数据预处理：缺失值处理

Python 实践练习 (shíjiàn liànxí)

Missing Value Handling — Hands-on Practice

授课教师：李阳源

工具：Python 3 + pandas + numpy + matplotlib + sklearn

实践概述 (shíjiàn gàishù)

同学们，今天我们要把刚才学的缺失值理论用Python代码来实践。

我们会从创建数据开始，一步一步体验缺失值的检测、分析和处理过程。

请大家打开Jupyter Notebook或任何Python编辑器，跟着我一起做。

环境准备 (huánjìng zhǔnbèi) — 安装必要的库

首先，确认你的电脑上已经安装了这些库。如果没有，请运行以下命令：

```
pip install pandas numpy matplotlib scikit-learn missingno
```

然后在Python中导入（dǎorù）所有需要的库：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import missingno as msno
from sklearn.impute import SimpleImputer, KNNImputer
import warnings
warnings.filterwarnings('ignore')
```

> 💡 *如果安装时遇到问题，可以尝试：pip install --user 库名*

第一部分：创建模拟数据 (chàngjiàn mónǐ shùjù)

我们先创建一个模拟的调查问卷数据集，和今天讲课的内容一样。

这个数据集包含：年龄、性别、收入、睡眠时间、压力等变量。

Step 1.1: 创建基础数据

我们用numpy随机生成一些数据，然后人为地加入三种缺失机制的缺失值。

```
np.random.seed(42)
n = 200


# 创建基础数据
data = {
    'age': np.random.randint(18, 70, n),
    'gender': np.random.choice(['male', 'female'], n),
    'income': np.random.normal(8000, 3000, n).astype(int),
    'sleep_hours': np.round(np.random.normal(6.5, 1.5, n), 1),
    'stress': np.random.choice(['yes', 'no', 'unsure'], n,
             p=[0.4, 0.35, 0.25]),
    'score': np.random.normal(75, 15, n).astype(int)
}
df = pd.DataFrame(data)
```

这里我们生成了200条记录，包含6个变量。

现在这个数据集是完整的，没有缺失值。

下一步我们要人为制造三种不同类型的缺失值。

Step 1.2: 制造MCAR缺失值（完全随机缺失）

还记得吗？MCAR的缺失跟任何变量都没有关系，完全随机发生。

我们随机选择10%的行，把睡眠时间设为缺失。

```
# MCAR: 随机选择10%的行，把sleep_hours设为NaN
mcar_idx = np.random.choice(n, size=int(n*0.1), replace=False)
df.loc[mcar_idx, 'sleep_hours'] = np.nan
print(f'MCAR缺失数量: {df["sleep_hours"].isna().sum()}')
```

```
>>> MCAR缺失数量: 20
```

> 💡 *注意：这里的缺失是完全随机的，跟年龄、性别、收入等都没有关系。*

Step 1.3: 制造MAR缺失值（随机缺失）

MAR的缺失跟其他可观察变量有关。

我们让年龄大于50岁的人更容易不填写收入（跟讲课内容一致）。

```
# MAR: 年龄>50的人，40%的概率不填写收入
old_mask = df['age'] > 50
old_idx = df[old_mask].index
mar_idx = np.random.choice(old_idx,
          size=int(len(old_idx)*0.4), replace=False)
df.loc[mar_idx, 'income'] = np.nan
```

```
# 年轻人也有少量缺失（5%）
young_mask = df['age'] <= 50
young_idx = df[young_mask].index
mar_idx2 = np.random.choice(young_idx,
           size=int(len(young_idx)*0.05), replace=False)
df.loc[mar_idx2, 'income'] = np.nan
print(f'MAR缺失数量: {df["income"].isna().sum()}')
```

> 💡 *观察：老年人缺失比例明显更高，这就是MAR的特点——缺失跟“年龄”这个可观察变量有关。*

Step 1.4: 制造MNAR缺失值（非随机缺失）

MNAR的缺失跟缺失值本身直接相关。

我们让成绩低的学生更容易拒绝公开成绩（跟讲课内容一致）。

```
# MNAR: 成绩低于60分的学生，60%拒绝公开成绩
low_score = df['score'] < 60
low_idx = df[low_score].index
mnar_idx = np.random.choice(low_idx,
           size=int(len(low_idx)*0.6), replace=False)
df.loc[mnar_idx, 'score'] = np.nan
```

```
# 成绩高的也有很少缺失（3%）
high_score = df['score'] >= 60
high_idx = df[high_score].index
```

```
mnar_idx2 = np.random.choice(high_idx,
            size=int(len(high_idx)*0.03), replace=False)
df.loc[mnar_idx2, 'score'] = np.nan
print(f'MNAR缺失数量: {df["score"].isna().sum()}')
```

> 💡 *想一想：如果我们用均值填充这些缺失的成绩，结果会怎样？平均成绩会被高估！因为缺失的大多是低分。*

第二部分：缺失值检测 (quēshī zhì jiǎn cè)

数据准备好了，现在我们来检测缺失值。

Step 2.1: 基本缺失值统计

用isnull()和sum()来查看每个变量有多少缺失值：

```
# 查看每个变量的缺失数量
print('=== 缺失值数量 (quēshī zhì shùliàng) ===')
print(df.isnull().sum())
print()
```

```
# 查看缺失率 (quēshīlǜ)
print('=== 缺失率 (%) ===')
missing_pct = (df.isnull().sum() / len(df) * 100).round(1)
print(missing_pct)
```

预期输出（yùqī shūchū）：

```
>>> === 缺失值数量 ===
    age             0
    gender          0
    income         29
    sleep_hours    20
    stress          0
    score          20
```

可以看到：income、sleep_hours、score三个变量有缺失值。

其他变量（age、gender、stress）是完整的。

Step 2.2: 可视化缺失模式 (kěshìhuà quēshī móshì)

用missingno库来可视化，让缺失模式更直观（zhíguān）：

```
# 矩阵图 (jǔzhèn tú) - 白色表示缺失
plt.figure(figsize=(10, 6))
msno.matrix(df)
plt.title('缺失值矩阵图 (Missing Value Matrix)', fontsize=14)
plt.tight_layout()
plt.show()
```

图中白色的线条表示缺失值的位置，黑色表示有值。

可以直观地看到哪些变量、哪些行有缺失。

```
# 柱状图 (zhùzhuàng tú) - 每个变量的完整数量
plt.figure(figsize=(10, 4))
msno.bar(df, color='steelblue')
plt.title('各变量数据完整性 (Data Completeness)', fontsize=14)
plt.tight_layout()
plt.show()
```

> 💡 *试试看：msno.heatmap(df)可以看变量之间缺失的相关性！*

Step 2.3: 分组分析MAR特征

让我们验证一下income的缺失是否跟年龄有关（MAR特征）：

```
# 按年龄分组查看收入缺失率
df['age_group'] = pd.cut(df['age'],
                         bins=[17, 30, 40, 50, 70],
                         labels=['18-30', '31-40', '41-50', '51-70'])

missing_by_age = df.groupby('age_group')['income'].apply(
    lambda x: round(x.isna().mean() * 100, 1)
)
print('各年龄段收入缺失率(%):')
print(missing_by_age)
```

你应该能看到：51-70岁的缺失率明显高于其他年龄段。

这就证实了我们设计的MAR机制！

第三部分：删除方法 (shānchú fāngfǎ)

Step 3.1: 列表删除法 (Listwise Deletion)

还记得吗？列表删除法就是把有缺失值的整行删掉。

```
# Listwise Deletion: 删除所有包含缺失值的行
df_listwise = df.dropna()


print(f'原始数据行数: {len(df)}')
print(f'删除后行数: {len(df_listwise)}')
print(f'损失数据: {len(df) - len(df_listwise)} 行')
print(f'损失比例: {(1 - len(df_listwise)/len(df))*100:.1f}%')
```

```
>>> 原始数据行数: 200
    删除后行数: ~140
    损失数据: ~60 行
    损失比例: ~30.0%
```

看到了吗？我们删掉了大约30%的数据！这就是列表删除法的最大缺点。

Step 3.2: 成对删除法 (Pairwise Deletion)

成对删除法在pandas里体现为计算时只排除相关缺失值：

```
# Pairwise: 计算相关系数时自动使用成对删除
corr_pairwise = df[['age','income','sleep_hours','score']].corr()
print('相关系数矩阵 (Pairwise):')
print(corr_pairwise.round(3))
```

```
# 对比: Listwise方式的相关系数
corr_listwise = df[['age','income','sleep_hours','score']].dropna().corr()
print('\n相关系数矩阵 (Listwise):')
print(corr_listwise.round(3))
```

对比两个矩阵，你会发现结果可能不同。

Pairwise保留了更多数据，但每对变量用的样本量不同。

第四部分：简单替换方法 (jiǎndān tìhuàn)

Step 4.1: 均值替换 (jūnzhí tìhuàn)

用均值来填充sleep_hours的缺失值：

```
df_mean = df.copy()


# 方法一：直接用fillna
mean_val = df_mean['sleep_hours'].mean()
print(f'sleep_hours的均值: {mean_val:.2f}')
df_mean['sleep_hours'].fillna(mean_val, inplace=True)
```

```
# 方法二：用sklearn的SimpleImputer
imputer_mean = SimpleImputer(strategy='mean')
df_mean[['income']] = imputer_mean.fit_transform(df_mean[['income']])
```

```
print('均值替换后缺失值检查:')
print(df_mean[['sleep_hours','income']].isnull().sum())
```

现在sleep_hours和income的缺失值都被填充了。

Step 4.2: 中位数替换 (zhōngwèishù tìhuàn)

中位数对异常值（yìchángzhí）更稳健（wěnjiàn）：

```
df_median = df.copy()


imputer_median = SimpleImputer(strategy='median')
df_median[['income']] = imputer_median.fit_transform(
    df_median[['income']])


print(f'均值填充的income均值: {df_mean["income"].mean():.0f}')
print(f'中位数填充的income均值: {df_median["income"].mean():.0f}')
```

> 💡 *比较两种方法的结果，均值和中位数填充后的分布可能会有差异！*

Step 4.3: 查看均值替换对分布的影响

这是均值替换最大的缺点——会改变数据分布：

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# 原始数据分布
axes[0].hist(df['sleep_hours'].dropna(), bins=20,
             alpha=0.7, color='steelblue', label='原始数据')
axes[0].set_title('原始 sleep_hours 分布')
axes[0].legend()

# 均值填充后的分布
axes[1].hist(df_mean['sleep_hours'], bins=20,
             alpha=0.7, color='coral', label='均值填充后')
axes[1].axvline(mean_val, color='red', linestyle='--',
               label=f'均值={mean_val:.1f}')
axes[1].set_title('均值填充后 sleep_hours 分布')
axes[1].legend()

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

右图中均值处会出现一个明显的“尖峰（jiānfēng）”，这就是分布被扭曲的证据！

第五部分：高级替换方法 (gāojí tìhuàn)

Step 5.1: KNN替换法

KNN方法用最相似的K个样本的值来填充缺失值。

这比均值替换更智能，因为它考虑了数据之间的相似性。

```
# 准备数值型变量
df_knn = df[['age', 'income', 'sleep_hours', 'score']].copy()


# KNN替换 (K=5)
knn_imputer = KNNImputer(n_neighbors=5)
df_knn_filled = pd.DataFrame(
    knn_imputer.fit_transform(df_knn),
    columns=df_knn.columns
)


print('KNN替换后缺失值检查:')
print(df_knn_filled.isnull().sum())
```

所有缺失值都被填充了！而且每个填充值都是根据最相似的5个样本计算出来的。

> 💡 *试试修改n_neighbors的值（比如K=3或K=10），看看结果有什么变化！*

Step 5.2: 对比不同方法的效果

让我们把不同方法的填充结果放在一起比较：

```
# 比较income变量的不同填充方法
fig, axes = plt.subplots(1, 3, figsize=(18, 5))


axes[0].hist(df['income'].dropna(), bins=25, alpha=0.7,
             color='steelblue')
axes[0].set_title('原始数据 (Original)')
axes[0].set_xlabel('income')


axes[1].hist(df_mean['income'], bins=25, alpha=0.7,
             color='coral')
axes[1].set_title('均值填充 (Mean Imputation)')
axes[1].set_xlabel('income')
```

```
axes[2].hist(df_knn_filled['income'], bins=25, alpha=0.7,
             color='mediumseagreen')
axes[2].set_title('KNN填充 (KNN Imputation)')
axes[2].set_xlabel('income')


plt.suptitle('三种方法比较: income变量分布', fontsize=14)
plt.tight_layout()
plt.show()
```

观察三张图：KNN填充的分布通常比均值填充更接近原始分布。

第六部分：综合统计对比 (zōnghé tǒngjì duìbǐ)

最后，我们用一个表格来总结各种方法的效果。

Step 6.1: 统计指标对比表

```
# 创建对比表格
comparison = pd.DataFrame({
    '原始数据': df['income'].describe(),
    '均值填充': df_mean['income'].describe(),
    '中位数填充': df_median['income'].describe(),
    'KNN填充': df_knn_filled['income'].describe()
}).round(1)

print('=== income变量统计对比 ===')
print(comparison)
```

仔细观察（zǐxì guānchá）每种方法的统计指标：

—— 均值填充会让标准差（biāozhǔnchā）变小（因为加了很多相同的值）。

—— KNN填充的标准差通常更接近原始数据。

Step 6.2: MNAR偏差验证

让我们特别看看MNAR的score变量——这是最危险的情况：

```
# 计算真实均值 vs 填充后均值
# 注意：我们知道原始完整数据的均值约为75
print(f'原始完整数据的score均值：~75')
print(f'缺失后观测到的score均值：{df["score"].mean():.1f}')
print(f'均值填充后的score均值：{df_mean["score"].mean():.1f}' \
      if 'score' in df_mean else '')
```

你会发现：

因为低分学生拒绝公开成绩，观测到的均值比真实均值高。

用均值填充后，偏差会进一步加剧（jiājù）！

这就是MNAR最危险的地方——简单的替换方法会导致系统性偏差！

第七部分：实践总结 (shíjiàn zǒngjié)

同学们，今天的实践练习到这里就结束了！

我们来总结一下今天学到了什么：

- ☑ 学会了如何用Python创建包含三种缺失机制的数据集
- ☑ 掌握了isnull()、missingno等缺失值检测工具
- ☑ 体验了列表删除法和成对删除法的区别
- ☑ 实践了均值、中位数、KNN等替换方法
- ☑ 理解了MNAR情况下简单替换的危险性
- ☑ 学会了用可视化和统计指标比较不同方法的效果

课后练习（kèhòu liànxí）— Homework

请完成以下练习：

1. 尝试用众数（mode）填充stress变量（如果它有缺失值的话）。

   提示：SimpleImputer(strategy='most_frequent')

2. 修改KNN的K值（K=3, 5, 10, 20），观察填充结果的变化。

   思考：K值太小或太大会有什么问题？

3. 用以下代码尝试多重替换（Multiple Imputation）：

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer


iter_imputer = IterativeImputer(max_iter=10, random_state=42)
df_iter = pd.DataFrame(
    iter_imputer.fit_transform(df_knn),
    columns=df_knn.columns
)
print(df_iter.describe().round(1))
```

4. 比较所有方法的结果，写一份简短的分析报告（不超过500字）。

加油！下节课我们继续学习异常值处理的实践！