```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
%matplotlib inline
```

```
df=pd.read_csv("garments_worker_productivity.csv",header=0,index_col=0,parse_dates=True,squeeze=True)
```

```
<ipython-input-37-bc8db97eedde>:1: FutureWarning: The squeeze argument has been deprecated and will be remo

  df=pd.read_csv("garments_worker_productivity.csv",header=0,index_col=0,parse_dates=True,squeeze=True)
```

```
df.head()
```

	quarter	department	day	team	targeted_productivity	smv	wip	over_
date								
2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	
2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	
2015-01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	

```
df.shape
```

```
(1197, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1197 entries, 2015-01-01 to 2015-03-11
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   quarter                1197 non-null   object
 1   department             1197 non-null   object
 2   day                    1197 non-null   object
 3   team                   1197 non-null   int64
 4   targeted_productivity  1197 non-null   float64
 5   smv                    1197 non-null   float64
 6   wip                    691 non-null    float64
 7   over_time              1197 non-null   int64
 8   incentive              1197 non-null   int64
 9   idle_time              1197 non-null   float64
 10  idle_men               1197 non-null   int64
 11  no_of_style_change     1197 non-null   int64
 12  no_of_workers          1197 non-null   float64
 13  actual_productivity    1197 non-null   float64
dtypes: float64(6), int64(5), object(3)
memory usage: 140.3+ KB
```

There are mising values in wip column

```
df.isnull().sum()
```

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

```
df.describe()
```

	team	targeted_productivity	smv	wip	over_time	i
count	1197.000000	1197.000000	1197.000000	691.000000	1197.000000	11
mean	6.426901	0.729632	15.062172	1190.465991	4567.460317	
std	3.463963	0.097891	10.943219	1837.455001	3348.823563	1
min	1.000000	0.070000	2.900000	7.000000	0.000000	
25%	3.000000	0.700000	3.940000	774.500000	1440.000000	
50%	6.000000	0.750000	15.260000	1039.000000	3960.000000	
75%	9.000000	0.800000	24.260000	1252.500000	6960.000000	
max	12.000000	0.800000	54.560000	23122.000000	25920.000000	36

In order to have a quick overwiev of the data pairplot diagram will be used here

Categorical features

Quarter,department,team and day are categorical features

```
categorical_cols = ['quarter', 'department', 'day', 'team','no_of_style_change']
```

```
df.head()
```

	quarter	department	day	team	targeted_productivity	smv	wip	over_
date								
2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	
2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	

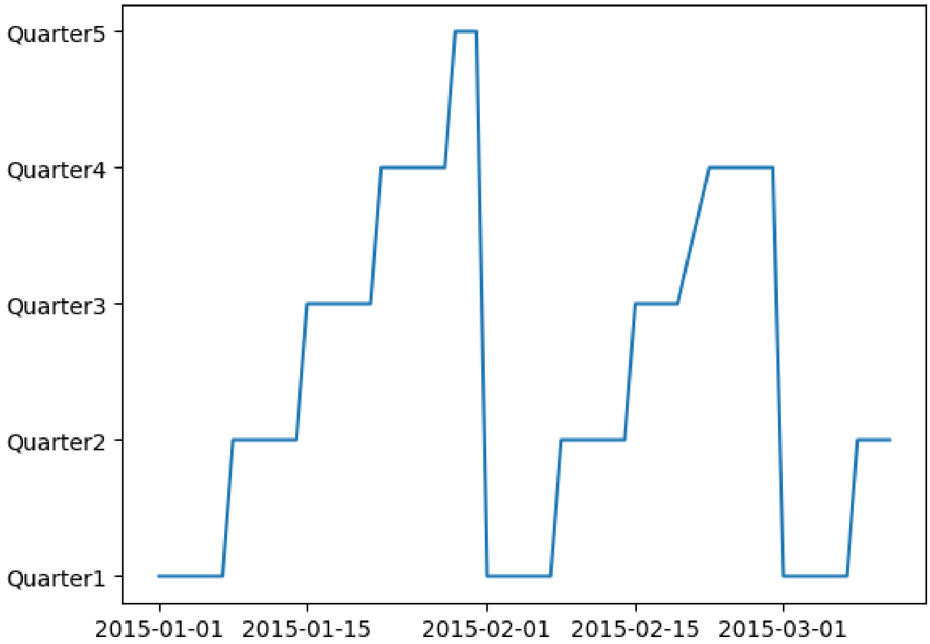
Quater1

```
df['quarter'].value_counts()
```

```
Quarter1    360
Quarter2    335
Quarter4    248
Quarter3    210
Quarter5     44
Name: quarter, dtype: int64
```

There are 5 quarters such as quarter1, quarter2, quarter3, quarter4, quarter5 which are not evenly distributed.

```
pyplot.plot(df.index,df.quarter)
plt.show()
```



When we checked the dates it can be observed that there is a repeated pattern for all quarters with time except Quarter5. We need to look into Quarter5 deeply. There should be a reason for that exception

```
df_1=df[df['quarter']=='Quarter5']
```

```
df_1.shape
```

```
(44, 14)
```

```
df_1.index
```

```
DatetimeIndex(['2015-01-29', '2015-01-29', '2015-01-29', '2015-01-29',
               '2015-01-29', '2015-01-29', '2015-01-29', '2015-01-29',
```

```
               '2015-01-29', '2015-01-29', '2015-01-29', '2015-01-29',
               '2015-01-29', '2015-01-29', '2015-01-29', '2015-01-29',
               '2015-01-29', '2015-01-29', '2015-01-29', '2015-01-29',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31',
               '2015-01-31', '2015-01-31', '2015-01-31', '2015-01-31'],
              dtype='datetime64[ns]', name='date', freq=None)
```

Quarter5 contains 2 days as 29th and 31th of January.

Department

```
df.department.value_counts()
```

```
sweing        691
finishing     257
finishing     249
Name: department, dtype: int64
```

There are three departments namely sweing,finishing and finishing.But we need to combine them into two group.

```
df=df.replace(['finishing '], ['finishing'])
```

```
df.department.value_counts()
```

```
sweing       691
finishing    506
Name: department, dtype: int64
```

Day

```
df.day.value_counts()
```

```
Wednesday    208
Sunday       203
Tuesday      201
Thursday     199
Monday       199
Saturday     187
Name: day, dtype: int64
```

Friday is not a working day

Numeric features

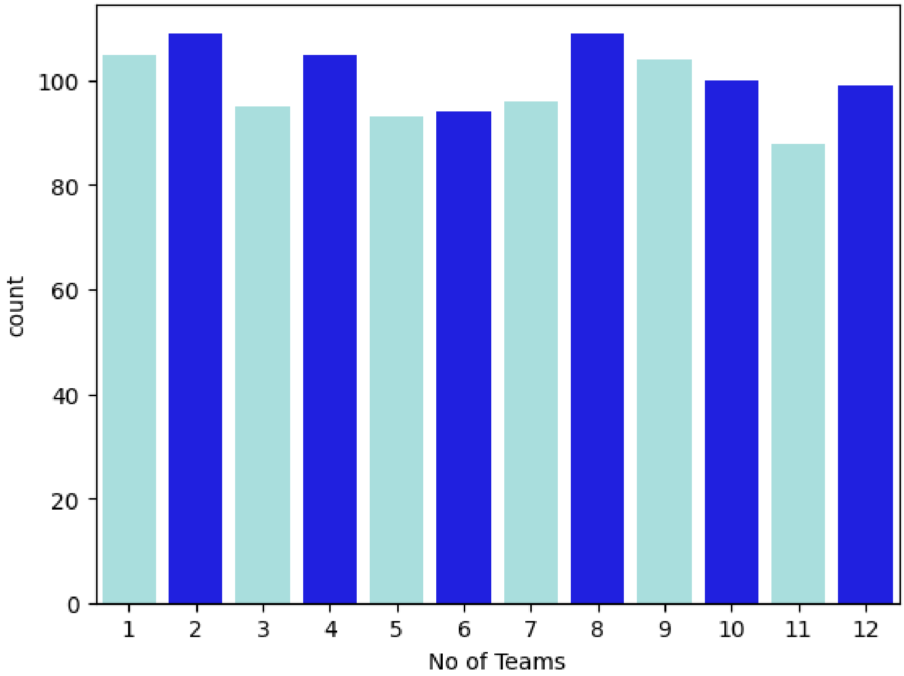
```
df.select_dtypes(include=np.number).columns.tolist()
```

```
['team',
 'targeted_productivity',
 'smv',
 'wip',
 'over_time',
 'incentive',
 'idle_time',
 'idle_men',
 'no_of_style_change',
 'no_of_workers',
 'actual_productivity']
```

In this case we have 11 numeric features as given above but 'no_of_style_change' will be handled as categorical feature.

Team

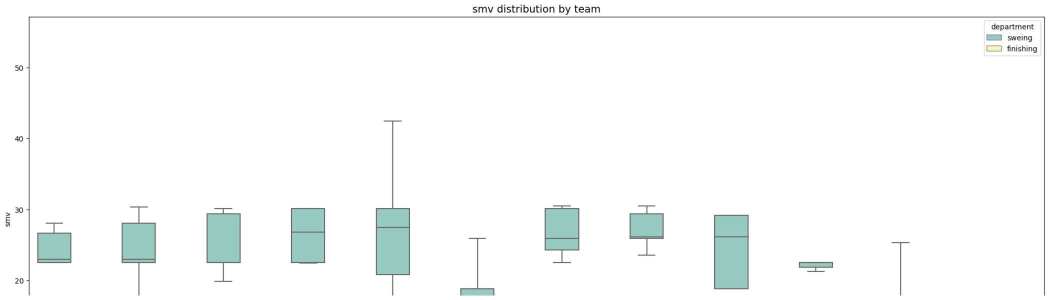
```
ax = sns.countplot(x = 'team', data = df, palette=["#A0E7E5", "#0000FF"])
plt.xlabel('No of Teams')
plt.show()
```



There are 12 teams.Because it is highly desirable among the decision makers in the garment industry to track, analyse and predict the productivity performance of the working teams in their factories our analysis will be on team basis.

SMV

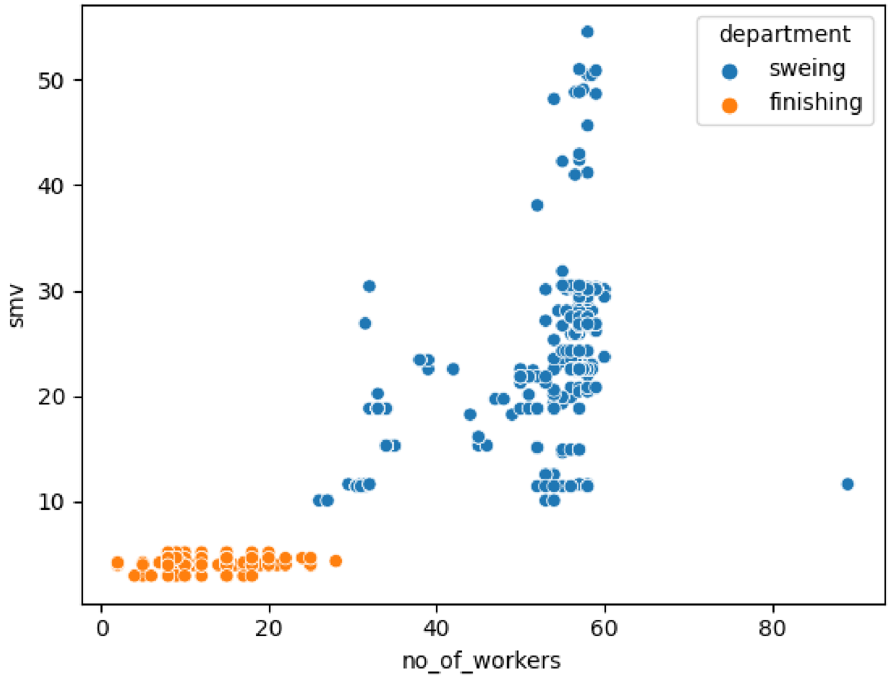
```
plt.figure(figsize=(25, 10))
palette = "Set3"

sns.boxplot(x="team", y="smv", hue="department", data = df,
     palette = palette, fliersize = 0)

plt.title('smv distribution by team',fontsize= 14)
plt.show()
```



When we look into smv boxplot on team basis with department seperation, it can be clearly seen that while there are fluctuations between teams in the sewing department, the finishing department has almost evenly distributed smv values for each team.

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

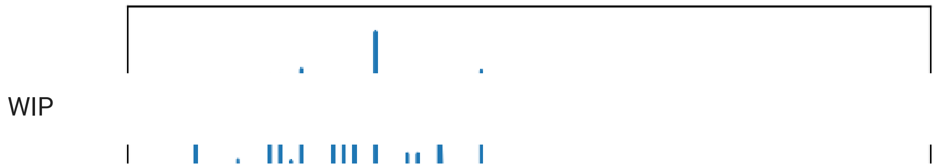
```
<Axes: xlabel='no_of_workers', ylabel='smv'>
```



For the finishing department SMV does not change

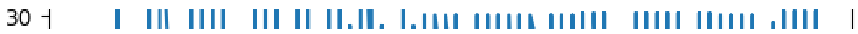
```
pyplot.plot(df.index,df.smv)
```

```
[<matplotlib.lines.Line2D at 0x7f11afdce1c0>]
```



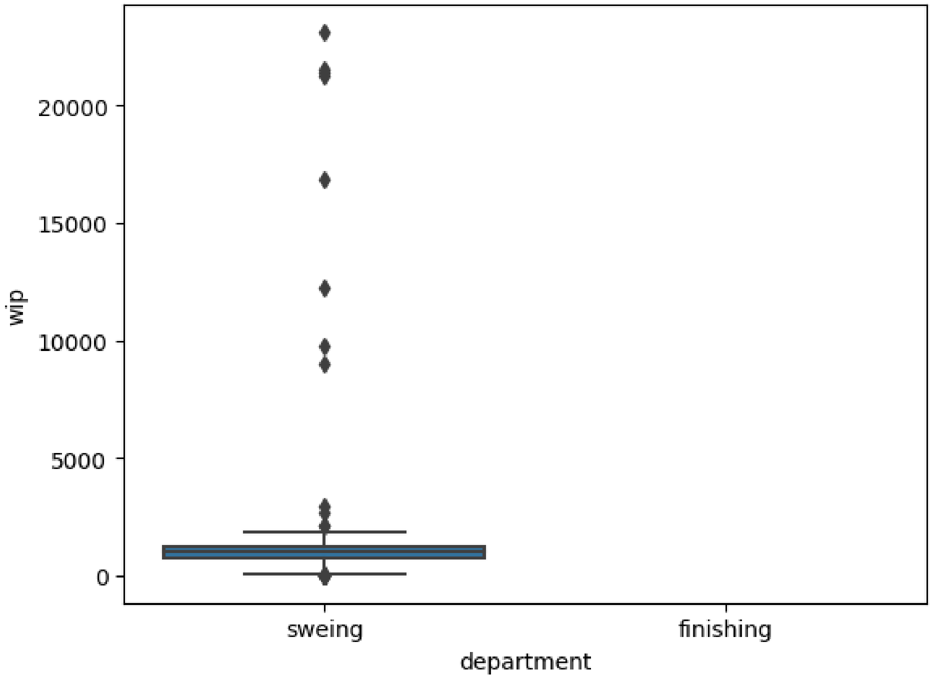

```
df.wip.isnull().sum()
```

```
506
```




```
sns.boxplot(x='department',y='wip',data=df)
```

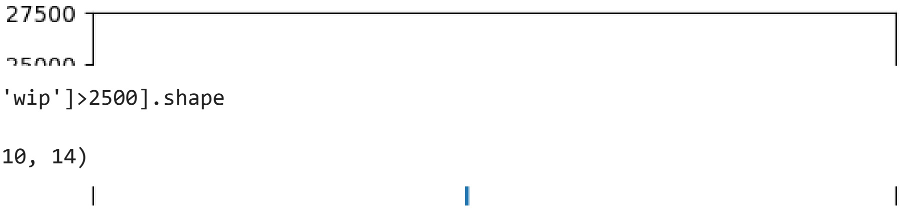
```
<Axes: xlabel='department', ylabel='wip'>
```



All null values belongs to the finishing department. The finishing department needs to get a work from the sewing department. This result could mean that the finishing department has no work in progress while waiting for work from the sewing department. So we can replace the null values with zero

```
pyplot.plot(df.index,df.wip)
plt.yticks(np.arange(0,30000,step=2500))
```

```
([<matplotlib.axis.YTick at 0x7f11afd2a490>,
  <matplotlib.axis.YTick at 0x7f11afd25df0>,
  <matplotlib.axis.YTick at 0x7f11afd1beb0>,
  <matplotlib.axis.YTick at 0x7f11afd1b1f0>,
  <matplotlib.axis.YTick at 0x7f11afcdd790>,
  <matplotlib.axis.YTick at 0x7f11afcddf70>,
  <matplotlib.axis.YTick at 0x7f11afce2790>,
  <matplotlib.axis.YTick at 0x7f11afce2f70>,
  <matplotlib.axis.YTick at 0x7f11afce2550>,
  <matplotlib.axis.YTick at 0x7f11afcdd5e0>,
  <matplotlib.axis.YTick at 0x7f11afcecc10>,
  <matplotlib.axis.YTick at 0x7f11afcf7430>],
 [Text(0, 0, '0'),
  Text(0, 2500, '2500'),
  Text(0, 5000, '5000'),
  Text(0, 7500, '7500'),
  Text(0, 10000, '10000'),
  Text(0, 12500, '12500'),
  Text(0, 15000, '15000'),
  Text(0, 17500, '17500'),
  Text(0, 20000, '20000'),
  Text(0, 22500, '22500'),
  Text(0, 25000, '25000'),
  Text(0, 27500, '27500')])
```




```
df[df['wip']>2500].shape
```

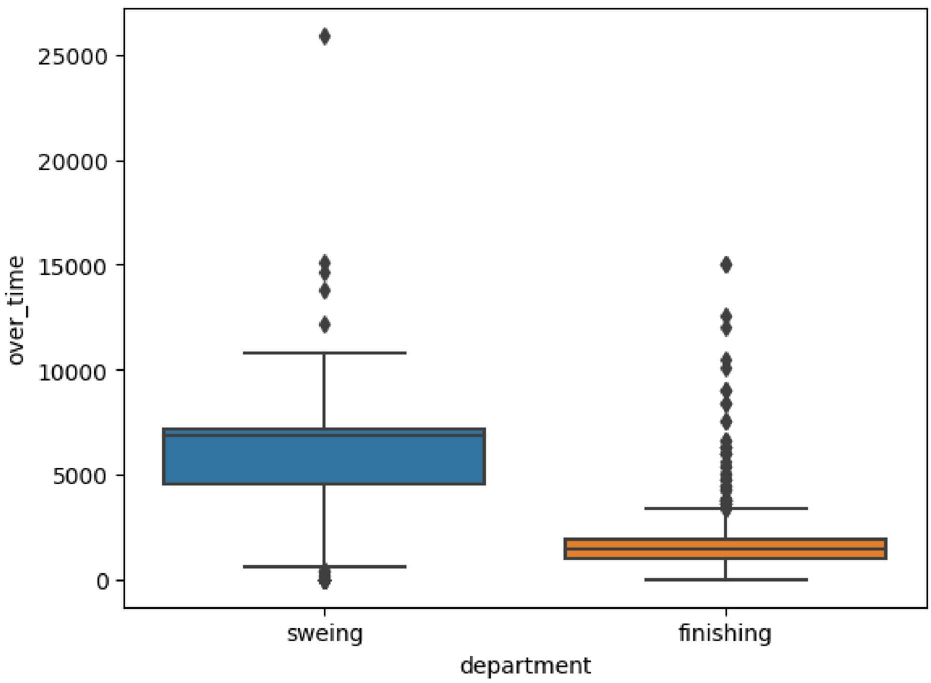
```
(10, 14)
```

```
df[df['wip']>2500]
```

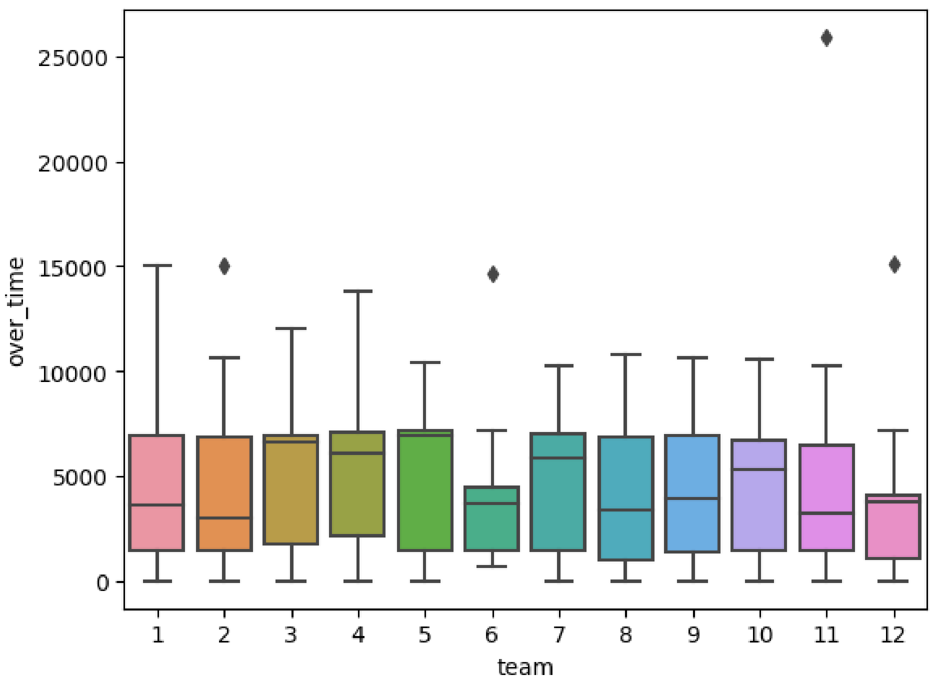
date	quarter	department	day	team	targeted_productivity	smv	wip	over_
2015-02-02	Quarter1	sweing	Monday	1	0.80	22.94	16882.0	
2015-02-02	Quarter1	sweing	Monday	2	0.80	22.52	21385.0	
2015-02-02	Quarter1	sweing	Monday	3	0.80	22.52	21266.0	
2015-02-02	Quarter1	sweing	Monday	10	0.80	22.52	21540.0	
2015-02-02	Quarter1	sweing	Monday	12	0.80	15.26	12261.0	
2015-02-02	Quarter1	sweing	Monday	4	0.80	22.52	23122.0	
2015-02-02	Quarter1	sweing	Monday	9	0.75	29.12	8992.0	
2015-02-02	Quarter1	sweing	Monday	11	0.70	20.55	9792.0	
2015-02-02	Quarter1	sweing	Monday	6	0.70	18.79	2984.0	
2015-02-02	Quarter1	sweing	Monday	7	0.70	24.26	2698.0	

Overtime

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




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

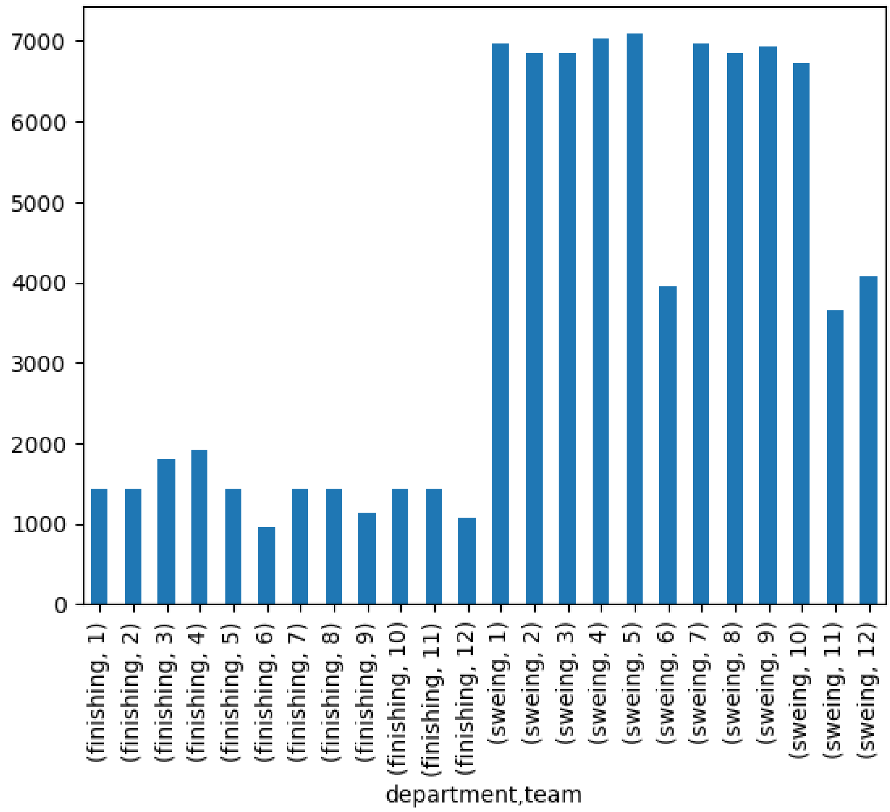



```
over_time_by_team_department = df.groupby(['department', 'team']).median()['over_time']

for team in range(1, 12):
    for department in ['sweing', 'finishing']:
        print('Median over_time of team {} {}s: {}'.format(team, department, over_time_by_team_department[departı
print('Median over_time of teams: {}'.format(df['over_time'].median()))
```

```
Median over_time of team 1 sweings: 6960.0
Median over_time of team 1 finishings: 1440.0
Median over_time of team 2 sweings: 6840.0
Median over_time of team 2 finishings: 1440.0
Median over_time of team 3 sweings: 6840.0
Median over_time of team 3 finishings: 1800.0
```

```
Median over_time of team 4 sweings: 7020.0
Median over_time of team 4 finishings: 1920.0
Median over_time of team 5 sweings: 7080.0
Median over_time of team 5 finishings: 1440.0
Median over_time of team 6 sweings: 3960.0
Median over_time of team 6 finishings: 960.0
Median over_time of team 7 sweings: 6960.0
Median over_time of team 7 finishings: 1440.0
Median over_time of team 8 sweings: 6840.0
Median over_time of team 8 finishings: 1440.0
Median over_time of team 9 sweings: 6930.0
Median over_time of team 9 finishings: 1140.0
Median over_time of team 10 sweings: 6720.0
Median over_time of team 10 finishings: 1440.0
Median over_time of team 11 sweings: 3660.0
Median over_time of team 11 finishings: 1440.0
Median over_time of teams: 3960.0
```

```
over_time_by_team_department.plot.bar()
plt.show()
```



