student

May 23, 2021

- Student name: Tamjid Ahsan
- Student pace: Full Time
- Scheduled project review date/time: May 27, 2021, 05:00 PM [DST]
- Instructor name: James Irving
- Blog post URL: TBA

1 INTRODUCTION

The garment industry one of the highly labor-intensive industries that needs large number of human resources to efficient and keep up with demand for garment products across the globe. Because of this inherent dependency on human capital, the production of a garment company comprehensively relies on the productivity of the employees in different departments. Often actual productivity of the garment employees is not in line with targeted productivity that was set. This is a high priority for a organization to achieve deadline and maximize profit by ensuring proper utilization of resources. When any productivity gap occurs, the company faces a huge loss in production.

2 BUSINESS PROBLEM

A garment production pipeline consists of a handful of sequential processes, e.g., designing, sample confirmation, sourcing and merchandising, lay planning, marker planning, spreading and cutting, sewing, washing, finishing, and packaging, and then exporting if the order is a international one. An efficient garment production always consists of a line plan with explicit details of when the production will be started, how many pieces are expected, and when the order needs to be completed. To complete a whole production within a target time, these sequential processes need to be to performed efficiently. In order to meet the production goals, the associated industrial engineers strategically set a targeted productivity value against each working team in the manufacturing process. However, it is a common scenario that the actual productivity does not align with the target for several factors, both internal and external.

I shall use various machine learning techniques for predicting the productivity of the garment employees based on their previous data of internal factors.

More specifically: - Predict bad performance of workers. Optimize model for precision. - Focus on predicting bad performance, don't want to miss much of those. - Focus on maximizing true negatives and minimizing false positives while tackling model overfitting.

3 IMPORTS

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
# all this codes are displayed in the appendix of the notebook
# custom functions and packages loader
import imports_and_functions as fun
from imports_and_functions.packages import *

# notebook styling packages
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks='True', grid='False')
## to reset to default theme
# jtplot.reset()
```

4 OBTAIN

The data is obtained from UCI Machine Learning Repository, titled "Productivity Prediction of Garment Employees Data Set" by Abdullah Al Imran[1]. Which can be found here. A copy of the data is in this repository at `/data/garments_worker_productivity.csv`.

The collected dataset contains the production data of the sewing and finishing department for three months from January 2015 to March 2015 of a renowned garment manufacturing company in Bangladesh[2]. The dataset consists of 1197 instances and includes 13 attributes.

Features with explanation.

- `date`: Date in MM-DD-YYYY format.
- `day`: Day of the Week.
- `quarter`: A portion of the month. A month was divided into four quarters.
- `department`: Associated department with the instance.
- `team_no`: Associated team number with the instance.
- `no_of_workers`: Number of workers in each team.
- `no_of_style_change`: Number of changes in the style of a particular product.
- `targeted_productivity`: Targeted productivity set by the Authority for each team for each day.
- `smv`: Standard Minute Value, it is the allocated time for a task.
- `wip`: Work in progress. Includes the number of unfinished items for products.
- `over_time`: Represents the amount of overtime by each team in minutes.
- `incentive`: Represents the amount of financial incentive (in BDT[3]) that enables or motivates a particular course of action.
- `idle_time`: The amount of time when the production was interrupted due to several reasons.
- `idle_men`: The number of workers who were idle due to production interruption.
- `actual_productivity`: The actual % of productivity that was delivered by the workers. It ranges from 0-1[4].

---

[1] Rahim, M. S., Imran, A. A., & Ahmed, T. (2021). Mining the Productivity Data of

Garment Industry. International Journal of Business Intelligence and Data Mining, 1(1), 1. [2] Bangladesh is a developing country which is the second largest apparel exporting country in the world. [3] 1 USD = 84.83 BDT, as of May 23,2021. Check here from Bangladesh Bank. [4] Measured by production production engineers of the organization. Methodology of this calculation is not public.

Reference:

@article{Rahim_2021, doi = {10.1504/ijbidm.2021.10028084}, url = {[Web Link]}, year = 2021, publisher = {Inderscience Publishers}, volume = {1}, number = {1}, pages = {1}, author = {Md Shamsur Rahim and Abdullah Al Imran and Tanvir Ahmed}, title = {Mining the Productivity Data of Garment Industry}, journal = {International Journal of Business Intelligence and Data Mining} }

5 SCRUB & EXPLORE

5.1 data

[3]:
```
# this dataset is also available for use from the following url.
# this analysis can be also performed by directly loading data from UCI.
# df = pd.read_csv(
#     'https://archive.ics.uci.edu/ml/machine-learning-databases/00597/
↪garments_worker_productivity.csv'
# )
```

[4]:
```
# loading data from local source
df = pd.read_csv('./data/garments_worker_productivity.csv')
```

[5]:
```
# 10 sample of the dataset. Data loading successful.
df.sample(10)
```

[5]:
```
           date   quarter  … no_of_workers actual_productivity
797   2/16/2015  Quarter3  …          10.0            0.629417
693   2/10/2015  Quarter2  …          18.0            0.966759
530   1/31/2015  Quarter5  …           5.0            0.971867
325   1/19/2015  Quarter3  …          56.5            0.750518
1055   3/4/2015  Quarter1  …          57.0            0.800333
1036   3/3/2015  Quarter1  …           8.0            0.702778
890   2/23/2015  Quarter4  …          59.0            0.800137
470   1/27/2015  Quarter4  …          58.0            0.700386
597    2/3/2015  Quarter1  …           8.0            0.495417
945   2/26/2015  Quarter4  …          59.0            0.800809

[10 rows x 15 columns]
```

[6]:
```
df.dtypes
```

[6]:
```
date                      object
quarter                   object
department                object
day                       object
team                       int64
targeted_productivity    float64
smv                      float64
wip                      float64
over_time                  int64
incentive                  int64
idle_time                float64
idle_men                   int64
no_of_style_change         int64
no_of_workers            float64
actual_productivity      float64
dtype: object
```

Observation: * every feature has correct data type except `team`. * `team` is a categorical data which is labeled numerically.

[7]:
```
fun.check_NaN(df)
```

[7]:

	name	is_null	not_null
0	date	0	1197
1	quarter	0	1197
2	department	0	1197
3	day	0	1197
4	team	0	1197
5	targeted_productivity	0	1197
6	smv	0	1197
7	wip	506	691
8	over_time	0	1197
9	incentive	0	1197
10	idle_time	0	1197
11	idle_men	0	1197
12	no_of_style_change	0	1197
13	no_of_workers	0	1197
14	actual_productivity	0	1197

`wip` has NaN values. Those are not missing. For those days where there were no work in progress, data is empty. Those can be safely filled with 0.

[8]:
```
fun.check_duplicates(df, verbose=True)
```

```
date >> number of uniques: 59
['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 >> number of uniques: 5
['Quarter1' 'Quarter2' 'Quarter3' 'Quarter4' 'Quarter5']
------------------------------------------------------------
department >> number of uniques: 3
['sweing' 'finishing ' 'finishing']
------------------------------------------------------------
day >> number of uniques: 6
['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
------------------------------------------------------------
team >> number of uniques: 12
[ 8  1 11 12  6  7  2  3  9 10  5  4]
------------------------------------------------------------
targeted_productivity >> number of uniques: 9
[0.8  0.75 0.7  0.65 0.6  0.35 0.5  0.07 0.4 ]
------------------------------------------------------------
smv >> number of uniques: 70
[26.16  3.94 11.41 25.9  28.08 19.87 19.31  2.9  23.69  4.15 11.61 45.67
 21.98 31.83 12.52 42.41 20.79 50.48  4.3  22.4  42.27 27.13 14.61 51.02
 22.52 14.89 22.94 48.68 41.19 48.84 26.87 20.4  49.1  15.26 54.56 40.99
 29.12  4.08 42.97 15.09 30.4  48.18 20.1  38.09 18.79 23.54 50.89 24.26
 20.55 30.1  25.31 10.05 18.22  5.13 29.4  30.33 19.68 21.25  4.6   3.9
 22.53 21.82 27.48 26.66 20.2  15.28 26.82 16.1  23.41 30.48]
------------------------------------------------------------
wip >> number of uniques: 549, showing top 150 values
[1108.   nan  968. 1170.  984.  795.  733.  681.  872.  578.  668.  861.
  772.  913. 1261.  844. 1005.  659. 1152. 1138.  610.  944.  544. 1072.
  539. 1278. 1227. 1039.  878. 1033.  782. 1216.  513.  734. 1202.  884.
 1255. 1047.  678.  712. 1037.  757.  759. 1083.  666. 1187. 1305.  716.
  925.  963. 1101. 1035.  910. 1209.  590.  808. 1179. 1324. 1135.  776.
  990.  986.  924. 1120. 1066. 1144.  413.  568. 1189.  942. 1050. 1026.
  783.  857.  548.  411.  287.  724. 1122.  970. 1158.  660.  749.  893.
  887. 1335. 1082. 1075.  966. 1095. 1383. 1012.  896.  805.  762. 1043.
  831.  562. 1208. 1099. 1093. 1031. 1233.  941.  843.  760.  737.  381.
 1141. 1004.  581. 1073. 1156. 1211. 1126. 1063.  723.  465.  530. 1297.
  715. 1150. 1232. 1218. 1159.  972. 1092.  965.  816.  947.  838. 1086.
 1160. 1177. 1281. 1369. 1084.  391. 1102. 1076.  917. 1044. 1067. 1396.
 1292.  171. 1128.  865.  825. 1163.]
------------------------------------------------------------
over_time >> number of uniques: 143
[ 7080   960  3660  1920  6720  6900  6000  6480  2160  7200  1440  6600
  5640  1560  6300  6540 13800  6975  7020  6780  4260  6660  4320  6960
```

```
  2400  3840  4800  4440  1800  2700 10620 10350  9900  5310 10170  4470
 10530 10440  5490  5670  9720 12600 10050 15120 14640   900 25920 10260
  2760  4710  9540  7680  3600  6420  7980  3240  8220  6930  8460  7350
  5400  1620  1980  2970  7320  5100  3390  1260  3420  8970  4950 10080
  9810  6570  5040  4380  3630  8280  6120  5580  3720  5760  7470 10500
  6360  4140  8400 12180  9000 15000 10770 12000  9360  3060  2520   720
  3780 10320   360  6840  1080  1200  4080   240  5880  6240  4200  3960
   600  2280  5940  1320  5460  2040  4020  3000  3360  5820  6060  2640
  7500  2880   120  3300     0  3480  7380  4560  7140  5160  5280   840
  5520   480  8160  5700  2820  5340  1680  7560  1700  4680  3120]
------------------------------------------------------------
incentive >> number of uniques: 48
[  98    0   50   38   45   34   44   63   56   40   60   26   75   23
   35   69   88   30   54   37   70   27   21   24   94   29   81   55
  119   90  113   46  100   53   93   49  138   33   32   62   65  960
 1080 2880 3600 1440 1200   25]
------------------------------------------------------------
idle_time >> number of uniques: 12
[  0.   90.  150.  270.  300.    2.    5.    8.    4.5   3.5   4.    6.5]
------------------------------------------------------------
idle_men >> number of uniques: 10
[ 0 10 15 45 37 30 35 20 25 40]
------------------------------------------------------------
no_of_style_change >> number of uniques: 3
[0 1 2]
------------------------------------------------------------
no_of_workers >> number of uniques: 61
[59.   8.  30.5 56.  57.5 55.  54.  18.  60.  12.  20.  17.  56.5 54.5
 29.5 31.5 31.  55.5 58.  10.  16.  32.  58.5 15.   5.  57.  53.  51.5
  2.   9.   7.  19.  28.  34.  89.  14.  25.  52.   4.  21.  35.  51.
 33.  11.  33.5 22.  26.  27.  59.5 50.  44.  49.  47.  48.  42.  24.
 45.  46.  39.  38.   6. ]
------------------------------------------------------------
actual_productivity >> number of uniques: 879, showing top 150 values
[0.94072542 0.8865     0.80057049 0.80038194 0.800125   0.75516667
 0.75368348 0.75309753 0.75042783 0.72112696 0.71220525 0.7070459
 0.70591667 0.67666667 0.59305556 0.54072917 0.52118    0.43632639
 0.98802469 0.98788044 0.95627083 0.94527778 0.90291667 0.80072531
 0.80032294 0.80031864 0.80023729 0.80014865 0.78729969 0.78244792
 0.75024303 0.7018125  0.70013404 0.69996522 0.62833333 0.6253125
 0.99138889 0.93164583 0.91522917 0.87971448 0.86167901 0.85056949
 0.85043644 0.85034513 0.80059806 0.80023784 0.8000302  0.79210417
 0.75922839 0.75034846 0.68270833 0.66760417 0.60343218 0.34583333
 0.96105903 0.93951389 0.89366319 0.87539062 0.82083333 0.80441667
 0.80068437 0.80025096 0.80024601 0.80007652 0.763375   0.75927083
 0.7504     0.66458333 0.60002874 0.96678135 0.93649621 0.89916667
 0.88868687 0.85814394 0.85050231 0.80964015 0.80590909 0.80059447
 0.80027383 0.80014097 0.80012872 0.80007657 0.75054546 0.75005785
```

```
 0.68106061 0.64998328 0.61625    0.95142046 0.8805303  0.85013677
 0.83       0.82718654 0.81337121 0.80464015 0.80034377 0.80024675
 0.8        0.70048083 0.66651515 0.41211983 0.33011364 0.94768939
 0.9199054  0.90021572 0.89172348 0.85018182 0.83575758 0.82135417
 0.80049725 0.80010714 0.80002493 0.77979167 0.73598485 0.71262626
 0.51560606 0.34995139 0.23370548 0.985      0.93034038 0.91158974
 0.85117411 0.84695076 0.81742424 0.81710227 0.80102821 0.80034644
 0.8001171  0.75009835 0.67324528 0.67007576 0.62888258 0.38800781
 0.33797349 0.93532197 0.92564394 0.87306818 0.82829546 0.69018282
 0.66808712 0.65359848 0.60913826 0.60022985 0.59734849 0.59043561
 0.4731348  0.45297963 0.95515151 0.9422138  0.90545454 0.85052217]
------------------------------------------------------------
```

[8]:

	name	duplicated	not_duplicated
0	date	1138	59
1	quarter	1192	5
2	department	1194	3
3	day	1191	6
4	team	1185	12
5	targeted_productivity	1188	9
6	smv	1127	70
7	wip	648	549
8	over_time	1054	143
9	incentive	1149	48
10	idle_time	1185	12
11	idle_men	1187	10
12	no_of_style_change	1194	3
13	no_of_workers	1136	61
14	actual_productivity	318	879

- `smv` depends on product.
- `department` has issue with naming
- Value of 'Quarter5' in `quarter` is inconsistent with data description.

every other feature is clean and coherent.

[9]:
```python
# looking into `quarter`
df[df.quarter=='Quarter5']
```

[9]:

	date	quarter	…	no_of_workers	actual_productivity
498	1/29/2015	Quarter5	…	57.0	1.000230
499	1/29/2015	Quarter5	…	10.0	0.989000
500	1/29/2015	Quarter5	…	57.0	0.950186
501	1/29/2015	Quarter5	…	57.5	0.900800
502	1/29/2015	Quarter5	…	56.0	0.900130
503	1/29/2015	Quarter5	…	10.0	0.899000
504	1/29/2015	Quarter5	…	8.0	0.877552
505	1/29/2015	Quarter5	…	8.0	0.864583
506	1/29/2015	Quarter5	…	10.0	0.856950

```
507  1/29/2015  Quarter5  ...           10.0            0.853667
508  1/29/2015  Quarter5  ...           58.0            0.850362
509  1/29/2015  Quarter5  ...           58.0            0.850170
510  1/29/2015  Quarter5  ...           59.0            0.800474
511  1/29/2015  Quarter5  ...           10.0            0.773333
512  1/29/2015  Quarter5  ...           35.0            0.750647
513  1/29/2015  Quarter5  ...            9.0            0.634667
514  1/29/2015  Quarter5  ...           51.0            0.600598
515  1/29/2015  Quarter5  ...           33.0            0.500118
516  1/29/2015  Quarter5  ...            8.0            0.492500
517  1/29/2015  Quarter5  ...           55.0            0.487920
518  1/31/2015  Quarter5  ...           58.0            1.000457
519  1/31/2015  Quarter5  ...           57.0            1.000230
520  1/31/2015  Quarter5  ...           10.0            0.971867
521  1/31/2015  Quarter5  ...            8.0            0.971867
522  1/31/2015  Quarter5  ...           10.0            0.971867
523  1/31/2015  Quarter5  ...           10.0            0.971867
524  1/31/2015  Quarter5  ...           15.0            0.971867
525  1/31/2015  Quarter5  ...            2.0            0.971867
526  1/31/2015  Quarter5  ...            9.0            0.971867
527  1/31/2015  Quarter5  ...            2.0            0.971867
528  1/31/2015  Quarter5  ...           10.0            0.971867
529  1/31/2015  Quarter5  ...            8.0            0.971867
530  1/31/2015  Quarter5  ...            5.0            0.971867
531  1/31/2015  Quarter5  ...           10.0            0.971867
532  1/31/2015  Quarter5  ...           56.0            0.920237
533  1/31/2015  Quarter5  ...           57.5            0.900537
534  1/31/2015  Quarter5  ...           57.0            0.850611
535  1/31/2015  Quarter5  ...           58.0            0.850362
536  1/31/2015  Quarter5  ...           35.0            0.750647
537  1/31/2015  Quarter5  ...           59.0            0.656764
538  1/31/2015  Quarter5  ...           54.0            0.650148
539  1/31/2015  Quarter5  ...           33.0            0.600711
540  1/31/2015  Quarter5  ...           56.0            0.388830
541  1/31/2015  Quarter5  ...           54.0            0.286985

[44 rows x 15 columns]
```

Those data are for January 29, Thursday; and 31, Saturday of 2015. I can not come up with any rational for this treatment, Thus leaving it at is. Another option is to merge these with `Quarter4`.

[10]:
```
df.describe().transpose().round(2).style.format("{0:,.2f}")
```

[10]:
```
<pandas.io.formats.style.Styler at 0x2b306fddbe0>
```

- Can spot unusual low value for `targeted productivity`. Further investigation is required.
- `incentive` has a wide range. I can further comment on those after I peek into their distribution.

[11]:

```
fun.distribution_of_features(df,color_plot='gold')
```


Histogram plots of the dataset
Histplot of Date
Histplot of Quarter
Histplot of Department
Histplot of Day
Histplot of Team
Histplot of Targeted_Productivity
Histplot of Smv
Histplot of Wip
Histplot of Over_Time
Histplot of Incentive
Histplot of Idle_Time
Histplot of Idle_Men
Histplot of No_Of_Style_Change
Histplot of No_Of_Workers
Histplot of Actual_Productivity
Count
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
Quarter1
Quarter2
Quarter3
Quarter4
Quarter5
sweing
finishing
finishing
Thursday
Saturday
Sunday
Monday
Tuesday
Wednesday

- None of them are normally distributed.
- Most of them are skewed. e.g., `idle_men`, `idle_time`, `incentive`, `wip`.
- `target` has few regular occurring values.
- `smv`, `overtime` has some very high values

5.2 Feature engineering

5.2.1 Creating target; `performance`

I am treating this as a binary classification model. For this I am converting `actual_productivity` into a binary class. Logic behind this operation is, if the `actual_productivity` is greater than `targeted_productivity` then its a 1, and 0 otherwise. I am not encoding in text as most of the model requirs numerical data in target. This eliminates the need for label encoding. And for binary classification this does not create confusion while looking at reports of model performance.

[12]:
```
# binary target class, int
lst = []
for x in zip(df.targeted_productivity, df.actual_productivity):
    # % change in variables
    delta = np.log(x[1] / x[0])
    if delta < 0:
        lst.append(0)
    else:
        lst.append(1)
df['performance'] = lst
```

[13]:
```
# checking for class imbalance
df.performance.value_counts(1)
```

[13]:
```
1    0.730994
0    0.269006
Name: performance, dtype: float64
```

Straight away I can spot a class imbalance issue. I have to address this later while modeling.

5.2.2 Cleaning `wip`

[14]:
```
# filling NaN's with 0, meaning no wip for that session
df['wip'] = df['wip'].fillna(0)
```

5.2.3 Text cleaning in department categories

[15]:
```
df['department'].value_counts()
```

[15]:
```
sweing       691
finishing    257
```

