EPL448: Data Mining on the Web – Lab 4

University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: Β109, ΘΕΕ01

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

Data Manipulation (Lab 4)

- NumPy
- SciPy
- **Pandas**

Data Visualization (Lab 5)

- **matplotlib**
- **seaborn**

Data Pre-processing (Labs 6-7)

Machine Learning (Labs 8-9)

- **SciKit-Learn**

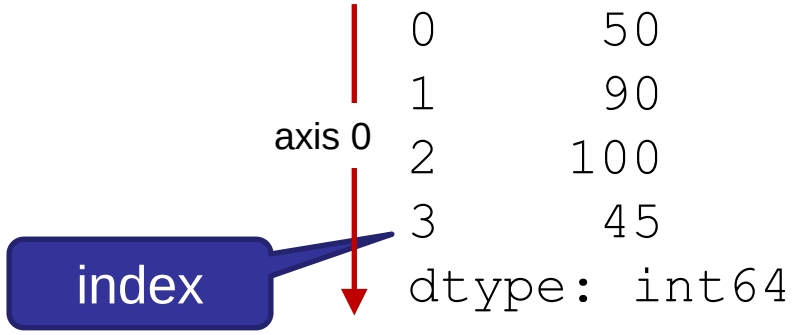
Introduction to Pandas

- Pre-installed in Anaconda
  - No need to install it separately
  - import it to a notebook using: `import pandas as pd`
    - pd is the de facto abbreviation for Pandas used by the data science community

- Primary data structures in Pandas:
  - Series
  - DataFrames

Pandas: Series & DataFrames Examples

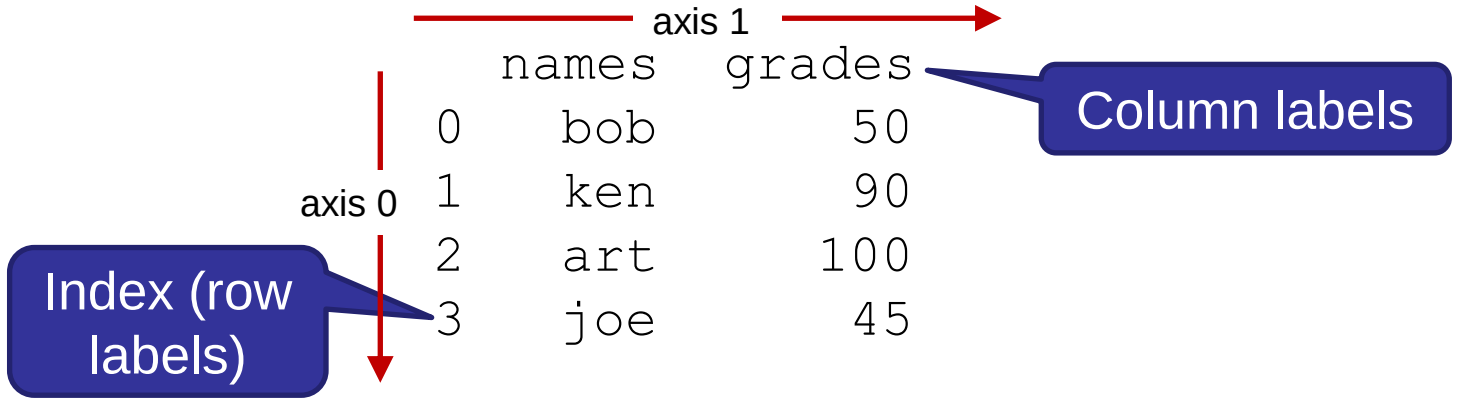
- A **Series** is a one-dimensional array with axis labels



Axis labels are stored in the index. Series support both integer-based and string-based indexing. The default, if index is not set, is integer-based starting from 0.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

- A **DataFrame** is a two-dimensional tabular dataset with labeled axes



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

Read data using Pandas

```
import pandas as pd # load pandas library and create alias pd

# Load dataset iris_data.csv and create dataframe object
df = pd.read_csv('iris_data.csv') # you can read a file from url as well:
df = pd.read_csv('http://www.cs.ucy.ac.cy/courses/EPL448/labs/LAB04/iris_data.csv')
```

- There is a number of pandas commands to read other data formats:

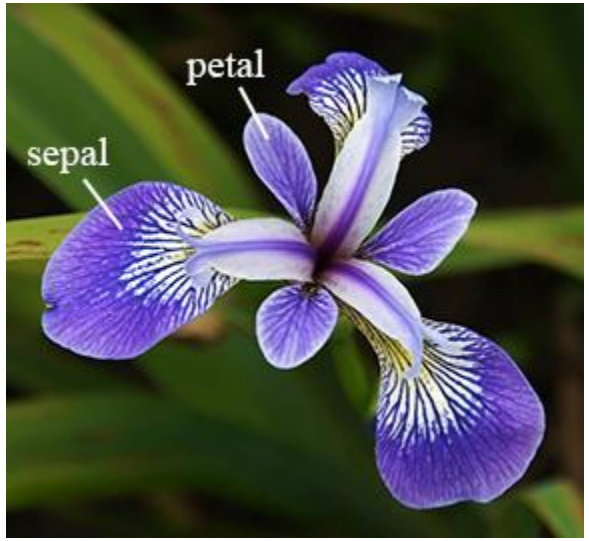
```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_xml('myfile.xml')
pd.read_sql_query('select * from iris', conn)
```

- A file may not always have a header row
  - pandas to assign default column names (but not informative)
  - you can specify (column) names yourself

```
df = pd.read_csv('https://www.cs.ucy.ac.cy/courses/EPL448/labs/LAB04/iris_data2.csv',
        names=['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class'])
```

Dataset: The iris dataset

- iris flowers dataset
  - “hello world” dataset in machine learning and statistics
- Small dataset with 150 observations of iris flowers
  - each observation has 4 columns of measurements (or variables or features) of the flowers (in centimeters)
  - 5th column is the species (class) of the flower observed
  - all observed flowers belong to one of three species




```
sepal-length,sepal-width,petal-length,petal-width,class
5.1,3.5,1.4,0.2,Iris-setosa
5.9,3.0,4.2,1.5,Iris-versicolor
5.8,2.7,5.1,1.9,Iris-virginica
4.6,3.1,1.5,0.2,Iris-setosa
```

- More info: https://en.wikipedia.org/wiki/Iris_flower_data_set

Dataset Overview

- Features:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm

- Classes (labels):
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

Read data using Pandas: Example

In [ ]:
```
# Read csv file
df = pd.read_csv('iris_data.csv')
print(df)
```

Out[ ]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
..	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

By default, only 10 rows (first & last 5 rows) of the DataFrame are printed. If the number of columns is also large, only 4 columns (first & last 2) are printed.

Read data using Pandas: Example

In [ ]:
```
# Read csv file
df = pd.read_csv('iris_data.csv')
df # jupyter can print the value of the last statement of a cell without a print() function in a more visually-appealing way
```

Out[ ]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

By default, only 10 rows (first & last 5 rows) of the DataFrame are printed. If the number of columns is also large, only 4 columns (first & last 2) are printed.

DataFrames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements (number of all values)
shape	return a tuple representing the dimensionality (rows, columns)
values	numpy representation of the data without index and column names

Data Frames methods

Unlike attributes, methods have *parentheses*. See some methods below.
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate summary statistics (for numeric columns only)
max(), min()	return max/min values. Select all numeric columns using `numeric_only=True`
sum()	return the sum of values. Select all numeric columns using `numeric_only=True`
mean(), median(), std()	return mean/median/standard deviation. Select all numeric columns using `numeric_only=True`
dropna()	drop all rows with at least one missing value (set axis param to 1 to drop columns)
drop()	drop specified rows or columns
count()	count non-NA cells
value_counts()	returns counts of unique values in descending order so that the first element is the most frequently-occurring element
apply()	applies a function along an axis of the DataFrame
replace()	replaces values with other values
cut()	bin (group) values into discrete intervals
astype(type)	casts to a specified data type (e.g. astype(int))

