# 附 录

|  |
| --- |
| **附录** **1** 天气和风向编码表 |
| 无 |
| |  |  | | --- | --- | | **原始天气** | **编码** | | 多云 | 1 | | 浮尘 | 2 | | 雷阵雨 | 3 | | 霾 | 4 | | 晴 | 5 | | 雾 | 6 | | 小雨 | 7 | | 扬沙 | 8 | | 阴 | 9 | | 阵雨 | 10 | | 中雨 | 11 | | 大雨 | 12 | | 暴雨 | 13 | | 雨 | 14 |  |  |  | | --- | --- | | **原始风向** | **编码** | | 北风 | 1 | | 东北风 | 2 | | 东风 | 3 | | 东南风 | 4 | | 南风 | 5 | | 西北风 | 6 | | 西风 | 7 | | 西南风 | 8 | |

|  |
| --- |
| **附录** **2** 发电数据清洗（python代码） |
| A1\_1-4发电数据缺失补充.py |
| 1. import os 2. import pandas as pd 3. import numpy as np 4. from scipy.optimize import curve\_fit 5. import matplotlib.pyplot as plt 6. from tqdm import tqdm 7. plt.rcParams['font.sans-serif'] = ['SimHei'] 8. plt.rcParams['axes.unicode\_minus'] = False 9. *# ======== 高斯函数 ========* 10. def gaussian(x, A, mu, sigma): 11. return A \* np.exp(-((x - mu) \*\* 2) / (2 \* sigma \*\* 2)) 12. *# ======== 单文件修复 ========* 13. def process\_power\_data(input\_path: str, output\_path: str): 14. """ 15. input\_path : 原始 Excel（含“时间、当日累计发电量kwh”） 16. output\_path: 修复后 Excel 保存路径 17. """ 18. *# 1) 若目标文件夹不存在，自动创建* 19. os.makedirs(os.path.dirname(output\_path), exist\_ok=True) 20. *# 2) 读取 & 生成完整 5 min 索引* 21. raw\_df = pd.read\_excel(input\_path, parse\_dates=['时间']) 22. df = raw\_df.drop\_duplicates('时间').set\_index('时间').sort\_index() 23. full\_index = pd.date\_range(df.index.min().floor('5min'), 24. df.index.max().ceil('5min'), 25. freq='5min', name='时间') 26. df = df.reindex(full\_index) 27. df['原始标记'] = df['当日累计发电量kwh'].notna() 28. *# 3) 功率、累计差* 29. df['累计发电量kwh'] = df['当日累计发电量kwh'].ffill() 30. df['功率'] = df['累计发电量kwh'].diff().clip(lower=0).fillna(0) 31. *# 4) 按天处理* 32. all\_dfs = [] 33. for day, day\_df in df.groupby(df.index.date): 34. day\_df = day\_df.copy() 35. day\_df['当日分钟'] = (day\_df.index - day\_df.index.normalize()).total\_seconds() / 60 36. *# —— 高斯拟合* 37. valid = (day\_df['功率'] > 0) & day\_df['原始标记'] 38. if valid.sum() > 10: 39. x, y = day\_df.loc[valid, ['当日分钟', '功率']].values.T 40. try: 41. dur = x.max() - x.min() 42. popt, \_ = curve\_fit(gaussian, x, y, p0=[y.max(), x.mean(), dur / 2], maxfev=5000) 43. day\_df['拟合功率'] = gaussian(day\_df['当日分钟'], \*popt).clip(lower=0) 44. except Exception: 45. day\_df['拟合功率'] = day\_df['功率'].rolling(6, min\_periods=1).mean() 46. else: 47. day\_df['拟合功率'] = day\_df['功率'] 48. *# —— 修复规则* 49. day\_df['累计差'] = day\_df['累计发电量kwh'].diff().fillna(0) 50. day\_df['修复功率'] = day\_df['功率'] 51. *# 1) 功率=0 & 累计差=0* 52. flat = (day\_df['功率'] == 0) & (day\_df['累计差'] == 0) 53. missing = flat & (~day\_df['原始标记'])          *# 真缺失* 54. keep0   = flat & day\_df['原始标记']            *# 夜间/停机* 55. day\_df.loc[missing, '修复功率'] = day\_df.loc[missing, '拟合功率'] 56. day\_df.loc[keep0,   '修复功率'] = 0 57. *# 2) 22 点-次日 2 点 强制 0* 58. night = (day\_df.index.hour >= 22) | (day\_df.index.hour < 2) 59. day\_df.loc[night, '修复功率'] = 0 60. *# 3) 0 → 大跳增* 61. jump\_thr = 400 62. prev = day\_df['功率'].shift(1).fillna(0) 63. jump\_mask = (prev == 0) & (day\_df['功率'] > jump\_thr) 64. day\_df.loc[jump\_mask, '修复功率'] = day\_df.loc[jump\_mask, '拟合功率'] 65. *# 4) 累计突增前段回填* 66. big = day\_df['累计差'] > 300 67. for ts in day\_df.index[big]: 68. for i in range(1, 13): 69. idx = ts - pd.Timedelta(minutes=5 \* i) 70. if idx not in day\_df.index: 71. break 72. if day\_df.at[idx, '功率'] > 0 or day\_df.at[idx, '累计差'] != 0: 73. break 74. if day\_df.at[idx, '修复功率'] == 0: 75. day\_df.at[idx, '修复功率'] = day\_df.at[idx, '拟合功率'] 76. all\_dfs.append(day\_df) 77. merged = pd.concat(all\_dfs) 78. merged['修复累计发电量'] = merged['修复功率'].fillna(0).cumsum() + merged['累计发电量kwh'].iloc[0] 79. merged[['累计发电量kwh', '功率', '拟合功率', '修复功率', '修复累计发电量']].to\_excel(output\_path) 80. print(f"✅ {os.path.basename(input\_path)} 处理完成 → {output\_path}") 81. *# ======== 批量调用 ========* 82. if \_\_name\_\_ == '\_\_main\_\_': 83. *# 待处理的电站 Excel 文件名* 84. stations = ["电站1发电数据.xlsx", 85. "电站2发电数据.xlsx", 86. "电站3发电数据.xlsx", 87. "电站4发电数据.xlsx"] 88. *# 输出目录* 89. out\_dir = r"A1\_发电数据\_缺失值补充" 90. os.makedirs(out\_dir, exist\_ok=True) 91. for fname in stations: 92. in\_path  = fname 93. base     = os.path.splitext(fname)[0] + "\_修复结果.xlsx" 94. out\_path = os.path.join(out\_dir, base) 95. process\_power\_data(in\_path, out\_path) |

