**异常检测报告**

**学号: 300000000 姓名: 张大炮 班级: 人工智能3班**

1. **目标**

针对实际获取的某大楼光伏电站的数据，判断数据中存在的异常。

1. **数据**

数据文件pv.csv记录了来源于某大楼的光伏电站的数据，其中数据时间段为2020/5/7 15:00:00---2023/10/25 16:15:00，采样频率为15min，每个时刻获取了两个物理量：功率和发电量，共121542个数据时刻。

1. **任务**
2. 分别画出两个物理量的趋势折线图以及散点图、盒图，并分析数据中可能存在的异常；

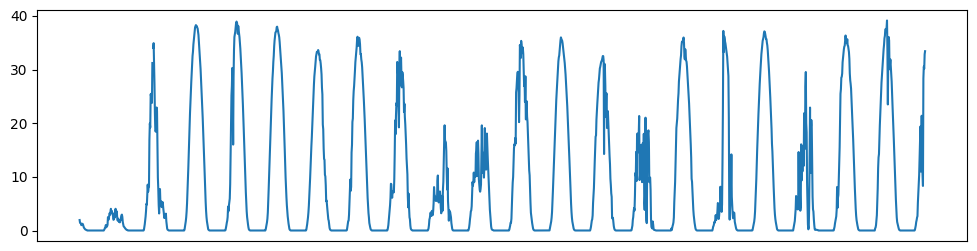
首先采用数据集中前2000个数据点进行观察，画个折线图看看什么情况，应该是周期性用电，时高时低。

# plot

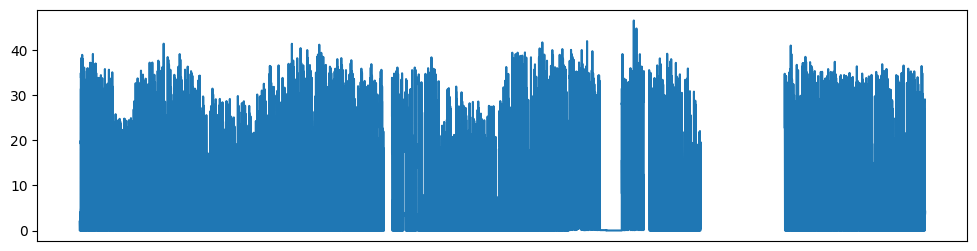
plt.figure(figsize=(12, 3))

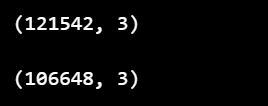
plt.plot(pv\_data['time'], pv\_data['power'])

plt.xticks([])

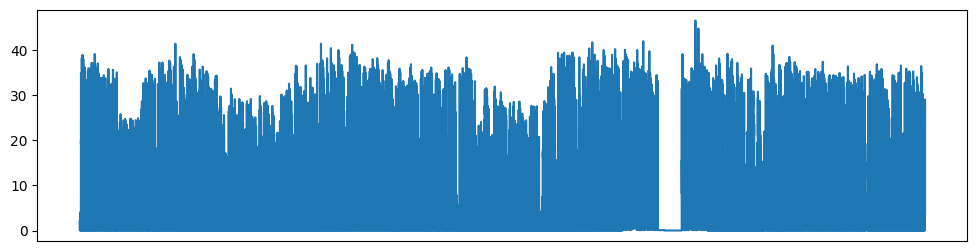


再看一眼全部数据，原来有nan，直接全部drop掉

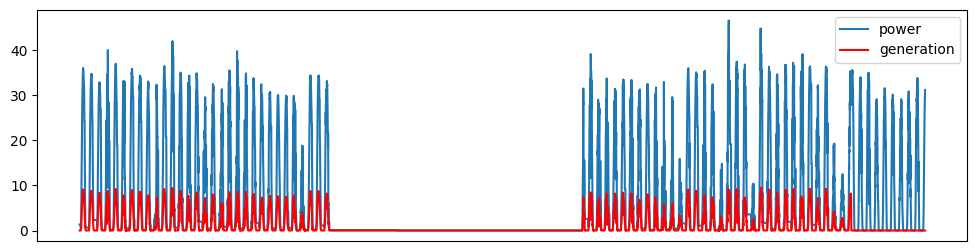




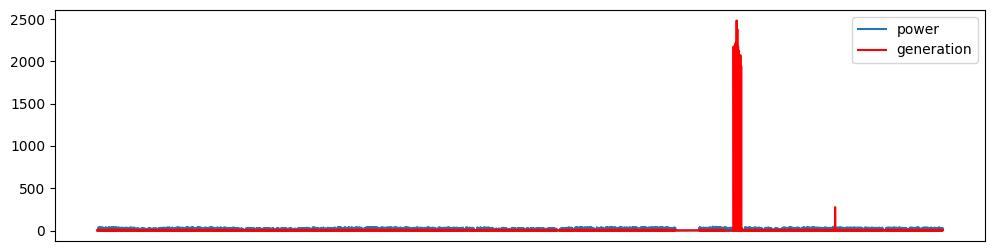
重新画图观察，发现还是有非常奇怪的裂缝，手动寻找一下具体位置。



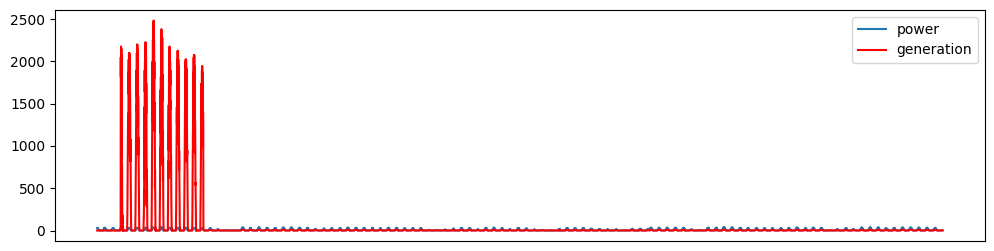
手动找了一下，大约再70000-80000之间的位置，数据是相对正常的，因为没用电也没发电，反倒是后半段数据出现功率很大但没发出电的情况，这就是可能的异常了。



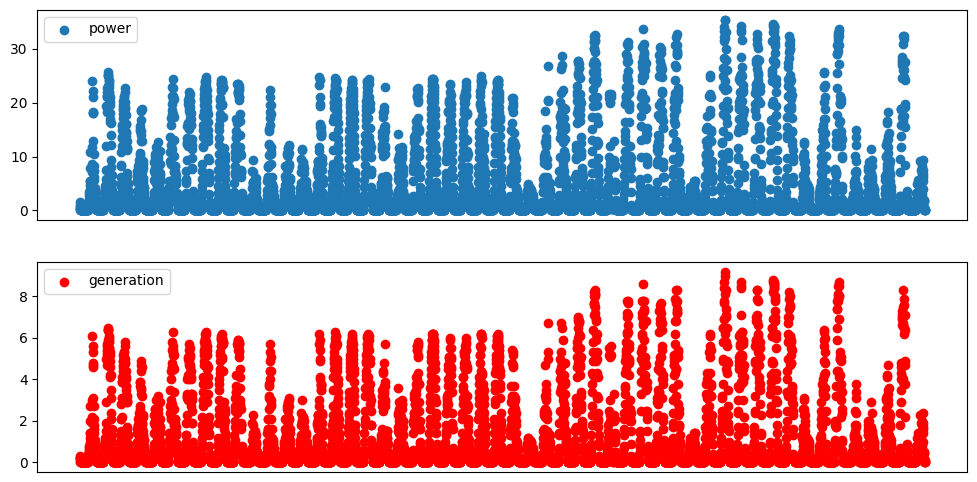
又画了一个两个指标同时存在的图，可见还有发电量爆炸的情况，也是异常之一。

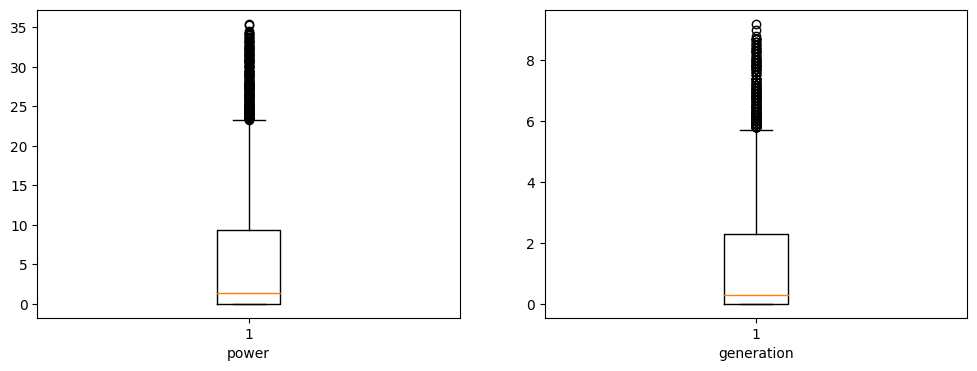


精确定位该位置，发现是在80000-90000之间



散点图、盒图也画一下，完成一下任务。

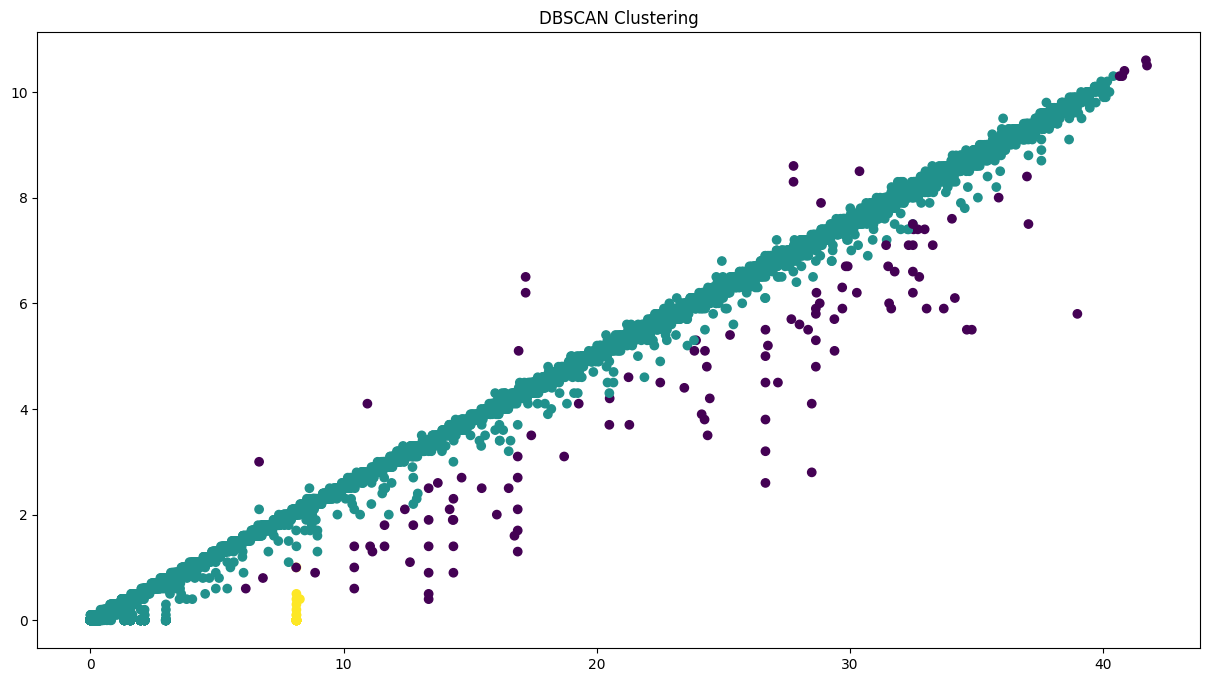


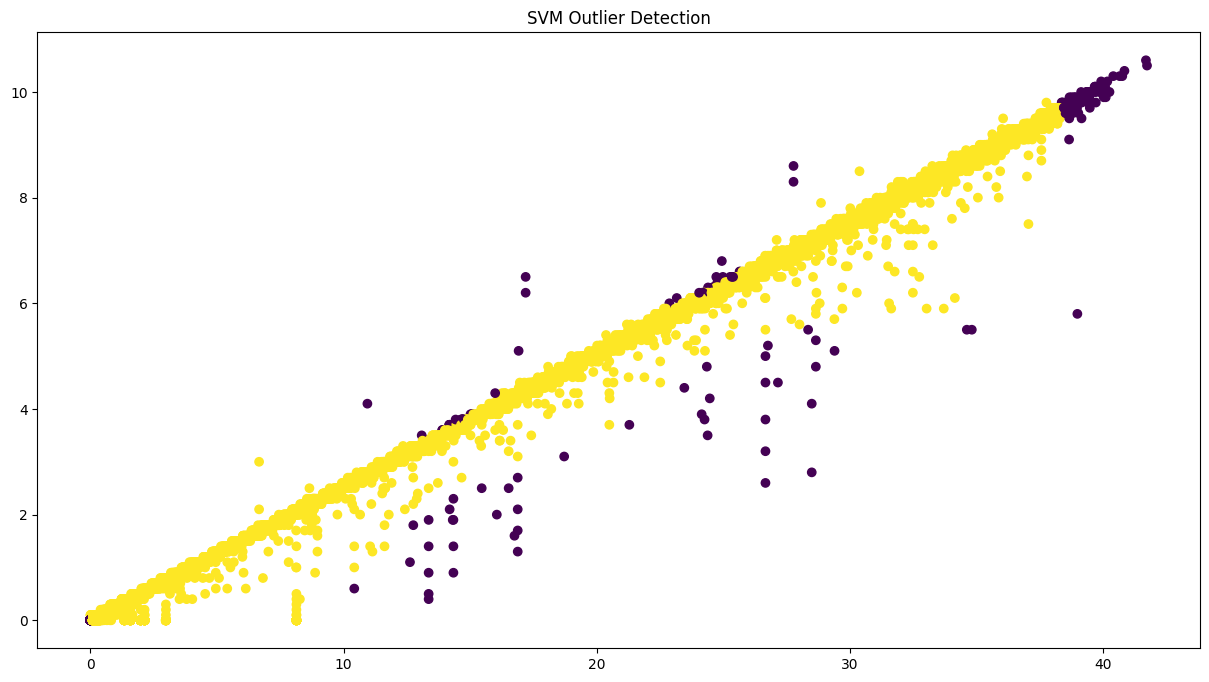


1. 直接利用某时刻的信息来判断异常。如现有121542个数据时刻形成同样大小的数据样本，每个样本由两个属性：功率和发电量组成，用两种异常检测的方法**如DBSCAN或一类SVM判断异常样本（也可采用较先进的方法）**，并进行对比分析。**（注意：数据中存在遗失值的情况）**

**为了实现进展速度，此处采用的数据分为两部分，55000-70000的正常数据点和70000-85000的不正常数据点。**

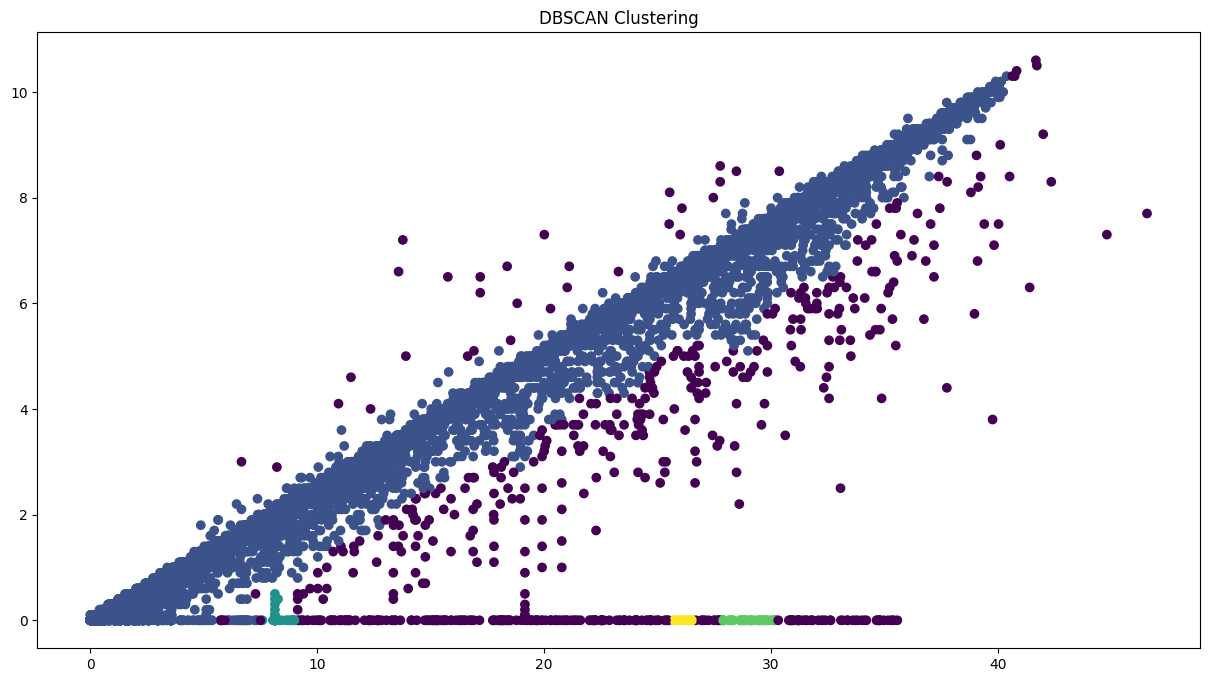
首先采用数据表现正常的55000-70000阶段进行测试分析，输入DBSCAN算法中，eps设置为0.5，即聚类范围为0.5，minsample为20，即超过20个算作一类。

****

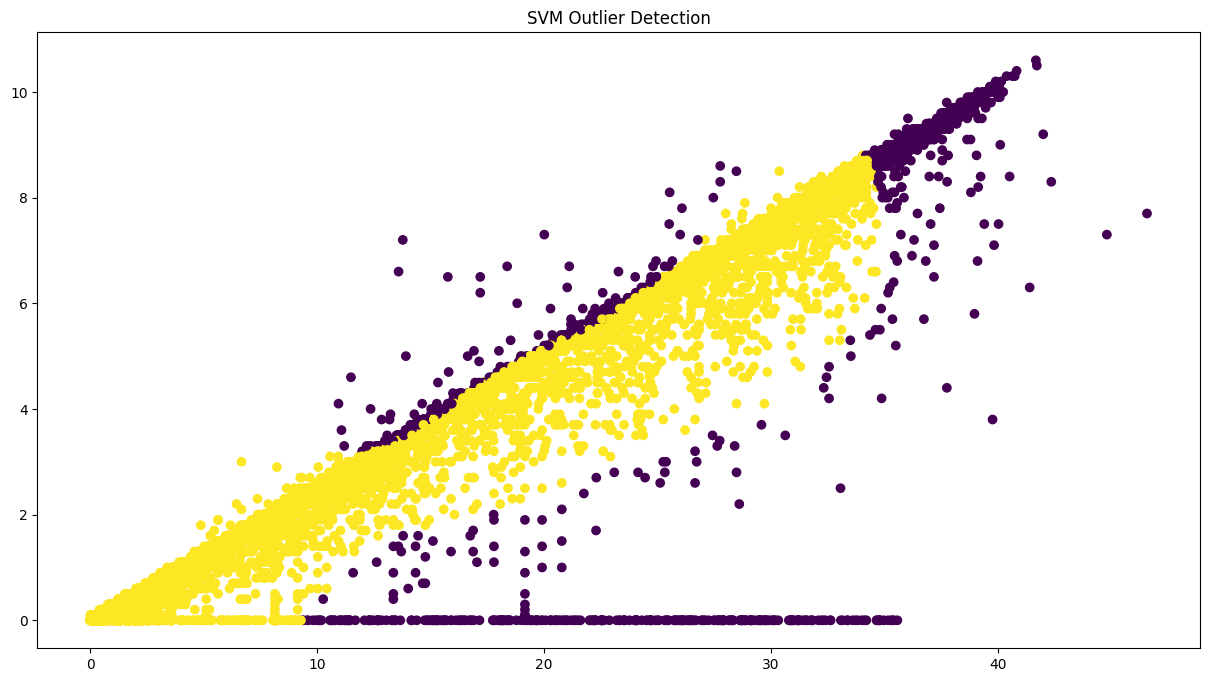
DBSCAN算法的表现较好，虽然离群点中有的较为密集构成新的类别，但不影响分类结果。

然后使用OneClassSVM进行分析。nu=0.02。这是目前SVM表现最好的一种情况了，但是在数值较大的情况下仍然会误分类，接下来引入不正常数据，即70000-85000之间的数据，参与聚类，观察异常检测效果。

首先是DBSCAN算法，依然拥有较好的表现。能够很好的将下部异常的部分区别开来，并且很好的分离出了离群点。



但是SVM这边就没有很好的表现了，它不能很好的区分开离群点，并且数值较大的部分被隔离出了正常范围。



但是我们观察所有的正常数据点可以观测到一个经验现象，那就是正常的数据往往是围绕着y=0.25x这个线性函数的，所以我们或许可以直接计算每个点到这个函数的距离，然后计算3 sigma，出界的就算异常点。

X = pv\_data[['power', 'generation']]

print(X.shape)

X['dist'] = np.abs(-1 \* X['power'] + 4 \* X['generation']) / np.sqrt(4\*\*2 + -1\*\*2)

print(X.shape)

# d = abs(A \* x1 + B \* y1 + C) / np.sqrt(A\*\*2 + B\*\*2)

# Ax + By + C = 0

# y = 0.25x

# 4x - y = 0

# 假设 df 是你的数据框

column\_name = 'dist'

upper\_bound = X['dist'].mean() + X['dist'].std() \* 3

lower\_bound = X['dist'].mean() - X['dist'].std() \* 3

print(f"{column\_name}的 3σ范围：{lower\_bound} ~ {upper\_bound}")

# X['color'] = 0 if X['dist'] >= lower\_bound and X['dist'] <= upper\_bound else 1

X['color'] = X['dist'].apply(lambda x: 0 if x >= lower\_bound and x <= upper\_bound else 1)

print(X.shape)

print(X['color'].value\_counts())

X = X.to\_numpy()

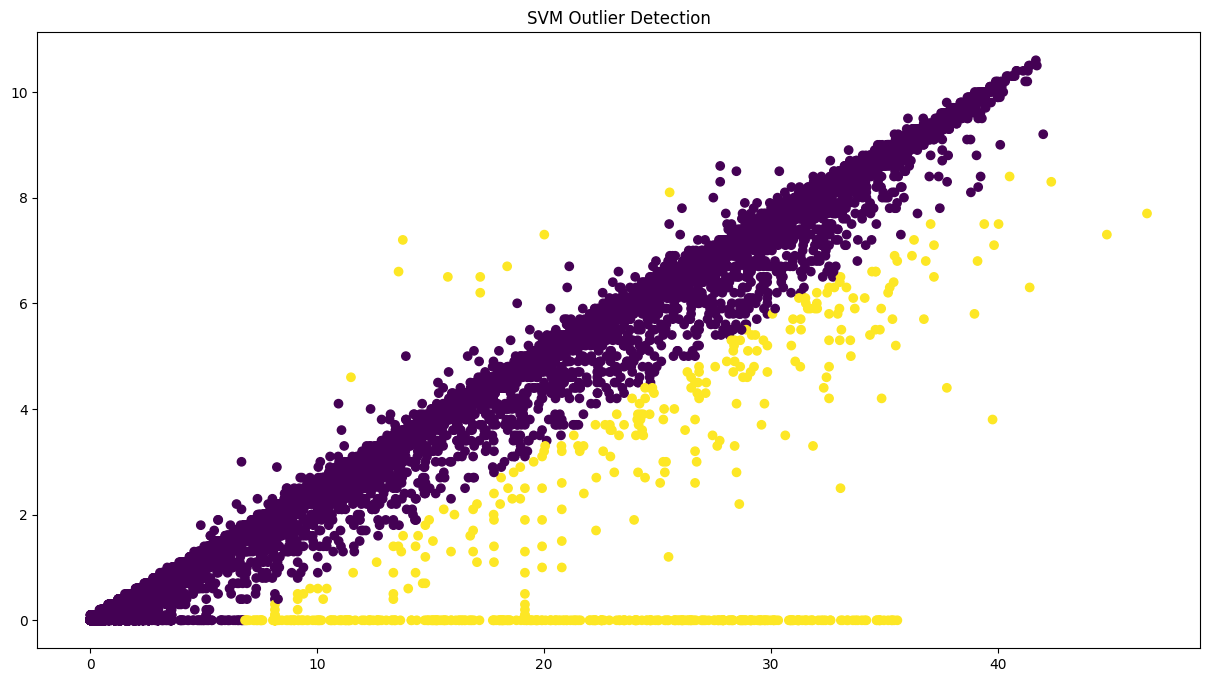
# 绘制结果

plt.figure(figsize=(15, 8))

plt.scatter(X[:, 0], X[:, 1], c=X[:, 3])

plt.title('SVM Outlier Detection')

plt.show()



这样看起来分类效果很不错，虽然容纳进去了一部分下部0值异常点，但是仍然强于SVM。

1. 利用功率这一物理量的数据趋势信息来判断异常。一般针对功率这一时间序列数据采用滑动窗口+步长的方式形成多属性样本，采用两种异常检测的方法如DBSCAN和一类SVM判断异常样本，并进行相应的对比分析。

注：利用时间序列的数据采用滑动窗口+步长的方式生成样本的代码见dataset\_power.py，修改窗口、步长的大小，统计样本数量，并分析异常检测的结果

由于特征数变多了，而且多属性完全是由滑动窗口带来的，我首先想到的是利用describe函数进行统计量构建。

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

pv\_data = pd.read\_csv('power.txt', header=None).dropna(how='all').fillna(0).drop\_duplicates()

print(pv\_data.shape)

def func(x : pd.Series):

    res = x.describe()

    res['skew'] = x.skew()

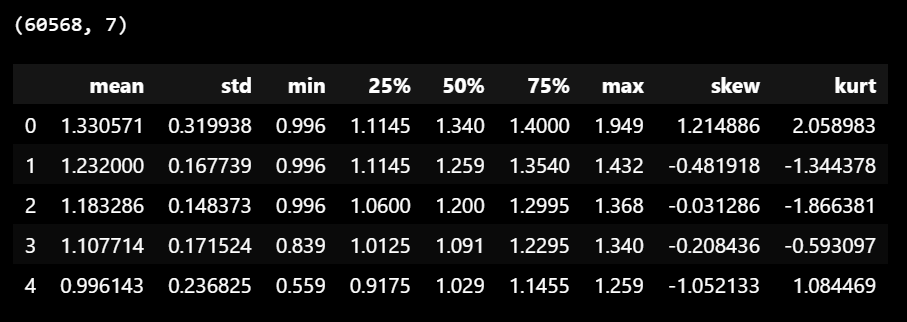
    res['kurt'] = x.kurt()

    return res

pv\_static = pv\_data.apply(func, axis=1).drop('count', axis=1)

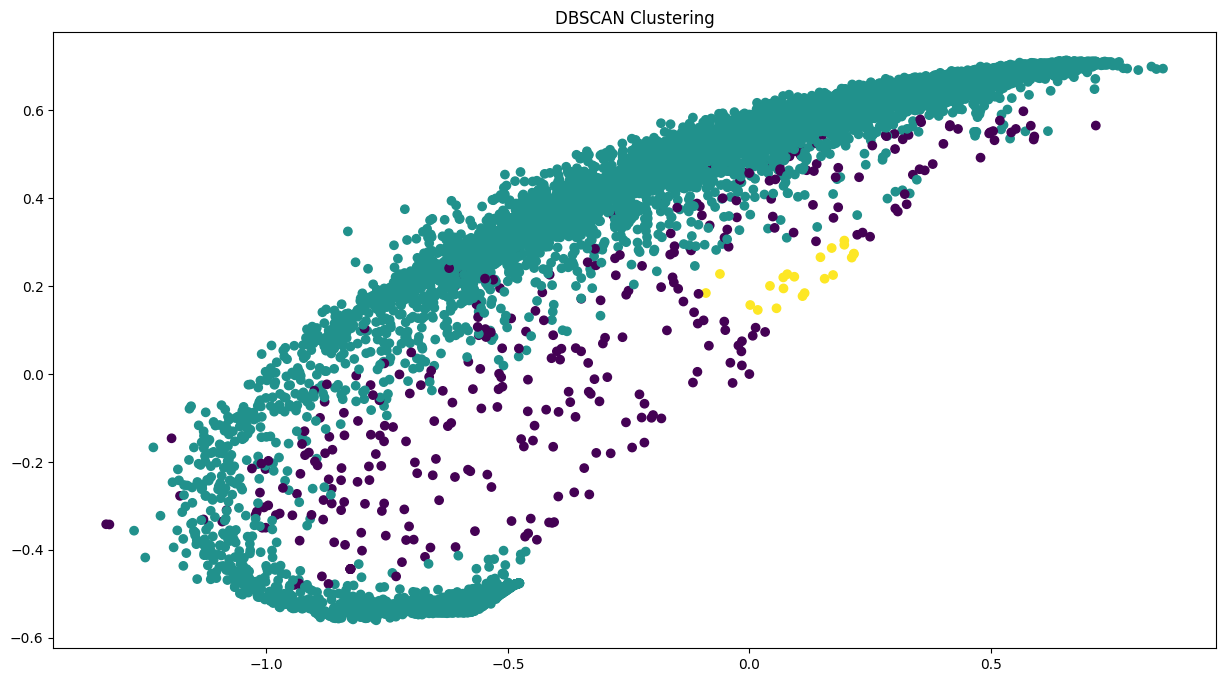
pv\_static.head()

除去无用的全0点，计算统计量，包括分位数，均值，标准差，偏度和峰度，等待接下来的DBSCAN和SVM。



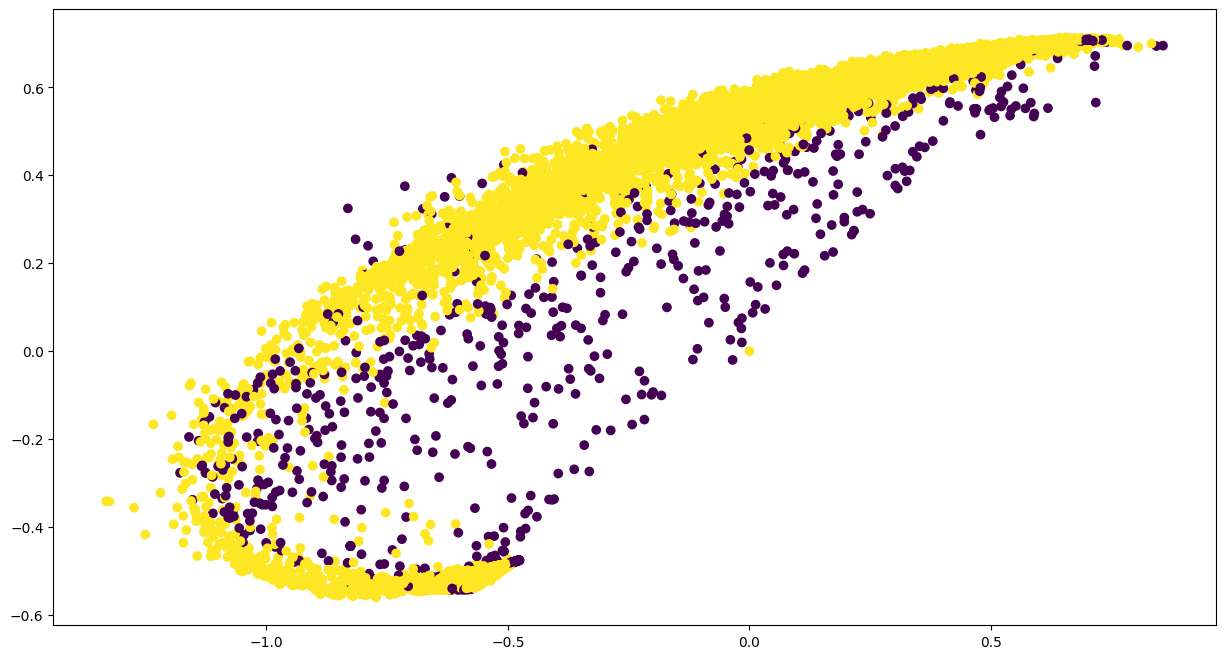
然后由于这些统计量之间具有较强的关联性，此处对除前两列外做标准归一化，然后drop掉前两列，接下来就是一个类似manifold的数据了。

首先使用DBSCAN进行聚类。



可见DBSCAN依旧强劲，可以检测出离群点。

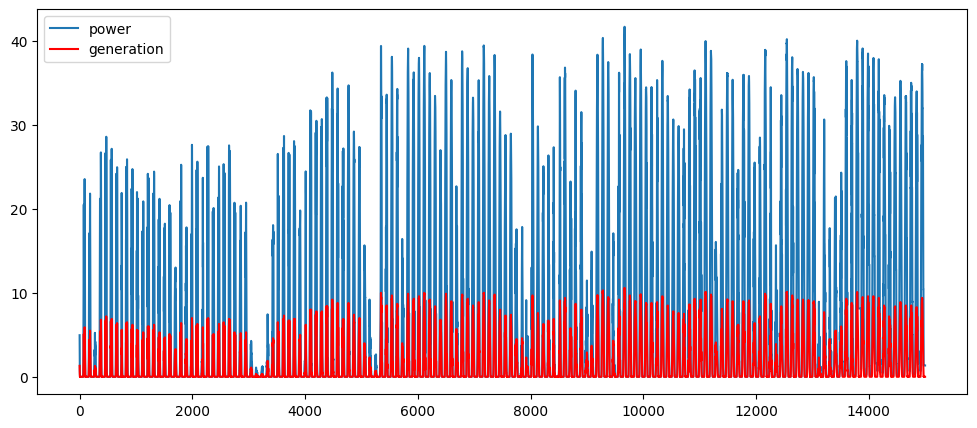
然后看SVM的表现。



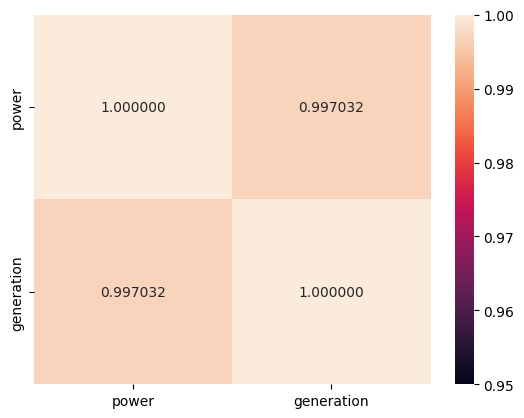
效果与dbscan相近，但是不好评价，感觉都差不多。

1. 探索同时利用功率和发电量两个物理量的数据变化趋势信息来判断异常样本。**（该任务选做，有一定程度的加分）**

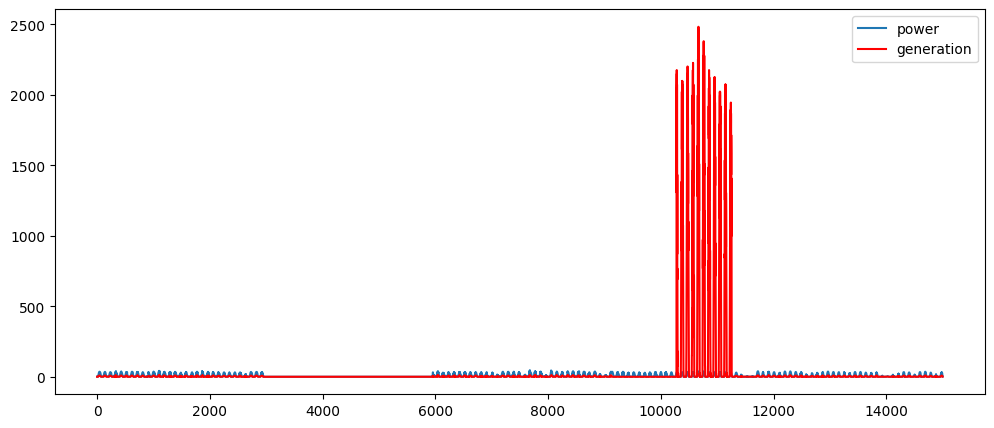
首先观测生数据55000-70000处，打印plot图像。

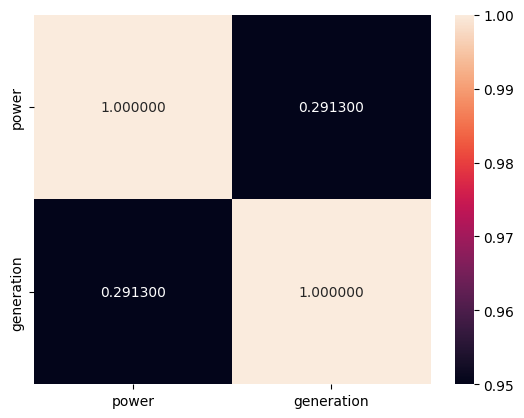


可见其线性相关性极强，所以利用两个物理量的变化趋势，首先就可以从相关性强弱入手进行判断，为了证实推测，计算二者的pearson相关系数矩阵，打印热图观察。



二者的线性相关度高达99%，真离谱，现在使用错误数据进行测试。在数据70000-85000的异常数据区块中计算线性相关性。





其相关性直线下降，可见二者在强线性相关情况下是正常数据，反之为异常数据。我们首先计算全局数据中二者线性相关的函数，虽然上文中已经先验性的推测过了，但是并不科学。我们首先使用linear回归求解，然后对所有数据进行predict，计算准确度。

此处首先取用前六万条相对正常的数据训练线性回归模型。

data = pv\_data[['power','generation']][:60000]

from sklearn.preprocessing import StandardScaler

power\_scaler = StandardScaler()

generation\_scaler = StandardScaler()

data['power'] = power\_scaler.fit\_transform(data['power'].to\_numpy().reshape(-1, 1))

data['generation'] = generation\_scaler.fit\_transform(data['generation'].to\_numpy().reshape(-1, 1))

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

X = data[['power']]

y = data['generation']

lr.fit(X=X, y=y)

y\_pred = lr.predict(X)

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

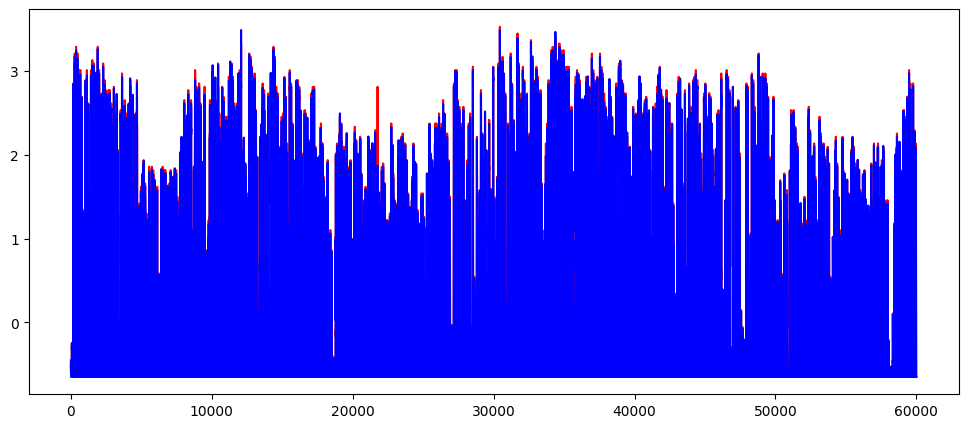
y\_pred = generation\_scaler.inverse\_transform(y\_pred.reshape(-1, 1))

y = generation\_scaler.inverse\_transform(y.to\_numpy().reshape(-1, 1))

print(mean\_squared\_error(y\_pred, y))

print(mean\_absolute\_error(y\_pred, y))

图像表示其精确度极高。



计算其平均mae为0.03774908462954722，平均mse为0.02273548027575829，几乎为0。

接下来为了保证实验具有普遍性，采用全部数据。

x = power\_scaler.transform(pv\_data['power'].to\_numpy().reshape(-1, 1))

x = pd.DataFrame(x, columns=['power'])

x = lr.predict(x)

pv\_data['generation\_pred'] = generation\_scaler.inverse\_transform(x.reshape(-1, 1))

pv\_data['sub'] = np.abs(pv\_data['generation\_pred'] - pv\_data['generation']) \*\* 2

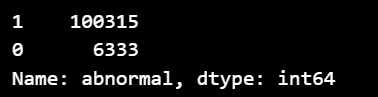
pv\_data.head()

为了更好地割开异常值，此处对所有数据计算误差平方，与mse进行对比。

v\_data['abnormal'] = pv\_data['sub'].apply(lambda x: 0 if x >= 0.0227 else 1)

print(pv\_data['abnormal'].value\_counts())

统计离群点总数可得。



大约6%的异常点，非常合理，打印图像看看。

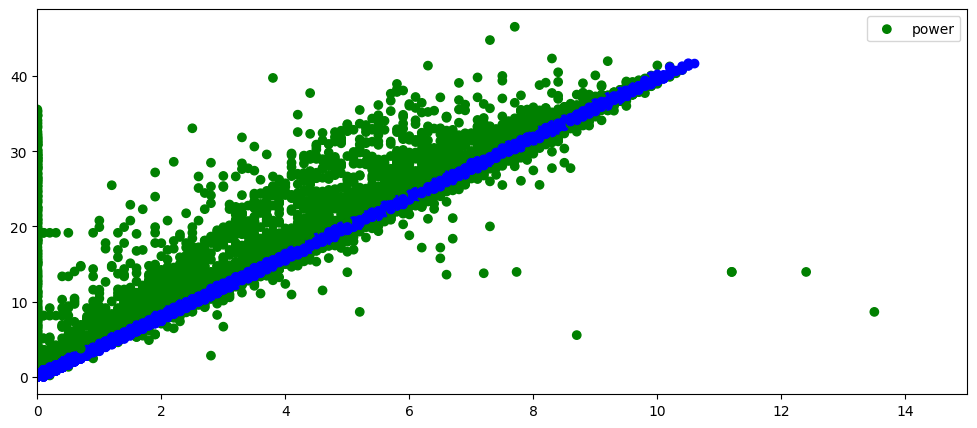
plt.figure(figsize=(12, 5))

pv\_data['color'] = pv\_data['abnormal'].apply(lambda x: 'green' if x == 0 else 'blue')

plt.scatter(pv\_data['generation'], pv\_data['power'], c=pv\_data['color'])

plt.legend()

plt.xlim(0, 15)



清晰的分界线，极高的精确度，完全依赖于前期的特征工程。

PS：大部分离谱的点在界外，被plt.xlim(0, 15)过滤了，没有包进图像。

1. **结论**

**异常检测没听说过，这是第一次接触，上手了DBSCAN算法，我很深信一句话就是特征工程的好坏直接决定最终精确度，在此次实践中我充分实践了这一点并且取得了极好的效果。**