Cloud Computing 2022/23 spring

Practice 8 - Parallel Spark DataFrames

In this Practice session, you will continue working with the Apache Spark framework in Python. You will learn how to create Spark DataFrame applications using the PyCharm Python IDE.

References

- Spark DataFrame/SQL programming guide: https://spark.apache.org/docs/latest/sql-programming-guide.html
- Spark Python DataFrame API - https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html#DataFrame-Creation
- Spark DataFrame/SQL Functions - https://spark.apache.org/docs/latest/api/sql/index.html
- Spark DataFrame/SQL CheatSheet! - https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PySpark_SQL_Cheat_Sheet_Python.pdf

Dataset Description

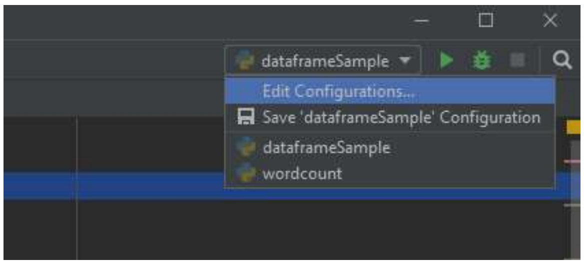
The dataset that we will analyze using Spark DataFrame API is the **Beach Weather Stations - Automated Sensors** dataset from the City of Chicago. The data is collected by the weather sensors at beaches along Chicago's Lake Michigan lakefront.

- Name: Beach Weather Stations - Automated Sensors
- Dataset source: https://data.cityofchicago.org/d/k7hf-8y75
  - You can download by selecting `Export -> CSV`
- Dataset attributes (column names) are:
  1. Station Name: string
  2. Measurement Timestamp: string
  3. Air Temperature: double
  4. Wet Bulb Temperature: double
  5. Humidity: integer
  6. Rain Intensity: double
  7. Interval Rain: double
  8. Total Rain: double
  9. Precipitation Type: integer
  10. Wind Direction: integer
  11. Wind Speed: double
  12. Maximum Wind Speed: double
  13. Barometric Pressure: double
  14. Solar Radiation: integer
  15. Heading: integer
  16. Battery Life: double
  17. Measurement Timestamp Label: string
  18. Measurement ID: string

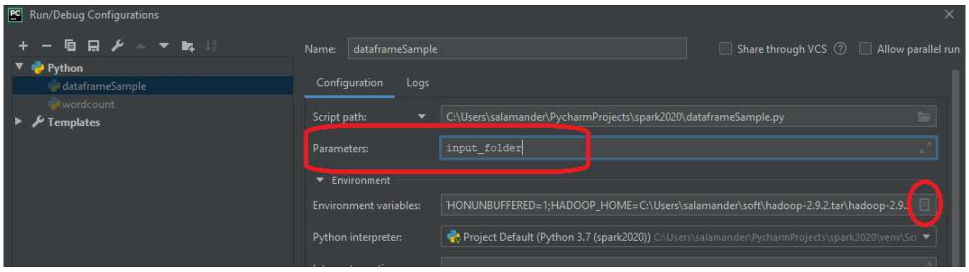
Exercise 8.1. Configuring PyCharm IDE for Spark Python

We will again use the Python PyCharm IDE to simplify working with Spark Python DataFrame scripts. We will download and run an example Spark DataFrame script.

- Open PyCharm and create a new Python project.
  - Similar to lab 7, create a new *VirtualEnv* and add the `pyspark==2.4.8` package
- Download the following Python Spark DataFrame example dataframe_example.py file and move it inside your PySpark project.
- Try to run the dataframe_example.py (You will get an error about the missing arguments, but it will generate a run configuration for it)
- Modify the run configuration of the dataframe_example.py script.



- Create a new folder for input files (It does not have to be inside your Spark Python project)
  - Download the dataset `beach-weather-stations-automated-sensors-1.csv` file and move it into the input folder
  - Add the path to the input folder as the first argument of the script, just like you did in the previous lab.


Run/Debug Configurations

Name: dataframeSample

Script path: C:\Users\salamander\PycharmProjects\spark2020\dataframeSample.py

Parameters: input_folder

Environment variables: HONUNBUFFERED=1;HADOOP_HOME=C:\Users\salamander\soft\hadoop-2.9.2.tar\hadoop-2.9..

Python interpreter: Project Default (Python 3.7 (spark2020)) C:\Users\salamander\PycharmProjects\spark2020\venv\Sc

- ONLY in Windows: Set up the HADOOP_HOME environment variable, just like we did in the last lab. It should link to the HADOOP folder you downloaded and unpacked, and used in lab 7!
  - NB! When using Windows, this Hadoop folder must include the winutils.exe and other files we copied inside the bin subfolder!
- Run the Python DataFrame script.
  - This example prints the results out directly to the console
  - If you want to store the results in a filesystem instead, you could use the `df.write.format("csv")` DataFrame command
  - This script controls how many arguments are given. You will have to modify this part if you want to add additional parameters, such as output folder location

NB! You may see errors like "ERROR:root:Exception while sending command." These errors are unrelated to the DataFrame code but seem to be an issue with properly closing the Spark session.

- You can ignore these errors as long as all the DataFrames are shown properly.

Exercise 8.2. Familiarizing with the example Spark DataFrame script

Let's take a look at the example dataframe_example.py script.

- Dataset files are loaded directly as a Spark DataFrame using the `spark.read` command.
  - Read command supports many different formats like csv, xml, json or plaintext.
  - We can automatically detect data types for columns by using the “inferSchema” option
  - In addition, when the CSV file contains a header line, which specifies the names of the column, we can use the “header” option to automatically read the column names from there

  -
    ```
        dataset = spark.read \
                      .option("inferSchema", True) \
                      .option("header", True) \
                      .csv(input_folder)
    ```
  - This means we do not need to worry about removing the header lines from input files.
- To print out the first 10 rows of the DataFrame without truncating any values (the second argument is set to `False`), the script uses the command:
  - `dataset.show(10, False)`
- To show DataFrame schema (structure), consisting of the names and types of all columns, the script uses:
  - `dataset.printSchema()`
- At the end of the script Spark session is stopped with `spark.stop()` command.
- NB! When modifying the script later, make sure you add any new code after the `spark.read`, but before the `spark.stop()` code lines.

Exercise 8.3. Extending the example DataFrame script

Let's take a look at and try out some typical DataFrame manipulation and data analytics operations on the dataset:

1. To filter the number of columns in the DataFrame, use the select command to select only the Station Name and Humidity columns, like this:
   - `result = dataset.select("Station Name", "Humidity")`
   - `result.show(10, False)`

2. To filter the content based on some values inside the DataFrame columns, we can use the filter command and provide a conditional statement as an argument to the filter command.
    - `result = dataset.filter("Humidity < 40")`
    - `result.show(10, False)`
    - The conditional statement supports the same kind of language as SQL conditional statements, and you can refer to the columns by their labels.
    - You can use backticks (`) around column labels when you need to address ones that include spaces, like this:
        - ``result = dataset.filter('Humidity < 40 and `Air Temperature` > 10')``
        - `result.show(10, False)`
3. Creating new Columns
    - Spark `withColumn(new_column_name, expression)` method can be used to create new columns.
    - For example, if we want to create a new column by multiplying two existing columns:
        - `dataset = dataset.withColumn("newColumn", dataset["Wet Bulb Temperature"] * dataset["Humidity"])`
            - In here, when we need to address a specific column in a typical python operation (multiplication), then we can use `dataset["Wet Bulb Temperature"]` to address the Wet Bulb Temperature column. Another way to address specific DataFrame columns is with a dot, like this: `dataset.newColumn`, but it does not work when column names include spaces!
4. We can also apply aggregation methods to compute statistics on the dataset through the `agg()` operation, like this:
    - `result = dataset.agg(sparkFun.avg("Humidity"))`
    - `result.show()`
        - The `agg()` function takes the aggregation function as an argument, which in this case is the average (avg) function. The average function takes the name of the column as an argument, specifying which column the aggregation function should be applied on.
        - NB! For the above command to work, you need to import additional functions from the pyspark.sql.functions library. Add the following line to the start of the script:
            - `import pyspark.sql.functions as sparkFun`
    - When we want to aggregate data separately for different weather stations, we can use the group by the statement:
        - `result = dataset.groupBy("Station Name").agg(sparkFun.avg("Humidity"))`
        - `result.show()`
    - It is also possible to apply multiple different aggregation functions inside a single agg() method call by separating the functions by commas. Spark will create a separate column in the resulting DataFrame for each of the specified functions.
    - A nice overview of available aggregation functions is here:
        - https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-aggregate-functions.html
5. Saving DataFrames as files.
    - To save the dataset into a file, we can use the write statement:

    -
      ```
      result.write.format("csv")  \
              .option("header", True) \
              .option("compression", "gzip") \
              .save("output_folder")
      ```
    - Similarly, you can change the save format into xml, json, or plain text.
    - To force Spark write output as a single file, you can use:
        - `result.coalesce(1).write.format("json").save(output_folder)`
        - `coalesce(N)` re-partitions the DataFrame or RDD into N partitions.
        - NB! But be careful when using coalesce(N); your program will crash if the whole DataFrame does not fit into the memory of N processes.

Individual task

- Using the Spark operations explained in this exercise, perform the following task:
    1. Compute the **Average Humidity** of all stations (group by stations) when the **Air Temperature** was higher than 20.
    2. Store the results as CSV files.

Exercise 8.4. Using Spark DataFrame API to perform simple statistical analysis

Using the knowledge gained up to this point, solve the following individual tasks using Apache Spark DataFrame API:

1. For each **Station**, compute the average **Solar Radiation**.
2. Let's also compute **minimum** and **maximum** in addition to average.
3. Let's now compute these statistics for each day and station.
    - You can extract the day value from the **Measurement Timestamp** field by using some of the available string manipulation functions in the pyspark.sql.functions library to remove everything but the date string
        - NB! It is suggested to **AVOID** using date parsing functions, as they appear to have difficulties with the format of date in this dataset. Use string manipulation functions instead
    - Spark DataFrame/SQL Functions - https://spark.apache.org/docs/latest/api/sql/index.html
4. Order the result by average solar radiation in descending order
    - **Answer:** What station had the highest average solar radiation on which day? What was the value?

If you need some more examples of which Spark DataFrame functions are available and how to use them, then the Spark Python API has a very nice overview of available operations with simple examples here:

- https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html#DataFrame-Creation

And DataFrame/sql functions here:

- https://spark.apache.org/docs/latest/api/sql/index.html

The resulting DataFrame, BEFORE sorting, should look something like this:

```
+--------------------+----------+------------------+----------------------+------------------+
|        Station Name|       day|Min Solar radiation|Average Solar radiation|Max Solar radiation|
+--------------------+----------+------------------+----------------------+------------------+
|Foster Weather St...|07/15/2015|                 0|     248.69565217391303|               843|
|Oak Street Weathe...|11/07/2015|                 1|      88.82608695652173|               535|
|Foster Weather St...|12/03/2015|                 0|     19.416666666666668|                87|
|Foster Weather St...|02/16/2016|                 0|      86.91304347826087|               708|
|Oak Street Weathe...|03/11/2016|                 2|     194.26315789473685|               717|
|Foster Weather St...|03/24/2016|                 0|                  29.05|               329|
|63rd Street Weath...|06/26/2016|                 0|                  323.0|              1123|
|63rd Street Weath...|06/27/2016|                 0|      321.0833333333333|               881|
|Foster Weather St...|08/17/2016|                 0|                    0.0|                 0|
|Oak Street Weathe...|11/04/2016|                 0|      85.21739130434783|               506|
+--------------------+----------+------------------+----------------------+------------------+
```

- NB! This is the output using a different dataset. Your output can be different from this result.

Bonus exercise

The goal of the bonus task is to investigate the Spark DataFrame-based Machine learning library, to cluster the dataset records into K different clusters using the Spark K-means clustering method, and visualize the results as a graph to the user.

This requires you to:

1. Import additional libraries that may be needed, such as pandas, matplotlib
2. Define features based on which the clustering is performed. You should use the "Humidity", "Wind Speed" columns.
   - It would also be good to filter out (can use filter()) very clear outliers from the source data.
3. Configure the kMeans model (set the value of k to be 4) and train/fit it to the dataset
4. Apply the kMeans model to the dataset to compute into which cluster each dataset record belongs to (this creates a prediction column, prediction = cluster id).
5. Convert the Spark DataFrame to Pandas DataFrame
6. Use Pandas DataFrame df.plot.scatter() method to generate a scatterplot graph that has features as axis' and cluster id as color.
7. Use mapPlotLib dataframe.show() to display the graph visually to the user.

Also, the dataset contains a large number of null values. Some machine learning methods can not properly deal with such values. To avoid removing all rows containing null values, we can convert all null values to 0 instead for the pure sake of simplicity. This can be done globally in the DataFrame with the following operation: `dataset = dataset.na.fill(0)`

- NB! You should note that most often, this is not a smart thing to do in machine learning, as assuming 0 values when data was actually missing may result in a completely wrong model!

Deliverables:

1. Python code
2. Output of the program (use show() to print out the snapshot of the dataframes)
3. Screenshot of the generated graph

Deliverables:

- Python script from Exercise 8.3
- Python script from the Exercise 8.4 (only the final version, no need to save intermediate versions of the script separately)
- Output of the Python scripts from Exercises 8.3, 8.4
  - The output should contain the resulting DataFrame content.
  - You can save the respective result as a CSV file or save df.show(20) command's output manually into a text file.
- Answer to the question raised at the end of Exercise 8.4.

Task	Lab 8 - spark dataframes
Current submission	ZIP (02:31 29.03.2023) If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File No file chosen
Comment	
	Submit

Potential issues and solutions

- **NB!**You are experiencing "java.lang.UnsatisfiedLinkError: org.apache.hadoop.io.nativeio.NativeIO$Windows.access(.....)":
  - Please try one of these two:
    1. Try downgrading to Spark 2.4.7 in your Python Project
       - Open project settings, where you added "pyspark", and choose "Specify Version" after opening the pyspark package details
    2. Use OS Environment variables instead of PyCharm run configuration of HADOOP_HOME. Since we will use set them at OS level , remove these from PyCharm run conf.
       - Set a new Windows Environment Variable HADOOP_HOME with the correct directory path for hadoop.
       - Append /bin; to your Windows User PATH, (replace with correct full path)
       - After above steps, restart PyCharm and try running the project again
- **NB!** You may see errors like:
  - `"ERROR:root:Exception while sending command."`
  - `"ConnectionRefusedError: [WinError 10061] No connection could be made because the target machine actively refused it"`
  - These errors are not related to the DataFrame code, but seems to be a issue about properly closing the Spark session.
  - You can ignore these errors as long as all the DataFrames are shown properly in the output.
- It is suggested to use Java 8 as the default Java in your computer.
- Be careful with the code indentation in Python script. Python is very strict when it comes to mixing tabs and spaces.
  - Check indentation manual for Python: https://docs.python.org/2.0/ref/indentation.html
- Issues with input path location in Windows
  - To avoid relative path errors in Windows you may have to specify a valid full path for Spark warehouse directory. Change the SparkSession line and add **spark.sql.warehouse.dir** configuration :

```
spark = SparkSession\
        .builder \
        .appName("DataFrame Example") \
        .config('spark.sql.warehouse.dir', 'file:///C:/Users/pjakovits/SparkWarehouse') \
        .getOrCreate()
```