Optimizing Garment Manufacturing: Predicting Employee Productivity with Machine Learning

Name: Erning Xu

Institution: Brown University

Date: 2024/10/25

GitHub: https://github.com/ErningXu/Midterm.git

Introduction

Problem Description

- **Goal:** predicting the productivity of garment employees based on various factors
- **Regression problem**
- **Target variable**: employee productivity

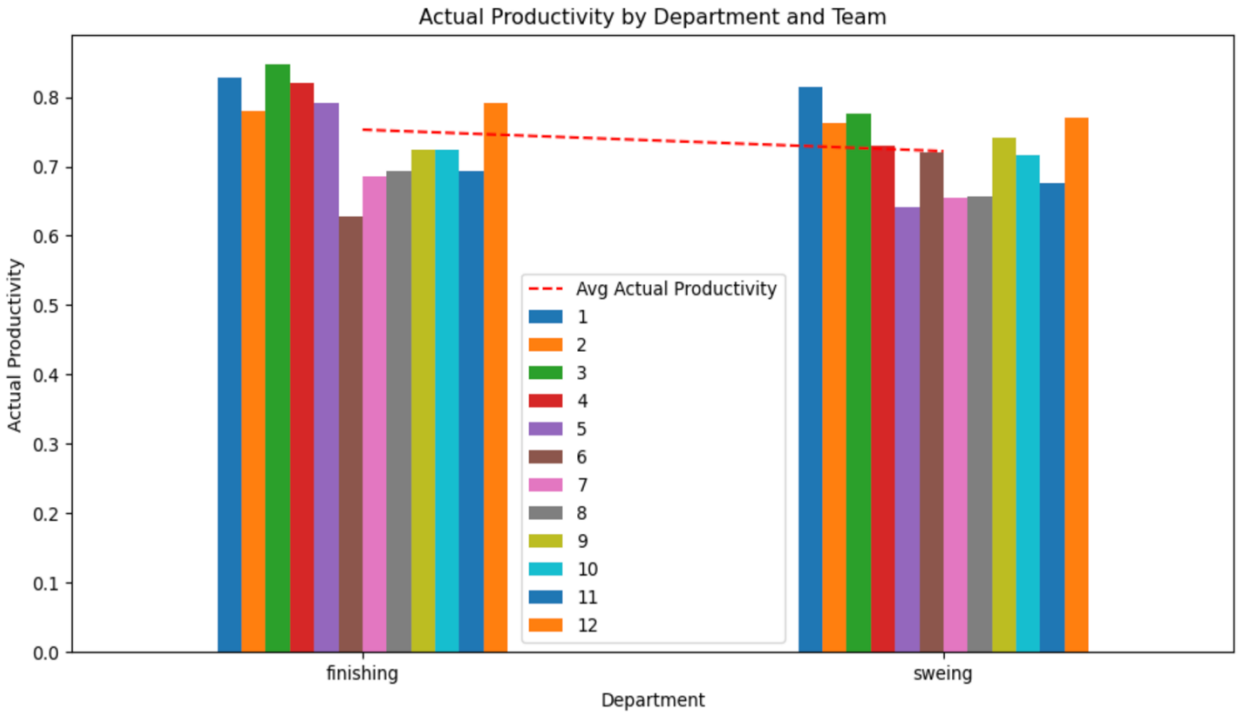
Why is this Important?

- **Optimizing** resource allocation
- **Improving** operational efficiency
- **Increasing** output without sacrificing quality.

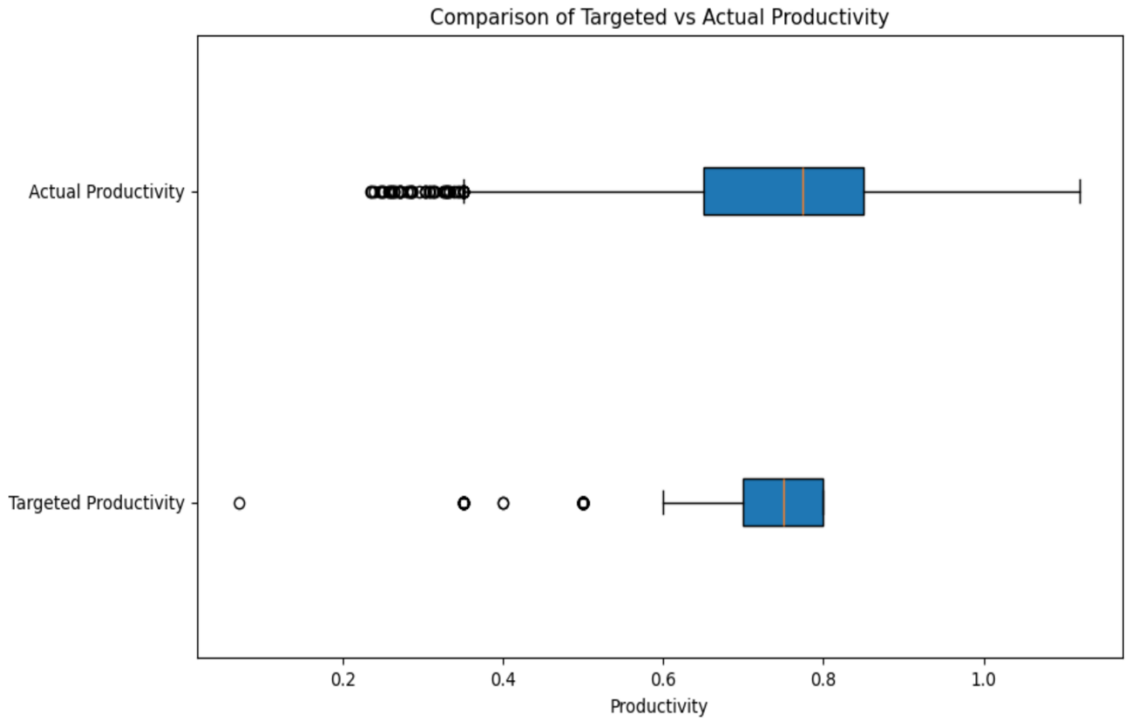
Data Collection

- **Data source:** UCI Machine Learning Repository
- Data was collected from a garment manufacturing facility in 2015

Exploratory Data Analysis

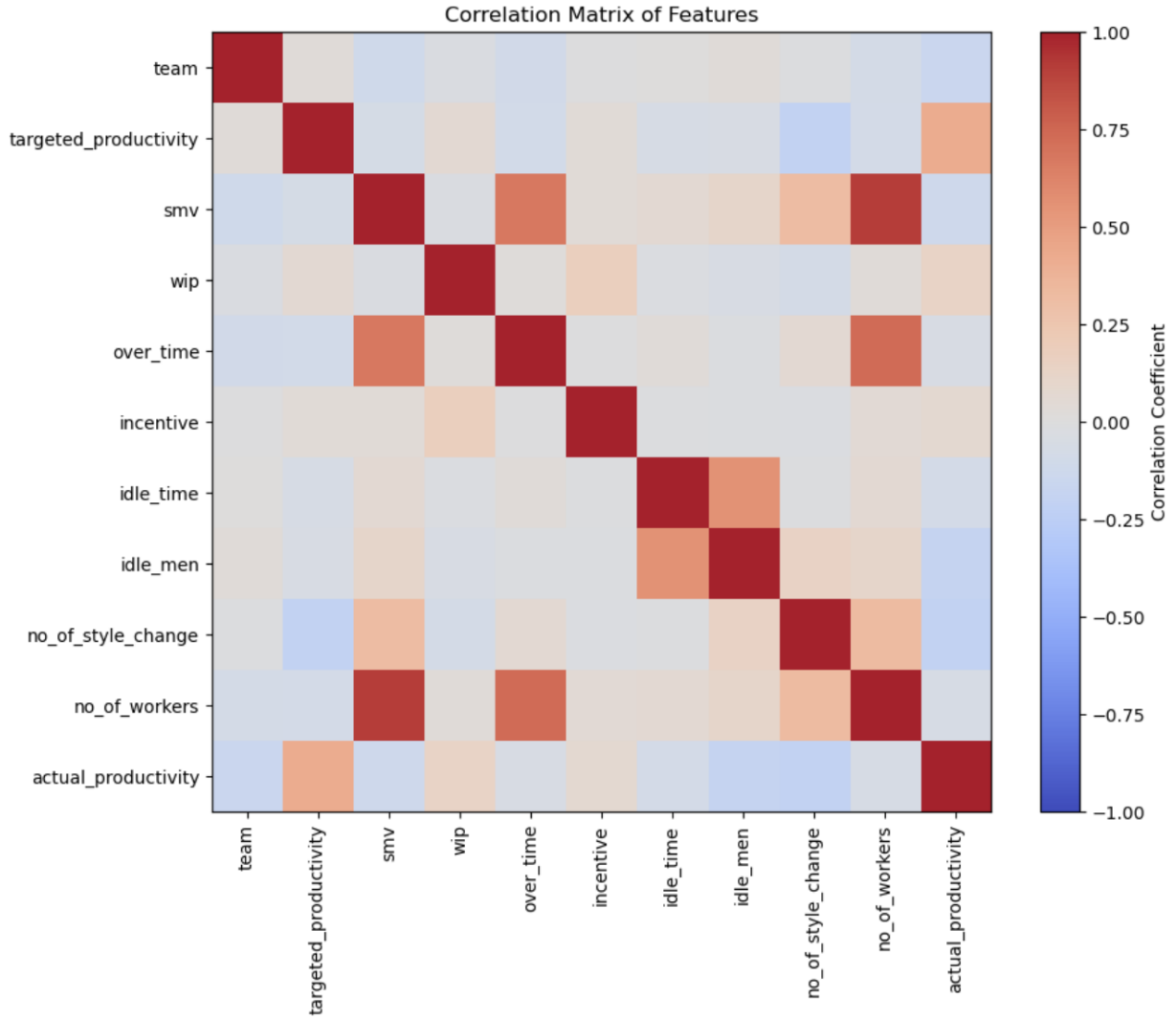


(This bar chart compares average actual productivity across different departments, with the sewing department showing lower average productivity than finishing.)



(This boxplot compares targeted productivity against actual productivity, which shows actual productivity always falls short of the target)

Exploratory Data Analysis



Splitting

```python
from sklearn.model_selection import TimeSeriesSplit
df['date'] = pd.to_datetime(df['date'], format='%m/%d/%Y')
data = df.sort_values(by='date')
X = df.drop(columns=['date', 'actual_productivity'])
y = df['actual_productivity']

tscv = TimeSeriesSplit(n_splits=3)
splits = list(tscv.split(X, y))
train_index = splits[0][0]
val_index = splits[1][1]
test_index = splits[2][1]
X_train = X.iloc[train_index]
y_train = y.iloc[train_index]
X_val = X.iloc[val_index]
y_val = y.iloc[val_index]
X_test = X.iloc[test_index]
y_test = y.iloc[test_index]


train_date_range = (data.iloc[train_index]['date'].min(), data.iloc[train_index]['date'].max())
val_date_range = (data.iloc[val_index]['date'].min(), data.iloc[val_index]['date'].max())
test_date_range = (data.iloc[test_index]['date'].min(), data.iloc[test_index]['date'].max())
```

Time Series Splitting:

- The data is ordered chronologically
- Using the **earlier data** for training and the **later data** for testing to avoid data leakage.

Train-Test Split:

- **Training Data**: The earlier 70% of the data.
- **Testing Data**: The most recent 30% of the data, which helps evaluate the model's performance on unseen future data.

Preprocessing

```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
onehot_ftrs = ['department', 'day']
std_ftrs = ['targeted_productivity', 'smv', 'wip', 'over_time', 'incentive', 'idle_time', 'idle_men', 'no_of_style_change', 'team', 'no_of_work

preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), onehot_ftrs),
        ('std', StandardScaler(), std_ftrs)])

clf = Pipeline(steps=[('preprocessor', preprocessor)])

X_train_prep = clf.fit_transform(X_train)
X_val_prep = clf.transform(X_val)
X_test_prep = clf.transform(X_test)

print(X_train.shape)
print(X_train_prep.shape)
print(X_train_prep)
```

✓ 0.0s Python

```
(300, 13)
(300, 18)
```

Handling Categorical Variables:

- Categorical features like **department**, **day**, and **quarter** were handled using **one-hot encoding**.

Standardization:

- Numeric features like **idle time**, **SMV (standard minute value)**, and **incentives** were scaled using **StandardScaler** to normalize the data.

Missing Values:

- 506 missing **Continuous Values** in wip

Number of Features:

- **Before Preprocessing**: 13
- **After Preprocessing**: 18