Explore DataFrames

- Dimensions of Dataset

```
# shape
print(df.shape)
```

```
(150, 5)
```

- Peek at the Data

```
# head (10 first lines)
print(df.head(10))
```

```
   sepal-length  sepal-width  petal-length  petal-width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
5           5.4          3.9           1.7          0.4  Iris-setosa
6           4.6          3.4           1.4          0.3  Iris-setosa
7           5.0          3.4           1.5          0.2  Iris-setosa
8           4.4          2.9           1.4          0.2  Iris-setosa
9           4.9          3.1           1.5          0.1  Iris-setosa
```

- Data type for each column

```
print(df.dtypes)
```

```
sepal-length    float64
sepal-width     float64
petal-length    float64
petal-width     float64
class            object
```

- Statistical summary

```
# descriptions
print(df.describe())
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

Explore DataFrames

```
#List DataFrame columns
print(df.columns)
```

```
Index(['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class'],
dtype='object')
```

```
#Get DataFrame values
print(df.values)
```

```
[[5.1 3.5 1.4 0.2 'Iris-setosa']
 [4.9 3.0 1.4 0.2 'Iris-setosa']
 [4.7 3.2 1.3 0.2 'Iris-setosa']
  ...
 [6.5 3.0 5.2 2.0 'Iris-virginica']
 [6.2 3.4 5.4 2.3 'Iris-virginica']
 [5.9 3.0 5.1 1.8 'Iris-virginica']]
```

Select a column in a Data Frame

Method 1: Use the column name in square brackets:

```
df['sepal-width']
```

Can be used to select more than one column:

```
df[['sepal-width', 'sepal-length']]
```

Method 2: Use the column name as an attribute:

```
df.column_name
```

Note: column names can be used as attributes when they are valid variable names (starting with a letter followed by a letter, a number or an underscore _ and do not conflict with a predefined attribute e.g. max, min, etc. For example, we cannot use df.sepal-length since - is considered as the minus symbol.

Drop rows / columns

- `drop()` method can be used to remove index (rows) or columns by specifying label names and corresponding axis, or by specifying directly index or column names
  - Drops rows if `axis = 0` or `'index'` (default), drops columns of `axis = 1` or `'columns'`
  - Returns a new DataFrame without the removed rows or columns

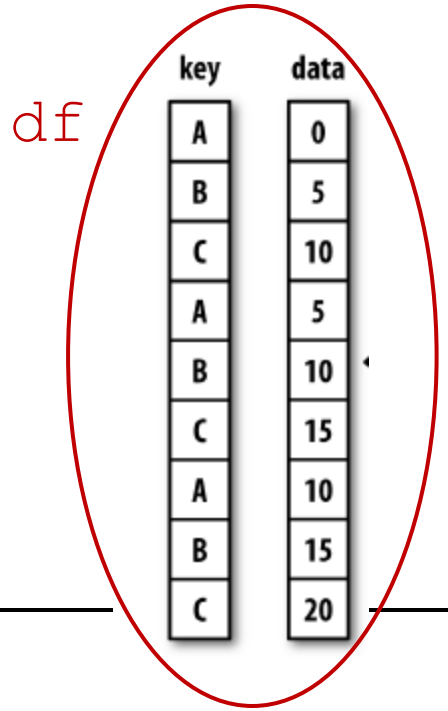
In [ ]:
```
# drop 2 columns
df_new = df.drop(columns=['sepal-length', 'sepal-width'])
# alternative command with the same effect
df_new = df.drop(['sepal-length', 'sepal-width'], axis=1)
# drop index (rows) with index = 3 and 5
df_new = df.drop(index=[3, 5])
# alternative command with the same effect
df_new = df.drop([3, 5])
# alternative command with the same effect
df_new = df.drop([3, 5], axis=0)
```

DataFrame processing operations

- Group-by method
  - splits data into groups by a categorical col, e.g. group faculty members by rank or gender
- Aggregate (or agg) method
  - an aggregation method is one which takes multiple individual values and returns a summary; in most of the cases, this summary is a single value
  - apply multiple aggregation methods on one or more columns or groups
- Splitting (binning) operation
  - bins (groups) values into discrete intervals; convert numerical to categorical, e.g. split faculty member into categorical groups based on their age: 25-39 young, 40-59 middle, 60+ old
- Filtering operation
  - selects a subset of data based on one or more conditions
- Slicing operation
  - selects a subset of data by row or column position/label
- Sorting operation
  - sorts data by the values of one or more column(s)

Grouping data using groupby()

- Using "group by" method we can:
  - Split the data into groups based on some criteria (e.g. the values of a column)
  - Once a groupby object is created, we can run aggregation methods (e.g. sum, mean) on each group and combine the results into a data structure

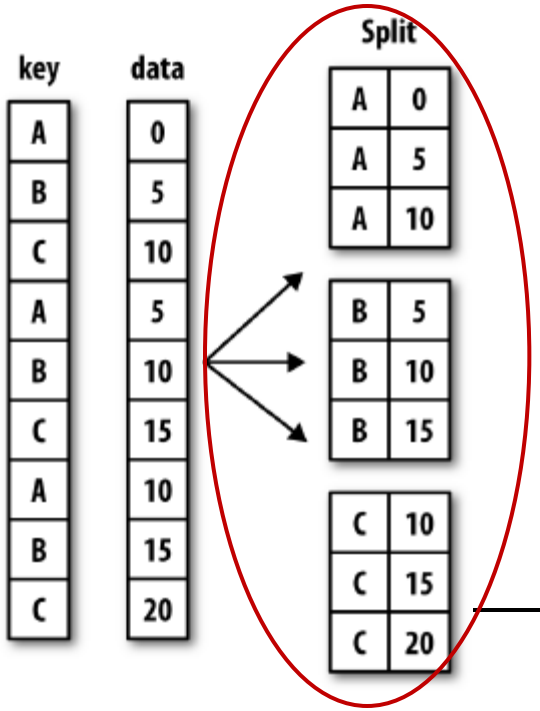


Grouping data using groupby()

- Using "group by" method we can:
  - Split the data into groups based on some criteria (e.g. the values of a column)
  - Once a groupby object is created, we can run aggregation methods (e.g. sum, mean) on each group and combine the results into a data structure

```
df.groupby('key')
```

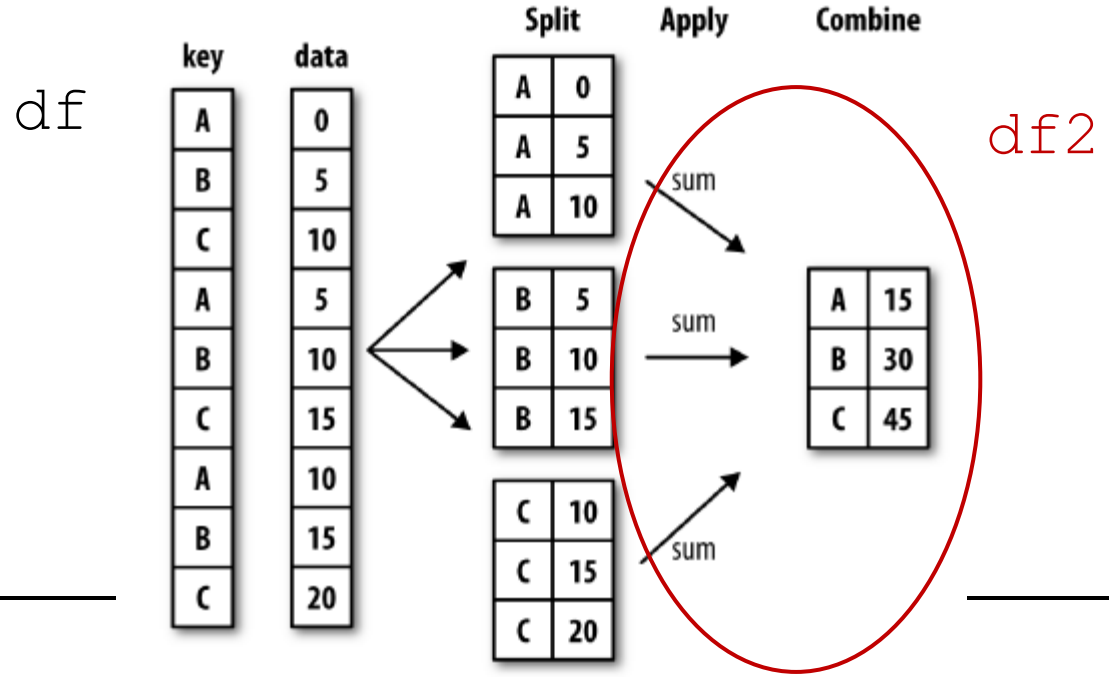
df



Grouping data using groupby()

- Using "group by" method we can:
  - Split the data into groups based on some criteria (e.g. the values of a column)
  - Once a groupby object is created, we can run aggregation methods (e.g. sum, mean) on each group and combine the results into a data structure

```
df2 = df.groupby('key').sum()
```



Grouping data using groupby(): Example

- Using "group by" method we can:
  - Split the data into groups based on some criteria (e.g. the values of a column)
  - Once a groupby object is created, we can run aggregation functions (e.g. sum, mean) on each group and combine results into a data structure

```
In [ ]: #Group data using class column which has categorical data
        df_groupby_class = df.groupby('class')
```

```
In [ ]: #Calculate mean value for each numeric column per each group
        df_groupby_class.mean(numeric_only=True)
```

class	sepal-length	sepal-width	petal-length	petal-width
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
Iris-virginica	6.588	2.974	5.552	2.026

Groups

When applying groupby, a new grouby object is returned having the specified group as index name

Grouping data using groupby()

- On the groupby object we can first isolate column(s) and then run aggregation functions (statistics) for each group:

```
In [ ]: # Isolate sepal-length and calculate mean for each flower class:
        df_groupby_class[['sepal-length']].mean(numeric_only=True)
```

```
Out[ ]:                  sepal-length
        class
        Iris-setosa       91786.230769
        Iris-versicolor   81362.789474
        Iris-virginica   123624.804348
```

Group names

Note: If **single brackets** are used to specify the column (e.g. sepal-length), then the output is **Pandas Series** object. When **double brackets** are used the output is a **DataFrame**. Although some operations may overlap, many operations (methods) applied on **Series** and **DataFrames** differ due to the structural differences between the 2 data structures.

Grouping data using groupby()

- `groupby()` can be performed on multiple columns

In [ ]:
```
#Group data using class and other column(s) which have categorical data
df_class = df.groupby(['class', '...'])
df_class.mean(numeric_only=True)
```

- `groupby` performance notes:
  - by default, the group names are sorted during the groupby operation. You may want to set sort=False for potential processing speedup:

In [ ]:
```
#Calculate mean sepal-length for each flower class:
df.groupby('class', sort=False)[['sepal-length']].mean()
```

Aggregations on multiple columns or groups

- `agg()` method allows applying multiple aggregation methods on one or more columns or groups

In [ ]:
```
# get mean values for all columns
df_groupby_class.agg(['mean'])
```

Out[ ]:

	sepal-length	sepal-width	petal-length	petal-width
	mean	mean	mean	mean
class				
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
Iris-virginica	6.588	2.974	5.552	2.026

In [ ]:
```
# multiple aggregations (max, mean, std) for all columns
df_groupby_class.agg(['max','mean','std'])
```

Out[ ]:

	sepal-length			sepal-width			petal-length			petal-width		
	max	mean	std	max	mean	std	max	mean	std	max	mean	std
class												
Iris-setosa	5.8	5.006	0.352490	4.4	3.418	0.381024	1.9	1.464	0.173511	0.6	0.244	0.107210
Iris-versicolor	7.0	5.936	0.516171	3.4	2.770	0.313798	5.1	4.260	0.469911	1.8	1.326	0.197753
Iris-virginica	7.9	6.588	0.635880	3.8	2.974	0.322497	6.9	5.552	0.551895	2.5	2.026	0.274650

Split data values into bins (groups)

- `cut()` method bins (groups) values into discrete intervals
- Useful for going from a numerical (continuous) variable to a discrete (categorical) variable
  - For example, cut() could convert ages to groups of age ranges e.g. (0-12] → child, (12-18] → teenager, (18-60] → adult, (60-∞] → elder)
- Supports binning into equal-sized bins, or a pre-specified array of bins
- Returns an array-like object representing the respective bin for each value of the column to be split

Split data values into bins (groups)

- Example 1: Split flowers by their sepal-length into 3 equal-sized bins (all bins have the same number of flowers)

