■ Mastering PySpark: Essential Cheatsheet

Mastering PySpark: Essential Cheatsheet

Basics: Loading and Exploring Data

```
# Create a Spark session
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

# Read CSV data into a DataFrame
df = spark.read.csv('/path/to/your/input/file')

# Show a preview of the data
df.show()

# Display the first and last n rows
df.head(5)
df.tail(5)

# Display as JSON (in-memory, caution with large
datasets)
df = df.limit(10)
print(json.dumps([row.asDict(recursive=True) for row in
df.collect()], indent=2))
```

Common Operations: Columns and Rows

```
# Get column names, types, and schema
df.columns
df.dtypes
df.schema

# Get row and column count
df.count()
len(df.columns)

# Write output to disk
df.write.csv('/path/to/your/output/file')

# Convert to Pandas DataFrame
df = df.toPandas()
```

Filtering and Sorting

```
df = df.filter(df.age > 25)
```

```
df = df.filter((df.age > 25) & (df.is_adult == 'Y'))

# Sort results
df = df.orderBy(df.age.asc())
df = df.orderBy(df.age.desc())
```

Joins: Combining Datasets

```
# Left join with another dataset
df = df.join(person_lookup_table, 'person_id', 'left')

# Join on different columns
df = df.join(other_table, df.id == other_table.person_id,
'left')

# Join on multiple columns
df = df.join(other_table, ['first_name', 'last_name'], 'left')
```

Column Operations: Transforming Data

```
# Add new columns
df = df.withColumn('status', F.lit('PASS'))

# Construct dynamic columns
df = df.withColumn('full_name',
F.when((df.fname.isNotNull() & df.lname.isNotNull()),
F.concat(df.fname, df.lname)).otherwise(F.lit('N/A')))

# Select, drop, and rename columns
df = df.select('name', 'age',
F.col('dob').alias('date_of_birth'))
df = df.drop('mod_dt', 'mod_username')
df = df.withColumnRenamed('dob', 'date_of_birth')
```

Casting, Null Handling, and Duplicates

```
# Cast a column to a different type
df = df.withColumn('price', df.price.cast(T.DoubleType()))

# Replace nulls with specific values
df = df.fillna({'first_name': 'Tom', 'age': 0})

# Coalesce null values
df = df.withColumn('last_name', F.coalesce(df.last_name,
df.surname, F.lit('N/A')))

# Drop duplicate rows
df = df.dropDuplicates()
df = df.distinct()

# Drop duplicates considering specific columns
```

df = df.dropDuplicates(['name', 'height'])

String Operations: Filters and Functions

```
# String filters
df = df.filter(df.name.contains('o'))
df = df.filter(df.name.startswith('Al'))
df = df.filter(df.name.endswith('ice'))
df = df.filter(df.is_adult.isNull())
df = df.filter(df.first_name.isNotNull())
df = df.filter(df.name.like('Al%'))
df = df.filter(df.name.rlike('[A-Z]*ice$'))
df = df.filter(df.name.isin('Bob', 'Mike'))

# String functions
df = df.withColumn('short_id', df.id.substr(0, 10))
df = df.withColumn('name', F.trim(df.name))
df = df.withColumn('id', F.lpad('id', 4, '0'))
df = df.withColumn('full_name', F.concat('fname', F.lit(' '),
'lname'))
df = df.withColumn('full_name', F.concat_ws('-', 'fname',
'lname'))
df = df.withColumn('id', F.regexp_replace(id, '0F1(.*)', '1F1-
$1'))
df = df.withColumn('id', F.regexp_extract(id, '[0-9]*', 0))
```

Number Operations: Mathematical Functions

```
# Mathematical operations
df = df.withColumn('price', F.round('price', 0))
df = df.withColumn('price', F.floor('price'))
df = df.withColumn('price', F.ceil('price'))
df = df.withColumn('price', F.abs('price'))
```

df = df.withColumn('exponential_growth', F.pow('x', 'y'))

```
df = df.withColumn('least', F.least('subtotal', 'total'))
df = df.withColumn('greatest', F.greatest('subtotal', 'total'))
```

Date and Timestamp Operations

```
# Date and timestamp operations
df = df.withColumn('current_date', F.current_date())
df = df.withColumn('date_of_birth',
F.to_date('date_of_birth', 'yyyy-MM-dd'))
df = df.withColumn('time_of_birth',
F.to_timestamp('time_of_birth', 'yyyy-MM-dd HH:mm:ss'))
df = df.filter(F.year('date_of_birth') == F.lit('2017'))
df = df.withColumn('three_days_after',
F.date_add('date_of_birth', 3))
df = df.withColumn('three_days_before',
F.date_sub('date_of_birth', 3))
```

```
df = df.withColumn('next_month',
F.add_months('date_of_birth', 1))
df = df.withColumn('days_between', F.datediff('start',
'end'))
df = df.withColumn('months_between',
F.months_between('start', 'end'))
df = df.filter((F.col('date_of_birth') >= F.lit('2017-05-10')) &
(F.col('date_of_birth') <= F.lit('2018-07-21')))
```

Array Operations: Functions and

Transformations

```
# Array operations
df = df.withColumn('full_name', F.array('fname', 'lname'))
df = df.withColumn('empty_array_column', F.array([]))
df = df.withColumn('first_element',
F.col("my_array").getItem(0))
df = df.withColumn('array_length', F.size('my_array'))
df = df.withColumn('flattened', F.flatten('my_array'))
df = df.withColumn('unique_elements',
F.array_distinct('my_array'))
df = df.withColumn('elem_ids',
F.transform(F.col('my_array'), lambda x: x.getField('id')))
df = df.select(F.explode('my_array'))
```

Struct Operations: Making and Accessing

Struct Columns

```
# Struct operations
df = df.withColumn('my_struct', F.struct(F.col('col_a'),
F.col('col_b')))
df = df.withColumn('col_a',
F.col('my_struct').getField('col_a'))
```

Aggregation Operations: Basic and Advanced

```
# Aggregation operations
df =
df.groupBy('gender').agg(F.max('age').alias('max_age_by_g
ender'))
```

df =

```
df.groupBy('age').agg(F.collect_set('name').alias('person_n
ames'))
```

```
# Window functions for selecting the latest row in each
group
from pyspark.sql import Window as W
window = W.partitionBy("first_name",
"last_name").orderBy(F.desc("date"))
```

```
df = df.withColumn("row_number",
F.row_number().over(window))
df = df.filter(F.col("row_number") ==
1).drop("row_number")
```

Repartitioning and UDFs (User Defined

```
Functions)

 # Repartitioning

df = df.repartition(1)

 # UDFs (User Defined Functions)

times_two_udf = F.udf(lambda x: x * 2)
df = df.withColumn('age', times_two_udf(df.age))
```

random_name_udf = F.udf(lambda: random.choice(['Bob',

```
'Tom', 'Amy', 'Jenna']))
df = df.withColumn('name', random_name_udf())
```

Useful Functions and Transformations

```
 # Flatten nested struct columns

flat_df = flatten(df, '_')

 # Lookup and replace values from another DataFrame

df = lookup_and_replace(people, pay_codes, 'id',
'pay_code_id', 'pay_code_desc')
```