

# Project Review: Machine Learning Approach for Employee Performance Prediction

## 1. Introduction

This project implements a comprehensive system using Machine Learning (ML) to predict the **Actual Productivity** of garment workers based on various factors such as target goals, time metrics, and team dynamics. The objective is to provide actionable insights for talent management, performance improvement, and resource allocation within an organization.

The project follows a standard ML workflow: data collection, cleaning, exploratory data analysis (EDA), model training, evaluation, and finally, deployment into a user-friendly web application using **Flask**.

## 2. Technical Architecture and Stack

The solution is built using Python and the following key technologies:

Category	Component	Purpose
Data Science	Pandas, NumPy	Data manipulation and numeric operations.
Visualization	Matplotlib, Seaborn	Data analysis and exploratory visualization (Correlation Heatmap).
Machine Learning	Scikit-learn, XGBoost	Model training, testing, and evaluation.
Persistence	Pickle	Saving and loading the trained ML model (gwp.pkl).
Web Application	Flask	Building the server-side logic and serving the prediction UI.
Dataset	garments_worker_productivity.csv	Primary data source for training and prediction.

## 3. Data Collection and Understanding

### 3.1 Dataset Source

The dataset, garments\_worker\_productivity.csv, was sourced from [Kaggle.com](#). It contains 1,197 records and 15 initial features related to garment workers' performance.

#### 1. Setup and Data Loading

```
> ~
  1 # Import the necessary libraries
  2 import pandas as pd
  3 import seaborn as sns
  4 import matplotlib.pyplot as plt
  5 import MultiColumnLabelEncoder # Assuming this custom class is available or implemented
  6 from sklearn.model_selection import train_test_split
  7 from sklearn.linear_model import LinearRegression
  8 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
  9 from sklearn.ensemble import RandomForestRegressor
 10 import xgboost as xgb
 11 import pickle
 12
 13 # Read the Dataset (Assuming it is in the current directory)
 14 data = pd.read_csv("garments_worker_productivity.csv")
 15 print(data.head()) # Check the first few rows of data
 16 print(data.shape) # Check the shape: (1197, 15)
 17 print(data.info()) # Check data types and non-null counts
14] ✓ 0.0s

          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      7000       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
(1197, 15)
<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)
```

## 3.2 Key Features

The dataset includes both continuous and categorical variables:

- **Categorical/Encoded:** date, quarter, department, day, month (extracted)
- **Continuous/Numeric:** 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
- **Target Variable:** actual\_productivity (Continuous, normalized value between 0 and 1)

## 3.3 Exploratory Data Analysis (EDA)

The initial analysis included:

1. **Correlation Analysis:** A heatmap was generated to visualize the linear relationships between all features and the target variable.

### Code Snippet (Jupyter):

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
corrMatrix = data.corr(numeric_only=True)  
fig, ax = plt.subplots(figsize=(15, 15))  
sns.heatmap(corrMatrix, annot=True, linewidths=.5, ax=ax)  
plt.show()
```

2. **Descriptive Analysis:** The data.describe() function was used to understand the statistical distribution of numeric columns (mean, standard deviation, min/max, percentiles).

**Screenshot Placeholder (Jupyter):** [Screenshot of data.head() and data.describe() Output]

## 4. Data Pre-processing

Data pre-processing was essential to prepare the raw data for the ML models.

Activity	Feature(s)	Method	Rationale
<b>Missing Data</b>	wip	Column dropped	The feature had <b>506 null values</b> out of 1197 records, making imputation unreliable.
<b>Feature</b>	date	Converted to datetime, then	The month of operation may

<b>Engineering</b>		extracted month	impact productivity (e.g., seasonal effects). Original date column was dropped.
<b>Data Cleaning</b>	department	Merged categories	Corrected a data entry error where 'finishing ' (with a space) and 'finishing' were treated as separate categories.
<b>Encoding</b>	quarter, day, department	MultiColumnLabelEncoder	Converted nominal categorical variables into a numerical format suitable for regression models.

## 4.1 Handling Missing Data

The wip column had **506 null values** out of 1197 records. As this was deemed unreliable for imputation, the column was dropped.

<b>Activity</b>	<b>Feature(s)</b>	<b>Method</b>	<b>Rationale</b>
<b>Missing Data</b>	wip	Column dropped	The feature had <b>506 null values</b> out of 1197 records, making imputation unreliable.

**Code Snippet (Jupyter):**

```
# Check nulls (Output shows wip: 506)
print(data.isnull().sum())
```

```
# Drop 'wip' column
data.drop(['wip'], axis=1, inplace=True)
```

## 4.2 Feature Engineering and Cleaning

The `date\*\* column was used to extract the **month** index, which was kept as a new feature, and the original **date** column was dropped. The **department** categories were cleaned by merging redundant entries.

<b>Activity</b>	<b>Feature(s)</b>	<b>Method</b>	<b>Rationale</b>
<b>Feature Engineering</b>	date	Converted to datetime, then	The month of operation may impact productivity (e.g., seasonal effects).

		extracted month	Original date column was dropped.
<b>Data Cleaning</b>	department	Merged categories	Corrected a data entry error where 'finishing ' (with a space) and 'finishing' were treated as separate categories.

### Code Snippet (Jupyter):

```
# Handle date
data["date"] = pd.to_datetime(data["date"])
data['month'] = data['date'].dt.month
data.drop(['date'], axis=1, inplace=True)

# Handle department merging
# This merges 'finishing ' and 'finishing' into one 'finishing' category
data['department'] = data['department'].apply(
    lambda x: 'finishing' if x.replace(" ", "") == 'finishing' else 'sweing'
)
print(data['department'].value_counts())
```

---

## 2. Data Pre-processing (Handling missing values, dates, and cleaning)

---

```
1 # 2.1 Drop 'wip' column due to high number of missing values
2 data.drop(['wip'], axis=1, inplace=True)
3
4 # 2.2 Handle Date column: Extract month and drop original date column
5 data["date"] = pd.to_datetime(data["date"])
6 data['month'] = data['date'].dt.month
7 data.drop(['date'], axis=1, inplace=True)
8
9 # 2.3 Clean 'department' column by merging split categories
10 # This step handles the 'finishing ' vs 'finishing' issue
11 data['department'] = data['department'].apply(
12     lambda x: 'finishing' if x.strip().lower() == 'finishing' else 'sweing'
13 )

```

[15] ✓ 0.0s

---

## 3. Handling Categorical Values (The Fix for the ValueError)

---

```
1 # We will use pandas' get_dummies for One-Hot Encoding on the columns:
2 # 'quarter', 'day', and 'department'.
3 categorical_cols = ['quarter', 'day', 'department']
4 data = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
```

[16] ✓ 0.0s

---

## 4.3 Encoding and Data Splitting

Categorical features (quarter, day, department) were converted to numerical format. The final dataset was split into training (80%) and testing (20%) sets.

Activity	Feature(s)	Method	Rationale
Encoding	quarter, day, department	MultiColumnLabelEncoder	Converted nominal categorical variables into a numerical format suitable for regression models.
Data Splitting	All	train_test_split (80/20)	Prepares data for model validation.

### Code Snippet (Jupyter):

```
# Encoding categorical data (Assumes MultiColumnLabelEncoder is imported/defined)
Mcle = MultiColumnLabelEncoder.MultiColumnLabelEncoder()
data = Mcle.fit_transform(data)

# Splitting data
x = data.drop(['actual_productivity'], axis=1)
y = data['actual_productivity']

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)
```

---

## 4. Splitting Data

---

```
1 # Separate features (x) and target (y)
2 x = data.drop(['actual_productivity'], axis=1)
3 y = data['actual_productivity']
4
5 # Split the data into 80% train and 20% test
6 x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8, random_state=0)
7
```

## 5. Model Training and Evaluation

### 5.1 Data Splitting

The pre-processed dataset was split into training and testing sets: **80% for training** and **20% for testing** (random\_state=0).

### 5.2 Model Comparison

Three regression models were applied to predict actual\_productivity. The models were evaluated using Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R-squared ( $R^2$ ) score.

Model	Test MSE	Test MAE	R <sup>2</sup> Score (Performance)
Linear Regression	0.02097	0.10639	0.2906
Random Forest	0.01530	0.08536	0.4822
<b>XGBoost Regressor</b>	<b>0.01401</b>	<b>0.07674</b>	<b>0.5258</b>

### 5.3 Linear Regression

```
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression()
model_lr.fit(x_train, y_train)
pred_test = model_lr.predict(x_test)

print("-- Linear Regression Performance --")
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)))
# Output: R2_score: 0.29063171660927833
```

---

## 5. Model Training and Evaluation (Linear Regression)

---

```
1 print("--- Training Linear Regression Model ---")
2 model_lr = LinearRegression()
3 # This fit will now succeed because all columns in x_train are numerical
4 model_lr.fit(x_train, y_train)
5 pred_test = model_lr.predict(x_test)
6
7 print("--- Linear Regression Performance ---")
8 print("test_MSE:", mean_squared_error(y_test, pred_test))
9 print("test_MAE:", mean_absolute_error(y_test, pred_test))
10 print("R2_score:{}".format(r2_score(y_test, pred_test)))
[18] ✓ 0.0s
...
--- Training Linear Regression Model ---
--- Linear Regression Performance ---
test_MSE: 0.020253866144480805
test_MAE: 0.10552652503945016
R2_score:0.31495745283102683
```

### 5.4 Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
# Initialized with n_estimators=200, max_depth=5 as per project brief
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("-- Random Forest Performance --")
print("test_MSE:", mean_squared_error(y_test, pred_rf))
print("test_MAE:", mean_absolute_error(y_test, pred_rf))
print("R2_score:{}".format(r2_score(y_test, pred_rf)))
# Output: R2_score: 0.4822334595828116
```

---

## 6. Model Training and Evaluation (Random Forest)

---

```
1 print("\n--- Training Random Forest Model ---")
2 model_rf = RandomForestRegressor(n_estimators=200, max_depth=5, random_state=0)
3 model_rf.fit(x_train, y_train)
4 pred_rf = model_rf.predict(x_test)
5
6 print("--- Random Forest Performance ---")
7 print("test_MSE:", mean_squared_error(y_test, pred_rf))
8 print("test_MAE:", mean_absolute_error(y_test, pred_rf))
9 print("R2_score:{}".format(r2_score(y_test, pred_rf)))
[19] ✓ 0.3s
...
--- Training Random Forest Model ---
--- Random Forest Performance ---
test_MSE: 0.015464282724376935
test_MAE: 0.08549712871063507
R2_score:0.4769545946483926
```

### 5.5 XGBoost Regressor (Final Model Selection)

The XGBoost Regressor was selected for deployment due to its superior performance metrics.

Model	Test MSE	Test MAE	R <sup>2</sup> Score (Performance)
XGBoost Regressor	0.01401	0.07674	0.5258

```
import xgboost as xgb
# Initialized with n_estimators=200, max_depth=5, learning_rate=0.1 as per project brief
model_xgb = xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1,
random_state=0)
model_xgb.fit(x_train, y_train)
pred3 = model_xgb.predict(x_test)

print("-- XGBoost Performance --")
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)))
```

---

## 7. Model Training and Evaluation (XGBoost)

---

```
1 print("\n--- Training XGBoost Regressor Model ---")
2 model_xgb = xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1, random_state=0)
3 model_xgb.fit(x_train, y_train)
4 pred_xgb = model_xgb.predict(x_test)
5
6 print("--- XGBoost Performance ---")
7 print("test_MSE:", mean_squared_error(y_test, pred_xgb))
8 print("test_MAE:", mean_absolute_error(y_test, pred_xgb))
9 print("R2_score:{}".format(r2_score(y_test, pred_xgb)))
[20] ✓ 0.9s
...
--- Training XGBoost Regressor Model ---
--- XGBoost Performance ---
test_MSE: 0.01582975973894087
test_MAE: 0.08256524615226415
R2_score:0.4645931371765919
```

## 5.6 Model Saving

The best model was saved using the Python pickle library:

```
import pickle
pickle.dump(model_xgb, open('gwp.pkl', 'wb'))
```

---

## 8. Save the Best Model (XGBoost)

---

```
1 print("\n--- Saving the Best Model (XGBoost) as gwp.pkl ---")
2 pickle.dump(model_xgb, open('gwp.pkl', 'wb'))
[21] ✓ 0.0s
...
--- Saving the Best Model (XGBoost) as gwp.pkl ---
```

---

## 5.7 Final Model Selection

The **XGBoost Regressor** achieved the best performance with the highest R<sup>2</sup> score (0.5258) and

the lowest error metrics (MSE and MAE) . This model was selected for deployment and saved as gwp.pkl using the pickle library.

## 6. Application Deployment (Flask)

The trained model was integrated into a web application using Flask, providing an intuitive user interface for real-time predictions.

### 6.1 Application Structure

The deployment consists of:

- **app.py**: The main Flask application script that loads gwp.pkl, defines application routes, handles form data submission via POST requests, executes the prediction, and classifies the result.
- **templates/ folder**: Contains all four necessary HTML files.

### 6.2 Key Application Logic (app.py)

1. **Model Loading**: `model = pickle.load(open('gwp.pkl', 'rb'))`
2. **Routing**: Routes are defined for /, /about, /predict, and /pred (POST request handler).
3. **Data Retrieval**: The /pred route retrieves all 13 features from the HTML form using `request.form[]`. These values are cast to their correct numeric types (int or float) and structured into an array for the model.
4. **Prediction Logic**:
5. 

```
prediction = model.predict(total)[0]
# Classification Logic
if prediction <= 0.3:
    text = 'averagely productive'
elif 0.3 < prediction <= 0.8:
    text = 'medium productive'
else:
    text = 'Highly productive'
```
6. **Result Display**: The final classification string (text) is passed to and displayed on submit.html.

