PySpark Basics Cheat Sheet
for Data Engineering

DIGGIBYTE
CREATING VALUE WITH DATA

PySpark

PySpark is a Python API for Apache Spark.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read different format files.

```
> from pyspark.sql import SparkSession
> spark = SparkSession \
          .builder \
          .appName('PySpark Practice') \
          .master('local[2]') \
          .getOrCreate()
```

Creating DataFrames

Creating DataFrame

```
> empData=[(101,'Sam', 32),
           (201,'John', 40),
           (301,'David', 28)
           ]
> empSchema= ['emp_id', 'emp_name', 'age']
> empDF= spark.createDataFrame(data=empData, schema=
empSchema)
```

Creating DataFrame from Custom Schema

```
> from pyspark.sql.types import StructType,
StructField, IntegerType, StringType
> empSchema=
StructType([StructField('emp_id',IntegerType(), True),
            StructField('emp_name', StringType(),
True),
            StructField('age', IntegerType(), True)
           ])
> empDF.show(truncate= False)
+------+--------+---+
|emp_id|emp_name|age|
+------+--------+---+
|   101|     Sam| 32|
|   201|    John| 40|
|   301|   David| 28|
+------+--------+---+
```

Creating DataFrame by Reading Files

InferSchema option

```
> empDF= spark.read.format('csv').options(header=
'True', InferSchema= 'True', sep=
',').load('./emp.csv')
```

Different File Formats: csv, text, json, parquet, avro, orc can be read with different options

Custom Schema option

```
> empDF= spark.read.format('csv').options(header=
'True', sep= ',').schema(empSchema).load('./emp.csv')
```

Mode() option

Different mode options while reading file are
'PERMISSIVE' – read all records from file,
'DROPMALFORMED' - delete bad records & don't read them
'FAILFAST' – raise error(SparkException) if there are bad records in file.
by default mode =' PERMISSIVE'

```
> empDF= spark.read.format("csv").options(mode=
"PERMISSIVE", header= "true", sep=
',').schema(empSchema).load("./emp.csv")
```

Duplicate Values

```
> empDF = empDF.dropDuplicates()
```

Queries

```
> from pyspark.sql import functions as F
```

Select

```
> empDF.select("emp_name").show()
```

Show all entries in `emp_name` column

```
> empDF.select("emp_name","age") \
      .show()
```

Show all entries in `emp_name`, `age` and `type`

```
> empDF.select("emp_name",
               "age",explode("phoneNumber") \
               .alias("contactInfo")) \
               .select("contactInfo.type",
               "emp_name",
               "age").show()
>empDF.select(empDF["emp_name"],df["age"]+)
      .show()
```

Show all entries in `emp_name` and `age`,add `1` to the entries of `age`

```
>empDF.select(empDF ['age'] >24).show()
```

Show all entries where `age >28`

When

```
> empDF.select("emp_name",
               F.when(df.age > 32, 1) \
              .otherwise(0)) \.show()
```

Show `emp_name` and 0 or 1 dependingon `age >32`

```
>empDF[empDF.emp_name.isin("Sam","John")]
                     .collect()
```

Show `emp_name` if in the given options

Like

```
>df.select("emp_id",df.emp_name.like("Sam")
\ .show()
```

Show `emp_nane`, and `emp_id` is `TRUE` if `emp_name` is like `Sam`

Startswith - Endswith

```
> df.select("emp_name",df. emp_id \
                     .startswith("Sm")).show()
```

Show `emp_id`, and `TRUE` if `emp_name` starts with `Sm`

```
> df.select(df.emp_name.endswith("hn")) \
      .show()
```

Show last names ending in `hn`

Substring

```
> df.select(df.emp_name.substr(1, 3) \
                               .alias("name")) \
      .collect()
```

Return substrings of `emp_name`

Between

```
> df.select(df.age.between(32, 40)) \
```

Show `age`: values are `TRUE` if between `32` and `40`

Add, Update & Remove Columns

To add new column to DataFrame

```
> empDF = empDF.withColumn('city',lit('Mumbai'))
```

To rename column name of DataFrame

```
> empDF = empDF.withColumnRenamed('age', 'emp_age')
```

To drop column of DataFrame

```
> empDF= empDF.drop('city')
```

JOINS

```
> joinDF= empDF.join(deptDF, 'dept_id', 'FULLOUTER')
```

Different Types of Joins are INNER, FULLOUTER, RIGHTOUTER, LEFTOUTER.

REGEX_REPLACE(): Replace one value with other value in column

```
from pyspark.sql.functions import regexp_replace
> deptDF.withColumn('dept_name', regexp_replace('dept_name', 'HR',
'Human Resource'))
```

TRIM() : trim space from left/right/ both_

```
from pyspark.sql.functions import ltrim, rtrim, trim

> deptDF.withColumn('dept_name', ltrim('dept_name'))
```

groupBy

`> empDF.groupBy("age")\ .count().show()`	Group by `age`, count the membersin the groups

Filter

`> empDF.filter(df["age"]>28).show()`	Filter entries of `age`, only keep those records of which the values are `>28`

Sort

```
> empDF.sort(empDF.age.desc()).collect()
> empDF.sort("age", ascending=False).collect()
> empDF.orderBy(["age","city"],ascending=[0,1])\
  .collect()
```

Fill & Fillna

```
> df.na.fill(value=0).show()
> df.na.fill(value=0,subset=["city"]).show()

> df.fillna("unknown",["city"]) \
  .fillna("",["age"]).show()
```

Inspect Data

```
> df.show()         Display first 20 rows n truncate
                    column value to 20 characters of DF
> df.head()         Return first 'n' row
> df.first()        Return first row
> df.take(2)        Return the first 'n' rows
> df.schema         Return the schema of DF
> df.dtype          Return DF column names and data types
> df.columns        Returns column names as list
```

Date timestamp

date_format() - convert date from one format to another

```
> df.select(current_date().alias("current_date"), \
  date_format(current_timestamp(),"yyyy MM dd
hh:mm:ss")).alias("yyyy MM dd").show()
```

unix_timestamp() - convert Timestamp into Unix Epoch Time

```
> df.select(date_format(current_timestamp(),"yyyy MM dd
  hh:mm:ss")).withColumn('unix_epoch_time',
  unix_timestamp('current_date')).show()
eg - Timestamp         : 2022 08 21 09:11:48
into Unix Epoch Time : 1661040000
```

from_unixtime()- convert Unix Epoch Time into Timestamp

Write Files from DataFrame & Save

```
>empDF.write.format('parquet').mode('overwrite').save(path
= './Output/Employee.parquet')
>empDF.write.mode('overwrite').csv(path='./Output/Employee
.txt', header='True', sep= '\t')
```

Mode options are append, overwrite, error, ignore.

Mentor: Anjali Punyasri

Prepared by Monali Sahare & Suraj Kumar