Statistics with Python

Morteza Mohammadi

Department of Statistics

University of Zabol

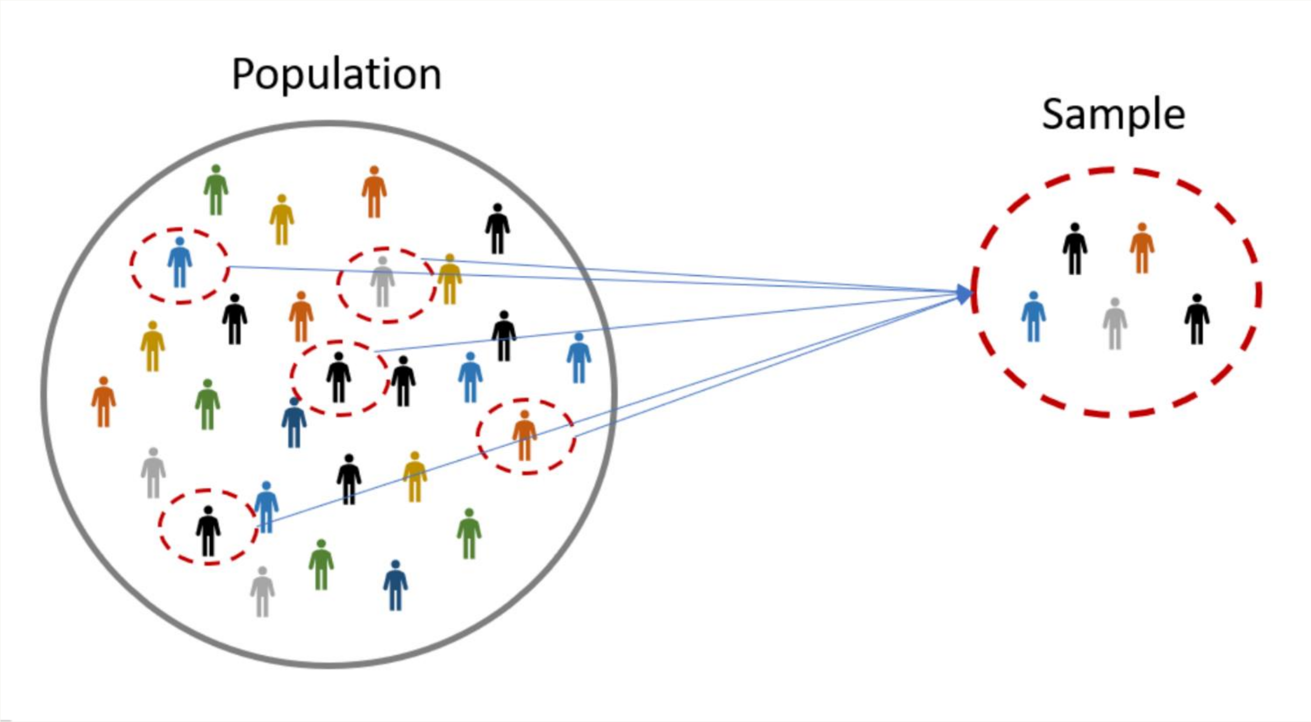
Python Statistics Fundamentals: How to Describe Your Data – Real Python

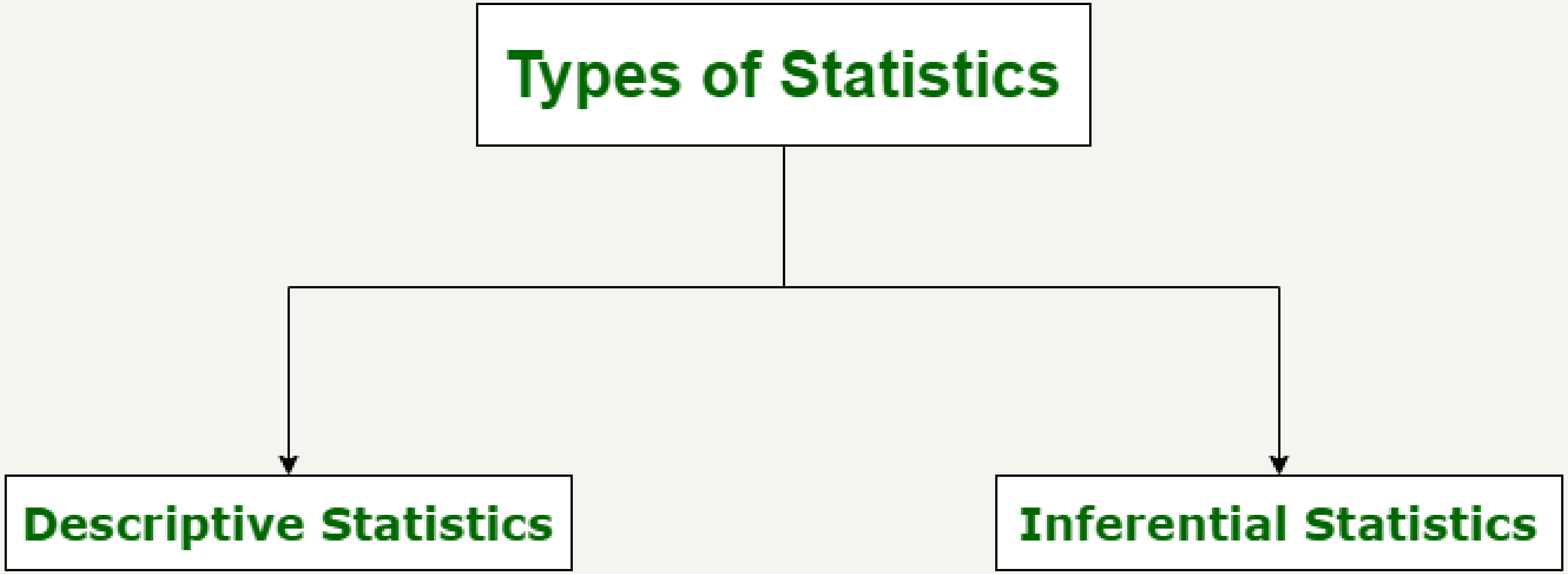
Statistics

- The science of collecting, analyzing, presenting, and interpreting data.

Population and Samples

- In statistics, the population is a set of all elements or items that you're interested in. This subset of a population is called a sample. Ideally, the sample should preserve the essential statistical features of the population to a satisfactory extent. That way, you'll be able to use the sample to glean conclusions about the population.




Types of Statistics
Descriptive Statistics
Inferential Statistics

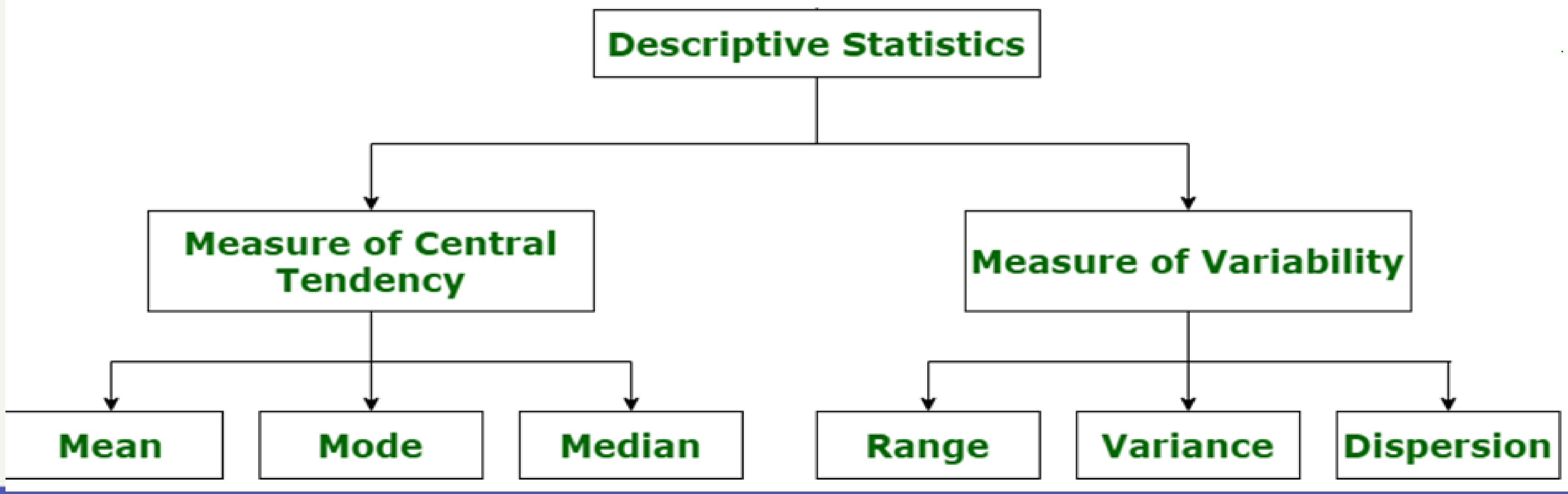
Descriptive statistics

Descriptive statistics is about describing and summarizing data. It uses two main approaches:

- **The quantitative approach** describes and summarizes data numerically.
- **The visual approach** illustrates data with charts, plots, histograms, and other graphs.

The quantitative approach

1. **Central tendency** tells you about the centers of the data. Useful measures include the mean, median, and mode.
2. **Variability** tells you about the spread of the data. Useful measures include range, variance, and standard deviation.

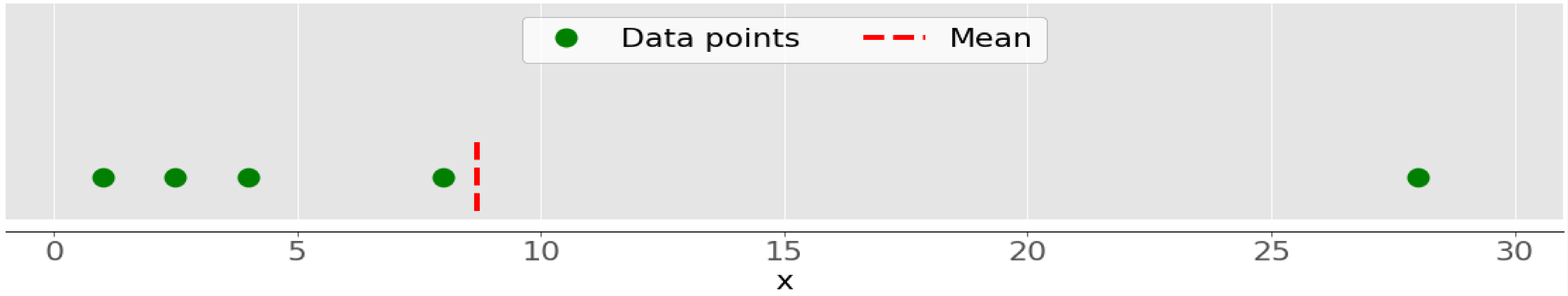


Mean

The ***mean*** is the sum of the observations divided by the number of them. For a variable y with n observations y_1, y_2, ..., y_n in a sample from some population, the mean $\bar{y}$ is

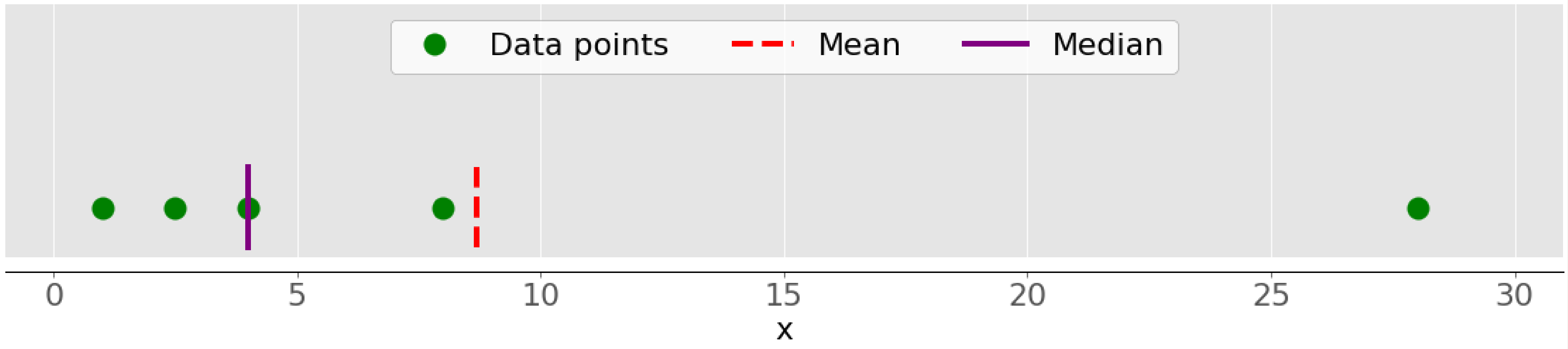
$$\bar{y} = \frac{y_1 + y_2 + \cdots + y_n}{n} = \frac{\sum_{i=1}^{n} y_i}{n}.$$

The green dots represent the data points 1, 2.5, 4, 8, and 28. The red dashed line is their mean, or (1 + 2.5 + 4 + 8 + 28) / 5 = 8.7.

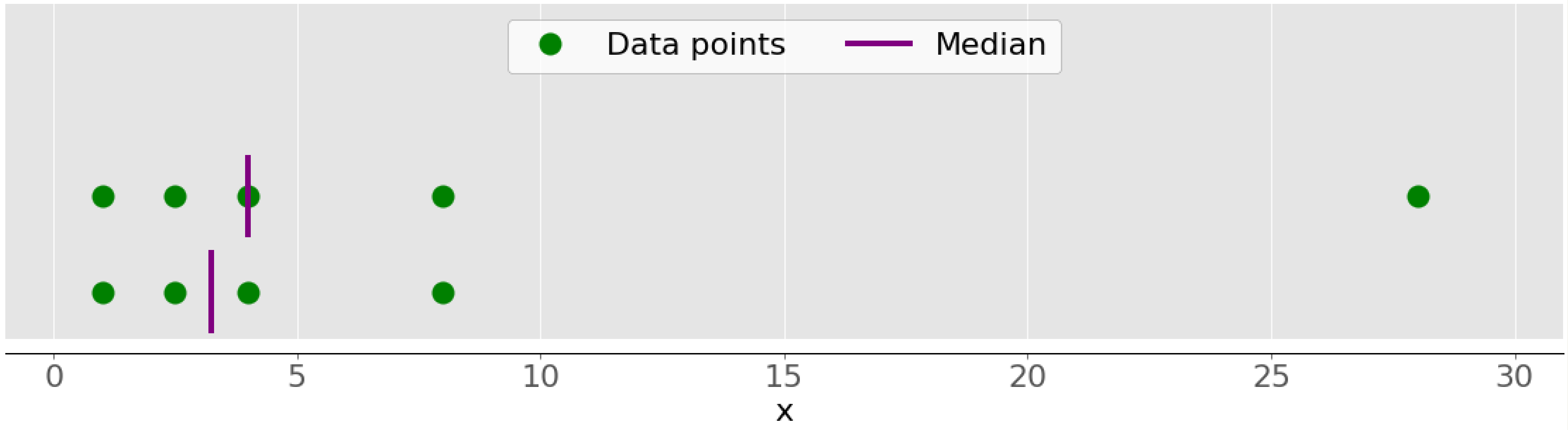


Median

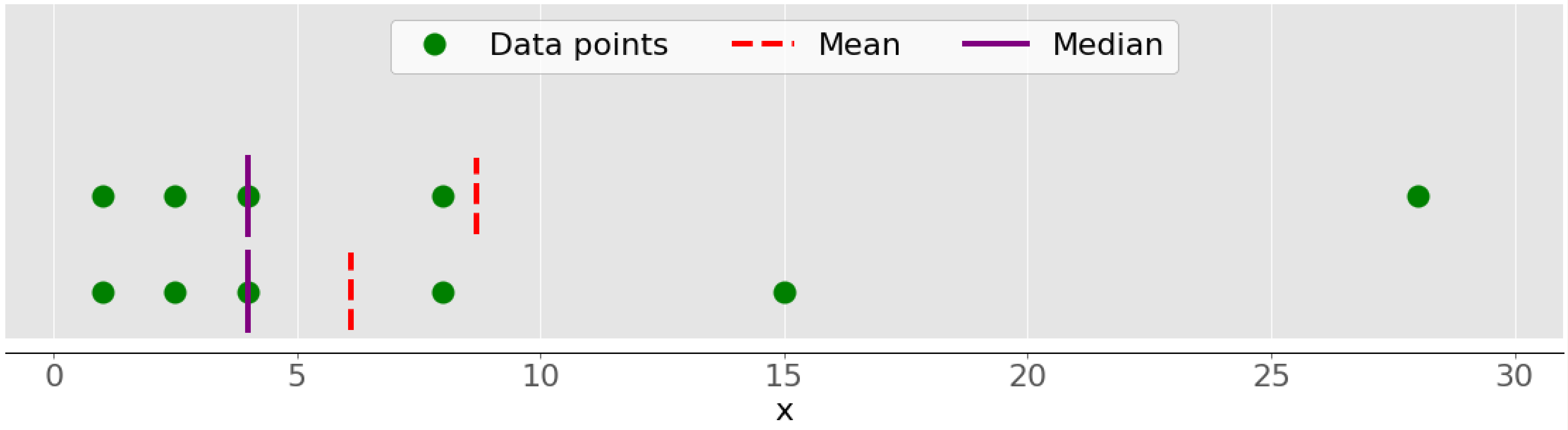
- The **sample median** is the middle element of a sorted dataset.



The dataset can be sorted in increasing or decreasing order. If the number of elements n of the dataset is odd, then the median is the value at the middle position: $0.5(n + 1)$. If n is even, then the median is the arithmetic mean of the two values in the middle, that is, the items at the positions $0.5n$ and $0.5n + 1$.



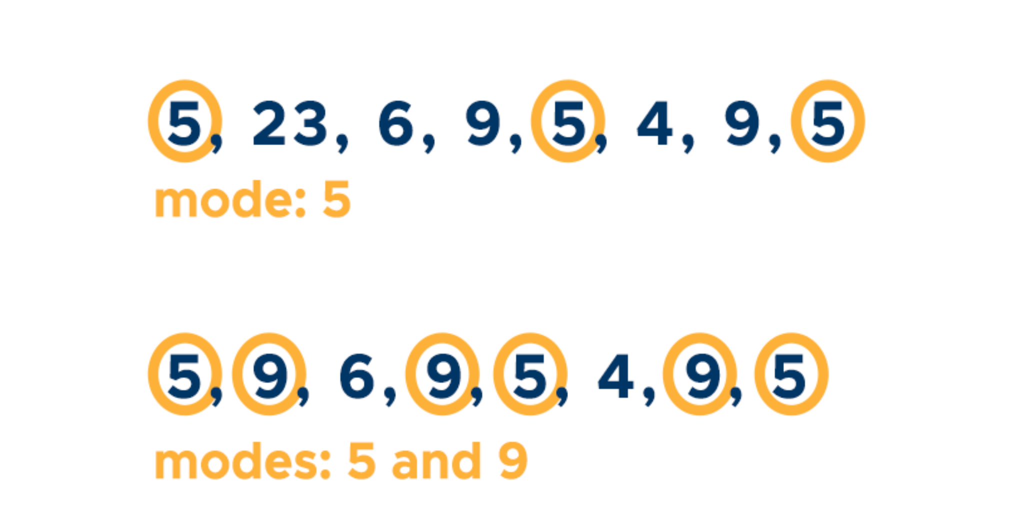
- The mean is heavily affected by outliers, but the median only depends on outliers either slightly or not at all.



Mode

- The **sample mode** is the value in the dataset that occurs most frequently.

Example:



Choosing Python Statistics Libraries

- **statistics** is a built-in Python library for descriptive statistics.
- **NumPy** is a third-party library for numerical computing, optimized for working with single- and multi-dimensional arrays. This library contains many routines for statistical analysis.
- **SciPy** is a third-party library for scientific computing based on NumPy. It offers additional functionality compared to NumPy, including **scipy.stats** for statistical analysis.
- **pandas** is a third-party library for numerical computing based on NumPy.
- **Matplotlib** is a third-party library for data visualization.

Calculating Descriptive Statistics

```
In [1]: import math
        import statistics
        import numpy as np
        import scipy.stats
        import pandas as pd
```

```
In [9]: x = [8.0, 1, 2.5, 4, 28.0]
        sum(x) / len(x)

Out[9]: 8.7
```

```
In [10]: statistics.mean(x)

Out[10]: 8.7
```

Descriptive Statistics using statistics Library

```
>>> import statistics
>>> x = [8.0, 1, 2.5, 4, 28.0]
>>> x
[8.0, 1, 2.5, 4, 28.0]
>>> sum(x) / len(x)
8.7
>>> statistics.mean(x)
8.7
```

```
>>> statistics.median(x)
4
>>> statistics.median(x[:-1])
3.25
```

x[:-1] is [1, 2.5, 4, 8.0]

```
>>> statistics.mode(x)
8
>>> statistics.multimode(x)
[8.0, 1, 2.5, 4, 28.0]
```

```
>>> statistics.multimode([12, 15, 21, 15, 12])
12, 15
```

Descriptive Statistics using numpy Library

```
>>> import numpy as np
>>> x = [8.0, 1, 2.5, 4, 28.0]
>>>y= np.array(x)
>>> y
array([ 8. ,  1. ,  2.5, 4. , 28. ])
>>> np.mean(y)
8.7
```

```
>>> np.median(y)
4.0
```

```
>>> import scipy.stats
#from scipy import stats
>>> stats.mode([12, 15,  21, 12])
ModeResult(mode=array([12]), count=array([2]))
```

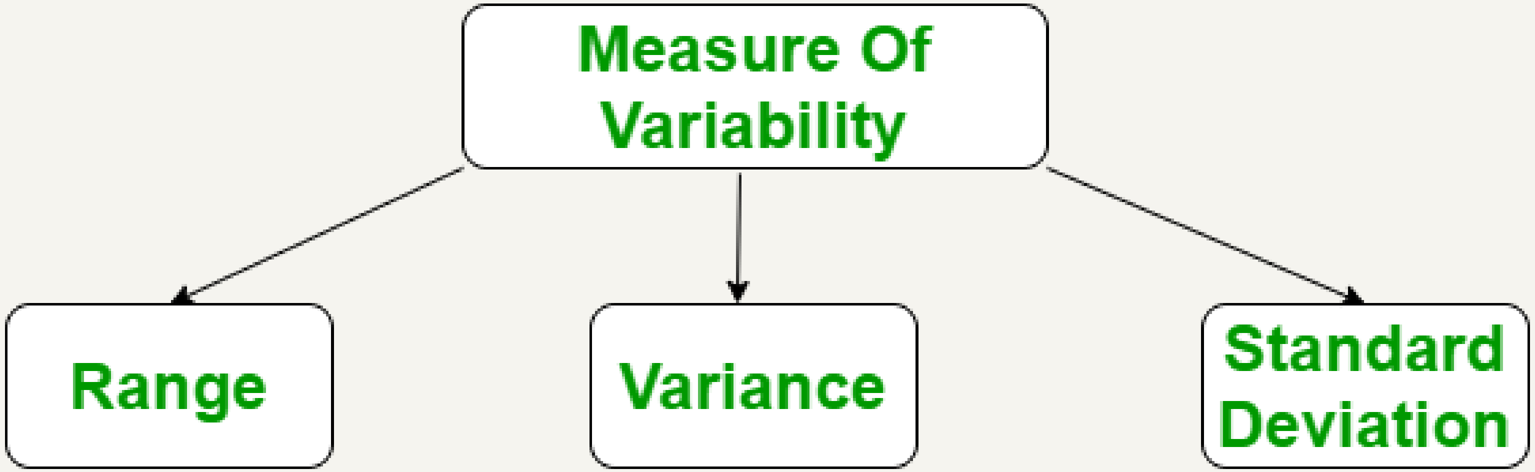
Descriptive Statistics using pandas Library

```
>>> import pandas as pd
>>> x = [8.0, 1, 2.5, 4, 28.0]
>>> z= pd.Series(x)
>>> z
0     8.0
1     1.0
2     2.5
3     4.0
4    28.0
dtype: float64
>>> mean_ = z.mean()
>>> mean_
8.7
```

```
>>> z.median()
4.0
```

Measures of Variability

- The measures of central tendency aren’t sufficient to describe data. You’ll also need the **measures of variability** that quantify the spread of data points.



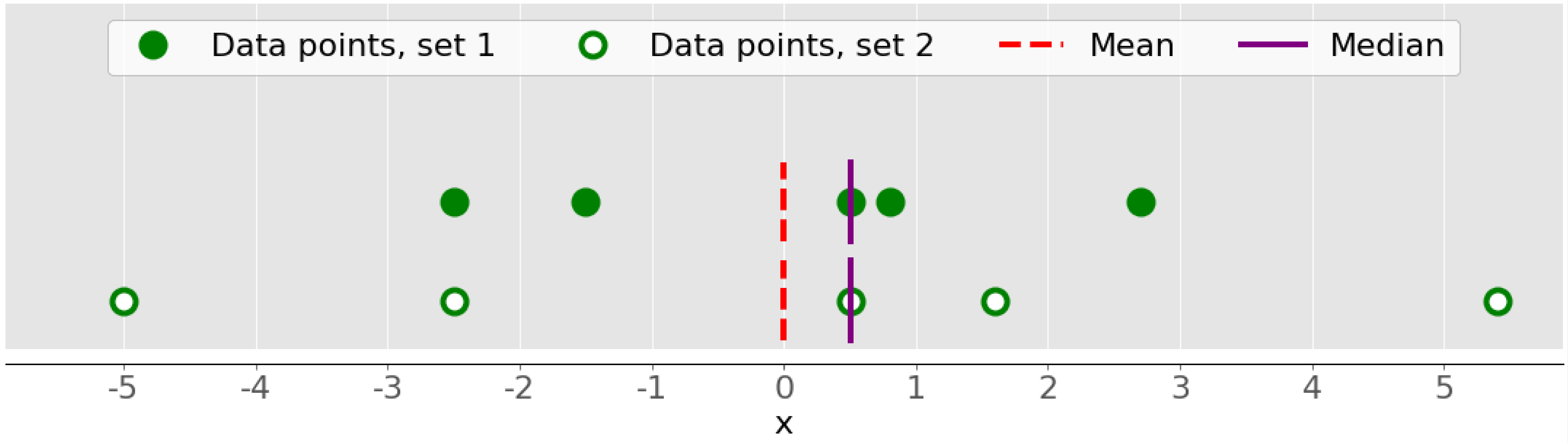
Standard deviation and variance

For a variable y with n observations y_1, y_2, ..., y_n in a sample from some population, the ***standard deviation*** s is

$$s = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1}} = \sqrt{\frac{(y_1 - \bar{y})^2 + (y_2 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2}{n-1}}.$$

The standard deviation is the positive square root of the ***variance*** s^2,

$$s^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1}.$$


Data points, set 1
Data points, set 2
Mean
Median
-5
-4
-3
-2
-1
0
1
2
3
4
5
x

Example: $3, 4, 6, 7, 7, 9, 13$ $n = 7$

x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
3	-4	16
4	-3	9
6	-1	1
7	0	0
7	0	0
9	2	4
13	6	36
	Total:	66

$$\bar{x} = \frac{3 + 4 + 6 + 7 + 7 + 9 + 13}{7}$$

$$\bar{x} = 7$$

$$66 \div 6 = 11$$

The variance = 11

```
>>> var_ = statistics.variance(x)
>>> var_
123.2
>>> var_ = np.var(y, ddof=1)
>>> var_
123.19999999999999
>>> var_ = y.var(ddof=1)
>>> var_
123.19999999999999
>>> z.var(ddof=1)
123.19999999999999
```

It's very important to specify the parameter `ddof=1`. That's how you set the delta degrees of freedom to `1`. This parameter allows the proper calculation of s^2, with $(n - 1)$ in the denominator instead of n.

Standard Deviation

- The **sample standard deviation** is another measure of data spread. It's connected to the sample variance, as standard deviation, s, is the positive square root of the sample variance.
- The standard deviation is often more convenient than the variance because it has the same unit as the data points.

```
>>> std_ = var_ ** 0.5
>>> std_
11.099549540409285
>>> std_ = statistics.stdev(x)
>>> std_
11.099549540409287
>>> np.std(y, ddof=1)
11.099549540409285
>>> y.std(ddof=1)
11.099549540409285
>>> z.std(ddof=1)
11.099549540409285
```

Ranges

- The **range of data** is the difference between the maximum and minimum element in the dataset.

```
>>> x = [8.0, 1, 2.5, 4, 28.0]
>>>y= np.array(x)
>>> np.ptp(y)
27.0
>>> z= pd.Series(x)
>>> np.ptp(z)
27.0
```

ptp: peak-to-peak

Alternatively, you can use built-in Python, NumPy, or pandas functions and methods to calculate the maxima and minima of sequences:

✓ .max() and .min() from NumPy

✓ .max() and .min() from pandas

```
>>> y.max() - y.min()
27.0
>>> z.max() - z.min()
27.0
```

Percentiles

- The **sample p percentile** is the element in the dataset such that p% of the elements in the dataset are less than or equal to that value. Also, $(100 - p)$% of the elements are greater than or equal to that value.
- Each dataset has three **quartiles**, which are the percentiles that divide the dataset into four parts:
- **The first quartile** is the sample 25th percentile. It divides roughly 25% of the smallest items from the rest of the dataset.
- **The second quartile** is the sample 50th percentile or the **median**. Approximately 25% of the items lie between the first and second quartiles and another 25% between the second and third quartiles.
- **The third quartile** is the sample 75th percentile. It divides roughly 25% of the largest items from the rest of the dataset.

```
>>> x = [-5.0, -1.1, 0.1, 2.0, 8.0, 12.8, 21.0, 25.8, 41.0]
>>> statistics.quantiles(x, n=2)
[8.0]
>>> statistics.quantiles(x, n=4)
[0.1, 8.0, 21.0]
```

```
>>> y = np.array(x)
>>> np.percentile(y, 5)
-3.44
>>> np.percentile(y, 95)
34.919999999999995
```

```
>>> np.percentile(y, [25, 50, 75])
array([ 0.1,  8. , 21. ])
>>> np.median(y)
8.0
```

```
>>> z= pd.Series(y)
>>> z.quantile(0.05)
-3.44
>>> z.quantile(0.95)
34.919999999999995
>>> z.quantile([0.25, 0.5, 0.75])
0.25     0.1
0.50     8.0
0.75    21.0
dtype: float64
```

Interquartile range

- The **interquartile range** is the difference between the first and third quartile.

```
>>> quartiles = np.quantile(y, [0.25, 0.75])
>>> quartiles[1] - quartiles[0]
20.9
>>> quartiles = z.quantile([0.25, 0.75])
>>> quartiles[0.75] - quartiles[0.25]
20.9
```

Visualizing Data

In this section, you'll learn how to present your data visually using the following graphs:

- Bar charts
- Pie charts
- Histograms
- Box plots
- X-Y plots

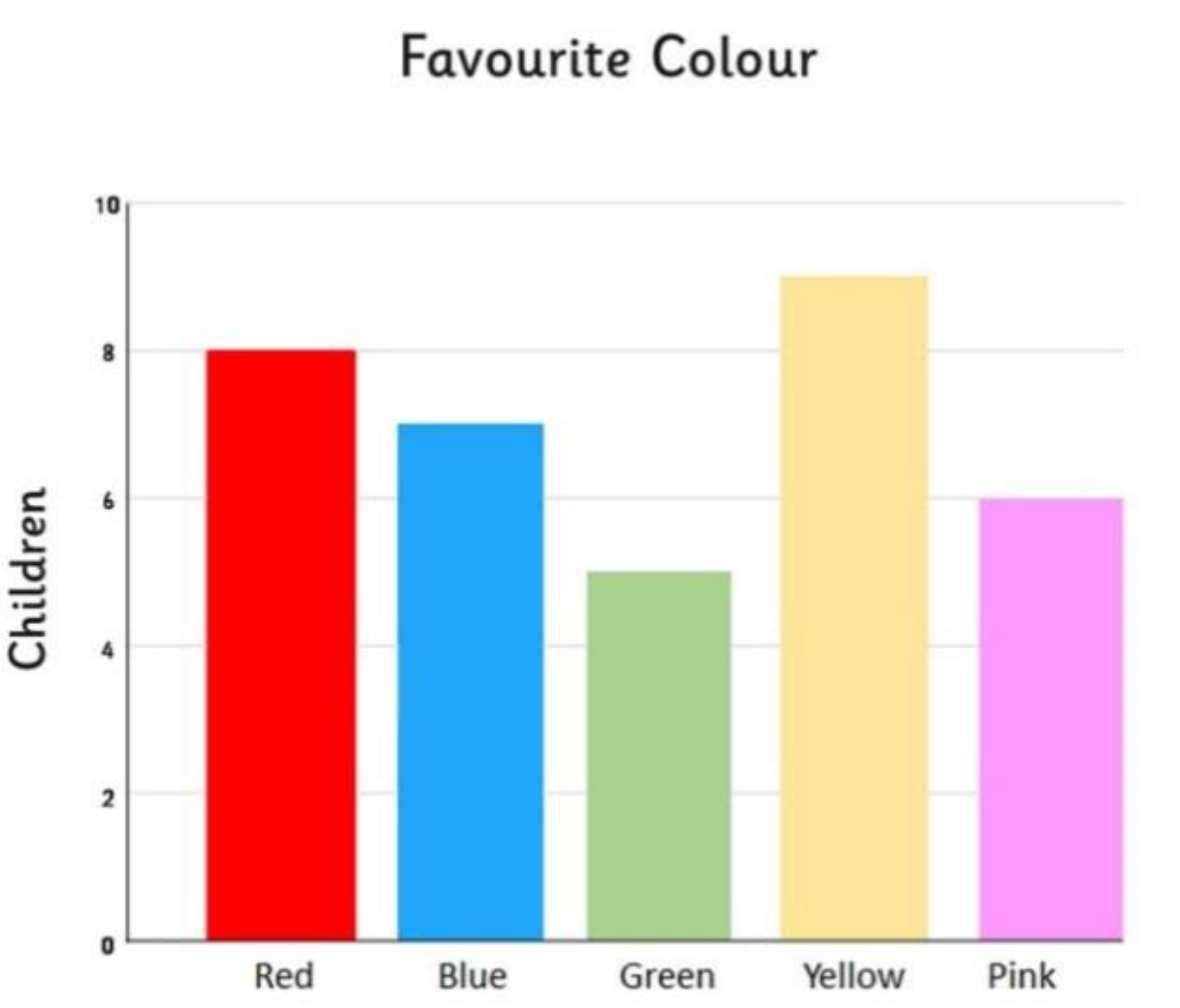
Python library

- matplotlib.pyplot is a very convenient and widely-used library, though it's not the only Python library available for this purpose. You can import it like this:

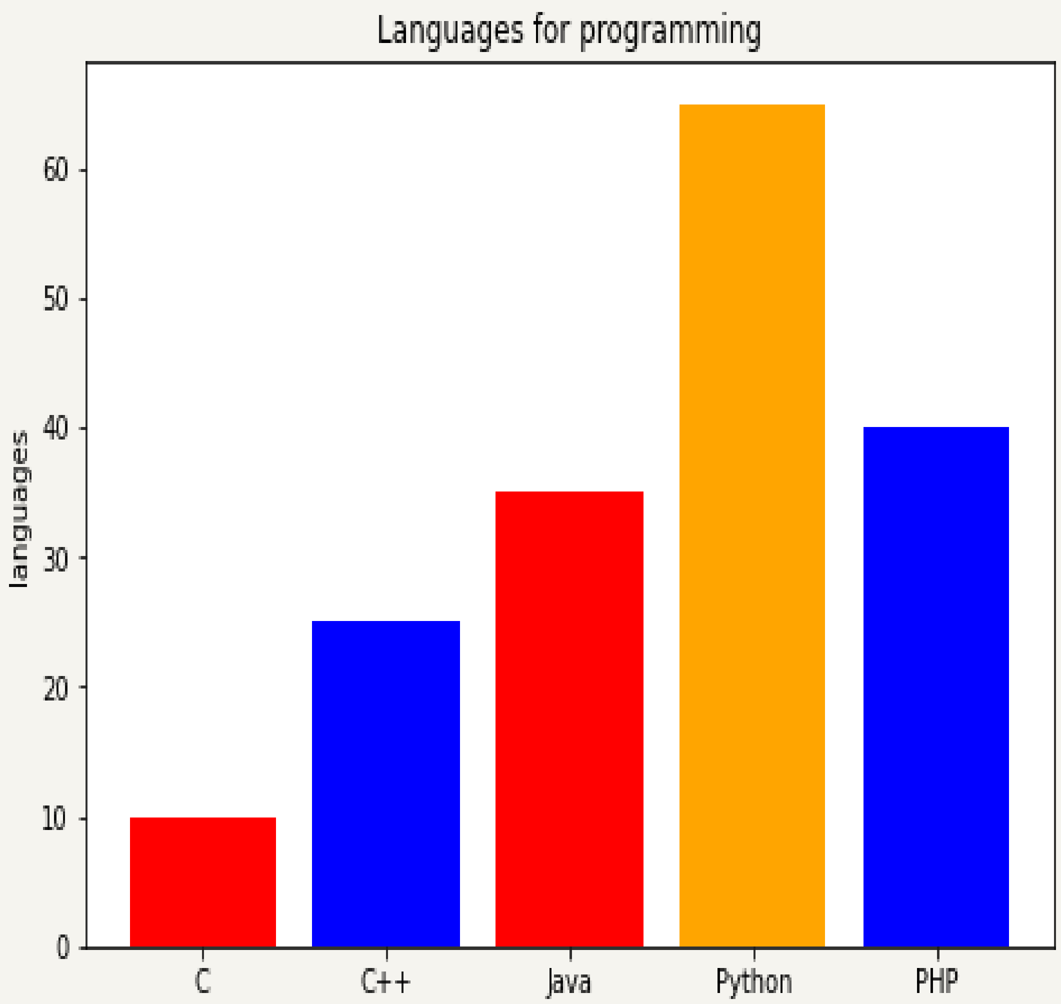
```
>>> import matplotlib.pyplot as plt
```

Bar Charts

- **Bar charts** also illustrate data that correspond to given labels or discrete numeric values.
- The bar chart shows parallel rectangles called **bars**.



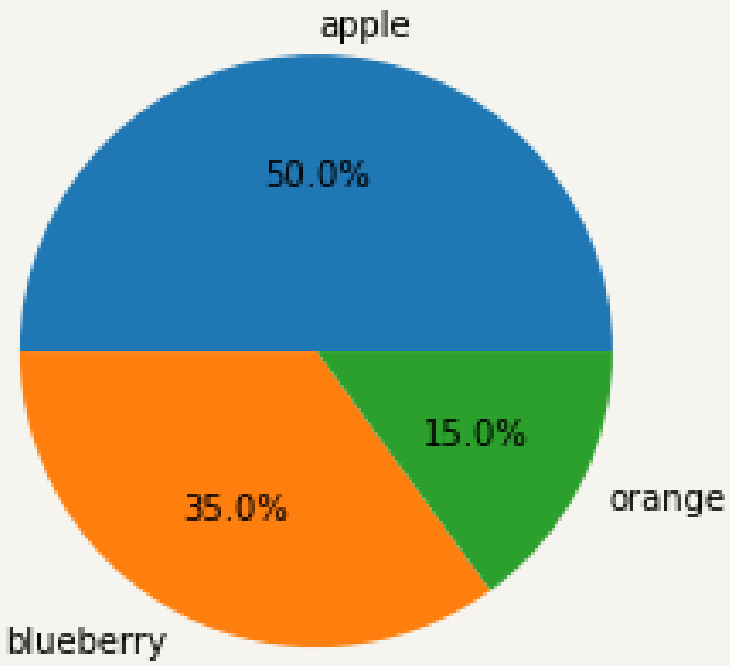
```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_axes([0, 0, 1, 1])
>>> langs = ['C', 'C++', 'Java', 'Python', 'PHP']
>>> students = [10,25,35,65,40]
>>> bar_colors = ['red', 'blue', 'red', 'orange','blue']
>>> ax.bar(langs,students, color=bar_colors)
>>> ax.set_ylabel('languages')
>>> ax.set_title('Languages for programming')
>>> plt.show()
```



Pie Charts

- **Pie charts** represent data with a small number of labels and given relative frequencies. They work well even with the labels that can't be ordered (like nominal data).
- A pie chart is a circle divided into multiple slices. Each slice corresponds to a single distinct label from the dataset and has an area proportional to the relative frequency associated with that label.

```
import matplotlib.pyplot as plt
labels = 'apple', 'blueberry', 'orange'
sizes = [100, 70, 30]


fig, ax = plt.subplots()
ax.pie(sizes, labels=labels,autopct='%0.1f%%')
plt.show()
```



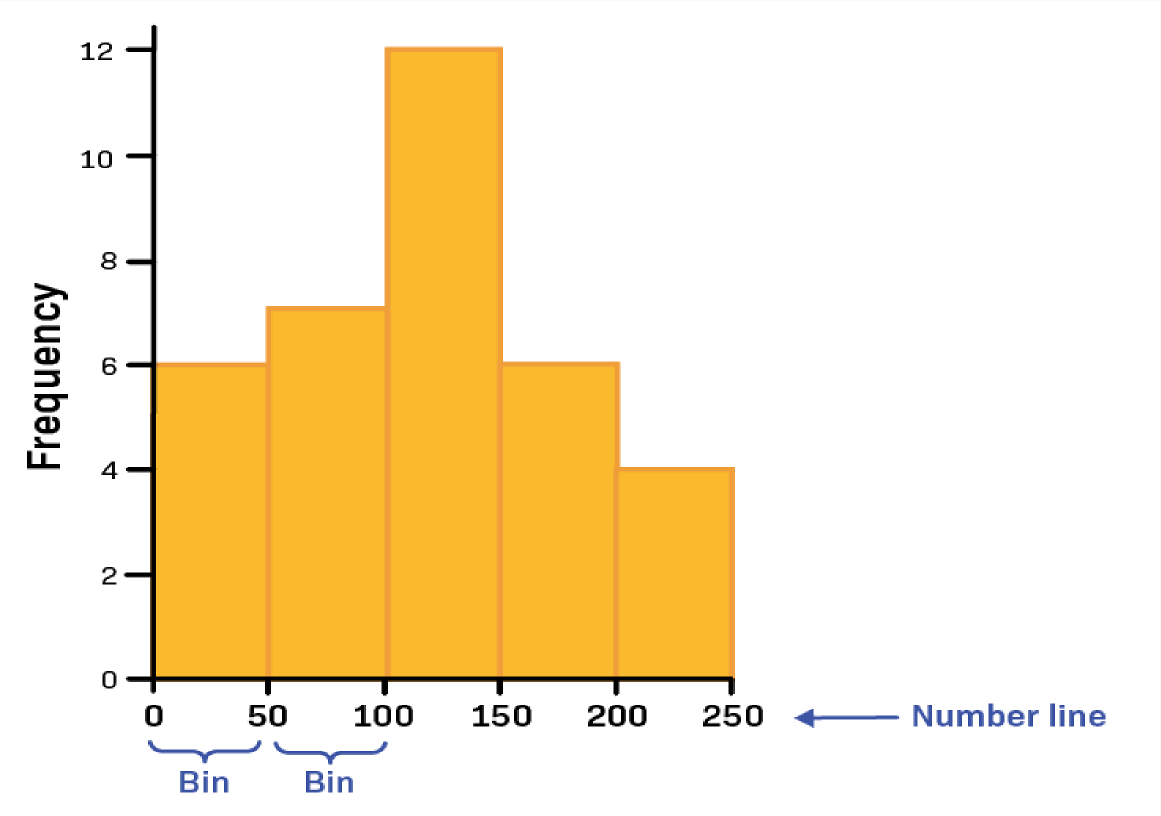
labels. autopct defines the format of the relative frequencies shown on the figure.

autopct = '%.1f' # display the percentage value to 1 decimal place

autopct = '%.2f' # display the percentage value to 2 decimal places

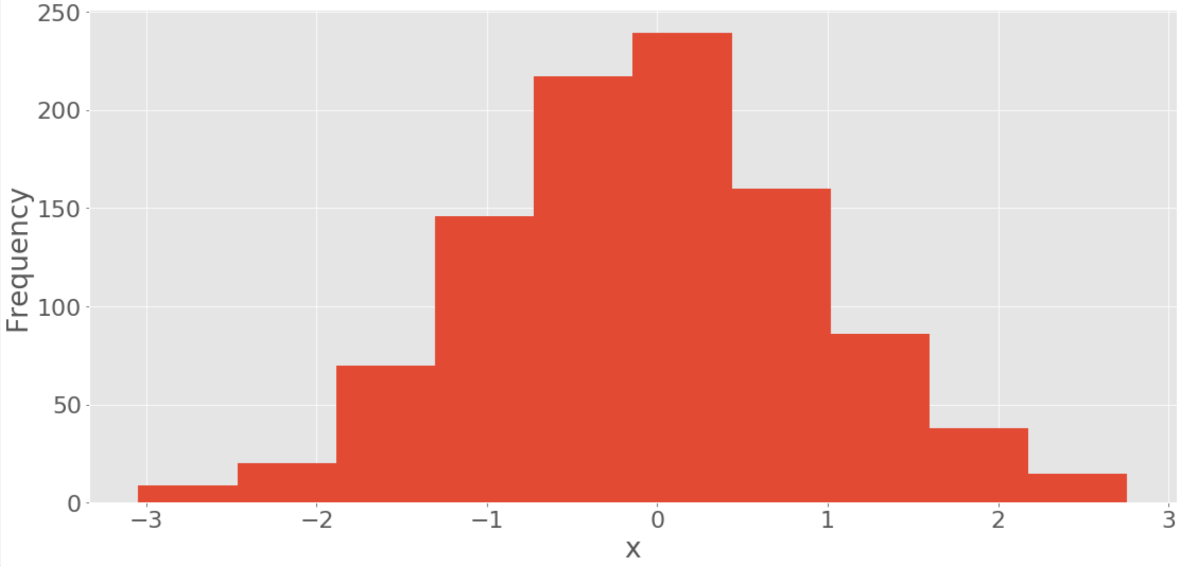
Histograms

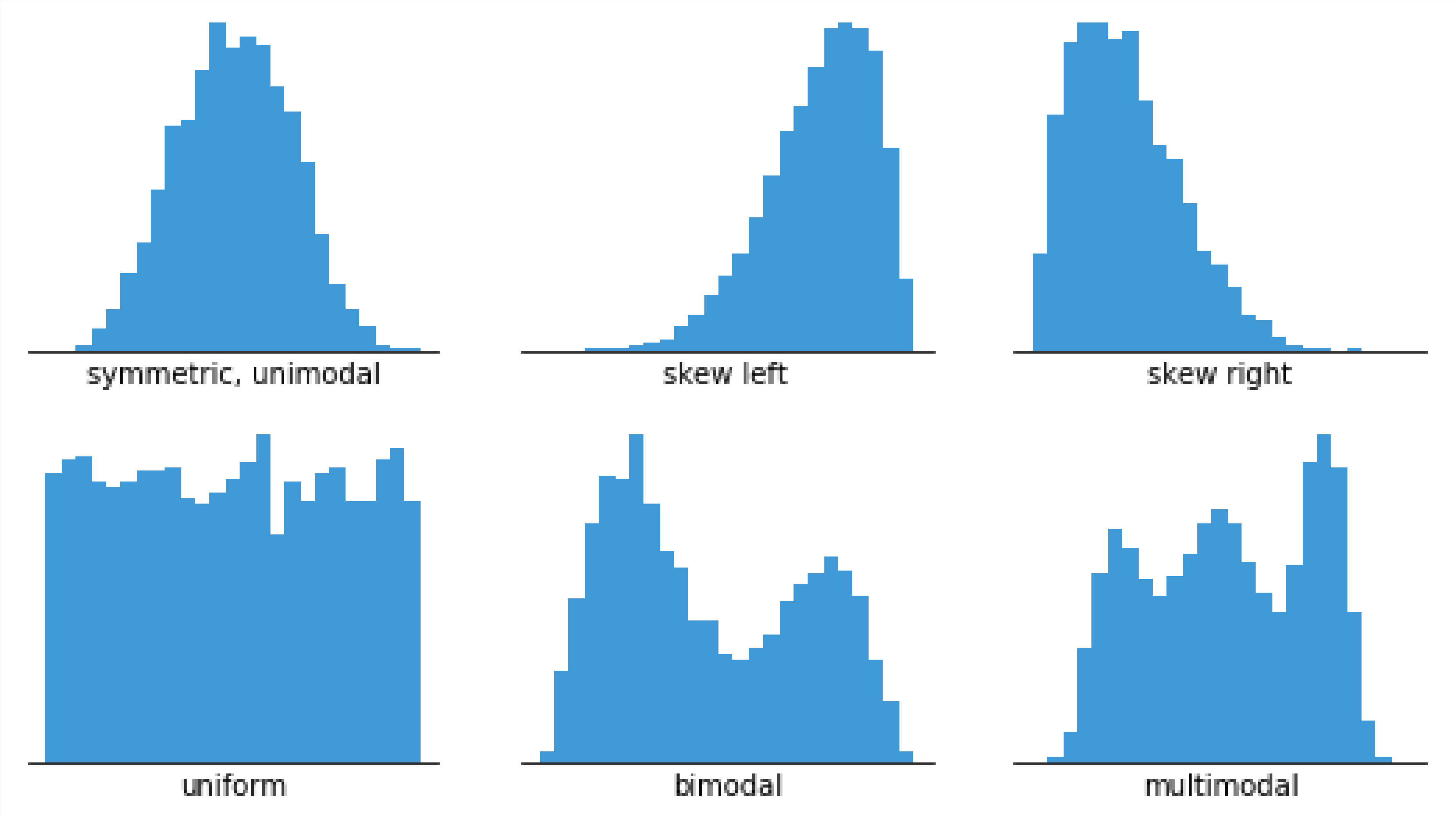
- Histograms are particularly useful when there are a large number of unique values in a dataset. The histogram divides the values from a sorted dataset into intervals, also called **bins**.
- The **frequency** is the number of elements of the dataset with the values between the edges of the bin.



```
>>> x = np.random.randn(1000)
>>> hist, bin_edges = np.histogram(x, bins=10)
>>> hist
array([  9,  20,  70, 146, 217, 239, 160,  86,  38,  15])
>>> bin_edges
array([-3.04614305, -2.46559324, -1.88504342, -1.3044936 , -0.72394379,
       -0.14339397,  0.43715585,  1.01770566,  1.59825548,  2.1788053 , 2.75935511])
```

```
fig, ax = plt.subplots()
ax.hist(x, bin_edges, cumulative=False)
ax.set_xlabel('x')
ax.set_ylabel('Frequency')
plt.show()
```

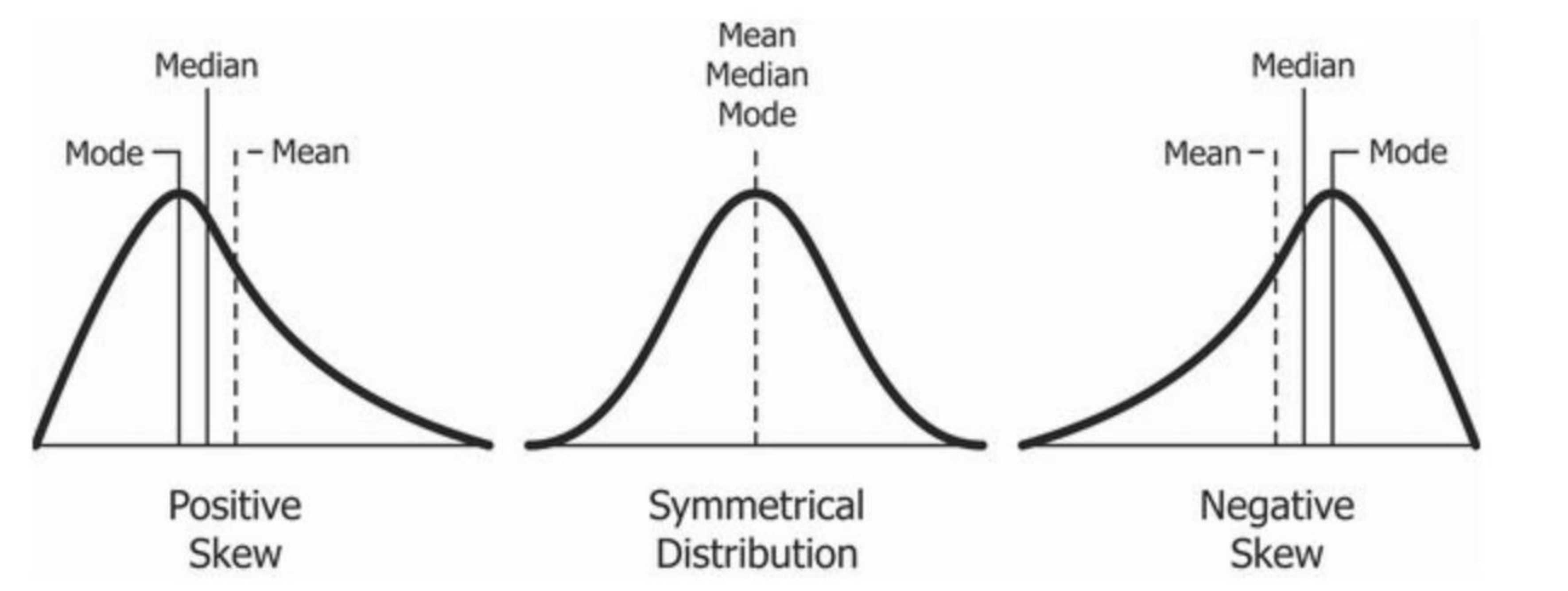



symmetric, unimodal
skew left
skew right
uniform
bimodal
multimodal

Skewness

$$\text{Skew} = \frac{n}{(n-1)(n-2)}\sum\left(\frac{x_i - \bar{x}}{s}\right)^3$$

- The **sample skewness** measures the asymmetry of a data sample.



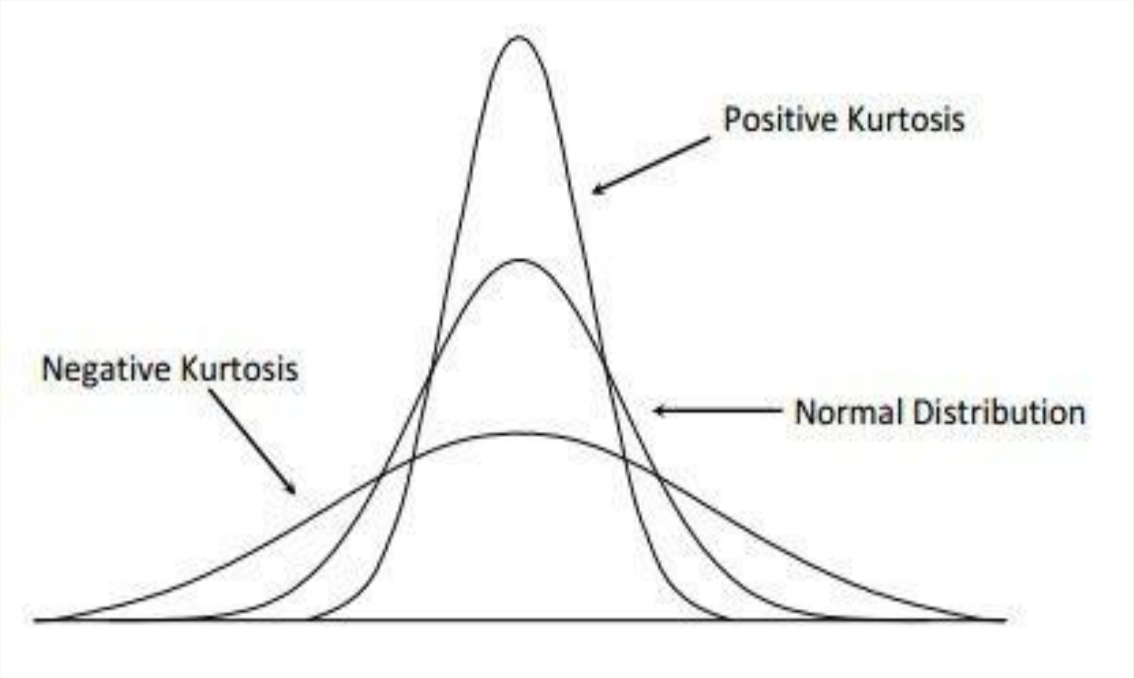
```
>>> x = [8.0, 1, 2.5, 4, 28.0]
>>> y= np.array(x)
>>> scipy.stats.skew(y)
1.3061163034727836

>>> from scipy.stats import skew
>>> skew([1,2,2,3,3,3,4,4,5])
0
```

Kurtosis

$$\text{Kurtosis} = \left\{ \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum \left(\frac{x_i - \bar{x}}{s} \right)^4 \right\} - \frac{3(n-1)^2}{(n-2)(n-3)}$$

Kurtosis is a measure of the tailedness of a distribution. Tailedness is how often outliers occur. Excess kurtosis is the tailedness of a distribution relative to a normal distribution.

- Distributions with no kurtosis (medium tails)
- Distributions with negative kurtosis (thin tails)
- Distributions with positive kurtosis (fat tails)



$$\text{Kurtosis} = \left\{ \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum \left(\frac{x_i - \bar{x}}{s} \right)^4 \right\} - \frac{3(n-1)^2}{(n-2)(n-3)}$$

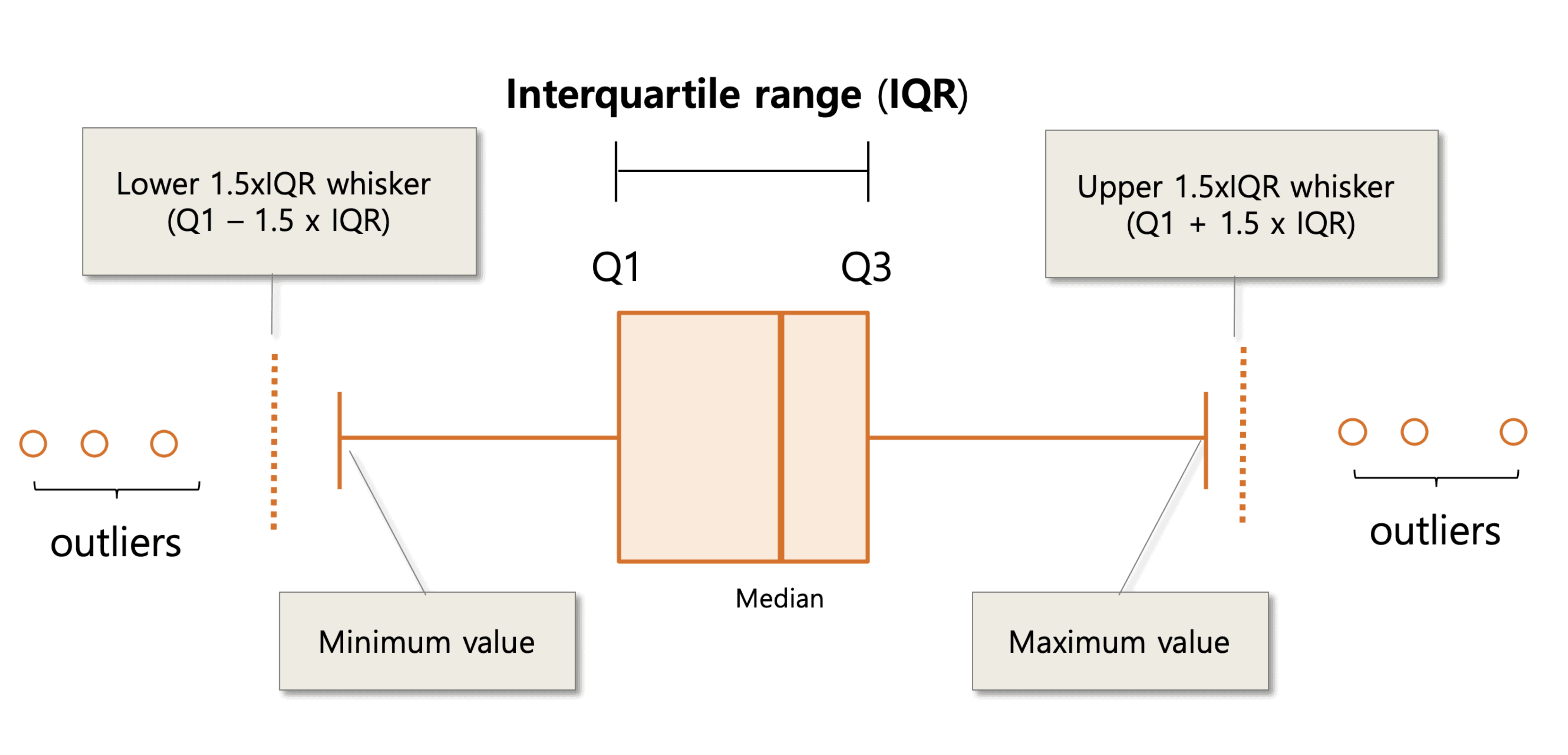
```
>>> y = np.random.randn(10000)
>>> kurtosis(y)
0.08013383529171136
```

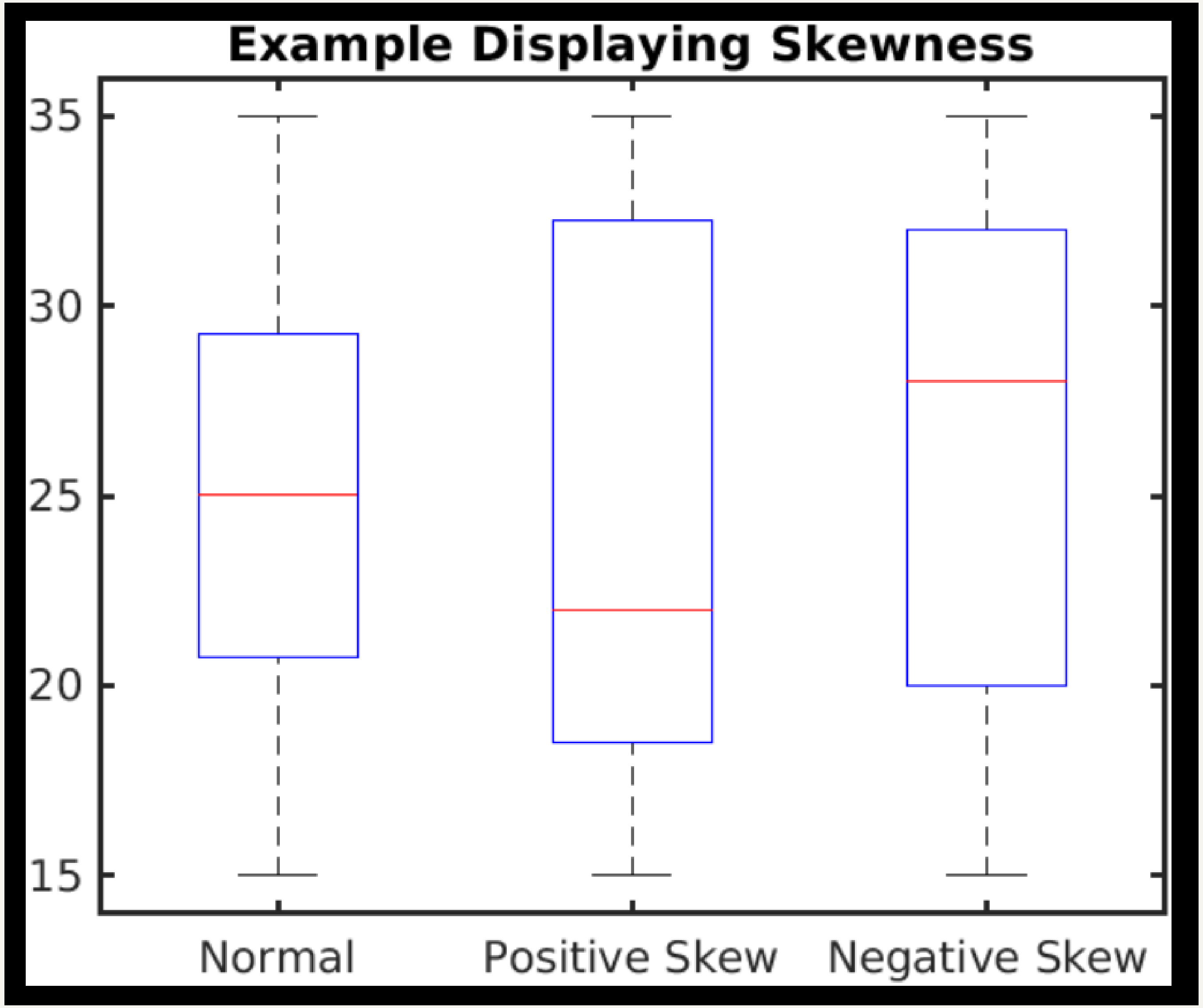
```
>>> from scipy.stats import kurtosis
>>> kurtosis([1,2,2,3,3,3,4,4,5])
-0.75
```

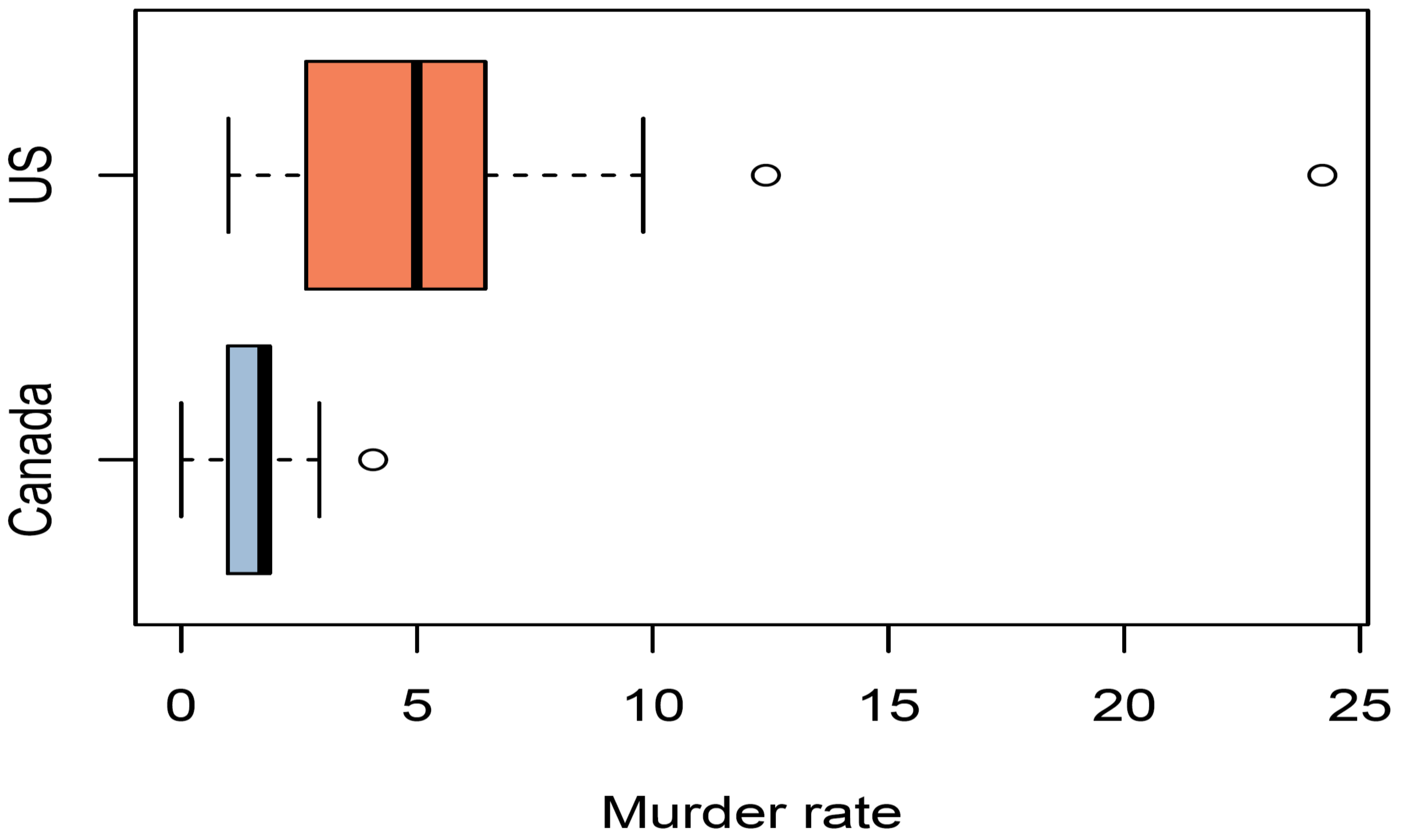
Box Plots

- The **box plot** is an excellent tool to visually represent descriptive statistics of a given dataset. It can show the range, interquartile range, median, mode, outliers, and all quartiles.

Box plot


Interquartile range (IQR)
Lower 1.5xIQR whisker
(Q1 – 1.5 x IQR)
Upper 1.5xIQR whisker
(Q1 + 1.5 x IQR)
Q1
Q3
outliers
outliers
Median
Minimum value
Maximum value


Example Displaying Skewness
35
30
25
20
15
Normal
Positive Skew
Negative Skew


US
Canada
0
5
10
15
20
25
Murder rate

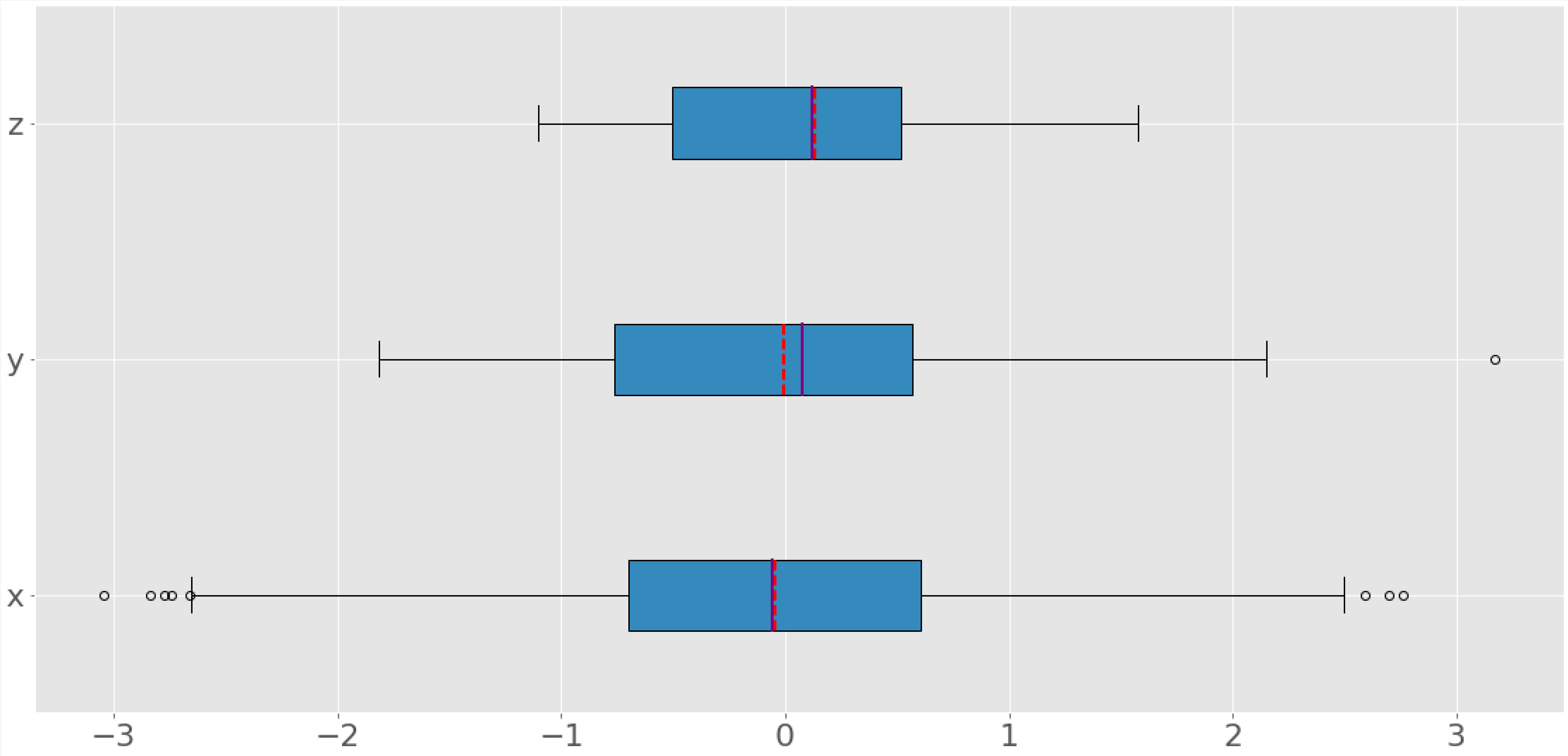
pseudo-random numbers

- You'll use pseudo-random numbers to get data to work with. You don't need knowledge on random numbers to be able to understand this section. You just need some arbitrary numbers, and pseudo-random generators are a convenient tool to get them. The module np.random generates arrays of pseudo-random numbers:

- Normally distributed numbers are generated with np.random.randn().
- Uniformly distributed integers are generated with np.random.randint().

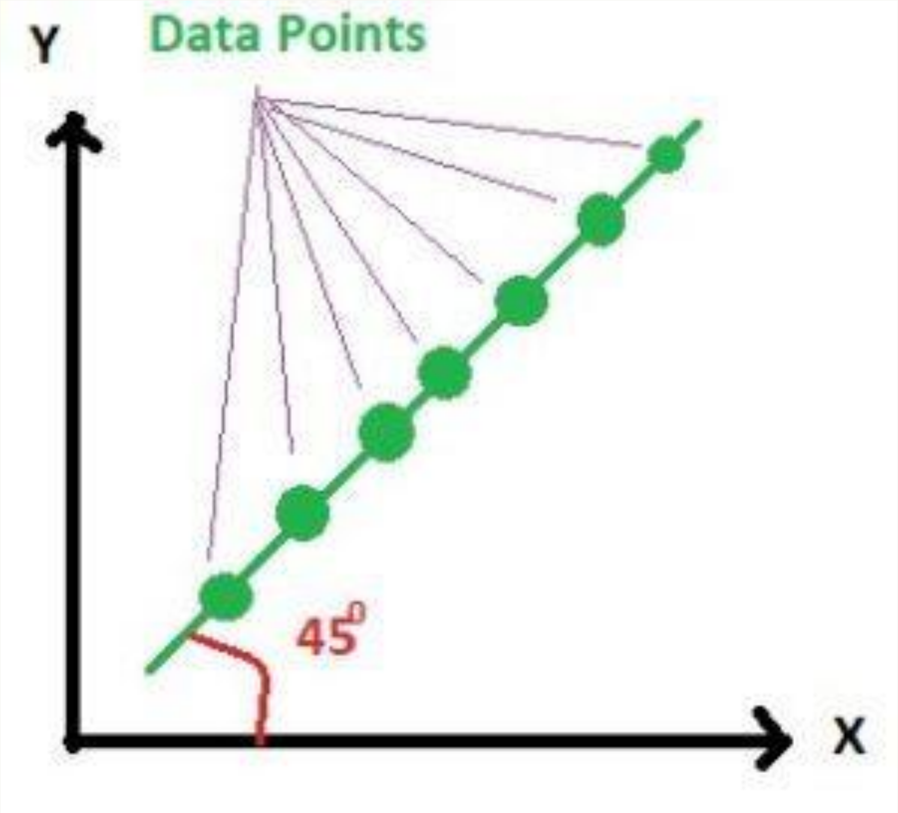
```
>>> np.random.seed(seed=0)
>>> x = np.random.randn(1000)
>>> y = np.random.randn(100)
>>> z = np.random.randn(10)
```

```
fig, ax = plt.subplots()
ax.boxplot((x, y, z), vert=False, showmeans=True, meanline=True,
        labels=('x', 'y', 'z'), patch_artist=True,
        medianprops={'linewidth': 2, 'color': 'purple'},
        meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```

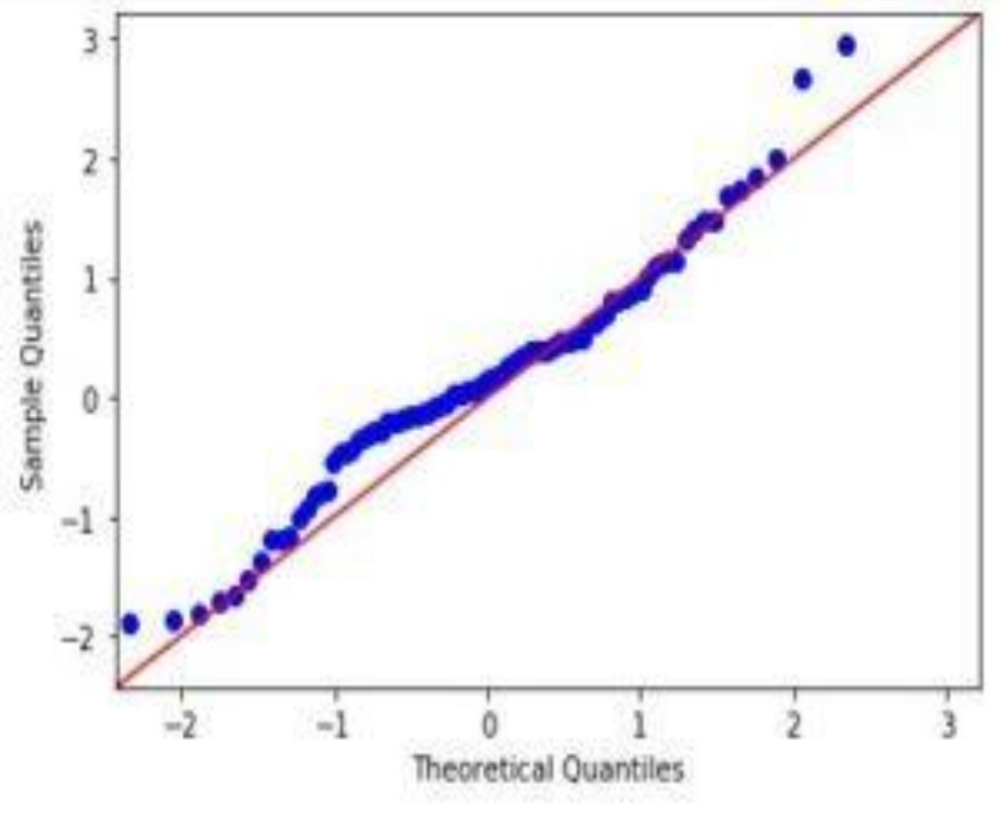

z
y
x
−3
−2
−1
0
1
2
3

qqplot (Quantile-Quantile Plot)

- When the quantiles of two variables are plotted against each other, then the plot obtained is known as quantile – quantile plot or qqplot. This plot provides a summary of whether the distributions of two variables are similar or not with respect to the locations.



```
>>> import numpy as np
>>> import statsmodels.api as sm
>>> import pylab as py

>>> data_points = np.random.normal(0, 1, 100)

sm.qqplot(data_points, line ='45')
py.show()
```



Summary of Descriptive Statistics

- SciPy and pandas offer useful routines to quickly get descriptive statistics with a single function or method call.

```
>>> y = np.random.randn(1000)
>>> result = scipy.stats.describe(y)
>>> result
DescribeResult(nobs=1000, minmax=(-2.8036034681024034,
3.486978995062244), mean=-0.012284543860440592,
variance=0.978831430182489, skewness=0.10302164651020802,
kurtosis=0.12043164014731911)
```

```
>>> import pandas as pd
>>> y = np.random.randn(1000)
>>> z=pd.Series(y)
>>> z.describe()
count      1000.00
mean       0.024883
std        1.011591
min       -3.335723
25%       -0.678700
50%        0.004405
75%        0.683536
max        3.417201
dtype: float64
```

Working With 2D Data

```
>>> a = np.array([[1, 1, 1],
...               [2, 3, 1],
...               [4, 9, 2],
...               [8, 27, 4],
...               [16, 1, 1]])
>>> a
array([[ 1,  1,  1],
       [ 2,  3,  1],
       [ 4,  9,  2],
       [ 8, 27,  4],
       [16,  1,  1]])
```

```
>>> np.mean(a)
5.4
>>> a.mean()
5.4
>>> np.median(a)
2.0
>>> a.var(ddof=1)
53.40000000000001
```

Axes

The functions and methods you've used so far have one optional parameter called axis, which is essential for handling 2D data. axis can take on any of the following values:

- axis=None says to calculate the statistics across all data in the array.
- axis=0 says to calculate the statistics across all rows, that is, for each column of the array. This behavior is often the default for SciPy statistical functions.
- axis=1 says to calculate the statistics across all columns, that is, for each row of the array.

```
>>> np.mean(a, axis=0)
array([6.2, 8.2, 1.8])
>>> a.mean(axis=0)
array([6.2, 8.2, 1.8])
```

```
>>> np.mean(a, axis=1)
array([ 1.,  2.,  5., 13.,  6.])
>>> a.mean(axis=1)
array([ 1.,  2.,  5., 13.,  6.])
```

```
array([[ 1,  1,  1],
       [ 2,  3,  1],
       [ 4,  9,  2],
       [ 8, 27,  4],
       [16,  1,  1]])
```

```
>>> scipy.stats.describe(a, axis=None, ddof=1)
DescribeResult(nobs=15, minmax=(1, 27), mean=5.4, variance=53.400000000000001, skewness=2.264965290423389, kurtosis=5.212690982795767)
>>> scipy.stats.describe(a, ddof=1)  # Default: axis=0
DescribeResult(nobs=5, minmax=(array([1, 1, 1]), array([16, 27,  4])), mean=array([6.2, 8.2, 1.8]), variance=array([ 37.2, 121.2,   1.7]), skewness=array([1.32531471, 1.79809454, 1.71439233]), kurtosis=array([1.30376344, 3.14969121, 2.66435986]))
>>> scipy.stats.describe(a, axis=1, ddof=1)
DescribeResult(nobs=3, minmax=(array([1, 1, 2, 4, 1]), array([ 1,  3,  9, 27, 16])), mean=array([ 1.,  2.,  5., 13.,  6.]), variance=array([  0.,   1.,  13., 151., 75.]), skewness=array([0.        , 0.        , 1.15206964, 1.52787436, 1.73205081]), kurtosis=array([-3. , -1.5, -1.5, -1.5, -1.5]))
```

Data Frames

```
>>> row_names = ['first', 'second', 'third', 'fourth', 'fifth']
>>> col_names = ['A', 'B', 'C']
>>> df = pd.DataFrame(a, index=row_names, columns=col_names)
>>> df
         A   B   C
first    1   1   1
second   2   3   1
third    4   9   2
fourth   8  27   4
fifth   16   1   1
```

```
>>> df.mean()
A    6.2
B    8.2
C    1.8
dtype: float64
>>> df.var()
A     37.2
B    121.2
C      1.7
dtype: float64
```

```
>>> df.mean(axis=1)
first      1.0
second     2.0
third      5.0
fourth    13.0
fifth      6.0
dtype: float64
>>> df.var(axis=1)
first       0.0
second      1.0
third      13.0
fourth    151.0
fifth      75.0
dtype: float64
```

```
>>> df['A']
first      1
second     2
third      4
fourth     8
fifth     16
Name: A, dtype: int64
```

```
>>> df['A'].mean()
6.2
>>> df['A'].var()
37.20000000000001
```

```
>>> df.describe()
              A          B        C
count   5.00000   5.000000  5.00000
mean    6.20000   8.200000  1.80000
std     6.09918  11.009087  1.30384
min     1.00000   1.000000  1.00000
25%     2.00000   1.000000  1.00000
50%     4.00000   3.000000  1.00000
75%     8.00000   9.000000  2.00000
max    16.00000  27.000000  4.00000
```

THANK YOU FOR YOUR ATTENTION

WISH YOU LUCK