graphlab.SFrame.groupby

`SFrame.` `groupby` ***(key_columns, operations, *args)***

Perform a group on the key_columns followed by aggregations on the columns listed in operations.

The operations parameter is a dictionary that indicates which aggregation operators to use and which columns to use them on. The available operators are SUM, MAX, MIN, COUNT, AVG, VAR, STDV, CONCAT, SELECT_ONE, ARGMIN, ARGMAX, and QUANTILE. For convenience, aggregators MEAN, STD, and VARIANCE are available as synonyms for AVG, STDV, and VAR. See `aggregate` for more detail on the aggregators.

Parameters: **key_columns** : string | list[string]

Column(s) to group by. Key columns can be of any type other than dictionary.

operations : dict, list

Dictionary of columns and aggregation operations. Each key is a output column name and each value is an aggregator. This can also be a list of aggregators, in which case column names will be automatically assigned.

***args**

All other remaining arguments will be interpreted in the same way as the operations argument.

Returns: **out_sf** : SFrame

A new SFrame, with a column for each groupby column and each aggregation operation.

See also

`aggregate`

Examples

Suppose we have an SFrame with movie ratings by many users.

```
>>> import graphlab.aggregate as agg
>>> url = 'http://s3.amazonaws.com/gl-testdata/rating_data_example.csv'
>>> sf = graphlab.SFrame.read_csv(url)
>>> sf
+---------+----------+--------+
| user_id | movie_id | rating |
+---------+----------+--------+
|  25904  |   1663   |   3    |
|  25907  |   1663   |   3    |
|  25923  |   1663   |   3    |
|  25924  |   1663   |   3    |
|  25928  |   1663   |   2    |
|  25933  |   1663   |   4    |
|  25934  |   1663   |   4    |
|  25935  |   1663   |   4    |
|  25936  |   1663   |   5    |
|  25937  |   1663   |   2    |
|   ...   |   ...    |  ...   |
+---------+----------+--------+
[10000 rows x 3 columns]
```

Compute the number of occurrences of each user.

```
>>> user_count = sf.groupby(key_columns='user_id',
...                         operations={'count': agg.COUNT()})
>>> user_count
+---------+-------+
| user_id | count |
+---------+-------+
|  62361  |   1   |
|  30727  |   1   |
|  40111  |   1   |
|  50513  |   1   |
|  35140  |   1   |
|  42352  |   1   |
|  29667  |   1   |
|  46242  |   1   |
|  58310  |   1   |
|  64614  |   1   |
|   ...   |  ...  |
+---------+-------+
[9852 rows x 2 columns]
```

Compute the mean and standard deviation of ratings per user.

```
>>> user_rating_stats = sf.groupby(key_columns='user_id',
...                                 operations={
...                                     'mean_rating': agg.MEAN('rating'),
...                                     'std_rating': agg.STD('rating')
...                                 })
>>> user_rating_stats
+---------+-------------+------------+
| user_id | mean_rating | std_rating |
+---------+-------------+------------+
|  62361  |     5.0     |    0.0     |
|  30727  |     4.0     |    0.0     |
|  40111  |     2.0     |    0.0     |
|  50513  |     4.0     |    0.0     |
|  35140  |     4.0     |    0.0     |
|  42352  |     5.0     |    0.0     |
|  29667  |     4.0     |    0.0     |
|  46242  |     5.0     |    0.0     |
|  58310  |     2.0     |    0.0     |
|  64614  |     2.0     |    0.0     |
|   ...   |     ...     |    ...     |
+---------+-------------+------------+
[9852 rows x 3 columns]
```

Compute the movie with the minimum rating per user.

```
>>> chosen_movies = sf.groupby(key_columns='user_id',
...                             operations={
...                                 'worst_movies': agg.ARGMIN('rating','movie_id')
...                             })
>>> chosen_movies
+---------+-------------+
| user_id | worst_movies |
+---------+-------------+
|  62361  |     1663    |
|  30727  |     1663    |
|  40111  |     1663    |
|  50513  |     1663    |
|  35140  |     1663    |
|  42352  |     1663    |
|  29667  |     1663    |
|  46242  |     1663    |
|  58310  |     1663    |
|  64614  |     1663    |
|   ...   |     ...     |
+---------+-------------+
[9852 rows x 2 columns]
```

Compute the movie with the max rating per user and also the movie with the maximum imdb-ranking per user.

```
>>> sf['imdb-ranking'] = sf['rating'] * 10
>>> chosen_movies = sf.groupby(key_columns='user_id',
...         operations={('max_rating_movie','max_imdb_ranking_movie'):
agg.ARGMAX(('rating','imdb-ranking'),'movie_id')})
>>> chosen_movies
+---------+------------------+------------------------+
| user_id | max_rating_movie | max_imdb_ranking_movie |
+---------+------------------+------------------------+
|  62361  |       1663       |         16630          |
|  30727  |       1663       |         16630          |
|  40111  |       1663       |         16630          |
|  50513  |       1663       |         16630          |
|  35140  |       1663       |         16630          |
|  42352  |       1663       |         16630          |
|  29667  |       1663       |         16630          |
|  46242  |       1663       |         16630          |
|  58310  |       1663       |         16630          |
|  64614  |       1663       |         16630          |
|   ...   |       ...        |          ...           |
+---------+------------------+------------------------+
[9852 rows x 3 columns]
```

Compute the movie with the max rating per user.

```
>>> chosen_movies = sf.groupby(key_columns='user_id',
            operations={'best_movies': agg.ARGMAX('rating','movie')})
```

Compute the movie with the max rating per user and also the movie with the maximum imdb-ranking per user.

```
>>> chosen_movies = sf.groupby(key_columns='user_id',
           operations={('max_rating_movie','max_imdb_ranking_movie'):
agg.ARGMAX(('rating','imdb-ranking'),'movie')})
```

Compute the count, mean, and standard deviation of ratings per (user, time), automatically assigning output column names.

```
>>> sf['time'] = sf.apply(lambda x: (x['user_id'] + x['movie_id']) % 11 + 2000)
>>> user_rating_stats = sf.groupby(['user_id', 'time'],
...                                [agg.COUNT(),
...                                 agg.AVG('rating'),
...                                 agg.STDV('rating')])
>>> user_rating_stats
+------+---------+-------+---------------+----------------+
| time | user_id | Count | Avg of rating | Stdv of rating |
+------+---------+-------+---------------+----------------+
| 2006 |  61285  |   1   |      4.0      |      0.0       |
| 2000 |  36078  |   1   |      4.0      |      0.0       |
| 2003 |  47158  |   1   |      3.0      |      0.0       |
| 2007 |  34446  |   1   |      3.0      |      0.0       |
| 2010 |  47990  |   1   |      3.0      |      0.0       |
| 2003 |  42120  |   1   |      5.0      |      0.0       |
| 2007 |  44940  |   1   |      4.0      |      0.0       |
| 2008 |  58240  |   1   |      4.0      |      0.0       |
| 2002 |   102   |   1   |      1.0      |      0.0       |
| 2009 |  52708  |   1   |      3.0      |      0.0       |
| ...  |   ...   |  ...  |      ...      |      ...       |
+------+---------+-------+---------------+----------------+
[10000 rows x 5 columns]
```

The groupby function can take a variable length list of aggregation specifiers so if we want the count and the 0.25 and 0.75 quantiles of ratings:

```
>>> user_rating_stats = sf.groupby(['user_id', 'time'], agg.COUNT(),
...                                {'rating_quantiles': agg.QUANTILE('rating',[0.25,
0.75])})
>>> user_rating_stats
+------+---------+-------+------------------------+
| time | user_id | Count |    rating_quantiles    |
+------+---------+-------+------------------------+
| 2006 |  61285  |   1   | array('d', [4.0, 4.0]) |
| 2000 |  36078  |   1   | array('d', [4.0, 4.0]) |
| 2003 |  47158  |   1   | array('d', [3.0, 3.0]) |
| 2007 |  34446  |   1   | array('d', [3.0, 3.0]) |
| 2010 |  47990  |   1   | array('d', [3.0, 3.0]) |
| 2003 |  42120  |   1   | array('d', [5.0, 5.0]) |
| 2007 |  44940  |   1   | array('d', [4.0, 4.0]) |
| 2008 |  58240  |   1   | array('d', [4.0, 4.0]) |
| 2002 |   102   |   1   | array('d', [1.0, 1.0]) |
| 2009 |  52708  |   1   | array('d', [3.0, 3.0]) |
| ...  |   ...   |  ...  |          ...           |
+------+---------+-------+------------------------+
[10000 rows x 4 columns]
```

To put all items a user rated into one list value by their star rating:

```
>>> user_rating_stats = sf.groupby(["user_id", "rating"],
...                                {"rated_movie_ids":agg.CONCAT("movie_id")})
>>> user_rating_stats
+--------+---------+----------------------+
| rating | user_id |     rated_movie_ids  |
+--------+---------+----------------------+
|   3    |  31434  | array('d', [1663.0]) |
|   5    |  25944  | array('d', [1663.0]) |
|   4    |  38827  | array('d', [1663.0]) |
|   4    |  51437  | array('d', [1663.0]) |
|   4    |  42549  | array('d', [1663.0]) |
|   4    |  49532  | array('d', [1663.0]) |
|   3    |  26124  | array('d', [1663.0]) |
|   4    |  46336  | array('d', [1663.0]) |
|   4    |  52133  | array('d', [1663.0]) |
|   5    |  62361  | array('d', [1663.0]) |
|  ...   |   ...   |         ...          |
+--------+---------+----------------------+
[9952 rows x 3 columns]
```

To put all items and rating of a given user together into a dictionary value:

```
>>> user_rating_stats = sf.groupby("user_id",
...                                {"movie_rating":agg.CONCAT("movie_id",
"rating")})
>>> user_rating_stats
+---------+--------------+
| user_id | movie_rating |
+---------+--------------+
|  62361  |  {1663: 5}   |
|  30727  |  {1663: 4}   |
|  40111  |  {1663: 2}   |
|  50513  |  {1663: 4}   |
|  35140  |  {1663: 4}   |
|  42352  |  {1663: 5}   |
|  29667  |  {1663: 4}   |
|  46242  |  {1663: 5}   |
|  58310  |  {1663: 2}   |
|  64614  |  {1663: 2}   |
|   ...   |     ...      |
+---------+--------------+
[9852 rows x 2 columns]
```