```
In [ ]: pd.cut(df['sepal-length'], bins=3)
```

```
Out[ ]: 0         (4.296, 5.5]
        1         (4.296, 5.5]
        2         (4.296, 5.5]
        3         (4.296, 5.5]
        4         (4.296, 5.5]
                      ...
        145         (5.5, 6.7]
        146         (5.5, 6.7]
        147         (5.5, 6.7]
        148         (5.5, 6.7]
        149         (5.5, 6.7]
        Name: sepal-length, Length: 150, dtype: category
        Categories (3, interval[float64, right]): [(4.296, 5.5] < (5.5, 6.7] < (6.7, 7.9]]
```

belong to the first bin

belong to the second bin

The result of the cut method is a new column that shows the bin each flower belongs to according to its sepal-length

Split data values into bins (groups)

- Example 1: Split flowers by their sepal-length into 3 equal-sized bins (a label for each split can be defined)

```
In [ ]: pd.cut(df['sepal-length'], bins=3, labels=['low', 'medium', 'high'])
```

```
Out[ ]: 0          low
        1          low
        2          low
        3          low
        4          low
                  ...
        145     medium
        146     medium
        147     medium
        148     medium
        149     medium
        Name: sepal-length, Length: 150, dtype: category
        Categories (3, object): ['low' < 'medium' < 'high']
```

belong to the first bin

belong to the second bin

Split data values into bins (groups)

- Example 2: Split flowers by their sepal-length into 3 pre-specified array of bins (a label for each split can be defined)

```
In [ ]: pd.cut(df['sepal-length'], bins=[4, 5, 6, np.inf], labels=['small','medium','large'])
```

```
Out[ ]: 0       medium
        1        small
        2        small
        3        small
        4        small
                 ...
        145      large
        146      large
        147      large
        148      large
        149     medium
        Name: sepal-length, Length: 150, dtype: category
        Categories (3, object): ['small' < 'medium' < 'large']
```

belongs to the second bin

belong to the first bin

belong to the third bin

belongs to the second bin

Data Frame: Filtering

- In order to filter data we can apply Boolean indexing. For example, if we want to filter rows in which the petal length value is greater than 5.5 cm:

```
#Select only those rows that petal-length is more than 5.5:
df_sub = df[ df['petal-length'] > 5.5 ]
```

```
0       False
1       False
2       False
3       False
4       False
        ...
145     False
146     False
147     False
148     False
149     False
Name: petal-length,
Length: 150, dtype:
bool
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

```
#Select only those rows that contain Iris setosa flowers:
df_i = df[ df['class'] == 'Iris-setosa' ]
```

Data Frame: Filtering

- Symbol & refers to AND condition which means meeting both the criteria:

```
df_1 = df[(df['class'] == 'Iris-setosa') & (df['petal-length'] > 5.5) ]
```

- Symbol | refers to OR condition which means meeting any of the criteria:

```
df_2 = df[ (df['class'] == 'Iris-virginica') | (df['petal-length'] > 2) ]
```

Data Frame: Filtering

- Select rows which a specific column involves specific values

```
df_3 = df[ df['petal-length'].isin([5,10,15]) ]
```

- Select rows which a specific column contains a specific letter

```
df_4 = df[ df['rank'].str.contains('f')]
```

- Select rows with NaN values in specific column

```
df_5 = df[ df['class'].isnull() ]
```

- Select rows with NaN values in any column

```
df_6 = df[ df.isnull().any(axis=1) ]
```

Data Frames: Slicing

- There is a number of ways to get a slice of the DataFrame:
  - select one or more columns
  - select one or more rows
  - select a subset of rows and columns

- Rows and columns can be selected by their position or label

Data Frames: Slicing (selecting columns)

- When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

```
In [ ]: #Select sepal-length column:
        df['sepal-length']
```

- When we need to select more than one columns and/or make the output to be a DataFrame, we should use double brackets:

```
In [ ]: #Select sepal-length and sepal-width columns:
        df[['sepal-length','sepal-width']]
```

Data Frames: Slicing (selecting rows)

- If we need to select a subset of rows, we can specify the range using :

```
In [ ]: #Select rows by their position:
        df[10:20]
```

- The first row has a position 0, and the last value in the range is not returned:
  - So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9
- BUT (*):

```
In [ ]: #Does not show the first row. USE iloc instead (see next slide)
        df[0]
```

(*) The primary purpose of the DataFrame indexing operator, [] is to select columns.

DataFrames: method iloc

- If we need to select a single **row using its integer position** we can use the method iloc:

```
In [ ]: #Select a row by its position:
        df.iloc[0]
```

```
Out[ ]: sepal-length            5.1
        sepal-width             3.5
        petal-length            1.4
        petal-width             0.2
        class           Iris-setosa
        Name: 0, dtype: object
```

← Result as Pandas Series

```
In [ ]: #Select a row by its position (returns Dataframe):
        df.iloc[[0]]
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa

← Result as Pandas DataFrame

DataFrames: method iloc

- If we need to select a range of rows and/or columns, **using their** (integer) **positions** we can use method iloc:

```
In [ ]: #Select rows and columns by their positions:
        df.iloc[10:20,[0, 2, 3]]
```

Out[ ]:

	sepal-length	petal-length	petal-width
10	5.4	1.5	0.2
11	4.8	1.6	0.2
12	4.8	1.4	0.1
13	4.3	1.1	0.1
14	5.8	1.2	0.2
15	5.7	1.5	0.4
16	5.4	1.3	0.4
17	5.1	1.4	0.3
18	5.7	1.7	0.3
19	5.1	1.5	0.3

DataFrames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  # (i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # All rows of the first column
df.iloc[:, -1] # All rows of the last column
```

```
df.iloc[0:7]        # First 7 rows
df.iloc[:, 0:2]     # All rows of the first 2 columns
df.iloc[1:3, 0:2]   # Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  # 1st and 6th rows and 2nd and 4th columns
```

DataFrames: method loc

- If we need to select a range of rows and/or columns, **using their labels** we can use method loc:

In [ ]:
```
#Select rows and columns by their labels:
df.loc[10:20,['sepal-width','sepal-length']]
```

Out[ ]:

	sepal-width	sepal-length
10	3.7	5.4
11	3.4	4.8
12	3.0	4.8
13	3.0	4.3
14	4.0	5.8
15	4.4	5.7
16	3.9	5.4
17	3.5	5.1
18	3.8	5.7
19	3.8	5.1
20	3.4	5.4

Data Frames: Sorting

- We can sort the data by the values of a specified column. By default, the sorting will occur in ascending order (change using ascending Boolean parameter) and a new dataframe is returned.

In [ ]:
```
# Create a new data frame from the original,
# sorted by the column sepal-length
df_sorted = df.sort_values(by = 'sepal-length')
df_sorted.head()
```

Out[ ]:

	sepal-length	sepal-width	petal-length	petal-width	class
13	4.3	3.0	1.1	0.1	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa

Data Frames: Sorting

- We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['sepal-length', 'sepal-width'],
ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	sepal-length	sepal-width	petal-length	petal-width	class
13	4.3	3.0	1.1	0.1	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
47	4.6	3.2	1.4	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa