Module 4 Cheat Sheet: DataFrames and Spark SQL

Package/Method	Description	Code Example

appName()	A name for your job to display on the cluster web UI.	`from pyspark.sql import SparkSession`
		`spark = SparkSession.builder.appName("MyApp").getOrCreate()`
		Copied!

createDataFrame()	Used to load the data into a Spark DataFrame.	`from pyspark.sql import SparkSession`
		`spark = SparkSession.builder.appName("MyApp").getOrCreate()`
		`data = [("Jhon", 30), ("Peter", 25), ("Bob", 35)]`
		`columns = ["name", "age"]`
		Copied!
		Creating a DataFrame
		`df = spark.createDataFrame(data, columns)`
		Copied!

createTempView()	Create a temporary view that can later be used to query the data. The only required parameter is the name of the view.	`df.createOrReplaceTempView("cust_tbl")` Copied!
fillna()	Used to replace NULL/None values on all or selected multiple DataFrame columns with either zero (0), empty string, space, or any constant literal values.	Replace NULL/None values in a DataFrame `filled_df = df.fillna(0)` Copied! Replace with zero
filter()	Returns an iterator where the items are filtered through a function to test if the item is accepted or not.	`filtered_df = df.filter(df['age'] > 30)` Copied!
getOrCreate()	Get or instantiate a SparkContext and register it as a singleton object.	`spark = SparkSession.builder.getOrCreate()` Copied!
groupby()	Used to collect the identical data into groups on DataFrame and perform count, sum, avg, min, max functions on the grouped data.	Grouping data and performing aggregation `grouped_df = df.groupBy("age").agg({"age": "count"})` Copied!
head()	Returns the first *n* rows for the object based on position.	Returning the first 5 rows `first_5_rows = df.head(5)` Copied!
import	Used to make code from one module accessible in another. Python imports are crucial for a successful code structure. You may reuse code and keep your projects manageable by using imports effectively, which can increase your productivity.	`from pyspark.sql import SparkSession` Copied!

pd.read_csv()	Required to access data from the CSV file from Pandas that retrieves data in the form of the data frame.	`import pandas as pd`
		Copied!
		Reading data from a CSV file into a DataFrame
		`df_from_csv = pd.read_csv("data.csv")`
		Copied!

pip	To ensure that requests will function, the pip program searches for the package in the Python Package Index (PyPI), resolves any dependencies, and installs everything in your current Python environment.	`pip list` Copied!
pip install	The pip install <package> command looks for the latest version of the package and	`pip install pyspark`

Package/Method	Description	Code Example
	installs it.	Copied!
printSchema()	Used to print or display the schema of the DataFrame or data set in tree format along with the column name and data type. If you have a DataFrame or data set with a nested structure, it displays the schema in a nested tree format.	`df.printSchema()` Copied!

rename()	Used to change the row indexes and the column labels.	`import pandas as pd`
		Copied!
		Create a sample DataFrame
		`data = {'A': [1, 2, 3], 'B': [4, 5, 6]}`
		`df = pd.DataFrame(data)`
		Copied!
		Rename columns
		`df = df.rename(columns={'A': 'X', 'B': 'Y'})`
		Copied!
		The columns 'A' and 'B' are now renamed to 'X' and 'Y'
		`print(df)`
		Copied!

select()	Used to select one or multiple columns, nested columns, column by index, all columns from the list, by regular expression from a DataFrame. select() is a transformation function in Spark and returns a new DataFrame with the selected columns.	`selected_df = df.select('name', 'age')` Copied!
show()	Spark DataFrame show() is used to display the contents of the DataFrame in a table row and column format. By default, it shows only twenty rows, and the column values are truncated at twenty characters.	`df.show()` Copied!

sort()	Used to sort DataFrame by ascending or descending order based on single or multiple columns.	Sorting DataFrame by a column in ascending order
		`sorted_df = df.sort("age")`
		Copied!
		Sorting DataFrame by multiple columns in descending order
		`sorted_df_desc = df.sort(["age", "name"], ascending=[False, True])`
		Copied!

SparkContext()	It is an entry point to Spark and is defined in org.apache.spark package since version 1.x and used to programmatically create Spark RDD, accumulators, and broadcast variables on the cluster.	`from pyspark import SparkContext`
		Copied!
		Creating a SparkContext
		`sc = SparkContext("local", "MyApp")`
		Copied!

SparkSession	It is an entry point to Spark, and creating a SparkSession instance would be the first statement you would write to the program with RDD, DataFrame, and dataset	`from pyspark.sql import SparkSession`
		Copied!
		Creating a SparkSession
		`spark = SparkSession.builder.appName("MyApp").getOrCreate()`
		Copied!

Package/Method	Description	Code Example
spark.read.json()	Spark SQL can automatically infer the schema of a JSON data set and load it as a DataFrame. The read.json() function loads data from a directory of JSON files where each line of the files is a JSON object. Note that the file offered as a JSON file is not a typical JSON file.	`json_df = spark.read.json("customer.json")` Copied!

spark.sql()	To issue any SQL query, use the sql() method on the SparkSession instance. All spark.sql queries executed in this manner return a DataFrame on which you may perform further Spark operations if required.	`result = spark.sql("SELECT name, age FROM cust_tbl WHERE age > 30")`
		`result.show()`
		Copied!

spark.udf.register()	In PySpark DataFrame, it is used to register a user-defined function (UDF) with Spark, making it accessible for use in Spark SQL queries. This allows you to apply custom logic or operations to DataFrame columns using SQL expressions.	Registering a UDF (User-defined Function)
		`from pyspark.sql.functions import udf`
		`from pyspark.sql.types import StringType`
		`def my_udf(value):`
		`return value.upper()`
		`spark.udf.register("my_udf", my_udf, StringType())`
		Copied!

where()	Used to filter the rows from DataFrame based on the given condition. Both filter() and where() functions are used for the same purpose.	Filtering rows based on a condition `filtered_df = df.where(df['age'] > 30)` Copied!

withColumn()	Transformation function of DataFrame used to change the value, convert the data type of an existing column, create a new column, and many more.	Adding a new column and performing transformations
		`from pyspark.sql.functions import col`
		`new_df = df.withColumn("age_squared", col("age") ** 2)`
		Copied!

withColumnRenamed()	Returns a new DataFrame by renaming an existing column.	Renaming an existing column `renamed_df = df.withColumnRenamed("age", "years_old")` Copied!

Skills Network