Google

#GeoForGood19

Earth Engine Table Processing

By example with Joins

Eric Engle / 2019-09-17

Agenda

Examples of table processing; filters, joins, groups, and more

- Motivating example
- Overview of Joins and Filters
- Simple join
- Inverted join
- Grouping
- Inner join
- Compound groups
- Non-local filters
- Questions

Motivating Example

A simple observation:

*We don't want power stations in **our** backyard.*

A made-up question (with made-up answers 👍)

Are power stations in the backyard of vulnerable habitats?

Motivating Example - Ecoregions

1. Find “**Resolve Ecoregions 2017**”, and import it as “**regions**”.
2. Run:

```
print(regions.reduceColumns('frequencyHistogram', ['NNH_NAME']));
```

3. Compare your results:

```
Nature Imperiled: 276
Nature Could Recover: 294
Nature Could Reach Half Protected: 346
Half Protected: 125
N/A: 64
```

This “Nature Needs Half” scale isn't detailed. But it gives an idea of how stressed an ecoregion is.

Motivating Example - Power Stations

1. Find “**Global Power Plant Database**”, and import it as “**stations**”.
2. Run:

```
print(stations.reduceColumns(’frequencyHistogram’, ['fuel1']));
```

3. Compare your results:

Biomass: 1290 Coal: 2172 Cogeneration: 43 Gas: 3068 Geothermal: 186	Hydro: 7034 Nuclear: 199 Oil: 2925 Other: 36 Petcoke: 8	Solar: 5424 Storage: 39 Waste: 1143 Wave and Tidal: 10 Wind: 5084

Overview - Joins

If we relate regions to stations by whether they spatially intersect, then

- **Simple join**
  Get each ecoregion that contains at least 1 station.
- **Inverted join**
  Get each ecoregion that contains no stations.
- **Inner join**
  Get all *region,station* pairs that intersect.

- **saveAll join**
  Add a list of stations to each ecoregion.
- **saveFirst join**
  Find the best station according to some property.
- **saveBest join**
  Find the best station according to the filter.

Overview - Filters

Join inputs and outputs can be filtered. Only some filters can control the join:

- **Geometry**
  intersects, disjoint, contains, isContained, withinDistance
- **Ordering**
  greaterThan, greaterThanOrEquals, lessThan, lessThanOrEquals
- **Equality**
  equals, notEquals, inList, listContains
- **Numbers**
  maxDifference
- **Strings**
  stringContains, stringEndsWith, stringStartsWith
- **Dates**
  dateRangeContains

A couple of notes:

1. Non-join filters work better and are easier to use when applied to the inputs.
2. Some filters have simpler versions, e.g. "greaterThan" vs. "gt".
   The simpler version is easier for filtering 1 table, but can't join 2 tables.

Simple join

Compare NNH categories with power stations:

1. Run:

```
var filter = ee.Filter.intersects({
  leftField: '.geo', rightField: '.geo'});
print(ee.Join.simple().apply(regions, stations, filter)
  .reduceColumns('frequencyHistogram', ['NNH_NAME']));
```

2. Compare your results:

```
Half Protected: 44
Nature Could Reach Half Protected: 226
Nature Could Recover: 208
Nature Imperiled: 190
```

Inverted join

Compare NNH categories without power stations:

1. Run:

```
print(ee.Join.inverted().apply(regions, stations, filter)
  .reduceColumns('frequencyHistogram', ['NNH_NAME']));
```

2. Compare your results:

```
Half Protected: 81
Nature Could Reach Half Protected: 120
Nature Could Recover: 86
Nature Imperiled: 86
```

2x "Half Protected" regions without stations, and 2x "Could Reach Half" regions with stations...

Grouping

That was interesting, but ignores how much capacity we're talking about. So sum station capacity by region, and group by NNH name:

Run:

```
print(ee.Join.saveAll('stations').apply(regions, stations, filter)
  .map(sumRegionCapacity).reduceColumns(
    ee.Reducer.sum().group(), ['NNH_NAME', 'capacity']));
function sumRegionCapacity(region) {
  var matches = ee.List(region.get('stations'));
  var capacities = matches.map(function(f) {
    return ee.Feature(f).getNumber('capacitymw');
  });
  return region.set('capacity', capacities.reduce('sum'));
}
```

Grouping - Cont.

Compare your results:

```
Half Protected: 36877.92372
Nature Could Reach Half Protected: 1031062.7612490002
Nature Could Recover: 1541769.2586080013
Nature Imperiled: 2243652.195029999
```

Some made-up observations:

- <1% of capacity is in half protected regions.
  Protection works?
- ~50% of capacity is in significant peril.
  Drill down by urbanization?
- ~50% of capacity is in regions that could go either way.
  Drill into NNH scoring system to identify most actionable sites?

Inner join

An inner join is not as flexible, but could be easier:

```
print(ee.Join.inner('region', 'station')
  .apply(regions, stations, filter)
  .map(copyCapacity)
  .reduceColumns(ee.Reducer.sum().group(), ['NNH_NAME', 'capacitymw']));
function copyCapacity(pair) {
  var region = ee.Feature(pair.get('region'));
  var station = ee.Feature(pair.get('station'));
  return region.set('capacitymw', station.get('capacitymw'));
}
```

Inner join - Cont.

Compare your results:

```
Half Protected: 36877.92372
Nature Could Reach Half Protected: 1031062.7612489976
Nature Could Recover: 1541769.2586080045
Nature Imperiled: 2243652.195029997
```

Compound groups

What if we want to group capacity by both NNH name and fuel type?

```
print(ee.Join.inner('region', 'station')
  .apply(regions, stations, filter)
  .map(merge)
  .reduceColumns(ee.Reducer.sum().group(), ['group', 'capacity']));
function merge(f) {
  var region = ee.Feature(f.get('region'));
  var name = ee.String(region.get('NNH_NAME'));
  var station = ee.Feature(f.get('station'));
  return ee.Feature(null, {
    group: name.cat(',').cat(station.get('fuel1')),
    capacity: station.get('capacitymw')
  });
}
```

Compound groups - Cont.

The result is a list of groups, e.g.

```
{"group": "Half Protected,Gas", "sum": 11.68}.
```

A little rearranging and we can make a nice Pivot Table in Sheets.

(Yes, this **should** be easier).

Non-local filters

To filter based on multiple rows, you need a join!

Find stations with max capacity in each Mangrove region:

```
var mangroves = regions.filter(ee.Filter.eq('BIOME_NAME', 'Mangroves'));
var results = ee.Join.saveFirst('station', 'capacitymw', false)
  .apply(mangroves, stations, filter)
  .map(function(f) { return ee.Feature(f.get('station')); });
print(results.reduceColumns('frequencyHistogram', ['fuel1']));
Map.addLayer(results);
```

Non-local filters - Cont.

Compare results:

```
Coal: 1
Gas: 7
Hydro: 1
Nuclear: 3
Oil: 6
Waste: 1
```

Thank you!

Any questions?

http://bit.ly/eejoins_2019