|  |
| --- |
| **附录** **3** 发电数据小时化小处理（python代码） |
| A1\_1-4小时处理.py |
| 1. import pandas as pd 2. *# 要处理的四个修复结果文件名* 3. input\_files = [ 4. "电站1发电数据\_修复结果.xlsx", 5. "电站2发电数据\_修复结果.xlsx", 6. "电站3发电数据\_修复结果.xlsx", 7. "电站4发电数据\_修复结果.xlsx" 8. ] 9. for file in input\_files: 10. try: 11. *# 读取数据* 12. df = pd.read\_excel(file, parse\_dates=['时间']) 13. df = df.sort\_values('时间').reset\_index(drop=True) 14. *# 计算每日归零的修复累计发电量* 15. df['修复累计发电量（每日归零）'] = 0.0 16. df['日期'] = df['时间'].dt.date 17. for date, group in df.groupby('日期'): 18. cumulative = group['修复功率'].cumsum() 19. df.loc[group.index, '修复累计发电量（每日归零）'] = cumulative 20. df.drop(columns=['日期'], inplace=True) 21. *# 按小时汇总* 22. hourly\_df = df.resample('1h', on='时间').agg({ 23. '功率': 'sum', 24. '拟合功率': 'sum', 25. '修复功率': 'sum', 26. '累计发电量kwh': 'last', 27. '修复累计发电量': 'last', 28. '修复累计发电量（每日归零）': 'last' 29. }).reset\_index() 30. *# 输出文件名（不覆盖）* 31. new\_filename = file.replace("修复结果.xlsx", "每小时汇总.xlsx") 32. hourly\_df.to\_excel(new\_filename, index=False) 33. print(f"✅ 已保存：{new\_filename}") 34. except Exception as e: 35. print(f"❌ 处理失败：{file}，错误：{e}") |

|  |
| --- |
| **附录** **4** 环境检测仪数据缺失补充（python代码） |
| A1\_1-4天气小时处理.py |
| 1. import pandas as pd 2. import numpy as np 3. from scipy.optimize import curve\_fit 4. import matplotlib.pyplot as plt 5. plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] 6. plt.rcParams['axes.unicode\_minus'] = False 7. *# 高斯模型* 8. def gaussian(x, A, mu, sigma): 9. return A \* np.exp(- ((x - mu) \*\* 2) / (2 \* sigma \*\* 2)) 10. def repair\_radiation\_data(input\_file, output\_file): 11. df = pd.read\_excel(input\_file) 12. df.columns = ['时间', '辐照强度'] 13. df['时间'] = pd.to\_datetime(df['时间']).dt.floor('min') 14. df = df.drop\_duplicates('时间').set\_index('时间').sort\_index() 15. *# 重建完整时间索引（按分钟）* 16. full\_index = pd.date\_range(df.index.min(), df.index.max(), freq='1min') 17. df = df.reindex(full\_index) 18. df.index.name = '时间' 19. df['原始'] = df['辐照强度'].notna() 20. days = pd.unique(df.index.date) 21. all\_dfs = [] 22. for day in days: 23. day\_df = df.loc[str(day)].copy() 24. day\_df['分钟'] = (day\_df.index - day\_df.index.normalize()).total\_seconds() / 60 25. valid\_data = day\_df.loc[day\_df['辐照强度'].notna(), ['分钟', '辐照强度']] 26. if len(valid\_data) >= 10: 27. try: 28. x = valid\_data['分钟'].values 29. y = valid\_data['辐照强度'].values 30. p0 = [y.max(), x[np.argmax(y)], (x.max() - x.min()) / 4] 31. popt, \_ = curve\_fit(gaussian, x, y, p0=p0, maxfev=8000) 32. day\_df['拟合值'] = gaussian(day\_df['分钟'], \*popt).clip(lower=0) 33. except Exception as e: 34. print(f"拟合失败：{day}，使用滑动平均。错误：{e}") 35. day\_df['拟合值'] = valid\_data['辐照强度'].rolling(6, min\_periods=1).mean().reindex(day\_df.index, method='nearest').fillna(0) 36. else: 37. day\_df['拟合值'] = valid\_data['辐照强度'].rolling(6, min\_periods=1).mean().reindex(day\_df.index, method='nearest').fillna(0) 38. day\_df['修复强度'] = day\_df['辐照强度'] 39. day\_df.loc[day\_df['辐照强度'].isna(), '修复强度'] = day\_df.loc[day\_df['辐照强度'].isna(), '拟合值'] 40. all\_dfs.append(day\_df) 41. final\_df = pd.concat(all\_dfs) 42. final\_df[['辐照强度', '拟合值', '修复强度', '原始']].to\_excel(output\_file) 43. *# 可视化最近一天* 44. last\_day = final\_df.index.max().normalize() 45. plot\_df = final\_df.loc[last\_day:last\_day + pd.Timedelta(days=1)] 46. plt.figure(figsize=(15, 5)) 47. plt.plot(plot\_df.index, plot\_df['辐照强度'], label='原始值', alpha=0.6) 48. plt.plot(plot\_df.index, plot\_df['修复强度'], '--', label='修复后') 49. plt.title("辐照强度拟合修复示例（最近一天）") 50. plt.xlabel("时间") 51. plt.ylabel("W/m²") 52. plt.legend() 53. plt.grid() 54. plt.tight\_layout() 55. plt.savefig("辐照强度\_修复示意图.png") 56. plt.close() 57. print("✅ 修复完成，结果已保存：", output\_file) 58. *# 执行示例* 59. repair\_radiation\_data("电站1环境检测仪数据.xlsx", "电站1\_辐照强度\_修复结果.xlsx") |

|  |
| --- |
| **附录** 6环境检测仪数据小时化处理（python代码） |
| A1\_1.py |
| 1. import pandas as pd 2. import numpy as np 3. from scipy import stats 4. from openpyxl.styles import PatternFill 5. import matplotlib.pyplot as plt 6. *# 设置中文字体支持* 7. plt.rcParams['font.sans-serif'] = ['SimHei'] 8. plt.rcParams['axes.unicode\_minus'] = False 9. *# 夜间时段定义* 10. NIGHT\_HOURS = (0, 5)  *# 0:00-5:00视为夜间* 11. NIGHT\_FILL = PatternFill(start\_color='FFF2CC', end\_color='FFF2CC', fill\_type='solid') 12. def load\_and\_clean\_data(power\_file, irrad\_file): 13. """ 14. 加载并清洗发电和辐照数据 15. :return: 清洗后的发电数据、辐照数据 16. """ 17. *# ========== 1. 发电数据处理 ==========* 18. df\_power = pd.read\_excel( 19. power\_file, 20. parse\_dates=['时间'], 21. usecols=['时间', '当日累计发电量kwh'] 22. ).rename(columns={'当日累计发电量kwh': '累计发电量'}) 23. df\_power = df\_power.sort\_values('时间').set\_index('时间') 24. hourly\_power = df\_power.resample('1h').last().ffill() 25. hourly\_power['小时发电量'] = hourly\_power.groupby(hourly\_power.index.date)['累计发电量'].diff().fillna(0) 26. hourly\_power['小时发电量'] = hourly\_power['小时发电量'].clip(lower=0) 27. *# ========== 2. 辐照数据处理 ==========* 28. df\_irrad = pd.read\_excel( 29. irrad\_file, 30. parse\_dates=['时间'], 31. usecols=['时间', '辐照强度w/m2'] 32. ).rename(columns={'辐照强度w/m2': '辐照强度'}) 33. df\_irrad = df\_irrad.set\_index('时间').resample('1h').mean() 34. *# ========== 3. 高级清洗（夜间修正数据）==========* 35. *# 处理发电数据夜间值* 36. night\_mask\_power = (hourly\_power.index.hour >= NIGHT\_HOURS[0]) & (hourly\_power.index.hour < NIGHT\_HOURS[1]) 37. hourly\_power.loc[night\_mask\_power, '小时发电量'] = 0 38. *# 处理辐照数据夜间值* 39. night\_mask\_irrad = (df\_irrad.index.hour >= NIGHT\_HOURS[0]) & (df\_irrad.index.hour < NIGHT\_HOURS[1]) 40. df\_irrad.loc[night\_mask\_irrad, '辐照强度'] = 0 41. return hourly\_power, df\_irrad 42. def export\_to\_excel(df\_power, df\_irrad, output\_power\_path, output\_irrad\_path): 43. """数据导出到Excel""" 44. *# 发电数据导出* 45. if not df\_power.empty: 46. df\_power.to\_excel(output\_power\_path, index=True) 47. print(f"发电数据已保存到 {output\_power\_path}") 48. else: 49. print("警告：发电数据为空，未生成文件") 50. *# 辐照数据导出* 51. if not df\_irrad.empty: 52. df\_irrad.to\_excel(output\_irrad\_path, index=True) 53. print(f"辐照数据已保存到 {output\_irrad\_path}") 54. else: 55. print("警告：辐照数据为空，未生成文件") 56. if \_\_name\_\_ == "\_\_main\_\_": 57. input\_files = { 58. 'power\_file': '电站4发电数据.xlsx', 59. 'irrad\_file': '电站4环境监测仪数据.xlsx' 60. } 61. try: 62. *# 清洗数据* 63. df\_power, df\_irrad = load\_and\_clean\_data( 64. input\_files['power\_file'], 65. input\_files['irrad\_file'] 66. ) 67. *# 导出数据* 68. export\_to\_excel( 69. df\_power, 70. df\_irrad, 71. '清洗后的发电数据.xlsx', 72. r'A1\_辐射数据\_缺失补充\电站4环境检测仪数据\_每小时汇总.xlsx' 73. ) 74. *# 生成趋势图* 75. plt.figure(figsize=(15, 5)) 76. plt.plot(df\_power.index, df\_power['小时发电量'], label='发电量') 77. plt.plot(df\_irrad.index, df\_irrad['辐照强度'], label='辐照强度', alpha=0.7) 78. plt.title('发电量与辐照强度趋势') 79. plt.legend() 80. plt.savefig('电站4趋势对比图.png') 81. plt.show() 82. except Exception as e: 83. print(f"处理失败: {str(e)}") |

|  |
| --- |
| **附录** 7 天气重复值处理（python代码） |
| A1\_天气\_L.py |
