Linear Regression Project

1 Introduction

1.1 Motivation

The garment sector is one of the most important industries in this current period of industrial globalization. The garment industry plays a key role in the growth of the economy, especially for developing countries. It is a highly labor-intensive sector that requires a significant amount of human resources to fulfill the global demand for garment products.

The production of a garment company is largely determined by the productivity of the employees across its different departments. Consequently, when the targeted productivity is not satisfied, the company will likely suffer a huge loss.

Aiming to address this problem by predicting a team's actual productivity, I used the Productivity Prediction of Garment Employees Data Set from the UCI Machine Learning Repository to build a simple linear regression model.

1.2 Contents

2 Data Cleaning

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [2]:
```
data = pd.read_csv('garments_worker_productivity.csv')
data.head()
```

Out[2]:

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

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   date                   1197 non-null   object 
 1   quarter                1197 non-null   object 
 2   department             1197 non-null   object 
 3   day                    1197 non-null   object 
 4   team                   1197 non-null   int64  
 5   targeted_productivity  1197 non-null   float64
 6   smv                    1197 non-null   float64
 7   wip                    691 non-null    float64
 8   over_time              1197 non-null   int64  
 9   incentive              1197 non-null   int64  
 10  idle_time              1197 non-null   float64
 11  idle_men               1197 non-null   int64  
 12  no_of_style_change     1197 non-null   int64  
 13  no_of_workers          1197 non-null   float64
 14  actual_productivity    1197 non-null   float64
dtypes: float64(6), int64(5), object(4)
memory usage: 140.4+ KB
```

In [4]:
```
# Correct the mispelling
data['department'] = data['department'].apply(lambda x: 'finishing' if x == ('finishing ' or 'finishing') else 'sewing
```

In [5]:
```
# Converting over_time to over_time_hour
data['over_time_hour'] = data['over_time'].apply(lambda x: x /60)
data.drop('over_time', axis = 1, inplace = True)
```

Since 'over_time' counts in minutes, the value is large and the variance will also be large. I don't want this order of magnitude difference to affect my subsequent analysis, so I converted minutes to hours.

In [6]:
```
# Numeric Variables
all_colums = list (data)
numeric_colums = ['targeted_productivity', 'smv', 'wip', 'over_time_hour',
                'incentive', 'idle_time', 'idle_men','no_of_workers', 'actual_productivity']
```

In [7]:
```
# Categorical Variables
categorical_colums = ['quarter', 'department','day','team','no_of_style_change']
data['quarter'] = data['quarter'].apply(lambda x: 0 if x == 'Quarter1' else
                                        (1 if x =='Quarter2' else
                                        (2 if x == 'Quarter3' else 3)))
data['department'] = data['department'].apply(lambda x: 0 if x == 'finishing' else 1)
data['day'] = data['day'].apply(lambda x: 0 if x == 'Thursday' else
                                (1 if x == 'Saturday' else 2))
data['no_of_style_change'] = data['no_of_style_change'].apply(lambda x: 0 if x == 0 else 1)
```

For 'day': I find that workers don't work on Friday, I want to consider whether the productivity will be significantly different before and after the day off.

For 'no_of_style_change': Most of values are 0, while only a few are 1 or 2, so I treat it as a categorical variables to show whether workers' working style was changed.

In [8]:
```
# Variables Information
```

· date : Date in MM-DD-YYYY
· day : Day of the Week

```
= 0 if on Thursday (the day before the rest day)
= 1 if on Saturday(the day after the rest day)
= 2 if between Sunday and Wednesday
```

· quarter : A portion of the month

```
= 0 if in Quarter1
= 1 if in Quarter2
= 2 if in Quarter3
= 3 if in Quarter4
```

· department : Associated department with the instance

```
= 0 if in finishing department
= 1 if in sewing department
```

· team_no : Associated team number with the instance
· no_of_workers : Number of workers in each team
· no_of_style_change : Number of changes in the style of a particular product

```
= 0 if did not change
= 1 if changed
```

· targeted_productivity : Targeted productivity set by the Authority for each team for each day
· smv : Standard Minute Value, it is the allocated time for a task
· wip : Work in progress. Includes the number of unfinished items for products
· over_time : Represents the amount of overtime by each team in minutes
· incentive : Represents the amount of financial incentive (in BDT) that enables or motivates a particular course of action
· idle_time : The amount of time when the production was interrupted due to several reasons
· idle_men : The number of workers who were idle due to production interruption
· actual_productivity : The actual % of productivity that was delivered by the workers. It ranges from 0-1

3 Descriptive Analysis

3.1 Inspect the data

In [9]:
```python
# Look for any missing observations
print(data.isnull().any())

# Count the number of missing obs per variable
print(data.isnull().sum())
```

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

In [10]:
```
# Replace the missing values with 0
print(data[data.wip.isna()]['department'].unique)

from numpy import nan
data['wip'].replace(np.nan, 0, inplace = True)
print(data.isnull().any())
```

```
<bound method Series.unique of 1       0
6       0
13      0
14      0
15      0
       ..
1192    1
1193    1
1194    1
1195    1
1196    1
Name: department, Length: 506, dtype: int64>
date                     False
quarter                  False
department               False
day                      False
team                     False
targeted_productivity    False
smv                      False
wip                      False
incentive                False
idle_time                False
idle_men                 False
no_of_style_change       False
no_of_workers            False
actual_productivity      False
over_time_hour           False
dtype: bool
```

The only variable in my dataset with null values is wip (work in progress), which includes the number of unfinished items for products. I imputed the missing values with 0 because I determined that all of the NAs were under the finishing department; and in general, the finishing department does not have any WIP.

In [11]:
```
round(data.describe(),2)
```

Out[11]:

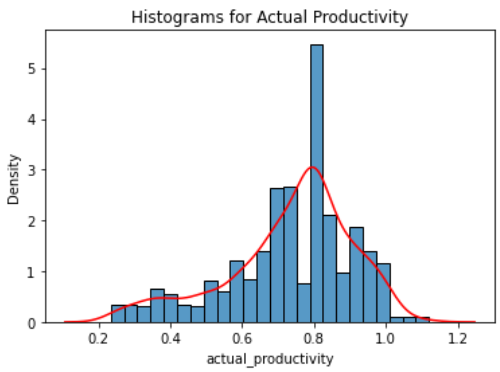
	quarter	department	day	team	targeted_productivity	smv	wip	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actua
count	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	1197.00	
mean	1.36	0.79	1.51	6.43	0.73	15.06	687.23	38.21	0.73	0.37	0.12	34.61	
std	1.15	0.41	0.76	3.46	0.10	10.94	1514.58	160.18	12.71	3.27	0.33	22.20	
min	0.00	0.00	0.00	1.00	0.07	2.90	0.00	0.00	0.00	0.00	0.00	2.00	
25%	0.00	1.00	1.00	3.00	0.70	3.94	0.00	0.00	0.00	0.00	0.00	9.00	
50%	1.00	1.00	2.00	6.00	0.75	15.26	586.00	0.00	0.00	0.00	0.00	34.00	
75%	2.00	1.00	2.00	9.00	0.80	24.26	1083.00	50.00	0.00	0.00	0.00	57.00	
max	3.00	1.00	2.00	12.00	0.80	54.56	23122.00	3600.00	300.00	45.00	1.00	89.00	

3.2 Distribution: Histograms with Density Plot

(1) Numeric Variables

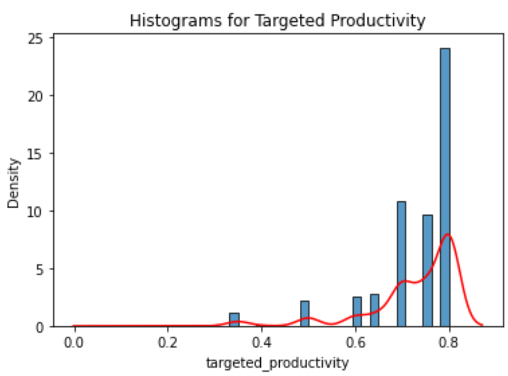
In [12]:
```python
# actual_productivity
plt.title('Histograms for Actual Productivity')
sns.histplot(data.actual_productivity, stat = 'density')
sns.kdeplot(data.actual_productivity, color = 'red')
```

Out[12]: <AxesSubplot:title={'center':'Histograms for Actual Productivity'}, xlabel='actual_productivity', ylabel='Density'>



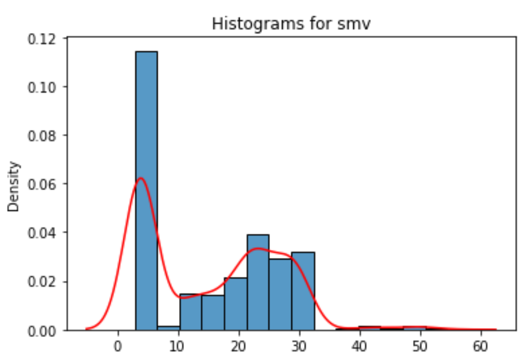
In [13]:
```python
# targeted_productivity
plt.title('Histograms for Targeted Productivity')
sns.histplot(data.targeted_productivity, stat = 'density')
sns.kdeplot(data.targeted_productivity, color = 'red')
```

Out[13]: <AxesSubplot:title={'center':'Histograms for Targeted Productivity'}, xlabel='targeted_productivity', ylabel='Density'>



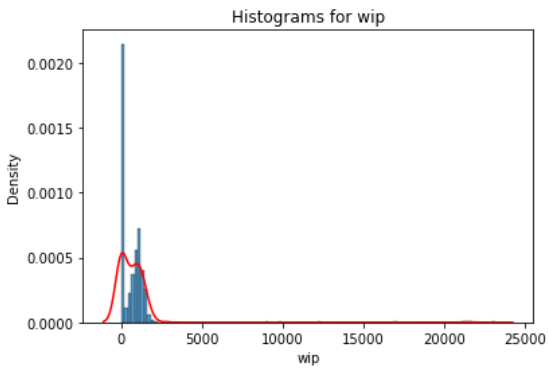
In [14]:
```python
# smv
plt.title('Histograms for smv')
sns.histplot(data.smv, stat = 'density')
sns.kdeplot(data.smv, color = 'red')
```

Out[14]: <AxesSubplot:title={'center':'Histograms for smv'}, xlabel='smv', ylabel='Density'>



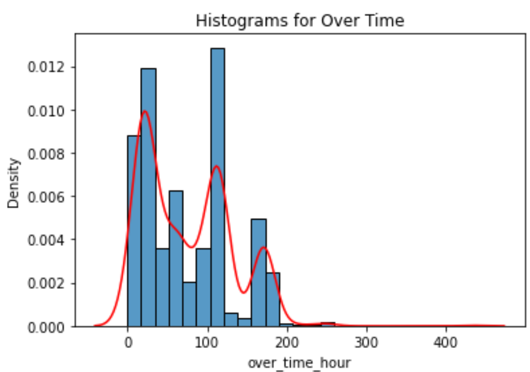
In [15]:
```python
# wip
plt.title('Histograms for wip')
sns.histplot(data.wip, stat = 'density')
sns.kdeplot(data.wip, color = 'red')
```

Out[15]:
```
<AxesSubplot:title={'center':'Histograms for wip'}, xlabel='wip', ylabel='Density'>
```



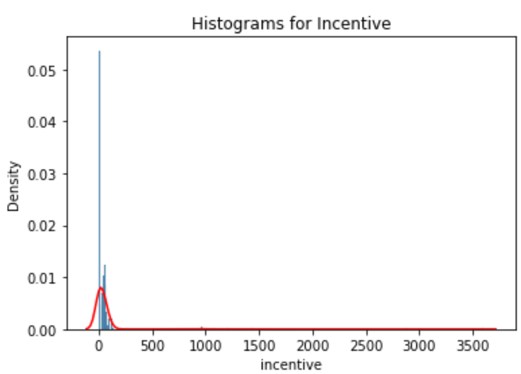
In [16]:
```python
# over_time_hour
plt.title('Histograms for Over Time')
sns.histplot(data.over_time_hour, stat = 'density')
sns.kdeplot(data.over_time_hour, color = 'red')
```

Out[16]:
```
<AxesSubplot:title={'center':'Histograms for Over Time'}, xlabel='over_time_hour', ylabel='Density'>
```



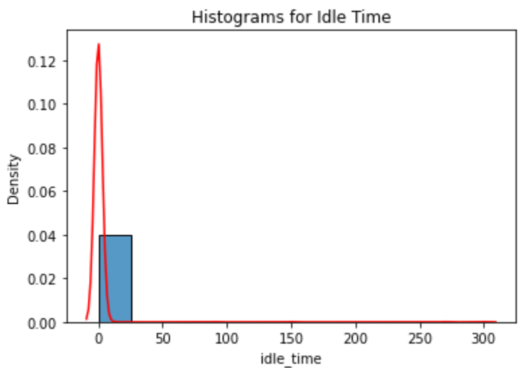
In [17]:
```python
# incentive
plt.title('Histograms for Incentive')
sns.histplot(data.incentive, stat = 'density')
sns.kdeplot(data.incentive, color = 'red')
```

Out[17]:
```
<AxesSubplot:title={'center':'Histograms for Incentive'}, xlabel='incentive', ylabel='Density'>
```



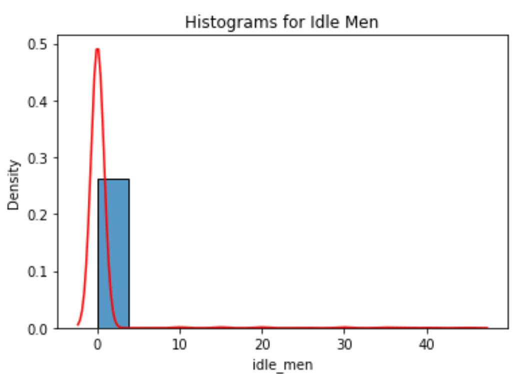
```
In [18]: # idle_time
plt.title('Histograms for Idle Time')
sns.histplot(data.idle_time, stat = 'density')
sns.kdeplot(data.idle_time, color = 'red')
```

Out[18]: <AxesSubplot:title={'center':'Histograms for Idle Time'}, xlabel='idle_time', ylabel='Density'>



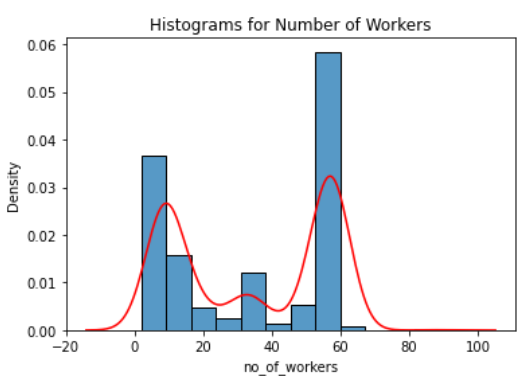

```
In [19]: # idle_men
plt.title('Histograms for Idle Men')
sns.histplot(data.idle_men, stat = 'density')
sns.kdeplot(data.idle_men, color = 'red')
```

Out[19]: <AxesSubplot:title={'center':'Histograms for Idle Men'}, xlabel='idle_men', ylabel='Density'>




