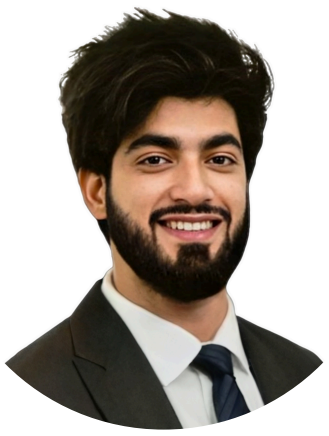

Tajamul Khan

PySpark Cheat Sheet

@Tajamulkhann

PySpark ?

PySpark is a tool that lets you use **Python** to work with **big data** using **Apache Spark**. It helps you process huge amounts of data quickly and in parallel across many computers.

Think of it like **pandas** for **big data** — but faster and built for scale.

You can use PySpark to:

- Clean and analyze large datasets
- Run SQL queries on big data
- Build machine learning models
- Handle real-time data

```
from pyspark.sql import SparkSession
spark =
SparkSession.builder.appName("App").
getOrCreate()
```

Reading & Writing Data

- **spark.read.csv("file.csv", header=True, inferSchema=True):** Read CSV with headers and infer schema.
- **spark.read.json("file.json"):** Read JSON file into DataFrame.
- **spark.read.parquet("file.parquet") :** Read Parquet format file.
- **spark.read.option("multiLine", True).json("file.json"):** Handle multi-line JSON.
- **spark.read.text("file.txt"):** Read a plain text file.
- **spark.read.format("jdbc").options(...).load():** Load data from JDBC source.
- **df.write.csv("output.csv", header=True):** Write to CSV.
- **df.write.mode("overwrite").parquet ("out.parquet"):** Overwrite and write to Parquet.

@Tajamulkhann

Data Exploration

- **df.show():** Display first 20 rows.
- **df.show(10, truncate=False):** Show 10 rows with full column values.
- **df.printSchema():** Print schema of DataFrame.
- **df.describe().show():** Summary stats for numeric columns.
- **df.summary().show():** Count, mean, stddev, min, max.
- **df.columns:** List of column names.
- **df.dtypes:** Get column names and their data types.
- **df.count():** Total number of rows.

@Tajamulkhann

Data Cleaning

- **df.dropna():** Drop rows with any nulls.
- **df.dropna(how="all"):** Drop rows where all values are null.
- **df.dropna(subset=["col1", "col2"]):** Drop rows with nulls in specific columns.
- **df.fillna(0):** Replace all nulls with 0.
- **df.fillna({"col": "missing"}):** Replace nulls in a column with specific value.
- **df.dropDuplicates():** Remove duplicate rows.
- **df.dropDuplicates(["col1", "col2"]):** Remove duplicates based on columns.
- **df = df.withColumn("col", df["col"].cast("integer")):** Convert data type.
- **df = df.filter(df["col"] <= 1000):** Remove outliers conditionally.

@Tajamulkhann

Data Manipulation

- **df.withColumn("d_col", df["col"] * 2):** Create new column with transformation.
- **df.withColumn("log_col", F.log(df["col"])):** Log transformation.
- **df.withColumn("flag", F.when(df["col"] > 100, 1).otherwise(0)):** Conditional flag column.
- **df.withColumnRenamed("old", "new"):** Rename a column.
- **df.selectExpr("col1 + col2 as total"):** Use SQL expression to manipulate columns.
- **df = df.drop("col1", "col2"):** Drop multiple columns.
- **df = df.select(F.col("col1"), F.col("col2")):** Select multiple columns using ‘col’.
- **df = df.withColumn("day", F.dayofmonth("date_col")):** Extract day from date column.

@Tajamulkhann

Filtering & Conditions

- **`df.filter(df["col"] > 50)`:** Filter rows where column > 50.
- **`df.where(df["status"] == "active")`:** Filter rows using `where`.
- **`df.filter((df["col1"] > 10) & (df["col2"] < 100))`:** Filter with multiple conditions.
- **`df.filter(df["col"].isin("A", "B"))`:** Filter with multiple matching values.
- **`df.filter(df["col"].isNotNull())`:** Keep rows with non-null values.
- **`df.withColumn("category", F.when(df["score"] > 80, "High").otherwise("Low"))`:** Categorize values.
- **`df.where(~df["col"].isin("A", "B"))`:** Filter rows not in list.
- **`df = df.limit(100)`:** Limit rows for preview or sampling.

@Tajamulkhann

Aggregation & Grouping

- **`df.groupBy("col").count()`**: Count rows per group.
- **`df.groupBy("col").sum("sales")`**: Sum of a column per group.
- **`df.groupBy("col").avg("score")`**: Average per group.
- **`df.groupBy("region").agg(F.max("sales"), F.min("sales"))`**: Multiple aggregations.
- **`df.agg(F.mean("amount")).show()`**: **Aggregate without grouping.**
- **`df.groupBy("col1", "col2").agg(F.sum("val"))`**: Group by multiple columns.
- **`df.rollup("col").sum("val").show()`**: Rollup total + subtotals.
- **`df.cube("col").sum("val").show()`**: Cube (all combinations).

@Tajamulkhann

Sorting & Duplicates

- **df.orderBy("col"):** Sort ascending.
- **df.orderBy(df["col"].desc()):** Sort descending.
- **df.sort("col1", "col2"):** Sort by multiple columns.
- **df.sortWithinPartitions("col") :** Sort data within partition.
- **df = df.dropDuplicates():** Remove all duplicate rows.
- **df.dropDuplicates(["col"]):** Remove duplicates on column.
- **df = df.distinct():** Return unique rows.
- **df = df.limit(10):** Return top 10 rows.

@Tajamulkhann

Joins & Merge

- **`df1.join(df2, "key")`:** Inner join on column.
- **`df1.join(df2, "key", "left")`:** Left join.
- **`df1.join(df2, "key", "right")`:** Right join.
- **`df1.join(df2, "key", "outer")`:** Full outer join.
- **`df1.join(df2, df1["id"] == df2["id"], "inner")`:** Join using condition.
- **`df1.crossJoin(df2)`:** Cartesian join.
- **`df1.union(df2)`:** Append rows (same schema).
- **`df1.unionByName(df2)`:** Union using column names.

@Tajamulkhann

Found Helpful ?

Repost

Follow for more!