4_GroupBy_and_Aggregate_Functions

February 22, 2022

1 GroupBy and Aggregate Functions

how to use GroupBy and Aggregate methods on a DataFrame.

- **GroupBy** allows you to **group rows together based on some column value**,

  for example, you could group together sales data by the day the sale occured, or group repeast customer data based off the name of the customer. Once you've performed the GroupBy operation you can use an aggregate function off that data.

- **An aggregate function** aggregates multiple rows of data into a single output, such as taking the sum of inputs, or **counting the number of inputs**.

  Let's see some examples on an example dataset!

[1]:
```python
from pyspark.sql import SparkSession
```

[2]:
```python
# May take a little while on a local computer
spark = SparkSession.builder.appName("groupbyagg").getOrCreate()
```

```
22/02/22 11:57:43 WARN Utils: Your hostname, ThinkCentre resolves to a loopback
address: 127.0.1.1; using 10.180.5.223 instead (on interface eno1)
22/02/22 11:57:43 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
22/02/22 11:57:44 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
```

Read in the customer sales data

[3]:
```python
df = spark.read.csv('sales_info.csv',inferSchema=True,header=True)
```

[4]:
```python
df.printSchema()
```

```
root
 |-- Company: string (nullable = true)
 |-- Person: string (nullable = true)
 |-- Sales: double (nullable = true)
```

[5]:
```
df.show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|    Sam|200.0|
|   GOOG|Charlie|120.0|
|   GOOG|  Frank|340.0|
|   MSFT|   Tina|600.0|
|   MSFT|    Amy|124.0|
|   MSFT|Vanessa|243.0|
|     FB|   Carl|870.0|
|     FB|  Sarah|350.0|
|   APPL|   John|250.0|
|   APPL|  Linda|130.0|
|   APPL|   Mike|750.0|
|   APPL|  Chris|350.0|
+-------+-------+-----+
```

[6]:
```
# How many rows?
df.count()
```

[6]:
```
12
```

1.1 GroupBy

Let's group together by company!

Signature: df.groupBy(*cols)

Groups the `DataFrame` using the specified columns, so we can run aggregation on them.

`groupby` is an alias for `groupBy`.

param cols:

- list of columns to group by.
- Each element should be a column name (string) or an expression
- Returns a GroupedDataObject

[7]:
```
df.groupBy("Company")
```

[7]:
```
<pyspark.sql.group.GroupedData at 0x7ff08fe2e6d0>
```

This returns a GroupedData object, off of which you can all various methods

[8]:
```
# Mean
df.groupBy("Company").mean().show()
```

```
+-------+-----------------+
|Company|       avg(Sales)|
+-------+-----------------+
|   APPL|            370.0|
|   GOOG|            220.0|
|     FB|            610.0|
|   MSFT|322.3333333333333|
+-------+-----------------+
```

[9]:
```
# Count
df.groupBy("Company").count().show()
```

```
+-------+-----+
|Company|count|
+-------+-----+
|   APPL|    4|
|   GOOG|    3|
|     FB|    2|
|   MSFT|    3|
+-------+-----+
```

[10]:
```
# Max
df.groupBy("Company").max().show()
```

```
+-------+----------+
|Company|max(Sales)|
+-------+----------+
|   APPL|     750.0|
|   GOOG|     340.0|
|     FB|     870.0|
|   MSFT|     600.0|
+-------+----------+
```

[11]:
```
# Min
df.groupBy("Company").min().show()
```

```
+-------+----------+
|Company|min(Sales)|
+-------+----------+
|   APPL|     130.0|
|   GOOG|     120.0|
|     FB|     350.0|
|   MSFT|     124.0|
+-------+----------+
```

[12]:
```python
# Sum
df.groupBy("Company").sum().show()
```

```
+-------+----------+
|Company|sum(Sales)|
+-------+----------+
|   APPL|    1480.0|
|   GOOG|     660.0|
|     FB|    1220.0|
|   MSFT|     967.0|
+-------+----------+
```

Check out this link for more info on other methods: http://spark.apache.org/docs/latest/api/python/pyspark.sql.h
sql-module

1.2 Aggregation

- Not all methods need a groupby call, instead you can just call the generalized .agg() method, that will call the aggregate across all rows in the dataframe column specified.
- It can take in arguments as a single column, or create multiple aggregate calls all at once using dictionary notation.

For example:

[13]:
```python
# Max sales across everything
df.agg({'Sales':'max'}).show()
```

```
+----------+
|max(Sales)|
+----------+
|     870.0|
+----------+
```

[14]:
```python
# same result
df.groupBy().max().show()
```

```
+----------+
|max(Sales)|
+----------+
|     870.0|
+-----------+
```

[15]:
```
# Could have done this on the group by object as well:
```

[16]:
```
grouped = df.groupBy("Company")
```

[17]:
```
grouped.agg({"Sales":'max'}).show()
```

```
+-------+----------+
|Company|max(Sales)|
+-------+----------+
|   APPL|     750.0|
|   GOOG|     340.0|
|     FB|     870.0|
|   MSFT|     600.0|
+-------+----------+
```

1.2.1 SQL Functions

There are a variety of functions you can import from pyspark.sql.functions. Check out the documentation for the full list available: http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions

[18]:
```
from pyspark.sql.functions import countDistinct, avg, stddev
```

[19]:
```
df.select(countDistinct("Sales")).show()
```

```
[Stage 70:=====================================================>(197 + 3) / 200]
```

```
+---------------------+
|count(DISTINCT Sales)|
+---------------------+
|                   11|
+---------------------+
```

Often you will want to change the name, use the .alias() method for this:

[20]:
```
df.select(countDistinct("Sales").alias("Distinct Sales")).show()
```

```
+--------------+
|Distinct Sales|
+--------------+
|            11|
+--------------+
```

[21]:
```
df.select(avg('Sales')).show()
```

```
+-----------------+
|       avg(Sales)|
+-----------------+
|360.5833333333333|
+-----------------+
```

[22]:
```
df.select(stddev("Sales")).show()
```

```
+------------------+
|stddev_samp(Sales)|
+------------------+
|250.08742410799007|
+------------------+
```

That is a lot of precision for digits! Let's use the format_number to fix that!

[23]:
```
from pyspark.sql.functions import format_number
```

[24]:
```
sales_std = df.select(stddev("Sales").alias('std'))
```

[25]:
```
sales_std
```

[25]:
```
DataFrame[std: double]
```

[26]:
```
sales_std.show()
```

```
+------------------+
|               std|
+------------------+
|250.08742410799007|
+------------------+
```

[27]:
```
# format_number("col_name",decimal places)
sales_std.select(format_number('std',2).alias('std_2digits')).show()
```

```
+-----------+
|std_2digits|
+-----------+
|     250.09|
+-----------+
```

[28]:
```
# or with this one liner
df.select(stddev("Sales").alias('std')).select(format_number('std',2).
 ↪alias('std_2digits')).show()
```

```
+-----------+
|std_2digits|
+-----------+
|     250.09|
+-----------+
```

1.3 Order By

You can easily sort with the orderBy method:

[29]:
```
# OrderBy
# Ascending
df.orderBy("Sales").show()

# this produces the same result
# df.orderBy(df["Sales"]).show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|Charlie|120.0|
|   MSFT|    Amy|124.0|
|   APPL|  Linda|130.0|
|   GOOG|    Sam|200.0|
|   MSFT|Vanessa|243.0|
|   APPL|   John|250.0|
|   GOOG|  Frank|340.0|
|     FB|  Sarah|350.0|
|   APPL|  Chris|350.0|
|   MSFT|   Tina|600.0|
|   APPL|   Mike|750.0|
|     FB|   Carl|870.0|
+-------+-------+-----+
```

[30]:
```
# Descending call off the column itself.
df.orderBy(df["Sales"].desc()).show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|     FB|   Carl|870.0|
|   APPL|   Mike|750.0|
|   MSFT|   Tina|600.0|
|     FB|  Sarah|350.0|
|   APPL|  Chris|350.0|
|   GOOG|  Frank|340.0|
|   APPL|   John|250.0|
|   MSFT|Vanessa|243.0|
|   GOOG|    Sam|200.0|
|   APPL|  Linda|130.0|
|   MSFT|    Amy|124.0|
|   GOOG|Charlie|120.0|
+-------+-------+-----+
```

2 SQL Queries on DF

[31]:
```
df.createOrReplaceTempView("sales_df")
```

[32]:
```
spark.sql("SELECT Person,Company from sales_df WHERE sales > 300").show()
```

```
+------+-------+
|Person|Company|
+------+-------+
| Frank|   GOOG|
|  Tina|   MSFT|
|  Carl|     FB|
| Sarah|     FB|
|  Mike|   APPL|
| Chris|   APPL|
+------+-------+
```

[33]:
```
spark.sql("SELECT * from sales_df WHERE sales > 300 and Company='APPL'").show()
```

```
+-------+------+-----+
|Company|Person|Sales|
+-------+------+-----+
|   APPL|  Mike|750.0|
|   APPL| Chris|350.0|
+-------+------+-----+
```

[34]:
```
spark.sql("SELECT * from sales_df WHERE sales > 300 and Company='APPL' order by␣
 ↪sales").show()
```

```
+-------+------+-----+
|Company|Person|Sales|
+-------+------+-----+
|   APPL| Chris|350.0|
|   APPL|  Mike|750.0|
+-------+------+-----+
```

```
[35]: spark.sql("SELECT * from sales_df WHERE sales > 300 and Company='APPL' order by␣
      ↪sales desc").show()
```

```
+-------+------+-----+
|Company|Person|Sales|
+-------+------+-----+
|   APPL|  Mike|750.0|
|   APPL| Chris|350.0|
+-------+------+-----+
```

```
[36]: spark.sql("SELECT * from sales_df WHERE sales > 300 and Company like '%F%'␣
      ↪order by sales desc").show()
```

```
+-------+------+-----+
|Company|Person|Sales|
+-------+------+-----+
|     FB|  Carl|870.0|
|   MSFT|  Tina|600.0|
|     FB| Sarah|350.0|
+-------+------+-----+
```

Check out the documentation for more! https://spark.apache.org/docs/latest/sql-programming-guide.html