数据预处理

编码（Encoding）实践工作册

编程语言：Python ｜ 库：scikit-learn / pandas / category_encoders

适用对象：数据预处理课程学生

版本：1.0 ｜ 2026年3月

第一章 工作册使用说明

1.1 学习目标

本工作册旨在帮助学生在学习了编码理论知识后，通过动手实践来巩固和深化理解。完成本工作册后，你将能够：

- 理解并实现 Label Encoding、One-Hot Encoding、Ordinal Encoding 等常见编码方法
- 掌握 Target Encoding、Binary Encoding、Frequency Encoding 等进阶编码技术
- 能够根据数据特征选择合适的编码方法
- 使用 scikit-learn、pandas 和 category_encoders 库完成实际编码任务

1.2 环境准备

请确保已安装以下 Python 库：

```
pip install pandas numpy scikit-learn category_encoders


# 验证安装
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, OrdinalEncoder
import category_encoders as ce
print('环境准备完成！')
```

1.3 示例数据集

本工作册使用以下示例数据集，请先运行以下代码创建数据：

```
import pandas as pd
import numpy as np


data = pd.DataFrame({
    'city': ['北京', '上海', '广州', '北京', '深圳',
             '上海', '广州', '北京', '深圳', '上海'],
```

```
    'education': ['博士', '硕士', '学士', '硕士', '博士',
                  '学士', '高中', '博士', '硕士', '学士'],
    'color': ['红', '蓝', '绿', '红', '蓝',
              '绿', '红', '蓝', '绿', '红'],
    'salary': [25000, 20000, 15000, 22000, 28000,
               12000, 8000, 26000, 21000, 13000]
})
print(data)
```

数据集包含以下字段：

字段名	类型	示例值	编码类型
city	名义型	北京/上海/广州/深圳	无序
education	有序型	高中/学士/硕士/博士	有序
color	名义型	红/蓝/绿	无序
salary	数值型	25000	目标变量

第二章 标签编码（Label Encoding）

2.1 理论回顾

标签编码将每个类别值映射为一个整数。例如：北京→0、上海→1、广州→2、深圳→3。

适用场景：有序变量或树模型中的无序变量

注意事项：会引入数值大小关系，不适合线性模型中的无序变量

2.2 示例代码

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data['city_label'] = le.fit_transform(data['city'])

print('编码结果：')
print(data[['city', 'city_label']])
print()
print('类别映射关系：')
for cls, idx in zip(le.classes_, range(len(le.classes_))):
    print(f'  {cls} -> {idx}')
```

2.3 动手练习

练习 2-1：对 color 列进行标签编码

要求：使用 LabelEncoder 对 color 列进行编码，并将结果存储在新列 color_label 中

提示：参考上方示例代码的结构

☑ **在下方编写你的代码：**

练习 2-2：反向解码

要求：将编码后的数字还原为原始类别名称

提示：使用 le.inverse_transform() 方法

在下方编写你的代码：

提示： 标签编码的顺序是按字母/拼音排序的，不是按出现顺序。可以通过 le.classes_ 查看具体映射关系。

第三章 独热编码（One-Hot Encoding）

3.1 理论回顾

独热编码为每个类别创建一个二进制列。某个样本属于该类别则为1，否则为0。这种方法不会引入虚假的大小关系。

适用场景：无序分类变量，类别数量较少（一般 < 10）

注意事项：类别过多时会导致维度爆炸，可配合 drop='first' 使用

3.2 方法一：pandas get_dummies

```
# 方法一：使用 pandas
city_dummies = pd.get_dummies(data['city'], prefix='city')
print(city_dummies)


# 合并到原数据
data_encoded = pd.concat([data, city_dummies], axis=1)
print(data_encoded.head())
```

3.3 方法二：scikit-learn OneHotEncoder

```
from sklearn.preprocessing import OneHotEncoder


ohe = OneHotEncoder(sparse_output=False, drop=None)
city_ohe = ohe.fit_transform(data[['city']])


# 查看特征名
print('特征名:', ohe.get_feature_names_out())
print(city_ohe[:5])
```

3.4 动手练习

练习 3-1：对 color 列进行独热编码

要求：分别使用 pd.get_dummies 和 OneHotEncoder 两种方法

对比两种方法的输出结果有何不同

☑ **在下方编写你的代码：**

练习 3-2：使用 drop='first' 避免多重共线性

要求：在 OneHotEncoder 中设置 drop='first'，观察输出变化

思考：为什么删除一列不会丢失信息？

☑ **在下方编写你的代码：**

> **提示：**当类别数量超过 10 个时，应考虑使用其他编码方法（如 Target Encoding）来避免维度爆炸问题。

第四章 序数编码（Ordinal Encoding）

4.1 理论回顾

序数编码用于有自然顺序的分类变量。与标签编码不同，序数编码允许我们自定义编码顺序，从而正确反映变量的序关系。

适用场景：学历、评级、年龄段等具有明确顺序的变量

优势：保留了变量的序关系，不增加维度

4.2 示例代码

```
from sklearn.preprocessing import OrdinalEncoder


# 定义顺序：高中 < 学士 < 硕士 < 博士
edu_order = [['高中', '学士', '硕士', '博士']]
oe = OrdinalEncoder(categories=edu_order)
data['education_ordinal'] = oe.fit_transform(data[['education']])


print(data[['education', 'education_ordinal']])
print()
print('编码映射：')
for cat, val in zip(edu_order[0], range(len(edu_order[0]))):
    print(f'  {cat} -> {val}')
```

4.3 动手练习

练习 4-1：对客户满意度进行序数编码

要求：创建包含客户满意度的数据（'非常不满意' < '不满意' < '一般' < '满意' < '非常满意'）

使用 OrdinalEncoder 进行编码，并验证顺序是否正确

☑ **在下方编写你的代码：**

练习 4-2：思考题

问题：如果将 city 列使用 OrdinalEncoder 编码会有什么问题？

请在下方写出你的分析（可以用中文回答）

☑ **在下方编写你的代码：**

第五章 目标编码（Target Encoding）

5.1 理论回顾

目标编码用目标变量的统计量（如均值）来替换类别变量。例如，用每个城市的平均工资来替代城市名称。

适用场景：高基数分类变量（类别数量多）

注意事项：容易导致目标泄露，建议使用交叉验证方式计算

5.2 示例代码

方法一：手动实现

```
# 手动计算每个城市的平均工资
city_mean = data.groupby('city')['salary'].mean()
data['city_target'] = data['city'].map(city_mean)


print('目标编码结果：')
print(data[['city', 'salary', 'city_target']])
print()
print('城市平均工资：')
print(city_mean)
```

方法二：category_encoders 库

```
import category_encoders as ce


te = ce.TargetEncoder(cols=['city'])
data['city_target_ce'] = te.fit_transform(
    data['city'], data['salary']
)
print(data[['city', 'city_target', 'city_target_ce']])
```

5.3 动手练习

练习 5-1：对 education 列进行目标编码

要求：分别用手动方式和 category_encoders 实现

对比两种方法的结果是否一致，思考为什么

☑ **在下方编写你的代码：**

练习 5-2：目标泄露实验

要求：将数据分为训练集和测试集，分别在全量数据和仅训练集上计算目标编码

观察并分析两种方式的差异

☑ **在下方编写你的代码：**

```
```

> **提示：** category_encoders 的 TargetEncoder 默认使用贝叶斯平滑来减少目标泄露，因此结果可能与手动计算的均值略有不同。

第六章 二进制编码（Binary Encoding）

6.1 理论回顾

二进制编码先将类别转换为整数，再将整数转换为二进制表示。它是 One-Hot Encoding 和 Label Encoding 的折中方案，编码后的维度为 log2(n)。

适用场景：中等基数的分类变量（10-100个类别）

优势：维度远小于 One-Hot，同时不引入虚假序关系

6.2 示例代码

```
import category_encoders as ce


be = ce.BinaryEncoder(cols=['city'])
data_binary = be.fit_transform(data[['city']])


print('二进制编码结果：')
print(data_binary)
print()
print(f'原始类别数: {data["city"].nunique()}')
print(f'编码后维度: {data_binary.shape[1]}')
```

6.3 动手练习

练习 6-1：手动实现二进制编码

要求：不使用 category_encoders，用纯 Python/pandas 实现二进制编码

提示：先用 LabelEncoder 获取整数，再将整数转为二进制位

☑ **在下方编写你的代码：**

练习 6-2：维度对比

要求：分别对包含20个类别的变量进行 One-Hot 和 Binary 编码

对比两种方法的输出维度，计算维度减少的比例

☑ **在下方编写你的代码：**

第七章 频率编码（Frequency Encoding）

7.1 理论回顾

频率编码用每个类别出现的频率（或计数）来替代类别名称。当类别的出现频率与目标变量有关时，这种方法非常有效。

适用场景：类别频率与目标变量相关的情况

注意事项：如果不同类别有相同频率，会产生相同的编码值

7.2 示例代码

```
# 计算频率
freq = data['city'].value_counts(normalize=True)
data['city_freq'] = data['city'].map(freq)


print('频率编码结果：')
print(data[['city', 'city_freq']])
print()


# 计数编码
count = data['city'].value_counts()
data['city_count'] = data['city'].map(count)
print('计数编码结果：')
print(data[['city', 'city_count']])
```

7.3 动手练习

练习 7-1：对多列进行频率编码

要求：封装一个 freq_encode(df, columns) 函数，能同时对多个列进行频率编码

对 city 和 color 列同时进行频率编码并查看结果

☑ **在下方编写你的代码：**

练习 7-2：频率冲突问题

要求：创建一个数据集，使两个不同类别有相同的出现频率

思考并实现解决频率冲突的方法（提示：加入微小扰动或结合其他编码）

☑ **在下方编写你的代码：**

第八章　综合实战练习

8.1 编码方法选择指南

编码方法	数据类型	基数范围	维度增长	推荐模型
Label	有序/无序	不限	无	树模型
One-Hot	无序	< 10	n列	线性模型
Ordinal	有序	不限	无	所有模型
Target	无序/有序	> 10	无	所有模型
Binary	无序	10-100	log2(n)列	所有模型
Frequency	无序	不限	无	树模型

8.2 综合挑战题

挑战 A：完整的数据编码流程

使用以下数据集，对每个分类变量选择最合适的编码方法并实现：

```
challenge_data = pd.DataFrame({
    'gender': ['男', '女', '男', '女', '男'] * 20,
    'blood_type': ['A', 'B', 'O', 'AB', 'A'] * 20,
    'job_level': ['初级', '中级', '高级', '专家', '初级'] * 20,
    'province': [f'省份_{i}' for i in range(100)],
    'target': np.random.randint(0, 2, 100)
})
```

☑ **在下方编写你的代码：**

挑战 B：编码效果对比实验

要求：对同一个变量分别应用三种不同的编码方法，训练决策树模型，比较准确率

提示：使用 sklearn.tree.DecisionTreeClassifier 和 sklearn.model_selection.cross_val_score

☑ **在下方编写你的代码：**

第九章　自我检测与总结

9.1 知识检测

请判断以下说法是否正确，在括号中填写 ✓ 或 ✗：

（ ）Label Encoding 适合用于线性回归模型中的无序分类变量

（ ）One-Hot Encoding 会引入多重共线性问题，可以通过 drop='first' 解决

（ ）Ordinal Encoding 要求变量必须具有自然顺序

（ ）Target Encoding 的主要风险是维度爆炸

（ ）Binary Encoding 的输出维度为 log2(n)

（ ）Frequency Encoding 不会出现不同类别编码相同的问题

9.2 快速参考卡

编码方法	导入语句	核心代码
Label	`from sklearn.preprocessing import LabelEncoder`	`le.fit_transform(data['col'])`
One-Hot	`from sklearn.preprocessing import OneHotEncoder`	`ohe.fit_transform(data[['col']])`
Ordinal	`from sklearn.preprocessing import OrdinalEncoder`	`oe.fit_transform(data[['col']])`
Target	`import category_encoders as ce`	`ce.TargetEncoder(cols=['col'])`
Binary	`import category_encoders as ce`	`ce.BinaryEncoder(cols=['col'])`
Frequency	`pandas (no extra lib)`	`data['col'].map(freq)`

9.3 学习笔记

请在下方记录你在实践过程中的心得体会和疑问：

在下方编写你的代码：