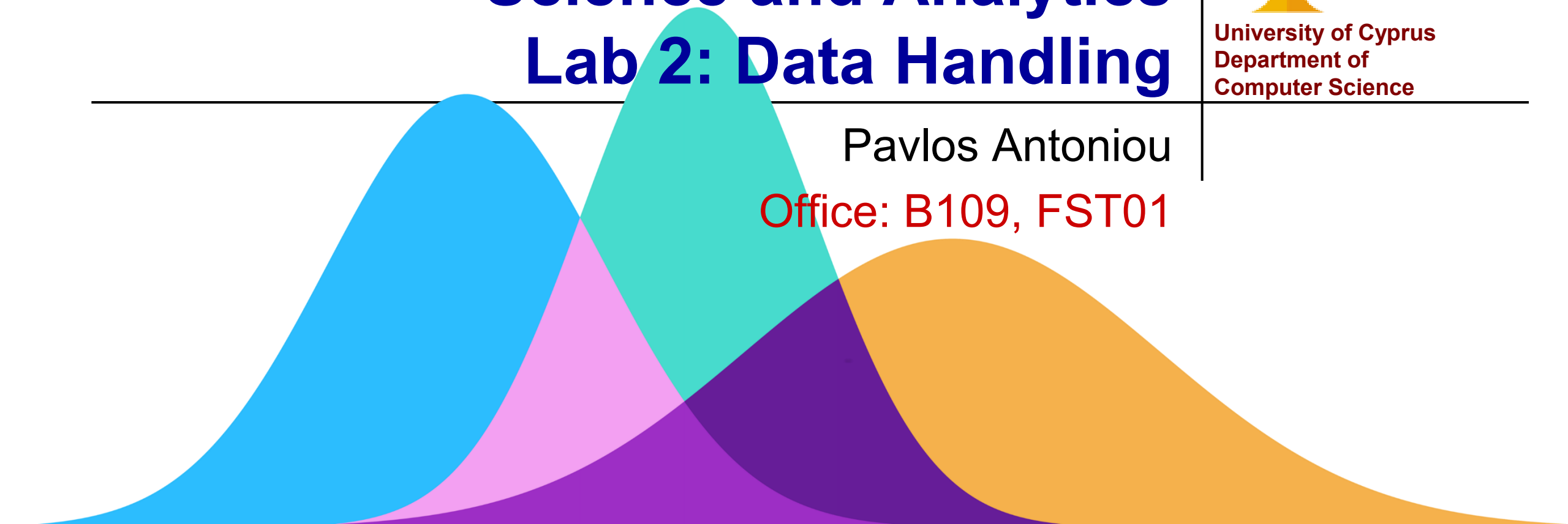
DSC510: Introduction to Data Science and Analytics
Lab 2: Data Handling



University of Cyprus
Department of
Computer Science

Pavlos Antoniou

Office: B109, FST01

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

Data Manipulation and Pre-processing (Labs 2, 4, 5)

- NumPy
- SciPy
- **Pandas**

Data Visualization (Lab 3)

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

Machine Learning (Labs 6-11)

- **SciKit-Learn**

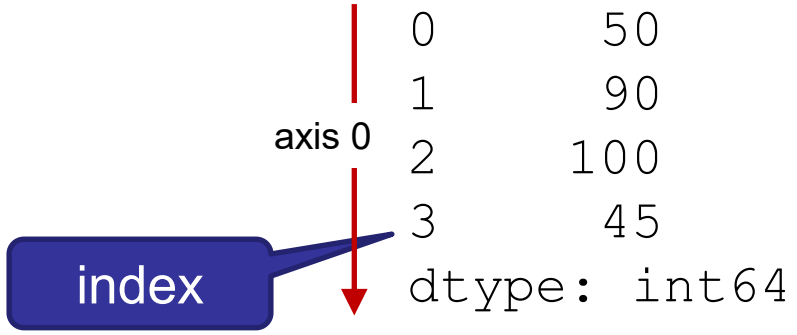
Introduction to Pandas

- 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

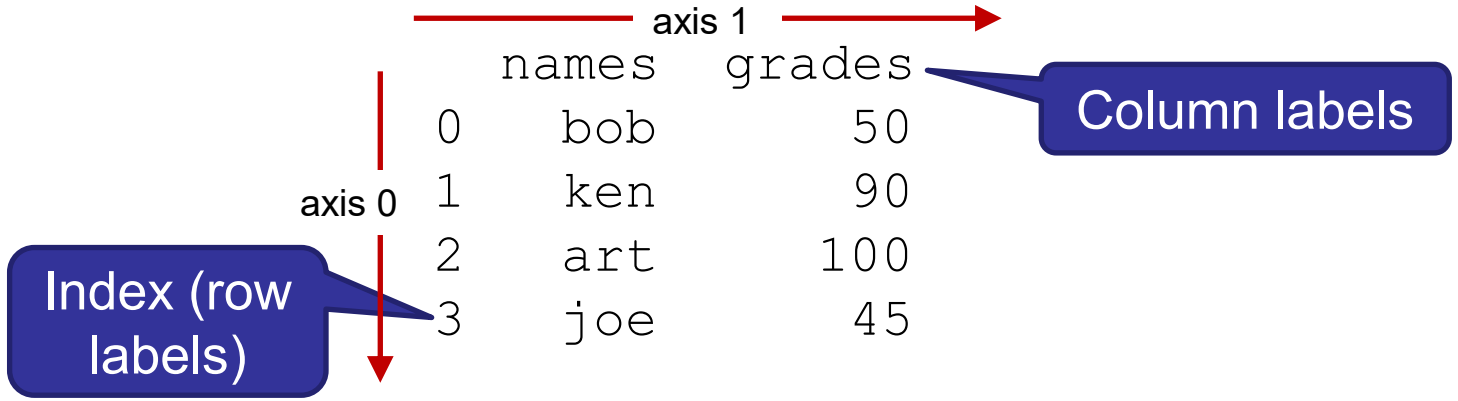
- A **Series** is a one-dimensional array with axis labels



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

Pandas: Create Dataframes

- Pandas DataFrames can be created using various inputs like:
  - List
  - Dictionary
  - Series
  - Another DataFrame
  - Files (e.g. csv / json)
  - Databases (e.g SQLite, MySQL, PostgreSQL)

Read data using Pandas

```
In [ ]: # Read csv file (salaries.csv can be found here)
        df = pd.read_csv('salaries.csv')  # you can read a file from url as well:
        df = pd.read_csv('https://www.cs.ucy.ac.cy/courses/DSC510/data/salaries.csv')
```

Note: read_csv command has many optional arguments to fine-tune the data import process.

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

```
# Read json file

pd.read_json('myfile.json')

# Read excel file

pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])

# Read xml file

pd.read_xml('myfile.xml')

# Read from database (connection to database must be established in advance)

pd.read_sql_query('select * from iris', conn)

...
```

Read data using Pandas: Example

```
rank,discipline,phd,service,sex,salary
Prof,B,56,49,Male,186960
Prof,A,12,6,Male,93000
Prof,A,23,20,Male,110515
Prof,A,40,31,Male,131205
Prof,B,20,18,Male,104800
Prof,A,20,20,Male,122400
AssocProf,A,20,17,Male,81285
Prof,A,18,18,Male,126300
Prof,A,29,19,Male,94350
```

```
In [ ]: # Read csv file (salaries of university faculty)
        df = pd.read_csv('salaries.csv')
        print(df)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800
..	...	...	...	...	...	...
73	Prof	B	18	10	Female	105450
74	AssocProf	B	19	6	Female	104542
75	Prof	B	17	17	Female	124312
76	Prof	A	28	14	Female	109954
77	Prof	A	23	15	Female	109646

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 (salaries of university faculty)
        df = pd.read_csv('salaries.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[ ]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800
...	...	...	...	...	...	...
73	Prof	B	18	10	Female	105450
74	AssocProf	B	19	6	Female	104542
75	Prof	B	17	17	Female	124312
76	Prof	A	28	14	Female	109954
77	Prof	A	23	15	Female	109646

78 rows × 6 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.

A bit more reading

- A file may not always have a header row (column titles)
  - pandas assigns default (but not informative) column names (a list of integers)
  - can be implicitly specified while reading data file

```
In [ ]: # Read csv file (salaries2.csv can be found here)
        df = pd.read_csv('salaries2.csv',
             names=['rank', 'discipline', 'phd', 'service', 'sex', 'salary'])
```

```
1  Prof,B,56,49,Male,186960
2  Prof,A,12,6,Male,93000
3  Prof,A,23,20,Male,110515
4  Prof,A,40,31,Male,131205
5  Prof,B,20,18,Male,104800
6  Prof,A,20,20,Male,122400
7  AssocProf,A,20,17,Male,81285
8  Prof,A,18,18,Male,126300
9  Prof,A,29,19,Male,94350
10 Prof,A,51,51,Male,57800
```

Explore DataFrames

In [ ]:
```
#List first 5 records (rows)
df.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

```
#List first 15 records
df.head(15)
```

```
#List last 5 records
df.tail()
```

Data Frame data types

```
In [ ]: #Check data types for all columns
        df.dtypes
```

```
Out[ ]: rank           object
        discipline     object
        phd             int64
        service         int64
        sex            object
        salary          int64
        dtype: object
```

The most general dtype. Will be assigned to your column if column has strings or mixed types (numbers and strings)

Numeric characters. 64 refers to the memory allocated (in bits) to hold each value of the given column

Hands-on exercises for submission

1. Load the following csv file `'https://www.cs.ucy.ac.cy/courses/DSC510/data/salaries.csv'` to a Pandas DataFrame. Then print the first 10 and the first 20 records.

2. Print the last 5 records of the dataframe.

Answer to all numbered questions you will see throughout this lab using any Python software (JupyterLab is recommended) in a single .ipynb file. See the last slide for submission guidelines and the deadline.

Data Frames attributes

Pandas **DataFrames** have *attributes* and *methods*. See some attributes below.

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 as a 2D array without index and column names

Explore DataFrames

In [ ]:
```
#List DataFrame columns
df.columns
```

Out[ ]:
```
Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')
```

In [ ]:
```
#Get DataFrame values
df.values
```

Out[ ]:
```
array([['Prof', 'B', 56, 49, 'Male', 186960],
       ['Prof', 'A', 12, 6, 'Male', 93000],
       ['Prof', 'A', 23, 20, 'Male', 110515],
       ...
       ['Prof', 'B', 17, 17, 'Female', 124312],
       ['Prof', 'A', 28, 14, 'Female', 109954],
       ['Prof', 'A', 23, 15, 'Female', 109646]], dtype=object)
```

Hands-on exercises for submission

3. Print the number of records (rows) of the df dataframe.

4. Print how many elements in total are in the df dataframe.

5. Print the column names of the df dataframe.

6. Print the types of columns of the df dataframe.

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)`

	phd	service	salary
count	78.000000	78.000000	78.000000
mean	19.705128	15.051282	108023.782051
std	12.498425	12.139768	28293.661022
min	1.000000	0.000000	57800.000000
25%	10.250000	5.250000	88612.500000
50%	18.500000	14.500000	104671.000000
75%	27.750000	20.750000	126774.750000
max	56.000000	51.000000	186960.000000

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))

Hands-on exercises for submission

7. Give the statistics summary for the numeric columns in the dataset.

8. Calculate standard deviation for all numeric columns.

9. Print the mean values of the first 50 records in the dataset.
   Hint: use head() method to subset the first 50 records and then calculate the mean.

Select a column in a Data Frame

Method 1: Use the column name in square brackets:

```
df['sex']
```

Can be used to select more than one column:

```
df[['salary', 'sex']]
```

Method 2: Use the column name as an attribute:

```
df.sex
```

Note: there is an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1.

