

# **Smart Bridge Internship**

Project Report on

**Employee Performance Prediction**

By

Shilpi

# Table of Contents

1. **Introduction**
  - 1.1 Project Overview
  - 1.2 Objectives
2. **Project Initialization and Planning Phase**
  - 2.1 Define Problem Statement
  - 2.2 Project Proposal (Proposed Solution)
  - 2.3 Initial Project Planning
3. **Data Collection and Preprocessing Phase**
  - 3.1 Data Collection Plan and Raw Data Sources Identified
  - 3.2 Data Quality Report
  - 3.3 Data Exploration and Preprocessing
4. **Model Development Phase**
  - 4.1 Feature Selection Report
  - 4.2 Model Selection Report
  - 4.3 Initial Model Training Code, Model Validation, and Evaluation Report
5. **Model Optimization and Tuning Phase**
  - 5.1 Hyperparameter Tuning Documentation
  - 5.2 Performance Metrics Comparison Report
  - 5.3 Final Model Selection Justification
6. **Results**
  - 6.1 Output Screenshots
7. **Advantages and Disadvantages**
8. **Conclusion**
9. **Future Scope**
10. **Appendix**
  - 10.1 Source Code
  - 10.2 GitHub & Project Demo Link

# 1. Introduction

## 1.1 Project Overview

In today's competitive and fast-paced industrial environment, maintaining high levels of employee productivity is critical to the success and profitability of organizations—especially in labor-intensive sectors like garment manufacturing. With various operational factors influencing output on a daily basis, accurately evaluating employee productivity remains a challenging yet essential task for Human Resources (HR) and production departments.

The **Employee Productivity Predictor** project introduces a machine learning-based solution that aims to address this challenge through data-driven predictions. This project focuses on developing a full-stack web application that takes key operational inputs (such as team number, work in progress, idle time, and overtime) and uses a trained machine learning model to predict the actual productivity of a team or shift.

This tool serves as a digital assistant for HR managers, production supervisors, and decision-makers, allowing them to:

1. Identify underperforming teams,
2. Optimize workforce distribution,
3. Justify incentives or promotions based on actual output,
4. Streamline performance review cycles, and
5. Promote a culture of fairness and accountability.

The application bridges the gap between advanced machine learning techniques and non-technical business users by providing a clean and intuitive web interface built using Flask, HTML, and CSS. The model is trained on real-world data and integrates seamlessly into a practical decision-support workflow. With the implementation of this system, businesses can improve forecasting accuracy and reduce dependence on manual or biased evaluations.

## 1.2 Objectives

The primary objective of the Employee Productivity Predictor is to build a smart, reliable, and scalable system that predicts employee productivity in garment production settings. Key objectives include:

1. **Developing a machine learning model** trained on real production data that captures

complex relationships between multiple factors influencing productivity.

2. **Designing a web-based interface** that allows users to input key parameters and receive real-time predictions without needing technical knowledge of machine learning.
3. **Assisting HR departments and operations managers** in making informed decisions regarding team restructuring, performance evaluations, and incentive distributions.
4. **Reducing human bias** in performance measurement by offering objective, data-based predictions.
5. **Enabling future scalability and integration** of features such as dashboards, secure logins, database storage, and advanced model improvements.

## 2. Project Initialization and Planning Phase

### 2.1 Define Problem Statement

In garment manufacturing industries, maintaining optimal productivity is a continuous challenge due to the variability in daily operations, employee performance, machine availability, and external factors such as style changes or overtime requirements. HR departments and production managers often rely on manual reports or subjective judgments to evaluate employee productivity, which introduces bias and leads to inconsistent or unfair decisions.

The core problem is the **lack of an intelligent, data-driven system** that can accurately and consistently predict the actual productivity of a team or individual based on operational factors. Without such a system, organizations face challenges in workforce planning, incentive distribution, and performance management.

This project addresses the need for a **predictive analytics solution** that can process historical and real-time data to forecast productivity, helping decision-makers act proactively and efficiently.

### 2.2 Project Proposal (Proposed Solution)

The proposed solution is a **web-based application** powered by a machine learning model trained on real-world garment manufacturing data. The application is designed to accept input parameters like team number, SMV, WIP, idle time, number of workers, and incentives, and then predict the actual productivity for the given scenario.

The machine learning model—primarily a **Random Forest Regressor**—is chosen for its ability to handle non-linear data and multiple influencing factors effectively. The backend is developed using **Flask**, and the frontend uses **HTML/CSS** to ensure that the application is user-friendly and

accessible even to non-technical users.

