Assignment02

import all the packages needed

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

Significant earthquakes since 2150 B.C.

Read the file

In [11]:
```
Sig_Eqs = pd.read_csv('earthquakes-2023-11-04_01-46-34_+0800.tsv', sep
Sig_Eqs.tail()
```

Out[11]:

	Search Parameters	Year	Mo	Dy	Hr	Country	Mag	Deaths
6394	NaN	2023.0	10.0	7.0	8.0	PAPUA NEW GUINEA	6.9	NaN
6395	NaN	2023.0	10.0	7.0	6.0	AFGHANISTAN	6.3	1480.0
6396	NaN	2023.0	10.0	8.0	20.0	JAPAN	4.9	NaN
6397	NaN	2023.0	10.0	11.0	0.0	AFGHANISTAN	6.3	3.0
6398	NaN	2023.0	10.0	15.0	3.0	AFGHANISTAN	6.3	4.0

1.1 [5 points] Compute the total number of deaths caused by earthquakes since 2150 B.C. in each country, and then print the top ten countries along with the total number of deaths.

First use the `groupby()` method to group earthquakes based on their respective countries, subsequently employ the `sum()` method to calculate the sum including the death number, and then employ the "sort_values" method to arrange them in descending order according to the number of fatalities in the dataset.

In [12]:
```
# Death number in each country
Death_num = Sig_Eqs.groupby('Country').sum().sort_values('Deaths', asc
print(Death_num['Deaths'].head(10))
```

```
Country
CHINA         2075045.0
TURKEY        1188881.0
IRAN          1011449.0
ITALY          498478.0
SYRIA          439224.0
HAITI          323478.0
AZERBAIJAN     317219.0
JAPAN          279085.0
ARMENIA        191890.0
PAKISTAN       145083.0
Name: Deaths, dtype: float64
```

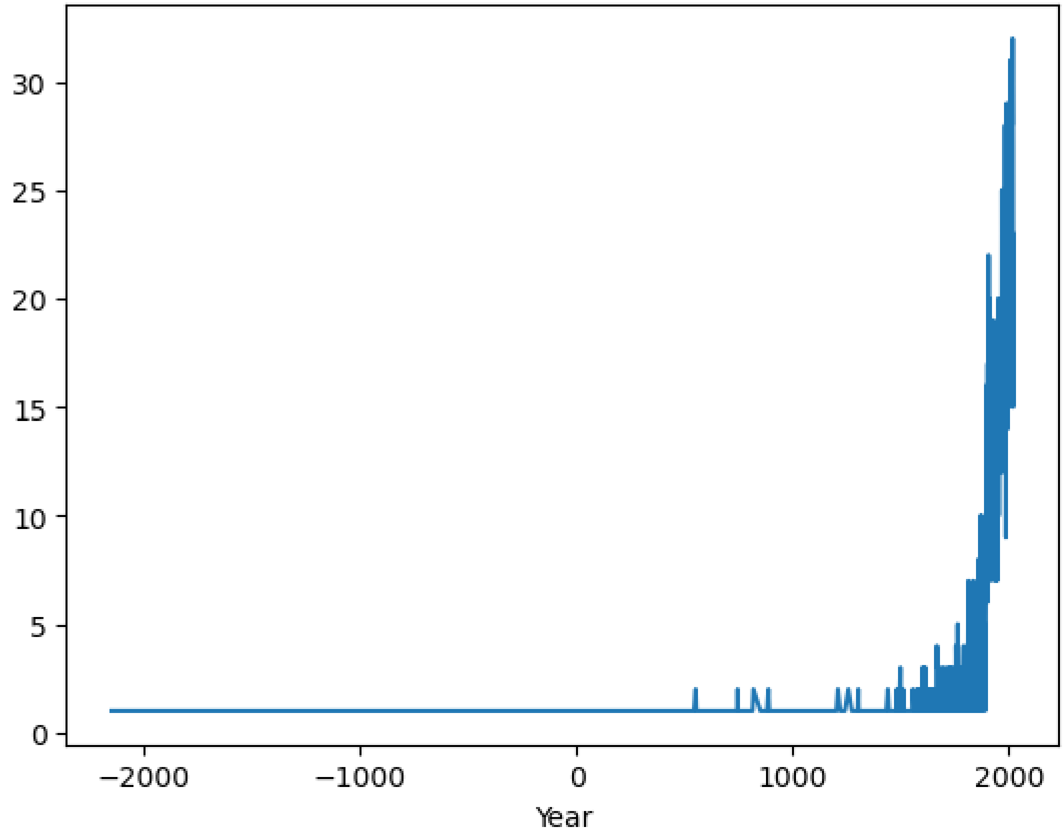
1.2 [10 points] Compute the total number of earthquakes with magnitude larger than 6.0 (use column Mag as the magnitude) worldwide each year, and then plot the time series. Do you observe any trend? Explain why or why not?

Use a new DataFrame `Big_Eqs` to record Earthquakes with magnitude larger than 6.0 and use `groupby()` method to get the number of Earthquakes per year. It can be seen that the number of earthquakes with magnitude larger than 6.0 is 2943, and the number has inreased a lot in recent centries.I hypothesize that this is due to an improvement in observation capabilities, resulting in a greater number of earthquakes being documented

In [13]:
```
Big_Eqs = Sig_Eqs.loc[Sig_Eqs['Mag'] > 6.0]

Big_Eqs.groupby('Year').describe()['Mag']['count'].plot()
```

Out[13]: <Axes: xlabel='Year'>



In [5]:
```
Big_Eqs.groupby('Year').describe()
```

...

In [14]:
```
Big_Eqs.shape
```

Out[14]: (2946, 8)

1.3 [10 points] Write a function `CountEq_LargestEq` that returns both (1) the total number of earthquakes since 2150 B.C. in a given country AND (2) the date of the largest earthquake ever happened in this country. Apply `CountEq_LargestEq` to every country in the file, report your results in a descending order.

First use `fillna()` method to replace all the nan in Mag by -1. Then classifying earthquake statistical data by country using the `groupby()` method. Counting the occurrence of earthquakes using by 'Count' of the 'Year' column, as this column has minimal missing values, allowing me to count all earthquake occurrences.

In [15]:
```
Sig_Eqs.fillna(-1, inplace = True)
Sig_Eqs
```

Out[15]:

	Search Parameters	Year	Mo	Dy	Hr	Country	Mag	Deaths
0	[]	-1.0	-1.0	-1.0	-1.0	-1	-1.0	-1.0
1	-1	-2150.0	-1.0	-1.0	-1.0	JORDAN	7.3	-1.0
2	-1	-2000.0	-1.0	-1.0	-1.0	SYRIA	-1.0	-1.0
3	-1	-2000.0	-1.0	-1.0	-1.0	TURKMENISTAN	7.1	1.0
4	-1	-1610.0	-1.0	-1.0	-1.0	GREECE	-1.0	-1.0
...	...	...	...	...	...	...	...	...
6394	-1	2023.0	10.0	7.0	8.0	PAPUA NEW GUINEA	6.9	-1.0
6395	-1	2023.0	10.0	7.0	6.0	AFGHANISTAN	6.3	1480.0
6396	-1	2023.0	10.0	8.0	20.0	JAPAN	4.9	-1.0
6397	-1	2023.0	10.0	11.0	0.0	AFGHANISTAN	6.3	3.0
6398	-1	2023.0	10.0	15.0	3.0	AFGHANISTAN	6.3	4.0

6399 rows × 8 columns

In [16]:
```
Sig_Eqs_Count_byYear = Sig_Eqs.groupby('Country')['Year'].count().sort
Sig_Eqs_Count_byYear = Sig_Eqs_Count_byYear.rename(columns={'Year':'Co
Sig_Eqs_Count_byYear
```

Out[16]:

	Country	Count
0	CHINA	620
1	JAPAN	414
2	INDONESIA	411
3	IRAN	384
4	TURKEY	335
...	...	...
152	SUDAN	1
153	SRI LANKA	1
154	NORWAY	1
155	PALAU	1
156	ZAMBIA	1

157 rows × 2 columns

Use `idxmax()` to get the max earthquake data

In [17]:
```
Lar_eq = Sig_Eqs.groupby('Country')['Mag'].idxmax()
Lar_eq = Sig_Eqs.loc[Lar_eq][['Year','Mo','Dy','Country']]
Lar_eq
```

Out[17]:

	Year	Mo	Dy	Country
6315	2021.0	10.0	12.0	-1
2727	1909.0	7.0	7.0	AFGHANISTAN
2395	1893.0	6.0	14.0	ALBANIA
4449	1980.0	10.0	10.0	ALGERIA
5010	1998.0	3.0	25.0	ANTARCTICA
...	...	...	...	...
2409	1894.0	4.0	29.0	VENEZUELA
3323	1935.0	11.0	1.0	VIETNAM
4864	1993.0	3.0	12.0	WALLIS AND FUTUNA (FRENCH TERRITORY)
4500	1982.0	12.0	13.0	YEMEN
6064	2017.0	2.0	24.0	ZAMBIA

157 rows × 4 columns

Merge the two dataset to get a new dataset that contain both Count of earthquake number and date of earthquake

In [18]:
```
eq_count_year = pd.merge(Sig_Eqs_Count_byYear, Lar_eq,on = ['Country']
eq_count_year = eq_count_year.set_index('Country')
eq_count_year
```

Out[18]:

	Count	Year	Mo	Dy
Country				
CHINA	620	1668.0	7.0	25.0
JAPAN	414	2011.0	3.0	11.0
INDONESIA	411	2004.0	12.0	26.0
IRAN	384	856.0	12.0	22.0
TURKEY	335	1939.0	12.0	26.0
...	...	...	...	...
SUDAN	1	1993.0	8.0	1.0
SRI LANKA	1	1882.0	1.0	-1.0
NORWAY	1	1819.0	8.0	31.0
PALAU	1	1914.0	10.0	23.0
ZAMBIA	1	2017.0	2.0	24.0

157 rows × 4 columns

Define the function, use the country name as the argument, search the value in the dataframe `eq_count_year` and print

In [19]:
```
def CountEq_LargestEq(Country):
    Country = Country.upper()
    count = eq_count_year.loc[Country]['Count']
    year = eq_count_year.loc[Country]['Year']
    month = eq_count_year.loc[Country]['Mo']
    day = eq_count_year.loc[Country]['Dy']
    print('The number of earthquake in ' + Country + ' is ' + str(int(
    print('P.S. if there are \'-1\' in the date, it means the data has
```

In [20]:
```
CountEq_LargestEq('SRI LANKA')
```

```
The number of earthquake in SRI LANKA is 1, the date of largest ear
thquake is 1882-1--1
P.S. if there are '-1' in the date, it means the data has lost
```

2. Wind speed in Shenzhen during the past 10 years

Read the file and keep only `DATE` and `WND` columns

In [3]:
```
wind_speed = pd.read_csv('2281305.csv')[['DATE','WND']]
```

```
C:\Users\fengx\AppData\Local\Temp\ipykernel_20236\3597857283.py:1:
DtypeWarning: Columns (4,8,9,12,15,21,22,24,26,31,33,34) have mixed
types. Specify dtype option on import or set low_memory=False.
  wind_speed = pd.read_csv('2281305.csv')[['DATE','WND']]
```

Use `split()` to split the `WND` column to get the data needed and then merge the two dataframe

In [4]:
```
temp = wind_speed['WND'].str.split(',', expand = True)
wind_speed = pd.merge(wind_speed, temp, left_index = True, right_index
```

Rename the columns

In [5]:
```
wind_speed = wind_speed.rename(columns={0:'WD', 1:'WD_QC', 2:'OB', 3:'

# Calcul the true wind speed
wind_speed['WS'] = wind_speed['WS'].astype(int).div(10)
```

Check if all the data with WS_QC = 9 has WS = 999.0

In [73]:
```
wind_speed.loc[(wind_speed['WS_QC'] == '9') & (wind_speed['WS'] != 999
```

Out[73]:

DATE	WND	WD	WD_QC	OB	WS	WS_QC

In [75]:
```
wind_speed.loc[(wind_speed['WS_QC'] != '9') & (wind_speed['WS'] == 999
```

Out[75]:

DATE	WND	WD	WD_QC	OB	WS	WS_QC

But the data with WD_QC = 9 dont necessarily need WS = 999.9

In [77]:
```
wind_speed.loc[wind_speed['WD_QC'] == '9']
```

Out[77]:

	DATE	WND	WD	WD_QC	OB	WS	WS_QC
1	2010-01-02T01:00:00	999,9,V,0010,1	999	9	V	1.0	1
2	2010-01-02T02:00:00	999,9,C,0000,1	999	9	C	0.0	1
16	2010-01-02T16:00:00	999,9,V,0010,1	999	9	V	1.0	1
25	2010-01-03T01:00:00	999,9,C,0000,1	999	9	C	0.0	1
26	2010-01-03T02:00:00	999,9,V,0010,1	999	9	V	1.0	1
...	...	...	...	...	...	...	...
111935	2020-09-09T23:00:00	999,9,V,0010,1	999	9	V	1.0	1
111937	2020-09-10T01:00:00	999,9,V,0010,1	999	9	V	1.0	1
111938	2020-09-10T02:00:00	999,9,V,0010,1	999	9	V	1.0	1
111960	2020-09-10T22:00:00	999,9,V,0010,1	999	9	V	1.0	1
111962	2020-09-11T00:00:00	999,9,V,0010,1	999	9	V	1.0	1

6363 rows × 7 columns

In [76]:
```
wind_speed.loc[(wind_speed['WD_QC'] == '9') & (wind_speed['WS'] != 999
```

Out[76]:

	DATE	WND	WD	WD_QC	OB	WS	WS_QC
1	2010-01-02T01:00:00	999,9,V,0010,1	999	9	V	1.0	1
2	2010-01-02T02:00:00	999,9,C,0000,1	999	9	C	0.0	1
16	2010-01-02T16:00:00	999,9,V,0010,1	999	9	V	1.0	1
25	2010-01-03T01:00:00	999,9,C,0000,1	999	9	C	0.0	1
26	2010-01-03T02:00:00	999,9,V,0010,1	999	9	V	1.0	1
...	...	...	...	...	...	...	...
111935	2020-09-09T23:00:00	999,9,V,0010,1	999	9	V	1.0	1
111937	2020-09-10T01:00:00	999,9,V,0010,1	999	9	V	1.0	1
111938	2020-09-10T02:00:00	999,9,V,0010,1	999	9	V	1.0	1
111960	2020-09-10T22:00:00	999,9,V,0010,1	999	9	V	1.0	1
111962	2020-09-11T00:00:00	999,9,V,0010,1	999	9	V	1.0	1

5725 rows × 7 columns

Change the date to datatime from string and set as index

In [7]:
```
wind_speed['date'] = pd.to_datetime(wind_speed['DATE'])
wind_speed = wind_speed.drop(columns = ['DATE'])
wind_speed = wind_speed.set_index('date')
```

In [8]:
```
wind_speed_fil = wind_speed.loc[wind_speed['WS'] != 999.9]
wind_speed_fil
```

Out[8]:

date	WND	WD	WD_QC	OB	WS	WS_QC
2010-01-02 00:00:00	040,1,N,0020,1	040	1	N	2.0	1
2010-01-02 01:00:00	999,9,V,0010,1	999	9	V	1.0	1
2010-01-02 02:00:00	999,9,C,0000,1	999	9	C	0.0	1
2010-01-02 03:00:00	140,1,N,0010,1	140	1	N	1.0	1
2010-01-02 04:00:00	300,1,N,0040,1	300	1	N	4.0	1
...	...	...	...	...	...	...
2020-09-11 17:00:00	170,1,N,0030,1	170	1	N	3.0	1
2020-09-11 18:00:00	180,1,N,0040,1	180	1	N	4.0	1
2020-09-11 19:00:00	220,1,V,0030,1	220	1	V	3.0	1
2020-09-11 20:00:00	260,1,N,0030,1	260	1	N	3.0	1
2020-09-11 21:00:00	310.1.V.0020.1	310	1	V	2.0	1

Use `resample()` and `mean()` to get the monthly mean wind speed

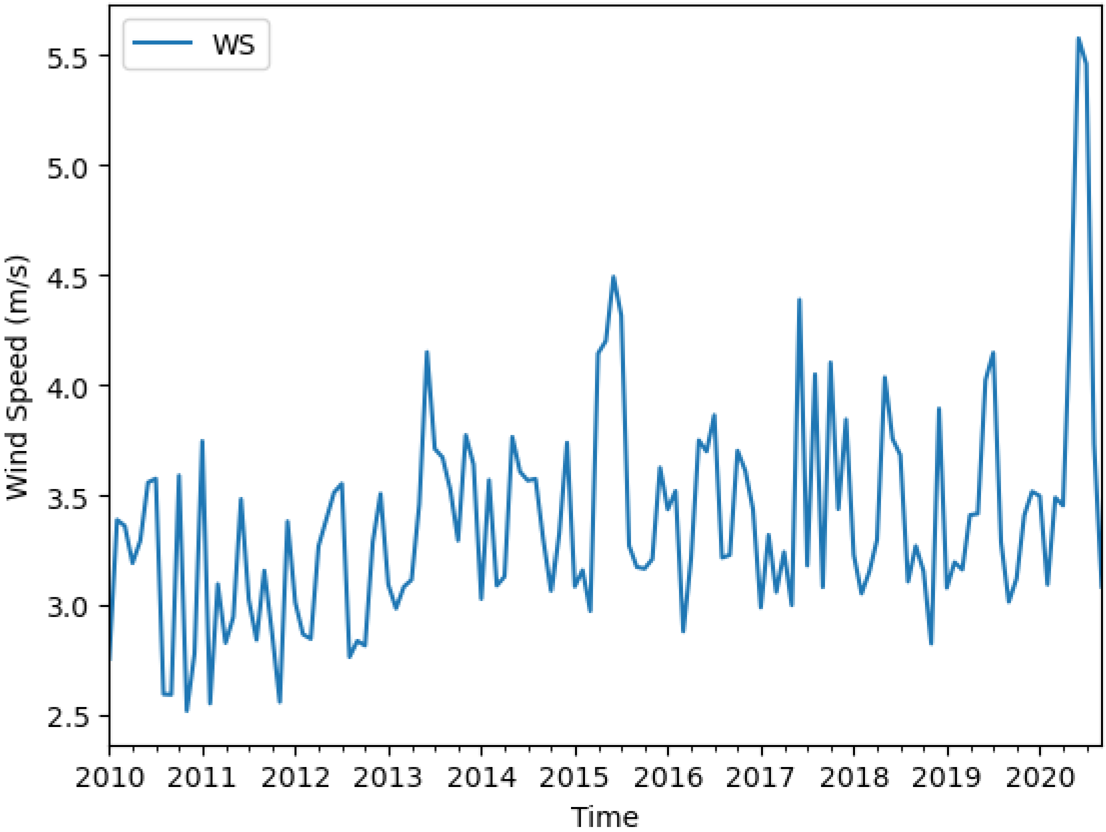
In [22]:
```
monthly_wind_speed = wind_speed_fil.resample('M')['WS'].mean()
monthly_wind_speed
```

Out[22]:
```
date
2010-01-31    2.756267
2010-02-28    3.388060
2010-03-31    3.360700
2010-04-30    3.191341
2010-05-31    3.293640
                ...
2020-05-31    4.362198
2020-06-30    5.575800
2020-07-31    5.459140
2020-08-31    3.733608
2020-09-30    3.085019
Freq: M, Name: WS, Length: 129, dtype: float64
```

plot the result

In [24]:
```
monthly_wind_speed.plot()
plt.xlabel('Time')
plt.ylabel('Wind Speed (m/s)')
plt.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x27a30d5e500>



3. Explore a data set

I choose the data from 中国环境监测总站及香港环保署全国空气质量数据 and choose 20190316 as the dataset to explore as this dataset has sereral missing value

In [2]:
```
poll_data = pd.read_csv('china_cities_20190316.csv')
poll_data
```

Out[2]:

	date	hour	type	北京	天津	石家庄	唐山	秦皇岛	邯郸	保定	...	阿苏
0	20190316	0	AQI	66.00	41.00	80.00	43.00	50.00	65.00	63.00	...	190.
1	20190316	0	PM2.5	29.00	19.00	37.00	21.00	27.00	37.00	30.00	...	83.
2	20190316	0	PM2.5_24h	11.00	12.00	23.00	17.00	17.00	27.00	19.00	...	58.
3	20190316	0	PM10	82.00	41.00	109.00	43.00	50.00	79.00	75.00	...	330.
4	20190316	0	PM10_24h	48.00	48.00	79.00	40.00	47.00	83.00	56.00	...	198.
...	...	...	...	...	...	...	...	...	...	...	...	.
355	20190316	23	O3_24h	87.00	95.00	105.00	94.00	96.00	113.00	103.00	...	176.
356	20190316	23	O3_8h	68.00	48.00	77.00	56.00	58.00	72.00	66.00	...	114.
357	20190316	23	O3_8h_24h	76.00	77.00	95.00	80.00	83.00	107.00	85.00	...	130.
358	20190316	23	CO	0.28	1.34	0.66	1.25	1.34	0.70	0.83	...	1.
359	20190316	23	CO_24h	0.52	0.89	0.63	1.67	1.28	1.13	0.57	...	0.

360 rows × 370 columns

It can be seen that several columns has missing values, use `dropna(axis = 1)` to drop the columns has missing values

In [3]:
```
poll_data_cleaned = poll_data.dropna(axis = 1)
poll_data_cleaned
```

Out[3]:

	date	hour	type	北京	天津	石家庄	唐山	秦皇岛	邯郸	保定	...	海
0	20190316	0	AQI	66.00	41.00	80.00	43.00	50.00	65.00	63.00	...	29.
1	20190316	0	PM2.5	29.00	19.00	37.00	21.00	27.00	37.00	30.00	...	19.
2	20190316	0	PM2.5_24h	11.00	12.00	23.00	17.00	17.00	27.00	19.00	...	15.
3	20190316	0	PM10	82.00	41.00	109.00	43.00	50.00	79.00	75.00	...	29.
4	20190316	0	PM10_24h	48.00	48.00	79.00	40.00	47.00	83.00	56.00	...	43.
...	...	...	...	...	...	...	...	...	...	...	...	.
355	20190316	23	O3_24h	87.00	95.00	105.00	94.00	96.00	113.00	103.00	...	105.
356	20190316	23	O3_8h	68.00	48.00	77.00	56.00	58.00	72.00	66.00	...	97.
357	20190316	23	O3_8h_24h	76.00	77.00	95.00	80.00	83.00	107.00	85.00	...	101.
358	20190316	23	CO	0.28	1.34	0.66	1.25	1.34	0.70	0.83	...	0.
359	20190316	23	CO_24h	0.52	0.89	0.63	1.67	1.28	1.13	0.57	...	0.

360 rows × 308 columns

From the dimension, it can be seen that 62 columns with missing has been deleted from the dataframe

Then creat a new dataframe only contain O3 concentration

In [4]:
```
O3_conc = poll_data_cleaned.loc[(poll_data_cleaned['hour'] == 1) & (po
for i in range(1,24):
    temp = poll_data_cleaned.loc[(poll_data_cleaned['hour'] == i) & (p
    O3_conc = pd.concat([O3_conc, temp], ignore_index=True)
```

Show the data to check if the function get the right result

In [5]:
```
O3_conc
```

Out[5]:

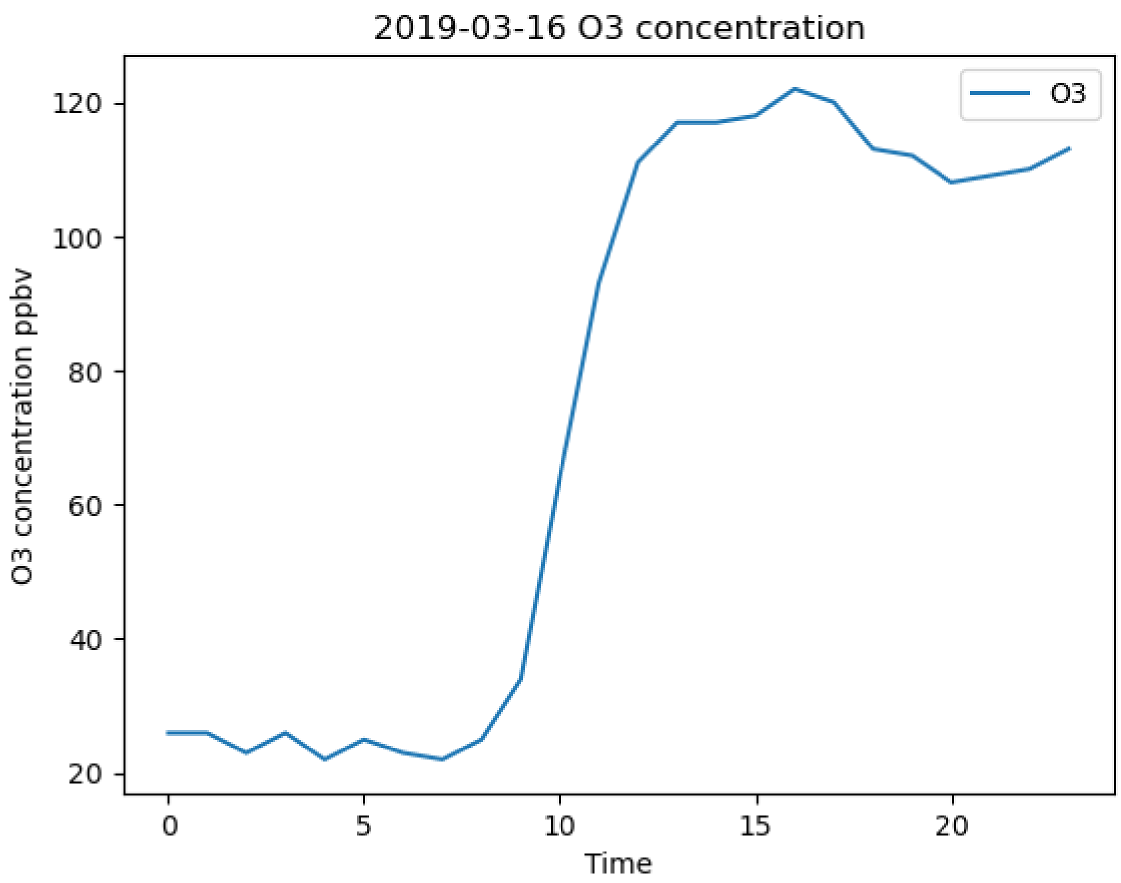
	date	hour	type	北京	天津	石家庄	唐山	秦皇岛	邯郸	保定	...	海西州	吴忠	中卫
0	20190316	1	O3	25.0	43.0	25.0	46.0	27.0	13.0	48.0	...	83.0	75.0	45.0
1	20190316	1	O3	25.0	43.0	25.0	46.0	27.0	13.0	48.0	...	83.0	75.0	45.0
2	20190316	2	O3	13.0	36.0	21.0	50.0	17.0	8.0	39.0	...	85.0	57.0	48.0
3	20190316	3	O3	8.0	27.0	13.0	49.0	15.0	5.0	34.0	...	87.0	44.0	47.0
4	20190316	4	O3	11.0	18.0	6.0	29.0	14.0	7.0	33.0	...	86.0	34.0	56.0
5	20190316	5	O3	11.0	12.0	4.0	9.0	14.0	18.0	25.0	...	88.0	46.0	53.0
6	20190316	6	O3	9.0	8.0	7.0	8.0	17.0	12.0	12.0	...	90.0	55.0	49.0
7	20190316	7	O3	7.0	7.0	20.0	7.0	13.0	10.0	9.0	...	84.0	46.0	55.0
8	20190316	8	O3	10.0	14.0	16.0	12.0	16.0	28.0	22.0	...	87.0	30.0	54.0
9	20190316	9	O3	20.0	28.0	25.0	21.0	31.0	47.0	39.0	...	84.0	37.0	53.0
10	20190316	10	O3	33.0	44.0	42.0	28.0	48.0	68.0	50.0	...	89.0	52.0	62.0
11	20190316	11	O3	39.0	54.0	57.0	42.0	54.0	88.0	56.0	...	92.0	65.0	66.0
12	20190316	12	O3	44.0	64.0	77.0	54.0	63.0	101.0	63.0	...	92.0	81.0	86.0
13	20190316	13	O3	57.0	75.0	89.0	72.0	72.0	104.0	71.0	...	94.0	97.0	102.0
14	20190316	14	O3	75.0	83.0	95.0	86.0	81.0	108.0	81.0	...	99.0	108.0	112.0
15	20190316	15	O3	84.0	91.0	101.0	94.0	88.0	111.0	95.0	...	102.0	110.0	106.0
16	20190316	16	O3	86.0	91.0	104.0	93.0	94.0	113.0	103.0	...	103.0	108.0	104.0
17	20190316	17	O3	86.0	85.0	106.0	92.0	95.0	112.0	102.0	...	104.0	111.0	102.0
18	20190316	18	O3	82.0	69.0	104.0	82.0	90.0	107.0	97.0	...	105.0	113.0	102.0
19	20190316	19	O3	75.0	40.0	87.0	52.0	77.0	81.0	72.0	...	101.0	107.0	90.0
20	20190316	20	O3	61.0	35.0	62.0	62.0	48.0	39.0	55.0	...	100.0	87.0	68.0
21	20190316	21	O3	48.0	31.0	52.0	43.0	26.0	27.0	37.0	...	92.0	63.0	64.0
22	20190316	22	O3	56.0	26.0	56.0	19.0	20.0	46.0	34.0	...	82.0	41.0	52.0
23	20190316	23	O3	53.0	18.0	57.0	16.0	14.0	49.0	25.0	...	92.0	20.0	32.0

24 rows × 308 columns

Use `plot()` to plot the time series of O3 concentration in Shenzhen

In [6]:
```
O3_conc['深圳'].plot()
plt.xlabel('Time')
plt.ylabel('O3 concentration ppbv')
plt.title('2019-03-16 O3 concentration')
plt.legend(labels = ['O3'])
```

Out[6]:
```
<matplotlib.legend.Legend at 0x28330bec9d0>
```



Use `max()` to get the maximum value of O3 concentration in each city

In [13]:
```
O3_max = O3_conc.drop(['type', 'date', 'hour'], axis=1).max()
O3_max
```

Out[13]:
```
北京          86.0
天津          91.0
石家庄        106.0
唐山          94.0
秦皇岛         95.0
             ...
昌吉州         29.0
阿克苏地区      176.0
和田地区        80.0
伊犁哈萨克州    101.0
五家渠         26.0
Length: 305, dtype: float64
```

In [31]:
```
print(O3_max.sort_values(ascending=False).head(10))
```

```
江门       196.0
广州       193.0
台州       182.0
阿克苏地区    176.0
太仓       169.0
东莞       167.0
西双版纳州    166.0
义乌       165.0
信阳       165.0
上海       159.0
dtype: float64
```

Use min() to get the mean value of O3 concentration in each city

In [32]:
```
O3_min = O3_conc.drop(['type', 'date', 'hour'], axis=1).min()
O3_min.sort_values().head(10)
```

Out[32]:
```
五家渠    2.0
金华     2.0
巴中     2.0
中山     2.0
眉山     3.0
吕梁     3.0
邢台     3.0
长沙     3.0
佛山     3.0
咸阳     3.0
dtype: float64
```

Use mean() to get the mean value of O3 concentration in each city

In [33]:
```
O3_mean = O3_conc.drop(['type', 'date', 'hour'], axis=1).mean()
O3_mean.sort_values(ascending=False).head(10)
```

Out[33]:
```
莆田      109.416667
大理州     104.500000
阿拉善盟    104.458333
厦门      102.791667
湛江      101.083333
迪庆州      99.833333
茂名       99.500000
泉州       99.208333
保山       98.500000
汕头       97.583333
dtype: float64
```

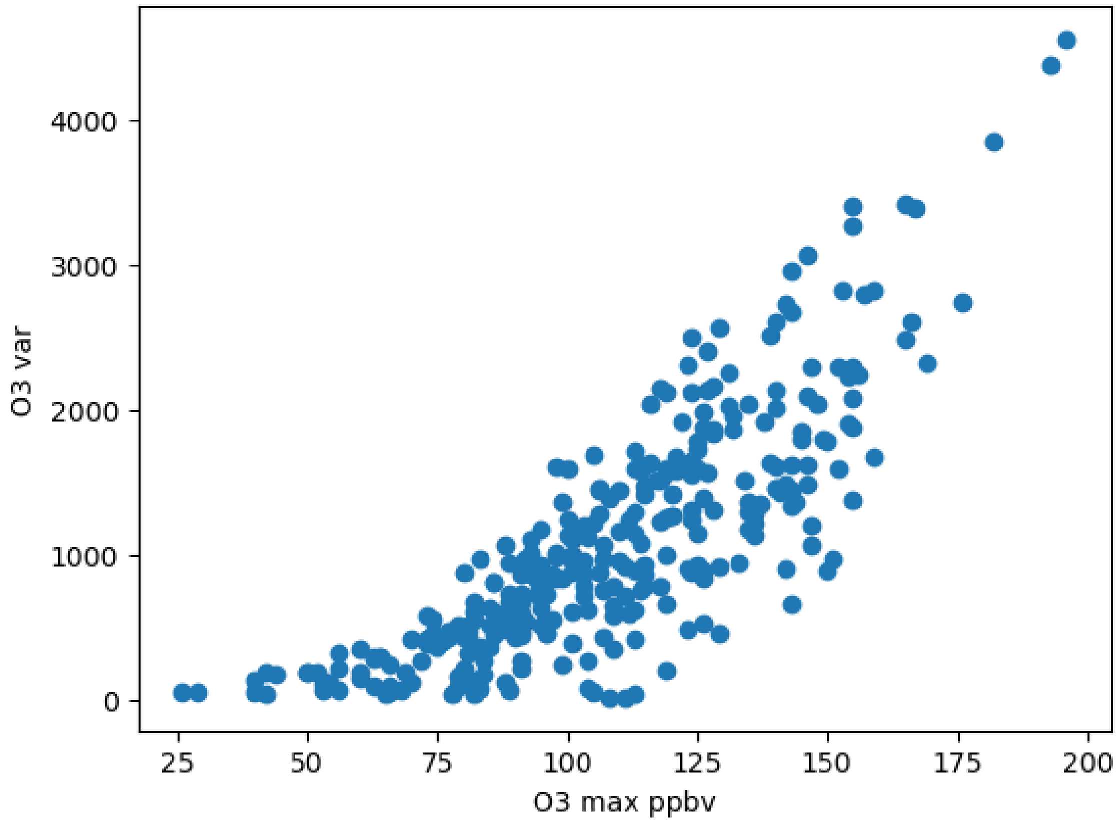
In [34]:
```
O3_var = O3_conc.drop(['type', 'date', 'hour'], axis=1).var()
O3_var.sort_values(ascending=False).head(10)
```

Out[34]:
```
江门    4552.737319
广州    4380.688406
台州    3845.673913
义乌    3421.606884
金华    3399.326087
东莞    3385.201087
句容    3267.070652
徐州    3065.809783
随州    2958.253623
佛山    2827.391304
dtype: float64
```

It can be seen that the city with the highest O3_max experiences substantial overlap with the city having the highest O3_var, whereas there is little correlation between cities with the highest mean O3 concentrations and these two factors. I speculate that this may be attributed to the possibility that cities with high mean O3 concentrations could be enduring a prolonged pollution episode, whereas cities with high variances and max values may experience brief pollution events primarily due to other factors, occurring during the peak of photochemical activity in the late afternoons

In [26]:
```
plt.scatter(O3_max, O3_var)
plt.xlabel('O3 max ppbv')
plt.ylabel('O3 var')
```

Out[26]:
```
Text(0, 0.5, 'O3 var')
```



Calculate the correlation between O3 max and O3 var. The result shows a strong correlation. This is because the lowest concentration of O3 in a day is typically near 0, so when the maximum concentration of O3 is high, the variance will also increase, resulting in a large variance in O3.

In [27]:
```
corr = np.corrcoef(O3_max, O3_var)[0,1]
print(corr)
```

```
0.8150119959666451
```