Fachhochschule Kiel
University of Applied Sciences

Tools and Programming Languages for Data Science
Pandas: Data Exploration

Study Program Data Science
Prof. Dr. Tillmann Schwörer

Our course agenda

- **Introduction and overview**
- **NumPy**: Basic data handling with Numpy arrays
- **Pandas**
  - Exploratory data analysis
  - Data consolidation
  - Data cleaning
- **Data visualization using Matplotlib and Seaborn**
- **Interacting with APIs**
- **Interacting with SQL databases**
- **Version Control with Git and GitHub**
- **Advanced Python**

Python foundations

Data types

Operators

Functions

Control flow and iterators

Programming concepts & paradigms

See also Precourse Programming

Tooling

Installation

Visual Studio Code

Jupyter Notebooks

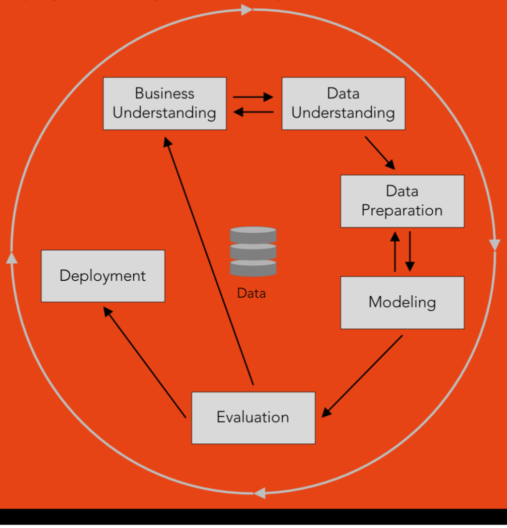
Packages

Virtual Environments

Git and Github

Python

Data Science Workflow



NumPy pandas matplotlib

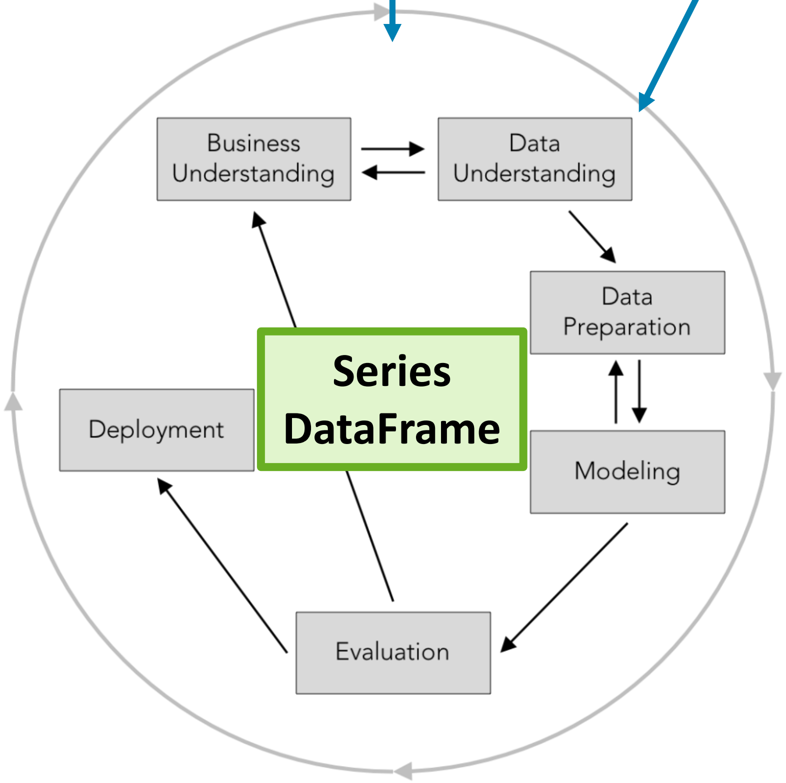
Pandas in the Data Science Process

Data Input and Output

- Python objects
- Files
- Databases

Data Exploration

- Overview
- Selecting data subsets
- Sorting
- Aggregating
- Visualizing



Data Preparation

Data Cleaning

- Identify missing values, duplicates, outliers, …
- Remove, replace, rename, …

Data Enhancing

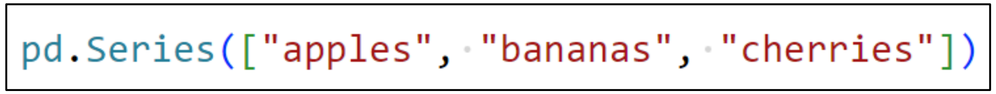
- Combining multiple Series / DataFrames
- Reshaping
- Feature Engineering
- Operations on datetime, string, and categorical data

Pandas Series and DataFrame

Pandas Series

index

0	"apples"
1	"bananas"
2	"cherries



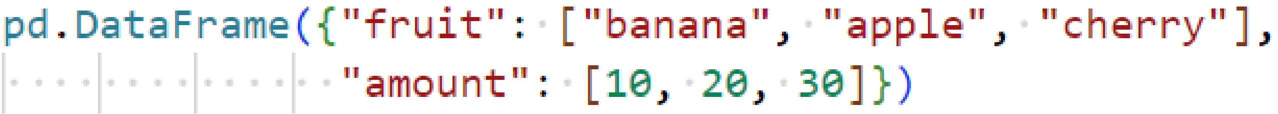
```
pd.Series(["apples", "bananas", "cherries"])
```

Pandas DataFrame

columns

index

	fruit	amount
0	"apples"	3
1	"bananas"	2
2	"cherries	5



```
pd.DataFrame({"fruit": ["banana", "apple", "cherry"],
              "amount": [10, 20, 30]})
```

Attributes of a DataFrame

	fruit	amount
0	"apples"	3
1	"bananas"	2
2	"cherries	5

df.index →

```
RangeIndex(start=0, stop=3, step=1)
```

df.columns →

```
Index(['fruit', 'amount'], dtype='object')
```

df.dtypes →

```
fruit      object
amount      int64
dtype: object
```

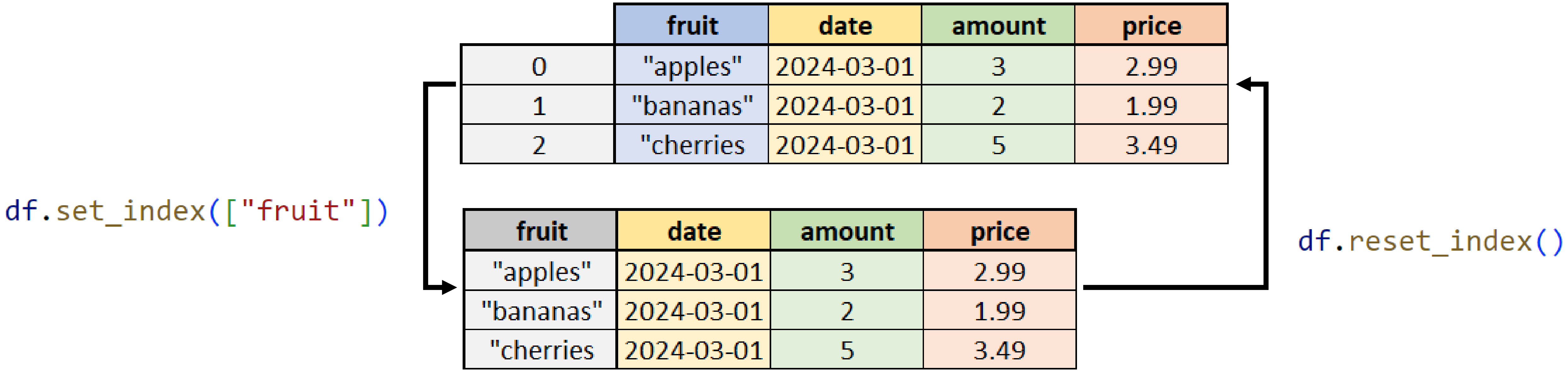
df.shape →

```
(3, 2)
```

df.size →

```
6
```

Index



	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49

```
df.set_index(["fruit"])
```

fruit	date	amount	price
"apples"	2024-03-01	3	2.99
"bananas"	2024-03-01	2	1.99
"cherries	2024-03-01	5	3.49

```
df.reset_index()
```

- **It <u>can</u> be useful to use an index based on one or multiple columns of the DataFrame:**
  - Fast selection of data
  - Intuitive and fast merge of multiple DataFrames
  - Efficient operations on time series data
  - Some visualization packages work naturally with index columns

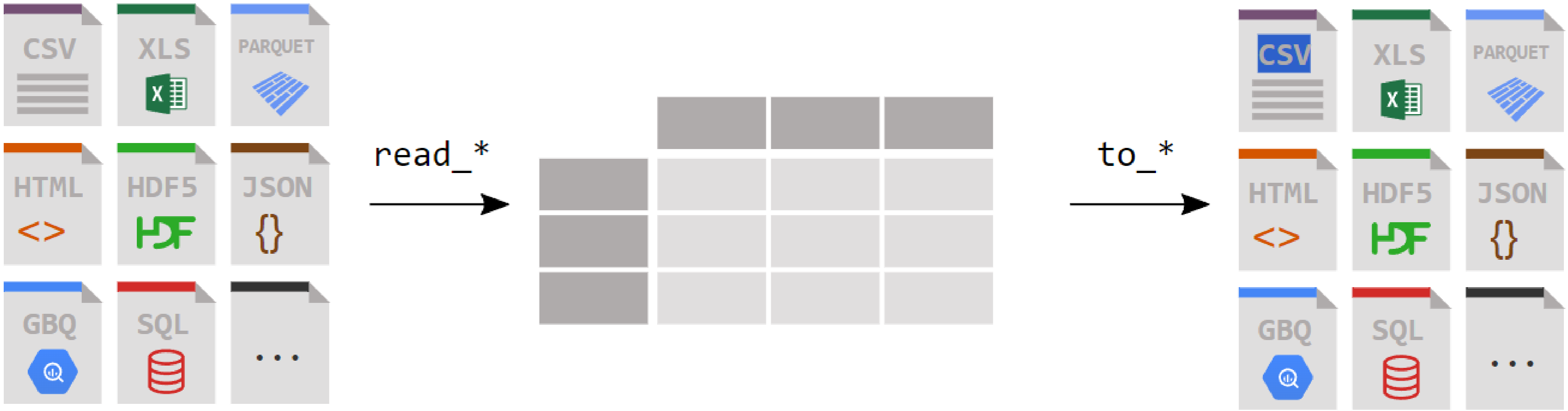
Data Input and Output

Typically, we read in data from files

- **pd**.read_csv(*filename*)
- **pd**.read_excel(*filename*)
- **pd**.read_sql()
- …

Or write data back into some files

- **df**.to_csv(*filename*)
- **df**.to_excel(*filename*)
- **df**.to_sql()
- …

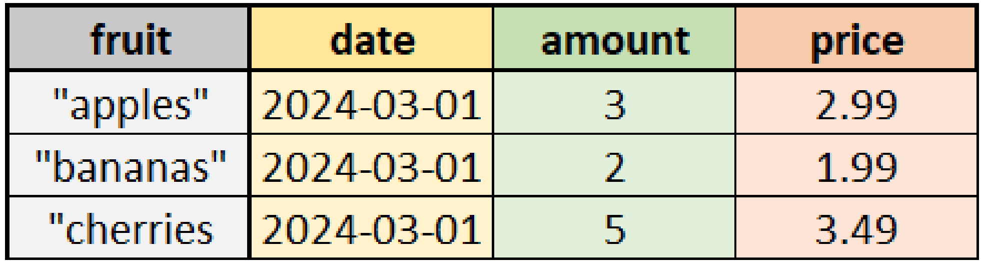


Selecting Data Subsets

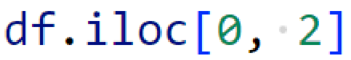
.iloc accessor

Selecting rows and columns based on INTEGER LOCATION

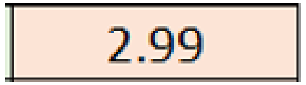
- **.iloc[***row_indices, col_indices***]**
- We specifiy row and column indices in same way as for Numpy arrays
  - start:stop:step
  - Zero-based
  - Negative indexing allowed



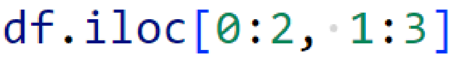
fruit	date	amount	price
"apples"	2024-03-01	3	2.99
"bananas"	2024-03-01	2	1.99
"cherries	2024-03-01	5	3.49



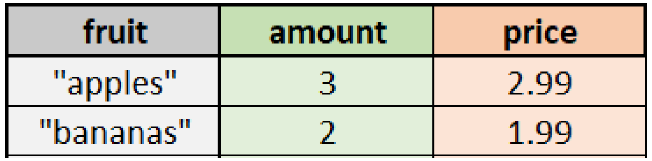
```
df.iloc[0, 2]
```



2.99



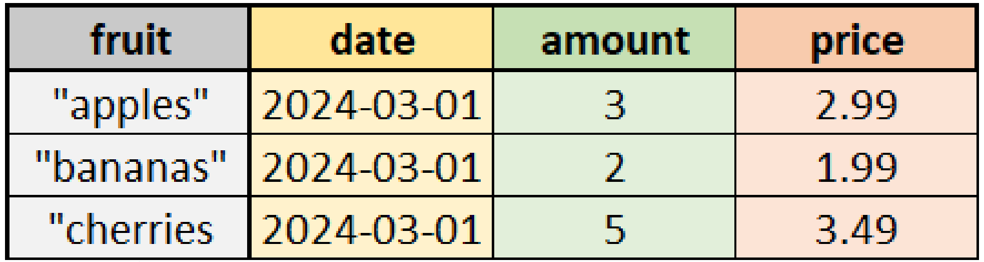
```
df.iloc[0:2, 1:3]
```



fruit	amount	price
"apples"	3	2.99
"bananas"	2	1.99

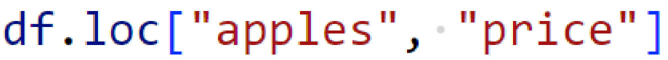
loc accessors

Selecting rows and columns based on LABELS

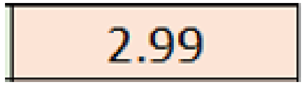


fruit	date	amount	price
"apples"	2024-03-01	3	2.99
"bananas"	2024-03-01	2	1.99
"cherries	2024-03-01	5	3.49