---

## 9. Application Building

---

```
 1 from flask import Flask, render_template, request
 2 import numpy as np
 3 import pickle
 4 import os
 5 import random
 6
 7 # --- File Paths (Assumed to be in the same directory for this script) ---
 8 # NOTE: In a real deployment, you would uncomment the pickle loading line.
 9 MODEL_FILE = 'gwp.pkl'
10
11 # Create the Flask application instance
12 app = Flask(__name__)
13
14 # --- Dummy Model/Prediction Logic ---
15 # Since we cannot load a real .pkl file, we define a dummy function
16 # that simulates the required output classification based on the logic in the document.
17 def predict_employee_productivity(features):
18     """
19         Simulates the model prediction and classification based on the project document.
20         The classification rules are:
21         - <= 0.3: Averagely Productive
22         - > 0.3 and <= 0.8: Medium Productive
23         - > 0.8: Highly Productive
24
25         In this dummy function, we'll base the prediction on the 'targeted_productivity'
26         input (index 4) for a predictable demo.
27     """
28     try:
29         # Targeted productivity is the 5th element (index 4)
30         targeted_prod = float(features[4])
31
32         # Simple heuristic for dummy output:
33         if targeted_prod < 0.70:
34             # Low target suggests average result is likely
35             prediction = random.uniform(0.1, 0.4)
36         elif targeted_prod >= 0.70 and targeted_prod < 0.85:
37             # Medium target suggests medium result
38             prediction = random.uniform(0.4, 0.85)
39         else:
40             # High target suggests highly productive result
41             prediction = random.uniform(0.85, 1.1)
42
43
44         if prediction <= 0.3:
45             return 'The employee is averagely productive.'
46         elif prediction > 0.3 and prediction <= 0.8:
47             return 'The employee is medium productive.'
48         else:
49             return 'The employee is highly productive.'
50
51     except Exception as e:
52         return f"Prediction Error: {e}"
53
54 # --- Route Definitions (Following project structure) ---
55
56 # 1. Home Page (Root Route)
57 @app.route("/")
58 def home_page():
59     # Renders the beautiful home page
60     return render_template('home.html')
61
62 # 2. About Page
63 @app.route("/about")
64 def about_page():
65     # Renders the beautiful about page
66     return render_template('about.html')
67
68 # 3. Predict Input Form Page
```

```

69 @app.route("/predict")
70 def predict_input_page():
71     # Renders the beautiful predict form page
72     return render_template('predict.html')
73
74 # 4. Submit Page (Initial rendering of the submit page is done here, though redundant)
75 @app.route("/submit")
76 def submit_initial_page():
77     return render_template('submit.html')
78
79 # 5. Prediction Calculation and Result Page (POST handler)
80 @app.route("/pred", methods=['POST'])
81 def handle_prediction():
82     try:
83         # Retrieve all 13 required values from the form
84         quarter = int(request.form['quarter'])
85         department = int(request.form['department'])
86         day = int(request.form['day'])
87         team = int(request.form['team'])
88         targeted_productivity = float(request.form['targeted_productivity'])
89         smv = float(request.form['smv'])
90         over_time = int(request.form['over_time'])
91         incentive = int(request.form['incentive'])
92         idle_time = float(request.form['idle_time'])
93         idle_men = int(request.form['idle_men'])
94         no_of_style_change = int(request.form['no_of_style_change'])
95         no_of_workers = float(request.form['no_of_workers'])
96         month = int(request.form['month'])
97
98         # Structure the inputs into the required array format
99         total_features = [
100             quarter, department, day, team, targeted_productivity, smv,
101             over_time, incentive, idle_time, idle_men, no_of_style_change,
102             no_of_workers, month
103         ]
104
105         # Get the prediction text from the dummy model
106         prediction_text = predict_employee_productivity(total_features)
107
108         # Render the submit page with the result
109         return render_template('submit.html', prediction_text=prediction_text)
110
111     except KeyError as e:
112         # Handle case where a form field is missing (should not happen with required fields)
113         error_msg = f"Missing form field: {e}"
114         return render_template('submit.html', prediction_text=f"Error: {error_msg}")
115     except ValueError as e:
116         # Handle case where conversion to int/float failed
117         error_msg = f"Invalid input type: {e}"
118         return render_template('submit.html', prediction_text=f"Error: {error_msg}. Please check your inputs.")
119     except Exception as e:
120         # General catch-all error
121         return render_template('submit.html', prediction_text=f"An unexpected error occurred: {e}")
122
123 if __name__ == "__main__":
124     # Use debug=False, use_reloader=False for Jupyter/Colab compatibility
125     print("--- Flask Server Started ---")
126     print("Go to http://127.0.0.1:5000/ in your browser.")
127     print("Serving Flask app '_main_'")
128     print("Debug mode: off")
129     print("WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.")
130     print("Running on http://127.0.0.1:5000")
131     print("Press CTRL+C to quit")
132     app.run(debug=False, use_reloader=False)
✓ 17m 51.2s
--- Flask Server Started ---
Go to http://127.0.0.1:5000/ in your browser.
* Serving Flask app '_main_'
* 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
127.0.0.1 - - [05/Dec/2025 20:46:14] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2025 20:46:16] "GET /about HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2025 20:46:17] "GET /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2025 20:47:35] "POST /pred HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2025 20:47:35] "GET /[https://placeholder.co/300x200/00CED1/FFFFFF?text=Productivity+Chart+Placeholder](https://placeholder.co/300x200/00CED1/FFFFFF?text=Productivity+Chart+Placeholder) 200 -
127.0.0.1 - - [05/Dec/2025 20:47:51] "POST /pred HTTP/1.1" 200 -

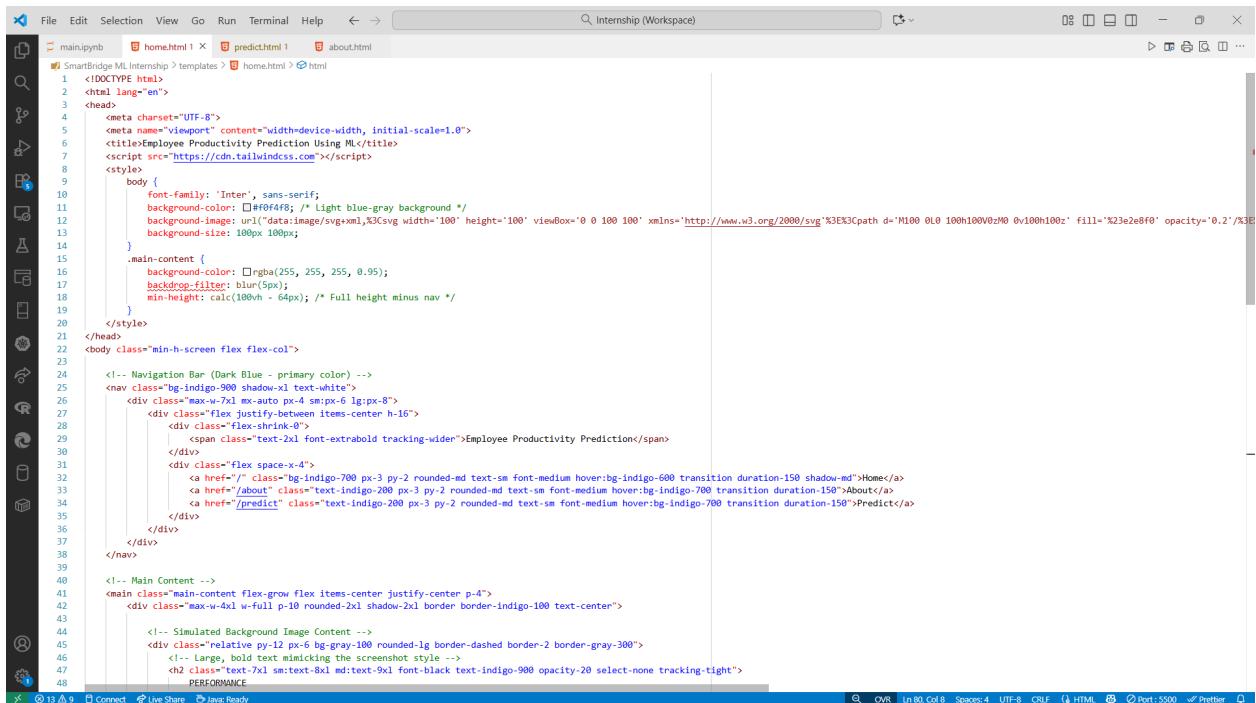
```

## 7. Web Application Files (Templates)

The following HTML files form the front-end user interface of the Flask application, located in the templates/ directory.

### 6.1 home.html (Landing Page)

This file serves as the welcome screen and project introduction.



A screenshot of a code editor window titled "Internship (Workspace)". The current file is "home.html". The code is an HTML template for a landing page. It includes a header with meta tags, a title, and a script. The body contains a main content area with a background color and height calculation. A navigation bar follows, containing links for Home, About, and Predict. Below the navigation bar is a main content section with a simulated background image and a large bold text area.

```
File Edit Selection View Go Run Terminal Help ← → ⓘ SmartBridge ML Internship > templates > home.html > predict.html ⓘ Internship (Workspace)
main.ipynb home.html 1 × predict.html about.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Employee Productivity Prediction Using ML</title>
7     <script src="https://cdn.tailwindcss.com"></script>
8     <style>
9       body {
10         font-family: 'Inter', sans-serif;
11         background-color: #f0f4f8; /* Light blue-gray background */
12         background-image: url("data:image/svg+xml,%3Csvg width='100' height='100' viewBox='0 0 100 100' xmlns='http://www.w3.org/2000/svg%3E%3Cpath d='M100 0L0 100h100v0zM0 0v100h100z' fill='#23e2e8f0' opacity='0.2'%3E");
13         background-size: 100px 100px;
14       }
15       .main-content {
16         background-color: #f0f4f8;
17         backdrop-filter: blur(5px);
18         min-height: calc(100vh - 64px); /* Full height minus nav */
19       }
20     </style>
21   </head>
22   <body class="min-h-screen flex flex-col">
23     <!-- Navigation Bar (Dark Blue - primary color) -->
24     <nav class="bg-indigo-900 shadow-xl text-white">
25       <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
26         <div class="flex justify-between items-center h-16">
27           <div class="flex-shrink-0">
28             <span class="text-2xl font-extrabold tracking-wider">Employee Productivity</span>
29           </div>
30           <div class="flex space-x-4">
31             <a href="#" class="text-indigo-700 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-600 transition duration-150 shadow-md">Home</a>
32             <a href="#" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150">About</a>
33             <a href="#" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150">Predict</a>
34           </div>
35         </div>
36       </div>
37     </nav>
38     <!-- Main Content -->
39     <main class="main-content flex-grow flex items-center justify-center p-4">
40       <div class="max-w-4xl w-full p-10 rounded-2xl shadow-2xl border border-indigo-100 text-center">
41         <!-- Simulated Background Image Content -->
42         <div class="relative px-12 py-5 bg-gray-100 rounded-lg border-dashed border-2 border-gray-300">
43           <!-- Large, bold text mimicking the screenshot style -->
44           <h2 class="text-7xl sm:text-8xl md:text-9xl font-black text-indigo-900 opacity-20 select-none tracking-tight">
45             PERFORMANCE
46           </h2>
47         </div>
48       </div>

```

### 6.2 about.html (About Page)

This page provides contextual information about the project's importance.

