ScaDS.AI
DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS AND ARTIFICIAL INTELLIGENCE

Bio-Image Data Science Training

TRAINING: Tabular Data Wrangling

SPEAKER: Matthias Täschner

Including material from Robert Haase

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

SACHSEN Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

AGENDA

- pandas for tabular data
- DataFrame and Series
- Creating DataFrames
- Selecting from DataFrames
  - Label-location
  - Integer-location
  - Boolean indexing
- Combining DataFrames
- Handle Missing Data
- Tidy Data

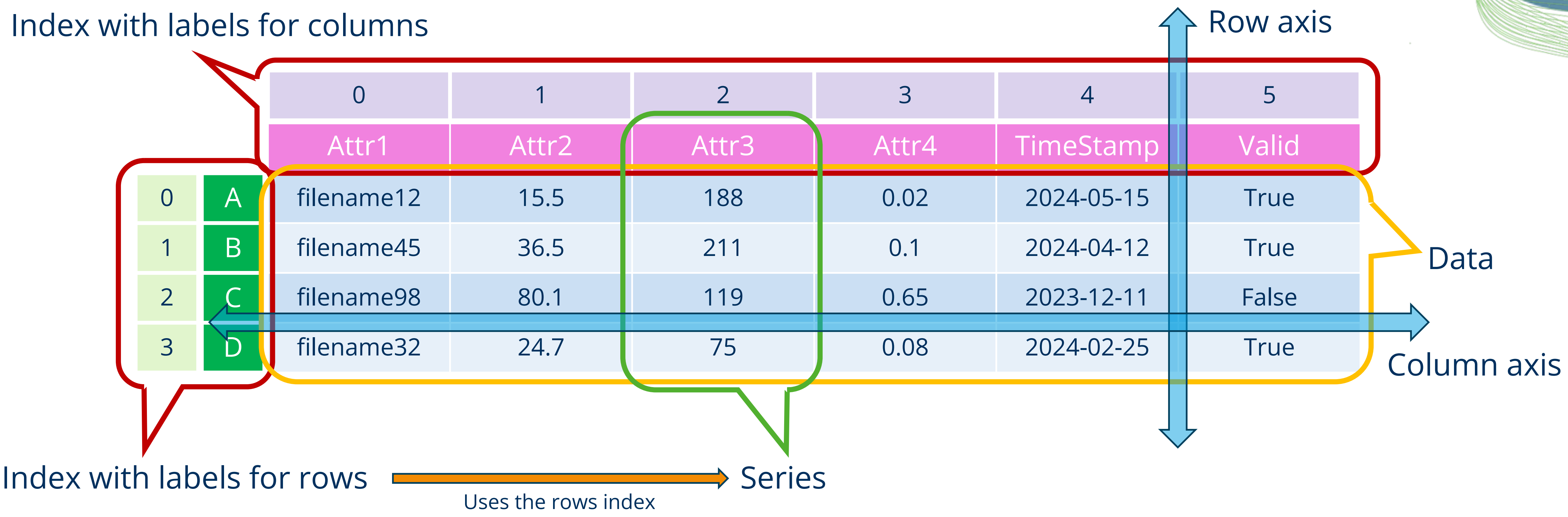
pandas for tabular data

pandas is an open-source Python library for data manipulation and analysis

- Development started in 2008, recent stable version is 2.2.2
- Offers data structures and operations tailored for tabular data and time series analysis
- Core structures
  - Series (1-dimensional labeled array with index, i.e., a one-column table)
  - DataFrame (2-dimensional data structure with labels and index)
- Built on top of
  - NumPy for performant numerical operations in Python
  - SciPy for scientific computations in Python
  - Matplotlib for data visualization in Python
- A good starting point is the 10 minutes to pandas tutorial

© pandas via NumFOCUS, Inc.
https://pandas.pydata.org/

DataFrame and Series



Index with labels for columns

Row axis

		0	1	2	3	4	5
		Attr1	Attr2	Attr3	Attr4	TimeStamp	Valid
0	A	filename12	15.5	188	0.02	2024-05-15	True
1	B	filename45	36.5	211	0.1	2024-04-12	True
2	C	filename98	80.1	119	0.65	2023-12-11	False
3	D	filename32	24.7	75	0.08	2024-02-25	True

Data

Column axis

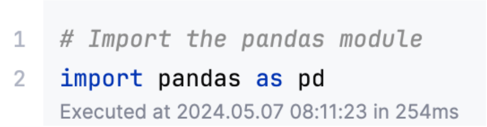
Index with labels for rows → Series

Uses the rows index

Creating DataFrames

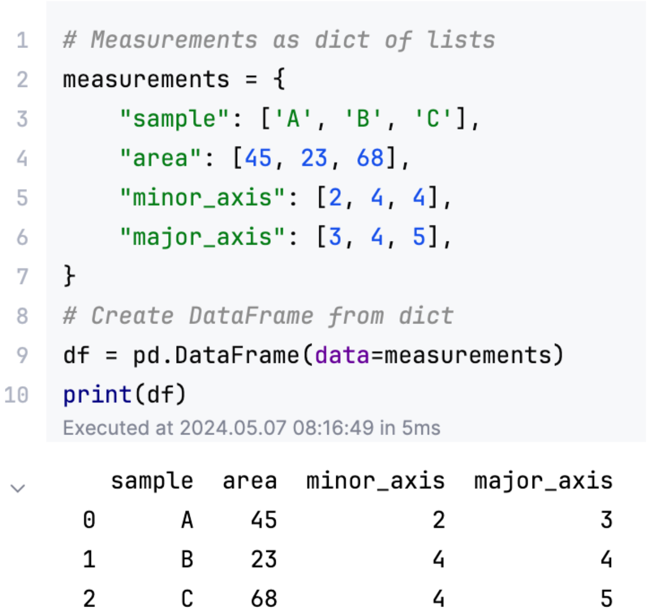
- DataFrames can be created from different inputs
- Let’s assume we have the following data available from measurements

sample	area	minor_ axis	major_ axis
A	45	2	3
B	23	4	4
C	68	4	4



```
# Import the pandas module
import pandas as pd
```

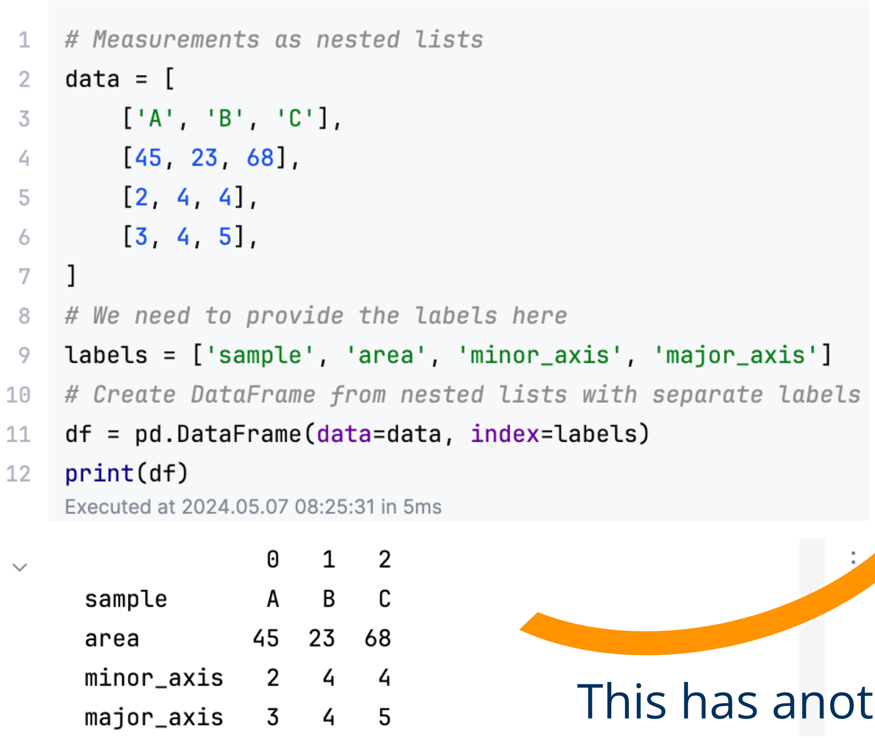
Executed at 2024.05.07 08:11:23 in 254ms



```
# Measurements as dict of lists
measurements = {
    "sample": ['A', 'B', 'C'],
    "area": [45, 23, 68],
    "minor_axis": [2, 4, 4],
    "major_axis": [3, 4, 5],
}
# Create DataFrame from dict
df = pd.DataFrame(data=measurements)
print(df)
```

Executed at 2024.05.07 08:16:49 in 5ms

```
  sample  area  minor_axis  major_axis
0      A    45           2           3
1      B    23           4           4
2      C    68           4           5
```

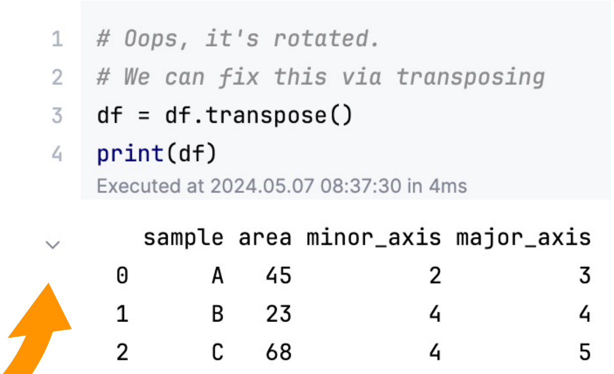


```
# Measurements as nested lists
data = [
    ['A', 'B', 'C'],
    [45, 23, 68],
    [2, 4, 4],
    [3, 4, 5],
]
# We need to provide the labels here
labels = ['sample', 'area', 'minor_axis', 'major_axis']
# Create DataFrame from nested lists with separate labels
df = pd.DataFrame(data=data, index=labels)
print(df)
```

Executed at 2024.05.07 08:25:31 in 5ms

```
             0   1   2
sample       A   B   C
area        45  23  68
minor_axis   2   4   4
major_axis   3   4   5
```

This has another format??



```
# Oops, it's rotated.
# We can fix this via transposing
df = df.transpose()
print(df)
```

Executed at 2024.05.07 08:37:30 in 4ms

```
  sample area minor_axis major_axis
0      A   45          2          3
1      B   23          4          4
2      C   68          4          5
```

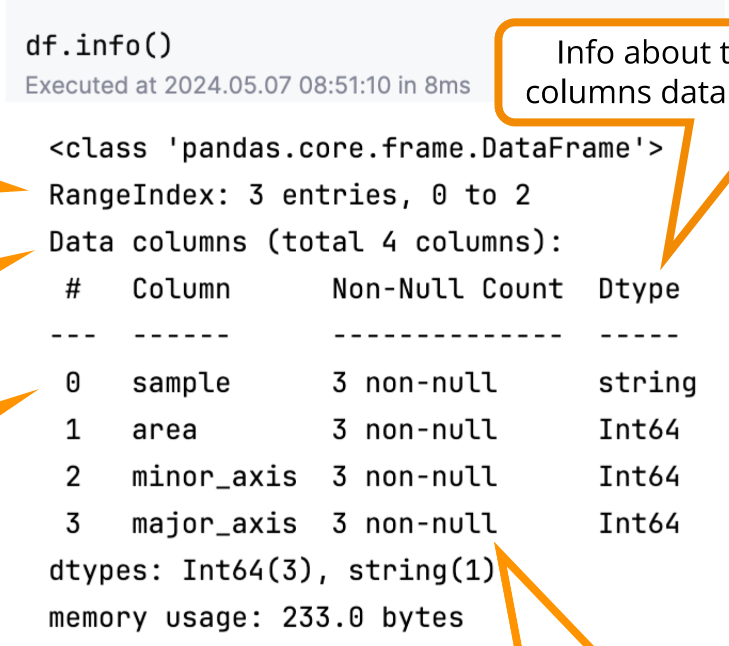
Creating DataFrames

- DataFrames provide some convenient methods to get an overview on the structure and data

Info about the rows index

Info about the columns

Info about the columns index



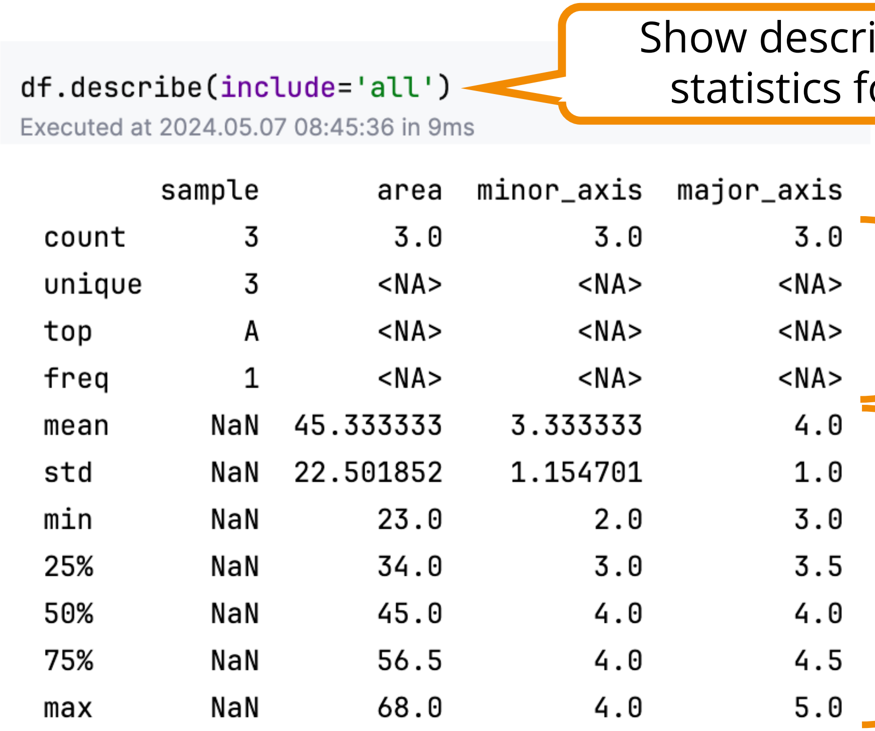
```
df.info()
Executed at 2024.05.07 08:51:10 in 8ms
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sample      3 non-null      string
 1   area        3 non-null      Int64
 2   minor_axis  3 non-null      Int64
 3   major_axis  3 non-null      Int64
dtypes: Int64(3), string(1)
memory usage: 233.0 bytes
```

Info about the columns data type

First info about missing data

Show descriptive statistics for…



```
df.describe(include='all')
Executed at 2024.05.07 08:45:36 in 9ms
```

```
       sample       area  minor_axis  major_axis
count       3        3.0         3.0         3.0
unique      3       <NA>        <NA>        <NA>
top         A       <NA>        <NA>        <NA>
freq        1       <NA>        <NA>        <NA>
mean      NaN  45.333333    3.333333         4.0
std       NaN  22.501852    1.154701         1.0
min       NaN       23.0         2.0         3.0
25%       NaN       34.0         3.0         3.5
50%       NaN       45.0         4.0         4.0
75%       NaN       56.5         4.0         4.5
max       NaN       68.0         4.0         5.0
```

…categorical data

…numerical data

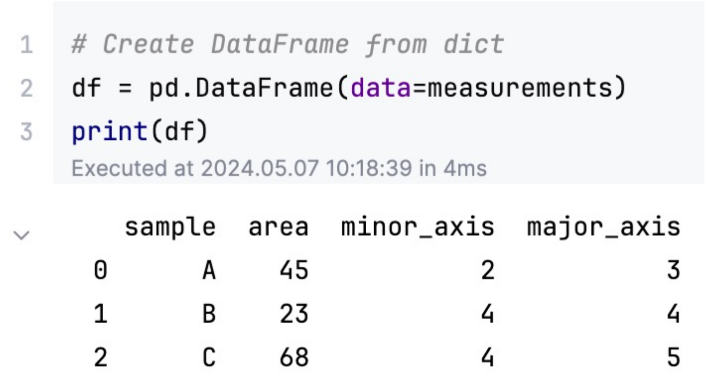
Selecting from DataFrames
Overview

There are different ways to select data from a DataFrame

Operation	Syntax	Result
Select one column	df[column label]	Series
Select several columns	df[list of column labels]	DataFrame
Select one row by label	df.loc[row label]	Series
Select several rows by label	df.loc[list of row labels]	DataFrame
Select rows by slicing on labels	df.loc[start row label : end row label]	DataFrame
Select one row by integer location	df.iloc[index number]	Series
Select several rows by integer location	df.iloc[list of index numbers]	DataFrame
Select rows by slicing on integer location	df.iloc[start index number : end index number]	DataFrame
Select rows and columns by label	df.loc[list of row labels , list of column labels]	DataFrame
… there are more possibilities		

Selecting from DataFrames
Label-location

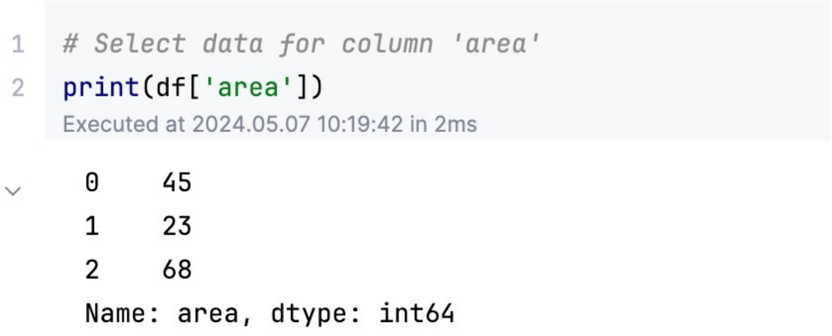
Here, we use the columns’ labels and rows’ labels to select data – [ ] and loc[ ]



```
# Create DataFrame from dict
df = pd.DataFrame(data=measurements)
print(df)
```

Executed at 2024.05.07 10:18:39 in 4ms

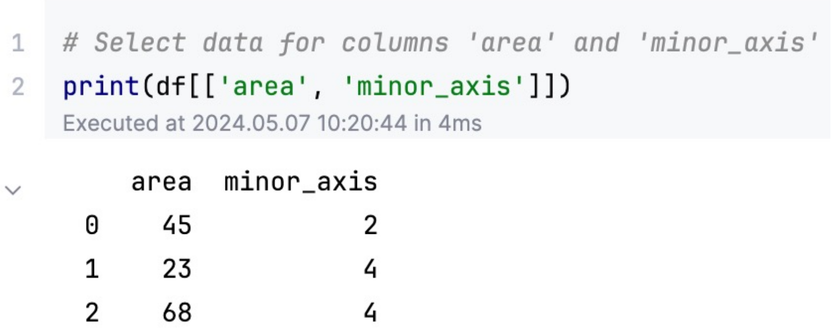
```
  sample  area  minor_axis  major_axis
0      A    45           2           3
1      B    23           4           4
2      C    68           4           5
```



```
# Select data for column 'area'
print(df['area'])
```

Executed at 2024.05.07 10:19:42 in 2ms

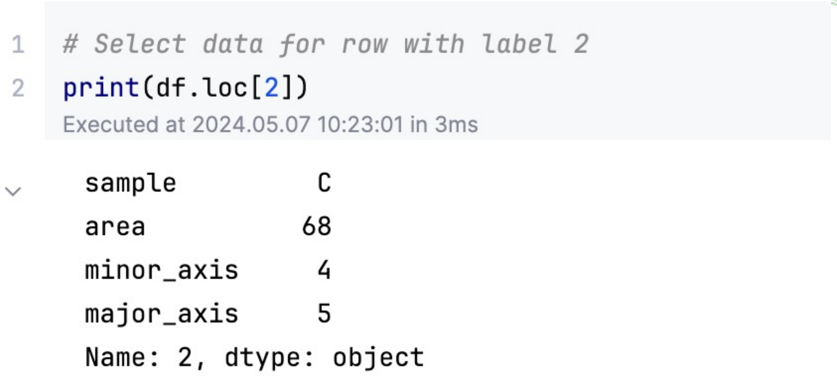
```
0    45
1    23
2    68
Name: area, dtype: int64
```



```
# Select data for columns 'area' and 'minor_axis'
print(df[['area', 'minor_axis']])
```

Executed at 2024.05.07 10:20:44 in 4ms

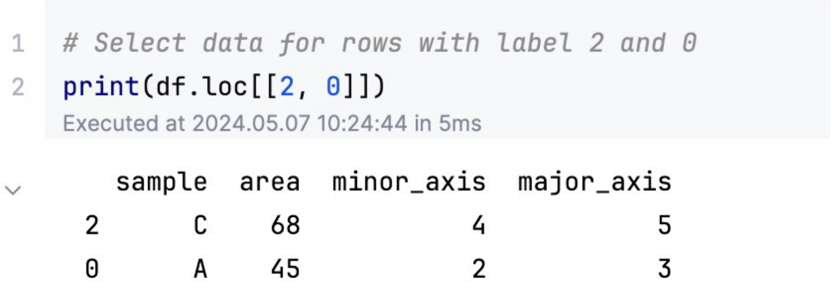
```
   area  minor_axis
0    45           2
1    23           4
2    68           4
```



```
# Select data for row with label 2
print(df.loc[2])
```

Executed at 2024.05.07 10:23:01 in 3ms

```
sample         C
area          68
minor_axis     4
major_axis     5
Name: 2, dtype: object
```



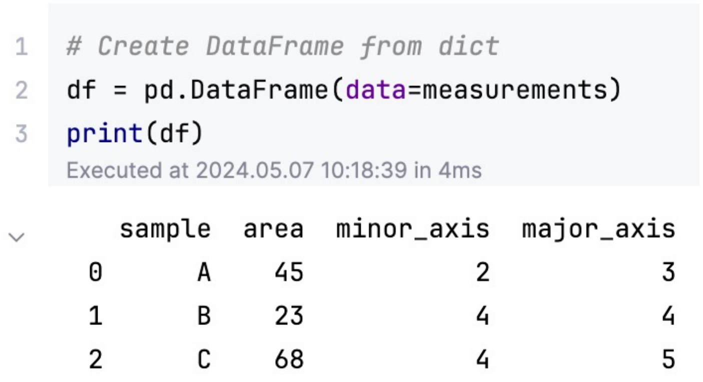
```
# Select data for rows with label 2 and 0
print(df.loc[[2, 0]])
```

Executed at 2024.05.07 10:24:44 in 5ms

```
  sample  area  minor_axis  major_axis
2      C    68           4           5
0      A    45           2           3
```

Selecting from DataFrames
Integer-location

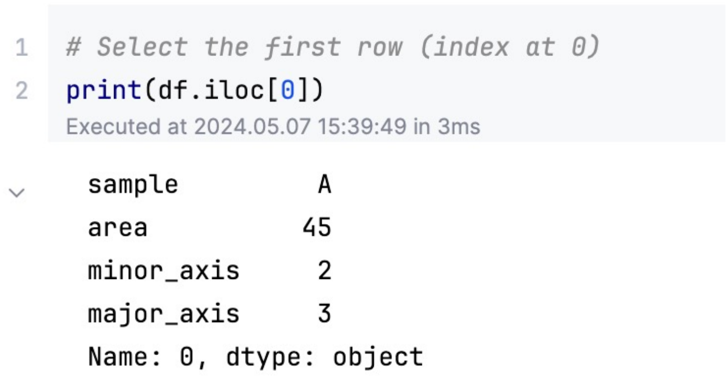
Here, we use the integer indices of the rows and columns to select data – iloc[ ]



```
# Create DataFrame from dict
df = pd.DataFrame(data=measurements)
print(df)
```

Executed at 2024.05.07 10:18:39 in 4ms

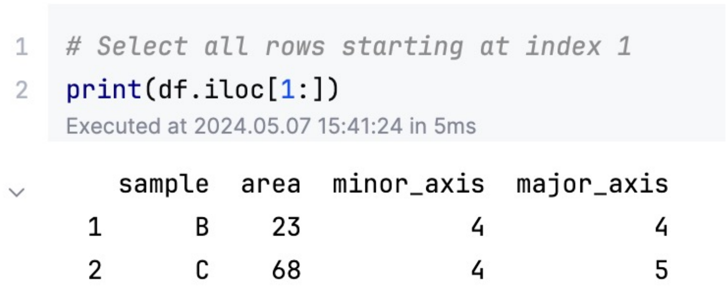
```
  sample  area  minor_axis  major_axis
0      A    45           2           3
1      B    23           4           4
2      C    68           4           5
```



```
# Select the first row (index at 0)
print(df.iloc[0])
```

Executed at 2024.05.07 15:39:49 in 3ms

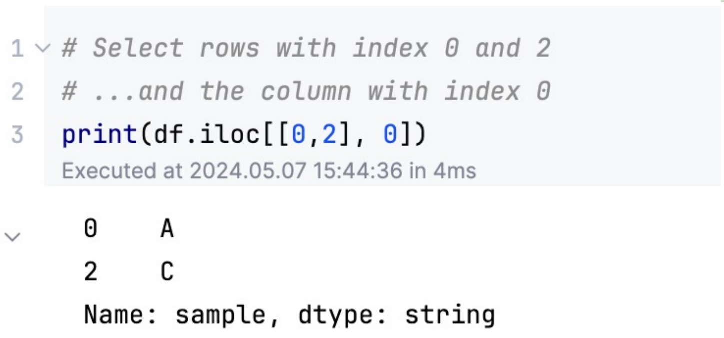
```
sample         A
area          45
minor_axis     2
major_axis     3
Name: 0, dtype: object
```



```
# Select all rows starting at index 1
print(df.iloc[1:])
```

Executed at 2024.05.07 15:41:24 in 5ms

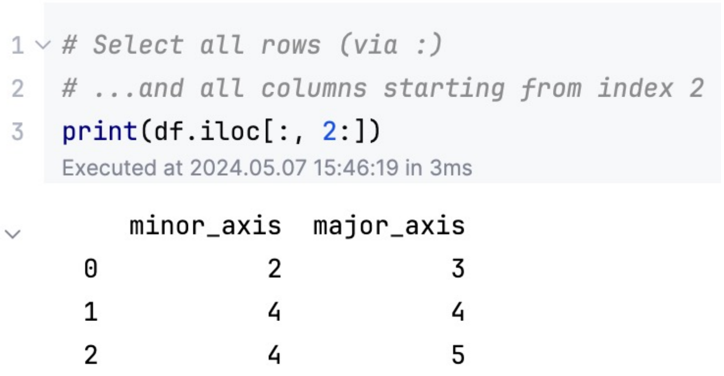
```
  sample  area  minor_axis  major_axis
1      B    23           4           4
2      C    68           4           5
```



```
# Select rows with index 0 and 2
# ...and the column with index 0
print(df.iloc[[0,2], 0])
```

Executed at 2024.05.07 15:44:36 in 4ms

```
0    A
2    C
Name: sample, dtype: string
```



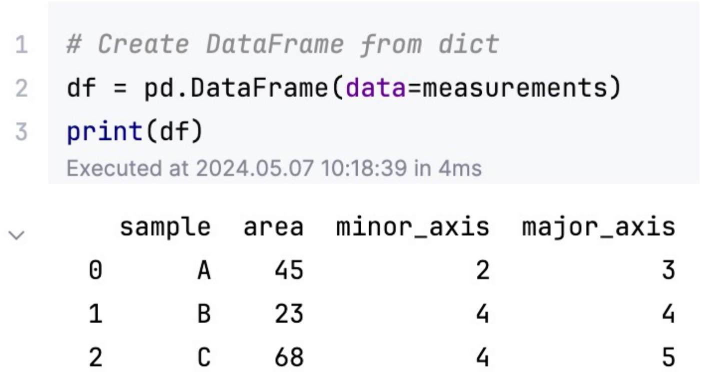
```
# Select all rows (via :)
# ...and all columns starting from index 2
print(df.iloc[:, 2:])
```

Executed at 2024.05.07 15:46:19 in 3ms

```
   minor_axis  major_axis
0           2           3
1           4           4
2           4           5
```

Selecting from DataFrames
Boolean indexing

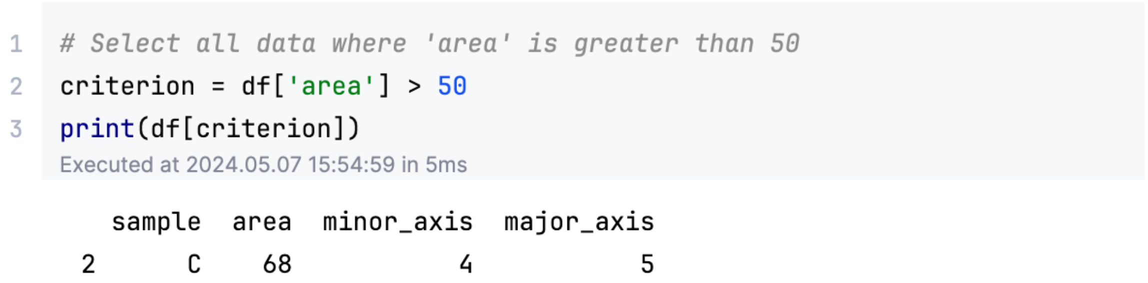
Here, we use conditional / logical queries (masking) to select data – Boolean vectors



```python
# Create DataFrame from dict
df = pd.DataFrame(data=measurements)
print(df)
```

Executed at 2024.05.07 10:18:39 in 4ms

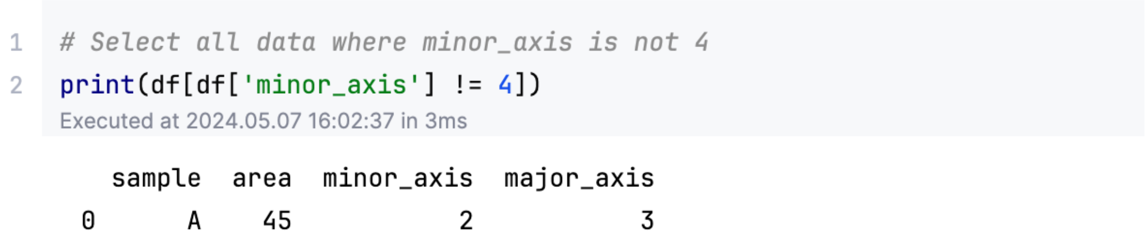
```
  sample  area  minor_axis  major_axis
0      A    45           2           3
1      B    23           4           4
2      C    68           4           5
```



```python
# Select all data where 'area' is greater than 50
criterion = df['area'] > 50
print(df[criterion])
```

Executed at 2024.05.07 15:54:59 in 5ms

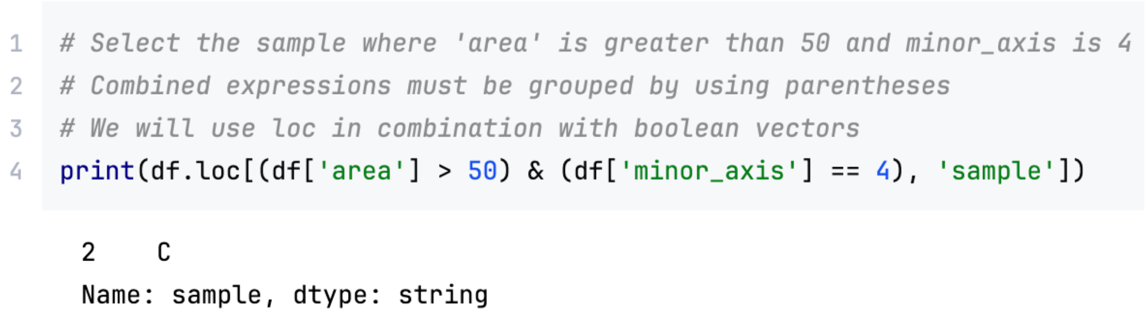
```
  sample  area  minor_axis  major_axis
2      C    68           4           5
```



```python
# Select all data where minor_axis is not 4
print(df[df['minor_axis'] != 4])
```

Executed at 2024.05.07 16:02:37 in 3ms

```
  sample  area  minor_axis  major_axis
0      A    45           2           3
```



```python
# Select the sample where 'area' is greater than 50 and minor_axis is 4
# Combined expressions must be grouped by using parentheses
# We will use loc in combination with boolean vectors
print(df.loc[(df['area'] > 50) & (df['minor_axis'] == 4), 'sample'])
```

```
2    C
Name: sample, dtype: string
```

Combining DataFrames

There are 3 main operations to combine DataFrames

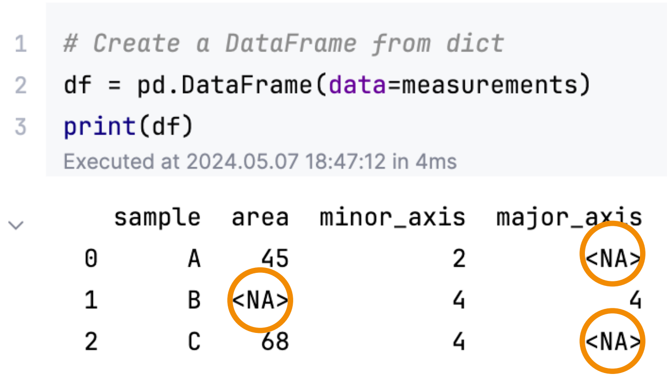
- concat combines an arbitrary number of pandas objects (DataFrames, Series)
  - pandas.concat([df1, df2])
- merge performs SQL-like combination on two pandas objects
  - pandas.merge(df1, df2) or df1.merge(df2)
- join is a DataFrame method to combine it with an arbitrary number of other pandas objects
  - df1.join(df2) or df1.join([df2, df3])

All operations provide parameters for further control, e.g.,

- Along which axis the combination is performed
- What kind of set logic (union or intersection) to use for combination
- On which column (index) a combination is performed
- …

Handle missing data

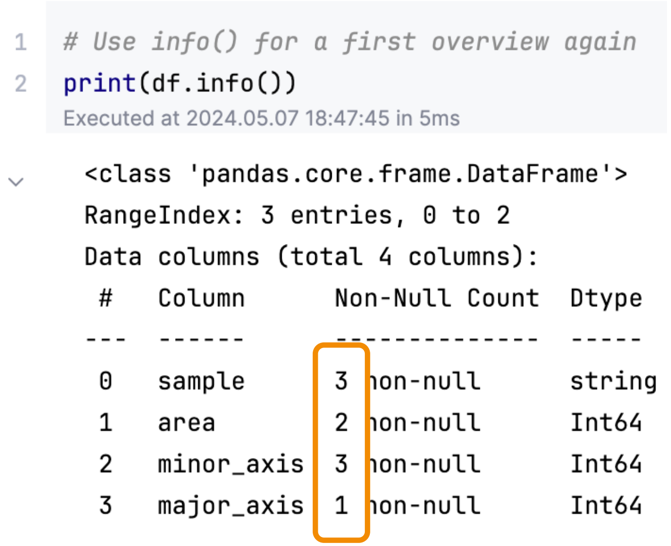
Often values are missing, and we need to detect and handle them



```
# Create a DataFrame from dict
df = pd.DataFrame(data=measurements)
print(df)
```

Executed at 2024.05.07 18:47:12 in 4ms

```
  sample  area  minor_axis  major_axis
0      A    45           2        <NA>
1      B  <NA>           4           4
2      C    68           4        <NA>
```

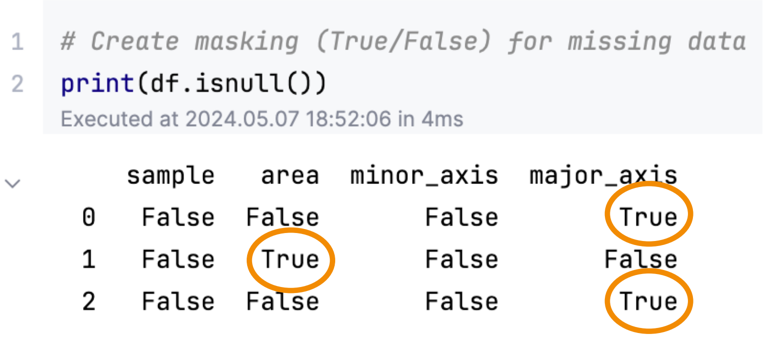


```
# Use info() for a first overview again
print(df.info())
```

Executed at 2024.05.07 18:47:45 in 5ms

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sample      3 non-null      string
 1   area        2 non-null      Int64
 2   minor_axis  3 non-null      Int64
 3   major_axis  1 non-null      Int64
```

pandas' isnull() provides a Boolean masking for missing values

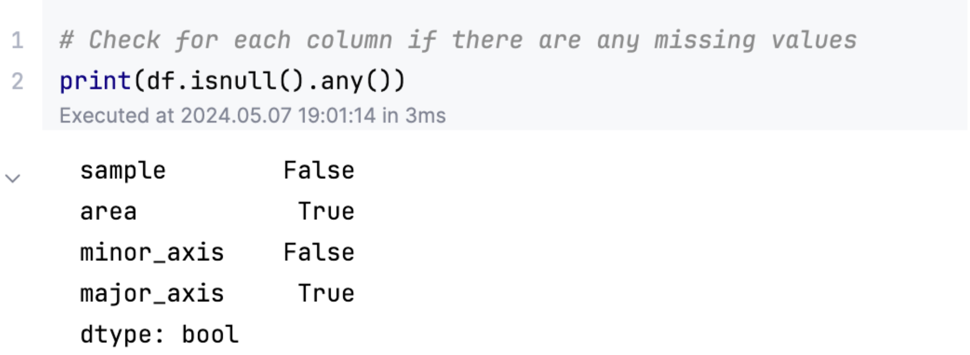


```
# Create masking (True/False) for missing data
print(df.isnull())
```

Executed at 2024.05.07 18:52:06 in 4ms

```
  sample   area  minor_axis  major_axis
0  False  False       False        True
1  False   True       False       False
2  False  False       False        True
```

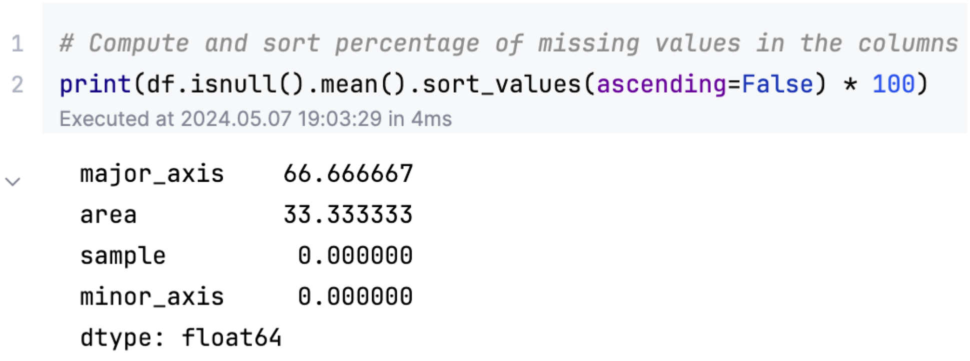
With this masking we can do further checks, and (since True==1 and False==0) even math



```
# Check for each column if there are any missing values
print(df.isnull().any())
```

Executed at 2024.05.07 19:01:14 in 3ms

```
sample        False
area           True
minor_axis    False
major_axis     True
dtype: bool
```



```
# Compute and sort percentage of missing values in the columns
print(df.isnull().mean().sort_values(ascending=False) * 100)
```

Executed at 2024.05.07 19:03:29 in 4ms

```
major_axis    66.666667
area          33.333333
sample         0.000000
minor_axis     0.000000
dtype: float64
```

Handle missing data

But what to do with the knowledge about missing values in the data…

- Ignore and go on with the analysis?
- Remove all samples with missing values from the data?
- Try to fill in the gaps (“data imputation”)?

It depends…
Open for further discussion

Tidy Data

In pandas, you *can* use hierarchical indices (MultiIndex) to “stack” data

		Month	January					
		Day in month	01			02		
		Measurement	Temp	Wind	Pressure	Temp	Wind	Pressure
Country	City	Station						
DE	Leipzig	DE102						
		DE205						
	Berlin	DE035						
		DE962						
GB	London	GB147						
		GB906						
	Bristol	GB781						
		GB006						

Looks nice for humans, but is tough to analyze

Tidy Data

For data analysis, “tidy data” works better:

- Each variable is a column
- Each observation is a row
- Each type of observation has its own DataFrame

Station	Temp	Wind	Pressure
DE102			
DE205			
DE035			
DE962			
GB147			
GB906			
GB781			
GB006			

Measurements for January 1st

Date	Temp	Wind	Pressure
2024-01-01			
2024-01-02			
2024-01-03			
2024-01-04			
2024-01-05			
2024-01-06			
2024-01-07			
2024-01-08			

Measurements for Station DE102

Any questions or remarks?

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

SACHSEN

Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.