华数杯C题

所有问题代码，直接全部运行即可

In [1]:

```
# TODO import
import os
import sys
import hmz
import scipy
# import mitosheet
import numpy as np
import pandas as pd
from copy import copy

# import sympy
# from sympy import limit
# from sympy import diff
# from sympy import integrals

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 r2_score
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import classification_report, roc_auc_score

import plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
plotly.offline.init_notebook_mode()

# import chart_studio
# import chart_studio.plotly as py
# from chart_studio.plotly import plot, iplot
# chart_studio.tools.set_credentials_file(username='zhiliao0824', api_key='qrGJtJRojwpsVJ3nosn0')

import cufflinks as cf
cf.set_config_file(
```

```
    offline=True,
    world_readable=True,
    theme='white',  # cf.getThemes()
)

import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['KaiTi']  # SimHei KaiTi
plt.rcParams['axes.unicode_minus'] = False

import cv2 as cv

# import torch
# import torchvision
# import torch.nn as nn
# import torch.nn.functional as F
# import torch.utils.data as Data
# from torch.utils.data import DataLoader
# from torch.utils.data.dataset import Dataset

# import pylatex
# import latexify

import warnings
warnings.filterwarnings("ignore")

from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'
```

问题1

处理数据

In [2]:
```
def concert_data(
    data,
    orginal_type=["int64", "float64"],
    convert_type=["int32", "float32"],
    category_proportion=0.5,
):
    """转换数据的数据类型，减少其内存空间
    :param data: 待转换数据类型的数据
```

```python
    """
    # number
    for type_i in range(len(orginal_type)):
        data_converted = data.select_dtypes(
            include=[orginal_type[type_i]],
        ).astype(convert_type[type_i])
        data[data_converted.columns] = data_converted
    # string
    data_string = data.select_dtypes(include=["object"]).copy()
    for column in data_string.columns:
        unique_num = len(data_string[column].unique())
        total_num = len(data_string[column])
        if unique_num / total_num < category_proportion:
            data.loc[:, column] = data_string[column].astype('category')
    return data
```

In [3]:

```python
InteractiveShell.ast_node_interactivity = 'all'
sheet_name = ["datat1", "data2", "data3"]

data1 = pd.read_excel('C题数据.xlsx', sheet_name=sheet_name[0])[:-1]
data1_uninternotated = copy(data1[0::2])  # 未插层--插层前
data1_uninternotated.fillna(0, inplace=True)  # NaN
data1_uninternotated.reset_index(drop=True, inplace=True)  # index
data1_uninternotated = concert_data(data1_uninternotated)  # str:object -> str:category, 64 -> 32
data1_uninternotated
data1_uninternotated.info()
data1_uninternotated_unnumbered = copy(data1_uninternotated.drop(columns=['编号']))
data1_uninternotated_unnumbered
data1_uninternotated_unnumbered.info()
data1_interpolated = copy(data1[1::2])  # 插层--插层后
data1_interpolated.reset_index(drop=True, inplace=True)  # index
data1_interpolated = concert_data(data1_interpolated)  # str:object -> str:category, 64 -> 32
data1_interpolated.iloc[:, 0] = data1_uninternotated.iloc[:, 0]  # 组号
data1_interpolated
data1_interpolated.info()
data1_interpolated_unnumbered = copy(data1_interpolated.drop(columns=['编号']))
data1_interpolated_unnumbered
data1_interpolated_unnumbered.info()

data2 = pd.read_excel('C题数据.xlsx', sheet_name=sheet_name[1], index_col=0, header=0)[3:7]
data2.dropna(axis=1, inplace=True)  # NaN
data2 = concert_data(data2)  # 64 -> 32
data2  # index:接收距离(cm), columns:热风速度(r/min)
data2.info()

data3 = pd.read_excel('C题数据.xlsx', sheet_name=sheet_name[2])
```

```
data3 = concert_data(data3)  # 64 -> 32
data3
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   组号         25 non-null     float32
 1   编号         25 non-null     category
 2   厚度mm       25 non-null     float32
 3   孔隙率（%）      25 non-null     float32
 4   压缩回弹性率（%）   25 non-null     float32
 5   过滤阻力Pa     25 non-null     float32
 6   过滤效率（%）     25 non-null     float32
 7   透气性 mm/s   25 non-null     float32
 8   插层率（%）      25 non-null     float32
dtypes: category(1), float32(8)
memory usage: 1.0 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   组号         25 non-null     float32
 1   厚度mm       25 non-null     float32
 2   孔隙率（%）      25 non-null     float32
 3   压缩回弹性率（%）   25 non-null     float32
 4   过滤阻力Pa     25 non-null     float32
 5   过滤效率（%）     25 non-null     float32
 6   透气性 mm/s   25 non-null     float32
 7   插层率（%）      25 non-null     float32
dtypes: float32(8)
memory usage: 928.0 bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   组号         25 non-null     float32
 1   编号         25 non-null     category
 2   厚度mm       25 non-null     float32
 3   孔隙率（%）      25 non-null     float32
 4   压缩回弹性率（%）   25 non-null     float32
 5   过滤阻力Pa     25 non-null     float32
 6   过滤效率（%）     25 non-null     float32
 7   透气性 mm/s   25 non-null     float32
 8   插层率（%）      25 non-null     float32
dtypes: category(1), float32(8)
```

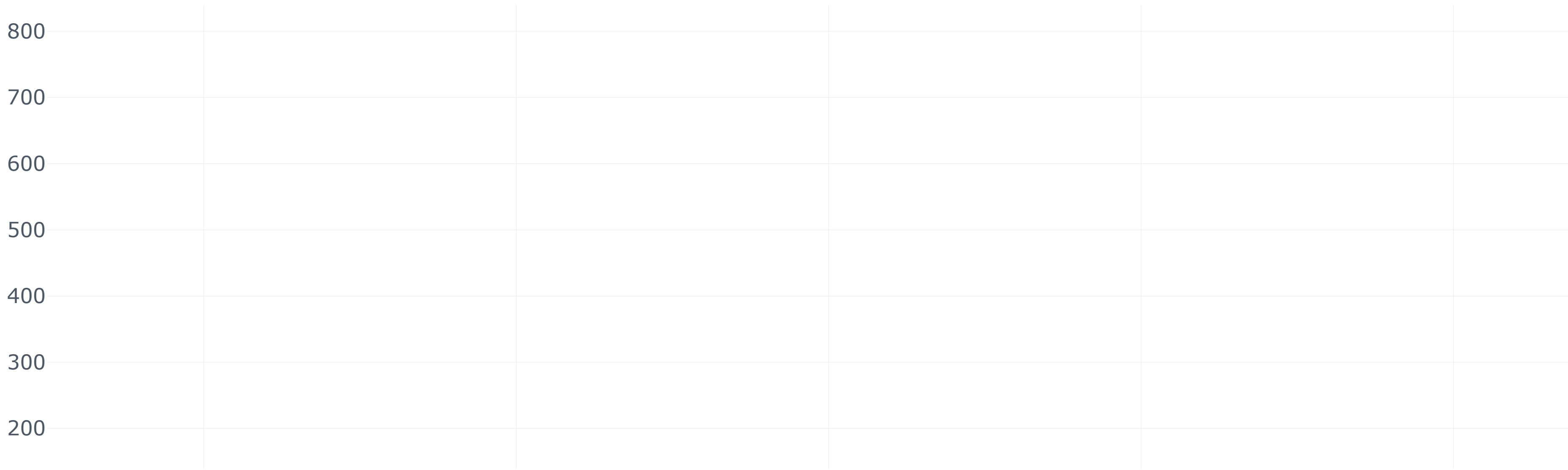
```
memory usage: 1.0 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   组号         25 non-null     float32
 1   厚度mm       25 non-null     float32
 2   孔隙率（%）      25 non-null     float32
 3   压缩回弹性率（%）   25 non-null     float32
 4   过滤阻力Pa     25 non-null     float32
 5   过滤效率（%）     25 non-null     float32
 6   透气性 mm/s   25 non-null     float32
 7   插层率（%）      25 non-null     float32
dtypes: float32(8)
memory usage: 928.0 bytes
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, 20 to 35
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   800     4 non-null      float32
 1   900     4 non-null      float32
 2   1000    4 non-null      float32
 3   1100    4 non-null      float32
 4   1200    4 non-null      float32
dtypes: float32(5)
memory usage: 112.0+ bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75 entries, 0 to 74
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   接收距离(cm)     75 non-null     int32  
 1   热风速度(r/min)  75 non-null     int32  
 2   厚度mm         75 non-null     float32
 3   孔隙率（%）        75 non-null     float32
 4   压缩回弹性率（%）     75 non-null     float32
 5   过滤阻力Pa       75 non-null     float32
 6   过滤效率（%）       75 non-null     float32
 7   透气性 mm/s     75 non-null     float32
dtypes: float32(6), int32(2)
memory usage: 2.5 KB
```

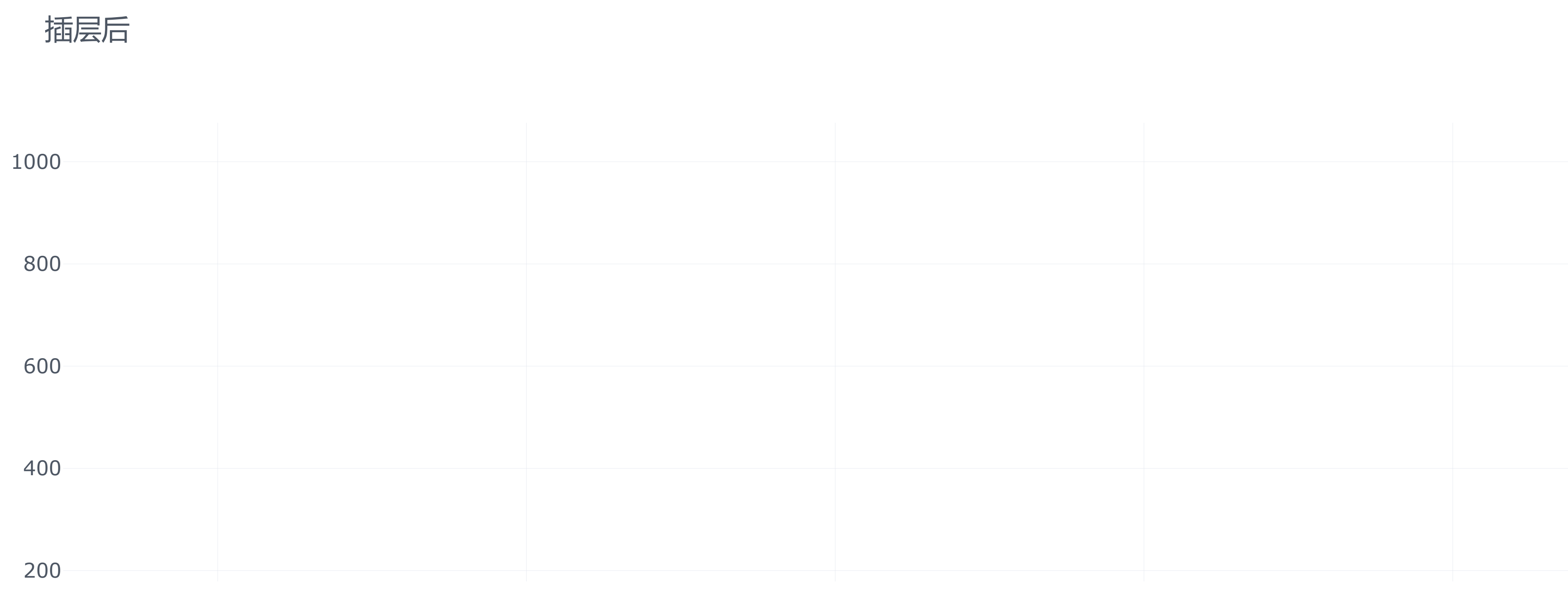
箱型图

箱型图（未标准化）

In [4]:
```
# TODO 数据未标准化箱型图

# 插层前
data1_uninternotated_unnumbered.iloc[:, 1:-1].iplot(
    kind='box',
    title='插层前',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE'],
    boxpoints='outliers',
)
# 插层后
data1_interpolated_unnumbered.iloc[:, 1:-1].iplot(
    kind='box',
    title='插层后',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE'],
    boxpoints='suspectedoutliers',
)
```

插层前





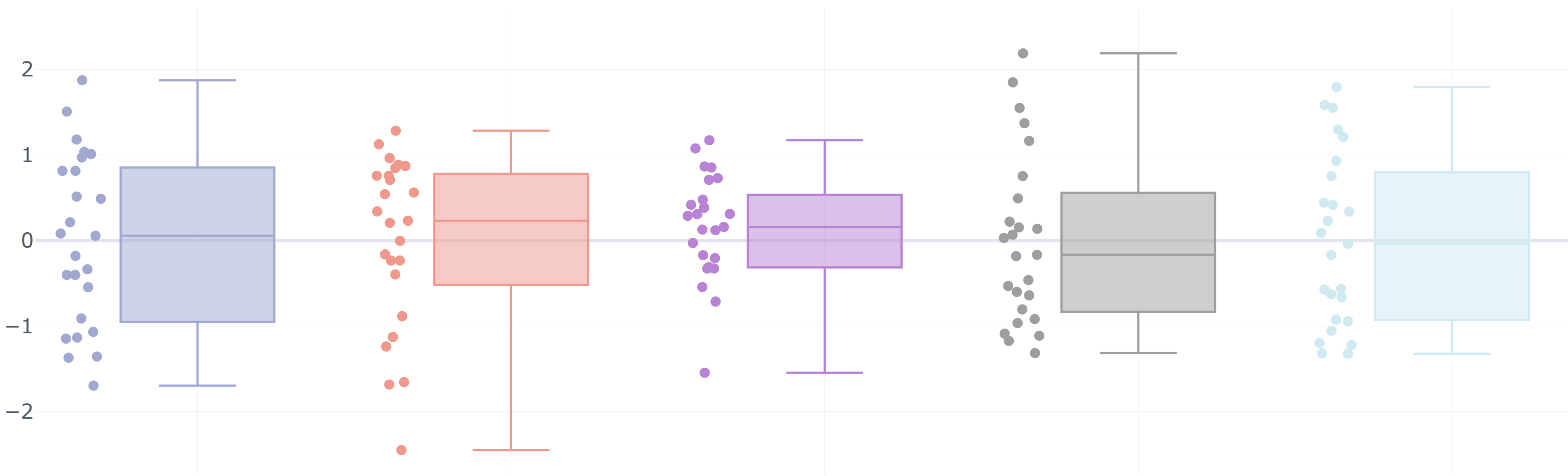
箱型图（标准化）

In [5]:

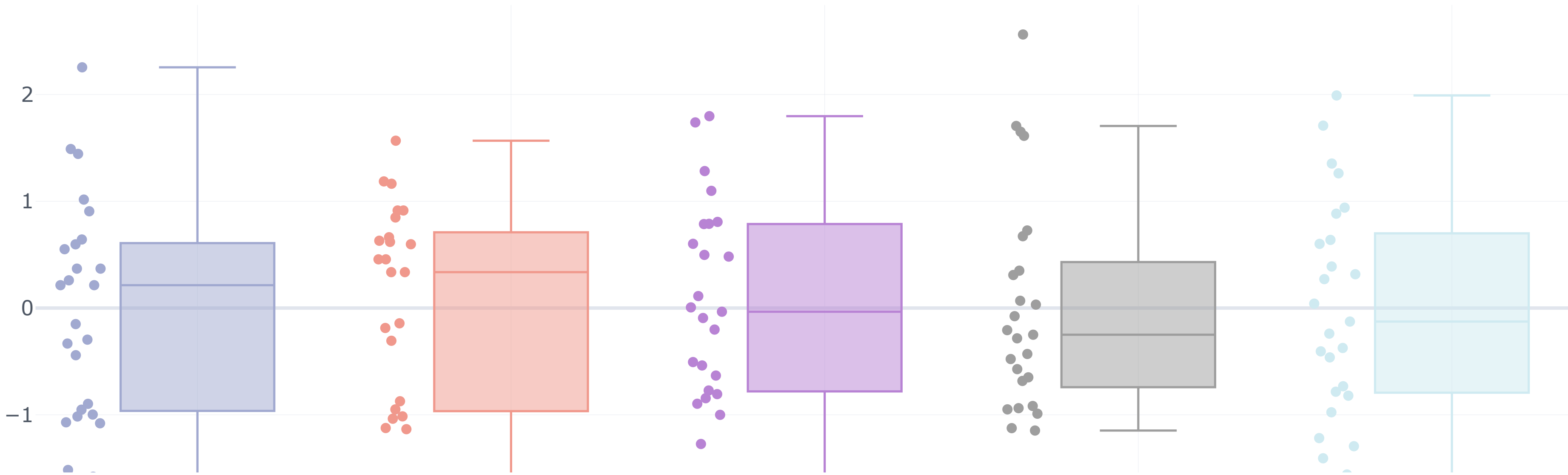
```
# TODO 数据标准化箱型图
from hmz.math_model.process import Dimensionalize

# 插层前--数据未标准化箱型图
dim = Dimensionalize(data1_uninternotated_unnumbered.iloc[:, 1:-1])
data1_uninternotated_unnumbered_std = dim.fit(method='standard')
data1_uninternotated_unnumbered_std.iplot(
    kind='box',
    title='插层前结构变量与产品性能',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE', '#F6CAE5'],
    boxpoints='all',
```

```python
)
data1_uninternotated_unnumbered_std.figure(
    kind='box',
    title='插层前结构变量与产品性能',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE', '#F6CAE5'],
    boxpoints='all',
).write_image('./img/问题1-插层前结构变量与产品性能箱型图.svg')

# 插层后--数据标准化箱型图
dim = Dimensionalize(data1_interpolated_unnumbered.iloc[:, 1:-1])
data1_interpolated_unnumbered_std = dim.fit(method='standard')
data1_interpolated_unnumbered_std.iplot(
    kind='box',
    title='插层后结构变量与产品性能',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE', '#F6CAE5'],
    boxpoints='all',
)
data1_interpolated_unnumbered_std.figure(
    kind='box',
    title='插层后结构变量与产品性能',
    colors=['#A1A9D0', '#F0988C', '#B883D4', '#9E9E9E', '#CFEAF1', '#C4A5DE', '#F6CAE5'],
    boxpoints='all',
).write_image('./img/问题1-插层后结构变量与产品性能箱型图.svg')
```

插层前结构变量与产品性能



插层后结构变量与产品性能



插层前后对比

In [6]:
```
InteractiveShell.ast_node_interactivity = 'last'

colors = ["#929fff", "#9de0ff", "#ffa897", "#af87fe", "#7dc3fe", "#bb60b2",
          "#433e7c", "#f47a75", "#009db2", "#024b51", "#0780cf", "#765005"]

plot_line_data1 = copy(data1_uninternotated_unnumbered.iloc[:, 1:-1])
plot_line_data1.columns = list(map(lambda x: '插层前-' + x, plot_line_data1.columns))
plot_line_data2 = copy(data1_interpolated_unnumbered.iloc[:, 1:-1])
plot_line_data2.columns = list(map(lambda x: '插层后-' + x, plot_line_data2.columns))
plot_line_data = pd.concat([plot_line_data1, plot_line_data2], axis=1)
```

```
plot_line_data
plot_line_data.info()
```

Out[6]:

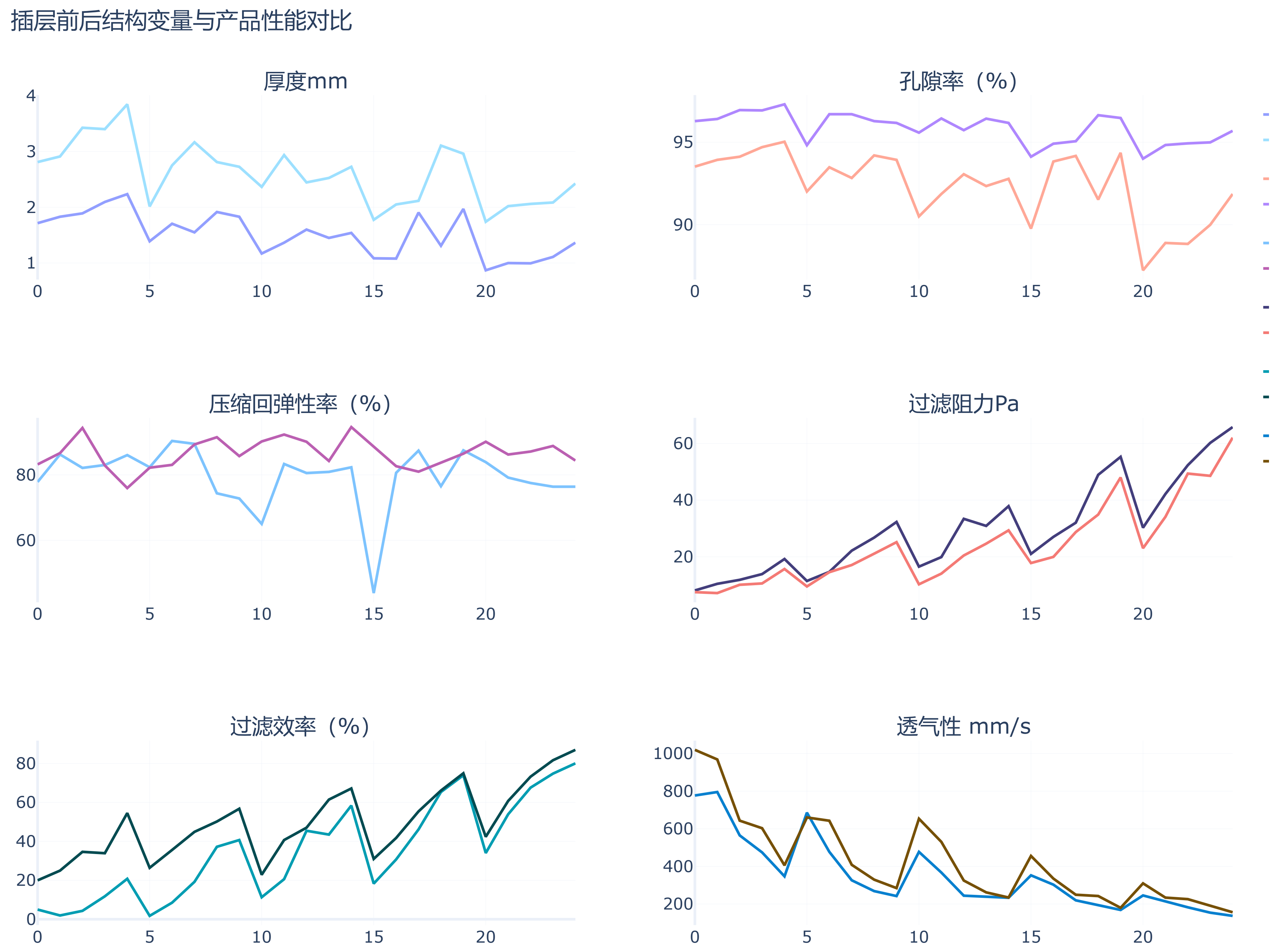
	插层前-厚度mm	插层前-孔隙率（%）	插层前-压缩回弹性率（%）	插层前-过滤阻力Pa	插层前-过滤效率（%）	插层前-透气性 mm/s	插层后-厚度mm	插层后-孔隙率（%）	插层后-压缩回弹性率（%）	插层后-过滤阻力Pa	插层后-过滤效率（%）	插层后-透气性 mm/s
0	1.715	93.519997	77.839996	8.130000	4.967000	777.099976	2.810	96.279999	83.199997	7.533000	19.966999	1019.669983
1	1.830	93.930000	86.230003	10.470000	1.933000	795.570007	2.910	96.410004	86.650002	7.200000	24.966999	968.630005
2	1.890	94.120003	82.120003	11.870000	4.300000	564.929993	3.425	96.949997	94.330002	10.133000	34.599998	643.400024
3	2.095	94.699997	83.010002	13.900000	11.767000	474.500000	3.400	96.930000	82.879997	10.600000	33.900002	603.169983
4	2.235	95.029999	86.040001	19.230000	20.767000	347.230011	3.845	97.300003	75.970001	15.700000	54.500000	405.829987
...	...	...	...	...	...	...	...	...	...	...	...	...
20	0.870	87.230003	83.889999	30.270000	34.000000	245.699997	1.740	94.000000	90.120003	23.033001	42.333000	309.929993
21	1.000	88.889999	79.169998	42.169998	53.900002	211.399994	2.020	94.830002	86.209999	34.099998	60.733002	234.130005
22	0.995	88.830002	77.519997	52.330002	67.567001	182.630005	2.060	94.930000	87.120003	49.367001	73.199997	226.029999
23	1.110	89.989998	76.389999	60.200001	74.766998	154.199997	2.085	94.989998	88.830002	48.567001	81.633003	191.100006
24	1.365	91.860001	76.389999	65.769997	80.032997	137.169998	2.425	95.690002	84.379997	62.033001	86.967003	156.330002

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   插层前-厚度mm        25 non-null     float32
 1   插层前-孔隙率（%）      25 non-null     float32
 2   插层前-压缩回弹性率（%）   25 non-null     float32
 3   插层前-过滤阻力Pa      25 non-null     float32
 4   插层前-过滤效率（%）     25 non-null     float32
 5   插层前-透气性 mm/s    25 non-null     float32
 6   插层后-厚度mm        25 non-null     float32
 7   插层后-孔隙率（%）      25 non-null     float32
 8   插层后-压缩回弹性率（%）   25 non-null     float32
 9   插层后-过滤阻力Pa      25 non-null     float32
 10  插层后-过滤效率（%）     25 non-null     float32
 11  插层后-透气性 mm/s    25 non-null     float32
dtypes: float32(12)
memory usage: 1.3 KB
```

In [7]:

```
InteractiveShell.ast_node_interactivity = 'last'
```

```python
fig = plotly.subplots.make_subplots(rows=3, cols=2, subplot_titles=data1_interpolated_unnumbered.columns[1:])
for i in range(6):
    row, col = i // 2 + 1, i % 2 + 1
    trace_pre = go.Scatter(
        x=plot_line_data.index,
        y=plot_line_data.iloc[:, 0 + i],
        line=dict(color=colors[2 * i]),
#         width=4,
        name=plot_line_data.columns[0 + i],
        legendgroup=data1_interpolated_unnumbered.columns[1:][i],
#         marker=dict(colorscale="RdBu"),
    )
    fig.append_trace(trace_pre, row, col)
    trace_suc = go.Scatter(
        x=plot_line_data.index,
        y=plot_line_data.iloc[:, 6 + i],
        line=dict(color=colors[2 * i + 1]),
#         dash='dash',  # 'dash', 'dot', and 'dashdot'
        name=plot_line_data.columns[6 + i],
        legendgroup=data1_interpolated_unnumbered.columns[1:][i],
#         marker=dict(colorscale="Viridis"),
    )
    fig.append_trace(trace_suc, row, col)
fig['layout'].update(
    height=800,
    width=1200,
    title='插层前后结构变量与产品性能对比',
    template='plotly_white',
)
fig.write_image('./img/问题1-插层前后结构变量与产品性能对比.svg')
fig.show()
```



插层前后结构变量与产品性能对比
厚度mm
孔隙率（%）
压缩回弹性率（%）
过滤阻力Pa
过滤效率（%）
透气性 mm/s

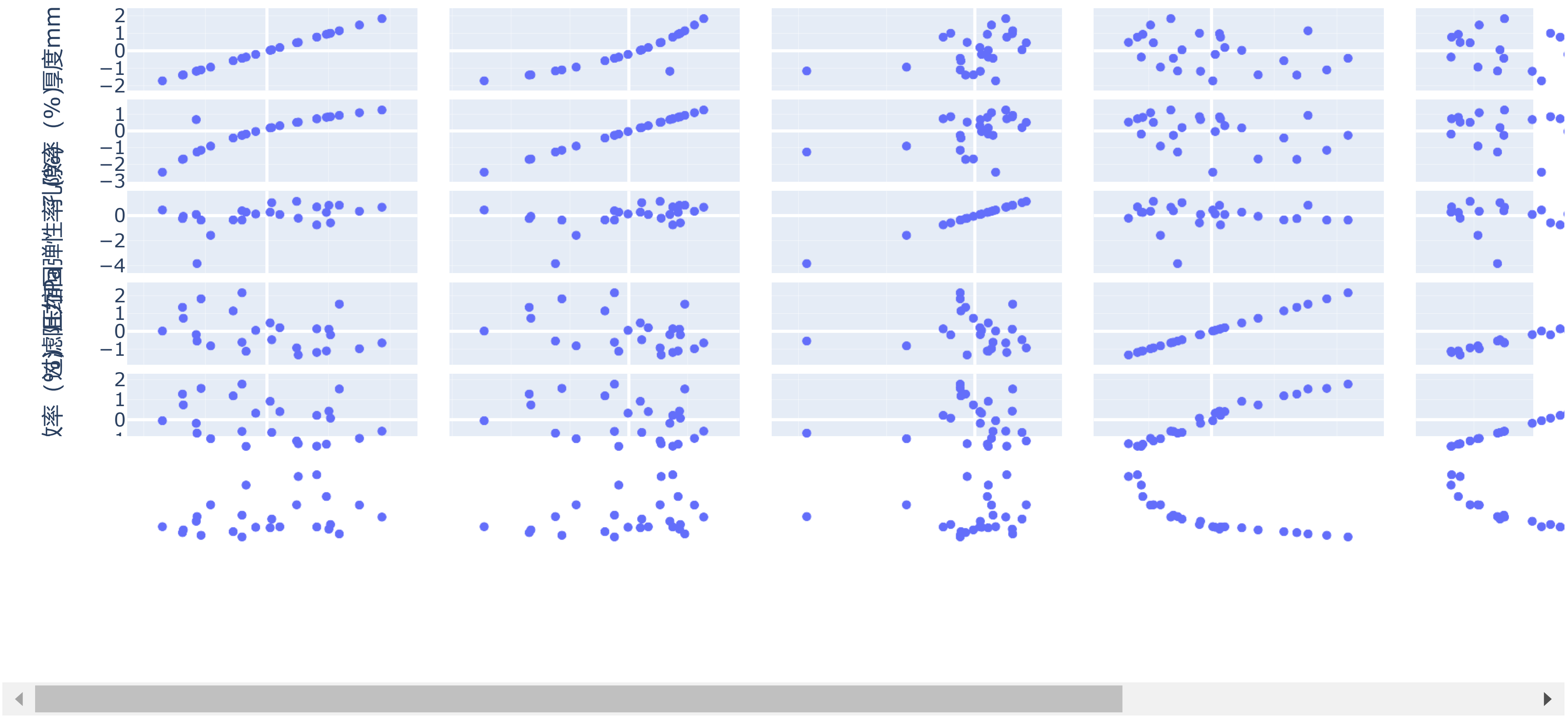
In [ ]:

结构变量之间、产品性能之间：矩阵散点图&相关系数热力图

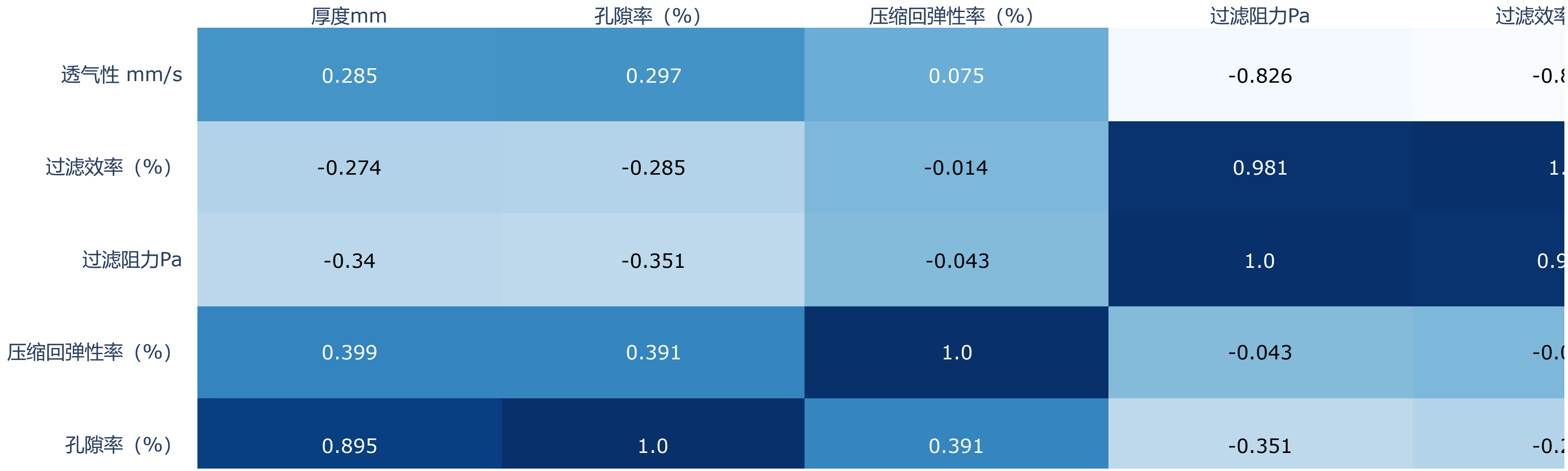
In [8]:

```
# TODO 矩阵散点图&相关系数热力图
def plot_ScatterMatrix_Heatmap(data, fig_pre_name, save=False, height=900, width=1100):
    # 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', height=900, width=1100)
    fig.show()

    # 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='blues',
    )
    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', height=650, width=800)
    figure.show()
    return None
save = True
plot_ScatterMatrix_Heatmap(data1_uninternotated_unnumbered_std, '插层前结构变量、产品性能', save=save)
plot_ScatterMatrix_Heatmap(data1_interpolated_unnumbered_std, '插层后结构变量、产品性能', save=save)
```

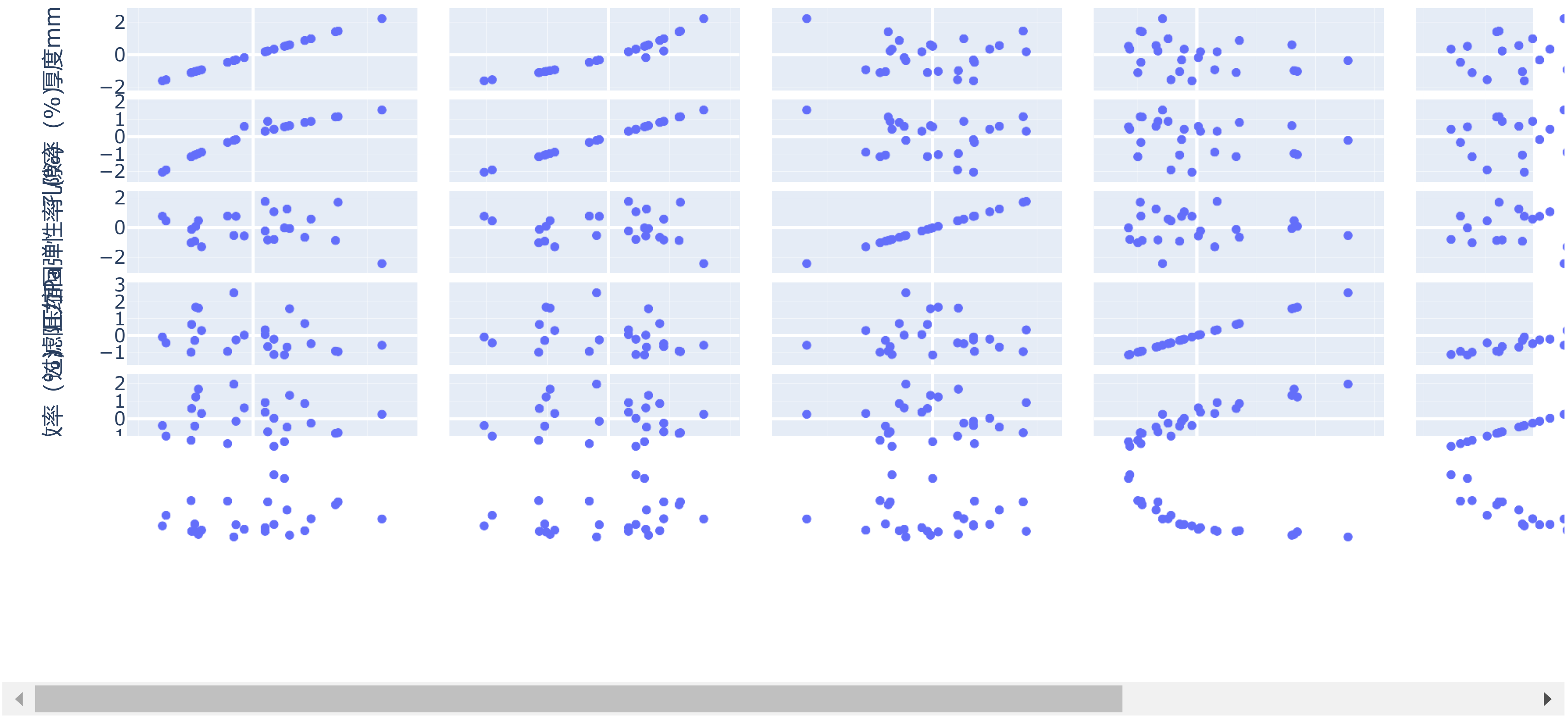
插层前结构变量、产品性能的矩阵散点图

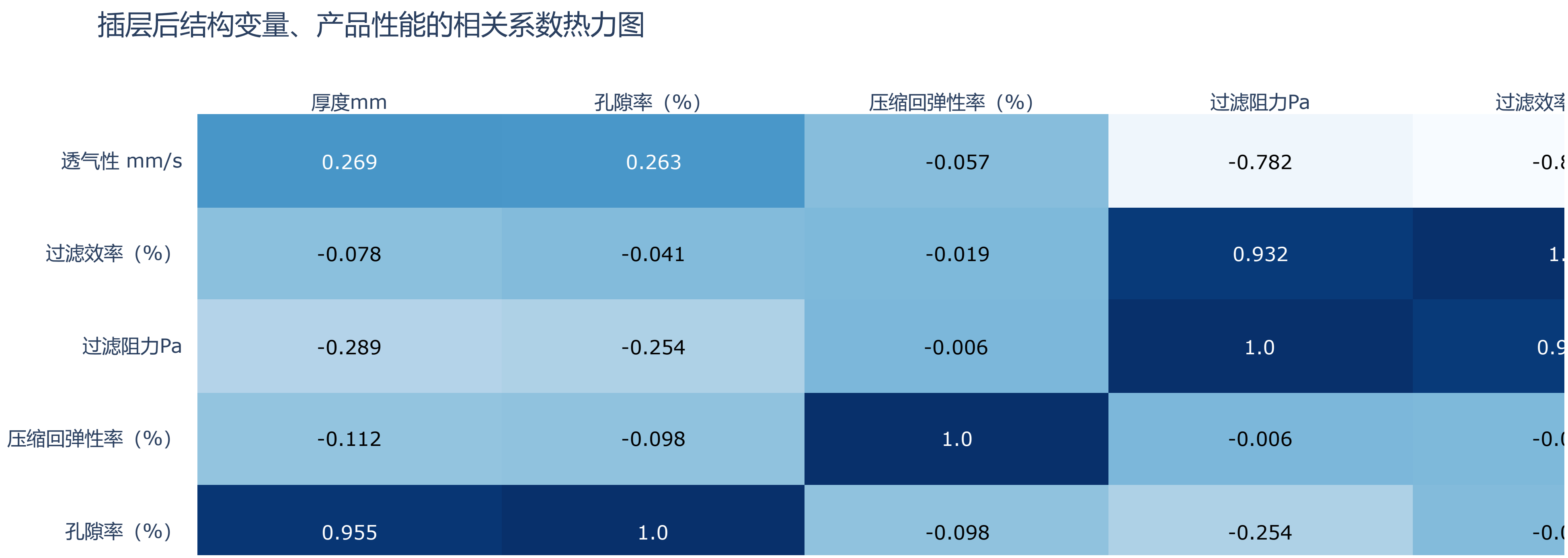


插层前结构变量、产品性能的相关系数热力图



插层后结构变量、产品性能的矩阵散点图



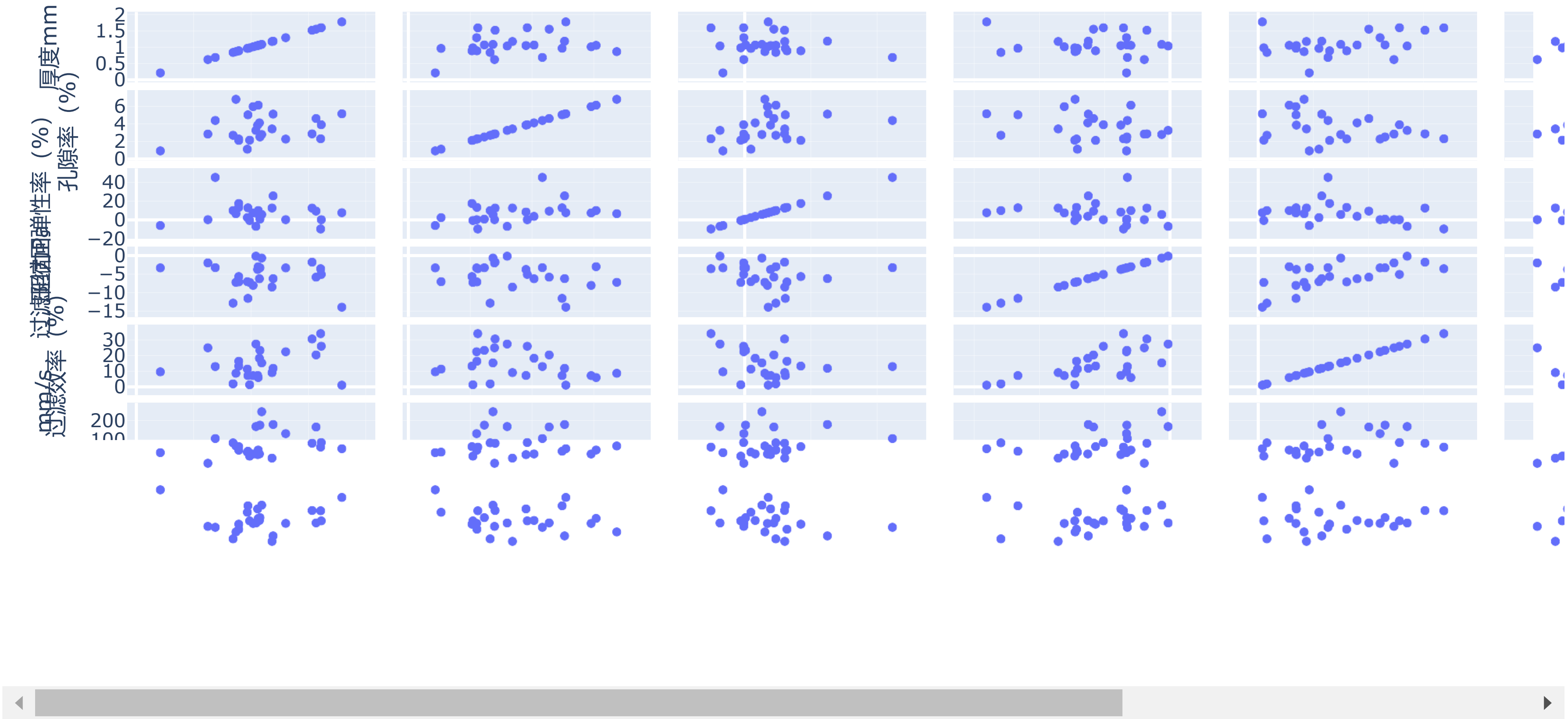


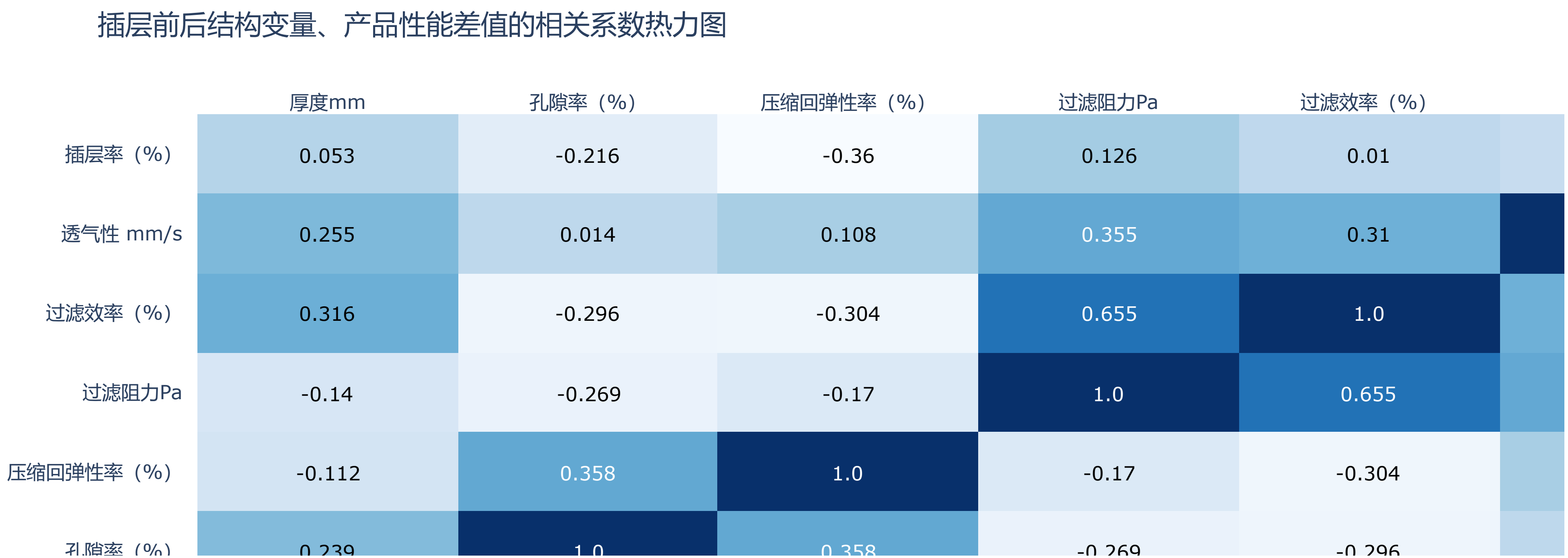
灰色关联分析

In [9]:

```python
plot_ScatterMatrix_Heatmap(
    data1_interpolated_unnumbered.iloc[:, 1:] - data1_uninternotated_unnumbered.iloc[:, 1:],
    '插层前后结构变量、产品性能差值',
    save=True,
)
```

插层前后结构变量、产品性能差值的矩阵散点图





In [10]:

```
from hmz.math_model.evaluate import GRA

InteractiveShell.ast_node_interactivity = 'all'

gra_data = data1_interpolated_unnumbered.iloc[:, 1:] - data1_uninternotated_unnumbered.iloc[:, 1:]
columns = list(gra_data.columns)
gra_data = gra_data.loc[:, columns[-1:] + columns[:-1]]
gra_data
gra_data.info()

gra = GRA(gra_data)
gra_score = gra.score()
gamma = pd.DataFrame(gra.gamma(), index=list(gra_data.columns[1:]), columns=['灰色关联度']).T
```

```
gamma
gamma.to_csv('./灰色关联度.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   插层率（%）     25 non-null     float32
 1   厚度mm      25 non-null     float32
 2   孔隙率（%）     25 non-null     float32
 3   压缩回弹性率（%）  25 non-null     float32
 4   过滤阻力Pa     25 non-null     float32
 5   过滤效率（%）     25 non-null     float32
 6   透气性 mm/s   25 non-null     float32
dtypes: float32(7)
memory usage: 828.0 bytes
灰色关联度:
[0.8959108  0.8476055  0.7327718  0.8260256  0.81745905 0.7853022 ]

灰色关联系数:
        厚度mm    孔隙率（%）   压缩回弹性率（%）    过滤阻力Pa   过滤效率（%）   透气性 mm/s
0   0.576020  0.796022   0.835506  1.475805  0.539845  1.672746
1   0.081890  0.367839   1.015504  0.492871  0.526189  1.248016
2   0.044790  0.559471   0.334995  1.056592  0.739672  0.312419
3   0.357915  0.205906   0.858976  0.254423  0.696714  0.885537
4   0.124973  0.707635   2.756264  0.726668  0.989451  0.567735
5   0.143453  0.082528   0.737080  0.373357  0.996039  1.062008
6   0.103448  0.060354   1.874436  0.838196  1.023852  1.361789
7   0.540669  0.159046   0.967954  0.041888  0.840206  0.160957
8   0.013843  0.216058   1.575688  0.199313  0.092507  0.008802
9   0.220045  0.035701   1.199816  0.665509  0.513998  0.033619
10  0.766514  1.115243   3.173759  0.775920  0.469974  2.031568
11  0.580944  0.442825   0.389211  0.175319  0.535077  1.325751
12  0.566117  0.550143   1.119752  2.088241  0.101250  0.870792
13  0.027254  0.204327   0.496543  0.153469  0.287851  0.652056
14  0.980026  0.856734   1.601493  1.408600  0.500449  0.093232
15  0.047278  0.560338   5.564355  0.107109  0.198861  0.701269
16  0.402163  0.985698   1.011925  0.038035  0.526750  0.859836
17  2.015413  1.954904   3.102322  1.622404  1.564539  1.809826
18  0.249025  0.440073   0.884174  0.595368  1.846941  1.225723
19  0.033029  0.341657   1.085884  0.349513  0.872984  0.789899
20  0.307778  1.436262   0.376130  0.794689  0.087483  0.370383
21  0.096979  0.851340   0.140334  0.594132  0.365314  0.535028
22  0.064413  0.694168   0.294091  0.516173  0.651408  0.460421
23  0.661943  0.133899   0.174712  0.509735  1.080606  1.062489
24  0.456667  0.339848   0.317768  0.766285  0.948687  1.173318

权重:
```

```
        厚度mm    孔隙率（%）  压缩回弹性率（%）    过滤阻力Pa   过滤效率（%）  透气性 mm/s
weight  0.18265  0.172802   0.149391  0.168402  0.166656    0.1601

最终得分:
    final score
0      0.973884
1      0.600727
2      0.506125
3      0.530008
4      0.935031
5      0.549473
6      0.839154
7      0.443688
8      0.325647
9      0.428718
10     1.341093
11     0.571725
12     0.873699
13     0.292675
14     0.901835
15     1.100177
16     0.626808
17     1.993092
18     0.857920
19     0.558102
20     0.568300
21     0.432384
22     0.444852
23     0.606176
24     0.664604

灰色关联度:
[0.8959108  0.8476055  0.7327718  0.8260256  0.81745905 0.7853022 ]
```

插层率对其他xxx的影响（忽略，最终未使用）

In [11]:

```python
data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='厚度mm')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='孔隙率（%）')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='压缩回弹性率（%）')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='过滤阻力Pa')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='过滤效率（%）')
# data1_interpolated_unnumbered.iloc[:, 1:].sort_values(by='插层率（%）').iplot(x='插层率（%）', y='透气性 mm/s')
```

数据处理（补充）

In [12]:
```
InteractiveShell.ast_node_interactivity = 'all'

data3_data2 = copy(data3.iloc[:, :2].drop_duplicates())
data2_data1 = data3_data2.reset_index(drop=True, inplace=False)  # index

def add_specf_df(data, is_numbered, specf_df=data2_data1):
    data = pd.concat([data, specf_df], axis=1)
    columns = list(data.columns)
    i = 2 if is_numbered else 1
    columns = copy(columns[:i] + columns[-2:] + columns[i:-2])
    data = data[columns]
```

```python
    return data

def update_df(df1, df2, df3, df4):
    df1 = add_specf_df(df1, is_numbered=True)
    df2 = add_specf_df(df2, is_numbered=False)
    df3 = add_specf_df(df3, is_numbered=True)
    df4 = add_specf_df(df4, is_numbered=False)
    return df1, df2, df3, df4

data1_uninternotated, data1_uninternotated_unnumbered, data1_interpolated, data1_interpolated_unnumbered = \
    update_df(data1_uninternotated, data1_uninternotated_unnumbered,
              data1_interpolated, data1_interpolated_unnumbered)
data1_uninternotated
data1_uninternotated.info()
data1_uninternotated_unnumbered
data1_uninternotated_unnumbered.info()
data1_interpolated
data1_interpolated.info()
data1_interpolated_unnumbered
data1_interpolated_unnumbered.info()
```

Out[12]:

	组号	编号	接收距离(cm)	热风速度(r/min)	厚度mm	孔隙率（%）	压缩回弹性率（%）	过滤阻力Pa	过滤效率（%）	透气性 mm/s	插层率（%）
0	1.0	1#	40	800	1.715	93.519997	77.839996	8.130000	4.967000	777.099976	0.0
1	2.0	1#	40	900	1.830	93.930000	86.230003	10.470000	1.933000	795.570007	0.0
2	3.0	1#	40	1000	1.890	94.120003	82.120003	11.870000	4.300000	564.929993	0.0
3	4.0	1#	40	1100	2.095	94.699997	83.010002	13.900000	11.767000	474.500000	0.0
4	5.0	1#	40	1200	2.235	95.029999	86.040001	19.230000	20.767000	347.230011	0.0
...	...	...	...	...	...	...	...	...	...	...	...
20	21.0	1#	20	800	0.870	87.230003	83.889999	30.270000	34.000000	245.699997	0.0
21	22.0	1#	20	900	1.000	88.889999	79.169998	42.169998	53.900002	211.399994	0.0
22	23.0	1#	20	1000	0.995	88.830002	77.519997	52.330002	67.567001	182.630005	0.0
23	24.0	1#	20	1100	1.110	89.989998	76.389999	60.200001	74.766998	154.199997	0.0
24	25.0	1#	20	1200	1.365	91.860001	76.389999	65.769997	80.032997	137.169998	0.0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   组号           25 non-null     float32
 1   编号           25 non-null     category
 2   接收距离(cm)     25 non-null     int32
 3   热风速度(r/min)  25 non-null     int32
 4   厚度mm         25 non-null     float32
 5   孔隙率（%）       25 non-null     float32
 6   压缩回弹性率（%）     25 non-null     float32
 7   过滤阻力Pa       25 non-null     float32
 8   过滤效率（%）       25 non-null     float32
 9   透气性 mm/s     25 non-null     float32
 10  插层率（%）       25 non-null     float32
dtypes: category(1), float32(8), int32(2)
memory usage: 1.2 KB
```

Out[12]:

	组号	接收距离(cm)	热风速度(r/min)	厚度mm	孔隙率（%）	压缩回弹性率（%）	过滤阻力Pa	过滤效率（%）	透气性 mm/s	插层率（%）
0	1.0	40	800	1.715	93.519997	77.839996	8.130000	4.967000	777.099976	0.0
1	2.0	40	900	1.830	93.930000	86.230003	10.470000	1.933000	795.570007	0.0
2	3.0	40	1000	1.890	94.120003	82.120003	11.870000	4.300000	564.929993	0.0
3	4.0	40	1100	2.095	94.699997	83.010002	13.900000	11.767000	474.500000	0.0
4	5.0	40	1200	2.235	95.029999	86.040001	19.230000	20.767000	347.230011	0.0
...	...	...	...	...	...	...	...	...	...	...
20	21.0	20	800	0.870	87.230003	83.889999	30.270000	34.000000	245.699997	0.0
21	22.0	20	900	1.000	88.889999	79.169998	42.169998	53.900002	211.399994	0.0
22	23.0	20	1000	0.995	88.830002	77.519997	52.330002	67.567001	182.630005	0.0
23	24.0	20	1100	1.110	89.989998	76.389999	60.200001	74.766998	154.199997	0.0
24	25.0	20	1200	1.365	91.860001	76.389999	65.769997	80.032997	137.169998	0.0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   组号           25 non-null     float32
 1   接收距离(cm)     25 non-null     int32  
 2   热风速度(r/min)  25 non-null     int32  
 3   厚度mm         25 non-null     float32
 4   孔隙率（%）       25 non-null     float32
 5   压缩回弹性率（%）     25 non-null     float32
 6   过滤阻力Pa       25 non-null     float32
 7   过滤效率（%）       25 non-null     float32
 8   透气性 mm/s     25 non-null     float32
 9   插层率（%）       25 non-null     float32
dtypes: float32(8), int32(2)
memory usage: 1.1 KB
```

Out[12]:

	组号	编号	接收距离(cm)	热风速度(r/min)	厚度mm	孔隙率（%）	压缩回弹性率（%）	过滤阻力Pa	过滤效率（%）	透气性 mm/s	插层率（%）
0	1.0	2#	40	800	2.810	96.279999	83.199997	7.533000	19.966999	1019.669983	36.439999
1	2.0	2#	40	900	2.910	96.410004	86.650002	7.200000	24.966999	968.630005	24.740000
2	3.0	2#	40	1000	3.425	96.949997	94.330002	10.133000	34.599998	643.400024	31.450001
3	4.0	2#	40	1100	3.400	96.930000	82.879997	10.600000	33.900002	603.169983	19.370001
4	5.0	2#	40	1200	3.845	97.300003	75.970001	15.700000	54.500000	405.829987	31.190001
...	...	...	...	...	...	...	...	...	...	...	...
20	21.0	2#	20	800	1.740	94.000000	90.120003	23.033001	42.333000	309.929993	11.320000
21	22.0	2#	20	900	2.020	94.830002	86.209999	34.099998	60.733002	234.130005	19.350000
22	23.0	2#	20	1000	2.060	94.930000	87.120003	49.367001	73.199997	226.029999	24.020000
23	24.0	2#	20	1100	2.085	94.989998	88.830002	48.567001	81.633003	191.100006	35.880001
24	25.0	2#	20	1200	2.425	95.690002	84.379997	62.033001	86.967003	156.330002	32.950001

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   组号           25 non-null     float32 
 1   编号           25 non-null     category
 2   接收距离(cm)     25 non-null     int32   
 3   热风速度(r/min)  25 non-null     int32   
 4   厚度mm         25 non-null     float32 
 5   孔隙率（%）       25 non-null     float32 
 6   压缩回弹性率（%）     25 non-null     float32 
 7   过滤阻力Pa       25 non-null     float32 
 8   过滤效率（%）       25 non-null     float32 
 9   透气性 mm/s     25 non-null     float32 
 10  插层率（%）       25 non-null     float32 
dtypes: category(1), float32(8), int32(2)
memory usage: 1.2 KB
```

Out[12]:

	组号	接收距离(cm)	热风速度(r/min)	厚度mm	孔隙率（%）	压缩回弹性率（%）	过滤阻力Pa	过滤效率（%）	透气性 mm/s	插层率（%）
0	1.0	40	800	2.810	96.279999	83.199997	7.533000	19.966999	1019.669983	36.439999
1	2.0	40	900	2.910	96.410004	86.650002	7.200000	24.966999	968.630005	24.740000
2	3.0	40	1000	3.425	96.949997	94.330002	10.133000	34.599998	643.400024	31.450001
3	4.0	40	1100	3.400	96.930000	82.879997	10.600000	33.900002	603.169983	19.370001
4	5.0	40	1200	3.845	97.300003	75.970001	15.700000	54.500000	405.829987	31.190001
...	...	...	...	...	...	...	...	...	...	...
20	21.0	20	800	1.740	94.000000	90.120003	23.033001	42.333000	309.929993	11.320000
21	22.0	20	900	2.020	94.830002	86.209999	34.099998	60.733002	234.130005	19.350000
22	23.0	20	1000	2.060	94.930000	87.120003	49.367001	73.199997	226.029999	24.020000
23	24.0	20	1100	2.085	94.989998	88.830002	48.567001	81.633003	191.100006	35.880001
24	25.0	20	1200	2.425	95.690002	84.379997	62.033001	86.967003	156.330002	32.950001

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   组号           25 non-null     float32
 1   接收距离(cm)     25 non-null     int32  
 2   热风速度(r/min)  25 non-null     int32  
 3   厚度mm         25 non-null     float32
 4   孔隙率（%）        25 non-null     float32
 5   压缩回弹性率（%）     25 non-null     float32
 6   过滤阻力Pa       25 non-null     float32
 7   过滤效率（%）       25 non-null     float32
 8   透气性 mm/s     25 non-null     float32
 9   插层率（%）        25 non-null     float32
dtypes: float32(8), int32(2)
memory usage: 1.1 KB
```

问题2

选择模型

In [13]:
```python
from sklearn.linear_model import LinearRegression       # 线性回归
from sklearn.linear_model import Lasso, LassoCV         # 岭回归
from sklearn.linear_model import Ridge, RidgeCV         # LASSO回归
from sklearn.neighbors import KNeighborsRegressor       # KNN回归
from sklearn.neural_network import MLPRegressor         # 多层感知机回归
from sklearn.svm import SVR                             # SVM回归
from sklearn.tree import DecisionTreeRegressor          # 决策树回归
from sklearn.tree import ExtraTreeRegressor             # 极限树回归
from sklearn.ensemble import RandomForestRegressor      # 随机森林回归
from sklearn.ensemble import AdaBoostRegressor          # AdaBoost回归
from sklearn.ensemble import GradientBoostingRegressor  # 梯度提升回归
from sklearn.ensemble import BaggingRegressor           # Bagging回归

from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor

from catboost import CatBoostRegressor
from xgboost import XGBRegressor, XGBRFRegressor
from lightgbm import LGBMRegressor

import joblib
```

```python
from hmz.math_model.predict import predict_accuracy
```

In [14]:
```python
# from sklearn.preprocessing import MinMaxScaler

XY = copy(data3.iloc[:, :5])
X = np.array(XY.iloc[:, :2])
Y = np.array(XY.iloc[:, 2:])

# mms = MinMaxScaler()
# X = mms.fit_transform(X)
# Y = mms.fit_transform(Y)

colors = ["red", "lightpink", "darkorange", "khaki", "green", "lightgreen", "blue", "lightblue"]
colors = ["red", "lightpink", "green", "lightgreen", "blue", "lightblue"]

xpred = np.array([
    [38, 33, 28, 23, 38, 33, 28, 23],
    [850, 950, 1150, 1250, 1250, 1150, 950, 850],
]).T

def score(model, X, Y, xtrain, xtest, ytrain, ytest, Ypred):
    accuracy = predict_accuracy(Y, Ypred, type=1, th=0.005, con=0.8)
    print("准确率：", accuracy)
#     print("训练集分数：R^2 =", model.score(xtrain, ytrain))  # 负值：拟合效果糟糕，模型完全不能使用
#     print("测试集分数：R^2 =", model.score(xtest, ytest))  # -> 1：拟合效果很好，模型可以使用
#     print(r2_score(model.predict(xtrain), ytrain))
#     print(r2_score(model.predict(xtest), ytest))
#     scores = cross_val_score(model, X, Y, cv=10, scoring="r2")
#     print(scores.mean())
    return None

models = [
    LinearRegression(),
    Lasso(),
    Ridge(),
    KNeighborsRegressor(),
    MLPRegressor(),
    SVR(kernel='rbf', degree=200, gamma='auto'),  # 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'; ``'scale', 'auto'
    DecisionTreeRegressor(),
    ExtraTreeRegressor(),
    RandomForestRegressor(),
    AdaBoostRegressor(),
    GradientBoostingRegressor(),
    BaggingRegressor(),
    HistGradientBoostingRegressor(),
```

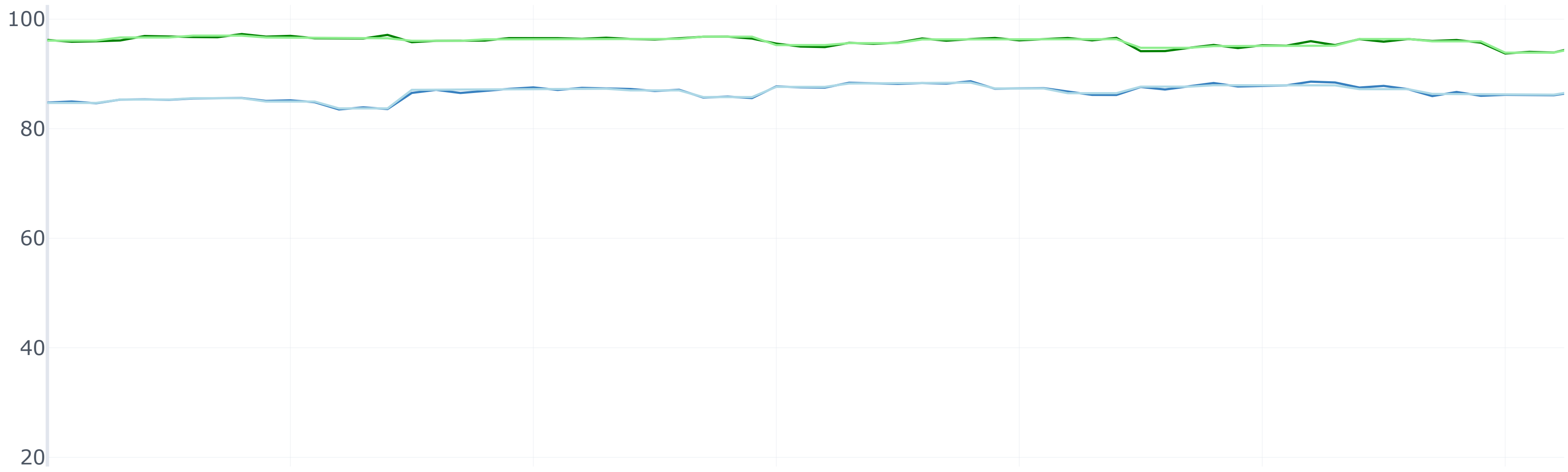
```
    CatBoostRegressor(verbose=0),
    XGBRegressor(),
    XGBRFRegressor(),
    LGBMRegressor(),
]
models_name = [
    "线性回归",
    "LASSO套索回归",
    "Ridge岭回归",
    "KNN回归",
    "多层感知机回归",
    "SVM回归",
    "决策树回归",
    "极限树回归",
    "随机森林回归",
    "AdaBoost回归",
    "梯度提升回归",
    "Bagging回归",
    "基于直方图的梯度提升回归",
    "CatBoost回归",
    "XGBoost回归",
    "XGBoostRF回归",
    "LightGBM回归",
]
models_select = {
    "线性回归": False,
    "LASSO套索回归": False,
    "Ridge岭回归": False,
    "KNN回归": False,
    "多层感知机回归": False,
    "SVM回归": False,
    "决策树回归": False,
    "极限树回归": True,  # 1
    "随机森林回归": True,  # 2
    "AdaBoost回归": False,
    "梯度提升回归": False,
    "Bagging回归": False,
    "基于直方图的梯度提升回归": False,
    "CatBoost回归": False,
    "XGBoost回归": True,  # 0
    "XGBoostRF回归": False,
    "LightGBM回归": False,
}
saved_models_name = ['xgb', 'et', 'rf']

def run_Regressor_models(
```

```python
    x=xpred, X=X, Y=Y, colors=colors, plot_fig=False, save_fig=False,
    models=models, models_name=models_name, models_select=models_select,
    saved_models_name=saved_models_name, save_model=False,
):
    """
    :param x: np.ndarray
    :param X: np.ndarray
    :param Y: np.ndarray
    XGBoost(), ET(), RF()
    """
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3)
    ypreds_s = []
    for model, model_name, model_idx in zip(models, models_name, range(len(models_name))):
#         print(models_select[model_name])
        if not models_select[model_name]:
            continue
        print("模型: ", model_name)
        ypreds = []
        ytests = []
        for i in range(Y.shape[1]):
            # 训练
            model.fit(xtrain, ytrain[:, i])
            # 测试
            ytest_ = model.predict(X)
            score(model, X, Y[:, i], xtrain, xtest, ytrain[:, i], ytest[:, i], ytest_)
            ytests.append(list(ytest_))
            # 预测
            ypred = model.predict(x)
            ypreds.append(list(ypred))
            # 保存
            if save_model:
                if model_name == "XGBoost回归" and i == 0:
                    joblib.dump(model, './saved_model/pre0_xgb.pkl')
                elif model_name == "极限树回归" and i == 1:
                    joblib.dump(model, './saved_model/pre1_et.pkl')
                elif model_name == "随机森林回归" and i == 2:
                    joblib.dump(model, './saved_model/pre2_rf.pkl')
        ypreds_s.append(ypreds)

        if plot_fig:
            # 画图
            ytests = pd.DataFrame(ytests, index=list(XY.columns[2:5])).T
            ytests.columns = map(lambda x: x + '-预测值', ytests.columns)
            ytesttemp = pd.DataFrame(Y, columns=list(XY.columns[2:5]))
            ytesttemp.columns = map(lambda x: x + '-真实值', ytesttemp.columns)
            yplot = pd.concat([ytesttemp, ytests], axis=1)
```

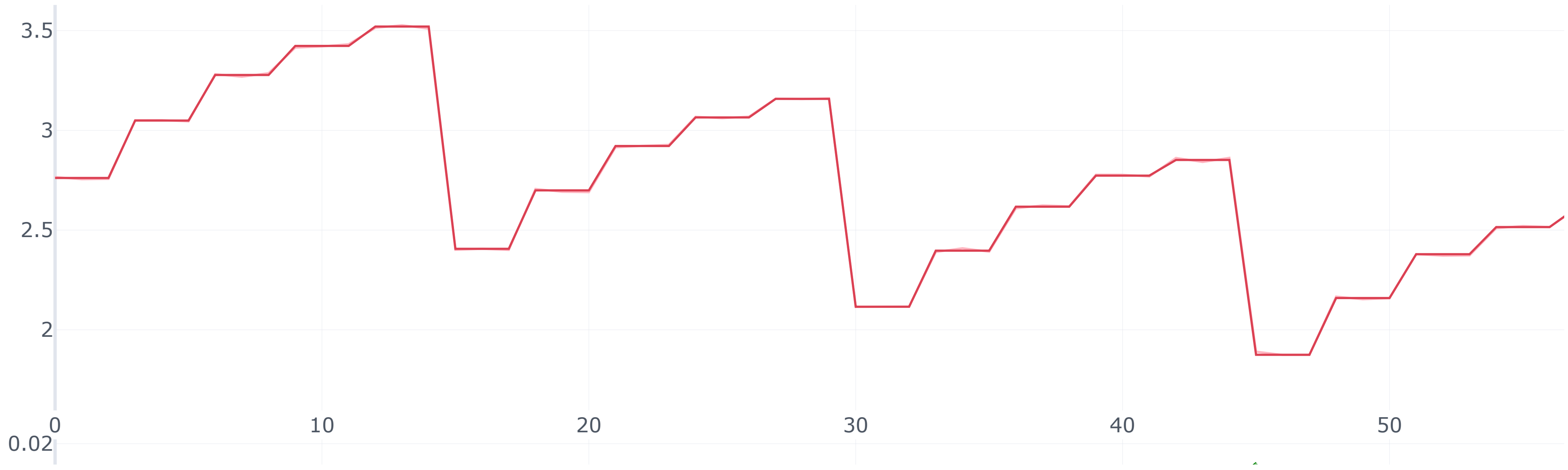
```python
    #         print(yplot.shape)
    #         print(yplot.head())
    #         print(ypreds.shape)
    #         print(ypreds.head())
    #         print(ytest_temp.shape)
    #         print(ytest_temp.head())

            title = '基于' + model_name + '的预测模型'
            yplot.iplot(
                color=colors[0::2] + colors[1::2],
                title=title,
            )
            if save_fig:
                yplot.figure(
                    color=colors[0::2] + colors[1::2],
                    title=title,
                ).write_image("./img/问题2-" + title + ".svg")

            yplot.columns = ['真实值'] * 3 + ['预测值'] * 3
            columns = list(XY.columns[2:5])
            for i in range(3):
                title = '基于' + model_name + '的预测模型——' + columns[i]
                yplot.iloc[:, i::3].iplot(
                    kind='spread',
                    color=[colors[i * 2 + 1], colors[i * 2]],
                    title=title,
                )
                if save_fig:
                    yplot.iloc[:, i::3].figure(
                        kind='spread',
                        color=[colors[i * 2 + 1], colors[i * 2]],
                        title=title,
                    ).write_image("./img/问题2-" + title + ".svg")

        print('-'*125)
    return ypreds_s
all_models_ypreds_q2 = np.array(run_Regressor_models(plot_fig=True, save_fig=True, save_model=True))
all_models_ypreds_q2
```

```
模型:  极限树回归
准确率:  0.9676401773238965
准确率:  0.9249176664432979
准确率:  0.9249265354576991
```

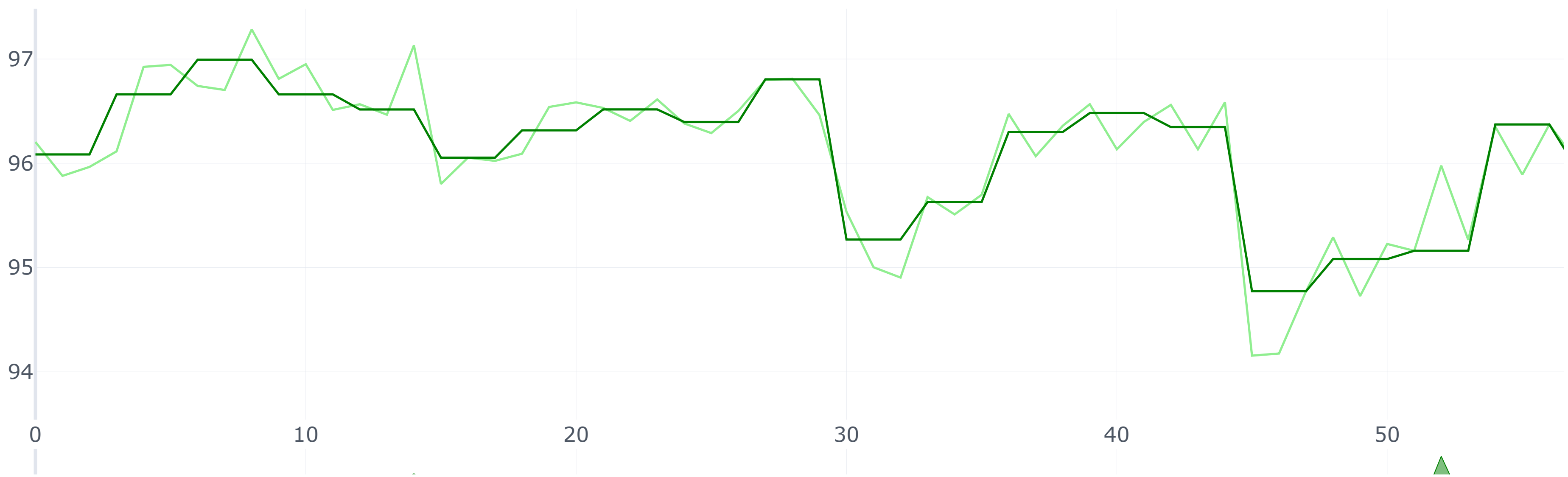
基于极限树回归的预测模型

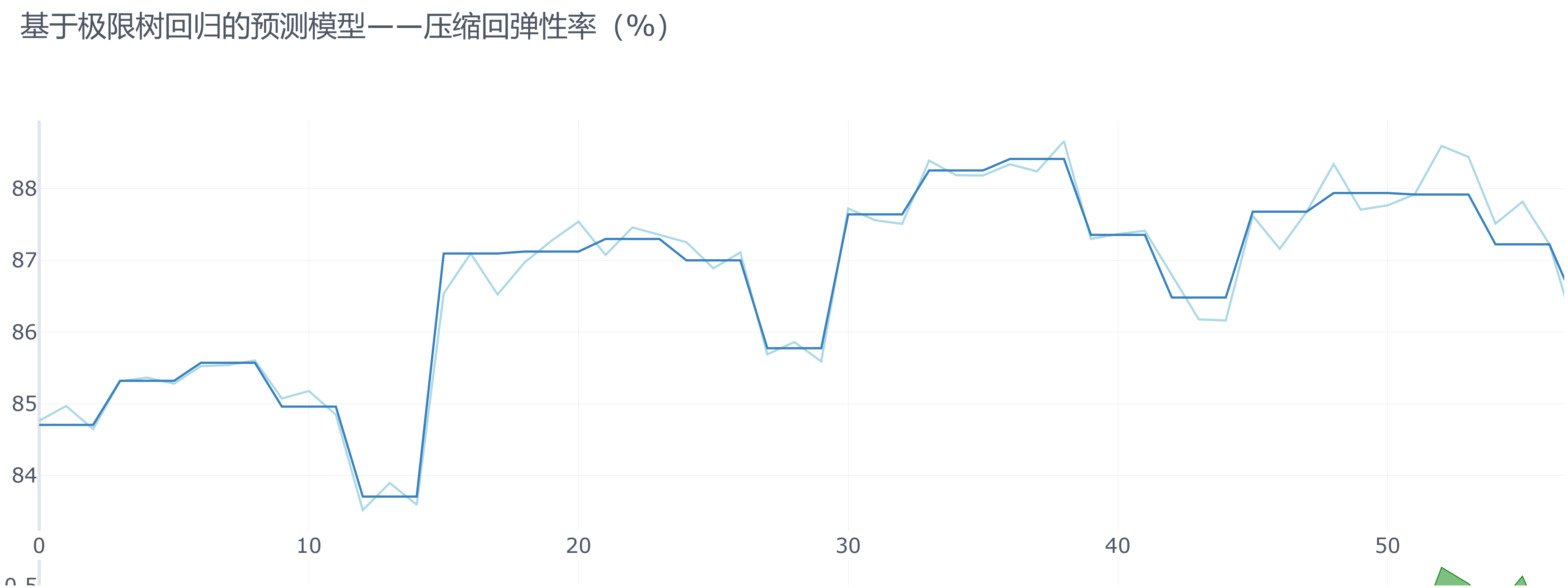


基于极限树回归的预测模型——厚度mm



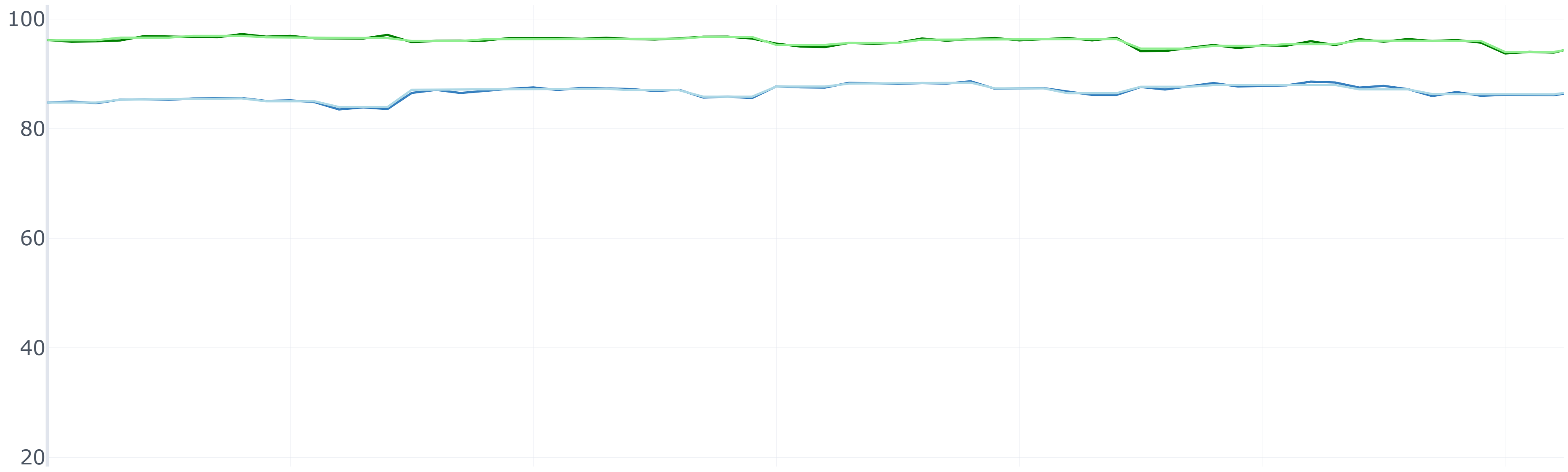
基于极限树回归的预测模型——孔隙率（%）



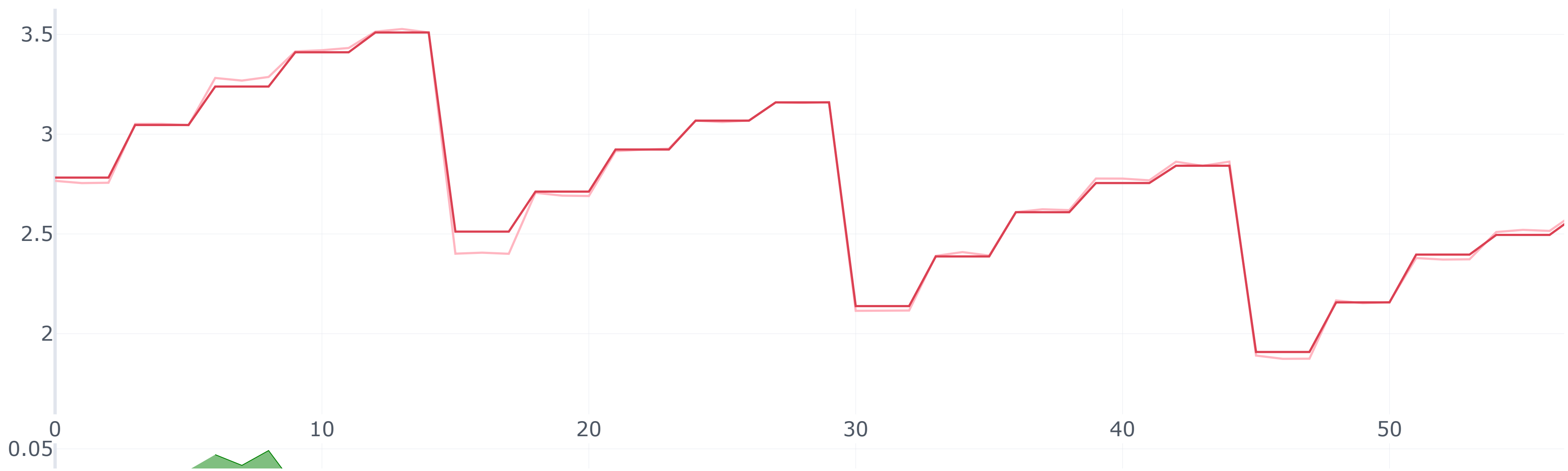


模型：　随机森林回归
准确率：　0.6038533538344204
准确率：　0.935574843455885
准确率：　0.9142364183417401

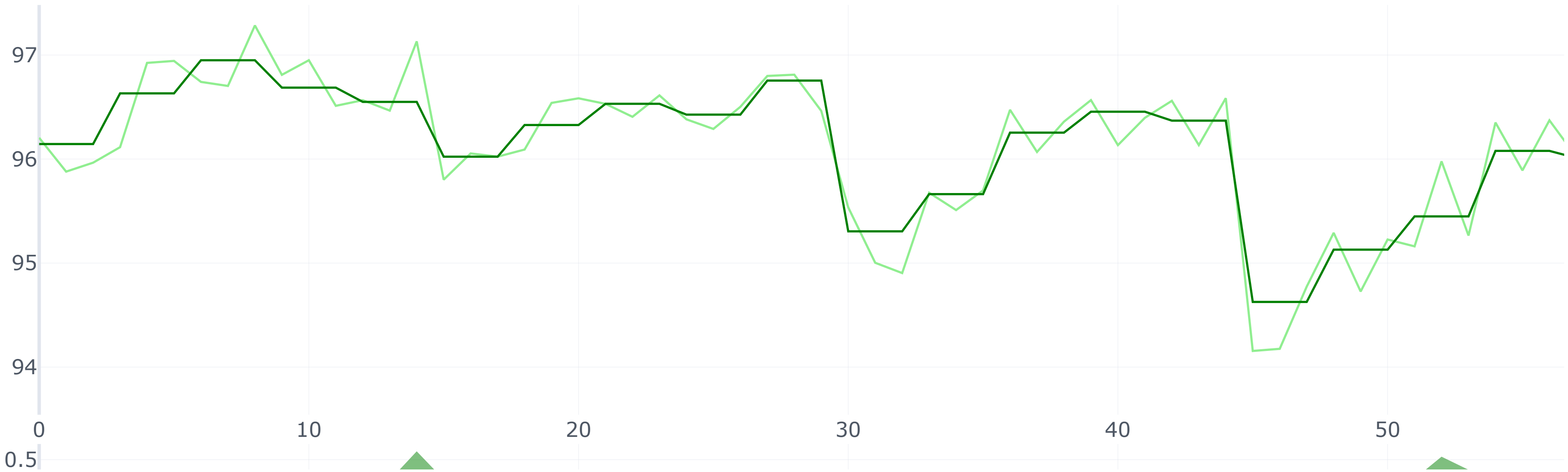
基于随机森林回归的预测模型



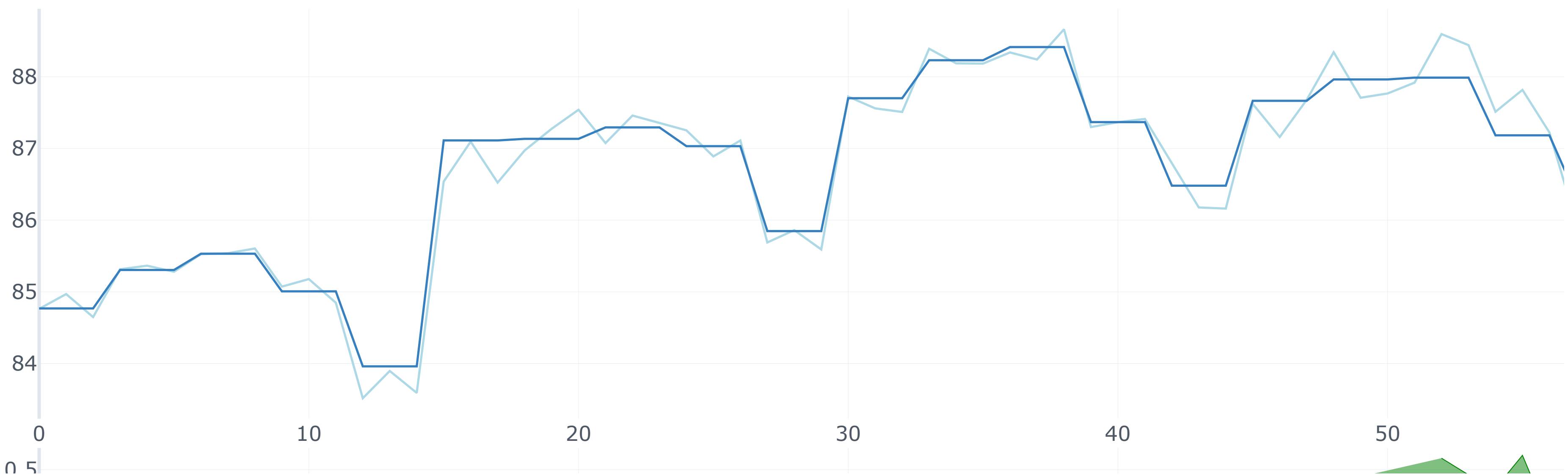
基于随机森林回归的预测模型——厚度mm



基于随机森林回归的预测模型——孔隙率（%）

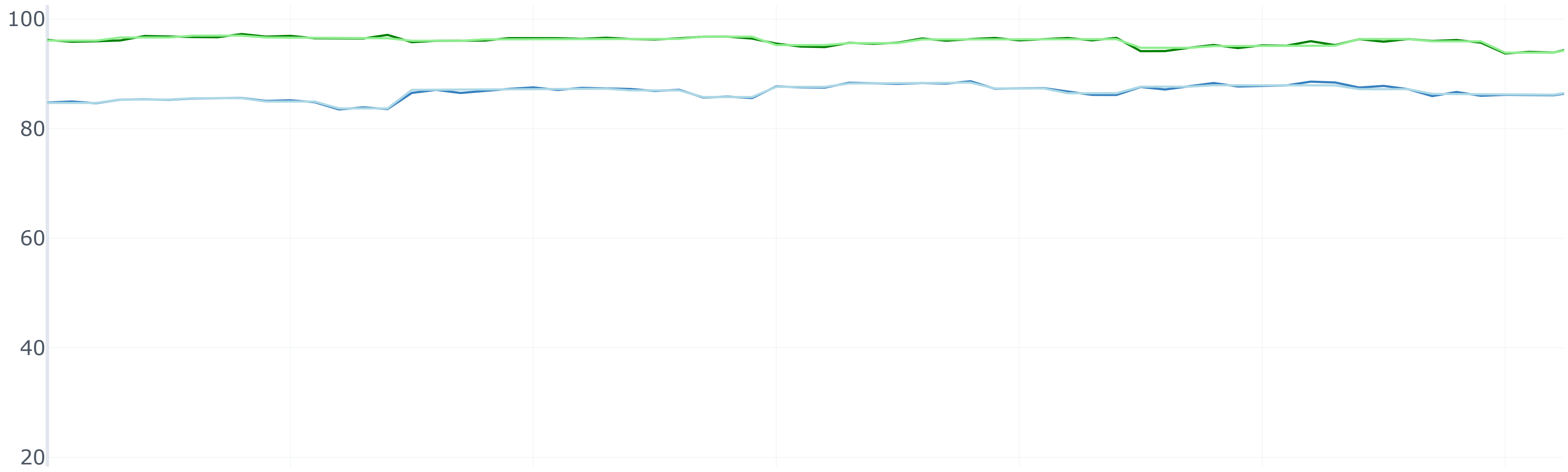


基于随机森林回归的预测模型——压缩回弹性率（%）

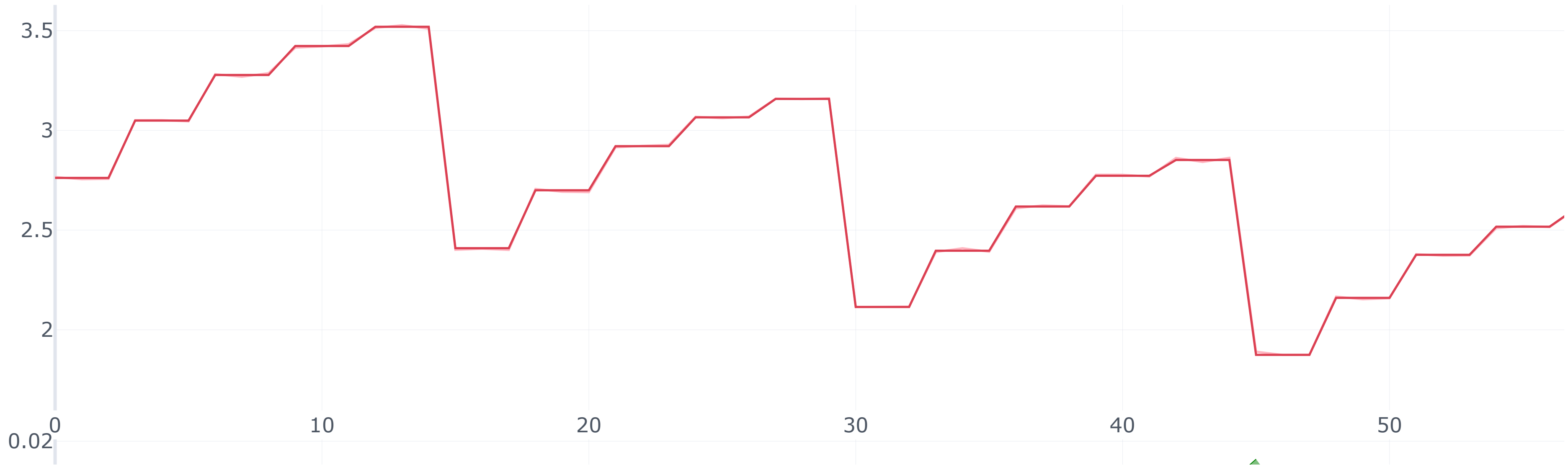


模型： XGBoost回归
准确率： 0.9676220567263663
准确率： 0.9355845125522465
准确率： 0.9249269273051371

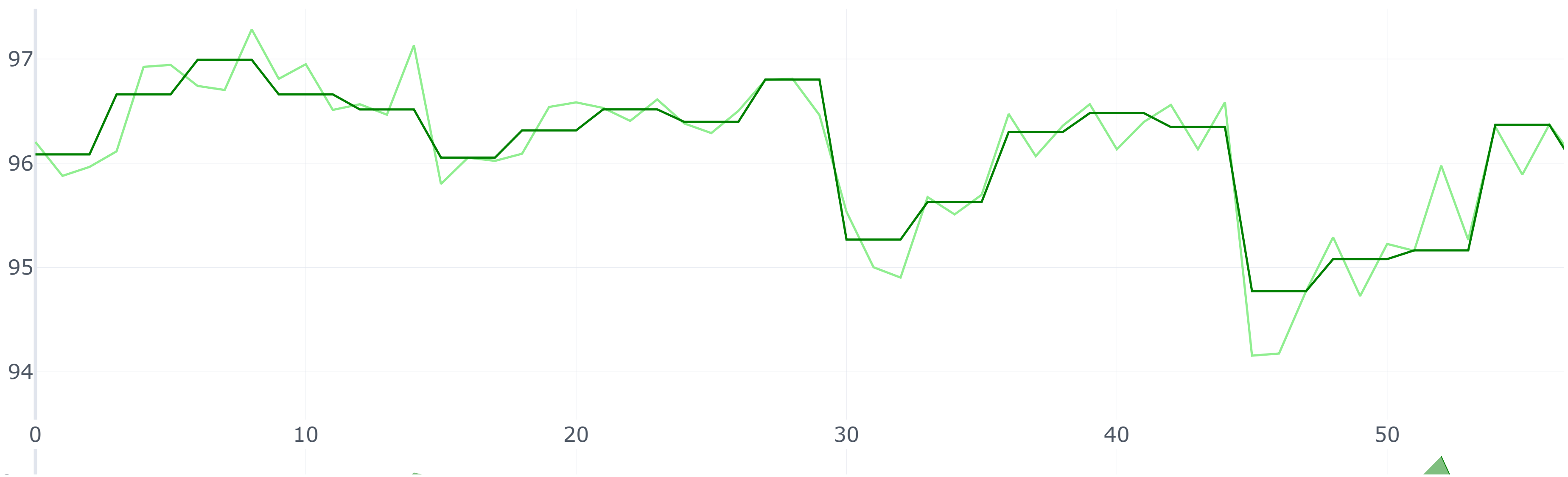
基于XGBoost回归的预测模型



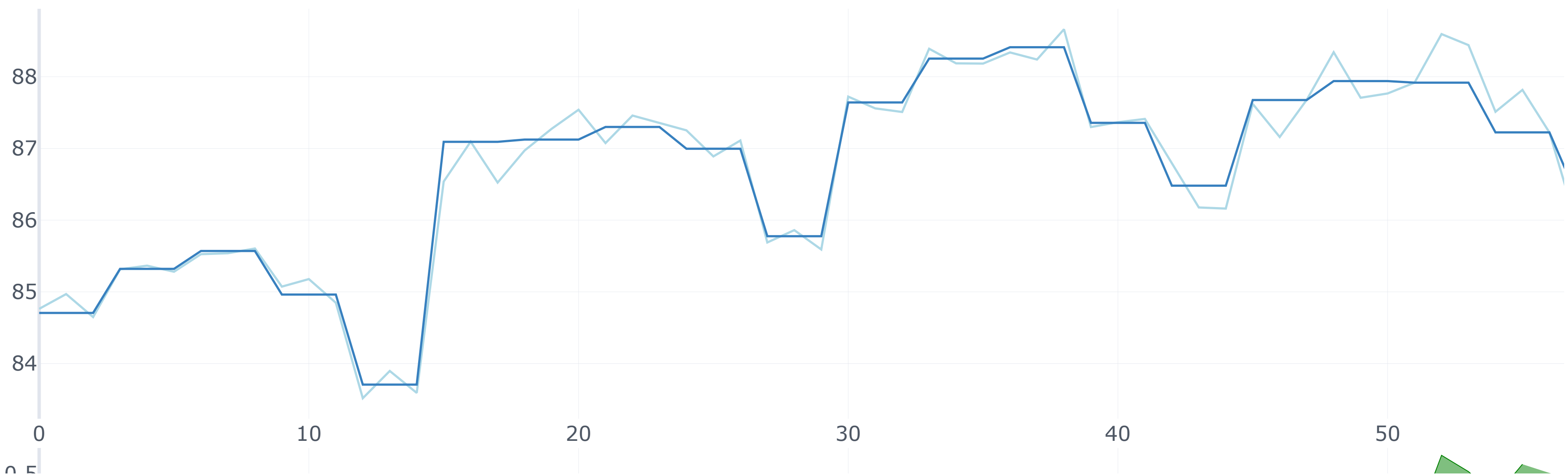
基于XGBoost回归的预测模型——厚度mm



基于XGBoost回归的预测模型——孔隙率（%）



基于XGBoost回归的预测模型——压缩回弹性率（%）



```
Out[14]: array([[[ 2.76130402,  2.39697003,  2.77314377,  2.40274191,
                  3.15791488,  2.77314377,  2.39697003,  1.98269733],
                [96.31589127, 96.51655833, 96.37236786, 95.9524231 ,
                 96.80519867, 96.39583969, 95.08143107, 94.77352142],
                [84.70593262, 87.12225342, 86.48152542, 85.13269043,
                 83.70683289, 85.77477264, 88.2532196 , 86.91458639]],

               [[ 2.7823657 ,  2.71739246,  2.7690421 ,  2.59206242,
                  3.50957404,  3.08188874,  2.38730479,  1.90849573],
                [96.14445255, 96.32734538, 96.45573221, 96.01032563,
                 96.54976737, 96.43939534, 95.66271658, 94.62646967],
                [84.76908995, 87.14407931, 87.36721671, 86.32856071,
                 83.96126417, 87.01517197, 88.23864825, 87.66418777]],

               [[ 3.0490427 ,  2.92080164,  2.85151792,  2.61077833,
                  3.5192771 ,  3.1576283 ,  2.61824155,  2.16034007],
                [96.66072083, 96.51689911, 96.34745789, 95.95244598,
                 96.51647186, 96.80378723, 96.30002594, 95.08094788],
                [85.32053375, 87.29962158, 86.48087311, 86.35616302,
                 83.70688629, 85.77590942, 88.41117096, 87.93922424]]])
```

In [ ]:

In [15]:
```python
# TODO save answer
writer = pd.ExcelWriter('question2-answer.xlsx')
sheet_names = ["极限树回归", "随机森林回归", "XGBoost回归"]
for i in range(3):
    data_temp = pd.DataFrame(all_models_ypreds_q2[i], index=list(XY.columns)[-3:]).T
    data_temp.to_excel(writer, sheet_names[i])
writer.save()
writer.close()
```

In [ ]:

问题3

分析关系

In [16]:
```python
from hmz.math_model.evaluate import PP, PPR
```

In [ ]:

In [17]:
```
XY = copy(data3.iloc[:, 2:])
X1 = XY.iloc[:, :3]
X2= XY.iloc[:, 3:]

pp = PP(X1)
pp.forward(
    columns=[0, 1],
    process_type=[1, 1],
    process_values=[None, None],
)
weight, _ = pp.weight()
print(weight)  # [ 0.02938399  0.99932456 -0.02206817]

pp = PP(X2)
pp.forward(
    columns=[0, 1],
    process_type=[1, 1],
    process_values=[None, None],
)
weight, _ = pp.weight()
print(weight)  # [-0.66214501 -0.16667322  0.73060524]
```

```
0
正向化处理后的矩阵:
       厚度mm     孔隙率（%）   压缩回弹性率（%）
0   2.76601  96.204430  84.763268
1   2.75493  95.879204  84.969391
2   2.75659  95.965271  84.648598
3   3.05117  96.114388  85.315208
4   3.05148  96.924713  85.365067
..      ...        ...        ...
70  2.32876  95.012779  86.629562
71  2.32818  95.307404  85.896751
72  2.42207  96.046860  85.250839
73  2.40229  95.588753  85.144913
74  2.40319  95.913101  85.120468

[75 rows x 3 columns]
标准化处理后的矩阵:
       厚度mm   孔隙率（%）   压缩回弹性率（%）
0   0.12057  0.11586    0.11300
1   0.12009  0.11547    0.11327
2   0.12016  0.11557    0.11285
3   0.13300  0.11575    0.11374
4   0.13302  0.11673    0.11380
..      ...      ...        ...
70  0.10151  0.11442    0.11549
71  0.10149  0.11478    0.11451
72  0.10558  0.11567    0.11365
73  0.10472  0.11512    0.11351
74  0.10476  0.11551    0.11348

[75 rows x 3 columns]
```

Out[17]:

	厚度mm	孔隙率（%）	压缩回弹性率（%）
0	0.120574	0.115858	0.113000
1	0.120091	0.115466	0.113274
2	0.120163	0.115570	0.112847
3	0.133004	0.115749	0.113736
4	0.133017	0.116725	0.113802
...	...	...	...
70	0.101513	0.114423	0.115488
71	0.101488	0.114777	0.114511
72	0.105581	0.115668	0.113650
73	0.104718	0.115116	0.113508
74	0.104758	0.115507	0.113476

```
[ 0.83322882  0.5446984  -0.09504405]
0
正向化处理后的矩阵:
       过滤阻力Pa     过滤效率（%）     透气性 mm/s
0   24.756050  48.211700  654.149170
1   26.008209  51.443321  609.854614
2   26.222300  49.679939  635.112183
3   27.167860  48.599380  521.793030
4   25.323790  45.963821  553.848633
..        ...        ...         ...
70  28.862761  80.012337  206.124680
71  30.559219  81.993637  206.336365
72  28.296640  83.095757  209.181519
73  27.499001  83.201111  208.566986
74  29.158541  82.628143  210.166260

[75 rows x 3 columns]
标准化处理后的矩阵:
     过滤阻力Pa  过滤效率（%）   透气性 mm/s
0   0.09770  0.10481   0.17094
1   0.10264  0.11183   0.15936
2   0.10348  0.10800   0.16596
3   0.10722  0.10565   0.13635
4   0.09994  0.09992   0.14473
..      ...      ...       ...
70  0.11391  0.17394   0.05386
71  0.12060  0.17824   0.05392
72  0.11167  0.18064   0.05466
73  0.10852  0.18087   0.05450
74  0.11507  0.17962   0.05492

[75 rows x 3 columns]
```

Out[17]:

	过滤阻力Pa	过滤效率（%）	透气性 mm/s
0	0.097698	0.104806	0.170938
1	0.102640	0.111831	0.159364
2	0.103485	0.107998	0.165964
3	0.107217	0.105649	0.136352
4	0.099939	0.099920	0.144728
...	...	...	...
70	0.113905	0.173937	0.053863
71	0.120600	0.178244	0.053919
72	0.111671	0.180640	0.054662
73	0.108523	0.180869	0.054501
74	0.115073	0.179623	0.054919

```
[-0.29524646 -0.75424891  0.58646237]
```

In [ ]:

选择模型

In [18]:

```
from sklearn.linear_model import LinearRegression       # 线性回归
from sklearn.linear_model import Lasso, LassoCV         # 岭回归
from sklearn.linear_model import Ridge, RidgeCV         # LASSO回归
from sklearn.neighbors import KNeighborsRegressor       # KNN回归
from sklearn.neural_network import MLPRegressor         # 多层感知机回归
from sklearn.svm import SVR                             # SVM回归
from sklearn.tree import DecisionTreeRegressor          # 决策树回归
from sklearn.tree import ExtraTreeRegressor             # 极限树回归
from sklearn.ensemble import RandomForestRegressor      # 随机森林回归
from sklearn.ensemble import AdaBoostRegressor          # AdaBoost回归
from sklearn.ensemble import GradientBoostingRegressor  # 梯度提升回归
from sklearn.ensemble import BaggingRegressor           # Bagging回归

from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor

from catboost import CatBoostRegressor
from xgboost import XGBRegressor, XGBRFRegressor
```

```python
from lightgbm import LGBMRegressor

from hmz.math_model.predict import predict_accuracy
```

In [ ]:

In [19]:

```python
XY = copy(data3.iloc[:, 2:])
X = np.array(XY.iloc[:, :3])
Y = np.array(XY.iloc[:, 3:])

colors = ["red", "lightpink", "darkorange", "khaki", "green", "lightgreen", "blue", "lightblue"]
colors = ["red", "lightpink", "green", "lightgreen", "blue", "lightblue"]

xpred = np.array([
    [38, 33, 28, 23, 38, 33, 28, 23],
    [850, 950, 1150, 1250, 1250, 1150, 950, 850],
]).T

def score(model, X, Y, xtrain, xtest, ytrain, ytest, Ypred):
    accuracy = predict_accuracy(Y, Ypred, type=1, th=0.05, con=0.8)
    print("准确率: ", accuracy)
#     print("训练集分数: R^2 =", model.score(xtrain, ytrain))  # 负值: 拟合效果糟糕, 模型完全不能使用
#     print("测试集分数: R^2 =", model.score(xtest, ytest))  # -> 1: 拟合效果很好, 模型可以使用
#     print(r2_score(model.predict(xtrain), ytrain))
#     print(r2_score(model.predict(xtest), ytest))
#     scores = cross_val_score(model, X, Y, cv=10, scoring="r2")
#     print(scores.mean())
    return None

models = [
    LinearRegression(),
    Lasso(),
    Ridge(),
    KNeighborsRegressor(),
    MLPRegressor(),
    SVR(kernel='rbf', degree=100, gamma='auto'),  # 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'; ``'scale', 'auto'
    DecisionTreeRegressor(),
    ExtraTreeRegressor(),
    RandomForestRegressor(),
    AdaBoostRegressor(),
    GradientBoostingRegressor(),
    BaggingRegressor(),
    HistGradientBoostingRegressor(),
    CatBoostRegressor(verbose=0),
    XGBRegressor(),
    XGBRFRegressor(),
```

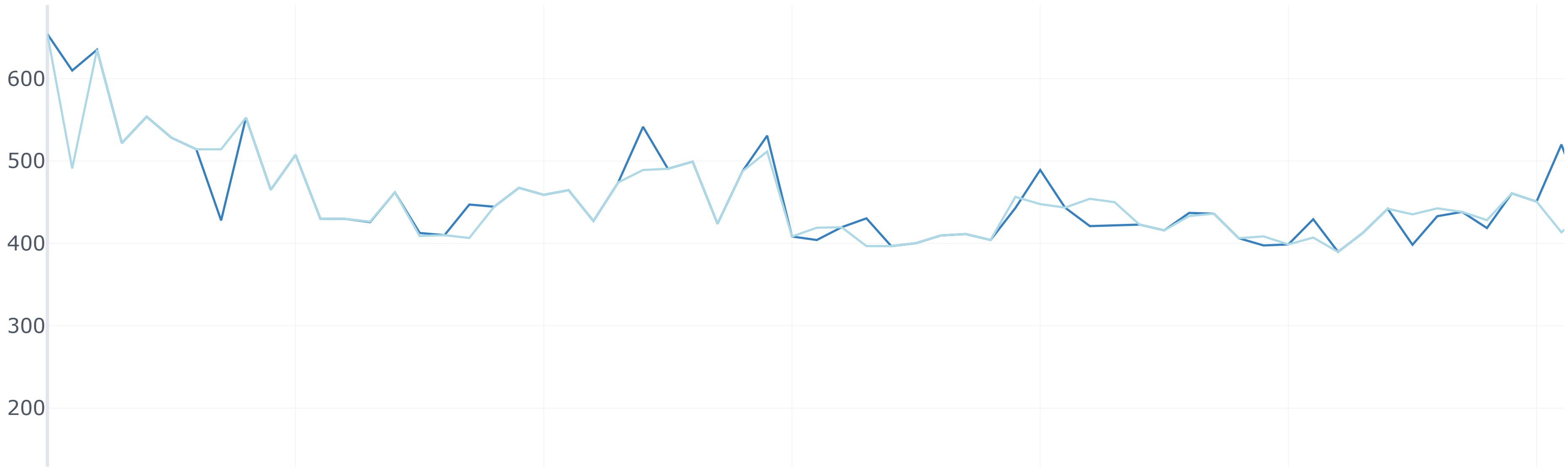
```python
    LGBMRegressor(),
]
models_name = [
    "线性回归",
    "LASSO套索回归",
    "Ridge岭回归",
    "KNN回归",
    "多层感知机回归",
    "SVM回归",
    "决策树回归",
    "极限树回归",
    "随机森林回归",
    "AdaBoost回归",
    "梯度提升回归",
    "Bagging回归",
    "基于直方图的梯度提升回归",
    "CatBoost回归",
    "XGBoost回归",
    "XGBoostRF回归",
    "LightGBM回归",
]
models_select = {
    "线性回归": False,
    "LASSO套索回归": False,
    "Ridge岭回归": False,
    "KNN回归": False,
    "多层感知机回归": False,
    "SVM回归": False,
    "决策树回归": False,
    "极限树回归": False,
    "随机森林回归": False,
    "AdaBoost回归": False,
    "梯度提升回归": False,
    "Bagging回归": False,
    "基于直方图的梯度提升回归": False,
    "CatBoost回归": False,
    "XGBoost回归": True,
    "XGBoostRF回归": False,
    "LightGBM回归": False,
}
def run_Regressor_models(
    x=xpred, X=X, Y=Y, colors=colors, plot_fig=False, save_fig=False,
    models=models, models_name=models_name, models_select=models_select,
    save_model=False,
):
    """
```

```
    :param x: np.ndarray
    :param X: np.ndarray
    :param Y: np.ndarray
    """
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3)
    ypreds_s = []
    for model, model_name, model_select in zip(models, models_name, models_select.keys()):
        if not models_select[model_select]:
            continue
        print("模型：", model_name)
        ytests = []
        ypreds = []
        for i in range(Y.shape[1]):
            # 训练
            model.fit(xtrain, ytrain[:, i])
            # 预测
            ytest_ = model.predict(X)
            score(model, X, Y[:, i], xtrain, xtest, ytrain[:, i], ytest[:, i], ytest_)
            ytests.append(list(ytest_))
#             # 预测
#             ypred = model.predict(x)
#             ypreds.append(list(ypred))
            # 保存
            if i == 0:
                joblib.dump(model, './saved_model/suc0_xgb.pkl')
            elif i == 1:
                joblib.dump(model, './saved_model/suc1_xgb.pkl')
            elif i == 2:
                joblib.dump(model, './saved_model/suc2_xgb.pkl')
#         ypreds_s.append(ypreds)

        if plot_fig:
            columns = list(XY.columns[-3:])
            ytests = pd.DataFrame(ytests, index=columns).T
            ytests.columns = map(lambda x: x + '-预测值', ytests.columns)
            ytesttemp = pd.DataFrame(Y, columns=columns)
            ytesttemp.columns = map(lambda x: x + '-真实值', ytesttemp.columns)
            yplot = pd.concat([ytesttemp, ytests], axis=1)

            # 画图1
            title = '基于' + model_name + '的预测模型'
            yplot.iplot(
                color=colors[0::2] + colors[1::2],
                title=title,
            )
            if save_fig:
```

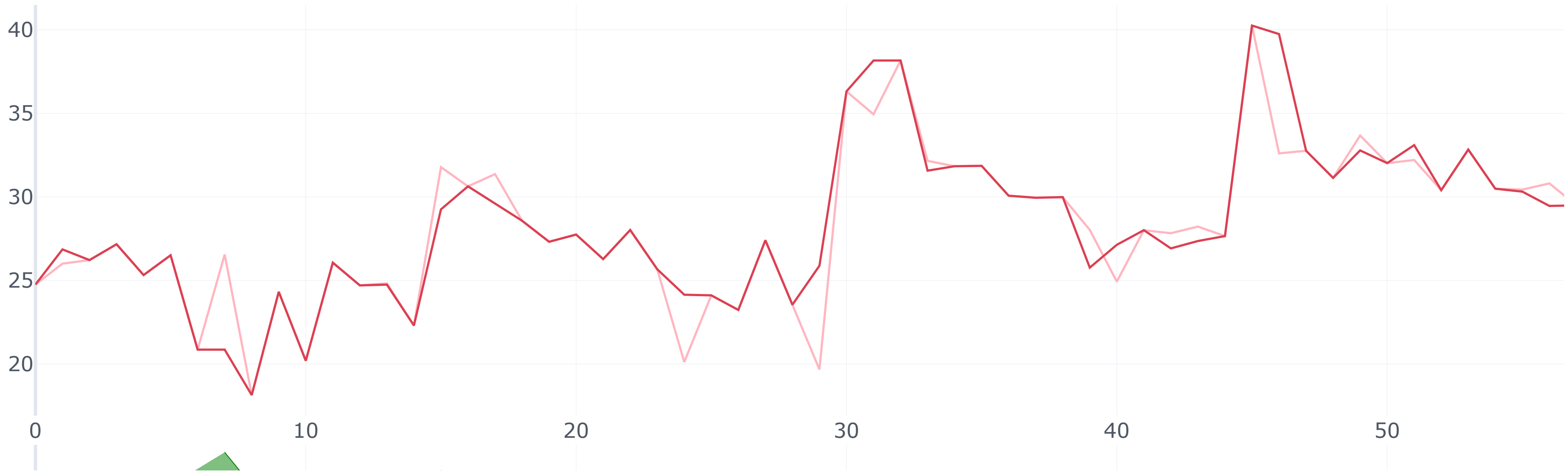
```python
                yplot.figure(
                    color=colors[0::2] + colors[1::2],
                    title='基于' + model_name + '的预测模型',
                ).write_image("./img/问题3-" + title + ".svg")
            # 画图2
            yplot.columns = ['真实值'] * 3 + ['预测值'] * 3
            columns = list(XY.columns[-3:])
            columns[-1] = '透气性 mm/s'
            for i in range(3):
                title = '基于' + model_name + '的预测模型——' + columns[i]
                yplot.iloc[:, i::3].iplot(
                    kind='spread',
                    color=[colors[i * 2 + 1], colors[i * 2]],
                    title=title,
                )
                if save_fig:
                    yplot.iloc[:, i::3].figure(
                        kind='spread',
                        color=[colors[i * 2 + 1], colors[i * 2]],
                        title=title,
                    ).write_image("./img/问题3-" + title + ".svg")
        print('-'*125)
    return ypreds_s
all_models_ypreds_q3 = np.array(run_Regressor_models(plot_fig=True, save_fig=True, save_model=True))
all_models_ypreds_q3
```

```
模型： XGBoost回归
准确率： 0.8535077590545018
准确率： 0.8858960922559103
准确率： 0.8548131998678048
```

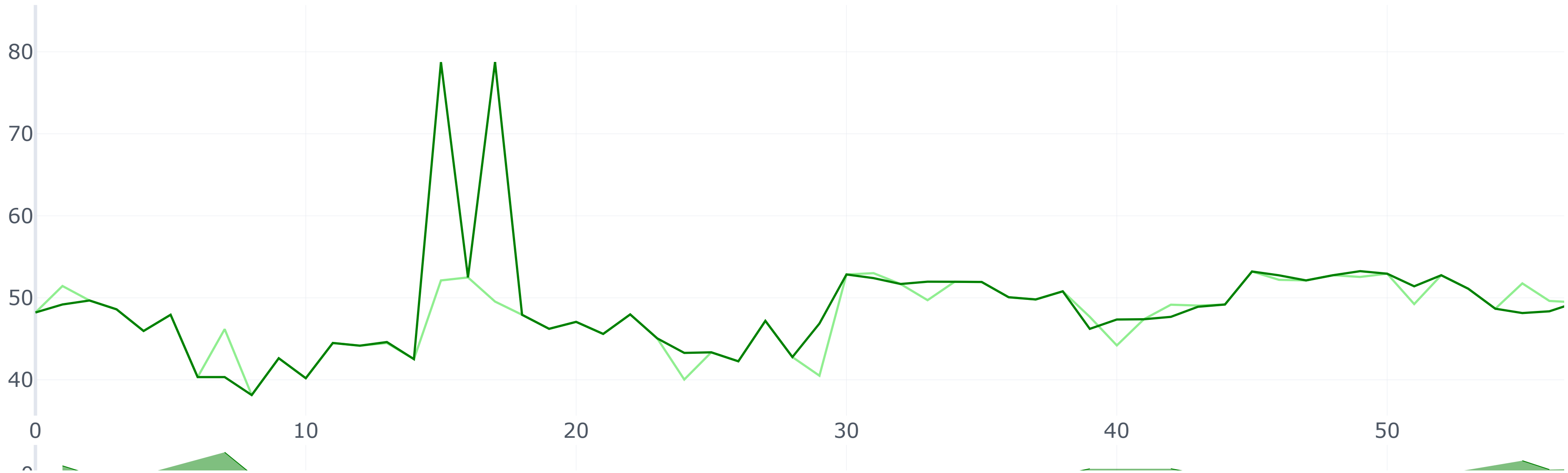
基于XGBoost回归的预测模型

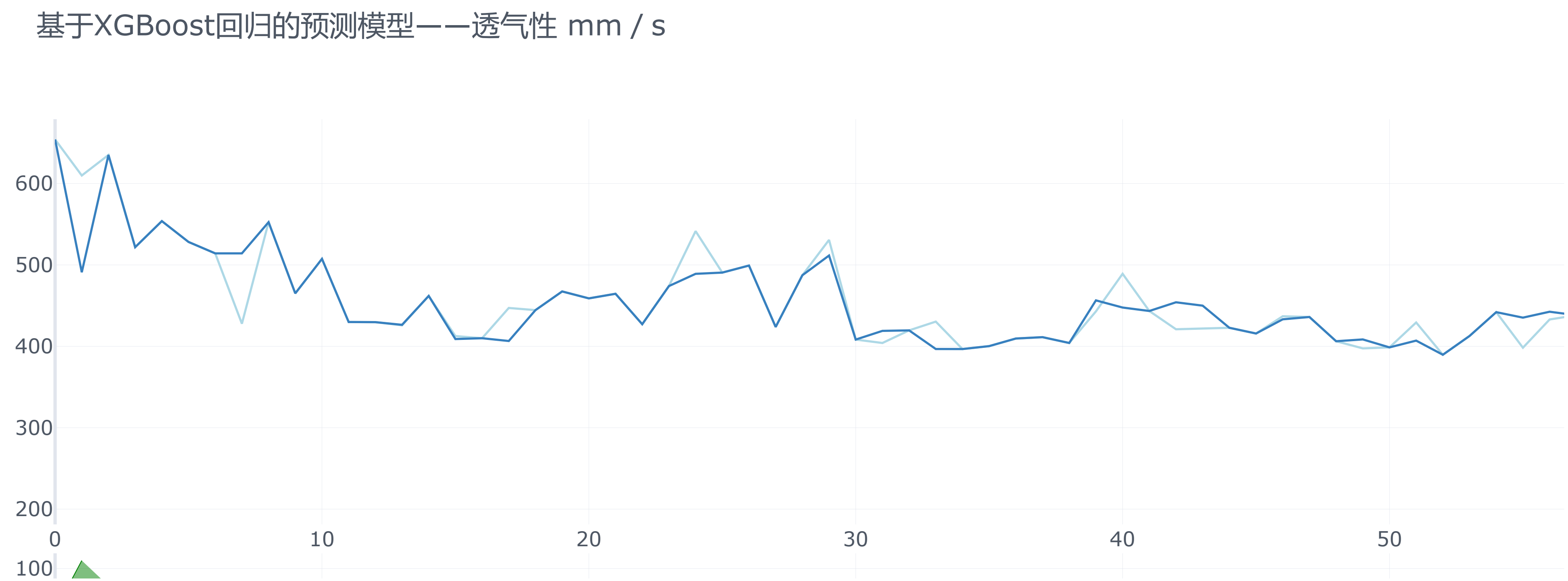


基于XGBoost回归的预测模型——过滤阻力Pa



基于XGBoost回归的预测模型——过滤效率（%）






```
-----------------------------------------------------------------------------------------------------------------------
```

Out[19]: array([], dtype=float64)

In [ ]:

规划求解

In [ ]:

In [20]:
```
from numba import jit
from sko.GA import GA
from time import time
from scipy.optimize import minimize
```

```python
from hmz.kinky_tricks.kinky_tricks import FORE

jieshoujuli_min, refengsudu_min = data3.min()[:2]
jieshoujuli_max, refengsudu_max = data3.max()[:2]
best_model1 = XGBRegressor()
best_model2 = XGBRegressor()

model_pre1, model_pre2, model_pre3 = joblib.load('./saved_model/pre0_xgb.pkl', 'r'), \
    joblib.load('./saved_model/pre1_et.pkl', 'r'), joblib.load('./saved_model/pre2_rf.pkl', 'r')
model_suc1, model_suc2, model_suc3 = joblib.load('./saved_model/suc0_xgb.pkl'), \
    joblib.load('./saved_model/suc1_xgb.pkl', 'r'), joblib.load('./saved_model/suc2_xgb.pkl', 'r')
model1s = [model_pre1, model_pre2, model_pre3]
model2s = [model_suc1, model_suc2, model_suc3]

def run_2part_models(
    X, model1s=model1s, model2s=model2s,
#     colors=colors, plot_fig=False, save_fig=False,
#     models=models, models_name=models_name, models_select=models_select,
):
    """
    :param model1s: 第一部分的模型
    :param model2s: 第二部分的模型
    :param X: np.ndarray (n, 2) 输入数据
    :return: Y np.ndarray (n, 3) 输出数据
    """
    YasX = []
    for model1 in model1s:
        if isinstance(model1, XGBRegressor):
            y1pred = model1.predict(X, validate_features=False)
        else:
            y1pred = model1.predict(X)
        YasX.append(list(y1pred))
    YasX = np.squeeze(YasX)[None, :]  # (n, 3)

    Y = []
    for model2 in model2s:
        if isinstance(model2, XGBRegressor):
            y2pred = model2.predict(YasX, validate_features=False)
        else:
            y2pred = model2.predict(YasX)
        Y.append(list(y2pred))
    Y = np.squeeze(Y)[None, :]  # (n, 3)

    return Y

def run_2part_models_func(dis, spd):
```

```python
    X = np.array([[dis, spd]])  # (n, 2)
    Y = run_2part_models(X)  # (n, 3)
    y = Y[0, 1]
#     print(Y)  # 过滤阻力Pa    过滤效率（%）    透气性 mm/s
#     print(y)  # 过滤效率（%）
#     print(X, round(y, 10))  # check
    return -y

def run_2part_models_fun(x):
    """sko -> scipy"""
    return run_2part_models_func(x[0], x[1])

# @jit
def select_2part_models(type_=0):
    """
    :param type_: 规划类型
        0: sko
        1: scipy
        2: gurobi
        3: linspace
    """
    xbest, ybest = None, None
    if type_ == 0:
        ga = GA(
            run_2part_models_func,
            n_dim=2,  # 接收距离(cm), 热风速度(r/min)
            size_pop=50,
            max_iter=10,
            lb=[jieshoujuli_min, refengsudu_min],
            ub=[jieshoujuli_max, refengsudu_max],
#             constraint_ueq=(lambda x: abs(f((inverter_num, x[0], x[1], x[2])) - NEED) - EPS, ),
        )
        xbest, ybest = ga.run()
    elif type_ == 1:
        opt = minimize(
            run_2part_models_fun,
            x0=[25., 1100.],
        #     x0=[gate_wide_max, gate_wide_max, gate_len_max - 0.1],
            bounds=((jieshoujuli_min, jieshoujuli_max), (refengsudu_min, refengsudu_max)),
            method='trust-constr',  #  SLSQP trust-constr
            options={'gtol': 1e-9, 'disp': True},
        )
        xbest = opt.x
        ybest = run_2part_models_fun(xbest)
    elif type_ == 2:
        ...
```

```python
    elif type_ == 3:
        def step_by_epoch(x1l, x1h, x2l, x2h, num=100):
            x1 = np.linspace(x1l, x1h, num)
            x2 = np.linspace(x2l, x2h, num)
            xbest, ybest = [x1l, x2l], 0
            for x1_ in x1:
                for x2_ in x2:
                    yres = -run_2part_models_func(x1_, x2_)
                    if yres > ybest:
                        xbest = [x1_, x2_]
                        ybest = yres
            return xbest, ybest
        xbest, ybest = [0, 0], 0
        epoch = 10
        x1l, x1h, x2l, x2h = jieshoujuli_min, jieshoujuli_max, refengsudu_min, refengsudu_max
        for i in range(epoch):
            xbest_, ybest_ = step_by_epoch(x1l, x1h, x2l, x2h)
            if ybest_ > ybest:
                xbest = xbest_
                ybest = ybest_
            x1, x2 = xbest_
            x1l = max(jieshoujuli_min, 0.5 * x1)
            x1h = min(jieshoujuli_max, 1.5 * x1)
            x2l = max(refengsudu_min, 0.5 * x2)
            x2h = min(refengsudu_max, 1.5 * x2)
        with FORE:
            print('-'*125)
    return xbest, ybest

# y = run_2part_models(np.array([[50, 800]]))
# y

# y = run_2part_models_func(50, 800)
# y

t0 = time()
xbest, ybest = select_2part_models(type_=0)  # sko
# xbest, ybest = select_2part_models(type_=1)  # scipy
# xbest, ybest = select_2part_models(type_=2)  # gurobi
# xbest, ybest = select_2part_models(type_=3)  # linspace
print("time:", time() - t0)
xbest, ybest
```

```
time: 27.857622385025024
```

Out[20]:

```
(array([  21.93175168, 1196.89991651]), array([-82.94813], dtype=float32))
```

In [ ]:

In [21]:

```
# check
y = -run_2part_models_func(20.2020202, 1151.51515152)
```

过滤效率曲面图

In [ ]:

In [22]:

```
from numba import jit
from time import time

InteractiveShell.ast_node_interactivity = 'last'

def run_2part_models_func(dis, spd, pprint=False):
    X = np.array([[dis, spd]])  # (n, 2)
    Y = run_2part_models(X)  # (n, 3)
    y = Y[0, 1]
    if pprint:
        print(Y)  # 过滤阻力Pa  过滤效率（%）    透气性 mm/s
        print(y)  # 过滤效率（%）
        print(X, round(y, 10))  # check
    return -y

# @jit
def surface_data(
    step1=0.2, step2=4,
    x1l=jieshoujuli_min,
    x1h=jieshoujuli_max,
    x2l=refengsudu_min,
    x2h=refengsudu_max
):
    x1 = np.arange(x1l, x1h, step1)
    x2 = np.arange(x2l, x2h, step2)
    z = []
    len1 = len(x1)
    len2 = len(x2)
    for idx, i in enumerate(x1):
        z_ = []
        for j in x2:
            z_.append(-run_2part_models_func(i, j))
        z.append(z_)
        if idx % 1 == 0:
            with FORE(c='RED'):
```

```python
                print('-'*59, idx, '/', len1, '-'*59)
    z = pd.DataFrame(z, index=list(x1), columns=list(x2))
    return z

t0 = time()
Z = surface_data(step1=1, step2=20)
print("time:", time() - t0)
```

```
----------------------------------------------------------- 0 / 20 -----------------------------------------------------------
----------------------------------------------------------- 1 / 20 -----------------------------------------------------------
----------------------------------------------------------- 2 / 20 -----------------------------------------------------------
----------------------------------------------------------- 3 / 20 -----------------------------------------------------------
----------------------------------------------------------- 4 / 20 -----------------------------------------------------------
----------------------------------------------------------- 5 / 20 -----------------------------------------------------------
----------------------------------------------------------- 6 / 20 -----------------------------------------------------------
----------------------------------------------------------- 7 / 20 -----------------------------------------------------------
----------------------------------------------------------- 8 / 20 -----------------------------------------------------------
----------------------------------------------------------- 9 / 20 -----------------------------------------------------------
----------------------------------------------------------- 10 / 20 -----------------------------------------------------------
----------------------------------------------------------- 11 / 20 -----------------------------------------------------------
----------------------------------------------------------- 12 / 20 -----------------------------------------------------------
----------------------------------------------------------- 13 / 20 -----------------------------------------------------------
----------------------------------------------------------- 14 / 20 -----------------------------------------------------------
----------------------------------------------------------- 15 / 20 -----------------------------------------------------------
----------------------------------------------------------- 16 / 20 -----------------------------------------------------------
----------------------------------------------------------- 17 / 20 -----------------------------------------------------------
----------------------------------------------------------- 18 / 20 -----------------------------------------------------------
----------------------------------------------------------- 19 / 20 -----------------------------------------------------------
time: 25.59749984741211
```

In [23]:

```python
InteractiveShell.ast_node_interactivity = 'last'

fig = go.Figure(
    data=[go.Surface(
        x=list(Z.columns),
        y=list(Z.index),
        z=Z.values,
    )],
)
fig.update_layout(
    title='过滤效率',
    autosize=False,
    width=800, height=800,
    xaxis={"title": "接收距离(cm)"},
    yaxis={"title": "热风速度(r/min)"},
)
fig.write_image('./img/问题3-过滤效率曲面图.svg')
```

```python
fig.write_html('./问题3-过滤效率曲面图.html')
# fig.show()
```

3D曲线图从 `.ipynb` 文件导出 `.html` 文件有问题，建议到以下网址查看该图像 问题3-过滤效率曲面图.html，或者直接打开 `.ipynb` 文件，取消注释 `fig.show()` 运行查看

问题4

接收距离不大可能大于100cm，热空气速度也不大可能大于2000 r/min

厚度尽量不要超过3mm，压缩回弹性率尽量不要低于85%

追求过滤效率高和过滤阻力小的目标的**工艺参数**

In [24]:
```python
from hmz.math_model.process import forward_procession
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler, StandardScaler
```

In [25]:
```python
# alpha * 过滤效率, beta * 过滤阻力
```

前部分模型

In [ ]:
```python

```

In [26]:
```python
# from sklearn.preprocessing import MinMaxScaler

XY = copy(data3.iloc[:, :5])
X = np.array(XY.iloc[:, :2])
Y = np.array(XY.iloc[:, 2:])

# mms = MinMaxScaler()
# X = mms.fit_transform(X)
# Y = mms.fit_transform(Y)

colors = ["red", "lightpink", "darkorange", "khaki", "green", "lightgreen", "blue", "lightblue"]
colors = ["red", "lightpink", "green", "lightgreen", "blue", "lightblue"]

xpred = np.array([
    [38, 33, 28, 23, 38, 33, 28, 23],
    [850, 950, 1150, 1250, 1250, 1150, 950, 850],
]).T
```

```python
def score(model, X, Y, xtrain, xtest, ytrain, ytest, Ypred):
    accuracy = predict_accuracy(Y, Ypred, type=1, th=0.005, con=0.8)
    print("准确率：", accuracy)
#     print("训练集分数：R^2 =", model.score(xtrain, ytrain))  # 负值：拟合效果糟糕，模型完全不能使用
#     print("测试集分数：R^2 =", model.score(xtest, ytest))  # -> 1: 拟合效果很好，模型可以使用
#     print(r2_score(model.predict(xtrain), ytrain))
#     print(r2_score(model.predict(xtest), ytest))
#     scores = cross_val_score(model, X, Y, cv=10, scoring="r2")
#     print(scores.mean())
    return None

models = [
    LinearRegression(),
    Lasso(),
    Ridge(),
    KNeighborsRegressor(),
    MLPRegressor(),
    SVR(kernel='rbf', degree=200, gamma='auto'),  # 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'; ``'scale', 'auto'
    DecisionTreeRegressor(),
    ExtraTreeRegressor(),
    RandomForestRegressor(),
    AdaBoostRegressor(),
    GradientBoostingRegressor(),
    BaggingRegressor(),
    HistGradientBoostingRegressor(),
    CatBoostRegressor(verbose=0),
    XGBRegressor(),
    XGBRFRegressor(),
    LGBMRegressor(),
]
models_name = [
    "线性回归",
    "LASSO套索回归",
    "Ridge岭回归",
    "KNN回归",
    "多层感知机回归",
    "SVM回归",
    "决策树回归",
    "极限树回归",
    "随机森林回归",
    "AdaBoost回归",
    "梯度提升回归",
    "Bagging回归",
    "基于直方图的梯度提升回归",
    "CatBoost回归",
    "XGBoost回归",
```

```python
    "XGBoostRF回归",
    "LightGBM回归",
]
models_select = {
    "线性回归": False,
    "LASSO套索回归": False,
    "Ridge岭回归": False,
    "KNN回归": False,
    "多层感知机回归": False,
    "SVM回归": False,
    "决策树回归": False,
    "极限树回归": True,  # 1
    "随机森林回归": True,  # 2
    "AdaBoost回归": False,
    "梯度提升回归": False,
    "Bagging回归": False,
    "基于直方图的梯度提升回归": False,
    "CatBoost回归": False,
    "XGBoost回归": True,  # 0
    "XGBoostRF回归": False,
    "LightGBM回归": False,
}
saved_models_name = ['xgb', 'et', 'rf']

def run_Regressor_models(
    x=xpred, X=X, Y=Y, colors=colors, plot_fig=False, save_fig=False,
    models=models, models_name=models_name, models_select=models_select,
    saved_models_name=saved_models_name, save_model=False,
):
    """
    :param x: np.ndarray
    :param X: np.ndarray
    :param Y: np.ndarray
    XGBoost(), ET(), RF()
    """
    xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.3)
    ypreds_s = []
    for model, model_name, model_idx in zip(models, models_name, range(len(models_name))):
#         print(models_select[model_name])
        if not models_select[model_name]:
            continue
        print("模型: ", model_name)
        ypreds = []
        ytests = []
        for i in range(Y.shape[1]):
            # 训练
```

```
        model.fit(xtrain, ytrain[:, i])
        # 测试
        ytest_ = model.predict(X)
        score(model, X, Y[:, i], xtrain, xtest, ytrain[:, i], ytest[:, i], ytest_)
        ytests.append(list(ytest_))
        # 预测
        ypred = model.predict(x)
        ypreds.append(list(ypred))
        # 保存
        if save_model:
            if model_name == "XGBoost回归" and i == 0:
                joblib.dump(model, './saved_model/pre0_xgb.pkl')
            elif model_name == "极限树回归" and i == 1:
                joblib.dump(model, './saved_model/pre1_et.pkl')
            elif model_name == "随机森林回归" and i == 2:
                joblib.dump(model, './saved_model/pre2_rf.pkl')
    ypreds_s.append(ypreds)

    if plot_fig:
        # 画图
        ytests = pd.DataFrame(ytests, index=list(XY.columns[2:5])).T
        ytests.columns = map(lambda x: x + '-预测值', ytests.columns)
        ytesttemp = pd.DataFrame(Y, columns=list(XY.columns[2:5]))
        ytesttemp.columns = map(lambda x: x + '-真实值', ytesttemp.columns)
        yplot = pd.concat([ytesttemp, ytests], axis=1)
#         print(yplot.shape)
#         print(yplot.head())
#         print(ypreds.shape)
#         print(ypreds.head())
#         print(ytest_temp.shape)
#         print(ytest_temp.head())

        title = '基于' + model_name + '的预测模型'
        yplot.iplot(
            color=colors[0::2] + colors[1::2],
            title=title,
        )
        if save_fig:
            yplot.figure(
                color=colors[0::2] + colors[1::2],
                title=title,
            ).write_image("./img/问题2-" + title + ".svg")

        yplot.columns = ['真实值'] * 3 + ['预测值'] * 3
        columns = list(XY.columns[2:5])
        for i in range(3):
```

```python
                title = '基于' + model_name + '的预测模型——' + columns[i]
                yplot.iloc[:, i::3].iplot(
                    kind='spread',
                    color=[colors[i * 2 + 1], colors[i * 2]],
                    title=title,
                )
                if save_fig:
                    yplot.iloc[:, i::3].figure(
                        kind='spread',
                        color=[colors[i * 2 + 1], colors[i * 2]],
                        title=title,
                    ).write_image("./img/问题2-" + title + ".svg")

        print('-'*125)
    return ypreds_s
all_models_ypreds_q41 = np.array(run_Regressor_models(plot_fig=True, save_fig=True, save_model=True))
all_models_ypreds_q41
```

```
模型：  极限树回归
准确率：  0.9783027482740771
准确率：  0.967596518288745
准确率：  0.935611197528982
```

模型： 随机森林回归
准确率： 0.6035103178338299
准确率： 0.9569203921292283
准确率： 0.9142168641267546

```
----------------------------------------------------------------------------------------------------------------------------------
模型： XGBoost回归
准确率： 0.9782985855635877
准确率： 0.9675964901521802
准确率： 0.9462774250066528
```