03_pandas_groupby

August 13, 2021

1 Pandas: Groupby

`groupby` is an amazingly powerful function in pandas. But it is also complicated to use and understand. The point of this lesson is to make you feel confident in using `groupby` and its cousins, `resample` and `rolling`.

These notes are loosely based on the Pandas GroupBy Documentation.

The "split/apply/combine" concept was first introduced in a paper by Hadley Wickham: https://www.jstatsoft.org/article/view/v040i01.

1.1 Credit:

this comes from Abernathys open book, which we will be looking at a lot! https://earth-env-data-science.github.io/lectures/core_python/python_fundamentals.html

Imports:

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
%matplotlib inline
```

First we read in some Earthquake data.

In this case, we are loading **straight from a webpage** – its like a relative path, but pointed at a remote file

[2]:
```
url = 'http://www.ldeo.columbia.edu/~rpa/usgs_earthquakes_2014.csv'

df = pd.read_csv(url, parse_dates=['time'], index_col='id')

# don't worry about this step:
df['country'] = df.place.str.split(', ').str[-1]
df_small = df[df.mag<4]
df = df[df.mag>4]
df.head()
```

[2]:
```
                              time  latitude  longitude   depth  mag magType  \
id
usc000mqlp 2014-01-31 23:08:03.660   -4.9758   153.9466  110.18  4.2      mb
```

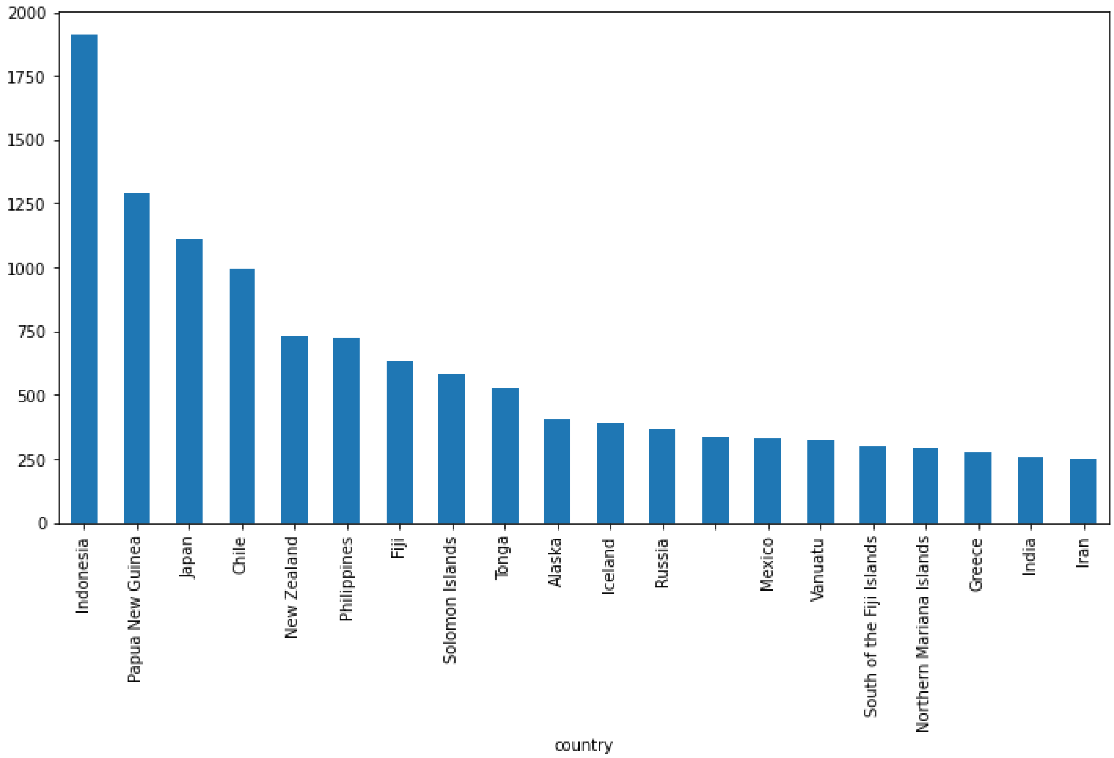
```
usc000mqln 2014-01-31 22:54:32.970  -28.1775  -177.9058   95.84  4.3      mb
usc000mqls 2014-01-31 22:49:49.740  -23.1192   179.1174  528.34  4.4      mb
usc000mf1x 2014-01-31 22:19:44.330   51.1569  -178.0910   37.50  4.2      mb
usc000mqlm 2014-01-31 21:56:44.320   -4.8800   153.8434  112.66  4.3      mb

            nst    gap   dmin   rms net                   updated  \
id
usc000mqlp  NaN   98.0  1.940  0.61  us  2014-04-08T01:43:19.000Z
usc000mqln  NaN  104.0  1.063  1.14  us  2014-04-08T01:43:19.000Z
usc000mqls  NaN   80.0  5.439  0.95  us  2014-04-08T01:43:19.000Z
usc000mf1x  NaN    NaN    NaN  0.83  us  2014-04-08T01:43:19.000Z
usc000mqlm  NaN  199.0  1.808  0.79  us  2014-04-08T01:43:19.000Z

                                       place        type  \
id
usc000mqlp  115km ESE of Taron, Papua New Guinea  earthquake
usc000mqln  120km N of Raoul Island, New Zealand  earthquake
usc000mqls             South of the Fiji Islands  earthquake
usc000mf1x    72km E of Amatignak Island, Alaska  earthquake
usc000mqlm  100km ESE of Taron, Papua New Guinea  earthquake

                              country
id
usc000mqlp           Papua New Guinea
usc000mqln                New Zealand
usc000mqls  South of the Fiji Islands
usc000mf1x                     Alaska
usc000mqlm           Papua New Guinea
```

1.2 An Example

This is an example of a "one-liner" that you can accomplish with groupby.

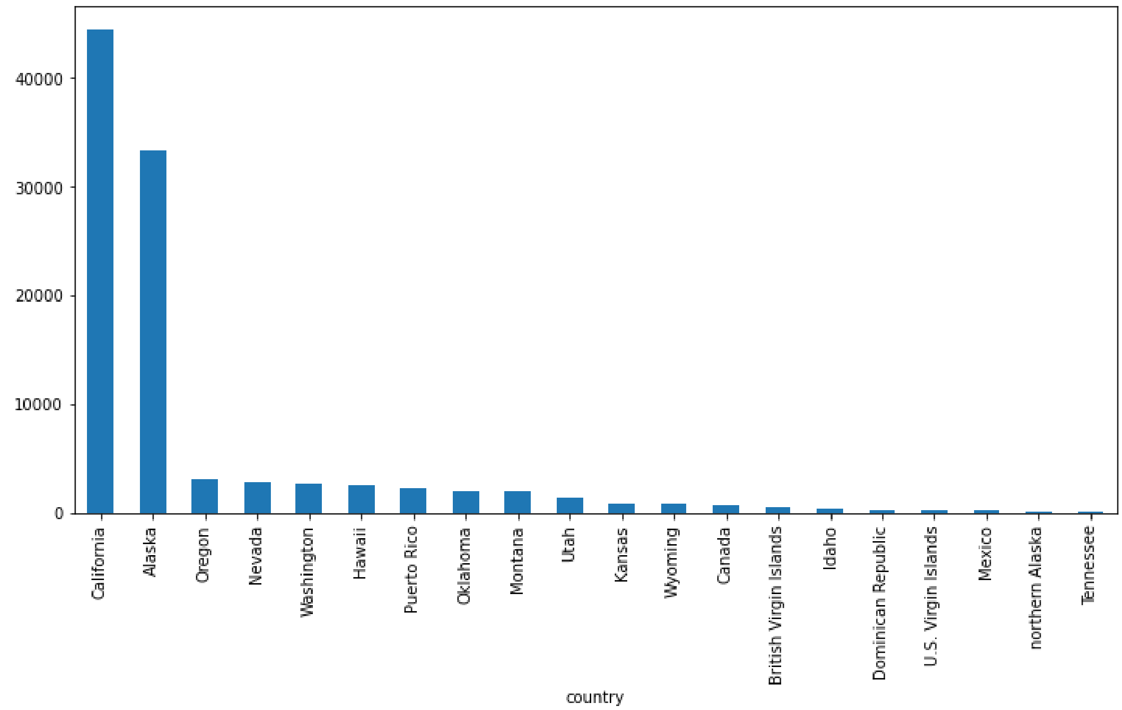
[3]:
```
df.groupby('country').mag.count().nlargest(20).plot(kind='bar', figsize=(12,6))
```

[3]:
```
<AxesSubplot:xlabel='country'>
```



[4]:
```
df_small.groupby('country').mag.count().nlargest(20).plot(kind='bar',␣
↪figsize=(12,6))
```

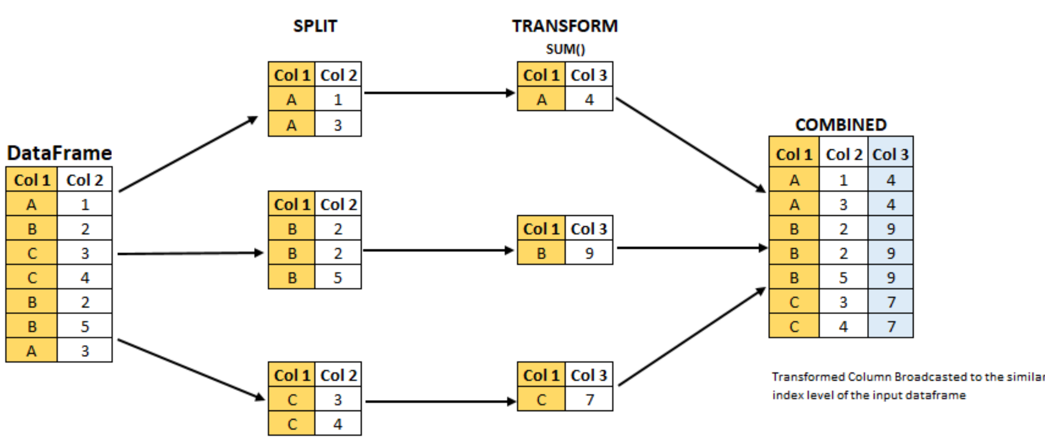
[4]: <AxesSubplot:xlabel='country'>



1.3 What Happened?

Let's break apart this operation a bit. The workflow with `groupby` can be divided into three general steps:

1. **Split**: Partition the data into different groups based on some criterion.
2. **Apply**: Do some caclulation within each group. Different types of "apply" steps might be
3. *Aggregation*: Get the mean or max within the group.
4. *Transformation*: Normalize all the values within a group
5. *Filtration*: Eliminate some groups based on a criterion.
6. **Combine**: Put the results back together into a single object.



1.3.1 The groupby method

Both `Series` and `DataFrame` objects have a groupby method. It accepts a variety of arguments, but the simplest way to think about it is that you pass another series, whose unique values are used to split the original object into different groups.

via https://medium.com/analytics-vidhya/split-apply-combine-strategy-for-data-mining-4fd6e2a0cc

[5]:
```python
df.groupby(df.country)
```

[5]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc7369358e0>
```

There is a shortcut for doing this with dataframes: you just pass the column name:

[6]:
```python
df.groupby('country')
```

[6]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc73abea340>
```

1.3.2 The `GroubBy` object

When we call, `groupby` we get back a `GroupBy` object, which is like a dictionary, where the keys are each group, and the values are the data that correspond to that group

[7]:
```python
gb = df.groupby('country')
gb
```

[7]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc73a8beb80>
```

The length tells us how many groups were found:

[8]:
```python
len(gb)
```

[8]:
```
262
```

All of the groups are available as a dictionary via the `.groups` attribute:

[9]:
```python
groups = gb.groups
len(groups)
```

[9]:
```
262
```

[10]:
```python
# list(groups.keys())
groups.keys()
```

[10]:
```
dict_keys(['', 'Afghanistan', 'Alaska', 'Albania', 'Algeria', 'American Samoa',
'Angola', 'Anguilla', 'Antarctica', 'Argentina', 'Arizona', 'Aruba', 'Ascension
Island region', 'Australia', 'Azerbaijan', 'Azores Islands region', 'Azores-Cape
St. Vincent Ridge', 'Balleny Islands region', 'Banda Sea', 'Bangladesh',
'Barbados', 'Barbuda', 'Bay of Bengal', 'Bermuda', 'Bhutan', 'Bolivia', 'Bosnia
and Herzegovina', 'Bouvet Island', 'Bouvet Island region', 'Brazil', 'British
Indian Ocean Territory', 'British Virgin Islands', 'Burma', 'Burundi',
```

'California', 'Canada', 'Cape Verde', 'Carlsberg Ridge', 'Cayman Islands',
'Celebes Sea', 'Central East Pacific Rise', 'Central Mid-Atlantic Ridge',
'Chagos Archipelago region', 'Chile', 'China', 'Christmas Island', 'Colombia',
'Comoros', 'Cook Islands', 'Costa Rica', 'Crozet Islands region', 'Cuba',
'Cyprus', 'Davis Strait', 'Democratic Republic of the Congo', 'Djibouti',
'Dominica', 'Dominican Republic', 'Drake Passage', 'East Timor', 'East of
Severnaya Zemlya', 'East of the Kuril Islands', 'East of the North Island of New
Zealand', 'East of the Philippine Islands', 'East of the South Sandwich
Islands', 'Easter Island region', 'Eastern Greenland', 'Ecuador', 'Ecuador
region', 'Egypt', 'El Salvador', 'Eritrea', 'Ethiopia', 'Falkland Islands
region', 'Federated States of Micronesia region', 'Fiji', 'Fiji region',
'France', 'French Polynesia', 'French Southern Territories', 'Galapagos Triple
Junction region', 'Georgia', 'Greece', 'Greenland', 'Greenland Sea',
'Guadeloupe', 'Guam', 'Guatemala', 'Gulf of Alaska', 'Haiti', 'Hawaii',
'Honduras', 'Iceland', 'Idaho', 'India', 'India region', 'Indian Ocean Triple
Junction', 'Indonesia', 'Iran', 'Iraq', 'Italy', 'Japan', 'Japan region',
'Jordan', 'Kansas', 'Kazakhstan', 'Kermadec Islands region', 'Kosovo', 'Kuril
Islands', 'Kyrgyzstan', 'Labrador Sea', 'Laptev Sea', 'Macedonia', 'Macquarie
Island region', 'Malawi', 'Malaysia', 'Mariana Islands region', 'Martinique',
'Mauritania', 'Mauritius', 'Mauritius - Reunion region', 'Mexico', 'Micronesia',
'Mid-Indian Ridge', 'Molucca Sea', 'Mongolia', 'Montana', 'Montenegro',
'Morocco', 'Mozambique', 'Mozambique Channel', 'Nepal', 'New Caledonia', 'New
Mexico', 'New Zealand', 'Nicaragua', 'Niue', 'North Atlantic Ocean', 'North
Indian Ocean', 'North Korea', 'North of Ascension Island', 'North of Franz Josef
Land', 'North of New Zealand', 'North of Severnaya Zemlya', 'North of Svalbard',
'Northern East Pacific Rise', 'Northern Mariana Islands', 'Northern Mid-Atlantic
Ridge', 'Northwest of Australia', 'Norway', 'Norwegian Sea', 'Off the coast of
Central America', 'Off the coast of Ecuador', 'Off the coast of Oregon', 'Off
the east coast of the North Island of New Zealand', 'Off the south coast of
Australia', 'Off the west coast of northern Sumatra', 'Oklahoma', 'Oman',
'Oregon', 'Owen Fracture Zone region', 'Pacific-Antarctic Ridge', 'Pakistan',
'Palau', 'Palau region', 'Panama', 'Papua New Guinea', 'Peru', 'Peru-Ecuador
border region', 'Philippine Islands region', 'Philippines', 'Poland',
'Portugal', 'Portugal region', 'Prince Edward Islands', 'Prince Edward Islands
region', 'Puerto Rico', 'Republic of the Congo', 'Reykjanes Ridge', 'Romania',
'Russia', 'Russia region', 'Saint Helena', 'Saint Lucia', 'Saint Vincent and the
Grenadines', 'Samoa', 'Santa Cruz Islands region', 'Saudi Arabia', 'Scotia Sea',
'Sea of Okhotsk', 'Serbia', 'Slovenia', 'Socotra region', 'Solomon Islands',
'Somalia', 'South Africa', 'South Atlantic Ocean', 'South Carolina', 'South
Georgia Island region', 'South Georgia and the South Sandwich Islands', 'South
Indian Ocean', 'South Napa Earthquake', 'South Sandwich Islands', 'South
Sandwich Islands region', 'South Shetland Islands', 'South Sudan', 'South of
Africa', 'South of Australia', 'South of Panama', 'South of Tasmania', 'South of
Tonga', 'South of the Fiji Islands', 'South of the Kermadec Islands', 'South of
the Mariana Islands', 'Southeast Indian Ridge', 'Southeast central Pacific
Ocean', 'Southeast of Easter Island', 'Southern East Pacific Rise', 'Southern
Mid-Atlantic Ridge', 'Southern Pacific Ocean', 'Southwest Indian Ridge',

```
'Southwest of Africa', 'Southwest of Australia', 'Southwestern Atlantic Ocean',
'Spain', 'Sudan', 'Svalbard and Jan Mayen', 'Sweden', 'Syria', 'Taiwan',
'Tajikistan', 'Tanzania', 'Thailand', 'Tonga', 'Tonga region', 'Trinidad and
Tobago', 'Tristan da Cunha region', 'Turkey', 'Turkmenistan', 'Uganda',
'Ukraine', 'United Kingdom', 'Utah', 'Uzbekistan', 'Vanuatu', 'Vanuatu region',
'Venezuela', 'Vietnam', 'Wallis and Futuna', 'West Chile Rise', 'West of
Australia', 'West of Macquarie Island', 'West of Vancouver Island', 'West of the
Galapagos Islands', 'Western Australia', 'Western Indian-Antarctic Ridge',
'Yemen', 'Zambia', 'north of Ascension Island', 'northern Mid-Atlantic Ridge',
'south of Panama', 'western Xizang'])
```

And you can get a specific group by key.

[11]:
```python
gb.get_group('Chile').head()
```

[11]:
```
                              time  latitude  longitude   depth  mag magType  \
id
usc000mqlq 2014-01-31 20:00:16.000  -33.6550   -71.9810   25.10  4.5      mb
usc000mql6 2014-01-31 13:48:23.000  -18.0690   -69.6630  149.10  4.3      mb
usc000mqk8 2014-01-30 14:20:56.560  -19.6118   -70.9487   15.16  4.1      mb
usc000mdi2 2014-01-30 10:02:14.000  -32.1180   -71.7860   25.70  4.5     mwr
usc000mqeh 2014-01-29 18:58:23.000  -18.6610   -69.6440  123.10  4.8      mb

            nst    gap   dmin   rms net                   updated  \
id
usc000mqlq  NaN    NaN    NaN  1.63  us  2014-04-08T01:43:19.000Z
usc000mql6  NaN    NaN    NaN  1.77  us  2014-04-08T01:43:18.000Z
usc000mqk8  NaN  159.0  1.227  1.34  us  2014-04-08T01:43:17.000Z
usc000mdi2  NaN    NaN    NaN  1.10  us  2015-01-30T21:28:21.955Z
usc000mqeh  NaN    NaN    NaN  1.52  us  2014-04-08T01:43:16.000Z

                                     place        type country
id
usc000mqlq  34km WSW of San Antonio, Chile  earthquake   Chile
usc000mql6         17km NW of Putre, Chile  earthquake   Chile
usc000mqk8       107km NW of Iquique, Chile  earthquake   Chile
usc000mdi2       64km NW of La Ligua, Chile  earthquake   Chile
usc000mqeh          51km S of Putre, Chile  earthquake   Chile
```

1.4 Aggregation

Now that we know how to create a `GroupBy` object, let's learn how to do aggregation on it.

One way us to use the `.aggregate` method, which accepts another function as its argument. The result is automatically combined into a new dataframe with the group key as the index.

[12]:
```python
gb.aggregate(np.max).head()
```

[12]:
```
                               time  latitude  longitude   depth  mag magType  \
country
            2014-12-31 14:49:19.200  -37.5219    78.9418  248.18  6.9     mww
Afghanistan 2014-12-27 06:37:50.010   37.0112    71.6062  248.39  5.6     mww
Alaska      2014-12-30 21:22:21.580   67.9858   179.9288  266.61  7.9     mww
Albania     2014-05-20 04:43:25.500   41.5297    20.2804   28.26  5.0     mwr
Algeria     2014-12-26 17:55:18.140   36.9391     5.6063   21.40  5.5     mww

               nst    gap    dmin   rms net                   updated  \
country
               NaN  195.0  28.762  1.47  us  2015-03-17T02:38:27.040Z
Afghanistan    NaN  172.0   3.505  1.55  us  2015-06-22T20:12:10.712Z
Alaska       152.0  338.0   7.712  2.15  us  2015-05-30T05:34:08.822Z
Albania        NaN   69.0   1.299  1.34  us  2015-01-30T15:28:03.533Z
Algeria        NaN  174.0   3.250  1.45  us  2015-03-17T02:37:18.040Z

                                              place        type
country
                         99km NW of Visokoi Island,  earthquake
Afghanistan        8km SE of Ashkasham, Afghanistan  earthquake
Alaska       9km WSW of Little Sitkin Island, Alaska  earthquake
Albania                      6km NE of Durres, Albania  earthquake
Algeria                   5km SSW of Bougara, Algeria  earthquake
```

By default, the operation is applied to every column. That's usually not what we want. We can use both . or [] syntax to select a specific column to operate on. Then we get back a series.

[13]:
```python
gb.mag.aggregate(np.max).head()
```

[13]:
```
country
               6.9
Afghanistan    5.6
Alaska         7.9
Albania        5.0
Algeria        5.5
Name: mag, dtype: float64
```

[14]:
```python
gb.mag.aggregate(np.max).nlargest(10)
```

[14]:
```
country
Chile                           8.2
Alaska                          7.9
Solomon Islands                 7.6
Papua New Guinea                7.5
El Salvador                     7.3
Mexico                          7.2
Fiji                            7.1
Indonesia                       7.1
```

```
Southern East Pacific Rise    7.0
                              6.9
Name: mag, dtype: float64
```

There are shortcuts for common aggregation functions:

[15]:
```python
gb.mag.max().nlargest(10)
```

[15]:
```
country
Chile                         8.2
Alaska                        7.9
Solomon Islands               7.6
Papua New Guinea              7.5
El Salvador                   7.3
Mexico                        7.2
Fiji                          7.1
Indonesia                     7.1
Southern East Pacific Rise    7.0
                              6.9
Name: mag, dtype: float64
```

[16]:
```python
df.groupby('country').mag.mean().nlargest(10)
```

[16]:
```
country
South Napa Earthquake           6.020000
Bouvet Island region            5.750000
South Georgia Island region     5.450000
Barbados                        5.400000
New Mexico                      5.300000
Easter Island region            5.162500
Malawi                          5.100000
Drake Passage                   5.033333
North Korea                     5.000000
Saint Lucia                     5.000000
Name: mag, dtype: float64
```

[17]:
```python
df.groupby('country').mag.std().nlargest(10)
```

[17]:
```
country
Barbados                                  1.555635
Bouvet Island region                      1.484924
Puerto Rico                               0.957601
Off the coast of Ecuador                  0.848528
Palau region                              0.777817
East of the South Sandwich Islands        0.606495
Southern East Pacific Rise                0.604508
South Indian Ocean                        0.602194
Prince Edward Islands region              0.595259
```

```
Panama                                     0.591322
Name: mag, dtype: float64
```

We can also apply multiple functions at once:

[18]:
```
gb.mag.aggregate([np.min, np.max, np.mean]).head()
```

[18]:
```
             amin  amax      mean
country
              4.1   6.9  4.582544
Afghanistan   4.1   5.6  4.410656
Alaska        4.1   7.9  4.515025
Albania       4.1   5.0  4.391667
Algeria       4.1   5.5  4.583333
```

1.5 Transformation

The key difference between aggregation and transformation is that aggregation returns a *smaller* object than the original, indexed by the group keys, while *transformation* returns an object with the same index (and same size) as the original object. Groupby + transformation is used when applying an operation that requires information about the whole group.

In this example, we standardize the earthquakes in each country so that the distribution has zero mean and unit variance. We do this by first defining a function called `standardize` and then passing it to the `transform` method.

I admit that I don't know why you would want to do this. `transform` makes more sense to me in the context of time grouping operation. See below for another example.

[19]:
```
def standardize(x):
    return (x - x.mean())/x.std()

mag_standardized_by_country = gb.mag.transform(standardize)
mag_standardized_by_country.head()
```

[19]:
```
id
usc000mqlp   -0.915774
usc000mqln   -0.675696
usc000mqls   -0.282385
usc000mf1x   -0.684915
usc000mqlm   -0.666807
Name: mag, dtype: float64
```

1.6 Time Grouping

We already saw how pandas has a strong built-in understanding of time. This capability is even more powerful in the context of `groupby`. With datasets indexed by a pandas `DateTimeIndex`, we can easily group and resample the data using common time units.

To get started, let's load the timeseries data we already explored in past lessons.

[20]:
```
import urllib

# this is a way to load data straight from the web

header_url = 'ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/HEADERS.
↪txt'
with urllib.request.urlopen(header_url) as response:
    data = response.read().decode('utf-8')
lines = data.split('\n')
headers = lines[1].split(' ')

ftp_base = 'ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/'
dframes = []
for year in range(2017, 2020):
    data_url = f'{year}/CRND0103-{year}-NY_Millbrook_3_W.txt'
    df = pd.read_csv(ftp_base + data_url, parse_dates=[1],
                     names=headers, header=None, sep='\s+',
                     na_values=[-9999.0, -99.0])
    dframes.append(df)

df = pd.concat(dframes)
df = df.set_index('LST_DATE')
```

[21]:
```
df.head()
```

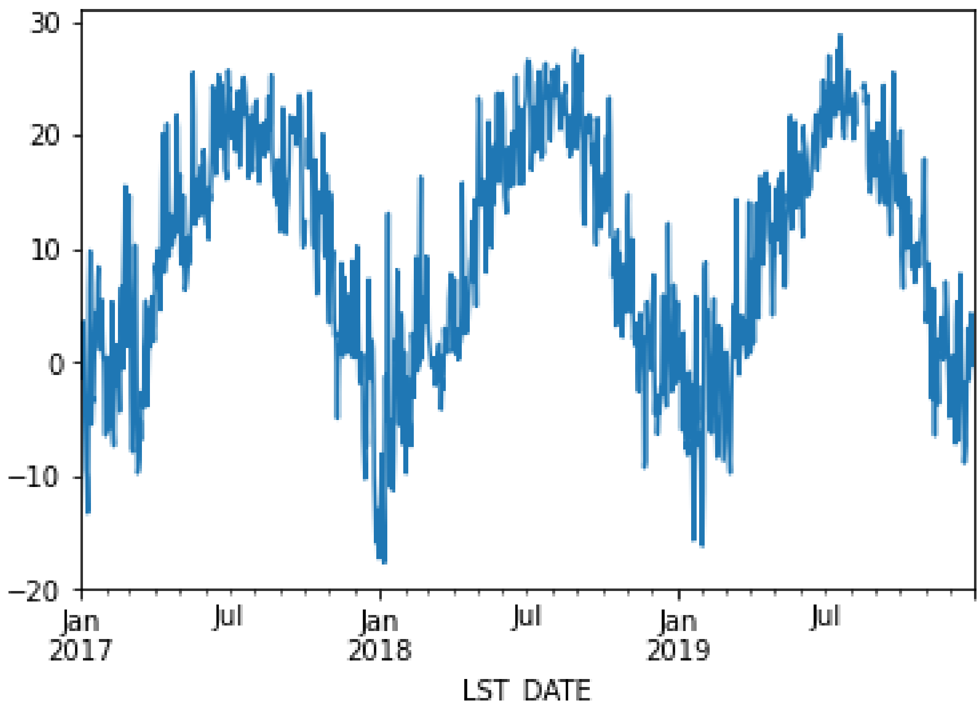
[21]:
```
            WBANNO  CRX_VN  LONGITUDE  LATITUDE  T_DAILY_MAX  T_DAILY_MIN  \
LST_DATE
2017-01-01   64756   2.422     -73.74     41.79          6.6         -5.4
2017-01-02   64756   2.422     -73.74     41.79          4.0         -6.8
2017-01-03   64756   2.422     -73.74     41.79          4.9          0.7
2017-01-04   64756   2.422     -73.74     41.79          8.7         -1.6
2017-01-05   64756   2.422     -73.74     41.79         -0.5         -4.6

            T_DAILY_MEAN  T_DAILY_AVG  P_DAILY_CALC  SOLARAD_DAILY  …  \
LST_DATE                                                            …
2017-01-01           0.6          2.2           0.0           8.68  …
2017-01-02          -1.4         -1.2           0.0           2.08  …
2017-01-03           2.8          2.7          13.1           0.68  …
2017-01-04           3.6          3.5           1.3           2.85  …
2017-01-05          -2.5         -2.8           0.0           4.90  …

           SOIL_MOISTURE_10_DAILY  SOIL_MOISTURE_20_DAILY  \
LST_DATE
2017-01-01                    NaN                   0.207
2017-01-02                    NaN                   0.205
2017-01-03                    NaN                   0.205
2017-01-04                    NaN                   0.215
```

```
2017-01-05                      NaN                  0.215

            SOIL_MOISTURE_50_DAILY  SOIL_MOISTURE_100_DAILY  \
LST_DATE
2017-01-01                   0.152                    0.175
2017-01-02                   0.151                    0.173
2017-01-03                   0.150                    0.173
2017-01-04                   0.153                    0.174
2017-01-05                   0.154                    0.177

            SOIL_TEMP_5_DAILY  SOIL_TEMP_10_DAILY  SOIL_TEMP_20_DAILY  \
LST_DATE
2017-01-01               -0.1                 0.0                 0.6
2017-01-02               -0.2                 0.0                 0.6
2017-01-03               -0.1                 0.0                 0.5
2017-01-04               -0.1                 0.0                 0.5
2017-01-05               -0.1                 0.0                 0.5

            SOIL_TEMP_50_DAILY  SOIL_TEMP_100_DAILY
LST_DATE
2017-01-01                 1.5                  3.4 NaN
2017-01-02                 1.5                  3.3 NaN
2017-01-03                 1.5                  3.3 NaN
2017-01-04                 1.5                  3.2 NaN
2017-01-05                 1.4                  3.1 NaN

[5 rows x 28 columns]
```

This timeseries has daily resolution, and the daily plots are somewhat noisy.

[22]:
```
df.T_DAILY_MEAN.plot()
```

[22]:
```
<AxesSubplot:xlabel='LST_DATE'>
```



A common way to analyze such data in climate science is to create a "climatology," which contains the average values in each month or day of the year. We can do this easily with groupby. Recall that `df.index` is a pandas `DateTimeIndex` object.

[23]:
```
monthly_climatology = df.groupby(df.index.month).mean()
monthly_climatology
```