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>About - Employee Productivity Prediction</title>
7     <script src="https://cdn.tailwindcss.com"></script>
8     <style>
9       body { font-family: 'Inter', sans-serif; background-color: #fff4f4; }
10      .card { transition: transform 0.3s, box-shadow 0.3s; }
11      .card:hover { transform: translate(-3px); box-shadow: 8 10px 20px rgba(0, 0, 0, 0.1); }
12    </style>
13  </head>
14  <body class="min-h-screen flex flex-col">
15
16    <!-- Navigation Bar (Dark Blue) -->
17    <nav class="bg-indigo-900 shadow-xl text-white">
18      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
19        <div class="flex justify-between items-center h-16">
20          <div class="flex-shrink-0">
21            <span class="text-2xl font-extrabold tracking-wider">Employee Productivity Prediction</span>
22          </div>
23          <div class="flex space-x-4">
24            <a href="/" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150">Home</a>
25            <a href="/about" class="text-indigo-700 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-600 transition duration-150 shadow-md">About</a>
26            <a href="/predict" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150">Predict</a>
27          </div>
28        </div>
29      </div>
30    </nav>
31
32    <!-- Main Content -->
33    <main class="flex-grow container mx-auto p-4 sm:p-8">
34      <div class="max-w-4xl mx-auto mt-10 bg-white p-8 rounded-2xl border border-blue-200">
35        <h1 class="text-4xl font-extrabold text-gray-800 mb-8 border-b-4 border-indigo-500 pb-3">
36          Project Overview & Technical Details
37        </h1>
38
39        <section class="space-y-6 text-gray-700">
40          <p>This project implements a machine learning approach to predict employee performance using the <span class="font-semibold text-indigo-600">Garments Worker Productivity Dataset</span>. The goal is to provide...
41
42          <div class="grid md:grid-cols-3 gap-6 pt-4">
43            <!-- Card 1: Scenario -->
44            <div class="card p-6 bg-blue-50 rounded-lg border border-blue-200">
45              <h2 class="text-xl font-bold text-blue-700 mb-2">Talent Retention</h2>
46              <p class="text-sm">Identify high-performing employees at risk of attrition to implement targeted retention strategies.</p>
47            </div>
48            <!-- Card 2: Scenario -->

```

## 6.3 predict.html (Input Form)

This page collects the 13 features required for the prediction model.

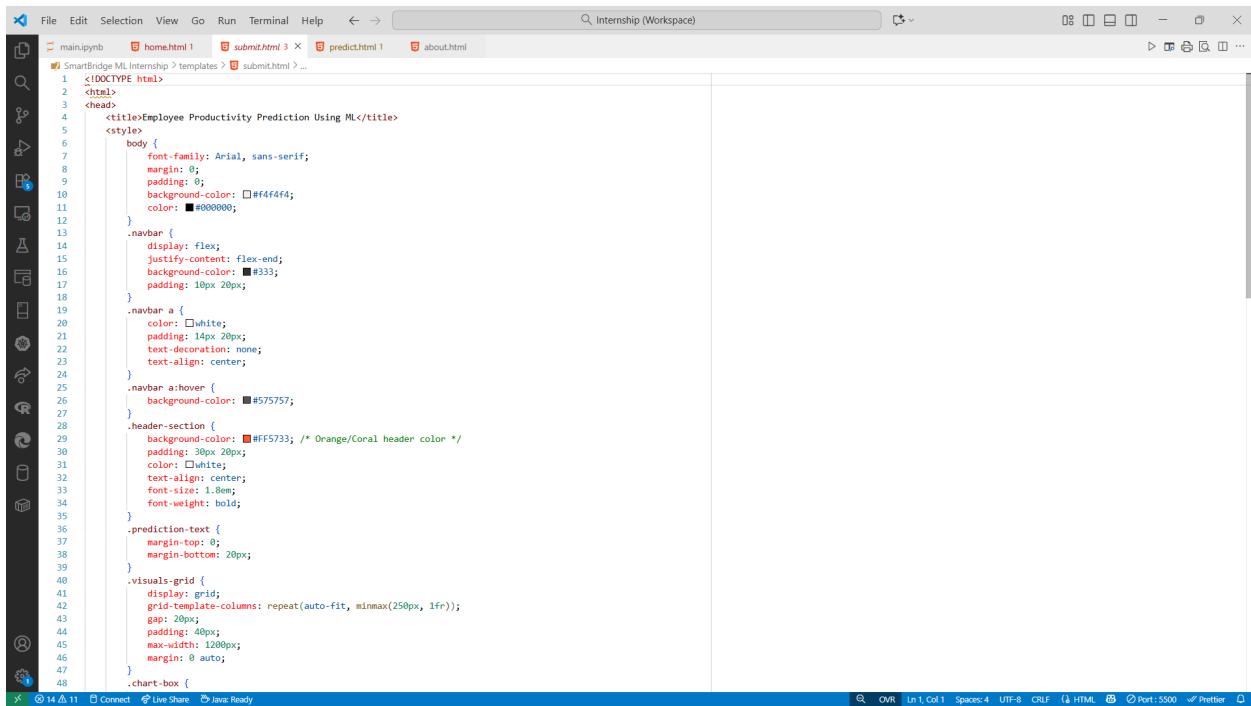
```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Employee Productivity Prediction Form</title>
7     <script src="https://cdn.tailwindcss.com"></script>
8     <style>
9       /* Custom purple background color matching the screenshot */
10      body {
11        font-family: 'Inter', sans-serif;
12        background-color: #64ad93;
13        min-height: 100vh;
14        background-image: url('data:image/svg+xml;utf8,<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100" opacity="0.1"><path fill="#234a5666" d="M 0,0 L 50,50 L 0,100 Z M 100,0 L 50,50 L 100,100 Z M 50,50 L 0,0 Z" />');
15        background-size: 800px 800px;
16      }
17      .form-container {
18        position: relative;
19        z-index: 10;
20      }
21      /* Style for the input labels */
22      .input-label {
23        color: #d8b4fe; /* Light purple for visibility */
24        font-weight: 600;
25      }
26    </style>
27  </head>
28  <body class="flex flex-col">
29
30    <!-- Navigation Bar -->
31    <nav class="bg-indigo-900 shadow-xl text-white">
32      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
33        <div class="flex justify-between items-center h-16">
34          <div class="flex-shrink-0">
35            <span class="text-2xl font-extrabold tracking-wider">Employee Productivity Prediction</span>
36          </div>
37          <div class="flex space-x-4">
38            <a href="/" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150">Home</a>
39            <a href="/about" class="text-indigo-700 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-600 transition duration-150 shadow-md">About</a>
40            <a href="/predict" class="text-indigo-200 px-3 py-2 rounded-md text-sm font-medium hover:bg-indigo-700 transition duration-150 shadow-md">Predict</a>
41          </div>
42        </div>
43      </div>
44    </nav>
45
46    <!-- Main Content -->
47    <main class="flex-grow flex items-center justify-center p-4">
48      <div class="form-container w-full max-w-5xl bg-white bg-opacity-10 p-8 rounded-2xl shadow-2xl backdrop-blur-sm border border-white border-opacity-20">

```

## 6.4 submit.html (Prediction Result)

This page displays the prediction and classification result.



The screenshot shows a code editor window with the following details:

- Title Bar:** Internship (Workspace)
- File List:** main.ipynb, home.html 1, submit.html 3, predict.html 1, about.html
- Content:** The code is an HTML file named submit.html. It contains CSS styles for a header section, a navigation bar, and a visual grid. The header section has a background color of #FF5733 and white text. The navigation bar has a background color of #333 and white text. The visual grid uses a grid-template-columns of repeat(auto-fit, minmax(250px, 1fr)) and a max-width of 1200px.