| 1. import pandas as pd 2. import numpy as np 3. import os 4. def process\_group(group): 5. """处理时间重复的分组数据（增强版）""" 6. if len(group) == 1: 7. return group.iloc[0] 8. *# 列分类定义* 9. mean\_cols = ["最高温度", "最低温度", "风速", "湿度"] 10. mode\_cols = ["当前温度", "天气", "风向"] 11. sun\_cols = ["日出时间", "日落时间"] 12. *# ====== 均值列处理 ======* 13. try: 14. *# 计算初始均值（至少需要1个有效值）* 15. initial\_mean = group[mean\_cols].mean(skipna=True) 16. *# 填充缺失值并计算距离* 17. filled\_data = group[mean\_cols].fillna(initial\_mean) 18. distances = (filled\_data - initial\_mean).abs().sum(axis=1) 19. *# 删除距离最大的行（至少保留1条）* 20. filtered\_group = group.drop(distances.idxmax()) 21. *# 计算最终均值（允许全空值）* 22. final\_mean = filtered\_group[mean\_cols].mean(skipna=True) 23. except Exception as e: 24. print(f"处理分组时发生错误（时间：{group.name}）: {str(e)}") 25. final\_mean = pd.Series([np.nan] \* len(mean\_cols), index=mean\_cols) 26. *# ====== 众数列处理 ======* 27. mode\_values = {} 28. for col in mode\_cols: 29. try: 30. *# 使用原始数据计算众数（非处理后的数据）* 31. non\_null = group[col].dropna() 32. if len(non\_null) > 0: 33. mode\_val = non\_null.mode() 34. mode\_values[col] = mode\_val[0] if not mode\_val.empty else np.nan 35. else: 36. mode\_values[col] = np.nan 37. except: 38. mode\_values[col] = np.nan 39. *# ====== 日出日落时间处理 ======* 40. sun\_values = {} 41. for col in sun\_cols: 42. try: 43. *# 优先使用处理后的数据，其次用原始数据* 44. non\_null = filtered\_group[col].dropna() 45. if non\_null.empty: 46. non\_null = group[col].dropna() 47. sun\_values[col] = non\_null.iloc[0] if not non\_null.empty else np.nan 48. except: 49. sun\_values[col] = np.nan 50. return pd.Series({ 51. \*\*final\_mean.to\_dict(), 52. \*\*mode\_values, 53. \*\*sun\_values, 54. "时间": group.name 55. }) 56. *# ====== 主处理流程 ======* 57. if \_\_name\_\_ == "\_\_main\_\_": 58. *# 读取原始数据* 59. input\_path = "电站4天气数据.xlsx" 60. origin\_df = pd.read\_excel(input\_path) 61. *# 时间格式处理* 62. origin\_df["时间"] = pd.to\_datetime(origin\_df["时间"], errors="coerce") 63. *# 步骤1：数据拆分* 64. duplicate\_mask = origin\_df.duplicated("时间", keep=False) 65. duplicate\_df = origin\_df[duplicate\_mask].sort\_values("时间") 66. unique\_df = origin\_df[~duplicate\_mask] 67. *# 步骤2：处理重复数据* 68. processed\_data = [] 69. if not duplicate\_df.empty: 70. for time, group in duplicate\_df.groupby("时间"): 71. try: 72. processed\_row = process\_group(group) 73. processed\_data.append(processed\_row) 74. except Exception as e: 75. print(f"处理时间 {time} 时发生严重错误: {str(e)}") 76. continue 77. *# 创建结果DataFrame* 78. processed\_df = pd.DataFrame(processed\_data, columns=origin\_df.columns) 79. *# 步骤3：合并数据* 80. final\_df = pd.concat([unique\_df, processed\_df], axis=0) 81. *# 最终排序去重（处理后的数据优先）* 82. final\_df = final\_df.sort\_values("时间", ascending=True) 83. final\_df = final\_df.drop\_duplicates("时间", keep="first") 84. *# 创建输出目录* 85. output\_dir = "A1\_天气数据\_重复值整合" 86. os.makedirs(output\_dir, exist\_ok=True) 87. *# 保存结果（保留原始格式）* 88. output\_path = os.path.join(output\_dir, "电站4天气数据整合.xlsx") 89. final\_df.to\_excel(output\_path, index=False, engine="openpyxl") 90. *# 打印验证信息* 91. print("\n" + "=" \* 40) 92. print(f"原始数据记录数: {len(origin\_df)}") 93. print(f"发现重复时间数: {len(duplicate\_df.groupby('时间'))}") 94. print(f"最终有效记录数: {len(final\_df)}") 95. print(f"已清除重复记录: {len(origin\_df) - len(final\_df)}") 96. print(f"结果文件已保存至: {os.path.abspath(output\_path)}") 97. print("=" \* 40) |

|  |
| --- |
| **附录** 8天气小时化处理（python代码） |
| A1\_1-4天气小时处理.py |
| 1. import pandas as pd 2. import numpy as np 3. def process\_weather\_hourly(file\_path): 4. *# 读取数据* 5. df = pd.read\_excel(file\_path, parse\_dates=['时间']) 6. *# 保留所需列* 7. df = df[['时间', '当前温度', '最高温度', '最低温度', '天气', '风向', '风速', '湿度', '日出时间', '日落时间']] 8. *# 提取日期* 9. df['日期'] = df['时间'].dt.date 10. *# 提取每个日期中第一次出现的日出/日落时间* 11. sunrise\_sunset = df.groupby('日期')[['日出时间', '日落时间']].first().reset\_index() 12. *# 时间四舍五入到整点（分钟 >=30 进1小时）* 13. df['时间'] = df['时间'].dt.round('h') 14. *# 去重：防止同一小时有多条记录* 15. df = df.drop\_duplicates('时间').set\_index('时间') 16. *# 构造完整小时时间索引* 17. time\_range = pd.date\_range(start=df.index.min(), end=df.index.max(), freq='1h') 18. df\_full = df.reindex(time\_range) 19. *# 数值列插值* 20. numeric\_cols = ['当前温度', '最高温度', '最低温度', '风速', '湿度'] 21. df\_full[numeric\_cols] = df\_full[numeric\_cols].interpolate(method='linear') 22. *# 类别列用前向填充* 23. df\_full['天气'] = df\_full['天气'].ffill() 24. df\_full['风向'] = df\_full['风向'].ffill() 25. *# 重置索引* 26. df\_full.reset\_index(inplace=True) 27. df\_full.rename(columns={'index': '时间'}, inplace=True) 28. *# 添加日期列用于合并* 29. df\_full['日期'] = df\_full['时间'].dt.date 30. *# 合并日出/日落时间* 31. df\_full = df\_full.merge(sunrise\_sunset, on='日期', how='left') 32. *# 删除辅助列* 33. df\_full.drop(columns=['日期'], inplace=True) 34. return df\_full 35. *# 使用示例* 36. result = process\_weather\_hourly('电站4天气数据整合.xlsx') 37. result.to\_excel('电站4天气数据整合\_每小时汇总.xlsx', index=False) 38. print("整理完成，已保存为 '电站4天气数据整合\_每小时汇总.xlsx'") |

|  |
| --- |
| **附录** 9问题二求解（python代码） |
| A2.py |
| 1. import os 2. import pandas as pd 3. import numpy as np 4. import matplotlib.pyplot as plt 5. from scipy.optimize import curve\_fit 6. from datetime import timedelta 7. plt.rcParams['font.sans-serif'] = ['SimHei'] 8. plt.rcParams['axes.unicode\_minus'] = False 9. *# 全局配置* 10. stations = { 11. '电站1': {'capacity': 4998.30, 'tilt': 15, 'clean\_date': '2024-12-05'}, 12. '电站2': {'capacity': 5581.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 13. '电站3': {'capacity': 4456.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 14. '电站4': {'capacity': 1794.61, 'tilt': 0, 'clean\_date': '2025-01-02'} 15. } 16. def compute\_theoretical\_power(row, eta, alpha1, beta, tref, h0, h1, ifactor): 17. G = row['辐照强度'] 18. T\_env = row['环境温度'] 19. H = row['湿度'] 20. W = row['风速'] 21. if G <= 0: 22. return 0 23. T\_back = T\_env + G / (h0 + h1 \* W) 24. P\_pred = eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta \* H) \* ifactor 25. return max(P\_pred, 0) 26. def sliding\_statistics(series, window=168): 27. return series.rolling(window=window, min\_periods=1).mean(), series.rolling(window=window, min\_periods=1).std() 28. def process\_station(station\_name, config): 29. print(f"\n>>> 正在处理：{station\_name}") 30. *# 文件路径* 31. power\_file = f"A2/A2\_发电数据/{station\_name}发电数据\_每小时汇总.xlsx" 32. env\_file = f"A2/A2\_辐射数据/{station\_name}环境检测仪数据\_每小时汇总.xlsx" 33. weather\_file = f"A2/A2\_天气数据/{station\_name}天气数据整合\_每小时汇总.xlsx" 34. *# 读取数据* 35. power\_df = pd.read\_excel(power\_file, parse\_dates=['时间'])[['时间', '修复累计发电量（每日归零）']].rename( 36. columns={'修复累计发电量（每日归零）': '累计发电量'}) 37. env\_df = pd.read\_excel(env\_file, parse\_dates=['时间'])[['时间', '辐照强度']] 38. weather\_df = pd.read\_excel(weather\_file, parse\_dates=['时间'])[['时间', '当前温度', '风速', '湿度']].rename( 39. columns={'当前温度': '环境温度'}) 40. *# 合并数据* 41. df = power\_df.merge(env\_df, on='时间').merge(weather\_df, on='时间') 42. df = df.set\_index('时间').sort\_index() 43. df['实际功率'] = df['累计发电量'].diff().fillna(0).clip(lower=0) 44. *# 参数* 45. cap = config['capacity'] 46. beta = config['tilt'] 47. area = cap / 0.18 / 1000 48. eta = cap / area / 1000 49. tref = 25 50. alpha1 = 0.004 51. beta\_h = 0.02 52. k = 0.2 if beta > 0 else 0 53. ifactor = 1 - k \* np.sin(np.radians(beta)) 54. clean\_date = pd.to\_datetime(config['clean\_date']) 55. *# 拟合模型* 56. train\_df = df[(df.index.date >= clean\_date.date()) & (df.index.date <= (clean\_date + timedelta(days=2)).date())] 57. valid = train\_df[(train\_df['辐照强度'] > 200) & (train\_df['实际功率'] > 0)] 58. def model\_func(xdata, h0, h1): 59. G, T\_env, H, W = xdata 60. T\_back = T\_env + G / (h0 + h1 \* W) 61. return eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta\_h \* H) \* ifactor 62. xdata = np.array([valid['辐照强度'], valid['环境温度'], valid['湿度'], valid['风速']]) 