Introduction to coding with python™

Workshop 3 – 22-09-2023

Swinda Falkena – s.k.j.falkena@uu.nl – BBG 604
Institute of Marine and Atmospheric Research

Last time

- NumPy, Matplotlib and SciPy

Today

- Importing data with Pandas, NetCDF4 and xarray datacamp

Python For Data Science
Importing Data Cheat Sheet

Learn Python online at www.DataCamp.com

Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Using the context manager with

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline()) #Read a single line
        print(file.readline())
        print(file.readline())
```

Table Data: Flat Files

Importing Flat Files with NumPy

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',', #String used to separate values
                      skiprows=2, #Skip the first 2 lines
                      usecols=[0,2], #Read the 1st and 3rd column
                      dtype=str) #The type of the resulting array
```

Files with mixed data type

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True, #Look for column header
                         dtype=None)
>>> data_array = np.recfromcsv(filename)
#The default dtype of the np.recfromcsv() function is None
```

Importing Flat Files with Pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5, #Number of rows of file to read
                       header=None, #Row number to use as col names
                       sep='\t', #Delimiter to use
                       comment='#', #Character to split comments
                       na_values=[""]) #String to recognize as NA/NaN
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype #Data type of array elements
>>> data_array.shape #Array dimensions
>>> len(data_array) #Length of array
```

Pandas DataFrames

```
>>> df.head() #Return first DataFrame rows
>>> df.tail() #Return last DataFrame rows
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> data_array = data.values #Convert a DataFrame to an a NumPy array
```

SAS File

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata File

```
>>> data = pd.read_stata('urbanpop.dta')
```

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                           'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager with

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Exploring Dictionaries

Querying relational databases with pandas

```
>>> print(mat.keys()) #Print dictionary keys
>>> for key in data.keys(): #Print dictionary keys
        print(key)
 meta
 quality
 strain
>>> pickled_data.values() #Return dictionary values
>>> print(mat.items()) #Returns items in list format of (key, value) tuple pairs
```

Accessing Data Items with Keys

```
 >>> for key in data ['meta'].keys() #Explore the HDF5
structure
        print(key)
  Description
  DescriptionURL
  Detector
  Duration
  GPSstart
  Observatory
  Type
  UTCstart
#Retrieve the value for a key
 >>> print(data['meta']['Description'].value)
```

Navigating Your FileSystem

Magic Commands

```
!ls  #List directory contents of files and directories
%cd .. #Change current working directory
%pwd #Return the current working directory path
```

