# 异常值检测与处理

# 实战练习工作册

## 基于UCI Heart Disease数据集

| 项目 | 内容 |
| --- | --- |
| 数据集 | UCI Heart Disease (Cleveland) — 303条记录，14个变量 |
| 目标 | 实践多种异常值检测方法并分析对ML模型性能的影响 |
| 难度 | ★★★☆☆ (中级) |
| 预计时间 | 约90~120分钟 |

| 步骤 | 内容 | 核心方法 |
| --- | --- | --- |
| STEP 1 | 数据加载与探索性数据分析 | 描述性统计、箱线图、IQR |
| STEP 2 | 四种高级异常值检测方法实现 | DBSCAN、Mahalanobis、OC-SVM、PCA |
| STEP 3 | 异常值处理前/后模型性能对比 | Random Forest、Logistic Regression |
| STEP 4 | 时间序列异常值检测 | Rolling Z-Score、IQR、EWMA |

# 目录

目录 ..... 2
学习目标 ..... 3
环境配置 ..... 4
STEP 1. 数据加载与探索性数据分析 (EDA) ..... 5
1-1. 数据加载 ..... 5
练习 1-1：数据基本探索 ..... 6
1-2. 数据预处理 ..... 7
练习 1-2：用箱线图可视化异常值 ..... 7
STEP 2. 四种高级异常值检测方法实现 ..... 9
2-0. 数据标准化 ..... 9
2-1. DBSCAN（基于密度的空间聚类） ..... 9
练习 2-1：DBSCAN参数实验 ..... 10
2-2. 马哈拉诺比斯距离 (Mahalanobis Distance) ..... 10
练习 2-2：马氏距离分布可视化 ..... 11
2-3. 单类支持向量机 (One-Class SVM) ..... 11
练习 2-3：nu参数实验 ..... 12
2-4. 基于PCA的异常值检测 ..... 12
练习 2-4：四种方法对比可视化 ..... 13
练习 2-5：方法间交集分析 ..... 13
STEP 3. 异常值处理前/后模型性能对比 ..... 15
3-1. 数据准备与评估函数 ..... 15
练习 3-1：全面实验 ..... 16
练习 3-2：结果可视化 ..... 16
练习 3-3：Winsorization（异常值替换）vs 删除对比 ..... 17
STEP 4. 时间序列数据中的异常值检测 ..... 18
4-0. 生成时间序列数据 ..... 18
练习 4-1：Rolling Z-Score ..... 19
练习 4-2：Rolling IQR ..... 19
练习 4-3：EWMA控制图 ..... 19
练习 4-4：检测性能定量评估 ..... 20
总结与进阶作业 ..... 21
核心要点 ..... 21
实践建议 ..... 21
进阶作业 ..... 21

# 学习目标

1. 理解异常值(outlier)的概念和类型（点异常/上下文异常/集体异常）
2. 能够解释4种高级异常值检测算法的原理
3. 通过实验验证异常值处理对机器学习模型性能的影响
4. 实现时间序列数据中的异常值检测方法

# 环境配置

请在Jupyter Notebook或Google Colab中运行以下代码，安装并导入所需库。

```
# 安装库（如需要请取消注释）
# !pip install pandas numpy scikit-learn matplotlib seaborn scipy -q


Import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.spatial.distance import mahalanobis

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.linear_model import LogisticRegression
from sklearn.svm import OneClassSVM
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import (precision_score, recall_score, f1_score)
from collections import Counter

import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (12, 6)
sns.set_style('whitegrid')
```

# STEP 1. 数据加载与探索性数据分析 (EDA)

**背景知识：什么是异常值(Outlier)？**

异常值(Outlier)是与其他观测值显著不同的数据点。可能是测量误差、数据输入错误，或确实是极端观测值。

**点异常 (Point Outlier)**：单个数据点异常

**上下文异常 (Contextual Outlier)**：仅在特定上下文中异常（例：夏天出现零下温度）

**集体异常 (Collective Outlier)**：单独正常但作为集合异常

## 1-1. 数据加载

从UCI机器学习资源库加载Heart Disease (Cleveland)数据集。

| 变量名 | 说明 | 类型 |
| --- | --- | --- |
| age | 年龄 | 连续 |
| sex | 性别 (1=男, 0=女) | 分类 |
| cp | 胸痛类型 (0~3) | 分类 |
| trestbps | 静息血压 (mm Hg) | 连续 |
| chol | 血清胆固醇 (mg/dl) | 连续 |
| fbs | 空腹血糖 > 120 mg/dl | 分类 |
| restecg | 静息心电图 (0~2) | 分类 |
| thalach | 最大心率 | 连续 |
| exang | 运动诱发心绞痛 | 分类 |
| oldpeak | ST段压低 | 连续 |
| slope | ST段斜率 | 分类 |
| ca | 主要血管数 (0~3) | 连续 |
| thal | 铊扫描结果 | 分类 |
| target | 心脏病诊断 | 目标 |

```
column_names = [
    'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
    'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'
]
```

```
url =os.path.join(os.path.dirname(os.path.dirname(__file__)),
'DataSet','processed.cleveland.data')

df = pd.read_csv(url, names=column_names, na_values='?')
print(f'Shape: {df.shape}')
df.head()
```

## 练习 1-1：数据基本探索

**练习 1-1：请根据TODO提示自行编写代码**

TODO 1：使用 df.info() 查看数据类型和缺失值

TODO 2：使用 df.describe() 查看描述性统计量

TODO 3：使用 df.isnull().sum() 查看各列缺失值数量

```
# TODO 1: 查看数据信息
# 在此处编写代码：



# TODO 2: 描述性统计量
# 在此处编写代码：



# TODO 3: 缺失值检查
# 在此处编写代码：
```

参考答案

```
# TODO 1
df.info()

# TODO 2
df.describe().round(2)

# TODO 3
print(df.isnull().sum())
print(f'\n缺失值总数: {df.isnull().sum().sum()}个')
```

## 1-2. 数据预处理

```
# 目标变量二值化: 0=正常, 1=心脏病
df['target'] = (df['target'] > 0).astype(int)
print(df['target'].value_counts())

# 删除缺失值
df = df.dropna()

# 定义连续型变量
continuous_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

### 练习 1-2：用箱线图可视化异常值

**IQR方法回顾**

**Q1** (第25百分位数), **Q3** (第75百分位数), **IQR = Q3 - Q1**

**异常值边界**：低于Q1 - 1.5×IQR 或 高于Q3 + 1.5×IQR

**练习 1-2：绘制5个连续变量的箱线图并计算IQR异常值数量**

TODO：使用 axes[i].boxplot(df[col].dropna(), ...) 绘制箱线图

TODO：用IQR方法计算异常值数量

```
fig, axes = plt.subplots(1, 5, figsize=(20, 5))
fig.suptitle('Box Plot - Continuous Features', fontsize=14)

for i, col in enumerate(continuous_cols):
    # TODO: 绘制箱线图
    # axes[i].boxplot(df[col].dropna(), ...)

    axes[i].set_title(col)

    # TODO: 计算IQR异常值
    # q1 = df[col].quantile(0.25)
    # q3 = df[col].quantile(0.75)
    # iqr = q3 - q1
    # n_outliers = ...

plt.tight_layout()
plt.show()
```

### 参考答案

```
for i, col in enumerate(continuous_cols):
    axes[i].boxplot(df[col].dropna(), vert=True, patch_artist=True,
                    boxprops=dict(facecolor='#4ECDC4', alpha=0.7))
    axes[i].set_title(col)
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    n_outliers = ((df[col]<q1-1.5*iqr) | (df[col]>q3+1.5*iqr)).sum()
    axes[i].set_xlabel(f'IQR Outliers: {n_outliers}', color='red')
```

### 检查点

1. 哪个变量的IQR异常值最多？
2. chol（胆固醇）变量的分布是什么形状？
3. 这些异常值是测量误差还是有意义的极端值？

# STEP 2. 四种高级异常值检测方法实现

| 方法 | 原理 | 优点 | 缺点 |
|---|---|---|---|
| DBSCAN | 基于密度的聚类 | 可检测非球形簇 | eps、min_samples敏感 |
| Mahalanobis | 考虑协方差的距离 | 反映变量间相关性 | 需要正态分布假设 |
| One-Class SVM | 非线性边界学习 | 高维数据有效 | 需设置nu参数 |
| PCA | 基于重建误差 | 降维+检测同时进行 | 仅捕捉线性关系 |

## 2-0. 数据标准化

⚠ **异常值检测前必须进行标准化(Standardization)！**变量间尺度差异会导致基于距离的算法产生偏差。

```
X_continuous = df[continuous_cols].copy()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_continuous)
X_scaled_df = pd.DataFrame(X_scaled, columns=continuous_cols, index=df.index)

outlier_results = {}  # 存储各方法结果
```

## 2-1. DBSCAN（基于密度的空间聚类）

📖 **DBSCAN算法原理**

① 对每个点计算eps半径内的邻居数

② 邻居数 ≥ min_samples → 核心点 (Core Point)

③ 核心点互连 → 形成簇

④ 不属于任何簇的点 → 噪声（异常值）

**核心参数：**

**eps**：邻居搜索半径。越小检测出的异常值越多

**min_samples**：成为核心点所需的最小邻居数

```
dbscan = DBSCAN(eps=1.8, min_samples=10)
db_labels = dbscan.fit_predict(X_scaled)
```

```
db_outliers = db_labels == -1       # 噪声 = -1
n_clusters = len(set(db_labels)) - (1 if -1 in db_labels else 0)
outlier_results['DBSCAN'] = df.index[db_outliers].tolist()

print(f'簇数: {n_clusters}')
print(f'异常值(噪声): {db_outliers.sum()}个 ({db_outliers.sum()/len(df)*100:.1f}%)')
```

### 练习 2-1：DBSCAN 参数实验

**调整eps和min_samples，观察异常值数量的变化**

运行以下代码并分析结果。

思考：eps设得太小会怎样？太大呢？

```
eps_values = [0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
min_samples_values = [5, 10, 15, 20]

for eps in eps_values:
    for ms in min_samples_values:
        db = DBSCAN(eps=eps, min_samples=ms)
        labels = db.fit_predict(X_scaled)
        n_c = len(set(labels)) - (1 if -1 in labels else 0)
        n_out = (labels == -1).sum()
        print(f'eps={eps:.1f}, min_samples={ms:2d} -> {n_c}个簇, {n_out}个异常值')
```

## 2-2. 马哈拉诺比斯距离 (Mahalanobis Distance)

**马哈拉诺比斯距离原理**

**欧氏距离**：$d = \sqrt{(x_1-\mu_1)^2 + (x_2-\mu_2)^2}$ → 忽略变量间相关性

**马氏距离**：$d = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$ → 用协方差矩阵($\Sigma$)修正相关性

**异常值判定**：$d^2$ 服从$\chi^2(p)$分布，p值 < 0.001则为异常值

```
mean = X_continuous.mean().values
cov_matrix = X_continuous.cov().values
cov_inv = np.linalg.inv(cov_matrix)  # 逆矩阵
```

```
mahal_distances = np.array([
    mahalanobis(X_continuous.iloc[i].values, mean, cov_inv)
    for i in range(len(X_continuous))
])

# χ²检验判定异常值 (p < 0.001)
p_values = 1 - stats.chi2.cdf(mahal_distances**2, df=len(continuous_cols))
mahal_outliers = p_values < 0.001
outlier_results['Mahalanobis'] = df.index[mahal_outliers].tolist()
```

### 练习 2-2：马氏距离分布可视化

**绘制距离的直方图并标注χ²临界值**

TODO：绘制mahal_distances的直方图 (bins=30)

TODO：用红色虚线标注χ²临界值

参考答案

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.hist(mahal_distances, bins=30, color='#3498db', edgecolor='white',
        alpha=0.7, density=True)
threshold = np.sqrt(stats.chi2.ppf(1-0.001, df=len(continuous_cols)))
ax.axvline(threshold, color='red', linestyle='--', linewidth=2,
           label=f'Threshold (p<0.001): {threshold:.2f}')
ax.legend(); plt.show()
```

## 2-3. 单类支持向量机 (One-Class SVM)

**One-Class SVM原理**

① 仅用正常数据训练（无监督学习）

② 用RBF核映射到高维空间

③ 学习一个从原点最大化分离数据的超平面

④ 超平面外侧（负值）的点 → 异常值 (-1)

**nu (0~1)**：异常值比例上限。nu=0.1 → 约10%以下判为异常

**gamma**：RBF核的范围参数

```
ocsvm = OneClassSVM(kernel='rbf', nu=0.1, gamma='auto')
svm_labels = ocsvm.fit_predict(X_scaled)

svm_outliers = svm_labels == -1
outlier_results['OneClass_SVM'] = df.index[svm_outliers].tolist()

print(f'正常(+1): {(svm_labels==1).sum()}个')
print(f'异常值(-1): {svm_outliers.sum()}个 ({svm_outliers.sum()/len(df)*100:.1f}%)')
```

### 练习 2-3：nu 参数实验

将**nu从0.01调整到0.3，观察异常值比例变化并绘制折线图**

参考答案

```
nu_values = [0.01, 0.05, 0.1, 0.15, 0.2, 0.3]
outlier_counts = []
for nu in nu_values:
    model = OneClassSVM(kernel='rbf', nu=nu, gamma='auto')
    labels = model.fit_predict(X_scaled)
    outlier_counts.append((labels == -1).sum())

plt.plot(nu_values, outlier_counts, 'bo-', linewidth=2)
plt.xlabel('nu'); plt.ylabel('Outliers')
plt.title('One-Class SVM: Effect of nu'); plt.show()
```

## 2-4. 基于 PCA 的异常值检测

**PCA重建误差检测原理**

① 用PCA提取k个主成分 (k < 原始维度)

② 压缩到主成分空间: X → X_pca

③ 重建回原始空间: X_pca → X_reconstructed

④ 计算重建误差: ||X - X_reconstructed||²

**核心思想**：正常数据 → 重建误差小 / 异常值 → 重建误差大

**临界值**：超过95th百分位数 → 判为异常值

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_reconstructed = pca.inverse_transform(X_pca)

reconstruction_error = np.sum((X_scaled - X_reconstructed)**2, axis=1)
threshold_pca = np.percentile(reconstruction_error, 95)
pca_outliers = reconstruction_error > threshold_pca
outlier_results['PCA'] = df.index[pca_outliers].tolist()
```

## 练习 2-4：四种方法对比可视化

**在PCA 2D散点图上标记各方法检测出的异常值**

正常数据：绿色(#2ecc71)小圆点

异常值：红色(#e74c3c) X标记

参考答案

```
fig, axes = plt.subplots(2, 2, figsize=(14, 12))
methods = {'DBSCAN': db_outliers, 'Mahalanobis': mahal_outliers,
           'One-Class SVM': svm_outliers, 'PCA': pca_outliers}

for ax, (name, mask) in zip(axes.flat, methods.items()):
    normal = ~mask
    ax.scatter(X_pca[normal,0], X_pca[normal,1],
               c='#2ecc71', alpha=0.5, s=20, label=f'Normal ({normal.sum()})')
    ax.scatter(X_pca[mask,0], X_pca[mask,1],
               c='#e74c3c', alpha=0.8, s=40, marker='x',
               label=f'Outlier ({mask.sum()})')
    ax.set_title(name); ax.legend(); ax.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
```

## 练习 2-5：方法间交集分析

```
all_sets = {k: set(v) for k, v in outlier_results.items()}
outlier_counter = Counter()
for name, idx_set in all_sets.items():
    for idx in idx_set:
        outlier_counter[idx] += 1

print(f'2种以上方法检测为异常: {sum(1 for v in outlier_counter.values() if v>=2)}个')
print(f'3种以上方法检测为异常: {sum(1 for v in outlier_counter.values() if v>=3)}个')
```

**检查点**

1. 哪种方法检测出的异常值最多/最少？
2. 为什么各方法检测到的异常值特征不同？
3. 实际工作中应该选择哪种方法？

**核心观点：**不要依赖单一方法，综合多种方法结果的集成方法最为稳定！

# STEP 3. 异常值处理前/后模型性能对比

> **核心问题：删除异常值后模型性能一定会提升吗？**
>
> 本步骤将通过实验回答这个问题。

| 项目 | 内容 |
| --- | --- |
| 模型 | Random Forest, Logistic Regression |
| 评估 | 5折交叉验证 |
| 指标 | Accuracy, F1-Score, Precision, Recall |
| 对比条件 | 原始 vs DBSCAN/Mahalanobis/OC-SVM/PCA/IsolationForest删除 |

## 3-1. 数据准备与评估函数

```
feature_cols = [c for c in df.columns if c != 'target']
X_full = df[feature_cols].copy()
y_full = df['target'].copy()
for col in X_full.columns:
    X_full[col] = pd.to_numeric(X_full[col], errors='coerce')
X_full = X_full.fillna(X_full.median())

# 添加Isolation Forest
iso = IsolationForest(contamination=0.1, random_state=42)
iso_labels = iso.fit_predict(StandardScaler().fit_transform(X_full))
outlier_results['Isolation_Forest'] = df.index[iso_labels==-1].tolist()
```

```
def evaluate_model(X, y, model):
    if len(X) < 10:
        return {'accuracy': np.nan, 'f1': np.nan}
    X_s = StandardScaler().fit_transform(X)
    return {
        'accuracy': cross_val_score(model, X_s, y, cv=5,
                                    scoring='accuracy').mean(),
        'f1': cross_val_score(model, X_s, y, cv=5,
                              scoring='f1_weighted').mean(),
    }
```

## 练习 3-1：全面实验

对每种异常值删除方法训练模型并比较性能

```
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
}

datasets = {'(A) 原始数据': df.index.tolist()}
for method, outlier_idx in outlier_results.items():
    clean_idx = [i for i in df.index if i not in outlier_idx]
    datasets[f'(删除) {method}'] = clean_idx

results_table = []
for ds_name, idx_list in datasets.items():
    X_sub = X_full.loc[idx_list]
    y_sub = y_full.loc[idx_list]
    for model_name, model in models.items():
        scores = evaluate_model(X_sub, y_sub, model)
        results_table.append({'Dataset': ds_name, 'Model': model_name,
                              'N': len(idx_list), **scores})

results_df = pd.DataFrame(results_table)
results_df.round(4)
```

## 练习 3-2：结果可视化

用条形图比较各模型的F1-Score

参考答案

```
fig, ax = plt.subplots(figsize=(12, 7))
pivot = results_df.pivot(index='Dataset', columns='Model', values='f1')
pivot.plot(kind='barh', ax=ax, color=['#3498db','#e67e22'])
ax.set_title('F1 Score Comparison'); ax.set_xlim(0.5, 1.0)
for c in ax.containers:
    ax.bar_label(c, fmt='%.3f', fontsize=8, padding=3)
plt.tight_layout(); plt.show()
```

## 练习 3-3：Winsorization（异常值替换）vs 删除对比

**什么是Winsorization？**

不删除异常值，而是将其替换为上下界边界值(clipping)的方法。无数据损失，可减小异常值影响。

```
X_winsor = X_full.copy()
for col in continuous_cols:
    if col in X_winsor.columns:
        lower = X_winsor[col].quantile(0.05)
        upper = X_winsor[col].quantile(0.95)
        X_winsor[col] = X_winsor[col].clip(lower, upper)

for model_name, model in models.items():
    scores = evaluate_model(X_winsor, y_full, model)
    print(f'{model_name}: Accuracy={scores["accuracy"]:.4f}, F1={scores["f1"]:.4f}')
```

**检查点**

1. 删除异常值是否总是提升了性能？
2. 哪种异常值删除方法最有效？
3. Winsorization(替换) vs 删除 — 哪种更稳定？

**核心观点：**删除异常值不一定提升性能。小数据集中替换可能更安全。

# STEP 4. 时间序列数据中的异常值检测

**时间序列异常值的类型**

**(1) 点异常**：突然的尖峰

**(2) 水平转移**：均值突然变化

**(3) 渐进漂移**：缓慢增加/减少

**场景：心脏病患者每日心率(thalach) 365天监测模拟**

## 4-0. 生成时间序列数据

```
np.random.seed(42)
n_days = 365
base_hr = df['thalach'].mean()

t = np.arange(n_days)
seasonal = 5 * np.sin(2 * np.pi * t / 365)   # 季节性波动
trend = -0.01 * t                             # 微弱下降趋势
noise = np.random.normal(0, 8, n_days)          # 随机噪声
heart_rate = base_hr + seasonal + trend + noise

# 注入异常值
anomaly_indices = []

# (1) 点异常: 8个突然尖峰
point_anomalies = np.random.choice(range(30, 335), 8, replace=False)
heart_rate[point_anomalies] += np.random.choice([-1,1],8) * np.random.uniform(35,55,8)
anomaly_indices.extend(point_anomalies)

# (2) 水平转移: 2周异常上升 (Day 180~194)
heart_rate[180:195] += 25
anomaly_indices.extend(range(180, 195))

# (3) 渐进漂移 (Day 300~319)
heart_rate[300:320] += np.linspace(0, 40, 20)
anomaly_indices.extend(range(300, 320))
anomaly_indices = sorted(set(anomaly_indices))

dates = pd.date_range('2024-01-01', periods=n_days, freq='D')
ts = pd.Series(heart_rate, index=dates)
```

## 练习 4-1：Rolling Z-Score

**Rolling Z-Score原理**

在滑动窗口内计算均值和标准差，得到Z分数。|Z| > threshold的点判为异常值。

**用window=21, threshold=2.5实现Rolling Z-Score检测**

```
window = 21
threshold = 2.5

# TODO: 计算滑动均值、滑动标准差、Z分数
# rolling_mean = ts.rolling(window=window, center=True).mean()
# rolling_std = ts.rolling(window=window, center=True).std()
# z_scores = (ts - rolling_mean) / rolling_std
# z_outliers = np.abs(z_scores) > threshold
```

## 练习 4-2：Rolling IQR

```
rolling_q1 = ts.rolling(window=window, center=True).quantile(0.25)
rolling_q3 = ts.rolling(window=window, center=True).quantile(0.75)
rolling_iqr = rolling_q3 - rolling_q1
iqr_lower = rolling_q1 - 1.5 * rolling_iqr
iqr_upper = rolling_q3 + 1.5 * rolling_iqr
iqr_outliers = (ts < iqr_lower) | (ts > iqr_upper)
```

## 练习 4-3：EWMA 控制图

**EWMA原理**

EWMA（指数加权移动平均）对近期数据赋予更大权重。超出控制界限(±2.5σ)则判为异常值。

```
ewma = ts.ewm(span=21).mean()
ewma_std = ts.ewm(span=21).std()
ewma_upper = ewma + 2.5 * ewma_std
ewma_lower = ewma - 2.5 * ewma_std
ewma_outliers = (ts > ewma_upper) | (ts < ewma_lower)
```

## 练习 4-4：检测性能定量评估

**与Ground Truth对比，计算Precision, Recall, F1-Score**

```
true_labels = np.zeros(n_days)
true_labels[anomaly_indices] = 1

ts_methods = {
    'Rolling Z-Score': z_outliers.values.astype(int),
    'Rolling IQR':     iqr_outliers.values.astype(int),
    'EWMA Control':    ewma_outliers.values.astype(int)
}

for name, pred in ts_methods.items():
    valid = ~np.isnan(pred)
    p = precision_score(true_labels[valid], pred[valid], zero_division=0)
    r = recall_score(true_labels[valid], pred[valid], zero_division=0)
    f1 = f1_score(true_labels[valid], pred[valid], zero_division=0)
    print(f'{name}: Precision={p:.3f}, Recall={r:.3f}, F1={f1:.3f}')
```

**检查点**

1. 哪种方法最擅长检测点异常？
2. 为什么水平转移和渐进漂移难以检测？
3. 如何提高Recall？需要什么调整？

# 总结与进阶作业

## 核心要点

| 步骤 | 核心见解 |
|---|---|
| STEP 1 | 用单变量IQR快速识别异常值 |
| STEP 2 | 不同方法检测到的异常值特征不同 |
| STEP 3 | 删除异常值 ≠ 性能总是提升 |
| STEP 4 | 时序异常需根据类型选择合适方法 |

## 实践建议

**1.** 不要依赖单一方法 → 使用集成方法

**2.** 结合领域知识 → 医疗数据的异常值可能是重要信号

**3.** 通过实验决定删除 vs 替换 vs 保留

**4.** 始终用交叉验证验证稳定性

## 进阶作业

| 作业 | 标题 | 说明 |
|---|---|---|
| 作业 1 | Isolation Forest参数调优 | 将contamination从0.01调至0.3，寻找最优值 |
| 作业 2 | 集成异常值检测器 | 实现4种方法的投票(voting)融合检测器 |
| 作业 3 | 应用于其他数据集 | 将相同流程应用于Wine Quality或Breast Cancer数据集 |
| 作业 4 | LSTM自编码器 | 用PyTorch实现基于LSTM AutoEncoder的时序异常检测 |

🎉 辛苦了！