Guidelines for Coding Conventions in PySpark, Spark Java, and Spark Scala

Introduction:

Coding conventions are essential to ensure code readability, maintainability, and consistency across different modules and projects. In this document, we will outline coding conventions for PySpark, Spark Java, and Spark Scala.

Naming Conventions:

It is essential to follow consistent naming conventions for variables, functions, classes, and packages.

For PySpark, we suggest following PEP8 naming conventions:

- Variables and functions: lowercase, with underscores separating words (e.g., my_variable)
- Classes: CamelCase, starting with a capital letter (e.g., MyClass)
- Packages: lowercase, with dots separating words (e.g., my.package)

For Spark Java and Spark Scala, we suggest following the Java naming conventions:

- Variables and functions: camelCase, starting with a lowercase letter (e.g., myVariable)
- Classes: CamelCase, starting with a capital letter (e.g., MyClass)
- Packages: lowercase, with dots separating words (e.g., my.package)

Indentation and Spacing:

Proper indentation and spacing are essential to ensure code readability.

For PySpark, we suggest following PEP8 indentation and spacing conventions:

- Use 4 spaces for indentation
- Use a single space after commas and operators, but no spaces between the function name and opening parenthesis (e.g., my_function(arg1, arg2))

For Spark Java and Spark Scala, we suggest following the Java indentation and spacing conventions:

- Use 4 spaces for indentation
- Use a single space after commas and operators, but no spaces between the function name and opening parenthesis (e.g., myFunction(arg1, arg2))

Avoid Hardcoded Values:

Avoid hardcoded values in your code. Instead, use constants or configuration files to store them.

Miscellaneous Conventions:

- Use meaningful variable and function names.
  #Example :
  temperature_data = spark.read.csv(INPUT_FILE_PATH)

- Break long lines of code to make them more readable.
  # Example :
  my_long_variable_name = function1(argument1, argument2, argument3,
  argument4, argument5, argument6)
- Use comments to describe the code, especially for complex logic
  # Example
  # This code block does some complex logic to find the max temperature.
  max_temp = temperature_data.groupBy("location").agg(F.max("temperature"))

- Try to create function with max 20 lines of code, if it is more than that create a separate function

- Try to create function with max 20 lines of code, if it is more than that create a separate function
- A function can not accept more than 3 parameters