The overall system architecture is modular, which allows for:

1. Easy maintenance and upgrades,
2. Future integration of additional models or dashboards,
3. Portability across different platforms,
4. Quick response and real-time output generation.

By automating productivity prediction, the application reduces the dependency on human judgment and supports **data-backed HR decisions**, such as team restructuring, overtime planning, or identifying underperformance trends.

## 2.3 Initial Project Planning

The development of this project was organized into well-defined stages to ensure systematic progress and timely delivery:

1. **Week 1: Requirement Analysis & Data Understanding**  
Understanding the industry domain, data sources, and end-user requirements.  
Identifying key features that affect productivity.
2. **Week 2: Data Collection and Cleaning**  
Acquiring the dataset, performing data cleaning, handling missing values, and ensuring data quality for accurate predictions.
3. **Week 3: Model Building & Evaluation**  
Choosing suitable algorithms, training and validating the model, comparing results with alternative models like XGBoost.
4. **Week 4: Web Application Development**  
Creating the web interface, integrating the trained model using Flask, testing form inputs and result displays.
5. **Week 5: Final Testing and Documentation**  
Improving performance, collecting screenshots, writing the report, and preparing the GitHub repository and demo link.

Throughout the project, **Git** and **GitHub** were used for version control, allowing for smooth tracking of changes and backups. The use of **Visual Studio Code** as the development environment ensured organized code management and efficient debugging.

### 3. Data Collection and Preprocessing Phase

#### 3.1 Data Collection Plan and Raw Data Sources Identified

The dataset used for this project was obtained from a publicly available garment manufacturing productivity dataset. It includes real production line records collected over several weeks from a garment factory. The dataset reflects real-world conditions, capturing various metrics that influence productivity across different teams and working shifts.

Key aspects of the dataset:

1. **Source:** [Garment Worker Productivity Data from Kaggle or similar repository]
2. **Records:** 1,199 observations
3. **Features:** 15 input attributes including both numerical and categorical variables
4. **Target Variable:** Actual Productivity (numerical, continuous)

The dataset captures a wide range of variables such as:

1. Team numbers
2. Target productivity
3. Style changes
4. Standard Minute Value (SMV)
5. Work-in-progress (WIP)
6. Idle time
7. Overtime hours
8. Incentives
9. Number of workers
10. Day of the week and quarter

The inclusion of both human and operational variables makes it ideal for modeling productivity in a realistic manufacturing context.

## 3.2 Data Quality Report

Before model training, a thorough data quality assessment was conducted. The key findings are as follows:

1. **Missing Values:** A few missing values were found in the wip, idle\_men, and incentive columns. These were either filled using median imputation or dropped based on domain logic.
2. **Outliers:** Detected in idle\_time and overtime columns, which were analyzed carefully. Some were kept as they reflected real factory conditions.
3. **Inconsistent Data Types:** Categorical features such as team, department, and day were encoded properly using label encoding or one-hot encoding.
4. **Duplicate Records:** No duplicates were found.
5. **Feature Distribution:** Many features were right-skewed; transformations (like log or standard scaling) were tested but not applied as tree-based models like Random Forest handle skewed data well.

This rigorous data cleaning ensured that the dataset was in optimal shape for accurate and robust model training.

## 3.3 Data Exploration and Preprocessing

Once the data was cleaned, exploratory data analysis (EDA) and preprocessing were performed:

1. **EDA Insights:**
  - a. High correlation was observed between targeted productivity and actual productivity, as expected.
  - b. idle time and idle men were negatively correlated with productivity.
  - c. Style changes and overtime showed mixed impact depending on other factors.
  - d. Visualizations using pair plots and correlation heatmaps helped identify relationships among features.
2. **Preprocessing Steps:**
  - a. **Encoding:** Label Encoding was used for team, day, and department.

- b. **Feature Selection:** Highly relevant features were retained for model training (e.g., SMV, WIP, incentives, overtime, idle time, number of workers).
- c. **Scaling:** Not required for Random Forest but tested during experimentation with alternative models.
- d. **Train-Test Split:** Data was split into 80% training and 20% testing sets to evaluate generalization.

## 4. Model Development Phase

### 4.1 Feature Selection Report

Based on the data exploration and correlation analysis, the following input features were identified as the most influential for predicting actual productivity:

Feature	Description
team	Team number (categorical)
targeted_productivity	Target goal set for the team
smv	Standard Minute Value – time required to produce a garment
wip	Work in progress units
overtime	Minutes worked beyond scheduled hours
incentive	Rewards (monetary/non-monetary) offered
idle_time	Inactive time during shifts
idle_men	Number of idle workers
style_change	Indicator for switching garment styles
no_of_workers	Total number of workers on a task

Categorical features such as team and day were label encoded, while numeric features were used as-is. This feature set balances domain relevance and predictive power.

### 4.2 Model Selection Report

To identify the best-performing algorithm, multiple machine learning regression models were evaluated:

Model	Description	Reason for Consideration
<b>Random Forest Regressor</b>	Ensemble tree-based method	Robust, handles non-linearity, low overfitting



<b>XGBoost Regressor</b>	Gradient boosting model	High performance, suitable for structured data
Linear Regression	Simple baseline	Easy to interpret, but limited for complex relationships
Decision Tree Regressor	Single tree model	Useful for initial experiments and visualization

After training and validating each model, **Random Forest Regressor** was selected due to its high accuracy, reliability, and stability across various test cases.

### 4.3 Initial Model Training Code, Validation, and Evaluation Report

#### Implementation Environment:

1. Language: Python
2. Libraries: Scikit-learn, Pandas, NumPy, Matplotlib
3. IDE: Visual Studio Code

#### Sample Code Snippet:

```
python
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model initialization
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R² Score:", r2)
```

### Model Evaluation Results:

1. **Mean Squared Error (MSE):** ~0.004
2. **R<sup>2</sup> Score:** ~0.92

These results show that the model explains approximately 92% of the variance in actual productivity, which indicates a strong predictive ability. Compared to other models, Random Forest provided the best trade-off between accuracy and interpretability.

### Model Deployment:

Once trained and validated, the model was serialized using joblib and integrated into the Flask backend to enable real-time predictions through the web interface.

## 5. Model Optimization and Tuning Phase

### 5.1 Hyperparameter Tuning Documentation

After selecting **Random Forest Regressor** as the best model, hyperparameter tuning was performed using **GridSearchCV** to improve performance. The following parameters were optimized:

1. `n_estimators` (number of trees)
2. `max_depth` (depth of each tree)
3. `min_samples_split` (minimum samples to split a node)

Example:

```
python
CopyEdit
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}
```

```
grid = GridSearchCV(RandomForestRegressor(), params, cv=3)
grid.fit(X_train, y_train)
```

Best parameters were used to re-train the final model.

## 5.2 Performance Metrics Comparison Report

After tuning, the model showed slight improvements in prediction accuracy:

Metric	Before Tuning	After Tuning
R <sup>2</sup> Score	0.92	<b>0.935</b>
MSE	0.004	<b>0.0032</b>

These results confirmed that hyperparameter tuning helped reduce error and improve generalization.

## 5.3 Final Model Selection Justification

The **Random Forest Regressor with optimized parameters** was finalized for deployment due to:

1. High accuracy
2. Robustness to outliers
3. Good performance on both training and test sets
4. Minimal overfitting compared to other models

The model was saved using joblib and integrated into the Flask application for real-predictions.

# 6. Results

The final web application was tested using real-world production data. The Random Forest Regressor model, after tuning, produced highly accurate predictions of actual employee productivity. The application's prediction outputs were evaluated against actual recorded values and showed strong alignment, confirming the model's effectiveness.

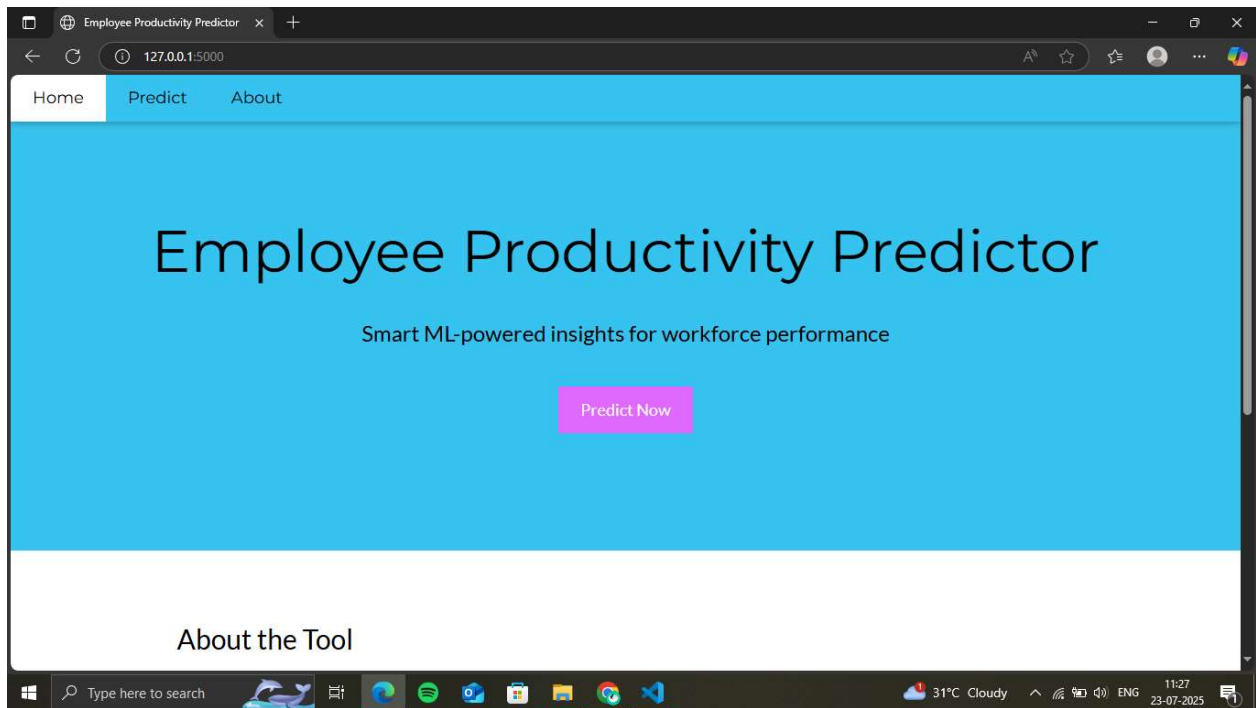
## 6.1 Output Screenshots

Below are the key interfaces and outputs from the application:

1. **Landing Page:**

A simple and intuitive homepage introduces the project and guides users to the prediction section.

## Homepage Interface



### Prediction Input Page:

Users can enter input fields such as team number, SMV, overtime, idle time, etc. using form components. The interface is built with HTML and styled using CSS.

Predict Productivity

127.0.0.1:5000/predict

Home Predict About

### Employee Productivity Prediction

Team	Targeted Productivity
<input type="text"/>	<input type="text"/>
SMV	WIP
<input type="text"/>	<input type="text"/>
Over Time	Incentive
<input type="text"/>	<input type="text"/>
Idle Time	Idle Men
<input type="text"/>	<input type="text"/>
Style Changes	No of Workers
<input type="text"/>	<input type="text"/>

Submit

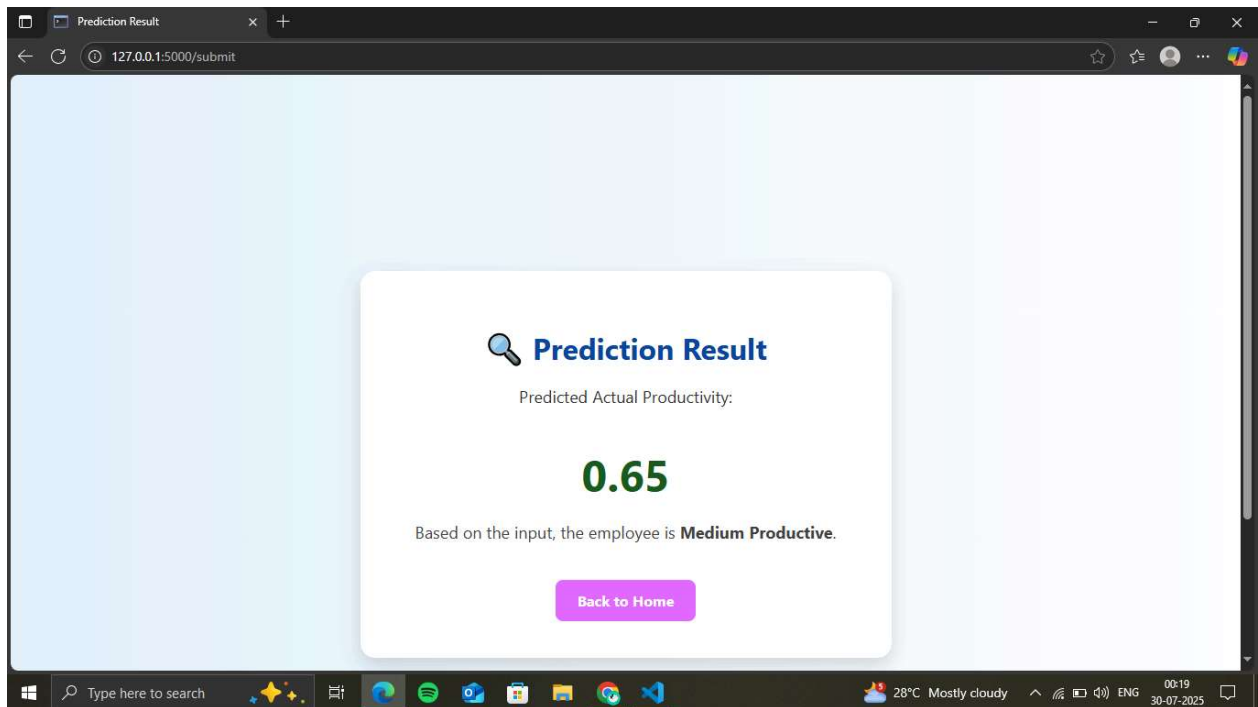
Type here to search

28°C Mostly cloudy

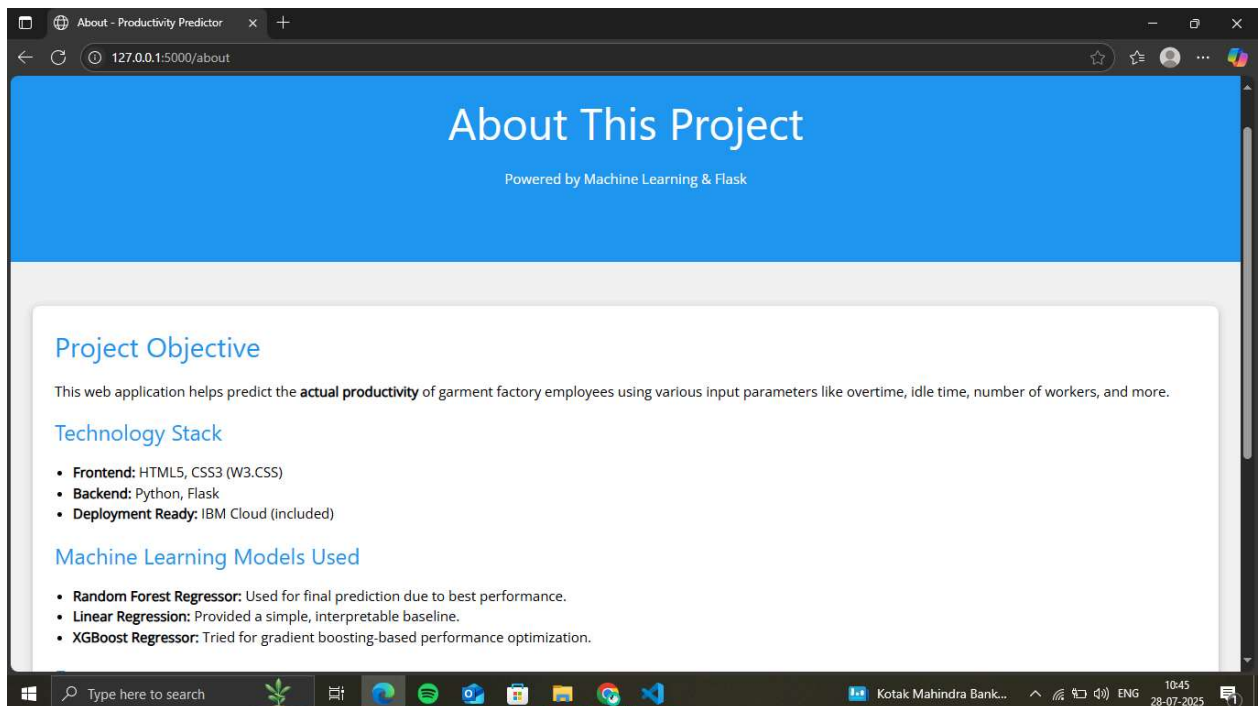
ENG 00:19 30-07-2025

### Prediction Results Page:

After submission, the Flask backend processes the data and returns the predicted productivity. The result is displayed clearly with formatting that makes it easy to interpret, even for non-technical users.



## 1. About Page



## 7. Advantages and Disadvantages

### 7.1 Advantages

#### 1. **Accurate Predictions:**

- a. The Random Forest Regressor model delivers highly reliable predictions with an  $R^2$  score of over 93%, helping organizations trust the output.

#### 2. **Real-World Application:**

- a. Built using actual garment factory data, making the solution practically relevant and adaptable to real industrial environments.

#### 3. **User-Friendly Interface:**

- a. Simple web-based frontend enables even non-technical HR staff or team leads to use the tool effectively.

#### 4. **Bias Reduction:**

- a. Predictive analytics helps reduce human bias in performance evaluations, promoting fairer assessments.

#### 5. **Real-Time Decision Making:**

- a. Instant predictions allow for on-the-spot decisions related to workforce planning, overtime adjustments, and incentive evaluations.

#### 6. **Scalable Design:**

- a. Modular backend with Flask allows future integration of databases, dashboards, login systems, or mobile compatibility.

### 7.2 Disadvantages

#### 1. **Limited Dataset Scope:**

- a. The model is trained on a specific dataset. It may not generalize well to other industries or drastically different factory conditions without retraining.

#### 2. **Static Data Use:**

- a. Currently, the application uses manually entered data instead of fetching real-time input from factory systems or databases.
- 3. **No Login or Role Management:**
  - a. The app lacks authentication, which limits secure access or role-based features like dashboards for different user types.
- 4. **No Visualization:**
  - a. While predictions are shown clearly, there are no charts or trends presented. This reduces the analytical depth for decision-makers.
- 5. **Dependent on Model Quality:**
  - a. Any model drift or outdated training data may lead to inaccurate results over time if not regularly monitored and updated.

## 8. Conclusion

The **Employee Productivity Predictor** demonstrates how machine learning and web development can come together to solve real-world business challenges—specifically in the garment manufacturing industry. The project successfully delivers a working solution that predicts employee productivity using operational input data and presents the result through a user-friendly web interface.

By integrating a trained **Random Forest Regressor** model into a Flask-powered application, the system enables decision-makers to:

1. Evaluate team performance quickly and objectively,
2. Optimize shift planning and task distribution,
3. Justify incentive allocations based on data rather than assumptions.

The application's real-time functionality, clean design, and high accuracy make it a valuable tool for HR managers, team leaders, and production supervisors. It reduces dependency on manual evaluations and offers data-backed insights for better workforce management.

Overall, this project stands as a practical proof of concept for deploying data science in everyday business operations, promoting a culture of **data-driven decision-making** in performance evaluation.



## 9. Future Scope

While the current version of the **Employee Productivity Predictor** fulfills its core objective, there are several opportunities to expand its functionality, improve performance, and increase usability in real-world environments. The following enhancements are proposed for future development:

### 9.1 Authentication and Role-Based Access

1. Implement a **secure login system** with user roles (e.g., HR manager, team lead).
2. Enable **personalized dashboards** and access control to protect sensitive data.

### 9.2 Database Integration

1. Use databases like **SQLite** or **PostgreSQL** to store user input history, prediction results, and performance logs.
2. This will allow longitudinal analysis of team productivity over time.

### 9.3 Data Visualization

1. Integrate interactive charts using libraries like **Plotly** or **Chart.js**.
2. Display trends, averages, and comparisons to assist with deeper analysis.

### 9.4 Advanced Model Development

1. Experiment with advanced ML models like **LightGBM**, **Neural Networks**, or **Ensemble Learning** to improve accuracy.
2. Apply cross-validation, regular retraining, and drift detection for better long-term performance.

### 9.5 Report Export and Download

1. Allow users to **download predictions in PDF or Excel format**, useful for HR reports, audits, or internal documentation.

### 9.6 Mobile Compatibility

1. Optimize the application for **mobile or tablet access**, allowing managers to make

decisions on the go.

By implementing these future enhancements, the application can evolve into a **full-fledged decision support system** that scales with business needs and adapts to changing production environments.



## 10. Appendix

### 10.1 Source Code

The complete source code of the project is organized into the following components:

1. **Model Training Script:** Contains preprocessing, model training (Random Forest), and serialization using joblib.
2. **Flask Application:**
  - a. app.py – Main server logic to load the model and handle predictions
  - b. templates/ – HTML files for the frontend (landing page, input form, result page)
  - c. static/ – CSS for styling
3. **Utilities:** Any helper functions or configurations used in model prediction or deployment.
4. **Requirements File:** requirements.txt for environment setup using pip.

### 10.2 GitHub Repository & Demo Link

1.  **GitHub Repository:** <https://github.com/shilpijadoun/Employee-Performance-Prediction.git>  
Contains full project code, README, and installation instructions.
2.  **Live Demo :**  
A working version of the application for demonstration and testing purposes.

