

# PS2 报告说明

任务 1:

引用说明:

其中数据分组与聚合（`groupby` + `sum`/`size`）参考 pandas 官方“分组操作”教程，涵盖`groupby`的基本用法及常用聚合函数（如`sum`统计总和、`size`统计数量）。数据排序（`sort\_values`）数据降序排序，并取前 N 行（`head(10)`）。参考 pandas 官方对`sort\_values`方法的说明，包括`by`（排序依据列）和`ascending`（排序方向）参数的使用。

绘制时间序列图（`matplotlib.pyplot.plot`）用`matplotlib`绘制年份与地震数量的趋势线图。参考 Matplotlib 官方“基础绘图”教程，讲解`plot`函数绘制线图的基本步骤，包括标题（`title`）、坐标轴标签（`xlabel`/`ylabel`）等设置。

自定义函数与遍历唯一值（`unique` + 函数应用）定义`CountEq\_LargestEq`函数统计各国地震信息，并通过`unique()`获取所有国家并遍历处理。参考 GeeksforGeeks 教程，介绍如何获取 Series 中的唯一值（如所有国家名称）。自定义函数应用：Real Python 的 pandas 教程，讲解如何结合遍历和自定义函数处理分组数据。DataFrame 基础操作（`columns`/`head`/`reset\_index`）查看列名（`columns`）、预览数据（`head`）、重置索引（`reset\_index`）。

```
import pandas as pd
import matplotlib.pyplot as plt

Sig_Eqs = pd.read_csv("earthquakes-data.tsv", sep="\t")

print("数据集中所有的列名:")
for col in Sig_Eqs.columns:
    print(f"- {col}")

print("\n数据前5行预览:")
print(Sig_Eqs.head())

数据集中所有的列名:
- Search Parameters
- Id
- Year
- Mo
- Dy
- Hr
- Mn
- Sec
- Tsc
- Vol
- Country
- Area
- Region
- Location Name
- Latitude
- Longitude
- Focal Depth (km)
- Mag
- Mw
- Ms
- Mb
- Ml
- Mfa
- Uri
- MMI Int
- Deaths
- Death Description
- Missing Description
- Injuries
- Injuries Description
- Death (SMail)
- Damage Description
- Houses Destroyed
- Houses Destroyed Description
- Houses Damaged
```

1.1:

```
[2]: # 任务1.1: 计算每个国家的总死亡人数并取前10名
# 1. 处理缺失值: 死亡人数为空的按0处理
Sig_Eqs["Deaths"] = Sig_Eqs["Deaths"].fillna(0)
# 2. 按国家分组, 计算死亡总数
country_death_counts = Sig_Eqs.groupby('Country')[['Deaths']].sum().reset_index()
# 3. 按死亡总数降序排序, 取前10名
top10_countries = country_death_counts.sort_values(by='Deaths', ascending=False).head(10)
# 4. 显示结果
print("== 任务1.1: 地震死亡总数前10的国家 ==")
print(top10_countries)

== 任务1.1: 地震死亡总数前10的国家 ==
   Country      Deaths
28      CHINA  2139210.0
142     TURKEY  1199742.0
65      IRAN   1014453.0
69      ITALY   498219.0
132     SYRIA   419226.0
58      HAITI   323484.0
10  AZERBAIJAN   319251.0
71      JAPAN   242445.0
6   ARMENIA   191890.0
103     PAKISTAN  145083.0
```

## 1.2:

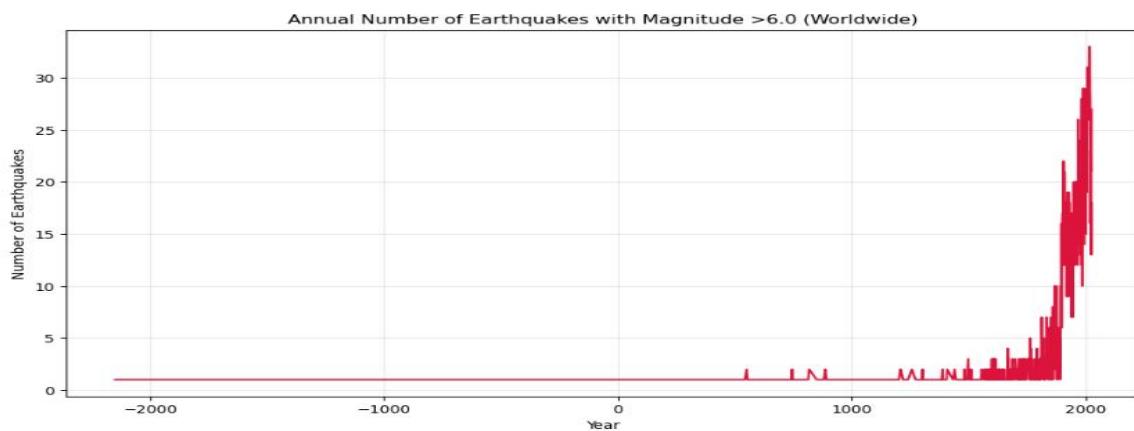
```
# 任务1.2: 每年全球震级>6.0的地震总数及时间序列

# 1. 筛选震级>6.0的地震, 排除震级为0的行
filtered_quakes = Sig_Eqs[(Sig_Eqs['Mag'] > 6.0) & (Sig_Eqs['Mag'].notna())]

# 2. 按年份分组, 统计每年的地震数量
yearly_counts = filtered_quakes.groupby('Year').size().reset_index(name='Quake_Count')

# 3. 绘制时间序列图
plt.figure(figsize=(12, 6))
plt.plot(yearly_counts['Year'], yearly_counts['Quake_Count'], color='crimson')
plt.title('Annual Number of Earthquakes with Magnitude >6.0 (Worldwide)')
plt.xlabel('Year')
plt.ylabel('Number of Earthquakes')
plt.grid(alpha=0.3)
plt.show()

# 4. 趋势分析
print("==== 任务1.2: 趋势分析 ===")
print("观察结果: 从图中可以看到, 1900年之后震级>6.0的地震记录数量明显增加。")
print("原因解释: 这并非实际地震数量增加, 主要是由于近现代地震监测技术进步, ")
print("早期数据因记录不完善导致数量偏低, 而现代能更全面地记录地震事件。")
```



==== 任务1.2: 趋势分析 ====  
观察结果: 从图中可以看到, 1900年之后震级>6.0的地震记录数量明显增加。  
原因解释: 这并非实际地震数量增加, 主要是由于近现代地震监测技术进步,  
早期数据因记录不完善导致数量偏低, 而现代能更全面地记录地震事件。

## 1.3:

```
# 任务1.3: 统计各国地震总数及最大地震日期
def CountEqLargestEq(country_name):
    # 1. 筛选出指定国家的地震
    country_quakes = Sig_Eqs[Sig_Eqs['Country'] == country_name].copy()

    # 2. 计算地震总数
    total_quakes = len(country_quakes)

    # 3. 找到该国震级最大的地震, 并输出震级及对应的年份
    if total_quakes > 0:
        return (country_name, "无地震记录")
    else:
        valid_quakes = country_quakes[country_quakes['Mag'].notna()]
        if valid_quakes.empty:
            return (country_name, "无有效震级记录")
        largest_quake = valid_quakes.loc[valid_quakes['Mag'].idxmax()]

        # 4. 返回日期, 并返回震级及对应的年份
        year = int(largest_quake['Year']) if pd.notna(largest_quake['Year']) else '未知'
        month = int(largest_quake['Mo']) if pd.notna(largest_quake['Mo']) else '未知'
        day = int(largest_quake['Do']) if pd.notna(largest_quake['Do']) else '未知'
        largest_date = f'{year}年{month}月{day}日'

        return (total_quakes, largest_date)

    # 5. 返回日期, 并返回震级及对应的年份
    # 6. 将所有国家的地震记录合并
    all_countries = Sig_Eqs['Country'].unique()
    results = []

    for country in all_countries:
        total, date = CountEqLargestEq(country)
        results.append((country, total, date))

    # 7. 将结果整理成DataFrame并按地震总数降序排序
    results_df = pd.DataFrame(results)
    sorted_results = results_df.sort_values(by='Total_Quakes', ascending=False)

    # 打印结果
    print("==== 任务1.3: 各国地震统计 (按总数降序) ====")
    print(sorted_results)

    # 打印总计
    print(f'[ {len(sorted_results)} rows x 3 columns ]')
```

Country	Total_Quakes	Largest_Quake_Date
CHINA	623	1668年7月25日
JAPAN	424	2011年3月11日
INDONESIA	421	2004年12月26日
IRAN	388	856年12月22日
TURKEY	358	1939年12月26日
***	***	***
BURUNDI	1	2004年2月24日
ZAMBIA	1	2017年2月24日
MADAGASCAR	1	2017年1月11日
COMOROS	1	2018年5月15日
NAN	0	无地震记录

## 任务 2:

1. 读取 CSV 文件通过`low\_memory=False`避免因列数据类型混合导致的警告。参考资料 pandas 官方文档对`read\_csv`的说明，重点解释`low\_memory`参数（控制是否分块推断数据类型，关闭可避免混合类型警告）。 2. 字符串拆分（`str.split`）将`WND`字段按逗号拆分为多列，提取风向、风速等子信息。参考 pandas 官方“字符串处理”教程，详解`str.split`方法（按分隔符拆分字符串，`expand=True`将结果转换为 DataFrame）。 3. 数值转换（`pd.to\_numeric`与`errors='coerce'`）将拆分后的风速原始值转换为数值类型，无效值转为`NaN`。参考 pandas 官方对`to\_numeric`的说明，重点讲解`errors='coerce'`参数（将无法转换的字符串强制转为`NaN`，避免报错）。 4. 条件过滤（布尔索引与`isin`）通过多条件组合（风速有效、质量码合格等）筛选有效数据。参考（1）pandas 布尔索引教程：讲解如何用`&`（且）组合条件，实现数据过滤。（2）`isin`方法：pandas 文档说明如何检查值是否在指定列表中（如质量码是否为`['0','1','4','5','9']`）。 5. 日期时间处理（`pd.to\_datetime`与`dt.strftime`）将`DATE`列转换为 datetime 格式，并提取“年月”字符串用于分组。参考（1）`pd.to\_datetime`：pandas 官方时间序列教程，讲解如何将字符串转换为 datetime 类型（`errors='coerce'`处理无效日期）。（2）`dt.strftime`：pandas 文档中`DatetimeProperties`的说明，讲解如何用字符串格式化日期（如`"%Y-%m"`转为“年-月”）。 6. 分组聚合与排序（`groupby` + `mean` + `sort\_values`）按“年月”分组计算平均风速，并按时间排序。参考 pandas 分组聚合教程，涵盖`groupby`后用`mean`计算均值，以及`sort\_values`按分组键（年月）排序的用法。 7. 自定义绘图（`matplotlib.pyplot.plot`与图表美化）绘制月平均风速趋势图，设置标记点、旋转标签、调整布局等。参考 Matplotlib 官方绘图教程，详解`plot`函数的`marker`（标记点）、`linestyle`（线样式）参数，以及`xticks(rotation)`（旋转 x 轴标签）、`tight\_layout`（自动调整布局）等美化方法。

```
import pandas as pd
import matplotlib.pyplot as plt

# 1. 读取数据
# 读取带有质量码的 CSV
df = pd.read_csv('2281305.csv', low_memory=False)

# 2. 正确解析WND字段（设置分隔符）
# WND字段修正示例：“040,2,N,0020,1”，多部分含义：
# [风向,风速,风向质量码,风速质量码,风速有效]
# 使用逗号分隔，按列拆分
wnd_parts = df['WND'].str.split(',', expand=True) # 按逗号拆分，逐列赋值

# 提取风速质量码，转换为数值
df['wind_speed_raw'] = pd.to_numeric(wnd_parts[3], errors='coerce')
# 计算平均风速
df['wind_speed'] = df['wind_speed_raw'] / 10

# 提取风向质量码
df['speed_quality'] = wnd_parts[4]
# 提取风向代码
df['type_code'] = wnd_parts[2]

# 3. 处理无效数据
# 清理条件：
# 1. 风速质量码不是有效值（9999）
# 2. 风速质量码不等于0/1/4/5/9，数据无效
# 3. 风速质量码不等于0/1/4/5/9，数据无效
filter_mask = ((df['wind_speed_raw'] != 9999) &
               (df['speed_quality'].isin(['0', '1', '4', '5', '9'])) &
               (df['type_code'] != '9'))
filtered_df = df[filter_mask].copy() # 等效于filtered_df=df[filter_mask].copy()

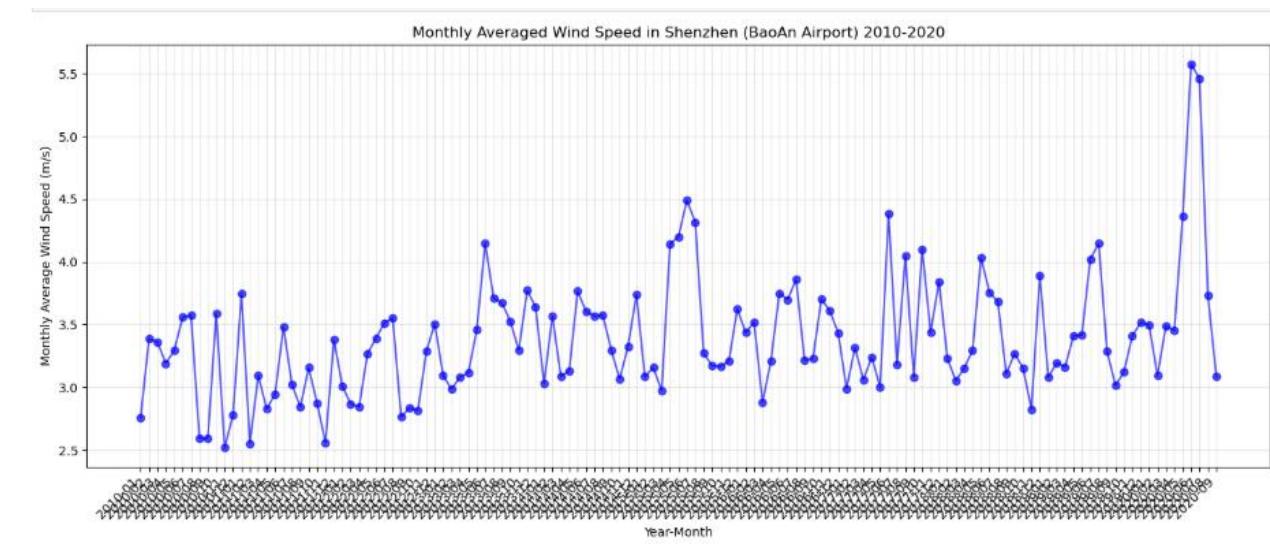
# 4. 处理时间
# 将DATE转化为datetime格式
filtered_df['DATE'] = pd.to_datetime(filtered_df['DATE'], errors='coerce')
# 筛选出日期缺失的数据
filtered_df = filtered_df.dropna(subset=['DATE'])

# 提取“年月”用于分组
filtered_df['year_month'] = filtered_df['DATE'].dt.strftime('%Y-%m')

# 5. 计算月平均风速
# 提取月分组，计算平均风速
monthly_avg_wind = filtered_df.groupby('year_month')[['wind_speed']].mean().reset_index()
# 排序
monthly_avg_wind = monthly_avg_wind.sort_values('year_month')

# 6. 绘制趋势图
plt.figure(figsize=(14, 8))
plt.plot(monthly_avg_wind['year_month'],
         monthly_avg_wind['wind_speed'],
         marker='o', linestyle='--', color='blue', alpha=0.7)

plt.xlabel('Year-Month', fontsize=18)
plt.ylabel('Monthly Average Wind Speed (m/s)', fontsize=18)
plt.title('Monthly Averaged Wind Speed in Shenzhen (BaoAn Airport) 2010-2020', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



### 任务 3:

引用说明:

1. 数值转换与缺失值处理（`pd.to\_numeric` 与 `dropna`） 将“Cited by”列转换为数值类型（无效值转为`NaN`），并删除“Cited by”和“Year”列的缺失值。参考（1）`pd.to\_numeric`：pandas 官方文档，说明`errors='coerce`参数（强制无效值为`NaN`）的用法。（2）`dropna`：pandas 官方文档，讲解`subset`参数（指定需检查缺失值的列）的使用。
2. 分组聚合与时间序列绘图，按年份分组计算总被引次数，并用`matplotlib`绘制趋势图。参考（1）`groupby`与`sum`：pandas 分组聚合教程，说明按列分组后用`sum`计算总和的逻辑。（2）`matplotlib.plot`：Matplotlib 官方教程，详解`marker`（数据点标记）、`rotation`（坐标轴标签旋转）等参数的绘图设置。
3. 描述性统计（`describe`）生成“Cited by”列的基本统计量（均值、标准差、分位数等）。参考 pandas 官方文档对`describe`方法的说明，解释输出结果中各统计量的含义（如`count`为非缺失值数量、`mean`为均值）。
4. 正态性检验（`stats.shapiro`）用 Shapiro-Wilk 检验判断数据是否服从正态分布。参考 SciPy 官方文档对`shapiro`函数的说明，包括统计量和 p 值的解读（ $p < 0.05$  通常认为不服从正态分布）。
5. 独立样本 t 检验（`stats.ttest\_ind`）比较 2018 年和 2019 年被引次数的均值是否存在显著差异。参考 SciPy 官方文档对`ttest\_ind`的说明，解释`equal\_var=False`（不假设等方差，即 Welch t 检验）和`nan\_policy='omit'`（忽略缺失值）的参数用法。
6. 单因素方差分析（`stats.f\_oneway`）检验多个年份（2017、2018、2019）的被引次数均值是否存在显著差异。参考 SciPy 官方文档对`f\_oneway`的说明，解释方差分析的适用场景（比较两组及以上独立样本的均值）及结果解读（ $p < 0.05$  表示存在显著差异）。
7. 皮尔逊相关性分析（`stats.pearsonr`）分析“Year”与“Cited by”之间的线性相关程度及显著性。参考 SciPy 官方文档对`pearsonr`的说明，解释相关系数（范围[-1,1]，绝对值越大相关性越强）和 p 值（判断相关性是否显著）的含义。

```

# 进行初步数据处理
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats

# 3.1 加载并清洗数据
excel_file = pd.ExcelFile(r'C:\Users\Morechain&Hwachin\open access ecotoxicology research dataset1.xlsx')
df = excel_file.parse('open access ecotoxicology resea')

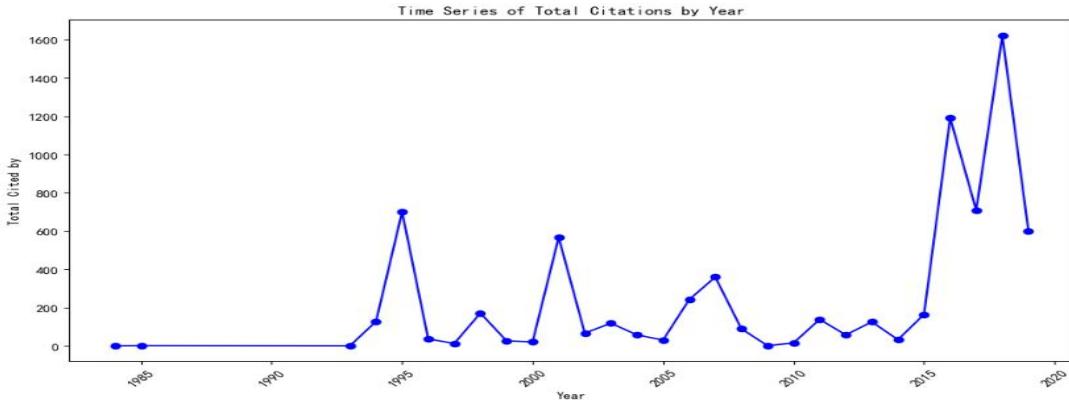
# 处理"Cited by"列，转换为数值类型，无法转换的设为NaN
df['Cited by'] = df.to_numeric(df['Cited by'], errors='coerce')

# 删除"Cited by"和"Year"列的缺失值（保证时间序列完整性）
df_clean = df.dropna(subset=['Cited by', 'Year']).copy()

# 3.2 绘制时间序列图
time_series = df_clean.groupby('Year')['Cited by'].sum().reset_index()

plt.figure(figsize=(10, 6))
plt.plot(time_series['Year'], time_series['Cited by'], marker='o', color='blue')
plt.xlabel('Year')
plt.ylabel('Total Cited by')
plt.title('Time Series of Total Citations by Year')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```



```

# 3.3 进行至少5种统计检验
cited_by = df_clean['Cited by']

# 1. 描述性统计
desc_stats = cited_by.describe()
print("----- 统计检验 1: 描述性统计 -----")
print(desc_stats)

# 2. 正态性检验
shapiro_stat, shapiro_p = stats.shapiro(cited_by)
print(f"\n----- 统计检验 2: Shapiro-Wilk正态性检验 ----- \n统计量 = ({shapiro_stat:.4f}), p值 = ({shapiro_p:.4f})")
if shapiro_p < 0.05:
    print(" 结论: 数据**不服从**正态分布 (p < 0.05) ")
else:
    print(" 结论: 数据**服从**正态分布 (p > 0.05) ")

# 3. 独立样本 t 检验 (2018 vs 2019)
year_2018 = df_clean[df_clean['Year'] == '2018'][['Cited by']]
year_2019 = df_clean[df_clean['Year'] == '2019'][['Cited by']]
if len(year_2018) > 0 and len(year_2019) > 0:
    t_stat, t_p = stats.ttest_ind(year_2018, year_2019, equal_var=False, nan_policy='omit')
    print(f"\n----- 统计检验 3: 独立样本 t 检验 (2018 vs 2019) ----- \n统计量 = ({t_stat:.4f}), p值 = ({t_p:.4f})")
    if t_p < 0.05:
        print(" 结论: 两年被引次数均值**存在显著差异** (p < 0.05) ")
    else:
        print(" 结论: 两年被引次数均值**无显著差异** (p > 0.05) ")
else:
    print("2018 或 2019 年数据长度为 0, 无法进行独立样本 t 检验")

# 4. 单因素方差分析 (2017, 2018, 2019)
years = ['2017', '2018', '2019']
groups = [df_clean[df_clean['Year'] == y][['Cited by']] for y in years]
valid_years = [y for y in groups if len(y) > 0]
if len(valid_years) >= 2:
    f_stat, anova_p = stats.f_oneway(*valid_years)
    print(f"\n----- 统计检验 4: 单因素方差分析 (2017, 2018, 2019) ----- \n统计量 = ({f_stat:.4f}), p值 = ({anova_p:.4f})")
    if anova_p < 0.05:
        print(" 结论: 至少有一个年份的被引次数均值**存在显著差异** (p < 0.05) ")
    else:
        print(" 结论: 各年份被引次数均值**无显著差异** (p > 0.05) ")
else:
    print(" 部分年份数据长度为 0, 无法进行单因素方差分析")

# 5. 相关性分析 (Year与Cited by)
df_clean.loc[:, 'Year_num'] = pd.to_numeric(df_clean['Year']) # 通过.loc赋值, 避免警告
corr, corr_p = stats.pearsonr(df_clean['Year_num'], df_clean['Cited by'])
print(f"\n----- 统计检验 5: 皮尔逊相关性分析 (Year与Cited by) ----- \n相关系数 = ({corr:.4f}), p值 = ({corr_p:.4f})")
if corr_p < 0.05:
    print(" 结论: Year与Cited by**存在显著相关性** (p < 0.05) ")
else:
    print(" 结论: Year与Cited by**无显著相关性** (p > 0.05) ")

```

```

----- 统计检验 1: 描述性统计 -----
count      507.000000
mean       14.390533
std        33.617374
min        1.000000
25%       2.000000
50%       5.000000
75%       14.000000
max       489.000000
Name: Cited by, dtype: float64

----- 统计检验 2: Shapiro-Wilk正态性检验 -----
统计量 = 0.3708, p值 = 0.0000
结论: 数据**不服从**正态分布 (p < 0.05)
2018 或 2019 年数据长度为 0, 无法进行独立样本 t 检验
部分年份数据长度为 0, 无法进行单因素方差分析

----- 统计检验 5: 皮尔逊相关性分析 (Year与Cited by) -----
相关系数 = -0.3584, p值 = 0.0000
结论: Year与Cited by**存在显著相关性** (p < 0.05)

```