A 题 光伏板积灰程度检测及灰尘清洗策略

光伏电站在运行过程中，由于环境因素，光伏板可能出现积灰等情况，造成发电效率降低，通过运维数据及时检测出积灰状况，并加以干预，对提高发电效率提升电站的经济效益具有重要意义。由于影响发电效率的因素很多，包括环境温度、背板温度、天气状况以及季节性变化等，这些均是不可控因素，而积灰对发电效率的影响是可人工干预的，而发电效率的高低受这些因素的综合影响，为此，需要分析出灰尘的影响指数。

附件给出了 4 个电站实时采集到的发电量数据、现场辐照数据以及当地气象数据。建立数学模型，解决以下问题。

问题 1

对附件中的数据进行清洗（不排除存在个别缺失、异常的数据），给出具体的算法，并按时间刻度（一个小时）对所需的指标数据进行整理。

问题 1 分析

在问题 1 中，核心任务是对提供的光伏电站发电量、辐照强度、天气等数据进行清洗与整理，以确保后续分析能够基于准确、完整的数据进行。首先，数据的缺失和异常值需要通过适当的填补方法进行处理，常见的方法包括均值填充、插值法等。其次，为了进行后续的分析，需要对不同时间段的数据进行统一的重采样（例如按小时进行重采样），确保所有数据的时间刻度一致。异常值检测可以使用统计方法如 z-score 或 IQR 来识别和处理不合理的数据点。最终，清洗后的数据将作为后续建模的基础，确保数据的质量不会对建模结果造成负面影响。通过这些步骤，我们可以得到一个格式统一、缺失值填补完备的清洁数据集，为问题 2 和问题 3 的进一步建模提供基础。

解题思路：

详细建模过程：

1. **数据读取与初步检查：**

通过读取各个电站的数据文件（发电量、天气数据、环境监测数据等），我们首先需要检查数据的结构与内容，确保所有的列名、数据类型和时间格式一致。

逐个检查数据中的缺失值、重复数据以及异常值（如负值或超出合理范围的值）。

2. **缺失值处理：**

缺失值可以通过多种方式处理，常见的方法包括：

均值或中位数填补：如果数据中某一列缺失较多，可以使用该列的均值或中位数进行填补。

插值法：对于时间序列数据，通常使用线性插值法填补缺失值，以保持数据的连续性。

前后数据填充：对于缺失的时间点，可以使用前后时间点的数据填充，确保数据的连贯性。

3. **异常值检测与处理：**

使用**统计方法**（如 z-score、IQR）检测异常值。例如，设定一个合理的标准差范围，任何超出这个范围的数据都可以视为异常值进行修正或删除。
对于发电量和辐照强度等变量，通常会有合理的区间范围，可以根据电站的装机容量、历史数据等进行限定。

4. **时间同步与重采样：**

所有数据必须基于相同的时间轴进行对齐。通常数据会按天或更短的时间间隔采集（例如每小时），需要确保数据的时间戳一致。
使用 Pandas 的 resample()函数，将所有数据按小时重采样，并对缺失的小时进行填充。

5. **数据整合与输出：**

对每个电站的数据进行整理，将不同来源（如发电量、天气、环境监测数据等）的数据合并在一起，形成统一的时间序列数据。
输出清洗后的数据文件，供后续分析使用。

数学公式与模型：

1. **缺失值填补**（均值填补）：

对于每个缺失的数据点 x_i，使用该列的均值 μ_x 填补： $x_i = \mu_x$

2. **异常值检测：**

使用 z-score 方法来检测异常值。若某数据点的 z-score 超出 3，则可以认为是异常值：

$z_i = \frac{x_i - \mu_x}{\sigma_x}$。其中，$x_i$ 是数据点，μ_x 是该列数据的均值，σ_x 是标准差。

3. **插值法：**

对于缺失值，使用线性插值法填补。若时间序列中某时刻数据缺失，可以根据前后的数据进行插值填补：$x_t = x_{t-1} + \frac{(x_t - 1 - x_{t-1})}{2}$

