DATA MANAGEMENT WITH PYSPARK

INTRODUCTION

This set of exercises will demonstrate how to import, clean, merge, and reshape a dataset using PySpark.

Our goals for this lab are:

- Import data
- Subset data
- Create appropriate dtypes
- Create new variables as needed
- Merge data

DATA USED IN THIS EXERCISE:

We will use the following datasets in this exercise:

- `data/manufacturing/faults_features.csv`
- `data/manufacturing/faults_labels.csv`
- `data/manufacturing/names_features.txt`
- `data/manufacturing/names_labels.txt`

Concept of the data:

- Some researchers conducted a range of experiments on various steel plates, noting seven classes of fault occurring.
- The `faults_features` lists the 27 features contributing to the observed fault type, plus a primary ID.
- The `faults_labels` table contains the associated fault type and corresponding primary ID.
- These data files have no header; the text files `names_features` and `names_labels` contain the names of the fields for each file, respectively.

QUESTIONS

1) Start by writing the import statements for PySpark. Also, set up the working directory to point at the folder you have established for this class. Confirm the working directory is what you expect.
2) Import the two datasets mentioned above (`faults_features` and `faults_labels`), along with the respective text files holding their field names:
    a. First inspect the text file with a text editor
    b. Import the datasets and save the result (hint: use `inferSchema=True`)
    c. Apply the names to the respective dataset. Because Spark DataFrames are immutable, you may need to create a new DataFrame for this
    d. We want this as a single DataFrame. Join the two sets together, on their primary ID, called `ID` in both sets (hint: you *may* use `.join()`)
    e. Check the resulting files: how many rows and columns?
3) Write some basic subset and manipulations for these data:
    a. Print the schema of the DataFrame
    b. Print the first 5 rows of the combined set using `.show()`
    c. Obtain a numerical summary of each variable (don't forget to use `.show()` to collect results from the distributed partitions)
    d. There were a lot of results for this. Select the `X_Minimum` column only (use `.select()`)
    e. Rename this column to `x_min` (using `.withColumnRenamed()`)
4) Perform some sorting and filtering:
    a. Sort the DataFrame by x_min, ascending and descending
    b. Filter the DataFrame to find entries where x_min is greater than 1000. Provide a count for the number of times this occurs. Give a summary of the basic descriptive statistics for x_min under this condition
5) Perform some data cleaning:
    a. The "Other_Faults" column should only contain 1 or 0. Determine the unique values of this field (.distinct())
    b. Set any other values to 0. There are a few ways to do this (you may like the PySpark analog to NumPy's where, called 'when,' from the pyspark.sql.functions library)
    c. Cast the cleaned column to be the same data type as the other fault labels (e.g. "Pastry")
    d. Check to see if there is any missing data. Quantify this, if appropriate

    e. Fill any missing values with 0. This is appropriate, because the researchers communicated the origin of the missing data to you when you asked them about it
    f. Create a new column, "named_faults," which is the logical inverse of "Other_Faults"
6) Perform some and uni- and bi-variate analysis:
    a. Group the DataFrame by this new column, "named_faults," and obtain a count within each level
    b. Create a table comparing the unique values of "named_faults" and "Other_Faults"
    c. Obtain the quartile ranges for x_min (hint: .approxQuantile())
    d. Obtain the Pearson correlation coefficient between x_min and X_Maximum (hint: .corr())
    e. Similarly, between x_min and Y_Minimum
    f. Obtain the covariance between "named_faults" and "Other_Faults" (.cov())
    g. Display a cross-tabulation between these two variables (.crosstab())
7) Perform more grouping operations:
    a. Show the counts for each level of "named_faults"
    b. Display the mean x_min value for each of these levels
    c. Do the same, but for the levels of the "Pastry" fault type
    d. Again, display the mean x_min value, but for levels grouped first by "named_faults" and then by "Pastry"
    e. Produce a pivot table showing mean x_min values for these groups against each other
8) Exporting and persisting PySpark DataFrames:
    a. Convert your main DataFrame from a Spark SQL to a pandas type (use .toPandas()). Note that in this case it is OK, because the resulting object will fit in memory
    b. Save your DataFrame to a parquet format (.write.parquet())
9) Don't forget to stop your Spark Session with spark.stop() !!!

