数据准备分析文档

一、代码文件名

01_data_prep.py

二、依赖的包

Python 标准库

- `os` – 文件和目录操作
- `glob` – 文件路径模式匹配

第三方Python包

- `pandas` – 数据处理和分析
- `numpy` – 数值计算

三、方法分析

整体流程概述

本脚本是整个分析流程的第一步，负责从TCGA宫颈癌原始数据中提取、清洗和整合表达矩阵及临床信息。输出的数据将被后续所有分析脚本使用。

详细步骤说明

步骤1：配置路径和创建输出目录

```
BASE_DIR = r"d:\Works\R_learning\data\fig7-9\prognostic"
DATA_DIR = os.path.join(BASE_DIR, "data")
SAMPLE_SHEET_PATH = os.path.join(BASE_DIR, "gdc_sample_sheet.tsv")
CLINICAL_PATH = os.path.join(BASE_DIR, "clinical.tsv")
OUTPUT_DIR = r"d:\Works\R_learning\project\processed_data"
```

定义原始数据路径和输出目录，如果输出目录不存在则创建。

步骤2：读取样本映射表

```
sample_sheet = pd.read_csv(SAMPLE_SHEET_PATH, sep="\t")
file_id_to_case = dict(zip(sample_sheet['File ID'], sample_sheet['Case ID']
))
file_name_to_case = dict(zip(sample_sheet['File Name'], sample_sheet['Case
ID']))
```

建立文件ID/文件名到病人ID（Case ID）的映射关系，用于后续将文件名转换为可读的TCGA样本ID。

步骤3：读取和清洗临床数据

3.1 处理缺失值标记

```
clinical = pd.read_csv(CLINICAL_PATH, sep="\t",
                       na_values=["--", "not reported", "[Not Applicable]",
                                  "[Not Available]", "[Discrepancy]", "'--"])
```

使用 `na_values` 参数识别TCGA数据中常见的缺失值表示方式。

3.2 列名适配和重命名

由于TCGA GDC下载的数据列名可能带有 `category.field` 格式的前缀，脚本定义了一个映射字典来标准化列名：

- `cases.submitter_id` → `case_submitter_id`
- `demographic.vital_status` → `vital_status`
- `demographic.days_to_death` → `days_to_death`
- `diagnoses.days_to_last_follow_up` → `days_to_last_follow_up`
- 等等（包括年龄、种族、分期、分级等）

3.3 临床变量清洗

实施多个清洗函数以标准化临床变量：

- **Stage清洗**：统一为I, II, III, IV格式

```
def clean_stage(val):
    val = str(val).replace("Stage ", "").strip()
    if val.startswith("IV"): return "IV"
    # 类似处理III, II, I
```

- **T分期清洗**：取前两个字符（T1, T2, T3, T4, TX），Tis转为T0
- **M分期清洗**：统一为M0, M1, MX
- **N分期清洗**：统一为N0, N1, NX

3.4 去除重复病人记录

```
clinical_clean = clinical_clean.drop_duplicates(subset=['case_submitter_id'], keep='first')
```

TCGA数据中同一病人可能有多条记录，仅保留第一条以避免统计膨胀。

3.5 计算生存数据

```python
def get_survival_time(row):
    d_death = row.get('days_to_death')
    if pd.notna(d_death): return float(d_death)
    d_last = row.get('days_to_last_follow_up')
    if pd.notna(d_last): return float(d_last)
    return None

def get_status(row):
    status = row.get('vital_status')
    if status == 'Dead': return 1
    if status == 'Alive': return 0
    return None
```

- **OS_time**：优先使用 `days_to_death`，如果为空则使用 `days_to_last_follow_up`
- **OS_status**：死亡=1，存活=0

步骤4：遍历并合并表达矩阵

4.1 查找所有TSV文件

```python
all_files = glob.glob(os.path.join(DATA_DIR, "**", "*.tsv"), recursive=True)
```

递归搜索 `data/` 目录下所有 `.tsv` 文件。

4.2 逐文件读取和处理

对每个文件：

1. 通过文件名或父文件夹名匹配Case ID
2. 读取TSV文件（TCGA STAR Counts格式）
3. **关键清洗步骤**：
   - 过滤掉统计行（gene_id以'N_'开头的行，如N_unmapped）
   - 确保gene_name存在且不为空
4. 提取两种数据：
   - **TPM数据**（`tpm_unstranded` 列）：用于生存分析和风险评分
   - **Raw Counts数据**（`unstranded` 列）：用于DESeq2差异分析
5. 处理重复基因名：
   - TPM：分组取平均
   - Counts：分组求和

步骤5：构建和保存最终矩阵

5.1 TPM矩阵（Log2转换）

```
expression_matrix = pd.DataFrame(merged_data_tpm)
expression_matrix = expression_matrix.fillna(0)
expression_matrix_log = np.log2(expression_matrix + 1)
expression_matrix_log.to_csv("expression_matrix.csv")
```

- 填充缺失值为0
- 进行Log2(TPM+1)转换
- 保存为 `expression_matrix.csv`

5.2 Raw Counts矩阵

```
expression_matrix_counts = pd.DataFrame(merged_data_counts)
expression_matrix_counts = expression_matrix_counts.fillna(0).astype(int)
expression_matrix_counts.to_csv("expression_matrix_counts.csv")
```

- 填充缺失值为0
- 转换为整数类型
- 保存为 `expression_matrix_counts.csv`

输出文件

1. **clinical_cleaned.csv**：清洗后的临床数据，包含标准化的生存信息和临床变量
2. **expression_matrix.csv**：Log2转换后的TPM表达矩阵（行=基因，列=样本）
3. **expression_matrix_counts.csv**：Raw Counts表达矩阵（行=基因，列=样本）