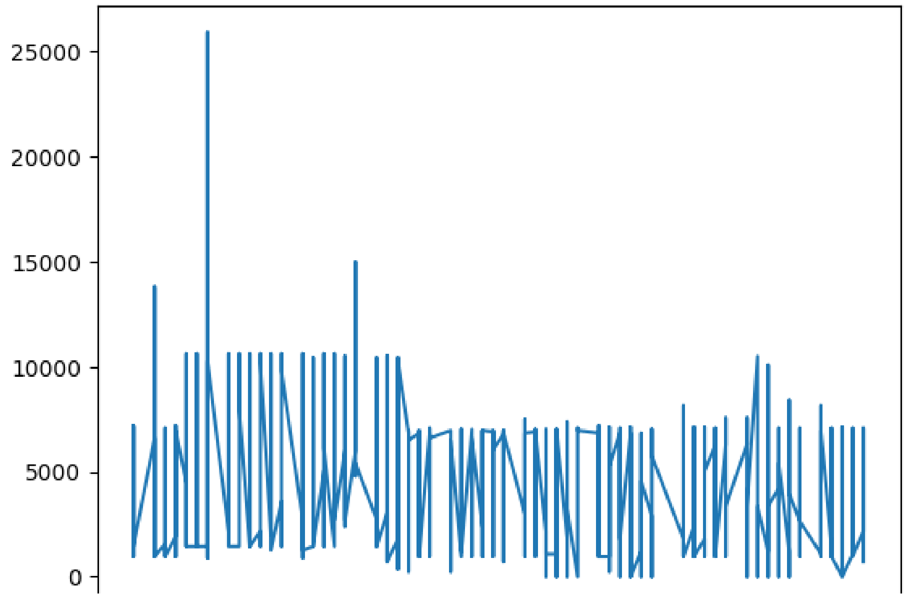

```
over_time_by_team_department.head()
```

```
department  team
finishing   1       1440.0
            2       1440.0
            3       1800.0
            4       1920.0
            5       1440.0
Name: over_time, dtype: float64
```

Finishing department has relatively lower over_time values regarding sweing department. In sweing department team6, team11 and team12 have the lowest over_time values.

```
pyplot.plot(df.index,df.over_time)
```

```
[<matplotlib.lines.Line2D at 0x7f11af96d7c0>]
```




```
df[df['over_time']>12000].shape
```

```
(8, 14)
```

```
df[df['over_time']>12000]
```

	quarter	department	day	team	targeted_productivity	smv	wip	over_
date								
2015-01-03	Quarter1	sweing	Saturday	4	0.70	23.69	544.0	
2015-01-08	Quarter2	finishing	Thursday	4	0.80	3.94	NaN	
2015-01-08	Quarter2	sweing	Thursday	12	0.80	11.61	548.0	
2015-01-08	Quarter2	sweing	Thursday	6	0.80	11.41	411.0	
2015-01-08	Quarter2	sweing	Thursday	11	0.35	12.52	287.0	
2015-01-22	Quarter4	sweing	Thursday	1	0.70	22.94	1384.0	
2015-01-22	Quarter4	finishing	Thursday	1	0.70	3.94	NaN	
2015-01-22	Quarter4	finishing	Thursday	2	0.70	3.94	NaN	

No significant relationship was found when peak values of over_time were observed with respect to time

Incentive

```
pyplot.plot(df.index,df.incentive)
```

```
[<matplotlib.lines.Line2D at 0x7f11af8d8880>]
```


```
df[df['incentive']>150].shape
```

```
(10, 14)
```

```
df[df['incentive']>150]
```

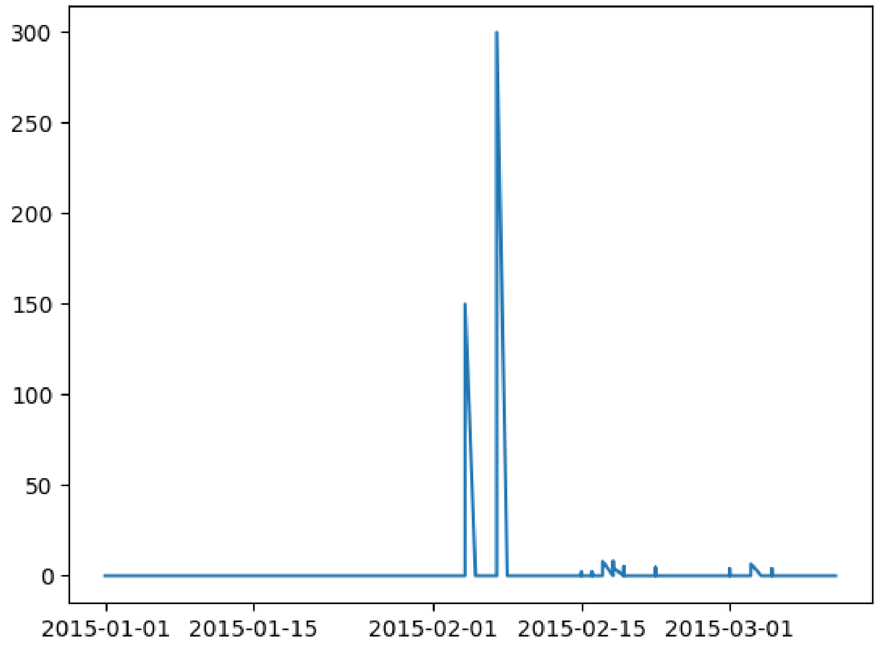
date	quarter	department	day	team	targeted_productivity	smv	wip	over_time
2015-03-09	Quarter2	finishing	Monday	11	0.80	2.90	NaN	(
2015-03-09	Quarter2	finishing	Monday	12	0.80	4.60	NaN	(
2015-03-09	Quarter2	finishing	Monday	5	0.60	3.94	NaN	(
2015-03-09	Quarter2	finishing	Monday	9	0.75	2.90	NaN	(
2015-03-09	Quarter2	finishing	Monday	3	0.80	4.60	NaN	(
2015-03-09	Quarter2	finishing	Monday	4	0.75	3.94	NaN	(
2015-03-09	Quarter2	finishing	Monday	1	0.75	3.94	NaN	(
2015-03-09	Quarter2	finishing	Monday	2	0.70	3.90	NaN	(
2015-03-09	Quarter2	finishing	Monday	10	0.70	2.90	NaN	(
2015-03-09	Quarter2	finishing	Monday	8	0.65	3.90	NaN	(

All of the highest incentive values belong to the finishing department on March 9, Quarter2.

Idle time

```
pyplot.plot(df.index,df.idle_time)
```

```
[<matplotlib.lines.Line2D at 0x7f11af852ac0>]
```




```
df[df['idle_time']>20].shape
```

```
(4, 14)
```

```
df[df['idle_time']>20]
```

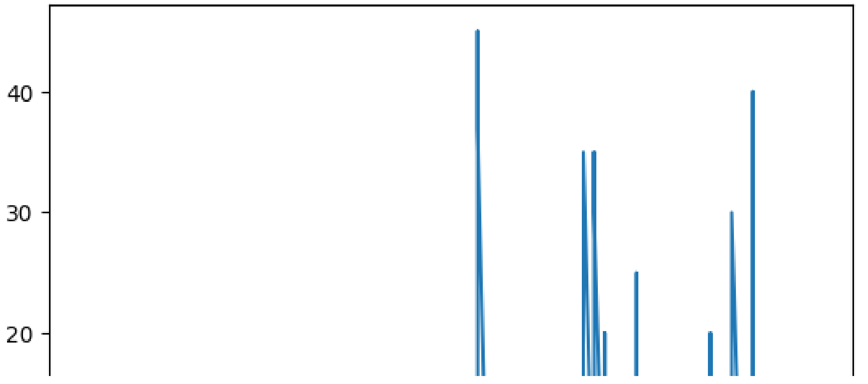
date	quarter	department	day	team	targeted_productivity	smv	wip	ovei
2015-02-04	Quarter1	sweing	Wednesday	5	0.65	30.10	326.0	
2015-02-04	Quarter1	sweing	Wednesday	4	0.35	30.10	287.0	
2015-02-07	Quarter1	sweing	Saturday	7	0.70	24.26	658.0	
2015-02-07	Quarter1	sweing	Saturday	8	0.70	24.26	652.0	

All of the highest idle men belong to the sweing department on feb 4 and 7,quarter1

Idle men

```
pyplot.plot(df.index,df.idle_men)
```

```
[<matplotlib.lines.Line2D at 0x7f11af759fa0>]
```




```
df[df['idle_men']>0].shape
```

```
(18, 14)
```

```
df[df['idle_men']>0]
```

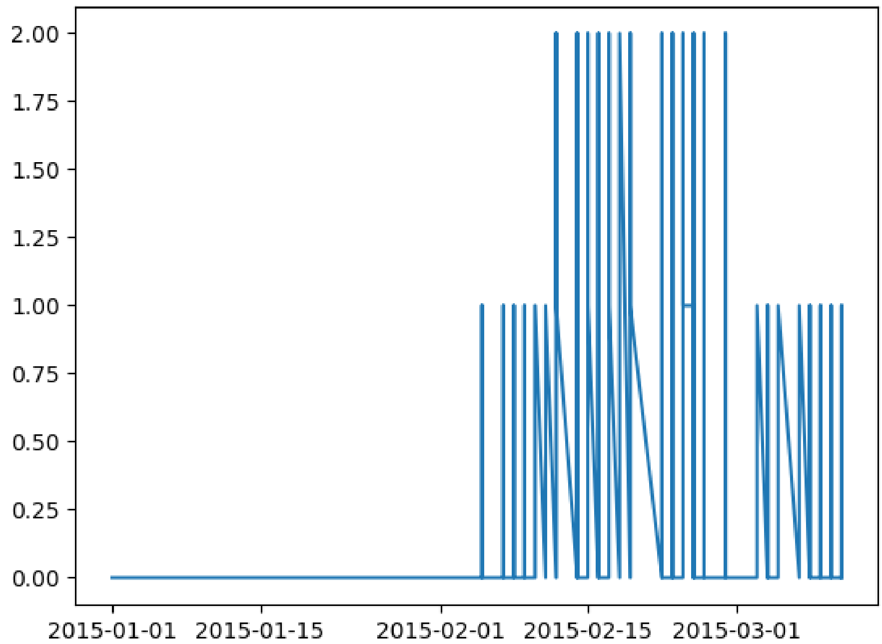
date	quarter	department	day	team	targeted_productivity	smv	wip
2015-02-04	Quarter1	sweing	Wednesday	5	0.65	30.10	326.0
2015-	Quarter1	sweing	Wednesday	4	0.35	30.10	287.0

All of the idle men department values belong to sweing group.

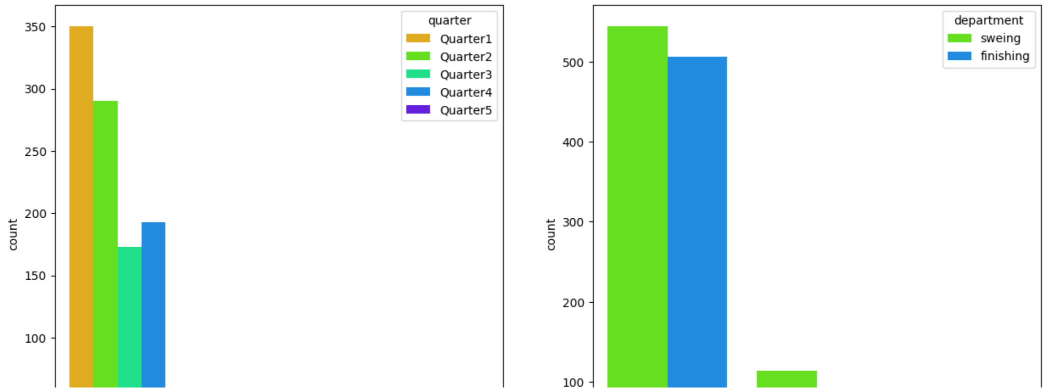
Number of style change

```
pyplot.plot(df.index,df.no_of_style_change)
```

```
[<matplotlib.lines.Line2D at 0x7f11af717e80>]
```



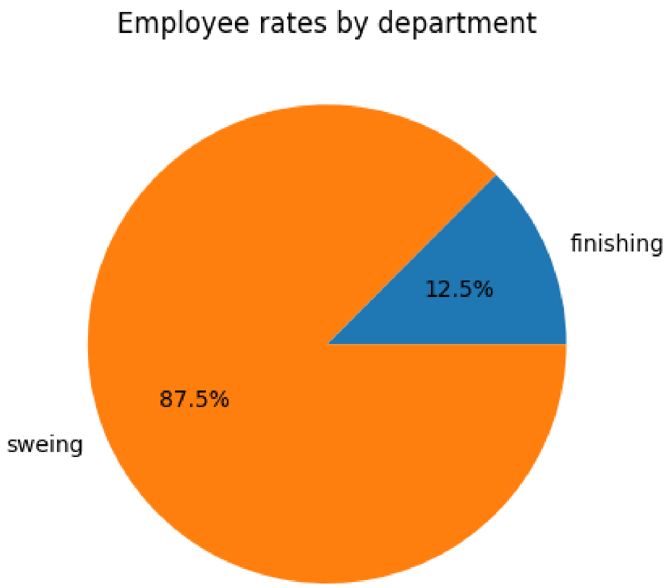

```
palette = 'gist_rainbow'
fig, axs = plt.subplots(ncols=2, figsize=(15, 7))

sns.countplot(x='no_of_style_change', hue='quarter', data=df, ax=axs[0], palette=palette)
axs[0].set_xlabel('no_of_style_change')

sns.countplot(x='no_of_style_change', hue='department', data=df, ax=axs[1], palette=palette)
axs[1].set_xlabel('no_of_style_change')

plt.show()
```



There is no no_of_style_change in Quarter5 and all of changes occured in the sweing department

Number of workers

```
data = df.groupby(['department']).no_of_workers.sum()
data.plot.pie(title="Employee rates by department",autopct='%1.1f%%')
plt.ylabel(None)
plt.show()
```

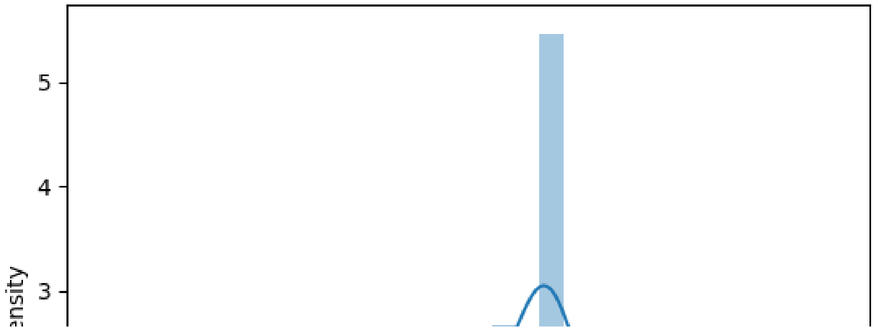


Employee rates in sweing and finishing departments are respectively 87.5 and 12.5

Actual Productivity

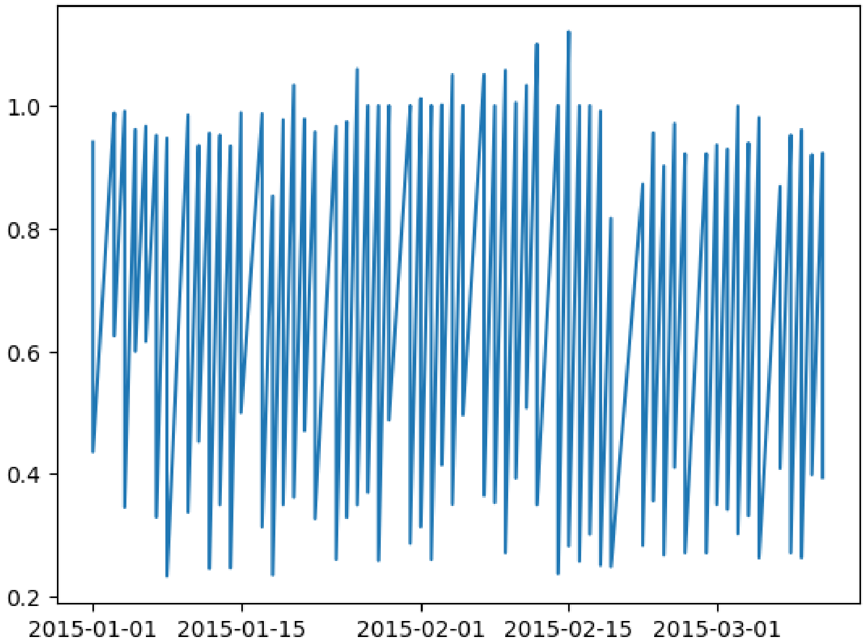
```
sns.distplot(df.actual_productivity)
```

```
<ipython-input-99-b069244992ad>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.actual_productivity)
<Axes: xlabel='actual_productivity', ylabel='Density'>
```