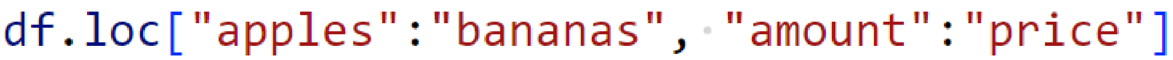
Index column



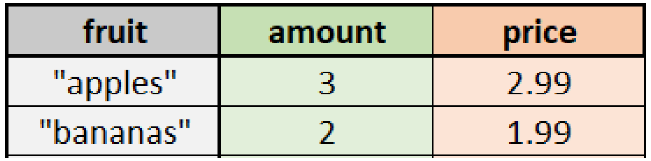
```
df.loc["apples", "price"]
```



2.99

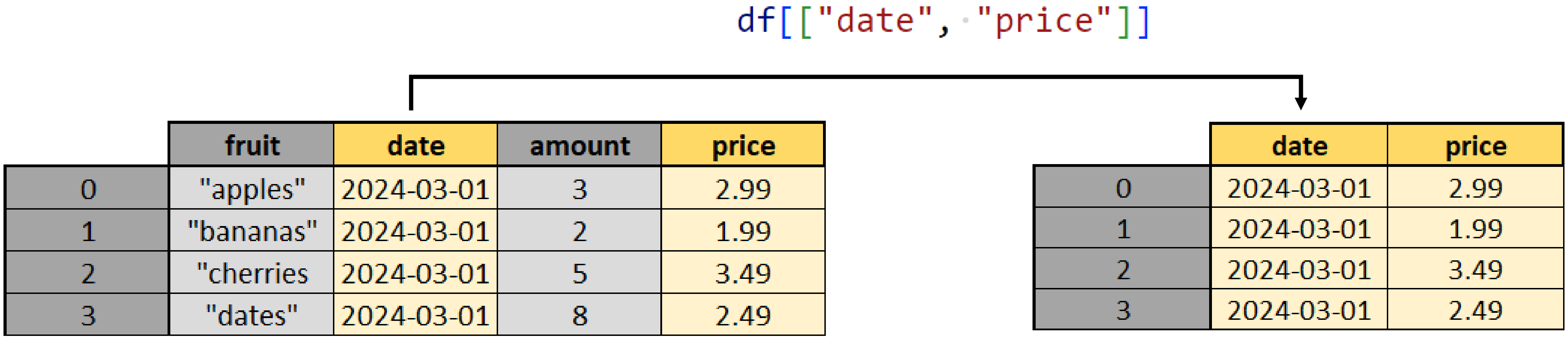


```
df.loc["apples":"bananas", "amount":"price"]
```



fruit	amount	price
"apples"	3	2.99
"bananas"	2	1.99

Selecting only columns



```
df[["date", "price"]]
```

	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49

	date	price
0	2024-03-01	2.99
1	2024-03-01	1.99
2	2024-03-01	3.49
3	2024-03-01	2.49

- **df[[“price”]]** → returns DataFrame with 1 column
- **df[“price”]** or **df.price** → returns Series

Selecting rows using boolean mask

Conditional Expression that evaluates to a Pandas Series of length equal to number of rows in DataFrame

```
df.price < 2.5
```

	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49

False
True
False
True

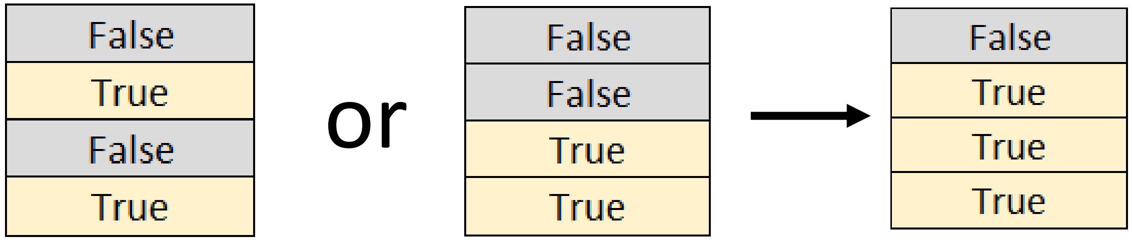
```
df[df.price < 2.5]
```

	fruit	date	amount	price
1	"bananas"	2024-03-01	2	1.99
3	"dates"	2024-03-01	8	2.49

Multiple conditions: or operator “|”

```
(df.price < 2.5) | (df.amount > 4)
```

	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49



False	or	False	→	False
True		False		True
False		True		True
True		True		True

```
df[(df.price < 2.5) | (df.amount > 4)]
```

	fruit	date	amount	price
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49

Multiple conditions: and operator “&”

```
(df.price < 2.5) & (df.amount > 4)
```

	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49

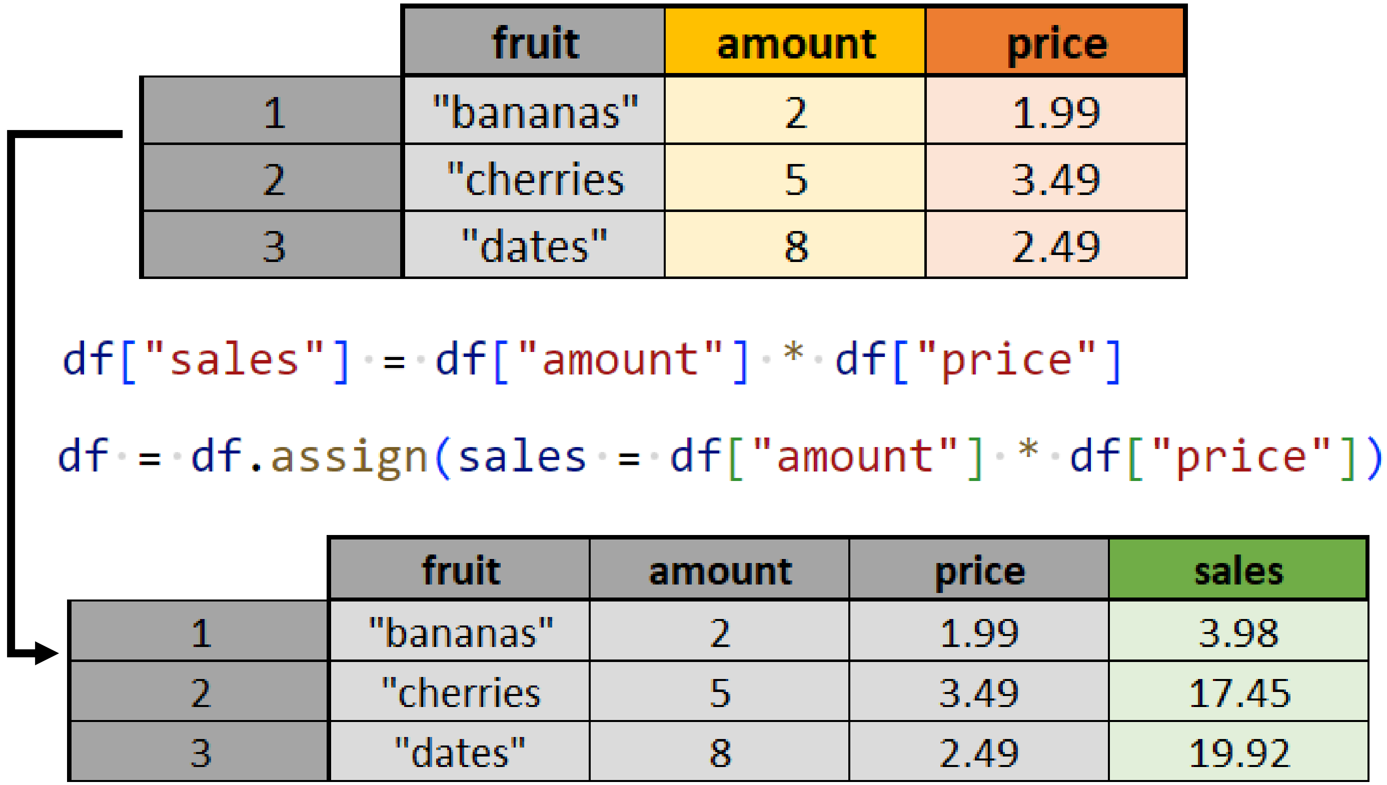
False		False		False
True	and	False	→	False
False		True		False
True		True		True

```
df[(df.price < 2.5) & (df.amount > 4)]
```

	fruit	date	amount	price
3	"dates"	2024-03-01	8	2.49

Manipulating Data

Derive new columns from existing ones



	fruit	amount	price
1	"bananas"	2	1.99
2	"cherries	5	3.49
3	"dates"	8	2.49

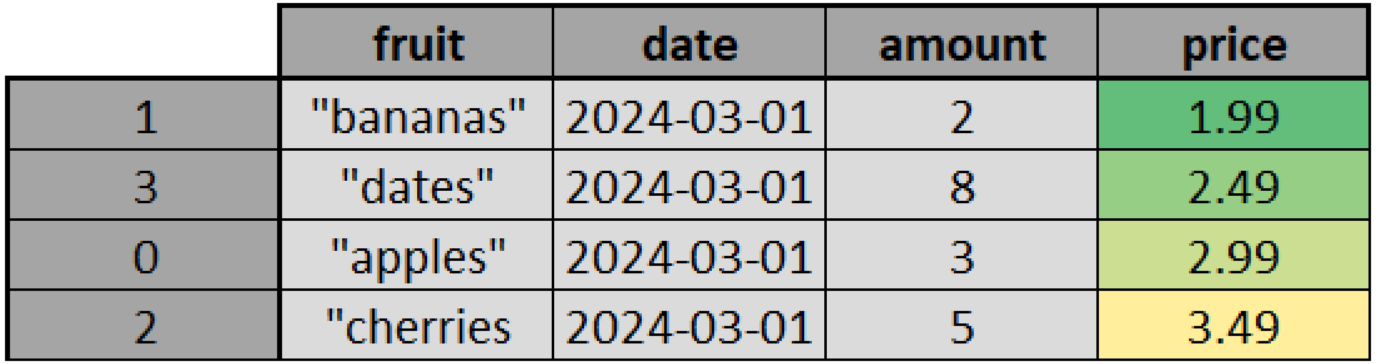
```
df["sales"] = df["amount"] * df["price"]

df = df.assign(sales = df["amount"] * df["price"])
```

	fruit	amount	price	sales
1	"bananas"	2	1.99	3.98
2	"cherries	5	3.49	17.45
3	"dates"	8	2.49	19.92

- Calculations are performed elementwise
- No need for a loop
- To overwrite an existing column with new values we can use the same syntax
- The assign method creates a new DataFrame, while the first approach modifies the existing DataFrame

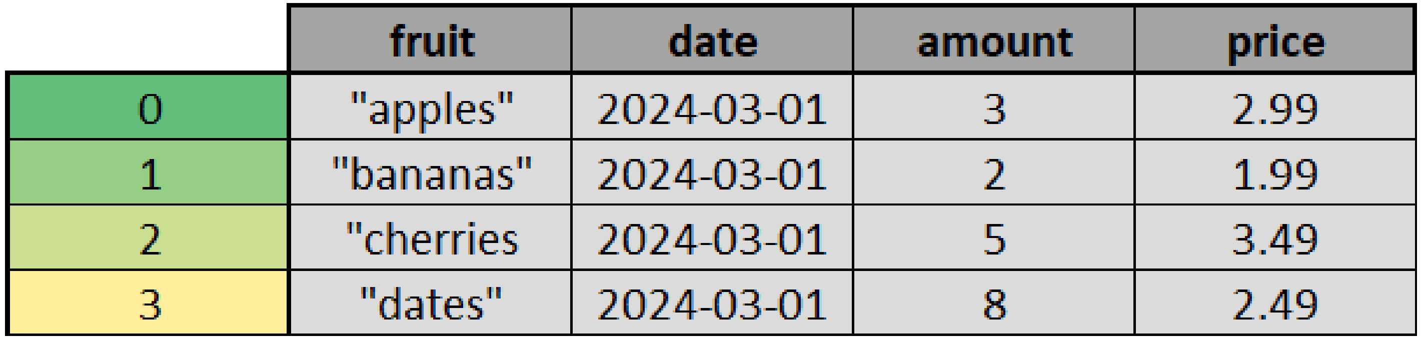
Sorting

- **sort_values**: sort according to a column / list of multiple columns
- **sort_index**: sort according to an index / multiple indices
- Argument **ascending=False** for inverse sorting



	fruit	date	amount	price
1	"bananas"	2024-03-01	2	1.99
3	"dates"	2024-03-01	8	2.49
0	"apples"	2024-03-01	3	2.99
2	"cherries	2024-03-01	5	3.49

```
df.sort_values("price")
```



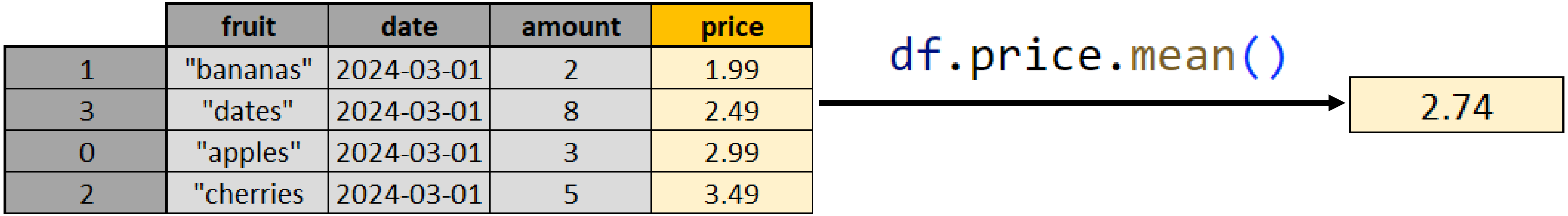
	fruit	date	amount	price
0	"apples"	2024-03-01	3	2.99
1	"bananas"	2024-03-01	2	1.99
2	"cherries	2024-03-01	5	3.49
3	"dates"	2024-03-01	8	2.49

```
df.sort_index()
```

Aggregating Data

Aggregation Functions

- Statistical methods for Series and DataFrames
  - **count**: number of non-missing observations
  - **sum, mean, min, max**
  - **std**: standard deviation
  - **var**: variance
  - **cumsum**: cumulative sum
  - …
- Most operations return a single aggregated value (sum, mean, …) per column

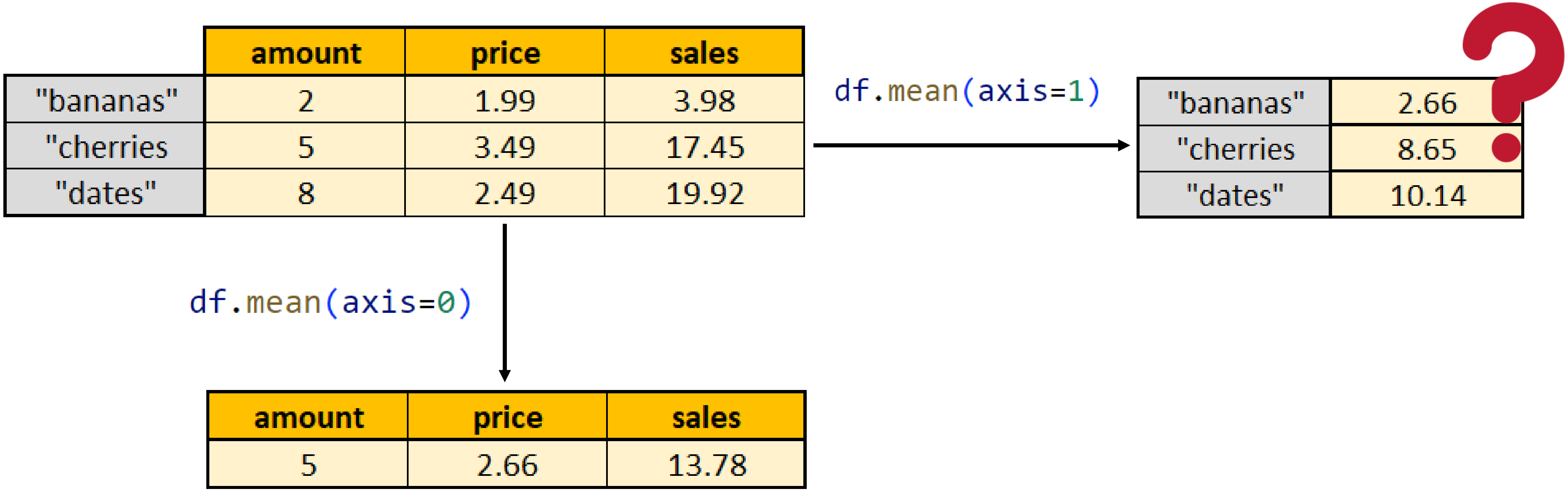


	fruit	date	amount	price
1	"bananas"	2024-03-01	2	1.99
3	"dates"	2024-03-01	8	2.49
0	"apples"	2024-03-01	3	2.99
2	"cherries	2024-03-01	5	3.49

```
df.price.mean()
```

Useful aggregation arguments

- **axis**: axis of the DataFrame along which the statistic is computed (default: 0)
- **numeric_only**: calculate statistics only for numeric columns (default: False)
- **skipna**: ignore NaN (missing values) (default: True)



	amount	price	sales
"bananas"	2	1.99	3.98
"cherries	5	3.49	17.45
"dates"	8	2.49	19.92

```
df.mean(axis=1)
```

"bananas"	2.66
"cherries	8.65
"dates"	10.14

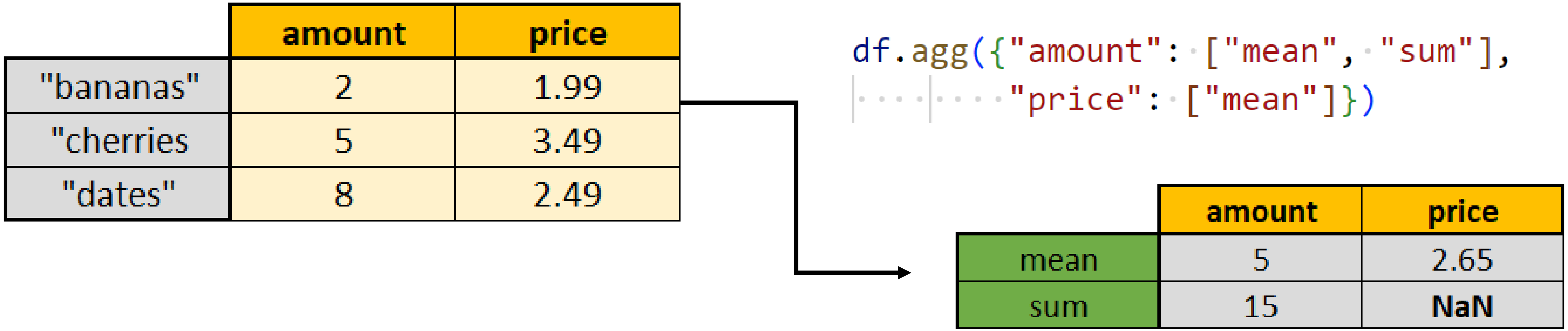
```
df.mean(axis=0)
```

amount	price	sales
5	2.66	13.78

agg method

The agg method is useful for more complex aggregation scenarios. It accepts:

- Custom aggregation function
- String function name: ‘mean’, ‘max’
- List of multiple aggregation functions: [np.mean, ‘max’]
- Dictionary with column <-> aggregation function mapping



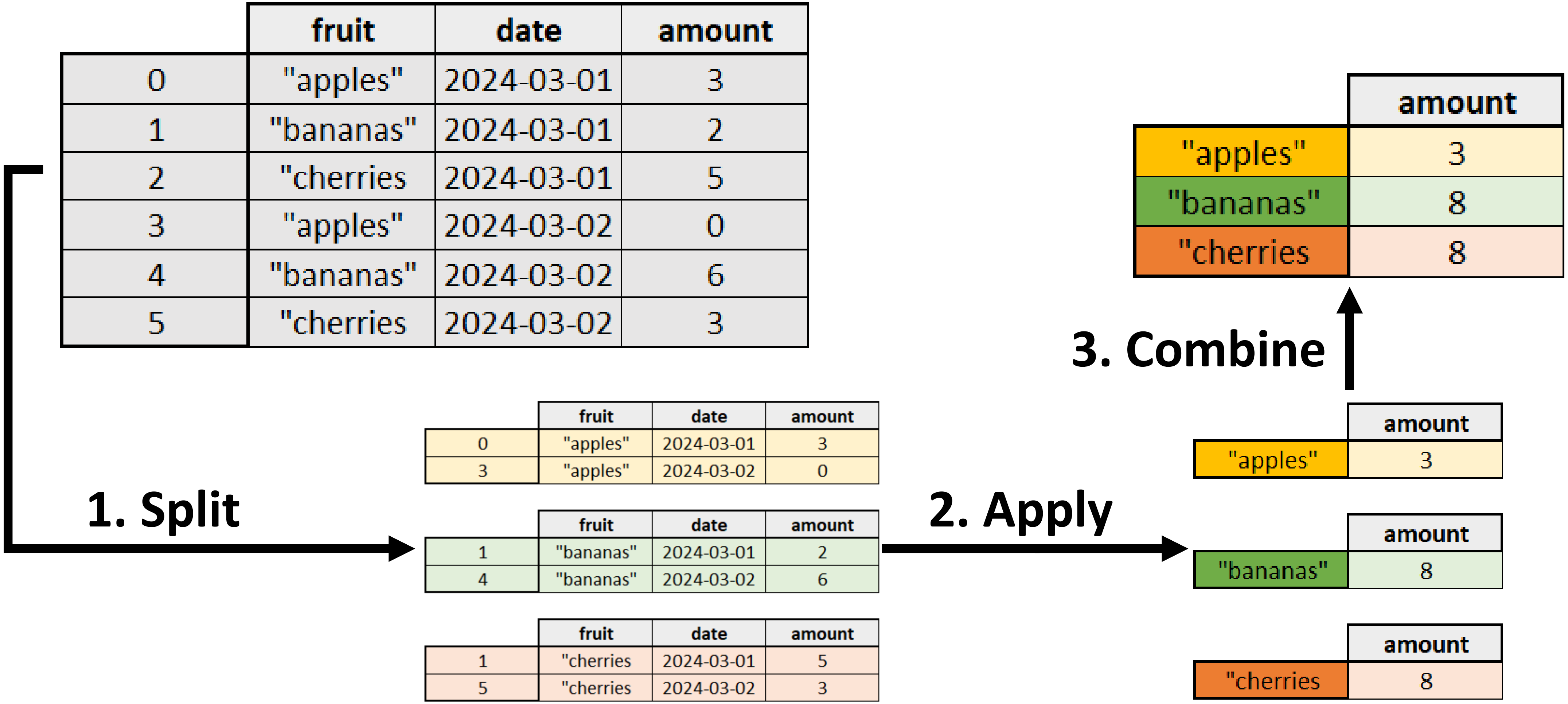
	amount	price
"bananas"	2	1.99
"cherries	5	3.49
"dates"	8	2.49

```
df.agg({"amount": ["mean", "sum"],
        "price": ["mean"]})
```

	amount	price
mean	5	2.65
sum	15	NaN

Grouped DataFrames

Grouped Aggregation: split - apply - combine



	fruit	date	amount
0	"apples"	2024-03-01	3
1	"bananas"	2024-03-01	2
2	"cherries	2024-03-01	5
3	"apples"	2024-03-02	0
4	"bananas"	2024-03-02	6
5	"cherries	2024-03-02	3

1. Split

	fruit	date	amount
0	"apples"	2024-03-01	3
3	"apples"	2024-03-02	0

	fruit	date	amount
1	"bananas"	2024-03-01	2
4	"bananas"	2024-03-02	6

	fruit	date	amount
1	"cherries	2024-03-01	5
5	"cherries	2024-03-02	3

2. Apply

	amount
"apples"	3

	amount
"bananas"	8

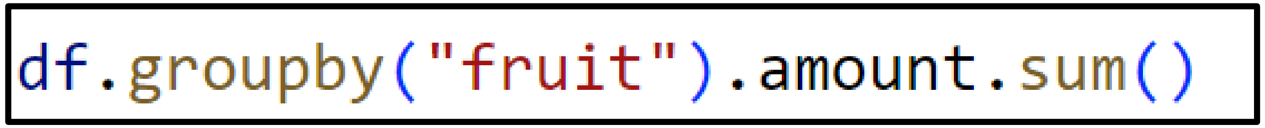
	amount
"cherries	8

3. Combine

	amount
"apples"	3
"bananas"	8
"cherries	8

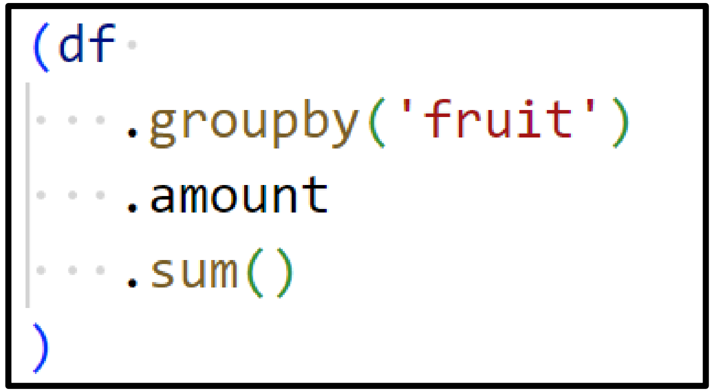
```
df.groupby("fruit").amount.sum()
```

Method chaining



```python
df.groupby("fruit").amount.sum()
```

- We <u>can</u> arrange a sequence of Pandas operations in a chain for the purpose of
  - Readability
  - Consisness
  - Maintainability
- Optionally, we can break long lines



```python
(df 
    .groupby('fruit')
    .amount
    .sum()
)
```