FINAL REPORT

GARMENT WORKER PRODUCTIVITY PREDICTION

TEAM:

NIKHIL PULAGAM

MULE VIGNESH

SHREYAS REDDY GUVVALA

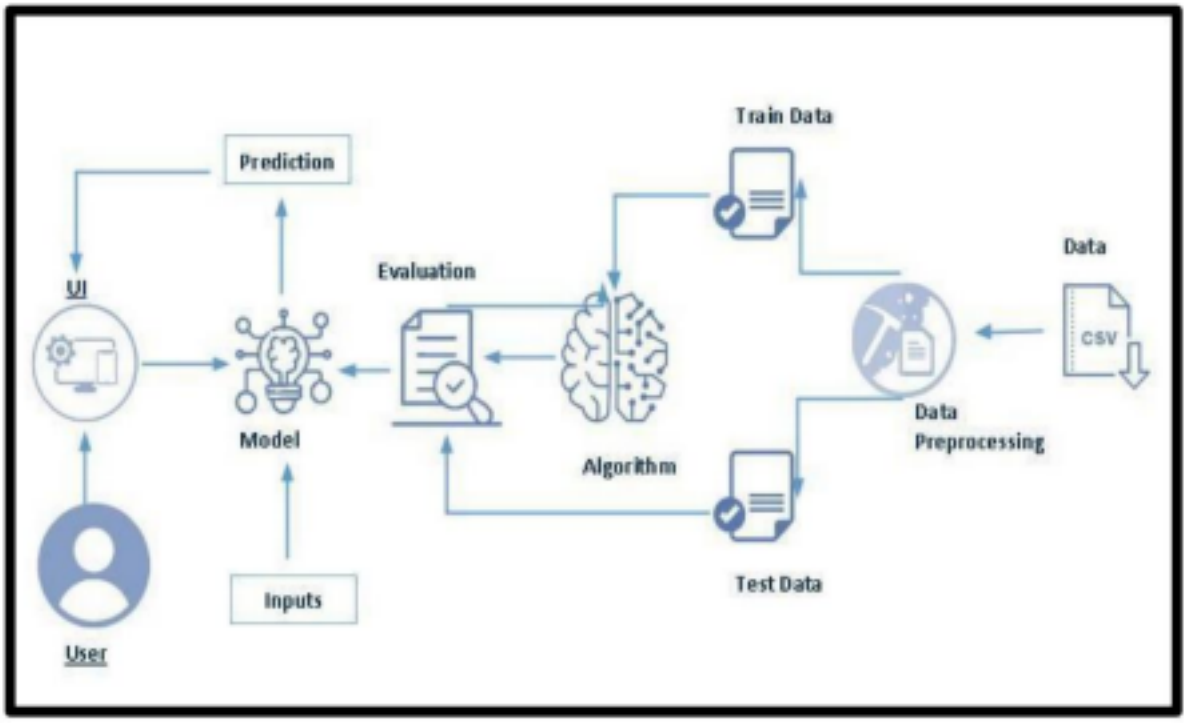
1. Introduction

a. Project overviews:

The garment industry is one of the largest industries in the world, and garment worker productivity is a crucial factor in determining the success and profitability of a company. In this project, we aim to develop a machine learning model that predicts the productivity of garment workers based on a given set of features. Our dataset contains information on various attributes of garment production, including the quarter, department, day, team number, time allocated, unfinished items, over time, incentive, idle time, idle men, style change, number of workers, and actual productivity. We will use this dataset to train and evaluate our predictive model.

The development of an accurate garment worker productivity prediction model using machine learning can have significant implications in various domains, including manufacturing, human resources, and supply chain management. This model can help companies identify the factors that affect worker productivity and take corrective actions to improve efficiency, reduce costs, and enhance their competitiveness in the market.

Technical Architecture:



b. Objectives:

Define Problem / Problem Understanding

- Specify the business proble,

Business requirements
Literature Survey
Social or Business Impact

Data Collection & Preparation

- Collect the dataset
- Data Preparation

Exploratory Data Analysis

- Descriptive statistical
- Visual Analysis

Collect the dataset

- Training the model in multiple algorithms
- Testing the model

Performance Testing & Hyperparameter Tuning

- Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning

Model Deployment

- Save the best model
- Integrate with Web Framework

Project Demonstration & Documentation

- Record explanation Video for project end to end solution
- Project Documentation-Step by step project development procedure

2. Project Initialization and Planning Phase

    a. Define Problem Statement

    To ensure that the garment worker productivity prediction model meets business requirements and can be deployed for public use, it should follow the following rules and requirements: Accuracy: The model should have a high level of accuracy in predicting worker productivity, with a low margin of error. This is crucial to ensure that the predictions are reliable and trustworthy.

    - Privacy and security: The model should be developed in accordance with privacy and security regulations to protect user data. This includes ensuring that sensitive data is stored securely and implementing proper data access controls.
    - Interpretability: The model should be interpretable, meaning that the predictions can be explained and understood by the end-users. This is important to build trust in the model and to allow users to make informed decisions based on the predictions.
    - User interface: The model should have a user-friendly interface that is easy to use and understand. This is important to ensure that the model can be deployed for public use, even by individuals who may not have technical expertise.

    2.2Project Proposal

    A literature survey for a garment worker productivity prediction project would involve researching and reviewing existing studies, articles, and other publications related to machine learning in the field of manufacturing and workforce management. The survey would aim to gather information on current methods for predicting productivity in the

garment industry, including feature selection, data pre-processing, and machine learning algorithms used for prediction.

The survey would also examine any gaps in knowledge and research opportunities in the field of garment worker productivity prediction, including the use of new machine learning techniques, such as reinforcement learning, and the integration of other data sources, such as wearable technology and environmental sensors.

a. Initial Project Planning

Social Impact: The garment worker productivity prediction model has the potential to improve the lives of garment workers by promoting fair and efficient workforce management practices. The model can identify factors that affect worker productivity and suggest ways to improve working conditions, incentive schemes, and training programs. This can lead to higher job satisfaction, better pay, and improved working conditions for garment workers.
Business Impact: The garment worker productivity prediction model can have significant implications for the garment manufacturing industry. The model can help manufacturers optimize their workforce management strategies, reduce idle time, and increase productivity. This can lead to higher profits, lower production costs, and improved quality of goods produced. Additionally, the model can assist in identifying areas for process improvement, such as reducing rework and improving supply chain efficiency.

3. Data Collection and Preprocessing Phase

a. Data Collection Plan and Raw Data Sources Identified

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

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset. Download the dataset from the above link. Let us understand and analyze the dataset properly using visualization techniques.

Note: There are several approaches for understanding the data. But we have applied some of it here. You can also employ a variety of techniques.

Activity 1.1: Importing the libraries

Import the libraries required for this machine learning project, as shown in the image below.

```
# Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
```

Activity 1.2: Read the Dataset

Our dataset format could be in .csv, excel files,.txt, .json, and so on. With the help of pandas, we can read the dataset.

Since our dataset is a csv file, we use read_csv() which is pandas function to read the dataset. As a parameter we have to give the directory of the csv file.

df.head() will display first 5 rows of the dataset.

```
df = pd.read_csv('garments_worker_productivity.csv')
```

```
df.head()
```

Python

quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	59.0	0.940725
Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	8.0	0.886500
Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	56.0	0.800382

b. Data Exploration and Preprocessing

Data preparation, also known as data preprocessing, is the process of cleaning, transforming, and organizing raw data before it can be used in a data analysis or machine learning model.

The activity include following steps:

- removing missing values
- handling outliers
- encoding categorical variables
- normalizing data

Note: These are general steps to take in pre-processing before feeding data to machine learning for training. The pre-processing steps differ depending on the dataset. Depending on the condition of your dataset,

you may or may not have to go through all these steps.

Activity 2.1: Handling Missing Values

Let's first figure out what kind of data is in our columns by using df.info(). We may deduce from this that our columns include data of the types "object", "float64" and “int64”.

```
df.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
```

```
df.isna().sum()
```

```
quarter                  0
department               0
day                      0
team_number              0
time_allocated           0
unfinished_items       506
over_time                0
incentive                0
idle_time                0
idle_men                 0
style_change             0
no_of_workers            0
actual_productivity      0
dtype: int64
```

This line of code is used to count the number of missing or null values in a pandas DataFrame. It returns a list of the total number of missing values in each column of the DataFrame.

```
df['unfinished_items'] = df['unfinished_items'].fillna(df['unfinished_items'].mean())
```

This line of code fills the missing values in the “unfinished_items” column with the column mean.

Activity 2.2: Handling Independent Columns

```
df.drop(['date', 'targeted_productivity'], axis=1, inplace=True)
df.head()
```

	quarter	department	day	team	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
0	Quarter1	sweing	Thursday	8	26.16	1108.0	7080	98	0.0	0	0	59.0	0.940725
1	Quarter1	finishing	Thursday	1	3.94	NaN	960	0	0.0	0	0	8.0	0.886500
2	Quarter1	sweing	Thursday	11	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
3	Quarter1	sweing	Thursday	12	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
4	Quarter1	sweing	Thursday	6	25.90	1170.0	1920	50	0.0	0	0	56.0	0.800382

+ Code + Markdown

This line of code drops the columns date and targeted_productivity from the dataframe.

```
df = df.rename(columns={
    'team' : 'team_number',
    'smv' : 'time_allocated',
    'wip' : 'unfinished_items',
    'no_of_style_change' : 'style_change'
})
df
```

The rename() method is being used to change the names of some columns. The method takes a dictionary as its argument, where the keys are the original column names and the values are the new names.

```
df['quarter'].unique()
```

```
array(['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4', 'Quarter5'],
      dtype=object)
```

```
df['quarter'] = df['quarter'].str.replace('Quarter5','Quarter1')
```

The first line of code selects a specific column named "quarter" from a table of data. The unique()method is then used to list all the different values in that column.
The second line of code changes some of the values in the "quarter" column. Specifically, it replaces any instances of the text "Quarter5" with the text "Quarter1" using the str.replace()method.

```
# Extract the numerical part from the 'quarter' column
df['quarter'] = df['quarter'].str.extract(r'(\d+)')
df['quarter']
```

The first line of code is using the str.extract() method on the "quarter" column to extract the numerical part of each value. It uses a regular expression pattern r'(\d+)' to match any sequence of one or more digits in each value.
The second line of code is assigning the modified "quarter" column back to the same "quarter" column, effectively replacing the original column with the modified one.

```
df['department'] = df['department'].str.replace('sweing','sewing')
df['department'] = df['department'].str.replace('finishing ','finishing')
```

The first line of code is using the str.replace() method on the "department" column to replace any instances of the misspelled word

"sweing" with the correct spelling "sewing".
The second line of code is also using the str.replace() method on the "department" column to replace any instances of the word "finishing " (with a space at the end) with the word "finishing" (without a space at the end). This will remove the extra space and standardize the spelling of the word.

```
df['team_number'] = df['team_number'].astype(int)
df['over_time'] = df['over_time'].astype(int)
df['incentive'] = df['incentive'].astype(int)
df['idle_time'] = df['idle_time'].astype(int)
df['idle_men'] = df['idle_men'].astype(int)
df['style_change'] = df['style_change'].astype(int)
```

```
df['time_allocated'] = df['time_allocated'].astype(int)
```

```
df['unfinished_items'] = df['unfinished_items'].astype(int)
```

```
df['idle_time'] = df['idle_time'].astype(int)
```

```
df['no_of_workers'] = df['no_of_workers'].astype(int)
```

Each line of code is using the astype() method to convert the data type of a specific column to an integer type. The column name is specified on the left-hand side of the equation, and the new integer type is specified as an argument to the astype() method.
Activity 2.3: Handling Categorical Values

```
lc = LabelEncoder()
```

```
print('Before encoding: ', df['department'].unique())
df['department'] = lc.fit_transform(df['department'])
print('After encoding: ',df['department'].unique())
```

This code is encoding the values in a column named "department" by using a LabelEncoder() object to convert the original values into numerical encoded values. The original and encoded values are printed before and after the encoding is performed.

```
Before encoding:  ['sewing' 'finishing']
After encoding:  [1 0]
```

```
print('Before encoding: ', df['day'].unique())
df['day'] = lc.fit_transform(df['day'])
print('After encoding: ',df['day'].unique())
```

```
Before encoding:  ['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
After encoding:  [3 1 2 0 4 5]
```

This code is encoding the values in a column named "day" by using a LabelEncoder() object to convert the original values into numerical encoded values. The original and encoded values are printed before and after the encoding is performed.

3.3 Data Exploration and Preprocessing

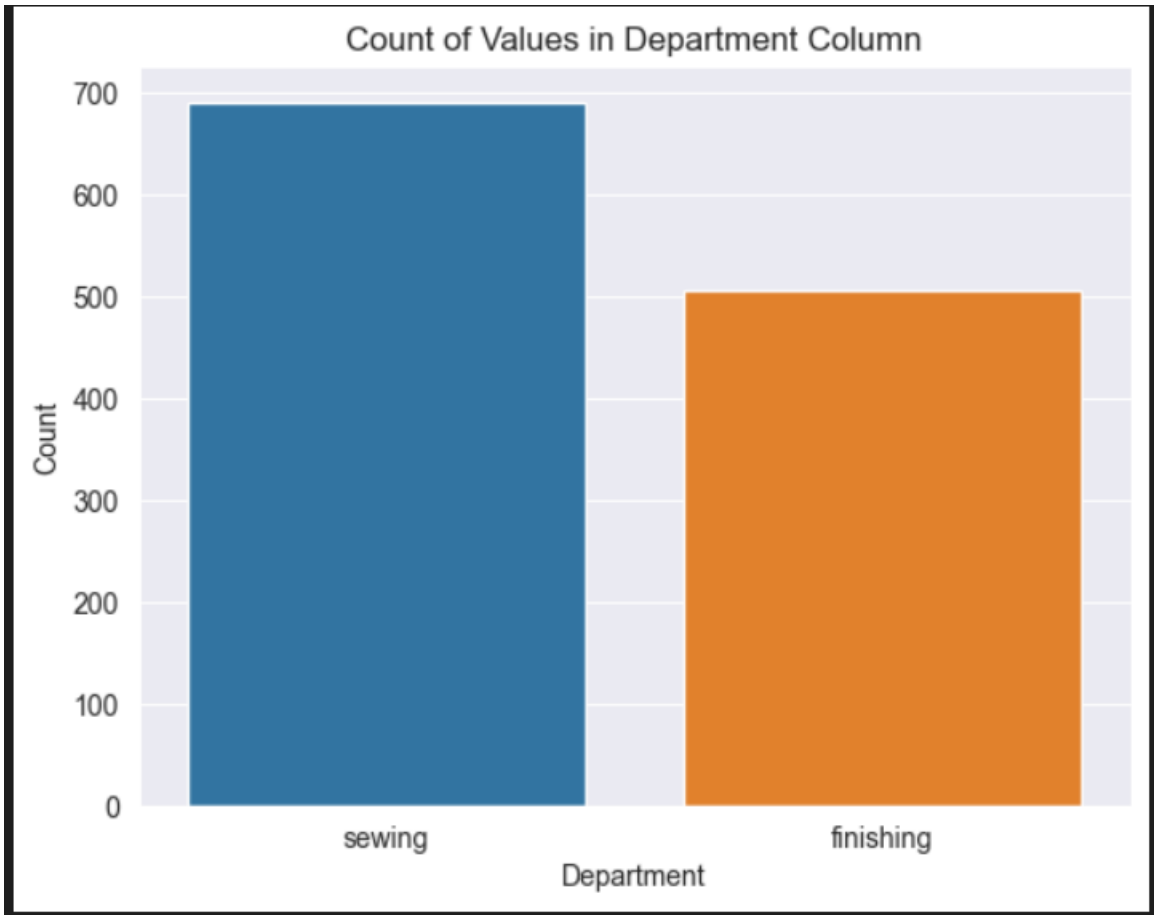
Visual analysis is the process of examining and understanding data via the use of visual representations such as charts, plots, and graphs. It is a method for quickly identifying patterns, trends, and outliers in data, which can aid in gaining insights and making sound decisions.

Activity 2.1: Univariate analysis

Univariate analysis is a statistical method used to analyse a single variable in a dataset. This analysis focuses on understanding the distribution, central tendency, and dispersion of a single variable.

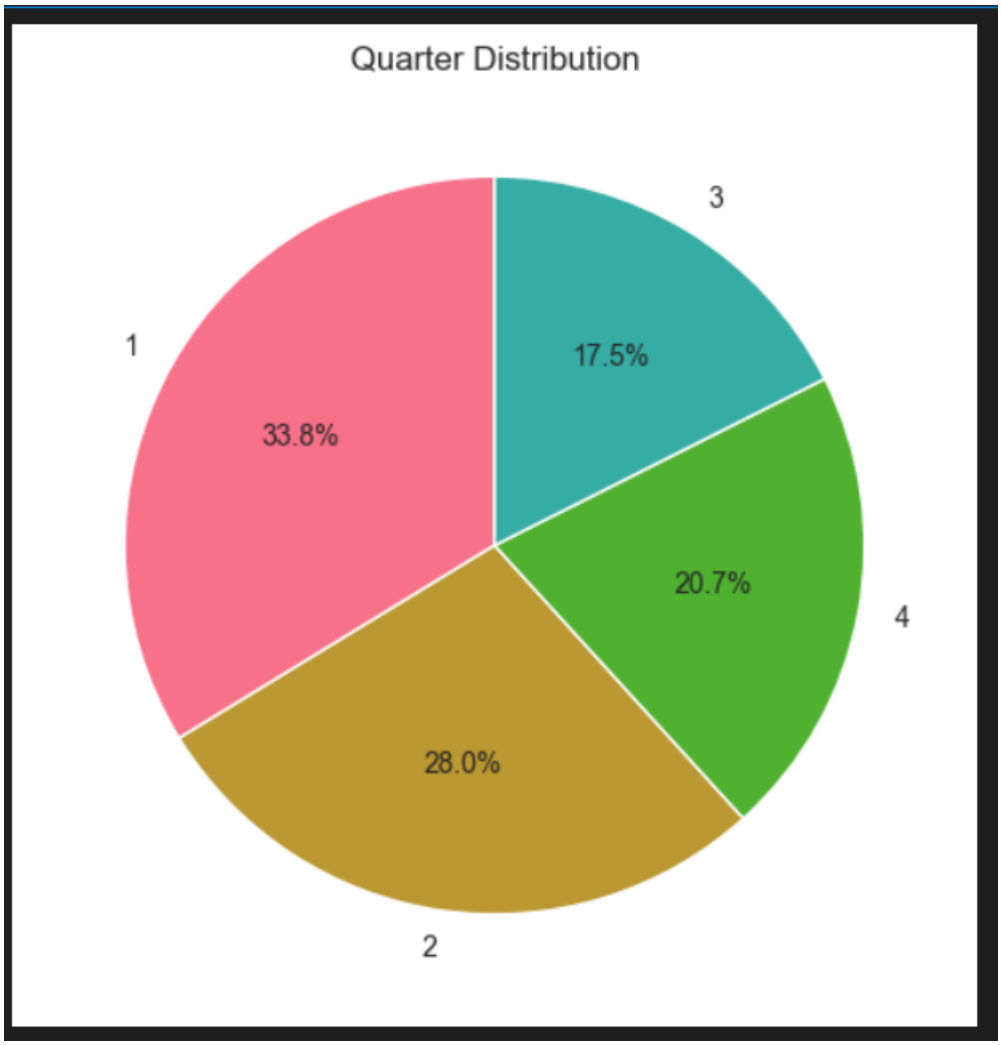
```
sns.countplot(data=df, x='department')
plt.xlabel('Department')
plt.ylabel('Count')
plt.title('Count of Values in Department Column')
plt.show()
```

This code creates a bar chart using the Seaborn library to show the number of occurrences of each unique value in the "department" column of a Pandas DataFrame. The x-axis represents the unique values of the "department" column, and the y-axis represents the number of times each unique value appears in the column. The plt.xlabel(), plt.ylabel(), and plt.title() functions are used to add labels and a title to the plot. Finally, plt.show() is used to display the plot.




```
quarter_counts = df['quarter'].value_counts()
plt.figure(figsize=(8, 6))
sns.set_palette("husl")  # Set custom color palette
plt.pie(quarter_counts, labels=quarter_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Quarter Distribution')
plt.show()
```

This code generates a pie chart using the Seaborn library to show the distribution of different quarters in a dataset. The value_counts() function counts the number of occurrences of each quarter in the dataset, and the resulting counts are used to create the pie chart. The pie chart shows the proportion of the total number of entries in the dataset that corresponds to each quarter. The title of the pie chart is "Quarter Distribution".



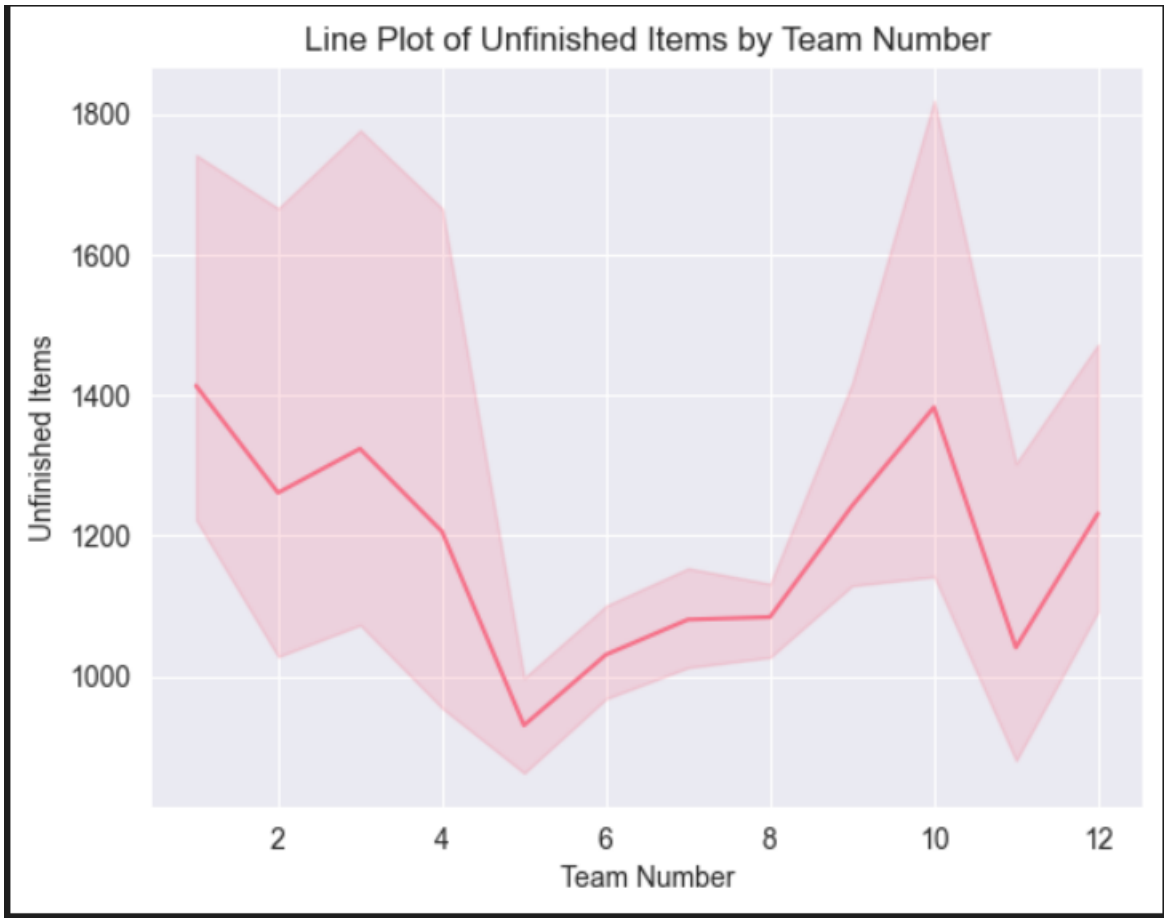
Activity 2.2: Bivariate analysis

Bivariate analysis is a statistical method used to analyse the relationship between two variables in a dataset. This analysis focuses on examining how changes in one variable are related to changes in another variable.

```
sns.lineplot(data=df, x='team_number', y='unfinished_items')
plt.xlabel('Team Number')
plt.ylabel('Unfinished Items')
plt.title('Line Plot of Unfinished Items by Team Number')
plt.show()
```

This code generates a line plot using the Seaborn library to show the relationship between team number and the number of unfinished items.

The line plot shows how the number of unfinished items varies across different teams. The x-axis represents the team number, and the y-axis represents the number of unfinished items. The title of the line plot is "Line Plot of Unfinished Items by Team Number".

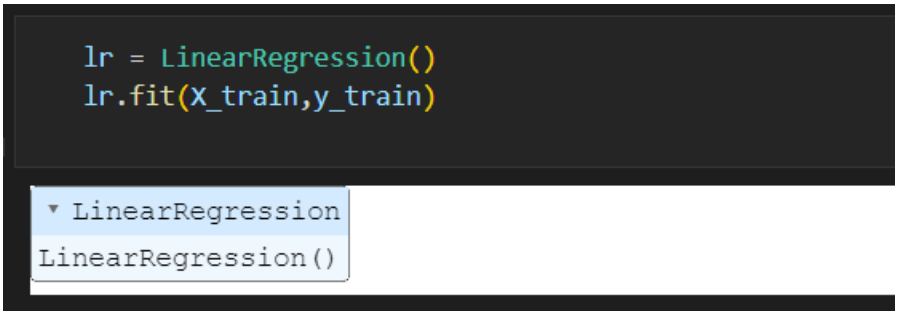


4. Model Development Phase

a. Feature Selection Report

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying seven regression algorithms. The best model is saved based on its performance.

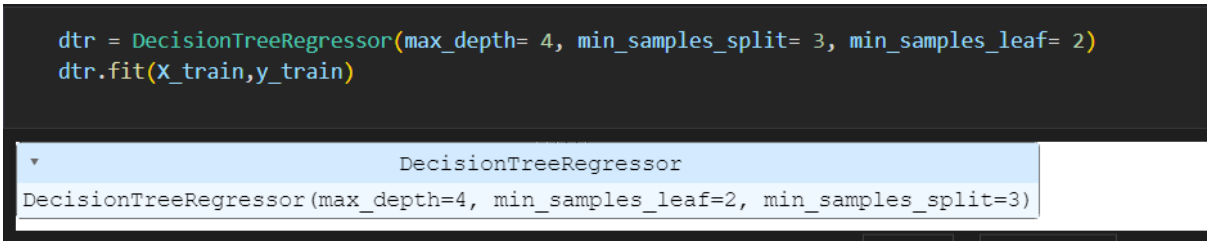
Activity 1.1: Linear Regression Model



This code performs linear regression modeling using the scikit-learn library.

It creates a LinearRegression object named "lr". The "fit" method is then called on the training data X_train and y_train. This method trains the model by finding the coefficients for the linear equation that best fits the data.

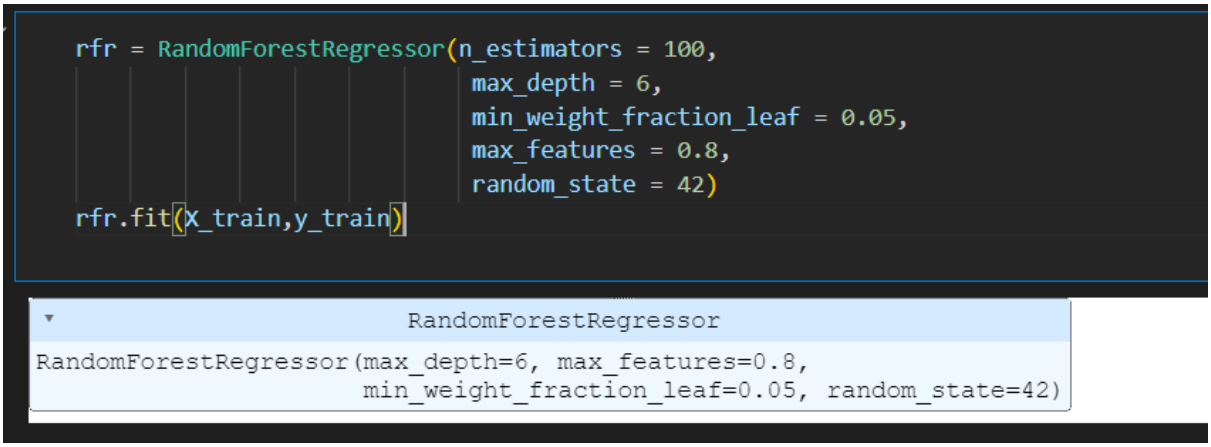
Activity 1.2: Decision Tree Regressor Model



This code is training a decision tree regression model using the training data (X_train and y_train) with the specified hyperparameters

(max_depth=4, min_samples_split=3, min_samples_leaf=2). The model is stored in the variable 'dtr'. After training, the model will be able to make predictions on new data based on the relationships learned from the training data.

Activity 1.3: Random Forest Regressor Model



The code creates an instance of the RandomForestRegressor class with certain hyperparameters set. The hyperparameters specify the number of trees to use in the random forest, the maximum depth of each tree, the minimum weight fraction required to be at a leaf node, the maximum number of features to consider when splitting a node, and a random state to ensure reproducibility of results. The rfr object is then trained on the training data X_train and y_train using the fit method.

The trained model can then be used to make predictions on new data.

b. Model Selection Report

Activity 1.4: Gradient Boosting Regressor Model

```
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=42)
gbr.fit(X_train,y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=1, random_state=42)
```

The code is fitting a Gradient Boosting Regressor model to the training data X_train and y_train using n_estimators equal to 100 (the number of trees in the forest), learning_rate equal to 0.1 (the step size shrinkage used to prevent overfitting), max_depth equal to 1 (the maximum depth of the individual regression estimators), and random_state equal to 42 (to ensure reproducibility of results).

The fitted model can then be used to make predictions on new data using the predict() method.

Activity 1.5: Extreme Gradient Boost Regressor Model

```
xgb = XGBRegressor(n_estimators=300, learning_rate=0.05,
                   max_leaves = 3, random_state = 1)
xgb.fit(X_train, y_train)
```

This code is training an XGBoost regression model. The XGBRegressor function is being used to create the model. The parameters passed to the function include the number of estimators, learning rate, maximum number of leaves, and random state. The model is trained on the training set (X_train and y_train) using the fit() method. Once trained, the model can be used to make predictions on new data.

4.3 Initial Model Training Code, Model Validation and Evaluation Report

Activity 1.6: Bagging Regressor Model

```
# Define base model
base_model = XGBRegressor(n_estimators=700, learning_rate=0.06, max_depth=2, max_leaves=3, random_state=1)

# Create bagging regressor
bagging_reg = BaggingRegressor(base_model, n_estimators=100, random_state=42)


# Fit bagging regressor
bagging_reg.fit(X_train, y_train)
```

This code creates a machine learning model using the XGBoost algorithm. The model is designed to predict a numeric value (the target variable) based on a set of input variables.

To improve the accuracy of the model, the Bagging technique is used. This involves creating multiple versions of the model, each with slightly different training data, and combining their predictions to create a final prediction.

Finally, the model is trained using a set of input data (X_train) and the corresponding target values (y_train). This involves adjusting the weights of the various components of the model until it can accurately predict the target values based on the input data.

Activity 1.7: Boosting Regressor Model

```
# Define base model
base_model = XGBRegressor(n_estimators=700, learning_rate=0.06, max_depth=2, max_leaves=3, random_state=1)

# Create AdaBoost regressor
boosting_reg = AdaBoostRegressor(base_model, n_estimators=100, learning_rate=0.1, random_state=42)

# Fit AdaBoost regressor
boosting_reg.fit(X_train, y_train)
```

This code creates a machine learning model to predict a numeric value based on a set of input variables using the XGBoost algorithm as the base model.

To improve the accuracy of the model, the AdaBoost technique is used. AdaBoost stands for Adaptive Boosting and works by creating multiple versions of the model, each with slightly different training data, and weighting the predictions of each model based on its accuracy.

Finally, the model is trained using a set of input data (X_train) and the corresponding target values (y_train). This involves adjusting the weights of the various components of the model until it can accurately predict the target values based on the input data.

5. Model Optimization and Tuning Phase

    a. Hyperparameter Tuning Documentation

To check the model performance on test and train data, we calculate the RMSE score. The RMSE score tells us how far off the predictions of the model are, on average, from the actual values.

Activity 1.1: Linear Regression Model

```
# training score
mse = mean_squared_error(y_train, predict_train)
rmse_lr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_train)
```

```
Root Mean Squared Error: 0.16226529653729893
```

```
# testing score
mse = mean_squared_error(y_test, predict_test)
rmse_lr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_test)
```

```
Root Mean Squared Error: 0.16116562949494187
```

Activity 1.2: Decision Tree Regressor Model

```
    # training score
    mse = mean_squared_error(y_train, predict_train_dtr)
    rmse_dtr_train = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_dtr_train)
```

```
Root Mean Squared Error: 0.13187559206436333
```

```
    # testing score
    mse = mean_squared_error(y_test, predict_test_dtr)
    rmse_dtr_test = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_dtr_test)
```

```
Root Mean Squared Error: 0.12918875831022705
```

Activity 1.3: Random Forest Regressor Model

```
    # training score
    mse = mean_squared_error(y_train, predict_train_rfr)
    rmse_rfr_train = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_rfr_train)
```

```
Root Mean Squared Error: 0.13066329578222882
```

```
# testing score
mse = mean_squared_error(y_test, predict_test_rfr)
rmse_rfr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_rfr_test)
```

```
Root Mean Squared Error: 0.12721255996349562
```

Activity 1.4: Gradient

Boosting Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_gbr)
rmse_gbr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_train)
```

```
Root Mean Squared Error: 0.14244277376076936
```

```
# testing score
mse = mean_squared_error(y_test, predict_test_gbr)
rmse_gbr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_test)
```

```
Root Mean Squared Error: 0.1394815884261522
```

Activity 1.5: Extreme

Gradient Boost Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_xgb)
rmse_xgb_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_xgb_train)
```

```
Root Mean Squared Error: 0.057574476336590054
```

```
    # testing score
    mse = mean_squared_error(y_test, predict_test_xgb)
    rmse_xgb_test = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_xgb_test)
```

```
Root Mean Squared Error: 0.12343275775750703
```

Activity 1.6: Bagging Regressor Model

```
    # Evaluate performance
    train_rmse_b = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse_b = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("Bagging Regressor:")
    print(f"Training RMSE: {train_rmse_b}")
    print(f"Testing RMSE: {test_rmse_b}")
```

```
Bagging Regressor:
Training RMSE: 0.11003378916688866
Testing RMSE: 0.11605208654607595
```

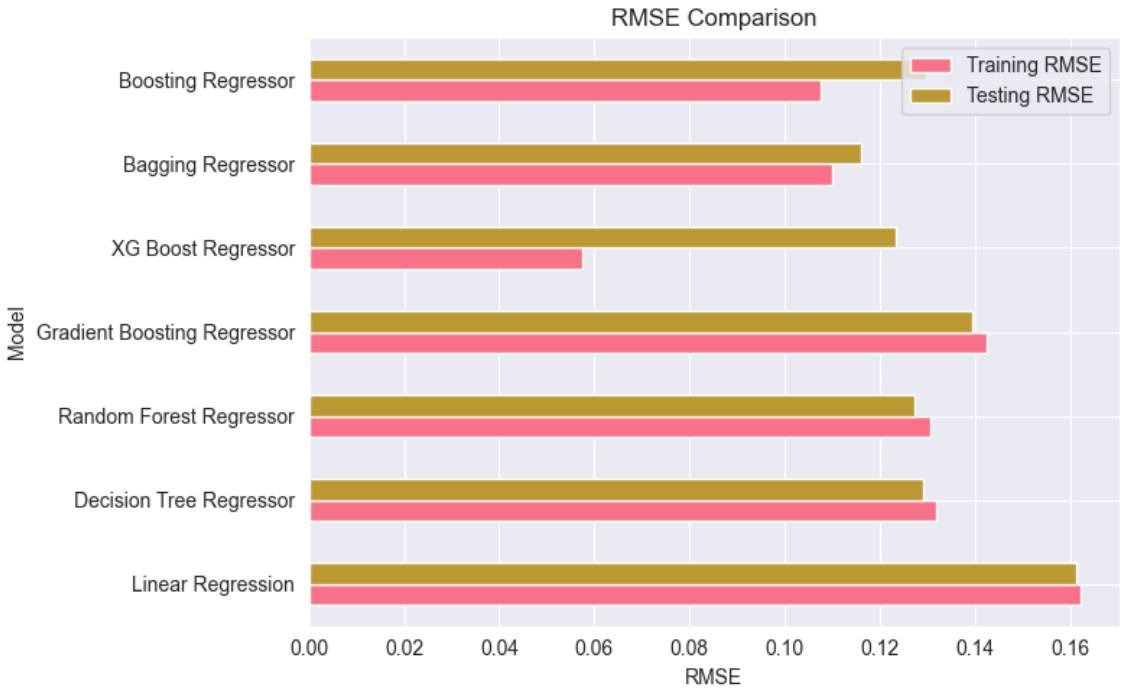
Activity 1.7: Boosting Regressor Model

```
    # Evaluate performance
    train_rmse_bo = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse_bo = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("AdaBoost Regressor:")
    print(f"Training RMSE: {train_rmse_bo}")
    print(f"Testing RMSE: {test_rmse_bo}")
```

```
AdaBoost Regressor:
Training RMSE: 0.1075964215336082
Testing RMSE: 0.1297762247652773
```

b. Performance Metrics Comparison Report

This code creates a Pandas Data Frame named "results" that contains the model names and root mean squared errors (RMSE) for both the training and testing data for each of the seven regression models: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG Boost Regressor, Bagging Regressor, and Boosting Regressor.



To determine which model is best, we should look for the model with the lowest RMSE value on the test data. This suggests that the model predicts value closer to the actual values. In this out of the seven models selected Boosting Regressor satisfies the conditions and hence selected.

c. Final Model Selection Justification

```
# dumping the selected model
pickle.dump(boosting_reg,open('productivity.pkl','wb'))
```

This code uses the "pickle" library in Python to save the trained Boosting Regressor model named "boosting_reg" as a file named "productivity.pkl". The "dump" method from the pickle library is used to save the model object in a serialized form that can be used again later. The "wb" parameter indicates that the file should be opened in binary mode to write data to it.

In this section, we will be building a web application that would help us integrate the machine learning model we have built and trained.

A user interface is provided for the users to enter the values for predictions. The entered values are fed into the saved model, and the prediction is displayed on the UI.

The section has following task:

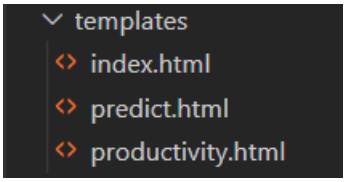
- Building HTML pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project we create three html files:

- index.html
- predict.html
- productivity.html

and save these html files in the templates folder.



Activity 2.2: Build Python code:

Importing the libraries

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

This code first loads the saved Boosting Regressor model from the "productivity.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
model1 = pickle.load(open('productivity.pkl', 'rb'))
app=Flask(__name__)
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.
The "render_template()" method is used to render an HTML template named "index.html". The “index.html” is the home page.

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

The route in this case is "/predict". When a user accesses the "/predict" route of the website, this function “index()” is called.
The "render_template()" method is used to render an HTML template named "predict.html".

```
@app.route('/predict') # rendering the html template
def index() :
    return render_template('predict.html')
```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/data_predict", and the method is set to GET and POST.The function "predict()" is then associated with this route.
The input data is collected from an HTML form and includes information such as the quarter, department, day of the week, team number, time allocated, unfinished items, overtime, incentive, idle time, idle men, style change, and number of workers.
The code converts some of the input data into a format that can be used by the machine learning model. For example, the department input is converted from a string to a binary value (1 for sewing, 0 for finishing), and the day of the week input is converted to a numerical value (0-6).
The model is then used to make a prediction based on the input data. The

prediction is rounded to 4 decimal places and multiplied by 100 to convert it to a percentage.
The prediction value is passed to the HTML template 'productivity.html' where it is displayed as a text message. The message informs the user of the predicted productivity based on the input data.

```
@app.route('/data_predict', methods=['GET','POST'])
def predict() :

    quarter = int(request.form['Quarter'])

    department = request.form['Department']
    if department == 'Sewing':
        department = 1
    if department == 'sewing':
        department = 1
    if department == 'Finishing':
        department = 0
    if department == 'finishing':
        department = 0

    day = request.form['Day of the week']
    if day == 'Monday':
        day = 0
    if day == 'Tuesday':
        day = 4
    if day == 'Wednesday':
        day = 5
    if day == 'Thursday':
        day = 3
    if day == 'Saturday':
        day = 1
    if day == 'Sunday':
        day = 2
```

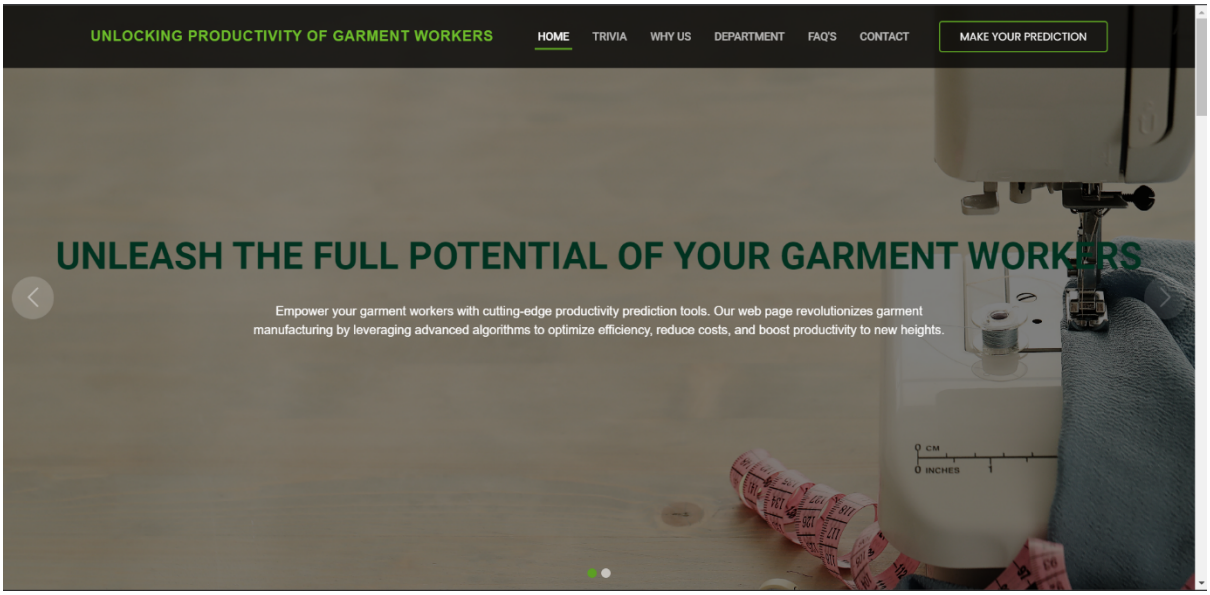
```
team = int(request.form['Team Number'])
time = int(request.form['Time Allocated'])
items = int(request.form['Unfinished Items'])
over_time = int(request.form['Over time'])
incentive = int(request.form['Incentive'])
idle_time = int(request.form['Idle Time'])
idle_men = int(request.form['Idle Men'])
style = int(request.form['Style Change'])
workers = int(request.form['Number of Workers'])

prediction = model1.predict(pd.DataFrame([[quarter,department,day,team,time,items,over_time,incentive,idle_time,idle_men,style,workers]], co
   'over_time', 'incentive','idle_time','idle_men','style_change', 'no_of_workers']))


prediction = (np.round(prediction,4))*100


return render_template('productivity.html', prediction_text ="is {}".format(prediction))
```

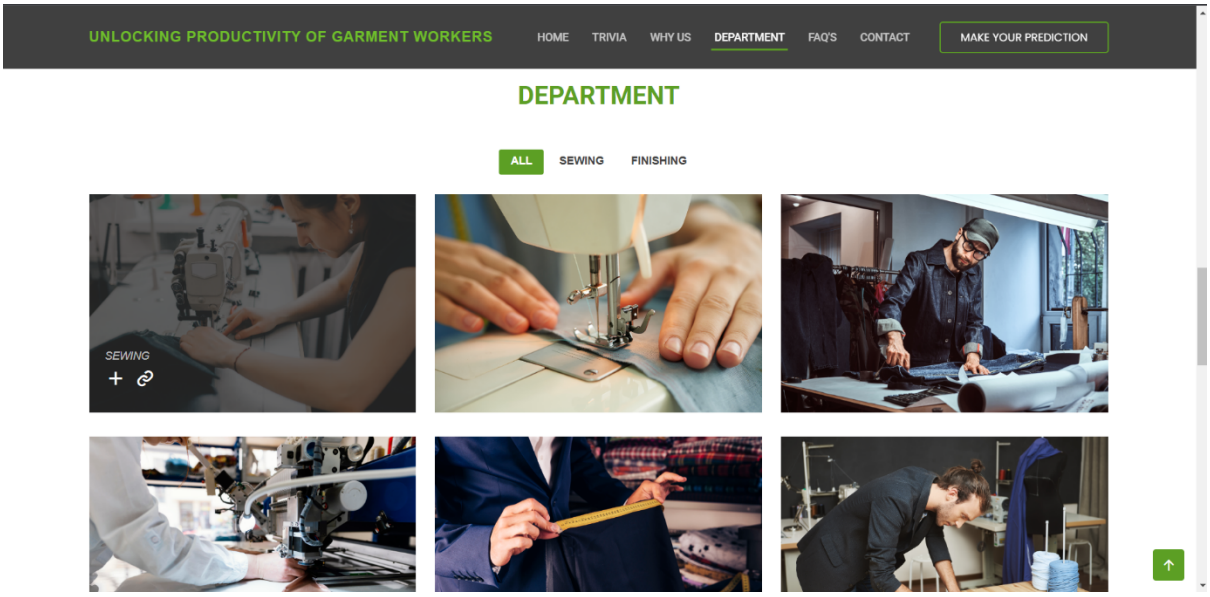
Main Function:
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.

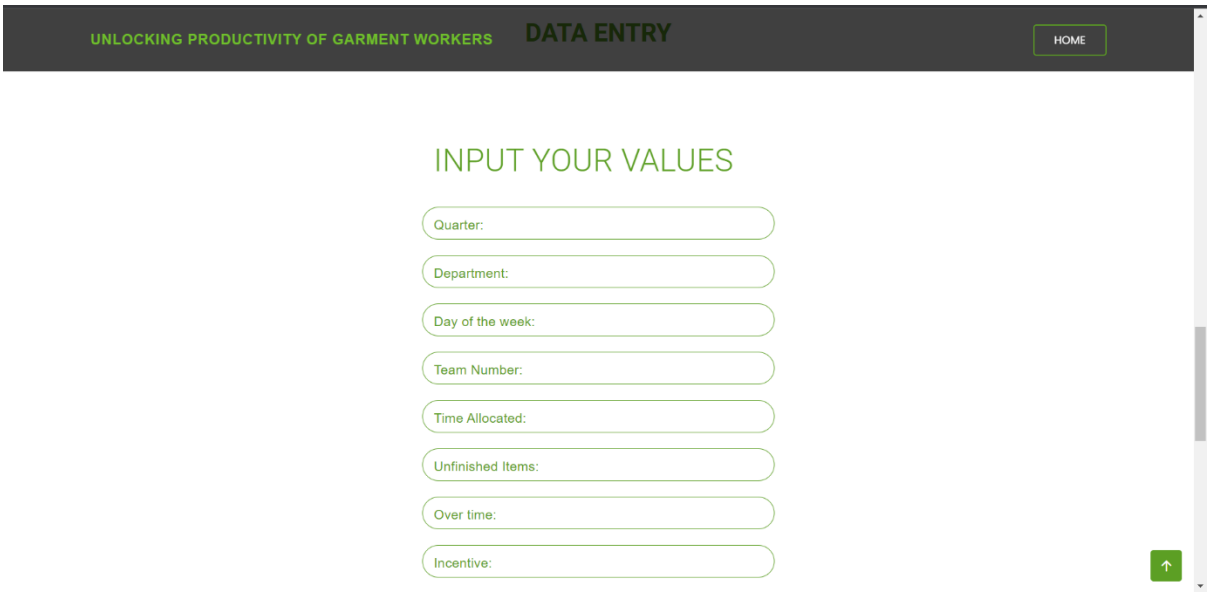
```
if __name__ == '__main__':
    app.run()
```

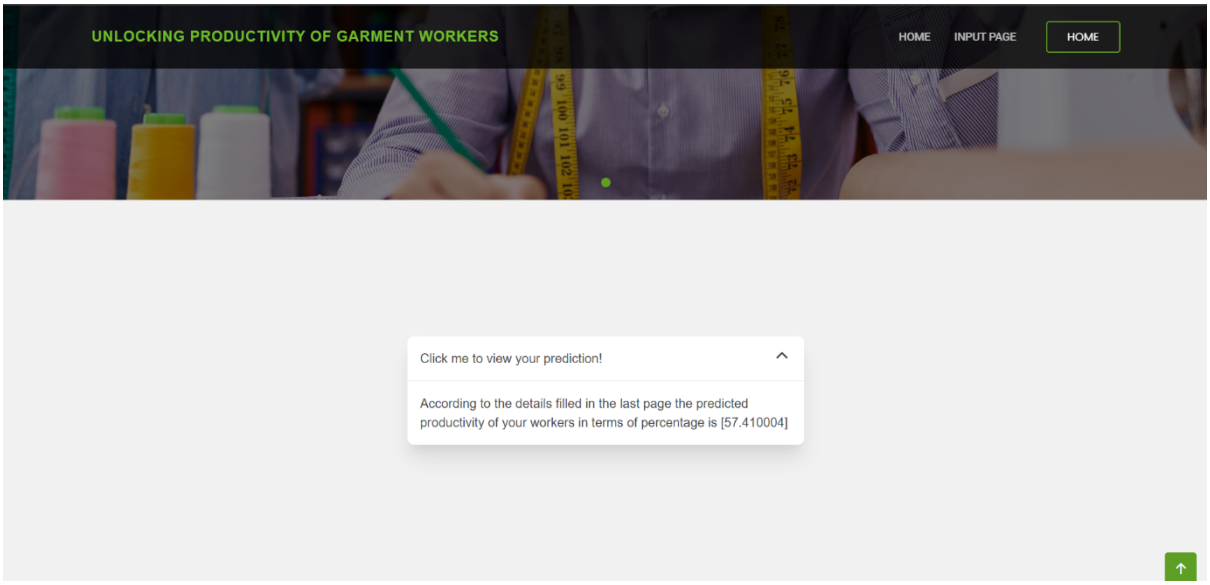
Activity 2.3: Run the web application

When you run the “main.py” file this window will open in the output terminal. Copy thehttp://127.0.0.1:5000and paste this link in your browser.


UNLOCKING PRODUCTIVITY OF GARMENT WORKERS
HOME
TRIVIA
WHY US
DEPARTMENT
FAQ'S
CONTACT
MAKE YOUR PREDICTION
UNLEASH THE FULL POTENTIAL OF YOUR GARMENT WORKERS
Empower your garment workers with cutting-edge productivity prediction tools. Our web page revolutionizes garment manufacturing by leveraging advanced algorithms to optimize efficiency, reduce costs, and boost productivity to new heights.


UNLOCKING PRODUCTIVITY OF GARMENT WORKERS
HOME
TRIVIA
WHY US
DEPARTMENT
FAQ'S
CONTACT
MAKE YOUR PREDICTION
DEPARTMENT
ALL
SEWING
FINISHING
SEWING





6. Results
7. Advantage and disadvantages

Advantages

1. **Improved Efficiency:**

    a. **Resource Allocation:** Predictive models can help managers allocate resources more effectively, ensuring that skilled workers are assigned to tasks where they can be most productive.
    b. **Time Management:** Anticipating productivity levels can aid in better scheduling, reducing downtime, and ensuring deadlines are met.

2. **Enhanced Quality Control:**
    a. **Consistent Output:** By understanding productivity patterns, companies can maintain consistent quality in their output.
    b. **Early Detection of Issues:** Predictive analytics can flag potential issues early, allowing for timely interventions to maintain quality.

3. **Cost Reduction:**
    a. **Labor Costs:** Optimizing worker productivity can lead to significant savings in labor costs.
    b. **Inventory Management:** Accurate productivity predictions help in better inventory management, reducing waste and excess stock.

4. **Strategic Planning:**
    a. **Capacity Planning:** Companies can plan for future capacity needs more accurately, ensuring they are neither overstaffed nor understaffed.
    b. **Investment Decisions:** Data-driven insights into productivity can inform investment in new technologies or training programs.

5. **Employee Development:**
    a. **Targeted Training:** Identifying areas where workers are less productive can help in designing targeted training programs to improve skills.

b. **Performance Feedback:** Workers receive more accurate feedback on their performance, aiding in personal development and motivation.

Disadvantages

1. **Data Dependency:**
   a. **Quality of Data:** Predictive accuracy is heavily dependent on the quality and quantity of data available. Poor data can lead to inaccurate predictions.
   b. **Data Privacy:** Collecting and storing detailed productivity data raises concerns about worker privacy and data security.

2. **Resistance to Change:**
   a. **Worker Opposition:** Employees might resist productivity monitoring and prediction, fearing increased surveillance or job insecurity.
   b. **Management Resistance:** Some managers might be skeptical of relying on predictive models, preferring traditional methods.

3. **Implementation Costs:**
   a. **Initial Investment:** Developing and implementing predictive models can be costly, requiring investment in technology and expertise.
   b. **Maintenance Costs:** Continuous updating and maintenance of predictive systems are necessary to ensure ongoing accuracy and relevance.

4. **Potential for Misuse:**
   a. **Overemphasis on Metrics:** There is a risk of focusing too much on productivity metrics at the expense of worker well-being and job satisfaction.
   b. **Unrealistic Expectations:** Predictive models might set unrealistic productivity targets, leading to increased pressure

and potential burnout among workers.

5. **Complexity:**
    a. **Model Complexity:** Developing accurate predictive models can be complex and require specialized knowledge in data science and machine learning.
    b. **Dynamic Factors:** Productivity can be influenced by numerous dynamic factors such as worker health, external economic conditions, and seasonal variations, making predictions challenging.

8. Conclusion

Predicting garment worker productivity presents a promising approach to enhancing efficiency, quality, and cost management within the industry. The advantages include better resource allocation, improved quality control, and strategic planning, all of which contribute to more effective operations and informed decision-making. However, these benefits are tempered by challenges such as the need for high-quality data, potential resistance from workers and management, and the costs associated with implementing and maintaining predictive systems. Additionally, there is a risk of overemphasis on productivity metrics at the expense of worker well-being. To harness the full potential of productivity prediction, companies must address these challenges thoughtfully, ensuring a balanced approach that considers both technological advancements and the human factors involved.

9. Future Scope

Advanced Analytics and AI:

- **Machine Learning Enhancements:** Utilizing more sophisticated machine learning algorithms and deep learning techniques can improve the accuracy and reliability of productivity predictions.

- **Real-Time Analytics:** Implementing real-time data analytics can provide immediate insights, allowing for quicker adjustments and interventions.

  Integration with IoT:

- **Smart Wearables:** Integrating smart wearables to monitor worker movements and health can provide more granular data, enhancing predictive accuracy.
- **IoT-Enabled Machinery:** Using IoT-enabled machinery to collect detailed performance data can lead to a more comprehensive understanding of factors affecting productivity.

  Customized Worker Training:

- **Personalized Training Programs:** Predictive models can identify specific areas where individual workers need improvement, leading to more effective, personalized training programs.
- **Gamification:** Introducing gamification elements to training based on predictive insights can enhance engagement and motivation among workers.

  Enhanced Worker Well-being:

- **Health and Safety Monitoring:** Predictive models can incorporate data related to worker health and safety, ensuring that productivity improvements do not come at the cost of well-being.
- **Work-Life Balance:** Insights from productivity data can help in designing schedules that balance productivity with work-life balance, reducing burnout.

  Sustainability and Ethical Practices:

- **Sustainable Practices:** Predictive models can help in optimizing resource use, reducing waste, and promoting sustainable practices within the garment industry.
- **Ethical Labor Practices:** Ensuring that productivity improvements align with fair labor practices can enhance the industry's reputation and worker satisfaction.
-

10. Appendix
10.1 Source Code
10.2. GitHub & Project Demo Lin

FINAL REPORT

GARMENT WORKER PRODUCTIVITY

PREDICTION

TEAM:

NIKHIL PULAGAM

MULE VIGNESH

SHERYAS

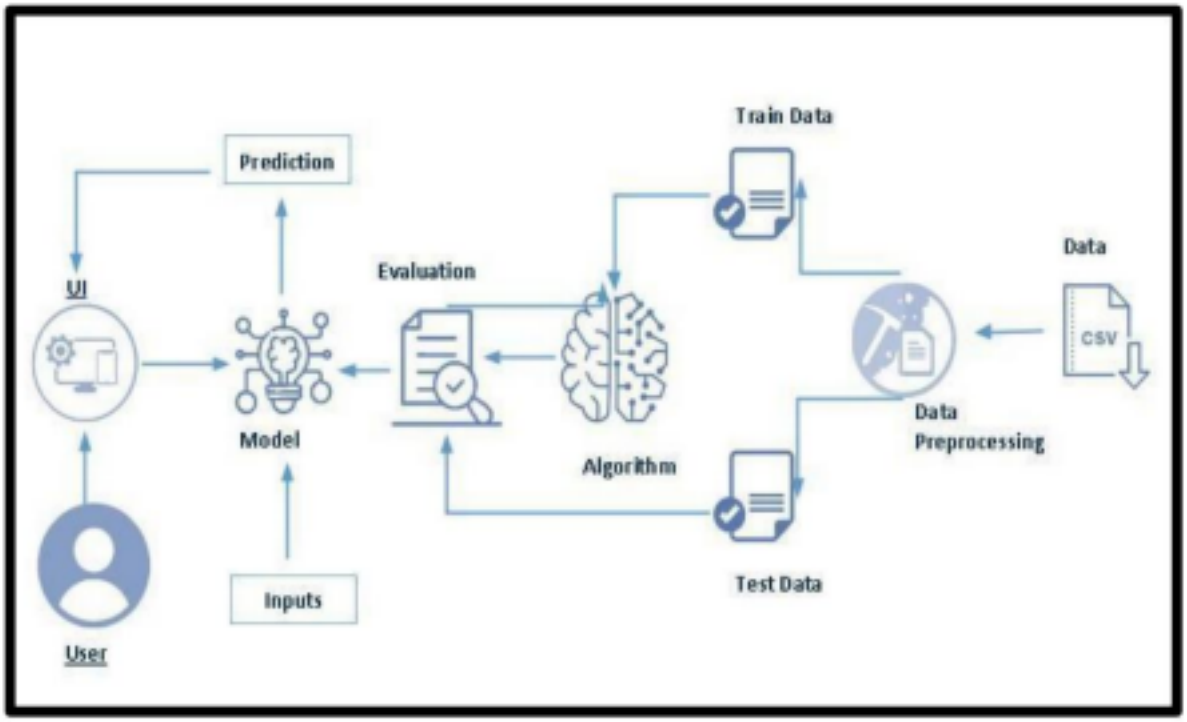
1. Introduction

    a. Project overviews:

    The garment industry is one of the largest industries in the world, and garment worker productivity is a crucial factor in determining the success and profitability of a company. In this project, we aim to develop a machine learning model that predicts the productivity of garment workers based on a given set of features. Our dataset contains information on various attributes of garment production, including the quarter, department, day, team number, time allocated, unfinished items, over time, incentive, idle time, idle men, style change, number of workers, and actual productivity. We will use this dataset to train and evaluate our predictive model.

The development of an accurate garment worker productivity prediction model using machine learning can have significant implications in various domains, including manufacturing, human resources, and supply chain management. This model can help companies identify the factors that affect worker productivity and take corrective actions to improve efficiency, reduce costs, and enhance their competitiveness in the market.

Technical Architecture:



b. Objectives:

Define Problem / Problem Understanding

- Specify the business proble,

Business requirements
Literature Survey
Social or Business Impact

Data Collection & Preparation

- Collect the dataset
- Data Preparation

Exploratory Data Analysis

- Descriptive statistical
- Visual Analysis

Collect the dataset

- Training the model in multiple algorithms
- Testing the model

Performance Testing & Hyperparameter Tuning

- Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning

Model Deployment

- Save the best model
- Integrate with Web Framework

Project Demonstration & Documentation

- Record explanation Video for project end to end solution
- Project Documentation-Step by step project development procedure

2. Project Initialization and Planning Phase

    a. Define Problem Statement

    To ensure that the garment worker productivity prediction model meets business requirements and can be deployed for public use, it should follow the following rules and requirements: Accuracy: The model should have a high level of accuracy in predicting worker productivity, with a low margin of error. This is crucial to ensure that the predictions are reliable and trustworthy.

    - Privacy and security: The model should be developed in accordance with privacy and security regulations to protect user data. This includes ensuring that sensitive data is stored securely and implementing proper data access controls.
    - Interpretability: The model should be interpretable, meaning that the predictions can be explained and understood by the end-users. This is important to build trust in the model and to allow users to make informed decisions based on the predictions.
    - User interface: The model should have a user-friendly interface that is easy to use and understand. This is important to ensure that the model can be deployed for public use, even by

individuals who may not have technical expertise.

2.2Project Proposal

A literature survey for a garment worker productivity prediction project would involve researching and reviewing existing studies, articles, and other publications related to machine learning in the field of manufacturing and workforce management. The survey would aim to gather information on current methods for predicting productivity in the garment industry, including feature selection, data pre-processing, and machine learning algorithms used for prediction.

The survey would also examine any gaps in knowledge and research opportunities in the field of garment worker productivity prediction, including the use of new machine learning techniques, such as reinforcement learning, and the integration of other data sources, such as wearable technology and environmental sensors.

a. Initial Project Planning

Social Impact: The garment worker productivity prediction model has the potential to improve the lives of garment workers by promoting fair and efficient workforce management practices. The model can identify factors that affect worker productivity and suggest ways to improve working conditions, incentive schemes, and training programs. This can lead to higher job satisfaction, better pay, and improved working conditions for garment workers.
Business Impact: The garment worker productivity prediction model can have significant implications for the garment manufacturing industry. The model can help manufacturers optimize their workforce management strategies, reduce idle time, and increase productivity. This can lead to higher profits, lower production costs, and improved quality of goods produced. Additionally, the model can assist in identifying areas for process improvement, such as reducing rework and improving supply

chain efficiency.

3. Data Collection and Preprocessing Phase

   a. Data Collection Plan and Raw Data Sources Identified

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

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset. Download the dataset from the above link. Let us understand and analyze the dataset properly using visualization techniques.

Note: There are several approaches for understanding the data. But we have applied some of it here. You can also employ a variety of techniques.

Activity 1.1: Importing the libraries

Import the libraries required for this machine learning project, as shown in the image below.

```
# Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
```

Activity 1.2: Read the Dataset

Our dataset format could be in .csv, excel files,.txt, .json, and so on. With the help of pandas, we can read the dataset.
Since our dataset is a csv file, we use read_csv() which is pandas function to read the dataset. As a parameter we have to give the directory of the csv file.
df.head() will display first 5 rows of the dataset.

```
df = pd.read_csv('garments_worker_productivity.csv')
```

```
df.head()
```

Python

quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	59.0	0.940725
Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	8.0	0.886500
Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	56.0	0.800382

b. Data Exploration and Preprocessing

Data preparation, also known as data preprocessing, is the process of cleaning, transforming, and organizing raw data before it can be used in a data analysis or machine learning model.

The activity include following steps:

- removing missing values
- handling outliers
- encoding categorical variables
- normalizing data

Note: These are general steps to take in pre-processing before feeding data to machine learning for training. The pre-processing steps differ depending on the dataset. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling Missing Values

Let's first figure out what kind of data is in our columns by using df.info(). We may deduce from this that our columns include data of the types "object", "float64" and “int64”.

```
df.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
```

```
df.isna().sum()
```

```
quarter                0
department             0
day                    0
team_number            0
time_allocated         0
unfinished_items     506
over_time              0
incentive              0
idle_time              0
idle_men               0
style_change           0
no_of_workers          0
actual_productivity    0
dtype: int64
```

This line of code is used to count the number of missing or null values in a pandas DataFrame. It returns a list of the total number of missing values in each column of the DataFrame.

```
df['unfinished_items'] = df['unfinished_items'].fillna(df['unfinished_items'].mean())
```

This line of code fills the missing values in the “unfinished_items” column with the column mean.

Activity 2.2: Handling Independent Columns

```
df.drop(['date', 'targeted_productivity'], axis=1, inplace=True)
df.head()
```

	quarter	department	day	team	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
0	Quarter1	sweing	Thursday	8	26.16	1108.0	7080	98	0.0	0	0	59.0	0.940725
1	Quarter1	finishing	Thursday	1	3.94	NaN	960	0	0.0	0	0	8.0	0.886500
2	Quarter1	sweing	Thursday	11	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
3	Quarter1	sweing	Thursday	12	11.41	968.0	3660	50	0.0	0	0	30.5	0.800570
4	Quarter1	sweing	Thursday	6	25.90	1170.0	1920	50	0.0	0	0	56.0	0.800382

+ Code + Markdown

This line of code drops the columns date and targeted_productivity from the dataframe.

```
df = df.rename(columns={
    'team' : 'team_number',
    'smv' : 'time_allocated',
    'wip' : 'unfinished_items',
    'no_of_style_change' : 'style_change'
})
df
```

The rename() method is being used to change the names of some columns. The method takes a dictionary as its argument, where the keys are the original column names and the values are the new names.

```
df['quarter'].unique()
```

```
array(['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4', 'Quarter5'],
      dtype=object)
```

```
df['quarter'] = df['quarter'].str.replace('Quarter5','Quarter1')
```

The first line of code selects a specific column named "quarter" from a table of data. The unique()method is then used to list all the different values in that column.
The second line of code changes some of the values in the "quarter" column. Specifically, it replaces any instances of the text "Quarter5" with the text "Quarter1" using the str.replace()method.

```
# Extract the numerical part from the 'quarter' column
df['quarter'] = df['quarter'].str.extract(r'(\d+)')
df['quarter']
```

The first line of code is using the str.extract() method on the "quarter"

column to extract the numerical part of each value. It uses a regular expression pattern r'(\d+)' to match any sequence of one or more digits in each value.
The second line of code is assigning the modified "quarter" column back to the same "quarter" column, effectively replacing the original column with the modified one.

```
df['department'] = df['department'].str.replace('sweing','sewing')
df['department'] = df['department'].str.replace('finishing ','finishing')
```

The first line of code is using the str.replace() method on the "department" column to replace any instances of the misspelled word "sweing" with the correct spelling "sewing".
The second line of code is also using the str.replace() method on the "department" column to replace any instances of the word "finishing " (with a space at the end) with the word "finishing" (without a space at the end). This will remove the extra space and standardize the spelling of the word.

```
df['team_number'] = df['team_number'].astype(int)
df['over_time'] = df['over_time'].astype(int)
df['incentive'] = df['incentive'].astype(int)
df['idle_time'] = df['idle_time'].astype(int)
df['idle_men'] = df['idle_men'].astype(int)
df['style_change'] = df['style_change'].astype(int)
```

```
df['time_allocated'] = df['time_allocated'].astype(int)
```

```
df['unfinished_items'] = df['unfinished_items'].astype(int)
```

```
df['idle_time'] = df['idle_time'].astype(int)
```

```
df['no_of_workers'] = df['no_of_workers'].astype(int)
```

Each line of code is using the astype() method to convert the data type

of a specific column to an integer type. The column name is specified on the left-hand side of the equation, and the new integer type is specified as an argument to the astype() method.
Activity 2.3: Handling Categorical Values

```
lc = LabelEncoder()
```

```
print('Before encoding: ', df['department'].unique())
df['department'] = lc.fit_transform(df['department'])
print('After encoding: ',df['department'].unique())
```

This code is encoding the values in a column named "department" by using a LabelEncoder() object to convert the original values into numerical encoded values. The original and encoded values are printed before and after the encoding is performed.

```
Before encoding:  ['sewing' 'finishing']
After encoding:  [1 0]
```

```
print('Before encoding: ', df['day'].unique())
df['day'] = lc.fit_transform(df['day'])
print('After encoding: ',df['day'].unique())
```

```
Before encoding:  ['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
After encoding:  [3 1 2 0 4 5]
```

This code is encoding the values in a column named "day" by using a LabelEncoder() object to convert the original values into numerical encoded values. The original and encoded values are printed before and

after the encoding is performed.

