PS2

1. Significant earthquakes since 2150 B.C.

The **Significant Earthquake Database** contains information on destructive earthquakes from 2150 B.C. to the present. On the top left corner, select all columns and download the entire significant earthquake data file in .tsv format by clicking the `Download TSV File` button. Click the variable name for more information. Read the file (e.g., `earthquakes-2022-10-18_09-17-48_+0800.tsv`) as an object and name it `Sig_Eqs`.

In [1]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns',None)

Sig_Eqs = pd.read_csv('data_files\earthquakes-2022-10-29_09-50-02_+0800.tsv',
                      skiprows=[1],sep='\t')
Sig_Eqs.head()
```

Out[1]:

	Search Parameters	Year	Mo	Dy	Hr	Mn	Sec	Tsu	Vol	Country	Area	Region	Location Name
0	NaN	-2150	NaN	NaN	NaN	NaN	0.0	NaN	NaN	JORDAN	NaN	140	JORDAN: BAB-A-DARAA,AL-KARAK
1	NaN	-2000	NaN	NaN	NaN	NaN	NaN	1.0	NaN	SYRIA	NaN	130	SYRIA: UGARIT
2	NaN	-2000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	TURKMENISTAN	NaN	40	TURKMENISTAN: W
3	NaN	-1610	NaN	NaN	NaN	NaN	NaN	3.0	1351.0	GREECE	NaN	130	GREECE: THERA ISLAND (SANTORINI)
4	NaN	-1566	NaN	NaN	NaN	NaN	0.0	NaN	NaN	ISRAEL	NaN	140	ISRAEL: ARIHA (JERICHO)

1.1

Compute the total number of deaths caused by earthquakes since 2150 B.C. in each country, and then print the top 20 countries along with the total number of deaths.

Solution:

In [2]:
```
Sig_Eqs.groupby('Country')['Total Deaths'].sum().sort_values(ascending=0).head(20)
```

Out[2]:
```
Country
CHINA           2041903.0
TURKEY           927459.0
IRAN             758647.0
SYRIA            437700.0
ITALY            422678.0
JAPAN            355140.0
HAITI            323772.0
AZERBAIJAN       310119.0
INDONESIA        282153.0
ARMENIA          189000.0
PAKISTAN         143712.0
ECUADOR          134428.0
TURKMENISTAN     110412.0
PERU              96161.0
PORTUGAL          82531.0
GREECE            80271.0
IRAQ              70200.0
CHILE             70174.0
INDIA             62396.0
TAIWAN            57705.0
Name: Total Deaths, dtype: float64
```

1.2

Compute the total number of earthquakes with magnitude larger than 3.0 (use column `Ms` as the magnitude) worldwide each year, and then plot the time series. Do you observe any trend? Explain why or why not?

Solutions:

In [3]:
```
import matplotlib.pyplot as plt
```

In [4]:
```
%matplotlib inline
```

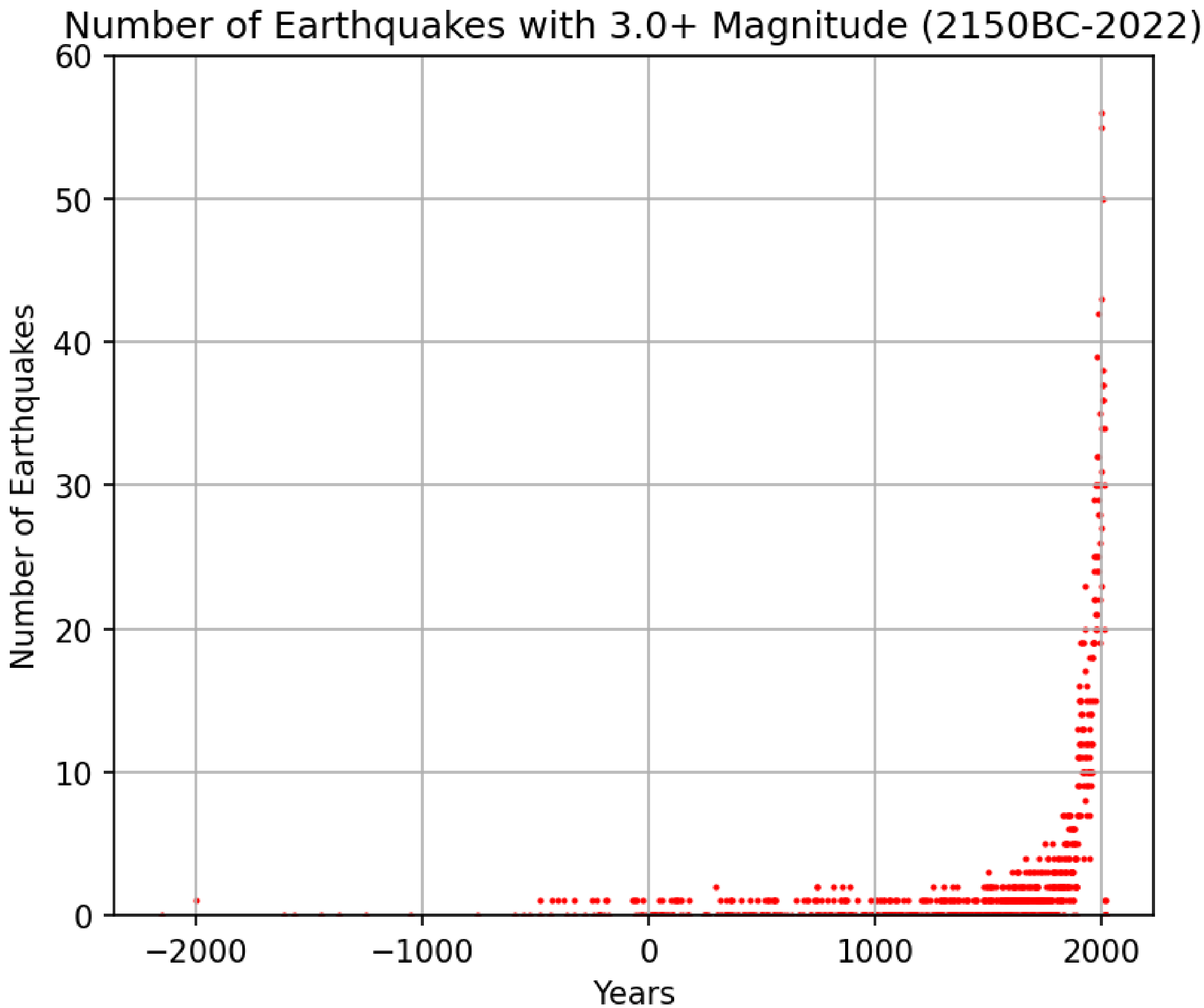
In [5]:
```
def mag_filter(mag):
    if mag>3.0:
        return 1
    else:
        return 0
Sig_Eqs['Mag Count'] = Sig_Eqs['Ms'].map(mag_filter)
Sig_Eqs.head()
```

Out[5]:

	Search Parameters	Year	Mo	Dy	Hr	Mn	Sec	Tsu	Vol	Country	Area	Region	Location Name
0	NaN	-2150	NaN	NaN	NaN	NaN	0.0	NaN	NaN	JORDAN	NaN	140	JORDAN: BAB-A-DARAA,AL-KARAK
1	NaN	-2000	NaN	NaN	NaN	NaN	NaN	1.0	NaN	SYRIA	NaN	130	SYRIA: UGARIT
2	NaN	-2000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	TURKMENISTAN	NaN	40	TURKMENISTAN: W
3	NaN	-1610	NaN	NaN	NaN	NaN	NaN	3.0	1351.0	GREECE	NaN	130	GREECE: THERA ISLAND (SANTORINI)
4	NaN	-1566	NaN	NaN	NaN	NaN	0.0	NaN	NaN	ISRAEL	NaN	140	ISRAEL: ARIHA (JERICHO)

In [6]:
```
magct = Sig_Eqs.groupby('Year')['Mag Count'].sum()
```

In [78]:
```
fig, ax = plt.subplots(figsize=(6,5),dpi=150)
ax.scatter(magct.index,magct,color='r',s = 1)
ax.set(title = 'Number of Earthquakes with 3.0+ Magnitude (2150BC-2022)',
       xlabel = 'Years',
       ylabel = 'Number of Earthquakes',
       ylim = (0,60))
ax.grid()
```



TREND: There are seldom earthquakes records with a magnitude larger than 3.0 before AC1000. The world has seen a sharp increase on number of earthquakes since AC1500 each year, from less than 10 times a year to more than 70 times a year. It may reflect a more frequent crustal activity. However, wider-spreaded and more acurate earthquake detections may also contribute to this trend.

1.3

Write a function `CountEq_LargestEq` that returns (1) the total number of earthquakes since 2150 B.C. in a given country AND (2) date and location 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.

Solution:

In [8]:
```
# Define a function that returns a DataFrame
# that contains tol., mag, date, and location info...
def CountEq_LargestEq(country):
    tol_eqs = Sig_Eqs.groupby('Country')['Mag Count'].count()[country] #Total Earthequakes
    ctry_eqs = Sig_Eqs.loc[Sig_Eqs['Country']==country] #Select all records of a certain country
    max_mag = ctry_eqs['Mag'].max(0) #Find the maximum magnitude
    col_name = ['Country','Year','Mo','Dy', #Select columns of interests
                'Mag','Location Name','Latitude','Longitude']
    info = ctry_eqs.loc[ctry_eqs['Mag']==max_mag][col_name]
    col_name.insert(col_name.index('Year'),'Tol. Earthquakes')
```

```python
    info = info.reindex(columns = col_name) #Insert column 'Tol. Earthquakes' in
    info['Tol. Earthquakes'] = tol_eqs
    info.rename(columns = {'Mag':'Max Magnitude'}, inplace=True)
    return info
```

In [9]:
```python
# A simple test
CountEq_LargestEq('CHINA')
```

Out[9]:

	Country	Tol. Earthquakes	Year	Mo	Dy	Max Magnitude	Location Name	Latitude	Longitude
977	CHINA	616	1668	7.0	25.0	8.5	CHINA: SHANDONG PROVINCE	35.3	118.6

In [10]:
```python
Sig_Eqs['Country']
```

Out[10]:
```
0             JORDAN
1              SYRIA
2       TURKMENISTAN
3             GREECE
4             ISRAEL
            ...
6332          MEXICO
6333          MEXICO
6334       INDONESIA
6335            IRAN
6336            PERU
Name: Country, Length: 6337, dtype: object
```

In [11]:
```python
# Make a list of all countries
all_country = np.unique(list(Sig_Eqs['Country']))
# Delete the 'nan' value
all_country = np.delete(all_country,-1)
```

In [12]:
```python
# Concatenate each outputs of the function as a DataFrame
result = pd.DataFrame()
for i in all_country:
    entry = CountEq_LargestEq(i)
    result = pd.concat([result,entry])
```

In [13]:
```python
# Reset the indexes
result = result.reset_index(drop=1)
```

In [14]:
```python
# Combine year, month and date
y = result['Year'].map(str)
m = result['Mo'].map(str).apply(lambda x:x[:-2])
d = result['Dy'].map(str).apply(lambda x:x[:-2])
result['Date'] = y + '-' + m + '-' + d
# Delete abundant columns
result.drop(['Year','Mo','Dy'],axis=1)
```

Out[14]:

	Country	Tol. Earthquakes	Max Magnitude	Location Name	Latitude	Longitude	Date
0	AFGHANISTAN	62	8.1	AFGHANISTAN: HINDU-KUSH	36.500	70.500	1909-7-7
1	ALBANIA	56	7.5	ALBANIA: HIMARA, DHERMI, KUC, KUDHESI, VLORE, ...	40.200	19.700	1893-6-14
2	ALGERIA	57	7.1	ALGERIA: NORTHERN	36.199	1.374	1980-10-10
3	ANTARCTICA	5	8.1	BALLENY ISLANDS	-62.877	149.527	1998-3-25
4	ANTIGUA AND BARBUDA	3	8.0	ANTIGUA; SAINT KITTS AND NEVIS	17.500	-61.500	1690-4-16
...	...	...	...	...	...	...	...
165	VENEZUELA	66	8.2	VENEZUELA: MERIDA,TOVAR; COLOMBIA: N SANTANDER	8.500	-71.700	1894-4-29
166	VIETNAM	5	6.8	VIETNAM: DIEN BIEN PHU	21.185	103.770	1935-11-1
167	WALLIS AND FUTUNA (FRENCH TERRITORY)	1	6.4	FUTUNA ISLAND	-14.385	-178.252	1993-3-12
168	YEMEN	10	6.0	YEMEN: DHAMAR	14.701	44.379	1982-12-13
169	ZAMBIA	1	5.9	ZAMBIA: KAPUTA	-8.440	30.031	2017-2-24

170 rows × 7 columns

2. Air temperature in Shenzhen during the past 25 years

In this problem set, we will examine how air temperature changes in Shenzhen during the past 25 years using the hourly weather data measured at the BaoAn International Airport. The data set is from `NOAA Integrated Surface Dataset`. Download the file `Baoan_Weather_1998_2022.csv`, move the `.csv` file to your `working directory`.

Read page `10-11` (`POS 88-92` and `POS 93-93`) of the comprehensive `user guide` for the detailed format of the air temperature data (use column `TMP`). Explain how you filter the data in your report.

METHOD:

The user guide explains the format of the air temperature data recoreded in the file. One typical record of temperature is

$$+0270, 1$$

The data is comprised of three parts. The first one is the sign that can be seen as part of the value it self. The second part is the temperature value. In this case, it is 0270, with a scaling factor 10, representing +27 degrees celsius. The last part is the air temperature quality code, separated by a comma, placed at the rear of the record. The user guide provides with a detailed interpretion of the quality code. But normally, it remains 1, claiming the temperature data has "passed all quality control checks".

We only focus on the value of temperature. So we need to extract the value from the record and store the quality code elsewhere. A extra step is needed to take, converting the raw value into real temperature value using the scaling factor.

Another data we will use is the time information, which is stored in the `DATE` column containing both date and acurate time. The temperature is measured at an hourly step. Years, months and day are separated by one single hyphen while time elements are separated by colon with a capitalized T at the beginning.

TASK: Plot monthly averaged air temperature against the observation time. Is there a trend in monthly averaged air temperature in the past 25 years?

Solution:

In [15]:
```
# We only need to read columns we are interested in
# that are date and tmp
import pandas as pd
tmp = pd.read_csv('data_files\Baoan_Weather_1998_2022.csv',usecols=['DATE','TMP'])
tmp
```

Out[15]:

	DATE	TMP
0	1998-01-01T00:00:00	+0186,1
1	1998-01-01T01:00:00	+0220,1
2	1998-01-01T02:00:00	+0240,1
3	1998-01-01T03:00:00	+0221,1
4	1998-01-01T04:00:00	+0240,1
...	...	...
235669	2022-10-10T20:00:00	+0210,1
235670	2022-10-10T21:00:00	+0201,1
235671	2022-10-10T21:00:00	+0200,1
235672	2022-10-10T22:00:00	+0200,1
235673	2022-10-10T23:00:00	+0200,1

235674 rows × 2 columns

In [16]:
```
# Use to_datetime to interpret original date info
tmp['YEAR'] = pd.to_datetime(tmp['DATE']).dt.year
tmp['MONTH'] = pd.to_datetime(tmp['DATE']).dt.month
tmp
```

Out[16]:

	DATE	TMP	YEAR	MONTH
0	1998-01-01T00:00:00	+0186,1	1998	1
1	1998-01-01T01:00:00	+0220,1	1998	1
2	1998-01-01T02:00:00	+0240,1	1998	1
3	1998-01-01T03:00:00	+0221,1	1998	1
4	1998-01-01T04:00:00	+0240,1	1998	1
...	...	...	...	...
235669	2022-10-10T20:00:00	+0210,1	2022	10
235670	2022-10-10T21:00:00	+0201,1	2022	10
235671	2022-10-10T21:00:00	+0200,1	2022	10
235672	2022-10-10T22:00:00	+0200,1	2022	10
235673	2022-10-10T23:00:00	+0200,1	2022	10

235674 rows × 4 columns

In [17]:

```
# Split the column TMP into two columns
# 'temp value' and 'quality code'
tmp = tmp.join(tmp['TMP'].str.split(',',1, expand=True))\
    .rename(columns={0:'temp value',1:'QUALITY CODE'})
```

In [18]:

```
# Check the code for abnormal values
abnormal = tmp.loc[tmp['QUALITY CODE']!='1']\
    .sort_values('QUALITY CODE')['QUALITY CODE']
abnormal.unique()
```

Out[18]:

```
array(['2', '5', '9'], dtype=object)
```

In [19]:

```
# Code 2 for suspect values
# Code 5 for value from NCEI
# Code 9 for passed values if value presents
# But in our case code 9 are just missing values
tmp.loc[tmp['QUALITY CODE']=='9']
```

Out[19]:

	DATE	TMP	YEAR	MONTH	temp value	QUALITY CODE
1075	1998-02-24T18:00:00	+9999,9	1998	2	+9999	9
2118	1998-04-17T01:00:00	+9999,9	1998	4	+9999	9
2379	1998-04-29T08:00:00	+9999,9	1998	4	+9999	9
2384	1998-04-29T13:00:00	+9999,9	1998	4	+9999	9
3525	1998-06-22T23:00:00	+9999,9	1998	6	+9999	9
...	...	...	...	...	...	...
145761	2014-08-22T15:00:00	+9999,9	2014	8	+9999	9
145765	2014-08-22T18:00:00	+9999,9	2014	8	+9999	9
145777	2014-08-23T03:00:00	+9999,9	2014	8	+9999	9
145781	2014-08-23T06:00:00	+9999,9	2014	8	+9999	9
145785	2014-08-23T09:00:00	+9999,9	2014	8	+9999	9

797 rows × 6 columns

In [20]:

```
# We can tolerate suspected data, so we
# only remove missing data
tmp = tmp.loc[tmp['QUALITY CODE']!='9']
# Introduce the scaling factor to obtain real temperature values
tmp['TEMPERATURE'] = tmp['temp value'].map(int).apply(lambda x:x/10)
# Concatenate year and month for 'groupby' later
tmp['MONTHS'] = tmp['YEAR'].map(str) + '-' + tmp['MONTH'].map(str)
tmp
```

Out[20]:

	DATE	TMP	YEAR	MONTH	temp value	QUALITY CODE	TEMPERATURE	MONTHS
0	1998-01-01T00:00:00	+0186,1	1998	1	+0186	1	18.6	1998-1
1	1998-01-01T01:00:00	+0220,1	1998	1	+0220	1	22.0	1998-1
2	1998-01-01T02:00:00	+0240,1	1998	1	+0240	1	24.0	1998-1
3	1998-01-01T03:00:00	+0221,1	1998	1	+0221	1	22.1	1998-1
4	1998-01-01T04:00:00	+0240,1	1998	1	+0240	1	24.0	1998-1
...	...	...	...	...	...	...	...	...
235669	2022-10-10T20:00:00	+0210,1	2022	10	+0210	1	21.0	2022-10
235670	2022-10-10T21:00:00	+0201,1	2022	10	+0201	1	20.1	2022-10
235671	2022-10-10T21:00:00	+0200,1	2022	10	+0200	1	20.0	2022-10
235672	2022-10-10T22:00:00	+0200,1	2022	10	+0200	1	20.0	2022-10
235673	2022-10-10T23:00:00	+0200,1	2022	10	+0200	1	20.0	2022-10

234877 rows × 8 columns

In [21]:

```
# Get the monthly averaged temperatures
ave = tmp.groupby('MONTHS').mean().sort_values(['YEAR','MONTH'])
print(len(ave))
ave.head()
```

298

Out[21]:

MONTHS	YEAR	MONTH	TEMPERATURE
1998-1	1998.0	1.0	15.233447
1998-2	1998.0	2.0	16.875304
1998-3	1998.0	3.0	19.971246
1998-4	1998.0	4.0	25.228365
1998-5	1998.0	5.0	27.098454

In [22]:

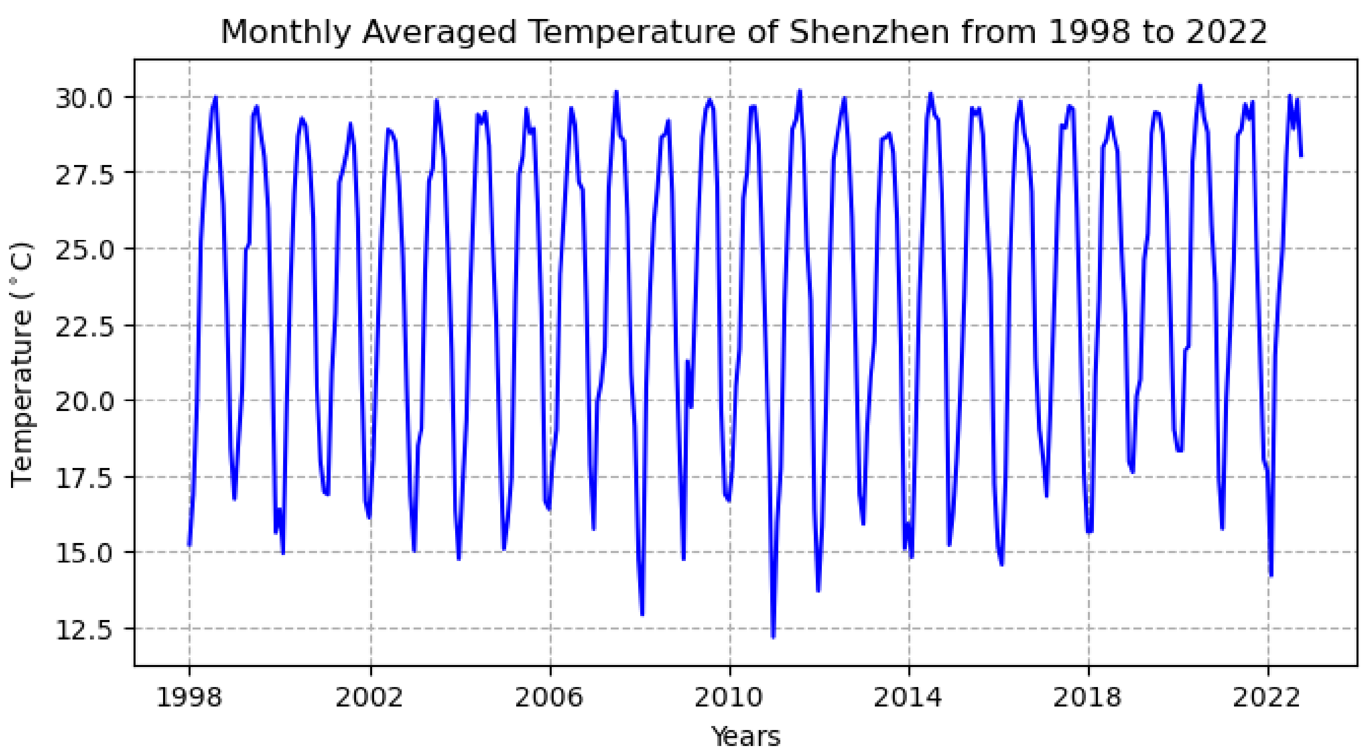
```
# Calculate the distance between each x ticks
dis = np.append(np.arange(0,288,48),288)
dis
```

Out[22]:

```
array([  0,  48,  96, 144, 192, 240, 288])
```

In [23]:

```
# Plot the figure
fig, ax = plt.subplots(figsize=(8,4),dpi=100)
ax.set_title('Monthly Averaged Temperature of Shenzhen from 1998 to 2022')
ax.plot(ave.index,ave['TEMPERATURE'],color = 'b')
ax.set_xlabel('Years')
ax.set_ylabel('Temperature '+r'$(^\circ$C)')
ax.xaxis.set_ticks(dis,np.arange(1998,2023,4))
ax.grid(ls = 'dashed')
```



3. Global collection of hurricanes

The International Best Track Archive for Climate Stewardship (IBTrACS) project is the most complete global collection of tropical cyclones available. It merges recent and historical tropical cyclone data from multiple agencies to create a unified, publicly available, best-track dataset that improves inter-agency comparisons. IBTrACS was developed collaboratively with all the World Meteorological Organization (WMO) Regional Specialized Meteorological Centres, as well as other organizations and individuals from around the world.

In this problem set, we will use all storms available in the IBTrACS record since 1842. Download the file `ibtracs.ALL.list.v04r00.csv`, move the `.csv` file to your `working directory`. Read Column Variable Descriptions for variables in the file. Examine the first few lines of the file.

Below we provide an example to load the file as a `pandas` dataframe. Think about the options being used and why, and modify when necessary.

In [24]:
```
df = pd.read_csv('data_files\ibtracs.ALL.list.v04r00.csv',
                 usecols=range(17),
                 skiprows=[1],
                 parse_dates=['ISO_TIME'],
                 na_values=['NOT_NAMED', 'NAME'],
                 dtype={'NAME': str})
df.head()
```

Out[24]:

	SID	SEASON	NUMBER	BASIN	SUBBASIN	NAME	ISO_TIME	NATURE	LAT	LON	WMO_V
0	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 03:00:00	NR	10.9000	80.3000	
1	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 06:00:00	NR	10.8709	79.8265	
2	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 09:00:00	NR	10.8431	79.3524	
3	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 12:00:00	NR	10.8188	78.8772	
4	1842298N11080	1842	1	NI	BB	NaN	1842-10-25 15:00:00	NR	10.8000	78.4000	

3.1

Group the data on Storm Identifie (`SID`), report names (`NAME`) of the 10 largest hurricanes according to wind speed (`WMO_WIND`).

In [25]:
```
# The original dataset makes me headache
# It stores all wind speed data in string type with missing values filled by a space
# So I have to write a function to replace all the spaces
# by a certian negative number (say,-999) before set them all to integers
def remove_space(c):
    if c == ' ':
        return -999
    else:
        return int(c)
df['WMO_WIND'] = df['WMO_WIND'].apply(remove_space)
```

In [26]:
```
# Group the data on SID and find the maximum WND SPD for each hurrican
tops = df.groupby(['SID','NAME'])['WMO_WIND'].max().sort_values(ascending=0)
```

In [27]:
```
# Print the top ten to check
tops.head(10)
```

Out[27]:
```
SID            NAME
2015293N13266  PATRICIA    185
1980214N11330  ALLEN       165
2019236N10314  DORIAN      160
1997253N12255  LINDA       160
1988253N12306  GILBERT     160
2005289N18282  WILMA       160
2017242N16333  IRMA        155
1998295N12284  MITCH       155
2005261N21290  RITA        155
2009288N07267  RICK        155
Name: WMO_WIND, dtype: int64
```

3.2

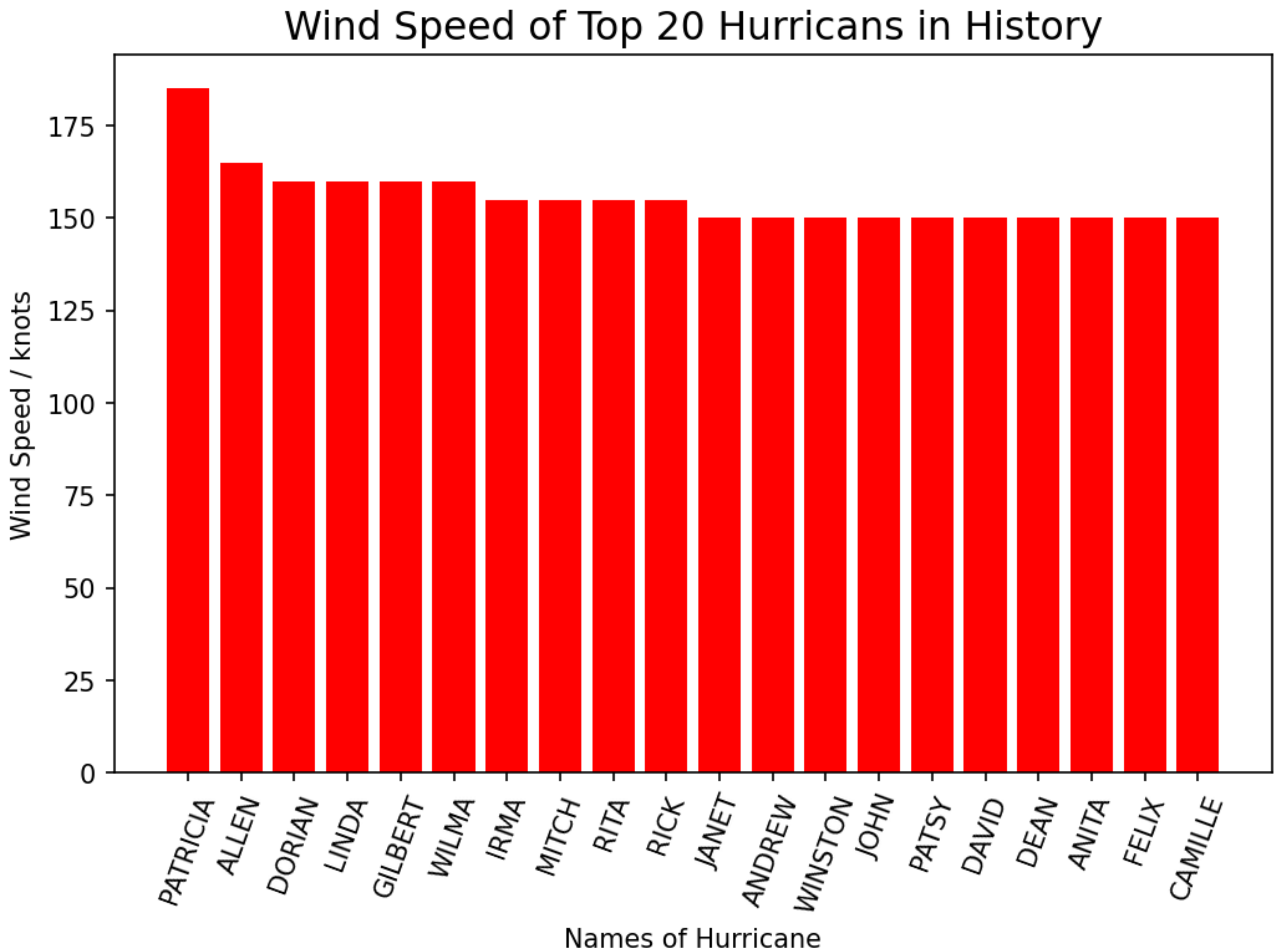
Make a bar chart of the wind speed (WMO_WIND) of the 20 strongest-wind hurricanes.

In [28]:
```
# It gets tricky when you try to deal with MultiIndex Series,
# so I'd better reset the index and get a comforting DataFrame for sake.
tops = tops.reset_index()
tops.head()
```

Out[28]:

	SID	NAME	WMO_WIND
0	2015293N13266	PATRICIA	185
1	1980214N11330	ALLEN	165
2	2019236N10314	DORIAN	160
3	1997253N12255	LINDA	160
4	1988253N12306	GILBERT	160

In [79]:
```
# Plot the bar chart
top_20 = tops.head(20)
fig, ax = plt.subplots(figsize=(8,5),dpi=150)
ax.bar(top_20['NAME'],top_20['WMO_WIND'],color = 'r')
ax.set_title('Wind Speed of Top 20 Hurricans in History',fontsize=15)
ax.set_xlabel('Names of Hurricane')
ax.set_ylabel('Wind Speed / knots')
ax.xaxis.set_tick_params(rotation=70)
```



3.3

Plot the count of all datapoints by Basin as a bar chart.

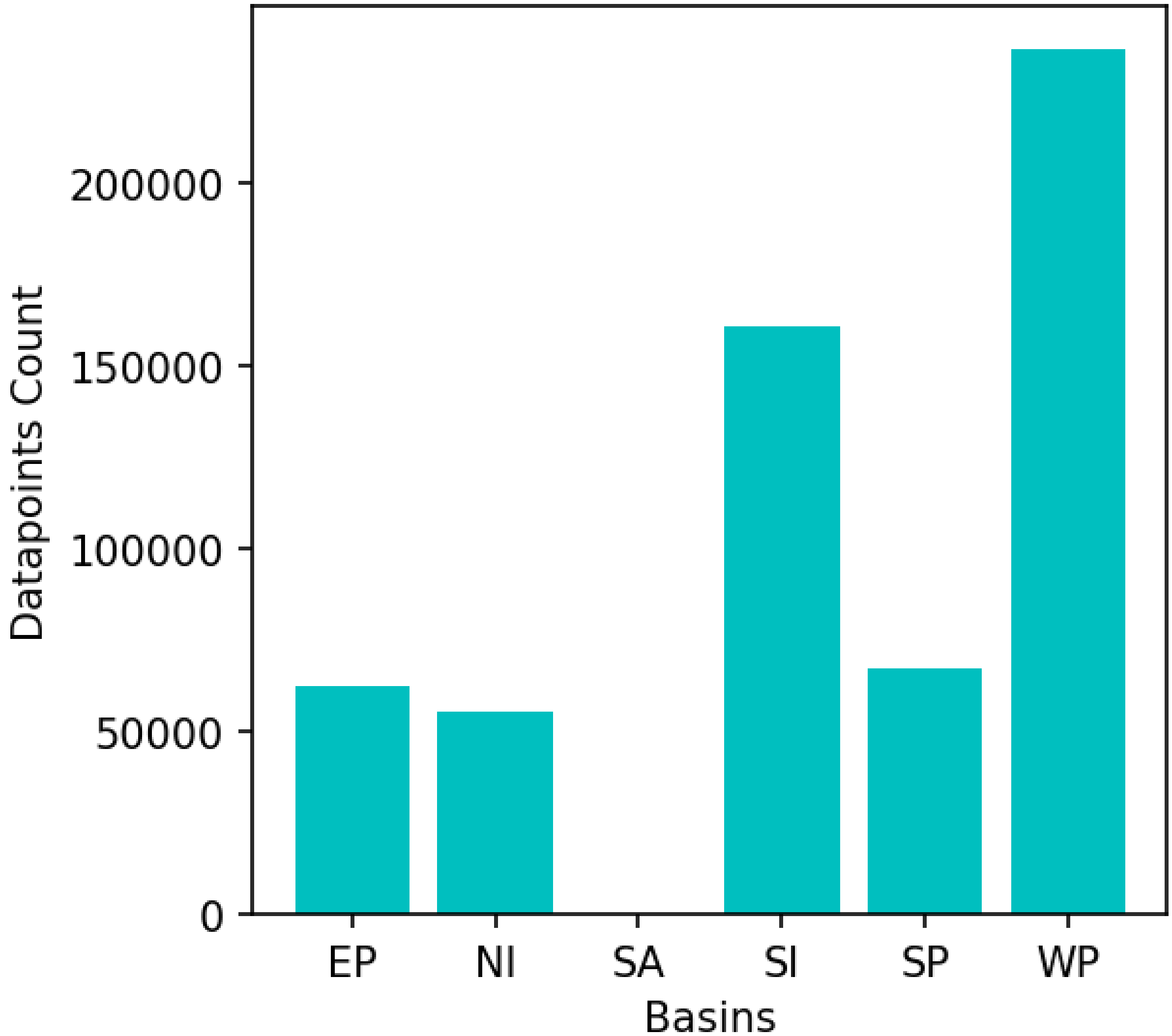
In [30]:
```
bsn = df.groupby(['BASIN']).count()
bsn
```

Out[30]:

BASIN	SID	SEASON	NUMBER	SUBBASIN	NAME	ISO_TIME	NATURE	LAT	LON	WMO_WIND	WMC
EP	62412	62412	62412	62412	55511	62412	62412	62412	62412	62412	
NI	55402	55402	55402	55402	4008	55402	55402	55402	55402	55402	
SA	119	119	119	119	0	119	119	119	119	119	
SI	160668	160668	160668	160668	80051	160668	160668	160668	160668	160668	
SP	67119	67119	67119	67119	39459	67119	67119	67119	67119	67119	
WP	236576	236576	236576	236576	151598	236576	236576	236576	236576	236576	

In [31]:
```
# Plot the bar chart
fig, ax = plt.subplots(figsize=(4,4),dpi=150)
ax.bar(bsn.index,bsn['SID'],color='c')
ax.set_xlabel('Basins')
ax.set_ylabel('Datapoints Count')
```

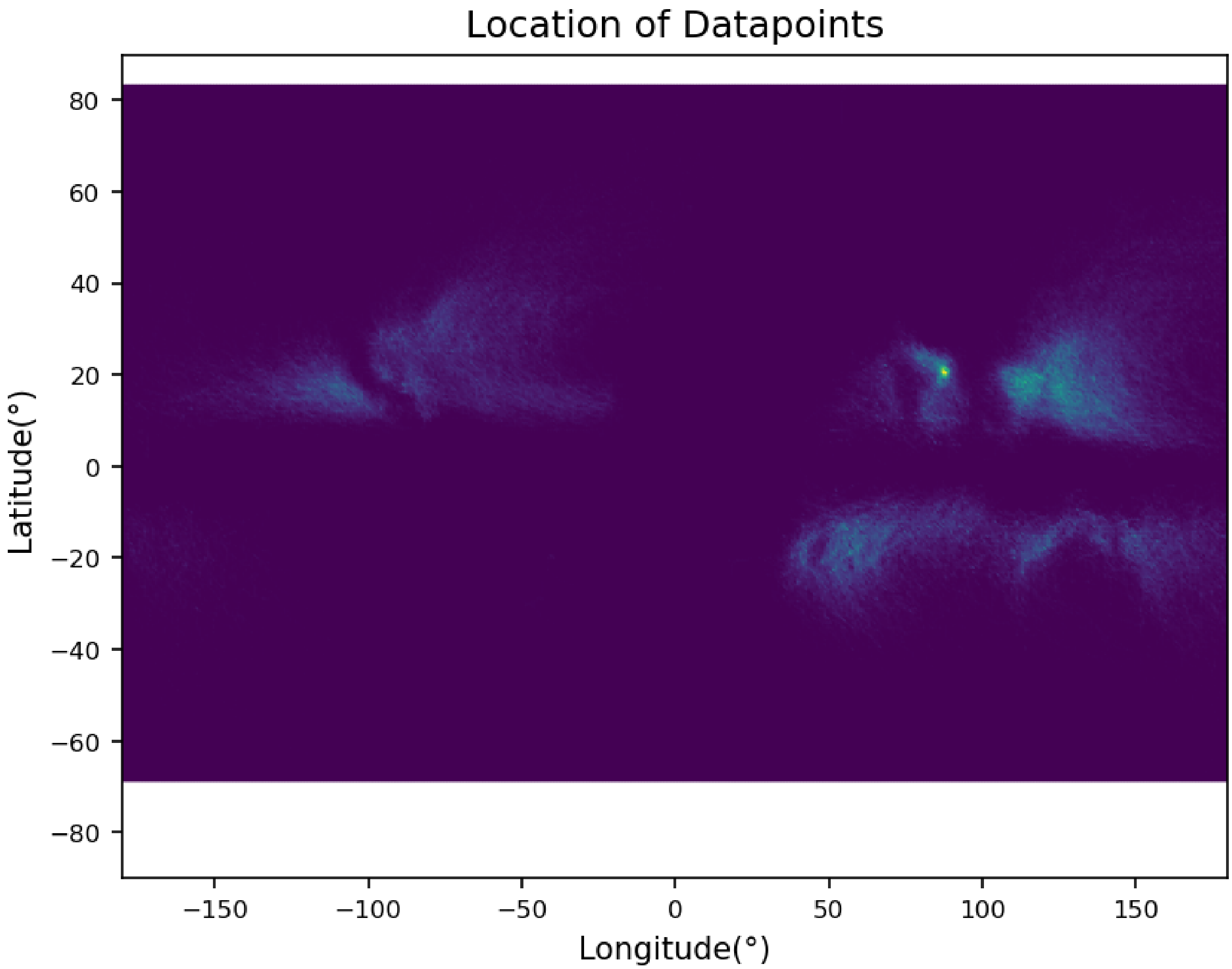
Out[31]: `Text(0, 0.5, 'Datapoints Count')`



3.4

Make a hexbin plot of the location of datapoints in Latitude and Longitude.

In [32]:
```
# Plot the hexbin plot
fig, ax = plt.subplots(dpi=150)
ax.hexbin(df['LON'],df['LAT'],gridsize=500)
ax.set_xlabel('Longitude(°)',fontsize=10)
ax.set_ylabel('Latitude(°)',fontsize=10)
ax.set(title='Location of Datapoints',
       xlim=(-180,180),ylim=(-90,90))
ax.xaxis.set_tick_params(labelsize=8)
ax.yaxis.set_tick_params(labelsize=8)
```

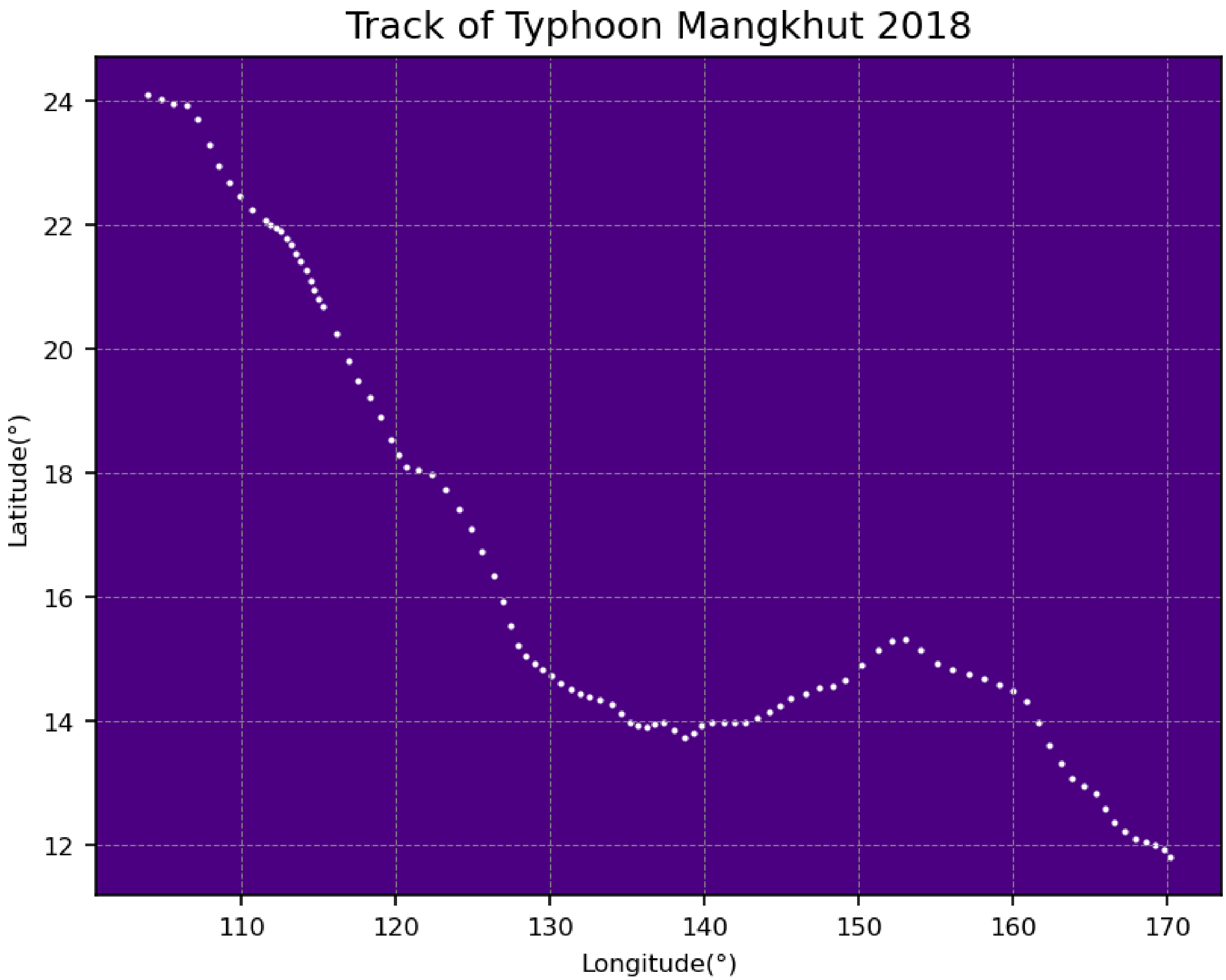


3.5

Find Typhoon Mangkhut (from 2018) and plot its track as a scatter plot.

In [33]:
```
tmk = df.loc[(df['NAME']=='MANGKHUT')&(df['SEASON']==2018)]
```

In [34]:
```
fig, ax = plt.subplots(dpi=150)
ax.scatter(tmk['LON'],tmk['LAT'],s = 1,c='1')
ax.set_xlabel('Longitude(°)',fontsize=8)
ax.set_ylabel('Latitude(°)',fontsize=8)
ax.set(title='Track of Typhoon Mangkhut 2018',
      facecolor='indigo')
ax.xaxis.set_tick_params(labelsize=8)
ax.yaxis.set_tick_params(labelsize=8)
ax.grid(color = 'grey', linestyle = 'dashed', linewidth=0.5)
```



3.6

Create a filtered dataframe that contains only data since 1970 from the Western North Pacific (“WP”) and Eastern North Pacific (“EP”) Basin. Use this for the rest of the problem set.

In [35]:
```
ndf = df.loc[(df['SEASON']>1969)&((df['BASIN']=='WP')|(df['BASIN']=='EP'))].copy()
ndf.head()
```

Out[35]:

	SID	SEASON	NUMBER	BASIN	SUBBASIN	NAME	ISO_TIME	NATURE	LAT	LON
350394	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 00:00:00	TS	7.00000	151.400
350395	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 03:00:00	TS	7.24752	151.205
350396	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 06:00:00	TS	7.50000	151.000
350397	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 09:00:00	TS	7.75747	150.772
350398	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 12:00:00	TS	8.00000	150.500

3.7

Plot the number of datapoints per day.

In [36]:
```
ndf['DATE'] = ndf['ISO_TIME'].dt.date
ndf.head()
```

Out[36]:

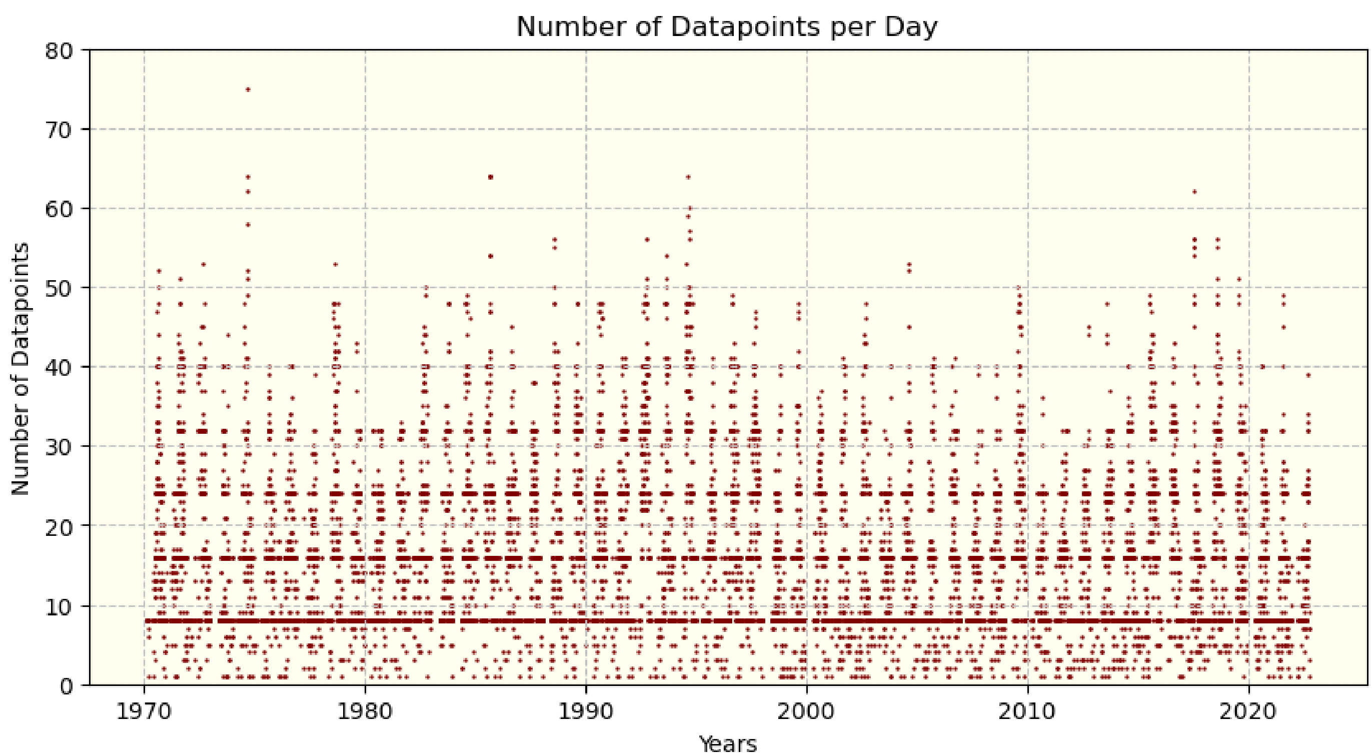
	SID	SEASON	NUMBER	BASIN	SUBBASIN	NAME	ISO_TIME	NATURE	LAT	LON
350394	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 00:00:00	TS	7.00000	151.400
350395	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 03:00:00	TS	7.24752	151.205
350396	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 06:00:00	TS	7.50000	151.000
350397	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 09:00:00	TS	7.75747	150.772
350398	1970050N07151	1970	22	WP	MM	NANCY	1970-02-19 12:00:00	TS	8.00000	150.500

In [37]:
```
# Label this filtered dataframe(Series actually) with dp_per_day
dp_per_day = ndf.groupby('DATE')['ISO_TIME'].count()
dp_per_day
```

Out[37]:
```
DATE
1970-02-19    8
1970-02-20    8
1970-02-21    8
1970-02-22    8
1970-02-23    8
             ..
2022-10-04    9
2022-10-05    7
2022-10-09    1
2022-10-10    7
2022-10-12    3
Name: ISO_TIME, Length: 10817, dtype: int64
```

In [38]:
```
fig, ax = plt.subplots(figsize=(10,5),dpi=100)
ax.set(title = 'Number of Datapoints per Day',
       facecolor = '#FFFFF0',
       xlabel = 'Years',
       ylabel = 'Number of Datapoints',
       ylim = (0,80))
ax.grid(ls = 'dashed', color = 'silver')
ax.scatter(dp_per_day.index,dp_per_day,s=0.5, color = 'maroon')
```

Out[38]:
```
<matplotlib.collections.PathCollection at 0x296d832c160>
```



3.8

Calculate the climatology of datapoint counts as a function of `day of year`. The day of year is the sequential day number starting with day 1 on January 1st.

In [39]:
```
# We have to fill the blank values.
# So firstly, get a DataFrame with all days from 1970-01-01 to 2022-10-12
rng = pd.date_range(start = '1970-01-01', end = '2022-10-12')
all_days = pd.DataFrame({'DATE':rng,'COUNTS':0})
all_days
```

Out[39]:

	DATE	COUNTS
0	1970-01-01	0
1	1970-01-02	0
2	1970-01-03	0
3	1970-01-04	0
4	1970-01-05	0
...	...	...
19273	2022-10-08	0
19274	2022-10-09	0
19275	2022-10-10	0
19276	2022-10-11	0
19277	2022-10-12	0

19278 rows × 2 columns

In [40]:
```
# Modify dp_per_day for merge operation later
dp_per_day = dp_per_day.reset_index()
dp_per_day['DATE'] = pd.to_datetime(dp_per_day['DATE'])
dp_per_day
```

Out[40]:

	DATE	ISO_TIME
0	1970-02-19	8
1	1970-02-20	8
2	1970-02-21	8
3	1970-02-22	8
4	1970-02-23	8
...	...	...
10812	2022-10-04	9
10813	2022-10-05	7
10814	2022-10-09	1
10815	2022-10-10	7
10816	2022-10-12	3

10817 rows × 2 columns

In [41]:
```
# Merge the two DataFrame
# The function will help us add blank days automatically
dp_per_day = dp_per_day.merge(all_days, on = 'DATE', how = 'outer').sort_values('DATE')
dp_per_day
```

Out[41]:

	DATE	ISO_TIME	COUNTS
10817	1970-01-01	NaN	0
10818	1970-01-02	NaN	0
10819	1970-01-03	NaN	0
10820	1970-01-04	NaN	0
10821	1970-01-05	NaN	0
...	...	...	...
19276	2022-10-08	NaN	0
10814	2022-10-09	1.0	0
10815	2022-10-10	7.0	0
19277	2022-10-11	NaN	0
10816	2022-10-12	3.0	0

19278 rows × 3 columns

In [42]:

```
# Fill the NA values with 0
dp_per_day.fillna(0, inplace = True)
dp_per_day
```

Out[42]:

	DATE	ISO_TIME	COUNTS
10817	1970-01-01	0.0	0
10818	1970-01-02	0.0	0
10819	1970-01-03	0.0	0
10820	1970-01-04	0.0	0
10821	1970-01-05	0.0	0
...	...	...	...
19276	2022-10-08	0.0	0
10814	2022-10-09	1.0	0
10815	2022-10-10	7.0	0
19277	2022-10-11	0.0	0
10816	2022-10-12	3.0	0

19278 rows × 3 columns

In [43]:

```
# Simplify the DataFrame by drop useless columns
dp_per_day = dp_per_day.set_index(dp_per_day['DATE'].dt.dayofyear).drop('COUNTS',axis = 1)
dp_per_day
```

Out[43]:

DATE	DATE	ISO_TIME
1	1970-01-01	0.0
2	1970-01-02	0.0
3	1970-01-03	0.0
4	1970-01-04	0.0
5	1970-01-05	0.0
...	...	...
281	2022-10-08	0.0
282	2022-10-09	1.0
283	2022-10-10	7.0
284	2022-10-11	0.0
285	2022-10-12	3.0

19278 rows × 2 columns

In [44]:

```
# Obatin the CLIMATOLOGY using groupby!
clmt = dp_per_day.groupby(dp_per_day.index).mean()
clmt
```

Out[44]:

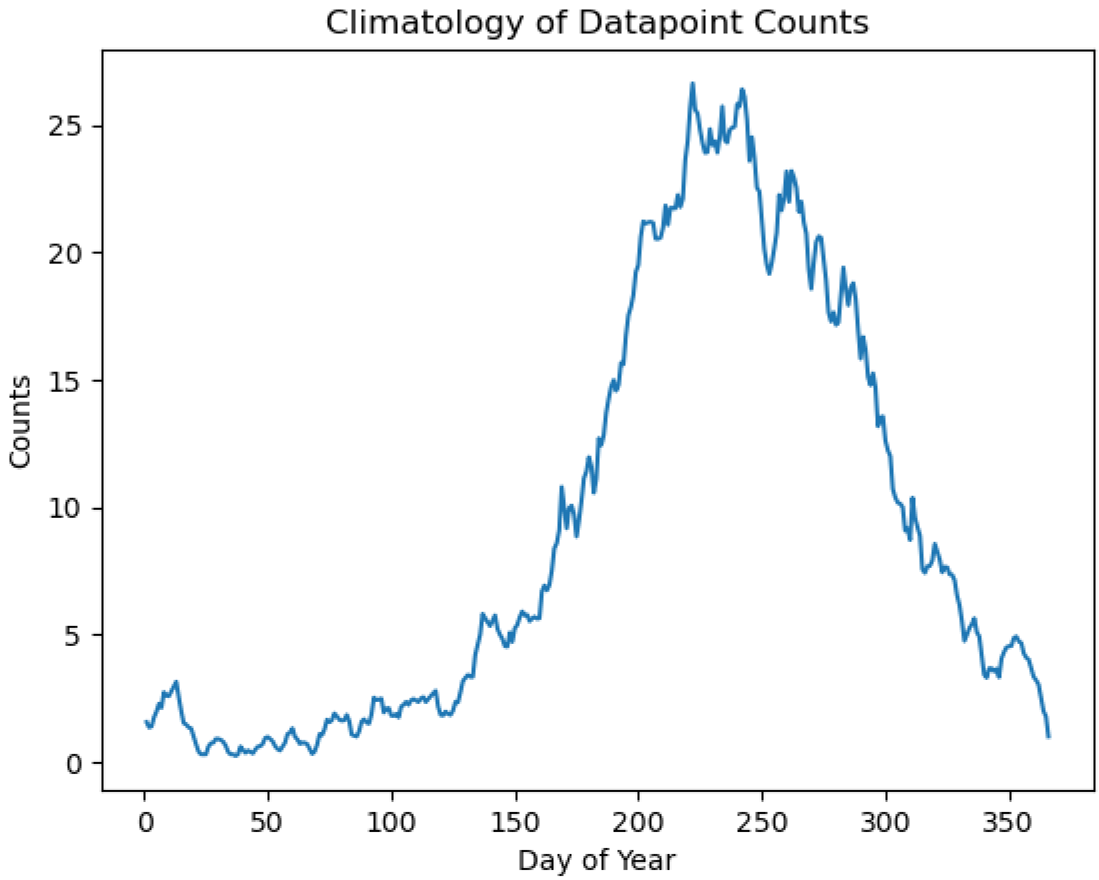
DATE	ISO_TIME
1	1.566038
2	1.358491
3	1.396226
4	1.754717
5	1.981132
...	...
362	3.038462
363	2.538462
364	2.000000
365	1.788462
366	1.000000

366 rows × 1 columns

In [45]:

```
# Plot the climatology
fix, ax = plt.subplots(dpi=100)
ax.plot(clmt)
ax.set(title = 'Climatology of Datapoint Counts',
       xlabel = 'Day of Year',
       ylabel = 'Counts')
```

Out[45]:
```
[Text(0.5, 1.0, 'Climatology of Datapoint Counts'),
 Text(0.5, 0, 'Day of Year'),
 Text(0, 0.5, 'Counts')]
```



3.9

Calculate the anomaly of daily counts from the climatology.

In [46]:
```
# We need to calculate the deviation between values and the baseline
# The first step is to merge dp_per_day with climatology
dp_per_day = dp_per_day.merge(clmt, left_index=True, right_index=True)
dp_per_day['D'] = dp_per_day['ISO_TIME_x'] - dp_per_day['ISO_TIME_y']
dp_per_day
```

Out[46]:

DATE	DATE	ISO_TIME_x	ISO_TIME_y	D
1	1970-01-01	0.0	1.566038	-1.566038
1	1971-01-01	0.0	1.566038	-1.566038
1	1972-01-01	0.0	1.566038	-1.566038
1	1973-01-01	0.0	1.566038	-1.566038
1	1974-01-01	0.0	1.566038	-1.566038
...	...	...	...	...
366	2004-12-31	0.0	1.000000	-1.000000
366	2008-12-31	0.0	1.000000	-1.000000
366	2012-12-31	0.0	1.000000	-1.000000
366	2016-12-31	0.0	1.000000	-1.000000
366	2020-12-31	0.0	1.000000	-1.000000

19278 rows × 4 columns

In [47]:

```
# Decorate our DataFrame a bit
# and this is the final results
dp_per_day.index.name = 'DOY'
dp_per_day.columns = ['DATE','COUNTS','BASE','D']
dp_per_day = dp_per_day.sort_values('DATE')
dp_per_day
```

Out[47]:

DOY	DATE	COUNTS	BASE	D
1	1970-01-01	0.0	1.566038	-1.566038
2	1970-01-02	0.0	1.358491	-1.358491
3	1970-01-03	0.0	1.396226	-1.396226
4	1970-01-04	0.0	1.754717	-1.754717
5	1970-01-05	0.0	1.981132	-1.981132
...	...	...	...	...
281	2022-10-08	0.0	17.245283	-17.245283
282	2022-10-09	1.0	18.301887	-17.301887
283	2022-10-10	7.0	19.396226	-12.396226
284	2022-10-11	0.0	18.528302	-18.528302
285	2022-10-12	3.0	17.924528	-14.924528

19278 rows × 4 columns

3.10

Resample the anomaly timeseries at annual resolution and plot. So which years stand out as having anomalous hurricane activity?

In [48]:
```
# Obtain a Timeseries
dp_per_day = dp_per_day.set_index('DATE')
```

In [49]:
```
# Resample the anomaly at annual resolution
annual = dp_per_day.resample('Y').sum()
annual.head()
```

Out[49]:

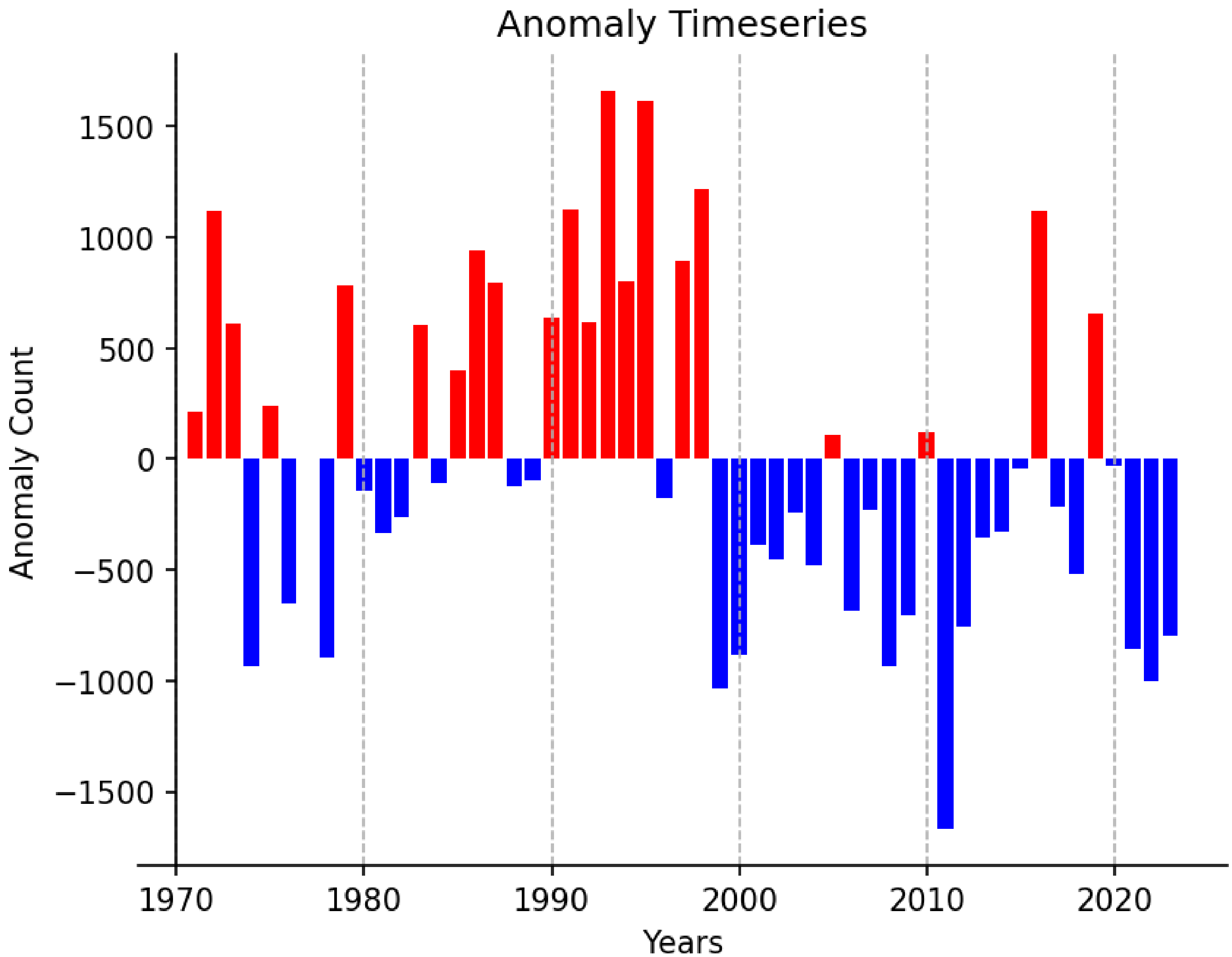
DATE	COUNTS	BASE	D
1970-12-31	3555.0	3339.192671	215.807329
1971-12-31	4459.0	3339.192671	1119.807329
1972-12-31	3952.0	3340.192671	611.807329
1973-12-31	2407.0	3339.192671	-932.192671
1974-12-31	3581.0	3339.192671	241.807329

In [50]:
```
# Dectect negative values
# (This is only for a better-looking plot)
annual['sign'] = annual['D'] > 0
annual.head()
```

Out[50]:

DATE	COUNTS	BASE	D	sign
1970-12-31	3555.0	3339.192671	215.807329	True
1971-12-31	4459.0	3339.192671	1119.807329	True
1972-12-31	3952.0	3340.192671	611.807329	True
1973-12-31	2407.0	3339.192671	-932.192671	False
1974-12-31	3581.0	3339.192671	241.807329	True

In [51]:
```
# Plot the Timeseries
fig, ax = plt.subplots(dpi = 150)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))
ax.set(title = 'Anomaly Timeseries',
       xlabel = 'Years',
       ylabel = 'Anomaly Count')
ax.bar(annual.index, annual['D'], width=300,
       color = annual.sign.map({True: 'r', False: 'b'}))
ax.grid(ls = 'dashed', axis = 'x')
```



4. Explore a data set

Browse the National Centers for Environmental Information (NCEI) or Advanced Global Atmospheric Gases Experiment (AGAGE) website. Search and download a data set you are interested in. You are also welcome to use data from your group in this problem set. But the data set should be in `csv`, `XLS`, or `XLSX` format, and have temporal information.

4.1

Load the `csv`, `XLS`, or `XLSX` file, and clean possible data points with missing values or bad quality.

For this exercise, I choose weather data of my hometown Jinzhou, Liaoning for investigation. The dataset cotains weather data from 1956 to 2022, with a sampling spacing of 3 hours or so. I will look at the wind and temperature values.

In [52]:

```
# Read the csv file and use only DEW, TMP, VIS, AND WND
jz_df = pd.read_csv('data_files\Jinzhou_Weather_1956_2022.csv',
                    usecols=['DATE','TMP','WND'])
jz_df
```

Out[52]:

	DATE	TMP	WND
0	1956-08-20T06:00:00	+0261,1	180,1,N,0098,1
1	1956-08-20T12:00:00	+0222,1	180,1,N,0051,1
2	1956-08-20T18:00:00	+0200,1	999,1,C,0000,1
3	1956-08-21T00:00:00	+0211,1	040,1,N,0021,1
4	1956-08-21T06:00:00	+0250,1	160,1,N,0082,1
...	...	...	...
166855	2022-10-25T09:00:00	+0157,1	200,1,N,0040,1
166856	2022-10-25T12:00:00	+0153,1	190,1,N,0040,1
166857	2022-10-25T15:00:00	+0150,1	210,1,N,0030,1
166858	2022-10-25T18:00:00	+0157,1	220,1,N,0030,1
166859	2022-10-25T21:00:00	+0148,1	230,1,N,0030,1

166860 rows × 3 columns

In [53]:

```
# Convert DATE column to datetime and set it as index
jz_df['DATE'] = pd.to_datetime(jz_df['DATE'])
jz_df = jz_df.set_index('DATE').copy()
```

In [54]:

```
# Obtain TMP info
jz_df = jz_df.join(jz_df['TMP'].str.split(',', 1, expand = True))\
    .rename(columns={0:'TMP_VALUE',1:'TMP_CODE'})
jz_df['TMP_VALUE'] = jz_df['TMP_VALUE'].astype(int)/10
# Obtain WND info
jz_df = jz_df.join(jz_df['WND'].str.split(',', 4, expand = True))\
    .rename(columns={0:'WND_DIR',1:'DIR_CODE',2:'WND_TYP',3:'WND_SPD',4:'SPD_CODE'})
jz_df['WND_SPD'] = jz_df['WND_SPD'].astype(int)/10
# Show DataFrame
jz_df
```

Out[54]:

DATE	TMP	WND	TMP_VALUE	TMP_CODE	WND_DIR	DIR_CODE	WND_TYP	WND_SPD	SPD_COI
1956-08-20 06:00:00	+0261,1	180,1,N,0098,1	26.1	1	180	1	N	9.8	
1956-08-20 12:00:00	+0222,1	180,1,N,0051,1	22.2	1	180	1	N	5.1	
1956-08-20 18:00:00	+0200,1	999,1,C,0000,1	20.0	1	999	1	C	0.0	
1956-08-21 00:00:00	+0211,1	040,1,N,0021,1	21.1	1	040	1	N	2.1	
1956-08-21 06:00:00	+0250,1	160,1,N,0082,1	25.0	1	160	1	N	8.2	
...	...	...	...	...	...	...	...	...	
2022-10-25 09:00:00	+0157,1	200,1,N,0040,1	15.7	1	200	1	N	4.0	
2022-10-25 12:00:00	+0153,1	190,1,N,0040,1	15.3	1	190	1	N	4.0	
2022-10-25 15:00:00	+0150,1	210,1,N,0030,1	15.0	1	210	1	N	3.0	
2022-10-25 18:00:00	+0157,1	220,1,N,0030,1	15.7	1	220	1	N	3.0	
2022-10-25 21:00:00	+0148,1	230,1,N,0030,1	14.8	1	230	1	N	3.0	

166860 rows × 9 columns

In [55]:

```
# Check data quality
codes = ['TMP_CODE','DIR_CODE','SPD_CODE']
for code in codes:
    print(code,jz_df.loc[jz_df[code]!='1'][code].unique())
```

```
TMP_CODE ['9' '2']
DIR_CODE ['9']
SPD_CODE ['9' '2' '5']
```

In [56]:

```
# Treating missing data
# Data around 1970' are missing so we only select data after 1975
jz_df = jz_df.loc['1975':]
jz_df = jz_df.loc[(jz_df['TMP_CODE']!='9')&(jz_df['DIR_CODE']!='9')\
                  &(jz_df['WND_SPD']!='9')&(jz_df['SPD_CODE']!='9')]
jz_df
```

Out[56]:

DATE	TMP	WND	TMP_VALUE	TMP_CODE	WND_DIR	DIR_CODE	WND_TYP	WND_SPD	SPD_COI
1975-01-01 00:00:00	-0120,1	040,1,N,0030,1	-12.0	1	040	1	N	3.0	
1975-01-01 03:00:00	-0060,1	020,1,N,0060,1	-6.0	1	020	1	N	6.0	
1975-01-01 06:00:00	-0040,1	360,1,N,0050,1	-4.0	1	360	1	N	5.0	
1975-01-01 09:00:00	-0060,1	360,1,N,0040,1	-6.0	1	360	1	N	4.0	
1975-01-01 12:00:00	-0070,1	999,1,C,0000,1	-7.0	1	999	1	C	0.0	
...	...	...	...	...	...	...	...	...	
2022-10-25 09:00:00	+0157,1	200,1,N,0040,1	15.7	1	200	1	N	4.0	
2022-10-25 12:00:00	+0153,1	190,1,N,0040,1	15.3	1	190	1	N	4.0	
2022-10-25 15:00:00	+0150,1	210,1,N,0030,1	15.0	1	210	1	N	3.0	
2022-10-25 18:00:00	+0157,1	220,1,N,0030,1	15.7	1	220	1	N	3.0	
2022-10-25 21:00:00	+0148,1	230,1,N,0030,1	14.8	1	230	1	N	3.0	

133166 rows × 9 columns

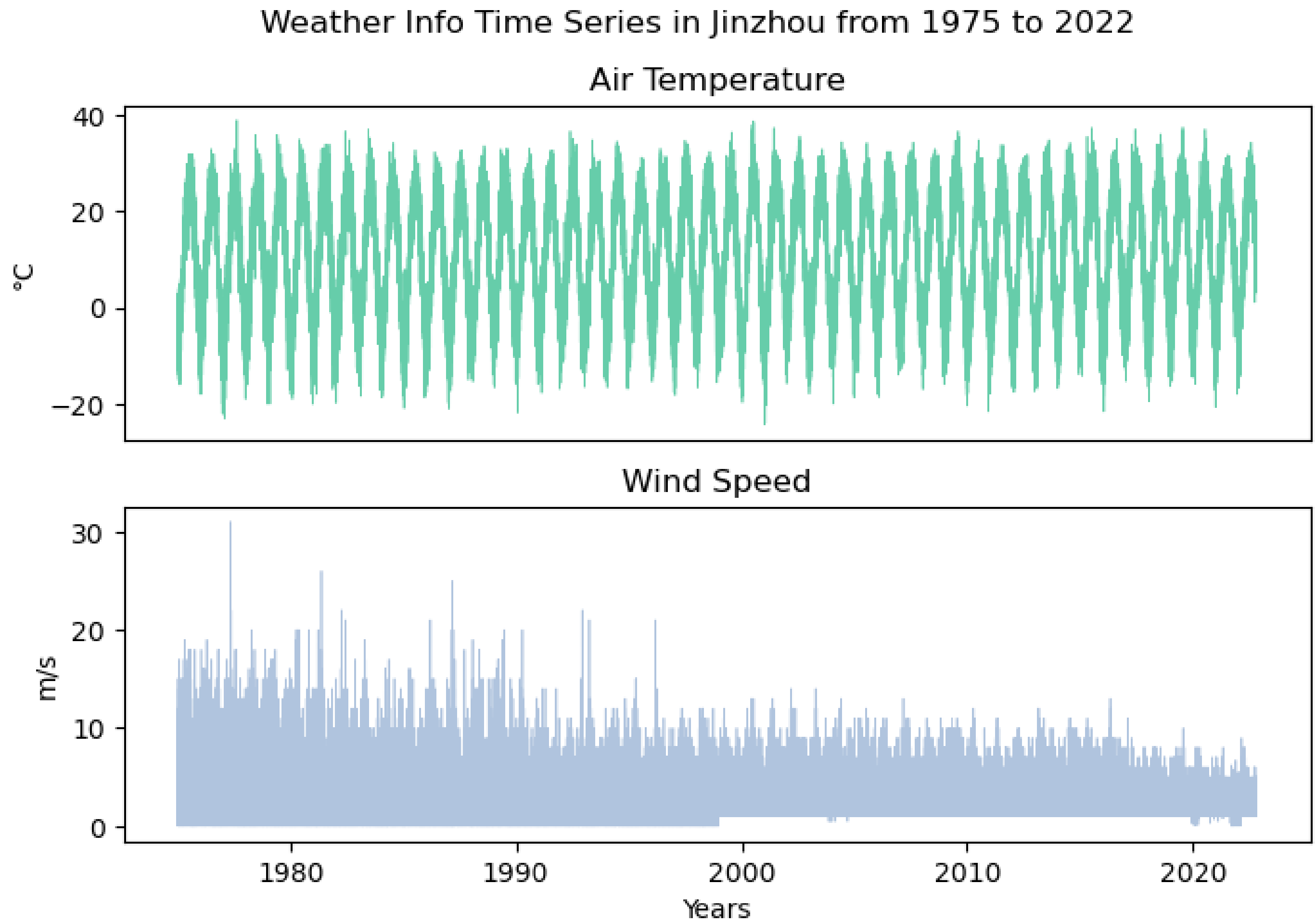
4.2

Plot the time series of a certain variable.

In [57]:

```
fig, axes = plt.subplots(2, 1, figsize = (8,5),dpi = 100)
plt.suptitle('Weather Info Time Series in Jinzhou from 1975 to 2022')
axes[0].plot(jz_df.index,jz_df['TMP_VALUE'],linewidth='0.5',color='mediumaquamarine')
axes[1].plot(jz_df.index,jz_df['WND_SPD'],linewidth='0.5',color='lightsteelblue')
axes[0].set(title='Air Temperature',
            xticks=[],
            ylabel='°C')
axes[1].set(title='Wind Speed',
            xlabel='Years',
            ylabel='m/s')
```

Out[57]: [Text(0.5, 1.0, 'Wind Speed'), Text(0.5, 0, 'Years'), Text(0, 0.5, 'm/s')]



4.3

Conduct at least 5 simple statistical checks with the variable, and report your findings.

In [58]:
```
# First I wanna know how temperature change among the years,
# So I plot yearly averaged temperature in on plot
jz_df['YEAR'] = jz_df.index
jz_df['YEAR'] = jz_df['YEAR'].dt.year
mean_df = jz_df.groupby('YEAR').mean()
mean_df.head()
```

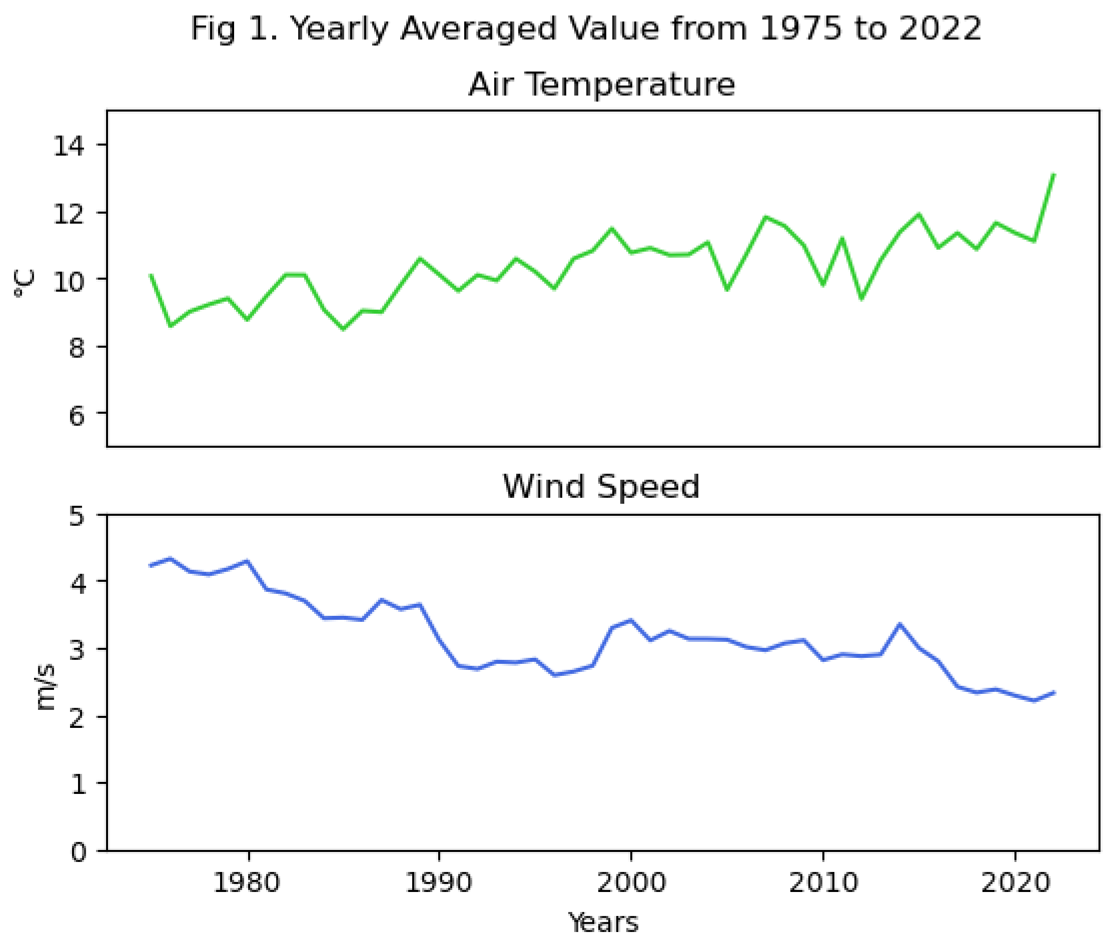
Out[58]:

	TMP_VALUE	WND_SPD
YEAR		
1975	10.054701	4.225241
1976	8.564066	4.323193
1977	8.995153	4.133474
1978	9.205659	4.088682
1979	9.388985	4.171084

In [59]:
```
fix, axes = plt.subplots(2,1,dpi=100)
plt.suptitle('Fig 1. Yearly Averaged Value from 1975 to 2022')
axes[0].plot(mean_df.index,mean_df['TMP_VALUE'],color = 'limegreen')
axes[1].plot(mean_df.index,mean_df['WND_SPD'],color= 'royalblue')
axes[0].set(title='Air Temperature',
            xticks=[],
```

```
            ylabel='°C',
            ylim=(5,15))
axes[1].set(title='Wind Speed',
            xlabel='Years',
            ylabel='m/s',
            ylim=(0,5.0))
```

Out[59]:
```
[Text(0.5, 1.0, 'Wind Speed'),
 Text(0.5, 0, 'Years'),
 Text(0, 0.5, 'm/s'),
 (0.0, 5.0)]
```



The air temperature in Jinzhou has seen an overall slight increase in spite of fluctuations, from around 9 degrees celcius to near 11 degrees celcius in 2021. Due to the lack of data of November and December in 2022, the averaged temperature value of 2022 is obviously higher than others.

Averaged wind speed in Jinzhou fluctuates between 2.5m/s and 4.5m/s. There is a downward trend for yearly averaged wind speed among this two decades.

In [60]:
```
# Secondly, I'd like to see how values vary within a year
jz_df['MONTH'] = jz_df.index
jz_df['MONTH'] = jz_df['MONTH'].dt.month
mean_mon_df = jz_df.groupby('MONTH').mean().drop('YEAR',axis = 1)
mean_mon_df
```

Out[60]:

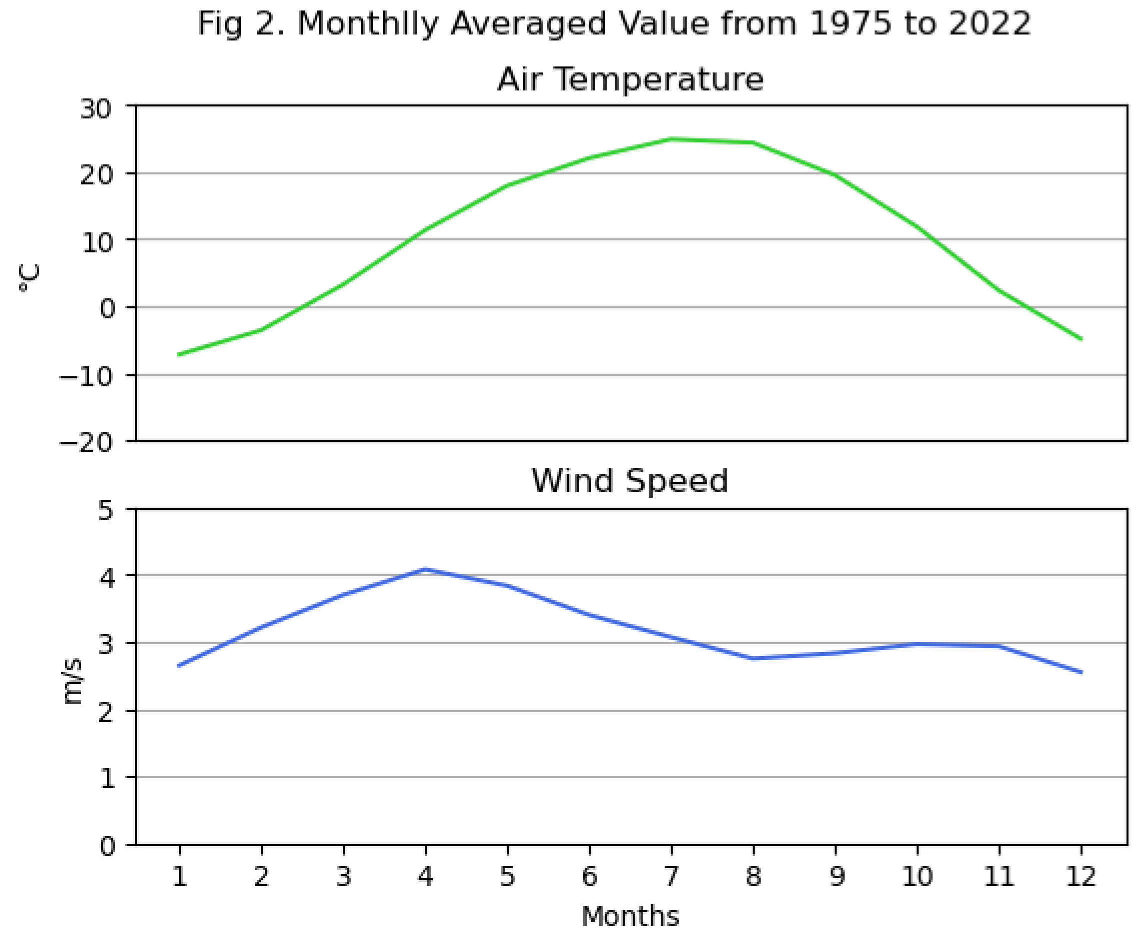
MONTH	TMP_VALUE	WND_SPD
1	-7.209591	2.649674
2	-3.590767	3.215609
3	3.212833	3.698956
4	11.322702	4.079224
5	17.923383	3.835659
6	22.038693	3.400450
7	24.859606	3.070100
8	24.341674	2.750390
9	19.514080	2.830390
10	11.835359	2.966948
11	2.331415	2.934162
12	-4.868151	2.550350

In [61]:

```
fix, axes = plt.subplots(2,1,dpi=100)
plt.suptitle('Fig 2. Monthlly Averaged Value from 1975 to 2022')
axes[0].plot(mean_mon_df.index,mean_mon_df['TMP_VALUE'],color = 'limegreen')
axes[1].plot(mean_mon_df.index,mean_mon_df['WND_SPD'],color='royalblue')
axes[0].grid(axis='y')
axes[1].grid(axis='y')
axes[0].set(title='Air Temperature',
            xticks=[],
            ylabel='°C',
            ylim=(-20,30))
axes[1].set(title='Wind Speed',
            xlabel='Months',
            xticks=np.arange(1,13,1),
            ylabel='m/s',
            ylim=(0,5.0))
```

Out[61]:

```
[Text(0.5, 1.0, 'Wind Speed'),
 Text(0.5, 0, 'Months'),
 [<matplotlib.axis.XTick at 0x296cc7ca160>,
  <matplotlib.axis.XTick at 0x296cc7ca400>,
  <matplotlib.axis.XTick at 0x296cc7cac10>,
  <matplotlib.axis.XTick at 0x296de7f3670>,
  <matplotlib.axis.XTick at 0x296de7f3dc0>,
  <matplotlib.axis.XTick at 0x296d1626550>,
  <matplotlib.axis.XTick at 0x296de7f3a60>,
  <matplotlib.axis.XTick at 0x296d1626b80>,
  <matplotlib.axis.XTick at 0x296d161b310>,
  <matplotlib.axis.XTick at 0x296d161ba30>,
  <matplotlib.axis.XTick at 0x296d1605190>,
  <matplotlib.axis.XTick at 0x296d16058e0>],
 Text(0, 0.5, 'm/s'),
 (0.0, 5.0)]
```

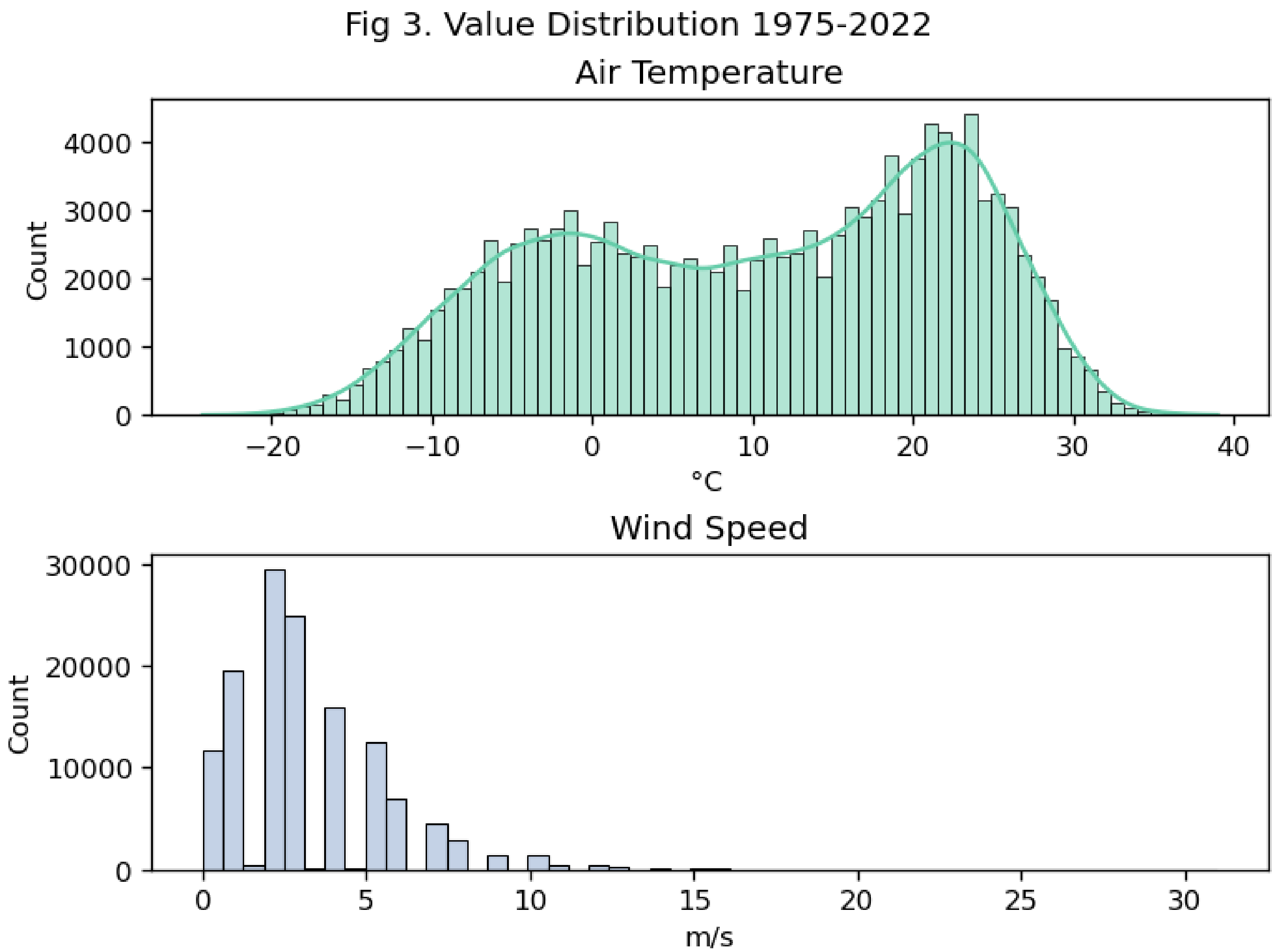


Jinzhou has distinct seasons. The average month temperature reaches its peak during July and August each year and Januaries are normally coldest.

Generally, April is the most windy month, with an averaged wind speed more than 4 meters per second. Winters see a relatively low speed of wind, which somehow contradicts with my common sense.

In [62]:
```
# I'd also like to see the value distribution of air temperature and wind speed.
import seaborn as sns
fig, axes = plt.subplots(2,1,layout='constrained',dpi=120)
plt.suptitle('Fig 3. Value Distribution 1975-2022')
sns.histplot(data=jz_df, x='TMP_VALUE',kde=True, ax = axes[0], color = 'mediumaquamarine')
sns.histplot(jz_df['WND_SPD'], bins=50 , ax = axes[1], color = 'lightsteelblue')
axes[0].set(title = 'Air Temperature',
            xlabel = '°C ')
axes[1].set(title = 'Wind Speed',
            xlabel = 'm/s')
```

Out[62]:
```
[Text(0.5, 1.0, 'Wind Speed'), Text(0.5, 0, 'm/s')]
```



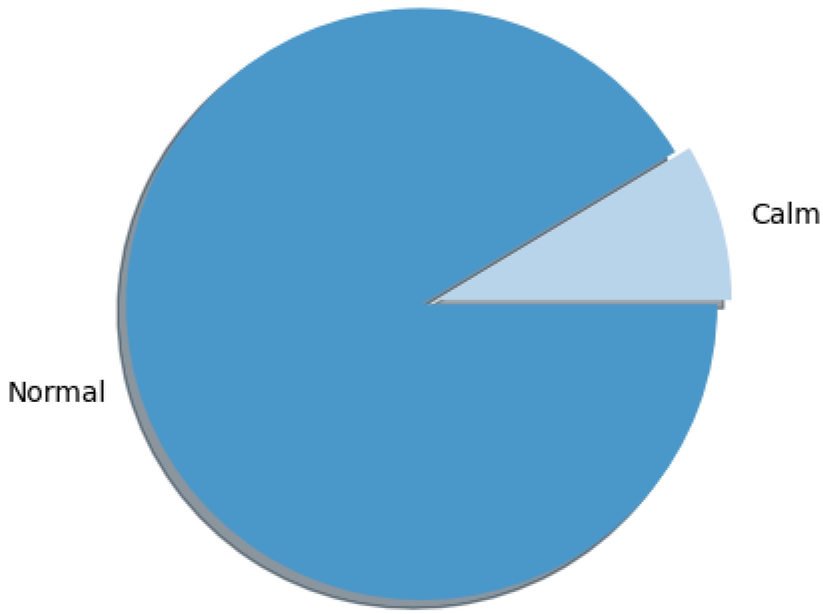
In [63]:
```
# By the way, have a look at the distribution of wind types
typ = jz_df.groupby('WND_TYP')['WND'].count()
typ
```

Out[63]:
```
WND_TYP
C     11446
N    121720
Name: WND, dtype: int64
```

In [64]:
```
fig, ax = plt.subplots(dpi=100)
colors = plt.get_cmap('Blues')(np.linspace(0.3, 0.6, len(typ)))
ax.set_title('Fig 4. Type of Wind')
ax.pie(typ, labels=['Calm','Normal'], colors = colors,
       explode = (0.05,0),
       shadow=True)
```

Out[64]:
```
([<matplotlib.patches.Wedge at 0x296e4dafbe0>,
  <matplotlib.patches.Wedge at 0x296e4da53a0>],
 [Text(1.1083276634931176, 0.30677319037993955, 'Calm'),
  Text(-1.0601395110791552, -0.29343520076645574, 'Normal')])
```

Fig 4. Type of Wind



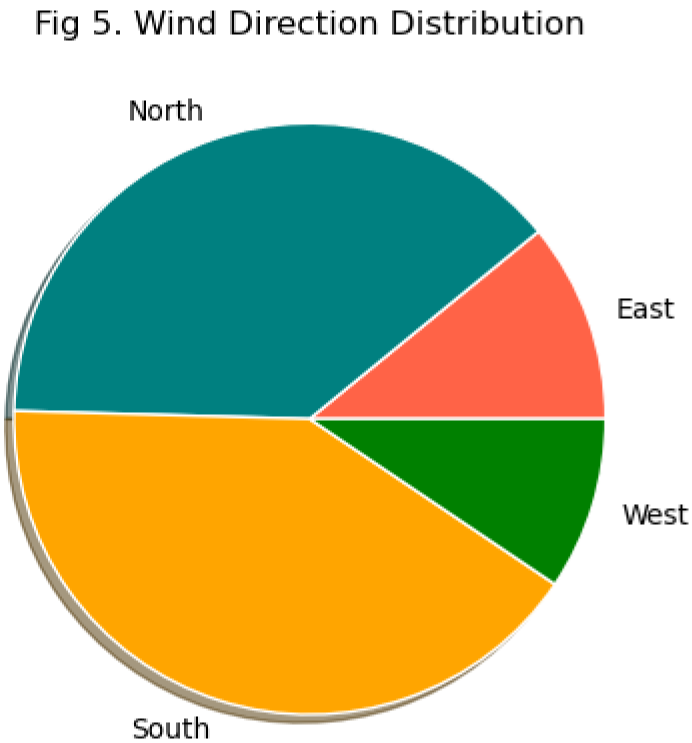
In [65]:
```
# How is the distribution of wind direction?
jz_df['WND_DIR'] = jz_df['WND_DIR'].astype(int)
```

In [66]:
```
jz_df['DIR'] = 'N'
jz_df.loc[ (jz_df['WND_DIR'] <=135 ) & (jz_df['WND_DIR'] > 45 ), ['DIR'] ] = 'E'
jz_df.loc[ (jz_df['WND_DIR'] <=225 ) & (jz_df['WND_DIR'] > 135 ), ['DIR'] ] = 'S'
jz_df.loc[ (jz_df['WND_DIR'] <=315 ) & (jz_df['WND_DIR'] > 225 ), ['DIR'] ] = 'W'
jz_df.loc[jz_df['WND_DIR'] == 999, ['DIR']] = 'MISSING'
```

In [67]:
```
drt = jz_df.groupby('DIR')['WND'].count().drop('MISSING')
```

In [68]:
```
fig, ax = plt.subplots(dpi=100)
ax.set_title('Fig 5. Wind Direction Distribution')
ax.pie(drt, labels=['East','North','South','West'], shadow=True,
       colors = ['tomato','teal','orange','green'],
       wedgeprops={"linewidth": 1, "edgecolor": "white"})
```

Out[68]:
```
([<matplotlib.patches.Wedge at 0x296e4cc2310>,
  <matplotlib.patches.Wedge at 0x296e4cc2a60>,
  <matplotlib.patches.Wedge at 0x296e4ccd1f0>,
  <matplotlib.patches.Wedge at 0x296e4ccd940>],
 [Text(1.0358335007939643, 0.3702012407230969, 'East'),
  Text(-0.35621251482276595, 1.040726978743052, 'North'),
  Text(-0.33591994301433764, -1.0474530022321977, 'South'),
  Text(1.0518835829881668, -0.3217777615684107, 'West')])
```



In [69]:
```
rct_yrs = jz_df['1980':].groupby(['YEAR','MONTH']).mean()
rct_yrs
```

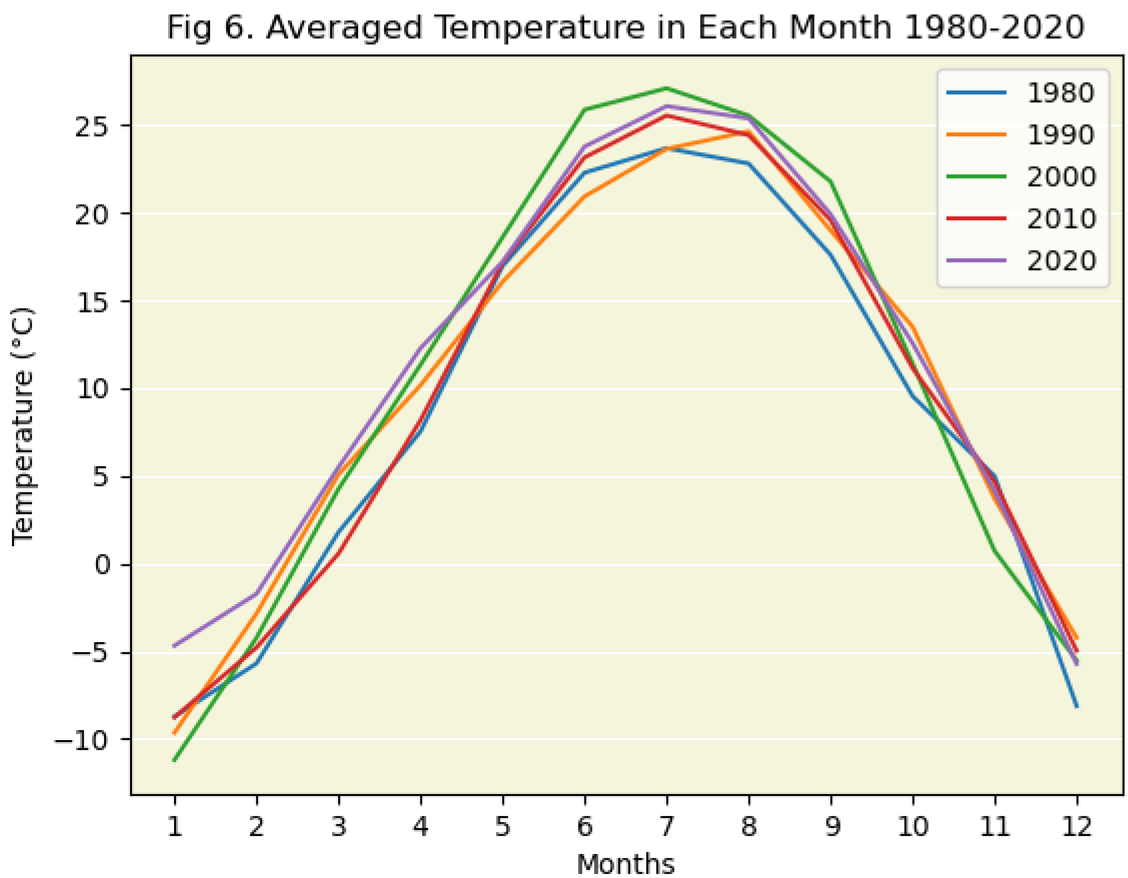
Out[69]:

YEAR	MONTH	TMP_VALUE	WND_DIR	WND_SPD
1980	1	-8.723577	343.296748	3.743902
	2	-5.693617	379.221277	3.170213
	3	1.780488	254.012195	4.825203
	4	7.514644	253.200837	6.983264
	5	16.935743	233.855422	6.337349
...	...	...	...	...
2022	6	21.620175	178.070175	2.293860
	7	25.657265	180.042735	1.965812
	8	24.894690	189.026549	2.318584
	9	20.791441	181.531532	2.337838
	10	12.249143	192.800000	2.434286

514 rows × 3 columns

In [70]:
```
fig, ax = plt.subplots(dpi=100)
years = np.arange(1980,2030,10)
for year in years:
    ax.plot(rct_yrs.loc[(year),'TMP_VALUE'])
    ax.legend(np.arange(1980,2030,10))
ax.set(title = 'Fig 6. Averaged Temperature in Each Month 1980-2020',
       facecolor = '#F5F5DC',
       xticks = np.arange(1,13,1),
       xlabel = 'Months',
```

```
        ylabel = 'Temperature (°C)')
ax.grid(color = '1', axis = 'y')
```

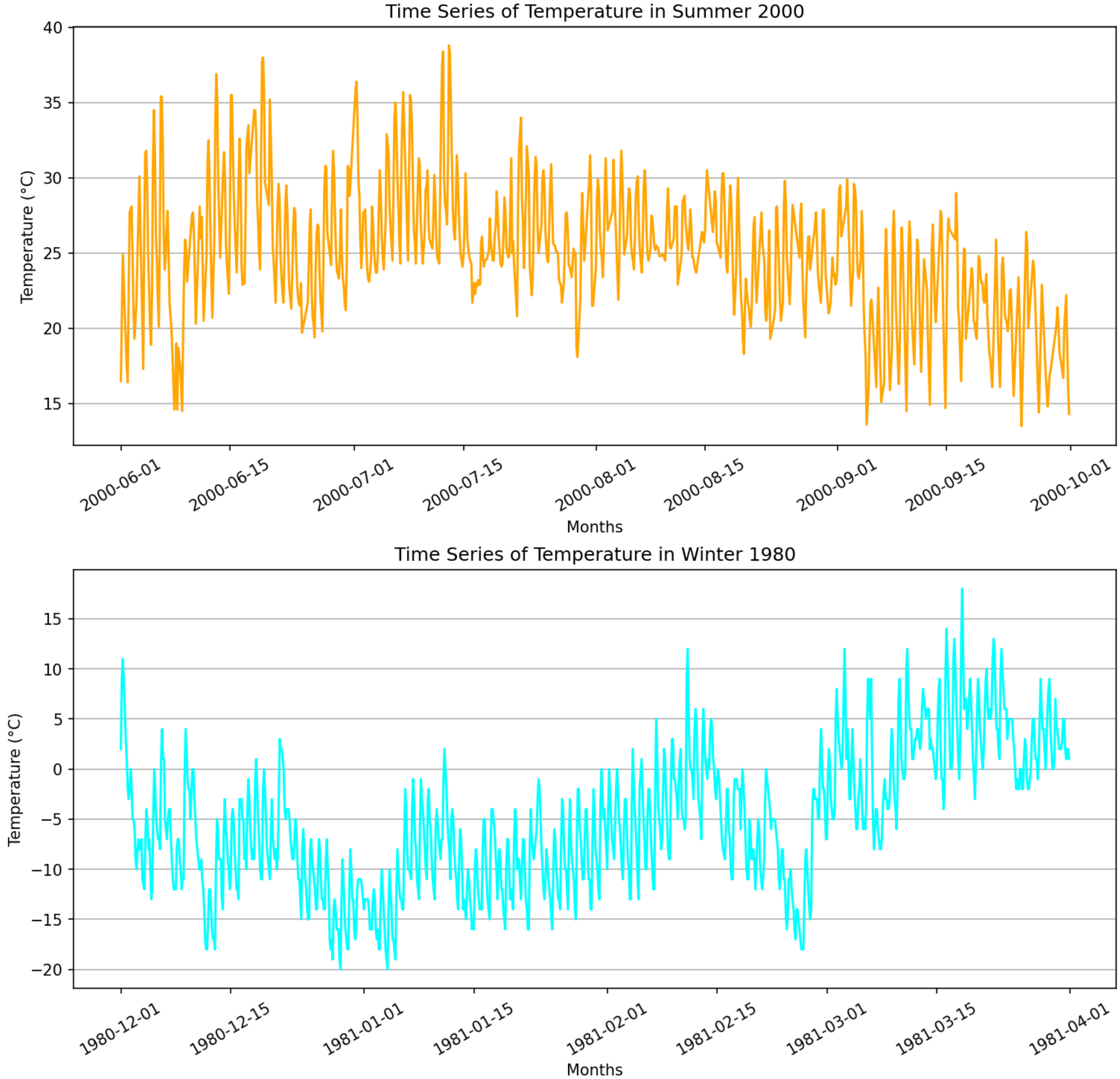


It sees like 1980 is the coldest year, having both the coldest winter ann the coldest summer, while 2000 experienced a significantly high-temperature summer and quite a cold spring also.

In [71]:

```
# Let's take a closer look like temperature in the Summer of 2000
fig, axes = plt.subplots(2,1,figsize = (10,10), dpi=150, layout = 'constrained')
plt.suptitle('Fig. 7 Detailed Time Series of Summer 2000 and Winter 1980',fontsize = 15)
axes[0].plot(jz_df.loc['2000-06':'2000-9','TMP_VALUE'], color = 'orange')
axes[0].set(title = 'Time Series of Temperature in Summer 2000',
        xlabel = 'Months',
        ylabel = 'Temperature (°C)')
axes[0].grid(axis = 'y')
axes[0].xaxis.set_tick_params(rotation = 30)
# And the Winter of 1980
axes[1].plot(jz_df.loc['1980-12':'1981-03','TMP_VALUE'], color = 'aqua')
axes[1].set(title = 'Time Series of Temperature in Winter 1980',
        xlabel = 'Months',
        ylabel = 'Temperature (°C)')
axes[1].grid(axis = 'y')
axes[1].xaxis.set_tick_params(rotation = 30)
```

Fig. 7 Detailed Time Series of Summer 2000 and Winter 1980



In [72]:
```
# One more thing, I want to look at rainfall trend.
jz_rf = pd.read_csv('data_files\Jinzhou_Precipitation.csv',usecols = ['DATE','PRCP'])
jz_rf.head()
```

Out[72]:

	DATE	PRCP
0	1951/1/1	0.0
1	1951/1/2	11.0
2	1951/1/3	0.0
3	1951/1/4	0.0
4	1951/1/5	0.0

In [73]:
```
# Try to plot Annual Precipitation from 1951 to 2022
jz_rf['DATE'] = pd.to_datetime(jz_rf['DATE'])
jz_rf['YEAR'] = jz_rf['DATE'].dt.year
jz_rf['MONTH'] = jz_rf['DATE'].dt.month
```

In [74]:
```
prcp = jz_rf.groupby('YEAR')['PRCP'].sum().map(lambda x:x/10)
prcp
```

Out[74]:
```
YEAR
1951    493.3
1952    334.7
1953    699.7
1954    615.9
1955    601.3
        ...
2018    495.4
2019    641.0
2020    526.9
2021    967.4
2022    538.7
Name: PRCP, Length: 72, dtype: float64
```

In [75]:
```
fig, ax = plt.subplots(figsize = (12,5),dpi = 100)
ax.plot(prcp)
ax.set(title = 'Fig 8. Annual Precipitation 1951-2022',
       facecolor = '#F0FFFF',
       xticks = np.arange(1950,2024,2),
       xlabel = 'Years',
       ylabel = 'Precipitation (mm)')
ax.xaxis.set_tick_params(rotation=45)
ax.grid(color = 'silver', axis = 'both', ls = 'dashed')
```