Asif Shah

@00794412

Machine Learning and Data Mining

Task 1

Title

Machine Learning Approach for Predicting and Optimizing Garment Industry Employee Productivity

Introduction

The garment manufacturing industry stands as a cornerstone of industrial globalization, characterized by its intensive labour requirements and significant dependence on human resources. In this sector, production efficiency and meeting global demand are directly tied to employee productivity across various departments. A common challenge in this industry is that the actual productivity of employees often fails to meet the targeted productivity set by authorities, leading to significant production losses (Imran et al., 2019).

Previous research has shown promising approaches to address this productivity gap. Imran et al. (2019) proposed a Deep Neural Network (DNN) model that achieved a Mean Absolute Error of 0.086, demonstrating significant improvement over baseline performance. Building on this work, Sabuj et al. (2022) developed an interpretable machine learning approach that achieved an even lower MAE of 0.072, while also incorporating explainable AI techniques to provide deeper insights into productivity factors.

This assessment focuses on using machine learning techniques to predict and optimize garment employee productivity.

Datasets

The dataset contains important attributes related to the garment manufacturing process and employee productivity.

- Number of Instances: 1,197
- Number of Features: 14 (including both integer and real-type variables)

Dataset Features:

- date: Date (MM-DD-YYYY)
- day: Day of the Week
- quarter: A portion of the month divided into four quarters
- department: The associated department
- team_no: The associated team number
- no_of_workers: Number of workers in each team
- no_of_style_change: Number of changes in product style
- targeted_productivity: The productivity target set for each team
- smv: Standard Minute Value (allocated time for tasks)
- wip: Work in progress (unfinished items)
- over_time: Overtime worked by each team (minutes)
- incentive: Financial incentives given (in BDT)
- idle_time: Time when production was interrupted
- idle_men: Number of workers idle due to interruptions
- actual_productivity: Actual productivity delivered (range: 0-1)

Data Source and Citation:

Dataset: Productivity Prediction of Garment Employees. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C51S6D.

Ethical, Social, and Legal Considerations

Worker Well-being:

This dataset provides valuable insights into productivity, but any optimization efforts must respect and consider worker well-being, preventing excessive workloads or unfair labour demands.

Privacy and Data Security:

While the dataset does not contain personally identifiable information, any analysis involving sensitive workforce data in real-world settings should adhere strictly to privacy and security regulations.

Machine learning models must be evaluated for bias to ensure that productivity enhancements are equitable and do not disproportionately favour or penalize certain teams or groups.

Sustainable Practices:

Efforts to optimize productivity should also consider environmental and social sustainability within the garment industry, ensuring recommendations align with responsible production practices.

EDA and Data preprocessing

Exploratory data analysis will help us select the most relevant features for modelling. Using Seaborn and Matplotlib, we will visualize the data and perform statistical analyses for deeper insights (Hafeez & Sial, 2021).

Data Inspection:

The dataset has 1197 records with 15 features.

```
[15]: #importing dataset
      df = pd.read_csv('garments_worker_productivity.csv', parse_dates=['date'])
      df.shape

[15]: (1197, 15)
```

Independent Variables:

All features listed under dataset heading except actual_productvity and targeted_productivity.

Dependent Variable:

productivity_label : This will be created in in **Future Engineering** step based on the difference between actual_productivity and target_productivity.

```
<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   datetime64[ns]
 1   quarter                1197 non-null   object
 2   department             1197 non-null   object
 3   day                    1197 non-null   object
 4   team                   1197 non-null   object
 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   object
 13  no_of_workers          1197 non-null   float64
 14  actual_productivity    1197 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(3), object(5)
memory usage: 140.4+ KB
None
```

Missing Values:

The dataset contains 506 missing values in the WIP column.

```
[21]: # Checking Null Values
      df.isnull().sum()

[21]: 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
```

Statistical Analysis:

- Targeted Productivity: Mean 0.73; most values fall between 0.70 and 0.80.

- SMV: Mean 15.06, reflecting varying task complexities.
- WIP: Highly variable; mean 1,190, indicating diverse production stages.
- Overtime: Mean 4,567 minutes; significant extended hours in some teams.
- Incentives: Mean 38.21; high variability, suggesting uneven distribution.
- Idle Time/Idle Men: Mostly zero; occasional peaks indicate sporadic inefficiencies.
- Actual Productivity: Mean 0.735 (SD 0.17); close to target but inconsistent.

```
[832]: df.describe()
```

[832]:

	date	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_workers	actual_produ
count	1197	1197.000000	1197.000000	691.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.(
mean	2015-02-04 10:56:50.526315776	0.729632	15.062172	1190.465991	4567.460317	38.210526	0.730159	0.369256	34.609858	0.
min	2015-01-01 00:00:00	0.070000	2.900000	7.000000	0.000000	0.000000	0.000000	0.000000	2.000000	0.:
25%	2015-01-18 00:00:00	0.700000	3.940000	774.500000	1440.000000	0.000000	0.000000	0.000000	9.000000	0.(
50%	2015-02-03 00:00:00	0.750000	15.260000	1039.000000	3960.000000	0.000000	0.000000	0.000000	34.000000	0.:
75%	2015-02-23 00:00:00	0.800000	24.260000	1252.500000	6960.000000	50.000000	0.000000	0.000000	57.000000	0.8
max	2015-03-11 00:00:00	0.800000	54.560000	23122.000000	25920.000000	3600.000000	300.000000	45.000000	89.000000	1.
std	NaN	0.097891	10.943219	1837.455001	3348.823563	160.182643	12.709757	3.268987	22.197687	0.

Numerical and Categorical:

Based on domain knowledge we divided the data into categorical and numerical features.

```
[835]: # define numerical & categorical columns
numeric_features = [feature for feature in df.columns if df[feature].dtype != 'O']
categorical_features = [feature for feature in df.columns if df[feature].dtype == 'O']

# print columns
print('We have {} numerical features : {}'.format(len(numeric_features), numeric_features))
print('\nWe have {} categorical features : {}'.format(len(categorical_features), categorical_features))
```

```
We have 10 numerical features : ['date', 'targeted_productivity', 'smv', 'wip', 'over_time', 'incentive', 'idle_time', 'idle_men', 'no_of_wor
kers', 'actual_productivity']

We have 5 categorical features : ['quarter', 'department', 'day', 'team', 'no_of_style_change']
```

Univariate Analysis:

Univariate Analysis of Categorical Features:

- Quarter 1 has the highest representation (30.08%), while there is strange Quarter5. Normally there are 4 quarters in the month, in the next step we will explore this more.
- The sewing department dominates (57.73%), while finishing departments share the remaining proportion. But there are **TWO** finishing departments which we need to see.
- Teams are not balanced; Team 8 and Team 2 lead (9.11%), and Team 11 has the lowest proportion (7.35%).
- 87.72% of records show no style changes; only 2.76% have two changes, indicating stable processes.

[837]:
```
# proportion of count data on categorical columns
for col in categorical_features:
    print(df[col].value_counts(normalize=True) * 100)
    print('----------------------------')
```

```
quarter
Quarter1    30.075188
Quarter2    27.986633
Quarter4    20.718463
Quarter3    17.543860
Quarter5     3.675856
Name: proportion, dtype: float64
----------------------------
department
sweing        57.727652
finishing     21.470343
finishing     20.802005
Name: proportion, dtype: float64
----------------------------
day
Wednesday    17.376775
Sunday       16.959064
Tuesday      16.791980
Thursday     16.624896
Monday       16.624896
Saturday     15.622389
Name: proportion, dtype: float64
----------------------------
team
8     9.106099
2     9.106099
1     8.771930
4     8.771930
9     8.688388
10    8.354219
12    8.270677
7     8.020050
3     7.936508
6     7.852966
5     7.769424
11    7.351713
Name: proportion, dtype: float64
----------------------------
no_of_style_change
0    87.719298
1     9.523810
2     2.756892
Name: proportion, dtype: float64
----------------------------
```

IMPORTANT: *Upon exploration we found out there was some human error involved in writing Finishing. It was proceeding SPACE, as It is not something that needs to be waited for Feature Engineering steps, we correct this here so that we do not have to take it along the way. Following also clears out our doubt of* ***Quarter5****.*

[839]:
```
# Filter the data for Quarter5 (as a string)
df_q = df[df['quarter'] == 'Quarter5']

# Print the shape and indices of the filtered DataFrame
print(df_q.shape)
print(df_q.index)

# Optional: Print the unique dates in Quarter5
print(df_q['date'].unique())
```

```
(44, 15)
Index([498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511,
       512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525,
       526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539,
       540, 541],
      dtype='int64')
<DatetimeArray>
['2015-01-29 00:00:00', '2015-01-31 00:00:00']
Length: 2, dtype: datetime64[ns]
```

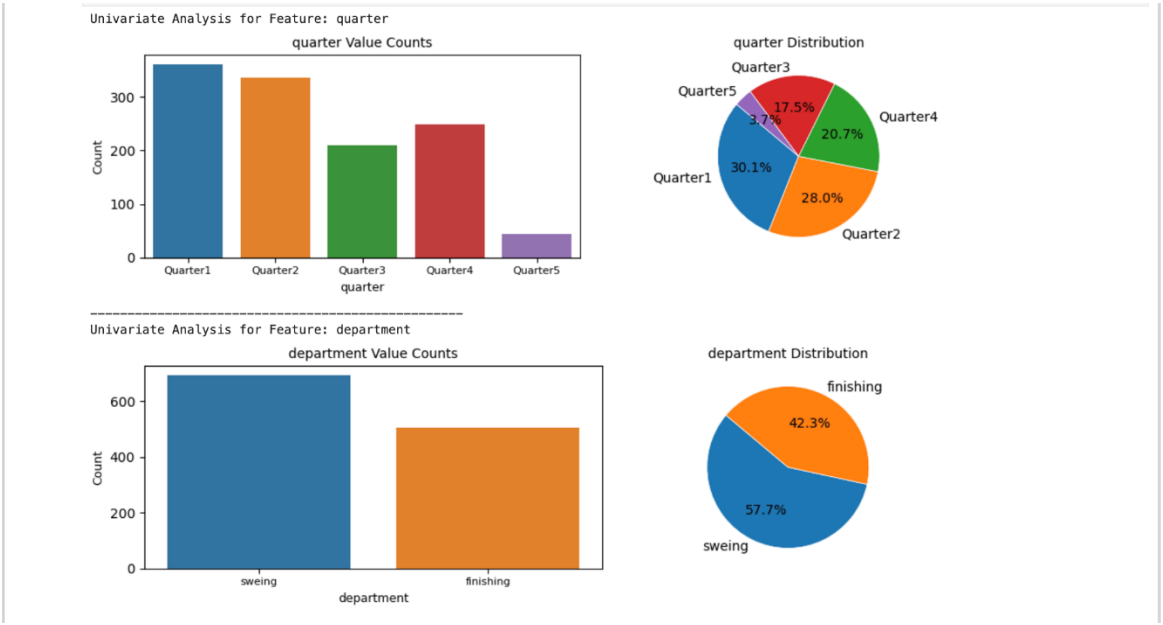
Quarter 5 contains 2 days as 29th and 31th of January.

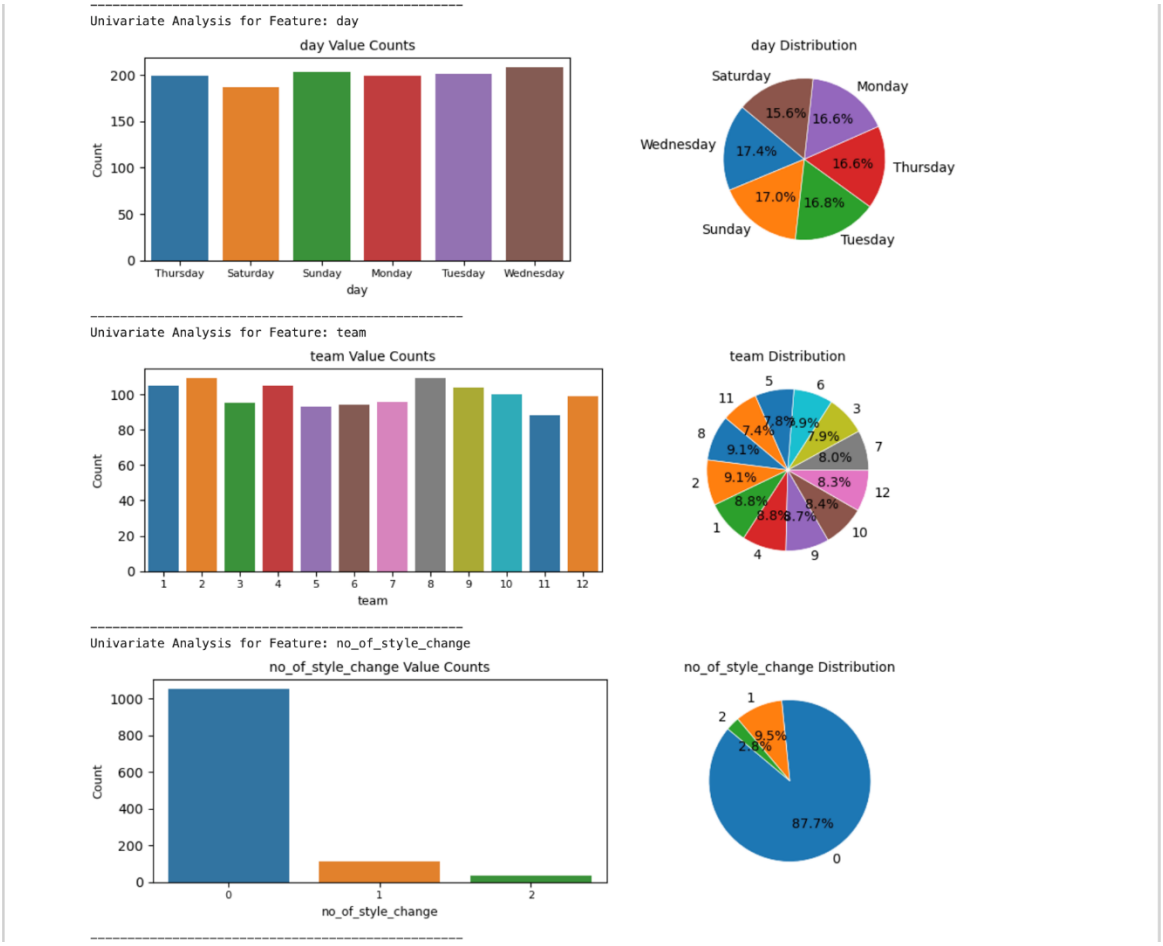
It was needed to sort this Human Error for furthur analysis.

[841]:
```
df['department'] = df['department'].str.strip()
df['department'].value_counts()
```

[841]:
```
department
sweing       691
finishing    506
Name: count, dtype: int64
```

Visualization distribution of same categorical features:

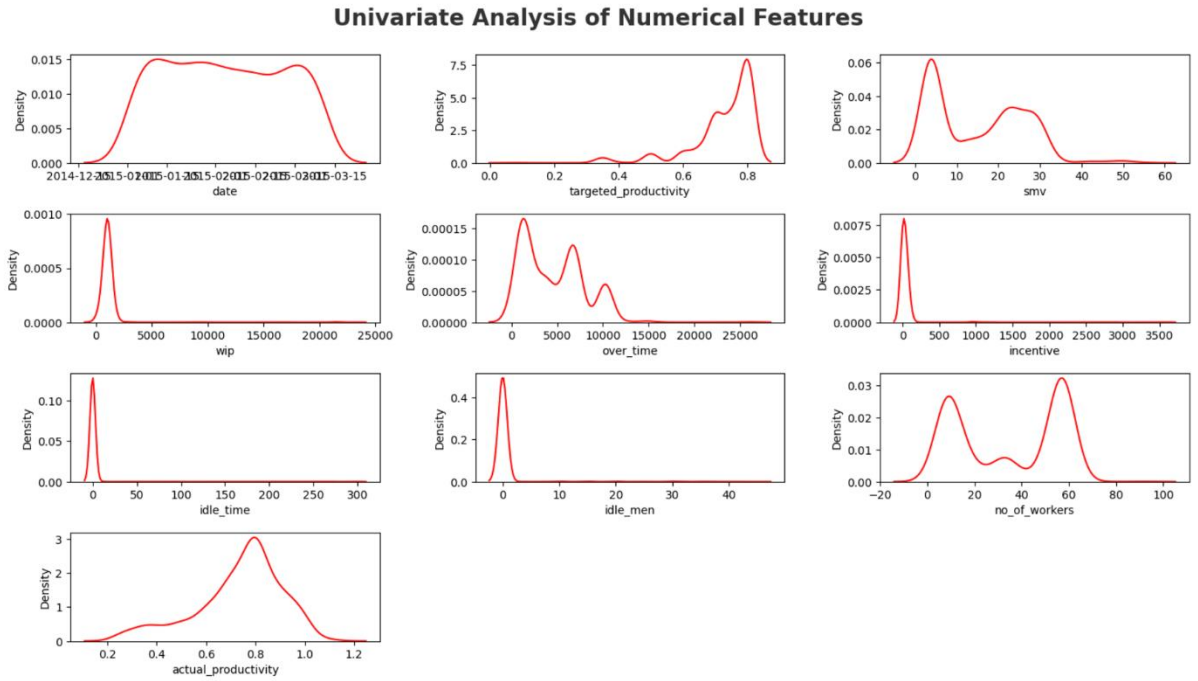




```python
def univariate_analysis(df, categorical_features):
    for feature in categorical_features:
        print(f"Univariate Analysis for Feature: {feature}")

        # Create a figure with 2 subplots in a row
        fig, axes = plt.subplots(1, 2, figsize=(10, 3))  # Adjust figure size

        # Get the color palette
        palette = sns.color_palette("tab10", df[feature].nunique())

        # 1. Bar Plot
        sns.countplot(x=feature, data=df, palette=palette, ax=axes[0])
        axes[0].set_title(f"{feature} Value Counts", fontsize=10)
        axes[0].set_xlabel(feature, fontsize=9)
        axes[0].set_ylabel("Count", fontsize=9)
        axes[0].tick_params(axis='x', labelsize=8)

        # 2. Pie Chart
        df[feature].value_counts().plot(
            kind='pie',
            colors=palette,
            autopct='%1.1f%%',
            ax=axes[1],
            startangle=140,
            wedgeprops={'linewidth': 0.5, 'edgecolor': 'white'}
        )
        axes[1].set_ylabel('')  # Hide y-axis label
        axes[1].set_title(f"{feature} Distribution", fontsize=10)

        plt.tight_layout()
        plt.show()
        print("-" * 50)

univariate_analysis(df, categorical_features)
```

Univariate Analysis of Numerical Feature:

The univariate analysis of numerical features reveals distinct distribution patterns. Both targeted productivity and actual productivity are left-skewed, indicating a concentration of values toward the higher end. In contrast, features like SMV, WIP, incentives, idle time, and idle men exhibit right-skewed distributions, with most values clustered toward the lower end and a few high outliers. Overtime also shows a right-skewed pattern, suggesting that extended work hours are less frequent but can be significant. Additionally, the number of workers displays a bimodal distribution, indicating two distinct groupings within the data.



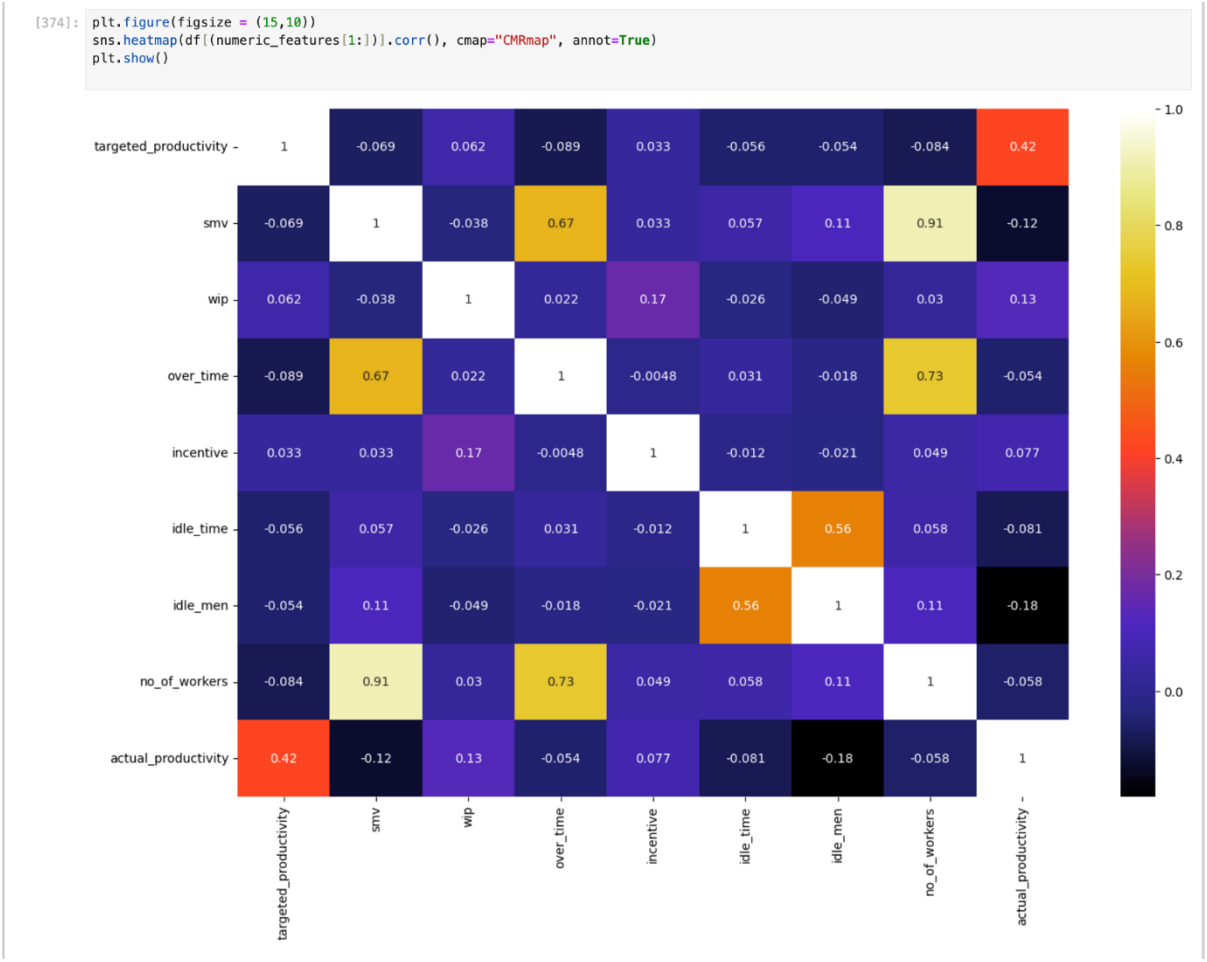

```
plt.figure(figsize=(15, 10))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numeric_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[numeric_features[i]], color='red')
    plt.xlabel(numeric_features[i])
    plt.tight_layout()
```

Multivariate Analysis

By first checking the multicollinearity of our numerical variables, we found that actual productivity and targeted productivity are moderately correlated, with a positive correlation of 0.42. This indicates that most of the time, the target is met, so we can consider using either one of these variables when comparing it with others. Targeted productivity shows a strong positive relationship with SMV and a moderate positive correlation with WIP.

The variables **date**, **SMV**, and **idle men** also exhibit a moderate positive correlation with **targeted productivity**. Additionally, **overtime** and **SMV** have a strong positive correlation, as do **number of workers** and **overtime**. Finally, **idle time** and **idle men** show a positive correlation of 0.56, indicating that idle time increases with more idle personnel.

```
[374]: plt.figure(figsize = (15,10))
sns.heatmap(df[(numeric_features[1:])].corr(), cmap="CMRmap", annot=True)
plt.show()
```



After the correlation map, we found the two productivity variables highly correlated. Thus, we will focus on *actual_productivity* for EDA, as it better reflects real outcomes and offers more meaningful insights into factors influencing productivity.

Productivity by Department:

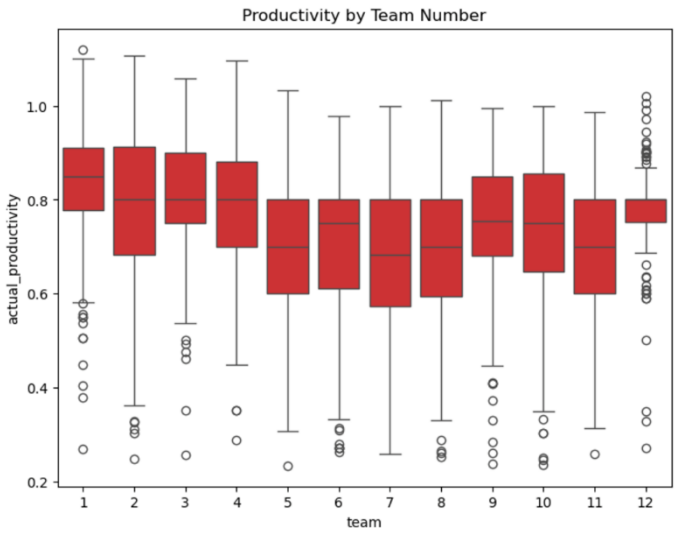
The sewing department shows a potential lagging in actual productivity, while the finishing department has good actual productivity and is performing well. We might need to investigate the cause of what is hindering the productivity in the sewing department.

[377]:
```
# Bar plots for categorical features
categorical_columns = ['department', 'team', 'quarter', 'day']

# Boxplot of productivity by department
plt.figure(figsize=(8, 6))
sns.boxplot(x='department', y='actual_productivity', data=df)
plt.title('Productivity by Department')
plt.show()
```

Productivity between Teams:

Some teams perform consistently well with no negative outliers like 4, 7, Other teams also show good performance but have some low-productivity outliers that should be investigated. Overall, the data indicates strong average productivity, with certain teams showing higher peaks. A few teams exhibit average performance, suggesting there is room for improvement.

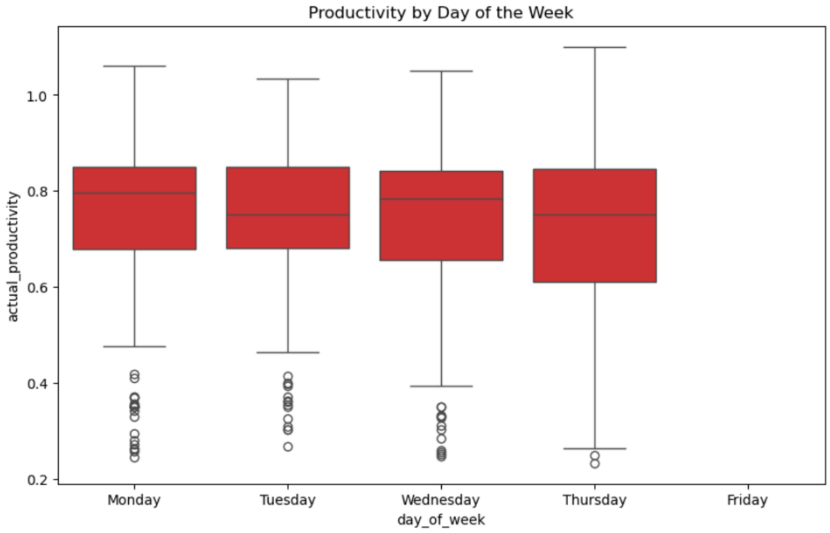
[378]:
```
# Boxplot of productivity by team number
plt.figure(figsize=(8, 6))
sns.boxplot(x='team', y='actual_productivity', data=df)
plt.title('Productivity by Team Number')
plt.show()
```



Productivity By Day of Week:

On Thursday, actual productivity is notably higher compared to other days, as the mean productivity is also elevated. In contrast, Monday, Tuesday, and Wednesday show some potential negative outliers that may require further investigation.

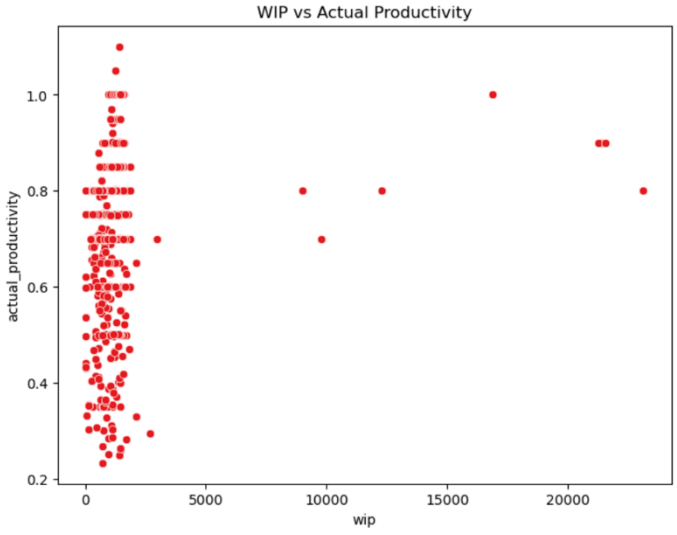
[379]:
```
# Convert 'date' column to datetime
df['date'] = pd.to_datetime(df['date'], format='%m-%d-%Y')

# Extract additional time-based features
df['day_of_week'] = df['date'].dt.day_name()
df['month'] = df['date'].dt.month

# Boxplot of productivity by day of the week
plt.figure(figsize=(10, 6))
sns.boxplot(x='day_of_week', y='actual_productivity', data=df, order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'])
plt.title('Productivity by Day of the Week')
plt.show()
```



Impact of WIP on Actual Productivity:

This is likely because the work in progress should decrease, indicating an increase in productivity, with only negligible outliers.

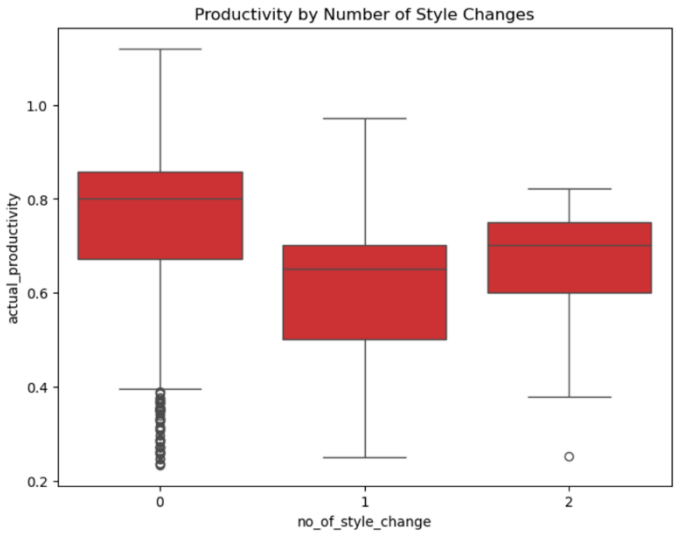
[381]:
```
# Scatter plot between WIP and actual productivity
plt.figure(figsize=(8, 6))
sns.scatterplot(x='wip', y='actual_productivity', data=df)
plt.title('WIP vs Actual Productivity')
plt.show()
```



Impact of Style Changes on Actual Productivity:

From this chart, we can conclude that a decrease in style changes results in higher productivity.

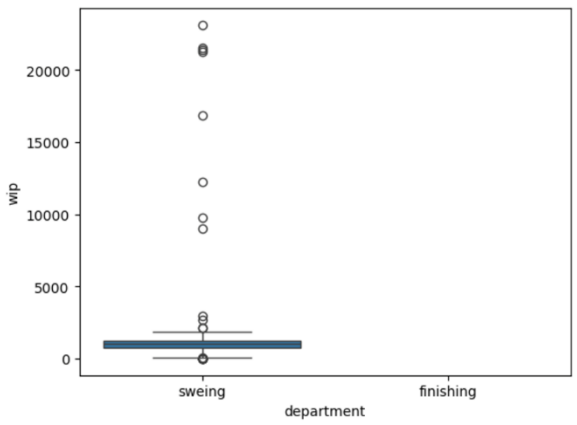
```
[382]: # Boxplot of productivity by number of style changes
       plt.figure(figsize=(8, 6))
       sns.boxplot(x='no_of_style_change', y='actual_productivity', data=df)
       plt.title('Productivity by Number of Style Changes')
       plt.show()
```



Work In Progress in Different Departments:

The finishing department has zero WIP, likely because the work is still in progress in the sewing department. So, zero values are valid.

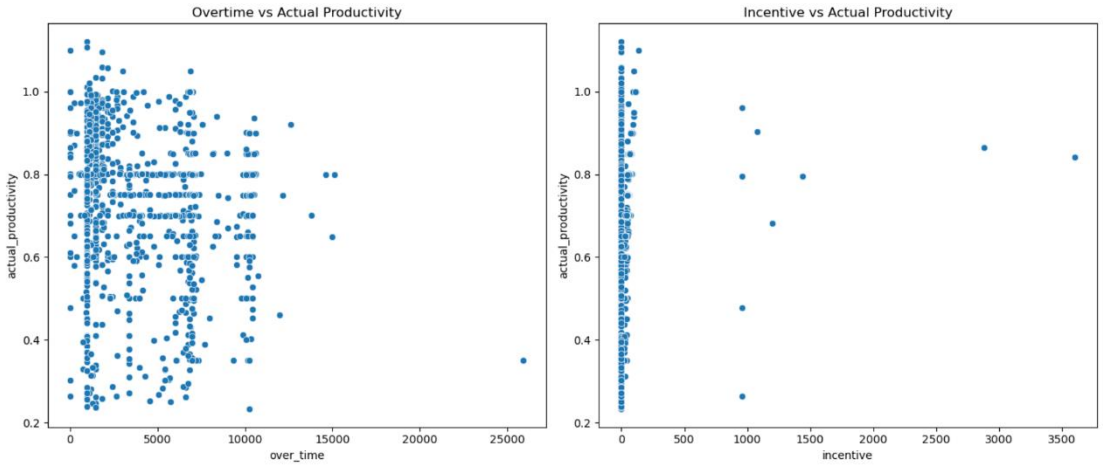
```
[87]: sns.boxplot(x='department',y='wip',data=df)
[87]: <Axes: xlabel='department', ylabel='wip'>
```



Impact of Overtime and Incentives on Actual Productivity:

Zero incentive is associated with higher actual productivity, but this does not necessarily imply that lower incentives lead to increased productivity.

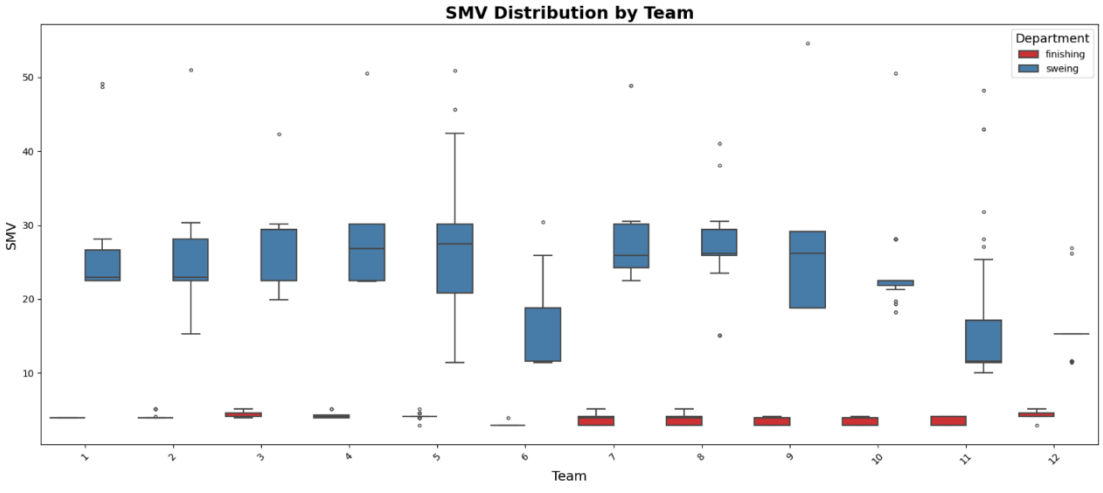
[89]:
```
plt.figure(figsize=(14, 6))

# Scatter plot for overtime vs actual productivity
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first plot
sns.scatterplot(x='over_time', y='actual_productivity', data=df)
plt.title('Overtime vs Actual Productivity')

# Scatter plot for incentive vs actual productivity
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second plot
sns.scatterplot(x='incentive', y='actual_productivity', data=df)
plt.title('Incentive vs Actual Productivity')

# Show the plots
plt.tight_layout()
plt.show()
```



Standard Minutes Time by Team:

The sewing department shows significant variability in SMV between teams, while the finishing department maintains more consistent SMV values across all teams.

[944]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20, 8))  # Adjusted size for a more balanced layout
palette = "Set1"  # Changed palette for a different color scheme

sns.boxplot(x="team", y="smv", hue="department", data=df,
            palette=palette, fliersize=3, linewidth=1.5)

plt.title('SMV Distribution by Team', fontsize=18, fontweight='bold')
plt.xlabel('Team', fontsize=14)
plt.ylabel('SMV', fontsize=14)
plt.xticks(rotation=45)  # Rotate x-axis labels if necessary
plt.legend(title='Department', title_fontsize='13')

plt.show()
```

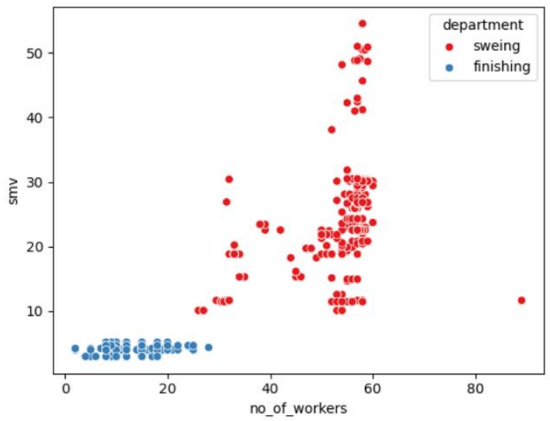


Impact of Workforce on Departmental Productivity:

In the sewing department, an increase in the number of workers impacts productivity, whereas this factor does not appear to influence the finishing department.

```
[398]: sns.scatterplot(data=df, x="no_of_workers", y="smv", hue="department")
```

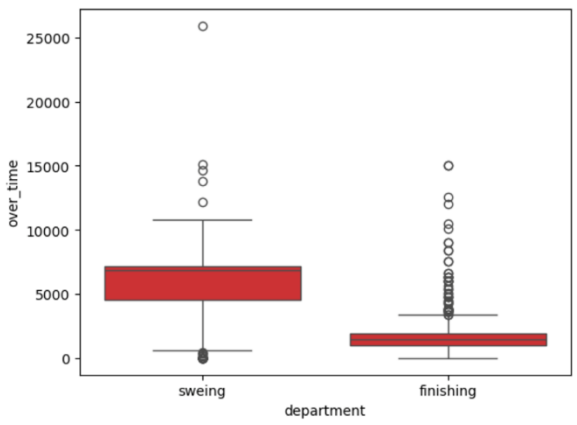
```
[398]: <Axes: xlabel='no_of_workers', ylabel='smv'>
```



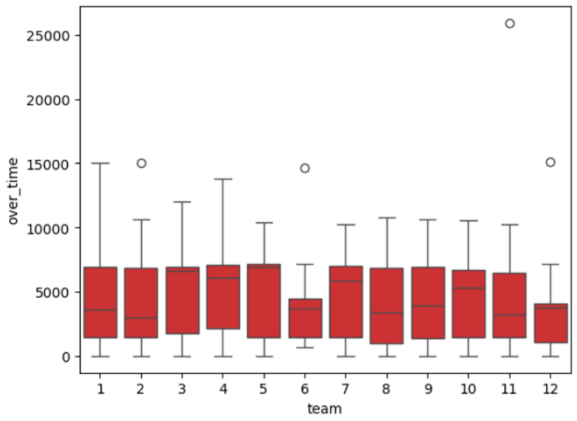
Overtime Analysis Across Departments and Teams

Although the sewing department shows higher average overtime, the finishing department contains significant outliers, suggesting occasional extreme overtime instances. This indicates potential variability in workload or operational issues within the finishing department. Additionally, there is noticeable fluctuation in overtime between teams.

```
[400]: sns.boxplot(x='department',y='over_time',data=df)
plt.show()
```



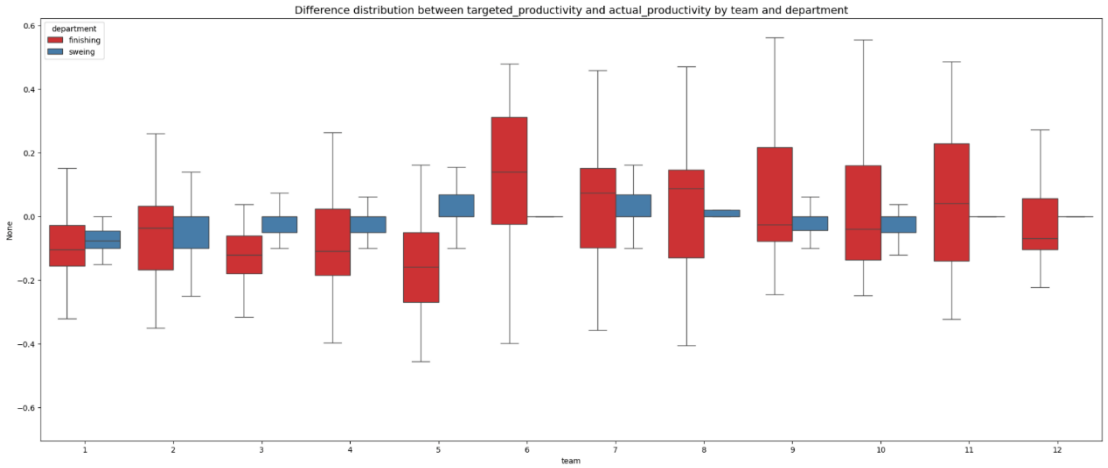
```
[403]: sns.boxplot(x='team',y='over_time',data=df)
       plt.show()
```



Difference distribution between targeted_productivity and actual_productivity by team and department

The sewing department demonstrates higher productivity compared to the finishing department, with greater fluctuations in productivity levels. This variability may be due to the higher work-in-progress (WIP) observed earlier in the EDA.

```
[405]: plt.figure(figsize=(25, 10))
       palette = "Set1"

       sns.boxplot(x = 'team', y = df.targeted_productivity-df.actual_productivity, data = df,
            palette = palette,hue='department',fliersize = 0)

       plt.title('Difference distribution between targeted_productivity and actual_productivity by team and department',fontsize= 14)
       plt.show()
```



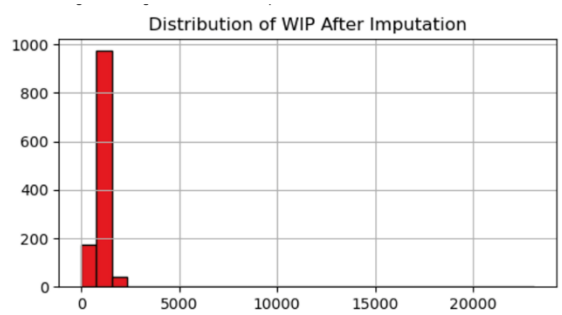
Data Preprocessing

This analysis helped us understand the data in alignment with business knowledge, guiding our feature selection process. Based on our findings, we are now ready to proceed with feature engineering. The steps we will focus on are:

1. *Handling Missing Values:*

Missing values in WIP are addressed by applying median imputation, as the median is less sensitive to outliers and preserves the data's central tendency.

```
imputer = SimpleImputer(strategy='median')

# Reshape the 'wip' column into a 2D array for the imputer
wip_values = df[['wip']]

# Fit the imputer and transform the 'wip' column to fill missing values
df['wip'] = imputer.fit_transform(wip_values)
```



2. *Encoding Categorical Features:*

Label Encoding: Applied to no_of_style_change as it represents ordinal categorical variables. Label encoding preserves the inherent order within categories, making it suitable for machine learning models (Brownlee, 2020).

One-Hot Encoding: Applied to department, day, and team since they are nominal features with no inherent order. This method creates binary columns for each category, ensuring that the data is model-friendly without introducing any unintended ordinal relationships (Chollet, 2018). Proper encoding of categorical variables enhances the performance and interpretability of machine learning algorithms (Hastie, Tibshirani, & Friedman, 2009).

855]:
```
df.columns
```

855]:
```
Index(['date', 'quarter', 'department', 'day', 'team', 'targeted_productivity',
       'smv', 'wip', 'over_time', 'incentive', 'idle_time', 'idle_men',
       'no_of_style_change', 'no_of_workers', 'actual_productivity'],
      dtype='object')
```

857]:
```
# Label Encoding

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to 'quarter' and 'no_of_style_change'
df['no_of_style_change'] = label_encoder.fit_transform(df['no_of_style_change'].astype(str))
```

859]:
```
# One-Hot Encoding
def create_dummies(df, column_name):
    return pd.concat([df, pd.get_dummies(df[column_name], prefix=column_name)], axis=1)

# Apply One-Hot Encoding to the relevant columns
columns_to_encode = ['department', 'day', 'team', 'quarter']
for col in columns_to_encode:
    df = create_dummies(df, col)

df.columns
```

859]:
```
Index(['date', 'quarter', 'department', 'day', 'team', 'targeted_productivity',
       'smv', 'wip', 'over_time', 'incentive', 'idle_time', 'idle_men',
       'no_of_style_change', 'no_of_workers', 'actual_productivity',
       'department_finishing', 'department_sweing', 'day_Monday',
       'day_Saturday', 'day_Sunday', 'day_Thursday', 'day_Tuesday',
       'day_Wednesday', 'team_1', 'team_2', 'team_3', 'team_4', 'team_5',
       'team_6', 'team_7', 'team_8', 'team_9', 'team_10', 'team_11', 'team_12',
       'quarter_Quarter1', 'quarter_Quarter2', 'quarter_Quarter3',
       'quarter_Quarter4', 'quarter_Quarter5'],
      dtype='object')
```

3. *Creating the Target Variable:*

‘productivity_label' target variable been created to classify performance based on the difference between actual and targeted productivity.

[105]:
```
# Creating the 'productivity_label' target variable based on the difference between actual_productivity and target_productivity
df['productivity_label'] = df['actual_productivity'] - df['targeted_productivity']

# Classify performance and assign numeric values (-1 for underperformed , 0 for expected , 1 for overperformed)
df['productivity_label'] = df['productivity_label'].apply(lambda x: 1 if x > 0 else (0 if x == 0 else -1))

# Display the updated dataframe
df.head()
```

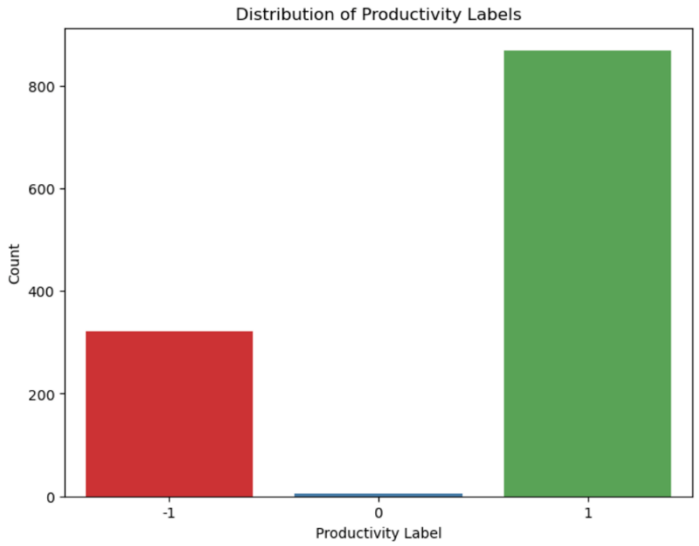
[105]:

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	...	team_9	team_10	team_11	team_12	quarter_Qua
0	2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	...	False	False	False	False	
1	2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	...	False	False	False	False	
2	2015-01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	...	False	False	True	False	
3	2015-01-01	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	...	False	False	False	True	
4	2015-01-01	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	...	False	False	False	False	

5 rows × 41 columns

```
OverPerformed: If the difference is positive (i.e., actual productivity exceeds targeted productivity), assign 1.
Expected: If the difference is zero (i.e., actual productivity equals targeted productivity), assign 0.
Underperformed: If the difference is negative (i.e., actual productivity is less than targeted productivity), assign -1.
```

```
productivity_label
 1    869
-1    322
 0      6
Name: count, dtype: int64
```



4. Creating Binary Class's and Addressing Class Imbalance with SMOTE:

The original productivity_label column had three classes (1.0, 0.0, -1.0), with the 0.0 class having very few samples, leading to a highly imbalanced dataset. To address this, the problem was simplified into a binary classification:

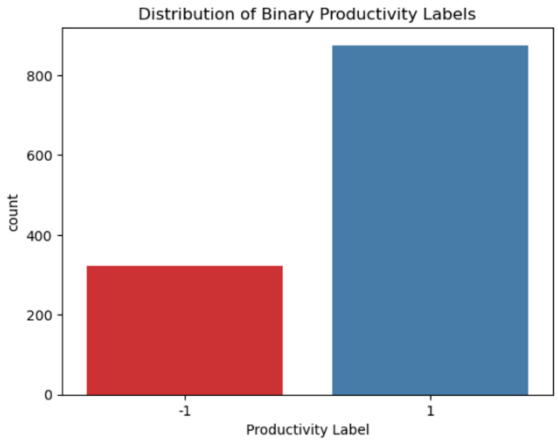
1. 1 represents both the original 1.0 (overperformed) and 0.0 (expected).
2. -1 remains unchanged for underperformance.

The target variable, productivity_label, originally had an imbalanced distribution with 875 samples in class 1 (overperformed) and 322 in class -1 (underperformance). This imbalance could lead to biased predictions, as the model might favor the majority class during training.

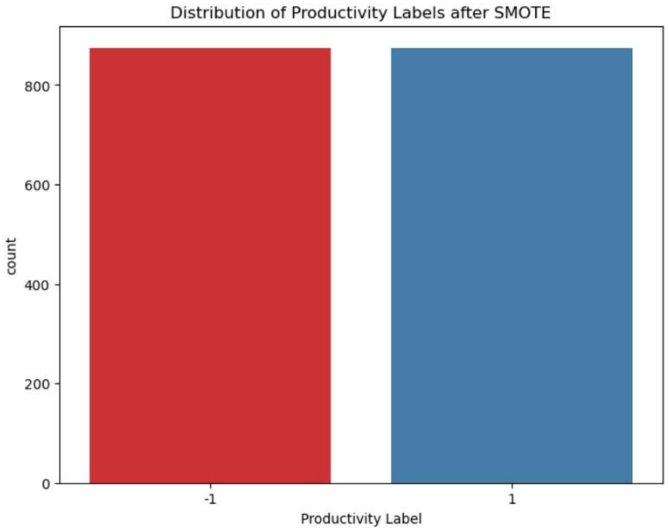
To address this, SMOTE (Synthetic Minority Oversampling Technique) was applied, generating synthetic samples for the minority class to balance the dataset (Chawla et al., 2002).

```
[868…
# Convert the 3-class 'productivity_label' into a binary classification problem
df['productivity_label'] = df['productivity_label'].apply(lambda x: -1 if x == -1 else 1)

# Display the value counts of the new target variable
print(df['productivity_label'].value_counts())

# Plot the distribution of the binary target variable
ax = sns.countplot(x='productivity_label', data=df, palette='Set1')
plt.xlabel('Productivity Label')
plt.title('Distribution of Binary Productivity Labels')
plt.show()
```

```
productivity_label
 1    875
-1    322
Name: count, dtype: int64
```



[216]:
```python
from imblearn.over_sampling import SMOTE
# Drop unnecessary columns (especially dates or non-numerical types)
df1 = df.drop(['quarter', 'department', 'day', 'team'], axis=1)

# Ensure datetime columns are dropped or converted if present
df1 = df1.select_dtypes(include=['int64', 'float64', 'bool'])
df1 = df1.apply(lambda x: x.astype(float) if x.dtypes == 'bool' else x)

# Separate features and target variable
X = df1.drop('productivity_label', axis=1)
y = df1['productivity_label']

# Apply SMOTE to balance the classes
smt = SMOTE(random_state=42)
X_smote, y_smote = smt.fit_resample(X, y)

# Create a DataFrame for the SMOTE-processed data
df_smote = pd.concat([pd.DataFrame(X_smote, columns=X.columns), pd.DataFrame(y_smote, columns=['productivity_label'])], axis=1)

# Plot class distribution after SMOTE
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='productivity_label', data=df_smote, palette='Set1')
plt.xlabel('Productivity Label')
plt.title('Distribution of Productivity Labels after SMOTE')
plt.show()
```



This balanced distribution improves the model's ability to learn patterns from both classes, resulting in more reliable and unbiased predictions for productivity performance.

Implementation (In Python)

Description and Justification of Algorithms Used

1. Random Forest Classifier: Random Forest is an ensemble method that combines multiple decision trees to improve classification accuracy. Each tree in the forest is trained on a subset of the data, and the final prediction is based on the majority vote, enhancing robustness, and reducing overfitting (Breiman, 2001).

Justification for Selection:

- Handling Mixed Data: Random Forest effectively manages both numerical and categorical features, aligning with our dataset's structure.
- Robustness to Class Imbalance: With the target variable balanced using SMOTE, Random Forest ensures reliable classification between overperformance and underperformance.
- Key Parameter: random_state=42 ensures consistent and reproducible results, crucial for comparing different models.

2. Logistic Regression: It is a linear model used for binary classification. It predicts the probability of an outcome and provides insights into feature importance through its coefficients (Peng et al., 2002).

Justification for Selection:

- **Baseline Performance:** As a simple, interpretable model, it serves as a benchmark to assess the performance of more complex models like Random Forest.
- **Feature Interpretability:** The model's coefficients help identify the most influential factors affecting employee productivity.
- **Key Parameter:** max_iter=1000 ensures convergence, especially when working with scaled data and balanced classes.

Experimental Procedure

1. Data Scaling and Splitting:

The dataset was split into 80% training and 20% testing sets. StandardScaler was used to scale features, ensuring a mean of 0 for all variables. This standardization ensures equal contribution from features with varying scales, such as incentive, over_time, and idle_time.

```
219]: from sklearn.model_selection import train_test_split

columns = ['smv',
       'wip', 'over_time', 'incentive', 'idle_time', 'idle_men',
       'no_of_workers',
       'quarter_Quarter1', 'quarter_Quarter2', 'quarter_Quarter3',
       'quarter_Quarter4', 'quarter_Quarter5', 'department_finishing',
       'department_sweing', 'day_Monday', 'day_Saturday', 'day_Sunday',
       'day_Thursday', 'day_Tuesday', 'day_Wednesday', 'team_1', 'team_2',
       'team_3', 'team_4', 'team_5', 'team_6', 'team_7', 'team_8', 'team_9',
       'team_10', 'team_11', 'team_12', 'no_of_style_change']

X = df[columns]
y = df['productivity_label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2,random_state=42)
print("Training set shape (X):", X_train.shape)
print("Testing set shape (X):", X_test.shape)
print("Training set shape (y):", y_train.shape)
print("Testing set shape (y):", y_test.shape)
print("-------------------------------------")
```

```
Training set shape (X): (957, 33)
Testing set shape (X): (240, 33)
Training set shape (y): (957,)
Testing set shape (y): (240,)
-------------------------------------
```

Scaling Dataset

```
223]: # Initialize StandardScaler
scaler = StandardScaler()

# Apply StandardScaler to training data and then to test data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

2. Feature Selection Using Variance Threshold:

Variance Threshold was applied to remove features with zero variance, retaining only those with variation (Guyon et al., 2002). Since no feature was found to be less important, all were retained.

```
Final selected features after VarianceThreshold: Index(['smv', 'wip', 'over_time', 'incentive', 'idle_time', 'idle_men',
       'no_of_workers', 'quarter_Quarter1', 'quarter_Quarter2',
       'quarter_Quarter3', 'quarter_Quarter4', 'quarter_Quarter5',
       'department_finishing', 'department_sweing', 'day_Monday',
       'day_Saturday', 'day_Sunday', 'day_Thursday', 'day_Tuesday',
       'day_Wednesday', 'team_1', 'team_2', 'team_3', 'team_4', 'team_5',
       'team_6', 'team_7', 'team_8', 'team_9', 'team_10', 'team_11', 'team_12',
       'no_of_style_change'],
      dtype='object')
Features removed: Index([], dtype='object')
```

```
# Feature selection using VarianceThreshold
variance_selector = VarianceThreshold(threshold=0.0)
X_train_fs = variance_selector.fit_transform(X_train)
X_test_fs = variance_selector.transform(X_test)

# Get the feature indices selected by VarianceThreshold
selected_variance_features = variance_selector.get_support()
selected_variance_columns = X.columns[selected_variance_features]

# Get the removed features
removed_features = X.columns[~selected_variance_features]

# Print the final selected and removed features
print(f"Final selected features after VarianceThreshold: {selected_variance_columns}")
print(f"Features removed: {removed_features}")

# Visualize the selected and removed features using a heatmap
new_features_boolean = np.isin(np.arange(X_train.shape[1]), selected_variance_features)
sns.heatmap(new_features_boolean.reshape(1, -1), cmap='rocket', cbar=False, annot=True, xticklabels=selected_variance_columns)
plt.title("Selected Features after VarianceThreshold")
plt.show()
```

Hyperparameter Tuning and Validation

RandomizedSearchCV was used to tune hyperparameters for both models which randomly samples hyperparameters and selects the best ones based on cross-validation performance, making it computationally efficient compared to exhaustive grid search (Bergstra & Bengio, 2012).

1. For **Random Forest** we tuned parameters like the number of trees **(**n_estimators), maximum depth (max_depth), and minimum samples for splitting and leaf nodes. And for **Logistic Regression** we focused on regularization strength (C), penalty type, solver, and maximum iterations.
2. **Cross-Validation:** 5-fold cross-validation was used during the hyperparameter search to ensure the model generalizes well.
3. After finding the best parameters, both models were re-trained on the full training set and evaluated on the test set using accuracy and a classification report.

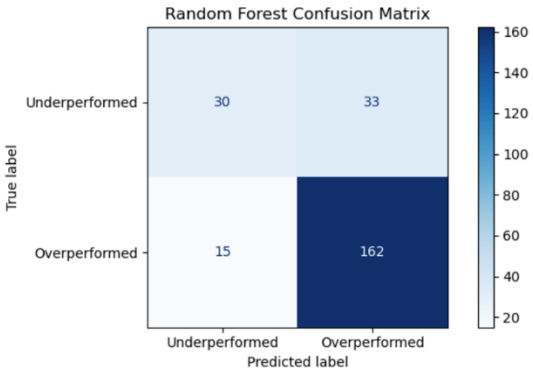
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Random Forest Hyperparameter Tuning
rf_random_grid = {
    'n_estimators': randint(50, 300),
    'max_depth': [None] + list(randint(10, 50).rvs(5)),
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5),
    'criterion': ['gini', 'entropy']
}

rf_random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_distributions=rf_random_grid,
    n_iter=50,
    cv=5,
    scoring='accuracy',
    random_state=42
)

rf_random_search.fit(X_train, y_train)

print("Best Random Forest Parameters:")
print(rf_random_search.best_params_)
print(f"Best CV Score: {rf_random_search.best_score_:.4f}")

# Logistic Regression Hyperparameter Tuning
lr_random_grid = {
    'C': uniform(0.001, 10),  # Regularization strength
    'penalty': ['l2'],
    'solver': ['lbfgs', 'newton-cg'],
    'max_iter': [500, 1000, 1500]
}

lr_random_search = RandomizedSearchCV(
    estimator=LogisticRegression(random_state=42),
    param_distributions=lr_random_grid,
    n_iter=50,
    cv=5,
    scoring='accuracy',
    random_state=42
)

lr_random_search.fit(X_train, y_train)

print("\nBest Logistic Regression Parameters:")
print(lr_random_search.best_params_)
print(f"Best CV Score: {lr_random_search.best_score_:.4f}")
```

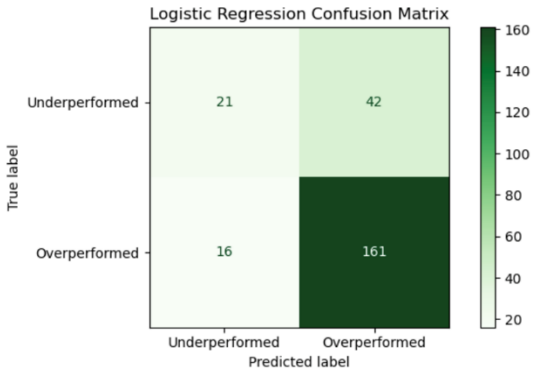
```
Best Random Forest Parameters:
{'criterion': 'gini', 'max_depth': 16, 'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 179}
Best CV Score: 0.8172

Best Logistic Regression Parameters:
{'C': 3.746401188473625, 'max_iter': 500, 'penalty': 'l2', 'solver': 'lbfgs'}
Best CV Score: 0.7544
```

```python
# Apply Best Parameters for Random Forest
best_rf = RandomForestClassifier(random_state=42, **rf_random_search.best_params_)
best_rf.fit(X_train, y_train)
rf_best_pred = best_rf.predict(X_test)

print("\nRandom Forest Tuned Results:")
print(f"Accuracy: {accuracy_score(y_test, rf_best_pred)}")
print(classification_report(y_test, rf_best_pred))

# Apply Best Parameters for Logistic Regression
best_lr = LogisticRegression(random_state=42, **lr_random_search.best_params_)
best_lr.fit(X_train, y_train)
lr_best_pred = best_lr.predict(X_test)

print("\nLogistic Regression Tuned Results:")
print(f"Accuracy: {accuracy_score(y_test, lr_best_pred)}")
print(classification_report(y_test, lr_best_pred))
```

```
Random Forest Tuned Results:
Accuracy: 0.8
              precision    recall  f1-score   support

          -1       0.67      0.48      0.56        63
           1       0.83      0.92      0.87       177

    accuracy                           0.80       240
   macro avg       0.75      0.70      0.71       240
weighted avg       0.79      0.80      0.79       240


Logistic Regression Tuned Results:
Accuracy: 0.7583333333333333
              precision    recall  f1-score   support

          -1       0.57      0.33      0.42        63
           1       0.79      0.91      0.85       177

    accuracy                           0.76       240
   macro avg       0.68      0.62      0.63       240
weighted avg       0.73      0.76      0.74       240
```

Visualizing of the results:






```
Random Forest Classification Report:
                precision    recall  f1-score   support

Underperformed       0.67      0.48      0.56        63
 Overperformed       0.83      0.92      0.87       177

      accuracy                           0.80       240
     macro avg       0.75      0.70      0.71       240
  weighted avg       0.79      0.80      0.79       240


Logistic Regression Classification Report:
                precision    recall  f1-score   support

Underperformed       0.57      0.33      0.42        63
 Overperformed       0.79      0.91      0.85       177

      accuracy                           0.76       240
     macro avg       0.68      0.62      0.63       240
  weighted avg       0.73      0.76      0.74       240
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

# Create a side-by-side plot
plt.figure(figsize=(14, 4))

true_label = ['Underperformed', 'Overperformed']

# Random Forest Confusion Matrix
plt.subplot(1, 2, 1)
ConfusionMatrixDisplay.from_predictions(
   y_test,
   rf_best_pred,
   cmap='Blues',
   display_labels=true_label,
   ax=plt.gca()
)
plt.title('Random Forest Confusion Matrix')

# Logistic Regression Confusion Matrix
plt.subplot(1, 2, 2)
ConfusionMatrixDisplay.from_predictions(
   y_test,
   lr_best_pred,
   cmap='Greens',
   display_labels=true_label,
   ax=plt.gca()
)
plt.title('Logistic Regression Confusion Matrix')

plt.tight_layout()
plt.show()

# Print Classification Reports
print("Random Forest Classification Report:")
print(classification_report(y_test, rf_best_pred, target_names=true_label))

print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, lr_best_pred, target_names=true_label))
```

Results analysis and discussion

Model performance was evaluated using accuracy, confusion matrix, and the classification report (precision, recall, F1-score). Accuracy measures overall correctness, while the confusion matrix details true/false positives and negatives. Precision, recall, and F1-score

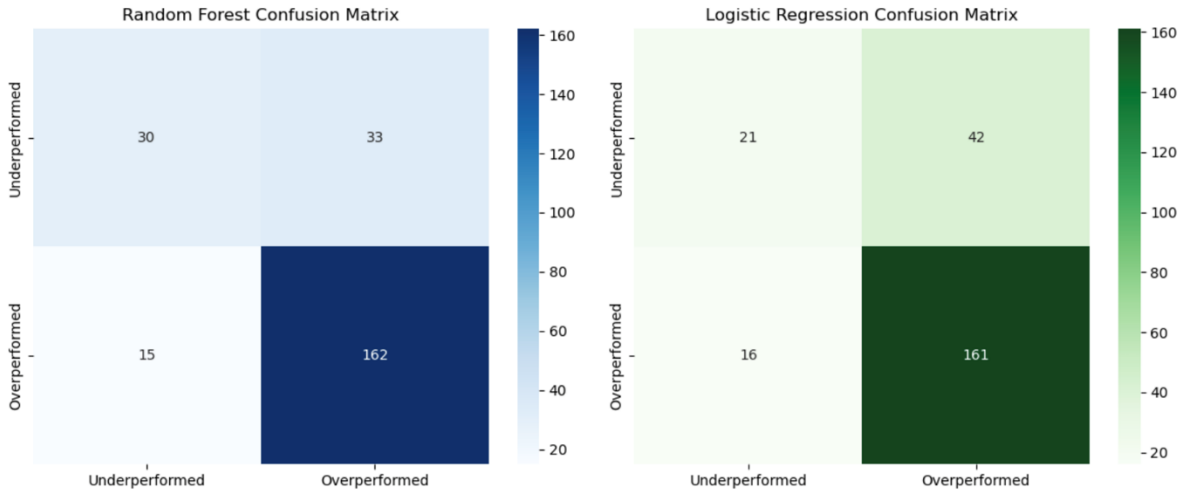
provide deeper insights into class-specific performance, essential for binary classification (Sokolova & Lapalme, 2009).

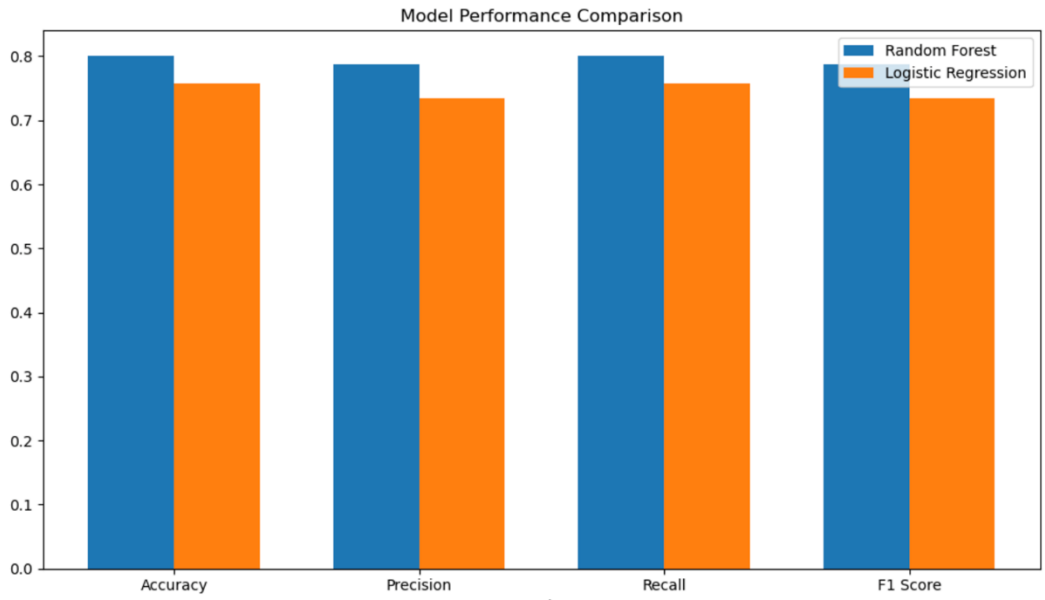
Results Presentation and Comparison:

The Random Forest model demonstrated robust performance, achieving an accuracy of 80%. It performed well across all classes, with precision, recall, and F1-score values close to or exceeding 0.7 for most classes. This indicates a balanced performance, making it well-suited for this classification task.

Logistic Regression, with an accuracy of 75%, underperformed compared to the Random Forest model. It exhibited greater variability in its ability to correctly classify instances, as evidenced by lower recall and F1-scores.

```
Performance Metrics Comparison:
Accuracy: Random Forest = 0.8000, Logistic Regression = 0.7583
Precision: Random Forest = 0.7877, Logistic Regression = 0.7339
Recall: Random Forest = 0.8000, Logistic Regression = 0.7583
F1 Score: Random Forest = 0.7882, Logistic Regression = 0.7352
```





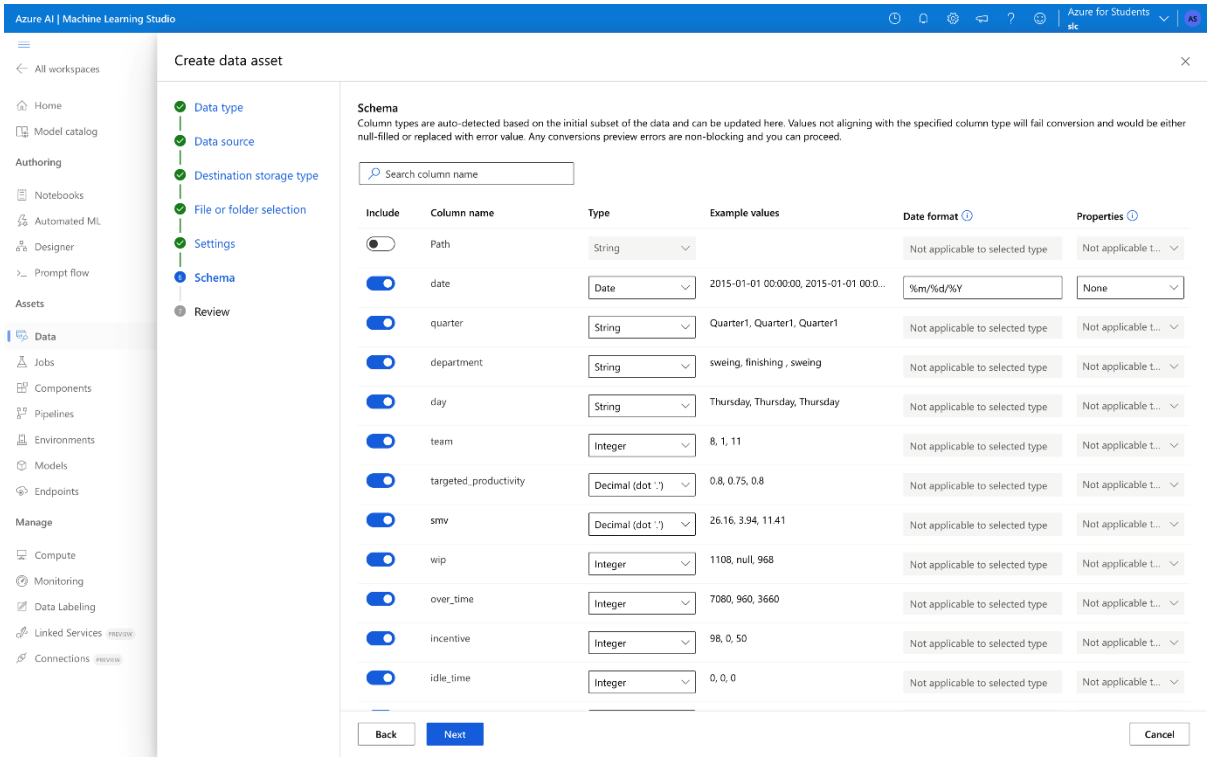
EDA and Data preprocessing (Part b)

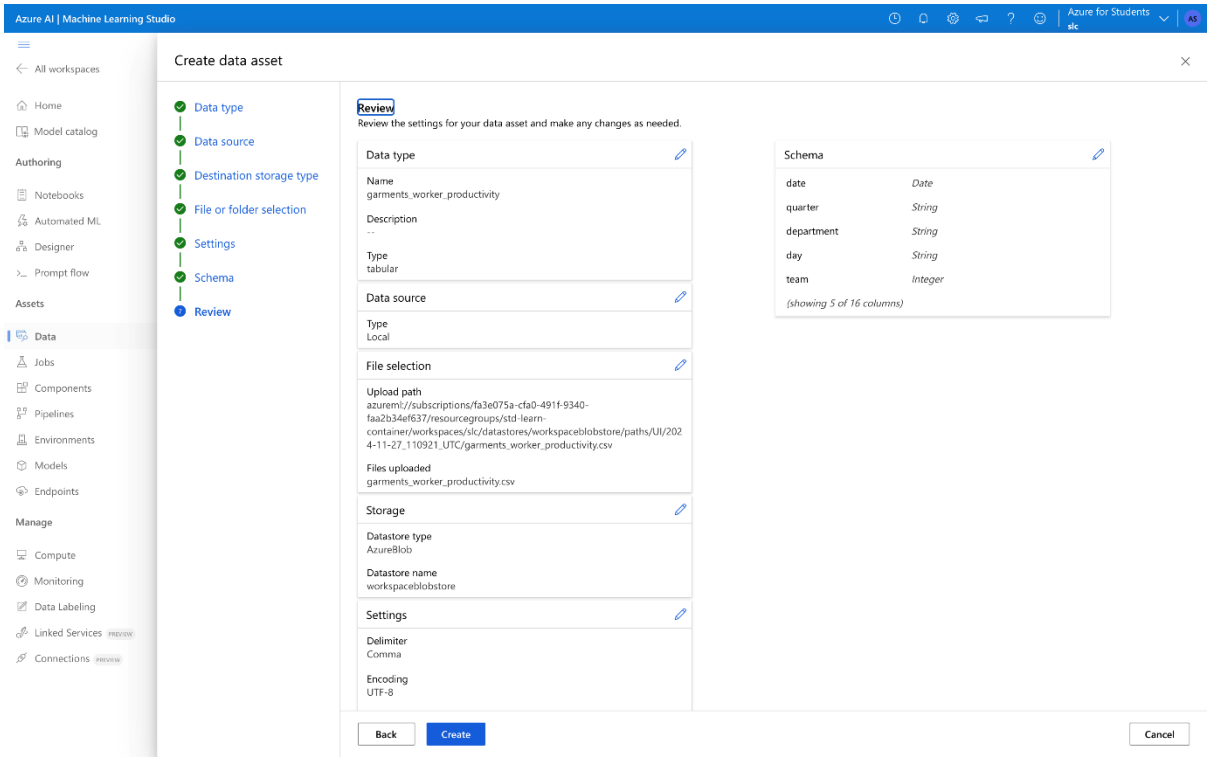
As we have already done EDA for this dataset, we will just be doing the preprocessing in this step in azure.

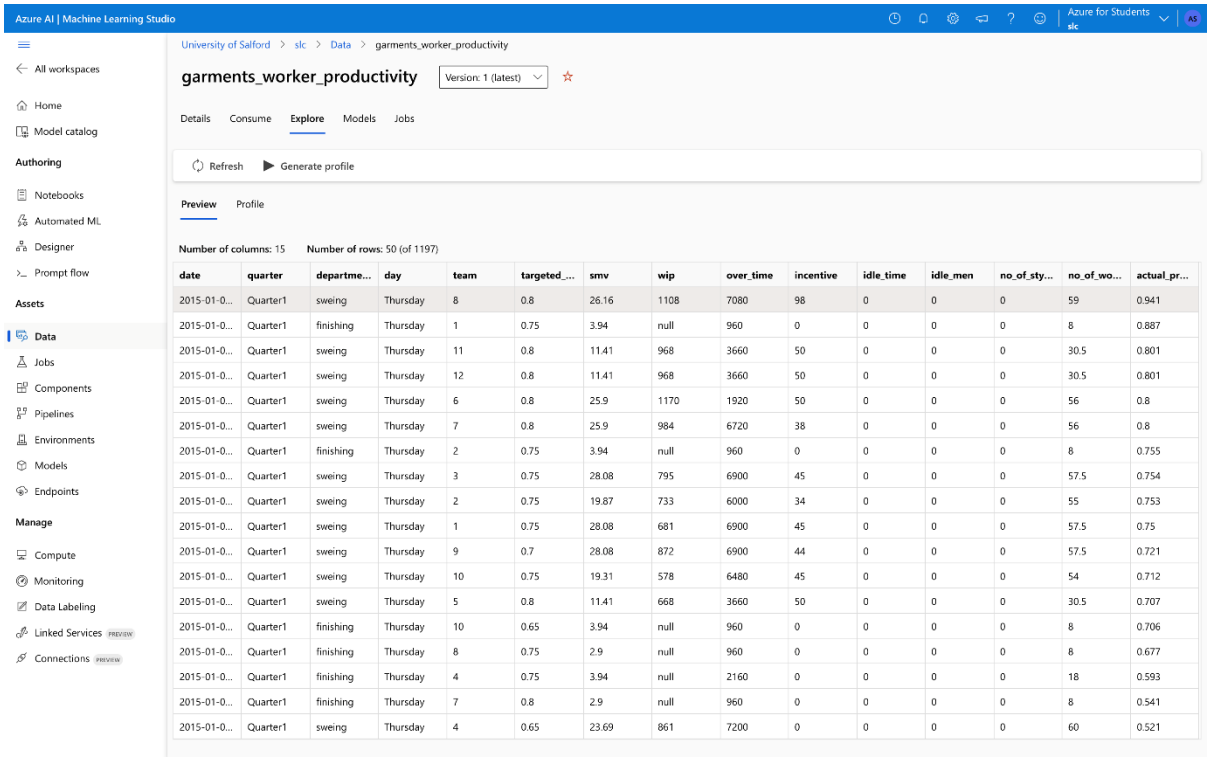
Previous steps: We already created workspace and compute for our workshop task, so I am using the same.

1. *Loading Dataset*

We loaded the dataset by going to Assets>Data and created new data by using “CREAT” button and set up the correct data type of our columns.


Azure AI | Machine Learning Studio
Azure for Students
slc
All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services PREVIEW
Connections PREVIEW
Create data asset
Data type
Data source
Destination storage type
File or folder selection
Settings
Schema
Review
Schema
Column types are auto-detected based on the initial subset of the data and can be updated here. Values not aligning with the specified column type will fail conversion and would be either null-filled or replaced with error value. Any conversions preview errors are non-blocking and you can proceed.
Search column name
Include
Column name
Type
Example values
Date format
Properties
Path
String
Not applicable to selected type
Not applicable t...
date
Date
2015-01-01 00:00:00, 2015-01-01 00:0...
%m/%d/%Y
None
quarter
String
Quarter1, Quarter1, Quarter1
department
String
sweing, finishing , sweing
day
String
Thursday, Thursday, Thursday
team
Integer
8, 1, 11
targeted_productivity
Decimal (dot '.')
0.8, 0.75, 0.8
smv
Decimal (dot '.')
26.16, 3.94, 11.41
wip
Integer
1108, null, 968
over_time
Integer
7080, 960, 3660
incentive
Integer
98, 0, 50
idle_time
Integer
0, 0, 0
Back
Next
Cancel


Azure AI | Machine Learning Studio
Azure for Students
slc
All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services PREVIEW
Connections PREVIEW
Create data asset
Data type
Data source
Destination storage type
File or folder selection
Settings
Schema
Review
Review
Review the settings for your data asset and make any changes as needed.
Data type
Name
garments_worker_productivity
Description
--
Type
tabular
Data source
Type
Local
File selection
Upload path
azureml://subscriptions/fa3e075a-cfa0-491f-9340-faa2b34ef637/resourcegroups/std-learn-container/workspaces/slc/datastores/workspaceblobstore/paths/UI/2024-11-27_110921_UTC/garments_worker_productivity.csv
Files uploaded
garments_worker_productivity.csv
Storage
Datastore type
AzureBlob
Datastore name
workspaceblobstore
Settings
Delimiter
Comma
Encoding
UTF-8
Schema
date Date
quarter String
department String
day String
team Integer
(showing 5 of 16 columns)
Back
Create
Cancel

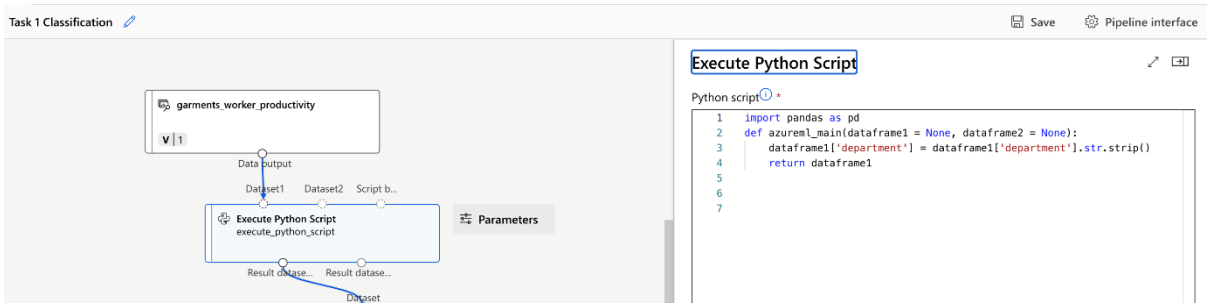


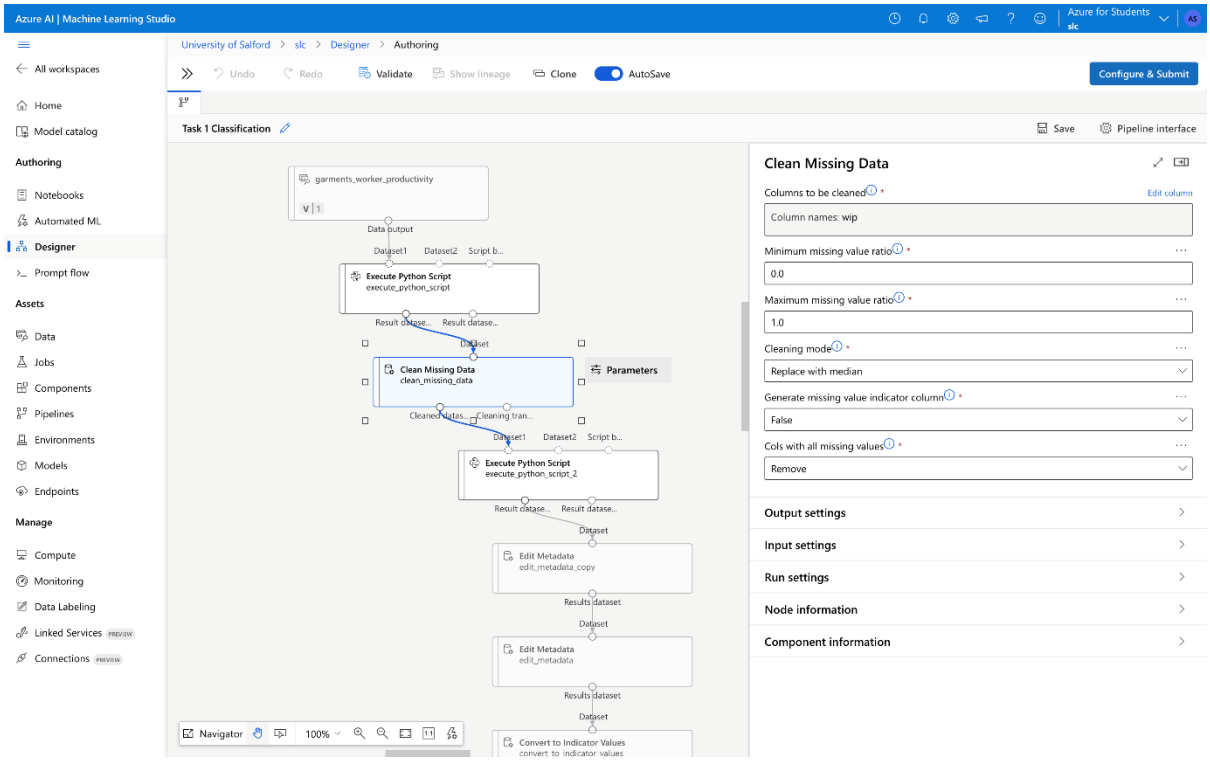
date	quarter	departme...	day	team	targeted_...	smv	wip	over_time	incentive	idle_time	idle_men	no_of_sty...	no_of_wo...	actual_pr...
2015-01-0...	Quarter1	sweing	Thursday	8	0.8	26.16	1108	7080	98	0	0	0	59	0.941
2015-01-0...	Quarter1	finishing	Thursday	1	0.75	3.94	null	960	0	0	0	0	8	0.887
2015-01-0...	Quarter1	sweing	Thursday	11	0.8	11.41	968	3660	50	0	0	0	30.5	0.801
2015-01-0...	Quarter1	sweing	Thursday	12	0.8	11.41	968	3660	50	0	0	0	30.5	0.801
2015-01-0...	Quarter1	sweing	Thursday	6	0.8	25.9	1170	1920	50	0	0	0	56	0.8
2015-01-0...	Quarter1	sweing	Thursday	7	0.8	25.9	984	6720	38	0	0	0	56	0.8
2015-01-0...	Quarter1	finishing	Thursday	2	0.75	3.94	null	960	0	0	0	0	8	0.755
2015-01-0...	Quarter1	sweing	Thursday	3	0.75	28.08	795	6900	45	0	0	0	57.5	0.754
2015-01-0...	Quarter1	sweing	Thursday	2	0.75	19.87	733	6000	34	0	0	0	55	0.753
2015-01-0...	Quarter1	sweing	Thursday	1	0.75	28.08	681	6900	45	0	0	0	57.5	0.75
2015-01-0...	Quarter1	sweing	Thursday	9	0.7	28.08	872	6900	44	0	0	0	57.5	0.721
2015-01-0...	Quarter1	sweing	Thursday	10	0.75	19.31	578	6480	45	0	0	0	54	0.712
2015-01-0...	Quarter1	sweing	Thursday	5	0.8	11.41	668	3660	50	0	0	0	30.5	0.707
2015-01-0...	Quarter1	finishing	Thursday	10	0.65	3.94	null	960	0	0	0	0	8	0.706
2015-01-0...	Quarter1	finishing	Thursday	8	0.75	2.9	null	960	0	0	0	0	8	0.677
2015-01-0...	Quarter1	finishing	Thursday	4	0.75	3.94	null	2160	0	0	0	0	18	0.593
2015-01-0...	Quarter1	finishing	Thursday	7	0.8	2.9	null	960	0	0	0	0	8	0.541
2015-01-0...	Quarter1	sweing	Thursday	4	0.65	23.69	861	7200	0	0	0	0	60	0.521

2. *Handling Values (Cleaning + Missing):*

After data loading, we went to the Designer Tab by Authoring>Designer. Designer is the main Screen where we will perform our whole task.

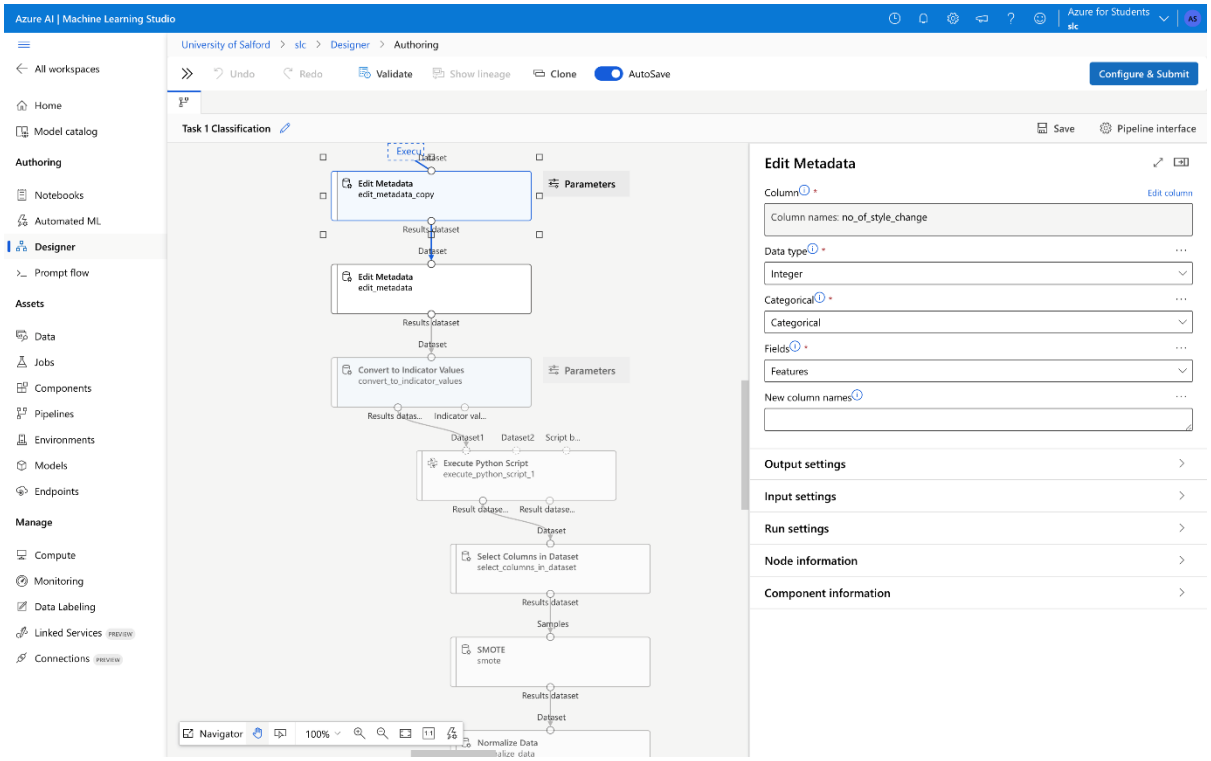
In EDA as earlier found an error in the department column and we fixed it by converted “finishing” to “finishing” (note the space) using “Execute Python Script” component and imputed “wip” columns missing values with median using “Clean Missing Data” component in Designer.



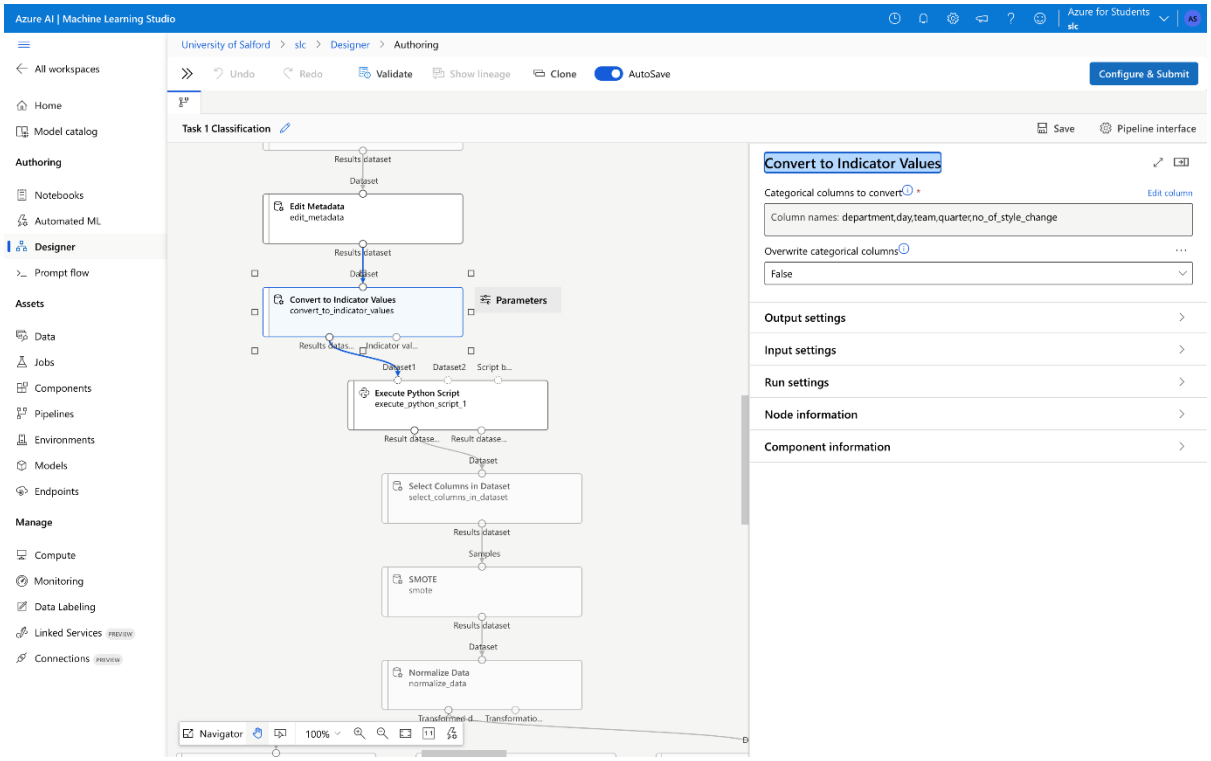


3. Encoding Categorical Features:

Label Encoding: Applied to no_of_style_change as it represents categorical or ordinal variables using “Edit Metadata” component.

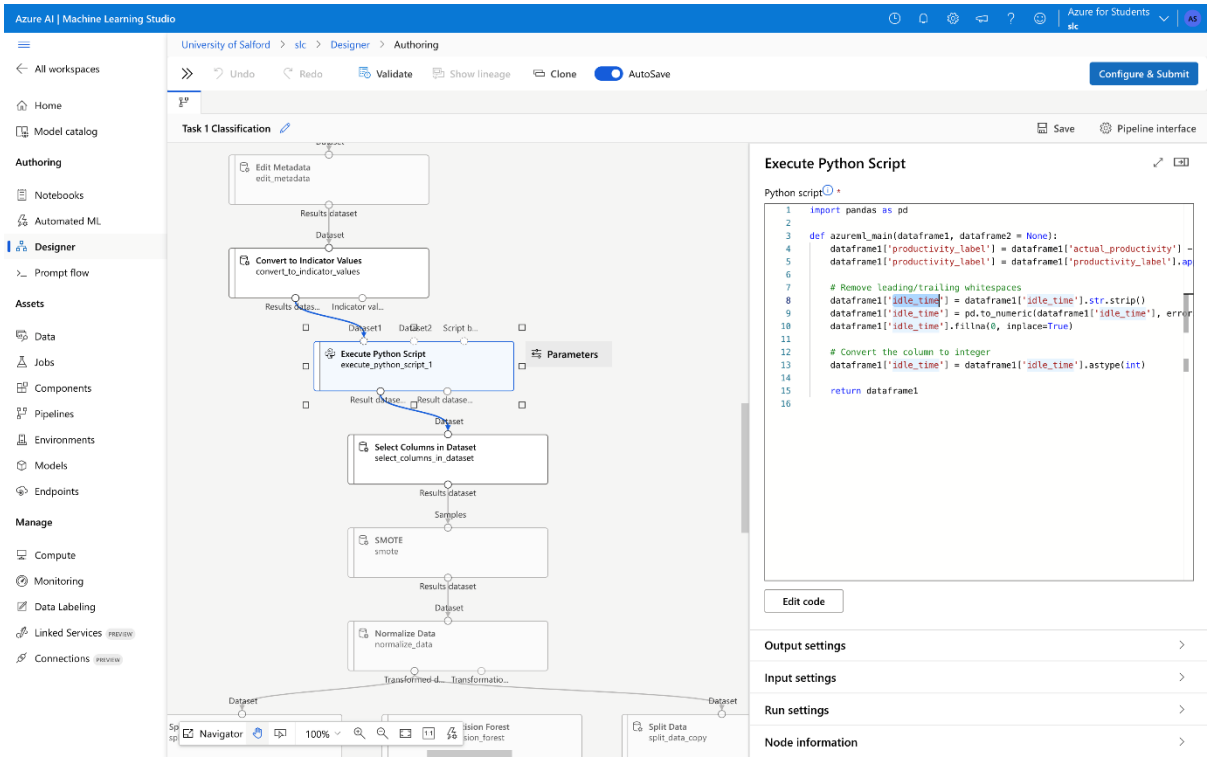


One-Hot Encoding: Applied to department, day, and team since they are nominal features with no inherent order using “Convert to Indicator Value component.”



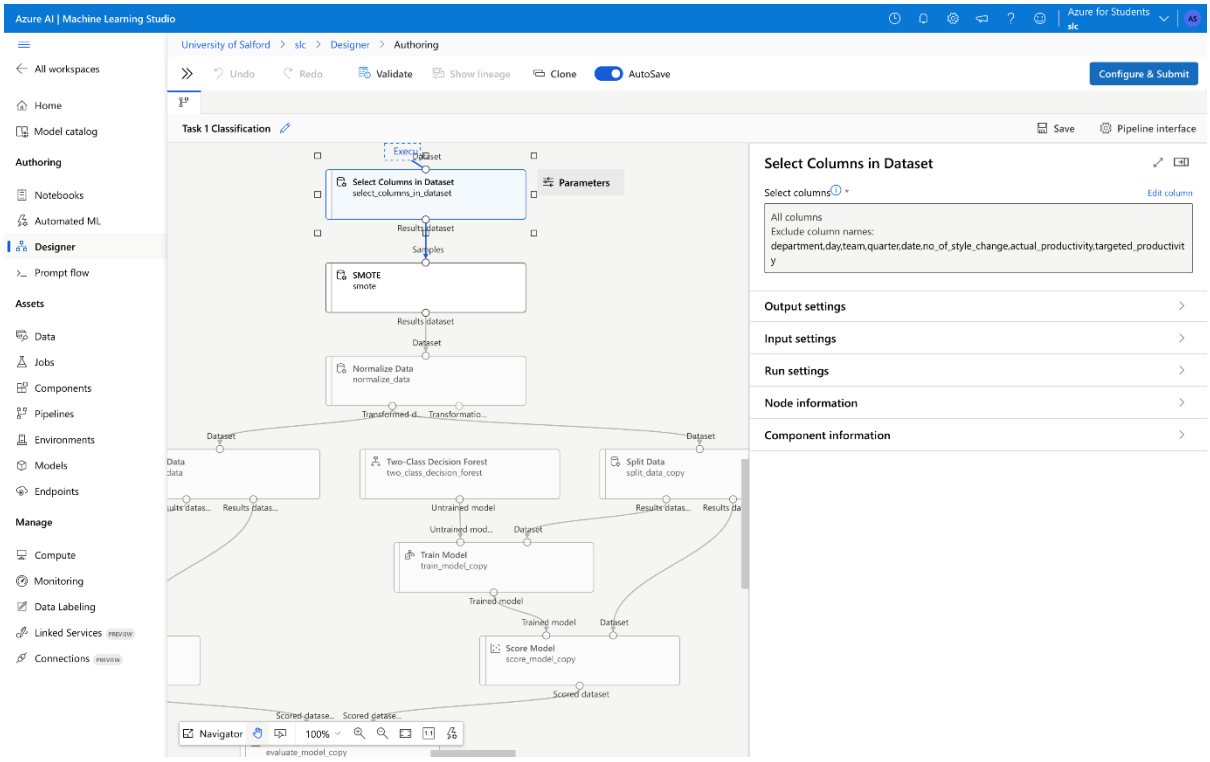
5. *Creating Binary Class’s and Addressing Class Imbalance with SMOTE:*

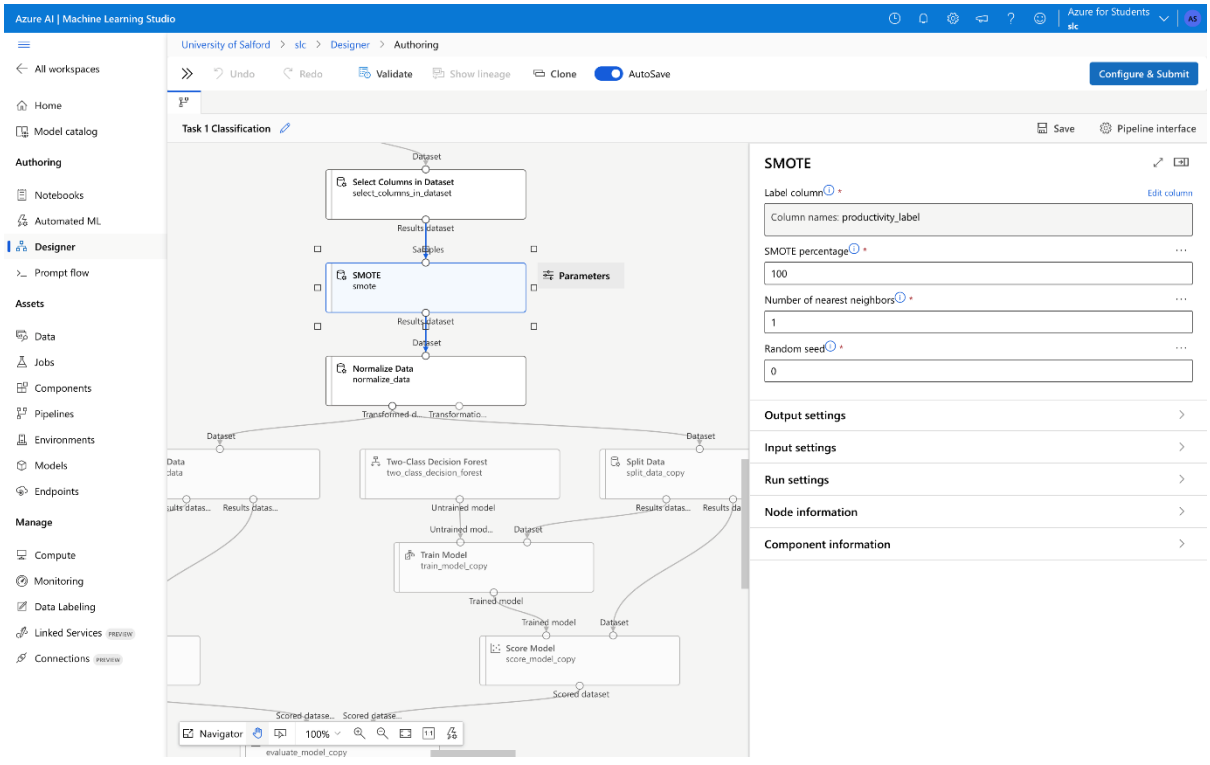
'productivity_label' target variable is created using “Execute Python Script” component and column type of “idle_time” converted to Numeric as somehow it was changed back to String.



3. *Addressing Class Imbalance with SMOTE:*

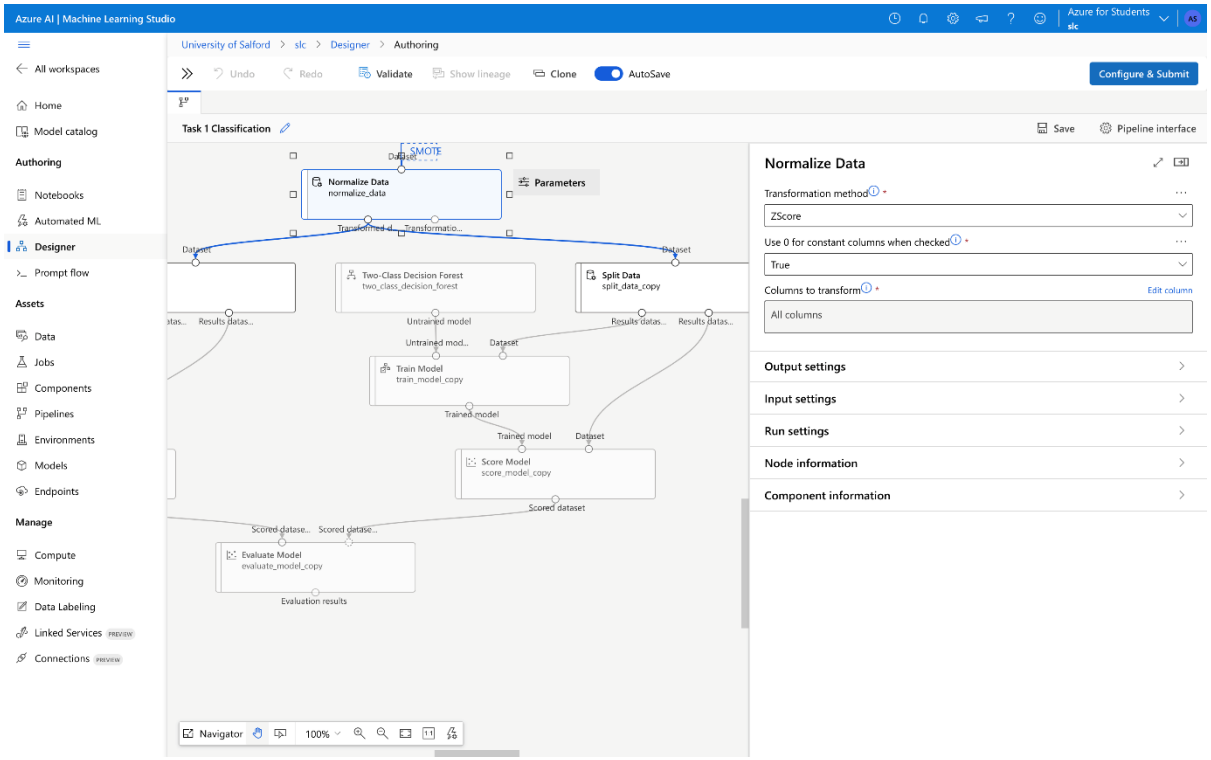
In the next steps we dropped columns which we' did not need and applied SMOTE using "SMOTE" component to balance our data.





4. Normalized Data

We normalized our data using “Normalize Data” component and with “ZScore” this time, whereas we used the “Standardizer” in python. We decided to use the **Normalizer** this time to scale the data into a unit vector, aiming to test if it improves the model's performance.


Azure AI | Machine Learning Studio
Azure for Students
slc
University of Salford > slc > Designer > Authoring
All workspaces
Undo
Redo
Validate
Show lineage
Clone
AutoSave
Configure & Submit
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services
Connections
Task 1 Classification
Save
Pipeline interface
SMOTE
Normalize Data
normalize_data
Parameters
Two-Class Decision Forest
two_class_decision_forest
Split Data
split_data_copy
Train Model
train_model_copy
Score Model
score_model_copy
Evaluate Model
evaluate_model_copy
Evaluation results
Navigator
100%
Normalize Data
Transformation method
ZScore
Use 0 for constant columns when checked
True
Columns to transform
Edit column
All columns
Output settings
Input settings
Run settings
Node information
Component information

Implementation (Part b)

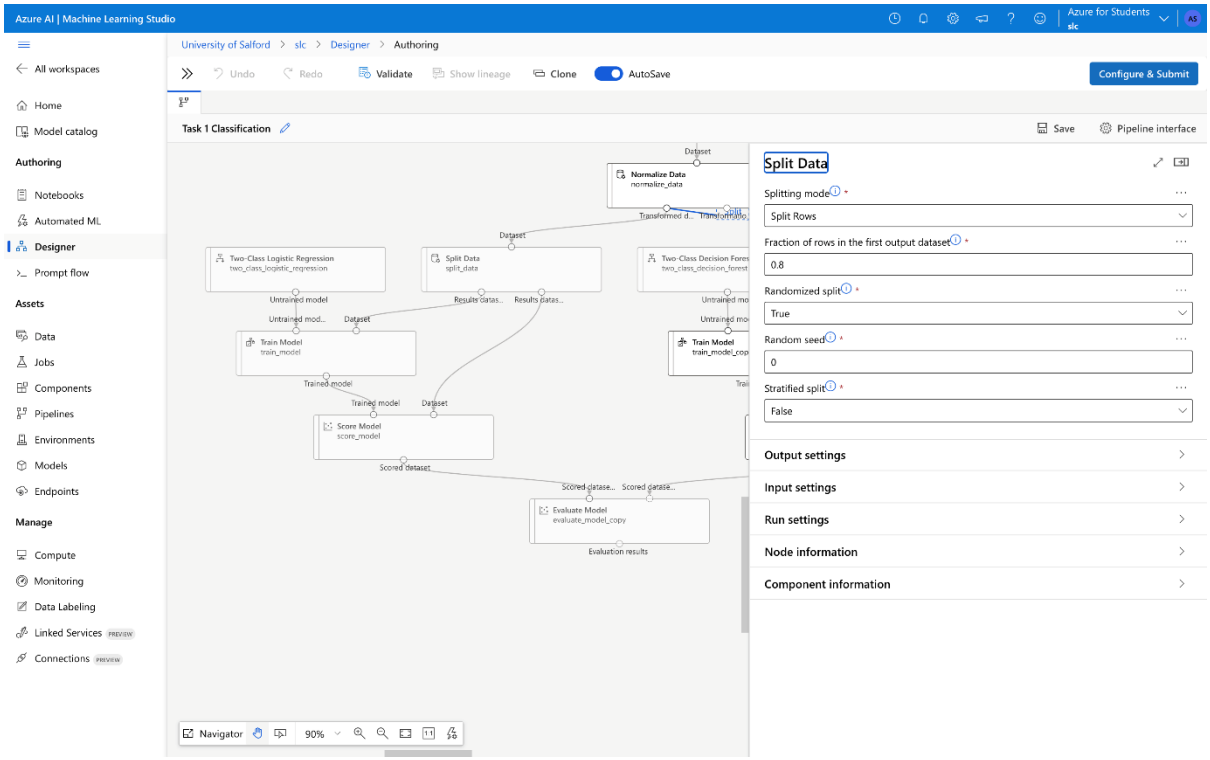
Description and Justification of Algorithms Used

Two-Class Decision Forest and **Random Forest Classifier** are interchangeable terms in the context of binary classification. So, we used **Two-Class Decision Forest** and other models used are Logistic regression. We have already written about why we selected these two models for this problem.

Experimental Procedure

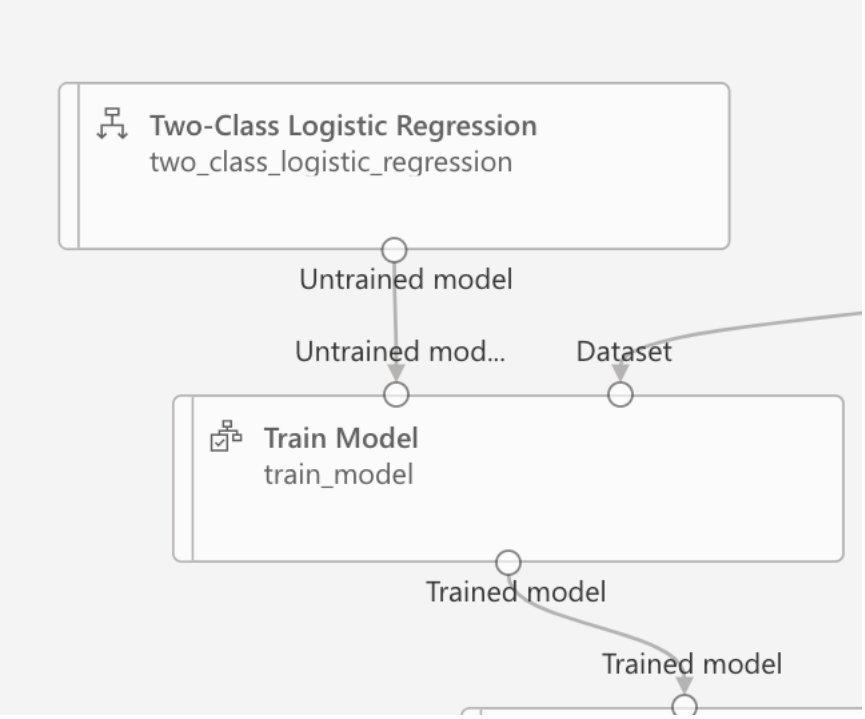
1. Data Scaling and Splitting:

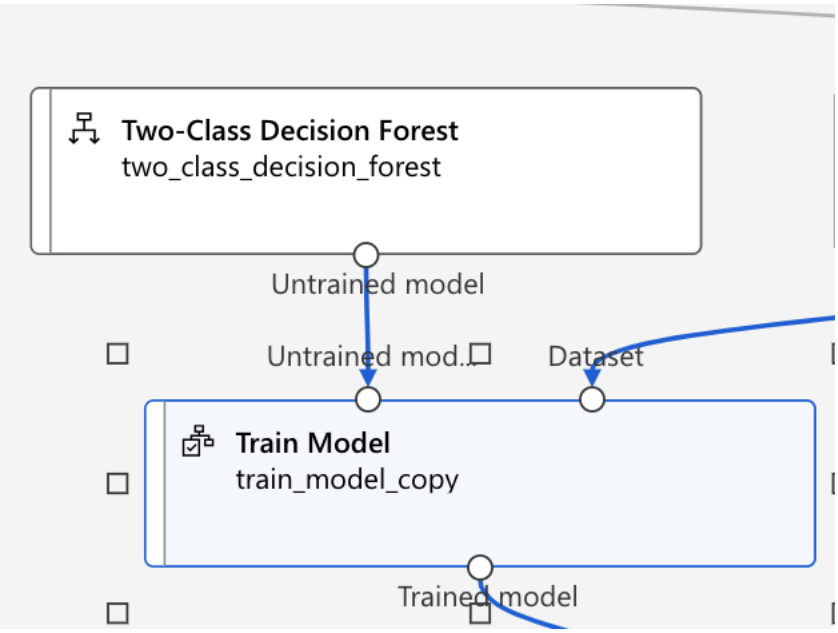
The dataset was split into training (80%) and testing (20%) sets using the Two "Split Data" component as our goal is to compare the performance of two models. We will use this 80% training dataset in the "Train Model" and testing dataset in the "Score Model" components.

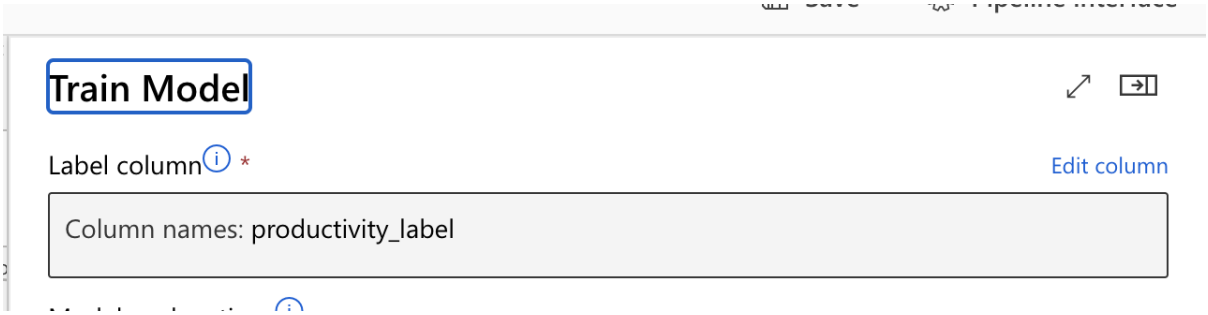


2. Training and Scoring Models.

After splitting the dataset, we used Two Model Components "Two-Class Logistic regression" and "Two Class Decision Forests" and connected their outputs to Two "Train Model" components and selected the Label Column as "productivity_label" in both Train Model components. We also connect the training data that we got from split data components to these "Train Model" components.

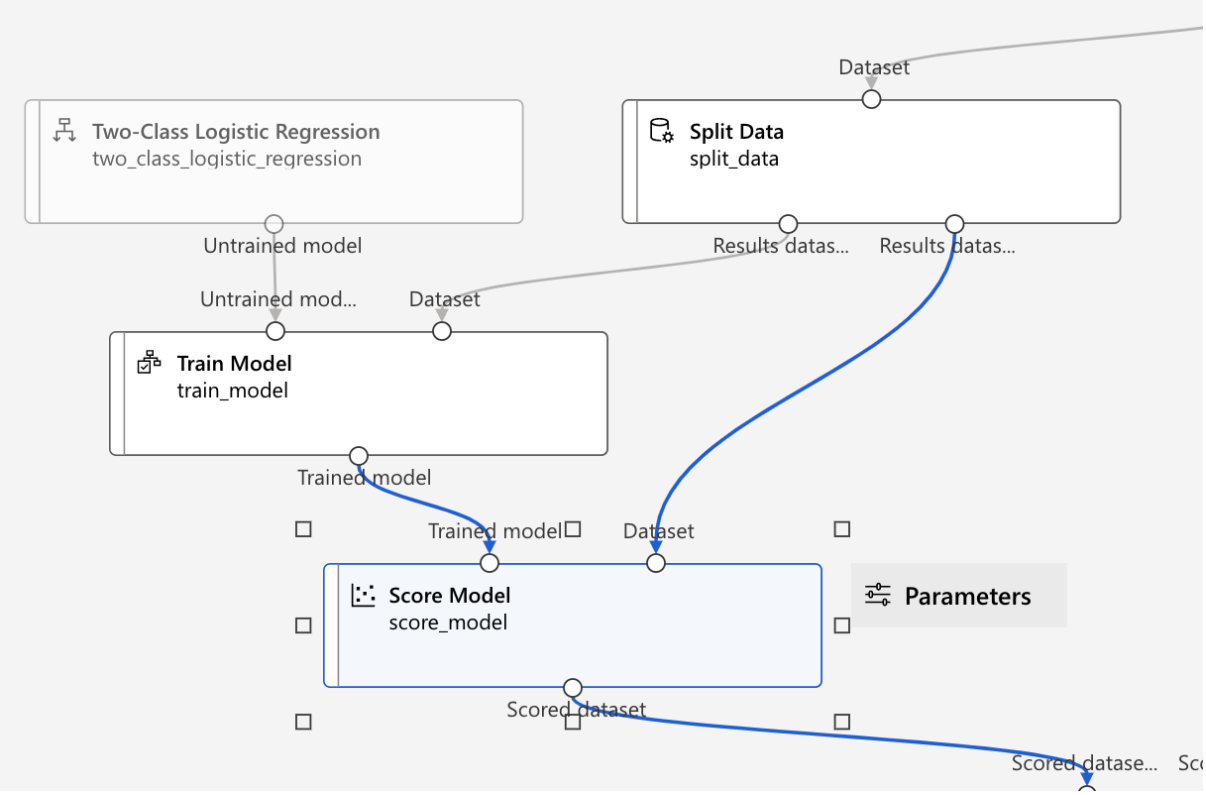


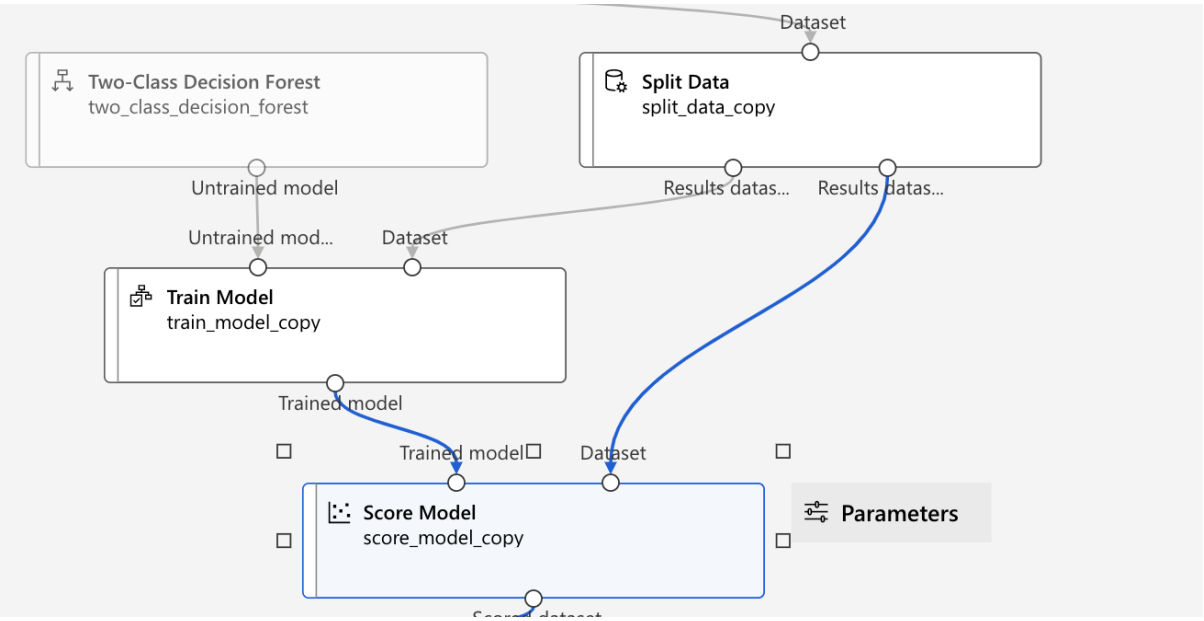




Now to score our training models. we used Two “Score Model” components. The “Score Model” requires test data and a trained model as inputs. We joined the test data that we got

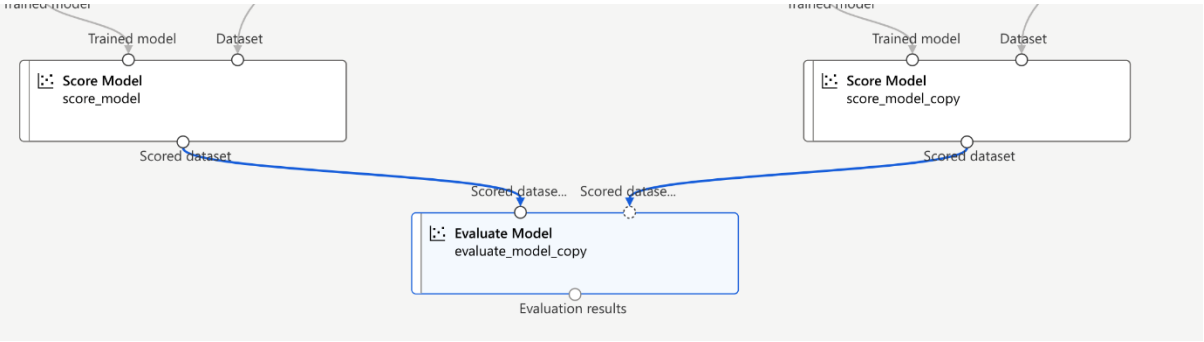
from “Split Model” and trained model from “Train Model” to this as we did this for both of our models.





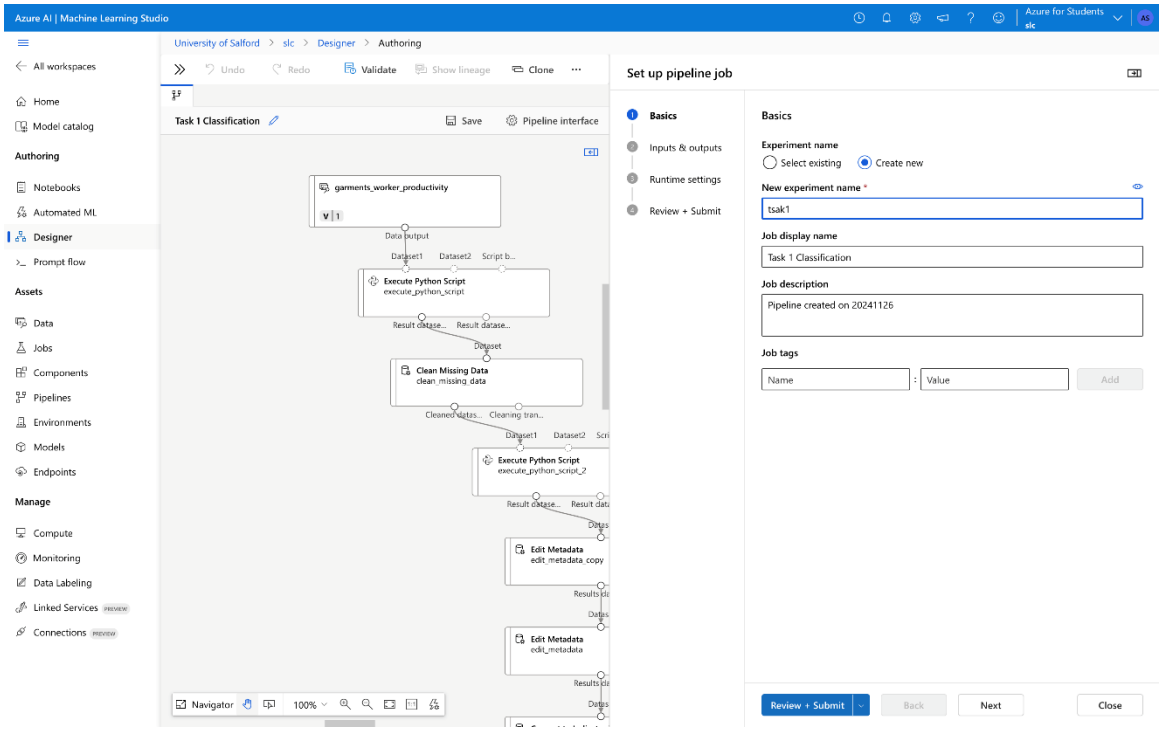
3. Evaluating Models

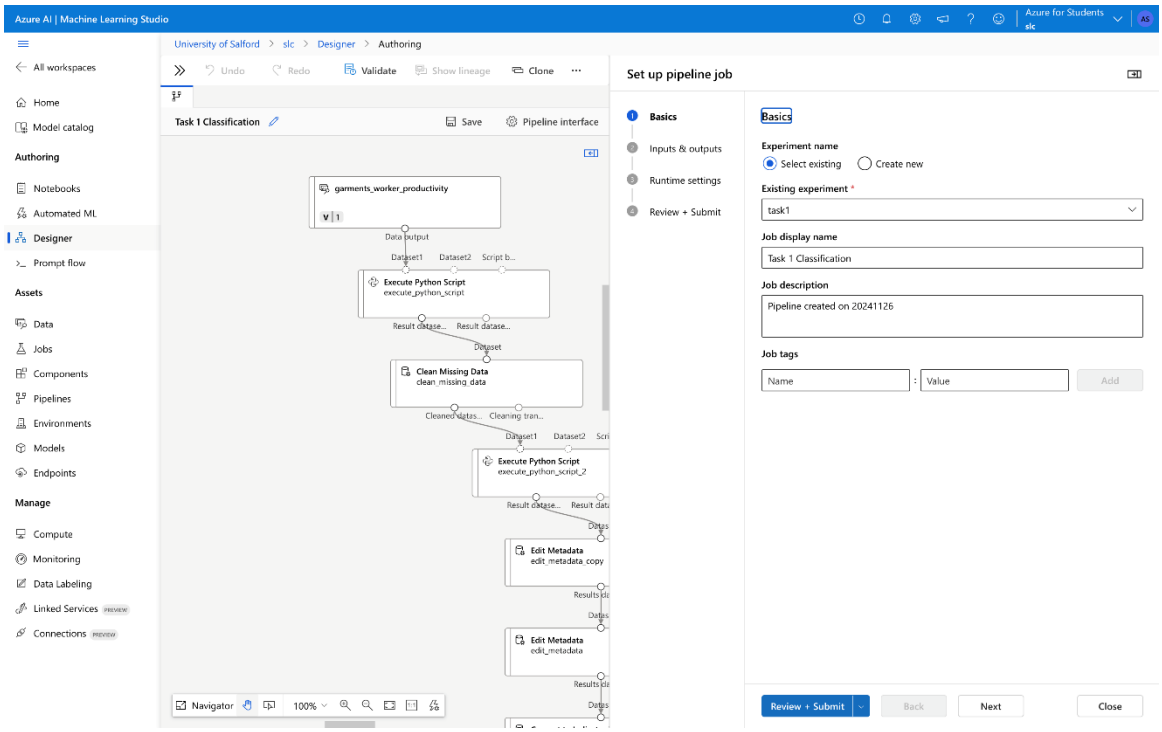
For evaluating models, we used single “Evaluate Model” and connected the output of our two “Scor Model” to it.



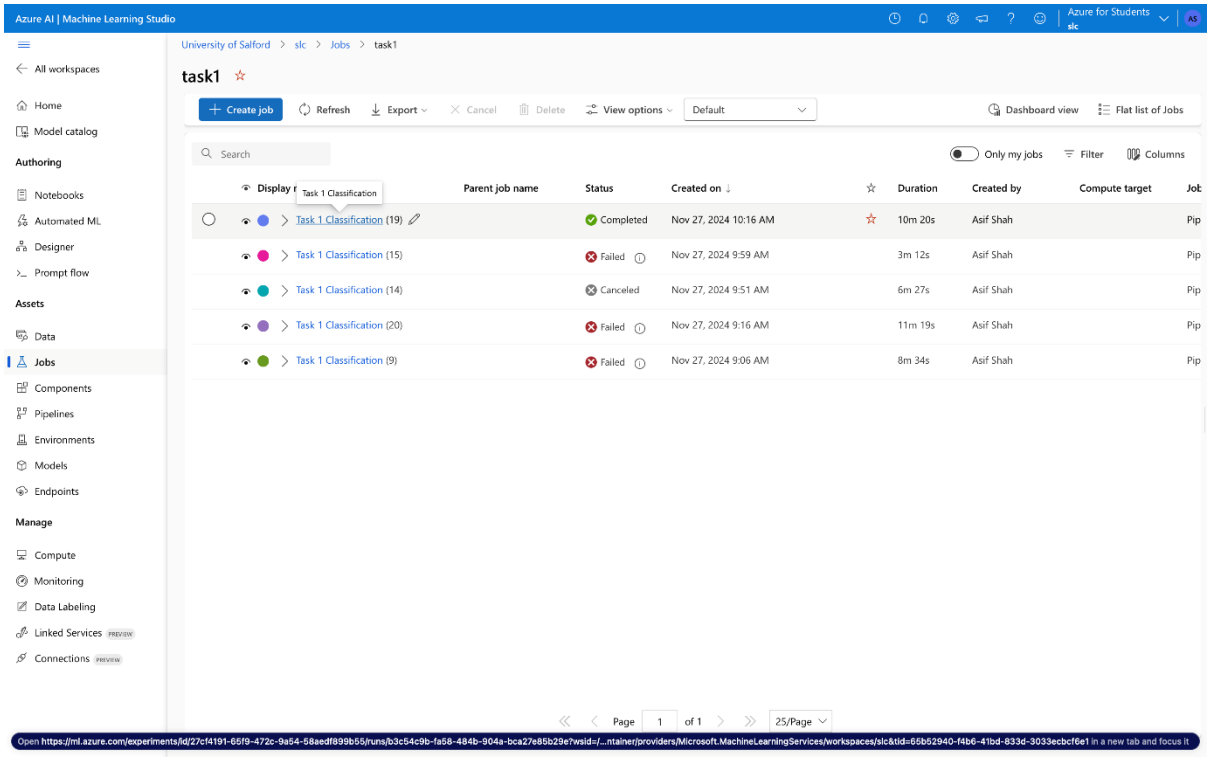
4. Running Pipeline and Viewing results

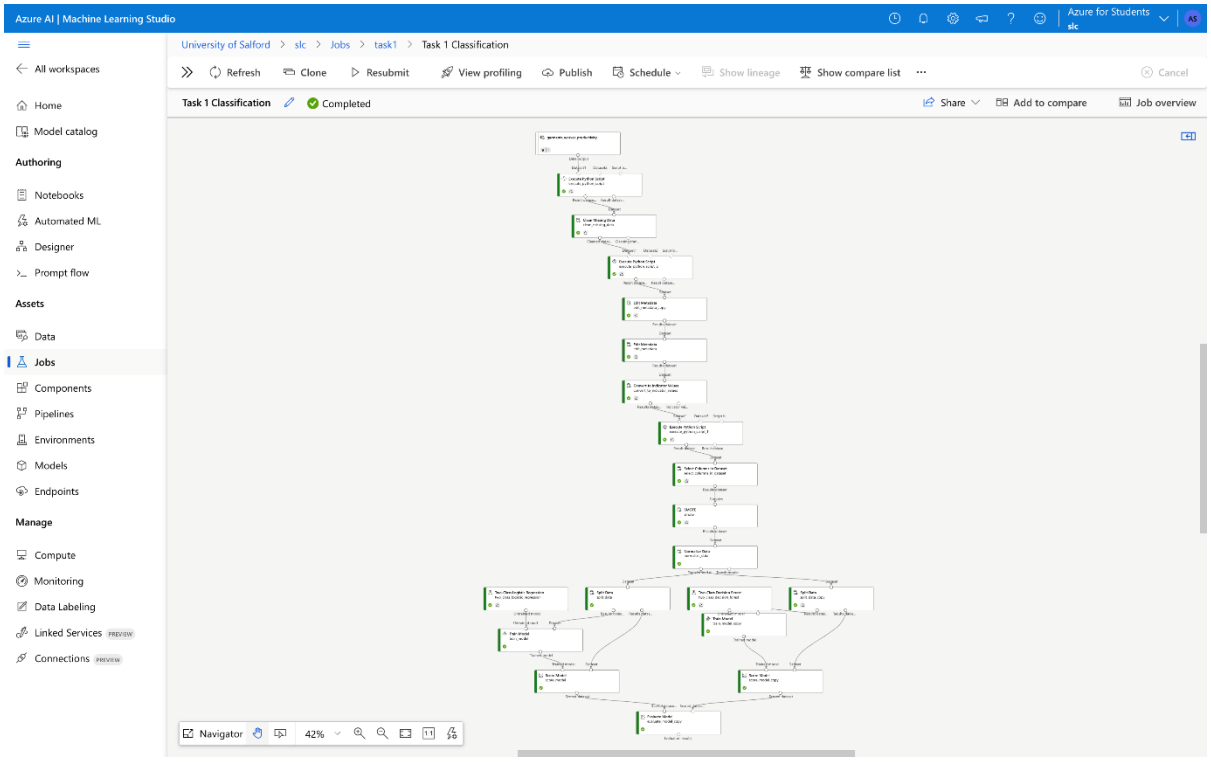
After all these steps we needed to run the Pipeline to execute the job. For this I head to top right corner of Designer and clicked “Configure and Submit,” created new “Experiment” and selected compute “Task1” and after the job is completed.

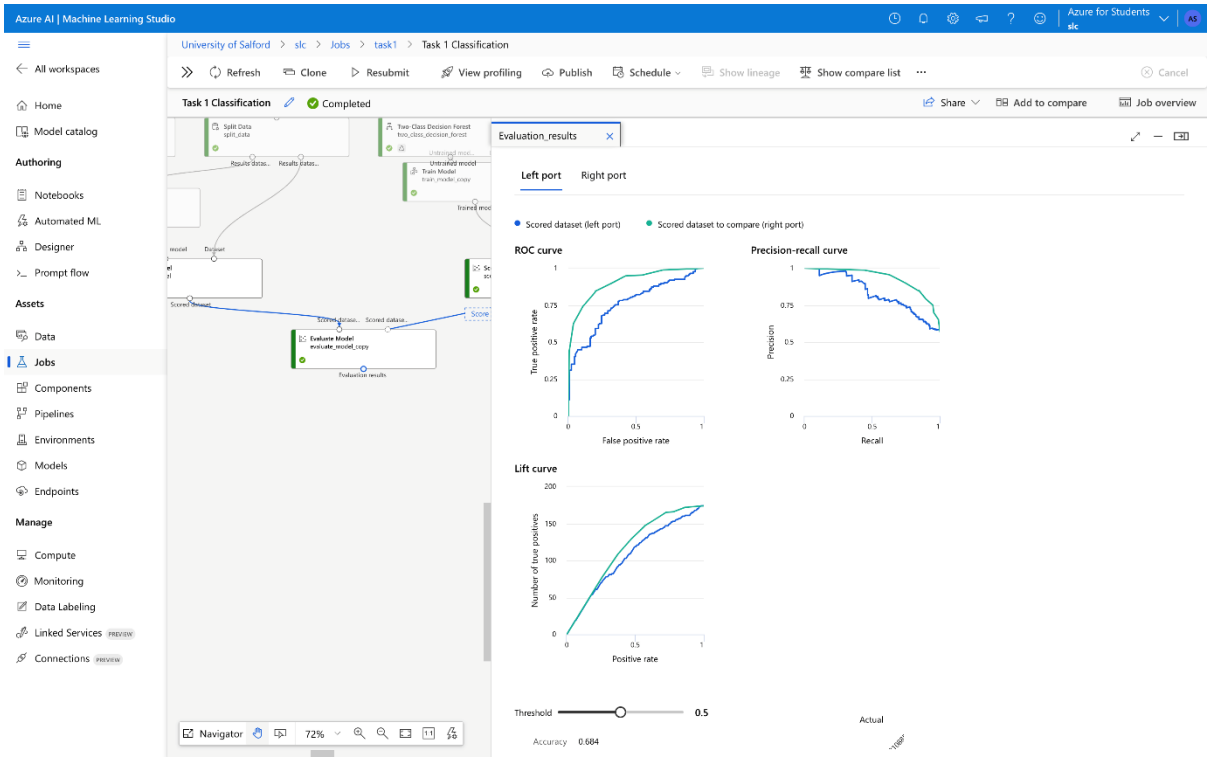




To view the results, I went to Assets>Job selected my job and upon right click on “Evaluation Model” got the results.

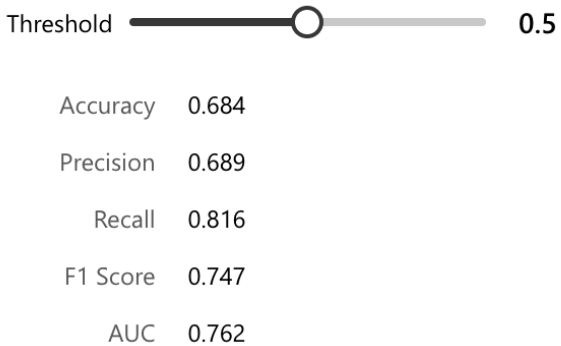

Azure AI | Machine Learning Studio
Azure for Students
slc
All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services PREVIEW
Connections PREVIEW
University of Salford > slc > Jobs > task1
task1
Create job
Refresh
Export
Cancel
Delete
View options
Default
Dashboard view
Flat list of Jobs
Search
Only my jobs
Filter
Columns
Display
Task 1 Classification
Parent job name
Status
Created on
Duration
Created by
Compute target
Job
Task 1 Classification (19)
Completed
Nov 27, 2024 10:16 AM
10m 20s
Asif Shah
Pip
Task 1 Classification (15)
Failed
Nov 27, 2024 9:59 AM
3m 12s
Asif Shah
Pip
Task 1 Classification (14)
Canceled
Nov 27, 2024 9:51 AM
6m 27s
Asif Shah
Pip
Task 1 Classification (20)
Failed
Nov 27, 2024 9:16 AM
11m 19s
Asif Shah
Pip
Task 1 Classification (9)
Failed
Nov 27, 2024 9:06 AM
8m 34s
Asif Shah
Pip
Page 1 of 1
25/Page
Open https://ml.azure.com/experiments/id/27cf4191-65f9-472c-9a54-58aedf899b55/runs/b3c54c9b-fa58-484b-904a-bca27e85b29e?wsid=/...ntainer/providers/Microsoft.MachineLearningServices/workspaces/slc&tid=65b52940-f4b6-41bd-833d-3033ecbcf6e1 in a new tab and focus it

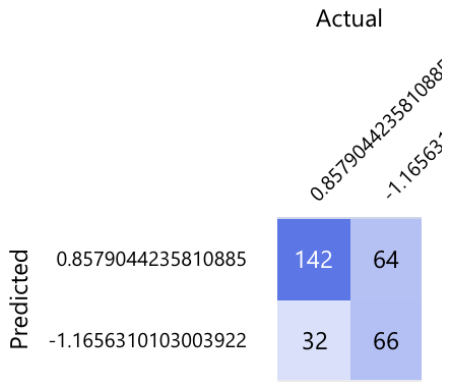

Azure AI | Machine Learning Studio
Azure for Students
slc
All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services PREVIEW
Connections PREVIEW
University of Salford > slc > Jobs > task1 > Task 1 Classification
Refresh
Clone
Resubmit
View profiling
Publish
Schedule
Show lineage
Show compare list
Cancel
Task 1 Classification
Completed
Share
Add to compare
Job overview
Navigator
42%



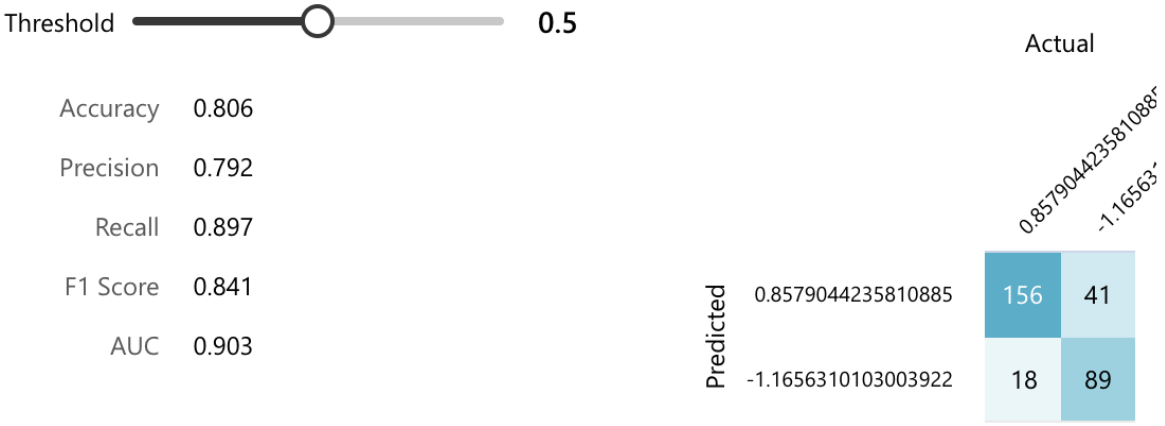
Results analysis and discussion

Two Class Logistic Regression:

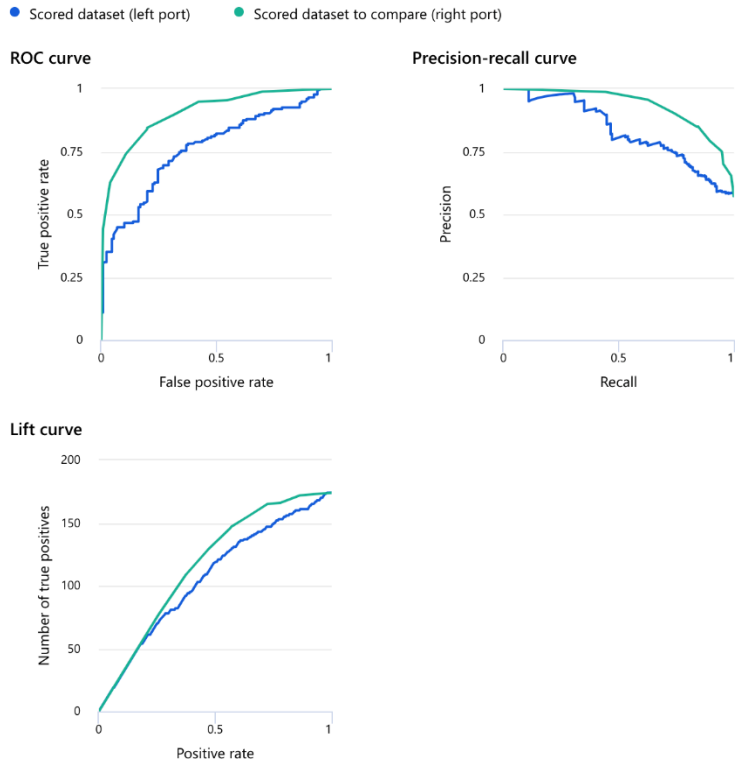




Two Class Decision Forests:



Combine Performance Overview:



To evaluate the performance of the models, we utilized accuracy, confusion matrix, and the classification report (precision, recall, F1-score).

Metric	**Decision Forests**	**Logistic Regression**
Accuracy	**0.806**	**0.684**
Precision	**0.792**	**0.689**
Recall	**0.897**	**0.816**

| **F1 Score** | **0.841** | **0.747** |
| **AUC** | **0.903** | **0.762** |

The Decision Forests model outperforms Logistic Regression across all metrics. It achieves higher accuracy (0.806 vs. 0.684), indicating more reliable overall predictions. The precision (0.792) suggests fewer false positives compared to Logistic Regression (0.689). Additionally, the Decision Forests model shows superior recall (0.897), capturing more positive cases than Logistic Regression (0.816).

The higher F1 score of Decision Forests reflects a better balance between precision and recall. Its significantly higher AUC (0.903 vs. 0.762) suggests stronger class separation, indicating more robust model discrimination.

In contrast, while Logistic Regression performs adequately, its lower scores across all metrics suggest potential limitations in handling complex relationships within the data. It may also indicate the need for more feature engineering or regularization.

Business Benefits:

The Random Forest model's superior accuracy and balanced performance make it highly effective for predicting productivity levels in the garment industry. It identifies key factors affecting employee productivity, enabling businesses to implement targeted interventions. This predictive approach allows organizations to:

- Optimize Resource Allocation: Allocate resources efficiently by identifying high and low-performing teams.
- Enhance Productivity Strategies: Develop tailored improvement plans based on model insights.
- Benchmark Performance: Utilize overperforming teams as benchmarks for best practices.

In comparison, while the Logistic Regression model is simpler and faster to train, it lacks robustness for precise classification due to the dataset's complexity and non-linear relationships.

Conclusions:

This assessment developed a machine learning approach to predict and optimize productivity in the garment industry, addressing the critical challenge of aligning targeted and actual productivity levels. The Random Forest model, achieving 80% accuracy, outperformed Logistic Regression (75%), demonstrating the potential of machine learning for productivity forecasting. The analysis highlighted key factors that influence productivity: department type, particularly sewing (which accounts for 57.7% of operations), and temporal factors (such as quarterly fluctuations). The correlation between targeted productivity and actual outcomes was found to be moderately positive (0.42), underscoring the importance of accurate goal setting.

Actionable Recommendations:

1. Real-Time Monitoring: Implement real-time productivity tracking using the Random Forest model to enable proactive management.
2. Resource Optimization: Adjust workforce allocation during peak periods and focus on department-specific patterns.
3. Incentive Structures: Develop data-driven incentive programs aligned with achievable targets to boost morale and performance.
4. Data Standardization: Improve data collection processes, particularly for tracking Work in Progress (WIP), to enhance model accuracy.

By systematically applying these insights, garment manufacturers can boost operational efficiency while maintaining fair labour practices and improving overall productivity.

References

Imran, A. A., Amin, M. N., Rifat, M. R. I., & Mehreen, S. (2019). Deep neural network approach for predicting the productivity of garment employees. https://doi.org/10.1109/codit.2019.8820486

Sabuj, H. H., Nuha, N. S., Gomes, P. R., Lameesa, A., & Alam, M. A. (2022). Interpretable Garment Workers' Productivity Prediction in Bangladesh Using Machine Learning Algorithms and Explainable AI. https://doi.org/10.1109/iccit57492.2022.10054863

Bhandari, A. (2024, October 9). Feature Scaling: Engineering, Normalization, and Standardization (Updated 2024). https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Hafeez, A., & Sial, A. (2021). Comparative analysis of data visualization libraries Matplotlib and Seaborn in Python. International Journal of Advanced Trends in Computer Science and Engineering, 10(1), 2770–2781. https://doi.org/10.30534/ijatcse/2021/391012021

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.

Brownlee, J. (2020). Machine learning algorithms from scratch. Machine Learning Mastery.

Chollet, F. (2018). Deep learning with Python. Manning Publications.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

Peng, H., Long, F., & Ding, C. (2002). Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions

on Pattern Analysis and Machine Intelligence, 27(8), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. Machine Learning, 46(1–3), 389–422. https://doi.org/10.1023/A:1012487302797

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(1), 281–305. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002