Spark RDD 실습 워크북

PySpark 를 활용한 단계별 데이터 처리 실습

📋 목차

🛠 실습 환경 준비

필요 환경

운영체제: Ubuntu 16.04 (또는 최신 버전)

Spark 버전: 3.4.0

Python 버전: 3.8.18

실행 환경: PySpark 대화형 셸

PySpark 시작하기

```
# PySpark 실행
pyspark
# 또는 Jupyter Notebook 에서
Cd jupyternotebook
```

🔥 기본 RDD 연산 실습

1 단계: RDD 생성과 기본 연산

```
from pyspark import SparkContext, SparkConf
# 配置 Spark
conf = SparkConf().setAppName("RDDExperiment").setMaster("local[*]")
sc = SparkContext(conf=conf)
sc.setLogLevel("WARN")
# 간단한 RDD 생성
numbers = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# 기본 액션 연산
print("전체 요소 개수:", numbers.count())
print("첫 번째 요소:", numbers.first())
```

Spark RDD 实践工作手册

使用 PySpark 进行分步数据处理实践

📋 目录

🛠 实验环境准备

所需环境

操作系统: Ubuntu 16.04 (或最新版本)

Spark 版本: 3.4.0

Python 版本: 3.8.18

运行环境: PySpark 交互式命令行

启动 PySpark

```
# 运行 PySpark
pyspark
# 或在 Jupyter Notebook 中
pip install pyspark
```

🔥 基础 RDD 操作实践

第 1 步：RDD 创建与基本操作

```
from pyspark import SparkContext, SparkConf
# 配置 Spark
conf = SparkConf().setAppName("RDDExperiment").setMaster("local[*]")
sc = SparkContext(conf=conf)
sc.setLogLevel("WARN")
# 创建简单 RDD
numbers = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# 基本行动操作
print("总元素个数:", numbers.count())
print("第一个元素:", numbers.first())
```

```
print("처음 3 개 요소:", numbers.take(3))
print("모든 요소:", numbers.collect())
```

```
print("前 3 个元素:", numbers.take(3))
print("所有元素:", numbers.collect())
```

2 단계: 변환 연산 실습

第 2 步：转换操作实践

```
# map 연산: 각 요소에 2 를 곱하기
doubled = numbers.map(lambda x: x * 2)
print("2 배 결과:", doubled.collect())
# filter 연산: 짝수만 필터링
evens = numbers.filter(lambda x: x % 2 == 0)
print("짝수만:", evens.collect())
# flatMap 연산: 각 숫자를 1 부터 그 숫자까지의 리스트로 변환
flat_result = numbers.flatMap(lambda x: range(1, x + 1))
print("flatMap 결과:", flat_result.collect())
```

```
# map 操作：每个元素乘以 2
doubled = numbers.map(lambda x: x * 2)
print("乘 2 结果:", doubled.collect())
# filter 操作：只保留偶数
evens = numbers.filter(lambda x: x % 2 == 0)
print("偶数:", evens.collect())
# flatMap 操作：将每个数字转换为从 1 到该数字的列表
flat_result = numbers.flatMap(lambda x: range(1, x + 1))
print("flatMap 结果:", flat_result.collect())
```

📊 학생 성적 데이터 분석

📊 学生成绩数据分析

데이터 파일 로드

加载数据文件

```
# 텍스트 파일 읽기
data_file = "file:///usr/local/spark/sparksqldata/chapter5-data01.txt"
lines = sc.textFile(data_file)
# 데이터 구조 확인
print("전체 라인 수:", lines.count())
print("첫 5 줄 샘플:", lines.take(5))
```

```
# 读取文本文件
data_file = "file:///usr/local/spark/sparksqldata/chapter5-data01.txt"
lines = sc.textFile(data_file)
# 确认数据结构
print("总行数:", lines.count())
print("前 5 行样本:", lines.take(5))
```

문제 1: 총 학생 수 계산

问题 1：计算学生总数

```
# 각 라인을 파싱하여 학생 이름 추출
students = lines.map(lambda line: line.split(',')[0])
# 중복 제거하여 고유한 학생 이름만 주출
unique_students = students.distinct()
# 총 학생 수
total_students = unique_students.count()
print(f"총 학생 수: {total_students}명")
# 학생 이름 목록 확인 (일부)
print("학생 목록 (처음 10 명):", unique_students.take(10))
```

```
# 解析每行提取学生姓名
students = lines.map(lambda line: line.split(',')[0])
# 去重获取唯一学生姓名
unique_students = students.distinct()
# 学生总数
total_students = unique_students.count()
print(f"学生总数: {total_students}名")
# 查看学生姓名列表（部分）
print("学生列表（前 10 名）:", unique_students.take(10))
```

문제 2: 총 개설 과목 수 계산

```
# 각 라인을 파싱하여 과목명 추출
courses = lines.map(lambda line: line.split(',')[1])
# 중복 제거하여 고유한 과목명만 추출
unique_courses = courses.distinct()
# 총 과목 수
total_courses = unique_courses.count()
print(f"총 개설 과목 수: {total_courses}개")
# 과목 목록 확인
print("개설 과목 목록:", unique_courses.collect())
```

问题 2：计算开设课程总数

```
# 解析每行提取课程名
courses = lines.map(lambda line: line.split(',')[1])
# 去重获取唯一课程名
unique_courses = courses.distinct()
# 课程总数
total_courses = unique_courses.count()
print(f"开设课程总数: {total_courses}门")
# 查看课程列表
print("开设课程列表:", unique_courses.collect())
```

문제 3: Tom 학생의 평균 성적 계산

```
# Tom 학생의 성적만 필터링
tom_records = lines.filter(lambda line: line.split(',')[0] == 'Tom')
# Tom 의 성적 추출
tom_scores = tom_records.map(lambda line: int(line.split(',')[2]))
# 평균 계산
tom_total = tom_scores.sum()
tom_count = tom_scores.count()
tom_average = tom_total / tom_count
print(f"Tom 의 총 성적: {tom_total}")
print(f"Tom 의 수강 과목 수: {tom_count}")
print(f"Tom 의 평균 성적: {tom_average:.2f}")
# Tom 의 과목별 성적 확인
tom_details = tom_records.map(lambda line: (line.split(',')[1], int(line.split(',')[2])))
print("Tom 의 과목별 성적:", tom_details.collect())
```

问题 3：计算 Tom 学生的平均成绩

```
# 筛选 Tom 学生的成绩记录
tom_records = lines.filter(lambda line: line.split(',')[0] == 'Tom')
# 提取 Tom 的成绩
tom_scores = tom_records.map(lambda line: int(line.split(',')[2]))
# 计算平均值
tom_total = tom_scores.sum()
tom_count = tom_scores.count()
tom_average = tom_total / tom_count
print(f"Tom 的总分: {tom_total}")
print(f"Tom 的选课数: {tom_count}")
print(f"Tom 的平均成绩: {tom_average:.2f}")
# 查看 Tom 的各科成绩
tom_details = tom_records.map(lambda line: (line.split(',')[1], int(line.split(',')[2])))
print("Tom 的各科成绩:", tom_details.collect())
```

문제 4: 각 학생의 수강 과목 수 계산

```
# (학생명, 1) 형태의 키-값 쌍 생성
student_course_pairs = lines.map(lambda line: (line.split(',')[0], 1))
# 학생별로 과목 수 집계
student_course_counts = student_course_pairs.reduceByKey(lambda a, b: a + b)
# 결과 출력 (상위 10 명)
```

问题 4：计算每个学生的选课数

```
# 创建（学生名，1）形式的键值对
student_course_pairs = lines.map(lambda line: (line.split(',')[0], 1))
# 按学生分组统计课程数
student_course_counts = student_course_pairs.reduceByKey(lambda a, b: a + b)
# 输出结果（前 10 名）
```

```
print("각 학생의 수강 과목 수 (상위 10 명):")
for student, count in student_course_counts.take(10):
    print(f"{student}: {count}과목")
# 전체 결과를 이름순으로 정렬
sorted_counts = student_course_counts.sortByKey()
print("\n 전체 학생 수강 과목 수 (ABC 순):")
for student, count in sorted_counts.collect():
    print(f"{student}: {count}과목")
```

```
print("各学生选课数（前 10 名）:")
for student, count in student_course_counts.take(10):
    print(f"{student}: {count}门课")
# 按姓名排序显示全部结果
sorted_counts = student_course_counts.sortByKey()
print("\n 全部学生选课数（按字母顺序）:")
for student, count in sorted_counts.collect():
    print(f"{student}: {count}门课")
```

문제 5: DataBase 과목 수강생 수

问题 5：DataBase 课程选课人数

```
# DataBase 과목 수강 기록 필터링
database_records = lines.filter(lambda line: line.split(',')[1] == 'DataBase')
# 수강생 수 계산
database_students = database_records.count()
print(f"DataBase 과목 수강생 수: {database_students}명")
# DataBase 수강생 명단
database_student_names = database_records.map(lambda line: line.split(',')[0])
print("DataBase 수강생 명단:", database_student_names.collect())
```

```
# 筛选 DataBase 课程记录
database_records = lines.filter(lambda line: line.split(',')[1] == 'DataBase')
# 计算选课人数
database_students = database_records.count()
print(f"DataBase 课程选课人数: {database_students}名")
# DataBase 选课学生名单
database_student_names = database_records.map(lambda line: line.split(',')[0])
print("DataBase 选课学生名单:", database_student_names.collect())
```

문제 6: 각 과목의 평균 성적 계산

问题 6：计算各门课程的平均分

```
# (과목명, 성적) 형태의 키-값 쌍 생성
course_score_pairs = lines.map(lambda line: (line.split(',')[1], int(line.split(',')[2])))
# 과목별 성적 그룹화
course_scores = course_score_pairs.groupByKey()
# 각 과목의 평균 계산
course_averages = course_scores.map(lambda x: (x[0], sum(x[1]) / len(x[1])))
# 결과 출력
print("각 과목의 평균 성적:")
for course, avg in course_averages.collect():
    print(f"{course}: {avg:.2f}점")
# 평균 성적순으로 정렬
sorted_averages = course_averages.sortBy(lambda x: x[1], ascending=False)
print("\n 평균 성적 순위:")
for course, avg in sorted_averages.collect():
    print(f"{course}: {avg:.2f}점")
```

```
# 创建（课程名，成绩）形式的键值对
course_score_pairs = lines.map(lambda line: (line.split(',')[1], int(line.split(',')[2])))
# 按课程分组成绩
course_scores = course_score_pairs.groupByKey()
# 计算各课程平均分
course_averages = course_scores.map(lambda x: (x[0], sum(x[1]) / len(x[1])))
# 输出结果
print("各课程平均分:")
for course, avg in course_averages.collect():
    print(f"{course}: {avg:.2f}分")
# 按平均分排序
sorted_averages = course_averages.sortBy(lambda x: x[1], ascending=False)
print("\n 平均分排名:")
for course, avg in sorted_averages.collect():
    print(f"{course}: {avg:.2f}分")
```

파일 처리 실습

실습 2: 데이터 중복 제거 프로그램

다음 파일을 파일 문서를 작성

fileA.txt :

20170101 x

20170102 y

20170103 x

20170104 y

20170105 z

20170106 z

fileB.txt

20170101 y

20170102 y

20170103 x

20170104 z

20170105 y

파일위치 : /usr/local/spark/sparksqldata

독립 애플리케이션 작성

```
# duplicate_removal.py
from pyspark import SparkContext, SparkConf
import subprocess
def main():
    # Spark 설정
    conf = SparkConf().setAppName("DuplicateRemoval")
    sc = SparkContext(conf=conf)
    subprocess.call(["hadoop","fs","-rm","-r", "output_dedup"])
    try:
        # 입력 파일 읽기
```

文件处理实践

实践 2：数据去重程序

编写以下文件文档:

fileA.txt :

20170101 x

20170102 y

20170103 x

20170104 y

20170105 z

20170106 z

fileB.txt

20170101 y

20170102 y

20170103 x

20170104 z

20170105 y

文件位置：: /usr/local/spark/sparksqldata

编写独立应用程序

```
# duplicate_removal.py
from pyspark import SparkContext, SparkConf
import subprocess
def main():
    # Spark 配置
    conf = SparkConf().setAppName("DuplicateRemoval")
    sc = SparkContext(conf=conf)
    subprocess.call(["hadoop","fs","-rm","-r", "output_dedup"])
    try:
        # 读取输入文件
```

```
    file_a = sc.textFile("file_a.txt")
    file_b = sc.textFile("file_b.txt")
    # 두 파일 합치기
    combined = file_a.union(file_b)
    # 중복 제거
    unique_lines = combined.distinct()
    # 정렬
    sorted_lines = unique_lines.sortBy(lambda line: line)
    # 결과 저장
    sorted_lines.saveAsTextFile("output_c")
    print("중복 제거 완료!")
    print(f"원본 라인 수: {combined.count()}")
    print(f"중복 제거 후: {unique_lines.count()}")
  finally:
    sc.stop()
if __name__ == "__main__":
  main()
```

```
    file_a = sc.textFile("file_a.txt")
    file_b = sc.textFile("file_b.txt")
    # 合并两个文件
    combined = file_a.union(file_b)
    # 去重
    unique_lines = combined.distinct()
    # 排序
    sorted_lines = unique_lines.sortBy(lambda line: line)
    # 保存结果
    sorted_lines.saveAsTextFile("output_c")
    print("去重完成！")
    print(f"原始行数: {combined.count()}")
    print(f"去重后行数: {unique_lines.count()}")
  finally:
    sc.stop()
if __name__ == "__main__":
  main()
```

테스트 데이터 생성

创建测试数据

```
# 파일 A 생성 (file_a.txt)
file_a_content = """20170101 x
20170102 y
20170103 x
20170104 y
20170105 z
20170106 z"""
# 파일 B 생성 (file_b.txt)
file_b_content = """20170101 y
20170102 y
20170103 x
20170104 z
```

```
# 文件 A 内容 (file_a.txt)
file_a_content = """20170101 x
20170102 y
20170103 x
20170104 y
20170105 z
20170106 z"""
# 文件 B 内容 (file_b.txt)
file_b_content = """20170101 y
20170102 y
20170103 x
20170104 z
```

```
20170105 y"""
# PySpark 에서 직접 테스트
file_a_rdd = sc.parallelize(file_a_content.split('\n'))
file_b_rdd = sc.parallelize(file_b_content.split('\n'))
# 합치기 및 중복 제거
combined_rdd = file_a_rdd.union(file_b_rdd)
unique_rdd = combined_rdd.distinct().sortBy(lambda x: x)
print("중복 제거 결과:")
for line in unique_rdd.collect():
    print(line)
```

```
20170105 y"""
# 在 PySpark 中直接测试
file_a_rdd = sc.parallelize(file_a_content.split('\n'))
file_b_rdd = sc.parallelize(file_b_content.split('\n'))
# 合并及去重
combined_rdd = file_a_rdd.union(file_b_rdd)
unique_rdd = combined_rdd.distinct().sortBy(lambda x: x)
print("去重结果:")
for line in unique_rdd.collect():
    print(line)
```

평균값 계산 프로그램

平均值计算程序

실습 3: 학생 평균 성적 계산

实践 3：学生平均成绩计算

독립 애플리케이션 작성

编写独立应用程序

```
# average_calculator.py
from pyspark import SparkContext, SparkConf
import os
def main():
# 출력 디렉토리 삭제 (로컬 파일 시스템 기준)
    os.system("hdfs dfs -rm -r /user/hadoop/student_averages")
    conf = SparkConf().setAppName("StudentAverageCalculator")
    sc = SparkContext(conf=conf)
    sc.setLogLevel('WARN')
    try:
        # 여러 과목 파일 읽기
        algorithm_rdd =
sc.textFile(""file:///usr/local/spark/sparksqldata/Algorithm.txt"
")
        database_rdd =
sc.textFile("file:///usr/local/spark/sparksqldata/Database.txt ")
        python_rdd =
sc.textFile("file:///usr/local/spark/sparksqldata/Python.txt”
        # 모든 성적 데이터 합치기
        all_scores =
algorithm_rdd.union(database_rdd).union(python_rdd)
        # (학생명, 성적) 형태로 파싱
        student_scores = all_scores.map(lambda line:
```

```
# average_calculator.py
from pyspark import SparkContext, SparkConf
import os
def main():
# 출력 디렉토리 삭제 (로컬 파일 시스템 기준)
    os.system("hdfs dfs -rm -r /user/hadoop/student_averages")
    conf = SparkConf().setAppName("StudentAverageCalculator")
    sc = SparkContext(conf=conf)
    sc.setLogLevel('WARN')
    try:
        # 读取多个科目文件
        algorithm_rdd =
sc.textFile(""file:///usr/local/spark/sparksqldata/Algorithm.txt"
")
        database_rdd =
sc.textFile("file:///usr/local/spark/sparksqldata/Database.txt ")
        python_rdd =
sc.textFile("file:///usr/local/spark/sparksqldata/Python.txt”
        # 合并所有成绩数据
        all_scores =
algorithm_rdd.union(database_rdd).union(python_rdd)
        # 解析为（学生名，成绩）格式
        student_scores = all_scores.map(lambda line:
```

```
line.split()).map(lambda parts: (parts[0], int(parts[1])))
    # 학생별 성적 그룹화
    grouped_scores = student_scores.groupByKey()
    # 평균 계산
    student_averages = grouped_scores.map(lambda x: (x[0], sum(x[1]) / len(x[1])))
    # 결과 정렬
    sorted_averages = student_averages.sortBy(lambda x: x[1], ascending=False)
    # 결과 출력
    print("학생별 평균 성적:")
    for student, avg in sorted_averages.collect():
        print(f"({student}, {avg:.2f})")
    # 파일로 저장
    result_rdd = sorted_averages.map(lambda x: f"({x[0]},{x[1]:.2f})")
    result_rdd.saveAsTextFile("student_averages")
  finally:
    sc.stop()
if __name__ == "__main__":
  main()
```

PySpark 에서 직접 테스트

```
# 테스트 데이터 생성
algorithm_data = ["소명 92", "소홍 87", "소신 82", "소리 90"]
database_data = ["소명 95", "소홍 81", "소신 89", "소리 85"]
python_data = ["소명 82", "소홍 83", "소신 94", "소리 91"]
# RDD 생성
algorithm_rdd = sc.parallelize(algorithm_data)
database_rdd = sc.parallelize(database_data)
python_rdd = sc.parallelize(python_data)
# 모든 데이터 합치기
all_data = algorithm_rdd.union(database_rdd).union(python_rdd)
# 파싱 및 계산
parsed_data = all_data.map(lambda line: line.split()).map(lambda parts: (parts[0], int(parts[1])))
grouped_data = parsed_data.groupByKey()
averages = grouped_data.map(lambda x: (x[0], round(sum(x[1]) / len(x[1]), 2)))
print("학생별 평균 성적:")
for student, avg in averages.collect():
```

```
line.split()).map(lambda parts: (parts[0], int(parts[1])))
    # 按学生分组成绩
    grouped_scores = student_scores.groupByKey()
    # 计算平均分
    student_averages = grouped_scores.map(lambda x: (x[0], sum(x[1]) / len(x[1])))
    # 排序结果
    sorted_averages = student_averages.sortBy(lambda x: x[1], ascending=False)
    # 输出结果
    print("学生平均成绩:")
    for student, avg in sorted_averages.collect():
        print(f"({student}, {avg:.2f})")
    # 保存到文件
    result_rdd = sorted_averages.map(lambda x: f"({x[0]},{x[1]:.2f})")
    result_rdd.saveAsTextFile("student_averages")
  finally:
    sc.stop()
if __name__ == "__main__":
  main()
```

在 PySpark 中直接测试

```
# 创建测试数据
algorithm_data = ["小明 92", "小红 87", "小新 82", "小丽 90"]
database_data = ["小明 95", "小红 81", "小新 89", "小丽 85"]
python_data = ["小明 82", "小红 83", "小新 94", "小丽 91"]
# 创建 RDD
algorithm_rdd = sc.parallelize(algorithm_data)
database_rdd = sc.parallelize(database_data)
python_rdd = sc.parallelize(python_data)
# 合并所有数据
all_data = algorithm_rdd.union(database_rdd).union(python_rdd)
# 解析并计算
parsed_data = all_data.map(lambda line: line.split()).map(lambda parts: (parts[0], int(parts[1])))
grouped_data = parsed_data.groupByKey()
averages = grouped_data.map(lambda x: (x[0], round(sum(x[1]) / len(x[1]), 2)))
print("学生平均成绩:")
for student, avg in averages.collect():
```

```
print(f"({student}, {avg})")
```

```
print(f"({student}, {avg})")
```

🔢 누적 변수(Accumulator) 실습

🔢 累加器(Accumulator)实践

문제 7: Accumulator 를 사용한 DataBase 수강생 수 계산

问题 7：使用累加器计算 DataBase 选课人数

```
# Accumulator 생성
database_counter = sc.accumulator(0)
# 각 라인을 검사하여 DataBase 과목이면 카운터 증가
def count_database_course(line):
    parts = line.split(',')
    if len(parts) >= 2 and parts[1] == 'DataBase':
        database_counter.add(1)
    return line
# 모든 라인에 대해 함수 적용
lines.foreach(count_database_course)
print(f"Accumulator 로 계산한 DataBase 수강생 수: {database_counter.value}명")
# 다른 방법과 비교
database_count_filter = lines.filter(lambda line: line.split(',')[1] == 'DataBase').count()
print(f"Filter 로 계산한 DataBase 수강생 수: {database_count_filter}명")
```

```
# 创建累加器
database_counter = sc.accumulator(0)
# 检查每行，如果是 DataBase 课程则计数器加 1
def count_database_course(line):
    parts = line.split(',')
    if len(parts) >= 2 and parts[1] == 'DataBase':
        database_counter.add(1)
    return line
# 对所有行应用函数
lines.foreach(count_database_course)
print(f"使用累加器计算的 DataBase 选课人数: {database_counter.value}名")
# 与其他方法比较
database_count_filter = lines.filter(lambda line: line.split(',')[1] == 'DataBase').count()
print(f"使用 Filter 计算的 DataBase 选课人数: {database_count_filter}名")
```

Accumulator 고급 사용법

累加器高级用法

```
# 여러 과목 동시 카운트
database_acc = sc.accumulator(0)
algorithm_acc = sc.accumulator(0)
python_acc = sc.accumulator(0)
def count_multiple_courses(line):
    parts = line.split(',')
    if len(parts) >= 2:
        course = parts[1]
        if course == 'DataBase':
            database_acc.add(1)
```

```
# 同时统计多门课程
database_acc = sc.accumulator(0)
algorithm_acc = sc.accumulator(0)
python_acc = sc.accumulator(0)
def count_multiple_courses(line):
    parts = line.split(',')
    if len(parts) >= 2:
        course = parts[1]
        if course == 'DataBase':
            database_acc.add(1)
```

```
        elif course == 'Algorithm':
            algorithm_acc.add(1)
        elif course == 'Python':
            python_acc.add(1)
# 한 번의 순회로 여러 과목 카운트
lines.foreach(count_multiple_courses)
print(f"DataBase: {database_acc.value}명")
print(f"Algorithm: {algorithm_acc.value}명")
print(f"Python: {python_acc.value}명")
```

```
        elif course == 'Algorithm':
            algorithm_acc.add(1)
        elif course == 'Python':
            python_acc.add(1)
# 一次遍历统计多门课程
lines.foreach(count_multiple_courses)
print(f"DataBase: {database_acc.value}名")
print(f"Algorithm: {algorithm_acc.value}名")
print(f"Python: {python_acc.value}名")
```

🎯 종합 실습 문제

도전 과제 1: 성적 통계 분석

```
# 전체 성적 분포 분석
all_scores = lines.map(lambda line: int(line.split(',')[2]))
# 기본 통계
total_records = all_scores.count()
sum_scores = all_scores.sum()
max_score = all_scores.max()
min_score = all_scores.min()
avg_score = sum_scores / total_records
print(f"전체 성적 통계:")
print(f"- 총 성적 기록 수: {total_records}")
print(f"- 평균 점수: {avg_score:.2f}")
print(f"- 최고 점수: {max_score}")
print(f"- 최저 점수: {min_score}")
```

🎯 综合实践题目

挑战题目 1：成绩统计分析

```
# 全体成绩分布分析
all_scores = lines.map(lambda line: int(line.split(',')[2]))
# 基本统计
total_records = all_scores.count()
sum_scores = all_scores.sum()
max_score = all_scores.max()
min_score = all_scores.min()
avg_score = sum_scores / total_records
print(f"全体成绩统计:")
print(f"- 总成绩记录数: {total_records}")
print(f"- 平均分: {avg_score:.2f}")
print(f"- 最高分: {max_score}")
print(f"- 最低分: {min_score}")
```

```
# 성적 구간별 분포
grade_distribution = all_scores.map(lambda score:
    'A' if score >= 90 else
    'B' if score >= 80 else
    'C' if score >= 70 else
    'D' if score >= 60 else 'F'
).countByValue()
print(f"\n 성적 분포:")
for grade, count in sorted(grade_distribution.items()):
    print(f"- {grade}: {count}개 ({count/total_records*100:.1f}%)")
```

```
# 按成绩等级分布
grade_distribution = all_scores.map(lambda score:
    'A' if score >= 90 else
    'B' if score >= 80 else
    'C' if score >= 70 else
    'D' if score >= 60 else 'F'
).countByValue()
print(f"\n 成绩分布:")
for grade, count in sorted(grade_distribution.items()):
    print(f"- {grade}: {count}个 ({count/total_records*100:.1f}%)")
```

도전 과제 2: 우수 학생 찾기

挑战题目 2：查找优秀学生

```
# 평균 80 점 이상 학생 찾기
student_scores = lines.map(lambda line: (line.split(',')[0],
int(line.split(',')[2])))
student_grouped = student_scores.groupByKey()
student_averages = student_grouped.map(lambda x: (x[0],
sum(x[1]) / len(x[1])))
excellent_students = student_averages.filter(lambda x: x[1] >= 80)
excellent_count = excellent_students.count()
print(f"평균 80 점 이상 우수 학생: {excellent_count}명")
print("우수 학생 목록:")
for student, avg in excellent_students.sortBy(lambda x: x[1],
ascending=False).collect():
    print(f"- {student}: {avg:.2f}점")
```

```
# 查找平均 80 分以上的学生
student_scores = lines.map(lambda line: (line.split(',')[0],
int(line.split(',')[2])))
student_grouped = student_scores.groupByKey()
student_averages = student_grouped.map(lambda x: (x[0],
sum(x[1]) / len(x[1])))
excellent_students = student_averages.filter(lambda x: x[1] >= 80)
excellent_count = excellent_students.count()
print(f"平均 80 分以上的优秀学生: {excellent_count}名")
print("优秀学生名单:")
for student, avg in excellent_students.sortBy(lambda x: x[1],
ascending=False).collect():
    print(f"- {student}: {avg:.2f}分")
```

📝 실습 정리 및 체크리스트

📝 实践总结及检查清单

완료해야 할 실습 항목

[ ] PySpark 기본 환경 설정

[ ] 기본 RDD 연산 (map, filter, reduce 등)

[ ] 키-값 RDD 연산 (groupByKey, reduceByKey 등)

[ ] 학생 성적 데이터 7 가지 문제 해결

[ ] 파일 중복 제거 독립 애플리케이션 작성

[ ] 평균 성적 계산 독립 애플리케이션 작성

[ ] Accumulator 를 활용한 카운팅

[ ] 도전 과제 완료

需要完成的实践项目

[ ] PySpark 基本环境设置

[ ] 基础 RDD 操作（map、filter、reduce 等）

[ ] 键值对 RDD 操作（groupByKey、reduceByKey 等）

[ ] 学生成绩数据 7 个问题解决

[ ] 文件去重独立应用程序编写

[ ] 平均成绩计算独立应用程序编写

[ ] 使用累加器进行计数

[ ] 挑战题目完成

학습 성과 확인

RDD 기본 개념 이해: Transformation 과 Action 의 차이점 설명 가능

키-값 쌍 처리: groupByKey, reduceByKey 등의 적절한 사용법 이해

파일 처리: 여러 파일을 읽고 처리하는 방법 숙지

성능 최적화: 불필요한 셔플링을 피하는 방법 이해

독립 애플리케이션: Spark 애플리케이션을 작성하고 실행하는 능력

학习成果确认

RDD 基本概念理解: 能够说明 Transformation 和 Action 的区别

键值对处理: 理解 groupByKey、reduceByKey 等的正确使用方法

文件处理: 掌握读取和处理多个文件的方法

性能优化: 理解避免不必要 shuffle 的方法

独立应用程序: 具备编写和运行 Spark 应用程序的能力

📚 추가 학습 자료

참고 문서

📚 额外学习资料

参考文档

Apache Spark 공식 문서:

PySpark API 문서:

다음 단계 학습 주제

Spark SQL: 구조화된 데이터 처리

DataFrames: 더 고수준의 데이터 추상화

Spark Streaming: 실시간 데이터 처리

MLlib: 머신러닝 라이브러리 활용

🎉 축하합니다! RDD 실습을 완료하셨습니다.

이 워크북을 통해 Spark RDD 의 핵심 개념과 실제 데이터 처리 방법을 익혔습니다. 다음 단계로 Spark SQL 이나 DataFrame 을 학습해보세요!

Apache Spark 官方文档:

PySpark API 文档:

下一步学习主题

Spark SQL: 结构化数据处理

DataFrames: 更高级的数据抽象

Spark Streaming: 实时数据处理

MLlib: 机器学习库应用

🎉 恭喜！您已完成 RDD 实践！

通过这个工作手册，您掌握了 Spark RDD 的核心概念和实际数据处理方法。接下来可以学习 Spark SQL 或 DataFrame！