[Apache Spark with Scala] {CheatSheet}

1. Spark Session and Context

- **Creating Spark Session**: `val spark = SparkSession.builder.appName("SparkApp").getOrCreate()`
- **Accessing Spark Context**: `val sc = spark.sparkContext`

2. Data Loading and Writing

- **Reading a CSV File**: `val df = spark.read.format("csv").option("header", "true").load("path/to/csv")`
- **Writing DataFrame to Parquet**: `df.write.parquet("path/to/output")`
- **Reading JSON File**: `val df = spark.read.json("path/to/json")`
- **Writing DataFrame to JSON**: `df.write.json("path/to/output")`

3. DataFrame Operations

- **Selecting Columns**: `df.select("column1", "column2")`
- **Filtering Rows**: `df.filter($"column" > value)`
- **Adding a New Column**: `df.withColumn("newColumn", $"existingColumn" + 1)`
- **Renaming a Column**: `df.withColumnRenamed("oldName", "newName")`
- **Dropping a Column**: `df.drop("column")`

4. Aggregation Functions

- **Group By and Aggregate**: `df.groupBy("column").agg(sum("otherColumn"))`
- **Calculating Average**: `df.groupBy("column").avg()`
- **Calculating Maximum**: `df.groupBy("column").max()`
- **Calculating Minimum**: `df.groupBy("column").min()`
- **Counting Values**: `df.groupBy("column").count()`

5. Join Operations

- **Inner Join**: `df1.join(df2, df1("id") === df2("id"))`
- **Left Outer Join**: `df1.join(df2, df1("id") === df2("id"), "left_outer")`
- **Right Outer Join**: `df1.join(df2, df1("id") === df2("id"), "right_outer")`
- **Full Outer Join**: `df1.join(df2, df1("id") === df2("id"), "full_outer")`

6. RDD Operations

By: Waleed Mousa

- **Creating an RDD**: `val rdd = sc.parallelize(Seq(1, 2, 3))`
- **Transforming with map**: `val rdd2 = rdd.map(x => x * x)`
- **Filtering Data**: `val filteredRdd = rdd.filter(x => x > 1)`
- **FlatMap Operation**: `val flatRdd = rdd.flatMap(x => Seq(x, x*2))`
- **Reducing Elements**: `val sum = rdd.reduce((x, y) => x + y)`

7. Working with Key-Value Pairs

- **Creating Pair RDD**: `val pairRdd = rdd.map(x => (x, x*2))`
- **Reducing by Key**: `val reduced = pairRdd.reduceByKey((x, y) => x + y)`
- **Grouping by Key**: `val grouped = pairRdd.groupByKey()`
- **Sorting by Key**: `val sorted = pairRdd.sortByKey()`
- **Map Values**: `val mappedValues = pairRdd.mapValues(x => x + 1)`

8. Data Partitioning

- **Repartitioning RDD**: `val repartitionedRdd = rdd.repartition(4)`
- **Coalescing RDD**: `val coalescedRdd = rdd.coalesce(2)`

9. SQL Queries on DataFrames

- **Creating Temp View**: `df.createOrReplaceTempView("tableView")`
- **Running SQL Query**: `val result = spark.sql("SELECT * FROM tableView WHERE column > value")`

10. UDFs and UDAFs

- **Defining UDF**: `val myUDF = udf((x: Int) => x * 2)`
- **Using UDF in DataFrame**: `df.withColumn("newCol", myUDF($"column"))`
- **Registering UDF for SQL**: `spark.udf.register("myUDF", myUDF)`
- **Using UDAF**: `val myUDAF = new MyUDAF(); df.groupBy("column").agg(myUDAF($"otherColumn"))`

11. Window Functions

- **Using Window Function**: `val windowSpec = Window.partitionBy("column").orderBy("otherColumn"); df.withColumn("rank", rank().over(windowSpec))`

12. Handling Missing and Null Values

By: Waleed Mousa

- **Filling Null Values**: `df.na.fill(0)`
- **Dropping Rows with Null**: `df.na.drop()`
- **Replacing Values**: `df.na.replace("column", Map("oldValue" -> "newValue"))`

13. Handling JSON and Complex Data Types

- **Extracting JSON Fields**: `df.withColumn("extractedField", get_json_object($"jsonColumn", "$.fieldName"))`
- **Working with Structs**: `df.select($"structColumn.fieldName")`

14. Reading and Writing Data from Various Sources

- **Reading from Parquet**: `val df = spark.read.parquet("path/to/parquet")`
- **Writing to CSV**: `df.write.format("csv").save("path/to/output")`
- **Reading from JDBC**: `val jdbcDF = spark.read.format("jdbc").option("url", jdbcUrl).option("dbtable", "tableName").load()`
- **Writing to JDBC**: `df.write.format("jdbc").option("url", jdbcUrl).option("dbtable", "tableName").save()`

15. Machine Learning with MLlib

- **Vector Assembler**: `val assembler = new VectorAssembler().setInputCols(Array("col1", "col2")).setOutputCol("features")`
- **Standard Scaler**: `val scaler = new StandardScaler().setInputCol("features").setOutputCol("scaledFeatures").fit(df)`
- **Linear Regression Model**: `val lr = new LinearRegression(); val model = lr.fit(df)`
- **KMeans Clustering**: `val kmeans = new KMeans().setK(2); val model = kmeans.fit(df)`

16. Streaming Data

- **Structured Streaming from Socket**: `val stream = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).load()`
- **Writing Streaming Data**: `stream.writeStream.format("console").start()`
- **Kafka Source for Streaming**: `val kafkaStream = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe", "topic").load()`

By: Waleed Mousa

- **Writing to Kafka in Streaming**: `stream.writeStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("topic", "outputTopic").start()`

17. Performance Tuning

- **Broadcast Variables**: `val broadcastVar = sc.broadcast(Array(1, 2, 3))`
- **Accumulators**: `val accumulator = sc.longAccumulator("MyAccumulator")`
- **Caching Data**: `df.cache()`
- **Checkpointing RDD**: `rdd.checkpoint()`

18. Advanced DataFrame Transformations

- **Pivoting Data**: `df.groupBy("column").pivot("pivotColumn").agg(sum("value"))`
- **Explode Array Column**: `df.withColumn("exploded", explode($"arrayColumn"))`
- **Rollup**: `df.rollup("col1", "col2").agg(sum("value"))`
- **Cube**: `df.cube("col1", "col2").agg(sum("value"))`

19. Handling Large Datasets

- **Broadcast Join Hint**: `df1.join(broadcast(df2), Seq("id"), "inner")`
- **Avoiding Shuffle with Coalesce**: `df.coalesce(1)`
- **Repartitioning for Parallelism**: `df.repartition(10)`

20. Dealing with Text Data

- **Regular Expression with rlike**: `df.filter($"column".rlike("regex"))`
- **Splitting Strings**: `df.withColumn("splitCol", split($"stringCol", "delimiter"))`
- **Concatenating Strings**: `df.withColumn("concatenated", concat_ws("-", $"col1", $"col2"))`

21. Working with Dates and Times

- **Current Date and Timestamp**: `df.withColumn("currentDate", current_date()).withColumn("currentTimestamp", current_timestamp())`
- **Date Formatting**: `df.withColumn("formattedDate", date_format($"dateCol", "yyyy-MM-dd"))`
- **Date Arithmetic**: `df.withColumn("datePlusDays", expr("dateCol + interval 5 days"))`

By: Waleed Mousa

22. Advanced SQL Queries

- **Registering DataFrame as a Temp View for SQL**: `df.createOrReplaceTempView("tempView"); spark.sql("SELECT * FROM tempView WHERE column > value")`
- **Complex SQL Query**: `spark.sql("SELECT col1, col2, SUM(col3) FROM tempView GROUP BY col1, col2")`

23. Error Handling and Debugging

- **Catching Exceptions in DataFrame Operations**: `Try(df.select("invalidColumn")) match { case Success(df) => df case Failure(e) => e.printStackTrace() }`

24. Interoperability with RDDs and DataFrames

- **Converting RDD to DataFrame**: `val df = rdd.toDF("column1", "column2")`
- **Converting DataFrame to RDD**: `val rdd = df.rdd`

25. External Data Sources and Formats

- **Reading Data from Avro Files**: `val df = spark.read.format("avro").load("path/to/avro")`
- **Writing Data to Avro**: `df.write.format("avro").save("path/to/output")`

26. Spark MLlib: Feature Transformers

- **StringIndexer for Categorical Features**: `val indexer = new StringIndexer().setInputCol("category").setOutputCol("categoryIndex")`
- **OneHotEncoder for Categorical Encoding**: `val encoder = new OneHotEncoder().setInputCol("index").setOutputCol("vector")`

27. Spark MLlib: Classification and Regression

- **Decision Tree Classifier**: `val dt = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")`
- **Linear Regression**: `val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3)`

28. Spark MLlib: Clustering

By: Waleed Mousa

- **KMeans Clustering**: `val kmeans = new KMeans().setK(3).setFeaturesCol("features")`
- **Gaussian Mixture Model**: `val gmm = new GaussianMixture().setK(3).setFeaturesCol("features")`

29. Spark MLlib: Model Evaluation

- **Binary Classification Evaluator**: `val evaluator = new BinaryClassificationEvaluator()`
- **Regression Evaluator (e.g., RMSE)**: `val regEvaluator = new RegressionEvaluator().setMetricName("rmse")`

30. Spark GraphX: Graph Processing

- **Creating a Graph from RDDs**: `val graph = Graph(verticesRDD, edgesRDD)`
- **Applying PageRank Algorithm**: `val ranks = graph.pageRank(0.0001).vertices`

31. Spark GraphX: Graph Algorithms

- **Connected Components**: `val cc = graph.connectedComponents().vertices`
- **Triangle Counting**: `val triangles = graph.triangleCount().vertices`

32. Working with Accumulators

- **Creating a Long Accumulator**: `val accumulator = sc.longAccumulator("MyAccumulator")`
- **Using Accumulator in RDD Operations**: `rdd.foreach(x => accumulator.add(x))`

33. Configurations and Tuning

- **Setting Dynamic Allocation**: `spark.conf.set("spark.dynamicAllocation.enabled", "true")`
- **Configuring Serialization**: `spark.conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")`

34. Spark Streaming

- **Creating DStream from Socket**: `val stream = ssc.socketTextStream("localhost", 9999)`
- **Stateful Transformation in Streaming**: `val stateDStream = stream.updateStateByKey(updateFunction)`

By: Waleed Mousa

35. Structured Streaming

- **Reading Stream from Kafka**: `val stream = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "host:port").option("subscribe", "topic").load()`
- **Writing Stream to Console**: `val query = stream.writeStream.outputMode("complete").format("console").start()`