# [ ELT Using PySpark ] ( CheatSheet )

## 1. Basic DataFrame Operations

- **Creating a DataFrame**: `spark.createDataFrame(data)`
- **Reading a CSV File**: `spark.read.csv("path/to/file.csv")`
- **Reading a JSON File**: `spark.read.json("path/to/file.json")`
- **Reading a Parquet File**: `spark.read.parquet("path/to/file.parquet")`
- **Writing DataFrame to CSV**: `df.write.csv("path/to/output")`
- **Writing DataFrame to JSON**: `df.write.json("path/to/output")`
- **Writing DataFrame to Parquet**: `df.write.parquet("path/to/output")`
- **Show DataFrame**: `df.show()`
- **Print Schema**: `df.printSchema()`
- **Column Selection**: `df.select("column1", "column2")`
- **Filtering Rows**: `df.filter(df["column"] > value)`
- **Adding a New Column**: `df.withColumn("newColumn", expression)`
- **Renaming a Column**: `df.withColumnRenamed("oldName", "newName")`
- **Dropping a Column**: `df.drop("column")`
- **Describe Data**: `df.describe().show()`
- **Counting Rows**: `df.count()`
- **Aggregating Data**: `df.groupBy("column").agg({"column": "sum"})`
- **Sorting Data**: `df.sort("column")`
- **Distinct Values**: `df.select("column").distinct()`
- **Sample Data**: `df.sample(withReplacement, fraction, seed)`

## 2. Advanced Data Manipulation

- **Joining DataFrames**: `df1.join(df2, "column")`
- **Union DataFrames**: `df1.union(df2)`
- **Pivot Table**: `df.groupBy("column").pivot("pivot_column").agg(count("another_column"))`
- **Window Functions**: `from pyspark.sql import Window; windowSpec = Window.partitionBy("column")`
- **DataFrame to RDD Conversion**: `df.rdd`
- **RDD to DataFrame Conversion**: `rdd.toDF()`
- **Caching a DataFrame**: `df.cache()`
- **Uncaching a DataFrame**: `df.unpersist()`
- **Collecting Data to Driver**: `df.collect()`

By: Waleed Mousa

- **Broadcast Join**: `from pyspark.sql.functions import broadcast; df1.join(broadcast(df2), "column")`

## 3. Handling Missing and Duplicate Data

- **Dropping Rows with Null Values**: `df.na.drop()`
- **Filling Null Values**: `df.na.fill(value)`
- **Dropping Duplicate Rows**: `df.dropDuplicates()`
- **Replacing Values**: `df.na.replace(["old_value"], ["new_value"])`

## 4. Data Transformation

- **UDF (User Defined Function)**: `from pyspark.sql.functions import udf; udf_function = udf(lambda z: custom_function(z))`
- **String Operations**: `from pyspark.sql.functions import lower, upper; df.select(upper(df["column"]))`
- **Date and Time Functions**: `from pyspark.sql.functions import current_date, current_timestamp; df.select(current_date())`
- **Numeric Functions**: `from pyspark.sql.functions import abs, sqrt; df.select(abs(df["column"]))`
- **Conditional Expressions**: `from pyspark.sql.functions import when; df.select(when(df["column"] > value, "true").otherwise("false"))`
- **Type Casting**: `df.withColumn("column", df["column"].cast("new_type"))`
- **Explode Function (Array to Rows)**: `from pyspark.sql.functions import explode; df.withColumn("exploded_column", explode(df["array_column"]))`
- **Pandas UDF**: `from pyspark.sql.functions import pandas_udf; @pandas_udf("return_type") def pandas_function(col1, col2): return operation`
- **Aggregating with Custom Functions**: `df.groupBy("column").agg(custom_agg_function(df["another_column"]))`
- **Window Functions (Rank, Lead, Lag)**: `from pyspark.sql.functions import rank, lead, lag; windowSpec = Window.orderBy("column"); df.withColumn("rank", rank().over(windowSpec))`
- **Handling JSON Columns**: `from pyspark.sql.functions import from_json, schema_of_json; df.withColumn("parsed_json", from_json(df["json_column"], schema_of_json))`

## 5. Data Profiling

- **Column Value Counts**: `df.groupBy("column").count()`
- **Summary Statistics for Numeric Columns**: `df.describe()`

- **Correlation Between Columns**: `df.stat.corr("column1", "column2")`
- **Crosstabulation and Contingency Tables**: `df.stat.crosstab("column1", "column2")`
- **Frequent Items in Columns**: `df.stat.freqItems(["column1", "column2"])`
- **Approximate Quantile Calculation**: `df.approxQuantile("column", [0.25, 0.5, 0.75], relativeError)`

## 6. Data Visualization (Integration with other libraries)

- **Convert to Pandas for Visualization**: `df.toPandas().plot(kind='bar')`
- **Histograms using Matplotlib**: `df.toPandas()["column"].hist()`
- **Box Plots using Seaborn**: `import seaborn as sns; sns.boxplot(x=df.toPandas()["column"])`
- **Scatter Plots using Matplotlib**: `df.toPandas().plot.scatter(x='col1', y='col2')`

