

# Data Science

## Chapter 3: Python for Data Science

# Chapter outline

## This chapter covers

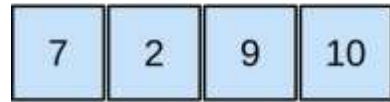
- Python: numPy library
- Data acquisition & preparation
- Python: Pandas library

# Numpy Library

- Numerical Python, or "Numpy" for short, is a foundational package on which many of the most common data science packages are built. Numpy provides us with high performance multi-dimensional arrays which we can use as vectors or matrices.
- The key features of numpy are:
  - ndarrays: n-dimensional arrays of the same data type which are fast and space-efficient. There are a number of built-in methods for ndarrays which allow for rapid processing of data without using loops (e.g., compute the mean).
  - Broadcasting: a useful tool which defines implicit behavior between multi-dimensional arrays of different sizes.
  - Vectorization: enables numeric operations on ndarrays.

# ndarrays

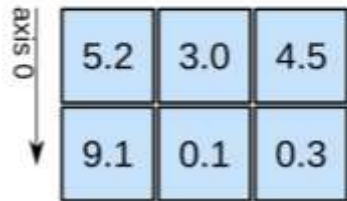
## 1D array



axis 0 →

shape: (4,)

## 2D array



axis 1 →

shape: (2, 3)

```
1 import numpy as np
2 #define a 2d np array
3 an_array = np.array([[11,12,13],[21,22,23]]) # Create a rank 2 array
4
5 print(an_array) # print the array
6 print("The shape is 2 rows, 3 columns: ", an_array.shape) # rows x columns
7 print("Accessing elements [0,0], [0,1], and [1,0] of the ndarray: ",
8       an_array[0, 0], ", ", an_array[0, 1], ", ", an_array[1, 0])
```

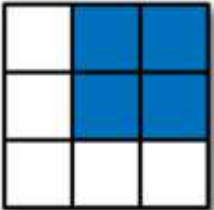
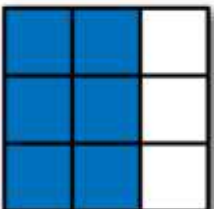
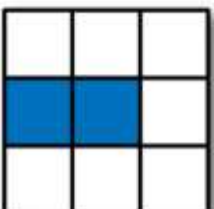
```
[[11 12 13]
```

```
 [21 22 23]]
```

```
The shape is 2 rows, 3 columns: (2, 3)
```

```
Accessing elements [0,0], [0,1], and [1,0] of the ndarray: 11 , 12 , 21
```

# Slice indexing

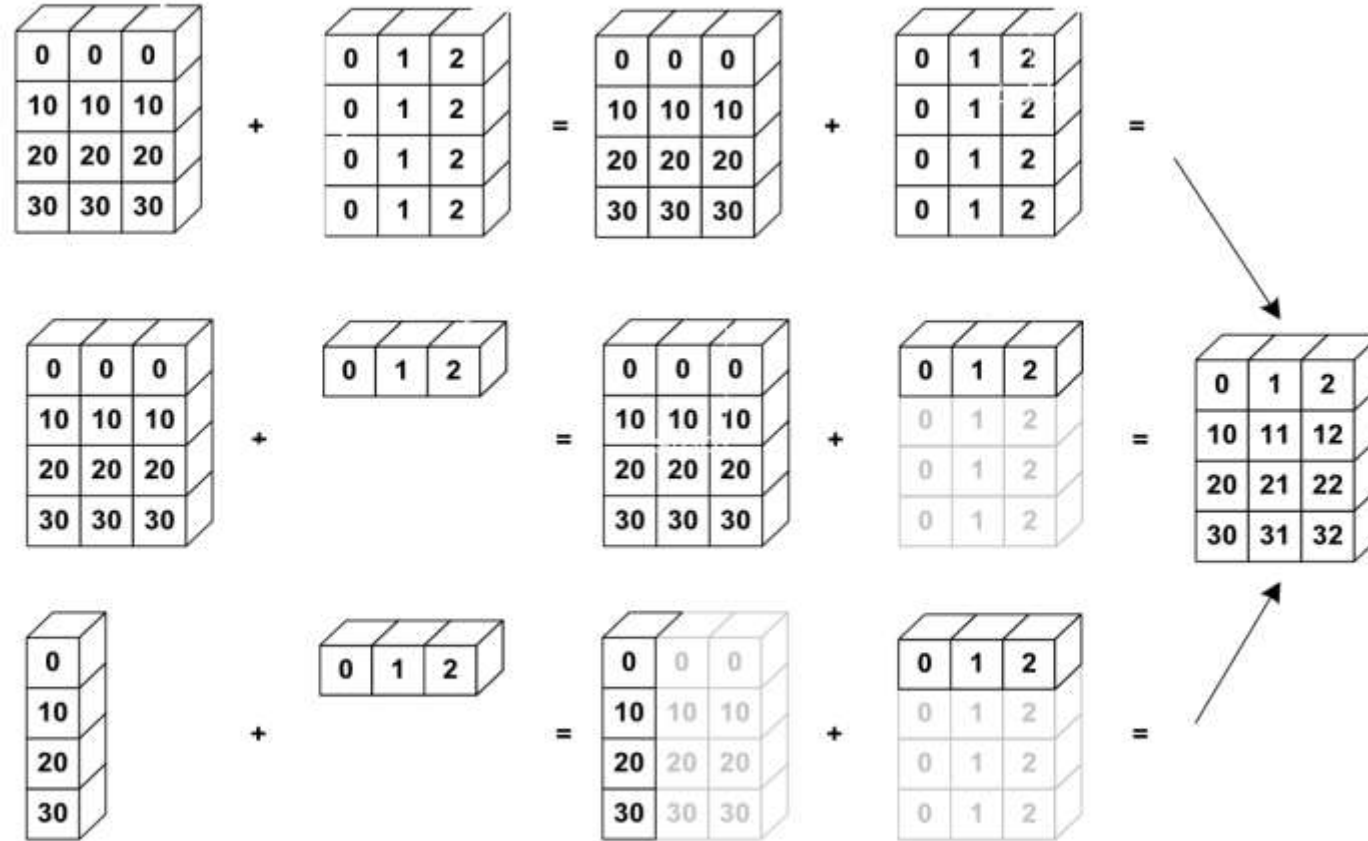
|                                                                                     | Expression                                                               | Shape                                                         |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------|---------------------------------------------------------------|
|    | <code>arr[:2, 1:]</code>                                                 | <code>(2, 2)</code>                                           |
|    | <code>arr[2]</code><br><code>arr[2, :]</code><br><code>arr[2:, :]</code> | <code>(3,)</code><br><code>(3,)</code><br><code>(1, 3)</code> |
|   | <code>arr[:, :2]</code>                                                  | <code>(3, 2)</code>                                           |
|  | <code>arr[1, :2]</code><br><code>arr[1:2, :2]</code>                     | <code>(2,)</code><br><code>(1, 2)</code>                      |

# Boolean indexing / filters

```
1 # create a 3x2 array
2 an_array = np.array([[11,12], [21, 22], [31, 32]])
3 print(an_array)
4
5 print(an_array[an_array>15])
6 print(an_array[an_array % 2 == 0])
7
8 an_array[an_array % 2 == 0] +=100#elets changed
9 print(an_array)
```

```
[[11 12]
 [21 22]
 [31 32]]
[21 22 31 32]
[12 22 32]
[[ 11 112]
 [ 21 122]
 [ 31 132]]
```

# Broadcasting



# Pandas Library

- Pandas is a Python library for data analysis. It offers a number of data exploration, cleaning and transformation operations that are critical in working with data in Python.
- pandas build upon numpy and scipy providing easy-to-use data structures and data manipulation functions with integrated indexing.
  - The main data structures pandas provides are Series and DataFrames.
  - The key features of pandas are:
    - Generating descriptive statistics on data
    - Data cleaning using built in pandas functions
    - Frequent data operations for subsetting, filtering, insertion, deletion and aggregation of data
    - Merging multiple datasets using dataframes
    - Working with timestamps and time-series data

# Pandas data structures

## pandas Series

- A 1-dimensional labeled array
- Supports many data types
- Axis labels -> index
  - Get and set values by index label

## Pandas DataFrames

- A 2-dimensional labeled data structure
- Columns can be of potentially different types
- Optionally parameters for fine-tuning:
  - index (row labels)
  - columns (column labels)

# Pandas Series

```
1 ser = pd.Series([100, 'foo', 300, 'bar', 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])
2 print(ser)
```

```
tom      100
bob      foo
nancy    300
dan      bar
eric     500
dtype: object
```

```
1 print(ser.index)
2
3 print(ser.loc['nancy']) #print(ser.iloc[2])
4
5 print(ser.loc[['nancy', 'bob']]) #print(ser.iloc[[2,1]])
```

```
Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')
```

```
300
```

```
nancy    300
bob      foo
dtype: object
```

# Pandas Dataframe

➤ Full documentation:

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>

The diagram illustrates the structure of a Pandas DataFrame. At the top, the word "Columns" is written in blue, with three arrows pointing down to the column headers: "Name", "Team", and "Number". Below the table, the word "Rows" is written in brown, with three arrows pointing to the row indices: 0, 1, and 2. At the bottom, the word "Data" is written in purple, with a bracket pointing to the data cells of the first three rows (rows 0, 1, and 2). The table itself has six columns: "Name", "Team", "Number", "Position", and "Age". The rows are indexed from 0 to 6. The data is as follows:

|   | <i>Name</i>     | <i>Team</i>    | <i>Number</i> | <i>Position</i> | <i>Age</i> |
|---|-----------------|----------------|---------------|-----------------|------------|
| 0 | Avery Bradley   | Boston Celtics | 0.0           | PG              | 25.0       |
| 1 | John Holland    | Boston Celtics | 30.0          | SG              | 27.0       |
| 2 | Jonas Jerebko   | Boston Celtics | 8.0           | PF              | 29.0       |
| 3 | Jordan Mickey   | Boston Celtics | NaN           | PF              | 21.0       |
| 4 | Terry Rozier    | Boston Celtics | 12.0          | PG              | 22.0       |
| 5 | Jared Sullinger | Boston Celtics | 7.0           | C               | NaN        |
| 6 | Evan Turner     | Boston Celtics | 11.0          | SG              | 27.0       |

# Acquire data

- `read_csv`
  - Input : Path to a Comma Separated File
  - Output: Pandas DataFrame object containing contents of the file
- `read_json`
  - Input : Path to a JSON file or a valid JSON String
  - Output: Pandas DataFrame or a Series object containing the contents
- `read_html` , `read_sql_query`, `read_sql_table`

# Explore data

➤ Describe(), corr(), min(), max, mean(), mode(), median(), std()

```
1 dataf = pd.read_csv('Data.csv')
```

```
1 dataf.describe()  
2 # for a column: dataf['Age'].describe()
```

|       | Age       | Salary       |
|-------|-----------|--------------|
| count | 9.000000  | 9.000000     |
| mean  | 38.777778 | 63777.777778 |
| std   | 7.693793  | 12265.579662 |
| min   | 27.000000 | 48000.000000 |
| 25%   | 35.000000 | 54000.000000 |
| 50%   | 38.000000 | 61000.000000 |
| 75%   | 44.000000 | 72000.000000 |
| max   | 50.000000 | 83000.000000 |

```
1 dataf.corr()
```

|        | Age      | Salary   |
|--------|----------|----------|
| Age    | 1.000000 | 0.982495 |
| Salary | 0.982495 | 1.000000 |

# Explore data

## ➤ any()

- Output: Returns whether ANY element is True
- Benefits: Can detect if a cell matches a condition very quickly

```
1 filter1 = dataf['Salary'] < 60000
```

```
1 filter1.any()
```

True

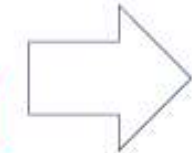
## ➤ all()

- Output: Returns whether ALL element is True
- Benefits: Can detect if a column or row matches a condition very quickly

# Clean data

## ➤ Replace

|   | 0           | 1           |
|---|-------------|-------------|
| 0 | -0.349596   | -2.017159   |
| 1 | 9999.000000 | 9999.000000 |
| 2 | 9999.000000 | 9999.000000 |
| 3 | 0.113889    | 0.616122    |
| 4 | 0.014707    | -1.731660   |
| 5 | 9999.000000 | 9999.000000 |
| 6 | 1.233087    | 0.720138    |
| 7 | 9999.000000 | 9999.000000 |
| 8 | 9999.000000 | 9999.000000 |
| 9 | 9999.000000 | 9999.000000 |



```
df=df.replace(9999.0, 0)  
df
```

|   | 0         | 1         |
|---|-----------|-----------|
| 0 | -0.349596 | -2.017159 |
| 1 | 0.000000  | 0.000000  |
| 2 | 0.000000  | 0.000000  |
| 3 | 0.113889  | 0.616122  |
| 4 | 0.014707  | -1.731660 |
| 5 | 0.000000  | 0.000000  |
| 6 | 1.233087  | 0.720138  |
| 7 | 0.000000  | 0.000000  |
| 8 | 0.000000  | 0.000000  |
| 9 | 0.000000  | 0.000000  |

9999.0000

0.0000

## ➤ fillna: fillna, backfill or by giving as parameter a Series

|   | 0         | 1        |
|---|-----------|----------|
| 0 | 0.061038  | 1.339673 |
| 1 | NaN       | NaN      |
| 2 | 1.578293  | 0.637435 |
| 3 | NaN       | NaN      |
| 4 | NaN       | NaN      |
| 5 | NaN       | NaN      |
| 6 | NaN       | NaN      |
| 7 | NaN       | NaN      |
| 8 | NaN       | NaN      |
| 9 | -1.145787 | 0.052887 |

|   | 0         | 1        |
|---|-----------|----------|
| 0 | 0.061038  | 1.339673 |
| 1 | 0.061038  | 1.339673 |
| 2 | 1.578293  | 0.637435 |
| 3 | 1.578293  | 0.637435 |
| 4 | 1.578293  | 0.637435 |
| 5 | 1.578293  | 0.637435 |
| 6 | 1.578293  | 0.637435 |
| 7 | 1.578293  | 0.637435 |
| 8 | 1.578293  | 0.637435 |
| 9 | -1.145787 | 0.052887 |

|   | 0         | 1        |
|---|-----------|----------|
| 0 | 0.061038  | 1.339673 |
| 1 | 1.578293  | 0.637435 |
| 2 | 1.578293  | 0.637435 |
| 3 | -1.145787 | 0.052887 |
| 4 | -1.145787 | 0.052887 |
| 5 | -1.145787 | 0.052887 |
| 6 | -1.145787 | 0.052887 |
| 7 | -1.145787 | 0.052887 |
| 8 | -1.145787 | 0.052887 |
| 9 | -1.145787 | 0.052887 |

# Clean data

## ➤ Drop

| df |           |           |           |
|----|-----------|-----------|-----------|
|    | 0         | 1         | 2         |
| 0  | NaN       | NaN       | -0.335410 |
| 1  | NaN       | NaN       | 0.685743  |
| 2  | 0.077005  | 0.085073  | 0.565144  |
| 3  | 0.394961  | -1.829587 | 0.494039  |
| 4  | 1.486227  | -0.480726 | -0.127278 |
| 5  | NaN       | NaN       | -0.047668 |
| 6  | NaN       | NaN       | -1.504804 |
| 7  | -0.530518 | -0.881817 | -2.687352 |
| 8  | 0.825376  | 1.042468  | -0.311527 |
| 9  | 0.097617  | 1.373572  | -0.682435 |

| df.dropna(axis=0) |           |           |           |
|-------------------|-----------|-----------|-----------|
|                   | 0         | 1         | 2         |
| 2                 | 0.077005  | 0.085073  | 0.565144  |
| 3                 | 0.394961  | -1.829587 | 0.494039  |
| 4                 | 1.486227  | -0.480726 | -0.127278 |
| 7                 | -0.530518 | -0.881817 | -2.687352 |
| 8                 | 0.825376  | 1.042468  | -0.311527 |
| 9                 | 0.097617  | 1.373572  | -0.682435 |

| df.dropna(axis=1) |           |
|-------------------|-----------|
|                   | 2         |
| 0                 | -0.335410 |
| 1                 | 0.685743  |
| 2                 | 0.565144  |
| 3                 | 0.494039  |
| 4                 | -0.127278 |
| 5                 | -0.047668 |
| 6                 | -1.504804 |
| 7                 | -2.687352 |
| 8                 | -0.311527 |
| 9                 | -0.682435 |

other: `df.dropna(how='all')`

`df.drop(dataf[filter2].index)`

# Clean data

## ➤ Linear Interpolation

df

|   | 0         | 1         | 2         |
|---|-----------|-----------|-----------|
| 0 | -0.260156 | -1.666998 | -0.492616 |
| 1 | NaN       | NaN       | 0.396817  |
| 2 | -1.263953 | -0.562550 | -1.670459 |
| 3 | -0.765183 | -0.619429 | 0.316981  |
| 4 | -0.165719 | -0.678431 | 0.485722  |
| 5 | -1.243191 | 0.494006  | 0.145171  |
| 6 | 0.373786  | -0.769120 | -0.929956 |
| 7 | NaN       | NaN       | -2.251816 |
| 8 | -0.536697 | 1.228345  | -1.040728 |
| 9 | 0.254220  | -0.021794 | 1.268333  |

df.interpolate()

|   | 0         | 1         | 2         |
|---|-----------|-----------|-----------|
| 0 | -0.260156 | -1.666998 | -0.492616 |
| 1 | -0.762055 | -1.114774 | 0.396817  |
| 2 | -1.263953 | -0.562550 | -1.670459 |
| 3 | -0.765183 | -0.619429 | 0.316981  |
| 4 | -0.165719 | -0.678431 | 0.485722  |
| 5 | -1.243191 | 0.494006  | 0.145171  |
| 6 | 0.373786  | -0.769120 | -0.929956 |
| 7 | -0.081455 | 0.229613  | -2.251816 |
| 8 | -0.536697 | 1.228345  | -1.040728 |
| 9 | 0.254220  | -0.021794 | 1.268333  |

# Types of Attributes

- There are different types of attributes
  - **Nominal – categorical** : Nominal values represent discrete units . It has no orders
    - Examples: eye color, sex (M, F).
  - **Ordinal – categorical** : Ordinal values represent discrete and ordered units.
    - Examples: rankings (e.g., taste of potato chips on a scale from 1-10), grades, height in {tall, medium, short}, educational background (master, PhD)
  - **Interval – numerical** : Interval values hold no true zero and can represent values below zero..
    - Examples: temperatures in Celsius or Fahrenheit.
  - **Ratio – numeric** : Ratio values are the same as interval values, with the difference that they never fall below zero.
    - Examples: temperature in Kelvin, length, time, counts

# Handle categorical data

## ➤ Simple Replacement of Categorical Data with a Number

- Find the categorical column and separate it out with a different dataframe:  
`df_categorical = df.select_dtypes(exclude=[np.number])`
- Find the values: `df_categorical['Size'].unique()`

```
1 df = pd.read_csv('Data.csv')  
2 ser=pd.Series([0,1,2],index=['Small', 'Medium', 'Large'])
```

```
1 df['Size']=df['Size'].replace(ser)
```

# Handle categorical data

## ➤ One-Hot Encoding

In the previous encoding, categorical data is converted to numerical data, and the values are assigned labels (such as 1, 2, and 3). Predictive models that use this numerical data for analysis might sometimes mistake these labels for some kind of order (for example, a model might think that a label of 3 is "better" than a label of 1, which is incorrect). In order to avoid this confusion, we can use one-hot encoding.

```
1 from sklearn.compose import ColumnTransformer
2 from sklearn.preprocessing import OneHotEncoder
3 #create an object of the ColumnTransformer class
4 df = pd.read_csv('Data.csv')
5 ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])])
6 X = ct.fit_transform(df) #X is a numpy array
7 df['country_fr'] = X[:,0]
8 df['country_ger'] = X[:,1]
9 df['country_sp'] = X[:,2]
10 df=df.drop(['Country'],axis=1)
11 #reorder the columns
12 df=df[['Age', 'Salary', 'country_fr','country_ger', 'country_sp','Purchased']]
13 print(df)
14
```

|   | Age  | Salary  | country_fr | country_ger | country_sp | Purchased |
|---|------|---------|------------|-------------|------------|-----------|
| 0 | 44.0 | 72000.0 | 1.0        | 0.0         | 0.0        | No        |
| 1 | 27.0 | 48000.0 | 0.0        | 0.0         | 1.0        | Yes       |
| 2 | 30.0 | 54000.0 | 0.0        | 1.0         | 0.0        | No        |
| 3 | 38.0 | 61000.0 | 0.0        | 0.0         | 1.0        | No        |
| 4 | 40.0 | NaN     | 0.0        | 1.0         | 0.0        | Yes       |