Distributed Computing

DIANE WOODBRIDGE, PH.D

Announcement

Individual Assignment

- HW 3 - Feb 23rd (Extended)
- HW 4 - March 3rd (Extended)
- Watch Replication/Sharding/CAP - Feb 27th

Group Assignment

- Task 2 (Workflow for data collection, pre-processing and storage) - Feb 25th

Quiz 2

- Feb 28th 9 AM

Contents

Contents

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  1. Scalar functions : Return a single value for each row based on calculations on one or more columns.
  2. Aggregate functions : Return a single value for a group of rows.
  3. Window functions : Return several values for a group of rows.
  4. User-defined functions (UDF) : Generate custom scalar or aggregate functions.
  5. Join : Join two data frames.

https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/sql/functions.html

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  1. Scalar functions : Return a single value for each row based on calculations on one or more columns.
     - Include **`from pyspark.sql.functions import *`**
       - Math – `abs, log, avg, count`, etc.
       - String – `length, concat, trim, contains` ,etc.
       - Time – `year, date_add, datediff, next_day` ,etc.

https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/sql/functions.html

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  2. Aggregate functions : Return a single value for a group of rows.
     - Can be used in combination with `groupBy()`.
       - **`.groupBy()`** - Take column names or a list of column objects.
     - Can use an aggregate function or **`.agg(list_of_aggregate_fuctions).`**
       - **aggregate function - `min(), max(), sum(), count(), etc.`**
     - Format
       - **`.groupBy(col).max(col)`**
       - **`.groupBy(col).agg(max(col), min(col),…)`**
     - Example : `df.groupBy().avg(<column-name>), df.select(avg(df.column-name))`

https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/functions.html
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.groupBy

Example 1

Create a dataframe from filtered_registered_business_sf.csv and find the top 5 zip codes with the most businesses.

Expected Output

```
+-----+-----+
|  zip|count|
+-----+-----+
|94110|12459|
|94103|10919|
|94109| 9623|
|94107| 9394|
|94102| 7962|
+-----+-----+
only showing top 5 rows
```

Example 1

Create a dataframe from filtered_registered_business_sf.csv and find the top 5 zip codes with the most businesses.

```
business_df.groupBy("zip").count().orderBy("count", ascending = False).show(5)
```

```
+-----+-----+
|  zip|count|
+-----+-----+
|94110|12459|
|94103|10919|
|94109| 9623|
|94107| 9394|
|94102| 7962|
+-----+-----+
only showing top 5 rows
```

Example 1 – Extra

Create a column named "onHoward" to see whether it is on Howard street.

Expected Output

```
+-----+--------------------+--------------+-------------+-----+--------+
|  zip|                name|        street|         city|state|OnHoward|
+-----+--------------------+--------------+-------------+-----+--------+
|94105|Stephens Institut...| 631 Howard St|San Francisco|   CA|    true|
|94103|Anderson Enterpri...|1525 Howard St|San Francisco|   CA|    true|
|94103|Avis Rent A Car S...| 821 Howard St|San Francisco|   CA|    true|
|94103|German Motors Cor...|1675 Howard St|San Francisco|   CA|    true|
|94103|German Motors Cor...|1675 Howard St|San Francisco|   CA|    true|
+-----+--------------------+--------------+-------------+-----+--------+
only showing top 5 rows
```

Example 1 – Extra

Create a column named "onHoward" to see whether it is on Howard street.

```
business_df.withColumn('OnHoward', business_df['street'].contains('Howard'))\
           .filter("OnHoward == True")\
           .show(5)
```

```
+-----+--------------------+-------------+-------------+-----+--------+
|  zip|                name|       street|         city|state|OnHoward|
+-----+--------------------+-------------+-------------+-----+--------+
|94105|Stephens Institut...| 631 Howard St|San Francisco|   CA|    true|
|94103|Anderson Enterpri...|1525 Howard St|San Francisco|   CA|    true|
|94103|Avis Rent A Car S...| 821 Howard St|San Francisco|   CA|    true|
|94103|German Motors Cor...|1675 Howard St|San Francisco|   CA|    true|
|94103|German Motors Cor...|1675 Howard St|San Francisco|   CA|    true|
+-----+--------------------+-------------+-------------+-----+--------+
only showing top 5 rows
```

Example 2

Calculate min, max, average num_bike_available per station_id from status_df.

Expected Output

```
+----------+-----------------------+
|station_id|min(num_bikes_available)|
+----------+-----------------------+
|        10|                      0|
|        11|                      0|
+----------+-----------------------+

+----------+-----------------------+
|station_id|avg(num_bikes_available)|
+----------+-----------------------+
|        10|      5.931166125246599|
|        11|      7.768450198057421|
+----------+-----------------------+

+----------+-----------------------+
|station_id|max(num_bikes_available)|
+----------+-----------------------+
|        10|                     15|
|        11|                     19|
+----------+-----------------------+
```

Example 2

Calculate min, max, average num_bike_available per station_id from status_df.

```
status_df.groupBy('station_id').min('num_bikes_available').show()
status_df.groupBy('station_id').avg('num_bikes_available').show()
status_df.groupBy('station_id').max('num_bikes_available').show()
```

```
+----------+-----------------------+
|station_id|min(num_bikes_available)|
+----------+-----------------------+
|        10|                      0|
|        11|                      0|
+----------+-----------------------+

+----------+-----------------------+
|station_id|avg(num_bikes_available)|
+----------+-----------------------+
|        10|       5.931166125246599|
|        11|       7.768450198057421|
+----------+-----------------------+

+----------+-----------------------+
|station_id|max(num_bikes_available)|
+----------+-----------------------+
|        10|                     15|
|        11|                     19|
+----------+-----------------------+
```

Example 2

Calculate min, max, average num_bike_available per station_id from status_df.

```
status_df.groupBy('station_id')\
         .agg(min('num_bikes_available'), avg('num_bikes_available'), max('num_bikes_available'))\
         .show()
```

```
+----------+-------------------------+-------------------------+-------------------------+
|station_id|min(num_bikes_available)|avg(num_bikes_available)|max(num_bikes_available)|
+----------+-------------------------+-------------------------+-------------------------+
|        10|                        0|       5.931166125246599|                       15|
|        11|                        0|       7.768450198057421|                       19|
+----------+-------------------------+-------------------------+-------------------------+
```

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  3. Window functions : Return several values for a group of rows.
     - Unlike aggregate functions, they don't group rows into a single output per group.
     - Need to add `from pyspark.sql.window import Window`
       - **Format**
       - `Window.partitionBy("col_name_1").orderBy("col_name_2").rowsBetween(start, end)`
       - `Window.partitionBy("col_name_1").orderBy("col_name_2").rangeBetween(start, end)`
         - 0 : current row
         - -n : n row before the current row
         - n : n row after the current row
           - Limitation of `rangeBetween()` - doesn't work with non-ordered windows. There can be only one expression and this expression must have a numerical data type.

https://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Window.html#pyspark.sql.Window

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  3. Window functions
     - **Steps**
       1. Construct a column definition with an aggregate function or window functions.
       2. Build a window specification and use it as an argument to over() function.
          1) Define the partition using partitionBy() function.
          2) Specify ordering in the partition using orderBy().

Additional Window Functions (+ aggregate functions)

Include **`from pyspark.sql.functions import *`**

Function Name	Description
lag(col, offset)	Returns the value that is offset rows before the current row
lead(col, offset)	Returns the value that is offset rows after the current row
row_number()	Returns a sequential number starting at 1 within a window partition.
rank()	Returns the rank of rows within a window partition.
percent_rank()	Returns the relative rank

```
Ex. count('zip').over(Window.partitionBy('state').orderBy('zip').rowsBetween(-1,1))
```

https://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html

Example 3

Create a dataframe using bike_share/status_million.csv

Return the current and previous number of bike at station_id = 10 at each time data was collected order by timestamp.

Expected Output

station_id	timestamp	num_bikes_available	prev_num_bikes_available
10	2014-09-01 00:00:03	9	null
10	2014-09-01 00:01:02	9	9
10	2014-09-01 00:02:02	9	9
10	2014-09-01 00:03:03	9	9
10	2014-09-01 00:04:02	9	9
10	2014-09-01 00:05:02	9	9
10	2014-09-01 00:06:02	9	9
10	2014-09-01 00:07:02	9	9
10	2014-09-01 00:08:02	9	9
10	2014-09-01 00:09:03	9	9

Example 3

Create a dataframe using bike_share/status_million.csv

Return the current and previous number of bike at station_id = 10 at each time data was collected order by timestamp.

```
status_df.filter("station_id == 10")\
         .select('station_id', 'timestamp', 'num_bikes_available',\
                 lag('num_bikes_available',1).over(Window.partitionBy('station_id').orderBy('timestamp'))\
                                                                                    )\
         .show(
```

Construct a column definition with an aggregate function or window functions.

```
+----------+-------------------+------------------+-----------------------+
|station_id|          timestamp|num_bikes_available|prev_num_bikes_available|
+----------+-------------------+------------------+-----------------------+
|        10|2014-09-01 00:00:03|                 9|                   null|
|        10|2014-09-01 00:01:02|                 9|                      9|
|        10|2014-09-01 00:02:02|                 9|                      9|
|        10|2014-09-01 00:03:03|                 9|                      9|
|        10|2014-09-01 00:04:02|                 9|                      9|
|        10|2014-09-01 00:05:02|                 9|                      9|
|        10|2014-09-01 00:06:02|                 9|                      9|
|        10|2014-09-01 00:07:02|                 9|                      9|
```

Example 3

Create a dataframe using bike_share/status_million.csv

Return the current and previous number of bike at station_id = 10 at each time data was collected order by timestamp.

```
status_df.filter("station_id == 10")\
         .select('station_id', 'timestamp', 'num_bikes_available',\
                 lag('num_bikes_available',1).over(Window.partitionBy('station_id').orderBy('timestamp'))\
                 .alias("prev_num_bikes_available"))\
         .show()
```

Build a window specification and use it as an argument to over() function.
1)Define the partition using partitionBy() function.
2)Specify ordering in the partition using orderBy().

```
+----------+-------------------+-------------------+-----------------------+
|station_id|          timestamp|num_bikes_available|prev_num_bikes_available|
+----------+-------------------+-------------------+-----------------------+
|        10|2014-09-01 00:00:03|                  9|                   null|
|        10|2014-09-01 00:01:02|                  9|                      9|
|        10|2014-09-01 00:02:02|                  9|                      9|
|        10|2014-09-01 00:03:03|                  9|                      9|
|        10|2014-09-01 00:04:02|                  9|                      9|
|        10|2014-09-01 00:05:02|                  9|                      9|
|        10|2014-09-01 00:06:02|                  9|                      9|
|        10|2014-09-01 00:07:02|                  9|                      9|
```

Example 3

Create a dataframe using bike_share/status_million.csv

Return the current and previous number of bike at station_id = 10 at each time data was collected order by timestamp.

```
status_df.filter("station_id == 10")\
         .select('station_id', 'timestamp', 'num_bikes_available',\
                 lag('num_bikes_available',1).over(Window.partitionBy('station_id').orderBy('timestamp'))\
                 .alias("prev_num_bikes_available"))\
         .show()
```

```
from pyspark.sql.window import Window

lag('num_bikes_available',1).over(Window.partitionBy('station_id').orderBy('timestamp'))
```

Example 3 - Extra

Return the current and next number of bike at station_id = 10 at each time data was collected order by timestamp.

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  4. User-defined functions (UDF) : Generate custom scalar or aggregate functions.
     - `udf((f=None, returnType=StringType)` lets you extend the built-in functionalities.
     - Format

```
from pyspark.sql.functions import *

udf_name = udf(lambda function definition or function name, returnType)

data_frame.select(udf_name(col))
```

https://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/udf.html
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.groupBy

Example 4

For business_df created from filtered_registered_business_sf.csv,

- Create a UDF called **check_sf** which checks whether a given value contains “San Francisco” or “SF”.
- Apply **check_sf** to ***business_df*** to check whether city is San Francisco.

Expected Output

```
+--------------------+---------+
|                name|sf(city)|
+--------------------+---------+
|   Tournahu George L|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
|Stephens Institut...|     true|
```

Example 4

For business_df created from filtered_registered_business_sf.csv,

- Create a UDF called **check_sf** which checks whether a given value contains “San Francisco” or “SF”.
- Apply **check_sf** to ***business_df*** to check whether city is San Francisco.

```
from pyspark.sql.functions import *
check_sf = udf(lambda x : ("San Francisco" in x) or ("SF" in x))
```

```
business_df.select('name', check_sf('city')).show()
```

```
+--------------------+--------------+
|                name|<lambda>(city)|
+--------------------+--------------+
|   Tournahu George L|          true|
|Stephens Institut...|          true|
|Stephens Institut...|          true|
|Stephens Institut...|          true|
|Stephens Institut...|          true|
```

Example 4

For business_df created from filtered_registered_business_sf.csv,

- Create a UDF called **check_sf** which checks whether a given value contains “San Francisco” or “SF”.
- Apply **check_sf** to ***business_df*** to check whether city is San Francisco.

```
business_df.select('name', check_sf('city')).printSchema()
```

executed in 26ms, finished 15:33:51 2021-01-31

```
root
 |-- name: string (nullable = false)
 |-- <lambda>(city): string (nullable = true)
```

Example 4

For business_df created from filtered_registered_business_sf.csv,

- Create a UDF called **check_sf** which checks whether a given value contains “San Francisco” or “SF”.
- Apply **check_sf** to ***business_df*** to check whether city is San Francisco.

```
from pyspark.sql.functions import *

check_sf_using_sf = udf(sf, BooleanType())
business_df.select('name', check_sf_using_sf('city')).show()
```

executed in 470ms, finished 15:34:01 2021-01-31

```
+--------------------+--------+
|                name|sf(city)|
+--------------------+--------+
|   Tournahu George L|    true|
|Stephens Institut...|    true|
|Stephens Institut...|    true|
|Stephens Institut...|    true|
|Stephens Institut...|    true|
|Stephens Institut...|    true|
```

Example 4

For business_df created from filtered_registered_business_sf.csv,

- Create a UDF called **check_sf** which checks whether a given value contains “San Francisco” or “SF”.
- Apply **check_sf** to ***business_df*** to check whether city is San Francisco.

```
business_df.select('name', check_sf_using_sf('city')).printSchema()
```

executed in 19ms, finished 15:34:09 2021-01-31

```
root
 |-- name: string (nullable = false)
 |-- sf(city): boolean (nullable = true)
```

SQL functions with DataFrame

Use SQL functions for calculation.

- Spark SQL supports most of SQL functions.
  5. Join
     - `.join(dataframe, condition, join_type)`
       - Join types : *`inner (default)`*, *`outer`*, *`left_outer`*, *`right_outer`*, *`leftsemi`*
         - *`outer`* : return unmatched rows from both tables
         - *`leftsemi`* : get the names within left table that appear in right table.
       - Format
         - `df1.join(df2, condition or column name, join_type)`

Like inner joins but only shows left table

See ex05.ipynb

https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.join

Try Example 5

What is the output of df.join(df2, 'name', 'leftsemi').select(df.name, df2.height).show()? Why it doesn't work?

Example 6

Apply inner, left_outer, right_outer and leftsemi join on business and supervisor dataframes.

Example 6

Apply inner, left_outer, right_outer and leftsemi join on business and supervisor dataframes.

```
business_df.join(supervisor_df, 'zip', 'inner').show(5)
```

```
[Stage 10:>                                                         (0 + 2) / 2]
+-----+--------------------+--------------------+-------------+-----+---+
|  zip|                name|              street|         city|state| id|
+-----+--------------------+--------------------+-------------+-----+---+
|94109|Stephens Institut...|1835-49 Van Ness Ave|San Francisco|   CA|  2|
|94109|Stephens Institut...|1835-49 Van Ness Ave|San Francisco|   CA|  6|
|94109|Stephens Institut...|1835-49 Van Ness Ave|San Francisco|   CA|  3|
|94109|Stephens Institut...|1835-49 Van Ness Ave|San Francisco|   CA|  5|
|94109|Stephens Institut...|        1055 Pine St|San Francisco|   CA|  2|
+-----+--------------------+--------------------+-------------+-----+---+
only showing top 5 rows
```

Example 6

Apply inner, left_outer, right_outer and leftsemi join on business and supervisor dataframes.

```
business_df.join(supervisor_df, 'zip', 'left_outer').show(5)
```

```
+-----+--------------------+--------------------+------------+-----+----+
|  zip|                name|              street|        city|state|  id|
+-----+--------------------+--------------------+------------+-----+----+
| 6002|      Integralis Inc|310 West Newberry Rd|   Bloomfield|   CT|null|
|94105|Stephens Institut...|180 New Montgomer...|San Francisco|   CA|   6|
|94105|Stephens Institut...|180 New Montgomer...|San Francisco|   CA|   3|
|94105| Barney & Barney Llc|1 Market St Steua...|San Francisco|   CA|   6|
|94105| Barney & Barney Llc|1 Market St Steua...|San Francisco|   CA|   3|
+-----+--------------------+--------------------+------------+-----+----+
only showing top 5 rows
```

Example 6

Apply inner, left_outer, right_outer and leftsemi join on business and supervisor dataframes.

```
business_df.join(supervisor_df, 'zip', 'leftsemi').show(5)
```

```
[Stage 25:>                                                         (0 + 2) / 2]
+-----+--------------------+--------------------+-------------+-----+
|  zip|                name|              street|         city|state|
+-----+--------------------+--------------------+-------------+-----+
|94109|Stephens Institut...|1835-49 Van Ness Ave|San Francisco|   CA|
|94109|Stephens Institut...|        1055 Pine St|San Francisco|   CA|
|94109|     Alioto F Co Inc|    440 Jefferson St|San Francisco|   CA|
|94109|     Haines Robert D|   786-792 Sutter St|San Francisco|   CA|
|94109|Avis Rent A Car S...|         675 Post St|San Francisco|   CA|
+-----+--------------------+--------------------+-------------+-----+
only showing top 5 rows
```

Contents

Registering DataFrame in the Table Catalog and Using SQL Commands

Registering DataFrame in the Table Catalog

- You can reference a DataFrame by its name by registering the DataFrame as a table.
  - Spark stores the table definition in the table catalog.
  - Save as table
    - Format : `.write.saveAsTable(`*name*, *format=None*, *mode=None*, *partitionBy=None)*
    - Once DataFrame is registered as a table, you can query its data using SQL expressions.
      - Using **`ss.sql(sql_query).`**
      - Also can still use spark-sql.
      - Spark will write data to a default table path. When the table is dropped, the default table path will be removed too.
  - Register a table **permanently using path (in a parquet format)**.
    - `Format : .write.option("path", "/some/path").saveAsTable(name)`
    - The table definitions will survive your application's restarts and are persistent.
      - ss.sql("select * from parquet.`/some/path`")

Parquet :
- Column-oriented data storage format available to any project in the **Hadoop** ecosystem.
- Efficient compression and encoding scheme.

https://spark.apache.org/docs/latest/sql-data-sources-load-save-functions.html
http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter.saveAsTable

Example 7

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

And find supervisor id for "Holbert Deneice M"

Expected Output

```
+---+
| id|
+---+
|  5|
|  3|
|  6|
|  2|
+---+
```

Example 7

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

And find supervisor id for "Holbert Deneice M"

```
business_df.write.saveAsTable('Business')
```

```
supervisor_df.write.saveAsTable('Supervisor')
```

Example 7

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

And find supervisor id for "Holbert Deneice M"

```
ss.sql("select id from Business JOIN Supervisor ON Business.zip = Supervisor.zip where name = 'Holbert Deneice M'")\
  .show()
```

Example 8

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

Kill the session and re-read from the saved files.

Expected Output

zip	name	street	city	state
94105	Barney & Barney Llc	1 Market St Steua...	San Francisco	CA
94109	Holbert Deneice M	1426 California St	San Francisco	CA
6002	Integralis Inc	310 West Newberry Rd	Bloomfield	CT
95603	Mcadams Pat G	10279 Mt Vernon Rd	Auburn	CA
95685	Young Gregory You...	14508 Shake Ridge Rd	Sutter+creek	CA
95037	The Backflow Guy Inc	305 Vineyard Town...	Morgan+hill	CA
94103	W J Britton & Co	1345 Mission St	San Francisco	CA
94112	W J Britton & Co	1000 Geneva Ave	San Francisco	CA
94103	W J Britton & Co	1618 Howard St	San Francisco	CA
94114	W J Britton & Co	2378 Market St	San Francisco	CA
94110	W J Britton & Co	3351 Cesar Chavez St	San Francisco	CA
94112	W J Britton & Co	911 Paris St	San Francisco	CA
94103	W J Britton & Co	30 Guerrero St	San Francisco	CA
94103	W J Britton & Co	1927 Market St	San Francisco	CA
94112	W J Britton & Co	5124 Mission St	San Francisco	CA
94103	W J Britton & Co	123 Van Ness Ave	San Francisco	CA
94112	W J Britton & Co	888 Geneva Ave	San Francisco	CA
94110	W J Britton & Co	1218 Valencia St	San Francisco	CA
94103	W J Britton & Co	131 Van Ness Ave	San Francisco	CA
94112	W J Britton & Co	990 Geneva Ave	San Francisco	CA

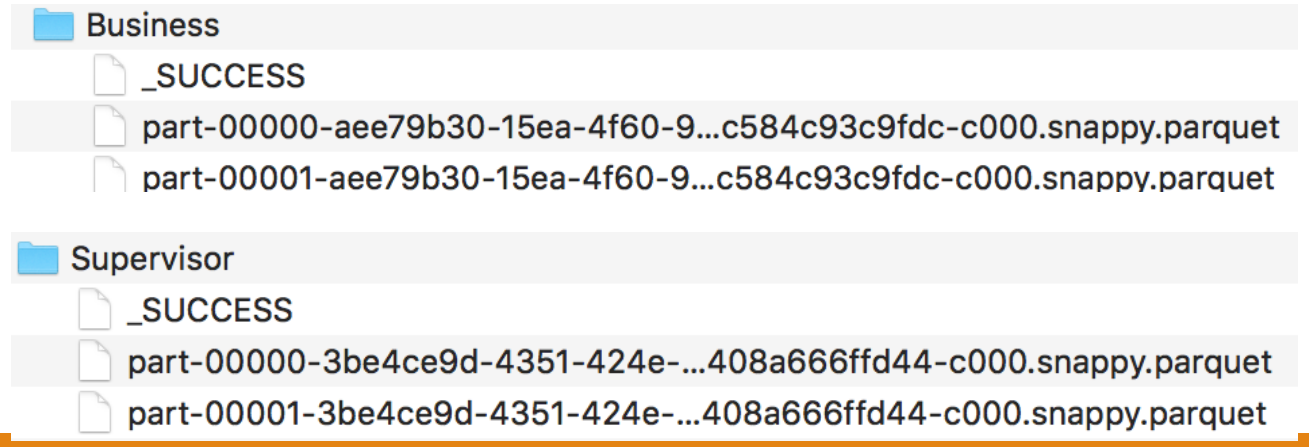
only showing top 20 rows

Example 8

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

```
business_df.write.option(
    "path", "/Users/dwoodbridge/Class/2019_MSDS697/Example/Day2/Business").saveAsTable('Business')
```

```
supervisor_df.write.option(
    "path", "/Users/dwoodbridge/Class/2019_MSDS697/Example/Day2/Supervisor").saveAsTable('Supervisor')
```



Parquet :

- Column-oriented data storage format oavailable to any project in the **Hadoop** ecosystem
- Efficient compression and encoding scheme.

https://parquet.apache.org/documentation/latest/

Example 8

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”.

Kill the session and re-read from the saved files.

```
sc.stop()
```

```
ss.sql("select * from parquet.`/Users/dwoodbridge/Class/2019_MSDS697/Example/Day2/Business`").show()
```

```
+-----+--------------------+--------------------+-------------+-----+
|  zip|                name|              street|         city|state|
+-----+--------------------+--------------------+-------------+-----+
|94105| Barney & Barney Llc|1 Market St Steua...|San Francisco|   CA|
|94109|   Holbert Deneice M|  1426 California St|San Francisco|   CA|
| 6002|      Integralis Inc|310 West Newberry Rd|   Bloomfield|   CT|
|95603|       Mcadams Pat G|  10279 Mt Vernon Rd|       Auburn|   CA|
|95685|Young Gregory You...|14508 Shake Ridge Rd| Sutter+creek|   CA|
|95037|The Backflow Guy Inc|305 Vineyard Town...|  Morgan+hill|   CA|
|94103|    W J Britton & Co|     1345 Mission St|San Francisco|   CA|
|94112|    W J Britton & Co|     1000 Geneva Ave|San Francisco|   CA|
|94103|    W J Britton & Co|      1618 Howard St|San Francisco|   CA|
|94114|    W J Britton & Co|      2378 Market St|San Francisco|   CA|
|94110|    W J Britton & Co|3351 Cesar Chavez St|San Francisco|   CA|
|94112|    W J Britton & Co|        911 Paris St|San Francisco|   CA|
```

Example 8

Save Supservisor DataFrame as “Supervisor” and Business DataFrame as “Business”. Kill the session and re-read from the saved files.

```
ss.sql("select * from parquet.`/Users/dwoodbridge/Class/2019_MSDS697/Example/Day2/Supervisor`").show()
```

```
+-----+---+
|  zip| id|
+-----+---+
|94115|  5|
|94116|  7|
|94116|  4|
|94117|  1|
|94117|  7|
|94117|  8|
|94117|  5|
|94118|  2|
|94118|  1|
|94118|  5|
|94121|  2|
|94121|  1|
|94122|  1|
|94122|  7|
|94122|  5|
```

Contents

Loading/Writing Data using SparkSQL

Datatypes

- JSON
- CSV
- Parquet

Data Sources

- GCS
- Relational Databases and Other DB (MongoDB) with JDBC

Covered in Task2

Will cover in Task3

Loading/Writing Data using SparkSQL

Datatypes

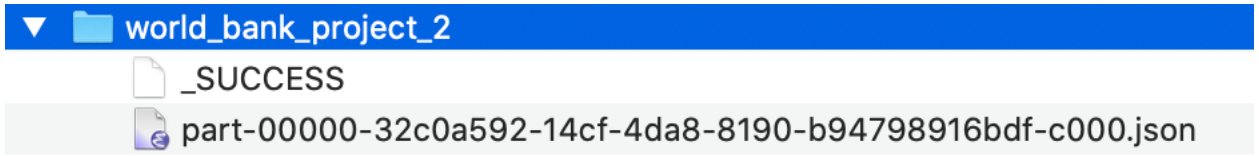
- JSON
  - Read
    - Spark can automatically infer a JSON schema.
    - Use `ss.read.json(<file_name>, etc).`
  - Write : Use `df.write.json(<file_path>, etc).`
- CSV
  - Read
    - Spark can automatically infer a CSV schema if inferSchmea is True.
    - Use `ss.read.csv(<file_name>, etc.).`
  - Write : Use `df.write.csv(<file_path>, etc).`
- Parquet
  - Read
    - Loads Parquet files, returning the result as a DataFrame.
    - Use `ss.read.parquet(path)`
  - Write : Use `df.write.parquet(<file_path>, etc).`

https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameReader.json
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter.json
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameReader.csv
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter.csv
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter.parquet
https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter.parquet

Example 9

Read world_bank_project.json as a DataFrame and write back as json in world_bank_project_2.

Expected Output



https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=read%20csv#pyspark.sql.DataFrameReader.json

Example 9

Read world_bank_project.json as a DataFrame and write back as json in world_bank_project_2.

```
from pyspark.sql import SparkSession
ss = SparkSession.builder.getOrCreate()
```

```
world_bank_prj = ss.read.json("../Data/world_bank_project/world_bank_project.json")
```

Example 9

Read world_bank_project.json as a DataFrame and write back as json in world_bank_project_2.

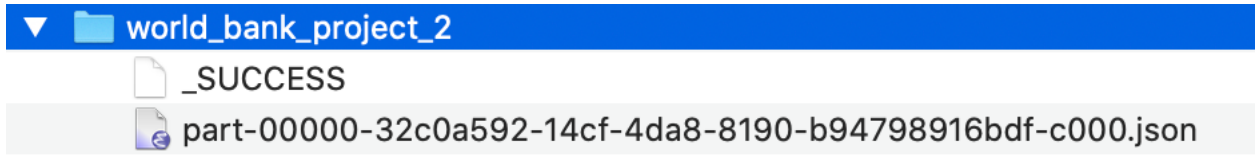
```
world_bank_prj.printSchema()
```

```
root
 |-- _id: struct (nullable = true)
 |    |-- $oid: string (nullable = true)
 |-- approvalfy: string (nullable = true)
 |-- board_approval_month: string (nullable = true)
 |-- boardapprovaldate: string (nullable = true)
 |-- borrower: string (nullable = true)
 |-- closingdate: string (nullable = true)
 |-- country_namecode: string (nullable = true)
 |-- countrycode: string (nullable = true)
 |-- countryname: string (nullable = true)
 |-- countryshortname: string (nullable = true)
 |-- docty: string (nullable = true)
 |-- envassesmentcategorycode: string (nullable = true)
 |-- grantamt: long (nullable = true)
 |-- ibrdcommamt: long (nullable = true)
 |-- id: string (nullable = true)
 |-- idacommamt: long (nullable = true)
```

Example 9

Read world_bank_project.json as a DataFrame and write back as json in world_bank_project_2.

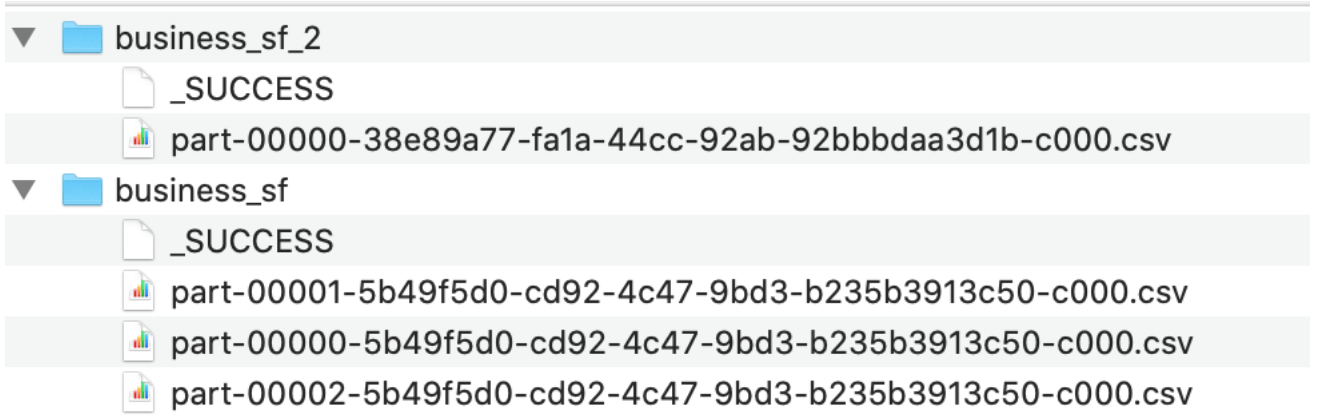
```
world_bank_prj.write.json("world_bank_project_2")
```



Example 10

Read filtered_registered_business_sf.csv with infering schema and write as a csv in business_sf.

Expected Output



https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=read%20csv#pyspark.sql.DataFrameReader.csv

Example 10

Read filtered_registered_business_sf.csv with infering schema and write as a csv in business_sf.

```
business_df = ss.read.csv("../Data/filtered_registered_business_sf.csv", inferSchema = True)
```

```
business_df.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
```

```
business_df.show(5)
```

```
+-----+--------------------+--------------------+-------------+---+
|  _c0|                 _c1|                 _c2|          _c3|_c4|
+-----+--------------------+--------------------+-------------+---+
|94123|   Tournahu George L|   3301 Broderick St|San Francisco| CA|
|94124|Stephens Institut...|    2225 Jerrold Ave|San Francisco| CA|
|94105|Stephens Institut...|180 New Montgomer...|San Francisco| CA|
|94108|Stephens Institut...|       540 Powell St|San Francisco| CA|
```

Example 10

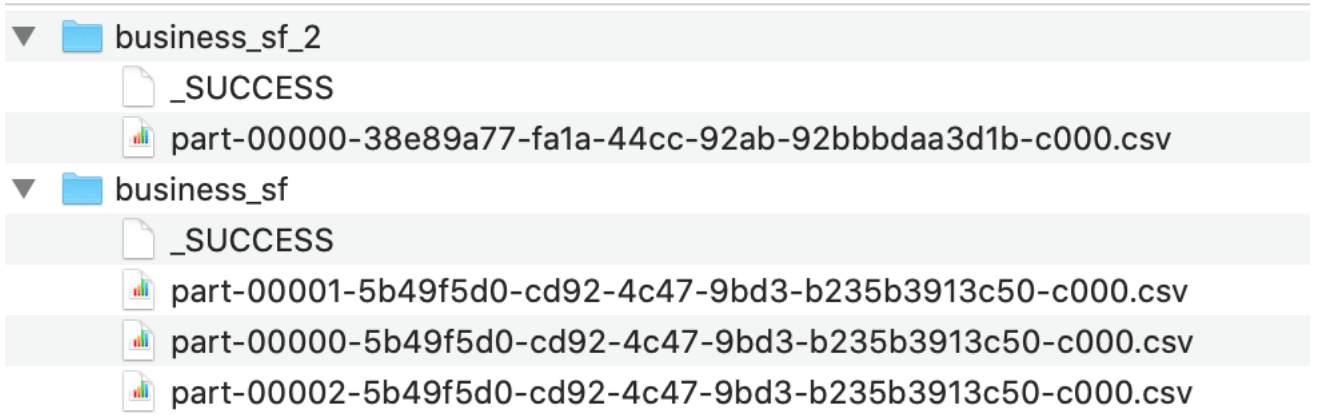
Read filtered_registered_business_sf.csv with infering schema and write as a csv in business_sf.

```
business_df.write.csv("business_sf")
```

executed in 557ms, finished 16:15:47 2021-01-31

```
business_df.coalesce(1).write.csv("business_sf_2")
```

executed in 631ms, finished 16:16:15 2021-01-31



Example 11

Read Business and Supervisor parquet files generated from Example 8.

Expected Output

```
+-----+---+
|  zip| id|
+-----+---+
|94115|  5|
|94116|  7|
|94116|  4|
|94117|  1|
|94117|  7|
|94117|  8|
|94117|  5|
|94118|  2|
|94118|  1|
|94118|  5|
|94121|  2|
|94121|  1|
|94122|  1|
|94122|  7|
```

Example 11

Read Business and Supervisor parquet files generated from Example 8.

```
business_df = ss.read.parquet("Business")
```

```
supervisor_df = ss.read.parquet("Supervisor")
```

Contents

Reference

Spark Online Documentation : http://spark.apache.org/docs/latest/

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

Zecevic, Petar, et al. Spark in Action, Manning, 2016.

Apache Parquet Online Documentation : https://parquet.apache.org/documentation/latest/