1_python_merge (1)

December 16, 2022

1 Import Library

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
warnings.filterwarnings('ignore')
```

2 1 Data Preparation

[3]:
```
#read data
cost = pd.read_csv('data/cost.csv',parse_dates=['Task␣
↪ID'],infer_datetime_format=True)
supplier = pd.read_csv('data/suppliers.csv')
task = pd.read_excel('data/tasks.xlsx',parse_dates=['Task ID'])
```

2.1 1.1 Verify missing value, remove data doesn't match

[4]:
```
#row data
supplier
```

[4]:
```
   Supplier ID   SF1   SF2   SF3  SF4  SF5  SF6  SF7  SF8   SF9  SF10  SF11  \
0           S1   100  1000  1000   50   20   10    2   80  2000   100  1000
1           S2   100  1000  1000   50   20   10    0   80  2000   100  1000
2           S3   100  1000  1000   50   20   10    2   80  2000   100  1000
3           S4   100  1000  1000   50   20   10    2   80  2000   100  1000
4           S5    10  1000   100  500  200   10    2   80   200   100  2000
..         ...   ...   ...   ...  ...  ...  ...  ...  ...   ...   ...   ...
59         S60    10   100   100    5  200   10    2    8   200   100   100
60         S61   100  2000  1000    5    2    1    1    8  4000   500  1000
61         S62  1000  1000  2000    5  200   10    2  500  4000   500   100
```

```
62          S63    100   1000   1000    50   200     1     2     8   4000     10   2000
63          S64    100   2000   1000   500     2     1     3     8   4000    500   1000

    SF12  SF13  SF14  SF15  SF16   SF17  SF18
0      5  1000  1000   500  5000    100    96
1      5  1000  1000   500  5000    100    96
2      5  1000  1000   500  5000      0    96
3      5  1000  1000   500  5000   5000    96
4      8  2000   100  2000  5000  15000    90
..   …     …     …     …     …      …     …
59     8  1000   100    50  5000   7500    98
60     1  1000  2000    50   500   1000    96
61     8  2000  1000   500  5000  15000    90
62     1   100  1000  2000  5000   7500    90
63     5  2000   100    50   500  50000    98

[64 rows x 19 columns]
```

[4]: task

```
[4]:       Task ID  TF1   TF2        TF3   TF4       TF5       TF6   TF7   TF8  \
0    2019-05-30  706  2539  428536374   367  0.144545  35342375  0.08   829
1    2019-09-26  697  2199  389831692   431  0.195998  20091114  0.05   460
2    2019-11-29  262  4156  500027098  1510  0.363330  89708355  0.18  1010
3    2020-01-03  469  4346  547810586  1376  0.316613  90478530  0.17  1097
4    2020-01-07  555  3934  521676289  1039  0.264108  69762831  0.13   943
..          …   …     …          …     …         …         …     …     …
125  2021-12-15    1  3200  490935166  1013  0.316563  54130582  0.11  1213
126  2021-12-16    1  3655  506899785  1422  0.389056  69650824  0.14  1336
127  2021-12-17    1  3721  523237621  1427  0.383499  72353297  0.14  1345
128  2021-12-21    1  3501  535604319  1316  0.375893  72548424  0.14  1314
129  2021-12-22    1  3683  545374101  1508  0.409449  83750259  0.15  1403

          TF9  …  TF107  TF108       TF109       TF110     TF111  TF112  \
0    0.326506  …    125      0   219407.09   903728.98  10174793      0
1    0.209186  …    149      0   467568.10  1868010.80  19221510      0
2    0.243022  …    394      0   228358.20  1358750.92  38657530      0
3    0.252416  …    394      0   258487.96  1781016.69  39064840      0
4    0.239705  …    394      0   285558.89  1869912.30  39064840      0
..        …  …    …      …         …           …         …      …
125  0.379063  …    204      0  2370854.38  8364205.81  66646372      0
126  0.365527  …    259      0  2225977.15  8046625.74  68781077      0
127  0.361462  …    263      0  2331590.62  8339577.56  68777865      0
128  0.375321  …    254      0  2407486.67  8753349.75  69920452      0
129  0.380939  …    212      0  2155307.60  8427018.28  67735822      0

    TF113     TF114     TF115  TF116
```

```
0       801  0.315479  26873722   0.06
1       723  0.328786  20359365   0.05
2       837  0.201396  28562699   0.06
3       706  0.162448  19579305   0.04
4       757  0.192425  24164620   0.05
..      …        …         …     …
125     185  0.057813   8004697   0.02
126     174  0.047606   8210251   0.02
127     141  0.037893   6644987   0.01
128     281  0.080263  14601484   0.03
129     235  0.063807  12060958   0.02

[130 rows x 117 columns]
```

[5]:
```
cost
```

[5]:
```
        Task ID Supplier ID      Cost
0    2019-05-30          S1  0.478219
1    2019-05-30          S2  0.444543
2    2019-05-30          S3  0.521679
3    2019-05-30          S4  0.307331
4    2019-05-30          S5  0.357689
…         …          …        …
7675 2021-12-22         S60  0.410605
7676 2021-12-22         S61  0.410376
7677 2021-12-22         S62  0.407884
7678 2021-12-22         S63  0.420536
7679 2021-12-22         S64  0.423008

[7680 rows x 3 columns]
```

[6]:
```
#no missing value in Task
task.isna().sum().sort_values(ascending=False)
```

[6]:
```
Task ID    0
TF74       0
TF86       0
TF85       0
TF84       0
          ..
TF35       0
TF34       0
TF33       0
TF32       0
TF116      0
Length: 117, dtype: int64
```

[7]:
```python
#no missing value in Cost
cost.isna().sum().sort_values(ascending=False)
```

[7]:
```
Task ID        0
Supplier ID    0
Cost           0
dtype: int64
```

[8]:
```python
#no missing value in supplier
supplier.isna().sum().sort_values(ascending=False)
```

[8]:
```
Supplier ID    0
SF10           0
SF17           0
SF16           0
SF15           0
SF14           0
SF13           0
SF12           0
SF11           0
SF9            0
SF1            0
SF8            0
SF7            0
SF6            0
SF5            0
SF4            0
SF3            0
SF2            0
SF18           0
dtype: int64
```

[9]:
```python
# check if cost ID matches task ID
print('all IDs match? ',set(cost['Task ID']) == set(task['Task ID']))

#check if we have multiple tasks on someday
if(len(set(task)) == len(task)):
    print("One task a day.")
else:
    print("A day has more then one task.")

#set unique group
cost_uni = cost['Task ID'].unique()
task_uni = task['Task ID'].unique()
print('number of tasks in Cost table is:',len(cost_uni))
print('number of tasks in Task table is:',len(task_uni))    #10 more Task Ids␣
 ↪in Task table
```

```
dif_in_datetype = pd.to_datetime(list(set(task_uni) - set(cost_uni)))

#for easily read, print out the ID not in Cost table
dif_in_datetype = dif_in_datetype.strftime('%Y.%m.%d')
print('IDs in task but not in cost are:\n',dif_in_datetype)

#remove the rows that has task ID in Cost but not has one in Task
task = task.loc[~task['Task ID'].isin(dif_in_datetype)]
print('new task dataframe shape after removing task IDs not in Cost',task.
 ↪shape)   # 120,117

print('number of suppliers',len(supplier['Supplier ID'].unique()),
     '\nshapes of Task,Supplier,Cost are',task.shape,supplier.shape,cost.
 ↪shape,'respectively.')
# while 7680 == 120 * 64. which means cost dataframe includes all cost of each␣
 ↪supplier to each task
```

```
all IDs match?  False
A day has more then one task.
number of tasks in Cost table is: 120
number of tasks in Task table is: 130
IDs in task but not in cost are:
 Index(['2020.10.16', '2021.09.16', '2020.10.13', '2021.03.31', '2020.10.21',
       '2021.08.13', '2020.10.09', '2020.03.09', '2020.10.19', '2020.01.28'],
      dtype='object')
new task dataframe shape after removing task IDs not in Cost (120, 117)
number of suppliers 64
shapes of Task,Supplier,Cost are (120, 117) (64, 19) (7680, 3) respectively.
```

2.2 1.2 Min Max variances calculating, and feature with low variances(<0.01) removing

[10]:
```
#feature 'cost' in Cost table
cost_describe = cost['Cost'].describe()
print(cost_describe)
print('variance of cost is:',cost_describe['std']**2)  #greater than 0.01
```

```
count    7680.000000
mean        0.416675
std         0.056270
min         0.280210
25%         0.376758
50%         0.418615
75%         0.450852
max         0.694525
Name: Cost, dtype: float64
```

```
variance of cost is: 0.003166297016947214
```

[11]:
```python
# we can see there are some features in Task with very low variances(<0.01),
↪which can be removed
# as such features remains nearly unchange to each row so are useless
task_des = task.drop(columns='Task ID').describe()
task_des
```

[11]:

	TF1	TF2	TF3	TF4	TF5
count	120.000000	120.000000	1.200000e+02	120.000000	120.000000
mean	83.100000	4436.525000	5.045244e+08	1516.933333	0.336457
std	190.002893	1515.094966	1.120227e+08	684.999758	0.079158
min	0.000000	1127.000000	7.149093e+07	367.000000	0.144471
25%	1.000000	3196.250000	4.434996e+08	997.750000	0.280037
50%	1.000000	4145.000000	5.057017e+08	1386.500000	0.331774
75%	3.000000	5839.500000	5.690280e+08	2036.000000	0.376324
max	706.000000	7908.000000	7.699058e+08	3344.000000	0.642413

	TF6	TF7	TF8	TF9	TF10	…
count	1.200000e+02	120.000000	120.000000	120.000000	1.200000e+02	…
mean	9.266722e+07	0.183750	1348.375000	0.304799	1.208782e+08	…
std	3.950462e+07	0.071185	511.151116	0.047781	3.792731e+07	…
min	2.009111e+07	0.050000	223.000000	0.189399	9.828626e+06	…
25%	6.051022e+07	0.137500	954.750000	0.268808	9.365975e+07	…
50%	8.541451e+07	0.180000	1203.000000	0.307448	1.121406e+08	…
75%	1.210465e+08	0.220000	1578.750000	0.344739	1.508170e+08	…
max	1.906866e+08	0.660000	2444.000000	0.388069	1.976945e+08	…

	TF107	TF108	TF109	TF110	TF111	TF112
count	120.000000	120.0	1.200000e+02	1.200000e+02	1.200000e+02	120.0
mean	263.666667	0.0	1.247831e+06	4.919536e+06	4.587822e+07	0.0
std	196.386060	0.0	8.474757e+05	2.471686e+06	1.392346e+07	0.0
min	0.000000	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.0
25%	155.750000	0.0	5.335741e+05	2.663429e+06	3.815472e+07	0.0
50%	203.500000	0.0	1.071940e+06	4.888876e+06	4.960835e+07	0.0
75%	271.500000	0.0	2.100152e+06	7.542380e+06	5.510589e+07	0.0
max	1244.000000	0.0	3.705798e+06	8.753350e+06	6.992045e+07	0.0

	TF113	TF114	TF115	TF116
count	120.000000	120.000000	1.200000e+02	120.000000
mean	382.808333	0.087880	1.680272e+07	0.034167
std	286.251793	0.062938	1.179023e+07	0.024030
min	30.000000	0.011561	1.031294e+06	0.000000
25%	169.500000	0.042683	7.908316e+06	0.020000
50%	275.500000	0.068094	1.338906e+07	0.030000
75%	512.000000	0.109090	2.222877e+07	0.050000
max	1211.000000	0.328786	5.880575e+07	0.120000

```
[8 rows x 116 columns]
```

[12]:

```python
drop_cols = [task_des.columns[i] for i in range(len(task_des.columns)) if␣
 ↪task_des.loc['std',task_des.columns[i]]**2<0.01]
task.drop(columns=drop_cols,inplace=True)
print('new task shape',task.shape,f'drop {len(drop_cols)} cols:',drop_cols)
task
```

```
new task shape (120, 84) drop 33 cols: ['TF5', 'TF7', 'TF9', 'TF11', 'TF13',
'TF15', 'TF31', 'TF35', 'TF38', 'TF39', 'TF42', 'TF46', 'TF47', 'TF50', 'TF51',
'TF53', 'TF57', 'TF61', 'TF63', 'TF64', 'TF66', 'TF75', 'TF79', 'TF84', 'TF88',
'TF92', 'TF96', 'TF100', 'TF104', 'TF108', 'TF112', 'TF114', 'TF116']
```

[12]:

```
        Task ID  TF1   TF2        TF3   TF4       TF6   TF8       TF10  TF12  \
0    2019-05-30  706  2539  428536374   367  35342375   829  109388318   452
1    2019-09-26  697  2199  389831692   431  20091114   460   72475671   287
2    2019-11-29  262  4156  500027098  1510  89708355  1010  112404944   812
3    2020-01-03  469  4346  547810586  1376  90478530  1097  122674168   834
4    2020-01-07  555  3934  521676289  1039  69762831   943  112768389   682
..          …    …     …          …     …         …     …          …     …
125  2021-12-15    1  3200  490935166  1013  54130582  1213  113855258   641
126  2021-12-16    1  3655  506899785  1422  69650824  1336  118556296   758
127  2021-12-17    1  3721  523237621  1427  72353297  1345  121170804   765
128  2021-12-21    1  3501  535604319  1316  72548424  1314  121685335   725
129  2021-12-22    1  3683  545374101  1508  83750259  1403  124469564   789

         TF14  …        TF102     TF103  TF105  TF106  TF107       TF109  \
0    65396430  …   9836131.71  52077023   6.15  25.10    125   219407.09
1    47355481  …   7342792.32  44103725   9.13  30.13    149   467568.10
2    83864672  …  10064509.99  50354895   5.06  23.75    394   228358.20
3    95043161  …   9122217.84  49212552   5.25  26.07    394   258487.96
4    85212722  …   8664591.29  48980163   5.80  27.35    394   285558.89
..          …  …            …         …      …      …      …           …
125  71871557  …  12679261.73  52010511   8.67  28.97    204  2370854.38
126  75955490  …  13593707.71  54145216   8.65  31.55    259  2225977.15
127  77390849  …  14013439.09  54142004   9.20  33.09    263  2331590.62
128  78239367  …  13593634.49  54226830   8.87  31.63    254  2407486.67
129  80863725  …  12767396.53  53082643   7.56  27.34    212  2155307.60

          TF110     TF111  TF113     TF115
0     903728.98  10174793    801  26873722
1    1868010.80  19221510    723  20359365
2    1358750.92  38657530    837  28562699
3    1781016.69  39064840    706  19579305
4    1869912.30  39064840    757  24164620
..          …         …      …         …
```

```
125  8364205.81  66646372    185   8004697
126  8046625.74  68781077    174   8210251
127  8339577.56  68777865    141   6644987
128  8753349.75  69920452    281  14601484
129  8427018.28  67735822    235  12060958

[120 rows x 84 columns]
```

[13]:
```
#Supplier table doesn't have any column that has variance smaller than 0.01
supplier_describe = supplier.drop(columns='Supplier ID').describe()
supplier_describe.loc['std',:]**2 > 0.01   #question: Do I need to reduce high␣
↪variance?
```

[13]:
```
SF1     True
SF2     True
SF3     True
SF4     True
SF5     True
SF6     True
SF7     True
SF8     True
SF9     True
SF10    True
SF11    True
SF12    True
SF13    True
SF14    True
SF15    True
SF16    True
SF17    True
SF18    True
Name: std, dtype: bool
```

2.3 1.3 Scale the data

[14]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# MinMaxScaler scale to range [0,1], so mutiply it by 2 and minus 1 to turn␣
↪into [-1,1]
# we don't use it in cost dataframe cause it does not need it
task[task.columns.values[1:]] = scaler.fit_transform(task[task.columns.values[1:
↪]])*2 -1
supplier[supplier.columns.values[1:]] = scaler.fit_transform(supplier[supplier.
↪columns.values[1:]])*2 -1
```

2.4 1.4 Calculate and visualize absoluate correlation

[15]:
```
#visualize the absolute correlation across the entire matrix
abs(task.drop(columns='Task ID').corr()).style.background_gradient(cmap="Blues")
```

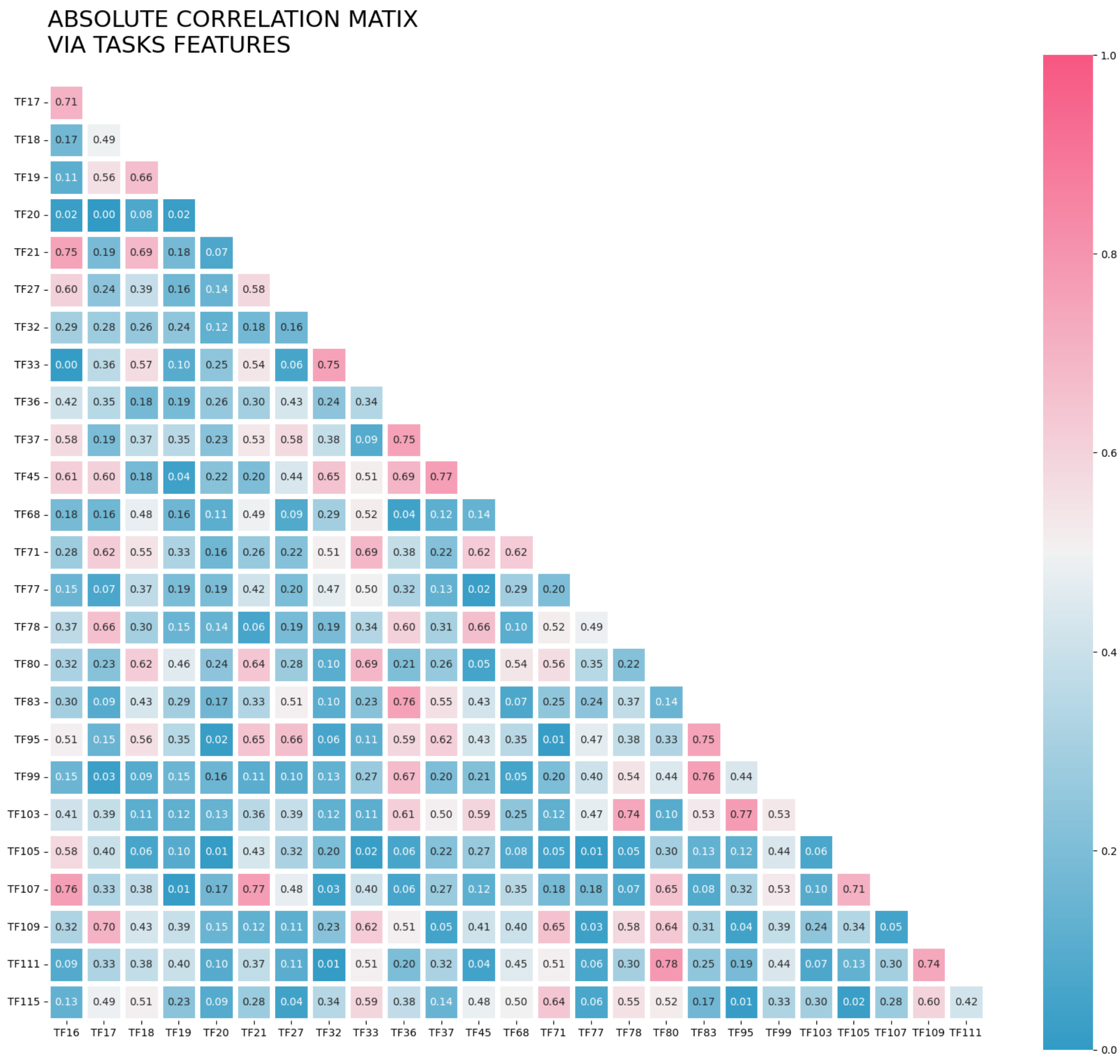
[15]: <pandas.io.formats.style.Styler at 0x1de23c6c250>

[16]:
```
# the correlation here is using Pearson Correlation coeficient
drop_cols = []
while True:
    corr = abs(task.drop(columns='Task ID').corr())
    corr_nums = (corr>=0.8).sum()-1
    corr_cols = corr.columns
    if max(corr_nums) == 0:
        break
    drop_col = corr_cols[np.argmax(corr_nums)]
    task.drop(columns=drop_col,inplace=True)
    drop_cols.append(drop_col)
print('cols drop:',drop_cols)
print(f'drop {len(drop_cols)} columns, now have {task.shape[1]-1} features,task␣
 ↪shape:',task.shape)
```

```
cols drop: ['TF2', 'TF48', 'TF40', 'TF56', 'TF81', 'TF97', 'TF41', 'TF58',
'TF12', 'TF49', 'TF91', 'TF10', 'TF94', 'TF98', 'TF101', 'TF28', 'TF62', 'TF65',
'TF90', 'TF102', 'TF8', 'TF29', 'TF55', 'TF14', 'TF30', 'TF59', 'TF93', 'TF4',
'TF24', 'TF34', 'TF44', 'TF72', 'TF54', 'TF60', 'TF85', 'TF89', 'TF106',
'TF110', 'TF3', 'TF6', 'TF22', 'TF25', 'TF69', 'TF73', 'TF86', 'TF1', 'TF23',
'TF26', 'TF43', 'TF52', 'TF67', 'TF70', 'TF74', 'TF76', 'TF82', 'TF87', 'TF113']
drop 57 columns, now have 26 features,task shape: (120, 27)
```

[17]:
```
#single corner absolute value after drop columns
task_corr = abs(task.drop(columns='Task ID').corr())
fig, ax = plt.subplots(figsize=(20, 30))
# mask
mask = np.triu(np.ones_like(task_corr, dtype=np.bool))
# adjust mask and df
mask = mask[1:, :-1]
corr = task_corr.iloc[1:,:-1].copy()
# color map
cmap = sns.diverging_palette(230, 0, 90, 60, as_cmap=True)
# plot heatmap
sns.heatmap(corr, mask=mask, annot=True, fmt=".2f",
           linewidths=5, cmap=cmap, vmin=0, vmax=1,
           cbar_kws={"shrink": .57}, square=True)
# ticks
yticks = [i.upper() for i in corr.index]
xticks = [i.upper() for i in corr.columns]
```

```
plt.yticks(plt.yticks()[0], labels=yticks, rotation=0)
plt.xticks(plt.xticks()[0], labels=xticks)
# title
title = 'ABSOLUTE CORRELATION MATIX\nVIA TASKS FEATURES\n'
plt.title(title, loc='left', fontsize=22)
plt.show()
```



2.5 1.5 Identify suppliers that never appear in top 20 suppliers in tasks

[18]:
```
cost['rank'] = cost.groupby('Task ID')['Cost'].rank(ascending=True)

good_supplier = set(cost.loc[cost['rank']<=20]['Supplier ID'])
print(f'there are {len(good_supplier)} good suppliers')
```

```
bad_sup = supplier.loc[~supplier['Supplier ID'].isin(good_supplier)]['Supplier␣
↪ID']
print('suppliers that never appear in top 20 suppliers in tasks: ',bad_sup.
↪values)
# remove supplier S2 from all data
cost = cost.loc[~cost['Supplier ID'].isin(bad_sup)]
supplier = supplier.loc[~supplier['Supplier ID'].isin(bad_sup)]
print('After removing, the Cost table shape is',cost.shape,',the Supplier table␣
↪shape is',supplier.shape,'.')
```

```
there are 63 good suppliers
suppliers that never appear in top 20 suppliers in tasks:  ['S2']
After removing, the Cost table shape is (7560, 4) ,the Supplier table shape is
(63, 19) .
```

2.6 1.6 Export data

[19]:
```
task.to_csv('task_after_preperation.csv')
```

[20]:
```
supplier.to_csv('supplier_after_preperation.csv')
```

[21]:
```
cost.to_csv('cost_after_preperation.csv')
```

3 2 Exploratory Data Analysis

3.1 2.1 Distribution of feature values in Task(boxplot)

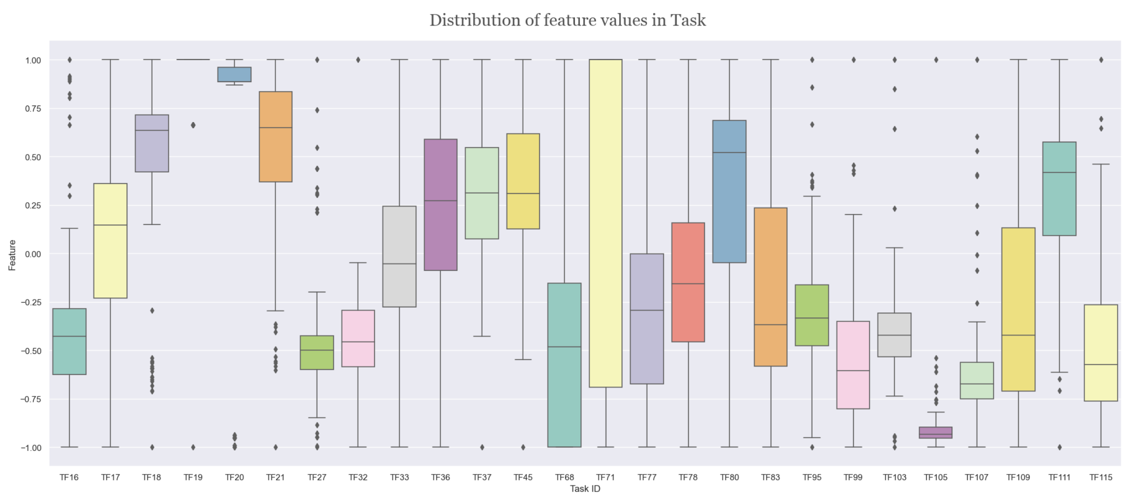
[22]:
```
#create a new dataframe for plot(first remove the 'Task ID' column, then melt␣
↪the dataframe to a long one)
task_for_eda = pd.melt(task.drop('Task ID',axis=1), value_vars=task.iloc[:,1:
↪],value_name='Feature',var_name='Task ID')
task_for_eda
```

[22]:
```
      Task ID   Feature
0        TF16 -0.559193
1        TF16  0.017794
2        TF16 -0.424278
3        TF16 -0.435824
4        TF16 -0.353895
…        …        …
3115   TF115 -0.758599
3116   TF115 -0.751483
3117   TF115 -0.805669
```

```
3118    TF115 -0.530236
3119    TF115 -0.618182

[3120 rows x 2 columns]
```

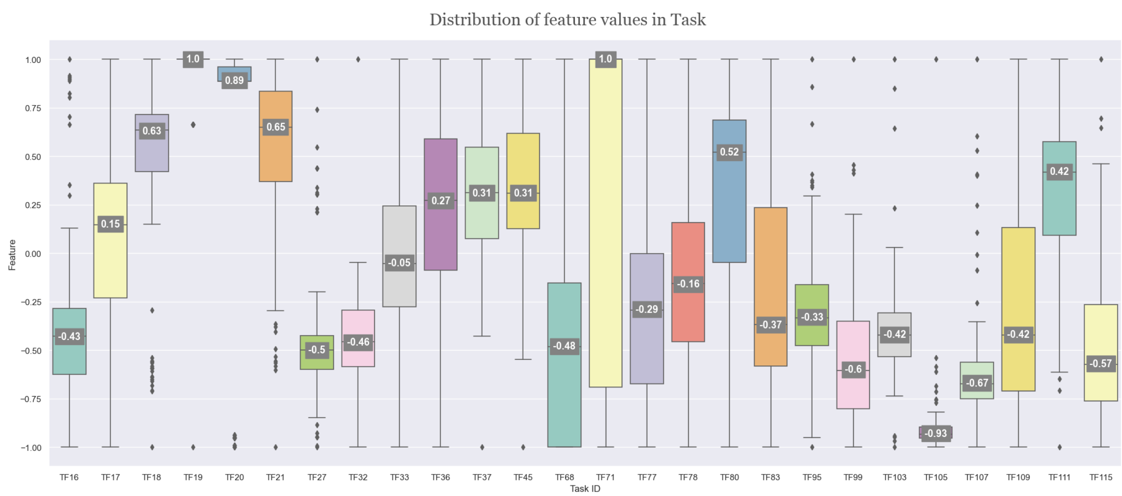
[23]:
```
#Decide to remove the outlier
#Since they aren't adding anything to the relationship we are trying to show␣
 ↪but rather making it difficult to see
sns.set(style='darkgrid')
fig, ax = plt.subplots(figsize=(25,10))
ax = sns.boxplot(x=task_for_eda['Task ID'],
                 y=task_for_eda['Feature'],
                 showfliers=True,
                 palette='Set3',
                 linewidth=1.2)

#set title
font_color = '#525252'
csfont = {'fontname':'Georgia'}
title = 'Distribution of feature values in Task'
fig.suptitle(title, y=.93, fontsize=22, color=font_color, **csfont)
```

[23]:
```
Text(0.5, 0.93, 'Distribution of feature values in Task')
```



[24]:
```
#Another Version
sns.set(style='darkgrid')
fig, ax = plt.subplots(figsize=(25,10))
ax = sns.boxplot(x=task_for_eda['Task ID'],
                 y=task_for_eda['Feature'],
                 showfliers=True,
```

```
                 palette='Set3',
                 linewidth=1.2)

#set title
font_color = '#525252'
csfont = {'fontname':'Georgia'}
title = 'Distribution of feature values in Task'
fig.suptitle(title, y=.93, fontsize=22, color=font_color, **csfont)

#Set Median
#add annotation
lines = ax.get_lines()
categories = ax.get_xticks()
for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],2)
    ax.text(cat,y,f'{y}',va='center',
            ha='center',size=13,color='w',weight='semibold',
        bbox=dict(facecolor='#828282', edgecolor='#828282'))
```



3.2 2.2 Distribution of Errors in Supplier

[5]:
```
#Set 'Task ID' as index for automatic index alignment
cost.index = cost['Task ID']
cost['task_id']=cost.pop('Task ID')
cost.head()
#Caculate the Min cost of each task
min_cost_byTask = pd.DataFrame(cost.groupby('task_id')['Cost'].min())
#Compute the error(the cost of chose supplier minus the Min cost of that task)
cost['error'] = pd.DataFrame(cost['Cost']-min_cost_byTask['Cost'])
```

```
cost['error_sqr'] = cost['error']**2
cost
```

[5]:
```
             Supplier ID      Cost     task_id     error  error_sqr
Task ID
2019-05-30            S1  0.478219  2019-05-30  0.187252   0.035063
2019-05-30            S2  0.444543  2019-05-30  0.153576   0.023585
2019-05-30            S3  0.521679  2019-05-30  0.230711   0.053228
2019-05-30            S4  0.307331  2019-05-30  0.016363   0.000268
2019-05-30            S5  0.357689  2019-05-30  0.066722   0.004452
…                    …       …         …          …         …
2021-12-22           S60  0.410605  2021-12-22  0.032737   0.001072
2021-12-22           S61  0.410376  2021-12-22  0.032508   0.001057
2021-12-22           S62  0.407884  2021-12-22  0.030016   0.000901
2021-12-22           S63  0.420536  2021-12-22  0.042668   0.001821
2021-12-22           S64  0.423008  2021-12-22  0.045140   0.002038

[7680 rows x 5 columns]
```

[26]:
```
#Create a figure and subplot
sns.set(style='darkgrid')
fig, ax = plt.subplots(figsize=(30,12))
#Create a boxplot
ax = sns.boxplot(x=cost['Supplier ID'],
                 y=cost['error'],
                 showfliers=True,
                 palette='Set3',
                 linewidth=1.2)
#Create title and set font's parameters
font_color = '#525252'
csfont = {'fontname':'Georgia'}
title = 'Distribution of Errors in Supplier'
fig.suptitle(title, y=.93, fontsize=22, color=font_color, **csfont)
```

[26]:
```
Text(0.5, 0.93, 'Distribution of Errors in Supplier')
```

[6]:
```
#Annotate each boxplot with the RMSE of each supplier for all tasks.
#RMSE: Root Mean Squared Error

#Compute RMSE for each upplier
sum_of_each_supplier_erro = cost.groupby('Supplier ID')['error_sqr'].sum()
tasks_numbers_of_each_supplier_has = cost.groupby('Supplier ID').size()
supplierRMSE = pd.DataFrame(np.sqrt(sum_of_each_supplier_erro/
 ↪tasks_numbers_of_each_supplier_has))
supplierRMSE
```

[6]:

	0
Supplier ID	
S1	0.081803
S10	0.037460
S11	0.033735
S12	0.048724
S13	0.039545
…	…
S63	0.050874
S64	0.040125
S7	0.045649
S8	0.053460
S9	0.051150

[64 rows x 1 columns]

[28]:
```
#Rename the column 'error' to 'RMSE'
supplierRMSE.rename(columns = {'error':'RMSE'}, inplace = True)
#The index sorted by string(eg.S2 is greater than S10)
#Set Index as a new column 'index'
```

```
supplierRMSE['index'] = supplierRMSE.index
#Get the number of 'index' column
supplierRMSE[['index']] = supplierRMSE[['index']].applymap(lambda x: int(x[1:]))
#Sort by 'index'
supplierRMSE.sort_values(by='index')
supplierRMSE = supplierRMSE.sort_values(by='index')
supplierRMSE
```

[28]:

```
                    0  index
Supplier ID
S1           0.267227      1
S3           0.311898      3
S4           0.181398      4
S5           0.219353      5
S6           0.200732      6
…                 …    …
S60          0.200927     60
S61          0.243796     61
S62          0.235453     62
S63          0.217579     63
S64          0.183294     64

[63 rows x 2 columns]
```

[29]:

```
#Annotate each boxplot with the RMSE of each supplier for all tasks.
#RMSE: Root Mean Squared Error
#Create a figure and subplot
sns.set(style='darkgrid')
fig, ax = plt.subplots(figsize=(30,12))
#Create a boxplot
ax = sns.boxplot(x=cost['Supplier ID'],
                 y=cost['error'],
                 showfliers=True,
                 palette='Set3',
                 linewidth=1.2)
#Create title and set font's parameters
font_color = '#525252'
csfont = {'fontname':'Georgia'}
title = 'Distribution of Errors in Supplier'
fig.suptitle(title, y=.93, fontsize=22, color=font_color, **csfont)

#add annotation
lines = ax.get_lines()
categories = ax.get_xticks()
for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],5)
```

```
    ax.text(cat,y+0.0045,round(supplierRMSE[0][cat],2),va='center',  #add␣
 ↪annotation a bit above the median line
            ha='center',size='small',color='b',weight='semibold')

#add the meaning of the number
ax.text(54,0.23,'Numbers on each boxplot mean the RMSE of each supplier\nRMSE:␣
 ↪Root Mean Squared Error',va='center',
            ha='center',size='large',color='gray',weight='semibold')
```

[29]:

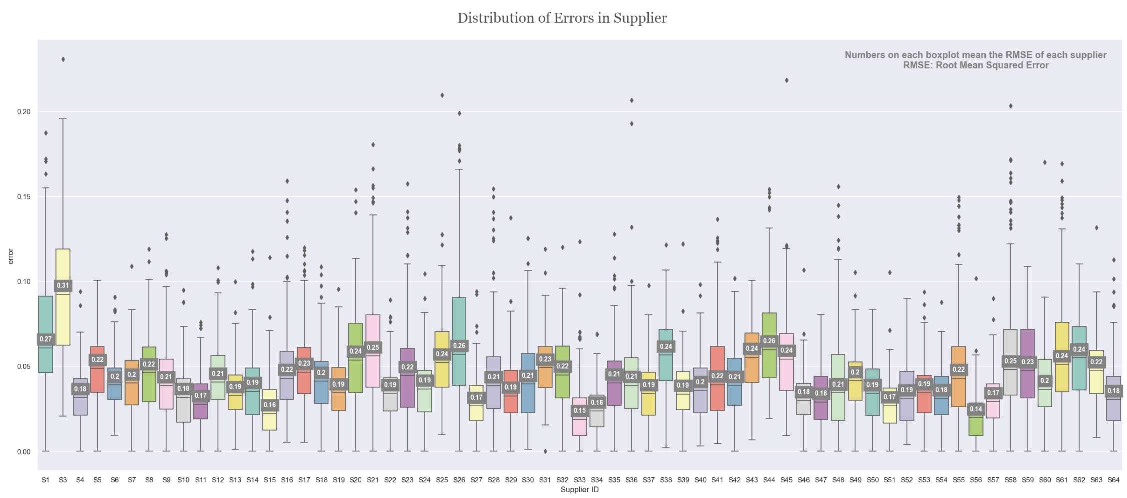
```
Text(54, 0.23, 'Numbers on each boxplot mean the RMSE of each supplier\nRMSE:
Root Mean Squared Error')
```

[30]:

```
#Another version
sns.set(style='darkgrid')
fig, ax = plt.subplots(figsize=(30,12))
#Create a boxplot
ax = sns.boxplot(x=cost['Supplier ID'],
                 y=cost['error'],
                 showfliers=True,
                 palette='Set3',
                 linewidth=1.2)
#Create title and set font's parameters
font_color = '#525252'
csfont = {'fontname':'Georgia'}
title = 'Distribution of Errors in Supplier'
fig.suptitle(title, y=.93, fontsize=22, color=font_color, **csfont)

#add annotation
lines = ax.get_lines()
categories = ax.get_xticks()
```

```python
for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],5)
    ax.text(cat,y+0.005,round(supplierRMSE[0][cat],2),va='center',  #add annotation a bit above the median line
            ha='center',size=10,color='w',weight='semibold',
        bbox=dict(facecolor='#828282', edgecolor='#828282'))

#add the meaning of the number
ax.text(54,0.23,'Numbers on each boxplot mean the RMSE of each supplier\nRMSE: Root Mean Squared Error',va='center',
            ha='center',size='large',color='gray',weight='semibold')
```

[30]: Text(54, 0.23, 'Numbers on each boxplot mean the RMSE of each supplier\nRMSE:
Root Mean Squared Error')



3.3 2.3 Heatmap

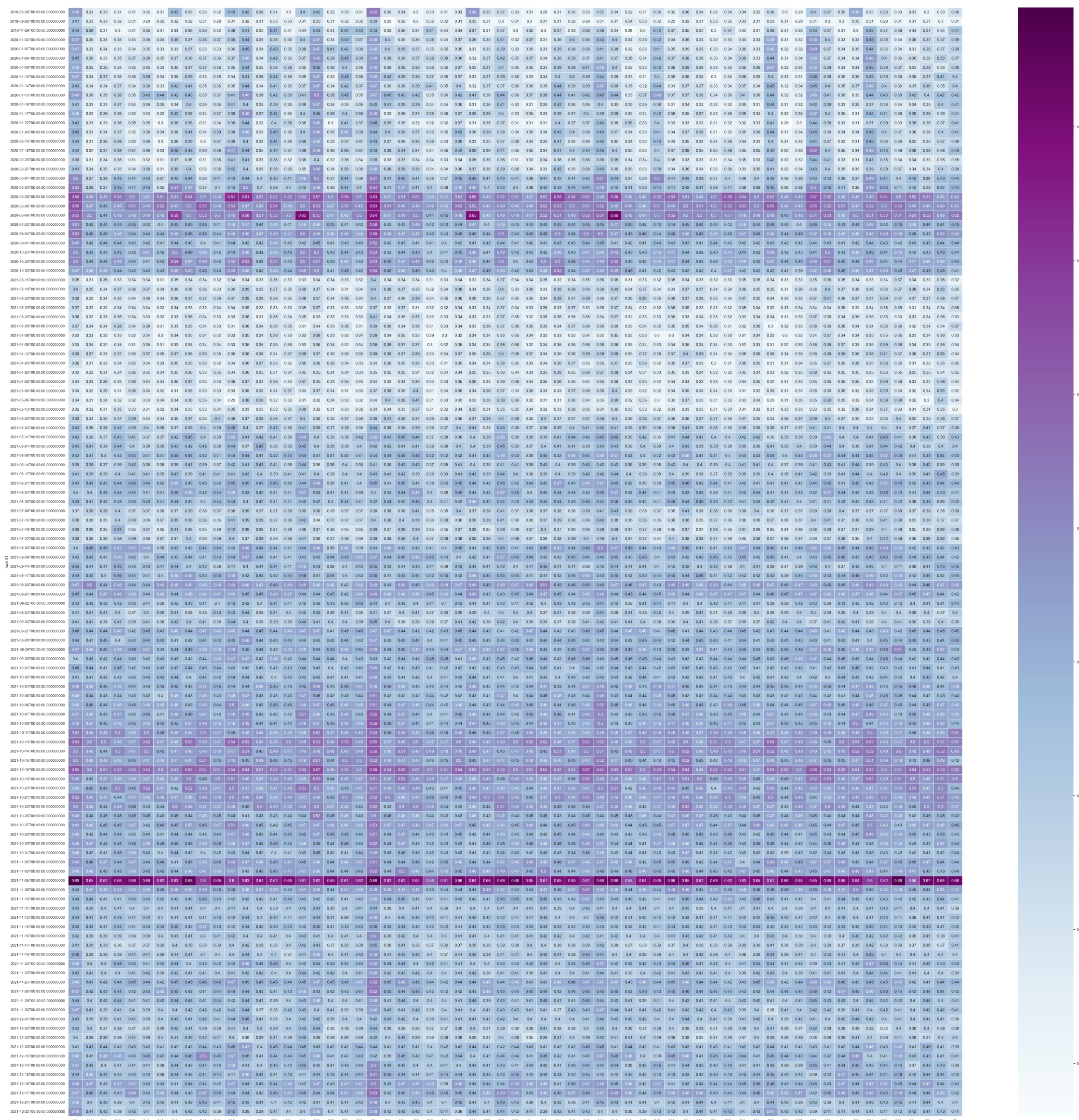
[31]:
```python
cost
```

[31]:
```
           Supplier ID      Cost  rank    task_id     error
Task ID
2019-05-30          S1  0.478219  62.0 2019-05-30  0.187252
2019-05-30          S3  0.521679  64.0 2019-05-30  0.230711
2019-05-30          S4  0.307331   7.0 2019-05-30  0.016363
2019-05-30          S5  0.357689  47.0 2019-05-30  0.066722
2019-05-30          S6  0.351982  43.0 2019-05-30  0.061014
…                 …        …   …        …        …
2021-12-22         S60  0.410605  27.0 2021-12-22  0.032737
2021-12-22         S61  0.410376  26.0 2021-12-22  0.032508
```

```
2021-12-22         S62  0.407884  24.0 2021-12-22  0.030016
2021-12-22         S63  0.420536  42.0 2021-12-22  0.042668
2021-12-22         S64  0.423008  48.0 2021-12-22  0.045140

[7560 rows x 5 columns]
```

[32]:
```python
df_heatmap = cost.iloc[:,:2].reset_index().pivot("Task ID", "Supplier ID",␣
 ↪"Cost")

plt.figure(figsize=(60,60))
sns.heatmap(df_heatmap, annot=True, linewidth=.5, cmap = "BuPu")
```

[32]:
```
<AxesSubplot: xlabel='Supplier ID', ylabel='Task ID'>
```



4 Model

4.1 3.1 Connect Tables

4.2 things to note:

1. Here we transform the dataframe into number index as it is more convenient for training
2. 'error' col in cost dataframe must be dropped as it leak info from Cost (which we will predict), and 'rank' either because rank depends on 'Cost'
3. try avoiding using DataFrame.append(), cause this can easily break original data types in DataFrame

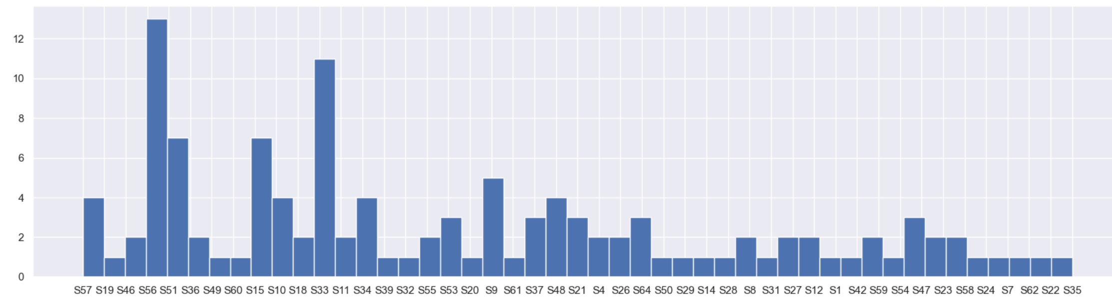
[33]:
```
cost = cost.reset_index()
cost
```

[33]:
```
         Task ID Supplier ID      Cost  rank     task_id     error
0     2019-05-30          S1  0.478219  62.0 2019-05-30  0.187252
1     2019-05-30          S3  0.521679  64.0 2019-05-30  0.230711
2     2019-05-30          S4  0.307331   7.0 2019-05-30  0.016363
3     2019-05-30          S5  0.357689  47.0 2019-05-30  0.066722
4     2019-05-30          S6  0.351982  43.0 2019-05-30  0.061014
…            …         …        …     …         …          …
7555 2021-12-22         S60  0.410605  27.0 2021-12-22  0.032737
7556 2021-12-22         S61  0.410376  26.0 2021-12-22  0.032508
7557 2021-12-22         S62  0.407884  24.0 2021-12-22  0.030016
7558 2021-12-22         S63  0.420536  42.0 2021-12-22  0.042668
7559 2021-12-22         S64  0.423008  48.0 2021-12-22  0.045140

[7560 rows x 6 columns]
```

[34]:
```
# from the diagram we can see the suppliers best chosen for each task␣
↪distributed unevenly, many suppliers never be chosen as best, and some␣
↪suppliers are chosen quite frequently, this may indicate suitable for␣
↪fitting a
# classification model to choose the best supplier and to calculate the RMSE␣
↪between them. But unfortunately this methods performs badly probably due to␣
↪very small number of training samples and large number of categories
# to be classified
supplier_chosen = cost[cost['rank']==1]
print(len(supplier_chosen['Supplier ID'].unique()),'unique suppliers appear')
supplier_chosen['Supplier ID'].hist(bins=len(supplier_chosen['Supplier ID'].
↪unique()),figsize=(20,5))
```

```
47 unique suppliers appear
```

[34]:
```
<AxesSubplot: >
```



[35]:
```python
supplier
```

[35]:
```
   Supplier ID       SF1       SF2       SF3       SF4       SF5       SF6  \
0           S1 -0.818182 -0.052632 -0.052632 -0.818182 -0.818182 -0.818182
2           S3 -0.818182 -0.052632 -0.052632 -0.818182 -0.818182 -0.818182
3           S4 -0.818182 -0.052632 -0.052632 -0.818182 -0.818182 -0.818182
4           S5 -1.000000 -0.052632 -1.000000  1.000000  1.000000 -0.818182
5           S6 -1.000000 -1.000000  1.000000 -1.000000  1.000000  1.000000
..         …        …        …        …        …        …        …
59         S60 -1.000000 -1.000000 -1.000000 -1.000000  1.000000 -0.818182
60         S61 -0.818182  1.000000 -0.052632 -1.000000 -1.000000 -1.000000
61         S62  1.000000 -0.052632  1.000000 -1.000000  1.000000 -0.818182
62         S63 -0.818182 -0.052632 -0.052632 -0.818182  1.000000 -1.000000
63         S64 -0.818182  1.000000 -0.052632  1.000000 -1.000000 -1.000000

         SF7       SF8       SF9      SF10      SF11      SF12      SF13  \
0   0.333333 -0.707317 -0.052632 -0.632653 -0.052632  0.142857 -0.052632
2   0.333333 -0.707317 -0.052632 -0.632653 -0.052632  0.142857 -0.052632
3   0.333333 -0.707317 -0.052632 -0.632653 -0.052632  0.142857 -0.052632
4   0.333333 -0.707317 -1.000000 -0.632653  1.000000  1.000000  1.000000
5  -0.333333 -1.000000  1.000000 -1.000000  1.000000 -1.000000 -1.000000
..       …        …        …        …        …        …        …
59  0.333333 -1.000000 -1.000000 -0.632653 -1.000000  1.000000 -0.052632
60 -0.333333 -1.000000  1.000000  1.000000 -0.052632 -1.000000 -0.052632
61  0.333333  1.000000  1.000000  1.000000 -1.000000  1.000000  1.000000
62  0.333333 -1.000000  1.000000 -1.000000  1.000000 -1.000000 -1.000000
63  1.000000 -1.000000  1.000000  1.000000 -0.052632  0.142857  1.000000

        SF14      SF15  SF16   SF17  SF18
0  -0.052632 -0.538462   0.2 -0.996   0.5
2  -0.052632 -0.538462   0.2 -1.000   0.5
3  -0.052632 -0.538462   0.2 -0.800   0.5
```

```
4   -1.000000  1.000000   0.2 -0.400  -1.0
5    1.000000 -0.538462   0.2 -0.700  -1.0
..        …         …    …     …     …
59  -1.000000 -1.000000   0.2 -0.700   1.0
60   1.000000 -1.000000  -1.0 -0.960   0.5
61  -0.052632 -0.538462   0.2 -0.400  -1.0
62  -0.052632  1.000000   0.2 -0.700  -1.0
63  -1.000000 -1.000000  -1.0  1.000   1.0

[63 rows x 19 columns]
```

[36]:

```python
df = cost.drop(columns=['rank','task_id','error'])
df = df.merge(task,on='Task ID',how='left')
df = df.merge(supplier,on='Supplier ID',how='left')
df.rename(columns={'Task ID':'taskid'},inplace=True) # rename for easier coding
df
```

[36]:

```
          taskid Supplier ID      Cost      TF16      TF17      TF18  TF19  \
0     2019-05-30          S1  0.478219 -0.559193 -0.350672 -0.294403   1.0
1     2019-05-30          S3  0.521679 -0.559193 -0.350672 -0.294403   1.0
2     2019-05-30          S4  0.307331 -0.559193 -0.350672 -0.294403   1.0
3     2019-05-30          S5  0.357689 -0.559193 -0.350672 -0.294403   1.0
4     2019-05-30          S6  0.351982 -0.559193 -0.350672 -0.294403   1.0
…            …         …          …         …         …         …    …
7555  2021-12-22         S60  0.410605 -0.507552  0.379247  0.726832   1.0
7556  2021-12-22         S61  0.410376 -0.507552  0.379247  0.726832   1.0
7557  2021-12-22         S62  0.407884 -0.507552  0.379247  0.726832   1.0
7558  2021-12-22         S63  0.420536 -0.507552  0.379247  0.726832   1.0
7559  2021-12-22         S64  0.423008 -0.507552  0.379247  0.726832   1.0

          TF20      TF21      TF27  …       SF9      SF10      SF11  \
0     0.961792 -0.379070 -0.599289  … -0.052632 -0.632653 -0.052632
1     0.961792 -0.379070 -0.599289  … -0.052632 -0.632653 -0.052632
2     0.961792 -0.379070 -0.599289  … -0.052632 -0.632653 -0.052632
3     0.961792 -0.379070 -0.599289  … -1.000000 -0.632653  1.000000
4     0.961792 -0.379070 -0.599289  …  1.000000 -1.000000  1.000000
…          …         …         …   …       …         …         …
7555 -0.951252  0.701413 -0.198464  … -1.000000 -0.632653 -1.000000
7556 -0.951252  0.701413 -0.198464  …  1.000000  1.000000 -0.052632
7557 -0.951252  0.701413 -0.198464  …  1.000000  1.000000 -1.000000
7558 -0.951252  0.701413 -0.198464  …  1.000000 -1.000000  1.000000
7559 -0.951252  0.701413 -0.198464  …  1.000000  1.000000 -0.052632

          SF12      SF13      SF14      SF15  SF16   SF17  SF18
0     0.142857 -0.052632 -0.052632 -0.538462   0.2 -0.996   0.5
1     0.142857 -0.052632 -0.052632 -0.538462   0.2 -1.000   0.5
2     0.142857 -0.052632 -0.052632 -0.538462   0.2 -0.800   0.5
```

```
3     1.000000  1.000000 -1.000000  1.000000   0.2 -0.400  -1.0
4    -1.000000 -1.000000  1.000000 -0.538462   0.2 -0.700  -1.0
…         …         …         …         …    …     …    …
7555  1.000000 -0.052632 -1.000000 -1.000000   0.2 -0.700   1.0
7556 -1.000000 -0.052632  1.000000 -1.000000  -1.0 -0.960   0.5
7557  1.000000  1.000000 -0.052632 -0.538462   0.2 -0.400  -1.0
7558 -1.000000 -1.000000 -0.052632  1.000000   0.2 -0.700  -1.0
7559  0.142857  1.000000 -1.000000 -1.000000  -1.0  1.000   1.0

[7560 rows x 47 columns]
```

4.3 3.2 split dataset by group indexes

[37]:
```
import torch
import os
import random
def set_seed(seed=42):
    os.environ["PYTHONASHSEED"] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.benchmark = False
    torch.backends.cudnn.deterministic = True
seed = 42
set_seed(42)
```

[38]:
```
groups = df.taskid.unique()
TestGroup = np.random.choice(groups,size=20,replace=False)
print('selected test group indexes',TestGroup)
```

```
selected test group indexes ['2021-05-20T00:00:00.000000000'
'2021-06-01T00:00:00.000000000'
 '2020-01-07T00:00:00.000000000' '2021-07-15T00:00:00.000000000'
 '2020-10-23T00:00:00.000000000' '2021-09-22T00:00:00.000000000'
 '2021-10-04T00:00:00.000000000' '2020-01-16T00:00:00.000000000'
 '2021-04-28T00:00:00.000000000' '2021-11-29T00:00:00.000000000'
 '2020-03-01T00:00:00.000000000' '2021-09-20T00:00:00.000000000'
 '2020-01-17T00:00:00.000000000' '2021-04-08T00:00:00.000000000'
 '2021-10-28T00:00:00.000000000' '2021-10-31T00:00:00.000000000'
 '2021-12-01T00:00:00.000000000' '2019-05-30T00:00:00.000000000'
 '2021-10-27T00:00:00.000000000' '2021-11-23T00:00:00.000000000']
```

[39]:
```python
# training sets are extracted by the 100 group ids not in TestGroup
# test sets are extracted by the 20 group ids in TestGroup
def split20(df,TestGroup): # split by 20 TestGroup
    X_train_all = df.loc[~df['taskid'].isin(TestGroup)]
    X_train = X_train_all.drop(columns=['Cost','taskid','Supplier ID'])
    y_train = X_train_all.Cost
    X_test_all = df.loc[df['taskid'].isin(TestGroup)]
    X_test = X_test_all.drop(columns=['Cost','taskid','Supplier ID'])
    y_test = X_test_all.Cost
    print('X_train,y_train,X_test,y_test shapes',X_train.shape,y_train.
 ↪shape,X_test.shape,y_test.shape)
    return X_train_all,X_train,y_train,X_test_all,X_test,y_test
X_train_all,X_train,y_train,X_test_all,X_test,y_test = split20(df,TestGroup)
```

```
X_train,y_train,X_test,y_test shapes (6300, 44) (6300,) (1260, 44) (1260,)
```

[40]:
```python
X_train
```

[40]:
```
          TF16      TF17      TF18      TF19      TF20      TF21      TF27  \
63    0.017794 -0.560807 -1.000000  0.662732  0.961792 -1.000000 -0.599289
64    0.017794 -0.560807 -1.000000  0.662732  0.961792 -1.000000 -0.599289
65    0.017794 -0.560807 -1.000000  0.662732  0.961792 -1.000000 -0.599289
66    0.017794 -0.560807 -1.000000  0.662732  0.961792 -1.000000 -0.599289
67    0.017794 -0.560807 -1.000000  0.662732  0.961792 -1.000000 -0.599289
…          …         …         …         …         …         …         …
7555 -0.507552  0.379247  0.726832  1.000000 -0.951252  0.701413 -0.198464
7556 -0.507552  0.379247  0.726832  1.000000 -0.951252  0.701413 -0.198464
7557 -0.507552  0.379247  0.726832  1.000000 -0.951252  0.701413 -0.198464
7558 -0.507552  0.379247  0.726832  1.000000 -0.951252  0.701413 -0.198464
7559 -0.507552  0.379247  0.726832  1.000000 -0.951252  0.701413 -0.198464

         TF32      TF33      TF36  …       SF9      SF10      SF11  \
63   -1.00000 -1.000000 -0.615202  … -0.052632 -0.632653 -0.052632
64   -1.00000 -1.000000 -0.615202  … -0.052632 -0.632653 -0.052632
65   -1.00000 -1.000000 -0.615202  … -0.052632 -0.632653 -0.052632
66   -1.00000 -1.000000 -0.615202  … -1.000000 -0.632653  1.000000
67   -1.00000 -1.000000 -0.615202  …  1.000000 -1.000000  1.000000
…         …         …         …  …         …         …         …
7555 -0.51879  0.008934  0.353919  … -1.000000 -0.632653 -1.000000
7556 -0.51879  0.008934  0.353919  …  1.000000  1.000000 -0.052632
7557 -0.51879  0.008934  0.353919  …  1.000000  1.000000 -1.000000
7558 -0.51879  0.008934  0.353919  …  1.000000 -1.000000  1.000000
7559 -0.51879  0.008934  0.353919  …  1.000000  1.000000 -0.052632

          SF12      SF13      SF14      SF15  SF16   SF17  SF18
63    0.142857 -0.052632 -0.052632 -0.538462   0.2 -0.996   0.5
64    0.142857 -0.052632 -0.052632 -0.538462   0.2 -1.000   0.5
```

```
65    0.142857 -0.052632 -0.052632 -0.538462   0.2 -0.800   0.5
66    1.000000  1.000000 -1.000000  1.000000   0.2 -0.400  -1.0
67   -1.000000 -1.000000  1.000000 -0.538462   0.2 -0.700  -1.0
…        …         …         …         …     …     …     …
7555  1.000000 -0.052632 -1.000000 -1.000000   0.2 -0.700   1.0
7556 -1.000000 -0.052632  1.000000 -1.000000  -1.0 -0.960   0.5
7557  1.000000  1.000000 -0.052632 -0.538462   0.2 -0.400  -1.0
7558 -1.000000 -1.000000 -0.052632  1.000000   0.2 -0.700  -1.0
7559  0.142857  1.000000 -1.000000 -1.000000  -1.0  1.000   1.0

[6300 rows x 44 columns]
```

4.4 3.3 train and score the models

1. Apart from using Ridge regression that is required to do in our group, we also evaluate all the other models mentioned on the coursework specification, in addition, we introduce xgboost, catboost, lightgbm here because methods of decision tree are more suitable for tabular than neural network, in terms of training speed, model size and performance, so we evaluate these three popular and advanced algorithms.

[41]:
```
from sklearn.linear_model import Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import␣
↪RandomForestRegressor,GradientBoostingRegressor,ExtraTreesRegressor
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
models = {'ridge':Ridge(random_state=seed),
         'lasso':Lasso(random_state=seed),
         'svr':SVR(),
         'decision_tree':DecisionTreeRegressor(random_state=seed),
         'random_forest':RandomForestRegressor(random_state=seed),
         'gradient_boost':GradientBoostingRegressor(random_state=seed),
         'extraTree':ExtraTreesRegressor(random_state=seed),
         'mlp':MLPRegressor(random_state=seed),
         'xgb':XGBRegressor(random_state=seed,tree_method='gpu_hist',gpu_id=0),
         'lgbm':LGBMRegressor(random_state=seed,device_type='gpu'),
         'cat':CatBoostRegressor(random_state=seed,verbose=False)}
```

[42]:
```
def train_models(models,X_train,y_train,X_test,y_test):
    times = []
    scores = []
    names = []
    for key in models.keys():
```

```
        print('fitting model:',key)
        model = models[key]
        names.append(key)
        start = time.time()
        model.fit(X_train,y_train)
        times.append(time.time()-start)
        scores.append(model.score(X_test,y_test))
    results = pd.DataFrame({'models':names,'time for fitting':times,'r2 scores':
 ↪scores})
    results.sort_values('r2 scores',ascending=False,inplace=True)
    return results
results = train_models(models,X_train,y_train,X_test,y_test)
results.to_csv('results_train100Group.csv')
results
```

```
fitting model: ridge
fitting model: lasso
fitting model: svr
fitting model: decision_tree
fitting model: random_forest
fitting model: gradient_boost
fitting model: extraTree
fitting model: mlp
fitting model: xgb
fitting model: lgbm
fitting model: cat
```

[42]:

	models	time for fitting	r2 scores
9	lgbm	0.760000	0.732255
4	random_forest	4.082638	0.706287
6	extraTree	3.504490	0.704255
10	cat	1.969002	0.697089
5	gradient_boost	1.261089	0.689984
8	xgb	0.598998	0.682105
0	ridge	0.007091	0.552317
3	decision_tree	0.068001	0.540076
2	svr	0.042001	0.001454
1	lasso	0.004999	-0.097891
7	mlp	0.442491	-1.428451

4.5 3.4 calucalte equation (1) and equation(2) of Test Group

4.5.1 error function

4.5.2 score function

[43]:
```
def evaluate(models,X_test_all,X_test,):
    true_test = X_test_all.groupby('taskid')['Cost'].min().reset_index()
    rmse_dict = {}
    for key in models.keys():
        model = models[key]
        X_test_all['pred']= models[key].predict(X_test)
        pred_test = X_test_all.groupby('taskid')['pred'].min().reset_index()
        pred_test.columns = ['taskid',key+'_pred']
        true_test = true_test.merge(pred_test,on='taskid')
        true_test[key+'_error'] = true_test['Cost'] - true_test[key+'_pred']
        true_test.drop(columns=key+'_pred',inplace=True)
        rmse_dict[key] = round(  np.sqrt((true_test[key+'_error']**2).mean()),␣
↪6)
    rmse_df = pd.DataFrame({'model':rmse_dict.keys(),'RMSE':rmse_dict.
↪values()}).sort_values('RMSE',ascending=True)
    return true_test,rmse_df
true_test,rmse_df = evaluate(models,X_test_all,X_test)
rmse_df
```

[43]:

	model	RMSE
3	decision_tree	0.016000
6	extraTree	0.021814
8	xgb	0.022095
4	random_forest	0.024301
10	cat	0.025238
9	lgbm	0.025735
5	gradient_boost	0.033552
0	ridge	0.040115
7	mlp	0.046557
2	svr	0.061460
1	lasso	0.070945

[44]:
```
def evaluate(models,X_test_all,X_test,):
    true_test = X_test_all.groupby('taskid')['Cost'].min().reset_index()
    rmse_dict = {}
    for key in models.keys():
        model = models[key]
        X_test_all['pred']= models[key].predict(X_test)
        pred_test = X_test_all.groupby('taskid')['pred'].min().reset_index()
        pred_test.columns = ['taskid',key+'_pred']
        true_test = true_test.merge(pred_test,on='taskid')
```

```
        true_test[key+'_error'] = true_test['Cost'] - true_test[key+'_pred']
        true_test.drop(columns=key+'_pred',inplace=True)
        rmse_dict[key] = round(  np.sqrt((true_test[key+'_error']**2).mean()),␣
↪6)
    rmse_df = pd.DataFrame({'model':rmse_dict.keys(),'RMSE':rmse_dict.
↪values()}).sort_values('RMSE',ascending=True)
    return true_test,rmse_df
true_test,rmse_df = evaluate(models,X_test_all,X_test)
true_test
```

[44]:

	taskid	Cost	ridge_error	lasso_error	svr_error
0	2019-05-30	0.290967	-0.103230	-0.126900	-0.103494
1	2020-01-07	0.316962	-0.037268	-0.100905	-0.065871
2	2020-01-16	0.309074	-0.060693	-0.108793	-0.068784
3	2020-01-17	0.309255	-0.046331	-0.108612	-0.068144
4	2020-03-01	0.355652	-0.038231	-0.062214	-0.036585
5	2020-10-23	0.403649	-0.053422	-0.014218	-0.054718
6	2021-04-08	0.296250	-0.046994	-0.121617	-0.088773
7	2021-04-28	0.313595	-0.004519	-0.104272	-0.058121
8	2021-05-20	0.333052	-0.038124	-0.084815	-0.069009
9	2021-06-01	0.371296	-0.031226	-0.046571	-0.062184
10	2021-07-15	0.343757	-0.017107	-0.074110	-0.051692
11	2021-09-20	0.415949	0.000181	-0.001918	-0.013515
12	2021-09-22	0.383797	-0.029056	-0.034070	-0.051348
13	2021-10-04	0.420855	-0.005223	0.002988	-0.022335
14	2021-10-27	0.419289	-0.021852	0.001422	-0.039631
15	2021-10-28	0.411130	-0.029340	-0.006737	-0.050686
16	2021-10-31	0.389170	-0.029995	-0.028697	-0.070831
17	2021-11-23	0.379568	-0.035839	-0.038298	-0.071179
18	2021-11-29	0.388197	-0.017587	-0.029670	-0.055455
19	2021-12-01	0.375877	-0.018421	-0.041990	-0.060109

	decision_tree_error	random_forest_error	gradient_boost_error
0	-0.015229	-0.035454	-0.044020
1	0.006365	-0.000449	-0.018999
2	0.002878	-0.012536	-0.030613
3	0.029045	0.006409	-0.029960
4	0.012827	-0.027866	-0.025376
5	0.015476	-0.013867	-0.028015
6	0.007320	-0.016210	-0.028788
7	0.010411	0.001414	-0.013135
8	-0.016452	-0.020962	-0.017943
9	0.013223	-0.004790	-0.019191
10	-0.005747	-0.011999	-0.024425
11	0.016263	-0.013814	-0.002710
12	-0.012966	-0.044888	-0.042197
13	0.024286	0.002877	-0.022479

14	0.022224	-0.009882	-0.021513
15	-0.005301	-0.019171	-0.035029
16	-0.010999	-0.048589	-0.068533
17	-0.026435	-0.043756	-0.056144
18	-0.015616	-0.035394	-0.040102
19	0.018515	-0.017563	-0.034980

	extraTree_error	mlp_error	xgb_error	lgbm_error	cat_error
0	-0.059421	-0.043410	-0.031684	-0.032012	-0.039067
1	0.004596	0.004891	0.007048	-0.005680	0.000335
2	0.001352	0.003390	-0.005839	-0.008251	-0.005448
3	0.004917	-0.018077	-0.002774	-0.004252	-0.008209
4	0.003604	-0.072660	-0.045032	-0.035450	0.009229
5	-0.005922	0.072766	-0.008554	-0.017168	-0.009158
6	-0.010086	0.097905	-0.004938	-0.016611	-0.032271
7	0.002048	0.076653	0.008799	-0.000792	-0.002593
8	-0.015778	-0.009403	0.000763	-0.019940	-0.010402
9	-0.005445	0.013060	-0.012940	-0.010178	-0.008664
10	-0.014761	0.070034	-0.006087	-0.011180	-0.021732
11	0.016663	0.054486	0.001784	-0.004610	0.001315
12	-0.032100	0.010816	-0.037253	-0.039996	-0.044516
13	-0.008049	0.008959	-0.017991	-0.014845	-0.020311
14	-0.013204	0.049475	-0.011710	-0.010325	-0.016783
15	-0.014862	0.007766	-0.015247	-0.033073	-0.025844
16	-0.032936	0.044919	-0.031141	-0.050709	-0.040325
17	-0.039378	0.003078	-0.052133	-0.049573	-0.049487
18	-0.025233	0.043158	-0.021085	-0.029500	-0.031410
19	-0.015768	0.018478	-0.006630	-0.026608	-0.028916

[45]:
```
rmse_df
```

[45]:

	model	RMSE
3	decision_tree	0.016000
6	extraTree	0.021814
8	xgb	0.022095
4	random_forest	0.024301
10	cat	0.025238
9	lgbm	0.025735
5	gradient_boost	0.033552
0	ridge	0.040115
7	mlp	0.046557
2	svr	0.061460
1	lasso	0.070945

[49]:
```
rmse_df_group = rmse_df.copy()
rmse_df_group.rename(columns={'RMSE':'RMSE with original␣
↪features'},inplace=True)
```

```
rmse_df_group
```

[49]:

	model	RMSE with original features
3	decision_tree	0.016000
6	extraTree	0.021814
8	xgb	0.022095
4	random_forest	0.024301
10	cat	0.025238
9	lgbm	0.025735
5	gradient_boost	0.033552
0	ridge	0.040115
7	mlp	0.046557
2	svr	0.061460
1	lasso	0.070945

5 4. Cross validation

[50]:
```
from sklearn.model_selection import LeaveOneGroupOut,cross_val_score
from sklearn.metrics import make_scorer

def error_logo(y,y_pred):
    return y.min() - y_pred.min()
error_logo_scorer = make_scorer(error_logo)
logo = LeaveOneGroupOut()

def cross_validation(models,df):
    X_all = df.drop(columns=['Cost','taskid','Supplier ID'])
    y_all = df.Cost
    names = []
    RMSEs = []
    times = []
    results = []
    for key in models.keys():
        model = models[key]
        names.append(key)
        start = time.time()
        result = cross_val_score(model,X_all,y_all,cv = logo.
 ↪split(X_all,y_all,groups=df.taskid),
                                 scoring= error_logo_scorer)
        times.append(time.time()-start)
        results.append(np.expand_dims(result,axis=0))
        RMSE = np.sqrt(np.mean(result**2))
        RMSEs.append(RMSE)
        print('cal:',key,'rmse',RMSE)
    results = np.concatenate(results,axis=0)
```

```
    rmse_df = pd.DataFrame({'model':names,'RMSE':RMSEs,'training time':times}).
↪sort_values('RMSE',ascending=True)
    return results,rmse_df
```

[51]:
```
results,rmse_df = cross_validation(models,df)
rmse_df.to_csv('rmse_original_cv.csv',index=False)
rmse_df
```

```
cal: ridge rmse 0.0425108297597745
cal: lasso rmse 0.06687175997718328
cal: svr rmse 0.060036531130537946
cal: decision_tree rmse 0.031634480510706668
cal: random_forest rmse 0.03320583500255394
cal: gradient_boost rmse 0.03628523079988236
cal: extraTree rmse 0.026459110811550588
cal: mlp rmse 0.08361586323721035
cal: xgb rmse 0.032048983278057136
cal: lgbm rmse 0.03530013738023043
cal: cat rmse 0.031039598371983113
```

[51]:

	model	RMSE	training time
6	extraTree	0.026459	566.940741
10	cat	0.031040	370.939780
3	decision_tree	0.031634	9.697467
8	xgb	0.032049	49.413791
4	random_forest	0.033206	587.068234
9	lgbm	0.035300	126.649835
5	gradient_boost	0.036285	191.182018
0	ridge	0.042511	0.757142
2	svr	0.060037	5.076892
1	lasso	0.066872	0.871144
7	mlp	0.083616	74.052562

[105]:
```
rmse_df.to_csv('rmse_original_cv.csv',index=False)
```

[107]:
```
rmse_df_cv = rmse_df.copy()
rmse_df_cv.rename(columns={'RMSE':'RMSE with original features'},inplace=True)
rmse_df_cv
```

[107]:

	model	RMSE with original features	training time
0	extraTree	0.026459	566.940741
1	cat	0.031040	370.939780
2	decision_tree	0.031634	9.697467
3	xgb	0.032049	49.413791
4	random_forest	0.033206	587.068234
5	lgbm	0.035300	126.649835
6	gradient_boost	0.036285	191.182018

7	ridge	0.042511	0.757142
8	svr	0.060037	5.076892
9	lasso	0.066872	0.871144
10	mlp	0.083616	74.052562

[ ]:
```
from tabulate import tabulate
print(tabulate(rmse_df_cv,headers='keys',tablefmt='psql'))
```

5.0.1 we decide to analysis the feature importances here to generate more useful features to improve the performance, and then we will do the hyperparameter optimization.

[52]:
```
def feature_importaces_trees(models,X_train):
    ''' this method only evaluate the feature importances in decision tree␣
↪methods,
    because there is feature_importances_ attribute in tree methods, for other␣
↪models
    like logistic regression, mlp, permutation importances is used, buts since␣
↪models other than tree
    methods do not perform well, we will not evaluate their feature importance
    '''
    all_importances = {}
    for key in models.keys():
        try:
            importances = models[key].feature_importances_
            importances = importances/ importances.sum() # scale all importance␣
↪values to range(0,1)
            all_importances[key] = importances
        except:
            print(f'model {key} has no feature_importances attribute')
    all_importances['feature_index'] = X_train.columns
    imp_df =pd.DataFrame(all_importances)
    imp_df = imp_df.set_index('feature_index')
    imp_df['mean_importance'] = imp_df.mean(axis=1)
    imp_df = imp_df.sort_values('mean_importance',ascending=False)
    return imp_df
imp_df =  feature_importaces_trees(models,X_train)
imp_df.head()
```

```
model ridge has no feature_importances attribute
model lasso has no feature_importances attribute
model svr has no feature_importances attribute
model mlp has no feature_importances attribute
```

[52]:

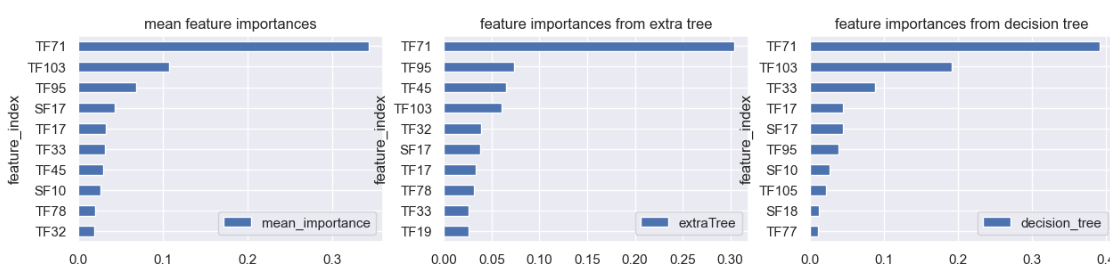
```
               decision_tree  random_forest  gradient_boost  extraTree  \
feature_index
TF71                0.391938       0.393629        0.361580   0.304049
TF103               0.191356       0.177584        0.132599   0.060735
TF95                0.039436       0.047737        0.123009   0.073810
SF17                0.044611       0.044930        0.044611   0.038373
TF17                0.044757       0.039362        0.023520   0.033545

                    xgb      lgbm       cat  mean_importance
feature_index
TF71           0.760661  0.014000  0.176371         0.343176
TF103          0.111302  0.022333  0.058100         0.107716
TF95           0.039464  0.030333  0.120844         0.067805
SF17           0.005014  0.066333  0.056517         0.042913
TF17           0.007597  0.036333  0.046924         0.033148
```

[101]:

```python
## find most importance features through mean importance and extra tree(which␣
↪performs
## best in cross validation), and decsion tree which performs best in training␣
↪on 100 tasks
fig,axes = plt.subplots(1,3,figsize=(15, 3))
tmp_df = imp_df.sort_values('mean_importance',ascending=False)
tmp_df[:10][::-1].plot.barh(y='mean_importance',title='mean feature␣
↪importances',ax=axes[0])
tmp_df = imp_df.sort_values('extraTree',ascending=False)
tmp_df[:10][::-1].plot.barh(y='extraTree',title='feature importances from extra␣
↪tree',ax=axes[1])
tmp_df = imp_df.sort_values('decision_tree',ascending=False)
tmp_df[:10][::-1].plot.barh(y='decision_tree',title='feature importances from␣
↪decision tree',ax=axes[2])
```

[101]:

```
<AxesSubplot: title={'center': 'feature importances from decision tree'},
ylabel='feature_index'>
```



1. from the diagrams of feature importances we can see TF71 is the most important, then followed by TF103 and TF 95

2. we will make more features based on these three features to see the if there is improvement

[54]:
```
# because these features are all anonymous  so we do all kinds of operations to␣
↪find the most useful ones and remove others
# beacause features of each task is fixed for each supplier and features of␣
↪each supplier are fixed for each task, so it is meaningless to do␣
↪aggregation features (as aggregation are all the same)
orig_cols= df.columns
# use the two features with top two feature importances
#df['TF71+TF103'] = df['TF71'] + df['TF103']
#df['TF71-TF103'] = df['TF71'] + df['TF103']
#df['TF71*TF103'] = df['TF71'] + df['TF103']
#df['TF71/TF103'] = df['TF71'] + df['TF103']

# use one task feature and one supplier feature to do the interaction
#df['TF71+SF17'] = df['TF71'] + df['SF17']
#df['TF71-SF17'] = df['TF71'] + df['SF17']
#df['TF71*SF17'] = df['TF71'] + df['SF17']
#df['TF71/SF17'] = df['TF71'] + df['SF17']

imp_cols = ['TF71','TF103','TF95']
more = task[['Task ID','TF71','TF103','TF95']]
for col in imp_cols:
    more['mean_diff_'+col] = more[col] - more[col].mean()
    more['max_diff_'+col] = more[col] - more[col].max()
    more['min_diff_'+col] = more[col] - more[col].min()
more.drop(columns=imp_cols,inplace=True)
more.rename(columns={'Task ID':'taskid'},inplace=True)
df = df.merge(more,on='taskid',how='left')
print(df.shape)
new_cols = [i for i in df.columns if i not in orig_cols]
print('new feature cols:',new_cols) # store the new features made, conveniently␣
↪to delete in the future if needed
```

```
(7560, 56)
new feature cols: ['mean_diff_TF71', 'max_diff_TF71', 'min_diff_TF71',
'mean_diff_TF103', 'max_diff_TF103', 'min_diff_TF103', 'mean_diff_TF95',
'max_diff_TF95', 'min_diff_TF95']
```

[55]:
```
# split the new dataset
X_train_all,X_train,y_train,X_test_all,X_test,y_test = split20(df,TestGroup)
results = train_models(models,X_train,y_train,X_test,y_test)
results
```

```
X_train,y_train,X_test,y_test shapes (6300, 53) (6300,) (1260, 53) (1260,)
fitting model: ridge
fitting model: lasso
fitting model: svr
```

```
fitting model: decision_tree
fitting model: random_forest
fitting model: gradient_boost
fitting model: extraTree
fitting model: mlp
fitting model: xgb
fitting model: lgbm
fitting model: cat
```

[55]:

	models	time for fitting	r2 scores
9	lgbm	1.064000	0.734767
10	cat	2.940001	0.713491
4	random_forest	5.273012	0.712781
5	gradient_boost	1.753035	0.711579
6	extraTree	4.572536	0.700002
8	xgb	0.616999	0.682105
0	ridge	0.004999	0.550632
3	decision_tree	0.101001	0.521099
2	svr	0.038000	0.002192
1	lasso	0.008001	-0.097891
7	mlp	0.572966	-0.370283

[57]:

```
# use the error function in the specification to evaluate again
true_test,rmse_df = evaluate(models,X_test_all,X_test)
rmse_df.to_csv('rmse_new_feature_test.csv')
rmse_df
```

[57]:

	model	RMSE
3	decision_tree	0.016880
8	xgb	0.022095
4	random_forest	0.023561
6	extraTree	0.023676
9	lgbm	0.025902
10	cat	0.026281
5	gradient_boost	0.032727
0	ridge	0.040161
7	mlp	0.046824
2	svr	0.064015
1	lasso	0.070945

[72]:

```
rmse_df_group = rmse_df_group.merge(rmse_df,on='model')
rmse_df_group.rename(columns={'RMSE':'RMSE with new features'},inplace=True)
rmse_df_group
```

[72]:

	model	RMSE with original features	RMSE with new features
0	decision_tree	0.016000	0.016880
1	extraTree	0.021814	0.023676

```
2            xgb                      0.022095                0.022095
3  random_forest                      0.024301                0.023561
4            cat                      0.025238                0.026281
5           lgbm                      0.025735                0.025902
6 gradient_boost                      0.033552                0.032727
7          ridge                      0.040115                0.040161
8            mlp                      0.046557                0.046824
9            svr                      0.061460                0.064015
10         lasso                      0.070945                0.070945
```

[73]:
```
# investigating overfitting
true_test,rmse_df = evaluate(models,X_train_all,X_train)
rmse_df.to_csv('rmse_new_feature_train.csv')
rmse_df
```

[73]:
```
            model      RMSE
3   decision_tree  0.000000
6       extraTree  0.000000
8             xgb  0.011751
4   random_forest  0.012078
10            cat  0.016573
9            lgbm  0.020264
5  gradient_boost  0.028753
0           ridge  0.034776
7             mlp  0.038363
2             svr  0.059944
1           lasso  0.067628
```

[74]:
```
rmse_df_group = rmse_df_group.merge(rmse_df,on='model')
rmse_df_group.rename(columns={'RMSE':'RMSE in training dataset'},inplace=True)
rmse_df_group
```

[74]:
```
            model  RMSE with original features  RMSE with new features  \
0   decision_tree                     0.016000                0.016880
1       extraTree                     0.021814                0.023676
2             xgb                     0.022095                0.022095
3   random_forest                     0.024301                0.023561
4             cat                     0.025238                0.026281
5            lgbm                     0.025735                0.025902
6  gradient_boost                     0.033552                0.032727
7           ridge                     0.040115                0.040161
8             mlp                     0.046557                0.046824
9             svr                     0.061460                0.064015
10          lasso                     0.070945                0.070945

    RMSE in training dataset
0                   0.000000
```

```
1                   0.000000
2                   0.011751
3                   0.012078
4                   0.016573
5                   0.020264
6                   0.028753
7                   0.034776
8                   0.038363
9                   0.059944
10                  0.067628
```

[86]:

```
results,rmse_df = cross_validation(models,df)
rmse_df.to_csv('rmse_new_features_cv.csv',index=False)
rmse_df
```

```
cal: ridge rmse 0.04262488448879486
cal: lasso rmse 0.06687175997718328
cal: svr rmse 0.06004436007029881
cal: decision_tree rmse 0.03031417864897235
cal: random_forest rmse 0.03291755620580411
cal: gradient_boost rmse 0.03618751744712178
cal: extraTree rmse 0.026423709610825658
cal: mlp rmse 0.0643756319895713
cal: xgb rmse 0.032048983278057136
cal: lgbm rmse 0.03601900806722524
cal: cat rmse 0.029265185849863876
```

[86]:

	model	RMSE	training time
6	extraTree	0.026424	578.368642
10	cat	0.029265	369.944771
3	decision_tree	0.030314	10.537847
8	xgb	0.032049	47.731749
4	random_forest	0.032918	666.319653
9	lgbm	0.036019	116.862945
5	gradient_boost	0.036188	225.033427
0	ridge	0.042625	0.993511
2	svr	0.060044	4.296559
7	mlp	0.064376	69.915019
1	lasso	0.066872	1.170350

[111]:

```
rmse_df_cv= rmse_df_cv.merge(rmse_df,on='model')
rmse_df_cv.rename(columns={'RMSE':'RMSE with new features'},inplace=True)
rmse_df_cv.drop(columns=['training time_x','training time_y'],inplace=True)
rmse_df_cv['diff_new_features'] = np.round(rmse_df_cv['RMSE with new features']␣
↪- rmse_df_cv['RMSE with original features'],6)
rmse_df_cv
```

[111]:

	model	RMSE with original features	RMSE with new features
0	extraTree	0.026459	0.026424
1	cat	0.031040	0.029265
2	decision_tree	0.031634	0.030314
3	xgb	0.032049	0.031632
4	random_forest	0.033206	0.032918
5	lgbm	0.035300	0.036566
6	gradient_boost	0.036285	0.036188
7	ridge	0.042511	0.042625
8	svr	0.060037	0.060044
9	lasso	0.066872	0.066872
10	mlp	0.083616	0.064376

	diff_new_features
0	-0.000035
1	-0.001774
2	-0.001320
3	-0.000417
4	-0.000288
5	0.001266
6	-0.000098
7	0.000114
8	0.000008
9	0.000000
10	-0.019240

1. here we can see after using the new features created by blind feature engineering on features with top importances, almost all models show improvements on r2 score.
2. although almost all models perform worse in training 100 groups and test 20 groups, they almost all show improvement in cross validation, meaning it is beneficial for overall performance, only 4 out of 11 show no improvement in cross validation, while mlp, catboost,decision tree and light gradient boost machine shows relatively large improvement. which implies there are much more things we can do in this area to improve the performance, due to the time limit, we stop exploring this methods here, leaving it as possible future work

6 5. Hyper-parameter optimization

Here first I illustrate why the original error function in the coursework pdf cannot be used directly as scorer function in gridsearch.

Because we define the problem as: 1. we use the error function to calculate the mean error among all tasks to evaluate whether a model is good or not, and sign of the error shows if the selected supplier has larger or smaller cost than the cheapest supplier 2. base on the above definition, we can say that we want to minimize the error in order to minimize the RMSE, that is, smaller error means smaller RMSE which are exactly what we want, and so that we can use it as the scoring function in grid search to find the best model with smallest error. 3. but in fact, base on the original function, smaller sum of absolute error does not mean smaller RMSE, and below is an example, this is because of operation of '**2' on the error in RMSE formula.

In order to address this, I modify the original error function by adding **2 and removing** 2 in the RMSE formula, so that they are consistent, and I add a minus to the error function because the gridsearch returns model with higher score, but smaller error is better so I make it negative.

[58]:
```
from sklearn.model_selection import GridSearchCV

X_all = df.drop(columns=['Cost','taskid','Supplier ID'])
y_all = df.Cost
ridge_grid = {'alpha':np.arange(0,30,1)}
ridge_search =␣
↪GridSearchCV(models['ridge'],param_grid=ridge_grid,refit=True,scoring=error_logo_scorer,
                              n_jobs=-1,cv=logo.split(X_all,y_all,groups=df.
↪taskid),verbose=2)
ridge_search = ridge_search.fit(X_all,y_all,groups=df.taskid)
```

```
Fitting 120 folds for each of 30 candidates, totalling 3600 fits
```

[59]:
```
a = np.array([ridge_search.cv_results_[f'split{i}_test_score'][ridge_search.
↪best_index_]for i in range(120)])
num=120
print(a[:num].sum())
print(abs(a[:num]).sum())
print((a[:num]**2).sum())
print('rmse is:',np.sqrt((a[:num]**2).mean())  )
```

```
-2.5845031468437503
3.8226495686562503
0.3321319734447177
rmse is: 0.05260956610135316
```

[60]:
```
b = np.array([ridge_search.cv_results_[f'split{i}_test_score'][ridge_search.
↪best_index_+1]for i in range(120)])
print(b[:num].sum())
print(abs(b[:num]).sum())
print((b[:num]**2).sum())
print('rmse is:',np.sqrt((b[:num]**2).mean())   )
```

```
-3.181482462720905
3.842153080310215
0.21802569332197874
rmse is: 0.042624884488795464
```

[62]:
```
chosen_model_names = ['ridge','lasso','svr','decision_tree']
param_grids ={ 'ridge': {'alpha':np.arange(0,30,1)},
              'lasso': {'alpha':np.linspace(0, 0.002, 30)},
```

```
              'svr':{'kernel':['linear','rbf','poly'],
                    'gamma':['scale','auto',0.0001,0.01,0.1,1,2,4,8],
                    'C':[0.1,0.5,1,2,4,8,16,32]},
             'decision_tree':{'criterion':["squared_error", "friedman_mse",␣
↪"absolute_error" "poisson"],
                              'splitter':['best','random'],
                               'max_depth': np.arange(3,24,3),
                               'min_samples_leaf': [1,2,5, 10, 20]}
             }
def error2_logo(y,y_pred):
    return -(y.min() - y_pred.min())**2   # here I square it in order to select␣
↪models that minize RMSE
error2_logo_scorer = make_scorer(error2_logo)
def grid_search(models,chosen_model_names,df,param_grids):
    searches = {}
    times = []
    names = []
    RMSEs = []
    X_all = df.drop(columns=['Cost','taskid','Supplier ID'])
    y_all = df.Cost
    for key in chosen_model_names:
        try:
            names.append(key)
            start = time.time()
            search =␣
↪GridSearchCV(models[key],param_grid=param_grids[key],refit=True,
                                  scoring=error2_logo_scorer,n_jobs=-1,
                                  cv=logo.split(X_all,y_all,groups=df.
↪taskid),verbose=2)
            search.fit(X_all,y_all,groups=df.taskid)
            times.append(time.time()-start)# log time consumed
            searches[key] = search

            errors = np.array([search.
↪cv_results_[f'split{i}_test_score'][search.best_index_] for i in␣
↪range(120)])# 120 groups(splits)
            RMSE = np.sqrt(-np.mean(errors))
            print(f'optimize model {key} gets RMSE:{RMSE}')
            RMSEs.append(RMSE)
        except:
            print(f'model {key} fail, check the code')
            return searches,RMSEs
    rmse_df = pd.DataFrame({'model':names,'RMSE_optimized':RMSEs,'optimizing␣
↪time':times}).sort_values('RMSE_optimized',ascending=True)
    return searches,rmse_df
```

```
grid_searches,grid_rmse_df =␣
↪grid_search(models,chosen_model_names,df,param_grids)
grid_rmse_df
```

```
Fitting 120 folds for each of 30 candidates, totalling 3600 fits
optimize model ridge gets RMSE:0.03509738014670141
Fitting 120 folds for each of 30 candidates, totalling 3600 fits
optimize model lasso gets RMSE:0.03613887370815572
Fitting 120 folds for each of 216 candidates, totalling 25920 fits
optimize model svr gets RMSE:0.02638153987435806
Fitting 120 folds for each of 210 candidates, totalling 25200 fits
optimize model decision_tree gets RMSE:0.02875933087864516
```

[62]:

	model	RMSE_optimized	optimizing time
2	svr	0.026382	724.009187
3	decision_tree	0.028759	112.601388
0	ridge	0.035097	12.117095
1	lasso	0.036139	68.784434

[66]:
```
grid_searches['ridge'].best_params_
```

[66]:
```
{'alpha': 13}
```

[67]:
```
grid_searches['svr'].best_params_
```

[67]:
```
{'C': 32, 'gamma': 0.01, 'kernel': 'rbf'}
```

[68]:
```
grid_searches['decision_tree'].best_params_
```

[68]:
```
{'criterion': 'friedman_mse',
 'max_depth': 18,
 'min_samples_leaf': 2,
 'splitter': 'random'}
```

[69]:
```
grid_searches['lasso'].best_params_
```

[69]:
```
{'alpha': 6.896551724137931e-05}
```

6.1 Using bayesian optimization for faster running and better performance than gridsearch

gridsearch use predefined parameters and search for every combination, which cost tons of time and easily get a sub-optimal solution due to high dependence on human expertise. So instead we use bayesian methods to explore the parameters space more efficiently

[45]:
```python
import optuna
from sklearn.model_selection import GroupKFold
from optuna.integration import LightGBMPruningCallback
```

[46]:
```python
def print_study(study,name):
    print(f"\t {name} Best value (rmse): {study.best_value:.5f}")
    print(f"\tBest params:")

    for key, value in study.best_params.items():
        print(f"\t\t{key}: {value}")
```

[ ]:
```python
def objective(trial, X_train, y_train,X_test,y_test):
    this_grid={"n_estimators": trial.suggest_categorical("n_estimators",␣
↪[300,200,100,]),
                                'loss':trial.
↪suggest_categorical('loss',['squared_error', 'absolute_error', 'huber',␣
↪'quantile']),
                                "learning_rate":trial.
↪suggest_categorical("learning_rate", [0.1, 0.05,0.2]),
                                'criterion':trial.
↪suggest_categorical('criterion', ['friedman_mse','squared_error']),
                                'min_samples_leaf': trial.
↪suggest_int('min_samples_leaf', 1,30),
                                   'min_samples_split':trial.
↪suggest_int('min_samples_split', 2,20),
                                   'max_depth':  trial.suggest_int('max_depth',␣
↪3,30),
                               'max_features':trial.
↪suggest_categorical('max_features',['auto','sqrt','log2']),
                               }
    model = GradientBoostingRegressor(random_state=seed,**this_grid)
    model.fit(X_train,y_train)

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error
name = 'gradient_boost'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=100)
print_study(study,name)
```

[ ]:
```python
def objective(trial, X_train, y_train,X_test,y_test):
    this_grid={"n_estimators": trial.suggest_categorical("n_estimators",␣
↪[300,200,100]),
```

```
                                'criterion':trial.
↪suggest_categorical('criterion',["squared_error", "friedman_mse",␣
↪"absolute_error", "poisson"]),
                                'max_depth':  trial.suggest_int('max_depth',␣
↪3,30),
                                'min_samples_split':trial.
↪suggest_int('min_samples_split', 2,20),
                                'min_samples_leaf': trial.
↪suggest_int('min_samples_leaf', 1,30),
                               'max_features':trial.
↪suggest_categorical('max_features',['auto','sqrt','log2']),
                               }
    model = RandomForestRegressor(random_state=seed,**this_grid)
    model.fit(X_train,y_train)

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error
name = 'random_forest'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=70)
print_study(study,name)
```

[ ]:
```
def objective(trial, X_train, y_train,X_test,y_test):
    this_grid={'criterion':trial.
↪suggest_categorical('criterion',["squared_error", "friedman_mse",␣
↪"absolute_error", "poisson"]),
                             'max_depth':  trial.suggest_int('max_depth', 3,30),
                             'min_samples_leaf': trial.
↪suggest_int('min_samples_leaf', 1,30),
                            'min_samples_split':trial.
↪suggest_int('min_samples_split', 2,20),
                            'max_features':trial.
↪suggest_categorical('max_features',['auto','sqrt','log2']),
    }
    model = ExtraTreesRegressor(random_state=seed,**this_grid)
    model.fit(X_train,y_train)

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error
name = 'extraTree'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=100)
```

```
print_study(study,name)
```

[ ]:
```
def objective(trial, X_train, y_train,X_test,y_test):
    this_grid={'hidden_layer_sizes': trial.
↪suggest_categorical("hidden_layer_sizes",
                                                                                          ␣
↪[(100,),(100,50),(50,100,50),(100,100)]),
                          'solver':trial.
↪suggest_categorical('solver',['adam','sgd']),
                          'alpha':trial.suggest_float('alpha', 0.0001,1),
                          'learning_rate':trial.
↪suggest_categorical('learning_rate',['constant','adaptive']),
                          'max_iter' : trial.suggest_categorical('max_iter',␣
↪[100,200,300,400]),
                          'activation' : trial.suggest_categorical('activation',␣
↪['tanh','relu']),
                          'early_stopping':True,
    }
    model = MLPRegressor(random_state=seed,**this_grid)
    model.fit(X_train,y_train)

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error
name = 'mlp'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=100)
print_study(study,name)
```

[ ]:
```
def objective(trial, X_train, y_train,X_test,y_test):
    param={
         'l2_leaf_reg': trial.suggest_int('l2_leaf_reg',1, 30,step=1),
        "max_depth": trial.suggest_int("max_depth", 1,16),
        "bootstrap_type": trial.suggest_categorical("bootstrap_type",␣
↪["Bayesian", "Bernoulli"]),
        "learning_rate":trial.suggest_categorical('learning_rate',[0.1,0.05,0.
↪2]),
            #'task_type':"GPU",
            'iterations':trial.
↪suggest_categorical('iterations',[1000,700,1500]),
    }
    if param["bootstrap_type"] == "Bayesian":
        param["bagging_temperature"] = trial.
↪suggest_float("bagging_temperature", 0, 10)
```

```
        param['bagging_temperature'] = trial.
↪suggest_float('bagging_temperature',0.0, 1.0)
    elif param["bootstrap_type"] == "Bernoulli":
        param["subsample"] = trial.suggest_float("subsample", 0.1, 1)
    model = CatBoostRegressor(random_state=seed,**param)
    model.fit(
            X_train,
            y_train,
            eval_set=[(X_test, y_test)],
            early_stopping_rounds=50,
            verbose=False
        )

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error
name = 'cat'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=100)
print_study(study,name)
```

```
[ ]: def objective(trial, X_train, y_train,X_test,y_test):
    this_grid={'max_depth': trial.suggest_int("max_depth", 3, 20, step=2),
        'gamma': trial.suggest_float('gamma', 0,9),
        'reg_alpha' : trial.suggest_int('reg_alpha', 0,40,step=1),
        'reg_lambda' : trial.suggest_float('reg_lambda', 0,1),
        'colsample_bytree' : trial.suggest_float('colsample_bytree', 0.5,1),
        'min_child_weight' : trial.suggest_int('min_child_weight', 0, 10,␣
↪step=1),
        'n_estimators': trial.suggest_categorical("n_estimators",␣
↪[300,200,100]),
        "learning_rate":trial.suggest_categorical("learning_rate", [0.1, 0.05,0.
↪2]),
        }
    model = XGBRegressor(tree_method='gpu_hist',random_state=seed,␣
↪gpu_id=0,**this_grid)
    model.fit(
            X_train,
            y_train,
            eval_set=[(X_test, y_test)],
            early_stopping_rounds=50,
            verbose=False
        )

    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
```

```
    return error
name = 'xgb'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test)
study.optimize(func, n_trials=100)
print_study(study,name)
```

[52]:
```
print_study(study,name)
```

```
        xgb Best value (rmse): 0.00062
        Best params:
                max_depth: 17
                gamma: 0.0021819783924399427
                reg_alpha: 0
                reg_lambda: 0.19555028051297343
                colsample_bytree: 0.9184434429497392
                min_child_weight: 3
                n_estimators: 200
                learning_rate: 0.1
```

[ ]:
```
# decision_tree extra_tree xgb random_for cat lgbm gradient_boost,mlp
def objective(trial, X_train, y_train,X_test,y_test,name):
    this_grid= {
        "n_estimators": trial.suggest_categorical("n_estimators",␣
 ↪[300,200,100]),
        "learning_rate":trial.suggest_categorical("learning_rate", [0.1, 0.05,0.
 ↪2]),
        "num_leaves": trial.suggest_int("num_leaves", 32, 160, step=32),
        "max_depth": trial.suggest_int("max_depth", 3, 15,step=2),
        "lambda_l1": trial.suggest_int("lambda_l1", 0, 10, step=1),
        "lambda_l2": trial.suggest_int("lambda_l2", 0, 10, step=1),
        "min_split_gain": trial.suggest_float("min_gain_to_split", 0, 15),
        "min_data_in_leaf":trial.suggest_int("min_data_in_leaf", 20, 100,␣
 ↪step=20),
        "bagging_fraction": trial.suggest_float(
            "bagging_fraction", 0.2, 1, step=0.1
        ),
        "feature_fraction": trial.suggest_float(
            "feature_fraction", 0.2, 1, step=0.1
        ),}


    model = LGBMRegressor(device_type='gpu',random_state=seed,**this_grid)
    model.fit(
        X_train,
        y_train,
        eval_set=[(X_test, y_test)],
```

```
        early_stopping_rounds=50,
        verbose=False)
    preds = model.predict(X_test)
    error = np.mean((y_test-preds)**2)
    return error

name = 'lgbm'
study = optuna.create_study(direction="minimize", study_name=name)
func = lambda trial: objective(trial,X_train, y_train,X_test,y_test,name)
study.optimize(func, n_trials=100)
print_study(study,name)
```

6.2 final evaluation of optimized model

[125]:
```
optimized_models = {'ridge':Ridge(random_state=seed,alpha=13),
         'lasso':Lasso(random_state=seed,alpha=6.896551724137931e-05),
         'svr':SVR(C=32,gamma=0.01,kernel='rbf'),
          'decision_tree':
 ↪DecisionTreeRegressor(random_state=seed,criterion='friedman_mse',splitter='random',min_samp
         'random_forest':
 ↪RandomForestRegressor(random_state=seed,n_estimators=100,criterion='friedman_mse',min_sampl
         'gradient_boost':
 ↪GradientBoostingRegressor(random_state=seed,n_estimators=100,loss='squared_error',learning_
 ↪05,min_samples_leaf=13,min_samples_split=4,max_depth=18,max_features='log2'),
         'extraTree':
 ↪ExtraTreesRegressor(random_state=seed,criterion='squared_error',min_samples_leaf=4,min_samp
         'mlp':
 ↪MLPRegressor(random_state=seed,hidden_layer_sizes=(100,100),solver='adam',alpha=0.
 ↪02626,learning_rate='adaptive',max_iter=300,activation='tanh'),
         'xgb':XGBRegressor(tree_method='gpu_hist',random_state=seed,␣
 ↪gpu_id=0,max_depth=17,gamma=0.00218,reg_alpha=0,reg_lambda=0.
 ↪19555,colsample_bytree=0.
 ↪91844,min_child_weight=3,n_estimators=200,learning_rate=0.1),
         'lgbm':LGBMRegressor(random_state=seed,n_estimator=300,learning_rate=0.
 ↪1,num_leaves=128,max_depth=7,lambda_l1=1,lambda_l2=7,min_gain_to_split=0.
 ↪01344,
                              min_data_in_leaf=60,bagging_fraction=0.
 ↪4,feature_fraction=1,device_type='gpu'),
         'cat':CatBoostRegressor(random_state=seed, l2_leaf_reg=7,max_depth=8,␣
 ↪bootstrap_type='Bernoulli',learning_rate=0.2,iterations=1000,subsample=0.
 ↪17406,verbose=False)}
```

[ ]:
```
results_final,rmse_df = cross_validation(optimized_models,df)
```

[100]:
```
rmse_df.to_csv('rmse_df_cv_optimized_new_feature.csv',index=False)
rmse_df
```

[100]:

	model	RMSE	training time
0	svr	0.026382	5.499053
1	cat	0.028245	547.435938
2	decision_tree_fromGrid	0.028759	4.995856
3	extraTree	0.029615	353.246168
4	gradient_boost	0.029936	85.056119
5	mlp	0.033124	457.562721
6	ridge	0.035097	0.945391
11	lasso	0.036139	1.089260
7	random_forest	0.037360	419.190563
8	xgb	0.040756	53.269071
9	decision_tree_fromBayesian	0.042112	7.663635
10	lgbm	0.043817	38.611378

[118]:
```
rmse_df = pd.read_csv('rmse_df_cv_optimized_new_feature.csv')
rmse_df
```

[118]:

	model	RMSE	training time
0	svr	0.026382	5.499053
1	cat	0.028245	547.435938
2	decision_tree	0.028759	4.995856
3	extraTree	0.029615	353.246168
4	gradient_boost	0.029936	85.056119
5	mlp	0.033124	457.562721
6	ridge	0.035097	0.945391
7	lasso	0.036139	1.089260
8	random_forest	0.037360	419.190563
9	xgb	0.040756	53.269071
10	lgbm	0.043817	38.611378

[120]:
```
rmse_df_cv= rmse_df_cv.merge(rmse_df.drop(columns='training time'),on='model')
rmse_df_cv.rename(columns={'RMSE':'RMSE with new features after␣
↪optimized'},inplace=True)
rmse_df_cv['diff_new_features_optimized'] = np.round(rmse_df_cv['RMSE with new␣
↪features after optimized'] - rmse_df_cv['RMSE with new features'],6)
rmse_df_cv
```

[120]:

	model	RMSE with original features	RMSE with new features	\
0	extraTree	0.026459	0.026424	
1	cat	0.031040	0.029265	
2	decision_tree	0.031634	0.030314	
3	xgb	0.032049	0.031632	
4	random_forest	0.033206	0.032918	
5	lgbm	0.035300	0.036566	

```
6   gradient_boost                    0.036285                0.036188
7            ridge                    0.042511                0.042625
8              svr                    0.060037                0.060044
9            lasso                    0.066872                0.066872
10             mlp                    0.083616                0.064376

    diff_new_features  RMSE with new features after optimized  \
0           -0.000035                                0.029615
1           -0.001774                                0.028245
2           -0.001320                                0.028759
3           -0.000417                                0.040756
4           -0.000288                                0.037360
5            0.001266                                0.043817
6           -0.000098                                0.029936
7            0.000114                                0.035097
8            0.000008                                0.026382
9            0.000000                                0.036139
10          -0.019240                                0.033124

    diff_new_features_optimized
0                      0.003191
1                     -0.001020
2                     -0.001555
3                      0.009123
4                      0.004443
5                      0.007250
6                     -0.006252
7                     -0.007528
8                     -0.033663
9                     -0.030733
10                    -0.031252
```

[121]:
```python
rmse_df_cv.sort_values('RMSE with new features after optimized',ascending=True)
```

[121]:
```
             model  RMSE with original features  RMSE with new features  \
8              svr                    0.060037                0.060044
1              cat                    0.031040                0.029265
2    decision_tree                    0.031634                0.030314
0        extraTree                    0.026459                0.026424
6   gradient_boost                    0.036285                0.036188
10             mlp                    0.083616                0.064376
7            ridge                    0.042511                0.042625
9            lasso                    0.066872                0.066872
4    random_forest                    0.033206                0.032918
3              xgb                    0.032049                0.031632
5             lgbm                    0.035300                0.036566
```

```
    diff_new_features  RMSE with new features after optimized  \
8            0.000008                                 0.026382
1           -0.001774                                 0.028245
2           -0.001320                                 0.028759
0           -0.000035                                 0.029615
6           -0.000098                                 0.029936
10          -0.019240                                 0.033124
7            0.000114                                 0.035097
9            0.000000                                 0.036139
4           -0.000288                                 0.037360
3           -0.000417                                 0.040756
5            0.001266                                 0.043817

    diff_new_features_optimized
8                     -0.033663
1                     -0.001020
2                     -0.001555
0                      0.003191
6                     -0.006252
10                    -0.031252
7                     -0.007528
9                     -0.030733
4                      0.004443
3                      0.009123
5                      0.007250
```

6.3 compare the results to 4

1. we can see almost all models improve a lot by the hyperparamter optimzation, and especially svr which performs one the worst before and also worst in r2, performs best in RMSE, showing it finding the relation between suppliers rather than precisely predicted the 'Cost' value.

[126]:

```
results = train_models(optimized_models,X_train,y_train,X_test,y_test)
results.to_csv('results_train100Group_optimized.csv')
results
```

```
fitting model: ridge
fitting model: lasso
fitting model: svr
fitting model: decision_tree
fitting model: random_forest
fitting model: gradient_boost
fitting model: extraTree
fitting model: mlp
fitting model: xgb
fitting model: lgbm
[LightGBM] [Warning] Unknown parameter: n_estimator
```

```
[LightGBM] [Warning] feature_fraction is set=1, colsample_bytree=1.0 will be
ignored. Current value: feature_fraction=1
[LightGBM] [Warning] min_data_in_leaf is set=60, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=60
[LightGBM] [Warning] min_gain_to_split is set=0.01344, min_split_gain=0.0 will
be ignored. Current value: min_gain_to_split=0.01344
[LightGBM] [Warning] bagging_fraction is set=0.4, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.4
[LightGBM] [Warning] lambda_l1 is set=1, reg_alpha=0.0 will be ignored. Current
value: lambda_l1=1
[LightGBM] [Warning] lambda_l2 is set=7, reg_lambda=0.0 will be ignored. Current
value: lambda_l2=7
fitting model: cat
```

[126]:

	models	time for fitting	r2 scores
10	cat	3.318579	0.738696
5	gradient_boost	0.663998	0.728293
4	random_forest	3.049063	0.719446
8	xgb	0.736572	0.717305
6	extraTree	2.500489	0.716465
9	lgbm	0.240001	0.687355
0	ridge	0.006004	0.570090
1	lasso	0.098997	0.558160
7	mlp	2.766034	0.518054
3	decision_tree	0.039000	0.393789
2	svr	0.032999	0.177325

[128]:

```
true_test,rmse_df = evaluate(optimized_models,X_test_all,X_test)
rmse_df
```

[128]:

	model	RMSE
10	cat	0.019637
3	decision_tree	0.023209
5	gradient_boost	0.024762
6	extraTree	0.027424
4	random_forest	0.031729
7	mlp	0.032136
8	xgb	0.032641
0	ridge	0.038681
9	lgbm	0.039678
1	lasso	0.039909
2	svr	0.053042

[129]:

```
rmse_df_group.merge(rmse_df,on='model')
```

[129]:

	model	RMSE with original features	RMSE with new features	\
0	decision_tree	0.016000	0.016880	

```
1        extraTree                     0.021814               0.023676
2              xgb                     0.022095               0.022095
3    random_forest                     0.024301               0.023561
4              cat                     0.025238               0.026281
5             lgbm                     0.025735               0.025902
6   gradient_boost                     0.033552               0.032727
7            ridge                     0.040115               0.040161
8              mlp                     0.046557               0.046824
9              svr                     0.061460               0.064015
10           lasso                     0.070945               0.070945

    RMSE in training dataset      RMSE
0                   0.000000  0.023209
1                   0.000000  0.027424
2                   0.011751  0.032641
3                   0.012078  0.031729
4                   0.016573  0.019637
5                   0.020264  0.039678
6                   0.028753  0.024762
7                   0.034776  0.038681
8                   0.038363  0.032136
9                   0.059944  0.053042
10                  0.067628  0.039909
```

[130]:

```python
rmse_df_group = rmse_df_group.merge(rmse_df,on='model')
rmse_df_group.rename(columns={'RMSE':'RMSE with new features␣
 ↪optimized'},inplace=True)
rmse_df_group['diff_new_feature_with_optimized'] = rmse_df_group['RMSE with new␣
 ↪features optimized'] - rmse_df_group['RMSE with new features']
rmse_df_group['diff_new_feature_with_original_feature'] = rmse_df_group['RMSE␣
 ↪with new features'] - rmse_df_group['RMSE with original features']
rmse_df_group
```

[130]:

```
             model  RMSE with original features  RMSE with new features  \
0    decision_tree                     0.016000               0.016880
1        extraTree                     0.021814               0.023676
2              xgb                     0.022095               0.022095
3    random_forest                     0.024301               0.023561
4              cat                     0.025238               0.026281
5             lgbm                     0.025735               0.025902
6   gradient_boost                     0.033552               0.032727
7            ridge                     0.040115               0.040161
8              mlp                     0.046557               0.046824
9              svr                     0.061460               0.064015
10           lasso                     0.070945               0.070945

    RMSE in training dataset  RMSE with new features optimized  \
```

0	0.000000	0.023209
1	0.000000	0.027424
2	0.011751	0.032641
3	0.012078	0.031729
4	0.016573	0.019637
5	0.020264	0.039678
6	0.028753	0.024762
7	0.034776	0.038681
8	0.038363	0.032136
9	0.059944	0.053042
10	0.067628	0.039909

	diff_new_feature_with_optimized	diff_new_feature_with_original_feature
0	0.006329	0.000880
1	0.003748	0.001862
2	0.010546	0.000000
3	0.008168	-0.000740
4	-0.006644	0.001043
5	0.013776	0.000167
6	-0.007965	-0.000825
7	-0.001480	0.000046
8	-0.014688	0.000267
9	-0.010973	0.002555
10	-0.031036	0.000000

[110]:
```
# feature importances
imp_df = feature_importaces_trees(optimized_models,X_train)
imp_df.head()
```

```
model ridge has no feature_importances attribute
model lasso has no feature_importances attribute
model svr has no feature_importances attribute
model mlp has no feature_importances attribute
```

[110]:

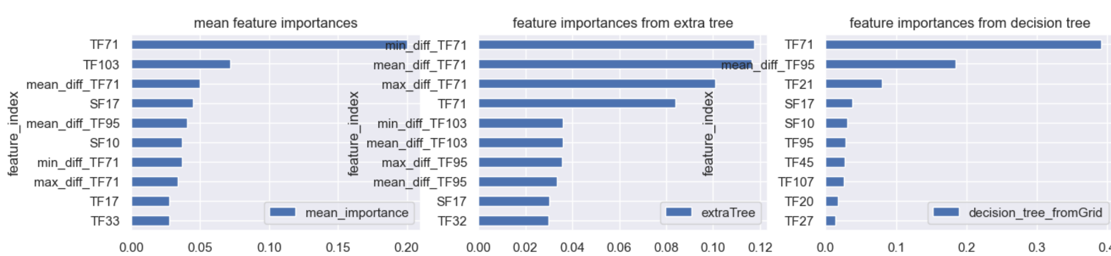
feature_index	decision_tree_fromBayesian	decision_tree_fromGrid \
TF71	0.440962	0.391458
TF103	0.177557	0.004086
mean_diff_TF71	0.000000	0.000433
SF17	0.041297	0.038240
mean_diff_TF95	0.045939	0.184909

feature_index	random_forest	gradient_boost	extraTree	xgb	lgbm \
TF71	0.123543	0.007329	0.084127	0.512655	0.017903
TF103	0.048185	0.065544	0.026581	0.211792	0.023018
mean_diff_TF71	0.105876	0.056841	0.116305	0.006082	0.012788

```
SF17                 0.034885         0.030664    0.030254  0.007264  0.104859
mean_diff_TF95       0.010330         0.014915    0.033384  0.000000  0.015345

                          cat  mean_importance
feature_index
TF71              0.021392         0.199921
TF103             0.020339         0.072138
mean_diff_TF71    0.098472         0.049599
SF17              0.070038         0.044688
mean_diff_TF95    0.017625         0.040306
```

[112]:
```
## find most importance features through mean importance and extra tree(which␣
↪performs
## best in cross validation), and decsion tree which performs best in training␣
↪on 100 tasks
fig,axes = plt.subplots(1,3,figsize=(15, 3))
tmp_df = imp_df.sort_values('mean_importance',ascending=False)
tmp_df[:10][::-1].plot.barh(y='mean_importance',title='mean feature␣
↪importances',ax=axes[0])
tmp_df = imp_df.sort_values('extraTree',ascending=False)
tmp_df[:10][::-1].plot.barh(y='extraTree',title='feature importances from extra␣
↪tree',ax=axes[1])
tmp_df = imp_df.sort_values('decision_tree_fromGrid',ascending=False)
tmp_df[:10][::-1].plot.barh(y='decision_tree_fromGrid',title='feature␣
↪importances from decision tree',ax=axes[2])
```

[112]:
```
&lt;AxesSubplot: title={'center': 'feature importances from decision tree'},
ylabel='feature_index'&gt;
```



1. from the diagram of feature importance of optimized models with new features we can see the generated features have positive impact on the results, making the performance better, which are proved effective by the feature importances ranking.

6.4 bias analysis

[132]:
```
optimized_models
```

[132]:
```
{'ridge': Ridge(alpha=13, random_state=42),
 'lasso': Lasso(alpha=6.896551724137931e-05, random_state=42),
 'svr': SVR(C=32, gamma=0.01),
 'decision_tree': DecisionTreeRegressor(criterion='friedman_mse', max_depth=18,
                       min_samples_leaf=2, random_state=42, splitter='random'),
 'random_forest': RandomForestRegressor(criterion='friedman_mse', max_depth=16,
                       max_features='auto', min_samples_leaf=8,
                       min_samples_split=16, random_state=42),
 'gradient_boost': GradientBoostingRegressor(learning_rate=0.05, max_depth=18,
max_features='log2',
                           min_samples_leaf=13, min_samples_split=4,
                           random_state=42),
 'extraTree': ExtraTreesRegressor(max_depth=20, max_features='auto',
min_samples_leaf=4,
                     min_samples_split=10, random_state=42),
 'mlp': MLPRegressor(activation='tanh', alpha=0.02626, hidden_layer_sizes=(100,
100),
              learning_rate='adaptive', max_iter=300, random_state=42),
 'xgb': XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.91844,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0.00218, gpu_id=0,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_bin=256,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
              max_depth=17, max_leaves=0, min_child_weight=3, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=0,
              num_parallel_tree=1, predictor='auto', random_state=42, …),
 'lgbm': LGBMRegressor(bagging_fraction=0.4, device_type='gpu',
feature_fraction=1,
               lambda_l1=1, lambda_l2=7, max_depth=7, min_data_in_leaf=60,
               min_gain_to_split=0.01344, n_estimator=300, num_leaves=128,
               random_state=42),
 'cat': <catboost.core.CatBoostRegressor at 0x1de7b188eb0>}
```

[133]:
```
rmse_df_cv
```

[133]:
```
             model  RMSE with original features  RMSE with new features  \
0        extraTree                     0.026459                0.026424
1              cat                     0.031040                0.029265
2    decision_tree                     0.031634                0.030314
3              xgb                     0.032049                0.031632
4    random_forest                     0.033206                0.032918
```

```
5            lgbm                     0.035300                0.036566
6   gradient_boost                    0.036285                0.036188
7            ridge                    0.042511                0.042625
8              svr                    0.060037                0.060044
9            lasso                    0.066872                0.066872
10             mlp                    0.083616                0.064376

    diff_new_features  RMSE with new features after optimized  \
0           -0.000035                                0.029615
1           -0.001774                                0.028245
2           -0.001320                                0.028759
3           -0.000417                                0.040756
4           -0.000288                                0.037360
5            0.001266                                0.043817
6           -0.000098                                0.029936
7            0.000114                                0.035097
8            0.000008                                0.026382
9            0.000000                                0.036139
10          -0.019240                                0.033124

    diff_new_features_optimized
0                      0.003191
1                     -0.001020
2                     -0.001555
3                      0.009123
4                      0.004443
5                      0.007250
6                     -0.006252
7                     -0.007528
8                     -0.033663
9                     -0.030733
10                    -0.031252
```

[ ]:

```
results_final,rmse_df = cross_validation(optimized_models,df)
rmse_df
```

[135]:

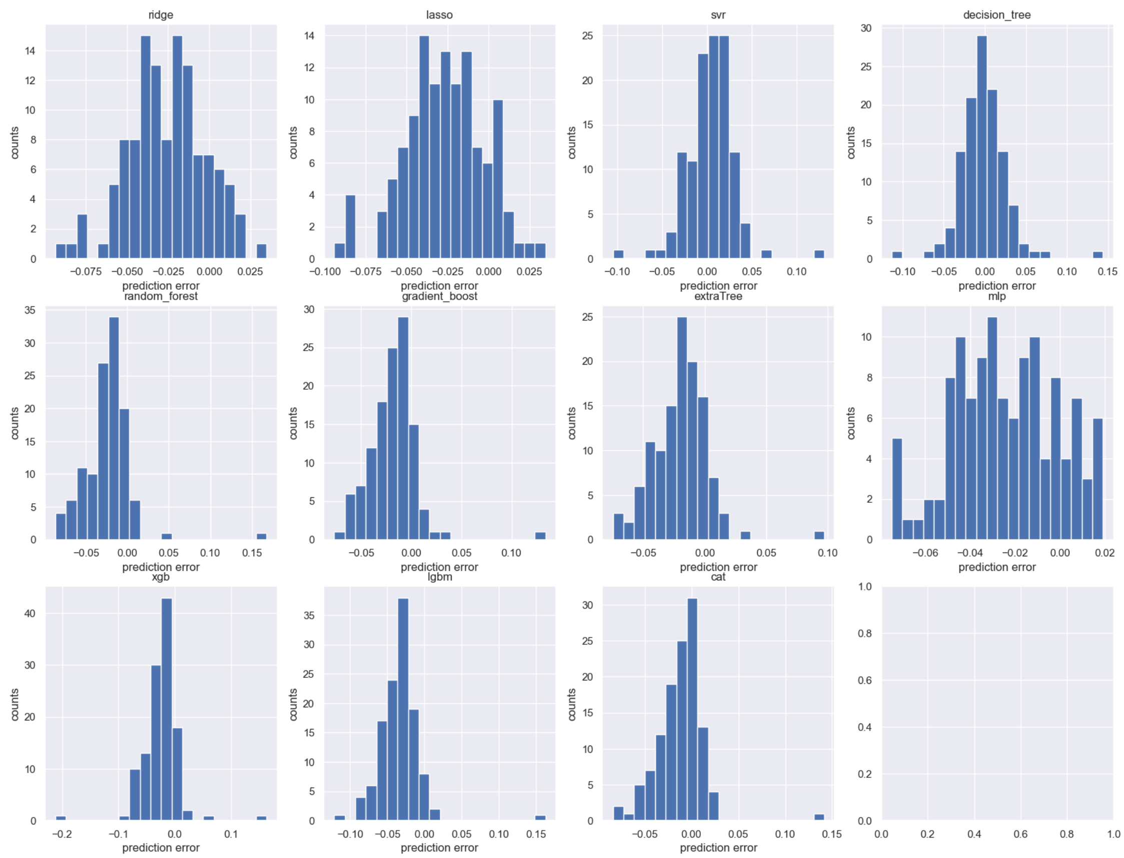
```
rmse_df
```

[135]:

	model	RMSE	training time
2	svr	0.026382	5.739604
10	cat	0.028245	613.595765
3	decision_tree	0.028759	5.410276
6	extraTree	0.029615	373.021894
5	gradient_boost	0.029936	84.956835
7	mlp	0.033124	482.263296
0	ridge	0.035097	1.005001
1	lasso	0.036139	18.496184

```
4    random_forest  0.037360     429.359955
8              xgb  0.041365      66.485939
9             lgbm  0.043580      42.462292
```

[192]:
```python
s=5
rows = 3
hor_num = 4
fig, axs = plt.subplots(rows, hor_num,figsize=(s*hor_num,s*rows))
for i in range(rows):
    for j in range(hor_num):
        if i*hor_num+j ==len(results_final):
            break
        axs[i,j].set_xlabel('prediction error')
        axs[i,j].set_ylabel('counts')
        axs[i,j].set_title(list(optimized_models.keys())[i*hor_num+j])
        axs[i,j].hist(results_final[i*hor_num+j],bins=20)
plt.savefig('prediction_distribution.png')
plt.show()
```



[160]:
```
print(optimized_models.keys())
bias = {}
new_rmse = {}
for index,key in enumerate(optimized_models.keys()):
    r = results_final[index]
    best = np.sqrt(np.mean(r**2))
    print('original score',best)
    for i in range(100):
        new_r = r+  (i/1000-50/1000)
        score = np.sqrt(np.mean(new_r**2))
        if score<best:
            best = score
            bias[key] =round( i/1000-50/1000,6)
            new_rmse[key] = best
    print(f'score of{key} after bias adjusted: {best}')
```

```
dict_keys(['ridge', 'lasso', 'svr', 'decision_tree', 'random_forest',
'gradient_boost', 'extraTree', 'mlp', 'xgb', 'lgbm', 'cat'])
original score 0.03509738014670146
score ofridge after bias adjusted: 0.023878018915509357
original score 0.03613887370815572
score oflasso after bias adjusted: 0.024360827621387065
original score 0.02638153987435806
score ofsvr after bias adjusted: 0.026137885636598433
original score 0.02875933087864516
score ofdecision_tree after bias adjusted: 0.028696656994009553
original score 0.03736028231749364
score ofrandom_forest after bias adjusted: 0.028444117116899793
original score 0.029935942903775138
score ofgradient_boost after bias adjusted: 0.02374835911331444
original score 0.02961487752655635
score ofextraTree after bias adjusted: 0.022202955751535053
original score 0.03312395413303757
score ofmlp after bias adjusted: 0.022881290169875435
original score 0.04136523985229596
score ofxgb after bias adjusted: 0.033339479238603496
original score 0.043580100246555266
score oflgbm after bias adjusted: 0.02804918051345538
original score 0.028244892906587684
score ofcat after bias adjusted: 0.02514706497319126
```

[161]:
```
bias
```

[161]:
```
{'ridge': 0.026,
 'lasso': 0.027,
 'svr': -0.004,
 'decision_tree': 0.002,
```

```
 'random_forest': 0.024,
 'gradient_boost': 0.018,
 'extraTree': 0.02,
 'mlp': 0.024,
 'xgb': 0.024,
 'lgbm': 0.033,
 'cat': 0.013}
```

[147]:
```
rmse_df = pd.DataFrame({'model':new_rmse.keys(),'final RMSE after bias␣
 ↪adjusting': new_rmse.values()})
```

[163]:
```
rmse_df_cv = rmse_df_cv.merge(rmse_df,on='model')
rmse_df_cv = rmse_df_cv.sort_values('final RMSE after bias␣
 ↪adjusting',ascending=True)
rmse_df_cv['diff_bias_to_new_features_optimized'] = rmse_df_cv['final RMSE␣
 ↪after bias adjusting'] - rmse_df_cv['RMSE with new features after optimized']
rmse_df_cv['diff_bias_to_original'] = rmse_df_cv['final RMSE after bias␣
 ↪adjusting'] - rmse_df_cv['RMSE with original features']
rmse_df_cv
```

[163]:
```
             model  RMSE with original features  RMSE with new features  \
0        extraTree                     0.026459                0.026424
10             mlp                     0.083616                0.064376
6   gradient_boost                     0.036285                0.036188
7            ridge                     0.042511                0.042625
9            lasso                     0.066872                0.066872
1              cat                     0.031040                0.029265
8              svr                     0.060037                0.060044
5             lgbm                     0.035300                0.036566
4    random_forest                     0.033206                0.032918
2    decision_tree                     0.031634                0.030314
3              xgb                     0.032049                0.031632

    diff_new_features  RMSE with new features after optimized  \
0           -0.000035                                 0.029615
10          -0.019240                                 0.033124
6           -0.000098                                 0.029936
7            0.000114                                 0.035097
9            0.000000                                 0.036139
1           -0.001774                                 0.028245
8            0.000008                                 0.026382
5            0.001266                                 0.043817
4           -0.000288                                 0.037360
2           -0.001320                                 0.028759
3           -0.000417                                 0.040756

    diff_new_features_optimized  final RMSE after bias adjusting  \
```

0	0.003191	0.022203
10	-0.031252	0.022881
6	-0.006252	0.023748
7	-0.007528	0.023878
9	-0.030733	0.024361
1	-0.001020	0.025147
8	-0.033663	0.026138
5	0.007250	0.028049
4	0.004443	0.028444
2	-0.001555	0.028697
3	0.009123	0.033339

	diff_bias_to_new_features_optimized	diff_bias_to_original
0	-0.007412	-0.004256
10	-0.010243	-0.060735
6	-0.006188	-0.012537
7	-0.011219	-0.018633
9	-0.011778	-0.042511
1	-0.003098	-0.005893
8	-0.000244	-0.033899
5	-0.015767	-0.007251
4	-0.008916	-0.004762
2	-0.000063	-0.002938
3	-0.007416	0.001290

[167]:
```
results_final.shape
```

[167]: (11, 120)

[ ]:
```
results_final_copy = results_final.copy()
for index,key in enumerate(bias.keys()):
    b=bias[key]
    results_final_copy[index] = results_final_copy[index]+b
#dict_keys(['ridge', 'lasso', 'svr', 'decision_tree', 'random_forest',␣
↪'gradient_boost', 'extraTree', 'mlp', 'xgb', 'lgbm', 'cat'])
bias.keys()
```

6.5 ensemble results of extra tree, gradient boost, mlp, ridge and lasso

[182]:
```
ensemble = (results_final_copy[6] + results_final_copy[7] +␣
↪results_final_copy[5]  +results_final_copy[0]+results_final_copy[1])/5 #␣
↪extraTree + mlp + gradient_boost
np.sqrt(np.mean(ensemble**2))
```

[182]: 0.020039957730295162

```
[ ]:
```