```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
```

```
# impord data
data =pd.read_csv("/content/garments_worker_productivity.csv")
# Looking at the dataset
data.head(5)
```

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

```
# describe data
data.describe()
```

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_cha
count	1197.000000	1197.000000	1197.000000	691.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000
mean	6.426901	0.729632	15.062172	1190.465991	4567.460317	38.210526	0.730159	0.369256	0.150
std	3.463963	0.097891	10.943219	1837.455001	3348.823563	160.182643	12.709757	3.268987	0.427
min	1.000000	0.070000	2.900000	7.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	3.000000	0.700000	3.940000	774.500000	1440.000000	0.000000	0.000000	0.000000	0.000
50%	6.000000	0.750000	15.260000	1039.000000	3960.000000	0.000000	0.000000	0.000000	0.000
75%	9.000000	0.800000	24.260000	1252.500000	6960.000000	50.000000	0.000000	0.000000	0.000
max	12.000000	0.800000	54.560000	23122.000000	25920.000000	3600.000000	300.000000	45.000000	2.000

```
# Looking at the features types
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
```

```
data.shape
```

```
data.shape
```

```
(1197, 15)
```

```
# Looking at the columns
data.columns
```

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

```
# Looking at the null values in each feature
data.isnull().sum()
```

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

```
# Percentage of null values in the dataset
data.isnull().sum()  / data.shape[0] * 100
```

```
date                      0.000000
quarter                   0.000000
department                0.000000
day                       0.000000
team                      0.000000
targeted_productivity     0.000000
smv                       0.000000
wip                      42.272348
over_time                 0.000000
incentive                 0.000000
idle_time                 0.000000
idle_men                  0.000000
no_of_style_change        0.000000
no_of_workers             0.000000
actual_productivity       0.000000
dtype: float64
```

```
# Separate categorical and numerical data for simplicity in analysis
cat = data.select_dtypes(include='object')
num = data.select_dtypes(exclude='object')
```

```
#showing the catgory columns
for col in cat.columns:
    print(f"{col}")
    print(cat[col].unique())
    print()
```

```
date
['1/1/2015' '1/3/2015' '1/4/2015' '1/5/2015' '1/6/2015' '1/7/2015'
 '1/8/2015' '1/10/2015' '1/11/2015' '1/12/2015' '1/13/2015' '1/14/2015'
 '1/15/2015' '1/17/2015' '1/18/2015' '1/19/2015' '1/20/2015' '1/21/2015'
 '1/22/2015' '1/24/2015' '1/25/2015' '1/26/2015' '1/27/2015' '1/28/2015'
 '1/29/2015' '1/31/2015' '2/1/2015' '2/2/2015' '2/3/2015' '2/4/2015'
 '2/5/2015' '2/7/2015' '2/8/2015' '2/9/2015' '2/10/2015' '2/11/2015'
 '2/12/2015' '2/14/2015' '2/15/2015' '2/16/2015' '2/17/2015' '2/18/2015'
 '2/19/2015' '2/22/2015' '2/23/2015' '2/24/2015' '2/25/2015' '2/26/2015'
 '2/28/2015' '3/1/2015' '3/2/2015' '3/3/2015' '3/4/2015' '3/5/2015'
 '3/7/2015' '3/8/2015' '3/9/2015' '3/10/2015' '3/11/2015']

quarter
['Quarter1' 'Quarter2' 'Quarter3' 'Quarter4' 'Quarter5']

department
['sweing' 'finishing ' 'finishing']
```

```
day
['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
```

```
# Department Attribute has space present in one of the values which needs to modification
print('Unique Values in Department before cleaning:')
print(data.department.unique())
print()
data['department'] = data.department.str.strip()
print('Unique Values in Department afer cleaning:')
print(data.department.unique())
```

```
Unique Values in Department before cleaning:
['sweing' 'finishing ' 'finishing']

Unique Values in Department afer cleaning:
['sweing' 'finishing']
```

```
#showing the numerical columns
for n in num.columns:
    print(n)
```

```
team
targeted_productivity
smv
wip
over_time
incentive
idle_time
idle_men
no_of_style_change
no_of_workers
actual_productivity
```

```
# checking the categorical feature for make sure all name and group is correct
cat.quarter.value_counts()
```

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

```
cat.department.value_counts()
```

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

```
cat.day.value_counts()
```

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

```
import matplotlib.pyplot as plt
plt.plot(data['date'], data['quarter'])
```

```
[<matplotlib.lines.Line2D at 0x7ff0dbdecfa0>]
```


```
data['date'] = pd.to_datetime(data['date'])
#Setting date column as index
data.set_index('date', drop = False, inplace=True)
data.head()
```

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_
date													
2015-01-01	2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	
2015-01-01	2015-01-01	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	
2015-01-01	2015-01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	
2015-01-01	2015-01-01	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	
2015-01-01	2015-01-01	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	

when we checked all feature for null value ,the result shows that "wip" feature has 42% null value and we try to replace the missing data with use interpolate function.

```
data['wip'].interpolate(method='time',inplace=True)
data.wip
```

```
date
2015-01-01    1108.0
2015-01-01     795.0
2015-01-01     968.0
2015-01-01     968.0
2015-01-01    1170.0
               ...
2015-03-11     935.0
2015-03-11     935.0
2015-03-11     935.0
2015-03-11     935.0
2015-03-11     935.0
Name: wip, Length: 1197, dtype: float64
```

One of our targets is to dedicate the time feature is an essential and effective factor or not. So, we can first clarify the month of the date and add this feature to our dataset.

```
# Clarify month of date and add to featurs
data['month'] = data['date'].dt.month_name()

data.head()
```

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_
date													
2015-01-01	2015-01-01	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	
2015-	2015-	Quarter1	finishing	Thursday	1	0.75	3.94	795.0	960	0	0.0	0	
01-01	01-01	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	
01-01	01-01												

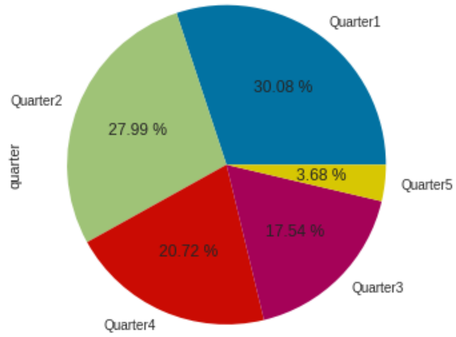
As you can see we add a one feature with month name

The data is divided into categorical features, and we will look at how they are made up

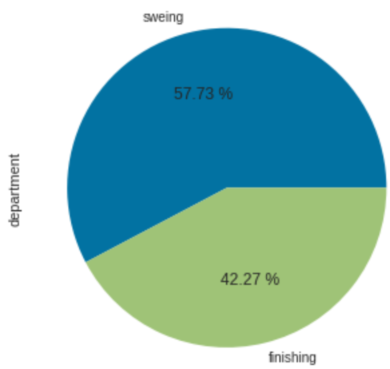
```
# Add the month and no_of_style_change features to categorical
cat = [ 'quarter', 'department', 'day', 'team', 'no_of_style_change','month']


# make  pie charts with "For" structure for categorical feature
for i in range(len(cat)):
    print(cat[i])
    data[cat[i]].value_counts().plot.pie(autopct='%.2f %%')
    plt.show()
    print()
```

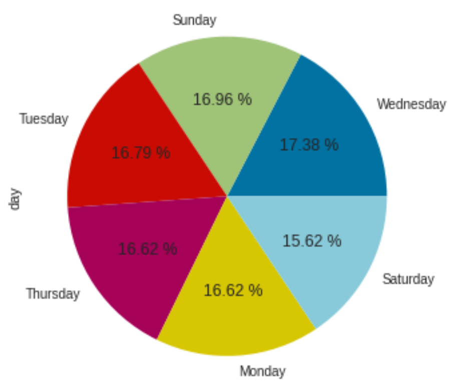
```
quarter
```




```
department
```

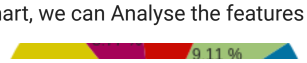



```
day
```



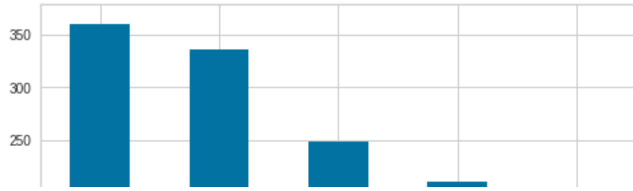

```
team
```

With a Bar chart, we can Analyse the features individually



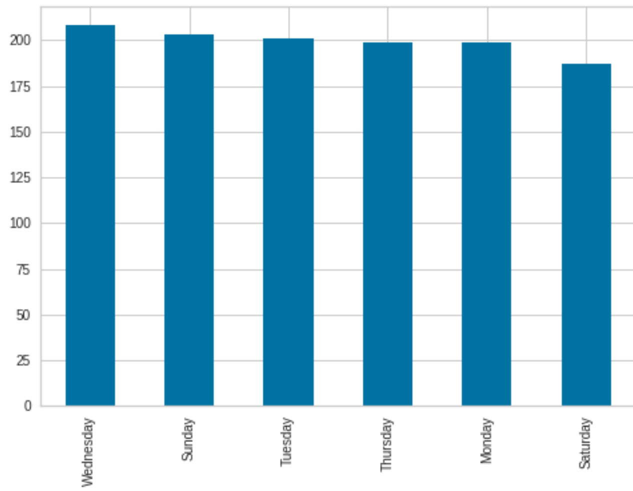

```
# Bar charts
data.quarter.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db8d2a00>
```



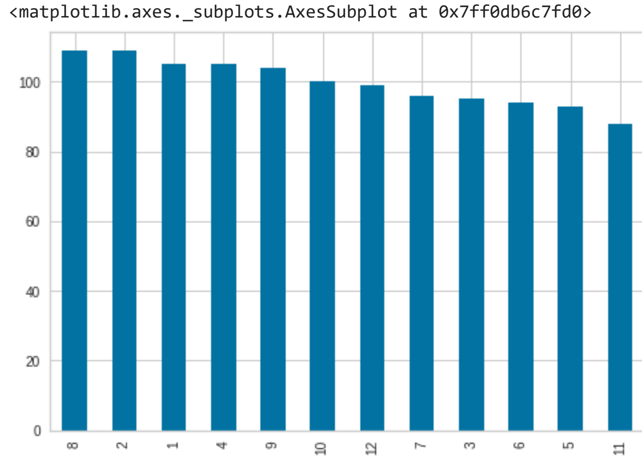

```
data.day.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db76dc40>
```




```
data.team.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db6c7fd0>
```

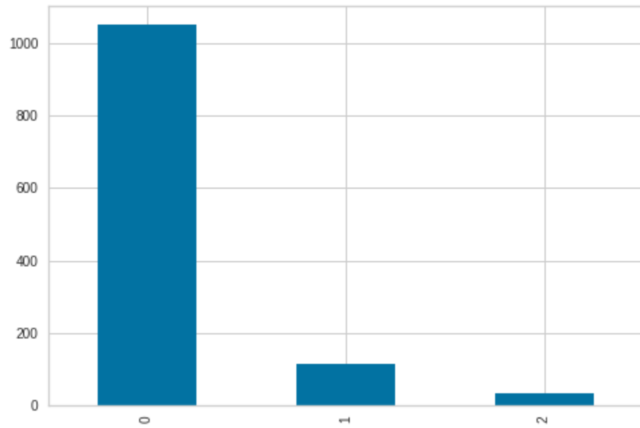



```
data.department.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db6f4f40>
```

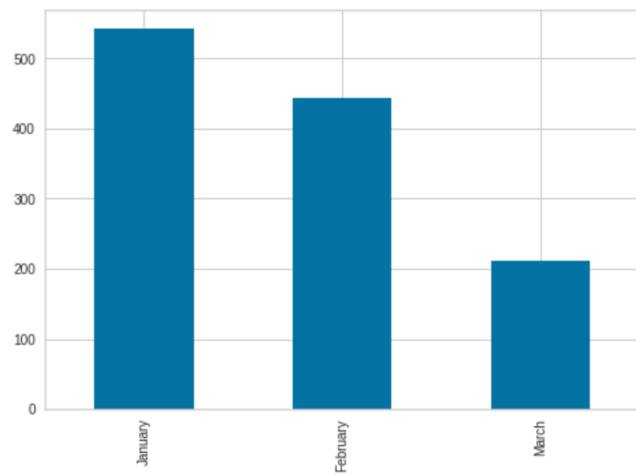
```
data.no_of_style_change.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db6ac700>
```




```
data.month.value_counts().plot(kind='bar')
```

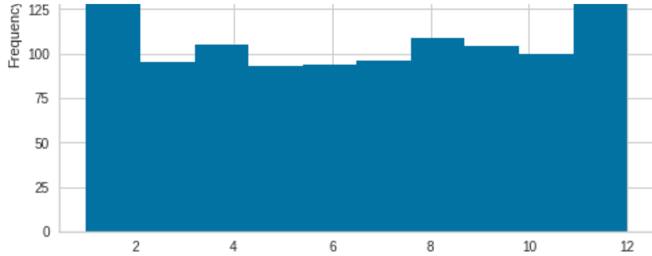
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0db5e11c0>
```



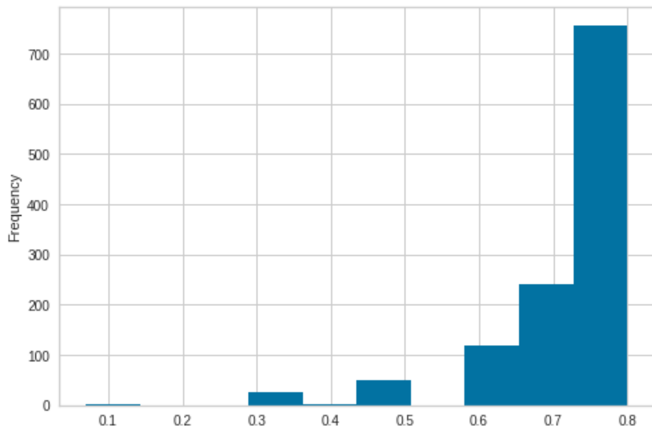

```
num.columns
```

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

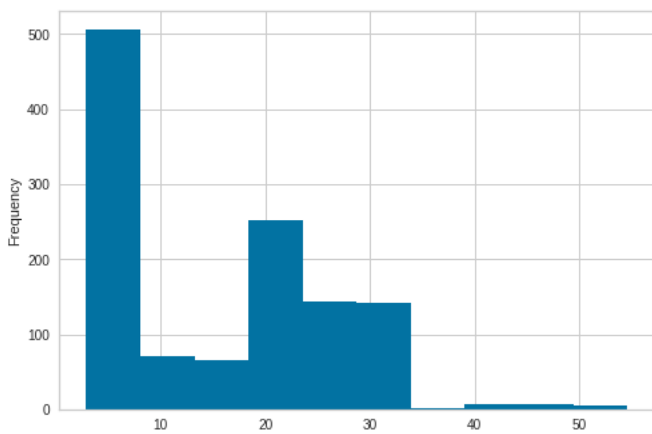
```
# Creat histogram with "FOR" for numerical feature
for i in range(len(num.columns)):
    print(num.columns[i])
    num.iloc[:, i].plot(kind='hist')
    plt.show()
    print()
```



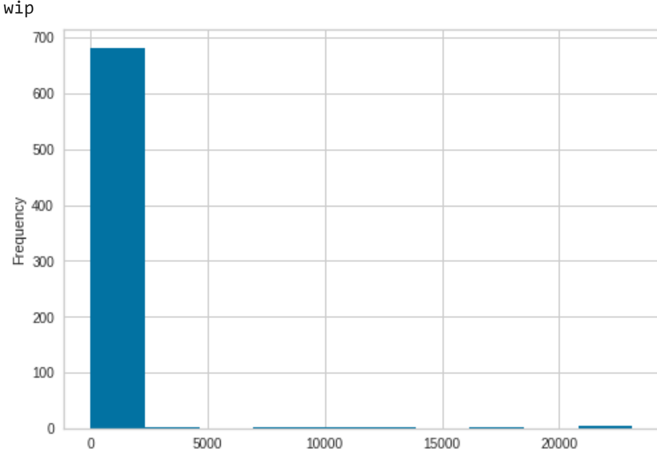
targeted_productivity



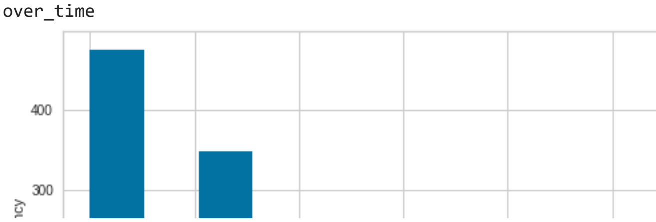
smv



wip

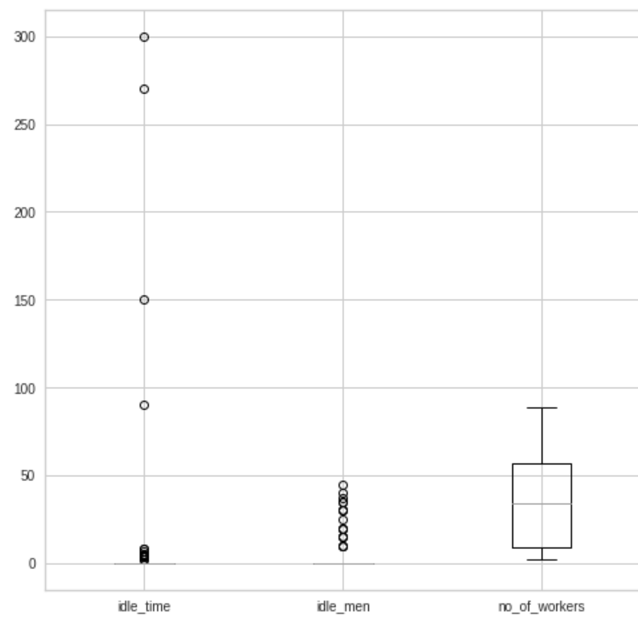


over_time



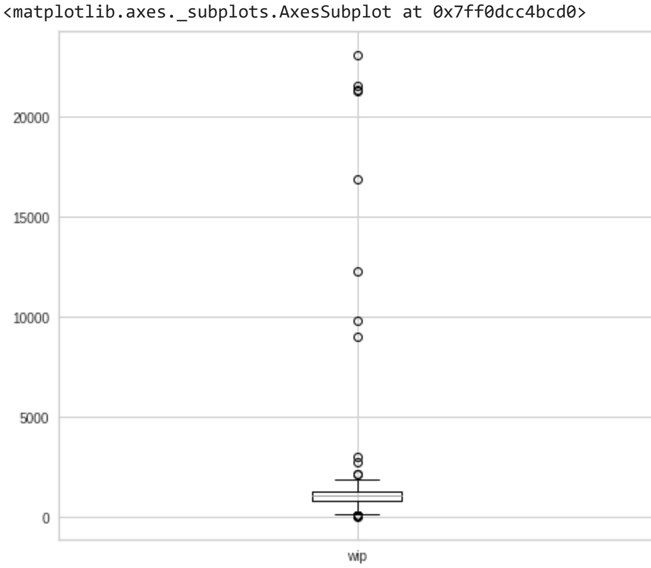
```
# Creat boxplots for numerical feature
num.boxplot(column = [ 'idle_time', 'idle_men', 'no_of_workers'],figsize=(8,8) )
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff0dbb4fee0>



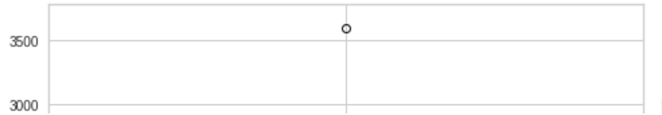

```
num.boxplot(column=['wip'],figsize=(8,7))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff0dcc4bcd0>



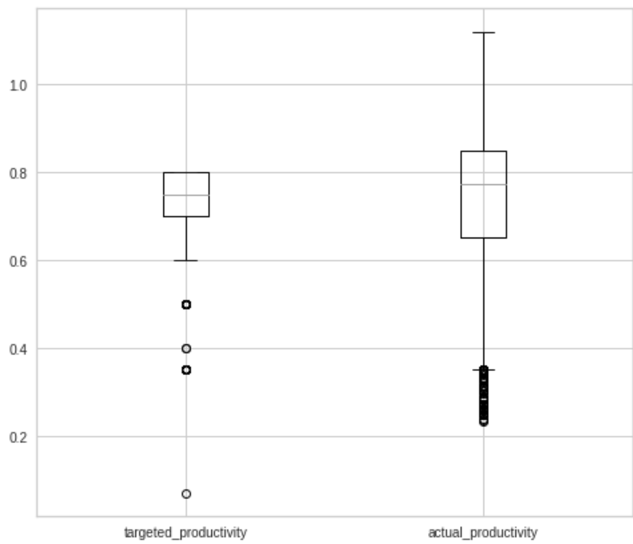
```
num.boxplot(column=['incentive'],figsize=(8,7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0dbeea6a0>
```

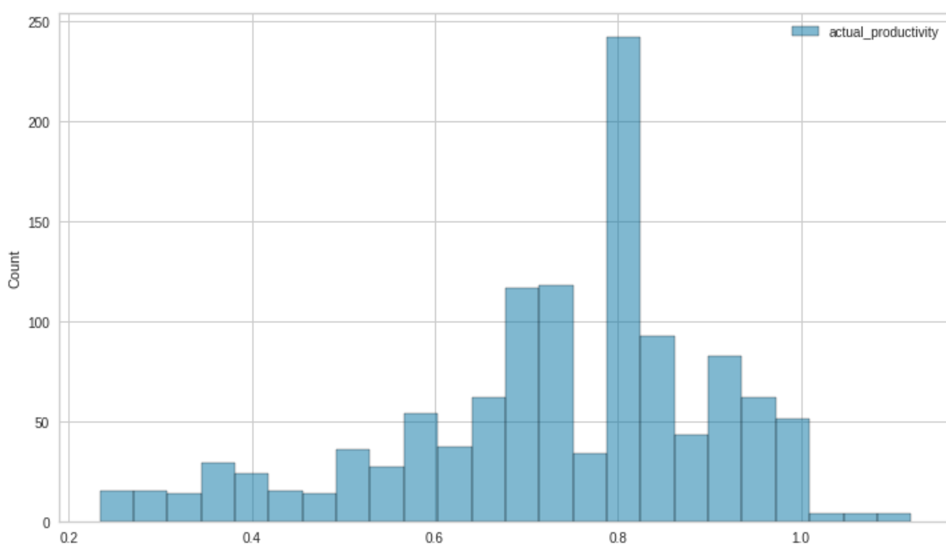



```
num.boxplot(column = [ 'targeted_productivity', 'actual_productivity'],figsize=(8,7))
```

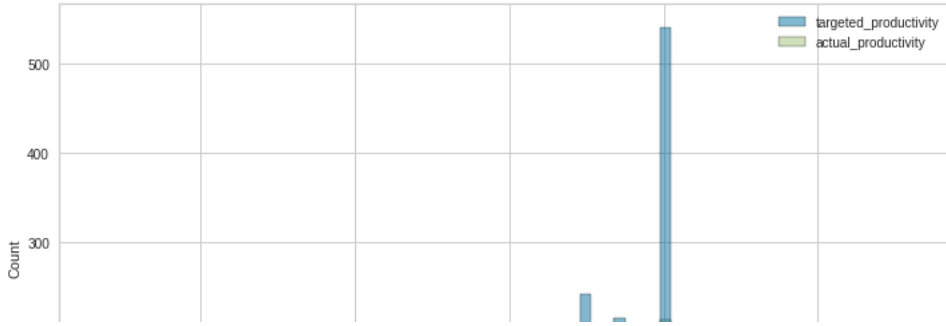
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0dc0dd070>
```



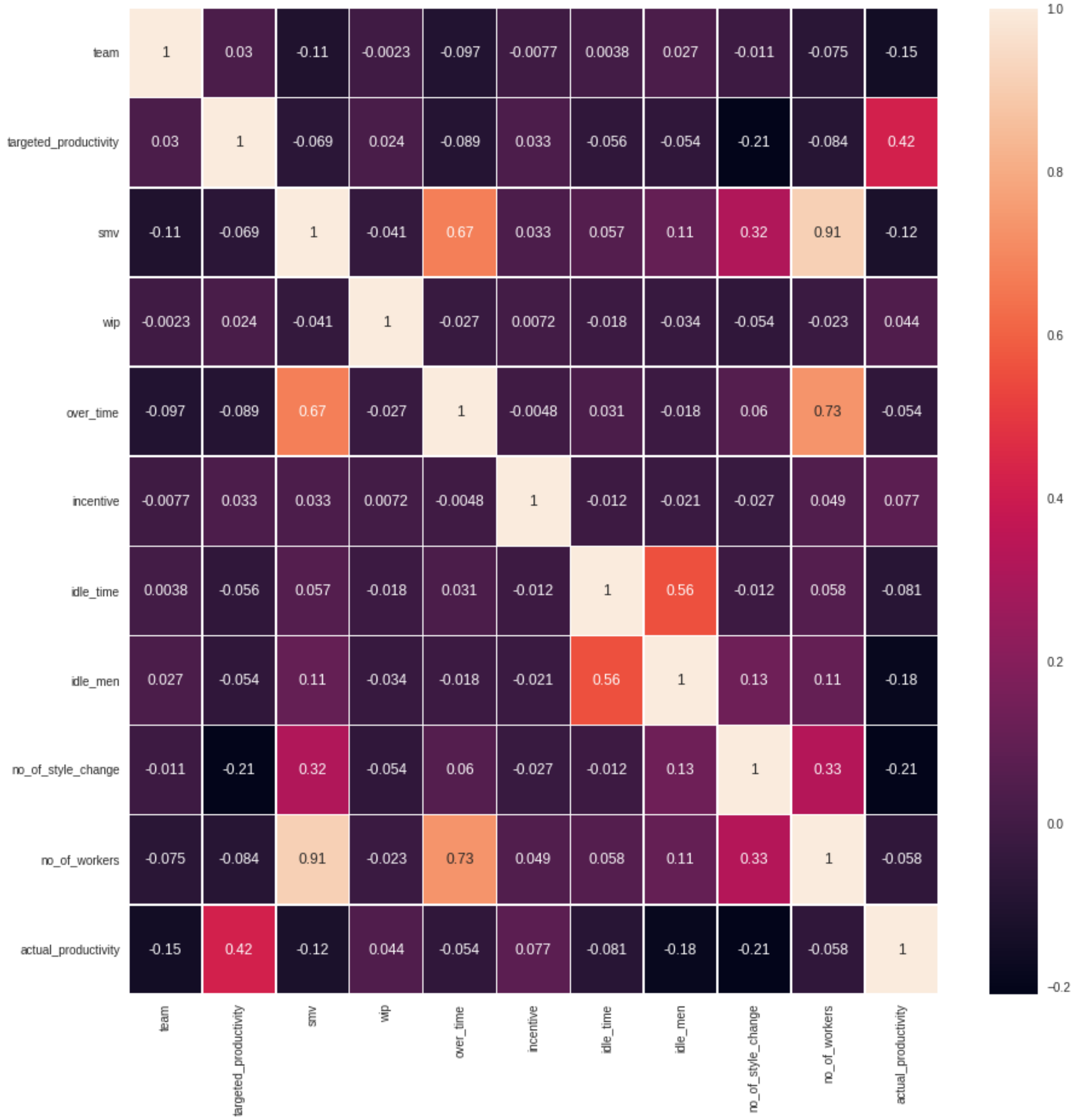
```
fig_dims = (12,7)
fig, ax = plt.subplots(figsize=fig_dims)
sns.histplot(data=data[['actual_productivity']],ax=ax)
plt.show()
```




```
fig_dims = (12,7)
fig, ax = plt.subplots(figsize=fig_dims)
sns.histplot(data=data[['targeted_productivity', 'actual_productivity']],ax=ax)
plt.show()
```




```
corrMatrix = data.corr()
fig, ax = plt.subplots(figsize=(15,15)) # Sample figsize in inches
sns.heatmap(corrMatrix, annot=True, linewidths=.5, ax=ax)
plt.show()
```