OS Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd() #Store the name of current directory in a string
>>> os.listdir(wd) #Output contents of the directory in a list
>>> os.chdir(path) #Change current working directory
>>> os.rename("test1.txt", #Rename a file
              "test2.txt")
>>> os.remove("test1.txt") #Delete an existing file
>>> os.mkdir("newdir") #Create a new directory
```

Learn Data Skills Online at www.DataCamp.com

Importing data

- NumPy has some built-in functions for importing data
- But there are some powerful libraries

Importing data

- **Pandas** = importing, writing and processing tabular data (i.e. spreadsheets)

- **NetCDF4** = importing and writing NetCDF files

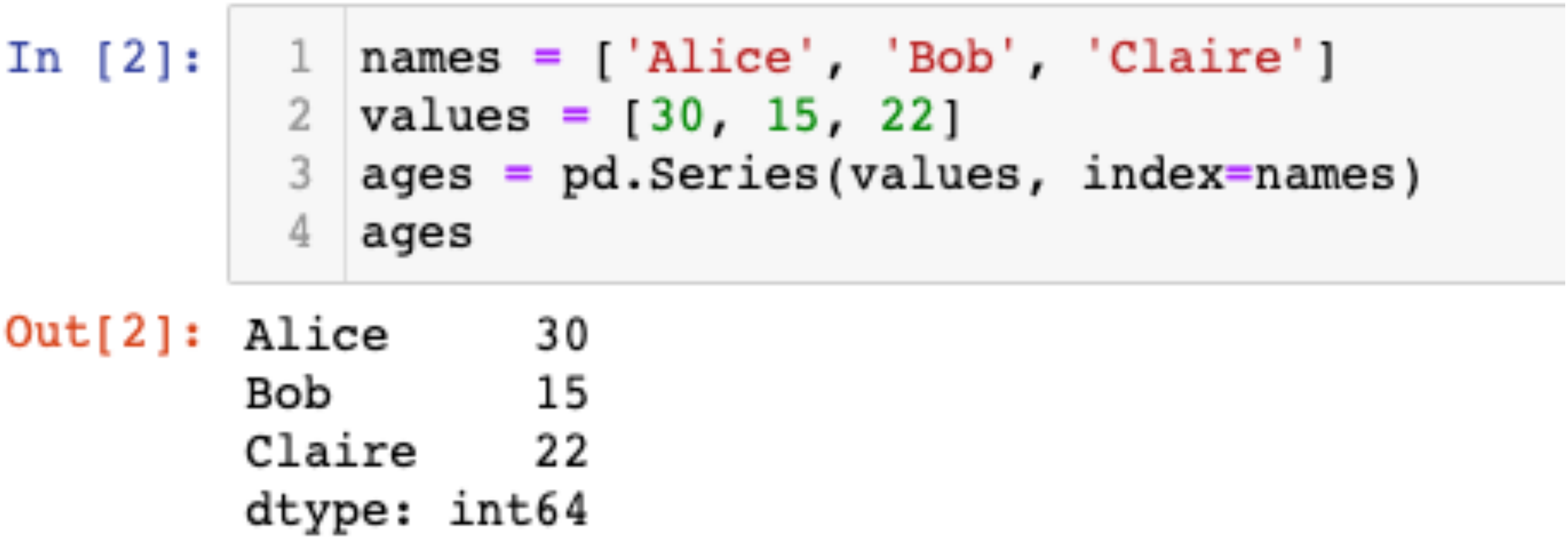
- **Xarray** = importing, writing and processing NetCDF files

Pandas

- **Series =** 1D labelled array

Pandas

- **Series =** 1D labelled array

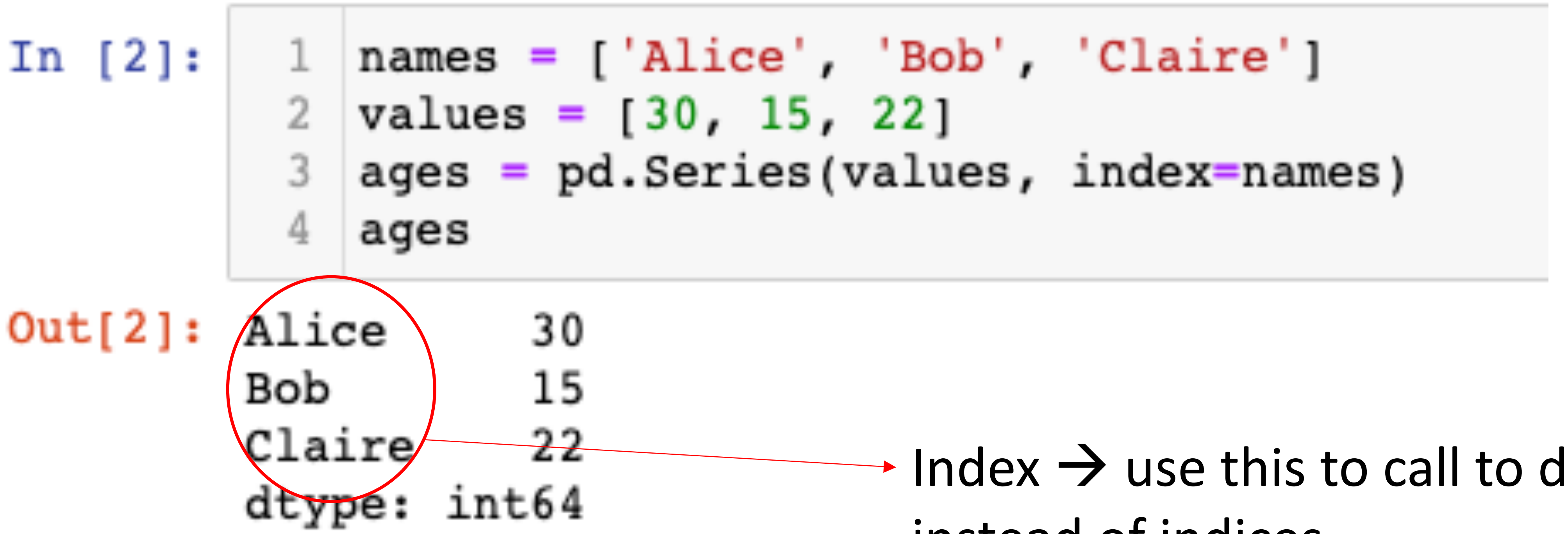


```
In [2]:  names = ['Alice', 'Bob', 'Claire']
         values = [30, 15, 22]
         ages = pd.Series(values, index=names)
         ages
```

```
Out[2]: Alice     30
        Bob       15
        Claire    22
        dtype: int64
```

Pandas

- **Series =** 1D labelled array



```python
names = ['Alice', 'Bob', 'Claire']
values = [30, 15, 22]
ages = pd.Series(values, index=names)
ages
```

