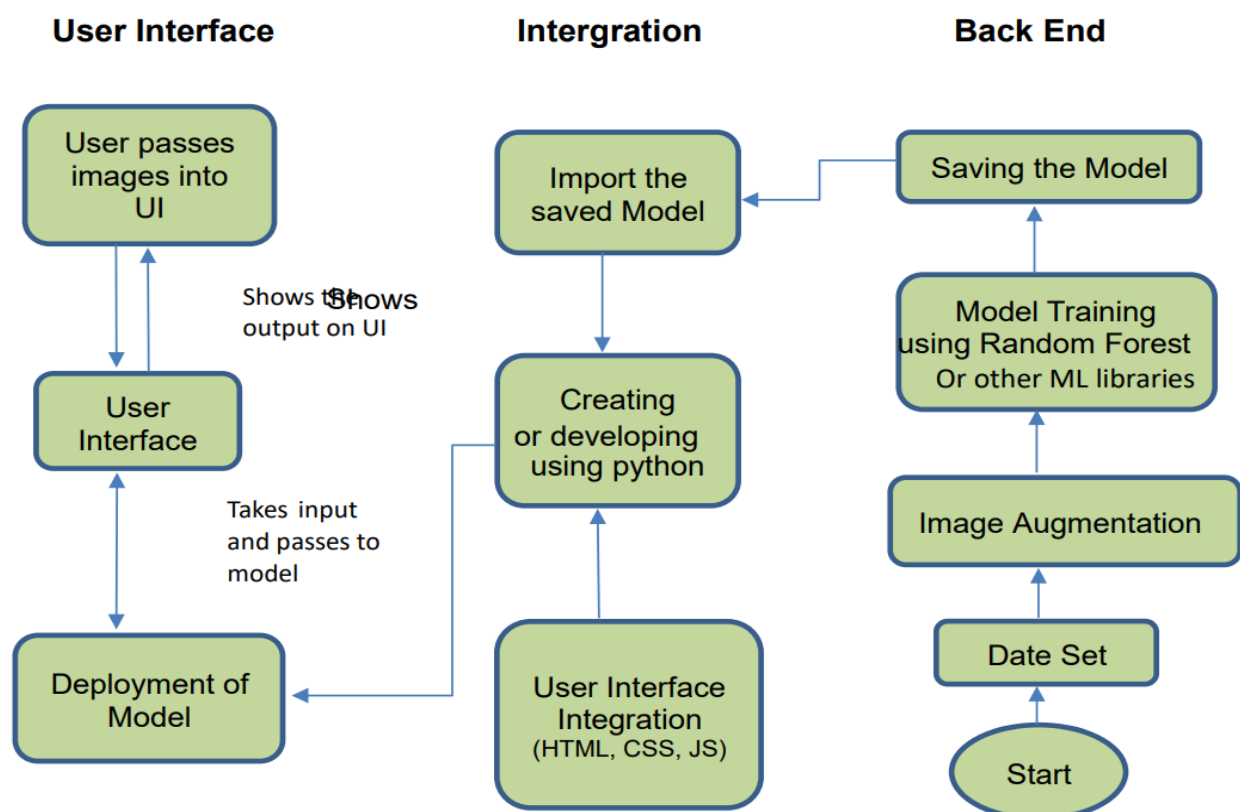


Garment Worker Productivity Prediction using Machine Learning

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:



Project Flow:

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

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
 - o Specify the business problem
 - o Business requirements
 - o Literature Survey
 - o Social or Business Impact
- Data Collection & Preparation
 - o Collect the dataset
 - o Data Preparation
- Exploratory Data Analysis
 - o Descriptive statistical
 - o Visual Analysis
- Collect the dataset
 - o Training the model in multiple algorithms
 - o Testing the model
- Performance Testing & Hyperparameter Tuning
 - o Testing model with multiple evaluation metrics
 - o Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
 - o Save the best model
 - o Integrate with Web Framework
- Project Demonstration & Documentation
 - o Record explanation Video for project end to end solution
 - o Project Documentation-Step by step project development procedure

Prior Knowledge:

You must have prior knowledge of following topics to complete this project. ● ML Concepts

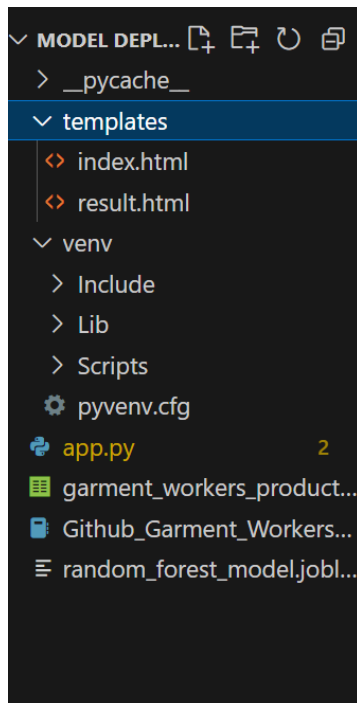
- Linear Regression: - <https://www.javatpoint.com/linear-regression-in-machine-learning>
- Decision Tree Regressor: - <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- Random Forest Regressor: - <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
- Gradient Boosting Regressor: - <https://www.javatpoint.com/gbm-in-machine-learning>
- Xtreme Gradient Boost Regressor: - <https://www.javatpoint.com/xgboost-ml-model-in-python>
- Bagging Regressor: - <https://www.javatpoint.com/bagging-vs-boosting> ○

Boosting Regressor: - <https://www.javatpoint.com/bagging-vs-boosting> ● Flask

Basics: - https://www.youtube.com/watch?v=Ij4l_CvBnt0

Project Structure:

Create a smart watch folder which contains files as shown below:



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.

- random_forest_model.joblib is where our machine learning model is saved. Further we will use this model for flask integration.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

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.

Activity 3: Literature Survey

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.

Activity 4: Social or Business Impact

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.

Milestone 2: Data collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

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.

Link:

https://archive.ics.uci.edu/dataset/597/productivity+prediction+of+garment+employee_s

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.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, BaggingRegressor, AdaBoostRegressor
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import warnings
```

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('garment_workers_productivity.csv')
df.head()
```

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_wo
0	1/1/2015	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	
1	1/1/2015	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	
2	1/1/2015	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	
3	1/1/2015	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	
4	1/1/2015	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	

Activity 2: Data Preparation

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
```

```
: df2.isnull().sum()

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

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.

```
df3 = df2.fillna({
    'wip': 0,
})
```

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

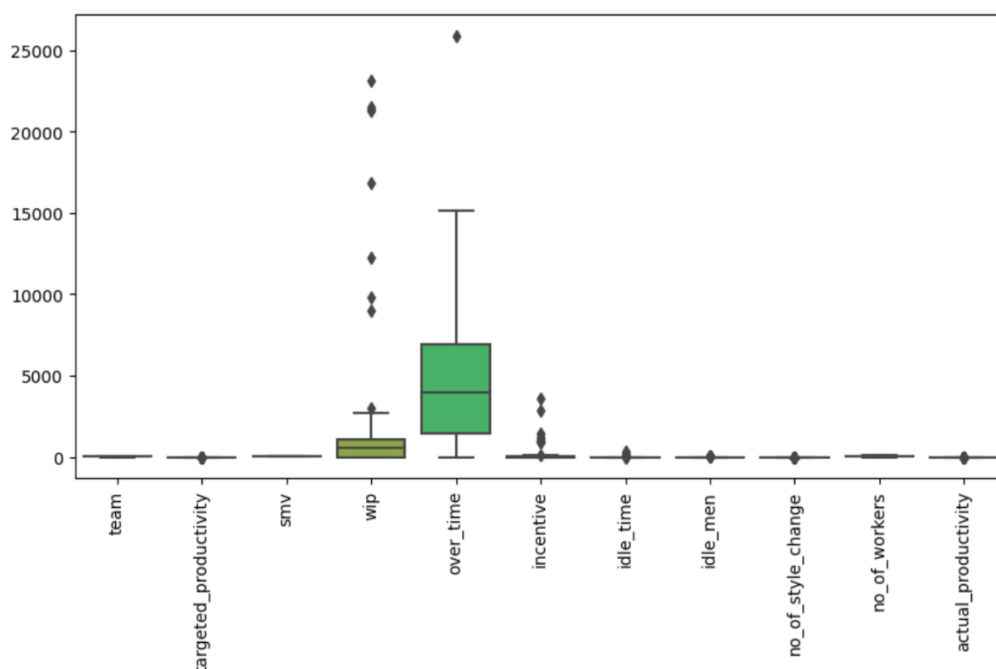
Activity 2.2: Outlier Detection and Removal

Outliers Detection

```
plt.figure(figsize=(10,5))
p = sns.boxplot(data = df3, orient = 'v',width=0.8)
plt.xticks(rotation=90)

(array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]),
 [Text(0, 0, 'team'),
  Text(1, 0, 'targeted_productivity'),
  Text(2, 0, 'smv'),
  Text(3, 0, 'wip'),
  Text(4, 0, 'over_time'),
  Text(5, 0, 'incentive'),
  Text(6, 0, 'idle_time'),
  Text(7, 0, 'idle_men'),
  Text(8, 0, 'no_of_style_change'),
  Text(9, 0, 'no_of_workers'),
  Text(10, 0, 'actual_productivity')])
```

Plot before removing outliers



Removing Outliers for 'incentive' column

```
: Q1 = df3.incentive.quantile(0.25)
: Q3 = df3.incentive.quantile(0.75)
: Q1, Q3
: (0.0, 50.0)

: IQR = Q3 - Q1
: IQR
: 50.0

: lower_limit = Q1 - 1.5*IQR
: upper_limit = Q3 + 1.5*IQR
: lower_limit, upper_limit
: (-75.0, 125.0)

: df3[(df3.incentive<lower_limit)|(df3.incentive>upper_limit)]
:
  quarter department team targeted_productivity smv wip over_time incentive idle_time idle_men no_of_style_change no_of_workers actual_p
730 Quarter2 sweing 1 0.80 22.52 1397.0 0 138 0.0 0 0 57.0
1128 Quarter2 finishing 11 0.80 2.90 0.0 0 960 0.0 0 0 8.0
1129 Quarter2 finishing 12 0.80 4.60 0.0 0 1080 0.0 0 0 9.0
1130 Quarter2 finishing 5 0.60 3.94 0.0 0 2880 0.0 0 0 12.0
1133 Quarter2 finishing 9 0.75 2.90 0.0 0 3600 0.0 0 0 15.0
1137 Quarter2 finishing 3 0.80 4.60 0.0 0 1440 0.0 0 0 12.0
1138 Quarter2 finishing 4 0.75 3.94 0.0 0 960 0.0 0 0 8.0
1139 Quarter2 finishing 1 0.75 3.94 0.0 0 960 0.0 0 0 8.0
1143 Quarter2 finishing 2 0.70 3.90 0.0 0 1200 0.0 0 0 10.0
1148 Quarter2 finishing 10 0.70 2.90 0.0 0 960 0.0 0 0 8.0
1149 Quarter2 finishing 8 0.65 3.90 0.0 0 960 0.0 0 0 8.0
< |

: df4 = df3[(df3.incentive>lower_limit)&(df3.incentive<upper_limit)]
: df4.shape
: (1186, 13)
```

This code snippet filters a DataFrame df3 by selecting rows where the 'incentive' column values are either less than a specified lower limit (lower_limit) or greater than a specified upper limit (upper_limit). The result is a DataFrame containing only the rows that meet this condition.

Removing Outliers for 'wip' column

```
: Q1 = df4.wip.quantile(0.25)
: Q3 = df4.wip.quantile(0.75)
: Q1, Q3
: (0.0, 1084.75)

: IQR = Q3 - Q1
: IQR
: 1084.75

: lower_limit = Q1 - 1.5*IQR
: upper_limit = Q3 + 1.5*IQR
: lower_limit, upper_limit
: (-1627.125, 2711.875)
```

Removing Outliers for 'over_time' column

```
] Q1 = df5.over_time.quantile(0.25)
Q3 = df5.over_time.quantile(0.75)
Q1, Q3
]: (1440.0, 6960.0)

]: IQR = Q3 - Q1
IQR
]: 5520.0

]: lower_limit = Q1 - 1.5*IQR
upper_limit = Q3 + 1.5*IQR
lower_limit, upper_limit
]: (-6840.0, 15240.0)

]: df5[(df5.over_time<lower_limit)|(df5.over_time>upper_limit)]
]:
   quarter  department  team  targeted_productivity  smv  wip  over_time  incentive  idle_time  idle_men  no_of_style_change  no_of_workers  actual_prodi
146  Quarter2      sweing    11                0.35  12.52  287.0    25920         38         0.0         0         0         54.0         0.0
<----->

]: df6 = df5[(df5.over_time>lower_limit)&(df5.over_time<upper_limit)]

]: df6.shape
]: (1176, 13)
```

Activity 2.3: Handling Categorical Values

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
for i in range(0, df6.shape[1]):
    if df6.dtypes[i] == 'object':
        df6.loc[:, df6.columns[i]] = le.fit_transform(df6.loc[:, df6.columns[i]])
```

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.

	quarter	department	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers
0	0	2	8	0.80	26.16	1108.0	7080	98	0.0	0	0	59.0
1	0	1	1	0.75	3.94	0.0	960	0	0.0	0	0	8.0
2	0	2	11	0.80	11.41	968.0	3660	50	0.0	0	0	30.5
3	0	2	12	0.80	11.41	968.0	3660	50	0.0	0	0	30.5
4	0	2	6	0.80	25.90	1170.0	1920	50	0.0	0	0	56.0

<----->

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.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

The purpose of descriptive analysis is to analyze the basic features of data using a statistical technique. In this case, Pandas has a useful function called describe. We can understand the unique, top, and frequent values of categorical features with this describe function. We can also get the count, mean, standard deviation, minimum, maximum, and percentile values of continuous features.

```
df.describe(include="all")
```

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time
count	1197	1197	1197	1197	1197.000000	1197.000000	1197.000000	691.000000	1197.000000	1197.000000	1197.000000
unique	59	5	3	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	3/11/2015	Quarter1	sweing	Wednesday	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	24	360	691	208	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	6.426901	0.729632	15.062172	1190.465991	4567.460317	38.210526	0.730159
std	NaN	NaN	NaN	NaN	3.463963	0.097891	10.943219	1837.455001	3348.823563	160.182643	12.709757
min	NaN	NaN	NaN	NaN	1.000000	0.070000	2.900000	7.000000	0.000000	0.000000	0.000000
25%	NaN	NaN	NaN	NaN	3.000000	0.700000	3.940000	774.500000	1440.000000	0.000000	0.000000
50%	NaN	NaN	NaN	NaN	6.000000	0.750000	15.260000	1039.000000	3960.000000	0.000000	0.000000
75%	NaN	NaN	NaN	NaN	9.000000	0.800000	24.260000	1252.500000	6960.000000	50.000000	0.000000
max	NaN	NaN	NaN	NaN	12.000000	0.800000	54.560000	23122.000000	25920.000000	3600.000000	300.000000

Activity 2: Visual analysis

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.

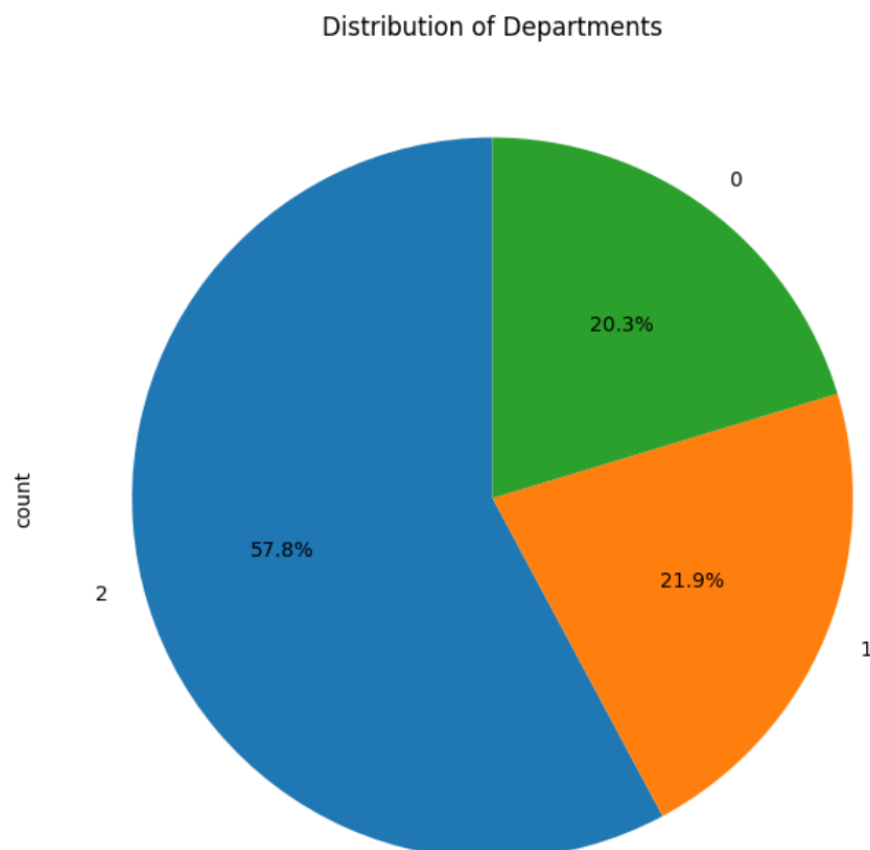
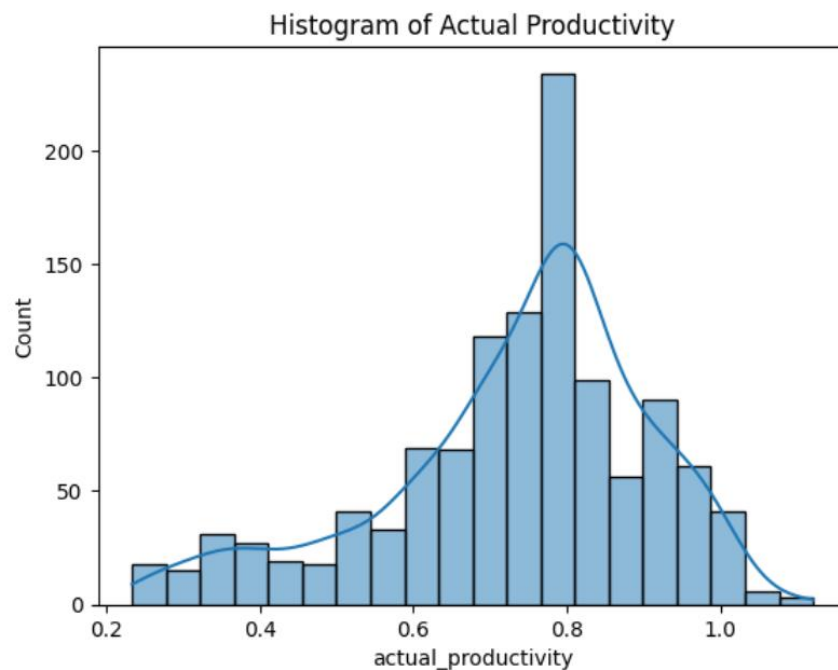
```
import matplotlib.pyplot as plt
import seaborn as sns

# Univariate analysis using a pie chart
plt.figure(figsize=(8, 8))
df6['department'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Departments')
plt.show()

# Univariate analysis using a histogram
sns.histplot(df6['actual_productivity'], bins=20, kde=True)
plt.title('Histogram of Actual Productivity')
plt.show()

# Descriptive statistics
print(df6['actual_productivity'].describe())
```

This code creates a histogram and piechart 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.



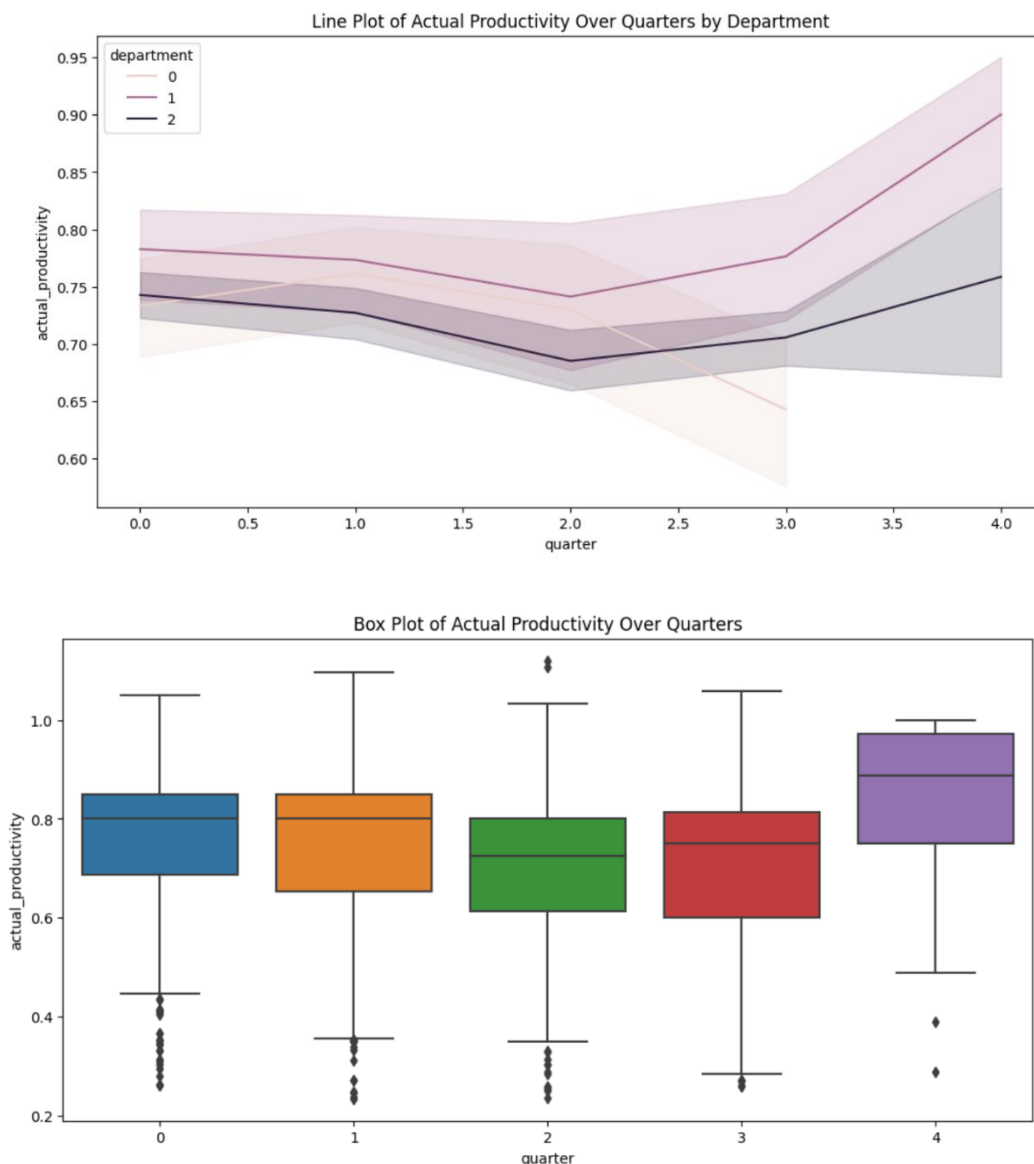
Activity 2.2: Bivariate analysis

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.

```
# Bivariate analysis using a line plot
plt.figure(figsize=(12, 6))
sns.lineplot(x='quarter', y='actual_productivity', data=df6, hue='department')
plt.title('Line Plot of Actual Productivity Over Quarters by Department')
plt.show()

# Bivariate analysis using a box plot
plt.figure(figsize=(12, 6))
sns.boxplot(x='quarter', y='actual_productivity', data=df6)
plt.title('Box Plot of Actual Productivity Over Quarters')
plt.show()
```

This code generates a line plot and box 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".

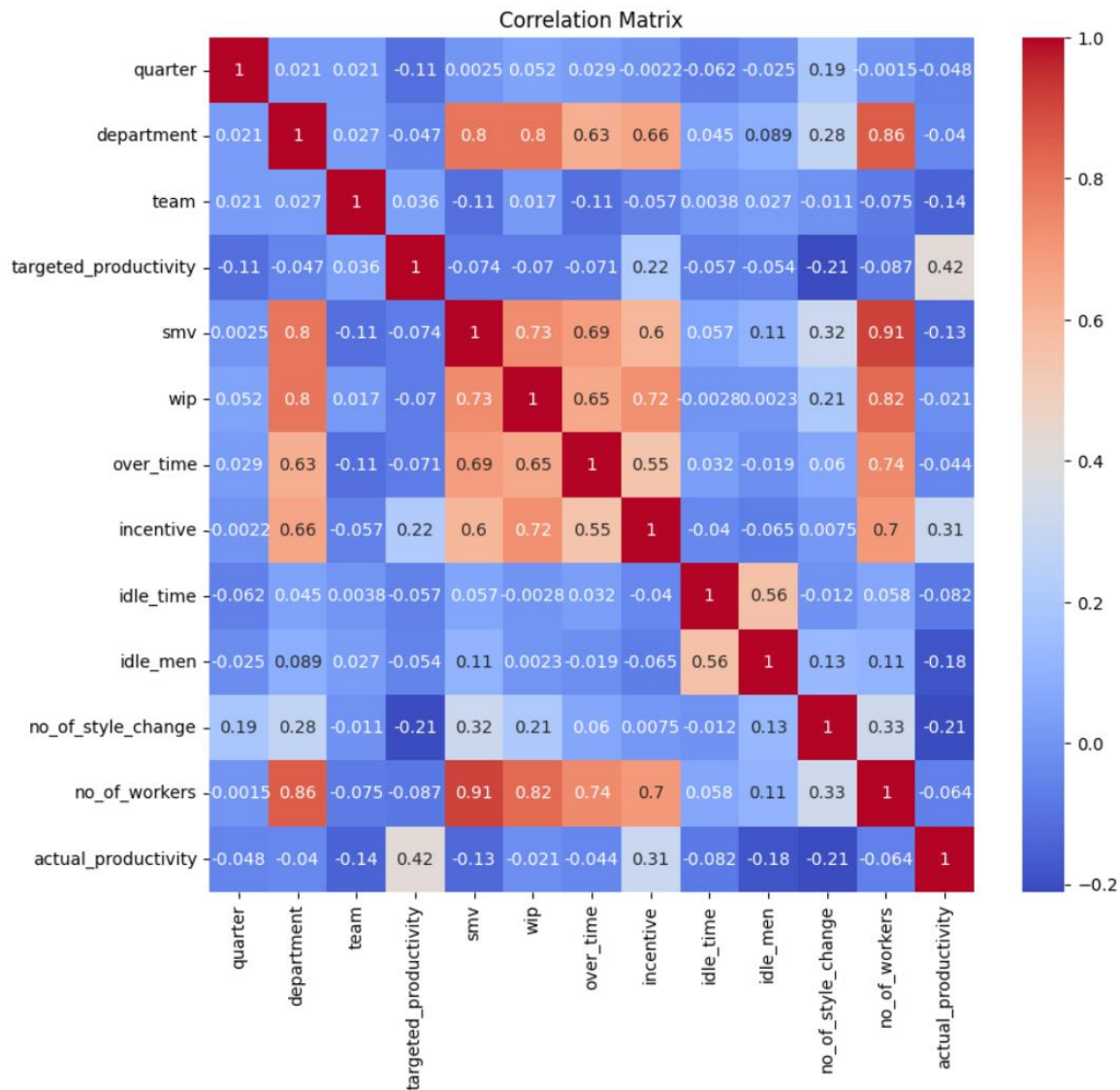


Activity 2.3: Multivariate analysis

Multivariate analysis is a statistical technique used to analyse data that involves more than two variables. It aims to understand the relationships between multiple variables in a dataset by examining how they are related to each other and how they contribute to a particular outcome or phenomenon.

```
# Correlation matrix
plt.figure(figsize=(10, 9))
correlation_matrix = df6.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

This code creates a heatmap to show the correlation between the numerical variables in the dataset. The darker the color of the square, the stronger the correlation between the variables. The annotated values in the square show the correlation coefficients.



Splitting data into train and test

Now let us split the Dataset into train and test sets. First split the dataset into X and y and then split the data set. The 'X' corresponds to independent features and 'y' corresponds to the target variable.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=42)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(823, 12)
(353, 12)
(823,)
(353,)
```

Applying Standard Scaler

We apply standard scaling to independent variables before training because it helps to normalize the data and brings all the features to a similar scale. This is important because some machine learning algorithms are sensitive to the scale of the input features, and having features on vastly different scales can lead to suboptimal performance.

```
#Standardization
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
x.iloc[:, :] = scaler.fit_transform(x.iloc[:, :])
```

```
x.head()
```

	quarter	department	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers
0	-1.160117	0.780739	0.455616	0.720481	1.008608	0.926217	0.760786	2.413020	-0.057985	-0.114022	-0.355135	1.097504
1	-1.160117	-0.468444	-1.571751	0.209020	-1.018120	-1.019391	-1.102067	-0.847073	-0.057985	-0.114022	-0.355135	-1.201499
2	-1.160117	0.780739	1.324487	0.720481	-0.336767	0.680382	-0.280220	0.816240	-0.057985	-0.114022	-0.355135	-0.187233
3	-1.160117	0.780739	1.614110	0.720481	-0.336767	0.680382	-0.280220	0.816240	-0.057985	-0.114022	-0.355135	-0.187233
4	-1.160117	0.780739	-0.123632	0.720481	0.984893	1.035087	-0.809855	0.816240	-0.057985	-0.114022	-0.355135	0.962269

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

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

```
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
▼ LinearRegression
LinearRegression()
```

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

```
# Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
```

```
▼ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

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

```
# Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
```

```
▼ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

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.

Activity 1.4: Gradient Boosting Regressor Model

```
# Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
```

```
▼ GradientBoostingRegressor
GradientBoostingRegressor(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: Bagging Regressor Model

```
# Bagging Regressor
bagging_model = BaggingRegressor(base_estimator=DecisionTreeRegressor(random_state=42), random_state=42)
bagging_model.fit(X_train, y_train)
```

```
► BaggingRegressor
► base_estimator: DecisionTreeRegressor
  ► DecisionTreeRegressor
```

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 1.6: Boosting Regressor Model

```
: # AdaBoost Regressor
adaboost_model = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(), random_state=42)
adaboost_model.fit(X_train, y_train)
```

```
: ► AdaBoostRegressor
► base_estimator: DecisionTreeRegressor
  ► DecisionTreeRegressor
```

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.

Milestone 5: Performance Testing

Activity 1: Check model performance on test and train data for each model

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.

```
from sklearn.metrics import mean_squared_error
import numpy as np

def calculate_rmse(model, X, y):
    y_pred = model.predict(X)
    mse = mean_squared_error(y, y_pred)
    rmse = np.sqrt(mse)
    return rmse

# Assuming you have already trained the models (linear_model, dt_model, rf_model, etc.)

# Calculate training and testing RMSE for each model
models = [lr_model, dt_model, rf_model, gb_model, bagging_model, adaboost_model]
for model in models:
    training_rmse = calculate_rmse(model, X_train, y_train)
    testing_rmse = calculate_rmse(model, X_test, y_test)

    print(f"Model: {model.__class__.__name__}")
    print(f"Training RMSE: {training_rmse}")
    print(f"Testing RMSE: {testing_rmse}")
    print("\n")
```

Activity 1.1: Linear Regression Model

```
Model: LinearRegression
Training RMSE: 0.1407011819804832
Testing RMSE: 0.13083482822830603
```

Activity 1.2: Decision Tree Regressor Model

```
Model: DecisionTreeRegressor
Training RMSE: 0.047160676328553114
Testing RMSE: 0.1433087250101286
```

Activity 1.3: Random Forest Regressor Model

```
Model: RandomForestRegressor
Training RMSE: 0.06279379468436488
Testing RMSE: 0.11471972585767
```

Activity 1.4: Gradient Boosting Regressor Model

Model: GradientBoostingRegressor
Training RMSE: 0.10282959042051591
Testing RMSE: 0.11202004224041091

Activity 1.5: Bagging Regressor Model

Model: BaggingRegressor
Training RMSE: 0.06652112234015904
Testing RMSE: 0.12441021614941541

Activity 1.6: Boosting Regressor Model

Model: AdaBoostRegressor
Training RMSE: 0.05255873478298113
Testing RMSE: 0.1278236391140091

Activity 2: Comparing models

```
import pandas as pd

# Assuming you have already calculated the training and testing RMSE for each model
results = []

models = [lr_model, dt_model, rf_model, gb_model, bagging_model, adaboost_model]
for model in models:
    training_rmse = calculate_rmse(model, X_train, y_train)
    testing_rmse = calculate_rmse(model, X_test, y_test)

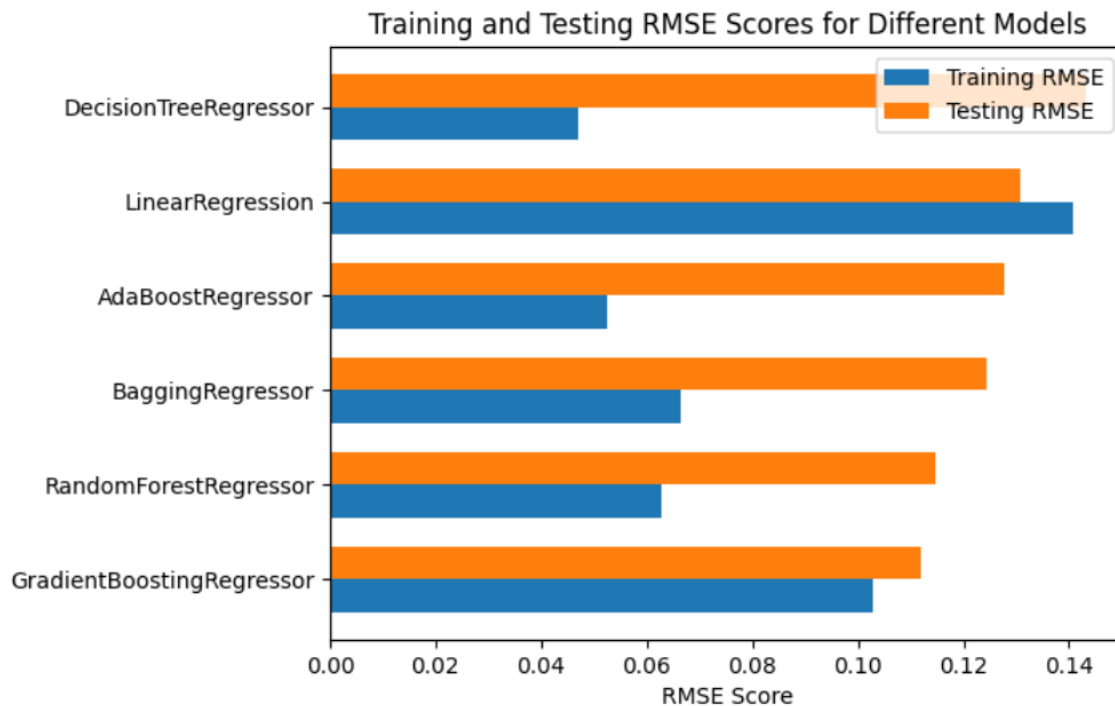
    results.append({
        'Model': model.__class__.__name__,
        'Training_RMSE': training_rmse,
        'Testing_RMSE': testing_rmse
    })

# Create a DataFrame from the results
results_df = pd.DataFrame(results)

# Display the results
print(results_df)
```

	Model	Training_RMSE	Testing_RMSE
0	LinearRegression	0.140701	0.130835
1	DecisionTreeRegressor	0.047161	0.143309
2	RandomForestRegressor	0.062794	0.114720
3	GradientBoostingRegressor	0.102830	0.112020
4	BaggingRegressor	0.066521	0.124410
5	AdaBoostRegressor	0.052559	0.127824

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.

Milestone 6: Model Deployment

Activity 1: Save the best model

```
import joblib

# Assuming best_rf_model is your trained Random Forest model
best_rf_model = RandomForestRegressor(max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=150)

# Fit the model to your entire dataset before saving
best_rf_model.fit(X, y)

# Save the model to a file
joblib.dump(best_rf_model, 'random_forest_model.joblib')
```

This code uses the "joblib" library in Python to save the trained random forest model named "random forest" as a file named "random_forest_model.joblib".

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.

Activity 2: Integrate with Web Framework

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:

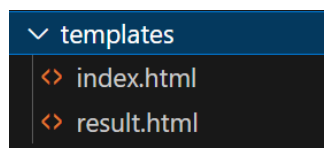
- Building HTML pages
- Building server side script
- Run the web application

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

```
from flask import Flask, render_template, request
from joblib import load
```

This code first loads the saved Boosting Regressor model from the "random_forest_model.joblib" file using the "joblib.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.

```
# Load the saved model
model = load('random_forest_model.joblib')
```

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".

```
return render_template('result.html', prediction=prediction[0])
```

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.

```
quarter = float(request.form['quarter'])
department = float(request.form['department'])
day = float(request.form['day'])
team = float(request.form['team'])
targeted_productivity = float(request.form['targeted_productivity'])
standard_minute_value = float(request.form['standard_minute_value'])
work_in_progress = float(request.form['work_in_progress'])
over_time = float(request.form['over_time'])
incentive = float(request.form['incentive'])
idle_men = float(request.form['idle_men'])
no_of_style_change = float(request.form['no_of_style_change'])
no_of_workers = float(request.form['no_of_workers'])

# Make a prediction
prediction = model.predict([[quarter, department, day, team, targeted_productivity,
                             standard_minute_value, work_in_progress, over_time,
                             incentive, idle_men, no_of_style_change, no_of_workers]])

return render_template('result.html', prediction=prediction[0])
```

Main Function:

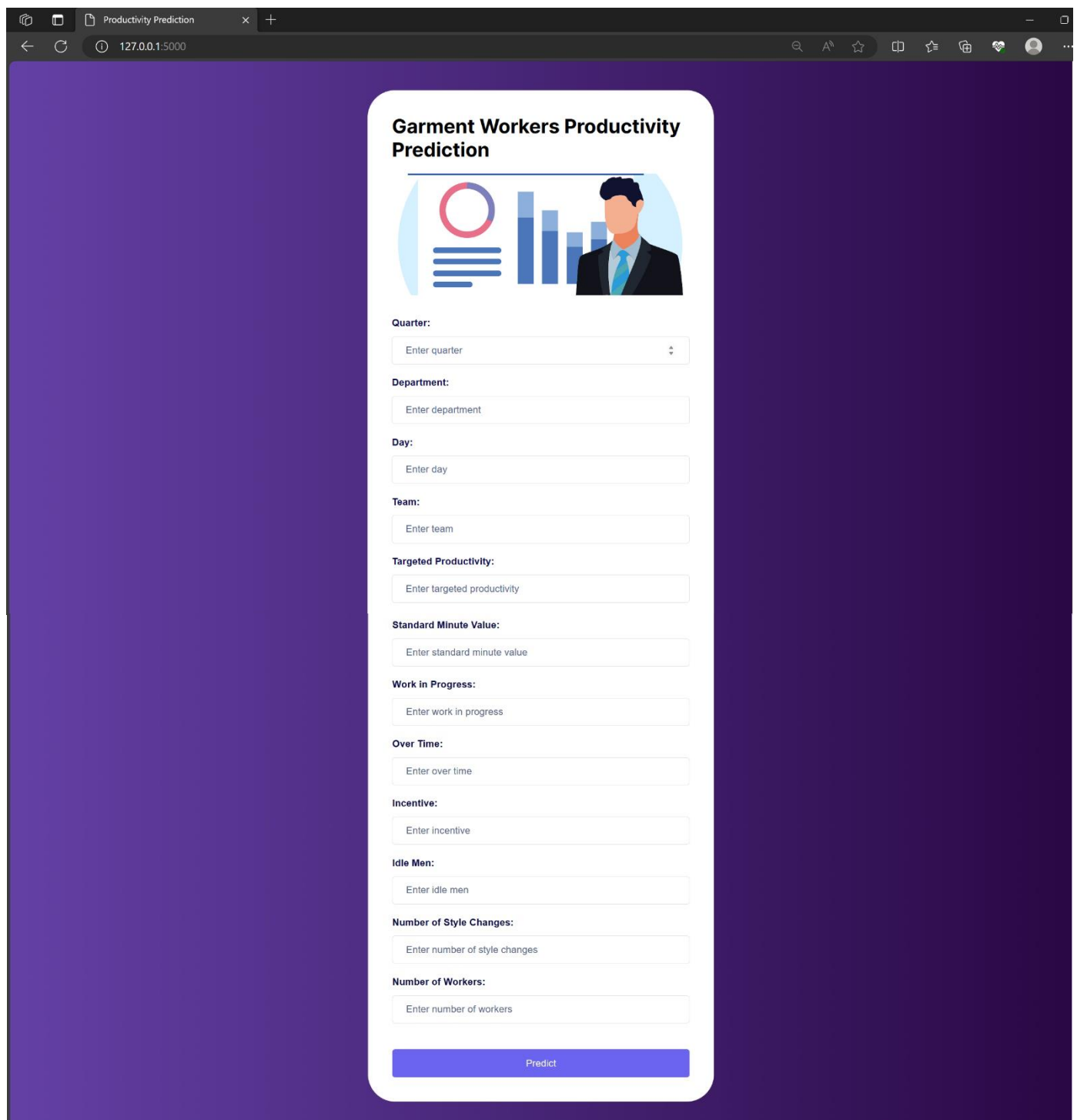
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.

Activity 2.3: Run the web application


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

```
PS D:\Internships\AI and ML\Final Project\IPYNB\Model deployment> set FLASK_APP=app.py
PS D:\Internships\AI and ML\Final Project\IPYNB\Model deployment> flask run
>>
* 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
```

Website Screenshots:



Garment Workers Productivity Prediction



Quarter:

Department:

Day:

Team:

Targeted Productivity:

Standard Minute Value:

Work in Progress:

Over Time:

Incentive:

Idle Men:

Number of Style Changes:

Number of Workers:

