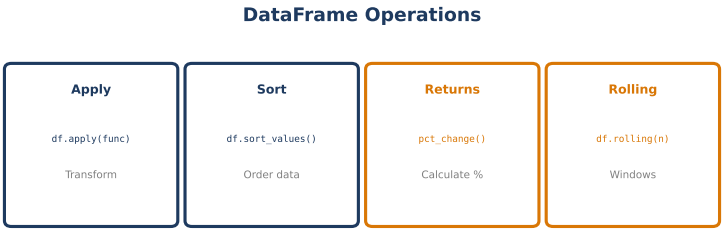
Lesson 08 Summary: Basic Operations

Data Science with Python – Key Concepts

Data Science Program

Operations Overview



Key Finance Operations:

`pct_change()` | `cumsum()` | `rolling().mean()` | `shift()`

Vectorized operations are fast - avoid loops!

Vectorized operations are fast and efficient

Apply Function

Apply custom functions to data:

- **To column:** `df['col'].apply(func)`
- **To DataFrame:** `df.apply(func, axis=0/1)`
- **Lambda:** `df['col'].apply(lambda x: x*2)`

Example – Categorize prices:

```
def categorize(x):
    return 'High' if x > 190 else 'Low'
df['Category'] = df['Price'].apply(categorize)
```

apply() is flexible but slower than vectorized operations

Sorting Data

Order data by values:

- **Single column:** `df.sort_values('col')`
- **Descending:** `df.sort_values('col', ascending=False)`
- **Multiple:** `df.sort_values(['A', 'B'])`
- **By index:** `df.sort_index()`

Finance use: Rank stocks by return, volume, or market cap

Sorting creates a new DataFrame by default

Calculating Returns

Percentage changes for time series:

- **Simple return:** `df['Price'].pct_change()`
- **Log return:** `np.log(df['Price']/df['Price'].shift(1))`
- **Cumulative:** `(1 + returns).cumprod()`

Key statistics:

```
mean_return = df['Return'].mean()
volatility = df['Return'].std() * np.sqrt(252)
```

Log returns are additive; simple returns compound

Rolling Windows

Calculate statistics over moving windows:

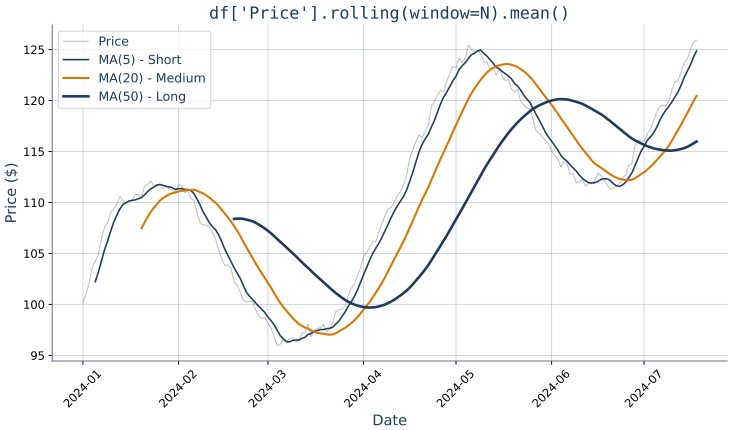
- **Moving average:** `df['Price'].rolling(20).mean()`
- **Rolling std:** `df['Return'].rolling(20).std()`
- **Min/max:** `df['Price'].rolling(20).min()`

Finance applications:

- Moving average crossover signals
- Rolling volatility (annualize with $\times\sqrt{252}$)
- Bollinger Bands

First n-1 values will be NaN for window size n

Moving Averages



Shorter windows are more responsive; longer are smoother

Value Counts

Count occurrences of categorical values:

- **Counts:** `df['col'].value_counts()`
- **Percentages:** `df['col'].value_counts(normalize=True)`
- **Include NaN:** `value_counts(dropna=False)`

Finance use:

- Count stocks by sector
- Signal distribution (buy/sell/hold)
- Rating breakdown

value_counts() is useful for categorical analysis

Arithmetic Operations

Column-wise calculations:

- **Add:** `df['A'] + df['B']`
- **Multiply:** `df['Price'] * df['Shares']`
- **Divide:** `df['A'] / df['B']`

Finance examples:

```
df['Spread'] = df['High'] - df['Low']
df['Midpoint'] = (df['High'] + df['Low']) / 2
df['Range_Pct'] = df['Spread'] / df['Close']
```

Vectorized operations are much faster than loops

Quick Reference

Essential Operations:

Operation	Syntax
Apply function	`df['col'].apply(func)`
Sort values	`df.sort_values('col')`
Percent change	`df['col'].pct_change()`
Rolling mean	`df['col'].rolling(n).mean()`
Rolling std	`df['col'].rolling(n).std()`
Value counts	`df['col'].value_counts()`
Shift	`df['col'].shift(1)`
Cumulative sum	`df['col'].cumsum()`

Master these for efficient financial analysis