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 (original dataframe remains intact)

In [ ]:
```
# drop 2 columns
df_new = df.drop(columns=['service', 'salary'])
# alternative command with the same effect
df_new = df.drop(['service', 'salary'], 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)
```

	rank	discipline	phd	sex	income
0	Prof	B	56	Male	9161040
1	Prof	A	12	Male	558000
2	Prof	A	23	Male	2210300
3	Prof	A	40	Male	4067355

	rank	discipline	phd	service	sex	salary	income
0	Prof	B	56	49	Male	186960	9161040
1	Prof	A	12	6	Male	93000	558000
2	Prof	A	23	20	Male	110515	2210300
4	Prof	B	20	18	Male	104800	1886400
6	AssocProf	A	20	17	Male	81285	1381845

Hands-on exercises for submission

10. Use the table of Pandas DataFrames methods to calculate the basic statistics only for the *salary* column

11. Print the unique values of the *salary* column

12. Count the unique values of the *salary* column

13. Calculate the average *salary*

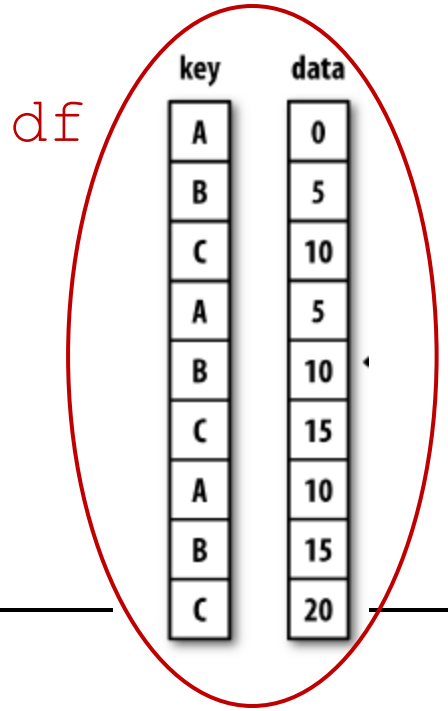
14. Delete the discipline column and print the new dataframe that is created.

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()

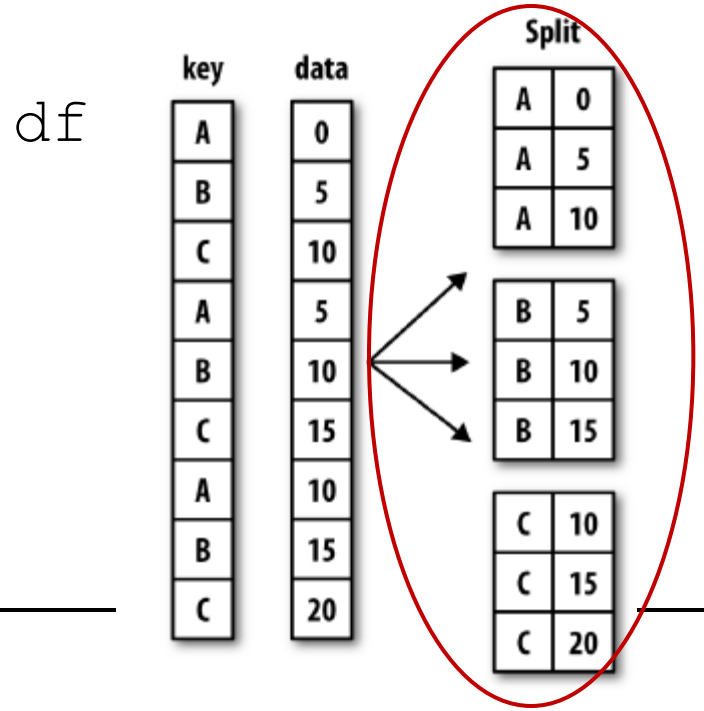
- Using "group by" method we can:
  - Split the data into groups based on some criteria (e.g. the values of a column)
  - Once a groupby object is created, we can run aggregation methods (e.g. sum, mean) on each group and combine the results into a data structure



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

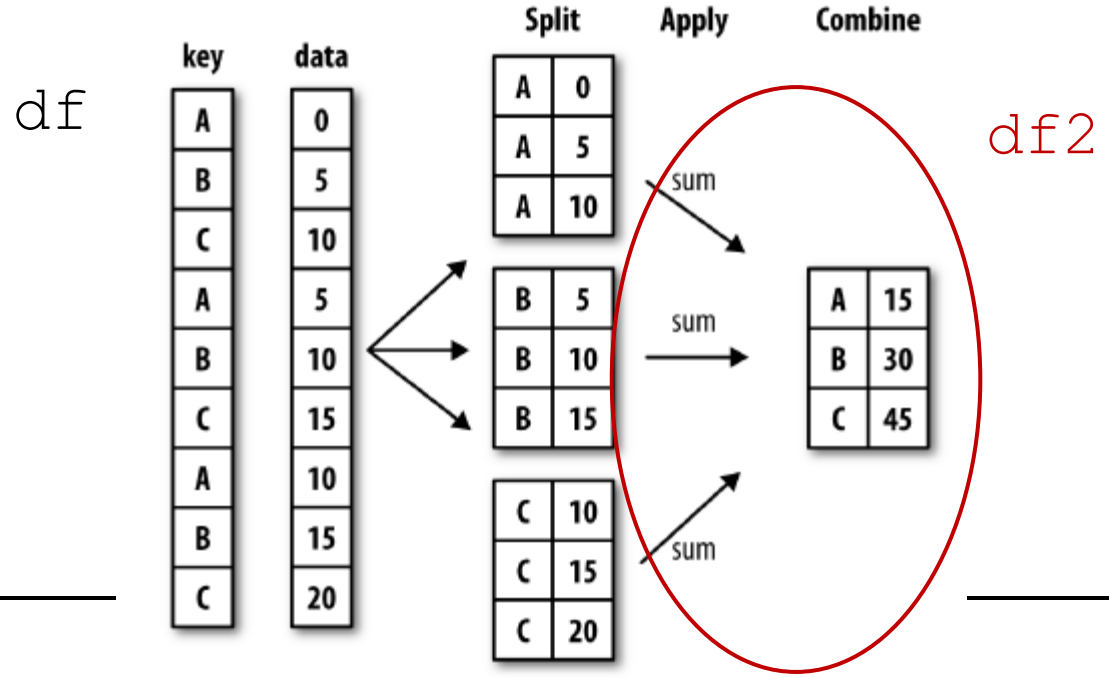
```
df.groupby('key')
```



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 methods (e.g. sum, mean) on each group and combine the results into a data structure

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

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

Groups

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

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 salary and calculate mean value for each rank:
        df_rank['salary'].mean(numeric_only=True)
```

```
Out[ ]: rank
        AssocProf     91786.230769
        AsstProf      81362.789474
        Prof         123624.804348
        Name: salary, dtype: float64
```

Group names

Note: If **single brackets** are used to select the column (e.g. ['salary'] instead of [['salary']]), then the output is **Pandas Series** object. When **double brackets** are used the output is a **DataFrame**.

```
In [ ]: # Isolate salary and calculate mean value for each rank:
        df_rank[['salary']].mean(numeric_only=True)
```

```
Out[ ]:                 salary
        rank
        AssocProf     91786.230769
        AsstProf      81362.789474
        Prof         123624.804348
```

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 rank and sex columns which have categorical data
df_rank_sex = df.groupby(['rank', 'sex'])
df_rank_sex.mean(numeric_only=True)
```

Multi-index: multi-level index

rank	sex	phd	service	salary
AssocProf	Female	15.500000	11.500000	88512.800000
	Male	13.666667	10.666667	102697.666667
AsstProf	Female	5.636364	2.545455	78049.909091
	Male	4.250000	1.750000	85918.000000
Prof	Female	23.722222	17.111111	121967.611111
	Male	29.214286	24.178571	124690.142857

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

In [ ]:
```
df.groupby(['rank'], sort=False).mean(numeric_only=True)
```

Aggregations on multiple columns or groups

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

```
In [ ]: df = pd.read_csv('salaries.csv')
        # get numeric columns
        df_numeric = df[['phd', 'service', 'salary']]
        # get mean values: the same as df_numeric.mean() - returns a Series
        df_numeric.agg('mean')
```

```
Out[ ]: phd            19.705128
        service        15.051282
        salary     108023.782051
        dtype: float64
```

```
df_numeric.agg(['mean']) #returns a DataFrame
             phd    service         salary
mean   19.705128  15.051282  108023.782051
```

```
In [ ]: # multiple aggregations for all columns - returns a DataFrame
        df_numeric.agg(['max','mean','std'])
```

```
Out[ ]:              phd    service         salary
        max    56.000000  51.000000  186960.000000
        mean   19.705128  15.051282  108023.782051
        std    12.498425  12.139768   28293.661022
```

Aggregations on multiple columns or groups

```
In [ ]: # multiple statistics on specific columns
        df_numeric[['service','salary']].agg(['max','mean','std'])
```

```
Out[ ]:          service          salary
        max    51.000000  186960.000000
        mean   15.051282  108023.782051
        std    12.139768   28293.661022
```

```
In [ ]: # multiple different aggregations for each selected column(s)
        df_numeric.agg({'phd': ['max','mean','std'], 'service': ['count', 'mean']})
```

```
Out[ ]:              phd       service
        max    56.000000           NaN
        mean   19.705128     15.051282
        std    12.498425           NaN
        count        NaN     78.000000
```

Aggregations on multiple columns or groups

- `agg()` on applying aggregation methods on groups (groupby)

```
In [ ]: # multiple statistics for each professor rank (for all columns)
        df.groupby('rank').agg(['max','mean','std'])
```

Out[ ]:

	phd			service			salary	
	max	mean	std	max	mean	std	max	mean	std
rank									
AssocProf	26	15.076923	5.589597	24	11.307692	5.879124	119800	91786.230769	18571.183714
AsstProf	11	5.052632	2.738079	6	2.210526	1.750522	97032	81362.789474	9381.245301
Prof	56	27.065217	10.185834	51	21.413043	11.255766	186960	123624.804348	24850.287853

```
In [ ]: #Calculate max, mean, std of the salary column for each professor rank
        df.groupby('rank')[['salary']].agg(['max','mean','std'])
```

Out[ ]:

	max	mean	std
rank			
AssocProf	119800	91786.230769	18571.183714
AsstProf	97032	81362.789474	9381.245301
Prof	186960	123624.804348	24850.287853

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 faculty members by their salary into 3 equal-sized bins (all bins have the same number of faculty members)

```
In [ ]: pd.cut(df.salary, bins=3)
```

```
Out[ ]: 0       (143906.667, 186960.0]
        1       (57670.84, 100853.333]
        2     (100853.333, 143906.667]
        3     (100853.333, 143906.667]
        4     (100853.333, 143906.667]
                      ...
        73    (100853.333, 143906.667]
        74    (100853.333, 143906.667]
        75    (100853.333, 143906.667]
        76    (100853.333, 143906.667]
        77    (100853.333, 143906.667]
        Name: salary, Length: 78, dtype: category
        Categories (3, interval[float64, right]): [(57670.84, 100853.333] < (100853.333,
        143906.667] < (143906.667, 186960.0]]
```

belongs to the third bin

belongs to the first bin

belong to the second bin

The result of the cut method is a new column that shows the bin each faculty member belongs to according to his/her salary

Split data values into bins (groups)

- Example 1: Split faculty members by their salary into 3 equal-sized bins (a label for each split can be defined)

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

```
Out[ ]: 0       high
        1        low
        2     medium
        3     medium
        4     medium
              ...
        73    medium
        74    medium
        75    medium
        76    medium
        77    medium
        Name: salary, Length: 78, dtype: category
        Categories (3, object): ['low' < 'medium' < 'high']
```

belongs to the third bin

belongs to the first bin

belong to the second bin

Split data values into bins (groups)

- Example 2: Split faculty members by their salary into 3 pre-specified array of bins

```
In [ ]: pd.cut(df.salary, bins=[0, 100000, 150000, np.inf], labels=['low', 'medium', 'high'])
```

```
Out[ ]: 0        high
        1         low
        2      medium
        3      medium
        4      medium
               ...
        73     medium
        74     medium
        75     medium
        76     medium
        77     medium
        Name: salary, Length: 78, dtype: category
        Categories (3, object): ['low' < 'medium' < 'high']
```

belongs to the third bin

belongs to the first bin

belong 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 salary value is greater than $120K:

```
In [ ]: #Select only those that earn more than 12000:
        df_sub = df[ df['salary'] > 120000 ]
```

```
0      True
1     False
2     False
3      True
4     False
      ...
73    False
74    False
75     True
76    False
77    False
Name: salary, Length:
78, dtype: bool
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

```
In [ ]: #Select only those rows that contain female professors:
        df_f = df[ df['sex'] == 'Female' ]
```

Data Frame: Filtering

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

```
df_1 = df[ (df['sex'] == 'Female') & (df['service'] > 20) ]
```

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

```
df_2 = df[ (df['sex'] == 'Female') | (df['service'] > 20) ]
```

Data Frame: Filtering

- Select rows which a specific column involves specific values

```
df_3 = df[ df['phd'].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['service'].isnull() ]
```

- Select rows with NaN values in any column

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

Hands-on exercises for submission

15. On the original dataframe with all columns, group all faculty members by sex and show only the average salary as a dataframe

	salary
sex	
Female	101002.410256
Male	115045.153846

16. On the original dataframe with all columns, select and print only female faculty members who earn more than 120000

	rank	phd	service	sex	salary
39	Prof	18	18	Female	129000
40	Prof	39	36	Female	137000
44	Prof	23	19	Female	151768
45	Prof	25	25	Female	140096
49	Prof	17	18	Female	122960
51	Prof	20	14	Female	127512
58	Prof	36	26	Female	144651
72	Prof	24	15	Female	161101
75	Prof	17	17	Female	124312

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 salary column:
        df['salary']
```

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

```
In [ ]: #Select rank and salary columns:
        df[['rank','salary']]
```

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] # single [] select columns, if first column name is not 0 => error
```

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

Data Frames: 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[ ]: rank            Prof
        discipline         B
        phd               56
        service           49
        sex             Male
        salary        186960
        Name: 0, dtype: object
```

← **Result as Pandas Series**

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

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960

← **Result as Pandas DataFrame**

Data Frames: 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, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
10	Prof	33	Male	128250
11	Prof	23	Male	134778
12	AsstProf	0	Male	88000
13	Prof	33	Male	162200
14	Prof	19	Male	153750
15	Prof	3	Male	150480
16	AsstProf	3	Male	75044
17	AsstProf	0	Male	92000
18	Prof	7	Male	107300
19	Prof	27	Male	150500

Data Frames: 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
```

Data Frames: 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,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
12	AsstProf	Male	88000
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
16	AsstProf	Male	75044
17	AsstProf	Male	92000
18	Prof	Male	107300
19	Prof	Male	150500
20	AsstProf	Male	92000

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 service
df_sorted = df.sort_values(by = 'service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

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

In [ ]:

```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Submission

- Implement all numbered questions in a single .ipynb file
- Run the file and save results within the same file
- Submit .ipynb file (with commands and results) to Moodle by Tuesday 23rd of September @ 23.59
  - Login to Moodle: https://moodle.cs.ucy.ac.cy/course/view.php?id=312 using your UCY credentials
  - Follow “Lab2 Submission” link
  - Upload your .ipynb file