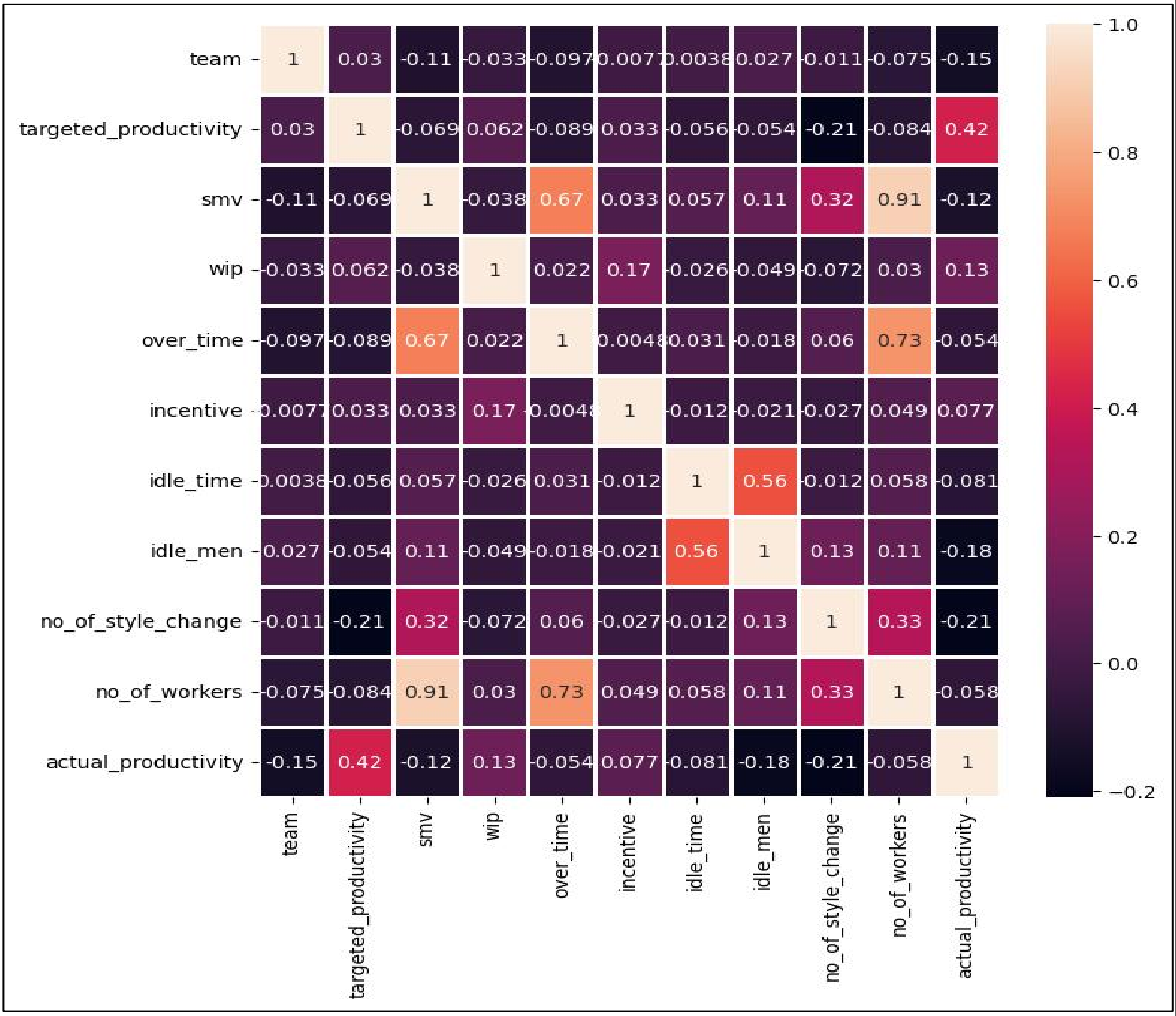
```
       date   quarter  department       day  team  targeted_productivity  \
0  1/1/2015  Quarter1      sweing  Thursday     8                   0.80   
1  1/1/2015  Quarter1  finishing   Thursday     1                   0.75   
2  1/1/2015  Quarter1      sweing  Thursday    11                   0.80   
3  1/1/2015  Quarter1      sweing  Thursday    12                   0.80   
4  1/1/2015  Quarter1      sweing  Thursday     6                   0.80   

     smv     wip  over_time  incentive  idle_time  idle_men  \
0  26.16  1108.0       7080         98        0.0         0   
1   3.94     NaN        960          0        0.0         0   
2  11.41   968.0       3660         50        0.0         0   
3  11.41   968.0       3660         50        0.0         0   
4  25.90  1170.0       1920         50        0.0         0   

   no_of_style_change  no_of_workers  actual_productivity  
0                   0           59.0             0.940725  
1                   0            8.0             0.886500  
2                   0           30.5             0.800570  
3                   0           30.5             0.800570  
4                   0           56.0             0.800382  
```

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

Correlation Analysis

```
[3]: corrMatrix = data.select_dtypes(include=['number']).corr()
     fig, ax = plt.subplots(figsize=(8,8))
     sns.heatmap(corrMatrix, annot=True, linewidths=1, ax=ax)
     plt.show()
```



Activity 4.Descriptive Analysis :

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

Descriptive Analysis

```
[4]: data.describe()
```

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
count	1197.000000	1197.000000	1197.000000	691.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000
mean	6.426901	0.729632	15.062172	1190.465991	4567.460317	38.210526	0.730159	0.369256	0.150376	34.609858	0.735091
std	3.463963	0.097891	10.943219	1837.455001	3348.823563	160.182643	12.709757	3.268987	0.427848	22.197687	0.174488
min	1.000000	0.070000	2.900000	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	0.233705
25%	3.000000	0.700000	3.940000	774.500000	1440.000000	0.000000	0.000000	0.000000	0.000000	9.000000	0.650307
50%	6.000000	0.750000	15.260000	1039.000000	3960.000000	0.000000	0.000000	0.000000	0.000000	34.000000	0.773333
75%	9.000000	0.800000	24.260000	1252.500000	6960.000000	50.000000	0.000000	0.000000	0.000000	57.000000	0.850253
max	12.000000	0.800000	54.560000	23122.000000	25920.000000	3600.000000	300.000000	45.000000	2.000000	89.000000	1.120437

Milestone 3: Data Preprocessing

Activity 1:Checking for null values

For checking the null values, data.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that in our dataset there is one feature which has high number of null values. So we drop that feature.

```
[7]: print(data.isnull().sum())

     date                       0
     quarter                    0
     department                 0
     day                        0
     team                       0
     targeted_productivity      0
     smv                        0
     wip                      506
     over_time                  0
     incentive                  0
     idle_time                  0
     idle_men                   0
     no_of_style_change         0
     no_of_workers              0
     actual_productivity        0
     dtype: int64

[8]: # Drop 'wip' column if exists
     if 'wip' in data.columns:
         data.drop(['wip'], axis=1, inplace=True)
```

- To find the shape of our data, data.shape method is used.
- To find the data type, data.info( ) function is used.

```
[5]: data.shape

[5]: (1197, 15)

[6]: data.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 1197 entries, 0 to 1196
     Data columns (total 15 columns):
      #   Column                 Non-Null Count  Dtype
     ---  ------                 --------------  -----
      0   date                   1197 non-null   object
      1   quarter                1197 non-null   object
      2   department             1197 non-null   object
      3   day                    1197 non-null   object
      4   team                   1197 non-null   int64
      5   targeted_productivity  1197 non-null   float64
      6   smv                    1197 non-null   float64
      7   wip                    691 non-null    float64
      8   over_time              1197 non-null   int64
      9   incentive              1197 non-null   int64
      10  idle_time              1197 non-null   float64
      11  idle_men               1197 non-null   int64
      12  no_of_style_change     1197 non-null   int64
      13  no_of_workers          1197 non-null   float64
      14  actual_productivity    1197 non-null   float64
     dtypes: float64(6), int64(5), object(4)
     memory usage: 140.4+ KB
```

Activity 2:Handling Date and Department Column

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

Handling Date and Department Column

```
[9]: if 'date' in data.columns:
         data['date'] = pd.to_datetime(data['date'])
         data['month'] = data['date'].dt.month
         data.drop(['date'], axis=1, inplace=True)
```

```
10]: if 'department' in data.columns:
         data['department'] = data['department'].apply(lambda x: 'finishing' if 'finishing' in x.lower() else 'sewing')
```

```
11]: categorical_cols = ['department', 'quarter', 'day'] if 'quarter' in data.columns else ['department', 'day']
```

Activity 3: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 **MultiColoumnLabelEncoder.**
- In our project, categorical features are quarter, department, day. With MultiColoumnLabelEncoder encoding is done.

Handling Categorical Values

```
[13]: from sklearn.base import BaseEstimator, TransformerMixin

      class MultiColumnLabelEncoder(BaseEstimator, TransformerMixin):
          def __init__(self, columns=None):
              self.columns = columns

          def fit(self, X, y=None):
              return self

          def transform(self, X):
              output = X.copy()
              if self.columns is not None:
                  for col in self.columns:
                      le = LabelEncoder()
                      output[col] = le.fit_transform(output[col])
              else:
                  for colname, col in output.items():
                      if output[colname].dtype == object:
                          le = LabelEncoder()
                          output[colname] = le.fit_transform(col)
              return output

      mcle = MultiColumnLabelEncoder(columns=categorical_cols)
      X_encoded = mcle.fit_transform(X_df)
```

```
[14]: scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X_encoded)
```

Activity 4:Splitting data into train and test set:

- 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.

Spilting Data into Train and Test

```
[15]: x_train, x_test, y_train, y_test = train_test_split(X_scaled, y, train_size=0.8, random_state=0)
```

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.

1.Linear Regression Model

```
[16]: model_lr = LinearRegression()
model_lr.fit(x_train, y_train)
pred_lr = model_lr.predict(x_test)

print("Linear Regression Performance:")
print("MAE:", mean_absolute_error(y_test, pred_lr))
print("MSE:", mean_squared_error(y_test, pred_lr))
print("R2:", r2_score(y_test, pred_lr))
```

```
Linear Regression Performance:
MAE: 0.10636001215550112
MSE: 0.020952954761876034
R2: 0.2913123154775418
```

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.

2.Random Forest Model

```
[17]: model_rf = RandomForestRegressor(n_estimators=200, max_depth=5, random_state=0)
model_rf.fit(x_train, y_train)
pred_rf = model_rf.predict(x_test)

print("\nRandom Forest Performance:")
print("MAE:", mean_absolute_error(y_test, pred_rf))
print("MSE:", mean_squared_error(y_test, pred_rf))
print("R2:", r2_score(y_test, pred_rf))
```

```
Random Forest Performance:
MAE: 0.08580681722601405
MSE: 0.015568129692169114
R2: 0.47344219899891493
```

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.

3.Xgboost Model

```
[18]: model_xgb = xgb.XGBRegressor(
          n_estimators=200,
          max_depth=5,
          learning_rate=0.1,
          random_state=0
      )
      model_xgb.fit(x_train, y_train)
      pred_xgb = model_xgb.predict(x_test)

      print("XGBoost Performance:")
      print("MAE:", mean_absolute_error(y_test, pred_xgb))
      print("MSE:", mean_squared_error(y_test, pred_xgb))
      print("R2:", r2_score(y_test, pred_xgb))

      XGBoost Performance:
      MAE: 0.07904341491293163
      MSE: 0.015048792808164598
      R2: 0.4910076286958249
```

Activity 4:Compare the model

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

Compare the Model and Evaluating the Performance of the Model

```
[19]: # Create a dictionary to store results
      results = {
          'Model': ['Linear Regression', 'Random Forest', 'XGBoost'],
          'MAE': [
              mean_absolute_error(y_test, pred_lr),
              mean_absolute_error(y_test, pred_rf),
              mean_absolute_error(y_test, pred_xgb)
          ],
          'MSE': [
              mean_squared_error(y_test, pred_lr),
              mean_squared_error(y_test, pred_rf),
              mean_squared_error(y_test, pred_xgb)
          ],
          'R2 Score': [
              r2_score(y_test, pred_lr),
              r2_score(y_test, pred_rf),
              r2_score(y_test, pred_xgb)
          ]
      }

      # Convert to DataFrame for easy comparison
      results_df = pd.DataFrame(results)
      print("Model Performance Comparison:")
      print(results_df.sort_values(by='R2 Score', ascending=False))

      Model Performance Comparison:
                     Model       MAE       MSE  R2 Score
      2              XGBoost  0.079043  0.015049  0.491008
      1        Random Forest  0.085807  0.015568  0.473442
      0    Linear Regression  0.106360  0.020953  0.291312
```

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().

Saving the best model

[20]:
```
with open("gwp.pkl", "wb") as f:
    pickle.dump(model_xgb, f)
```

Milestone 5: Application Building

In this section, we had build 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.

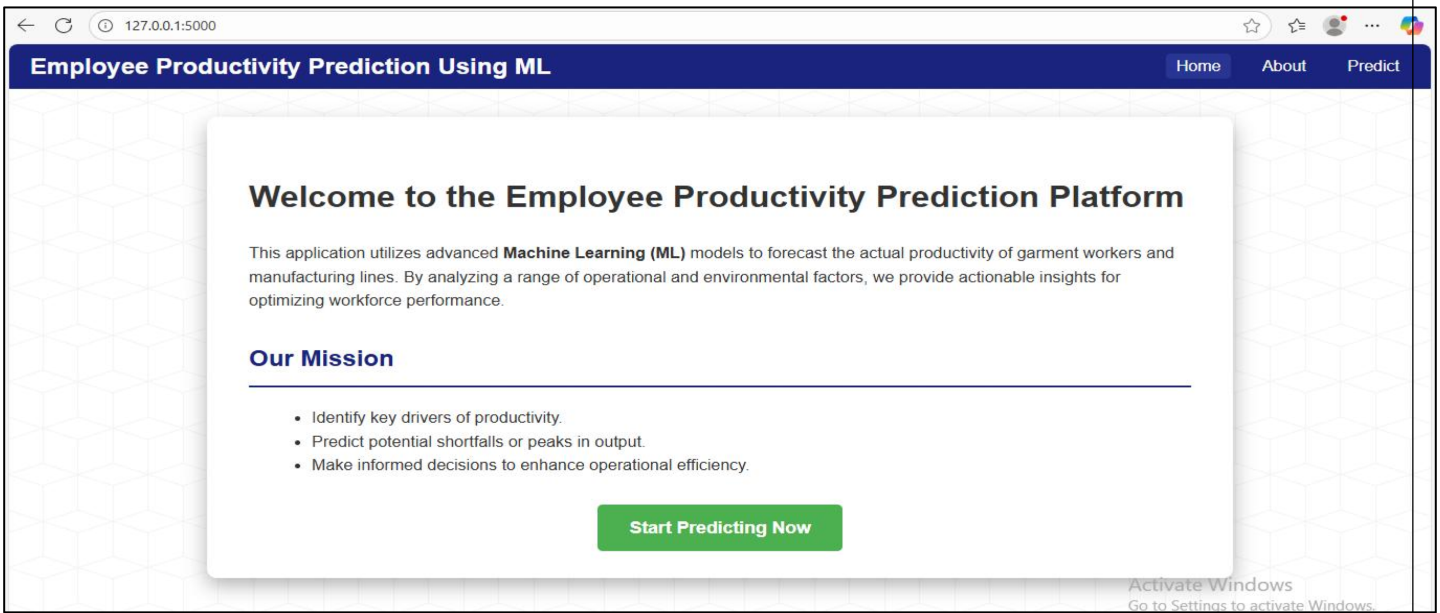
Activity 1:Building Html Pages

For this project we had created 4 HTML files namely

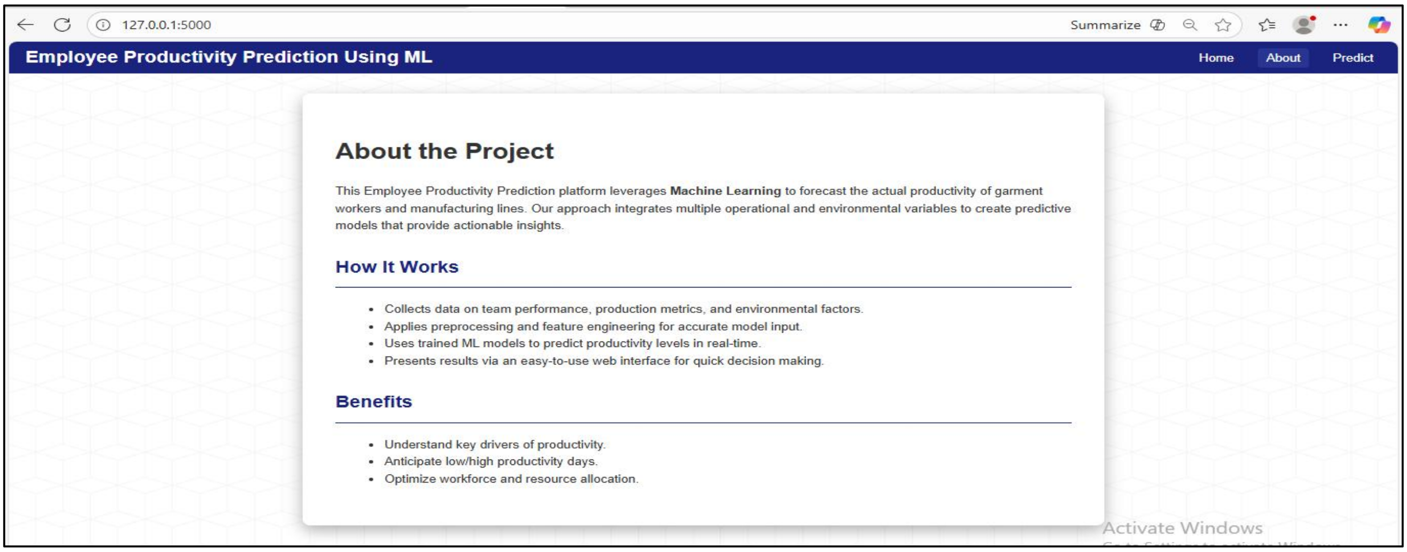
- about.html
- home.html
- predict.html
- submit.html

and saved them in templates folder.

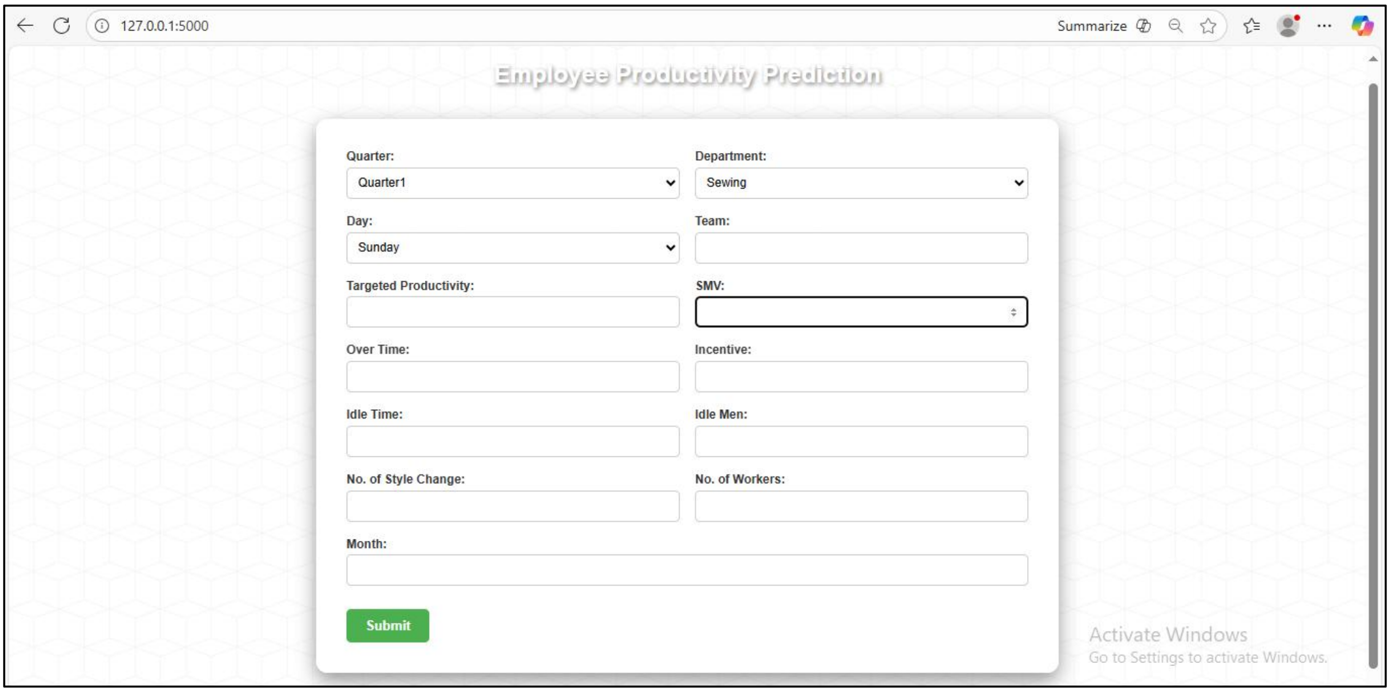
1. home.html

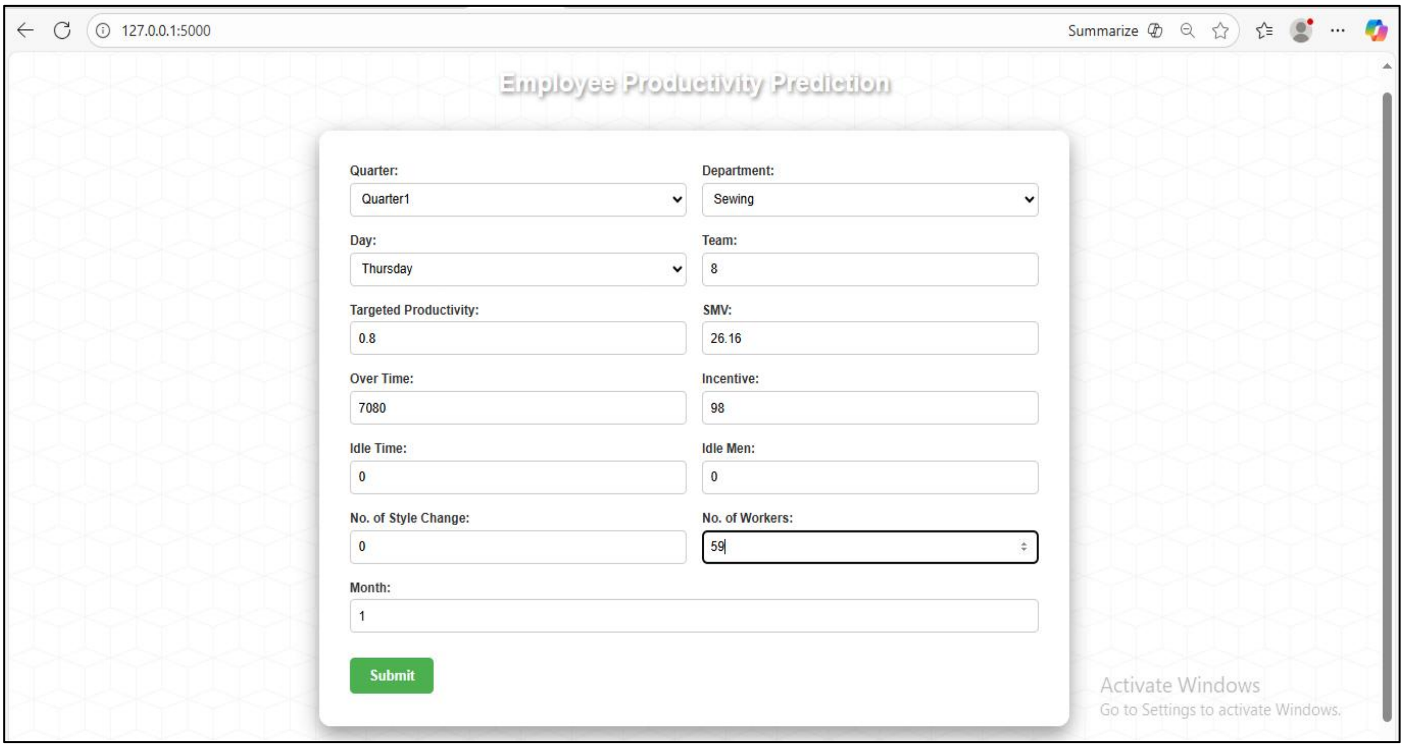


2. about.html

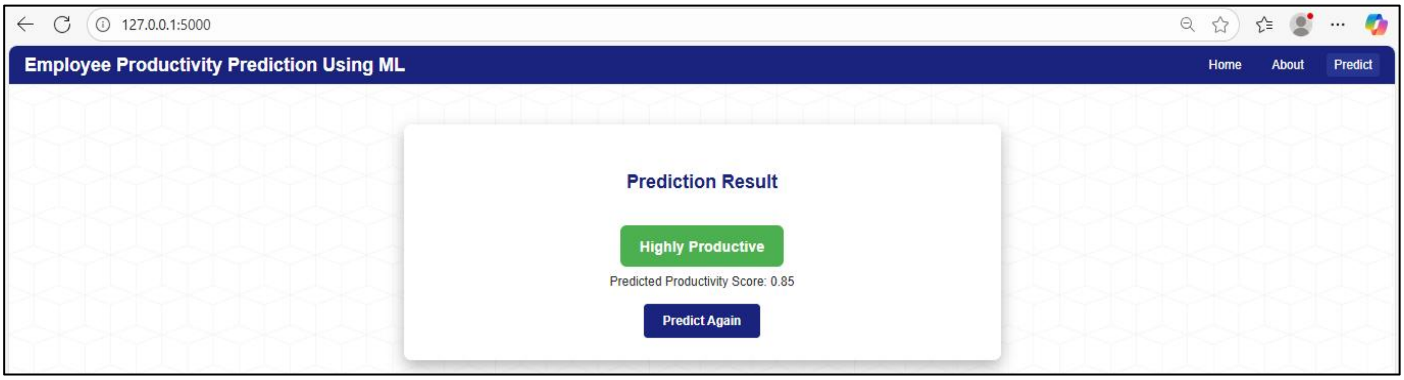


3. predict.html





4. submit.html



Activity 2:Build Python code

- Import the libraries
- 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
from flask import Flask, render_template, request
import pickle

app = Flask(__name__)

# Load the trained model
model = pickle.load(open('gwp.pkl', 'rb'))

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

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

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

@app.route("/submit", methods=['POST'])
def submit_prediction():
    # Map categorical values if needed (here assume numeric values are sent from form)
    quarter = int(request.form['quarter'])
    department = int(request.form['department'])
    day = int(request.form['day'])
    team = int(request.form['team'])
    targeted_productivity = float(request.form['targeted_productivity'])
    smv = float(request.form['smv'])
    over_time = int(request.form['over_time'])
    incentive = int(request.form['incentive'])
    idle_time = float(request.form['idle_time'])
    idle_men = int(request.form['idle_men'])
    no_of_style_change = int(request.form['no_of_style_change'])
    no_of_workers = float(request.form['no_of_workers'])
    month = int(request.form['month'])

    total = [[
        quarter, department, day, team, targeted_productivity,
        smv, over_time, incentive, idle_time, idle_men,
        no_of_style_change, no_of_workers, month
    ]]

    prediction = model.predict(total)[0]  # Get scalar value

    # Convert prediction score to text
    if prediction < 0.3:
        text = 'Low Productive'
    elif prediction < 0.8:
        text = 'Medium Productive'
    else:
        text = 'Highly Productive'

    # Pass both text and numeric score to template
    return render_template('submit.html', prediction_text=text, prediction_score=round(prediction, 2))

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