```
Alice     30
Bob       15
Claire    22
dtype: int64
```

Index → use this to call to data instead of indices

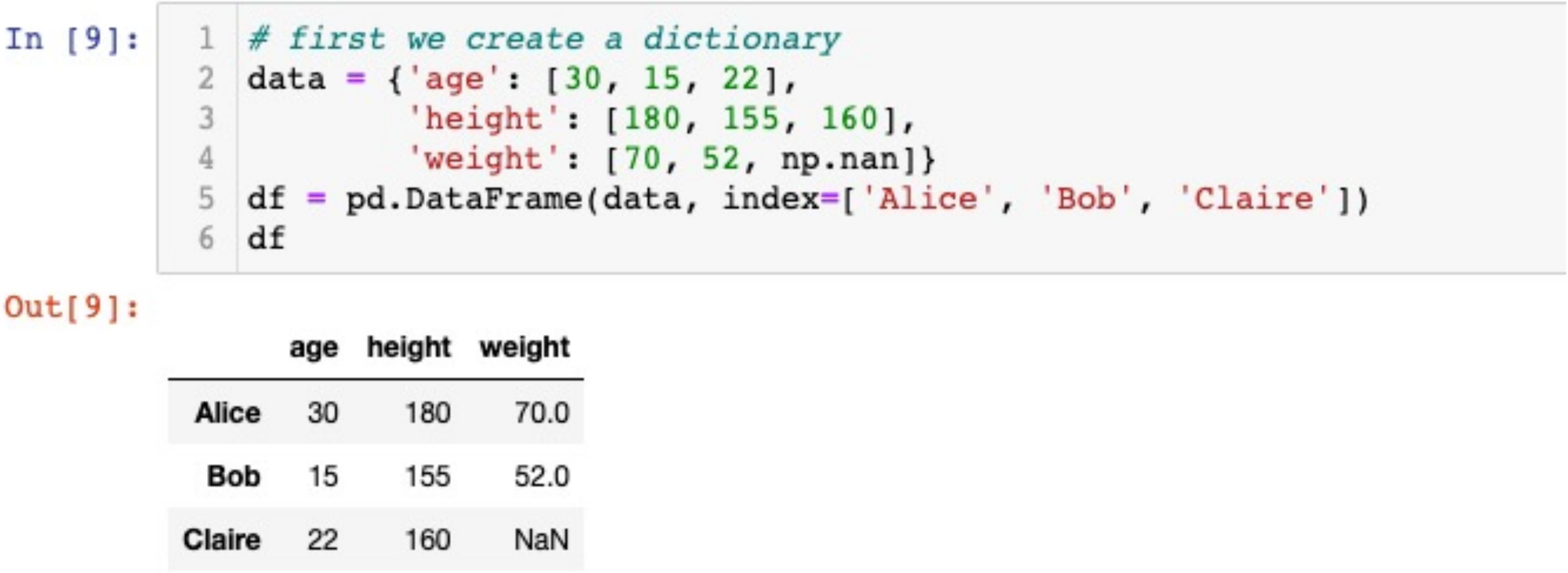
Pandas

- **Series =** 1D labelled array
- **DataFrame =** 2D labelled array (table)

Pandas

- **Series =** 1D labelled array

- **DataFrame =** 2D labelled array (table)



In [9]:

```
# first we create a dictionary
data = {'age': [30, 15, 22],
        'height': [180, 155, 160],
        'weight': [70, 52, np.nan]}
df = pd.DataFrame(data, index=['Alice', 'Bob', 'Claire'])
df
```

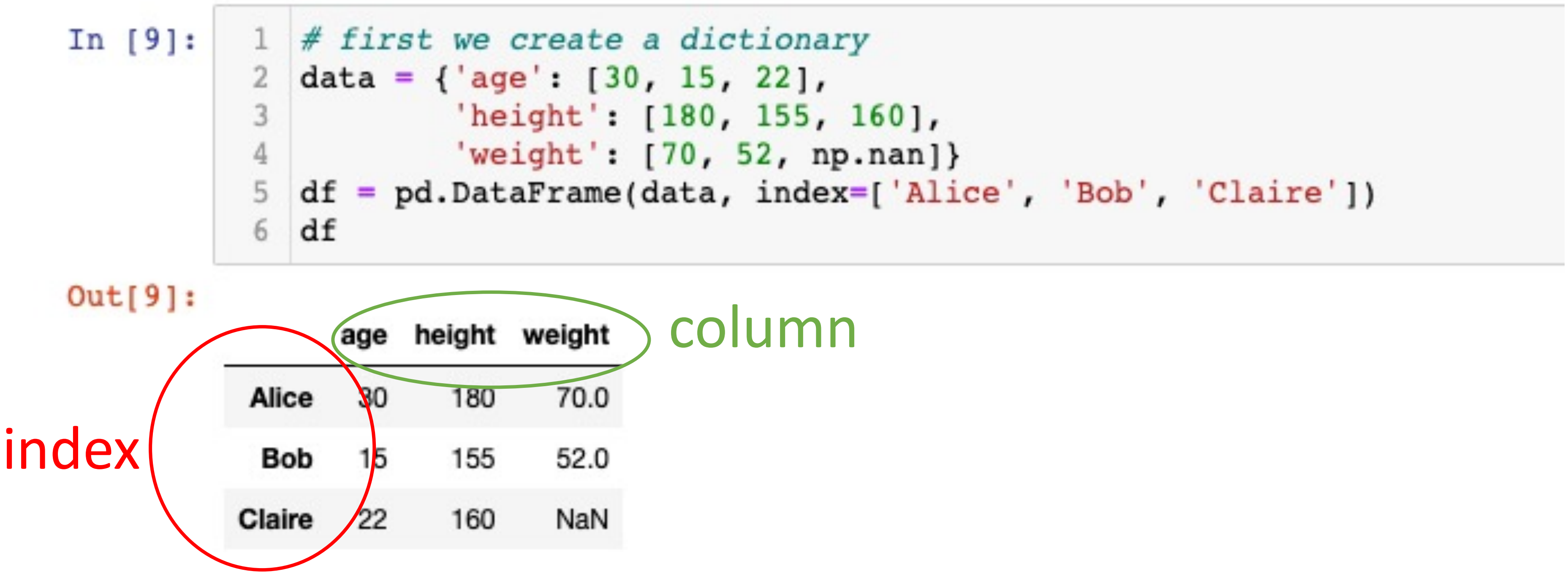
Out[9]:

	age	height	weight
Alice	30	180	70.0
Bob	15	155	52.0
Claire	22	160	NaN

Pandas

- **Series =** 1D labelled array

- **DataFrame =** 2D labelled array (table)



In [9]:

