



BIG DATA IN MACHINE LEARNING

Bài 4: PySpark SQL & DataFrame

Phòng LT & Mạng

<https://csc.edu.vn/lap-trinh-van-cSDL/Big-Data-In-Machine-Learning.html>

2020



Nội dung

1. Giới thiệu Spark SQL

2. Giới thiệu Spark DataFrame

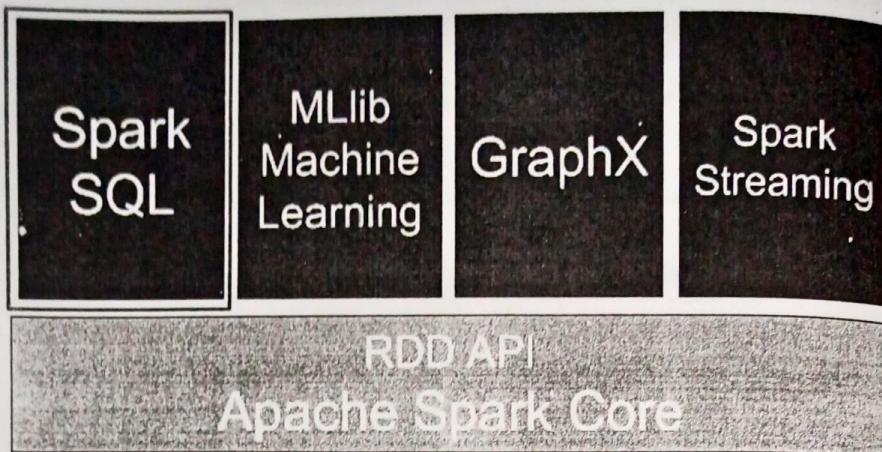
3. Làm việc với PySpark DataFrame

4. Làm việc với PySpark SQL





Giới thiệu Spark SQL



Big Data in Machine Learning

3

Giới thiệu Spark SQL



□ Spark SQL

- Là một module trong Spark, nó tích hợp xử lý với Spark's functional programming API.
- Hỗ trợ truy vấn dữ liệu thông qua SQL và Hive Query Language.
- Nếu đã quen thuộc với RDBMS, Spark SQL là một sự chuyển đổi dễ dàng.
- Spark SQL có nguồn gốc là Apache Hive thực thi trong Spark và được tích hợp với Spark stack. Vì Apache Hive có một số hạn chế nên Spark SQL được xây dựng để khắc phục những nhược điểm này và thay thế Apache Hive.



Big Data in Machine Learning

4

Giới thiệu Spark SQL

- Tích hợp relational processing lập trình chức năng của Spark, cung cấp hỗ trợ cho các nguồn dữ liệu khác nhau và cho phép thực hiện các SQL query với code transformation do đó tạo ra một công cụ rất mạnh.
- Làm mờ ranh giới giữa RDD và bảng quan hệ, cung cấp sự tích hợp chặt chẽ hơn nhiều giữa xử lý quan hệ và xử lý thủ tục (procedural processing), thông qua API DataFrame khai báo được tích hợp với Spark code.
- Cung cấp tối ưu hóa cao hơn: API DataFrame và API dataset là những cách để tương tác với Spark SQL.

Big Data in Machine Learning

5



Giới thiệu Spark SQL

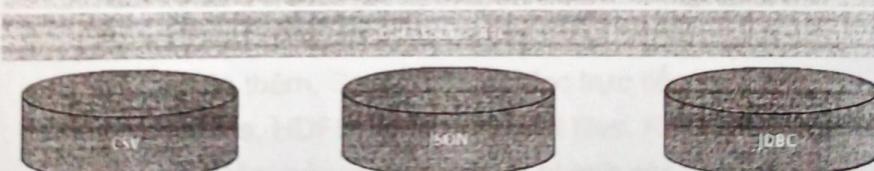
- Spark SQL cung cấp DataFrame APIs thực hiện các hoạt động relational operation cả trên các nguồn dữ liệu bên ngoài và built-in distributed collection của Spark.

Architecture Of Spark SQL

edureka!

DataFrame DSL

Spark SQL and HQL



<https://www.edureka.co/bigdata-spark-sql-tutorial/>

Big Data in Machine Learning

6

Giới thiệu Spark SQL

□ Đặc điểm của Spark SQL

• Tích hợp với Spark

- Spark SQL queries được tích hợp với Spark programs. Spark SQL cho chúng ta truy vấn dữ liệu có cấu trúc trong các Spark programs bằng cách sử dụng SQL hoặc DataFrame API (có thể được dùng trong Java, Scala, Python & R. Để thực thi các tính toán trực tiếp (streaming computation), developer đơn giản chỉ cần viết một phép tính hàng loạt (batch computation) dựa trên DataFrame/ Dataset API, và Spark tự động thực hiện tính toán trong một streaming fashion. Thiết kế mạnh mẽ này có nghĩa là các developer không cần quản lý thủ công các state, failures, hoặc giữ cho ứng dụng đồng bộ (sync) các công việc theo lô (batch job). Thay vào đó, streaming job luôn cho kết quả tương tự như batch job trên cùng một dữ liệu.

Big Data in Machine Learning

7

Giới thiệu Spark SQL

• Truy cập dữ liệu thống nhất

- DataFrames và SQL hỗ trợ cùng một cách phổ biến để truy cập nhiều nguồn dữ liệu khác nhau như Hive, Avro, Parquet, ORC, JSON, và JDBC. Nó kết nối dữ liệu thông qua các nguồn này. Vì vậy rất hữu ích khi chứa tất cả các nguồn hiện có vào Spark SQL.



• Tương thích với Hive

- Spark SQL thực thi các truy vấn Hive chưa sửa đổi trên dữ liệu hiện tại. Nó viết lại Hive front-end và meta store, cho phép tương thích hoàn toàn với dữ liệu, truy vấn và UDF của Hive hiện tại.

• Kết nối chuẩn (Standard Connectivity)

- Kết nối thông qua JDBC hoặc ODBC. JDBC và ODBC là các tiêu chuẩn về khả năng kết nối cho các business intelligence tool.



• Hiệu suất và khả năng mở rộng

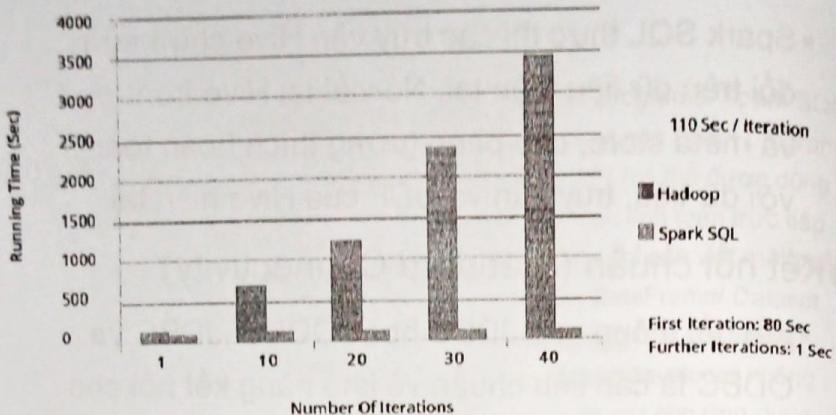
- Spark SQL kết hợp trình tối ưu hóa dựa trên chi phí (cost-based optimizer), code generation và columnar storage để làm cho các query trở nên linh hoạt kèm theo việc tính toán sử dụng hàng ngàn node sử dụng Spark engine (cung cấp khả năng chịu lỗi). Các interface được cung cấp bởi Spark SQL cho Spark nhiều thông tin hơn về cấu trúc của cả dữ liệu và tính toán. Trong nội bộ, Spark SQL sử dụng thông tin bổ sung để thực hiện việc tối ưu hóa thêm. Spark SQL có đọc trực tiếp từ nhiều nguồn (files, HDFS, JSON/Parquet files, RDD hiện có, Hive...). Đảm bảo việc thực hiện nhanh các truy vấn Hive hiện có.



Giới thiệu Spark SQL



Performance: Spark SQL Vs Hadoop



Runtime of Spark SQL vs Hadoop. Spark SQL is faster

Source: [Cloudera Apache Spark Blog](#)

Big Data in Machine Learning

11

Giới thiệu Spark SQL



• User Defined Function

- Spark SQL có các User-Defined Functions (UDFs).
- UDF là một tính năng của Spark.

Nội dung



1. Giới thiệu Spark SQL

2. Giới thiệu Spark DataFrame

3. Làm việc với PySpark DataFrame

4. Làm việc với PySpark SQL



Giới thiệu DataFrame



□ Trong Apache Spark, DataFrame là một cấu trúc dữ liệu theo định dạng row và column

- Mỗi column đại diện cho một feature/attribute
- Mỗi row đại diện cho một point/object dữ liệu riêng

□ Spark DataFrame như một bảng trong cơ sở dữ liệu quan hệ hoặc một bảng tính Excel có các tiêu đề Cột.

□ Spark bắt đầu với “RDD” syntax hơi khó học. Từ Spark 2.0 trở lên đã chuyển sang DataFrame syntax gọn gàng và dễ làm việc hơn.



Giới thiệu DataFrame

☐ PySpark DataFrame cũng chia sẻ một số đặc điểm chung với RDD:

- Immutable in nature (bất biến): có thể tạo DataFrame / RDD một lần nhưng không thể thay đổi nó. Tuy nhiên, chúng ta có thể chuyển đổi DataFrame / RDD sau khi áp dụng các phép biến đổi (transformation).
- Lazy Evaluations: một task không được thực hiện cho đến khi có một action được thực hiện.
- Distributed: RDD và DataFrame đều được phân tán dữ liệu.

Big Data in Machine Learning

15

Giới thiệu DataFrame

☐ Ưu điểm của DataFrame

- DataFrame được thiết kế để xử lý collection dữ liệu lớn có cấu trúc hoặc bán cấu trúc.
- Cấu trúc trong Spark DataFrame được tổ chức dưới dạng column được đặt tên giúp Apache Spark có thể hiểu schema của DataFrame. Do đó, giúp Spark có thể tối ưu hóa kế hoạch thực thi các truy vấn.
- DataFrame trong Apache Spark có khả năng xử lý petabyte dữ liệu.

Big Data in Machine Learning

16



Giới thiệu DataFrame

- DataFrame hỗ trợ cho nhiều nguồn và định dạng dữ liệu.
- Dữ liệu thống kê thường rất lộn xộn và chứa nhiều giá trị thiếu và sai và ngoài phạm vi. Vì vậy, một tính năng cực kỳ quan trọng của DataFrame là quản lý dữ liệu một cách rõ ràng.
- DataFrame hỗ trợ API cho các ngôn ngữ khác nhau như Python, R, Scala, Java.
- API DataFrame là một cách để tương tác với Spark SQL.



Nội dung



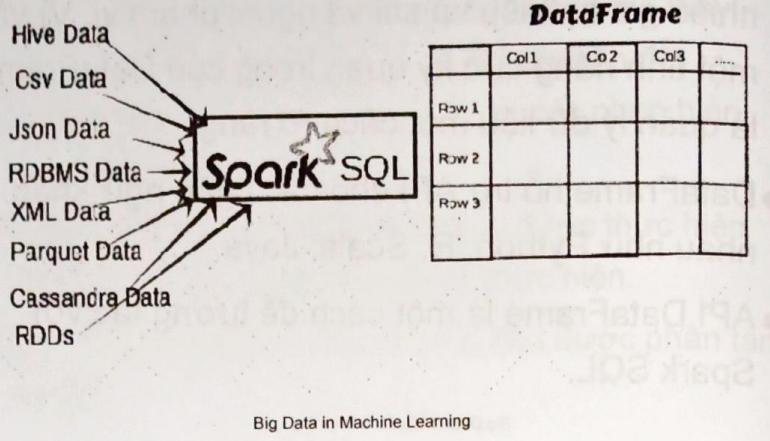
1. Giới thiệu Spark SQL
2. Giới thiệu Spark DataFrame
3. Làm việc với PySpark DataFrame
4. Làm việc với PySpark SQL



Làm việc với DataFrame

☐ Tạo DataFrame

Ways to Create DataFrame in Spark



Làm việc với DataFrame

- Một DataFrame trong Apache Spark có thể được tạo ra theo nhiều cách khác nhau:
 - Từ các Existing RDD.
 - Từ nhiều định dạng dữ liệu khác nhau như JSON, CSV...

Làm việc với DataFrame



• Từ các Existing RDD

```
from pyspark.sql import Row, SQLContext  
  
sc = SparkContext()  
  
sqlContext = SQLContext(sc)  
  
lst = [('John', 10), ('Lyna', 9), ('Samantha', 8), ('Tony', 10)]  
rdd = sc.parallelize(lst)  
people = rdd.map(lambda x: Row(name=x[0], marks=int(x[1])))  
schemaPeople = sqlContext.createDataFrame(people)  
schemaPeople.show()  
  
schemaPeople  
  
DataFrame[marks: bigint, name: string]
```

marks	name
10	John
9	Lyna
8	Samantha
10	Tony

Big Data in Machine Learning

21

Làm việc với DataFrame



• Từ hdfs csv file: dùng spark.read.csv()

```
spark = SparkSession(sc)  
  
file_name = "hdfs://SRVLT2:19000/people.csv"  
  
people = spark.read.csv(file_name, inferSchema = True, header = True)  
people.show(5)  
  
+---+-----+-----+-----+  
| _c0|person_id|      name|    sex|date of birth|  
+---+-----+-----+-----+  
|  0|     100|Penelope Lewis|female| 1990-08-31|  
|  1|     101| David Anthony| male| 1971-10-14|  
|  2|     102|     Ida Shipp|female| 1962-05-24|  
|  3|     103| Joanna Moore|female| 2017-03-10|  
|  4|     104|Lisandra Ortiz|female| 2020-08-05|  
+---+-----+-----+-----+  
only showing top 5 rows
```

Big Data in Machine Learning

22



Làm việc với DataFrame

- Từ hdfs json file: dùng spark.read.json()

```
file_name = "hdfs://SRVLT2:19000/data.json"
```

```
data = spark.read.json(file_name)
data.show(5)
```

```
+-----+-----+-----+-----+-----+
|     id|location|sampling_rate|      sensor|sensorvalues|      timestamp|
+-----+-----+-----+-----+-----+
|5810744647|[12.6, FR, 0, 11...|           null|[22349, 1, [14, N...|[12340422762, 1...|2019-12-20 03:22:01|
|5810744646|[35.2, DE, 0, 107...|           null|[21149, 7, [9, va...|[12340422768, 9...|2019-12-20 03:22:01|
|5810744645|[51.0, DE, 0, 293...|           null|[5811, 1, [14, No...|[12340422757, 6...|2019-12-20 03:22:01|
|5810744644|[34.4, BE, 0, 441...|           null|[8765, 1, [14, No...|[12340422756, 3...|2019-12-20 03:22:01|
|5810744643|[5.0, FR, 0, 1100...|           null|[21693, 1, [14, N...|[12340422754, 7...|2019-12-20 03:22:01|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```



Làm việc với DataFrame



Thao tác trên DataFrame

- Xem kiểu dữ liệu của các column: dùng <DataFrame_name>.printSchema()

Ví dụ:

```
df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- height: integer (nullable = true)
 |-- weight: integer (nullable = true)
 |-- bmi: double (nullable = true)
 |-- age: integer (nullable = true)
 |-- bmc: integer (nullable = true)
 |-- bmd: double (nullable = true)
 |-- fat: integer (nullable = true)
 |-- lean: integer (nullable = true)
 |-- pcfat: double (nullable = true)
```





Làm việc với DataFrame

- Hiển thị dòng dữ liệu:

```
<DataFrame_name>.show(n_rows)
```

- Đếm số dòng dữ liệu:

```
<DataFrame_name>.count()
```

- Xem thống kê dữ liệu:

```
<DataFrame_name>.describe(["column_names"]).show()
```



Làm việc với DataFrame

▪ Ví dụ

```
df.show(5)
```

```
+---+-----+-----+-----+-----+-----+-----+-----+-----+
| id|gender|height|weight| bmi|age| bmc| bmd| fat| lean|pcfat|
+---+-----+-----+-----+-----+-----+-----+-----+-----+
| 1| F| 150| 49| 21.8| 53| 1312| 0.88| 17802| 28600| 37.3|
| 2| M| 165| 52| 19.1| 65| 1309| 0.84| 8381| 40229| 16.8|
| 3| F| 157| 57| 23.1| 64| 1230| 0.84| 19221| 36057| 34.0|
| 4| F| 156| 53| 21.8| 56| 1171| 0.8| 17472| 33094| 33.8|
| 5| M| 160| 51| 19.9| 54| 1681| 0.98| 7336| 40621| 14.8|
+---+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
df.count()
```

1217



Làm việc với DataFrame

```
df.describe().show()
```

	summary	id gender	height	weight	bmi	age	bmi
bmi	fat	lean	pcfat				
count	1217	1217	1217	1217	1217	1217	1217
mean	614.518488085456	null	156.7239112571898	55.14379622021364	22.39539852095314	47.15201314708299	1724.914543960558
stddev	354.4705719473191	null	7.9777256820417035	9.404988688010084	3.0564419447471356	17.27550739904804	363.3490251436472
min	1	F	136	34	14.5	13	695
max	4277	M	19136	95	37.1	88	3046
0.85	40825		63059	48.4			

```
df.describe("height", "weight").show()
```

	summary	height	weight
	count	1217	1217
count	1217	1217	1217
mean	156.7239112571898	55.14379622021364	
stddev	7.9777256820417035	9.404988688010084	
min	136	34	
max	185	95	

Làm việc với DataFrame

- Hiển thị dữ liệu thống kê theo crosstab

- Ví dụ:

```
df_sub.crosstab("height", "gender").show(10)
```

height_gender	F	M
138	2	0
170	1	24
142	11	0
153	73	1
174	0	2
185	0	1
157	31	5
152	54	2
164	11	17
179	0	1

only showing top 10 rows

Làm việc với DataFrame

- Nhóm dữ liệu: dùng

`<Data_farme>.groupby("column_name")`

- Kèm thêm các function: agg, count, max, mean, min, sum...

```
df_sub.groupby('gender').mean('weight').show()
```

gender	avg(weight)
F	52.31090487238979
M	62.02253521126761

gender	count
F	862
M	355

Big Data in Machine Learning

29

Làm việc với DataFrame

- Ví dụ:

```
df_sub.groupby('gender').agg({'weight': 'mean'}).show()
```

gender	avg(weight)
F	52.31090487238979
M	62.02253521126761

```
df_sub.groupby('gender').agg({'weight': 'min', 'height':'min'}).show()
```

gender	min(weight)	min(height)
F	34	136
M	38	146

Big Data in Machine Learning

30

Làm việc với DataFrame

- Tạo sub_DataFrame:

```
<DataFrame_name>.select(["column_names])
```

* Ví dụ:

```
df_sub = df.select("id", "gender", "height", "weight")
df_sub.show(3)

+---+---+---+
| id|gender|height|weight|
+---+---+---+
| 1 | F   | 150  | 49 |
| 2 | M   | 165  | 52 |
| 3 | F   | 157  | 57 |
+---+---+---+
only showing top 3 rows
```

Làm việc với DataFrame

- Hiển thị dữ liệu duy nhất của cột:

```
<DataFrame_name>.select(["column_name
s]).distinct().show()
```

* Ví dụ:

```
df_sub.select('gender').distinct().show()
```

```
+---+
| gender|
+---+
| F |
| M |
+---+
```



Làm việc với DataFrame

- Sắp xếp dữ liệu trong cột: dùng

`<DataFrame_name>.orderBy(<DataFrame_name>.<column_name>.asc()/desc())`

- Ví dụ:

```
df.orderBy(df.age.desc()).show(3)
```

	id	gender	height	weight	bmi	age	bmc	bmd	fat	lean	pcfat
963	M	158	61	24.4	88	1670	1.02	18038	39526	30.5	
712	M	161	47	18.1	87	1678	1.01	12328	32725	26.4	
588	F	142	46	22.8	85	875	0.68	13946	31026	30.4	

only showing top 3 rows



Làm việc với DataFrame

- Tạo cột mới: dùng

`<DataFrame_name>.withColumn('column_name', function)`

- Ví dụ:

```
df_sub.withColumn('bmi', (df_sub.weight/(df_sub.height/100) **2)).show(5)
```

	id	gender	height	weight	bmi
1	F	150	49	21.77777777777778	
2	M	165	52	19.100091827364558	
3	F	157	57	23.124670372023203	
4	F	156	53	21.7784352399737	
5	M	160	51	19.921874999999996	

only showing top 5 rows





Làm việc với DataFrame

- Đổi tên cột: Dùng <DataFrame_name>.

.withColumnRenamed('old_name',
'new_name')

▪ Ví dụ

```
df_sub.withColumnRenamed('gender', 'sex').show(3)
```

	id	sex	height	weight
1	F	150	49	
2	M	165	52	
3	F	157	57	

only showing top 3 rows



Làm việc với DataFrame

- Xóa cột: dùng

<DataFrame_name>.drop(<column_names>)

▪ Ví dụ:

```
df.columns => df.drop('bmi', 'lean').columns
['id', 'gender', 'height', 'weight', 'age', 'bmc', 'bmd', 'fat', 'pcfat']
['id', 'gender', 'height', 'weight', 'age', 'bmc', 'bmd', 'fat', 'pcfat']
```





Làm việc với DataFrame

- Xóa dữ liệu trùng: dùng

<DataFrame_name>.dropDuplicates()

- Ví dụ:

```
people.count()
```

```
100000
```

```
people.dropDuplicates().count()
```

```
100000
```

không có dữ liệu trùng



Làm việc với DataFrame

- Xóa tất cả các dòng có dữ liệu null:

<DataFrame_name>.dropna()

- Ví dụ:

```
people.count()
```

```
100000
```

```
people.dropna().count()
```

```
98080
```



Làm việc với DataFrame



☐ Lọc dữ liệu DataFrame

- Dùng <DataFrame_name>.filter(condition)
- Hoặc <DataFrame_name>.where(condition)

= Ví dụ: # Return rows where name starts with "I" & ends with "p"
people_sub = people.filter(people.name.like('I%p'))
people_sub.count()

7

people_sub.show()

_c0	person_id	name	sex	date of birth
2	102	Ida Shipp	female	1962-05-24
5791	5891	Isaias Stepp	male	1974-04-13
9965	10065	Isabel Sharp	female	1967-11-28
58576	58676	Isadore Tripp	female	1971-06-05
68125	68225	Irene Kulp	female	1986-07-06
71572	71672	Ivan Ropp	male	1975-07-08
92061	92161	Inez Estep	female	1925-09-03

39

Làm việc với DataFrame



= Ví dụ:

Return rows where name contains 'Moore'
people_Moore = people.where(people.name.contains('Moore'))
people_Moore.count()

352

Return rows where name doesn't contain 'Moore'
people_not_Moore = people.where(~people.name.contains('Moore'))
people_not_Moore.count()

99648





Làm việc với DataFrame

▪ Ví dụ:

```
people_1999 = people.filter((people["date of birth"] >= '1999-01-01') & (people["date of birth"] <= '1999-12-31'))  
people_1999.count()  
1366
```

Hoặc

```
people_1999 = people.where((people["date of birth"] >= '1999-01-01') & (people["date of birth"] <= '1999-12-31'))  
people_1999.count()  
1366
```



Làm việc với DataFrame

□ Column string transformation

- Dùng các function trong pyspark.sql.functions như upper, lower,...

▪ Ví dụ:

```
from pyspark.sql.functions import *  
  
people.withColumn('upper_name', upper(col('name'))).show(3)  
  
+---+-----+-----+-----+  
| _c0|person_id|      name|   sex|date of birth|  upper_name|  
+---+-----+-----+-----+  
|  0|     100|Penelope Lewis|female| 1990-08-31|PENELOPE LEWIS|  
|  1|     101| David Anthony| male| 1971-10-14| DAVID ANTHONY|  
|  2|     102|    Ida Shipp|female| 1962-05-24|     IDA SHIPP|  
+---+-----+-----+-----+  
only showing top 3 rows
```



Làm việc với DataFrame

- Chuyển kiểu dữ liệu với `cast()` và các kiểu trong `pyspark.sql.types` như `IntegerType()`, ...

* Ví dụ:

```
from pyspark.sql.types import *

people.withColumn('year', substring('date of birth', pos=0, len= 4)
                  .cast(IntegerType())).show(3)

+-----+-----+-----+-----+
| _c0|person_id|      name|   sex|date of birth|year|
+-----+-----+-----+-----+
|  0|     100|Penelope Lewis|female| 1990-08-31|1990|
|  1|     101| David Anthony| male| 1971-10-14|1971|
|  2|     102|     Ida Shipp|female| 1962-05-24|1962|
+-----+-----+-----+-----+
only showing top 3 rows
```

Làm việc với DataFrame

□ Conditional clause

- `.when(<if condition>, <then x>)`
- `.otherwise()`

Làm việc với DataFrame

- .when(<if condition>, <then x>)

▪ Ví dụ:

```
df.select(df.height, when(df.height <= 150, "Short")).show(5)  
+-----+  
|height|CASE WHEN (height <= 150) THEN Short END|  
+-----+  
| 150 |                               Short |  
| 165 |                               null  |  
| 157 |                               null  |  
| 156 |                               null  |  
| 160 |                               null  |  
+-----+  
only showing top 5 rows
```



Làm việc với DataFrame

- Multiple .when()

▪ Ví dụ:

```
df.select(df.height, when(df.height <= 150, "Short")  
       .when(df.height <= 160, "Midium")).show(3)  
+-----+  
|height|CASE WHEN (height <= 150) THEN Short WHEN (height <= 160) THEN Midium END|  
+-----+  
| 150 |                               Short |  
| 165 |                               null  |  
| 157 |                               Midium |  
+-----+  
only showing top 3 rows
```



Làm việc với DataFrame

- .otherwise()

▪ Ví dụ:

```
df.select(df.height, when(df.height <= 150, "Short")
          .when(df.height <= 160, "Medium")
          .otherwise("Tall")).show(3)
```

```
+-----+
|height|CASE WHEN (height <= 150) THEN Short WHEN (height <= 160) THEN Medium ELSE Tall END|
+-----+
| 150|           Short|
| 165|           Tall|
| 157|           Medium|
+-----+
only showing top 3 rows
```



Làm việc với DataFrame

□ User defined functions – UDFs

- Là Python method
- Được đóng gói trong
pyspark.sql.functions.udf
- Được lưu trữ như variable
- Được gọi như một function Spark thông thường





Làm việc với DataFrame

▪ Ví dụ:

```
from pyspark.sql.functions import udf

def bmi_type(bmi):
    bmi_type = ""
    if bmi <= 18.5:
        bmi_type = "Underweight"
    elif bmi <= 24.9:
        bmi_type = "Normal weight"
    elif bmi <= 29.9:
        bmi_type = "Overweight"
    else:
        bmi_type = "Obesity"
    return bmi_type

udfBMI = udf(bmi_type, StringType())
df_sub.withColumn('bmi_type', udfBMI(df_sub.bmi)).show(3)

+---+-----+-----+-----+-----+
| id|gender|height|weight|      bmi|   bmi_type|
+---+-----+-----+-----+-----+
|  1|     F|    150|     49| 21.77777777777778|Normal weight|
|  2|     M|    165|     52|19.100091827364558|Normal weight|
|  3|     F|    157|     57|23.124670372023203|Normal weight|
+---+-----+-----+-----+-----+
only showing top 3 rows
```



Làm việc với DataFrame

□ Trực quan hóa dữ liệu

- Chuyển PySpark DataFrame thành DataFrame: dùng
`<Spark_DataFrame_name>.toPandas()`
- Sử dụng các biểu đồ dành cho DataFrame
để vẽ như histogram, barchart, ...



Làm việc với DataFrame



• Ví dụ:

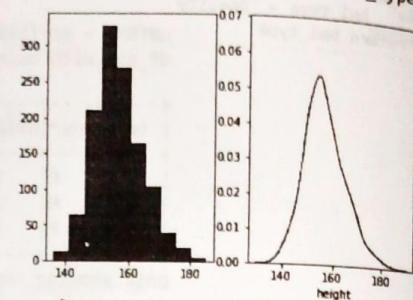
```
import matplotlib.pyplot as plt
import seaborn as sns

# Check the column names of df_sub
print("The column names of df_sub are", df_sub.columns)

# Convert to Pandas DataFrame
df_pandas = df_sub.toPandas()

The column names of df_sub are ['id', 'gender', 'height', 'weight', 'bmi', 'bmi_type']

plt.subplot(1,2,1)
# Create a histogram
plt.hist(df_pandas["height"])
plt.subplot(1,2,2)
sns.distplot(df_pandas["height"])
plt.show()
```



Big Data in Machine

51

Nội dung



1. Giới thiệu Spark SQL
2. Giới thiệu Spark DataFrame
3. Làm việc với PySpark DataFrame
4. Làm việc với PySpark SQL



Big Data in Machine Learning

52



Làm việc với PySpark SQL

□ Thực thi SQL Query

- Tạo table tạm/view: dùng

```
<DataFrameName>.createOrReplaceTempView("table_name")
```

- Ví dụ: Tạo table tạm/view people từ df_people

```
# Create a temporary table "people"
people_df.createOrReplaceTempView("people")
```



Làm việc với PySpark SQL



- Thực thi SQL Query: dùng spark.sql(query)

- query = 'SELECT field1, field2,... FROM table'

- Ví dụ:

```
# Construct a query to select the names of the people
# from the temporary table "people"
query = 'SELECT name FROM people'
```

```
# Assign the result of Spark's query to people_df_names
people_df_names = spark.sql(query)
```

```
# Print the top 10 names of the people
people_df_names.show(10)
```

name
Penelope Lewis
David Anthony
Ida Shipp
Joanna Moore
Lisandra Ortiz
David Simmons
Edward Hudson
Albert Jones
Leonard Cavender
Everett Vadala





Làm việc với PySpark SQL

- Truy vấn có điều kiện với **where**: dùng `spark.sql(query)`

- `query = 'SELECT field1, field2,... FROM table WHERE conditional'`

- Ví dụ:

```
# Filter the people table to select female sex
people_female_df = spark.sql('SELECT * FROM people WHERE sex=="female"')

# Filter the people table DataFrame to select male sex
people_male_df = spark.sql('SELECT * FROM people WHERE sex=="male"')

# Count the number of rows in both DataFrames
print("There are ", people_female_df.count(), " rows in the people_female_df and",
      people_male_df.count(), "rows in the people_male_df")

There are 49014 rows in the people_female_df and 49066 rows in the people_male_df
```



Làm việc với PySpark SQL

- Truy vấn với LIKE

- Ví dụ:

```
people_Vadala = spark.sql('SELECT * FROM people WHERE name like "%Vadala"')
people_Vadala.show()

+-----+-----+-----+-----+
| _c0|person_id|      name|    sex|date of birth|
+-----+-----+-----+-----+
|   9|      109|Everett Vadala| male| 2005-05-24|
| 77207|     77307| Marlyn Vadala|female| 1992-07-02|
+-----+-----+-----+-----+
```





Làm việc với PySpark SQL

▪ Ví dụ 2:

```
people_Albert = spark.sql('SELECT * FROM people WHERE name like "Albert%"')
people_Albert.count()
```

234

```
people_Albert.show(10);
```

_c0 person_id		name sex date of birth
7 187	Albert Jones male	1990-09-13
380 480	Albert Guillory male	2001-02-10
1577 1677	Albert Miller male	1976-09-16
1914 2014	Albert Goetz male	1978-03-06
2011 2111	Albert Clever male	1985-12-22
3193 3293	Albert Griffin male	1988-11-03
3419 3519	Albert Amundsen male	2031-09-20
3760 3860	Albert Pritchard male	1973-01-02
4714 4814	Albert Omeara male	1989-06-07
5777 5877	Albert Villarreal male	1946-04-11

only showing top 10 rows

Big Data in Machine Learning

57

Làm việc với PySpark SQL



• Truy vấn với SUBSTRING(column_name, from, length)

▪ Ví dụ:

```
name_sub = spark.sql('SELECT SUBSTRING(name, 8, 10) FROM people WHERE name like "Albert%"')
name_sub.show(10)
```

substring(name, 8, 10)
Jones
Guillory
Miller
Goetz
Clever
Griffin
Amundsen
Pritchard
Omeara
Villarreal

only showing top 10 rows

Big Data in Machine Learning

58

Làm việc với PySpark SQL

□ SQL vs DataFrame API

• Select

```
spark.sql('SELECT name FROM people').show(5)
```

```
+-----+  
|      name|  
+-----+  
|Penelope Lewis|  
| David Anthony|  
|   Ida Shipp|  
| Joanna Moore|  
|Lisandra Ortiz|  
+-----+  
only showing top 5 rows
```

```
people_df.select("name").show(5)
```

```
+-----+  
|      name|  
+-----+  
|Penelope Lewis|  
| David Anthony|  
|   Ida Shipp|  
| Joanna Moore|  
|Lisandra Ortiz|  
+-----+  
only showing top 5 rows
```



Làm việc với PySpark SQL

□ SQL vs DataFrame API

• Where

```
people_female_df = spark.sql('SELECT * FROM people WHERE sex=="female")  
  
people_female_df.count()  
  
49014
```

```
people_female_1 = people_df.where(people_df.sex == "female")  
people_female_1.count()  
  
49014
```



Làm việc với PySpark SQL



SQL vs DataFrame API

• Like

```
people_Vadala = spark.sql('SELECT * FROM people WHERE name like "%Vadala"')
people_Vadala.show()
```

_c0[person_id]	name	sex	date of birth	DOB
9	109 Everett Vadala	male	2005-05-24	2005-05-24
77287	77307 Marilyn Vadala	female	1992-07-02	1992-07-02

```
people_Vadala_1 = people_df.where(people_df.name.like("%Vadala"))
people_Vadala_1.show()
```

_c0[person_id]	name	sex	date of birth	DOB
9	109 Everett Vadala	male	2005-05-24	2005-05-24
77287	77307 Marilyn Vadala	female	1992-07-02	1992-07-02

Big Data in Machine Learning

61

Làm việc với PySpark SQL



SQL vs DataFrame API

• Startswith/ endswith

```
people_Albert = spark.sql('SELECT * FROM people WHERE name like "Albert%"')
people_Albert.count()
```

234

```
people_Albert_1 = people_df.where(people_df.name.startswith("Albert"))
people_Albert_1.count()
```

234

Big Data in Machine Learning

62

Làm việc với PySpark SQL

□ SQL vs DataFrame API

● Substring

```
name_sub = spark.sql('SELECT SUBSTRING(name, 8, 10) FROM people WHERE name like "Albert%"')
name_sub.show(5)
```

```
+-----+
|substring(name, 8, 10)|
+-----+
|          Jones|
|        Guillory|
|       Miller|
|      Goetz|
|     Clever|
+-----+
only showing top 5 rows
```

```
name_sub_1 = people_df.select(substring(people_df.name, pos= 8, len = 10))\
    .where(people_df.name.like("Albert%"))
name_sub_1.show(5)
```

```
+-----+
|substring(name, 8, 10)|
+-----+
|          Jones|
|        Guillory|
|       Miller|
|      Goetz|
|     Clever|
+-----+
only showing top 5 rows
```

Làm việc với PySpark SQL

□ Thao tác với dữ liệu

- Tìm và thay thế giá trị null: dùng

`<DataFrame_name>.na.fill(value)`

- Tìm giá trị và thay thế bằng giá trị mới: dùng

`<DataFrame_name>.replace(old_value,
new_value)`



Làm việc với PySpark SQL

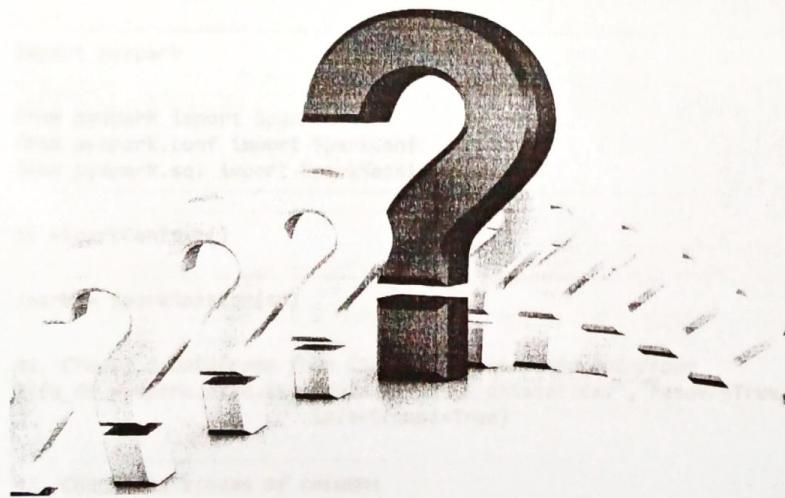


- Phân vùng dữ liệu: dùng repartition() của RDD

- Ví dụ:

```
people_df.repartition(10).rdd.getNumPartitions()
```

10





Chapter 4: Spark SQL & Dataframe

Ex 1: Fifa2018

Cho tập tin Fifa2018_dataset.csv

Yêu cầu:

1. Đọc tập tin Fifa2018_dataset.csv vào fifa_df.
2. In schema của fifa_df. Hiển thị 2 dòng đầu tiên của dữ liệu. Cho biết dữ liệu có bao nhiêu dòng?
3. Tạo view 'fifa_table' từ fifa_df
4. Hãy thực hiện SQL Query để lấy cột Age của các vận động viên có Nationality là "Germany" => fifa_germany_age. Hiển thị 3 dòng đầu của dữ liệu. In thống kê dữ liệu
5. Trực quan hóa dữ liệu fifa_germany_age. Nhận xét biểu đồ.
6. Từ fifa_df, cho biết mỗi độ tuổi có bao nhiêu cầu thủ. Độ tuổi trung bình của cầu thủ mỗi quốc gia là bao nhiêu?
7. Từ fifa_df, cho biết "Age" nhỏ nhất, "Age" lớn nhất, "Strength" nhỏ nhất, "Strength" lớn nhất
8. Liệt kê danh sách các "Club" (đuy nhất) theo 2 cách với Dataframe fifa_df và SQL query với fifa_table.
9. Từ fifa_df, sắp xếp dữ liệu giảm dần theo Age => fifa_df_desc.
10. Có bao nhiêu cầu thủ trong "Name" có "Cristiano" theo 2 cách với Dataframe fifa_df và SQL query với fifa_table. In tên các cầu thủ này.

```
In [1]: import findspark
findspark.init()
```

```
In [2]: import pyspark
```

```
In [3]: from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
```

```
In [4]: sc =SparkContext()
```

```
In [5]: spark = SparkSession(sc)
```

```
In [6]: #1. Create a DataFrame from CSV file. # Load the Dataframe
fifa_df = spark.read.csv("data/Fifa2018_dataset.csv", header=True,
inferSchema=True)
```

```
In [7]: #2. Check the schema of columns
# fifa_df.printSchema()
```

```
In [8]: # Show the first 3 observations
for row in fifa_df.head(3):
    print(row)
    print('\n')
#fifa_df.show(3)
```

```
Row(_c0=0, Name='Cristiano Ronaldo', Age=32, Photo='https://cdn.sofifa.org/48/18/players/20801.png', Nationality='Portugal', Flag='https://cdn.sofifa.org/flags/38.png', Overall=94, Potential=94, Club='Real Madrid CF', Club Logo='https://cdn.sofifa.org/24/18/teams/243.png', Value='€95.5M', Wage='€565K', Special=222, Acceleration='89', Aggression='63', Agility='89', Balance='63', Ball control='93', Composure='95', Crossing='85', Curve='81', Dribbling='91', Finishing='94', Free kick accuracy='76', GK diving='7', GK handling='11', GK kicking='15', GK positioning='14', GK reflexes='11', Heading accuracy='88', Interceptions='29', Jumping='95', Long passing='77', Long shots='92', Marking='22', Penalties='85', Positioning='95', Reactions='96', Short passing='83', Shot power='94', Sliding tackle='23', Sprint speed='91', Stamina='92', Standing tackle='31', Strength='80', Vision='85', Volleys='88', CAM=89.0, CB=53.0, CDM=62.0, CF=91.0, CM=82.0, ID=20801, LAM=89.0, LB=61.0, LCB=53.0, LCM=82.0, LDM=62.0, LF=91.0, LM=89.0, LS=92.0, LW=91.0, LWB=66.0, Preferred Positions='ST LW ', RAM=89.0, RB=61.0, RCB=53.0, RCM=82.0, RDM=62.0, RF=91.0, RM=89.0, RS=92.0, RW=91.0, RWB=66.0, ST=92.0)
```

```
Row(_c0=1, Name='L. Messi', Age=30, Photo='https://cdn.sofifa.org/48/18/players/158023.png', Nationality='Argentina', Flag='https://cdn.sofifa.org/flags/52.png', Overall=93, Potential=93, Club='FC Barcelona', Club Logo='https://cdn.sofifa.org/24/18/teams/241.png', Value='€105M', Wage='€565K', Special=2154, Acceleration='92', Aggression='48', Agility='90', Balance='95', Ball control='95', Composure='96', Crossing='77', Curve='89', Dribbling='97', Finishing='95', Free kick accuracy='90', GK diving='6', GK handling='11', GK kicking='15', GK positioning='14', GK reflexes='8', Heading accuracy='71', Interceptions='22', Jumping='68', Long passing='87', Long shots='88', Marking='13', Penalties='74', Positioning='93', Reactions='95', Short passing='88', Shot power='85', Sliding tackle='26', Sprint speed='87', Stamina='73', Standing tackle='28', Strength='59', Vision='90', Volleys='85', CAM=92.0, CB=45.0, CDM=59.0, CF=92.0, CM=84.0, ID=158023, LAM=92.0, LB=57.0, LCB=45.0, LCM=84.0, LDM=59.0, LF=92.0, LM=90.0, LS=88.0, LW=91.0, LWB=62.0, Preferred Positions='RW ', RAM=92.0, RB=57.0, RCB=45.0, RCM=84.0, RDM=59.0, RF=92.0, RM=90.0, RS=88.0, RW=91.0, RWB=62.0, ST=88.0)
```

```
In [9]: # Print the total number of rows
print("There are {} rows in the fifa_df DataFrame".format(fifa_df.count()))
```

There are 17981 rows in the fifa_df DataFrame

```
In [10]: #3. Create a temporary view of fifa_df
fifa_df.createOrReplaceTempView('fifa_table')
```



```
In [11]: # 4.  
# Construct the "query"  
query = '''SELECT Age FROM fifa_table WHERE Nationality == "Germany"'''  
# Apply the SQL "query"  
fifa_germany_age = spark.sql(query)
```

```
In [12]: fifa_germany_age.show(3)
```

```
+---+  
|Age|  
+---+  
| 31|  
| 27|  
| 28|  
+---+  
only showing top 3 rows
```

```
In [13]: # Generate basic statistics  
fifa_germany_age.describe().show()
```

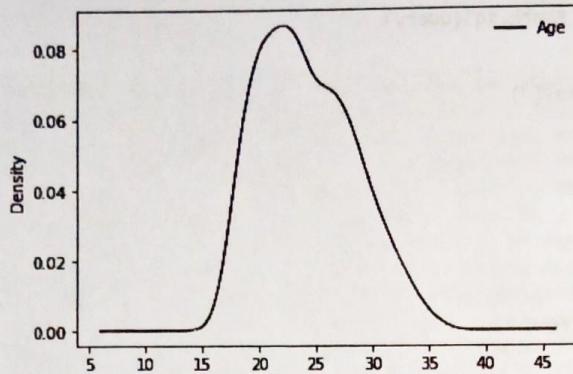
```
+-----+  
|summary|          Age|  
+-----+-----+  
| count|      1140|  
| mean|24.20263157894737|  
| stddev|4.197096712293752|  
| min|      16|  
| max|      36|  
+-----+
```

```
In [14]: #5.
```

```
# Convert fifa_germany_age to fifa_germany_age_pandas DataFrame  
fifa_germany_age_pandas = fifa_germany_age.toPandas()
```

```
In [15]: import matplotlib.pyplot as plt
```

```
In [16]: # Plot the 'Age' density of Germany Players  
fifa_germany_age_pandas.plot(kind='density')  
plt.show()
```



```
In [17]: #6.  
fifa_df.groupBy("Age").count().show()
```

Age	count
31	671
34	272
28	1051
26	1202
27	1152
44	2
22	1324
47	1
16	13
20	1245
40	8
19	1069
41	3
43	2
37	69
17	258
35	191
39	20
23	1394
38	36

only showing top 20 rows



```
In [18]: fifa_df.groupBy("Nationality").avg("Age").show()
```

Nationality	avg(Age)
Chad	25.0
Russia	25.23202614379085
Paraguay	26.10144927536232
Senegal	25.046511627906977
Sweden	25.119565217391305
Guyana	28.0
Eritrea	32.0
Philippines	25.666666666666668
Fiji	29.0
Turkey	25.127147766323824
Iraq	26.0
Germany	24.20263157894737
St Kitts Nevis	26.666666666666668
Comoros	27.11111111111111
Afghanistan	22.0
Ivory Coast	24.10891089108911
Sudan	22.5
France	24.634969325153374
Greece	24.418367346938776
Kosovo	23.9375

only showing top 20 rows

In [19]: #7.

```
from pyspark.sql import functions as F
fifa_df.groupBy("Nationality").agg(F.min("Age"),
                                    F.max("Age"),
                                    F.min("Strength"),
                                    F.max("Strength")).show()
```

Nationality	min(Age)	max(Age)	min(Strength)	max(Strength)
Chad	24	26	73	79
Paraguay	18	37	33	91
Russia	17	37	26	93
Senegal	18	34	37	94
Sweden	17	37	21	91
Guyana	25	34	47	75
Eritrea	32	32	85	85
Philippines	22	28	42	76
Fiji	29	29	57	57
Turkey	17	39	30	90
Iraq	21	30	55	86
Germany	16	36	28	94
St Kitts Nevis	23	32	32	84
Comoros	23	32	28	82
Afghanistan	19	27	40	58
Ivory Coast	17	34	38	92
Sudan	22	23	41	62
France	16	40	26	93
Greece	18	38	32	91
Kosovo	18	33	30	90

only showing top 20 rows



```
In [20]: #8.
fifa_df.select("Club").distinct().show()
+-----+
| Club|
+-----+
| Palermo|
| Yeovil Town|
| 1. FC Union Berlin|
| Santiago Wanderers|
| Carpi|
| Evkur Yeni Malaty...|
| Sagan Tosu|
| FC Basel|
| Argentinos Juniors|
| Karlsruher SC|
| Lorca Deportiva CF|
| SC Paderborn 07|
| Cheltenham Town|
| San Lorenzo de Al...|
| SC Freiburg|
| SpVgg Unterhaching|
| Atletico Nacional...|
| Universidad Católica|
| GFC Ajaccio|
| FC Luzern|
+-----+
only showing top 20 rows
```

```
In [21]: query = '''SELECT DISTINCT Club FROM fifa_table'''  
# Apply the SQL "query"  
fifa_clubs = spark.sql(query)  
fifa_clubs.show()
```

Club
Palermo
Yeovil Town
1. FC Union Berlin
Santiago Wanderers
Carpi
Evkur Yeni Malaty...
Sagan Tosu
FC Basel
Argentinos Juniors
Karlsruher SC
Lorca Deportiva CF
SC Paderborn 07
Cheltenham Town
San Lorenzo de Al...
SC Freiburg
SpVgg Unterhaching
Atletico Nacional...
Universidad Católica
GFC Ajaccio
FC Luzern

only showing top 20 rows

```
In [22]: #9.  
fifa_df_desc = fifa_df.orderBy(fifa_df["Age"].desc())
```

```
In [23]: fifa_df_desc.select("Name", "Age", "Strength").show(3)
```

Name	Age	Strength
B. Richardson	47	47
E. El Hadary	44	73
O. Pérez	44	66

only showing top 3 rows

```
In [24]: #10.  
people_with_Cristiano = fifa_df.where(fifa_df["Name"].contains("Cristiano"))  
people_with_Cristiano.count()
```

```
Out[24]: 3
```



In [25]: `people_with_Cristiano.select("Name").show()`

```
+-----+
|      Name|
+-----+
|Cristiano Ronaldo|
|    Cristiano|
|    Cristiano|
+-----+
```

In [26]: `query = '''SELECT * FROM fifa_table WHERE Name like "%Cristiano%"'''`
`people_with_Cristiano_2 = spark.sql(query)`
`people_with_Cristiano_2.count()`

Out[26]: 3

In [27]: `people_with_Cristiano_2.select("Name").show()`

```
+-----+
|      Name|
+-----+
|Cristiano Ronaldo|
|    Cristiano|
|    Cristiano|
+-----+
```



Chapter 4: Spark SQL & Dataframe

Ex2: Flights

Cho tập tin flights_small.csv và airports.csv

Yêu cầu:

1. Đọc tập tin flights_small.csv vào data.
2. In schema của data. Hiển thị 3 dòng đầu tiên của dữ liệu. Cho biết dữ liệu có bao nhiêu dòng?
3. Từ data hãy tạo một view có tên là flights_small
4. Từ view flights_small hãy tạo một dataframe có tên là flights. Hiển thị 3 dòng đầu tiên của dữ liệu
5. Trong flights, hãy tạo thêm cột duration_hours (= air_time/60)
6. Lọc dữ liệu các chuyến bay có distance > 2000 => long_flights1. Cho biết có bao nhiêu dòng dữ liệu thỏa điều kiện này? (bằng cả SQL và Dataframe)
7. Lọc dữ liệu các chuyến bay có $300 \geq \text{air_time} \geq 600$ => time_flights. Cho biết có bao nhiêu dòng dữ liệu thỏa điều kiện này? (bằng cả SQL và Dataframe)
8. Tạo dữ liệu selected1 từ flights với các cột "origin", "dest", "carrier". Tạo biến điều kiện lọc thứ nhất filterA là các chuyến bay có origin là "SEA", tạo biến điều kiện lọc thứ hai filterB là các chuyến bay có dest là "PDX". Từ đó hãy tạo Dataframe kết quả selected2 từ selected1 thỏa cả filterA và filterB. Cho biết có bao nhiêu dòng dữ liệu thỏa điều kiện trên.
9. Tạo riêng một biến avg_speed (alias "avg_speed") là tốc độ trung bình của chuyến bay (tính theo giờ) $\text{distance}/(\text{air_time}/60)$. Tạo Dataframe speed1 từ flights với các cột "origin", "dest", "tailnum" và cột "avg_speed" vừa tạo
10. Tạo Dataframe speed2 từ flights với các cột "origin", "dest", "tailnum" và cột $\text{avg_speed} = \text{distance}/(\text{air_time}/60)$ trong cùng một lệnh (dùng selectExpr)
11. Sử dụng aggregation method để: tìm air_time nhỏ nhất của các chuyến bay có origin là "PDX"; tìm distance lớn nhất của các chuyến bay có origin là "SEA"; tìm tổng duration các chuyến bay (theo giờ).
12. Nhóm dữ liệu và đếm số chuyến bay theo từng "tailnum"; nhóm dữ liệu và đếm số chuyến bay theo từng "origin"; nhóm dữ liệu và tính trung bình của air_time theo từng "origin"
13. Nhóm các chuyến bay theo "month", "dest" => by_month_dest. Tính trung bình "dep_delay" theo by_month_dest. Tính std "dep_delay" theo by_month_dest.
14. Đọc tập tin airports.csv vào airports. In schema của airports. Hiển thị 3 dòng đầu tiên của dữ liệu. Cho biết dữ liệu có bao nhiêu dòng?
15. Đổi tên cột "faa" trong airports thành "dest".
16. Tạo một Dataframe mới bằng cách kết hợp flights và airports theo "dest"

In [1]: `import findspark
findspark.init()`



```
In [2]: import pyspark

In [3]: from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession

In [4]: sc = SparkContext()

In [5]: spark = SparkSession(sc)

In [6]: #1.
    data = spark.read.csv("flights_small.csv", header=True,
                          inferSchema=True)

In [7]: #2.
    data.printSchema()

    root
    |-- year: integer (nullable = true)
    |-- month: integer (nullable = true)
    |-- day: integer (nullable = true)
    |-- dep_time: string (nullable = true)
    |-- dep_delay: string (nullable = true)
    |-- arr_time: string (nullable = true)
    |-- arr_delay: string (nullable = true)
    |-- carrier: string (nullable = true)
    |-- tailnum: string (nullable = true)
    |-- flight: integer (nullable = true)
    |-- origin: string (nullable = true)
    |-- dest: string (nullable = true)
    |-- air_time: string (nullable = true)
    |-- distance: integer (nullable = true)
    |-- hour: string (nullable = true)
    |-- minute: string (nullable = true)

In [8]: print("Number of rows:", data.count())
    Number of rows: 10000

In [9]: #3.
    data.createOrReplaceTempView("flights_small")

In [10]: #4. Create the DataFrame flights
    flights = spark.table("flights_small")
```

In [11]: # Show the head
`flights.show(3)`

```
+---+---+---+---+---+---+---+
|year|month|day|dep_time|dep_delay|arr_time|arr_delay|carrier|tailnum|flight|or
igin|dest|air_time|distance|hour|minute|
+---+---+---+---+---+---+---+
|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780|
SEA| LAX| 132| 954| 6| 58|
|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851|
SEA| HNL| 360| 2677| 10| 40|
|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755|
SEA| SFO| 111| 679| 14| 43|
+---+---+---+---+---+---+---+
only showing top 3 rows
```

In [12]: #5. Add duration_hrs
`flights = flights.withColumn("duration_hrs", flights.air_time/60)`

In [13]: # `flights.show(3)`

In [14]: # Show the first 3 observations
`for row in flights.head(3):
 print(row)
 print('\n')`

```
Row(year=2014, month=12, day=8, dep_time='658', dep_delay=-7, arr_time='935',
arr_delay=-5, carrier='VX', tailnum='N846VA', flight=1780, origin='SEA', dest='LAX',
air_time='132', distance=954, hour='6', minute='58', duration_hrs=2.2)
```

```
Row(year=2014, month=1, day=22, dep_time='1040', dep_delay=5, arr_time='150
5', arr_delay=5, carrier='AS', tailnum='N559AS', flight=851, origin='SEA', de
st='HNL', air_time='360', distance=2677, hour='10', minute='40', duration_hrs=
6.0)
```

```
Row(year=2014, month=3, day=9, dep_time='1443', dep_delay=-2, arr_time='165
2', arr_delay=2, carrier='VX', tailnum='N847VA', flight=755, origin='SEA', de
st='SFO', air_time='111', distance=679, hour='14', minute='43', duration_hrs=1.
85)
```

In [15]: #6. Filter flights by passing a string
`long_flights2 = flights.filter("distance > 2000")`

In [16]: `print("Number of rows (distance >2000):", long_flights2.count())`

Number of rows (distance >2000): 1481



```
In [17]: # Print the data to check
long_flights2.show(2)

+---+-----+-----+-----+-----+-----+-----+
|year|month|day|dep_time|dep_delay|arr_time|arr_delay|carrier|tailnum|flight|origin|dest|air_time|distance|hour|minute|duration_hrs|
+---+-----+-----+-----+-----+-----+-----+
|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851|
SEA| HNL| 360| 2677| 10| 40| 6.0|
|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24|
SEA| BOS| 290| 2496| 21| 56| 4.833333333333333|
+---+-----+-----+-----+-----+-----+-----+
only showing top 2 rows
```

```
In [18]: # Construct the "query"
query = '''SELECT * FROM flights_small WHERE distance > 2000'''
# Apply the SQL "query"
long_flights2_sql = spark.sql(query)
```

```
In [19]: long_flights2_sql.count()
```

```
Out[19]: 1481
```

```
In [20]: #7.
time_flights = flights.filter("air_time >= 300 and air_time <= 600")
```

```
In [21]: time_flights.show(2)
```

```
+---+-----+-----+-----+-----+-----+-----+
|year|month|day|dep_time|dep_delay|arr_time|arr_delay|carrier|tailnum|flight|origin|dest|air_time|distance|hour|minute|duration_hrs|
+---+-----+-----+-----+-----+-----+-----+
|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851|
SEA| HNL| 360| 2677| 10| 40| 6.0|
|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29|
SEA| OGG| 333| 2640| 9| 54| 5.55|
+---+-----+-----+-----+-----+-----+-----+
only showing top 2 rows
```

```
In [22]: print("Number of rows (air_time > 300 and air_time < 600):",
          time_flights.count())
```

```
Number of rows (air_time > 300 and air_time < 600): 440
```

```
In [23]: # Construct the "query"
query = '''SELECT * FROM flights_small WHERE air_time between 300 and 600'''
# Apply the SQL "query"
time_flights_sql = spark.sql(query)

In [24]: time_flights_sql.count()
Out[24]: 440

In [25]: #8.
    # Select set of columns
    selected1 = flights.select(flights.origin,
                                flights.dest, flights.carrier)

    # Define first filter
    filterA = flights.origin == "SEA"

    # Define second filter
    filterB = flights.dest == "PDX"

    # Filter the data, first by filterA then by filterB
    selected2 = selected1.filter(filterA).filter(filterB)

In [26]: selected2.show(5)
+---+---+---+
|origin|dest|carrier|
+---+---+---+
|  SEA| PDX|     00|
+---+---+---+
only showing top 5 rows

In [27]: print("Number of rows (origin = 'SEA' & dest = 'PDX'):",selected2.count())
Number of rows (origin = 'SEA' & dest = 'PDX'): 157

In [28]: #9.
    # Define avg_speed
    avg_speed = (flights.distance/(flights.air_time/60)).alias("avg_speed")

    # Select the correct columns
    speed1 = flights.select("origin", "dest", "tailnum", avg_speed)
```



In [29]: speed1.show(2)

```
+-----+-----+
|origin|dest|tailnum|      avg_speed|
+-----+-----+
|  SEA| LAX| N846VA|433.6363636363636|
|  SEA| HNL| N559AS|446.1666666666667|
+-----+-----+
only showing top 2 rows
```

In [30]: #10. Create the same table using a SQL expression

```
speed2 = flights.selectExpr("origin", "dest", "tailnum",
                            "distance/(air_time/60) as avg_speed")
```

In [31]: speed2.show(2)

```
+-----+-----+
|origin|dest|tailnum|      avg_speed|
+-----+-----+
|  SEA| LAX| N846VA|433.6363636363636|
|  SEA| HNL| N559AS|446.1666666666667|
+-----+-----+
only showing top 2 rows
```

In [32]: from pyspark.sql.types import IntegerType

In [33]: #11.

```
flights = flights.withColumn("air_time",
                             flights["air_time"].cast(IntegerType()))
# Find the shortest time from PDX in terms of air_time
flights.filter(flights.origin == "PDX").groupBy().min("air_time").show()

+-----+
|min(air_time)|
+-----+
|      24|
+-----+
```

In [34]: # Find the longest distance from SEA in terms of distance

```
flights.filter(flights.origin == "SEA").groupBy().max("distance").show()
```

```
+-----+
|max(distance)|
+-----+
|      2724|
+-----+
```



```
In [35]: # Total hours in the air
flights.withColumn("duration_hrs",
                    flights.air_time/60).groupBy().sum("duration_hrs").show()
```

```
+-----+
| sum(duration_hrs) |
+-----+
| 25289.600000000126 |
+-----+
```

```
In [36]: from pyspark.sql.functions import avg
```

```
In [37]: #12.
# Group by tailnum
by_plane = flights.groupBy("tailnum")
```

```
In [38]: # Number of flights each plane made
by_plane.count().show()
```

```
+-----+-----+
|tailnum|count|
+-----+-----+
| N442AS | 38 |
| N102UW | 2 |
| N3647Z | 4 |
| N38451 | 4 |
| N73283 | 4 |
| N513UA | 2 |
| N954WN | 5 |
| N388DA | 3 |
| N567AA | 1 |
| N516UA | 2 |
| N927DN | 1 |
| N8322X | 1 |
| N466SW | 1 |
| N6700 | 1 |
| N607AS | 45 |
| N622SW | 4 |
| N584AS | 31 |
| N914WN | 4 |
| N654AW | 2 |
| N336NW | 1 |
+-----+-----+
```

only showing top 20 rows

```
In [39]: # Group by origin
by_origin = flights.groupBy("origin").count()
```



```
In [40]: by_origin.show()
```

origin	count
SEA	6754
PDX	3246

```
In [41]: # Average air_time
flights.groupBy("origin").avg("air_time").show()
```

origin	avg(air_time)
SEA	160.4361496051259
PDX	137.11543248288737

```
In [42]: from pyspark.sql.types import IntegerType
import pyspark.sql.functions as F
```

```
print("Number of airports", airports.count())
Number of airports 3097
```

3/16/2020

Ex2_SparkDataframe_Flights - Jupyter Notebook

```
In [43]: # 13
flights = flights.withColumn("dep_delay",
                             flights["dep_delay"].cast(IntegerType()))
# Group by month and dest
by_month_dest = flights.groupBy("month", "dest")
# Average departure delay by month and destination
by_month_dest.avg("dep_delay").show()
+-----+-----+
|month|dest|      avg(dep_delay)|
```

month	dest	avg(dep_delay)
4	PHX	1.683333333333333
1	RDM	-1.625
5	ONT	3.555555555555554
7	OMA	-6.5
8	MDW	7.45
6	DEN	5.418181818181818
5	IAD	-4.0
12	COS	-1.0
11	ANC	7.529411764705882
5	AUS	-0.75
5	COS	11.666666666666666
2	PSP	0.6
4	ORD	0.14285714285714285
10	DFW	18.176470588235293
10	DCA	-1.5
8	JNU	18.125
11	KOA	-1.0
10	OMA	-0.6666666666666666
6	ONT	9.625
3	MSP	3.2

```
+-----+
only showing top 20 rows
```



Ex2_SparkDataframe_Flights - Jupyter Notebook

In [44]: # Standard deviation of departure delay
by_month_dest.agg(F.stddev("dep_delay")).show()

month	dest	stddev_samp(dep_delay)
4	PHX	15.003380033491737
1	RDM	8.830749846821778
5	ONT	18.895178691342874
7	OMA	2.1213203435596424
8	MDW	14.467659032985843
6	DEN	13.536905534420026
5	IAD	3.8078865529319543
12	COS	1.4142135623730951
11	ANC	18.604716401245316
5	AUS	4.031128874149275
5	COS	33.38163167571851
2	PSP	4.878524367060187
4	ORD	11.593882803741764
10	DFW	45.53019017606675
10	DCA	0.7071067811865476
8	JNU	40.79368823727514
11	KOA	1.8708286933869707
10	OMA	5.8594652770823155
6	ONT	25.98316762829351
3	MSP	21.556779370817555

only showing top 20 rows

In [45]: #14.

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

In [46]: airports.printSchema()

```
root
 |-- faa: string (nullable = true)
 |-- name: string (nullable = true)
 |-- lat: double (nullable = true)
 |-- lon: double (nullable = true)
 |-- alt: integer (nullable = true)
 |-- tz: integer (nullable = true)
 |-- dst: string (nullable = true)
```

In [47]: print("Number of lines:", airports.count())

Number of lines: 1397

3/16/2020

Ex2_SparkDataframe_Flights - Jupyter Notebook

In [48]: # Examine the data
airports.show(3)

faa	name	lat	lon	alt	tz	dst
04G Lansdowne Airport	41.1304722	-80.6195833	1044	-5	A	
06A Moton Field Munic...	32.4605722	-85.6800278	264	-5	A	
06C Schaumburg Regional	41.9893408	-88.1012428	801	-6	A	

only showing top 3 rows

In [49]: #15. Rename the faa column
airports = airports.withColumnRenamed("faa", "dest")

In [50]: airports.show(3)

dest	name	lat	lon	alt	tz	dst
04G Lansdowne Airport	41.1304722	-80.6195833	1044	-5	A	
06A Moton Field Munic...	32.4605722	-85.6800278	264	-5	A	
06C Schaumburg Regional	41.9893408	-88.1012428	801	-6	A	

only showing top 3 rows

In [51]: #16. Join the DataFrames
flights_with_airports = flights.join(airports,
on="dest", how="leftouter")

Ex2_SparkDataframe_Flights - Jupyter Notebook



In [52]: # Examine the new DataFrame
 flights_with_airports.show(3)

	dest	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	name	1
	lon	alt	tz	dst									
80	LAX	2014	12	8	658	-7	935	-5	VX	N846VA	17	SEA	132
36					954	61	58	2.2	Los Angeles Intl	33.9425		-118.408075	[126]
51	HNL	2014	1	22	1040	5	1505	5	AS	N559AS	8	SEA	360
81					2677	10	40	6.0	Honolulu Intl	21.3186	-157.922428	[13]	
55	SFO	2014	3	9	1443	-2	1652	2	VX	N847VA	7	SEA	111
72					679	141	43	1.85	San Francisco Intl	37.6189	-122.374889	[13]	

only showing top 3 rows



Chapter 4: Spark DataFrames

Ex3: walmart_stock

There are some basic questions about some stock market data, in this case Walmart Stock from the years 2012-2017.

Use the `walmart_stock.csv` file to Answer and complete the tasks below!

Start a simple Spark Session

In [5]:

Load the Walmart Stock CSV File, have Spark Infer the data types.

In [66]:

What are the column names?

In [67]:

Out[67]: ['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close']

What does the Schema look like?

In [68]:

```
root
|-- Date: timestamp (nullable = true)
|-- Open: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Close: double (nullable = true)
|-- Volume: integer (nullable = true)
|-- Adj Close: double (nullable = true)
```

Print out the first 5 rows.



In [76]:

```
Row(Date=datetime.datetime(2012, 1, 3, 0, 0), Open=59.970001, High=61.060001, Low=59.869999, Close=60.330002, Volume=12668800, Adj Close=52.619234999999996)
Row(Date=datetime.datetime(2012, 1, 4, 0, 0), Open=60.20998999999996, High=60.349998, Low=59.470001, Close=59.70998999999996, Volume=9593300, Adj Close=52.078475)
Row(Date=datetime.datetime(2012, 1, 5, 0, 0), Open=59.349998, High=59.619999, Low=58.369999, Close=59.419998, Volume=12768200, Adj Close=51.825539)
Row(Date=datetime.datetime(2012, 1, 6, 0, 0), Open=59.419998, High=59.450001, Low=58.869999, Close=59.0, Volume=8069400, Adj Close=51.45922)
Row(Date=datetime.datetime(2012, 1, 9, 0, 0), Open=59.029999, High=59.549999, Low=58.919998, Close=59.18, Volume=6679300, Adj Close=51.616215000000004)
```

Use describe() to learn about the DataFrame.

In [77]:

summary	Open	High	Low	Clo
se	Volume	Adj Close		
count	1258	1258	1258	12
mean	72.35785375357709	72.83938807631165	71.9186009594594	72.388449980127
8222093.481717011	67.23883848728146			
stddev	6.76809024470826	6.768186808159218	6.744075756255496	6.7568591637329
4519780.8431556	6.722609449996857			
min	56.38999899999996	57.060001	56.299999	56.4199
98	2094900	50.363689		
max	90.800003	90.970001	89.25	90.4700
01	80898100	84.91421600000001		

There are too many decimal places for mean and stddev in the describe() dataframe. Format the numbers to just show up to two decimal places. Pay careful attention to the datatypes that .describe() returns, we didn't cover how to do this exact formatting, but we covered something very similar. [Check this link for a hint](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Column.cast) (<http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Column.cast>)

In [78]:

```
root
|-- summary: string (nullable = true)
|-- Open: string (nullable = true)
|-- High: string (nullable = true)
|-- Low: string (nullable = true)
|-- Close: string (nullable = true)
|-- Volume: string (nullable = true)
|-- Adj Close: string (nullable = true)
```

In [79]:

In [80]:

	summary	Open	High	Low	Close	Volume
count	1,258.00	1,258.00	1,258.00	1,258.00	1258	8222093
mean	72.36	72.84	71.92	72.39	8222093	
stddev	6.77	6.77	6.74	6.76	4519781	
min	56.39	57.06	56.30	56.42	2094900	
max	90.80	90.97	89.25	90.47	80898100	

Create a new dataframe with a column called HV Ratio that is the ratio of the High Price versus volume of stock traded for a day.



In [81]:

```
+-----+
|      HV Ratio|
+-----+
| 4.819714653321546E-6|
| 6.290848613094555E-6|
| 4.669412994783916E-6|
| 7.367338463826307E-6|
| 8.915604778943901E-6|
| 8.644477436914568E-6|
| 9.351828421515645E-6|
| 8.29141562102703E-6|
| 7.712212102001476E-6|
| 7.071764823529412E-6|
| 1.015495466386981E-5|
| 6.576354146362592...|
| 5.90145296180676E-6|
| 8.547679455011844E-6|
| 8.420709512685392E-6|
| 1.041448341728929...|
| 8.316075414862431E-6|
| 9.721183814992126E-6|
| 8.029436027707578E-6|
| 6.307432259386365E-6|
+-----+
only showing top 20 rows
```

What day had the Peak High in Price?

In [88]:

Out[88]: `datetime.datetime(2015, 1, 13, 0, 0)`**What is the mean of the Close column?**

In [89]:

```
+-----+
|      avg(Close)|
+-----+
| 72.38844998012726|
+-----+
```

What is the max and min of the Volume column?

In [90]:

In [9?]:

max(Volume)	min(Volume)
88898100	2094900

How many days was the Close lower than 60 dollars?

In [10?]:

Out[10?]: 81

What percentage of the time was the High greater than 80 dollars ?

In other words, $(\text{Number of Days High}>80)/(\text{Total Days in the dataset})$

In [107]:

Out[107]: 9.141494435612083

What is the Pearson correlation between High and Volume?

Hint

<http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameStat>

In [110]:

corr(High, Volume)
-0.3384326061737161

What is the max High per year?



3/16/2020

In [117]:

Year	max(High)
2015	90.970001
2013	81.370003
2014	88.089996
2012	77.599998
2016	75.190002

What is the average Close for each Calendar Month?

In other words, across all the years, what is the average Close price for Jan, Feb, Mar, etc... Your result will have a value for each of these months.

In [121]:

Month	avg(Close)
1	71.44801958415842
2	71.306804443299
3	71.77794377570092
4	72.97361900952382
5	72.30971688679247
6	72.4953774245283
7	74.43971943925233
8	73.02981855454546
9	72.18411785294116
10	71.57854545454543
11	72.1110893069307
12	72.84792478301885