In [1]:

```
# import pandas
import pandas as pd
# import numpy
import numpy as np
# import matplotlib
from matplotlib import pyplot as plt
# make plots appear and be stored within the notebook
%matplotlib inline
```

In [21]:

```
#1
Sig_Eqs=pd.read_csv("earthquakes-2022-10-30_19-56-23_+0800.tsv",sep="\t",engine='python')
```

In [22]:

```
#Learn from Wenting Yuan
#1.1
#Aggregate the total number of deaths by country
total_0=Sig_Eqs.groupby(['Country']).sum()['Deaths']
#In descending order, show the top 20
total_0.sort_values(ascending=False).head(20)
```

Out[22]:

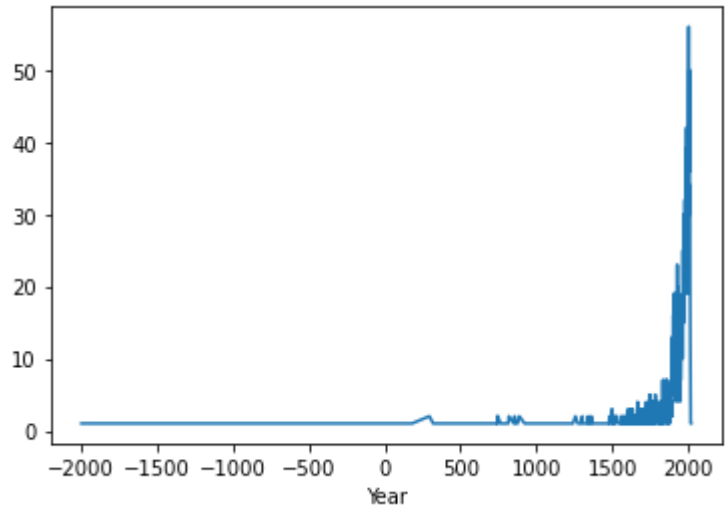
```
Country
CHINA           2075019.0
TURKEY          1134569.0
IRAN            1011446.0
ITALY            498477.0
SYRIA            439224.0
HAITI            323474.0
AZERBAIJAN       317219.0
JAPAN            278142.0
ARMENIA          191890.0
PAKISTAN         145083.0
IRAQ             136200.0
ECUADOR          135479.0
TURKMENISTAN     117412.0
PERU             102219.0
ISRAEL            90388.0
PORTUGAL          83531.0
GREECE            79174.0
CHILE             64276.0
INDIA             63491.0
TAIWAN            57135.0
Name: Deaths, dtype: float64
```

In [23]:

```
#1.2
#Add a new column for statistical data and calculate the total number of earthquakes
Sig_Eqs['Number']=1
total_1=Sig_Eqs.loc[Sig_Eqs['Ms']>3].groupby(['Year']).sum()['Number']
#plot
total_1.plot.line()
```

Out[23]:

<AxesSubplot:xlabel='Year'>



In [24]:

```
#1.3
#Use total_2 to define the highest magnitude of the earthquake, and then use the function to return,
total_2=Sig_Eqs.groupby(['Country']).max()['Ms']
def CountEq_LargestEq(country):
    return [total_0[country],total_2[country]]
CountEq_LargestEq("CHINA")
```

```
C:\Users\98671\AppData\Local\Temp\ipykernel_4140\1509182341.py:1: FutureWarning: Dro
pping invalid columns in DataFrameGroupBy.max is deprecated. In a future version, a
TypeError will be raised. Before calling .max, select only columns which should be v
alid for the function.
  total_2=Sig_Eqs.groupby(['Country']).max()['Ms']
```

Out[24]:

[2075019.0, 8.6]

In [25]:

```
#1.3
#Establish a cycle that outputs for the country with the corresponding death toll and maximum magnit
for i in set(Sig_Eqs['Country']):
#If the type is floating, it continues to run
    if type(i)==float:
          continue
#If not, the output
    else:
        print(i)
        print(CountEq_LargestEq(i))
```

```
UGANDA
[171.0, 7.0]
MONTENEGRO
[131.0, 6.9]
VENEZUELA
[44480.0, 8.4]
BOLIVIA
[111.0, 7.5]
FRANCE
[6454.0, 6.2]
SAINT LUCIA
[900.0, nan]
NEW ZEALAND
[476.0, 8.0]
CYPRUS
[42.0, 6.5]
AZERBAIJAN
[317219.0, 6.9]
HONDURAS
[12.0, 7.5]
CHILE
[64276.0, 8.7]
SRI LANKA
[0.0, nan]
TOGO
[0.0, nan]
BURUNDI
[3.0, nan]
SAINT VINCENT AND THE GRENADINES
[0.0, nan]
TAJIKISTAN
[16145.0, 7.4]
ECUADOR
[135479.0, 8.6]
FRENCH POLYNESIA
[0.0, 6.5]
RWANDA
[12.0, 4.9]
MALAWI
[13.0, 6.1]
MONTSERRAT
[0.0, nan]
GERMANY
[2000.0, 5.3]
UK
[1400.0, 6.0]
ARGENTINA
[22520.0, 7.5]
```

SAMOA
[0.0, 8.3]
MALAYSIA
[19.0, 6.2]
RUSSIA
[2472.0, 8.5]
CAMEROON
[0.0, nan]
ARMENIA
[191890.0, 6.8]
SPAIN
[3005.0, 6.7]
SERBIA
[7.0, 5.8]
LEBANON
[30208.0, nan]
SAUDI ARABIA
[0.0, 5.3]
NORTH KOREA
[0.0, 6.4]
ICELAND
[41.0, 6.6]
INDONESIA
[40137.0, 8.8]
SWITZERLAND
[410.0, nan]
NEW CALEDONIA
[0.0, 8.0]
SOUTH AFRICA
[55.0, 7.9]
LIBYA
[300.0, 5.4]
KAZAKHSTAN
[452.0, 8.4]
ANTIGUA AND BARBUDA
[0.0, 8.0]
ETHIOPIA
[200.0, 6.5]
BANGLADESH
[336.0, 7.6]
PORTUGAL
[83531.0, 8.5]
BRITISH VIRGIN ISLANDS
[0.0, nan]
VIETNAM
[0.0, 6.6]
SOUTH KOREA
[151.0, 6.5]
SIERRA LEONE
[0.0, nan]
GUATEMALA
[36189.0, 7.9]
TURKEY
[1134569.0, 7.8]
UZBEKISTAN
[19904.0, 7.0]
AUSTRIA
[5040.0, 5.7]
HAITI
[323474.0, 8.1]
AZORES (PORTUGAL)

[6354.0, 7.2]
ERITREA
[0.0, 6.8]
FRENCH GUIANA
[0.0, nan]
INDIAN OCEAN
[0.0, 8.1]
UK TERRITORY
[0.0, 7.6]
VANUATU
[6.0, 8.1]
ALGERIA
[39339.0, 7.7]
KIRIBATI
[0.0, nan]
NORWAY
[0.0, 5.8]
KERMADEC ISLANDS (NEW ZEALAND)
[0.0, 8.1]
BRAZIL
[2.0, 4.8]
ANTARCTICA
[0.0, 8.0]
GEORGIA
[285.0, 7.5]
TAIWAN
[57135.0, 8.0]
MACEDONIA
[1083.0, 6.2]
KYRGYZSTAN
[207.0, 7.5]
PACIFIC OCEAN
[0.0, 6.1]
JAMAICA
[2300.0, 7.8]
NETHERLANDS
[1.0, 5.2]
BELGIUM
[2.0, nan]
PERU
[102219.0, 8.6]
POLAND
[28.0, 3.1]
AFGHANISTAN
[14254.0, 8.1]
ROMANIA
[2701.0, 6.9]
BERING SEA
[0.0, 6.5]
EGYPT
[41765.0, 7.3]
TURKMENISTAN
[117412.0, 8.2]
TUNISIA
[48013.0, 5.6]
ATLANTIC OCEAN
[0.0, 8.3]
COLOMBIA
[5614.0, 7.9]
TONGA
[1.0, 8.2]

DJIBOUTI
[6.0, 6.3]
ITALY
[498477.0, 7.5]
SOUTH SUDAN
[31.0, 7.1]
CUBA
[54.0, 7.0]
PHILIPPINES
[6170.0, 8.7]
NEPAL
[21454.0, 8.2]
ALBANIA
[3132.0, 7.5]
KENYA
[1.0, 6.9]
HUNGARY
[85.0, nan]
PAKISTAN
[145083.0, 8.0]
PALAU
[0.0, 7.6]
CENTRAL AFRICAN REPUBLIC
[0.0, nan]
GRENADA
[0.0, nan]
THAILAND
[1.0, 5.9]
GHANA
[25.0, 6.4]
SYRIA
[439224.0, 7.6]
TRINIDAD AND TOBAGO
[2.0, 7.0]
BARBADOS
[3000.0, nan]
MARTINIQUE
[391.0, 7.9]
BHUTAN
[11.0, 6.1]
YEMEN
[4192.0, 6.0]
DOMINICAN REPUBLIC
[13.0, 8.0]
COMOROS
[0.0, nan]
IRELAND
[100.0, nan]
MOROCCO
[19829.0, 6.4]
CONGO
[80.0, 7.2]
JAPAN
[278142.0, 8.6]
CANADA
[302.0, 8.1]
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS
[0.0, 7.8]
BOSNIA-HERZEGOVINA
[48.0, 5.6]
NICARAGUA

[13460.0, 7.9]
WALLIS AND FUTUNA (FRENCH TERRITORY)
[5.0, 6.4]
FIJI
[2.0, 8.3]
LAOS
[0.0, 6.4]
MADAGASCAR
[2.0, nan]
USA TERRITORY
[4.0, 8.1]
PAPUA NEW GUINEA
[311.0, 8.2]
GREECE
[79174.0, 8.0]
SOLOMON ISLANDS
[56.0, 8.1]
BULGARIA
[138.0, 7.8]
ISRAEL
[90388.0, 6.4]
IRAQ
[136200.0, 6.4]
SOLOMON SEA
[0.0, 7.3]
GUINEA
[443.0, 6.2]
UKRAINE
[11.0, 6.8]
COTE D'IVOIRE
[0.0, nan]
MEXICO
[14726.0, 8.3]
MICRONESIA, FED. STATES OF
[0.0, 7.7]
SUDAN
[2.0, 5.1]
ZAMBIA
[0.0, nan]
CHINA
[2075019.0, 8.6]
INDIA
[63491.0, 8.7]
SLOVAKIA
[30.0, nan]
TANZANIA
[28.0, 7.3]
MYANMAR (BURMA)
[1167.0, 8.0]
GABON
[0.0, 6.2]
JORDAN
[0.0, 6.4]
MOZAMBIQUE
[4.0, 7.5]
MONGOLIA
[30.0, 8.7]
COSTA RICA
[2655.0, 7.7]
CZECH REPUBLIC
[2.0, nan]

GUADELOUPE
[5007.0, 7.2]
CANARY ISLANDS
[0.0, nan]
SLOVENIA
[23.0, 6.5]
AUSTRALIA
[12.0, 8.2]
PANAMA
[7.0, 8.3]
USA
[1360.0, 9.1]
IRAN
[1011446.0, 7.9]
EL SALVADOR
[6081.0, 7.8]
CROATIA
[5017.0, 7.2]
URUGUAY
[0.0, nan]

In [2]:

```
#2
#I didn't understand the topic very well, so I borrowed the code of Xiaoqiao Jiao
W=pd.read_csv("Baoan_Weather_1998_2022.csv")
W
```

```
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\4056314331.py:1: DtypeWarning: Col
umns (4,8,9,10,11,14,15,24,25,27,29,31,34,37,38,40,41,45,49,50) have mixed types. Sp
ecify dtype option on import or set low_memory=False.
  W=pd.read_csv("Baoan_Weather_1998_2022.csv")
```

Out[2]:

	STATION	DATE	SOURCE	REPORT_TYPE	CALL_SIGN	QUALITY_CONTROL
0	59493099999	1998-01-01T00:00:00	4	SY-MT	ZGSZ	V020
1	59493099999	1998-01-01T01:00:00	4	FM-15	ZGSZ	V020
2	59493099999	1998-01-01T02:00:00	4	FM-15	ZGSZ	V020
3	59493099999	1998-01-01T03:00:00	4	SY-MT	ZGSZ	V020
4	59493099999	1998-01-01T04:00:00	4	FM-15	ZGSZ	V020
...	...	...	...	...	...	...
235669	59493099999	2022-10-10T20:00:00	4	FM-15	99999	V020
235670	59493099999	2022-10-10T21:00:00	4	FM-12	99999	V020
235671	59493099999	2022-10-10T21:00:00	4	FM-15	99999	V020
235672	59493099999	2022-10-10T22:00:00	4	FM-15	99999	V020
235673	59493099999	2022-10-10T23:00:00	4	FM-15	99999	V020

235674 rows × 54 columns

In [3]:

```
W['month']=W['DATE']
# Extract the years and months, and then look at the average monthly temperature needs to be used
for i in range(235674):
    [a,b,c]=W['DATE'][i].split("T")[0].split("-")
    W['month'][i]=a+b
```

```
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\270593233.py:5: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  W['month'][i]=a+b
```

In [4]:

```
# Temperature data is processed
W['T']=W['SOURCE.1']
for i in range(235674):
    [a,flag]=W['TMP'][i].split(",")
    if flag=='9':
        W=W.drop(index=[i])
        # Remove unusable data
        continue
    else:
        W['T'][i]=int(a.split("+")[1])/10
```

