DataInterpolation

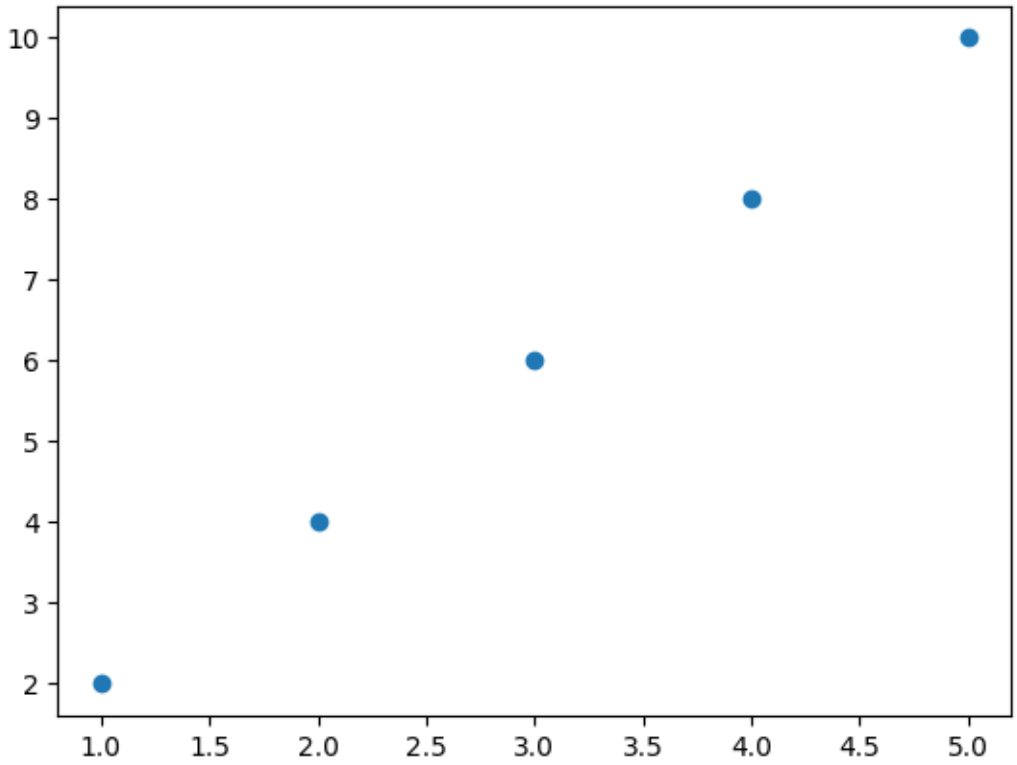
October 11, 2024

0.1 Data Interpolation

Data Interpolation is the process of estimating unknown values within a dataset based on the known values. In Python there are various libraries available that can be used for data interpolation, such as NumPy, SciPy, and Pandas. Here, is an example of how to perform data interpolation using the NumPy library.

[9]:
```
# 1. Linear Interpolation
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

plt.scatter(x, y)

# Interpolating data using linear interpoltaion
```

[9]:
```
<matplotlib.collections.PathCollection at 0x2406d425e90>
```



0.1.1 The np.interp() function in NumPy is used for 1-dimensional linear interpolation. It allows you to estimate unknown values that fall within the range of known data points. Interpolation is commonly used when you have discrete data points and want to find values in between those points.

[2]:
```
# np.interp(x, xp, fp, left=None, right=None, period=None)
```

0.1.2 Parameters

x: The input values (points at which you want to interpolate). These values must lie within the range defined by xp. xp: The x-coordinates of the data points (known values). This should be a 1D array of values sorted in increasing order. fp: The y-coordinates of the data points (known values corresponding to xp). This should have the same length as xp. left: (optional) The value to return for x < xp[0]. If not specified, the default is fp[0]. right: (optional) The value to return for x > xp[-1]. If not specified, the default is fp[-1]. period: (optional) A value for periodic interpolation. This is used when xp and fp are periodic, and x is outside the range of xp.

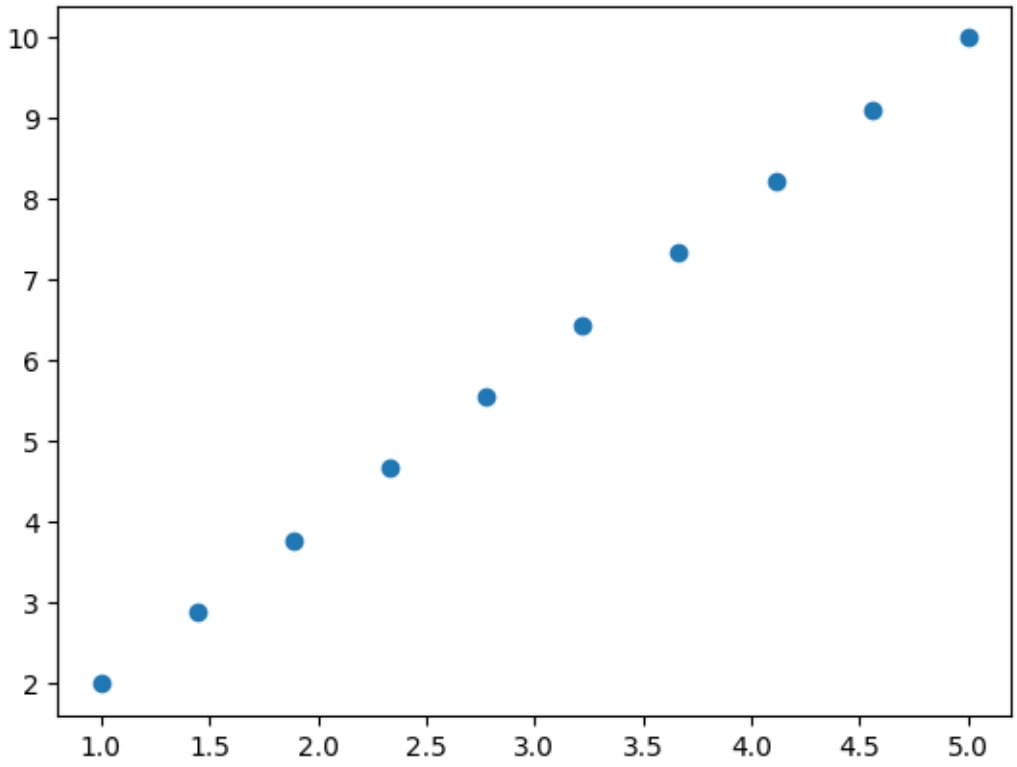
0.1.3 How It Works

The function takes the x-coordinates (xp) and their corresponding y-coordinates (fp) and uses linear interpolation to estimate the value of y for any input value in x. Linear interpolation means that it connects the known points with straight lines and finds the value on the line corresponding to the input value.

[11]:
```
x_new = np.linspace(1, 5, 10) # create new x values
y_interp = np.interp(x_new, x, y) # interpolating y values
print(y_interp)

plt.scatter(x_new, y_interp)
```

```
[ 2.          2.88888889  3.77777778  4.66666667  5.55555556  6.44444444
  7.33333333  8.22222222  9.11111111 10.        ]
```

[11]:
```
<matplotlib.collections.PathCollection at 0x2406d653510>
```



0.1.4 Overview of Cubic Interpolation

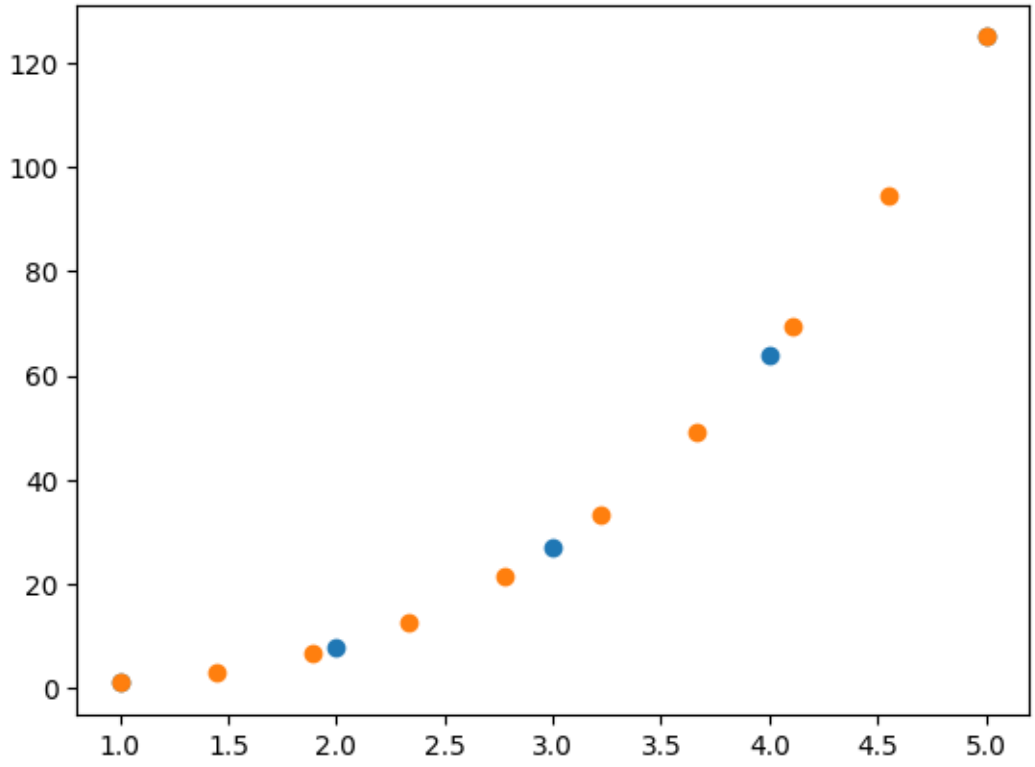
Cubic interpolation is a method of estimating unknown values that fall between known data points using cubic polynomials. Unlike linear interpolation (which connects points with straight lines), cubic interpolation provides a smoother curve by using polynomial functions of degree three. This approach can lead to better approximations, especially when the underlying data varies in a non-linear fashion.

[13]:
```
# 2. Cubic Interpolation
from scipy.interpolate import interp1d

x1 = np.array([1, 2, 3, 4, 5])
```

```
y1 = np.array([1, 8, 27, 64, 125])

# create cubic interpolation function
f = interp1d(x1, y1, kind='cubic')

# interpolating the data
x1_new = np.linspace(1, 5, 10)
y1_interp = f(x1_new)
print(y1_interp)

# Graph before interpolation
plt.scatter(x1, y1)

# Graph after interpolation
plt.scatter(x1_new, y1_interp)
```

```
[  1.           3.01371742   6.739369    12.7037037   21.43347051
  33.45541838  49.2962963   69.48285322  94.54183813 125.        ]
```

[13]: <matplotlib.collections.PathCollection at 0x2406d47a850>



0.1.5 Creating the Interpolation Function:

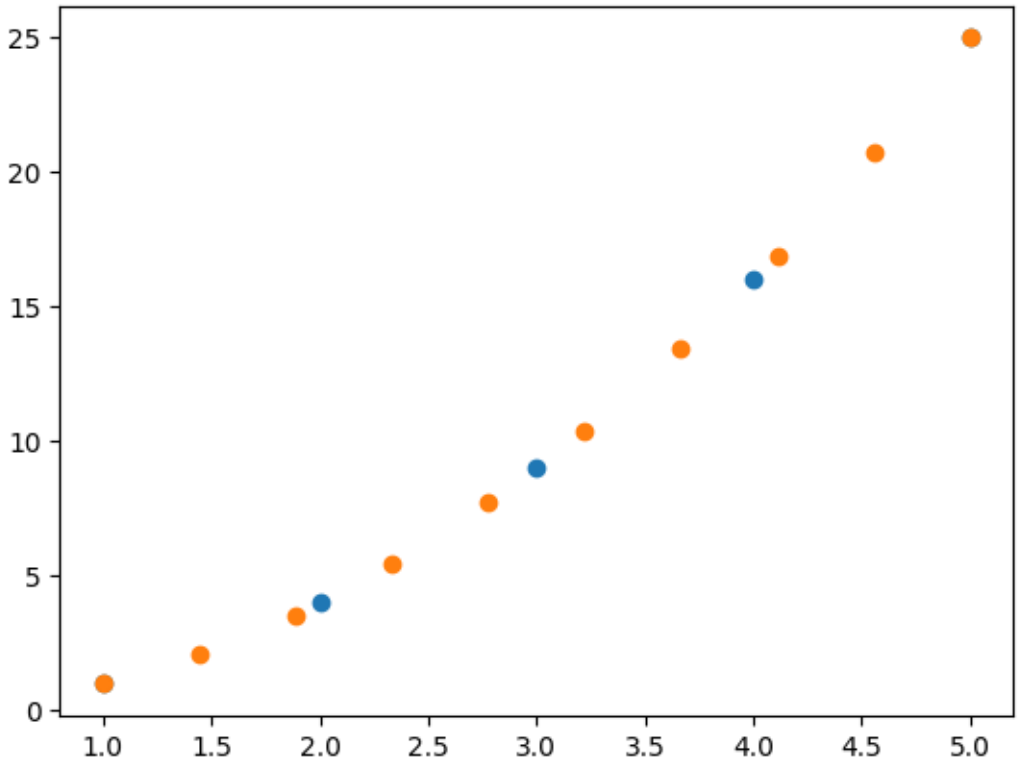
interp1d() is a function from the scipy.interpolate module that creates an interpolation function based on the provided data. The kind='cubic' argument specifies that we want to perform cubic interpolation. This function f will take new x-values and return corresponding interpolated y-values using cubic polynomials.

0.1.6 Generating New Data Points:

x1_new = np.linspace(1, 5, 10) creates an array of 10 evenly spaced values between 1 and 5. This will be used for interpolation. y1_interp = f(x1_new) uses the cubic interpolation function f to calculate the corresponding y-values for the new x-values in x1_new. This results in a smoother curve.

[36]:
```
# Polynomial Interpolation

x2 = np.array([1, 2, 3, 4, 5])
y2 = np.array([1, 4, 9, 16, 25])

# interpolate data using polynomial interpolation
p = np.polyfit(x2, y2, 2) # degree value

x2_new = np.linspace(1, 5, 10)
y2_interp = np.polyval(p, x2_new)

# Graph before interpolation
plt.scatter(x2, y2)

# Graph after interpolation
plt.scatter(x2_new, y2_interp)
```

[36]:
```
<matplotlib.collections.PathCollection at 0x1ba46ccd510>
```



0.1.7 Fitting a Polynomial:

np.polyfit(x2, y2, 2) is a function that fits a polynomial of degree 2 to the provided data points. The first argument is the x-coordinates (x2). The second argument is the y-coordinates (y2). The third argument (2) specifies the degree of the polynomial to fit. The output, p, is an array of coefficients for the polynomial, starting from the highest degree to the constant term. For example, if the result is [1, 0, 0], it represents the polynomial 1 ^2 + 0 + 0 (which is = ^2)

0.1.8 Evaluating the Polynomial:

np.polyval(p, x2_new) evaluates the polynomial (defined by the coefficients in p) at the new x-values in x2_new. The result, y2_interp, contains the corresponding y-values computed by the polynomial for each x-value in x2_new.

[ ]: