Introduction to Urban Data Science

CRP 4680/5680 Spring 2025

Lecture 2.2 Data Management (I)

Wenzheng Li
Hazel (Yujin) Lee

OUTLINE

○ Packages (libraries)

○ Basics of Pandas

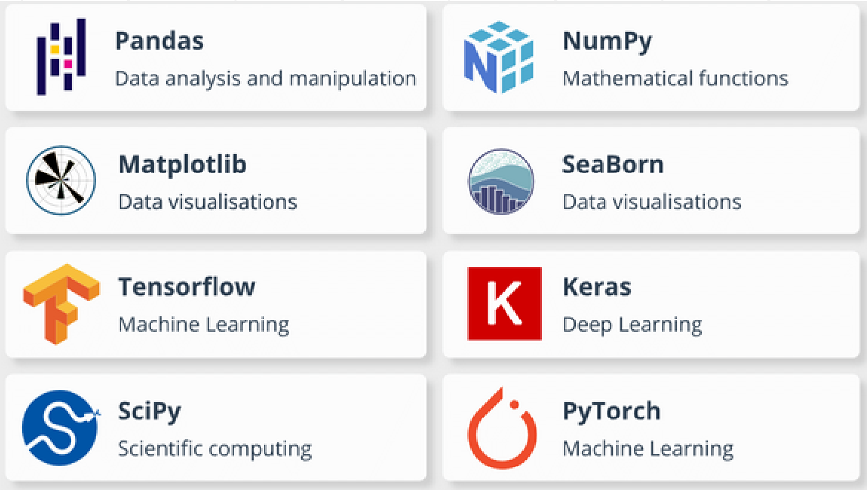
Packages (libraries)

1

Packages (also known as libraries or modules)

- Python’s standard library offers read-to-use solutions (functions and methods) to solve common programming problems without additional installation.
  - e.g., functions: print(), len(), range(), etc.

- **External Packages provide additional, specialized functionalities, but often need to be installed separately**
  - Install the package: Use a package manager like pip/conda (e.g., pip install pandas) ← We have completed this step.
  - Import the package: Load it into your script (e.g., import pandas as pd).



Pandas
Data analysis and manipulation

NumPy
Mathematical functions

Matplotlib
Data visualisations

SeaBorn
Data visualisations

Tensorflow
Machine Learning

Keras
Deep Learning

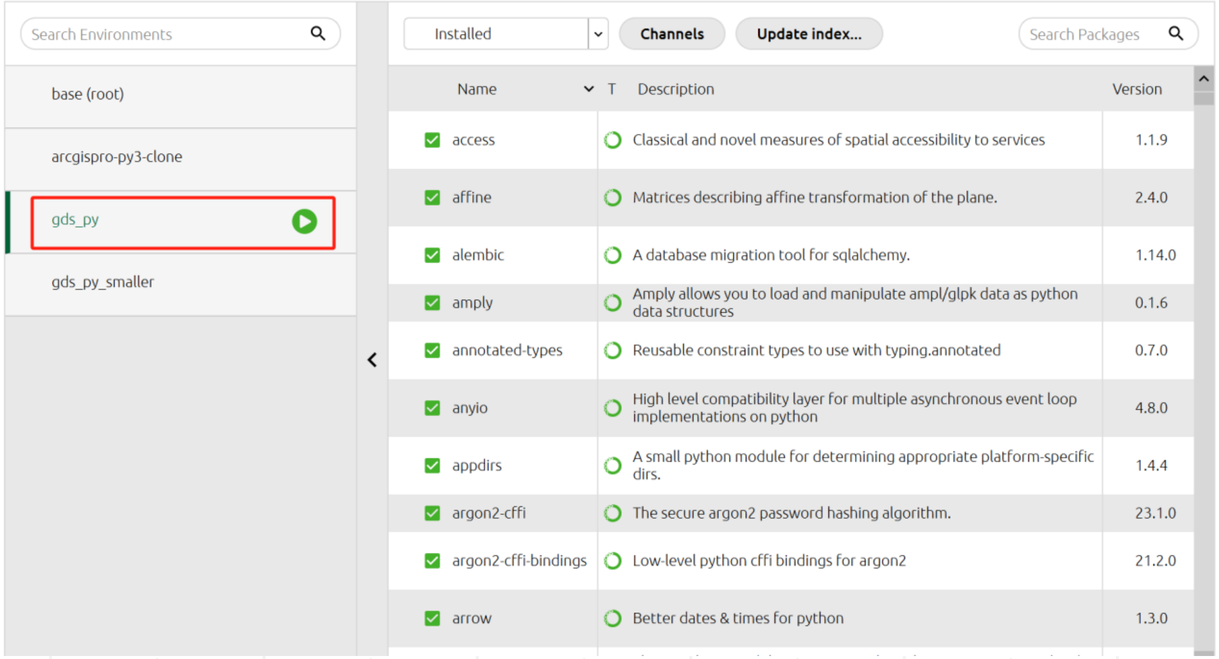
SciPy
Scientific computing

PyTorch
Machine Learning

The Python **virtual environment (the gds_py in our case)**

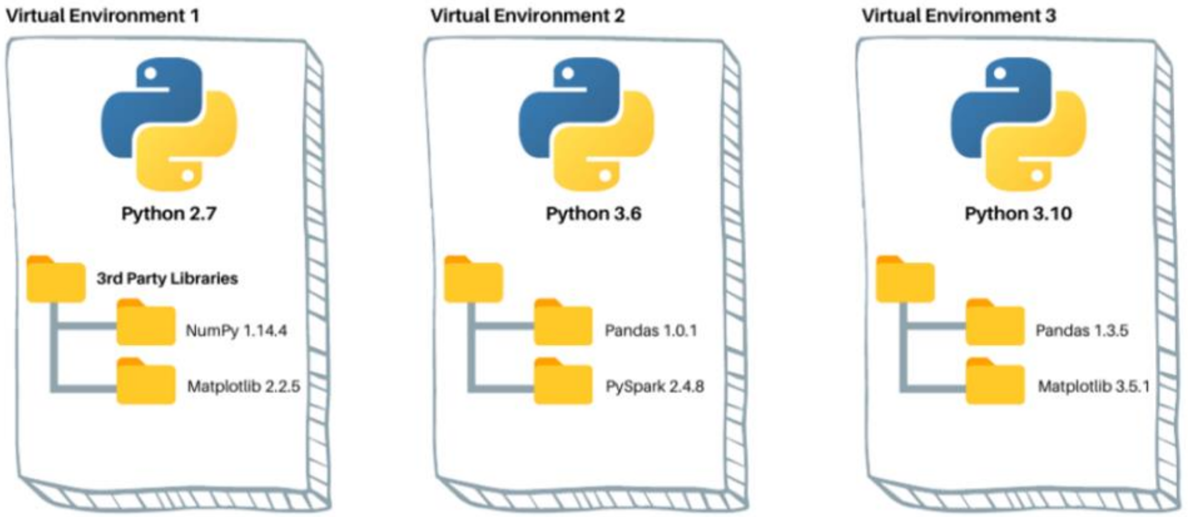
- gds_py: a container providing a fully working Jupyter Lab installation, additionally loaded with a comprehensive list of geospatial python packages.

- Virtual environment: a tool for dependency management and project isolation. It allows packages to be installed locally in an isolated directory for a particular project, as opposed to being installed globally.