4. **重采样：**

对时间序列数据进行重采样，假设数据以小时为单位，使用 resample()方法进行按小时的平均计算： $X_{\text{hour}} = \frac{1}{N}\sum_{i=1}^{N} x_i$

其中，N 为每小时的采样数，x_i 为每个时刻的观测值。

Python 代码：

```
import pandas as pd
import numpy as np

# 读取数据文件
df1 = pd.read_excel('电站 1 发电数据.xlsx')
df2 = pd.read_excel('电站 2 发电数据.xlsx')
df3 = pd.read_excel('电站 3 发电数据.xlsx')
df4 = pd.read_excel('电站 4 发电数据.xlsx')
```

```
# 检查数据缺失情况
print(df1.isnull().sum())

# 1. 缺失值处理 - 使用均值填补
df1.fillna(df1.mean(), inplace=True)

# 2. 异常值检测 - 使用 z-score 方法
from scipy.stats import zscore

df1['z_score'] = zscore(df1['generation'])  # 计算发电量的 z-score
df1.loc[df1['z_score'].abs() > 3, 'generation'] = np.nan  # 将异常
值替换为 NaN

# 3. 使用插值法填补缺失值
df1['generation'] = df1['generation'].interpolate(method='linear')

# 4. 时间重采样 - 按小时进行聚合
df1['time'] = pd.to_datetime(df1['time'])
df1 = df1.set_index('time').resample('H').mean().reset_index()

# 5. 输出清洗后的数据
df1.to_excel('清洗后的电站 1 数据.xlsx', index=False)

# 对其他电站数据重复相同的清洗操作
```

问题 2

因各种原因，辐照仪的度数与真实的辐照强度不一定完全一致（整体上可能存在一定程度的缩放），此外，还会受表面污染等影响使度数发生异常，常用的衡量发电效率的指标（比如 PR 值）可能会出现大幅波动，PR 值难以准确反映积灰程度。请构建衡量光伏板积灰程度的指标，通过该指标分析出历史清洗过的时间节点，并给出实时的清洗预警规则。

问题 2 分析

问题 2 的核心目标是根据辐照强度和实际发电量之间的关系构建一个积灰程度的量化指标。由于积灰会导致发电效率下降，首先需要通过回归分析建立理想发电量与辐照强度之间的关系，进而推测积灰对发电量的影响。通过计算理想发电量和实际发电量之间的差异，得出积灰指数（DI），作为反映积灰程度的主要指标。接下来，我们通过分析历史数据中清洗前后的发电量变化，识别清洗的时间节点和积灰的变化。为了实现实时的清洗预警，基于积灰指数构建一个阈值模型，当 DI 值超过设定的阈值时，触发清洗预警。此外，为了优化预警规则，我们可以引入机器学习算法（如决策树或随机森林）来学习积灰指数与清洗的关系，动态调整预警阈值，从而提高清洗决策的准确性和实时性。通过这些方法，能够有效预测积灰程度，确保光伏电站在积灰严重时及时进行清洗，避免发电效率的浪

费。

解题思路：

详细建模过程：

1. **积灰程度指标的构建：** 积灰程度的度量可以通过分析辐照强度与实际发电量之间的差异来实现。光伏发电的理想状态是发电量与辐照强度成正比，然而，由于积灰的影响，实际发电量通常会低于理想状态。因此，我们可以通过以下步骤来构建积灰程度的衡量指标：

   步骤 1：建立基准模型： 使用回归分析（如线性回归）来建立辐照强度与发电量之间的理想关系。假设发电量 P_{ideal} 与辐照强度 I 存在以下线性关系：

   $P_{\text{ideal}} = \alpha I + \beta$。其中，$\alpha$ 和 β 是回归系数。通过历史数据拟合这个模型，得到辐照强度与发电量的理论关系。

   步骤 2：计算实际发电量与理想发电量之间的差异： 根据历史数据计算实际发电量 P_{actual} 与预测的理想发电量 P_{ideal} 之间的差异，该差异可以作为积灰程度的初步指标。定义积灰指数（Dust Index, DI）为：$DI = \frac{P_{\text{ideal}} - P_{\text{actual}}}{P_{\text{ideal}}}$。这里，$P_{ideal}$ 为根据辐照强度预测的理想发电量，P_{actual} 为实际发电量。DI 越大，表示积灰程度越严重。

2. **清洗时间节点的历史分析：** 通过对历史数据进行分析，我们可以识别出清洗前后积灰程度的变化。特别是通过回顾清洗过的时间节点（如电站 1 在 2024 年 12 月 5 日清洗过），比较清洗前后的发电量变化，可以帮助我们推测清洗的有效性。

   步骤 3：清洗前后积灰变化的分析： 计算清洗前后的 DI 值，观察清洗对积灰程度的影响。如果清洗后 DI 值显著下降，说明清洗起到了有效的作用。

3. **实时清洗预警规则的构建：** 基于积灰程度指标（DI），我们可以构建一个实时清洗预警系统。当积灰指数超过某个阈值时，触发清洗预警。该阈值可以通过历史数据分析得到，也可以通过机器学习算法进行优化。

   步骤 4：清洗预警规则的建立： 设定一个积灰指数的阈值 $DI_{threshold}$，当当前 DI 值超过该阈值时，系统触发清洗预警。

$$\begin{cases} 1 & \text{if } DI > DI_{threshold} \\ 0 & \text{if } DI \le DI_{threshold} \end{cases}$$ 其中，1 表示需要清洗，0 表示不需要清洗。

   步骤 5：阈值的动态优化： 使用机器学习算法（如**决策树**、**支持向量机**或**随机森林**）来动态优化清洗预警规则。通过历史数据训练模型，根据积灰程度、天气状况、历史清洗记录等因素来预测清洗需求。

数学公式与模型：

1. **回归模型：** $P_{\text{ideal}} = \alpha I + \beta$ 其中，P_{ideal} 为理想发电量，I 为辐照强度，α 和 β 是回归系数。
2. **积灰指数（DI）：**

$$DI = \frac{P_{\text{ideal}} - P_{\text{actual}}}{P_{\text{ideal}}}$$

3. **清洗预警规则：**

$$\begin{cases} 1 & \text{if } DI > DI_{threshold} \\ 0 & \text{if } DI \le DI_{threshold} \end{cases}$$

4. **机器学习模型：** 使用决策树、随机森林等方法训练模型，输出清洗预警决策。

Python 代码：

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import numpy as np

# 读取数据
df1 = pd.read_excel('电站1发电数据.xlsx')
df1['time'] = pd.to_datetime(df1['time'])

# 1. 构建回归模型：发电量与辐照强度的关系
X = df1[['irradiance']]  # 辐照强度
y = df1['generation']    # 实际发电量

# 线性回归模型
model = LinearRegression()
model.fit(X, y)

# 计算理想发电量
df1['ideal_generation'] = model.predict(X)

# 2. 计算积灰指数 DI
df1['dust_index'] = (df1['ideal_generation'] - df1['generation']) /
df1['ideal_generation']

# 3. 历史清洗记录分析：清洗前后 DI 变化
# 假设电站 1 在 2024-12-05 清洗过
cleaning_date = pd.to_datetime('2024-12-05')
df1['cleaning'] = np.where(df1['time'] < cleaning_date, 0, 1)  # 标
记清洗后的数据

# 计算清洗前后 DI 变化
df1['di_diff'] = df1['dust_index'].diff()
```

```
# 4. 清洗预警规则
threshold = 0.1  # 假设阈值为 0.1
df1['cleaning_alert'] = np.where(df1['dust_index'] > threshold, 1,
0)

# 5. 使用机器学习优化预警规则：决策树或随机森林
X_train, X_test, y_train, y_test =
train_test_split(df1[['dust_index', 'irradiance']],
df1['cleaning_alert'], test_size=0.3, random_state=42)

rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)

# 预测清洗预警
df1['predicted_alert'] = rf_model.predict(df1[['dust_index',
'irradiance']])

# 输出清洗预警结果
df1.to_excel('清洗预警分析.xlsx', index=False)
```

问题 3

光伏板积灰会影响发电效率，进而影响电厂收益，但每次清洗需要费用，假设光伏板清洗成本市场均价约为 2 元/kW。如果清洗过于频繁，会增加电厂成本，请综合考虑这些因素，对清洗时间节点进行动态决策，并分析清洗价格对清洗决策的影响。

问题 3 分析

问题 3 的核心任务是设计一个动态决策模型，平衡清洗的成本和发电效率的提升。在这个模型中，每次清洗会产生成本，且过于频繁的清洗会导致电站的运营成本增加。因此，我们需要在考虑积灰对发电效率的影响的同时，综合分析清洗的频率与成本之间的关系。首先，建立发电效率与积灰程度之间的函数关系，假设积灰指数和发电量呈线性关系。每次清洗后，积灰程度会显著下降，发电效率恢复。为了优化清洗决策，使用强化学习中的 Q-learning 算法来进行动态决策。通过 Q-learning 模型，电站可以根据当前的积灰指数、发电量和清洗成本，学习最优的清洗策略。在训练过程中，模型会逐渐学习如何在不同的环境下选择“清洗”或“不清洗”这一动作，最大化长期的收益。最终，Q-learning 可以输出最优的清洗时间节点，平衡清洗成本和发电效率之间的关系，从而优化电站的运营收益。通过这些方法，能够实现动态调整清洗策略，在合理的成本范围内确保光伏电站的最优运行。

解题思路：

详细建模过程：

1. **清洗成本建模：** 每次清洗会产生成本，假设每次清洗的成本为 2 元/kW，且每次清洗可提升发电效率。假设电站的发电量为 $P(t)$，清洗的成本为 C，则：$C = 2 \times P(t)$ 其中，$P(t)$ 为电站在时刻 t 的发电量。
2. **发电效率与积灰的关系：** 清洗前，积灰导致的发电量下降可以通过积灰指数（DI）来表示。积灰指数 DI 越大，表示积灰越严重，发电量的下降越显著。假设发电量 $P(t)$ 与积灰指数 DI 之间存在如下关系：$P(t) = P_{\text{ideal}}(t) \times (1 - \alpha \times DI(t))$。其中，$P_{\text{ideal}}(t)$ 是理想的发电量（无积灰情况下），α 为一个常数，表示积灰对发电量的影响程度。
3. **清洗的影响：** 清洗后，积灰程度会显著下降，发电效率会恢复。假设清洗后，积灰指数减少 γ，清洗后的发电量变为：$P_{\text{after cleaning}} = P_{\text{ideal}}(t) \times (1 - \alpha \times (DI(t) - \gamma))$
4. **动态决策模型：** 我们的目标是设计一个动态决策模型，确定每个时间点是否进行清洗。动态决策问题可以用**强化学习**（Reinforcement Learning）来建模。强化学习中的**策略优化**可以帮助我们确定最优的清洗策略。

   状态空间：每个时间点的状态包括当前的积灰指数 DI(t)、当前的发电量 $P(t)$ 和清洗成本。

   动作空间：在每个时间点，动作可以是“清洗”或“不清洗”。

   奖励函数：我们希望最大化电站的总收益。假设电站的收益为：$R(t) = P(t) - C$ 在每个时间点，执行“清洗”或“不清洗”动作后，根据发电量和清洗成本得到相应的奖励。
5. **强化学习模型：** 我们可以使用 **Q-learning** 或深度 Q 网络（DQN）来解决该问题。Q-learning 是通过学习状态-动作值函数 Q 来进行最优策略的决策。其更新规则为：$Q(s,a) = Q(s,a) + \alpha\left(R(s,a) + \gamma \max_a Q\left(s',a\right) - Q(s,a)\right)$ 其中，s 表示状态，a 表示动作，$R(s,a)$ 是奖励，γ 是折扣因子，α 是学习率。
6. **动态清洗策略的优化：** 通过训练强化学习模型，优化清洗决策策略，得到每个时间点最优的清洗决策。最终，模型能够根据积灰程度和发电量的变化动态调整清洗策略。

数学公式与模型：

1. **清洗成本：** $C = 2 \times P(t)$
2. **发电量与积灰的关系：** $P(t) = P_{\text{ideal}}(t) \times (1 - \alpha \times DI(t))$
3. **清洗后发电量：** $P_{\text{after cleaning}} = P_{\text{ideal}}(t) \times (1 - \alpha \times (DI(t) - \gamma))$

4. **奖励函数：** $R(t)=P(t)-C$

5. **Q-learning 更新规则：**

$$Q(s,a)=Q(s,a)+\alpha\left(R(s,a)+\gamma\max_a Q\left(s',a\right)-Q(s,a)\right)$$

Python 代码：

```python
import numpy as np
import random
import pandas as pd

# 假设电站1的数据
df1 = pd.read_excel('电站1发电数据.xlsx')
df1['time'] = pd.to_datetime(df1['time'])

# 假设发电量与积灰指数（DI）之间的关系
alpha = 0.1  # 假设的影响因子
gamma = 0.5  # 假设清洗后积灰减少的程度

# 清洗成本
def cleaning_cost(P):
    return 2 * P

# 发电量与积灰指数的关系
def calculate_power(P_ideal, DI):
    return P_ideal * (1 - alpha * DI)

# 清洗后的发电量
def calculate_cleaning_power(P_ideal, DI):
    return P_ideal * (1 - alpha * (DI - gamma))

# Q-learning 参数
state_space = 10  # 假设积灰指数的离散化空间
action_space = 2  # 两种动作：清洗（1），不清洗（0）
Q = np.zeros((state_space, action_space))  # 初始化Q表

# 奖励函数
def reward(P, action, P_ideal, DI):
    cost = cleaning_cost(P) if action == 1 else 0
    P_after = calculate_cleaning_power(P_ideal, DI) if action == 1
else calculate_power(P_ideal, DI)
    return P_after - cost

# Q-learning 更新过程
```

```
def q_learning(episodes=1000, alpha=0.1, gamma=0.9):
    for episode in range(episodes):
        state = random.randint(0, state_space - 1)  # 初始状态
        for t in range(len(df1)):
            action = np.argmax(Q[state, :]) if random.random() >
0.1 else random.randint(0, action_space - 1)  # 贪婪策略
            P_ideal = df1['irradiance'][t]  # 假设辐照度为发电量的
理想值
            DI = df1['dust_index'][t]  # 当前积灰指数
            P = df1['generation'][t]  # 当前发电量
            r = reward(P, action, P_ideal, DI)  # 计算奖励
            next_state = state  # 状态更新，这里假设为状态离散化
（可以根据实际需求调整）
            Q[state, action] = Q[state, action] + alpha * (r +
gamma * np.max(Q[next_state, :]) - Q[state, action])  # 更新 Q 值
            state = next_state

# 训练 Q-learning 模型
q_learning()

# 输出最优的清洗策略
df1['cleaning_action'] = np.argmax(Q[:, 1], axis=1)  # 选择最优的动
作

# 输出决策结果
df1.to_excel('清洗决策分析.xlsx', index=False)
```