问题1

该 notebook 已经整理完毕，可以直接运行整个文件

注意:

1. OS: Windows（不同操作系统可能会有一些其他的问题）
2. python3.8.13（python 版本尽量一致，不一样也不一定会有问题，能运行就行）
3. 务必**下载**所需的第三方库，或者**注释**掉报错的库
4. （目录已经是完整的，如果没有乱修改目录，就不用管这个）运行该文件之前，务必先运行同一目录下的 pre_work.ipynb 文件，确保保存图片的目录存在，否则运行会报错并且图片无法保存！

In [1]:
```python
import hmz
from hmz.math_model.predict import BP, predict_accuracy

import mitosheet
import numpy as np
import pandas as pd
import plotly
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")

# from scipy.stats import permutation_test

import sklearn
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

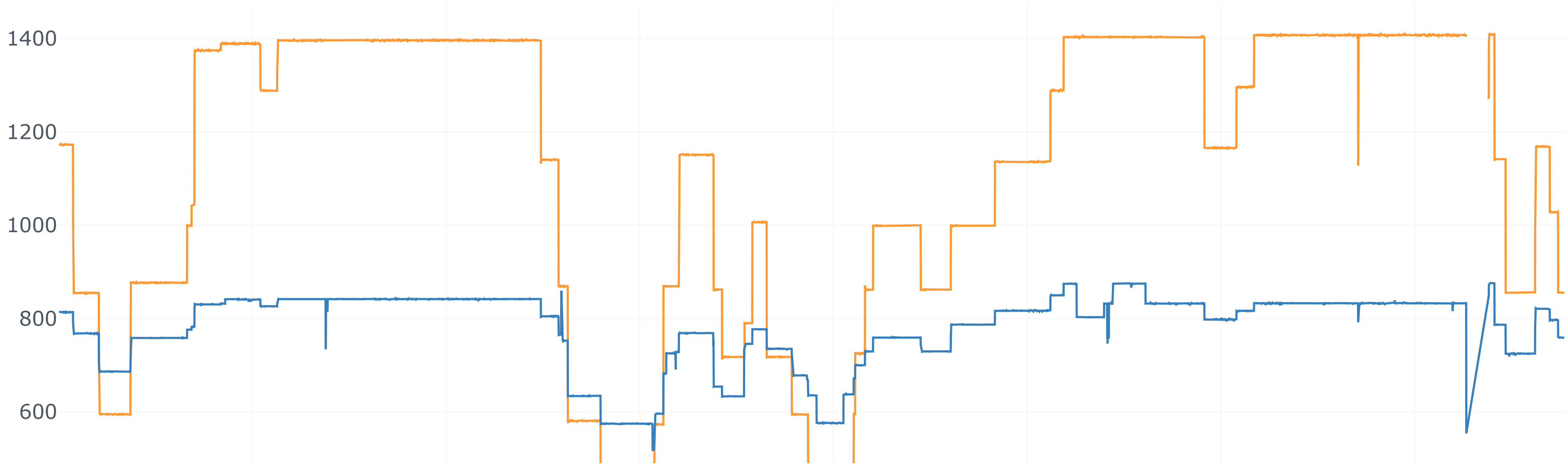
导入数据

观察数据

In [2]:
```
sheet1 = pd.read_excel(
    io="附件1(Attachment 1)2022-51MCM-Problem B.xlsx",
    index_col=None,
    sheet_name='温度(temperature)',
)
sheet2 = pd.read_excel(
    io="附件1(Attachment 1)2022-51MCM-Problem B.xlsx",
    index_col=None,
    sheet_name='产品质量(quality of the products)',
)
sheet3 = pd.read_excel(
    io="附件1(Attachment 1)2022-51MCM-Problem B.xlsx",
    index_col=None,
    sheet_name='原矿参数(mineral parameter)',
)
```

In [3]:
```
# mitosheet.sheet(sheet1, sheet2, sheet3)  # 查看数据，需要安装 mitosheet
```

In [4]:
```
# 绘制温度变化曲线
degree = sheet1.copy()
degree.columns = ['时间 (Time)', "系统I温度", "系统II温度"]
degree.figure(x='时间 (Time)').write_image("./img/问题1-系统1、2温度曲线.svg")
degree.iplot(x='时间 (Time)')
```



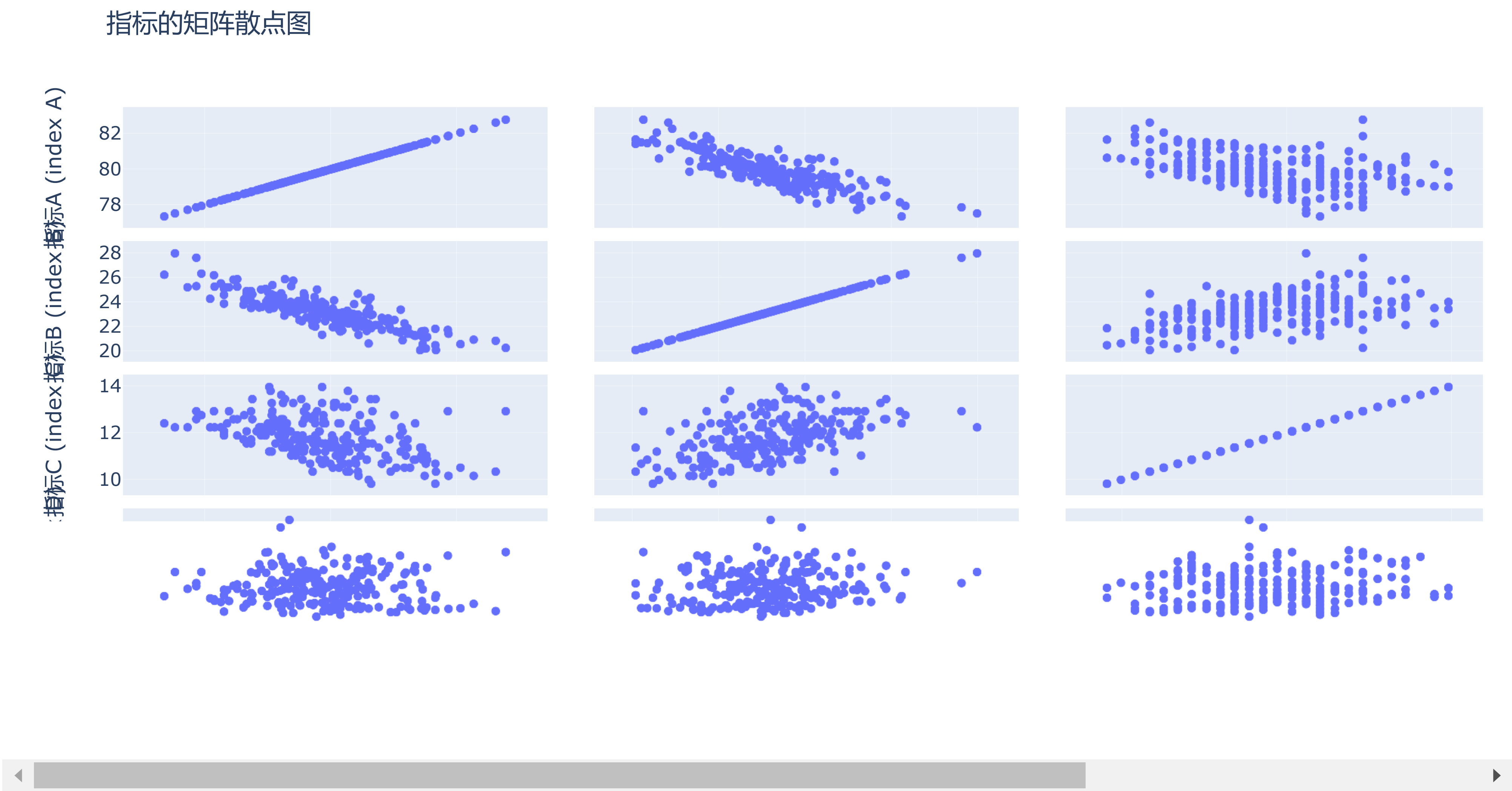
可以明显看到数据有缺失，具体如何处理见后面的代码（其实就是直接删除doge）

In [ ]:

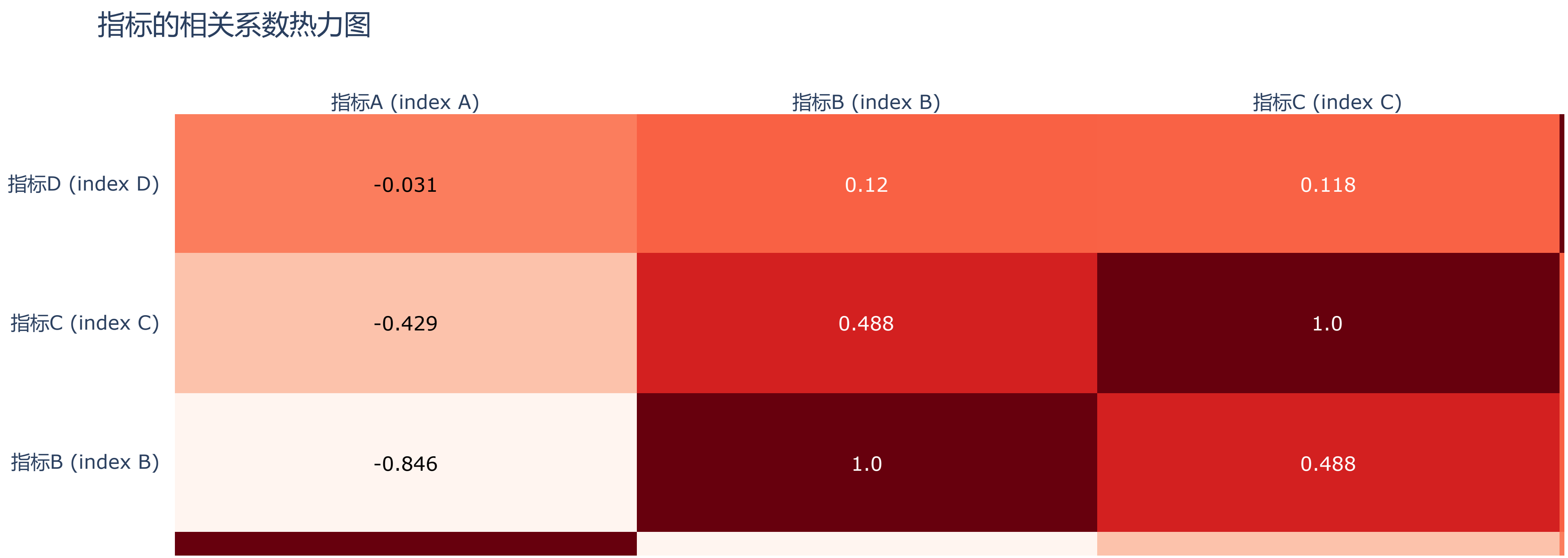
In [5]:

```
def norm(data):
    """原始数据标准化"""
    return (data - np.min(data)) / (np.max(data) - np.min(data))

def plot_ScatterMatrix_Heatmap(data, fig_pre_name, save=True):
    """绘制数据的矩阵散点图、相关系数热力图
    :param data: 数据
    :param fig_pre_name: 保存的文件名（不含路径！不含后缀！）
    :param save: 是否保存
```

```python
    :return: None
        notebook 自动展示绘制的图像
    """
    # 绘制矩阵散点图
    fig = px.scatter_matrix(
        data,
        title=fig_pre_name + '的矩阵散点图',
    )
    if save:
        fig.write_image('./img/问题1-' + fig_pre_name + '的矩阵散点图.svg')
    fig.show()

    # 绘制相关系数热力图
    corrs = data.corr(method='pearson')  # 'pearson', 'kendall', 'spearman'
    figure = ff.create_annotated_heatmap(
        z=corrs.values,
        x=list(corrs.columns),
        y=list(corrs.index),
        annotation_text=corrs.round(3).values,
        showscale=True,
        colorscale='reds',
    )
    figure.update_layout(title=fig_pre_name + '的相关系数热力图')
    if save:
        figure.write_image('./img/问题1-' + fig_pre_name + '的相关系数热力图.svg')
    figure.show()
    return None
index_data = sheet2.iloc[:, 1:]  # 除去时间这列的指标数据
plot_ScatterMatrix_Heatmap(index_data, '指标')
# index_data_norm = norm(index_data)  # norm data
# plot_ScatterMatrix_Heatmap(index_data_norm)  # norm data
```



指标的矩阵散点图
指标A (index A)
指标B (index B)
指标C (index C)
82
80
78
28
26
24
22
20
14
12
10




```
In [ ]:
```

准备数据

观察数据

```
In [6]: # todo 找到有效的温度数据
cond1= sheet1.iloc[:, 0].astype('string').apply(lambda x: x[14: 16]) == "50"
data_part1 = sheet1[cond1].iloc[:-2, :]
data_part1.index = [i for i in range(len(data_part1))]
print(data_part1.shape)
data_part1
```

```
(235, 3)
```

Out[6]:

	时间 (Time)	系统I温度 (Temperature of system I)	系统II温度 (Temperature of system II)
0	2022-01-13 00:50:00	1173.63	813.92
1	2022-01-13 01:50:00	854.55	767.64
2	2022-01-13 02:50:00	855.34	767.99
3	2022-01-13 03:50:00	853.57	766.20
4	2022-01-13 04:50:00	854.81	768.08
...	...	...	...
230	2022-01-22 17:50:00	1406.05	932.16
231	2022-01-22 18:50:00	1404.32	931.43
232	2022-01-22 19:50:00	1404.68	930.64
233	2022-01-22 20:50:00	1404.85	931.16
234	2022-01-22 21:50:00	1404.76	931.28

In [7]:

```python
# todo 找到有效的产品质量数据
sheet2_time_string = sheet2.iloc[:, 0].astype('string').apply(lambda x: x)
exp_date = ["2022-01-20 08:50:00", "2022-01-20 09:50:00", "2022-01-20 10:50:00"]
cond2 = sheet2_time_string.apply(lambda x: x == exp_date[0] or x == exp_date[1] or x == exp_date[2])
data_part2 = sheet2[cond2.apply(lambda x: not x)].iloc[2:, :]
print(data_part2.shape)
data_part2
```

```
(235, 5)
```

Out[7]:

	时间 (Time)	指标A (index A)	指标B (index B)	指标C (index C)	指标D (index D)
2	2022-01-13 02:50:00	78.15	26.21	12.93	14.59
3	2022-01-13 03:50:00	78.39	25.22	12.93	14.28
4	2022-01-13 04:50:00	79.22	24.60	12.41	13.70
5	2022-01-13 05:50:00	79.52	23.88	11.55	13.56
6	2022-01-13 06:50:00	80.04	23.48	11.55	13.47
...	...	...	...	...	...
235	2022-01-22 19:50:00	79.76	22.00	11.72	18.84
236	2022-01-22 20:49:00	80.51	22.00	11.37	18.53
237	2022-01-22 21:50:00	80.16	21.78	10.85	17.90
238	2022-01-22 22:50:00	79.79	22.58	11.20	17.05
239	2022-01-22 23:50:00	80.19	21.69	10.68	17.19

In [8]:

```
# todo 找到原矿参数数据
cnt = data_part1.iloc[:, 0].astype('string').apply(lambda x: x[8: 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[:-2, :].values, time_cnt, axis=0), columns=sheet3.columns)
print(data_part3.shape)
data_part3
```

```
(235, 5)
```

Out[8]:

	时间 (Time)	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)
0	2022-01-13	49.24	90.38	46.13	28.16
1	2022-01-13	49.24	90.38	46.13	28.16
2	2022-01-13	49.24	90.38	46.13	28.16
3	2022-01-13	49.24	90.38	46.13	28.16
4	2022-01-13	49.24	90.38	46.13	28.16
...	...	...	...	...	...
230	2022-01-22	54.74	93.05	49.03	21.48
231	2022-01-22	54.74	93.05	49.03	21.48
232	2022-01-22	54.74	93.05	49.03	21.48
233	2022-01-22	54.74	93.05	49.03	21.48
234	2022-01-22	54.74	93.05	49.03	21.48

In [9]:

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

In [ ]:

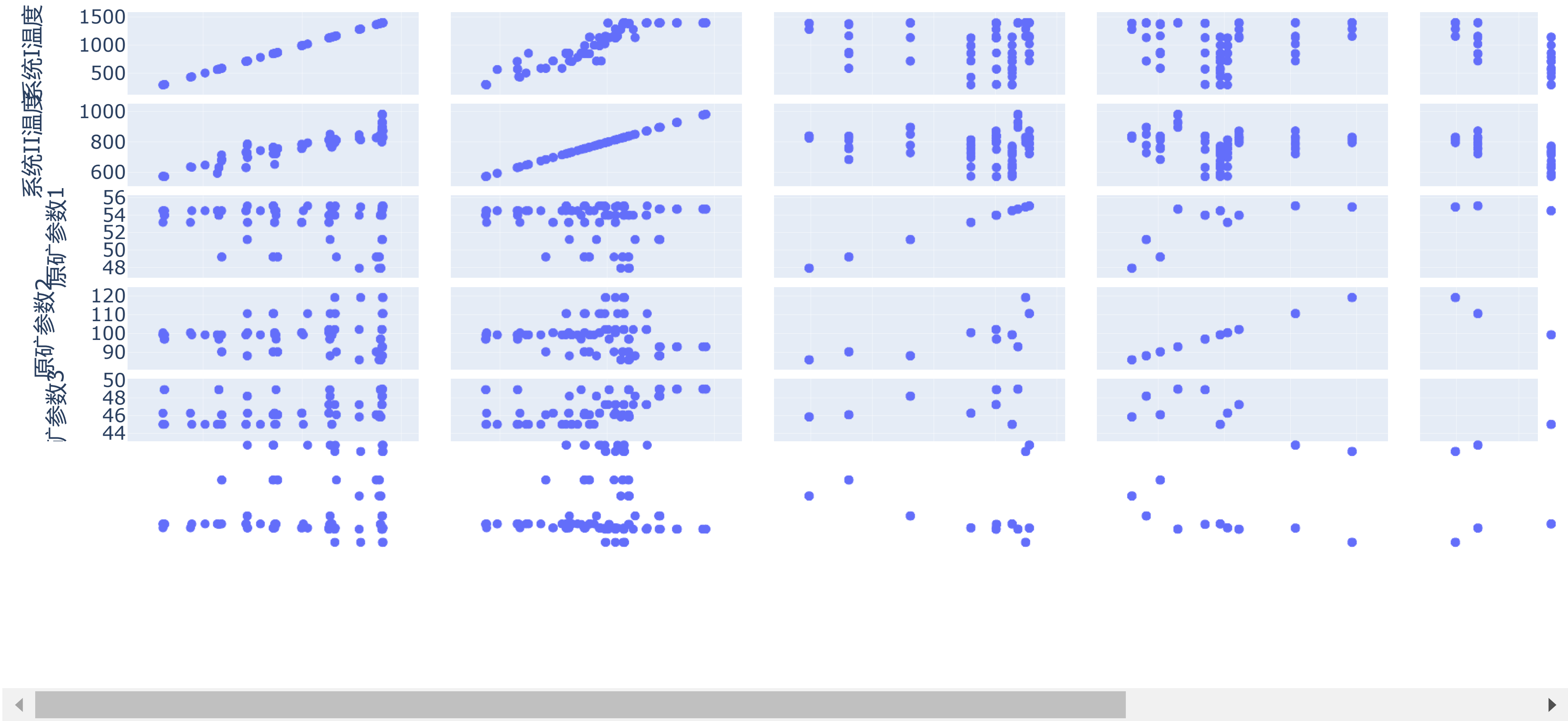
In [10]:

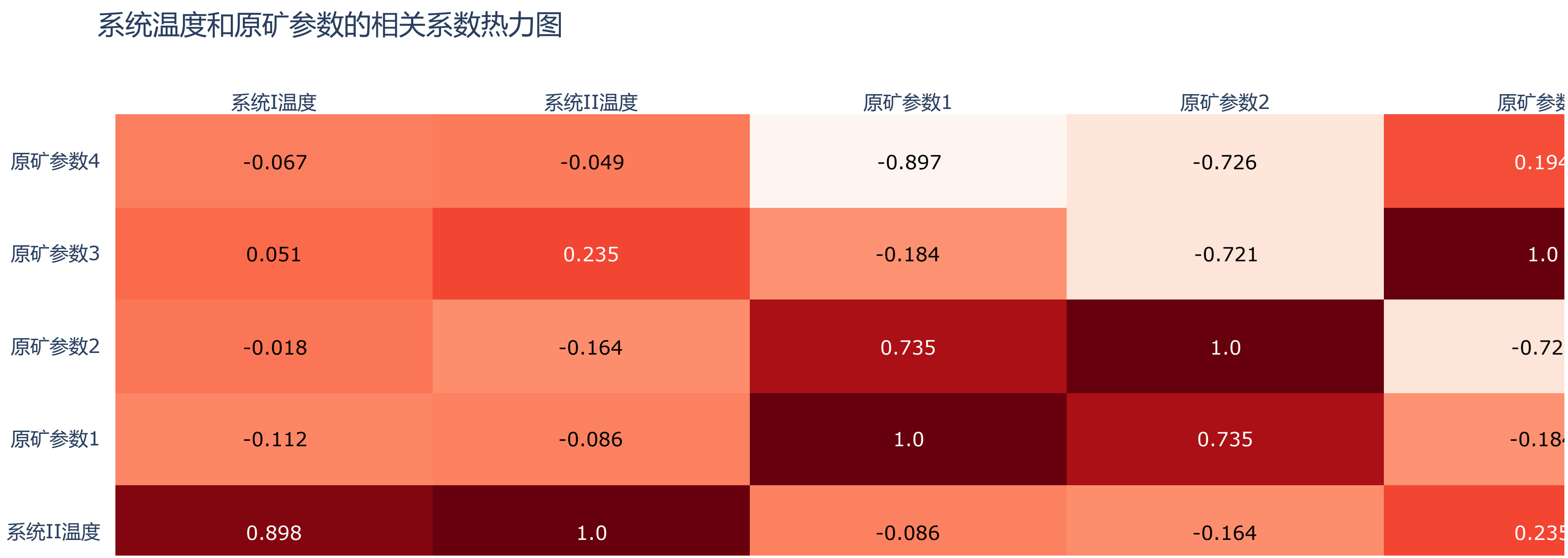
```
X = pd.concat([data_part1.iloc[:, 1:], data_part3.iloc[:, 1:]], axis=1)
Ys = data_part2.iloc[:, 1:]
```

In [11]:

```
XX = X.copy().astype(float)
XX.columns = ['系统I温度', '系统II温度', '原矿参数1', '原矿参数2', '原矿参数3', '原矿参数4']
plot_ScatterMatrix_Heatmap(XX, '系统温度和原矿参数')
```

系统温度和原矿参数的矩阵散点图





In [12]:
```
# mitosheet.sheet(X, Ys, analysis_to_replay="id-tmbibkgbvw")
```

In [ ]:
```
```

In [13]:
```
# TODO 另存为问题1的数据
X.to_csv("quention1-X_data.csv")
Ys.to_csv("quention1-Y_data.csv")
```

预测指标ABCD

分别使用不同的回归模型同时训练、预测4个指标，思路比较固定、清晰，不过多解释，直接看代码

如何判断预测的结果好坏：MSE(略)，R^2(略)

该题使用一下评测标准：

$$0.2(1 - Mape) + 0.8 * Accuracy_5$$

Ape(相对误差)：

$$Ape = \frac{|\hat{y} - y|}{y}$$

$Mape$(平均相对误差)：

$$Mape = \frac{1}{m}\sum_{i=1}^{m} Ape_i$$

$Accuracy_5$(5%准确率)：

$$Accuracy_5 = \frac{count(Ape \leq 0.05)}{count(total)}$$

In [14]:
```
index_num = Ys.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"]

data_to_predict = np.array(
    [[1404.89,859.77,52.75,96.87,46.61,22.91, ],
    [1151.75,859.77,52.75,96.87,46.61,22.91, ],],
)
```

In [15]:
```
th = 0.1  # n 准确率
con = 0.2  # 准确率的权重

def run_model(model_name, model, X=X, Ys=Ys, index_num=index_num):
    """
    :param model_name:
    :param model:
    :param X:
    :param Ys:
    :param index_num:
    :return: [yhats]
        默认不展示绘制图像（图像太多！太大！），需要可以打开该文件所在目录的 img 文件夹下，查看相关图像
    """
```

```
    data = []
    yhats = []
    print(model_name, ":\n")
    for i in range(index_num):
        Y = Ys.iloc[:, i]
        xtrain, xtest, ytrain, ytest = train_test_split(np.array(X),
                                                        np.array(Y),
                                                        test_size=0.3,
                                                        random_state=24,
                                                        shuffle=True)

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

        acc = predict_accuracy(ytest, yhat, type=1, th=th, con=con)  # todo 评价指标：回归
        print("accuracy:", acc)
        print("MSE:", MSE(yhat, ytest), "MAE:", MAE(yhat, ytest), end='')
        print("R2:", model.score(xtest, ytest))

        print("预测结果：", model.predict(data_to_predict))# todo 预测

        # todo 画图
        Yhat = model.predict(X)
        data.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=Y,
            name=index_name[i] + "-真实值",
            line=dict(color=index_colors[i * 2 + 1], width=1.5)),
        )
        data.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=Yhat,
            name=index_name[i] + "-预测值",
            line=dict(color=index_colors[i * 2], width=1.5)),
        )

        # todo 画图：点差图
        cols = str(Y.name)
        Yhat = pd.DataFrame(Yhat)
        Y.index = [i for i in range(len(Y))]
        Y_data = pd.concat([Y, Yhat], 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1-基于' +  model_name + '的' + cols + '预测模型.svg')
        Y_data.iplot(
            kind='spread',
            color=[index_colors[i * 2], index_colors[i * 2 + 1]],
            title='基于' +  model_name + '的' + cols + '预测模型',
        )
        print()
    fig = go.Figure(data=data, )

    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1-基于' + model_name + '的指标预测模型.svg')
    fig.show()
    return pd.DataFrame(yhats, index=index_name).T
```

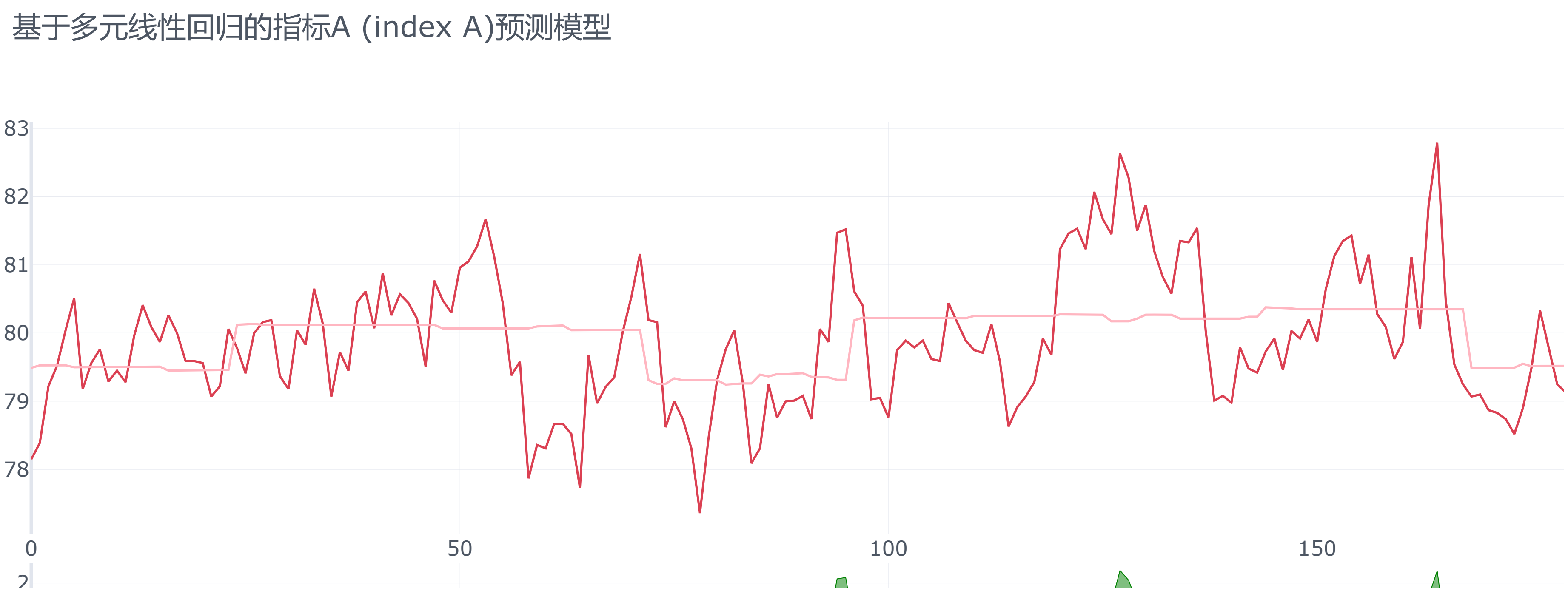
1. 使用线性回归

In [16]:

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.linear_model import LinearRegression as LR
from sklearn.preprocessing import PolynomialFeatures as PF

models_name = [
    "多元线性回归",
    "岭回归",
    "Lasso回归",
]

models_lr = [
    LR(),
    Ridge(),
    Lasso(),
]
```

```python
# 有几个线性回归模型，经测试多元线性回归最好，故使用多元线性回归，如果想使用其他线性回归模型，直接修改索引即可
for i, model in enumerate(models_lr[:1]):
    yhats = run_model(models_name[i], model, )
    plot_ScatterMatrix_Heatmap(yhats, '多元线性回归预测指标', save=True, )
```

```
多元线性回归 :

accuracy: 0.9933993277750093
MSE: 0.7214922798902208 MAE: 0.659732866837919R2: 0.15580124573991982
预测结果:  [79.93302491 80.01132231]
```



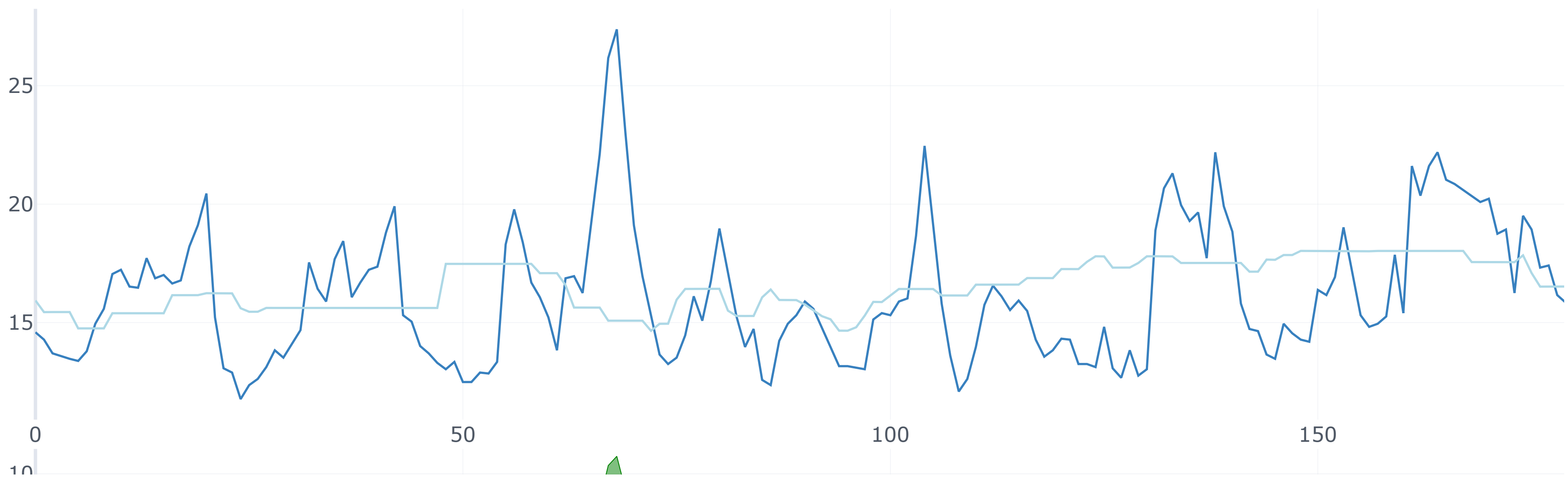

```
accuracy: 0.9601720347576297
MSE: 1.461767179606714 MAE: 0.9924382514846984R2: -0.007219919256046037
预测结果:  [23.14516619 23.21650179]
```

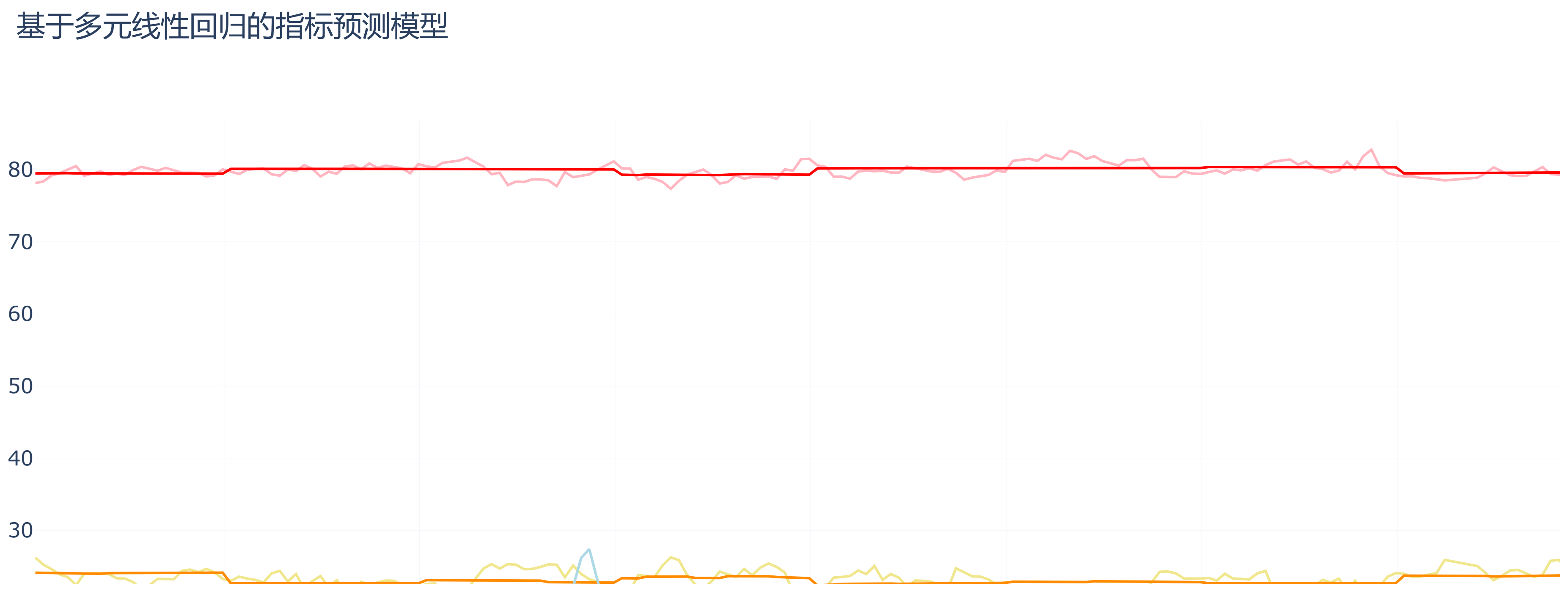
基于多元线性回归的指标B (index B)预测模型


```
accuracy: 0.9404482570195457
MSE: 0.5177133139285613 MAE: 0.5903599831961389R2: 0.3569966156331418
预测结果:  [11.44534955 11.94535264]
```

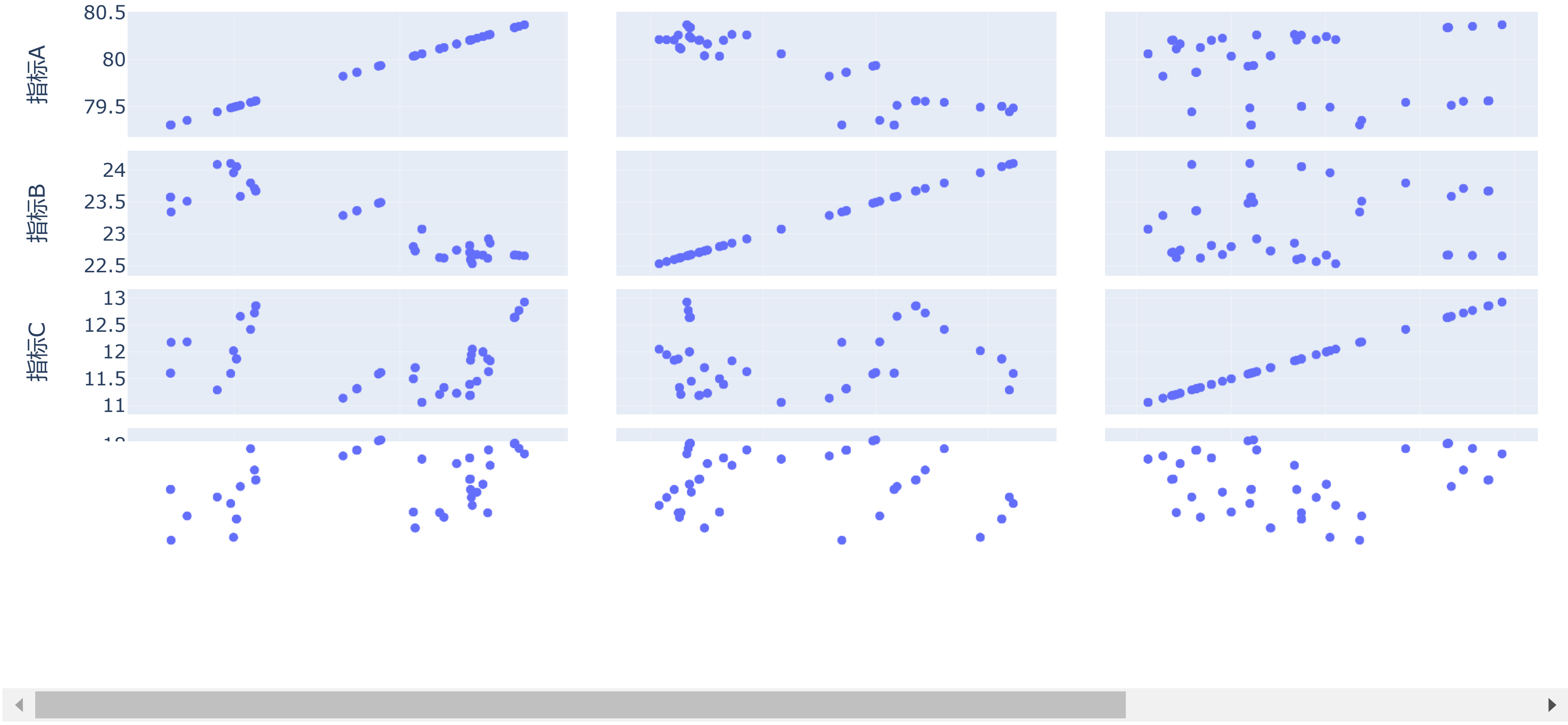
accuracy: 0.7964462701774956
MSE: 8.601332760132534 MAE: 2.1541856787566176R2: -0.18184282676575125
预测结果： [17.08368153 16.93674994]

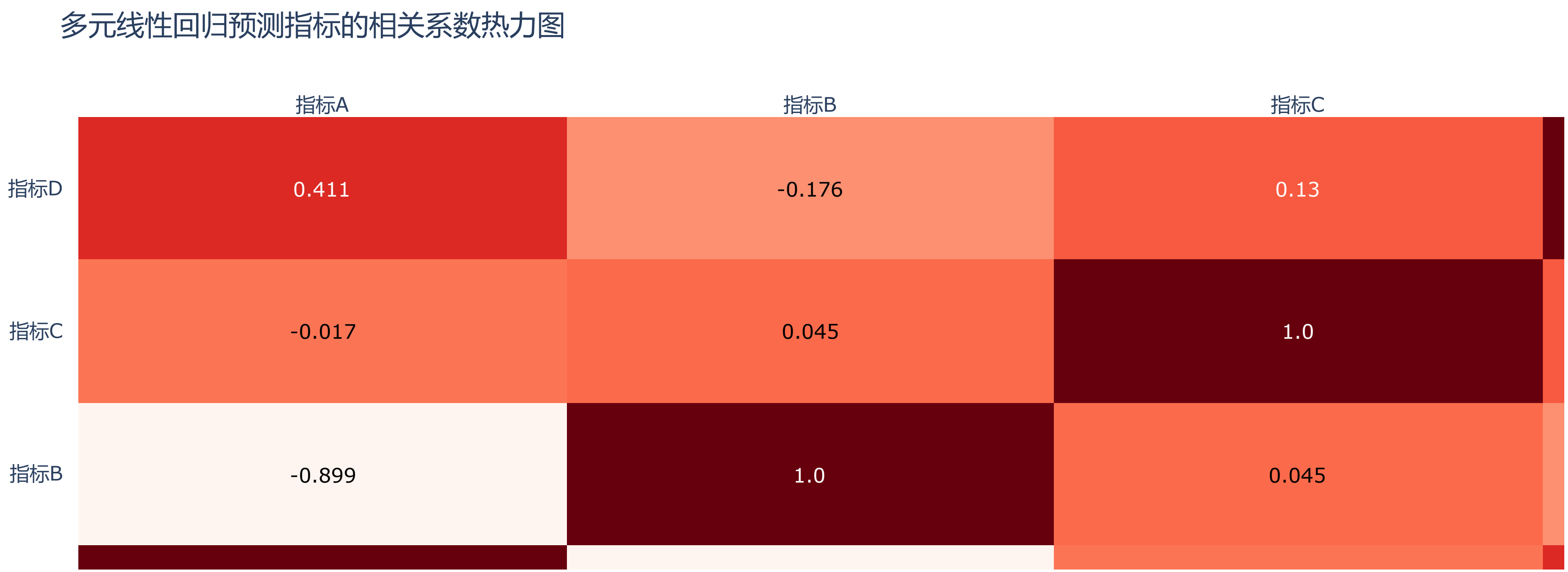
基于多元线性回归的指标D (index D)预测模型




基于多元线性回归的指标预测模型
80
70
60
50
40
30

多元线性回归预测指标的矩阵散点图





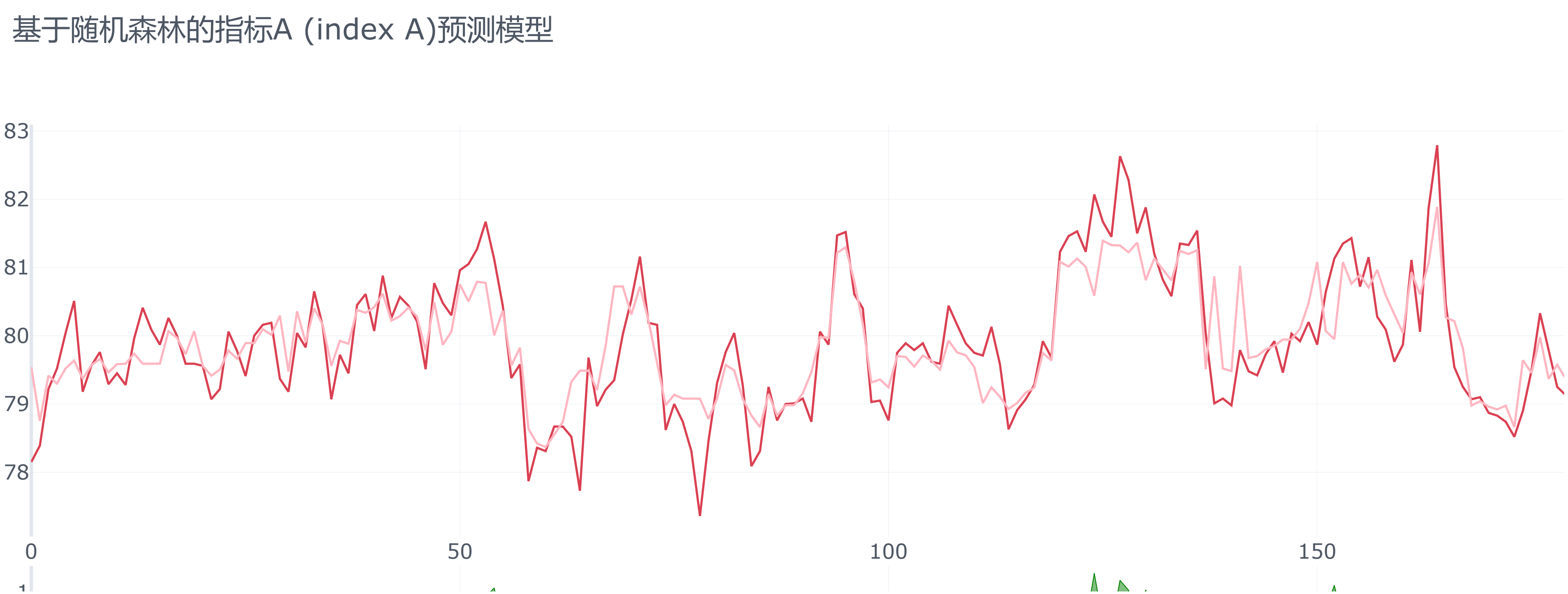
3. 使用随机森林

In [17]:

```
from sklearn.ensemble import RandomForestRegressor as RFR

models_name = ["随机森林"]
model_rf = [RFR(criterion='mae', n_estimators=100, random_state=0)]  # mse, friedman_mse, mae
for i, model in enumerate(model_rf):
    yhats = run_model(models_name[i], model, )
    plot_ScatterMatrix_Heatmap(yhats, "随机森林预测指标", save=True)
```

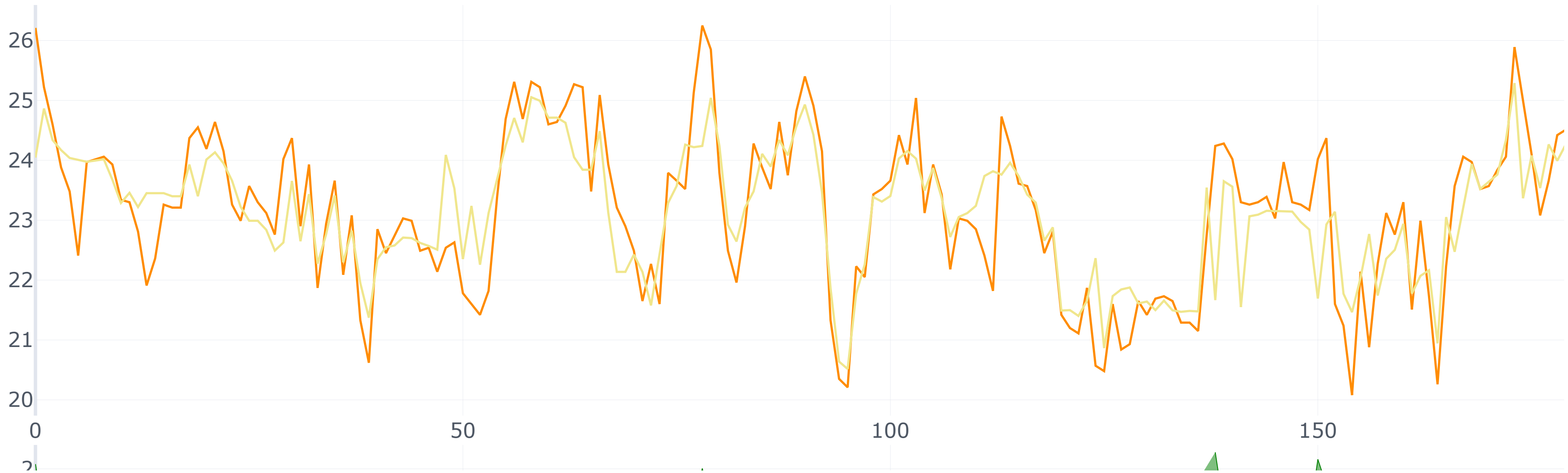
随机森林 :

accuracy: 0.9936863938529878
MSE: 0.59893996802819 MAE: 0.6300183098591678R2: 0.29919641695555765
预测结果: [80.2802 79.8394]



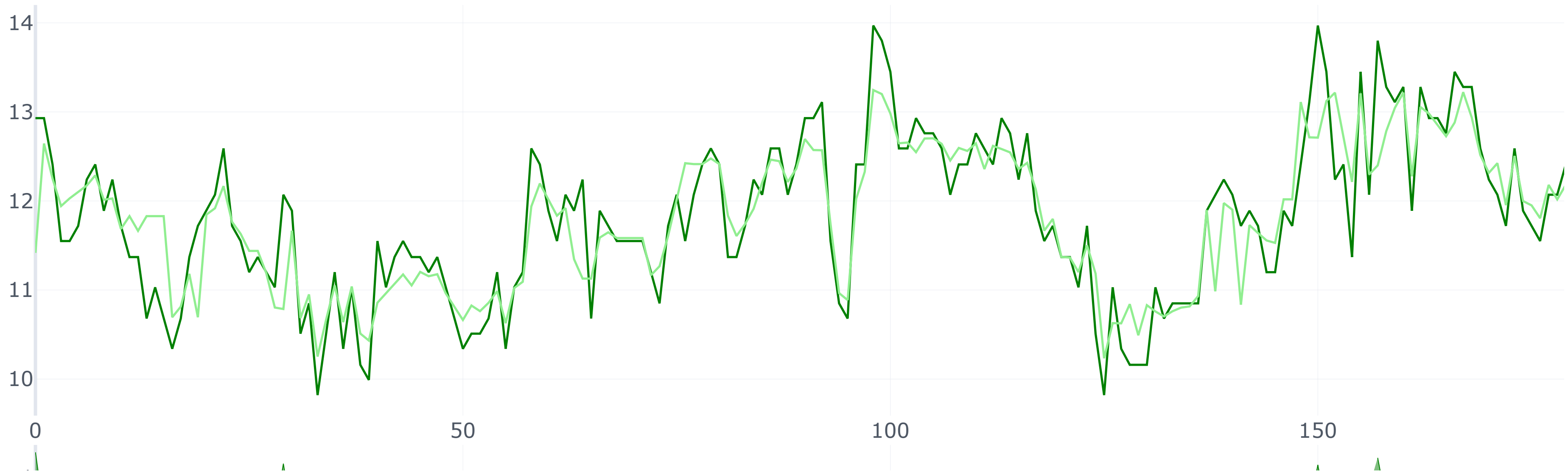
accuracy: 0.9670483312346942
MSE: 1.2237850875352152 MAE: 0.8747591549295797R2: 0.1567598901860643
预测结果: [23.0804 23.4745]

基于随机森林的指标B (index B)预测模型



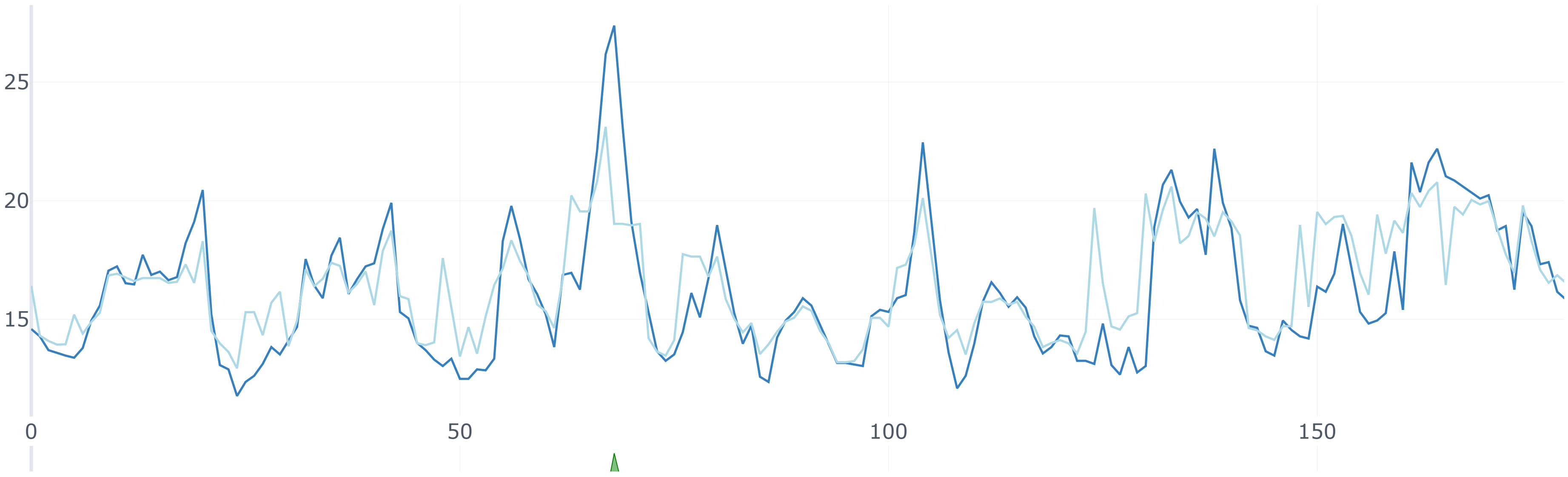
```
accuracy: 0.9522306768703401
MSE: 0.37306115862676203 MAE: 0.4569781690140852R2: 0.5366555560401716
预测结果:  [11.1525 11.8824]
```

基于随机森林的指标C (index C)预测模型

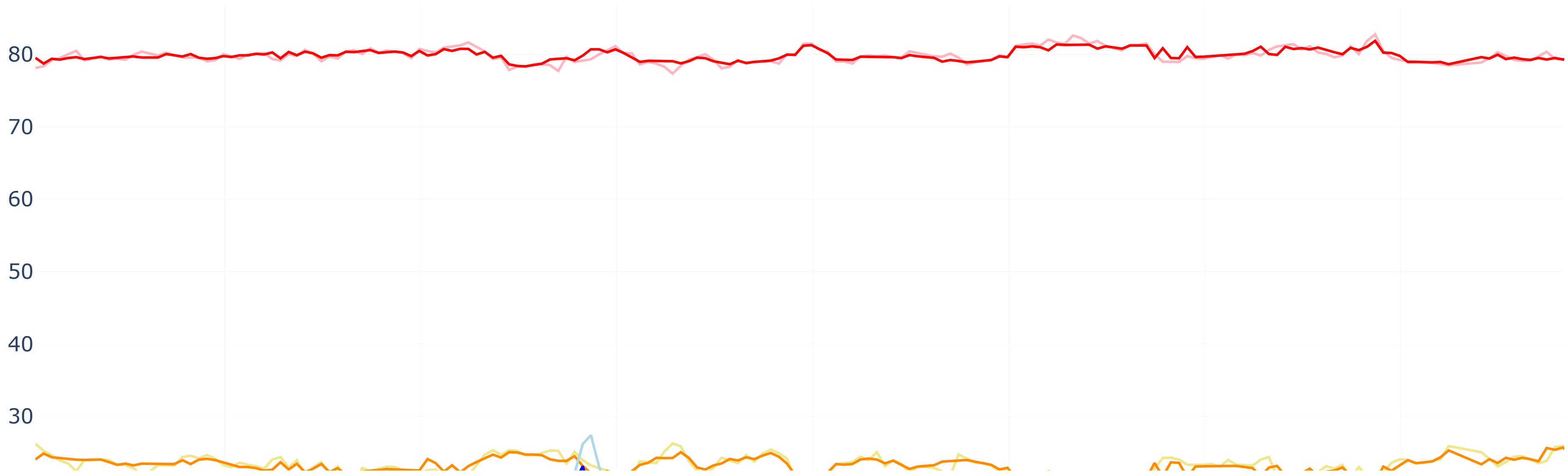


accuracy: 0.8050684225666819
MSE: 6.8053048117605455 MAE: 1.9620140845070406R2: 0.0649355280134839
预测结果： [18.7921 15.8917]

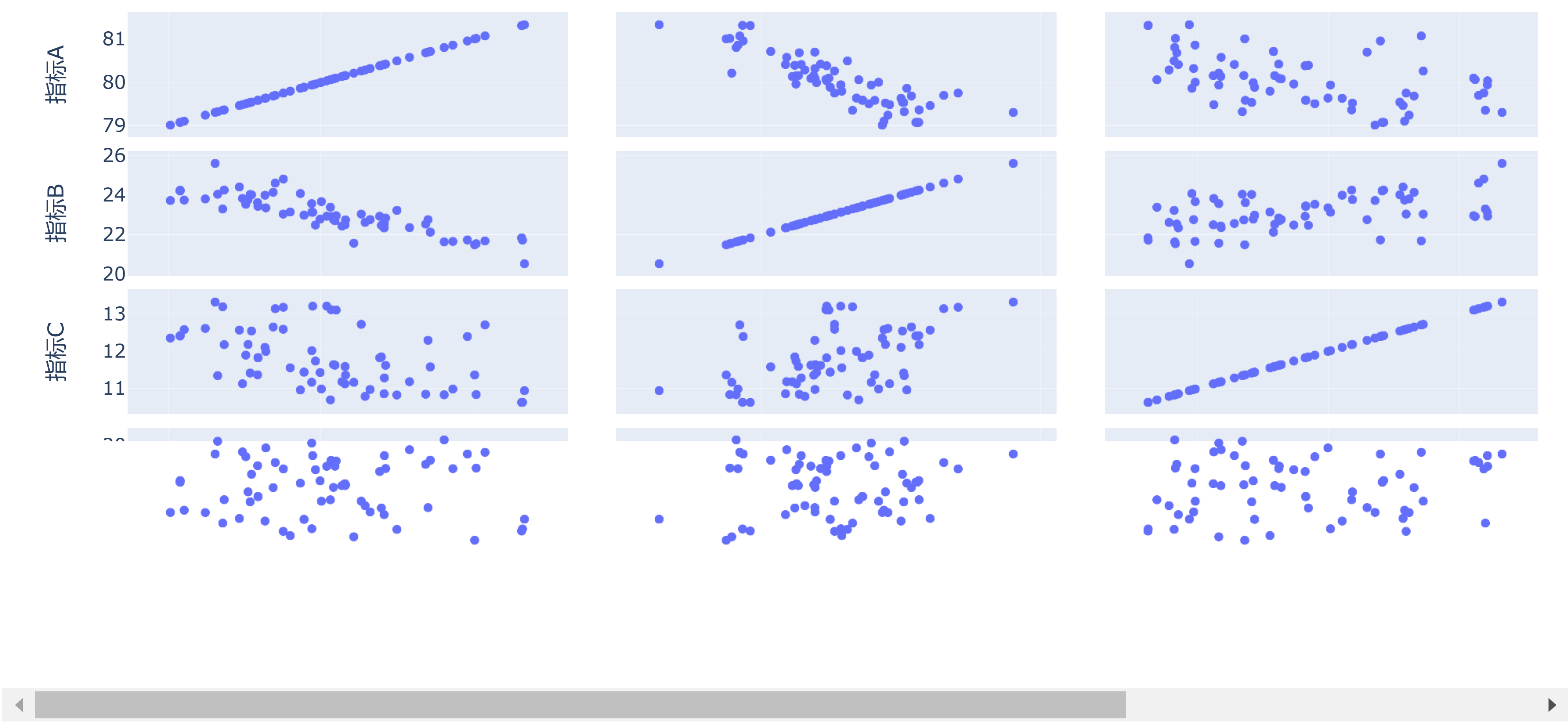
基于随机森林的指标D (index D)预测模型

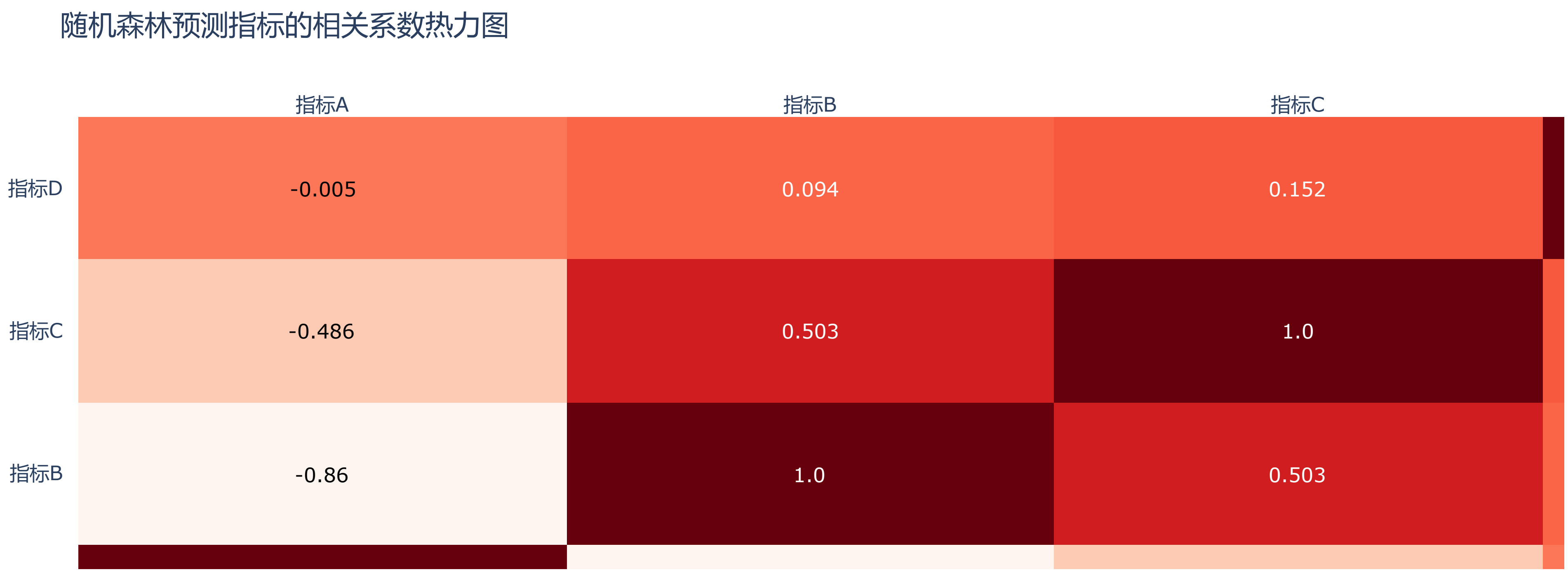


基于随机森林的指标预测模型



随机森林预测指标的矩阵散点图



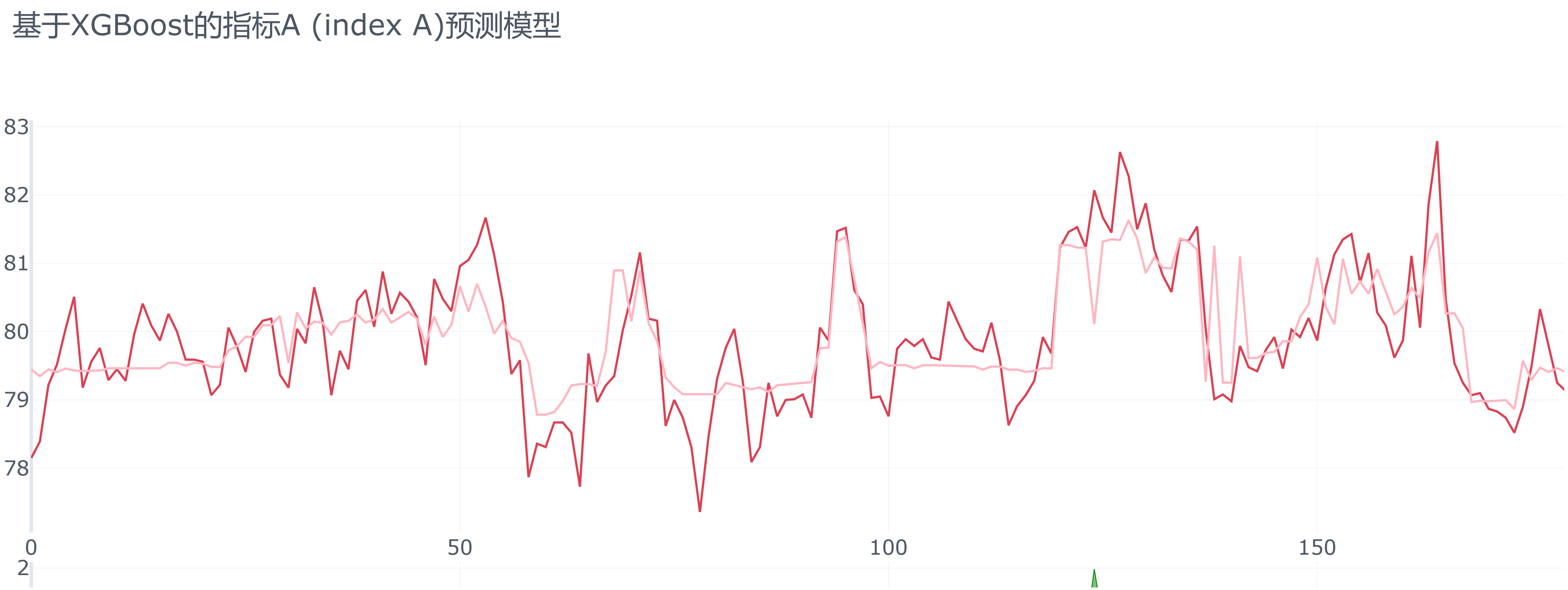


5. XGBoost

In [18]:
```python
from xgboost import XGBRegressor, XGBRFRegressor

models_xgb = [
    XGBRegressor(n_estimators=100, random_state=0),
    XGBRFRegressor(n_estimators=100, random_state=0),
]

for model in models_xgb[1:]:
    yhats = run_model("XGBoost", model, )
    plot_ScatterMatrix_Heatmap(yhats, "XGBoost预测指标", save=True)
```

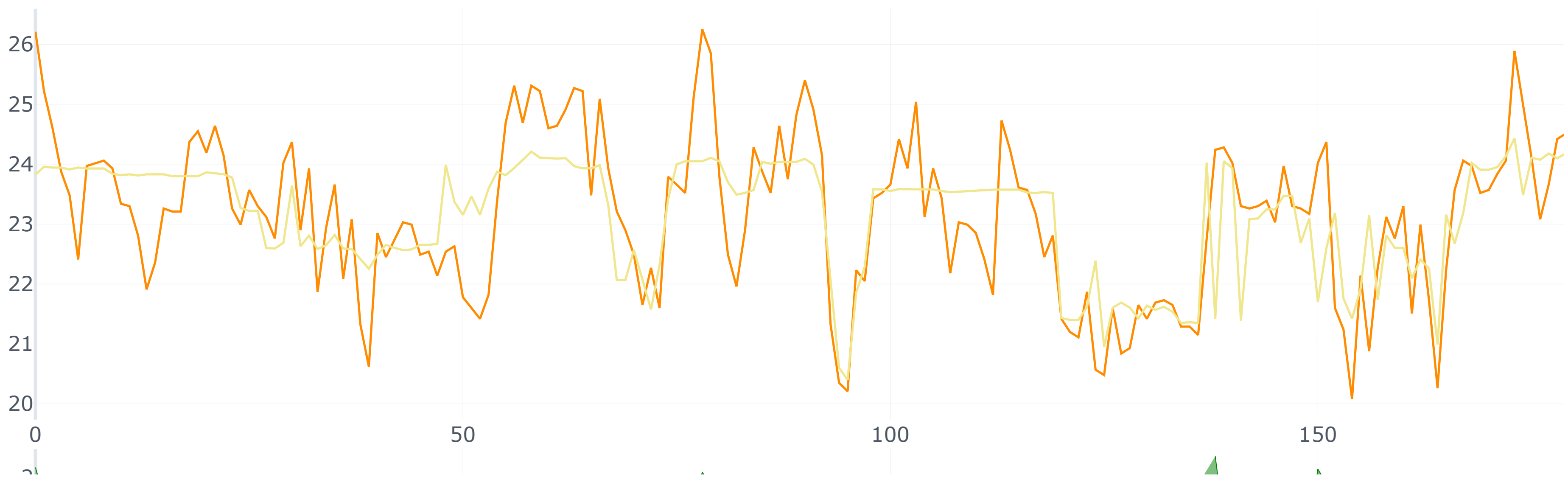
XGBoost :

accuracy: 0.9935959076028595
MSE: 0.6472435946370513 MAE: 0.6394866986341877R2: 0.24267763976829615
预测结果： [80.081894 79.78574]



accuracy: 0.962710125391463
MSE: 1.3158893486253935 MAE: 0.9193643843959755R2: 0.09329612679568178
预测结果： [23.201012 23.594017]

基于XGBoost的指标B (index B)预测模型

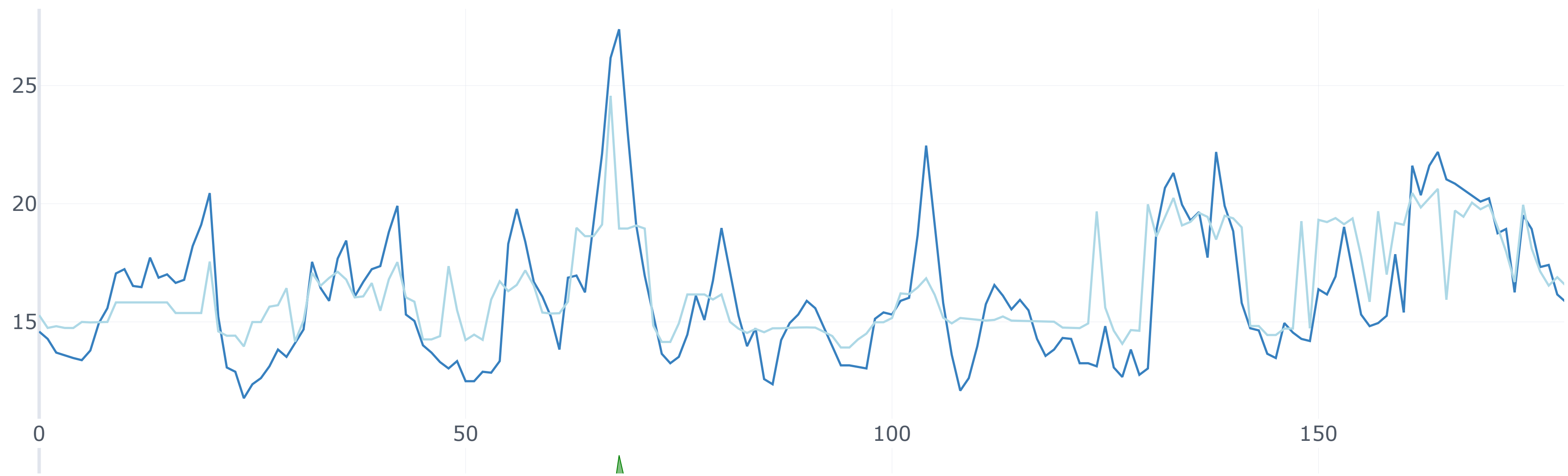



```
accuracy: 0.9535673409840772
MSE: 0.40311010268843317 MAE: 0.47935484496640496R2: 0.49933456736076687
预测结果:  [11.328869 11.941339]
```

基于XGBoost的指标C (index C)预测模型

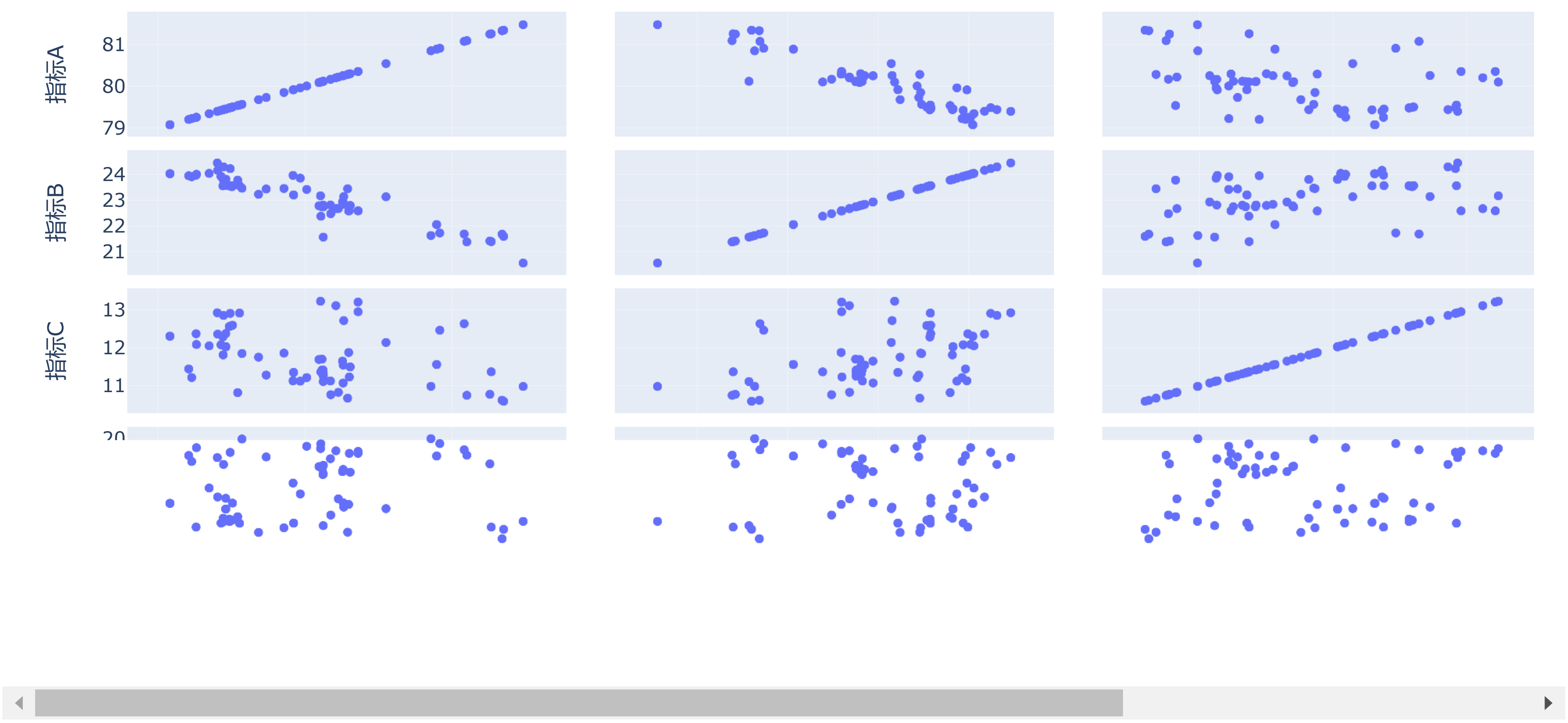
accuracy: 0.806067175807232
MSE: 6.76325801518573 MAE: 1.9539357564147088R2: 0.0707128541914398
预测结果： [18.555197 15.974333]

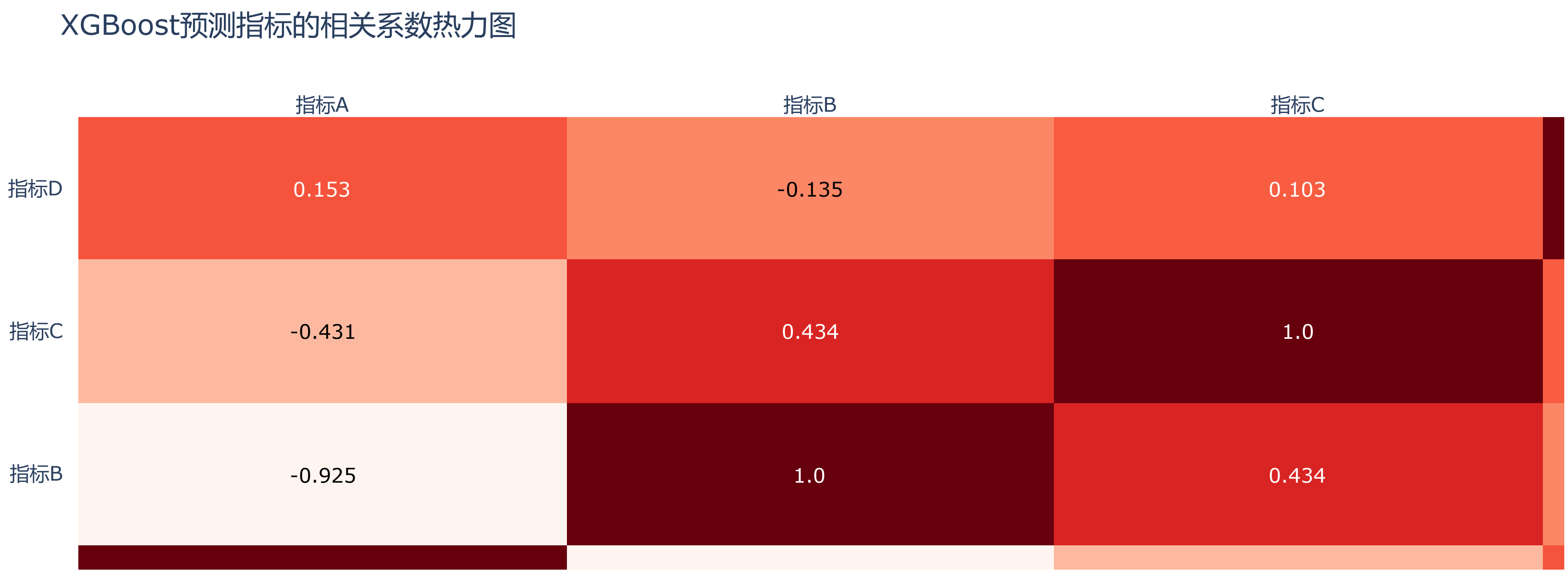
基于XGBoost的指标D (index D)预测模型



基于XGBoost的指标预测模型

XGBoost预测指标的矩阵散点图





8行，废废了

在以下代码中，BP效果不好，AutoGluon太占内存，时间太长，不适合在该题中使用！也**强烈不建议运行！**最终论文也没有写

4. BP 神经网络

要运行该代码必须安装 hmz 这个库

In [19]:
```
hidden_num = [8, 16, 32, 64, 128, 64, 32, 16, 8, 4]
epoch = 1000
optimizer = 'adam'
```

```python
def run_BP(X=X, Ys=Ys, index_num=index_num):
    data_to_predict = np.array(
        [[1404.89, 859.77, 52.75, 96.87, 46.61, 22.91, ],
         [1151.75, 859.77, 52.75, 96.87, 46.61, 22.91, ],
        ],
    )
#     data_to_predict = np.array(
#         [[1404.89,859.77,52.75,96.87,46.61,22.91, ],
#          [1151.75,859.77,52.75,96.87,46.61,22.91, ],
#          [1173.63, 813.92, 49.24, 90.38, 46.13, 28.16, ],
#          [854.55, 767.64, 49.24, 90.38, 46.13, 28.16, ],
#         ],
#     )  # 测试样例，如果上面运行结果一样的话，则使用该样例，只看前两个预测结果即可（pytorch的一些奇奇怪怪的bug？）
    data = []
    print("BP神经网络：")
    for i in range(index_num):
        Y = Ys.iloc[:, i]
        xtrain, xtest, ytrain, ytest = train_test_split(
            np.array(X, dtype=float), np.array(Y, dtype=float),
            test_size=0.3,
            random_state=10,
            shuffle=True,
        )
        bp = BP(
            X.shape[1], hidden_num, 1,
            epoch=epoch,
            optimizer=optimizer,
            normalization=True
        )
        bp.train(xtrain, ytrain)
        y_pre = bp.predict(xtest).cpu().detach().numpy()
        acc = predict_accuracy(ytest[:, None], y_pre, type=1)  # 回归
        print("accuracy:", acc)
        print("预测结果：", bp.predict(data_to_predict))
#         print(mean_squared_error(y_true=ytest[:, None], y_pred=y_pre))
#         print("MSE:", MSE(yhat, ytest), "MAE:", MAE(yhat, ytest), end='')
#         print("R2:", model.score(xtest, ytest))

        # todo 画图
        Yhat = bp.predict(np.array(X, dtype=float)).cpu().detach().numpy()
        data.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=Y,
            name=index_name[i] + "-真实值",
            line=dict(color=index_colors[i * 2 + 1], width=1.5)),
        )
```

```
        data.append(go.Scatter(
            x=data_part1.iloc[:, 0], y=np.squeeze(Yhat),
            name=index_name[i] + "-预测值",
            line=dict(color=index_colors[i * 2], width=1.5)),
        )

        # todo 画图：点差图
        cols = str(Y.name)
        Yhat = pd.DataFrame(Yhat)
        Y.index = [i for i in range(len(Y))]
        Y_data = pd.concat([Y, Yhat], 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BP神经网络的指标预测模型——' + cols,
        ).write_image('./img/问题1-基于BP神经网络的' + cols + '预测模型.svg')
        Y_data.iplot(
            kind='spread',
            color=[index_colors[i * 2 + 1], index_colors[i * 2]],
            title='基于BP神经网络的指标预测模型——' + cols,
        )
    fig = go.Figure(data=data)

    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BP神经网络的指标预测模型',
        annotations=annotations,
    )
    fig.write_image('./img/问题1-基于BP神经网络的指标预测模型.svg')
    fig.show()
    return None
```

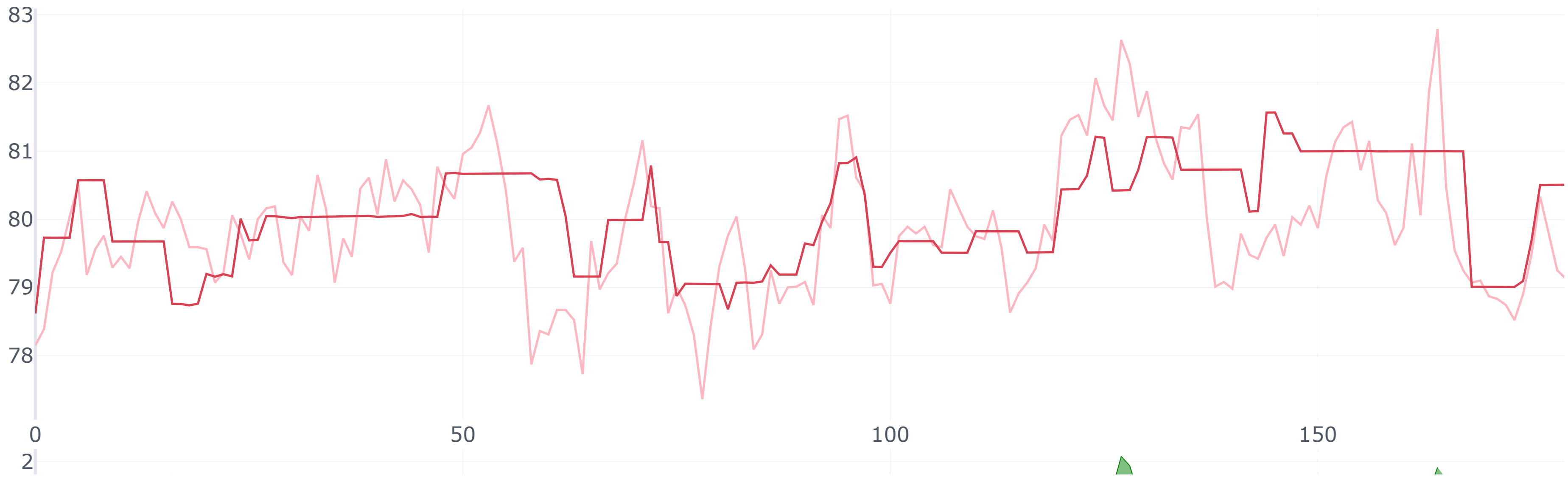
In [20]:
```
run_BP()
```

BP神经网络：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 8]              56
            Linear-2                   [64, 8]              56
              ReLU-3                   [64, 8]               0
              ReLU-4                   [64, 8]               0
            Linear-5                  [64, 16]             144
              ReLU-6                  [64, 16]               0
              ReLU-7                  [64, 16]               0
            Linear-8                  [64, 32]             544
              ReLU-9                  [64, 32]               0
             ReLU-10                  [64, 32]               0
           Linear-11                  [64, 64]           2,112
             ReLU-12                  [64, 64]               0
             ReLU-13                  [64, 64]               0
           Linear-14                 [64, 128]           8,320
             ReLU-15                 [64, 128]               0
             ReLU-16                 [64, 128]               0
           Linear-17                  [64, 64]           8,256
             ReLU-18                  [64, 64]               0
             ReLU-19                  [64, 64]               0
           Linear-20                  [64, 32]           2,080
             ReLU-21                  [64, 32]               0
             ReLU-22                  [64, 32]               0
           Linear-23                  [64, 16]             528
             ReLU-24                  [64, 16]               0
             ReLU-25                  [64, 16]               0
           Linear-26                   [64, 8]             136
             ReLU-27                   [64, 8]               0
             ReLU-28                   [64, 8]               0
           Linear-29                   [64, 4]              36
             ReLU-30                   [64, 4]               0
             ReLU-31                   [64, 4]               0
           Linear-32                   [64, 1]               5
           Linear-33                   [64, 1]               5
================================================================
Total params: 22,278
Trainable params: 22,278
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.55
Params size (MB): 0.08
```

```
Estimated Total Size (MB): 0.64
----------------------------------------------------------------
epoch: 999, train loss: 1.02, eval loss: 0.8: 100%|██████████████████████████████████| 1000/1000 [00:18<00:00, 54.28it/s]
accuracy: 0.9979631841916573
预测结果:  tensor([[nan],
        [nan]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

基于BP神经网络的指标预测模型——指标A (index A)



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 8]              56
            Linear-2                   [64, 8]              56
              ReLU-3                   [64, 8]               0
              ReLU-4                   [64, 8]               0
            Linear-5                  [64, 16]             144
              ReLU-6                  [64, 16]               0
              ReLU-7                  [64, 16]               0
            Linear-8                  [64, 32]             544
              ReLU-9                  [64, 32]               0
             ReLU-10                  [64, 32]               0
           Linear-11                  [64, 64]           2,112
             ReLU-12                  [64, 64]               0
             ReLU-13                  [64, 64]               0
           Linear-14                 [64, 128]           8,320
             ReLU-15                 [64, 128]               0
             ReLU-16                 [64, 128]               0
           Linear-17                  [64, 64]           8,256
             ReLU-18                  [64, 64]               0
             ReLU-19                  [64, 64]               0
           Linear-20                  [64, 32]           2,080
             ReLU-21                  [64, 32]               0
             ReLU-22                  [64, 32]               0
           Linear-23                  [64, 16]             528
             ReLU-24                  [64, 16]               0
             ReLU-25                  [64, 16]               0
           Linear-26                   [64, 8]             136
             ReLU-27                   [64, 8]               0
             ReLU-28                   [64, 8]               0
           Linear-29                   [64, 4]              36
             ReLU-30                   [64, 4]               0
             ReLU-31                   [64, 4]               0
           Linear-32                   [64, 1]               5
           Linear-33                   [64, 1]               5
================================================================
Total params: 22,278
Trainable params: 22,278
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.55
Params size (MB): 0.08
Estimated Total Size (MB): 0.64
----------------------------------------------------------------
```

```
epoch: 999, train loss: 1.77, eval loss: 1.02: 100%|██████████| 1000/1000 [00:17<00:00, 56.06it/s]
accuracy: 0.7446468846079601
预测结果:  tensor([[nan],
        [nan]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

基于BP神经网络的指标预测模型——指标B (index B)

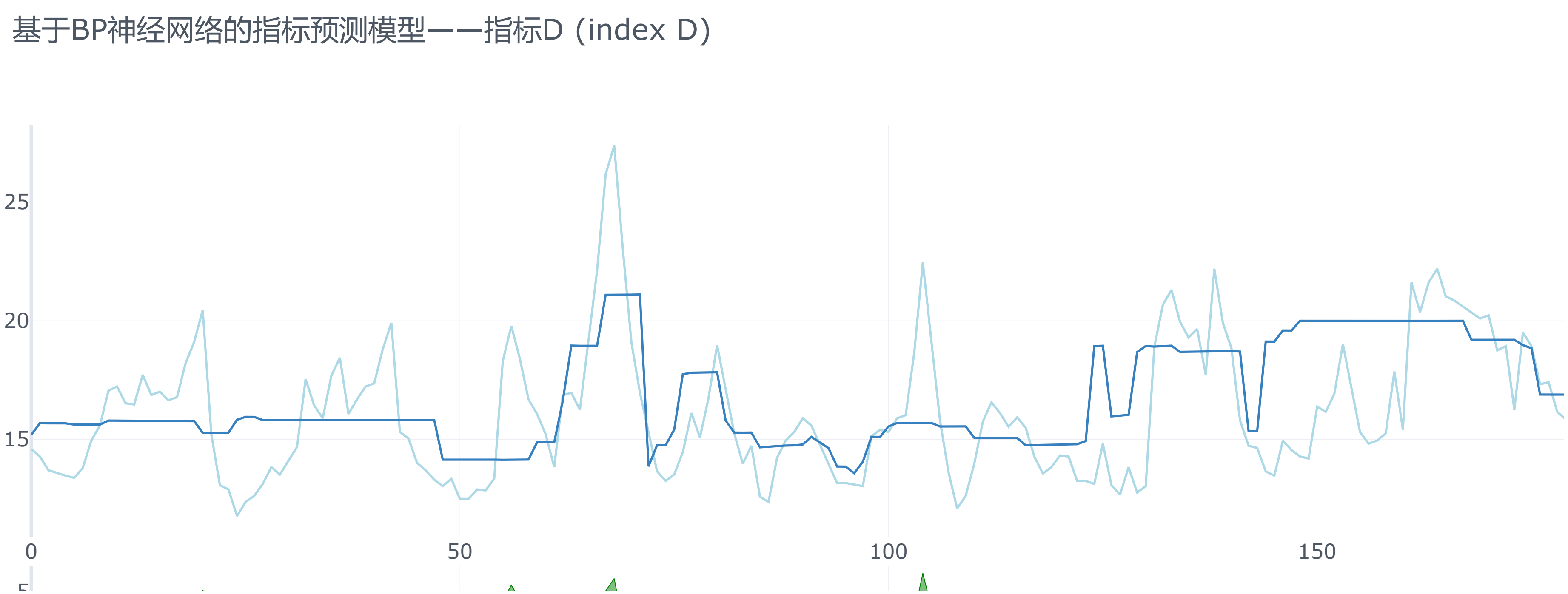
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 8]              56
            Linear-2                   [64, 8]              56
              ReLU-3                   [64, 8]               0
              ReLU-4                   [64, 8]               0
            Linear-5                  [64, 16]             144
              ReLU-6                  [64, 16]               0
              ReLU-7                  [64, 16]               0
            Linear-8                  [64, 32]             544
              ReLU-9                  [64, 32]               0
             ReLU-10                  [64, 32]               0
           Linear-11                  [64, 64]           2,112
             ReLU-12                  [64, 64]               0
             ReLU-13                  [64, 64]               0
           Linear-14                 [64, 128]           8,320
             ReLU-15                 [64, 128]               0
             ReLU-16                 [64, 128]               0
           Linear-17                  [64, 64]           8,256
             ReLU-18                  [64, 64]               0
             ReLU-19                  [64, 64]               0
           Linear-20                  [64, 32]           2,080
             ReLU-21                  [64, 32]               0
             ReLU-22                  [64, 32]               0
           Linear-23                  [64, 16]             528
             ReLU-24                  [64, 16]               0
             ReLU-25                  [64, 16]               0
           Linear-26                   [64, 8]             136
             ReLU-27                   [64, 8]               0
             ReLU-28                   [64, 8]               0
           Linear-29                   [64, 4]              36
             ReLU-30                   [64, 4]               0
             ReLU-31                   [64, 4]               0
           Linear-32                   [64, 1]               5
           Linear-33                   [64, 1]               5
================================================================
Total params: 22,278
Trainable params: 22,278
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.55
Params size (MB): 0.08
Estimated Total Size (MB): 0.64
----------------------------------------------------------------
```

```
epoch: 999, train loss: 0.8, eval loss: 0.35: 100%|██████████| 1000/1000 [00:17<00:00, 57.25it/s]
accuracy: 0.7549239537369776
预测结果：  tensor([[nan],
        [nan]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

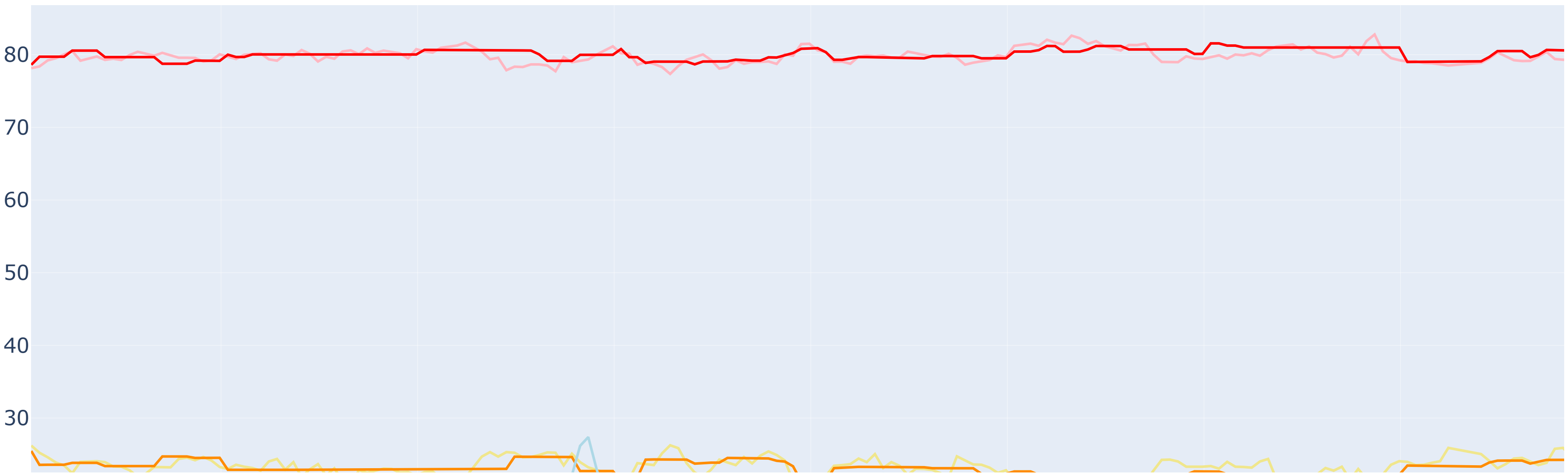
基于BP神经网络的指标预测模型——指标C (index C)

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [64, 8]              56
            Linear-2                   [64, 8]              56
              ReLU-3                   [64, 8]               0
              ReLU-4                   [64, 8]               0
            Linear-5                  [64, 16]             144
              ReLU-6                  [64, 16]               0
              ReLU-7                  [64, 16]               0
            Linear-8                  [64, 32]             544
              ReLU-9                  [64, 32]               0
             ReLU-10                  [64, 32]               0
           Linear-11                  [64, 64]           2,112
             ReLU-12                  [64, 64]               0
             ReLU-13                  [64, 64]               0
           Linear-14                 [64, 128]           8,320
             ReLU-15                 [64, 128]               0
             ReLU-16                 [64, 128]               0
           Linear-17                  [64, 64]           8,256
             ReLU-18                  [64, 64]               0
             ReLU-19                  [64, 64]               0
           Linear-20                  [64, 32]           2,080
             ReLU-21                  [64, 32]               0
             ReLU-22                  [64, 32]               0
           Linear-23                  [64, 16]             528
             ReLU-24                  [64, 16]               0
             ReLU-25                  [64, 16]               0
           Linear-26                   [64, 8]             136
             ReLU-27                   [64, 8]               0
             ReLU-28                   [64, 8]               0
           Linear-29                   [64, 4]              36
             ReLU-30                   [64, 4]               0
             ReLU-31                   [64, 4]               0
           Linear-32                   [64, 1]               5
           Linear-33                   [64, 1]               5
================================================================
Total params: 22,278
Trainable params: 22,278
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.55
Params size (MB): 0.08
Estimated Total Size (MB): 0.64
----------------------------------------------------------------
```

```
epoch: 999, train loss: 10.49, eval loss: 4.84: 100%|██████████████████████████████| 1000/1000 [00:17<00:00, 58.64it/s]
accuracy: 0.3764500507401932
预测结果：  tensor([[nan],
        [nan]], device='cuda:0', grad_fn=<AddmmBackward0>)
```



基于BP神经网络的指标预测模型



6. AutoGluon

要运行该代码必须安装 autogluon 这个库（还有 pytorch 等依赖）

强烈不建议运行！ 实在想运行的话，全选、取消注释，然后运行即可

In [21]:
```
# import autogluon
# from autogluon.tabular import TabularDataset, TabularPredictor
```

In [22]:
```
# cols = list(X.columns) + list(Ys.columns)
# test_data = pd.DataFrame(
#     np.concatenate((data_to_predict, np.array([[0,0,0,0], [0,0,0,0]])), axis=1),
```

```
#     columns=cols,
# )  # test_data

# for i in range(index_num):
#     print(index_name[i])
#     Y = Ys.iloc[:, i]
#     xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3, random_state=10, shuffle=True)

#     l = range(len(xtrain))
#     xtrain = pd.DataFrame(xtrain, index=l)
#     ytrain = pd.DataFrame(ytrain, index=l)
#     train_data = pd.concat([xtrain, ytrain], axis=1)

#     predictor = TabularPredictor(label=pd.DataFrame(ytrain).columns[0]).fit(
#         train_data,
#         auto_stack=True,
#         verbosity=2,
#     )
#     leaderboard = predictor.leaderboard(test_data)
#     results = predictor.fit_summary() # display detailed summary of fit() process
#     print(pd.DataFrame(leaderboard))
#     y_pred = predictor.predict(test_data)
#     print("Predictions:  \n", y_pred)

#     acc = predict_accuracy(ytest, y_pred, type=1)  # 回归
#     print("accuracy:", acc)
#     print(mean_squared_error(y_true=ytest, y_pred=y_pred))

#     perf = predictor.evaluate_predictions(y_true=y_test, y_pred=y_pred, auxiliary_metrics=True)

#     # todo 预测
# #     print("预测结果: ", model.predict(data_to_predict))
```

In [ ]: