实训1：CSV数据清洗与统计

· ***目标***：掌握Pandas和数据预处理。

· ***任务***：读取销售数据CSV（包含日期、产品、销量、价格），清洗数据，生成统计报告。

· ***详细要求***：

o 读取CSV文件，至少100行，字段：日期（YYYY-MM-DD）、产品（字符串）、销量（整数）、价格（浮点数）。

o 处理缺失值：销量用均值填充，价格用中位数填充。

o 移除异常值：销量<0或价格>1000的记录。

o 计算统计指标：总销量、平均价格、按产品分组的销量总和。

o 输出：清洗后数据保存为cleaned_sales.csv，统计报告保存为sales_report.txt（包含总销量、平均价格、分组统计）。

o 代码需包含异常处理（如文件不存在、格式错误）。

o 添加日志记录，输出每次操作的时间和结果。

· ***技能***：Pandas操作、文件I/O、异常处理、日志记录。

```
import pandas as pd
import numpy as np
import logging
import os
import re
from datetime import datetime

# 配置日志记录
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("sales_data_processing.log"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger()


def clean_and_process_sales_data(input_file="sales_data.csv"):
    """
    处理销售数据的完整流程:
    1. 读取CSV文件
    2. 清洗数据（处理缺失值、异常值、格式错误）
    3. 生成统计报告
    4. 保存清洗后的数据和统计报告
    """
    try:
        # 1. 读取数据
        logger.info(f"开始处理销售数据，输入文件: {input_file}")
```

```
        logger.info(f"当前工作目录: {os.getcwd()}")

        # 检查文件是否存在
        if not os.path.exists(input_file):
            raise FileNotFoundError(f"文件不存在: {input_file}")

        # 读取CSV文件
        df = pd.read_csv(input_file)
        logger.info(f"成功读取数据，共 {len(df)} 行记录")

        # 检查数据行数
        if len(df) < 100:
            logger.warning(f"警告: 文件仅包含 {len(df)} 行数据，少于要求的100
行")

        # 2. 数据清洗
        logger.info("开始数据清洗...")

        # 检查列名
        required_columns = ['date', 'product', 'sales', 'price']
        missing_cols = [col for col in required_columns if col not in
df.columns]
        if missing_cols:
            raise ValueError(f"CSV文件缺少必要的列: {',
'.join(missing_cols)}")

        # 转换日期格式
        date_errors = []
        for i, date_str in enumerate(df['date']):
            try:
                # 尝试解析日期
                pd.to_datetime(date_str, format='%Y-%m-%d')
            except:
                date_errors.append(i)
                # 尝试修复常见日期格式问题
                if re.match(r'\d{4}/\d{2}/\d{2}', date_str):
                    df.at[i, 'date'] = date_str.replace('/', '-')
                elif re.match(r'\d{2}-\d{2}-\d{4}', date_str):
                    parts = date_str.split('-')
                    df.at[i, 'date'] = f"{parts[2]}-{parts[0]}-{parts[1]}"

        if date_errors:
            logger.warning(f"发现并修复了 {len(date_errors)} 条日期格式问题")

        # 确保日期列是datetime类型
        df['date'] = pd.to_datetime(df['date'], errors='coerce')

        # 处理缺失值
        # 销量用均值填充
        if df['sales'].isna().any():
            # 计算均值时排除负值
            valid_sales = df[df['sales'] >= 0]['sales']
            mean_quantity = int(valid_sales.mean()) if not
valid_sales.empty else 0
            df['sales'] = df['sales'].fillna(mean_quantity)
            logger.info(f"填充了 {df['sales'].isna().sum()} 个缺失的销量值，使
用均值: {mean_quantity}")