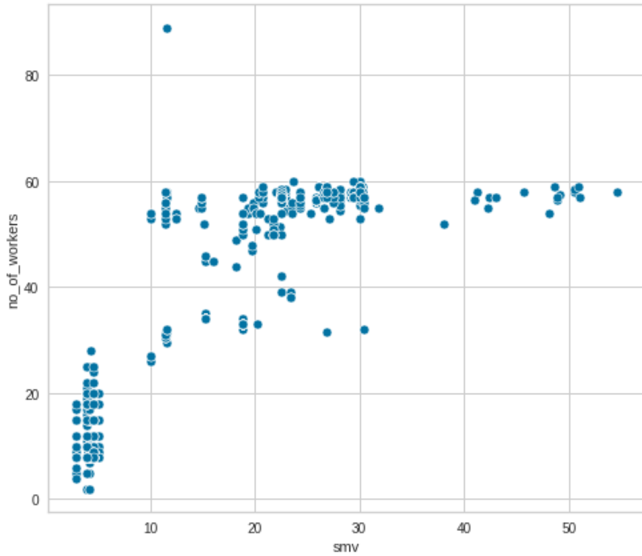


	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actual_productivity
team	1	0.03	-0.11	-0.0023	-0.097	-0.0077	0.0038	0.027	-0.011	-0.075	-0.15
targeted_productivity	0.03	1	-0.069	0.024	-0.089	0.033	-0.056	-0.054	-0.21	-0.084	0.42
smv	-0.11	-0.069	1	-0.041	0.67	0.033	0.057	0.11	0.32	0.91	-0.12
wip	-0.0023	0.024	-0.041	1	-0.027	0.0072	-0.018	-0.034	-0.054	-0.023	0.044
over_time	-0.097	-0.089	0.67	-0.027	1	-0.0048	0.031	-0.018	0.06	0.73	-0.054
incentive	-0.0077	0.033	0.033	0.0072	-0.0048	1	-0.012	-0.021	-0.027	0.049	0.077
idle_time	0.0038	-0.056	0.057	-0.018	0.031	-0.012	1	0.56	-0.012	0.058	-0.081
idle_men	0.027	-0.054	0.11	-0.034	-0.018	-0.021	0.56	1	0.13	0.11	-0.18
no_of_style_change	-0.011	-0.21	0.32	-0.054	0.06	-0.027	-0.012	0.13	1	0.33	-0.21
no_of_workers	-0.075	-0.084	0.91	-0.023	0.73	0.049	0.058	0.11	0.33	1	-0.058
actual_productivity	-0.15	0.42	-0.12	0.044	-0.054	0.077	-0.081	-0.18	-0.21	-0.058	1

- we can see that there is a very high correlation between SMV and no_of_workers.
- over_time and no_of)workers are correlated

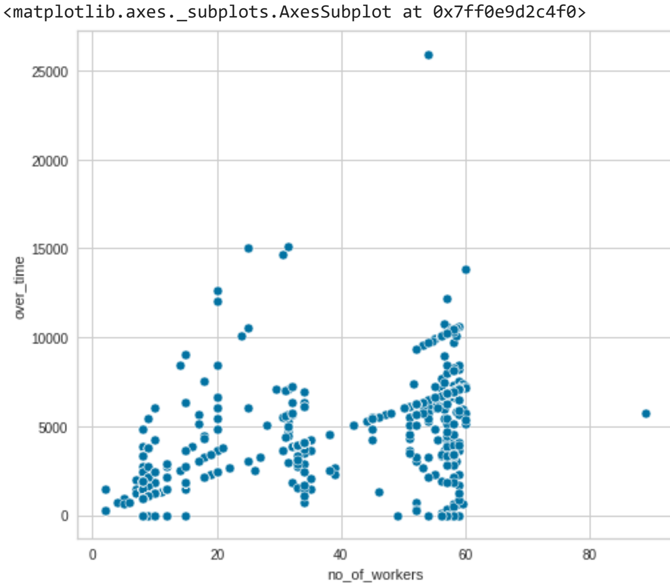
```
fig, ax = plt.subplots(figsize = (8,7))
sns.scatterplot(x=data['smv'], y=data['no_of_workers'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0dc7e9310>
```




```
fig, ax = plt.subplots(figsize = (8,7))
sns.scatterplot(x=data['no_of_workers'], y=data['over_time'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0e9d2c4f0>
```



Let us look at few features which have numeric values but are in fact categorical

```
#check few feature which in fact they are categorical
data['no_of_style_change'].value_counts()
```

```
0    1050
1     114
2      33
Name: no_of_style_change, dtype: int64
```

Encoding: Performing encoding on the categorical data

```
data[['quarter', 'department', 'day', 'team','no_of_style_change','month']]
```

	quarter	department	day	team	no_of_style_change	month
date						
2015-01-01	Quarter1	sweing	Thursday	8	0	January
2015-01-01	Quarter1	finishing	Thursday	1	0	January
2015-01-01	Quarter1	sweing	Thursday	11	0	January
2015-01-01	Quarter1	sweing	Thursday	12	0	January
2015-01-01	Quarter1	sweing	Thursday	6	0	January
...	...	...	...	...	...	...
2015-03-11	Quarter2	finishing	Wednesday	10	0	March

```
data['team']=data['team'].astype(str)
data['no_of_style_change']=data['no_of_style_change'].astype(str)
data=pd.get_dummies(data)

# Dropping Date column as it is no longer needed
data.drop(columns = 'date',inplace = True)
data.head()
```

	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_workers	actual_productivity	quarter_Quart
date										
2015-01-01	0.80	26.16	1108.0	7080	98	0.0	0	59.0	0.940725	
2015-01-01	0.75	3.94	795.0	960	0	0.0	0	8.0	0.886500	
2015-01-01	0.80	11.41	968.0	3660	50	0.0	0	30.5	0.800570	
2015-01-01	0.80	11.41	968.0	3660	50	0.0	0	30.5	0.800570	
2015-01-01	0.80	25.90	1170.0	1920	50	0.0	0	56.0	0.800382	

5 rows × 40 columns

Scaling

```
from sklearn.preprocessing import MinMaxScaler
cols_to_scale  = ['smv', 'wip', 'over_time', 'incentive', 'no_of_workers', 'idle_time', 'idle_men']
min_max_scaler = MinMaxScaler()
data[cols_to_scale] = min_max_scaler.fit_transform(data[cols_to_scale])
data.head()
```

After encoding and scaling, we can see and describe now dataset shap

```
data.describe()
```

	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_workers	actual_producti
count	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.000000	1197.0C
mean	0.729632	0.235427	0.053902	0.176214	0.010614	0.002434	0.008206	0.374826	0.73
std	0.097891	0.211832	0.094878	0.129198	0.044495	0.042366	0.072644	0.255146	0.17
min	0.070000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.23
25%	0.700000	0.020132	0.034999	0.055556	0.000000	0.000000	0.000000	0.080460	0.65
50%	0.750000	0.239257	0.044646	0.152778	0.000000	0.000000	0.000000	0.367816	0.77
75%	0.800000	0.413473	0.053515	0.268519	0.013889	0.000000	0.000000	0.632184	0.85
max	0.800000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.12

8 rows × 40 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1197 entries, 2015-01-01 to 2015-03-11
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   targeted_productivity  1197 non-null   float64
 1   smv                    1197 non-null   float64
 2   wip                    1197 non-null   float64
 3   over_time              1197 non-null   float64
 4   incentive              1197 non-null   float64
 5   idle_time              1197 non-null   float64
 6   idle_men               1197 non-null   float64
 7   no_of_workers          1197 non-null   float64
 8   actual_productivity    1197 non-null   float64
 9   quarter_Quarter1       1197 non-null   uint8  
 10  quarter_Quarter2       1197 non-null   uint8  
 11  quarter_Quarter3       1197 non-null   uint8  
 12  quarter_Quarter4       1197 non-null   uint8  
 13  quarter_Quarter5       1197 non-null   uint8  
 14  department_finishing   1197 non-null   uint8  
 15  department_sweing      1197 non-null   uint8  
 16  day_Monday             1197 non-null   uint8  
 17  day_Saturday           1197 non-null   uint8  
 18  day_Sunday             1197 non-null   uint8  
 19  day_Thursday           1197 non-null   uint8  
 20  day_Tuesday            1197 non-null   uint8  
 21  day_Wednesday          1197 non-null   uint8  
 22  team_1                 1197 non-null   uint8  
 23  team_10                1197 non-null   uint8  
 24  team_11                1197 non-null   uint8  
 25  team_12                1197 non-null   uint8  
 26  team_2                 1197 non-null   uint8  
 27  team_3                 1197 non-null   uint8  
 28  team_4                 1197 non-null   uint8  
 29  team_5                 1197 non-null   uint8  
 30  team_6                 1197 non-null   uint8  
 31  team_7                 1197 non-null   uint8  
 32  team_8                 1197 non-null   uint8  
 33  team_9                 1197 non-null   uint8  
 34  no_of_style_change_0   1197 non-null   uint8  
 35  no_of_style_change_1   1197 non-null   uint8  
 36  no_of_style_change_2   1197 non-null   uint8  
 37  month_February         1197 non-null   uint8  
 38  month_January          1197 non-null   uint8  
 39  month_March            1197 non-null   uint8  
dtypes: float64(9), uint8(31)
memory usage: 129.8 KB
```

```
#Training & Splitting
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import KFold, cross_val_score
#from sklearn.model_selection import StratifiedKFold, cross_val_score
import warnings
warnings.filterwarnings('ignore')

X, y = data.drop(['actual_productivity'], axis=1), data['actual_productivity']#.astype(int)

clf = DecisionTreeClassifier(random_state=42)

##ss = ShuffleSplit(train_size=0.6, test_size=0.3, n_splits = 5)
#scores = cross_val_score(clf, X, y, cv = ss)

k_folds = KFold(n_splits = 5)
scores = cross_val_score(clf, X, y, cv = k_folds)

#sk_folds = StratifiedKFold(n_splits = 5)
#scores = cross_val_score(clf, X, y, cv = sk_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

```
Cross Validation Scores:  [nan nan nan nan nan]
Average CV Score:  nan
Number of CV Scores used in Average:  5
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
print(X_train.shape)
```

```
(837, 39)
```

```
input_shape = 39
```

```
from sklearn.linear_model import Ridge
from sklearn.feature_selection import RFE
# Creating an estimator
ridge = Ridge()
# Creating RFE object
rfe = RFE(estimator = ridge, n_features_to_select = input_shape, verbose = 1)
# Fitting the training data into our model
rfe.fit(X_train, y_train)
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

```
[('targeted_productivity', True, 1),
 ('smv', True, 1),
 ('wip', True, 1),
 ('over_time', True, 1),
 ('incentive', True, 1),
 ('idle_time', True, 1),
 ('idle_men', True, 1),
 ('no_of_workers', True, 1),
 ('quarter_Quarter1', True, 1),
 ('quarter_Quarter2', True, 1),
 ('quarter_Quarter3', True, 1),
 ('quarter_Quarter4', True, 1),
 ('quarter_Quarter5', True, 1),
 ('department_finishing', True, 1),
 ('department_sweing', True, 1),
 ('day_Monday', True, 1),
 ('day_Saturday', True, 1),
 ('day_Sunday', True, 1),
 ('day_Thursday', True, 1),
 ('day_Tuesday', True, 1),
 ('day_Wednesday', True, 1),
 ('team_1', True, 1),
 ('team_10', True, 1),
 ('team_11', True, 1),
 ('team_12', True, 1),
 ('team_2', True, 1),
 ('team_3', True, 1),
 ('team_4', True, 1),
 ('team_5', True, 1),
 ('team_6', True, 1),
 ('team_7', True, 1),
 ('team_8', True, 1),
```

```
 ('team_9', True, 1),
 ('no_of_style_change_0', True, 1),
 ('no_of_style_change_1', True, 1),
 ('no_of_style_change_2', True, 1),
 ('month_February', True, 1),
 ('month_January', True, 1),
 ('month_March', True, 1)]
```

```
# Let us look at the columns which have been supported by the RFE
RFE_ridge_support_columns = X_train.columns[rfe.support_]
RFE_ridge_support_columns
```

```
Index(['targeted_productivity', '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_10', 'team_11',
       'team_12', 'team_2', 'team_3', 'team_4', 'team_5', 'team_6', 'team_7',
       'team_8', 'team_9', 'no_of_style_change_0', 'no_of_style_change_1',
       'no_of_style_change_2', 'month_February', 'month_January',
       'month_March'],
      dtype='object')
```

```
# Preparing a new dataset containing only the RFE support columns data
X_train = X_train[RFE_ridge_support_columns]

# Preparing a new dataset containing only the RFE support columns data
X_test = X_test[RFE_ridge_support_columns]
X_test.head()
```

	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_workers	quarter_Quarter1	quarter_Qua
date										
2015-03-09	0.7	0.366241	0.067878	0.231481	0.008333	0.0	0.0	0.551724	0	
2015-01-24	0.8	0.000000	0.031149	0.062500	0.000000	0.0	0.0	0.080460	0	
2015-02-10	0.8	0.239257	0.038633	0.157407	0.013889	0.0	0.0	0.367816	0	
2015-02-12	0.7	0.024197	0.026087	0.069444	0.000000	0.0	0.0	0.149425	0	
2015-01-13	0.8	0.445219	0.052087	0.395833	0.016667	0.0	0.0	0.632184	0	

5 rows × 39 columns

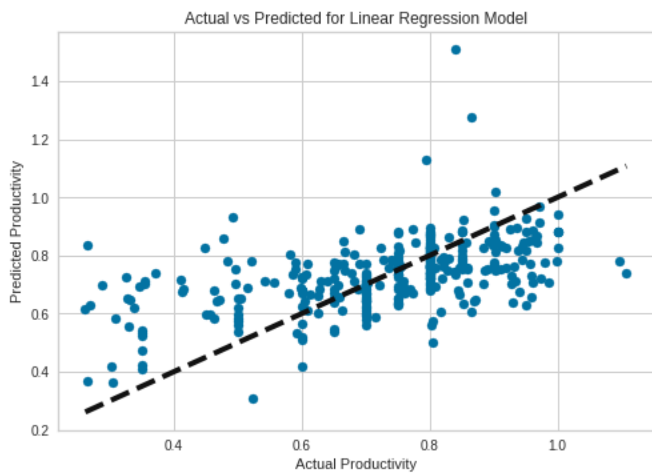
```
# Applying metric's formula
from sklearn import metrics
models_metrics = pd.DataFrame(columns = [ 'models','mae','mse', 'rmse' ,'mape', 'R2'])
def evaluate_model(model,Y_actual,Y_Predicted, df):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    mae=metrics.mean_absolute_error(Y_actual, Y_Predicted)
    mse=metrics.mean_squared_error(Y_actual, Y_Predicted)
    rmse=np.sqrt(metrics.mean_squared_error(Y_actual, Y_Predicted))
    r2 = metrics.r2_score(Y_actual, Y_Predicted)
    df2 = {'models':model,'mae':mae,'mse':mse, 'rmse':rmse, 'mape':mape, 'R2': r2}
    df = df.append(df2, ignore_index = True)
    return df
```

```
#Model Building
from sklearn.linear_model import Ridge, Lasso
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn import linear_model
```

```
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

#Linear Regression Model
# Building a model
model_linear = LinearRegression()
model_linear.fit(X_train,y_train)
#Prediction using test set
y_linear_pred = model_linear.predict(X_test)
models_metrics = evaluate_model('Linear Regression', y_test, y_linear_pred, models_metrics)
```

```
plt.scatter(y_test, y_linear_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Linear Regression Model")
plt.show()
```

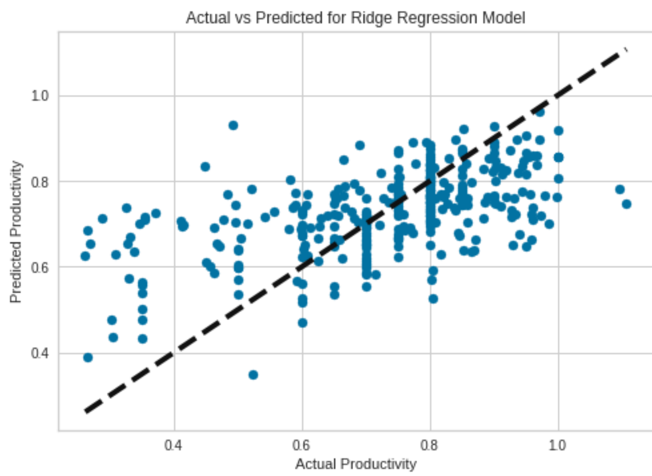



```
#Lasso Model
# Building a model
model_lasso = Lasso()
model_lasso.fit(X_train,y_train)
#Prediction using test set
y_lasso_pred = model_lasso.predict(X_test)
models_metrics = evaluate_model('Lasso Regression', y_test, y_lasso_pred, models_metrics)
```

```
plt.scatter(y_test, y_lasso_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Lasso Regression Model")
plt.show()
```


```
#Ridge Model
# Building a model
model_ridge = Ridge()
model_ridge.fit(X_train,y_train)
#Prediction using test set
y_ridge_pred = model_ridge.predict(X_test)
# Checking with metrics
models_metrics = evaluate_model('Ridge Regression', y_test, y_ridge_pred, models_metrics)
```

```
plt.scatter(y_test, y_ridge_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Ridge Regression Model")
plt.show()
```

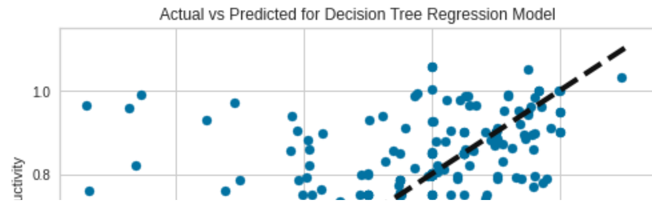



```
models_metrics
```

	models	mae	mse	rmse	mape	R2
0	Linear Regression	0.104630	0.020869	0.144462	17.649078	0.245628
1	Lasso Regression	0.127489	0.027683	0.166383	22.537563	-0.000680
2	Ridge Regression	0.102079	0.018746	0.136915	17.483005	0.322390

```
#Decision Tree
model_dt = DecisionTreeRegressor(random_state = 0)
model_dt.fit(X_train,y_train)
#Prediction using test set
y_dt_pred = model_dt.predict(X_test)
# Checking with metrics
models_metrics = evaluate_model('Decision Tree Regression', y_test, y_dt_pred, models_metrics)
```

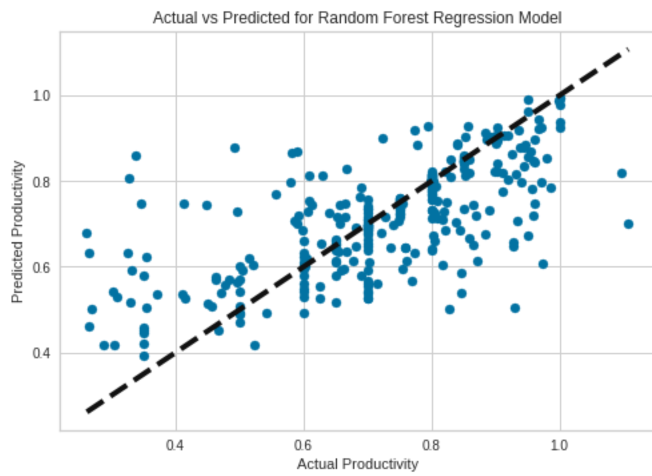
```
plt.scatter(y_test, y_dt_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Decision Tree Regression Model")
plt.show()
```




```
#Random Forest
# Building a model
model_rf = RandomForestRegressor(n_estimators = 100 ,  random_state = 10)
model_rf.fit(X_train,y_train)
#Prediction using test set
y_rf_pred = model_rf.predict(X_test)
# Checking with metrics
models_metrics = evaluate_model('Random Forest Regression', y_test, y_rf_pred, models_metrics)
```

0.4 0.6 0.8 1.0

```
plt.scatter(y_test, y_rf_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Random Forest Regression Model")
plt.show()
```



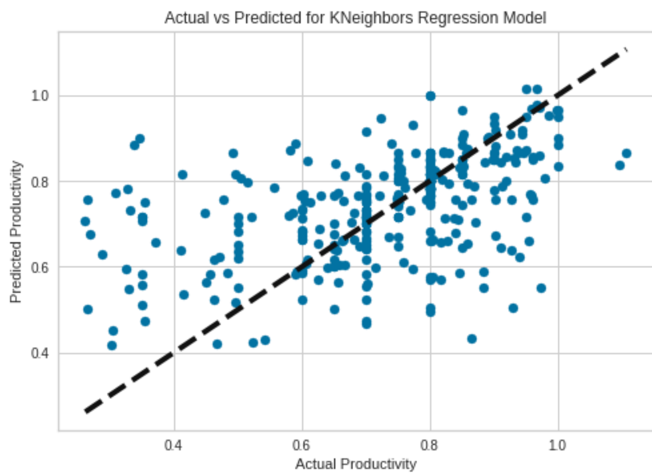

```
#SVR Models
model_svr = SVR(C=25)
model_svr.fit(X_train, y_train)
y_svr_pred = model_svr.predict(X_test)
# model_svr.score(x2_test, y2_test)
models_metrics = evaluate_model('Support Vector Regression', y_test, y_svr_pred, models_metrics)


plt.scatter(y_test, y_svr_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for SVR Regression Model")
plt.show()
```

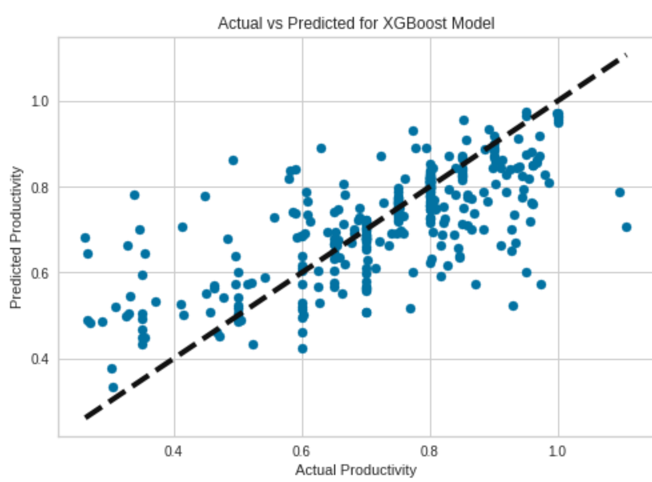

```
#KNeighbors Regression
model_kn = KNeighborsRegressor()
model_kn.fit(X_train, y_train)
y_kn_pred = model_kn.predict(X_test)
models_metrics = evaluate_model('KNeighbour Regression def', y_test, y_kn_pred, models_metrics)
```

```
model_kn = KNeighborsRegressor(n_neighbors=3)
model_kn.fit(X_train, y_train)
y_kn_pred = model_kn.predict(X_test)
models_metrics = evaluate_model('KNeighbour Regression 3', y_test, y_kn_pred, models_metrics)
```

```
plt.scatter(y_test, y_kn_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for KNeighbors Regression Model")
plt.show()
```

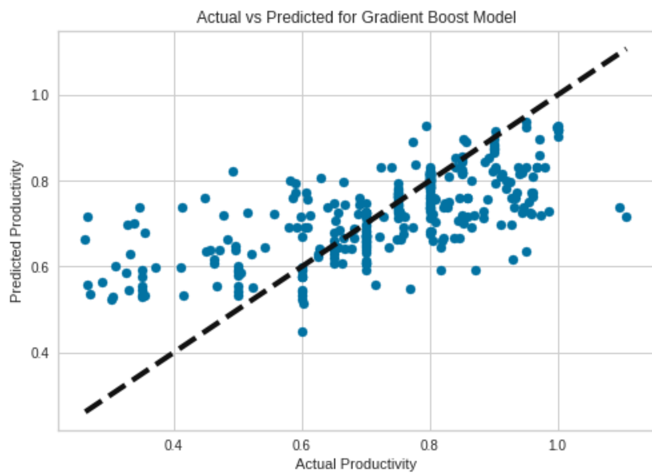



```
#XG Boost
import xgboost as xgb
xgbr = xgb.XGBRegressor(verbosity = 0)
xgbr.fit(X_train, y_train)
y_xgb_pred = xgbr.predict(X_test)
models_metrics = evaluate_model('XGBoost', y_test, y_xgb_pred, models_metrics)
plt.scatter(y_test, y_xgb_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for XGBoost Model")
plt.show()
```



```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingRegressor
model_gbr = GradientBoostingRegressor(alpha=0.9,learning_rate=0.05, max_depth=2, min_samples_leaf=5, min_samples_split=2, n_estimators=100, r

model_gbr.fit(X_train, y_train)
y_gbr_pred = model_gbr.predict(X_test)
models_metrics = evaluate_model('Gradient Boost', y_test, y_gbr_pred, models_metrics)
plt.scatter(y_test, y_gbr_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel("Actual Productivity")
plt.ylabel("Predicted Productivity")
plt.title("Actual vs Predicted for Gradient Boost Model")
plt.show()
```




```
models_metrics
```

	models	mae	mse	rmse	mape	R2
0	Linear Regression	0.104630	0.020869	0.144462	17.649078	0.245628
1	Lasso Regression	0.127489	0.027683	0.166383	22.537563	-0.000680
2	Ridge Regression	0.102079	0.018746	0.136915	17.483005	0.322390
3	Decision Tree Regression	0.097947	0.030112	0.173529	16.671211	-0.088481
4	Random Forest Regression	0.076509	0.014719	0.121320	13.286285	0.467959
5	Support Vector Regression	0.125738	0.026929	0.164101	20.415145	0.026575
6	KNeighbour Regression def	0.103148	0.022045	0.148477	18.535759	0.203116
7	KNeighbour Regression 3	0.100779	0.022295	0.149316	17.986969	0.194076
8	XGBoost	0.079848	0.014027	0.118434	13.393031	0.492969
9	Gradient Boost	0.086104	0.015304	0.123711	15.096265	0.446784

0s completed at 11:41 PM