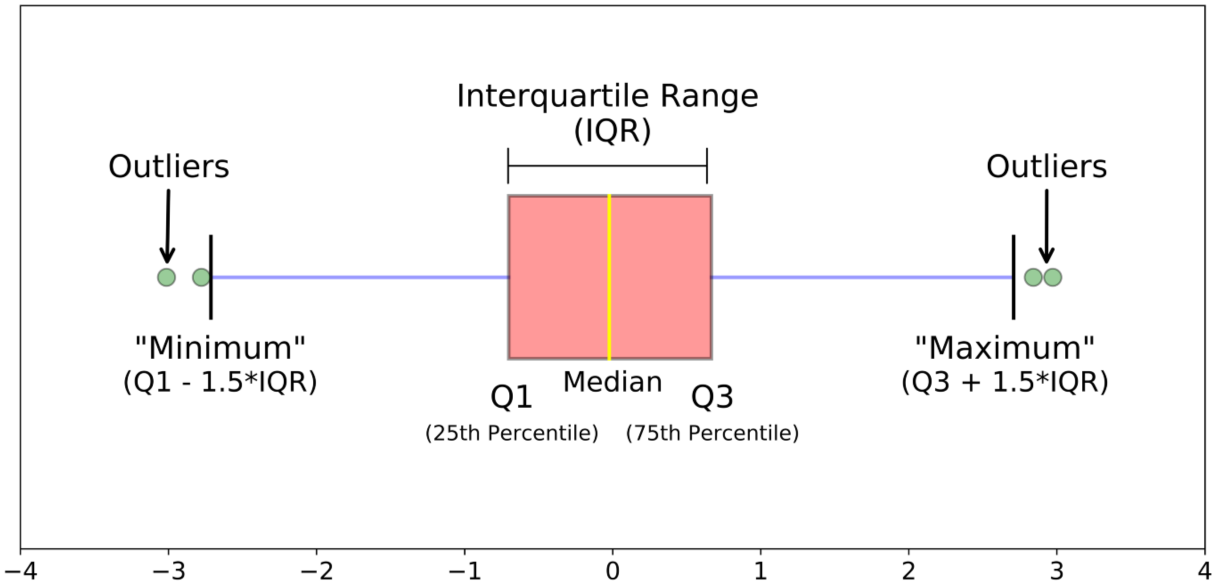
Boxplot

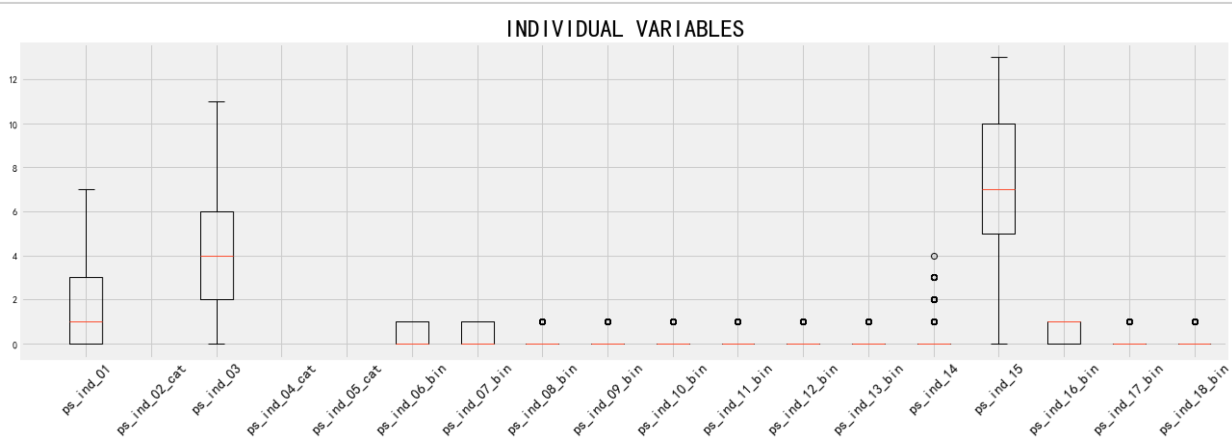


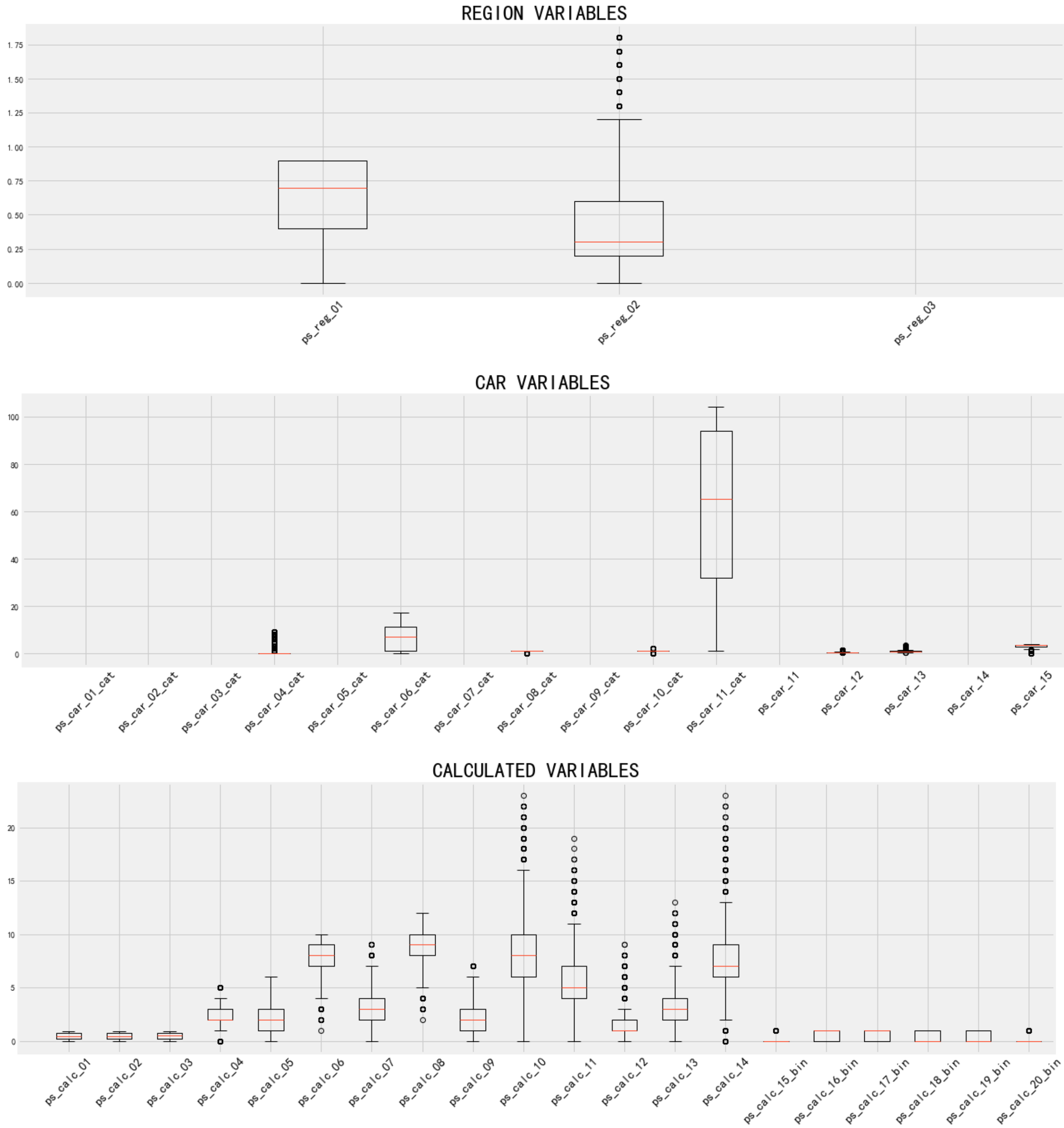
In [4]:
```
sample = fullset.sample(random_state=0,frac = .1)
sample = sample.replace(-1,np.nan)
```

In [5]:
```
from data_management import meta
```

In [6]:
```
metadata = meta(train,test)
```

In [7]:
```
plt.figure(figsize = (20,6))
ind_idx = metadata[metadata['category'] == 'individual'].index
ticks = list(ind_idx)
ticks = [''] + ticks
box1 = plt.boxplot(sample[ind_idx])
rang = range(0, len(ticks))
plt.title("individual variables".upper(),fontsize = 25)
plt.xticks(rang, ticks, rotation=45, fontsize = 15)
pass

plt.figure(figsize = (20,6))
ind_idx = metadata[metadata['category'] == 'region'].index
ticks = list(ind_idx)
ticks = [''] + ticks
box2 = plt.boxplot(sample[ind_idx])
rang = range(0, len(ticks))
plt.title("region variables".upper(),fontsize = 25)
plt.xticks(rang, ticks, rotation=45, fontsize = 15)
pass

plt.figure(figsize = (20,6))
ind_idx = metadata[metadata['category'] == 'car'].index
ticks = list(ind_idx)
ticks = [''] + ticks
box3 = plt.boxplot(sample[ind_idx])
rang = range(0, len(ticks))
plt.title("car variables".upper(),fontsize = 25)
plt.xticks(rang, ticks, rotation=45, fontsize = 15)
pass

plt.figure(figsize = (20,6))
ind_idx = metadata[metadata['category'] == 'calculated'].index
ticks = list(ind_idx)
ticks = [''] + ticks
box4 = plt.boxplot(sample[ind_idx])
rang = range(0, len(ticks))
plt.title("calculated variables".upper(),fontsize = 25)
plt.xticks(rang, ticks, rotation=45, fontsize = 15)
pass
```





In [8]:
```python
# from matplotlib.cbook import boxplot_stats
# boxplot_data = sample[metadata.index]
# for idx,i in enumerate(boxplot_stats(boxplot_data)):
#     outliers = i['fliers'] if len(i['fliers']) > 0 else 'No Outliers'
#     print(f"{boxplot_data.columns[idx]}==>{outliers}")
```

Zscore

> The Z-score（标准分数） is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured.

> Z-score is finding the distribution of data where mean is 0 and standard deviation is 1 i.e. normal distribution.

$$z = \frac{(x - \mu)}{\sigma}$$

In [9]:
```
from scipy import stats
z_score = np.abs(stats.zscore(fullset))
```

In [10]:
```
threshold = 3
np.where(z_score > 3)
```

Out[10]:
```
(array([      2,       5,      10, ..., 1488023, 1488023, 1488023],
       dtype=int64),
 array([ 3, 21, 26, ..., 26, 35, 36], dtype=int64))
```

In [11]:
```
np.where(z_score > 3)[0].shape
```

Out[11]:
```
(524548,)
```

In [12]:
```
# for r,c,in zip(np.where(z_score > 3)[0],np.where(z_score > 3)[1]):
#     print(fullset.iloc[r,c])
```

IQR

> The interquartile range (IQR), also called the midspread or middle 50%, or technically H-spread, is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = Q3 − Q1.

In [13]:
```
IFrame(width="853",height="480",src = "https://www.youtube.com/embed/wsKyhc
```

Out[13]:

In [14]:
```
iqr_df = fullset.drop(['id','target'],axis=1)
Q1 = iqr_df.quantile(0.25)
Q3 = iqr_df.quantile(0.75)
IQR = Q3 - Q1
```

In [15]:
```
cond = (iqr_df < (Q1 - 1.5 * IQR)) |(iqr_df > (Q3 + 1.5 * IQR))
```

In [16]:
```
iqr_df[~cond].replace(np.nan,'异常值')
```

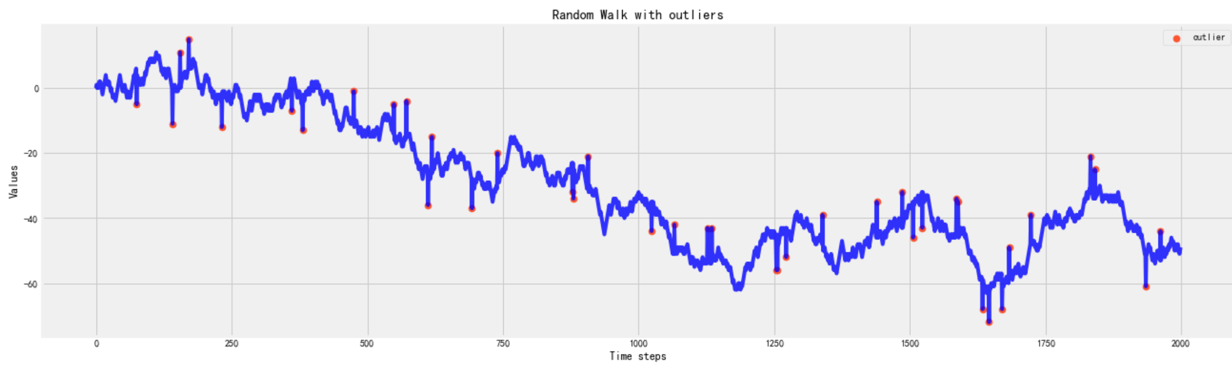
Out[16]:

	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_(
0	2	2.0	5	1	0.0	0	
1	1	1.0	7	0	0.0	0	
2	5	异常值	9	1	0.0	0	
3	0	1.0	2	0	0.0	1	
4	0	2.0	0	1	0.0	1	
...	...	...	...	...	...	...	
1488023	0	1.0	6	0	0.0	0	
1488024	5	3.0	5	1	0.0	0	
1488025	0	1.0	5	0	0.0	1	
1488026	6	1.0	5	1	0.0	0	
1488027	7	1.0	4	1	0.0	0	

1488028 rows × 57 columns

In [17]:
```
def random_walk_with_outliers(origin, n_steps, perc_outliers=0.0, outlier_m

    assert (perc_outliers >= 0.0) & (perc_outliers <= 1.0)

    #set seed for reproducibility
    np.random.seed(seed)

    # possible steps
    steps = [-1, 1]

    # simulate steps
    steps = np.random.choice(a=steps, size=n_steps-1)
    rw = np.append(origin, steps).cumsum(0)

    # add outliers
    n_outliers = int(np.round(perc_outliers * n_steps, 0))
    indices = np.random.randint(0, len(rw), n_outliers)
    rw[indices] = rw[indices] + steps[indices + 1] * outlier_mult

    return rw, indices
```

In [18]:
```
plt.figure(figsize = [20,6])
rw, outlier_ind = random_walk_with_outliers(0, 2000, 0.02,seed = 2022)
plt.plot(np.arange(len(rw)), rw, c = 'b',alpha = .8)
plt.scatter(outlier_ind, rw[outlier_ind], c='#fc5531', label='outlier',s =
plt.title('Random Walk with outliers')
plt.xlabel('Time steps')
plt.ylabel('Values')
plt.legend();
```



In [19]:
```
def evaluation(series, true_indices, detected_indices):
    # calculate metrics
    tp = list(set(detected_indices).intersection(set(true_indices)))
    fp = list(set(detected_indices).difference(set(true_indices)))
    fn = list(set(true_indices).difference(set(detected_indices)))
    perc_detected = 100 * len(tp) / len(true_indices)

    # create the plot
    fix, ax = plt.subplots(2, 1, figsize=(25, 6*3))

    ax[0].plot(np.arange(len(series)), series,c = '#24292e',alpha = .8, lin
    ax[0].scatter(true_indices, series[true_indices], c='g', label='true ou
    ax[0].set_title('实际离群值',fontsize = 35)
    ax[0].legend(fontsize = 20)

    ax[1].plot(np.arange(len(series)), series, c = '#3a9cfb',alpha = .8, li
    ax[1].scatter(tp, series[tp], c='g', label='true positive',s = 100)
    ax[1].scatter(fp, series[fp], c='r', marker = 'x', label='false positiv
    ax[1].scatter(fn, series[fn], c='k', marker = 'x', label='false negativ
    ax[1].set_title('处理后结果对比',fontsize = 35)
    ax[1].legend(fontsize = 20)


    # print out summary
    print('-' * 25 + ' Summary ' + '-' * 25)
    print(f'序列中离群值数量: {len(true_indices)}')
    print(f'所检测出来的数量: {len(detected_indices)}')
    print(f'检测正确的数量: {len(tp)} ({perc_detected:.2f}% of all outliers).
    print('-' * 59)

    return tp, fp, fn
```

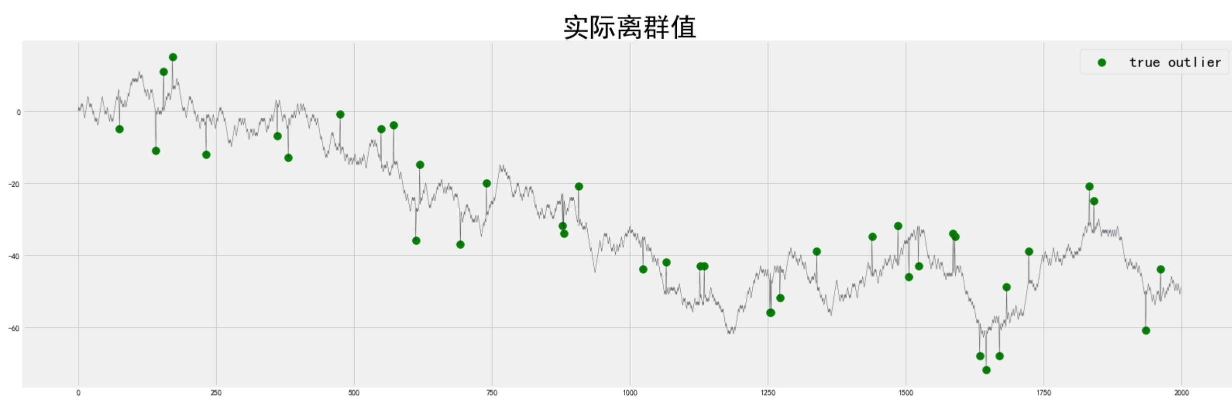
In [20]:
```
def hampel_filter(input_series, window_size, n_sigmas=3):

    n = len(input_series)
    new_series = input_series.copy()

    k = 1.4826

    indices = []

    # possibly use np.nanmedian
    for i in range((window_size),(n - window_size)):
        window_median = np.median(input_series[(i - window_size):(i + windo
        k_MAD = k * np.median(np.abs(input_series[(i - window_size):(i + wi
        if (np.abs(input_series[i] - window_median) > n_sigmas * k_MAD):
            new_series[i] = window_median
            indices.append(i)
    return new_series, indices
```

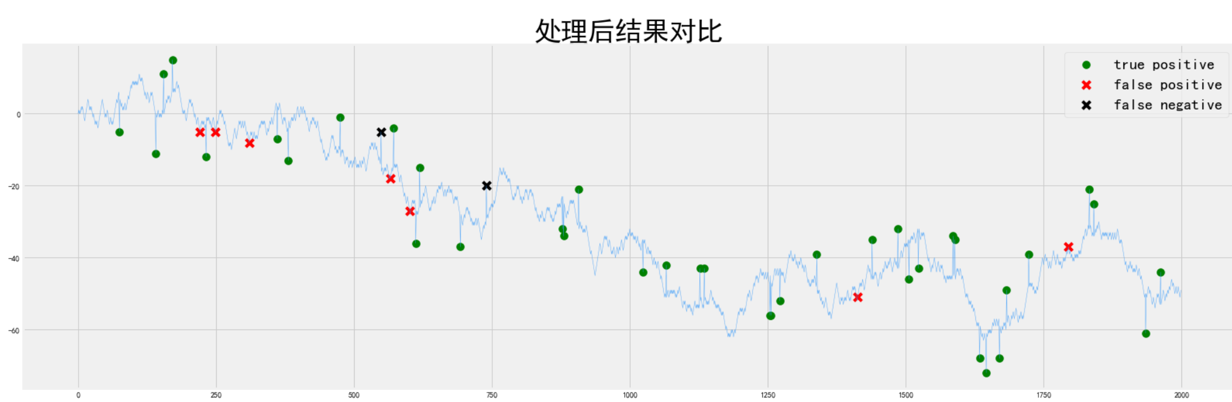
In [21]:
```
def iqr_filter(input_series, window_size):

    n = len(input_series)
    new_series = input_series.copy()
    indices = []
    # possibly use np.nanmedian
    for i in range((window_size),(n - window_size)):
        s = input_series[(i - window_size):(i + window_size)]
        s = pd.Series(s)
        Q1 = s.quantile(0.25)
        Q3 = s.quantile(0.75)
        IQR = Q3 - Q1
        cond = (input_series[i] < (Q1 - 1.5 * IQR)) |(input_series[i] > (Q3
        if cond:
            new_series[i] = input_series[i]
            indices.append(i)
    return new_series, indices
```

In [22]:
```
res_iqr, detected_outliers_iqr = iqr_filter(rw, 10)
res_hampel, detected_outliers_hampel = hampel_filter(rw, 10)
```

In [23]:
```
# Hampel
tp, fp, fn = evaluation(rw, outlier_ind, detected_outliers_hampel)
```

```
------------------------- Summary -------------------------
序列中离群值数量: 40
所检测出来的数量: 45
检测正确的数量: 38 (95.00% of all outliers).
-----------------------------------------------------------
```

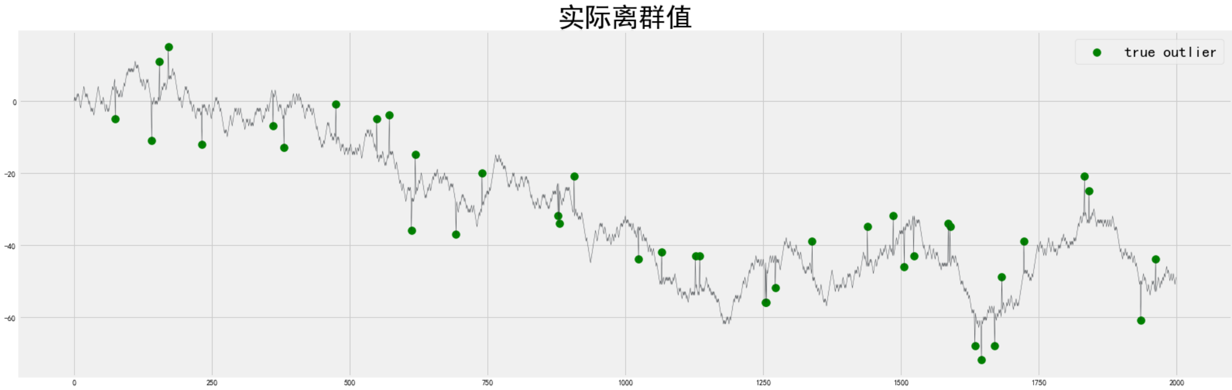




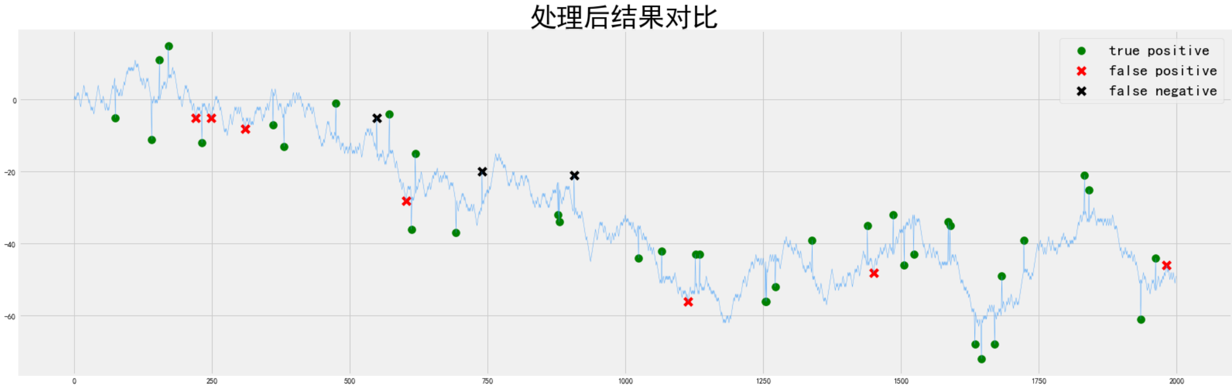
In [24]:
```
# IQR
tp, fp, fn = evaluation(rw, outlier_ind, detected_outliers_iqr)
```

```
------------------------- Summary -------------------------
序列中离群值数量: 40
所检测出来的数量: 44
检测正确的数量: 37 (92.50% of all outliers).
-----------------------------------------------------------
```

实际离群值



处理后结果对比



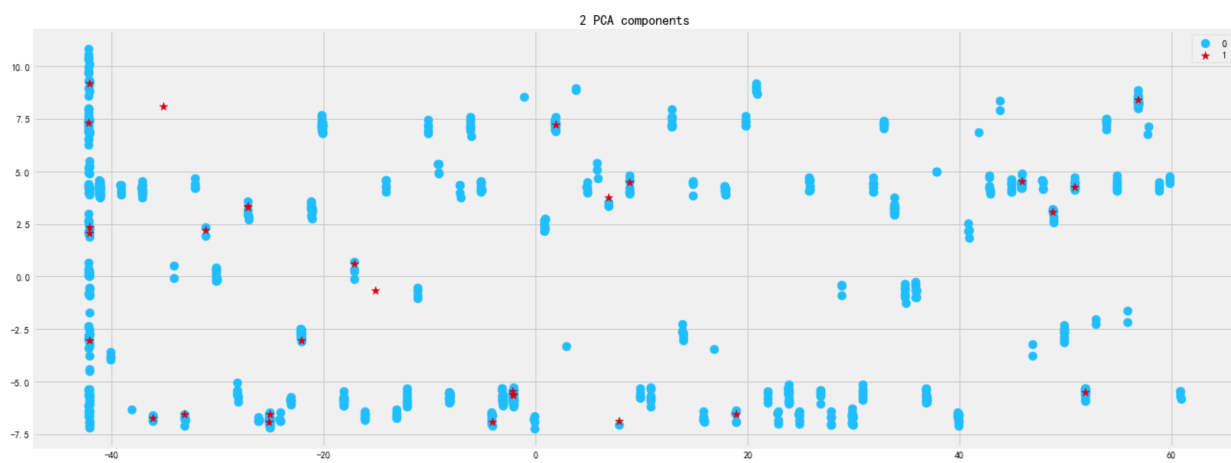
Outliers are not labeled! (otherwise it’s just imbalanced classification)

在没有数据先验知识的情况下确定异常值。这属于于无监督聚类。

In [25]:
```
from sklearn.cluster import DBSCAN
```

In [26]:
```
def plot_2d_space(X, y, label='Classes'):

    colors = ['#20beff', '#d80012']
    markers = ['o', '*']
    plt.figure(figsize = [20,8])
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
            X[y==l, 1],
            c=c, label=l, marker=m,
            s = 80
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()
```

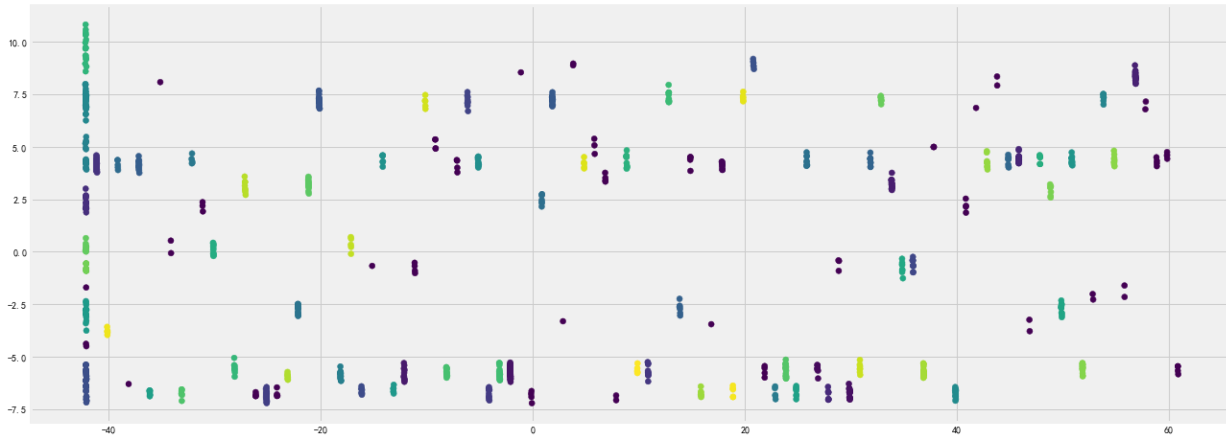
In [27]:
```
df = train.sample(1000)
X = df.drop(['id','target'],axis=1)
y = df.target

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plot_2d_space(X_pca, y, '2 PCA components')
```



In [28]:
```
dbscan=DBSCAN()
dbscan.fit(X_pca)
pass
```

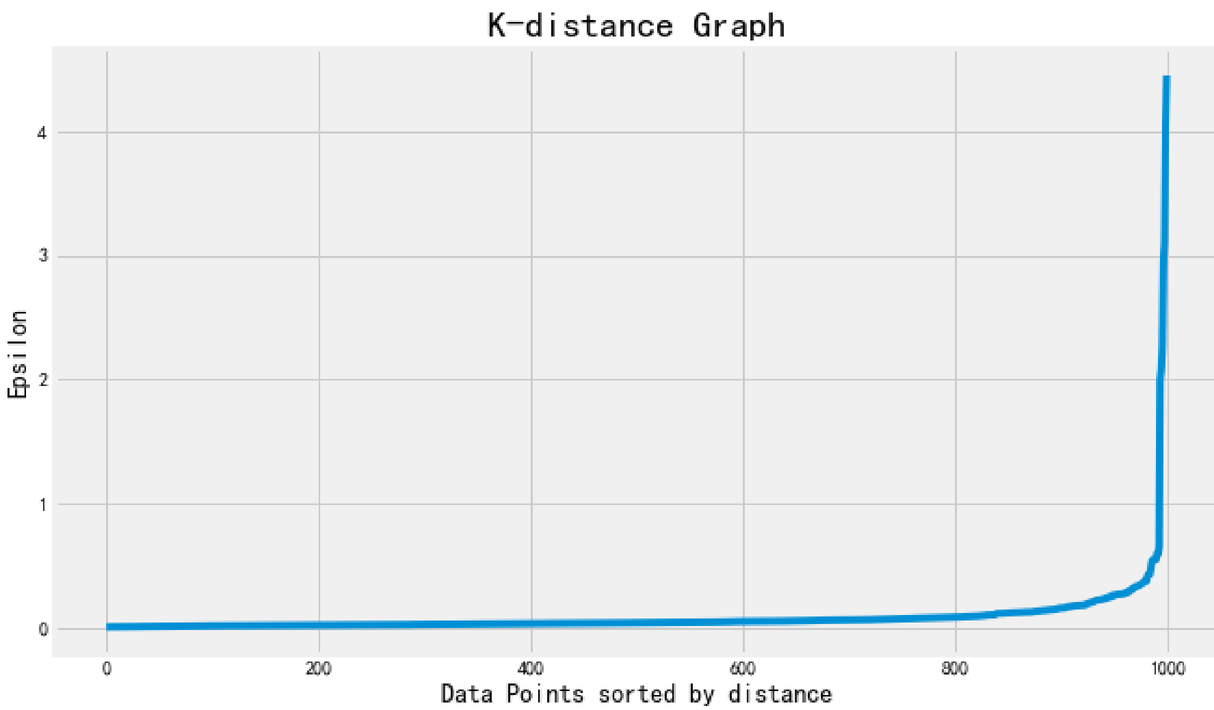
In [29]:
```
# visualize outputs
colors = dbscan.labels_
plt.figure(figsize = [20,8])
plt.scatter(X_pca[:,0], X_pca[:,1], c = colors)
```

Out[29]:
```
<matplotlib.collections.PathCollection at 0x23bc85842c8>
```



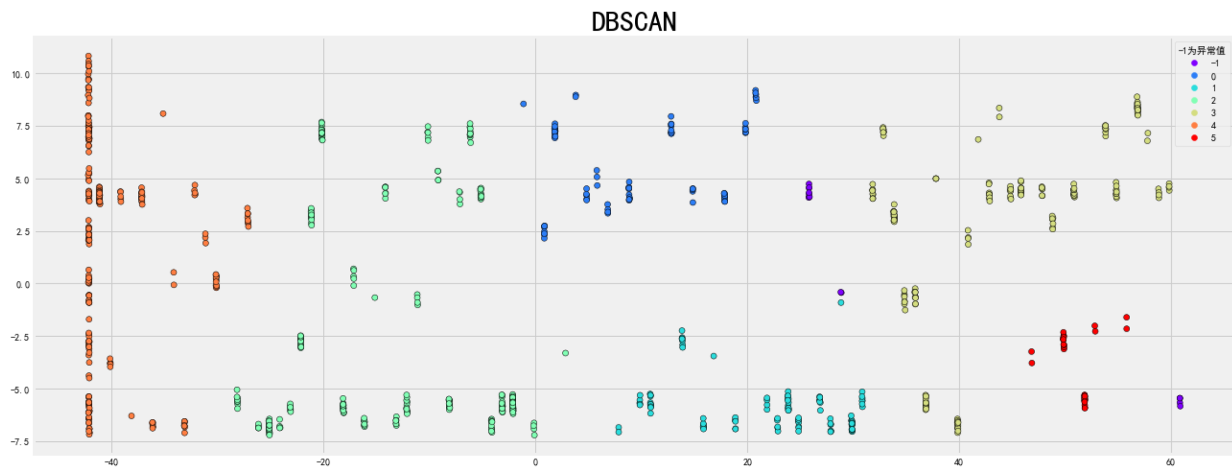
In [30]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=10)
nbrs = neigh.fit(X_pca)
distances, indices = nbrs.kneighbors(X_pca)
```

In [31]:
```
# Plotting K-distance Graph
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(10,6))
plt.plot(distances)
plt.title('K-distance Graph',fontsize=20)
plt.xlabel('Data Points sorted by distance',fontsize=14)
plt.ylabel('Epsilon',fontsize=14)
plt.show()
```



In [32]:
```
# 根据上图, eps = 5
model = DBSCAN(eps = 5, min_samples = 10).fit(X_pca)

colors = model.labels_
colormap = list(pd.Series(colors).unique())
plt.figure(figsize = [20,8])
scatter = plt.scatter(X_pca[:,0], X_pca[:,1], c = colors, cmap = 'rainbow',
plt.title('DBSCAN', fontsize = 30)
plt.legend(handles=scatter.legend_elements()[0],
           labels=sorted(colormap),
           title="-1为异常值")
```

Out[32]: <matplotlib.legend.Legend at 0x23bc8397848>

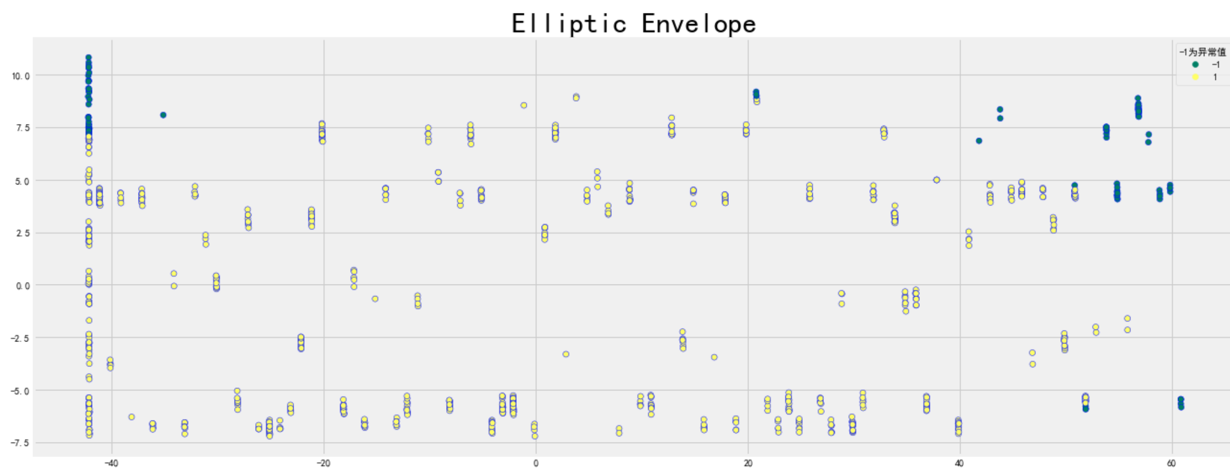


In [33]:
```
from sklearn.covariance import EllipticEnvelope
```

In [34]:
```
ee = EllipticEnvelope(contamination=0.1).fit(X_pca)
pred = ee.predict(X_pca)
```

In [35]:
```
colors = pred
colormap = list(pd.Series(colors).unique())
plt.figure(figsize = [20,8])
scatter = plt.scatter(X_pca[:,0], X_pca[:,1], c = colors, cmap = 'summer',e
plt.title('Elliptic Envelope', fontsize = 30)
plt.legend(handles=scatter.legend_elements()[0],
           labels=sorted(colormap),
           title="-1为异常值")
```

Out[35]: <matplotlib.legend.Legend at 0x23bc857c8c8>



In [36]:
```
IFrame(width="853",height="480",src = "https://www.youtube.com/embed/02I84i
```

Out[36]:

In [37]:
```
# D'Agostino and Pearson's Test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import normaltest
seed(1)
stat, p = normaltest(X_pca[:,0])
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

```
Statistics=1579.299, p=0.000
Sample does not look Gaussian (reject H0)
```

In [38]:
```
# D'Agostino and Pearson's Test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import normaltest
seed(1)
stat, p = normaltest(X_pca[:,1])
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

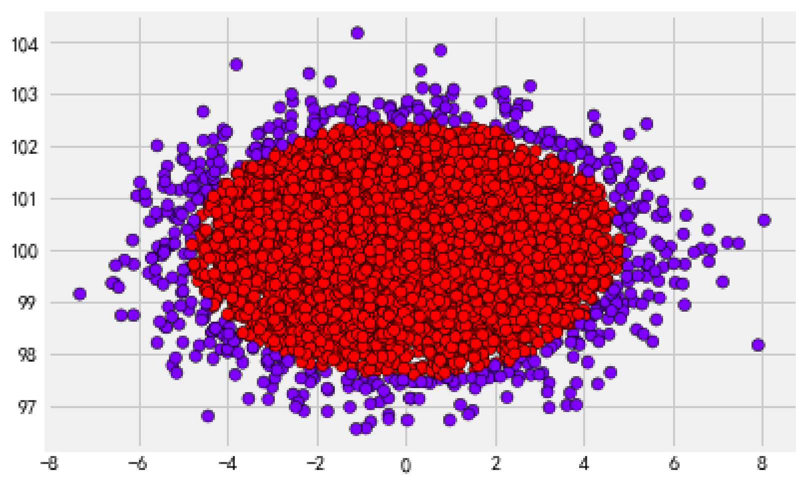
```
Statistics=5742.955, p=0.000
Sample does not look Gaussian (reject H0)
```

In [39]:
```
x1 = 2*np.random.randn(10000)
x2 = np.random.randn(10000)+100
```

In [40]:
```
gaussian_demo = np.concatenate((x1.reshape(-1,1),x2.reshape(-1,1)),axis=1)
ee = EllipticEnvelope(contamination=0.05).fit(gaussian_demo)
pred = ee.predict(gaussian_demo)
```

In [41]:
```
colors = pred
plt.scatter(x1, x2, c = colors, cmap = 'rainbow',edgecolors='black')
```

Out[41]:
```
<matplotlib.collections.PathCollection at 0x23bc86e7cc8>
```

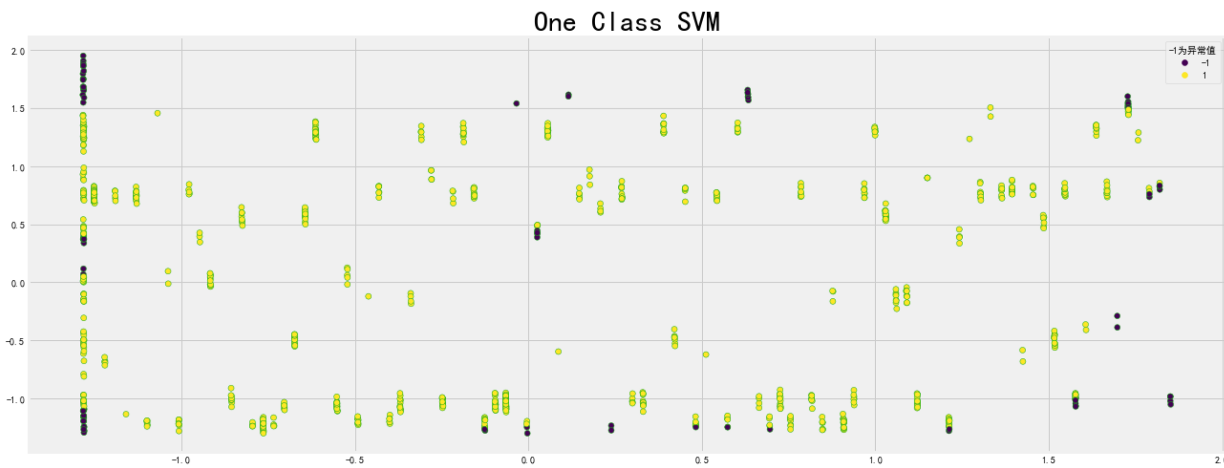


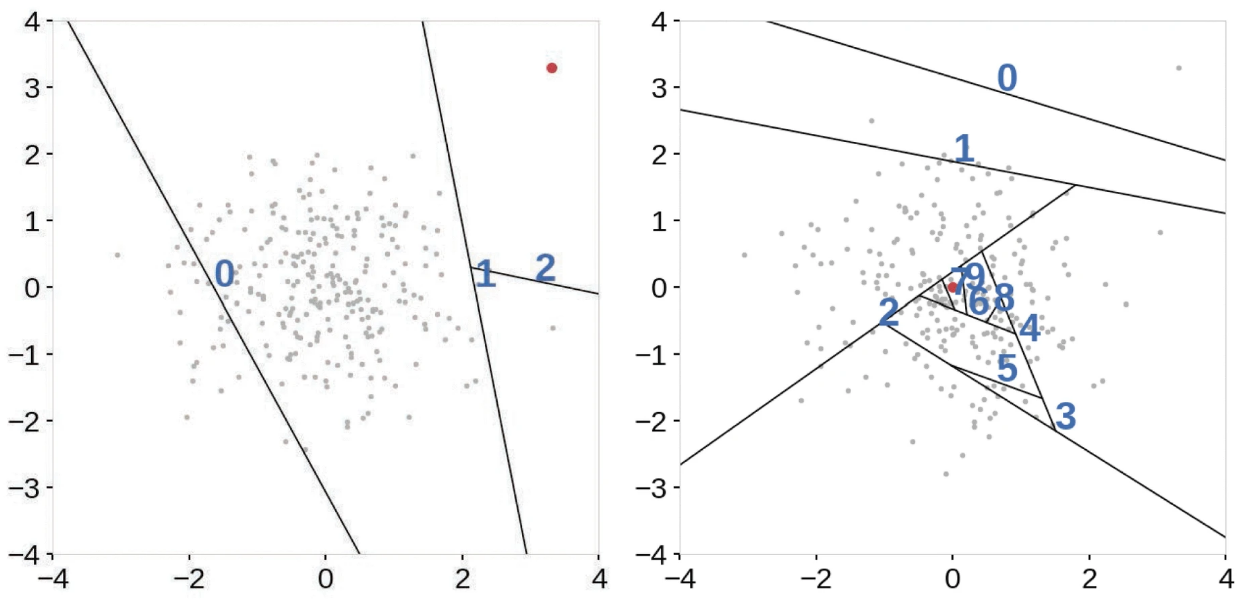
In [42]:
```
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
```

In [43]:
```
scaler = StandardScaler()
X_pca_scaled = scaler.fit_transform(X_pca)
oneclass = OneClassSVM(nu=.1).fit(X_pca_scaled)
pred = oneclass.predict(X_pca_scaled)
```

In [44]:
```
colors = pred
colormap = list(pd.Series(colors).unique())
plt.figure(figsize = [20,8])
scatter = plt.scatter(X_pca_scaled[:,0], X_pca_scaled[:,1], c = colors,edge
plt.title('One Class SVM', fontsize = 30)
plt.legend(handles=scatter.legend_elements()[0],
           labels=sorted(colormap),
           title="-1为异常值")
```

Out[44]: <matplotlib.legend.Legend at 0x23bc87a1308>





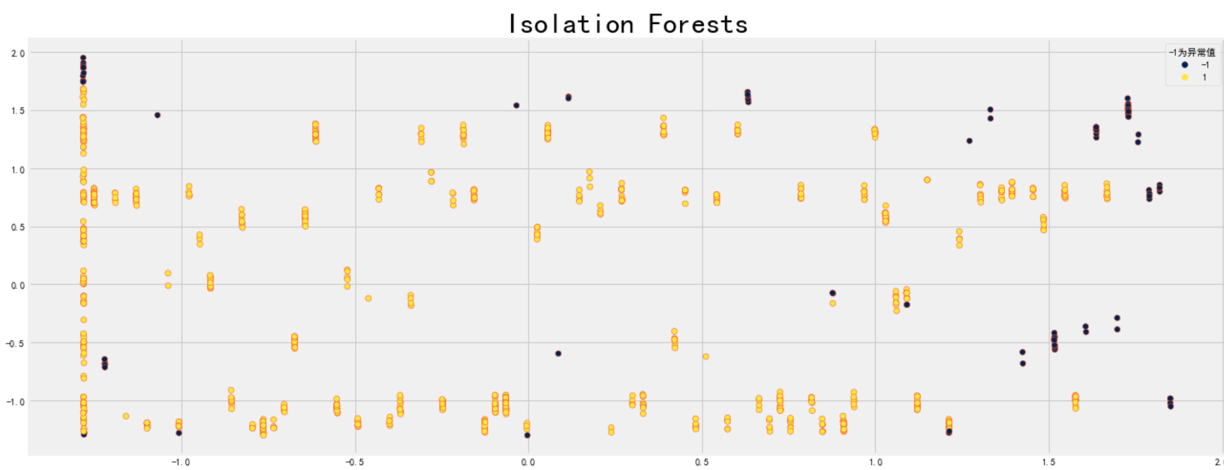
(a) Anomaly (b) Nominal

In [45]:
```
from sklearn.ensemble import IsolationForest
```

In [46]:
```
model=IsolationForest(n_estimators=50, max_samples='auto', contamination=fl
model.fit(X_pca_scaled)

scores =model.decision_function(X_pca_scaled)
pred =model.predict(X_pca_scaled)
```

In [47]:
```
colors = pred
colormap = list(pd.Series(colors).unique())
plt.figure(figsize = [20,8])
scatter = plt.scatter(X_pca_scaled[:,0], X_pca_scaled[:,1], c = colors, cma
plt.title('Isolation Forests', fontsize = 30)
plt.legend(handles=scatter.legend_elements()[0],
           labels=sorted(colormap),
           title="-1为异常值")
```

Out[47]: <matplotlib.legend.Legend at 0x23bc8829788>

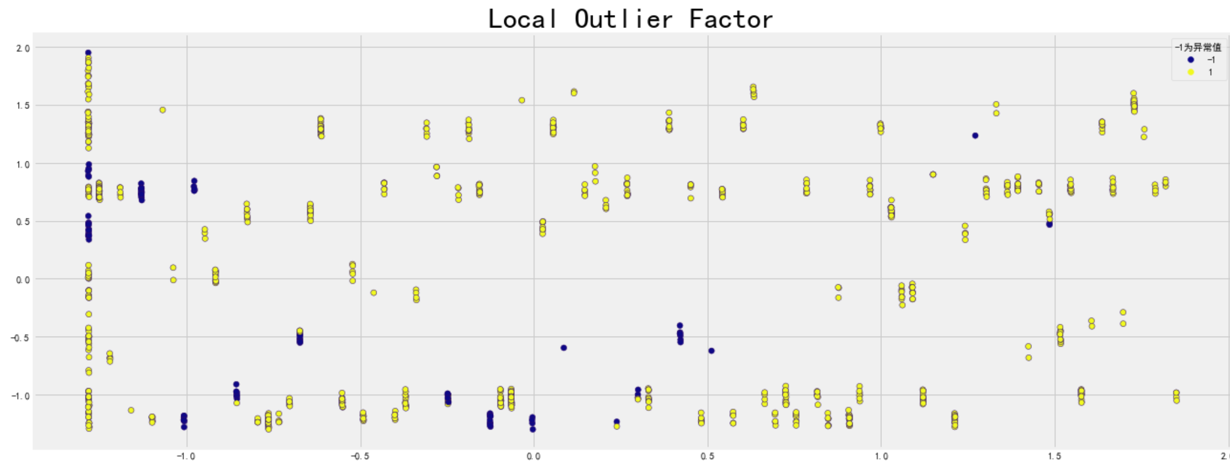


In [48]:
```
from sklearn.neighbors import LocalOutlierFactor
```

In [49]:
```
model = LocalOutlierFactor(n_neighbors=35, contamination=.1)
pred = model.fit_predict(X_pca_scaled)
scores = model.negative_outlier_factor_
```

In [50]:
```
colors = pred
colormap = list(pd.Series(colors).unique())
plt.figure(figsize = [20,8])
scatter = plt.scatter(X_pca_scaled[:,0], X_pca_scaled[:,1], c = colors, cma
plt.title('Local Outlier Factor', fontsize = 30)
plt.legend(handles=scatter.legend_elements()[0],
           labels=sorted(colormap),
           title="-1为异常值")
```

Out[50]: <matplotlib.legend.Legend at 0x23bcd95b488>



In [51]:
```
# pip install tushare
import tushare as ts
```

In [52]:
```
df = ts.get_k_data('sh', start='2016-01-01')
```

本接口即将停止更新，请尽快使用Pro版接口：https://waditu.com/document/2

In [53]:
```
df.set_index('date').close.plot(figsize = [20,5],title = '收盘价时序趋势图')
```

Out[53]: <AxesSubplot:title={'center':'收盘价时序趋势图'}, xlabel='date'>