In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

A copy of this notebook. (pandas_indexing.ipynb)

Course Report

Display the `pd_course.csv` file.

In [2]:
```
with open('pd_course.csv') as f : print(f.read())
```

```
Name,Assignments,Midterm1,Midterm2,Final
Max Score,100,100,100,100
Weight,10,20,20,50
N Student,80,65,72,74
M Student,75,52,62,64
O Student,100,70,80,80
P Student,94,45,65,55
```

Course records can be handled with `pandas`. Read in a course record file and generate the total mark.

In [3]:
```
pd.read_csv('pd_course.csv')
```

Out[3]:

	Name	Assignments	Midterm1	Midterm2	Final
0	Max Score	100	100	100	100
1	Weight	10	20	20	50
2	N Student	80	65	72	74
3	M Student	75	52	62	64
4	O Student	100	70	80	80
5	P Student	94	45	65	55

Extract the max score and weight rows and drop these rows. `iloc` indexes the data frame by position.

In [4]:
```
course = pd.read_csv('pd_course.csv') # reread to ensure data
max_score = course.iloc[0] # use index location to reference row
weight = course.iloc[1]
print(max_score)
print()
print(weight)
course.drop( course.index[0:2], inplace=True)
course.index -= 2 # adjust the index values
course
```

```
Name           Max Score
Assignments          100
Midterm1             100
Midterm2             100
Final                100
Name: 0, dtype: object

Name           Weight
Assignments        10
Midterm1           20
Midterm2           20
Final              50
Name: 1, dtype: object
```

Out[4]:

	Name	Assignments	Midterm1	Midterm2	Final
0	N Student	80	65	72	74
1	M Student	75	52	62	64
2	O Student	100	70	80	80
3	P Student	94	45	65	55

Set each work product to a ratio between 0 and 1.

In [5]:
```
(course.iloc[:,1:] / max_score[1:]) # indexing is used to remove Name location
```

Out[5]:

	Assignments	Midterm1	Midterm2	Final
0	0.8	0.65	0.72	0.74
1	0.75	0.52	0.62	0.64
2	1	0.7	0.8	0.8
3	0.94	0.45	0.65	0.55

Multiply by weight.

In [6]:
```
(course.iloc[:,1:] / max_score[1:])*weight[1:]
```

Out[6]:

	Assignments	Midterm1	Midterm2	Final
0	8	13	14.4	37
1	7.5	10.4	12.4	32
2	10	14	16	40
3	9.4	9	13	27.5

Sum the rows (note the `axis` keyword).

In [7]:
```
((course.iloc[:,1:] / max_score[1:])*weight[1:]).sum(axis=1)
```

Out[7]:
```
0    72.4
1    62.3
2    80.0
3    58.9
dtype: float64
```

Create a new column with the total score.

In [8]:
```
course['Total'] = ((course.iloc[:,1:] / max_score[1:])*weight[1:]).sum(axis=1)
course
```

Out[8]:

	Name	Assignments	Midterm1	Midterm2	Final	Total
0	N Student	80	65	72	74	72.4
1	M Student	75	52	62	64	62.3
2	O Student	100	70	80	80	80.0
3	P Student	94	45	65	55	58.9

Sort by total mark.

In [9]:
```
course.sort_values('Total')
```

Out[9]:

	Name	Assignments	Midterm1	Midterm2	Final	Total
3	P Student	94	45	65	55	58.9
1	M Student	75	52	62	64	62.3
0	N Student	80	65	72	74	72.4
2	O Student	100	70	80	80	80.0

Reverse the order of sort.

In [10]:
```
course.sort_values('Total', ascending=False)
```

Out[10]:

	Name	Assignments	Midterm1	Midterm2	Final	Total
2	O Student	100	70	80	80	80.0
0	N Student	80	65	72	74	72.4
1	M Student	75	52	62	64	62.3
3	P Student	94	45	65	55	58.9

Sort by name.

In [11]:
```
course.sort_values('Name')
```

Out[11]:

	Name	Assignments	Midterm1	Midterm2	Final	Total
1	M Student	75	52	62	64	62.3
0	N Student	80	65	72	74	72.4
2	O Student	100	70	80	80	80.0
3	P Student	94	45	65	55	58.9

Find the minimum value in each column. Note the minimum is also found in the 'Name' column.

In [12]:
```
course.min()
```

Out[12]:
```
Name           M Student
Assignments           75
Midterm1              45
Midterm2              62
Final                 55
Total               58.9
dtype: object
```

Find the maximum.

In [13]:
```
course.max()
```

Out[13]:
```
Name           P Student
Assignments          100
Midterm1              70
Midterm2              80
Final                 80
Total                 80
dtype: object
```

The standard deviation is found with:

In [14]:
```
course.std()
```

Out[14]:
```
Assignments    11.701140
Midterm1       11.518102
Midterm2        8.015610
Final          11.026483
Total           9.626699
dtype: float64
```

Add these rows to the table and replace the name with the statistics name.

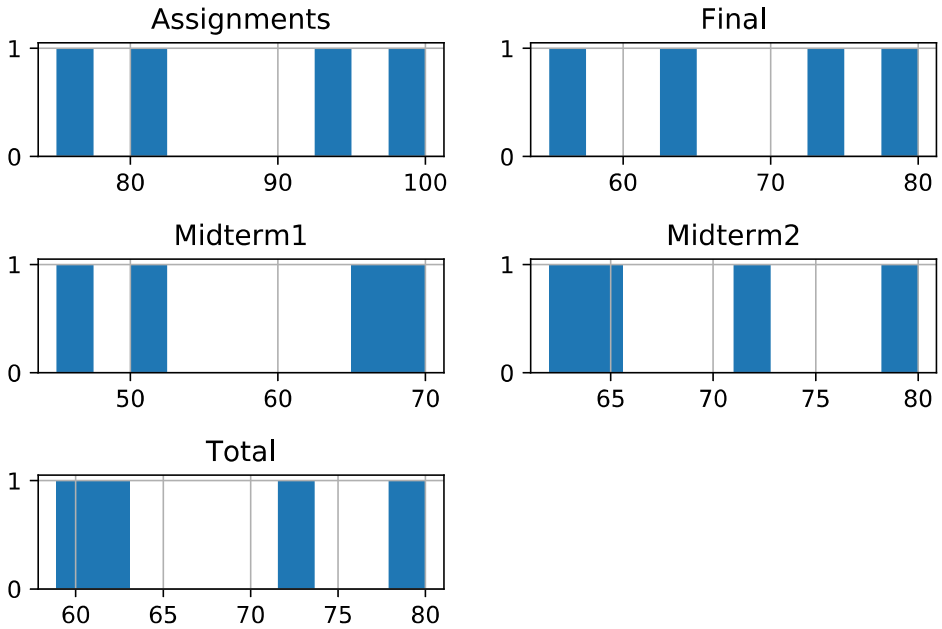
In [15]:
```
std_nm = course.std(); std_nm['Name'] = 'std'
min_nm = course.min(); min_nm['Name'] = 'min'
max_nm = course.max(); max_nm['Name'] = 'max'
course_stat = course.append([std_nm, min_nm, max_nm],ignore_index=True)
course_stat
```

Out[15]:

	Name	Assignments	Midterm1	Midterm2	Final	Total
0	N Student	80.00000	65.000000	72.00000	74.000000	72.400000
1	M Student	75.00000	52.000000	62.00000	64.000000	62.300000
2	O Student	100.00000	70.000000	80.00000	80.000000	80.000000
3	P Student	94.00000	45.000000	65.00000	55.000000	58.900000
4	std	11.70114	11.518102	8.01561	11.026483	9.626699
5	min	75.00000	45.000000	62.00000	55.000000	58.900000
6	max	100.00000	70.000000	80.00000	80.000000	80.000000

Plot a histogram, in general 4 data values is too few.

In [16]:
```
course.hist()
plt.tight_layout() #  nice layout
```



OECD Population

Population data from OECD Population (https://data.oecd.org/pop/population.htm).

The start of the CSV file is:

In [17]:
```
with open('DP_LIVE_10102017143725341.csv') as f :
    l = [ next(f) for _ in range(5)]
print( ''.join(l))
```

```
"LOCATION","INDICATOR","SUBJECT","MEASURE","FREQUENCY","TIME","Value","Flag Code
s"
"AUS","POP","TOT","AGRWTH","A","1957",2.270316,
"AUS","POP","TOT","AGRWTH","A","1958",2.095436,
"AUS","POP","TOT","AGRWTH","A","1959",2.174355,
"AUS","POP","TOT","AGRWTH","A","1960",2.177804,
```

Read in csv file and display the column headers. Each column can be treated as a series.

In [18]:
```
pop = pd.read_csv('DP_LIVE_10102017143725341.csv') # returns a DataFrame
print('type=',type(pop))
print(pop.columns)
```

```
type= <class 'pandas.core.frame.DataFrame'>
Index(['LOCATION', 'INDICATOR', 'SUBJECT', 'MEASURE', 'FREQUENCY', 'TIME',
       'Value', 'Flag Codes'],
      dtype='object')
```

Display the first 5 rows from the population dataset.

In [19]:
```
pop.head()
```

Out[19]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
0	AUS	POP	TOT	AGRWTH	A	1957	2.270316	NaN
1	AUS	POP	TOT	AGRWTH	A	1958	2.095436	NaN
2	AUS	POP	TOT	AGRWTH	A	1959	2.174355	NaN
3	AUS	POP	TOT	AGRWTH	A	1960	2.177804	NaN
4	AUS	POP	TOT	AGRWTH	A	1961	2.269586	NaN

The last eight rows contain:

In [20]:
```
pop.tail(8)
```

Out[20]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
9555	LVA	POP	WOMEN	MLN_PER	A	2005	1.212	NaN
9556	LVA	POP	WOMEN	MLN_PER	A	2006	1.200	NaN
9557	LVA	POP	WOMEN	MLN_PER	A	2007	1.190	NaN
9558	LVA	POP	WOMEN	MLN_PER	A	2008	1.177	NaN
9559	LVA	POP	WOMEN	MLN_PER	A	2009	1.160	NaN
9560	LVA	POP	WOMEN	MLN_PER	A	2010	1.138	NaN
9561	LVA	POP	WOMEN	MLN_PER	A	2011	1.118	NaN
9562	LVA	POP	WOMEN	MLN_PER	A	2012	1.104	NaN

Some of the columns contain description from a limited set describing the data on the row. The following uses the `python` sets to collect these descriptions. In general, large columns could result in large sets.

In [21]:
```
loc = set(pop['LOCATION'])
indicator = set(pop['INDICATOR'])
subject = set(pop['SUBJECT'])
measure = set(pop['MEASURE'])
freq = set(pop['FREQUENCY'])
time = set(pop['TIME'])
flags = set(pop['Flag Codes'])
print('loc:', loc)
print('indictor:', indicator)
print('subject:', subject)
print('measure:', measure)
print('freq:', freq)
print('time:', time)
print('flags:', flags)
```

```
loc: {'EST', 'NZL', 'RUS', 'FIN', 'CHE', 'EU28', 'AUS', 'ESP', 'IND', 'TUR', 'SW
E', 'USA', 'HUN', 'AUT', 'LUX', 'MEX', 'BRA', 'IDN', 'GRC', 'NLD', 'ZAF', 'POL',
'KOR', 'OECD', 'PRT', 'SVK', 'CHN', 'CHL', 'ISL', 'NOR', 'DEU', 'CZE', 'DNK', 'F
RA', 'IRL', 'LVA', 'JPN', 'BEL', 'ITA', 'GBR', 'CAN', 'ISR', 'COL', 'SVN'}
indictor: {'POP'}
subject: {'WOMEN', 'TOT', 'MEN'}
measure: {'MLN_PER', 'AGRWTH'}
freq: {'A'}
time: {1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1
962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 197
5, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988,
1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 20
02, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014}
flags: {nan, 'B'}
```

The `VALUE` column can contain `AGRWTH` annual growth as a percentage or `MLN_PER` population size in millions. The `LOCATION` column is the country or group of countries. The `SUBJECT` column identifies either total, men, or women in the population. The `FREQUENCY` column contains only the `A` value, which indicates annual. The `TIME` column contains a year from 1950 to 2014.

The first five rows for Canada is given by:

All the unique values can also be determined with:

```
In [22]: pop.LOCATION.unique() # notice the nice pretty printing
```

```
Out[22]: array(['AUS', 'AUT', 'BEL', 'CAN', 'CZE', 'DNK', 'FIN', 'FRA', 'DEU',
                'GRC', 'HUN', 'ISL', 'IRL', 'ITA', 'JPN', 'KOR', 'LUX', 'MEX',
                'NLD', 'NZL', 'NOR', 'POL', 'PRT', 'SVK', 'ESP', 'SWE', 'CHE',
                'TUR', 'GBR', 'USA', 'BRA', 'CHL', 'COL', 'EST', 'ISR', 'RUS',
                'SVN', 'CHN', 'IND', 'IDN', 'ZAF', 'EU28', 'OECD', 'LVA'], dtype=object)
```

Notice that pop['LOCATION'] is also used to access the column.

```
In [23]: pop['LOCATION'].unique()
```

```
Out[23]: array(['AUS', 'AUT', 'BEL', 'CAN', 'CZE', 'DNK', 'FIN', 'FRA', 'DEU',
                'GRC', 'HUN', 'ISL', 'IRL', 'ITA', 'JPN', 'KOR', 'LUX', 'MEX',
                'NLD', 'NZL', 'NOR', 'POL', 'PRT', 'SVK', 'ESP', 'SWE', 'CHE',
                'TUR', 'GBR', 'USA', 'BRA', 'CHL', 'COL', 'EST', 'ISR', 'RUS',
                'SVN', 'CHN', 'IND', 'IDN', 'ZAF', 'EU28', 'OECD', 'LVA'], dtype=object)
```

```
In [24]: pop.SUBJECT.unique()
```

```
Out[24]: array(['TOT', 'MEN', 'WOMEN'], dtype=object)
```

```
In [25]: pop[ pop.LOCATION == 'CAN'].head() # same as boolean array selection in numpy
```

Out[25]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
730	CAN	POP	TOT	AGRWTH	A	1951	2.132820	NaN
731	CAN	POP	TOT	AGRWTH	A	1952	3.198771	NaN
732	CAN	POP	TOT	AGRWTH	A	1953	2.666486	NaN
733	CAN	POP	TOT	AGRWTH	A	1954	2.972973	NaN
734	CAN	POP	TOT	AGRWTH	A	1955	2.682287	NaN

The following expression returns a boolean series that is true if the `LOCATION` is `CAN`. This boolean series selects rows from the full DataFrame. Only the indices from 725 to 740 are displayed to show both True and False.

In [26]:
```
(pop.LOCATION == 'CAN')[725:740]
```

Out[26]:
```
725    False
726    False
727    False
728    False
729    False
730     True
731     True
732     True
733     True
734     True
735     True
736     True
737     True
738     True
739     True
Name: LOCATION, dtype: bool
```

Boolean series can be combined to further refine the selected rows. In this case the rows of women population from Canada are selected. The parenthesis are required since & has higher precedence than ==.

In [27]:
```
can_w = pop[ (pop.LOCATION == 'CAN') & (pop.SUBJECT == 'WOMEN') &
               (pop.MEASURE == 'MLN_PER')]
can_w.head()
```

Out[27]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
924	CAN	POP	WOMEN	MLN_PER	A	1950	6.889	NaN
925	CAN	POP	WOMEN	MLN_PER	A	1951	7.055	NaN
926	CAN	POP	WOMEN	MLN_PER	A	1952	7.271	NaN
927	CAN	POP	WOMEN	MLN_PER	A	1953	7.461	NaN
928	CAN	POP	WOMEN	MLN_PER	A	1954	7.679	NaN

The table rows can also be selected by their index. `can_w` only contains part of `pop` for Canada women in millions.

In [28]:
```
pop[922:930] # assumes the index is a sequence 0 to N
```

Out[28]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
922	CAN	POP	TOT	MLN_PER	A	2013	35.154	NaN
923	CAN	POP	TOT	MLN_PER	A	2014	35.540	NaN
924	CAN	POP	WOMEN	MLN_PER	A	1950	6.889	NaN
925	CAN	POP	WOMEN	MLN_PER	A	1951	7.055	NaN
926	CAN	POP	WOMEN	MLN_PER	A	1952	7.271	NaN
927	CAN	POP	WOMEN	MLN_PER	A	1953	7.461	NaN
928	CAN	POP	WOMEN	MLN_PER	A	1954	7.679	NaN
929	CAN	POP	WOMEN	MLN_PER	A	1955	7.888	NaN

The index in `can_w` is the index from `pop`.

```
In [29]: can_w[0:5]
```

Out[29]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
924	CAN	POP	WOMEN	MLN_PER	A	1950	6.889	NaN
925	CAN	POP	WOMEN	MLN_PER	A	1951	7.055	NaN
926	CAN	POP	WOMEN	MLN_PER	A	1952	7.271	NaN
927	CAN	POP	WOMEN	MLN_PER	A	1953	7.461	NaN
928	CAN	POP	WOMEN	MLN_PER	A	1954	7.679	NaN

These rows can be indexed to give only the rows from 1970 to 1980.

```
In [30]: can_w[ (can_w.TIME >= 1970) & (can_w.TIME <= 1980) ]
```

Out[30]:

	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value	Flag Codes
944	CAN	POP	WOMEN	MLN_PER	A	1970	10.813	NaN
945	CAN	POP	WOMEN	MLN_PER	A	1971	10.936	NaN
946	CAN	POP	WOMEN	MLN_PER	A	1972	11.075	NaN
947	CAN	POP	WOMEN	MLN_PER	A	1973	11.220	NaN
948	CAN	POP	WOMEN	MLN_PER	A	1974	11.385	NaN
949	CAN	POP	WOMEN	MLN_PER	A	1975	11.561	NaN
950	CAN	POP	WOMEN	MLN_PER	A	1976	11.726	NaN
951	CAN	POP	WOMEN	MLN_PER	A	1977	11.878	NaN
952	CAN	POP	WOMEN	MLN_PER	A	1978	12.011	NaN
953	CAN	POP	WOMEN	MLN_PER	A	1979	12.142	NaN
954	CAN	POP	WOMEN	MLN_PER	A	1980	12.308	NaN

The columns that are nolonger useful can be dropped with:

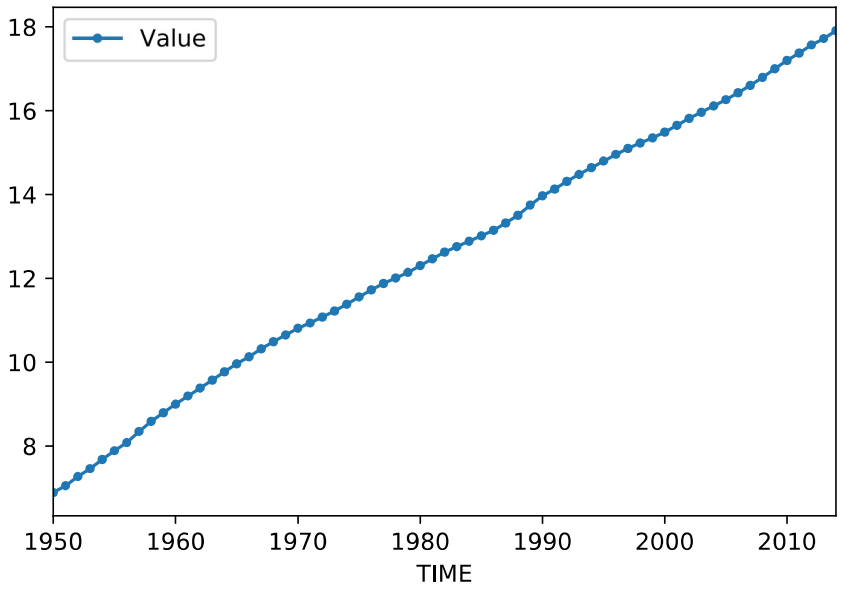
In [31]:
```
print('columns', can_w.columns)
w = can_w.drop(['LOCATION', 'INDICATOR', 'SUBJECT', 'MEASURE', 'FREQUENCY'], axis=1
)
# axis=1 specifies columns
print('before flag codes drop',w.columns)
w = w.drop('Flag Codes', axis=1)
w = w.set_index('TIME') # change the index to the time colum
print(w.columns)
w.loc[1955:1965] # index
```

```
columns Index(['LOCATION', 'INDICATOR', 'SUBJECT', 'MEASURE', 'FREQUENCY', 'TIME
',
       'Value', 'Flag Codes'],
      dtype='object')
before flag codes drop Index(['TIME', 'Value', 'Flag Codes'], dtype='object')
Index(['Value'], dtype='object')
```

Out[31]:

	Value
TIME	
1955	7.888
1956	8.081
1957	8.347
1958	8.590
1959	8.793
1960	8.997
1961	9.194
1962	9.384
1963	9.574
1964	9.771
1965	9.963

In [32]:
```
w.plot( style='.-'); None
```



Select the Canadian male population and change the index to `TIME`.

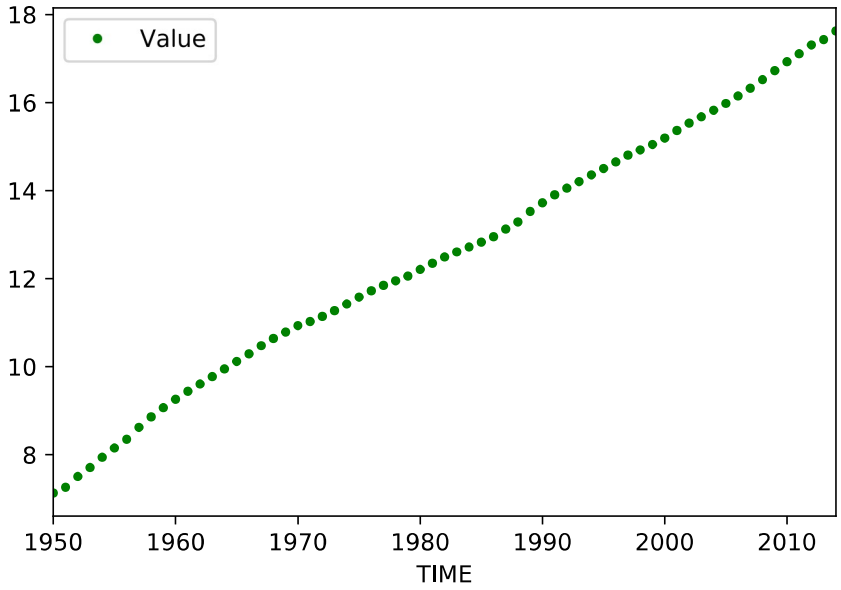
In [33]:
```
can_m = pop[ (pop.LOCATION == 'CAN') & (pop.SUBJECT == 'MEN')
                & (pop.MEASURE == 'MLN_PER')]
m = can_m.set_index('TIME')
m.columns
```

Out[33]:
```
Index(['LOCATION', 'INDICATOR', 'SUBJECT', 'MEASURE', 'FREQUENCY', 'Value',
       'Flag Codes'],
      dtype='object')
```

Notice that only the numeric columns are plotted.

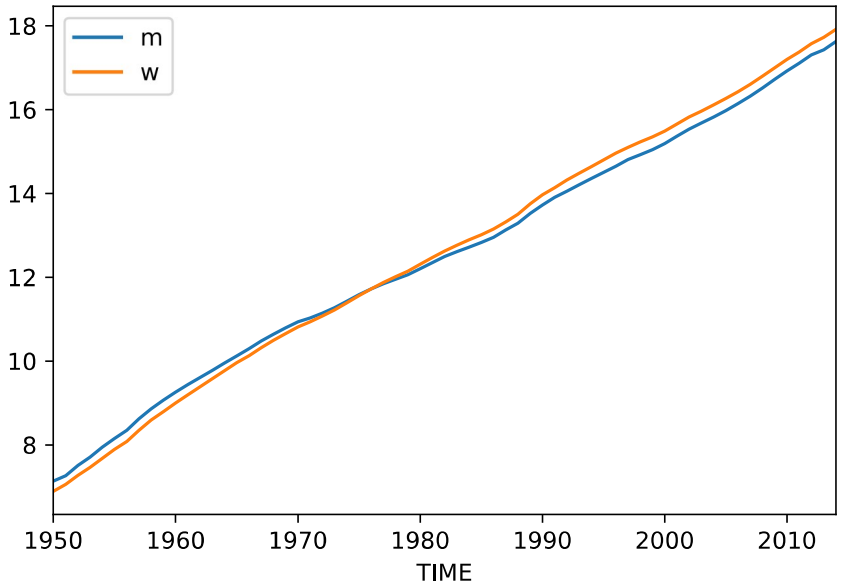
In [34]:
```
m.plot(style='g.')
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab7b20e2e8>
```



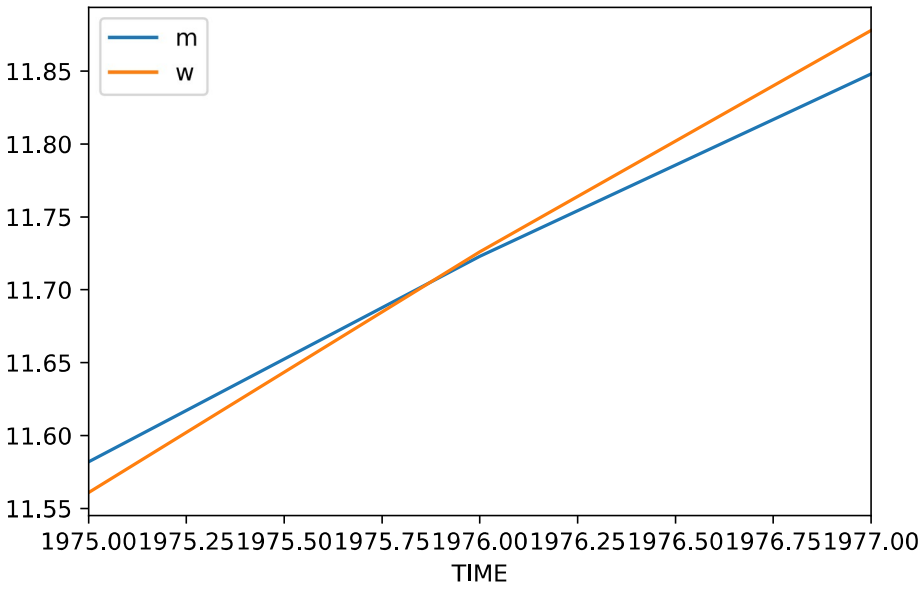
The men and women populations can be copied to a new `DataFrame`. `m.Value` is a `pd.Series`. So is `w.Value`. The index is used to align the values. Thus `TIME` as an index must be used.

In [35]:
```
mw = pd.DataFrame( { 'm' : m.Value, 'w' : w.Value })
mw.plot()
```
None



Zooming in using location index.

In [36]:
```
mw.loc[1975:1977].plot()
```
None



Does the above plot show anything interesting.

The following shows the first entries where there was fewer men than women.

In [37]:
```
mw[(mw.m - mw.w) < 0.0].head() # find the first values where men pop is less then women pop
```

Out[37]:

	m	w
TIME		
1976	11.723	11.726
1977	11.848	11.878
1978	11.952	12.011
1979	12.059	12.142
1980	12.208	12.308

A column containing the difference is added with:

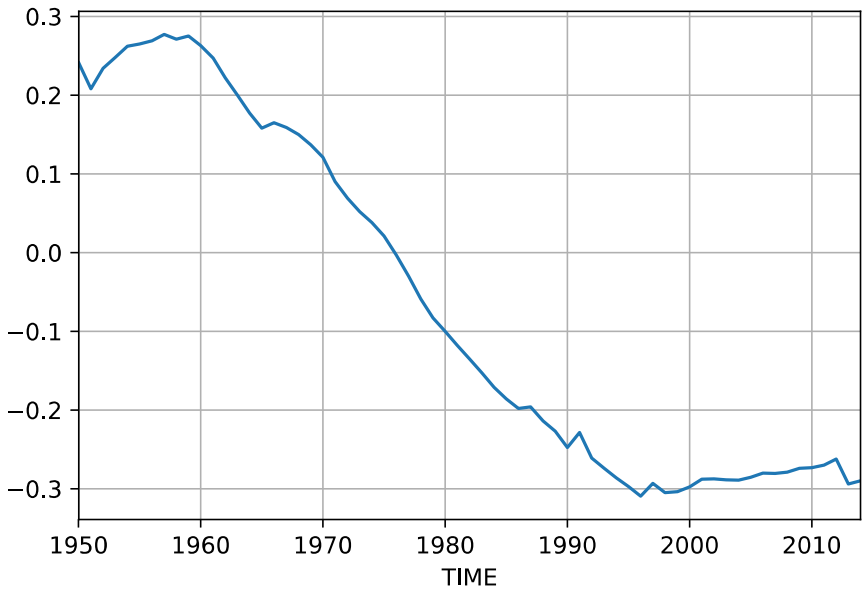
In [38]:
```
mw['diff'] = mw.m - mw.w
mw.loc[1974:1977]
```

Out[38]:

	m	w	diff
TIME			
1974	11.423	11.385	0.038
1975	11.582	11.561	0.021
1976	11.723	11.726	-0.003
1977	11.848	11.878	-0.030

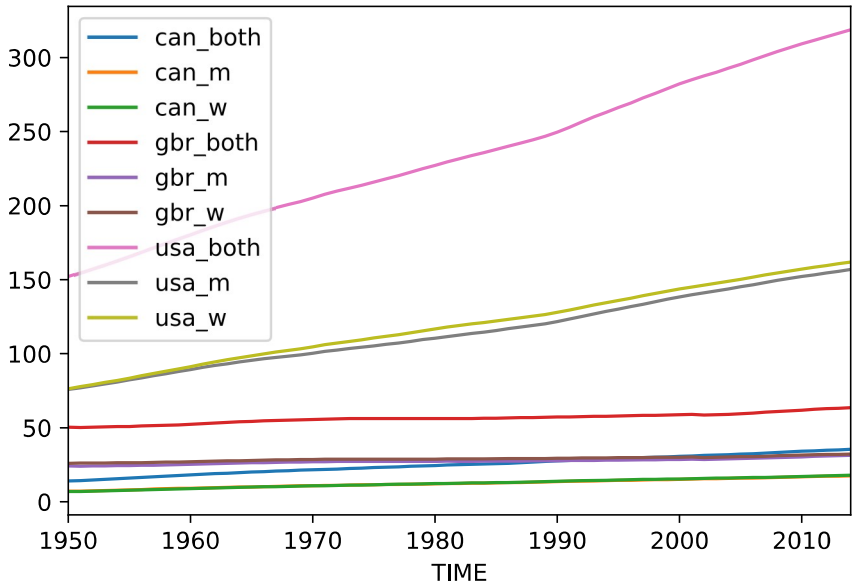
A plot off the difference shows its changes over time.

In [39]:
```
mw['diff'].plot(grid=True) # turing on the grid helps
None
```



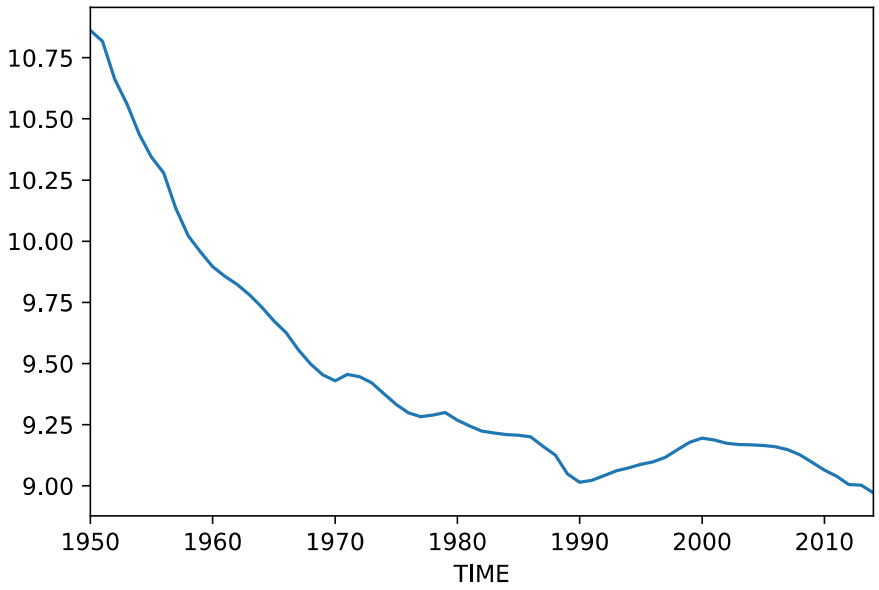
When pandas data frame columns are added together, the index is used to align them.

In [40]:
```
def get_men_women( country, pop ) :
    c = pop[ (pop.LOCATION == country ) ]
    c_men = c[ (c.SUBJECT == 'MEN') & (c.MEASURE == 'MLN_PER') ].Value
    c_women = c[ (c.SUBJECT == 'WOMEN') & (c.MEASURE == 'MLN_PER') ].Value
    return (c_men, c_women, c_men+c_women)

tpop = pop.set_index('TIME')
can_men, can_women, can_both = get_men_women( 'CAN', tpop)
usa_men, usa_women, usa_both = get_men_women( 'USA', tpop)
gbr_men, gbr_women, gbr_both = get_men_women( 'GBR', tpop)

comp = pd.DataFrame(
    {'can_m' : can_men,
     'can_w' : can_women,
     'can_both' : can_both,
     'usa_m' : usa_men,
     'usa_w' : usa_women,
     'usa_both' : usa_both,
     'gbr_m' : gbr_men,
     'gbr_w' : gbr_women,
     'gbr_both' : gbr_both
    } )
comp.plot()
None
```



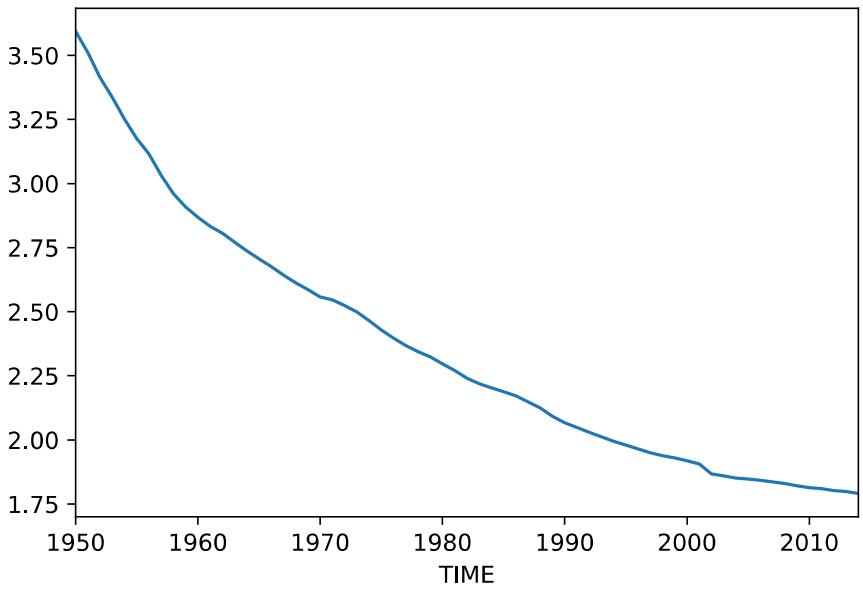
Ratio of USA to CAN population.

In [41]:
```
(comp.usa_both/comp.can_both).plot()
```
None



Ratio of GBR to CAN population.

In [42]:
```
(comp.gbr_both/comp.can_both).plot()
```
None



Daily weather data

In [43]:
```
with open('eng-daily-01012017-12312017.csv') as f :
    l = [ next(f) for _ in range(25)]
print( ''.join(l))
```

```
"Station Name","ST. JOHN'S INTL A"
"Province","NEWFOUNDLAND"
"Latitude","47.62"
"Longitude","-52.75"
"Elevation","140.50"
"Climate Identifier","8403505"
"WMO Identifier","71801"
"TC Identifier","YYT"

"Legend"
"A","Accumulated"
"C","Precipitation occurred, amount uncertain"
"E","Estimated"
"F","Accumulated and estimated"
"L","Precipitation may or may not have occurred"
"M","Missing"
"N","Temperature missing but known to be > 0"
"S","More than one occurrence"
"T","Trace"
"Y","Temperature missing but known to be < 0"
"[empty]","No data available"
"^","The value displayed is based on incomplete data"
"†","Data for this day has undergone only basic quality checking"
"‡","Partner data that is not subject to review by the National Climate Archives
"
```

Look at the daily weather data.

In [44]:
```
w = pd.read_csv('eng-daily-01012017-12312017.csv',skiprows=25,index_col=0,parse_dat
es=['Date/Time'])
w.drop( ['Year','Month','Day'], axis=1, inplace=True)
print(w.columns)
w.head()
```

```
Index(['Data Quality', 'Max Temp (°C)', 'Max Temp Flag', 'Min Temp (°C)',
       'Min Temp Flag', 'Mean Temp (°C)', 'Mean Temp Flag',
       'Heat Deg Days (°C)', 'Heat Deg Days Flag', 'Cool Deg Days (°C)',
       'Cool Deg Days Flag', 'Total Rain (mm)', 'Total Rain Flag',
       'Total Snow (cm)', 'Total Snow Flag', 'Total Precip (mm)',
       'Total Precip Flag', 'Snow on Grnd (cm)', 'Snow on Grnd Flag',
       'Dir of Max Gust (10s deg)', 'Dir of Max Gust Flag',
       'Spd of Max Gust (km/h)', 'Spd of Max Gust Flag'],
      dtype='object')
```

Out[44]:

	Data Quality	Max Temp (°C)	Max Temp Flag	Min Temp (°C)	Min Temp Flag	Mean Temp (°C)	Mean Temp Flag	Heat Deg Days (°C)	Heat Deg Days Flag	Cool Deg Days (°C)	...	Total Snow (cm)	Total Snow Flag	Total Prec (mm)
Date/Time														
2017-01-01	‡	-0.7	NaN	-4.3	NaN	-2.5	NaN	20.5	NaN	0.0	...	0.0	T	0.0
2017-01-02	‡	-0.2	NaN	-1.8	NaN	-1.0	NaN	19.0	NaN	0.0	...	26.0	NaN	19.9
2017-01-03	‡	-1.3	NaN	-5.0	NaN	-3.2	NaN	21.2	NaN	0.0	...	0.0	T	0.0
2017-01-04	‡	0.0	NaN	-4.9	NaN	-2.5	NaN	20.5	NaN	0.0	...	0.4	NaN	3.2
2017-01-05	‡	7.0	NaN	-0.2	NaN	3.4	NaN	14.6	NaN	0.0	...	0.0	NaN	8.2

5 rows × 23 columns

Check the values of the 'Max Temp Flag' column.

In [45]:
```
w[w.columns[2]].unique()
```

Out[45]:
```
array([nan, 'M'], dtype=object)
```

The start of the `eng-daily-01012017-12312017.csv` file indicates that 'M' is missing.

In [46]:
```
w[w[w.columns[2]] == 'M' ]
```

Out[46]:

	Data Quality	Max Temp (°C)	Max Temp Flag	Min Temp (°C)	Min Temp Flag	Mean Temp (°C)	Mean Temp Flag	Heat Deg Days (°C)	Heat Deg Days Flag	Cool Deg Days (°C)	...	Total Snow (cm)	Total Snow Flag	Total Preci (mm)
Date/Time														
2017-03-11	‡	NaN	M	-6.5	E	NaN	M	NaN	M	NaN	...	11.2	NaN	9.0

1 rows × 23 columns

This row could be deleted, but `pandas` can also ignore NaN values.

In [47]:
```
w.columns
```

Out[47]:
```
Index(['Data Quality', 'Max Temp (°C)', 'Max Temp Flag', 'Min Temp (°C)',
       'Min Temp Flag', 'Mean Temp (°C)', 'Mean Temp Flag',
       'Heat Deg Days (°C)', 'Heat Deg Days Flag', 'Cool Deg Days (°C)',
       'Cool Deg Days Flag', 'Total Rain (mm)', 'Total Rain Flag',
       'Total Snow (cm)', 'Total Snow Flag', 'Total Precip (mm)',
       'Total Precip Flag', 'Snow on Grnd (cm)', 'Snow on Grnd Flag',
       'Dir of Max Gust (10s deg)', 'Dir of Max Gust Flag',
       'Spd of Max Gust (km/h)', 'Spd of Max Gust Flag'],
      dtype='object')
```

Checking the 'Min Temp Flag' flag show that there is no missing data.

In [48]:
```
w[w['Min Temp Flag'] == 'M' ]
```

Out[48]:

	Data Quality	Max Temp (°C)	Max Temp Flag	Min Temp (°C)	Min Temp Flag	Mean Temp (°C)	Mean Temp Flag	Heat Deg Days (°C)	Heat Deg Days Flag	Cool Deg Days (°C)	...	Total Snow (cm)	Total Snow Flag	Total Preci (mm)
Date/Time														

0 rows × 23 columns

Find all the column names with Flag.

In [49]:
```
[ n for n in w.columns if 'Flag' in n]
```

Out[49]:
```
['Max Temp Flag',
 'Min Temp Flag',
 'Mean Temp Flag',
 'Heat Deg Days Flag',
 'Cool Deg Days Flag',
 'Total Rain Flag',
 'Total Snow Flag',
 'Total Precip Flag',
 'Snow on Grnd Flag',
 'Dir of Max Gust Flag',
 'Spd of Max Gust Flag']
```

Drop these columns.

In [50]:
```
w.drop([ n for n in w.columns if 'Flag' in n], axis=1, inplace=True )
w.columns
```

Out[50]:
```
Index(['Data Quality', 'Max Temp (°C)', 'Min Temp (°C)', 'Mean Temp (°C)',
       'Heat Deg Days (°C)', 'Cool Deg Days (°C)', 'Total Rain (mm)',
       'Total Snow (cm)', 'Total Precip (mm)', 'Snow on Grnd (cm)',
       'Dir of Max Gust (10s deg)', 'Spd of Max Gust (km/h)'],
      dtype='object')
```

What does 'Data Quality' contain.

In [51]:
```
w['Data Quality'].unique()
```

Out[51]:
```
array(['ǂ'], dtype=object)
```

Nothing interesting, so it too can be dropped.

In [52]:
```
w.drop( 'Data Quality', axis=1, inplace=True)
w.columns
```

Out[52]:
```
Index(['Max Temp (°C)', 'Min Temp (°C)', 'Mean Temp (°C)',
       'Heat Deg Days (°C)', 'Cool Deg Days (°C)', 'Total Rain (mm)',
       'Total Snow (cm)', 'Total Precip (mm)', 'Snow on Grnd (cm)',
       'Dir of Max Gust (10s deg)', 'Spd of Max Gust (km/h)'],
      dtype='object')
```

The resulting data frame is now much easier to look at.

In [53]:
```
w.head()
```

Out[53]:

	Max Temp (°C)	Min Temp (°C)	Mean Temp (°C)	Heat Deg Days (°C)	Cool Deg Days (°C)	Total Rain (mm)	Total Snow (cm)	Total Precip (mm)	Snow on Grnd (cm)	Dir of Max Gust (10s deg)	Spd of Max Gust (km/h)
Date/Time											
2017-01-01	-0.7	-4.3	-2.5	20.5	0.0	0.0	0.0	0.0	4.0	16.0	61
2017-01-02	-0.2	-1.8	-1.0	19.0	0.0	0.1	26.0	19.9	7.0	15.0	67
2017-01-03	-1.3	-5.0	-3.2	21.2	0.0	0.0	0.0	0.0	30.0	33.0	59
2017-01-04	0.0	-4.9	-2.5	20.5	0.0	2.8	0.4	3.2	30.0	16.0	80
2017-01-05	7.0	-0.2	3.4	14.6	0.0	8.2	0.0	8.2	18.0	17.0	85

A check of the rows, shows that the data stops in August.

In [54]:
```
w['2017-07-25':'2017-08-03']
```

Out[54]:

	Max Temp (°C)	Min Temp (°C)	Mean Temp (°C)	Heat Deg Days (°C)	Cool Deg Days (°C)	Total Rain (mm)	Total Snow (cm)	Total Precip (mm)	Snow on Grnd (cm)	Dir of Max Gust (10s deg)	Spd of Max Gust (km/h)
Date/Time											
2017-07-25	15.3	10.2	12.8	5.2	0.0	10.8	0.0	10.8	NaN	17.0	33
2017-07-26	19.1	10.7	14.9	3.1	0.0	0.0	0.0	0.0	NaN	25.0	44
2017-07-27	24.9	11.5	18.2	0.0	0.2	0.0	0.0	0.0	NaN	28.0	54
2017-07-28	22.5	9.8	16.2	1.8	0.0	0.0	0.0	0.0	NaN	25.0	39
2017-07-29	20.5	9.6	15.1	2.9	0.0	1.4	0.0	1.4	NaN	26.0	30
2017-07-30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-07-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-08-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-08-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-08-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [55]:
```
w.drop(w['2017-07-30':].index, inplace=True)
w.tail()
```

Out[55]:

	Max Temp (°C)	Min Temp (°C)	Mean Temp (°C)	Heat Deg Days (°C)	Cool Deg Days (°C)	Total Rain (mm)	Total Snow (cm)	Total Precip (mm)	Snow on Grnd (cm)	Dir of Max Gust (10s deg)	Spd of Max Gust (km/h)
Date/Time											
2017-07-25	15.3	10.2	12.8	5.2	0.0	10.8	0.0	10.8	NaN	17.0	33
2017-07-26	19.1	10.7	14.9	3.1	0.0	0.0	0.0	0.0	NaN	25.0	44
2017-07-27	24.9	11.5	18.2	0.0	0.2	0.0	0.0	0.0	NaN	28.0	54
2017-07-28	22.5	9.8	16.2	1.8	0.0	0.0	0.0	0.0	NaN	25.0	39
2017-07-29	20.5	9.6	15.1	2.9	0.0	1.4	0.0	1.4	NaN	26.0	30

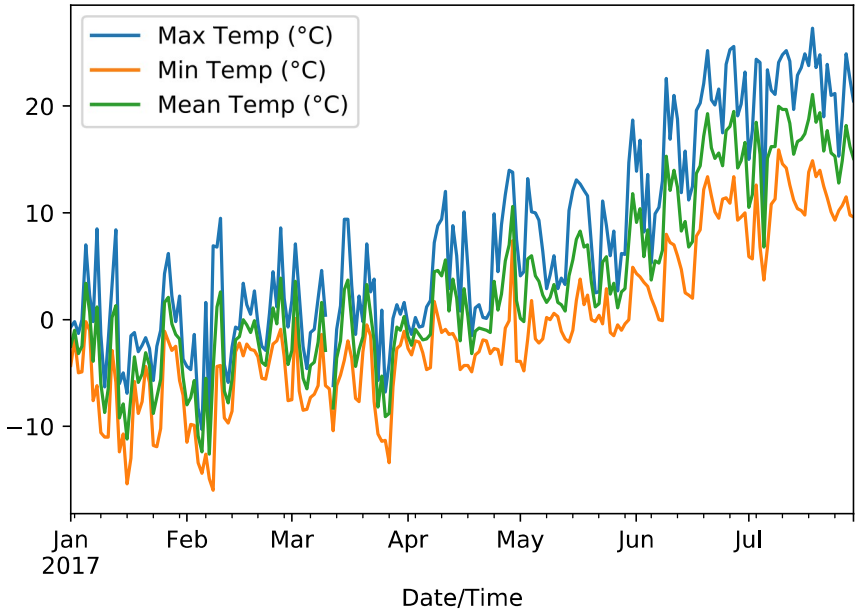
The **cleaned** data can now be saved with:

In [56]:
```
w.to_csv('w2017.csv')
with open('w2017.csv') as f :
    l = [ next(f) for _ in range(5)]
print( ''.join(l))
```

```
Date/Time,Max Temp (°C),Min Temp (°C),Mean Temp (°C),Heat Deg Days (°C),Cool Deg
Days (°C),Total Rain (mm),Total Snow (cm),Total Precip (mm),Snow on Grnd (cm),Di
r of Max Gust (10s deg),Spd of Max Gust (km/h)
2017-01-01,-0.7,-4.3,-2.5,20.5,0.0,0.0,0.0,0.0,4.0,16.0,61
2017-01-02,-0.2,-1.8,-1.0,19.0,0.0,0.1,26.0,19.9,7.0,15.0,67
2017-01-03,-1.3,-5.0,-3.2,21.2,0.0,0.0,0.0,0.0,30.0,33.0,59
2017-01-04,0.0,-4.9,-2.5,20.5,0.0,2.8,0.4,3.2,30.0,16.0,80
```

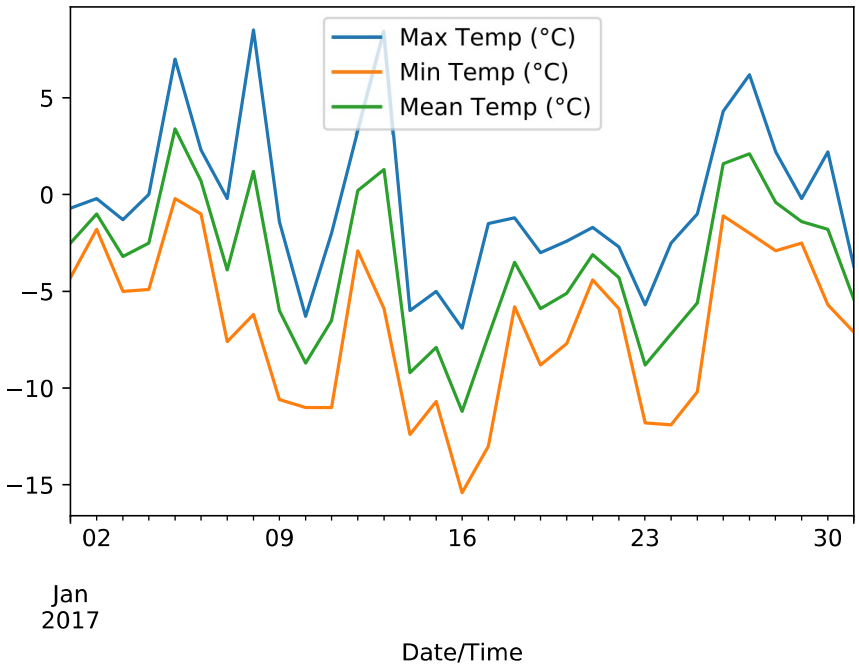
Plot the temperature data with:

In [57]:
```
temp_col = [ n for n in w.columns if 'Temp' in n] # get only the temperature columns
w[ temp_col ].plot(); None
```



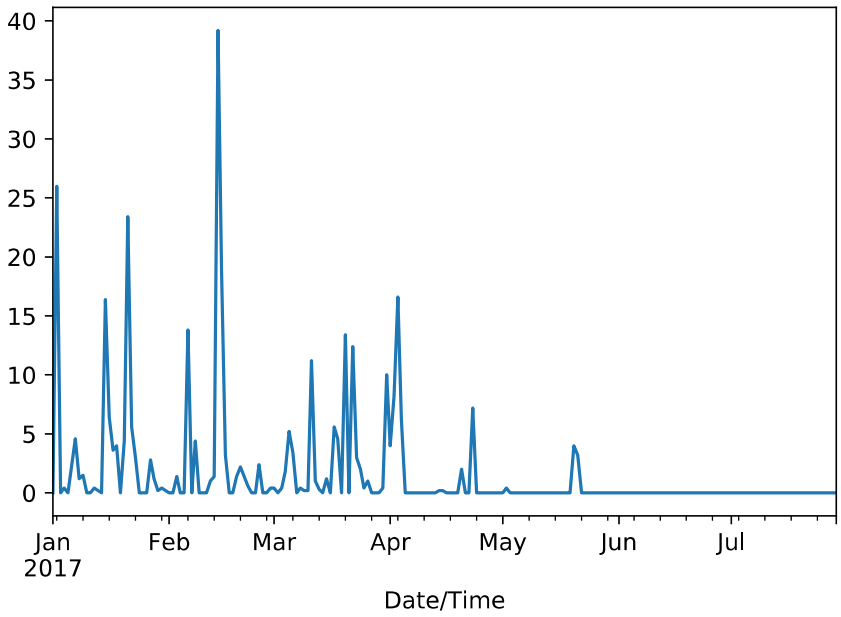
A plot of January data is:

In [58]:
```
w[ temp_col ]['2017-01'].plot(); None # what is happening to the legend?
```



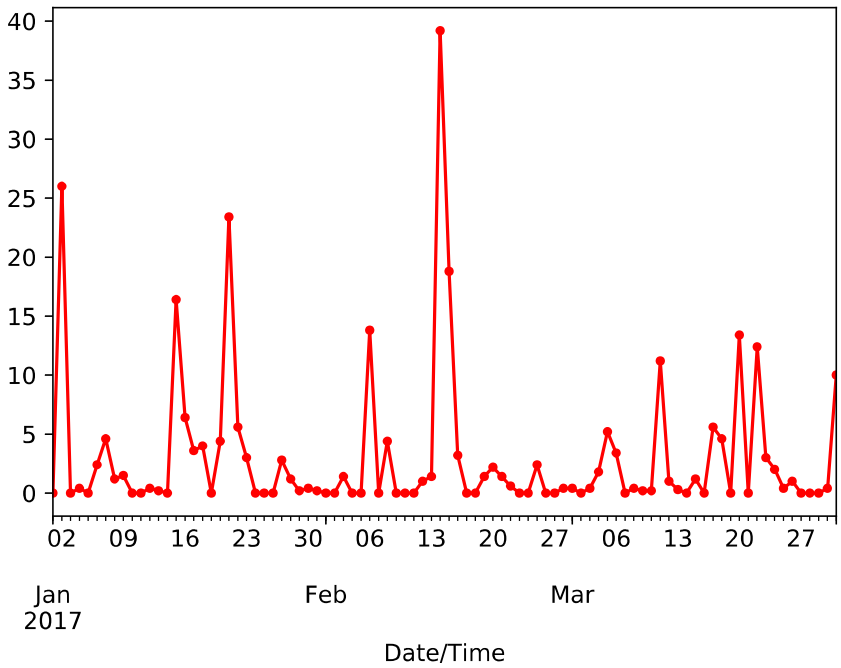
A plot of the snow fall is:

In [59]:
```
snow_col = 'Total Snow (cm)'
w[ snow_col ].plot(); None
```



Restricted to Janurary to March.

In [60]:
```
snow_col = 'Total Snow (cm)'
w[ snow_col ]['2017-01':'2017-03'].plot(style='.-r') # plot the samples
None
```



The nth largest snow fall days is given by:

In [61]:
```
w[ snow_col ].nlargest(10)
```

Out[61]:
```
Date/Time
2017-02-14    39.2
2017-01-02    26.0
2017-01-21    23.4
2017-02-15    18.8
2017-04-03    16.6
2017-01-15    16.4
2017-02-06    13.8
2017-03-20    13.4
2017-03-22    12.4
2017-03-11    11.2
Name: Total Snow (cm), dtype: float64
```

All the records can be examined with:

In [62]:
```
w.nlargest(5, snow_col) # nlargest accepts column names
```

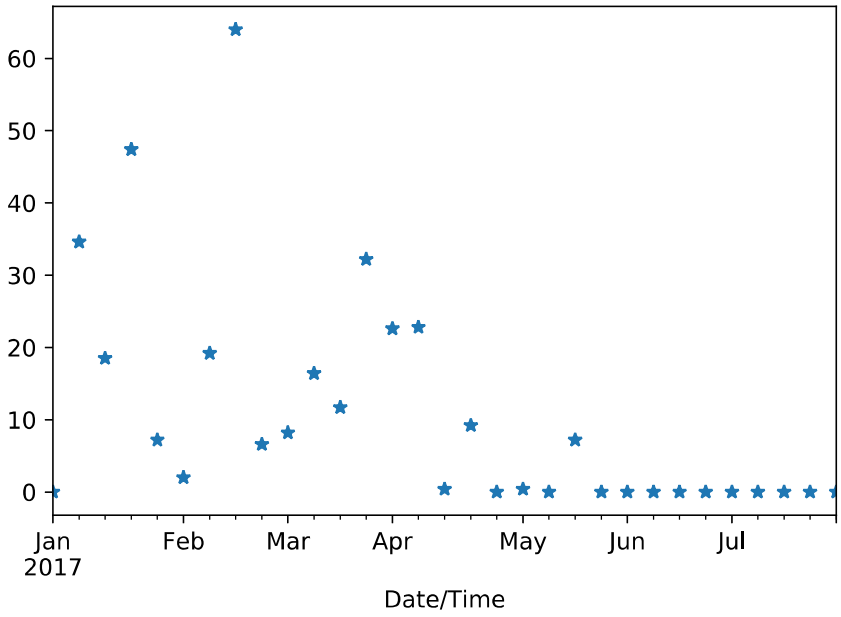
Out[62]:

	Max Temp (°C)	Min Temp (°C)	Mean Temp (°C)	Heat Deg Days (°C)	Cool Deg Days (°C)	Total Rain (mm)	Total Snow (cm)	Total Precip (mm)	Snow on Grnd (cm)	Dir of Max Gust (10s deg)	Spd of Max Gust (km/h)
Date/Time											
2017-02-14	-0.7	-3.0	-1.9	19.9	0.0	1.0	39.2	27.6	12.0	8.0	96
2017-01-02	-0.2	-1.8	-1.0	19.0	0.0	0.1	26.0	19.9	7.0	15.0	67
2017-01-21	-1.7	-4.4	-3.1	21.1	0.0	0.0	23.4	12.2	32.0	36.0	100
2017-02-15	-0.9	-2.2	-1.6	19.6	0.0	0.0	18.8	10.8	57.0	1.0	78
2017-04-03	0.2	-2.0	-0.9	18.9	0.0	11.7	16.6	24.7	29.0	5.0	89

Resampling by weeks allows the worst snow week to be found.

In [63]:
```
ww = w[ snow_col].resample('1W')
mww = ww.sum() # sum the snow fall over the week
mww.plot(style='*')
mww.nlargest(5)
```

Out[63]:
```
Date/Time
2017-02-19    64.0
2017-01-22    47.4
2017-01-08    34.6
2017-03-26    32.2
2017-04-09    22.8
Name: Total Snow (cm), dtype: float64
```



The rainest month is found with:

In [64]:
```
wettest_mon = w['Total Rain (mm)'].resample('1M').sum()
wettest_mon.sort_values(ascending=False)
```

Out[64]:
```
Date/Time
2017-06-30    104.8
2017-07-31     79.7
2017-03-31     76.8
2017-01-31     74.5
2017-04-30     59.3
2017-02-28     41.2
2017-05-31     38.4
Name: Total Rain (mm), dtype: float64
```

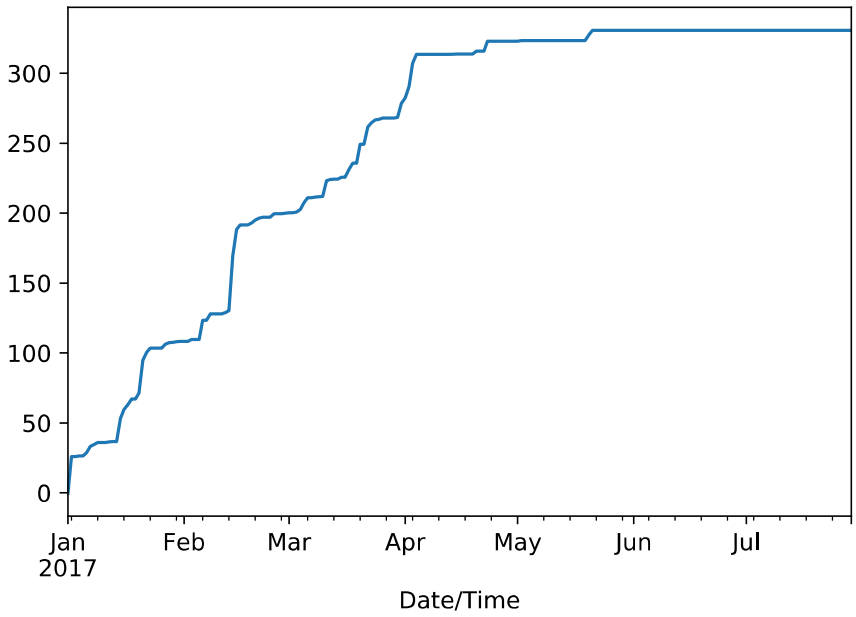
The month with the most precipation.

In [65]:
```
most_precip = w['Total Precip (mm)'].resample('1M').sum()
most_precip.sort_values(ascending=False)
```

Out[65]:
```
Date/Time
2017-01-31    158.1
2017-03-31    137.5
2017-06-30    104.8
2017-02-28    102.6
2017-04-30     88.7
2017-07-31     79.7
2017-05-31     45.2
Name: Total Precip (mm), dtype: float64
```

Cumulative snow fall is:

In [66]:
```
w[snow_col].cumsum().plot() # cumsum is the cumulative sum
None
```



In [67]:
```
cs = w[snow_col].cumsum() # integrate
cs.diff().plot(style='.-') # differentation
(w[snow_col]+15).plot(style='.-') # offset
None
```