BigDataAnalysis BDA2

Helena Llorens Lluís (hllor282) Yi Yang (yiyan338)

May 2025

1 ASSIGNMENT 1

What are the lowest and highest temperatures measured each year for the period 1950- 2014. Provide the lists sorted in the descending order with respect to the maximum temperature. In this exercise you will use the temperature-readings.csv file.

- code

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import col, split, avg, max as spark_max, min as
↪  spark_min, countDistinct, count
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
↪  FloatType

# create a SparkContext which tells Spark how to access a cluster
spark = SparkSession.builder.appName("exercise 1").getOrCreate()
sc = spark.sparkContext

# create distributed datasets
# This path is to the file on hdfs
temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
temps = lines.map(lambda x: (
    x[0],                   # station number
    int(x[1][0:4]),         # year
    int(x[1][5:7]),         # month
    x[2],                   # time
    float(x[3]),            # temperature
    x[4]                    # quality
    ))

# schema for the dataframe
schema = StructType([
    StructField("station_num", StringType()),
    StructField("year", IntegerType()),
    StructField("month", IntegerType()),
    StructField("time", StringType()),
    StructField("temp", FloatType()),
    StructField("quality", StringType()),
])
df = spark.createDataFrame(temps, schema = schema)

# filter the years
filtered_df = df.where((col("year") >= 1950) & (col("year") <= 2014))

# get max temperature per year
max_temps = filtered_df.groupBy("year") \
                       .agg(spark_max("temp").alias("temp"))

# get the station number for the max temperatures
max_temps = max_temps.join(filtered_df, ["year", "temp"]) \
                     .select("year", "station_num", "temp") \
                     .sort("temp", ascending = False)

# get min temperature per year
min_temps = filtered_df.groupBy("year") \
                       .agg(spark_min("temp").alias("temp"))

# get the station number for the min temperatures
min_temps = min_temps.join(filtered_df, ["year", "temp"]) \
                     .select("year", "station_num", "temp") \
                     .sort("temp", ascending = False)

# save result
max_temps.write.csv("BDA/output/max_temps", header=True)
min_temps.write.csv("BDA/output/min_temps", header=True)
```

- Maximum *(partial results)*

```
[
    year,station_num,temp
    1975,86200,36.1
    1992,63600,35.4
    1994,117160,34.7
    2014,96560,34.4
    2010,75250,34.4
    1989,63050,33.9
    1982,94050,33.8
    1968,137100,33.7
    1966,151640,33.5
    1983,98210,33.3
    2002,78290,33.3
    2002,78290,33.3
    1970,103080,33.2
    1986,76470,33.2
    2000,62400,33.0
    1956,145340,33.0
    1959,65160,32.8
    1991,137040,32.7
    2006,75240,32.7
    1988,102540,32.6
]
```

- Minimum *(partial results)*