[23]:
```
           WBANNO    CRX_VN  LONGITUDE  LATITUDE  T_DAILY_MAX  T_DAILY_MIN  \
LST_DATE
1         64756.0  2.555333     -73.74     41.79     2.064516    -7.703226
2         64756.0  2.555333     -73.74     41.79     5.582143    -5.095238
3         64756.0  2.555333     -73.74     41.79     6.093548    -4.308602
4         64756.0  2.555333     -73.74     41.79    15.212222     2.637778
5         64756.0  2.555333     -73.74     41.79    20.736559     8.646237
6         64756.0  2.555333     -73.74     41.79    25.176667    11.970000
7         64756.0  2.555333     -73.74     41.79    28.738710    15.726882
8         64756.0  1.930495     -73.74     41.79    26.951163    14.865116
9         64756.0  2.555333     -73.74     41.79    23.584270    11.620225
10        64756.0  2.615548     -73.74     41.79    17.941758     6.195604
11        64756.0  2.622000     -73.74     41.79     8.260000    -2.223333
12        64756.0  2.622000     -73.74     41.79     3.316129    -5.867742

          T_DAILY_MEAN  T_DAILY_AVG  P_DAILY_CALC  SOLARAD_DAILY  …  \
LST_DATE                                                         …
```

```
1          -2.821505   -2.539785      3.274194       5.343011  …
2           0.246429    0.496429      3.292857       8.661786  …
3           0.882796    1.151613      3.040860      12.818602  …
4           8.921111    9.103333      3.116667      14.033333  …
5          14.684946   14.890323      3.392473      16.514301  …
6          18.572222   18.910000      3.116667      20.695333  …
7          22.233333   22.161290      4.025806      21.516667  …
8          20.909302   20.762791      4.647674      17.980581  …
9          17.597753   17.351685      3.795506      13.392697  …
10         12.068132   12.105495      4.059140       8.731319  …
11          3.012222    3.154444      3.783333       6.064778  …
12         -1.282796   -0.983871      3.348387       4.411613  …

          SOIL_MOISTURE_10_DAILY  SOIL_MOISTURE_20_DAILY  \
LST_DATE
1                       0.247762                0.198514
2                       0.247500                0.204087
3                       0.230386                0.192886
4                       0.218744                0.191944
5                       0.206613                0.181387
6                       0.145856                0.138689
7                       0.099527                0.094817
8                       0.143919                0.122547
9                       0.138101                0.116966
10                      0.174022                0.133286
11                      0.225822                0.189756
12                      0.231649                0.188172

          SOIL_MOISTURE_50_DAILY  SOIL_MOISTURE_100_DAILY  SOIL_TEMP_5_DAILY  \
LST_DATE
1                       0.154194                 0.174161           0.186022
2                       0.156690                 0.175976           0.779762
3                       0.157398                 0.174613           2.005376
4                       0.156344                 0.173100           9.504444
5                       0.150903                 0.170215          16.760215
6                       0.131656                 0.161329          21.883333
7                       0.116602                 0.143032          25.444086
8                       0.129826                 0.144393          24.310465
9                       0.124528                 0.148390          20.435955
10                      0.130615                 0.147568          15.015385
11                      0.158711                 0.172222           6.501111
12                      0.158742                 0.172882           1.944086

          SOIL_TEMP_10_DAILY  SOIL_TEMP_20_DAILY  SOIL_TEMP_50_DAILY  \
LST_DATE
1                   0.186022            0.649462            1.527957
2                   0.714286            0.847619            1.208333
```

```
3                    1.954839             1.987097             2.208602
4                    9.306667             8.537778             7.555556
5                   16.690323            15.579570            14.225806
6                   21.841111            20.652222            19.091111
7                   25.536559            24.307609            22.758696
8                   24.462791            23.770930            22.986047
9                   20.569663            20.329213            20.237079
10                  15.110989            15.319780            15.970330
11                   6.528889             7.393333             8.755556
12                   1.915054             2.551613             3.708602

          SOIL_TEMP_100_DAILY
LST_DATE
1                    2.986022 NaN
2                    2.076190 NaN
3                    2.760215 NaN
4                    6.393333 NaN
5                   12.348387 NaN
6                   17.082222 NaN
7                   20.911957 NaN
8                   22.015116 NaN
9                   20.147191 NaN
10                  16.851648 NaN
11                  10.811111 NaN
12                   5.580645 NaN

[12 rows x 27 columns]
```

Each row in this new dataframe respresents the average values for the months (1=January, 2=February, etc.)

We can apply more customized aggregations, as with any groupby operation. Below we keep the mean of the mean, max of the max, and min of the min for the temperature measurements.

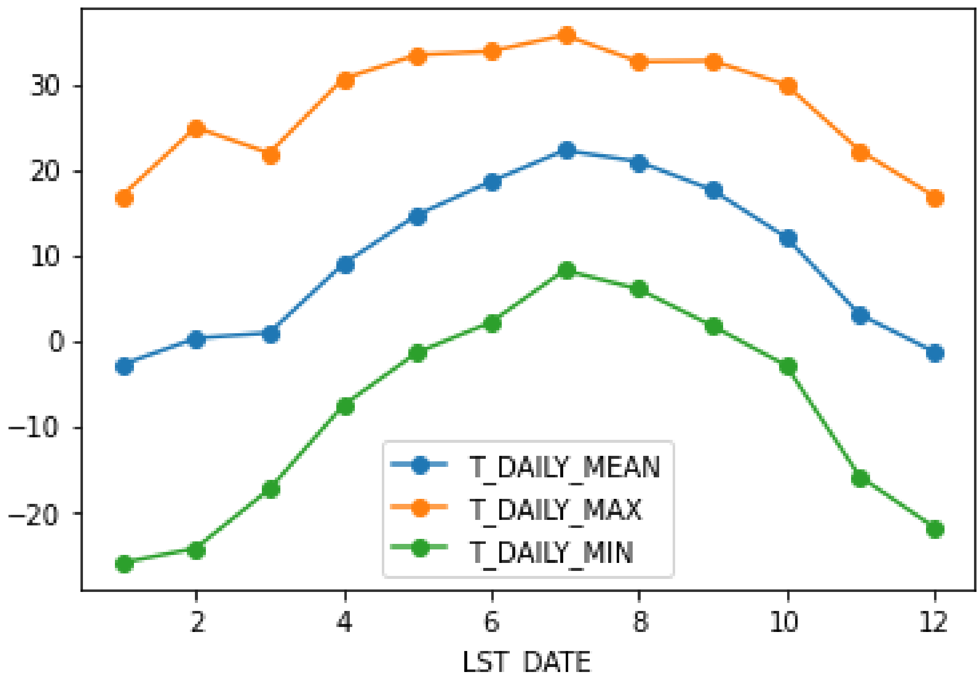
[24]:
```
monthly_T_climatology = df.groupby(df.index.month).aggregate({'T_DAILY_MEAN':␣
 ↪'mean',
                                                              'T_DAILY_MAX':␣
 ↪'max',
                                                              'T_DAILY_MIN':␣
 ↪'min'})
monthly_T_climatology.head()
```

[24]:
```
          T_DAILY_MEAN  T_DAILY_MAX  T_DAILY_MIN
LST_DATE
1            -2.821505         16.9        -26.0
2             0.246429         24.9        -24.3
3             0.882796         21.9        -17.4
4             8.921111         30.6         -7.6
```

```
5           14.684946        33.4        -1.4
```

[25]:
```
monthly_T_climatology.plot(marker='o')
```

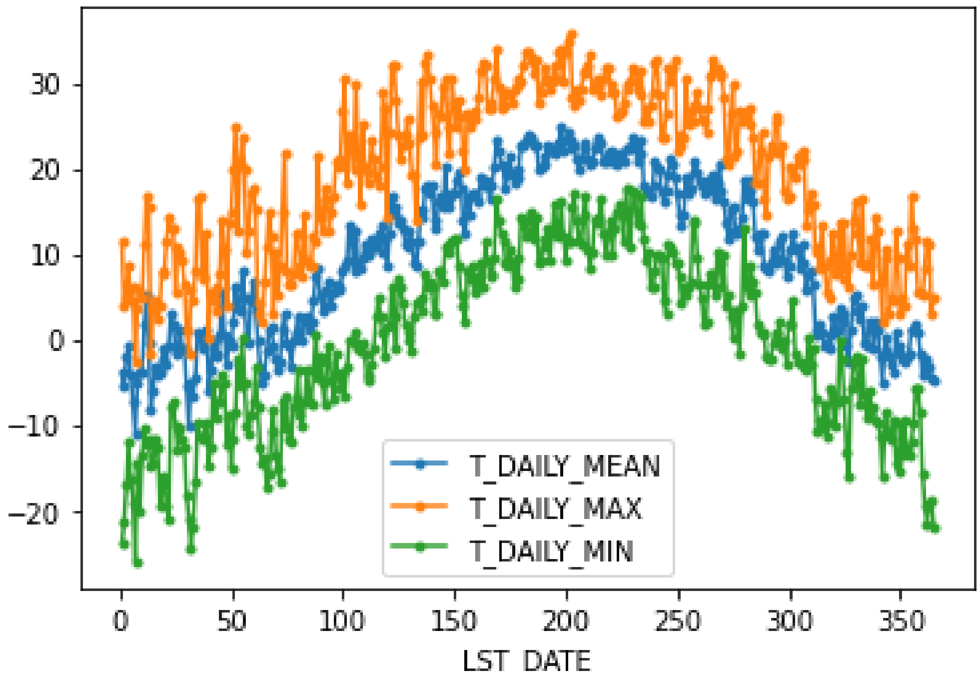
[25]:
```
<AxesSubplot:xlabel='LST_DATE'>
```



If we want to do it on a finer scale, we can group by day of year.

[26]:
```
daily_T_climatology = df.groupby(df.index.dayofyear).aggregate({'T_DAILY_MEAN':␣
↪'mean',
                                                                'T_DAILY_MAX':␣
↪'max',
                                                                'T_DAILY_MIN':␣
↪'min'})
daily_T_climatology.plot(marker='.')
```

[26]:
```
<AxesSubplot:xlabel='LST_DATE'>
```



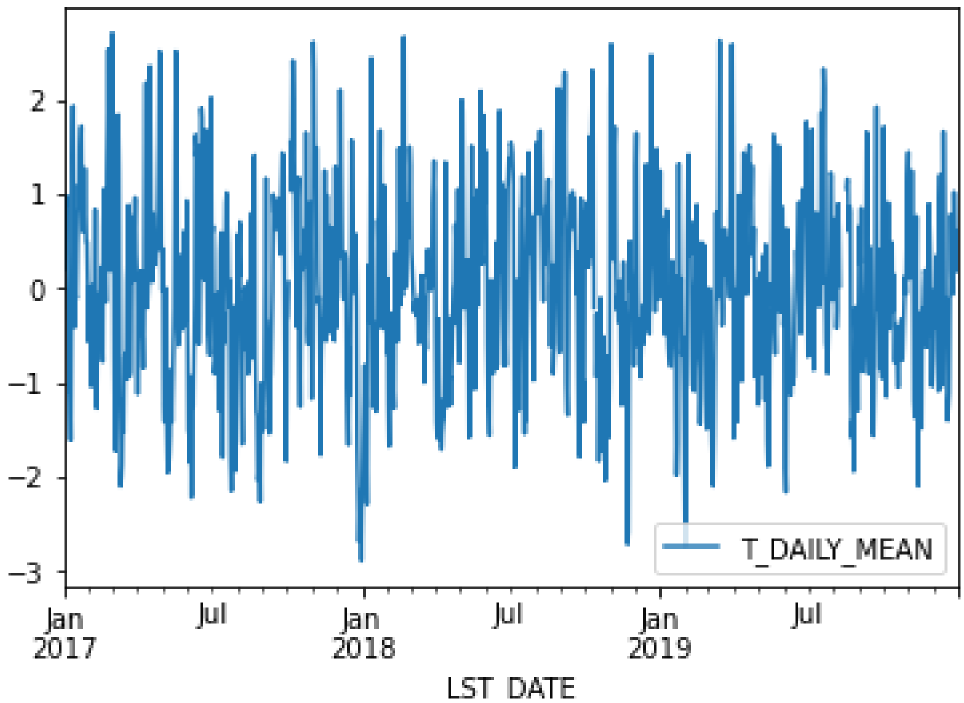
1.6.1 Calculating anomalies

A common mode of analysis in climate science is to remove the climatology from a signal to focus only on the "anomaly" values. This can be accomplished with transformation.

[27]:
```
def standardize(x):
    return (x - x.mean())/x.std()

anomaly = df.groupby(df.index.month).transform(standardize)
anomaly.plot(y='T_DAILY_MEAN')
```

```
<ipython-input-27-fd49119536d3>:4: FutureWarning: Dropping invalid columns in
DataFrameGroupBy.transform is deprecated. In a future version, a TypeError will
be raised. Before calling .transform, select only columns which should be valid
for the transforming function.
  anomaly = df.groupby(df.index.month).transform(standardize)
```

[27]: `<AxesSubplot:xlabel='LST_DATE'>`



2
1
0
−1
−2
−3
T_DAILY_MEAN
Jan
2017
Jul
Jan
2018
Jul
Jan
2019
Jul
LST_DATE