In **pandas**, we use **interpolation** instead of `ffill` **(forward fill)** and `bfill` **(backward fill)** when we want to estimate missing values more accurately, rather than just propagating the previous or next known value.

Key Differences:

Method	How it Works	When to Use
`ffill` **(Forward Fill)**	Fills missing values with the last known value	Works well for time series or categorical data where values don't change frequently
`bfill` **(Backward Fill)**	Fills missing values with the next known value	Suitable for cases where future values are valid replacements
`interpolate()`	Estimates missing values based on surrounding data points using linear or other methods	Best for numerical or continuous data, ensuring smoother transitions

Why Use `interpolate()`?

1. **More Accurate for Numerical Data** – Instead of just copying values, it estimates missing points based on trends in the data.
2. **Supports Different Methods** – Linear, polynomial, spline, etc., allowing more flexibility.
3. **Preserves Trends** – Especially useful in **time series** and **scientific datasets** where values change gradually.

Interpolation Methods in Pandas (`interpolate()`)

The `interpolate()` method in pandas provides different techniques to estimate missing values. The choice of method depends on the data type and trend. Below are some key methods with examples:

---

1. Linear Interpolation (Default)

Formula:

$y=y1+(y2−y1)(x2−x1)×(x−x1)y = y\_1 + \frac{(y\_2 - y\_1)}{(x\_2 - x\_1)} \times (x - x\_1)y=y1+(x2−x1)(y2−y1)×(x−x1)$

- This method assumes a straight-line relationship between two known points.
- Works well for **smooth** numerical data.

Example:

```
python
CopyEdit
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame({'A': [1, np.nan, np.nan, 4, 5]})
df_linear = df.interpolate(method='linear')

print(df_linear)
```

Output:

```
css
CopyEdit
     A
0  1.0
1  2.0
2  3.0
3  4.0
4  5.0
```

Missing values (2.0 and 3.0) are estimated linearly between 1 and 4.

2. Polynomial Interpolation

- Uses polynomial functions to estimate values.
- Suitable for data that follows a **curved** trend.
- You must specify the degree of the polynomial.

Example (Quadratic - Degree 2):

```
python
CopyEdit
df_poly = df.interpolate(method='polynomial', order=2)
print(df_poly)
```

This method fits a quadratic function through the available points.

---

3. Spline Interpolation (Smooth Curves)

- A more flexible version of polynomial interpolation.
- Fits a **smooth curve** using splines.
- You need to specify the order of the spline.

Example (Cubic Spline):

```
python
CopyEdit
df_spline = df.interpolate(method='spline', order=3)
print(df_spline)
```

This is useful when data has **non-linear trends** and requires smooth transitions.

---

4. Time-based Interpolation

- Ideal for **time series data**.
- Uses the index as time to estimate missing values.

Example:

```
python
CopyEdit
df_time = df.interpolate(method='time')
print(df_time)
```

Works only if the index is a `DatetimeIndex`*.*

---

5. Piecewise Cubic Hermite Interpolation (`pchip`)

- Preserves monotonicity, avoiding overshooting issues in steep trends.
- Suitable for **irregular data**.

Example:

```
python
CopyEdit
df_pchip = df.interpolate(method='pchip')
print(df_pchip)
```

Choosing the Right Method:

Method	Best Use Case
`linear`	Simple and smooth numerical data
`polynomial`	When data follows a curved trend
`spline`	When smooth transitions are needed
`time`	For time-series data
`pchip`	When avoiding overshooting is important