STAT243 Lecture 2.2 Reading data from text files into Python

1 Core Python functions

- 使用 Pandas 读取文本数据：
  - `read_table` 与 `read_csv` 读取分隔符文本；
    关键参数：
    - `sep` 分隔符
    - `header` 是否有表头。
  - 定宽文本使用 `read_fwf()` 读取为 DataFrame。
  - 类型推断：Pandas 会自动推断，但更稳妥做法是通过 `dtype` 明确指定各列类型。
  - `sep` 支持正则：可以用正则表达式按空白或复杂分隔模式拆分。
- 示例 1：`RTADataSub.csv` 初读入与缺失处理。

Python

```python
import os, pandas as pd

# 初次读取：不指定 dtype 时，多数列为 object
dat = pd.read_table(os.path.join('..', 'data', 'RTADataSub.csv'),
                    sep = ',', header = None)
print(dat.dtypes.head())
# Output:
# 0    object
# 1    object
# 2    object
# 3    object
# 4    object
# dtype: object

print(dat.loc[0,1])
# Output:
# 2336

print(type(dat.loc[0,1]))
# Output:
# <class 'str'>

# 将 'x' 视为缺失
dat2 = pd.read_table(os.path.join('..', 'data', 'RTADataSub.csv'),
                     sep = ',', header = None, na_values = 'x')
print(dat2.dtypes.head())
# Output:
# 0    object
# 1    float64
# 2    float64
# 3    float64
# 4    float64
# dtype: object

print(dat2.loc[:,1].unique())
# Output:
# array([2336.,   nan, 1450., ...])
```

- 示例 2：`hivSequ.csv` 中通过 `dtype` 精确控制多列类型。

Python

```python
dat = pd.read_table(os.path.join('..', 'data', 'hivSequ.csv'),
                    sep = ',', header = 0,
                    dtype = {
                        'PatientID': int,
                        'Resp': int,
                        'PR Seq': str,
                        'RT Seq': str,
                        'VL-t0': float,
                        'CD4-t0': int
                    })
print(dat.dtypes)
# Output:
# PatientID      int64
# Resp           int64
# PR Seq        object
# RT Seq        object
# VL-t0        float64
# CD4-t0         int64
# dtype: object

print(dat.loc[0,'PR Seq'])
# Output:
# CCTCAAATCACTCTTT...
```

- 只读取部分列：使用 `usecols` 指定需要的列可以减少内存与 I/O。
- 读入前检查：建议先在终端用 `less` 等预览原始文件，以便发现分隔符、缺失标记、列数不齐等问题。
- 若文件列宽不齐（ragged lines）：可以先逐行读为字符串，再按切片处理固定位置字段。

Python

```python
file_path = os.path.join('..', 'data', 'precip.txt')
with open(file_path, 'r') as file:
    lines = file.readlines()

id = [line[3:11] for line in lines]
year = [int(line[17:21]) for line in lines]
month = [int(line[21:23]) for line in lines]
nvalues = [int(line[27:30]) for line in lines]
print(year[0:5])
# Output:
# [2010, 2010, 2010, 2010, 2010]
```

- 上述 `precip.txt` 实际为 fixed-width，使用 `pandas.read_fwf()` 更合适。

Remark

`with` 语句是标准的 Python 文件上下文管理方式，离开代码块会自动关闭文件句柄。

2 Connections and streaming

- 除了普通文件，还可以从不同连接读取：压缩文件、归档、子进程输出、网络资源等。

Python

```python
import gzip
with gzip.open('dat.csv.gz', 'r') as file:
    lines = file.readlines()
```

```
# Output:
# [b'...first line...', b'...']  # 示例

import zipfile
with zipfile.ZipFile('dat.zip', 'r') as archive:
    with archive.open('data.txt', 'r') as file:
        lines = file.readlines()
# Output:
# [b'...first line...', b'...']  # 示例

import subprocess, io
command = "ls -al"
output = subprocess.check_output(command, shell = True)
with io.BytesIO(output) as stream:
    content = stream.readlines()
# Output:
# [b'total ...', b'drwxr-xr-x ...', ...]  # 示例

# 直接从 URL 读取（制表符分隔）
df = pd.read_csv("https://download.bls.gov/pub/time.series/cu/cu.item", sep="\t")
print(df.shape)
# Output:
# (N, M)  # 示例：行列尺寸
```

- 流式/分块读取（online processing / streaming / chunking）：适用于大文件。

Python

```
file_path = os.path.join('..', 'data', 'RTADataSub.csv')
chunksize = 50
with pd.read_csv(file_path, chunksize = chunksize) as reader:
    for chunk in reader:
        print(f'Read {len(chunk)} rows.')
        # Output:
        # Read 50 rows.
```

- 将字符串当作类文件对象读取，便于对接只接受 file-like 的 API。

Python

```
file_path = os.path.join('..', 'data', 'precip.txt')
with open(file_path, 'r') as file:
    text = file.read()

stringIOtext = io.StringIO(text)

df = pd.read_fwf(stringIOtext, header = None, widths = [3,8,4,2,4,2])
print(df.head().shape)
# Output:
# (5, 6)  # 示例：前五行的形状
```

- 也可以创建连接用于写出，但需要先显式打开连接（写出在下一节）。

3 File paths

1. 避免在代码中硬编码绝对路径，改用项目内的相对路径。

Python

```
dat = pd.read_csv('../data/cpds.csv')
```

```python
print(dat.shape)
# Output:
# (N, M)  # 示例
```

2. 使用 UNIX 风格分隔符具有跨平台性；Windows 风格 `\` 在 Linux/Mac 上不可用。

Python

```python
# good: 跨平台
pd.read_csv('../data/cpds.csv')
# Output:
# (N, M)  # 示例

# bad: 在 Linux/Mac 上不可用
pd.read_csv('..\\data\\cpds.csv')
# Output:
# FileNotFoundError: [Errno 2] No such file or directory: '..\\data\\cpds.csv'  # 示例
```

3. 更好的方法是: 使用 `os.path.join` 构造不依赖于操作系统的路径。

Python

```python
pd.read_csv(os.path.join('..', 'data', 'cpds.csv'))
```

4 Reading data quickly: Arrow and Polars

- Apache Arrow：列式内存布局，Python 通过 PyArrow 调用。
  - 同列值连续存放，支持高效访问。
  - 可从 Parquet、Arrow 原生格式、文本读取。
  - 按需读取，避免整表常驻内存。
- 其他避免全量入内存方案：`Dask`、`numpy.load(mmap_mode=...)`。
- polars：强调速度的 DataFrame 库，常见对比示例如下。

Python

```python
import polars, time

t0 = time.time()
dat = pd.read_csv(os.path.join('..', 'data', 'airline.csv'))
t1 = time.time()

dat2 = polars.read_csv(os.path.join('..', 'data', 'airline.csv'), null_values = ['NA'])
t2 = time.time()
print(f"Timing for Pandas: {t1-t0}.")
# Output:
# Timing for Pandas: 0.99.
print(f"Timing for Polars: {t2-t1}.")
# Output:
# Timing for Polars: 0.91.
```