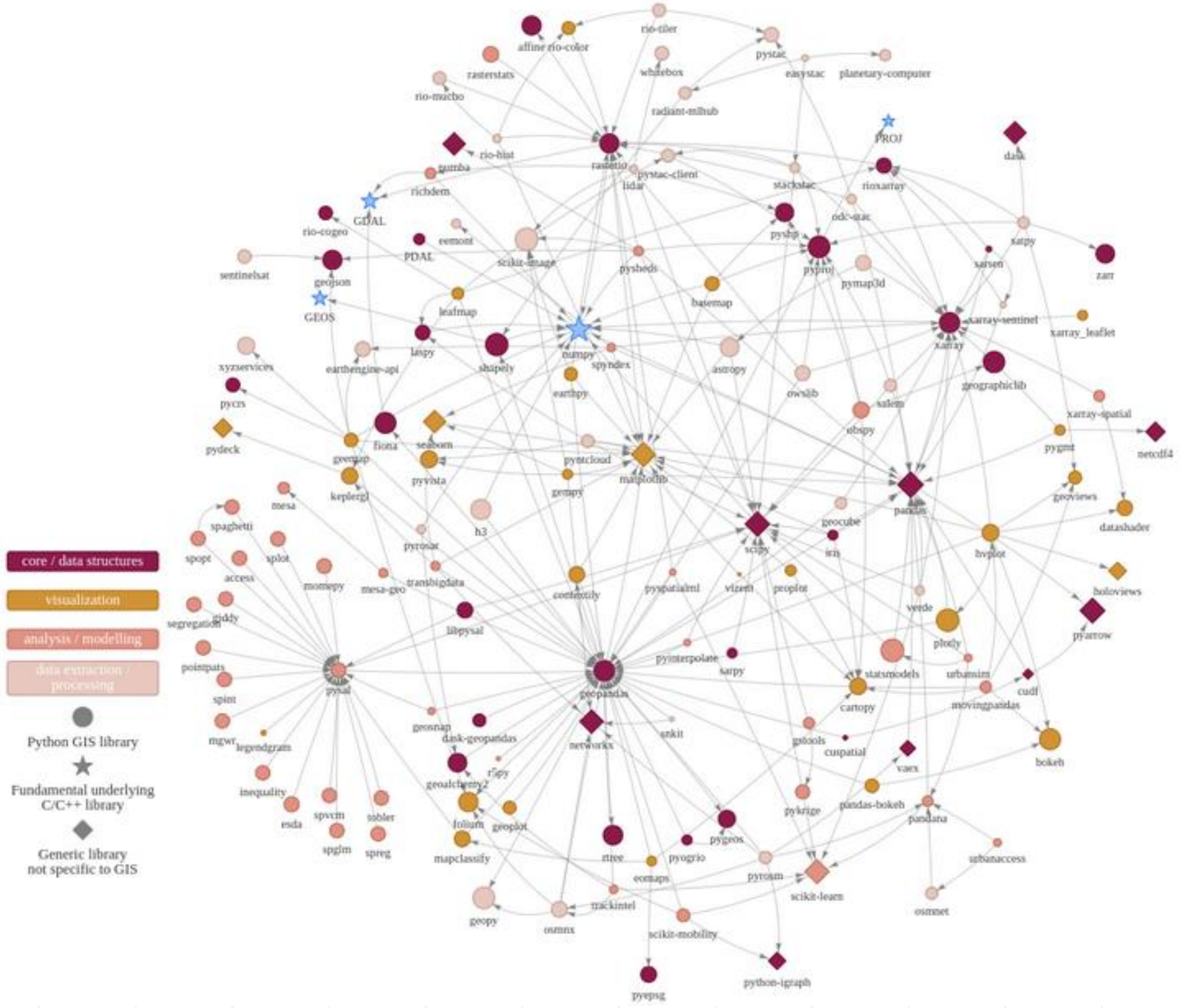


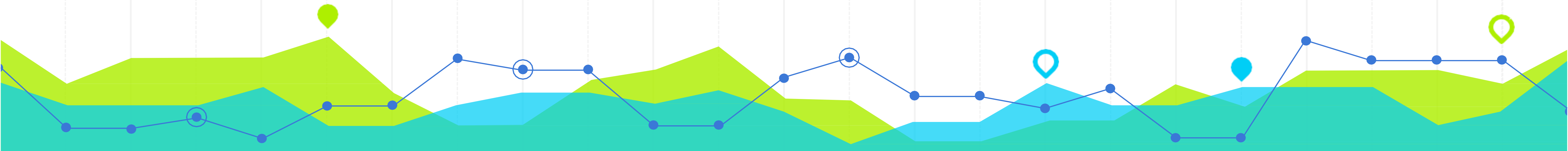
base (root)
arcgispro-py3-clone
gds_py
gds_py_smaller

	Name	T	Description	Version
☑	access	○	Classical and novel measures of spatial accessibility to services	1.1.9
☑	affine	○	Matrices describing affine transformation of the plane.	2.4.0
☑	alembic	○	A database migration tool for sqlalchemy.	1.14.0
☑	amply	○	Amply allows you to load and manipulate ampl/glpk data as python data structures	0.1.6
☑	annotated-types	○	Reusable constraint types to use with typing.annotated	0.7.0
☑	anyio	○	High level compatibility layer for multiple asynchronous event loop implementations on python	4.8.0
☑	appdirs	○	A small python module for determining appropriate platform-specific dirs.	1.4.4
☑	argon2-cffi	○	The secure argon2 password hashing algorithm.	23.1.0
☑	argon2-cffi-bindings	○	Low-level python cffi bindings for argon2	21.2.0
☑	arrow	○	Better dates & times for python	1.3.0



[Python OS Ecosystem for GIS and Earth Observation]





Basics of Pandas

2

Pandas?

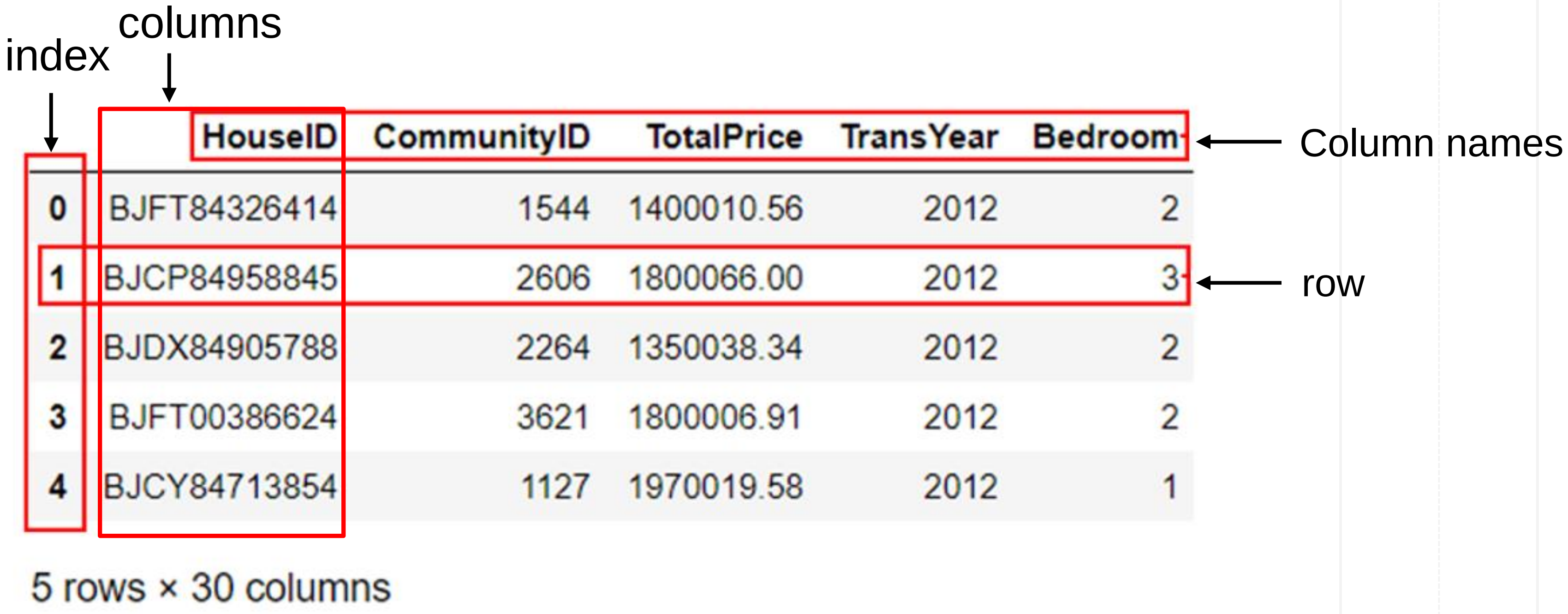
- Pandas is a tool in Python that allows us to read, write, and manage datasets in a variety of format (e.g., .xlsx, .csv, .pickle) through **Dataframes.**

- **data structures**
  - Data is stored in a structure called a **DataFrame**.
  - A Dataframe is tabular structure with labeled rows and columns—in many ways similar to Excel or Google Spreadsheets.
  - It supports operations like arithmetic, col/rows selection, filtering, and grouping, etc.

- Pandas is built on top of
  - Numpy: multi-dimensional arrays and scientific computing
  - Matplotlib: plotting
  - Python Standard library

Pandas?

- Pandas provides two new data types—Series and Dataframe.
  - ***Dataframe****:* A tabular structure with three key components: **columns, rows, and an index**.
  - ***Series:*** A one-dimensional array, representing a single column of data. You can think of Series objects as fancier versions of list.



columns

index

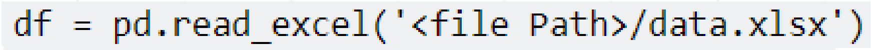
Column names

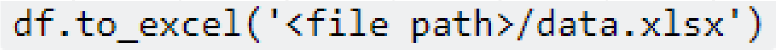
row

	HouseID	CommunityID	TotalPrice	TransYear	Bedroom
0	BJFT84326414	1544	1400010.56	2012	2
1	BJCP84958845	2606	1800066.00	2012	3
2	BJDX84905788	2264	1350038.34	2012	2
3	BJFT00386624	3621	1800006.91	2012	2
4	BJCY84713854	1127	1970019.58	2012	1

5 rows × 30 columns

Importing and exporting dataset in Pandas

- Pandas can import and export dataset in many data formats.
  - To read (import) a file (e.g., .csv, .xlsx) from a folder and present it as a DataFrame in Python.
    - 

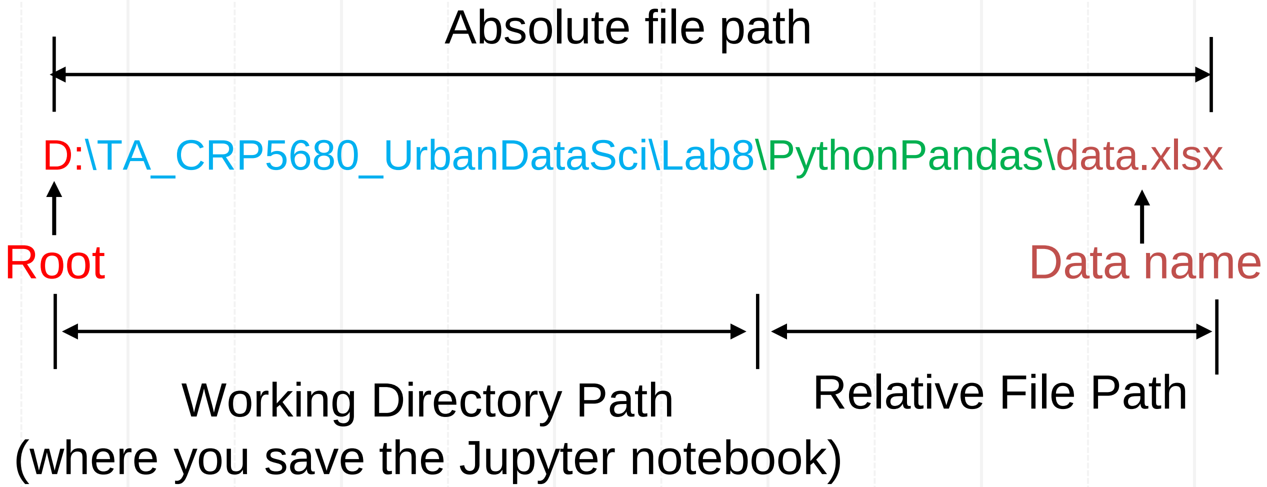
      ```
      df = pd.read_excel('<file Path>/data.xlsx')
      ```
  - To write (export) a DataFrame to a specific file (e.g., .xlsx) using Pandas.
    - 

      ```
      df.to_excel('<file path>/data.xlsx')
      ```
  - Pandas also supports reading and writing other formats, like .json, .html, .pickle. Check here all the data format Pandas can read.

File Path?

○ How to read a file path in Pandas?

- **We can always use an absolute file path—**file path that starts from the root of the file system.



Absolute file path

D:\TA_CRP5680_UrbanDataSci\Lab8\PythonPandas\data.xlsx

Root

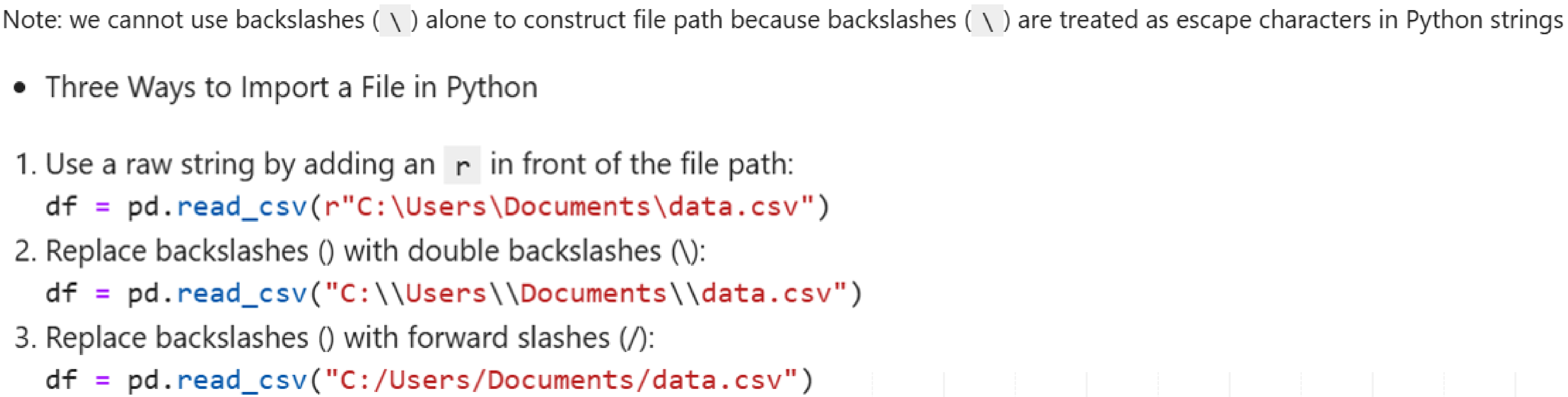
Data name

Working Directory Path
(where you save the Jupyter notebook)

Relative File Path

- **We can use a relative file path**
  - an incomplete file path that is joined to your **current working directory** to create an absolute file path.
  - **the relative path = the absolute path - the current working directory**

File Path in Pandas



Note: we cannot use backslashes (`\`) alone to construct file path because backslashes (`\`) are treated as escape characters in Python strings

- Three Ways to Import a File in Python

1. Use a raw string by adding an `r` in front of the file path:

```
df = pd.read_csv(r"C:\Users\Documents\data.csv")
```

2. Replace backslashes () with double backslashes (\):

```
df = pd.read_csv("C:\\Users\\Documents\\data.csv")
```

3. Replace backslashes () with forward slashes (/):

```
df = pd.read_csv("C:/Users/Documents/data.csv")
```

Pandas built-in functions

See codebook

- df.head(5) — check out the top five rows
- df.tail(5) — check the last five rows
- df.shape — explore the shape (dimensions) of the Dataframe. How many rows and how many columns?

- df.columns
  - print out column names
  - returns a Pandas object rather than a list
  - convert to a list: list(df.columns)

Indexing and Slicing a Dataframe

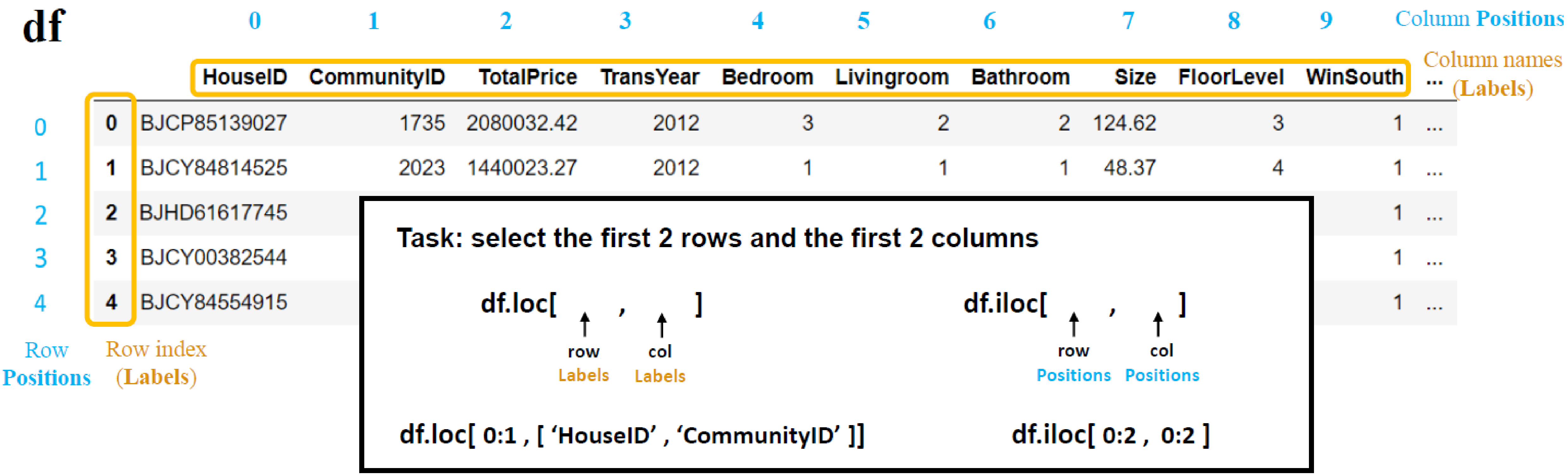
how to select a subset of a Dataframe?

- ***Indexing***: simply selecting a particular row or column from a Dataframe.
- ***Slicing*****:** Selecting a subset of rows and columns.

- Three ways of selecting particular rows and columns of a Dataframe
  - df[ ]: Basic indexing, primarily used for selecting columns.
  - df.loc[rows_***label*** , columns_***label***]: Select rows and columns by their labels.
  - df.iloc[row_***position*** , column_***position***]: Select rows and columns by their numerical positions.

- A ***label***: one name in the column list or an index in the row index (the column at far left).
- A ***position***: the corresponding position of column name or index in a sequence, starting from zero.

Label and Position



df

0 1 2 3 4 5 6 7 8 9 Column Positions

Column names (Labels)

Row Positions	Row index (Labels)	HouseID	CommunityID	TotalPrice	TransYear	Bedroom	Livingroom	Bathroom	Size	FloorLevel	WinSouth	...
0	0	BJCP85139027	1735	2080032.42	2012	3	2	2	124.62	3	1	...
1	1	BJCY84814525	2023	1440023.27	2012	1	1	1	48.37	4	1	...
2	2	BJHD61617745									1	...
3	3	BJCY00382544									1	...
4	4	BJCY84554915									1	...

Task: select the first 2 rows and the first 2 columns

df.loc[↑ , ↑]
row Labels
col Labels

df.iloc[↑ , ↑]
row Positions
col Positions

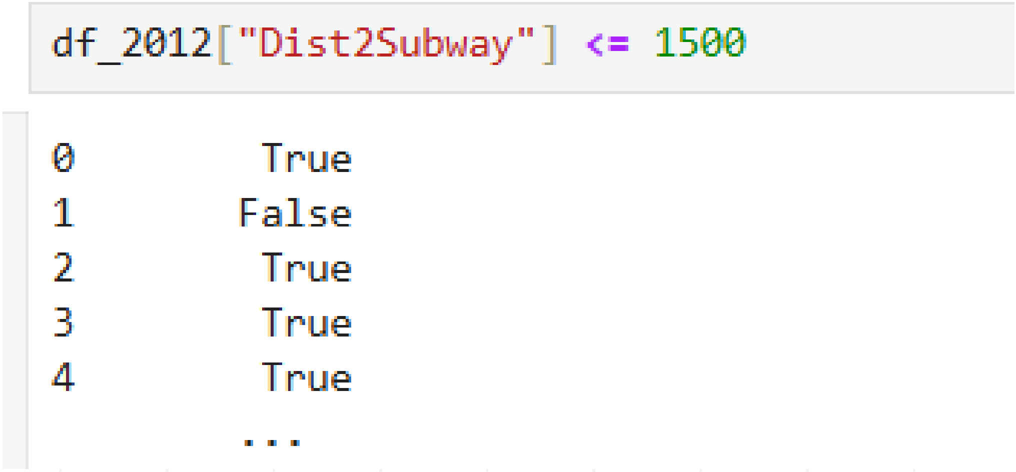
```
df.loc[ 0:1 , [ ‘HouseID’ , ‘CommunityID’ ]]
```

```
df.iloc[ 0:2 ,  0:2 ]
```

Filtering DataFrames

- how to filter rows from a DataFrame based on a condition

Goal: to filter out rows where the column Dist2Subway is less than or equal to 1500



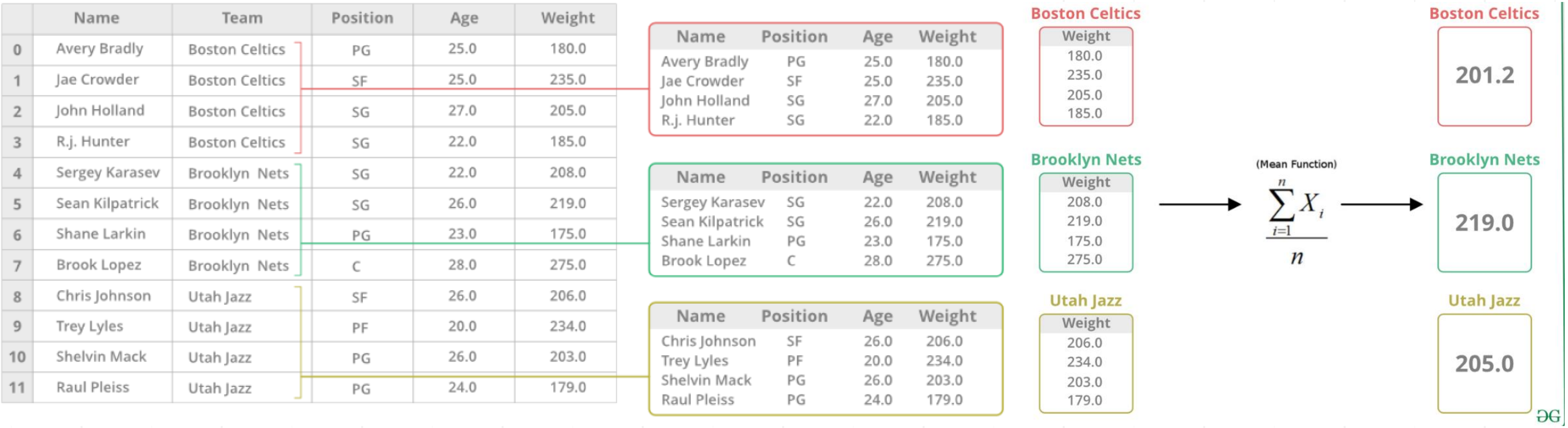
```
df_2012["Dist2Subway"] <= 1500
```

```
0         True
1        False
2         True
3         True
4         True
        ...
```

- `df[df["Dist2Subway"] <= 1500]`:
  - step1, `df["Dist2Subway"] <= 1500` return a series with values of ***False*** or ***True*** (boolean type);
  - step2, it is enclosed by `df.loc[]` and can return a subset of the candidate rows
  - step3, assign the returned DataFrame to a new dataframe called `df_subway`

df.groupby()

The df.**groupby()** function allows us to group data based on specific column(s) and then apply aggregate functions like mean, sum, count, etc., to analyze grouped subsets.



	Name	Team	Position	Age	Weight
0	Avery Bradly	Boston Celtics	PG	25.0	180.0
1	Jae Crowder	Boston Celtics	SF	25.0	235.0
2	John Holland	Boston Celtics	SG	27.0	205.0
3	R.j. Hunter	Boston Celtics	SG	22.0	185.0
4	Sergey Karasev	Brooklyn Nets	SG	22.0	208.0
5	Sean Kilpatrick	Brooklyn Nets	SG	26.0	219.0
6	Shane Larkin	Brooklyn Nets	PG	23.0	175.0
7	Brook Lopez	Brooklyn Nets	C	28.0	275.0
8	Chris Johnson	Utah Jazz	SF	26.0	206.0
9	Trey Lyles	Utah Jazz	PF	20.0	234.0
10	Shelvin Mack	Utah Jazz	PG	26.0	203.0
11	Raul Pleiss	Utah Jazz	PG	24.0	179.0

Name	Position	Age	Weight
Avery Bradly	PG	25.0	180.0
Jae Crowder	SF	25.0	235.0
John Holland	SG	27.0	205.0
R.j. Hunter	SG	22.0	185.0

Name	Position	Age	Weight
Sergey Karasev	SG	22.0	208.0
Sean Kilpatrick	SG	26.0	219.0
Shane Larkin	PG	23.0	175.0
Brook Lopez	C	28.0	275.0

Name	Position	Age	Weight
Chris Johnson	SF	26.0	206.0
Trey Lyles	PF	20.0	234.0
Shelvin Mack	PG	26.0	203.0
Raul Pleiss	PG	24.0	179.0

Boston Celtics

Weight
180.0
235.0
205.0
185.0

Brooklyn Nets

Weight
208.0
219.0
175.0
275.0

Utah Jazz

Weight
206.0
234.0
203.0
179.0

(Mean Function)

$$\frac{\sum_{i=1}^{n} X_i}{n}$$

Boston Celtics

201.2

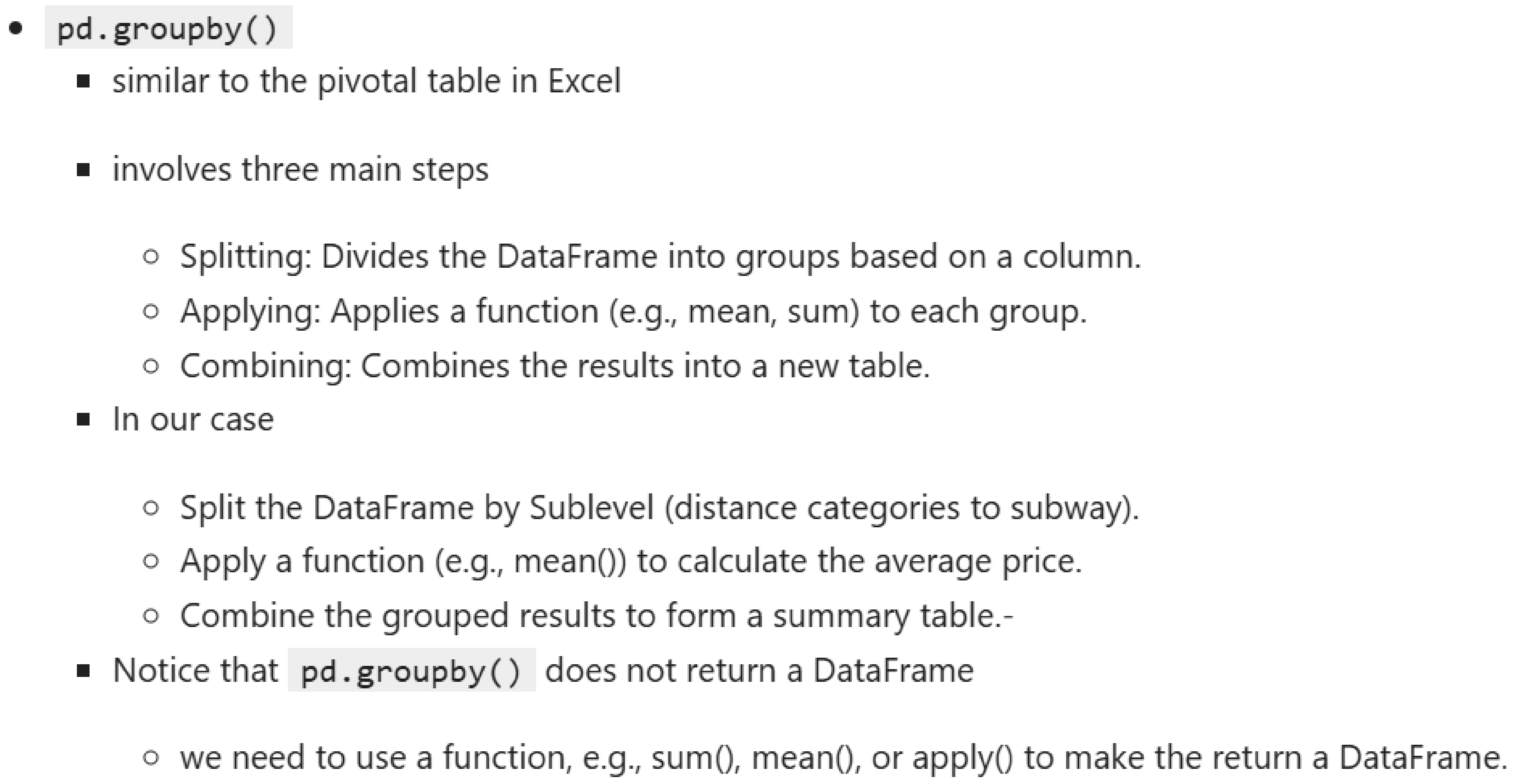
Brooklyn Nets

219.0

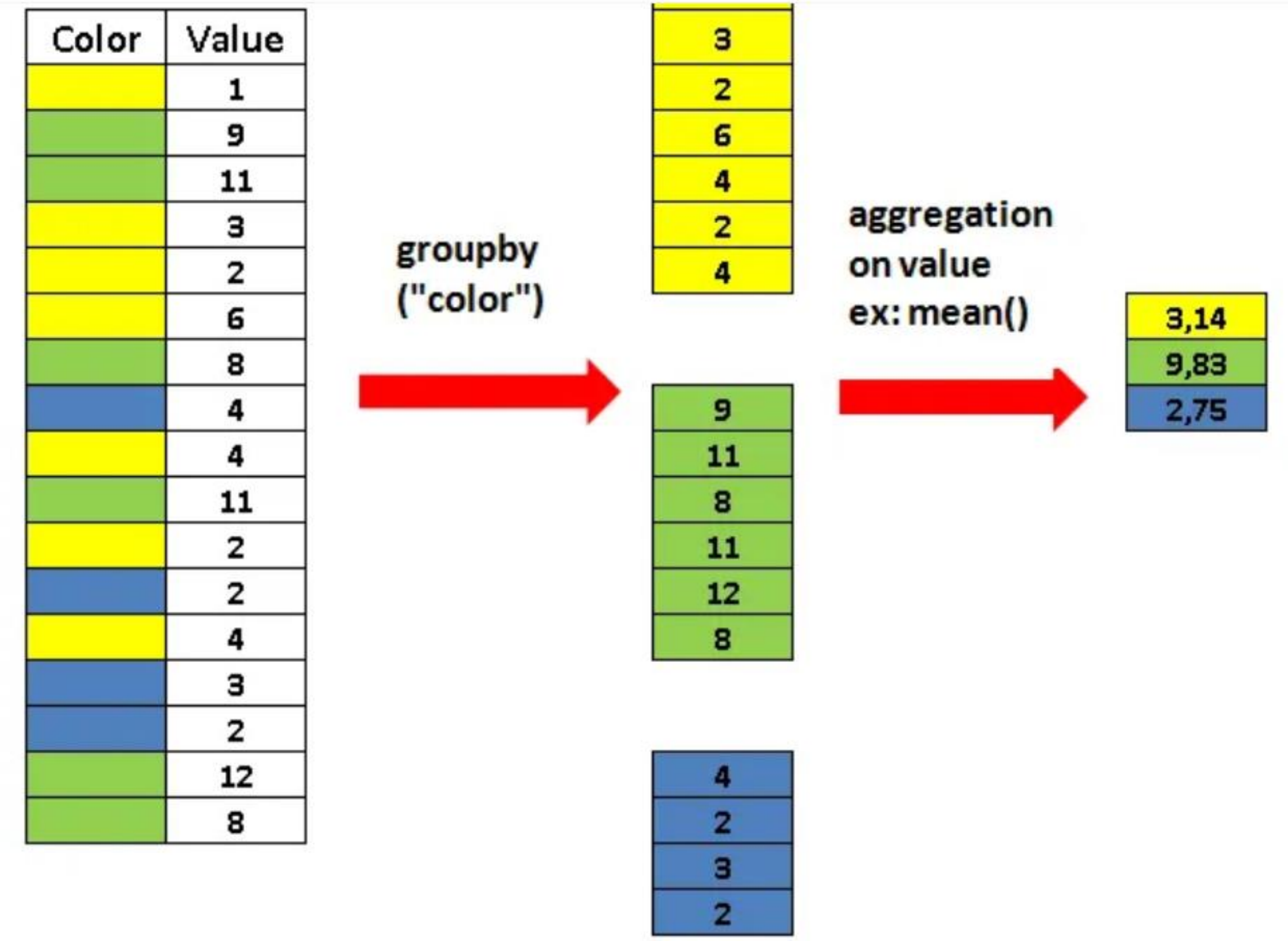
Utah Jazz

205.0

df.groupby()



- `pd.groupby()`
  - similar to the pivotal table in Excel
  - involves three main steps
    - Splitting: Divides the DataFrame into groups based on a column.
    - Applying: Applies a function (e.g., mean, sum) to each group.
    - Combining: Combines the results into a new table.
  - In our case
    - Split the DataFrame by Sublevel (distance categories to subway).
    - Apply a function (e.g., mean()) to calculate the average price.
    - Combine the grouped results to form a summary table.-
  - Notice that `pd.groupby()` does not return a DataFrame
    - we need to use a function, e.g., sum(), mean(), or apply() to make the return a DataFrame.



Color	Value
🟨	1
🟩	9
🟩	11
🟨	3
🟨	2
🟨	6
🟩	8
🟦	4
🟨	4
🟩	11
🟨	2
🟦	2
🟨	4
🟦	3
🟦	2
🟩	12
🟩	8

groupby ("color")

3
2
6
4
2
4

9
11
8
11
12
8

4
2
3
2

aggregation on value ex: mean()

3,14
9,83
2,75

Credit Image