Python for Hydrology

Session 5 – Precipitation Data Manipulation

Objective:

Manipulate long tables of precipitation and streamflow, visualize the data, filter it, and make relationships.

Precipitation

Start creating the notebook and folder for the Session 5

Import the libraries which we are going to work with

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime
```

Copy and paste the ".txt" files located in the folder "**Data**" of this Session. The files are

- precipitation.txt
- streamflow.txt

Let's try to **read** the "precipitation.txt" file

```
pp = pd.read_csv('precipitation.txt')
```

The previous command returns an **error**

```
ParserError: Error tokenizing data. C error: Expected 1 fields in line
17, saw 2
```

If you **open the ".txt" file** (You can open it in Jupyter Lab), you could find that there is a big heading.

```
# ---------------------------------- WARNING ----------------------------------------
# Some of the data that you have obtained from this U.S. Geological Survey database
# may not have received Director's approval. Any such data values are qualified
# as provisional and are subject to revision. Provisional data are released on the
# condition that neither the USGS nor the United States Government may be held liable
# for any damages resulting from its use.
#
# Additional info: https://help.waterdata.usgs.gov/policies/provisional-data-statement
#
# File-format description:  https://help.waterdata.usgs.gov/faq/about-tab-delimited-output
# Automated-retrieval info: https://help.waterdata.usgs.gov/faq/automated-retrievals
#
# Contact:   gs-w_support_nwisweb@usgs.gov
# retrieved: 2020-10-09 10:58:51 EDT       (caww01)
#
# Data for the following 1 site(s) are contained in this file
#    USGS 393938104572101  HARVARD GUL PRECIP STA AT SLAVENS SC AT DENVER,CO
# -----------------------------------------------------------------------------------
#
# Data provided for site 393938104572101
#            TS   parameter     statistic     Description
#         20300       00045     00006     Precipitation, total, inches (Sum)
#
# Data-value qualification codes included in this output:
#
#     A  Approved for publication -- Processing and review completed.
#     P  Provisional data subject to revision.
#
agency_cd	site_no	datetime	20300_00045_00006	20300_00045_00006_cd
5s	15s	20d	14n	10s
USGS	393938104572101	2009-05-05	0.00	A
USGS	393938104572101	2009-05-06	0.00	A
USGS	393938104572101	2009-05-07	0.00	A
USGS	393938104572101	2009-05-08	0.01	A
USGS	393938104572101	2009-05-09	0.08	A
```

We need to **skip this heading** by the number of rows that have commented text, which is equal to 28.

```
pp = pd.read_csv('precipitation.txt',skiprows=28)
pp.head()
```

```
pp = pd.read_csv('precipitation.txt',skiprows=28)
pp.head()
```

	agency_cd\tsite_no\tdatetime\t20300_00045_00006\t20300_00045_00006_cd
0	5s\t15s\t20d\t14n\t10s
1	USGS\t393938104572101\t2009-05-05\t0.00\tA
2	USGS\t393938104572101\t2009-05-06\t0.00\tA
3	USGS\t393938104572101\t2009-05-07\t0.00\tA
4	USGS\t393938104572101\t2009-05-08\t0.01\tA

If you see the table's text carefully, you could find that there is a "\t" separator or more commonly called "tab." You need to **make Python know the kind of delimiter.**

```
pp = pd.read_csv('precipitation.txt',skiprows=28, delimiter='\t',)
pp.head()
```

```
pp = pd.read_csv('precipitation.txt',skiprows=28, delimiter='\t',)
pp.head()
```

	agency_cd	site_no	datetime	20300_00045_00006	20300_00045_00006_cd
0	5s	15s	20d	14n	10s
1	USGS	393938104572101	2009-05-05	0.00	A
2	USGS	393938104572101	2009-05-06	0.00	A
3	USGS	393938104572101	2009-05-07	0.00	A
4	USGS	393938104572101	2009-05-08	0.01	A

If you **look at the row with index 1**, you could see strange values. These values are related to the extend of the values in the column. We don't need them, so we get rid of them by filtering with the index.

```
pp = pp.iloc[1:]
pp
```

```
pp = pp.iloc[1:]
pp.head()
```

	agency_cd	site_no	datetime	20300_00045_00006	20300_00045_00006_cd
1	USGS	393938104572101	2009-05-05	0.00	A
2	USGS	393938104572101	2009-05-06	0.00	A
3	USGS	393938104572101	2009-05-07	0.00	A
4	USGS	393938104572101	2009-05-08	0.01	A
5	USGS	393938104572101	2009-05-09	0.08	A

To **change the names of the columns**

```
pp.columns = ['Agency', 'SiteNumber', 'datetime', 'pp' , 'code']
pp
```

```
pp.columns = ['Agency', 'SiteNumber', 'datetime', 'pp' , 'code']
pp.head()
```

	Agency	SiteNumber	datetime	pp	code
1	USGS	393938104572101	2009-05-05	0.00	A
2	USGS	393938104572101	2009-05-06	0.00	A
3	USGS	393938104572101	2009-05-07	0.00	A
4	USGS	393938104572101	2009-05-08	0.01	A
5	USGS	393938104572101	2009-05-09	0.08	A

I want to add the precipitation unit (which is inches) and make all of them start with a capitalized letter, so we can **change the names** of which are not well-formatted.

```
pp = pp.rename(columns =
{'datetime':'Datetime','pp':"Precipitation_in",'code':"Code"})
pp
```

```
pp = pp.rename(columns = {'datetime':'Datetime','pp':"Precipitation_in",'code':"Code"})
pp.head()
```

	Agency	SiteNumber	Datetime	Precipitation_in	Code
1	USGS	393938104572101	2009-05-05	0.00	A
2	USGS	393938104572101	2009-05-06	0.00	A
3	USGS	393938104572101	2009-05-07	0.00	A
4	USGS	393938104572101	2009-05-08	0.01	A
5	USGS	393938104572101	2009-05-09	0.08	A

The **index** is not modified automatically, so let's **reset it**.

```
pp.reset_index()
```

```
pp.reset_index()
```

	index	Agency	SiteNumber	Datetime	Precipitation_in	Code
0	1	USGS	393938104572101	2009-05-05	0.00	A
1	2	USGS	393938104572101	2009-05-06	0.00	A
2	3	USGS	393938104572101	2009-05-07	0.00	A
3	4	USGS	393938104572101	2009-05-08	0.01	A
4	5	USGS	393938104572101	2009-05-09	0.08	A
...	...	...	...	...	...	...
4170	4171	USGS	393938104572101	2020-10-04	NaN	NaN
4171	4172	USGS	393938104572101	2020-10-05	NaN	NaN
4172	4173	USGS	393938104572101	2020-10-06	NaN	NaN
4173	4174	USGS	393938104572101	2020-10-07	Ssn	P
4174	4175	USGS	393938104572101	2020-10-08	Ssn	P

But beware! because if you do the previous script, **it creates a new column**. You need to specify an extra argument as follows:

```
pp = pp.reset_index(drop=True)
```

```
pp.head()
```

```
pp = pp.reset_index(drop=True)
pp.head()
```

	Agency	SiteNumber	Datetime	Precipitation_in	Code
0	USGS	393938104572101	2009-05-05	0.00	A
1	USGS	393938104572101	2009-05-06	0.00	A
2	USGS	393938104572101	2009-05-07	0.00	A
3	USGS	393938104572101	2009-05-08	0.01	A
4	USGS	393938104572101	2009-05-09	0.08	A

Let's **plot the precipitation**, but when you try to do that, you get an error message that mentions no numeric data to plot.

```
pp['Precipitation_in'].plot()
```

```
~\miniconda3\lib\site-packages\pandas\plotting\_matplotlib\core.py in _compute_plot_data(self)
    416         # no non-numeric frames or series allowed
    417         if is_empty:
--> 418             raise TypeError("no numeric data to plot")
    419
    420         # GH25587: cast ExtensionArray of pandas (IntegerArray, etc.) to

TypeError: no numeric data to plot
```

The previous error happens because we are trying to plot non-numerical values. If you see the tail of the table, **you find some string-format values**.

```
pp['Precipitation_in'].tail()
```

```
pp['Precipitation_in'].tail()
```

```
4170    NaN
4171    NaN
4172    NaN
4173    Ssn
4174    Ssn
Name: Precipitation_in, dtype: object
```

A quick way to **convert the values** is by using the following function. It tries to convert all the rows into numbers, but it prompts an error as it finds some text values.

```
pd.to_numeric(pp['Precipitation_in'])
```

```
pd.to_numeric(pp['Precipitation_in'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
pandas\_libs\lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "Ssn"

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-11-9d829a27102e> in <module>
----> 1 pd.to_numeric(pp['Precipitation_in'])

~\miniconda3\lib\site-packages\pandas\core\tools\numeric.py in to_numeric(arg, errors, downcast)
    150         coerce_numeric = errors not in ("ignore", "raise")
    151         try:
--> 152             values = lib.maybe_convert_numeric(
    153                 values, set(), coerce_numeric=coerce_numeric
    154             )

pandas\_libs\lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "Ssn" at position 4173
```

You need to specify an extra argument to **force the conversion of the string**. It converts the strings into “nan” values.

```
pd.to_numeric(pp['Precipitation_in'], errors='coerce')
```

```
pd.to_numeric(pp['Precipitation_in'], errors='coerce')
```

```
0       0.00
1       0.00
2       0.00
3       0.01
4       0.08
        ...
4170     NaN
4171     NaN
4172     NaN
4173     NaN
4174     NaN
Name: Precipitation_in, Length: 4175, dtype: float64
```

We can filter those values using **Booleans**, identifying which are “nan”

```
pd.to_numeric(pp['Precipitation_in'], errors='coerce').notnull()
```

```
pd.to_numeric(pp['Precipitation_in'], errors='coerce').notnull()
```

```
0        True
1        True
2        True
3        True
4        True
        ...
4170    False
4171    False
4172    False
4173    False
4174    False
Name: Precipitation_in, Length: 4175, dtype: bool
```

Using the previous script, we can **create a new DataFrame** with the rows that have numerical values of precipitation.

```
pp_filtered = pp[pd.to_numeric(pp['Precipitation_in'],
errors='coerce').notnull()]
pp_filtered.head()
```

```
pp_filtered = pp[pd.to_numeric(pp['Precipitation_in'], errors='coerce').notnull()]
pp_filtered.head()
```

	Agency	SiteNumber	Datetime	Precipitation_in	Code
0	USGS	393938104572101	2009-05-05	0.00	A
1	USGS	393938104572101	2009-05-06	0.00	A
2	USGS	393938104572101	2009-05-07	0.00	A
3	USGS	393938104572101	2009-05-08	0.01	A
4	USGS	393938104572101	2009-05-09	0.08	A

But if you see the values, you could find that **they are not formatted as numbers**.

```
pp_filtered['Precipitation_in'][0]
```

```
pp_filtered['Precipitation_in'][0]
```

```
'0.00'
```

As there are no more strings in our filtered DataFrame, there is no problem if we **convert it to numeric**

```
pp_filtered['Precipitation_in'] =
pd.to_numeric(pp_filtered['Precipitation_in'])
```

```
pp_filtered['Precipitation_in'] = pd.to_numeric(pp_filtered['Precipitation_in'])
```

```
<ipython-input-16-aa05d2a3e60f>:1: SettingWithCopyWarning:
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/use
-copy
  pp_filtered['Precipitation_in'] = pd.to_numeric(pp_filtered['Precipitation_in'])
```

You can **check the first value and its type**, and it must be a numeric format.

```
pp_filtered['Precipitation_in'][0],
pp_filtered['Precipitation_in'][0].dtype
```

```
pp_filtered['Precipitation_in'][0], pp_filtered['Precipitation_in'][0].dtype
```

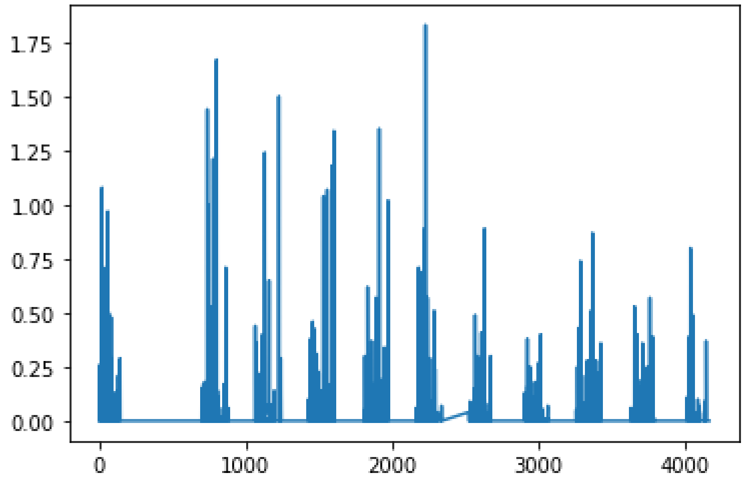
```
(0.0, dtype('float64'))
```

If you **plot** it you get the following view:

```
pp_filtered['Precipitation_in'].plot()
```

```
pp_filtered['Precipitation_in'].plot()
```

```
<AxesSubplot:>
```



Let’s deal with the **date format**. You can see that it has an object type.

```
pp_filtered['Datetime'].dtype
```

```
pp_filtered['Datetime'].dtype
```

```
dtype('O')
```

We can **change it to date** format easily with Pandas.

```
pp_filtered['Datetime'] = pd.to_datetime(pp_filtered['Datetime'])
pp_filtered['Datetime'].iloc[0]
```

```
pp_filtered['Datetime'] = pd.to_datetime(pp_filtered['Datetime'])
pp_filtered['Datetime'].iloc[0]
```

```
<ipython-input-82-a6c8bbb1e016>:1: SettingWithCopyWarning:
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/panda
-copy
  pp_filtered['Datetime'] = pd.to_datetime(pp_filtered['Datetime'])
Timestamp('2009-05-05 00:00:00')
```

Then **make the date column be the index**.

```
pp_filtered = pp_filtered.set_index('Datetime')
pp_filtered.head()
```

```
pp_filtered = pp_filtered.set_index('Datetime')
pp_filtered.head()
```

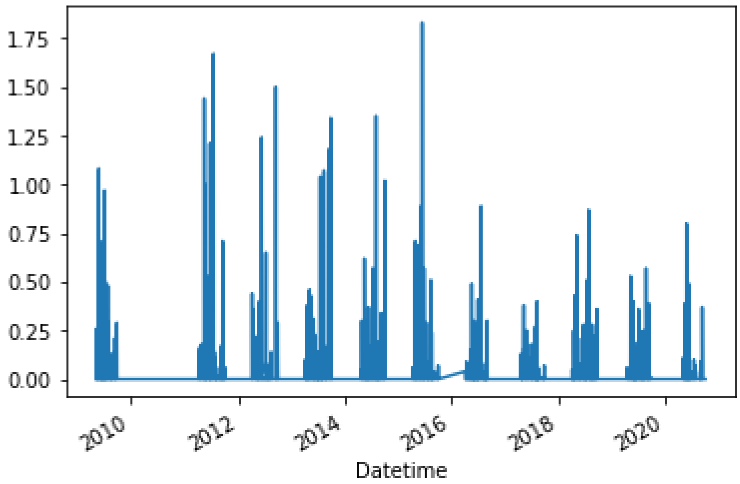
	Agency	SiteNumber	Precipitation_in	Code
Datetime				
2009-05-05	USGS	393938104572101	0.00	A
2009-05-06	USGS	393938104572101	0.00	A
2009-05-07	USGS	393938104572101	0.00	A
2009-05-08	USGS	393938104572101	0.01	A
2009-05-09	USGS	393938104572101	0.08	A

With this set-up, you can get the dates on the X-axis.

```
pp_filtered['Precipitation_in'].plot()
```

```
pp_filtered['Precipitation_in'].plot()
```

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

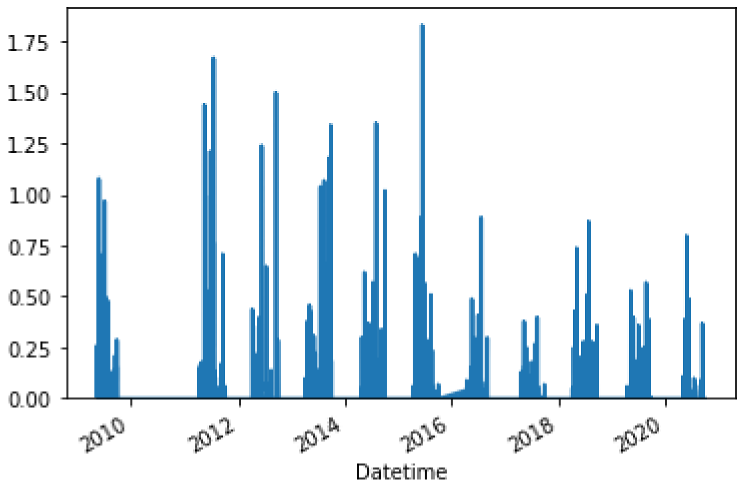


You can plot it as an **area under the lines**.

```
pp_filtered['Precipitation_in'].plot(kind='area')
```

```
pp_filtered['Precipitation_in'].plot(kind='area')
```

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

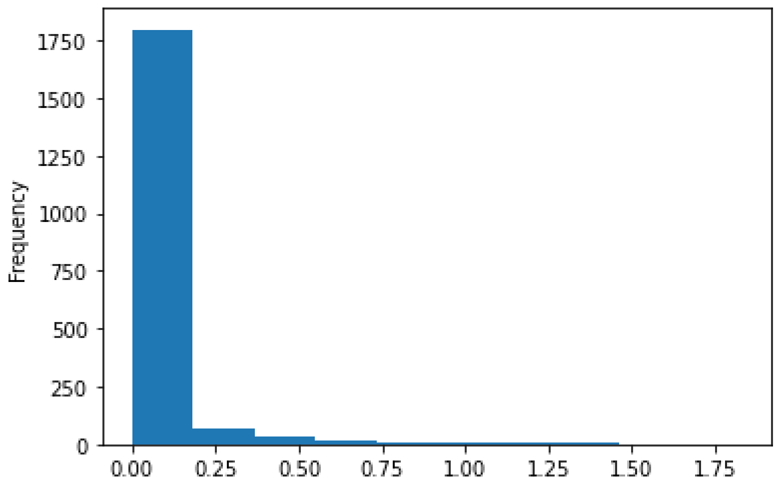


Or as a **histogram**.

```
pp_filtered['Precipitation_in'].plot(kind='hist')
```


```
pp_filtered['Precipitation_in'].plot(kind='hist')
```


```
<AxesSubplot:ylabel='Frequency'>
```

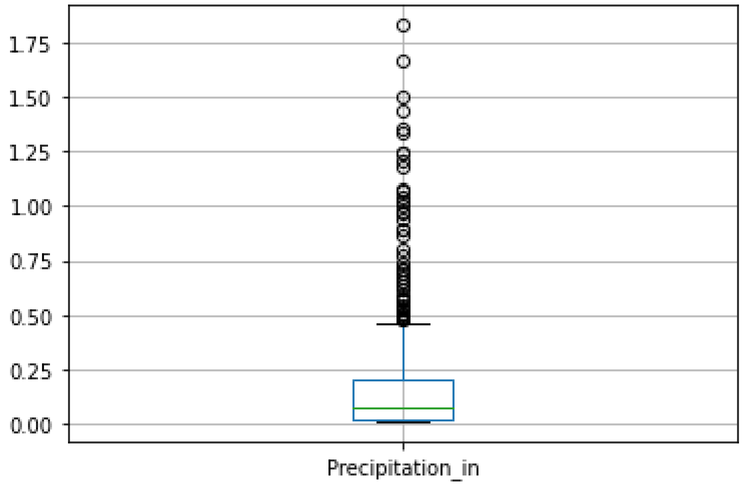


Plot it as a **boxplot** for the values greater than 0.

```
pp_filtered[pp_filtered['Precipitation_in']>0].boxplot()
```

```
pp_filtered[pp_filtered['Precipitation_in']>0].boxplot()
```

```
<AxesSubplot:>
```



The **scatter plot can't be made directly.**

```
pp_filtered['Precipitation_in'].plot(kind='scatter')
```

```
pp_filtered['Precipitation_in'].plot(kind='scatter')
```

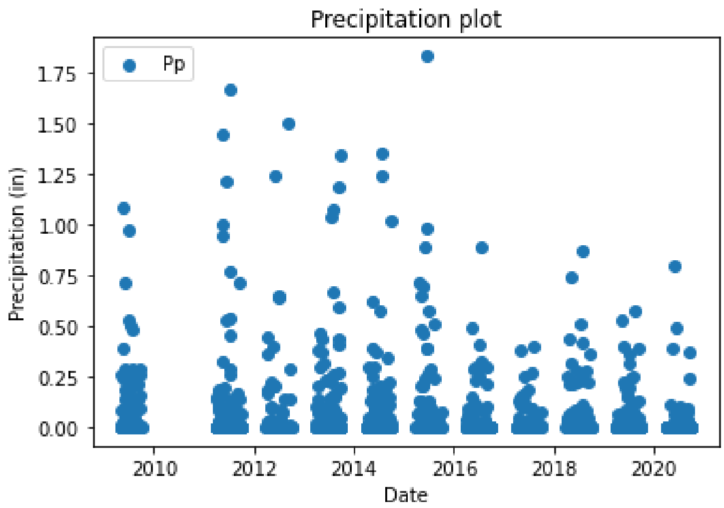
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-3d258a38fc98> in <module>
----> 1 pp_filtered['Precipitation_in'].plot(kind='scatter')

~\miniconda3\lib\site-packages\pandas\plotting\_core.py in __call__(self, *args, **kwargs)
    894                 return plot_backend.plot(data, x=x, y=y, kind=kind, **kwargs)
    895             else:
--> 896                 raise ValueError(f"plot kind {kind} can only be used for data frames")
    897         elif kind in self._series_kinds:
    898             if isinstance(data, ABCDataFrame):

ValueError: plot kind scatter can only be used for data frames
```

But you can make it **declaring a figure.**

```
fig, ax = plt.subplots()
ax.scatter(pp_filtered.index, pp_filtered['Precipitation_in'].values,
label='Pp')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
ax.set_title('Precipitation plot')
```

```
ax.legend(loc='upper left')
plt.show()
```

```
fig, ax = plt.subplots()
ax.scatter(pp_filtered.index, pp_filtered['Precipitation_in'].values, label='Pp')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
ax.set_title('Precipitation plot')
ax.legend(loc='upper left')
plt.show()
```



You can get the **range of dates** regarding the precipitation data with "min" and "max"

```
pp_filtered.index.max(),pp_filtered.index.min()
```

```
pp_filtered.index.max(),pp_filtered.index.min()
```

```
(Timestamp('2020-09-30 00:00:00'), Timestamp('2009-05-05 00:00:00'))
```

You can use "**describe**" to see some statistic values regarding the DataFrame columns

```
pp_filtered['Precipitation_in'].describe()
```

```
pp_filtered['Precipitation_in'].describe()
```

```
count    1941.000000
mean        0.046208
std         0.155938
min         0.000000
25%         0.000000
50%         0.000000
75%         0.010000
max         1.830000
Name: Precipitation_in, dtype: float64
```

You can get the **statistics of a given range.**

```
pp_filtered['Precipitation_in'].loc['2019-07-01':'2020-06-
30'].describe()
```

```
pp_filtered['Precipitation_in'].loc['2019-07-01':'2020-06-30'].describe()
```

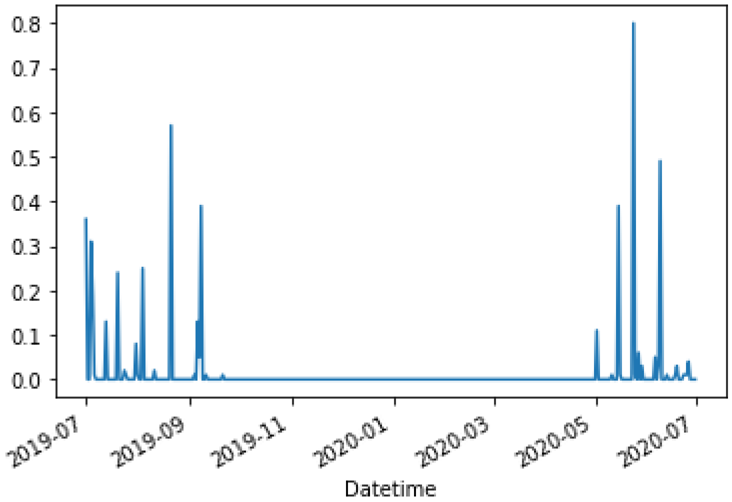
```
count    180.000000
mean       0.027278
std        0.100961
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        0.800000
Name: Precipitation_in, dtype: float64
```

And **plot the range** too.

```
pp_filtered['Precipitation_in'].loc['2019-07-01':'2020-06-30'].plot()
```

```
pp_filtered['Precipitation_in'].loc['2019-07-01':'2020-06-30'].plot()
```

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

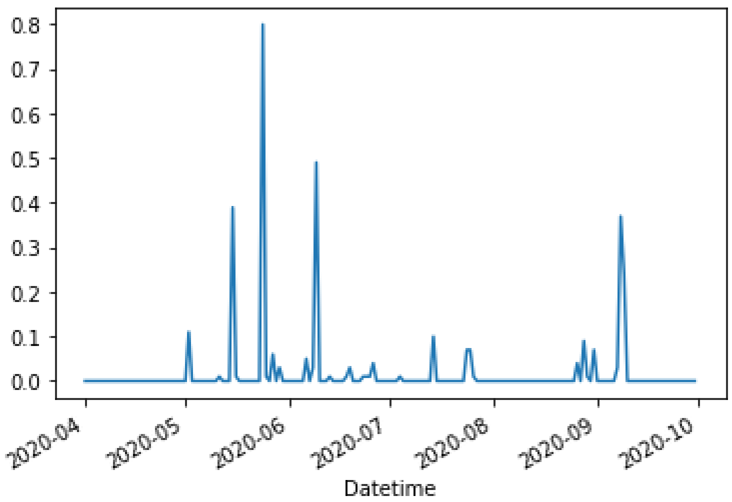


Or plot the values **giving only the starting date**.

```
pp_filtered['Precipitation_in'].loc['2020-04-1':].plot()
```

```
pp_filtered['Precipitation_in'].loc['2020-04-1':].plot()
```

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

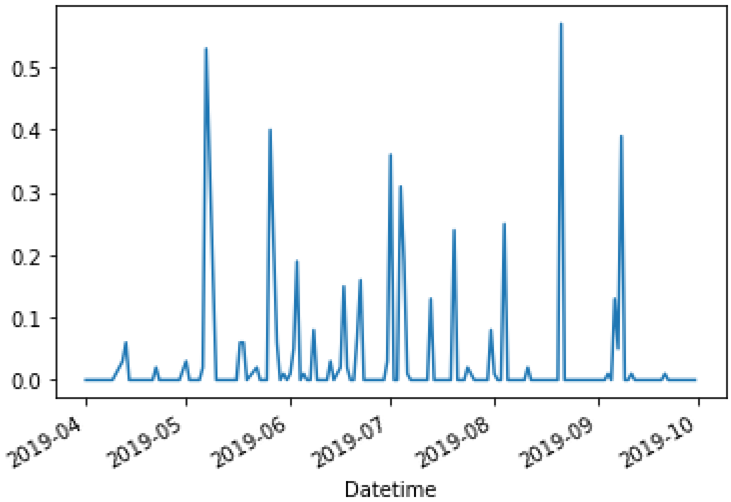


If you check the distribution, you could see that this station's **precipitations are presented from April till October approximately**.

```
pp_filtered['Precipitation_in'].loc['2019-04-1':'2019-09-30'].plot()
```

```
pp_filtered['Precipitation_in'].loc['2019-04-1':'2019-09-30'].plot()
```

<AxesSubplot:xlabel='Datetime'>



Let's create **DataFrames of the years 2015 to 2019 filtering the dates for the months with precipitation.**

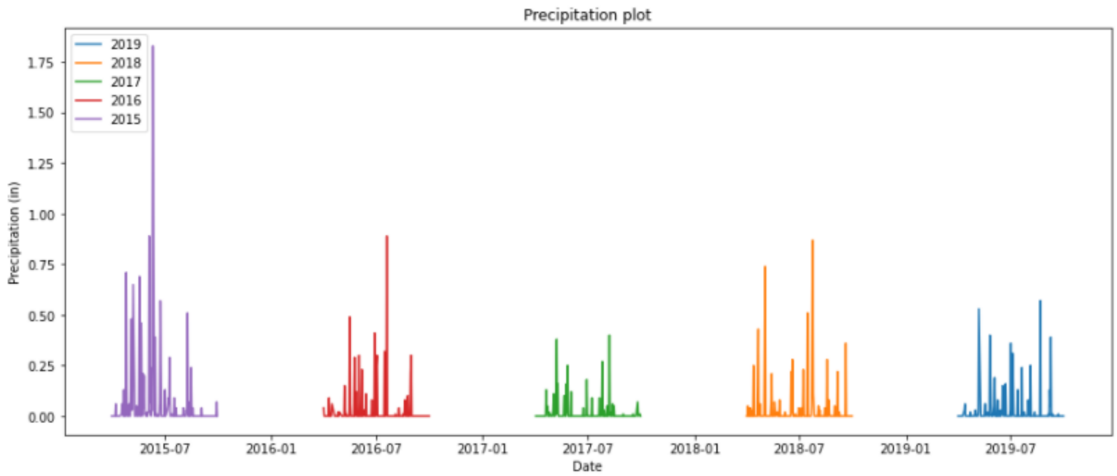
```
y2019 = pp_filtered['Precipitation_in'].loc['2019-04-1':'2019-09-30']
y2018 = pp_filtered['Precipitation_in'].loc['2018-04-1':'2018-09-30']
y2017 = pp_filtered['Precipitation_in'].loc['2017-04-1':'2017-09-30']
y2016 = pp_filtered['Precipitation_in'].loc['2016-04-1':'2016-09-30']
y2015 = pp_filtered['Precipitation_in'].loc['2015-04-1':'2015-09-30']
```

If you **plot them** you get the following:

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(y2019.index, y2019, label='2019')
ax.plot(y2018.index, y2018, label='2018')
ax.plot(y2017.index, y2017, label='2017')
ax.plot(y2016.index, y2016, label='2016')
ax.plot(y2015.index, y2015, label='2015')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
```

```
ax.set_title('Precipitation plot')
ax.legend(loc='upper left')
plt.show()
```

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(y2019.index, y2019, label='2019')
ax.plot(y2018.index, y2018, label='2018')
ax.plot(y2017.index, y2017, label='2017')
ax.plot(y2016.index, y2016, label='2016')
ax.plot(y2015.index, y2015, label='2015')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
ax.set_title('Precipitation plot')
ax.legend(loc='upper left')
plt.show()
```



We cannot plot them with a shared range because, as we filtered them before, there are **differences in each year's data length**.

```
y2019.shape,y2018.shape,y2017.shape,y2016.shape,y2015.shape
```

```
y2019.shape,y2018.shape,y2017.shape,y2016.shape,y2015.shape
```

```
((176,), (183,), (170,), (176,), (177,))
```

Let's create the DataFrame again, but **do not eliminate the "nan" value**s this time.

```
pp['Datetime']=pd.to_datetime(pp['Datetime'])
pp_ordered = pp.set_index('Datetime')
pp_ordered['Precipitation_in'] =
pd.to_numeric(pp_ordered['Precipitation_in'], errors='coerce')
pp_ordered.head()
```

```
pp['Datetime']=pd.to_datetime(pp['Datetime'])
pp_ordered = pp.set_index('Datetime')
pp_ordered['Precipitation_in'] = pd.to_numeric(pp_ordered['Precipitation_in'], errors='coerce')
pp_ordered.head()
```

	Agency	SiteNumber	Precipitation_in	Code
Datetime				
2009-05-05	USGS	393938104572101	0.00	A
2009-05-06	USGS	393938104572101	0.00	A
2009-05-07	USGS	393938104572101	0.00	A
2009-05-08	USGS	393938104572101	0.01	A
2009-05-09	USGS	393938104572101	0.08	A

Create the yearly DataFrames again

```
y2019 = pp_ordered['Precipitation_in'].loc['2019-04-1':'2019-09-30']
y2018 = pp_ordered['Precipitation_in'].loc['2018-04-1':'2018-09-30']
y2017 = pp_ordered['Precipitation_in'].loc['2017-04-1':'2017-09-30']
y2016 = pp_ordered['Precipitation_in'].loc['2016-04-1':'2016-09-30']
y2015 = pp_ordered['Precipitation_in'].loc['2015-04-1':'2015-09-30']
```

Verify that the shape is the same.

```
y2019.shape,y2018.shape,y2017.shape,y2016.shape,y2015.shape
```

```
y2019.shape,y2018.shape,y2017.shape,y2016.shape,y2015.shape
```

```
((183,), (183,), (183,), (183,), (183,))
```

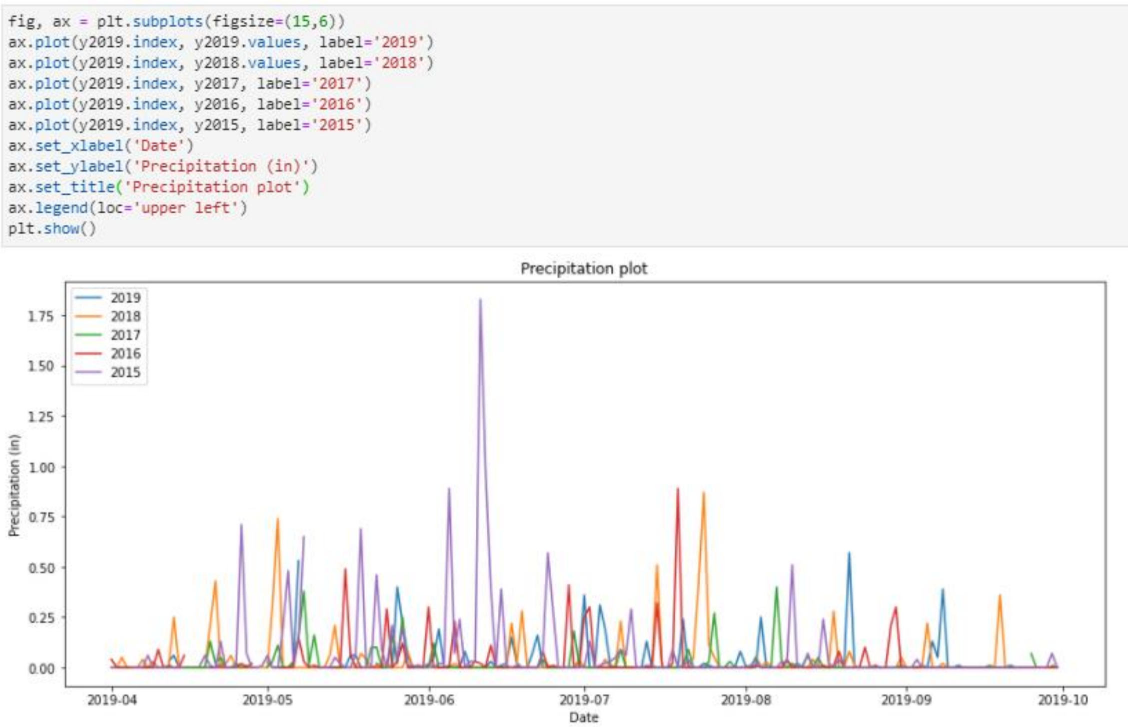
Plot the data again

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(y2019.index, y2019, label='2019')
ax.plot(y2018.index, y2018, label='2018')
ax.plot(y2017.index, y2017, label='2017')
ax.plot(y2016.index, y2016, label='2016')
ax.plot(y2015.index, y2015, label='2015')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
ax.set_title('Precipitation plot')
ax.legend(loc='upper left')
```

```
plt.show()
```

As they have the same index, you can **plot them with the same date range.**

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(y2019.index, y2019.values, label='2019')
ax.plot(y2019.index, y2018.values, label='2018')
ax.plot(y2019.index, y2017, label='2017')
ax.plot(y2019.index, y2016, label='2016')
ax.plot(y2019.index, y2015, label='2015')
ax.set_xlabel('Date')
ax.set_ylabel('Precipitation (in)')
ax.set_title('Precipitation plot')
ax.legend(loc='upper left')
plt.show()
```



If you want to see **how many "nan" values** are there in the DataFrame, subtract the data's length and the "count" of them.

```
len(pp_ordered['Precipitation_in']) -
pp_ordered['Precipitation_in'].count()
```

```
len(pp_ordered['Precipitation_in']) - pp_ordered['Precipitation_in'].count()
```

```
2234
```

You can do it in another way too:

```
pp_ordered['Precipitation_in'].isnull().sum()
```

```
pp_ordered['Precipitation_in'].isnull().sum()
```

```
2234
```

If you want to see the **number of days in the DataFrame**, you can subtract the last record with the first one.

```
pp_ordered.index[-1] - pp_ordered.index[0]
```

```
pp_ordered.index[-1] - pp_ordered.index[0]
```

```
Timedelta('4174 days 00:00:00')
```

You can check if the value makes sense by **adding 4174 days to the starting date.**

```
pp_ordered.index[0] + datetime.timedelta(days=4174)
```

```
pp_ordered.index[0] + datetime.timedelta(days=4174)
```

```
Timestamp('2020-10-08 00:00:00')
```

Based on the previous DataFrame, we can **create columns of the Year, Month, and Day.**

```
pp_ordered['Day'] = pp_ordered.index.day
pp_ordered['Month'] = pp_ordered.index.month
pp_ordered['Year'] = pp_ordered.index.year
```

```
pp_ordered
```

```
pp_ordered['Day'] = pp_ordered.index.day
pp_ordered['Month'] = pp_ordered.index.month
pp_ordered['Year'] = pp_ordered.index.year
pp_ordered
```

Datetime	Agency	SiteNumber	Precipitation_in	Code	Day	Month	Year
2009-05-05	USGS	393938104572101	0.00	A	5	5	2009
2009-05-06	USGS	393938104572101	0.00	A	6	5	2009
2009-05-07	USGS	393938104572101	0.00	A	7	5	2009
2009-05-08	USGS	393938104572101	0.01	A	8	5	2009
2009-05-09	USGS	393938104572101	0.08	A	9	5	2009
...	...	...	...	...	...	...	...
2020-10-04	USGS	393938104572101	NaN	NaN	4	10	2020
2020-10-05	USGS	393938104572101	NaN	NaN	5	10	2020
2020-10-06	USGS	393938104572101	NaN	NaN	6	10	2020
2020-10-07	USGS	393938104572101	NaN	P	7	10	2020
2020-10-08	USGS	393938104572101	NaN	P	8	10	2020

4175 rows × 7 columns

We can make a count of them by year using "**Groupby**"

```
pp_ordered.groupby('Year').count()
```

```
pp_ordered.groupby('Year').count()
```

Year	Agency	SiteNumber	Precipitation_in	Code	Day	Month
2009	241	241	149	149	241	241
2010	365	365	0	0	365	365
2011	365	365	187	187	365	365
2012	366	366	183	183	366	366
2013	365	365	183	183	365	365
2014	365	365	178	178	365	365
2015	365	365	177	177	365	365
2016	366	366	176	176	366	366
2017	365	365	170	170	365	365
2018	365	365	183	183	365	365
2019	365	365	176	176	365	365
2020	282	282	179	181	282	282

Group by Month

```
pp_ordered.groupby('Month').count()
```

```
pp_ordered.groupby('Month').count()
```

Month	Agency	SiteNumber	Precipitation_in	Code	Day	Year
1	341	341	0	0	341	341
2	311	311	0	0	311	311
3	341	341	0	0	341	341
4	330	330	273	273	330	330
5	368	368	327	327	368	368
6	360	360	330	330	360	360
7	372	372	341	341	372	372
8	372	372	340	340	372	372
9	360	360	326	326	360	360
10	349	349	4	6	349	349
11	330	330	0	0	330	330
12	341	341	0	0	341	341

Make **monthly summations** of values with a fixed date range.

```
pp_ordered[['Month','Precipitation_in']].loc['2019-1-1':'2019-12-
31'].groupby('Month').sum()
```

```
pp_ordered[['Month','Precipitation_in']].loc['2019-1-1':'2019-12-31'].groupby('Month').sum()
```

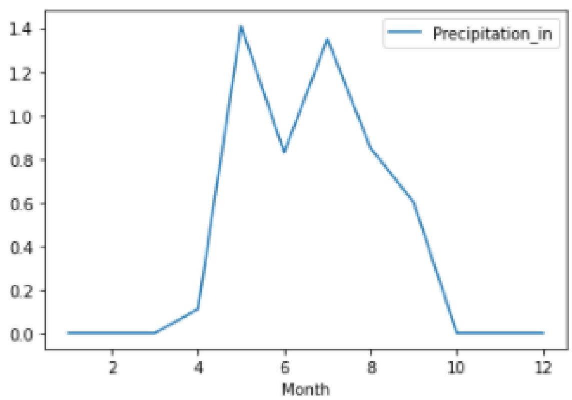
Month	Precipitation_in
1	0.00
2	0.00
3	0.00
4	0.11
5	1.41
6	0.83
7	1.35
8	0.85
9	0.60
10	0.00
11	0.00
12	0.00

Plot the Monthy cumulative precipitations for a given date range.

```
pp_ordered[['Month','Precipitation_in']].loc['2019-1-1':'2019-12-31'].groupby('Month').sum().plot()
```

```
pp_ordered[['Month','Precipitation_in']].loc['2019-1-1':'2019-12-31'].groupby('Month').sum().plot()
```

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

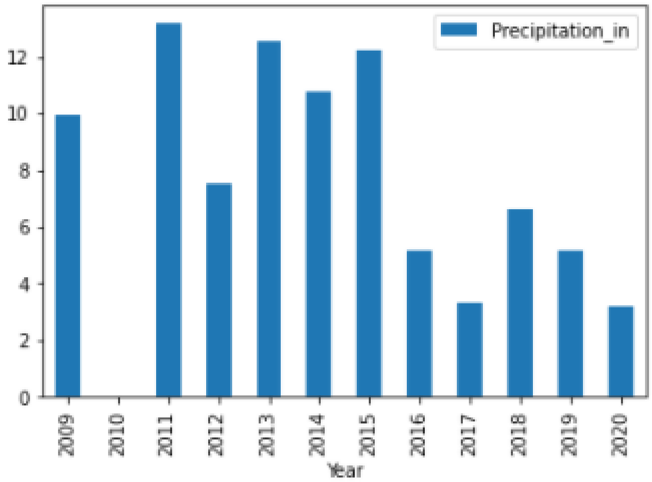


Or make a **bar plot**.

```
pp_ordered[['Year','Precipitation_in']].groupby('Year').sum().plot(kin
d='bar')
```

```
pp_ordered[['Year','Precipitation_in']].groupby('Year').sum().plot(kind='bar')
```

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



Export the "**pp_ordered**" DataFrame as csv.

```
pp_ordered.to_csv('pp.csv')
```

Streamflow

As we have done before, we are going to **open a streamflow file.**

```
streamflow =
pd.read_csv('streamflow.txt',skiprows=30,delimiter='\t',names=['Agency
', 'SiteNumber', 'Date', 'Flow_cfs' , 'Code']).iloc[1:]
streamflow
```

```
streamflow = pd.read_csv('streamflow.txt',skiprows=30,delimiter='\t',names=['Agency', 'SiteNumber', 'Date', 'Flow_cfs' , 'Code']).iloc[1:]
streamflow.head()
```

	Agency	SiteNumber	Date	Flow_cfs	Code
1	USGS	06719505	2000-01-01	83.0	A:e
2	USGS	06719505	2000-01-02	83.0	A:e
3	USGS	06719505	2000-01-03	84.0	A:e
4	USGS	06719505	2000-01-04	82.0	A:e
5	USGS	06719505	2000-01-05	81.0	A:e

Convert its values to a numerical type.

```
streamflow['Flow_cfs'] =
pd.to_numeric(streamflow['Flow_cfs'],errors='coerce')
streamflow.head()
```

```
streamflow['Flow_cfs'] = pd.to_numeric(streamflow['Flow_cfs'],errors='coerce')
streamflow.head()
```

	Agency	SiteNumber	Date	Flow_cfs	Code
1	USGS	06719505	2000-01-01	83.0	A:e
2	USGS	06719505	2000-01-02	83.0	A:e
3	USGS	06719505	2000-01-03	84.0	A:e
4	USGS	06719505	2000-01-04	82.0	A:e
5	USGS	06719505	2000-01-05	81.0	A:e

Treat the “dates” column

```
streamflow['Date'] = pd.to_datetime(streamflow['Date'])
streamflow['Date'][1]
```

```
streamflow['Date'] = pd.to_datetime(streamflow['Date'])
streamflow['Date'][1]
```

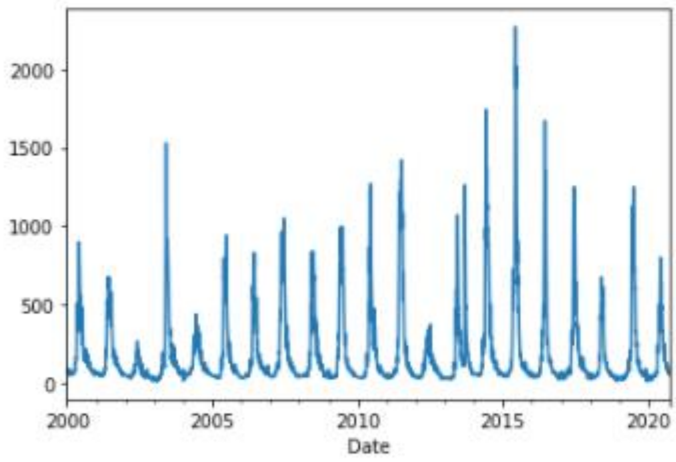
```
Timestamp('2000-01-01 00:00:00')
```

Make a **plot of the flow** (which is in cubic feet per second)

```
streamflow['Flow_cfs'].plot()
```

```
streamflow['Flow_cfs'].plot()
```

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



As we have more data regarding the streamflow than the precipitation values, let's **crop it to the range of the precipitation file dates.**

```
streamflow =
streamflow.loc[pp_ordered.index.min():pp_ordered.index.max()]
streamflow.head()
```

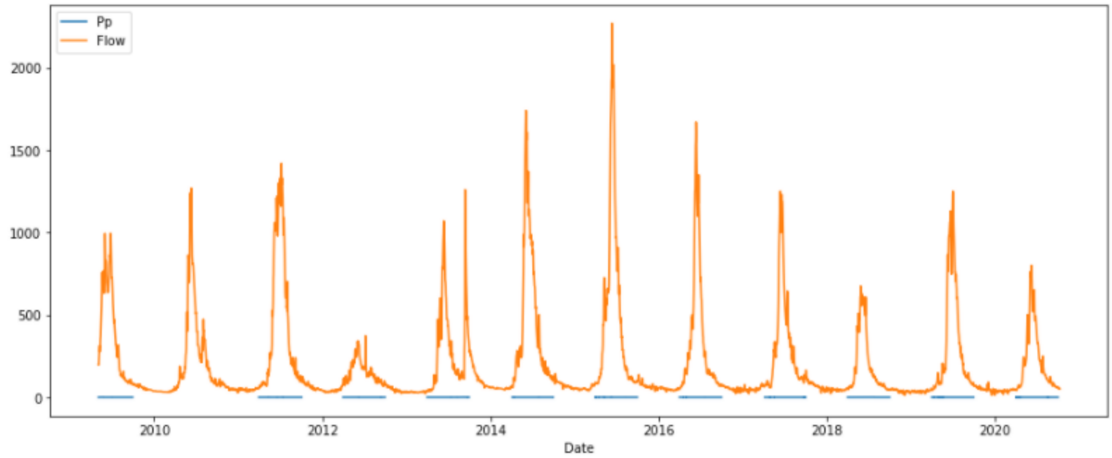
```
streamflow = streamflow.loc[pp_ordered.index.min():pp_ordered.index.max()]
streamflow.head()
```

Date	Agency	SiteNumber	Flow_cfs	Code
2009-05-05	USGS	06719505	198.0	A
2009-05-06	USGS	06719505	196.0	A
2009-05-07	USGS	06719505	235.0	A
2009-05-08	USGS	06719505	286.0	A
2009-05-09	USGS	06719505	290.0	A

If we **plot both of them**, we could see the precipitation as lines because of its low values.

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(pp_ordered.index, pp_ordered['Precipitation_in'], label='Pp')
ax.plot(streamflow.index, streamflow['Flow_cfs'], label='Flow')
ax.set_xlabel('Date')
ax.legend(loc='upper left')
plt.show()
```

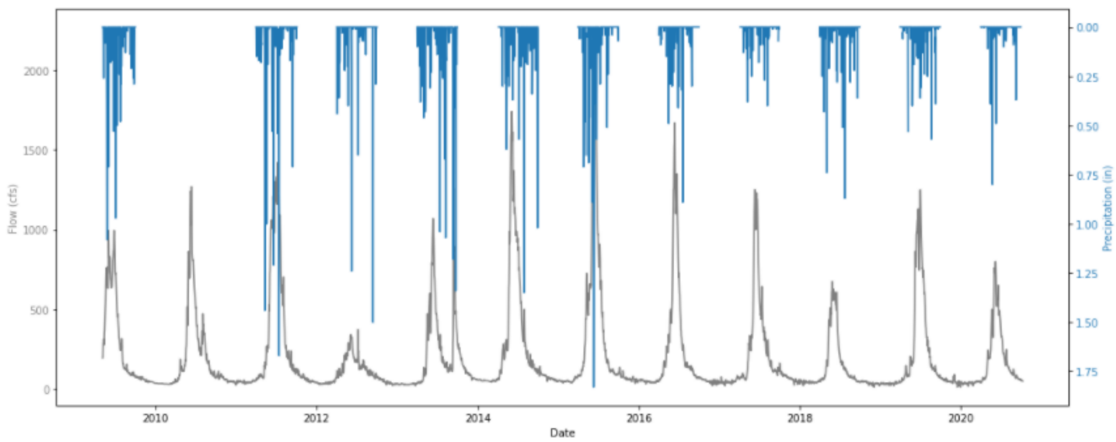

```
fig, ax = plt.subplots(figsize=(15,6))
ax.plot(pp_ordered.index, pp_ordered['Precipitation_in'], label='Pp')
ax.plot(streamflow.index, streamflow['Flow_cfs'], label='Flow')
ax.set_xlabel('Date')
ax.legend(loc='upper left')
plt.show()
```

We could relate the streamflow with the precipitation values to **plot the precipitation in a different axis**. As a result, we can see a relationship with the peaks in the streamflow and precipitation peaks.

```
fig, ax = plt.subplots(figsize=(15,6))

color = 'tab:gray'
ax.set_xlabel('Date')
ax.set_ylabel('Flow (cfs)', color=color)
ax.plot(streamflow.index, streamflow['Flow_cfs'], color=color)
ax.tick_params(axis='y', labelcolor=color)

ax1 = ax.twinx()

color = 'tab:blue'
ax1.set_ylabel('Precipitation (in)', color=color)#, color=color
ax1.plot(pp_ordered.index, pp_ordered['Precipitation_in'],
color=color)
ax1.tick_params(axis='y', labelcolor=color)#
ax1.set_ylim(ax1.get_ylim()[::-1])

fig.tight_layout()
plt.show()
```


```
fig, ax = plt.subplots(figsize=(15,6))

color = 'tab:gray'
ax.set_xlabel('Date')
ax.set_ylabel('Flow (cfs)', color=color)
ax.plot(streamflow.index, streamflow['Flow_cfs'], color=color)
ax.tick_params(axis='y', labelcolor=color)

ax1 = ax.twinx()

color = 'tab:blue'
ax1.set_ylabel('Precipitation (in)', color=color)#, color=color
ax1.plot(pp_ordered.index, pp_ordered['Precipitation_in'], color=color)
ax1.tick_params(axis='y', labelcolor=color)#
ax1.set_ylim(ax1.get_ylim()[::-1])

fig.tight_layout()
plt.show()
```

Finally, let’s **save the streamflow file** for using it later.

```
streamflow.to_csv('sf.csv')
```