```
In [20]: # no_of_workers
plt.title('Histograms for Number of Workers')
sns.histplot(data.no_of_workers, stat = 'density')
sns.kdeplot(data.no_of_workers, color = 'red')
```

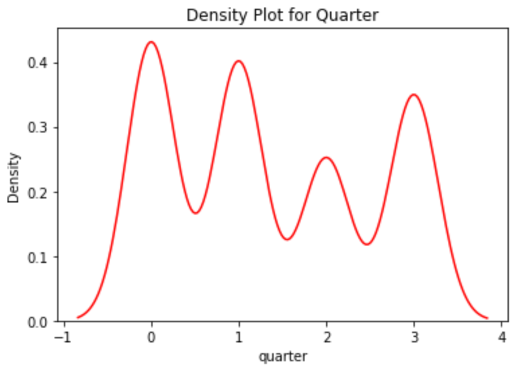
Out[20]: <AxesSubplot:title={'center':'Histograms for Number of Workers'}, xlabel='no_of_workers', ylabel='Density'>



(2) Categorical Variables

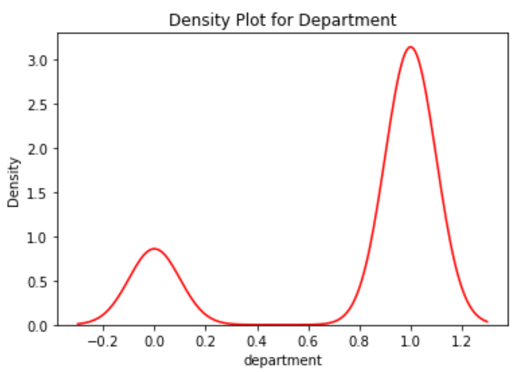
In [21]:
```python
# quarter
plt.title('Density Plot for Quarter')
sns.kdeplot(data.quarter, color = 'red')
```

Out[21]: <AxesSubplot:title={'center':'Density Plot for Quarter'}, xlabel='quarter', ylabel='Density'>



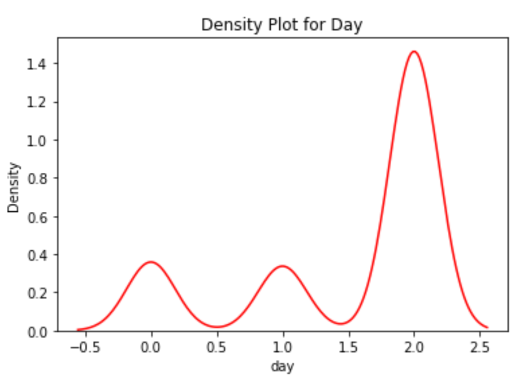
In [22]:
```python
# department
plt.title('Density Plot for Department')
sns.kdeplot(data.department, color = 'red')
```

Out[22]: <AxesSubplot:title={'center':'Density Plot for Department'}, xlabel='department', ylabel='Density'>



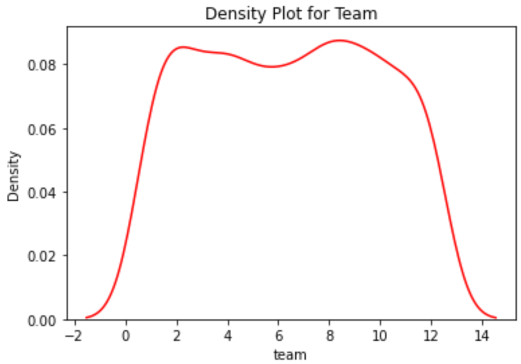
In [23]:
```python
# day
plt.title('Density Plot for Day')
sns.kdeplot(data.day, color = 'red')
```

Out[23]: <AxesSubplot:title={'center':'Density Plot for Day'}, xlabel='day', ylabel='Density'>



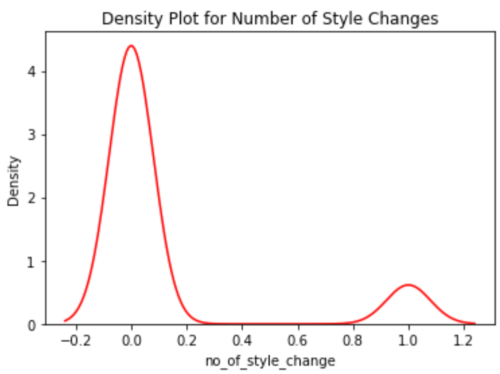
In [24]:
```
# team
plt.title('Density Plot for Team')
sns.kdeplot(data.team, color = 'red')
```

Out[24]:
```
<AxesSubplot:title={'center':'Density Plot for Team'}, xlabel='team', ylabel='Density'>
```



In [25]:
```
# no_of_style_change
plt.title('Density Plot for Number of Style Changes')
sns.kdeplot(data.no_of_style_change, color = 'red')
```

Out[25]:
```
<AxesSubplot:title={'center':'Density Plot for Number of Style Changes'}, xlabel='no_of_style_change', ylabel='Densi
ty'>
```

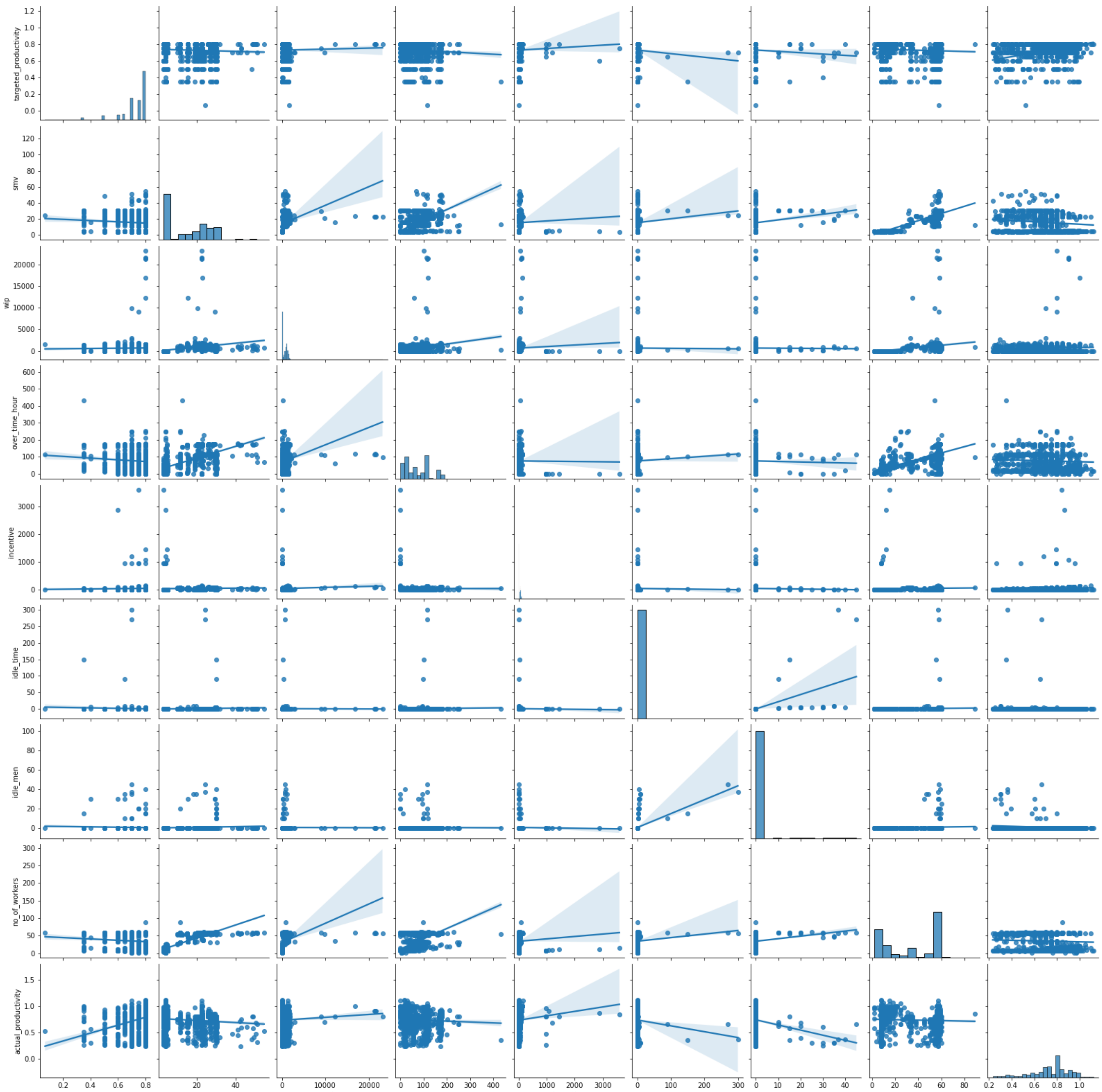


3.3 Linear Relationship

3.3.1 Scatterplot Matrix

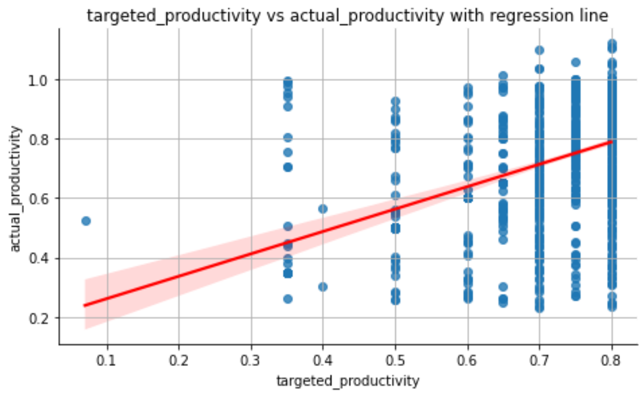
In [26]:
```
sns.pairplot(data[numeric_colums], kind = 'reg')
```

Out[26]:
```
<seaborn.axisgrid.PairGrid at 0x7f8db07aebb0>
```

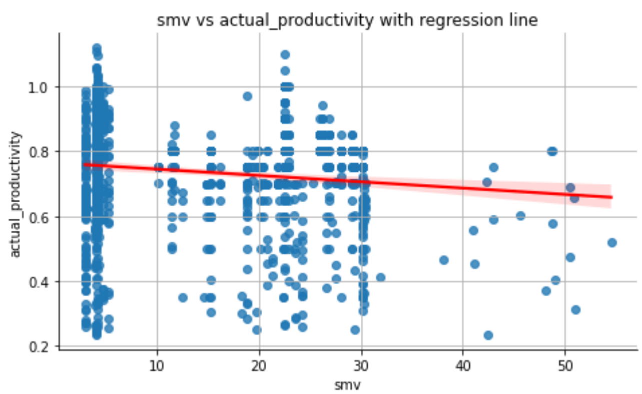


3.3.2 Scatterplots between Predictor Variables and Predicted Variable

In [27]:
```
# targeted_productivity vs actual_productivity

sns.lmplot(data = data, x='targeted_productivity', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('targeted_productivity vs actual_productivity with regression line')
plt.grid()
plt.show()
```

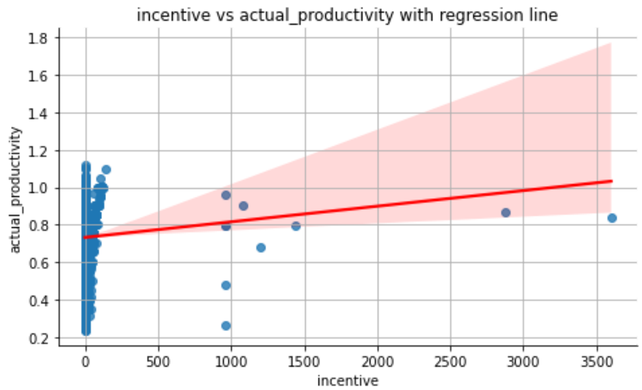


In [28]:
```
# smv vs actual_productivity
sns.lmplot(data = data, x='smv', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('smv vs actual_productivity with regression line')
plt.grid()
plt.show()
```

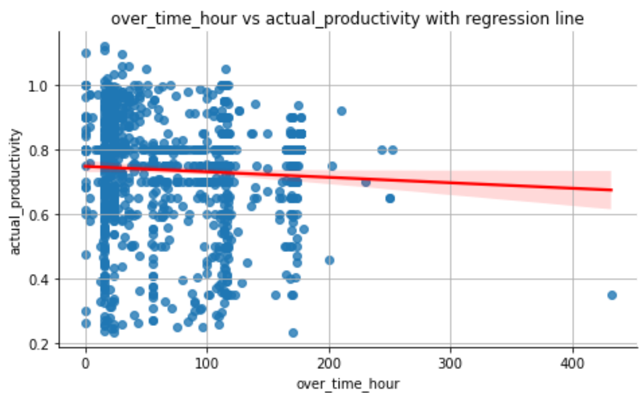


In [29]:
```
# wip vs actual_productivity
sns.lmplot(data = data, x='wip', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('wip vs actual_productivity with regression line')
plt.grid()
plt.show()
```

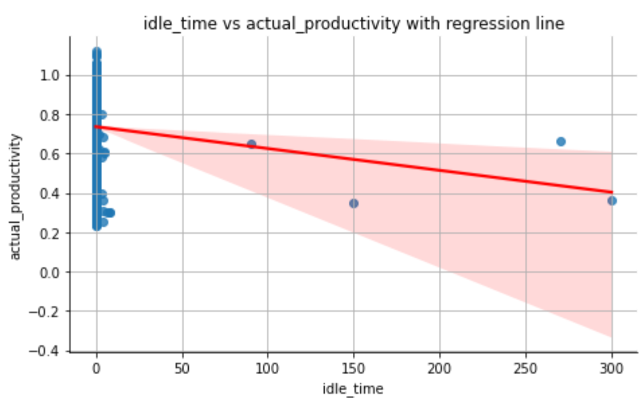
In [30]:
```
# incentive vs actual_productivity
sns.lmplot(data = data, x='incentive', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('incentive vs actual_productivity with regression line')
plt.grid()
plt.show()
```



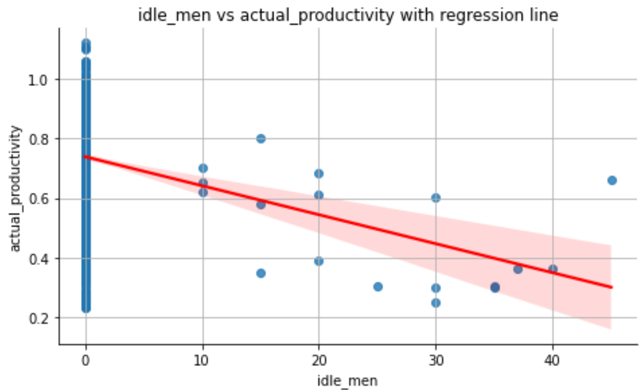
In [31]:
```
# over_time_hour vs actual_productivity
sns.lmplot(data = data, x='over_time_hour', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('over_time_hour vs actual_productivity with regression line')
plt.grid()
plt.show()
```



In [32]:
```
# idle_time vs actual_productivity
sns.lmplot(data = data, x='idle_time', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('idle_time vs actual_productivity with regression line')
plt.grid()
plt.show()
```



In [33]:
```
# idle_men vs actual_productivity
sns.lmplot(data = data, x='idle_men', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('idle_men vs actual_productivity with regression line')
plt.grid()
plt.show()
```



In [34]:
```
# no_of_workers vs actual_productivity
sns.lmplot(data = data, x='no_of_workers', y='actual_productivity',
line_kws = {'color':'red'},height = 4, aspect = 1.7, ci = 95)
plt.title('no_of_workers vs actual_productivity with regression line')
plt.grid()
plt.show()
```

Based on scatterplots between my numeric predictor variables (targeted_productivity, smv, wip, over_time_hour, incentive, idle_time, idle_men, no_of_workers) and predicted (actual productivity), I did not see a strong linear relationship between predictor variables and actual productivity.

Consequently, I performed linear transformation tests on all predictor variables in section 3.3.3. For variables with values strictly greater than zero, I performed Box-Cox tests to obtain a lambda value for a power transformation. For variables with values equal to or less than zero, I performed Yeo-Johnson tests to obtain a lambda value for a power transformation.

The results and plots of these linear transformations can be found in section 3.3.3. I compared histograms and QQ-Plots of the original data with those of the transformed data to check for increased normality.

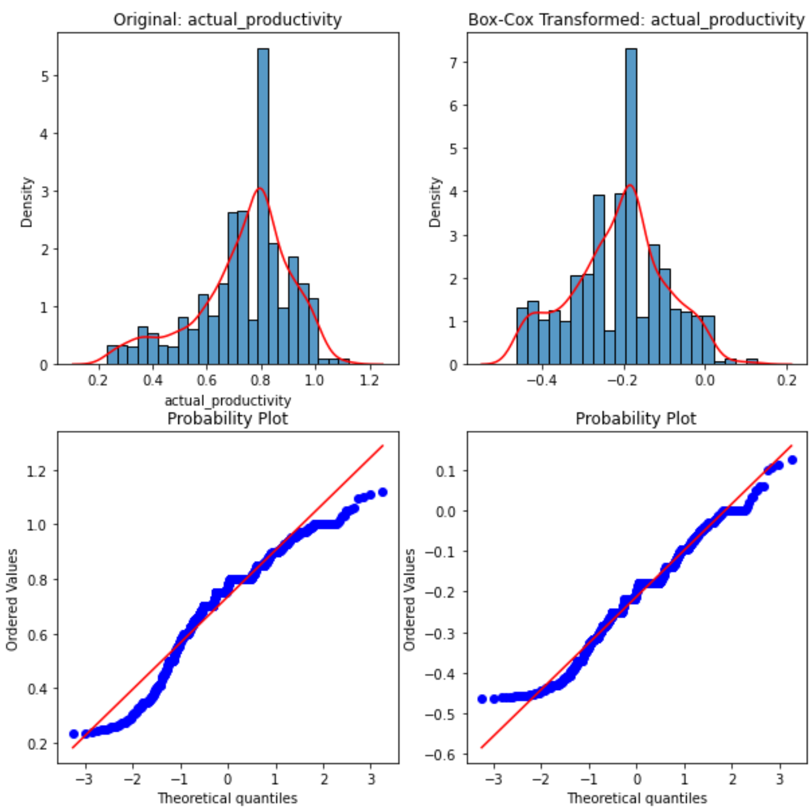
Without using the transformed data, I run the risk of an unstable regression model and inaccurate coefficient estimates.

3.3.3 Data Transformation

In [35]:
```
# actual_productivity
bc_actual_productivity, lambda_actual_productivity = stats.boxcox(data['actual_productivity'])
print('actual_productivity:' ,lambda_actual_productivity)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['actual_productivity'], stat = 'density')
sns.kdeplot(data.actual_productivity, color = 'red')
plt.title('Original: actual_productivity')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_actual_productivity, stat = 'density')
sns.kdeplot(bc_actual_productivity, color = 'red')
plt.title('Box-Cox Transformed: actual_productivity')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.actual_productivity, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_actual_productivity, dist = "norm", plot = plt)
```