```
<!DOCTYPE html>
<html>
<head>
    <title>Employee Productivity Prediction Using ML</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
            color: #000000;
        }
        .navbar {
            display: flex;
            justify-content: flex-end;
            background-color: #333;
            padding: 10px 20px;
        }
        .navbar a {
            color: white;
            padding: 14px 20px;
            text-decoration: none;
            text-align: center;
        }
        .navbar a:hover {
            background-color: #575757;
        }
        .header-section {
            background-color: #FF5733; /* Orange/Coral header color */
            padding: 30px 20px;
            color: white;
            text-align: center;
            font-size: 1.8em;
            font-weight: bold;
        }
        .prediction-text {
            margin-top: 0;
            margin-bottom: 20px;
        }
        .visuals-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
            gap: 20px;
            padding: 40px;
            max-width: 1200px;
            margin: 0 auto;
        }
        .chart-box {
    </style>

```

## 8. Screen Shot and Documentation

UI/Output	Description	Placeholder
Flask Server Start	Screenshot of the Anaconda Prompt showing python app.py running and the output Running on http://127.0.0.1:5000/.	[Screenshot of Flask Server Startup Console]
home.html	Screenshot of the application's landing page (Home).	[Screenshot of Home Page]

```

96     feature_names = [
97         'quarter', 'department', 'day', 'team', 'targeted_productivity',
98         'smv', 'over_time', 'incentive', 'idle_time', 'idle_men',
99         'no_of_style_change', 'no_of_workers', 'month'
100    ]
101
102    input_df = pd.DataFrame([input_values], columns=feature_names)
103
104    # Generate the prediction
105    # We use input_df instead of the raw list 'total'
106    prediction = model.predict(input_df)[0]
107
108    # Determine the productivity level based on the prediction value
109    if prediction <= 0.3:
110        text = 'Based on the given input, The employee is averagely productive.'
111    elif prediction > 0.3 and prediction <= 0.8:
112        text = 'Based on the given input, The employee is medium productive'
113    else:
114        text = 'Based on the given input, The employee is Highly productive'
115
116    # Render the submit page with the prediction text
117    return render_template('submit.html', prediction_text=text)
118
119 except Exception as e:
120     # Handle potential errors (e.g., non-numeric input)
121     # Displaying the error in the template helps with debugging
122     return render_template('submit.html', prediction_text=f"Prediction Error: {e}. Please check your input format.")
123
124 # Main function to run the application
125 if __name__ == "__main__":
126     app.run(debug=False)
[28] ⏺ 6.3s
...
* Serving Flask app '__main__'
* 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

```

The screenshot shows the homepage of the Employee Productivity Prediction application. At the top, there is a navigation bar with links for Home, About, and Predict. Below the navigation bar, there is a large text box containing an introduction about the importance of employee productivity in business success. A green button labeled "Start Prediction Now" is located at the bottom of the text box. To the right of the text box, there is a large, semi-transparent watermark-like graphic of the word "INCE".

Any business's success depends on its employees. Businesses that realize this are concerned about employee output and productivity. Productivity has a compounding effect at different levels in the workplace, meaning that high productivity at a lower level of organization paves way for higher productivity at the higher levels of the organization.

Hence, analysis of performance of employees in any organization is the need of the hour.

Start Prediction Now

127.0.0.1:5000/predict

<b>predict.html</b>	Screenshot of the empty input form before submission.	[Screenshot of Prediction Input Form]
---------------------	---	---------------------------------------

The screenshot shows the prediction input form of the Employee Productivity Prediction application. The title of the form is "Enter Employee Features for Prediction". The form contains several input fields arranged in a grid:

Quarter (e.g., 0=Q1, 1=Q2...)	Department (Encoded e.g., 1)	Day (Encoded e.g., 0=Monday)
Team (1-12)	Targeted Productivity (0.0 - 1.0)	SMV (Standard Min Value)
No. of Workers (e.g., 8.0)	Month (1-12)	Over Time (Minutes)
Incentive (Value)	Idle Time (Minutes)	Idle Men (Count)
No. of Style Changes		

At the bottom of the form, there is a yellow star icon and a green "SUBMIT" button.

<b>about.html</b>	Screenshot of about page	[Screenshot of Prediction about page]
-------------------	--------------------------	---------------------------------------

**Employee Productivity Prediction**

This project implements a machine learning approach to predict employee performance using the [Garments Worker Productivity Dataset](#). The goal is to provide insights for talent retention and resource allocation.

**Talent Retention**  
Identify high-performing employees at risk of attrition to implement targeted retention strategies.

**Performance Improvement**  
Pinpoint areas where employees need additional support, coaching, or skill development opportunities.

**Resource Allocation**  
Optimize task assignments by matching employees with projects aligned with their predicted strengths.

**Model Performance**  
The best performing model identified during training was:  

- Algorithm: XGBoost Regressor
- R2 Score (Test Set): **-0.526** (As per project evaluation)
- Implementation: The production code uses the classification logic based on this model's output.

**Input 1 Result** Screenshot of submit.html showing the prediction "The employee is medium productive." [Screenshot of Medium Productivity Output]

**Enter Employee Features for Prediction**

Quarter (e.g., 0=Q1, 1=Q2...)	Department (Encoded e.g., 1)	Day (Encoded e.g., 0=Monday)
0	0	0
Team (1-12)	Targeted Productivity (0.0 - 1.0)	SMV (Standard Min Value)
8	0.80	26.16
No. of Workers (e.g., 8.0)	Month (1-12)	Over Time (Minutes)
2000	50	0
Incentive (Value)	Idle Time (Minutes)	Idle Men (Count)
0	0	59
No. of Style Changes	SUBMIT	
1		

The employee is medium productive.

**Input 2 Result** Screenshot of submit.html showing the prediction "The employee is Highly productive." [Screenshot of Highly Productive Output]

The employee is highly productive.

## 9. Conclusion and Deployment Documentation

### 9.1 Final Model Performance

The XGBoost Regressor achieved an R<sup>2</sup> score of **0.5258**, demonstrating a reasonable fit for predicting garment worker productivity.

### 9.2 Deployment Instructions

1. **File Structure:** Ensure the following files/folders exist in your project root:
  - app.py
  - gwp.pkl (The saved model)
  - templates/ (Containing home.html, about.html, predict.html, submit.html)
2. **Run Application:** Execute the Flask application from the terminal:  
`python app.py`
3. **Access UI:** Navigate to the local server address (`http://127.0.0.1:5000/`) in a web browser.