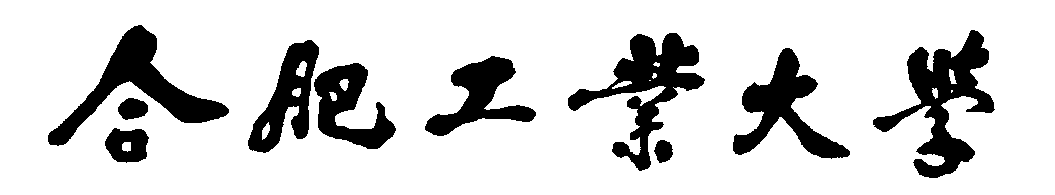
****

分布式程序设计

综合设计报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机科学与技术21-1班 |
| 学生姓名及学号 | 2021214710-杨程锦 |
| 课程教学班号 | 0500020X--001 |
| 任 课 教 师 | 张赞 |
| 实验指导教师 | 张赞 |
| 实验地点 | 翡翠湖校区科教楼C座机房 |
| 2023～2024学年第 一 学期 | |

**综合设计名称：**  综合设计一

时间∶ 2024 年 1 月 12 日

|  |
| --- |
| **预习内容** |
| **一、目的和要求∶**  从网页爬取气象数据，并使用Spark对气象数据进行处理分析，并对分析结果进行可视化。 |
| **二、任务∶**  （1）本次实验所采用的数据自行从气象网站爬取，网站不限。主要是最近24小时各个城市的天气数据，包括但不限于：时间点（整点）、整点气温、整点降水量、风力、整点气压、相对湿度等。数据规模要求超过1000个市县，允许有部分区域和部分时间点数据存在缺失或异常。将数据获取与存储过程截图（包括主要代码）并配必要说明文字保存到设计报告中。  （2）使用spark分析获取的气象数据，要求分析三个方面以上信息（鼓励同学们自由发挥）,例如：各个城市过去24小时的平均气温、湿度、降水量情况、过去24小时全国平均气温最高（或最低）的20座城市、过去24小时全国平均降水量最高（或最低）的20座城市、分析气象情况与地域的关系等。将数据分析过程截图（包括主要代码）并配必要说明文字保存到设计报告中。  （3）使用可视化工具，如python matplotlib库（也可以使用其他工具），绘制柱状图或折线图展示（2）中分析的结果。将可视化过程和结果截图（包括主要代码）并配必要说明文字保存到设计报告中。 |
| **三、准备方案，包括以下内容：**  1、本地系统：windows10 虚拟机系统：Ubuntu 20.04.3  2、工具：PyCharm 2021.3.1  3、环境：本地Python：3.7.4 虚拟机Python：3.8.10  Spark：3.5.0  Hadoop：3.3.1  JDK：1.8.162  4、程序清单：  爬取天气数据：main.py  Spark数据分析：analyze\_avg\_rainfall\_by\_province.py；  analyze\_avg\_temperature\_by\_province.py；  highest\_lowest\_avg\_rain.py；  highest\_lowest\_avg\_temperature.py；  humidity.py；  pressure.py；  rain.py；  temperature.py；  可视化：seerain.py；seerain\_province.py；seetemperature.py；seetemperature\_province.py； |

|  |
| --- |
| **内容** |
| **一、仪器、设备：** |
| **二、内容与步骤（过程及数据记录）：**  1、天气数据爬取  （1）本次数据从中央气象台（http://[www.nmc.cn）爬取；包含了34](http://www.nmc.cn）爬取；包含了34)个省份，2442个城市，58368条数据。数据时间为2024/1/11 18:00:00到2024/1/12 20:00:00。        若部分数据缺失将用9999来代替。  （2）数据爬取  打开网站[www.nmc.cn](http://www.nmc.cn)，打开省份和城市的气象详情界面。    查看网页源码，可以查看到各地天气的网址。  通过F12打开控制台，选中NetWork，选择Fetch/XHR，点击ASX?\_=1705076734569，点击Preview，即可查看到json数据包对应的地址。    通过观察可以发现，json包含的字段为code、province、city、url，在这个链接中，可以得到山西省所有的城市的名字、所属省份、城市编号、已经天气页面对应的网页地址。  获取各城市最近24小时整点天气数据的步骤如下：  首先，通过访问URL http://www.nmc.cn/f/rest/province 获取省份数据。  根据省份数据中的三位编码（例如，山西省编码为“ASX”），构建新的URL http://www.nmc.cn/f/rest/province/+省份三位编码，以获取特定省份的城市数据。  从城市数据中获取各城市的编号（例如，太原市编号为“53772”）。  利用城市编号构建新的URL http://www.nmc.cn/f/rest/passed/+城市编号，以获取某个城市最近24小时整点天气数据。   1. 爬取省份数据        1. 爬取市数据        1. 爬取天气数据       2、数据清洗  将数据爬取的json经过处理存放到csv数据库文件中，为了方便操作，在weather.csv文件中加入了两个字段为province和city。  3、Spark数据分析  本次实验分析了下列数据   |  | | --- | | 过去降雨量 | | 过去温度 | | 过去气压 | | 过去湿度 | | 平均降雨最多20个城市 | | 平均温度最高20个城市 | | 各省平均降雨 | | 各省平均温度 |   (1)过去降雨量:    SparkSession 创建:  通过 SparkSession.builder.master("local").  appName("passed\_rain\_analyse").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，并且指定了文件头。  数据处理： 选择了包含 'province'、'city\_name'、'rain1h' 列的 DataFrame，并将 'rain1h' 列的数据类型转换为 DecimalType(scale=1)。然后使用 filter 筛选了雨量小于 1000 的数据。  分组和排序： 对处理后的数据进行了分组，按 'province' 和 'city\_name' 进行分组，使用 sum 算子求和 'rain1h' 列，最后按照 'rain24h' 列进行降序排序。  缓存和保存结果： 使用 cache 缓存了处理后的数据，并将结果写入了本地的 CSV 文件。  (2)过去温度    SparkSession 创建： 创建了一个 Spark 会话，指定本地运行和应用程序名称为 "passed\_temperature\_analyse"。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province'、'city\_name'、'temperature'、以及通过 date\_format 和 hour 函数生成的 'date' 和 'hour' 列。温度数据被转换为 DecimalType(scale=1) 类型。  过滤保留指定小时的数据： 通过 df\_temperature.filter 筛选保留了指定小时（2, 8, 12, 19）的温度数据。  温度统计分析： 对保留的数据进行了分组，按 'province'、'city\_name' 和 'date' 进行分组，并使用 avg 计算每组的平均温度。然后通过 filter 确保每组的温度数据都包括指定的四个小时。最后使用 sort 对平均温度进行升序排序。  缓存DataFrame： 使用 df\_avg\_temperature.cache() 缓存了处理后的数据。  将结果写入CSV文件：  使用 df\_avg\_temperature.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  返回前10行结果： 通过 df\_avg\_temperature.limit(10).collect() 返回前10行的结果。  (3)过去气压    SparkSession 创建： 创建了一个 Spark 会话，指定本地运行和应用程序名称为 "passed\_pressure\_analyse"。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province'、'city\_name'、'pressure' 列，并将 'pressure' 列的数据类型转换为 double。同时，通过 date\_format 和 hour 函数生成了 'date' 和 'hour' 列。  过滤气压大于1300的数据： 通过 df\_pressure.filter 筛选了气压小于等于 1300 的数据。  过滤保留指定小时的数据： 通过 df\_filtered\_pressure.filter 筛选保留了指定小时（2, 8, 12, 19）的气压数据。  气压统计分析： 对保留的数据进行了分组，按 'province'、'city\_name' 和 'date' 进行分组，并使用 avg 计算每组的平均气压。然后通过 filter 确保每组的气压数据都包括指定的四个小时。最后使用 sort 对平均气压进行升序排序。  缓存DataFrame： 使用 df\_avg\_pressure.cache() 缓存了处理后的数据。  将结果写入CSV文件： 使用 df\_avg\_pressure.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  返回前10行结果： 通过 df\_avg\_pressure.limit(10).collect() 返回前10行的结果。  (4)过去湿度    创建 SparkSession： 使用  SparkSession.builder.master("local").  appName("passed\_humidity\_analyse").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  数据处理：  湿度列转换： 使用 df.withColumn 将 'humidity' 列的数据类型转换为 DecimalType(scale=1)。  日期格式修正： 使用 F.to\_timestamp 修正 'time' 列的日期格式。  选择需要的列： 从原始数据中选择了 'province'、'city\_name'、'humidity' 列，并生成 'date' 和 'hour' 列。  过滤数据： 筛选了湿度小于等于100且时间是指定小时的数据。  湿度统计分析：  分组： 对处理后的数据进行了按 'province'、'city\_name' 和 'date' 分组。  聚合： 使用 F.avg 和 F.count 计算每组的平均湿度和湿度数据的数量。  过滤数据： 通过 filter 确保每组的湿度数据都包括指定的四个小时。  排序： 按照平均湿度降序排序。  缓存DataFrame： 使用 df\_avg\_humidity.cache() 缓存了处理后的数据。  将结果写入CSV文件： 使用 df\_avg\_humidity.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  返回前10行结果： 通过 df\_avg\_humidity.limit(10).collect() 返回前10行的结果。  (5)最高降雨量的20个城市    创建 SparkSession：  使SparkSession.builder.master("local").  appName("rainfall\_analysis").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province'、'city\_name'、'rain1h' 列，并将 'rain1h' 列的数据类型转换为 double。同时，通过 date\_format 和 hour 函数生成了 'date' 和 'hour' 列。  过滤数据： 筛选了过去24小时内的数据，且排除了降水量异常值（大于等于0且小于等于200）。  降水量统计分析： 对筛选后的数据按 'province' 和 'city\_name' 进行分组，使用 avg 计算每组的平均降水量，并按照降序（或升序，取决于 is\_highest 参数）排序，最后限制结果为前20座城市。  缓存DataFrame： 使用 df\_avg\_rainfall.cache() 缓存了处理后的数据。  将结果写入CSV文件： 使用 df\_avg\_rainfall.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  返回结果： 通过 df\_avg\_rainfall.collect() 返回所有结果。  (6)最高温度的20个城市    创建 SparkSession：  使用 SparkSession.builder.master("local")  .appName("temperature\_analysis").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province'、'city\_name'、'temperature' 列，并将 'temperature' 列的数据类型转换为 double。同时，通过 date\_format 和 hour 函数生成了 'date' 和 'hour' 列。  过滤数据： 筛选了过去24小时内的数据，且排除了气温异常值（大于等于-50且小于等于50）。  气温统计分析： 对筛选后的数据按 'province' 和 'city\_name' 进行分组，使用 avg 计算每组的平均气温，并按照降序（或升序，取决于 is\_highest 参数）排序，最后限制结果为前20座城市。  缓存DataFrame： 使用 df\_avg\_temperature.cache() 缓存了处理后的数据。  将结果写入CSV文件：使用 df\_avg\_temperature.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  返回结果： 通过 df\_avg\_temperature.collect() 返回所有结果。  (7)各省平均降雨量    创建 SparkSession：  使用 SparkSession.builder.master("local")  .appName("avg\_rainfall\_analysis").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province' 和 'rain1h' 列，并将 'rain1h' 列的数据类型转换为 double。  过滤数据： 筛选了降雨量大于等于0且小于等于200的数据，排除了异常值。  平均降雨量统计分析： 对筛选后的数据按 'province' 进行分组，使用 avg 计算每个省份的平均降雨量，并按照降序排序。  缓存DataFrame： 使用 df\_avg\_rainfall.cache() 缓存了处理后的数据。  将结果写入CSV文件： 使用 df\_avg\_rainfall.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  打印结果： 使用 df\_avg\_rainfall.show(truncate=False) 打印了统计结果。  返回结果： 通过 df\_avg\_rainfall.collect() 返回所有结果。  (7)各省平均温度    创建 SparkSession：  使用 SparkSession.builder.master("local")  .appName("avg\_temperature\_analysis").getOrCreate() 创建了一个 Spark 会话，指定了本地运行和应用程序名称。  读取CSV文件： 使用 spark.read.csv 读取了指定文件的 CSV 数据，文件头已经被指定为 header=True。  选择需要的列： 从原始数据中选择了 'province' 和 'temperature' 列，并将 'temperature' 列的数据类型转换为 double。  过滤数据： 筛选了气温大于等于-50且小于等于50的数据，排除了异常值。  平均气温统计分析： 对筛选后的数据按 'province' 进行分组，使用 avg 计算每个省份的平均气温，并按照降序排序。  缓存DataFrame： 使用 df\_avg\_temperature.cache() 缓存了处理后的数据。  将结果写入CSV文件： 使用 df\_avg\_temperature.coalesce(1).write.csv 将结果写入本地的 CSV 文件。  打印结果： 使用 df\_avg\_temperature.show(truncate=False) 打印了统计结果。  返回结果： 通过 df\_avg\_temperature.collect() 返回所有结果。  4、可视化展示  本实验采用python的matplotlib和pandas库，对最高降雨量的20个城市、最高温度的20个城市、各省平均降雨量、各省平均温度进行可视化处理。  (1) 最高降雨量的20个城市    (2) 最高温度的20个城市    (3) 各省平均降雨量    (4) 各省平均温度 |

|  |
| --- |
| **三、结果分析、思考题解答∶**  过去降雨量：    过去气压：    过去湿度：    过去温度：    平均降雨最多20个城市：    平均温度最高20个城市：    各省平均降雨量：    各省平均温度：    可视化：  降雨量最多的20个城市：      平均温度最高的20个城市：      各省平均降雨量：      各省平均温度：      结果分析：  实验结果理想  1、实验python爬虫技术爬取了中央气象台网站的天气数据  2、使用Spark技术进行了数据分析，提取出了全国城市降雨量、全国城市温度、全国城市湿度、全国城市气压、全国气温最高的20个城市、全国降雨量最高的20个城市、各省平均降雨量、各省平均气温。  3、使用python的matplotlib和pandas库可视化展示出了全国气温最高的20个城市、全国降雨量最高的20个城市、各省平均降雨量、各省平均气温。  改进：  在Spark数据分析过程中，由于需要处理多个不同天气类型的数据，频繁地使用textFile()读取数据导致程序效率降低。一种常见的优化手段是通过定义一个RDD来存储数据，但由于RDD是一种过程数据，RDD之间进行相互迭代计算，新的RDD生成时，老的RDD会被清除。这种特性最大程度地利用了资源，但对于重复使用同一数据进行多次处理的情况，必须基于RDD的血缘关系重新执行整个计算流程，从最开始的RDD1构建出需要的RDD，这显然降低了程序执行效率。  为了解决这个问题，我采用cache()这个API（例如，rdd.cache()）。它将指定的RDD保存在内存中，防止其在运算过程中被清除，从而显著提高了程序执行效率。这种优化方案避免了重复读取数据，使得在处理不同的天气数据时能够更加高效地利用已缓存的RDD。 |
| **四、感想、体会、建议∶**  这次实验对我来说是一项具有挑战性但也带来巨大收获的任务。当公布了综合设计实验的要求时，我感到有些不安。因为要涉及到爬虫技术，而我对爬虫一窍不通；Spark也并不是我非常熟悉的技术；还有可视化展示方面我也毫无经验。因此，一开始感到心里有些畏惧。然而，在仔细梳理实验步骤后，我决定首先学习Python爬虫技术。我在网上找到了相关教程，投入了一整天的时间学习。经过这一天，我基本掌握了爬虫技术，随后开始爬取天气网站的数据。在这个过程中，遇到了一些问题，有些数据不完整，需要调整代码；有些网站的内容无法正常爬取，经过反复检查发现是自己的爬取路径有问题。最终，我成功地爬取到了所需的数据。接着，我重新学习了一遍Spark技术。虽然这个过程耗时较长，学了两三天，但我收获颇丰，对Spark有了更深入的理解。紧接着，我进行了可视化学习。整个实验过程花费了很多时间，但我也从中学到了很多之前不懂、不熟悉的技术，让我感到收获颇丰。 |
| **综合设计成绩∶**  **指导教师签名：**  年 月 日 |

附录：

**rain.py**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, sum, desc

from pyspark.sql.types import DecimalType

def passed\_rain(filename):

print("Begin to analyze passed rain")

spark = SparkSession.builder.master("local").appName("passed\_rain\_analyse").getOrCreate()

# 读取CSV文件

df = spark.read.csv(f"file://{filename}", header=True)

# 选择相关列并转换雨量数据类型

df\_rain = df.select(col('province'), col('city\_name'), col('rain1h').cast(DecimalType(scale=1))).filter(col('rain1h') < 1000) # 筛选数据，去除无效数据

# 分组、求和、排序

df\_rain\_sum = df\_rain.groupBy("province", "city\_name").agg(sum("rain1h").alias("rain24h")).sort(desc("rain24h"))

# 缓存数据

df\_rain\_sum.cache()

# 将结果写入CSV文件

df\_rain\_sum.coalesce(1).write.csv("file:///home/hadoop/exp/passed\_rain")

print("End analyzing passed rain")

# 返回前20行结果

return df\_rain\_sum.head(20)

result = passed\_rain("/usr/local/spark/weather1.csv")

**pressure.py**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, date\_format, hour, avg, count, format\_number

def passed\_pressure(filename):

print("begin to analyse passed pressure")

# 1. 创建SparkSession

spark = SparkSession.builder.master("local").appName("passed\_pressure\_analyse").getOrCreate()

# 2. 读取CSV文件

df = spark.read.csv(f"file://{filename}", header=True)

# 3. 选择需要的列

df\_pressure = df.select(

col('province'),

col('city\_name'),

col('pressure').cast('double'), # 将气压列转换为double类型

date\_format(col('time'), "yyyy-MM-dd").alias("date"),

hour(col('time')).alias("hour")

)

# 4. 过滤气压大于1300的数据

df\_filtered\_pressure = df\_pressure.filter(df\_pressure['pressure'] <= 1300)

# 5. 过滤保留指定小时的数据

df\_4point\_pressure = df\_filtered\_pressure.filter(df\_filtered\_pressure['hour'].isin([2, 8, 12, 19]))

# 6. 气压统计分析

df\_avg\_pressure = df\_4point\_pressure.groupBy("province", "city\_name", "date").agg(avg("pressure").alias("avg\_pressure")).filter(count("pressure") == 4).sort(col("avg\_pressure")).select("province", "city\_name", "date", format\_number('avg\_pressure', 1).alias("avg\_pressure"))

# 7. 缓存DataFrame

df\_avg\_pressure.cache()

# 8. 将结果写入csv文件

df\_avg\_pressure.coalesce(1).write.csv("file:///home/hadoop/exp/passed\_pressure")

print("end analysing passed pressure")

# 9. 返回前10行结果

avg\_pressure\_list = df\_avg\_pressure.limit(10).collect()

return avg\_pressure\_list

result = passed\_pressure("/usr/local/spark/weather1.csv")

**humidity.py**

from pyspark.sql import SparkSession

from pyspark.sql import functions as F

from pyspark.sql.types import DecimalType

def passed\_humidity\_analyse(filename):

print("begin to analyse passed humidity")

spark = SparkSession.builder.master("local").appName("passed\_humidity\_analyse").getOrCreate()

df = spark.read.csv(f"file://{filename}", header=True)

# 转换湿度列为DecimalType

df = df.withColumn('humidity', df['humidity'].cast(DecimalType(scale=1)))

# 修正日期格式

df = df.withColumn('time', F.to\_timestamp(df['time'], "yyyy-MM-dd HH:mm"))

df\_humidity = df.select(

df['province'],

df['city\_name'],

df['humidity'],

F.date\_format(df['time'], "yyyy-MM-dd").alias("date"),

F.hour(df['time']).alias("hour")

).filter((df['humidity'] <= 100) & (F.hour(df['time']).isin([2, 8, 12, 19])))

df\_4point\_humidity = df\_humidity.filter(df\_humidity['hour'].isin([2, 8, 12, 19]))

df\_avg\_humidity = df\_4point\_humidity.groupBy("province", "city\_name", "date").agg(

F.avg("humidity").alias("avg\_humidity"),

F.count("humidity").alias("count\_humidity")

).filter("count\_humidity == 4").sort("avg\_humidity", ascending=False).select(

"province", "city\_name", "date", F.format\_number('avg\_humidity', 1).alias("avg\_humidity")

)

df\_avg\_humidity.cache()

# 写入CSV文件

df\_avg\_humidity.coalesce(1).write.csv("file:///home/hadoop/exp/passed\_humidity")

print("end analysing passed humidity")

avg\_humidity\_list = df\_avg\_humidity.limit(10).collect()

return avg\_humidity\_list

result = passed\_humidity\_analyse("/usr/local/spark/weather1.csv")

**analyze\_avg\_temperature\_by\_province.py**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, avg, format\_number

def analyze\_avg\_temperature\_by\_province(filename):

print("Begin average temperature analysis by province")

# 创建SparkSession

spark = SparkSession.builder.master("local").appName("avg\_temperature\_analysis").getOrCreate()

# 读取CSV文件

df = spark.read.csv(f"file://{filename}", header=True)

# 选择需要的列

df\_temperature = df.select(

col('province'),

col('temperature').cast('double')

)

# 过滤异常值

df\_filtered\_temperature = df\_temperature.filter(

(df\_temperature['temperature'] >= -50) & (df\_temperature['temperature'] <= 50)

)

# 平均气温统计分析

df\_avg\_temperature = df\_filtered\_temperature.groupBy("province").agg(avg("temperature").alias("avg\_temperature")).sort(col("avg\_temperature"), ascending=False)

# 缓存DataFrame

df\_avg\_temperature.cache()

# 将结果写入csv文件

df\_avg\_temperature.coalesce(1).write.csv("file:///home/hadoop/exp/avg\_temperature\_by\_province")

# 打印结果

df\_avg\_temperature.show(truncate=False)

print("End average temperature analysis by province")

# 返回结果

avg\_temperature\_list = df\_avg\_temperature.collect()

return avg\_temperature\_list

# 示例：统计各个省的平均气温

result\_avg\_temperature\_by\_province = analyze\_avg\_temperature\_by\_province("/usr/local/spark/weather1.csv")

**highest\_lowest\_avg\_rainfall.py**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, date\_format, hour, avg, count, format\_number

def highest\_lowest\_avg\_rainfall(filename, is\_highest=True):

print("begin to analyze past 24 hours average rainfall")

# 1. 创建SparkSession

spark = SparkSession.builder.master("local").appName("rainfall\_analysis").getOrCreate()

# 2. 读取CSV文件

df = spark.read.csv(f"file://{filename}", header=True)

# 3. 选择需要的列

df\_rainfall = df.select(

col('province'),

col('city\_name'),

col('rain1h').cast('double'),

date\_format(col('time'), "yyyy-MM-dd").alias("date"),

hour(col('time')).alias("hour")

)

# 4. 过滤保留过去24小时的数据，并排除异常值

df\_24hours\_rainfall = df\_rainfall.filter(

(df\_rainfall['hour'] >= 2) & (df\_rainfall['hour'] <= 19) &

(df\_rainfall['rain1h'] >= 0) & (df\_rainfall['rain1h'] <= 200)

)

# 5. 降水量统计分析

df\_avg\_rainfall = df\_24hours\_rainfall.groupBy("province", "city\_name").agg(avg("rain1h").alias("avg\_rainfall")).sort(col("avg\_rainfall"), ascending=not is\_highest).limit(20)

# 6. 缓存DataFrame

df\_avg\_rainfall.cache()

# 7. 打印结果

df\_avg\_rainfall.coalesce(1).write.csv("file:///home/hadoop/exp/highest\_avg\_rainfall")

print("end analyzing past 24 hours average rainfall")

# 8. 返回结果

avg\_rainfall\_list = df\_avg\_rainfall.collect()

return avg\_rainfall\_list

# 示例：统计过去24小时全国平均降水量最高的20座城市，排除异常值

result\_highest\_rainfall = highest\_lowest\_avg\_rainfall("/usr/local/spark/weather1.csv", is\_highest=True)