PySpark Cheat Sheet

getOrCreate

Get or create a PySpark session. If a session has already been created, return that session; otherwise, create a new one.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("myApp").getOrCreate()
df = spark.createDataFrame([(1, "John", "Doe"), (2, "Jane", "Doe")], ["id",
"first_name", "last_name"])
df.show()


+---+----------+---------+
| id|first_name|last_name|
+---+----------+---------+
|  1|      John|      Doe|
|  2|      Jane|      Doe|
+---+----------+---------+
```

read

Reads a CSV file into a PySpark DataFrame.

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)

+---+-----+
| id| name|
+---+-----+
|  1|John |
|  2|Jane |
|  3|Bob  |
+---+-----+
```

show

Displays the contents of a PySpark DataFrame.

```
df.show()

+---+------+
| id| name|
+---+------+
|  1|John  |
|  2|Jane  |
|  3|Bob   |
+---+------+
```

select

Selects columns from a PySpark DataFrame.

```
df.select("name")

+------+
| name|
+------+
|John  |
|Jane  |
|Bob   |
+------+
```

filter

Filters rows of a PySpark DataFrame based on a condition.

```
df.filter(df["id"] == 1)

+---+-----+
| id|name|
+---+-----+
|  1|John|
+---+-----+
```

limit

Limits the number of rows returned by a PySpark DataFrame.

```
df.limit(2)

+---+------+
| id|  name|
+---+------+
|  1|John  |
|  2|Jane  |
+---+------+
```

withColumn

Adds a new column to a PySpark DataFrame.

```
df.withColumn("length", len(df["name"]))

+---+------+-------+
| id|  name|length|
+---+------+-------+
|  1|John  |      4|
|  2|Jane  |      4|
|  3|Bob   |      3|
+---+------+-------+
```

withColumnRenamed

Renames a column in a PySpark DataFrame.

```
df.withColumnRenamed("name", "first_name")

+---+----------+
| id|first_name|
+---+----------+
|  1|      John|
|  2|      Jane|
|  3|       Bob|
+---+----------+
```

Col

Returns a Column based on the given column name or expression.

```
from pyspark.sql.functions import col

df.select(col("age"))

+---+
|age|
+---+
| 25|
| 30|
| 35|
+---+
```

dtypes

Returns df column names and data types

```
df.dtypes

[('id', 'bigint'), ('first_name', 'string'), ('last_name', 'string')]
```

schema

Returns the schema of df

```
df.schema

StructType(List(
  StructField(id,LongType,true),
  StructField(first_name,StringType,true),
  StructField(last_name,StringType,true)))
```

describe

Computes the summary statistics

```
df.describe()

+-------+------------------+----------+---------+
|summary|                id|first_name|last_name|
+-------+------------------+----------+---------+
|  count|                 2|         2|        2|
|   mean|               1.5|      null|     null|
| stddev|0.7071067811865476|      null|     null|
|    min|                 1|      Jane|      Doe|
|    max|                 2|      John|      Doe|
+-------+------------------+----------+---------+
```

distinct

Returns distinct rows (removes duplicates)

```
df.select('first_name', 'last_name').distinct()
+----------+---------+
|first_name|last_name|
+----------+---------+
|      Jane|      Doe|
|      John|      Doe|
+----------+---------+



df.select('last_name').distinct()
+---------+
|last_name|
+---------+
|      Doe|
+---------+
```

printSchema

prints the schema of df

```
df.printSchema()

root
 |-- id: long (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
```

cast

cast column to a different data type

```
from pyspark.sql.functions import col

data = [("John", 25), ("Jane", 30), ("Bob", 35)]
df = spark.createDataFrame(data, ["name", "age"])
df.printSchema()

root
 |-- name: string (nullable = true)
 |-- age: integer (nullable = true)

df = df.select(col("name"), col("age").cast("string"))
df.printSchema()
```

```
root
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
```

dropna

Drops rows containing Null of NaN values (returns a new DataFrame)

```
df.dropna()

Before:
+----+-----+-----+
| id | age | city|
+----+-----+-----+
|  1 |   30|  NYC|
|  2 | None|  LA |
|  3 |   25|None |
|  4 |   45|  SF |
+----+-----+-----+

after:
+----+---+---+
| id |age|city|
+----+---+---+
|  1 | 30|NYC|
|  4 | 45| SF|
+----+---+---+
```

isNotNull

Returns true if if the value in column is not null, otherwise returns falls

```
df.withColumn("isNotNull", df.first_name.isNotNull())

+---+----------+---------+---------+
| id|first_name|last_name|isNotNull|
+---+----------+---------+---------+
|  1|      John|      Doe|     true|
|  2|      null|      Doe|    false|
+---+----------+---------+---------+
```

IsNull

Returns true if the value is null, else false

```
df.withColumn("isNull", df.first_name.isNull())

+---+----------+---------+------+
| id|first_name|last_name|isNull|
+---+----------+---------+------+
|  1|      John|      Doe| false|
|  2|      null|      Doe|  true|
+---+----------+---------+------+
```

groupBy

Groups rows of a PySpark DataFrame by one or more columns.

```
df.groupBy("name").count()

+-----+-----+
| name|count|
+-----+-----+
|John |    1|
|Jane |    1|
|Bob  |    1|
+-----+-----+
```

Count

Returns the count of the number of rows in the DataFrame.

```
df.count()
```

```
3
```

sort

Returns a new DataFrame sorted by the specified column(s).

```
df.sort("salary")

+-------+------+----------+
|   name|salary|department|
+-------+------+----------+
|Michael|  2500|        HR|
|   Andy|  4500|        IT|
|  Alice|  5000|        IT|
|  James|  5500|        HR|
|  Emily|  9000|        IT|
+-------+------+----------+
```

orderBy

Alias for sort().

agg

Compute aggregates and returns the result as a DataFrame.

```
df.agg({"salary": "max"})

+-----------+
|max(salary)|
+-----------+
|       9000|
+-----------+
```

```
from pyspark.sql.functions import countDistinct, avg

df.agg(countDistinct("department"), avg("salary"))
sql
+--------------------------+-----------------+
|count(DISTINCT department)|      avg(salary)|
+--------------------------+-----------------+
|                        2 |6666.666666666667|
+--------------------------+-----------------+
```

sum

Computes the sum of the given column(s).

```
from pyspark.sql.functions import sum

df.select(sum("salary"))

+-----------+
|sum(salary)|
+-----------+
|      33333|
+-----------+
```

avg

Computes the average of the given column(s).

```
from pyspark.sql.functions import avg

df.select(avg("salary"))

+-----------+
|avg(salary)|
+-----------+
|       6666|
+-----------+
```

max

Computes the maximum value of the given column(s).

```
from pyspark.sql.functions import max

+-----------+
|max(salary)|
+-----------+
|      10000|
+-----------+
```

min

Computes the maximum value of the given column(s)

```
from pyspark.sql.functions import max

+-----------+
|min(salary)|
+-----------+
|        600|
+-----------+
```

pow

Returns the value of the first argument raised to the power of the second argument.

```
from pyspark.sql.functions import pow

df.select(pow(col("age"), 2))

+--------+
|pow(age)|
+--------+
|     625|
|     900|
|    1225|
+--------+
```

sqrt

Returns the square root of the specified column.

```
from pyspark.sql.functions import sqrt

df.select(sqrt(col("age")))

+--------------------+
|       SQRT(age)    |
+--------------------+
|5.0                 |
|5.477225575051661   |
|5.916079783099616   |
+--------------------+
```

sin, cos

Returns the sine or cosine of the specified column.

```
from pyspark.sql.functions import sin, cos

df.select(sin(col("age")), cos(col("age")))

+---------------------+---------------------+
|          SIN(age)   |          COS(age)   |
+---------------------+---------------------+
|-0.13235175009777303| 0.9912028118634735   |
|-0.9880316240928618 | 0.15425144988758405|
| 0.42818266949615195|-0.9036922050919365  |
+---------------------+---------------------+
```

udf

User-Defined Functions (UDFs) allow you to extend the functionality of Spark by providing custom transformations.

```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

# Define the UDF
square_udf = udf(lambda x: x*x, IntegerType())

# Apply the UDF to a column
df.select("name", square_udf("age"))

+-------+---------+
|   name|UDF(age) |
+-------+---------+
|Michael|     625 |
|   Andy|     900 |
| Justin|    1225 |
+-------+---------+
```

to_timestamp

Converts the given column to a timestamp with a specified format.

```
from pyspark.sql.functions import to_timestamp

# Create a new dataframe with a date column
dates = [("2022-02-01 00:00:01",), ("2022-02-02 00:01:02",), ("2022-02-03 01:02:03",)]
df_dates = spark.createDataFrame(dates, ["date"])

# Convert the date to a timestamp
df_dates.select(to_timestamp("date", "yyyy-MM-dd HH:mm:ss"))

+-----------------------+
|to_timestamp(date, ...)|
+-----------------------+
|2022-02-01 00:00:01    |
|2022-02-02 00:01:02    |
|2022-02-03 01:02:03    |
+-----------------------+
```