```
actual_productivity: 2.0484897908782704
```

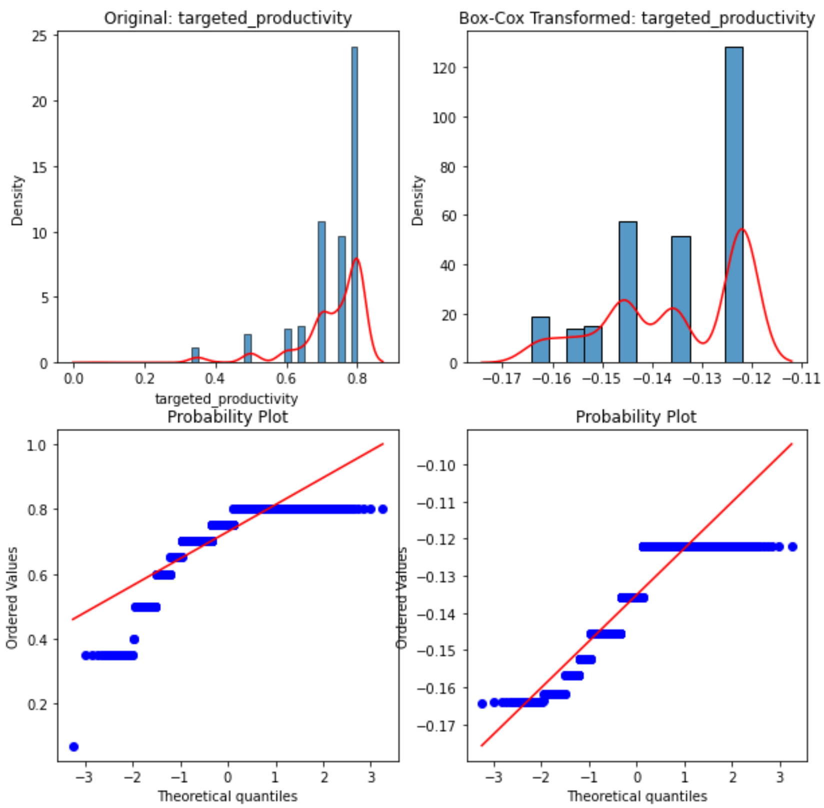
Out[35]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([-0.46331654, -0.46285921, -0.46236293, ...,  0.10578563,
          0.1142644 ,  0.12805617])),
 (0.11494652726109751, -0.21263868729275726, 0.9915716184996908))
```



In [36]:
```
# targeted_productivity
bc_targeted_productivity, lambda_targeted_productivity = stats.boxcox(data['targeted_productivity'])
print('lambda_targeted_productivity:' ,lambda_targeted_productivity)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['targeted_productivity'], stat = 'density')
sns.kdeplot(data.targeted_productivity, color = 'red')
plt.title('Original: targeted_productivity')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_targeted_productivity, stat = 'density')
sns.kdeplot(bc_targeted_productivity, color = 'red')
plt.title('Box-Cox Transformed: targeted_productivity')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.targeted_productivity, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_targeted_productivity, dist = "norm", plot = plt)
```

```
lambda_targeted_productivity: 6.090518949445242
```

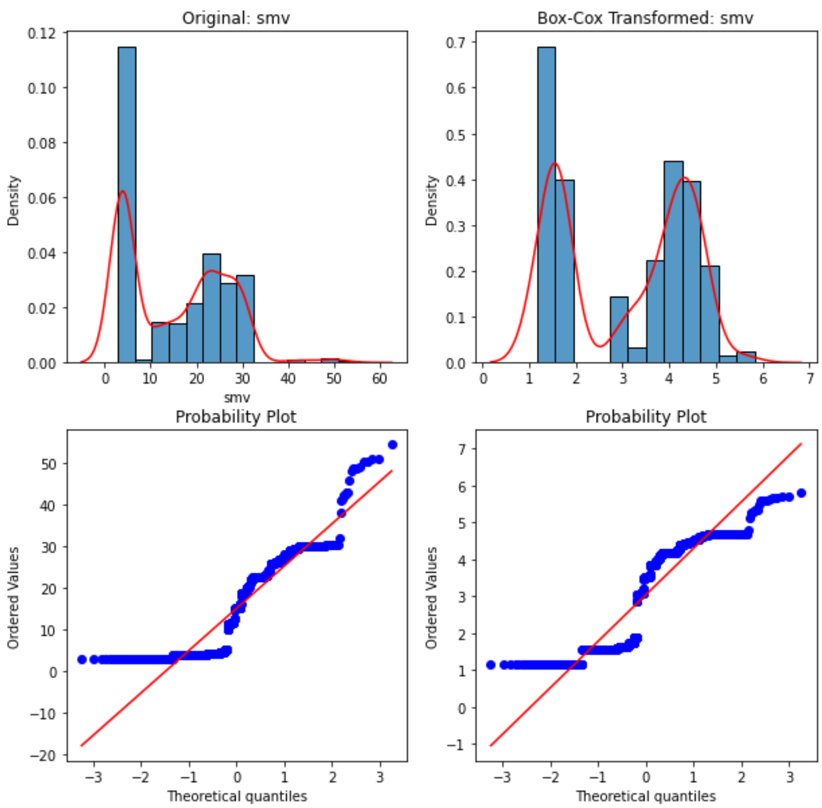
Out[36]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([-0.16418961, -0.16391516, -0.16391516, ..., -0.12200896,
         -0.12200896, -0.12200896])),
 (0.012483976989740303, -0.1351609079422609, 0.910188998778435))
```



In [37]:
```python
# smv
bc_smv, lambda_smv = stats.boxcox(data['smv'])
print('lambda_smv:' ,lambda_smv)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['smv'], stat = 'density')
sns.kdeplot(data.smv, color = 'red')
plt.title('Original: smv')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_smv, stat = 'density')
sns.kdeplot(bc_smv, color = 'red')
plt.title('Box-Cox Transformed: smv')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.smv, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_smv, dist = "norm", plot = plt)
```

```
lambda_smv: 0.1776369586616808
```

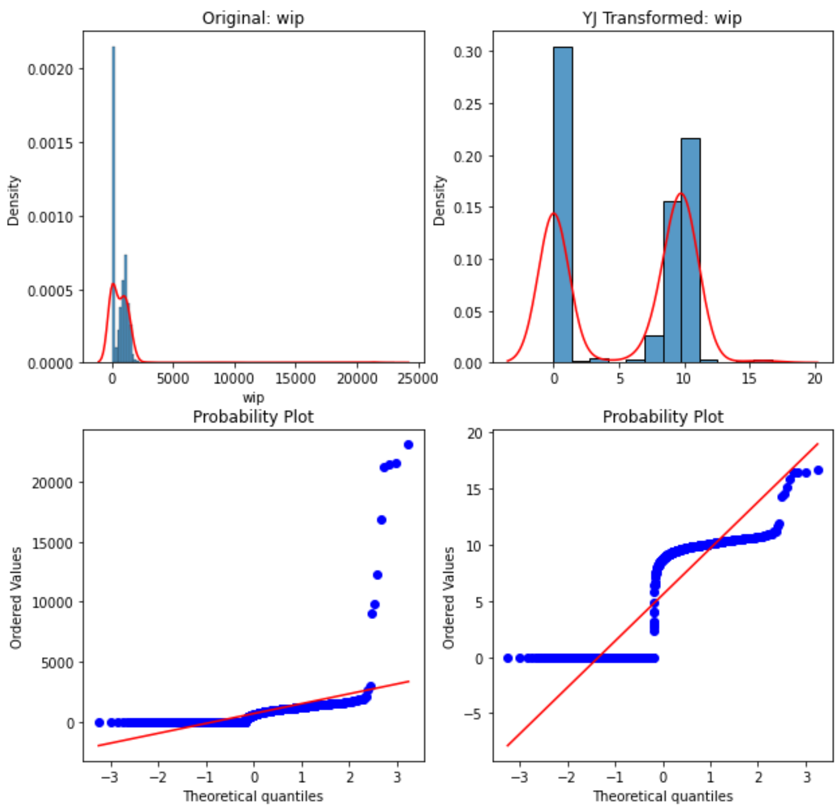
Out[37]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([1.17205563, 1.17205563, 1.17205563, ..., 5.68484032, 5.68996913,
         5.82566369])),
 (1.256259358034622, 3.0377415716366216, 0.9168287359794698))
```



In [38]:
```
# wip
yj_wip, lambda_wip = stats.yeojohnson(data['wip'])
print('lambda_wip:' ,lambda_wip)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['wip'],stat = 'density')
sns.kdeplot(data.wip, color = 'red')
plt.title('Original: wip')
ax1 = fig.add_subplot(2,2,2)
sns.histplot(yj_wip,stat = 'density')
sns.kdeplot(yj_wip, color = 'red')
plt.title('YJ Transformed: wip')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.wip, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(yj_wip, dist = "norm", plot = plt)
```

```
lambda_wip: 0.09350857608068355
```

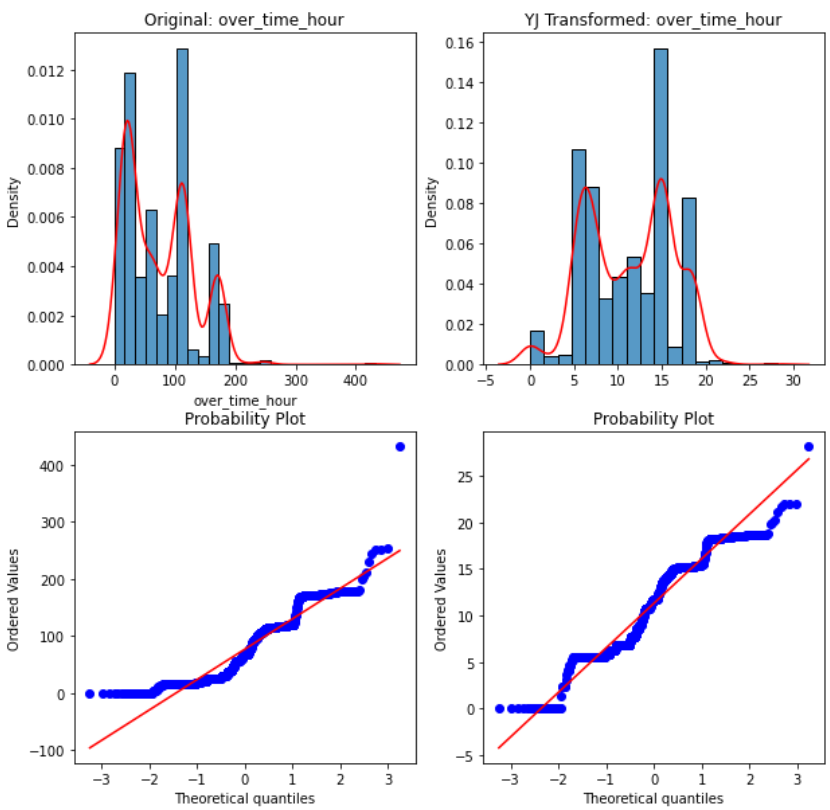
Out[38]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([ 0.        ,  0.        ,  0.        , ..., 16.47348955,
         16.49184157, 16.67259973])),
 (4.133913646292112, 5.541317179932486, 0.85200427096373))
```



In [39]:
```
# over_time_hour
yj_over_time_hour, lambda_over_time_hour = stats.yeojohnson(data['over_time_hour'])
print('lambda_over_time_hour:' ,lambda_over_time_hour)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['over_time_hour'],stat = 'density')
sns.kdeplot(data.over_time_hour, color = 'red')
plt.title('Original: over_time_hour')
ax1 = fig.add_subplot(2,2,2)
sns.histplot(yj_over_time_hour,stat = 'density')
sns.kdeplot(yj_over_time_hour, color = 'red')
plt.title('YJ Transformed: over_time_hour')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.over_time_hour, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(yj_over_time_hour, dist = "norm", plot = plt)
```

```
lambda_over_time_hour: 0.42066372722289586
```

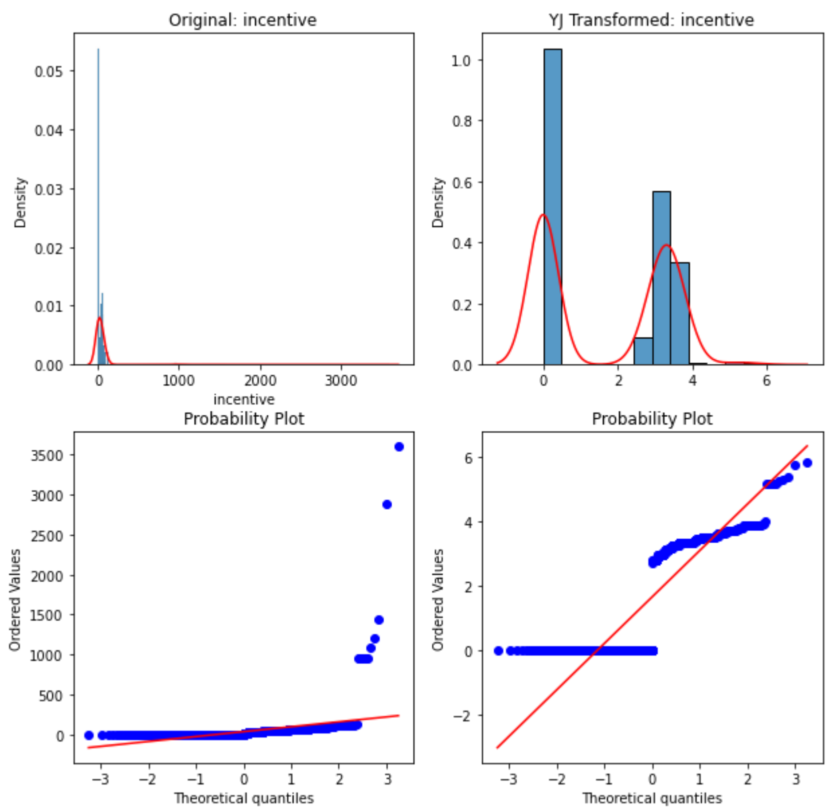
Out[39]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([ 0.        ,  0.        ,  0.        , ..., 21.91801257,
         21.99926056, 28.18184802])),
 (4.775832556741404, 11.27973113137297, 0.9718742636564588))
```



In [40]:
```
# incentive
yj_incentive, lambda_incentive = stats.yeojohnson(data['incentive'])
print('lambda_incentive:' ,lambda_incentive)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['incentive'],stat = 'density')
sns.kdeplot(data.incentive, color = 'red')
plt.title('Original: incentive')
ax1 = fig.add_subplot(2,2,2)
sns.histplot(yj_incentive,stat = 'density')
sns.kdeplot(yj_incentive, color = 'red')
plt.title('YJ Transformed: incentive')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.incentive, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(yj_incentive, dist = "norm", plot = plt)
```

```
lambda_incentive: -0.08680103833754024
```

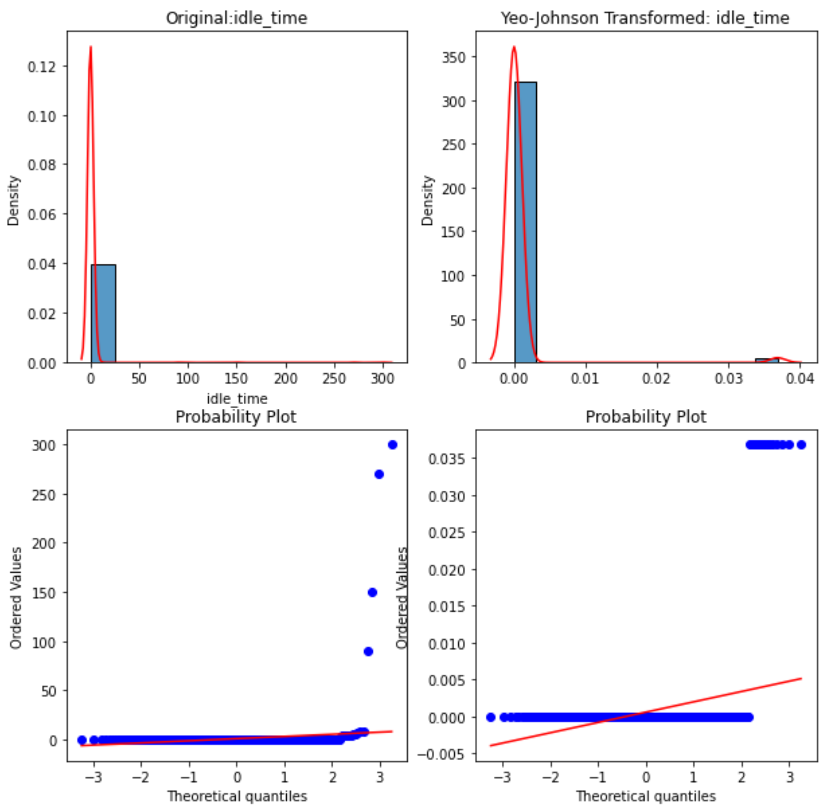
Out[40]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([-0.        , -0.        , -0.        , ...,  5.39286746,
          5.75050227,  5.8611543 ])),
 (1.4428312474502782, 1.6564769768545535, 0.8501472350714572))
```



In [41]:
```python
# idle_time
yj_idle_time, lambda_idle_time = stats.yeojohnson(data['idle_time'])
print('lambda_idle_time:' ,lambda_idle_time)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['idle_time'],stat = 'density')
sns.kdeplot(data.idle_time, color = 'red')
plt.title('Original:idle_time')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(yj_idle_time,stat = 'density')
sns.kdeplot(yj_idle_time, color = 'red')
plt.title('Yeo-Johnson Transformed: idle_time')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.idle_time, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(yj_idle_time, dist = "norm", plot = plt)
```

```
lambda_idle_time: -27.101026651966954
```

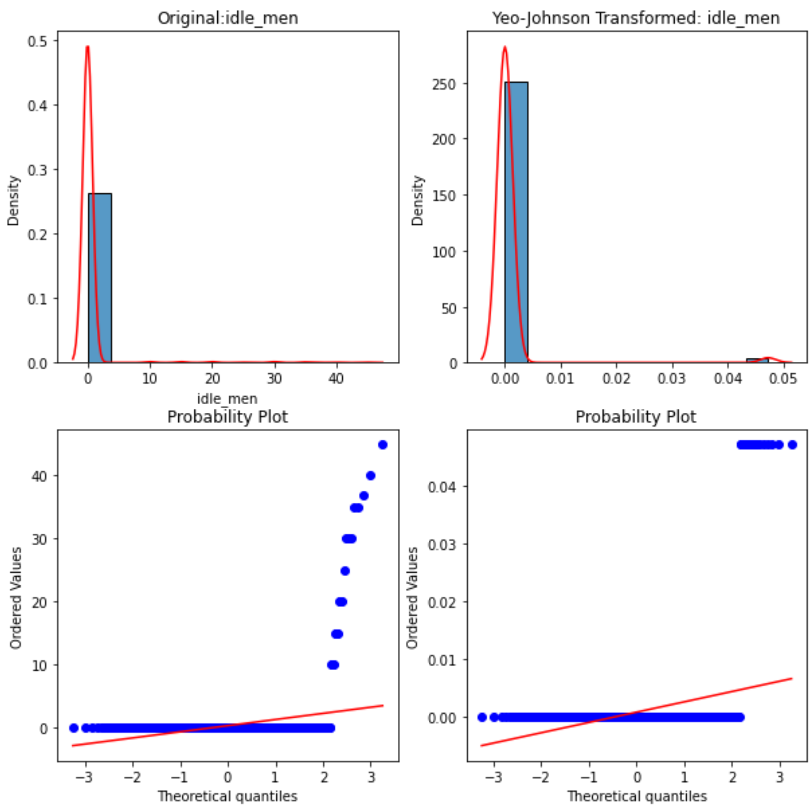
Out[41]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([-0.        , -0.        , -0.        , ...,  0.03689897,
          0.03689897,  0.03689897])),
 (0.0013962705612846989, 0.0005548717463023053, 0.3101876064149331))
```



In [42]:
```python
# idle_men
yj_idle_men, lambda_idle_men = stats.yeojohnson(data['idle_men'])
print('lambda_idle_men:' ,lambda_idle_men)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['idle_men'],stat = 'density')
sns.kdeplot(data.idle_men, color = 'red')
plt.title('Original:idle_men')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(yj_idle_men,stat = 'density')
sns.kdeplot(yj_idle_men, color = 'red')
plt.title('Yeo-Johnson Transformed: idle_men')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.idle_men, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(yj_idle_men, dist = "norm", plot = plt)
```

```
lambda_idle_men: -21.166391561132784
```

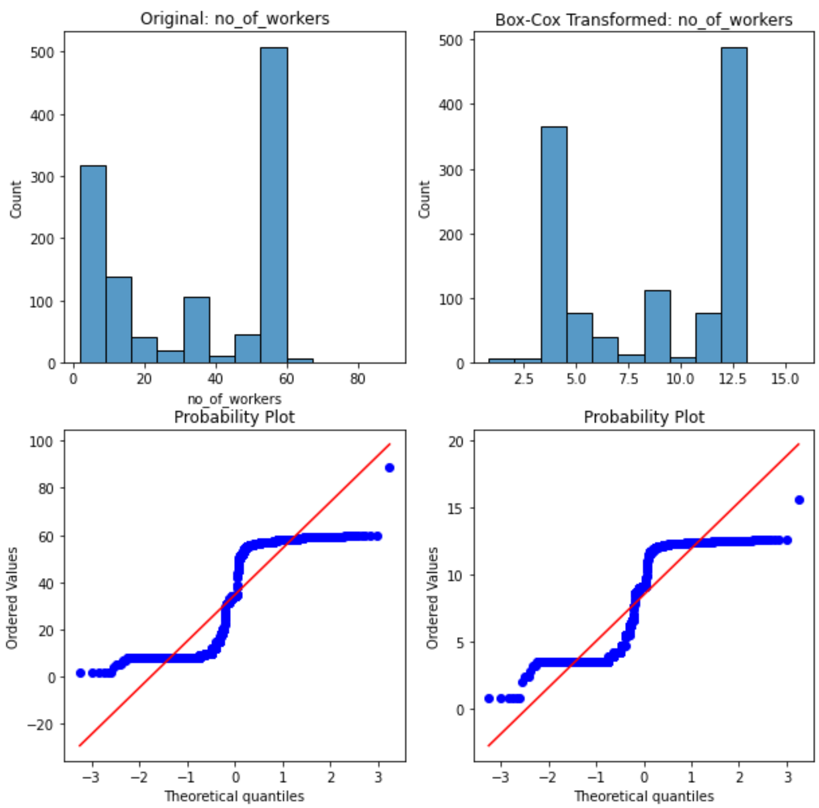
Out[42]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([-0.        , -0.        , -0.        , ...,  0.04724471,
          0.04724471,  0.04724471])),
 (0.0017877570480279244, 0.0007104467448563837, 0.31018760641493254))
```



In [43]:
```
# no_of_workers
bc_no_of_workers, lambda_no_of_workers = stats.boxcox(data['no_of_workers'])
print('lambda_no_of_workers:' ,lambda_no_of_workers)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['no_of_workers'])
plt.title('Original: no_of_workers')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_no_of_workers)
plt.title('Box-Cox Transformed: no_of_workers')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.no_of_workers, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_no_of_workers, dist = "norm", plot = plt)
```

```
lambda_no_of_workers: 0.4743260491827718
```

Out[43]:
```
((array([-3.249076  , -2.98775606, -2.84217619, ...,  2.84217619,
          2.98775606,  3.249076  ]),
  array([ 0.82067826,  0.82067826,  0.82067826, ..., 12.59273215,
         12.59273215, 15.61606155])),
 (3.4477448088742086, 8.460680099303255, 0.8866492124258766))
```

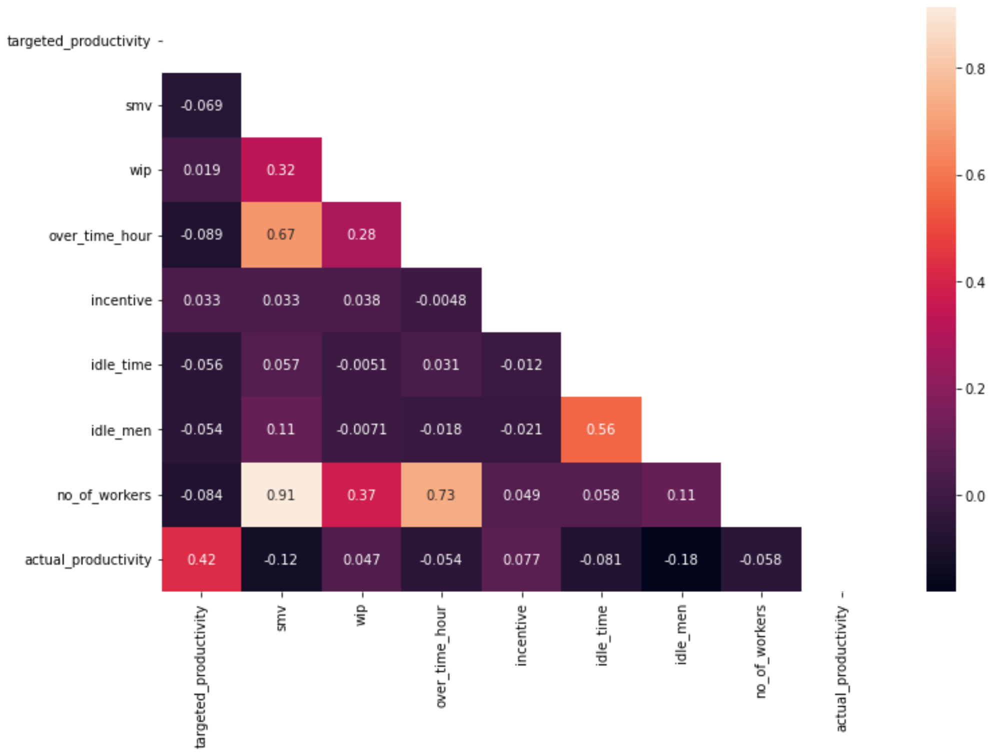


3.4 Correlation Plot

3.4.1 Heatmap

In [44]:
```
plt.figure(figsize = (12,8))
corr = data[numeric_colums].corr()
mask = np.zeros(corr.shape, dtype = bool)
mask[np.triu_indices(len(mask))] = True
sns.heatmap(corr, annot = True, mask = mask)
```

Out[44]: <AxesSubplot:>



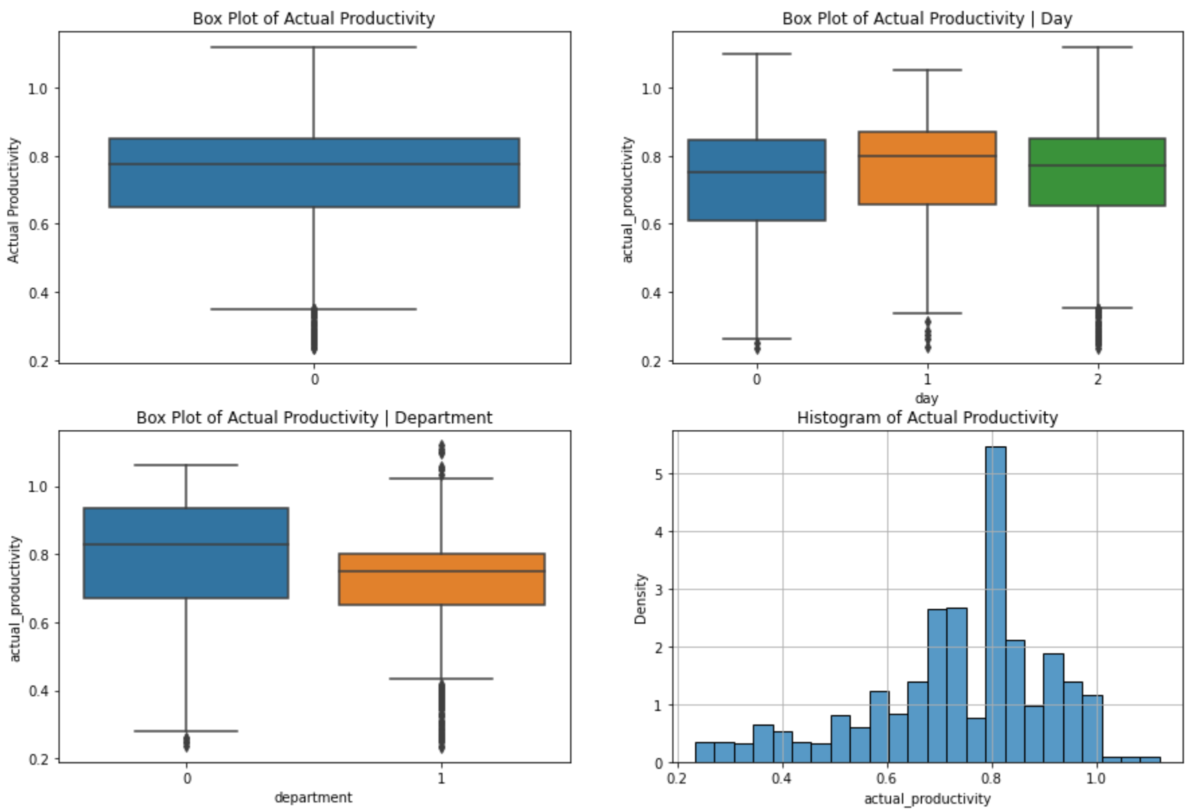
I can see from the heatmap that some variables have high correlations. Like:
· no_of_workers vs smv (0.91)
· no_of_worker vs over_time_hour (0.73)
· over_time_hour vs smv (0.67)

In general, I will need to do the collinearity test, but since I focuse on simple regression in this project, I will leave them along for the time being.

3.5 Outliers and Unusual Features

(1) Numeric Variables

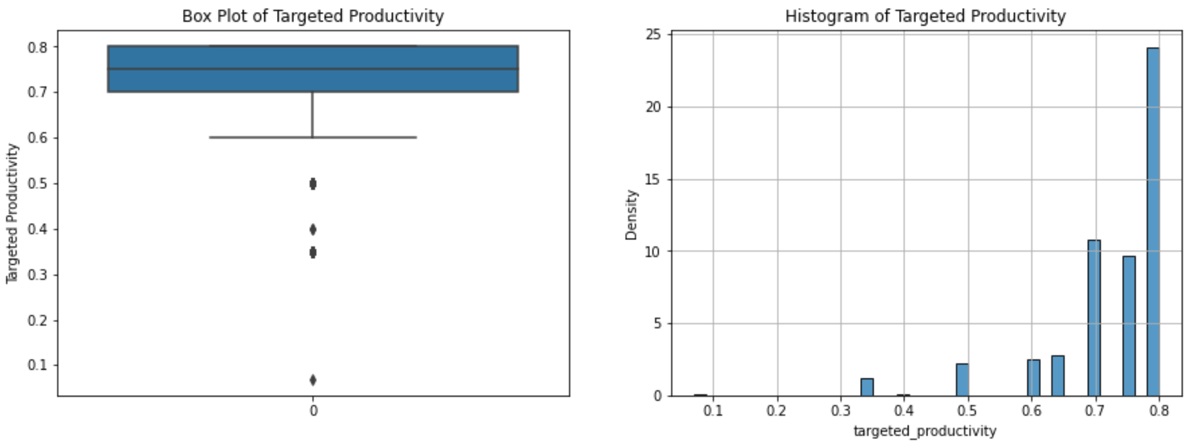
In [45]:
```
# actual productivity
fig = plt.figure(figsize = (15,10))
ax1 = fig.add_subplot(2,2,1)
plt.title("Box Plot of Actual Productivity")
sns.boxplot(data = data.actual_productivity)
plt.ylabel("Actual Productivity")
ax2 = fig.add_subplot(2,2,2)
plt.title("Box Plot of Actual Productivity | Day")
sns.boxplot(x='day',y='actual_productivity', data = data)
plt.ylabel('actual_productivity')
ax3 = fig.add_subplot(2,2,3)
plt.title("Box Plot of Actual Productivity | Department")
sns.boxplot(x='department',y='actual_productivity', data = data)
plt.ylabel('actual_productivity')
ax2 = fig.add_subplot(2,2,4)
plt.title("Histogram of Actual Productivity")
sns.histplot(data.actual_productivity, stat = "density")
plt.grid()
```



From the box plot and histogram, I observed that the distribution of actual productivity seems to be left-skewed.

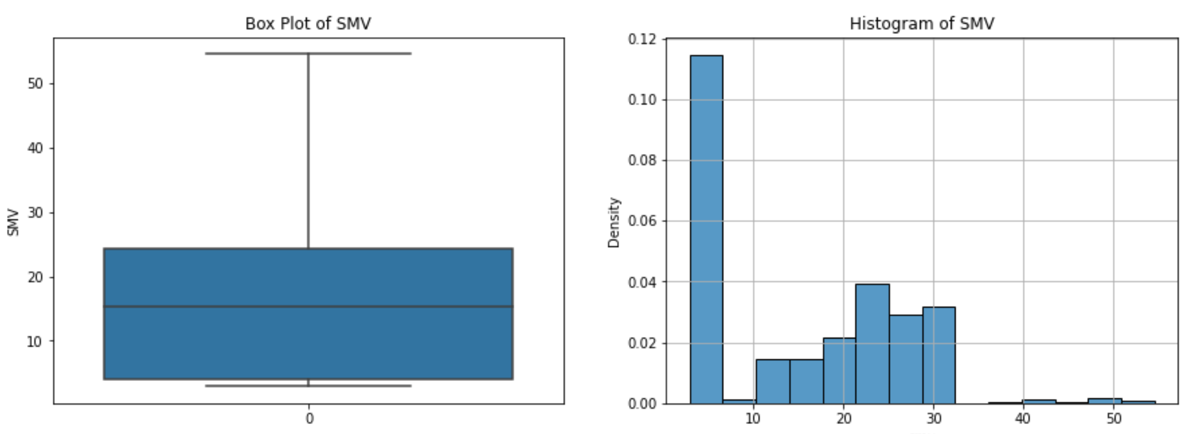
I observed that the actual productivity of the workers tends to be higher on Saturday, the first day back after their day off (Friday), and the productivity in finishing department tends to be higher.

In [46]:
```
# targeted productivity
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Targeted Productivity")
sns.boxplot(data = data.targeted_productivity)
plt.ylabel("Targeted Productivity")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Targeted Productivity")
sns.histplot(data.targeted_productivity, stat = "density")
plt.grid()
```



From the box plot and histogram, I observed that the distribution of targeted productivity seems to be left-skewed.

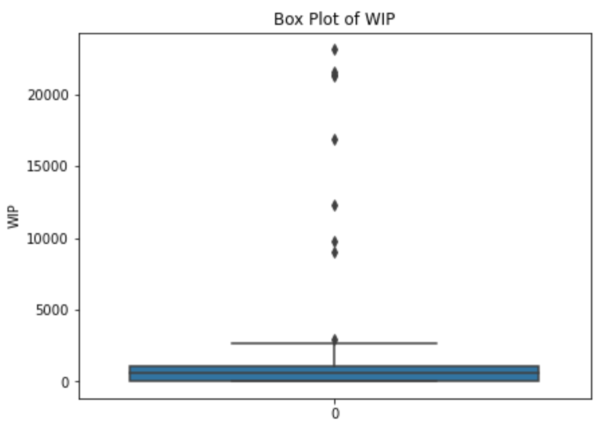
In [47]:
```
# smv
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of SMV")
sns.boxplot(data = data.smv)
plt.ylabel("SMV")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of SMV")
sns.histplot(data.smv, stat = "density")
plt.grid()
```

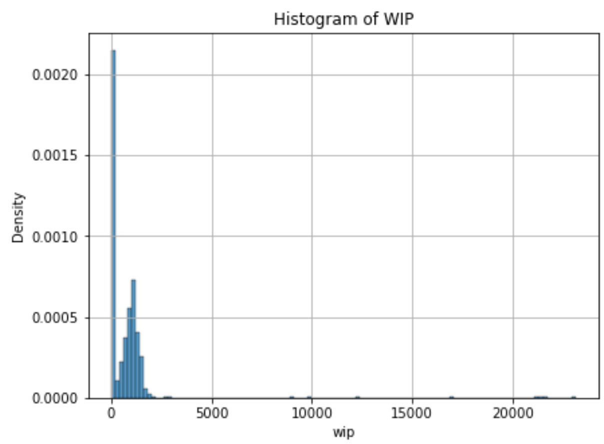


From the box plot and histogram, I observed that the distribution of SMV seems to be right-skewed.

In [48]:
```
# wip
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of WIP")
sns.boxplot(data = data.wip)
plt.ylabel("WIP")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of WIP")
sns.histplot(data.wip, stat = "density")
plt.grid()
print(data.wip.describe())
```

```
count     1197.000000
mean       687.228070
std       1514.582341
min          0.000000
25%          0.000000
50%        586.000000
75%       1083.000000
max      23122.000000
Name: wip, dtype: float64
```





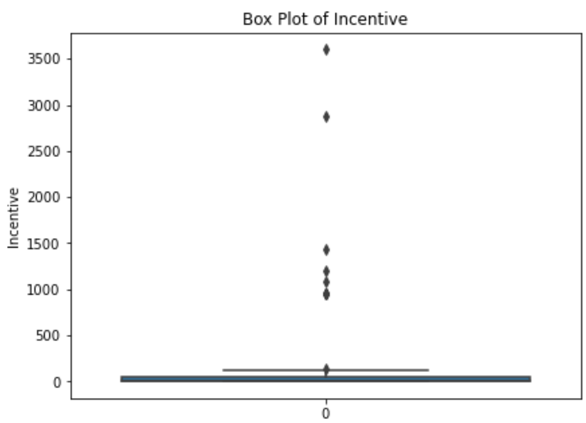
In [49]:
```
data.wip.value_counts()
```

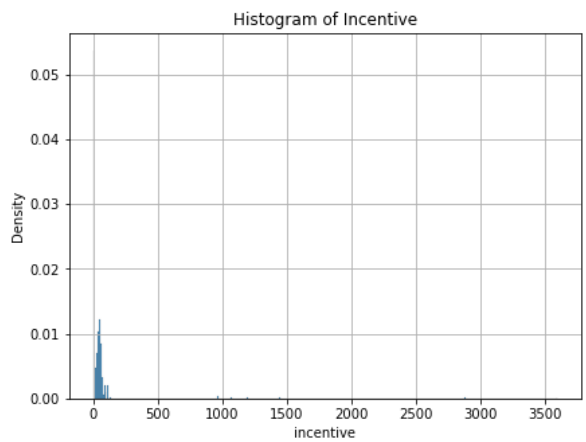
Out[49]:
```
0.0       506
1039.0      5
1282.0      4
1422.0      3
1216.0      3
         ... 
1635.0      1
1519.0      1
1337.0      1
1118.0      1
914.0       1
Name: wip, Length: 549, dtype: int64
```

From the box plot and histogram, I observed that the distribution of wip seems to be right-skewed. In particular, I note an unusually high outlier for wip of 23,122 unfinished items. An extreme value such as this may significantly affect the mean and standard deviation, as well as subsequent analysis and results.

In [50]:
```
# incentive
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Incentive")
sns.boxplot(data = data.incentive)
plt.ylabel("Incentive")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Incentive")
sns.histplot(data.incentive, stat = "density")
plt.grid()
print(data.incentive.describe())
```

```
count    1197.000000
mean       38.210526
std       160.182643
min         0.000000
25%         0.000000
50%         0.000000
75%        50.000000
max      3600.000000
Name: incentive, dtype: float64
```

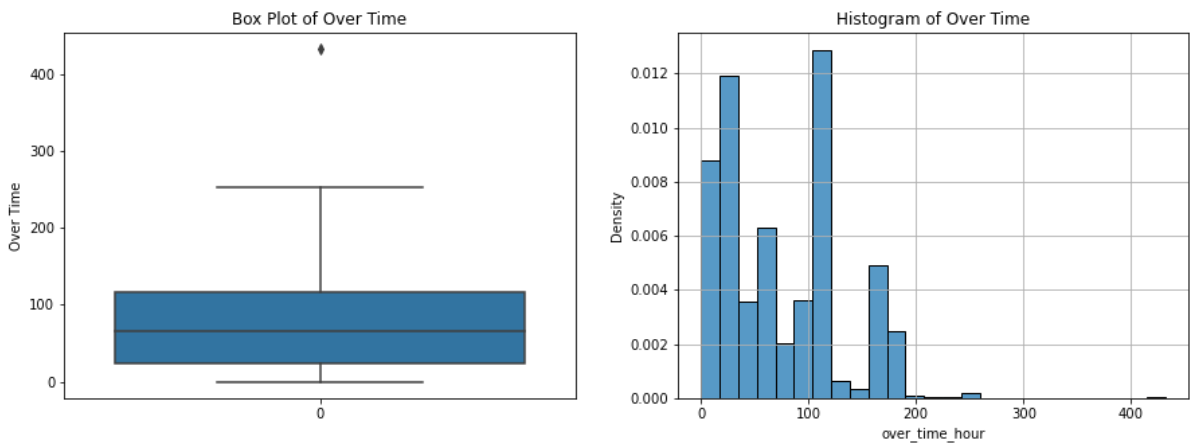




From the box plot and histogram, I observed several high outliers amongst the data points for incentive (measured in BDT, Bangladeshi Taka). These extreme values may significantly affect my analysis and results, such as the mean and standard deviation.

In [51]:
```
# over time
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Over Time")
sns.boxplot(data = data.over_time_hour)
plt.ylabel("Over Time")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Over Time")
sns.histplot(data.over_time_hour, stat = "density")
plt.grid()
print(data.over_time_hour.describe())
```

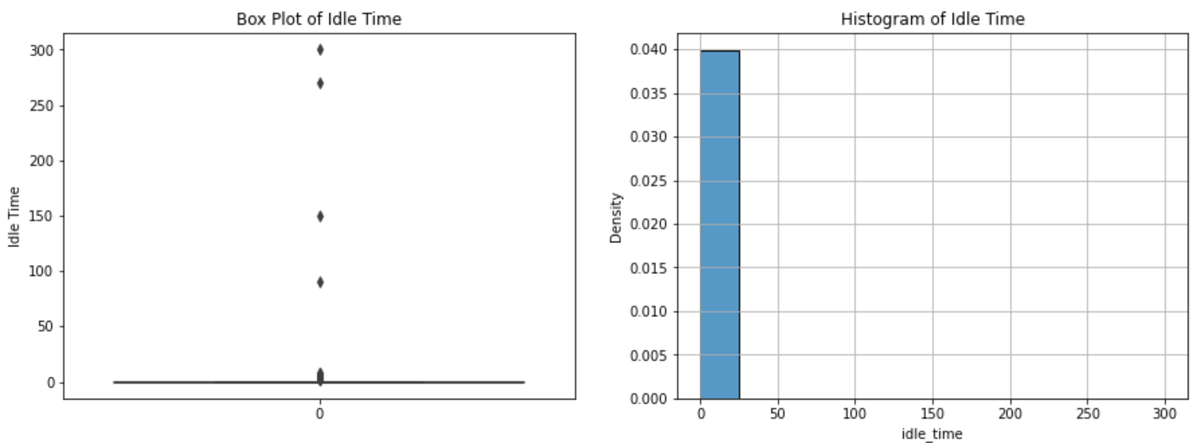
```
count    1197.000000
mean       76.124339
std        55.813726
min         0.000000
25%        24.000000
50%        66.000000
75%       116.000000
max       432.000000
Name: over_time_hour, dtype: float64
```



From the box plot and histogram, I noted an extreme outlier for over time of 432 minutes. An extreme value such as this may significantly affect the mean and standard deviation, as well as subsequent analysis and results.

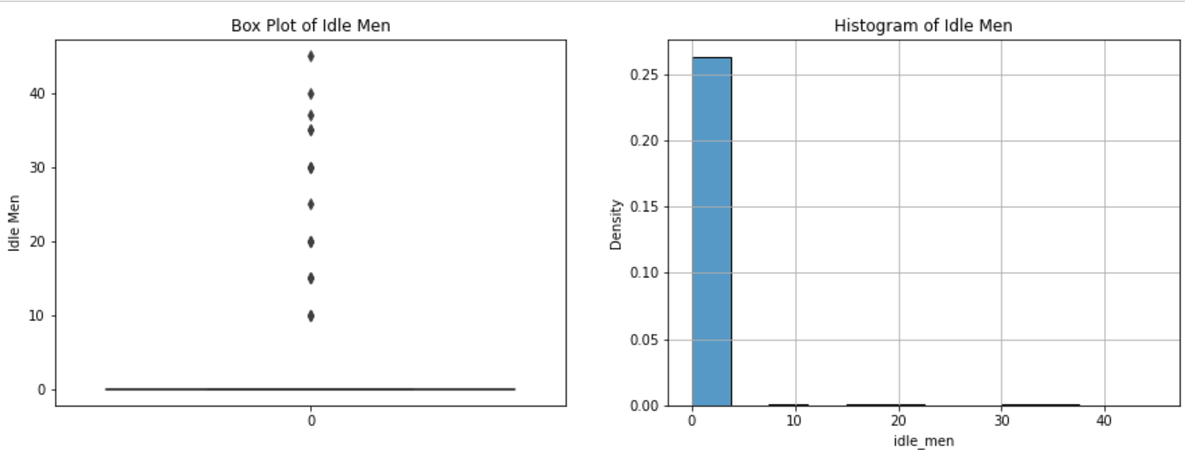
In [52]:
```
# idle time
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Idle Time")
sns.boxplot(data = data.idle_time)
plt.ylabel("Idle Time")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Idle Time")
sns.histplot(data.idle_time, stat = "density")
plt.grid()
print(data.idle_time.describe())
```

```
count    1197.000000
mean        0.730159
std        12.709757
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       300.000000
Name: idle_time, dtype: float64
```



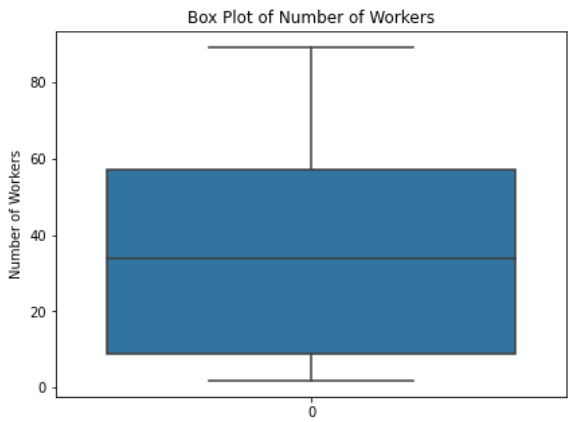
There is a high outlier for idle time of 300 minutes. An extreme value such as this may significantly affect the mean and standard deviation, as well as subsequent analysis.

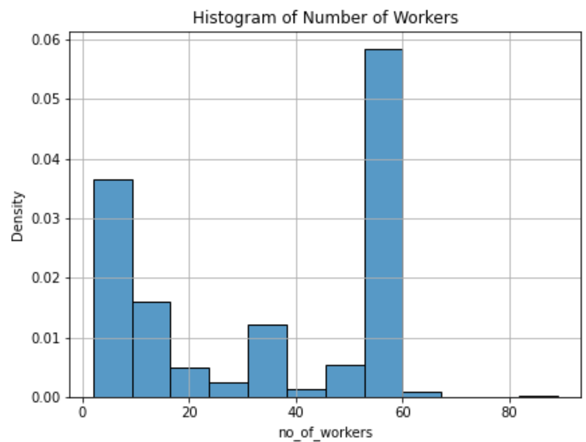
In [53]:
```
# idle men
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Idle Men")
sns.boxplot(data = data.idle_men)
plt.ylabel("Idle Men")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Idle Men")
sns.histplot(data.idle_men, stat = "density")
plt.grid()
```



From the box plot and histogram, I observed a handful of outliers for the idle men predictor. The distribution may be be skewed to the right.

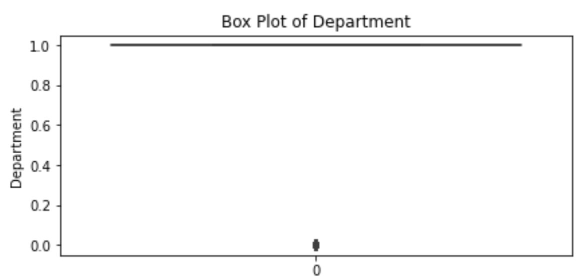
In [54]:
```python
# number of workers
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Number of Workers")
sns.boxplot(data = data.no_of_workers)
plt.ylabel("Number of Workers")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Number of Workers")
sns.histplot(data.no_of_workers, stat = "density")
plt.grid()
```

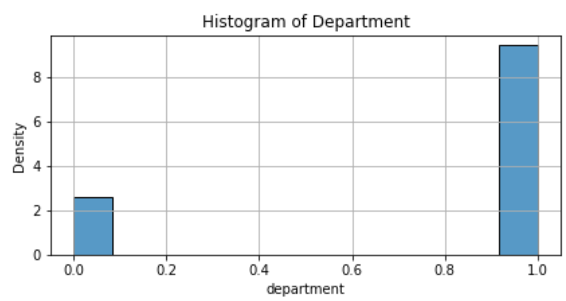




(2) Categorical Variables

In [55]:
```python
# department
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Department")
sns.boxplot(data = data.department)
plt.ylabel("Department")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Department")
sns.histplot(data.department, stat = "density")
plt.grid()
```

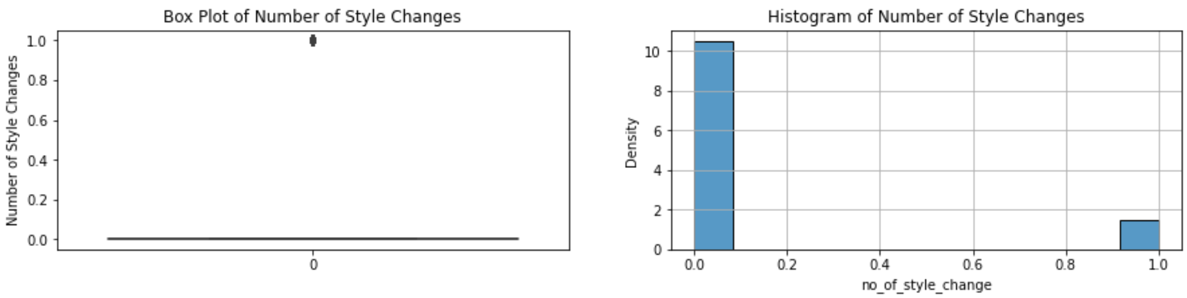




There is not a very balanced split between the two department categories, finishing and sewing. Ideally, the data would be split about 50-50 between the two departments because I would like a fair representation of each group when doing my analysis.

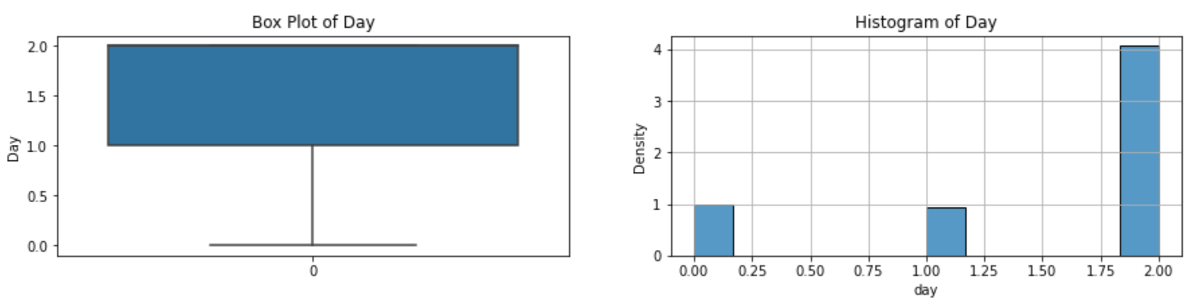
In [56]:
```python
# style changes
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Number of Style Changes")
sns.boxplot(data = data.no_of_style_change)
plt.ylabel("Number of Style Changes")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Number of Style Changes")
sns.histplot(data.no_of_style_change, stat = "density")
plt.grid()
print(data.no_of_style_change.value_counts())
```

```
0    1050
1     147
Name: no_of_style_change, dtype: int64
```

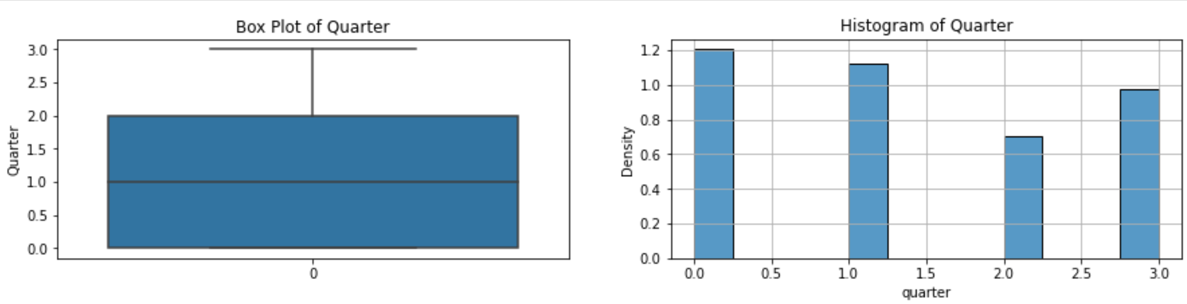


There is an unbalanced split between the two categories for number of style changes. Ideally, the data would be split about 50-50 between the style of a product being changed and not being changed because I would like a fair representation of each group when doing my analysis.

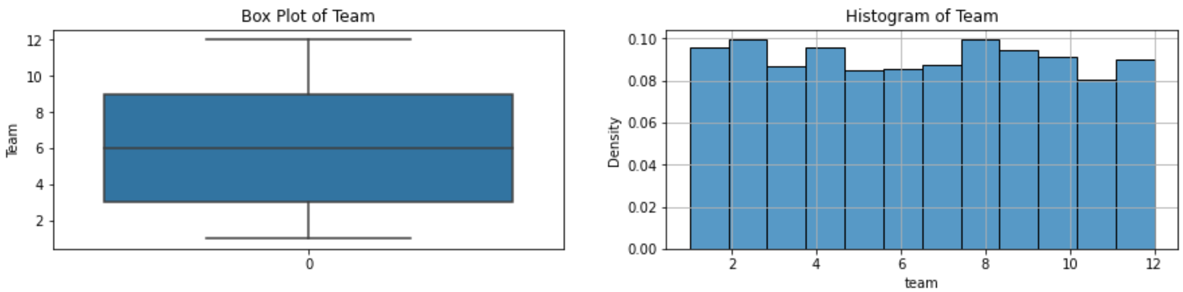
In [57]:
```python
# day
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Day")
sns.boxplot(data = data.day)
plt.ylabel("Day")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Day")
sns.histplot(data.day, stat = "density")
plt.grid()
```



In [58]:
```python
# quarter
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Quarter")
sns.boxplot(data = data.quarter)
plt.ylabel("Quarter")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Quarter")
sns.histplot(data.quarter, stat = "density")
plt.grid()
```



In [59]:
```
# team
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Team")
sns.boxplot(data = data.team)
plt.ylabel("Team")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Team")
sns.histplot(data.team, stat = "density")
plt.grid()
```



For quaters and teams, I found that the number of samples selected from different categories were relatively uniform, which could made the analisys more reasonable.

4 Variable Selecting

4.1 Boruta Algorithm

In [60]:
```
pip install Boruta
```

```
Requirement already satisfied: Boruta in /opt/anaconda3/lib/python3.9/site-packages (0.3)
Requirement already satisfied: scipy>=0.17.0 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (1.7.1)
Requirement already satisfied: scikit-learn>=0.17.1 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (0.2
4.2)
Requirement already satisfied: numpy>=1.10.4 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (1.20.3)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn>=0.17.1
->Boruta) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn
>=0.17.1->Boruta) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [61]:
```
pip install BorutaShap
```

```
Requirement already satisfied: BorutaShap in /opt/anaconda3/lib/python3.9/site-packages (1.0.16)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (4.62.3)
Requirement already satisfied: statsmodels in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.12.2)
Requirement already satisfied: shap>=0.34.0 in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.41.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.20.3)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.3.4)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (3.4.3)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.24.2)
Requirement already satisfied: seaborn in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.11.2)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.7.1)
Requirement already satisfied: slicer==0.0.7 in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->Borut
aShap) (0.0.7)
Requirement already satisfied: packaging>20.9 in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->Boru
taShap) (21.0)
Requirement already satisfied: cloudpickle in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->BorutaS
hap) (2.0.0)
Requirement already satisfied: numba in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->BorutaShap)
(0.54.1)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/anaconda3/lib/python3.9/site-packages (from packaging>20.9->
shap>=0.34.0->BorutaShap) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3/lib/python3.9/site-packages (from matplotlib->Bor
utaShap) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/python3.9/site-packages (from matplotlib->BorutaSh
ap) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in /opt/anaconda3/lib/python3.9/site-packages (from matplotlib->BorutaS
hap) (8.4.0)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/lib/python3.9/site-packages (from matplotlib->
BorutaShap) (2.8.2)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.9/site-packages (from cycler>=0.10->matplotlib->Bor
utaShap) (1.16.0)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.9/site-packages (from numba->shap>=0.34.0->B
orutaShap) (58.0.4)
Requirement already satisfied: llvmlite<0.38,>=0.37.0rc1 in /opt/anaconda3/lib/python3.9/site-packages (from numba->
shap>=0.34.0->BorutaShap) (0.37.0)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas->BorutaShap)
(2021.3)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn->Boruta
Shap) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn
->BorutaShap) (2.2.0)
Requirement already satisfied: patsy>=0.5 in /opt/anaconda3/lib/python3.9/site-packages (from statsmodels->BorutaSha
p) (0.5.2)
Note: you may need to restart the kernel to use updated packages.
```

In [62]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.9/site-packages (0.24.2)
Requirement already satisfied: scipy>=0.19.1 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.7.
1)
Requirement already satisfied: numpy>=1.13.3 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.2
0.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-lear
n) (2.2.0)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.1.
0)
Note: you may need to restart the kernel to use updated packages.
```

pip install ipywidgets

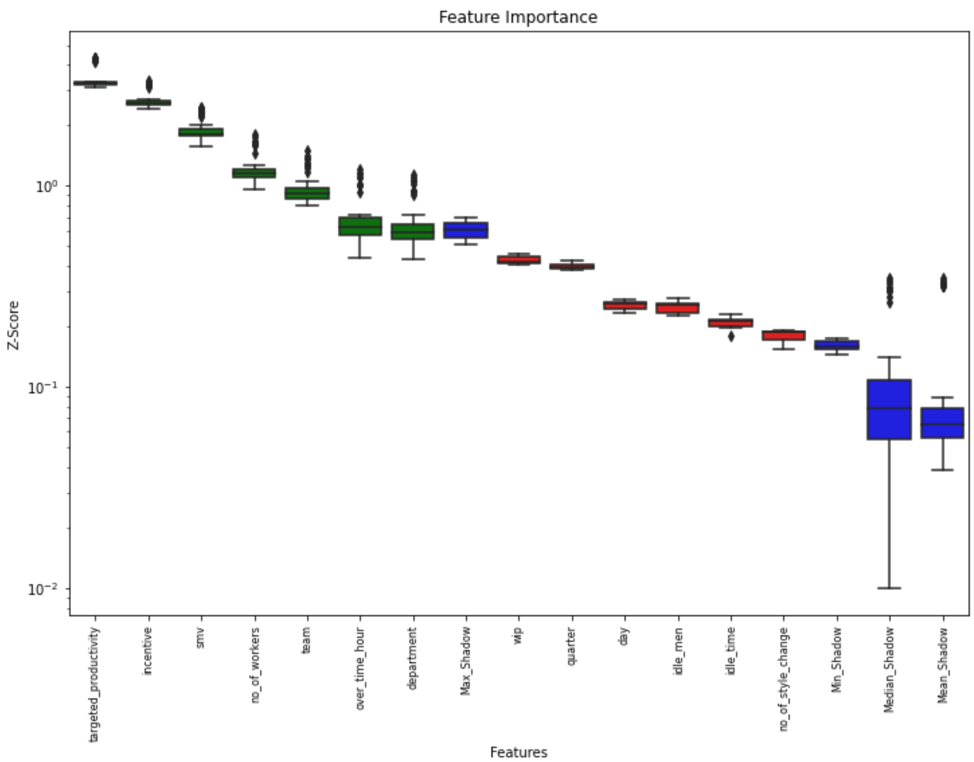
In [63]:
```
from BorutaShap import BorutaShap
from sklearn.ensemble import RandomForestRegressor
```

In [64]:
```
boruta_data = data[['actual_productivity','quarter','department','day','team','targeted_productivity', 'smv', 'wip', '
                'no_of_style_change', 'incentive', 'idle_time', 'idle_men','no_of_workers']].copy()
x = boruta_data.iloc[:,1:]
y = boruta_data['actual_productivity']
print(y.shape)
print(x.shape)
```

```
(1197,)
(1197, 13)
```

In [65]:
```
Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
Feature_Selector.fit(X=x, y=y, n_trials=50, random_state=0)
Feature_Selector.plot(which_features='all')
```

```
  0%|          | 0/50 [00:00<?, ?it/s]

7 attributes confirmed important: ['incentive', 'smv', 'team', 'department', 'no_of_workers', 'targeted_productivit
y', 'over_time_hour']
6 attributes confirmed unimportant: ['wip', 'quarter', 'day', 'no_of_style_change', 'idle_men', 'idle_time']
0 tentative attributes remains: []
```



In [66]:
```
Feature_Selector.Subset()
```

Out[66]:

	incentive	smv	team	department	no_of_workers	targeted_productivity	over_time_hour
0	98	26.16	8	1	59.0	0.80	118.0
1	0	3.94	1	0	8.0	0.75	16.0
2	50	11.41	11	1	30.5	0.80	61.0
3	50	11.41	12	1	30.5	0.80	61.0
4	50	25.90	6	1	56.0	0.80	32.0
...	...	...	...	...	...	...	...
1192	0	2.90	10	1	8.0	0.75	16.0
1193	0	3.90	8	1	8.0	0.70	16.0
1194	0	3.90	7	1	8.0	0.65	16.0
1195	0	2.90	9	1	15.0	0.75	30.0
1196	0	2.90	6	1	6.0	0.70	12.0

1197 rows × 7 columns

Based on the Boruta Algorithm test, the top 2 predictors selected are 'targeted_productivity' and 'incentive'.

4.2 Mallows Cp

Because the purpose is to identify the two best variables, for efficiency and convenience, I did not include all variables into Mallows Cp test, instead, I selected 7 variables which preformed well in the the Boruta Algorithm test.

7 attributes confirmed important in Boruta Algorithm test: ('no_of_workers', 'smv', 'department', 'team', 'targeted_productivity', 'over_time_hour', 'incentive')

In [67]:
```
pip install RegscorePy
```

```
Requirement already satisfied: RegscorePy in /opt/anaconda3/lib/python3.9/site-packages (1.1)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/site-packages (from RegscorePy) (1.3.4)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from RegscorePy) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas->Re
gscorePy) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas->RegscorePy)
(2021.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.9/site-packages (from python-dateutil>=2.7.3->
pandas->RegscorePy) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [68]:
```
from RegscorePy import mallow
import itertools
```

In [69]:
```
workers', 'smv', 'department', 'team', 'targeted_productivity', 'over_time_hour', 'incentive']].copy()

ductivity ~ no_of_workers + smv + department + team + targeted_productivity + over_time_hour + incentive', data = data)




es', 'CP'])
```

In [70]:
```
for L in range(1, len(subdat.columns[1:]) + 1):
    for subset in itertools.combinations(subdat.columns[1:],L):
        formula1 = 'actual_productivity ~ ' + '+'.join(subset)
        results = smf.ols(formula = formula1, data = data).fit()
        y_sub = results.fittedvalues
        p = len(subset)+1
        cp = mallow.mallow(y, y_pred, y_sub, k, p)
        storage_cp = storage_cp.append({'Variables': subset, 'CP': cp}, ignore_index = True)
```

In [71]:
```
storage_cp = storage_cp.sort_values(by ='CP', axis = 0)
storage_cp.head()
```

Out[71]:

	Variables	CP
120	(no_of_workers, smv, department, team, targete...	6.130438
126	(no_of_workers, smv, department, team, targete...	8.000000
98	(no_of_workers, smv, department, team, targete...	11.866429
119	(no_of_workers, smv, department, team, targete...	13.606823
105	(no_of_workers, smv, team, targeted_productivi...	21.759361

Based on the Boruta Algorithm test, the top 2 predictors selected are 'targeted_productivity' and 'team'.

5 Model Building and Evaluating

After using Boruta Algorithm and Mallows Cp method, I have identified three variables: 'targeted_productivity', 'incentive', and 'team'.

5.1 Model Building

(1) Model 1 : actual_productivity ~ targeted_productivity

In [72]:

```
reg1 = smf.ols('actual_productivity ~ targeted_productivity', data = data)
results1 = reg1.fit()
print(results1.params)
results1.summary()
```

```
Intercept                0.186787
targeted_productivity    0.751479
dtype: float64
```

Out[72]:

OLS Regression Results

Dep. Variable:	actual_productivity	**R-squared:**	0.178
Model:	OLS	**Adj. R-squared:**	0.177
Method:	Least Squares	**F-statistic:**	258.3
Date:	Tue, 18 Oct 2022	**Prob (F-statistic):**	9.00e-53
Time:	19:39:38	**Log-Likelihood:**	509.00
No. Observations:	1197	**AIC:**	-1014.
Df Residuals:	1195	**BIC:**	-1004.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1868	0.034	5.427	0.000	0.119	0.254
targeted_productivity	0.7515	0.047	16.072	0.000	0.660	0.843

Omnibus:	102.766	**Durbin-Watson:**	1.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	168.973
Skew:	-0.614	**Prob(JB):**	2.03e-37
Kurtosis:	4.371	**Cond. No.**	15.7

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

(2) Model 2 : actual_productivity ~ incentive

In [73]:
```
reg2 = smf.ols('actual_productivity ~ incentive', data = data)
results2 = reg2.fit()
print(results2.params)
results2.summary()
```

```
Intercept    0.731905
incentive    0.000083
dtype: float64
```

Out[73]:

OLS Regression Results

Dep. Variable:	actual_productivity	**R-squared:**	0.006
Model:	OLS	**Adj. R-squared:**	0.005
Method:	Least Squares	**F-statistic:**	7.042
Date:	Tue, 18 Oct 2022	**Prob (F-statistic):**	0.00807
Time:	19:39:38	**Log-Likelihood:**	395.39
No. Observations:	1197	**AIC:**	-786.8
Df Residuals:	1195	**BIC:**	-776.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.7319	0.005	141.515	0.000	0.722	0.742
incentive	8.337e-05	3.14e-05	2.654	0.008	2.17e-05	0.000

Omnibus:	105.802	**Durbin-Watson:**	0.745
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	132.581
Skew:	-0.798	**Prob(JB):**	1.62e-29
Kurtosis:	3.335	**Cond. No.**	169.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

(3) Model 3 : actual_productivity ~ team

In [74]:
```
reg3 = smf.ols('actual_productivity ~ team', data = data)
results3 = reg3.fit()
print(results3.params)
results3.summary()
```

```
Intercept    0.783248
team        -0.007493
dtype: float64
```

Out[74]:

OLS Regression Results

Dep. Variable:	actual_productivity	**R-squared:**	0.022
Model:	OLS	**Adj. R-squared:**	0.021
Method:	Least Squares	**F-statistic:**	27.04
Date:	Tue, 18 Oct 2022	**Prob (F-statistic):**	2.34e-07
Time:	19:39:38	**Log-Likelihood:**	405.26
No. Observations:	1197	**AIC:**	-806.5
Df Residuals:	1195	**BIC:**	-796.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.7832	0.011	74.458	0.000	0.763	0.804
team	-0.0075	0.001	-5.200	0.000	-0.010	-0.005

Omnibus:	114.442	**Durbin-Watson:**	0.745
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	146.211
Skew:	-0.839	**Prob(JB):**	1.78e-32
Kurtosis:	3.336	**Cond. No.**	15.6

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The variable team is a categorical variable that the value is relatively arbitrary. I believe that directly including it into the model has no practical explanatory significance. However, I can draw the conclusion that the specific production team a worker is assigned to will have an impact on the worker's actual productivity.

By comparing the R-squared, adjusted R-squared, AIC and BIC of the first two models, I found that model 1 has a higher R-squared and adjusted R-squared while the AIC and BIC are lower. Combined with the fact that targeted_productivity is selected by both Boruta and Mallows, I will use transformed targeted_productivity to construct the model again.

(4) Model 4 : actual_productivity ~ transformed targeted_productivity

In [75]:
```
#add transformed targeted productivity data as new column in dataframe
data['bc_targeted_productivity'] = bc_targeted_productivity

reg4 = smf.ols('actual_productivity ~ bc_targeted_productivity', data = data)
results4 = reg4.fit()
print(results4.params)
results4.summary()
```

```
Intercept                   1.490679
bc_targeted_productivity    5.590285
dtype: float64
```

Out[75]:

OLS Regression Results

Dep. Variable:	actual_productivity	**R-squared:**	0.192
Model:	OLS	**Adj. R-squared:**	0.192
Method:	Least Squares	**F-statistic:**	284.6
Date:	Tue, 18 Oct 2022	**Prob (F-statistic):**	1.94e-57
Time:	19:39:38	**Log-Likelihood:**	519.72
No. Observations:	1197	**AIC:**	-1035.
Df Residuals:	1195	**BIC:**	-1025.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.4907	0.045	33.113	0.000	1.402	1.579
bc_targeted_productivity	5.5903	0.331	16.870	0.000	4.940	6.240

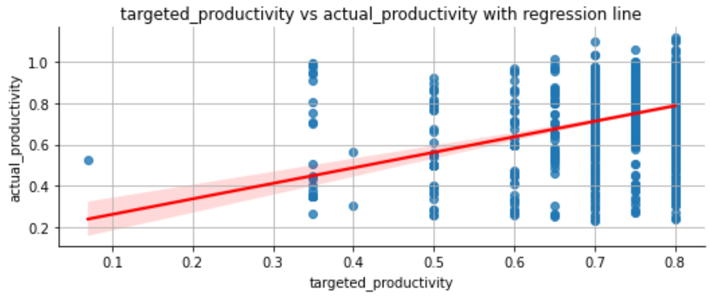
Omnibus:	110.374	**Durbin-Watson:**	1.004
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	161.803
Skew:	-0.695	**Prob(JB):**	7.33e-36
Kurtosis:	4.144	**Cond. No.**	74.4

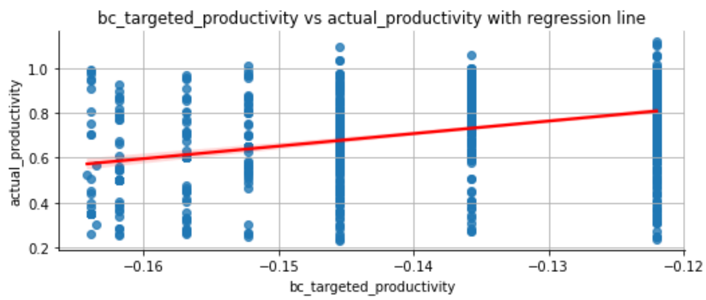
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

5.2 Linearly Transformed Data Evaluating

(1) Scatter Plot: Scatterplots between Original/Transformed Data and Predicted Variable

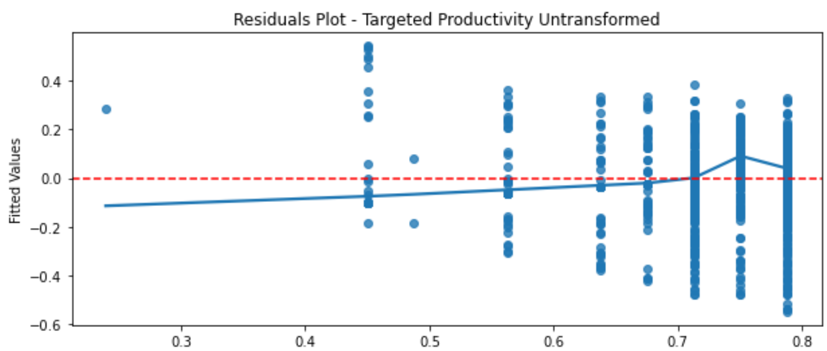
In [76]:
```
#scatter plot with actual productivity and untransformed data
sns.lmplot(data = data, x='targeted_productivity', y='actual_productivity',
line_kws = {'color':'red'},height = 3, aspect = 2.5, ci = 95)
plt.title('targeted_productivity vs actual_productivity with regression line')
plt.grid()
#scatter plot with actual productivity and transformed data
sns.lmplot(data = data, x='bc_targeted_productivity', y='actual_productivity',
line_kws = {'color':'red'},height = 3, aspect = 2.5, ci = 95)
plt.title('bc_targeted_productivity vs actual_productivity with regression line')
plt.grid()
plt.show()
```

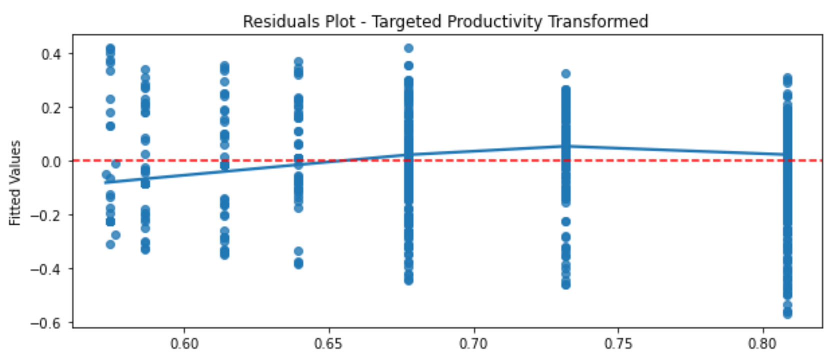




(2) Residual Plot: Original/Transformed Residuals and Fitted Values

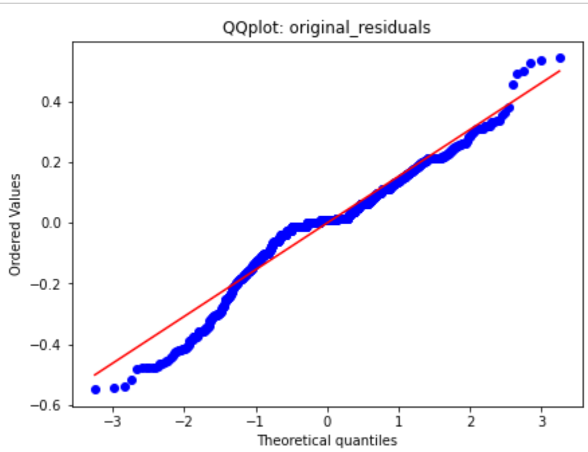
In [77]:
```
#original data
plt.figure(figsize = (10, 4))
sns.regplot(x = results1.fittedvalues, y = results1.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.ylabel("Fitted Values")
plt.title("Residuals Plot - Targeted Productivity Untransformed")
plt.show()

#transformed data
plt.figure(figsize = (10, 4))
sns.regplot(x = results4.fittedvalues, y = results4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.ylabel("Fitted Values")
plt.title("Residuals Plot - Targeted Productivity Transformed")
plt.show()
```

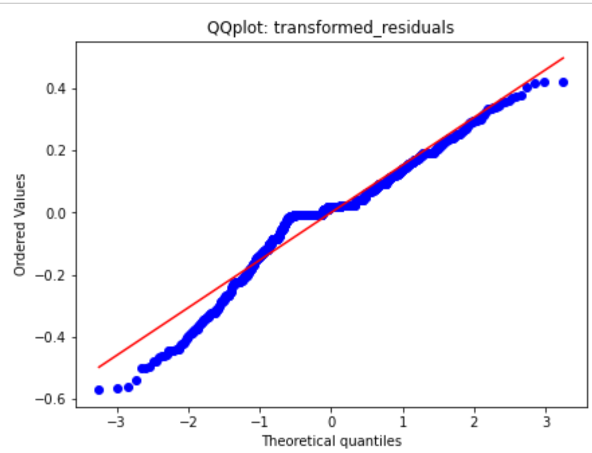




(3) Residuals Analysis

In [78]:
```
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
stats.probplot(results1.resid, dist = "norm", plot = plt)
plt.title('QQplot: original_residuals')
ax2 = fig.add_subplot(1,2,2)
stats.probplot(results4.resid, dist = "norm", plot = plt)
plt.title('QQplot: transformed_residuals')
plt.show()
```



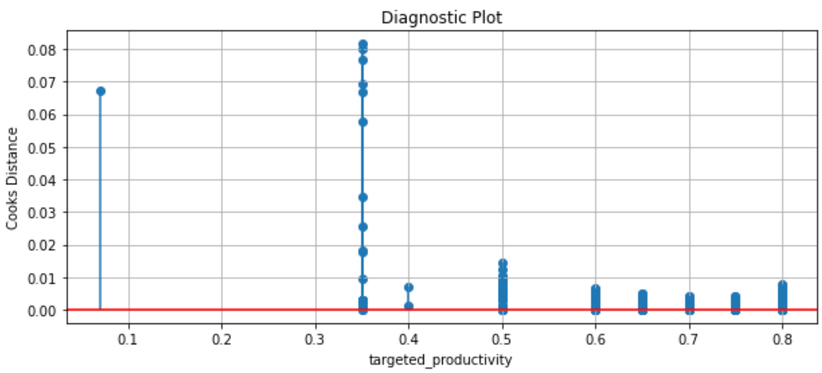


To transform nonlinear data, I performed a Box-Cox evaluation. For my selected predictor variable 'targeted_productivity,' this returned a power transformation with a lambda of value of 6.09. Based on my evaluation of the linearly transformed data, I are choosing to build my regression model with the original data. Looking at the correlation plot, the residuals plot, and the QQ plot, I have determined that the marginal improvement in linearity, constant variance, and normality is not enough to justify transforming my data with a power of 6.09, as it would leave us with a largely uninterpretable model.

5.3 Cook's Distance Plots

Model 1: Cook's Distance Plots

In [79]:
```
influence = results1.get_influence()
cooks = influence.cooks_distance

plt.figure(figsize = (10, 4))
plt.scatter(data['targeted_productivity'], cooks[0])
plt.axhline(0, color = 'red')
plt.vlines(x = data['targeted_productivity'], ymin = 0, ymax = cooks[0])
plt.xlabel('targeted_productivity')
plt.ylabel('Cooks Distance')
plt.title("Diagnostic Plot")
plt.grid()
```

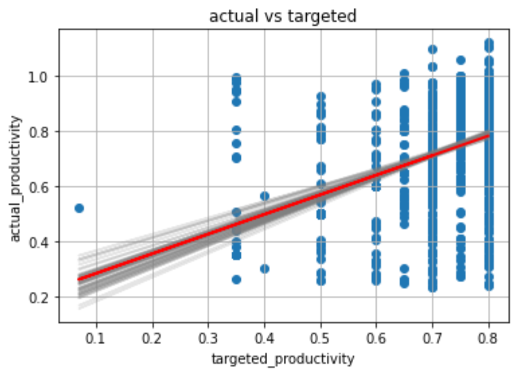


5.4 Bootstrapping

In [80]:
```
from scipy.stats import bootstrap
from sklearn.linear_model import LinearRegression
```

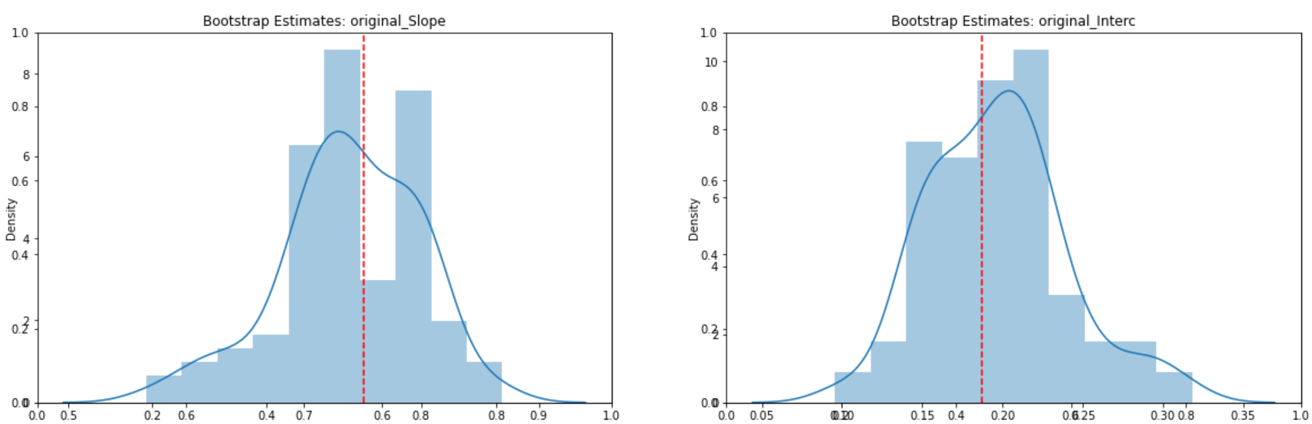
Model 1: Bootstrapping

In [81]:
```python
boot_slopes = []
boot_interc = []
n_boots = 100
n_points = data.shape[0]
plt.figure()
for _ in range(n_boots):
    sample_df = data.sample(n=n_points, replace=True)
    ols_model_temp = smf.ols(formula = 'actual_productivity ~ targeted_productivity', data=sample_df)
    results_temp = ols_model_temp.fit()

    boot_interc.append(results_temp.params[0])
    boot_slopes.append(results_temp.params[1])

    y_pred_temp = ols_model_temp.fit().predict(sample_df['targeted_productivity'])
    plt.plot(sample_df['targeted_productivity'], y_pred_temp, color='grey', alpha=0.2)

y_pred = ols_model_temp.fit().predict(data['targeted_productivity'])
plt.scatter(data['targeted_productivity'], data['actual_productivity'])
plt.plot(data['targeted_productivity'], y_pred, linewidth=2,color = 'red')
plt.grid(True)
plt.xlabel('targeted_productivity')
plt.ylabel('actual_productivity')
plt.title('actual vs targeted')
plt.show()
```



In [82]:
```python
fig,(ax1, ax2) = plt.subplots(1,2, figsize = (20,6))
ax1 = fig.add_subplot(1,2,1)
sns.distplot(boot_slopes, ax = ax1)
plt.axvline(x=0.751479,color='red', linestyle='--')
plt.title('Bootstrap Estimates: original_Slope')
ax2 = fig.add_subplot(1,2,2)
sns.distplot(boot_interc, ax = ax2)
plt.axvline(x=0.186787,color='red', linestyle='--')
plt.title('Bootstrap Estimates: original_Interc')
```

Out[82]:
```
Text(0.5, 1.0, 'Bootstrap Estimates: original_Interc')
```



Based on these two histograms, I found that the estimated parameter histograms realized by bootstrapping were close to normally distributed, and the initial estimated parameters represented by the red line are located in the center of the histograms, indicating that the estimated parameters I got initially are good.

5.5 Cross-Validation

Model 1: 5-Fold Cross-Validation

In [83]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

x = data[['targeted_productivity']]
y = data[['actual_productivity']]

regr = LinearRegression()
scores = cross_val_score(regr, x, y, cv=5, scoring='neg_mean_squared_error')
print('5-Fold CV MSE Scores:', scores)
```

```
5-Fold CV MSE Scores: [-0.01622384 -0.02687604 -0.03048272 -0.03201203 -0.02270309]
```

In [84]:
```
import math

avg_MSE = -scores.mean()
print("Average MSE:", avg_MSE)
avg_RSME = math.sqrt(-scores.mean())
print("Average RMSE:", avg_RSME)
```

```
Average MSE: 0.025659544042946326
Average RMSE: 0.16018596705999663
```

Based on the 5-Fold CV MSE scores, my model produced relatively consistent out of sample predictions for each fold.

By looking at the average MSE of my model across all five folds, I can get an overall estimate of how my model performed across all the data. Accounting for the fact that the MSE tends to penalize large errors much more than small errors, my model's average MSE of 0.0257 indicates a favorable performance evaluation.

Next, I looked at the RMSE in order to further evaluate how well my model fit the data. I found that on average, predicted productivity is off from actual productivity by approximately 0.16. Comparing the RMSE to the range of values of actual productivity, which spans from approximately 0 to 1, I conclude that my model can predict the data relatively accurately, but there is room for improvement and could better fit the dataset.

5.6 Testing and Training

In [85]:
```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score

x = data[['targeted_productivity']]
y = data[['actual_productivity']]

# Perform an OLS fit using all the data
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_

# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 0.102972540821329
MSE: 0.022512789138623427
RMSE: 0.15004262440594482
```

My model's out of sample prediction accuracy showed little variation between samples, as evidenced by the similar MSE and RMSE values I obtained from 5-fold cross validation and the Holdout Method. The consistency of these evaluation metrics indicates a positive assessment of my model's predictive performance.

Similarly, the low MAE score suggests that the values of actual productivity predicted by my model and the actual values are very close in absolute value.

6 Conclusion

Ultimately, I can draw several key economic insights from my model.

The interpretation of my intercept parameter suggests that the estimated value of actual productivity is 0.187 when targeted productivity is zero. Targeted productivity in this dataset represents a value strategically set by industrial engineers for their working team (Imran et al., 2019). Moreover, based on the slope parameter, my regression model predicts that for a one unit increase in targeted productivity, actual productivity is expected to increase by approximately 0.752, all else being equal.

One possible conclusion of the positive linear relationship between targeted productivity and actual productivity is that setting a higher goal for your team will incentivize them to work more, and therefore be more productive.

While my model is robust, I do not necessarily suggest that garment manufacturing companies looking to increase productivity should solely set a higher level of targeted productivity. For the model, I included only one variable, and there were variables combinations with scores in the variable selection analysis I performed. In addition, I also recommend for engineers to consider maximizing profit by minimizing production loss. In other words, by setting targeted productivity as close as possible to their team's actual productivity, which will range from team to team due to a number of factors including monetary incentives, type of team, number of employees on the team, and numbers of works in progress.

7 Reference

Al Imran, A., Rahim, M.S. and Ahmed, T. (2021) 'Mining the productivity data of the garment industry', Int. J. Business intelligence and Data Mining, Vol. 19, No. 13, pp.319-342

Al Imran, A., Amin, N., Rifat, R.I., Mehreen, S. (2019) 'Deep Neural Network Approach for Predicting the Productivity of Garment Employmees,' 2019 6th International Conference on Control, Decision and Decision and Information Technologies, April 23-26, 2019. Accessed 17 October 20222.