Week 3: Spark DataFrames

Overview

- Spark APIs (Review)
- Spark DataFrames
  - Creation
  - Manipulation
  - User-defined functions

Spark APIs

- Three APIs:
  - RDDs (last week)
  - DataSets
  - DataFrames

Spark RDDs:

- Resilient Distributed Datasets
- Low-level
- Support creation, transformations, and actions
- Creation:
  - parallelize() an existing data structure; load a file with textFile()
    - transformations return a new RDD (e.g. map(), reduce())
- Actions return non-RDDs (e.g. count(), collect())

Spark Datasets

- Distributed collection of data
- Only available via Scala and Java (i.e. no Python interface)
- We will not be using Datasets

Spark Dataframes

- Our main focus will be on Spark DataFrames
- DataFrames are Spark Datasets organized into named columns (sound familiar?)
- They're *tables*
- conceptually very similar to Pandas and R DataFrames

Spark Dataframes: Getting Started

- all interaction is via a SparkSession:

```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
     .master("local[*]") \
    .appName("Python Spark SQL basic example")\
    .getOrCreate()
```

- entry point to programming Spark with the DataFrame API
- to create a SparkSession, use the builder pattern shown above
- more or less equivalent to SparkContext

Creating Spark Dataframes

- once you have a SparkSession you can create a DataFrame
- a DataFrame can be created from:
  - a list
  - an RDD
  - a specially-formatted JSON file

Creating A Dataframe From A List

- list of tuples: include a list of column names
- list of values: specify value type

```
# create a DataFrame from a list of tuples
df_from_other_list = spark.createDataFrame(
    [('Chris',67),('Frank',70)], ['name','score'])
df_from_other_list.show()

+-----+-----+
| name|score|
+-----+-----+
|Chris|   67|
|Frank|   70|
+-----+-----+
```

```
# create a DataFrame from a list of values;
# note that you must specify the data type
from pyspark.sql.types import FloatType
df_from_list = spark.createDataFrame(
    [1.0,2.0,3.0,4.0,5.0], FloatType())
df_from_list.show()

+-----+
|value|
+-----+
|  1.0|
|  2.0|
|  3.0|
|  4.0|
|  5.0|
+-----+
```

Displaying and extracting DataFrame contents

- `df.show()`: shows the first 20 entries
- `df.first()`: shows the first entry
- `df.head(n)` or `df.take(n)` [default is 5]
  - shows the first n entries
- `df.collect()`
  - returns a python list of the DataFrame Rows (DANGER)

Creating A Dataframe From An RDD

- simple of you're ok with default column names
- need to create a pyspark.sql.Row if you want better column names

```
# create an RDD
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lot_rdd = sc.parallelize([('Chris',67),('Frank',70)])
```

```
# create a DataFrame from an RDD
dfPeople = spark.createDataFrame(lot_rdd)
dfPeople.show()

# create a Row to include column names
from pyspark.sql import Row

lot_rdd_named_columns = lot_rdd \
    .map(lambda x: Row(name=x[0], score=int(x[1])))
dfPeople_named_columns = spark \
    .createDataFrame(lot_rdd_named_columns)
dfPeople_named_columns.show()
```

```
+-----+---+
|   _1| _2|
+-----+---+
|Chris| 67|
|Frank| 70|
+-----+---+

+-----+-----+
| name|score|
+-----+-----+
|Chris|   67|
|Frank|   70|
+-----+-----+
```

Creating A Dataframe From A JSON File

```
# read a specially formatted JSON file (one JSON object per line)
df = spark.read.json("business.json")
# Displays the content of the DataFrame to stdout
df.show()
```

```
+--------------------+--------------------+--------------------+--------------------+-------------+--------------------+------
-+-------------+--------------+--------------------+--------------------+-----------+------------+-----+-----+
|             address|          attributes|         business_id|          categories|         city|                hours|is_open|     latitude|     longitude|                name|      neighborhood|postal_code|review_count|stars|state|
+--------------------+--------------------+--------------------+--------------------+-------------+--------------------+------
-+-------------+--------------+--------------------+--------------------+-----------+------------+-----+-----+
|4855 E Warner Rd,...|[true,null,null,n...|FYWN1wneV18bWNgQj...|[Dentists, Genera...|     Ahwatukee|[7:30-17:00,7:30-...|
1|   33.3306902|  -111.9785992|    Dental by Design|
```

Inferring Schema

- From previous example, df.printSchema() gives:

```
root
 |-- address: string (nullable = true)
 |-- attributes: struct (nullable = true)
 |    |-- AcceptsInsurance: boolean (nullable = true)
 |    |-- AgesAllowed: string (nullable = true)
 |    |-- Alcohol: string (nullable = true)
 |    |-- Ambience: struct (nullable = true)
 |    |    |-- casual: boolean (nullable = true)
 |    |    |-- classy: boolean (nullable = true)
 |    |    |-- divey: boolean (nullable = true)
 |    |    |-- hipster: boolean (nullable = true)
 |    |    |-- intimate: boolean (nullable = true)
 |    |    |-- romantic: boolean (nullable = true)
 |    |    |-- touristy: boolean (nullable = true)
 |    |    |-- trendy: boolean (nullable = true)
 |    |    |-- upscale: boolean (nullable = true)
 |    |-- BYOB: boolean (nullable = true)
 |    |-- BYOBCorkage: string (nullable = true)
 |    |-- BestNights: struct (nullable = true)
 |    |    |-- friday: boolean (nullable = true)
 |    |    |-- monday: boolean (nullable = true)
 |    |    |-- saturday: boolean (nullable = true)
 |    |    |-- sunday: boolean (nullable = true)
 |    |    |-- thursday: boolean (nullable = true)
 |    |    |-- tuesday: boolean (nullable = true)
 |    |    |-- wednesday: boolean (nullable = true)
 |-- is_open: long (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- name: string (nullable = true)
 |-- neighborhood: string (nullable = true)
 |-- postal_code: string (nullable = true)
 |-- review_count: long (nullable = true)
 |-- stars: double (nullable = true)
 |-- state: string (nullable = true)
```

Creating A Dataframe From A File (General)

- Spark can load a number of different formats: json, parquet, jdbc, orc, libsvm, csv, text

```
df=spark.read.load("foo.json", format="json")
```

Describing a DataFrame`

`df.columns`: show columns names

`df.dtypes`: show data types

`df.describe().show()`: calculate simple statistics

`df.count()`: count number of entries

Column Selection

```
# Select only the "name" column
df.select("name").show()
```

```
+--------------------+
|                name|
+--------------------+
|    Dental by Design|
| Stephen Szabo Salon|
|Western Motor Veh...|
|    Sports Authority|
|Brick House Taver...|
|             Messina|
|          BDJ Realty|
|         Soccer Zone|
|    Any Given Sundae|
|Detailing Gone Mo...|
|   East Coast Coffee|
|CubeSmart Self St...|
|T & T Bakery and ...|
|Complete Dental Care|
|Showmars Governme...|
|      Alize Catering|
|      T & Y Nail Spa|
|Meineke Car Care ...|
|Senior's Barber Shop|
|Maxim Bakery & Re...|
+--------------------+
only showing top 20 rows
```

Creating a new column

```
from pyspark.sql.functions import col
df.withColumn('new',col('old'))
```

- supply name of new column, as well as the data source (possibly another column with some optional transformations)

Deleting a column

```
df.drop('someColumn')
```

Filtering

```
# Select businesses with 4 or more stars
df.filter(df['stars'] >= 4).show()
```

DataFrame Rows

- Abstraction of rows

```
result = df.filter(someCondition).collect()
row = result[0]
row.asDict().values()
for item in result[0]:
    print(item)
for item in result[0].asDict():
    print(item)
for item in result[0].asDict().keys():
    print(item)
```

GroupBy and Sorting

```
# Count businesses by stars
df.groupBy("stars").count().show()
```

```
+-----+-----+
|stars|count|
+-----+-----+
|  3.5|32038|
|  4.5|24796|
|  2.5|16148|
|  1.0| 3788|
|  4.0|33492|
|  3.0|23142|
|  2.0| 9320|
|  1.5| 4303|
|  5.0|27540|
+-----+-----+
```

```
# Count businesses by stars and sort the output
df.groupBy("stars").count().sort("stars",ascending=False).show()
```

```
+-----+-----+
|stars|count|
+-----+-----+
|  5.0|27540|
|  4.5|24796|
|  4.0|33492|
|  3.5|32038|
|  3.0|23142|
|  2.5|16148|
|  2.0| 9320|
|  1.5| 4303|
|  1.0| 3788|
+-----+-----+
```

Creating a new dataframe with a subset of columns

```
df.toDF('column1','column2')
```

Renaming columns

```
df.withColumnRenamed('old','new').show()
```

Explode

- create a row for each value in a list/array/etc.

```
# create a DataFrame from a list of tuples
df_from_other_list2 = spark.createDataFrame(
    [('Chris',[67,42]),('Frank',[70,72])], ['name','scores']
)
df_from_other_list2.show()
```

```
+-----+--------+
| name|  scores|
+-----+--------+
|Chris|[67, 42]|
|Frank|[70, 72]|
+-----+--------+
```

```
from pyspark.sql.functions import explode
```

```
df_exploded = df_from_other_list2.withColumn('score',explode('scores'))
```

```
df_exploded.show()
```

```
+-----+--------+-----+
| name|  scores|score|
+-----+--------+-----+
|Chris|[67, 42]|   67|
|Chris|[67, 42]|   42|
|Frank|[70, 72]|   70|
|Frank|[70, 72]|   72|
+-----+--------+-----+
```

When... Otherwise

```
import pyspark.sql.functions as F
from pyspark.sql.functions import col
df_exploded.withColumn('good', F.when(df_exploded['score']>50,1).otherwise(0)).show()
```

```
+-----+--------+-----+----+
| name|  scores|score|good|
+-----+--------+-----+----+
|Chris|[67, 42]|   67|   1|
|Chris|[67, 42]|   42|   0|
|Frank|[70, 72]|   70|   1|
|Frank|[70, 72]|   72|   1|
+-----+--------+-----+----+
```

User-defined functions (UDFs)

- wrapping python functions
- need to specify output type

User-defined functions (UDFs)

- similar to map and apply in pandas
- wrapping plain old python functions
  - need to specify output type