## 2.7.2 时间序列分析

### 2.7.2.1 AoT预处理后温度数据读取与时间序列图表打印

AoT预处理后温度数据时间跨度为2020年12个月，时间精度为1小时，庞大的数据量很难通过对数据的直接查看洞悉数据整体概况和局部变化情况，因此需要借助图表形式直观观察。查看数据的目的初步包括：1，不同节点时间轴向上的温度数值变化，可以观察不同时间尺度的数值变化（例如小时、日、周或月，和任意时间区段等）。通常因为节点位置环境不同，不同节点温度变化应存在差异；2，查看各个节点测量数值的时间轴向分布区段。一方面，确定单独节点轴向上测量数据的时间跨度和断续情况，较高的跨度和连续度有利于处理潜在更多的数据分析内容和更高的分析精度。再者，纵向上观察各节点测量数据的对应情况，如果同一时间段更多的节点均具有数据，这有利于节点间影响因素的分析；3，需要能够通过节点测量值观察图表，对应到目标节点空间分布位置，确定节点所在环境，对应城市相关空间因素（例如建筑分布和高度、植被覆盖度、节点所在全景图等），分析测量值的空间因素和空间关系，及时空分析等。

参数管理使用AttrDict()方法（具体查看“Cityscapes数据集——参数管理”一节），首先配置读写PostgreSQL数据库参数于db子属性下，并读取在“城市环境传感器AoT（SAGE）数据预处理”一节中预处理写入到数据库表名为“temperature”的温度数据。

from database import postSQL2gpd,gpd2postSQL  
from util\_misc import AttrDict  
\_\_C=AttrDict()   
args=\_\_C  
  
\_\_C.db=AttrDict()   
\_\_C.db.UN='postgres'  
\_\_C.db.PW='123456'  
\_\_C.db.DB='AoT\_20220831'  
\_\_C.db.GC='geometry'   
\_\_C.db.db\_info=dict(geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)

aot\_temperature\_gdf=postSQL2gpd(table\_name="temperature",geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)  
print(f"aot\_temperature\_gdf rows:{len(aot\_temperature\_gdf)}")  
aot\_temperature\_gdf.head(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is temperature.  
aot\_temperature\_gdf rows:523877

|  | **timestamp** | **node\_id** | **subsystem** | **sensor** | **parameter** | **value\_hrf\_sum** | **value\_hrf\_count** | **value\_hrf\_average** | **value\_hrf\_min** | **value\_hrf\_max** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** | **ts** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2020/01/01 00:30:00 | 001e06112e77 | lightsense | tmp421 | temperature | -8.52 | 142 | -0.060 | -0.06 | -0.06 | AoT\_Chicago | 06B | Ashland Ave & 59th St Chicago IL | 41.786756 | -87.664343 | AoT Chicago (S) [C] | 2018/02/26 00:00:00 | None | POINT (444797.242 4626313.694) | 2020-01-01 00:30:00 |
| **1** | 2020/01/01 00:30:00 | 001e061183bf | metsense | htu21d | temperature | -203.42 | 143 | -1.423 | -1.58 | -1.21 | AoT\_Chicago | 11A | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | None | POINT (445239.085 4631090.243) | 2020-01-01 00:30:00 |
| **2** | 2020/01/01 00:30:00 | 001e061183bf | metsense | pr103j2 | temperature | -143.40 | 143 | -1.003 | -1.25 | -0.65 | AoT\_Chicago | 11A | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | None | POINT (445239.085 4631090.243) | 2020-01-01 00:30:00 |

用于同一测量内容的传感器，通常有多个不同类型，因为不同类型传感器的测量精度不同，同一时间下不同传感器测量值存在差异，例如测量温度的传感器由字段subsystem和sensor确定，通过读取“sensors.csv”文件，提取parameter字段为temperature值，可以查看测量温度包含的所有传感器类型。

import pandas as pd  
  
AoT\_sensors\_fp='./data/AoT\_Chicago.complete.2022-08-31/sensors.csv'  
AoT\_sensors=pd.read\_csv(AoT\_sensors\_fp)  
AoT\_sensors\_temperature=AoT\_sensors[AoT\_sensors["parameter"]=="temperature"]  
AoT\_sensors\_temperature

|  | **ontology** | **subsystem** | **sensor** | **parameter** | **hrf\_unit** | **hrf\_minval** | **hrf\_maxval** | **datasheet** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **26** | /sensing/meteorology/temperature | metsense | bmp180 | temperature | C | -40.0 | 85.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **27** | /sensing/meteorology/temperature | metsense | htu21d | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **28** | /sensing/meteorology/temperature | metsense | pr103j2 | temperature | C | -55.0 | 80.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **29** | /sensing/meteorology/temperature | metsense | tmp112 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **30** | /sensing/meteorology/temperature | metsense | tsys01 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **57** | /system/environment/temperature | lightsense | hih6130 | temperature | C | -25.0 | 85.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **59** | /system/environment/temperature | wagman | htu21d | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **65** | /system/environment/temperature | lightsense | tmp421 | temperature | C | -55.0 | 127.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **74** | /system/other/temperature | chemsense | lps25h | temperature | C | -30.0 | 105.0 | https://github.com/waggle-sensor/sensors/raw/m... |
| **76** | /system/other/temperature | chemsense | at0 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors |
| **77** | /system/other/temperature | chemsense | at1 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors |
| **78** | /system/other/temperature | chemsense | at2 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors |
| **79** | /system/other/temperature | chemsense | at3 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors |
| **80** | /system/other/temperature | chemsense | sht25 | temperature | C | -40.0 | 125.0 | https://github.com/waggle-sensor/sensors/raw/m... |

为了查看不同温度传感器的测量值，提取同一节点，例如001e06113107，截取开始的2020/01/01 00:30:00时间段测量值，可以初步查看得知部分传感器测量值均值value\_hrf\_average偏高，例如“bmp180”为7.773、“hih6130”为4.674等，且各个时间段获取测量值的各类传感器类型可能不同，因此直接用所有各类传感器的测量值做分析可能并不准确。同时，从各个传感器获取的数据量存在差异，例如“pr103j2”采样数为85584条，而“tsys01”为46351条。因此，需要均衡上述各影响因素，确定提取一类传感器测量值或者组合几类用于分析。

t\_sample\_nodeID=aot\_temperature\_gdf[aot\_temperature\_gdf["node\_id"]=="001e06113107"]  
t\_sample\_nodeID[:11]

|  | **timestamp** | **node\_id** | **subsystem** | **sensor** | **parameter** | **value\_hrf\_sum** | **value\_hrf\_count** | **value\_hrf\_average** | **value\_hrf\_min** | **value\_hrf\_max** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** | **ts** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **97** | 2020/01/01 00:30:00 | 001e06113107 | chemsense | at0 | temperature | -203.57 | 142 | -1.434 | -1.71 | -1.24 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **98** | 2020/01/01 00:30:00 | 001e06113107 | chemsense | at1 | temperature | -165.22 | 142 | -1.164 | -1.46 | -0.95 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **99** | 2020/01/01 00:30:00 | 001e06113107 | chemsense | at2 | temperature | -90.47 | 142 | -0.637 | -0.97 | -0.38 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **100** | 2020/01/01 00:30:00 | 001e06113107 | chemsense | at3 | temperature | -17.60 | 142 | -0.124 | -0.49 | 0.15 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **101** | 2020/01/01 00:30:00 | 001e06113107 | chemsense | lps25h | temperature | 392.73 | 142 | 2.766 | 2.43 | 3.02 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **102** | 2020/01/01 00:30:00 | 001e06113107 | lightsense | hih6130 | temperature | 663.73 | 142 | 4.674 | 4.50 | 4.86 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **103** | 2020/01/01 00:30:00 | 001e06113107 | lightsense | tmp421 | temperature | 271.96 | 142 | 1.915 | 1.75 | 2.12 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **104** | 2020/01/01 00:30:00 | 001e06113107 | metsense | bmp180 | temperature | 1103.80 | 142 | 7.773 | 7.40 | 8.20 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **105** | 2020/01/01 00:30:00 | 001e06113107 | metsense | htu21d | temperature | -189.98 | 142 | -1.338 | -1.69 | -0.95 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **106** | 2020/01/01 00:30:00 | 001e06113107 | metsense | pr103j2 | temperature | -256.60 | 142 | -1.807 | -2.35 | -1.35 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |
| **107** | 2020/01/01 00:30:00 | 001e06113107 | metsense | tsys01 | temperature | -192.52 | 142 | -1.356 | -1.79 | -0.79 | AoT\_Chicago | 072 | 7801 S Lawndale Ave Chicago IL | 41.751142 | -87.71299 | AoT Chicago (S) [CA] {ComEd} | 2017/01/01 00:00:00 | None | POINT (440722.179 4622392.015) | 2020-01-01 00:30:00 |

这里提取subsystem为metsense，sensor为pr103j2的采样数据用于分析。定义\_\_C的子属性aot，用于存储传感器相关信息。

\_\_C.aot=AttrDict()   
\_\_C.aot.subsystem="metsense"   
\_\_C.aot.sensor="pr103j2" #"tsys01"（46351条） "tmp112"（41326条） "pr103j2"（85584条） "htu21d"（45864条） "bmp180"（测量数据不准确）   
  
aot\_temperature\_subsystemNsensor\_gdf=aot\_temperature\_gdf[(aot\_temperature\_gdf.subsystem==args.aot.subsystem)&(aot\_temperature\_gdf.sensor==args.aot.sensor)]  
print(f"aot\_temperature\_subsystemNsensor\_gdfrows:{len(aot\_temperature\_subsystemNsensor\_gdf)}")  
gpd2postSQL(aot\_temperature\_subsystemNsensor\_gdf,table\_name="temperature\_metsense\_pr103j2",myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)   
aot\_temperature\_subsystemNsensor\_gdf.head(3)

aot\_temperature\_subsystemNsensor\_gdfrows:85584  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is temperature\_metsense\_pr103j2.

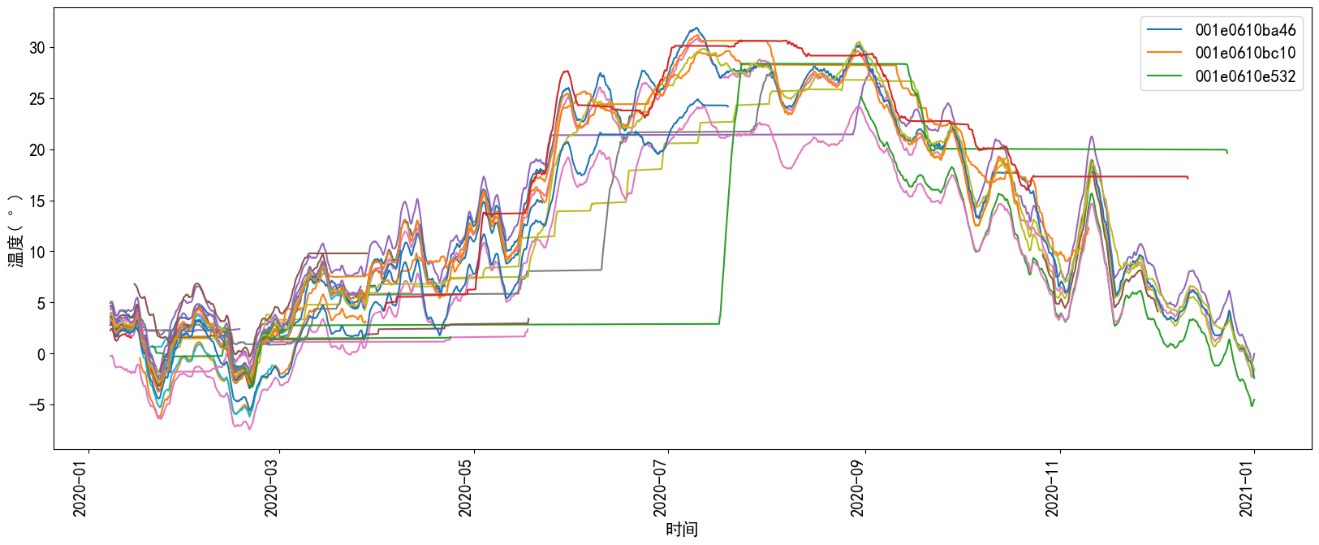
|  | **timestamp** | **node\_id** | **subsystem** | **sensor** | **parameter** | **value\_hrf\_sum** | **value\_hrf\_count** | **value\_hrf\_average** | **value\_hrf\_min** | **value\_hrf\_max** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** | **ts** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | 2020/01/01 00:30:00 | 001e061183bf | metsense | pr103j2 | temperature | -143.40 | 143 | -1.003 | -1.25 | -0.65 | AoT\_Chicago | 11A | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | None | POINT (445239.085 4631090.243) | 2020-01-01 00:30:00 |
| **6** | 2020/01/01 00:30:00 | 001e06117b41 | metsense | pr103j2 | temperature | -188.90 | 144 | -1.312 | -1.65 | -0.95 | AoT\_Chicago | 0BA | 60th & Michigan Ave Chicago IL | 41.785715 | -87.622214 | AoT Chicago (S) [CP] | 2019/05/16 00:00:00 | None | POINT (448297.079 4626171.925) | 2020-01-01 00:30:00 |
| **33** | 2020/01/01 00:30:00 | 001e0611462f | metsense | pr103j2 | temperature | -622.25 | 143 | -4.351 | -4.65 | -3.75 | AoT\_Chicago | 081 | Wallace St & Pershing Rd Chicago IL | 41.823527 | -87.641054 | AoT Chicago (S) [C] | 2018/02/27 00:00:00 | None | POINT (446762.872 4630381.536) | 2020-01-01 00:30:00 |

首先尝试用matplotlib库打印时间序列数据，定义time\_series\_plot()函数。因为一起打印较大的数据量不易观察，因此提供rolling\_window参数用于数据平滑，这里配置平滑参数为24\*7。从打印结果来看各个节点因为所处环境不同，同一时间段温度存在差异，但整体数值分布符合冬冷夏暖的变化趋势。

def time\_series\_plot(df\_,datetime\_column,val\_column,group\_column,span=None,rolling\_window=None,figsize=(10,10),font\_size=10,legend=True,\*\*setting):  
 '''  
 打印所有列或给定列的时间序列数据。  
  
 Parameters  
 ----------  
 df\_ : DataFrame  
 计算数据.  
 datetime\_column : string  
 时间列.  
 val\_column : string  
 数值列.  
 group\_column : string  
 分组（分类）列.   
 span : list(string), optional  
 打印时间区间，（表述时间的字符串）. The default is None.  
 rolling\_window : int, optional  
 数据平滑. The default is None.  
 figsize : tuple(int,float), optional  
 图表大小. The default is (10,10).  
 font\_size : int or float, optional  
 字体大小. The default is 10.  
 legend : bool, optional  
 是否打印图例. The default is True.  
 \*\*setting : kw-argument(string)  
 包括xlabel，ylabel，title.  
  
 Returns  
 -------  
 df\_span : DataFrame  
 提取时间区间内的数据行.  
 '''  
 import matplotlib.pyplot as plt  
 from pylab import mpl  
 import matplotlib.dates as md  
 from matplotlib.dates import DayLocator, HourLocator, DateFormatter, drange  
   
 plt.rc('axes', unicode\_minus=False) # 解决图表负号不正确显示问题  
 mpl.rcParams['font.sans-serif']=['SimHei'] # 解决中文字符乱码问题  
 plt.figure()  
 plt.rcParams.update({'font.size':font\_size}) # must set in top   
   
 setting\_dict={"xlabel":"X",  
 "ylabel":"Y",  
 "title":None,  
 "xticks\_rotation":0  
 }  
 setting\_dict.update(setting)   
 fig, ax=plt.subplots(figsize=figsize)  
   
 df=df\_.copy(deep=True)  
 df.sort\_values(by=[datetime\_column],inplace=True)  
 df.set\_index(datetime\_column,inplace=True)   
   
 if span:  
 df\_span=df[span[0]:span[1]]   
 else:  
 df\_span=df   
   
 df\_selection=df\_span[[val\_column,group\_column]]  
 df\_selection\_group=df\_selection.groupby(group\_column)  
 labels=[]  
 for label,g in df\_selection\_group:   
 labels.append(label)  
 g.drop([group\_column],axis=1,inplace=True)  
 if rolling\_window:  
 df\_span\_rolling=g.rolling(rolling\_window).mean()   
 df\_span\_rolling.plot(ax=ax,label=label,legend=legend)   
 else:  
 g.plot(ax=ax,label=label,legend=legend)   
   
 # https://matplotlib.org/stable/api/\_as\_gen/matplotlib.axes.Axes.set.html   
 ax.set(xlabel=setting\_dict["xlabel"],  
 ylabel=setting\_dict["ylabel"],  
 title=setting\_dict["title"],  
 )   
 if legend:  
 ax.legend(labels[:3])  
 plt.xticks(rotation=setting\_dict["xticks\_rotation"])  
   
 return df\_span

\_=time\_series\_plot(aot\_temperature\_subsystemNsensor\_gdf,'ts',"value\_hrf\_average","node\_id",rolling\_window=24\*7,xlabel="时间",ylabel="温度( °)",figsize=(20,8),font\_size=15,legend=True,xticks\_rotation=90)

<Figure size 640x480 with 0 Axes>

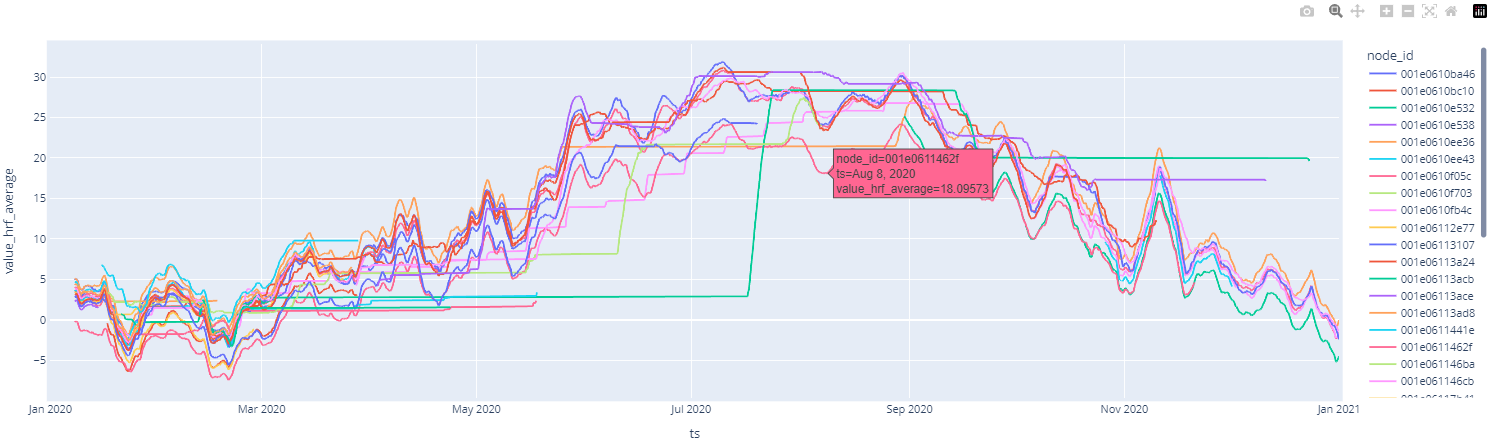


上述图表仅显示了3个图例用于说明，虽然能够以颜色区分不同节点测量值，但是对于较多的节点数量，并不易于用颜色区分查看各条测量值属于哪一个节点，因此可以借助Plotly库提供的交互方式查看数据。同样平滑数据减少干扰，定义函数df\_group\_rolling\_window()实现，用plotly.express.line()方法打印，当鼠标置于图上某一曲线时，会弹出信息框，其中包括节点ID信息，从而方便直观的查看数据属于哪个节点，及对应的具体时间和温度值为多少。

def df\_group\_rolling\_window(df\_,datetime\_column,val\_column,group\_column,rolling\_window=24):  
 '''  
 指定时间列、数值列和分组列，按分组列对数值列执行给定跨度（rolling\_window）的数据平滑  
  
 Parameters  
 ----------  
 df\_ : DataFrame  
 包含时间列、数值列和分组列的数据.  
 datetime\_column : string  
 时间列.  
 val\_column : string  
 数值列.  
 group\_column : string  
 分组列.  
 rolling\_window : int, optional  
 数据平滑跨度. The default is 24.  
  
 Returns  
 -------  
 df\_selection\_group\_rolling\_resetIdx : DataFrame  
 仅保留时间列、数值列和分组列，数据平滑后的数据.  
  
 '''   
 df=df\_.copy(deep=True)  
 df.sort\_values(by=[datetime\_column],inplace=True)  
 df.set\_index(datetime\_column,inplace=True)   
 df\_selection=df[[val\_column,group\_column]]  
 df\_selection\_group\_rolling=df\_selection.groupby(group\_column)[val\_column].rolling(rolling\_window).mean()  
 df\_selection\_group\_rolling\_resetIdx=df\_selection\_group\_rolling.reset\_index()  
  
 return df\_selection\_group\_rolling\_resetIdx  
   
df\_selection\_group\_rolling\_resetIdx=df\_group\_rolling\_window(aot\_temperature\_subsystemNsensor\_gdf,'ts',"value\_hrf\_average","node\_id",rolling\_window=24\*7)   
df\_selection\_group\_rolling\_resetIdx.head(3)

|  | **node\_id** | **ts** | **value\_hrf\_average** |
| --- | --- | --- | --- |
| **0** | 001e0610ba46 | 2020-01-01 00:30:00 | NaN |
| **1** | 001e0610ba46 | 2020-01-01 01:30:00 | NaN |
| **2** | 001e0610ba46 | 2020-01-01 02:30:00 | NaN |

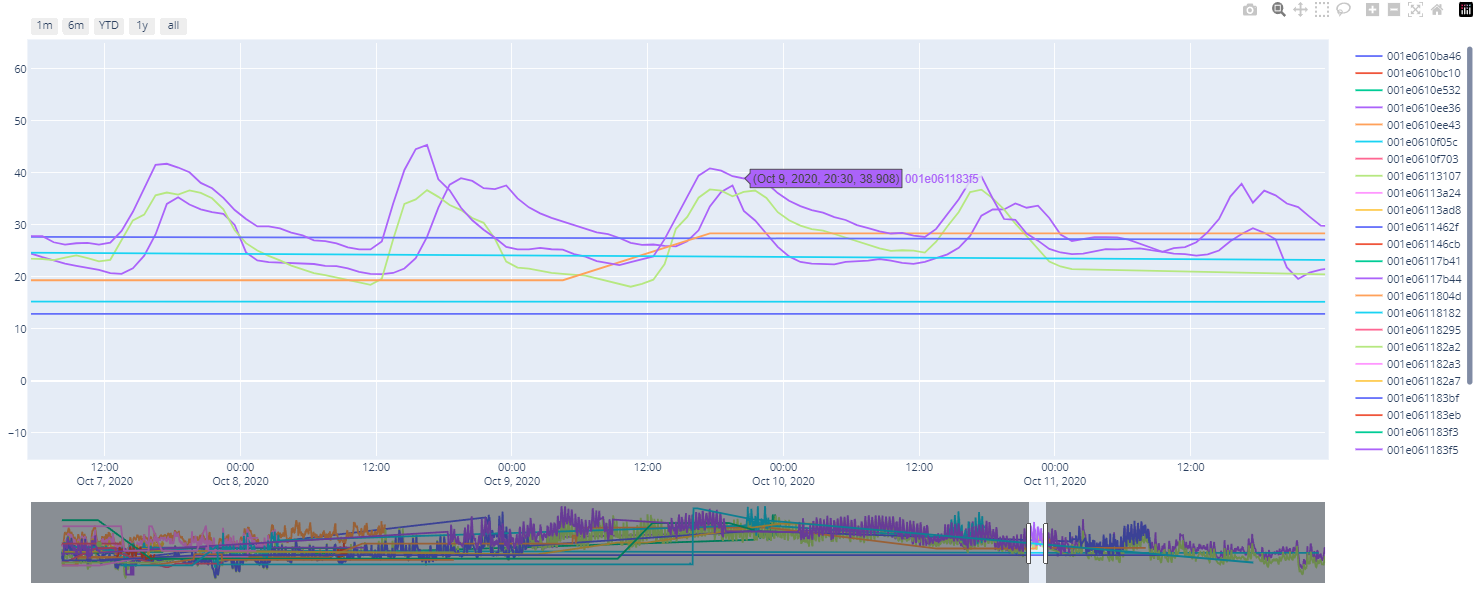
import plotly.express as px  
fig=px.line(df\_selection\_group\_rolling\_resetIdx, x="ts", y="value\_hrf\_average", color='node\_id')  
fig.update\_layout(  
 autosize=True,  
 # width=1000,  
 height=500,  
 margin=dict(  
 l=50,  
 r=50,  
 b=50,  
 t=50,  
 pad=4  
 ),  
 paper\_bgcolor="white",  
)  
fig.show()



上述交互图表的方法需要平滑数据，避免不同节点测量值间的干扰，但是平滑后的数据无法查看原始时间精度值的变化，通常只能查看年度、月度或周，较长时间段的变化趋势，因此定义下述time\_series\_plotly\_range\_slider\_plot()函数，使用Plotly库提供的窗口滑块缩放X轴向（时间轴向）功能，可以自由调节显示的时间区段，能够查看到最小时间精度（1小时）数值变化情况；也可以显示整个时间跨度，查看整体曲线变化趋势。

def time\_series\_plotly\_range\_slider\_plot(df\_,datetime\_column,val\_column,group\_column,\*\*setting):  
 '''  
 含时间列，数值列和分组列，按分组打印时间序列数值数据。含显示窗口滑块  
  
 Parameters  
 ----------  
 df\_ : DataFrame  
 含时间列，数值列和分组列的数据.  
 datetime\_column : string  
 时间列.  
 val\_column : string  
 数值列.  
 group\_column : string  
 分组列.  
 \*\*setting : kw-argument(string)  
 包括参数及默认值：  
 setting\_dict={"margin\_l":50,  
 "margin\_r":50,  
 "margin\_b":50,  
 "margin\_t":50,  
 "margin\_pad":4,  
 "paper\_bgcolor":"white",  
 "fig\_height":None,  
 "fig\_width":None,  
 "fig\_autosize":True,  
 "title\_text":""  
  
 Returns  
 -------  
 None.  
  
 '''   
 import plotly.graph\_objects as go  
 import pandas as pd  
 import numpy as np  
   
 setting\_dict={"margin\_l":50,  
 "margin\_r":50,  
 "margin\_b":50,  
 "margin\_t":50,  
 "margin\_pad":4,  
 "paper\_bgcolor":"white",  
 "fig\_height":None,  
 "fig\_width":None,  
 "fig\_autosize":True,  
 "title\_text":""  
 }  
 setting\_dict.update(setting)   
   
 fig = go.Figure()  
 df\_group=df\_.groupby(group\_column)  
 for label,df in df\_group:   
 fig.add\_trace(go.Scatter(x=df[datetime\_column],y=df[val\_column],name=label))  
   
 fig.update\_layout(  
 title\_text=setting\_dict["title\_text"],  
 autosize=setting\_dict["fig\_autosize"],  
 width=setting\_dict["fig\_width"],  
 height=setting\_dict["fig\_height"],  
 margin=dict(  
 l=setting\_dict["margin\_l"],  
 r=setting\_dict["margin\_r"],  
 b=setting\_dict["margin\_b"],  
 t=setting\_dict["margin\_t"],  
 pad=setting\_dict["margin\_pad"]),  
 paper\_bgcolor=setting\_dict["paper\_bgcolor"],  
 )  
   
 # Add range slider  
 fig.update\_layout(  
 xaxis=dict(  
 rangeselector=dict(  
 buttons=list([  
 dict(count=1,  
 label="1m",  
 step="month",  
 stepmode="backward"),  
 dict(count=6,  
 label="6m",  
 step="month",  
 stepmode="backward"),  
 dict(count=1,  
 label="YTD",  
 step="year",  
 stepmode="todate"),  
 dict(count=1,  
 label="1y",  
 step="year",  
 stepmode="backward"),  
 dict(step="all")  
 ])  
 ),  
 rangeslider=dict(  
 visible=True  
 ),  
 type="date"  
 )  
 )   
  
 fig.show()

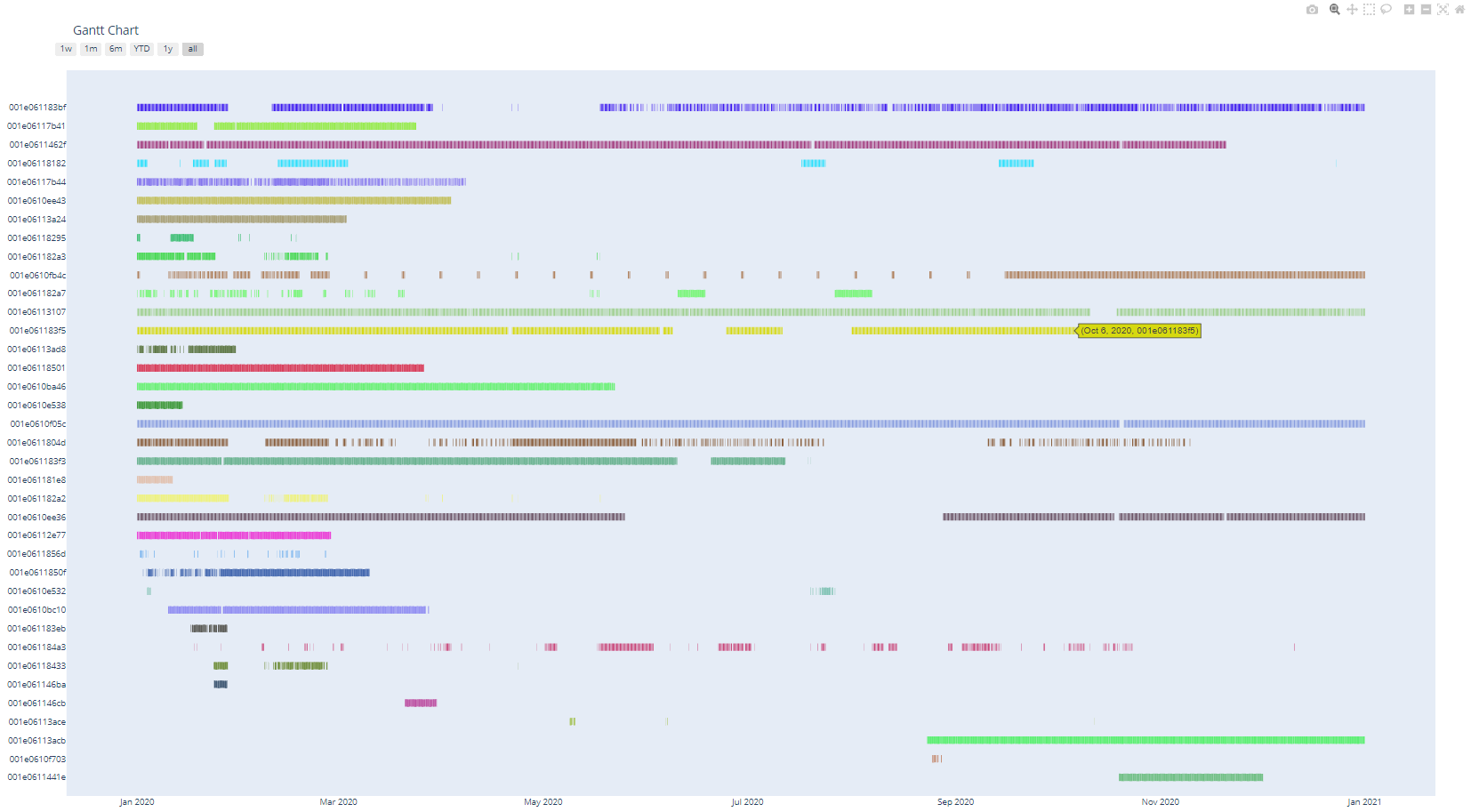
import plotly.io as pio  
pio.renderers.default="jupyterlab" #包括"browser"，"jupyterlab"，"notebook"...等  
time\_series\_plotly\_range\_slider\_plot(aot\_temperature\_subsystemNsensor\_gdf,"ts", "value\_hrf\_average", 'node\_id',fig\_height=700)



将在“城市环境传感器AoT（SAGE）数据预处理”一节中定义的gantt\_chart\_H()函数置于模块util\_misc中调用，打印温度传感器为“pr103j2”所获取的测量值在各个节点上的测量情况，包含“pr103j2”传感器的节点有37个，但是各节点获取的测量数据时间轴向上分布各异，可以发现节点’001e0610f05c’、’001e06113107’具有较连续的测量值，其次为’001e0611462f’和’001e061183bf’等。

from util\_misc import gantt\_chart\_H  
  
aot\_temperature\_subsystemNsensor\_gdf=postSQL2gpd(table\_name="temperature\_metsense\_pr103j2",geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)  
\_=gantt\_chart\_H(aot\_temperature\_subsystemNsensor\_gdf,"node\_id","ts")

C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning: Did not recognize type 'geometry' of column 'geometry'  
 pandas\_sql.meta.reflect(bind=pandas\_sql.connectable, only=[sql])  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is temperature\_metsense\_pr103j2.  
category\_field number:37  
category\_field:  
['001e061183bf' '001e06117b41' '001e0611462f' '001e06118182'  
 '001e06117b44' '001e0610ee43' '001e06113a24' '001e06118295'  
 '001e061182a3' '001e0610fb4c' '001e061182a7' '001e06113107'  
 '001e061183f5' '001e06113ad8' '001e06118501' '001e0610ba46'  
 '001e0610e538' '001e0610f05c' '001e0611804d' '001e061183f3'  
 '001e061181e8' '001e061182a2' '001e0610ee36' '001e06112e77'  
 '001e0611856d' '001e0611850f' '001e0610e532' '001e0610bc10'  
 '001e061183eb' '001e061184a3' '001e06118433' '001e061146ba'  
 '001e061146cb' '001e06113ace' '001e06113acb' '001e0610f703'  
 '001e0611441e']  
  
  
100%|██████████| 37/37 [00:00<00:00, 71.44it/s]



根据时间序列数据提供的node\_id字段，读取AoT数据预处理存入数据库的节点空间分布信息表AoT\_nodes，在地图上显示包含有“pr103j2”传感器测量值的节点（标识有节点ID文字的点）。也配置有变量nodes4search，可以用于标识需要特定查看的节点，以黄底红字加粗显示。下图特定节点为具有良好连续性测量值的节点。

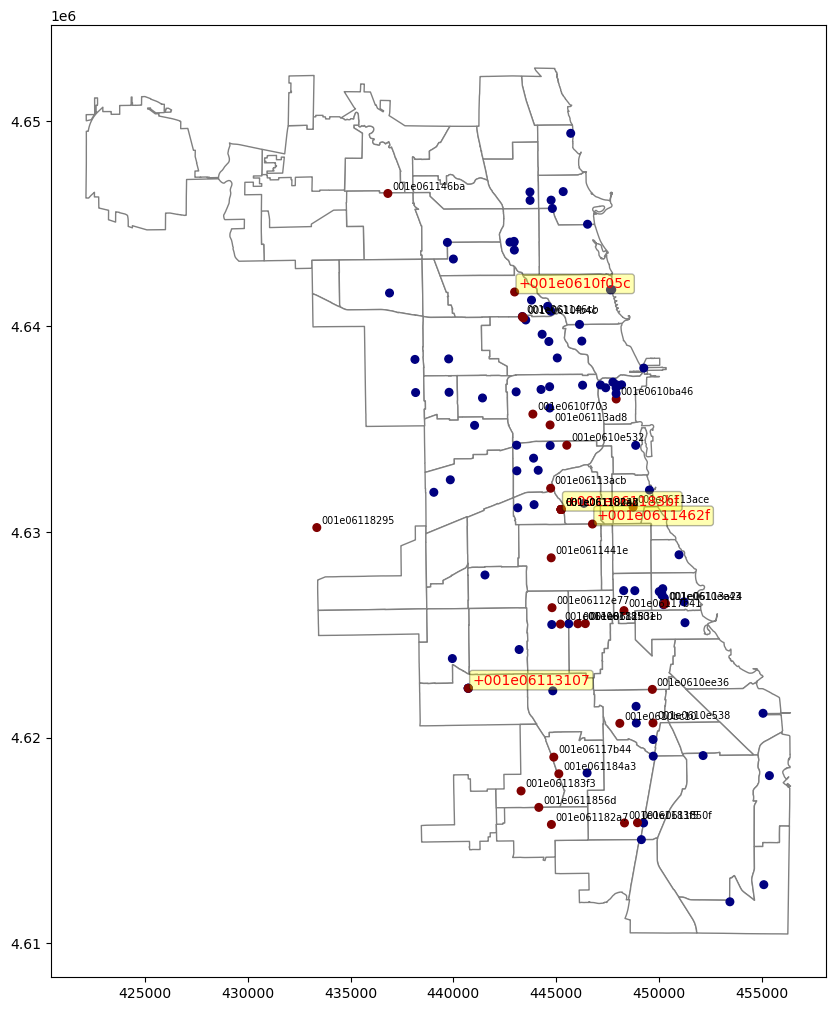
AoT\_nodes\_gdf=postSQL2gpd(table\_name="AoT\_nodes",\*\*args.db.db\_info)  
subsystemNsensor\_nodeIDs=aot\_temperature\_subsystemNsensor\_gdf.node\_id.unique()  
AoT\_nodes\_gdf["mask\_subsystemNsensor"]=AoT\_nodes\_gdf.node\_id.apply(lambda row:1 if row in subsystemNsensor\_nodeIDs else 0)  
AoT\_nodes\_gdf.head(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is AoT\_nodes.  
  
  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning:  
  
Did not recognize type 'geometry' of column 'geometry'

|  | **node\_id** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** | **mask\_subsystemNsensor** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 001e0610ba46 | AoT\_Chicago | 004 | State St & Jackson Blvd Chicago IL | 41.878377 | -87.627678 | AoT Chicago (S) [C] | 2017/10/09 00:00:00 | None | POINT (447918.217 4636463.149) | 1 |
| **1** | 001e0610ba3b | AoT\_Chicago | 006 | 18th St & Lake Shore Dr Chicago IL | 41.858136 | -87.616055 | AoT Chicago (S) | 2017/08/08 00:00:00 | None | POINT (448866.515 4634208.863) | 0 |
| **2** | 001e0610f02f | AoT\_Chicago | 00A | Lake Shore Drive & Fullerton Ave Chicago IL | 41.926261 | -87.630758 | AoT Chicago (S) [CA] | 2018/05/07 00:00:00 | None | POINT (447701.759 4641781.517) | 0 |

import matplotlib.pyplot as plt  
fig, ax=plt.subplots(figsize=(10,15))  
Chicago\_community\_areas=postSQL2gpd(table\_name='Chicago\_community\_areas',\*\*args.db.db\_info)   
Chicago\_community\_areas.plot(ax=ax,color='white', edgecolor='gray')  
AoT\_nodes\_gdf.plot(column='mask\_subsystemNsensor',ax=ax,markersize=30,cmap="jet")  
  
nodes4search=['001e06113107','001e0610f05c','001e0611462f','001e061183bf']  
for x, y, label in zip(AoT\_nodes\_gdf.geometry.x, AoT\_nodes\_gdf.geometry.y, AoT\_nodes\_gdf.node\_id):  
 if label in subsystemNsensor\_nodeIDs:  
 if label in nodes4search:  
 ax.annotate(f"+{label}", xy=(x, y), xytext=(3, 3), textcoords="offset points",fontsize=10,color="r",bbox=dict(boxstyle='round,pad=0.2', fc='yellow', alpha=0.3))  
 else:  
 ax.annotate(label, xy=(x, y), xytext=(3, 3), textcoords="offset points",fontsize=7)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is Chicago\_community\_areas.



### 2.7.2.2 白噪声检验、平稳性检验和周期性模式判断

#### 1）白噪声检验

如果序列值彼此之间没有任何相关性，过去的行为并不影响将来的行为趋势，则该序列为随机序列，即为白噪声序列。白噪声序列因为数值为随机，通常不具有分析的价值，因此对时间序列数据往往要进行白噪声检验来确定待分析序列是否适合用于进一步的分析，例如时间序列建模，空间相关性等。[statsmodels](https://www.statsmodels.org/stable/index.html)①库提供的acorr\_ljungbox方法包含BP检验（Box-Pierce test）和LB检验（Ljung-Box test）来判断一个序列是否为白噪声，两种假设检验的思路基本一致，如果序列前若干项相关系数都为零，那么检验统计量渐进服从于卡方分布，要是统计量的值在右端拒绝域内则可以拒绝原假设，判定该序列并非白噪声。

获取的温度传感器时间序列数据，通过图表打印查看各节点数值变化，能够发现测量值具有一定规律性，因此进一步通过[statsmodels.stats.diagnostic.acorr\_ljungbox](https://www.statsmodels.org/dev/generated/statsmodels.stats.diagnostic.acorr_ljungbox.html)方法计算确定。配置corr\_ljungbox输入参数boxpierce=True，除计算Ljung-Box test外，也计算Box-Pierce test，判断返回的lb\_pvalue和bp\_pvalue是否大于给定的显著性水平（默认配置为0.05），如果均大于无法拒绝原假设，则认为该数据为白噪声，即纯粹的随机数据。

为了方便对各个节点分别执行白噪声检验，对预处理的温度时间序列数据根据指定的分组列（node\_id），数值列（value\_hrf\_average），和用于索引的时间列（ts）执行pandas库提供的pivot方法实现转置，将节点作为列名，时间作为索引值，如果对应时间行无数据则为NaN。

aot\_temperature\_subsystemNsensor\_pivot=aot\_temperature\_subsystemNsensor\_gdf.pivot(values="value\_hrf\_average",columns="node\_id",index="ts")  
  
from util\_misc import print\_html  
print\_html(aot\_temperature\_subsystemNsensor\_pivot)

| **node\_id** | **001e0610ba46** | **001e0610bc10** | **001e0610e532** | **001e0610e538** | **001e0610ee36** | **001e0610ee43** | **001e0610f05c** | **001e0610f703** | **001e0610fb4c** | **001e06112e77** | **001e06113107** | **001e06113a24** | **001e06113acb** | **001e06113ace** | **001e06113ad8** | **001e0611441e** | **001e0611462f** | **001e061146ba** | **001e061146cb** | **001e06117b41** | **001e06117b44** | **001e0611804d** | **001e06118182** | **001e061181e8** | **001e06118295** | **001e061182a2** | **001e061182a3** | **001e061182a7** | **001e061183bf** | **001e061183eb** | **001e061183f3** | **001e061183f5** | **001e06118433** | **001e061184a3** | **001e06118501** | **001e0611850f** | **001e0611856d** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ts** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2020-01-01 00:30:00** | 0.593 | NaN | NaN | -3.228 | -0.367 | -0.923 | -1.535 | NaN | -0.891 | -1.932 | -1.807 | -0.806 | NaN | NaN | -0.59 | NaN | -4.351 | NaN | NaN | -1.312 | 2.666 | -1.326 | -0.273 | -1.377 | 0.380 | -1.325 | -0.784 | 5.335 | -1.003 | NaN | -1.642 | -1.601 | NaN | NaN | -0.245 | NaN | NaN |
| **2020-01-01 01:30:00** | 0.455 | NaN | NaN | -2.850 | 0.057 | -0.858 | -1.278 | NaN | -0.902 | -1.642 | -1.470 | -0.526 | NaN | NaN | NaN | NaN | -4.169 | NaN | NaN | -0.926 | 2.300 | -1.138 | -0.014 | -1.094 | 0.386 | -1.138 | -0.515 | 5.617 | -0.814 | NaN | -1.389 | -1.249 | NaN | NaN | 0.117 | NaN | NaN |
| **2020-01-01 02:30:00** | 0.325 | NaN | NaN | -2.724 | 0.091 | -0.685 | -1.280 | NaN | -0.855 | -1.608 | -1.510 | -0.385 | NaN | NaN | -0.80 | NaN | -4.095 | NaN | NaN | -0.901 | 0.034 | -1.060 | 0.232 | -1.121 | 0.619 | -1.058 | -0.456 | 5.651 | -0.758 | NaN | -1.208 | -1.085 | NaN | NaN | 0.297 | NaN | NaN |
| **2020-01-01 03:30:00** | 0.538 | NaN | NaN | -3.001 | 0.191 | -0.658 | -1.294 | NaN | -0.521 | -1.589 | -1.660 | -0.516 | NaN | NaN | NaN | NaN | -4.107 | NaN | NaN | -1.050 | -0.050 | -1.011 | 0.293 | -1.155 | 1.397 | -1.009 | -0.492 | 5.750 | -0.685 | NaN | -1.366 | -1.182 | NaN | NaN | 0.199 | NaN | NaN |
| **2020-01-01 04:30:00** | 0.246 | NaN | NaN | -2.932 | 0.886 | -0.409 | -1.451 | NaN | -0.801 | -1.552 | -1.620 | -0.094 | NaN | NaN | NaN | NaN | -3.961 | NaN | NaN | -0.871 | -0.096 | -0.910 | 0.484 | -0.990 | 1.386 | -0.859 | -0.449 | NaN | -0.613 | NaN | -1.313 | -1.197 | NaN | NaN | 0.294 | NaN | NaN |

白噪声检验定义的函数white\_noise\_Ljung\_box\_pierc()调用定义的函数white\_noise\_testing()，返回值包括BP检验和LB检验结果，并根据显著性水平参数alpha返回是否为白噪声的判断。

def white\_noise\_testing(df,value\_column,lags=[6,12]):  
 '''  
 白噪声检验(纯随机性检验)，statsmodels.stats.diagnostic.acorr\_ljungbox方法计算，包括Ljung-Box test和Box-Pierce test  
  
 Parameters  
 ----------  
 df : DataFrame  
 包含检测列的DataFrame格式数据.  
 value\_column : string  
 待检测列名.  
 lags : int, optional  
 延迟数. The default is [6,12].  
  
 Returns  
 -------  
 stat\_pvalue : dict  
 返回Ljung-Box test和Box-Pierce统计值，及p\_value值.  
  
 '''  
 import statsmodels.stats.diagnostic as diag  
   
 df\_dropped=df[value\_column].dropna()  
 stat\_pvalue=diag.acorr\_ljungbox(df\_dropped, lags=lags, boxpierce=True, model\_df=0, period=None, return\_df=None)  
 return stat\_pvalue   
  
def white\_noise\_Ljung\_box\_pierce(df,lags=[6,12],alpha=0.05):  
 '''  
 白噪声批量检验(纯随机性检验)，statsmodels.stats.diagnostic.acorr\_ljungbox方法计算，包括Ljung-Box test和Box-Pierce test。并根据计算p-value值是否大于显著性水平（0.05或0.01），判断是否为白噪声  
  
 Parameters  
 ----------  
 df: DataFrame  
 待检验的数据，检验内容为所有列.  
 lags : int, optional  
 延迟数. The default is [6,12].  
 alpha : float, optional  
 显著性水平. The default is 0.05.  
  
 Returns  
 -------  
 stat\_pvalue\_dict : dict  
 包括Ljung-Box test和Box-Pierce test的统计值和p-value.  
 is\_white\_nose : dict  
 各个节点（node）是否为白噪音.  
  
 '''  
 from tqdm import tqdm  
  
 columns=df.columns  
 stat\_pvalue\_dict={}  
 is\_white\_nose={}  
 for c in tqdm(columns):   
 try:  
 stat\_pvalue=white\_noise\_testing(df,c,lags=lags)  
 lb\_pvalue=stat\_pvalue[1]  
 bp\_pvalue=stat\_pvalue[3]   
 stat\_pvalue\_dict[c]={"stat":{"lb\_stat":stat\_pvalue[0],"bp\_stat":stat\_pvalue[2]},"p-value":{"lb\_pvalue":lb\_pvalue,"bp\_pvalue":bp\_pvalue}}  
 is\_white\_nose[c]=dict(zip(["lag\_{}".format(i) for i in lags],[True if i>alpha and j>alpha else False for i,j in zip(lb\_pvalue,bp\_pvalue)]))  
 except:  
 print("{}-error!".format(c))   
 return stat\_pvalue\_dict,is\_white\_nose

import pandas as pd  
temperature\_stat\_pvalue\_dict,temperature\_is\_white\_nose=white\_noise\_Ljung\_box\_pierce(aot\_temperature\_subsystemNsensor\_pivot,lags=[12,24,24\*7])  
  
node\_sampe='001e0610ba46'  
print(temperature\_stat\_pvalue\_dict[node\_sampe],'\n',temperature\_is\_white\_nose[node\_sampe])  
temperature\_is\_white\_noise\_df=pd.DataFrame.from\_dict(temperature\_is\_white\_nose,orient='index')  
temperature\_is\_white\_noise\_df

100%|██████████| 37/37 [00:00<00:00, 287.61it/s]  
  
001e0610e532-error!  
001e0610f703-error!  
001e06113ace-error!  
001e061146ba-error!  
001e0611856d-error!  
{'stat': {'lb\_stat': array([ 30413.76341513, 48294.64370466, 121744.96311353]), 'bp\_stat': array([ 30343.54687994, 48118.79112619, 119570.72878674])}, 'p-value': {'lb\_pvalue': array([0., 0., 0.]), 'bp\_pvalue': array([0., 0., 0.])}}   
 {'lag\_12': False, 'lag\_24': False, 'lag\_168': False}

|  | **lag\_12** | **lag\_24** | **lag\_168** |
| --- | --- | --- | --- |
| **001e0610ba46** | False | False | False |
| **001e0610bc10** | False | False | False |
| **001e0610e538** | False | False | False |
| **001e0610ee36** | False | False | False |
| **001e0610ee43** | False | False | False |
| **001e0610f05c** | False | False | False |
| **001e0610fb4c** | False | False | False |
| **001e06112e77** | False | False | False |
| **001e06113107** | False | False | False |
| **001e06113a24** | False | False | False |
| **001e06113acb** | False | False | False |
| **001e06113ad8** | False | False | False |
| **001e0611441e** | False | False | False |
| **001e0611462f** | False | False | False |
| **001e061146cb** | False | False | False |
| **001e06117b41** | False | False | False |
| **001e06117b44** | False | False | False |
| **001e0611804d** | False | False | False |
| **001e06118182** | False | False | False |
| **001e061181e8** | False | False | False |
| **001e06118295** | False | False | False |
| **001e061182a2** | False | False | False |
| **001e061182a3** | False | False | False |
| **001e061182a7** | False | False | False |
| **001e061183bf** | False | False | False |
| **001e061183eb** | False | False | False |
| **001e061183f3** | False | False | False |
| **001e061183f5** | False | False | False |
| **001e06118433** | False | False | False |
| **001e061184a3** | False | False | False |
| **001e06118501** | False | False | False |
| **001e0611850f** | False | False | False |

从上述分析结果来看，参与检验的37个节点，无效节点5个，有效节点32个。这32个节点的白噪声检验，在延迟数（lags）配置上包括3个量，为12（小时）,24（小时）和168（小时），对应半天、一天和7周时间，从返回结果可以确定，所有节点温度测量值均不是白噪声，可用于进一步的分析，获得有价值的研究结论。

#### 2）平稳性检验

如果要对时间序列建立预测模型，例如ARMA（autoregressive moving average）或ARIMA（autoregressive integrated moving average），需要时间序列数据具有平稳性（意味均值和方差保持不变）。通过检验温度传感器时间序列数据的平稳性，也可以确定温度测量值的变化趋势，是否数值稳定在一个固定区间内，还是会有剧烈的随机波动。计算使用[statsmodels.tsa.stattools.adfuller](https://www.statsmodels.org/devel/generated/statsmodels.tsa.stattools.adfuller.html)方法，如adfuller（ADF）返回的p-value值小于给定的显著性水平（默认配置为0.05），则数据平稳；否则无法拒绝原假设，数据不平稳。

平稳性检验定义的函数stationary\_stat()调用定义的函数stationary\_adfulle，返回值是根据配置的alpha显著性水平参数确定各列（节点）时间序列值是否平稳。

def stationary\_adfuller(df,value\_column,autolag=None,alpha=0.05,print\_detail=False):   
 '''  
 计算时间序列的平稳性（statsmodels的adfuller方法）  
  
 Parameters  
 ----------  
 df : DataFrame  
 包含计算列的DataFrame格式数据.  
 value\_column : string  
 待计算的列名.  
 autolag : string, optional  
 {“AIC”, “BIC”, “t-stat”, None}，具体参考 https://www.statsmodels.org/devel/generated/statsmodels.tsa.stattools.adfuller.html. The default is None.  
 alpha : float, optional  
 显著性水平(p-value). The default is 0.05.  
 print\_detail : bool, optional  
 是否打印统计细节信息. The default is False.  
  
 Returns  
 -------  
 is\_stationary : dict  
 节点名：是否平稳布尔值.  
  
 '''  
 from statsmodels.tsa.stattools import adfuller   
   
 df\_drop=df[value\_column].dropna()  
 adfuller\_test=adfuller(df\_drop,autolag=autolag) # https://www.statsmodels.org/devel/generated/statsmodels.tsa.stattools.adfuller.html  
 is\_stationary=True if adfuller\_test[1]<=alpha else False  
 if print\_detail:  
 print('ADF statistic: %f' % adfuller\_test[0])  
 print('p-value: %f' % adfuller\_test[1])  
 print('critical values:')  
 for key, value in adfuller\_test[4].items():  
 print('\t%s: %.3f' % (key, value))   
 return is\_stationary  
  
def stationary\_stat(df\_,autolag=None,alpha=0.05,print\_detail=False,diff\_periods=None):  
 '''  
 批量计算所有节点的平稳性  
  
 Parameters  
 ----------  
 df\_: DataFrame  
 待检验的数据，检验内容为所有列.   
 autolag : string, optional  
 {“AIC”, “BIC”, “t-stat”, None}. The default is None.  
 alpha : float, optional  
 显著性水平(p-value). The default is 0.05.  
 print\_detail : bool, optional  
 是否打印统计细节信息. The default is False.  
  
 Returns  
 -------  
 is\_stationary\_dict : dict(node:bool)  
 返回各个节点是否平稳的布尔值，为True则平稳.  
  
 '''  
 from tqdm import tqdm  
  
 df=df\_.copy(deep=True)  
 is\_stationary\_dict={}  
 columns=df.columns  
 valid\_columns=[]  
  
 if diff\_periods:  
 df=df.diff(diff\_periods)  
  
 for c in tqdm(columns):  
 try:  
 is\_stationary=stationary\_adfuller(df,c,autolag=autolag,alpha=alpha,print\_detail=print\_detail)  
 is\_stationary\_dict[c]=is\_stationary  
 valid\_columns.append(c)  
 except:  
 print("sample size is too short to calculate.")  
  
 print("is\_stationary number={} in {} of valid nodes.".format(list(is\_stationary\_dict.values()).count(True),len(valid\_columns)))  
 return is\_stationary\_dict

temperature\_is\_stationary\_dict=stationary\_stat(aot\_temperature\_subsystemNsensor\_pivot)  
temperature\_is\_stationary\_df=pd.DataFrame.from\_dict(temperature\_is\_stationary\_dict,orient='index',columns=["temperature\_is\_stationary"])  
temperature\_is\_stationary\_df[temperature\_is\_stationary\_df["temperature\_is\_stationary"]==False]

100%|██████████| 37/37 [00:00<00:00, 197.33it/s]  
  
is\_stationary number=23 in 37 of valid nodes.

|  | **temperature\_is\_stationary** |
| --- | --- |
| **001e0610e532** | False |
| **001e0610e538** | False |
| **001e0610f703** | False |
| **001e06113ace** | False |
| **001e061146ba** | False |
| **001e061146cb** | False |
| **001e06118182** | False |
| **001e061181e8** | False |
| **001e061182a7** | False |
| **001e061183bf** | False |
| **001e061183eb** | False |
| **001e06118433** | False |
| **001e061184a3** | False |
| **001e0611856d** | False |

从上述结果得知，对于温度测量值各节点具有平稳性的有23个，而不平稳的有14个，这与时间序列图表打印结果符合，从图表中也可以发现有些节点测量值波动较大且不稳定。下述实验则指定了参数diff\_periods为1，对时间序列执行一阶差分，计算一阶差分后的平稳性得知，不平稳的节点3个，平稳的节点34个。

对于获取的各类具有一定趋势的线性时间序列数据，有时数据并不平稳，但是又需要该数据进行后续分析使用只能在平稳性数据下才可使用的估计和检验方法，这时通常对不平稳的数据做一阶差分或多阶差分，即用后期数据减去前一期数据的结果，通过平稳性检验，原不具有平稳性的数据如果通过差分具有了平稳性，则可认为该时间数据“已经为具有平稳性”，消除了原数据的一些波动，使其趋于平稳。（一阶）差分的意义是为了消除时间序列数据中包含的随机性趋势，以令世界序列数据趋于平稳[1]431。从一阶差分后平稳性检验结果来看，由原来不平稳节点数14个减少至3个，有11个节点由不平稳转换为平稳数据。

temperature\_is\_stationary\_dict=stationary\_stat(aot\_temperature\_subsystemNsensor\_pivot,diff\_periods=1)  
temperature\_is\_stationary\_df=pd.DataFrame.from\_dict(temperature\_is\_stationary\_dict,orient='index',columns=["temperature\_is\_stationary"])  
temperature\_is\_stationary\_df[temperature\_is\_stationary\_df["temperature\_is\_stationary"]==False]

100%|██████████| 37/37 [00:00<00:00, 202.76it/s]  
  
is\_stationary number=34 in 37 of valid nodes.

|  | **temperature\_is\_stationary** |
| --- | --- |
| **001e0610f703** | False |
| **001e06113ace** | False |
| **001e0611856d** | False |

#### 3）周期性模式判断

使用自相关系数和偏自相关系数来检验时间序列数据与给定延迟数/滞后数（lag）错位自身的时间序列数据之间是否具有相关性，使用statsmodel库提供的acf和pacf计算，寻找时间序列数据重复模式，例如是否具有周期性。通过statsmodel库提供的plot\_acf和plot\_pacf图表打印时间序列数据，可以观察到时间序列数据的周期性。

温度测量值的平稳性检验中有部分数据数值变化较大，通过自相关系数计算和图表打印，部分节点不容易发现周期性的模式，因此同时对一阶差分后的时序测量值进行自相关系数计算，结果图表较之未差分前，可清晰判断具有周期性模式，周期约为24小时（天）。

* ACF和PACF

自相关系数（autocorrelation function，ACF），是将一列数据按照滞后数提取新一列数据，计算原数据和新数据之间的相关性，是数据与自身延迟的相关，是它们之间时间滞后的函数。ACF用于寻找一列数据的重复模式，例如被噪声掩盖的周期性信号等。

偏自相关系数（Partial autocorrelation function, PACF），在时间序列分析中，静止的时间序列与其自身滞后值的部分相关性（移除了中间变量的间接影响）。对于时间序列，与的偏自相关系数是指去掉{ xt }{ xt-k}的简单相关系数。

def autocorrelation\_time\_series(df, value\_column,nlags=None,plot=False,bins\_list=[-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1],title="ACF",xticks\_step=6,save\_path=None,plot\_show=False):   
 '''  
 计算DataFrame给定列的自相关系数并打印图表，及统计区间频数  
  
 Parameters  
 ----------  
 df : DataFrame  
 含计算列的DataFrame格式数据.  
 value\_column : string  
 待计算列名.  
 nlags : int, optional  
 滞后（延迟）数. The default is None.  
 plot : bool, optional  
 是否打印图表. The default is False.  
 bins\_list : list(float), optional  
 分类区间. The default is [-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1].  
  
 Returns  
 -------  
 acf : list(float)  
 各个延迟数下的自相关系数.  
 categories\_counts : pandas.core.series.Series (int)  
 自相关系数区间频数统计.  
  
 '''  
 import matplotlib.pyplot as plt  
 from statsmodels import api as sm  
 import pandas as pd  
 import os  
   
 value\_series=df[value\_column]  
 value\_series.dropna(inplace=True)  
 # get the autocorrelation coefficient  
 if nlags is not None:  
 acf=sm.tsa.acf(value\_series, nlags=nlags)  
 else:  
 acf=sm.tsa.acf(value\_series, nlags=len(value\_series))   
   
 bins\_tuples=[(bins\_list[i],bins\_list[i+1]) for i in range(len(bins\_list)-1)]   
 bins=pd.IntervalIndex.from\_tuples(bins\_tuples)  
 categories\_counts=pd.cut(acf, bins).value\_counts()   
   
 if plot:   
 fig=plt.figure(figsize=(12,8))  
 ax=fig.add\_subplot(111)  
 plt.rc('axes', unicode\_minus=False)  
 if nlags is not None:  
 fig=sm.graphics.tsa.plot\_acf(value\_series, lags=nlags,ax=ax,title=title)  
 else:  
 fig=sm.graphics.tsa.plot\_acf(value\_series, lags=len(value\_series)-1,ax=ax,title=title)  
 ax.xaxis.set\_ticks\_position('bottom')   
 fig.tight\_layout()  
   
 old\_xticks=ax.get\_xticks()  
 xticks\_step=xticks\_step  
 new\_xticks=list(range(0,int(old\_xticks[-1]),xticks\_step))  
 ax.set\_xticks(new\_xticks)  
 if save\_path:  
 plt.savefig(os.path.join(save\_path,"{}.jpg".format(title)))  
 if plot\_show:  
 plt.show()  
 plt.close('all')  
 return acf,categories\_counts  
  
def partial\_autocorrelation\_time\_series(df, value\_column,nlags=None,plot=False,bins\_list=[-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1],title="PACF",xticks\_step=6,save\_path=None,plot\_show=False):   
 '''  
 计算DataFrame给定列的偏自相关系数并打印图表，及统计区间频数  
  
 Parameters  
 ----------  
 df : DataFrame  
 含计算列的DataFrame格式数据.  
 value\_column : string  
 待计算列名.  
 nlags : int, optional  
 滞后（延迟）数. The default is None.  
 plot : bool, optional  
 是否打印图表. The default is False.  
 bins\_list : list(float), optional  
 分类区间. The default is [-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1].  
  
 Returns  
 -------  
 pacf : list(float)  
 各个延迟数下的偏自相关系数.  
 categories\_counts : pandas.core.series.Series (int)  
 偏自相关系数区间频数统计.  
  
 '''  
 import matplotlib.pyplot as plt  
 from statsmodels import api as sm  
 import pandas as pd  
 import os  
   
 value\_series=df[value\_column]  
 value\_series.dropna(inplace=True)  
 # get the autocorrelation coefficient  
 if nlags is not None:  
 pacf=sm.tsa.pacf(value\_series, nlags=nlag,method='ywm')  
 else:  
 pacf=sm.tsa.pacf(value\_series, nlags=len(value\_series),method='ywm')   
   
 bins\_tuples=[(bins\_list[i],bins\_list[i+1]) for i in range(len(bins\_list)-1)]   
 bins=pd.IntervalIndex.from\_tuples(bins\_tuples)  
 categories\_counts=pd.cut(pacf, bins).value\_counts()   
   
 if plot:  
 fig=plt.figure(figsize=(12,8))  
 ax=fig.add\_subplot(111)  
 plt.rc('axes', unicode\_minus=False)  
 if nlags is not None:  
 fig=sm.graphics.tsa.plot\_pacf(value\_series, lags=nlags,ax=ax,title=title,method='ywm')  
 else:  
 fig=sm.graphics.tsa.plot\_pacf(value\_series, lags=len(value\_series)-1,ax=ax,title=title,method='ywm')  
 ax.xaxis.set\_ticks\_position('bottom')   
 fig.tight\_layout()  
   
 old\_xticks=ax.get\_xticks()  
 xticks\_step=xticks\_step  
 new\_xticks=list(range(0,int(old\_xticks[-1]),xticks\_step))  
 ax.set\_xticks(new\_xticks)   
 if save\_path:  
 plt.savefig(os.path.join(save\_path,"{}.jpg".format(title)))   
 if plot\_show:  
 plt.show()  
 plt.close('all')  
 return pacf,categories\_counts  
  
def acf\_pacf\_counter(df\_,nlags=7,plot=False,save\_path\_acf=None,save\_path\_pacf=None,diff\_periods=None,plot\_show=False):  
 '''  
 所有节点（nodes）自相关autocorrelation和偏自相关partial autocorrelation计算及频数统计  
  
 Parameters  
 ----------  
 df\_: DataFrame  
 待检验的数据，检验内容为所有列.   
 nlags : int, optional  
 延迟数. The default is 7.  
 plot : bool, optional  
 是否打印图表,True为打印. The default is False.  
  
 Returns  
 -------  
 acf\_counter\_dict : dict  
 返回自相关系数字典（node:acf）.  
 pacf\_counter\_dict : dict  
 返回偏自相关字典（node:pacf）.  
  
 '''  
  
 df=df\_.copy(deep=True)  
 columns=df.columns  
 acf\_counter\_dict={}  
 pacf\_counter\_dict={}  
 acf\_insufficient\_num=0  
 pacf\_insufficient\_num=0   
 if diff\_periods:  
 df=df.diff(diff\_periods)  
 for column in columns:  
 try:  
 acf,acf\_categories\_counts=autocorrelation\_time\_series(df,column,plot=plot,nlags=nlags,title="acf\_{}".format(column),save\_path=save\_path\_acf,plot\_show=plot\_show)  
 acf\_counter\_dict[column]=acf\_categories\_counts   
 except:  
 acf\_insufficient\_num+=1  
 print("acf\_{}-Insufficient sample size!".format(column))   
  
 try:   
 pacf,pacf\_categories\_counts=partial\_autocorrelation\_time\_series(df,column,plot=plot,nlags=nlags,title="pacf\_{}".format(column),save\_path=save\_path\_pacf,plot\_show=plot\_show)   
 pacf\_counter\_dict[column]=pacf\_categories\_counts  
 except:  
 pacf\_insufficient\_num+=1  
 print("pacf\_{}-Insufficient sample size!".format(column))   
  
 acf\_counter\_sum=pd.DataFrame(acf\_counter\_dict.values()).sum(axis=0)   
 pacf\_counter\_sum=pd.DataFrame(pacf\_counter\_dict.values()).sum(axis=0)  
 print("\_"\*50)  
 print("acf\_valid column number={};pacf\_valid columnnumber={}".format(len(columns)-acf\_insufficient\_num,len(columns)-pacf\_insufficient\_num))  
 print("\_"\*50)  
 print("acf\_counter\_sum={},\n,pacf\_counter\_sum={};".format(acf\_counter\_sum,pacf\_counter\_sum))  
 return acf\_counter\_dict,pacf\_counter\_dict

save\_path\_acf='./graph/temperature\_acf'  
save\_path\_pacf='./graph/temperature\_pacf'  
temperature\_acf\_counter\_dict,temperature\_pacf\_counter\_dict=acf\_pacf\_counter(aot\_temperature\_subsystemNsensor\_pivot,nlags=24\*7,plot=True,save\_path\_acf=save\_path\_acf,save\_path\_pacf=save\_path\_pacf,diff\_periods=1)

pacf\_001e0610ba46-Insufficient sample size!  
pacf\_001e0610bc10-Insufficient sample size!  
acf\_001e0610e532-Insufficient sample size!  
pacf\_001e0610e532-Insufficient sample size!  
pacf\_001e0610e538-Insufficient sample size!  
pacf\_001e0610ee36-Insufficient sample size!  
pacf\_001e0610ee43-Insufficient sample size!  
pacf\_001e0610f05c-Insufficient sample size!  
acf\_001e0610f703-Insufficient sample size!  
pacf\_001e0610f703-Insufficient sample size!  
pacf\_001e0610fb4c-Insufficient sample size!  
pacf\_001e06112e77-Insufficient sample size!  
pacf\_001e06113107-Insufficient sample size!  
pacf\_001e06113a24-Insufficient sample size!  
pacf\_001e06113acb-Insufficient sample size!  
acf\_001e06113ace-Insufficient sample size!  
pacf\_001e06113ace-Insufficient sample size!  
pacf\_001e06113ad8-Insufficient sample size!  
pacf\_001e0611441e-Insufficient sample size!  
pacf\_001e0611462f-Insufficient sample size!  
acf\_001e061146ba-Insufficient sample size!  
pacf\_001e061146ba-Insufficient sample size!  
pacf\_001e061146cb-Insufficient sample size!  
pacf\_001e06117b41-Insufficient sample size!  
pacf\_001e06117b44-Insufficient sample size!  
pacf\_001e0611804d-Insufficient sample size!  
pacf\_001e06118182-Insufficient sample size!  
pacf\_001e061181e8-Insufficient sample size!  
pacf\_001e06118295-Insufficient sample size!  
pacf\_001e061182a2-Insufficient sample size!  
pacf\_001e061182a3-Insufficient sample size!  
pacf\_001e061182a7-Insufficient sample size!  
pacf\_001e061183bf-Insufficient sample size!  
pacf\_001e061183eb-Insufficient sample size!  
pacf\_001e061183f3-Insufficient sample size!  
pacf\_001e061183f5-Insufficient sample size!  
pacf\_001e06118433-Insufficient sample size!  
pacf\_001e061184a3-Insufficient sample size!  
pacf\_001e06118501-Insufficient sample size!  
pacf\_001e0611850f-Insufficient sample size!  
acf\_001e0611856d-Insufficient sample size!  
pacf\_001e0611856d-Insufficient sample size!  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
acf\_valid column number=32;pacf\_valid columnnumber=0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
acf\_counter\_sum=(-1.0, -0.75] 0  
(-0.75, -0.5] 0  
(-0.5, -0.25] 0  
(-0.25, 0.0] 3073  
(0.0, 0.25] 2168  
(0.25, 0.5] 128  
(0.5, 0.75] 7  
(0.75, 1.0] 32  
dtype: int64,  
,pacf\_counter\_sum=Series([], dtype: float64);

* 温度测量值自相关系数 (ACF) 一阶差分 lag=24×7



* 温度测量值偏自相关系数(PACF) 一阶差分 lag=24×7



注释（Notes）：

① statsmodels，是一个Python模块，为许多不同统计模型的估计提供了类和函数，进行统计测试和统计推断。每个估计器都有一个广泛的结果统计列表，与现有的统计包进行测试，确保它们的正确性（<https://www.statsmodels.org/stable/index.html>）。

参考文献（References）:

[1] 詹姆斯·H·斯托克; 马克·W·沃森. 沈根祥,孙燕等译. 计量经济学[M]格致出版社\_上海三联书店\_上海人民出版社: 上海, 2012.04 (第三版).