|Sustainable water management

Python for Hydrology

Session 7 – Statistics II

Objective:

Develop interpolations, correlations, and calculate uncertainty intervals.

Interpolation

Most of what we deal with in this Session is not useful for precipitation trends or long records of precipitations. However, they are useful for post-processing or pre-processing any information.

Create the folder and notebook for this Session.

Start importing the libraries needed.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d
import scipy.stats as st
```

Interpolations are useful when you have data gaps, first generate two arrays "x", "y"

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(-x)
```

From Scipy, we are going to use "interp1d" which makes a 1-dimensional interpolation. It returns an interpolation function

```
f = interp1d(x, y)
f
```

```
f = interp1d(x, y)
f
```

```
<scipy.interpolate.interpolate.interp1d at 0x2051507cf40>
```

This allows us to pass any "X" value and returns us the "Y" interpolated values.

```
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
ynew
```

```
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
ynew
```

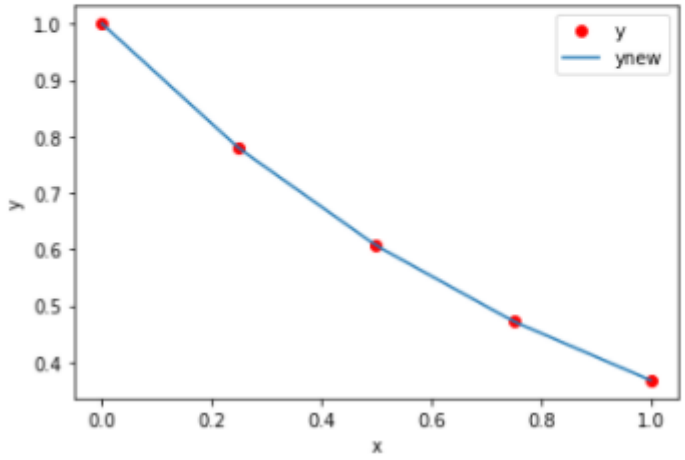
```
array([1.        , 0.98194292, 0.96388584, 0.94582876, 0.92777168,
       0.90971461, 0.89165753, 0.87360045, 0.85554337, 0.83748629,
       0.81942921, 0.80137213, 0.78331505, 0.76825363, 0.75419077,
       0.7401279 , 0.72606503, 0.71200216, 0.6979393 , 0.68387643,
       0.66981356, 0.65575069, 0.64168783, 0.62762496, 0.61356209,
       0.60105457, 0.5901024 , 0.57915023, 0.56819806, 0.55724589,
       0.54629371, 0.53534154, 0.52438937, 0.5134372 , 0.50248503,
       0.49153285, 0.48058068, 0.47023416, 0.4617046 , 0.45317504,
       0.44464548, 0.43611592, 0.42758636, 0.4190568 , 0.41052724,
       0.40199768, 0.39346812, 0.38493856, 0.376409  , 0.36787944])
```

Now, we can plot the points and the interpolated data.

```
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x205170e8250>
```



We can also specify unique values as follows:

```
f(0.5)
```

```
f(0.5)
```

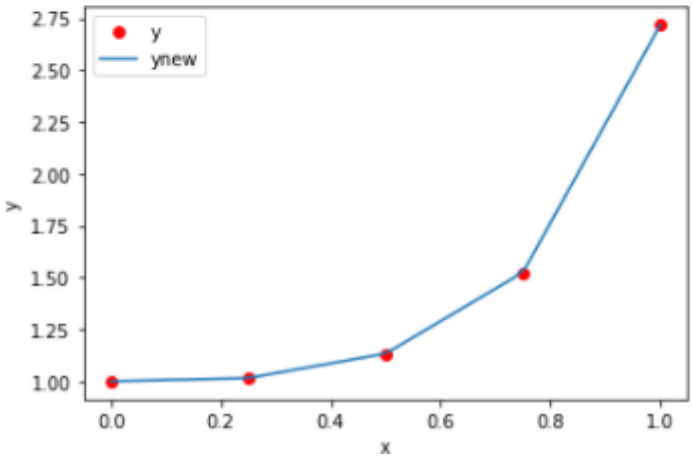
```
array(0.60653066)
```

The advantage of using this kind of interpolation is that it returns the interpolation of any kind of values.

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y)
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y)
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x205185cf490>
```

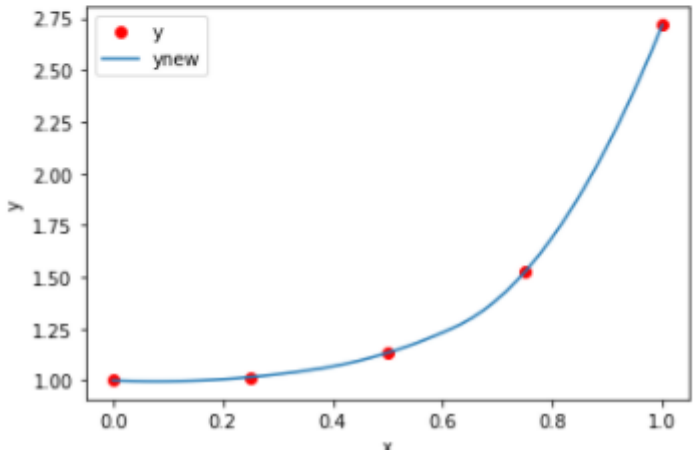


By default, the method is specified with linear interpolation, but we can change it to a quadratic.

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y,'quadratic')
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y,'quadratic')
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2051a674970>
```

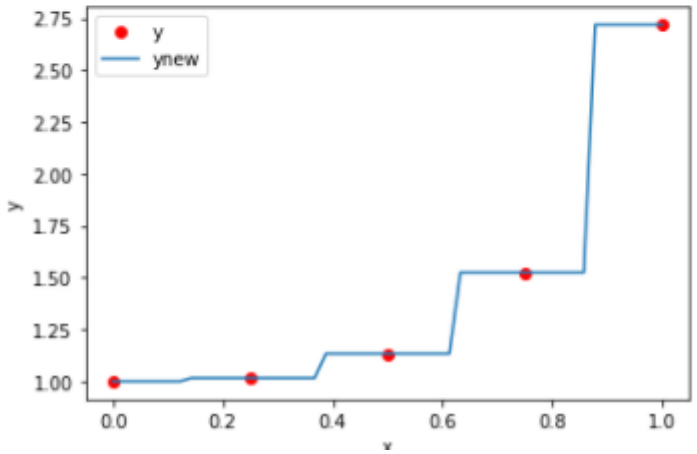


Or any other methods that you could find in the documentation of Scipy

```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y,'nearest')
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

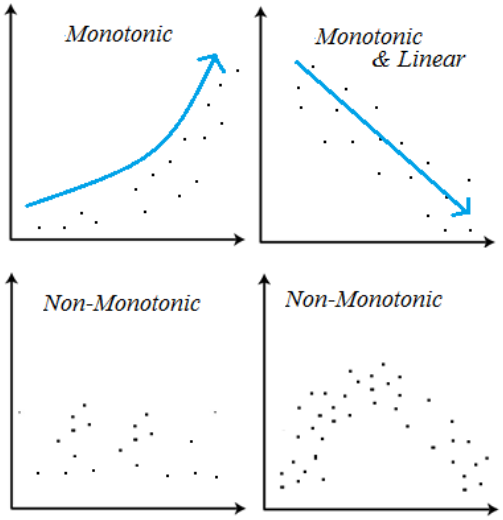
```
# generate data
x = np.linspace(0,1,5)
y = np.exp(x**3)
f = interp1d(x, y,'nearest')
xnew = np.linspace(x.min(), x.max())
ynew = f(xnew)
# plot
plt.plot(x, y, 'ro', label='y')
plt.plot(xnew, ynew, '-', label='ynew')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x205182c6b20>
```

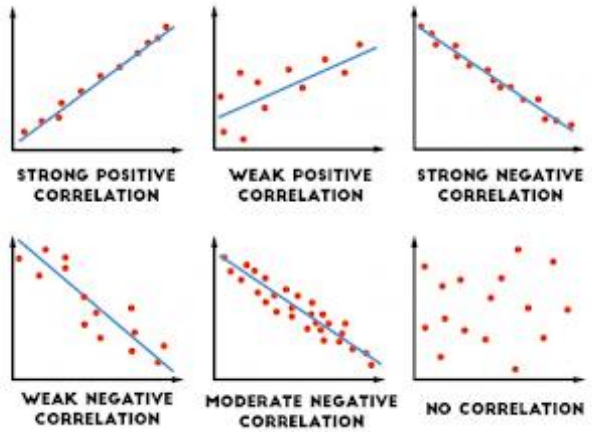


Correlations

Correlations are a measurement of statistical dependence. Correlations have two types: Monotonic and Non-Monotonic. The following image can help you have a better understanding of this types of correlation.



Based on the distribution of the data, the correlation can range from a strong correlation to no correlation, as can be described in the following image:

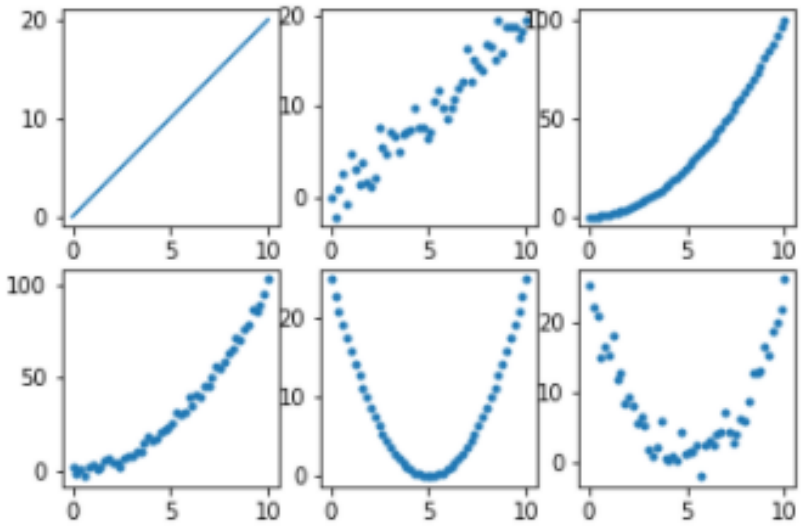


We can easily plot these kinds of distribution in Python. In the following script we follow the next steps:

1. Create a "x" variable that stores a range of values
2. Create a "y1" variable that has a linear relationship with "x"
3. Create a "y2" variable equal to "y1" plus some random numbers that cause dispersion in the data.
4. Create a "y3" variable that has a quadratic relationship with "x"
5. Create a "y4" variable equal to "y2" plus some random numbers that cause dispersion in the data
6. Create a "y5" variable that has a quadratic relationship
7. Create a "y6" variable equal to "y5" plus some random numbers that cause dispersion in the data

After defining the previous variables, plot the data.

```
x = np.linspace(0,10)
y1 = 2*x
y2 = y1 + 2*np.random.randn(50)
y3 = x**2
y4 = y3 + 2*np.random.randn(50)
y5 = (x-5)**2
y6 = y5 + 2*np.random.randn(50)

fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) = plt.subplots(nrows=2,ncols=3)
ax1.plot(x, y1)
ax2.plot(x, y2, '.')
ax3.plot(x, y3, '.')
ax4.plot(x, y4, '.')
ax5.plot(x, y5, '.')
ax6.plot(x, y6, '.')
```



Base on these variables, we can calculate the correlations. We use 3 main correlations:

1. **Pearson's r** is the most commonly used measure of correlation and sometimes called the linear correlation coefficient because r measures the linear association between two variables.
2. **Spearman's ρ** is a nonparametric, rank-based correlation coefficient that depends only on the data ranks and not the observations themselves. Therefore, ρ is resistant to outliers and can be implemented even when some of the data are censored, such as concentrations known only as less than an analytical detection limit.
3. **Kendall tau τ** much like Spearman's ρ, measures the strength of the monotonic relation between x and y and is a rank-based procedure. Just as with ρ, τ is resistant to the effect of outliers and, because τ also depends only on the ranks of the data and not the observations themselves, it can be implemented even in cases where some of the data are categorical, such as censored observations

To calculate the correlations mentioned, we can use Scipy. For instance, if we want to calculate the Pearson correlation for the linear equation, we use the following script that returns Pearson's correlation coefficient and a Two-tailed P-value. The last one evaluates if the mean is significantly greater than x and if the mean is significantly less than x.

```
st.pearsonr(x,y1)
```

```
st.pearsonr(x,y1)

(0.9999999999999998, 0.0)
```

To calculate all the correlation values we can use the following script:

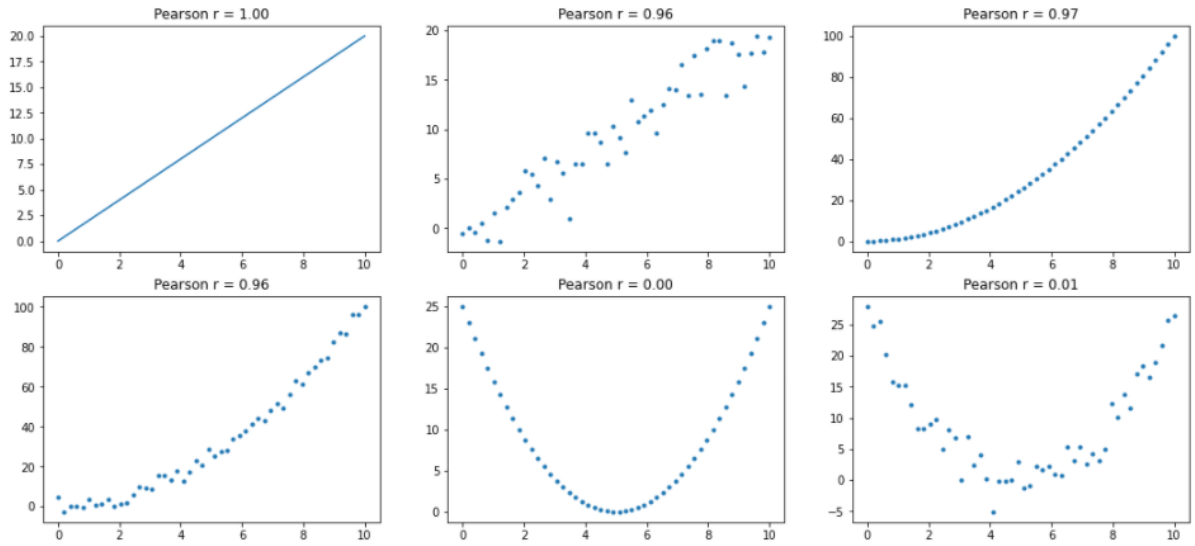
```
r1, p1 = st.pearsonr(x,y1)
r2, p2 = st.pearsonr(x,y2)
r3, p3 = st.pearsonr(x,y3)
r4, p4 = st.pearsonr(x,y4)
r5, p5 = st.pearsonr(x,y5)
r6, p6 = st.pearsonr(x,y6)

# print r's
print(F"{r1:.2f},{r2:.2f},{r3:.2f},{r4:.2f},{r5:.2f},{r6:.2f}")
```

```
r1, p1 = st.pearsonr(x,y1)
r2, p2 = st.pearsonr(x,y2)
r3, p3 = st.pearsonr(x,y3)
r4, p4 = st.pearsonr(x,y4)
r5, p5 = st.pearsonr(x,y5)
r6, p6 = st.pearsonr(x,y6)

# print r's
print(F"{r1:.2f},{r2:.2f},{r3:.2f},{r4:.2f},{r5:.2f},{r6:.2f}")
```

```
1.00,0.97,0.97,0.96,0.00,-0.01
```

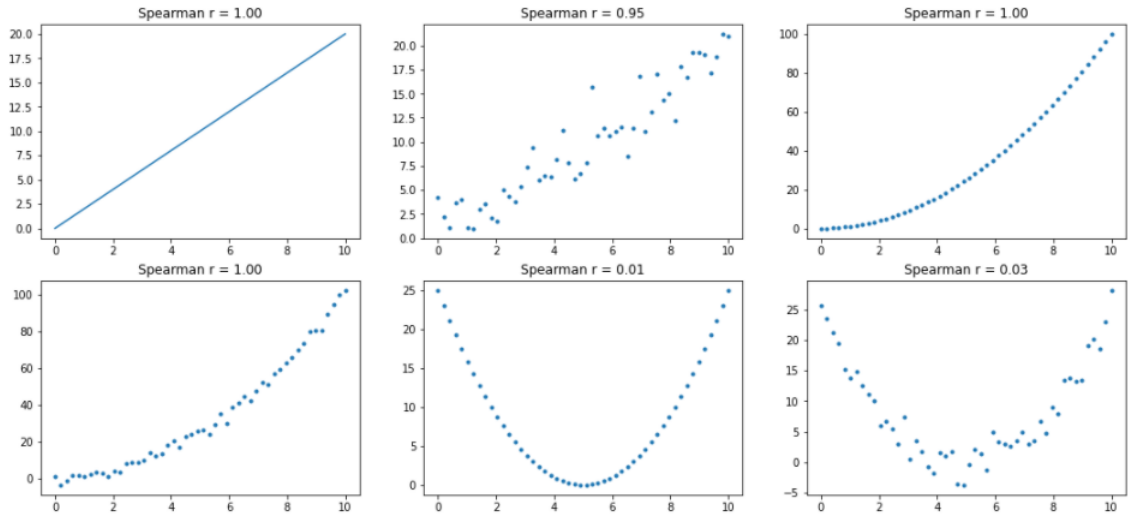
Sometimes having raw data like this is hard to comprehend. Let's couple this result with the previous graph.

```
x = np.linspace(0,10)
y1 = 2*x
y2 = y1 + 2*np.random.randn(50)
y3 = x**2
y4 = y3 + 2*np.random.randn(50)
y5 = (x-5)**2
y6 = y5 + 2*np.random.randn(50)

r1, p1 = st.pearsonr(x,y1)
r2, p2 = st.pearsonr(x,y2)
r3, p3 = st.pearsonr(x,y3)
r4, p4 = st.pearsonr(x,y4)
r5, p5 = st.pearsonr(x,y5)
r6, p6 = st.pearsonr(x,y6)

fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) =
plt.subplots(nrows=2,ncols=3,figsize=(18,8))
ax1.plot(x, y1)
ax1.set_title(F'Pearson r = {r1:.2f}')
ax2.plot(x, y2, '.')
ax2.set_title(F'Pearson r = {r2:.2f}')
ax3.plot(x, y3, '.')
ax3.set_title(F'Pearson r = {r3:.2f}')
ax4.plot(x, y4, '.')
ax4.set_title(F'Pearson r = {r4:.2f}')
ax5.plot(x, y5, '.')
ax5.set_title(F'Pearson r = {r5:.2f}')
ax6.plot(x, y6, '.')
ax6.set_title(F'Pearson r = {r6:.2f}')
```



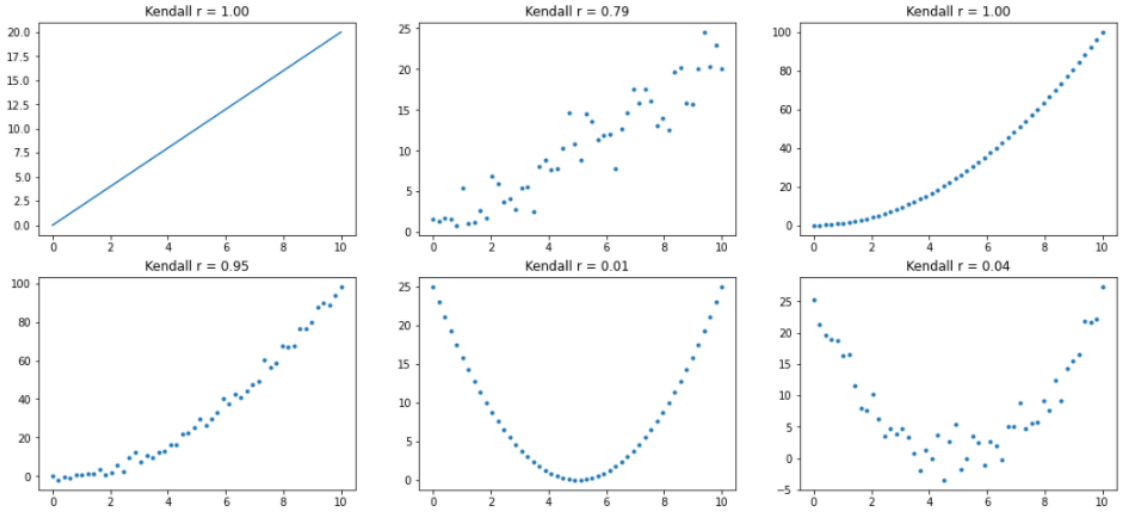
Making the calculations with Spearmen correlation

```
x = np.linspace(0,10)
y1 = 2*x
y2 = y1 + 2*np.random.randn(50)
y3 = x**2
y4 = y3 + 2*np.random.randn(50)
y5 = (x-5)**2
y6 = y5 + 2*np.random.randn(50)

rho1, p1 = st.spearmanr(x,y1)
rho2, p2 = st.spearmanr(x,y2)
rho3, p3 = st.spearmanr(x,y3)
rho4, p4 = st.spearmanr(x,y4)
rho5, p5 = st.spearmanr(x,y5)
rho6, p6 = st.spearmanr(x,y6)

fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) =
plt.subplots(nrows=2,ncols=3,figsize=(18,8))
ax1.plot(x, y1)
ax1.set_title(F'Spearman r = {rho1:.2f}')
ax2.plot(x, y2, '.')
ax2.set_title(F'Spearman r = {rho2:.2f}')
ax3.plot(x, y3, '.')
ax3.set_title(F'Spearman r = {rho3:.2f}')
ax4.plot(x, y4, '.')
ax4.set_title(F'Spearman r = {rho4:.2f}')
ax5.plot(x, y5, '.')
ax5.set_title(F'Spearman r = {rho5:.2f}')
ax6.plot(x, y6, '.')
ax6.set_title(F'Spearman r = {rho6:.2f}')
```



And calculating the correlations with Kendall correlation.

```
x = np.linspace(0,10)
y1 = 2*x
y2 = y1 + 2*np.random.randn(50)
y3 = x**2
y4 = y3 + 2*np.random.randn(50)
y5 = (x-5)**2
y6 = y5 + 2*np.random.randn(50)

tau1, p1 = st.kendalltau(x,y1)
tau2, p2 = st.kendalltau(x,y2)
tau3, p3 = st.kendalltau(x,y3)
tau4, p4 = st.kendalltau(x,y4)
tau5, p5 = st.kendalltau(x,y5)
tau6, p6 = st.kendalltau(x,y6)

fig, ((ax1,ax2,ax3),(ax4,ax5,ax6)) =
plt.subplots(nrows=2,ncols=3,figsize=(18,8))
ax1.plot(x, y1)
ax1.set_title(F'Kendall r = {tau1:.2f}')
ax2.plot(x, y2, '.')
ax2.set_title(F'Kendall r = {tau2:.2f}')
ax3.plot(x, y3, '.')
ax3.set_title(F'Kendall r = {tau3:.2f}')
ax4.plot(x, y4, '.')
ax4.set_title(F'Kendall r = {tau4:.2f}')
ax5.plot(x, y5, '.')
ax5.set_title(F'Kendall r = {tau5:.2f}')
ax6.plot(x, y6, '.')
ax6.set_title(F'Kendall r = {tau6:.2f}')
```



Can we make correlations of precipitation with runoff? It is not recommended because these two variables depend on multiple external factors.

Read the precipitation and streamflow values

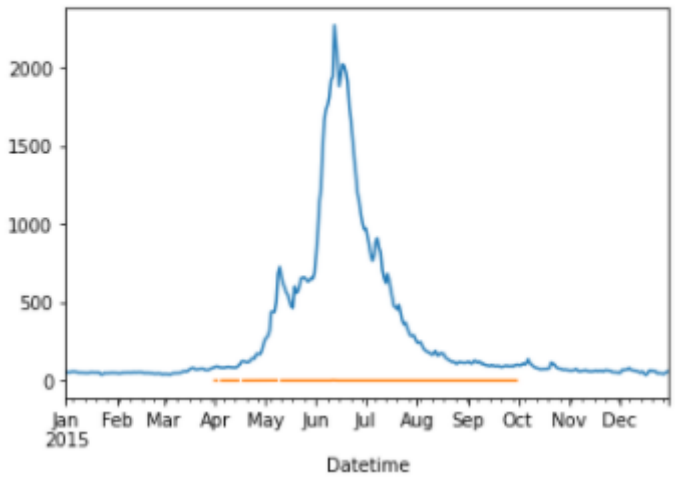
```
pp = pd.read_csv('pp.csv',index_col='Datetime',parse_dates=True)
sf = pd.read_csv('sf.csv',index_col='Date',parse_dates=True)
```

If you plot both of them, you could not see a relationship, mainly because precipitation is in inches and flow in cubic feet per second.

```
sf['Flow_cfs'].loc['2015'].plot()
(pp['Precipitation_in'].loc['2015']).plot()
```

```
sf['Flow_cfs'].loc['2015'].plot()
(pp['Precipitation_in'].loc['2015']).plot()
```

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

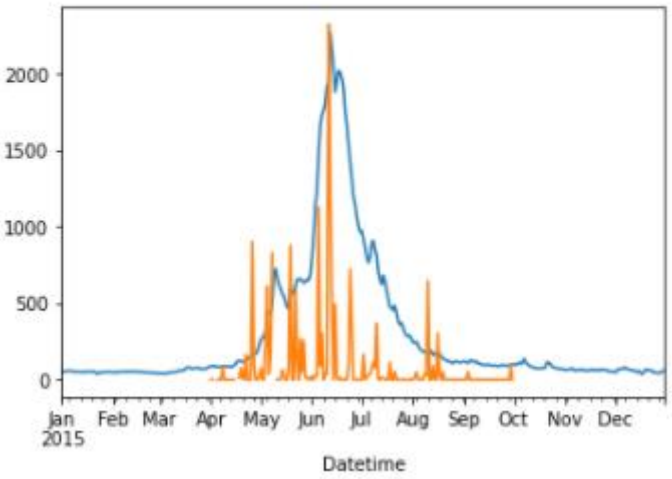


Multiply by a factor the precipitation values, you now have a better visualization, but you can not see a relationship.

```
sf['Flow_cfs'].loc['2015'].plot()
(pp['Precipitation_in'].loc['2015']*25.4*50).plot()
```

```
sf['Flow_cfs'].loc['2015'].plot()
(pp['Precipitation_in'].loc['2015']*25.4*50).plot()
```

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

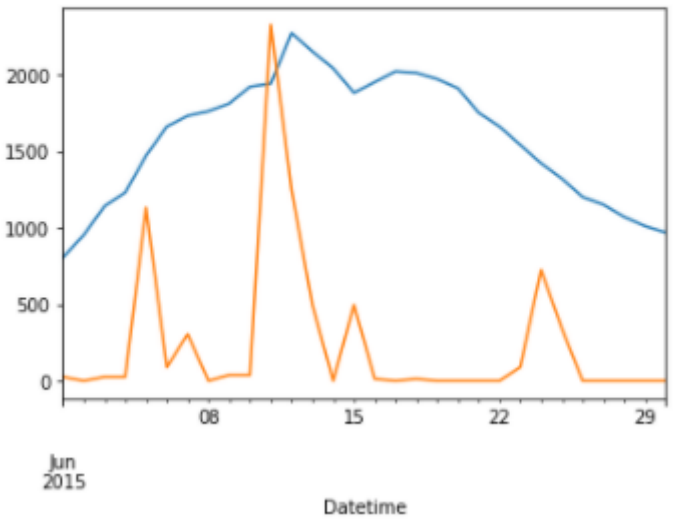


If you reduce the range of the data to monthly values, you could not see a relation.

```
sf['Flow_cfs'].loc['2015-06'].plot()
(pp['Precipitation_in'].loc['2015-06']*25.4*50).plot()
```

```
sf['Flow_cfs'].loc['2015-06'].plot()
(pp['Precipitation_in'].loc['2015-06']*25.4*50).plot()
```

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

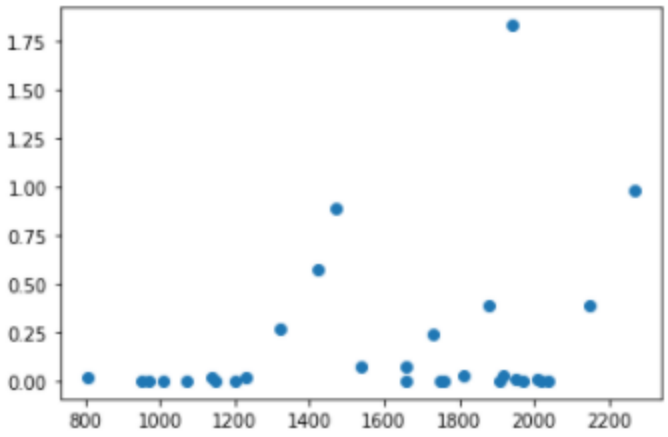


A scatter plot of them doesn't show a trend too.

```
plt.scatter(sf['Flow_cfs'].loc['2015-
06'],pp['Precipitation_in'].loc['2015-06'])
```

```
plt.scatter(sf['Flow_cfs'].loc['2015-06'],pp['Precipitation_in'].loc['2015-06'])
```

```
<matplotlib.collections.PathCollection at 0x2ae2d92e700>
```



If you calculate the correlation coefficients you would get low values too.

```
rpearson, ppearson = st.pearsonr(sf['Flow_cfs'].loc['2015-
06'],pp['Precipitation_in'].loc['2015-06'])
rspearman, pspearman = st.spearmanr(sf['Flow_cfs'].loc['2015-
06'],pp['Precipitation_in'].loc['2015-06'])
rkendalltau, pkendalltau = st.kendalltau(sf['Flow_cfs'].loc['2015-
06'],pp['Precipitation_in'].loc['2015-06'])

rpearson,rspearman,rkendalltau
```

```
rpearson, ppearson = st.pearsonr(sf['Flow_cfs'].loc['2015-06'],pp['Precipitation_in'].loc['2015-06'])
rspearman, pspearman = st.spearmanr(sf['Flow_cfs'].loc['2015-06'],pp['Precipitation_in'].loc['2015-06'])
rkendalltau, pkendalltau = st.kendalltau(sf['Flow_cfs'].loc['2015-06'],pp['Precipitation_in'].loc['2015-06'])

rpearson,rspearman,rkendalltau
```

```
(0.2858896410942486, 0.22639073111895844, 0.16677635188200526)
```

A better approach to see how we can apply a relationship is with reservoirs. Most reservoirs have equations that relate the reservoir's elevation with the volume, so making a relationship between these two variables should return us a high correlation.

Let's import the 'reservoir.csv' file.

```
pd.read_csv('reservoir.csv').head()
```

```
pd.read_csv('reservoir.csv').head()
```

	Elevation of the reservoir (masl)	Volume in (Hm3)	Unnamed: 2
0	4535.28	58.23	NaN
1	4535.28	58.25	NaN
2	4535.29	58.30	NaN
3	4535.29	58.30	NaN
4	4535.29	58.28	NaN

There is an extra column in the DataFrame. We can get rid of it and store the DataFrame in a variable as follows:

```
res = pd.read_csv('reservoir.csv')[['Elevation of the reservoir
(masl)','Volume in (Hm3)']]
res.head()
```

```
res = pd.read_csv('reservoir.csv')[['Elevation of the reservoir (masl)','Volume in (Hm3)']]
res.head()
```

	Elevation of the reservoir (masl)	Volume in (Hm3)
0	4535.28	58.23
1	4535.28	58.25
2	4535.29	58.30
3	4535.29	58.30
4	4535.29	58.28

If you make a scatter plot of them, you could see a strong correlation. It could have a better fit if we add all the decimals used in the calculated values.

```
plt.scatter(res['Elevation of the reservoir (masl)'],res['Volume in
(Hm3)'])
```

Calculating the correlation coefficients, we get values ranging from 0.88 to 0.99, which supports our hypothesis related to the height vs volume in reservoir.

```
rpearson, ppearson = st.pearsonr(res['Elevation of the reservoir
(masl)'],res['Volume in (Hm3)'])
rspearman, pspearman = st.spearmanr(res['Elevation of the reservoir
(masl)'],res['Volume in (Hm3)'])
rkendalltau, pkendalltau = st.kendalltau(res['Elevation of the
reservoir (masl)'],res['Volume in (Hm3)'])

rpearson,rspearman,rkendalltau
```

```
rpearson, ppearson = st.pearsonr(res['Elevation of the reservoir (masl)'],res['Volume in (Hm3)'])
rspearman, pspearman = st.spearmanr(res['Elevation of the reservoir (masl)'],res['Volume in (Hm3)'])
rkendalltau, pkendalltau = st.kendalltau(res['Elevation of the reservoir (masl)'],res['Volume in (Hm3)'])

rpearson,rspearman,rkendalltau
```

```
(0.9870988872923259, 0.9371970331428452, 0.8797232766491183)
```

Uncertainty intervals

Uncertainty intervals are useful when you have data dispersed. Most of the times, when you develop a model, you get different measurements, and you could see the distribution of your data as "noise"

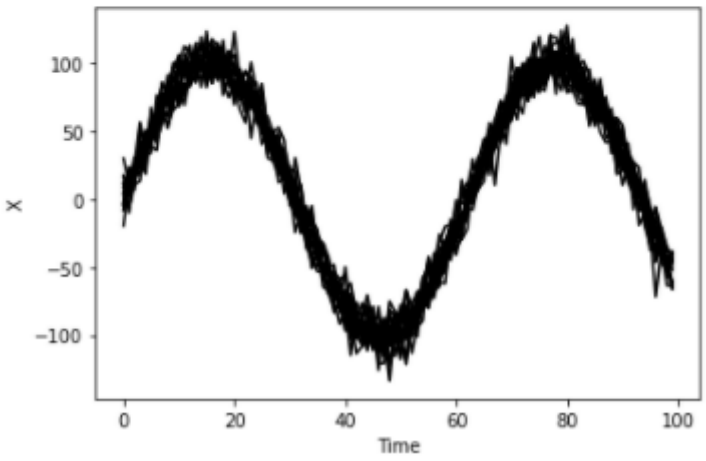
For example, plot the following information that contains"

- "x" as an array of ordered values
- "X" which is the compilation of the "x" value. If you see its shape, it returns (20,100)
- "e" is the error generated to the plot or, in other words, the "noise"
- "X_err" is the merge of the "X" array and the noise added.

```
# generate some data
x = 100*np.sin(np.linspace(0,10,100))
X = np.vstack([x]*20)
e = 10*np.random.randn(20,100)

X_err = X+e

plt.plot(X_err.T, 'k')
plt.xlabel('Time')
plt.ylabel('X')
```

```
# generate some data
x = 100*np.sin(np.linspace(0,10,100))
X = np.vstack([x]*20)
e = 10*np.random.randn(20,100)

X_err = X+e

plt.plot(X_err.T, 'k')
plt.xlabel('Time')
plt.ylabel('X')
```

```
Text(0, 0.5, 'X')
```



As you could see in the previous graph, there is a lot of "noise" In the plot. For the uncertainty, we are going to calculate the percentiles 10 and 90. Scipy allows us to do this quickly as follows:

```
st.scoreatpercentile(X_err, 10)
```

```
st.scoreatpercentile(X_err, 10)
```

```
-86.89087353862217
```

You are only getting one value. If you want to calculate the percentiles for each column of data, you need to slice and iterate over them.

You can use the following script that would return data for each of the 100 columns:

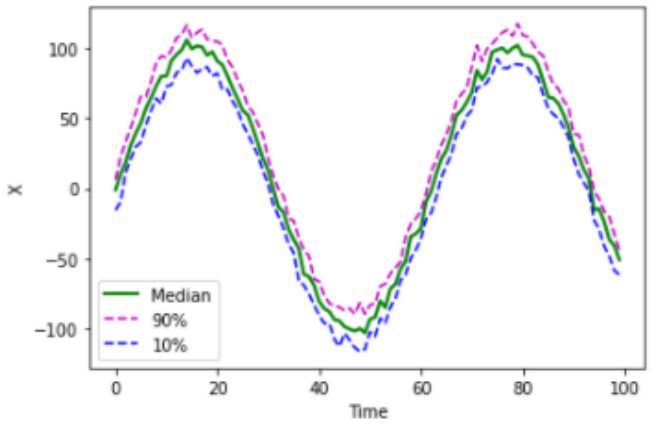
```
[st.scoreatpercentile(X_err[:,i], 10) for i in range(X_err.shape[1])]
```

We can store them in variables and plot the percentiles 10, 50 and 90.

```
ll = [st.scoreatpercentile(X_err[:,i], 10) for i in
range(X_err.shape[1])] # 10th percentile
ml = [st.scoreatpercentile(X_err[:,i], 50) for i in
range(X_err.shape[1])] # 50th percentile
ul = [st.scoreatpercentile(X_err[:,i], 90) for i in
range(X_err.shape[1])] # 90th percentile

plt.plot(ml, 'g', lw=2, label='Median')
plt.plot(ul, '--m', label='90%')
plt.plot(ll, '--b', label='10%')
plt.xlabel('Time')
plt.ylabel('X')
plt.legend(loc='best')
```

```
ll = [st.scoreatpercentile(X_err[:,i], 10) for i in range(X_err.shape[1])] # 10th percentile
ml = [st.scoreatpercentile(X_err[:,i], 50) for i in range(X_err.shape[1])] # 50th percentile
ul = [st.scoreatpercentile(X_err[:,i], 90) for i in range(X_err.shape[1])] # 90th percentile

plt.plot(ml, 'g', lw=2, label='Median')
plt.plot(ul, '--m', label='90%')
plt.plot(ll, '--b', label='10%')
plt.xlabel('Time')
plt.ylabel('X')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x2ae29dedfd0>
```



If you don't like this visualization, you can choose to develop a plot of the area between the uncertainty interavals.

```
plt.plot(ml, 'g', lw=2, label='Median')
```

```
plt.fill_between(range(100), ul, ll, color='k', alpha=0.4,
label='90%')
plt.xlabel('Time')
plt.ylabel('X')
plt.legend(loc='best')
```

```
plt.plot(ml, 'g', lw=2, label='Median')
plt.fill_between(range(100), ul, ll, color='k', alpha=0.4, label='90%')
plt.xlabel('Time')
plt.ylabel('X')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x2ae29a61e20>
```