登革热疾病传播的回归预测分析

摘要

项目来源：来自于 DRIVENDATA 平台上正在进行的竞赛项目

项目主办方： DRIVENDATA

竞赛目标： 根据描述温度、降水、植被等变化的环境变量来预测每周（每个位置）的登革热病例数；进而实现预测下一次的登革热疾病大流行。

竞赛数据： 本次比赛的数据来自多个来源，登革热监测数据由美国疾病控制和预防中心以及国防部海军医学研究第 6 单位和武装部队健康监测中心与秘鲁政府和美国大学合作提供。环境和气候数据由美国商务部下属机构国家海洋和大气管理局 (NOAA) 提供。

目标变量：数据集当中的 total_cases 变量。

特征变量：

features	explain	example
week_start_date	以 yyyy-mm-dd 格式给出的日期	1994-05-07
total_cases	目标变量	22
station_max_temp_c	最高温度	33.3
station_avg_temp_c	平均温度	27.7571428571
station_precip_mm	总降水量	10.5
station_min_temp_c	最低温度	22.8
station_diur_temp_rng_c	昼夜温差	7.7
precipitation_amt_mm	总降水量	68.0
reanalysis_sat_precip_amt_mm	总降水量	68.0
reanalysis_dew_point_temp_k	平均露点温度	295.235714286
reanalysis_air_temp_k	气温	298.927142857
reanalysis_relative_humidity_percent	平均相对湿度	80.3528571429
reanalysis_specific_humidity_g_per_kg	平均比湿	16.6214285714
reanalysis_precip_amt_kg_per_m2	总降水量	14.1
reanalysis_max_air_temp_k	最高气温	301.1
reanalysis_min_air_temp_k	最低气温	297.0
reanalysis_avg_temp_k	平均气温	299.092857143
reanalysis_tdtr_k	昼夜温差	2.67142857143
ndvi_se	城市东南的植被指数	0.1644143
ndvi_sw	城市西南的植被指数	0.0652
ndvi_ne	城市东北的植被指数	0.1321429
ndvi_nw	城市西北的植被指数	0.08175

图 1-1 变量情况

任务描述：预测测试集中每个（city,year,weekofyear）的 total_cases 标签。有两个城市，圣胡安（sj）和伊基托斯（iq），每个城市的测试数据分别跨越 5 年和 3 年。将提交一份包含对两个城市的预测的文件。每个城市的数据都与一个城市列相连，该列指示主键：圣胡安（sj）和伊基托斯（iq）。测试集是一个纯粹的未来数据，这意味着测试数据是连续的，与任何训练数据都不重叠。此外，缺失的值都被填充为 NaN。

竞赛评价指标：本次比赛使用的指标是平均绝对误差。目标是最小化 MAE。

一、提出问题

1. 在数据集中存在相当大一部分缺失值的时候，应该采取某种方式填充还是删除呢？

2. 当特征变量中有多个都是表示同维度的指标的时候应该如何进行特征选择呢？是否可以取出其中一个代表某些指标呢？

3. 两个城市的登革热疾病传播的异同之处，结论是？

二、获取数据

1. 导包

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
matplotlib.rc("font", family='Microsoft YaHei')
plt.rcParams["font.sans-serif"]=['SimHei']
plt.rcParams["axes.unicode_minus"]=False
plt.rcParams.update({'font.size':13 }) #全局字体大小
plt.rcParams["font.family"]=['SimHei']
plt.rcParams['axes.unicode_minus']=False
```

图 2-1 导包

2. 读入数据集

```python
train = pd.read_csv('./data/dengue_features_train.csv',index_col=[0,1,2])
test = pd.read_csv('./data/dengue_features_test.csv',index_col=[0,1,2])
target = pd.read_csv('./data/dengue_labels_train.csv',index_col=[0,1,2])
```

图 2-2 读取数据集

3. 查看数据集

目标变量：

```python
target.describe().T
```

	count	mean	std	min	25%	50%	75%	max
total_cases	1456.0	24.675137	43.596	0.0	5.0	12.0	28.0	461.0

图 2-3 目标变量描述

训练集：

```
train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ndvi_ne	1262.0	0.142294	0.140531	-0.406250	0.044950	0.128817	0.248483	0.508357
ndvi_nw	1404.0	0.130553	0.119999	-0.456100	0.049217	0.121429	0.216600	0.454429
ndvi_se	1434.0	0.203783	0.073860	-0.015533	0.155087	0.196050	0.248846	0.538314
ndvi_sw	1434.0	0.202305	0.083903	-0.063457	0.144209	0.189450	0.246982	0.546017
precipitation_amt_mm	1443.0	45.760388	43.715537	0.000000	9.800000	38.340000	70.235000	390.600000
reanalysis_air_temp_k	1446.0	298.701852	1.362420	294.635714	297.658929	298.646429	299.833571	302.200000
reanalysis_avg_temp_k	1446.0	299.225578	1.261715	294.892857	298.257143	299.289286	300.207143	302.928571
reanalysis_dew_point_temp_k	1446.0	295.246356	1.527810	289.642857	294.118929	295.640714	296.460000	298.450000
reanalysis_max_air_temp_k	1446.0	303.427109	3.234601	297.800000	301.000000	302.400000	305.500000	314.000000
reanalysis_min_air_temp_k	1446.0	295.719156	2.565364	286.900000	293.900000	296.200000	297.900000	299.900000
reanalysis_precip_amt_kg_per_m2	1446.0	40.151819	43.434399	0.000000	13.055000	27.245000	52.200000	570.500000
reanalysis_relative_humidity_percent	1446.0	82.161959	7.153897	57.787143	77.177143	80.301429	86.357857	98.610000
reanalysis_sat_precip_amt_mm	1443.0	45.760388	43.715537	0.000000	9.800000	38.340000	70.235000	390.600000
reanalysis_specific_humidity_g_per_kg	1446.0	16.746427	1.542494	11.715714	15.557143	17.087143	17.978214	20.461429
reanalysis_tdtr_k	1446.0	4.903754	3.546445	1.357143	2.328571	2.857143	7.625000	16.028571
station_avg_temp_c	1413.0	27.185783	1.292347	21.400000	26.300000	27.414286	28.157143	30.800000
station_diur_temp_rng_c	1413.0	8.059328	2.128568	4.528571	6.514286	7.300000	9.566667	15.800000
station_max_temp_c	1436.0	32.452437	1.959318	26.700000	31.100000	32.800000	33.900000	42.200000
station_min_temp_c	1442.0	22.102150	1.574066	14.700000	21.100000	22.200000	23.300000	25.600000
station_precip_mm	1434.0	39.326360	47.455314	0.000000	8.700000	23.850000	53.900000	543.300000

图 2-4 训练集描述

测试集：

```
test.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ndvi_ne	373.0	0.126050	0.164353	-0.463400	-0.001500	0.110100	0.263329	0.500400
ndvi_nw	405.0	0.126803	0.141420	-0.211800	0.015975	0.088700	0.242400	0.649000
ndvi_se	415.0	0.207702	0.079102	0.006200	0.148670	0.204171	0.254871	0.453043
ndvi_sw	415.0	0.201721	0.092028	-0.014671	0.134079	0.186471	0.253243	0.529043
precipitation_amt_mm	414.0	38.354324	35.171126	0.000000	8.175000	31.455000	57.772500	169.340000
reanalysis_air_temp_k	414.0	298.818295	1.469501	294.554286	297.751429	298.547143	300.240357	301.935714
reanalysis_avg_temp_k	414.0	299.353071	1.306233	295.235714	298.323214	299.328571	300.521429	303.328571
reanalysis_dew_point_temp_k	414.0	295.419179	1.523099	290.818571	294.335714	295.825000	296.643571	297.794286
reanalysis_max_air_temp_k	414.0	303.623430	3.101817	298.200000	301.425000	302.750000	305.800000	314.100000
reanalysis_min_air_temp_k	414.0	295.743478	2.761109	286.200000	293.500000	296.300000	298.275000	299.700000
reanalysis_precip_amt_kg_per_m2	414.0	42.171135	48.909514	0.000000	9.430000	25.850000	56.475000	301.400000
reanalysis_relative_humidity_percent	414.0	82.499810	7.378243	64.920000	77.397143	80.330000	88.328929	97.982857
reanalysis_sat_precip_amt_mm	414.0	38.354324	35.171126	0.000000	8.175000	31.455000	57.772500	169.340000
reanalysis_specific_humidity_g_per_kg	414.0	16.927088	1.557868	12.537143	15.792857	17.337143	18.174643	19.598571
reanalysis_tdtr_k	414.0	5.124569	3.542870	1.485714	2.446429	2.914286	8.171429	14.485714
station_avg_temp_c	404.0	27.369587	1.232608	24.157143	26.514286	27.483333	28.319048	30.271429
station_diur_temp_rng_c	404.0	7.810991	2.449718	4.042857	5.928571	6.642857	9.812500	14.725000
station_max_temp_c	413.0	32.534625	1.920429	27.200000	31.100000	32.800000	33.900000	38.400000
station_min_temp_c	407.0	22.368550	1.731437	14.200000	21.200000	22.200000	23.300000	26.700000
station_precip_mm	411.0	34.278589	34.655966	0.000000	9.100000	23.600000	47.750000	212.000000

图 2-5 测试集描述

三、数据预处理和探索性分析

数据预处理：

1. 合并数据集（train 和 target）

```python
# 合并数据集
train_ = pd.concat([train,target],axis=1)
train_
```

city	year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	reanalysis_max_air_temp_k	...
sj	1990	18	1990-04-30	0.122600	0.103725	0.198483	0.177617	12.42	297.572857	297.742857	292.414286	299.8	...
		19	1990-05-07	0.169900	0.142175	0.162357	0.155486	22.82	298.211429	298.442857	293.951429	300.9	...
		20	1990-05-14	0.032250	0.172967	0.157200	0.170843	34.54	298.781429	298.878571	295.434286	300.5	...
		21	1990-05-21	0.128633	0.245067	0.227557	0.235886	15.36	298.987143	299.228571	295.310000	301.4	...
		22	1990-05-28	0.196200	0.262200	0.251200	0.247340	7.52	299.518571	299.664286	295.821429	301.9	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...
iq	2010	21	2010-05-28	0.342750	0.318900	0.256343	0.292514	55.30	299.334286	300.771429	296.825714	309.7	...
		22	2010-06-04	0.160157	0.160371	0.136043	0.225657	86.47	298.330000	299.392857	296.452857	308.5	...
		23	2010-06-11	0.247057	0.146057	0.250357	0.233714	58.94	296.598571	297.592857	295.501429	305.5	...
		24	2010-06-18	0.333914	0.245771	0.278886	0.325486	59.67	296.345714	297.521429	295.324286	306.1	...
		25	2010-06-25	0.298186	0.232971	0.274214	0.315757	63.22	298.097143	299.835714	295.807143	307.8	...

1456 rows × 22 columns

图 3-1 合并数据集

2. 拼接训练集和测试集，方便进行后续的数据预处理

```python
# 拼接训练集和测试集，方便进行后续的数据预处理
data = pd.concat([train_,test])
data
```

city	year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	reanalysis_max_air_temp_k	...
sj	1990	18	1990-04-30	0.122600	0.103725	0.198483	0.177617	12.42	297.572857	297.742857	292.414286	299.8	...
		19	1990-05-07	0.169900	0.142175	0.162357	0.155486	22.82	298.211429	298.442857	293.951429	300.9	...
		20	1990-05-14	0.032250	0.172967	0.157200	0.170843	34.54	298.781429	298.878571	295.434286	300.5	...
		21	1990-05-21	0.128633	0.245067	0.227557	0.235886	15.36	298.987143	299.228571	295.310000	301.4	...
		22	1990-05-28	0.196200	0.262200	0.251200	0.247340	7.52	299.518571	299.664286	295.821429	301.9	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...
iq	2013	22	2013-05-28	0.301471	0.380029	0.280629	0.383186	41.12	297.774286	298.964286	295.638571	305.5	...
		23	2013-06-04	0.247600	0.296343	0.285371	0.350357	71.52	297.167143	298.328571	295.845714	306.3	...
		24	2013-06-11	0.238729	0.251029	0.252586	0.249771	78.96	295.831429	296.607143	294.894286	304.6	...
		25	2013-06-18	0.310429	0.302700	0.406614	0.403943	39.54	295.778571	297.400000	293.648571	305.9	...
		26	2013-06-25	0.339467	0.240071	0.356943	0.273600	51.80	297.372857	299.000000	294.615714	307.3	...

1872 rows × 22 columns

图 3-2 拼接训练集和测试集

3. 查看数据缺失值情况

```
data.isnull().sum()
```

```
week_start_date                           0
ndvi_ne                                 237
ndvi_nw                                  63
ndvi_se                                  23
ndvi_sw                                  23
precipitation_amt_mm                     15
reanalysis_air_temp_k                    12
reanalysis_avg_temp_k                    12
reanalysis_dew_point_temp_k              12
reanalysis_max_air_temp_k                12
reanalysis_min_air_temp_k                12
reanalysis_precip_amt_kg_per_m2          12
reanalysis_relative_humidity_percent     12
reanalysis_sat_precip_amt_mm             15
reanalysis_specific_humidity_g_per_kg    12
reanalysis_tdtr_k                        12
station_avg_temp_c                       55
station_diur_temp_rng_c                  55
station_max_temp_c                       23
station_min_temp_c                       23
station_precip_mm                        27
total_cases                             416
dtype: int64
```

图 3-3 缺失值情况

4. 数据清洗

1. 填充数据

对于植被指数相关变量，即'ndvi_...'等四列中缺失的数据，采取利用现有值的均值进行填充

1)计算植被指数均值

```python
# 根据数据集中 未缺失的四个方位的 ndvi_ 数据 计算均值
ndvi_mean = []
for i in range(len(data)):
    ndvi_ = []
    ndvi_ = ndvi_ = data.iloc[i,1:5] # 获取四个方位的 ndvi_ 数据
    null = int(pd.isnull(ndvi_).sum()) # 每行数据为空个数
    if null != 4: # 若某行数据仅缺失某个'ndvi..'数据，则算其均值
        new_ndvi = [x for x in ndvi_ if np.isnan(x)==False ]
        ndvi_mean.append(round(np.mean(new_ndvi),5))
    else:  # 若某行数据四个'ndvi..'数据都缺失，则均值为 NAN
        ndvi_mean.append(np.NAN)
data['ndvi_mean']=ndvi_mean
```

图 3-4 计算植被指数均值

2)对于'ndvi_...'等四列中缺失的数据采用均值' ndvi_mean' 进行填充

```python
# 缺失的四个方位的 ndvi_ 数据 进行均值 替换
for col in ['ndvi_ne','ndvi_nw','ndvi_se','ndvi_sw']:
    df = data[data[col].isnull()].copy()
    df.loc[:,col] = df.ndvi_mean
    data[data[col].isnull()] = df
```

图 3-5 填充四列植被指数的缺失值

3)再次查看缺失数据情况

```
data.isnull().sum()
```

```
week_start_date                          0
ndvi_ne                                 23
ndvi_nw                                 23
ndvi_se                                 23
ndvi_sw                                 23
precipitation_amt_mm                    15
reanalysis_air_temp_k                   12
reanalysis_avg_temp_k                   12
reanalysis_dew_point_temp_k             12
reanalysis_max_air_temp_k               12
reanalysis_min_air_temp_k               12
reanalysis_precip_amt_kg_per_m2         12
reanalysis_relative_humidity_percent    12
reanalysis_sat_precip_amt_mm            15
reanalysis_specific_humidity_g_per_kg   12
reanalysis_tdtr_k                       12
station_avg_temp_c                      55
station_diur_temp_rng_c                 55
station_max_temp_c                      23
station_min_temp_c                      23
station_precip_mm                       27
total_cases                            416
ndvi_mean                               23
dtype: int64
```

图 3-6 再次查看缺失值

4) 分别查看某一个特征的缺失值情况

‘precipitation_amt_mm’（总降水量）：

```
data[data.precipitation_amt_mm.isnull()]
```

precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	reanalysis_max_air_temp_k	...	reanalysis_sat_precip_amt_mm	reanalysis_specific_humidity_g_per_kg
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	297.798571	298.057143	294.650000	300.2	...	NaN	16.060000
NaN	297.898571	298.107143	293.628571	300.1	...	NaN	15.012857
NaN	297.472857	297.678571	292.967143	299.3	...	NaN	14.381429
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

图 3-7 ‘precipitation_amt_mm’（总降水量）的缺失值情况

‘reanalysis_specific_humidity_g_per_kg’(平均比湿):

```
data[data.reanalysis_specific_humidity_g_per_kg.isnull()]
```

precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	reanalysis_max_air_temp_k	...	reanalysis_sat_precip_amt_mm	reanalysis_specific_humidity_g_per_kg
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

图 3-8 ‘reanalysis_specific_humidity_g_per_kg’(平均比湿)的缺失值情况

‘reanalysis_avg_temp_k’(平均气温)

```
data[data.reanalysis_avg_temp_k.isnull()]
```

precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	reanalysis_dew_point_temp_k	reanalysis_max_air_temp_k	...	reanalysis_sat_precip_amt_mm	reanalysis_specific_humidity_g_per_kg
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

图 3-9 ‘reanalysis_avg_temp_k’(平均气温)的缺失值情况

从上述可得，大部分的特征缺失值情况是：对于某行数据，多个特征变量出现缺失值，结合此次项目的主题是疾病大流行的预测分析，这种情况并不建议采用任何填充的方式，因此采取删除此部分数据集。

5）划分数据集，并执行相关数据清洗操作

```
# 删除不需要用到的字段
data = data.drop(['week_start_date','ndvi_mean'],axis=1)
```

图 3-10 删除无用字段

测试集:

```
# 取出测试集到 test_temp
test_temp = data.loc[data.total_cases.isnull()==True]
```

```
# 测试集去除目标列 'total_cases'
test_data = test_temp.drop('total_cases',axis=1)
# 对上述仍有少数缺失值的数据进行均值填充
test_data.fillna(value=test_data.mean(numeric_only=True),inplace=True)
```

图 3-11 从清理后的数据集划分出测试集

查看测试集清洗情况：

```
test_data.isnull().sum()
```

```
ndvi_ne                                  0
ndvi_nw                                  0
ndvi_se                                  0
ndvi_sw                                  0
precipitation_amt_mm                     0
reanalysis_air_temp_k                    0
reanalysis_avg_temp_k                    0
reanalysis_dew_point_temp_k              0
reanalysis_max_air_temp_k                0
reanalysis_min_air_temp_k                0
reanalysis_precip_amt_kg_per_m2          0
reanalysis_relative_humidity_percent     0
reanalysis_sat_precip_amt_mm             0
reanalysis_specific_humidity_g_per_kg    0
reanalysis_tdtr_k                        0
station_avg_temp_c                       0
station_diur_temp_rng_c                  0
station_max_temp_c                       0
station_min_temp_c                       0
station_precip_mm                        0
dtype: int64
```

图 3-12 测试集情况

训练集：

由上可知，特征变量的数据缺失较多，故而不采取替换操作，进行删除操作

```
# 取出训练集到 train_data
train_data = data.loc[data.total_cases.isnull()==False]
# 将'total_cases'转为 int 类型
train_data = train_data.astype({'total_cases':'int64'})
# 执行删除缺失数据
train_data.dropna(subset=data.columns.to_list()[:-1],how='any',axis=0,inplace=True)
```

图 3-13 删除训练集中缺失特征较多的数据

查看训练数据的变量情况：

```
print(f'变量列：{train_data.columns}; \n个数：{len(train_data.columns)}')
```

```
变量列：Index(['ndvi_ne', 'ndvi_nw', 'ndvi_se', 'ndvi_sw', 'precipitation_amt_mm',
       'reanalysis_air_temp_k', 'reanalysis_avg_temp_k',
       'reanalysis_dew_point_temp_k', 'reanalysis_max_air_temp_k',
       'reanalysis_min_air_temp_k', 'reanalysis_precip_amt_kg_per_m2',
       'reanalysis_relative_humidity_percent', 'reanalysis_sat_precip_amt_mm',
       'reanalysis_specific_humidity_g_per_kg', 'reanalysis_tdtr_k',
       'station_avg_temp_c', 'station_diur_temp_rng_c', 'station_max_temp_c',
       'station_min_temp_c', 'station_precip_mm', 'total_cases'],
      dtype='object');
个数：21
```

图 3-14 训练集的特征变量情况

查看训练集清洗情况：

```
train_data.isnull().sum()
```

```
ndvi_ne                                  0
ndvi_nw                                  0
ndvi_se                                  0
ndvi_sw                                  0
precipitation_amt_mm                     0
reanalysis_air_temp_k                    0
reanalysis_avg_temp_k                    0
reanalysis_dew_point_temp_k              0
reanalysis_max_air_temp_k                0
reanalysis_min_air_temp_k                0
reanalysis_precip_amt_kg_per_m2          0
reanalysis_relative_humidity_percent     0
reanalysis_sat_precip_amt_mm             0
reanalysis_specific_humidity_g_per_kg    0
reanalysis_tdtr_k                        0
station_avg_temp_c                       0
station_diur_temp_rng_c                  0
station_max_temp_c                       0
station_min_temp_c                       0
station_precip_mm                        0
total_cases                              0
dtype: int64
```

图 3-15 训练集的情况

6)按照城市划分数据

该数据集中有两个城市：San Juan（sj）和 Iquitos（iq）

由于题目说明登革热的传播可能在两者之间遵循不同的模式，我们将按照城市划分数据集，为每个城市训练单独的模型。

```
sj_train_data = train_data.loc['sj']
iq_train_data = train_data.loc['iq']
```

```
print('城市sj的训练数据集情况:',sj_train_data.shape)
print('城市iq的训练数据集情况:',iq_train_data.shape)
```

```
城市sj的训练数据集情况: (911, 21)
城市iq的训练数据集情况: (472, 21)
```

图 3-16 按照城市划分训练集

探索性分析：

1.城市 San Juan（sj）：

1)目标变量的分布

```
plt.figure(figsize=(15,5))
ax1 = plt.subplot(1,2,1)
sns.histplot(data=sj_train_data,x='total_cases',kde=True)
ax1.set_title('城市sj的目标变量total_cases的分布直方图')
ax2 = plt.subplot(1,2,2)
sns.boxplot(data=sj_train_data,y='total_cases')
ax2.set_title('城市sj的目标变量total_cases的箱线图')
plt.show()
```

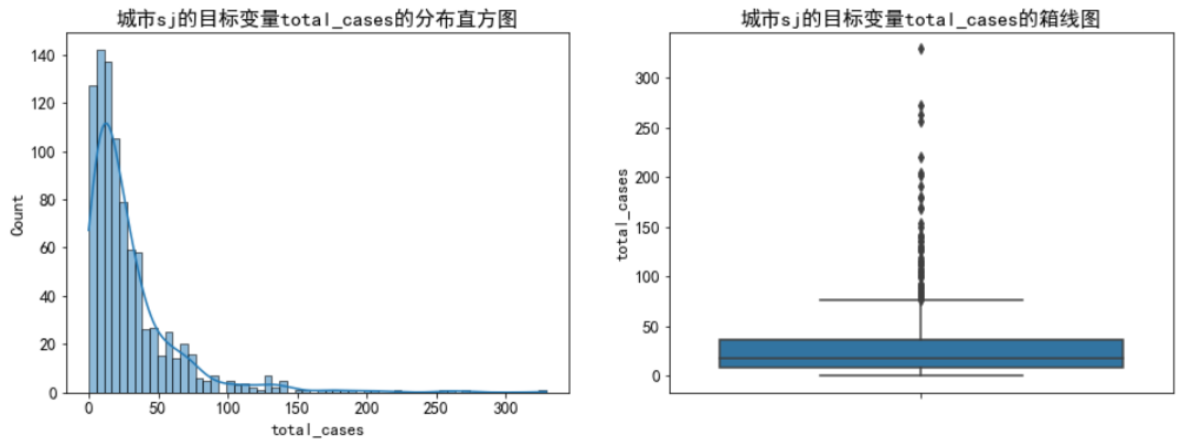
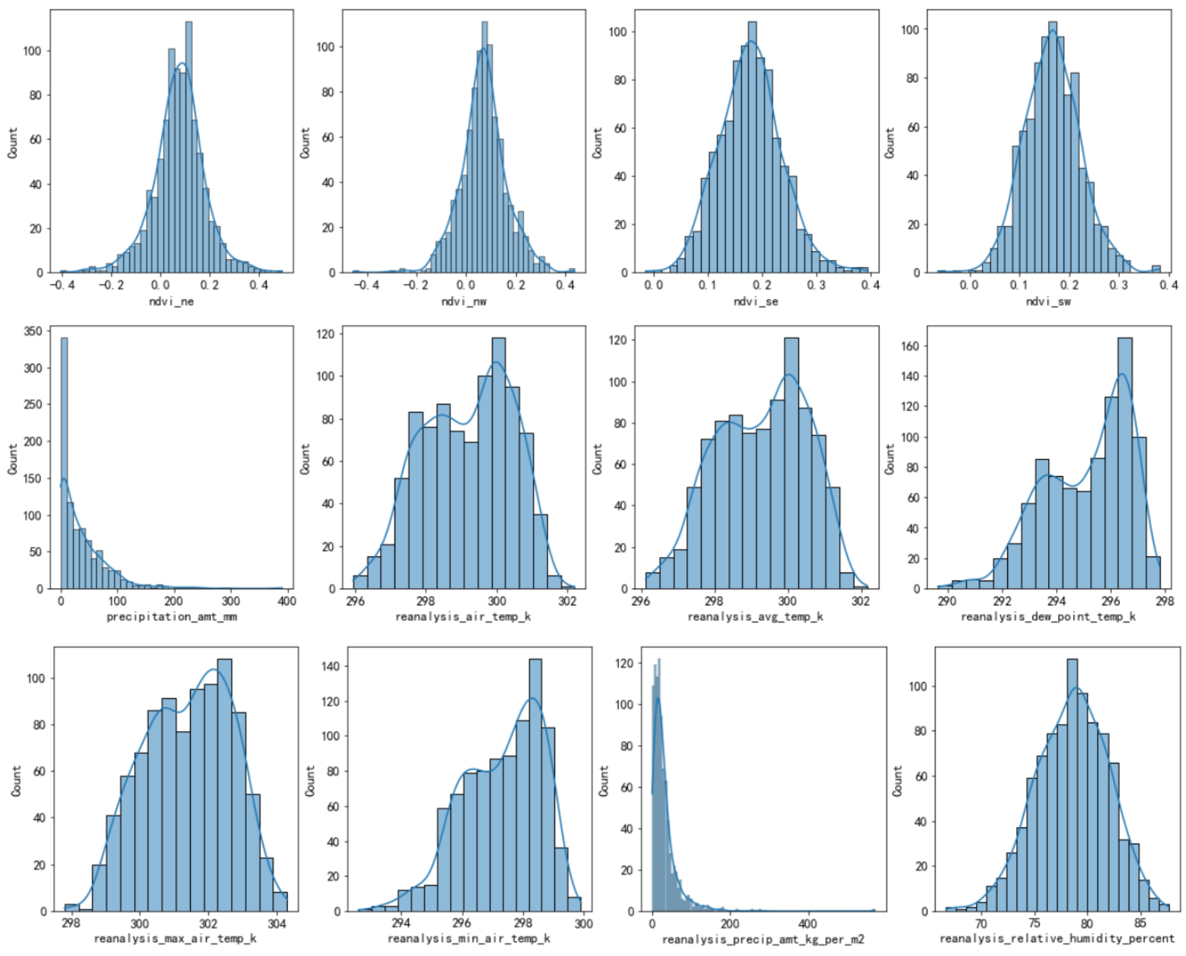


图 3-17 城市 sj 的目标变量分布情况

2)特征变量的分布

```python
#城市sj数值型变量分布（直方图）
plt.figure(figsize=(20,28))
i=1
for temp in numerical:
    plt.subplot(5,4,i)
    sns.histplot(data = sj_train_data,x=temp,kde=True)
    i+=1
plt.show()
```



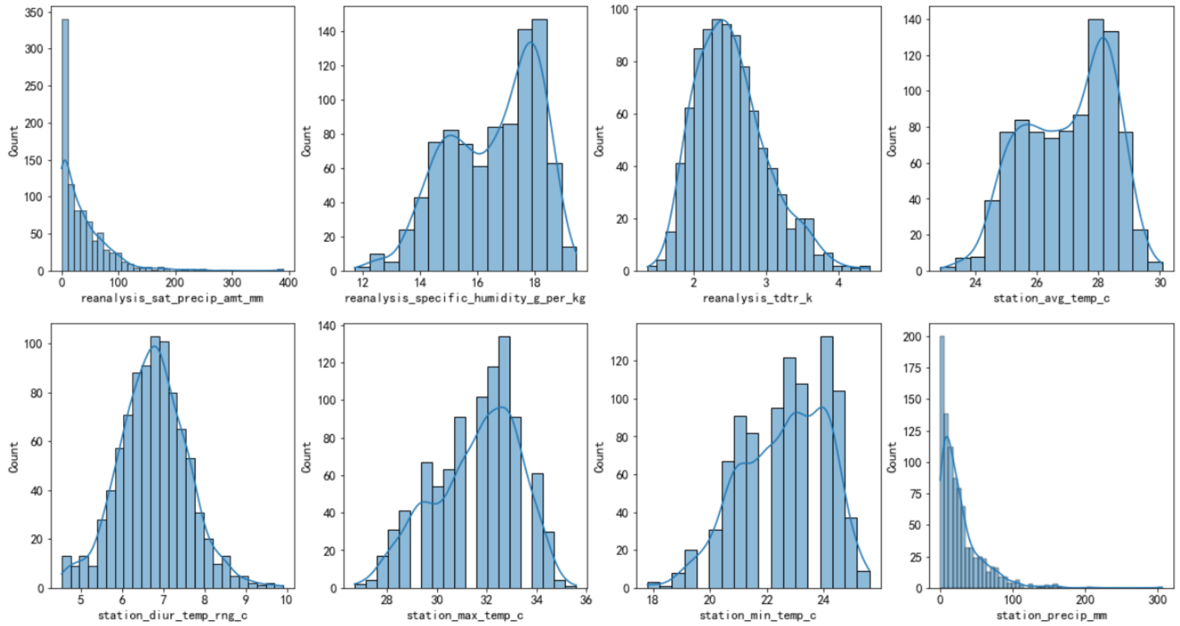
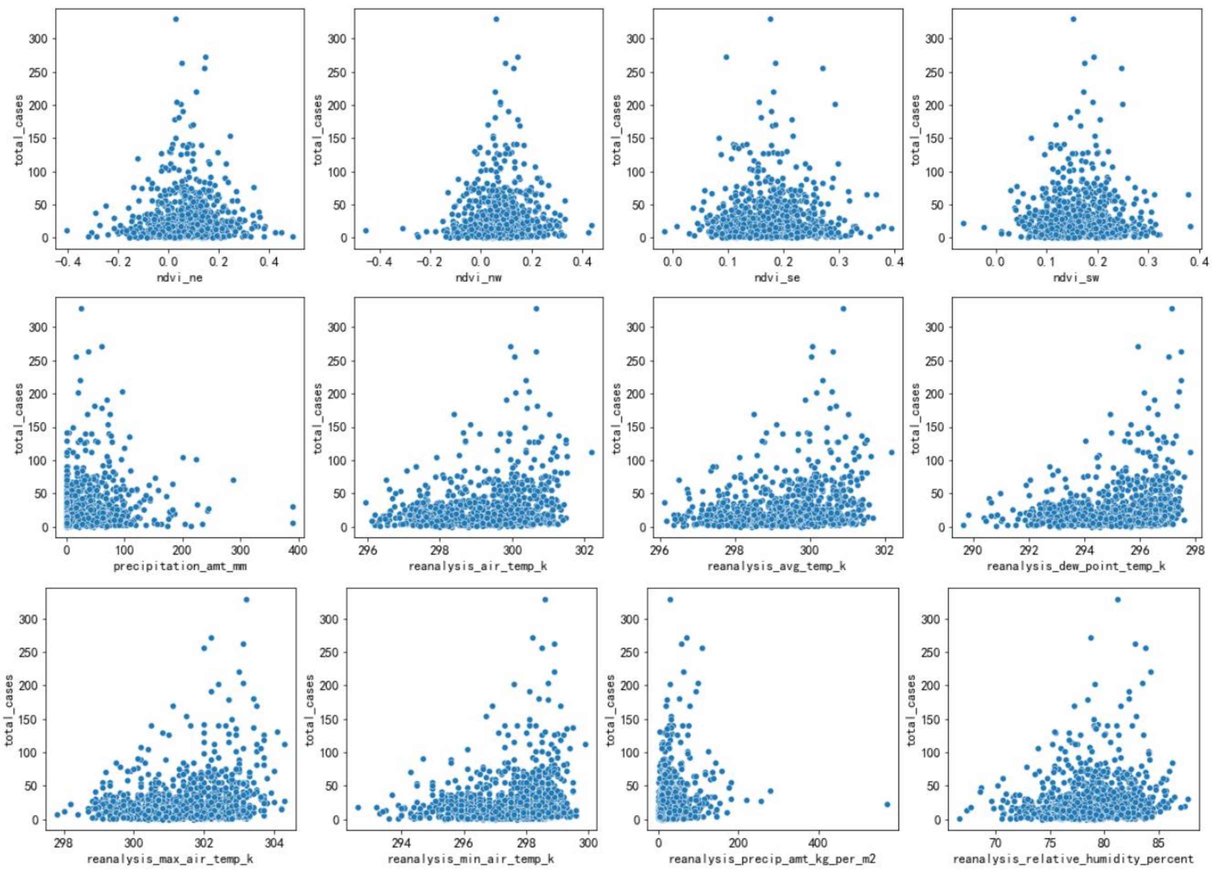


图 3-18 城市 sj 特征变量的分布情况(直方图)

```python
# 城市sj数值型变量和目标变量total_cases分布情况（散点图）
plt.figure(figsize=(20,25))
i = 1
for temp in numerical:
    ax = plt.subplot(5,4,i)
    sns.scatterplot(data=sj_train_data,x=temp,y='total_cases')
    i += 1
plt.show()
```



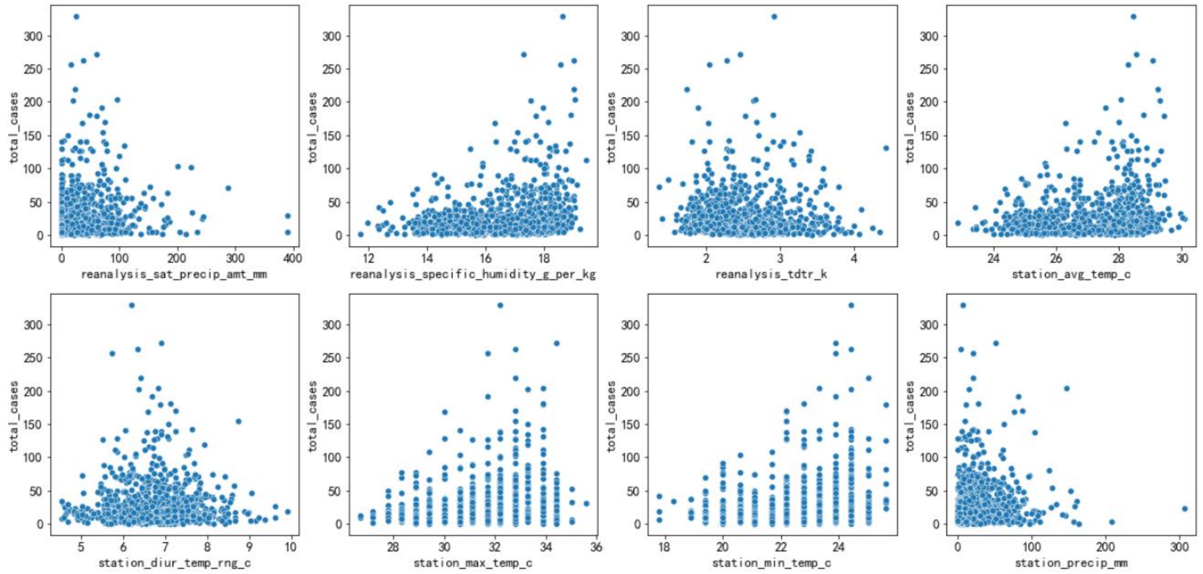


图 3-19 城市 sj 特征变量与目标变量的相关分布情况(散点图)

城市 sj 的 EDA 特征分析：

sj_train_data 的'precipitation_amt_mm'、'reanalysis_precip_amt_kg_per_m2'、'reanalysis_sat_precip_amt_mm'、'station_precip_mm'存在明显的偏态！

上述各特征与目标变量均呈现一定的相关性！

2. 城市 Iquitos（iq）：

1) 目标变量的分布

```
plt.figure(figsize=(15,5))
ax1 = plt.subplot(1,2,1)
sns.histplot(data=iq_train_data,x='total_cases',kde=True)
ax1.set_title('城市iq的目标变量total_cases的分布直方图')
ax2 = plt.subplot(1,2,2)
sns.boxplot(data=iq_train_data,y='total_cases')
ax2.set_title('城市iq的目标变量total_cases的箱线图')
plt.show()
```

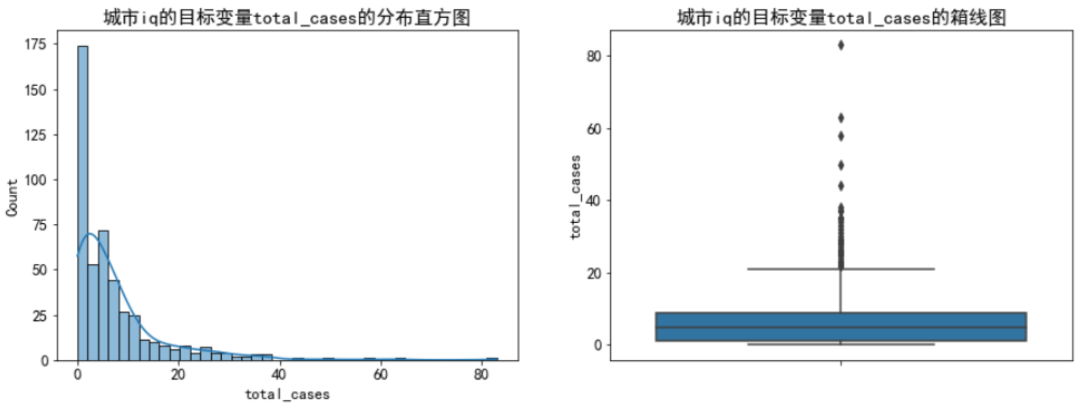


图 3-20 城市 iq 的目标变量分布情况

2) 特征变量的分布

```
# 城市iq数值型变量分布（直方图）
plt.figure(figsize=(20,28))
i=1
for temp in numerical:
    plt.subplot(5,4,i)
    sns.histplot(data = iq_train_data,x=temp,kde=True)
    i+=1
plt.show()
```

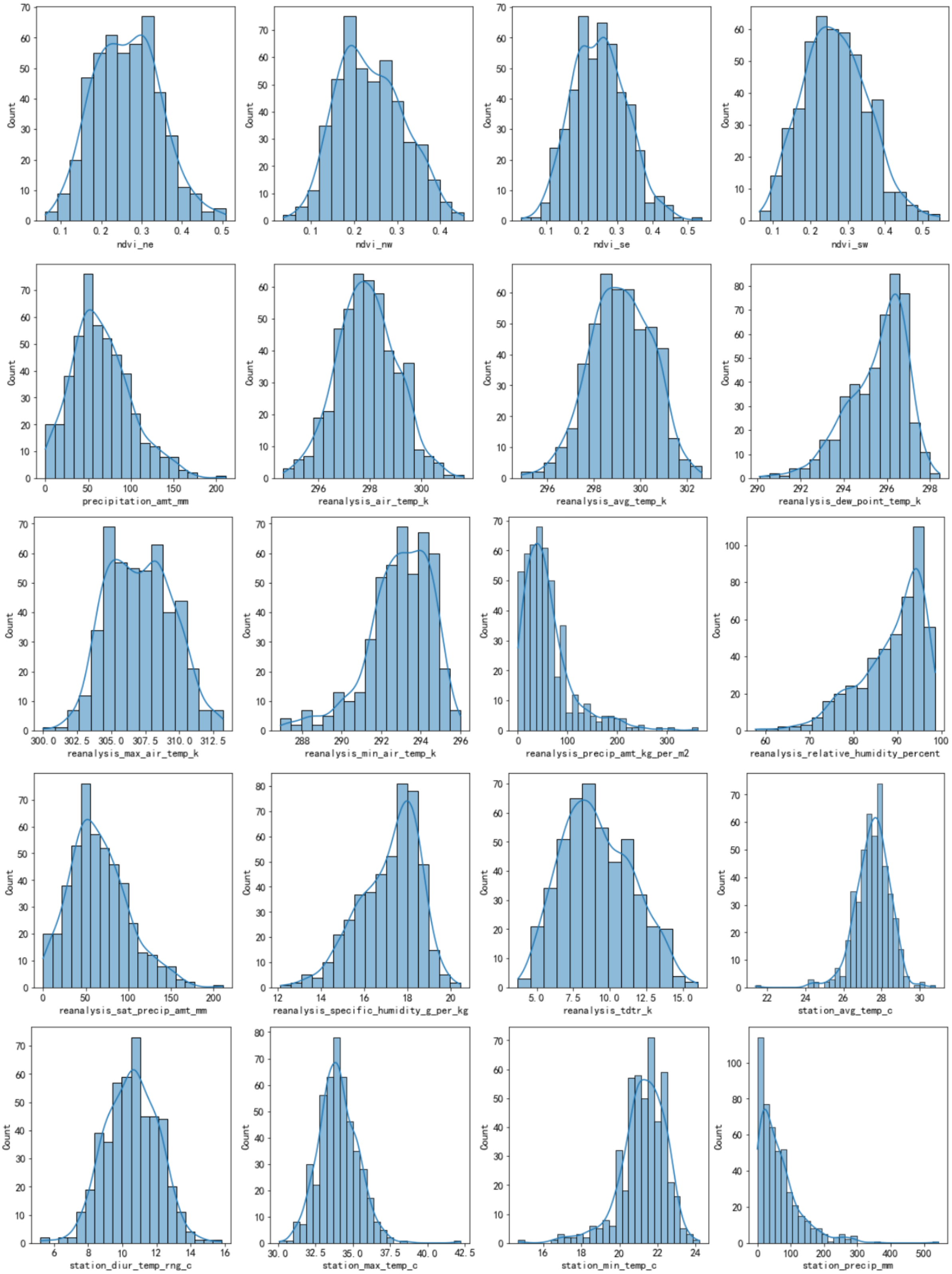


图 3-21 城市 iq 特征变量的分布情况(直方图)

```python
# 城市iq数值型变量和目标变量total_cases分布情况（散点图）
plt.figure(figsize=(20,25))
i = 1
for temp in numerical:
    plt.subplot(5,4,i)
    sns.scatterplot(data=iq_train_data,x=temp,y='total_cases')
    i += 1
plt.show()
```

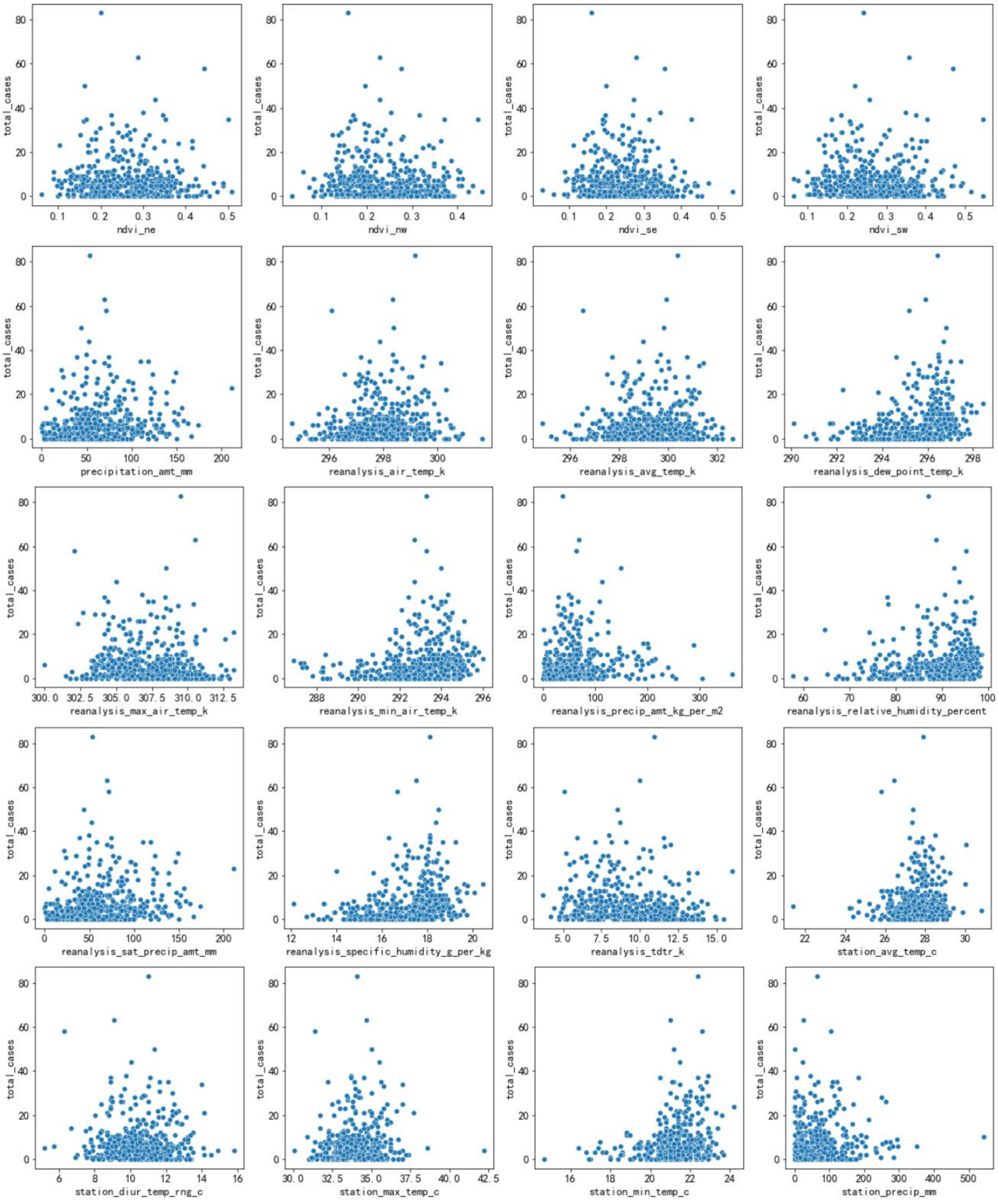


图 3-22 城市 iq 特征变量与目标变量的相关分布情况(散点图)

城市 iq 的 EDA 特征分析：

iq_train_data 的'reanalysis_precip_amt_kg_per_m2'、'reanalysis_relative_humidity_percent'、'station_precip_mm'存在明显的偏态！

上述各特征与目标变量均呈现一定的相关性！

结合两个城市分析：两个城市不同否方位的植被指数分布大致是相似的，而且数值区间差距很小，因此可以考虑后续合并特征变量，减少特征数量。

四、特征工程

1. 合并植被指数特征变量

```
# 将'ndvi_ne','ndvi_nw','ndvi_se','ndvi_sw'合并为'ndvi_mean'
train_data['ndvi_mean'] = train_data.iloc[:,0:4].mean(axis=1)
test_data['ndvi_mean'] = test_data.iloc[:,0:4].mean(axis=1)
# 去除替换前的四列
train_data.drop(['ndvi_ne','ndvi_nw','ndvi_se','ndvi_sw'],axis=1,inplace=True)
test_data.drop(['ndvi_ne','ndvi_nw','ndvi_se','ndvi_sw'],axis=1,inplace=True)
```

图 4-1 合并植被指数特征变量

2. 消除偏态

按照城市划分训练集

```
sj_train_data = train_data.loc['sj']
iq_train_data = train_data.loc['iq']
```

图 4-2 按照城市划分训练集

1） sj_train_data 消除偏态

```
# 备份 sj_train_data
sj_train_data_ = sj_train_data.copy()
```

```
# sj_train_data消除偏态
skewness_lt = ['precipitation_amt_mm','reanalysis_precip_amt_kg_per_m2',
               'reanalysis_sat_precip_amt_mm','station_precip_mm']
for temp in skewness_lt:
    sj_train_data_.loc[:,temp] = sj_train_data_.loc[:,temp].copy().map(lambda x:np.log(x+1))
```

图 4-3 城市 sj 训练集消除偏态

查看处理情况：

```
display('sj_train_data消除偏态后的部分数据分布情况:')
plt.figure(figsize=(12,8))
i=1
for temp in skewness_lt:
    plt.subplot(2,2,i)
    sns.histplot(data = sj_train_data_,x=temp,kde=True)
    i+=1
plt.show()
```

'sj_train_data消除偏态后的部分数据分布情况:'

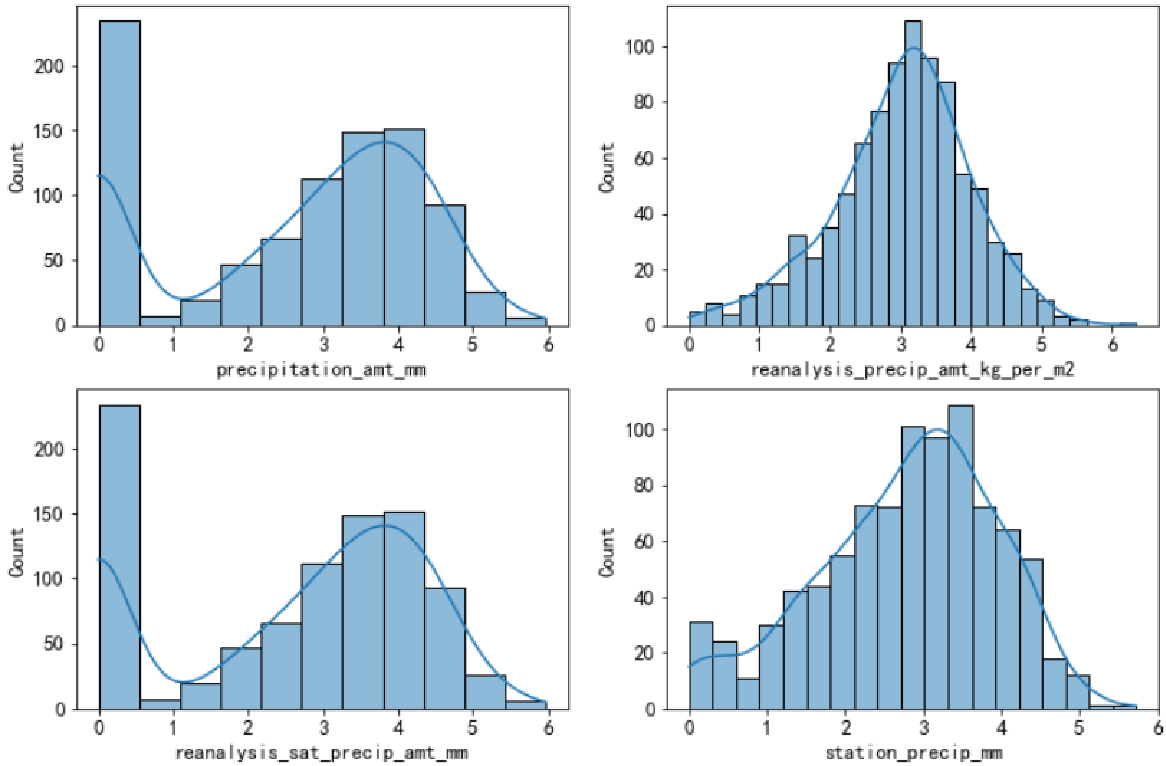


图 4-4 查看城市 sj 训练集消除偏态后的情况

2) iq_train_data 消除偏态

```
# 备份 iq_train_data
iq_train_data_ = iq_train_data.copy()
```

```
# iq_train_data
skewness_lt = ['reanalysis_precip_amt_kg_per_m2',
               'reanalysis_relative_humidity_percent','station_precip_mm']
for temp in skewness_lt:
    iq_train_data_.loc[:,temp] = iq_train_data_.loc[:,temp].copy().map(lambda x:np.log(x+1))
```

图 4-5 城市 iq 训练集消除偏态

查看处理情况：

```
display('iq_train_data消除偏态后的部分数据分布情况:')
plt.figure(figsize=(12,8))
i=1
for temp in skewness_lt:
    plt.subplot(2,2,i)
    sns.histplot(data = iq_train_data_,x=temp,kde=True)
    i+=1
plt.show()
```

'iq_train_data消除偏态后的部分数据分布情况:'

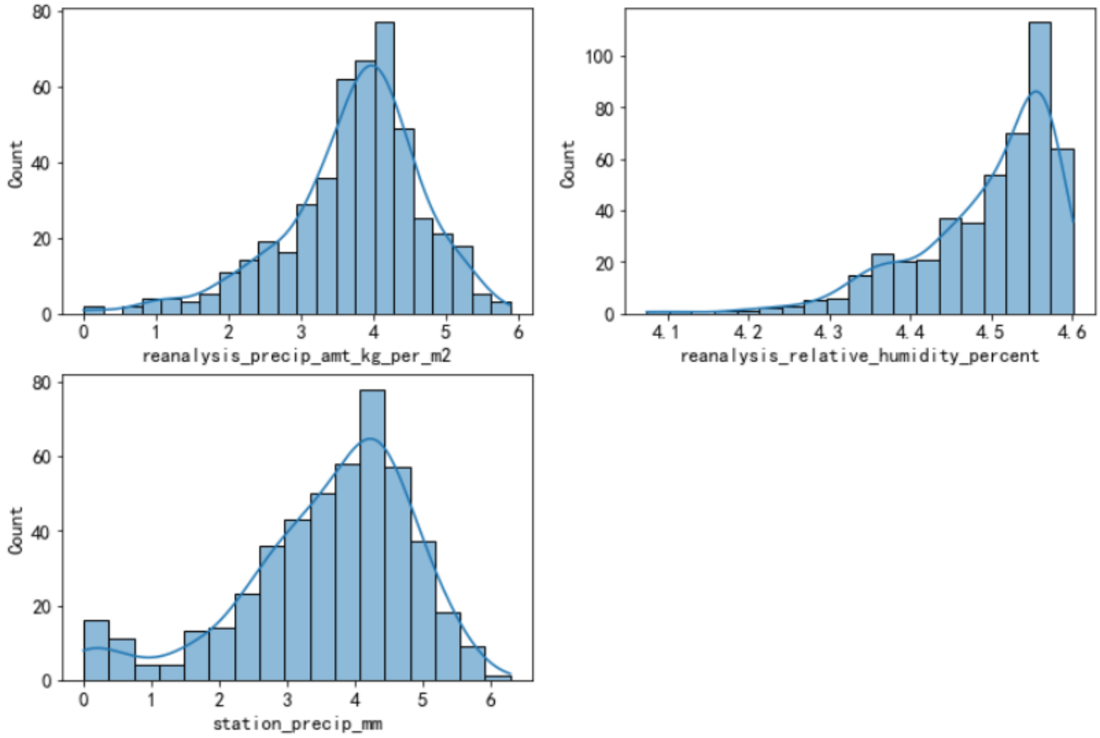


图 4-6 查看城市 sj 训练集消除偏态后的情况

3. 标准化

从数据的探索性分析以及特征处理的情况来看，两个城市的数据分布都近似服从正态分布，因此选择 Z-score 标准化处理。

1) sj_train_data_

```
# sj_train_data_
from sklearn.preprocessing import StandardScaler
Stan_scaler = StandardScaler()
temp = sj_train_data_.drop('total_cases',axis=1)
sj_results = Stan_scaler.fit_transform(temp)
sj_train_Stan = pd.DataFrame(data=sj_results,columns=temp.columns)
```

图 4-7 城市 sj 训练集标准化

2) iq_train_data_

```
# iq_train_data_
from sklearn.preprocessing import StandardScaler
Stan_scaler = StandardScaler()
temp = iq_train_data_.drop('total_cases',axis=1)
iq_results = Stan_scaler.fit_transform(temp)
iq_train_Stan = pd.DataFrame(data=iq_results,columns=temp.columns)
```

图 4-8 城市 iq 训练集标准化

3)测试集

```
# 测试集标准化
from sklearn.preprocessing import StandardScaler
Stan_scaler = StandardScaler()
test_results = Stan_scaler.fit_transform(test_data)
#test_data.iloc[:,0:17] = test_results
test_Stan = pd.DataFrame(index=test_data.index,data=test_results,columns=test_data.columns)
```

图 4-9 测试集标准化

4. 划分数据集

```
# sj
sj_X = sj_train_data_.drop('total_cases',axis=1)
sj_Y = sj_train_data_['total_cases']
```

```
# sj 标准化
sj_X_Stan = sj_train_Stan
sj_Y_Stan = sj_train_data_['total_cases']
```

```
# iq
iq_X = iq_train_data_.drop('total_cases',axis=1)
iq_Y = iq_train_data_['total_cases']
```

```
# iq 标准化
iq_X_Stan = iq_train_Stan
iq_Y_Stan = iq_train_data_['total_cases']
```

```
# 测试集
sj_test_X = test_data.loc['sj']
iq_test_X = test_data.loc['iq']
```

```
# 测试集 标准化
sj_test_X_Stan = test_Stan.loc['sj']
iq_test_X_Stan = test_Stan.loc['iq']
```

图 4-10 划分数据集

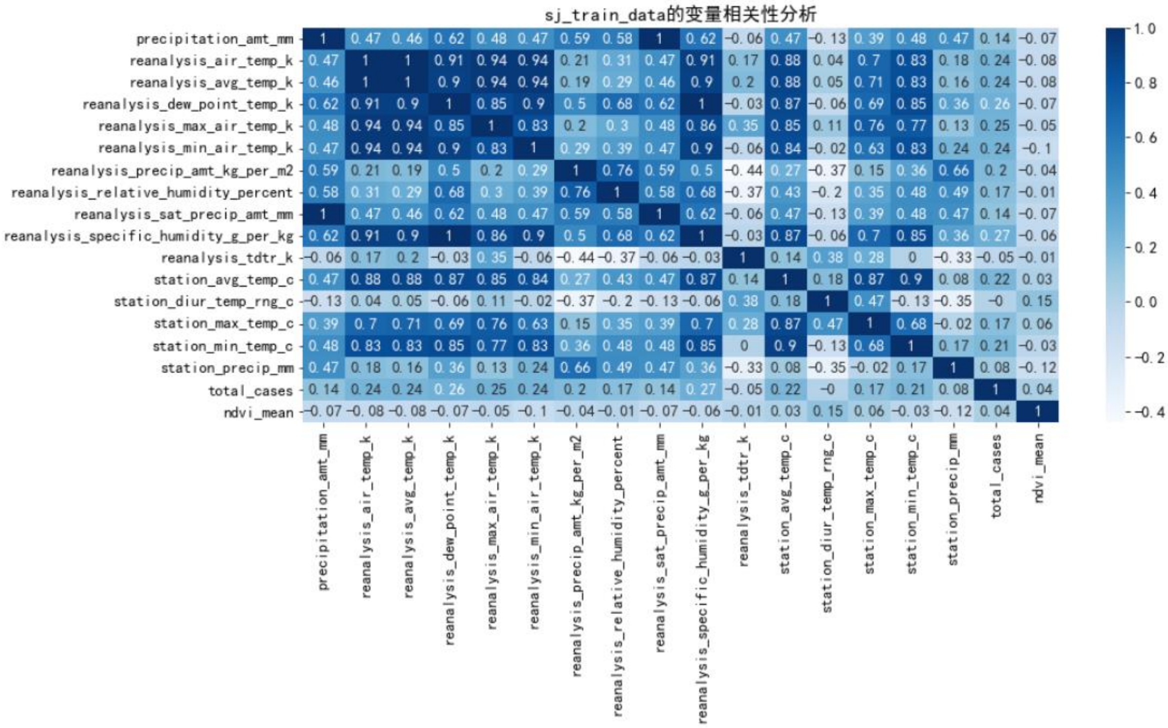
5. 特征选择

由前期数据预处理和探索性分析可得，数据分布近似服从正态分布，因此选取变量相关性筛选和基于 F 检验过滤的高相关过滤法进行特征选择

1) sj

城市 sj 变量相关性筛选（筛选相关性为 0 的特征）

```
# sj_train_data_的变量相关性分析
corr_sj = sj_train_data_.corr()
corr_sj = round(corr_sj,2)
plt.figure(figsize=(14,6))
sns.heatmap(corr_sj,annot=True,cmap=plt.cm.Blues,fmt='g')
plt.title('sj_train_data的变量相关性分析')
plt.show()
```




```
sj_train_corr_df = corr_sj['total_cases'].sort_values(ascending=False)
sj_corr_lt = sj_train_corr_df[sj_train_corr_df.values>0].index.to_list()
sj_corr_lt = corr_lt[1:]
```

```
print('相关性筛选后的特征名称：')
display(sj_corr_lt)
print('筛选后的特征数量为：',len(sj_corr_lt))
```

```
相关性筛选后的特征名称：
['reanalysis_dew_point_temp_k',
 'reanalysis_max_air_temp_k',
 'reanalysis_min_air_temp_k',
 'reanalysis_air_temp_k',
 'reanalysis_avg_temp_k',
 'station_avg_temp_c',
 'station_min_temp_c',
 'reanalysis_precip_amt_kg_per_m2',
 'station_max_temp_c',
 'reanalysis_relative_humidity_percent',
 'precipitation_amt_mm',
 'reanalysis_sat_precip_amt_mm',
 'station_precip_mm',
 'ndvi_mean']
筛选后的特征数量为： 14
```

图 4-11 城市 sj 的变量相关性筛选

高相关过滤法（F 检验）

```
# 重构数据集
sj_X = sj_X.loc[:,sj_corr_lt]
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
f_value, p_value = f_classif(sj_X,sj_Y)
#根据 p 值，得出 k 值
k = f_value.shape[0] - (p_value > 0.05).sum()
#筛选后特征
X_classif = SelectKBest(f_classif, k=k).fit(sj_X, sj_Y)
print('过滤前的特征维度:',sj_X.shape)
print('高相关过滤法(F检验过滤)后的数据特征维度:',X_classif.transform(sj_X).shape)
print('高相关过滤法(F检验过滤)后的数据特征名称:\n',X_classif.get_feature_names_out())
```

```
过滤前的特征维度: (911, 14)
高相关过滤法(F检验过滤)后的数据特征维度: (911, 6)
高相关过滤法(F检验过滤)后的数据特征名称:
 ['reanalysis_dew_point_temp_k' 'reanalysis_max_air_temp_k'
 'reanalysis_min_air_temp_k' 'reanalysis_air_temp_k'
 'reanalysis_avg_temp_k' 'station_min_temp_c']
```

图 4-12 城市 sj 数据的高相关过滤法

按照特征选择的重要变量重构 sj 的数据集

```
# 获取 F检验过滤的重要特征列表
sj_lt = X_classif.get_feature_names_out().tolist()
# 重构数据集
sj_X_Stan = sj_X_Stan.loc[:,sj_lt]
```

```
# 重构数据集
sj_X = sj_X.loc[:,sj_lt]
```

对sj数据集划分出验证数据集

```
# 对sj数据集划分出验证数据集
from sklearn.model_selection import train_test_split
sj_X_train,sj_X_test,sj_Y_train,sj_Y_test =  train_test_split(sj_X,sj_Y,test_size=0.2,random_state=10)
```

```
# 对sj数据集划分出验证数据集
from sklearn.model_selection import train_test_split
sj_X_train_S,sj_X_test_S,sj_Y_train_S,sj_Y_test_S =  train_test_split(sj_X_Stan,sj_Y_Stan,test_size=0.2,random_state=10)
```

```
# 测试集选择筛选过的特征
sj_test_X = sj_test_X.loc[:,sj_lt]
```

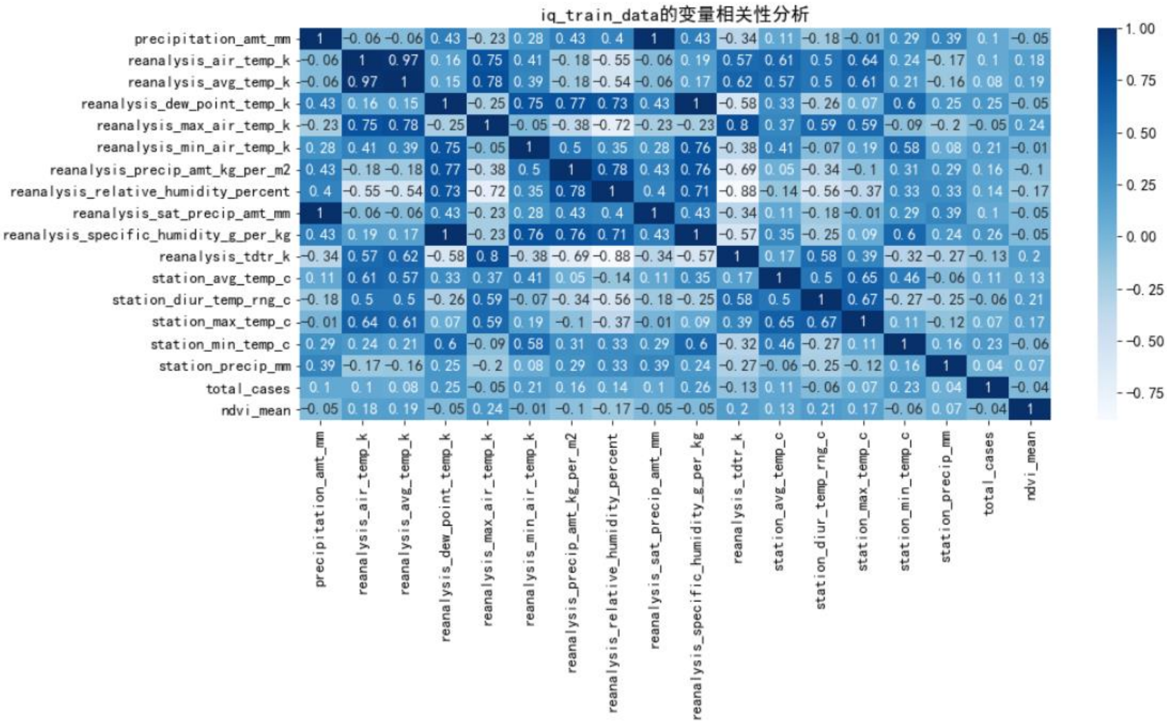
```
# 测试集选择筛选过的特征
sj_test_X_Stan = sj_test_X_Stan.loc[:,sj_lt]
```

图 4-13 按照筛选结果重构城市 sj 的数据集

2) iq

城市 iq 变量相关性筛选（筛选相关性为 0 的特征）

```
# iq_train_data的变量相关性分析
corr_iq = iq_train_data_.corr()
corr_iq = round(corr_iq,2)
plt.figure(figsize=(14,6))
sns.heatmap(corr_iq,annot=True,cmap=plt.cm.Blues,fmt='g')
plt.title('iq_train_data的变量相关性分析')
plt.show()
```



```
iq_train_corr_df = corr_iq['total_cases'].sort_values(ascending=False)
iq_corr_lt = iq_train_corr_df[iq_train_corr_df.values>0].index.to_list()
iq_corr_lt = iq_corr_lt[1:]
```

```
print('相关性筛选后的特征名称：')
display(iq_corr_lt)
print('筛选后的特征数量为：',len(iq_corr_lt))
```

```
相关性筛选后的特征名称：
['reanalysis_specific_humidity_g_per_kg',
 'reanalysis_dew_point_temp_k',
 'station_min_temp_c',
 'reanalysis_min_air_temp_k',
 'reanalysis_precip_amt_kg_per_m2',
 'reanalysis_relative_humidity_percent',
 'station_avg_temp_c',
 'reanalysis_air_temp_k',
 'precipitation_amt_mm',
 'reanalysis_sat_precip_amt_mm',
 'reanalysis_avg_temp_k',
 'station_max_temp_c',
 'station_precip_mm']
筛选后的特征数量为： 13
```

图 4-14 城市 iq 的变量相关性筛选

城市 iq 高相关过滤法（F 检验）

```
# 重构数据集
iq_X = iq_X.loc[:,iq_corr_lt]
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
f_value, p_value = f_classif(iq_X,iq_Y)
#根据 p 值，得出 k 值
k = f_value.shape[0] - (p_value > 0.05).sum()
#筛选后特征
X_classif = SelectKBest(f_classif, k=k).fit(iq_X, iq_Y)
print('过滤前的特征维度:',iq_X.shape)
print('高相关过滤法(F检验过滤)后的数据特征维度:',X_classif.transform(iq_X).shape)
print('高相关过滤法(F检验过滤)后的数据特征名称:\n',X_classif.get_feature_names_out())
```

```
过滤前的特征维度: (472, 13)
高相关过滤法(F检验过滤)后的数据特征维度: (472, 6)
高相关过滤法(F检验过滤)后的数据特征名称:
 ['reanalysis_specific_humidity_g_per_kg' 'reanalysis_dew_point_temp_k'
 'station_min_temp_c' 'reanalysis_min_air_temp_k'
 'reanalysis_relative_humidity_percent' 'station_precip_mm']
```

图 4-15 城市 iq 数据的高相关过滤法

按照特征选择的重要变量重构 iq 的训练集

```
# 获取 F检验过滤的重要特征列表
iq_lt = X_classif.get_feature_names_out().tolist()
# 重构数据集
iq_X = iq_X.loc[:,iq_lt]
```

对iq数据集划分出验证数据集

```
# 对iq数据集划分出验证数据集
from sklearn.model_selection import train_test_split
iq_X_train,iq_X_test,iq_Y_train,iq_Y_test =  train_test_split(iq_X,iq_Y,test_size=0.2,random_state=101)
```

```
# 对iq标准化后的数据集划分出验证数据集
from sklearn.model_selection import train_test_split
iq_X_train_S,iq_X_test_S,iq_Y_train_S,iq_Y_test_S =  train_test_split(iq_X_Stan,iq_Y_Stan,test_size=0.2,random_state=101)
```

```
# 测试集选择筛选过的特征
iq_test_X = iq_test_X.loc[:,iq_lt]
```

```
# 标准化的测试集选择筛选过的特征
iq_test_X_Stan = iq_test_X_Stan.loc[:,iq_lt]
```

图 4-16 按照筛选结果重构城市 iq 的数据集

五、 建模、模型优化和模型评价

1. 城市 sj

1) 模型评价以及交叉验证函数(实现代码复用)

```
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_score
def cross_val(model):  #标准化数据的交叉验证
    pred_mae = cross_val_score(model, sj_X_Stan, sj_Y_Stan, cv=10,scoring='neg_mean_absolute_error')
    pred_r2 = cross_val_score(model, sj_X_Stan, sj_Y_Stan, cv=10,scoring='r2')
    return pred_mae.mean(),pred_r2.mean()
def cross_val_1(model):  #交叉验证
    pred_mae = cross_val_score(model, sj_X, sj_Y, cv=10,scoring='neg_mean_absolute_error')
    pred_r2 = cross_val_score(model, sj_X, sj_Y, cv=10,scoring='r2')
    return pred_mae.mean(),pred_r2.mean()
def print_evaluate(true, predicted): # 显示模型评估指标
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('----------------------')
def evaluate(true, predicted): # 返回模型评估指标
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    return mae, mse, rmse
```

图 5-1 城市 sj 的模型评价函数等

2) 模型建模,调优，训练，模型评价，交叉验证

a. 线性回归

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(sj_X_train_S,sj_Y_train_S)
sj_Y_pred = LR.predict(sj_X_test_S)
```

```
sj_results_df = pd.DataFrame(
    data=[["LinearRegression", *evaluate(sj_Y_test_S, sj_Y_pred),*cross_val(LinearRegression())]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])

print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train_S,LR.predict(sj_X_train_S))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test_S,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 21.342095583064726
MSE: 1129.0565455441952
RMSE: 33.60143665893164
----------------------
evaluation of sj_X_test:
MAE: 22.223937996530157
MSE: 1117.755902366062
RMSE: 33.432856628862304
----------------------
```

图 5-2 城市 sj 线性回归

b. 岭回归

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
grid_Ridge = GridSearchCV(estimator=Ridge(random_state=10),
                    param_grid={'alpha':[0.001,0.01,0.1,1]},
                    cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_Ridge.fit(sj_X_Stan,sj_Y_Stan)
print(f'最优得分为：{grid_Ridge.best_score_}\n最优参数为：{grid_Ridge.best_params_}',)
```

```
最优得分为：-22.907137203086396
最优参数为：{'alpha': 1}
```

```
from sklearn.linear_model import Ridge
Ridge_Model = Ridge(alpha=1)
Ridge_Model.fit(sj_X_train_S,sj_Y_train_S)
sj_Y_pred = Ridge_Model.predict(sj_X_test_S)
```

```
results_Ridge = pd.DataFrame(
    data=[["Ridge", *evaluate(sj_Y_test_S, sj_Y_pred),*cross_val(Ridge_Model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_Ridge],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train_S,Ridge_Model.predict(sj_X_train_S))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test_S,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 21.355114029867675
MSE: 1130.15458598023
RMSE: 33.61777187709248
-----------------------
evaluation of sj_X_test:
MAE: 22.257759074443157
MSE: 1118.9153518057185
RMSE: 33.450192104167634
-----------------------
```

图 5-3 城市 sj 岭回归

c. Lasso 回归

```
#构造不同的Lambda值
alphas = np.linspace(0.001,1,20)
#设置交叉验证的参数，使用均方误差评估
lasso_cv = LassoCV(alphas=alphas,cv=5,max_iter=10000)
lasso_cv.fit(sj_X_Stan,sj_Y_Stan)
# 获取最佳的Lasso回归alpha
print('Lasso回归最佳的alpha:',lasso_cv.alpha_)
```

```
Lasso回归最佳的alpha: 0.001
```

```
Lasso_model = Lasso(alpha=lasso_cv.alpha_,max_iter=10000)
Lasso_model.fit(sj_X_train_S,sj_Y_train_S)
sj_Y_pred = Lasso_model.predict(sj_X_test_S)
```

```
results_Lasso = pd.DataFrame(
    data=[["Lasso", *evaluate(sj_Y_test_S, sj_Y_pred),*cross_val(Lasso_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_Lasso],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train_S,Lasso_model.predict(sj_X_train_S))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test_S,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 21.34194024799269
MSE: 1129.057510672334
RMSE: 33.601451020340384
-----------------------
evaluation of sj_X_test:
MAE: 22.224373379810313
MSE: 1117.758678399937
RMSE: 33.43289814538873
-----------------------
```

图 5-4 城市 sj Lasso 回归

d. 弹性网回归

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
grid_ElasticNet = GridSearchCV(estimator=ElasticNet(random_state=10,max_iter=10000),
                  param_grid={'alpha':[0.001,0.01,0.1,1,10],
                              'l1_ratio':[0.3,0.5,0.7,0.9]},
                  cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_ElasticNet.fit(sj_X_Stan,sj_Y_Stan)
print(f'最优得分为：{grid_ElasticNet.best_score_}\n最优参数为：{grid_ElasticNet.best_params_}',)
```

```
最优得分为：-22.851469186843104
最优参数为：{'alpha': 10, 'l1_ratio': 0.3}
```

```
ElasticNet_Model = ElasticNet(alpha=10,l1_ratio=0.3,random_state=10,max_iter=10000)
ElasticNet_Model.fit(sj_X_train_S,sj_Y_train_S)
sj_Y_pred = ElasticNet_Model.predict(sj_X_test_S)
```

```
results_ElasticNet = pd.DataFrame(
    data=[["ElasticNet", *evaluate(sj_Y_test_S, sj_Y_pred),*cross_val(ElasticNet_Model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_ElasticNet],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train_S,ElasticNet_Model.predict(sj_X_train_S))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test_S,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 21.91043335693144
MSE: 1200.215377492488
RMSE: 34.64412471823307
-----------------------
evaluation of sj_X_test:
MAE: 22.237062835008174
MSE: 1152.1370319698797
RMSE: 33.94314410849236
-----------------------
```

图 5-5 城市 sj 弹性网回归

e. XGBoost 回归

```
from xgboost import XGBRegressor
#XGBR_model = XGBRegressor()
from sklearn.model_selection import GridSearchCV
grid_XGBR = GridSearchCV(estimator=XGBRegressor(random_state=10),
                   param_grid={'n_estimators':[100,200,300,500],'max_leaves':[2,4,6],
                               'max_depth':[3,4,5],'learning_rate':[0.04,0.06,0.1],
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_XGBR.fit(sj_X,sj_Y)
print(f'最优得分为：{grid_XGBR.best_score_}\n最优参数为：{grid_XGBR.best_params_}',)
```

```
最优得分为：-23.528626546044826
最优参数为：{'learning_rate': 0.04, 'max_depth': 3, 'max_leaves': 2, 'n_estimators': 100}
```

```
XGBR_model = XGBRegressor(n_estimators=100,learning_rate=0.04,max_depth=3,max_leaves=2,random_state=10)
XGBR_model.fit(sj_X_train,sj_Y_train)
sj_Y_pred = XGBR_model.predict(sj_X_test)
```

```
results_XGBRegressor = pd.DataFrame(
    data=[["XGBRegressor", *evaluate(sj_Y_test, sj_Y_pred),*cross_val_1(XGBR_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_XGBRegressor],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train,XGBR_model.predict(sj_X_train))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 18.146226675955806
MSE: 741.9933745774871
RMSE: 27.239555330024885
-----------------------
evaluation of sj_X_test:
MAE: 21.355872201137856
MSE: 1131.6936105011225
RMSE: 33.64065413307421
-----------------------
```

图 5-6 城市 sj XGBoost 回归

f. 梯度提升回归

```
from sklearn.ensemble import GradientBoostingRegressor
grid_GBR = GridSearchCV(estimator=GradientBoostingRegressor(random_state=10),
                   param_grid={'n_estimators':[100,200,300],'learning_rate':[0.01,0.1,0.3],
                               'max_depth':[2,3,5],'min_samples_split':[2,4,6,10],
                               'min_samples_leaf':[3,5,7,10,12]
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_GBR.fit(sj_X,sj_Y)
print(f'最优得分为：{grid_GBR.best_score_}\n最优参数为：{grid_GBR.best_params_}',)
```

```
最优得分为：-22.703247330833722
最优参数为：{'learning_rate': 0.01, 'max_depth': 2, 'min_samples_leaf': 12, 'min_samples_split': 2, 'n_estimators': 100}
```

```
GBR_model = GradientBoostingRegressor(n_estimators=100,learning_rate=0.01, max_depth=2,
                                      min_samples_leaf=12, min_samples_split=2,random_state=10)
GBR_model.fit(sj_X_train,sj_Y_train)
sj_Y_pred = GBR_model.predict(sj_X_test)
```

```
results_GBR = pd.DataFrame(
    data=[["GradientBoostingRegressor", *evaluate(sj_Y_test, sj_Y_pred),*cross_val_1(GBR_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_GBR],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train,GBR_model.predict(sj_X_train))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 21.054582720469302
MSE: 1078.8321833849645
RMSE: 32.84558088061413
-----------------------
evaluation of sj_X_test:
MAE: 21.28912700069776
MSE: 1079.2596372571475
RMSE: 32.85208725875949
-----------------------
```

图 5-7 城市 sj 梯度提升回归

g. K 近邻回归

```
from sklearn.neighbors import KNeighborsRegressor
grid_GBR = GridSearchCV(estimator=KNeighborsRegressor(),
                   param_grid={'n_neighbors': [i for i in range(1, 20)]
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_GBR.fit(sj_X_Stan,sj_Y_Stan)
print(f'最优得分为：{grid_GBR.best_score_}\n最优参数为：{grid_GBR.best_params_}',)
```

```
最优得分为：-22.89607168787497
最优参数为：{'n_neighbors': 11}
```

```
KNN_model = KNeighborsRegressor(n_neighbors=11)
KNN_model.fit(sj_X_train_S,sj_Y_train_S)
sj_Y_pred = KNN_model.predict(sj_X_test_S)
```

```
results_KNN = pd.DataFrame(
    data=[["KNeighborsRegressor", *evaluate(sj_Y_test_S, sj_Y_pred),*cross_val(KNN_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)","CV(R2)"])
sj_results_df = pd.concat([sj_results_df,results_KNN],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(sj_Y_train_S,KNN_model.predict(sj_X_train_S))
print("evaluation of sj_X_test:")
print_evaluate(sj_Y_test_S,sj_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 19.54420579420579
MSE: 984.3945826900372
RMSE: 31.37506307069417
-----------------------
evaluation of sj_X_test:
MAE: 22.065573770491802
MSE: 1159.896220024387
RMSE: 34.05724915527364
-----------------------
```

图 5-8 城市 sjK 近邻回归

最终汇总各个回归模型的评价结果到 sj_results_df 中

```
sj_results_df
```

	Model	MAE	MSE	RMSE	CV(-MAE)	CV(R2)
0	LinearRegression	22.223938	1117.755902	33.432857	-22.645848	-0.637521
1	Ridge	22.257759	1118.915352	33.450192	-22.636063	-0.641864
2	Lasso	22.224373	1117.758678	33.432898	-22.645800	-0.637633
3	ElasticNet	22.237063	1152.137032	33.943144	-22.792327	-0.584247
4	XGBRegressor	21.355872	1131.693611	33.640654	-22.560550	-0.597935
5	GradientBoostingRegressor	21.289127	1079.259637	32.852087	-22.531417	-0.553101
6	KNeighborsRegressor	22.065574	1159.896220	34.057249	-23.054189	-0.930852

图 5-9 城市 sj 各模型评价指标的 Dataframe

可视化呈现城市 sj 各模型评价指标

```
# 可视化呈现城市sj各模型评估结果
sj_results_df_.loc[:,['MAE','RMSE','CV(-MAE)','CV(R2)']].plot(kind='barh',figsize=(15,7))
plt.grid(which='major', axis='x', linewidth=0.75, linestyle='-', color='0.75',dashes=(15,8))
plt.title('城市sj各回归模型的评估指标',fontsize=18)
plt.yticks(fontsize=13)
plt.ylabel('Model',fontsize=15)
plt.xlim(-25,45)
plt.legend(fontsize=12)
plt.show()
```

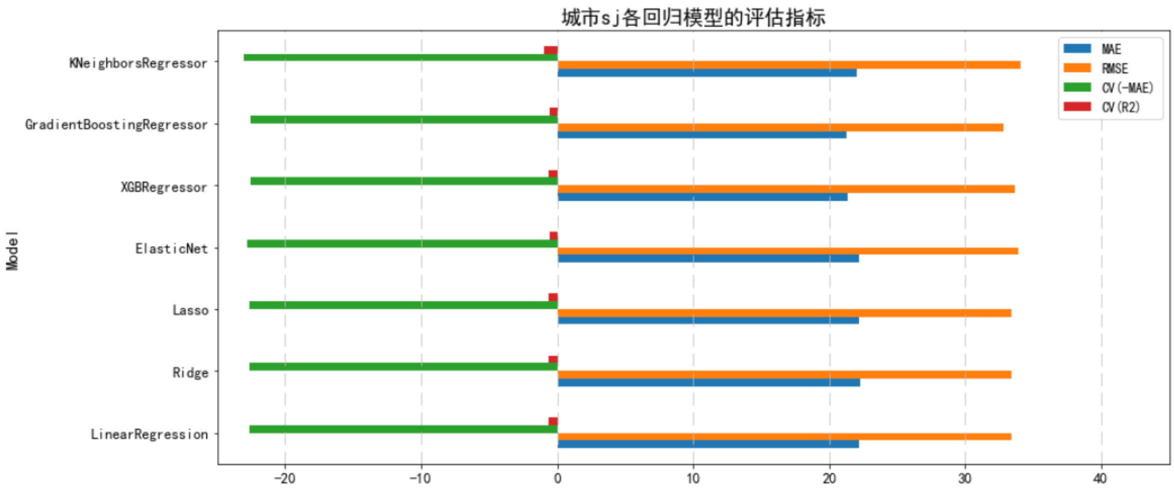


图 5-10 城市 sj 各回归模型的评估指标

结果分析：结合此次竞赛的评价指标是 MAE,综合上述各个模型的评价结果，选取 XGBoost 回归、K 近邻回归作为后续模型应用的备选方案。

2. 城市 iq

1)模型评价以及交叉验证函数(实现代码复用)

```
from sklearn import metrics
from sklearn.model_selection import cross_val_score
def cross_val(model):  # 交叉验证
    pred_mae = cross_val_score(model, iq_X_Stan, iq_Y_Stan, cv=6,scoring='neg_mean_absolute_error')
    pred_r2 = cross_val_score(model, iq_X_Stan, iq_Y_Stan, cv=6,scoring='r2')
    return pred_mae.mean(), pred_r2.mean()
def cross_val_1(model):  # 交叉验证
    pred_mae = cross_val_score(model, iq_X, iq_Y, cv=6,scoring='neg_mean_absolute_error')
    pred_r2 = cross_val_score(model, iq_X, iq_Y, cv=6,scoring='r2')
    return pred_mae.mean(), pred_r2.mean()
def print_evaluate(true, predicted):# 显示模型评估指标
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('-----------------------')
def evaluate(true, predicted):# 返回模型评估指标
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    return mae, mse, rmse
```

图 5-11 城市 iq 的模型评价函数等

2)模型建模,调优，训练，模型评价，交叉验证

a. 线性回归

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(iq_X_train_S,iq_Y_train_S)
iq_Y_pred = LR.predict(iq_X_test_S)
```

```
iq_results_df = pd.DataFrame(
    data=[["LinearRegression", *evaluate(iq_Y_test_S, iq_Y_pred),*cross_val(LinearRegression())]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])

print("evaluation of iq_X_train:")
print_evaluate(iq_Y_train_S,LR.predict(iq_X_train_S))
print("evaluation of iq_X_test:")
print_evaluate(iq_Y_test_S,iq_Y_pred)
```

```
evaluation of iq_X_train:
MAE: 6.011797799507716
MSE: 83.3809638357929
RMSE: 9.131317749141845
-----------------------
evaluation of iq_X_test:
MAE: 6.050467806154736
MSE: 76.58674814484941
RMSE: 8.75138549858532
-----------------------
```

图 5-12 城市 iq 线性回归

b. 岭回归

```
from sklearn.model_selection import GridSearchCV
grid_Ridge = GridSearchCV(estimator=Ridge(random_state=10),
                   param_grid={'alpha':[0.001,0.1,0.5,1,10]},
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_Ridge.fit(iq_X_Stan,iq_Y_Stan)
print(f'最优得分为：{grid_Ridge.best_score_}\n最优参数为：{grid_Ridge.best_params_}',)
```

```
最优得分为：-6.480476039828728
最优参数为：{'alpha': 0.001}
```

```
from sklearn.linear_model import Ridge
Ridge_Model = Ridge(alpha=0.001)
Ridge_Model.fit(iq_X_train_S,iq_Y_train_S)
iq_Y_pred = Ridge_Model.predict(iq_X_test_S)
```

```
results_Ridge = pd.DataFrame(
    data=[["Ridge", *evaluate(iq_Y_test_S, iq_Y_pred),*cross_val(Ridge_Model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])
iq_results_df = pd.concat([iq_results_df,results_Ridge],ignore_index=True)
print("evaluation of iq_X_train:")
print_evaluate(iq_Y_train_S,Ridge_Model.predict(iq_X_train_S))
print("evaluation of iq_X_test:")
print_evaluate(iq_Y_test_S,iq_Y_pred)
```

```
evaluation of iq_X_train:
MAE: 6.01188629280125
MSE: 83.3809856369654
RMSE: 9.131318942900057
-----------------------
evaluation of iq_X_test:
MAE: 6.04897242310999
MSE: 76.56683403954551
RMSE: 8.750247655897832
-----------------------
```

图 5-13 城市 iq 岭回归

c. Lasso 回归

```
#构造不同的Lambda值
alphas = np.logspace(0.0001,1,20)
#设置交叉验证的参数，使用均方误差评估
lasso_cv = LassoCV(alphas=alphas,cv=5,max_iter=100000)
lasso_cv.fit(iq_X_Stan,iq_Y_Stan)
# 获取最佳的Lasso回归alpha
print('Lasso回归最佳的alpha:',lasso_cv.alpha_)
```

```
Lasso回归最佳的alpha: 1.0002302850208247
```

```
Lasso_model = Lasso(alpha=lasso_cv.alpha_,max_iter=10000)
Lasso_model.fit(iq_X_train_S,iq_Y_train_S)
iq_Y_pred = Lasso_model.predict(iq_X_test_S)
```

```
evaluation of iq_X_train:
MAE: 6.166715127992656
MSE: 88.06128435965707
RMSE: 9.384097418487142
-----------------------
evaluation of iq_X_test:
MAE: 5.701296859312024
MSE: 72.0388559207694
RMSE: 8.487570672505143
-----------------------
```

图 5-14 城市 iq Lasso 回归

d. 弹性网回归

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
grid_ElasticNet = GridSearchCV(estimator=ElasticNet(random_state=10,max_iter=10000),
                    param_grid={'alpha':[0.001,0.01,0.1,1,10],
                                'l1_ratio':[0.3,0.5,0.7,0.9]},
                    cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_ElasticNet.fit(iq_X_Stan,iq_Y_Stan)
print(f'最优得分为：{grid_ElasticNet.best_score_}\n最优参数为：{grid_ElasticNet.best_params_}',)
```

```
最优得分为：-6.271177039312138
最优参数为：{'alpha': 1, 'l1_ratio': 0.5}
```

```
ElasticNet_Model = ElasticNet(alpha=1,l1_ratio=0.5,random_state=10,max_iter=10000)
ElasticNet_Model.fit(iq_X_train_S,iq_Y_train_S)
iq_Y_pred = ElasticNet_Model.predict(iq_X_test_S)
```

```
results_ElasticNet = pd.DataFrame(
    data=[["ElasticNet", *evaluate(iq_Y_test_S, iq_Y_pred),*cross_val(ElasticNet_Model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])
iq_results_df = pd.concat([iq_results_df,results_ElasticNet],ignore_index=True)
print("evaluation of iq_X_train:")
print_evaluate(iq_Y_train_S,ElasticNet_Model.predict(iq_X_train_S))
print("evaluation of iq_X_test:")
print_evaluate(iq_Y_test_S,iq_Y_pred)
```

```
evaluation of iq_X_train:
MAE: 6.149524608376098
MSE: 87.62225367949156
RMSE: 9.36067592001195
-----------------------
evaluation of iq_X_test:
MAE: 5.746940511928989
MSE: 72.18680408637783
RMSE: 8.496281780071671
-----------------------
```

图 5-15 城市 iq 弹性网回归

e. 梯度提升回归

```
from sklearn.ensemble import GradientBoostingRegressor
grid_GBR = GridSearchCV(estimator=GradientBoostingRegressor(random_state=10),
                   param_grid={'n_estimators':[100,150,200],'learning_rate':[0.01,0.1,0.3],
                               'max_depth':[2,3,4,5],'min_samples_split':[2,4,6,10],
                               'min_samples_leaf':[3,5,7,10,12],'subsample':[0.2,0.5]
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_GBR.fit(iq_X,iq_Y)
print(f'最优得分为: {grid_GBR.best_score_}\n最优参数为: {grid_GBR.best_params_}',)
```

```
最优得分为: -6.264109561565626
最优参数为: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_leaf': 12, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 0.5}
```

```
GBR_model = GradientBoostingRegressor(n_estimators=100,learning_rate=0.01, max_depth=2,subsample=0.5,
                                      min_samples_leaf=12, min_samples_split=2,random_state=10)
GBR_model.fit(iq_X_train,iq_Y_train)
iq_Y_pred = GBR_model.predict(iq_X_test)
```

```
results_GBR = pd.DataFrame(
    data=[["GradientBoostingRegressor", *evaluate(iq_Y_test, iq_Y_pred),*cross_val_1(GBR_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])
iq_results_df = pd.concat([iq_results_df,results_GBR],ignore_index=True)
print("evaluation of sj_X_train:")
print_evaluate(iq_Y_train,GBR_model.predict(iq_X_train))
print("evaluation of sj_X_test:")
print_evaluate(iq_Y_test,iq_Y_pred)
```

```
evaluation of sj_X_train:
MAE: 5.943200167917575
MSE: 82.48966447752893
RMSE: 9.082382092685208
-----------------------
evaluation of sj_X_test:
MAE: 5.702543560907818
MSE: 71.13652344610469
RMSE: 8.434247058635684
-----------------------
```

图 5-16 城市 iq 梯度提升回归

f. XGBoost 回归

```
from xgboost import XGBRegressor
#XGBR_model = XGBRegressor()
from sklearn.model_selection import GridSearchCV
grid_XGBR = GridSearchCV(estimator=XGBRegressor(random_state=10),
                   param_grid={'n_estimators':[100,200,300,400],'max_leaves':[2,4],'learning_rate':[0.01,0.02,0.04],
                               'max_depth':[1,2,3,4,5],'eta':[0.1,0.4,0.6],'subsample':[0.2,0.4,0.6]
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_XGBR.fit(iq_X,iq_Y)
print(f'最优得分为: {grid_XGBR.best_score_}\n最优参数为: {grid_XGBR.best_params_}',)
```

```
最优得分为: -5.6958264099764
最优参数为: {'eta': 0.1, 'learning_rate': 0.01, 'max_depth': 1, 'max_leaves': 2, 'n_estimators': 100, 'subsample': 0.2}
```

```
XGBR_model = XGBRegressor(n_estimators=100,eta=0.1,max_depth=2,learning_rate=0.01,
                          max_leaves=2,subsample=0.2,random_state=10)
XGBR_model.fit(iq_X_train,iq_Y_train)
iq_Y_pred = XGBR_model.predict(iq_X_test)
```

```
results_XGBRegressor = pd.DataFrame(
    data=[["XGBRegressor", *evaluate(iq_Y_test, iq_Y_pred),*cross_val_1(XGBR_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])
iq_results_df = pd.concat([iq_results_df,results_XGBRegressor],ignore_index=True)
print("evaluation of iq_X_train:")
print_evaluate(iq_Y_train,XGBR_model.predict(iq_X_train))
print("evaluation of iq_X_test:")
print_evaluate(iq_Y_test,iq_Y_pred)
```

```
evaluation of iq_X_train:
MAE: 5.452108984600644
MSE: 89.25826225564776
RMSE: 9.44765908866571
-----------------------
evaluation of iq_X_test:
MAE: 4.907334153275741
MSE: 72.28988517349528
RMSE: 8.502345862966013
-----------------------
```

图 5-17 城市 iq XGBoost 回归

g. K 近邻回归

```
from sklearn.neighbors import KNeighborsRegressor
grid_GBR = GridSearchCV(estimator=KNeighborsRegressor(),
                   param_grid={'n_neighbors': [i for i in range(1,20)]
                              },
                   cv=5,scoring='neg_mean_absolute_error',n_jobs=-1)
grid_GBR.fit(iq_X,iq_Y)
print(f'最优得分为：{grid_GBR.best_score_}\n最优参数为：{grid_GBR.best_params_}',)
```

```
最优得分为：-6.583183827429717
最优参数为：{'n_neighbors': 19}
```

```
KNN_model = KNeighborsRegressor(n_neighbors=19)
KNN_model.fit(iq_X_train,iq_Y_train)
iq_Y_pred = KNN_model.predict(iq_X_test)
```

```
results_KNN = pd.DataFrame(
    data=[["KNeighborsRegressor", *evaluate(iq_Y_test, iq_Y_pred),*cross_val(KNN_model)]],
    columns=['Model', 'MAE', 'MSE', 'RMSE', "CV(-MAE)", "CV(R2)"])
iq_results_df = pd.concat([iq_results_df,results_KNN],ignore_index=True)
print("evaluation of iq_X_train:")
print_evaluate(iq_Y_train,KNN_model.predict(iq_X_train))
print("evaluation of iq_X_test:")
print_evaluate(iq_Y_test,iq_Y_pred)
```

```
evaluation of iq_X_train:
MAE: 6.022197403322631
MSE: 82.93614848233244
RMSE: 9.106928597630073
-----------------------
evaluation of iq_X_test:
MAE: 6.016066481994458
MSE: 76.73920396559264
RMSE: 8.760091550069134
-----------------------
```

图 5-18 城市 iq K 近邻回归

最终汇总城市 iq 的各个回归模型的评价结果到 iq_results_df 中

```
iq_results_df
```

	Model	MAE	MSE	RMSE	CV(-MAE)	CV(R2)
0	LinearRegression	6.050468	76.586748	8.751385	-6.611453	-0.579639
1	Ridge	6.048972	76.566834	8.750248	-6.613863	-0.577599
2	Lasso	5.701297	72.038856	8.487571	-6.240993	-0.544622
3	ElasticNet	5.746941	72.186804	8.496282	-6.256420	-0.524576
4	GradientBoostingRegressor	5.702544	71.136523	8.434247	-6.269202	-0.494313
5	XGBRegressor	4.907334	72.289885	8.502346	-5.691087	-0.227148
6	KNeighborsRegressor	6.016066	76.739204	8.760092	-6.509975	-0.581983

图 5-19 城市 iq 各模型评价指标的 Dataframe

可视化呈现城市 iq 各模型评价指标

```
# 可视化呈现城市iq各模型评估结果
iq_results_df_.loc[:,['MAE','CV(-MAE)','RMSE','CV(R2)']].plot(kind='barh',figsize=(12,7))
plt.grid(which='major', axis='x', linewidth=0.75, linestyle='-', color='0.75',dashes=(15,8))
plt.title('城市iq各回归模型的评估指标',fontsize=18)
plt.yticks(fontsize=13)
plt.ylabel('Model',fontsize=15)
plt.xlim(-10,15)
plt.legend(fontsize=12)
plt.show()
```

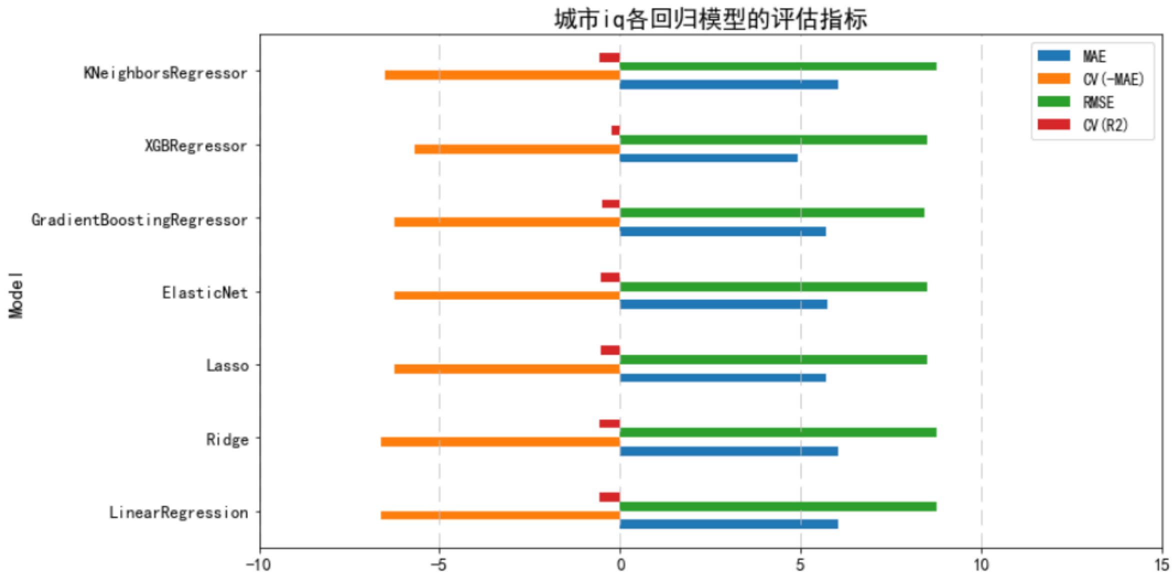


图 5-20 城市 iq 各回归模型的评估指标

结果分析：结合竞赛的评价指标是 MAE,综合上述各个模型的评价结果，选取 XGBRegressor 和 Lasso 作为后续模型应用的模型。

六、模型应用

1. 城市 sj

1）XGBoost 回归(XGBoostRegressor)

```
# 预测
sj_test_pred = XGBR_model.predict(sj_test_X)
# 转换预测数据
sj_test_pred = sj_test_pred.tolist()
sj_test_pred = [int(x) for x in sj_test_pred ]
# 重构测试数据结果
temp = sj_test_X.reset_index()
temp['city'] = 'sj'
temp['total_cases'] = sj_test_pred
# 输出测试提交结果
sj_submission_XGBR = temp.loc[:,['city','year','weekofyear','total_cases']]
sj_submission_XGBR
```

图 6-1 城市 sj XGBoost 回归模型应用

2)K 近邻回归(KNeighborsRegressor)

```
# 预测
sj_test_pred = KNN_model.predict(sj_test_X_Stan)
# 转换预测数据
sj_test_pred = sj_test_pred.tolist()
sj_test_pred = [int(x) for x in sj_test_pred ]
# 重构测试数据结果
temp = sj_test_X_Stan.reset_index()
temp['city'] = 'sj'
temp['total_cases'] = sj_test_pred
# 输出测试提交结果
sj_submission_KNN = temp.loc[:,['city','year','weekofyear','total_cases']]
sj_submission_KNN
```

图 6-2 城市 sj K 近邻回归模型应用

3)XGBoost 回归(XGBRegressor)

```
# 预测
sj_test_pred = XGBR_model.predict(sj_test_X)
# 转换预测数据
sj_test_pred = sj_test_pred.tolist()
sj_test_pred = [int(x) for x in sj_test_pred ]
# 重构测试数据结果
temp = sj_test_X.reset_index()
temp['city'] = 'sj'
temp['total_cases'] = sj_test_pred
# 输出测试提交结果
sj_submission_XGBR = temp.loc[:,['city','year','weekofyear','total_cases']]
sj_submission_XGBR
```

图 6-3 城市 sj XGBoost 回归模型应用

2.城市 iq

1) XGBoost 回归(XGBRegressor)

```
# 预测
iq_test_pred = XGBR_model.predict(iq_test_X)
# 转换预测数据
iq_test_pred = iq_test_pred.tolist()
iq_test_pred = [int(x) for x in iq_test_pred ]
# 重构测试数据结果
temp = iq_test_X.reset_index()
temp['city'] = 'iq'
temp['total_cases'] = iq_test_pred
# 输出测试提交结果
iq_submission_XGBR = temp.loc[:,['city','year','weekofyear','total_cases']]
iq_submission_XGBR
```

图 6-4 城市 iq XGBoost 回归模型应用

2）Lasso 回归

```python
# 预测
iq_test_pred = Lasso_model.predict(iq_test_X_Stan)
# 转换预测数据
iq_test_pred = iq_test_pred.tolist()
iq_test_pred = [int(x) for x in iq_test_pred ]
# 重构测试数据结果
temp = iq_test_X.reset_index()
temp['city'] = 'iq'
temp['total_cases'] = iq_test_pred
# 输出测试提交结果
iq_submission_Lasso = temp.loc[:,['city','year','weekofyear','total_cases']]
iq_submission_Lasso
```

图 6-5 城市 iq Lasso 回归模型应用

七、数据分析结论

1. 模型应用结果：

5折交叉

sj:XGBR,iq:XGBR,MAE=28.4423

sj:GBR,iq:XGBR;MAE=26.6418

sj:KNN,iq:XGBR,MAE=26.2091

sj:KNN,iq:Lasso,MAE=27.5385

10折交叉

sj:GBR,iq:XGBR;MAE=26.8077

sj:KNN,iq:XGBR,MAE=26.3077

26.6418	Garvey	2023-06-04 01:34:43 UTC
26.2091	Garvey	2023-06-04 00:21:36 UTC
28.4423	Garvey	2023-06-06 05:35:02 UTC
26.3077	Garvey	2023-06-05 00:04:00 UTC
26.8077	Garvey	2023-06-05 01:07:57 UTC

图 7-1 模型应用结果

2. 参与 DRIVENDATA 平台上正在进行中的竞赛的结果

提交评价结果：2430/13416；排名前 18%

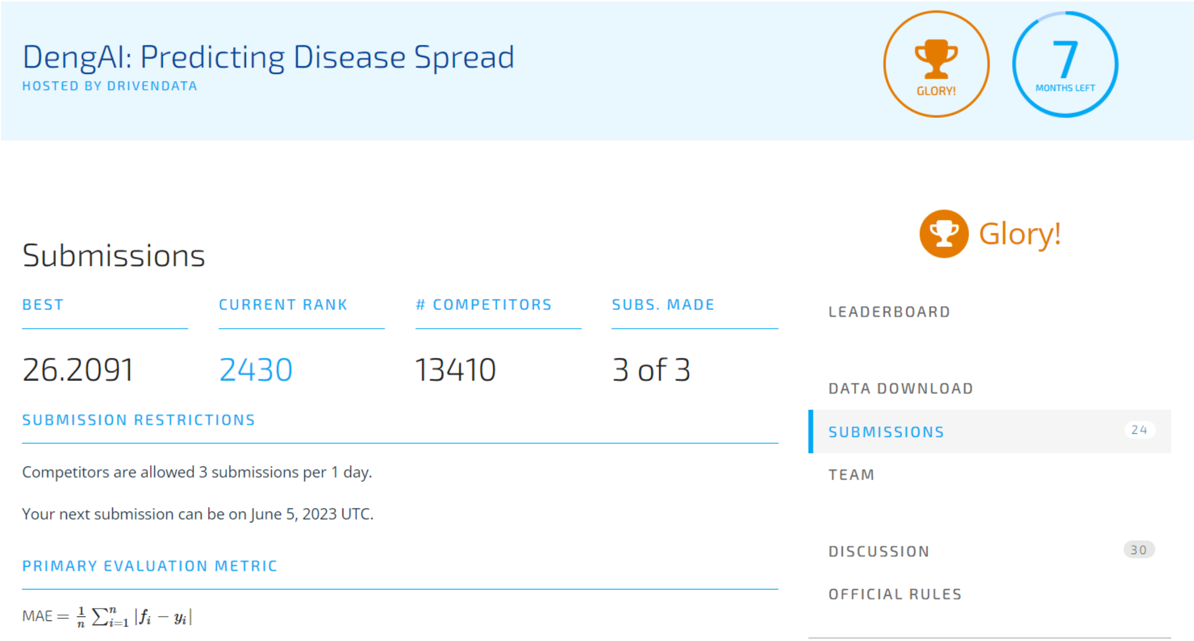


图 7-2 DRIVENDATA 平台的竞赛结果

3. 结论：

1) 结果总结：

综上所述，对于城市 San Juan(sj)采用模型 KneighborsRegressor，对于城市 Iquitos(iq)采用模型 XGBRegressor，并且进行提交竞赛平台使用隐藏测试集进行模型应用，综合 MAE 指标评价最优为 26.2091。

2) 问题总结：针对前文提出的问题：

1. 对于某些数据大量丢失的数据，结合此次是疾病预测的主题，本文认为应该做甄别删除操作，避免因为某些不正当的填充影响到模型的后续应用。

2. 对于来自不同数据来源的相似或者同一维度的特征数据，支持进行特征选择，但是不建议采取取其一操作，因为不能确定该维度特征是否是影响疾病传播的重要特征，从另外的方面，不同来源的同一维度特征对于数据的准确性提供了一定的保证。

3) 两城市模型的重要变量情况

对于城市 San Juan(sj)，筛选出的重要变量：

['reanalysis_dew_point_temp_k' 'reanalysis_max_air_temp_k''reanalysis_min_air_temp_k' 'reanalysis_air_temp_k''reanalysis_avg_temp_k' 'station_min_temp_c']

分别表示：平均露点温度 ，最高气温，最低气温(r)，气温，平均温度，最低温度(s)。

对于城市 Iquitos(iq)，筛选出的重要变量：

['reanalysis_specific_humidity_g_per_kg' 'reanalysis_dew_point_temp_k'
'station_min_temp_c' 'reanalysis_min_air_temp_k''station_precip_mm'
'reanalysis_relative_humidity_percent'],

分别表示：平均比湿，平均露点温度 ，最低温度(s)，最低气温(r),总降水量，平均相对湿度。

4)分析结论

从上述两个城市各自的模型重要变量的共同点来看，都包含：平均露点温度，最低气温,从这里可以得出上述共性特征是登革热疾病传播的重要影响因素,这些特征影响到登革热传播媒介:蚊子的活动。

从两个城市的不同点分布来看：

a．城市 San Juan(sj)的登革热疾病传播与各项气温特征有着紧密的联系，如气温，最高气温等，总体上看该城市的登革热疾病传播与气温维度的特征关系更明显、紧密。

b.城市 Iquitos(iq) 的登革热疾病传播与各项环境湿度指标联系紧密，如平均相对湿度，总降水量等，总体上看该城市的登革热疾病传播与环境湿度维度的特征关系更为明显、紧密

附录

数据：

训练集特征数据(前 15 行)

	city	year	weekofy...	week_st...	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	prec...	reanalysis_air...	reanalysis_a...	reanal...	rea...	rea...	re...	reanal...	rean...	reana...	reanal...	station_...	station...
1	sj	1990	18	1990-04...	0.1226	0.103...	0.1984...	0.177...	12.42	297.572857143	297.742857...	292.41...	299.8	295.9	32.0	73.365...	12.42	14.01...	2.6285...	25.4428...	6.9
2	sj	1990	19	1990-05...	0.1699	0.142...	0.1623...	0.155...	22.82	298.211428571	298.442857...	293.95...	300.9	296.4	17...	77.368...	22.82	15.37...	2.3714...	26.7142...	6.3714...
3	sj	1990	20	1990-05...	0.032...	0.172...	0.1572	0.170...	34.54	298.781428571	298.878571...	295.43...	300.5	297.3	26.1	82.052...	34.54	16.84...	2.3	26.7142...	6.4857...
4	sj	1990	21	1990-05...	0.128...	0.245...	0.2275...	0.235...	15.36	298.987142857	299.228571...	295.31	301.4	297.0	13.9	80.337...	15.36	16.67...	2.4285...	27.4714...	6.7714...
5	sj	1990	22	1990-05...	0.1962	0.2622	0.2512	0.247...	7.52	299.518571429	299.664285...	295.82...	301.9	297.5	12.2	80.46	7.52	17.21	3.0142...	28.9428...	9.3714...
6	sj	1990	23	1990-06...		0.17485	0.2543...	0.181...	9.58	299.63	299.764285...	295.85...	302.4	298.1	26...	79.891...	9.58	17.21...	2.1	28.1142...	6.9428...
7	sj	1990	24	1990-06...	0.1129	0.0928	0.2050...	0.210...	3.48	299.207142857	299.221428...	295.86...	301.3	297.7	38.6	82.0	3.48	17.23...	2.0428...	27.4142...	6.7714...
8	sj	1990	25	1990-06...	0.0725	0.0725	0.1514...	0.133...	151....	299.591428571	299.528571...	296.53...	300.6	298.4	30.0	83.375...	151....	17.97...	1.5714...	28.3714...	7.6857...
9	sj	1990	26	1990-06...	0.102...	0.146...	0.1255...	0.1236	19.32	299.578571429	299.557142...	296.37...	302.1	297.7	37...	82.768...	19.32	17.79	1.8857...	28.3285...	7.3857...
10	sj	1990	27	1990-07...		0.12155	0.1606...	0.202...	14.41	300.154285714	300.278571...	296.65...	302.3	298.7	28.4	81.281...	14.41	18.07...	2.0142...	28.3285...	6.5142...
11	sj	1990	28	1990-07...	0.192...	0.08235	0.1919...	0.152...	22.27	299.512857143	299.592857...	296.04...	301.8	298.0	43...	81.467...	22.27	17.41...	2.1571...	27.5571...	7.1571...
12	sj	1990	29	1990-07...	0.2916	0.2118	0.3012	0.280...	59.17	299.667142857	299.75	296.33...	302.0	297.3	40.9	82.144...	59.17	17.73...	2.4142...	28.1285...	6.9
13	sj	1990	30	1990-07...	0.150...	0.1717	0.2269	0.214...	16.48	299.558571429	299.635714...	295.96	301.8	297.1	42...	80.742...	16.48	17.34...	2.0714...	28.1142...	6.3571...
14	sj	1990	31	1990-07...		0.24715	0.3797	0.381...	32.66	299.862857143	299.95	296.17...	303.0	298.3	34.6	80.584...	32.66	17.59...	2.5857...	28.2428...	8.0857...
15	sj	1990	32	1990-08...		0.064...	0.1644...	0.138...	28.8	300.391428571	300.478571...	296.53...	302.5	298.8	20.0	79.65	28.8	17.95	2.3285...	28.2	7.5571...

训练集目标特征数据(前 15 行)

	city	year	weekofyear	total_cases
1	sj	1990	18	4
2	sj	1990	19	5
3	sj	1990	20	4
4	sj	1990	21	3
5	sj	1990	22	6
6	sj	1990	23	2
7	sj	1990	24	4
8	sj	1990	25	5
9	sj	1990	26	10
10	sj	1990	27	6
11	sj	1990	28	8
12	sj	1990	29	2
13	sj	1990	30	6
14	sj	1990	31	17
15	sj	1990	32	23

测试集特征数据(前 15 行)

	city	year	weeko...	w...	ndvi...	ndvi_nw	ndvi_se	ndvi_sw	prec...	reanalysis_ai...	reanal...	reanal...	rea...	rean...	rean...	reana...	rea...	reanalys...	rean...	st...	sta...	stat...	statio...
1	sj	2008	18	20...	-0.0...	-0.0189	0.10272...	0.0912	78.6	298.4928571...	298.55	294.52...	301.1	296.4	25.37	78.78...	78.6	15.9185...	3.128...	26...	7.0...	33.3	21.7
2	sj	2008	19	20...	-0.0...	-0.0124	0.08204...	0.0723...	12.56	298.4757142...	298.5...	294.39...	300.8	296.7	21.83	78.23	12.56	15.7914...	2.571...	26...	5.5...	30.0	22.2
3	sj	2008	20	20...	-0.0...		0.15108...	0.0915...	3.66	299.4557142...	299.3...	295.30...	302.2	296.4	4.12	78.27	3.66	16.6742...	4.428...	27...	7.7...	32.8	22.8
4	sj	2008	21	20...		-0.019...	0.12432...	0.1256...	0.0	299.69	299.7...	294.40...	303.0	296.9	2.2	73.01...	0.0	15.7757...	4.342...	28...	6.2...	33.3	24.4
5	sj	2008	22	20...	0.05...	0.039...	0.06226...	0.0759...	0.76	299.78	299.6...	294.76	302.3	297.3	4.36	74.08...	0.76	16.1371...	3.542...	27...	7.0...	33.3	23.3
6	sj	2008	23	20...	-0.0...	-0.030...	0.132	0.0835...	71.17	299.7685714...	299.7...	295.31...	301.9	297.6	22.55	76.55...	71.17	16.6671...	2.857...	28.0	5.1...	32.8	25.0
7	sj	2008	24	20...	-0.0...	-0.024...	0.13227...	0.1591...	48.99	300.0628571...	300.0...	295.65	302.4	297.5	13.1	76.84...	48.99	17.01	3.157...	27.4	6.0...	31.1	23.3
8	sj	2008	25	20...		0.08215	0.14437...	0.1167...	30.81	300.4842857...	300.5...	295.99...	303.5	297.5	7.2	76.87	30.81	17.42	3.9	28...	6.9...	34.4	24.4
9	sj	2008	26	20...	0.01...	0.0499	0.10057...	0.1173...	8.02	300.6014285...	300.6...	296.26...	302.5	298.5	17.1	77.39...	8.02	17.6785...	2.785...	28...	6.2...	32.8	23.9
10	sj	2008	27	20...	0.07...	0.10666	0.15542...	0.1649	17.52	300.4971428...	300.5...	296.41...	302.3	298.7	11.9	78.53...	17.52	17.8085...	2.228...	28...	4.6...	31.1	25.0
11	sj	2008	28	20...	-0.0...	0.006...	0.26028...	0.2147...	16.37	300.2142857...	300.3...	295.82...	301.7	299.0	19.86	77.02...	16.37	17.2014...	2.028...	27...	5.9...	31.1	23.9
12	sj	2008	29	20...			0.19584...	0.1761...	4.34	300.4485714...	300.6...	296.17...	302.3	298.7	5.49	77.61...	4.34	17.5714...	2.614...	28...	5.0...	31.7	26.1
13	sj	2008	30	20...	0.20...	0.4295	0.27768...	0.24565	3.39	300.5985714...	300.7...	296.51...	302.3	298.9	13.62	78.56	3.39	17.94	2.585...	28...	6.3...	32.8	24.4
14	sj	2008	31	20...	0.00...	0.0039	0.10906...	0.0864...	13.73	300.73	300.8...	296.20...	302.7	299.2	8.7	76.52...	13.73	17.6085...	2.8	28...	6.1...	32.8	25.0
15	sj	2008	32	20...	0.11...	0.0322	0.19418...	0.2057...	50.94	300.7414285...	300.8...	297.04...	302.5	299.1	43.5	80.37...	50.94	18.5471...	2.428...	28...	5.8...	32.2	23.9

42