# Introduction to PySpark DataFrames

BIG DATA FUNDAMENTALS WITH PYSPARK

## What are PySpark DataFrames?

- PySpark SQL is a Spark library for structured data. It provides more information about the structure of data and computation
- PySpark DataFrame is an immutable distributed collection of data with named columns
- Designed for processing both structured (e.g relational database) and semi-structured data (e.g JSON)
- Dataframe API is available in Python, R, Scala, and Java
- DataFrames in PySpark support both SQL queries ( `SELECT * from table` ) or expression methods ( `df.select()` )

## SparkSession - Entry point for DataFrame API

- SparkContext is the main entry point for creating RDDs
- SparkSession provides a single point of entry to interact with Spark DataFrames
- SparkSession is used to create DataFrame, register DataFrames, execute SQL queries
- SparkSession is available in PySpark shell as `spark`

## Creating DataFrames in PySpark

- Two different methods of creating DataFrames in PySpark
    - From existing RDDs using SparkSession's createDataFrame() method
    - From various data sources (CSV, JSON, TXT) using SparkSession's read method
- Schema controls the data and helps DataFrames to optimize queries
- Schema provides information about column name, type of data in the column, empty values etc.,

## Create a DataFrame from RDD

```
iphones_RDD = sc.parallelize([
    ("XS", 2018, 5.65, 2.79, 6.24),
    ("XR", 2018, 5.94, 2.98, 6.84),
    ("X10", 2017, 5.65, 2.79, 6.13),
    ("8Plus", 2017, 6.23, 3.07, 7.12)
])
```

```
names = ['Model', 'Year', 'Height', 'Width', 'Weight']
```

```
iphones_df = spark.createDataFrame(iphones_RDD, schema=names)
type(iphones_df)
```

```
pyspark.sql.dataframe.DataFrame
```

# Create a DataFrame from reading a CSV/JSON/TXT

```
df_csv = spark.read.csv("people.csv", header=True, inferSchema=True)
```

```
df_json = spark.read.json("people.json", header=True, inferSchema=True)
```

```
df_txt = spark.read.txt("people.txt", header=True, inferSchema=True)
```

- Path to the hle and two optional parameters
- Two optional parameters
  - `header=True` , `inferSchema=True`

## Let's practice

# Interacting with PySpark DataFrames

BIG DATA FUNDAMENTALS WITH PYSPARK

## DataFrame operators in PySpark

- DataFrame operations: Transformations and Actions
- DataFrame Transformations:
    - select(), hlter(), groupby(), orderby(), dropDuplicates() and withColumnRenamed()
- DataFrame Actions :
    - printSchema(), head(), show(), count(), columns and describe()

## select() and show() operations

- `select()` transformation subsets the columns in the DataFrame

```
df_id_age = test.select('Age')
```

  - `show()` action prints hrst 20 rows in the DataFrame

```
df_id_age.show(3)
```

```
+----+
|Age|
+----+
|  17|
|  17|
|  17|
+----+
only showing top 3 rows
```

## filter() and show() operations

- `filter()` transformation hlters out the rows based on a condition

```
new_df_age21 = new_df.filter(new_df.Age > 21)
new_df_age21.show(3)
```

```
+-------+------+---+
|User_ID|Gender|Age|
+-------+------+---+
|1000002|     M| 55|
|1000003|     M| 26|
|1000004|     M| 46|
+-------+------+---+
only showing top 3 rows
```

## groupby() and count() operations

- `groupby()` operation can be used to group a variable

```
test_df_age_group = test_df.groupby('Age')
test_df_age_group.count().show(3)
```

```
+---+------+
|Age| count|
+---+------+
| 26|219587|
| 17|     4|
| 55| 21504|
+---+------+
only showing top 3 rows
```

## orderby() Transformations

- `orderby()` operation sorts the DataFrame based one or more columns

```
test_df_age_group.count().orderBy('Age').show(3)
```

```
+---+------+
|Age|count|
+---+------+
|  0|15098|
| 17|    4|
| 18|99660|
+---+------+
only showing top 3 rows
```

## dropDuplicates()

- `dropDuplicates()` removes the duplicate rows of a DataFrame

```
test_df_no_dup = test_df.select('User_ID','Gender', 'Age').dropDuplicates()
test_df_no_dup.count()
```

```
5892
```

## withColumnRenamed Transformations

- `withColumnRenamed()` renames a column in the DataFrame

```
test_df_sex = test_df.withColumnRenamed('Gender', 'Sex')
test_df_sex.show(3)
```

```
+-------+---+---+
|User_ID|Sex|Age|
+-------+---+---+
|1000001|  F| 17|
|1000001|  F| 17|
|1000001|  F| 17|
+-------+---+---+
```

## printSchema()

- `printSchema()` operation prints the types of columns in the DataFrame

```
test_df.printSchema()
```

```
 |-- User_ID: integer (nullable = true)
 |-- Product_ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Occupation: integer (nullable = true)
 |-- Purchase: integer (nullable = true)
```

## columns actions

- `columns` operator prints the columns of a DataFrame

```
test_df.columns
```

```
['User_ID', 'Gender', 'Age']
```

## describe() actions

- `describe()` operation compute summary statistics of numerical columns in the DataFrame

```
test_df.describe().show()
```

```
+-------+------------------+------+------------------+
|summary|           User_ID|Gender|               Age|
+-------+------------------+------+------------------+
|  count|            550068|550068|            550068|
|   mean|1003028.8424013031|  null|30.382052764385495|
| stddev|1727.5915855307312|  null|11.866105189533554|
|    min|           1000001|     F|                 0|
|    max|           1006040|     M|                55|
+-------+------------------+------+------------------+
```

## Let's practice

BIG DATA FUNDAMENTALS WITH PYSPARK

# Interacting with DataFrames using PySpark SQL

BIG DATA FUNDAMENTALS WITH PYSPARK

## DataFrame API vs SQL queries

- In PySpark You can interact with SparkSQL through DataFrame API and SQL queries
- The DataFrame API provides a programmatic domain-specihc language (DSL) for data
- DataFrame transformations and actions are easier to construct programmatically
- SQL queries can be concise and easier to understand and portable
- The operations on DataFrames can also be done using SQL queries

## Executing SQL Queries

- The SparkSession `sql()` method executes SQL query
- `sql()` method takes a SQL statement as an argument and returns the result as DataFrame

```
df.createOrReplaceTempView("table1")
```

```
df2 = spark.sql("SELECT field1, field2 FROM table1")
df2.collect()
```

```
[Row(f1=1, f2='row1'), Row(f1=2, f2='row2'), Row(f1=3, f2='row3')]
```

## SQL query to extractdata

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Product_ID FROM test_table'''
```

```
test_product_df = spark.sql(query)
test_product_df.show(5)
```

```
+----------+
|Product_ID|
+----------+
| P00069042|
| P00248942|
| P00087842|
| P00085442|
| P00285442|
+----------+
```

## Summarizing and grouping data using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, max(Purchase) FROM test_table GROUP BY Age'''
```

```
spark.sql(query).show(5)
```

```
+-----+-------------+
|  Age|max(Purchase)|
+-----+-------------+
|18-25|        23958|
|26-35|        23961|
| 0-17|        23955|
|46-50|        23960|
|51-55|        23960|
+-----+-------------+
only showing top 5 rows
```

## Filtering columns using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, Purchase, Gender FROM table1 WHERE Purchase > 20000 AND Gender ==  "F"'''
```

```
spark.sql(query).show(5)
```

```
+-----+--------+------+
|  Age|Purchase|Gender|
+-----+--------+------+
|36-45|   23792|     F|
|26-35|   21002|     F|
|26-35|   23595|     F|
|26-35|   23341|     F|
|46-50|   20771|     F|
+-----+--------+------+
only showing top 5 rows
```

## Time to practice!

BIG DATA FUNDAMENTALS WITH PYSPARK