63. ydata = valid['实际功率'].values 64. popt, \_ = curve\_fit(model\_func, xdata, ydata, p0=[20, 5], maxfev=10000) 65. h0\_fit, h1\_fit = popt 66. *# 理论功率与DI* 67. df['理论功率'] = df.apply( 68. lambda row: compute\_theoretical\_power(row, eta, alpha1, beta\_h, tref, h0\_fit, h1\_fit, ifactor), axis=1) 69. df['DI'] = (df['理论功率'] - df['实际功率']) / df['理论功率'].replace(0, np.nan) \* 100 70. df['DI'] = df['DI'].clip(lower=0) 71. df['DI均值'], df['DI波动'] = sliding\_statistics(df['DI']) 72. *# 预警判断* 73. theta1, theta2 = (3, 4) if station\_name == '电站1' else (10, 5) 74. df['预警'] = (df['DI均值'] > theta1) & (df['DI波动'] < theta2) 75. *# 保存结果* 76. df.reset\_index().to\_excel(f"A2/ANS/{station\_name}\_DI分析结果.xlsx", index=False) 77. *# 绘图* 78. plt.figure(figsize=(12, 6)) 79. plt.plot(df.index, df['DI'], label='DI(%)', alpha=0.4) 80. plt.plot(df.index, df['DI均值'], label='DI均值', color='blue') 81. plt.axhline(theta1, color='red', linestyle='--', label='预警阈值') 82. plt.title(f"{station\_name} 积灰指数趋势") 83. plt.xlabel("时间") 84. plt.ylabel("DI (%)") 85. plt.legend() 86. *#plt.tight\_layout()* 87. if station\_name == '电站4': 88. plt.legend(loc='upper right') 89. else: 90. plt.tight\_layout() 91. plt.savefig(f"A2/ANS/{station\_name}\_DI趋势图.png") 92. plt.show() 93. plt.close() 94. print(f">>> 处理完成：{station\_name}") 95. *# 主程序* 96. for name, config in stations.items(): 97. process\_station(name, config) |

|  |
| --- |
| **附录** 10问题三第一小问求解（python代码） |
| A3\_1清洗节点决策.py |
| 1. import os 2. import pandas as pd 3. import numpy as np 4. import matplotlib.pyplot as plt 5. from scipy.optimize import curve\_fit 6. from datetime import timedelta 7. plt.rcParams['font.sans-serif'] = ['SimHei'] 8. plt.rcParams['axes.unicode\_minus'] = False 9. *# 全局配置* 10. stations = { 11. '电站1': {'capacity': 4998.30, 'tilt': 15, 'clean\_date': '2024-12-05'}, 12. '电站2': {'capacity': 5581.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 13. '电站3': {'capacity': 4456.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 14. '电站4': {'capacity': 1794.61, 'tilt': 0, 'clean\_date': '2025-01-02'} 15. } 16. def compute\_theoretical\_power(row, eta, alpha1, beta, tref, h0, h1, ifactor): 17. G = row['辐照强度'] 18. T\_env = row['环境温度'] 19. H = row['湿度'] 20. W = row['风速'] 21. if G <= 0: 22. return 0 23. T\_back = T\_env + G / (h0 + h1 \* W) 24. P\_pred = eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta \* H) \* ifactor 25. return max(P\_pred, 0) 26. def sliding\_statistics(series, window=168): 27. return series.rolling(window=window, min\_periods=1).mean(), series.rolling(window=window, min\_periods=1).std() 28. def dynamic\_cleaning\_decision(df, cap, p\_electric=0.582, theta\_safe=20): 29. C\_clean = 2 \* cap 30. df = df.copy() 31. df['发电损失'] = df['DI'] / 100 \* df['实际功率'] \* p\_electric 32. df['累积损失'] = df['发电损失'].cumsum() 33. df['清洗建议'] = False 34. if (df['累积损失'] >= C\_clean).any(): 35. t\_clean = df[df['累积损失'] >= C\_clean].index[0] 36. df.loc[t\_clean:, '清洗建议'] = True 37. elif (df['DI均值'] > theta\_safe).any(): 38. t\_clean = df[df['DI均值'] > theta\_safe].index[0] 39. df.loc[t\_clean:, '清洗建议'] = True 40. return df 41. def process\_station(station\_name, config): 42. print(f"\n>>> 正在处理：{station\_name}") 43. *# 文件路径* 44. power\_file = f"A2/A2\_发电数据/{station\_name}发电数据\_每小时汇总.xlsx" 45. env\_file = f"A2/A2\_辐射数据/{station\_name}环境检测仪数据\_每小时汇总.xlsx" 46. weather\_file = f"A2/A2\_天气数据/{station\_name}天气数据整合\_每小时汇总.xlsx" 47. *# 读取数据* 48. power\_df = pd.read\_excel(power\_file, parse\_dates=['时间'])[['时间', '修复累计发电量（每日归零）']].rename( 49. columns={'修复累计发电量（每日归零）': '累计发电量'}) 50. env\_df = pd.read\_excel(env\_file, parse\_dates=['时间'])[['时间', '辐照强度']] 51. weather\_df = pd.read\_excel(weather\_file, parse\_dates=['时间'])[['时间', '当前温度', '风速', '湿度']].rename( 52. columns={'当前温度': '环境温度'}) 53. *# 合并数据* 54. df = power\_df.merge(env\_df, on='时间').merge(weather\_df, on='时间') 55. df = df.set\_index('时间').sort\_index() 56. df['实际功率'] = df['累计发电量'].diff().fillna(0).clip(lower=0) 57. *# 参数* 58. cap = config['capacity'] 59. beta = config['tilt'] 60. area = cap / 0.18 / 1000 61. eta = cap / area / 1000 62. tref = 25 63. alpha1 = 0.004 64. beta\_h = 0.02 65. k = 0.2 if beta > 0 else 0 66. ifactor = 1 - k \* np.sin(np.radians(beta)) 67. clean\_date = pd.to\_datetime(config['clean\_date']) 68. *# 拟合模型* 69. train\_df = df[(df.index.date >= clean\_date.date()) & (df.index.date <= (clean\_date + timedelta(days=2)).date())] 70. valid = train\_df[(train\_df['辐照强度'] > 200) & (train\_df['实际功率'] > 0)] 71. def model\_func(xdata, h0, h1): 72. G, T\_env, H, W = xdata 73. T\_back = T\_env + G / (h0 + h1 \* W) 74. return eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta\_h \* H) \* ifactor 75. xdata = np.array([valid['辐照强度'], valid['环境温度'], valid['湿度'], valid['风速']]) 76. ydata = valid['实际功率'].values 77. popt, \_ = curve\_fit(model\_func, xdata, ydata, p0=[20, 5], maxfev=10000) 78. h0\_fit, h1\_fit = popt 79. *# 理论功率与DI* 80. df['理论功率'] = df.apply(lambda row: compute\_theoretical\_power(row, eta, alpha1, beta\_h, tref, h0\_fit, h1\_fit, ifactor), axis=1) 81. df['DI'] = (df['理论功率'] - df['实际功率']) / df['理论功率'].replace(0, np.nan) \* 100 82. df['DI'] = df['DI'].clip(lower=0) 83. df['DI均值'], df['DI波动'] = sliding\_statistics(df['DI']) 84. *# 添加清洗时间决策建议* 85. df = dynamic\_cleaning\_decision(df, cap) 86. *# 输出结果* 87. os.makedirs("A3/ANS\_1", exist\_ok=True) 88. df.reset\_index().to\_excel(f"A3/ANS\_1/{station\_name}\_DI分析结果\_含清洗建议.xlsx", index=False) 89. *# 绘图* 90. plt.figure(figsize=(12, 6)) 91. plt.plot(df.index, df['DI'], label='DI(%)', alpha=0.4) 92. plt.plot(df.index, df['DI均值'], label='DI均值', color='blue') 93. plt.axhline(20, color='green', linestyle='--', label='安全预警阈值20%') 94. plt.fill\_between(df.index, 0, df['DI'], where=df['清洗建议'], color='red', alpha=0.3, label='清洗建议时段') 95. plt.title(f"{station\_name} 积灰指数趋势及清洗建议") 96. plt.xlabel("时间") 97. plt.ylabel("DI (%)") 98. plt.legend() 99. plt.tight\_layout() 100. plt.savefig(f"A3/ANS\_1/{station\_name}\_清洗决策趋势图.png") 101. plt.close() 102. print(f">>> 处理完成：{station\_name}") 103. *# 主程序* 104. for name, config in stations.items(): 105. process\_station(name, config) |

|  |
| --- |
| **附录** 11问题三第二小问求解（python代码） |
| A3\_清洗价格对清洗决策影响.py |
| 1. import os 2. import pandas as pd 3. import numpy as np 4. import matplotlib.pyplot as plt 5. from scipy.optimize import curve\_fit 6. from datetime import timedelta 7. plt.rcParams['font.sans-serif'] = ['SimHei'] 8. plt.rcParams['axes.unicode\_minus'] = False 9. *# 全局配置* 10. stations = { 11. '电站1': {'capacity': 4998.30, 'tilt': 15, 'clean\_date': '2024-12-05'}, 12. '电站2': {'capacity': 5581.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 13. '电站3': {'capacity': 4456.00, 'tilt': 0, 'clean\_date': '2025-01-02'}, 14. '电站4': {'capacity': 1794.61, 'tilt': 0, 'clean\_date': '2025-01-02'} 15. } 16. def compute\_theoretical\_power(row, eta, alpha1, beta, tref, h0, h1, ifactor): 17. G = row['辐照强度'] 18. T\_env = row['环境温度'] 19. H = row['湿度'] 20. W = row['风速'] 21. if G <= 0: 22. return 0 23. T\_back = T\_env + G / (h0 + h1 \* W) 24. P\_pred = eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta \* H) \* ifactor 25. return max(P\_pred, 0) 26. def sliding\_statistics(series, window=168): 27. return series.rolling(window=window, min\_periods=1).mean(), series.rolling(window=window, min\_periods=1).std() 28. def dynamic\_cleaning\_count(df, cap, price\_list, p\_electric=0.582, theta\_safe=20, cooldown\_hours=72): 29. result = {} 30. for price in price\_list: 