In [95]:
```
import numpy as np
import matplotlib.pyplot as pp
import seaborn # extension to matplotlib but makes plots look prettier
```

Using function urllib.request.urlretrieve to download file from python standard module urllib

In this function,the first argument is link and second argument is local name of file

In [97]:
```
import urllib.request
urllib.request.urlretrieve("ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt",'stations.txt')
```

Out[97]:
```
('stations.txt', <email.message.Message at 0x1cb483bc400>)
```

The file has been downloaded. This function works in Python 3

We open the file in read mode and slice the first 10 lines of the file in the list

In [100]:
```
open('stations.txt','r').readlines()[:10]
```

Out[100]:
```
['ACW00011604  17.1167  -61.7833   10.1    ST JOHNS COOLIDGE FLD                    \n',
 'ACW00011647  17.1333  -61.7833   19.2    ST JOHNS                                 \n',
 'AE000041196  25.3330   55.5170   34.0    SHARJAH INTER. AIRP            GSN     41196\n',
 'AEM00041194  25.2550   55.3640   10.4    DUBAI INTL                             41194\n',
 'AEM00041217  24.4330   54.6510   26.8    ABU DHABI INTL                         41217\n',
 'AEM00041218  24.2620   55.6090  264.9    AL AIN INTL                            41218\n',
 'AF000040930  35.3170   69.0170 3366.0    NORTH-SALANG                   GSN     40930\n',
 'AFM00040938  34.2100   62.2280  977.2    HERAT                                  40938\n',
 'AFM00040948  34.5660   69.2120 1791.3    KABUL INTL                             40948\n',
 'AFM00040990  31.5000   65.8500 1010.0    KANDAHAR AIRPORT                       40990\n']
```

Details of the file: Station Code, geographic location(long and lati),sea level height in meters, station names, and some stations are tagged with GSN(GCOS Surface Network is a flag that indicates whether the station is part of the GCOS or not)

Lets gather some data. What we do is, we'll read all the lines and we skip those that do not that GSN. We collect the station names in the dictionary indexed by the station code(the first column)

Note: In python, it is possible to iterate through an open file, which returns the lines one by one

We check that if the string GSN is included in the line, if it exists, we take the line, split it, and assign the resulting field to python list. We will then use the first item in the list as the key for our dictionary and fifth and all following items, for name of stations using slicing operator. We join those strings using a space betwee them

In [103]:
```
stations = {}

for line in open('stations.txt','r'):
    if 'GSN' in line:
        fields =line.split()

        stations[fields[0]] = ' '.join(fields[4:])
```

How many Stations ?

In [105]:
```
len(stations)
```

Out[105]:
```
994
```

Lets take a look at few stations

In [220]:
```
{k: stations[k] for k in stations.keys() & {'AE000041196', 'ASN00001019', 'ASN00009789','CHM00053614','DAM00006011','EG000062414
                                            'GM000010962','IC000004013', 'KT000096995','LO000011934','USW00022536',
                                             'USW00023188','USW00014922','RSM00030710'}}
```

Out[220]:
```
{'USW00014922': 'MN MINNEAPOLIS/ST PAUL AP GSN HCN 72658',
 'USW00023188': 'CA SAN DIEGO LINDBERGH FLD GSN 72290',
 'LO000011934': 'POPRAD/TATRY GSN 11934',
 'IC000004013': 'STYKKISHOLMUR GSN 04013',
 'ASN00001019': 'KALUMBURU GSN 94100',
 'USW00022536': 'HI LIHUE WSO AP 1020.1 GSN 91165',
 'EG000062414': 'ASSWAN GSN 62414',
 'AE000041196': 'SHARJAH INTER. AIRP GSN 41196',
 'GM000010962': 'HOHENPEISSENBERG GSN 10962',
 'KT000096995': 'CHRISTMAS ISLAND AE GSN 96995',
 'DAM00006011': 'TORSHAVN GSN 06011',
 'RSM00030710': 'IRKUTSK GSN 30710',
 'CHM00053614': 'YINCHUAN GSN 53614',
 'ASN00009789': 'ESPERANCE GSN 94638'}
```

Lets take few and work on them

For this, we'll write a function that helps us look for interesting patterns in the station names. We'll call it findStation

We'll build a dictionary using a compreshension of station codes and names where the pattern that we are interested in is found within the name and then we print

In [108]:
```
def findStation(s):
    found = {code: name for code, name in stations.items() if s in name}
    print(found)
```

In [109]:
```
findStation('LIHUE')
```

```
{'USW00022536': 'HI LIHUE WSO AP 1020.1 GSN 91165'}
```

In [110]:
```
findStation('SAN DIEGO')
```

```
{'USW00023188': 'CA SAN DIEGO LINDBERGH FLD GSN 72290'}
```

In [111]:
```
findStation("MINNEAPOLIS")
```

```
{'USW00014922': 'MN MINNEAPOLIS/ST PAUL AP GSN HCN 72658'}
```

In [112]:
```
findStation('IRKUTSK')
```

```
{'RSM00030710': 'IRKUTSK GSN 30710'}
```

We'll focus on data of these four stations in this weather project

I will collect the code names of these station in the python list

In [114]:
```
dataStations = ['USW00022536','USW00023188','USW00014922','RSM00030710']
```

Now we'll load the temperature data from the noaa repository

I have downloaded the daily weather files for the above stations from the noaa repository

Lets look at the Li Hue

In [118]:
```
open('USW00022536.dly','r').readlines()[:10]
```

Out[118]:
```
['USW00022536195002TMAX  256  0  256  0  256  0  267  0  217  0  228  0  256  0  272  0  256  0  256  0  256  0  244  0  256  0
256  0  244  0  244  0  250  0  256  0  239  0  250  0  256  0  256  0  267  0  261  0  267  0  267  0  261  0  261  0-9999   -
9999   -9999   \n',
 'USW00022536195002TMIN  178  0  156  0  161  0  167  0  167  0  167  0  189  0  211  0  206  0  217  0  217  0  211  0  200  0
200  0  206  0  183  0  206  0  206  0  206  0  194  0  206  0  200  0  206  0  200  0  211  0  183  0  172  0  200  0-9999   -
9999   -9999   \n',
 'USW00022536195002PRCP    0  0    0  0    0  0    0  0  737  0  406  0   36  0   38  0    0T 0    0T 0    0  0    0T 0   18  0
5  0   10  0   18  0   15  0    5  0    0T 0    0T 0   23  0   10  0    3  0   48  0    0T 0    0T 0    0T 0    5  0-9999   -99
99   -9999   \n',
 'USW00022536195002SNOW    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0
0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0-9999   -99
99   -9999   \n',
 'USW00022536195002SNWD    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0
0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0    0  0-9999   -99
99   -9999   \n',
 'USW00022536195002WT03-9999   -9999   -9999   -9999       1  0-9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999
-9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999   -9999
-9999   -9999   \n',
 'USW00022536195002WT16-9999   -9999   -9999   -9999       1  X    1  X    1  X    1  X    1  X    1  X-9999       1  X    1  X
1  X    1  X    1  X    1  X    1  X    1  X    1  X    1  X    1  X    1  X-9999   -99
99   -9999   \n',
 'USW00022536195003TMAX  261  0  261  0  272  0  278  0  250  0  233  0  256  0  272  0  233  0  233  0  239  0  244  0  256  0
261  0  256  0  261  0  272  0  261  0  267  0  244  0  256  0  261  0  250  0  256  0  261  0  256  0  256  0  250  0  244  0
239  0  222  0\n',
 'USW00022536195003TMIN  211  0  172  0  144  0  139  0  156  0  178  0  200  0  156  0  172  0  161  0  178  0  189  0  189  0
200  0  189  0  183  0  200  0  194  0  200  0  194  0  206  0  200  0  194  0  189  0  200  0  194  0  194  0  189  0  178  0
189  0  161  0\n',
 'USW00022536195003PRCP    0T 0    3  0    0  0    0T 0  135  0    0T 0    0  0   41  0  876  0   30  0   20  0    0T 0    0T 0
10  0    0  0    0  0    0  0    3  0    0T 0   25  0    8  0    8  0   18  0   15  0    3  0   91  0   56  0   10  0    5  0
0T 0   61  0\n']
```

The given format of data is so messed up but we can see that it begins with the station code, followed by date, year and month and the weather observable such as temperature max or min, and then by bunch of numbers that represent the weather observable over days of month

I will start by defining a function to parse the file

Now with the help of numpy library, we read such file(np.genfromtxt)

The arguments need filename, the size of all fields, which columns we wish to keep,

The type of all the fields and the names we want to give to all the fields

The result which we wiil get is a numpy record array

In [121]:
```
def parseFile(filename):
    return np.genfromtxt(filename,
                         delimiter=dly_delimeter,
                         usecols=dly_usecols,
                         dtype=dly_dtype,
                         names = dly_names)
```

lets look at the columns and their data to understand how the data is organized in the above files

In [123]:
```
open('Columns Details.txt','r').readlines()
```

Out[123]:
```
['------------------------------\n',
 'Variable   Columns   Type\n',
 '------------------------------\n',
 'ID            1-11   Character\n',
 'YEAR         12-15   Integer\n',
 'MONTH        16-17   Integer\n',
 'ELEMENT      18-21   Character\n',
 'VALUE1       22-26   Integer\n',
 'MFLAG1       27-27   Character\n',
 'QFLAG1       28-28   Character\n',
 'SFLAG1       29-29   Character\n',
 'VALUE2       30-34   Integer\n',
 'MFLAG2       35-35   Character\n',
 'QFLAG2       36-36   Character\n',
 'SFLAG2       37-37   Character\n',
 '   .           .          .\n',
 '   .           .          .\n',
```

Now we will fill the arguments

length of the field: 11,4,2,2 represents (USW00022536195002TMAX) which is ID, year, month and element 5,1,1,1 represents Value1, MFlag1, QFlag1 and SFlag1 repreated 31 times(we'll not use flags)

In [125]:
```
dly_delimeter =[11,4,2,4] + [5,1,1,1]*31
```

Columns which we'll use are Year, month, elements which are 1,2,3 followed by all the values columns(Value1, Value2..)

In [ ]:
```
dly_usecols = [1,2,3] +[4*i for i in range(1,32)]
```

Data Types of columns are intergers, integers, string of 4 characters and followed by 31 integers

In [ ]:
```
dly_dtype = [np.int32, np.int32, (np.str_,4)] + [np.int32]*31
```

Names of the columns followed by number of days in month converted into strings

In [ ]:
```
dly_names = ['year','month','obs'] + [str(day) for day in range(1,31+1)]
```

In [126]:
```
lihue = parseFile('USW00022536.dly')
```

In [127]:
```
lihue
```

Out[127]:
```
array([(1950, 2, 'TMAX',   256,   256,   256,   267,   217,   228,   256,   272,   256,   256,   256,   244,   256,   256,   24
4,   244,   250,   256,   239,   250,   256,   256,   267,   261,   267,   267,   261,   261, -9999, -9999, -9999),
       (1950, 2, 'TMIN',   178,   156,   161,   167,   167,   167,   189,   211,   206,   217,   217,   211,   200,   200,   20
6,   183,   206,   206,   206,   194,   206,   200,   206,   200,   211,   183,   172,   200, -9999, -9999, -9999),
       (1950, 2, 'PRCP',     0,     0,     0,     0,   737,   406,    36,    38,     0,     0,     0,     0,    18,     5,    1
0,    18,    15,     5,     0,     0,    23,    10,     3,    48,     0,     0,     0,     5, -9999, -9999, -9999),
       ...,
       (2015, 9, 'WT03', -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,     1, -9999,     1, -9999, -9999, -999
9, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999),
       (2015, 9, 'WT08', -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,     1, -9999, -9999, -9999,
1, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,     1,     1, -9999, -9999, -9999, -9999, -9999, -9999),
       (2015, 9, 'WT10', -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,     1, -9999, -9999, -9999, -999
9, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999)],
      dtype=[('year', '<i4'), ('month', '<i4'), ('obs', '<U4'), ('1', '<i4'), ('2', '<i4'), ('3', '<i4'), ('4', '<i4'), ('5',
'<i4'), ('6', '<i4'), ('7', '<i4'), ('8', '<i4'), ('9', '<i4'), ('10', '<i4'), ('11', '<i4'), ('12', '<i4'), ('13', '<i4'), ('1
4', '<i4'), ('15', '<i4'), ('16', '<i4'), ('17', '<i4'), ('18', '<i4'), ('19', '<i4'), ('20', '<i4'), ('21', '<i4'), ('22', '<i
```

Even now the data is not in the proper format. In the above file, temperature for all days of a month are in the same row which is not right format becasue each month has different number of days. Instead each day shoul have different row

Also we'll associate each datapoint, with proper numpy datetime object

Lets write function that unrolls and applied the above transformations

startdate: We are creating a range of dates that correspond to a row. We specify the begining of the month by including only the year and the month in a string fed to numpy datetime object

dates: we then create a range of dates using numpy arange starting at the startdate, and at the startdate plus 1 months, and with step of one day

now we'll collect the data for the days from the records , we specify the field for which we are going to extract the data. We do that by enclosing the dates in enumerates, looping over the date and index over same time. And then by using that index, ti build the name of the record. Finally, we divide each datapoint by 10 since temperature are defined in tens of degrees

In [130]:
```
def unroll(record):
    startdate = np.datetime64('{}-{:02}'.format(record['year'],record['month']))
    dates = np.arange(startdate,startdate + np.timedelta64(1,'M'),np.timedelta64(1,'D'))

    rows = [(date,record[str(i+1)]/10) for i,date in enumerate(dates)]

    return np.array(rows,dtype=[('date','M8[D]'),('value','d')])
```

In [131]:
```
unroll(lihue[0])
```

Out[131]:
```
array([('1950-02-01', 25.6), ('1950-02-02', 25.6), ('1950-02-03', 25.6),
       ('1950-02-04', 26.7), ('1950-02-05', 21.7), ('1950-02-06', 22.8),
       ('1950-02-07', 25.6), ('1950-02-08', 27.2), ('1950-02-09', 25.6),
       ('1950-02-10', 25.6), ('1950-02-11', 25.6), ('1950-02-12', 24.4),
       ('1950-02-13', 25.6), ('1950-02-14', 25.6), ('1950-02-15', 24.4),
       ('1950-02-16', 24.4), ('1950-02-17', 25. ), ('1950-02-18', 25.6),
       ('1950-02-19', 23.9), ('1950-02-20', 25. ), ('1950-02-21', 25.6),
       ('1950-02-22', 25.6), ('1950-02-23', 26.7), ('1950-02-24', 26.1),
       ('1950-02-25', 26.7), ('1950-02-26', 26.7), ('1950-02-27', 26.1),
       ('1950-02-28', 26.1)], dtype=[('date', '<M8[D]'), ('value', '<f8')])
```

Now we'll take all the months contained in the file and concatenate them together into single numpy record array but we also want to select single observable such as Minimum Temperature

To do this, we'll write a function getObs, we'll write list comprehension containing the unrolled rows for each row of the parsed file and select only those that conatain our desired obs and then we'll feed this list to numpy concatenate

In [133]:
```
def getobs(filename,obs):
    return np.concatenate([unroll(row) for row in parseFile(filename) if row[2] == obs])
```

In [134]:
```
getobs('USW00022536.dly','TMIN')
```

Out[134]:
```
array([('1950-02-01',   17.8), ('1950-02-02',   15.6),
       ('1950-02-03',   16.1), ..., ('2015-09-28', -999.9),
       ('2015-09-29', -999.9), ('2015-09-30', -999.9)],
      dtype=[('date', '<M8[D]'), ('value', '<f8')])
```

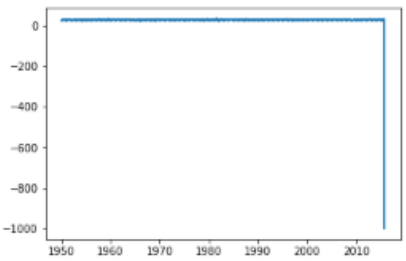
Lets get TMIN and TMAX from LIHUE

In [136]:
```
lihue_TMIN = getobs('USW00022536.dly','TMIN')
lihue_TMAX = getobs('USW00022536.dly','TMAX')
```

In [137]:
```
## We'll plot TMAX and see
%matplotlib inline
```

In [138]:
```
pp.plot(lihue_TMAX['date'],lihue_TMAX['value'])
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1cb50589e10>]
```



Something is strange. Maybe it's missing values. We have some vakues as -999.9 in out files.

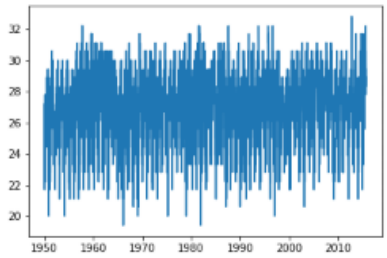
Lets replace those with np.nan and check

In [140]:
```
def getobs(filename,obs):
    data = np.concatenate([unroll(row) for row in parseFile(filename) if row[2] == obs])
    data['value'][data['value']==-999.9] = np.nan
    return data
```

In [141]:
```
lihue_TMIN = getobs('USW00022536.dly','TMIN')
lihue_TMAX = getobs('USW00022536.dly','TMAX')
```

In [142]:
```
pp.plot(lihue_TMAX['date'],lihue_TMAX['value'])
```

Out[142]:
```
[<matplotlib.lines.Line2D at 0x1cb50560a58>]
```

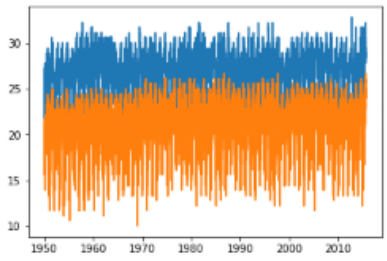


Much better. Plotting ignore the NAN values.

Lets plot TMAX and TMIN together

In [144]:
```
pp.plot(lihue_TMAX['date'],lihue_TMAX['value'])
pp.plot(lihue_TMIN['date'],lihue_TMIN['value'])

# Very Nosiy Data
```

Out[144]:
```
[<matplotlib.lines.Line2D at 0x1cb44e22240>]
```



What if we take mean values of these TMIN and TMAX ?

In [146]:
```
np.mean(lihue_TMAX['value']),np.mean(lihue_TMIN['value'])
```

Out[146]:
```
(nan, nan)
```

This is becasue mean of some values and nan is always nan

What do we do about the missing values ?

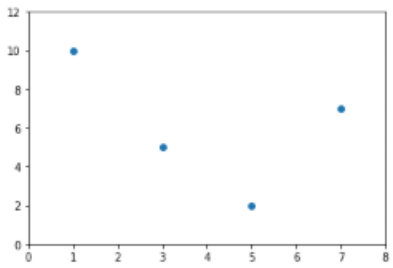
We will replace with the values of the neighbors i.e. we'll interpolate using np.interp

For example: Suppose we have array of two points

In [149]:
```
x = np.array([1,3,5,7],'d')
y = np.array([10,5,2,7],'d')
```

In [150]:
```
pp.plot(x,y,'o')
pp.axis([0,8,0,12])
```

Out[150]:
```
[0, 8, 0, 12]
```



What we are trying to do is we'are trying to get the closest point and the most reasonable way to do so is to draw lines between these points

In [151]:
```
xs = np.linspace(1,7)
xs
```

Out[151]:
```
array([1.        , 1.12244898, 1.24489796, 1.36734694, 1.48979592,
       1.6122449 , 1.73469388, 1.85714286, 1.97959184, 2.10204082,
       2.2244898 , 2.34693878, 2.46938776, 2.59183673, 2.71428571,
       2.83673469, 2.95918367, 3.08163265, 3.20408163, 3.32653061,
       3.44897959, 3.57142857, 3.69387755, 3.81632653, 3.93877551,
       4.06122449, 4.18367347, 4.30612245, 4.42857143, 4.55102041,
       4.67346939, 4.79591837, 4.91836735, 5.04081633, 5.16326531,
       5.28571429, 5.40816327, 5.53061224, 5.65306122, 5.7755102 ,
       5.89795918, 6.02040816, 6.14285714, 6.26530612, 6.3877551 ,
       6.51020408, 6.63265306, 6.75510204, 6.87755102, 7.        ])
```

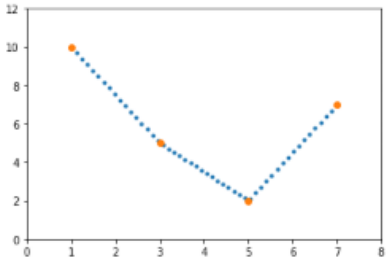
We are building interpolation between xs and x and y

In [152]:
```
ys = np.interp(xs,x,y)
ys
```

Out[152]:
```
array([10.        ,  9.69387755,  9.3877551 ,  9.08163265,  8.7755102 ,
        8.46938776,  8.16326531,  7.85714286,  7.55102041,  7.24489796,
        6.93877551,  6.63265306,  6.32653061,  6.02040816,  5.71428571,
        5.40816327,  5.10204082,  4.87755102,  4.69387755,  4.51020408,
        4.32653061,  4.14285714,  3.95918367,  3.7755102 ,  3.59183673,
        3.40816327,  3.2244898 ,  3.04081633,  2.85714286,  2.67346939,
        2.48979592,  2.30612245,  2.12244898,  2.10204082,  2.40816327,
        2.71428571,  3.02040816,  3.32653061,  3.63265306,  3.93877551,
        4.24489796,  4.55102041,  4.85714286,  5.16326531,  5.46938776,
        5.7755102 ,  6.08163265,  6.3877551 ,  6.69387755,  7.        ])
```

In [153]:
```
pp.plot(xs,ys,'.')
pp.plot(x,y,'o')

pp.axis([0,8,0,12])
```

Out[153]:
```
[0, 8, 0, 12]
```



We'll do same for filling the missing values of temperature

In [154]:
```
def fillnan(data):
    nan = np.isnan(data['value'])
    data['value'][nan] = np.interp(data['date'][nan],data['date'][~nan],data['value'][~nan])
    # interpolating for the values that are not nan
```

fillnan(lihue_TMIN):

I was trying to do arithmatic with dates. Can't do it(Gave me the error-> Cannot cast array data from dtype('<M8[D]') to dtype('float64') according to the rule 'safe') so converted dates explicitily to float64

The numpy astype method can be used to change the data types

In [156]:
```
def fillnan(data):
    dates_float = data['date'].astype(np.float64)
    nan = np.isnan(data['value'])
    data['value'][nan] = np.interp(dates_float[nan],dates_float[~nan],data['value'][~nan])# change the data['date']-> dates_floa
```

In [157]:
```
fillnan(lihue_TMAX)
```

In [158]:
```
fillnan(lihue_TMIN)
```

In [159]:
```
np.mean(lihue_TMAX['value']),np.mean(lihue_TMIN['value'])
```

Out[159]: (27.300908977192176, 21.051530250594173)

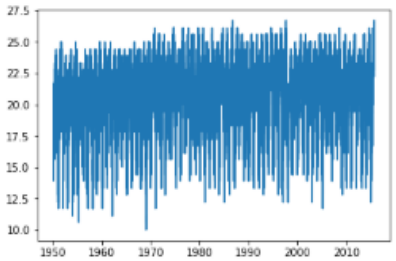
Although we have fiiled our missing values, our temperature data is still noisy. We need to smoothen it i.e. take average

Moisy data means lot of data is corrupted i.e.large amount of additional meaningless information

---

In [162]:
```
pp.plot(lihue_TMIN['date'],lihue_TMIN['value'])
```

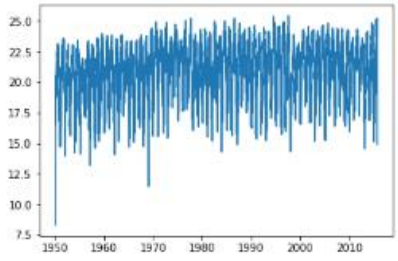
Out[162]: [<matplotlib.lines.Line2D at 0x1cb44e4d208>]



To get better results, we'll smooth averaging nearby values. We'll do this by taking running mean i.e. a mean over a limited window centered at a data point and then we can increase or decrease the size of window to adjust smoothining.We'll define a plot smooth function that will smoothen the data. Numpy function correlate multiplies sliding section of one dimensional array with another shorter array, computes the sum of multiplies values, and stores that in a new array(this is what we need for running mean).If we set the shorter array to one divided by it's size, the same option makes sure that the size of the resultant array is same as the original array

In [164]:
```
def smooth_plot(t, win=10):
    smoothed = np.correlate(t['value'],np.ones(win)/win,'same')
    pp.plot(t['date'],smoothed)
```

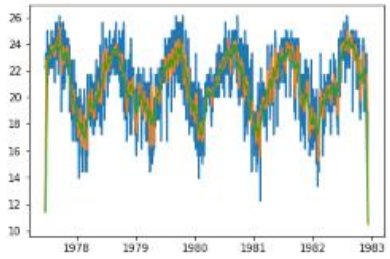
In [165]:
```
smooth_plot(lihue_TMIN)
```



In [166]:
```
lihue_TMIN.shape
```

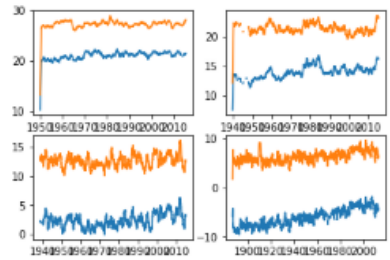
Out[166]:
```
(23983,)
```

In [167]:
```
pp.plot(lihue_TMIN[10000:12000]['date'],lihue_TMIN[10000:12000]['value'])# blue
smooth_plot(lihue_TMIN[10000:12000])# orange
smooth_plot(lihue_TMIN[10000:12000],30)# green with window size 30
```

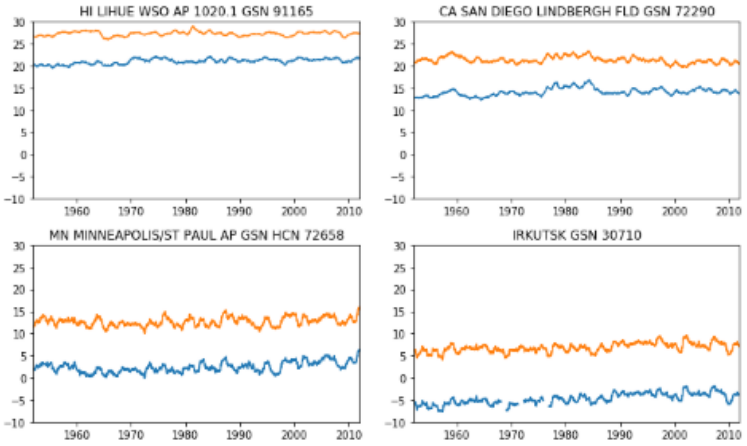


Lets do this for all of our four stations

In [169]:
```
for i,code in enumerate(dataStations):
    pp.subplot(2,2,i+1) # we tell matplotlib to place this plots in larger plot of array 2,2 and to get index, we enumerate data
    smooth_plot(getobs("{}.dly".format(code),'TMIN'),365)
    smooth_plot(getobs("{}.dly".format(code),'TMAX'),365)
```



In [170]:
```
pp.figure(figsize=(10,6))# Makes it a little bigger

for i,code in enumerate(dataStations):
    pp.subplot(2,2,i+1) # we tell matplotlib to place this plots in larger plot of array 2,2 and to get index, we enumerate data
    smooth_plot(getobs("{}.dly".format(code),'TMIN'),365)
    smooth_plot(getobs("{}.dly".format(code),'TMAX'),365)

    #Lets plot same range of years and temperature for all the stations by using axis function which conveniently plots datetime
    pp.axis(xmin=np.datetime64('1952'),xmax = np.datetime64('2012'),ymin = -10,ymax = 30)
    # title of each subplot
    pp.title(stations[code])

pp.tight_layout() # gives clean plots
```



In [171]:
```
# This is the final represnentation of how weather can be differnet is differnet locations
```

One final thing I want to show is computing records i.e. the most extreme(lowest or highest) temperatures on a particular day of a year across all the years. What we will do is we'll recast the temperature data into 2 dimensional array where each row is year, and each column is day. We will start by writing a function that extracts single year of data and for that we will use numpy boolean masking indexing

In [173]:
```
def selectyear(data, year):
    # numpy datatime object for the start and end of a year
    start = np.datetime64("{}".format(year))
    end = start + np.timedelta64(1,'Y')

    # building boolean mask we need by combining two condition and returning only temperatures
    return data[(data['date']>=start) & (data['date']<end)]['value']
```

In [174]:
```
selectyear(lihue_TMIN,1951)
```

Out[174]:
```
array([17.8, 17.8, 17.2, 18.3, 20. , 21.7, 22.2, 21.7, 22.2, 22.2, 21.7,
       21.7, 19.4, 18.9, 18.3, 18.3, 19.4, 20.6, 16.7, 14.4, 14.4, 17.2,
       15. , 12.8, 14.4, 13.3, 15. , 14.4, 17.8, 18.3, 16.7, 16.7, 17.2,
       16.7, 17.8, 14.4, 12.2, 13.3, 13.9, 12.8, 14.4, 14.4, 18.9, 19.4,
       20. , 18.3, 18.3, 18.9, 20. , 18.3, 19.4, 13.3, 11.7, 17.8, 17.2,
       17.8, 21.1, 22.2, 20.6, 22.2, 19.4, 20. , 18.3, 17.8, 17.8, 17.2,
       16.1, 18.9, 18.9, 18.9, 17.8, 16.7, 15.6, 13.3, 13.9, 15.6, 15.6,
       21.7, 20.6, 18.3, 15.6, 14.4, 17.2, 17.8, 18.9, 20. , 18.9, 18.3,
       18.3, 20.6, 20. , 18.9, 18.3, 21.1, 16.7, 21.7, 21.1, 21.7, 17.2,
       16.7, 17.8, 18.3, 16.1, 19.4, 20.6, 18.3, 18.3, 17.2, 18.9, 21.7,
       21.7, 21.7, 21.1, 20.6, 20.6, 20. , 21.7, 20. , 20. , 19.4, 18.3,
       21.7, 22.2, 22.8, 20. , 19.4, 17.2, 22.2, 22.8, 22.8, 22.8, 18.9,
       17.8, 18.3, 19.4, 22.8, 22.2, 22.2, 20.6, 21.1, 21.1, 22.8, 19.4,
       19.4, 20.6, 21.7, 23.3, 23.3, 22.2, 22.2, 23.3, 20. , 20. , 20.6,
       20. , 21.7, 20.6, 22.8, 23.3, 22.8, 22.2, 22.8, 23.3, 22.2, 22.8,
       21.1, 21.1, 22.2, 22.2, 20. , 20.6, 20. , 23.9, 23.3, 20. , 20. ,
       20.6, 20.6, 18.9, 19.4, 20.6, 20. , 22.2, 24.4, 23.9, 22.2, 22.8,
       22.8, 21.7, 23.3, 23.9, 22.2, 23.3, 21.1, 20.6, 21.7, 21.1, 23.9,
       24.4, 23.3, 22.2, 22.8, 22.2, 22.8, 22.8, 23.3, 24.4, 23.9, 23.9,
       22.2, 23.9, 24.4, 22.8, 22.8, 21.7, 22.8, 22.8, 22.8, 22.8, 23.9,
       20.6, 24.4, 24.4, 24.4, 22.2, 22.8, 22.8, 20.6, 21.1, 21.1, 22.2,
       23.9, 25. , 23.9, 21.7, 20.6, 23.9, 22.2, 21.7, 21.1, 21.1, 23.3,
       23.9, 22.2, 21.7, 22.8, 23.9, 21.1, 22.2, 21.7, 23.3, 23.9, 23.3,
       21.7, 21.7, 20. , 18.9, 20. , 20.6, 21.7, 21.7, 20.6, 20.6, 22.8,
       22.8, 22.8, 21.7, 22.8, 24.4, 23.3, 22.8, 22.8, 22.8, 22.2, 25. ,
       24.4, 24.4, 23.9, 22.8, 23.3, 21.7, 21.7, 22.8, 24.4, 20.6, 22.2,
       23.9, 22.2, 20.6, 21.7, 21.7, 22.2, 22.8, 22.2, 23.9, 21.7, 20. ,
       18.3, 21.1, 23.3, 21.1, 21.1, 20. , 20. , 21.7, 21.1, 21.1, 20.6,
```

Now lets arrange all the years in the matrix

For convenience we'll ignore extra day in leap years..this will give us little error but that's ok

In [176]:
```
lihue_tmin_all = np.vstack([selectyear(lihue_TMIN,year)[:365] for year in range(1951,2014+1)])
```

This is years in rows and days in columns for each year

In [177]:
```
lihue_tmin_all
```

Out[177]:
```
array([[17.8, 17.8, 17.2, ..., 20.6, 19.4, 18.9],
       [17.2, 19.4, 21.7, ..., 21.7, 20.6, 17.8],
       [17.8, 18.3, 19.4, ..., 13.3, 15. , 15. ],
       ...,
       [17.8, 17.2, 17.8, ..., 18.3, 18.9, 19.4],
       [20. , 20. , 19.4, ..., 19.4, 20. , 18.3],
       [17.8, 20. , 17.8, ..., 18.9, 16.1, 14.4]])
```

1951 to 2015 is 64 years and each year has 365 days

In [178]:
```
lihue_tmin_all.shape
```

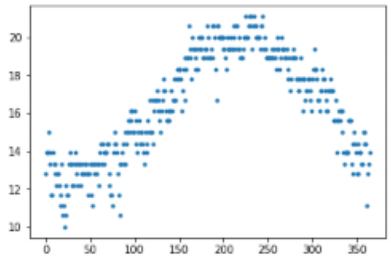
Out[178]:
```
(64, 365)
```

Now we will calculate max and minimun along each row

In [179]:
```
lihue_tmin_recodmin = np.min(lihue_tmin_all, axis=0)
lihue_tmin_recodmax = np.max(lihue_tmin_all,axis = 0)
```

In [180]:
```
pp.plot(lihue_tmin_recodmin,'.')
```

Out[180]:
```
[<matplotlib.lines.Line2D at 0x1cb4d9cb860>]
```



It looks weird maybe becasue the original data must have the resolution of Fahrenheit degree

We'll build our plot now but we also need stacked array of tmax data

In [183]:
```
lihue_tmax_all = np.vstack([selectyear(lihue_TMAX,year)[:365] for year in range(1951,2014+1)])
```

In [184]:
```
lihue_tmax_all
```

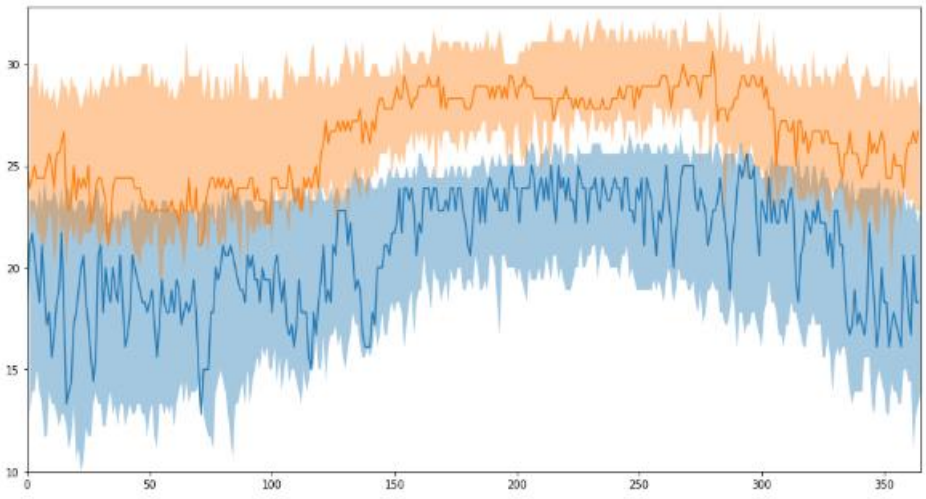
Out[184]:
```
array([[25.6, 25. , 25.6, ..., 23.3, 24.4, 25.6],
       [26.7, 25. , 24.4, ..., 26.7, 25. , 26.1],
       [24.4, 25.6, 26.1, ..., 24.4, 23.3, 25.6],
       ...,
       [25. , 27.2, 26.1, ..., 25.6, 25.6, 25.6],
       [23.9, 24.4, 23.3, ..., 25. , 25.6, 26.1],
       [26.1, 27.8, 25. , ..., 25. , 26.7, 24.4]])
```

We'll select year 2009 to plot against our records

In [193]:
```
pp.figure(figsize=(15,8))
pp.margins(0) # sets axis to the corner

days = np.arange(1,365+1)

#TMIN Data
# matpplotlib function to plot a shaded area is np.fill_between which requires one dimensional array for horizontal direction
pp.fill_between(days, np.min(lihue_tmin_all, axis =0), np.max(lihue_tmin_all,axis=0), alpha = 0.4)
# alpha value makes it partly transparent

# And now we will plot 2009 data against it
pp.plot(selectyear(lihue_TMIN,2009))


#TMAX Data
# matpplotlib function to plot a shaded area is np.fill_between which requires one dimensional array for horizontal direction
pp.fill_between(days, np.min(lihue_tmax_all, axis =0), np.max(lihue_tmax_all,axis=0), alpha = 0.4)
# alpha value makes it partly transparent

# And now we will plot 2009 data against it
pp.plot(selectyear(lihue_TMAX,2009))

pp.axis(xmax=365)
```

Out[193]:
```
(0.0, 365, 10.0, 32.8)
```

Out[193]:
(0.0, 365, 10.0, 32.8)



This tells us the min and max temperature for each day in 2009 year