In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
import lightgbm as lgb
```

In [ ]:
```
# Importing regression models from sklearn
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
from sklearn.svm import SVR
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [ ]:
```
df = pd.read_csv('emp_performance_dataset.csv')
df.head()
```

Out[ ]:

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no
0	9	0.75	3.94	NaN	960	0	0.0	0	
1	7	0.65	30.10	909.0	7080	0	0.0	0	
2	3	0.80	4.15	NaN	1440	0	0.0	0	
3	1	0.65	22.53	762.0	5040	0	0.0	0	
4	4	0.70	30.10	767.0	3300	50	0.0	0	

5 rows × 26 columns

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['team', 'targeted_productivity', 'smv', 'wip', 'over_time', 'incenti
ve',
       'idle_time', 'idle_men', 'no_of_style_change', 'no_of_workers', 'mon
th',
       'quarter_Quarter1', 'quarter_Quarter2', 'quarter_Quarter3',
       'quarter_Quarter4', 'quarter_Quarter5', 'department_finishing',
       'department_finishing ', 'department_sweing', 'day_Monday',
       'day_Saturday', 'day_Sunday', 'day_Thursday', 'day_Tuesday',
       'day_Wednesday', 'actual_productivity'],
      dtype='object')
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017 entries, 0 to 1016
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   team                   1017 non-null   int64  
 1   targeted_productivity  1017 non-null   float64
 2   smv                    1017 non-null   float64
 3   wip                    594 non-null    float64
 4   over_time              1017 non-null   int64  
 5   incentive              1017 non-null   int64  
 6   idle_time              1017 non-null   float64
 7   idle_men               1017 non-null   int64  
 8   no_of_style_change     1017 non-null   int64  
 9   no_of_workers          1017 non-null   float64
 10  month                  1017 non-null   int64  
 11  quarter_Quarter1       1017 non-null   int64  
 12  quarter_Quarter2       1017 non-null   int64  
 13  quarter_Quarter3       1017 non-null   int64  
 14  quarter_Quarter4       1017 non-null   int64  
 15  quarter_Quarter5       1017 non-null   int64  
 16  department_finishing   1017 non-null   int64  
 17  department_finishing   1017 non-null   int64  
 18  department_sweing      1017 non-null   int64  
 19  day_Monday             1017 non-null   int64  
 20  day_Saturday           1017 non-null   int64  
 21  day_Sunday             1017 non-null   int64  
 22  day_Thursday           1017 non-null   int64  
 23  day_Tuesday            1017 non-null   int64  
 24  day_Wednesday          1017 non-null   int64  
 25  actual_productivity    1017 non-null   float64
dtypes: float64(6), int64(20)
memory usage: 206.7 KB
```

- Cheching unique values

In [ ]:
```
df.nunique()
```

Out[ ]:
```
team                      12
targeted_productivity      9
smv                       67
wip                      489
over_time                137
incentive                 47
idle_time                 11
idle_men                   9
no_of_style_change         3
no_of_workers             60
month                      3
quarter_Quarter1           2
quarter_Quarter2           2
quarter_Quarter3           2
quarter_Quarter4           2
quarter_Quarter5           2
department_finishing       2
department_finishing       2
department_sweing          2
day_Monday                 2
day_Saturday               2
day_Sunday                 2
day_Thursday               2
day_Tuesday                2
day_Wednesday              2
actual_productivity      763
dtype: int64
```

- Checking for null values

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
team                      0
targeted_productivity     0
smv                       0
wip                     423
over_time                 0
incentive                 0
idle_time                 0
idle_men                  0
no_of_style_change        0
no_of_workers             0
month                     0
quarter_Quarter1          0
quarter_Quarter2          0
quarter_Quarter3          0
quarter_Quarter4          0
quarter_Quarter5          0
department_finishing      0
department_finishing      0
department_sweing         0
day_Monday                0
day_Saturday              0
day_Sunday                0
day_Thursday              0
day_Tuesday               0
day_Wednesday             0
actual_productivity       0
dtype: int64
```

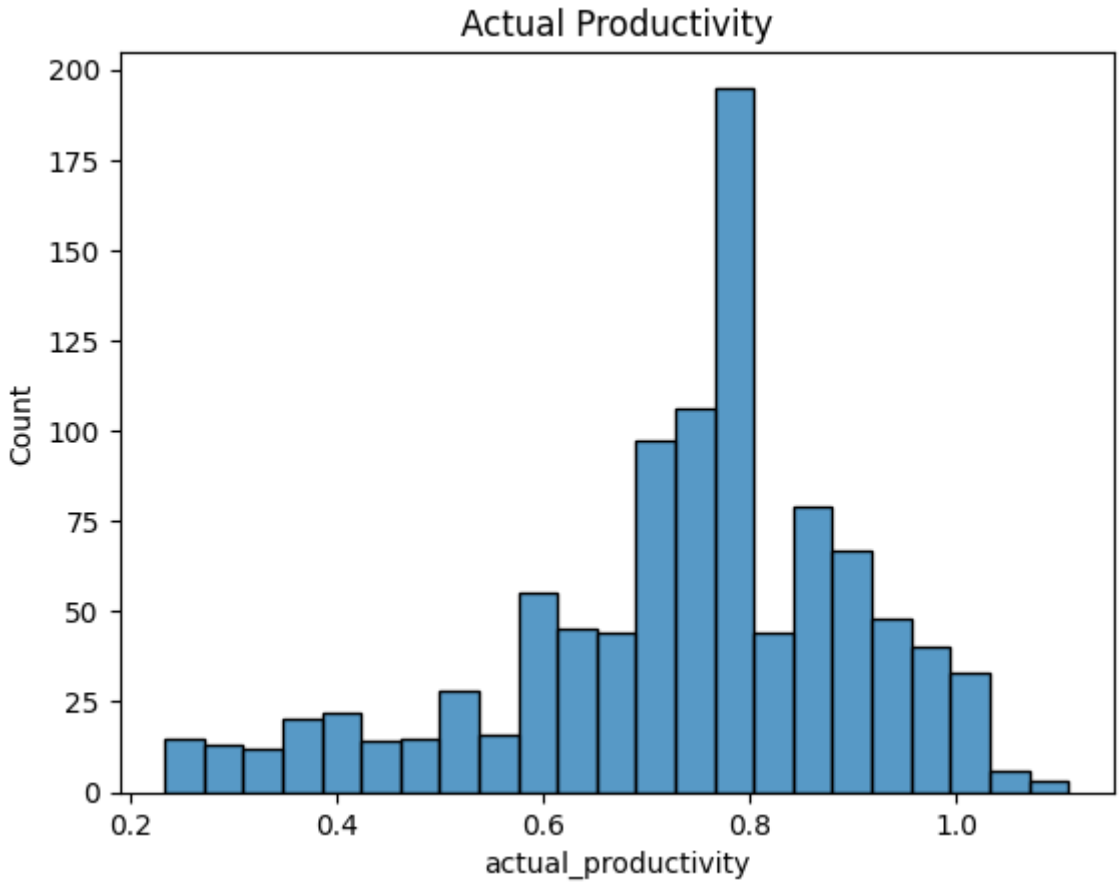
- Replacing null values in wip with it's mean

In [ ]:
```
df['wip'] = df['wip'].fillna(df['wip'].mean())
```

- plotting histogram of actual productivity

In [ ]:
```
sns.histplot(
    df['actual_productivity'],
    kde=False
).set(title='Actual Productivity')
```

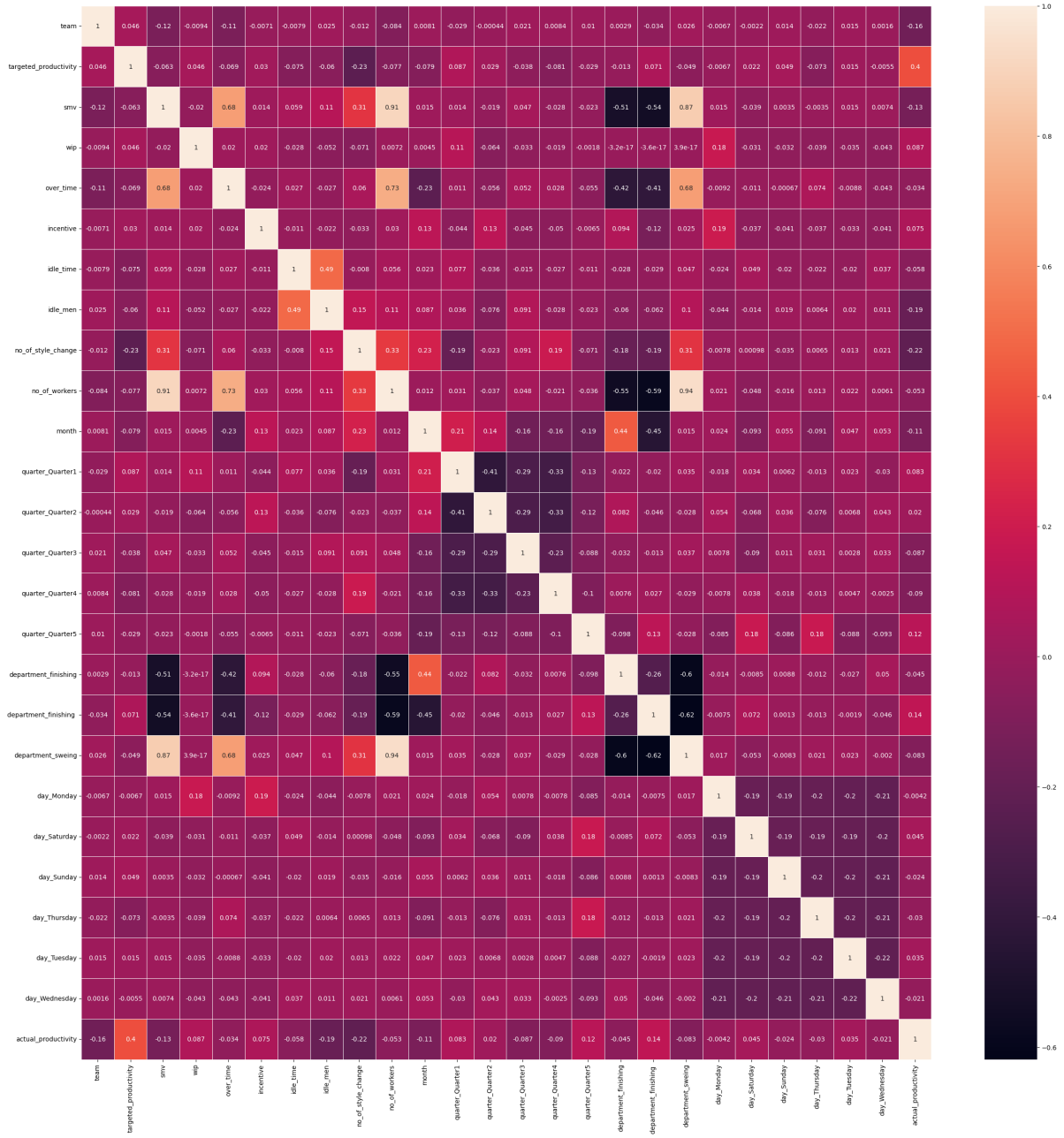
Out[ ]:
```
[Text(0.5, 1.0, 'Actual Productivity')]
```



- Checking correlation between attributes using a heatmap

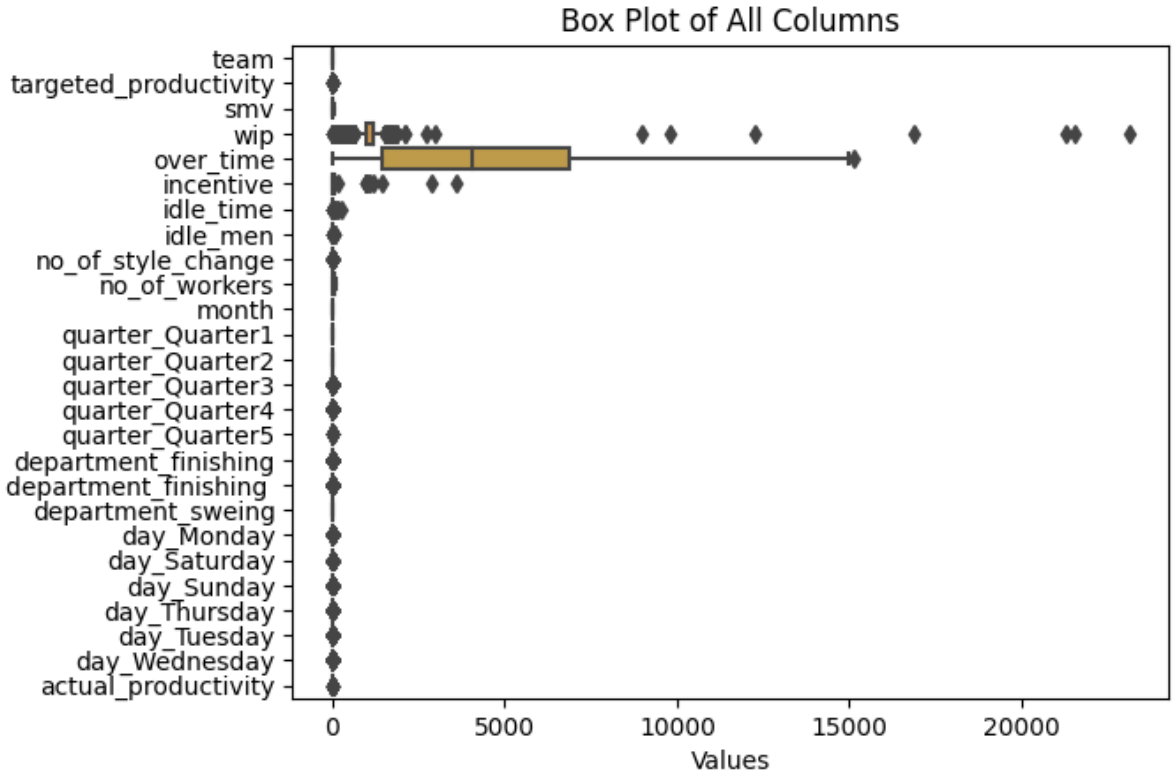
In [ ]:
```
corrMatrix = df.corr()
fig, ax = plt.subplots(figsize = (30, 30))
sns.heatmap(corrMatrix, annot=True, linewidths=.5, ax=ax)
```

Out[ ]:
```
<AxesSubplot: >
```



In [ ]:
```
sns.boxplot(data=df, orient='h')

plt.xlabel('Values')
plt.title('Box Plot of All Columns')
plt.show()
```



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

Out[ ]:

	team	targeted_productivity	smv	wip	over_time	inc
count	1017.000000	1017.000000	1017.000000	1017.000000	1017.000000	1017.0
mean	6.443461	0.730747	15.150492	1183.183502	4532.940020	40.6
std	3.472473	0.097384	10.946096	1370.450653	3275.997333	173.2
min	1.000000	0.070000	2.900000	7.000000	0.000000	0.0
25%	3.000000	0.700000	3.940000	963.000000	1440.000000	0.0
50%	7.000000	0.750000	15.260000	1183.183502	4080.000000	0.0
75%	9.000000	0.800000	24.260000	1183.183502	6900.000000	50.0
max	12.000000	0.800000	54.560000	23122.000000	15120.000000	3600.0

8 rows × 26 columns

In [ ]:
```
df.head()
X = np.array(df.drop(['actual_productivity'], axis=1, inplace=False))
y = np.array(df['actual_productivity'])
```

In [ ]:
```
# X.head()
# X.shape()
```

In [ ]:
```
# y.head()
# y.shape
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
team                     0
targeted_productivity    0
smv                      0
wip                      0
over_time                0
incentive                0
idle_time                0
idle_men                 0
no_of_style_change       0
no_of_workers            0
month                    0
quarter_Quarter1         0
quarter_Quarter2         0
quarter_Quarter3         0
quarter_Quarter4         0
quarter_Quarter5         0
department_finishing     0
department_finishing     0
department_sweing        0
day_Monday               0
day_Saturday             0
day_Sunday               0
day_Thursday             0
day_Tuesday              0
day_Wednesday            0
actual_productivity      0
dtype: int64
```

In [ ]:
```
# spliting data in train and test data
# X = np.array(X)
# y = np.array(y)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, rand
X_train.shape
X_test.shape
```

Out[ ]:
```
(204, 25)
```

- Getting all the regressors

In [ ]:
```
lr = LinearRegression()
dtr = DecisionTreeRegressor()
knr = KNeighborsRegressor()
rfr = RandomForestRegressor()
svr = SVR()
```

In [ ]:
```
# training on all the regressors
for regressor in (lr, dtr, knr, rfr, svr):
    regressor.fit(X_train, y_train)
    print(f'{regressor} score: {regressor.score(X_test, y_test) * 100}%')
```

```
LinearRegression() score: 14.85686859816161%
DecisionTreeRegressor() score: 4.736095792945417%
KNeighborsRegressor() score: 19.261320235459156%
RandomForestRegressor() score: 32.35471904376316%
SVR() score: 11.019300437287905%
```

In [ ]:
```
from lazypredict.Supervised import LazyRegressor
```

In [ ]:
```
lazyr = LazyRegressor()
models,predictions = lazyr.fit(X_train, X_test, y_train, y_test)
models
```

```
 98%|██████████| 41/42 [00:34<00:00,  1.13it/s]
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of t
esting was 0.001978 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 417
[LightGBM] [Info] Number of data points in the train set: 813, number of us
ed features: 23
[LightGBM] [Info] Start training from score 0.733224
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
100%|██████████| 42/42 [00:34<00:00,  1.22it/s]
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[ ]:

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
GradientBoostingRegressor	0.28	0.37	0.14	0.20
HistGradientBoostingRegressor	0.23	0.33	0.15	0.78
LGBMRegressor	0.22	0.31	0.15	0.38
RandomForestRegressor	0.20	0.30	0.15	0.36
AdaBoostRegressor	0.16	0.27	0.16	0.04
ExtraTreesRegressor	0.15	0.25	0.16	0.34
BaggingRegressor	0.14	0.24	0.16	0.04
XGBRegressor	0.06	0.17	0.17	0.32
NuSVR	0.06	0.17	0.17	0.08
LassoLarsCV	0.03	0.15	0.17	0.17
LassoCV	0.03	0.15	0.17	0.22
TransformedTargetRegressor	0.03	0.15	0.17	0.02
LinearRegression	0.03	0.15	0.17	0.14
ElasticNetCV	0.03	0.15	0.17	0.28
LassoLarsIC	0.03	0.15	0.17	0.07
RidgeCV	0.03	0.15	0.17	0.49
Ridge	0.03	0.15	0.17	0.00
HuberRegressor	0.02	0.14	0.17	0.02
BayesianRidge	0.02	0.14	0.17	0.31
SGDRegressor	0.02	0.14	0.17	0.01
LarsCV	0.01	0.13	0.17	0.14
Lars	-0.01	0.12	0.17	0.03
OrthogonalMatchingPursuitCV	-0.01	0.12	0.17	0.01
LinearSVR	-0.01	0.11	0.17	0.11
TweedieRegressor	-0.01	0.11	0.17	0.27
SVR	-0.02	0.11	0.17	0.03
GammaRegressor	-0.02	0.11	0.17	0.02
PoissonRegressor	-0.02	0.10	0.17	0.02
OrthogonalMatchingPursuit	-0.03	0.10	0.17	0.01
DecisionTreeRegressor	-0.06	0.07	0.18	0.01
QuantileRegressor	-0.14	-0.00	0.18	27.64
DummyRegressor	-0.15	-0.01	0.18	0.00
LassoLars	-0.15	-0.01	0.18	0.06
Lasso	-0.15	-0.01	0.18	0.06
ElasticNet	-0.15	-0.01	0.18	0.08
ExtraTreeRegressor	-0.24	-0.09	0.19	0.00

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
KNeighborsRegressor	-0.31	-0.15	0.20	0.01
MLPRegressor	-0.45	-0.27	0.21	0.57
PassiveAggressiveRegressor	-0.54	-0.35	0.21	0.01
GaussianProcessRegressor	-4.62	-3.92	0.40	0.44
KernelRidge	-18.42	-16.03	0.75	0.30
RANSACRegressor	-22.67	-19.76	0.83	0.19

In [ ]:
```
Gr = GradientBoostingRegressor()
Hgr = HistGradientBoostingRegressor()
Ada = AdaBoostRegressor()
knr = KNeighborsRegressor()
xbgi = xgb.XGBRegressor
Lgbmr = lgb.LGBMRegressor
```

In [ ]:
```
print("GradientBoostingRegressor")
Gr.fit(X, y)
y_pred = Gr.predict(X)
MSEGr=mean_squared_error(y_pred, y)
print('MSE for GradientBoostingRegressor        : ', MSEGr)
R2Gr=r2_score(y_pred, y)
print('R2 for GradientBoostingRegressor : ', R2Gr)


