问题3

In [1]:

```
import mitosheet
import numpy as np
import pandas as pd
import plotly as py
import cufflinks as cf
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff

cf.set_config_file(
    offline=True,
    world_readable=True,
    theme='white',        # 设置绘图风格
)

import warnings
warnings.filterwarnings("ignore")

import sklearn
import graphviz
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score

from colorama import Fore
def color(text):
    return Fore.RED + text + Fore.RESET
```

In [2]:

```
file_path = './附件2(Attachment 2)2022-51MCM-Problem B.xlsx'
sheet1 = pd.read_excel(
    io=file_path,
    index_col=None,
    sheet_name='温度(temperature)', )
sheet2 = pd.read_excel(
    io=file_path,
    index_col=None,
    sheet_name='产品质量(quality of the products)', )
```

```python
sheet3 = pd.read_excel(
    io=file_path,
    index_col=None,
    sheet_name='原矿参数(mineral parameter)', )
sheet4 = pd.read_excel(
    io=file_path,
    index_col=None,
    sheet_name='过程数据(process parameter)', )
```

In [ ]:

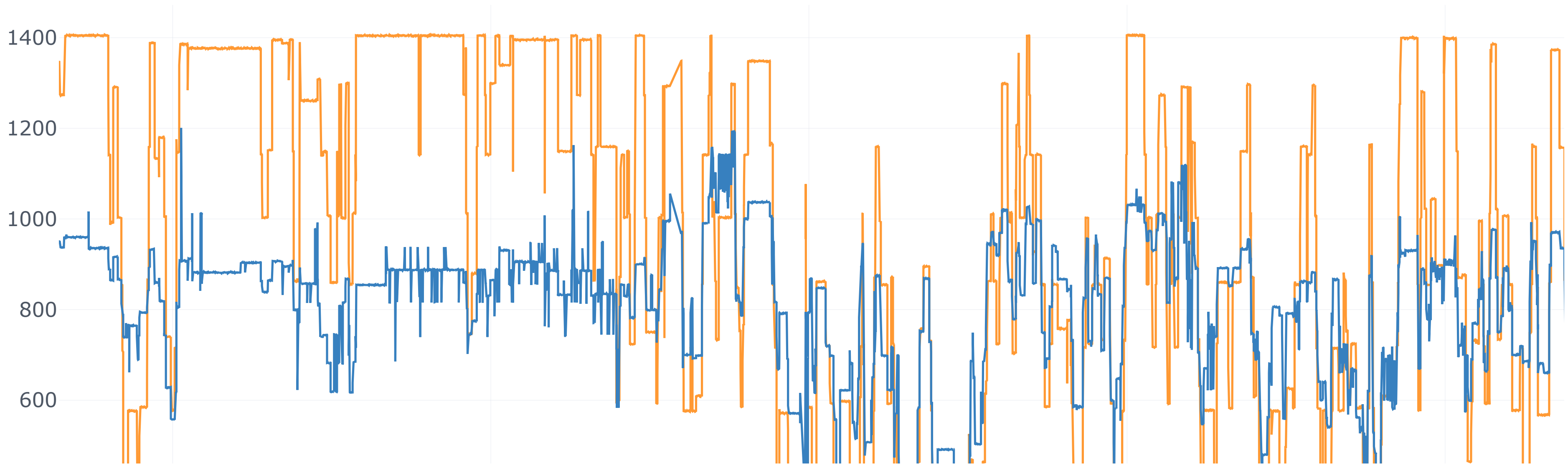
```python
# mitosheet.sheet(sheet1, sheet2, sheet3, sheet4, analysis_to_replay="id-tfyraljimv")
```

In [ ]:

```python

```

In [3]:

```python
sheet1.iplot(x='时间 (Time)')
```



准备数据

表1——温度(temperature)

In [4]:
```
# todo 找到有效的温度数据
sheet1_time_string = sheet1.iloc[:, 0].astype('string')

cond1 = sheet1_time_string.apply(lambda x: x[14: 16]) == "50"
data_part1 = sheet1[cond1].iloc[:-2, :]

exp_date1 = [
```

```
    "2022-02-03 20:50:00",
    "2022-02-26 13:50:00",
    "2022-03-21 06:50:00",
    "2022-04-04 10:50:00", "2022-04-04 15:50:00",
    "2022-03-10 10:50:00", "2022-03-10 11:50:00", "2022-03-10 12:50:00",
]
cond1 = sheet1_time_string.apply(lambda x: x in exp_date1)
data_part1 = data_part1[cond1.apply(lambda x: not x)]
data_part1.index = [i for i in range(len(data_part1))]
print(data_part1.shape)
# data_part1
# mitosheet.sheet(data_part1, analysis_to_replay="id-conydfcblv")
```

```
(1725, 3)
```

In [ ]:

```

```

表2——产品质量(quality of the products)

In [5]:

```
# todo 找到有效的产品质量数据
sheet2_time_string = sheet2.iloc[:, 0].astype('string')

exp_date2 = [
    "2022-02-20 23:50:00", "2022-02-21 00:50:00", "2022-02-21 01:50:00",

    "2022-02-21 09:50:00", "2022-02-21 10:50:00", "2022-02-21 02:50:00", "2022-02-21 03:50:00", "2022-02-21 04:50:00",
    "2022-02-21 05:50:00", "2022-02-21 06:50:00", "2022-02-21 07:50:00", "2022-02-21 08:50:00",

    "2022-02-26 06:50:00", "2022-02-26 07:50:00", "2022-02-26 08:50:00", "2022-02-26 09:50:00", "2022-02-26 10:50:00",

    "2022-04-08 00:50:00", "2022-04-08 01:50:00",
]

cond2 = sheet2_time_string.apply(lambda x: x in exp_date2)
data_part2_ = sheet2[cond2.apply(lambda x: not x)].iloc[2:, :]
data_part2_.index = [i for i in range(len(data_part2_))]
print(data_part2_.shape)
# data_part2_
# mitosheet.sheet(data_part2_, analysis_to_replay="id-etdktbfykz")
```

```
(1725, 5)
```

In [6]:

```
data_part2_
```

Out[6]:

	时间 (Time)	指标A (index A)	指标B (index B)	指标C (index C)	指标D (index D)
0	2022-01-25 02:50:00	79.08	23.52	12.41	17.86
1	2022-01-25 03:50:00	79.29	22.94	11.72	17.86
2	2022-01-25 04:50:00	79.95	21.42	10.68	17.63
3	2022-01-25 05:50:00	80.20	21.20	10.16	16.92
4	2022-01-25 06:50:00	80.38	20.75	10.16	15.75
...	...	...	...	...	...
1720	2022-04-07 19:50:00	79.82	23.84	11.03	13.52
1721	2022-04-07 20:50:00	78.98	25.36	11.37	12.85
1722	2022-04-07 21:50:00	78.86	25.40	11.37	11.42
1723	2022-04-07 22:50:00	79.10	25.58	11.37	11.55
1724	2022-04-07 23:50:00	79.32	24.82	11.03	11.55

In [ ]:

In [7]:

```
# todo 计算合格数量、合格率
cond10 = 77.78 < data_part2_.iloc[:, 1]
cond11 = data_part2_.iloc[:, 1] < 80.33
cond2 = data_part2_.iloc[:, 2] < 24.15
cond3 = data_part2_.iloc[:, 3] < 17.15
cond4 = data_part2_.iloc[:, 4] < 15.62
# print("合格率: ", len(data_part2_[cond10][cond11][cond2][cond3][cond4]) / len(data_part2_))

# todo 找到产品是否合格
def is_qualified(x):
    return 77.78 < x[1] < 80.33 and x[2] < 24.15 and x[3] < 17.15 and x[4] < 15.62
data_part2 = pd.DataFrame(data_part2_.apply(is_qualified, axis=1))
data_part2.columns = ['是否合格']
# print(len(data_part2))
# print(data_part2.sum() / len(data_part2))

print(data_part2.shape)
print(data_part2.sum(), data_part2.sum() / len(data_part2))
# data_part2
# mitosheet.sheet(data_part2, analysis_to_replay="id-mkgwgrqoel")
```

```
(1725, 1)
是否合格    472
dtype: int64 是否合格    0.273623
dtype: float64
```

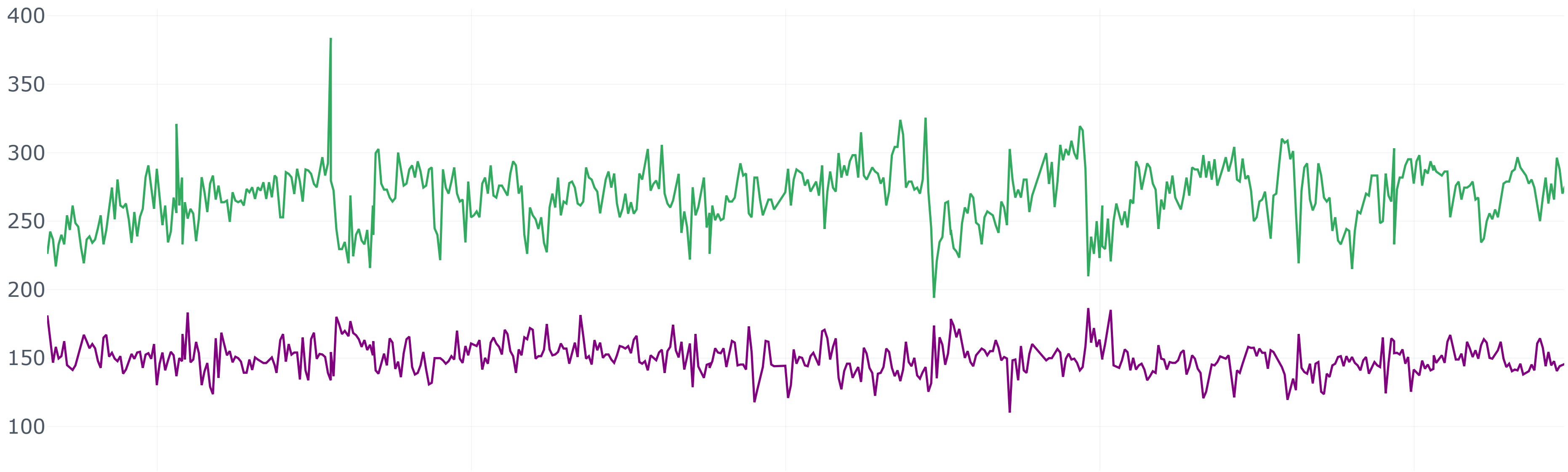
表3——原矿参数(mineral parameter)

In [8]:
```
# todo 找到原矿参数数据 3
cnt = data_part1.iloc[:, 0].astype('string').apply(lambda x: x[5: 10])
time_cnt = []
for i in pd.DataFrame(cnt).groupby(by='时间 (Time)'):
    time_cnt.append(len(i[1]))
data_part3 = pd.DataFrame(np.repeat(sheet3.iloc[:-4, :].values, time_cnt, axis=0), columns=sheet3.columns)
print(data_part3.shape)
# data_part3
# mitosheet.sheet(data_part3, analysis_to_replay="id-rvruqimmlr")
```

```
(1725, 5)
```

表4——过程数据(process parameter)

In [9]:
```
cols = ['时间 (Time)', "过程数据3 (Process parameter 3)", "过程数据4 (Process parameter 4)"]
proc_data = pd.DataFrame(sheet4)
proc_data.iplot(x='时间 (Time)')
print("相关系数: ")
proc_data.iloc[:, 1:].corr()
```



相关系数：

Out[9]:

	过程数据1 (Process parameter 1)	过程数据2 (Process parameter 2)	过程数据3 (Process parameter 3)	过程数据4 (Process parameter 4)
过程数据1 (Process parameter 1)	1.000000	NaN	0.058849	-0.147755
过程数据2 (Process parameter 2)	NaN	NaN	NaN	NaN
过程数据3 (Process parameter 3)	0.058849	NaN	1.000000	-0.497288
过程数据4 (Process parameter 4)	-0.147755	NaN	-0.497288	1.000000

In [ ]:

In [10]:

```
def norm(data):
    return (data - data.min()) / (data.max() - data.min())
```

```python
data_part4 = sheet4.apply(lambda x: (x[3] + x[4]), axis=1)
data_part4 = pd.concat([sheet4.iloc[:, 0], data_part4], axis=1).rename(columns={0: "原矿质量"})
print(data_part4.shape)
# data_part4
# mitosheet.sheet(data_part4, analysis_to_replay="id-lntnexsmmk")
```

(619, 2)

In [11]:

```python
exp_date4 = []
for i in exp_date1 + exp_date2:
    exp_date4.append(i[:-5] + "30")
# print(exp_date4)

sheet4_time_string = data_part4.iloc[:, 0].astype('string')
cond4 = sheet4_time_string.apply(lambda x: x[: -3] not in exp_date4)

data_part4_need = data_part4[cond4]
data_part4_need = data_part4_need.iloc[:-33, :]

for _ in range(5):
    data_part4_need.drop(index=np.random.randint(0, len(data_part4_need)), inplace=True)
data_part4_need.index = [i for i in range(len(data_part4_need))]

data_part4_need = pd.DataFrame(np.repeat(data_part4_need.values, 3, axis=0), columns=data_part4_need.columns)
print(data_part4_need.shape)
# data_part4_need
# mitosheet.sheet(data_part4_need, analysis_to_replay="id-owygulbcev")
```

(1725, 2)

In [12]:

```python
# mitosheet.sheet(data_part1, data_part2, data_part3, data_part4_need, analysis_to_replay="id-gwrmcdoymx")
```

In [13]:

```python
data_part4_need
```

Out[13]:

	时间 (Time)	原矿质量
0	2022-01-25 02:30:11	407.39
1	2022-01-25 02:30:11	407.39
2	2022-01-25 02:30:11	407.39
3	2022-01-25 05:30:13	406.89
4	2022-01-25 05:30:13	406.89
...	...	...
1720	2022-04-07 20:30:17	485.59
1721	2022-04-07 20:30:17	485.59
1722	2022-04-07 23:30:10	440.34
1723	2022-04-07 23:30:10	440.34
1724	2022-04-07 23:30:10	440.34

In [14]:
```
X = pd.concat([data_part1.iloc[:, 1:], data_part3.iloc[:, 1:], data_part4_need.iloc[:, 1:]], axis=1)
Ys = data_part2
```

In [15]:
```
# mitosheet.sheet(X, Ys, analysis_to_replay="id-qrekqeyfua")
```

In [ ]:

In [16]:
```
X.to_csv("quention3-X_data.csv")
Ys.to_csv("quention3-Y_data.csv")
```

In [17]:
```
cond = (pd.notna(X).iloc[:, 0] == True)
remain_index = X[cond].index
```

In [18]:
```
X = X[cond]
Y = Ys[cond].replace(to_replace=[True, False], value=[1, 0])
print(X.shape, Y.shape)
```

```
(1640, 7) (1640, 1)
```

In [19]:
```
# mitosheet.sheet(X, Y, analysis_to_replay="id-kzkchyyfcx")
```

In [ ]:

预测是否合格-思路1

思路一：二分类——直接预测是否合格，然后计算合格率

In [20]:

```
# 04-08 04-09 处理即将预测的数据
data_to_predict = np.array(
    [[341.40, 665.04, 52.88, 91.27, 47.22, 22.26, ],      # 8
     [1010.32, 874.47, 54.44, 92.12, 48.85, 21.83, ],],  # 8
)
data_to_predict = np.repeat(data_to_predict, [8, 9], axis=0)
data_to_predict = np.concatenate([data_to_predict, np.array(data_part4.iloc[586:586 + 17:, 1:])], axis=1)
# mitosheet.sheet(pd.DataFrame(data_to_predict), analysis_to_replay="id-edacdozavl")
```

In [21]:

```
feature_name = list(X.columns)
feature_name
```

Out[21]:

```
['系统I温度 (Temperature of system I)',
 '系统II温度 (Temperature of system II)',
 '原矿参数1 (Mineral parameter 1)',
 '原矿参数2 (Mineral parameter 2)',
 '原矿参数3 (Mineral parameter 3)',
 '原矿参数4 (Mineral parameter 4)',
 '原矿质量']
```

In [22]:

```
X
```

Out[22]:

	系统I温度 (Temperature of system I)	系统II温度 (Temperature of system II)	原矿参数1 (Mineral parameter 1)	原矿参数2 (Mineral parameter 2)	原矿参数3 (Mineral parameter 3)	原矿参数4 (Mineral parameter 4)	原矿质量
0	1347.49	950.40	55.26	108.03	43.29	20.92	407.39
1	1274.43	938.20	55.26	108.03	43.29	20.92	407.39
2	1273.86	938.16	55.26	108.03	43.29	20.92	407.39
3	1273.51	937.49	55.26	108.03	43.29	20.92	406.89
4	1272.84	936.67	55.26	108.03	43.29	20.92	406.89
...	...	...	...	...	...	...	...
1720	437.71	540.70	54.4	105.14	49.03	20.82	485.59
1721	494.23	557.21	54.4	105.14	49.03	20.82	485.59
1722	495.47	557.68	54.4	105.14	49.03	20.82	440.34
1723	494.41	572.00	54.4	105.14	49.03	20.82	440.34
1724	495.03	571.61	54.4	105.14	49.03	20.82	440.34

In [ ]:

1. 逻辑回归

In [23]:

```
from sklearn.linear_model import LogisticRegression as LR
from sklearn.linear_model import LogisticRegressionCV as LRCV

test_size = 0.3

models_lr = [
    LR(
        penalty="l2",
        C=1.0,
        random_state=None,
        solver="lbfgs",
        max_iter=3000,
        multi_class='ovr',
        verbose=0,
    ),
    LRCV(
        penalty="l2",
        random_state=None,
```

```
        solver="lbfgs",
        max_iter=3000,
        multi_class='ovr',
        verbose=0,
    ),
]

for model in models_lr:
    print("模型: ", model)
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=test_size, random_state=10, shuffle=True)

    # todo 训练训练集
    model.fit(xtrain, ytrain)
    # todo 预测测试集
    yhat = model.predict(xtest)

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhat))
    # auc
    print('AUC:', roc_auc_score(ytest, yhat))

    # todo 预测
    pred = model.predict(data_to_predict)
    print("预测结果: ", pred[:8])
    print(color(f"准确率: {pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果: ", pred[8:])
    print(color(f"准确率: {pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
```

```
模型： LogisticRegression(max_iter=3000, multi_class='ovr')
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.73      0.97      0.83       355
           1       0.44      0.05      0.09       137

    accuracy                           0.72       492
   macro avg       0.58      0.51      0.46       492
weighted avg       0.65      0.72      0.63       492

AUC: 0.5128713889174463
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
模型： LogisticRegressionCV(max_iter=3000, multi_class='ovr')
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.73      0.97      0.83       355
           1       0.40      0.04      0.08       137

    accuracy                           0.72       492
   macro avg       0.56      0.51      0.46       492
weighted avg       0.63      0.72      0.62       492

AUC: 0.5092217538809499
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
```

2. 决策树

In [24]:

```
from sklearn.tree import ExtraTreeClassifier as ETC, DecisionTreeClassifier as DTC

model_dt = [
    ETC(),
    DTC(),
]
for model in model_dt:
    print("模型： ", model)
```

```
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=test_size, random_state=10, shuffle=True)

    # todo 训练训练集
    model.fit(xtrain, ytrain)
    # todo 预测测试集
    yhat = model.predict(xtest)

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhat))
    # auc
    print('AUC:', roc_auc_score(ytest, yhat))

    # todo 预测
    pred = model.predict(data_to_predict)
    print("预测结果：", pred[:8])
    print(color(f"准确率：{pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果：", pred[8:])
    print(color(f"准确率：{pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
```

```
模型：  ExtraTreeClassifier()
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       355
           1       0.55      0.53      0.54       137

    accuracy                           0.75       492
   macro avg       0.69      0.68      0.68       492
weighted avg       0.75      0.75      0.75       492

AUC: 0.6819163154107125
预测结果：  [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果：  [0 0 0 1 1 0 0 0 0]
准确率：0.2222222222222222
----------------------------------------------------------------------------------------------------
模型：  DecisionTreeClassifier()
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.81      0.80      0.81       355
           1       0.51      0.53      0.52       137

    accuracy                           0.73       492
   macro avg       0.66      0.66      0.66       492
weighted avg       0.73      0.73      0.73       492

AUC: 0.6641821733319626
预测结果：  [1 1 1 1 1 1 1 1]
准确率：1.0
预测结果：  [0 0 0 1 1 1 1 0 0]
准确率：0.4444444444444444
----------------------------------------------------------------------------------------------------
```

3. 随机森林

In [25]:

```
from sklearn.ensemble import AdaBoostClassifier as ABC
from sklearn.ensemble import BaggingClassifier as BC
from sklearn.ensemble import ExtraTreesClassifier as ETC
from sklearn.ensemble import GradientBoostingClassifier as GBC
from sklearn.ensemble import RandomForestClassifier as RFC

model_rf = [
    ABC(),
```

```python
    BC(),
    ETC(),
    GBC(),
    RFC(),
]
for model in model_rf:
    print("模型: ", model)
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=test_size, random_state=10, shuffle=True)

    # todo 训练训练集
    model.fit(xtrain, ytrain)
    # todo 预测测试集
    yhat = model.predict(xtest)

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhat))
    # auc
    print('AUC:', roc_auc_score(ytest, yhat))

    # todo 预测
    pred = model.predict(data_to_predict)
    print("预测结果: ", pred[:8])
    print(color(f"准确率: {pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果: ", pred[8:])
    print(color(f"准确率: {pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
```

模型： AdaBoostClassifier()
主要分类指标的文本报告：

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.83       355
           1       0.53      0.28      0.36       137

    accuracy                           0.73       492
   macro avg       0.65      0.59      0.60       492
weighted avg       0.70      0.73      0.70       492
```

AUC: 0.5907988074431992
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0

模型： BaggingClassifier()
主要分类指标的文本报告：

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       355
           1       0.64      0.44      0.52       137

    accuracy                           0.77       492
   macro avg       0.72      0.67      0.69       492
weighted avg       0.76      0.77      0.76       492
```

AUC: 0.671090778246119
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0

模型： ExtraTreesClassifier()
主要分类指标的文本报告：

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       355
           1       0.62      0.56      0.59       137

    accuracy                           0.78       492
   macro avg       0.73      0.71      0.72       492
weighted avg       0.78      0.78      0.78       492
```

AUC: 0.7134162640074021

```
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
模型： GradientBoostingClassifier()
主要分类指标的文本报告：
              precision    recall  f1-score   support

           0       0.78      0.91      0.84       355
           1       0.60      0.34      0.43       137

    accuracy                           0.75       492
   macro avg       0.69      0.62      0.64       492
weighted avg       0.73      0.75      0.73       492

AUC: 0.6242212398478463
预测结果： [1 1 1 0 0 0 0 0]
准确率：0.375
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
模型： RandomForestClassifier()
主要分类指标的文本报告：
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       355
           1       0.63      0.50      0.56       137

    accuracy                           0.78       492
   macro avg       0.73      0.70      0.71       492
weighted avg       0.77      0.78      0.77       492

AUC: 0.6954867893492342
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
```

4. XGBoost

In [26]:

```
from xgboost import XGBClassifier as XGBC
from xgboost import XGBRFClassifier as XGBRFC
```

```python
models_xgb = [
    XGBC(),
    XGBRFC(),
]
for model in models_xgb:
    print("模型: ", model)
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=test_size, random_state=10, shuffle=True)

    # todo 训练训练集
    model.fit(xtrain.values, ytrain)
    # todo 预测测试集
    yhat = model.predict(xtest.values)

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhat))

    # auc
    print('AUC:', roc_auc_score(ytest, yhat))

    # todo 预测
    pred = model.predict(data_to_predict)
    print("预测结果: ", pred[:8])
    print(color(f"准确率: {pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果: ", pred[8:])
    print(color(f"准确率: {pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
```

```
模型：  XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
[18:35:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the defaul
t evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd li
ke to restore the old behavior.
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.82      0.85      0.84       355
           1       0.58      0.52      0.55       137

    accuracy                           0.76       492
   macro avg       0.70      0.69      0.69       492
weighted avg       0.75      0.76      0.76       492

AUC: 0.6858846509715225
预测结果：  [0 0 0 0 0 0 0 0 0]
准确率：0.0
预测结果：  [0 0 0 0 1 0 1 0 0]
准确率：0.2222222222222222
----------------------------------------------------------------------------------------------------
模型：  XGBRFClassifier(base_score=None, booster=None, colsample_bylevel=None,
                colsample_bytree=None, gamma=None, gpu_id=None,
                importance_type='gain', interaction_constraints=None,
                max_delta_step=None, max_depth=None, min_child_weight=None,
                missing=nan, monotone_constraints=None, n_estimators=100,
                n_jobs=None, num_parallel_tree=None,
                objective='binary:logistic', random_state=None, reg_alpha=None,
                scale_pos_weight=None, tree_method=None,
                validate_parameters=None, verbosity=None)
[18:35:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the defaul
t evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd li
ke to restore the old behavior.
主要分类指标的文本报告:
              precision    recall  f1-score   support

           0       0.78      0.94      0.85       355
           1       0.67      0.32      0.43       137
```

```
    accuracy                           0.77       492
   macro avg       0.72      0.63      0.64       492
weighted avg       0.75      0.77      0.74       492

AUC: 0.6295980261128816
预测结果： [0 0 0 0 0 0 0 0]
准确率：0.0
预测结果： [0 0 0 0 0 0 0 0 0]
准确率：0.0
----------------------------------------------------------------------------------------------------
```

5. BP

In [27]:
```python
from hmz.math_model.predict import BP

test_size = 0.3
hidden_num = [5]
lr=0.01
epoch=10
batch_size=64
activate='relu'
criterion=None
optimizer='adam'
normalization=True

def run_BP(X=X, Y=Y):
    xtrain, xtest, ytrain, ytest = train_test_split(
        np.array(X, dtype=float),
        np.squeeze(np.array(Y, dtype=float)),
        test_size=test_size,
        random_state=1,
        shuffle=True,
    )
#     print(ytrain, type(ytrain), type(ytrain), ytrain.shape)
    bp = BP(
        X.shape[1], hidden_num, 2,
        lr=lr,
        epoch=epoch,
        optimizer=optimizer,
    )
    bp.train(xtrain, ytrain)
    yhat = bp.predict(xtest).cpu().detach().numpy()

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
```

```python
    print(classification_report(ytest, yhat))
    # auc
    print('AUC:', roc_auc_score(ytest, yhat))

    # todo 预测
    pred = bp.predict(data_to_predict)
    print("预测结果：", pred[:8])
    print(color(f"合格率：{pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果：", pred[8:])
    print(color(f"合格率：{pred[8:].sum() / len(pred[8:])}"),  )
    return bp
bp = run_BP()
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 5]              40
            Linear-2                   [64, 5]              40
              ReLU-3                   [64, 5]               0
              ReLU-4                   [64, 5]               0
            Linear-5                   [64, 2]              12
            Linear-6                   [64, 2]              12
================================================================
Total params: 104
Trainable params: 104
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

```
epoch: 9, train acc: 0.72, train loss: 8.36, eval acc: 0.73, eval loss: 2.25: 100%|████| 10/10 [00:00<00:00, 19.79it/s]
```

```
主要分类指标的文本报告：
              precision    recall  f1-score   support

         0.0       0.76      0.96      0.85       358
         1.0       0.62      0.19      0.30       134

    accuracy                           0.75       492
   macro avg       0.69      0.57      0.57       492
weighted avg       0.72      0.75      0.70       492

AUC: 0.5746685566580505
预测结果： tensor([0, 0, 0, 0, 0, 0, 0, 0], device='cuda:0')
合格率：0.0
预测结果： tensor([1, 1, 1, 1, 1, 1, 1, 1, 1], device='cuda:0')
合格率：1.0
```

预测是否合格-思路2

思路二：回归——预测四个指标，然后根据所预测的指标判断是否合格，最后计算合格率

准备数据

In [28]:
```python
X = pd.concat([data_part1.iloc[:, 1:], data_part3.iloc[:, 1:], data_part4_need.iloc[:, 1:]], axis=1)
Ys = data_part2_.iloc[:, 1:]
```

In [29]:
```python
cond = (pd.notna(X).iloc[:, 0] == True)
remain_index = X[cond].index
```

In [30]:
```python
X = X[cond]
Y = Ys[cond].replace(to_replace=[True, False], value=[1, 0])
print(X.shape, Y.shape)
```

```
(1640, 7) (1640, 4)
```

预测指标：线性回归、决策树、随机森林、XGBoost、BP

In [31]:
```python
# todo 找到产品是否合格
def is_qualified2(x):
    return 77.78 < x[0] < 80.33 and x[1] < 24.15 and x[2] < 17.15 and x[3] < 15.62
```

首先查看哪个模型效果最好

In [32]:
```python
# todo all sklearn models: 所有 sklearn 中的（较好）回归模型

from copy import copy
from sklearn.linear_model import LinearRegression as LR
from sklearn.tree import ExtraTreeRegressor as ETC, DecisionTreeRegressor as DTC
from sklearn.ensemble import AdaBoostRegressor as ABC
from sklearn.ensemble import BaggingRegressor as BC
from sklearn.ensemble import ExtraTreesRegressor as ETC
from sklearn.ensemble import GradientBoostingRegressor as GBC
from sklearn.ensemble import RandomForestRegressor as RFC
from xgboost import XGBRegressor as XGBC
from xgboost import XGBRFRegressor as XGBRFC

models_name = [
    "线性回归",
    "决策树",
    "随机森林",
    "XGBoost",
]

models = [
    LR(),
    ETC(),
    RFC(),
    XGBC(),
#     XGBRFC(),
]


for model in models:
    print("模型: ", model)
    xx = np.array(X)  # (n, 7)
    yy = np.array(Y)  # (n, 4)
    xtrain, xtest, ytrain, ytest = train_test_split(
        xx, yy,
        test_size=0.3,
        random_state=10,
        shuffle=True,
    )

    yhats = []
    ypreds = []
    for i in range(yy.shape[1]):
        m = copy(model)
        m.fit(xtrain, ytrain[:, i])
        yhat = m.predict(xtest)
```

```
        yhats.append(list(yhat))
        ypred = m.predict(data_to_predict)
        ypreds.append(list(ypred))

    yhats = pd.DataFrame(yhats, index=["指标A", "指标B", "指标C", "指标D"]).T
    ytest = pd.DataFrame(ytest, columns=["指标A", "指标B", "指标C", "指标D"])

    yhats = np.squeeze(pd.DataFrame(yhats.apply(is_qualified2, axis=1)))
    ytest = np.squeeze(pd.DataFrame(ytest.apply(is_qualified2, axis=1)))

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhats))
    # auc
    print('AUC:', roc_auc_score(ytest, yhats))
    print()

    # todo 预测结果
    ypreds = pd.DataFrame(ypreds, index=["指标A", "指标B", "指标C", "指标D"]).T
    ypreds = np.array(pd.DataFrame(ypreds.apply(is_qualified2, axis=1)))
    pred = ypreds
#     pred = np.reshape(ypreds, (-1, 1))
    print("预测结果: ", pred[:8])
    print(color(f"合格率: {pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果: ", pred[8:])
    print(color(f"合格率: {pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
```

```
模型： LinearRegression()
主要分类指标的文本报告:
              precision    recall  f1-score   support

       False       0.77      0.62      0.69       355
        True       0.35      0.53      0.42       137

    accuracy                           0.59       492
   macro avg       0.56      0.57      0.55       492
weighted avg       0.66      0.59      0.61       492

AUC: 0.5748740618895858

预测结果： [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
合格率：0.0
预测结果： [[False]
 [False]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [False]
 [False]]
合格率：0.5555555555555556
----------------------------------------------------------------------------------------------------
模型： ExtraTreesRegressor()
主要分类指标的文本报告:
              precision    recall  f1-score   support

       False       0.84      0.85      0.85       355
        True       0.60      0.59      0.60       137

    accuracy                           0.78       492
   macro avg       0.72      0.72      0.72       492
weighted avg       0.78      0.78      0.78       492

AUC: 0.7209725506322607
```

```
预测结果： [[ True]
 [ True]
 [ True]
 [False]
 [False]
 [False]
 [ True]
 [ True]]
合格率：0.625
预测结果： [[False]
 [False]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [False]
 [False]]
合格率：0.5555555555555556
----------------------------------------------------------------------------------------------------
模型： RandomForestRegressor()
主要分类指标的文本报告：
              precision    recall  f1-score   support

       False       0.83      0.84      0.83       355
        True       0.57      0.54      0.55       137

    accuracy                           0.76       492
   macro avg       0.70      0.69      0.69       492
weighted avg       0.75      0.76      0.76       492

AUC: 0.6911997532641102

预测结果： [[ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]]
合格率：1.0
预测结果： [[False]
 [False]
 [ True]
 [ True]
```

```
 [ True]
 [ True]
 [ True]
 [False]
 [ True]]
合格率：0.6666666666666666
----------------------------------------------------------------------------------------------------
模型： XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
主要分类指标的文本报告：
              precision    recall  f1-score   support

       False       0.85      0.83      0.84       355
        True       0.59      0.61      0.60       137

    accuracy                           0.77       492
   macro avg       0.72      0.72      0.72       492
weighted avg       0.78      0.77      0.77       492

AUC: 0.7234707515163977

预测结果： [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
合格率：0.0
预测结果： [[False]
 [False]
 [ True]
 [False]
 [False]
 [ True]
 [False]
 [False]
 [False]]
```

合格率：0.2222222222222222

In [33]:
```python
# todo my bp nn model: 自己写的 BP 神经网络
from hmz.math_model.predict import BP

test_size = 0.3
hidden_num = [10]
lr=0.01
epoch=10
batch_size=64
activate='relu'
criterion=None
optimizer='adam'
normalization=True

def run_BP(X=X, Y=Y):
    print("BP模型：")
    xx = np.array(X, dtype=float)  # (n, 7)
    yy = np.array(Y, dtype=float)  # (n, 4)
    xtrain, xtest, ytrain, ytest = train_test_split(
        xx, yy,
        test_size=0.3,
        random_state=10,
        shuffle=True,
    )

    yhats = []
    ypreds = []
    for i in range(yy.shape[1]):
        bp = BP(
            X.shape[1], hidden_num, 1,
            lr=lr,
            epoch=epoch,
            optimizer=optimizer,
            activate='sigmoid',
        )

        bp.train(xtrain, ytrain[:, i])
        yhat = bp.predict(xtest).cpu().detach().numpy()
        yhats.append(list(yhat))
        ypred = bp.predict(data_to_predict).cpu().detach().numpy()
        ypreds.append(list(np.squeeze(ypred)))
#     print(ypreds)
    yhats = pd.DataFrame(yhats, index=["指标A", "指标B", "指标C", "指标D"]).T
    ytest = pd.DataFrame(ytest, columns=["指标A", "指标B", "指标C", "指标D"])
```

```
    yhats = np.squeeze(pd.DataFrame(yhats.apply(is_qualified2, axis=1))).astype(float)
    ytest = np.squeeze(pd.DataFrame(ytest.apply(is_qualified2, axis=1))).astype(float)
    print(ytest)
#     print(yhats, ytest)
    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhats))
    # auc
    print('AUC:', roc_auc_score(ytest, yhats))
    print()

    # todo 预测结果
    ypreds = pd.DataFrame(ypreds, index=["指标A", "指标B", "指标C", "指标D"]).T
    ypreds = np.array(pd.DataFrame(ypreds.apply(is_qualified2, axis=1)))
    pred = ypreds
#     pred = np.reshape(ypreds, (-1, 1))
    print("预测结果: ", pred[:8])
    print(color(f"合格率: {pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果: ", pred[8:])
    print(color(f"合格率: {pred[8:].sum() / len(pred[8:])}"),  )
    print('-'*100)
    return bp
bp = run_BP()
```

```
BP模型:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 10]              80
            Linear-2                   [64, 10]              80
           Sigmoid-3                   [64, 10]               0
           Sigmoid-4                   [64, 10]               0
            Linear-5                    [64, 1]              11
            Linear-6                    [64, 1]              11
================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
epoch: 9, train loss: 62632.07, eval loss: 16166.99: 100%|██████████| 10/10 [00:00<00:00, 22.90it/s]
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 10]              80
            Linear-2                   [64, 10]              80
           Sigmoid-3                   [64, 10]               0
           Sigmoid-4                   [64, 10]               0
            Linear-5                    [64, 1]              11
            Linear-6                    [64, 1]              11
================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
epoch: 9, train loss: 2210.94, eval loss: 518.19: 100%|████████████████████████████████| 10/10 [00:00<00:00, 22.50it/s]
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 10]              80
            Linear-2                   [64, 10]              80
           Sigmoid-3                   [64, 10]               0
           Sigmoid-4                   [64, 10]               0
            Linear-5                    [64, 1]              11
            Linear-6                    [64, 1]              11
================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
epoch: 9, train loss: 45.21, eval loss: 10.84: 100%|███████████████████████████████████| 10/10 [00:00<00:00, 22.87it/s]
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 10]              80
            Linear-2                   [64, 10]              80
           Sigmoid-3                   [64, 10]               0
           Sigmoid-4                   [64, 10]               0
            Linear-5                    [64, 1]              11
            Linear-6                    [64, 1]              11
================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
```

```
epoch: 9, train loss: 317.33, eval loss: 82.73: 100%|██████████████████████████████████| 10/10 [00:00<00:00, 21.42it/s]
```

```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
      ...
487    1.0
488    0.0
489    0.0
490    1.0
491    0.0
Name: 0, Length: 492, dtype: float64
主要分类指标的文本报告：
              precision    recall  f1-score   support

         0.0       0.72      1.00      0.84       355
         1.0       0.00      0.00      0.00       137

    accuracy                           0.72       492
   macro avg       0.36      0.50      0.42       492
weighted avg       0.52      0.72      0.60       492

AUC: 0.5

预测结果： [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
合格率：0.0
预测结果： [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
合格率：0.0
----------------------------------------------------------------------------------------------------
```

最终筛选出 XGBoost 效果最好，四个指标都是

多次运行取到模型结果满意为止doge

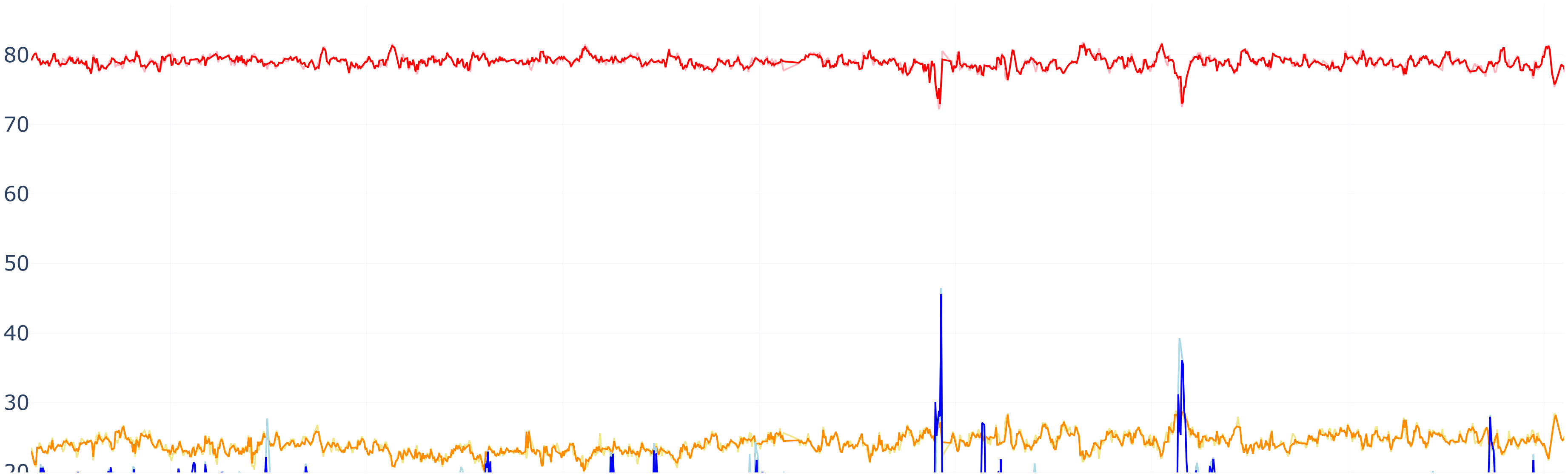
In [35]:
```
index_num = Y.shape[1]
index_name = ["指标A", "指标B", "指标C", "指标D"]
index_colors = ["red", "lightpink", "darkorange", "khaki", "green", "lightgreen", "blue", "lightblue"]

models_name = [
    "XGBoost",
]

models = [
    XGBC(),
]
model_name = 'XGBoost'
datadata = []
width = 1000
height = 700

for model in models:
    print("模型: ", model)
    xx = np.array(X)  # (n, 7)
    yy = np.array(Y)  # (n, 4)
    xtrain, xtest, ytrain, ytest = train_test_split(
        xx, yy,
        test_size=0.3,
        random_state=10,
        shuffle=True,
    )

    yhats = []
    ypreds = []
    for i in range(yy.shape[1]):
        m = copy(model)
        m.fit(xtrain, ytrain[:, i])
        yhat = m.predict(xtest)
        yhats.append(list(yhat))

        ypred = m.predict(data_to_predict)
        ypreds.append(list(ypred))

        # 画图
        fig_y = m.predict(xx)
        datadata.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=yy[:, i],
            name=index_name[i] + "-真实值",
```

```
            line=dict(color=index_colors[i * 2 + 1], width=1)),
        )
        datadata.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=fig_y,
            name=index_name[i] + "-预测值",
            line=dict(color=index_colors[i * 2], width=1)),
        )

        # todo 画图：点差图
        cols = ["指标A", "指标B", "指标C", "指标D"][i]
        Yhat = pd.DataFrame(fig_y)
        Y.index = [i for i in range(len(yy[:, i]))]
        Y_data = pd.concat([pd.Series(yy[:, i]), pd.Series(fig_y)], axis=1)
        Y_data.columns = ["真实值", "预测值"]

        Y_data.figure(
            kind='spread',
            color=[index_colors[i * 2 + 1], index_colors[i * 2]],
            title='基于' +  model_name + '的' + cols + '预测模型',
        ).write_image('./img/问题3-基于' +  model_name + '的' + cols + '预测模型.svg', width=width, height=height)


    yhats = pd.DataFrame(yhats, index=["指标A", "指标B", "指标C", "指标D"]).T
    ytest = pd.DataFrame(ytest, columns=["指标A", "指标B", "指标C", "指标D"])

    yhats = np.squeeze(pd.DataFrame(yhats.apply(is_qualified2, axis=1)))
    ytest = np.squeeze(pd.DataFrame(ytest.apply(is_qualified2, axis=1)))

    # 主要分类指标的文本报告
    print('主要分类指标的文本报告:')
    print(classification_report(ytest, yhats))
    # auc
    print('AUC:', roc_auc_score(ytest, yhats))
    print()

    # todo 预测结果
    ypreds = pd.DataFrame(ypreds, index=["指标A", "指标B", "指标C", "指标D"]).T
    ypreds = np.array(pd.DataFrame(ypreds.apply(is_qualified2, axis=1)))
    pred = ypreds
#     pred = np.reshape(ypreds, (-1, 1))
    print("预测结果：", pred[:8])
    print(color(f"合格率：{pred[:8].sum() / len(pred[:8])}"),  )
    print("预测结果：", pred[8:])
    print(color(f"合格率：{pred[8:].sum() / len(pred[8:])}"),  )
    print()
```

```python
# 画图

fig = go.Figure(data=datadata, )

annotations = []
annotations.append(dict(
    x=0.5, y=-0.1,
    xref='paper', yref='paper',
    xanchor='center', yanchor='top',
    text='时间',
    font=dict(size=16),
    showarrow=False,
))
fig.update_layout(
    title='基于' +  model_name + '的指标预测模型',
    annotations=annotations,
    template="plotly_white",
)
fig.write_image('./img/问题3-基于' + model_name + '的指标预测模型.svg', width=width, height=height)
fig.show()
```

```
模型： XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
主要分类指标的文本报告:
              precision    recall  f1-score   support

       False       0.85      0.83      0.84       355
        True       0.59      0.61      0.60       137

    accuracy                           0.77       492
   macro avg       0.72      0.72      0.72       492
weighted avg       0.78      0.77      0.77       492

AUC: 0.7234707515163977

预测结果： [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
合格率：0.0
预测结果： [[False]
 [False]
 [ True]
 [False]
 [False]
 [ True]
 [False]
 [False]
 [False]]
合格率：0.2222222222222222
```

基于XGBoost的指标预测模型




```
In [ ]:
```