```

```
        # 价格用中位数填充
        if df['price'].isna().any():
            # 计算中位数时排除异常值
            valid_prices = df[df['price'] <= 1000]['price']
            median_price = valid_prices.median() if not valid_prices.empty else 0
            df['price'] = df['price'].fillna(median_price)
            logger.info(f"填充了 {df['price'].isna().sum()} 个缺失的价格值，使用中位数: {median_price:.2f}")

        # 移除异常值
        initial_count = len(df)
        # 销量 < 0 或 价格 > 1000
        df = df[(df['sales'] >= 0) & (df['price'] <= 1000) & (df['price'] > 0)]
        removed_count = initial_count - len(df)
        logger.info(f"移除了 {removed_count} 条异常记录（销量<0或价格>1000或价格<=0）")

        # 处理非数值型价格数据
        if df['price'].dtype != float and df['price'].dtype != int:
            # 尝试转换价格为数值型
            df['price'] = pd.to_numeric(df['price'], errors='coerce')
            # 再次填充转换失败的价格
            if df['price'].isna().any():
                valid_prices = df[df['price'] <= 1000]['price']
                median_price = valid_prices.median() if not valid_prices.empty else 0
                df['price'] = df['price'].fillna(median_price)
                logger.info(f"转换并填充了 {df['price'].isna().sum()} 个非数值型价格")

        # 3. 生成统计报告
        logger.info("计算统计指标...")

        # 总销量
        total_sales = df['sales'].sum()

        # 平均价格
        avg_price = df['price'].mean()

        # 按产品分组的销量总和
        grouped_sales = df.groupby('product')['sales'].sum().reset_index()
        grouped_sales.columns = ['产品', '总销量']

        # 按日期分组的销量总和
        daily_sales = df.groupby('date')['sales'].sum().reset_index()
        daily_sales.columns = ['日期', '总销量']

        # 4. 保存清洗后的数据
        cleaned_file = "cleaned_sales.csv"
        df.to_csv(cleaned_file, index=False, encoding='utf-8')
        logger.info(f"清洗后的数据已保存至: {cleaned_file} ({len(df)} 行记录)")

        # 5. 保存统计报告
        report_file = "sales_report.txt"
        with open(report_file, 'w', encoding='utf-8') as f:
            f.write("=" * 50 + "\n")
```

```
            f.write("销售数据统计报告\n")
            f.write(f"生成时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
            f.write(f"原始数据行数: {initial_count}\n")
            f.write(f"清洗后数据行数: {len(df)}\n")
            f.write(f"移除异常记录数: {removed_count}\n")
            f.write("=" * 50 + "\n\n")

            f.write(f"总销量: {total_sales:,}\n")
            f.write(f"平均价格: {avg_price:.2f}\n\n")

            f.write("按产品分组的销量统计:\n")
            f.write("-" * 50 + "\n")
            f.write(grouped_sales.to_string(index=False))
            f.write("\n\n")

            f.write("按日期分组的销量统计:\n")
            f.write("-" * 50 + "\n")
            f.write(daily_sales.to_string(index=False))
            f.write("\n\n")

            f.write("=" * 50 + "\n")
            f.write("报告结束\n")

        logger.info(f"统计报告已保存至: {report_file}")
        logger.info("数据处理完成!")

        return df, total_sales, avg_price, grouped_sales, daily_sales

    except FileNotFoundError as e:
        logger.error(f"文件错误: {str(e)}")
        return None, None, None, None, None
    except ValueError as e:
        logger.error(f"数据格式错误: {str(e)}")
        return None, None, None, None, None
    except Exception as e:
        logger.exception(f"处理过程中发生未预期的错误: {str(e)}")
        return None, None, None, None, None


def generate_sample_data():
    """生成样本数据文件"""
    logger.info("生成样本数据...")

    # 生成日期范围
    start_date = datetime(2025, 1, 1)
    end_date = datetime(2025, 4, 10)
    dates = [start_date + pd.DateOffset(days=i) for i in range((end_date - start_date).days + 1)]

    # 生成样本数据
    np.random.seed(42)
    products = ['Phone', 'Laptop', 'Tablet', 'Monitor', 'Headphones']

    data = {
        'date': np.random.choice(dates, 150),
        'product': np.random.choice(products, 150),
        'sales': np.random.randint(1, 100, 150),
```

```
        'price': np.random.uniform(50, 800, 150)
    }

    # 添加一些缺失值和异常值
    df = pd.DataFrame(data)
    df.loc[10:15, 'sales'] = np.nan
    df.loc[20:25, 'price'] = np.nan
    df.loc[30, 'sales'] = -5
    df.loc[35, 'price'] = 1500
    df.loc[40, 'price'] = "invalid"  # 非数值型数据

    # 添加日期格式问题
    df.loc[45, 'date'] = "2025/02/15"  # 错误的分隔符
    df.loc[50, 'date'] = "02-15-2025"  # 错误的格式

    # 保存为CSV
    df.to_csv('sales_data.csv', index=False)
    logger.info("样本数据文件 'sales_data.csv' 已创建")
    return df


if __name__ == "__main__":
    # 如果数据文件不存在，生成样本数据
    if not os.path.exists("sales_data.csv"):
        logger.info("未找到销售数据文件，生成样本数据...")
        generate_sample_data()

    # 处理销售数据
    cleaned_df, total_sales, avg_price, grouped_sales, daily_sales =
clean_and_process_sales_data()

    # 如果处理成功，打印部分结果
    if cleaned_df is not None:
        print("\n数据处理摘要:")
        print(f"清洗后记录数: {len(cleaned_df)}")
        print(f"总销量: {total_sales:,}")
        print(f"平均价格: {avg_price:.2f}")
        print("\n按产品分组销量统计:")
        print(grouped_sales.head())
```