## 7. Data Import/Export

- **Reading Data from JDBC Sources**: `spark.read.format("jdbc").options(url="jdbc_url", dbtable="table_name").load()`
- **Writing Data to JDBC Sources**: `df.write.format("jdbc").options(url="jdbc_url", dbtable="table_name").save()`
- **Reading Data from HDFS**: `spark.read.text("hdfs://path/to/file")`
- **Writing Data to HDFS**: `df.write.save("hdfs://path/to/output")`
- **Creating DataFrames from Hive Tables**: `spark.table("hive_table_name")`

## 8. Working with Large Data

- **Partitioning Data**: `df.repartition(numPartitions)`
- **Coalesce Partitions**: `df.coalesce(numPartitions)`
- **Reading Data in Chunks**: `spark.read.option("maxFilesPerTrigger", 1).csv("path/to/file.csv")`
- **Optimizing Data for Skewed Joins**: `df.repartition("skewed_column")`
- **Handling Data Skew in Joins**: `df1.join(df2.hint("broadcast"), "column")`

## 9. Spark SQL

- **Running SQL Queries on DataFrames**: `df.createOrReplaceTempView("table"); spark.sql("SELECT * FROM table")`

- **Registering UDF for SQL Queries**: `spark.udf.register("udf_name", lambda x: custom_function(x))`
- **Using SQL Functions in DataFrames**: `from pyspark.sql.functions import expr; df.withColumn("new_column", expr("SQL_expression"))`

## 10. Machine Learning and Advanced Analytics

- **VectorAssembler for Feature Vectors**: `from pyspark.ml.feature import VectorAssembler; assembler = VectorAssembler(inputCols=["col1", "col2"], outputCol="features")`
- **StandardScaler for Feature Scaling**: `from pyspark.ml.feature import StandardScaler; scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")`
- **Building a Machine Learning Pipeline**: `from pyspark.ml import Pipeline; pipeline = Pipeline(stages=[assembler, scaler, ml_model])`
- **Train-Test Split**: `train, test = df.randomSplit([0.7, 0.3])`
- **Model Fitting and Predictions**: `model = pipeline.fit(train); predictions = model.transform(test)`
- **Cross-Validation for Model Tuning**: `from pyspark.ml.tuning import CrossValidator; crossval = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid)`
- **Hyperparameter Tuning**: `from pyspark.ml.tuning import ParamGridBuilder; paramGrid = ParamGridBuilder().addGrid(model.param, [value1, value2]).build()`

## 11. Graph and Network Analysis

- **Creating a GraphFrame**: `from graphframes import GraphFrame; g = GraphFrame(vertices_df, edges_df)`
- **Running Graph Algorithms**: `results = g.pageRank(resetProbability=0.15, maxIter=10)`
- **Subgraphs and Motif Finding**: `g.find("(a)-[e]->(b); (b)-[e2]->(a)")`

## 12. Streaming Data Processing

- **Reading from a Stream**: `spark.readStream.format("source").load()`
- **Writing to a Stream**: `query = df.writeStream.outputMode("append").format("console").start()`
- **Window Operations on Streaming Data**: `df.groupBy(window(df["timestamp"], "1 hour")).count()`

- **Handling Late Data and Watermarks**: `df.withWatermark("timestamp", "2 hours")`
- **Triggering Streaming**: `df.writeStream.trigger(processingTime='1 hour').start()`
- **Streaming Aggregations**: `df.groupBy("group_column").agg({"value": "sum"})`
- **Reading from Kafka**: `spark.readStream.format("kafka").option("kafka.bootstrap.servers", "host:port").option("subscribe", "topic").load()`
- **Writing to Kafka**: `df.writeStream.format("kafka").option("kafka.bootstrap.servers", "host:port").option("topic", "topic").start()`

## 13. Advanced Dataframe Transformations

- **Handling Complex Data Types (Arrays, Maps)**: `df.selectExpr("explode(array_column) as value")`
- **Flattening Nested Structures**: `df.selectExpr("struct_col.*")`
- **Pivoting and Unpivoting Data**: `df.groupBy("group_col").pivot("pivot_col").sum()`
- **Creating Buckets or Bins**: `from pyspark.ml.feature import Bucketizer; bucketizer = Bucketizer(splits=[0, 10, 20], inputCol="feature", outputCol="bucketed_feature")`
- **Normalization of Data**: `from pyspark.ml.feature import Normalizer; normalizer = Normalizer(inputCol="features", outputCol="normFeatures", p=1.0)`

## 14. Data Quality and Validation

- **Data Integrity Checks**: `df.checkpoint()`
- **Schema Validation**: `df.schema == expected_schema`
- **Data Completeness Validation**: `df.count() == expected_count`
- **Column Value Range Validation**: `df.filter((df["column"] >= lower_bound) & (df["column"] <= upper_bound))`

## 15. Integration with Other Data Systems

- **Reading from Hive Tables**: `spark.sql("SELECT * FROM hive_table")`
- **Writing to Hive Tables**: `df.write.saveAsTable("hive_table")`
- **Connecting to External Databases**: `df.write.jdbc(url, table, properties)`
- **Using Hadoop File System (HDFS)**: `df.write.save("hdfs://path/to/save")`

## 16. Performance Tuning and Optimization

- **Broadcast Variables for Join Optimization**: `from pyspark.sql.functions import broadcast; df1.join(broadcast(df2), "join_column")`
- **Caching Intermediate Data**: `df.cache()`
- **Repartitioning for Parallel Processing**: `df.repartition(num_partitions)`
- **Avoiding Shuffle and Spill to Disk**: `df.coalesce(num_partitions)`
- **Tuning Spark Configuration and Parameters**: `spark.conf.set("spark.executor.memory", "2g")`

## 17. Exploratory Data Analysis Techniques

- **Histograms for Exploring Distributions**: `df.select('column').rdd.flatMap(lambda x: x).histogram(10)`
- **Quantile and Percentile Analysis**: `df.approxQuantile("column", [0.25, 0.5, 0.75], 0.0)`
- **Exploring Data with Spark SQL**: `spark.sql("SELECT * FROM df_table").show()`
- **Calculating Covariance and Correlation**: `df.stat.cov("col1", "col2")`, `df.stat.corr("col1", "col2")`

## 18. Dealing with Different Data Formats

- **Handling Avro Files**: `df.write.format("avro").save("path")`, `spark.read.format("avro").load("path")`
- **Dealing with ORC Files**: `df.write.orc("path")`, `spark.read.orc("path")`
- **Handling XML Data**: (Using spark-xml library)
- **Dealing with Binary Files**: `spark.read.format("binaryFile").load("path")`

## 19. Handling Geospatial Data

- **Using GeoSpark for Geospatial Operations**: (Integrating with GeoSpark library)
- **Geospatial Joins and Calculations**: (Using location-based joins and UDFs for distance calculations)

## 20. Time Series Data Handling

- **Time Series Resampling and Aggregation**: `df.groupBy(window(df["timestamp"], "1 hour")).agg({"value": "mean"})`

- **Time Series Window Functions**: `from pyspark.sql.functions import window; df.groupBy(window("timestamp", "1 hour")).mean()`

## 21. Advanced Machine Learning Operations

- **Custom Machine Learning Models with MLlib**: `from pyspark.ml.classification import LogisticRegression; lr = LogisticRegression()`
- **Text Analysis with MLlib**: `from pyspark.ml.feature import Tokenizer; tokenizer = Tokenizer(inputCol="text", outputCol="words")`
- **Model Evaluation and Metrics**: `from pyspark.ml.evaluation import BinaryClassificationEvaluator; evaluator = BinaryClassificationEvaluator()`
- **Model Persistence and Loading**: `model.save("path")`, `ModelType.load("path")`

## 22. Graph Analysis with GraphFrames

- **Creating GraphFrames for Network Analysis**: `from graphframes import GraphFrame; g = GraphFrame(vertices_df, edges_df)`
- **Running Graph Algorithms (e.g., PageRank, Shortest Paths)**: `g.pageRank(resetProbability=0.15, maxIter=10)`

## 23. Custom Transformation and UDFs

- **Applying Custom Transformations**: `df.withColumn("custom_col", custom_udf("column"))`
- **Vector Operations for ML Features**: `from pyspark.ml.linalg import Vectors; df.withColumn("vector_col", Vectors.dense("column"))`

## 24. Logging and Monitoring

- **Logging Operations in Spark**: `spark.sparkContext.setLogLevel("WARN")`

## 25. Best Practices and Patterns

- **Following Data Partitioning Best Practices**: (Optimizing partition strategy for data size and operations)
- **Efficient Data Serialization**: (Using Kryo serialization for performance)
- **Optimizing Data Locality**: (Ensuring data is close to computation resources)
- **Error Handling and Recovery Strategies**: (Implementing try-catch logic and checkpointing)

## 26. Security and Compliance

- **Data Encryption and Security**: (Configuring Spark with encryption and security features)
- **GDPR Compliance and Data Anonymization**: (Implementing data masking and anonymization)

## 27. Advanced Data Science Techniques

- **Deep Learning Integration (e.g., with TensorFlow)**: (Using Spark with TensorFlow for distributed deep learning)
- **Complex Event Processing in Streams**: (Using structured streaming for event pattern detection)

## 28. Cloud Integration

- **Running Spark on Cloud Platforms (e.g., AWS, Azure, GCP)**: (Setting up Spark clusters on cloud services)
- **Integrating with Cloud Storage Services**: (Reading and writing data to cloud storage like S3, ADLS, GCS)