Spark DataFrame

DSCI 551

Wensheng Wu

Create & display dataframes

- country = spark.read.json('country.json') # also supports read.csv(…)
- city = spark.read.json('city.json')
- cl = spark.read.json('countrylanguage.json')

- country.show() # show top 20 rows as a table
  - Similar to country.head() in Pandas
- country.show(5)

- Also has head(5)/take(5), tail(5), collect()
  - return an array of Row(…)'s instead
  - Row object works like a dictionary
- also has first()
  - return a single Row(…)

Creating data frames

- df = spark.createDataFrame([('Tom', 80), ('Alice', None)], ["name", "height"])

Projection

- Selecting a subset of columns

- country[['Continent', 'Region']] # similar to Pandas

- Alternative:
  - country[[country.Continent, country.Region]]
  - country.select('Continent', 'Region') # this returns a dataframe
  - country.select(country.LifeExpectancy.alias('le')) # with renaming
  - country['Continent', 'Region'] # this returns a dataframe
  - country['Continent'] # this returns a column

Derived data in projection

- country.select('name', (country.GNP - country.GNPOld).alias('diff')).show(5)

```
+-----------+-------+
|       name|   diff|
+-----------+-------+
|      Aruba|   35.0|
|Afghanistan| 5976.0|
|     Angola|-1336.0|
|   Anguilla|   63.2|
|    Albania|  705.0|
+-----------+-------+
```

Renaming columns

- country.withColumnRenamed('Name', 'CountryName').show()

Adding a new column => a new dataframe

- import pyspark.sql.functions as fc
- df = spark.createDataFrame([(1,1), (1,2), (1,3)], ['a', 'b'])
- >>> df.show()
  ```
  +---+---+
  |  a|  b|
  +---+---+
  |  1|  1|
  |  1|  2|
  |  1|  3|
  +---+---+
  ```
- df1 = df.withColumn('c', fc.col('b'))
- df1.show()
-
  ```
  +---+---+---+
  |  a|  b|  c|
  +---+---+---+
  |  1|  1|  1|
  |  1|  2|  2|
  |  1|  3|  3|
  +---+---+---+
  ```

More examples

- df.withColumn('c', fc.when(df.b > 2, 1).otherwise(-1)).show()

```
+---+---+---+
|  a|  b|  c|
+---+---+---+
|  1|  1| -1|
|  1|  2| -1|
|  1|  3|  1|
+---+---+---+
```

- df.withColumn('c', fc.lit(1)).show()

```
+---+---+---+
|  a|  b|  c|
+---+---+---+
|  1|  1|  1|
|  1|  2|  1|
|  1|  3|  1|
+---+---+---+
```

Selection/filtering

- Selecting a subset of rows

- country[country.GNP > 10000] # similar to Pandas

- Alternative:
  - country.filter(country.GNP > 10000)
  - country.filter("GNP > 10000")
  - country.filter('GNP > 10000 and GNP < 50000') # filter takes SQL style where-condition
  - country.where('GNP > 10000')

- country[(country.GNP > 10000) & (country.GNP < 50000)] # similar to Pandas
  - Other logical operators: |, ~

Distinct

- country[['Continent', 'Region']].distinct()

- Alternative:
  - country[['Continent', 'Region']].dropDuplicates()
    - Similar to drop_duplicates()/unique() in Pandas

Groupby without aggregation

- country.groupBy('Continent') or country.groupby('Continent')
  - Similar to groupby in Pandas

- Need to aggregate so that we can show the grouping details
  - country.groupBy('Continent').count()[['Continent']].show()

- May also use:
  - country[['Continent']].distinct()

Aggregation w/o groupby

- country.agg({'GNP': 'max'})

- import pyspark.sql.functions as fc
- country.agg(fc.max('GNP').alias('max_gnp'))
  - Similar to agg(max_gnp = pd.NamedAgg('GNP', 'max')) in Pandas

- country.agg(fc.max('GNP').alias('max_gnp'), fc.min('GNP').alias('min_gnp')).show()

```
+---------+-------+
|  max_gnp|min_gnp|
+---------+-------+
|8510700.0|    0.0|
+---------+-------+
```

Group by with aggregation

- import pyspark.sql.functions as fc
- country.groupBy('Continent').agg(fc.max("GNP").alias("max_gnp"),\
                    fc.count("*").alias("cnt")).show()

=>

Select Continent, max(GNP) max_gnp, count(*) cnt

From country

Group by Continent

Group by with having

- import pyspark.sql.functions as fc
- country.groupBy('Continent').agg(fc.max("GNP").alias("max_gnp"),
            fc.count("*").alias("cnt")).filter('cnt > 5').show()

=>

select Continent, max(GNP) max_gnp, count(*) cnt

from country

group by Continent

having cnt > 5

Counting w/o group by

- country.count()
  - Note different from Pandas count()

=>

select count(*)

from country

Aggregating one or more columns

- country.groupBy('Continent').max('GNP').show()

- country.groupBy(['Continent', 'Region']).max('GNP').show()

Order by

- import pyspark.sql.functions as fc

- country.orderBy('Continent')

- country.orderBy(fc.desc('Continent'))
  - Alternative: country.orderBy(country.Continent.desc())

- country.orderBy(['Continent', 'GNP'], ascending=[True, False])

- Alternatives: replacing orderBy with sort
  - country.sort(fc.desc('Continent'), fc.desc('GNP'))
  - country.sort(['Continent', 'GNP'], ascending=[True, False])

Aggregation function

- count
- max,min
- avg/mean
- sum

Example: putting them together

- country[(country.GNP > 1000) & (country.GNP < 10000)].groupBy('Continent', 'Region').agg(fc.mean('LifeExpectancy').alias('avg_le'), fc.count('*').alias('cnt')).filter('cnt > 5').orderBy(fc.desc('avg_le')).show()

=> select Continent, avg(LifeExpecancy) avg_le, count(*) cnt

from country

group by Continent

having cnt > 5

order by avg_le desc

Limit

- res = country[(country.GNP > 1000) & (country.GNP < 10000)].groupBy('Continent', 'Region').agg(fc.mean('LifeExpectancy').alias('avg_le'), fc.count('*').alias('cnt'))
- res[res.cnt > 5].orderBy(fc.desc('avg_le')).limit(2)

=>

select Continent, Region, avg(LifeExpectancy) avg_le, count(*) cnt

from country

group by Continent, Region

having cnt > 5

order by avg_le desc

limit 2

Join

- country.join(city, country.Capital == city.ID)

- country.join(city, (country.Capital == city.ID) & (country.Population > city.Population))

- Alternative:
  - country.join(city, [country.Capital == city.ID, country.Population > city.Population])

Natural join

- cl.join(city, 'CountryCode')
  - Equivalent to: cl.join(city, cl.CountryCode == city.CountryCode)

⇒select *

   from countrylanguage natural join city

Outer join

- country.join(city, country.Capital == city.ID, how='left').filter("ID is null")
  - Or filter(city.ID.isNull())

=>

select *

from country left outer join city on country.Capital = city.ID

where city.ID is null

Union

- usa = cl[(cl.CountryCode == 'USA')][['Language', "IsOfficial"]]
- can = cl[(cl.CountryCode == 'CAN')][['Language', "IsOfficial"]]
- usa_can = cl[(cl.CountryCode == 'USA') | (cl.CountryCode == 'CAN')][['Language', "IsOfficial"]]

- Bag union
  - usa.union(can) or usa.unionAll(can)
- Set union
  - usa.union(can).distinct()

```
>>> usa.orderBy('Language').show()       >>> can.orderBy('Language').show()
+----------+----------+                  +----------------+----------+
|  Language|IsOfficial|                  |        Language|IsOfficial|
+----------+----------+                  +----------------+----------+
|   Chinese|         F|                  |         Chinese|         F|
|   English|         T|                  |           Dutch|         F|
|    French|         F|                  |         English|         T|
|    German|         F|                  |Eskimo Languages|         F|
|   Italian|         F|                  |          French|         T|
|  Japanese|         F|                  |          German|         F|
|    Korean|         F|                  |         Italian|         F|
|    Polish|         F|                  |          Polish|         F|
|Portuguese|         F|                  |      Portuguese|         F|
|   Spanish|         F|                  |         Punjabi|         F|
|   Tagalog|         F|                  |         Spanish|         F|
|Vietnamese|         F|                  |       Ukrainian|         F|
+----------+----------+                  +----------------+----------+
```

Intersection

- Bag intersection:
  - usa_can.intersectAll(usa_can)

- Set intersection:
  - usa_can.intersect(usa_can)
  - Note it removes duplicates

```
+----------------+----------+
|        Language|IsOfficial|
+----------------+----------+
|         Chinese|         F|
|         Chinese|         F|
|           Dutch|         F|
|         English|         T|
|         English|         T|
|Eskimo Languages|         F|
|          French|         F|
|          French|         T|
|          German|         F|
|          German|         F|
|          Italian|        F|
|          Italian|        F|
...
```

```
+----------------+----------+
|        Language|IsOfficial|
+----------------+----------+
|         Chinese|         F|
|           Dutch|         F|
|         English|         T|
|Eskimo Languages|         F|
|          French|         F|
|          French|         T|
|          German|         F|
|          Italian|        F|
...
```

Subtract (set semantics)

```
|         French|        F|
|         French|        T|
|         French|        T|
|         French|        F|
```

- uc = usa_can.union(usa_can)
- uc.orderBy('Language').show()

- uc.subtract(can).show()

```
+---------+----------+
| Language|IsOfficial|
+---------+----------+
| Japanese|         F|
|  Tagalog|         F|
|   French|         F|
|Vietnamese|        F|
|   Korean|         F|
+---------+----------+
```

```
+------------------+----------+
|          Language|IsOfficial|
+------------------+----------+
|           Chinese|         F|
|             Dutch|         F|
|           English|         T|
| Eskimo Languages|         F|
|            French|         T|
|            German|         F|
|           Italian|         F|
|            Polish|         F|
|        Portuguese|         F|
|           Punjabi|         F|
|           Spanish|         F|
|          Ukrainian|        F|
+------------------+----------+
```

CAN

Accessing Spark from Python

- Interactive shell:
  - bin/pyspark
  - A SparkSession object spark will be automatically created

- bin/pyspark --master local[4]
  - This starts Spark on local host with 4 threads
  - "--master" specifies the location of Spark master node

Accessing Spark from Python

- Standalone program
  - Executed using spark-submit script
  - E.g., bin/spark-submit <your Python Spark script>

- You may find many Python Spark examples under
  - examples/src/main/python

Dataframe to RDD and back

- df.rdd.toDF()

- df.rdd.map(lambda r: (r['a'], r['b'], 1)).toDF(['a', 'b', 'c']).show()

```
+---+---+---+
|  a|  b|  c|
+---+---+---+
|  1|  1|  1|
|  1|  2|  1|
|  1|  3|  1|
+---+---+---+
```

Spark SQL

- country.createOrReplaceTempView('country')

- spark.sql('select Name from country limit 5').show()

```
+-----------+
|       Name|
+-----------+
|      Aruba|
|Afghanistan|
|     Angola|
|   Anguilla|
|    Albania|
+-----------+
```

Resources

- Important classes of Spark SQL and DataFrames
  - https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

- read.csv(...), read.json(...)
  - https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameReader