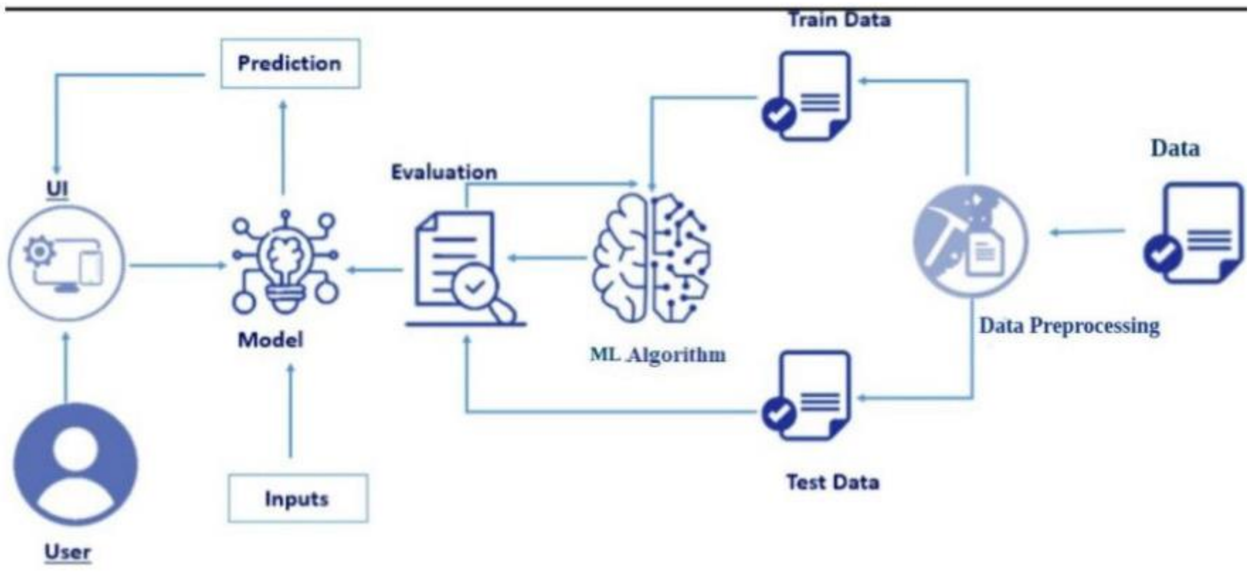
Date	06 November 2024
Team ID	592545
Project Name	Garment Worker Productivity Prediction using Machine Learning

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:



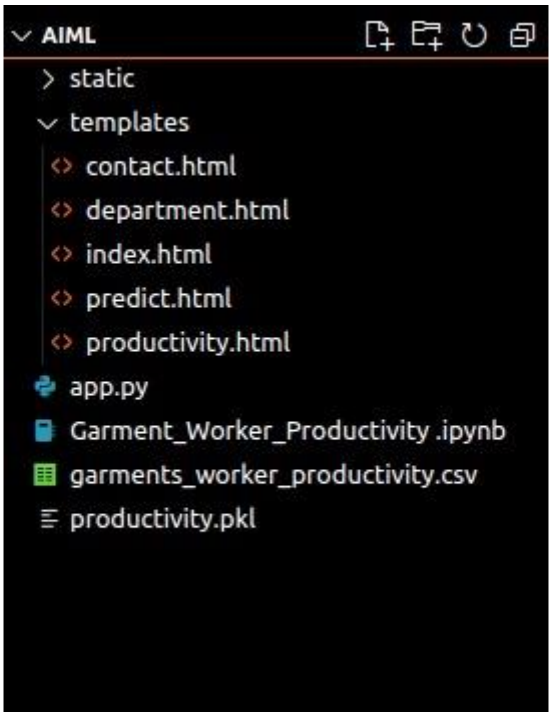
Project Flow:

- 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
  - 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

Project Structure:



```
AIML
  > static
  v templates
      contact.html
      department.html
      index.html
      predict.html
      productivity.html
  app.py
  Garment_Worker_Productivity .ipynb
  garments_worker_productivity.csv
  productivity.pkl
```

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

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.

Activity 2: Business requirements

To guarantee that the apparel trader productivity guess model meets trade necessities and maybe redistributed for public use, it should understand the following rules and necessities:

- **Accuracy:** The accuracy requirement entails that the model should consistently make predictions that closely align with actual worker productivity. A low margin of error is essential to instill confidence in the model's reliability, ensuring that businesses can make data-driven decisions with precision.
- **Privacy and security:** To protect user data, the model should adhere to privacy and security regulations. This includes the secure storage of sensitive data, encryption of data in transit, and the implementation of strict data access controls to prevent unauthorized access.
- **Interpretability:** Model interpretability means that the predictions generated by the model can be explained and understood by end-users. This is crucial for building trust in the model's output and allowing users to comprehend how and why certain predictions were made, which can be especially important in regulated industries.
- **Data Retention Policies:** Develop clear policies for the retention and deletion of data. These policies should align with legal requirements and principles of data minimization, ensuring that data is only retained for as long as necessary.
- **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.
- **Cost-Benefit Analysis:** Conduct a cost-benefit analysis to assess the economic value of implementing the model. This involves comparing the costs of model development and deployment against the expected benefits, such as cost savings or revenue increases.
- **Documentation and Reporting:** Maintain comprehensive documentation about the model's development, training data, and performance metrics. Regularly generate reports that provide insights into how the model is performing and its impact on business operations.

Activity 3: Literature Survey

A literature survey for a garment worker productivity prediction project involves an extensive examination of existing research, scholarly articles, and publications within the domains of machine learning, manufacturing, and workforce management. The overarching goal of this survey is to acquire a comprehensive understanding of the current landscape of productivity prediction methods within the garment industry. Key aspects to explore include feature selection strategies, data pre-processing techniques, and the machine learning algorithms that are commonly employed to predict productivity in this sector.

Furthermore, the literature survey will delve into identifying gaps in knowledge and untapped research opportunities within the realm of garment worker productivity prediction. This encompasses the exploration of emerging machine learning techniques like reinforcement learning, which could potentially offer novel insights and capabilities in this context. Additionally, the survey will explore the prospects of integrating diverse data sources, such as wearable technology and environmental sensors, to enhance the accuracy and scope of productivity predictions.

In essence, this literature survey serves as a critical foundational step in the garment worker productivity prediction project, aiding in the identification of best practices, research gaps, and innovative avenues for harnessing machine learning to optimize workforce performance in the garment industry.

Activity 4: Social or Business Impact

Social Impact: The garment worker productivity prediction model holds the potential to positively impact the lives of garment workers. By identifying the factors influencing worker productivity, the model can pave the way for fair and efficient workforce management practices. It can recommend improvements in working conditions, incentive programs, and training initiatives, resulting in a more satisfying work experience for garment workers. This, in turn, can lead to better compensation, enhanced working conditions, and overall improved job satisfaction among the workforce.

Business Impact: The implications of the garment worker productivity prediction model are substantial for the garment manufacturing industry. The model serves as a catalyst for optimizing workforce management strategies, efficiently reducing idle time, and boosting overall productivity. The resulting benefits are higher profitability, lower production costs, and superior product quality. Moreover, the model aids in pinpointing areas ripe for process enhancement, such as reducing rework and streamlining supply chain operations, contributing to a more competitive and efficient manufacturing landscape.

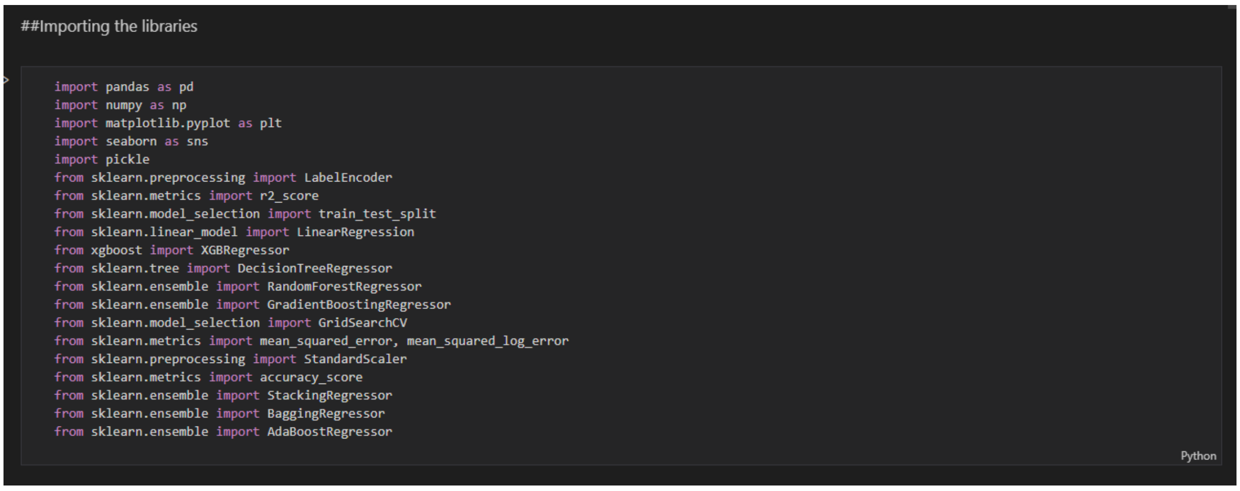
Milestone 2: Data collection & Preparation

Machine Learning depends massively on data. It is ultimately an important aspect that from treasure training likely. So, this division allows you to load the necessary dataset.

Activity 1: Collect the dataset

Link: https://www.kaggle.com/datasets/ishadss/productivity-prediction-of-garment-employe es

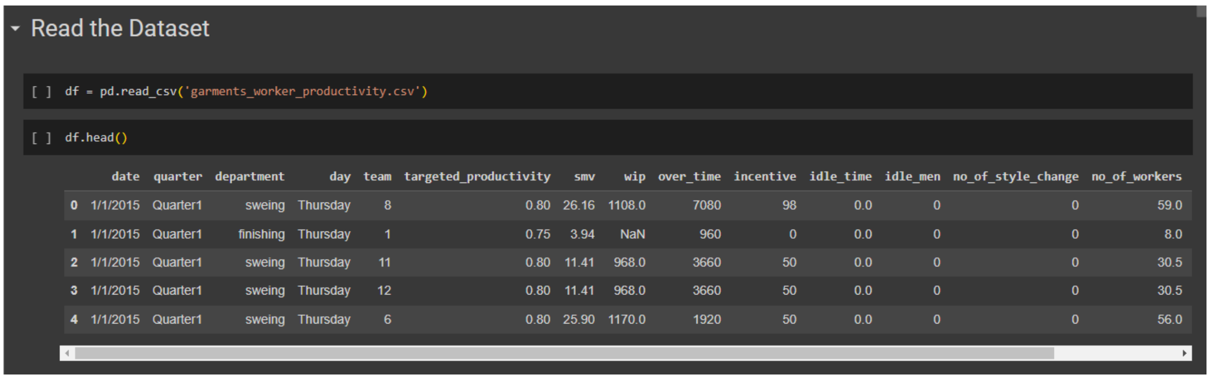
Activity 1.1: Importing the libraries



##Importing the libraries

```python
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



Read the Dataset

```
[ ] df = pd.read_csv('garments_worker_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_workers
0	1/1/2015	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	59.0
1	1/1/2015	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	8.0
2	1/1/2015	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	30.5
3	1/1/2015	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	30.5
4	1/1/2015	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	56.0

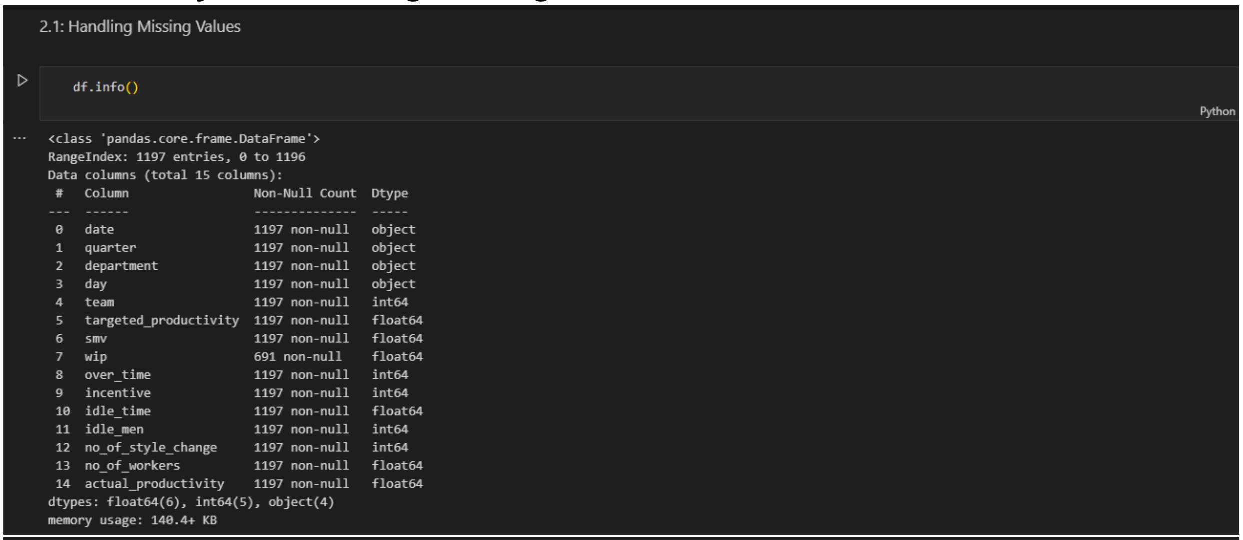
Activity 2: Data Preparation

Data preparation, often referred to as data preprocessing, constitutes the crucial phase of refining, reshaping, and structuring raw data, making it suitable for deployment in data analysis or training machine learning models. This critical process encompasses several key activities:

- **Outlier Management:** To enhance the data's quality and reliability, dealing with outliers is paramount. These exceptional data points may need adjustments or special handling, depending on the context.
- **Handling Missing Values:** One of the initial steps is addressing the presence of missing data points. This involves strategies such as imputation, removal, or other methods to fill in or discard incomplete information.
- **Categorical Variable Encoding:** Transforming categorical variables into a numerical format is a fundamental task. Various techniques like one-hot encoding or label encoding are employed to make these variables machine-readable.
- **Data Normalization:** Normalizing data helps to bring all features to a consistent scale, preventing certain features from dominating others. Techniques like min-max scaling or standardization are applied to achieve this.

It's essential to note that these are general guidelines for preprocessing, and the specific steps undertaken can vary depending on the nature and quality of the dataset. Not all datasets will necessitate all of these steps, and the extent of preprocessing may be tailored to the dataset's unique characteristics and requirements.

Activity 2.1: Handling Missing Values

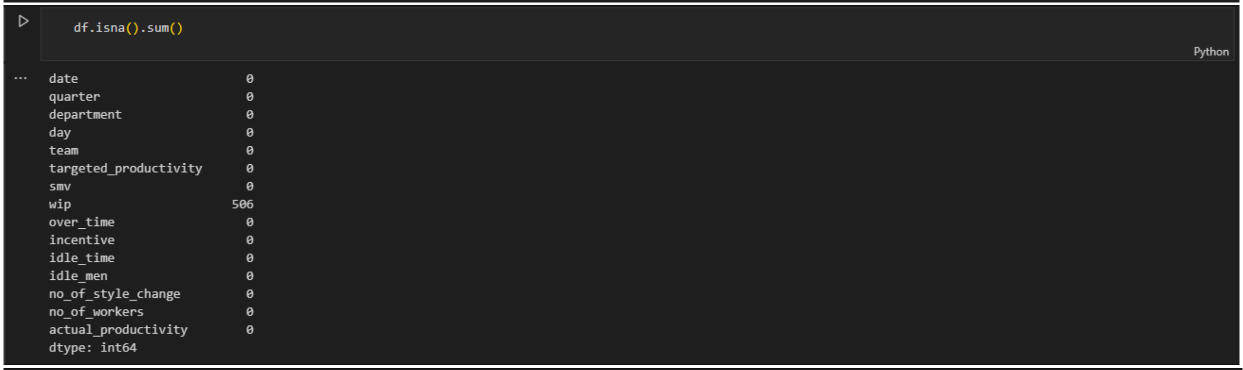


2.1: Handling Missing Values

```
df.info()
```

Python

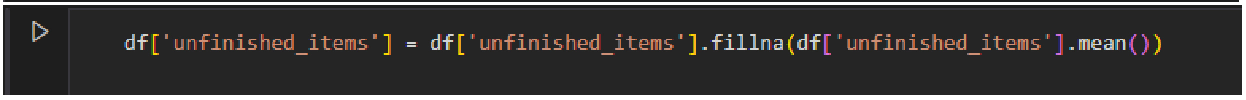
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   date                   1197 non-null   object
 1   quarter                1197 non-null   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()
```

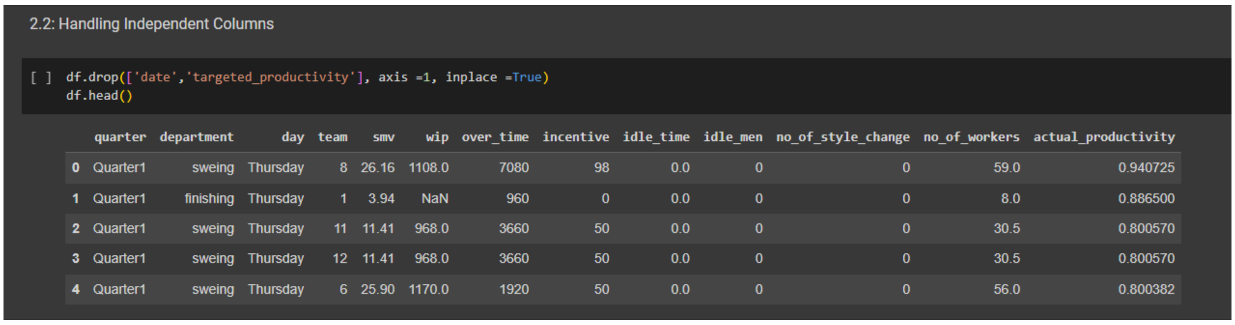
Python

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



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

Activity 2.2: Handling Independent Columns

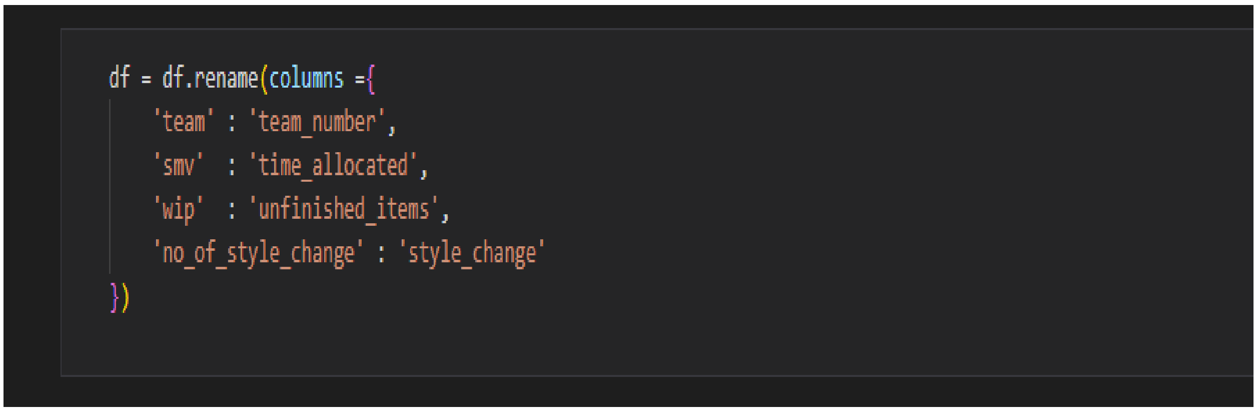


2.2: Handling Independent Columns

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

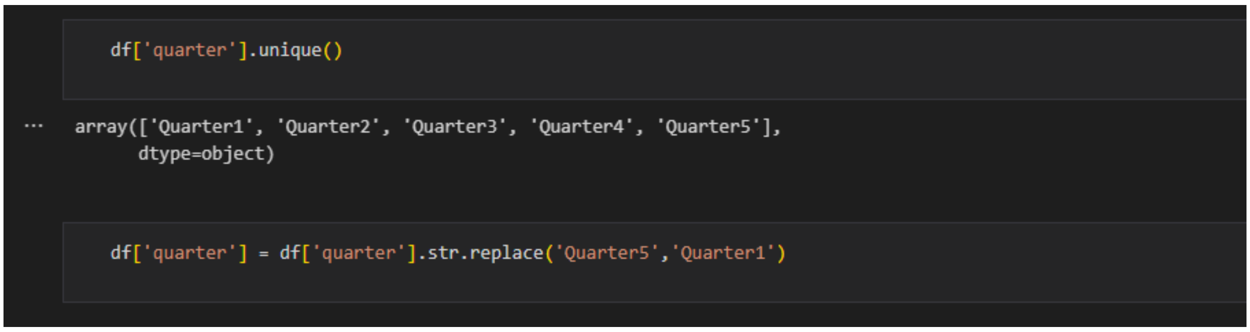
	quarter	department	day	team	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
0	Quarter1	sweing	Thursday	8	26.16	1108.0	7080	98	0.0	0	0	59.0	0.940725
1	Quarter1	finishing	Thursday	1	3.94	NaN	960	0	0.0	0	0	8.0	0.886500
2	Quarter1	sweing	Thursday	11	11.41	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

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



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

The "rename()" method proves invaluable when it comes to altering column names within the dataset. This method is designed to accept a dictionary as its input, where the original column names are represented as keys and the desired new names as corresponding values.

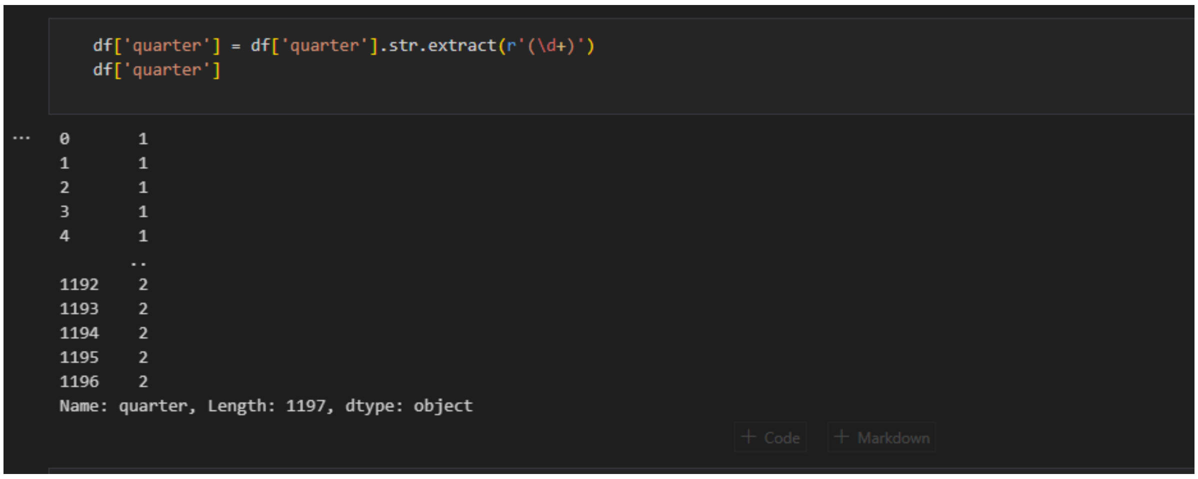


```python
df['quarter'].unique()
```

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

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

In the initial line of code, we target a specific column titled "quarter" within a dataset. We proceed by employing the "unique()" method to compile and display an array of distinct values contained within that particular column.
Subsequently, the following line of code is dedicated to data manipulation. Here, we identify and modify certain entries within the "quarter" column. To be more precise, any occurrences of the text "Quarter5" are replaced with "Quarter1" utilizing the "str.replace()" method.

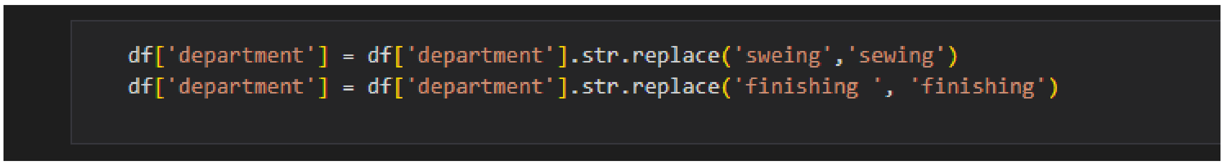


```python
df['quarter'] = df['quarter'].str.extract(r'(\d+)')
df['quarter']
```

```
0       1
1       1
2       1
3       1
4       1
       ..
1192    2
1193    2
1194    2
1195    2
1196    2
Name: quarter, Length: 1197, dtype: object
```

In the initial line of code, the "str.extract()" method is applied to the "quarter" column to isolate the numerical component from each entry. This is accomplished by employing a regular expression pattern, denoted as r'(\d+)', which identifies any consecutive sequence of one or more digits within each value.

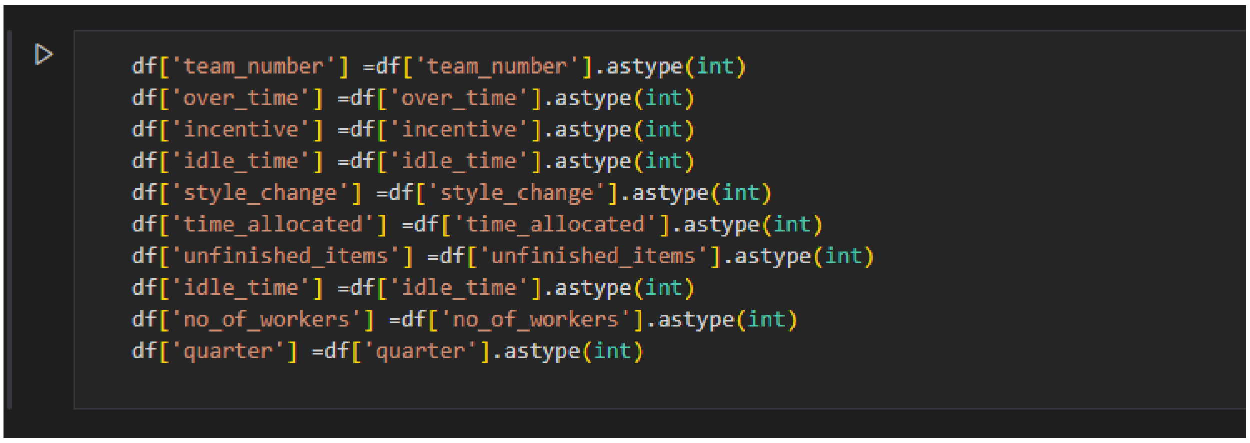
Subsequently, the second line of code reassigns the altered "quarter" column back to the same "quarter" column. Essentially, this operation replaces the original column with the modified version, preserving the dataset's integrity.



```
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".

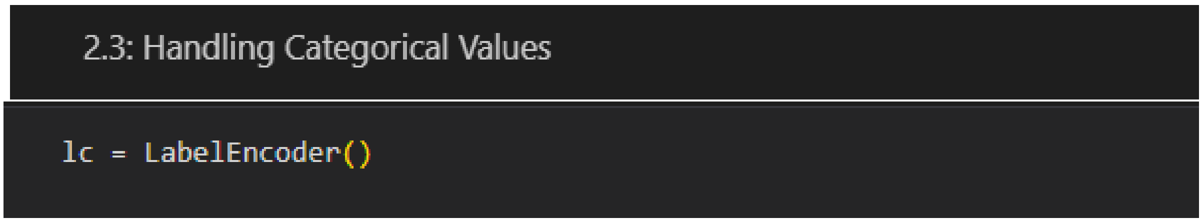
The second line of code is also using the str.replace() method on the "department" column to replace any instances of the word "finishing " (with a space at the end) with the word "finishing" (without a space at the end). This will remove the extra space and standardize the spelling of the word.



```
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['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)
df['quarter'] =df['quarter'].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.

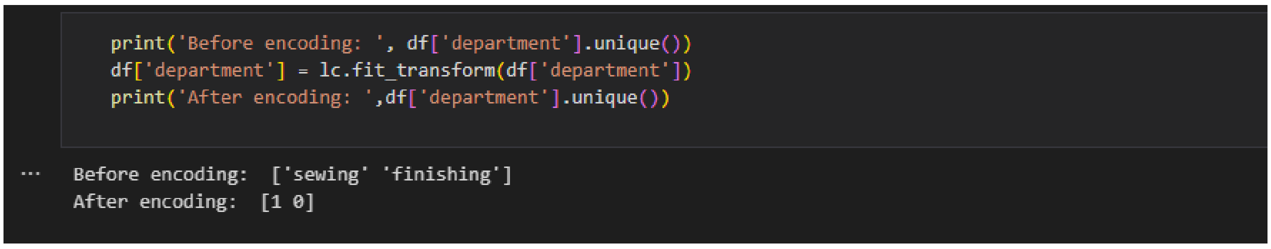
Activity 2.3: Handling Categorical Values



2.3: Handling Categorical Values

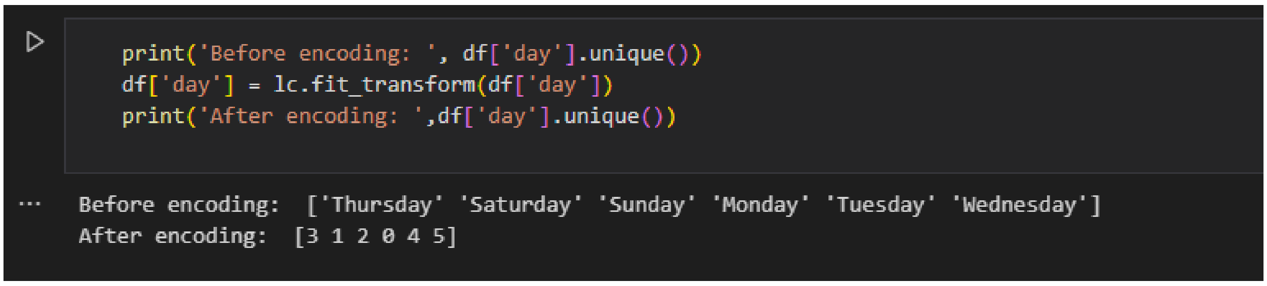
```
lc = LabelEncoder()
```

This code snippet is responsible for encoding the values within a column labeled "department." It accomplishes this by employing a "LabelEncoder()" object to transform the original values into numerical representations. Before and after the encoding process, the script displays the original and newly encoded values for reference.



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

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



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

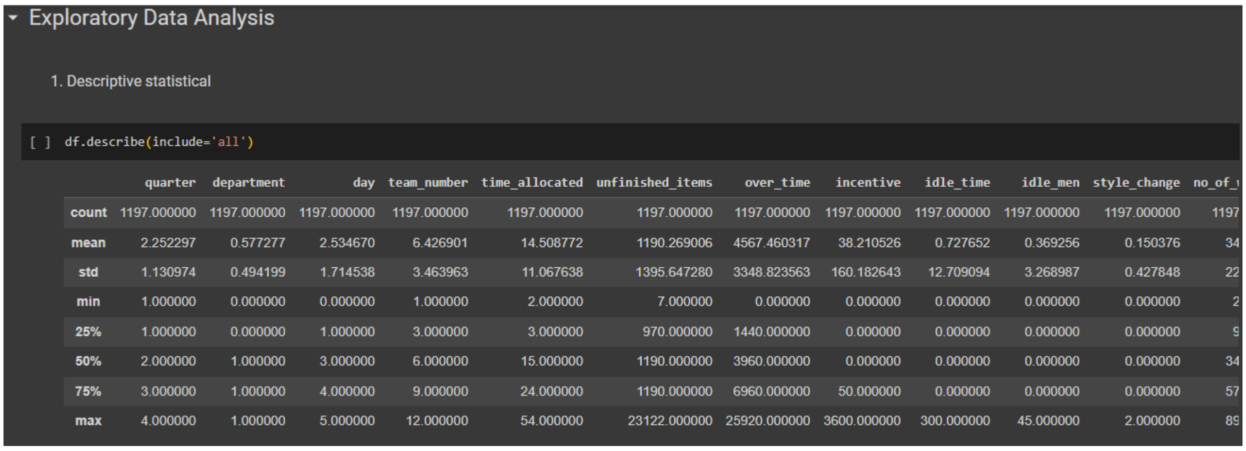
```
Before encoding:  ['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
After encoding:  [3 1 2 0 4 5]
```

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.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis serves the fundamental purpose of examining key data attributes through statistical methods. Pandas provides a valuable tool known as "describe" for this task. With the "describe" function, we can gain insights into categorical features by uncovering their unique, top, and frequently occurring values. Additionally, for continuous features, we can extract critical statistics such as count, mean, standard deviation, minimum, maximum, and percentile values.



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

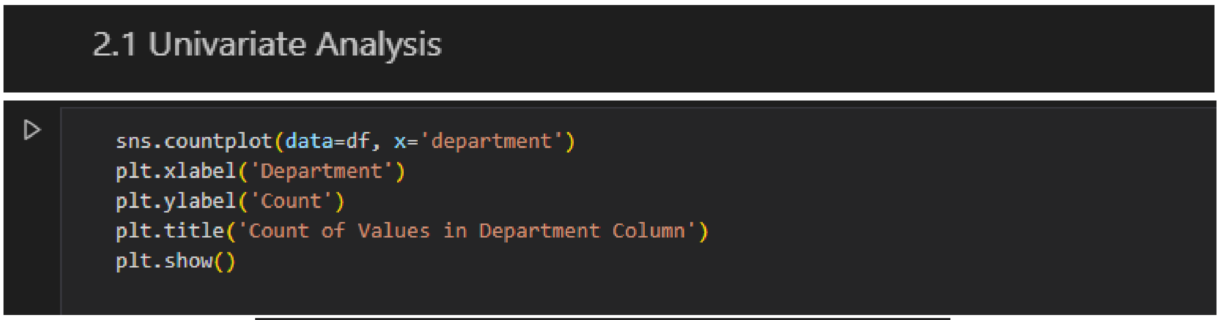
	quarter	department	day	team_number	time_allocated	unfinished_items	over_time	incentive	idle_time	idle_men	style_change	no_of_
count	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197
mean	2.252297	0.577277	2.534670	6.426901	14.508772	1190.269006	4567.460317	38.210526	0.727652	0.369256	0.150376	34
std	1.130974	0.494199	1.714538	3.463963	11.067638	1395.647280	3348.823563	160.182643	12.709094	3.268987	0.427848	22
min	1.000000	0.000000	0.000000	1.000000	2.000000	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2
25%	1.000000	0.000000	1.000000	3.000000	3.000000	970.000000	1440.000000	0.000000	0.000000	0.000000	0.000000	9
50%	2.000000	1.000000	3.000000	6.000000	15.000000	1190.000000	3960.000000	0.000000	0.000000	0.000000	0.000000	34
75%	3.000000	1.000000	4.000000	9.000000	24.000000	1190.000000	6960.000000	50.000000	0.000000	0.000000	0.000000	57
max	4.000000	1.000000	5.000000	12.000000	54.000000	23122.000000	25920.000000	3600.000000	300.000000	45.000000	2.000000	89

Activity 2: Visual analysis

Visual analysis entails the exploration and comprehension of data through the utilization of visual elements like charts, plots, and graphs. This approach enables the swift recognition of data patterns, trends, and anomalies, fostering a deeper understanding and facilitating informed decision-making.

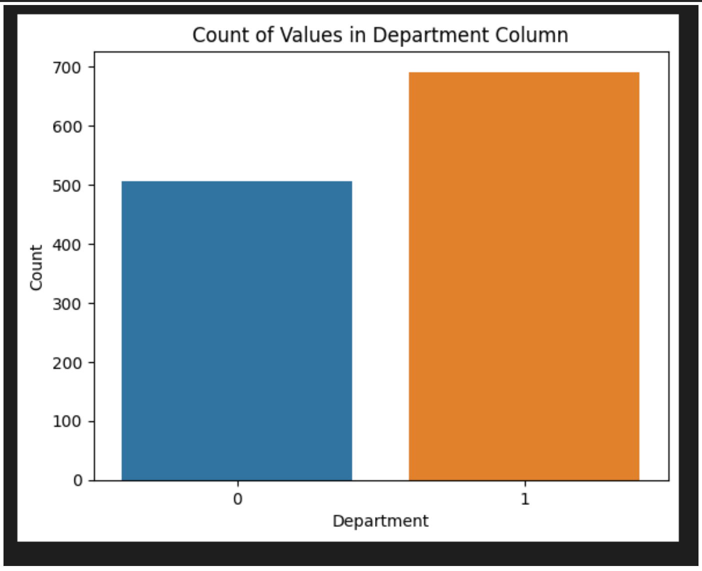
Activity 2.1: Univariate analysis

Univariate analysis is a statistical approach employed to examine an individual variable within a dataset. This analysis is centered on exploring the distribution, central tendencies, and variability of a single variable.

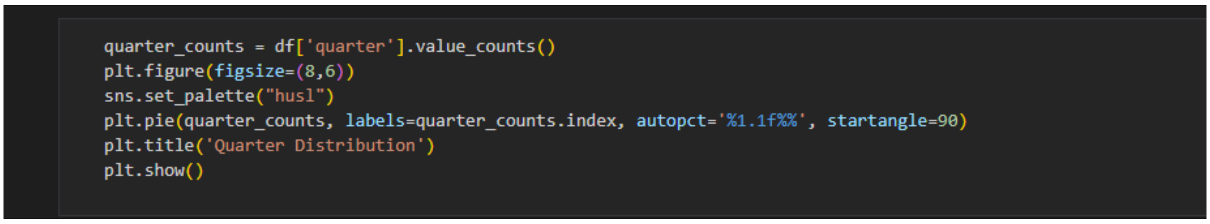


2.1 Univariate Analysis

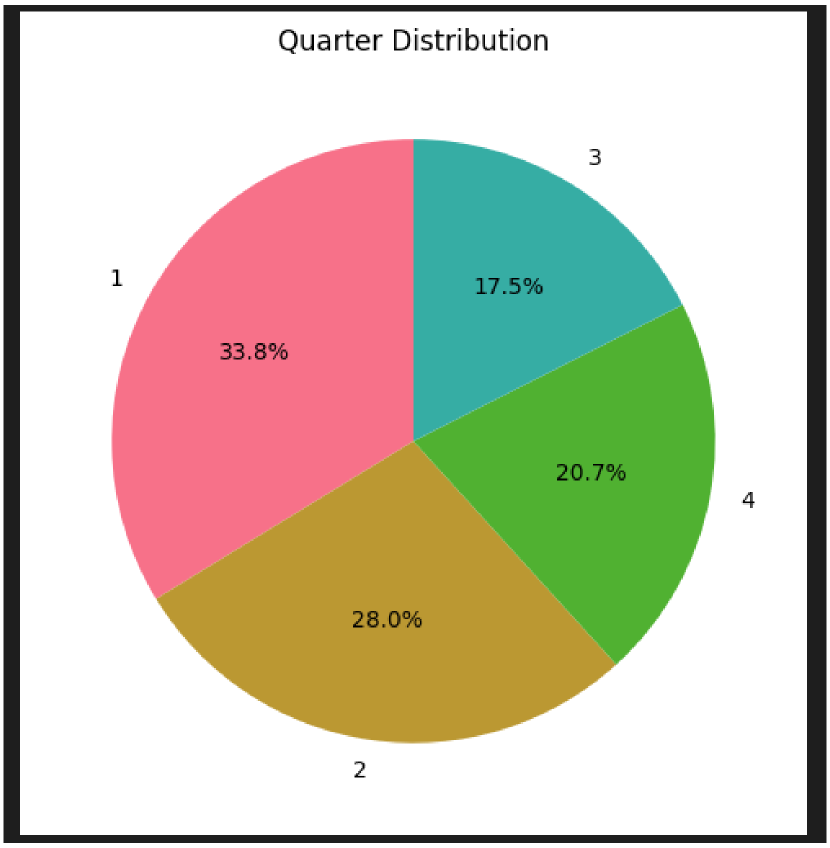
```
sns.countplot(data=df, x='department')
plt.xlabel('Department')
plt.ylabel('Count')
plt.title('Count of Values in Department Column')
plt.show()
```



This code leverages the Seaborn library to craft a bar chart. Its purpose is to visually depict the frequency of each distinct value within the "department" column found in a Pandas DataFrame. In the resulting plot, the x-axis serves to represent the unique values contained within the "department" column, while the y-axis quantifies the occurrences of each value in the dataset.
To enhance the chart's interpretability, the functions plt.xlabel(), plt.ylabel(), and plt.title() come into play, allowing for the addition of labels and a descriptive title to the plot. Finally, the visualization is brought to life by invoking plt.show(), presenting the bar chart for observation.



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

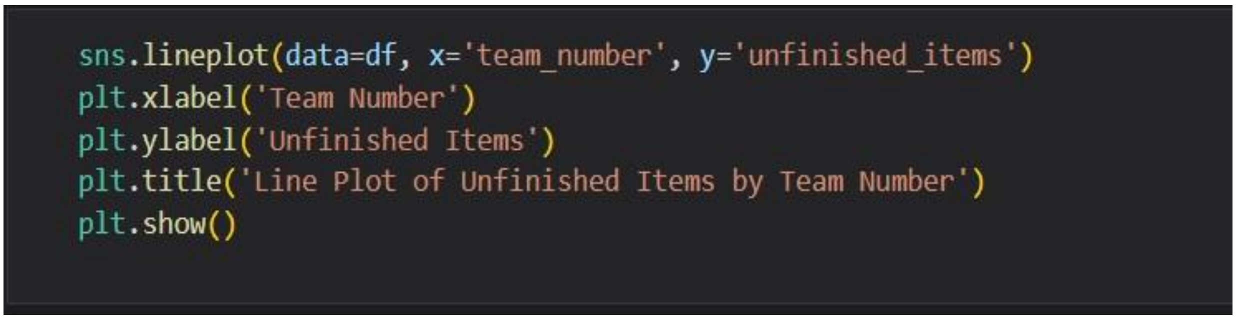


This piece of code leverages the Seaborn library to generate a pie chart, illustrating the distribution of quarters within a dataset. It employs the "value_counts()" function to tally the occurrences of each quarter in the dataset, and these counts serve as the foundation for crafting the pie chart. The chart effectively showcases the relative proportions of entries in the dataset corresponding to each quarter. The title of this pie chart is aptly labeled as "Quarter Distribution."

Activity 2.2: Bivariate analysis

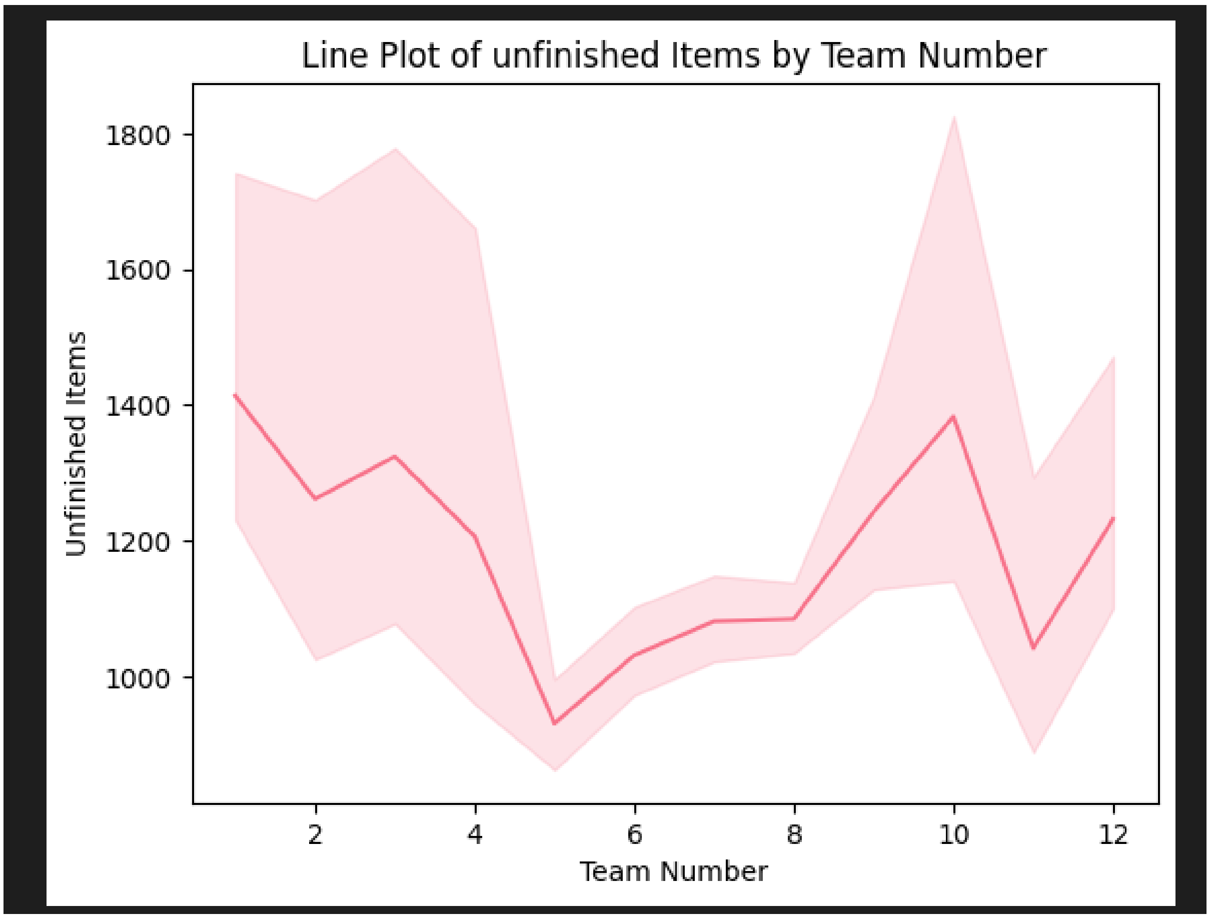
Bivariate analysis stands as a statistical methodology employed to explore the interplay between two variables within a dataset. This analytical approach is dedicated to scrutinizing how alterations in one variable are associated with shifts in another variable. It delves into uncovering relationships and dependencies between these paired variables, shedding light on their correlations and interactions.

2.2 Bivariate Analysis



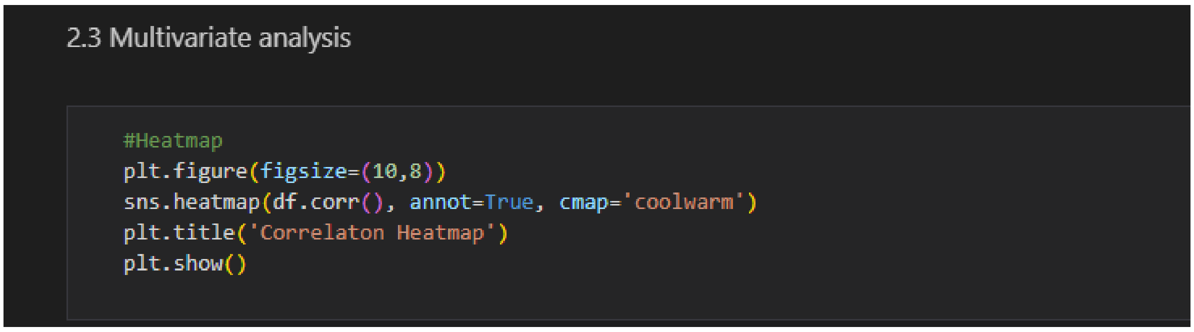
```
sns.lineplot(data=df, x='team_number', y='unfinished_items')
plt.xlabel('Team Number')
plt.ylabel('Unfinished Items')
plt.title('Line Plot of Unfinished Items by Team Number')
plt.show()
```

This code leverages the Seaborn library to produce a line plot that illustrates the correlation between team number and the quantity of incomplete items. The line plot vividly displays the fluctuations in the number of unfinished items among various teams. The x-axis portrays the team numbers, while the y-axis depicts the quantity of unfinished items. The title of this informative line plot is "Unfinished Items vs. Team Number: A Line Plot."



Activity 2.3: Multivariate analysis

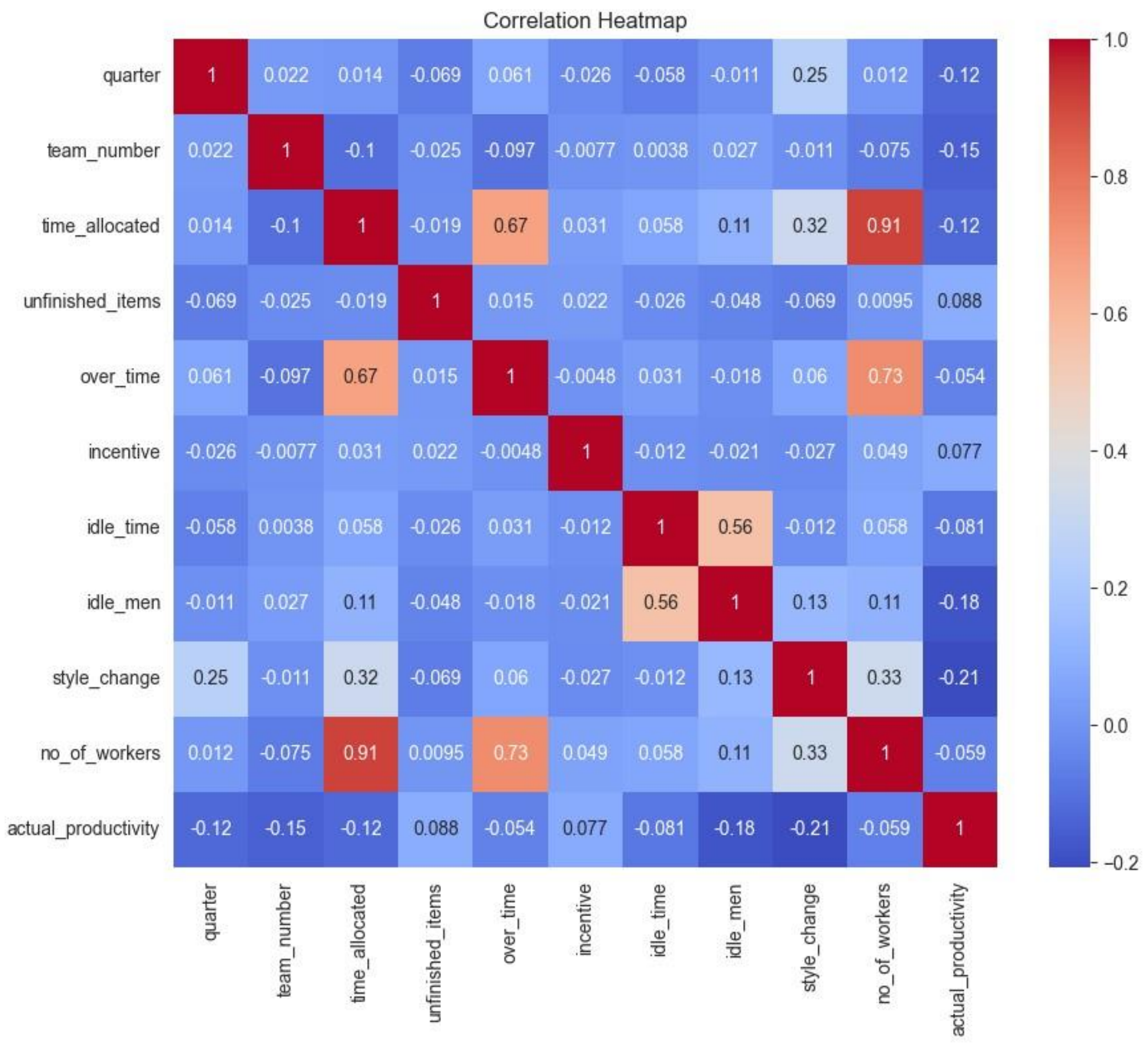
Multivariate analysis is a statistical method employed to scrutinize data encompassing more than two variables. Its primary objective is to unravel the intricate connections among multiple variables within a dataset, shedding light on their interdependencies and elucidating their collective impact on a specific outcome or phenomenon.



2.3 Multivariate analysis

```
#Heatmap
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlaton Heatmap')
plt.show()
```

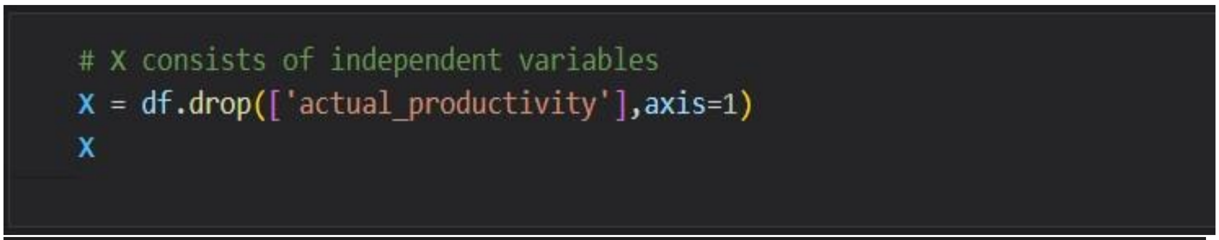
The provided code generates a heatmap that visually represents the correlations among the numerical variables within the dataset. In this visualization, the intensity of color within each square indicates the strength of the correlation between the variables. Additionally, the annotated values within the squares depict the specific correlation coefficients.



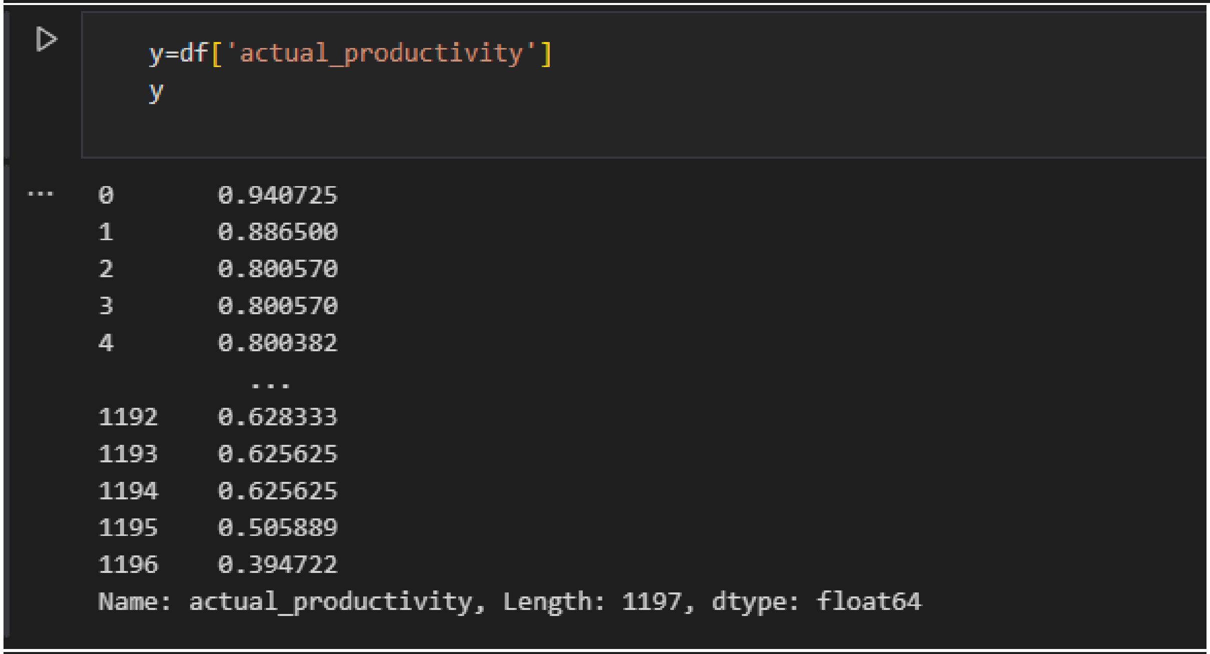
	quarter	team_number	time_allocated	unfinished_items	over_time	incentive	idle_time	idle_men	style_change	no_of_workers	actual_productivity
quarter	1	0.022	0.014	-0.069	0.061	-0.026	-0.058	-0.011	0.25	0.012	-0.12
team_number	0.022	1	-0.1	-0.025	-0.097	-0.0077	0.0038	0.027	-0.011	-0.075	-0.15
time_allocated	0.014	-0.1	1	-0.019	0.67	0.031	0.058	0.11	0.32	0.91	-0.12
unfinished_items	-0.069	-0.025	-0.019	1	0.015	0.022	-0.026	-0.048	-0.069	0.0095	0.088
over_time	0.061	-0.097	0.67	0.015	1	-0.0048	0.031	-0.018	0.06	0.73	-0.054
incentive	-0.026	-0.0077	0.031	0.022	-0.0048	1	-0.012	-0.021	-0.027	0.049	0.077
idle_time	-0.058	0.0038	0.058	-0.026	0.031	-0.012	1	0.56	-0.012	0.058	-0.081
idle_men	-0.011	0.027	0.11	-0.048	-0.018	-0.021	0.56	1	0.13	0.11	-0.18
style_change	0.25	-0.011	0.32	-0.069	0.06	-0.027	-0.012	0.13	1	0.33	-0.21
no_of_workers	0.012	-0.075	0.91	0.0095	0.73	0.049	0.058	0.11	0.33	1	-0.059
actual_productivity	-0.12	-0.15	-0.12	0.088	-0.054	0.077	-0.081	-0.18	-0.21	-0.059	1

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.

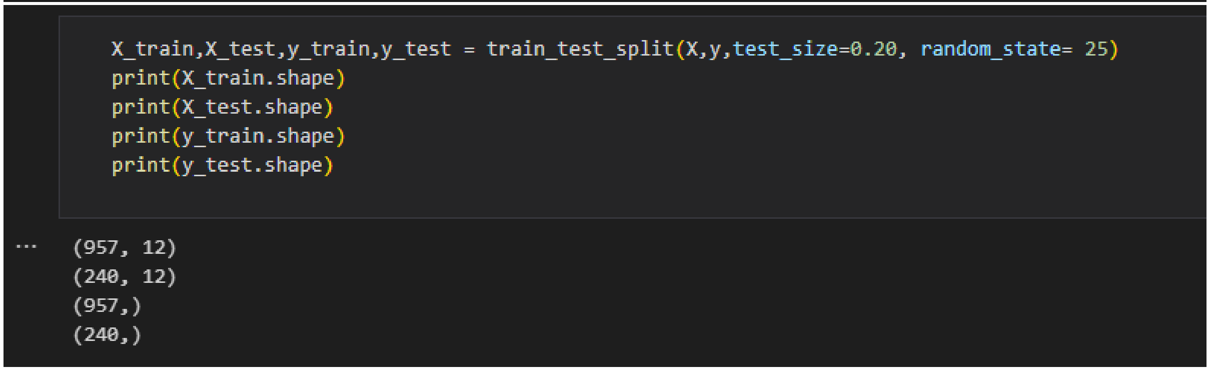


```
# X consists of independent variables
X = df.drop(['actual_productivity'],axis=1)
X
```



```
y=df['actual_productivity']
y
```

```
0       0.940725
1       0.886500
2       0.800570
3       0.800570
4       0.800382
          ...
1192    0.628333
1193    0.625625
1194    0.625625
1195    0.505889
1196    0.394722
Name: actual_productivity, Length: 1197, dtype: float64
```

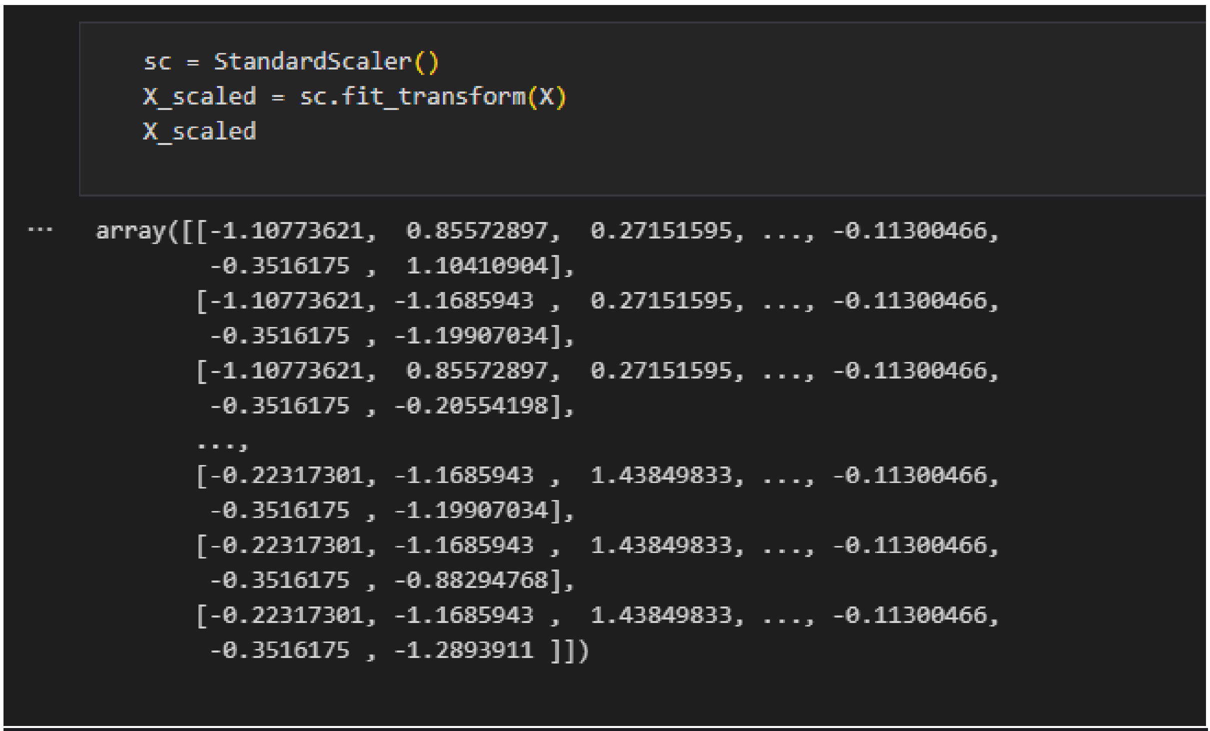


```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20, random_state= 25)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(957, 12)
(240, 12)
(957,)
(240,)
```

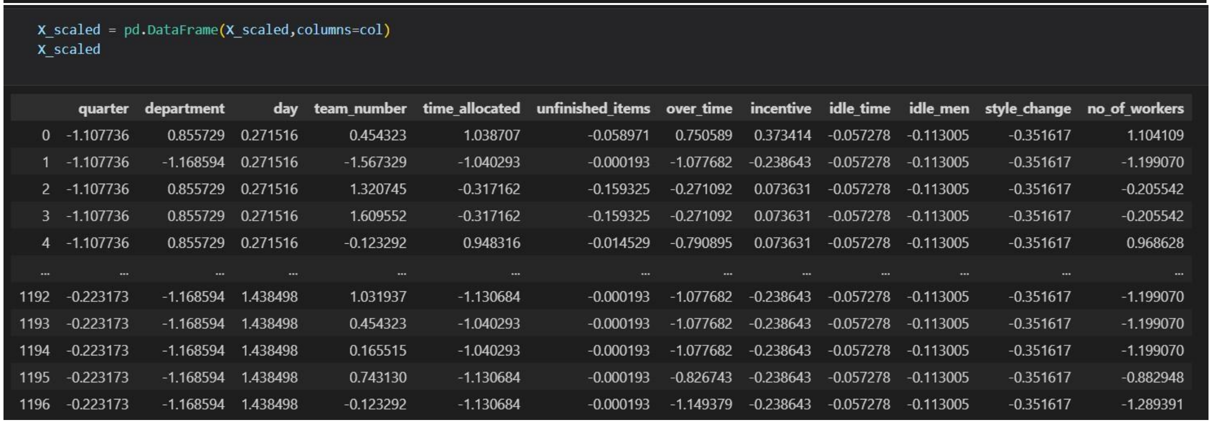
Applying Standard Scalar

Prior to commencing the training process, we implement standard scaling on the independent variables. This crucial preprocessing step serves to normalize the data, ensuring that all features are brought to a consistent scale. This is of paramount importance as certain machine learning algorithms exhibit sensitivity to the scale of input features. When features display significant disparities in scale, it can detrimentally affect the performance of the algorithms, potentially resulting in suboptimal outcomes.



```
sc = StandardScaler()
X_scaled = sc.fit_transform(X)
X_scaled
```

```
array([[-1.10773621,  0.85572897,  0.27151595, ..., -0.11300466,
        -0.3516175 ,  1.10410904],
       [-1.10773621, -1.1685943 ,  0.27151595, ..., -0.11300466,
        -0.3516175 , -1.19907034],
       [-1.10773621,  0.85572897,  0.27151595, ..., -0.11300466,
        -0.3516175 , -0.20554198],
       ...,
       [-0.22317301, -1.1685943 ,  1.43849833, ..., -0.11300466,
        -0.3516175 , -1.19907034],
       [-0.22317301, -1.1685943 ,  1.43849833, ..., -0.11300466,
        -0.3516175 , -0.88294768],
       [-0.22317301, -1.1685943 ,  1.43849833, ..., -0.11300466,
        -0.3516175 , -1.2893911 ]])
```



```
X_scaled = pd.DataFrame(X_scaled,columns=col)
X_scaled
```

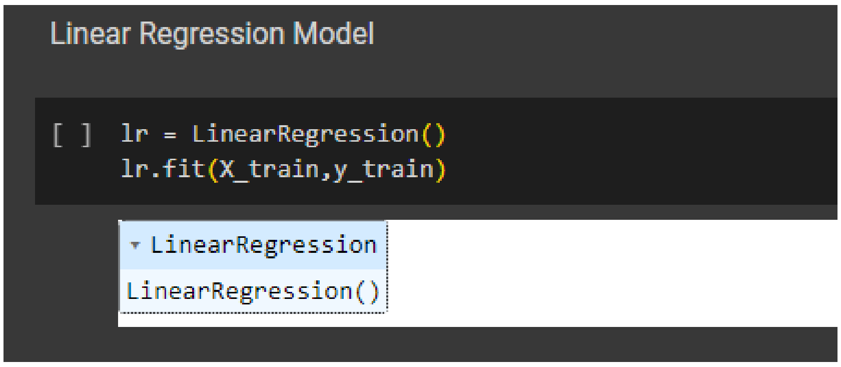
	quarter	department	day	team_number	time_allocated	unfinished_items	over_time	incentive	idle_time	idle_men	style_change	no_of_workers
0	-1.107736	0.855729	0.271516	0.454323	1.038707	-0.058971	0.750589	0.373414	-0.057278	-0.113005	-0.351617	1.104109
1	-1.107736	-1.168594	0.271516	-1.567329	-1.040293	-0.000193	-1.077682	-0.238643	-0.057278	-0.113005	-0.351617	-1.199070
2	-1.107736	0.855729	0.271516	1.320745	-0.317162	-0.159325	-0.271092	0.073631	-0.057278	-0.113005	-0.351617	-0.205542
3	-1.107736	0.855729	0.271516	1.609552	-0.317162	-0.159325	-0.271092	0.073631	-0.057278	-0.113005	-0.351617	-0.205542
4	-1.107736	0.855729	0.271516	-0.123292	0.948316	-0.014529	-0.790895	0.073631	-0.057278	-0.113005	-0.351617	0.968628
...	...	...	...	...	...	...	...	...	...	...	...	...
1192	-0.223173	-1.168594	1.438498	1.031937	-1.130684	-0.000193	-1.077682	-0.238643	-0.057278	-0.113005	-0.351617	-1.199070
1193	-0.223173	-1.168594	1.438498	0.454323	-1.040293	-0.000193	-1.077682	-0.238643	-0.057278	-0.113005	-0.351617	-1.199070
1194	-0.223173	-1.168594	1.438498	0.165515	-1.040293	-0.000193	-1.077682	-0.238643	-0.057278	-0.113005	-0.351617	-1.199070
1195	-0.223173	-1.168594	1.438498	0.743130	-1.130684	-0.000193	-0.826743	-0.238643	-0.057278	-0.113005	-0.351617	-0.882948
1196	-0.223173	-1.168594	1.438498	-0.123292	-1.130684	-0.000193	-1.149379	-0.238643	-0.057278	-0.113005	-0.351617	-1.289391

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



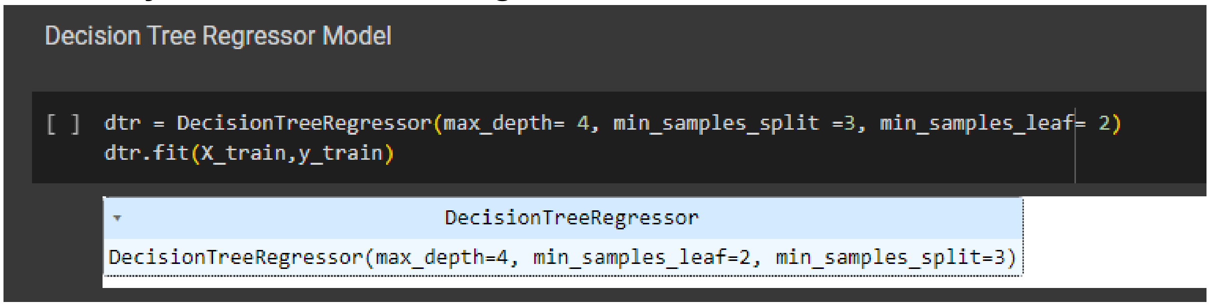
Linear Regression Model

```
[ ] lr = LinearRegression()
    lr.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

This code exemplifies linear regression modeling utilizing the scikit-learn library.

Activity 1.2: Decision Tree Regressor Model

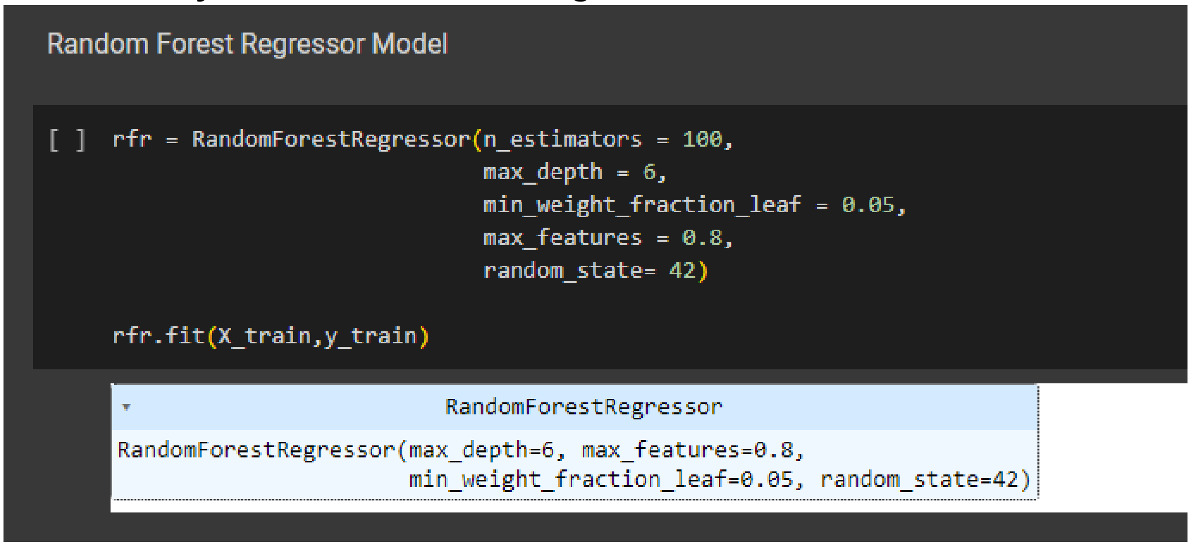


Decision Tree Regressor Model

```
[ ] dtr = DecisionTreeRegressor(max_depth= 4, min_samples_split =3, min_samples_leaf= 2)
    dtr.fit(X_train,y_train)
```

```
▾                              DecisionTreeRegressor
DecisionTreeRegressor(max_depth=4, min_samples_leaf=2, min_samples_split=3)
```

Activity 1.3: Random Forest Regressor Model



Random Forest Regressor Model

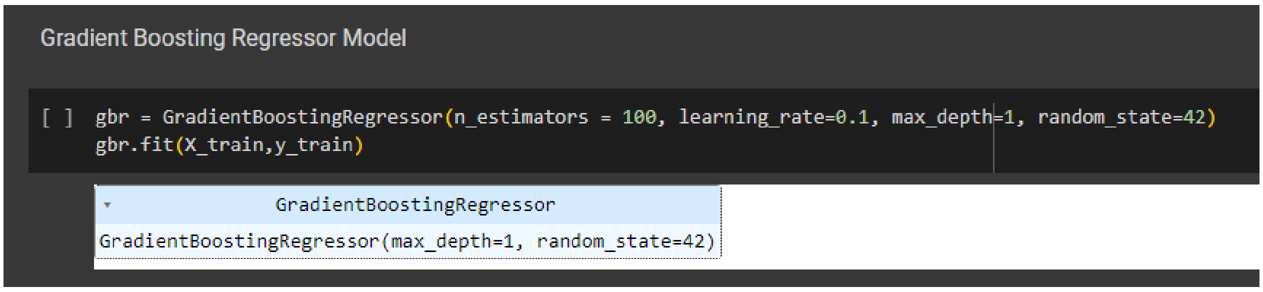
```
[ ] rfr = RandomForestRegressor(n_estimators = 100,
                                max_depth = 6,
                                min_weight_fraction_leaf = 0.05,
                                max_features = 0.8,
                                random_state= 42)

    rfr.fit(X_train,y_train)
```

```
▾                          RandomForestRegressor
RandomForestRegressor(max_depth=6, max_features=0.8,
                      min_weight_fraction_leaf=0.05, random_state=42)
```

The provided code instantiates a RandomForestRegressor class with predefined hyperparameters. These hyperparameters govern critical aspects of the random forest, including the number of trees, maximum tree depth, minimum weight fraction at leaf nodes, the maximum features considered when splitting nodes, and a random state for result reproducibility. Subsequently, the "rfr" object is trained using the "fit" method on the training data X_train and y_train.

Activity 1.4: Gradient Boosting Regressor Model

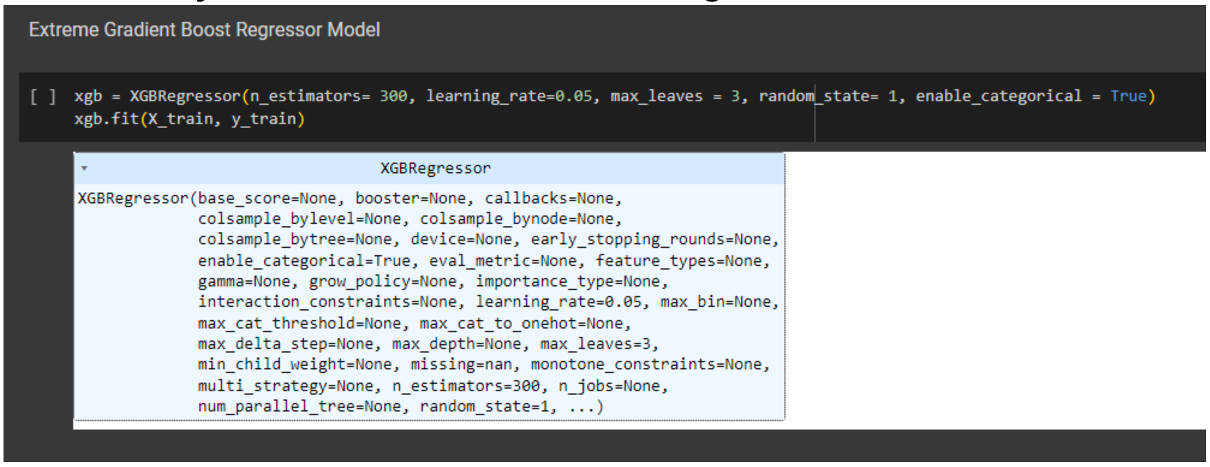


Gradient Boosting Regressor Model

```
[ ] 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)
```

Activity 1.5: Extreme Gradient Boost Regressor Model

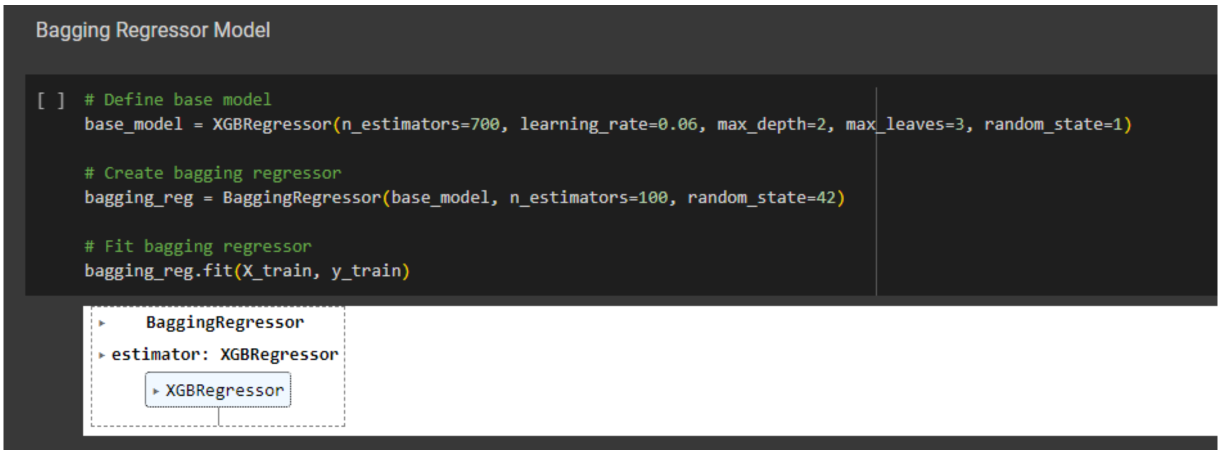


Extreme Gradient Boost Regressor Model

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

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=True, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=3,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=300, n_jobs=None,
             num_parallel_tree=None, random_state=1, ...)
```

Activity 1.6: Bagging Regressor Model



Bagging Regressor Model

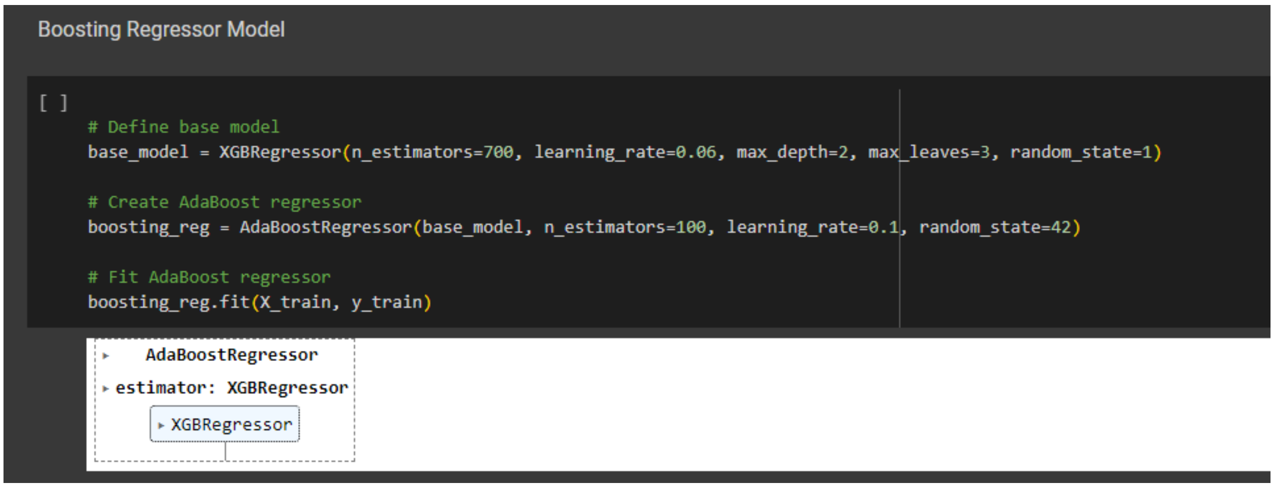
```
[ ] # Define base model
    base_model = XGBRegressor(n_estimators=700, learning_rate=0.06, max_depth=2, max_leaves=3, random_state=1)

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

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

```
BaggingRegressor
estimator: XGBRegressor
  XGBRegressor
```

Activity 1.7: Boosting Regressor Model



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)
```

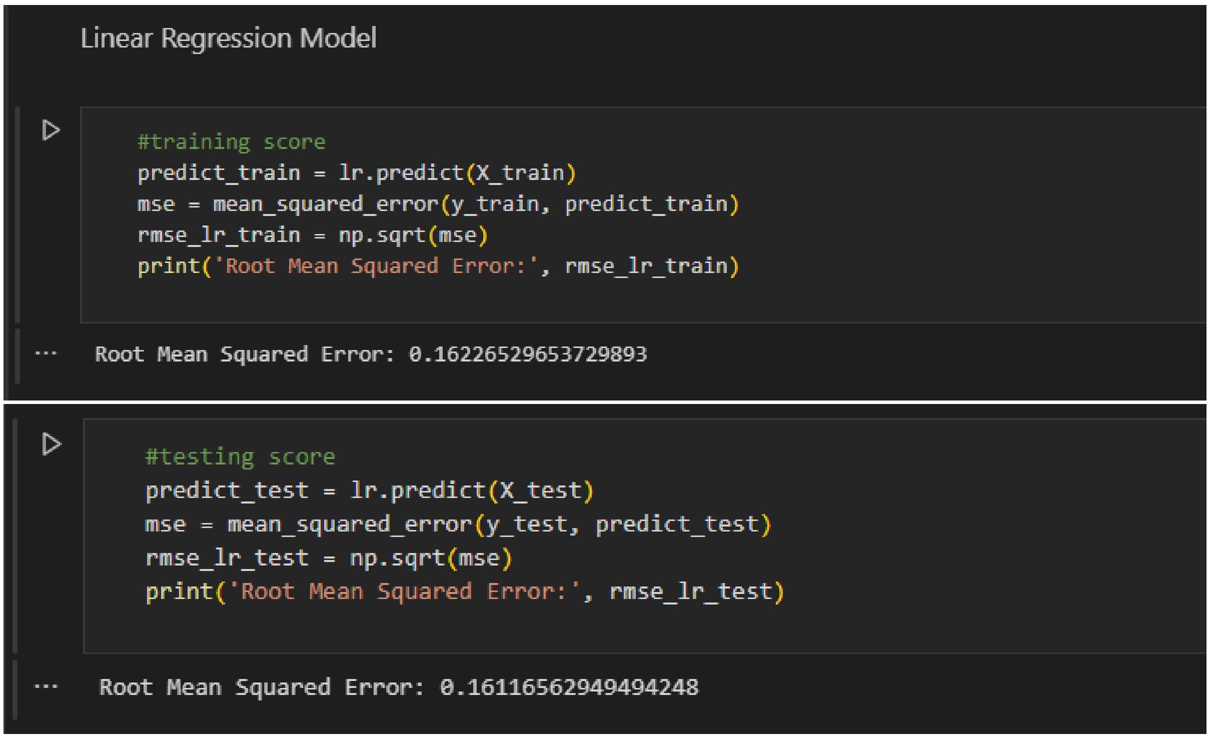
```
▸    AdaBoostRegressor
▸ estimator: XGBRegressor
      ▸ XGBRegressor
```

Milestone 5: Performance Testing

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

To assess the model's performance on both the test and training datasets, we employ the Root Mean Square Error (RMSE) score. This score serves as a metric that quantifies the average disparity between the model's predictions and the actual values.

Activity 1.1: Linear Regression Model



Linear Regression Model

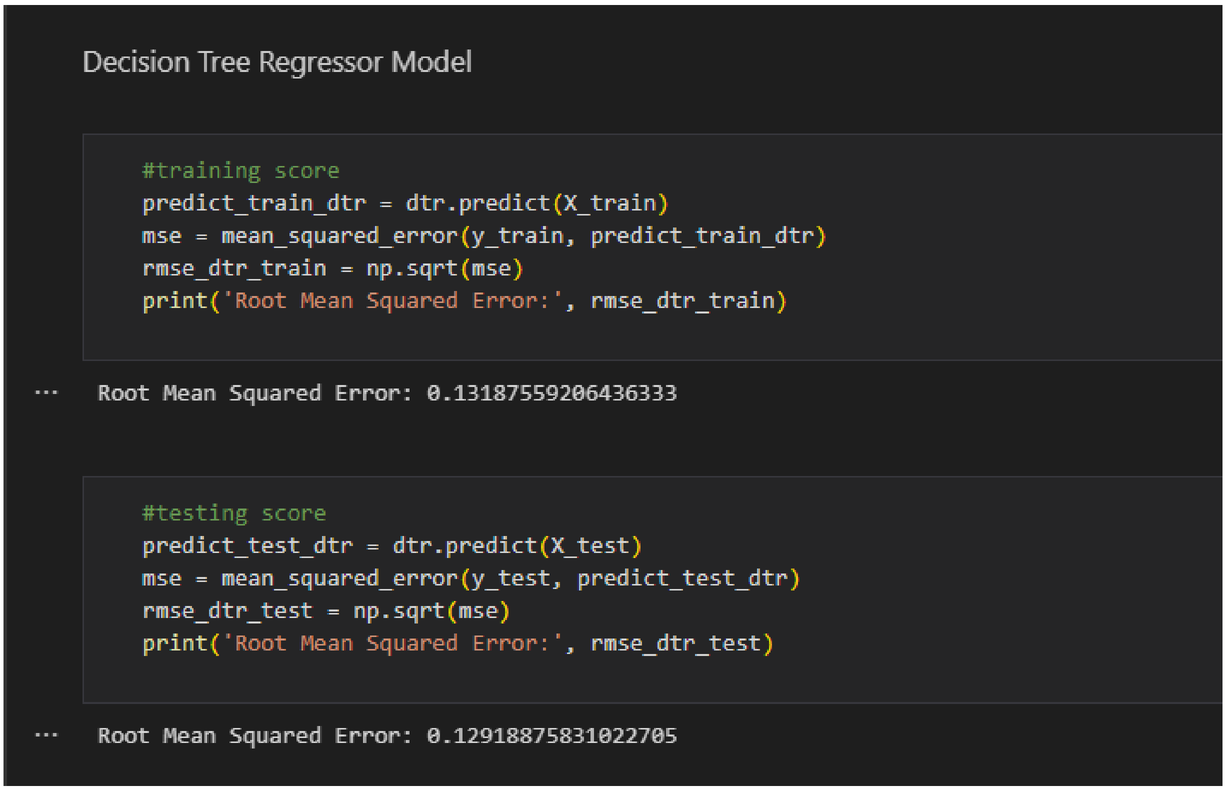
```
#training score
predict_train = lr.predict(X_train)
mse = mean_squared_error(y_train, predict_train)
rmse_lr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_train)
```

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

```
#testing score
predict_test = lr.predict(X_test)
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: 0.16116562949494248
```

Activity 1.2: Decision Tree Regressor Model



Decision Tree Regressor Model

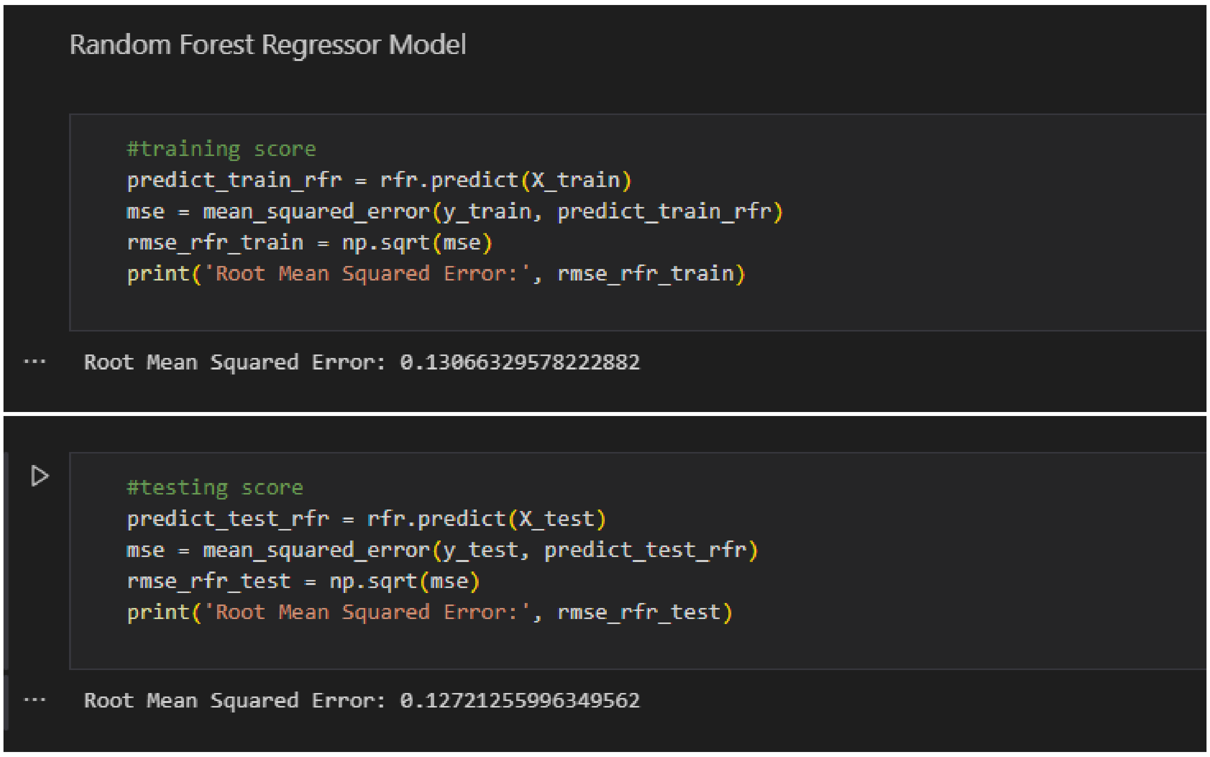
```
#training score
predict_train_dtr = dtr.predict(X_train)
mse = mean_squared_error(y_train, predict_train_dtr)
rmse_dtr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_dtr_train)
```

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

```
#testing score
predict_test_dtr = dtr.predict(X_test)
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



Random Forest Regressor Model

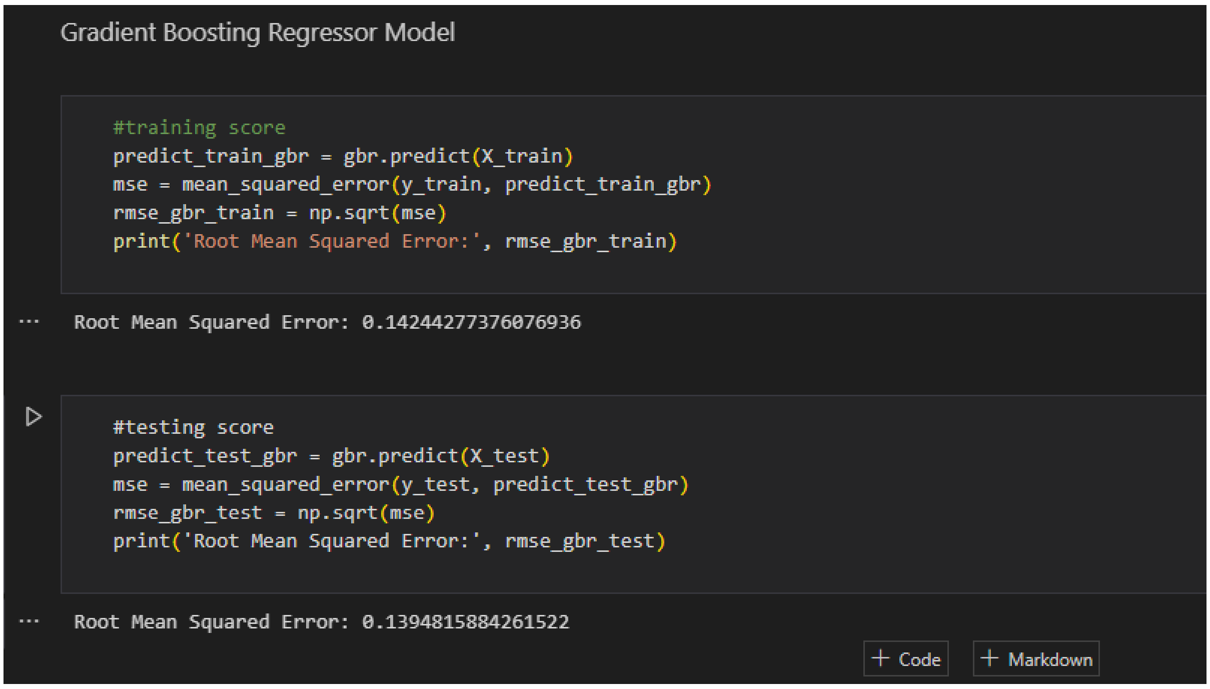
```
#training score
predict_train_rfr = rfr.predict(X_train)
mse = mean_squared_error(y_train, predict_train_rfr)
rmse_rfr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_rfr_train)
```

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

```
#testing score
predict_test_rfr = rfr.predict(X_test)
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



Gradient Boosting Regressor Model

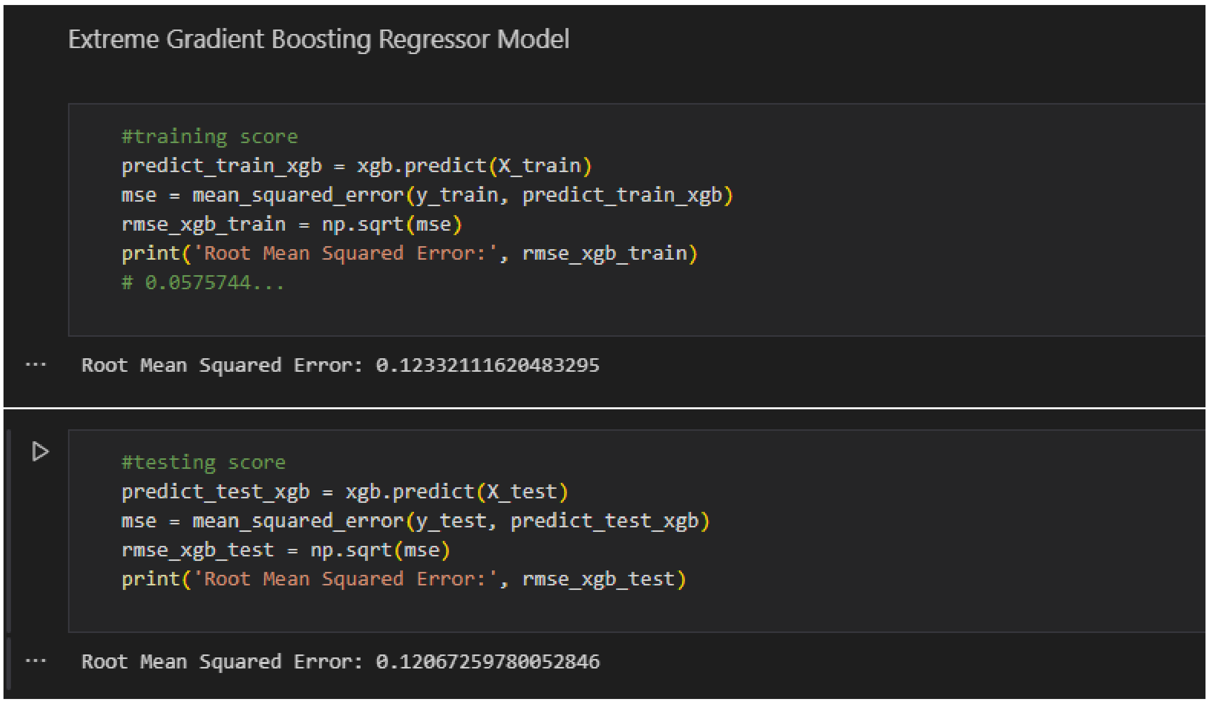
```
#training score
predict_train_gbr = gbr.predict(X_train)
mse = mean_squared_error(y_train, predict_train_gbr)
rmse_gbr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_train)
```

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

```
#testing score
predict_test_gbr = gbr.predict(X_test)
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



Extreme Gradient Boosting Regressor Model

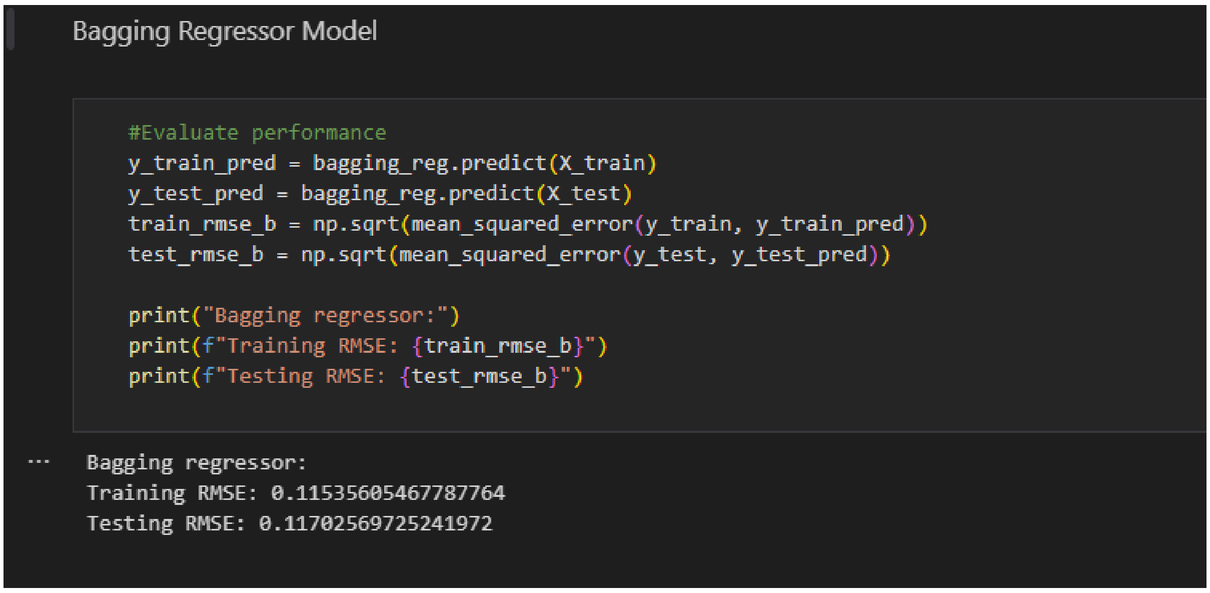
```
#training score
predict_train_xgb = xgb.predict(X_train)
mse = mean_squared_error(y_train, predict_train_xgb)
rmse_xgb_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_xgb_train)
# 0.0575744...
```

```
Root Mean Squared Error: 0.12332111620483295
```

```
#testing score
predict_test_xgb = xgb.predict(X_test)
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: 0.12067259780052846
```

Activity 1.6: Bagging Regressor Model



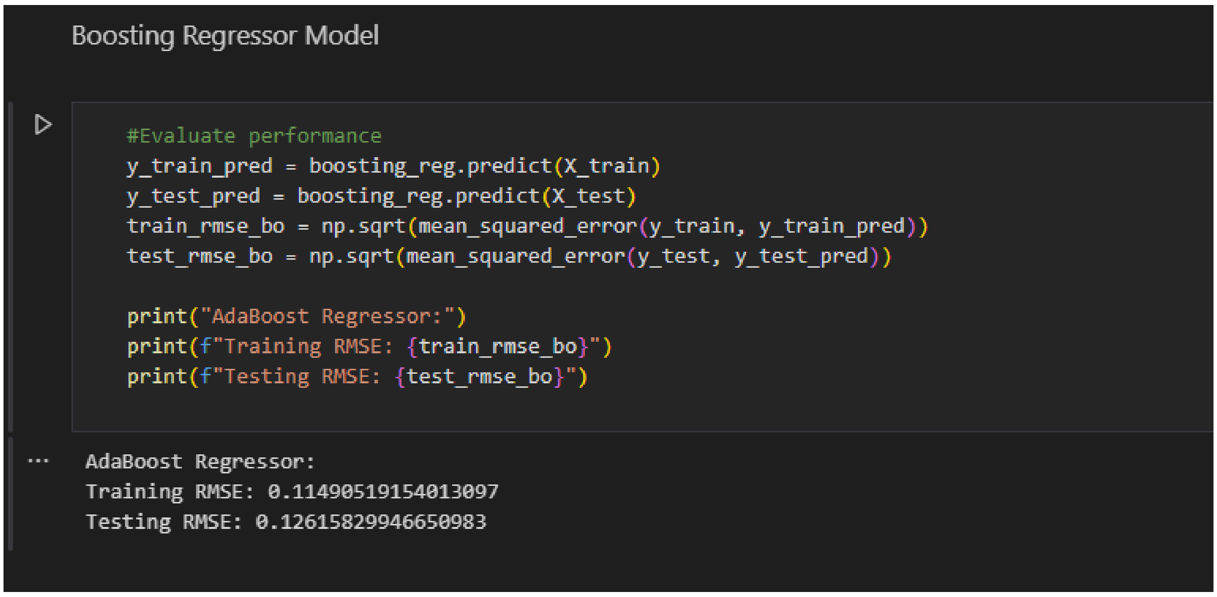
Bagging Regressor Model

```python
#Evaluate performance
y_train_pred = bagging_reg.predict(X_train)
y_test_pred = bagging_reg.predict(X_test)
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: 0.11535605467787764
Testing RMSE: 0.11702569725241972
```

Activity 1.7: Boosting Regressor Model



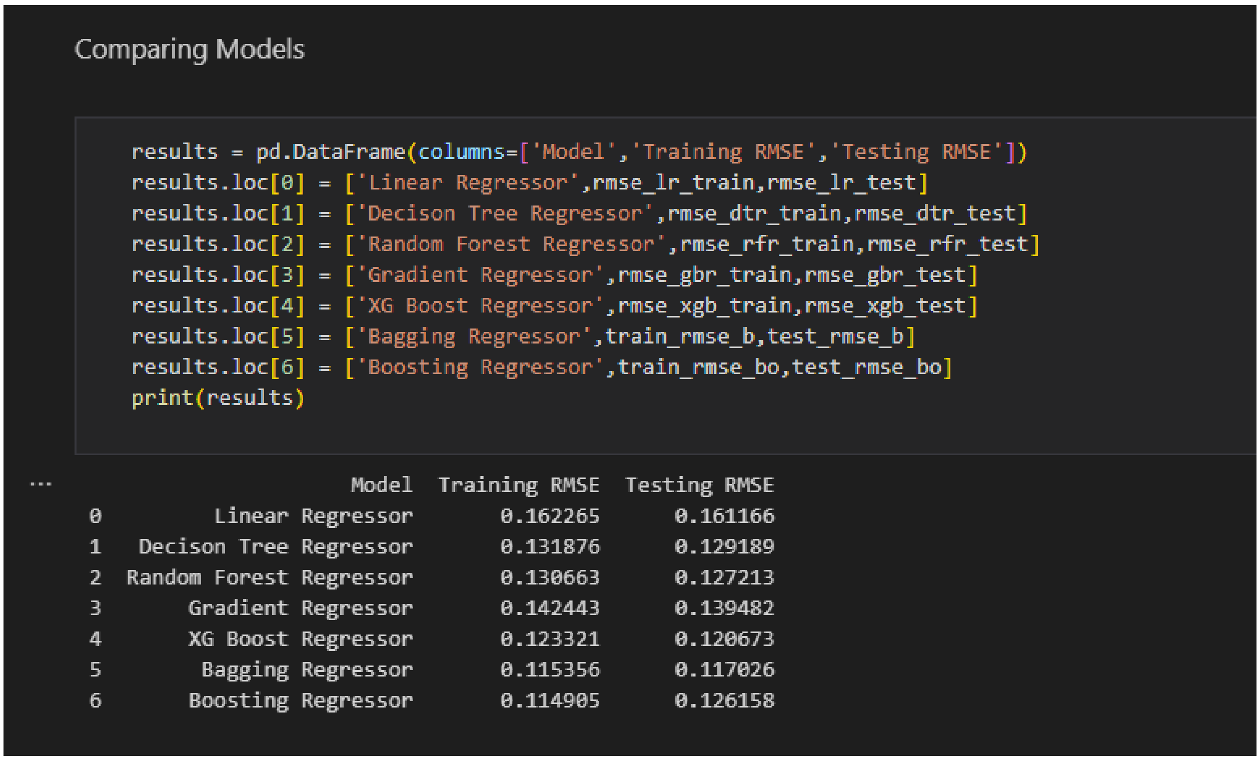
Boosting Regressor Model

```python
#Evaluate performance
y_train_pred = boosting_reg.predict(X_train)
y_test_pred = boosting_reg.predict(X_test)
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: 0.11490519154013097
Testing RMSE: 0.12615829946650983
```

Activity 2: Comparing models

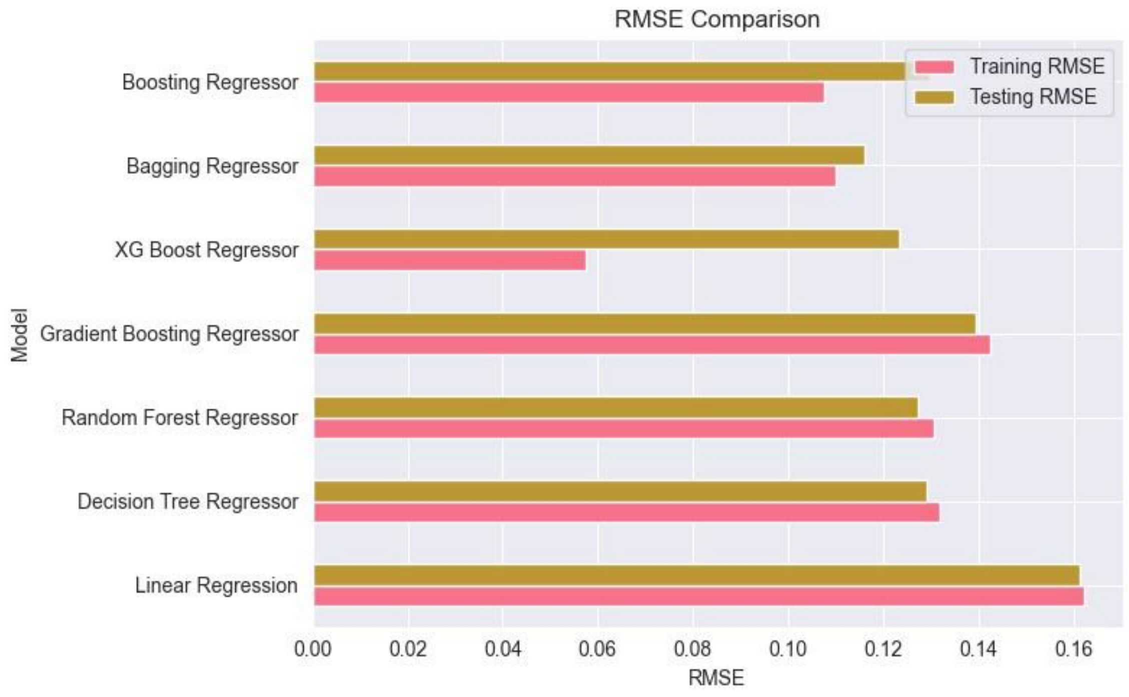


Comparing Models

```
results = pd.DataFrame(columns=['Model','Training RMSE','Testing RMSE'])
results.loc[0] = ['Linear Regressor',rmse_lr_train,rmse_lr_test]
results.loc[1] = ['Decison Tree Regressor',rmse_dtr_train,rmse_dtr_test]
results.loc[2] = ['Random Forest Regressor',rmse_rfr_train,rmse_rfr_test]
results.loc[3] = ['Gradient Regressor',rmse_gbr_train,rmse_gbr_test]
results.loc[4] = ['XG Boost Regressor',rmse_xgb_train,rmse_xgb_test]
results.loc[5] = ['Bagging Regressor',train_rmse_b,test_rmse_b]
results.loc[6] = ['Boosting Regressor',train_rmse_bo,test_rmse_bo]
print(results)
```

```
                     Model  Training RMSE  Testing RMSE
0         Linear Regressor       0.162265      0.161166
1   Decison Tree Regressor       0.131876      0.129189
2  Random Forest Regressor       0.130663      0.127213
3       Gradient Regressor       0.142443      0.139482
4       XG Boost Regressor       0.123321      0.120673
5        Bagging Regressor       0.115356      0.117026
6       Boosting Regressor       0.114905      0.126158
```

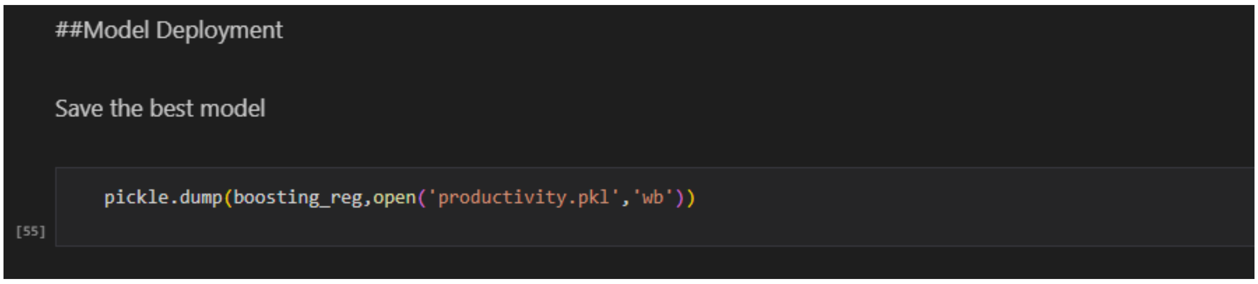
This code creates a Pandas Data Frame named "results" that contains the model names and root mean squared errors (RMSE) for both the training and testing data for each of the seven regression models: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG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



##Model Deployment

Save the best model

```
pickle.dump(boosting_reg,open('productivity.pkl','wb'))
```

[55]

In this code snippet, we employ the "pickle" library in Python to preserve the trained Boosting Regressor model, denoted as "boosting_reg," under the file name "productivity.pkl." We utilize the "dump" method provided by the pickle library to serialize the model object, ensuring it can be retrieved and reused in the future. The use of "wb" as a parameter signifies that the file is to be opened in binary mode for writing data.

Activity 2: Integrate with Web Framework

This section involves the creation of a web application that facilitates the seamless integration of our machine learning model, which we've meticulously developed and trained.
Within this web application, we furnish users with a user-friendly interface for inputting values required for predictions. These user-provided values are then fed into the stored model, and the resultant predictions are displayed on the user interface.
This activity encompasses several key tasks, including:

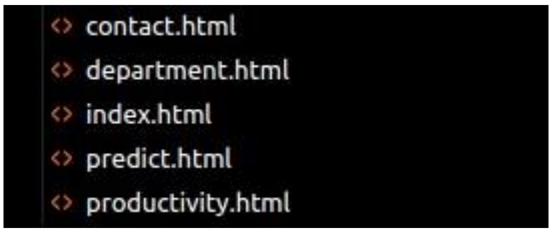
- Building HTML pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project, we've crafted three HTML files:

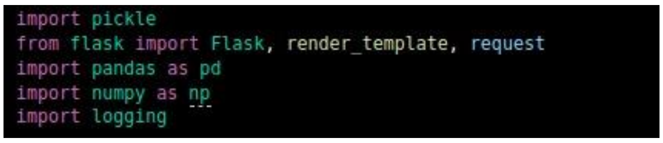
- index.html
- predict.html
- productivity.html

These HTML files are meticulously organized within the "templates" folder, setting the foundation for the web application's user interface.

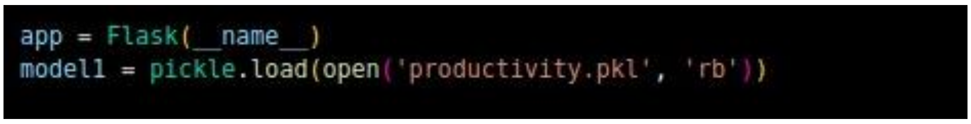

```
contact.html
department.html
index.html
predict.html
productivity.html
```

Activity 2.2: Build Python code:

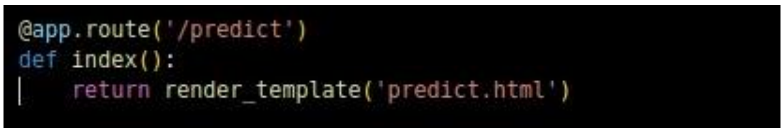
Importing the libraries


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

This code begins by retrieving the pre-trained Boosting Regressor model stored in the "productivity.pkl" file. This is achieved through the use of the "pickle.load()" method, with the "rb" parameter signifying that the file should be opened in binary mode to read its contents.
Following the model retrieval, the code proceeds to instantiate a new Flask web application object, denoted as "app," employing the Flask constructor. The "name" argument designates the current module as the application's name, as specified for Flask to use.

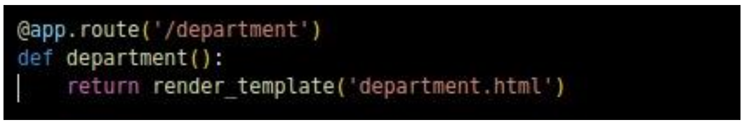

```
app = Flask(__name__)
model1 = pickle.load(open('productivity.pkl', 'rb'))
```

This piece of code establishes a new route within the Flask web application, employing the "@app.route()" decorator. In this instance, the designated route is the root path, denoted as "/", which serves as the default when visiting the website.
The function "home()" is linked to this route. Consequently, when a user navigates to the website's root path, this function is automatically invoked.
To render the user interface, the "render_template()" method is employed, tasked with presenting the contents of an HTML template entitled "index.html." This "index.html" file, in essence, serves as the homepage of the web application.

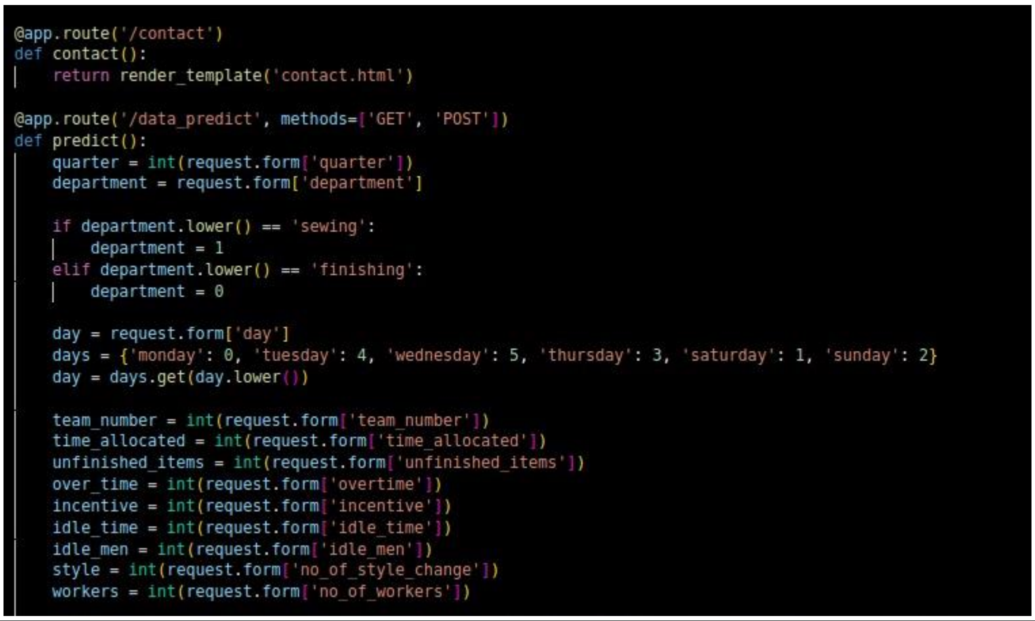

```
@app.route('/predict')
def index():
    return render_template('predict.html')
```

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('/department')
def department():
    return render_template('department.html')
```

This code establishes an additional route within the Flask web application, leveraging the "@app.route()" decorator. The route, denoted as "/data_predict," accommodates both GET and POST requests. The function "predict()" is intricately linked to this route for handling incoming data.
The input data originates from an HTML form and encompasses a range of attributes, including the quarter, department, day of the week, team number, time allocation, unfinished tasks, overtime, incentives, idle periods, idle workforce, style variations, and the count of workers.
To facilitate the model's comprehension, the code transforms some of the input data. For instance, the department input, represented as a string, is converted into a binary format (1 for sewing, 0 for finishing). Similarly, the day of the week input is translated into a numerical value ranging from 0 to 6.
Subsequently, the model is invoked to generate a prediction based on the processed input data. The prediction is then rounded to four decimal places and scaled by a factor of 100, effectively converting it into a percentage.
The resulting prediction value is transmitted to the HTML template labeled 'productivity.html,' where it is dynamically presented as a text message. This message serves as a clear and informative notification to the user, offering insights into the anticipated productivity levels derived from the input data.



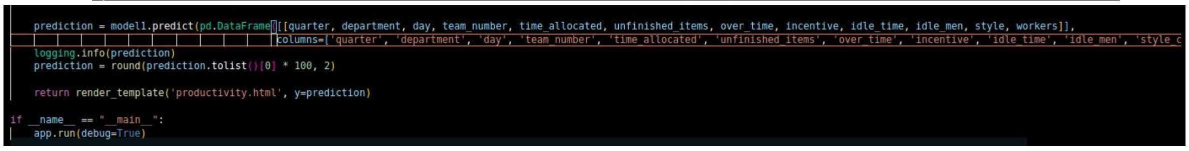
```python
@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/data_predict', methods=['GET', 'POST'])
def predict():
    quarter = int(request.form['quarter'])
    department = request.form['department']

    if department.lower() == 'sewing':
        department = 1
    elif department.lower() == 'finishing':
        department = 0

    day = request.form['day']
    days = {'monday': 0, 'tuesday': 4, 'wednesday': 5, 'thursday': 3, 'saturday': 1, 'sunday': 2}
    day = days.get(day.lower())

    team_number = int(request.form['team_number'])
    time_allocated = int(request.form['time_allocated'])
    unfinished_items = int(request.form['unfinished_items'])
    over_time = int(request.form['overtime'])
    incentive = int(request.form['incentive'])
    idle_time = int(request.form['idle_time'])
    idle_men = int(request.form['idle_men'])
    style = int(request.form['no_of_style_change'])
    workers = int(request.form['no_of_workers'])
```



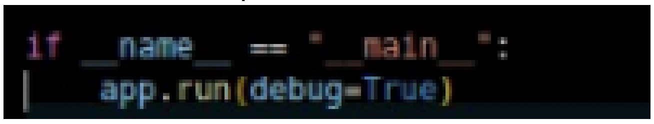
```python
    prediction = modell.predict(pd.DataFrame([[quarter, department, day, team_number, time_allocated, unfinished_items, over_time, incentive, idle_time, idle_men, style, workers]],
                                             columns=['quarter', 'department', 'day', 'team_number', 'time_allocated', 'unfinished_items', 'over_time', 'incentive', 'idle_time', 'idle_men', 'style_c
    logging.info(prediction)
    prediction = round(prediction.tolist()[0] * 100, 2)

    return render_template('productivity.html', y=prediction)

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

Main Function:

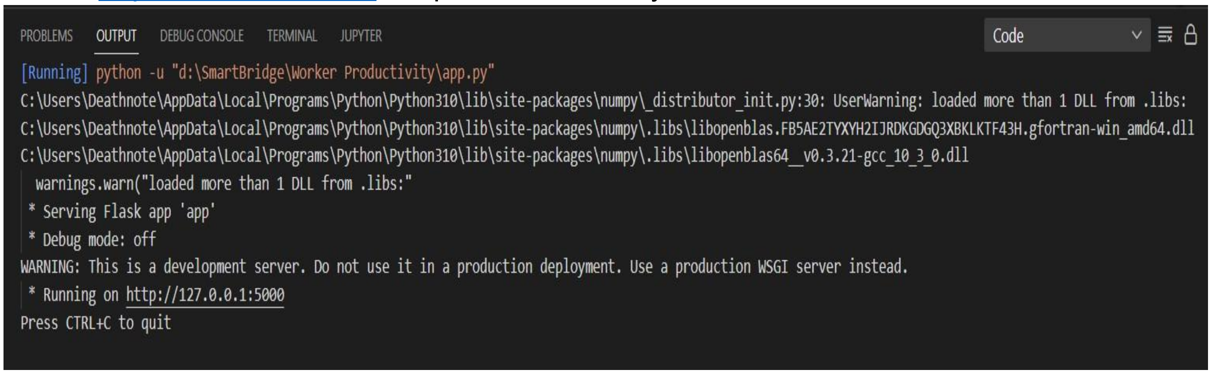
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.



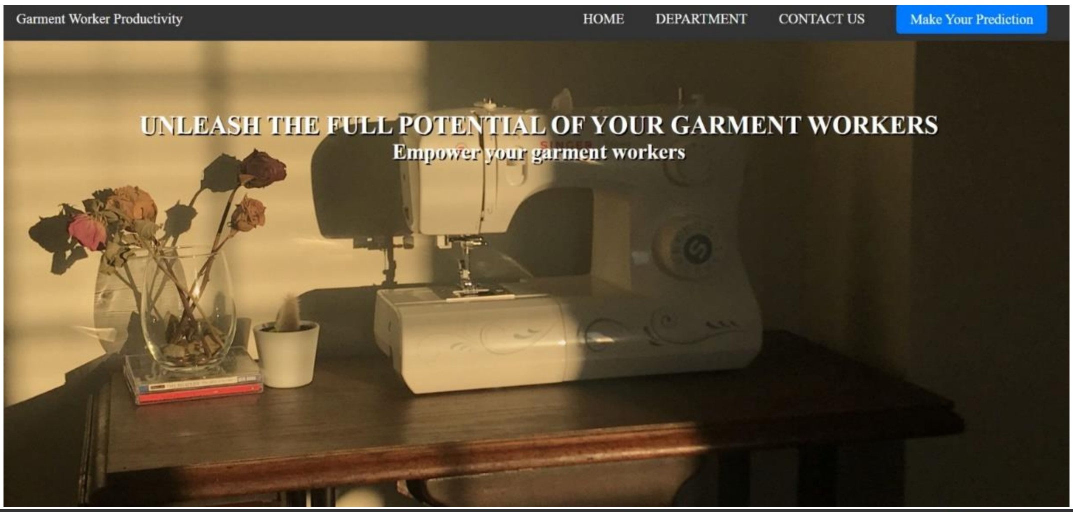
```python
if __name__ == "__main__":
    app.run(debug=True)
```

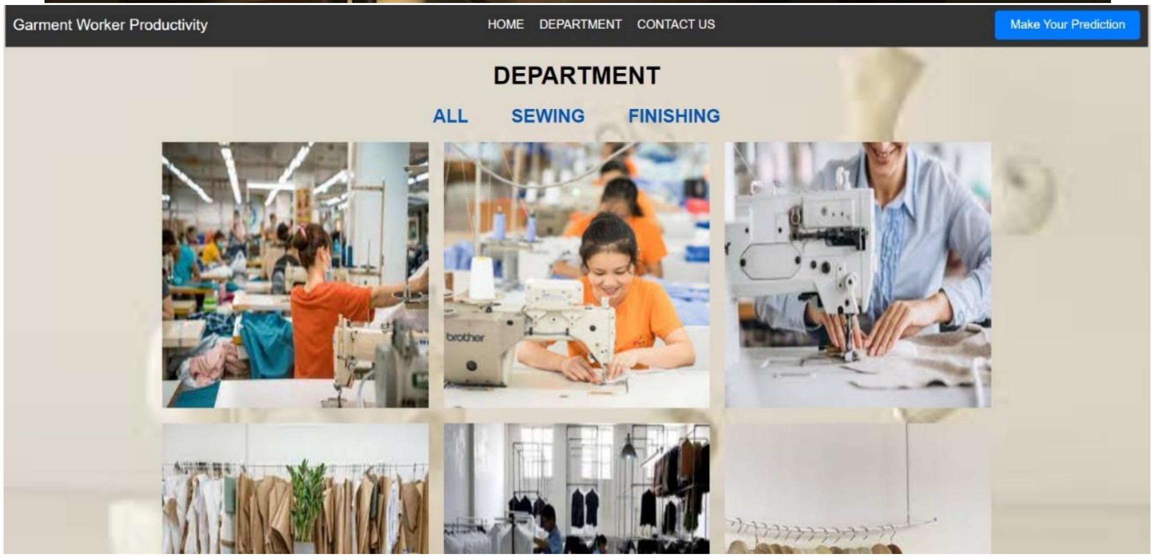
Activity 2.3: Run the web application

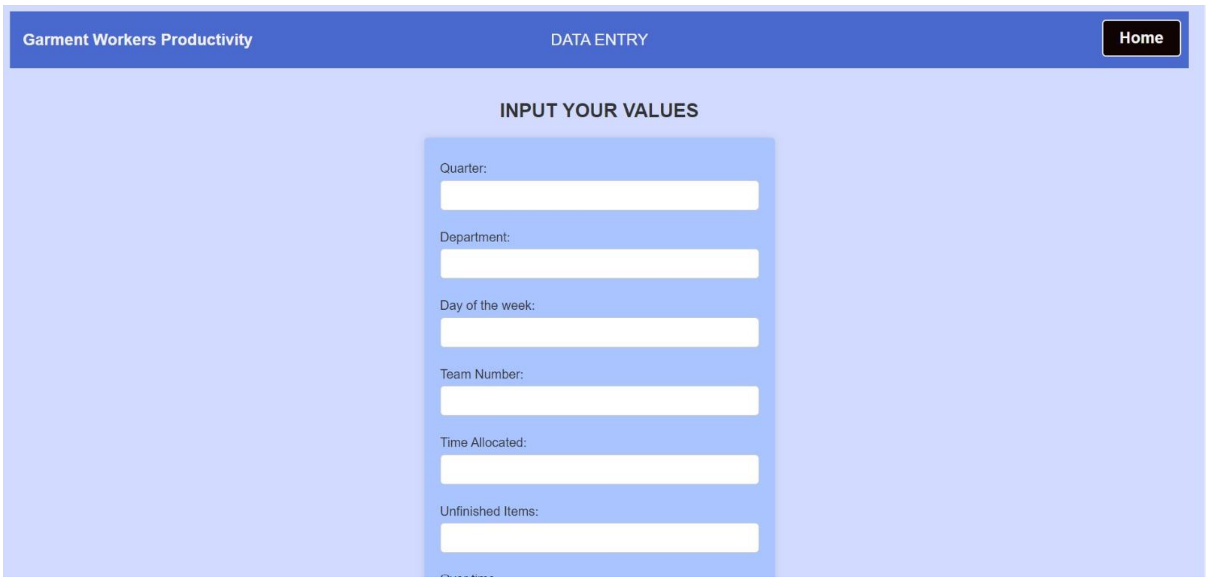
When you run the "main.py" file this window will open in the output terminal. Copy the http://127.0.0.1:5000 and paste this link in your browser.



```
[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
```


Garment Worker Productivity
HOME
DEPARTMENT
CONTACT US
Make Your Prediction
UNLEASH THE FULL POTENTIAL OF YOUR GARMENT WORKERS
Empower your garment workers





Garment Workers Productivity

DATA ENTRY

Home

INPUT YOUR VALUES

Quarter:

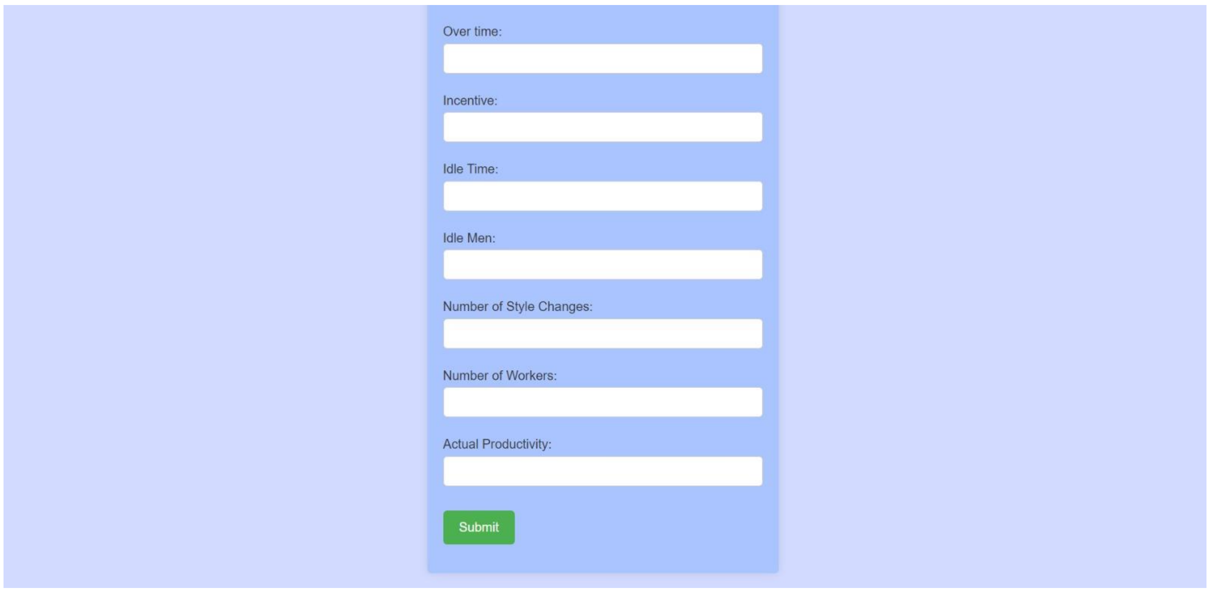
Department:

Day of the week:

Team Number:

Time Allocated:

Unfinished Items:



Over time:

Incentive:

Idle Time:

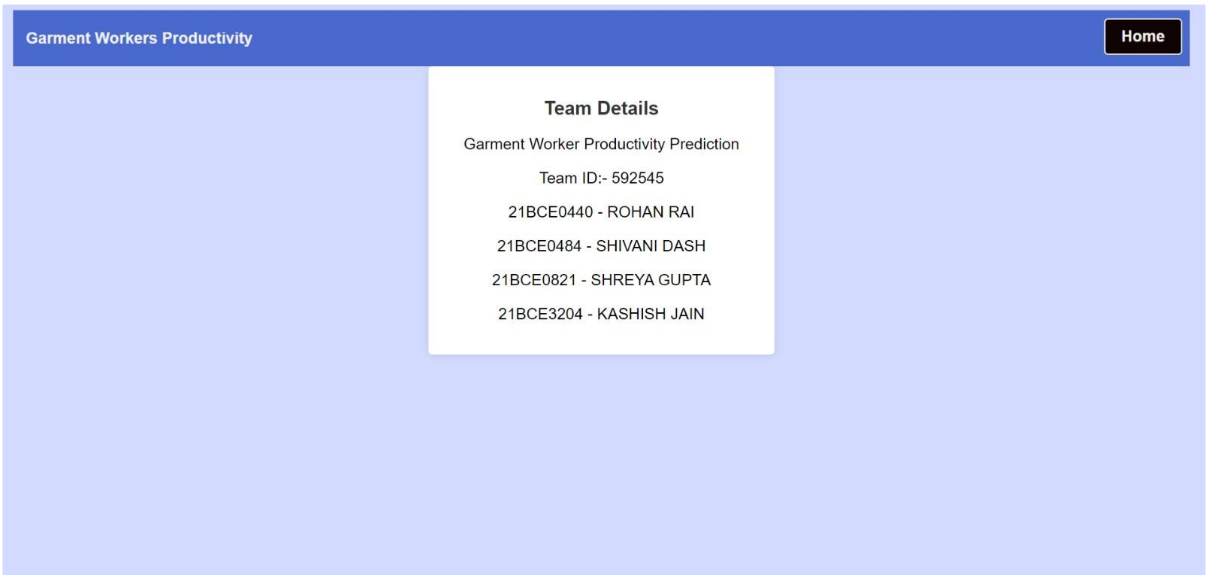
Idle Men:

Number of Style Changes:

Number of Workers:

Actual Productivity:

Submit



Team Details

Garment Worker Productivity Prediction

Team ID:- 592545

21BCE0440 - ROHAN RAI

21BCE0484 - SHIVANI DASH

21BCE0821 - SHREYA GUPTA

21BCE3204 - KASHISH JAIN

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution.

https://drive.google.com/file/d/1Xu_QHJUutN69YgXRJAnLxqhXoJgIUHop/view?usp=sharing

Activity 2:- Project Documentation-Step by step project development procedure.

https://docs.google.com/document/d/1HFCORNg0Ps6rzIvMur8GLAxfq46H3kP6/edit?usp=sharing&ouid=106396798939514752462&rtpof=true&sd=true