31. C\_clean = price \* cap 32. *#print(C\_clean)* 33. cleaned\_times = 0 34. acc\_loss = 0 35. last\_clean\_time = None 36. df\_copy = df.copy() 37. df\_copy['发电损失'] = df\_copy['DI'] / 100 \* df\_copy['实际功率'] \* p\_electric 38. df\_copy['清洗建议'] = False 39. for t in df\_copy.index: 40. *# 冷却期判断 3天内不再清洗* 41. if last\_clean\_time is not None and (t - last\_clean\_time).total\_seconds() < cooldown\_hours \* 3600: 42. continue 43. acc\_loss += df\_copy.at[t, '发电损失'] 44. di\_mean = df\_copy.at[t, 'DI均值'] 45. triggered = False 46. if acc\_loss >= C\_clean: 47. triggered = True 48. elif di\_mean > theta\_safe: 49. triggered = True 50. if triggered: 51. df\_copy.at[t, '清洗建议'] = True 52. cleaned\_times += 1 53. acc\_loss = 0 54. last\_clean\_time = t  *# 设置当前清洗时间，启动冷却期* 55. result[round(price, 1)] = cleaned\_times 56. return result 57. def process\_station(station\_name, config): 58. print(f"\n>>> 正在处理：{station\_name}") 59. *# 文件路径* 60. power\_file = f"A2/A2\_发电数据/{station\_name}发电数据\_每小时汇总.xlsx" 61. env\_file = f"A2/A2\_辐射数据/{station\_name}环境检测仪数据\_每小时汇总.xlsx" 62. weather\_file = f"A2/A2\_天气数据/{station\_name}天气数据整合\_每小时汇总.xlsx" 63. *# 读取数据* 64. power\_df = pd.read\_excel(power\_file, parse\_dates=['时间'])[['时间', '修复累计发电量（每日归零）']].rename( 65. columns={'修复累计发电量（每日归零）': '累计发电量'}) 66. env\_df = pd.read\_excel(env\_file, parse\_dates=['时间'])[['时间', '辐照强度']] 67. weather\_df = pd.read\_excel(weather\_file, parse\_dates=['时间'])[['时间', '当前温度', '风速', '湿度']].rename( 68. columns={'当前温度': '环境温度'}) 69. *# 合并数据* 70. df = power\_df.merge(env\_df, on='时间').merge(weather\_df, on='时间') 71. df = df.set\_index('时间').sort\_index() 72. df['实际功率'] = df['累计发电量'].diff().fillna(0).clip(lower=0) 73. *# 参数* 74. cap = config['capacity'] 75. beta = config['tilt'] 76. area = cap / 0.18 / 1000 77. eta = cap / area / 1000 78. tref = 25 79. alpha1 = 0.004 80. beta\_h = 0.02 81. k = 0.2 if beta > 0 else 0 82. ifactor = 1 - k \* np.sin(np.radians(beta)) 83. clean\_date = pd.to\_datetime(config['clean\_date']) 84. *# 拟合模型* 85. train\_df = df[(df.index.date >= clean\_date.date()) & (df.index.date <= (clean\_date + timedelta(days=2)).date())] 86. valid = train\_df[(train\_df['辐照强度'] > 200) & (train\_df['实际功率'] > 0)] 87. def model\_func(xdata, h0, h1): 88. G, T\_env, H, W = xdata 89. T\_back = T\_env + G / (h0 + h1 \* W) 90. return eta \* G \* (1 - alpha1 \* (T\_back - tref)) \* (1 - beta\_h \* H) \* ifactor 91. xdata = np.array([valid['辐照强度'], valid['环境温度'], valid['湿度'], valid['风速']]) 92. ydata = valid['实际功率'].values 93. popt, \_ = curve\_fit(model\_func, xdata, ydata, p0=[20, 5], maxfev=10000) 94. h0\_fit, h1\_fit = popt 95. *# DI计算* 96. df['理论功率'] = df.apply(lambda row: compute\_theoretical\_power(row, eta, alpha1, beta\_h, tref, h0\_fit, h1\_fit, ifactor), axis=1) 97. df['DI'] = (df['理论功率'] - df['实际功率']) / df['理论功率'].replace(0, np.nan) \* 100 98. df['DI'] = df['DI'].clip(lower=0) 99. df['DI均值'], df['DI波动'] = sliding\_statistics(df['DI']) 100. *# 模拟不同价格下的清洗次数* 101. price\_range = np.arange(0.1, 5.1, 0.1)  *#* 102. result = dynamic\_cleaning\_count(df, cap, price\_range) 103. *# 保存CSV* 104. os.makedirs("A3/ANS\_2", exist\_ok=True) 105. pd.DataFrame(list(result.items()), columns=['清洗价格(元/kW)', '清洗次数']).to\_excel(f"A3/ANS\_2/{station\_name}\_价格清洗次数统计.xlsx", index=False) 106. *# 绘图* 107. plt.figure(figsize=(10, 5)) 108. plt.plot(list(result.keys()), list(result.values()), marker='o', label=station\_name) 109. plt.title(f"{station\_name} 不同清洗价格下清洗次数变化趋势") 110. plt.xlabel("清洗价格 (元/kW)") 111. plt.ylabel("清洗次数") 112. plt.grid(True) 113. plt.tight\_layout() 114. plt.savefig(f"A3/ANS\_2/{station\_name}\_清洗价格影响趋势图.png") 115. plt.close() 116. *# 主程序* 117. for name, config in stations.items(): 118. process\_station(name, config) |