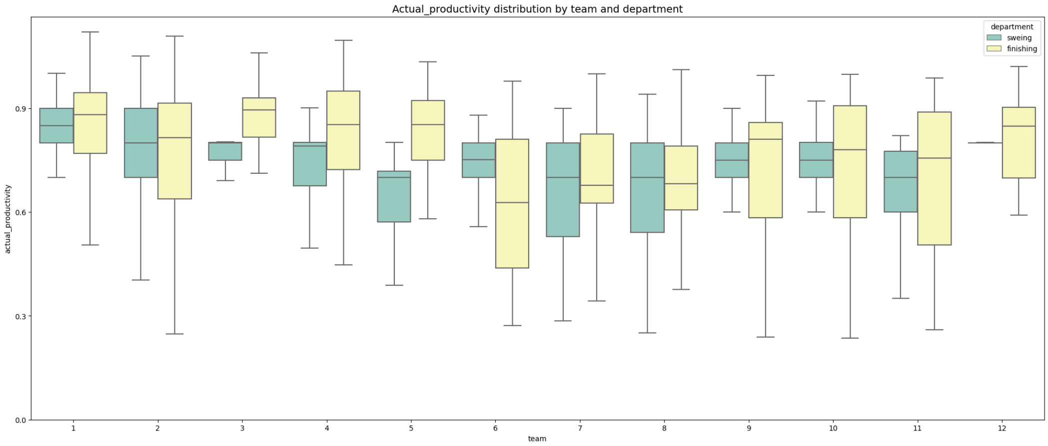

```
pyplot.plot(df.index,df.actual_productivity)
```

```
[<matplotlib.lines.Line2D at 0x7f11af797280>]
```



There is no an obvious pattern with respect to time in actual_productivity

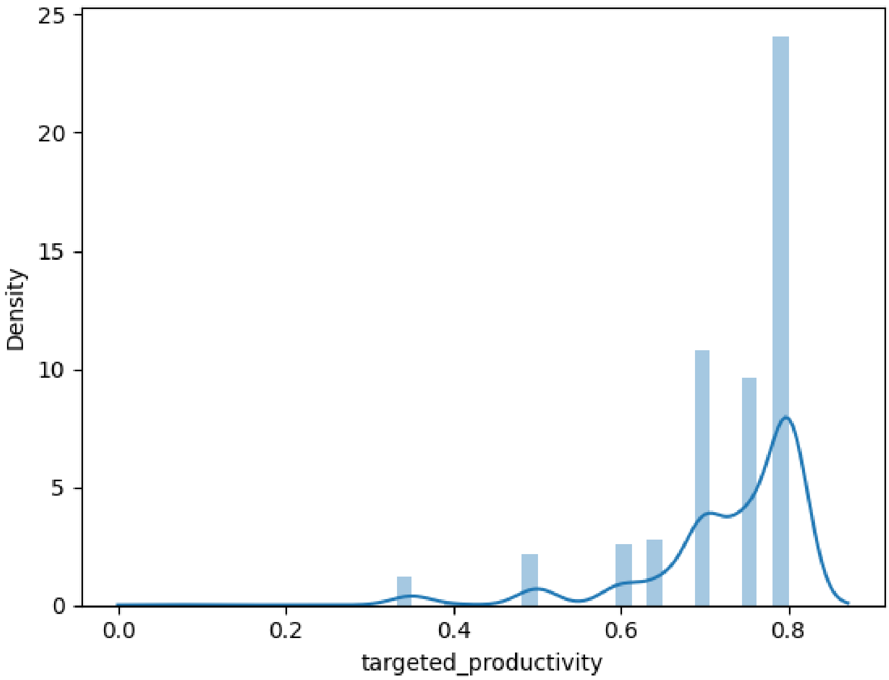
```
plt.figure(figsize=(25, 10))
palette = "Set3"

sns.boxplot(x = 'team', y = 'actual_productivity', data = df,
     palette = palette,hue='department',fliersize = 0)
plt.yticks(np.arange(0,1.2,step=0.3))
plt.title('Actual_productivity distribution by team and department',fontsize= 14)
plt.show()
```



Target Productivity

```
sns.distplot(df['targeted_productivity'])
```

```
<ipython-input-102-6b852d0a2865>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['targeted_productivity'])
<Axes: xlabel='targeted_productivity', ylabel='Density'>
```



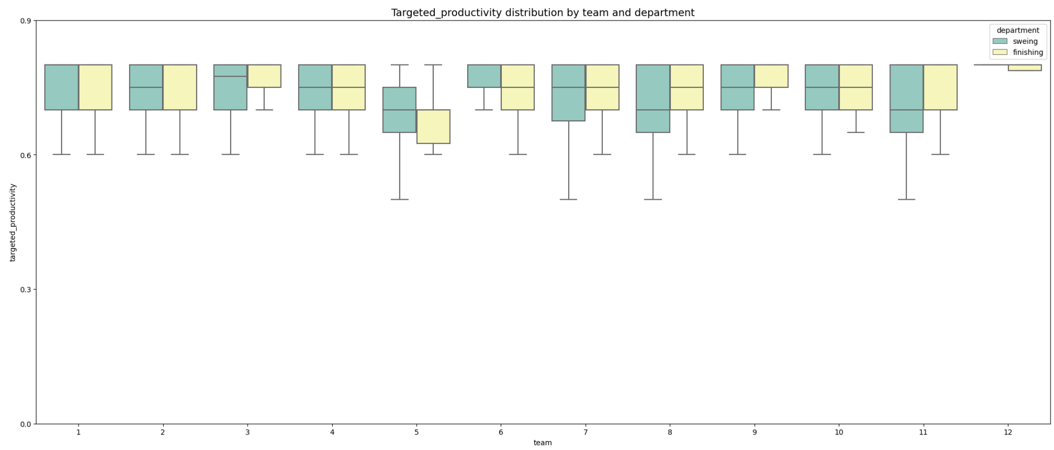

```
df.targeted_productivity.value_counts()
```

```
0.80    540
0.70    242
0.75    216
0.65     63
0.60     57
```

```
0.50     49
0.35     27
0.40      2
0.07      1
Name: targeted_productivity, dtype: int64
```

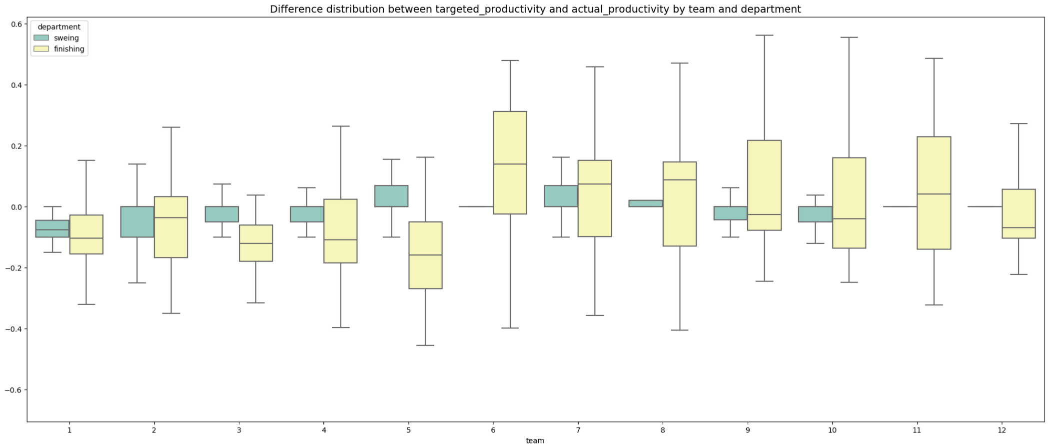
When we checked the dates it can be observed that there is a repeated pattern for all quarters with time except Quarter5. We need to look into Quarter5 deeply. There should be a reason for that exception

```
plt.figure(figsize=(25, 10))
palette = "Set3"

sns.boxplot(x = 'team', y = 'targeted_productivity', data = df,
     palette = palette,hue='department',fliersize = 0)
plt.yticks(np.arange(0,1.2,step=0.3))
plt.title('Targeted_productivity distribution by team and department',fontsize= 14)
plt.show()
```



Actual vs Targeted production

```
plt.figure(figsize=(25, 10))
palette = "Set3"

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'
plt.show()
```



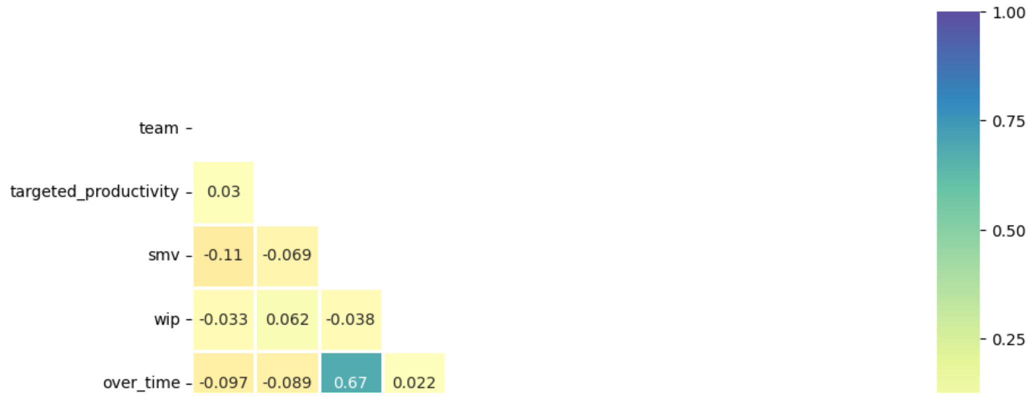
There are both negative and positive variations from targeted_productivity on team and department basis.

Exploratory data analysis

Correlation heatmap

```
corr=df.corr()
mask=np.zeros(corr.shape,dtype=bool)
mask[np.triu_indices(len(mask))]=True


plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True,vmin=-1,vmax=1,cmap='Spectral',square=True,mask=mask,linecolor='white',linewidths=1)
```

```
<Axes: >
```



Highest positive correlation:

1.No_of workers and smv (0.91)

2.No_of workers and over_time (0.73)

3.Over_time and smv (0.67)

4.Idle_men and Idle_time (0.56)

Positive correlation:

1.No_of workers and no_of_style_change(0.33)

2.No_of_style_changehas and smv (0.32)

There isnt any obvious negative correlation between features

Filling missing values of wip

```
df['wip'].fillna(0,inplace=True)
```

```
df['wip'].isnull().sum()
```

```
0
```

One hot encoding

Some columns have identified that may be useful for predicting productivity range:

quarter

department

day

team

no_of_style_change

Before we build our model, we need to prepare these columns for machine learning.

```
def create_dummies(df,column_name):
    dummies = pd.get_dummies(df[column_name],prefix=column_name)
    df = pd.concat([df,dummies],axis=1)
    return df

df = create_dummies(df,"quarter")
df = create_dummies(df,"department")
df = create_dummies(df,"day")
df = create_dummies(df,"team")
```

```
df.columns
```

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

Label encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["no_of_style_change_enc"] = le.fit_transform(df["no_of_style_change"])
df.head()
```

date	quarter	department	day	team	targeted_productivity	smv	wip	over_
2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	
2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	0.0	
2015-01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	

5 rows × 40 columns

```
df.columns
```

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

Creating Target_Label for productivity

```
df['diff']=df.actual_productivity-df.targeted_productivity
df.columns
```

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

```
df['diff'].describe()
```

```
count    1197.000000
mean        0.005459
std         0.160082
min        -0.561958
25%        -0.009556
50%         0.000505
75%         0.099111
max         0.644375
Name: diff, dtype: float64
```

```
df['Target_label']=np.nan
df.head()
df.loc[df['diff']<0,'Target_label'] = -1
df.loc[(df['diff']==0), 'Target_label'] = 0
df.loc[df['diff']>0, 'Target_label'] = 1
df.head()
```

date	quarter	department	day	team	targeted_productivity	smv	wip	over_
2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	
2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	0.0	
2015-01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	
2015-01-01	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	

5 rows × 42 columns

If the difference between actual_productivity and targeted_productivity is positive it means productivity is in the range of over_performed,

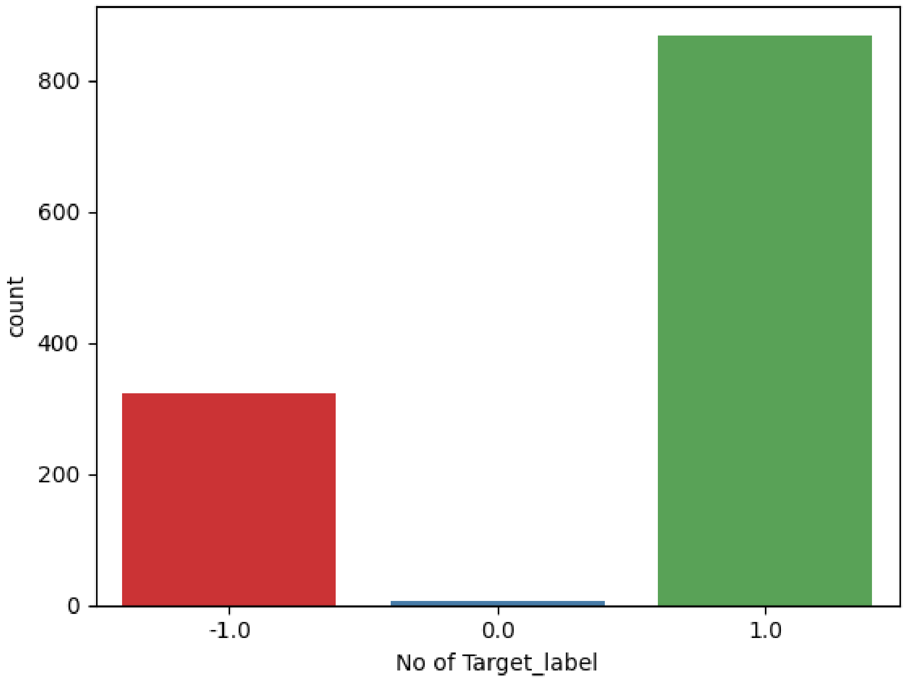
If the difference between actual_productivity and targeted_productivity is equal to 0 it meansproductivity is in the range of as expected,

If the difference between actual_productivity and targeted_productivity is negative it means productivity is in the range of under_performed

```
df[df['Target_label']==0]
```

	quarter	department	day	team	targeted_productivity	smv	wip	over_
date								
2015-01-10	Quarter2	sweing	Saturday	10	0.8	28.08	1082.0	
2015-01-11	Quarter2	sweing	Sunday	2	0.8	28.08	805.0	
2015-01-11	Quarter2	sweing	Sunday	10	0.8	28.08	762.0	
2015-01-12	Quarter2	sweing	Monday	2	0.8	28.08	737.0	

```
ax = sns.countplot(x = 'Target_label', data = df, palette='Set1')
plt.xlabel('No of Target_label')

plt.show()
```



As it can be seen from the graph above, there is imbalance so it is needed to be handled

```
df['Target_label'].value_counts()
```

```
 1.0    869
-1.0    322
 0.0      6
Name: Target_label, dtype: int64
```

From the value counts above, it can be seen that the dataset ist imbalanced due to the large number of unbalanced observations. In this case, a binary classification problem can be modelled that predicts whether productivity is in the range of over_performed or not.

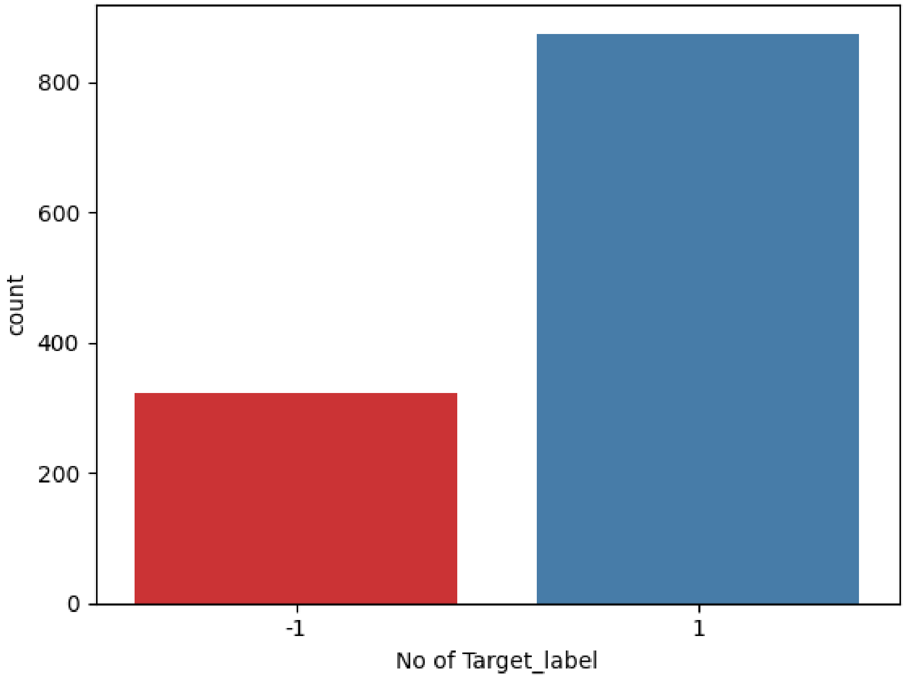
As part of our preprocessing,it is needed to turn the 3 class labels into 2 labels:

```
df['Target_label'] = [-1 if x==-1 else 1 for x in df['Target_label']]
```

```
df['Target_label'].value_counts()
```

```
 1    875
-1    322
Name: Target_label, dtype: int64
```

```
ax = sns.countplot(x = 'Target_label', data = df, palette='Set1')
plt.xlabel('No of Target_label')

plt.show()
```



Balancing data

```
!pip install imbalanced-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.9/dist-packages (0.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (from imbalanced-lea
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from imbalanced-lea
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.9/dist-packages (from imbalanc
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from imbalanced-lear
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from imbalar
```

```
import imblearn
print(imblearn.__version__)
```

```
0.10.1
```

```
df1=df.drop(['quarter', 'department', 'day', 'team'],axis=1)
```

```
from imblearn.over_sampling import SMOTE
X = df1.loc[:, df1.columns != 'Target_label']
y = df1.Target_label
smt = SMOTE()
X_smote, y_smote = smt.fit_resample(X, y)
plt.figure(figsize=(12, 8))
plt.title('Repartition after SMOTE')
plt.scatter(X_smote[y_smote==1], X_smote[y_smote==1], label='class 1')
plt.scatter(X_smote[y_smote==-1], X_smote[y_smote==-1], label='class -1')
plt.legend()
plt.grid(False)
```

```
plt.show()
```

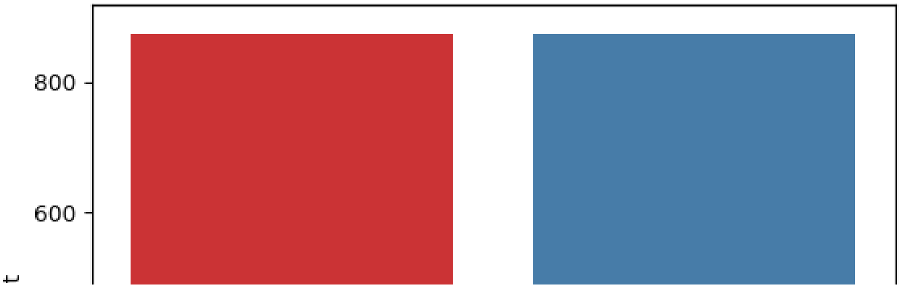

```
X_smote.shape, y_smote.shape
```

```
((1750, 37), (1750,))
```

```
df = pd.concat([pd.DataFrame(X_smote), pd.DataFrame(y_smote)], axis=1)
df.shape
```

```
(1750, 38)
```

```
ax = sns.countplot(x = 'Target_label', data = df, palette='Set1')
plt.xlabel('No of Target_label')

plt.show()
```



Splitting train and test data

```
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_enc']

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

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2,random_state=0)
```

```
X_train.shape
```

```
(1400, 33)
```

```
y_train.shape
```

```
(1400,)
```

```
X_test.shape
```

```
(350, 33)
```

```
y_test.shape
```

```
(350,)
```

Scaling

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Build Models

Logistic Regression

Decision Tree Classifiers

Random Forests

Support Vector Machines

K-Nearest Neighbors

Gaussian Naive Bayes

LinearDiscriminantAnalysis

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import precision_score,recall_score,accuracy_score
```

Machine learning classifier training and validating

```
df_perf_metrics = pd.DataFrame(columns=[
    'Model', 'Accuracy_Training_Set', 'Accuracy_Test_Set', 'Precision',
    'Recall', 'f1_score'
])
models_trained_list = []

def get_perf_metrics(model, i):
    # model name
    model_name = type(model).__name__
    print("Training {} model...".format(model_name))
    # Fitting of model
    model.fit(X_train, y_train)
    print("Completed {} model training.".format(model_name))
    # Predictions
    y_pred = model.predict(X_test)
    # Add to ith row of dataframe - metrics
    df_perf_metrics.loc[i] = [
        model_name,
        model.score(X_train, y_train),
        model.score(X_test, y_test),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred),
    ]
    print("Completed {} model's performance assessment.".format(model_name))
```

```
models_list = [LogisticRegression(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               SVC(),
               KNeighborsClassifier(),
               GaussianNB(),LinearDiscriminantAnalysis()
               ]
```

```
from sklearn.metrics import r2_score,f1_score
for n, model in enumerate(models_list):
    get_perf_metrics(model, n)
```

```
Training LogisticRegression model...
Completed LogisticRegression model training.
Completed LogisticRegression model's performance assessment.
Training DecisionTreeClassifier model...
Completed DecisionTreeClassifier model training.
Completed DecisionTreeClassifier model's performance assessment.
Training RandomForestClassifier model...
Completed RandomForestClassifier model training.
Completed RandomForestClassifier model's performance assessment.
```

```
Training SVC model...
Completed SVC model training.
Completed SVC model's performance assessment.
Training KNeighborsClassifier model...
Completed KNeighborsClassifier model training.
Completed KNeighborsClassifier model's performance assessment.
Training GaussianNB model...
Completed GaussianNB model training.
Completed GaussianNB model's performance assessment.
Training LinearDiscriminantAnalysis model...
Completed LinearDiscriminantAnalysis model training.
Completed LinearDiscriminantAnalysis model's performance assessment.
```

```
df_perf_metrics
```

	Model	Accuracy_Training_Set	Accuracy_Test_Set	Precision	Rec
0	LogisticRegression	0.658571	0.668571	0.711409	0.592
1	DecisionTreeClassifier	0.998571	0.785714	0.813253	0.754
2	RandomForestClassifier	0.998571	0.865714	0.875000	0.860
3	SVC	0.657857	0.662857	0.707483	0.581
4	KNeighborsClassifier	0.842143	0.780000	0.814815	0.737
5	GaussianNB	0.641429	0.617143	0.695652	0.446
6	LinearDiscriminantAnalysis	0.791429	0.808571	0.780000	0.871

RandomForestClassifier model

```
rfc = RandomForestClassifier()
parameters = {
    "n_estimators":[5,10,50,100,250],
    "max_depth":[2,4,8,16,32,None]}
```

```
from sklearn.model_selection import GridSearchCV
cv = GridSearchCV(rfc,parameters,cv=5)
cv.fit(X_train,y_train.values.ravel())
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
```

```
display(cv)
```

```
Best parameters are: {'max_depth': None, 'n_estimators': 250}


0.717 + or -0.029 for the {'max_depth': 2, 'n_estimators': 5}
0.698 + or -0.02 for the {'max_depth': 2, 'n_estimators': 10}
0.685 + or -0.016 for the {'max_depth': 2, 'n_estimators': 50}
0.689 + or -0.035 for the {'max_depth': 2, 'n_estimators': 100}
0.686 + or -0.029 for the {'max_depth': 2, 'n_estimators': 250}
```

```
0.744 + or -0.026 for the {'max_depth': 4, 'n_estimators': 5}
0.757 + or -0.02 for the {'max_depth': 4, 'n_estimators': 10}
0.764 + or -0.023 for the {'max_depth': 4, 'n_estimators': 50}
0.758 + or -0.025 for the {'max_depth': 4, 'n_estimators': 100}
0.757 + or -0.029 for the {'max_depth': 4, 'n_estimators': 250}
0.803 + or -0.027 for the {'max_depth': 8, 'n_estimators': 5}
0.8 + or -0.016 for the {'max_depth': 8, 'n_estimators': 10}
0.813 + or -0.01 for the {'max_depth': 8, 'n_estimators': 50}
0.816 + or -0.01 for the {'max_depth': 8, 'n_estimators': 100}
0.811 + or -0.008 for the {'max_depth': 8, 'n_estimators': 250}
0.851 + or -0.017 for the {'max_depth': 16, 'n_estimators': 5}
0.85 + or -0.016 for the {'max_depth': 16, 'n_estimators': 10}
0.864 + or -0.014 for the {'max_depth': 16, 'n_estimators': 50}
0.87 + or -0.018 for the {'max_depth': 16, 'n_estimators': 100}
0.874 + or -0.018 for the {'max_depth': 16, 'n_estimators': 250}
0.833 + or -0.004 for the {'max_depth': 32, 'n_estimators': 5}
0.856 + or -0.022 for the {'max_depth': 32, 'n_estimators': 10}
0.869 + or -0.015 for the {'max_depth': 32, 'n_estimators': 50}
0.874 + or -0.026 for the {'max_depth': 32, 'n_estimators': 100}
0.876 + or -0.019 for the {'max_depth': 32, 'n_estimators': 250}
0.845 + or -0.023 for the {'max_depth': None, 'n_estimators': 5}
0.854 + or -0.023 for the {'max_depth': None, 'n_estimators': 10}
0.873 + or -0.017 for the {'max_depth': None, 'n_estimators': 50}
0.875 + or -0.015 for the {'max_depth': None, 'n_estimators': 100}
0.878 + or -0.015 for the {'max_depth': None, 'n_estimators': 250}
```

```
model = cv.best_estimator_
y_pred = model.predict(X_test)
print('Accuracy: ', accuracy_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
print('Recall: ', recall_score(y_test, y_pred))
print('f1-score: ', f1_score(y_test, y_pred))
```

```
Accuracy:  0.8771428571428571
Precision:  0.8908045977011494
Recall:  0.8659217877094972
f1-score:  0.878186968838527
```

```
df1_perf_metrics = pd.DataFrame(columns=[
    'Model', 'Accuracy_Training_Set', 'Accuracy_Test_Set', 'Precision',
    'Recall', 'f1_score'
])

def get_perf_metrics_t(model):
    model = cv.best_estimator_
    model_name =RandomForestClassifier()

    print('Training RandomForestClassifier()')
    model.fit(X_train, y_train)
    print('Completed RandomForestClassifier()')
    y_pred = model.predict(X_test)

    df1_perf_metrics.loc[0] = [
        model_name,
        model.score(X_train, y_train),
        model.score(X_test, y_test),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred),
    ]

    print("Completed RandomForestClassifier() model's performance assessment.")

get_perf_metrics_t(model)
```

```
Training RandomForestClassifier()
Completed RandomForestClassifier()
Completed RandomForestClassifier() model's performance assessment.
```

```
df1_perf_metrics
```

	Model	Accuracy_Training_Set	Accuracy_Test_Set	Precision	Reca
0	RandomForestClassifier()	0.998571	0.868571	0.88	0.8603

✓ 0s completed at 1:45 PM