print(" HistGradientBoostingRegressor")
Hgr.fit(X, y)
y_pred = Hgr.predict(X)
MSEHGr=mean_squared_error(y_pred, y)
print('MSE HistGradientBoostingRegressor        : ', MSEHGr)
R2HGr=r2_score(y_pred, y)
print('R2 for HistGradientBoostingRegressor     : ', R2HGr)

print('AdaBoostRegressor')
Ada.fit(X, y)
y_pred = Ada.predict(X)
MSEada=mean_squared_error(y_pred, y)
print('MSE for AdaBoostRegressor        : ', MSEada)
R2ada=r2_score(y_pred, y)
print('R2 for AdaBoostRegressor : ', R2ada)

print('KNeighborsRegressor')
knr.fit(X, y)
y_pred = knr.predict(X)
MSEknr=mean_squared_error(y_pred, y)
print('MSE for KNeighborsRegressor: ', MSEknr)
R2knr=r2_score(y_pred, y)
print('R2 for KNeighborsRegressor: ', R2knr)
```

```
GradientBoostingRegressor
MSE for GradientBoostingRegressor       :  0.009968464773837726
R2 for GradientBoostingRegressor        :  0.40081288774685
 HistGradientBoostingRegressor
MSE HistGradientBoostingRegressor       :  0.005142214241329054
R2 for HistGradientBoostingRegressor    :  0.7550591906283303
AdaBoostRegressor
MSE for AdaBoostRegressor       :  0.017241871565336363
R2 for AdaBoostRegressor        :  -0.787922839076264
KNeighborsRegressor
MSE for KNeighborsRegressor:  0.016849722870148895
R2 for KNeighborsRegressor:  -0.33338838704006135
```

- XGBregressor

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

Out[ ]:

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idl
count	1017.00	1017.00	1017.00	1017.00	1017.00	1017.00	1017.00	
mean	6.44	0.73	15.15	1183.18	4532.94	40.69	0.56	
std	3.47	0.10	10.95	1370.45	3276.00	173.24	10.09	
min	1.00	0.07	2.90	7.00	0.00	0.00	0.00	
25%	3.00	0.70	3.94	963.00	1440.00	0.00	0.00	
50%	7.00	0.75	15.26	1183.18	4080.00	0.00	0.00	
75%	9.00	0.80	24.26	1183.18	6900.00	50.00	0.00	
max	12.00	0.80	54.56	23122.00	15120.00	3600.00	270.00	

8 rows × 26 columns

In [ ]:
```
xgb_r = xgb.XGBRegressor(
    objective ='reg:pseudohubererror',
    huber_slope=1,
    n_estimators = 10,
    max_depth=6,
    booster='dart',
    learning_rate=0.40,
    min_child_weight= 1,
    importance_type='gain',
    base_score=0.5,
    seed = 123
)



xgb_r.fit(X_train, y_train)
pred = xgb_r.predict(X_test)

xgbmse = mean_squared_error(y_test, pred)
xgbrsqure = r2_score(y_test, pred)

print(f'R2 Score: {xgbrsqure * 100}%')
```

```
R2 Score: 38.13377531661442%
```

In [ ]:
```
model = xgb.XGBRegressor(
    objective ='reg:pseudohubererror',
    n_estimators = 10,
    max_depth=6,
    booster='gbtree',
    learning_rate=0.79,
    min_child_weight= 1,
    importance_type='gain',
    colsample_bytree = 1,
    base_score=0.51,
    seed = 123
)

# model = GradientBoostingRegressor()
model.fit(X, y)
pred= model.predict(X)
xgbrsqure = r2_score(y, pred)
print(f'R2 Score : {xgbrsqure}')
# from sklearn.model_selection import KFold
# folds = KFold(n_splits = 5)
```

```
R2 Score : 0.859375043229464
```

In [ ]:
```
pred = model.predict(X)

xgbmse = mean_squared_error(y, pred)
xgbrsqure = r2_score(y, pred)
mae = mean_absolute_error(y, pred)


print(f"MSE : {xgbmse}")
print(f'R2 Score: {xgbrsqure}')
print(f'RMSE : {xgbmse**0.5}')
print(f'MAE : {mae}')
```

```
MSE : 0.004268241855426315
R2 Score: 0.859375043229464
RMSE : 0.06533178288877715
MAE : 0.040647754637355094
```

Saving the model

In [ ]:
```
import pickle
pickle.dump(model, open('model.pkl', 'wb'))
```

In [ ]:
```
df.head()
```

Out[ ]:

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	n
0	9	0.75	3.94	1183.18	960	0	0.00	0	
1	7	0.65	30.10	909.00	7080	0	0.00	0	
2	3	0.80	4.15	1183.18	1440	0	0.00	0	
3	1	0.65	22.53	762.00	5040	0	0.00	0	
4	4	0.70	30.10	767.00	3300	50	0.00	0	

5 rows × 26 columns

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

Out[ ]:

	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idl
count	1017.00	1017.00	1017.00	1017.00	1017.00	1017.00	1017.00	
mean	6.44	0.73	15.15	1183.18	4532.94	40.69	0.56	
std	3.47	0.10	10.95	1370.45	3276.00	173.24	10.09	
min	1.00	0.07	2.90	7.00	0.00	0.00	0.00	
25%	3.00	0.70	3.94	963.00	1440.00	0.00	0.00	
50%	7.00	0.75	15.26	1183.18	4080.00	0.00	0.00	
75%	9.00	0.80	24.26	1183.18	6900.00	50.00	0.00	
max	12.00	0.80	54.56	23122.00	15120.00	3600.00	270.00	

8 rows × 26 columns

Testing the saved model

In [ ]:
```
# sample_test = np.array([
#     5, 0.9, 20, 1000, 3000, 50, 0.2,  0.3, 0.3, 37, 1, 1, 1, 1, 0, 1, 1, 0
# ])

sample_test = np.array(df.iloc[853][:-1])
print(list(df.iloc[853][:-1]))
# print(len(sample_test))
# print(sample_test)
pickle_model = pickle.load(open('model.pkl', 'rb'))

print(df.iloc[853])

productivity = pickle_model.predict([sample_test])

print(f'{productivity[0] * 100}%')
```

```
[1.0, 0.8, 3.94, 1183.1835016835016, 1440.0, 0.0, 0.0, 0.0, 0.0, 8.0, 1.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
team                        1.00
targeted_productivity       0.80
smv                         3.94
wip                      1183.18
over_time                1440.00
incentive                   0.00
idle_time                   0.00
idle_men                    0.00
no_of_style_change          0.00
no_of_workers               8.00
month                       1.00
quarter_Quarter1            0.00
quarter_Quarter2            1.00
quarter_Quarter3            0.00
quarter_Quarter4            0.00
quarter_Quarter5            0.00
department_finishing        0.00
department_finishing        1.00
department_sweing           0.00
day_Monday                  1.00
day_Saturday                0.00
day_Sunday                  0.00
day_Thursday                0.00
day_Tuesday                 0.00
day_Wednesday               0.00
actual_productivity         0.96
Name: 853, dtype: float64
96.87544107437134%
```

In [ ]: