

Python For Data Science Cheat Sheet

PySpark - SQL Basics



PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.



Initializing SparkSession

A SparkSession can be used to create DataFrames, register DataFrames as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
Infer Schema
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
Specify Schema
>>> people = parts.map(lambda p: Row(name=p[0],
    age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
    field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
+-----+
| name | age |
+-----+
| Mine | 28 |
| Filip | 29 |
| Jonathan | 30 |
+-----+
```

From Spark Data Sources

```
JSON
>>> df = spark.read.json("customer.json")
>>> df.show()
+-----+-----+-----+-----+-----+
| address | age | firstName | lastName | phoneNumber |
+-----+-----+-----+-----+-----+
| [New York,10021,N...] | 25 | John | Smith | [[212 555-1234,ho... |
| [New York,10021,N...] | 21 | Jane | Doe | [[322 888-1234,ho... |
+-----+-----+-----+-----+-----+
>>> df2 = spark.read.load("people.json", format="json")
Parquet files
>>> df3 = spark.read.load("users.parquet")
TXT files
>>> df4 = spark.read.text("people.txt")
```

Inspect Data

```
>>> df.dtypes
>>> df.show()
>>> df.head()
>>> df.first()
>>> df.take(2)
>>> df.schema
```

Return df column names and data types
Display the content of df
Return first n rows
Return first row
Return the first n rows
Return the schema of df

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
Select
>>> df.select("firstName").show()
>>> df.select("firstName", "lastName") \
    .show()
>>> df.select("firstName",
    "age",
    explode("phoneNumber") \
    .alias("contactInfo")) \
    .select("contactInfo.type",
    "firstName",
    "age") \
    .show()
>>> df.select(df["firstName"],df["age"]+ 1) \
    .show()
>>> df.select(df["age"] > 24).show()
When
>>> df.select("firstName",
    F.when(df.age > 30, 1) \
    .otherwise(0)) \
    .show()
>>> df[df.firstName.isin("Jane","Boris")] \
    .collect()
Like
>>> df.select("firstName",
    df.lastName.like("Smith")) \
    .show()
Startswith - Endswith
>>> df.select("firstName",
    df.lastName \
    .startswith("Sm")) \
    .show()
>>> df.select(df.lastName.endswith("th")) \
    .show()
Substring
>>> df.select(df.firstName.substr(1, 3) \
    .alias("name")) \
    .collect()
Between
>>> df.select(df.age.between(22, 24)) \
    .show()
```

Show all entries in firstName column
Show all entries in firstName, age and type
Show all entries in firstName and age, add 1 to the entries of age
Show all entries where age >24
Show firstName and 0 or 1 depending on age >30
Show firstName if in the given options
Show firstName, and lastName is TRUE if lastName is like Smith
Show firstName, and TRUE if lastName starts with Sm
Show last names ending in th
Return substrings of firstName
Show age: values are TRUE if between 22 and 24

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
    .withColumn('postalCode',df.address.postalCode) \
    .withColumn('state',df.address.state) \
    .withColumn('streetAddress',df.address.streetAddress) \
    .withColumn('telePhoneNumber',
    explode(df.phoneNumber.number)) \
    .withColumn('telePhoneType',
    explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age") \
    .count() \
    .show()
```

Group by age, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of age, only keep those records of which the values are >24

Sort

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

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
    .replace(10, 20) \
    .show()
```

Replace null values
Return new df omitting rows with null values
Return new df replacing one value with another

Repartitioning

```
>>> df.repartition(10) \
    .rdd \
    .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

df with 10 partitions
df with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people") \
    .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert df into an RDD
Convert df into a RDD of string
Return the contents of df as Pandas DataFrame

Write & Save to Files

```
>>> df.select("firstName", "city") \
    .write \
    .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
    .write \
    .save("namesAndAges.json", format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