```
[
    year,station_num,temp
    1990,166870,-35.0
    1990,147270,-35.0
    1952,192830,-35.5
    1974,179950,-35.6
    1974,166870,-35.6
    1954,113410,-36.0
    1992,179960,-36.1
    1975,157860,-37.0
    1972,167860,-37.5
    1995,182910,-37.6
    2000,169860,-37.6
    1957,159970,-37.8
    1983,191900,-38.2
    1989,166870,-38.2
    1953,183760,-38.4
    2009,179960,-38.5
    1993,191900,-39.0
    1984,123480,-39.2
    1984,191900,-39.2
    2008,179960,-39.3
]
```

2 ASSIGNMENT 2

Count the number of readings for each month in the period of 1950-2014 which are higher than 10 degrees.Repeat the exercise, this time taking only distinct readings from each station. That is, if a station reported a reading above 10 degrees in some month, then it appears only once in the count for that month. In this exercise, you will use the temperature-readings.csv file.

- code

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import col, split, avg, max as spark_max, min as
↪  spark_min, countDistinct, count
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
↪  FloatType

# create a SparkContext which tells Spark how to access a cluster
spark = SparkSession.builder.appName("exercise 2").getOrCreate()
sc = spark.sparkContext

# create distributed datasets
# This path is to the file on hdfs
temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
```

```
temps = lines.map(lambda x: (
    x[0],                   # station number
    int(x[1][0:4]),         # year
    int(x[1][5:7]),         # month
    x[2],                   # time
    float(x[3]),            # temperature
    x[4]                    # quality
    ))

# schema for the dataframe
schema = StructType([
    StructField("station_num", StringType()),
    StructField("year", IntegerType()),
    StructField("month", IntegerType()),
    StructField("time", StringType()),
    StructField("temp", FloatType()),
    StructField("quality", StringType()),
])
df = spark.createDataFrame(temps, schema = schema)

# filter years and temperature over 10ºC
filtered_df = df.where((col("year") >= 1950) & (col("year") <= 2014) &
↪  (col("temp") > 10))

# count
count_temp = filtered_df.groupBy("year", "month") \
                        .count().alias("count")

count_temp = count_temp.sort("count", ascending = False)

count_temp.write.csv("BDA/output/count")

# count distinct
count_temp = filtered_df.groupBy("year", "month") \
                        .agg(countDistinct("station_num").alias("count_dist"))

count_temp = count_temp.sort("count_dist", ascending = False)

count_temp.write.csv("BDA/output/count_distinct")
```

- Count of readings(temperature higher than 10 degrees) per month *(partial results)*

```
[
    year,month,value
    2014,7,147681
    2011,7,146656
    2010,7,143419
    2012,7,137477
    2013,7,133657
    2009,7,133008
    2011,8,132734
    2009,8,128349
    2013,8,128235
    2003,7,128133
    2002,7,127956
    2006,8,127622
    2008,7,126973
    2002,8,126073
    2005,7,125294
    2011,6,125193
    2012,8,125037
    2006,7,124794
    2010,8,124417
    2014,8,124045
    1997,7,123496
]
```

- Count of distinct readings(temperature higher than 10 degree) per month: *(partial results)*

```
[
    year,month,value
    1972,10,378
    1973,5,377
    1973,6,377
    1973,9,376
    1972,8,376
    1972,6,375
    1972,5,375
    1971,8,375
    1972,9,375
    1971,6,374
    1971,9,374
    1972,7,374
    1971,5,373
    1973,8,373
    1974,8,372
    1974,6,372
    1974,9,370
    1970,8,370
    1973,7,370
    1974,5,370
    1971,7,370
]
```

3 ASSIGNMENT 3

Find the average monthly temperature for each available station in Sweden. Your result should include average temperature for each station for each month in the period of 1960- 2014. Bear in mind that not every station has the readings for each month in this timeframe. In this exercise you will use the temperature-readings.csv file.

- code

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import col, split, avg, max as spark_max, min as
↪  spark_min, countDistinct, count
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
↪  FloatType

# create a SparkContext which tells Spark how to access a cluster
spark = SparkSession.builder.appName("exercise 3").getOrCreate()
sc = spark.sparkContext

# create distributed datasets
# This path is to the file on hdfs
temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
temps = lines.map(lambda x: (
    x[0],                   # station number
    int(x[1][0:4]),         # year
    int(x[1][5:7]),         # month
    x[2],                   # time
    float(x[3]),            # temperature
    x[4]                    # quality
    ))

# schema for the dataframe
schema = StructType([
    StructField("station_num", StringType()),
    StructField("year", IntegerType()),
    StructField("month", IntegerType()),
    StructField("time", StringType()),
    StructField("temp", FloatType()),
    StructField("quality", StringType()),
])
df = spark.createDataFrame(temps, schema = schema)

# filter years
filtered_df = df.where((col("year") >= 1960) & (col("year") <= 2014))

# average monthly temperature per station
avg_temp = filtered_df.groupBy("station_num", "year", "month") \
                      .agg(avg("temp").alias("avg_temp"))

avg_temp = avg_temp.sort("avg_temp", ascending = False)

# save result
avg_temp.write.csv("BDA/output/avg_temp")
```

- The average monthly temperature for each available station. *(partial results)*

```
[
    station,year,month,avgMonthlyTemperature
    96000,2014,7,26.299999237060547
    65450,1994,7,23.654838664557346
    95160,1994,7,23.50537642612252
    75120,1994,7,23.268817204301076
    105260,1994,7,23.143820259008514
    85280,1994,7,23.108602175148583
    54550,1983,8,23.0
    54550,1975,8,22.9625
    96550,1994,7,22.957894760265685
    96000,1994,7,22.931182820309875
    106070,1994,7,22.822580665670415
    173960,1972,7,22.776666831970214
    54300,1994,7,22.760215082476215
    85210,1994,7,22.755913970290973
    65450,2006,7,22.740860190442813
    75120,2006,7,22.73010758430727
    103080,1994,7,22.708602125926685
    92100,1994,7,22.69892466965542
    94180,1994,7,22.681720487533077
    83230,1994,7,22.577419393806046
]
```

4 ASSIGNMENT 4

Provide a list of stations with their associated maximum measured temperatures and maximum measured daily precipitation. Show only those stations where the maximum temperature is between 25 and 30 degrees and maximum daily precipitation is between 100 mm and 200mm. In this exercise you will use the temperature-readings.csv and precipitation-readings.csv files.

- code

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, max as spark_max, sum as spark_sum
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
↪  FloatType

# Create a SparkSession
spark = SparkSession.builder.appName("exercise 4").getOrCreate()
sc = spark.sparkContext

# Read and parse temperature dataset
temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
temp_lines = temperature_file.map(lambda x: x.split(";")).map(lambda x: (
    x[0],                   # station_num
    x[1],                   # date
    float(x[3])             # temperature
))

temp_schema = StructType([
    StructField("station_num", StringType()),
    StructField("date", StringType()),
    StructField("temp", FloatType()),
])

df_temp = spark.createDataFrame(temp_lines, schema=temp_schema)

# Get maximum temperature per station
max_temp_per_station = df_temp.groupBy("station_num") \
                              .agg(spark_max("temp").alias("max_temp")) \
                              .filter((col("max_temp") >= 25) & (col("max_temp")
                              ↪   <= 30))

# Read and parse precipitation dataset
precipitation_file = sc.textFile("BDA/input/precipitation-readings.csv")
prec_lines = precipitation_file.map(lambda x: x.split(";")).map(lambda x: (
    x[0],                   # station_num
    x[1],                   # date
    float(x[3])             # precipitation
))

prec_schema = StructType([
    StructField("station_num", StringType()),
    StructField("date", StringType()),
    StructField("prec", FloatType()),
])

df_prec = spark.createDataFrame(prec_lines, schema=prec_schema)

# Sum daily precipitation per station
daily_prec = df_prec.groupBy("station_num", "date") \
                    .agg(spark_sum("prec").alias("daily_prec"))

# Max daily precipitation per station
max_prec_per_station = daily_prec.groupBy("station_num") \
                                 .agg(spark_max("daily_prec").alias("max_prec"))
                                 ↪   \
                                 .filter((col("max_prec") >= 100) &
                                 ↪   (col("max_prec") <= 200))

# Join filtered temp and prec on station number
final_result = max_temp_per_station.join(max_prec_per_station, on="station_num",
↪   how="inner") \
                                   .select("station_num", "max_temp", "max_prec")
                                   ↪   \
                                   .orderBy("station_num", ascending=False)

# Save result
final_result.write.csv("BDA/output/exercise4", header=True)
```

No results were returned for the query filtering stations with a maximum temperature between 25°C and 30°C and a maximum daily precipitation between 100 mm and 200 mm. This suggests that no stations in the dataset satisfy both criteria.

5 ASSIGNMENT 5

Calculate the average monthly precipitation for the Östergotland region (list of stations is provided in a separate file) for the period 1993-2016. In orderto dothis, you willfirstneed to calculate the total monthly precipitation for each station before calculating the monthly average (by averaging over stations).
In this exercise you will use the precipitation-readings.csv and stations-Ostergotland.csv files. HINT (not for the SparkSQL lab): Avoid using joins here! stations-Ostergotland.csv is small and if distributed will cause a number of unnecessary shuffles when joined with precipitationRDD. If you distribute precipitation-readings.csv then either repartition your stations RDD to 1 partition or make use of the collect function to acquire a python list and broadcast function to broadcast the list to all nodes.

- code

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import col, split, avg, max as spark_max, min as
↪  spark_min, countDistinct, count, sum
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
↪  FloatType

# create a SparkContext which tells Spark how to access a cluster
spark = SparkSession.builder.appName("exercise 5").getOrCreate()
sc = spark.sparkContext

# create distributed datasets
# This path is to the file on hdfs

# precipitation file
precipitation_file = sc.textFile("BDA/input/precipitation-readings.csv")
prec_lines =precipitation_file.map(lambda line: line.split(";"))

prec_lines = prec_lines.map(lambda x: (
    x[0],                   # stat_num
    int(x[1][0:4]),         # year
    int(x[1][5:7]),         # month
    x[2],                   # time
    float(x[3]),            # precipitation (mm)
    x[4]                    # quality
    ))

# precipitation dataset schema
prec_schema = StructType([
    StructField("station_num", StringType()),
    StructField("year", IntegerType()),
    StructField("month", IntegerType()),
    StructField("time", StringType()),
    StructField("prec", FloatType()),
    StructField("quality", StringType()),
])
```

```
df_prec = spark.createDataFrame(prec_lines, schema = prec_schema)

# stations Ostergotland dataset
oster_file = sc.textFile("BDA/input/stations-Ostergotland.csv")
oster_lines = oster_file.map(lambda line: line.split(";"))

oster_lines = oster_lines.map(lambda x: (
    x[0],                   # stat_num
    x[1]                    # stat_name
    ))

# stations Ostergotland dataset schema
oster_schema = StructType([
    StructField("station_num", StringType()),
    StructField("station_name", StringType()),
])
df_oster = spark.createDataFrame(oster_lines, schema = oster_schema)

# filter years
filtered_df = df_prec.where((col("year") >= 1993) & (col("year") <= 2016))

# get only Ostergotland stations
stat_oster = filtered_df.join(df_oster, ["station_num"])

# average precipitation per station
avg_prec = stat_oster.groupBy("station_num", "year", "month") \
                      .agg(sum("prec").alias("avg_prec"))

# average precipitation per month
avg_prec_month = avg_prec.groupBy("year", "month") \
                .agg(avg("avg_prec").alias("avg_prec_month")) \
                .orderBy(["year", "month"], ascending = False)

# save results
avg_prec_month.write.csv("BDA/output/avg_prec")
```

- Monthly precipitation average for stations in Ostergotland. *(partial results)*

```
[
    year,month,avgMonthlyPrecipitation
    2016,7,0.0
    2016,6,47.662500091828406
    2016,5,29.250000204890966
    2016,4,26.900000237859786
    2016,3,19.96250029373914
    2016,2,21.56250028219074
    2016,1,22.325000344775617
    2015,12,28.92500019259751
    2015,11,63.887500293552876
    2015,10,2.2625000346451998
    2015,9,101.30000030715019
    2015,8,26.9875001097098
    2015,7,119.09999992512167
    2015,6,78.66250023245811
    2015,5,93.22500019986182
    2015,4,15.337500078603625
    2015,3,42.612500292249024
    2015,2,24.825000358745456
    2015,1,59.11250052880496
    2014,12,35.462500284425914
    2014,11,52.42500038724393
]
```