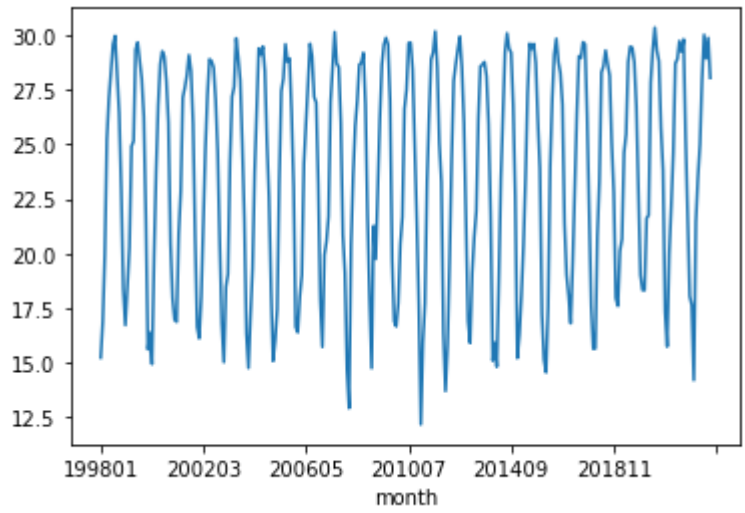
```
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\1903939622.py:10: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  W['T'][i]=int(a.split("+")[1])/10
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\1903939622.py:10: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  W['T'][i]=int(a.split("+")[1])/10
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\1903939622.py:10: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

In [5]:

```
# plot
W.groupby(['month']).mean()['T'].plot()
```

Out[5]:

<AxesSubplot:xlabel='month'>



In [3]:

```
#3
#Load the archive
df = pd.read_csv('ibtracs.ALL.list.v04r00.csv',
                 usecols=range(17),
                 skiprows=[1, 2],
                 parse_dates=['ISO_TIME'],
                 na_values={'NAME':'NOT_NAMED','WMO_WIND': ' '})
df.head()
```

```
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\3808063794.py:1: DtypeWarning: C
olumns (5) have mixed types. Specify dtype option on import or set low_memory=Fals
e.
  df = pd.read_csv('ibtracs.ALL.list.v04r00.csv',
```

Out[3]:

	SID	SEASON	NUMBER	BASIN	SUBBASIN	NAME	ISO_TIME	NATURE	LAT	LON	WI
0	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 06:00:00	NR	10.8709	79.8265	
1	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 09:00:00	NR	10.8431	79.3524	
2	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 12:00:00	NR	10.8188	78.8772	
							1842-10-				

In [28]:

```
#3.1
df.groupby(['SID','NAME']).max().sort_values('WMO_WIND',ascending=False).head(10)['WMO_WIND']
```

```
C:\Users\98671\AppData\Local\Temp\ipykernel_23484\2319268501.py:1: FutureWarning: Dr
opping invalid columns in DataFrameGroupBy.max is deprecated. In a future version, a
TypeError will be raised. Before calling .max, select only columns which should be v
alid for the function.
  df.groupby(['SID','NAME']).max().sort_values('WMO_WIND',ascending=False).head(10)
['WMO_WIND']
```

Out[28]:

```
SID            NAME
2015293N13266  PATRICIA    185.0
1980214N11330  ALLEN       165.0
1988253N12306  GILBERT     160.0
2005289N18282  WILMA       160.0
1997253N12255  LINDA       160.0
2019236N10314  DORIAN      160.0
2009288N07267  RICK        155.0
2017242N16333  IRMA        155.0
2005261N21290  RITA        155.0
1998295N12284  MITCH       155.0
Name: WMO_WIND, dtype: float64
```

In [4]:

```
#3.2
#Since I had limited abilities, I borrowed from Xiaoqiao Jiao
WN=df.groupby(['SID']).max().sort_values('WMO_WIND',ascending=False).head(20)
WN.iloc[:,7]=pd.DataFrame(WN.iloc[:,7],dtype=np.float)
WN.iloc[:,7].plot(kind='bar')
```

```
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\331824171.py:1: FutureWarning: D
ropping invalid columns in DataFrameGroupBy.max is deprecated. In a future versio
n, a TypeError will be raised. Before calling .max, select only columns which shou
ld be valid for the function.
  WN=df.groupby(['SID']).max().sort_values('WMO_WIND',ascending=False).head(20)
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\331824171.py:2: DeprecationWarni
ng: `np.float` is a deprecated alias for the builtin `float`. To silence this warn
ing, use `float` by itself. Doing this will not modify any behavior and is safe. I
f you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdoc
s/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0
-notes.html#deprecations)
  WN.iloc[:,7]=pd.DataFrame(WN.iloc[:,7],dtype=np.float)
```

Out[4]:

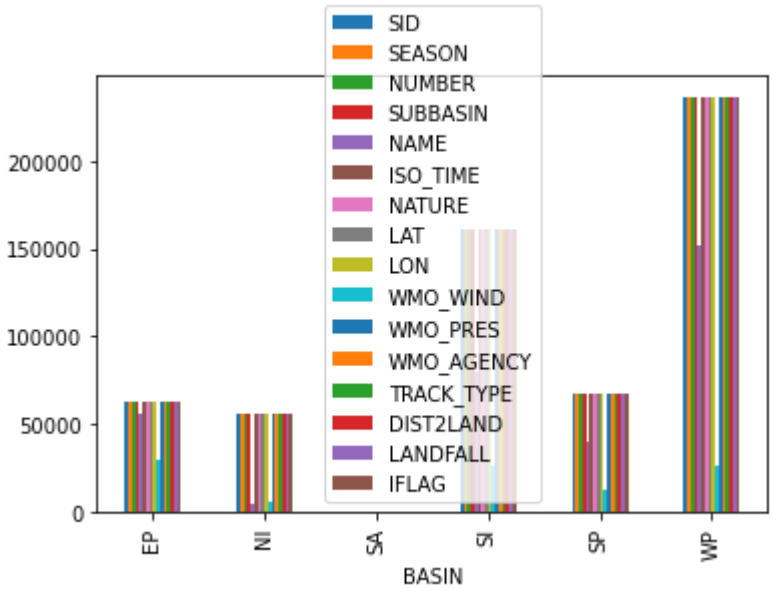
```
<AxesSubplot:xlabel='SID'>
```

In [5]:

```
#3.3
df.groupby(['BASIN']).count().plot(kind='bar')
```

Out[5]:

```
<AxesSubplot:xlabel='BASIN'>
```

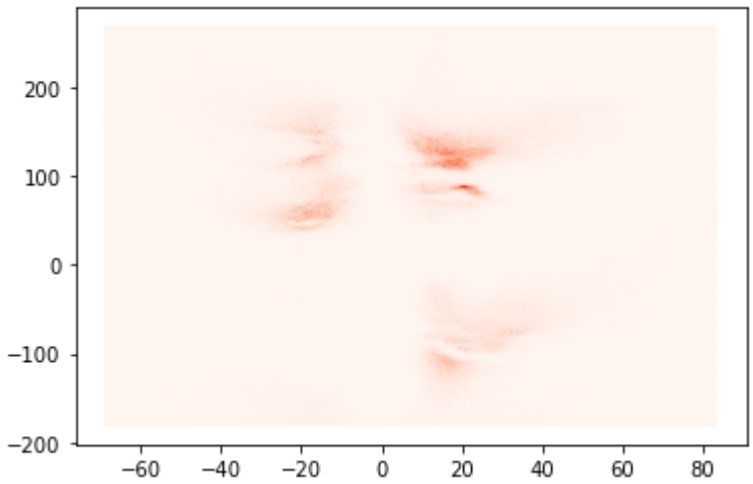


In [6]:

```
#3.4
##Since I had limited abilities, I borrowed from Xiaoqiao Jiao
x=df.iloc[:,8]
y=df.iloc[:,9]
plt.hexbin(x, y, gridsize = 500,cmap='Reds')
```

Out[6]:

<matplotlib.collections.PolyCollection at 0x1ada9a23eb0>

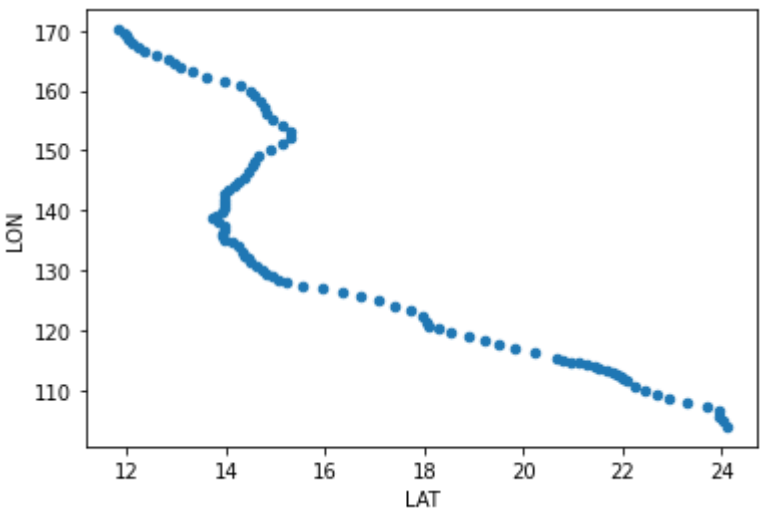


In [7]:

```
# 3.5
df.loc[(df['NAME']=='MANGKHUT') & (df['SEASON']==2018)].plot.scatter(x='LAT',y='LON')
```

Out[7]:

<AxesSubplot:xlabel='LAT', ylabel='LON'>



In [8]:

```
# 3.6
df_1970=df.loc[(df['SEASON']>1970)&((df['BASIN']=='WP')|(df['BASIN']=='EP'))]
df_1970
```

Out[8]:

	SID	SEASON	NUMBER	BASIN	SUBBASIN	NAME	ISO_TIME	NATURE
357392	1971008N07139	1971	3	WP	MM	SARAH	1971-01-08 00:00:00	TS
357393	1971008N07139	1971	3	WP	MM	SARAH	1971-01-08 03:00:00	TS
357394	1971008N07139	1971	3	WP	MM	SARAH	1971-01-08 06:00:00	TS
357395	1971008N07139	1971	3	WP	MM	SARAH	1971-01-08 09:00:00	TS
357396	1971008N07139	1971	3	WP	MM	SARAH	1971-01-08 12:00:00	TS
...	...	...	...	...	...	...	...	...
707084	2022275N10316	2022	76	EP	MM	JULIA	2022-10-10 15:00:00	TS
707085	2022275N10316	2022	76	EP	MM	JULIA	2022-10-10 18:00:00	NR
707173	2022286N15151	2022	80	WP	MM	NaN	2022-10-12 12:00:00	NR
707174	2022286N15151	2022	80	WP	MM	NaN	2022-10-12 15:00:00	NR
707175	2022286N15151	2022	80	WP	MM	NaN	2022-10-12 18:00:00	NR

172797 rows × 17 columns

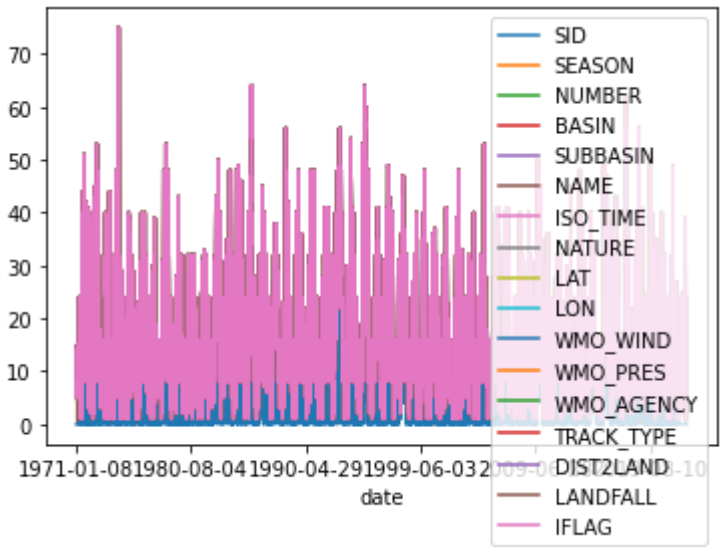
In [9]:

```
# 3.7
df_1970['date']=df_1970['ISO_TIME'].dt.strftime("%Y-%m-%d")
df_1970.groupby(['date']).count().plot()
```

```
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\3238627107.py:2: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_1970['date']=df_1970['ISO_TIME'].dt.strftime("%Y-%m-%d")
```

Out[9]:

<AxesSubplot:xlabel='date'>



In [10]:

```
# 3.8
#Since I had limited abilities, I borrowed from Xiaoqiao Jiao
import calendar
#Build two arrays to represent that the 1st of each month is the day of the year, normal represents
normal=[1,32,60,91,121,152,182,213,244,274,305,335]
leap=[1,32,61,92,122,153,183,214,245,275,306,336]
df_1970['DOF']=df_1970['SEASON']
# The establishment calculation is a function of the day of the year
def dof(D):
    [year,month,day]=D.split("-")
    if calendar.isleap(int(year)):
        return leap[int(month)-1]+int(day)-1
    else:
        return normal[int(month)-1]+int(day)-1
for i in range(len(df_1970)):
    df_1970.iloc[i,18]=dof(df_1970.iloc[i,17])
```

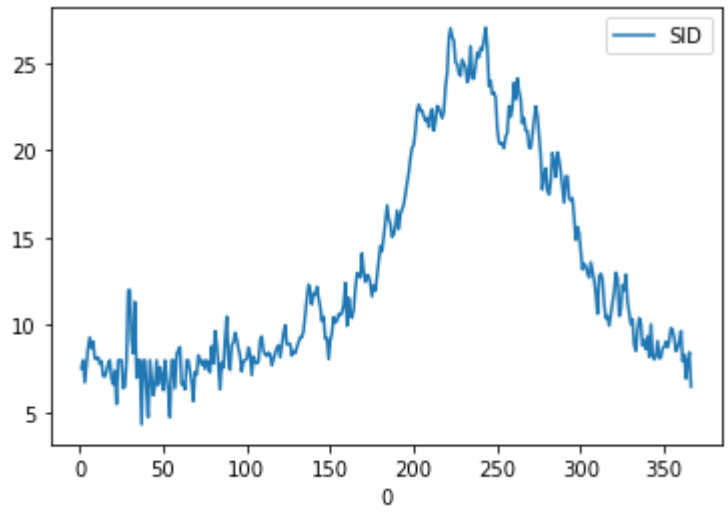
```
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\1894707143.py:6: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_1970['DOF']=df_1970['SEASON']
```

In [11]:

```
Q1=df_1970.groupby(['date']).count()['SID']
Q2=df_1970.groupby(['date']).mean()['DOF']
array=[0]*len(Q1)
Q3=pd.Series(data=array,index=Q1.index)
for i in range(len(Q2)):
    Q3[i]=int(Q2[i])
P=pd.concat([Q1,Q3],axis=1)
# Group by day of year and average
P.groupby([0]).mean().plot()
```

Out[11]:

<AxesSubplot:xlabel='0'>



In [97]:

```
# Define the function Returns the average number of typhoons in a given day of year
def clima_dof(d):
    return P.groupby([0]).mean().loc[d][0]
clima_dof(366)
```

Out[97]:

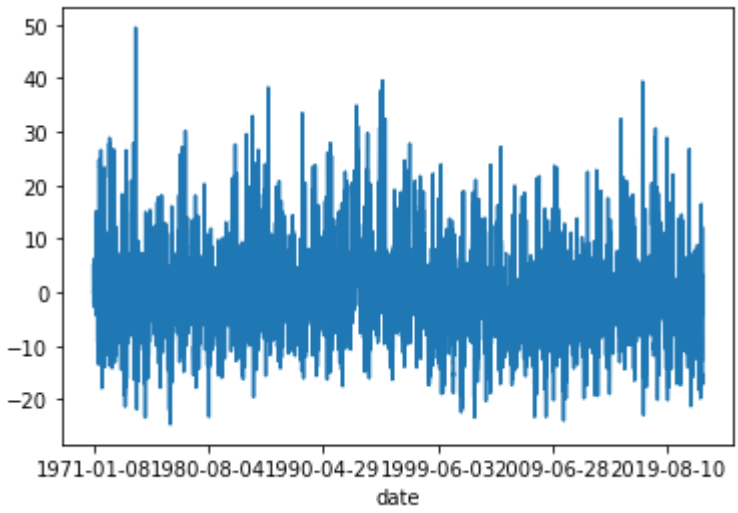
6.5

In [13]:

```
# 3.9
#Since I had limited abilities, I borrowed from Xiaoqiao Jiao
#Copy a new table
P_copy=P.copy()
P_copy['anomaly']=P_copy.iloc[:,0]
for i in range(len(P)):
    P_copy.iloc[i,2]=P_copy.iloc[i,0]-clima_dof(P_copy.iloc[i,1])
P_copy['anomaly'].plot()
```

Out[13]:

<AxesSubplot:xlabel='date'>



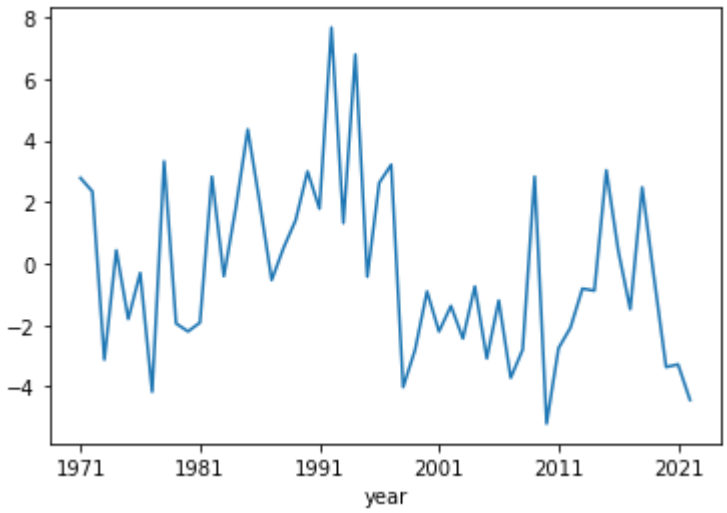
In [14]:

```
# 3.10
#Since I had limited abilities, I borrowed from Xiaoqiao Jiao
P_copy['date'] = P_copy.index
# Extract the years in the date and use them as the annual average
P_copy['year']=P_copy['date']
for i in range(len(P_copy)):
    P_copy['year'][i]=P_copy['date'][i].split("-")[0]
P_copy.groupby(['year']).mean()['anomaly'].plot()
```

```
C:\Users\zwc15\AppData\Local\Temp\ipykernel_13852\84352840.py:7: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  P_copy['year'][i]=P_copy['date'][i].split("-")[0]
```

Out[14]:

```
<AxesSubplot:xlabel='year'>
```



In [15]:

```
P_copy.groupby(['year']).mean()['anomaly']
# From the results, 1992, 1994, 2010 were the most abnormal years
```

Out[15]:

```
year
1971     2.780170
1972     2.339433
1973    -3.124258
1974     0.430385
1975    -1.802952
1976    -0.305495
1977    -4.167689
1978     3.319792
1979    -1.947654
1980    -2.207329
1981    -1.918813
1982     2.821711
1983    -0.409756
1984     1.809274
1985     4.356397
1986     1.975023
1987    -0.535556
1988     0.514886
1989     1.402370
1990     2.991816
1991     1.781754
1992     7.666144
1993     1.314375
1994     6.788119
1995    -0.427993
1996     2.622056
1997     3.210332
1998    -4.013642
1999    -2.800127
2000    -0.907160
2001    -2.209833
2002    -1.377742
2003    -2.436437
2004    -0.743783
2005    -3.083058
2006    -1.200343
2007    -3.714239
2008    -2.797557
2009     2.823854
2010    -5.201969
2011    -2.750902
2012    -2.082631
2013    -0.815115
2014    -0.882389
2015     3.030031
2016     0.414488
2017    -1.484889
2018     2.475726
2019    -0.479715
2020    -3.364472
2021    -3.277406
2022    -4.430465
Name: anomaly, dtype: float64
```

In [79]:

```
#4
#Global Summary of the Year (GSOY), Version 1
#4.1
GSY=pd.read_csv("USC00218450.csv",engine='python')
GSY
```

Out[79]:

	STATION	DATE	LATITUDE	LONGITUDE	ELEVATION	NAME	CDSD	CDSD_ATTRIBUTES	CLD
0	USC00218450	1961	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	NaN	NaN	Na
1	USC00218450	1962	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	NaN	NaN	Na
2	USC00218450	1963	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	395.5	NaN	395
3	USC00218450	1964	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	461.7	NaN	461

In [83]:

```
#4.1
#Remove columns that are all null and fill in the rest with values from the next row
GSY=GSY.dropna(axis=1,how="all")
GSY=GSY.fillna(axis=0,method='bfill')
GSY
```

Out[83]:

	STATION	DATE	LATITUDE	LONGITUDE	ELEVATION	NAME	CDSD	CLDD	CLI
0	USC00218450	1961	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	395.5	395.5	
1	USC00218450	1962	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	395.5	395.5	
2	USC00218450	1963	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	395.5	395.5	
3	USC00218450	1964	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	461.7	461.7	
4	USC00218450	1965	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	250.5	250.5	
...	...	...	...	...	...	...	...	...	
56	USC00218450	2018	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	660.6	660.6	
57	USC00218450	2019	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	291.8	291.8	
58	USC00218450	2020	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	385.8	385.8	
59	USC00218450	2021	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	488.9	488.9	
60	USC00218450	2022	44.9902	-93.17995	295.7	UNIVERSITY OF MN ST. PAUL, MN US	488.9	488.9	

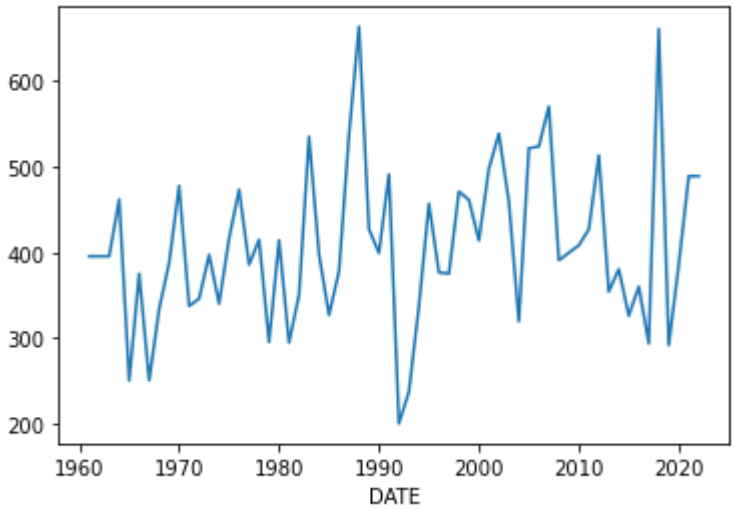
61 rows × 178 columns

In [87]:

```
#4.2
#Plot the CLDD column with time
GSY.groupby(['DATE']).mean()['CLDD'].plot()
```

Out[87]:

<AxesSubplot:xlabel='DATE'>



In [88]:

```
#4.3
#Observe the average value of CLDD
GSY['CLDD'].mean()
```

Out[88]:

407.7967213114755

In [89]:

```
#4.3
#Observe the sum value of CLDD
GSY['CLDD'].sum()
```

Out[89]:

24875.6

In [91]:

```
#4.3
#Observe the max value of CLDD
GSY['CLDD'].max()
```

Out[91]:

663.2

In [92]:

```
#4.3
#Observe the min value of CLDD
GSY['CLDD'].min()
```

Out[92]:

200.6

In [104]:

```
#4.3
#Observe the median of CLDD
l=GSY['CLDD'].count()
S=GSY['CLDD'].sort_values()
m=S[(l-1)/2]
print(m)
```

491.0

In [ ]:

```

```