```
finishing     249
Name: department, dtype: int64
```

[16]:
```
# cleaning spaces
df['department'] = df['department'].str.strip()
# checking
df['department'].value_counts()
```

[16]:
```
sweing       691
finishing    506
Name: department, dtype: int64
```

5.2.4 Cleaning quarter

[17]:
```
# as identified before, cleaning by merging Quarter5 with Quarter4
df.at[df[df.quarter == 'Quarter5'].index, 'quarter'] = 'Quarter4'
```

[18]:
```
# checking
df[df.quarter == 'Quarter5']
```

[18]:
```
Empty DataFrame
Columns: [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, performance]
Index: []
```

5.2.5 Cleaning targeted_productivity

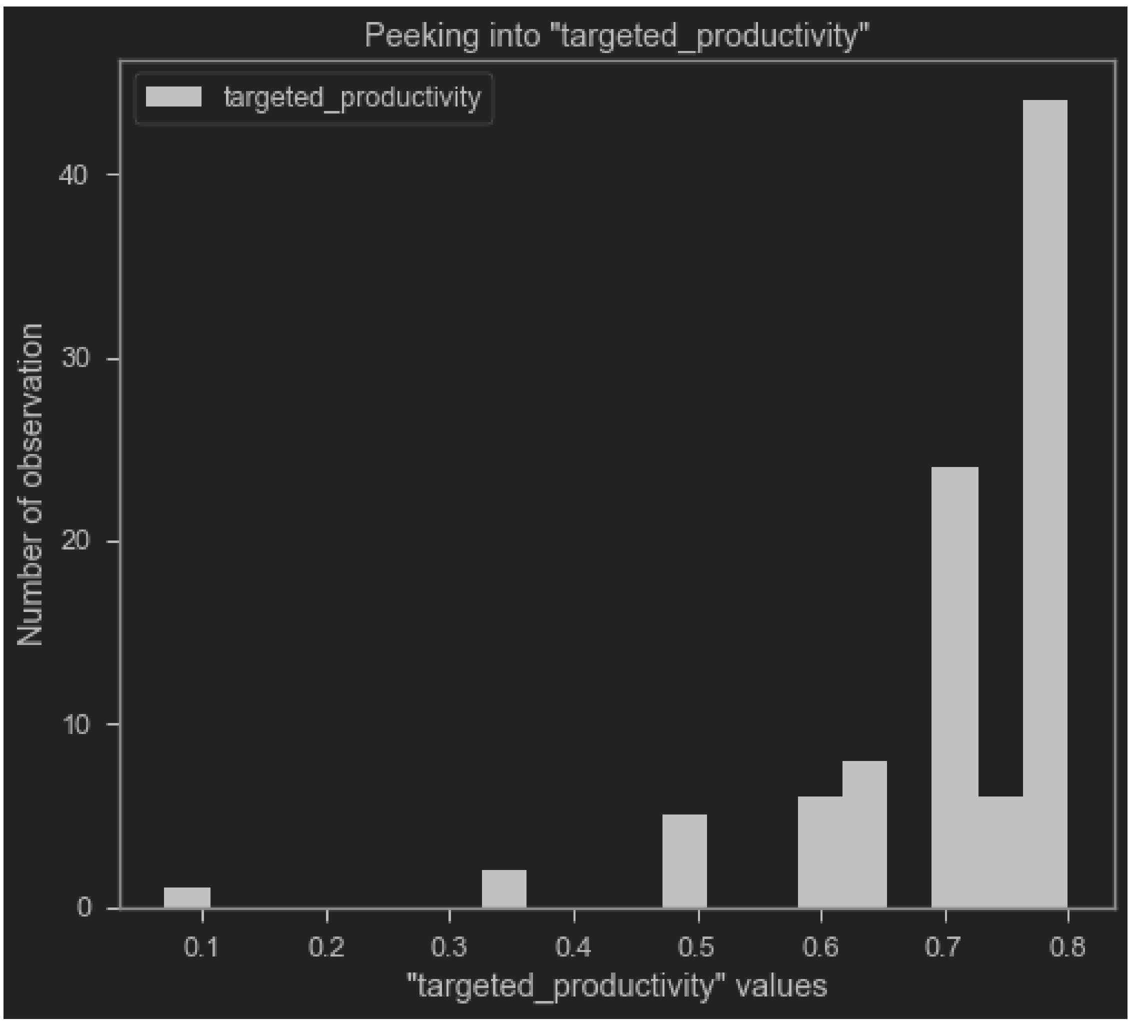
[19]:
```
# correcting possible error in data
# according to my plot, I am isolating data to pin point issue
df[df.targeted_productivity<.3]
```

[19]:
```
          date   quarter  … actual_productivity performance
633   2/5/2015  Quarter1  …            0.522845           1

[1 rows x 16 columns]
```

[20]:
```
df[df.team == 7]['targeted_productivity'].hist(color='silver',
                                                 grid=False,
                                                 legend=True,
                                                 bins=20)
plt.title('Peeking into "targeted_productivity"')
plt.xlabel('"targeted_productivity" values')
plt.ylabel('Number of observation')

print(f"""targeted_productivity stats:
{'*'*30}
```

```
 {df.targeted_productivity.mode()[0]}\t : mode
 {round(df.targeted_productivity.mean(),2)}\t : mean
 {df.targeted_productivity.quantile(.25)}\t : 25% quantile
{'*'*30}""")
```

```
targeted_productivity stats:
******************************
 0.8     : mode
 0.73    : mean
 0.7     : 25% quantile
******************************
```



From this plot I can safely assume that this a data entry error. Setting a target so low does not make any sense. I am filling this with the 25% quantile .

[21]:
```python
df.at[df[df.targeted_productivity < .3].index,
      'targeted_productivity'] = df.targeted_productivity.quantile(.25)

df[df.targeted_productivity < .3]
```

[21]:
```
Empty DataFrame
Columns: [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, performance]
Index: []
```

No error remains.

5.2.6 Drop features

Dropping `date` as this is not useful for modeling and timing is captured in `day` and `quarter` features, `actual_productivity` as this is the target in continuous format.

[22]:
```python
df.drop(columns=['date', 'actual_productivity'], inplace=True)
```

5.2.7 dtype casting

[23]:
```python
df.dtypes
```

[23]:
```
quarter                   object
department                object
day                       object
team                       int64
targeted_productivity    float64
smv                      float64
wip                      float64
over_time                  int64
incentive                  int64
idle_time                float64
idle_men                   int64
no_of_style_change         int64
no_of_workers            float64
performance                int64
dtype: object
```

[24]:
```python
# setting all categorical variabls as 'category'.
df['quarter'] = df['quarter'].astype('category')
df['department'] = df['department'].astype('category')
df['day'] = df['day'].astype('category')
df['team'] = df['team'].astype('category')
```

[25]:
```python
df.dtypes
```

[25]:
```
quarter                    category
department                 category
day                        category
team                       category
targeted_productivity       float64
smv                         float64
wip                         float64
over_time                     int64
incentive                     int64
idle_time                   float64
idle_men                      int64
no_of_style_change            int64
no_of_workers               float64
performance                   int64
dtype: object
```

Everything cleaned.

[26]:
```
# cleaned dataset for modeling
df
```

[26]:
```
       quarter department  … no_of_workers performance
0     Quarter1     sweing  …          59.0           1
1     Quarter1  finishing  …           8.0           1
2     Quarter1     sweing  …          30.5           1
3     Quarter1     sweing  …          30.5           1
4     Quarter1     sweing  …          56.0           1
…          …         …  …           …           …
1192  Quarter2  finishing  …           8.0           0
1193  Quarter2  finishing  …           8.0           0
1194  Quarter2  finishing  …           8.0           0
1195  Quarter2  finishing  …          15.0           0
1196  Quarter2  finishing  …           6.0           0

[1197 rows x 14 columns]
```

5.2.8 EDA

Data preparation for visuals

[27]:
```
# custom colour pallate
cust_pal = ['#f22b07', 'lime']
cust_pal2 = ['gold', 'silver']
# dataset for eda
df_eda = df.copy()

# binning data for visuals
```

```python
#### smv ###
df_eda['smv_bin'] = pd.qcut(x=df_eda.smv,
                            q=4,
                            labels=['quick', 'normal', 'long', 'extra_long'])
#### wip ####
# filling NaN's with 0, meaning no wip for that session
df_eda['wip'] = df_eda['wip'].fillna(0)
# intervals for binning
bins = pd.IntervalIndex.from_tuples([(0, 1), (1, 150), (150, 500), (500, 2500),
                                     (2500, 1e6)],
                                    closed='left')
# binning
wip_size = pd.cut(df_eda['wip'].tolist(), bins=bins)
# naming categories
wip_size.categories = ['no_wip', 'small', 'med', 'large', 'xl']
# appending to df_eda
df_eda['wip_bin'] = wip_size

#### incentive ####
# intervals for binning
bins = pd.IntervalIndex.from_tuples([(0, 30), (30, 80), (80, 500), (500, 2500),
                                     (2500, 1e6)],
                                    closed='left')
# binning
incentive_size = pd.cut(df_eda['incentive'].tolist(), bins=bins)
# naming categories
incentive_size.categories = [
    'no_incentive', 'minor', 'med', 'large', 'generous'
]
# appending to df_eda
df_eda['incentive_bin'] = incentive_size
```

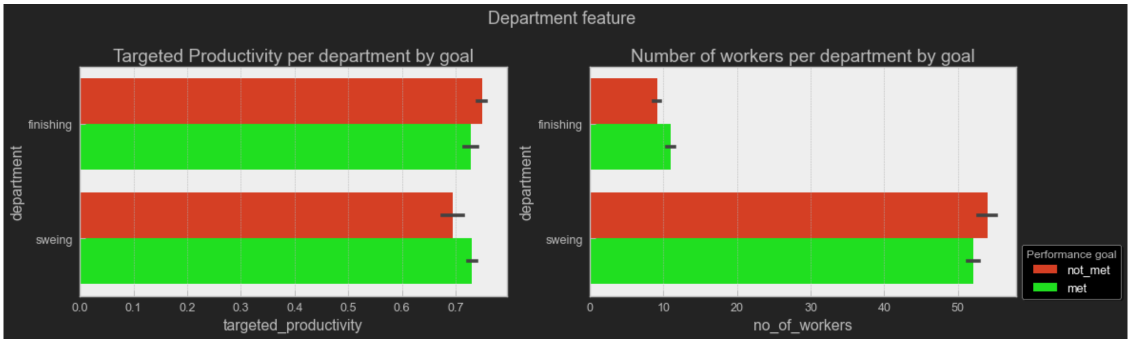
My rational for binning this features at those interval are explained in details when I explore those more in the following section.

---

which department has better performance

[28]:
```python
with plt.style.context('bmh'):
    plt.figure(figsize=(16.5, 5))
    plt.subplot(121)
    ax = sns.barplot(data=df_eda,
                     y='department',
                     ci=95,
                     x='targeted_productivity',
                     hue='performance',
                     palette=cust_pal)
    # remove legend
```

```
    ax.legend([], [], frameon=False)
    # add legend, ## comment out `remove legend` step and uncomment follwing␣
↪code to add legend
    # legend_labels, _ = ax.get_legend_handles_labels()
    # ax.legend(
    #     legend_labels,
    #     ['not_met', 'met'],
    #     bbox_to_anchor=(1.26, 1),
    #     title_fontsize=12,
    #     title='Performance')
    ax.set_title('Targeted Productivity per department by goal')

    plt.subplot(122, sharey=ax)
    ax1 = sns.barplot(data=df_eda,
                      y='department',
                      x='no_of_workers',
                      hue='performance',
                      ci=95,
                      palette=cust_pal)
    legend_labels, _ = ax1.get_legend_handles_labels()
    ax1.legend(legend_labels, ['not_met', 'met'],
               title_fontsize=12,labelcolor='white',
               title='Performance goal',facecolor='black',
               bbox_to_anchor=(1.26, .25),
               shadow=True,
               fancybox=True)
    ax1.set_title('Number of workers per department by goal')

    plt.suptitle('Department feature', size=18, weight=4)
    plt.tight_layout()
    plt.show()
```



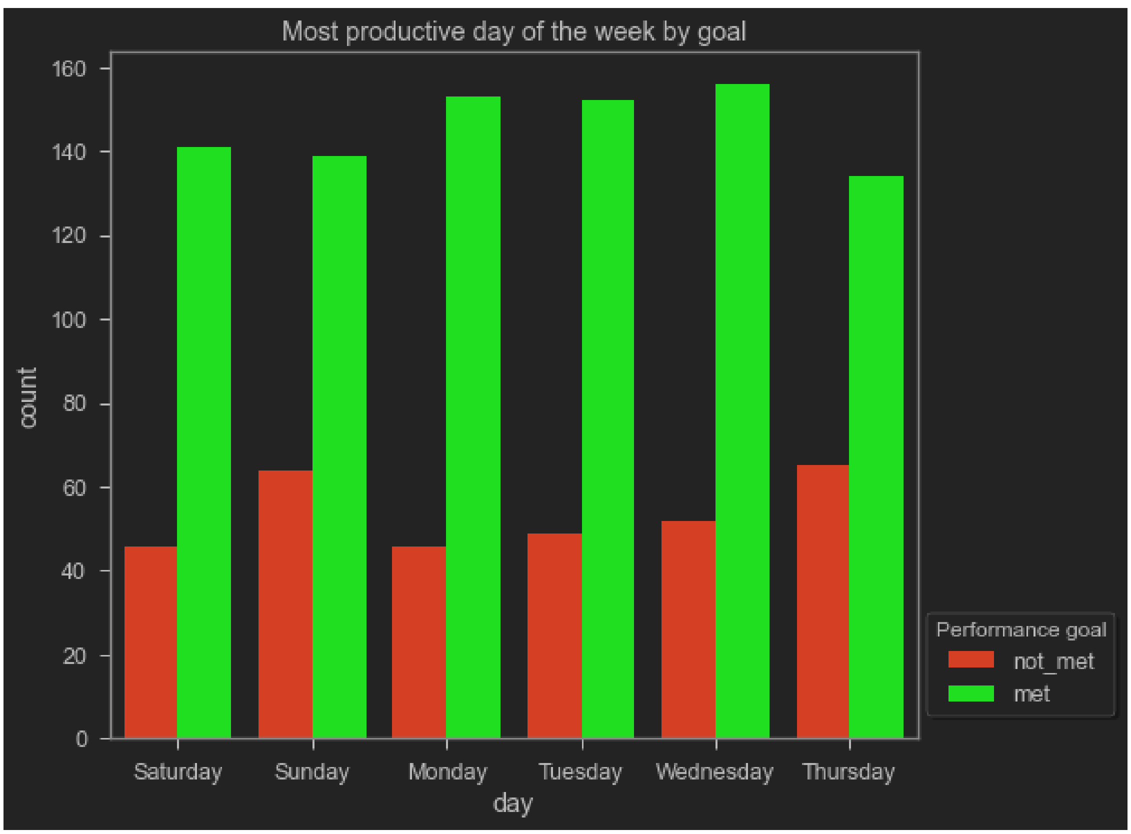
Finishing department and sewing department has similar targets and finishing department often fail to meet daily goal. This can be explained by the small size of finishing department. Adding a few workers can be beneficial.

productive day of the week

[29]:
```
df_eda.day.cat.categories
```

[29]:
```
Index(['Monday', 'Saturday', 'Sunday', 'Thursday', 'Tuesday', 'Wednesday'],
dtype='object')
```

[30]:
```
ax1 = sns.countplot(
    data=df_eda,
    hue='performance',
    x='day',
    palette=cust_pal,
    order=['Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday'])
legend_labels, _ = ax1.get_legend_handles_labels()
ax1.legend(legend_labels, ['not_met', 'met'],
           bbox_to_anchor=(1.26, .2),
           title_fontsize=12,
           title='Performance goal',
           shadow=True,
           fancybox=True)
ax1.set_title('Most productive day of the week by goal')
plt.show()
```

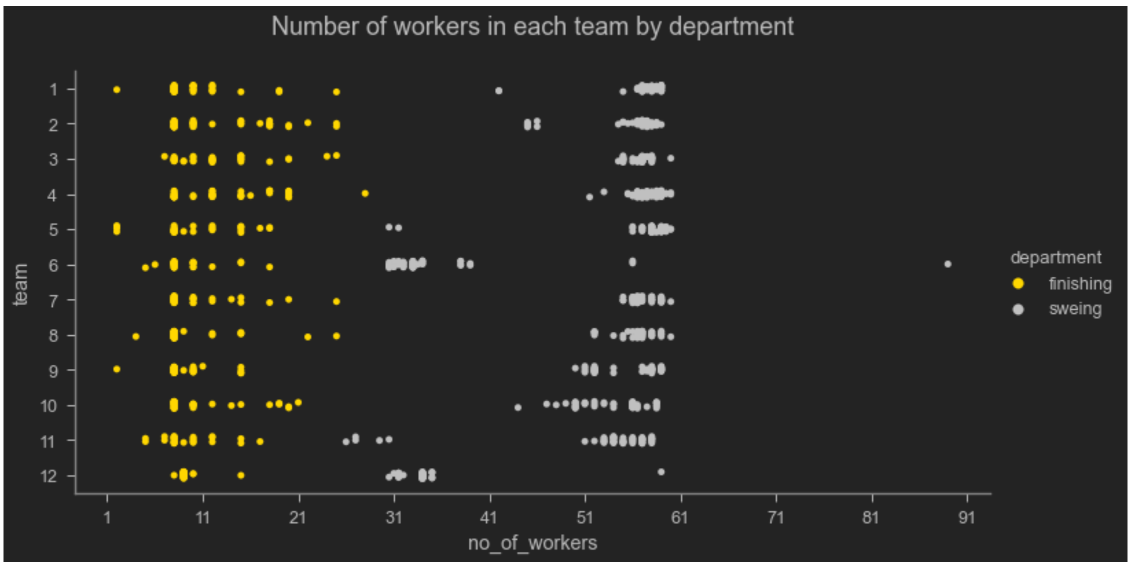


Overall same pattern with slight high level of goal not met on Sunday and Thursday.

exploring team

team size

[31]:
```
g = sns.catplot(data=df_eda,
                y='team',
                x='no_of_workers',
                hue='department',
                palette=cust_pal2, aspect=2)
plt.xticks(ticks=np.arange(1, 95, 10), rotation=0)
# g.set(yticks=df_eda.team.cat.categories)
plt.title(f'Number of workers in each team by department\n',size=18,weight=4)
plt.show()
```



Generally finishing department worker size is low.

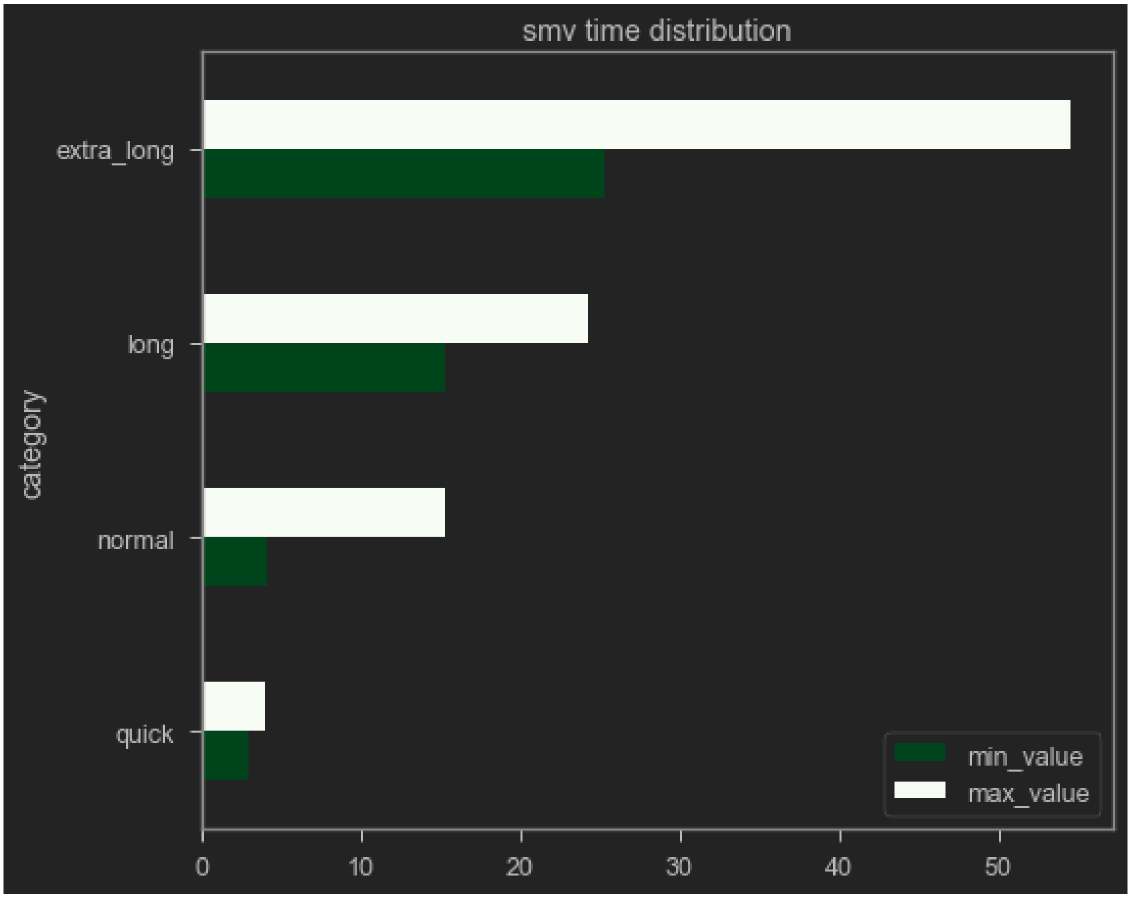
efficient team

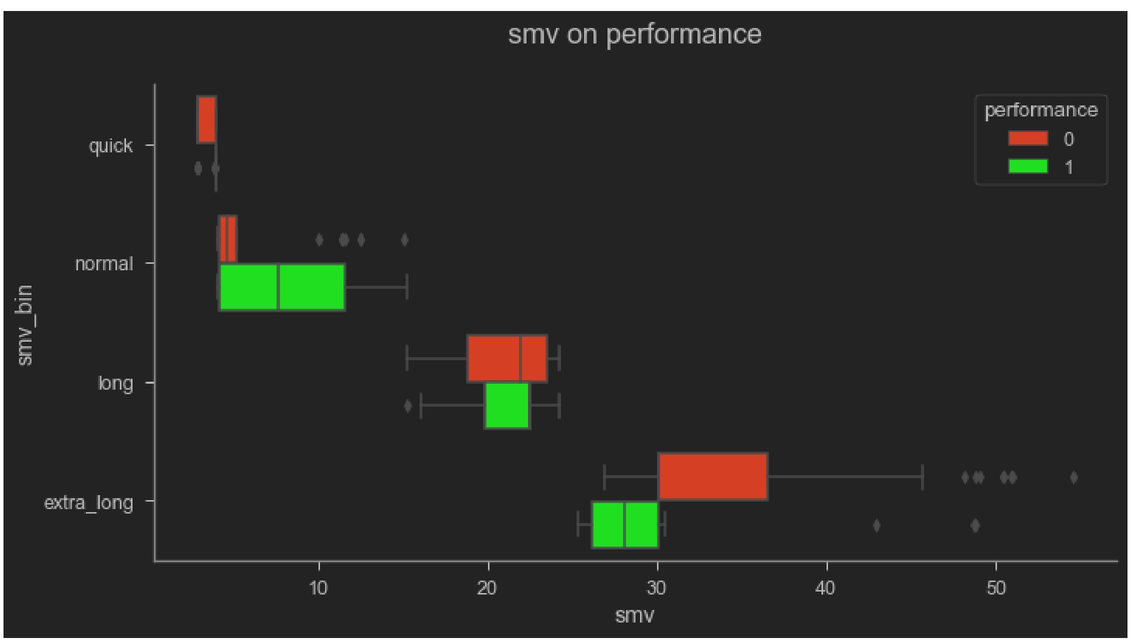
[32]:
```
g = sns.catplot(data=df_eda,
                y='team',
                x='department',
                hue='performance',
                palette=cust_pal,
                aspect=2)
g.set(yticks=df_eda.team.unique())
plt.title(f'Efficiency of teams\n', size=18, weight=4)

plt.show()
```

Finishing department fails to achieve goal more often.

smv on performance

[33]:
```
# empty list to hold data
lst = []
# preparing data of smv_bin distribution
for i in zip(df_eda.smv_bin.cat.categories,
             df_eda.groupby('smv_bin')['smv'].agg('min'),
             df_eda.groupby('smv_bin')['smv'].agg('max')):
    temp_dict = {'category': i[0], 'min_value': i[1], 'max_value': i[2]}
    lst.append(temp_dict)
# plotting
pd.DataFrame(lst).set_index('category').plot.barh(
    colormap='Greens_r', title='smv time distribution')
sns.catplot(y="smv_bin",
            x='smv',
            kind="box",
            hue="performance",
            data=df_eda,
            aspect=2,
            palette=cust_pal,
            legend_out=False)
plt.title(f'smv on performance\n', size=18, weight=4)
plt.show()
```

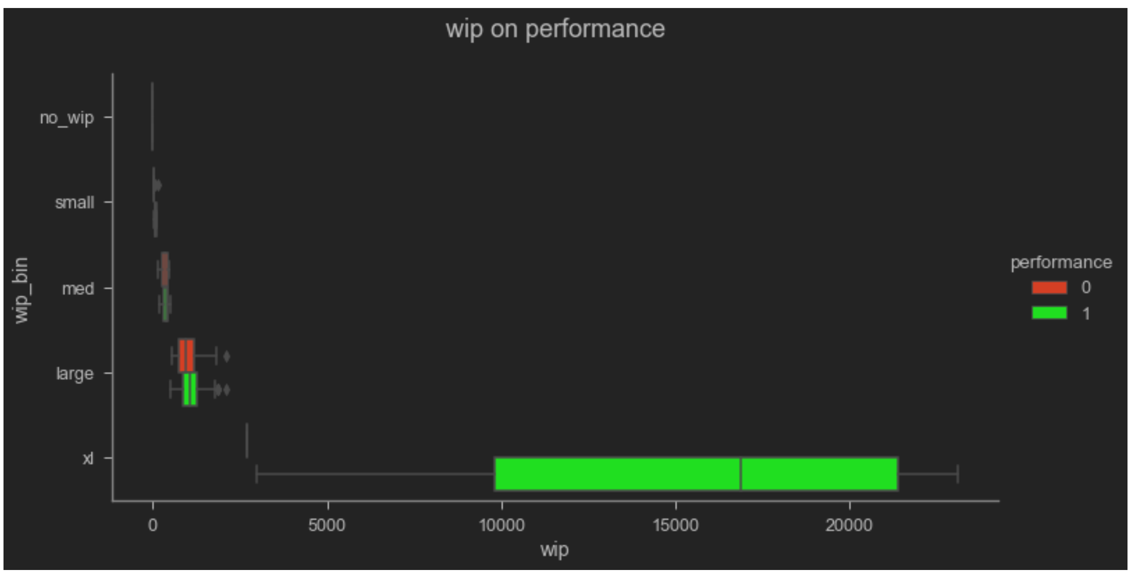

smv time distribution
category
extra_long
long
normal
quick
min_value
max_value
0
10
20
30
40
50


smv on performance
smv_bin
quick
normal
long
extra_long
performance
0
1
10
20
30
40
50
smv

wip on performance

[34]:
```
# empty list to hold data
lst = []
# preparing data of wip_bin distribution
for i in zip(df_eda.wip_bin.cat.categories,
             df_eda.groupby('wip_bin')['wip'].agg('min'),
             df_eda.groupby('wip_bin')['wip'].agg('max')):
    temp_dict = {'category': i[0], 'min_value': i[1], 'max_value': i[2]}
    lst.append(temp_dict)
display(
    pd.DataFrame(lst).set_index('category').style.format(
        "{:.0f}").set_properties(**{'color': 'lawngreen'}))
sns.catplot(y="wip_bin",
            x='wip',
            kind="box",
            hue="performance",
            data=df_eda,
            aspect=2,
            palette=cust_pal)
plt.title(f'wip on performance\n', size=18, weight=4)
plt.show()
```

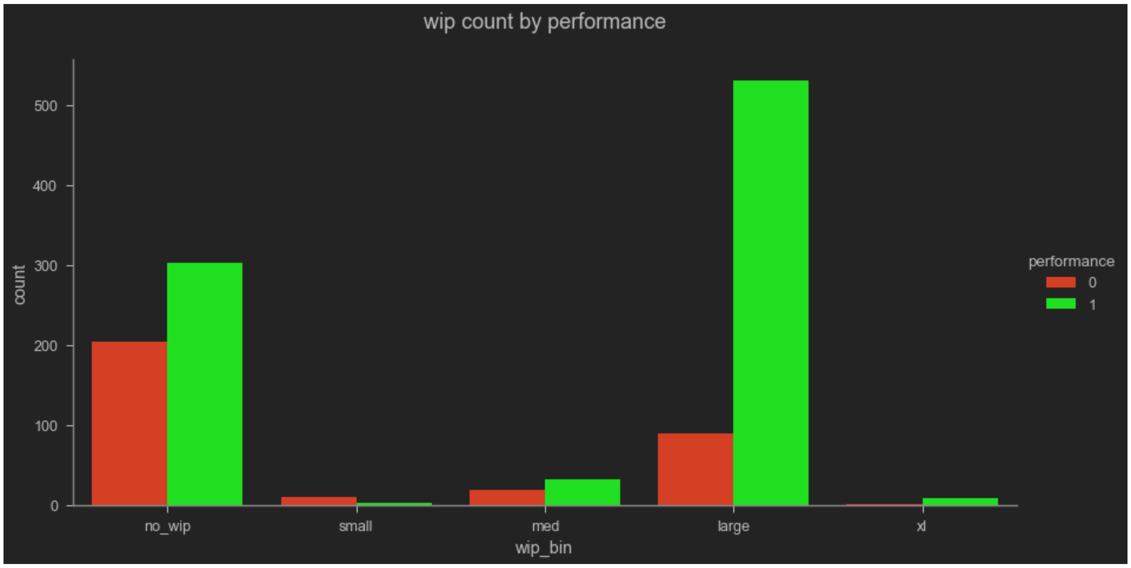
```
<pandas.io.formats.style.Styler at 0x2b3082b9700>
```



At higher wip there is less chance of failing.

[35]:
```
sns.catplot(x="wip_bin",
            kind="count",
            hue="performance",
```

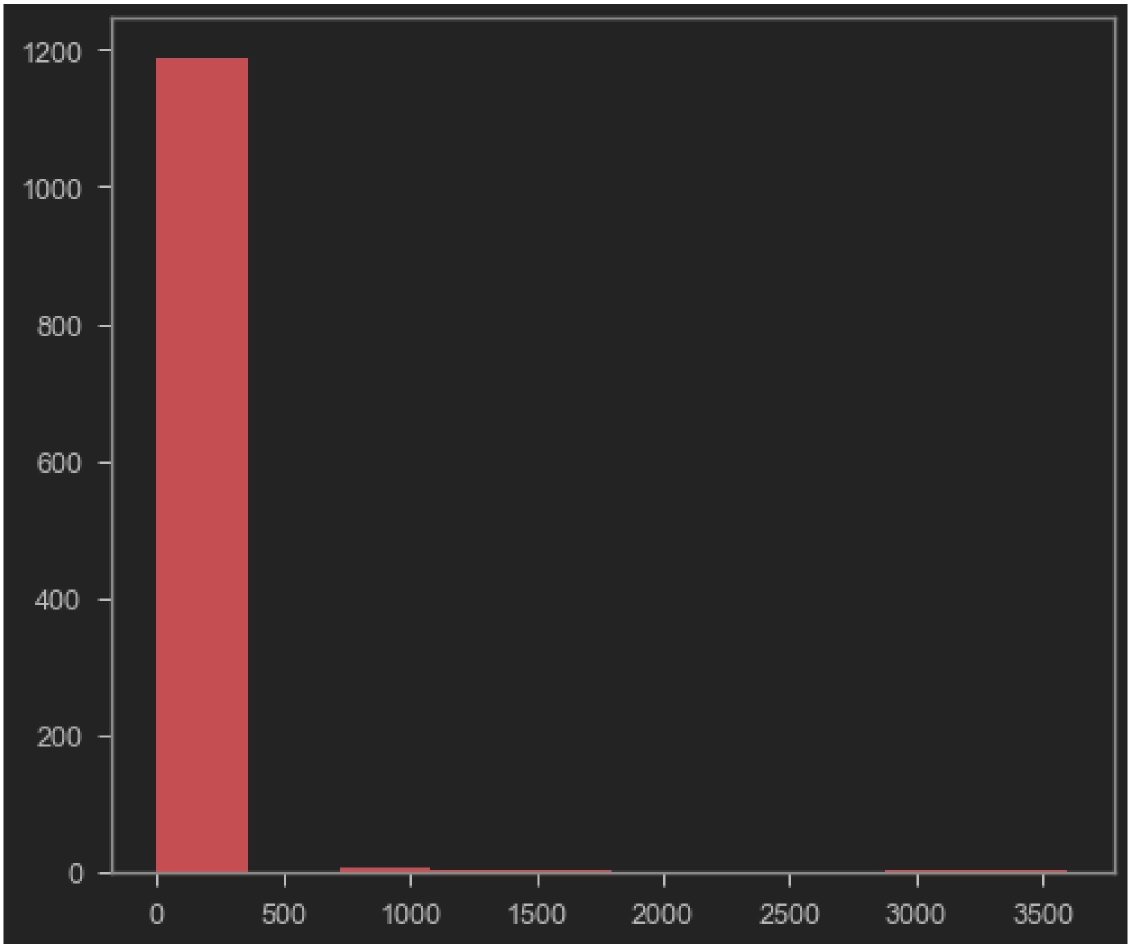
```
             data=df_eda,
             aspect=2,
             height=6,
             palette=cust_pal)
plt.title(f'wip count by performance\n', size=18, weight=4)
plt.show()
```



Same pattern, low wip does not necessarily mean a good workday. Some leftover work for the next day can mean that there is a greater chance of meeting that days goal.

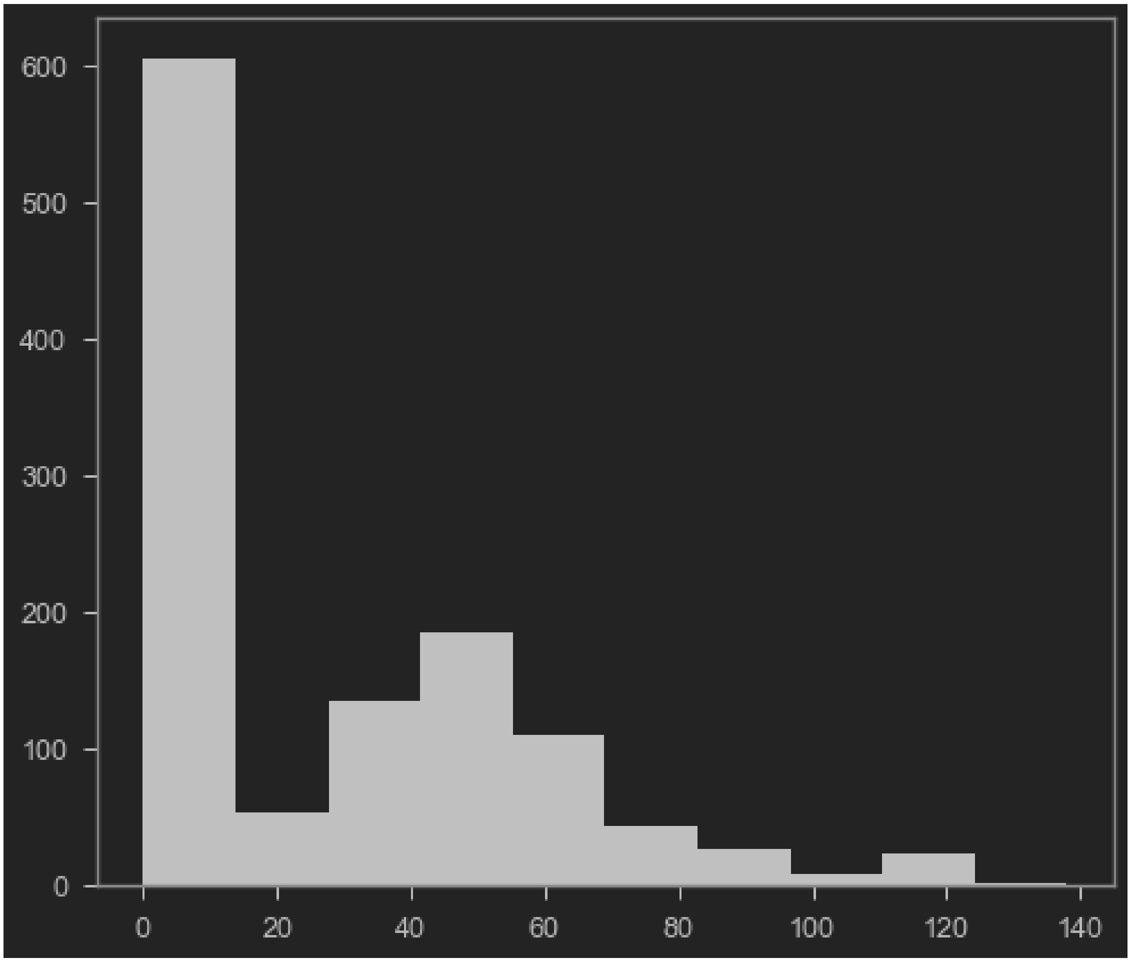
incentive on performance

[36]:
```
df_eda['incentive'].hist(color='r',grid=False);
```



`incentive` distribution is highly skewed. lets slice by 500 BDT.

[37]:
```
df_eda[df_eda['incentive']<500]['incentive'].hist(color='silver',grid=False);
```



Most of the day there is no incentive payment.

[38]:
```
print(
    f"""`incentive` and `performance` have a {df_eda[df_eda['incentive']<500]
.corr()['incentive']['performance'].round(4)} correlaiton"""
)
```

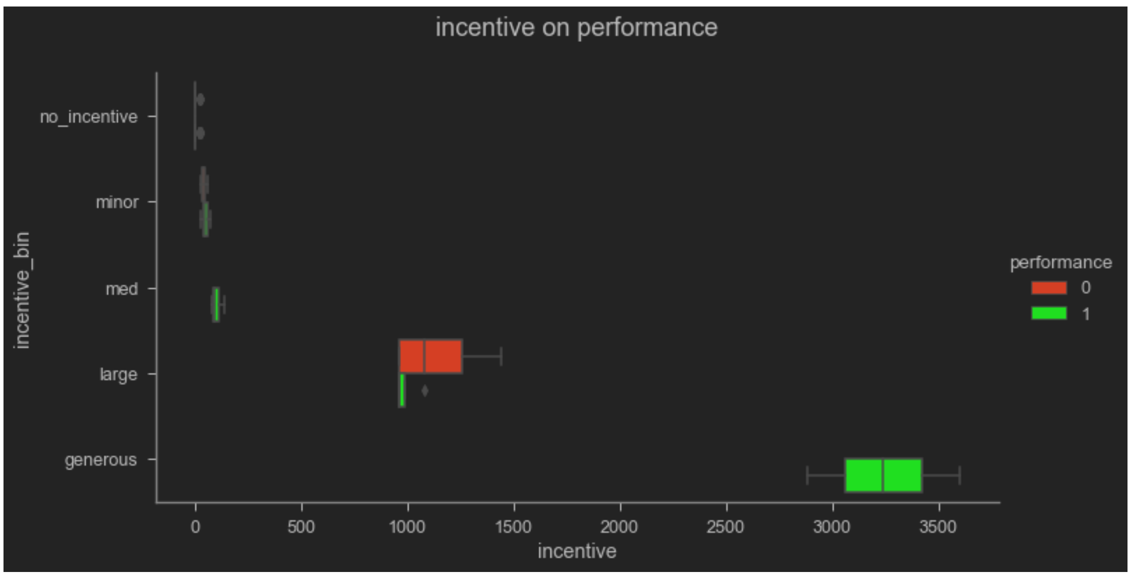
```
`incentive` and `performance` have a 0.4001 correlaiton
```

A word of caution, high correlation does not necessarily mean causation.

[39]:
```
# empty list to hold data
lst = []
# preparing data of incentive_bin distribution
for i in zip(df_eda.incentive_bin.cat.categories,
             df_eda.groupby('incentive_bin')['incentive'].agg('min'),
             df_eda.groupby('incentive_bin')['incentive'].agg('max')):
    temp_dict = {'category': i[0], 'min_value': i[1], 'max_value': i[2]}
    lst.append(temp_dict)
```

```
display(
    pd.DataFrame(lst).set_index('category').style.format(
        "{:.0f}").set_properties(**{'color': 'lawngreen'}))
sns.catplot(y="incentive_bin",
            x='incentive',
            kind="box",
            hue="performance",
            data=df_eda,
            aspect=2,
            palette=cust_pal)
plt.title(f'incentive on performance\n', size=18, weight=4)
plt.show()
```

```
<pandas.io.formats.style.Styler at 0x2b3073ec580>
```



After binning it can be seen that at higher incentive the performance is better, as no goal unmet at generous incentive.

5.2.9 preparing data for model

[40]:
```
print(f"""numeric cols: {df.select_dtypes('number').columns.tolist()}
categorical cols: {df.select_dtypes('category').columns.tolist()}""")
```

```
numeric cols: ['targeted_productivity', 'smv', 'wip', 'over_time', 'incentive',
'idle_time', 'idle_men', 'no_of_style_change', 'no_of_workers', 'performance']
categorical cols: ['quarter', 'department', 'day', 'team']
```

split using sklearn I am using train-test split approach here. Other option is to use train-validation-test data split approach. As the data set is relatively small, the later approach makes my train data have fewer samples to train on. This is a real issue for model performance for some of the models used. They perform better with more train data.

[41]:
```
X = df.drop(columns='performance').copy()
y = df['performance'].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25)
```

[42]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[42]:
```
((897, 13), (897,), (300, 13), (300,))
```

[43]:
```
print(f"""Class balance y_train:
{y_train.value_counts(1)}
""")
print(f"""Class balance y_test:
{y_test.value_counts(1)}
""")
```

```
Class balance y_train:
1    0.729097
0    0.270903
Name: performance, dtype: float64

Class balance y_test:
1    0.736667
0    0.263333
Name: performance, dtype: float64
```

Distribution of target class is somewhat consistent. Can be re-run for different distribution. But this is not necessary as I am tackling class imbalance issue with `SMOTENC`.

Addressing class imbalance using `SMOTENC`

[44]:
```
# keeping a class imbalanced dataset set for evaluation of preocess
X_imba, y_imba = X.copy(), y.copy()
X_imba.shape, y_imba.shape
```

[44]:
```
((1197, 13), (1197,))
```

[45]:
```
# creating a copy just to be safe.
XX = X.copy()
yy = y.copy()

# first four features are categorical;
# in the original paper (Rahim, 2021) attached to this dataset also
# considered `team` as categorical feature.
smotenc_features=[True]*4+[False]*9
# initialize SMOTENC
```

```
oversampling = SMOTENC(categorical_features=smotenc_features,n_jobs=-1)
# fiting
XX_oversampled, yy_oversampled = oversampling.fit_sample(XX,yy)
# updating dataset
X_train, y_train = XX_oversampled.copy(), yy_oversampled.copy()
X_train.shape, y_train.shape
```

[45]: ((1750, 13), (1750,))

OHE using pandas

[46]:
```
# as a reference point for pipelining process validation
pd.get_dummies(X_train).shape, pd.get_dummies(X_test).shape
```

[46]: ((1750, 33), (300, 33))

Pipelining

[47]:
```
# testing pipeline. for each model I shall use slightly diffrent pipeline.
_test_pipe_train, _test_pipe_test = fun.dataset_preprocessing_pipeline(
    X_train, X_test, scaler=fun.RobustScaler(), drop='first')
display(_test_pipe_train.describe().T.round(2),
        _test_pipe_test.describe().T.round(2))
```

	count	mean	std	min	25%	50%	75%	max
targeted_productivity	1750.0	-0.20	0.89	-4.00	-0.50	0.0	0.50	0.50
smv	1750.0	0.14	0.59	-0.44	-0.38	0.0	0.62	2.22
wip	1750.0	0.44	1.29	-0.13	-0.13	0.0	0.87	22.88
over_time	1750.0	0.15	0.59	-0.60	-0.38	0.0	0.62	4.04
incentive	1750.0	0.76	3.57	0.00	0.00	0.0	1.00	90.00
idle_time	1750.0	0.98	14.79	0.00	0.00	0.0	0.00	300.00
idle_men	1750.0	0.57	3.66	0.00	0.00	0.0	0.00	45.00
no_of_style_change	1750.0	0.13	0.39	0.00	0.00	0.0	0.00	2.00
no_of_workers	1750.0	0.01	0.46	-0.59	-0.47	0.0	0.53	1.18
quarter_Quarter2	1750.0	0.27	0.45	0.00	0.00	0.0	1.00	1.00
quarter_Quarter3	1750.0	0.19	0.39	0.00	0.00	0.0	0.00	1.00
quarter_Quarter4	1750.0	0.24	0.43	0.00	0.00	0.0	0.00	1.00
department_sweing	1750.0	0.51	0.50	0.00	0.00	1.0	1.00	1.00
day_Saturday	1750.0	0.15	0.35	0.00	0.00	0.0	0.00	1.00
day_Sunday	1750.0	0.18	0.39	0.00	0.00	0.0	0.00	1.00
day_Thursday	1750.0	0.18	0.39	0.00	0.00	0.0	0.00	1.00
day_Tuesday	1750.0	0.16	0.37	0.00	0.00	0.0	0.00	1.00
day_Wednesday	1750.0	0.17	0.37	0.00	0.00	0.0	0.00	1.00
team_2	1750.0	0.09	0.29	0.00	0.00	0.0	0.00	1.00
team_3	1750.0	0.06	0.23	0.00	0.00	0.0	0.00	1.00
team_4	1750.0	0.08	0.27	0.00	0.00	0.0	0.00	1.00
team_5	1750.0	0.09	0.29	0.00	0.00	0.0	0.00	1.00
team_6	1750.0	0.09	0.28	0.00	0.00	0.0	0.00	1.00
team_7	1750.0	0.10	0.30	0.00	0.00	0.0	0.00	1.00

team_8	1750.0	0.10	0.30	0.00	0.00	0.0	0.00	1.00
team_9	1750.0	0.09	0.29	0.00	0.00	0.0	0.00	1.00
team_10	1750.0	0.08	0.28	0.00	0.00	0.0	0.00	1.00
team_11	1750.0	0.07	0.26	0.00	0.00	0.0	0.00	1.00
team_12	1750.0	0.07	0.25	0.00	0.00	0.0	0.00	1.00

	count	mean	std	min	25%	50%	75%	max
targeted_productivity	300.0	-0.27	0.96	-4.00	-0.50	0.00	0.50	0.50
smv	300.0	0.20	0.56	-0.44	-0.38	0.20	0.62	2.03
wip	300.0	0.59	1.62	-0.13	-0.13	0.52	0.96	21.16
over_time	300.0	0.26	0.61	-0.60	-0.34	0.14	0.65	2.11
incentive	300.0	1.07	5.43	0.00	0.00	0.57	1.25	90.00
idle_time	300.0	0.55	8.67	0.00	0.00	0.00	0.00	150.00
idle_men	300.0	0.27	2.40	0.00	0.00	0.00	0.00	30.00
no_of_style_change	300.0	0.15	0.43	0.00	0.00	0.00	0.00	2.00
no_of_workers	300.0	0.09	0.44	-0.59	-0.43	0.06	0.53	0.59
quarter_Quarter2	300.0	0.28	0.45	0.00	0.00	0.00	1.00	1.00
quarter_Quarter3	300.0	0.20	0.40	0.00	0.00	0.00	0.00	1.00
quarter_Quarter4	300.0	0.23	0.42	0.00	0.00	0.00	0.00	1.00
department_sweing	300.0	0.60	0.49	0.00	0.00	1.00	1.00	1.00
day_Saturday	300.0	0.15	0.35	0.00	0.00	0.00	0.00	1.00
day_Sunday	300.0	0.18	0.38	0.00	0.00	0.00	0.00	1.00
day_Thursday	300.0	0.16	0.36	0.00	0.00	0.00	0.00	1.00
day_Tuesday	300.0	0.18	0.39	0.00	0.00	0.00	0.00	1.00
day_Wednesday	300.0	0.16	0.37	0.00	0.00	0.00	0.00	1.00
team_2	300.0	0.11	0.32	0.00	0.00	0.00	0.00	1.00
team_3	300.0	0.08	0.28	0.00	0.00	0.00	0.00	1.00
team_4	300.0	0.07	0.25	0.00	0.00	0.00	0.00	1.00
team_5	300.0	0.07	0.26	0.00	0.00	0.00	0.00	1.00
team_6	300.0	0.10	0.30	0.00	0.00	0.00	0.00	1.00
team_7	300.0	0.09	0.29	0.00	0.00	0.00	0.00	1.00
team_8	300.0	0.06	0.23	0.00	0.00	0.00	0.00	1.00
team_9	300.0	0.11	0.31	0.00	0.00	0.00	0.00	1.00
team_10	300.0	0.07	0.26	0.00	0.00	0.00	0.00	1.00
team_11	300.0	0.06	0.24	0.00	0.00	0.00	0.00	1.00
team_12	300.0	0.09	0.29	0.00	0.00	0.00	0.00	1.00

pipeline is working!

6 MODEL

6.1 dummy model

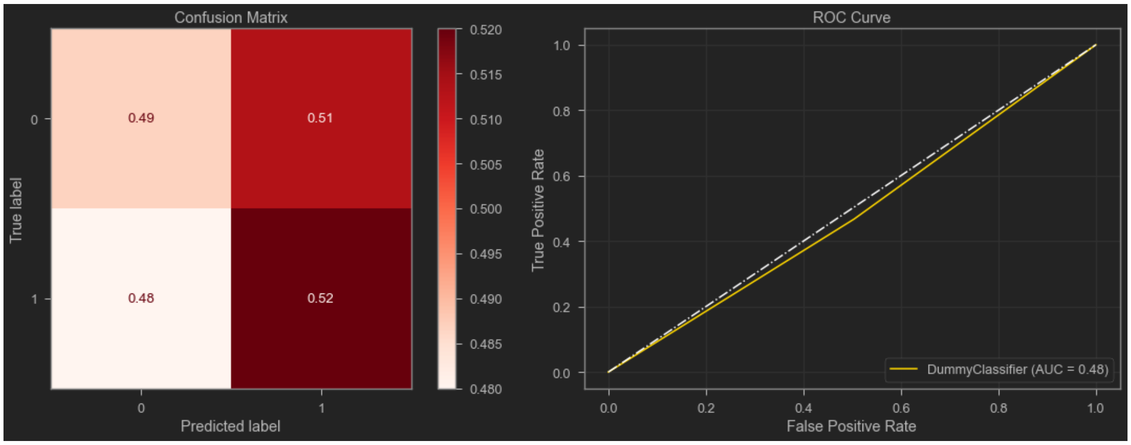
[48]:

```
# SMOTENC'ed, StandardScaled and OHE'ed data
X_train_dummy, X_test_dummy = fun.dataset_preprocessing_pipeline(
    X_train, X_test)

dummy_classifier = DummyClassifier(strategy='stratified')
```

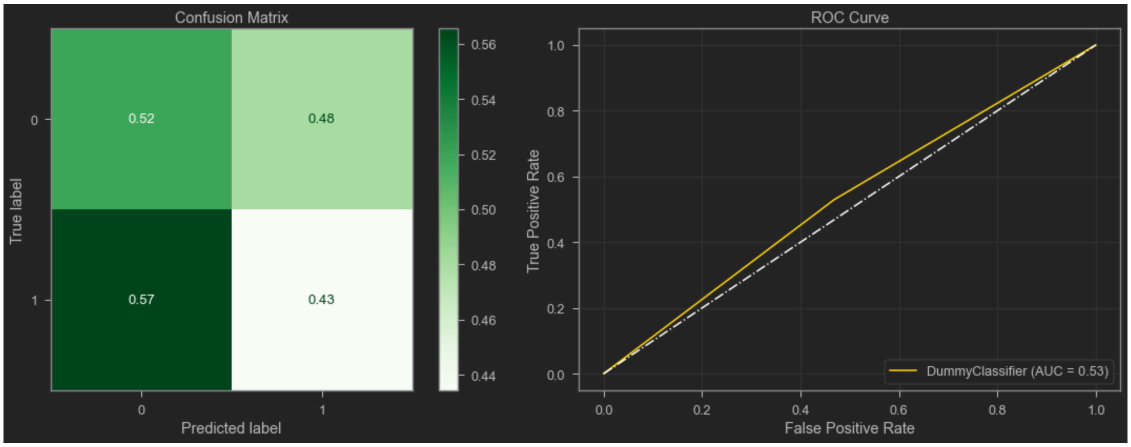
```python
print(f"""Class balance y_train:
{y_train.value_counts(1)}
""")
print(f"""Class balance y_test:
{y_test.value_counts(1)}
""")
print(f"""{'-'*30}""")
fun.model_report(dummy_classifier,
                 X_train=X_train_dummy,
                 y_train=y_train,
                 X_test=X_test_dummy,
                 y_test=y_test,
                 cmap=['Reds', 'Greens'],
                 show_train_report=True)
```

```
Class balance y_train:
1    0.5
0    0.5
Name: performance, dtype: float64

Class balance y_test:
1    0.736667
0    0.263333
Name: performance, dtype: float64

------------------------------

<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.4931
Test accuracy score: 0.5
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on train data of:
        DummyClassifier(strategy='stratified')
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.52      0.53      0.52       875
           1       0.52      0.52      0.52       875

    accuracy                           0.52      1750
   macro avg       0.52      0.52      0.52      1750
```

```
weighted avg       0.52      0.52      0.52      1750

************************************************************
```




```
================================================================================
================================================================================
==========
************************************************************
Classification report on test data of:
    DummyClassifier(strategy='stratified')
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.28      0.48      0.36        79
           1       0.75      0.56      0.64       221

    accuracy                           0.54       300
   macro avg       0.52      0.52      0.50       300
weighted avg       0.63      0.54      0.57       300

************************************************************
```



This is a worthless model. The f1 score is low, model accuracy is .5. This is not even better than flipping a coin to predict, which should be correct at random.

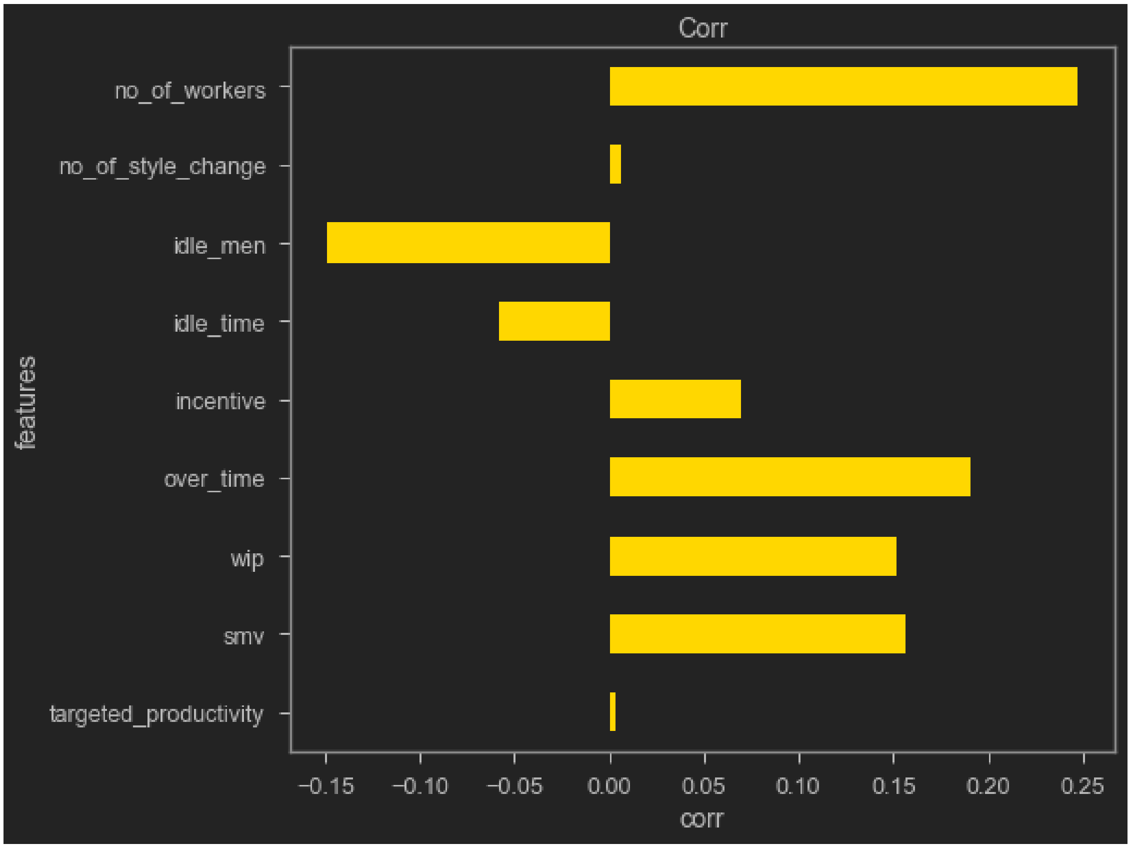
NOTE: there is no random seed for anything. For later run the sample drawn from train-test split can have a different subset, thus making most of my model evaluation commentary invalid. This is true for all of my modeling process.

6.2 logistic regression

6.2.1 filter with Pearson corr

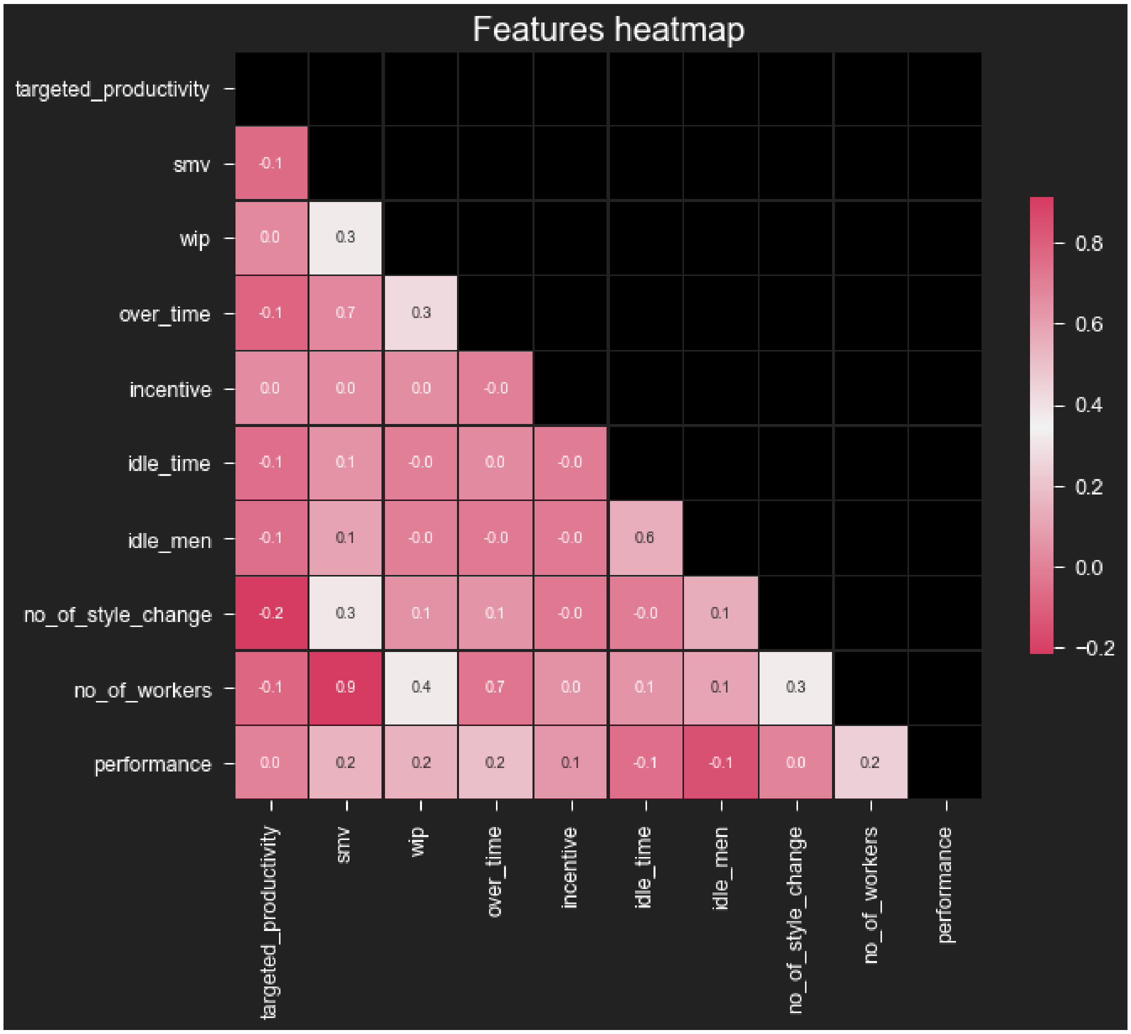
[49]:
```
df.corr()['performance'][:-1].plot(kind='barh',color='gold')
plt.title('Corr')
plt.xlabel('corr')
plt.ylabel('features')
```

[49]:
```
Text(0, 0.5, 'features')
```



Most of them are correlated with the target except `no_of_style_change` and `targeted_productivity`.

[49]:
```
fun.heatmap_of_features(df);
```



- No significant correlation is detected except `no_of_worker` and `smv`.
- `overtime` and `no_of_worker` is correlated.

[50]:
```
display(fun.top_correlated_features(df))
print(f"""Features should be dropped: {
                                    fun.drop_features_based_on_correlation(
                                    df, threshold=0.75)
                                        }""")
```

```
           feature_combo  correlation
0  smv and no_of_workers     0.912176

Features should be dropped: {'no_of_workers'}
```

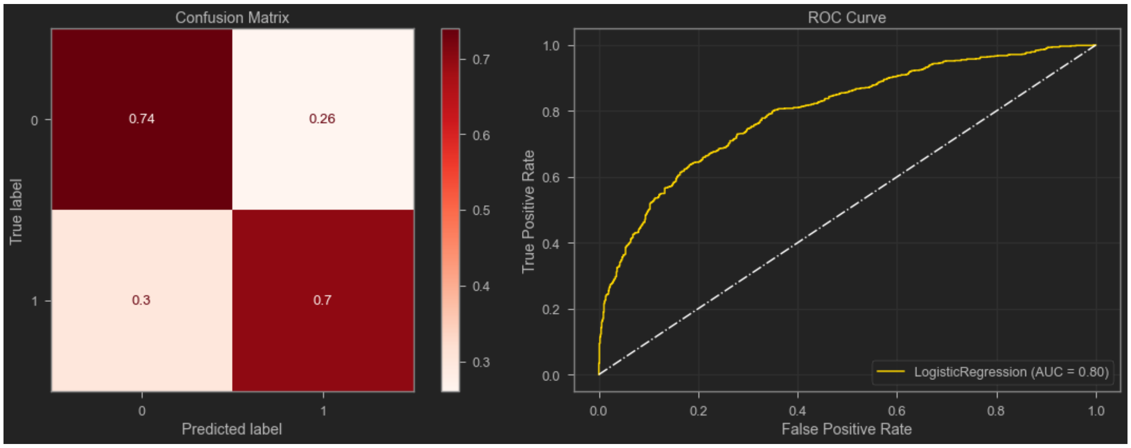
[51]:
```
# droping from train and test data
X_train_dropped_ = X_train.drop('no_of_workers',axis=1)
X_test_dropped_ = X_test.drop('no_of_workers',axis=1)
```

```python
# SMOTENC'ed, StandardScaled, correlated feature dropped and OHE'ed data
X_train_log_reg, X_test_log_reg = fun.dataset_preprocessing_pipeline(
    X_train_dropped_, X_test_dropped_, drop='first')
```

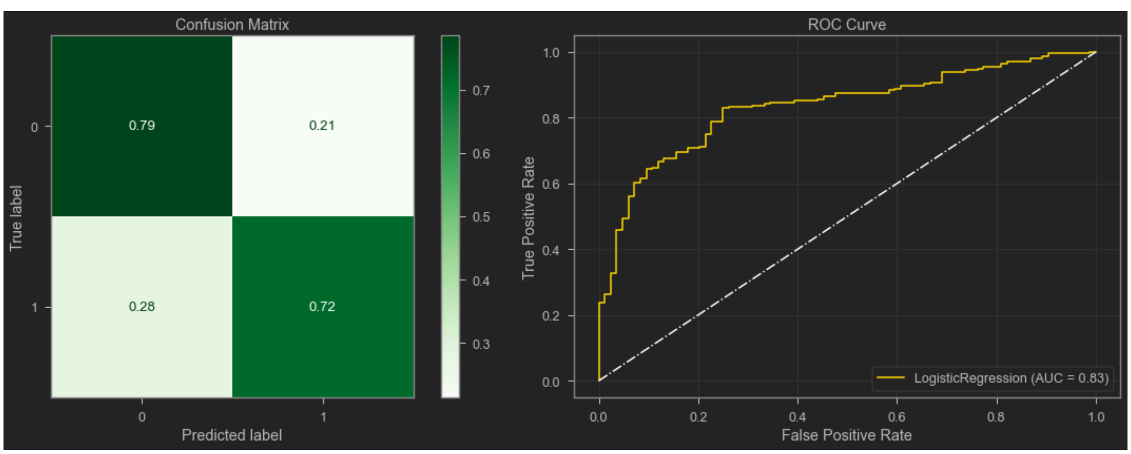
6.2.2 logistic regression classifier

[52]:
```python
# logistic regression classifier
logreg = LogisticRegression(C=1e5, max_iter=1000, class_weight='balanced')
# score of logistic regression classifier
fun.model_report(logreg,
             X_train=X_train_log_reg,
             y_train=y_train,
             X_test=X_test_log_reg,
             y_test=y_test,show_train_report=True)
```

```
<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.7194
Test accuracy score: 0.7367
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on train data of:
        LogisticRegression(C=100000.0, class_weight='balanced', max_iter=1000)
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.71      0.74      0.72       875
           1       0.73      0.70      0.71       875

    accuracy                           0.72      1750
   macro avg       0.72      0.72      0.72      1750
weighted avg       0.72      0.72      0.72      1750

************************************************************
```

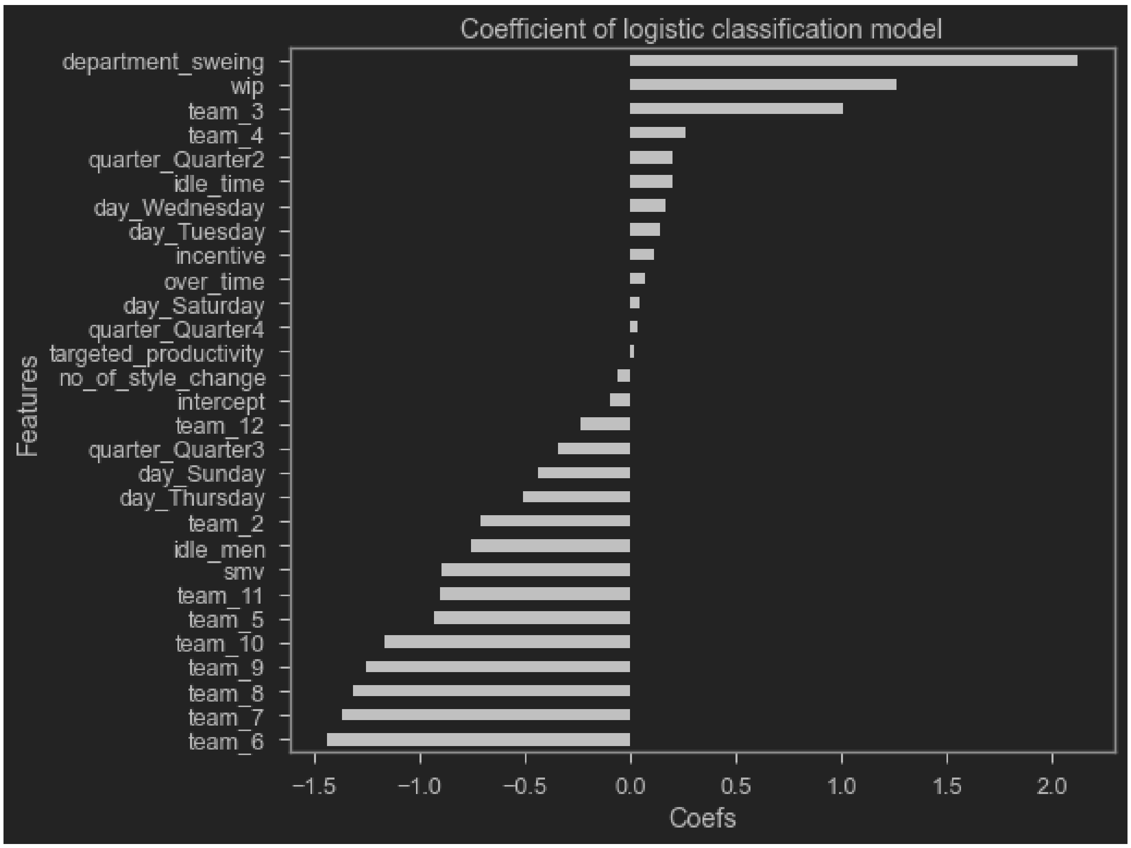



```
================================================================================
================================================================================
==========
************************************************************
Classification report on test data of:
    LogisticRegression(C=100000.0, class_weight='balanced', max_iter=1000)
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.52      0.79      0.63        84
           1       0.90      0.72      0.80       216

    accuracy                           0.74       300
   macro avg       0.71      0.75      0.71       300
weighted avg       0.79      0.74      0.75       300

************************************************************
```



Overall good performance. Can detect majority of true negatives and positives, with good recall and and f1. ROC curve also looks good.

[53]:
```
fun.coefficients_of_model_binary(
    logreg, X_train_log_reg, log_scale=True).sort_values().plot(kind='barh',
                                                                color='silver')
plt.title('Coefficient of logistic classification model')
plt.xlabel('Coefs')
plt.ylabel('Features');
```



By looking at the coefs of the model, I can have a idea of feature importance and their impact on the prediction. `department_sweing` and `wip` has highest coef. and some teams are under performing, but teams of both department share number as identifier.

6.2.3 grid search with Cross Validation

[54]:
```
logreg_gs = LogisticRegression(max_iter=1e4,
                               class_weight='balanced',
                               n_jobs=-1)
params = {
    'C': [.1, 1, 10, 100, 10000, 1e6, 1e12],
```

```
    'tol': [0.0001, 0.001, 0.01, .1],
    'penalty': ['l1', 'l2', 'elasticnet', None],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}
gridsearch_logreg = GridSearchCV(estimator=logreg_gs,
                                 param_grid=params,
                                 n_jobs=-1,
                                 scoring='precision')
gridsearch_logreg
```

[54]:
```
GridSearchCV(estimator=LogisticRegression(class_weight='balanced',
                                          max_iter=10000.0, n_jobs=-1),
             n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100, 10000, 1000000.0,
                               1000000000000.0],
                         'penalty': ['l1', 'l2', 'elasticnet', None],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag',
                                    'saga'],
                         'tol': [0.0001, 0.001, 0.01, 0.1]},
             scoring='precision')
```

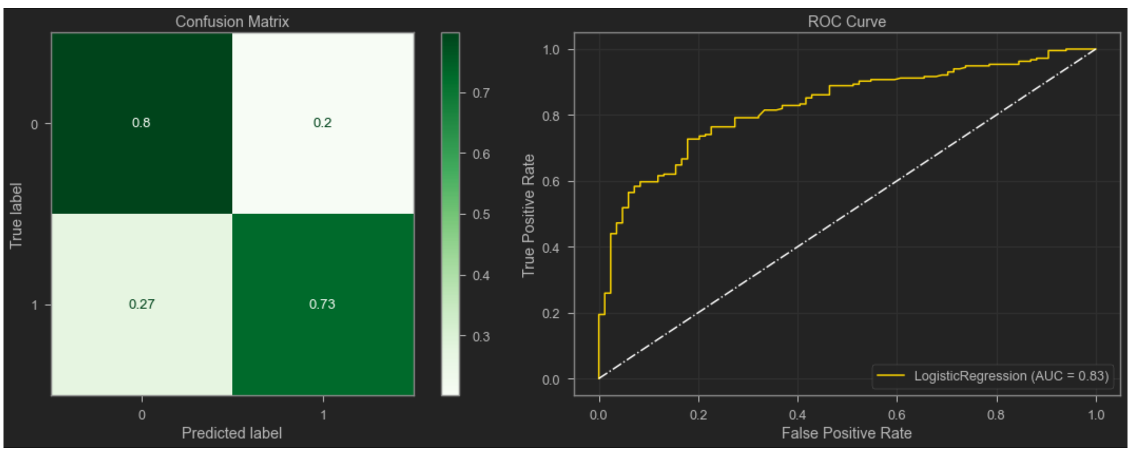
[55]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_logreg.fit(X_train_log_reg, y_train)
print(f"Best Parameters by gridsearch:\t{gridsearch_logreg.best_params_}")
print(f"Best Estimator by gridsearch:\t{gridsearch_logreg.best_estimator_}")
```

```
Best Parameters by gridsearch:  {'C': 0.1, 'penalty': 'l1', 'solver': 'saga',
'tol': 0.0001}
Best Estimator by gridsearch:   LogisticRegression(C=0.1,
class_weight='balanced', max_iter=10000.0, n_jobs=-1,
                   penalty='l1', solver='saga')
```

[56]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    logreg_gs_best = gridsearch_logreg.best_estimator_
    fun.model_report(logreg_gs_best, X_train_log_reg, y_train, X_test_log_reg,
                 y_test)
```

```
<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.728
Test accuracy score: 0.7467
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********
```

```
************************************************************
Classification report on test data of:
    LogisticRegression(C=0.1, class_weight='balanced', max_iter=10000.0,
n_jobs=-1,
                   penalty='l1', solver='saga')
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.53      0.80      0.64        84
           1       0.90      0.73      0.81       216

    accuracy                           0.75       300
   macro avg       0.72      0.76      0.72       300
weighted avg       0.80      0.75      0.76       300

************************************************************
```



Very minimal improvement overall.

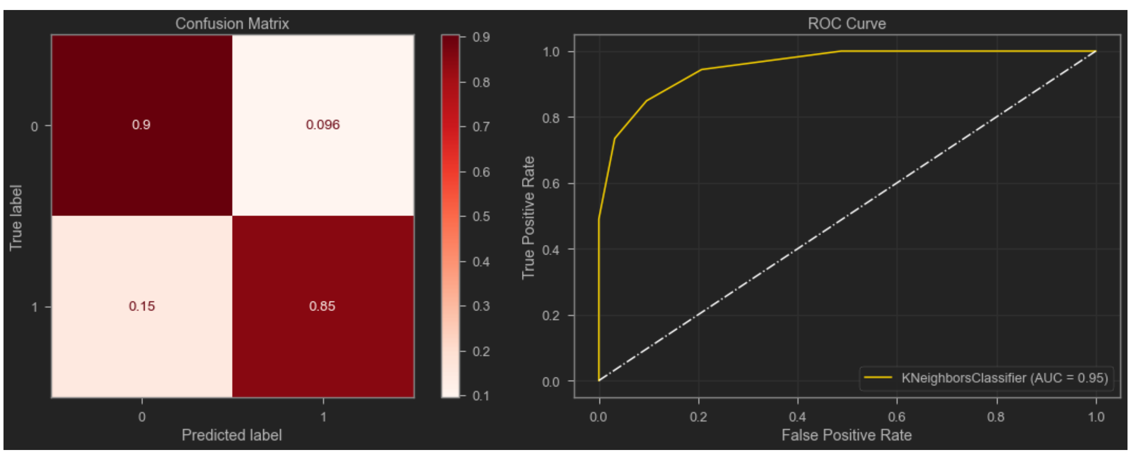
At this point I can tackle outliers by removing them based on Z-score or IQR or other method; and considering scaling options can be done here. But chance of data loss is higher. Moreover, disruption of distribution of data is required for this process. Moving on to next type of model.

6.3 KNN Clustering

[57]:
```
X_train_knn, X_test_knn = fun.dataset_preprocessing_pipeline(X_train, X_test)

knn = KNeighborsClassifier()
fun.model_report(knn,
                 X_train_knn,
                 y_train,
                 X_test_knn,
```

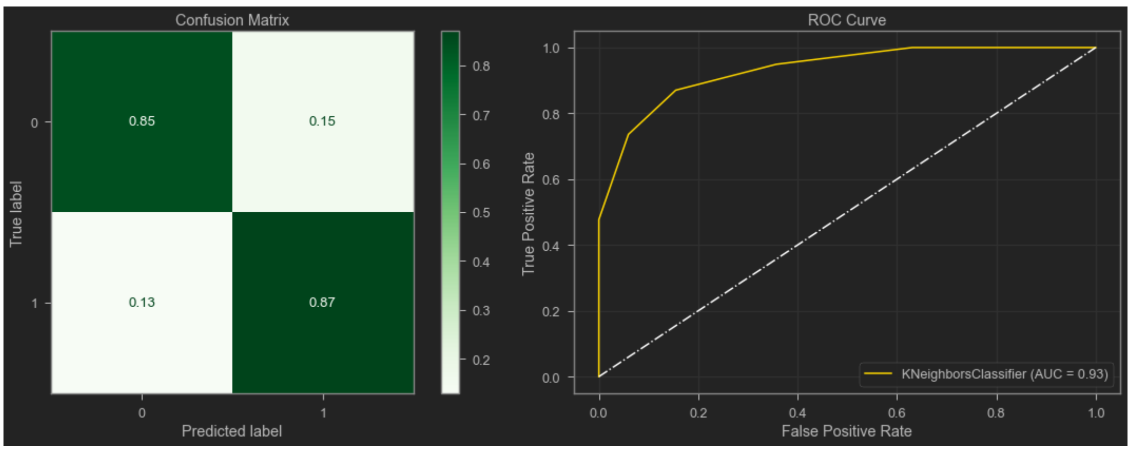
```
                  y_test,
                  show_train_report=True)
```

```
<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.8766
Test accuracy score: 0.8633
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on train data of:
        KNeighborsClassifier()
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       875
           1       0.90      0.85      0.87       875

    accuracy                           0.88      1750
   macro avg       0.88      0.88      0.88      1750
weighted avg       0.88      0.88      0.88      1750

************************************************************
```




```
================================================================================
================================================================================
==========
************************************************************
```

```
Classification report on test data of:
    KNeighborsClassifier()
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.72      0.85      0.78        84
           1       0.94      0.87      0.90       216

    accuracy                           0.86       300
   macro avg       0.83      0.86      0.84       300
weighted avg       0.87      0.86      0.87       300

************************************************************
```



Way better performance than previous model. True negative and positives are better, all the metrics are looking good. ROC curve is improved. But this can be better better by some hyperparameter tuning.

6.3.1 grid search with Cross Validation

[58]:
```
knn_gs = KNeighborsClassifier(n_jobs=-1)
params = {
    'n_neighbors': list(range(1, 31, 2)),
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2, 2.5, 3, 4],
    'leaf_size': [30, 40],
#     'metric': ['minkowski', 'manhattan', 'euclidean']
}
gridsearch_knn = GridSearchCV(estimator=knn_gs,
                              param_grid=params,
                              n_jobs=-1,
```

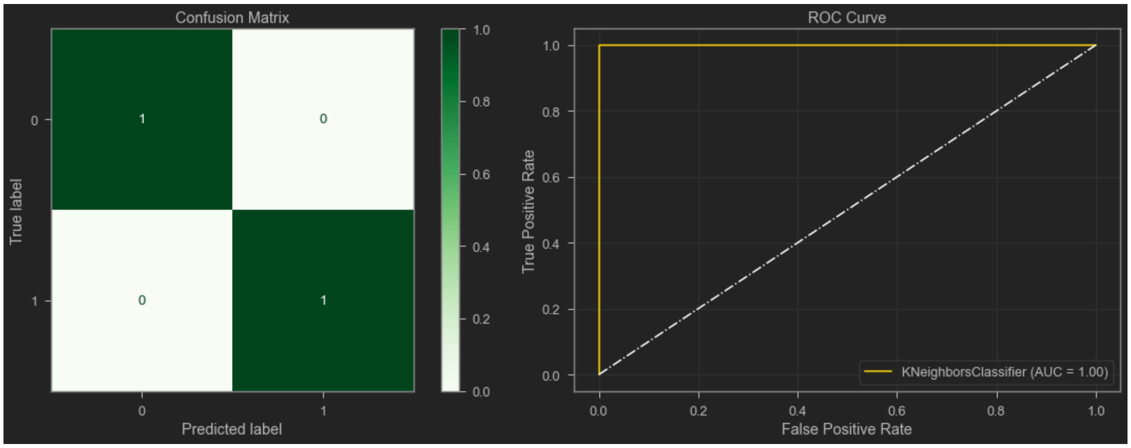
```
                                 scoring='precision',return_train_score=True)
gridsearch_knn
```

[58]:
```
GridSearchCV(estimator=KNeighborsClassifier(n_jobs=-1), n_jobs=-1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'leaf_size': [30, 40],
                         'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21,
                                         23, 25, 27, 29],
                         'p': [1, 2, 2.5, 3, 4],
                         'weights': ['uniform', 'distance']},
             return_train_score=True, scoring='precision')
```

[59]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_knn.fit(X_train_knn, y_train)
print(f"Best Parameters by gridsearch:\t{gridsearch_knn.best_params_}")
print(f"Best Estimator by gridsearch:\t{gridsearch_knn.best_estimator_}")

knn_gs_best = gridsearch_knn.best_estimator_
fun.model_report(knn_gs_best, X_train_knn, y_train, X_test_knn,
             y_test)
```

```
Best Parameters by gridsearch:  {'algorithm': 'auto', 'leaf_size': 30,
'n_neighbors': 17, 'p': 1, 'weights': 'distance'}
Best Estimator by gridsearch:   KNeighborsClassifier(n_jobs=-1, n_neighbors=17,
p=1, weights='distance')

<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.9994
Test accuracy score: 1.0
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on test data of:
    KNeighborsClassifier(n_jobs=-1, n_neighbors=17, p=1, weights='distance')
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        84
           1       1.00      1.00      1.00       216

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
```

```
weighted avg       1.00      1.00      1.00       300

************************************************************
```



Perfect result. It can predict with certainty. It has all the scores perfect across all the metrics and ROC curve is perfect.

[60]:
```
knn_gs_best.get_params()
```

[60]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': -1,
 'n_neighbors': 17,
 'p': 1,
 'weights': 'distance'}
```

These are the best parameters.

[61]:
```
pd.DataFrame(
    gridsearch_knn.cv_results_).sort_values(by='rank_test_score')[:10].T
```

[61]:
```
                                                             81  …
971
mean_fit_time                                         0.0498116  …
0.0202043
std_fit_time                                          0.0332004  …
0.0324251
mean_score_time                                        0.266023  …
0.276663
std_score_time                                        0.0821157  …
0.140997
```

```
param_algorithm                                              auto  …
brute
param_leaf_size                                                30  …
30
param_n_neighbors                                              17  …
15
param_p                                                         1  …
1
param_weights                                            distance  …
distance
params              {'algorithm': 'auto', 'leaf_size': 30, 'n_neig…  …
{'algorithm': 'brute', 'leaf_size': 30, 'n_nei…
split0_test_score                                        0.709091  …
0.714286
split1_test_score                                           0.768  …
0.760331
split2_test_score                                        0.942623  …
0.942149
split3_test_score                                        0.858491  …
0.853211
split4_test_score                                        0.868217  …
0.858268
mean_test_score                                          0.829284  …
0.825649
std_test_score                                          0.0817516  …
0.0800801
rank_test_score                                                 1  …
9
split0_train_score                                              1  …
1
split1_train_score                                              1  …
1
split2_train_score                                              1  …
1
split3_train_score                                              1  …
1
split4_train_score                                              1  …
1
mean_train_score                                                1  …
1
std_train_score                                                 0  …
0

[25 rows x 10 columns]
```

Here is a sneak peek of all the models created by grid search. It found 8 perfect models.

6.4 ensemble methods

[54]:
```
X_train_ensbl, X_test_ensbl = fun.dataset_preprocessing_pipeline(
    X_train, X_test)
```

6.4.1 Random Forest™

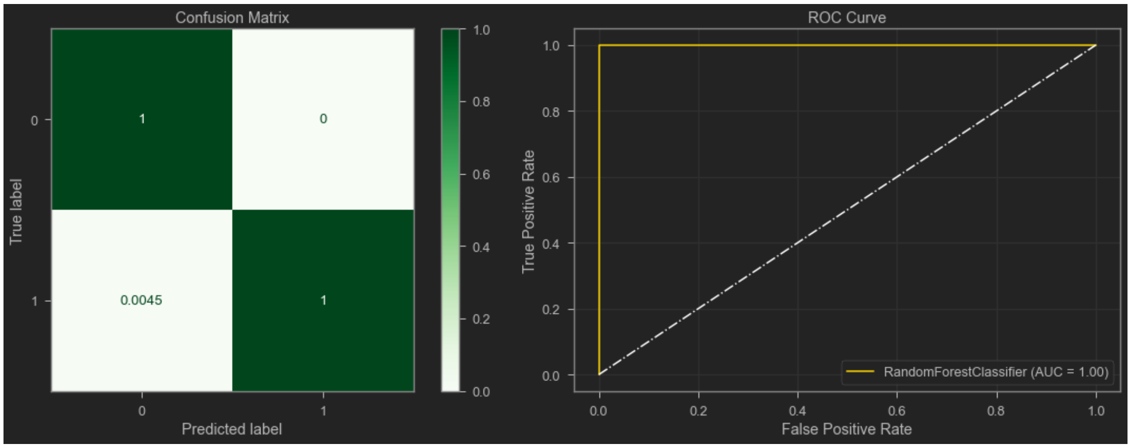
`Random Forest` is a trademark of Leo Breiman and Adele Cutler and is licensed exclusively to "Salford Systems", subsidiary of "Minitab, LLC", for the commercial release of the software. Random Forest A.K.A. random decision forests. This is one of the extensively used black-box models. KKN and RF can be both classified as weighted neighborhoods schemes. I am using scikit-learn's implementation of the concept.

RF generally requires less tuning for acceptable performance. Thus I am using random decision forest here, as I got a good result using KNN after some hyperparameter tuning via grid search with cross validation.

[55]:
```
rf_clf = RandomForestClassifier()
```

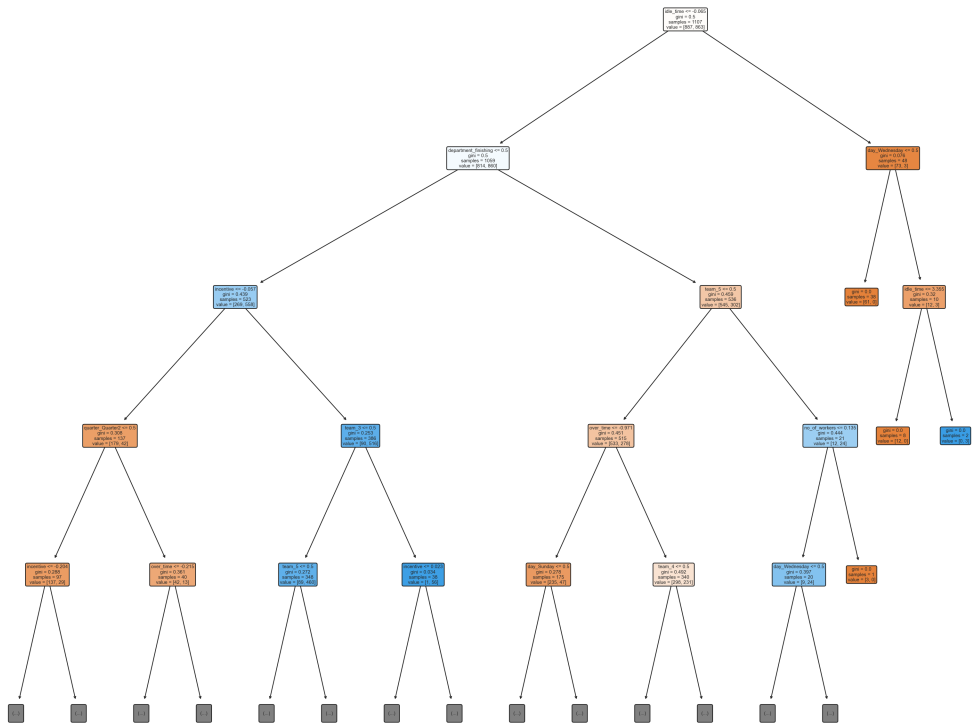
[56]:
```
fun.model_report(rf_clf, X_train_ensbl, y_train, X_test_ensbl,
             y_test)
```

```
<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.9994
Test accuracy score: 0.9967
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on test data of:
    RandomForestClassifier()
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        79
           1       1.00      1.00      1.00       221

    accuracy                           1.00       300
   macro avg       0.99      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300

************************************************************
```



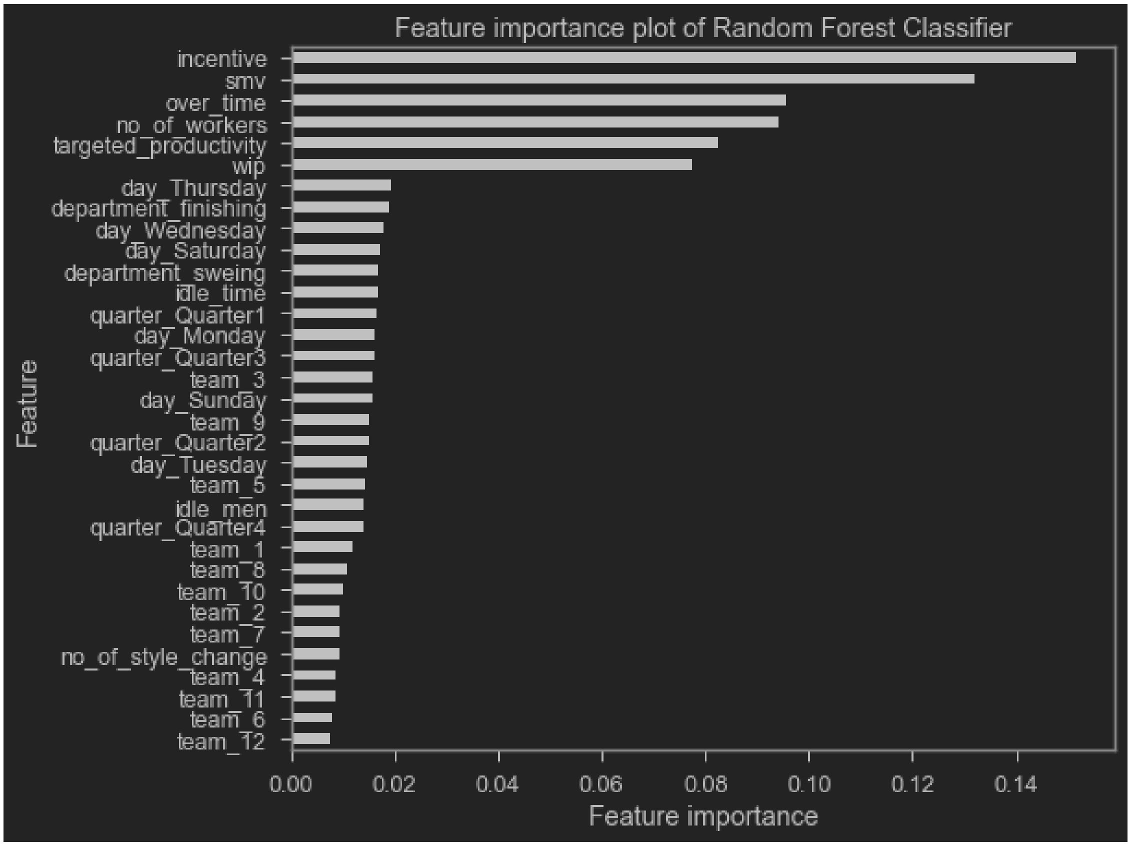
As expected I have the same level of performance with the out-of-the-box model without any tuning. Lets look at the first few nods of the tree of the 10th tree. I choose 10th at random. This output is not friendly to see in a notebook. A copy of this can be found at `'./saved_model/rf_clf_sample_4.pdf'` in side this repository as a pdf file.

[65]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(figsize=(6, 5), dpi=600)
    fig.tight_layout()
    tree.plot_tree(rf_clf.estimators_[10],
                   feature_names=X_train_ensbl.columns,
                   max_depth=4,
                   filled=True,
                   rounded=True)
    fig.savefig('./saved_model/rf_clf_sample_4.pdf',
                dpi=300,
                orientation='landscape',
                facecolor='white')
```



[66]:
```python
# used parameters for the classifier
print("Parameter used for the model:")
print(rf_clf.get_params())
# feature importance
pd.DataFrame(rf_clf.feature_importances_,
             index=X_test_ensbl.columns).sort_values(by=0).plot(kind='barh',
                                                                  legend='',
                                                                  color='silver')
plt.title('Feature importance plot of Random Forest Classifier')
plt.ylabel('Feature')
plt.xlabel('Feature importance');
```

```
Parameter used for the model:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None,
'verbose': 0, 'warm_start': False}
```



Most important features are, in descending order, `incentive`, `smv`, `overtime`, `no_of_workers`, `targeted_productivity` and so on. This means those features were often used for building the forests. And those features make real life sense also.

6.5 Selecting Best model

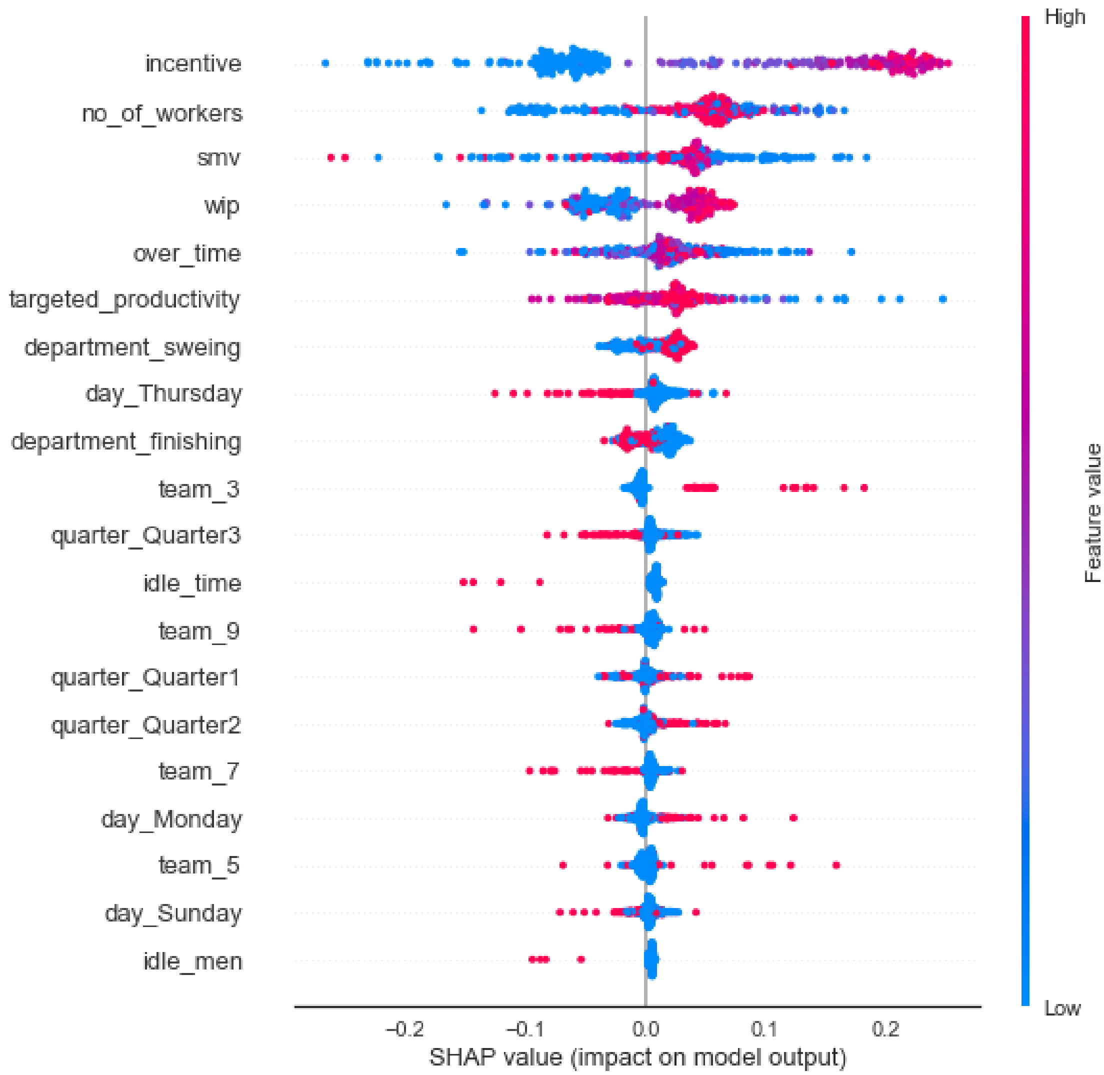
Random Forest model is the best one, this can achieve perfect prediction with minimal effort. This is `rf_clf` model object in this notebook.

7 INTERPRET

[58]:
```
# init shap
shap.initjs()
```

<IPython.core.display.HTML object>

[76]:
```
# explain all the predictions in the test set
explainer = shap.TreeExplainer(rf_clf)
shap_values = explainer.shap_values(X_test_ensbl)
with plt.style.context('seaborn-white'):
    shap.summary_plot(shap_values[1], X_test_ensbl)
```



This plot is showing contribution of each feature on a machine learning model prediction. This graph is for detecting goal met class. - Few explanations:

Features	Probability of goal met	Probability of goal not met
department_finishing	Lower Value	Higher Value
department_sweing	High Value	Lower Value
idle_men	Lower Value	
idle_time	Lower Value	
incentive	High Value	Lower Value
no_of_workers	Above Average	Below Average And High
over_time	Average	Below Average And High
smv	Above Average	Below Average And High
targeted_productivity	Lower Value	Higher Value
wip	High Value	Lower Value

Here weights explained in tabular form

[127]:
```
eli5.format_as_dataframe(eli5.explain_weights(
    rf_clf, feature_names=list(X_test_ensbl.columns)))
```

[127]:

	feature	weight	std
0	incentive	0.158867	0.068949
1	smv	0.124824	0.040667
2	no_of_workers	0.109437	0.040290
3	over_time	0.102029	0.025587
4	targeted_productivity	0.099527	0.027682
5	team	0.098398	0.020972
6	wip	0.079831	0.051646
7	quarter_Quarter1	0.018690	0.006853
8	department_finishing	0.017462	0.029487
9	department_sweing	0.017232	0.030081
10	day_Monday	0.016512	0.005913
11	quarter_Quarter2	0.016494	0.006474
12	day_Thursday	0.016414	0.007749
13	day_Wednesday	0.015862	0.006943
14	day_Saturday	0.015456	0.005627
15	quarter_Quarter3	0.015272	0.006772
16	idle_time	0.013611	0.014364
17	day_Sunday	0.013542	0.005258
18	day_Tuesday	0.013159	0.005338
19	quarter_Quarter4	0.012499	0.005511

[62]:
```
fun.javascript_formatter(shap.force_plot(explainer.expected_value[0],
                                         shap_values[0],
                                         feature_names=X_test_ensbl.columns).
                                         _repr_html_())
```

```
<IPython.core.display.HTML object>
```

This interactive plot has all the instances and their impact on the final model.

Peeking into one of the citizens of the random forest voters. I picked this at random using a random number generator.

[52]:
```
# x = np.random.randint(100)
x
```

[162]:
```
eli5.explain_prediction(
    rf_clf, X_test_ensbl.iloc[x], feature_names=list(X_test_ensbl.columns))
```

[162]:
```
Explanation(estimator='RandomForestClassifier()', description='\nFeatures with
largest coefficients.\n\nFeature weights are calculated by following decision
paths in trees\nof an ensemble (or a single tree for
DecisionTreeClassifier).\nEach node of the tree has an output score, and
```

contribution of a feature\non the decision path is how much the score changes
from parent to child.\nWeights of all features sum to the output score or proba
of the estimator.\n\nCaveats:\n1. Feature weights just show if the feature
contributed positively or\n   negatively to the final score, and does not show
how increasing or\n   decreasing the feature value will change the
prediction.\n2. In some cases, feature weight can be close to zero for an
important feature.\n   For example, in a single tree that computes XOR function,
the feature at the\n   top of the tree will have zero weight because expected
scores for both\n   branches are equal, so decision at the top feature does not
change the\n   expected score. For an ensemble predicting XOR functions it might
not be\n   a problem, but it is not reliable if most trees happen to choose the
same\n   feature at the top.\n', error=None, method='decision path',
is_regression=False, targets=[TargetExplanation(target=0,
feature_weights=FeatureWeights(pos=[FeatureWeight(feature='<BIAS>',
weight=0.5003314285714288, std=None, value=1.0), FeatureWeight(feature='smv',
weight=0.07076519085527301, std=None, value=-0.8577537234005602),
FeatureWeight(feature='incentive', weight=0.06011705649214153, std=None,
value=-0.2189406314710275), FeatureWeight(feature='quarter_Quarter1',
weight=0.0483561045227838, std=None, value=1.0),
FeatureWeight(feature='day_Monday', weight=0.0435149643326248, std=None,
value=0.0), FeatureWeight(feature='quarter_Quarter4',
weight=0.041273857748578016, std=None, value=0.0),
FeatureWeight(feature='day_Tuesday', weight=0.03849542581057023, std=None,
value=1.0), FeatureWeight(feature='wip', weight=0.038463571076397744, std=None,
value=-0.4418515571470569), FeatureWeight(feature='quarter_Quarter2',
weight=0.02821981430124622, std=None, value=0.0),
FeatureWeight(feature='day_Sunday', weight=0.025274536036780954, std=None,
value=0.0), FeatureWeight(feature='department_finishing',
weight=0.0111738669209691, std=None, value=1.0),
FeatureWeight(feature='department_sweing', weight=0.010145044419543132,
std=None, value=0.0), FeatureWeight(feature='quarter_Quarter5',
weight=0.00364026860903804, std=None, value=0.0),
FeatureWeight(feature='no_of_style_change', weight=0.0003035217769745219,
std=None, value=-0.3271940724868952)], neg=[FeatureWeight(feature='team',
weight=-0.09614332301151607, std=None, value=-0.8014936519070163),
FeatureWeight(feature='targeted_productivity', weight=-0.03282142213768466,
std=None, value=0.7741878270877041), FeatureWeight(feature='day_Thursday',
weight=-0.023811271305801594, std=None, value=0.0),
FeatureWeight(feature='quarter_Quarter3', weight=-0.021701270617635115,
std=None, value=0.0), FeatureWeight(feature='no_of_workers',
weight=-0.02105683638554456, std=None, value=-0.8734547046551392),
FeatureWeight(feature='day_Wednesday', weight=-0.013705187414853113, std=None,
value=0.0), FeatureWeight(feature='day_Saturday', weight=-0.003624187656538897,
std=None, value=0.0), FeatureWeight(feature='idle_men',
weight=-0.0030624971115985273, std=None, value=-0.1644966842882116),
FeatureWeight(feature='idle_time', weight=-0.002418086486213879, std=None,
value=-0.06540067580079735), FeatureWeight(feature='over_time',

```
weight=-0.0017305693469633321, std=None, value=-0.84378326060226)],
pos_remaining=0, neg_remaining=0), proba=0.7, score=None, weighted_spans=None,
heatmap=None)], feature_importances=None, decision_tree=None,
highlight_spaces=None, transition_features=None, image=None)
```

7.1 saving model

[66]:
```
# fun.save_model(rf_clf,custom_prefix='ta')
```

```
File saved: ta_RandomForestClassifier_5_22_2021_19_2_31.joblib
```

[67]:
```
lod_mod = joblib.load(
    './saved_model/ta_RandomForestClassifier_5_22_2021_19_2_31.joblib')
lod_mod
```

8 RECOMMENDATION & CONCLUSION

This model can be used with confidence for predicting employee performance. It can detect both True negatives and positives with high precision.

- Few insights where to focus
  - `incentive` is very important decider for performance.
  - tune optimal `no_of_workers` for better performance.

9 NEXT STEPS

- do a multi-class prediction by further binning of target.
- fit a model with entire data and prepare for production use.
- fine-tune functions for general use and add options for users.
- mend appendix contents

10 APPENDIX

10.1 all functions and imports from the `functions.py` and `packages.py`

[148]:
```
fun.show_py_file_content('./imports_and_functions/functions.py')
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder, RobustScaler, Qu
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from IPython.display import display, HTML, Markdown
from sklearn import metrics
from imblearn.over_sampling import SMOTE, SMOTENC
import joblib
```

```python
import time
### function name starting with "z_" are experimental and not fully tested ###
# Future plan: restructure functions to behave as attached to class using OOP#
# handle multinomial target plotting, use modin in place of pandas

def its_alive(str_='Hellow World!! I am Alive!!!'):
    """testing import"""
    print(str_)

def check_NaN(df):
    """
    Checks for NaN in the pandas DataFrame and spits a DataFrame of report.
    Uses df.isnull() method.

    Parameters:
    ===========
    df = pandas.DataFrame

    Returns:
    ========
    pandas.DataFrame

    ---version 0.9---
    """
    null_checking = []
    for column in df.columns:
        not_null = df[column].isnull().value_counts()[0]
        try:
            is_null = df[column].isnull().value_counts()[1]
        except:
            is_null = 0
        temp_dict = {'name': column, 'is_null': is_null, 'not_null': not_null}
        null_checking.append(temp_dict)
    df_ = pd.DataFrame(null_checking)
    return df_


def check_duplicates(df, verbose=False, limit_output=True, limit_num=150):
    """
    Checks for duplicates in the pandas DataFrame and return a Dataframe of report.

    Parameters:
    ===========
    df = pandas.DataFrame
    verbose = `int` or `boolean`; default: `False`
                `True` returns DataFrame with details of features.
                `False` returns DataFrame without details of features.
    limit_output = `int` or `boolean`; default: `True`
```

```python
                `True` limits featurs display to 150.
                `False` details of unique features.
    limit_num = `int`, limit number of uniques; default: 150,

    Returns:
    ========
    pandas.DataFrame

    ---version 1.2---
    """
    dup_checking = []
    for column in df.columns:
        not_duplicated = df[column].duplicated().value_counts()[0]
        try:
            duplicated = df[column].duplicated().value_counts()[1]
        except:
            duplicated = 0
        temp_dict = {
            'name': column,
            'duplicated': duplicated,
            'not_duplicated': not_duplicated
        }
        dup_checking.append(temp_dict)
    df_ = pd.DataFrame(dup_checking)

    if verbose:
        if limit_output:
            for col in df:
                if (len(df[col].unique()))<=limit_num:
                    print(f"{col} >> number of uniques: {len(df[col].unique())}\n{df[col].uniq
                else:
                    print(f"{col} >> number of uniques: {len(df[col].unique())}, showing top {
                print(f"{'_'*60}")
        else:
            for col in df:
                print(f"{col} >> number of uniques: {len(df[col].unique())}\n{df[col].unique()
    return df_


def num_col_for_plotting(row, col=3):
    """
    +++ formatting helper function +++
    __________________________________
    Returns number of rows to plot

    Parameters:
    ===========
    row = int;
```

```python
    col = int; default col: 3

    Return:
    =======
    `int` == row // col
    """
    if row % col != 0:
        return (row // col) + 1
    else:
        return row // col


def distribution_of_features(df, n_cols=3, fig_size=(16, 26), color_plot='gold', kde_show=True
    """
    Plots distribution of features in a pandas.DataFrame.
    Does not work on feature encoded as category.

    Parameters:
    ===========
    df             = pandas.DataFrame,
    n_cols         = int; default: 3,
                       controls number of columns per row of the figure.
    fig_size       = tuple (length, height); default: (16, 26),
                       controls the figure size of the output.
    color_plot     = str; default: 'gold',
                       controls color of the histplot and kde plot.
    kde_show       = `int` or `boolean`; default: `True`,
                       `True` shows kde plot.
                       `False` does not show kde plot.
    label_rotation = int; default: 45,
                       sets x label rotation.
    set_loglevel = str; default: 'warning',
                       The log level of matplotlib warning handler.
                       - options = {"notset", "debug", "info", "warning", "error", "critical"}


    ---version 1.2---
    """
    plt.set_loglevel(set_loglevel)

    fig, axes = plt.subplots(nrows=num_col_for_plotting(len(df.columns),
                                                        col=n_cols),
                             ncols=n_cols,
                             figsize=fig_size,
                             sharey=False)
    for ax, column in zip(axes.flatten(), df):
        sns.histplot(x=column, data=df, color=color_plot, ax=ax, kde=kde_show)
        ax.set_title(f'Histplot of {column.title()}')
```

```python
        ax.tick_params('x', labelrotation=label_rotation)
        sns.despine()
        plt.tight_layout()
        plt.suptitle('Histogram plots of the dataset',
                     fontsize=20,
                     fontweight=3,
                     va='bottom')
    plt.show()



def dataset_preprocessing_pipeline(X_train, X_test, scaler=StandardScaler(), drop=None):
    """
    Takes X_train, and X_test DataFrames. Then seperates DataFrame by categorical and numerica
    Returns transformed DataFrames.

    All transforming steps are done using scikit-learn preprocessing, pipeline, and compose ob

    Parameters:
    ===========
    X_train = pandas.DataFrame object; no default,
                training split of the DataFrame.
    X_test  = pandas.DataFrame object; no default,
                testing split of the DataFrame.
    scaler  = `sklarn scaler object` or `None`; default: StandardScaler(),
                *** IMPORT desired scaler before using. ***
                *** OR call with this module. all of them are imported and ready
                to use inside this module.***
                Available options:
                - StandardScaler: removes the mean and scales the data to
                    unit variance.
                - MinMaxScaler: rescales the data set such that all feature
                    values are in the range [0, 1]
                - RobustScaler: is based on percentiles and are therefore not
                    influenced by a few number of very large marginal outliers.
                - QuantileTransformer: applies a non-linear transformation
                    such that the probability density function of each feature
                    will be mapped to a uniform or Gaussian distribution.
                - PowerTransformer: applies a power transformation to each
                    feature to make the data more Gaussian-like in order to
                    stabilize variance and minimize skewness.
                - MaxAbsScaler: is similar to `MinMaxScaler` except that the
                    values are mapped in the range [0, 1]
                - Normalizer: rescales the vector for each sample to have
                    unit norm, independently of the distribution of the samples.
                - None: does not scale data.
    drop    = str or `None`; default: None.
                Option to control OneHotEncoder droping.
```

```
            - None : retain all features (the default).
            - 'first' : drop the first category in each feature. If only one
              category is present, the feature will be dropped entirely.
            - 'if_binary' : drop the first category in each feature with two
              categories. Features with 1 or more than 2 categories are
              left intact.
            - array : ``drop[i]`` is the category in feature ``X[:, i]`` that
              should be dropped.
Return:
=======
X_train = modified pandas.DataFrame
X_test  = modified pandas.DataFrame

NOTE:
    - possible error if test data has unseen category; creating new
      DataFrame will fail.
    - Source can be modified to add more preprocessing steps.

Next steps:
- add oversampling
- return pipeline

---version 0.9.9---
"""
# isolating numerical features
nume_cols = X_train.select_dtypes('number').columns.to_list()
# isolating categorical features
cate_cols = X_train.select_dtypes('category').columns.to_list()

# pipeline for processing categorical features
pipe_cate = Pipeline([('ohe', OneHotEncoder(sparse=False, drop=drop))])
# pipeline for processing numerical features
pipe_nume = Pipeline([('scaler', scaler)])

# Coulmn transformer
preprocessor = ColumnTransformer([
    ('numerical_features', pipe_nume, nume_cols),
    ('categorical_features', pipe_cate, cate_cols)
])

## creating a pandas.DataFrame with appropriate column name
# creating modified X_train
ret_X_train = pd.DataFrame(
    preprocessor.fit_transform(X_train),
    columns=nume_cols +
    preprocessor.named_transformers_['categorical_features'].
    named_steps['ohe'].get_feature_names(cate_cols).tolist())
```

```
    # creating modified X_test
    # NOTE: possible error if test data has unseen category, in this step.
    # for debugging such error modify this and its processing.
    ret_X_test = pd.DataFrame(
        preprocessor.transform(X_test),
        columns=nume_cols +
        preprocessor.named_transformers_['categorical_features'].
        named_steps['ohe'].get_feature_names(cate_cols).tolist())
    return ret_X_train, ret_X_test

def coefficients_of_model_binary(model,X_train_data, log_scale=True):
    """
    Returns a pandas.Series object with intercept and coeffients of a logistic regression mode

    Parameters:
    ===========
    model        = object; No Default.
                   fitted sklearn model object with a coef_ and intercept_ attribute.
    X_train_data = pandas.DataFrame; No Default.
                   DataFrame of independant variables. Should be train-test splitted.
                   Use train data.
    log_scale    = boolean; default: True.
                   `True` for keeping log scale of coefficients.
                   `False` for converting to normal scale.

    """
    coeffs = pd.Series(model.coef_.flatten(), index=X_train_data.columns)
    coeffs['intercept'] = model.intercept_[0]
    if log_scale is False:
        coeffs = np.exp(coeffs)
    return coeffs

def coefficients_of_model(model, log_scale=True):
    """
    Returns a pandas.Series object with intercept and coeffients.

    Parameters:
    ===========
    model        = object; No Default.
                   fitted sklearn model object with a coef_ and intercept_ attribute.
    log_scale    = boolean; default: True.
                   `True` for keeping log scale of coefficients.
                   `False` for converting to normal scale.

    """
    coeffs = pd.Series(model.coef_.flatten())
    coeffs['intercept'] = model.intercept_[0]
    if log_scale is False:
```

```
        coeffs = np.exp(coeffs)
    return coeffs

def save_model(model, location='',custom_prefix=''):
    """
    Saves object locally as binary format with and as joblib.
    Adds local machine time to name to the file for easy recongnition.

    Parameters:
    ===========
    model = object to save,
    location = str; default: '',
            File save location. If empty, i.e., "", saves at root.
    custom_prefix = str; default: '',
            Adds prefix to file

    Future plan:
    - modify naming options

    --version 0.0.1--

    """
    def str_model_(model):
        """Helper function to get model class display statement, this text
        conversion breaks code if performed in ``save_model`` function's
        local space. This function is to isolate from the previous function's
        local space."""
        str_model = str(model.__class__).split('.')[-1][:-2]
        return str_model

    # get model name
    name = str_model_(model)
    # save time
    month = str(time.localtime().tm_mon)
    day = str(time.localtime().tm_mday)
    year = str(time.localtime().tm_year)
    hour = str(time.localtime().tm_hour)
    min_ = str(time.localtime().tm_min)
    sec = str(time.localtime().tm_sec)
    save_time = '_'.join([month, day, year, hour, min_, sec])

    file_name_ = '_'.join([custom_prefix, name, save_time])
    save_path = location+file_name_+'.joblib'
    joblib.dump(model, save_path)
    print(f'File saved: {save_path}')

def heatmap_of_features(df, annot_format='.1f'):
    """
```

```
    Return a masked heatmap of the given DataFrame

    Parameters:
    ===========
    df              = pandas.DataFrame object.
    annot_format    = str, for formatting; default: '.1f'

    Example of `annot_format`:
    --------------------------
    .1e = scientific notation with 1 decimal point (standard form)
    .2f = 2 decimal places
    .3g = 3 significant figures
    .4% = percentage with 4 decimal places

    Note:
    =====
    Rounding error can happen if '.1f' is used.

    -- version: 1.1 --
    """
    with plt.style.context('dark_background'):
        plt.figure(figsize=(10, 10), facecolor='k')
        mask = np.triu(np.ones_like(df.corr(), dtype=bool))
        cmap = sns.diverging_palette(3, 3, as_cmap=True)
        ax = sns.heatmap(df.corr(),
                    mask=mask,
                    cmap=cmap,
                    annot=True,
                    fmt=annot_format,
                    linecolor='k',
                    annot_kws={"size": 9},
                    square=True,
                    linewidths=.5,
                    cbar_kws={"shrink": .5})
        plt.title(f'Features heatmap', fontdict={"size": 20})
        plt.show()
        return ax

def top_correlated_features(df, limit=.75, verbose=False):
    """
    Input a Pandas DataFrame to get top correlated (based on absolute value) features filtered

    Parameters:
    ===========
    df          = pandas.DataFrame object.
    limit       = float; default: .75
    verbose     = boolean; default: False.
                `True` returns DataFrame without filtering by cutoff.
```

```python
                `False` returns DataFrame filted by cutoff.
    """
    df_corr = df.corr().abs().unstack().reset_index().sort_values(
        0, ascending=False)
    df_corr.columns = ["feature_0", 'feature_1', 'correlation']
    df_corr['keep_me'] = df_corr.apply(
        lambda x: False if x['feature_0'] == x['feature_1'] else True, axis=1)
    df_corr['feature_combo'] = df_corr.apply(
        lambda x: ' and '.join(set(x[['feature_0', 'feature_1']])), axis=1)

    corr_features = df_corr[df_corr.keep_me == True][[
        'feature_combo', 'correlation'
    ]].drop_duplicates().reset_index(drop='index')
    # features with correlation more than 75%
    if verbose == True:
        return corr_features
    else:
        return corr_features[corr_features.correlation > limit]

def drop_features_based_on_correlation(df, threshold=0.75):
    """
    Returns features with high collinearity.

    Parameters:
    ===========
    df = pandas.DataFrame; no default.
            data to work on.
    threshold = float; default: .75.
            Cut off value of check of collinearity.

    -- ver: 1.0 --
    """
    # Set of all the names of correlated columns
    feature_corr = set()
    corr_matrix = df.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            # absolute coeff value
            if abs(corr_matrix.iloc[i, j]) > threshold:
                # getting the name of column
                colname = corr_matrix.columns[i]
                feature_corr.add(colname)
    return feature_corr

def show_py_file_content(file='./imports_and_functions/functions.py'):
    """
    displays content of a py file output formatted as python code in jupyter notebook.
```

```
    Parameter:
    ==========
    file = `str`; default: './imports_and_functions/functions.py',
                path to the py file.
    """
    with open(file, 'r') as f:
        x = f"""```python
{f.read()}
```"""
        display(Markdown(x))

def z_dataset_preprocessing_pipeline(X_train,
                                     X_test,
                                     y_train=None,
                                     scaler=StandardScaler(),
                                     drop=None,
                                     oversampling=True,
                                     return_pipeline_object=False):
    """ ######## Work in progress. Code works good enough.
    Takes X_train, and X_test DataFrames. Then seperates DataFrame by categorical and numerica
    Returns transformed DataFrames.

    All transforming steps are done using scikit-learn preprocessing, pipeline, and compose ob

    :::: MAKE SURE EVERY FEATURE HAS CORRECT DATA TYPE; EITHER CATEGORICAL OR NUMERICAL :::

    Parameters:
    ===========

    X_train = pandas.DataFrame object; no default,
                training split of the DataFrame.
    X_test = pandas.DataFrame object; no default,
                testing split of the DataFrame.
    scaler = `sklarn scaler object` or `None`; default: StandardScaler(),
                *** IMPORT desired scaler before using. ***
                *** OR call with this module. all of them are imported and ready
                to use inside this module.***
                Available options:
                - StandardScaler: removes the mean and scales the data to
                    unit variance.
                - MinMaxScaler: rescales the data set such that all feature
                    values are in the range [0, 1]
                - RobustScaler: is based on percentiles and are therefore not
                    influenced by a few number of very large marginal outliers.
                - QuantileTransformer: applies a non-linear transformation
                    such that the probability density function of each feature
                    will be mapped to a uniform or Gaussian distribution.
                - PowerTransformer: applies a power transformation to each
```
```

```
                feature to make the data more Gaussian-like in order to
                stabilize variance and minimize skewness.
            - MaxAbsScaler: is similar to `MinMaxScaler` except that the
                values are mapped in the range [0, 1]
            - Normalizer: rescales the vector for each sample to have
                unit norm, independently of the distribution of the samples.
            - None: does not scale data. #::: NOT TESTED :::#
drop    = str or `None`; default: None.
            Option to control OneHotEncoder droping.
            - None : retain all features (the default).
            - 'first' : drop the first category in each feature. If only one
              category is present, the feature will be dropped entirely.
            - 'if_binary' : drop the first category in each feature with two
              categories. Features with 1 or more than 2 categories are
              left intact.
            - array : ``drop[i]`` is the category in feature ``X[:, i]`` that
              should be dropped.
oversampling = boolean; default: True,
                turn oversampling on or off;
            - `True` oversamples.
            - `False` no oversampling.
return_pipeline_object= boolean; default: False, {not sure how it might be useful though #
                control object return.
            - `True` returns object.
            - `False` does not return object.
NOTE:
    - possible error if test data has unseen category; creating new
      DataFrame will fail.
    - Source can be modified to add more preprocessing steps.

Stage: Coding

Next steps:
- use OOP to make this a class.
- Add oversampling method changing option.
- add imputer in the pipeline.
- add and remove steps in pipeline option.

---version 0.0.1 beta---
"""
# isolating numerical features
nume_cols = X_train.select_dtypes('number').columns.to_list()
# isolating categorical features
cate_cols = X_train.select_dtypes('category').columns.to_list()

# pipeline for processing categorical features
pipe_cate = Pipeline([('ohe', OneHotEncoder(sparse=False, drop=drop))])
# pipeline for processing numerical features
```

```python
    pipe_nume = Pipeline([('scaler', scaler)])

    # Coulmn transformer
    preprocessor = ColumnTransformer([
        ('numerical_features', pipe_nume, nume_cols),
        ('categorical_features', pipe_cate, cate_cols)
    ])

    # creating a pandas.DataFrame with appropriate header
    # creating modified X_train
    ret_X_train = pd.DataFrame(
        preprocessor.fit_transform(X_train),
        columns=nume_cols +
        preprocessor.named_transformers_['categorical_features'].
        named_steps['ohe'].get_feature_names(cate_cols).tolist())

    # creating modified X_test
    ## NOTE: possible error if test data has unseen category, in this step.
    ## for debugging such error modify this, and its processing steps `in pipe_cate`.
    ret_X_test = pd.DataFrame(
        preprocessor.transform(X_test),
        columns=nume_cols +
        preprocessor.named_transformers_['categorical_features'].
        named_steps['ohe'].get_feature_names(cate_cols).tolist())

    # NEW ADDITION
    if oversampling:
        smotenc_features = [True] * len(nume_cols) + [False] * len(
            preprocessor.named_transformers_['categorical_features'].
            named_steps['ohe'].get_feature_names(cate_cols).tolist())
        oversampling_ = SMOTENC(categorical_features=smotenc_features,
                                n_jobs=-1)
        X_train_oversampled = oversampling_.fit_sample(ret_X_train, y_train)

    if return_pipeline_object:
        if oversampling:
            return preprocessor, X_train_oversampled, ret_X_test
        else:
            return preprocessor, ret_X_train, ret_X_test
    else:
        if oversampling:
            return X_train_oversampled, ret_X_test
        else:
            return ret_X_train, ret_X_test

def z_experimental_model_report_(model,
                                 X_train,
                                 y_train,
```

```
                                      X_test,
                                      y_test,
                                      cmap=['Reds', 'Greens'],
                                      normalize='true',
                                      figsize=(16, 6),
                                      show_train_report=False,
                                      show_train_roc=False,
                                      fitted_model=False,
                                      display_labels=['not_met', 'met']):
""" ### Work in progress, code works. Bulding upon the working version of the code.###
Report of model performance using train-test split dataset.
Shows train and test score, Confusion Matrix and, ROC Curve of performane of test data.

Intended to work ONLY on model where target has properly encoded binomial class value.

Parameters:
===========
model     = object, scikit-learn model object; no default.
X_train   = pandas.DataFrame, predictor variable training data split; no default,
y_train   = pandas.DataFrame, target variable training data split; no default,
X_test    = pandas.DataFrame, predictor variable test data split; no default,
y_test    = pandas.DataFrame, target variable test data split; no default,
cmap      = str, colormap of Confusion Matrix; default: 'Greens',
normalize = str, normalize count of Confusion Matrix; default: 'true',
            - `true` to normalize counts.
            - `false` to show raw scounts.
figsize   = tuple ``(lenght, height)``, figsize of output; default: (16, 6),
show_train_report= boolean; default: False,
            - True, to show report.
            - False, to turn off report.
fitted_model = False,
display_labels = ['not_met', 'met']

Future plan:
- `save model` option in local drive using joblib or pickle
- return fitted model
- diffrent scorer option for report
- turn off testing model performance on test data
- bring fuctionality from the old model
- rebuild for multiclass using yellowbricks
- another version of code for reporting already fitted model #-code ready-#
- return reusable report object
- add labaling options for 0 and 1 target class in report ===> confusion matrix. #-code re
- rethink control flow of display options, am I using more code than necessary?

Stage: Concept, idea generation.

Changelog:
```

```
- built skeleton
- added fitted_model
- added display_labels

---version 0.0.1 pre-alpha---
"""
def str_model_(model):
    """Helper function to get model class display statement, this text conversion breaks c
    performed in ``model_report`` function's local space. This function is to isolate from
    previous function's local space. Can use class here"""
    str_model = str(model.__class__).split('.')[-1][:-2]
    display(
        HTML(
            f"""<strong>Report of {str_model} type model using train-test split dataset.</s
        ))

str_model_(model)
X_train = X_train.copy()
y_train = y_train.copy()
if fitted_model is False:
    model.fit(X_train, y_train)
print(f"{'*'*90}")
train = model.score(X_train, y_train)
test = model.score(X_test, y_test)
print(f"""Train accuracy score: {train.round(4)}""")
print(f"""Test accuracy score: {test.round(4)}""")
if abs(train - test) <= .05:
    print(
        f"    No over or underfitting detected, diffrence of scores did not cross 5% thresh
    )
elif (train - test) > .05:
    print(
        f"    Possible Overfitting, diffrence of scores {round(abs(train-test)*100,2)}% cro
    )
elif (train - test) < -.05:
    print(
        f"    Possible Underfitting, diffrence of scores {round(abs(train-test)*100,2)}% cr
    )
print(f"{'*'*90}")
print("")
print(f"{'*'*60}")

if (show_train_roc) & (show_train_report):
    print(f"""Classification report on train data of:
    {model}""")
    print(f"{'-'*60}")
    print(metrics.classification_report(y_train, model.predict(X_train)))
    print(f"{'*'*60}")
```

```
    print(f"{'*'*60}")
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)
    metrics.plot_confusion_matrix(model,
                                  X_train,
                                  y_train,
                                  cmap=cmap[0],
                                  normalize=normalize, display_labels=display_labels,
                                  ax=ax[0])
    ax[0].title.set_text('Confusion Matrix')
    metrics.plot_roc_curve(model,
                           X_train,
                           y_train,
                           color='gold',
                           ax=ax[1])
    ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
    ax[1].grid()
    ax[1].title.set_text('ROC Curve')

    plt.tight_layout()
    plt.show()
    print(f"{'*'*60}")
elif (show_train_report is True) & (show_train_roc is False):
    print(f"""Classification report on train data of:
    {model}""")
    print(f"{'-'*60}")
    print(metrics.classification_report(y_train, model.predict(X_train)))
    print(f"{'*'*60}")
    print(f"{'*'*60}")
elif show_train_roc:
    print(f"""Confusion Matrix and ROC curve on train data of:
    {model}""")
    print(f"{'-'*60}")
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)
    metrics.plot_confusion_matrix(model,
                                  X_train,
                                  y_train,
                                  cmap=cmap[0],
                                  normalize=normalize, display_labels=display_labels,
                                  ax=ax[0])
    ax[0].title.set_text('Confusion Matrix')
    metrics.plot_roc_curve(model,
                           X_train,
                           y_train,
                           color='gold',
                           ax=ax[1])
    ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
    ax[1].grid()
    ax[1].title.set_text('ROC Curve')
```

```
        plt.tight_layout()
        plt.show()
        print(f"{'*'*60}")

    print(f"""Classification report on test data of:
    {model}""")
    print(f"{'-'*60}")
    print(metrics.classification_report(y_test, model.predict(X_test)))
    print(f"{'*'*60}")

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)
    metrics.plot_confusion_matrix(model,
                                  X_test,
                                  y_test,
                                  cmap=cmap[1],
                                  normalize=normalize, display_labels=display_labels,
                                  ax=ax[0])
    ax[0].title.set_text('Confusion Matrix')
    metrics.plot_roc_curve(model,
                           X_test,
                           y_test,
                           color='gold',
                           ax=ax[1])
    ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
    ax[1].grid()
    ax[1].title.set_text('ROC Curve')

    plt.tight_layout()
    plt.show()

def model_report(model,
                 X_train,
                 y_train,
                 X_test,
                 y_test,
                 cmap=['Reds','Greens'],
                 normalize='true',
                 figsize=(16, 6),
                 show_train_report=False,
                 unfitted_model=True):
    """
    Report of model performance using train-test split dataset.
    Shows train and test score, Confusion Matrix and, ROC Curve of performane of test data.

    Intended to work ONLY on model where target has properly encoded binomial class value.

    Parameters:
```

```
===========
model     = object, scikit-learn model object; no default.
X_train   = pandas.DataFrame, predictor variable training data split; no default,
y_train   = pandas.DataFrame, target variable training data split; no default,
X_test    = pandas.DataFrame, predictor variable test data split; no default,
y_test    = pandas.DataFrame, target variable test data split; no default,
cmap      = list of str, colormap of Confusion Matrix; default: ['Reds','Greens'],
            cmap of train and test data
normalize = str, normalize count of Confusion Matrix; default: 'true',
            - `true` to normalize counts.
            - `false` to show raw scounts.
figsize   = tuple ``(lenght, height)``, figsize of output; default: (16, 6),
show_train_report = boolean; default: False,
            - True, to show report.
            - False, to turn off report.
unfitted_model = bool; default: True,
            - if True, fits model to train data and generates report.
            - if False, does not fits model and generates report.
            Use False for previously fitted model.

---version 0.9.14---
"""
def str_model_(model):
    """Helper function to get model class display statement, this text conversion breaks c
    performed in ``model_report`` function's local space. This function is to isolate from
    previous function's local space."""
    str_model = str(model.__class__).split('.')[-1][:-2]
    display(
        HTML(
            f"""<strong>Report of {str_model} type model using train-test split dataset.</s
        ))

str_model_(model)
X_train = X_train.copy()
y_train = y_train.copy()
if unfitted_model:
    model.fit(X_train, y_train)
print(f"{'*'*90}")
train = model.score(X_train, y_train)
test = model.score(X_test, y_test)
print(f"""Train accuracy score: {train.round(4)}""")
print(f"""Test accuracy score: {test.round(4)}""")
if abs(train - test) <= .05:
    print(
        f"    No over or underfitting detected, diffrence of scores did not cross 5% thresh
    )
elif (train - test) > .05:
    print(
```

```
        f"    Possible Overfitting, diffrence of scores {round(abs(train-test)*100,2)}% cr
    )
elif (train - test) < -.05:
    print(
        f"    Possible Underfitting, diffrence of scores {round(abs(train-test)*100,2)}% c
    )
print(f"{'*'*90}")
print("")
print(f"{'*'*60}")

if show_train_report:
    print(f"""Classification report on train data of:
    {model}""")
    print(f"{'-'*60}")
    print(metrics.classification_report(y_train, model.predict(X_train)))
    print(f"{'*'*60}")
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)
    metrics.plot_confusion_matrix(model,
                                  X_train,
                                  y_train,
                                  cmap=cmap[0],
                                  normalize=normalize,
                                  ax=ax[0])
    ax[0].title.set_text('Confusion Matrix')
    metrics.plot_roc_curve(model,
                           X_train,
                           y_train,
                           color='gold',
                           ax=ax[1])
    ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
    ax[1].grid()
    ax[1].title.set_text('ROC Curve')

    plt.tight_layout()
    plt.show()
    print(f"{'='*170}")
    print(f"{'*'*60}")

print(f"""Classification report on test data of:
{model}""")
print(f"{'-'*60}")
print(metrics.classification_report(y_test, model.predict(X_test)))
print(f"{'*'*60}")

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)
metrics.plot_confusion_matrix(model,
                              X_test,
                              y_test,
```

```
                                    cmap=cmap[1],
                                    normalize=normalize,
                                    ax=ax[0])
    ax[0].title.set_text('Confusion Matrix')
    metrics.plot_roc_curve(model,
                           X_test,
                           y_test,
                           color='gold',
                           ax=ax[1])
    ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
    ax[1].grid()
    ax[1].title.set_text('ROC Curve')

    plt.tight_layout()
    plt.show()

def convert_html_to_image(st="Does nothing"):
    """Does nothing right now."""
    # use this
    # pip install imgkit
    print(st)

def javascript_formatter(javascript,background_color='White', font_color='black'):
    """
    Helper fuction to jormat javascript object's background and font color. Helpful to use in 

    Parameters:
    ===========
    javascript = str; no default,
                javascript formated as html string.
    background_color= str; default: 'White',
                Note: follow html syntax and convention
    font_color= str; default: 'black',
                Note: follow html syntax and convention

    --version 0.0.1--
    """
    display(HTML(f"""<div style="background-color:{background_color}; color:{font_color};">{ja

def show_path():
    """Show path locations"""
    import sys
    for p in sys.path:
        print(p)
```

[149]:
```
fun.show_py_file_content('./imports_and_functions/packages.py')
```

```
# core operational packeges
```

```python
import os
import warnings
import joblib

# dataset manupulation
import pandas as pd
pd.set_option('display.max_columns', 0)
import numpy as np
from IPython.display import display, HTML, Markdown

# plotting
import matplotlib.pyplot as plt
import matplotlib.ticker as plticker
import seaborn as sns

# Machine Learning
# preprocessing
from sklearn import set_config
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder
# from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn import metrics
from imblearn.over_sampling import SMOTE,SMOTENC

# model
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from xgboost import XGBClassifier, XGBRFClassifier
import xgboost as xgb

from catboost import CatBoostClassifier

# Model Explainers and explorers
# from yellowbrick.classifier.rocauc import roc_auc, precision_recall_curve, confusion_matrix,
from sklearn import tree
import shap
import eli5
```

10.2 KNN interpretation with shap

[166]:
```
# exp = shap.KernelExplainer(knn_gs_best.predict_proba,shap.sample(X_train_knn))
# shapval = exp.shap_values(X_test_knn)
```

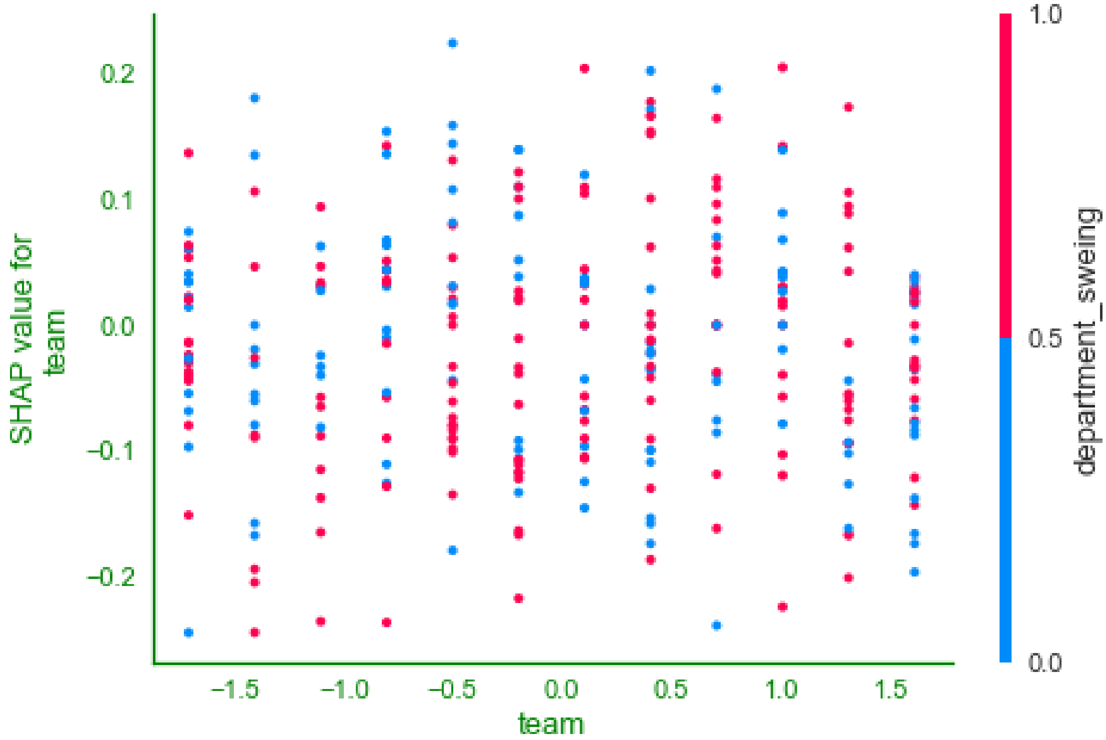
[58]:
```
shapval = joblib.load('shapval_knn_best.joblib')
```

[59]:
```
with plt.style.context('seaborn-white'):
    shap.summary_plot(shapval[0],X_test_knn)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-59-3e8a00ab54da> in <module>
      1 with plt.style.context('seaborn-white'):
----> 2     shap.summary_plot(shapval[0],X_test_knn)

NameError: name 'X_test_knn' is not defined
```

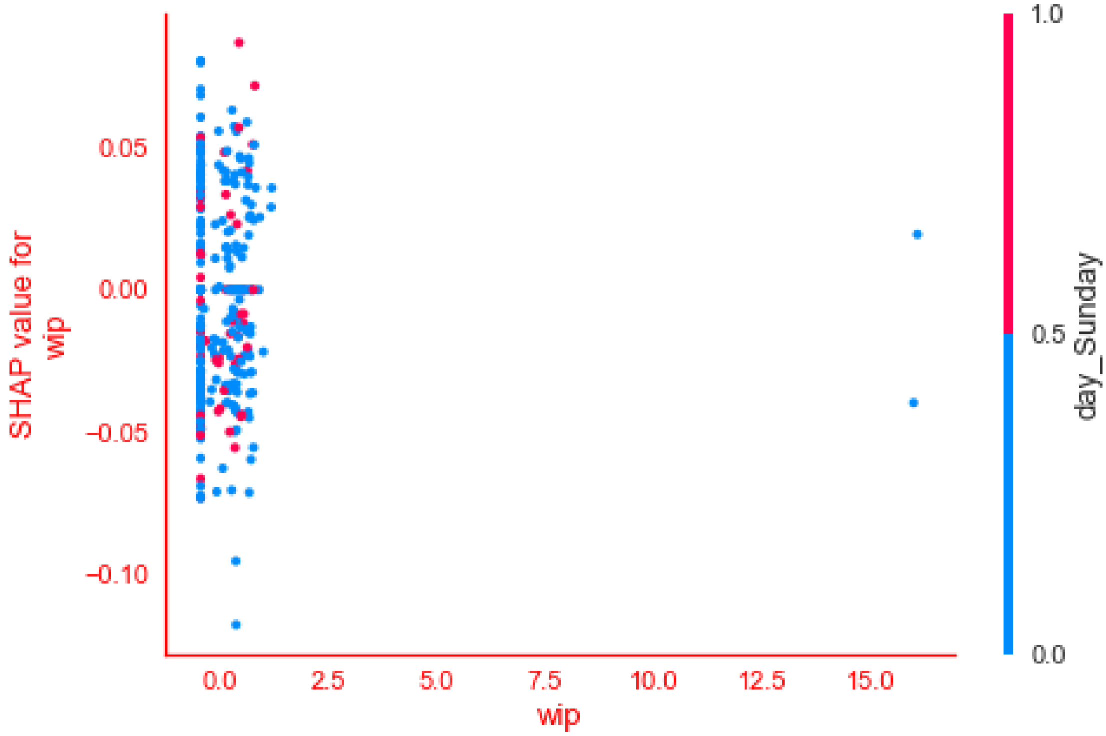
[90]:
```
with plt.style.context('seaborn-white'):
    shap.dependence_plot('team',
                         shapval[0],
                         X_test_knn,
                         feature_names=X_test_knn.columns,
                         axis_color='green')
```

```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
will become an error two minor releases later. Please pass vmin/vmax directly to
the norm when creating it.
```



[92]:
```
with plt.style.context('seaborn-white'):
    shap.dependence_plot('wip',
                         shapval[0],
                         X_test_knn,
                         feature_names=X_test_knn.columns,
                         axis_color='red')
```

```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
will become an error two minor releases later. Please pass vmin/vmax directly to
the norm when creating it.
```



10.3 Other models tried

10.3.1 ensemble methods

[ ]:
```
X_train_ensbl, X_test_ensbl = fun.dataset_preprocessing_pipeline(X_train,␣
↪X_test)
```

RandomForestTM grid search with Cross Validation

[75]:
```
rf_clf_gs = RandomForestClassifier(n_jobs=-1)
params = {
    'criterion': ["gini", "entropy"],
    'max_depth': [2, 3, 4, None],
    'min_samples_leaf': [1, 2, 3, 4],
    'class_weight': ["balanced", "balanced_subsample"]
}
gridsearch_rf_clf = GridSearchCV(estimator=rf_clf_gs,
                                 param_grid=params,
                                 n_jobs=-1,
                                 scoring='precision')
gridsearch_rf_clf
```

[75]:
```
GridSearchCV(estimator=RandomForestClassifier(n_jobs=-1), n_jobs=-1,
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
```

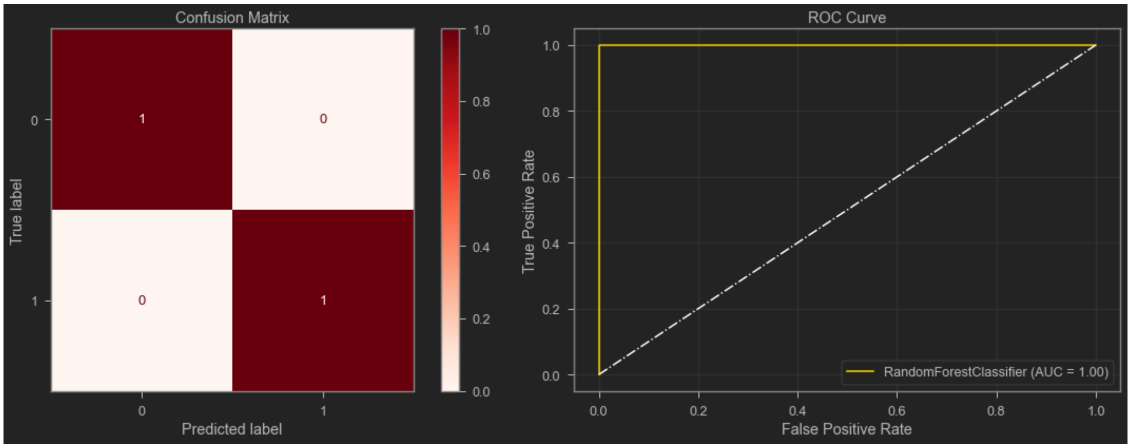
```
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 3, 4, None],
                         'min_samples_leaf': [1, 2, 3, 4]},
             scoring='precision')
```

[76]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_rf_clf.fit(X_train_ensbl, y_train)
print(f"Best Parameters by gridsearch:\t{gridsearch_rf_clf.best_params_}")
print(f"Best Estimator by gridsearch:\t{gridsearch_rf_clf.best_estimator_}")

rf_clf_gs_best = gridsearch_rf_clf.best_estimator_
fun.model_report(rf_clf_gs_best, X_train_ensbl, y_train, X_test_ensbl,
             y_test,show_train_report=True)
```

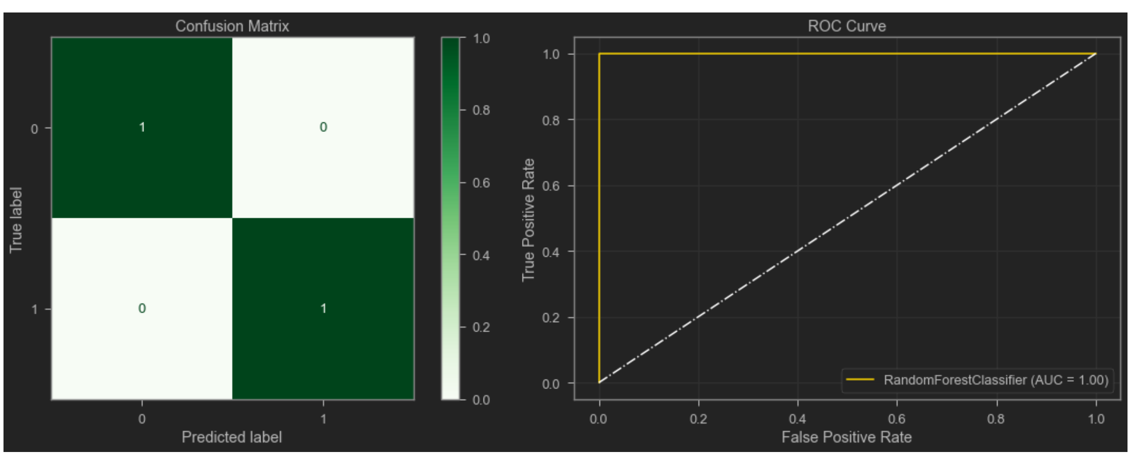
```
Best Parameters by gridsearch:  {'class_weight': 'balanced', 'criterion':
'entropy', 'max_depth': None, 'min_samples_leaf': 1}
Best Estimator by gridsearch:   RandomForestClassifier(class_weight='balanced',
criterion='entropy', n_jobs=-1)
```

```
<IPython.core.display.HTML object>
```

```
********************************************************************************
**********
Train accuracy score: 1.0
Test accuracy score: 1.0
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on train data of:
        RandomForestClassifier(class_weight='balanced', criterion='entropy',
n_jobs=-1)
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       875
           1       1.00      1.00      1.00       875

    accuracy                           1.00      1750
   macro avg       1.00      1.00      1.00      1750
weighted avg       1.00      1.00      1.00      1750

************************************************************
```




```
================================================================================
================================================================================
==========
************************************************************
Classification report on test data of:
    RandomForestClassifier(class_weight='balanced', criterion='entropy',
n_jobs=-1)
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        77
           1       1.00      1.00      1.00       223

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300

************************************************************
```

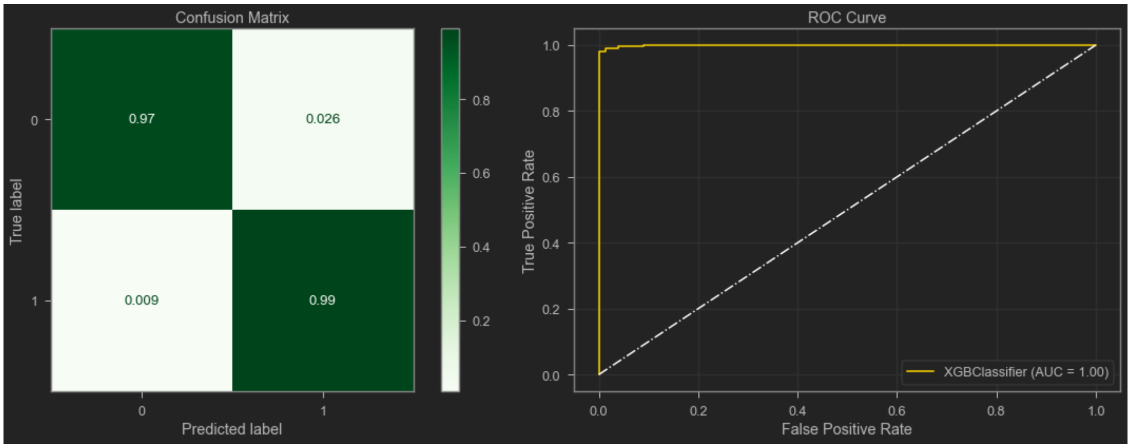


Performance drop can be if explained for different 'max_depth' because of the internal cross validation process hindering class imbalance of the train data which was not addressed by `class_weight='balanced'` parameter and trees were not allowed to expand as required.

XGBClassifier

[77]:
```
xgg_clf = XGBClassifier(n_jobs=-1)
fun.model_report(xgg_clf, X_train_ensbl, y_train, X_test_ensbl,
                 y_test)
```

```
<IPython.core.display.HTML object>

********************************************************************************
**********
Train accuracy score: 0.992
Test accuracy score: 0.9867
    No over or underfitting detected, diffrence of scores did not cross 5%
thresh hold.
********************************************************************************
**********

************************************************************
Classification report on test data of:
    XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        77
           1       0.99      0.99      0.99       223

    accuracy                           0.99       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.99      0.99      0.99       300

************************************************************
```



grid search with Cross Validation

[ ]:
```
xgg_clf_gs = XGBClassifier(n_jobs=-1,verbosity=0,objective='binary:
↪logistic',eval_metric='error')#"rank:pairwise","count:poisson"␣
↪#'logloss','auc'
params = {
    'criterion': ["gini", "entropy"],
    'max_depth': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'class_weight': ["balanced", "balanced_subsample"],
    'ccp_alpha': [0.0, 0.05, 0.1, 0.2, 0.3],
    'importance_type':["gain", "weight", "cover", "total_gain","total_cover"],
}
gridsearch_xgg_clf_gs = GridSearchCV(estimator=xgg_clf_gs,
                                     param_grid=params,
                                     n_jobs=-1,
                                     scoring='precision')  #'roc_auc_ovr_weighted'
gridsearch_xgg_clf_gs
```

[ ]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_xgg_clf_gs.fit(X_train_ensbl, y_train)
#     print(f"Best Parameters by gridsearch:\t{gridsearch_xgg_clf_gs.
↪best_params_}")
#     print(f"Best Estimator by gridsearch:\t{gridsearch_xgg_clf_gs.
↪best_estimator_}")


    xgg_clf_gs_best = gridsearch_xgg_clf_gs.best_estimator_
    xgb.plot_importance(xgg_clf_gs_best)
    fun.model_report(xgg_clf_gs_best, X_train_ensbl, y_train, X_test_ensbl,
                 y_test)
```

XGBRFClassifier

```
[ ]: xgg_rf_clf = XGBRFClassifier()
     fun.model_report(xgg_rf_clf, X_train_ensbl, y_train, X_test_ensbl,
                  y_test)
```

grid search with Cross Validation

```
[ ]: xgg_rf_clf_gs = XGBRFClassifier(
         n_jobs=-1,
         verbosity=0,
         objective='binary:logistic',
     )  #"rank:pairwise","count:poisson" #'logloss','auc','error'
     # params = {
     #     'criterion': ["gini", "entropy"],
     #     'max_depth': [2, 3, 4],
     #     'min_samples_leaf': [1, 2, 3, 4],
     #     'class_weight': ["balanced", "balanced_subsample"],
     #     'ccp_alpha': [0.0, 0.01]
     # }
     params = {
         'criterion': ["gini", "entropy"],
         'max_depth': [2, 3, 4],
         'min_samples_leaf': [1, 2, 3, 4],
         'class_weight': ["balanced", "balanced_subsample"],
         'ccp_alpha': [0.0, 0.05, 0.1, 0.2, 0.3],
         'importance_type':
         ["gain", "weight", "cover", "total_gain", "total_cover"],
         'eval_metric': ['logloss', 'auc', 'error']
     }
     gridsearch_xgg_rf_clf = GridSearchCV(
         estimator=xgg_rf_clf_gs,
         param_grid=params,
         n_jobs=-1,
         scoring='roc_auc')  #'roc_auc_ovr_weighted'
     gridsearch_xgg_rf_clf
```

```
[ ]: with warnings.catch_warnings():
         warnings.simplefilter("ignore")
         gridsearch_xgg_rf_clf.fit(X_train_ensbl, y_train)
     #     print(f"Best Parameters by gridsearch:\t{gridsearch_xgg_rf_clf.
      ↪best_params_}")
     #     print(f"Best Estimator by gridsearch:\t{gridsearch_xgg_rf_clf.
      ↪best_estimator_}")

         xgg_rf_clf_gs_best = gridsearch_xgg_rf_clf.best_estimator_
         xgb.plot_importance(xgg_rf_clf_gs_best)
         fun.model_report(xgg_rf_clf_gs_best, X_train_ensbl, y_train, X_test_ensbl,
                      y_test)
```

catboost

[ ]:
```
model = CatBoostClassifier(task_type='GPU',
                           auto_class_weights='SqrtBalanced',
                           eval_metric='Precision',
                           devices=[0, 1],
                           min_data_in_leaf=3,
                           iterations=500)

#### Abailable options for `eval_metric` ####
# 'Logloss', 'CrossEntropy', 'CtrFactor', 'RMSE', 'Lq', 'MAE', 'Quantile',
# 'Expectile', 'LogLinQuantile', 'MAPE', 'Poisson', 'MSLE',
# 'MedianAbsoluteError', 'SMAPE', 'Huber', 'Tweedie', 'RMSEWithUncertainty',
# 'MultiClass', 'MultiClassOneVsAll', 'PairLogit', 'PairLogitPairwise',
# 'YetiRank', 'YetiRankPairwise', 'QueryRMSE', 'QuerySoftMax',
# 'QueryCrossEntropy', 'StochasticFilter', 'StochasticRank',
# 'PythonUserDefinedPerObject', 'PythonUserDefinedMultiRegression',
# 'UserPerObjMetric', 'UserQuerywiseMetric', 'R2', 'NumErrors', 'FairLoss',
# 'AUC', 'Accuracy', 'BalancedAccuracy', 'BalancedErrorRate', 'BrierScore',
# 'Precision', 'Recall', 'F1', 'TotalF1', 'MCC', 'ZeroOneLoss',
# 'HammingLoss', 'HingeLoss', 'Kappa', 'WKappa', 'LogLikelihoodOfPrediction',
# 'NormalizedGini', 'PRAUC', 'PairAccuracy', 'AverageGain', 'QueryAverage',
# 'QueryAUC', 'PFound', 'PrecisionAt', 'RecallAt', 'MAP', 'NDCG', 'DCG',
# 'FilteredDCG', 'MultiRMSE', 'Combination'

cat_features = list(X_train.select_dtypes('category').columns)

model.fit(X_train,
          y_train,
          cat_features=cat_features,
          eval_set=(X_test, y_test),
          plot=True,
          silent=True,
          use_best_model=True)
print(f'{"-"*90}')
train = model.score(X_train, y_train)
test = model.score(X_test, y_test)
print(f"""Train score: {train.round(4)}""")
print(f"""Test score: {test.round(4)}""")
print(f"")
print(metrics.classification_report(y_test, model.predict(X_test)))
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
metrics.plot_confusion_matrix(model,
                              X_test,
                              y_test,
                              cmap='Greens',
                              normalize='true',
                              ax=ax[0])
```

```python
ax[0].title.set_text('Confusion Matrix')
metrics.plot_roc_curve(model, X_test, y_test, color='gold', ax=ax[1])
ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
ax[1].grid()
ax[1].title.set_text('ROC Curve')

plt.tight_layout()
plt.show()
```

[55]:
```python
##### grid search with Cross Validation
```

[56]:
```python
# from sklearn.metrics import make_scorer, accuracy_score, recall_score,␣
↪precision_score,precision_recall_curve, f1_score

# with warnings.catch_warnings():
#     warnings.simplefilter("ignore")
#     clf = CatBoostClassifier(task_type='GPU', iterations=2)
#     params = {
#         'eval_metric': ['Precision', 'Accuracy', 'Recall', 'AUC', 'F1'],
#         'depth': [4, 5, 6],
#         'loss_function': ['Logloss', 'CrossEntropy'],
#         'l2_leaf_reg': [1, 3, 5, 7, 9],
#         'auto_class_weights': ['SqrtBalanced', 'Balanced', None],
#         'leaf_estimation_method': ['Newton', 'Gradient'],
#         'logging_level': ['Silent']
#     }
#     scorer = make_scorer(f1_score)
#     clf_grid = GridSearchCV(estimator=clf,
#                             param_grid=params,
#                             scoring=scorer,
#                             cv=5)
#     clf_grid.fit(X_train_ensbl, y_train)
#     print(clf_grid.best_params_)
#     clf_grid_best = clf_grid.best_estimator_
#     fun.model_report(clf_grid_best, X_train_ensbl, y_train, X_test_ensbl,
#                      y_test)

# # create pool, then pass to frid search
# model.grid_search()
```

10.3.2 Support Vector Machines

[ ]:
```python
X_train_svm, X_test_svm  = fun.dataset_preprocessing_pipeline(X_train, X_test)
```

lin

[ ]:
```python
svc_linear = SVC(kernel='linear', C=100,class_weight='balanced')
fun.model_report(svc_linear, X_train_svm, y_train, X_test_svm,
```

```
                 y_test)
```

rbf

[ ]:
```
svc_rbf = SVC(kernel='rbf', C=1, gamma='auto', class_weight='balanced', tol=.8)
fun.model_report(svc_rbf, X_train_svm, y_train, X_test_svm,
                 y_test)
```

poly

[ ]:
```
svc_poly = SVC(kernel='poly',
               degree=8,
               C=1,
               gamma='scale',
               class_weight='balanced')
fun.model_report(svc_poly, X_train_svm, y_train, X_test_svm,
                 y_test)
```

sigmoid

[ ]:
```
svc_sig = SVC(kernel='sigmoid', C=2, class_weight='balanced')
fun.model_report(svc_sig, X_train_svm, y_train, X_test_svm,
                 y_test)
```

grid search with Cross Validation

[ ]:
```
svc_linear_gs = SVC(class_weight="balanced")
params = {
    'C': [1, 10, 1e2, 1e3],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],
    'tol': [0.001, .5, 1, 5],
}
gridsearch_svc_linear = GridSearchCV(
    estimator=svc_linear_gs,
    param_grid=params,
    n_jobs=-1,
    scoring='roc_auc')  #'roc_auc_ovr_weighted'
gridsearch_svc_linear
```

[ ]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_svc_linear.fit(X_train_svm, y_train)
print(f"Best Parameters by gridsearch:\t{gridsearch_svc_linear.best_params_}")
print(f"Best Estimator by gridsearch:\t{gridsearch_svc_linear.best_estimator_}")

svc_linear_gs_best = gridsearch_svc_linear.best_estimator_
fun.model_report(svc_linear_gs_best, X_train_svm, y_train, X_test_svm,
                 y_test)
```