```python
# first we create a dictionary
data = {'age': [30, 15, 22],
        'height': [180, 155, 160],
        'weight': [70, 52, np.nan]}
df = pd.DataFrame(data, index=['Alice', 'Bob', 'Claire'])
df
```

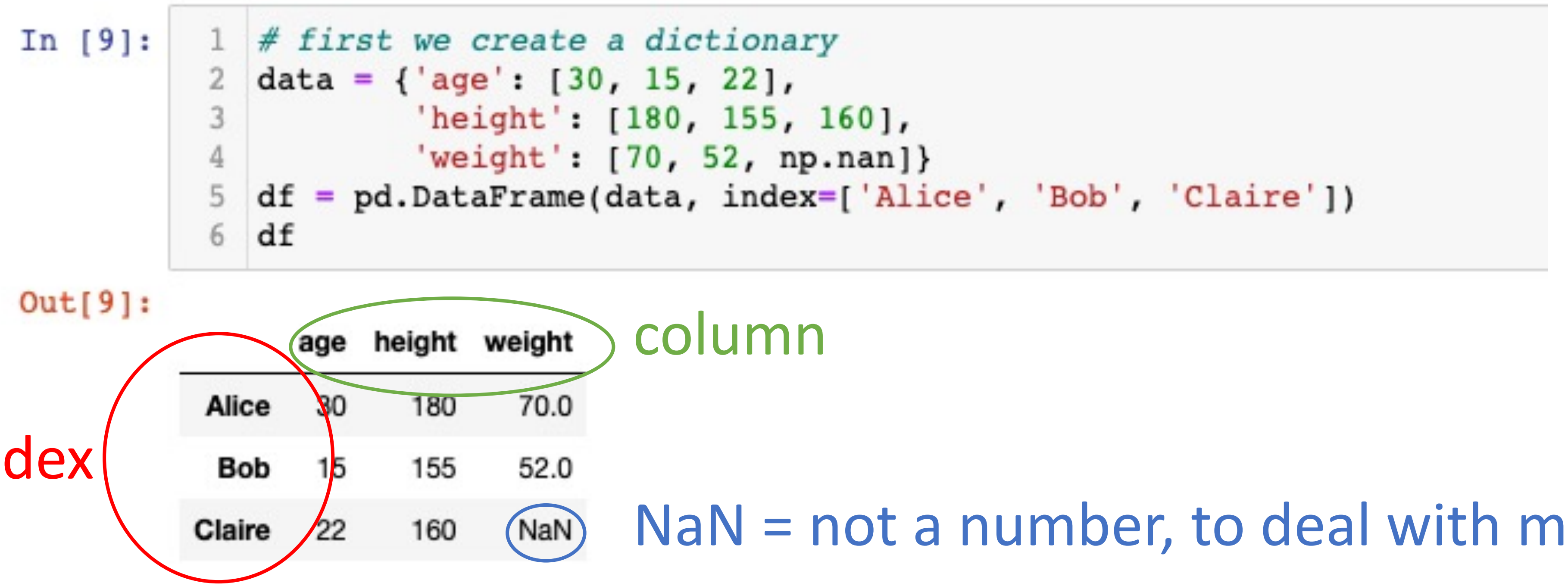
Out[9]:

```
        age  height  weight
Alice    30     180    70.0
Bob      15     155    52.0
Claire   22     160     NaN
```

Pandas

- **Series =** 1D labelled array

- **DataFrame =** 2D labelled array (table)



In [9]:

```python
# first we create a dictionary
data = {'age': [30, 15, 22],
        'height': [180, 155, 160],
        'weight': [70, 52, np.nan]}
df = pd.DataFrame(data, index=['Alice', 'Bob', 'Claire'])
df
```

Out[9]:

	age	height	weight
Alice	30	180	70.0
Bob	15	155	52.0
Claire	22	160	NaN

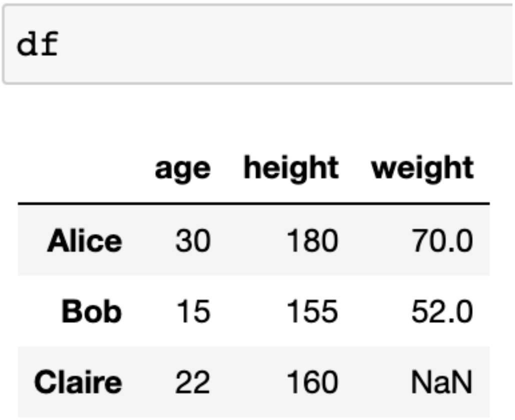
column

index

NaN = not a number, to deal with missing data

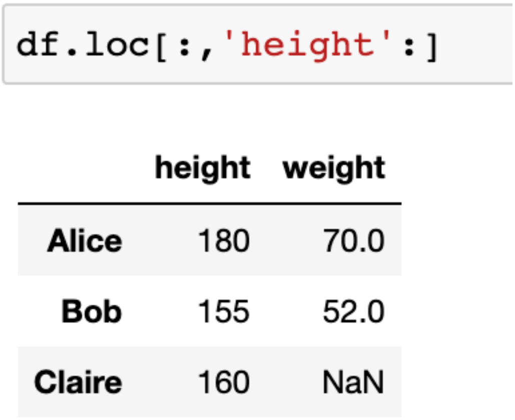
Pandas

- **Indexing:** 2 methods (`.loc[:,:]` `.iloc[:,:]`)



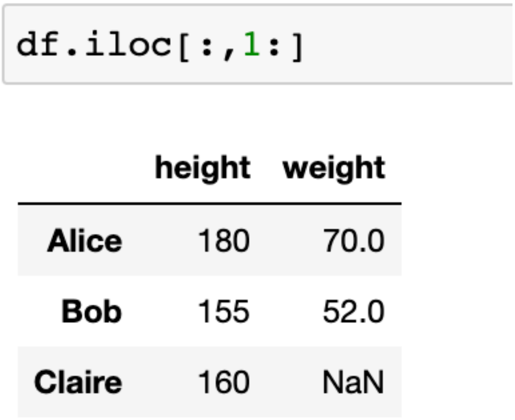
```
df
```

	age	height	weight
Alice	30	180	70.0
Bob	15	155	52.0
Claire	22	160	NaN



```
df.loc[:,'height':]
```

	height	weight
Alice	180	70.0
Bob	155	52.0
Claire	160	NaN

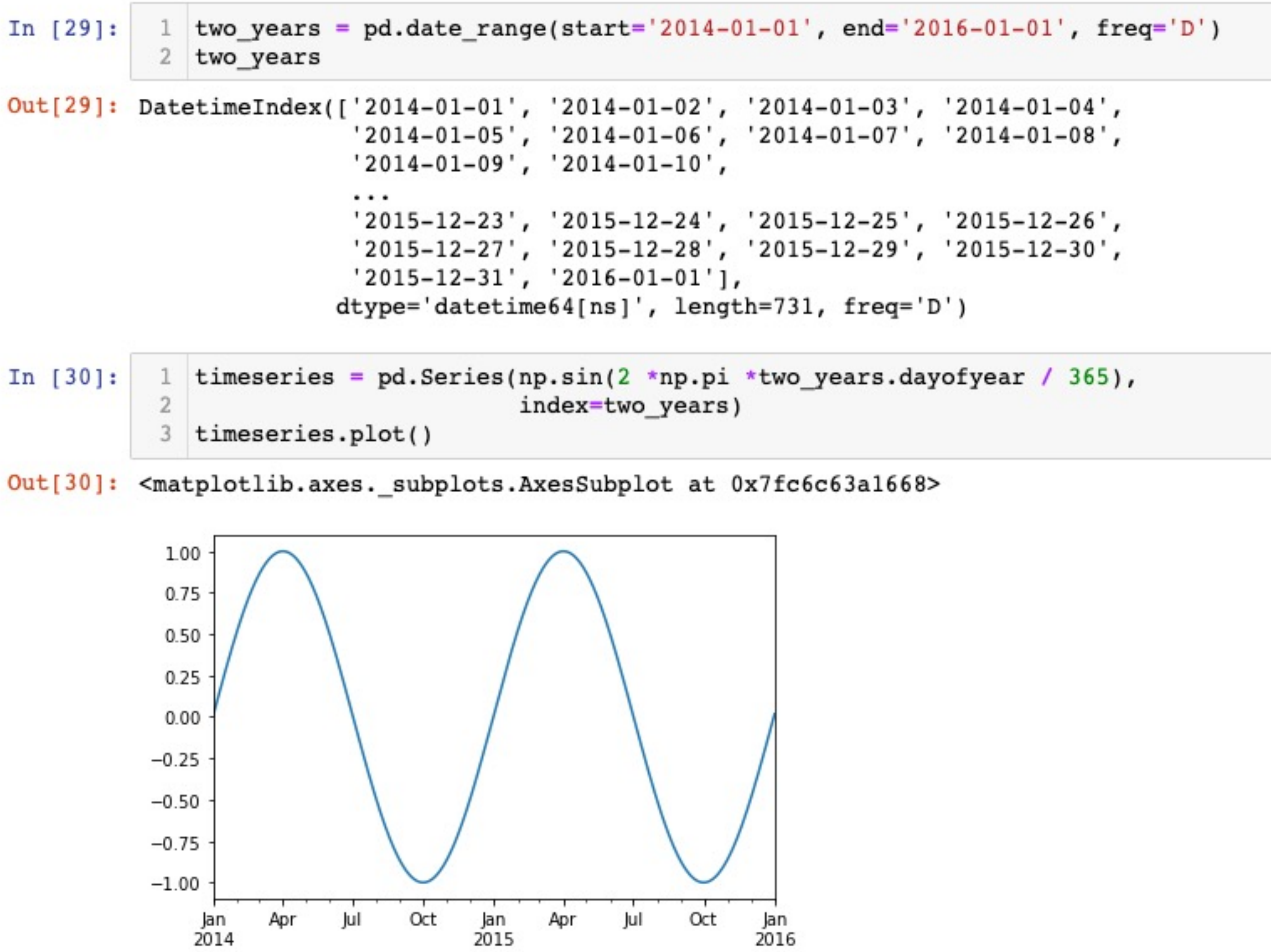


```
df.iloc[:,1:]
```

	height	weight
Alice	180	70.0
Bob	155	52.0
Claire	160	NaN

Pandas

- **Timearrays**



```
In [29]:  two_years = pd.date_range(start='2014-01-01', end='2016-01-01', freq='D')
          two_years

Out[29]: DatetimeIndex(['2014-01-01', '2014-01-02', '2014-01-03', '2014-01-04',
                        '2014-01-05', '2014-01-06', '2014-01-07', '2014-01-08',
                        '2014-01-09', '2014-01-10',
                        ...
                        '2015-12-23', '2015-12-24', '2015-12-25', '2015-12-26',
                        '2015-12-27', '2015-12-28', '2015-12-29', '2015-12-30',
                        '2015-12-31', '2016-01-01'],
                       dtype='datetime64[ns]', length=731, freq='D')
```

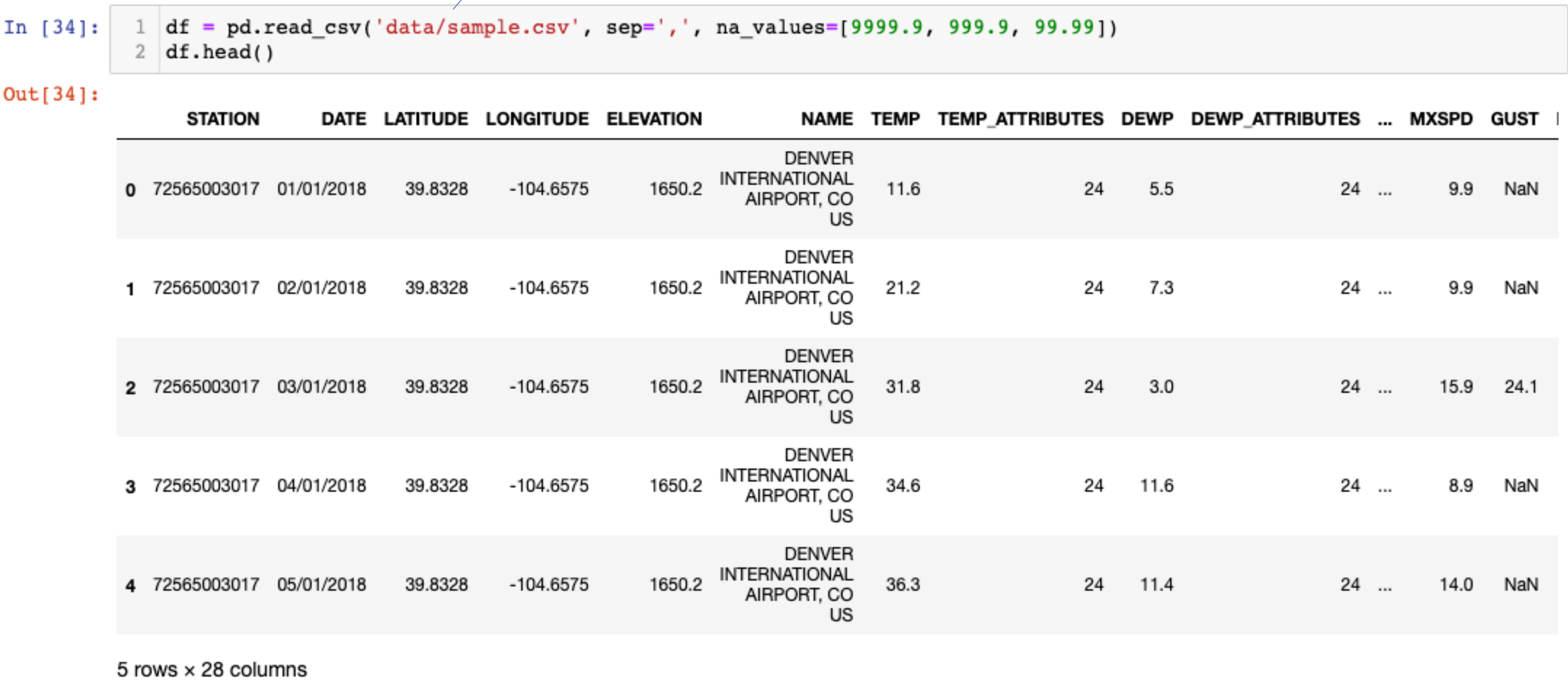
```
In [30]:  timeseries = pd.Series(np.sin(2 *np.pi *two_years.dayofyear / 365),
                                 index=two_years)
          timeseries.plot()

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc6c63a1668>
```

Pandas

Data in same folder as notebook? -> provide name

Otherwise give the path to the datafile



In [34]:

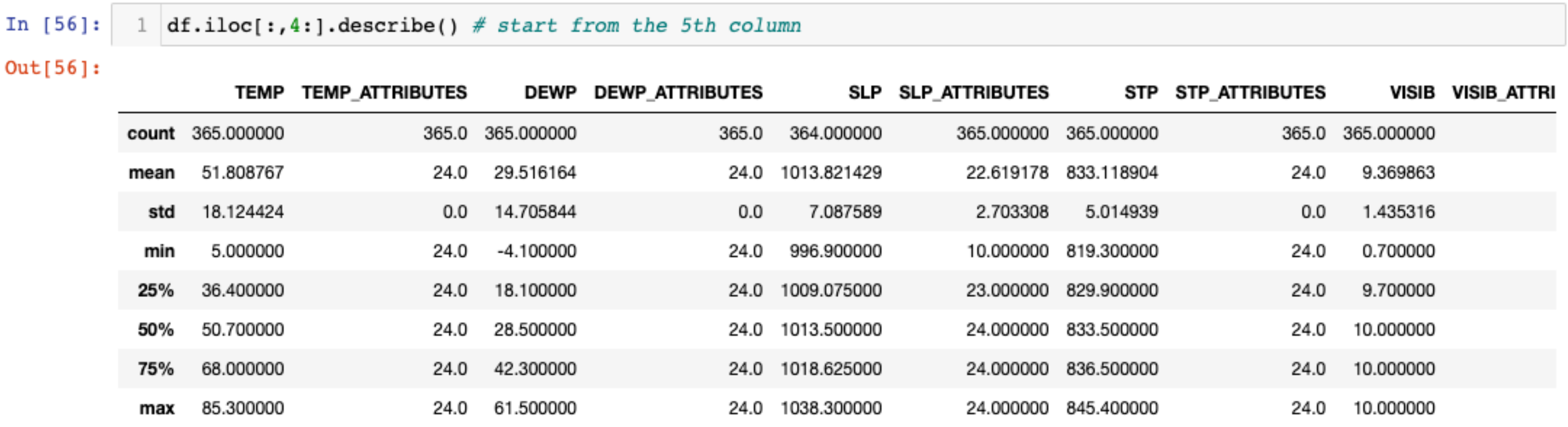
```
df = pd.read_csv('data/sample.csv', sep=',', na_values=[9999.9, 999.9, 99.99])
df.head()
```

Out[34]:

	STATION	DATE	LATITUDE	LONGITUDE	ELEVATION	NAME	TEMP	TEMP_ATTRIBUTES	DEWP	DEWP_ATTRIBUTES	...	MXSPD	GUST
0	72565003017	01/01/2018	39.8328	-104.6575	1650.2	DENVER INTERNATIONAL AIRPORT, CO US	11.6	24	5.5	24	...	9.9	NaN
1	72565003017	02/01/2018	39.8328	-104.6575	1650.2	DENVER INTERNATIONAL AIRPORT, CO US	21.2	24	7.3	24	...	9.9	NaN
2	72565003017	03/01/2018	39.8328	-104.6575	1650.2	DENVER INTERNATIONAL AIRPORT, CO US	31.8	24	3.0	24	...	15.9	24.1
3	72565003017	04/01/2018	39.8328	-104.6575	1650.2	DENVER INTERNATIONAL AIRPORT, CO US	34.6	24	11.6	24	...	8.9	NaN
4	72565003017	05/01/2018	39.8328	-104.6575	1650.2	DENVER INTERNATIONAL AIRPORT, CO US	36.3	24	11.4	24	...	14.0	NaN

5 rows × 28 columns

Pandas



In [56]:

```
df.iloc[:,4:].describe() # start from the 5th column
```

Out[56]:

	TEMP	TEMP_ATTRIBUTES	DEWP	DEWP_ATTRIBUTES	SLP	SLP_ATTRIBUTES	STP	STP_ATTRIBUTES	VISIB	VISIB_ATTRI
count	365.000000	365.0	365.000000	365.0	364.000000	365.000000	365.000000	365.0	365.000000	
mean	51.808767	24.0	29.516164	24.0	1013.821429	22.619178	833.118904	24.0	9.369863	
std	18.124424	0.0	14.705844	0.0	7.087589	2.703308	5.014939	0.0	1.435316	
min	5.000000	24.0	-4.100000	24.0	996.900000	10.000000	819.300000	24.0	0.700000	
25%	36.400000	24.0	18.100000	24.0	1009.075000	23.000000	829.900000	24.0	9.700000	
50%	50.700000	24.0	28.500000	24.0	1013.500000	24.000000	833.500000	24.0	10.000000	
75%	68.000000	24.0	42.300000	24.0	1018.625000	24.000000	836.500000	24.0	10.000000	
max	85.300000	24.0	61.500000	24.0	1038.300000	24.000000	845.400000	24.0	10.000000	

Pandas

- Tutorial: examples of built-in statistics and plotting and practice with weather station data from NOAA
  Workshop 3a – pandas.ipynb

NetCDF4

- **NetCDF** = - efficient storage of multidimensional data
  - accessible with (almost) all programming languages
  - machine-independent (portable)
  - easy access of a subset
  - description of data included

NetCDF4

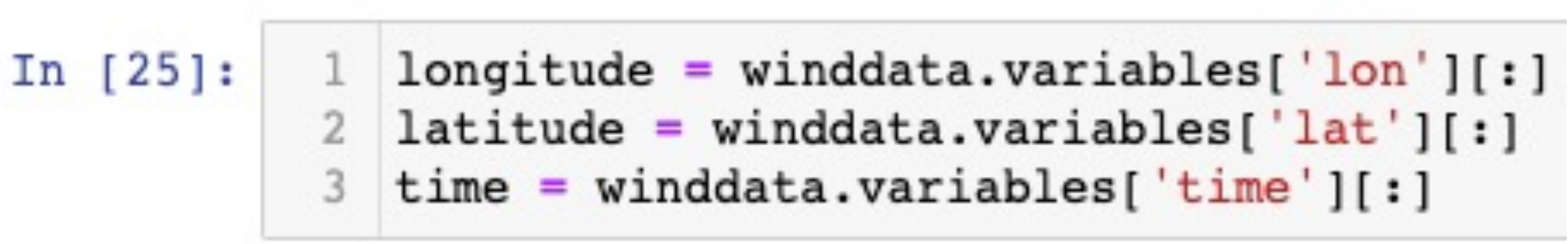
- Import data with: nc.Dataset(‘netcdf file’)

NetCDF4

- Import data with: nc.Dataset(‘netcdf file’)
- Data.variables

NetCDF4

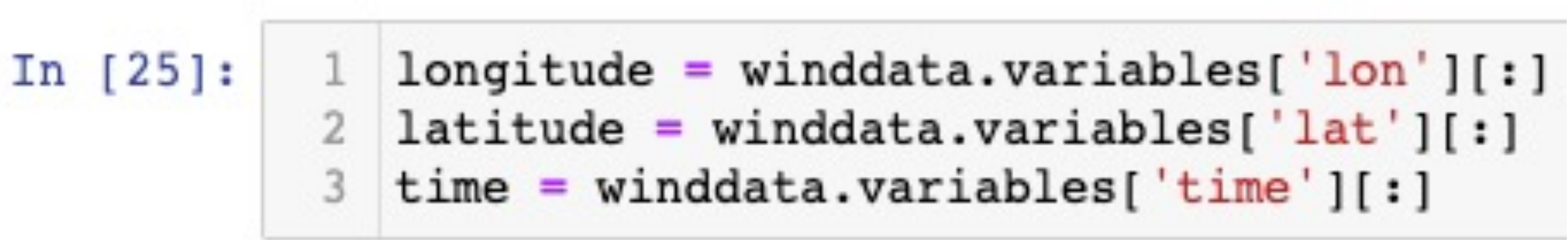
- Import data with: nc.Dataset(‘netcdf file’)
- Data.variables



```
longitude = winddata.variables['lon'][:]
latitude = winddata.variables['lat'][:]
time = winddata.variables['time'][:]
```

NetCDF4

- Import data with: nc.Dataset(‘netcdf file’)
- Data.variables



```
longitude = winddata.variables['lon'][:]
latitude = winddata.variables['lat'][:]
time = winddata.variables['time'][:]
```

Tutorial: practice with importing and interpreting output of wind model of the KNMI
Workshop 3b – netcdf4.ipynb

xarray

- Higher level interface for dealing with NetCDF files

xarray

- Higher level interface for dealing with NetCDF files
- Similar to pandas, i.e. label bases indexing, built-in functions for analysis and plotting

xarray

- Higher level interface for dealing with NetCDF files
- Similar to pandas, i.e. label bases indexing, built-in functions for analysis and plotting
- Two structures:
  - **DataArray** = 1 variable field, e.g. temperature(lon,lat,z,t)
  - **DataSet** = multiple DataArrays, that may share coordinates, e.g. temperature(lon,lat,z,t) and salinity(lon,lat,z,t)

xarray

- For each variable, the dimensions are labelled:

  In NumPy, the dimensions were just 0,1,2, …
  here you can give them names, e.g., lon, lat, time

xarray

- For each variable, the dimensions are labelled:

  In NumPy, the dimensions were just 0,1,2, …
  here you can give them names, e.g., lon, lat, time

Easy to extract variables at certain positions or time and perform computations on them

- Tutorial: practice with temperature data from NASA
  Workshop 3c – xarray.ipynb