3.3 Data Exploration and Preprocessing

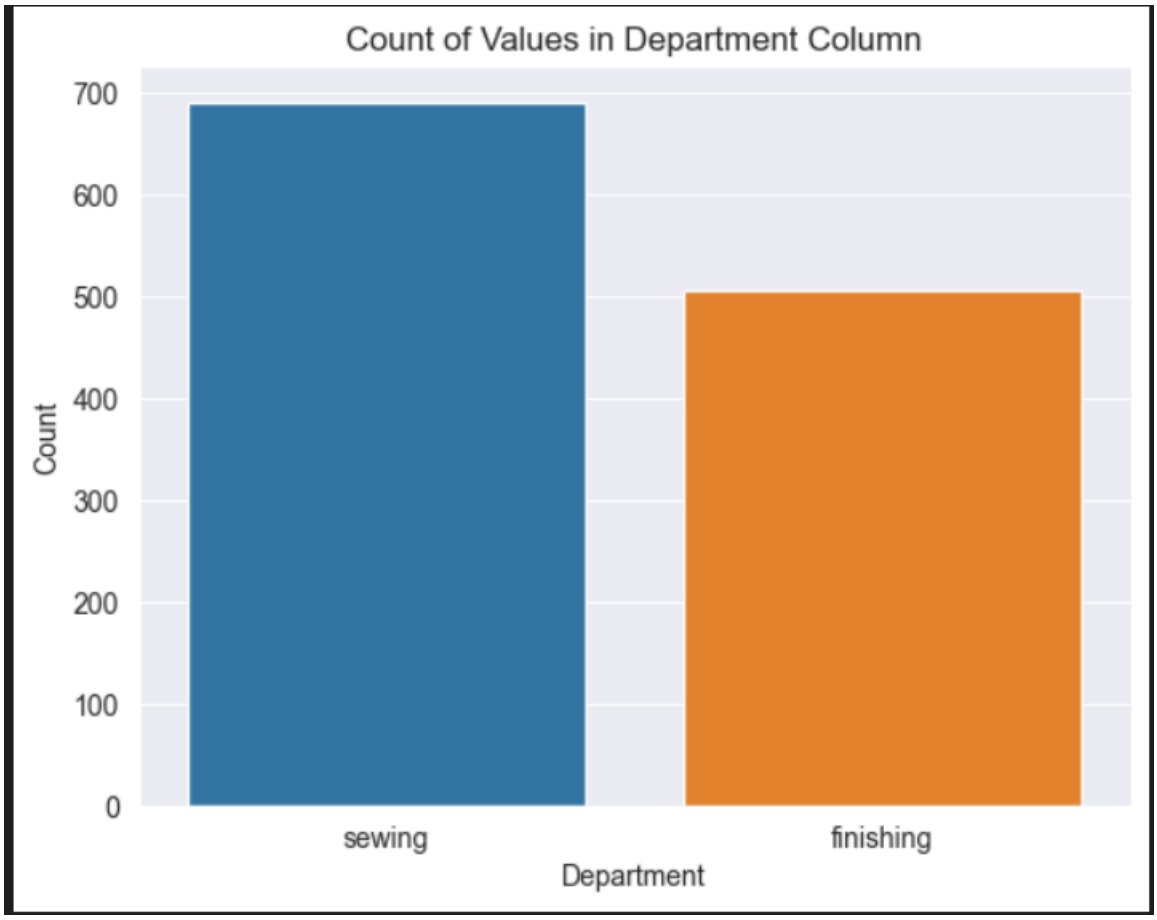
Visual analysis is the process of examining and understanding data via the use of visual representations such as charts, plots, and graphs. It is a method for quickly identifying patterns, trends, and outliers in data, which can aid in gaining insights and making sound decisions.

Activity 2.1: Univariate analysis

Univariate analysis is a statistical method used to analyse a single variable in a dataset. This analysis focuses on understanding the distribution, central tendency, and dispersion of a single variable.

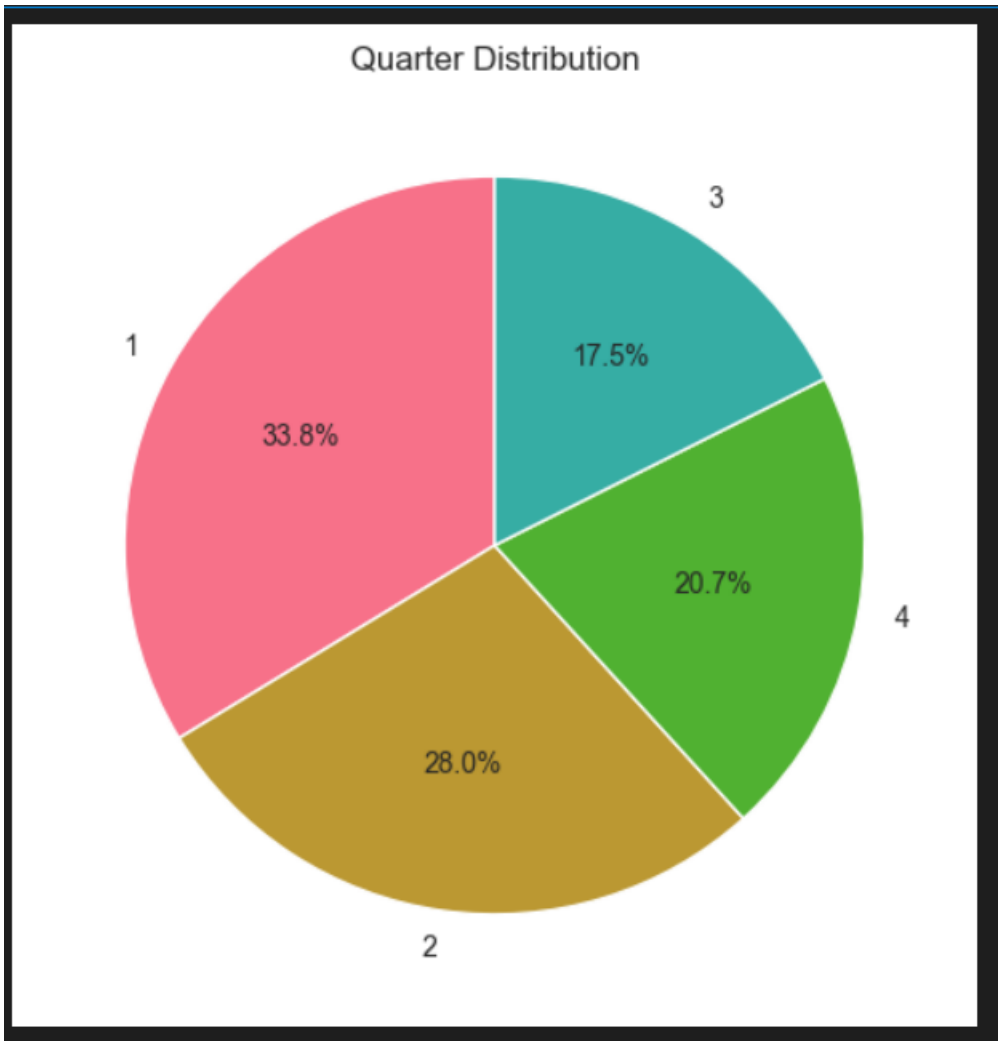
```
sns.countplot(data=df, x='department')
plt.xlabel('Department')
plt.ylabel('Count')
plt.title('Count of Values in Department Column')
plt.show()
```

This code creates a bar chart using the Seaborn library to show the number of occurrences of each unique value in the "department" column of a Pandas DataFrame. The x-axis represents the unique values of the "department" column, and the y-axis represents the number of times each unique value appears in the column. The plt.xlabel(), plt.ylabel(), and plt.title() functions are used to add labels and a title to the plot. Finally, plt.show() is used to display the plot.




```
quarter_counts = df['quarter'].value_counts()
plt.figure(figsize=(8, 6))
sns.set_palette("husl")  # Set custom color palette
plt.pie(quarter_counts, labels=quarter_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Quarter Distribution')
plt.show()
```

This code generates a pie chart using the Seaborn library to show the distribution of different quarters in a dataset. The value_counts() function counts the number of occurrences of each quarter in the dataset, and the resulting counts are used to create the pie chart. The pie chart shows the proportion of the total number of entries in the dataset that corresponds to each quarter. The title of the pie chart is "Quarter Distribution".



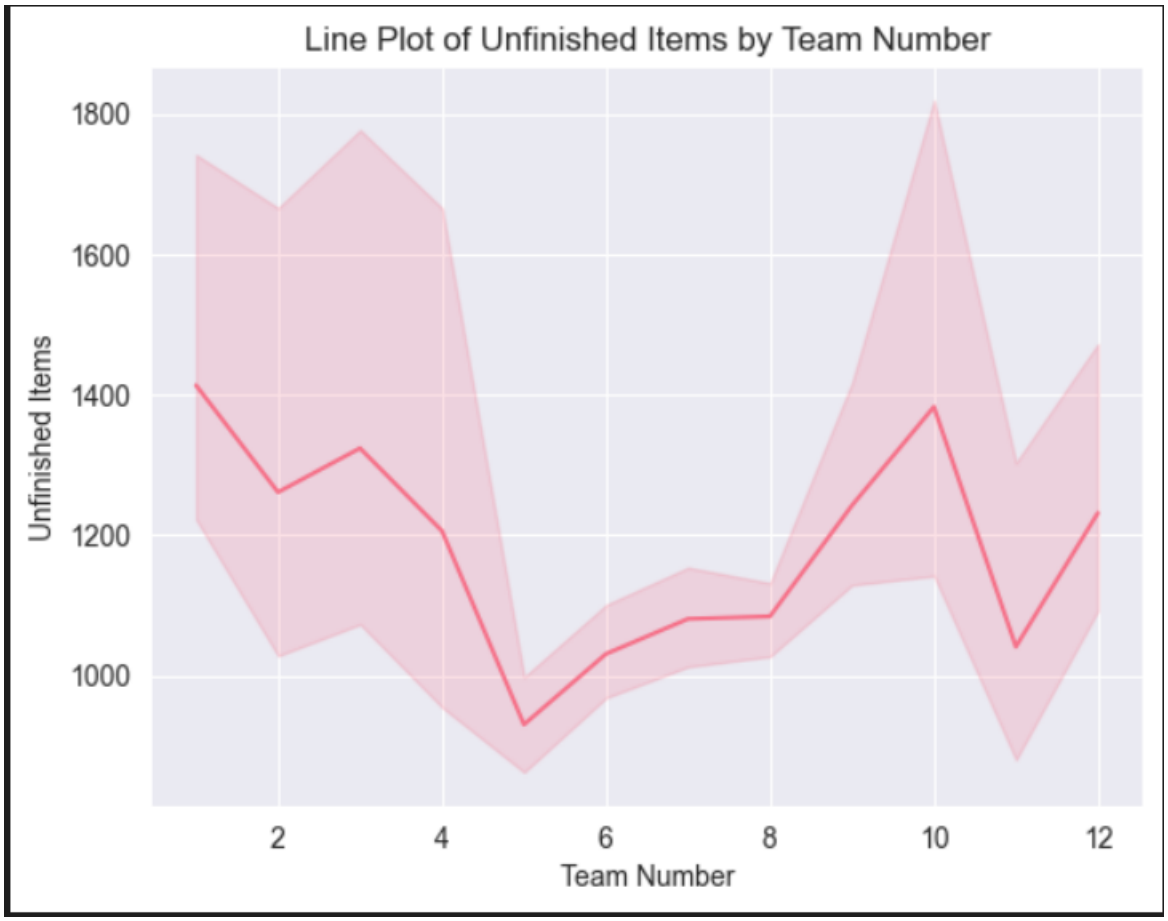
Activity 2.2: Bivariate analysis

Bivariate analysis is a statistical method used to analyse the relationship between two variables in a dataset. This analysis focuses on examining how changes in one variable are related to changes in another variable.

```
sns.lineplot(data=df, x='team_number', y='unfinished_items')
plt.xlabel('Team Number')
plt.ylabel('Unfinished Items')
plt.title('Line Plot of Unfinished Items by Team Number')
plt.show()
```

This code generates a line plot using the Seaborn library to show the relationship between team number and the number of unfinished items.

The line plot shows how the number of unfinished items varies across different teams. The x-axis represents the team number, and the y-axis represents the number of unfinished items. The title of the line plot is "Line Plot of Unfinished Items by Team Number".

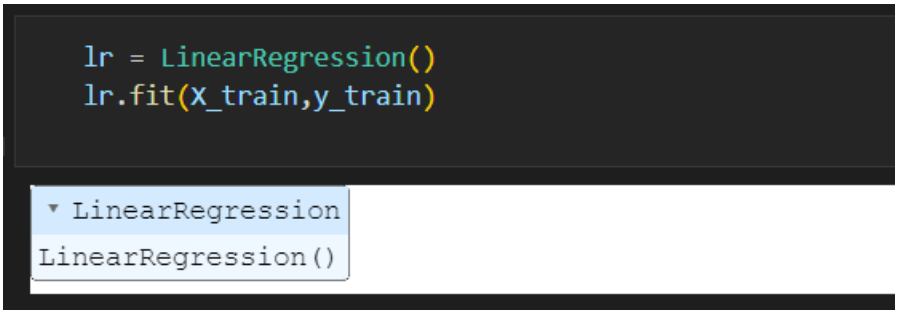


4. Model Development Phase

a. Feature Selection Report

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying seven regression algorithms. The best model is saved based on its performance.

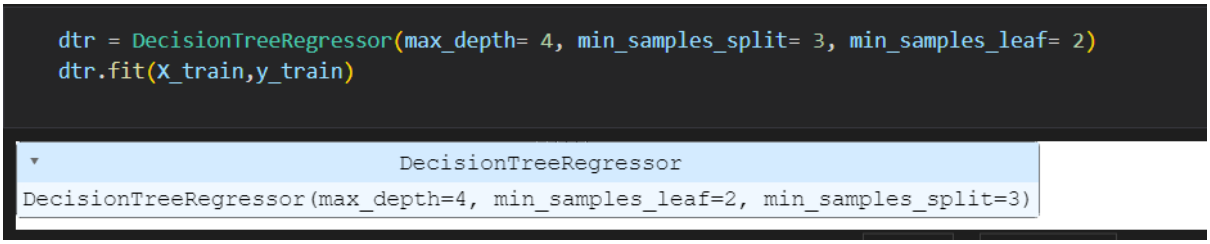
Activity 1.1: Linear Regression Model



This code performs linear regression modeling using the scikit-learn library.

It creates a LinearRegression object named "lr". The "fit" method is then called on the training data X_train and y_train. This method trains the model by finding the coefficients for the linear equation that best fits the data.

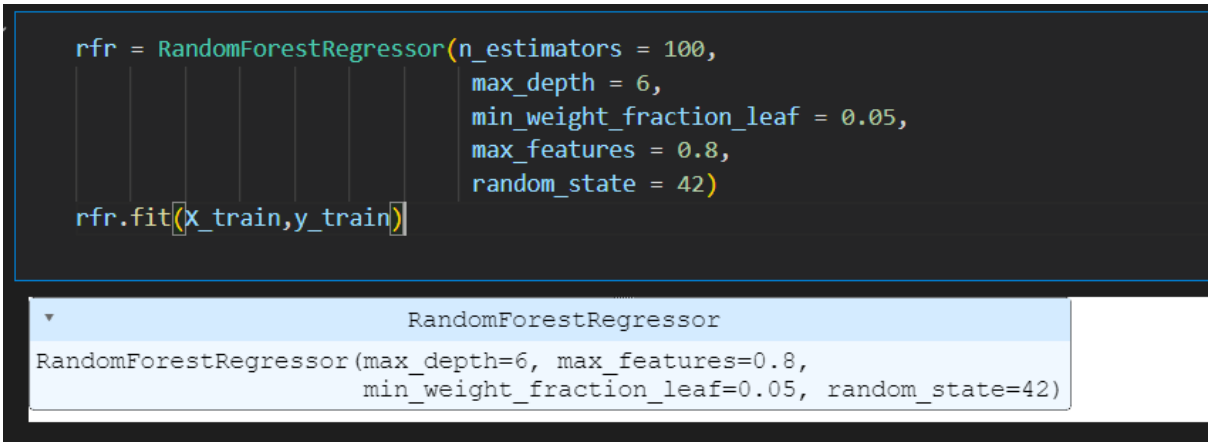
Activity 1.2: Decision Tree Regressor Model



This code is training a decision tree regression model using the training data (X_train and y_train) with the specified hyperparameters

(max_depth=4, min_samples_split=3, min_samples_leaf=2). The model is stored in the variable 'dtr'. After training, the model will be able to make predictions on new data based on the relationships learned from the training data.

Activity 1.3: Random Forest Regressor Model



The code creates an instance of the RandomForestRegressor class with certain hyperparameters set. The hyperparameters specify the number of trees to use in the random forest, the maximum depth of each tree, the minimum weight fraction required to be at a leaf node, the maximum number of features to consider when splitting a node, and a random state to ensure reproducibility of results. The rfr object is then trained on the training data X_train and y_train using the fit method.

The trained model can then be used to make predictions on new data.

b. Model Selection Report

Activity 1.4: Gradient Boosting Regressor Model

```
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=42)
gbr.fit(X_train,y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=1, random_state=42)
```

The code is fitting a Gradient Boosting Regressor model to the training data X_train and y_train using n_estimators equal to 100 (the number of trees in the forest), learning_rate equal to 0.1 (the step size shrinkage used to prevent overfitting), max_depth equal to 1 (the maximum depth of the individual regression estimators), and random_state equal to 42 (to ensure reproducibility of results).

The fitted model can then be used to make predictions on new data using the predict() method.

Activity 1.5: Extreme Gradient Boost Regressor Model

```
xgb = XGBRegressor(n_estimators=300, learning_rate=0.05,
                   max_leaves = 3, random_state = 1)
xgb.fit(X_train, y_train)
```

This code is training an XGBoost regression model. The XGBRegressor function is being used to create the model. The parameters passed to the function include the number of estimators, learning rate, maximum number of leaves, and random state. The model is trained on the training set (X_train and y_train) using the fit() method. Once trained, the model can be used to make predictions on new data.

4.3 Initial Model Training Code, Model Validation and Evaluation Report

Activity 1.6: Bagging Regressor Model

```
# Define base model
base_model = XGBRegressor(n_estimators=700, learning_rate=0.06, max_depth=2, max_leaves=3, random_state=1)

# Create bagging regressor
bagging_reg = BaggingRegressor(base_model, n_estimators=100, random_state=42)


# Fit bagging regressor
bagging_reg.fit(X_train, y_train)
```

This code creates a machine learning model using the XGBoost algorithm. The model is designed to predict a numeric value (the target variable) based on a set of input variables.

To improve the accuracy of the model, the Bagging technique is used. This involves creating multiple versions of the model, each with slightly different training data, and combining their predictions to create a final prediction.

Finally, the model is trained using a set of input data (X_train) and the corresponding target values (y_train). This involves adjusting the weights of the various components of the model until it can accurately predict the target values based on the input data.

Activity 1.7: Boosting Regressor Model

```
# Define base model
base_model = XGBRegressor(n_estimators=700, learning_rate=0.06, max_depth=2, max_leaves=3, random_state=1)

# Create AdaBoost regressor
boosting_reg = AdaBoostRegressor(base_model, n_estimators=100, learning_rate=0.1, random_state=42)

# Fit AdaBoost regressor
boosting_reg.fit(X_train, y_train)
```

This code creates a machine learning model to predict a numeric value based on a set of input variables using the XGBoost algorithm as the base model.

To improve the accuracy of the model, the AdaBoost technique is used. AdaBoost stands for Adaptive Boosting and works by creating multiple versions of the model, each with slightly different training data, and weighting the predictions of each model based on its accuracy.

Finally, the model is trained using a set of input data (X_train) and the corresponding target values (y_train). This involves adjusting the weights of the various components of the model until it can accurately predict the target values based on the input data.

5. Model Optimization and Tuning Phase

    a. Hyperparameter Tuning Documentation

To check the model performance on test and train data, we calculate the RMSE score. The RMSE score tells us how far off the predictions of the model are, on average, from the actual values.

Activity 1.1: Linear Regression Model

```
# training score
mse = mean_squared_error(y_train, predict_train)
rmse_lr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_train)
```

```
Root Mean Squared Error: 0.16226529653729893
```

```
# testing score
mse = mean_squared_error(y_test, predict_test)
rmse_lr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_test)
```

```
Root Mean Squared Error: 0.16116562949494187
```

Activity 1.2: Decision Tree Regressor Model

```
    # training score
    mse = mean_squared_error(y_train, predict_train_dtr)
    rmse_dtr_train = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_dtr_train)
```

```
Root Mean Squared Error: 0.13187559206436333
```

```
    # testing score
    mse = mean_squared_error(y_test, predict_test_dtr)
    rmse_dtr_test = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_dtr_test)
```

```
Root Mean Squared Error: 0.12918875831022705
```

Activity 1.3: Random Forest Regressor Model

```
    # training score
    mse = mean_squared_error(y_train, predict_train_rfr)
    rmse_rfr_train = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_rfr_train)
```

```
Root Mean Squared Error: 0.13066329578222882
```

```
# testing score
mse = mean_squared_error(y_test, predict_test_rfr)
rmse_rfr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_rfr_test)
```

```
Root Mean Squared Error: 0.12721255996349562
```

Activity 1.4: Gradient

Boosting Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_gbr)
rmse_gbr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_train)
```

```
Root Mean Squared Error: 0.14244277376076936
```

```
# testing score
mse = mean_squared_error(y_test, predict_test_gbr)
rmse_gbr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_test)
```

```
Root Mean Squared Error: 0.1394815884261522
```

Activity 1.5: Extreme

Gradient Boost Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_xgb)
rmse_xgb_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_xgb_train)
```

```
Root Mean Squared Error: 0.057574476336590054
```

```
    # testing score
    mse = mean_squared_error(y_test, predict_test_xgb)
    rmse_xgb_test = np.sqrt(mse)
    print('Root Mean Squared Error:', rmse_xgb_test)
```

```
Root Mean Squared Error: 0.12343275775750703
```

Activity 1.6: Bagging Regressor Model

```
    # Evaluate performance
    train_rmse_b = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse_b = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("Bagging Regressor:")
    print(f"Training RMSE: {train_rmse_b}")
    print(f"Testing RMSE: {test_rmse_b}")
```

```
Bagging Regressor:
Training RMSE: 0.11003378916688866
Testing RMSE: 0.11605208654607595
```

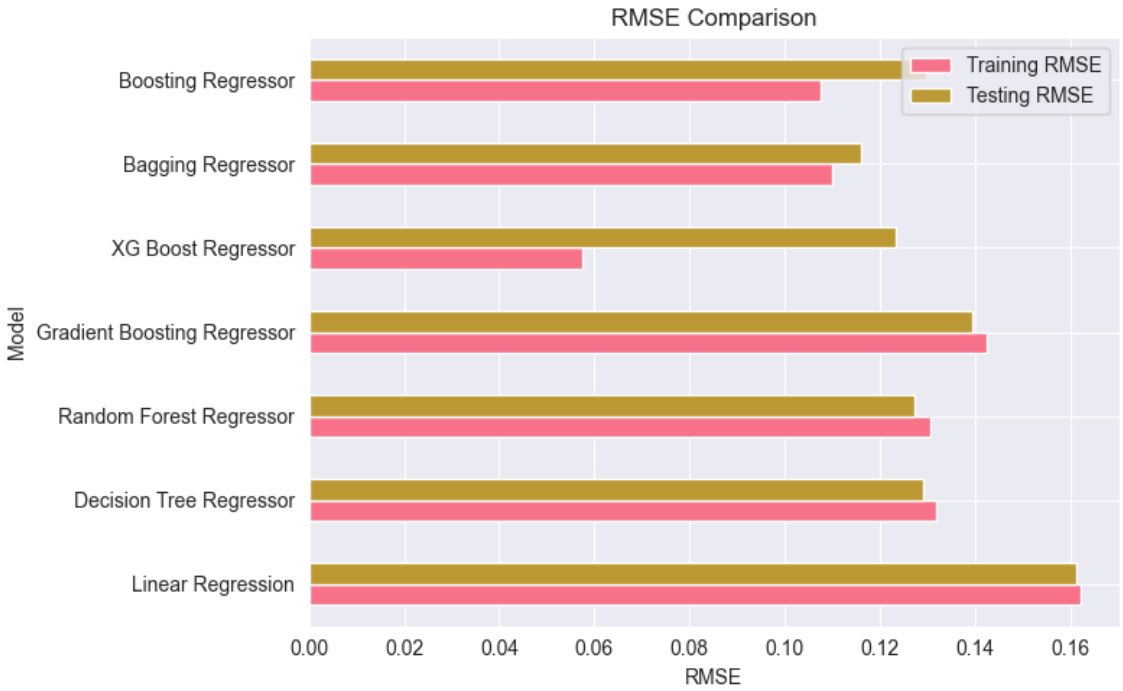
Activity 1.7: Boosting Regressor Model

```
    # Evaluate performance
    train_rmse_bo = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse_bo = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("AdaBoost Regressor:")
    print(f"Training RMSE: {train_rmse_bo}")
    print(f"Testing RMSE: {test_rmse_bo}")
```

```
AdaBoost Regressor:
Training RMSE: 0.1075964215336082
Testing RMSE: 0.1297762247652773
```

b. Performance Metrics Comparison Report

This code creates a Pandas Data Frame named "results" that contains the model names and root mean squared errors (RMSE) for both the training and testing data for each of the seven regression models: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG Boost Regressor, Bagging Regressor, and Boosting Regressor.



To determine which model is best, we should look for the model with the lowest RMSE value on the test data. This suggests that the model predicts value closer to the actual values. In this out of the seven models selected Boosting Regressor satisfies the conditions and hence selected.

c. Final Model Selection Justification

```
# dumping the selected model
pickle.dump(boosting_reg,open('productivity.pkl','wb'))
```

This code uses the "pickle" library in Python to save the trained Boosting Regressor model named "boosting_reg" as a file named "productivity.pkl". The "dump" method from the pickle library is used to save the model object in a serialized form that can be used again later. The "wb" parameter indicates that the file should be opened in binary mode to write data to it.

In this section, we will be building a web application that would help us integrate the machine learning model we have built and trained.

A user interface is provided for the users to enter the values for predictions. The entered values are fed into the saved model, and the prediction is displayed on the UI.

The section has following task:

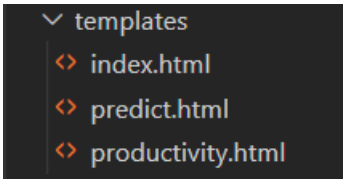
- Building HTML pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project we create three html files:

- index.html
- predict.html
- productivity.html

and save these html files in the templates folder.



Activity 2.2: Build Python code:

Importing the libraries

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

This code first loads the saved Boosting Regressor model from the "productivity.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
model1 = pickle.load(open('productivity.pkl', 'rb'))
app=Flask(__name__)
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.
The "render_template()" method is used to render an HTML template named "index.html". The “index.html” is the home page.

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

The route in this case is "/predict". When a user accesses the "/predict" route of the website, this function “index()” is called.
The "render_template()" method is used to render an HTML template named "predict.html".

```
@app.route('/predict') # rendering the html template
def index() :
    return render_template('predict.html')
```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/data_predict", and the method is set to GET and POST.The function "predict()" is then associated with this route.
The input data is collected from an HTML form and includes information such as the quarter, department, day of the week, team number, time allocated, unfinished items, overtime, incentive, idle time, idle men, style change, and number of workers.
The code converts some of the input data into a format that can be used by the machine learning model. For example, the department input is converted from a string to a binary value (1 for sewing, 0 for finishing), and the day of the week input is converted to a numerical value (0-6).
The model is then used to make a prediction based on the input data. The

prediction is rounded to 4 decimal places and multiplied by 100 to convert it to a percentage.
The prediction value is passed to the HTML template 'productivity.html' where it is displayed as a text message. The message informs the user of the predicted productivity based on the input data.

```
@app.route('/data_predict', methods=['GET','POST'])
def predict() :

    quarter = int(request.form['Quarter'])

    department = request.form['Department']
    if department == 'Sewing':
        department = 1
    if department == 'sewing':
        department = 1
    if department == 'Finishing':
        department = 0
    if department == 'finishing':
        department = 0

    day = request.form['Day of the week']
    if day == 'Monday':
        day = 0
    if day == 'Tuesday':
        day = 4
    if day == 'Wednesday':
        day = 5
    if day == 'Thursday':
        day = 3
    if day == 'Saturday':
        day = 1
    if day == 'Sunday':
        day = 2
```

```
team = int(request.form['Team Number'])
time = int(request.form['Time Allocated'])
items = int(request.form['Unfinished Items'])
over_time = int(request.form['Over time'])
incentive = int(request.form['Incentive'])
idle_time = int(request.form['Idle Time'])
idle_men = int(request.form['Idle Men'])
style = int(request.form['Style Change'])
workers = int(request.form['Number of Workers'])

prediction = model1.predict(pd.DataFrame([[quarter,department,day,team,time,items,over_time,incentive,idle_time,idle_men,style,workers]], c
   'over_time', 'incentive','idle_time','idle_men','style_change', 'no_of_workers']))


prediction = (np.round(prediction,4))*100


return render_template('productivity.html', prediction_text ="is {}".format(prediction))
```

Main Function:
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.

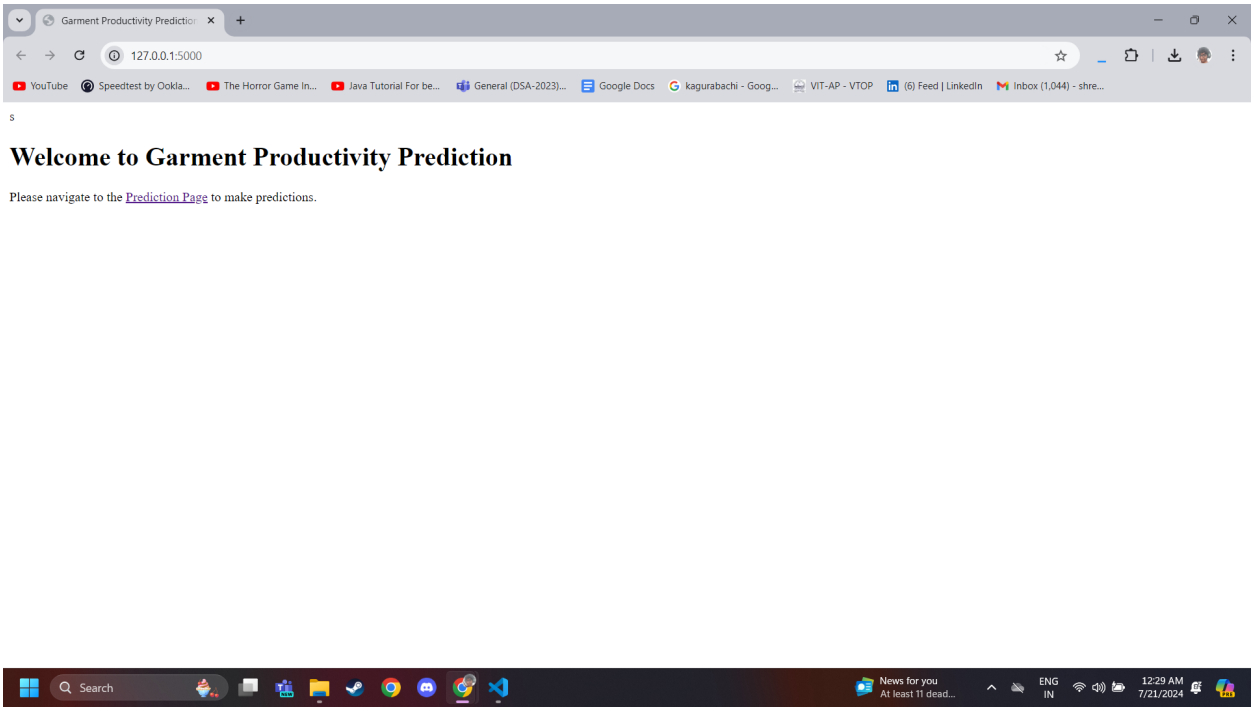
```
if __name__ == '__main__':
    app.run()
```

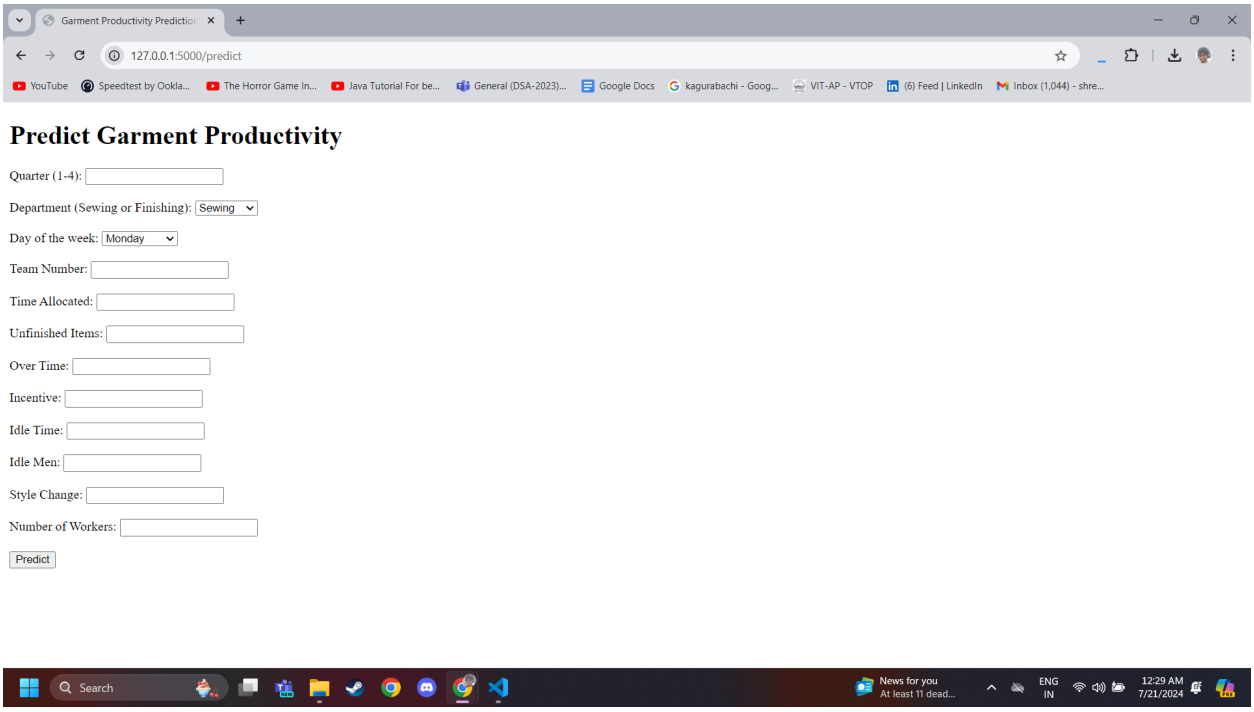
Activity 2.3: Run the web application

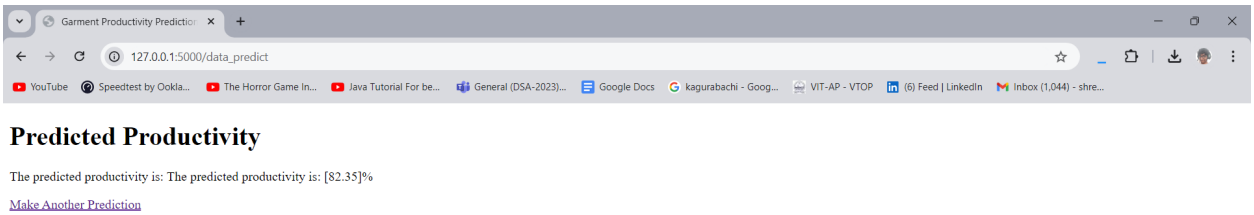
When you run the “main.py” file this window will open in the output terminal. Copy thehttp://127.0.0.1:5000and paste this link in your browser.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER                                  Code
[Running] python -u "d:\SmartBridge\Worker Productivity\app.py"
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
 * 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
```

6. Results


Garment Productivity Prediction
127.0.0.1:5000
YouTube
Speedtest by Ookla...
The Horror Game In...
Java Tutorial For be...
General (DSA-2023)...
Google Docs
kagurabachi - Goog...
VIT-AP - VTOP
(6) Feed | LinkedIn
Inbox (1,044) - shre...
s
Welcome to Garment Productivity Prediction
Please navigate to the Prediction Page to make predictions.
Search
News for you
At least 11 dead...
ENG
IN
12:29 AM
7/21/2024


Garment Productivity Prediction
127.0.0.1:5000/predict
YouTube
Speedtest by Ookla...
The Horror Game In...
Java Tutorial For be...
General (DSA-2023)...
Google Docs
kagurabachi - Goog...
VIT-AP - VTOP
(6) Feed | LinkedIn
Inbox (1,044) - shre...
Predict Garment Productivity
Quarter (1-4):
Department (Sewing or Finishing): Sewing
Day of the week: Monday
Team Number:
Time Allocated:
Unfinished Items:
Over Time:
Incentive:
Idle Time:
Idle Men:
Style Change:
Number of Workers:
Predict
Search
News for you
At least 11 dead...
ENG
IN
12:29 AM
7/21/2024



DEMO LINK: (youtube):

7. Advantage and disadvantages

Advantages

1. **Improved Efficiency:**
    a. **Resource Allocation:** Predictive models can help managers

allocate resources more effectively, ensuring that skilled workers are assigned to tasks where they can be most productive.
   b. **Time Management:** Anticipating productivity levels can aid in better scheduling, reducing downtime, and ensuring deadlines are met.

2. **Enhanced Quality Control:**
   a. **Consistent Output:** By understanding productivity patterns, companies can maintain consistent quality in their output.
   b. **Early Detection of Issues:** Predictive analytics can flag potential issues early, allowing for timely interventions to maintain quality.

3. **Cost Reduction:**
   a. **Labor Costs:** Optimizing worker productivity can lead to significant savings in labor costs.
   b. **Inventory Management:** Accurate productivity predictions help in better inventory management, reducing waste and excess stock.

4. **Strategic Planning:**
   a. **Capacity Planning:** Companies can plan for future capacity needs more accurately, ensuring they are neither overstaffed nor understaffed.
   b. **Investment Decisions:** Data-driven insights into productivity can inform investment in new technologies or training programs.

5. **Employee Development:**
   a. **Targeted Training:** Identifying areas where workers are less productive can help in designing targeted training programs to improve skills.
   b. **Performance Feedback:** Workers receive more accurate

feedback on their performance, aiding in personal development and motivation.

Disadvantages

1. **Data Dependency:**
    a. **Quality of Data:** Predictive accuracy is heavily dependent on the quality and quantity of data available. Poor data can lead to inaccurate predictions.
    b. **Data Privacy:** Collecting and storing detailed productivity data raises concerns about worker privacy and data security.

2. **Resistance to Change:**
    a. **Worker Opposition:** Employees might resist productivity monitoring and prediction, fearing increased surveillance or job insecurity.
    b. **Management Resistance:** Some managers might be skeptical of relying on predictive models, preferring traditional methods.

3. **Implementation Costs:**
    a. **Initial Investment:** Developing and implementing predictive models can be costly, requiring investment in technology and expertise.
    b. **Maintenance Costs:** Continuous updating and maintenance of predictive systems are necessary to ensure ongoing accuracy and relevance.

4. **Potential for Misuse:**
    a. **Overemphasis on Metrics:** There is a risk of focusing too much on productivity metrics at the expense of worker well-being and job satisfaction.
    b. **Unrealistic Expectations:** Predictive models might set unrealistic productivity targets, leading to increased pressure and potential burnout among workers.

5. **Complexity:**
    a. **Model Complexity:** Developing accurate predictive models can be complex and require specialized knowledge in data science and machine learning.
    b. **Dynamic Factors:** Productivity can be influenced by numerous dynamic factors such as worker health, external economic conditions, and seasonal variations, making predictions challenging.

8. Conclusion

Predicting garment worker productivity presents a promising approach to enhancing efficiency, quality, and cost management within the industry. The advantages include better resource allocation, improved quality control, and strategic planning, all of which contribute to more effective operations and informed decision-making. However, these benefits are tempered by challenges such as the need for high-quality data, potential resistance from workers and management, and the costs associated with implementing and maintaining predictive systems. Additionally, there is a risk of overemphasis on productivity metrics at the expense of worker well-being. To harness the full potential of productivity prediction, companies must address these challenges thoughtfully, ensuring a balanced approach that considers both technological advancements and the human factors involved.

9. Future Scope

Advanced Analytics and AI:

- **Machine Learning Enhancements:** Utilizing more sophisticated machine learning algorithms and deep learning techniques can improve the accuracy and reliability of productivity predictions.
- **Real-Time Analytics:** Implementing real-time data analytics can

provide immediate insights, allowing for quicker adjustments and interventions.

Integration with IoT:

- **Smart Wearables:** Integrating smart wearables to monitor worker movements and health can provide more granular data, enhancing predictive accuracy.
- **IoT-Enabled Machinery:** Using IoT-enabled machinery to collect detailed performance data can lead to a more comprehensive understanding of factors affecting productivity.

Customized Worker Training:

- **Personalized Training Programs:** Predictive models can identify specific areas where individual workers need improvement, leading to more effective, personalized training programs.
- **Gamification:** Introducing gamification elements to training based on predictive insights can enhance engagement and motivation among workers.

Enhanced Worker Well-being:

- **Health and Safety Monitoring:** Predictive models can incorporate data related to worker health and safety, ensuring that productivity improvements do not come at the cost of well-being.
- **Work-Life Balance:** Insights from productivity data can help in designing schedules that balance productivity with work-life balance, reducing burnout.

Sustainability and Ethical Practices:

- **Sustainable Practices:** Predictive models can help in optimizing resource use, reducing waste, and promoting sustainable practices

within the garment industry.

- **Ethical Labor Practices:** Ensuring that productivity improvements align with fair labor practices can enhance the industry's reputation and worker satisfaction.
-

10. Appendix
10.1 Source Code:
(On github)

10.2. GitHub & Project Demo Link:

Project demo link:

Github repo link:

https://github.com/Shreyuboss/GarmentWorkerEfficiency.git