ChatGPT

基于Pandas的传染病报告卡数据处理方法

在处理来自中国大疫情网的传染病报告卡 CSV 数据时，可以利用Pandas库提供的强大数据处理功能。下面将按照常见步骤系统介绍具体方法，包括：加载数据和删除无用列、高效筛选行列、日期字符串转换为datetime及时间筛选、描述统计与分组聚合，并推荐相关的常用函数和操作技巧。

1. 加载CSV数据并删除无用列

首先使用 `pandas.read_csv()` 将CSV文件加载为DataFrame。例如：

```
import pandas as pd
df = pd.read_csv('reports.csv')  # 读取传染病报告卡数据
```

如果数据文件包含很多列但只需要其中一部分，可在读取时通过 `usecols` 参数指定需要的列，从而**加快加载速度并降低内存消耗** [1]。例如：

```
# 只读取所需的部分列
cols_to_use = ['报告日期', '发病日期', '疾病名称', '年龄', '性别', '省份']
df = pd.read_csv('reports.csv', usecols=cols_to_use)
```

加载后，我们通常先检查数据的基本情况：`df.shape` 查看行列数，`df.columns` 查看列名，`df.head()` 预览前几行，`df.info()` 查看各列类型和非空值数等。

接着，删除分析中不需要的无用列。Pandas提供了 `DataFrame.drop()` 函数用于按列名删除列 [2]。例如，若 `df` 包含一些冗余列 `"无用列1"` 和 `"无用列2"`，可以执行：

```
df = df.drop(columns=['无用列1', '无用列2'])
```

以上代码会删除指定的列 [2]（传入 `columns` 列表，`axis=1` 也表示按列操作）。如果希望直接在原DataFrame中修改，可加参数 `inplace=True`。需要注意，若CSV文件中存在索引列（例如因保存时未去除索引而出现的“Unnamed: 0”列），也可以用同样的方法将其删除。

2. 高效筛选所需的列和行

筛选列： 在DataFrame中选择我们需要的子集列非常简单。Pandas允许使用 `[]` 括号操作符一次选取一个或多个列 [3]。例如，如果只关心“疾病名称”、“发病日期”和“省份”三列，可以创建一个新DataFrame：

```
df_subset = df[['疾病名称', '发病日期', '省份']]
```

上述代码通过列名列表筛选出相应的列 [3]。另外，也可以使用 `df.filter()` 按照列名称模式筛选，或使用 `df.loc[:, 列名列表]` 达到同样目的。选择列时推荐直接使用列名列表的方式，语义清晰且操作简单。

筛选行： 对行的过滤通常基于条件（布尔索引）进行，这在Pandas中是高度矢量化且高效的。最常用方法是利用布尔条件筛选，即在`df[...]`中传入一个条件判断[4]。例如，筛选疾病为“肺结核”的数据：

```
tb_df = df[df['疾病名称'] == '肺结核']
```

上例中，`df['疾病名称'] == '肺结核'`会生成一个布尔Series，True表示满足条件的行，False则过滤掉。将该布尔Series传入`df[...]`即可得到仅包含疾病名称为“肺结核”的行[4]。可以组合多个条件进行筛选，需使用按位运算符如`&`（与）、`|`（或），同时用括号括起各个条件。例如筛选地区为“北京市”且年龄大于60的病例：

```
mask = (df['省份'] == '北京市') & (df['年龄'] > 60)
df_beijing_old = df[mask]
```

Pandas会对上述条件计算进行逐元素的矢量化比较，其效率远高于Python原生循环，即使对三四万行的数据也能快速完成筛选。

按值列表筛选： 若要筛选“疾病名称”属于某几个特定值的行，推荐使用`Series.isin()`方法。一旦生成表示成员关系的布尔Series，同样可以用于过滤[5]。例如：

```
# 筛选疾病名称为甲型或乙型肝炎的病例
df_hepatitis = df[df['疾病名称'].isin(['甲型肝炎', '乙型肝炎'])]
```

上例利用`isin`实现了基于列表的过滤[5]。类似地，`~df['列'].isin([...])`可以筛选不在列表内的值。对于数值范围筛选，可使用`between()`方法：如`df[df['年龄'].between(0, 14)]`筛选年龄在0到14之间的儿童病例[6]。

其他筛选技巧： Pandas还提供了`query()`方法，可用类SQL语法的字符串表达式筛选数据[7]。例如：`df.query("省份 == '北京市' and 年龄 > 60")`会得到北京市且年龄大于60的记录（注意列名非字母字符需要用反引号或先重命名[7]）。另外，`df.loc[]`和`df.iloc[]`方法分别可根据标签索引和位置索引筛选数据行列，当只需要按行标签或位置精确选取时非常有用。总体而言，以上方法都是**矢量化**操作，能够高效处理数万行的数据，而无需显式编写循环。

3. 日期字符串转换为Datetime并进行时间筛选

传染病报告卡中的日期字段通常是字符串格式（如“2020-05-01”或“20200501”等）。在分析前，需要先将其转换为pandas的日期时间类型`datetime64[ns]`，以便进行基于时间的运算和筛选。Pandas 提供了强大的`pd.to_datetime()`函数，能够将多种格式的字符串转换为datetime类型[8]：

```
# 将字符串格式的日期列转换为datetime类型
df['报告日期'] = pd.to_datetime(df['报告日期'])
df['发病日期'] = pd.to_datetime(df['发病日期'], format="%Y-%m-%d")
```

上例中，我们将“报告日期”和“发病日期”两列就地转换为了datetime。可以根据实际格式指定`format`参数以提高解析速度，例如格式为YYYY-MM-DD时使用`format="%Y-%m-%d"`。`pd.to_datetime`非常灵活，**可处**

理多种日期时间字符串格式，并提供诸如 `errors='coerce'` 等参数来处理解析失败的情况 [8]。转换完成后，可用 `df.dtypes` 验证列类型已经变为 `datetime64[ns]`。

将日期列转为datetime类型后，就可以方便地按时间过滤和分析数据：

- **按日期范围筛选：** 直接对datetime列进行比较筛选即可。例如筛选2020年的数据：

```
start_date = pd.to_datetime("2020-01-01")
end_date = pd.to_datetime("2020-12-31")
df_2020 = df[(df['报告日期'] >= start_date) & (df['报告日期'] <= end_date)]
```

或者利用 `between` 方法更简洁：`df_2020 = df[df['报告日期'].between('2020-01-01', '2020-12-31')]`。对于月度或季度范围筛选，也可以构造对应的开始和结束日期来应用同样逻辑。

- **按年份或月份筛选：** 利用DatetimeIndex的 `dt` 访问器提取日期的年、月、日等属性。例如筛选发病月份为1月的记录：`df_january = df[df['发病日期'].dt.month == 1]`。类似地，`df['发病日期'].dt.year == 2025` 可得到发病年份为2025的行。也可以先将日期列设为索引（如 `df.set_index('报告日期', inplace=True)`），然后使用 `df.loc['2020']` 快速取得2020年期间的数据。

通过将字符串转换为datetime，不仅方便筛选，还能利用Pandas丰富的时间序列功能进行按时段分组、重采样(resample)等更深入的分析。

4. 描述性统计与分组聚合分析

在清洗并筛选出需要的数据后，往往需要了解数据的总体分布情况，以及根据某些分类字段对数据进行汇总统计。

描述性统计： Pandas 提供了 `DataFrame.describe()` 方法来生成各数值列的常用统计量 [9]。例如：

```
print(df.describe())
```

该方法会一次性计算计数(count)、均值(mean)、标准差(std)、最小值、四分位数、中位数以及最大值等指标 [9]。它对每个数值型列给出上述统计量，非常便于了解诸如患者年龄等数值分布情况。如果DataFrame中有分类变量（object类型），可以用 `df.describe(include='all')` 查看其计数和类别数等信息。除了 `describe()` 之外，也可以对Series直接调用诸如 `df['年龄'].mean()`、`df['年龄'].median()`、`df['年龄'].min()` 等函数获取特定统计量。对于分类字段，常用 `value_counts()` 来统计每个类别的频数，例如：

```
df['疾病名称'].value_counts()
```

将返回每种疾病报告的病例数，有助于快速了解主要传染病的病例数分布。

分组聚合： 为了深入分析，不同维度（如不同疾病、不同地区、不同时间）的分组统计是关键步骤。Pandas的 `groupby()` 功能与SQL中的GROUP BY类似，能够根据一个或多个键对数据分组，然后对每组应用聚合函数计算统计量。例如，我们希望按“疾病名称”汇总病例数量和平均年龄：

```
grouped = df.groupby('疾病名称')
cases_per_disease = grouped.size()  # 每种疾病的报告数
avg_age_per_disease = grouped['年龄'].mean()  # 每种疾病的患者平均年龄

# 或一次性聚合多个指标：
stats_per_disease = df.groupby('疾病名称').agg({'姓名':'count', '年龄':'mean'})
```

上述代码通过 `groupby('疾病名称')` 将数据按疾病分类分组，然后分别计算了每组的元素个数（即病例数）和平均年龄。`grouped.size()` 返回各组的大小，也可以使用 `grouped['姓名'].count()` 达到类似效果（假设每行代表一名患者，“姓名”不为空）。使用 `agg` 则可以对不同列应用不同的聚合函数，并将结果组合成一个新的DataFrame。常用的聚合函数还包括 `sum()`、`min()`、`max()`、`median()`、`std()` 等 [10]。

例如，按省份统计病例总数，也可写作：

```
cases_per_province = df.groupby('省份')['姓名'].count()
```

如果想同时计算每省的病例数和平均年龄，可以用 `agg` 传入字典指定不同列的聚合：

```
prov_stats = df.groupby('省份').agg(病例数=('姓名','count'), 平均年龄=('年龄','mean'))
print(prov_stats.head())
```

通过 `groupby` 分组后应用聚合函数，我们就能够得到各组别的汇总统计值。[10] 指出，我们不仅可以计算均值，还能方便地使用sum, max, min 等聚合函数获取更多信息。例如，可以进一步计算每个城市的总销售额或每种疾病的总病例数等 [10]（本例中“销售额”对应于需要汇总的数值列，“城市”对应分组字段）。如果需要，可一次性应用多个聚合函数，例如 `grouped['年龄'].agg(['mean','median','max','min'])` 会给出每组年龄的均值、中位数、最大值、最小值。

需要注意，分组结果是一个GroupBy对象，直接打印只会显示分组对象的信息而非结果。如果要查看具体结果，需要像上面一样调用聚合函数（如 `mean()`、`count()` 等)或使用 `list(grouped)` 迭代取出分组。本质上，Pandas的分组操作遵循“拆分-应用-合并”的模式，将DataFrame按键拆分后，对各组应用函数计算，最后将各组结果合并得到新的数据结构。这一过程在几万行的数据集上通常也能快速执行，因为底层实现是优化过的C代码或矢量计算。

5. 常用函数、方法和操作技巧推荐

综合以上步骤，以下是适用于数据读取与清洗、筛选以及统计分析的一些Pandas常用函数和操作技巧：

- **`pd.read_csv()`**：用于读取CSV文件为DataFrame。常用参数有 `usecols`（指定需要读取的列，提高效率 [1]）、`nrows`（读取前N行用于测试）、`dtype`（指定列数据类型）等。
- **`DataFrame.drop()`**：删除行或列的通用方法。通过 `axis` 或 `columns` 参数指定按列删除，可一次移除不需要的多列 [2]。`inplace=True` 时会在原DataFrame上直接删除。
- **索引操作**：`df[]` 中传入列名列表可以选择子集列 [3]；传入布尔Series可以过滤行 [4]。`df.loc[row_indexer, col_indexer]` 根据标签选择行列，`df.iloc[]` 根据整数位置选择，用于更精细或复杂的索引操作。
- **条件筛选**：利用布尔条件进行过滤是Pandas的强项。例如 `df[df['列'] > value]`、`df[df['列'] == '某值']`、`df[df['列'].isin(['值1','值2'])]` 等 [5]。使用 `&`、`|` 组合多个条件时别忘了用括号括

起每个条件。`df.query('条件表达式')`提供了更接近SQL的查询风格，对于列名符合Python变量规则的情况非常方便（列名包含空格或中文时可用反引号括起或先重命名[7]）。

- **`pd.to_datetime()`**：日期转换利器。将字符串批量转换为datetime64类型，可解析多种日期格式[8]。常用参数有`format`（指定日期格式提高解析速度）、`errors='coerce'`（遇到不合法日期时返回NaT而非报错）等。日期列转换后，可使用`dt`属性访问年、月、日、星期几等进行进一步分析和过滤。
- **描述统计函数**：`df.describe()`一次性查看数据的分布特征[9]；`df.mean()`，`df.std()`，`df.min()`，`df.median()`等计算整列的统计量；`df['列'].value_counts()`快速统计类别型数据各值频数。还有`df.info()`可以帮助了解数据规模、各列类型和缺失值情况，方便在处理之前评估数据质量。
- **`DataFrame.groupby()`**：分组操作的核心。与聚合函数连用，如`grouped = df.groupby('列')`然后`grouped.size()`或`grouped['其他列'].mean()`等，可以得到各组的汇总指标[10]。对于更复杂的需求，使用`grouped.agg()`一次应用多个聚合函数甚至自定义函数。
- **其他技巧**：`pivot_table()`用于多维度汇总（类似Excel数据透视表），`pd.cut()`可以将连续数值离散化分箱，`df.sort_values()`对数据排序便于查看最大最小值，`df.fillna()/dropna()`处理缺失值等等。这些根据具体分析需求选择使用即可。

以上方法和函数覆盖了从数据加载、清洗筛选到基本统计分析的主要步骤。在处理实际的传染病报告卡数据（数万行规模）时，充分利用Pandas的矢量化操作和内置函数，既可以保证代码简洁易读，又能够高效地完成数据整理与分析任务。通过这些技术手段，相信可以更从容地应对中国大疫情网提供的疫情数据分析需求，为后续的深入研究（如进一步的建模或趋势分析）打下良好基础。[9][10]

---

[1] 史上最全！用Pandas读取CSV，看这篇就够了-腾讯云开发者社区-腾讯云
https://cloud.tencent.com/developer/article/1856554

[2] pandas删除没有列名的列_pandas筛选列没有列名-CSDN博客
https://blog.csdn.net/xiaoyw71/article/details/121472150

[3] [4] [5] [6] [7] Pandas中选择和过滤数据的终极指南_腾讯新闻
https://news.qq.com/rain/a/20231130A01XZD00

[8] 如何在 Pandas 中将 DataFrame 列转换为日期时间？-CDA数据分析师官网
https://www.cda.cn/bigdata/201800.html

[9] [10] 〖100天精通Python〗Day57：Python 数据分析_Pandas数据描述性统计，分组聚合，数据透视表和相关性分析-阿里云开发者社区
https://developer.aliyun.com/article/1346842