STUDENT ID: 31153291

STUDENT NAME: PEIYU LIU

Table of contents

1 Working with RDD

1.1 Data Preparation and Loading

1.1.1 Create SparkSession and SparkContext

Back to top

In [ ]:

```
# import libraries  that assignment nended.
from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark.rdd import RDD
from pyspark.sql.types import IntegerType
# do file literated
import os,sys
# do pyspark sql process
from pyspark.sql import functions as F

#create a SparkContext object using SparkSession
master = 'local[*]'
app_name = '31153291_Ass1'
#build a SparkConf
spark_conf = SparkConf().setMaster(master).setAppName(app_name)
spark = SparkSession.builder.config(conf=spark_conf).getOrCreate()
sc = spark.sparkContext
sc.setLogLevel('ERROR')
```

1.1.2 Import CSV files and Make RDD for each file

Back to top

Reference: list all files:https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory (https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory)

Literate all flight files in flight folder. Add files' names with files' path into list

In [ ]:

```
import os,sys

file_path = './flight-delays/'
files = os.listdir(file_path)
file_container = []
for file in files:
    if file.startswith('flight'): # only need flights*files
        print(file)
        file_container.append('./flight-delays/'+file )
```

Reference: string content join: https://www.w3schools.com/python/ref_string_join.asp (https://www.w3schools.com/python/ref_string_join.asp)

In [ ]:

```
# read all data from raw csv files
airports_raw_data = sc.textFile('./flight-delays/airports.csv')
# combine each flight data into one aggregate and divide by ','
flights_raw_data = sc.textFile(','.join(file_container))
```

1.1.2.1 Flights RDD

Back to top

- remove file headers
- rdd.first(): read first row
- lambda loop all rows
- filter to filtrate all rows that equal to header row
- split header into multiple values
- convert 'YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'FLIGHT_NUMBER' into integer format
- convert 'DEPARTURE_DELAY', 'ARRIVAL_DELAY', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'TAXI_OUT', 'TAXI_IN' into float format
- for loop to check each columns index in header
- after get all index of columns that I need to conver format use index

In [ ]:

```
# remove file headers
# rdd.first(): read first row
aps_header = airports_raw_data.first()
fls_header = flights_raw_data.first()

# lambda loop all rows
# filter to filtrate all rows that equal to header row
airports_rdd = airports_raw_data.filter(lambda flag: flag != aps_header)
flights_rdd = flights_raw_data.filter(lambda flag: flag != fls_header)

# split header into multiple values.
flight_header_list = fls_header.split(',')
airport_header_list = aps_header.split(',')

# convert 'YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'FLIGHT_NUMBER' into integer format.
int_fields = ['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'FLIGHT_NUMBER']
# convert 'DEPARTURE_DELAY', 'ARRIVAL_DELAY', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE',
# 'TAXI_OUT', 'TAXI_IN' into float format.
float_fields = ['DEPARTURE_DELAY', 'ARRIVAL_DELAY', 'ELAPSED_TIME', 'AIR_TIME',
                'DISTANCE','TAXI_OUT', 'TAXI_IN']

# for loop to check each columns index in header.
int_container = []
for flag1,flag2 in enumerate(flight_header_list):
    if flag2 in int_fields:
        int_container.append(flag1)

print(int_container)
print('**************************')

# for loop to check each columns index in header.
float_container = []
for flag1, flag2 in enumerate(flight_header_list):
    if flag2 in float_fields:
        float_container.append(flag1)

print(float_container)

# check index of LATITUDE and LONGITUDE.
airport_container = []
for flag1,flag2 in enumerate(airport_header_list):
    if flag2 == "LATITUDE":
        print("LATITUDE:"+str(flag1))
    else:
        if flag2 == "LONGITUDE":
            print("LONGITUDE:"+str(flag1))

# after get all index of columns that I need to conver format, then do convert actions below:
```

Reference:

- convert rows to dictionary row:https://stackoverflow.com/questions/49432167/how-to-convert-rows-into-a-list-of-dictionaries-in-pyspark (https://stackoverflow.com/questions/49432167/how-to-convert-rows-into-a-list-of-dictionaries-in-pyspark)
- change None value: https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python (https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python)
- python dict zip: https://www.codegrepper.com/code-examples/python/python+dict+zip (https://www.codegrepper.com/code-examples/python/python+dict+zip)
- Row actions:https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html)

In [ ]:

```
# Reference:
# convert rows to dictionary row:
#https://stackoverflow.com/questions/49432167/how-to-convert-rows-into-a-list-of-dictionaries-in-pyspark
# Use rdd Row function to convert data into Row format{column name: value}
from pyspark.sql import Row

# fucntion to split data by ',' into each row.
# literate all values in each row to check None value,
# None value loop will result in breakdown, so that change None value into readable format.
# Reference :
# change None value:
#https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python
def flightsRddConvertRow(rddRow):
    rowEach = rddRow.split(',')
    for flag1, flag2 in enumerate(rowEach):
        if flag1 in int_container:
            if flag2 == '':
                rowEach[flag1] = float('nan')
            else:
                rowEach[flag1] = int(flag2)
        if flag1 in float_container:
            if flag2 == '':
                rowEach[flag1] = float('nan')
            else:
                rowEach[flag1] = float(flag2)

# Reference: python dict zip: https://www.codegrepper.com/code-examples/python/python+dict+zip
# Zip key and value to dictionary and then use **dict to split dictionary,
# Use Row function to create a row object by using named arguments and values.
# Reference: Row actions:
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html
    return Row(**dict(zip(flight_header_list, rowEach)))

new_flights_rdd_row = flights_rdd.map(flightsRddConvertRow)
new_flights_rdd_row.take(1)
```

1.1.2.2 Airports RDD

Back to top

Reference:

convert rows to dictionary row:

- https://stackoverflow.com/questions/49432167/how-to-convert-rows-into-a-list-of-dictionaries-in-pyspark (https://stackoverflow.com/questions/49432167/how-to-convert-rows-into-a-list-of-dictionaries-in-pyspark)

change None value:

- https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python (https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python)

python dict zip:

- https://www.codegrepper.com/code-examples/python/python+dict+zip (https://www.codegrepper.com/code-examples/python/python+dict+zip)

Row actions:

- https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html)

In [ ]:

```
# fucntion to split data by ',' into each row.
# literate all values in each row to check None value,
# None value loop will result in breakdown, so that change None value into readable format.
# Reference :
# change None value:
# https://stackoverflow.com/questions/45489357/change-none-to-float-in-list-of-strings-python
def airportsRddConvertRow(rddRow):
    rowEach = rddRow.split(',')
    for flag1, flag2 in enumerate(rowEach):
        if flag1 in [5, 6]:  # latitude, langitude
            if flag2 == '':
                rowEach[flag1] = float('nan')
            else:
                rowEach[flag1] = float(flag2)

# Reference: python dict zip: https://www.codegrepper.com/code-examples/python/python+dict+zip
# Zip key and value to dictionary and then use **dict to split dictionary,
# Use Row function to create a row object by using named arguments and values.
# Reference: Row actions:
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html
    return Row(**dict(zip(airport_header_list, rowEach)))


new_airports_rdd_row = airports_rdd.map(airportsRddConvertRow)
print(new_airports_rdd_row.take(1))
```

1.1.3 Show RDD number of columns, and number of records

Back to top

Reference:

- partitioning function:https://kontext.tech/column/spark/299/data-partitioning-functions-in-spark-pyspark-explained (https://kontext.tech/column/spark/299/data-partitioning-functions-in-spark-pyspark-explained)

In [ ]:

```
def print_partitions(df):
    numPartitions = df.getNumPartitions()
    print(f"Number of partitions:{numPartitions}")
    partitions = df.glom().collect()
    for index,partition in enumerate (partitions):
        if len(partition)>0:
            print(f"Partition{index}: {len(partition)} records")

# flights records
print('Flights count:'+str(new_flights_rdd_row.count()))
print_partitions(new_flights_rdd_row)
print('-------------------------------------------------')
# airposts records
print('Airposts count:'+str(new_airports_rdd_row.count()))
print_partitions(new_airports_rdd_row)
```

1.2 Dataset Partitioning

1.2.1 Obtain the maximum arrival time

Back to top

Reference: RDD.max():http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.max.html (http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.max.html)

lambda to literate all ARRIVAL_DELAY value and use max function to calculate maximum value and filter out.

In [ ]:

```
maxArrivalDelay = new_flights_rdd_row.filter(lambda x: x['ARRIVAL_DELAY']).max(
    key=lambda x: x['ARRIVAL_DELAY'])
maxArrivalDelay['ARRIVAL_DELAY']
```

1.2.2 Obtain the minimum arrival time

Back to top

Reference:

- RDD.min():http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.min.html?highlight=rdd%20min#pyspark.RDD.min (http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.min.html?highlight=rdd%20min#pyspark.RDD.min)

lambda to literate all ARRIVAL_DELAY value and use min function to calculate minimum value and filter out.

In [ ]:

```
minArrivalDelay = new_flights_rdd_row.filter(lambda x:x['ARRIVAL_DELAY']).min(
    key = lambda x:x['ARRIVAL_DELAY'])
minArrivalDelay['ARRIVAL_DELAY']
```

1.2.3 Define hash partitioning function

Back to top

Reference: repartition: https://www.qedev.com/bigdata/104082.html (https://www.qedev.com/bigdata/104082.html)

- HashPartitioner partitions by calculating the hashCode for a given key.
- Then use partitionBy to repartition the data.

In [ ]:

```
# hash partition function
def hash_function(key):
    keyCount = 0
    for flag in key:
        keyCount += int(flag)
    return keyCount

# hash partitioning: use ARRIVAL_TIME as code.
key = new_flights_rdd_row.filter(lambda x: x['ARRIVAL_TIME'] != '').map(lambda x: (x['ARRIVAL_TIME'], x))

# Repartition 4
newHashRepartition = key.partitionBy(4, hash_function)
newHashRepartition.getNumPartitions()
```

1.2.4 Display the records in each partition

Back to top

In [ ]:

```
# records calculator that I designed before.
# records in each partition
print_partitions(newHashRepartition)
```

1.3 Query RDD

1.3.1 Collect a total number of flights for each month

Back to top

Reference: GroupBy key and use collect to count numbers:

- https://stackoverflow.com/questions/56895694/group-by-key-value-pyspark (https://stackoverflow.com/questions/56895694/group-by-key-value-pyspark)
- map function to reformat data as (Month,value) format.
- use groupby to analyse each month and mapvalue to records.

In [ ]:

```
key_value = new_flights_rdd_row.map(lambda x: (x['MONTH'], x))
key_value = key_value.groupByKey().mapValues(lambda x: len(x))
month_records = key_value.collect()

for key, value in month_records:
    print('Month:{0}--Numbers:{1}'.format(key, value))
```

1.3.2 Collect the average delay for each month

Back to top

Reference: PySpark reduceByKey():

- merge the values of each key https://sparkbyexamples.com/pyspark/pyspark-reducebykey-usage-with-examples/ (https://sparkbyexamples.com/pyspark/pyspark-reducebykey-usage-with-examples/)
- The average delay for each month:
  - filtrate none value
  - filtrate negative value, only calculate delay time
  - reformat data to {Month:delay_time, times}
  - calculate total delay time and times
  - time divide times.

In [ ]:

```
AvgDelayMonth = new_flights_rdd_row.filter(
    lambda flag: flag.ARRIVAL_DELAY != None).filter(
    lambda row: row.ARRIVAL_DELAY > 0).map(
    lambda flag: (flag.MONTH, [flag.ARRIVAL_DELAY, 1])).reduceByKey(
    lambda key, key_: [key[0] + key_[0], key[1] + key_[1]]).mapValues(
    lambda flag: flag[0] / flag[1]).collect()

AvgDelayMonth
```

2 Working with DataFrame

2.1. Data Preparation and Loading

2.1.1 Define dataframes and loading scheme

Back to top

use the module spark.read.format("csv")

In [ ]:

```
airportsDf = spark.read.format('csv').option('header', True).option('inferSchema', True).load(
    'flight-delays/airports.csv')

flightsDf = spark.read.format('csv').option('header', True).option('inferSchema', True).load(
    'flight-delays/flight*.csv')
```

2.1.2 Display the schema of the final two dataframes

Back to top

In [ ]:

```
airportsDf.printSchema()
flightsDf.printSchema()
```

2.2. Query Analysis

2.2.1 January flight events with ANC airport

Back to top

Reference: spark.sql functions:https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html (https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html)

- Filter to filtrate data as conditions,
- F.col select column name and set conditions.
- Use select to set result columns.

In [ ]:

```
from pyspark.sql import functions as F

janFlightEventsAncDf = flightsDf.filter(F.col('YEAR') == 2015).filter(F.col('MONTH') == 1).filter(
    F.col('ORIGIN_AIRPORT') == 'ANC').select(
    'MONTH', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', 'DISTANCE','ARRIVAL_DELAY')

janFlightEventsAncDf.toPandas()
```

2.2.2 Average Arrival Delay From Origin to Destination

Back to top

Reference: aggreate actions:https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.GroupedData.agg.html (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.GroupedData.agg.html)

- groupBy use conditions
- calculate average values and give a new name
- sort by ascending

In [ ]:

```
janFlightEventsAncAvgDf = janFlightEventsAncDf.groupBy('ORIGIN_AIRPORT', 'DESTINATION_AIRPORT').agg(
    F.mean('ARRIVAL_DELAY').alias('AVERAGE_DELAY')).sort('AVERAGE_DELAY', ascending=True)

janFlightEventsAncAvgDf.toPandas()
```

2.2.3 Join Query with Airports DataFrame

Back to top

Reference:

- default inner join:http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.join.html?highlight=inner%20join (http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.join.html?highlight=inner%20join)

In [ ]:

```
joinedSqlDf = airportsDf.join(janFlightEventsAncAvgDf, airportsDf.IATA_CODE
                              == janFlightEventsAncAvgDf.ORIGIN_AIRPORT)

joinedSqlDf.toPandas()
```

2.3. Analysis

2.3.1 Relationship between day of week with mean arrival delay, total time delay, and count flights

Back to top

- Ed clarification:https://edstem.org/au/courses/6038/discussion/565373 (https://edstem.org/au/courses/6038/discussion/565373)
- Total time delay is related to sum of arrival delay,
- numOfFlights is related to the number of flights,
- average of arrival delay is related to the mean of the arrival delay
- group by 'DAY_OF_WEEK'
- sort descending

In [ ]:

```
dayOfWeekDelayDf = flightsDf.filter(
    F.col('YEAR') == 2015).withColumn('ARRIVAL_DELAY', F.col('ARRIVAL_DELAY')).groupby('DAY_OF_WEEK').agg(
    F.mean('ARRIVAL_DELAY').alias('MeanArrivalDelay'), F.sum('ARRIVAL_DELAY').alias('TotalTimeDelay'),
    F.count('FLIGHT_NUMBER').alias('NumOfFlights')).sort('NumOfFlights', ascending=False)

dayOfWeekDelayDf.show()
```

What can you analyse from this query results?

Answer: From query results, it can be seen that the largest average delayed arrival time is Friday, and the smallest is Saturday. It can also be seen that the specific value of the average delay time per day during the week and the specific value of the total delay time. It can be seen that when the number of flights is large, the total delay time will be higher than when the number of flights is small. Quantitative changes cause qualitative changes.

2.3.2 Display mean arrival delay each month

Back to top

Reference:

- withcolumn:https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.withColumn.html (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.withColumn.html)
- groupBy:https://hendra-herviawan.github.io/pyspark-groupby-and-aggregate-functions.html (https://hendra-herviawan.github.io/pyspark-groupby-and-aggregate-functions.html)
- select columns and group
- aggregate columns with function processing
- create a new column with processing
- count times
- sort ascending

In [ ]:

```
monthDelayDf = flightsDf.withColumn('ARRIVAL_DELAY', F.col('ARRIVAL_DELAY')).groupby('MONTH').agg(
    F.mean('ARRIVAL_DELAY').alias('MeanArrivalDelay'), F.sum('ARRIVAL_DELAY').alias('TotalTimeDelay'),
    F.count('FLIGHT_NUMBER').alias('NumOfFlights')).sort('MeanArrivalDelay', ascending=True)

monthDelayDf.show()
```

What can you analyse from this query results?

Answer: From query results, you can see that the average delay time will have a negative value, and the aircraft may arrive early, so a negative value is normal. September and October have the least delays, and the number of flights per month is similar, and the difference is not very large. The most delayed flight is in June, so the total delay is longer.

2.3.3 Relationship between mean departure delay and mean arrival delay

Back to top

- divide by Month for groups
- Aggregate Function to calculate average value in specified columns.
- sort by average departure delay value, descending order.

In [ ]:

```
DepArrDelayDf = flightsDf.groupBy('MONTH').agg(
    F.mean('DEPARTURE_DELAY').alias('MeanDeptDelay'),
    F.mean('ARRIVAL_DELAY').alias('MeanArrivalDelay')).sort('MeanDeptDelay', ascending=False)

DepArrDelayDf.show()
```

What you can analyse from the relationship between two columns: Mean Departure Delay and Mean Arrival Delay?

Answer: From query results, you can see that the average delay time for departure and the average delay time for arrival are related. When the delay time for take-off is more, the delay time for arrival is more. Late departure and late arrival are related, and the latter is more likely to occur when the former occurs.

3 RDDs vs DataFrame vs Spark SQL

Implement the following queries using RDDs, DataFrames and SparkSQL separately. Log the time taken for each query in each approach using the “%%time” built-in magic command in Jupyter Notebook and discuss the performance difference of these 3 approaches.

Find the MONTH and DAY_OF_WEEK, number of flights, and average delay where TAIL_NUMBER = ‘N407AS’. Note number of flights and average delay should be aggregated separately. The average delay should be grouped by both MONTH and DAYS_OF_WEEK.

3.1 RDD Operation

Back to top

Reference:

- PySpark reduceByKey():merge the values of each key https://sparkbyexamples.com/pyspark/pyspark-reducebykey-usage-with-examples/ (https://sparkbyexamples.com/pyspark/pyspark-reducebykey-usage-with-examples/)
- mapValue collect:https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.mapValues.html (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.mapValues.html)
- select rows that TAIL_NUMBER is equal 'N407AS'
- filtrate none value
- change value format as {month,day,departureDelayTime,arrivalDelayTime}
- change value type {month,day,departureDelayTime,arrivalDelayTime} as string.
- change data value by a+b
- calculate average delay time and times
- time divide times.

In [ ]:

```
%%time
new_flights_rdd_row.filter(lambda flag: flag.TAIL_NUMBER == 'N407AS').filter(
    lambda flag: flag.ARRIVAL_DELAY != "").filter(
    lambda flag: flag.DEPARTURE_DELAY != "").map(
    lambda flag: [flag.MONTH, flag.DAY_OF_WEEK, flag.DEPARTURE_DELAY, flag.ARRIVAL_DELAY]).map(
    lambda flag: (str([flag[0], flag[1]]), [1, flag[2], flag[3]])).reduceByKey(
    lambda k, k_: [k[0] + k_[0], k[1] + k_[1], k[2] + k_[2]]).mapValues(
    lambda flag: [flag[0], flag[1] / flag[0], flag[2] / flag[0]]).collect()
```

3.2 DataFrame Operation

Back to top

- select rows that TAIL_NUMBER is equal 'N407AS'
- group data by 'MONTH', 'DAY_OF_WEEK', 'TAIL_NUMBER'
- aggregate functions to calculate average value and assign a new column
- count how many fight events
- sort by ascending order
- generate pandas view

In [ ]:

```
%%time

flightsDf.filter(F.col('TAIL_NUMBER') == 'N407AS').groupBy('MONTH', 'DAY_OF_WEEK', 'TAIL_NUMBER').agg(
    F.mean('DEPARTURE_DELAY').alias('AvgDepDelay'), F.mean('ARRIVAL_DELAY').alias('AvgArrDelay'),
    F.count('FLIGHT_NUMBER').alias('NumOfFlights')).sort(['MONTH'], ascending=True).toPandas()
```

3.3 Spark SQL OPERATION

Back to top

In [ ]:

```
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.createOrReplaceTempView.html
# Creates or replaces a local temporary view with this DataFrame.
flightsDf.createOrReplaceTempView("flightsSQL")
```

In [ ]:

```
%%time
spark.sql('''
SELECT MONTH,DAY_OF_WEEK,TAIL_NUMBER,
MEAN(DEPARTURE_DELAY) AS AvgDepDelay,
MEAN(ARRIVAL_DELAY) AS AvgArrDelay,
COUNT(FLIGHT_NUMBER) AS NumOfFlights
FROM
(
SELECT MONTH,DAY_OF_WEEK,FLIGHT_NUMBER,DEPARTURE_DELAY,ARRIVAL_DELAY,TAIL_NUMBER
FROM flightsSQL
WHERE TAIL_NUMBER = 'N407AS'
)
GROUP BY MONTH,DAY_OF_WEEK,TAIL_NUMBER
ORDER BY MONTH
''').toPandas()
```

3.4 Discussion

Back to top

Reference: https://ipython.readthedocs.io/en/stable/interactive/magics.html?highlight=%25time#magic-time (https://ipython.readthedocs.io/en/stable/interactive/magics.html?highlight=%25time#magic-time)

- The CPU time is divided into user and sys.
- The user value represents the time used by the program itself and the subroutines in the library it calls.
- sys is the execution time of system calls called directly or indirectly by the program.
- The wall time from the start of the program to the end of the program execution, including the time used by the CPU.

RDD Operation:

Time:

- CPU times: user 22.5 ms,
- sys: 10.8 ms,
- total: 33.3 ms,
- Wall time: 8.31 s

DataFrame Operation:

Time:

- CPU times: user 21.3 ms,
- sys: 4.83 ms,
- total: 26.2 ms,
- Wall time: 2.12 s #### SQL Operation: Time:
- CPU times: user 16.5 ms,
- sys: 3.79 ms,
- total: 20.3 ms,
- Wall time: 2.47 s

When using SQL Operation for data processing, the CPU usage time is the shortest, only 16.5ms.

The execution time of system is also the shortest, only 3.79ms.

But the shortest time between program start and end of operation is DataFrame Operation. Through time comparison of the three, the most efficient way to process the data this time should be DataFrame Operation.

Because the operation needs to select and filter the columns of the data, and then process them to find the result.

DataFrame is a distributed data set organized in named columns, which is equivalent to an optimized table in a relational database.

DataFrame is more efficient than RDD because it specifies a specific structure to constrain the data. And rdd occupies too much memory, so DataFrame Operation is more suitable for data processing under this condition.